ESP32-P4

Technical Reference Manual Pre-release v0.7

PRELIMINARY

ESPRESSIF

About This Document

The **ESP32-P4 Technical Reference Manual** is targeted at developers working on low level software projects that use the ESP32-P4 SoC. It describes the hardware modules listed below for the ESP32-P4 SoC and other products in ESP32-P4 series. The modules detailed in this document provide an overview, list of features, hardware architecture details, any necessary programming procedures, as well as register descriptions.

Navigation in This Document

Here are some tips on navigation through this extensive document:

- Release Status at a Glance on the very next page is a minimal list of all chapters from where you can directly jump to a specific chapter.
- Use the **Bookmarks** on the side bar to jump to any specific chapters or sections from anywhere in the document. Note this PDF document is configured to automatically display **Bookmarks** when open, which is necessary for an extensive document like this one. However, some PDF viewers or browsers ignore this setting, so if you don't see the **Bookmarks** by default, try one or more of the following methods:
  - Install a PDF Reader Extension for your browser;
  - Download this document, and view it with your local PDF viewer;
  - Set your PDF viewer to always automatically display the **Bookmarks** on the left side bar when open.
- Use the native **Navigation** function of your PDF viewer to navigate through the documents. Most PDF viewers support to go **Up**, **Down**, **Previous**, **Next**, **Back**, **Forward** and **Page** with buttons, menu, or hot keys.
- You can also use the built-in **GoBack** button on the upper right corner on each and every page to go back to the previous place before you click a link within the document. Note this feature may only work with some Acrobat-specific PDF viewers (for example, Acrobat Reader and Adobe DC) and browsers with built-in Acrobat-specific PDF viewers or extensions (for example, Firefox).

Release Status at a Glance

Note that this manual in still work in progress. See our release progress below:

No.	Chapter	Progress
	Part I. Microprocessor and Master	
1	Bus Architecture	Published
2	High-Performance CPU	Published
3	RISC-V Trace Encoder (TRACE)	Published
4	Processor Instruction Extensions (PIE)	Published
5	Low-Power CPU	Published
	Part II. System DMA	
6	GDMA Controller (GDMA-AHB, GDMA-AXI)	Published
7	VDMA Controller (VDMA)	Published
8	2D-DMA Controller (2D-DMA)	Published
	Part III. Memory Organization	
9	System and Memory	Published
10	eFuse Controller (EFUSE)	Published
	Part IV. System Component	
11	GPIO Matrix and IO MUX	Published
12	Reset and Clock	Published
13	Chip Boot Control	Published
14	Interrupt Matrix	Published
15	Event Task Matrix (ETM)	Published
16	Low-Power Management	Published
17	System Timer	Published
18	Timer Group (TIMG)	Published
19	Watchdog Timers (WDT)	Published
20	RTC Timer	Published
21	Permission Control (PMS)	Published
22	System Registers (SYSREG)	Published
23	Debug Assistant	Published
24	AXI Performance Monitor	Published
25	LP Mailbox	Published
26	Brown-out Detector	Published
27	Power Supply Detector (PSDET)	Published
	Part V. Cryptography/Security Component	
28	AES Accelerator (AES)	Published
29	ECC Accelerator (ECC)	Published
30	HMAC Accelerator (HMAC)	Published
31	RSA Accelerator (RSA)	Published
32	SHA Accelerator (SHA)	Published
33	RSA Digital Signature Peripheral (RSA_DS)	Published
34	ECDSA Digital Signature Peripheral (ECDSA_DS)	Published
35	External Memory Encryption and Decryption (XTS_AES)	Published

No.	Chapter	Progress
36	Random Number Generator (RNG)	Published
37	Key Manager	Published
	Part VI. Image and Voice Processing	
38	JPEG Codec	Published
39	Image Signal Processor (ISP)	Published
40	Pixel-Processing Accelerator (PPA)	Published
41	LCD and Camera Controller (LCD_CAM)	Published
42	H264 Encoder	Published
43	MIPI CSI	Published
44	MIPI DSI	Published
45	Voice Activity Detection (VAD)	Published
	Part VII. Connectivity Interface	
46	UART Controller (UART)	Published
47	SPI Controller (SPI)	Published
48	I2C Controller (I2C)	Published
49	Analog I2C Controller	Published
50	I3C Controller [to be added later]	86%
51	I2S Controller (I2S)	Published
52	LP I2S Controller	Published
53	Pulse Count Controller (PCNT)	Published
54	USB 2.0 High-Speed OTG	Published
55	USB 2.0 Full-Speed OTG	Published
56	USB Serial/JTAG Controller (USB_SERIAL_JTAG)	Published
57	Ethernet Media Access Controller (EMAC)	Published
58	Two-Wire Automotive Interface (TWAI)	Published
59	SD/MMC Host Controller (SDHOST)	Published
60	LED PWM Controller (LEDC)	Published
61	Motor Control PWM (MCPWM)	Published
62	Remote Control Peripheral (RMT)	Published
63	Parallel IO Controller (PARLIO)	Published
64	BitScrambler	Published
	Part VIII. Analog Signal Processing	
65	Touch Sensor (TOUCH)	Published
66	Temperature Sensor (TSENS)	Published
67	ADC Controller (ADC)	Published
68	Analog Voltage Comparator	Published

Note:

Check the link or the QR code to make sure that you use the latest version of this document:
https://www.espressif.com/documentation/esp32-p4_technical_reference_manual_en.pdf

Contents

I Microprocessor and Master 64

1 Bus Architecture 65

2 High-Performance CPU 68
2.1 Overview 68
2.2 Features 69
2.3 Terminology 70
2.4 Address Map 70
2.5 Configuration and Status Registers (CSRs) 71
2.5.1 Register Summary 71
2.5.2 Register Description 75
2.6 RISC-V Standard ISA Extensions Support 111
2.6.1 M Extension 111
2.6.1.1 Overview 111
2.6.1.2 Functional Description 111
2.6.2 C Extension 111
2.6.2.1 Overview 111
2.6.2.2 Functional Description 111
2.6.3 F Extension 112
2.6.3.1 Overview 112
2.6.3.2 Functional Description 112
2.6.3.3 Initialization 112
2.6.4 Zc (Z) Extension 113
2.6.4.1 Overview 113
2.6.4.2 Functional Description 113
2.6.4.3 Zc Instructions 114
2.6.4.4 Limitations 115
2.6.5 Bit Manipulation (Zb) Extension 116
2.6.5.1 Overview 116
2.6.5.2 Functional Description 116
2.6.6 A Extension 117
2.6.6.1 Overview 117
2.6.6.2 Functional Description 117
2.6.6.3 Load-Reserve (LR.W) Instruction 117
2.6.6.4 Store-Conditional (SC.W) Instruction 117
2.6.6.5 AMO Instructions 118
2.7 Custom Instruction Extensions 119
2.7.1 Hardware Loop 119
2.7.1.1 Overview 119
2.7.1.2 Features 119
2.7.1.3 Functional Description 119

2.7.1.4 Instructions/Operations/Modes Supported in HWLP 119
2.7.1.5 HWLP Constraints 120
2.7.1.6 Register Summary 120
2.7.1.7 Register Description 120
2.7.1.8 HWLP Instructions 126
2.7.2 Processor Instruction Extension 127
2.7.2.1 Overview 127
2.7.2.2 Functional Description 127
2.7.2.3 Initialization, Context Switching, and Exceptions 127
2.8 Memory-Mapped Registers 129
2.8.1 Overview 129
2.8.2 Features 129
2.8.3 Functional Description 129
2.9 Interrupt Controller 130
2.9.1 Overview 130
2.9.2 CLIC 130
2.9.2.1 Overview 130
2.9.2.2 CLIC CSRs 130
2.9.2.3 Interrupt Mapping 131
2.9.2.4 CLIC Parameters 131
2.9.2.5 Interrupt Level and Priority Encoding 131
2.9.2.6 CLIC Memory-Mapped Register Summary 132
2.9.2.7 CLIC Memory-Mapped Register Description 134
2.9.3 Core-Local Interrupts (CLINT) 137
2.9.3.1 Overview 137
2.9.3.2 Features 137
2.9.3.3 Software Interrupt 137
2.9.3.4 Timer Counter and Interrupt 137
2.9.3.5 Register Summary 137
2.9.3.6 Register Description 138
2.10 Memory Protection Unit 142
2.10.1 Standard Physical Memory Protection 142
2.10.1.1 Overview 142
2.10.1.2 Features 142
2.10.1.3 Functional Description 142
2.10.1.4 Register Summary 143
2.10.1.5 Register Description 143
2.10.2 Custom Physical Memory Attribute (PMA) Checker 147
2.10.2.1 Overview 147
2.10.2.2 Features 147
2.10.2.3 Functional Description 147
2.10.2.4 Register Summary 149
2.10.2.5 Register Description 150
2.11 Debug and Trace Support 152
2.11.1 Debug 152
2.11.1.1 Overview 152

2.11.1.2 Features 153
2.11.1.3 Functional Description 153
2.11.1.4 JTAG Control 153
2.11.1.5 Register Summary 154
2.11.1.6 Register Description 155
2.11.2 Debug Halt Groups 159
2.11.2.1 Overview 159
2.11.2.2 Features 159
2.11.2.3 Functional Description 159
2.11.2.4 Register Summary 160
2.11.2.5 Register Description 160
2.11.3 Hardware Trigger 161
2.11.3.1 Overview 161
2.11.3.2 Features 161
2.11.3.3 Functional Description 161
2.11.3.4 Trigger Execution Flow 162
2.11.3.5 Register Summary 162
2.11.3.6 Register Description 163
2.11.4 Trace 168
2.11.4.1 Overview 168
2.11.4.2 Features 168
2.11.4.3 Functional Description 168
2.12 Performance 169
2.12.1 Branch Prediction 169
2.12.1.1 Overview 169
2.12.1.2 Features 169
2.12.1.3 Functional Description 169
2.12.2 RAS 170
2.12.2.1 Overview 170
2.12.2.2 Features 170
2.12.2.3 Functional Description 171
2.12.3 Control Status Register for Performance Configuration 171
2.13 Custom Features 172
2.13.1 Bus Error Response 172
2.13.1.1 Overview 172
2.13.1.2 Functional Description 172
2.13.1.3 Control Status Registers 173
2.13.2 Dedicated IO 174
2.13.2.1 Overview 174
2.13.2.2 Features 174
2.13.2.3 Functional Description 174
2.13.2.4 Register Summary 175
2.13.2.5 Register Description 175
2.13.3 RunStall Support 177
2.13.3.1 Overview 177
2.13.3.2 Functional Description 177

2.13.3.3 Register Summary 177
2.13.4 Debug Assist Information 178
2.13.5 Core Lock-up 178
2.13.5.1 Overview 178
2.13.5.2 Functional Description 178
2.13.5.3 Register Summary 178
2.13.5.4 Register Description 178
2.13.6 External Wakeup 178
2.13.6.1 Overview 178
2.13.6.2 Functional Description 179

3 RISC-V Trace Encoder (TRACE) 180
3.1 Terminology 180
3.2 Introduction 181
3.3 Features 182
3.4 Architectural Overview 183
3.5 Functional Description 183
3.5.1 Synchronization 183
3.5.2 Address Mode 184
3.5.3 Optional Sideband Signals 184
3.5.4 Filtering 185
3.5.5 Anchor Tag 185
3.5.6 Memory Writing Mode 186
3.5.7 Automatic Restart 186
3.6 Encoder Output Packets 186
3.6.1 Header 186
3.6.2 Index 187
3.6.3 Payload 187
3.6.3.1 Format 3 Packets 187
3.6.3.2 Format 2 Packets 189
3.6.3.3 Format 1 Packets 190
3.7 Interrupt 191
3.8 Programming Procedures 192
3.8.1 Encoder Option Configuration 192
3.8.2 Filter Configuration 193
3.8.3 Enable Encoder 194
3.8.4 Disable Encoder 195
3.8.5 Notify 195
3.8.6 Decode Data Packets 196
3.8.7 AHB Configuration 196
3.8.8 Software Retention 197
3.9 Register Summary 198
3.10 Registers 199

4 Processor Instruction Extensions (PIE) 210
4.1 Overview 210

4.2 Features 210
4.3 Structure Overview 210
4.3.1 Internal Structure Diagram 210
4.3.2 Vector Register Bank 211
4.3.3 Arithmetic Logic Unit (ALU) 211
4.3.4 Multiply-and-Accumulate (MAC) Unit 212
4.3.4.1 QACC Accumulator Register 212
4.3.4.2 XACC Accumulator Register 212
4.3.5 Address Unit 212
4.4 Components of Extended Instruction Set 212
4.4.1 Registers 212
4.4.1.1 General-Purpose Registers 213
4.4.1.1.1 AR 213
4.4.1.2 Vector Registers 213
4.4.1.2.1 QR 213
4.4.1.3 Special Registers 213
4.4.1.3.1 SAR 214
4.4.1.3.2 SAR_BYTE 214
4.4.1.3.3 FFT_BIT_WIDTH 214
4.4.1.3.4 UA_STATE 214
4.4.1.3.5 Core Configuration and Status Register 215
4.4.2 Data Format and Alignment 215
4.4.3 Misaligned Access Handling 216
4.4.3.1 Force-alignment mode 216
4.4.3.2 Hardware misaligned mode 216
4.4.4 Data Overflow and Saturation 216
4.4.5 Data Rounding 217
4.5 Extended Instruction List 217
4.5.1 Read Instructions 220
4.5.2 Write Instructions 221
4.5.3 Data Exchange Instructions 222
4.5.4 Arithmetic Instructions 223
4.5.4.1 Vector Addition Instructions 223
4.5.4.2 Vector Multiplication Instructions 224
4.5.4.3 Vector Complex Multiplication Instructions 224
4.5.4.4 Vector Multiplication Accumulation Instructions 225
4.5.4.5 Vector and Scalar Multiplication Accumulation Instructions 226
4.5.4.6 Other Instructions 226
4.5.5 Comparison Instructions 227
4.5.6 Bitwise Logical Instructions 228
4.5.7 Shift Instructions 228
4.5.8 Fast Fourier Transform (FFT) Instructions 229
4.5.8.1 Butterfly Computation Instructions 229
4.5.8.2 Bit Reverse Instruction 229
4.5.8.3 Real FFT Instructions 229
4.6 Instruction Performance 230

4.6.1 Data Hazard 230
4.6.2 Hardware Resource Hazard 231
4.6.3 Control Hazard 232
4.7 Extended Instruction Functional Description 233
4.7.1 Instruction Symbols and Field Definitions 233
4.7.2 List of Instructions 234

5 Low-Power CPU 633
5.1 Overview 633
5.2 Features 633
5.3 Configuration and Status Registers (CSRs) 634
5.3.1 Register Summary 634
5.3.2 Registers 635
5.4 Interrupts and Exceptions 644
5.4.1 Interrupts 644
5.4.2 Interrupt Handling 645
5.4.3 Exceptions 645
5.5 Debugging 646
5.5.1 Features 646
5.5.2 Functional Description 646
5.5.3 Register Summary 647
5.5.4 Registers 647
5.6 Hardware Trigger 649
5.6.1 Features 649
5.6.2 Functional Description 649
5.6.3 Trigger Execution Flow 650
5.6.4 Register Summary 650
5.6.5 Registers 650
5.7 Performance Counter 653
5.8 System Access 654
5.8.1 Memory Access 654
5.8.2 Peripheral Access 654
5.9 Event Task Matrix Feature 655
5.10 Sleep and Wake-Up Process 655
5.10.1 Features 655
5.10.2 Process 655
5.10.3 Wake-Up Sources 657
5.10.4 Sleep Rejection 657

II System DMA 658

6 GDMA Controller (GDMA-AHB, GDMA-AXI) 659
6.1 Overview 659
6.2 Features 660
6.3 Architecture 661
6.4 Functional Description 662

6.4.1 Linked List 662
6.4.2 Peripheral-to-Memory and Memory-to-Peripheral Data Transfer 663
6.4.3 Memory-to-Memory Data Transfer 664
6.4.4 Enabling GDMA 664
6.4.5 Linked List Reading Process 665
6.4.6 Interrupt Response Mechanism for GDMA-AXI Linked List Switching 666
6.4.7 EOF 666
6.4.8 Accessing Memory 667
6.4.9 Arbitration 667
6.4.10 Bus Error Response Information Logging 668
6.4.11 Automatic Clock Gating 668
6.4.12 CRC Calculation 669
6.5 Event Task Matrix Feature 670
6.6 Interrupts 671
6.7 Programming Procedures 673
6.7.1 Programming Procedures for GDMA's Transmit Channel 673
6.7.2 Programming Procedures for GDMA's Receive Channel 673
6.7.3 Programming Procedures for Memory-to-Memory Transfer 673
6.7.4 Programming Procedures for Channel Priority and Weight 674
6.7.5 Programming Procedures for CRC Calculation 674
6.8 Register Summary 675
6.8.1 GDMA-AHB Register Summary 675
6.8.2 GDMA-AXI Register Summary 683
6.9 Registers 691
6.9.1 GDMA-AHB Registers 691
6.9.2 GDMA-AXI Registers 730

7 VDMA Controller (VDMA) 774
7.1 Overview 774
7.2 Terminology 774
7.3 Features 775
7.4 Architectural Overview 775
7.5 Functional Description 777
7.5.1 Transfer Hierarchy 777
7.5.2 Arbitration Scheme 779
7.5.2.1 Read Arbiter 779
7.5.2.2 Write Arbiter 780
7.5.3 Handshaking Interface 780
7.5.4 Transfer Control 781
7.5.4.1 Single-Block Transfer 781
7.5.4.2 Multi-Block Transfer 781
7.5.5 Flow Controller 786
7.5.6 Channel Suspend and Resume 786
7.5.7 Channel Disable 787
7.5.7.1 Disabling a Suspended Channel Before Transfer Completion 787
7.5.7.2 Disabling a Non-suspended Channel Before Transfer Completion 787

7.5.8 Low-Power Technique 788
7.5.8.1 Low-Power Technique for DMA Channels 788
7.5.8.2 Low-Power Technique for Slave Bus Interface 788
7.5.8.3 Low-Power Technique for AXI Master Interface Channels 789
7.5.8.4 Global Low-Power Technique 789
7.6 Interrupts 789
7.7 Programming Procedures 790
7.7.1 Common Programming Procedures 790
7.7.2 Programming Procedures for Shadow-Register-Based Multi-Block Transfer 790
7.7.3 Programming Procedures for Linked-List-Based Multi-Block Transfer 792
7.7.4 Programming Procedures for Single-Block Transfer 793
7.8 Register Summary 794
7.9 Registers 798

8 2D-DMA Controller (2D-DMA) 828
8.1 Overview 828
8.2 Features 828
8.3 Architecture 828
8.4 Functional Description 829
8.4.1 Transfer Mode 829
8.4.2 Linked List 830
8.4.3 Padding in 2D-MOD1 Mode and DSCR-PORT Mode 833
8.4.4 Peripheral-to-Memory and Memory-to-Peripheral Data Transfer 834
8.4.5 Memory-to-Memory Data Transfer 835
8.4.6 Macroblock Reordering 835
8.4.6.1 Macroblock Reordering in TX Direction 835
8.4.6.2 Macroblock Reordering in RX Direction 836
8.4.7 Color Space Conversion 837
8.4.8 Enabling 2D-DMA 840
8.4.9 Linked List Reading Process 841
8.4.10 EOF 841
8.4.11 Accessing Memory 842
8.4.12 Arbitration 842
8.5 Event Task Matrix Feature 843
8.6 Interrupts 844
8.7 Programming Procedures 846
8.7.1 General Configurations for 2D-DMA 846
8.7.2 Mode-Specific Configurations for 2D-DMA 846
8.7.3 Configurations for 2D-DMA's Transmit Channel 846
8.7.4 Configurations for 2D-DMA's Receive Channel 847
8.7.5 Configurations for Memory-to-Memory Transfer 847
8.7.6 Configurations for Channel Priority and Weight 847
8.7.7 Resetting 2D-DMA While it Runs 848
8.8 Register Summary 849
8.9 Registers 856

III Memory Organization 900

9 System and Memory 901
9.1 Overview 901
9.2 Features 901
9.3 Functional Description 902
9.3.1 Address Mapping 902
9.3.2 Internal Memory 905
9.3.3 External Memory 909
9.3.3.1 External Memory Address Mapping 909
9.3.3.2 Cache 909
9.3.3.3 Cache Operations 910
9.3.4 DMA Address Space 911
9.3.5 Modules/Peripherals Address Mapping 912
9.4 eFuse Controller (EFUSE) 916
9.4.1 Overview 916
9.4.2 Features 916
9.4.3 Functional Description 916
9.4.3.1 Structure 916
9.4.3.1.1 EFUSE_WR_DIS 926
9.4.3.1.2 EFUSE_RD_DIS 926
9.4.3.1.3 Data Storage 926
9.4.3.2 Programming of Parameters 928
9.4.3.3 Reading of Parameters by Users 929
9.4.3.4 eFuse VDDQ Timing 931
9.4.3.5 Parameters Used by Hardware Modules 931
9.4.4 Interrupts 931
9.4.5 Register Summary 933
9.4.6 Registers 940

IV System Component 980

10 GPIO Matrix and IO MUX 981
10.1 Overview 981
10.2 Features 981
10.2.1 HP GPIO Matrix and HP IO MUX 981
10.2.2 LP GPIO Matrix and LP IO MUX 982
10.3 Architectural Overview 982
10.4 Peripheral Input via GPIO Matrix 985
10.4.1 Overview 985
10.4.2 Signal Synchronization 985
10.4.3 GPIO Filter 986
10.4.4 Glitch Filter 986
10.4.5 Simple GPIO Input 987
10.4.6 GPIO Wakeup 987
10.4.6.1 HP GPIO Wakeup 987

10.4.6.2 LP GPIO Wakeup 987
10.4.7 Programming Procedure 988
10.4.7.1 HP GPIO Matrix 988
10.4.7.2 LP GPIO Matrix 989
10.5 Peripheral Output via GPIO Matrix 990
10.5.1 Overview 990
10.5.2 Simple GPIO Output 990
10.5.3 Sigma Delta Modulated Output (SDM) 991
10.5.3.1 Functional Description 991
10.5.3.2 SDM Configuration 991
10.5.4 Programming Procedure 992
10.5.4.1 HP GPIO Matrix 992
10.5.4.2 LP GPIO Matrix 993
10.6 Direct Input and Output via IO MUX 993
10.6.1 Overview 993
10.6.2 Functional Description 993
10.6.2.1 HP IO MUX 993
10.6.2.2 LP IO MUX 994
10.7 Analog Functions 994
10.7.1 Overview 994
10.7.2 Analog Functions 994
10.8 Pin Functions in Light-sleep 995
10.9 Pin Hold Feature 995
10.10 Hysteresis Characteristics of GPIO Pins 997
10.11 Power Supplies and Management of GPIO Pins 998
10.11.1 Power Supplies of GPIO Pins 998
10.11.2 Power Supply Management 998
10.12 HP Peripheral Signal List 998
10.13 LP Peripheral Signal List 1009
10.14 HP IO MUX Functions List 1011
10.15 LP IO MUX Functions List 1014
10.16 GPIO Pin Analog Functions List 1015
10.17 Event Task Matrix Function 1016
10.18 Interrupts 1017
10.19 Register Summary 1018
10.19.1 HP GPIO Matrix Register Summary 1018
10.19.2 HP IO MUX Register Summary 1020
10.19.3 GPIO EXT Register Summary 1022
10.19.4 LP GPIO Matrix Register Summary 1023
10.19.5 LP IO MUX Register Summary 1025
10.20 Registers 1025
10.20.1 HP GPIO Matrix Registers 1026
10.20.2 HP IO MUX Registers 1048
10.20.3 GPIO EXT Registers 1051
10.20.4 LP GPIO Matrix Registers 1067
10.20.5 LP IO MUX Registers 1076

11 Reset and Clock 1081
11.1 Reset 1081
11.1.1 Overview 1081
11.1.2 Architectural Overview 1081
11.1.3 Features 1081
11.1.4 Functional Description 1082
11.1.5 Peripheral Reset 1084
11.2 Clock 1084
11.2.1 Overview 1084
11.2.2 Architectural Overview 1085
11.2.3 Features 1085
11.2.4 Functional Description 1086
11.2.4.1 HP System Clock 1086
11.2.4.2 LP System Clock 1087
11.2.4.3 Peripheral Clocks 1088
11.3 Programming Procedures 1092
11.3.1 HP System Clock Configuration 1092
11.3.2 LP System Clock Configuration 1093
11.3.3 Peripheral Clock Reset and Configuration 1093
11.4 Register Summary 1095
11.4.1 Reset and Clock (HP_SYS_CLKRST) Register Summary 1095
11.4.2 LP Always on Clock and Reset (LP_CLKRST) Register Summary 1097
11.4.3 LP Peripheral Clock and Reset (LPPERI) Register Summary 1097
11.5 Registers 1099
11.5.1 Reset and Clock (HP_SYS_CLKRST) Registers 1099
11.5.2 LP Always on Clock and Reset (LP_CLKRST) Registers 1189
11.5.3 LP Peripheral Clock and Reset (LPPERI) Registers 1209

12 Chip Boot Control 1219
12.1 Overview 1219
12.2 Functional Description 1219
12.2.1 Default Configuration 1219
12.2.2 Boot Mode Control 1220
12.2.3 ROM Messages Printing Control 1222
12.2.4 JTAG Signal Source Control 1223

13 Interrupt Matrix 1224
13.1 Overview 1224
13.2 Interrupt Terminology in ESP32-P4 1224
13.2.1 Interrupt 1224
13.2.2 Interrupt Signal/Interrupt Source 1224
13.2.3 Interrupt Flow in ESP32-P4 1225
13.3 Features 1225
13.4 Functional Description 1226
13.4.1 Peripheral Interrupt Sources 1226
13.4.2 Assign Peripheral Interrupt Source to HP CPU Interrupt 1233

13.4.2.1 Assign One Peripheral Interrupt Source (*SOURCE*) to HP CPU*x* 1233
13.4.2.2 Assign Multiple Peripheral Interrupt Sources (*SOURCE*) to HP CPU*x* 1233
13.4.2.3 Disable HP CPU*x* Peripheral Interrupt Source (*SOURCE*) 1233
13.4.3 Query Current Interrupt Status of HP CPU*x* Peripheral Interrupt Source 1233
13.4.4 Interrupt Remapping 1233
13.5 Register Summary 1235
13.5.1 HP CPU0 Interrupt Matrix Register Summary 1235
13.5.2 HP CPU1 Interrupt Matrix Register Summary 1240
13.6 Registers 1247
13.6.1 HP CPU0 Interrupt Matrix Registers 1247
13.6.2 HP CPU1 Interrupt Matrix Registers 1257

14 Event Task Matrix (ETM) 1268
14.1 Overview 1268
14.2 Features 1268
14.3 Functional Description 1268
14.3.1 Architecture 1268
14.3.2 Events 1269
14.3.3 Tasks 1274
14.3.4 Event and Task Status 1279
14.3.5 Timing Considerations 1279
14.3.6 Channel Control 1281
14.4 Register Summary 1282
14.5 Registers 1286

15 Low-Power Management 1425
15.1 Overview 1425
15.2 Terminology 1425
15.3 Features 1425
15.4 Functional Description 1426
15.4.1 Power Scheme 1426
15.4.1.1 Regulators 1427
15.4.1.2 Digital Power Domains 1428
15.4.1.3 Analog Power Domains 1428
15.4.1.4 Battery Power Domain 1428
15.4.2 PMU 1429
15.4.2.1 PMU Main State Machine 1430
15.4.2.2 Sleep/Wake-up Controller 1431
15.4.2.3 Analog Power Controller 1432
15.4.2.4 Digital Power Controller 1433
15.4.2.5 Clock Controller 1435
15.4.2.6 Backup Controller 1436
15.4.2.7 System Controller 1437
15.4.2.8 Battery Power Controller 1437
15.4.2.9 Output Regulator Control 1438
15.4.2.9.1 Regulator Features 1438

15.4.2.9.2 Regulator Configuration Examples 1440
15.4.2.9.3 Software Configuration Steps 1440
15.5 Power Modes 1441
15.6 Event Task Matrix Feature 1441
15.7 Interrupts 1442
15.8 Register Summary 1444
15.9 Registers 1447

16 System Timer 1507
16.1 Overview 1507
16.2 Features 1507
16.3 System Timer Structure 1508
16.4 Clock Source Selection 1508
16.5 Functional Description 1509
16.5.1 Counter 1509
16.5.2 Comparator and Alarm 1510
16.5.3 Event Task Matrix 1511
16.5.4 Synchronization Operation 1511
16.6 Interrupts 1512
16.7 Programming Procedure 1512
16.7.1 Read Current Count Value 1512
16.7.2 Configure a One-Time Alarm in Target Mode 1513
16.7.3 Configure Periodic Alarms in Period Mode 1513
16.7.4 Update After Light-sleep 1513
16.8 Register Summary 1515
16.9 Registers 1517

17 Timer Group (TIMG) 1532
17.1 Overview 1532
17.2 Features 1532
17.3 Functional Description 1533
17.3.1 16-bit Prescaler and Clock Selection 1533
17.3.2 54-bit Time-base Counter 1534
17.3.3 Alarm Generation 1534
17.3.4 Timer Reload 1535
17.3.5 Event Task Matrix Feature 1536
17.3.6 RTC_SLOW_CLK Frequency Calculation 1537
17.3.7 Interrupts 1537
17.4 Configuration and Usage 1538
17.4.1 Timer as a Simple Clock 1538
17.4.2 Timer as One-shot Alarm 1538
17.4.3 Timer as Periodic Alarm by APB 1538
17.4.4 Timer as Periodic Alarm by ETM 1539
17.4.5 RTC_SLOW_CLK Frequency Calculation 1540
17.5 Register Summary 1541
17.6 Registers 1543

18 Watchdog Timers (WDT) 1557
18.1 Overview 1557
18.2 Digital Watchdog Timers 1558
18.2.1 Features 1558
18.2.2 Functional Description 1559
18.2.2.1 Clock Source and 32-Bit Counter 1559
18.2.2.2 Stages and Timeout Actions 1560
18.2.2.3 Write Protection 1560
18.2.2.4 Flash Boot Protection 1561
18.3 Super Watchdog 1561
18.3.1 Features 1561
18.3.2 Super Watchdog Controller 1561
18.3.2.1 Structure 1562
18.3.2.2 Workflow 1562
18.4 Interrupts 1562
18.5 Register Summary 1563
18.6 Registers 1563

19 RTC Timer 1572
19.1 Introduction 1572
19.2 Feature List 1572
19.3 Functional Description 1572
19.4 Event Task Matrix Feature 1574
19.5 Interrupts 1574
19.6 Register Summary 1576
19.7 Registers 1577

20 Permission Control (PMS) 1586
20.1 Overview 1586
20.2 Features 1588
20.3 Functional Description 1588
20.3.1 Architecture 1588
20.3.2 Address Ranges and Permissions Management 1589
20.3.2.1 Address Ranges Managed by HP APM and LP APM 1589
20.3.2.2 Address Ranges Managed by DMA APM 1590
20.4 Programming Procedure 1590
20.4.1 Configuring Access Permissions for HP CPU0/1 1591
20.4.2 Configuring Access Permissions for LP CPU 1591
20.4.3 Configuring Access Permission for DMA Masters 1592
20.5 Illegal Access and Interrupts 1592
20.6 Register Summary 1593
20.6.1 HP_DMA_PMS_REG 1593
20.6.2 HP_PERI_PMS_REG 1599
20.6.3 LP2HP_PERI_PMS_REG 1600
20.6.4 LP_PERI_PMS_REG 1600
20.6.5 HP2LP_PERI_PMS_REG 1602

20.7 Registers 1603
20.7.1 HP_DMA_PMS_REG 1603
20.7.2 HP_PERI_PMS_REG 1630
20.7.3 LP2HP_PERI_PMS_REG 1655
20.7.4 LP_PERI_PMS_REG 1668
20.7.5 HP2LP_PERI_PMS_REG 1677

21 System Registers (SYSREG) 1687
21.1 Overview 1687
21.2 Function Description 1687
21.2.1 HP System Registers 1687
21.2.1.1 External Memory Encryption/Decryption Configuration 1687
21.2.1.2 HP Peripherals Clock Configuration and Power Control 1687
21.2.1.3 HP Cache Clock and Reset Configuration 1687
21.2.1.4 HP SPM and HP L2MEM Clock Configuration 1688
21.2.1.5 HP SPM Parity Check Configuration 1688
21.2.1.6 HP L2MEM ECC Check Configuration 1689
21.2.1.7 BitScrambler Configuration 1689
21.2.1.8 Ethernet MAC Control 1689
21.2.1.9 USB 2.0 High-Speed OTG Control 1690
21.2.1.10 CPU Control and Record 1690
21.2.1.11 HP to LP Interrupt and Wakeup Control 1690
21.2.1.12 HP GPIO Hold Control 1690
21.2.1.13 HP GPIO Output Control 1690
21.2.1.14 Illegal Access and Unauthorized Access 1691
21.2.1.15 HP CPU Bus Timeout Protection 1691
21.2.1.16 AXI Matrix 1692
21.2.1.17 Post Write 1693
21.2.2 LP System Registers 1694
21.2.2.1 LP Timer Stall Control 1694
21.2.2.2 Focused Ion Beam (FIB) Control 1694
21.2.2.3 Boot Mode 1694
21.2.2.4 LP CPU Control 1694
21.2.2.5 Reset Control 1695
21.2.2.6 Wakeup Control 1696
21.2.2.7 Pad Hold Control 1696
21.2.2.8 Power Discharge Control 1696
21.2.2.9 LPROM and LP SPM Clock Control 1697
21.2.2.10 Analog Voltage Comparator 1697
21.2.2.11 Illegal Access and Unauthorized Access 1697
21.2.2.12 LP Bus Timeout Protection 1697
21.2.2.13 RNG Control 1698
21.3 Interrupt 1698
21.4 Register Summary 1700
21.4.1 HP Register Summary 1700
21.4.2 ICM Register Summary 1702

21.4.3 LP Register Summary 1703
21.5 Registers 1705
21.5.1 HP Registers 1705
21.5.2 ICM Register 1758
21.5.3 LP Registers 1768

22 Debug Assistant 1793
22.1 Overview 1793
22.2 Features 1793
22.3 Functional Description 1793
22.3.1 Region Read/Write Monitoring 1793
22.3.2 SP Monitoring 1794
22.3.3 PC Logging 1794
22.3.4 CPU/DMA Bus Access Logging 1794
22.4 Interrupts 1794
22.5 Recommended Operation 1795
22.5.1 Region Monitoring and SP Monitoring Configuration 1795
22.5.2 PC Logging Configuration 1798
22.5.3 CPU/DMA Bus Access Logging Configuration 1798
22.5.3.1 HP CPU0/1 Bus Access Logging Configuration 1798
22.5.3.2 DMA Bus Access Logging Configuration 1801
22.6 Register Summary 1805
22.6.1 HP CPU Bus Logging Configuration Register Summary 1805
22.6.2 DMA Bus Logging Configuration Register Summary 1805
22.6.3 Summary of Other Registers 1806
22.7 Registers 1810
22.7.1 HP CPU Bus Logging Configuration Registers 1810
22.7.2 DMA Bus Logging Configuration Registers 1817
22.7.3 Other Registers 1824

23 AXI Performance Monitor 1853
23.1 Overview 1853
23.2 Terminology 1853
23.3 Feature List 1854
23.4 Architectural Overview 1855
23.5 Functional Description 1855
23.5.1 Clock 1855
23.5.2 Reset 1856
23.5.3 Transaction ID Filtering 1856
23.5.4 Monitoring Metrics 1856
23.5.4.1 Maximum Outstanding Count for Read/Write Address Channel Transactions 1858
23.5.4.2 Read/Write Address Channel Latency 1858
23.5.4.3 Read/Write Data Channel Latency 1859
23.5.4.4 Write Data Channel WVALID Idle Cycles 1859
23.5.4.5 Read/Write Transaction Transfer Latency 1859
23.5.4.6 Average Bandwidth Measurement 1860

23.5.4.7 Instantaneous Bandwidth Measurement 1860
23.6 Interrupts 1862
23.7 Programming Procedures 1862
23.7.1 Using AXI Perf Mon to Measure Outstanding Transactions 1862
23.7.2 Using AXI Perf Mon to Measure Read Average Bandwidth 1863
23.7.3 Using AXI Perf Mon to Measure Read Instantaneous Bandwidth 1863
23.8 Register Summary 1865
23.9 Registers 1872

24 LP Mailbox 1894
24.1 Overview 1894
24.2 Features 1894
24.3 Functional Description 1895
24.3.1 Message Registers 1895
24.3.2 Interrupts 1895
24.3.3 Inter-Core Communication 1896
24.4 Register Summary 1897
24.5 Registers 1897

25 Brown-out Detector 1911
25.1 Introduction 1911
25.2 Feature List 1911
25.3 Architectural Overview 1912
25.4 Functional Description 1912
25.5 Interrupts 1915
25.6 Programming Procedures 1915
25.7 Register Summary 1917
25.8 Registers 1918

26 Power Supply Detector (PSDET) 1926
26.1 Overview 1926
26.2 Features 1926
26.3 Functional Description 1926

V Cryptography/Security Component 1927

27 AES Accelerator (AES) 1928
27.1 Introduction 1928
27.2 Features 1928
27.3 Clock and Reset 1928
27.4 AES Working Modes 1929
27.5 Typical AES Working Mode 1930
27.5.1 Key, Plaintext, and Ciphertext 1930
27.5.2 Endianness 1930
27.5.3 Operation Process 1933
27.6 DMA-AES Working Mode 1934

27.6.1 Key, Plaintext, and Ciphertext 1934
27.6.2 Endianness 1935
27.6.3 Standard Incrementing Function 1936
27.6.4 Block Number 1936
27.6.5 Initialization Vector 1936
27.6.6 Block Operation Process 1936
27.6.7 GCM Operation Process 1937
27.7 GCM Algorithm 1938
27.7.1 Hash Subkey 1939
27.7.2 J_0 1939
27.7.3 Authentication Tag 1939
27.7.4 AAD Block Number 1940
27.7.5 Number of Effective Bits of Incomplete Blocks 1940
27.8 Pseudo-Round Anti-DPA Function 1940
27.9 Interrupts 1941
27.10 Memory Summary 1941
27.11 Register Summary 1942
27.12 Registers 1943

28 ECC Accelerator (ECC) 1950
28.1 Overview 1950
28.2 Feature List 1950
28.3 ECC Basics 1950
28.3.1 Elliptic Curve and Points on the Curves 1950
28.3.2 Affine Coordinates and Jacobian Coordinates 1951
28.3.3 Memory Blocks 1951
28.3.4 Data and Data Block 1951
28.3.5 Writing Data 1952
28.3.6 Reading Data 1952
28.3.7 Standard Calculation and Jacobian Calculation 1952
28.4 Function Description 1952
28.4.1 Curve Mode 1952
28.4.2 Working Modes 1953
28.4.2.1 Affine Point Multiplication (Affine Point Multi) 1953
28.4.2.2 Affine Point Verification (Affine Point Verif) 1954
28.4.2.3 Affine Point Verification + Affine Point Multiplication (Affine Point Verif + Multi) 1954
28.4.2.4 Jacobian Point Multiplication (Jacobian Point Multi) 1954
28.4.2.5 Point Addition (Point Add) 1955
28.4.2.6 Jacobian Point Verification (Jacobian Point Verif) 1955
28.4.2.7 Affine Point Verification + Jacobian Point Multiplication (Affine Point Verif + Jacobian Point Multi) 1955
28.4.2.8 Mod Addition (Mod Add) 1956
28.4.2.9 Mod Subtraction (Mod Sub) 1956
28.4.2.10 Mod Multiplication (Mod Multi) 1956
28.4.2.11 Mod Division (Mod Div) 1957
28.4.3 Enhancing Anti-Attack Performance 1957

28.5 Clock and Reset 1957
28.6 Interrupts 1957
28.7 Programming Procedures 1958
28.8 Register Summary 1959
28.9 Registers 1960

29 HMAC Accelerator (HMAC) 1964
29.1 Main Features 1964
29.2 Functional Description 1964
29.2.1 Upstream Mode 1964
29.2.2 Downstream Mode - JTAG Enable Feature 1965
29.2.3 Downstream Mode - RSA_DS Key Derivation Feature 1965
29.2.4 HMAC eFuse Configuration 1966
29.2.5 HMAC Process (Detailed) 1967
29.2.5.1 Enable HMAC Module 1967
29.2.5.2 Configure HMAC Keys and Key Purposes 1967
29.2.5.3 Downstream Mode Process 1967
29.2.5.4 Upstream Mode Process 1968
29.3 HMAC Algorithm Details 1969
29.3.1 Padding Bits 1969
29.3.2 HMAC Algorithm Structure 1970
29.4 Register Summary 1972
29.5 Registers 1974

30 RSA Accelerator (RSA) 1981
30.1 Introduction 1981
30.2 Features 1981
30.3 Functional Description 1981
30.3.1 Definitions and Representations 1982
30.3.2 Large-Number Modular Exponentiation 1982
30.3.3 Large-Number Modular Multiplication 1983
30.3.4 Large-Number Multiplication 1984
30.3.5 Options for Additional Acceleration 1984
30.4 Interrupts 1986
30.5 Memory Summary 1987
30.6 Register Summary 1987
30.7 Registers 1988

31 SHA Accelerator (SHA) 1992
31.1 Introduction 1992
31.2 Features 1992
31.3 Working Modes 1992
31.4 Function Description 1993
31.4.1 Preprocessing 1993
31.4.1.1 Padding the Message 1994
31.4.1.2 Parsing the Message 1994

31.4.1.3 Setting the Initial Hash Value 1995
31.4.2 Hash Operation 1996
31.4.2.1 Typical SHA Mode Process 1996
31.4.2.2 DMA-SHA Mode Process 1998
31.4.3 Message Digest 1999
31.5 Interrupt 2000
31.6 Register Summary 2002
31.7 Registers 2004

32 RSA Digital Signature Peripheral (RSA_DS) 2008
32.1 Overview 2008
32.2 Features 2008
32.3 Functional Description 2008
32.3.1 Overview 2008
32.3.2 Private Key Operands 2009
32.3.3 Software Prerequisites 2009
32.3.4 RSA_DS Operation at the Hardware Level 2011
32.3.5 RSA_DS Operation at the Software Level 2011
32.4 Memory Summary 2014
32.5 Register Summary 2015
32.6 Registers 2016

33 ECDSA Digital Signature Peripheral (ECDSA_DS) 2019
33.1 Introduction 2019
33.2 Features 2019
33.3 ECDSA Basics 2019
33.3.1 Domain Parameters 2019
33.3.2 Key Generation 2020
33.3.3 Signature Generation 2020
33.3.4 Signature Verification 2021
33.4 Functional Description 2021
33.4.1 ECDSA_DS Working Modes 2021
33.4.2 Data and Data Block 2022
33.4.2.1 Writing Data 2023
33.4.2.2 Reading Data 2023
33.4.2.3 Padding the Message 2023
33.4.2.4 Parsing the Message 2024
33.4.3 Security Features 2024
33.4.3.1 High Anti-Attack Performance 2024
33.4.3.2 State-Dependent Register Access Control 2024
33.4.3.3 Hardware Occupation 2024
33.5 Programming Procedures 2025
33.5.1 ECDSA_DS Process 2025
33.5.1.1 IDLE Stage 2026
33.5.1.2 PREP Stage 2026
33.5.1.3 LOAD Stage 2026

33.5.1.4 PROC Stage 2026
33.5.1.5 POST Stage 2026
33.5.1.6 ECDSA_DS SHA Interface 2027
33.5.2 Clocks and Resets 2027
33.5.3 Interrupts 2028
33.6 Memory Blocks 2029
33.7 Register Summary 2030
33.8 Registers 2031

34 External Memory Encryption and Decryption (XTS_AES) 2038
34.1 Overview 2038
34.2 Features 2038
34.3 Module Structure 2038
34.4 Functional Description 2039
34.4.1 XTS Algorithm 2040
34.4.2 Key 2040
34.4.3 Target Memory Space 2041
34.4.4 Data Writing 2041
34.4.5 Manual Encryption Block 2042
34.4.6 Auto Encryption Block 2042
34.4.7 Auto Decryption Block 2043
34.5 Software Process 2043
34.6 Anti-DPA 2044
34.7 Register Summary 2046
34.8 Registers 2047

35 Random Number Generator (RNG) 2052
35.1 Introduction 2052
35.2 Feature List 2052
35.3 Functional Description 2052
35.4 Programming Procedure 2053
35.5 Register Summary 2054
35.6 Registers 2055

36 Key Manager 2056
36.1 Overview 2056
36.2 Background Knowledge 2056
36.3 Glossary 2057
36.4 Features 2058
36.5 Architectural Overview 2059
36.6 Functional Description 2060
36.6.1 HUK Generator 2060
36.6.1.1 HUK Generation Process 2060
36.6.1.2 HUK Status Check 2061
36.6.2 Key Manager 2061
36.6.2.1 Random Deploy Mode 2062

36.6.2.2 AES Deploy Mode 2062
36.6.2.3 ECDH0 Deploy Mode 2064
36.6.2.4 ECDH1 Deploy Mode 2065
36.6.2.5 Private Key Recovery Mode 2067
36.6.2.6 key_info Export Mode 2068
36.7 Programming Procedures 2068
36.7.1 HUK Generator 2068
36.7.1.1 IDLE Phase 2069
36.7.1.2 PREP Phase 2069
36.7.1.3 LOAD Phase 2069
36.7.1.4 PROC Phase 2069
36.7.1.5 GAIN Phase 2069
36.7.1.6 POST Phase 2070
36.7.2 Key Manager 2070
36.7.2.1 IDLE Phase 2070
36.7.2.2 PREP Phase 2074
36.7.2.3 LOAD Phase 2074
36.7.2.4 PROC Phase 2074
36.7.2.5 GAIN Phase 2075
36.7.2.6 POST Phase 2075
36.8 Programming Examples 2075
36.8.1 Software Process in Chip Private Key Deploy Phase 2076
36.8.2 Software Process in Chip Normal Startup Phase 2076
36.9 Interrupts 2077
36.10 Memory Blocks 2077
36.11 Register Summary 2079
36.11.1 HUK Register Summary 2079
36.11.2 Key Manager Register Summary 2079
36.12 Registers 2081
36.12.1 HUK Registers 2081
36.12.2 Key Manager Registers 2085

VI Image and Voice Processing 2094

37 JPEG Codec 2095
37.1 Terminology 2095
37.2 Introduction 2096
37.3 Features 2097
37.4 Architectural Overview 2098
37.5 Functional Description 2099
37.5.1 JPEG Encoder 2100
37.5.1.1 Pause 2100
37.5.1.2 Color Space Conversion 2100
37.5.1.3 Configurable Quantization Coefficient Table 2101
37.5.1.4 Stuffed Zero Byte 2103
37.5.1.5 EOI Marker 2103

37.5.2 JPEG Decoder 2103
37.5.2.1 Multiple Chrominance Components 2103
37.5.2.2 Parsing RST Marker 2104
37.5.2.3 Configurable Quantization Coefficient Table 2105
37.5.2.4 Configurable Huffman Table 2105
37.5.2.5 Timeout Detection 2107
37.6 Interrupts 2107
37.7 Programming Procedures 2109
37.7.1 JPEG Encoder 2109
37.7.2 JPEG Decoder 2112
37.7.3 Reset 2115
37.8 Register Summary 2117
37.9 Registers 2118

38 Image Signal Processor (ISP) 2140
38.1 Introduction 2140
38.2 Terminology 2141
38.3 Feature List 2141
38.4 Architectural Overview 2142
38.5 Functional Description 2142
38.5.1 ISP_Header 2142
38.5.2 ISP_Pipeline 2143
38.5.2.1 Black Level Correction (BLC) 2143
38.5.2.2 Defective Pixel Correction (DPC) 2143
38.5.2.3 Bayer Filter (BF) 2144
38.5.2.4 Lens Shading Correction (LSC) 2144
38.5.2.5 Demosaic 2146
38.5.2.6 White Balance Gain (WBG) 2146
38.5.2.7 Color Correction Matrix (CCM) 2146
38.5.2.8 Gamma Correction 2146
38.5.2.9 RGB2YUV 2147
38.5.2.10 Sharpen 2147
38.5.2.11 Contrast/Hue/Saturation/Luminance Adjustment (COLOR) 2147
38.5.2.12 YUV_Limit and YUV2RGB 2148
38.5.2.13 Cropping (CROP) 2148
38.5.2.14 Automatic Exposure Statistics (AE) 2148
38.5.2.15 Automatic Focus Statistics (AF) 2149
38.5.2.16 Automatic White Balance Statistics (AWB) 2149
38.5.2.17 Histogram Statistics (HIST) 2150
38.5.3 ISP_Tail 2150
38.5.4 Sequence Control 2151
38.5.5 Shadow Registers 2151
38.5.6 CSI_CM 2151
38.5.7 CSI_Bridge 2152
38.5.8 Color Mode and Byte Order 2152
38.5.8.1 Output Pixel Layout Format 2152

38.5.8.2 Byte Order 2153
38.6 Interrupts 2153
38.7 Programming Procedures 2155
38.7.1 ISP Clock Reset Configuration 2155
38.7.2 CSI_Bridge Configuration 2156
38.7.3 VDMA Configuration 2156
38.7.4 ISP General Configuration 2157
38.7.5 Enabling ISP for Image Capture from MIPI-CSI 2157
38.7.6 Enabling ISP for Image Capture from DVP 2157
38.7.7 Enabling ISP for Image Capture from VDMA 2158
38.7.8 LSC LUT Configuration 2158
38.7.9 AE Configuration 2158
38.7.10 AF Configuration 2159
38.7.11 AWB Configuration 2160
38.7.12 HIST Configuration 2161
38.7.13 ISP_Pipeline Module Update Configuration 2162
38.8 Register Summary 2163
38.8.1 ISP Register Summary 2163
38.8.2 MIPI CSI_Bridge Register Summary 2167
38.9 Registers 2169
38.9.1 ISP Registers 2169
38.9.2 MIPI CSI_Bridge Registers 2243

39 Pixel-Processing Accelerator (PPA) 2255
39.1 Overview 2255
39.2 Terminology 2255
39.3 Features 2255
39.4 Architectural Overview 2256
39.5 Functional Description 2257
39.5.1 PPA Color Space 2257
39.5.1.1 SRM Color Space 2258
39.5.1.2 BLEND Color Space 2260
39.5.2 PPA 2D-DMA Linked List Configuration 2261
39.5.2.1 SRM Special Configuration 2262
39.5.3 Scaling - Rotation - Mirroring (SRM) 2263
39.5.3.1 Basic Functionality 2263
39.5.3.2 Pixel Block Rearrangement 2264
39.5.4 Layer Blending (BLEND) 2265
39.5.4.1 Basic Functionality 2265
39.5.4.2 Filled Image Output 2266
39.6 Interrupts 2266
39.7 Programming Procedures 2267
39.7.1 PPA Clock Reset Configuration 2267
39.7.2 SRM Configuration 2267
39.7.3 BLEND CLUT Configuration 2268
39.7.4 BLEND Configuration 2269

39.7.5 BLEND Image Filling Configuration 2269
39.7.6 Error Handling 2270
39.8 Register Summary 2271
39.9 Registers 2273

40 LCD and Camera Controller (LCD_CAM) 2300
40.1 Overview 2300
40.2 Features 2300
40.3 Functional Description 2301
40.3.1 Block Diagram 2301
40.3.2 Signal Description 2301
40.3.3 LCD_CAM Module Clocks 2302
40.3.3.1 LCD Clock 2302
40.3.3.2 Camera Clock 2304
40.3.4 LCD_CAM Reset 2305
40.3.5 LCD_CAM Data Format Control 2305
40.3.5.1 LCD Data Format Control 2305
40.3.5.2 Camera Data Format Control 2308
40.3.6 YUV-RGB Data Format Conversion 2309
40.3.6.1 YUV Formats 2309
40.3.6.2 Format Conversion Configuration 2310
40.3.7 LCD_CAM Timing 2312
40.3.7.1 LCD Timing (RGB Format) 2312
40.3.7.2 LCD Timing (I8080/MOTO6800 Format) 2313
40.4 Interrupts 2313
40.5 Software Configuration Process 2314
40.5.1 Configure LCD (RGB Format) as TX Mode 2314
40.5.2 Configure LCD (I8080/MOTO6800 Format) as TX Mode 2314
40.5.3 Configure Camera as RX Mode 2316
40.6 Register Summary 2317
40.7 Registers 2318

41 H264 Encoder 2336
41.1 Overview 2336
41.2 Terminology 2336
41.3 Features 2336
41.4 Architecture 2337
41.5 Functional Description 2339
41.5.1 Encoder Algorithm Core (ENC_CORE) 2339
41.5.1.1 Architecture 2339
41.5.1.2 Original Video Color Space Convert 2340
41.5.1.3 Quantization Result Decimation 2341
41.5.1.4 Motion Vector (MV) Merging 2342
41.5.1.5 MB Level Rate Control 2342
41.5.1.6 Region of Interest (ROI) 2344
41.5.1.7 Data Replace For Debug Function 2345

41.5.2 H264 Dedicated DMA 2346
41.5.2.1 Architecture 2346
41.5.2.2 Linked List Descriptor 2347
41.5.2.3 Transfer Initialization 2349
41.5.2.4 Transfer Reset 2350
41.5.2.5 Data Replace For Debug Function 2350
41.5.2.6 Channel Configuration 2351
41.6 Interrupts 2352
41.7 Programming Procedures 2354
41.7.1 GOP Mode 2355
41.7.2 Dual Stream Mode 2358
41.7.3 Soft Reset 2361
41.8 Register Summary 2363
41.8.1 H264 Encoder Register Summary 2363
41.8.2 H264 DMA Register Summary 2366
41.9 Registers 2374
41.9.1 H264 Encoder Registers 2374
41.9.2 H264 DMA Registers 2414

42 MIPI CSI 2592
42.1 Introduction 2592
42.2 Terminology 2592
42.3 Feature List 2593
42.4 Architectural Overview 2594
42.5 Functional Description 2595
42.5.1 Data Lane State 2595
42.5.2 MIPI RX D-PHY Operation 2596
42.5.2.1 No-Power State 2597
42.5.2.2 Shut-Down State 2597
42.5.2.3 AFE Initialization 2599
42.5.2.4 Control Mode 2599
42.5.2.5 High-Speed Data Reception Mode 2599
42.5.2.6 Escape Mode 2600
42.5.3 MIPI RX D-PHY Configuration 2601
42.5.3.1 MIPI RX D-PHY Programming Interface 2601
42.5.3.2 MIPI RX D-PHY Test Code 2603
42.5.4 Descrambler 2603
42.5.5 Error Management 2604
42.6 Interrupts 2605
42.7 Programming Procedures 2607
42.7.1 Start High-Speed Data Reception 2607
42.7.1.1 Clock and Reset 2607
42.7.1.2 MIPI RX D-PHY and MIPI Host Initialization 2608
42.7.2 Stop High-Speed Data Reception 2608
42.8 Register Summary 2609
42.9 Registers 2611

43 MIPI DSI 2633

43.1 Introduction 2633

43.2 Feature List 2633

43.3 Architectural Overview 2634

43.4 Functional Description 2635

43.4.1 DSI Bridge 2635

43.4.2 DSI Host 2636

43.4.2.1 Video Transmission 2637

43.4.2.2 Command Transmission 2637

43.4.2.2.1 Command Transmission in LP mode 2638

43.4.2.2.2 Command Transmission in HS mode 2640

43.4.2.3 Video Pattern Generator 2640

43.4.2.3.1 Color Bar Pattern 2640

43.4.2.3.2 BER Testing Pattern 2641

43.4.2.4 Errors 2642

43.4.3 D-PHY 2642

43.4.3.1 D-PHY Operating Modes 2642

43.4.3.1.1 No Power State 2643

43.4.3.1.2 Shutdown State 2643

43.4.3.1.3 PLL-Locking and AFE Initialization 2645

43.4.3.1.4 Control Mode 2645

43.4.3.1.5 High-Speed Data Transfer 2645

43.4.3.1.6 Escape Mode 2646

43.4.3.1.7 Turnaround 2646

43.4.3.2 PLL Parameters 2646

43.4.3.3 Test Codes 2648

43.4.3.3.1 Normal Operation 2648

43.4.3.3.2 PLL Bias Current Selector / Filter Capacitance Control / VCO Control 2648

43.4.3.3.3 PLL CP Control / PLL Lock Bypass for Initialization and for ULP 2649

43.4.3.3.4 PLL LPF and CP control 2650

43.4.3.3.5 PLL Input Divider Ratio 2651

43.4.3.3.6 PLL Loop Divider Ratio 2651

43.4.3.3.7 PLL Input and Loop Divider Ratios Control 2652

43.4.3.3.8 AFE/BIAS/Bandgap Analog Programmability 2653

43.4.3.3.9 HS RX Control of Lane 0 2654

43.4.4 Clocks and Display Resolution 2655

43.5 Interrupts 2658

43.6 Programming Procedures 2661

43.6.1 DSI Bridge Configuration 2661

43.6.2 D-PHY configuration 2662

43.6.3 MIPI DSI common configuration 2662

43.6.4 System memory data output flow configuration 2662

43.6.5 Video pattern output flow configuration 2662

43.6.6 Using DCS command to operate Display register 2662

43.7 Register Summary 2664

43.7.1 MIPI DSI Bridge Register Summary 2664

43.7.2 MIPI DSI Host Register Summary 2665
43.8 Registers 2668
43.8.1 MIPI DSI Bridge Register 2668
43.8.2 MIPI DSI Host Register 2687

44 Voice Activity Detection (VAD) 2723
44.1 Introduction 2723
44.2 Feature List 2723
44.3 Architectural Overview 2723
44.4 Functional Description 2724
44.4.1 Algorithm Parameter Configuration 2724
44.4.2 Wake-up Source Configuration 2726
44.5 Interrupts 2726
44.6 Programming Procedure 2727
44.6.1 Automatic Operation Mode 2727
44.6.2 Manual Operation Mode 2727
44.7 Register Summary 2728
44.8 Registers 2729

VII Connectivity Interface 2732

45 UART Controller (UART) 2733
45.1 Overview 2733
45.2 Features 2733
45.3 UART Structure 2734
45.4 Functional Description 2735
45.4.1 Clock and Reset 2735
45.4.2 UART FIFO 2736
45.4.3 Baud Rate Generation and Detection 2736
45.4.3.1 Baud Rate Generation 2736
45.4.3.2 Baud Rate Detection 2737
45.4.4 UART Data Frame 2738
45.4.5 AT_CMD Character Structure 2739
45.4.6 RS485 2739
45.4.6.1 Driver Control 2739
45.4.6.2 Turnaround Delay 2740
45.4.6.3 Bus Snooping 2740
45.4.7 IrDA 2740
45.4.8 Wakeup 2741
45.4.9 Flow Control 2742
45.4.9.1 Hardware Flow Control 2743
45.4.9.2 Software Flow Control 2744
45.4.10 GDMA Mode 2745
45.4.11 Interrupts 2746
45.5 Programming Procedures 2747
45.5.1 Register Type 2747

45.5.2 Detailed Steps 2748
45.5.2.1 Initializing UART*n* 2748
45.5.2.2 Configuring UART*n* Communication 2748
45.5.2.3 Enabling UART*n* 2749
45.6 Register Summary 2750
45.6.1 UART Register Summary 2750
45.6.2 LP UART Register Summary 2751
45.6.3 UHCI Register Summary 2752
45.7 Registers 2754
45.7.1 UART Registers 2754
45.7.2 LP UART Registers 2777
45.7.3 UHCI Registers 2797

46 SPI Controller (SPI) 2819
46.1 Overview 2819
46.2 Glossary 2819
46.3 Features 2821
46.4 Architectural Overview 2823
46.5 Functional Description 2824
46.5.1 Data Modes 2824
46.5.2 Introduction to Bus Signals 2824
46.5.3 Bit Read/Write Order Control 2827
46.5.4 Unaligned Byte Transfer 2829
46.5.5 Transfer Types 2829
46.5.6 CPU-Controlled Data Transfer 2829
46.5.6.1 CPU-Controlled Master Transfer 2830
46.5.6.2 CPU-Controlled Slave Transfer 2832
46.5.7 DMA-Controlled Data Transfer 2833
46.5.7.1 DMA Configuration 2833
46.5.7.2 DMA TX/RX Buffer Length Control 2834
46.5.8 Data Flow Control (Take GP-SPI as an Example) 2835
46.5.8.1 GP-SPI Functional Blocks 2835
46.5.8.2 Data Flow Control as Master 2836
46.5.8.3 Data Flow Control as Slave 2837
46.5.9 GP-SPI as a Master 2838
46.5.9.1 State Machine 2838
46.5.9.2 Register Configuration for State and Bit Mode Control 2840
46.5.9.3 Full-Duplex Communication (1-bit Mode Only) 2844
46.5.9.4 Half-Duplex Communication (1/2/4/(8)-bit Mode) 2845
46.5.9.5 DMA-Controlled Configurable Segmented Transfer 2847
46.5.10 GP-SPI Works as a Slave 2851
46.5.10.1 Configurable Communication Formats 2851
46.5.10.2 CMD Values Supported in Half-Duplex Communication 2852
46.5.10.3 Slave Single Transfer and Slave Segmented Transfer 2855
46.5.10.4 Configuration of Slave Single Transfer 2855
46.5.10.5 Configuration of Slave Segmented Transfer in Half-Duplex 2856

46.5.10.6 Configuration of Slave Segmented Transfer in Full-Duplex 2856
46.6 CS Setup Time and Hold Time Control 2857
46.7 Clock Control 2858
46.7.1 GP-SPI Clock Control 2858
46.7.2 LP-SPI Clock Control 2859
46.7.3 Clock Phase and Polarity 2859
46.7.4 Clock Control as Master 2860
46.7.5 Clock Control as Slave 2861
46.8 GP-SPI Timing Compensation 2861
46.9 LP-SPI Wake-Up 2863
46.10 Differences Among LP-SPI, GP-SPI2, and GP-SPI3 2864
46.10.1 Feature Differences 2864
46.10.2 Register Differences 2865
46.10.3 Interrupt Differences 2868
46.11 Interrupt 2869
46.11.1 Interrupt Description 2869
46.11.2 Interrupts Used in Master and in Slave (Take GP-SPI as an Example) 2871
46.12 Register Summary 2872
46.12.1 GP-SPI2 Register Summary 2872
46.12.2 GP-SPI3 Register Summary 2874
46.12.3 LP-SPI Register Summary 2875
46.13 Register 2876
46.13.1 GP-SPI2 Register 2876
46.13.2 GP-SPI3 Register 2918
46.13.3 LP-SPI Register 2952

47 I2C Controller (I2C) 2979
47.1 Overview 2979
47.2 Features 2979
47.3 I2C Architecture 2980
47.4 Functional Description 2982
47.4.1 Clock Configuration 2982
47.4.2 SCL and SDA Noise Filtering 2983
47.4.3 SCL Clock Stretching 2983
47.4.4 Generating SCL Pulses in Idle State 2984
47.4.5 Synchronization 2984
47.4.6 Open-Drain Output 2985
47.4.7 Timing Parameter Configuration 2986
47.4.8 Timeout Control 2988
47.4.9 Command Configuration 2988
47.4.10 TX/RX RAM Data Storage 2989
47.4.11 Data Conversion 2990
47.4.12 Addressing Mode 2990
47.4.13 $R/\overline{W}$ Bit Check in 10-bit Addressing Mode 2991
47.4.14 To Start the I2C Controller 2991
47.5 Functional Differences Between LP_I2C and I2C 2992

47.6 Programming Example 2992
47.6.1 $I2C_{master}$ Writes to $I2C_{slave}$ with a 7-bit Address in One Command Sequence 2992
47.6.1.1 Introduction 2993
47.6.1.2 Configuration Example 2993
47.6.2 $I2C_{master}$ Writes to $I2C_{slave}$ with a 10-bit Address in One Command Sequence 2994
47.6.2.1 Introduction 2995
47.6.2.2 Configuration Example 2995
47.6.3 $I2C_{master}$ Writes to $I2C_{slave}$ with Two 7-bit Addresses in One Command Sequence 2996
47.6.3.1 Introduction 2996
47.6.3.2 Configuration Example 2997
47.6.4 $I2C_{master}$ Writes to $I2C_{slave}$ with a 7-bit Address in Multiple Command Sequences 2998
47.6.4.1 Introduction 2998
47.6.4.2 Configuration Example 2999
47.6.5 $I2C_{master}$ Reads $I2C_{slave}$ with a 7-bit Address in One Command Sequence 3000
47.6.5.1 Introduction 3000
47.6.5.2 Configuration Example 3001
47.6.6 $I2C_{master}$ Reads $I2C_{slave}$ with a 10-bit Address in One Command Sequence 3002
47.6.6.1 Introduction 3003
47.6.6.2 Configuration Example 3003
47.6.7 $I2C_{master}$ Reads $I2C_{slave}$ with Two 7-bit Addresses in One Command Sequence 3005
47.6.7.1 Introduction 3005
47.6.7.2 Configuration Example 3006
47.6.8 $I2C_{master}$ Reads $I2C_{slave}$ with a 7-bit Address in Multiple Command Sequences 3007
47.6.8.1 Introduction 3008
47.6.8.2 Configuration Example 3009
47.7 Interrupts 3011
47.8 Register Summary 3013
47.8.1 I2C Register Summary 3013
47.8.2 LP_I2C Register Summary 3014
47.9 Registers 3016
47.9.1 I2C Registers 3016
47.9.2 LP_I2C Registers 3040

48 Analog I2C Controller 3062
48.1 Introduction 3062
48.2 Feature List 3062
48.3 Architectural Overview 3063
48.4 Functional Description 3063
48.5 Programming Procedures 3064
48.6 Register Summary 3065
48.7 Registers 3066

49 I2S Controller (I2S) 3070
49.1 Overview 3070
49.2 Terminology 3070
49.3 Features 3071

49.4 System Architecture 3072
49.5 Supported Audio Standards 3074
49.5.1 TDM Philips Standard 3075
49.5.2 TDM MSB Alignment Standard 3075
49.5.3 TDM PCM Standard 3076
49.5.4 PDM Standard 3076
49.6 I2S TX/RX Clock 3077
49.7 I2S Reset 3079
49.8 I2S Master/Slave Mode 3079
49.8.1 Master/Slave TX Mode 3079
49.8.2 Master/Slave RX Mode 3080
49.9 Transmitting Data 3080
49.9.1 Data Format Control 3081
49.9.1.1 Bit Width Control of Channel Valid Data 3081
49.9.1.2 Endian Control of Channel Valid Data 3081
49.9.1.3 A-law/μ-law Compression and Decompression 3082
49.9.1.4 Bit Width Control of Channel TX Data 3082
49.9.1.5 Bit Order Control of Channel Data 3082
49.9.2 Channel Mode Control 3083
49.9.2.1 TDM TX Mode 3083
49.9.2.2 PDM TX Mode 3084
49.9.3 Synchronous Counter 3087
49.10 Receiving Data 3088
49.10.1 Channel Mode Control 3088
49.10.1.1 TDM RX Mode 3088
49.10.1.2 PDM RX Mode 3088
49.10.2 Data Format Control 3089
49.10.2.1 Bit Order Control of Channel Data 3089
49.10.2.2 Bit Width Control of Channel Storage (Valid) Data 3090
49.10.2.3 Bit Width Control of Channel RX Data 3090
49.10.2.4 Endian Control of Channel Storage Data 3090
49.10.2.5 A-law/μ-law Compression and Decompression 3091
49.11 Event Task Matrix Feature 3091
49.12 I2S Interrupts 3092
49.13 Software Configuration Process 3092
49.13.1 Configure I2S as TX Mode 3092
49.13.2 Configure I2S as RX Mode 3093
49.14 Register Summary 3094
49.15 Registers 3096

50 LP I2S Controller 3116
50.1 Introduction 3116
50.2 Feature List 3116
50.3 Architectural Overview 3117
50.4 Supported Audio Standards 3118
50.4.1 TDM Philips Standard 3119

50.4.2 TDM MSB Alignment Standard 3119
50.4.3 TDM PCM Standard 3120
50.4.4 PDM Standard 3120
50.5 RX Clock 3120
50.6 Reset 3122
50.7 Master/Slave RX Mode 3122
50.8 Receiving Data 3123
50.8.1 Channel Mode Control 3123
50.8.1.1 TDM RX Mode 3123
50.8.1.2 PDM RX Mode 3124
50.8.2 Data Format Control 3124
50.8.2.1 Bit Order Control of Channel Data 3124
50.8.2.2 Bit Width Control of Channel RX Data 3125
50.8.2.3 Bit Width Control of Channel RX Data 3125
50.8.2.4 Endian Control of Channel Storage Data 3125
50.8.3 Internal Memory 3125
50.9 Interrupts 3126
50.10 Register Summary 3127
50.11 Registers 3128

51 Pulse Count Controller (PCNT) 3142
51.1 Introduction 3142
51.2 Feature List 3142
51.3 Architectural Overview 3143
51.4 Functional Description 3144
51.5 Interrupts 3145
51.6 Programming Procedures 3146
51.6.1 Channel 0 Incrementing Independently 3146
51.6.2 Channel 0 Decrementing Independently 3147
51.6.3 Channel 0 and Channel 1 Incrementing Together 3147
51.7 Register Summary 3149
51.8 Registers 3150

52 USB 2.0 High-Speed OTG 3158
52.1 Overview 3158
52.2 Glossary 3158
52.3 Features 3159
52.3.1 General Features 3159
52.3.2 Device Mode Features 3159
52.3.3 Host Mode Features 3159
52.4 Functional Description 3160
52.4.1 Controller Core and Interfaces 3160
52.4.2 Memory Layout 3160
52.4.2.1 Control & Status Registers (CSRs) 3161
52.4.2.2 FIFO Access 3161
52.4.3 FIFO and Queue Organization 3162

52.4.3.1 Host Mode FIFOs and Queues 3162
52.4.3.2 Device Mode FIFOs 3163
52.4.4 Interrupt Hierarchy 3164
52.4.5 DMA Modes and Slave Mode 3166
52.4.5.1 Slave Mode 3166
52.4.5.2 Buffer DMA Mode 3166
52.4.5.3 Scatter/Gather DMA Mode 3166
52.4.6 Transaction and Transfer Level Operation 3167
52.4.6.1 Transaction and Transfer Level Operation in DMA Mode 3167
52.4.6.2 Transaction and Transfer Level Operation in Slave Mode 3167
52.4.7 OTG 3168
52.5 Registers 3169

53 USB 2.0 Full-Speed OTG 3170

53.1 Overview 3170
53.2 Glossary 3170
53.3 Features 3171
53.3.1 General Features 3171
53.3.2 Device Mode Features 3171
53.3.3 Host Mode Features 3171
53.4 Functional Description 3172
53.4.1 Controller Core and Interfaces 3172
53.4.2 Memory Layout 3173
53.4.2.1 Control & Status Registers (CSRs) 3173
53.4.2.2 FIFO Access 3174
53.4.3 FIFO and Queue Organization 3174
53.4.3.1 Host Mode FIFOs and Queues 3175
53.4.3.2 Device Mode FIFOs 3175
53.4.4 Interrupt Hierarchy 3176
53.4.5 Slave Mode and DMA Modes 3178
53.4.5.1 Slave Mode 3178
53.4.5.2 Buffer DMA Mode 3178
53.4.5.3 Scatter/Gather DMA Mode 3178
53.4.6 Transaction and Transfer Level Operation 3179
53.4.6.1 Transaction and Transfer Level Operation in DMA Mode 3179
53.4.6.2 Transaction and Transfer Level Operation in Slave Mode 3179
53.5 OTG 3180
53.5.1 OTG Interface 3181
53.5.2 ID Pin Detection 3182
53.5.3 Session Request Protocol (SRP) 3182
53.5.3.1 A-Device SRP 3182
53.5.3.2 B-Device SRP 3183
53.5.4 Host Negotiation Protocol (HNP) 3184
53.5.4.1 A-Device HNP 3184
53.5.4.2 B-Device HNP 3185
53.6 Registers 3186

54 USB Serial/JTAG Controller (USB_SERIAL_JTAG) 3187
54.1 Overview 3187
54.2 Features 3187
54.3 Functional Description 3189
54.3.1 CDC-ACM USB Interface Functional Description 3189
54.3.2 CDC-ACM Firmware Interface Functional Description 3190
54.3.3 USB-to-JTAG Interface: JTAG Command Processor 3191
54.3.4 USB-to-JTAG Interface: CMD_REP Usage Example 3192
54.3.5 USB-to-JTAG Interface: Response Capture Unit 3193
54.3.6 USB-to-JTAG Interface: Control Transfer Requests 3193
54.4 Recommended Operation 3194
54.5 Interrupts 3195
54.6 Register Summary 3197
54.7 Registers 3199

55 Ethernet Media Access Controller (EMAC) 3223
55.1 Overview 3223
55.2 Features 3223
55.3 Ethernet MAC Architecture 3225
55.4 Functional Description 3225
55.4.1 EMAC_CORE 3225
55.4.1.1 Transmission 3225
55.4.1.2 Reception 3227
55.4.2 EMAC_MTL (MAC Transaction Layer) 3228
55.4.3 EMAC_DMA 3229
55.4.3.1 Transmit Descriptors 3229
55.4.3.2 Receive Descriptors 3234
55.4.4 PHY Interface 3240
55.4.4.1 Media Independent Interface: MII 3240
55.4.4.2 Reduced Media Independent Interface: RMII 3242
55.4.4.3 Station Management Agent (SMA) Interface 3244
55.4.5 MAC Address Filtering 3245
55.4.5.1 Unicast Destination Address Filtering 3245
55.4.5.2 Multicast Destination Address Filtering 3245
55.4.5.3 Broadcast Address Filter 3245
55.4.5.4 Unicast Source Address Filter 3245
55.4.5.5 Inverse Filtering Operation 3246
55.4.6 Energy Efficient Ethernet (EEE) 3247
55.4.6.1 Transmission 3247
55.4.6.2 Reception 3247
55.4.7 Source Address, VLAN, and CRC Control 3247
55.4.7.1 Source Address Control 3248
55.4.7.2 VLAN Control 3248
55.4.7.3 CRC Control 3248
55.4.8 Time Stamp 3248
55.4.8.1 Transmit Path 3248

55.4.8.2 Receive Path 3248
55.4.9 Remote Wakeup 3249
55.4.10 Good Transmitted and Received Frames 3249
55.5 Programming Procedures 3249
55.5.1 MAC System Layer Configuration 3249
55.5.2 EMAC Initial Configuration 3251
55.5.3 Starting Transmission 3251
55.5.4 Starting Reception 3251
55.5.5 TX Entering and Exiting the LPI State 3252
55.5.6 RX Entering and Exiting the LPI State 3252
55.6 Interrupts 3252
55.7 Register Summary 3254
55.8 Registers 3257

56 Two-Wire Automotive Interface (TWAI) 3304
56.1 Features 3304
56.2 Protocol Overview 3305
56.2.1 TWAI Properties 3305
56.2.2 TWAI Messages 3306
56.2.2.1 Data Frames and Remote Frames 3306
56.2.2.2 Error and Overload Frames 3308
56.2.2.3 Interframe Space 3310
56.2.3 TWAI Errors 3310
56.2.3.1 Error Types 3310
56.2.3.2 Error States 3311
56.2.3.3 Error State Transition 3311
56.2.3.4 Error Counter Rules 3311
56.2.4 TWAI Bit Timing 3312
56.2.4.1 Nominal Bit 3312
56.2.4.2 Hard Synchronization and Resynchronization 3313
56.3 Architectural Overview 3314
56.3.1 Bit Timing Logic 3315
56.3.2 Bit Stream Processor 3315
56.3.3 Acceptance Filter 3315
56.3.4 Receive FIFO 3315
56.3.5 Error Management Logic 3315
56.3.6 Registers Block 3315
56.4 Functional Description 3316
56.4.1 Modes 3316
56.4.1.1 Reset Mode 3316
56.4.1.2 Operation Mode 3317
56.4.2 Bit Timing 3317
56.4.3 Transmit and Receive Buffers 3318
56.4.3.1 Overview of Buffers 3318
56.4.3.2 Frame Information 3319
56.4.3.3 Frame Identifier 3319

56.4.3.4 Frame Data 3320
56.4.4 Receive FIFO and Data Overruns 3320
56.4.5 Acceptance Filter 3321
56.4.5.1 Single-Filter Mode 3322
56.4.5.2 Dual-Filter Mode 3322
56.4.6 Error Management 3324
56.4.6.1 Error Warning Limit 3325
56.4.6.2 Error Passive 3325
56.4.6.3 Bus-Off and Bus-Off Recovery 3325
56.4.7 Error Code Capture 3326
56.4.8 Arbitration Lost Capture 3327
56.4.9 Transceiver Auto-Standby 3328
56.5 Interrupts 3329
56.6 Register Summary 3333
56.7 Registers 3335

57 SD/MMC Host Controller (SDHOST) 3352
57.1 Overview 3352
57.2 Features 3352
57.3 SD/MMC External Interface Signals 3353
57.4 Functional Description 3353
57.4.1 SD/MMC Host Controller Architecture 3353
57.4.1.1 Bus Interface Unit (BIU) 3354
57.4.1.2 Card Interface Unit (CIU) 3354
57.4.2 Command Path 3355
57.4.3 Data Path 3355
57.4.3.1 Data Transmit 3355
57.4.3.2 Data Receive 3356
57.5 Software Restrictions for CIU Operations 3357
57.6 RAM for Receiving and Transmitting Data 3358
57.6.1 TX RAM Module 3358
57.6.2 RX RAM Module 3358
57.7 DMA Linked List 3358
57.8 DMA Descriptor Format 3359
57.9 Programming Procedures 3361
57.9.1 Initializing Registers 3361
57.9.2 Sending Commands 3362
57.9.3 Initializing DMA 3362
57.9.4 Initializing DMA Transmission 3363
57.9.5 Initializing DMA Reception 3363
57.10 Clock Phase Selection 3364
57.11 Interrupt 3364
57.12 Register Summary 3366
57.13 Registers 3368

58 LED PWM Controller (LEDC) 3397

58.1 Overview 3397
58.2 Features 3397
58.3 Architectural Overview 3397
58.4 Functional Description 3398
58.4.1 Timers 3398
58.4.1.1 Clock Source 3399
58.4.1.2 Clock Divider Configuration 3399
58.4.1.3 20-Bit Counter 3400
58.4.2 PWM Generators 3401
58.4.3 Duty Cycle Fading 3402
58.4.3.1 Linear Duty Cycle Fading 3403
58.4.3.2 Gamma Curve Fading 3403
58.4.3.3 Suspend and Resume Duty Cycle Fading 3404
58.4.4 Event Task Matrix Feature 3405
58.5 Interrupts 3406
58.6 Programming Procedures 3406
58.7 Memory Blocks 3408
58.8 Register Summary 3409
58.9 Registers 3411

59 Motor Control PWM (MCPWM) 3423
59.1 Overview 3423
59.2 Features 3423
59.3 Modules 3426
59.3.1 Overview 3426
59.3.1.1 Timer Module 3426
59.3.1.2 Operator Module 3426
59.3.1.3 Capture Module 3427
59.3.1.4 ETM Module 3427
59.3.2 PWM Timer Module 3428
59.3.2.1 Configurations of the PWM Timer Module 3428
59.3.2.2 PWM Timer's Working Modes and Timing Event Generation 3428
59.3.2.3 Shadow Register of PWM Timer 3434
59.3.2.4 PWM Timer Synchronization and Phase Locking 3434
59.3.3 PWM Operator Module 3434
59.3.3.1 PWM Generator Module 3436
59.3.3.2 Dead Time Generator Module 3448
59.3.3.3 PWM Carrier Module 3453
59.3.3.4 Fault Detection Module 3455
59.3.4 Capture Module 3456
59.3.4.1 Introduction 3456
59.3.4.2 Capture Timer 3457
59.3.4.3 Capture Channel 3457
59.3.5 ETM Module 3457
59.3.5.1 Overview 3457
59.3.5.2 MCPWM-Related ETM Events 3457

59.3.5.3 MCPWM-Related ETM Tasks 3458
59.4 Interrupts 3459
59.5 Register Summary 3460
59.6 Registers 3463

60 Remote Control Peripheral (RMT) 3510
60.1 Overview 3510
60.2 Features 3510
60.3 Functional Description 3511
60.3.1 Architecture 3511
60.3.2 RAM 3512
60.3.2.1 Structure of RAM 3512
60.3.2.2 Use of RAM 3513
60.3.2.3 RAM Access 3513
60.3.3 Clock 3514
60.3.4 Transmitter 3514
60.3.4.1 Normal TX Mode 3515
60.3.4.2 Wrap TX Mode 3515
60.3.4.3 TX Modulation 3515
60.3.4.4 Continuous TX Mode 3516
60.3.4.5 Simultaneous TX Mode 3516
60.3.5 Receiver 3516
60.3.5.1 Normal RX Mode 3516
60.3.5.2 Wrap RX Mode 3517
60.3.5.3 RX Filtering 3517
60.3.5.4 RX Demodulation 3517
60.3.6 Configuration Update 3517
60.4 Interrupts 3518
60.5 Register Summary 3520
60.6 Registers 3522

61 Parallel IO Controller (PARLIO) 3538
61.1 Introduction 3538
61.2 Glossary 3538
61.3 Features 3538
61.4 Architectural Overview 3540
61.5 Functional Description 3540
61.5.1 Clock Generator 3540
61.5.2 Clock & Reset Restriction 3541
61.5.3 Master-Slave Mode 3543
61.5.4 Receive Modes of the RX Unit 3544
61.5.4.1 Level Enable Mode 3544
61.5.4.2 Pulse Enable Mode 3544
61.5.4.3 Software Enable Mode 3545
61.5.5 RX Unit GDMA SUC EOF Generation 3546
61.5.6 TX Unit EOF Generation 3546

61.5.7 RX Unit Timeout 3546
61.5.8 Unlimited Length Data Transmission of TX Unit 3546
61.5.9 TX Chip Select Function 3547
61.5.10 Output Clock Gating of TX Unit 3547
61.5.11 Bus Idle Value of TX Unit 3547
61.5.12 Data Transfer in a Single Frame 3547
61.5.13 Bit Reversal in One Byte 3548
61.6 Interrupts 3548
61.7 Programming Procedures 3549
61.7.1 Data Receiving Operation Process 3549
61.7.2 Data Transmitting Operation Process 3549
61.8 Application Examples 3550
61.8.1 Co-working with SPI 3550
61.8.2 Co-working with I2S 3552
61.8.3 Co-working with LCD 3552
61.9 Register Summary 3554
61.10 Registers 3555

62 BitScrambler 3566
62.1 Introduction 3566
62.2 Feature List 3567
62.3 Architectural Overview 3567
62.3.1 Data Path 3568
62.3.2 Control Path 3569
62.4 Functional Description 3570
62.4.1 Instructions 3570
62.4.2 Configuration Registers 3575
62.5 Programming Procedures 3577
62.6 Register Summary 3579
62.7 Registers 3580

VIII Analog Signal Processing 3592

63 Touch Sensor (TOUCH) 3593
63.1 Terminology 3593
63.2 Feature List 3593
63.3 Architectural Overview 3594
63.3.1 Touch Panel 3594
63.3.2 Capacitive Touch Pin 3595
63.3.3 Touch Sensor 3595
63.4 Functional Description 3597
63.4.1 Touch FSM 3597
63.4.2 Sampled Signal Preprocessing 3598
63.4.2.1 Measurement Process 3598
63.4.2.2 Trigger Source of Measurement 3600
63.4.2.3 Scan Mode 3600

63.4.2.4 Frequency Hopping 3601
63.4.3 Touch Detection 3603
63.4.3.1 Sampled Value 3603
63.4.3.2 Hardware Touch Detection 3604
63.4.4 Proximity Mode 3604
63.4.5 Sleep Mode 3605
63.4.6 Moisture Tolerance 3606
63.4.7 Water Rejection 3606
63.5 Interrupts 3606
63.6 Register Summary 3609
63.6.1 Interrupt and Status Register Summary 3609
63.6.2 Configuration Register Summary 3610
63.7 Registers 3613
63.7.1 Interrupt and Status Registers 3613
63.7.2 Configuration Registers 3620

64 Temperature Sensor (TSENS) 3635
64.1 Overview 3635
64.2 Features 3635
64.3 Architecture 3635
64.4 Functional Description 3636
64.4.1 Temperature Sensor Power Up 3636
64.4.2 Temperature Sensor Clock 3636
64.4.3 Wake-Up Modes for Automatic Temperature Monitoring 3636
64.4.4 Temperature Measurement Range and Offset 3637
64.4.5 Data Conversion 3637
64.5 Event Task Matrix Feature 3637
64.6 Interrupts 3638
64.7 Programming Procedure 3638
64.8 Register Summary 3639
64.9 Registers 3640

65 ADC Controller (ADC) 3646
65.1 Overview 3646
65.2 Terminology 3647
65.3 Features 3647
65.4 Architecture 3648
65.5 Functional Description 3649
65.5.1 SAR ADC Power Up 3649
65.5.2 SAR ADC Channels 3649
65.5.3 SAR ADC Clock 3650
65.5.4 SAR ADC Conversion and Attenuation 3650
65.5.5 HP ADC FSM 3651
65.5.6 HP ADC Pattern Table 3652
65.5.7 HP ADC Pattern Configuration Example for Multi-channel Sampling 3653
65.5.8 Dual HP ADC Sampling Control 3654

65.5.9 HP ADC Filters 3654
65.5.10 HP ADC Threshold Monitors 3654
65.5.11 HP ADC GDMA Support 3655
65.5.12 LP ADC Wake-Up Modes for Automatic Monitoring 3655
65.6 Event Task Matrix Feature 3655
65.7 Interrupts 3656
65.8 Programming Procedure 3657
65.8.1 HP ADC Multi-Channel Sampling Mode 3657
65.8.2 LP ADC One-shot Sampling Mode 3657
65.8.3 LP ADC Automatic Monitoring 3657
65.9 Register Summary 3659
65.9.1 HP ADC Register Summary 3659
65.9.2 LP ADC Register Summary 3659
65.10 Registers 3661
65.10.1 HP ADC Registers 3661
65.10.2 LP ADC Registers 3671

66 Analog Voltage Comparator 3688
66.1 Introduction 3688
66.2 Feature List 3688
66.3 Architectural Overview 3688
66.4 Functional Description 3689
66.5 Event Task Matrix Feature 3690
66.6 Interrupts 3690
66.7 Programming Procedures 3691

IX Appendix 3692

Related Documentation and Resources 3693

Glossary 3694
Abbreviations for Peripherals 3694
Abbreviations Related to Registers 3695
Access Types for Registers 3696

Programming Reserved Register Field 3698
Introduction 3698
Programming Reserved Register Field 3698

Interrupt Configuration Registers 3699

Revision History 3700

List of Tables

2.4-1 CPU Address Map 71
2.7-2 HWLP Instructions 126
2.8-1 Address Decoding for Memory-Mapped Register Access 129
2.8-2 Internal Memory Address Map 129
2.9-1 CLIC Interrupt Mapping 131
2.9-2 CLIC Level and Priority Encoding with CLICINTCTLBITS=3 132
2.9-4 Core-Local Interrupt Sources 137
2.11-1 JTAG Control 154
2.11-3 Virtual address in DPC upon Debug Mode Entry 157
2.11-5 NAPOT encoding for maddress 162

3.3-1 Trace Encoder Parameters 182
3.5-1 Optional sideband signals 184
3.6-1 Header Format 187
3.6-2 Index Format 187
3.6-3 Packet format 3 subformat 0 187
3.6-4 Packet format 3 subformat 1 188
3.6-5 Packet format 3 subformat 3 188
3.6-6 Packet format 2 189
3.6-7 Packet format 1 with address 190
3.6-8 Packet format 1 without address 191
3.7-1 CORE*n*_TRACE's Internal Interrupt Sources 192
3.8-1 Debug module trigger support (the action field of the mcontrol register) 193
3.8-2 Trace Status 195

4.4-1 Register List of ESP32-P4 Extended Instruction Set 212
4.4-2 Data Format and Alignment 215
4.4-3 Rounding Mode Configuration 217
4.4-4 Rounding Operation Results 217
4.5-1 Extended Instruction List 218
4.5-2 Read Instructions 221
4.5-3 Write Instructions 222
4.5-4 Data Exchange Instructions 222
4.5-5 Vector Addition Instructions 223
4.5-6 Vector Multiplication Instructions 224
4.5-7 Vector Complex Multiplication Instructions 225
4.5-8 Vector Multiplication Accumulation Instructions 225
4.5-9 Vector and Scalar Multiplication Accumulation Instructions 226
4.5-10 Other Instructions 227
4.5-11 Comparison Instructions 227
4.5-12 Bitwise Logical Instructions 228
4.5-13 Shift Instructions 228
4.5-14 Butterfly Computation Instructions 229
4.5-15 Bit Reverse Instruction 229

4.5-16 Real Number FFT Instructions 230
4.6-1 Five-Stage Pipeline of the ESP32-P4 Processor 230
4.7-1 Instruction Field Names and Descriptions 233

5.4-1 LP CPU Interrupt Mapping 644
5.4-2 LP CPU Exception Causes 645
5.7-1 Performance Counter 653
5.10-1 Wake Sources 657

6.4-1 GDMA-AHB Selecting Peripherals via Register Configuration 663
6.4-2 GDMA-AXI Selecting Peripherals via Register Configuration 664

7.4-1 Hardware Handshaking Interface and Peripherals Mapping Table 776
7.5-1 Register Update Methods for Multi-Block Transfer 782

8.4-1 Descriptor Field Description 831
8.4-2 Selecting Peripherals via Register Configuration in TX Direction 835
8.4-3 Selecting Peripherals via Register Configuration in RX Direction 835
8.4-4 Recommended Configurations for Macroblock Reordering in TX Direction 836
8.4-5 Recommended Configuration Table for Macroblock Reordering in 2D-DMA Receive Direction (Without Color Format Conversion) 836
8.4-6 Parameter Configuration for Color Space Conversion in TX Direction 838
8.4-7 Parameter Configuration for Color Space Conversion in RX Direction 839
8.4-8 RGBB888 Storage Format 840
8.4-9 YUV444 Storage Format 840
8.4-10 YUV422 Storage Format 840

9.3-1 Memory Address Mapping 904
9.3-2 Module/Peripheral Address Mapping 912
9.4-1 Parameters in eFuse BLOCK0 918
9.4-2 Secure Key Purpose Values 924
9.4-3 Parameters in BLOCK1 to BLOCK10 925
9.4-4 User Read Registers and Programming Registers 930
9.4-5 Configuration of Default VDDQ Timing Parameters 931
9.4-6 eFuse's Internal Interrupt Sources 932

10.4-1 HP GPIO Wakeup Signal Trigger and Clear Conditions 987
10.4-2 LP GPIO Wakeup Signal Trigger and Clear Conditions 988
10.8-1 Bit Used to Control IO MUX Functions in Light-sleep Mode 995
10.12-1 Peripheral Signals via HP GPIO Matrix 999
10.13-1 Peripheral Signals via LP GPIO Matrix 1010
10.14-1 HP IO MUX Pin Functions 1011
10.15-1 LP IO MUX Pin Functions 1014
10.16-1 GPIO Pin Analog Functions 1015
10.18-1 HP IO MUX and HP GPIO Matrix's Internal Interrupt Sources 1017
10.18-2 LP IO MUX and LP GPIO Matrix's Internal Interrupt Sources 1018

11.1-1 HP CPU Reset Source 1082

11.1-2 LP CPU Reset Source 1083
11.2-1 ROOT_CLK Clock Source 1086
11.2-2 LP_SLOW_CLK Clock Source Selection 1087
11.2-3 LP_FAST_CLK Clock Source Selection 1088
11.2-4 Derived HP Clock sources 1089
11.2-5 HP Peripheral Clocks 1089
11.2-6 Derived LP Clock Sources 1090
11.2-7 LP Peripheral Clocks 1091

12.2-1 Default Configuration of Strapping Pins 1220
12.2-2 Boot Mode Control 1220
12.2-3 ROM Message Printing Control 1222
12.2-4 JTAG Signal Source Control 1223

13.4-1 CPU Peripheral Interrupt Source Mapping/Status Registers and Peripheral Interrupt Sources 1228

14.3-1 Selectable Events for ETM Channel*n* 1269
14.3-2 Mappable Tasks for ETM Channel*n* 1275

15.4-1 Wake-up Sources 1432
15.4-2 HP System Peripherals' Function Clocks 1435
15.4-3 VO2/3/4 Registers 1439
15.4-4 Regulator Power Control 1439
15.4-5 Over-Current Protection Control 1440
15.4-6 Regulator Waiting Counter 1440
15.5-1 Preset Power Modes 1441

16.5-1 UNIT*n* Configuration Bits 1509
16.5-2 Trigger Point 1511
16.5-3 Synchronization Operation for Configuration Registers 1511
16.6-1 System Timer's Internal Interrupt Sources 1512

17.3-1 Alarm Generation When Up-Down Counter Increments 1535
17.3-2 Alarm Generation When Up-Down Counter Decrements 1535
17.3-3 TIMG*n*'s Internal Interrupt Sources 1537

18.2-1 Timeout Actions 1560

19.3-1 Configure RTC Timer to Log the Occurrence Time of Specific Events 1573
19.3-2 Target times Configuration 1573
19.5-1 RTC Timer's Internal Interrupt Sources 1574

20.1-1 Management Areas by PMP and APM 1586
20.5-1 APM Internal Interrupt Sources 1592

21.3-1 System Registers Interrupt Sources 1698

22.5-1 HP CPU0/1 Packet Format 1800
22.5-2 HP CPU LOST Packet Format 1800
22.5-3 DMA_2 Packet Format 1803

22.5-4 DMA_3 Packet Format 1803
22.5-5 DMA LOST Packet Format 1803

23.5-1 Setting of Measurement Metrics in Metric Select Registers 1857
23.5-2 Correspondence between Metric Select Registers and Range Statistics Metrics 1857
23.5-3 Correspondence between Agent Select Register Configuration Values and Actual Agents 1858
23.6-1 AXI PERF MON Internal Interrupt Sources 1862

24.3-1 LP Mailbox's Internal Interrupt Sources 1896

27.4-1 AES Accelerator Working Mode 1929
27.4-2 Key Length and Encryption/Decryption 1929
27.5-1 Working Status under Typical AES Working Mode 1930
27.5-2 Text Endianness Type for Typical AES 1930
27.5-2 Text Endianness Type for Typical AES 1931
27.5-3 Key Endianness Type for AES-128 Encryption and Decryption 1932
27.5-4 Key Endianness Type for AES-256 Encryption and Decryption 1932
27.6-1 Block Cipher Mode 1934
27.6-2 Working Status under DMA-AES Working mode 1934
27.6-3 TEXT-PADDING 1935
27.6-4 Text Endianness for DMA-AES 1935
27.9-1 AES's Internal Interrupt Sources 1941

28.3-1 ECC Accelerator Memory Blocks 1951
28.4-1 ECC Accelerator Curve Mode Selection 1953
28.4-2 Working Modes of ECC Accelerator 1953

29.2-1 HMAC Purposes and Configuration Value 1966

30.3-1 Acceleration Performance 1986
30.4-1 RSA's Internal Interrupt Source 1986
30.5-1 RSA Accelerator Memory Blocks 1987

31.3-1 SHA Accelerator Working Mode 1993
31.3-2 SHA Hash Algorithm Selection 1993
31.4-4 The Storage and Length of Message Digest from Different Algorithms 2000
31.5-1 SHA's Internal Interrupt Sources 2000

33.4-1 ECDSA_DS Elliptic Curve Selection 2022
33.4-2 ECDSA_DS SHA Algorithm 2022
33.4-3 ECDSA_DS Working State 2022
33.6-1 ECDSA_DS Memory Blocks 2029

34.4-1 Key generated based on Key_A, Key_B, and Key_C 2040
34.4-2 Mapping Between Offsets and Registers 2042

36.6-1 HUK Generator Working Mode 2060
36.9-1 Key Manager and HUK Generator's Internal Interrupt Sources 2077
36.10-1 Key Manager Memory Blocks 2078

36.10-2 HUK Generator Memory Blocks 2078

37.1-2 Marker Code Assignments 2096
37.2-1 The Number of Data Units in the MCU of Different Image Format 2096
37.5-1 Supported Image Formats for Color Space Conversion 2101
37.5-2 Position Number of Coefficients in a Quantization Coefficient Table (Example) 2101
37.5-3 Recommended 8-bit Precision Quantization Coefficient Table for the Luminance Component 2102
37.5-4 Recommended 8-bit Precision Quantization Coefficient Table for the Chrominance Component 2102
37.5-5 Sampling Factor and Image Format 2104
37.7-1 JPEG Encoder hb and vb Configuration 2110
37.7-2 DMA2D_OUT_MACRO_BLOCK_SIZE_CH*n* Configuration 2111
37.7-3 JPEG Decoder JPEG_HA and JPEG_VA Configuration 2112
37.7-4 JPEG Decoder hb and vb Configuration 2113
37.7-5 DMA2D_IN_MACRO_BLOCK_SIZE_CH*n* Configuration 2114
37.7-6 2D DMA Color Space Conversion Configuration 2114

38.5-1 Gain Storage Sequence in LSC LUTs 2145
38.5-2 ISP Output Pixel Layout Format 2152
38.6-1 ISP's Internal Interrupt Sources 2153

39.5-1 Pixel Storage Order 2257
39.5-2 PPA SRM Alpha Configuration 2259
39.5-3 BLEND CLUT Address 2261
39.5-4 PPA Linked List Configuration 2262
39.5-5 Automatic Padding Block Size Configuration 2263
39.5-6 Number of Pixel Block Rearrangements in SRM Output 2264
39.6-1 SRM Parameter Error Types 2266
39.6-2 BLEND Parameter Error Types 2267

40.3-1 LCD_CAM Input and Output Signal Descriptions 2302
40.3-2 LCD Data Format Control – Stage One 2307
40.3-3 LCD Output Data Reordering 2308
40.3-4 Camera Data Format Control 2309
40.3-5 Conversion Mode Control in 24-bit Mode 2311
40.3-6 Conversion Mode Control in Other Modes 2311

42.5-1 Lane State Descriptions 2596
42.5-2 Frequency Ranges Supported by MIPI RX D-PHY 2598
42.5-3 Test Data in HS RX Control of Lane 0 2603
42.5-4 Test Dout in HS RX Control of Lane 0 2603
42.5-5 Scrambler PRBS Initial Seed Values for Lane 1 and Lane 2 2604
42.5-6 Detectable Errors by MIPI CSI-2 Host 2604

43.4-1 input pixel arrangement 2635
43.4-2 DPI bit arrangement 2636
43.4-3 Color Bar Coding 2641
43.4-4 Frequency Ranges Supported by MIPI BD D-PHY 2644
43.4-5 M and N ratios 2647

43.4-6 VCO ranges 2647
43.4-7 PLL CP and LPF Control Bits 2647
43.4-8 Test Data of 0x10 2648
43.4-9 Test Dout of 0x10 2649
43.4-10 Test Data of 0x11 2649
43.4-11 Test Dout of 0x11 2650
43.4-12 Test Data of 0x12 2650
43.4-13 Test Dout of 0x12 2651
43.4-14 Test Data of 0x17 2651
43.4-15 Test Dout of 0x17 2651
43.4-16 Test Data of 0x18 2651
43.4-17 Test Dout of 0x18 2652
43.4-18 Test Dout of 0x18 2652
43.4-19 Test Data of 0x19 2652
43.4-20 Test Dout of 0x19 2653
43.4-21 Test Data of 0x22 2653
43.4-22 Test Dout of 0x22 2654
43.4-23 Test Dout of 0x22 2654
43.4-24 Test Data of 0x44 2655
43.4-25 Test Dout of 0x44 2655
43.5-1 DSI Bridge interrupts 2658
43.5-2 DSI Host Interrupts 2658

45.2-1 UART and LP UART Feature Comparison 2733
45.4-1 UART_CHAR_WAKEUP Mode Configuration 2742

46.5-1 Data Modes Supported by GP-SPI and LP-SPI 2824
46.5-2 Functional Description of SPI2/SPI3 and LP_SPI Bus Signals 2824
46.5-3 SPI2 Bus Signals Used in Various SPI Modes 2825
46.5-4 SPI3 Bus Signals Used in Various SPI Modes 2826
46.5-5 LP-SPI Bus Signals Used in Various SPI Modes 2827
46.5-6 Bit Order Control in GP-SPI (Take GP-SPI2 as an Example) 2828
46.5-7 Bit Order Control in LP-SPI 2828
46.5-8 Supported Transfer Types as Master or Slave 2829
46.5-9 Interrupt Trigger Condition on GP-SPI Data Transfer as Slave 2834
46.5-10 Registers Used for State Control in 1/2/4/8-bit Modes 2841
46.5-11 Sending Sequence of Command Value 2843
46.5-12 Sending Sequence of Address Value 2843
46.5-13 BM Table for CONF State 2849
46.5-14 An Example of CONF buffer*i* in Segment *i* 2850
46.5-15 BM Bit Value and Register to Be Updated in This Example 2850
46.5-16 CMD Values Supported in SPI Mode 2853
46.5-16 CMD Values Supported in SPI Mode 2854
46.5-17 CMD Values Supported in QPI Mode 2854
46.7-1 Clock Phase and Polarity Configuration as Master 2860
46.7-2 Clock Phase and Polarity Configuration as Slave 2861
46.9-1 LP-SPI Wake-Up Configuration When Working as a Slave 2864

46.10-1 Feature Differences Among GP-SPI2, GP-SPI3, and LP-SPI 2864
46.10-2 Register Differences 2865
46.10-3 Interrupt Differences 2868
46.10-3 Interrupt Differences 2869
46.11-1 Internal Interrupt Sources of LP-SPI and GP-SPI 2869
46.11-2 GP-SPI Master Mode Interrupts 2871
46.11-3 GP-SPI Slave Mode Interrupts 2872

47.3-1 I2C Timing Parameters (Cited from Table 5 in The I2C-bus specification Version 2.1) 2982
47.4-1 I2C Synchronous Registers 2985

49.4-1 I2S*n* Signal Description 3074
49.9-1 Bit Width of Channel Valid Data 3081
49.9-2 Endian of Channel Valid Data 3081
49.9-3 Data-Fetching Control in PDM Mode 3084
49.9-4 I2S*n* Channel Control for Raw PDM Data 3085
49.9-5 I2S*0* PCM-to-PDM Data Output 3085
49.10-1 PDM-to-PCM Data Input 3089
49.10-2 Channel Storage Data Width 3090
49.10-3 Channel Storage Data Endian 3090

50.3-1 LP I2S Signal Description 3118
50.8-1 PDM-to-PCM Data Input 3124
50.8-2 Channel Storage Data Endian 3125

51.4-1 Counter Mode. Rising edge of Input Pulse Signal. Control Signal in Low State 3144
51.4-2 Counter Mode. Rising edge of Input Pulse Signal. Control Signal in High State 3144
51.4-3 Counter Mode. Falling Edge of Input Pulse Signal. Control Signal in Low State 3144
51.4-4 Counter Mode. Falling Edge of Input Pulse Signal. Control Signal in High State 3145
51.5-1 PCNT's Internal Interrupt Sources 3146

52.4-1 IN and OUT Transactions in Slave Mode 3167

53.4-1 IN and OUT Transactions in Slave Mode 3179
53.5-1 UTMI+ OTG Interface 3181

54.3-1 Standard CDC-ACM Control Requests 3189
54.3-2 CDC-ACM Settings with RTS and DTR 3190
54.3-3 Commands of a Nibble 3192
54.3-4 USB-to-JTAG Control Requests 3193
54.3-5 JTAG Capability Descriptors 3194
54.4-1 Reset SoC into Download Mode 3195
54.4-2 Reset SoC into Booting from flash 3195

55.4-1 Transmit Descriptor 0 (TDES0) 3229
55.4-2 Transmit Descriptor 1 (TDES1) 3233
55.4-3 Transmit Descriptor 2 (TDES2) 3233
55.4-4 Transmit Descriptor 3 (TDES3) 3233
55.4-5 Transmit Descriptor 6 (TDES6) 3234

55.4-6 Transmit Descriptor 7 (TDES7) 3234
55.4-7 Receive Descriptor 0 (RDES0) 3234
55.4-8 Receive Descriptor 1 (RDES1) 3237
55.4-9 Receive Descriptor 2 (RDES2) 3238
55.4-10 Receive Descriptor 3 (RDES3) 3238
55.4-11 Receive Descriptor 4 (RDES4) 3238
55.4-12 Receive Descriptor 6 (RDES6) 3240
55.4-13 Receive Descriptor 7 (RDES7) 3240
55.4-14 Destination Address Filtering 3246
55.4-15 Source Address Filtering 3247

56.2-1 Data Frames and Remote Frames in SFF and EFF 3307
56.2-2 Error Frame 3309
56.2-3 Overload Frame 3309
56.2-4 Interframe Space 3310
56.2-5 Segments of a Nominal Bit Time 3313
56.4-1 Bit Information of TWAI_BUS_TIMING_0_REG (0x18) 3317
56.4-2 Bit Information of TWAI_BUS_TIMING_1_REG (0x1c) 3318
56.4-3 Buffer Layout for Standard Frame Format and Extended Frame Format 3318
56.4-4 TX/RX Frame Information (SFF/EFF); TWAI Address 0x40 3319
56.4-5 TX/RX Identifier 1 (SFF); TWAI Address 0x44 3319
56.4-6 TX/RX Identifier 2 (SFF); TWAI Address 0x48 3319
56.4-7 TX/RX Identifier 1 (EFF); TWAI Address 0x44 3320
56.4-8 TX/RX Identifier 2 (EFF); TWAI Address 0x48 3320
56.4-9 TX/RX Identifier 3 (EFF); TWAI Address 0x4c 3320
56.4-10 TX/RX Identifier 4 (EFF); TWAI Address 0x50 3320
56.4-11 Bit Information of TWAI_ERR_CODE_CAP_REG (0x30) 3326
56.4-12 Bit Information of Bits SEG.4 - SEG.0 3326
56.4-13 Bit Information of TWAI_ARB_LOST_CAP_REG (0x2c) 3327
56.5-1 TWAI's Internal Interrupt Sources 3329

57.3-1 SD/MMC Signal Description 3353
57.8-1 DES0 Descriptor Field 3359
57.8-2 DES1 Descriptor Field 3360
57.8-3 DES2 Descriptor Field 3361
57.8-4 DES3 Descriptor Field 3361

58.4-1 Commonly-used Frequencies and Resolutions 3401
58.7-1 LEDC Memory Blocks 3408

59.3-1 Functions of Each Submodule in the Operator Module 3427
59.3-2 Timing Events Used in PWM Generator 3437
59.3-3 Timing Events Priority When PWM Timer Increments 3437
59.3-4 Timing Events Priority when PWM Timer Decrements 3438
59.3-5 Dead Time Generator Switches Control Fields 3449
59.3-6 Typical Dead Time Generator Operating Modes 3449
59.3-7 MCPWM-Related ETM Events 3457

59.3-8 ETM Related Tasks 3458

60.3-1 Configuration Update 3518

61.5-1 Operations to Reset AXI Clock Domain with Clock Restrictions 3542
61.5-2 Requirements for TX Unit Operating as Slave with Clock Restrictions 3543
61.5-3 Requirements for RX Unit Operating as Slave with Clock Restrictions 3544
61.6-1 PARLIO's Internal Interrupt Sources 3548

62.4-1 Instruction Structure 3570
62.4-2 Effective CTL_MUX_SEL_n values when CTL_MUX_REL is 1 3571
62.4-3 CTL_MUX_SEL Values 3572
62.4-4 Instruction Opcode 3573
62.4-5 LOOP Opcode 3573
62.4-6 ADD Opcode 3574
62.4-7 IF Opcode 3574
62.4-8 IFN Opcode 3574
62.4-9 LDCTD Opcode 3575
62.4-10 LDCTI Opcode 3575
62.4-11 Settings for EOF modes 3577

63.3-1 ESP32-P4 Capacitive Touch Pins 3595
63.3-2 Capacitive Touch Sensor Output Signal 3596
63.4-1 touch_smooth_data Calculation 3603
63.4-2 benchmark Algorithm 3604
63.5-1 Touch Sensor's Internal Interrupt Sources 3608

64.4-1 Temperature Measurement Range and Offset 3637

65.5-1 SAR ADC Channels and Chip Pins 3649

66.4-1 Mapping Between PAD and GPIO 3689
66.6-1 Analog Voltage Comparator's Internal Interrupt Sources 3691

66.7-4 Configuration of ENA/RAW/ST Registers 3699

List of Figures

1.0-1 ESP32-P4 Bus Architecture 66

2.1-1 CPU Core Complex Block Diagram 68
2.11-1 Debug System Overview 152
2.12-1 Two-bit Saturation Counter 170
2.13-1 CPU Core State Transition Using RunStall 177

3.0-1 Trace Encoder Overview 180
3.2-1 Trace Flow Overview 181
3.6-1 Trace packet Format 186

4.3-1 Xespv2p2 Internal Structure (MAC) 211
4.6-1 Interlock Caused by Operand Dependency 231
4.6-2 Hardware Resource Hazard 231
4.6-3 Control Hazard 232

5.1-1 LP CPU Overview 633
5.10-1 Wake-Up and Sleep Flow of LP CPU 656

6.1-1 Modules that Share GDMA-AHB Channels 659
6.1-2 Modules that Share GDMA-AXI Channels 660
6.3-1 GDMA controller Architecture 661
6.4-1 Structure of a Linked List 662
6.4-2 Relationship among Linked Lists 665
6.4-3 CRC Calculation Matrix 670

7.4-1 VDMA Architecture 776
7.5-1 VDMA Transfer Hierarchy for Peripherals 778
7.5-2 VDMA Transfer Hierarchy for Memory 778
7.5-3 VDMA Read Arbiter 779
7.5-4 VDMA Write Arbiter 780
7.5-5 VDMA Linked List Item (Descriptor) 784
7.5-6 CH*n*_LLP_STATUS Write-Back Field of LLI 785

8.3-1 2D-DMA Architecture 829
8.4-1 Structure of a Linked List 830
8.4-2 Padding Illustration (Overall) 834
8.4-3 Padding Illustration (Detail) 834
8.4-4 Macroblock Layout of JPEG Decoder Output Data 837
8.4-5 Color Space Conversion 838

9.3-1 System Structure and Address Mapping 904
9.3-2 HP L2MEM Address Layout 907
9.3-3 Cache Structure 910
9.4-1 Data Flow in eFuse 917
9.4-2 Shift Register Circuit (first 32 bytes) 927

9.4-3 Shift Register Circuit (last 12 bytes) 927

10.3-1 Architecture of HP GPIO Matrix, HP IO MUX, LP GPIO Matrix, and LP IO MUX 983
10.3-2 Internal Structure of a Pad 984
10.4-1 GPIO Input Synchronized on Rising Edge or on Falling Edge of HP IO MUX Operating Clock 986
10.4-2 GPIO Filter Timing of GPIO Input Signals 986
10.4-3 Glitch Filter Timing Example 989
10.10-1 Example of Level Flip on the Chip Pad — Hysteresis Function Not Enabled 997
10.10-2 Example of Level Flip on the Chip Pad — Hysteresis Function Enabled 998

11.1-1 Reset Types 1081
11.2-1 HP and LP System Clock 1085
11.3-1 Clock Configuration Example 1094

12.2-1 Chip Boot Flow 1221

13.2-1 Interrupt Flow in ESP32-P4 1225
13.3-1 Interrupt Matrix Structure 1226

14.3-1 ETM Channel*n* Architecture 1269
14.3-2 Event Task Matrix Clock Architecture 1280

15.4-1 ESP32-P4 Power Scheme 1427
15.4-2 PMU Workflow 1430
15.4-3 VO1 Mode Control 1438
15.4-4 VO2/3/4 Mode Control 1439

16.3-1 System Timer Structure 1508
16.5-1 System Timer Alarm Generation 1509

17.1-1 Timer Group Overview 1532
17.3-1 Timer Group Architecture 1533

18.1-1 Watchdog Timers Overview 1558
18.2-1 Digital Watchdog Timers in ESP32-P4 1559
18.3-1 Super Watchdog Controller Structure 1562

20.1-1 PMP-APM Management Relation 1587
20.3-1 APM Architecture 1589

23.4-1 Architecture of AXI Perf Mon 1855

24.3-1 LP Mailbox Block Diagram 1895

25.3-1 Brown-out Detector Overview 1912
25.4-1 Structure of Brown-out Detector 1914

26.3-1 Structure of the ESP32-P4 Power Supply Detector 1926

27.7-1 GCM Encryption Process 1938

29.3-1 HMAC SHA-256 Padding Diagram 1969

29.3-2 HMAC Structure Schematic Diagram 1970

32.3-1 Software Preparations and Hardware Working Process 2010

33.5-1 ECDSA_DS Process 2025

34.3-1 Architecture of the External Memory Encryption and Decryption 2039

35.3-1 Noise Source 2052

36.5-1 Architecture of Key Manager and HUK Generator 2059
36.6-1 AES Deploy Mode 2063
36.6-2 ECDH0 Deploy Mode 2065
36.6-3 ECDH1 Deploy Mode 2066

37.1-1 JPEG Bitstream Format 2096
37.2-1 JPEG codec system connection block diagram 2097
37.4-1 JPEG encoder architecture 2098
37.4-2 JPEG decoder architecture 2099

38.1-1 ESP32-P4 ISP SoC Architecture 2140
38.4-1 ISP IP Architecture 2142
38.5-1 Gamma Curve 2147
38.5-2 Sub-Window Numbering 2148
38.5-3 AWB window 2150
38.5-4 ESP32-P4 CSI CM Architecture 2152
38.5-5 Byte Ordering 2153

39.1-1 PPA 2D-DMA Connection 2255
39.4-1 PPA Architecture 2256
39.5-1 PPA SRM Input Pixel Processing 2258
39.5-2 PPA SRM Alpha Configuration Logic 2259
39.5-3 2D-DMA Linked List 2261
39.5-4 SRM Procedure 2264

40.3-1 LCD_CAM Block Diagram 2301
40.3-2 LCD Clock 2303
40.3-3 Camera Clock 2304
40.3-4 LCD Frame Structure 2312
40.3-5 LCD Timing (RGB Format) 2313
40.3-6 LCD Timing (I8080/MOTO6800 Format) 2313

41.4-1 H264 Video Encoder Architecture 2338
41.5-1 ENC_CORE Architecture 2339
41.5-2 H264_DMA Architecture 2346
41.5-3 H264_DMA Linked List Descriptor 2348
41.7-1 Coding Order of Pictures in GOP Mode 2355
41.7-2 Coding Order of Pictures in Dual Stream Mode 2358

42.4-1 MIPI CSI Architecture 2594

42.5-1 Level of MIPI Data Lane Lines 2596
42.5-2 MIPI RX D-PHY Operating States and Modes 2597
42.5-3 MIPI CSI HS Data Reception Sequence 2600
42.5-4 ULPS Sequences for Clock and Data Lanes 2601
42.5-5 Control Interface Timing Diagram 2602
42.6-1 MIPI CSI Interrupt Mechanism 2606

43.3-1 MIPI DSI architecture 2634
43.4-1 Command Transmission Periods 2638
43.4-2 outvact_lpcmd_time for Non-Burst with Sync Pulse 2639
43.4-3 outvact_lpcmd_time for Burst or Non-Burst with Sync Event 2639
43.4-4 invact_lpcmd_time for Non-Burst with Sync Pulse 2639
43.4-5 invact_lpcmd_time for Burst or Non-Burst with Sync Event 2639
43.4-6 Vertical Color Bar 2640
43.4-7 Horizontal Color Bar 2641
43.4-8 RGB888 BER Testing Pattern 2642
43.4-9 PHY operating modes 2643
43.4-10 MIPI DSI HS Data Reception Sequence 2646
43.4-11 MIPI DSI clock architectural 2656
43.4-12 Video Timing 2657

44.3-1 ESP32-P4 VAD Architecture 2724

45.3-1 UART Structure 2734
45.4-1 UART Controllers Division 2737
45.4-2 The Timing Diagram of Weak UART Signals Along Negative Edges 2737
45.4-3 Structure of UART Data Frame 2738
45.4-4 AT_CMD Character Structure 2739
45.4-5 Driver Control Diagram in RS485 Mode 2740
45.4-6 The Timing Diagram of Encoding and Decoding in SIR mode 2741
45.4-7 IrDA Encoding and Decoding Diagram 2741
45.4-8 Hardware Flow Control Diagram 2743
45.4-9 Connection between Hardware Flow Control Signals 2744
45.4-10 Data Transfer in GDMA Mode 2745
45.5-1 UART Programming Procedures 2748

46.4-1 SPI Module Overview 2823
46.5-1 Data Buffer Used in CPU-Controlled Transfer 2830
46.5-2 GP-SPI Functional Blocks 2835
46.5-3 Data Flow Control in GP-SPI as Master 2836
46.5-4 Data Flow Control in GP-SPI as Slave 2837
46.5-5 GP-SPI State Machine as Master 2839
46.5-6 Full-Duplex Communication Between GP-SPI2 Master and a Slave 2844
46.5-7 Connection of GP-SPI2 to Flash and External RAM in 4-bit Mode 2847
46.5-8 SPI Quad I/O Read Command Sequence Sent by GP-SPI2 to Flash 2847
46.5-9 Configurable Segmented Transfer as Master 2848
46.6-1 Recommended CS Timing and Settings When Accessing External RAM 2857

46.6-2 Recommended CS Timing and Settings When Accessing Flash 2858
46.7-1 SPI Clock Mode 0 or 2 2859
46.7-2 SPI Clock Mode 1 or 3 2860
46.8-1 Timing Compensation Control Diagram in GP-SPI as Master 2862
46.8-2 Timing Compensation Example in GP-SPI2 as Master 2863

47.3-1 I2C Master Architecture 2980
47.3-2 I2C Slave Architecture 2980
47.3-3 I2C Protocol Timing (Cited from Fig.31 in The I2C-bus specification Version 2.1) 2981
47.4-1 I2C Timing Diagram 2986
47.4-2 Structure of I2C Command Registers 2988
47.6-1 $I2C_{master}$ Writing to $I2C_{slave}$ with a 7-bit Address 2993
47.6-2 $I2C_{master}$ Writing to a Slave with a 10-bit Address 2995
47.6-3 $I2C_{master}$ Writing to $I2C_{slave}$ with Two 7-bit Addresses 2996
47.6-4 $I2C_{master}$ Writing to $I2C_{slave}$ with a 7-bit Address in Multiple Sequences 2998
47.6-5 $I2C_{master}$ Reading $I2C_{slave}$ with a 7-bit Address 3000
47.6-6 $I2C_{master}$ Reading $I2C_{slave}$ with a 10-bit Address 3003
47.6-7 $I2C_{master}$ Reading N Bytes of Data from addrM of $I2C_{slave}$ with Two 7-bit Addresses 3005
47.6-8 $I2C_{master}$ Reading $I2C_{slave}$ with a 7-bit Address in Segments 3008

48.3-1 Analog I2C Controller Architecture 3063
48.4-1 Signal Phase 3064

49.4-1 ESP32-P4 I2S System Diagram 3073
49.5-1 TDM Philips Standard Timing Diagram 3075
49.5-2 TDM MSB Alignment Standard Timing Diagram 3076
49.5-3 TDM PCM Standard Timing Diagram 3076
49.5-4 PDM Standard Timing Diagram 3077
49.6-1 I2S*n* Clock Generator 3077
49.9-1 TX Data Format Control 3083
49.9-2 TDM Channel Control 3084
49.9-3 PDM Channel Control Example 3087

50.3-1 ESP32-P4 LP I2S Architecture 3117
50.4-1 TDM Philips Standard Timing Diagram 3119
50.4-2 TDM MSB Alignment Standard Timing Diagram 3119
50.4-3 TDM PCM Standard Timing Diagram 3120
50.4-4 PDM Standard Timing Diagram 3120
50.5-1 LP I2S Clock Generator 3121

51.1-1 PCNT Overview 3142
51.3-1 PCNT Architectural Overview 3143
51.6-1 Channel 0 Up Counting Diagram 3146
51.6-2 Channel 0 Down Counting Diagram 3147
51.6-3 Two Channels Up Counting Diagram 3147

52.4-1 OTG_HS System Architecture 3160
52.4-2 OTG_HS Register Layout 3161

52.4-3 Host Mode FIFOs 3163
52.4-4 Device Mode FIFOs 3164
52.4-5 OTG_HS Interrupt Hierarchy 3165
52.4-6 Scatter/Gather DMA Descriptor List 3166

53.4-1 OTG_FS System Architecture 3172
53.4-2 OTG_FS Register Layout 3173
53.4-3 Host Mode FIFOs 3175
53.4-4 Device Mode FIFOs 3176
53.4-5 OTG_FS Interrupt Hierarchy 3177
53.4-6 Scatter/Gather DMA Descriptor List 3178
53.5-1 A-Device SRP 3182
53.5-2 B-Device SRP 3183
53.5-3 A-Device HNP 3184
53.5-4 B-Device HNP 3185

54.2-1 USB Serial/JTAG High Level Diagram 3188
54.2-2 USB Serial/JTAG Block Diagram 3189

55.1-1 Ethernet MAC Functionality Overview 3223
55.3-1 Ethernet MAC Block Diagram 3225
55.4-1 Transmit Descriptor Linked List 3229
55.4-2 Receive Descriptor Linked List 3234
55.4-3 MII Interface 3241
55.4-4 MII Clock 3242
55.4-5 RMII Interface 3243
55.4-6 RMII Clock 3244

56.2-1 Bit Fields in Data Frames and Remote Frames 3307
56.2-2 Fields of an Error Frame 3309
56.2-3 Fields of an Overload Frame 3309
56.2-4 The Fields within an Interframe Space (*Error Passive) 3310
56.2-5 Layout of a Bit 3313
56.3-1 TWAI Overview Diagram 3314
56.4-1 Acceptance Filter 3322
56.4-2 Single-Filter Mode 3323
56.4-3 Dual-Filter Mode 3324
56.4-4 Error State Transition 3325
56.4-5 Positions of Arbitration Lost Bits 3327

57.2-1 SD/MMC Controller Topology 3352
57.3-1 SD/MMC Controller External Interface Signals 3353
57.4-1 SDIO Host Block Diagram 3354
57.4-2 Command Path State Machine 3355
57.4-3 Data Transmit State Machine 3356
57.4-4 Data Receive State Machine 3356
57.7-1 Linked List Structure 3359
57.8-1 Descriptor Format 3359

58.3-1 LED PWM Architecture 3398
58.3-2 Timer and PWM Generator Block Diagram 3398
58.4-1 Frequency Division When LEDC_CLK_DIV is a Non-Integer Value 3400
58.4-2 Relationship Between Counter And Resolution 3400
58.4-3 LED PWM Output Signal Diagram 3402
58.4-4 Output Signal of Linear Duty Cycle Fading 3403
58.4-5 Output Signal of Gamma Curve Fading 3404

59.2-1 MCPWM Module Overview 3424
59.3-1 Timer Module 3426
59.3-2 Operator Module 3426
59.3-3 Capture Module 3427
59.3-4 ETM Module 3427
59.3-5 Count-Up Mode Waveform 3429
59.3-6 Count-Down Mode Waveforms 3429
59.3-7 Count-Up-Down Mode Waveforms, Count-Down at Synchronization Event 3429
59.3-8 Count-Up-Down Mode Waveforms, Count-Up at Synchronization Event 3430
59.3-9 UTEP and UTEZ Generation in Count-Up Mode 3431
59.3-10 DTEP and DTEZ Generation in Count-Down Mode 3432
59.3-11 DTEP and UTEZ Generation in Count-Up-Down Mode 3432
59.3-12 Block Diagram of A PWM Operator 3435
59.3-13 Symmetrical Waveform in Count-Up-Down Mode 3439
59.3-14 Count-Up, Single Edge Asymmetric Waveform, with Independent Modulation on PWM*x*A and PWM*x*B — Active High 3440
59.3-15 Count-Up, Pulse Placement Asymmetric Waveform with Independent Modulation on PWM*x*A 3441
59.3-16 Count-Up-Down, Dual Edge Symmetric Waveform, with Independent Modulation on PWM*x*A and PWM*x*B — Active High 3442
59.3-17 Count-Up-Down, Dual Edge Symmetric Waveform, with Independent Modulation on PWM*x*A and PWM*x*B — Complementary 3443
59.3-18 Count-Up-Down, Fault or Synchronization Events, with Same Modulation on PWM*x*A and PWM*x*B 3444
59.3-19 Example of an NCI Software-Force Event on PWM*x*A 3446
59.3-20 Example of a CNTU Software-Force Event on PWM*x*B 3447
59.3-21 Options for Setting up the Dead Time Generator Module 3449
59.3-22 Active High Complementary (AHC) Dead Time Waveforms 3450
59.3-23 Active Low Complementary (ALC) Dead Time Waveforms 3451
59.3-24 Active High (AH) Dead Time Waveforms 3451
59.3-25 Active Low (AL) Dead Time Waveforms 3452
59.3-26 Example of Waveforms Showing PWM Carrier Action 3453
59.3-27 Example of the First Pulse and the Subsequent Sustaining Pulses of the PWM Carrier Submodule 3454
59.3-28 Possible Duty Cycle Settings for Sustaining Pulses in the PWM Carrier Submodule 3455

60.3-1 RMT Architecture 3511
60.3-2 Format of Pulse Code in RAM 3512

61.4-1 PARLIO Architecture 3540
61.5-1 PARLIO Clock Generation 3541

61.5-2 Positive Waveform 3543
61.5-3 Negative Waveform 3543
61.5-4 Level Enable Mode for RX Unit 3544
61.5-5 Sub-Modes of Pulse Enable Mode for RX Unit 3545
61.5-6 Software Enable Mode for RX Unit 3545

62.1-1 BitScrambler System Context Diagram 3567
62.3-1 BitScrambler Data Path Diagram 3568
62.3-2 BitScrambler Control Path Diagram 3569

63.3-1 Touch Sensor System 3594
63.3-2 Touch Sensor Internal Structure 3596
63.4-1 Touch FSM Structure and Mechanism 3597
63.4-2 TOUCH_OUT Signal Preprocessing 3598
63.4-3 Timing Diagram of Scan Mode 3601
63.4-4 Touch Sensor Frequency Hopping Sequence Timing 3602
63.4-5 Sensing Area 3605

64.3-1 Temperature Sensor Architecture 3636

65.1-1 SAR ADC Simplified Architecture 3646
65.4-1 SAR ADC Architecture 3648
65.5-1 HP ADC FSM Block Diagram 3651
65.5-2 ADC_SAR*x*_PATT_TAB1_REG Contains Patterns 0 - 3 3652
65.5-3 ADC_SAR*x*_PATT_TAB2_REG Contains Patterns 4 - 7 3652
65.5-4 ADC_SAR*x*_PATT_TAB3_REG Contains Patterns 8 - 11 3652
65.5-5 ADC_SAR*x*_PATT_TAB4_REG Contains Patterns 12 - 15 3652
65.5-6 Pattern Structure 3653
65.5-7 cmd0 configuration 3653
65.5-8 cmd1 Configuration 3653
65.5-9 GDMA Data Format 3655

66.3-1 Analog Voltage Comparator Architecture 3689

Part I

Microprocessor and Master

This part covers the essential processing elements of the system, diving into the microprocessor architecture with both high-performance and low-power CPUs. Details include CPU instruction extensions and trace encoding for debugging purposes.

Chapter 1

Bus Architecture

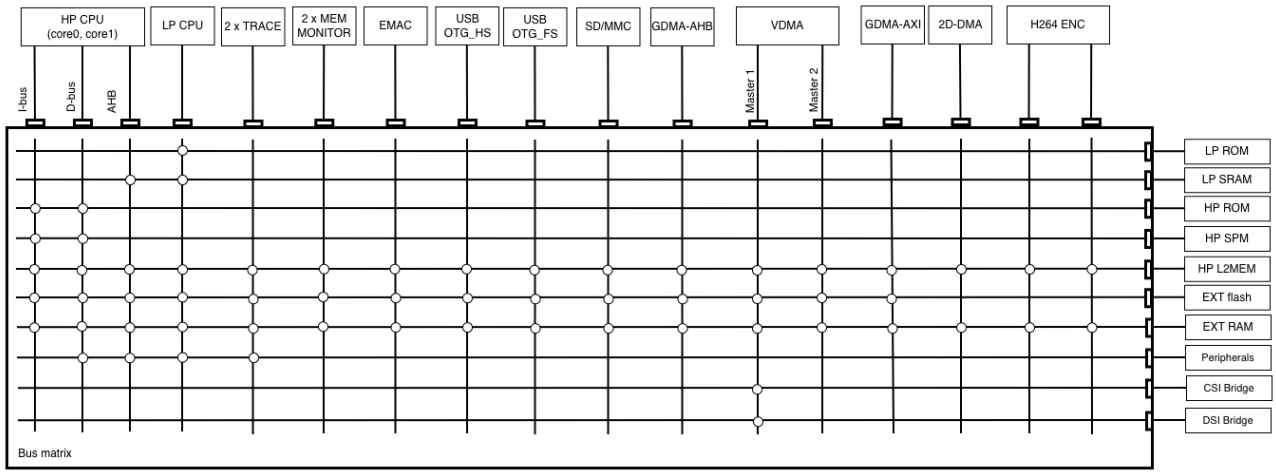
The ESP32-P4 SoC uses a multi-layer bus matrix as the core of its communication architecture, connecting multiple masters and slaves efficiently. This architecture supports concurrent access, effectively mitigating conflicts so that the high-performance and low-power RISC-V cores, DMA controllers, debug interfaces, and various peripherals can operate simultaneously with minimal contention. By optimizing data flow and system performance, the bus matrix ensures efficient system operation even when multiple high-speed peripherals are running at the same time.

The main system of the ESP32-P4 SoC consists of:

- Masters:
  - I-bus, D-bus and AHB bus of the high-performance (HP) 32-bit RISC-V processor 0
  - I-bus, D-bus and AHB bus of the high-performance (HP) 32-bit RISC-V processor 1
  - Low-power (LP) 32-bit RISC-V processor
  - TRACE0 (Trace Interface 0)
  - TRACE1 (Trace Interface 1)
  - HP L2MEM MONITOR (HP L2 Memory Monitor)
  - HP SPM MONITOR (HP Scratchpad Memory Monitor)
  - EMAC
  - USB OTG_HS (USB 2.0 High-Speed OTG)
  - USB OTG_FS (USB 2.0 Full-Speed OTG)
  - SD/MMC
  - GDMA-AHB
  - VDMA
  - GDMA-AXI
  - 2D-DMA
  - H264 ENC
- Slaves:
  - LP ROM
  - LP SRAM
  - HP ROM
  - HP SPM

- HP L2MEM
- External flash
- External RAM
- Peripherals (LP, HP and HP CPU peripherals)
- CSI Bridge
- DSI Bridge

The bus architecture of the SoC is shown below.



Note:

1. Circles at intersections indicate that the bus (column) can access the corresponding slave device (row).
2. Peripherals in the diagram include LP, HP, and HP CPU peripherals.
3. CPU in the diagram represents both high-performance RISC-V processor 0 and processor 1, with each core having independent I-bus, D-bus, and AHB bus interfaces.
4. The 2x MEM MONITOR in the diagram refers to HP L2MEM MONITOR and HP SPM MONITOR.

Figure 1.0-1. ESP32-P4 Bus Architecture

Below is the description of each bus shown in Figure 1.0-1 *ESP32-P4 Bus Architecture*:

- **HP CPU I-bus**: Connects the instruction bus of the high-performance (HP) 32-bit RISC-V processor to the bus matrix for instruction fetch. It can directly access the HP SPM or indirectly access other code-containing memories through the cache, including HP ROM, HP L2MEM, as well as external flash and external RAM. In most cases, the cache is transparent to users.
- **HP CPU D-bus**: Connects the data bus of the high-performance (HP) 32-bit RISC-V processor to the bus matrix for literal loading and debug access. It can directly access the HP SPM or access other code- or data-containing memories (such as HP ROM, external flash, and external RAM) through the cache. It can also directly access peripheral registers, but only for HP CPU peripherals. In most cases, the cache is transparent to users.
- **HP CPU AHB bus**: Connects the AHB bus of the high-performance (HP) 32-bit RISC-V processor to the bus matrix for accessing data in peripherals or memory. Through the AHB interface, both LP and HP peripherals can be accessed, and direct access to HP L2MEM, HP ROM, LP SRAM, and external flash

and external RAM can be configured via address mapping (refer to Chapter 9 *System and Memory* for detailed configuration of address mapping).

- **LP CPU bus**: Connects the low-power (LP) 32-bit RISC-V processor to the bus matrix, allowing the LP CPU to access peripherals or memory. Accessible targets include LP SRAM, LP ROM, HP L2MEM, HP ROM, external flash, external RAM, as well as LP, HP, and HP CPU peripherals.
- **TRACE bus**: Connects TRACE0 and TRACE1 modules to the bus matrix for debugging operations. Accessible targets include HP L2MEM and external RAM.
- **MEM MONITOR bus**: Connects HP L2MEM MONITOR and HP SPM MONITOR to the bus matrix for writing monitoring data from their respective monitored memory regions to memory. The two MEM MONITOR interfaces have the same access path. Accessible targets include HP L2MEM and external RAM.
- **EMAC bus**: Connects the Ethernet MAC controller to the bus matrix for high-speed network data transfer. Accessible targets include HP L2MEM and external RAM.
- **USB OTG_HS bus**: Connects the USB 2.0 High-Speed OTG controller to the bus matrix for USB 2.0 High-Speed data transfer. Accessible targets include HP L2MEM and external RAM.
- **USB OTG_FS bus**: Connects the USB 2.0 Full-Speed OTG controller to the bus matrix for USB 2.0 Full-Speed data transfer. Accessible targets include HP L2MEM and external RAM.
- **SD/MMC bus**: Connects the SD/MMC controller to the bus matrix for SD/MMC card data read and write operations. Accessible targets include HP L2MEM and external RAM, supporting high-speed data transfer through DMA.
- **GDMA bus**: The chip contains two types of GDMA controllers and one VDMA controller for high-speed data transfer.
  - **GDMA-AHB**: DMA controller based on AHB bus protocol
  - **GDMA-AXI**: DMA controller based on the AXI bus protocol
  - **VDMA**: Video DMA controller based on the AXI bus protocol

  Each DMA provides one connection to the bus matrix and one connection to external memory. GDMAs perform high-speed data transfers through these buses, including peripheral-to-memory, memory-to-peripheral, peripheral-to-peripheral, and memory-to-memory transfers. Accessible targets include HP L2MEM, and external flash and external RAM. For DMA accesses to external memory, it is recommended to configure longer burst lengths whenever possible to ensure optimal data throughput.
- **2D-DMA bus**: Connects the DMA2D graphics accelerator to the bus matrix for 2D graphics data transfer and processing. Accessible targets include HP L2MEM and external RAM.
- **H264 ENC bus**: Connects the H.264 video encoder to the bus matrix, containing two AXI interfaces with the same access path. Used for data access during video encoding. Accessible targets include HP L2MEM and external RAM.

Chapter 2

High-Performance CPU

2.1 Overview

ESP32-P4 is powered by Espressif's second-generation 32-bit CPU core complex based upon RISC-V Instruction Set Architecture (ISA). The CPU Core complex consists of dual RISC-V CPU cores with a dedicated core-local interrupt controller (CLIC), a debug block, and a core-local interrupt (CLINT) timer. Figure 2.1 shows the block diagram of the CPU core complex.

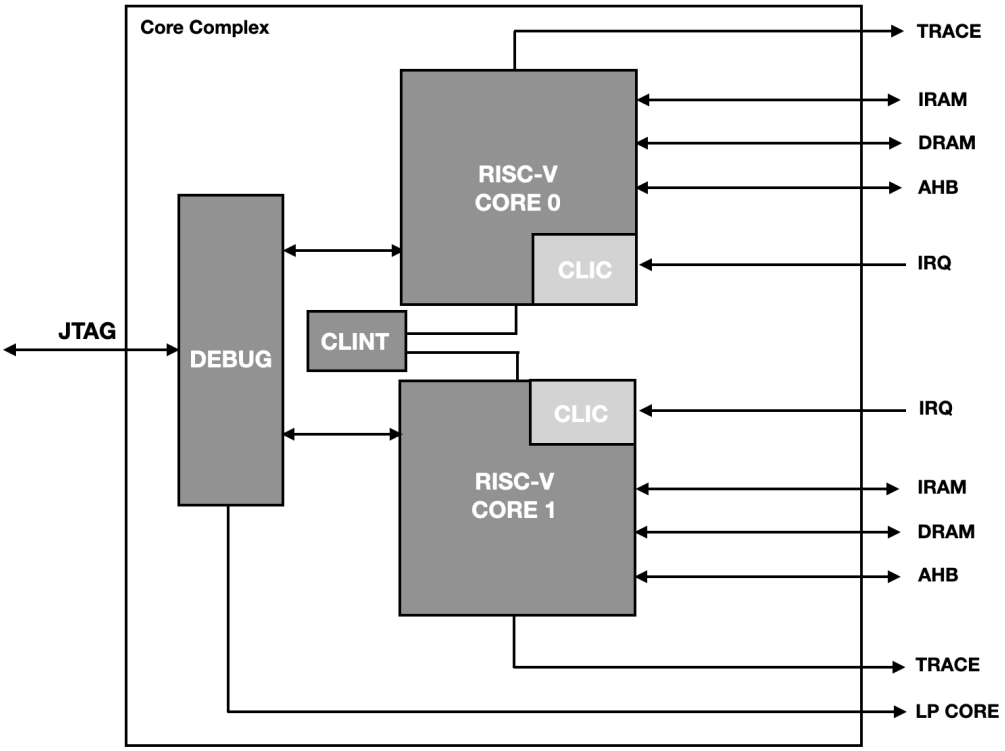


Figure 2.1-1. CPU Core Complex Block Diagram

Each RISC-V core in CPU core complex uses a 5-stage, in-order, scalar pipeline architecture optimized for area, power, and performance. Each RISC-V core has a built-in dedicated core-local interrupt controller (CLIC) and system bus (SYS BUS) interfaces for memory and peripheral access. The CLINT timer is shared between two cores. The DEBUG block provides a connection between the JTAG interface and the dual cores. It also supports LP core debug. Each core also provides an instruction trace interface for offline debugging.

2.2 Features

Each RISC-V CPU core in the CPU core complex implements the following standard RISC-V extensions:

- Mandatory base integer (I)
- Multiplication/Division (M)
- Atomic (A)
- Floating (F)
- Compressed (C)
- Code size reduction (Zc)
- Bit Manipulation (Zb)

In addition to the above standard extensions, each RISC-V CPU core also implements custom instructions (X) which can be broadly classified in the following two categories:

- Custom instructions to accelerate AI algorithms performance
- Custom hardware loop instructions to further improve performance and reduce code size for iterative AI algorithms execution

Apart from the above RISC-V instruction extensions support, each CPU core supports the features below:

- RISC-V RV32IMAFCX ISA with a five-stage pipeline that supports an operating clock frequency up to 360 MHz
- Compatible with RISC-V Instruction Set Manual, Volume I: RISC-V User-Level ISA, Version 2.2
- Compatible with RISC-V Instruction Set Manual, Volume II: RISC-V Privileged Architecture, Version 1.10
- Compatible with Core-Local Interrupt Controller (CLIC) RISC-V Privileged Architecture Extensions (10/10/2023)
- Compatible with RISC-V External Debug Support Version 0.13.2
- Zero wait cycle access to on-chip SRAM and cache for program and data access over IRAM/DRAM interface
- 32-bit AHB system bus for peripheral access
- Core-local interrupts (CLINT) dedicated for each privilege mode
- User (U) privilege mode execution
- Interrupt delegation to user mode
- Bus error response
- Standard physical memory protection (PMP) configurable up to 32 regions and custom attributes (PMA) configurable up to 16 regions for security
- Branch target buffer (BTB) with BHT and RAS for performance improvement
- Debug module (DM) compliant with the specification RISC-V External Debug Support Version 0.13.2 with external debugger support over an industry-standard JTAG/USB port

- Debugger with a direct system bus access (SBA) to memory and peripherals via Core 0 data bus
- Support for instruction trace for offline debug
- Hardware trigger compliant with the specification RISC-V External Debug Support Version 0.13.2 with up to 3 breakpoints/watchpoints
- Fast GPIO support (up to 8)
- Configurable events for core performance metrics

2.3 Terminology

To better illustrate the functionality of the High-Performance CPU, the following terms are used in this chapter.

ABI	Application binary interface
branch	An instruction that conditionally changes the execution flow
CLIC	Core-local interrupt controller
CLINT	Core-local interrupt
CSR	Control and status register
delta	A change in the program counter that is other than the difference between two instructions placed consecutively in memory
DTM	Debug transport module
FPR	Floating-point register
GPIO	General-purpose input/output
GPR	General-purpose register
HWLP	Hardware loop
hart	RISC-V hardware thread
LSB	Least significant bit
LR	Load reserved
MSB	Most significant bit
PMP	Physical memory protection
retire	The final stage of executing an instruction, when the machine state is updated
RO	Read-only field
R/W	Readable and writable field
SC	Store conditional
SoC	System on chip
WO	Write-only field
WARL	Write any read legal
W1	Write One

2.4 Address Map

The table below shows the address map of various regions accessible by CPU for instruction, data, and system bus peripherals.

Table 2.4-1. CPU Address Map

Region	Description	Start Address	End Address	Access Type
CPU	CLIC/CLINT registers	0x2000_0000	0x2FFF_FFFF	R/W
IRAM/DRAM	Instruction/Data region	0x3000_0000	0x3FEF_FFFF	R/W
CPU peripherals		0x3FF0_0000	0x3FF1_FFFF	R/W
IRAM/DRAM	Instruction/Data region	0x3FF2_0000	0x4FFF_FFFF	R/W
CPU peripherals	HP APB peripherals	0x5000_0000	0x500F_FFFF	R/W
LP RAM/ROM		0x5010_0000	0x5010_FFFF	R/W
LP APB peripherals		0x5011_0000	0x5012_FFFF	R/W
AHB	AHB peripheral region	*Default	*Default	R/W

***Default:** Addresses not matching any of the specified ranges above are accessed through AHB bus by a CPU core.

2.5 Configuration and Status Registers (CSRs)

2.5.1 Register Summary

Below is a list of CSRs implemented for each HP CPU core. All the implemented CSRs follow the standard mapping of bit fields as described in the RISC-V Instruction Set Manual, Volume II: Privileged Architecture, Version 1.10. It must be noted that even among the standard CSRs, not all bit fields have been implemented, limited by the subset of features implemented in the CPU. Refer to the next section for detailed description of the subset of fields implemented under each of these CSRs.

Name	Description	Address	Access
Machine Information CSRs			
mvendorid	Machine vendor ID	0xF11	RO
marchid	Machine architecture ID	0xF12	RO
mimpid	Machine implementation ID	0xF13	RO
mhartid	Machine hart ID	0xF14	RO
Machine Trap Setup CSRs			
mstatus	Machine mode status	0x300	R/W
misa [1]	Machine ISA	0x301	R/W
mideleg	Machine interrupt delegation register (INACTIVE IN CLIC MODE)	0x303	RO
mie	Machine interrupt enable register (INACTIVE IN CLIC MODE)	0x304	RO
mtvec [2]	Machine trap vector	0x305	R/W
mcounteren	Machine counter enable	0x306	R/W
mtvt	Machine vector interrupt base address (Refer to CLIC specifications)	0x307	R/W

[1] Although misa is specified as having both read and write access (R/W), its fields are hardwired and thus write has no effect. This is what would be termed WARL (Write Any Read Legal) in RISC-V terminology

[2] mtvec Holds trap vector configuration consisting of vector base address and a vector mode

Name	Description	Address	Access
Machine Trap Handling CSRs			
mscratch	Machine scratch	0x340	R/W
mepc	Machine trap program counter	0x341	R/W
mcause [3]	Machine trap cause	0x342	R/W
mtval	Machine trap value	0x343	R/W
mip	Machine interrupt pending (INACTIVE IN CLIC MODE)	0x344	RO
mnxti	Interrupt handler address and enable modifier (Refer to CLIC specifications)	0x345	R/W
mintstatus	Current interrupt levels (Refer to CLIC specifications)	0xFB1	RO
mintthresh	Interrupt threshold (Refer to CLIC specifications)	0x347	R/W
mscratchcsw	Conditional scratch swap on priv mode change (Refer to CLIC specifications)	0x348	R/W
mscratchcswl	Conditional scratch swap on level change (Refer to CLIC specifications)	0x349	R/W
mclicbase	Machine mode interrupt controller base address register (CUSTOM) (Refer to CLIC specifications)	0x350	RO
User Trap Setup CSRs			
ustatus	User mode status	0x000	R/W
utvec [4]	User trap vector	0x005	R/W
utvt	User vector interrupt base address (Refer to CLIC specifications)	0x007	R/W
User Trap Handling CSRs			
uscratch	User scratch	0x040	R/W
uepc	User trap program counter	0x041	R/W
ucause [5]	User trap cause	0x042	R/W
unxti	Interrupt handler address and enable modifier (Refer to CLIC specifications)	0x045	R/W
uintthresh	Interrupt threshold (Refer to CLIC specifications)	0x047	R/W
uclicbase	User mode interrupt controller base address (CUSTOM) (Refer to CLIC specifications)	0x050	RO
uintstatus	Current interrupt levels (Refer to CLIC specifications)	0xCB1	RO
Physical Memory Protection (PMP) CSRs			
pmpcfg0	Physical memory protection configuration	0x3A0	R/W
pmpcfg1	Physical memory protection configuration	0x3A1	R/W
pmpcfg2	Physical memory protection configuration	0x3A2	R/W
pmpcfg3	Physical memory protection configuration	0x3A3	R/W
pmpcfg4	Physical memory protection configuration	0x3A4	R/W
pmpcfg5	Physical memory protection configuration	0x3A5	R/W
pmpcfg6	Physical memory protection configuration	0x3A6	R/W
pmpcfg7	Physical memory protection configuration	0x3A7	R/W

[3] External interrupt IDs reflected in mcause include even those IDs which have been reserved by RISC-V standard for core internal sources.
[4] utvec holds trap vector configurations, including the vector base address and the vector mode
[5] External interrupt IDs reflected in mcause include even those IDs that have been reserved by the RISC-V standard for core internal sources.

Name	Description	Address	Access
pmpaddr*n* (*n*: 0-31)	Physical memory protection address register	0x3B0+0x1**n*	R/W
Trigger Module CSRs (shared with Debug Mode)			
tselect	Trigger select register	0x7A0	R/W
tdata1	Trigger abstract data 1	0x7A1	R/W
tdata2	Trigger abstract data 2	0x7A2	R/W
tdata3	Trigger abstract data 3	0x7A3	R/W
tinfo	Trigger information	0x7A4	R/W
tcontrol	Global trigger control	0x7A5	R/W
mcontext	Machine mode context register	0x7A8	R/W
Debug Mode CSRs (Accessible only in Debug Mode)			
dcsr	Debug control and status	0x7B0	R/W
dpc	Debug PC	0x7B1	R/W
dscratch0	Debug scratch register 0	0x7B2	R/W
dscratch1	Debug scratch register 1	0x7B3	R/W
dmcs2	Debug mode control status register 2	0x32	R/W
Machine Counter Setup			
mcountinhibit	Machine counter inhibit register	0x320	R/W
mhpmevent8	Machine performance-monitoring event selector	0x308	R/W
mhpmevent9	Machine performance-monitoring event selector	0x329	R/W
mhpmevent13	Machine performance-monitoring event selector	0x32D	R/W
Machine Counter/Timers			
mcycle	Machine cycle counter	0xB00	R/W
minstret	Machine instructions-retired counter	0xB02	R/W
mhpmcounter8	Machine performance-monitoring counter	0xB08	R/W
mhpmcounter9	Machine performance-monitoring counter	0xB09	R/W
mhpmcounter13	Machine performance-monitoring counter	0xB0D	R/W
mcycleh	Upper 32 bits of mcycle	0xB80	R/W
minstreth	Upper 32 bits of minstret	0xB82	R/W
mhpmcounter8h	Upper 32 bits of mhpmcounter8	0xB88	R/W
mhpmcounter9h	Upper 32 bits of mhpmcounter9	0xB89	R/W
mhpmcounter13h	Upper 32 bits of mhpmcounter13	0xB8D	R/W
Unprivileged/User Mode Floating-Point CSRs			
fflags	Floating-point accrued exceptions	0x001	R/W
frm	Floating-point dynamic rounding mode	0x002	R/W
fcsr	Floating-point control and status register (frm + fflags)	0x003	R/W
Unprivileged/User Mode Counter/Timers			
cycle	Cycle Counter for RDCYCLE instruction	0xC00	RO
time	Timer for RDTIME instruction	0xC01	RO
instret	Instructions-retired counter for RDINSTRET instruction	0xC02	RO
hpmcounter8	Performance-monitoring counter	0xC08	RO
hpmcounter9	Performance-monitoring counter	0xC09	RO
hpmcounter13	Performance-monitoring counter	0xC0D	RO
cycleh	Upper 32 bits of cycle	0xC80	RO

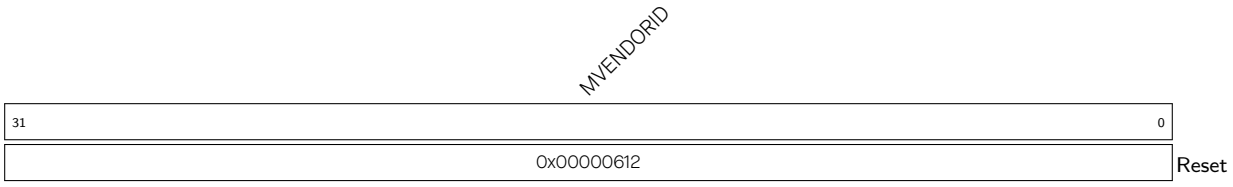
Name	Description	Address	Access
instreth	Upper 32 bits of instret	0xC82	RO
hpmcounter8h	Upper 32 bits of hpmcounter8	0xC88	RO
hpmcounter9h	Upper 32 bits of hpmcounter9	0xC89	RO
hpmcounter13h	Upper 32 bits of hpmcounter13	0xC8D	RO
Custom System CSRs (Custom)			
Floating-Point User Mode CSR (Custom)			
fxcr	Floating-point extended control register	0x800	R/W
GPIO Access CSRs (Custom)			
cpu_gpio_oen	GPIO output enable	0x803	R/W
cpu_gpio_in	GPIO input value	0x804	RO
cpu_gpio_out	GPIO output value	0x805	R/W
Extension Control and Status CSRs (Custom)			
mext_ill_reg	Machine extension illegal register	0x7F0	R/W
mhwloop_state_reg	Machine mode HWLP extension state register	0x7F1	R/W
mext_pie_status	Machine mode PIE extension status register	0x7F2	R/W
Zc Extension CSRs			
jvt	Machine table jump base vector and control register	0x017	R/W
Machine Extension Status CSRs (Custom)			
mexstatus	Machine extension control and status register	0x7E1	R/W
mhint	Machine implicit operation register	0x7C5	R/W
Physical Memory Attributes Checker (PMAC) CSRs (Custom)			
pma_cfg*n* (*n*: 0-15)	Physical memory attribute configuration register	0xBC0+0x1**n*	R/W
pma_addr*n* (*n*: 0-15)	Physical memory attribute address register	0xBD0+0x1**n*	R/W
Machine Hardware Loop CSRs (Custom)			
mhloop0_start_addr	32-bit start address for loop 0	0x7C6	R/W
mhloop0_end_addr	32-bit start address for loop 0	0x7C7	R/W
mhloop0_count	Iteration count for hardware loop 0	0x7C8	R/W
mhloop1_start_addr	32-bit start address for loop 1	0x7C9	R/W
mhloop1_end_addr	32-bit end address for loop 1	0x7CA	R/W
mhloop1_count	Iteration count for hardware loop 1	0x7CB	R/W
User Hardware Loop CSRs (Custom)			
uhloop0_start_addr	32-bit start address for loop 0	0x8C0	R/W
uhloop0_end_addr	32-bit end address for loop 0	0x8C1	R/W
uhloop0_count	Iteration count for hardware loop 0	0x8C2	R/W
uhloop1_start_addr	32-bit start address for loop 1	0x8C3	R/W
uhloop1_end_addr	32-bit end address for loop 1	0x8C4	R/W
uhloop1_count	Iteration count for hardware loop 1	0x8C5	R/W
uhwloop_state_reg	User HWLP state register	0x8C6	R/W
Bus Error Exception Context CSRs (Custom)			
ldpc0	Load bus error PC register 0	0xBE0	R/W
ldpc1	Load bus error PC register 1	0xBE1	R/W
ldtval0	Load bus error access address register 0	0xBE8	R/W
ldtval1	Load bus error access address register 1	0xBE9	R/W

Name	Description	Address	Access
stpc0	Store bus error PC register 0	0xBF0	R/W
stpc1	Store bus error PC register 1	0xBF1	R/W
stpc2	Store bus error PC register 2	0xBF2	R/W
sttval0	Store bus error access address register 0	0xBF8	R/W
sttval1	Store bus error access address register 1	0xBF9	R/W
sttval2	Store bus error access address register 2	0xBFA	R/W

Note that if write/set/clear operation is attempted on any of the CSRs which are read-only (RO), as indicated in the above table, the CPU will generate an illegal instruction exception.

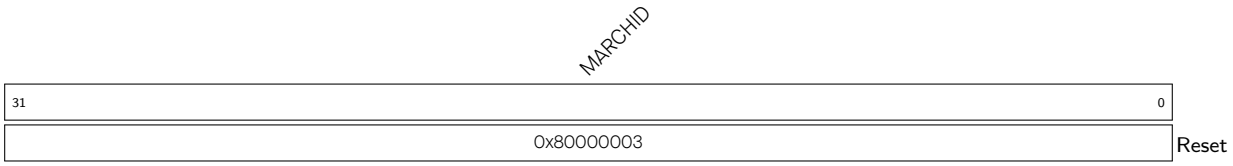
2.5.2 Register Description

Register 2.1. mvendorid (0xF11)



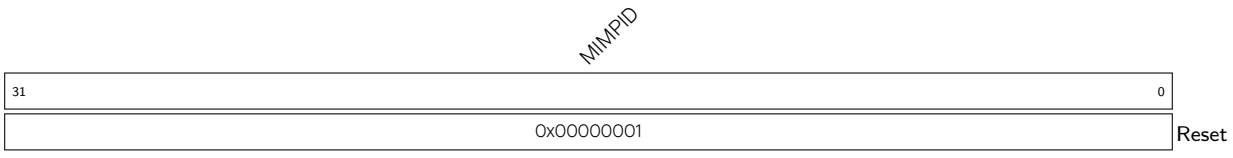
MVENDORID Represents Vendor ID. (RO)

Register 2.2. marchid (0xF12)



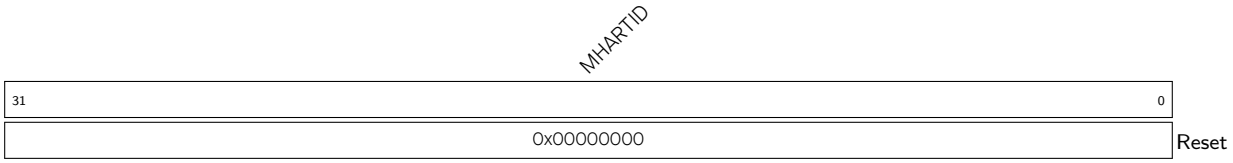
MARCHID Represents Architecture ID. (RO)

Register 2.3. mimpid (0xF13)



MIMPID Represents Implementation ID. (RO)

Register 2.4. mhartid (0xF14)



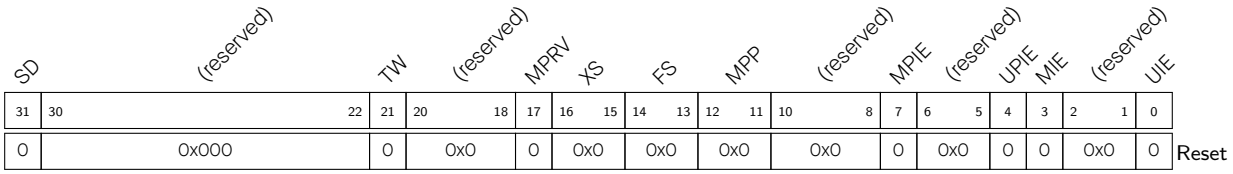
MHARTID Represents Hart ID.

0: Hart 0

1: Hart 1

(RO)

Register 2.5. mstatus (0x300)



SD Automatically set when either the FS or XS fields signal the presence of some dirty state that will require saving extension context to memory. (RO)

TW Configures whether the WFI (wait for interrupt) instruction can execute in less privileged modes.
0: WFI can execute in lower privilege modes.
1: WFI can only be executed in machine mode, and if executed in a less privileged mode, it will trigger an illegal instruction exception.
(R/W)

MPRV Configures whether to apply mstatus.MPP as the effective privilege mode, in which loads and stores execute, instead of the actual privilege mode in which the CPU is executing.
0: Not apply
1: Apply
Note that instruction protection is unaffected by this bit.
(R/W)

XS Represents the combined status of the custom extension states, including the HWLP and the PIE extensions.
0x0: All extension are Off
0x1: None dirty or clean, some on
0x2: None dirty, some clean
0x3: Some dirty
Note that in order to modify the state of either of these extensions, the corresponding CSRs mext_hwlp_status and mext_pie_status must be used.
(RO)

FS Represents the status of the floating-point unit, including the floating-point registers f0–f31 and the CSRs fcsr, frm, and fflags.
0x0: Off
0x1: Initial
0x2: Clean
0x3: Dirty
(R/W)

MPP Configures machine previous privilege mode (before trap).
0x0: User mode
0x3: Machine mode
Note: Only the lower bit is writable. Any write to the higher bit is ignored as it is directly tied to the lower bit.
(R/W)

Continued on the next page...

Register 2.5. mstatus (0x300)

Continued from the previous page...

MPIE Configures whether to enable the machine previous interrupt (before trap).

0: Disable

1: Enable

(R/W)

UPIE Configures whether to enable the user previous interrupt (before trap).

0: Disable

1: Enable

(R/W)

MIE Configures whether to enable the global machine interrupt.

0: Disable

1: Enable

(R/W)

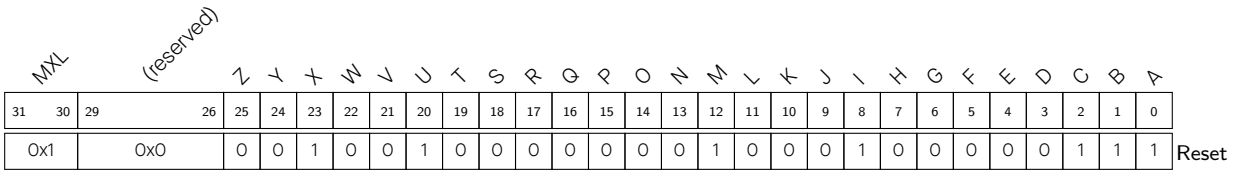
UIE Configures whether to enable the global user interrupt.

0: Disable

1: Enable

(R/W)

Register 2.6. misa (0x301)



MXL Machine XLEN = 1 (32-bit). (RO)

Z Reserved = 0. (RO)

Y Reserved = 0. (RO)

X Non-standard extensions present = 1. (RO)

W Reserved = 0. (RO)

V Reserved = 0. (RO)

U User mode implemented = 1. (RO)

T Reserved = 0. (RO)

S Supervisor mode implemented = 0. (RO)

R Reserved = 0. (RO)

Q Quad-precision floating-point extension = 0. (RO)

P Reserved = 0. (RO)

O Reserved = 0. (RO)

N User-level interrupts supported = 0. (RO)

M Integer Multiply/Divide extension = 1. (RO)

L Reserved = 0. (RO)

K Reserved = 0. (RO)

J Reserved = 0. (RO)

I RV32I base ISA = 1. (RO)

H Hypervisor extension = 0. (RO)

G Additional standard extensions present = 0. (RO)

F Single-precision floating-point extension = 0. (RO)

E RV32E base ISA = 0. (RO)

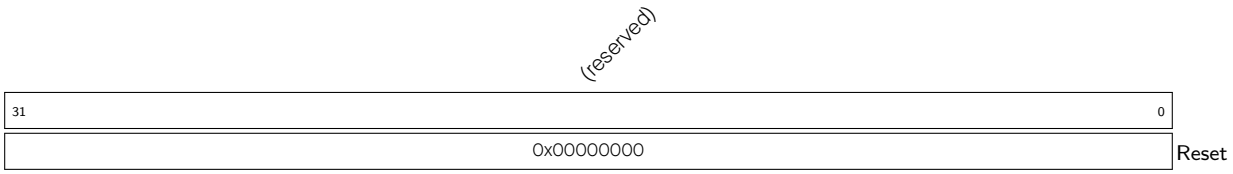
D Double-precision floating-point extension = 0. (RO)

C Compressed Extension = 1. (RO)

B Bit Manipulation = 1. (RO)

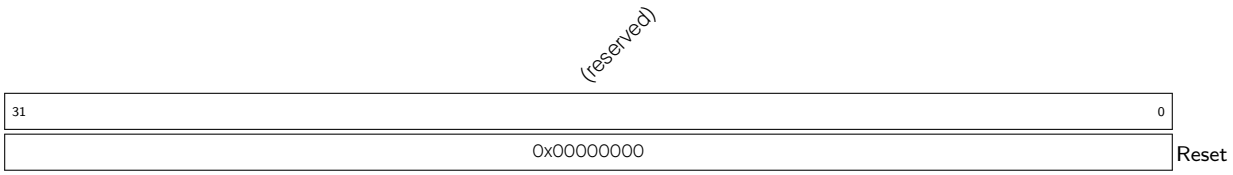
A Atomic Extension = 1. (RO)

Register 2.7. mideleg (0x303)



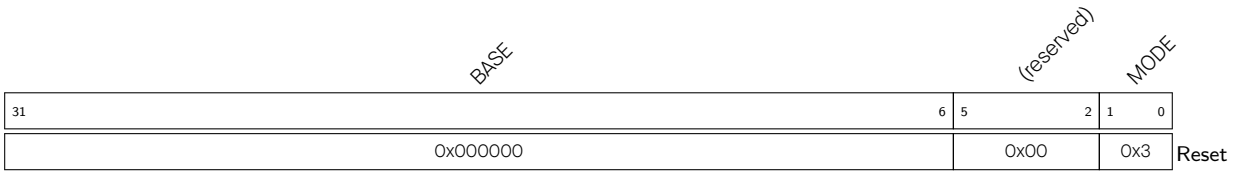
mideleg All bits are hardwired to 0 since only CLIC mode is available. (RO)

Register 2.8. mie (0x304)



mie All bits are hardwired to 0 since only CLIC mode is available. (RO)

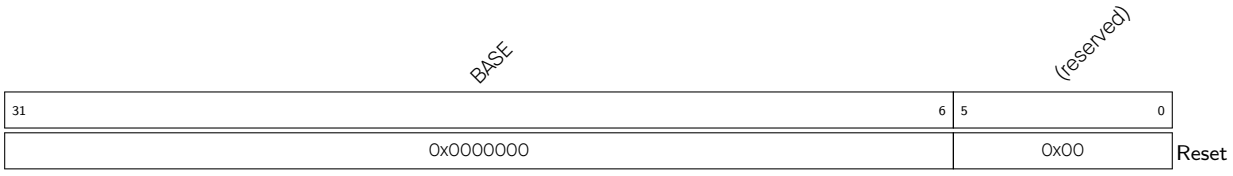
Register 2.9. mtvec (0x305)



BASE Configures the higher 26 bits of exception and non-vectored machine mode interrupt base address aligned to 64 bytes. (R/W)

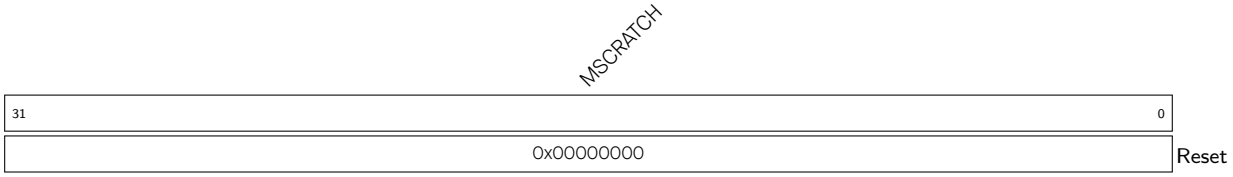
MODE Represents whether machine mode interrupts are operating in CLIC mode or vectored/non-vectored CLINT mode. Only CLIC mode **0x3** is available. (RO)

Register 2.10. mtvt (0x307)



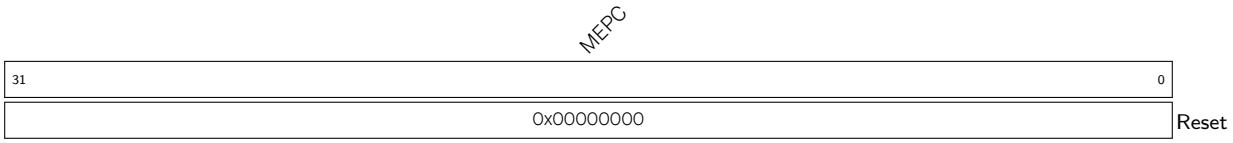
BASE Configures the higher 26 bits of CLIC machine mode interrupt vector base address aligned to 64 bytes. (R/W)

Register 2.11. mscratch (0x340)



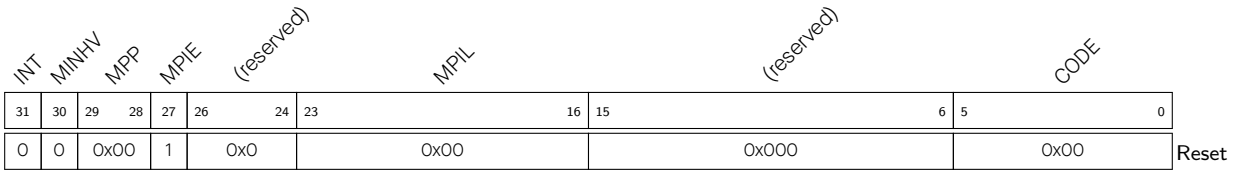
MSCRATCH Configures machine scratch information for custom use. (R/W)

Register 2.12. mepc (0x341)



MEPC Configures the machine trap/exception program counter. This is automatically updated with address of the instruction which was about to be executed while CPU encountered the most recent trap. (R/W)

Register 2.13. mcause (0x342)



INT Represents whether the CPU entered a trap due to interrupts.

0: The trap occurred due to an exception.

1: The trap occurred due to an interrupt.

(R/W)

MINHV Represents whether mepc is an address of an instruction or an address of a vector table entry.

1: Address of a table entry

0: Address of an instruction

(R/W)

MPP Represents the previous privilege mode, same as mstatus.MPP. (R/W)

MPIE Represents the previous interrupt enable, same as mstatus.MPIE. (R/W)

MPIL Represents the previous 8-bit interrupt level. (R/W)

CODE Represents the unique ID of the most recent exception or interrupt due to which CPU entered trap. Possible exception IDs are:

0x01: Instruction access fault

0x02: Illegal instruction

0x03: Hardware breakpoint/watchpoint or EBREAK

0x05: PMP load access fault

0x06: Misaligned store address or AMO address

0x07: Store access or AMO access fault

0x08: ECALL from U mode

0x0b: ECALL from M mode

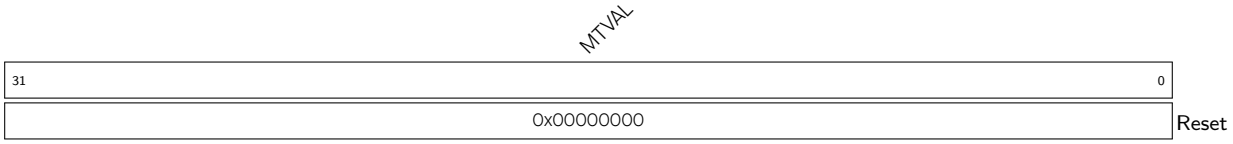
0x1f: Illegal PIE instruction

Other exception IDs are reserved.

Note: Exception ID 0x0 (instruction access misaligned) is not present because CPU always masks the lowest bit of the address during instruction fetch.

(R/W)

Register 2.14. mtval (0x343)



MTVAL Represents the machine trap value. This is automatically updated with an exception dependent data which may be useful for handling that exception.

Data is to be interpreted depending upon exception IDs:

0x01: Faulting virtual address of instruction

0x02: Faulting instruction opcode

0x05: Faulting data address of load operation

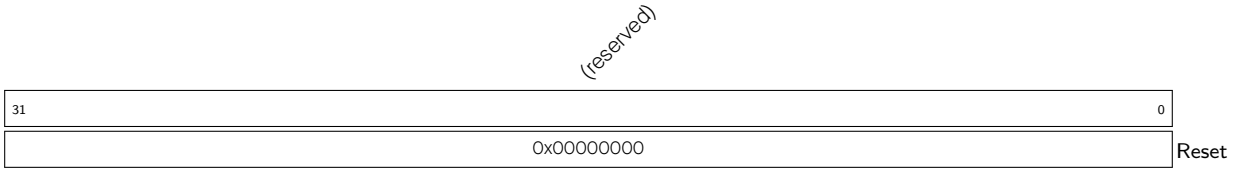
0x07: Faulting data address of store operation

0x1F: Faulting instruction opcode related to illegal PIE instruction

Note: The value of this register is not valid for other exception IDs and interrupts.

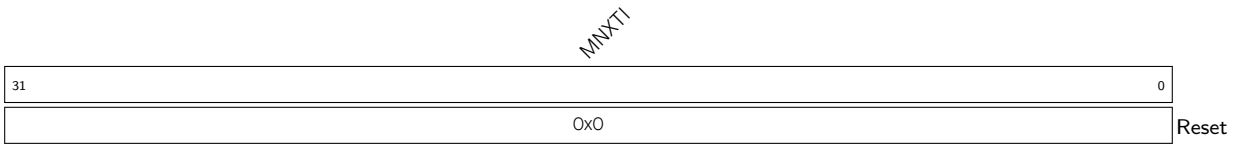
(R/W)

Register 2.15. mip (0x344)



All bits are hardwired to 0 since only CLIC mode is available. (RO)

Register 2.16. mnxti (0x345)



MNXTI Represents the next machine mode interrupt entry address and configures the interrupt enable status.

This CSR can be used by software to service the next horizontal interrupt for the same privilege mode when it has greater level than the saved interrupt context (held in mcause.MPIL) and greater level than the interrupt threshold of the corresponding privilege mode, without incurring the full cost of an interrupt pipeline flush and context save/restore.

This CSR is designed to be accessed using CSRRSI/CSRRCI instructions, where the value read is a pointer to an entry in the trap handler table and the write back updates the interrupt-enable status. In addition, writes to the mnxti have side-effects that update the interrupt context state. A read of the mnxti CSR using CSRR returns either zero, indicating there is no suitable interrupt to service or that the system is not in a CLIC mode, or returns a non-zero address of the entry in the trap handler table for software trap vectoring.

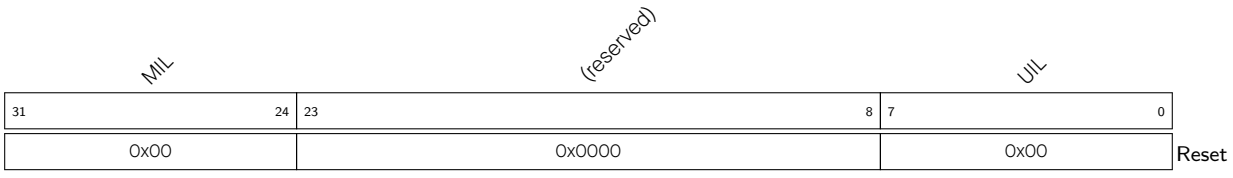
If the CSR instruction that accesses mnxti includes a write, the mstatus CSR is the one used for the read-modify-write portion of the operation, while the mcause register's exccode field and the mintstatus.MIL field can also be updated with the new interrupt id and level. If the interrupt is edge-triggered, then the pending bit is also zeroed.

The mnxti CSR is intended to be used inside an interrupt handler after an initial interrupt has been taken and mcause and mepc registers updated with the interrupted context and the ID of the interrupt.

If the pending interrupt is edge-triggered, hardware will automatically clear the corresponding pending bit when the CSR instruction that accesses mnxti includes a write. However, if the CSR instruction does not include write side effects, then no state update on any CSR occurs and thus the interrupt pending bit is not zeroed. This behavior allows software to optimize the selection and execution of interrupts using mnxti.

(R/W)

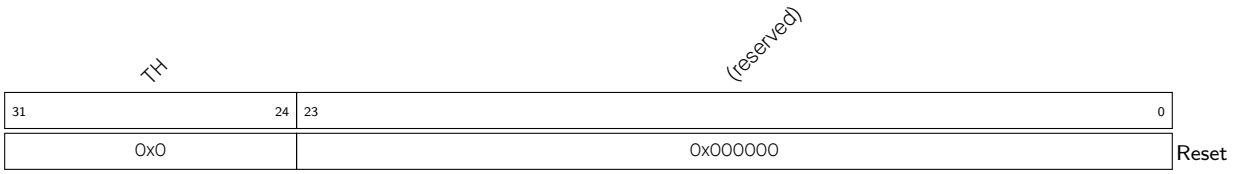
Register 2.17. mintstatus (0xFB1)



MIL Represents the active 8-bit interrupt level for current machine mode interrupt. (RO)

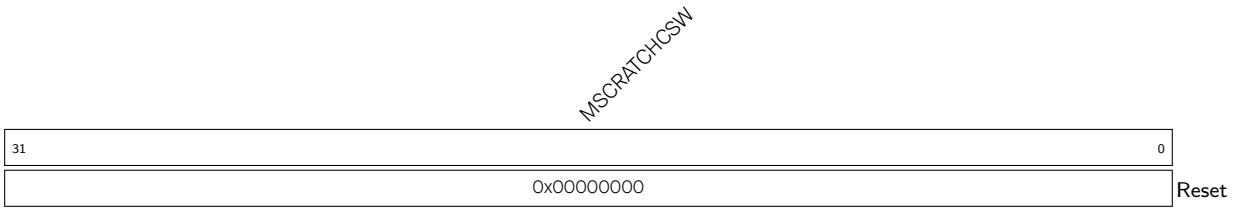
UIL Hardwired to 0x0 as user mode interrupts are not supported. (RO)

Register 2.18. mintthresh (0x347)



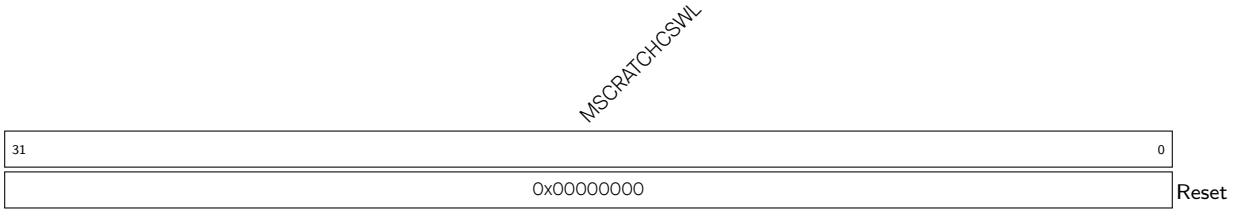
TH Configures the 8-bit level threshold of machine mode interrupts.
Note that the effective threshold level for machine mode interrupts is the maximum of mintthresh.TH and mintstatus.MIL. All machine mode pending interrupts with levels less than or equal to the effective threshold level are not allowed to preempt the execution.
(R/W)

Register 2.19. mscratchcsw (0x348)



MSCRATCHCSW Configures the MSCRATCH value by conditionally swapping its value with RS1, based on the current privilege mode and the previous privilege mode in MPP.
This is the conditional scratch swap CSR, which allows performing a scratch value swap conditionally, based on privilege mode change, in a single instruction.
When using CSRRW instruction to access this CSR, the value written into RD is either that of MSCRATCH, if MPP is different than the current privilege mode, or RS1 if MPP is the same as the current privilege mode. The MSCRATCH CSR value is updated with the original value of RS1 only if there is a privilege mode difference.
(R/W)

Register 2.20. mscratchcswl (0x349)



MSCRATCHCSWL Configures the MSCRATCH value by conditionally swapping its value with RS1, based on the current interrupt level in mintstatus.MIL and the previous interrupt level in mcause.MPIL.

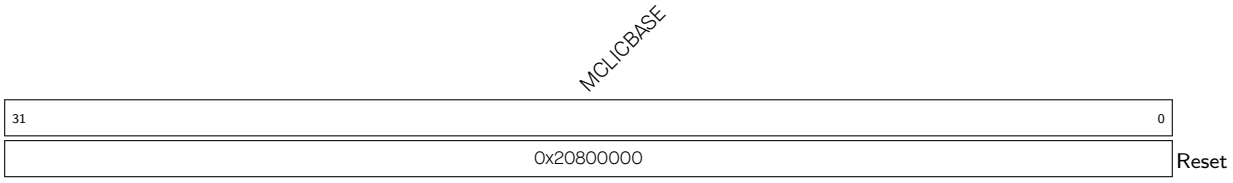
This is the conditional scratch swap CSR, which allows performing a scratch value swap conditionally, based on interrupt level change, in a single instruction.

When using CSRRW instruction to access this CSR, the value written into RD is either that of MSCRATCH, if only one of mcause.MPIL and mintstatus.MIL is equal to zero, or RS1 for all other cases.

The value of MSCRATCH is updated with the initial value of RS1 only if one of mcause.MPIL and mintstatus.MIL is equal to zero.

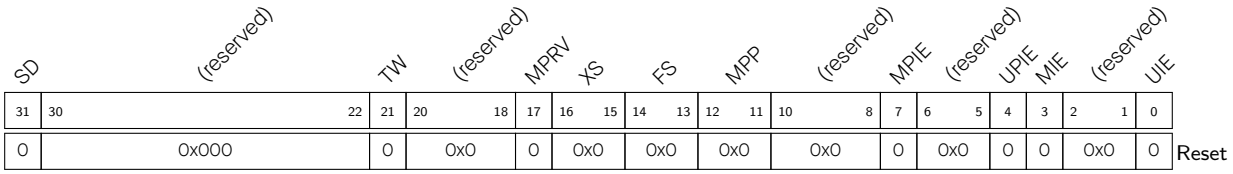
(R/W)

Register 2.21. mclicbase (0x350)



MCLICBASE Represents the base address of the CLIC memory-mapped registers for machine mode. (RO)

Register 2.22. ustatus (0x000)



SD Automatically set when either the FS or XS fields signal the presence of some dirty state that will require saving extension context to memory. (RO)

TW Represents whether the WFI (wait for interrupt) instruction can execute in less privileged modes.
0: WFI can execute in lower privilege modes.
1: WFI can only be executed in machine mode, and if executed in a less privileged mode, it will trigger an illegal instruction exception.
(RO)

MPRV Represents whether to apply ustatus.MPP as the effective privilege mode, in which loads and stores execute, instead of the actual privilege mode in which the CPU is executing.
0: Not apply
1: Apply
Note that instruction protection is unaffected by this bit.
(RO)

XS Represents the combined status of the custom extension states, including the HWLP and the PIE extensions.
0x0: All extension are Off
0x1: None dirty or clean, some on
0x2: None dirty, some clean
0x3: Some dirty
Note that in order to modify the state of either of these extensions, the corresponding CSRs mext_hwlp_status and mext_pie_status must be used.
(RO)

FS Configures the status of the floating-point unit, including the floating-point registers f0–f31 and the CSRs fcsr, frm, and fflags.
0x0: Off
0x1: Initial
0x2: Clean
0x3: Dirty
(R/W)

MPP Represents machine previous privilege mode (before trap).
0x0: User mode
0x3: Machine mode
Note: Only the lower bit is writable. Any write to the higher bit is ignored as it is directly tied to the lower bit.
(RO)

Continued on the next page...

Register 2.22. ustatus (0x000)

Continued from the previous page...

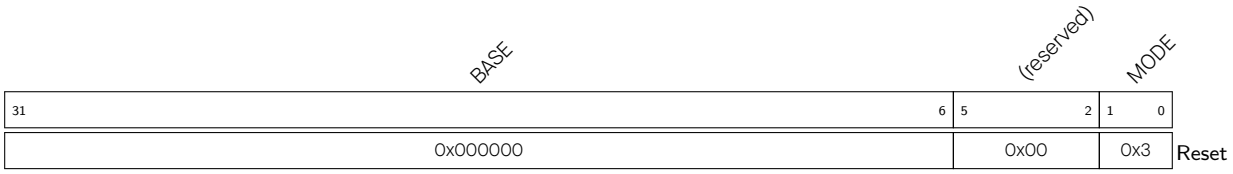
MPIE Represents machine previous interrupt enable status (before trap).
0: Disabled
1: Enabled
(RO)

UPIE Represents user previous interrupt enable status (before trap).
0: Disabled
1: Enabled
(RO)

MIE Represents global machine interrupt enable status.
0: Disabled
1: Enabled
(RO)

UIE Represents global user mode interrupt enable status.
0: Disabled
1: Enabled
(RO)

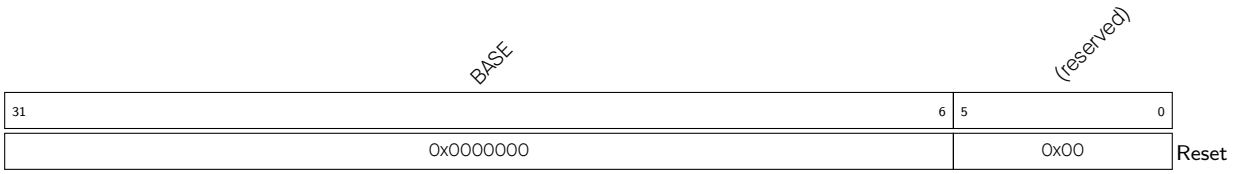
Register 2.23. utvec (0x005)



BASE Configures the higher 26 bits of exception and non-vectored user mode interrupt base address aligned to 64 bytes. (R/W)

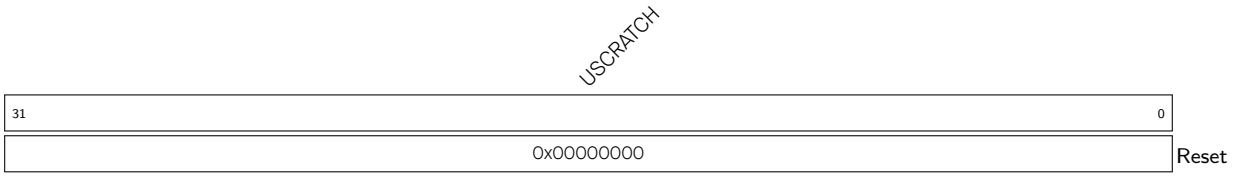
MODE Represents whether user mode interrupts are operating in CLIC mode or vectored/non-vectored CLINT mode. Only CLIC mode **0x3** is available. (RO)

Register 2.24. utvt (0x007)



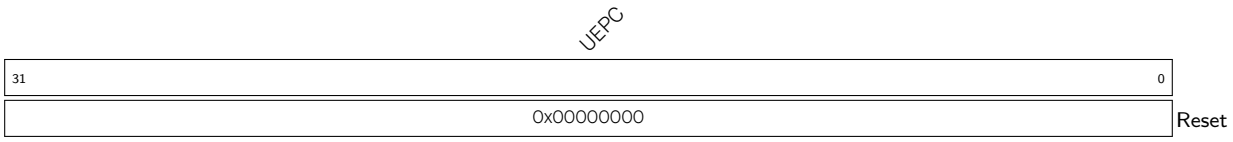
BASE Configures the higher 26 bits of CLIC user mode interrupt vector base address aligned to 64 bytes. (R/W)

Register 2.25. uscratch (0x040)



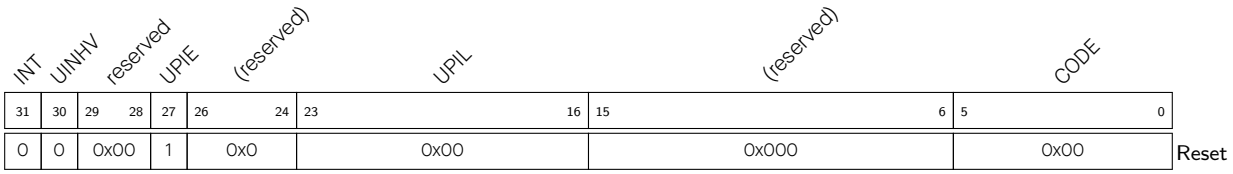
USCRATCH Configures user scratch information for custom use. (R/W)

Register 2.26. uepc (0x041)



UEPC Configures the user trap/exception program counter. This is automatically updated with address of the instruction which was about to be executed while CPU encountered the most recent trap. (R/W)

Register 2.27. ucause (0x042)



INT Represents whether the CPU entered a trap due to interrupts.

0: The trap occurred due to an exception.

1: The trap occurred due to an interrupt.

(R/W)

UINHV Represents whether uepc is an address of an instruction or an address of a vector table entry.

1: Address of a table entry

0: Address of an instruction

(R/W)

UPIE Represents the previous interrupt enable, same as ustatus.UPIE. (R/W)

UPIL Represents the previous 8-bit interrupt level. (R/W)

CODE Represents the unique ID of the most recent exception or interrupt due to which CPU entered trap. Possible exception IDs are:

0x01: Instruction access fault

0x02: Illegal instruction

0x03: Hardware breakpoint/watchpoint or EBREAK

0x05: PMP load access fault

0x06: Misaligned store address or AMO address

0x07: Store access or AMO access fault

0x08: ECALL from U mode

0x0b: ECALL from M mode

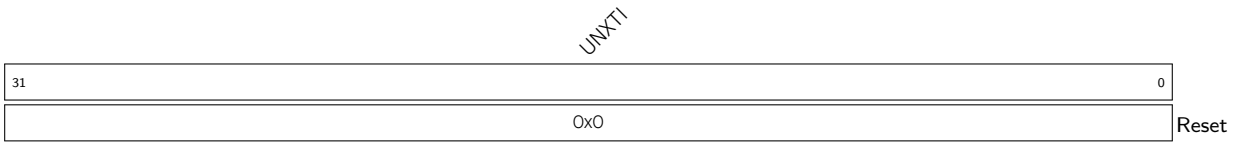
0x1f: Illegal PIE instruction

Other exception IDs are reserved.

Note: Exception ID 0x0 (instruction access misaligned) is not present because CPU always masks the lowest bit of the address during instruction fetch.

(R/W)

Register 2.28. unxti (0x045)



UNXTI Represents the next user mode interrupt entry address and configures the interrupt enable status.

This CSR can be used by software to service the next horizontal interrupt for the same privilege mode when it has greater level than the saved interrupt context (held in ucause.UPIL) and greater level than the interrupt threshold of the corresponding privilege mode, without incurring the full cost of an interrupt pipeline flush and context save/restore.

This CSR is designed to be accessed using CSRRSI/CSRRCI instructions, where the value read is a pointer to an entry in the trap handler table and the write back updates the interrupt-enable status. In addition, writes to the unxti have side-effects that update the interrupt context state.

A read of the unxti CSR using CSRR returns either zero, indicating there is no suitable interrupt to service or that the system is not in a CLIC mode, or returns a non-zero address of the entry in the trap handler table for software trap vectoring.

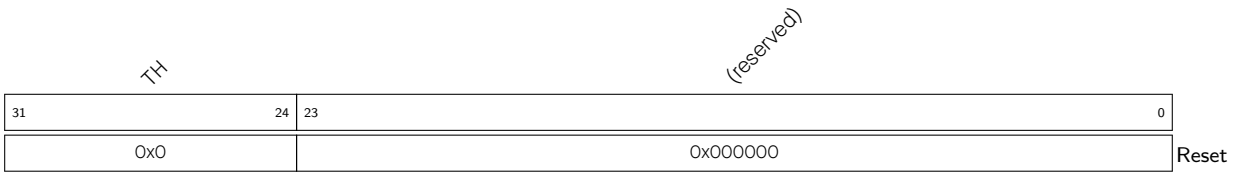
If the CSR instruction that accesses unxti includes a write, the ustatus CSR is the one used for the read-modify-write portion of the operation, while the ucause register's exccode field and the uintstatus.UIL field can also be updated with the new interrupt id and level. If the interrupt is edge-triggered, then the pending bit is also zeroed.

The unxti CSR is intended to be used inside an interrupt handler after an initial interrupt has been taken and ucause and uepc registers updated with the interrupted context and the ID of the interrupt.

If the pending interrupt is edge-triggered, hardware will automatically clear the corresponding pending bit when the CSR instruction that accesses unxti includes a write. However, if the CSR instruction does not include write side effects, then no state update on any CSR occurs and thus the interrupt pending bit is not zeroed. This behavior allows software to optimize the selection and execution of interrupts using unxti.

(R/W)

Register 2.29. uintthresh (0x047)

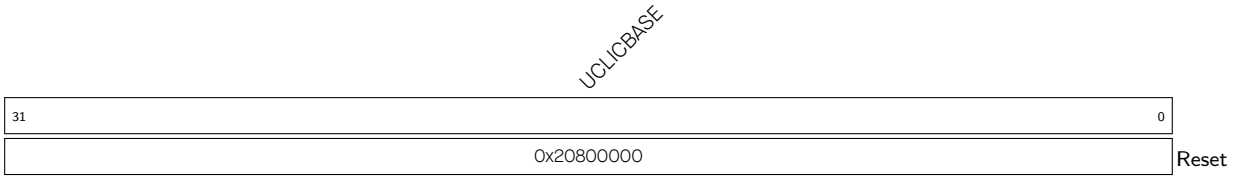


TH Configures the 8-bit level threshold of user mode interrupts.

Note that the effective threshold level for user mode interrupts is the maximum of uintthresh.TH and uintstatus.UIL. All user mode pending interrupts with levels less than or equal to the effective threshold level are not allowed to preempt the execution.

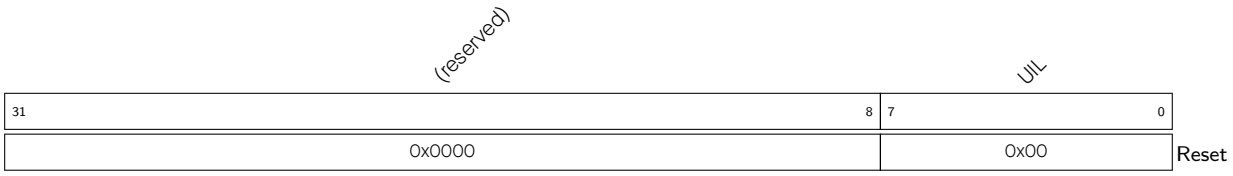
(R/W)

Register 2.30. uclicbase (0x050)



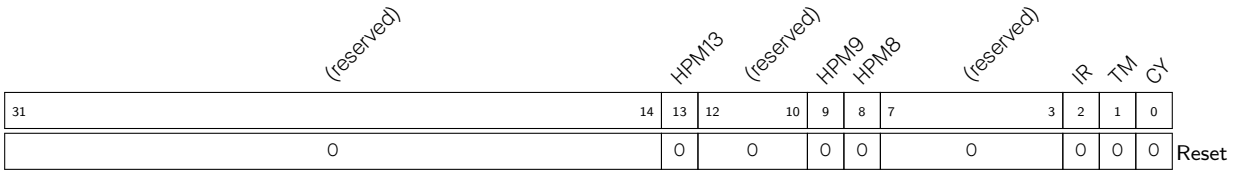
UCLICBASE Represents the base address of the CLIC memory-mapped registers for user mode. (RO)

Register 2.31. uintstatus (0xCB1)



UIL Represents the active 8-bit interrupt level for the current user mode interrupt. (RO)

Register 2.32. mcounteren (0x306)



HPM13 Configures inline]Configures users counters instead of machine counters access from user mode. Bhanu: Updated hpmcounter13 access from user mode.

0: Accessing hpmcounter13 CSR in user mode will generate illegal instruction exception.

1: Accessing hpmcounter13 CSR in user mode is valid.

(R/W)

HPM9 Configures hpmcounter9 access from user mode.

0: Accessing hpmcounter9 CSR in user mode will generate illegal instruction exception.

1: Accessing hpmcounter9 CSR in user mode is valid.

(R/W)

HPM8 Configures hpmcounter8 access from user mode.

0: Accessing hpmcounter8 CSR in user mode will generate illegal instruction exception.

1: Accessing hpmcounter8 CSR in user mode is valid.

(R/W)

CY Configures access of cycle counter from user mode.

0: Accessing cycle CSR in user mode will generate illegal instruction exception.

1: Accessing cycle CSR in user mode is valid.

(R/W)

TM Configures access of time counter from user mode.

0: Accessing time CSR in user mode will generate illegal instruction exception.

1: Accessing time CSR in user mode is valid.

(R/W)

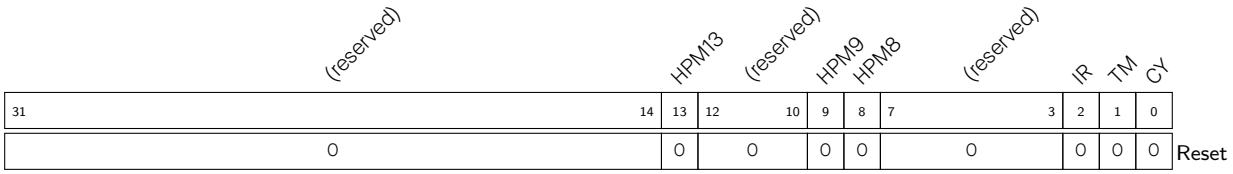
IR Configures access of instret counter from user mode.

0: Accessing instret CSR in user mode will generate illegal instruction exception.

1: Accessing instret CSR in user mode is valid.

(R/W)

Register 2.33. mcountinhibit (0x320)



HPM13 Configures the incrementing of mhpmcounter13.

0: Continue incrementing the mhpmcounter13 counter.

1: Stop incrementing the mhpmcounter13 counter.

(R/W)

HPM9 Configures the incrementing of mhpmcounter9.

0: Continue incrementing the mhpmcounter9 counter.

1: Stop incrementing the mhpmcounter9 counter.

(R/W)

HPM8 Configures the incrementing of mhpmcounter8.

0: Continue incrementing the mhpmcounter8 counter.

1: Stop incrementing the mhpmcounter8 counter.

(R/W)

IR Configures the incrementing of the instret counter.

0: Continue incrementing the instret counter.

1: Stop incrementing the instret counter.

(R/W)

TM Configures the incrementing of the time counter.

0: Continue incrementing the time counter.

1: Stop incrementing the time counter.

(R/W)

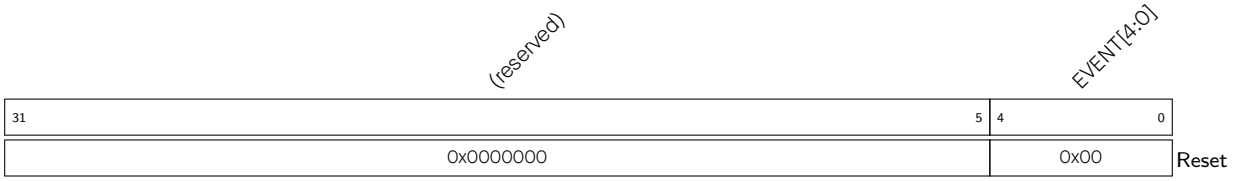
CY Configures the incrementing of the cycle counter.

0: Continue incrementing the cycle counter.

1: Stop incrementing the cycle counter.

(R/W)

Register 2.34. mhpmevent8 (0x328)

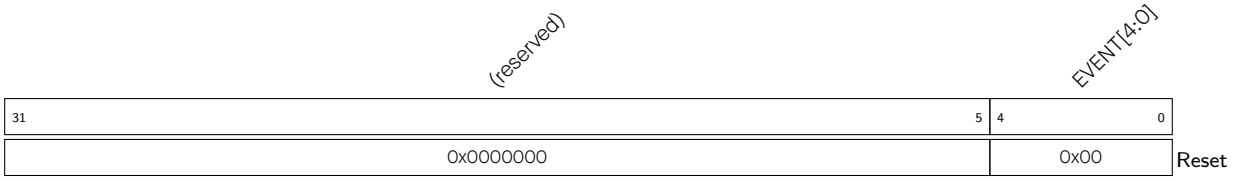


EVENT[4:0] Event selector for mhpmcounter8.

The only valid event value for this counter is 0x6, for conditional branch mispredictions.

(R/W)

Register 2.35. mhpmevent9 (0x329)

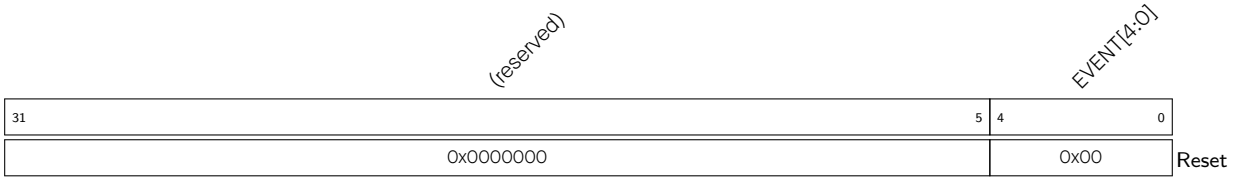


EVENT[4:0] Event selector for mhpmcounter9.

The only valid event value for this counter is 0x7, for conditional branch instructions.

(R/W)

Register 2.36. mhpmevent13 (0x32D)

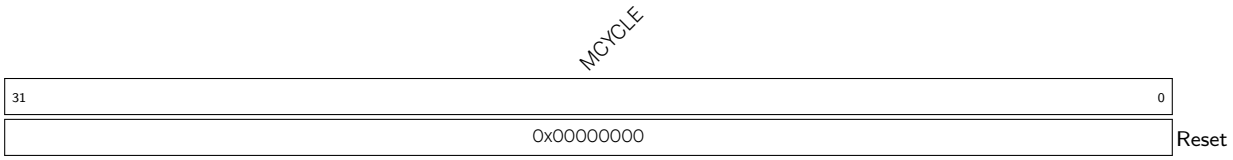


EVENT[4:0] Event selector for mhpmcounter13.

The only valid event value for this counter is 0xB, for store instructions.

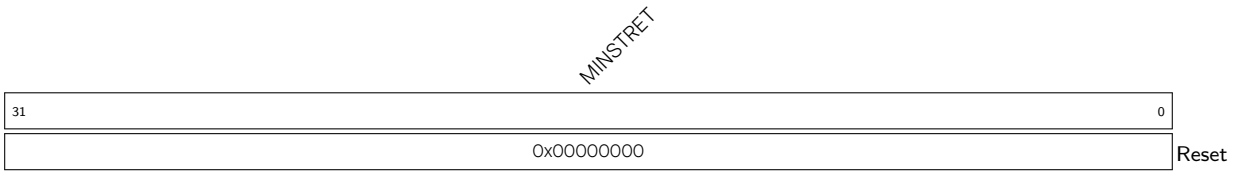
(R/W)

Register 2.37. mcycle (0xB00)



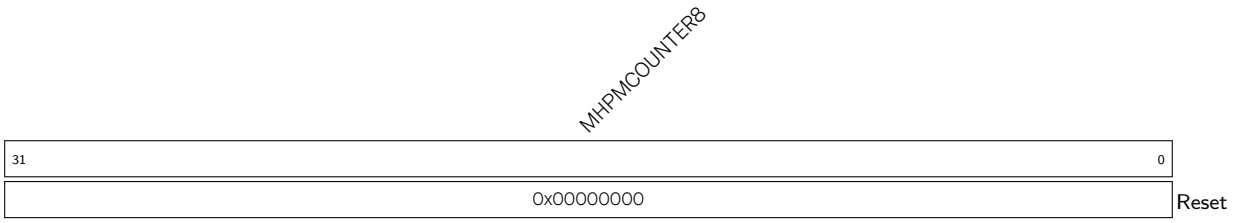
MCYCLE Represents the lower 32 bits of the machine mode cycle counter. (R/W)

Register 2.38. minstret (0xB02)



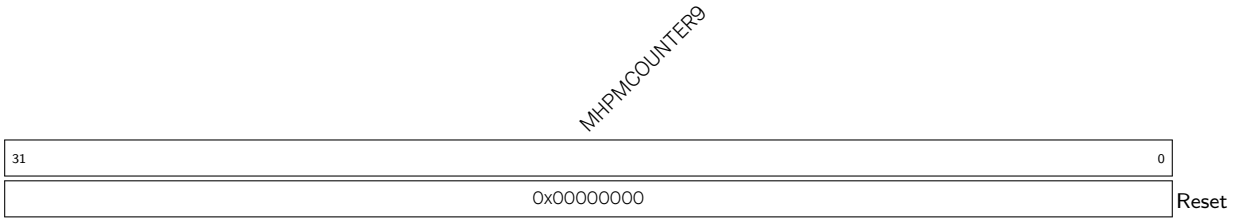
MINSTRET Represents the lower 32 bits of the machine instructions retired counter. (R/W)

Register 2.39. mhpmcounter8 (0xB08)



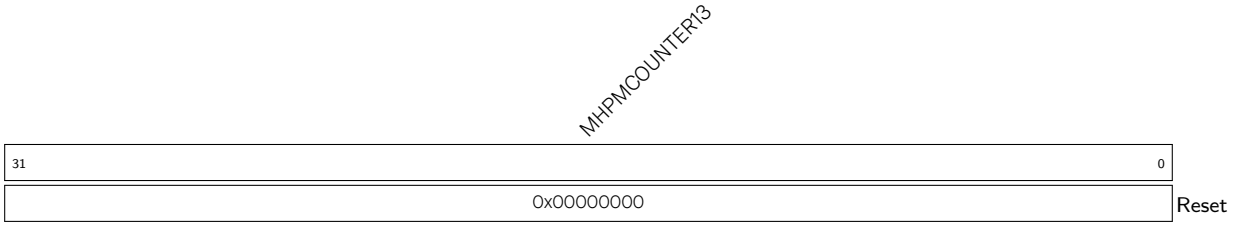
MHPMCOUNTER8 Represents the lower 32 bits of the machine performance-monitoring counter8. (R/W)

Register 2.40. mhpmcounter9 (0xB09)



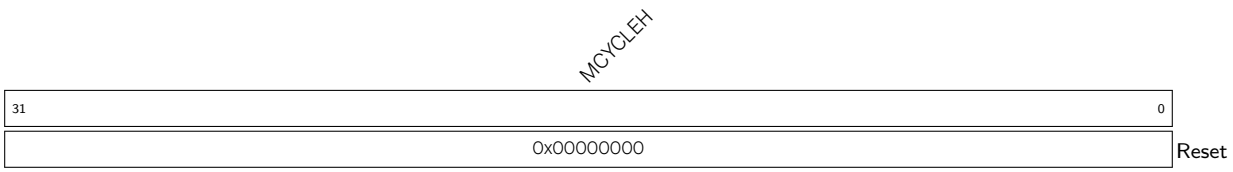
MHPMCOUNTER9 Represents the lower 32 bits of the machine performance-monitoring counter9. (R/W)

Register 2.41. mhpmcounter13 (0xB0D)



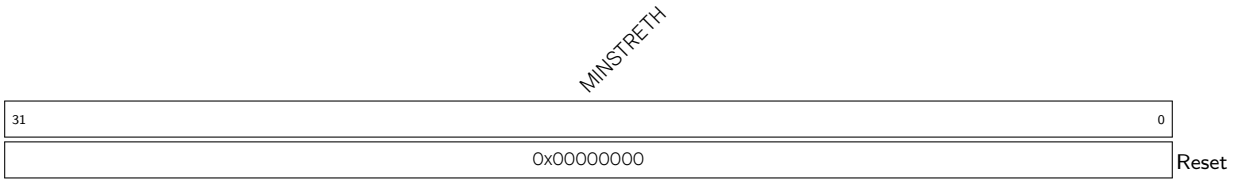
MHPMCOUNTER13 Represents the lower 32 bits of the machine performance-monitoring counter13. (R/W)

Register 2.42. mcycleh (0xB80)



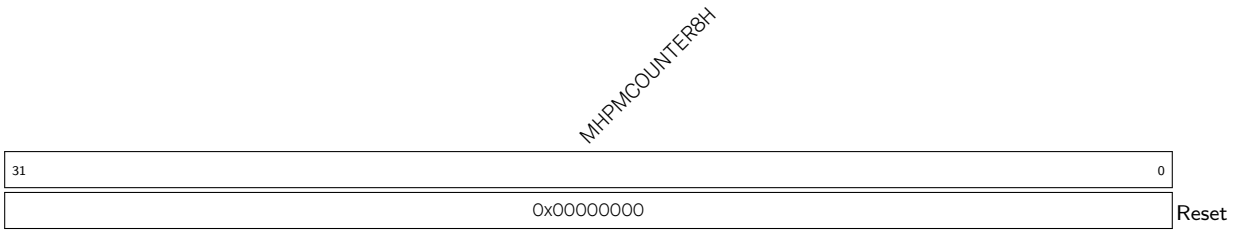
MCYCLEH Represents the upper 32 bits of mcycle counter. (R/W)

Register 2.43. minstreth (0xB82)



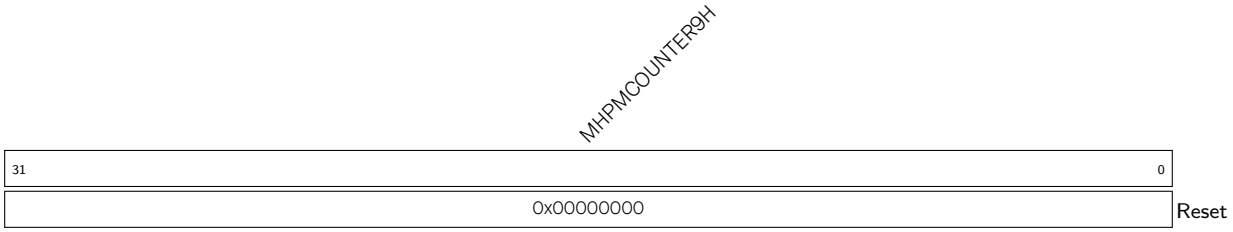
MINSTRETH Represents the upper 32 bits of minstret counter. (R/W)

Register 2.44. mhpmcounter8h (0xB88)



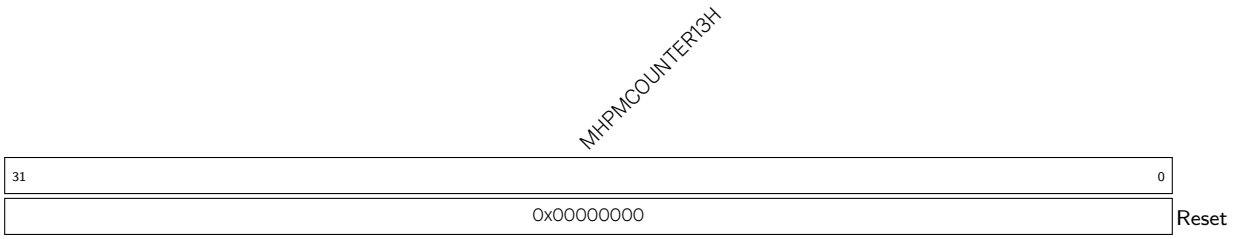
MHPMCOUNTER8H Represents the upper 32 bits of mhpmcounter8 counter. (R/W)

Register 2.45. mhpmcounter9h (0xB89)



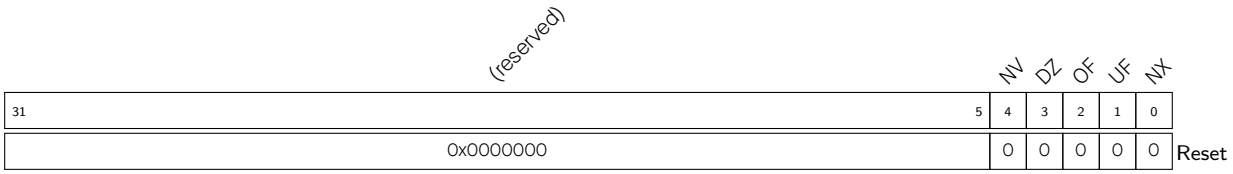
MHPMCOUNTER9H Represents the upper 32 bits of mhpmcounter9 counter. (R/W)

Register 2.46. mhpmcounter13h (0xB8D)



MHPMCOUNTER13H Represents the upper 32 bits of mhpmcounter13 counter. (R/W)

Register 2.47. fflags (0x001)



NV Represents the floating-point invalid operation flag. (R/W)

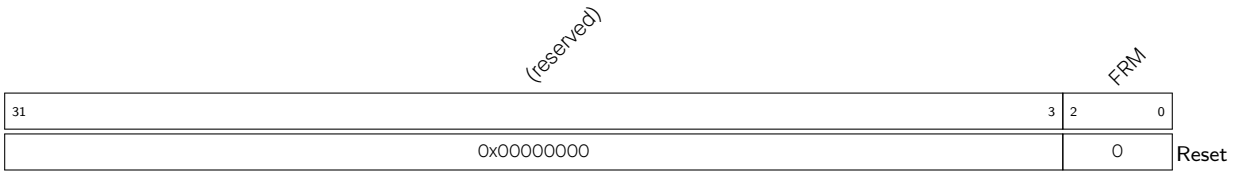
DZ Represents the floating-point divide-by-zero flag. (R/W)

OF Represents the floating-point overflow flag. (R/W)

UF Represents the floating-point underflow flag. (R/W)

NX Represents the floating-point inexact flag. (R/W)

Register 2.48. frm (0x002)



FRM Configures the floating-point rounding mode.

Valid rounding mode values:

0x0: RNE (Round to nearest, ties to even)

0x1: RTZ (Round towards zero)

0x2: RDN (Round down towards negative infinity)

0x3: RUP (Round up towards positive infinity)

0x4: RMM (Round to nearest, ties to maximum magnitude)

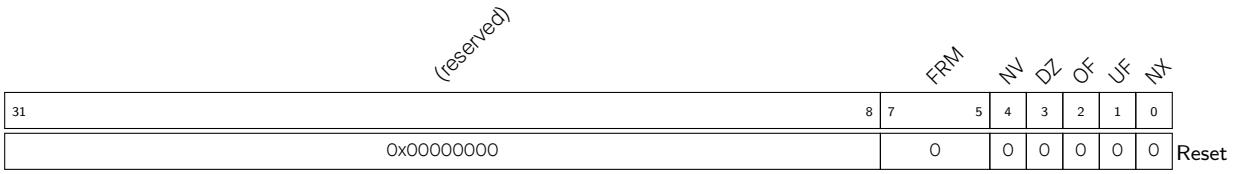
0x5: Reserved (Invalid. Reserved to future use)

0x6: Reserved (Invalid. Reserved to future use)

0x7: DYN (In Instruction's rm field, it selects dynamic rounding mode. In the rounding mode register, it is invalid)

(R/W)

Register 2.49. fcsr (0x003)



FRM Configures the floating-point rounding mode.

Valid rounding mode values:

0x0: RNE (Round to nearest, ties to even)

0x1: RTZ (Round towards zero)

0x2: RDN (Round down towards negative infinity)

0x3: RUP (Round up towards positive infinity)

0x4: RMM (Round to nearest, ties to maximum magnitude)

0x5: Reserved (Invalid. Reserved to future use)

0x6: Reserved (Invalid. Reserved to future use)

0x7: DYN (In Instruction's rm field, it selects dynamic rounding mode. In the rounding mode register, it is invalid)

(R/W)

NV Represents the floating-point invalid operation flag. (R/W)

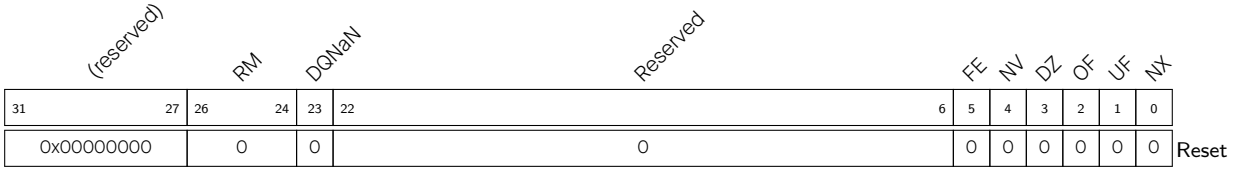
DZ Represents the floating-point divide-by-zero flag. (R/W)

OF Represents the floating-point overflow flag. (R/W)

UF Represents the floating-point underflow flag. (R/W)

NX Represents the floating-point inexact flag. (R/W)

Register 2.50. fxsr (0x800)



RM Mapped to frm bits. Write operation to this field will be reflected on fcsr register at the same time. (R/W)

DQNaN Configures the calculated value of QNaN output.

0: The calculated value of QNaN output is the fixed value specified in RISC-V, i.e., 0x7FC00000.

1: The calculated QNaN value is consistent with the IEEE 754 standard.

(R/W)

FE Floating-point exception accumulation bit. Mapped to FE bits. Write operation to this field will be reflected on fcsr register at the same time.

0: No floating-point exception occurred

1: Floating-point exception occurred

(R/W)

NV Represents the invalid operand exception flag. Mapped to NV bits. Write operation to this field will be reflected on fcsr register at the same time. (R/W)

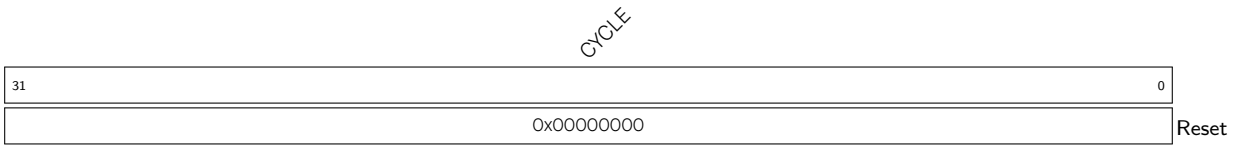
DZ Represents the divide-by-zero exception flag. Mapped to DZ bits. Write operation to this field will be reflected on fcsr register at the same time. (R/W)

OF Represents the overflow exception flag. Mapped to OF bits. Write operation to this field will be reflected on fcsr register at the same time. (R/W)

UF Represents the underflag. Mapped to UF bits. Write operation to this field will be reflected on fcsr register at the same time. (R/W)

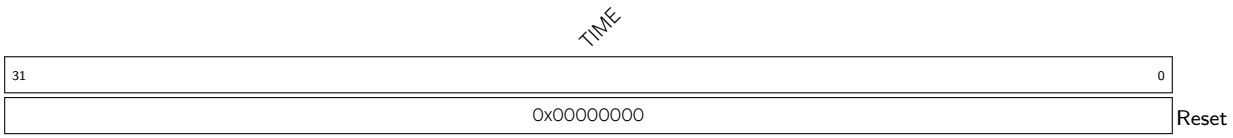
NX Represents the inexact flag. Mapped to NX bits. Write operation to this field will be reflected on fcsr register at the same time. (R/W)

Register 2.51. cycle (0xC00)



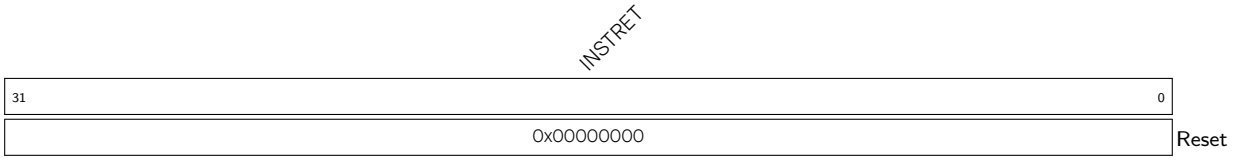
CYCLE Read-only mirror of mcycle. (RO)

Register 2.52. time (0xC01)



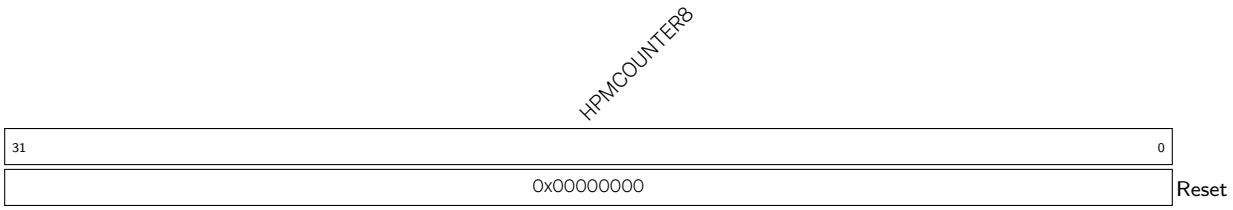
TIME Represents the timer value for RDTIME instruction. (RO)

Register 2.53. instret (0xC02)



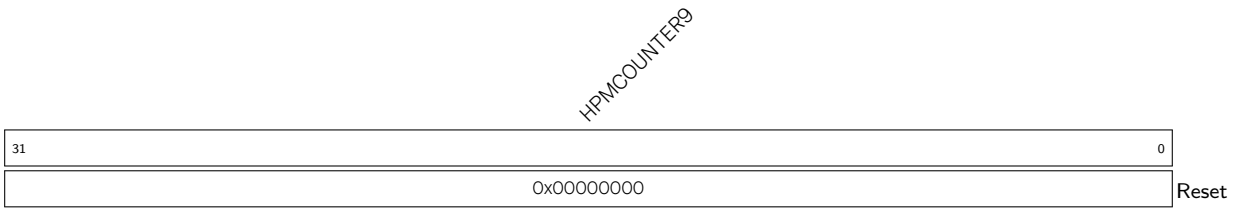
INSTRET Read-only mirror of minstret. (RO)

Register 2.54. hpmcounter8 (0xC08)



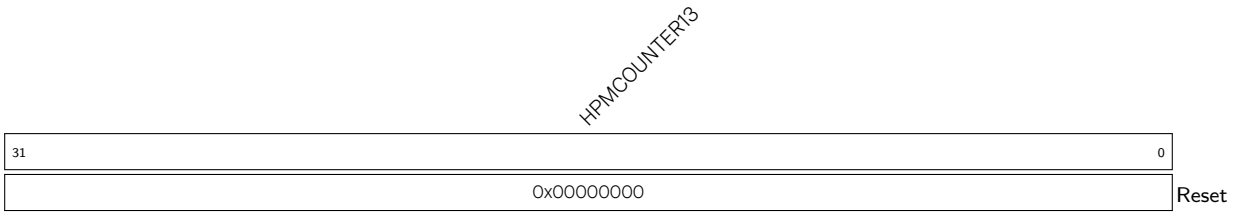
HPMCOUNTER8 Read-only mirror of mhpmcounter8. (RO)

Register 2.55. hpmcounter9 (0xC09)



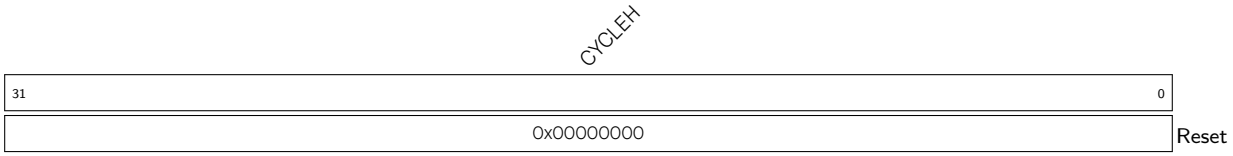
HPMCOUNTER9 Read-only mirror of mhpmcounter9. (RO)

Register 2.56. hpmcounter13 (0xC0D)



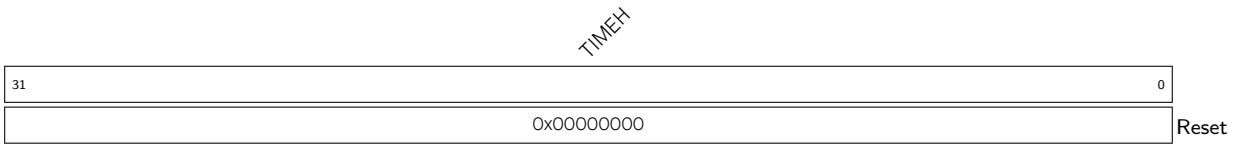
HPMCOUNTER13 Read-only mirror of mhpmcounter13. (RO)

Register 2.57. cycleh (0xC80)



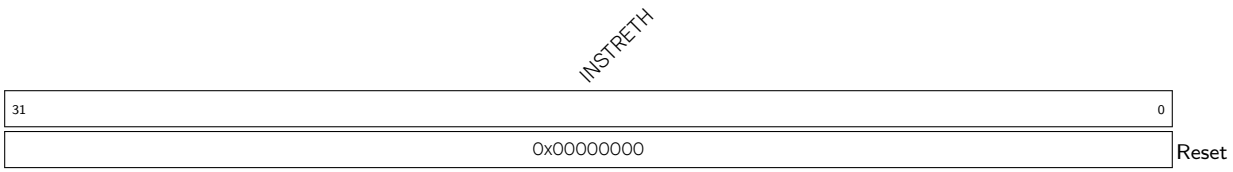
CYCLEH Read-only mirror of mcycleh. (RO)

Register 2.58. timeh (0xC81)



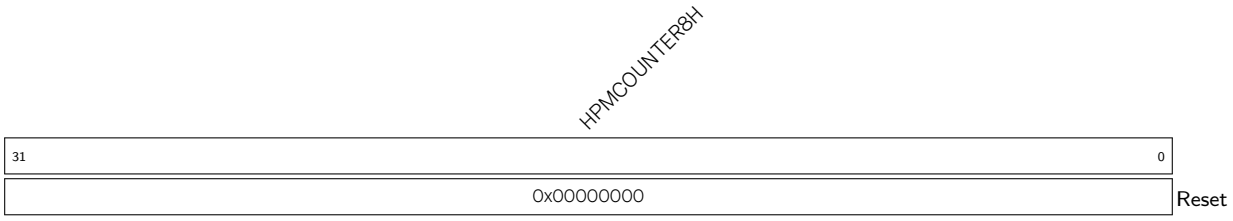
TIMEH Represents the upper 32 bits of the timer for RDTIME instruction. (RO)

Register 2.59. instreth (0xC82)



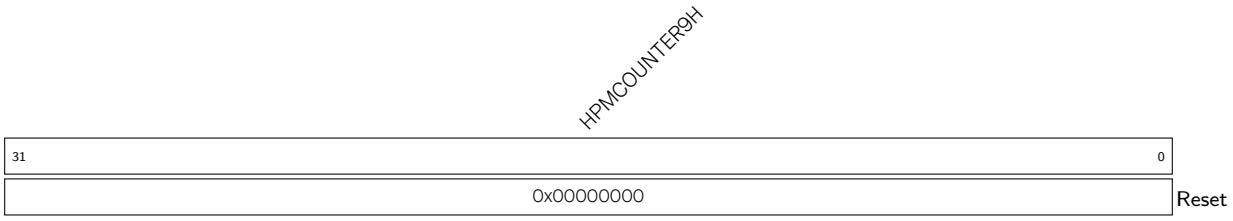
INSTRETH Read-only mirror of minstreth. (RO)

Register 2.60. hpmcounter8h (0xC88)



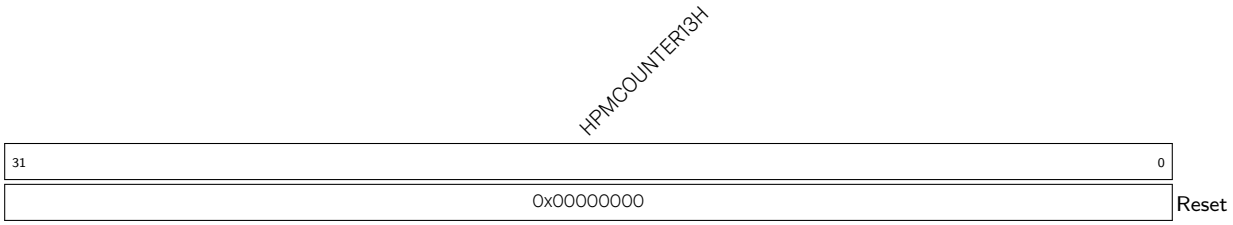
HPMCOUNTER8H Read-only mirror of mhpmcounter8h. (RO)

Register 2.61. hpmcounter9h (0xC89)



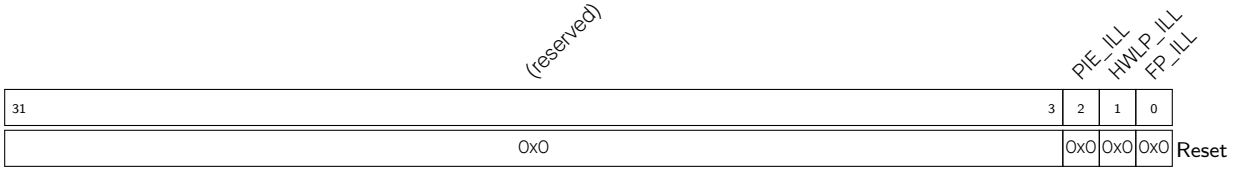
HPMCOUNTER9H Read-only mirror of mhpmcounter9h. (RO)

Register 2.62. hpmcounter13h (0xC8D)



HPMCOUNTER13H Read-only mirror of mhpmcounter13h. (RO)

Register 2.63. mext_ill (0x7F0)

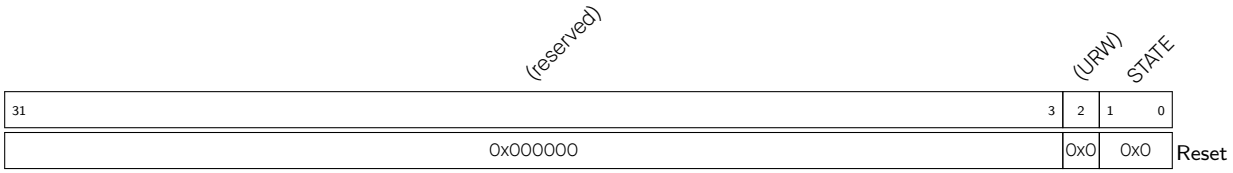


PIE_ILL Represents whether an illegal instruction exception occurred due to the CPU attempting to execute instructions belonging to the disabled PIE extension. (R/W)

HWLP_ILL Represents whether an illegal instruction exception occurred due to CPU attempting to execute instructions belonging to the disabled HWLP extension. (R/W)

FP_ILL Represents whether an illegal instruction exception occurred due to the CPU attempting to execute instructions belonging to the disabled RV32F extension, i.e. mstatus.FS=0x0. (R/W)

Register 2.64. mext_hwlp_status (0x7F1)



URW Represents whether to enable user-mode access for user-mode hardware loop CSRs.

0: Access is disabled

1: Access is enabled

(R/W)

STATE Configures state of the HWLP extension:

0x0: OFF (Accessing HWLP instructions/CSRs will trigger an illegal instruction exception)

0x1: INITIAL

0x2: CLEAN

0x3: DIRTY

Assuming state is not OFF, whenever HWLP CSRs are modified or HWLP instructions are executed the state will be updated to DIRTY. SW can set the state as INITIAL or CLEAN depending on the context. As specified in RISCV standard: When an extension's status is set to OFF, any instruction that attempts to read or write the corresponding state will cause an illegal instruction exception.

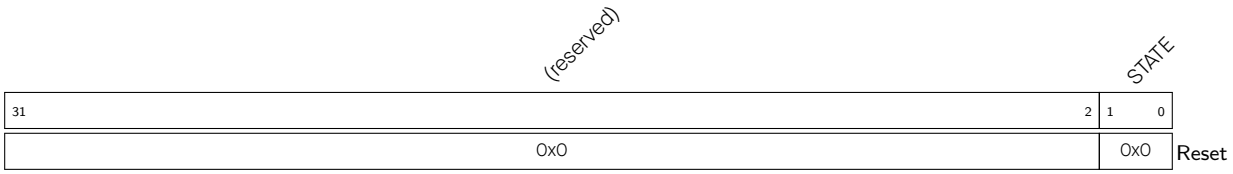
When the status is INITIAL, the corresponding state should have an initial constant value.

When the status is CLEAN, the corresponding state is potentially different from the initial value, but matches the last value stored on a context swap.

When the first iteration of HWLP is complete, the state changes to DIRTY. When the status is DIRTY, the corresponding state has potentially been modified since the last context save.

(R/W)

Register 2.65. mext_pie_status (0x7F2)



STATE Configures the state of the PIE extension:

0x0: OFF (Executing PIE instructions will trigger illegal instruction exception)

0x1: INITIAL

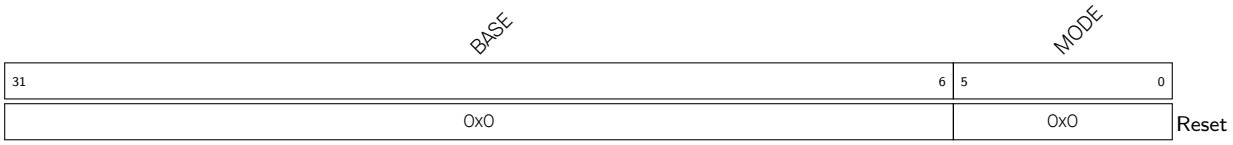
0x2: CLEAN

0x3: DIRTY

This is the extension state bits for the PIE extension, with similar functionality as the mstatus.FS bits of RV32F. Please refer to the "RISC-V Volume II - Privileged Architecture" for the standard interpretation of the STATE values.

(R/W)

Register 2.66. jvt (0x017)



BASE Represents jump table base address. (R/W)

MODE Represents configurable modes. This field is reserved to define the usage of this register. Currently, only one usage is defined, i.e., jump table. Other usages are reserved for future use).

0x0: The JTV_BASE is used as jump table

Others: reserved for future standard use

(R/W)

Register 2.67. mexstatus (0x7E1)



EXPT_VLD This bit is automatically set upon entering a trap handler.

This bit is only writable from debug mode.

(RO)

LOCKUP This bit is automatically set when double exceptions occur.

Upon lockup, the core will stall and wait for the debugger to resolve the lockup.

This bit is only writable from debug mode.

(RO)

BUS_ERR This bit is automatically set upon entering a load/store fault trap due to a bus error exception.

Upon entering the trap handler, this bit must be read to find if the load/store fault is due to a bus error.

When mexstatus.NMFT is enabled, this bit gets cleared automatically on executing MRET instruction.

When mexstatus.NMFT is disabled and mexstatus.PBEXE, this bit must be cleared manually before resuming execution, else it may re-trigger a pending bus error exception.

(R/W)

PBEXE Configures whether to enable the pending bus-error exception.

0: Disable

1: Enable

When mexstatus.PBEXE is enabled and mexstatus.NMFT is disabled, a synchronous exception is generated when mexstatus.BUS_ERR is set.

If mexstatus.NMFT is enabled, mexstatus.PBEXE has no effect.

Upon entering a trap the value of mexstatus.PBEXE is pushed into mexstatus.PPBEXE and mexstatus.PBEXE is set to 0, automatically.

Upon exiting a trap via MRET, the value of mexstatus.PPBEXE is popped back into mexstatus.PBEXE.

(R/W)

PPBEXE Configures whether to enable the previous pending bus-error exception.

0: Disable

1: Enable

Upon entering a trap, the value of mexstatus.PBEXE is pushed into mexstatus.PPBEXE.

Upon exiting a trap via MRET, the value of mexstatus.PPBEXE is popped back into mexstatus.PBEXE and mexstatus.PPBEXE is set to 1, automatically.

(R/W)

Continued on the next page...

Register 2.67. mexstatus (0x7E1)

Continued from the previous page...

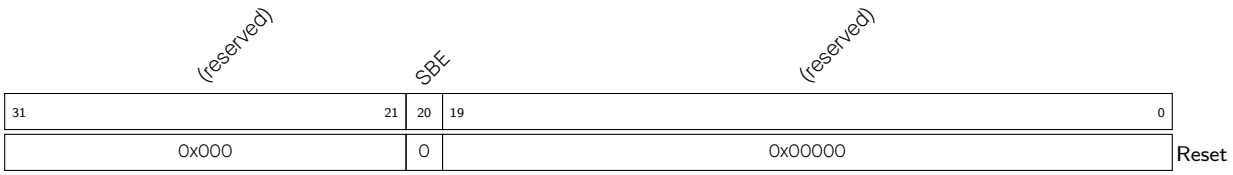
PMP_ERR This bit is automatically set upon entering load/store fault trap due to a PMP/PMA exception.
Upon entering the trap handler, this bit must be read to find if the load/store fault is due to PMP/PMA error.
This bit is cleared automatically upon MRET.
(R/W)

FETCH_ERR This bit is automatically set upon entering the fetch fault trap due to a bus-error exception.
Upon entering the trap handler, this bit must be read to find if the fetch fault is due to bus error.
This bit is cleared automatically upon MRET.
(R/W)

NMFT Configures whether to disable the memory fence on trap.
1: CPU will enter the trap handler upon exceptions/interrupts without waiting for pending memory load/store operations to finish (Default).
0: CPU will wait for pending memory load/store operations to finish before entering the trap handler.
(R/W)

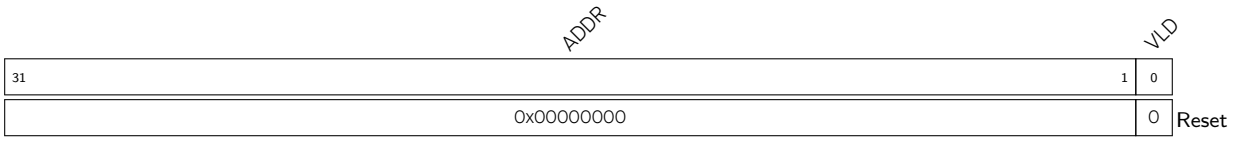
CLIC_INHV Configures whether to enable resuming faulting vector table fetch.
0: Disable
1: Enable (default). When it is set, mcause.minhv is used to decide if mepc holds an instruction address or a vector table address.
(R/W)

Register 2.68. mhint (0x7C5)



SBE Configures whether to enable the synchronous load/store bus-error exception.
0: Disable
1: Enable
(R/W)

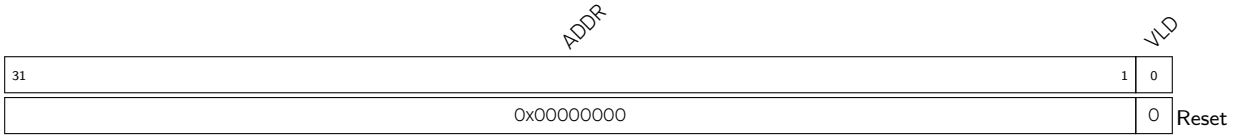
Register 2.69. ldpc0 (0xBE0)



VLD Configures whether the program-counter address value in this CSR is valid for a recent load bus-error exception.
0: Not valid
1: Valid
(R/W)

ADDR Configures the higher 31 bits of the half-word-aligned program counter corresponding to the same load bus-error exception. (R/W)

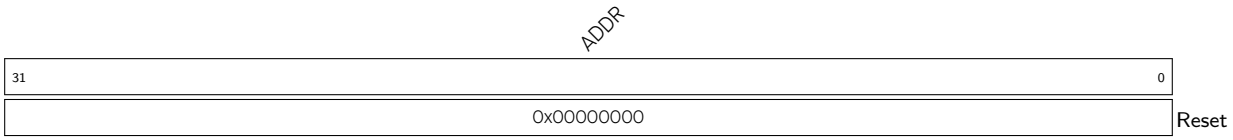
Register 2.70. ldpc1 (0xBE1)



VLD Configures whether the program-counter address value in this CSR is valid for a recent load bus-error exception.
0: Not valid
1: Valid
(R/W)

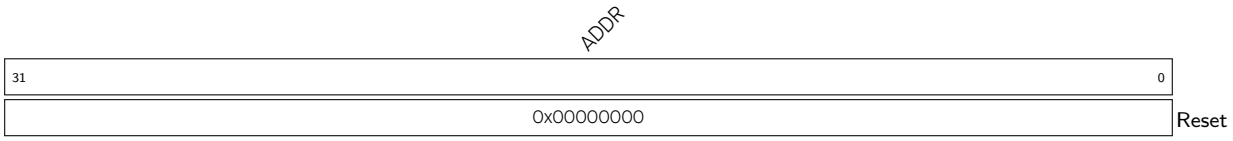
ADDR Configures the higher 31 bits of the half-word-aligned program counter corresponding to the same load bus-error exception. (R/W)

Register 2.71. ldtval0 (0xBE8)



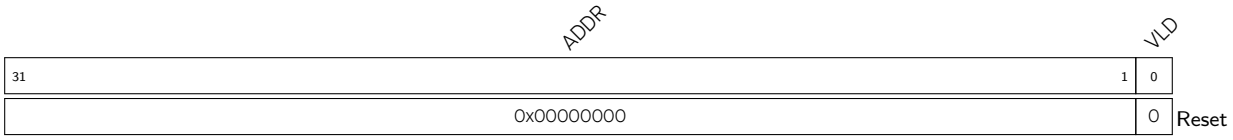
ADDR Configures the access address corresponding to the load bus-error exception recorded in ldpc0. (R/W)

Register 2.72. ldtval1 (0xBE9)



ADDR Configures the access address corresponding to the load bus-error exception recorded in ldpc1. (R/W)

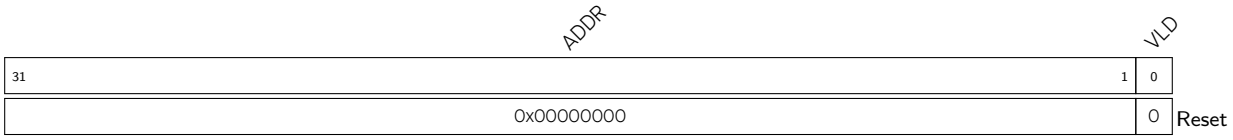
Register 2.73. stpc0 (0xBF0)



VLD Configures whether the program-counter address value in this CSR is valid for a recent store bus-error exception.
0: Not valid
1: Valid
(R/W)

ADDR Configures the higher 31 bits of the half-word-aligned program counter corresponding to the same store bus-error exception. (R/W)

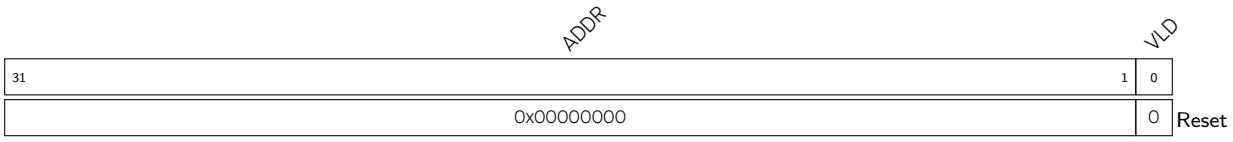
Register 2.74. stpc1 (0xBF1)



VLD Configures whether the program-counter address value in this CSR is valid for a recent store bus-error exception.
0: Not valid
1: Valid
(R/W)

ADDR Configures the higher 31 bits of the half-word-aligned program counter corresponding to the same store bus-error exception. (R/W)

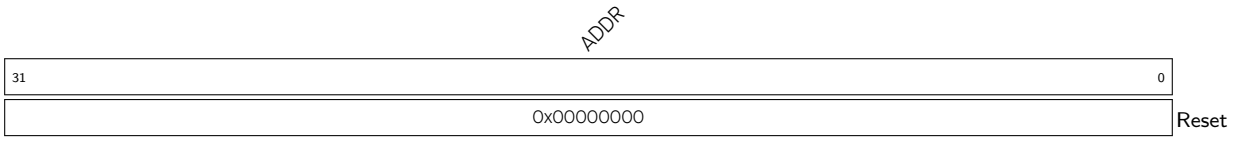
Register 2.75. stpc2 (0xBF2)



VLD Configures whether the program-counter address value in this CSR is valid for a recent store bus-error exception.
0: Not valid
1: Valid
(R/W)

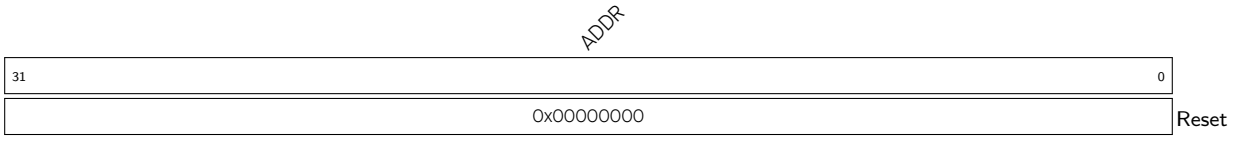
ADDR Configures the higher 31 bits of the half-word-aligned program counter corresponding to the same store bus-error exception. (R/W)

Register 2.76. sttval0 (0xBF8)



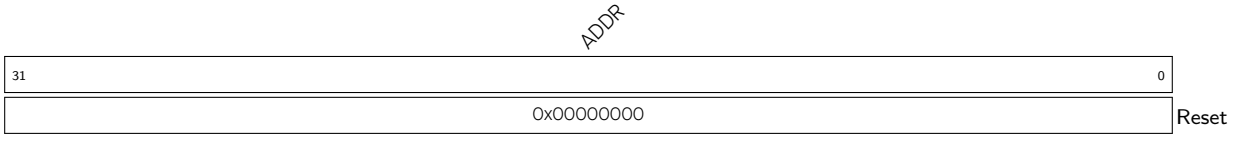
ADDR Configures the access address corresponding to the store bus-error exception recorded in stpc0. (R/W)

Register 2.77. sttval1 (0xBF9)



ADDR Configures the access address corresponding to the store bus-error exception recorded in stpc1. (R/W)

Register 2.78. sttval2 (0xBFA)



ADDR Configures the access address corresponding to the store bus-error exception recorded in stpc2. (R/W)

2.6 RISC-V Standard ISA Extensions Support

Each HP core implements mandatory base integer ISA and is compliant with version 2.0 of RV32I base integer instruction set specified in RISC-V Instruction Set Manual, Volume I: RISC-V User-Level ISA, Version 2.2.

2.6.1 M Extension

2.6.1.1 Overview

In addition to base RV32I instruction set, each HP core also implements standard integer multiplication and division extension and is fully compliant with version 2.0 of standard extension "M" specified RISC-V Instruction Set Manual, Volume I: RISC-V User-Level ISA, Version 2.2.

2.6.1.2 Functional Description

Instructions specified in "M" extension perform multiply and divide operations on the values held in two integer registers. Below are CPU cycle details taken by multiply and divide instructions in the absence of any data hazard in the pipeline.

- MUL operation takes 2 cycles
- DIV operation takes 1-19 cycles depending on operands

2.6.2 C Extension

2.6.2.1 Overview

Each HP core also implements RISC-V standard compressed instruction set extension named "C" and is fully compliant with version 2.0 of standard extension "C" specified in RISC-V Instruction Set Manual, Volume I: RISC-V User-Level ISA, Version 2.2. RVC is generic term used to specify C extension support. RVC reduces static and dynamic code size by adding short 16-bit instruction encoding for common operations. Typically, 50-60% of the RISC-V instructions in a program can be replaced with RVC instructions, resulting in a 25-35% code-size reduction.

2.6.2.2 Functional Description

RVC uses a simple compression scheme that offers a shorter 16-bit version of 32-bit RISC-V instructions for any of the below scenarios:

1. Immediate and address offset is small
2. One of the registers is a zero register (x0), the ABI lint register (x1), or the ABI stack pointer (x2)
3. Destination register and the first source register are identical, or
4. The registers used are the 8 most popular ones i.e. x8-x15.

2.6.3 F Extension

2.6.3.1 Overview

Each HP core implements the RISC-V standard single-precision floating-point instruction set extension, "RV32F" extension in short. The RV32F extension also provides its own register file and various CSRs for configuration and flag checking. Please refer to the RISC-V Instruction Set Manual, Volume I: RISC-V User-Level ISA, Version 2.2, F Standard Extension Version 2.0" for complete information.

2.6.3.2 Functional Description

- Various instructions to perform floating-point arithmetic operations, conversion/move operations to/from integer format, comparison operations, classification operations, and memory load/store operations.
- Single-precision 32-bit floating-point operations compliant with IEEE 754-2008 standard.
- Separate 32-bit wide floating-point register file with 32 registers.
- Configurable rounding modes.
- Flags for checking inexact, overflow, underflow, invalid and divide by zero outcomes.
- Subnormal and NaN (not a number) arithmetic
- Status bits to control the availability of the extension and track its usage in the context of a program execution.

2.6.3.3 Initialization

The RV32F extension is disabled by default and must be manually enabled using the mstatus.FS bits. Otherwise, an illegal instruction exception will be generated upon executing any of the instructions, or accessing any of the CSRs, which are part of the RV32F extension.

Please refer to the RISC-V Instruction Set Manual, Volume II: Privileged Architecture, Version 1.10, Extension Context Status in mstatus Register for complete information.

2.6.4 Zc (Z) Extension

2.6.4.1 Overview

Zc extension is useful for code size reduction. Each HP core implements this extension compatible with Zc-specification v1.0.4-3. The Zc-specification has multiple subset extensions, and the ESP32-P4HP CPU has implemented Zcb, Zcmp, and Zcmt extensions.

2.6.4.2 Functional Description

Zcb Extension

The Zcb extension depends on the C extension and provides more compressed instructions for the following scenarios:

- Immediate and address offset is small.
- Destination register and the first source register are identical.
- The registers used are the 8 most popular ones (x8-x15).

Zcmp Extension

The Zcmp extension is a set of instructions which may be executed as a series of existing 32-bit RISC-V instructions.

- PUSH, POP, POPRET operations used to reduce the size of function prologues and epilogues
  - The PUSH operation
    - adjusts the stack pointer to create the stack frame
    - pushes (stores) the registers specified in the register list to the stack frame
  - The POP operation
    - pops (loads) the registers in the register list from the stack frame
    - adjusts the stack pointer to destroy the stack frame
  - The POPRET operation
    - pops (loads) the register in the register list from the stack frame
    - cm.popretz also moves zero into a0 as the return value
    - adjusts the stack pointer to destroy the stack frame
    - executes a ret instruction to return from the function
- Move two a0-a1 into two registers of s0-s7
- Move two s0-s7 into a0-a1

Zcmt Extension

The Zcmt extension is also referred to as table jump. It reads the jump target address from the jump table and then jumps to it.

Table jump uses a 256-entry 32-bit wide table in instruction memory to contain function address. The table must be a minimum of 64-byte aligned. It is used as a form of dictionary compression to reduce the code size of jal, auipc+jalr, jr, auipc+jr instructions.

Table jump allows the linker to replace the following instruction sequences with a cm.jalt encoding, and an entry in the table:

- 32-bit j calls
- 32-bit jal ra calls

2.6.4.3 Zc Instructions

Instruction	Mnemonic	Description
Zcb extension		
c.lbu	c.lbu rd', uimm(rs1')	Loads a byte from the memory address formed by adding rs1' to the zero-extended immediate uimm. The resulting byte is zero-extended to 32 bits and is written to rd'
c.lhu	c.lhu rd', uimm(rs1')	Loads a halfword from the memory address formed by adding rs1' to the zero-extended immediate uimm. The resulting byte is zero-extended to 32 bits and is written to rd'
c.lh	c.lh rd', uimm(rs1')	Loads a halfword from the memory address formed by adding rs1' to the zero-extended immediate uimm. The resulting byte is sign-extended to 32 bits and is written to rd'
c.sb	c.sb rs2', uimm(rs1')	Stores the least significant byte of rs2' to the memory address formed by adding rs1' to the zero-extended immediate uimm
c.sh	c.sh rs2', uimm(rs1')	Stores the least significant halfword of rs2' to the memory address formed by adding rs1' to the zero-extended immediate uimm
c.zext.b	c.zext.b rsd'	Zero-extends the least significant byte of the operand to 32 bits by inserting zeros into all of the bits more significant than 7
c.sext.b	c.sext.b rsd'	Sign-extends the least significant byte of the operand to 32 bits by copying the most significant bit in the byte to all the more significant bits
c.zext.h	c.zext.h rsd'	Zero-extends the least significant halfword of the operand to 32 bits by inserting zeros into all of the bits more significant than 15
c.sext.h	c.sext.h rsd'	Sign-extends the least significant halfword of the operand to 32 bits by copying the most significant bit in the halfword to all the more significant bits
c.not	c.not rsd'	Takes the one's complement of rd'/rs1' and writes the result to the same register
c.mul	c.mul rsd'	Multiplies 32 bits of the source operands from rsd' and rs2' and writes the lowest 32 bits to the result to rsd'
Zcmp extension		
cm.push	cm.push {reg_list}, -stack_adj	Stores the registers in reg_list to the memory below the stack pointer, and then creates the stack frame by decrementing the stack pointer by stack_adj, including any additional stack space requested by the value of spimm

Instruction	Mnemonic	Description
cm.pop	cm.pop {reg_list}, stack_adj	Loads the registers in reg_list from stack memory, and then adjusts the stack pointer by stack_adj
cm.popret	cm.popret {reg_list}, stack_adj	Loads the registers in reg_list from stack memory, adjusts the stack pointer by stack_adj, then return to ra
cm.popretz	cm.popretz {reg_list}, stack_adj	Loads the registers in reg_list from stack memory, adjusts the stack pointer by stack_adj, moves zero to a0 and then return to ra
cm.mva01s	cm.mva01s rs1', rs2'	Moves r1s' into a0 and r2s' into a1
cm.mva01	cm.mva01 r1s', r2s'	Moves a0 into r1s' and a1 into r2s'
Zcmt extension		
cm.jt	cm.jt index	Reads an entry from the jump vector table in memory and jumps to the address that was read
cm.jalt	cm.jalt index	Reads an entry from the jump vector table in memory and jumps to the address that was read, linking to ra

For more details about the instructions please refer to Zc-specification v1.0.4-3.

2.6.4.4 Limitations

jvt adds architecture state to the context. Therefore, it must be saved and restored on context switches.

2.6.5 Bit Manipulation (Zb) Extension

2.6.5.1 Overview

The bit-manipulation (bitmanip) extension, also known as the Zb extension set, consists of several component extensions to the base RISC-V architecture. These extensions aim to improve code density, performance, and energy efficiency. The HP core fully supports the Zb extension. For Zb extension specifications, please refer to Zb-specification v1.0.0.

2.6.5.2 Functional Description

Although the bitmanip instructions are designed for general-purpose use, certain instructions are more applicable to specific domains. Hence, the bitmanip extension set is divided into several smaller extensions instead of a single large one. Each extension has its own Zb*-extension name, and comprises instructions that share related functionality, use cases, and often the same implementation logic. Some instructions are available in only one extension, while others are available in several. The instructions have mnemonics and encodings that are independent of the extensions in which they appear. Thus, when implementing extensions with overlapping instructions, there is no redundancy in logic or encoding.

2.6.6 A Extension

2.6.6.1 Overview

Support for atomic (A) extension is available in compliance with RISC-V Instruction Set Manual, Volume I: RISC-V User-Level ISA, Version 2.2, with an emphasis to guarantee forward progress, i.e. any situation that may cause data memory lock for an indefinite amount of time is prevented by their very functionality.

2.6.6.2 Functional Description

Atomic (A) Extension contains instructions that atomically read-modify-write memory to support synchronization between multiple RISC-V harts running in the same memory space. The two forms of atomic instruction provided are load-reserved/store-conditional instructions and atomic fetch-and-op memory instructions. The atomic instructions currently ignore the aq (acquire) and rl (release) bits as they are irrelevant to the current architecture, in which memory ordering is always guaranteed.

2.6.6.3 Load-Reserve (LR.W) Instruction

The LR.W instruction simply locks a 32-bit aligned memory address to which the load access is being performed. Once a 4-byte memory region is locked, it will remain locked, i.e. other harts won't be able to access this same memory location, until any of the following scenarios is encountered during execution:

- any load operation
- any store operation
- any interrupts/exceptions
- backward jump/taken backward branch
- JALR
- ECALL/EBREAK/MRET/URET
- FENCE/FENCE.I
- debug mode
- critical section exceeding 64 bytes
- data address in SC.W instruction not matching that in LR.W instruction

If any of the above happens, except SC.W, the memory lock will be released immediately. If an SC instruction is encountered instead, the lock will be released eventually (not immediately) in the manner described in Section 2.6.6.4.

If a misaligned address is encountered, it will cause an exception with mcause = 6.

2.6.6.4 Store-Conditional (SC.W) Instruction

The SC.W instruction first checks if the memory lock is still valid, and the address is the same as specified during the last LR.W instruction. If so, only then will it perform the store to memory, and later release the lock as soon as it gets an acknowledgement of operation completion from the memory.

On the other hand, if the lock is found to have been invalidated (due to any of the situations as described in Section 2.6.6.3), it will set a fail code (currently always 1) in the destination register rd.

If a misaligned address is encountered, it will cause an exception with mcause = 6.

2.6.6.5 AMO Instructions

An AMO instruction executes in three steps:

1. Read data from memory address given by rs1, and save it to destination register rd.
2. Combine the data in rd and rs2 according to the operation type and keep the result for Step 3 below.
3. Write the result obtained in Step 2 above to memory address given by rs1.

There are nine different AMO operations: SWAP, ADD, AND, OR, XOR, MAX, MIN, MAXU, and MINU.

During this whole process, the memory address is kept locked from being accessed by other harts. If a misaligned address is encountered, it will cause an exception with mcause = 6.

For AMO operations both load and store access faults (PMP/PMA) are checked in the 1st step itself. For such cases mcause = 7.

2.7 Custom Instruction Extensions

2.7.1 Hardware Loop

2.7.1.1 Overview

HWLP (hardware loop) is the implementation of "For" loop in hardware. A loop is configured with a start and an end address and the number of iterations before its execution. When instruction execution encounters the start address, the loop execution starts. When the end address is executed, it is considered as one loop count and the start address is fetched again. This procedure is repeated until the loop count is equal to the number of iterations configured.

2.7.1.2 Features

HWLP has the following features:

- Execution with no extra penalty compared to conditional branch instructions
- Support for two nested HWLPs
- Configurable using a single instruction (lp.setupi) in case of end address within 1 KB from the configuration instruction

2.7.1.3 Functional Description

The configuration of HWLP is done either through CSRs or through custom instructions of HWLP, which will be described in detail in subsequent sections. Any HWLP needs to be configured with a start, an end address, and an iteration count.

When a single loop is to be set up, either Loop0 or Loop1 CSRs can be configured. Before setting up a HWLP, configure STATE bits in mhwloop_state_reg CSR to any value other than 0 (OFF, default state). Otherwise, accessing HWLP configuration CSRs will cause an illegal instruction exception. To set up and execute HWLP in user mode, URW bit in mhwloop_state_reg CSR should be set to 1.

The core supports a maximum of two HWLPs, which can be executed in nested fashion. Each loop is configured separately in this case. Note that, the inner loop has higher priority than the outer loop in case the end instruction of both loops is common. Hardware Loop0 has to be considered as the inner loop and Loop1 as the outer loop while configuring nested loops, as Loop0 has a higher priority than Loop1 in hardware implementation.

The advantage of a HWLP over a conditional branch instruction is that it has zero penalty for its execution. Branch/JUMP instructions use prediction logic for evaluating branch condition and branch/jump address. If a miss-prediction happens, the core pipeline will be flushed and the correct address will be fetched, and thus a few cycles have already been wasted in fetch of the incorrect instructions before flush. As the HWLP does not need prediction of result or target address of conditional branch, it has no penalty, and thus has a higher throughput.

2.7.1.4 Instructions/Operations/Modes Supported in HWLP

- Load type instructions
- Store type instructions

- Arithmetic, logical, AIA instructions
- Zc-extension instructions
- Any interrupts/exceptions within the HWLP
- ECALL/EBREAK
- FENCE/FENCE.I/CSRx instructions
- Atomic instructions
- Debug mode
- Data address in SC.W instruction not matching that in LR.W instruction

2.7.1.5 HWLP Constraints

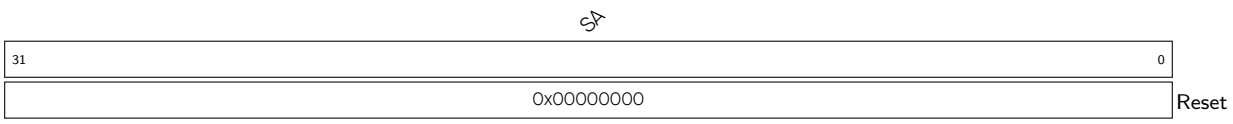
Please note that branch or jump instructions at the end of HWLP may cause unexpected behavior.

2.7.1.6 Register Summary

Name	Description	Address	Access
mhwloop0_start_addr	Machine Loop0 start address	0x7C6	R/W
mhwloop0_end_addr	Machine Loop0 end address	0x7C7	R/W
mhwloop0_count	Machine Loop0 count	0x7C8	R/W
mhwloop1_start_addr	Machine Loop1 start address	0x7C9	R/W
mhwloop1_end_addr	Machine Loop1 end address	0x7CA	R/W
mhwloop1_count	Machine Loop1 count	0x7CB	R/W
mext_ill_reg	Machine extension illegal register	0x7F0	R/W
mhwloop_state_reg	Machine HWLP state register	0x7F1	R/W
uhwloop0_start_addr	User Loop0 start address	0x8C0	R/W
uhwloop0_end_addr	User Loop0 end address	0x8C1	R/W
uhwloop0_count	User Loop0 count	0x8C2	R/W
uhwloop1_start_addr	User Loop1 start address	0x8C3	R/W
uhwloop1_end_addr	User Loop1 end address	0x8C4	R/W
uhwloop1_count	User Loop1 count	0x8C5	R/W
uhwloop_state_reg	User HWLP state register	0x8C6	R/W

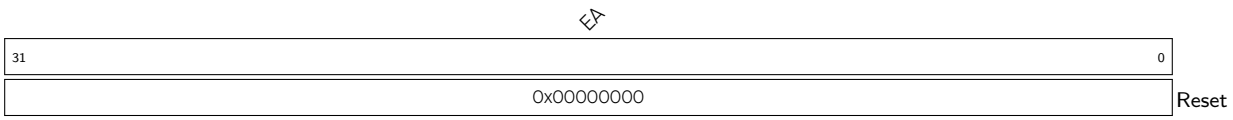
2.7.1.7 Register Description

Register 2.79. mhwloop0_start_addr (0x7C6)



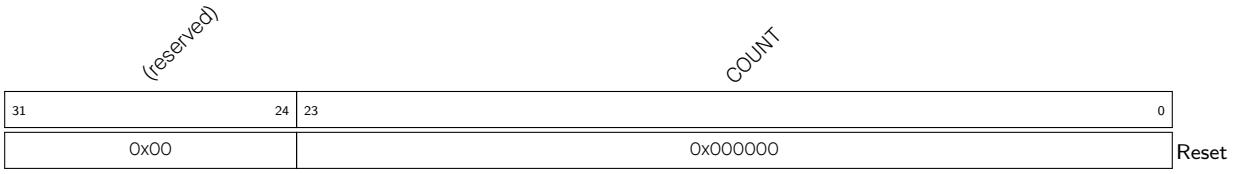
SA Configures the 32-bit address of the first instruction of HW Loop0. Valid when mhwloop_state_reg.STATE is not 0x0, that is, not OFF. (R/W)

Register 2.80. mhwloop0_end_addr (0x7C7)



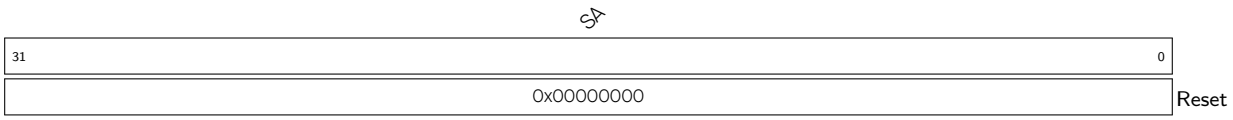
EA Configures the 32-bit address of the last instruction of HW Loop0. Valid when mhwloop_state_reg.STATE is not 0x0, that is, not OFF. (R/W)

Register 2.81. mhwloop0_count (0x7C8)



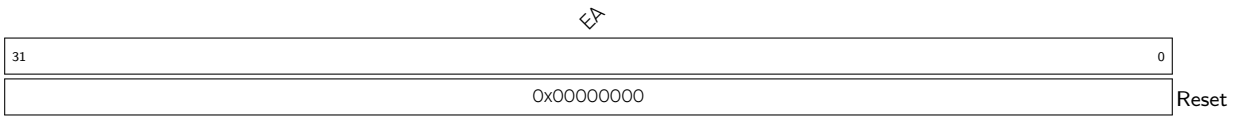
COUNT Configures the number of iterations of HW Loop0. Valid when mhwloop_state_reg.STATE is not 0x0, that is, not OFF. (R/W)

Register 2.82. mhwloop1_start_addr (0x7C9)



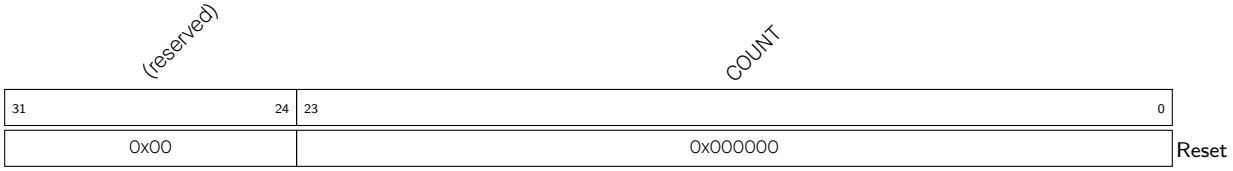
SA Configures the 32-bit address of the first instruction of HW Loop1. Valid when mhwloop_state_reg.STATE is not 0x0, that is, not OFF. (R/W)

Register 2.83. mhwloop1_end_addr (0x7CA)



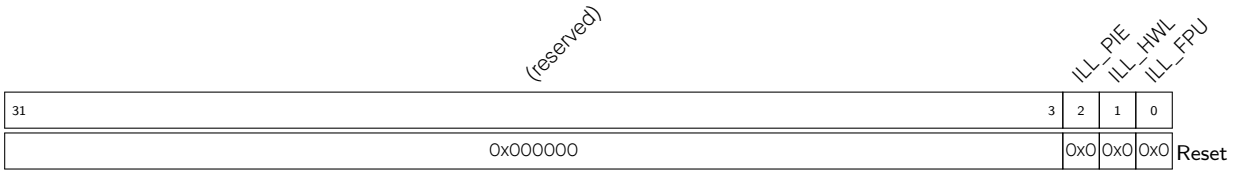
EA Configures the 32-bit address of the last instruction of HW Loop1. Valid when mhwloop_state_reg.STATE is not 0x0, that is, not OFF. (R/W)

Register 2.84. mhwloop1_count (0x7CB)



COUNT Configures the number of iterations of HW Loop1. Valid when mhwloop_state_reg.STATE is not 0x0, that is, not OFF. (R/W)

Register 2.85. mext_ill_reg (0x7F0)



ILL_FPU Represents whether an illegal instruction is triggered due to the disabled F extension.

0: Not triggered

1: Triggered

(RO)

ILL_HWL Represents whether the illegal instruction is triggered due to the disabled HWLP extension.

0: Not triggered

1: Triggered

(RO)

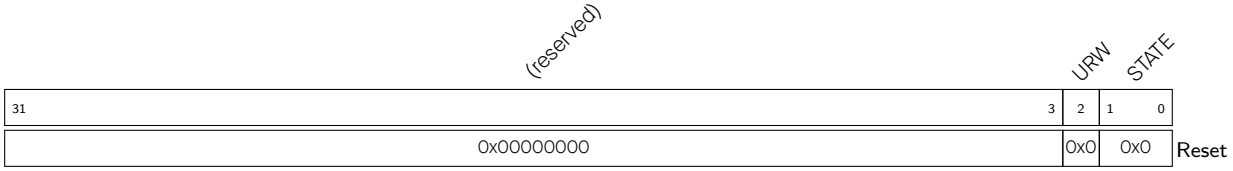
ILL_PIE Represents whether the illegal instruction is triggered due to the disabled PIE extension.

0: Not triggered

1: Triggered

(RO)

Register 2.86. mhwloop_state_reg (0x7F1)



STATE Configures the state of the HWLP extension.

0x0: OFF (Accessing HWLP instructions and CSRs will trigger an illegal instruction exception)

0x1: INITIAL

0x2: CLEAN

0x3: DIRTY

(R/W)

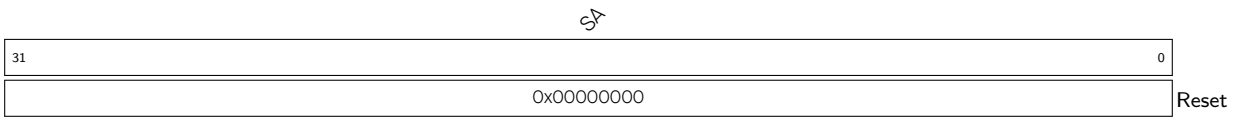
URW Configures user-mode access for user-mode HWLOOP CSRs.

0: User-mode access is disabled

1: User-mode access is enabled

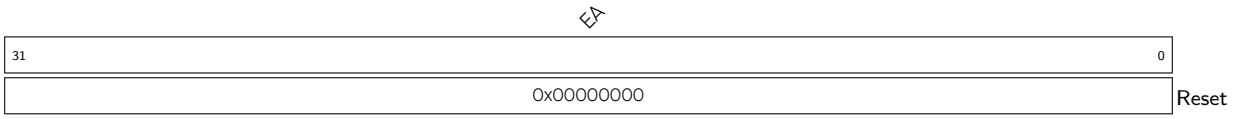
(R/W)

Register 2.87. uhwloop0_start_addr (0x8C0)



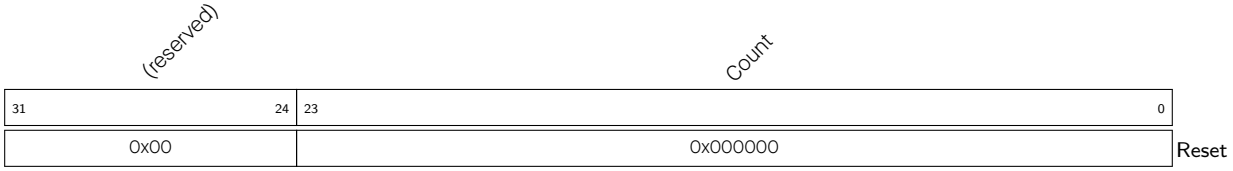
SA Configures the 32-bit address of the first instruction of HW Loop0 in user mode. Valid when mhwloop_state_reg.STATE is not 0x0, that is, not OFF. It is read-writable only when mhwloop_state_reg.URW is 1. (R/W)

Register 2.88. uhwloop0_end_addr (0x8C1)



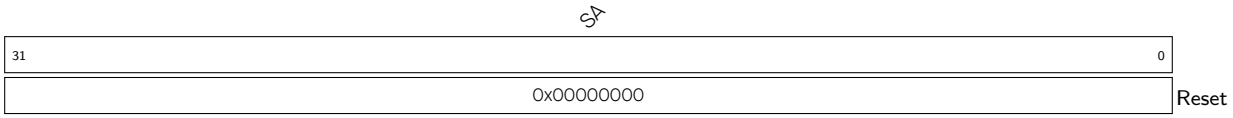
EA Configures the 32-bit address of the last instruction of HW Loop0 in user mode. Valid when mhwloop_state_reg.STATE is not 0x0, that is, not OFF. It is read-writable only when mhwloop_state_reg.URW is 1. (R/W)

Register 2.89. uhwloop0_count (0x8C2)



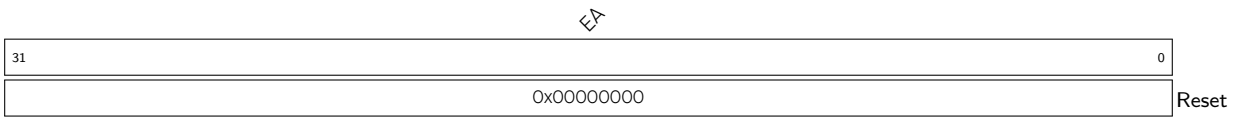
Count Configures the number of iterations of HW Loop0 in user mode. Valid when mhwloop_state_reg.STATE is not 0x0, that is, not OFF. It is read-writable only when mhwloop_state_reg.URW is 1. (R/W)

Register 2.90. uhwloop1_start_addr (0x8C3)



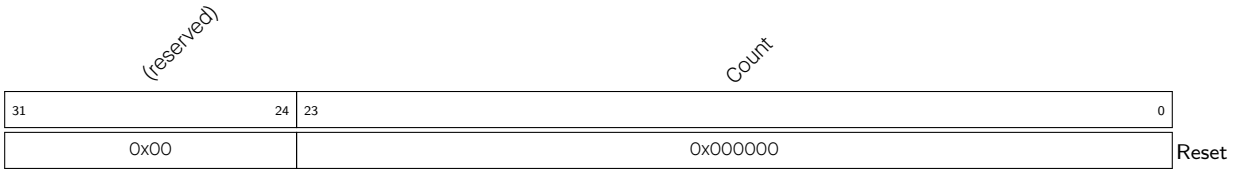
SA Configures the 32-bit address of the first instruction of HW Loop1 in user mode. Valid when mhwloop_state_reg.STATE is not 0x0, that is, not OFF. It is read-writable only when mhwloop_state_reg.URW is 1. (R/W)

Register 2.91. uhwloop1_end_addr (0x8C4)



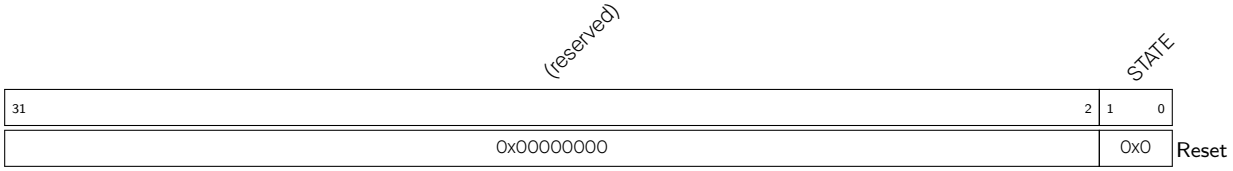
EA Configures the 32-bit address of the last instruction of HW Loop1 in user mode. Valid when mhwloop_state_reg.STATE is not 0x0, that is, not OFF. It is read-writable only when mhwloop_state_reg.URW is 1. (R/W)

Register 2.92. uhwloop1_count (0x8C5)



Count Configures the number of iterations of HW Loop1 in user mode. Valid when mhwloop_state_reg.STATE is not 0x0, that is, not OFF. It is read-writable only when mhwloop_state_reg.URW is 1. (R/W)

Register 2.93. uhwloop_state_reg (0x8C6)



STATE Configures the state of the HWLP extension in user mode.

0x0: OFF (Accessing HWLP instructions and CSRs will trigger an illegal instruction exception)

0x1: INITIAL

0x2: CLEAN

0x3: DIRTY

Bits are physically the same as mhwloop_state_reg. Ensure the permission is enabled in mhwloop_state_reg.URW field.

Assuming state is not OFF, whenever HWLP CSRs are modified, or HWLP instructions are executed, the state will be updated to DIRTY.

(R/W)

2.7.1.8 HWLP Instructions

The table below contains the HWLP instruction set.

Table 2.7-2. HWLP Instructions

SN	Instruction Mnemonic	Description	31:20	19:15	14:12	11:8	7	6:0
1	lp.setupi L, uimm_Count, uimm_EndOffset	StartAddr = PC+4 EndAddr = PC+uimm_EndOffset Count = uimm_Count	uimm_Count [11:0]	uimm_EndOffset [5:1]	101	uimm_EndOffset [9:6]	L (index)	0101011
2	lp.setup L, rs1, uimm_EndOffset	StartAddr = PC+4 EndAddr = PC+uimm_EndOffset Count = GPR[rs1]	uimm_EndOffset [12:1]	rs1 [4:0] (count)	100	X	L (index)	0101011
3	lp.setup.beqz L, rs1, EndInstrSize, uimm_EndOffset	StartAddr = PC+4 EndAddr = PC+uimm_EndOffset Count = GPR[rs1] Size of HWLP end instruction = 32 bits if EndInstrSize is 1, 16 bits if EndInstrSize is 0	uimm_EndOffset [12:1]	rs1 [4:0] (count)	100	Size of end instruction (0000 for 16-bit instruction and 0010 for 32-bit instruction)	L (index)	0101011
4	lp.starti L, uimm_StartOffset	StartAddr = PC+uimm_StartOffset	uimm_StartOffset [12:1]	X	000	X	L (index)	0101011
5	lp.endi L, uimm_EndOffset	EndAddr = PC+uimm_EndOffset	uimm_EndOffset [12:1]	X	001	X	L (index)	0101011
6	lp.count L, rs1	Count = GPR[rs1]	X	rs1 [4:0] (count)	010	X	L (index)	0101011
7	lp.counti L, uimm_Count	Count = uimm_Count	uimm_Count [11:0]	X	011	X	L (index)	0101011

GoBack

2.7.2 Processor Instruction Extension

2.7.2.1 Overview

The Processor Instruction Extension (PIE) ISA entails various custom 128-bit wide SIMD (Single Instruction Multiple Data) and complex instructions for accelerating edge inference and digital signal processing applications. These instructions are fully integrated into the CPU pipeline for seamless use with the standard RV32IMAFC ISA with which the HP core is compliant. PIE Extension support is compliant to Espressif's PIE V2.2.0 version.

2.7.2.2 Functional Description

- 8 x 128-bit wide integer vector register files
- Each 128-bit wide vector register can be used as 16 x 8-bit elements, 8 x 16-bit elements, or 4 x 32-bit elements for SIMD operations
- Up to 128-bit wide memory load/store operations directly to/from the vector registers
- Fused instructions for parallel execution of arithmetic and memory operations
- Vector and scalar MAC units capable of operating on 16 x 8-bit elements or 8 x 16-bit elements per cycle
- 512-bit wide vector register for accumulating results of 8/16-bit vector MAC operations
- 40-bit wide scalar register for accumulating results of 8/16-bit scalar MAC operations
- Instructions for parallel 16-bit FFT/complex math operations
- Final results are generated after rounding and saturation
- Configurable rounding modes and scaling (arithmetic right shift) parameter

For a complete description of all the available instructions and configurations, please refer to Chapter 4 *Processor Instruction Extensions (PIE)*.

2.7.2.3 Initialization, Context Switching, and Exceptions

The PIE extension is disabled by default and must be manually enabled using the mext_pie_status.STATE field. Otherwise, an illegal instruction exception will be generated upon executing any of the instructions that are part of the PIE extension.

To indicate that an illegal instruction exception, i.e. exception with mcause = 0x2, has occurred due to disabled PIE extension, the HP core will set the mext_ill.PIE_ILL bit.

When the PIE extension is enabled, i.e. mext_pie_status.STATE bits are non-zero, the execution of any PIE instructions would always cause the mext_pie_status.STATE bits to change to DIRTY (11).

To enable the PIE extension, follow the steps below:

1. Set the mext_pie_status.STATE bits to the INITIAL (01) state.
2. Use the PIE instructions to initialize the PIE register file and accumulator registers to zero, which will cause the mext_pie_status.STATE bits to change to DIRTY (11) automatically.
3. Since the value of the PIE registers is known at this point, set the mext_pie_status.STATE bits to CLEAN (10).

4. Execute the application program.
5. During execution of the application code, if any PIE instruction is encountered, the mext_pie_status.STATE bits will automatically change to DIRTY (11).

During context switching, whether to save/restore the PIE extension registers can be decided based on the value of the mext_pie_status.STATE bits. If the value is DIRTY, the state of the PIE register must be saved to the stack. After that, the state would be updated to CLEAN or OFF, depending on whether the next context (thread) has the PIE extension enabled or not.

Some PIE instructions have constraints on the register fields or immediate fields. e.g. for fused instructions which are performing load and arithmetic, with the same destination register for the two parallel operations, it will be considered ambiguous and thus cause a PIE illegal exception with mcause = 0x1f.

2.8 Memory-Mapped Registers

2.8.1 Overview

The tightly-coupled memory-mapped registers below are present within the core complex:

- CLIC registers for each HP core
- Shared CLINT registers

2.8.2 Features

Each HP CPU core supports:

- Access to the other HP core's CLIC registers
- Bus-error exception generation on invalid address access
- Access to CLINT registers. However, only Core 0 can modify CLINT timer value

2.8.3 Functional Description

The table below shows the address decoding used by each HP core to access CLIC and CLINT address range.

Table 2.8-1. Address Decoding for Memory-Mapped Register Access

Bits	Description
31:28	Fixed as 0x2, i.e. base address
27:20	Identifies the sub module: 0x0: CLINT 0x08: CLIC 0xB: CLIC user Others: Reserved
19:16	Controls access type 0x0: Access own registers 0x1: Access the other core's registers
15:0	Register offset

Table 2.8-2. Internal Memory Address Map

Block	Start Address	End Address
CLINT (Self)	0x20000000	0x2000FFFF
CLIC (Self)	0x20800000	0x2080FFFF
CLINT (Other Core)	0x20010000	0x2001FFFF
CLIC (Other Core)	0x20810000	0x2081FFFF

2.9 Interrupt Controller

2.9.1 Overview

Each HP core uses the CLIC + CLINT scheme for implementing and supporting interrupts.

CLINT stands for core-local interrupts, which are sources local to the HP core subsystem for generating timer and software interrupts for each HP core.

CLIC stands for core-local interrupt controller, which provides low-latency, vectored, pre-emptive interrupts for each HP core. Each HP core CLIC unit supports 32 external interrupts and 2 CLINT interrupts.

2.9.2 CLIC

2.9.2.1 Overview

The HP core CLIC implementation follows the official specification: Core-Local Interrupt Controller (CLIC) RISC-V Privileged Architecture Extensions (10/10/2023).

Only the CLIC mode of operation is supported by the HP core, i.e. mtvec.MODE is hardwired to 0x3. Hence, the basic RISC-V interrupt handling scheme and the associated CSRs (such as mie, mip, mideleg, uie, and uip) are unavailable.

HP core supports machine mode interrupt as well as user mode interrupt delegation.

2.9.2.2 CLIC CSRs

The following machine mode CSRs are relevant for interrupt handling in the CLIC scheme:

- mstatus
- mtvec
- mtvt
- mscratch
- mepc
- mcause
- mnxti
- mintthresh
- mintstatus
- mclicbase
- mscratchcsw
- mscratchcswl

As mentioned earlier, the CLIC mode is decided by the mtvec.MODE field. In each HP core, these bits are hardwired to 0x3, indicating that only the CLIC mode is supported.

In CLIC mode, the global interrupt enable for machine mode is still dictated by the mstatus.MIE bit.

interrupt can be either vectored or non-vectored depending on the clicintattr[i].SHV bit.

For a non-vectored interrupt, the HP core jumps to the trap handler base address as configured in mtvec.BASE field.

For a vectored interrupt, the HP core jumps to the 32-bit word address stored at the corresponding index in the vector table. For example, for the "i"th machine mode interrupt, first the 32-bit address stored at the memory location "mtvt + 4*i" will be loaded, after which the HP core will treat the loaded data as an address and jump to it. The base address of the vector table is configured in mtvt.

Upon taking an interrupt, the HP core will store the address of the instruction that was interrupted in mepc. Also, the HP core will update the value of mcause based on the interrupt's ID.

2.9.2.3 Interrupt Mapping

CLIC in each HP core supports 32 external interrupts and 2 core-local interrupts.

The IDs 16-47 are mapped to the external interrupts, while IDs 15 and below are reserved for the core-local interrupts.

Table 2.9-1. CLIC Interrupt Mapping

ID	Description
0-2	NA (Reserved)
3	CLINT M mode software interrupt
4-6	NA (Reserved)
7	CLINT M mode timer interrupt
8-15	NA (Reserved)
16	External interrupt 0
17	External interrupt 1
18	External interrupt 2
...	...
45	External interrupt 29
46	External interrupt 30
47	External interrupt 31

For detailed description of the core-local interrupt sources, please refer to Section 2.9.3.

The mapping of the 32 external interrupts to the interrupt sources in SoC is managed using the Interrupt Matrix. Please refer to Chapter 13 *Interrupt Matrix* for details on how to configure and map to SoC interrupts to CLIC interrupts for each HP core.

2.9.2.4 CLIC Parameters

The available features of a CLIC implementation can be discovered using the registers mclicbase and clicinfo:

- mclicbase is a read-only CSR which holds the base address of the CLIC memory-mapped registers for machine mode.
- clicinfo is a memory-mapped read-only register. It consists of the following fields:
  - NUM_INTERRUPTS: Represents the number of interrupts supported by the current CLIC implementation
  - CLICINTCTLBITS: Represents the number of programmable higher bits in clicintctl[i] and mintthresh.TH for encoding the level and priority of each interrupt
- Once the value of the mclicbase is known, the full address of the clicinfo can be computed by adding the corresponding address offset of 0x0004 (shown in Section 2.9.2.6) to the mclicbase value.

2.9.2.5 Interrupt Level and Priority Encoding

Table 2.9-2 shows the possible ways that level and priority can be specified for CLIC interrupts, based on the configured values of mcliccfg.MNLBITS and clicintctl[i] registers, and the fixed value of the CLICINTCTLBITS

parameter (0x3 for this CLIC implementation).

Table 2.9-2. CLIC Level and Priority Encoding with CLICINTCTLBITS=3

mnlbits[3:0]	clicintctl[i][7:0]	Interrupt Levels	Interrupt Priorities
0	`ppp.....`	255	31, 63, 95, 127, 159, 191, 223, 255
1	`lpp.....`	127, 255	31, 63, 95, 127
2	`llp.....`	63, 127, 191, 255	31, 63
3-8	`lll.....`	31, 63, 95, 127, 159, 191, 223, 255	31

mcliccfg.MNLBITS decides the number of higher bits in the clicintctl[i] registers which are to be used to encode the level (indicated as `'l'` in the above table), while the remaining lower bits are used to encode the priority (indicated as `'p'` in the above table). Since only the higher bits of clicintctl[i] are writeable as per the CLICINTCTLBITS parameter, hence the remaining lower bits are tied to 1 (indicated as '.' in the above table).

Level and priority are both used to give preference to one interrupt over the other. A higher level interrupt can preempt a lower level interrupt (assuming that mstatus.MIE is manually enabled inside an interrupt handler for allowing nesting), but in the case of two interrupts with same level and different priorities, the higher priority interrupt cannot preempt the lower priority interrupt while it is being serviced by the CPU. Thus, priority is only used as a tie-breaker to decide between two pending interrupts with the same level. Finally, if both level and priority are same for two interrupts, the one with the higher ID value is given preference.

The CPU internally uses the effective interrupt level threshold, which is defined as the maximum of mintstatus.MIL and mintthresh.TH, to decide if a new pending machine mode interrupt will be allowed to preempt the execution of an ongoing machine mode interrupt handler.

2.9.2.6 CLIC Memory-Mapped Register Summary

Listed below are all the CLIC machine mode memory-mapped registers. The address of each register is specified as an offset to the value of mclicbase, as well as the absolute address.

Note that the clicintip[i], clicintie[i], clicintattr[i] and clicintctl[i] registers are all byte sized. Still word and half-word aligned accesses to these registers will be performed atomically.

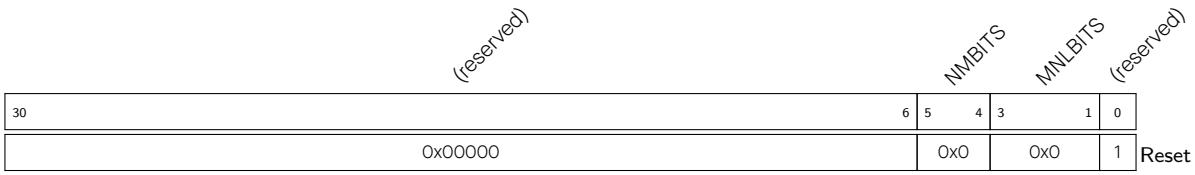
Machine Mode CLIC Registers				
Name	Description	MCLICBASE Offset	Absolute Address	Access
mcliccfg	CLIC machine mode global configuration register	0x0000	0x20800000	R/W
clicinfo	CLIC information register	0x0004	0x20800004	RO
mintthresh	CLIC machine mode interrupt threshold register	0x0008	0x20800008	R/W
clicintip[3]	Pending register for CLINT software interrupts	0x100C	0x2080100C	R/W
clicintie[3]	Enable register for CLINT software interrupts	0x100D	0x2080100D	R/W
clicintattr[3]	Attribute register for CLINT software interrupts	0x100E	0x2080100E	R/W
clicintctl[3]	Level control register for CLINT software interrupts	0x100F	0x2080100F	R/W
clicintip[7]	Pending register for CLINT timer interrupts	0x101C	0x2080101C	R/W
clicintie[7]	Enable register for CLINT timer interrupts	0x101D	0x2080101D	R/W
clicintattr[7]	Attribute register for CLINT timer interrupts	0x101E	0x2080101E	R/W

Name	Description	MCLICBASE Offset	Absolute Address	Access
clicintctl[7]	Level control register for CLINT timer interrupts	0x101F	0x2080101F	R/W
clicintip[16]	Pending register for external interrupt 0	0x1040	0x20801040	R/W
clicintie[16]	Enable register for external interrupt 0	0x1041	0x20801041	R/W
clicintattr[16]	Attribute register for external interrupt 0	0x1042	0x20801042	R/W
clicintctl[16]	Level control register for external interrupt 0	0x1043	0x20801043	R/W
clicintip[17]	Pending register for external interrupt 1	0x1044	0x20801044	R/W
clicintie[17]	Enable register for external interrupt 1	0x1045	0x20801045	R/W
clicintattr[17]	Attribute register for external interrupt 1	0x1046	0x20801046	R/W
clicintctl[17]	Level control register for external interrupt 1	0x1047	0x20801047	R/W
...	...	...	...	...
clicintip[n]	Pending register for nth external interrupt [6]	0x1040 + 4*n	0x20801040 + 4*n	R/W
clicintie[n]	Enable register for nth external interrupt [6]	0x1041 + 4*n	0x20801041 + 4*n	R/W
clicintattr[n]	Attribute register for nth external interrupt [6]	0x1042 + 4*n	0x20801042 + 4*n	R/W
clicintctl[n]	Level control register for nth external interrupt [6]	0x1043 + 4*n	0x20801043 + 4*n	R/W
....	....	....	....	....
clicintip[46]	Pending register for external interrupt 30	0x10B8	0x208010B8	R/W
clicintie[46]	Enable register for external interrupt 30	0x10B9	0x208010B9	R/W
clicintattr[46]	Attribute register for external interrupt 30	0x10BA	0x208010BA	R/W
clicintctl[46]	Level control register for external interrupt 30	0x10BB	0x208010BB	R/W
clicintip[47]	Pending register for external interrupt 31	0x10BC	0x208010BC	R/W
clicintie[47]	Enable register for external interrupt 31	0x10BD	0x208010BD	R/W
clicintattr[47]	Attribute register for external interrupt 31	0x10BE	0x208010BE	R/W
clicintctl[47]	Level control register for external interrupt 31	0x10BF	0x208010BF	R/W

[6] It must be noted that the external interrupts are assigned to CLIC interrupt IDs 16 and onwards. Thus, the MCLICBASE offset can also be written as 0x1000 + 4*i, where "i" represents the CLIC ID, and n = i + 16. On the other hand, the descriptions of these registers use the CLIC index "i" to specify the offset instead of the external interrupt index "n", which is used here just for a matter of convenience.

2.9.2.7 CLIC Memory-Mapped Register Description

Register 2.94. mcliccfg (0x20800000)



MNLBITS Configures the number of upper bits that are used globally (for each HP core) to encode the interrupt level in all the 8-bit clicintctl[i] registers for each machine mode interrupt. The only allowed values are in the range 0-8. Any higher value will be clamped to 8. (R/W)

NMBITS Configures the number of bits in clicintattr[i].MODE to be used globally (for each HP core) to represent the mode of an interrupt. For systems with machine mode only, this is hardwired to 0. This indicates that the clicintattr[i].MODE field is hardwired to 0x3 and therefore all interrupts are in machine mode only. (RO)

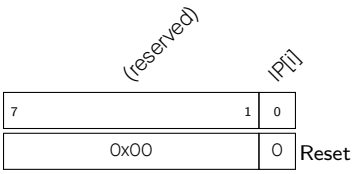
Register 2.95. clicinfo (0x20800004)



NUM_INTERRUPTS Represents the number of interrupts supported by this implementation of CLIC. Hardwired to 48. (RO)

CLICINTCTLBITS Represents the number of higher bits which are actually programmable in the 8-bit clicintctl[i] registers. Hardwired to 0x3. (RO)

Register 2.96. clicintip[i] (0x20801000 + 4*i)



IP[i] Represents the pending state for ith CLIC machine mode interrupt.

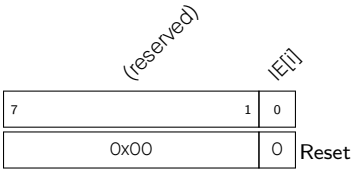
0: ith interrupt is not pending

1: ith interrupt is pending

- When the ith interrupt is configured as level type using clicintattr[i].TRIG, this bit is read-only, and to clear it the interrupt must be cleared from source.
- When the ith interrupt is configured as edge triggered using clicintattr[i].TRIG, this bit is R/W, and thus it is to be cleared by writing 0 to this register.

(R/W)

Register 2.97. clicintie[i] (0x20801001 + 4*i)



IE[i] Configures whether to enable the ith CLIC machine mode interrupt.

0: Disable

1: Enabled

(R/W)

Register 2.98. clicintattr[i] (0x20801002 + 4*i)



SHV Configures hardware vectoring for the ith CLIC machine mode interrupt.

0: ith interrupt is not hardware vectored. It means that upon taking this interrupt, the CPU will jump to the address configured in mtvec.

1: ith interrupt is hardware vectored. It means that upon taking this interrupt, the CPU will jump to the word address stored at (mtvt + 4*i) relative to the base address configured in mtvt.

(R/W)

TRIG Configures the trigger type and polarity of the ith CLIC machine mode interrupt.

0x0: interrupt is positive level-triggered

0x1: interrupt is positive edge-triggered

0x2: interrupt is negative level-triggered

0x3: interrupt is negative edge-triggered

(R/W)

MODE Configures the privilege mode of the ith CLIC interrupt.

0x0: User mode

0x3: Machine mode

Others: Reserved and not programmable

(R/W)

Register 2.99. clicintctl[i] (0x20801003 + 4*i)



CTL[i] Configures the level and priority of the ith CLIC machine mode interrupt.

- Since the CLICINTCTLBITS parameter is 3 in this implementation of CLIC, therefore the actual number of bits implemented in clicintctl[i] is the same, and thus the lower 5 bits of this register are hardwired to 1.
- The configured value of mcliccfg.MNLBITS determines the number of higher bits in clicintctl[i] which encode the level of the interrupt, while the remaining lower bits encode the priority.

(R/W)

2.9.3 Core-Local Interrupts (CLINT)

2.9.3.1 Overview

CLINT is the core-local interrupt (CLINT) block that includes software and timer interrupts and corresponding configuration registers. CLINT interrupt sources with IDs as shown below:

Table 2.9-4. Core-Local Interrupt Sources

ID	Description
3	M mode software interrupt
7	M mode timer interrupt

2.9.3.2 Features

- Two local level-type machine mode interrupt sources with programmable priorities
- Memory-mapped configuration and status registers
- 64-bit timer with interrupt functionality, overflow flags, and three sampling modes
- Software interrupt

2.9.3.3 Software Interrupt

An M mode software interrupt is controlled by setting or clearing the memory-mapped register MSIP.

The register bit clicintie[3] must be set for enabling the software interrupt at the core level.

Pending state of this interrupt can be checked by reading the corresponding bit of clicintip register of interrupt 3 in CLIC, that is clicintip[3] bit.

2.9.3.4 Timer Counter and Interrupt

The timer counter can be enabled by setting the mtime_EN bit in mtimectl.

mtimecmplo and mtimecmphi registers are set to the value for comparison with system timer with lower and higher 32 bits respectively.

The CPU provides two local memory-mapped 32-bit wide registers mtimeloadlo and mtimeloadhi, which has both read/write access, to load the system timer with a specific value.

Interrupt for M mode is asserted when 64-bit value of system timer value exceeds the 64-bit combined value of mtimecmplo and mtimecmphi.

Pending state of timer interrupt is reflected as IP bit of clicintip register of interrupt 7 in CLIC.

For de-asserting the pending timer interrupt in M mode, either IE bit of clicintie register of interrupt 7 in CLIC has to be cleared or the value of the mtimecmplo and mtimecmphi register needs to be updated.

2.9.3.5 Register Summary

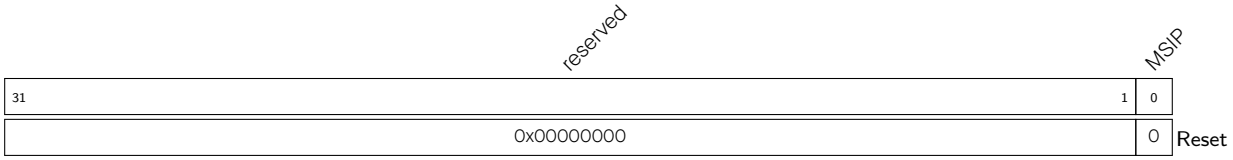
The addresses in this section are relative to CPU sub-system base address provided in Figure 9.3-1 in Chapter 9 *System and Memory*.

Name	Description	Address	Access
msip	Core-local machine software interrupt pending register	0x0000	R/W
mtimecmplo	Lower 32 bits of the core-local machine timer compare value	0x4000	R/W
mtimecmphi	Higher 32-bit core-local machine timer compare value	0x4004	R/W
mtimeloadlo	Lower 32 bits of core-local machine timer load value	0x4008	R/W
mtimeloadhi	Higher 32 bits of core-local machine timer load value	0x400C	R/W
mtimectl	Core-local machine timer interrupt control/status register	0x4010	R/W
mtimelo	Lower 32 bits of core-local timer counter value	0xBFF8	RO
mtimehi	Higher 32 bits core-local timer counter value	0xBFFC	RO

2.9.3.6 Register Description

The addresses in this section are relative to CPU sub-system base address provided in Figure 9.3-1 in Chapter 9 *System and Memory*.

Register 2.100. msip (0x0000)



MSIP Configures the pending status of the machine software interrupt.

0: Not pending

1: Pending

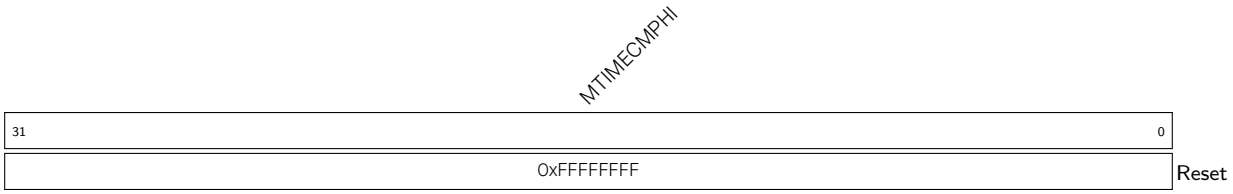
(R/W)

Register 2.101. mtimecmplo (0x4000)



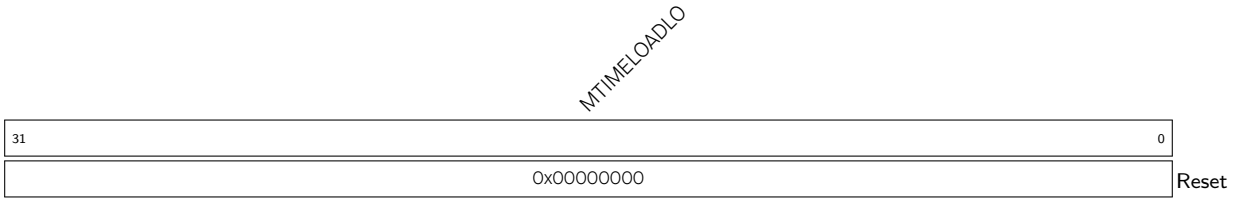
MTIMECMPLO Represents the value to compare with the lower 32 bits of the system counter. (R/W)

Register 2.102. mtimecmphi (0x4004)



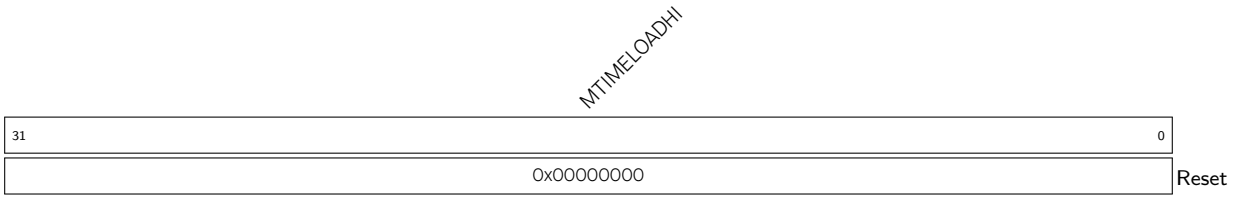
MTIMECMPHI Represents the value to compare with the higher 32 bits of the system counter. (R/W)

Register 2.103. mtimeloadlo (0x4008)



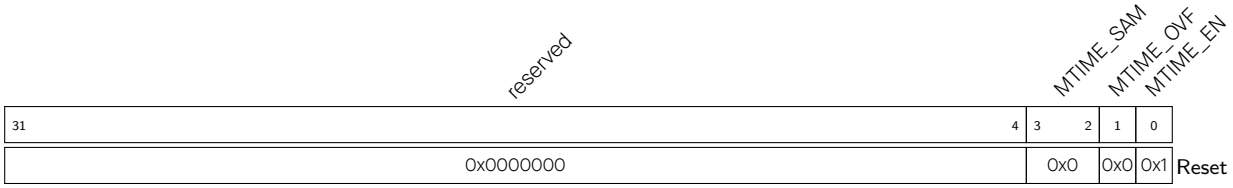
MTIME_LOADLO Represents the lower 32 bits to be loaded to the system counter. (R/W)

Register 2.104. mtimeloadhi (0x400C)



MTIME_LOADHI Represents the higher 32 bits to be loaded to the system counter. (R/W)

Register 2.105. mtimectl (0x4010)



MTIME_EN Configures whether to enable the system counter.

1: Enable

0: Disable

This bit is implemented only in Core 0 MTIMERCTL register. Writing the corresponding bit of Core 1's MTIMERCTL has no effect on the system timer. Since Core 1 can access Core 0's CLINT registers, it can run or pause the timer from MTIMERCTL register of Core 0.

(R/W)

MTIME_OVF Configures the overflow bit of the system counter.

1: Represents that system counter value is maximum, that is 0xFFFFFFFFFFFFFFFF

0: Otherwise

When the counter value is 0xFFFFFFFFFFFFFFFF, which is the maximum value, then this bit is set as 1 by hardware. Software can clear this bit by writing 0 to it. Writing 1 has no effect.

(R/W)

MTIME_SAM Configures the sampling mode of MTIMELO and MTIMEHI registers to allow accurate reading of the 64-bit system count. For the 64-bit system count, only one half of the read can be performed atomically. Therefore, the sampling mode allows the value of the other half to be sampled and stored in a buffer, and upon the next read of the other half, the sampled value from this buffer is retrieved instead of the actual counter.

0x0: No sampling (Default)

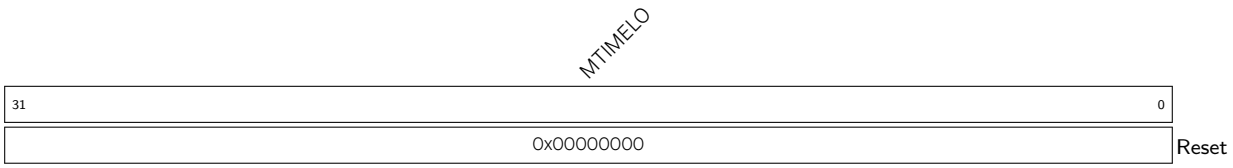
0x1: Sample upper half of the system count on reading MTIMELO

0x2: Sample lower half of the system count on reading MTIMEHI

0x3: Sample the other half of the system count on reading MTIMELO or MTIMEHI

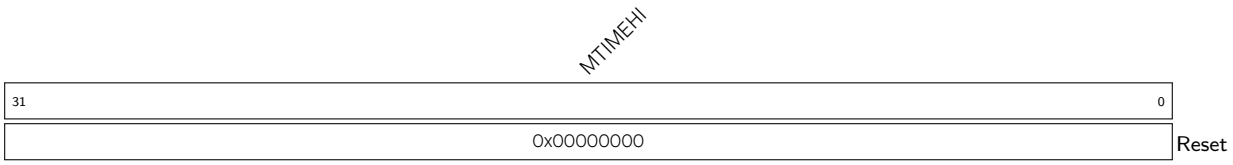
(R/W)

Register 2.106. mtimelo (0xBFF8)



MTIMELO Represents the current value of the lower 32 bits of the system counter. (RO)

Register 2.107. mtimehi (0xBFFC)



MTIMEHI Represents the current value of the higher 32 bits of the system counter. (RO)

2.10 Memory Protection Unit

Each HP core implements standard RISC-V physical memory protection (PMP) scheme along with custom physical memory attribute checker (PMAC) logic to prevent unauthorized memory access.

2.10.1 Standard Physical Memory Protection

2.10.1.1 Overview

The CPU core includes a physical memory protection (PMP) unit, which is fully compliant with RISC-V Instruction Set Manual, Volume II: Privileged Architecture, Version 1.10. It can be used by software to set memory access privileges (read, write and execute permissions) for required memory regions. In addition to standard PMP checks, the CPU core also implements custom physical memory attributes (PMA) checkers to provide additional permission checks based on pre-defined attributes.

2.10.1.2 Features

The PMP unit can be used to restrict access to physical memory. The main features are:

- Up to 32 PMP entries
- Minimum address granularity of 128 bytes
- Three address matching modes: OFF, TOR (Top of Range), and NAPOT (Naturally Aligned Power-of-Two). NA4 (Naturally Aligned 4-Byte Region) address matching mode is not supported
- Permission controls for read, write, and execute
- Lock function for each PMP entry
- Maximum physical space address of 4 GB

2.10.1.3 Functional Description

Software can program the PMP unit's configuration and address registers in order to contain faults and support secure execution. PMP CSRs can only be programmed in machine mode. Once the PMP unit is enabled by configuring PMP CSRs, write, read and execute permission checks are applied to all the accesses in user mode as per programmed values of the enabled 16 pmpcfgX and pmpaddrX registers (refer to Section 2.10.1.4).

By default, PMP grants permission to all accesses in machine mode and revokes permission of all access in user mode. This implies that it is mandatory to program the address range and valid permissions in pmpcfg and pmpaddr registers (refer to Section 2.10.1.4) for any valid access to pass through in user mode. However, it is not required for machine mode as PMP permits all accesses to go through by default. In cases where PMP checks are also required in machine mode, software can set the lock bit of required PMP entry to enable permission checks on it. Once the lock bit is set, it can only be cleared through CPU reset.

When any instruction is being fetched from a memory region without execute permissions, an exception is generated at processor level and exception cause is set as instruction access fault in mcause CSR. Similarly, any load/store access without valid read/write permissions will result in an exception generation with mcause updated as load access and store access fault respectively. In case of load/store access faults, violating address is captured in mtval CSR.

2.10.1.4 Register Summary

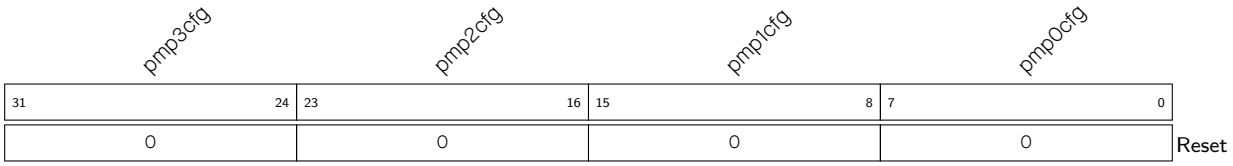
Below is a list of PMP CSRs supported by the CPU. These are only accessible from machine mode.

Name	Description	Address	Access
pmpcfg0	Physical memory protection configuration register	0x3A0	R/W
pmpcfg1	Physical memory protection configuration register	0x3A1	R/W
pmpcfg2	Physical memory protection configuration register	0x3A2	R/W
pmpcfg3	Physical memory protection configuration register	0x3A3	R/W
pmpcfg4	Physical memory protection configuration register	0x3A4	R/W
pmpcfg5	Physical memory protection configuration register	0x3A5	R/W
pmpcfg6	Physical memory protection configuration register	0x3A6	R/W
pmpcfg7	Physical memory protection configuration register	0x3A7	R/W
pmpaddr*n* (*n*: 0-31)	Physical memory protection address register	0x3B0+0x1**n*	R/W

2.10.1.5 Register Description

PMP unit implements all pmpcfg0-3 and pmpaddr0-15 CSRs as defined in RISC-V Instruction Set Manual, Volume II: Privileged Architecture, Version 1.10.

Register 2.108. pmpcfg0 (0x3A0)



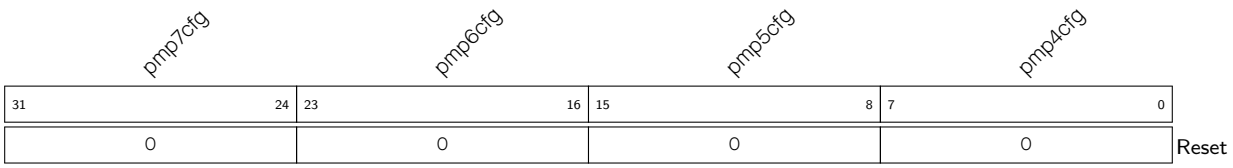
pmp3cfg Configuration register for PMP entry 3. (R/W)

pmp2cfg Configuration register for PMP entry 2. (R/W)

pmp1cfg Configuration register for PMP entry 1. (R/W)

pmp0cfg Configuration register for PMP entry 0. (R/W)

Register 2.109. pmpcfg1 (0x3A1)



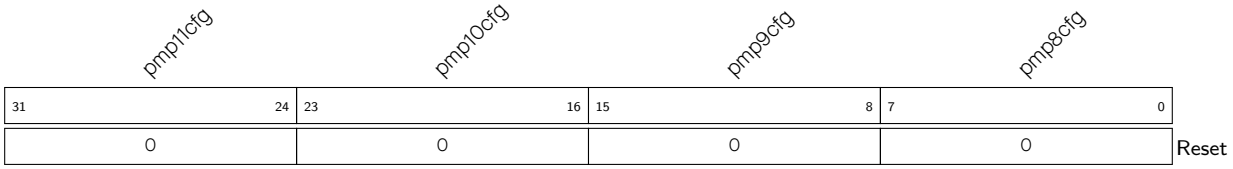
pmp7cfg Configuration register for PMP entry 7. (R/W)

pmp6cfg Configuration register for PMP entry 6. (R/W)

pmp5cfg Configuration register for PMP entry 5. (R/W)

pmp4cfg Configuration register for PMP entry 4. (R/W)

Register 2.110. pmpcfg2 (0x3A2)



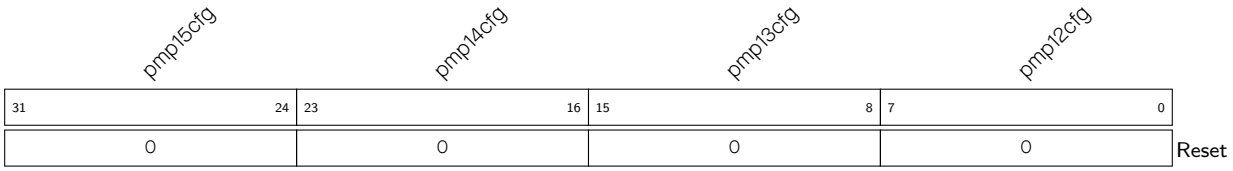
pmp11cfg Configuration register for PMP entry 11. (R/W)

pmp10cfg Configuration register for PMP entry 10. (R/W)

pmp9cfg Configuration register for PMP entry 9. (R/W)

pmp8cfg Configuration register for PMP entry 8. (R/W)

Register 2.111. pmpcfg3 (0x3A3)



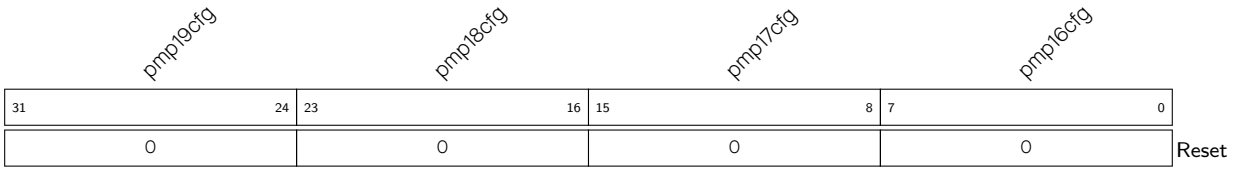
pmp15cfg Configuration register for PMP entry 15. (R/W)

pmp14cfg Configuration register for PMP entry 14. (R/W)

pmp13cfg Configuration register for PMP entry 13. (R/W)

pmp12cfg Configuration register for PMP entry 12. (R/W)

Register 2.112. pmpcfg4 (0x3A4)



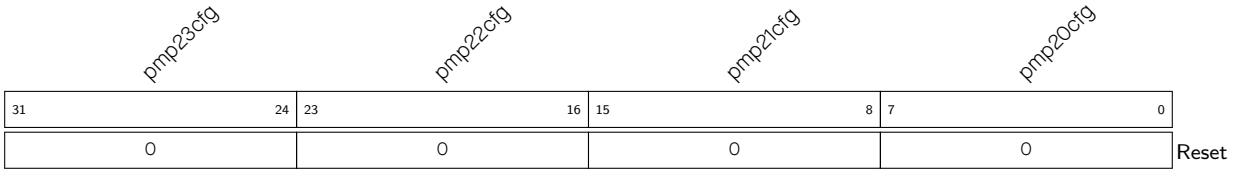
pmp19cfg Configuration register for PMP entry 19. (R/W)

pmp18cfg Configuration register for PMP entry 18. (R/W)

pmp17cfg Configuration register for PMP entry 17. (R/W)

pmp16cfg Configuration register for PMP entry 16. (R/W)

Register 2.113. pmpcfg5 (0x3A5)



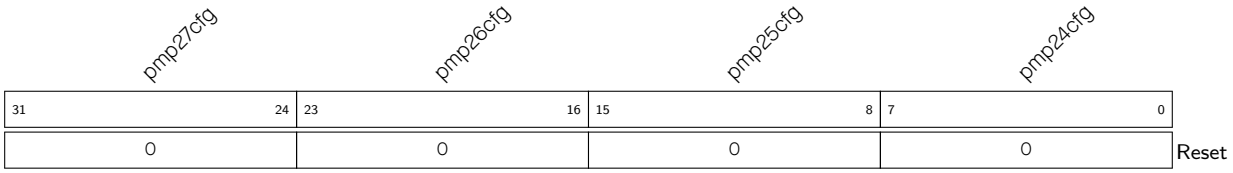
pmp23cfg Configuration register for PMP entry 23. (R/W)

pmp22cfg Configuration register for PMP entry 22. (R/W)

pmp21cfg Configuration register for PMP entry 21. (R/W)

pmp20cfg Configuration register for PMP entry 20. (R/W)

Register 2.114. pmpcfg6 (0x3A6)



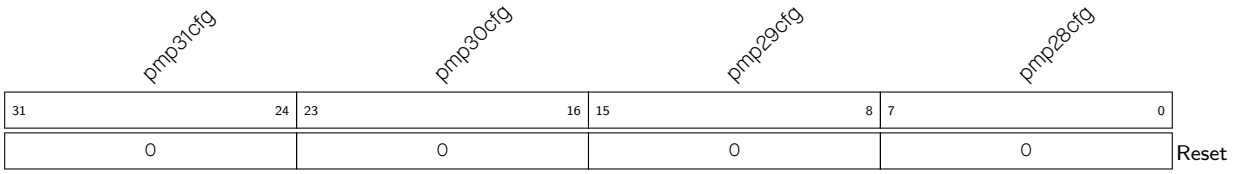
pmp27cfg Configuration register for PMP entry 27. (R/W)

pmp26cfg Configuration register for PMP entry 26. (R/W)

pmp25cfg Configuration register for PMP entry 25. (R/W)

pmp24cfg Configuration register for PMP entry 24. (R/W)

Register 2.115. pmpcfg7 (0x3A7)



pmp31cfg Configuration register for PMP entry 31. (R/W)

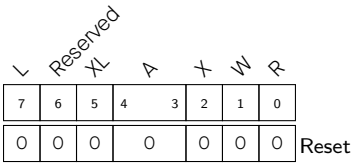
pmp30cfg Configuration register for PMP entry 30. (R/W)

pmp29cfg Configuration register for PMP entry 29. (R/W)

pmp28cfg Configuration register for PMP entry 28. (R/W)

The PMP configuration register format for any PMP entry is shown as follows.

Register 2.116. pmpXcfg



L Configures lock bit. Once set, it can only be cleared through a core reset. (R/W)

XL Configures custom-lock bit. Once set, it can only be cleared through a core reset.
0: PMP CSR accesses work normally
1: The corresponding pmpaddr and pmpcfg entry are locked and cannot be modified
(R/W)

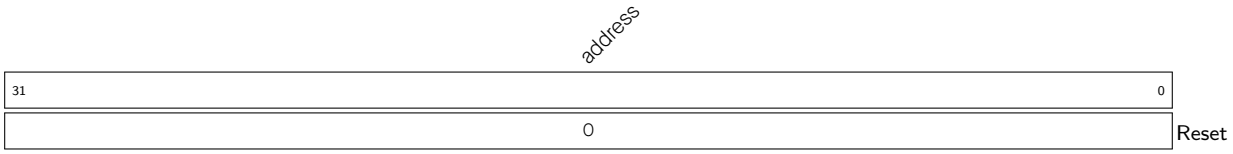
A Configures address matching mode.
0: OFF
1: TOR (Top of range)
2: Not Supported
3: NAPOT (Naturally aligned power-of-two region >= 128 Bytes)
(R/W)

X Configures execute permission. When set, execution from the address range is allowed. (R/W)

W Configures write permission. When set, memory writes to the address range are allowed. (R/W)

R Configures read permission. When set, memory reads from the address range are allowed. (R/W)

Register 2.117. pmpaddr*n* (*n*: 0-31) (0x3B0+0x1**n*)



address Configures the address for pmpaddr*n*. (R/W)

2.10.2 Custom Physical Memory Attribute (PMA) Checker

2.10.2.1 Overview

CPU core also implements a custom physical memory attributes checker (PMAC) to provide additional permission checks based on pre-defined memory types configured through custom CSRs. Please note that the PMA checks are applied irrespective of the core's privilege mode, i.e., it doesn't differentiate between machine and user mode.

2.10.2.2 Features

PMAC supports below features:

- 16 PMA entries
- Configurable memory type for defined memory regions
- Minimum address granularity of 128 bytes
- Three address matching mode: OFF, TOR (Top of Range) and NAPOT (Naturally Aligned Power-of-Two). NA4 (Naturally Aligned 4-Byte Region) address matching mode is not supported
- Permission controls for read, write and execute
- Lock function for each PMA entry
- Maximum physical space address of 4 GB
- Configurable attribute for defined memory regions

2.10.2.3 Functional Description

Software can program the PMAC unit's configuration and address registers in order to avoid faults due to access to invalid memory regions. PMAC CSRs can only be programmed in machine mode. Once enabled, write, read and execute permission checks are applied to all the accesses irrespective of privilege mode as per programmed values of enabled 16 pma_cfgX and pma_addrX registers (refer to Section 2.10.2.4). Access to entries marked as invalid memory types will result in fetch fault or load/store fault exception, as the case may be.

Exception generation and handling for PMAC related faults will be handled in similar way to PMP checks. When any instruction is being fetched from a memory region configured as a null or an invalid memory region, an exception is generated at the processor level and the exception cause is set as instruction access fault in mcause CSR. Similarly, any load/store access to a null or an invalid memory region will result in an exception generation with mcause updated as load access or store access fault respectively. In case of load/store access faults, violating address is captured in mtval CSR. For the PMAC entries configured as valid memory, the handling is same as for PMP checks.

A lock bit per entry is also provided in case software wants to disable programming of PMAC registers. Once the lock bit in any pma_cfgX register is set, respective pma_cfgX and pma_addrX registers can not be programmed further, unless a CPU reset cycle is applied.

A 4-bit field in PMAC CSRs is also provided to define attributes for memory regions. These bits are not used internally by the CPU core for any purpose. Based on address match, these attributes are provided on

load/store interface as side-band signals and are used by cache controller block for its internal operation.

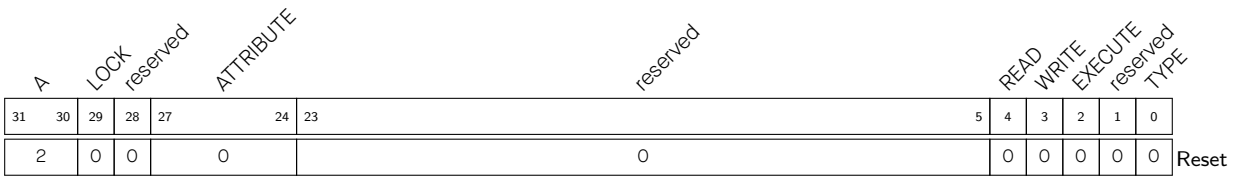
2.10.2.4 Register Summary

Below is a list of PMA CSRs supported by the CPU. These are only accessible from machine mode.

Name	Description	Address	Access
pma_cfg*n* (*n*: 0-15)	Physical memory attribute configuration register	0xBC0+0x1**n*	R/W
pma_addr*n* (*n*: 0-15)	Physical memory attribute address register	0xBD0+0x1**n*	R/W

2.10.2.5 Register Description

Register 2.118. pma_cfg*n* (*n*: 0-15) (0xBC0+0x1**n*)



A Configures address type. The functionality is the same as pmpcfg register's A field.

0x0: OFF

0x1: TOR

0x2: NA4

0x3: NAPOT

(R/W)

LOCK Configures whether to lock the corresponding pma_cfgX and pma_addrX.

0: Not lock

1: Lock. The write permission to the corresponding pma_cfgX and pma_addrX is revoked. It can only be unlocked by core reset.

(R/W)

ATTRIBUTE Configures the values to be driven on DRAM attribute ports.

Bit 27:

0: Cacheable

1: Non-cacheable

Bit 26:

0: Write-back

1: Write-through

Bit 25:

0: Write miss allocate

1: Write miss no allocate

Bit 24:

0: Read miss allocate

1: Read miss no allocate

(R/W)

READ Configures read-permission for the corresponding region.

0: Read not allowed

1: Read allowed

(R/W)

Continued on the next page...

Register 2.118. pma_cfg*n* (*n*: 0-15) (0xBC0+0x1**n*)

Continued from the previous page...

WRITE Configures write-permission for the corresponding region.

0: Write not allowed

1: Write allowed

(R/W)

EXECUTE Configures execute-permission for the corresponding region.

0: Execution not allowed

1: Execution allowed

(R/W)

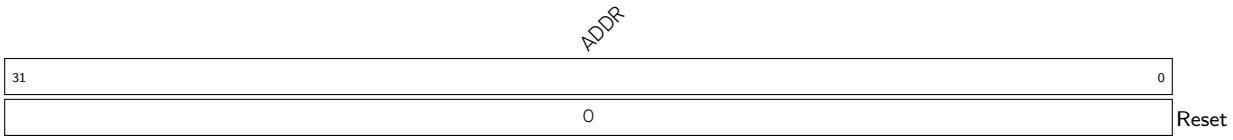
TYPE Configures region type.

0x0: Invalid memory region (RWX access will be treated as 0, even if programmed to 1)

0x1: Valid memory region (Programmed RWX access will be applicable)

(R/W)

Register 2.119. pma_addr*n* (*n*: 0-15) (0xBD0+0x1**n*)



ADDR Configures address. The functionality is same as pmpaddr register. (R/W)

2.11 Debug and Trace Support

2.11.1 Debug

2.11.1.1 Overview

This section describes how to debug software running on HP and LP CPU cores. Debug support is provided through standard JTAG pins and complies to the RISC-V External Debug Support Version 0.13.2 specification.

HP cores and LP core have their own dedicated JTAG DTM (debug transport module) in order to connect with respective Debug Modules. Both these JTAG DTM's are daisy-chained to provide debugger access to respective cores via a single JTAG interface. Depending on its debug requirements, Debugger can keep the unused DTM in a bypass state while debugging the required core. Both HP cores can be debugged through a single debug module (DM) connected to JTAG DTM (HP Cores), while the LP core is debugged through a separate debug module (DM) connected to JTAG DTM (LP Core).

Please note it is not possible to debug HP cores and LP core at same time as they are accessed through separate debug modules.

Figure 2.11-1 below shows the main components of External Debug Support.

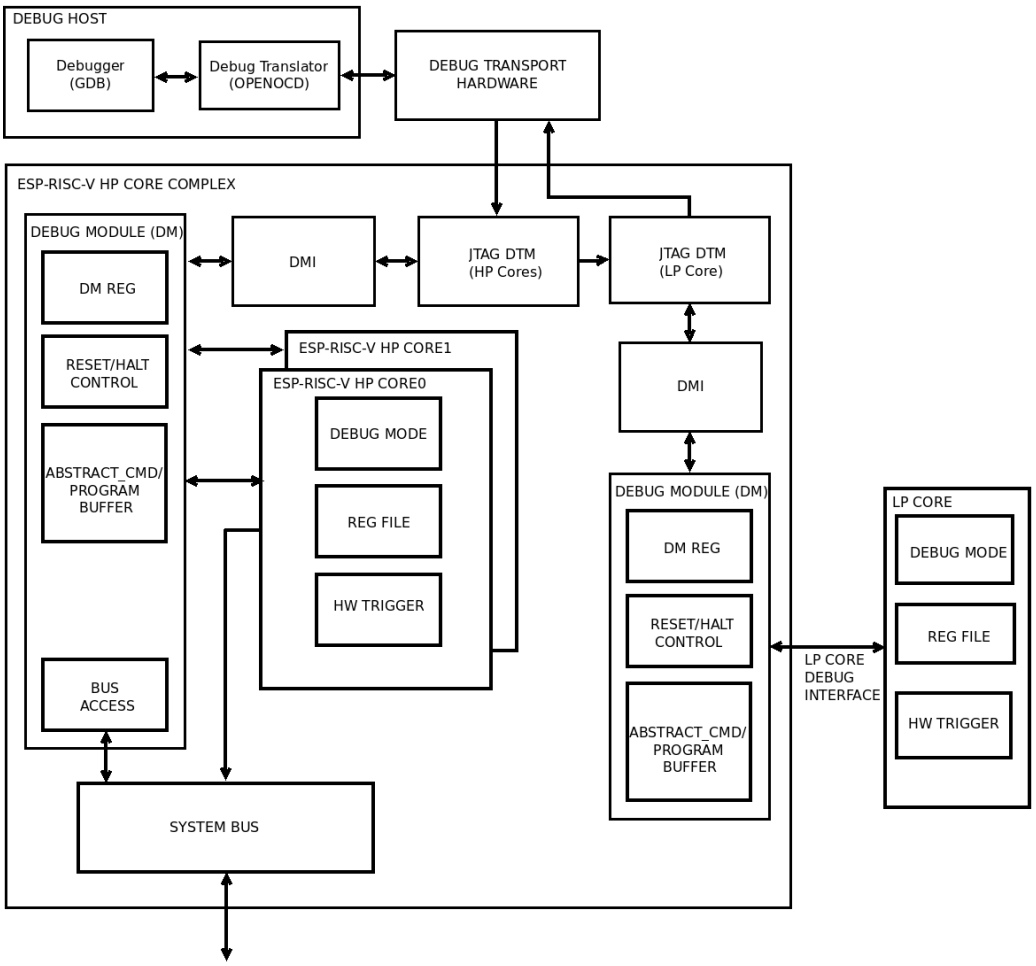


Figure 2.11-1. Debug System Overview

This section focuses on HP core debug features. For LP core debug features, please refer to Chapter 5 *Low-Power CPU*.

The user interacts with the Debug Host (e.g. laptop), which is running a debugger (e.g. GDB). The debugger communicates with a Debug Translator (e.g. OpenOCD, which may include a hardware driver) to communicate with Debug Transport Hardware (e.g. ESP-Prog adapter). The Debug Transport Hardware connects the Debug Host to the HP core's Debug Transport Module (DTM) through standard JTAG interface (bypassing LP core DTM). The DTM provides access to the Debug Module (DM) using the Debug Module Interface (DMI).

The DM allows the debugger to halt selected HP cores. Abstract commands provide access to GPRs (general-purpose registers). The Program Buffer allows the debugger to execute arbitrary code on the core, which allows access to additional CPU core state. Alternatively, additional abstract commands can provide access to additional CPU core state.

Each HP core contains Trigger Module supporting 3 triggers. When trigger conditions are met, respective core will halt spontaneously and inform the debug module that they have halted.

System bus access block allows memory and peripheral register access without affecting either HP core.

2.11.1.2 Features

The HP CPU supports the following basic debugging features for each HP core:

- Provides necessary information about the implementation to the debugger
- Allows the core to be halted and resumed
- Core registers (including CSRs) can be read/written by debugger
- Core can be debugged from the first instruction executed after reset
- Core can be reset through debugger
- Core can be halted on software breakpoint (planted breakpoint instruction)
- Hardware single-stepping
- Execute arbitrary instructions in the halted core by means of the program buffer. 2-word program buffer is supported
- System bus access is supported through word aligned address access
- Supports three Hardware Triggers (can be used as breakpoints/watchpoints) per HP core as described in Section 2.11.3
- Supports LP core debug through daisy chaining of its JTAG DTM

2.11.1.3 Functional Description

As mentioned earlier, Debug Scheme conforms to the RISC-V External Debug Support Version 0.13.2 specification. Please refer to it for functional operation details.

2.11.1.4 JTAG Control

Standard JTAG interface is the only way for DTM to access DM. The hardware provides two JTAG methods: PAD_to_JTAG and USB_to_JTAG.

- PAD_to_JTAG: The JTAG's signal source comes from IO.
- USB_to_JTAG: The JTAG's signal source comes from USB Serial/JTAG Controller.

Which JTAG method to use depends on many factors. The following table shows the configuration method.

Table 2.11-1. JTAG Control

Temporary disable JTAG 3,4	EFUSE_DIS_USB_SERIAL_JTAG 4	EFUSE_DIS_USB_JTAG 4	EFUSE_DIS_PAD_JTAG 4	EFUSE_JTAG_SEL_ENABLE 4	Strapping Pin GPIO34 5	USB JTAG Status	PAD JTAG Status
0	0	0	0	0	x 2	Available 1	Unavailable 1
0	0	0	0	1	1	Available	Unavailable
0	0	0	0	1	0	Unavailable	Available
0	0	0	1	x	x	Available	Unavailable
0	0	1	0	x	x	Unavailable	Available
0	0	1	1	x	x	Unavailable	Unavailable
0	1	0	0	0	x	Unavailable	Unavailable
0	1	0	0	1	0	Unavailable	Available
0	1	0	0	1	1	Unavailable	Unavailable
0	1	0	1	x	x	Unavailable	Unavailable
0	1	1	0	x	x	Unavailable	Available
0	1	1	1	x	x	Unavailable	Unavailable
1	x	x	x	x	x	Unavailable	Unavailable

Note:

1. Available: the corresponding JTAG function is available.
   Unavailable: the corresponding JTAG function is not available.
2. x: do not care.
3. "Temporary disable JTAG" means that if there are an even number of bits "1" in EFUSE_SOFT_DIS_JTAG[2:0], the JTAG function is turned on (the corresponding value in the table is 1), otherwise it is turned off (the corresponding value in the table is 0). However, under certain special conditions of the HMAC Accelerator in ESP32-P4, the JTAG function may be turned on even if there is an odd number of bits "1" in EFUSE_SOFT_DIS_JTAG[2:0]. For information on how HMAC affects JTAG functionality, please refer to Chapter HMAC Accelerator.
4. Please refer to Chapter eFuse Controller to get more information about eFuse.
5. Please refer to Chip Boot Control to get more information about the strapping pin GPIO34.

2.11.1.5 Register Summary

Below is the list of Debug CSRs supported by HP cores:

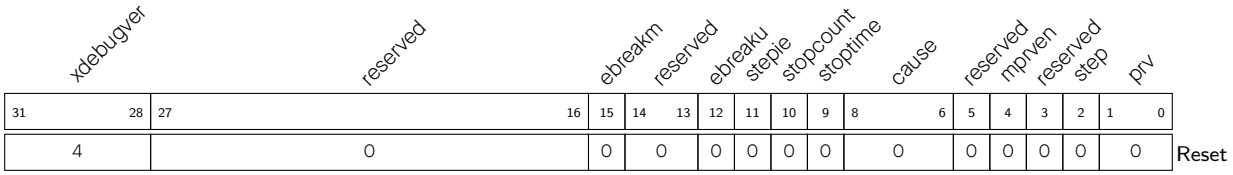
Name	Description	Address	Access
dcsr	Debug Control and Status Register	0x7B0	R/W
dpc	Debug PC Register	0x7B1	R/W
dscratch0	Debug Scratch Register 0	0x7B2	R/W
dscratch1	Debug Scratch Register 1	0x7B3	R/W

All the debug module registers are implemented in conformance to the specification RISC-V External Debug Support Version 0.13.2. Please refer to it for more details.

2.11.1.6 Register Description

Below are the details of Debug CSRs supported by HP cores:

Register 2.120. dcsr (0x7B0)



xdebugver Represents the debug version.

4: External debug support exists

(RO)

ebreakm When 1, ebreak instructions in Machine Mode enter debug mode.

(R/W)

ebreaku When 1, ebreak instructions in User/Application Mode enter debug mode.

(R/W)

stepie Configures interrupt control in debug mode. 0: Interrupts are disabled during single-stepping

1: Interrupts are enabled during single-stepping

(R/W)

Debugger must not change the value of this bit while the hart is running.

stopcount Configures mcycle counter control in debug mode. 0: Increment counters as usual.

1: Not increment any counters while in debug mode or on ebreak instructions that cause entry into debug mode. These counters include the cycle and instret CSRs.

(R/W)

stoptime Configures timer control in debug mode.

0: Increment timers as usual.

1: Not increment any hart-local timers while in debug mode.

(R/W)

cause Explains why debug mode was entered. When there are multiple reasons to enter debug mode in a single cycle, the cause with the highest priority number is the one written.

1: An ebreak instruction was executed. (priority 3)

2: The Trigger Module caused a halt. (priority 4, highest)

3: haltreq was set. (priority 1)

4: The CPU core single-stepped because the step was set. (priority 0, lowest)

5: The hart halted directory out of reset due to resethaltreq. (priority 2)

Other values are reserved for future use.

(RO)

mprven Configures whether to enable the functionality of MPRV bit in mstatus CSR during debug mode.

0: Disable

1: Enable

(R/W)

Continued on the next page...

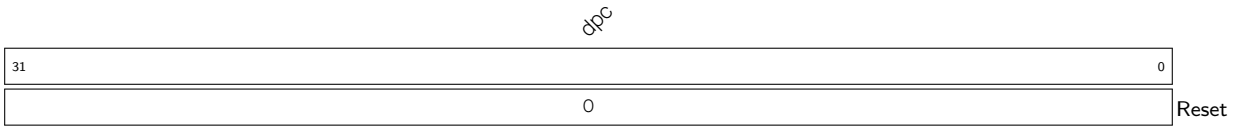
Register 2.120. dcsr (0x7B0)

Continued from the previous page...

step When set and not in debug mode, the core will only execute a single instruction and then enter debug mode.
If the instruction does not complete due to an exception, the core will immediately enter debug mode before executing the trap handler, with appropriate exception registers set.
Setting this bit does not mask interrupts. This is a deviation from the RISC-V External Debug Support Specification Version 0.13.
(R/W)

prv Contains the privilege level the core was operating in when debug mode was entered. A debugger can change this value to change the core's privilege level when exiting debug mode. Only **0x3** (machine mode) and **0x0** (user mode) are supported. (R/W)

Register 2.121. dpc (0x7B1)



dpc Upon entry to debug mode, dpc is written with the virtual address of the next instruction to be executed. When resuming, the CPU core's PC is updated to the virtual address stored in dpc. A debugger may write dpc to change where the CPU resumes. (R/W)

Table 2.11-3. Virtual address in DPC upon Debug Mode Entry

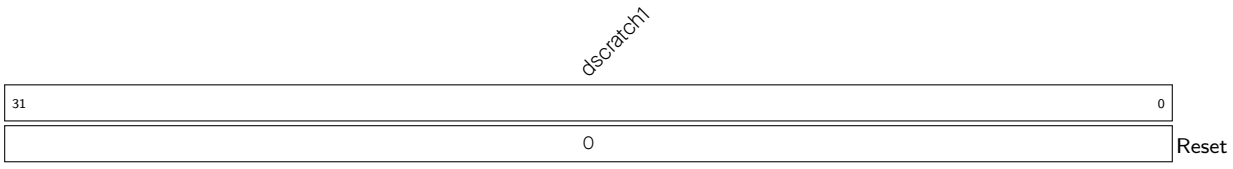
Cause	Virtual Address in DPC
ebreak	Address of ebreak instruction
Single Step	Address of the instruction that would be executed next if no debugging was going on i.e. pc+4 for 32-bit instruction that don't change program flow, the destination PC on taken jumps/branches, etc.
Trigger Module	If timing is 0, the address of the instruction which caused the trigger to fire. If timing is 1, the address of next instruction to be executed at the time that debug mode was entered.
Halt Request	Address of the next instruction to be executed at the time that debug mode was entered.

Register 2.122. dscratch0 (0x7B2)



dscratch0 Used by Debug Module internally. (R/W)

Register 2.123. dscratch1 (0x7B3)



dscratch1 Used by Debug Module internally. (R/W)

2.11.2 Debug Halt Groups

2.11.2.1 Overview

In a multi-core system, when the debugging software is running on a given core, it is useful that other cores do not change the state of the system. This requirement is addressed by synchronous halt and resume. It is important that halt/resume information is communicated as quickly as possible to other cores. So, it is better to do it based on chip infrastructure rather than commands through the debugger software running on the host.

2.11.2.2 Features

- Support cross-triggering between HP cores
- Overriding the RunStall functionality of a core

2.11.2.3 Functional Description

HP core debug module implements dmcs2 register specified in the upcoming specification, RISC-V External Debug Version 1.0-STABLE, to implement simultaneous halting/resuming based on hart groups.

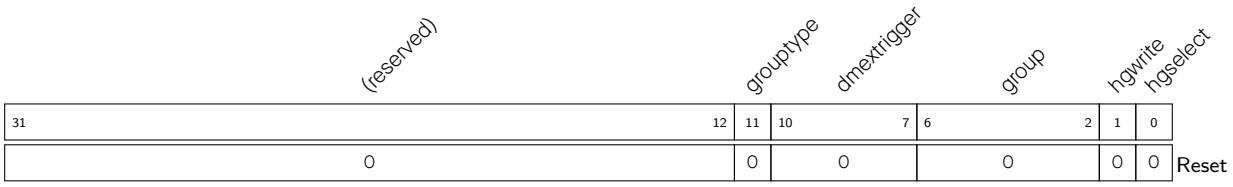
2.11.2.4 Register Summary

Below is a required control register implemented inside debug module.

Name	Description	Address	Access
dmcs2	Debug module control and status register 2	0x32	R/W

2.11.2.5 Register Description

Register 2.124. dmcs2 (0x32)



grouptype Configures the group type.

0 (halt group): The remaining fields in this register configure halt groups

1 (resume group): The remaining fields in this register configure resume groups

(R/W)

dmexttrigger Configures the currently selected DM external trigger. If a non-existent trigger value is written here, the hardware will change it to a valid value or 0 if no DM external triggers exist.

(R/W)

group When hgselect is 0, this field contains the group of the hart specified by hartsel.

When hgselect is 1, this field contains the group of the DM external trigger selected by dmexttrigger.

The value written to this field is ignored unless hgwrite is also written 1.

Group numbers are contiguous starting from 0, with the highest number being implementation-dependent, and possibly different between different group types. Debuggers should read back this field after writing to confirm they are using a supported hart group. If groups are not implemented, this entire field is 0.

(R/W)

hgwrite When 1 is written and hgselect is 0, for every selected hart, the DM will change its group to the value written to group, if the hardware supports that group for that hart. Implementations may also change the group of a minimal set of unselected harts in the same way, if that is necessary due to a hardware limitation. When 1 is written and hgselect is 1, the DM will change the group of the DM external trigger selected by dmexttrigger to the value written to group, if the hardware supports that group for that trigger. Writing 0 has no effect.

(R/W)

hgselect Tied to zero.

0: Operate on HART

(RO)

2.11.3 Hardware Trigger

2.11.3.1 Overview

2.11.3.2 Features

Each HP core implements a hardware trigger block to provide breakpoint and watchpoint capability for debugging. Each trigger has the following features:

- Three independent trigger units, each of which can be configured for matching the address of the program counter or load-store accesses
- Preempting execution by causing a breakpoint exception
- Halting execution and transferring control to debuggers
- Support for NAPOT (naturally aligned power-of-two regions) address encoding

2.11.3.3 Functional Description

The Hardware Trigger module provides seven CSRs, which are listed under Section register summary. Among these, tdata1 and tdata2 are abstract CSRs, which means they are shadow registers for accessing internal registers for each of the four trigger units, one at a time.

To choose a particular trigger unit write the index (0-2) of that unit into tselect CSR. When tselect is written with a valid index, the abstract CSRs tdata1 and tdata2 are automatically mapped to reflect internal registers of that trigger unit. Each trigger unit has two internal registers, namely mcontrol and maddress, which are mapped to tdata1 and tdata2, respectively.

Writing larger than allowed indexes to tselect will clip the written value to the largest valid index, which can be read back. This property may be used for enumerating the number of available triggers during initialization or when using a debugger.

Since software or debugger may need to know the type of the selected trigger to correctly interpret tdata1 and tdata2, the 4 bits (31-28) of tdata1 encodes the type of the selected trigger. This type field is read-only and always provides a value of 0x2 for every trigger, which stands for match type trigger, hence, it is inferred that tdata1 and tdata2 are to be interpreted as mcontrol and maddress. The information regarding other possible values can be found in the specification, RISC-V External Debug Support Version 0.13.2, but this trigger module only supports type 0x2.

Once a trigger unit has been chosen by writing its index to tselect, it will become possible to configure it by setting the appropriate bits in mcontrol CSR (tdata1) and writing the target address to maddress CSR (tdata2).

Each trigger unit can be configured to either cause breakpoint exception or enter debug mode, by writing to the action field of mcontrol. This bit can only be written from debugger, thus by default a trigger, if enabled, will cause breakpoint exception.

mcontrol for each trigger unit has a hit bit which may be read, after CPU halts or enters exception, to find out if this was the trigger unit that fired. This bit is set as soon as the corresponding trigger fires, but it has to be manually cleared before resuming operation. Although, failing to clear it does not affect normal execution in any way.

Each trigger unit only supports match on address, although this address could either be that of a load/store access or the virtual address of an instruction. The address and size of a region are specified by writing to maddress (tdata2) CSR for the selected trigger unit. Larger than 1 byte region sizes are specified through NAPOT (naturally aligned power-of-two) encoding (see Table 2.11-5) and enabled by setting match bit in mcontrol. Note that for NAPOT encoded addresses, by definition, the start address is constrained to be aligned to (i.e. an integer multiple of) the region size. The maximum size supported is 128 bytes.

Table 2.11-5. NAPOT encoding for maddress

maddress(31-0)	Start Address	Size (bytes)
aaa...aaaaaaaaa0	aaa...aaaaaaaaa0	2
aaa...aaaaaaaa01	aaa...aaaaaaaa00	4
aaa...aaaaaaa011	aaa...aaaaaaa000	8
aaa...aaaaaa0111	aaa...aaaaaa0000	16
	....	
aaa...0111111111	aaa...aaa0000000	2^8

tcontrol CSR is common to all trigger units. It is used for preventing triggers from causing repeated exceptions in machine mode while execution is happening inside a trap handler. This also disables breakpoint exceptions inside ISRs by default, although, it is possible to manually enable this right before entering an ISR, for debugging purposes. This CSR is not relevant if a trigger is configured to enter debug mode.

2.11.3.4 Trigger Execution Flow

When hart is halted and enters debug mode due to the firing of a trigger (action = 1):

- dpc is set to current PC (in decode stage)
- cause field in dcsr is set to 2, which means halt due to trigger
- hit bit is set to 1, corresponding to the trigger(s) which fired

When hart goes into trap due to the firing of a trigger (action = 0) :

- mepc is set to current PC (in decode stage)
- mcause is set to 3, which means breakpoint exception
- mpte is set to the value in mte right before trap
- mte is set to 0
- hit bit is set to 1, corresponding to the trigger(s) which fired

Note: If two different triggers fire at the same time, one with action = 0 and another with action = 1, then hart is halted and enters debug mode.

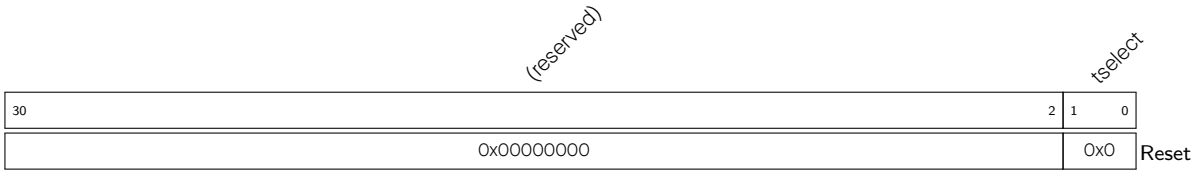
2.11.3.5 Register Summary

Below is a list of Trigger Module CSRs supported by the CPU. These are only accessible from machine mode.

Name	Description	Address	Access
tselect	Trigger select register	0x7A0	R/W
tdata1	Trigger abstract data 1	0x7A1	R/W
tdata2	Trigger abstract data 2	0x7A2	R/W
tdata3	Trigger abstract data 3	0x7A3	R/W
tinfo	Trigger information	0x7A4	RO
tcontrol	Global trigger control	0x7A5	R/W
mcontext	Trigger context	0x7A8	R/W

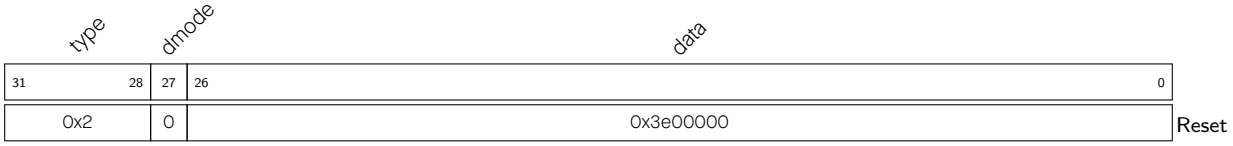
2.11.3.6 Register Description

Register 2.125. tselect (0x7A0)



tselect Configures the index (0-3) of the selected trigger unit. (R/W)

Register 2.126. tdata1 (0x7A1)

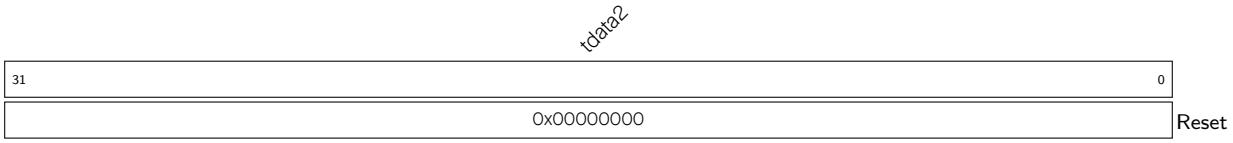


type Represents the trigger type. This field is reserved since only match type (0x2) triggers are supported. (RO)

dmode This is set to 1 if a trigger is being used by the debugger.
0: Both Debug and M mode can write the tdata1 and tdata2 registers at the selected tselect.
1: Only Debug Mode can write the tdata1 and tdata2 registers at the selected tselect. Writes from other modes are ignored.
Note: Only writable from debug mode.
(R/W)

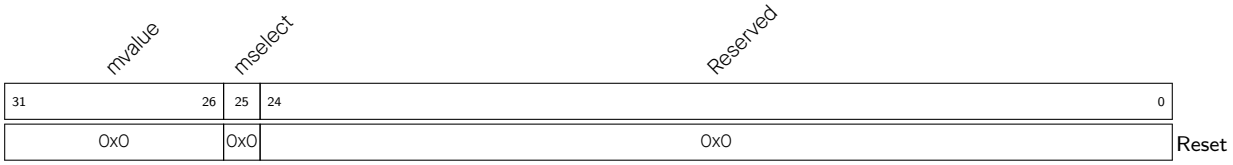
data Configures the abstract tdata1 content. This will always be interpreted as fields of mcontrol since only match type (0x2) triggers are supported. (R/W)

Register 2.127. tdata2 (0x7A2)



tdata2 Configures the abstract tdata2 content. This will always be interpreted as maddress since only match type (0x2) triggers are supported. (R/W)

Register 2.128. tdata3 (0x7A3)

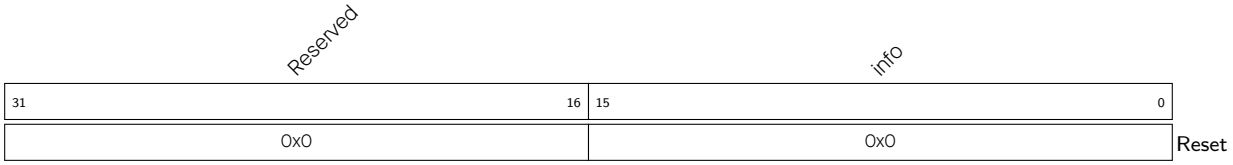


mvalue Data used together with mselect. (R/W)

mselect 0: Ignore mvalue.
1: This trigger will only match if the low bits of mcontext equal mvalue.
(R/W)

Reserved Read as 0. (RO)

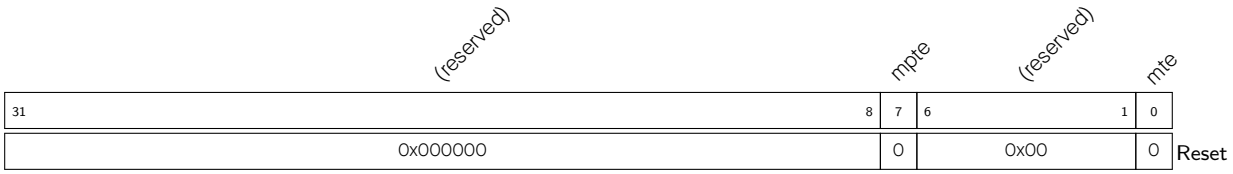
Register 2.129. tinfo (0x7A4)



Reserved Read as 0. (RO)

info One bit for each possible type enumerated in tdata1. Bit N corresponds to type N. If the bit is set, then that type is supported by the currently selected trigger. If the currently selected trigger does not exist, this field contains 1. If the type is not writable, this register may be unimplemented, in which case reading it causes an illegal instruction exception. In this case, the debugger can read the only supported type from tdata1. (RO)

Register 2.130. tcontrol (0x7A5)



mpte Configures whether to enable the machine mode previous trigger.

When CPU is taking a machine mode trap, the value of mte is automatically pushed into this.

When CPU is executing MRET, its value is popped back into mte, so this becomes 0.

(R/W)

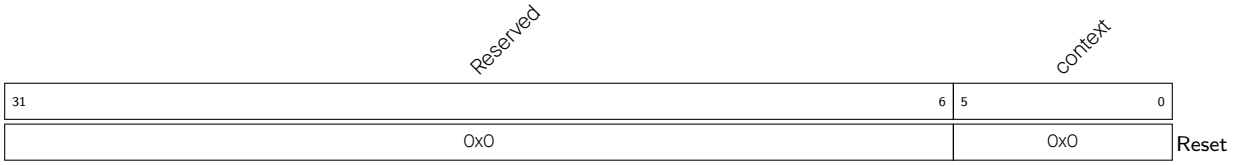
mte Configures whether to enable the machine mode trigger.

When CPU is taking a machine mode trap, its value is automatically pushed into mpte, so this becomes 0 and triggers with action=0 are disabled globally.

When CPU is executing MRET, the value of mpte is automatically popped back into this.

(R/W)

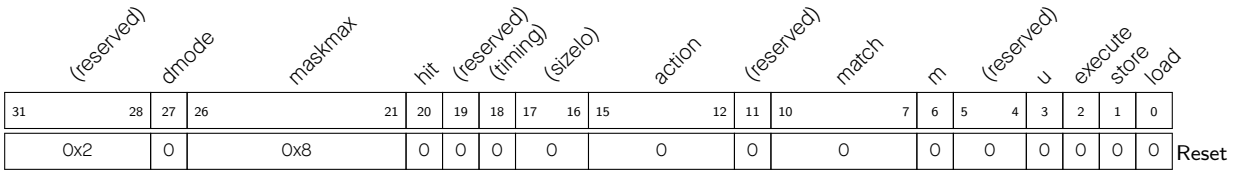
Register 2.131. MCONTEXT (0x7A8)



Reserved Read as 0. (RO)

context Writable in M mode and Debug Mode. Machine mode software can write a context number to this register, which can be used to set triggers that only fire in that specific context. (R/W)

Register 2.132. mcontrol (0x7A1)



dmode Same as dmode in tdata1. (RW*)

maskmax Represents the largest natural power-of-two (NAPOT) range supported when match is 1. It is always read as 0x8, which means only the lowest 8 bits can be masked. (RO)

hit This is found to be 1 if the selected trigger had fired previously. This bit needs to be cleared manually. (R/W)

timing Set this field to enable load/store trigger action timing.

0: Action for this trigger will be taken just before the instruction that triggered it is executed, but after all preceding instructions are committed.

1: Action for this trigger will be taken after the instruction that triggered it is executed. It should be taken before the next instruction is executed.

(R/W)

sizelo This field contains the lowest two bits of access address.

0x0: The trigger will attempt to match against an access of any size

0x1: The trigger will attempt to match against 8-bit memory accesses

0x2: The trigger will attempt to match against 16-bit memory accesses

0x3: The trigger will attempt to match against 32-bit memory accesses

(R/W)

action Configures the selected trigger to perform one of the available actions when firing. Valid options are:

0x0: cause breakpoint exception.

0x1: enter debug mode (only valid when dmode = 1)

0x2: Enable trace on trigger[0] hit

0x3: Enable trace on trigger[1] hit

0x4: Enable trace on trigger[2] hit

Note: Writing an invalid value will set this to the default value 0x0.

(R/W)

match Configures the selected trigger to perform one of the available matching operations on a data/instruction address. Valid options are:

0x0: exact byte match, i.e. address corresponding to one of the bytes in an access must match the value of maddress exactly.

0x1: NAPOT match, i.e. at least one of the bytes of an access must lie in the NAPOT region specified in maddress.

Note: Writing a larger value will clip it to the largest possible value 0x1.

(R/W)

Continued on the next page...

Register 2.132. mcontrol (0x7A1)

Continued from the previous page...

m Set this field to enable the selected trigger to operate in machine mode. (R/W)

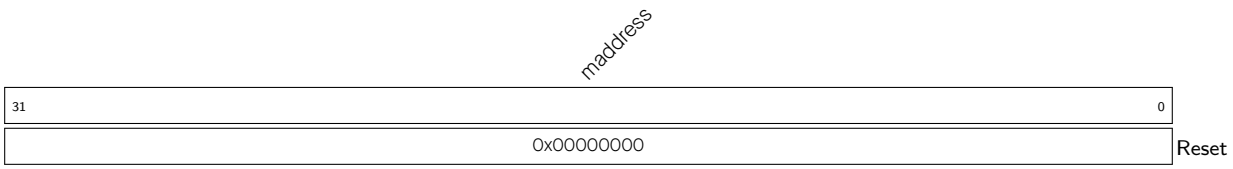
u Set this field to enable the selected trigger to operate in user mode. (R/W)

execute Set this field to enable the selected trigger to fire right before an instruction with the matching virtual address executed by the CPU. (R/W)

store Set this field to enable the selected trigger to fire right before a store operation with the matching data address executed by the CPU. (R/W)

load Set this field to enable the selected trigger to fire right before a load operation with the matching data address executed by the CPU. (R/W)

Register 2.133. maddress (0x7A2)



maddress Configures the address used by the selected trigger when performing a match operation. This is decoded as NAPOT when match=1 in mcontrol. (R/W)

2.11.4 Trace

2.11.4.1 Overview

In order to support non-intrusive software debug, the CPU core provides an instruction trace interface which provides relevant information for offline debug purpose. This interface provides relevant information to Trace Encoder block, which compresses the information and stores in memory allocated for it. Software decoders can read this information from trace memory without interrupting the CPU core and re-generate the actual program execution by the CPU core.

2.11.4.2 Features

The CPU core supports instruction trace feature and provides below information to Trace Encoder as mandated in RISC-V Processor Trace Version 1.0:

- Number of instructions being retired.
- Occurrence of exception and interrupt along with cause and trap values.
- Current privilege level of hart.
- Instruction type of retired instructions for jumps, branches and return.
- Instruction address for instructions retired before and after program counter changes.
- Trace enabling on trigger hit as per action bit in mcontrol.

2.11.4.3 Functional Description

Each HP core implements mandatory instruction delta tracing, also known as branch tracing. It works by tracking execution from a known start address by sending information about deltas taken by a program. Deltas are typically introduced by jump, call, return, branch type instructions and also by interrupts and exceptions. All such deltas along with additional details about cause and actual instructions/addresses are communicated over high bandwidth instruction trace interface output from the core. Trace Encoder operates on the information on this trace interface and compresses the information for storage in memory for offline debug by a decoder. More information about the encoding is available in Chapter 3 *RISC-V Trace Encoder (TRACE)*.

The core does not have any internal registers to provide control over instruction trace interface. All register controls are available in 3 *RISC-V Trace Encoder (TRACE)* block.

2.12 Performance

2.12.1 Branch Prediction

2.12.1.1 Overview

A core does branch prediction in terms of branch is taken or not taken and the target address of conditional/unconditional branch type instruction prior to its execution. Based on the prediction, the core starts fetching from the respective PC. Then, it validates the prediction on the execution of the instruction. If it is found to be miss-predicted, then the core flushes the pipeline and starts fetching from the correct PC. The prediction improves the throughput as the core continues fetching instructions based on prediction instead of waiting till the execution of branch instruction. The details about the prediction technique used by the core are described in the following section.

2.12.1.2 Features

- Prediction of both results and target address of conditional/unconditional branch type instructions
- Branch target prediction of instructions including BEQ, BNE, BLT, BLTU, BGE, BGEU, C.BEQZ, C.BNEZ, JAL, J, CJ
- Conditional branch prediction includes instructions such as: BEQ, BNE, BLT, BLTU, BGE, BGEU, C.BEQZ, C.BNEZ
- Uses 2-bit saturating counter for more accuracy
- Uses global history of conditional branches for prediction which improves performance
- Target prediction in one clock cycle for unconditional branch, that is, 0 cycle penalty when prediction or speculation is correct

2.12.1.3 Functional Description

The core uses Branch History Table (BHT) along with Branch Target Buffer (BTB) for prediction of branch and jump type instructions.

BHT is 512x16-bit SRAM memory for each core. It is indexed by Virtual Branch History Register (VBHR) while performing prediction and by Global Branch History Register (GBHR) while updating or correcting the result (taken or not taken) on execution of branch-type instructions. GBHR contains the results of the last 13 branch-type instructions executed. VBHR contains the results of the last 12 branch-type instructions and predicted results of the current branch instruction. Thus, both registers hold the trajectory of previously executed branches. Based on 13 bits of VBHR and GBHR, 2 bits of saturation counter value stored in BHT are either read or written respectively.

As shown in Figure 2.12-1, 2-bit saturation counter indicates if a branch is

- weakly not taken (00)
- strongly not taken (01)
- weakly taken (10)
- strongly taken (11)

If the result of prediction is weakly or strongly not taken, then the branch is predicted as not taken. Similar is the case for weakly/strongly taken.

While updating BHT with the actual result of branch execution, the following transition logic is used to update the respective saturation counter.

BTB stores the 32-bit branch target address. It can hold a maximum of 16 entries of target addresses. It is indexed by the last 16 bits of the PC of conditional/unconditional branch-type instruction. Thus, the core predicts the result of branch-type instructions based on BHT and jumps to the target obtained from BTB if the branch is predicted as taken. For unconditional branches, the core directly jumps to the target obtained from BTB, as it does not need prediction in this case. When the actual result and the target of branch-type instructions are obtained during execution, the core takes a corrective jump by flushing the pipeline in case of misprediction. BTB and BHT entries are corrected accordingly.

Figure 2.12-1 shows 2-bit saturation scheme details used in each HP core.

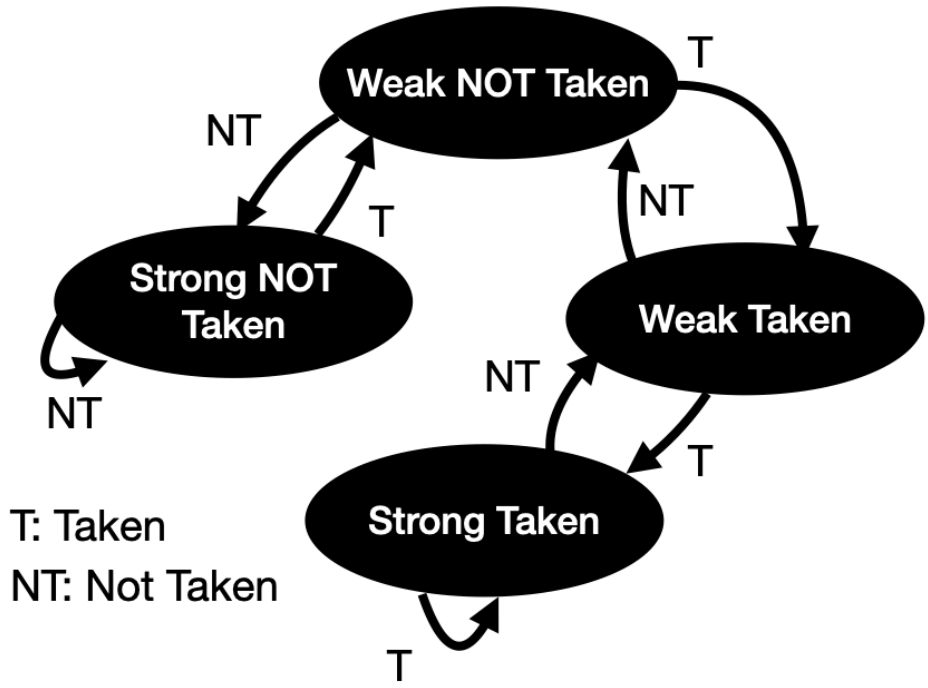


Figure 2.12-1. Two-bit Saturation Counter

2.12.2 RAS

2.12.2.1 Overview

Return Address Stack (RAS) is a stack maintained by the core to push the return addresses of jump and link (JAL/JALR) type instructions. Thus, the core can jump to the return address as soon as the RET instruction is pre-decoded, enhancing the performance of function call return.

2.12.2.2 Features

- Increased core performance by avoiding pipeline stall on return of function call

- Able to store up to four levels of function call nesting

2.12.2.3 Functional Description

When JAL/JALR/C.JALR instruction is pre-decoded by the core, then the 32-bit PC of the next instruction is pushed to RAS.

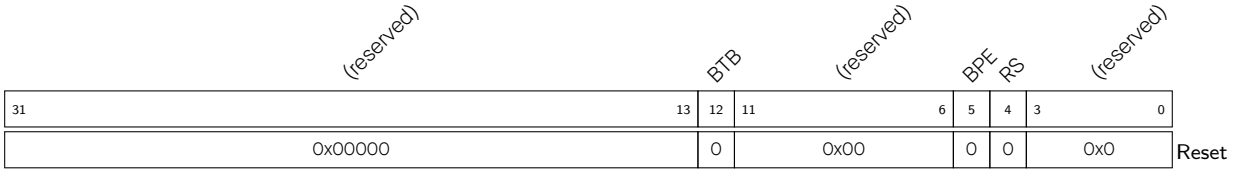
The address is popped out from RAS when the return type instruction (JALR, C.JR, C.JALR) is pre-decoded by the core. The core then starts fetching from the popped address. On execution of the return instruction, if the popped address does not match the address stored in the link register, then the core flushes the pipeline and starts to fetch from the address in the link register. RAS can store a maximum of four entries of return addresses, that is, four levels of function call nesting are supported.

RAS prediction depends on the following conditions:

Target Register (Rd)	Source Register (Rs)	RAS action
rd != x1	rs1 != x1	No action
rd != x1	rs1 == x1	POP
rd == x1	rs1 != x1	PUSH
rd == x1	rs1 == x1	PUSH and POP

2.12.3 Control Status Register for Performance Configuration

Register 2.134. mhcr (0x7C1)



RS Configures whether to enable the return address stack logic for prediction.
0: Disable
1: Enable
(R/W)

BPE Configures whether to enable conditional branch prediction.
0: Disable
1: Enable
(R/W)

BTB Configures whether to enable branch target buffer.
0: Disable
1: Enable
(R/W)

2.13 Custom Features

2.13.1 Bus Error Response

2.13.1.1 Overview

In the HP core, load/store bus errors are handled as asynchronous exceptions, which means that by the time such an exception is detected, the CPU may have executed some of the instructions after the actual load/store instruction that caused the exception.

Due to the asynchronous nature of these exceptions, handling them requires more context than that available via the standard CSRs mcause, mepc and mtval, which are otherwise sufficient for synchronous exceptions. Therefore, additional CSRs have been provided to allow software to gain all the necessary context required to handle and resume a program that has encountered an asynchronous bus-error exception.

In the following sections, these bus-error-specific CSRs are described and the recommended approach is to gain context about a bus-error exception.

2.13.1.2 Functional Description

There are two ways to handle load/store bus errors:

- Synchronous bus-errors mode:
  When this mode is enabled, (by setting mhint.SBE bit) bus-error exceptions can be handled just like other synchronous exceptions. Therefore, execution traps at the exact PC corresponding to the bus error causing load/store instruction, and mepc holds the same PC.
  The side effect of enabling this mode is that it increases the latency of all load/store instructions.
- Asynchronous bus-errors mode (default):
  - For handling bus errors in asynchronous mode, it is recommended (although not necessary) to clear the mexstatus.NMFT bit at startup.
  - In asynchronous mode, bus errors cause the CPU to enter trap handler with MCAUSE value 5 or 7 (depending upon the type of operation which caused the error), and mepc will hold the PC of the last instruction that was about to be executed when CPU detected the bus error(s) due to (one or more) previously executed load/store instruction(s).
  - Upon entering the exception handler for cause 5 or 7, the mexstatus.BUS_ERR bit must be checked to confirm if it is due to load/store bus-error. If so, this bit must be cleared to acknowledge.
  - Next, bits ldpc0.vld, ldpc1.vld, stpc0.vld, stpc1.vld and stpc2.vld must be checked to confirm if they are valid. If so, these must be cleared, and the corresponding address fields (ldpc0.addr, ldpc1.addr, stpc0.addr, stpc1.addr, and stpc2.addr) can be extracted from these CSRs to get the program counters at which these faulting load/store instructions were executed. The CSRs ldtval0, ldtval1, sttval0, sttval1 and sttval2 hold the memory access addresses corresponding to the respective loads/stores. Note that the lower indexed CSRs hold the newest bus errors, while the higher indexed CSRs hold the older bus errors.
  - Note that, even if the cause is either 5 or 7, both load and store bus error CSRs must be checked as sometimes there are more than one load/store instructions (or misaligned accesses which are split into multiple loads/stores) that have been executed and are in flight. Prior to entering the trap

handler for the first encountered bus error, the CPU will allow all pending memory operations to finish and record any bus errors caused by them, thus it is important to check all the CSRs to make sure that all such exceptions were caught and handled.

- Also note that recovering a normal program execution after encountering a bus-error exception is not as simple as for synchronous cases, since program execution may have proceeded quite far ahead of the bus-error-causing instructions.

2.13.1.3 Control Status Registers

Name	Description	Address	Access
mexstatus	Machine extension control and status register.	0x7E1	R/W
mhint	Machine implicit operation register	0x7C5	R/W
ldpc0	Load bus error PC register 0	0xBE0	R/W
ldpc1	Load bus error PC register 1	0xBE1	R/W
ldtval0	Load bus error access address register 0	0xBE8	R/W
ldtval1	Load bus error access address register 1	0xBE9	R/W
stpc0	Store bus error PC register 0	0xBF0	R/W
stpc1	Store bus error PC register 1	0xBF1	R/W
stpc2	Store bus error PC register 2	0xBF2	R/W
sttval0	Store bus error access address register 0	0xBF8	R/W
sttval1	Store bus error access address register 1	0xBF9	R/W
sttval2	Store bus error access address register 2	0xBFA	R/W

2.13.2 Dedicated IO

2.13.2.1 Overview

Normally, GPIOs are an APB peripheral, which means that changes to outputs and reads from inputs can get stuck in write buffers or behind other transfers. They are slower because the APB bus runs at a lower speed than the CPU core. As an alternative, the CPU core implements I/O processor-specific CPU registers (CSRs) which are directly connected to the GPIO matrix or IO pads. As these registers can be accessed in one instruction, the speed is fast.

2.13.2.2 Features

- 8 dedicated IOs directly mapped on GPIOs
- No latency for driving output ports
- Two CPU cycle latency for sensing input values

2.13.2.3 Functional Description

The CPU core has a set of 8 inputs and outputs (pin value + pin output enable). These input and output ports are directly connected to the GPIO matrix, through which they can be mapped on top-level pads. Please refer to Chapter 10 *GPIO Matrix and IO MUX* for more details.

The CPU implements three custom CSRs:

- GPIO_IN is read-only and reflects the input value.
- GPIO_OUT is R/W and reflects the output value for the GPIOs.
- GPIO_OEN is R/W and reflects the output enable state for the GPIOs. It controls the pad direction. Programming high would mean the pad should be configured in output mode. Programming low means it should be configured in input mode.

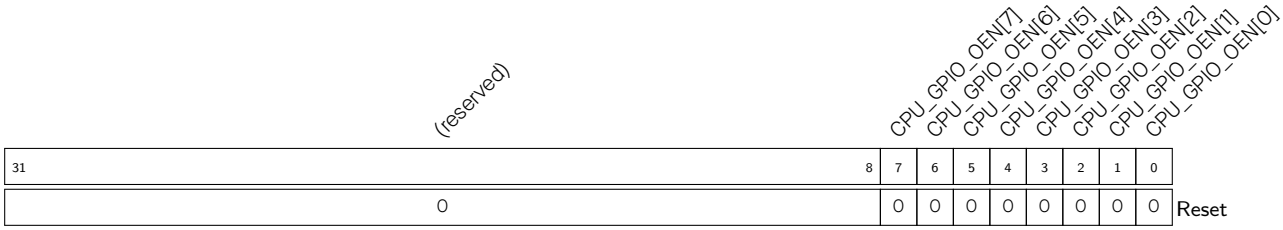
2.13.2.4 Register Summary

Below is a list of custom dedicated IO CSRs implemented inside the core.

Name	Description	Address	Access
cpu_gpio_oen	GPIO Output Enable	0x803	R/W
cpu_gpio_in	GPIO Input Value	0x804	RO
cpu_gpio_out	GPIO Output Value	0x805	R/W

2.13.2.5 Register Description

Register 2.135. cpu_gpio_oen (0x803)



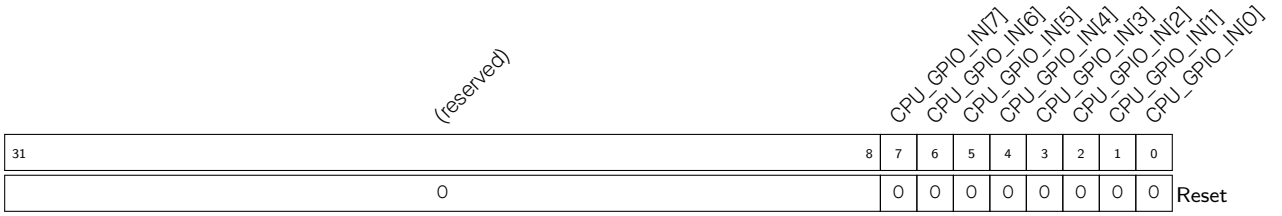
CPU_GPIO_OEN Configures whether to enable GPIOn (n=0 ~ 21) output. CPU_GPIO_OEN[7:0] correspond to output enable signals cpu_gpio_out_oen[7:0] in Table 10.12-1 *Peripheral Signals via GPIO Matrix*. CPU_GPIO_OEN value matches that of cpu_gpio_out_oen. CPU_GPIO_OEN is the enable signal of CPU_GPIO_OUT.

0: Disable GPIO output

1: Enable GPIO output

(R/W)

Register 2.136. cpu_gpio_in (0x804)



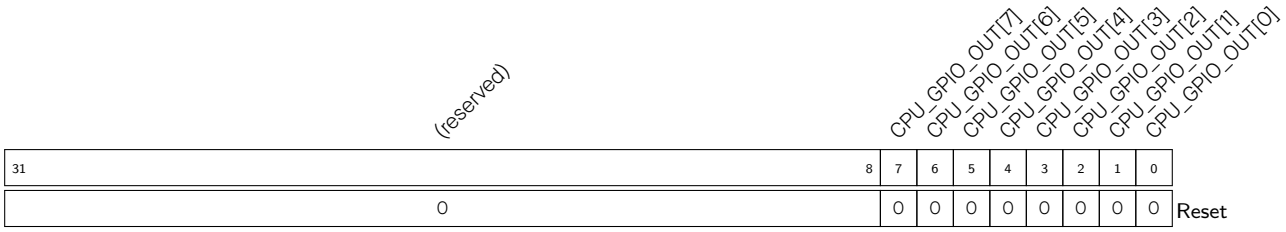
CPU_GPIO_IN Represents GPIOn (n=0 ~ 21) input value. It is a CPU CSR to read input value (1=high, 0=low) from SoC GPIO pin.

CPU_GPIO_IN[7:0] correspond to input signals cpu_gpio_in[7:0] in Table 10.12-1 *Peripheral Signals via GPIO Matrix* .

CPU_GPIO_IN[7:0] can only be mapped to GPIO pins through GPIO matrix. For details please refer to Section 10.4 in Chapter *GPIO Matrix and IO MUX*.

(RO)

Register 2.137. cpu_gpio_out (0x805)



CPU_GPIO_OUT Configures GPIOn (n=0 ~ 21) output value. It is a CPU CSR to write value (1=high, 0=low) to SoC GPIO pin. The value takes effect only when CPU_GPIO_OEN is set.

CPU_GPIO_OUT[7:0] correspond to output signals cpu_gpio_out[7:0] in Table 10.12-1 *Peripheral Signals via GPIO Matrix*.

CPU_GPIO_OUT[7:0] can only be mapped to GPIO pins through GPIO matrix. For details please refer to Section 10.5 in Chapter *GPIO Matrix and IO MUX*.

(R/W)

2.13.3 RunStall Support

2.13.3.1 Overview

The RunStall signal allows an external agent to stall the processor and shut off much of the clock tree to save operating power. The RunStall input allows external logic to stall the processor's internal pipeline. Any in-progress instructions in the pipeline will complete after the assertion of the RunStall input line. After the pipeline is stalled, the core outputs a handshake signal "corestalled" to indicate successful stalling of the CPU pipeline.

2.13.3.2 Functional Description

The RunStall input can be used in the following two scenarios:

- As a mechanism to initialize instruction and data RAMs after a reset. If the RunStall input is asserted during reset and remains asserted after the reset is removed, the processor will be in a stalled state. Instruction and data RAMs can be initialized and after these RAMs have been initialized, the RunStall can be released, allowing the processor to start code execution. This mechanism allows the processor's reset vector to point to an address in local instruction RAM, and the reset code can be written to the instruction RAM by an external agent before the processor starts executing code.
- To freeze the processor's internal pipeline. Assertion of RunStall reduces the processor's active power dissipation without using or requiring interrupts and the WAITI option. However, the power savings resulting from use of the RunStall signal will not be as great as for those using WAITI, because the RunStall logic allows some portions of the processor to be active even though the pipeline is stalled.

Figure 2.13-1 shows the timing diagram for RunStall and CoreStalled signals and CPU core state transition.

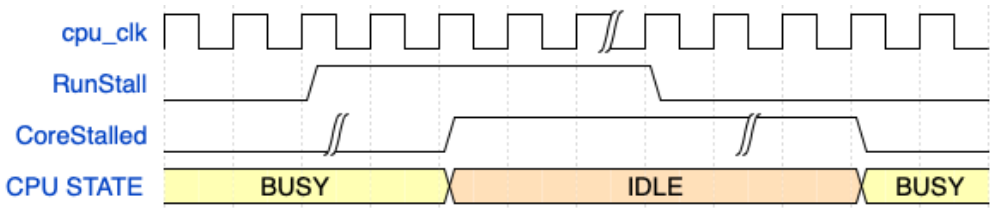


Figure 2.13-1. CPU Core State Transition Using RunStall

2.13.3.3 Register Summary

The SoC registers in Section 21.2.1.10 can be used to assert RunStall to HP cores.

2.13.4 Debug Assist Information

The HP CPU core complex provides additional debug information to the Debug Assistant module in ESP32-P4. For details, please refer to Chapter *22 Debug Assistant*.

2.13.5 Core Lock-up

2.13.5.1 Overview

Currently, there is no exception enable register defined in the RISC-V programming model, and when returning from the exception service program to the normal program flow, the exception vector number in the MCAUSE register is not cleared, so software developers cannot easily know whether the CPU is currently processing an exception or running a normal program track by the registers defined in the programming model. To address this concern, a custom LOCKUP field has been implemented. It is readable through the custom CSR.

2.13.5.2 Functional Description

When the CPU responds to an exception, it will determine whether the current program flow is in the exception service program (through EXPT_VLD in MEXSTATUS field). If the CPU is processing an exception and triggers a new exception before returning from the exception service program, the CPU will be locked. When CPU is locked:

- the lock-up signal is set high to inform the SoC that the CPU is in a locked state
- the CPU stops instruction fetch and execution
- the CPU keeps the PC at the instruction PC that triggers the CPU lock
- the LOCKUP field in the MEXSTATUS register is set to 1

In a locked-up state, a debugger can still break in and execute debug commands. It is recommended to reset the core when a lock state is detected.

There are some exceptions to the lock-up rule. Exceptions caused by ECALL or EBREAK, when nested in any exception type, will not trigger CPU lock-up.

2.13.5.3 Register Summary

The mexstatus.LOCKUP bit provides lock-up status of individual core.

2.13.5.4 Register Description

Please refer to mexstatus CSR.

2.13.6 External Wakeup

2.13.6.1 Overview

Each core features a dedicated input port for wake-up via an external event. This functionality is controlled and configured through registers in the SoC.

2.13.6.2 Functional Description

To save power, each CPU core can enter a sleep state by executing the WFI instruction. It can be awakened by three types of events: an interrupt, a halt request, or a signal from its dedicated wake-up input port. A high-level pulse lasting a single clock cycle on this dedicated port is sufficient to wake the corresponding core.

For details on how to configure this port, please refer to Chapter *21 System Registers (SYSREG)*.

Chapter 3

RISC-V Trace Encoder (TRACE)

The high-performance CPU (HP CPU) of ESP32-P4 supports the instruction trace interface through the trace encoder. The trace encoder connects to HP CPU's instruction trace interface, compresses the information into smaller packets, and then stores the packets into internal SRAM (see Chapter 9 *System and Memory*).

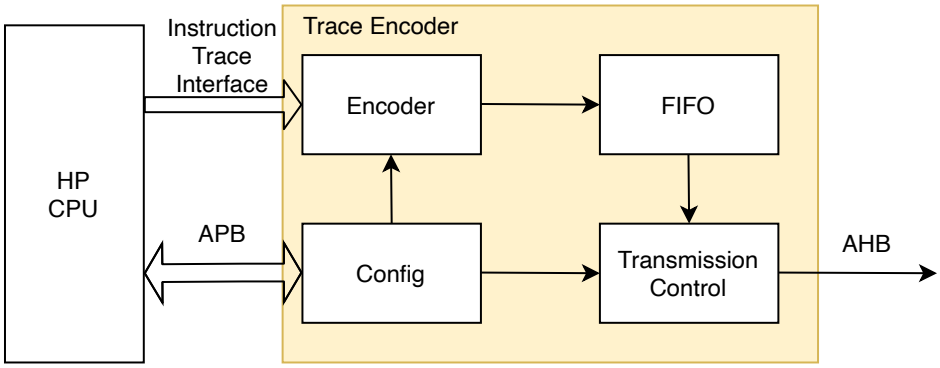


Figure 3.0-1. Trace Encoder Overview

3.1 Terminology

To better illustrate the functions of the RISC-V Trace Encoder, the following terms are used in this chapter.

hart	RISC-V hardware thread
branch	an instruction which conditionally changes the execution flow
uninferable discontinuity (updiscon)	a program counter change that can not be inferred from the program binary alone
delta	a change in the program counter that is other than the difference between two instructions placed consecutively in memory
trap	the transfer of control to a trap handler caused by either an exception or an interrupt
qualification	an instruction that meets the filtering criteria passes the qualification, and will be traced
te_inst	the name of the packet type emitted by the encoder
retire	the final stage of executing an instruction, when the machine state is updated
EPC	exception program counter

3.2 Introduction

In complex systems, understanding program execution flow is not straightforward. This may be due to a number of factors, for example, interactions with other cores, peripherals, real-time events, poor implementations, or some combination of all of the above.

It is hard to use a debugger to monitor the program execution flow of a running system in real time, as this is intrusive and might affect the running state. However, it is important to provide the visibility of program execution.

That is where instruction trace comes in, which provides trace of the program execution.

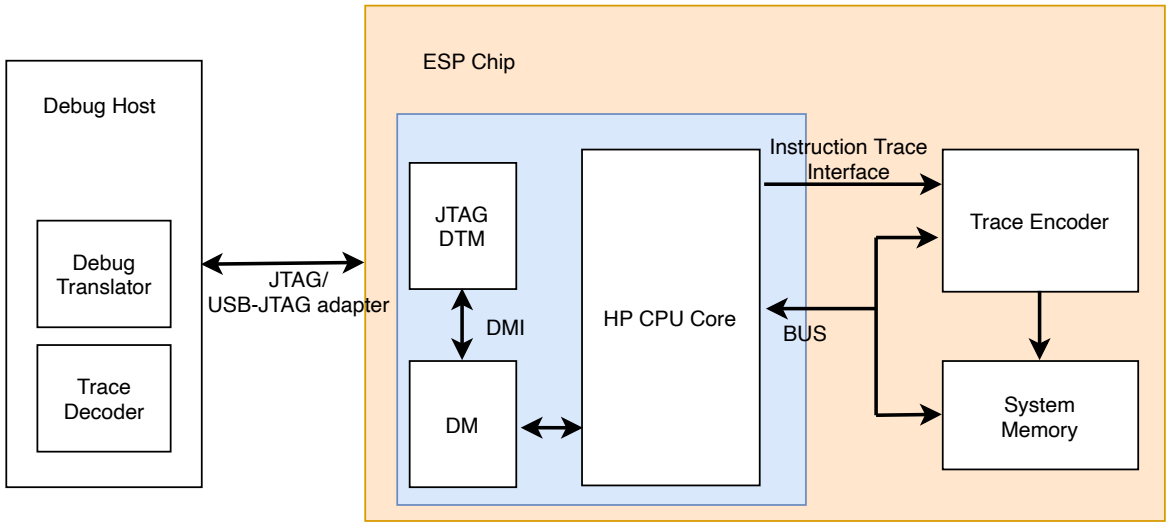


Figure 3.2-1. Trace Flow Overview

Figure 3.2-1 shows the instruction delta trace flow. Each HP Core has a private encoder that can work independently, and the trace flow of each core is the same:

- The HP CPU core provides an instruction trace interface that outputs the instruction information executed by the HP CPU. Such information includes instruction address, instruction type, etc. For more details about ESP32-P4 HP CPU's instruction trace interface, please refer to Chapter 2 *High-Performance CPU*.
- The trace encoder collects the relevant instruction trace information from HP CPU's instruction trace interface, compresses the information into lower bandwidth packets, and then stores the packets in system memory.
- The debugger (debug host) can dump the trace packets from the system memory via JTAG or USB Serial/JTAG, and use a decoder to decompress and reconstruct the program execution flow. The trace decoder, usually software on an external PC, takes in the trace packets and reconstructs the program instruction flow with the program binary that runs on the originating hart. This decoding step can be done offline or in real time while the hart is executing.

3.3 Features

- Compatible with Efficient Trace for RISC-V Version 2.0. See Table 3.3-1 for the implemented parameters
- Support for delta address mode and full address mode
- Support for a filter unit
- Support for notifying an instruction address via debug trigger or filter unit
- Support for the following sideband signals to control trace data flow:
  - Support for debugging trigger to start or end encoder
  - When the hart is halted, the encoder can report the last packet and then stop
  - When the hart is reset, the encoder can report the last packet and then stop
  - Support for stalling the hart when FIFO is almost full
- Arbitrary address range of the trace memory size
- Configurable synchronization modes:
  - Synchronization counter counts by packet
  - Synchronization counter counts by cycle
  - Synchronization counter can be disabled
- Trace lost status to indicate packet loss
- Automatic restart after packet loss
- Memory writing in the loop or non-loop mode
- Support two interrupts:
  - Triggered when the packet size exceeds the configured memory space
  - Triggered when a packet is lost
- FIFO (128 × 8 bits) to buffer packets
- AHB burst transmission with configurable burst length

Table 3.3-1. Trace Encoder Parameters

Parameter Name	Value	Description
arch_p	0	Initial version
bpred_size_p	0	Branch prediction mode is not supported
cache_size_p	0	Jump target cache mode is not supported
call_counter_size_p	0	Implicit return mode is not supported
ctype_width_p	0	Width of the ctype bus
context_width_p	0	Width of context bus
ecause_width_p	6	Width of exception cause
ecause_choice_p	0	Multiple choice is not supported
f0s_width_p	0	Format 0 packets are not supported
filter_context_p	0	Filtering on context is not supported

Parameter Name	Value	Description
filter_excint_p	1	Filtering on exception cause or interrupt is supported
filter_privilege_p	1	Filtering on privilege is supported
filter_tval_p	1	Filtering on trap value is supported
iaddress_lsb_p	1	Compressed instructions are supported
iaddress_width_p	32	The instruction bus is 32-bit
iretire_width_p	1	Width of the iretire bus
ilastsize_width_p	0	Width of the ilastsize
itype_width_p	3	Width of the itype bus
nocontext_p	1	Exclude context from te_inst packets
notime_p	1	Exclude time from te_inst packets
privilege_width_p	1	Only machine and user mode are supported
retires_p	1	Maximum number of instructions that can be retired per block
return_stack_size_p	0	Implicit return mode is not supported
sijump_p	0	Sequentially inferable jump mode is not supported
taken_branches_p	1	Only one instruction retired per cycle
impdef_width_p	0	Not implemented

For detailed descriptions of the above parameters, please refer to the Efficient Trace for RISC-V Version 2.0 > Chapter Parameters and Discovery.

3.4 Architectural Overview

This chapter mainly introduces the implementation details of ESP32-P4's trace encoder.

As shown in Figure 3.0-1, the trace encoder contains an encoder, a FIFO, a register configuration module, and a transmission control module.

The encoder receives the HP CPU's instruction information via the instruction trace interface, compresses it into different packets, and writes it to the internal FIFO.

The transmission control module writes the data in the FIFO to the internal SRAM through the AHB bus.

The FIFO is 128 deep and 8-bit wide. When the memory bandwidth is insufficient, the FIFO may overflow, and packet loss occurs. If a packet is lost, the encoder will send a packet to tell that a packet is lost, stop working until the FIFO is empty, and also trigger an interrupt meanwhile if the interrupt is enabled.

3.5 Functional Description

3.5.1 Synchronization

In order to make the trace robust there must be regular synchronization points within the trace. Synchronization is accomplished by sending a full valued instruction address. When the synchronization counter value reaches the value of the TRACE_RESYNC_PROLONGED field of the TRACE_RESYNC_PROLONGED_REG register, the encoder will send a synchronization packet (format 3 subformat 0, see Section 3.6.3.1).

The synchronization counter is configured via TRACE_RESYNC_MODE:

- 0: Disable the synchronization counter
- 1: Invalid. No effect
- 2: Synchronization counter counts by packet
- 3: Synchronization counter counts by cycle

You can adjust the trace bandwidth by increasing the value of TRACE_RESYNC_PROLONGED to reduce the frequency of sending synchronization packets, thereby reducing the bandwidth occupied by packets.

3.5.2 Address Mode

ESP32-P4 supports two address modes: delta address mode and full address mode.

- Delta address mode (default): In delta address mode, addresses are encoded as the difference between the actual address of the current instruction and the actual address of the instruction reported in the previous packet that contained an address. This differential encoding requires fewer bits than the full address, and thus results in more efficient trace compression.
- Full address mode: In full address mode, all addresses in the trace are encoded as absolute addresses instead of in differential form. This kind of encoding is always less efficient, but it can be a useful debugging aid for software decoder developers. This mode is enabled by setting TRACE_FULL_ADDRESS.

3.5.3 Optional Sideband Signals

The Efficient Trace for RISC-V Version 2.0 > Chapter Optional Sideband Signals has provided some optional sideband signals for encoder control. ESP32-P4 has implemented the following functions:

Table 3.5-1. Optional sideband signals

Signal	Function
trigger[2:0]	• A pulse on bit 0 will cause the encoder to start tracing, and continue until further notice, subject to other filtering criteria also being met. • A pulse on bit 1 will cause the encoder to stop tracing until further notice. • A pulse on bit 2 will cause the encoder to report a specific address. This function is enabled by setting TRACE_DM_TRIGGER_ENA. The trigger signal is asserted upon a trigger match. For example, if the action field of mcontrol is 2, while the trigger matched it will assert trigger bit0; if the action is 3, while the trigger matched it will assert trigger bit1. For details, please refer to Table 3.8-1.
halted	Hart is halted. Upon assertion, the encoder will output a packet to report the address of the last instruction retired before halting, followed by a support packet to indicate that tracing has stopped. Upon deassertion, the encoder will start tracing again, commencing with a synchronization packet. This function is enabled by setting TRACE_HALT_ENA.

Signal	Function
reset	Hart is in reset. Behavior is as described above for halted.
stall	Stall request to hart. Some applications may require lossless trace, which can be achieved by using this signal to stall the hart if the trace encoder is unable to output a trace packet (internal FIFO almost full). This function is enabled by setting TRACE_STALL_ENA.

3.5.4 Filtering

Filtering provides a mechanism to control the criteria for encoder to produce a trace. For example, it may be desirable to trace:

- When the instruction address is within a particular range
- Starting from one instruction address and continue until a second instruction address
- For one or more specified privilege levels
- Exception and/or interrupt handlers for specified exception causes

The filtering unit has a filter and a comparator unit.

Each comparator unit is actually a pair of comparators (Primary and Secondary, or P, S) allowing a bounded range to be matched with a single unit if required, and offers:

- Input selected from iaddress and tval
- A range of arithmetic options ($<$, $>$, $=$, $!=$, etc.) independently selectable for each comparator
- Secondary match value may be used as a mask for the primary comparator
- The two comparators can be combined in several ways: P, $P\&\&S$, $!(P\&\&S)$, latch (set on P clear on S)
- Each comparator can also be used to explicitly report a particular instruction address

Each filter can specify filtering against instruction from the hart, and offers:

- One run-time selectable comparator unit to match
- Multiple choice selection for priv and cause inputs
- Select matching for interrupt

3.5.5 Anchor Tag

Since the length of data packets is variable, in order to identify boundaries between data packets when packed packets are written to memory, the ESP32-P4 encoder inserts zero bytes between data packets:

- The maximum packet length is 13 bytes, so a sequence of at least 14 zero bytes cannot occur within a normal packet. Therefore, the first non-zero byte seen after a sequence of at least 14 zero bytes must be the first byte of a packet.
- Every time when 128 packets are transmitted, the encoder writes 14 zero bytes to the memory partition boundary as anchor tags.

3.5.6 Memory Writing Mode

When writing the trace memory, the size of the trace packets might exceed the capacity of the memory. In this case, you can choose whether to wrap around the trace memory or not by configuring the memory writing mode:

- Loop mode: When the size of the trace packets exceeds the capacity of the trace memory (namely when TRACE_MEM_CURRENT_ADDR_REG reaches the value of TRACE_MEM_END_ADDR_REG), the trace memory is wrapped around, so that the encoder loops back to the memory's starting address TRACE_MEM_START_ADDR_REG, and old data in the memory will be overwritten by new data.
- Non-loop mode: When the size of the trace packets exceeds the capacity of the trace memory, the trace memory is not wrapped around. The encoder stops at TRACE_MEM_END_ADDR_REG, and old data will be retained.

3.5.7 Automatic Restart

When packets are lost due to FIFO overflow, the encoder will stop working and need to be resumed by software manually or restarted by hardware automatically. If the TRACE_RESTART_ENA bit of TRACE_TRIGGER_REG is set, once the FIFO is empty, the encoder can automatically be restarted.

If the automatic restart feature is enabled, the encoder will be restarted in any case. Therefore, to disable the encoder, the automatic restart feature must be disabled first by clearing the TRACE_RESTART_ENA bit of the TRACE_TRIGGER_REG register.

3.6 Encoder Output Packets

This section mainly introduces ESP32-P4 trace encoder output packet format.

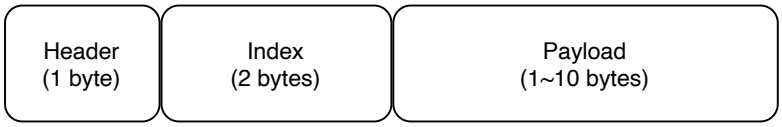


Figure 3.6-1. Trace packet Format

A packet includes header, index and payload. Header, index and payload are transmitted sequentially in bit stream form, from the fields listed at the top of tables below to the fields listed at the bottom. If a field consists of multiple bits, then the least significant bit is transmitted first.

3.6.1 Header

Header is 1-byte long. The format of header is shown in Table 3.6-1.

Table 3.6-1. Header Format

Field	Bits	Description	Value
length	5	Length of whole packet	4 ~ 13
flow	2	Reserved	0
timestamp	1	Reserved	0

3.6.2 Index

Index has 2 bytes. The format of index is shown in Table 3.6-2.

Table 3.6-2. Index Format

Field	Bits	Description	Value
index	16	The index of each packet	0~65535

3.6.3 Payload

The length of payload ranges from 1 byte to 10 bytes.

3.6.3.1 Format 3 Packets

Format 3 packets are used for synchronization, and report supporting information. There are 4 subformats defined in the specification. ESP32-P4 only supports 3 of them.

Format 3 Subformat 0 - Synchronization

This packet contains all the information the decoder needs to fully identify an instruction. It is sent for the first traced instruction (unless that instruction also happens to be the first one in an exception handler), and when synchronization has been scheduled by expiry of the synchronization timer. The payload length is 5 bytes.

Table 3.6-3. Packet format 3 subformat 0

Field name	Bits	Description
format	2	11 (sync): Synchronization
subformat	2	00 (start): Start of tracing, or resync
branch	1	Set to 0 if the address points to a branch instruction, and the branch was taken. Set to 1 if the instruction is not a branch or if the branch is not taken.
privilege	1	The privilege level of the reported instruction
address	31	Full instruction address. The address must be left shifted 1 bit in order to recreate the original byte address.
sign_extend	3	Reserved

Format 3 Subformat 1 - Exception

This packet also contains all the information the decoder needs to fully identify an instruction. It is sent following an exception or interrupt, and includes the cause, the 'trap value' (for exceptions), and the address

of the trap handler or of the exception itself. The length is variable: if the interrupt field is 1, the tvalepc field is omitted and the length is 6 bytes, otherwise the length is 10 bytes.

Table 3.6-4. Packet format 3 subformat 1

Field name	Bits	Description
format	2	11 (sync): Synchronization
subformat	2	01 (exception): Exception cause and trap handler address
branch	1	Set to 0 if the address points to a branch instruction, and the branch was taken. Set to 1 if the instruction is not a branch or if the branch is not taken.
privilege	1	The privilege level of the reported instruction
ecause	6	Exception cause
interrupt	1	Interrupt
theaddr	1	When set to 1, the address field points to the trap handler address. When set to 0, the address field points to the EPC (exception program counter) for an exception at the target for an updiscon, and is undefined for other exceptions and interrupts.
address	31	Full instruction address. The value of this field must be left shifted 1 bit in order to recreate the original byte address.
tvalepc	32	Value from appropriate utval/stval/mtval CSR (control/status register). Omitted if the interrupt is 1
sign_extend	3	Reserved

Format 3 Subformat 3 - Support

This packet provides supporting information to aid the decoder. It is issued when the trace is ended, filter match is failed, or a packet is lost. The length is 2 bytes.

Table 3.6-5. Packet format 3 subformat 3

Field name	Bits	Description
format	2	11 (sync): Synchronization
subformat	2	11 (support): Supporting information for the decoder
enable	1	Indicates if the encoder is enabled
encoder_mode	1	Always 0. Only supported by branch trace
qual_status	2	Indicates qualification status: • 00 (no_change): No change to filter qualification • 01 (ended_rep): Qualification ended, preceding instruction sent explicitly to indicate last qualification instruction • 10 (trace lost): One or more packets lost • 11 (ended_upd): Qualification ended, preceding te_inst would have been sent anyway due to an updiscon, even if wasn't the last qualified instruction

Field name	Bits	Description
ioptions	6	Indicates optional modes: • sequentially inferred jumps (Bits 5): When set to 1, the targets of sequentially inferable jumps will not be reported. (This feature is not implemented, so bit 5 should always be 0) • branch prediction (Bit 4): When set to 1, the branch prediction is enabled. (This feature is not implemented, so bit 4 should always be 0) • jump target cache (Bit 3): When set to 1, the jump target cache is enabled. (This feature is not implemented, so bit 3 should always be 0) • full address (Bit 2): – 0: delta address mode – 1: full address mode • implicit exception (Bit 1): Exclude address from format 3 subformat 1 packet if trap vector can be determined from ecause. (This feature is not implemented, so bit 1 should always be 0) • implicit return (Bit 0): When set to 1, function return addresses will not be reported. (This feature is not implemented, bit 0 should always be 0)
sign_extend	2	Reserved

3.6.3.2 Format 2 Packets

This packet contains only an instruction address, and is used when the address of an instruction must be reported (for example if the instruction is the target for an updiscon, or is the last instruction before exception), and there is no reported branch information.

The length is variable if delta address mode is enabled, ranging from 5 bytes to 8 bytes. The address field can be 8/16/24/32 bits, inferred from the packet length. For example, if the header length is n bytes, the address bits are $n - 4$ bytes. A sign bit is required to extend the address field to 32 bits.

Table 3.6-6. Packet format 2

Field name	Bits	Description
format	2	10 (addr-only): No branch information
notify	1	If the value of this bit is different from the MSB of address, it indicates that this packet is reporting an instruction that is not the target of an uninferable discontinuity because a notification was requested via trigger[2] or a filter matched.
updiscon	1	If the value of this bit is different from notify, it indicates that this packet is reporting the instruction following an uninferable discontinuity and is also the instruction before an exception, privilege change or resync.

Field name	Bits	Description
zero_extend	4	Reserved
address	Variable	Delta instruction address. These bits can be 8/16/24/32.

3.6.3.3 Format 1 Packets

This packet includes branch information, and is used when either the branch information must be reported (for example because the branch map is full), or when the address of instruction must be reported, and there has been at least one branch since the previous packet.

The packet length is variable if delta address mode is enabled, ranging from 5 bytes to 12 bytes. The address field can be 8/16/24/32 bits, inferred from the packet length. For example:

- The header length is n bytes
- The value of the format field is 1
- The value of the branch field is:
  - 0: the packet is format 1 without address
  - 1: branch_map is 1 bit wide, zero_ext is 6 bits wide, and address is $n - 5$ bits wide
  - 2 ~ 3: branch_map is 3 bits wide, zero_ext is 4 bits wide, and address is $n - 5$ bits wide
  - 4 ~ 7: branch_map is 7 bits wide, zero_ext is 0 bits wide, and address is $n - 5$ bits wide
  - 8 ~ 15: branch_map is 15 bits wide, zero_ext is 0 bits wide, and address is $n - 6$ bits wide
  - 16 ~ 31: branch_map is 31 bits wide, zero_ext is 0 bits wide, and address is $n - 8$ bits wide

Format 1 - address, branch_map

The length is variable.

Table 3.6-7. Packet format 1 with address

Field name	Bits	Description
format	2	01: Includes branch information
branches	5	Number of valid bits branch_map. The number of bits of branch_map is determined as follows: • 0: (cannot occur for this format) • 1: 1 bit • 2 ~ 3: 3 bits • 4 ~ 7: 7 bits • 8 ~ 15: 15 bits • 16 ~ 31: 31 bits For example if branches = 12, branch_map is 15-bit long, and the 12 LSBs are valid.

Field name	Bits	Description
branch_map	Variable	An array of bits indicating whether branches are taken or not. Bit 0 represents the oldest branch instruction executed. For each bit: • 0: branch taken • 1: branch not taken The field Bits is variable and determined by the branches field.
notify	1	If the value of this bit is different from the MSB of address, it indicates that this packet is reporting an instruction that is not the target of an uninferable discontinuity because a notification was requested via trigger[2] or a filter matched.
updiscon	1	If the value of this bit is different from notify, it indicates that this packet is reporting the instruction following an uninferable discontinuity and is also the instruction before an exception, privilege change or resync.
zero_ext	Variable	The length of this field is determined by branches as follows: • 1: 6 bits • 2 ~ 3: 4 bits • 4 ~ 31: 0 bits
address	Variable	Delta instruction address. These bits can be 8/16/24/32

Format 1 - no address, branch_map

The length is 5 bytes.

Table 3.6-8. Packet format 1 without address

Field name	Bits	Description
format	2	01: includes branch information
branches	5	Number of valid bits in branch_map. The length of branch_map is 31 bits. Only 0 valid.
branch_map	31	An array of bits indicating whether branches are taken or not. Bit 0 represents the oldest branch instruction executed. For each bit: • 0: branch taken • 1: branch not taken
sign_extend	2	Reserved

3.7 Interrupt

ESP32-P4's CORE*n*_TRACE (i.e., the trace encoder of core 0 or core 1) can generate the CORE*n*_TRACE_INTR interrupt signal that will be sent to the Interrupt Matrix.

There are several internal interrupt sources from CORE*n*_TRACE that can generate the above interrupt signal. The interrupt sources from CORE*n*_TRACE are listed with their trigger conditions and the resulted interrupt signals in Table 3.7-1.

Table 3.7-1. CORE*n*_TRACE's Internal Interrupt Sources

Internal Interrupt Source	Trigger Condition	Interrupt Signal
TRACE_MEM_FULL_INTR	Triggered when the packet size exceeds the capacity of the trace memory, namely when TRACE_MEM_CURRENT_ADDR_REG reaches the value of TRACE_MEM_END_ADDR_REG. If necessary, this interrupt can be enabled to notify the HP CPU for processing, such as applying for a new memory space again	CORE*n*_TRACE_INTR
TRACE_FIFO_OVERFLOW_INTR	Triggered when the internal FIFO overflows and one or more packets have been lost.	CORE*n*_TRACE_INTR

Note:
For definitions of *interrupt*, *interrupt signal*, *interrupt source*, and their correlations, please refer to Chapter 13 *Interrupt Matrix* > Section 13.2 *Interrupt Terminology in ESP32-P4*.

Each interrupt source can be configured by a common set of registers that are described in Section *Interrupt Configuration Registers*. The specific registers can be found in Section 3.9 *Register Summary*.

3.8 Programming Procedures

3.8.1 Encoder Option Configuration

The Encoder has some optional configuration options.

- Full address mode
  - The encoder uses delta address mode by default. Users can set TRACE_FULL_ADDRESS to enable full address mode for debugging.
- Restart after packet loss
  - Once the trace packet is lost due to FIFO overflow, the encoder automatically stops. If TRACE_RESTART_ENA is set, the encoder will automatically restart once the FIFO is empty.
- Stall CPU while FIFO is almost full
  - If TRACE_STALL_ENA is set, when the FIFO is almost full, a stall request will be asserted to the CPU, and the CPU will be stalled until the request is deasserted.
- Stop tracing when the CPU is halted

  If TRACE_HALT_ENA is set, when the CPU is halted, the encoder will output a packet to report the address of the last instruction before halting, followed by a support packet to indicate that tracing has stopped. Upon the deassertion of the halted signal, the encoder will start tracing again, commencing with a synchronization packet.
- Stop tracing when the CPU is in reset

The encoder will be reset separately, if TRACE_RESET_ENA is set. When the CPU is in reset, the encoder will output a packet to report the address of the last instruction before reset, followed by a support packet to indicate that tracing has stopped. Once the CPU exits from reset, the encoder will start tracing again, commencing with a synchronization packet.

- Using trigger outputs from the Debug Module

  The debug module of the RISC-V hart has a trigger unit. This defines a match control register (mcontrol) containing a 4-bit action field, and reserves codes 2 ~ 4 of this field for trace use. ESP32-P4 has implemented this feature. The values of the action field are listed in Table 3.8-1.

Table 3.8-1. Debug module trigger support (the action field of the mcontrol register)

Value	Description
2	Trace-on: start trace
3	Trace-off: end trace
4	Trace-notify: notify the encoder to report an address

For details about the debug module trigger unit, please refer to Chapter 2 *High-Performance CPU*.

3.8.2 Filter Configuration

The filter unit described in Section 3.5.4 can be configured as follows:

1. Select one of the following match modes:
   - Set TRACE_MATCH_PRIVILEGE to enable privilege match mode:
     - Configure TRACE_MATCH_CHOICE_PRIVILEGE to specify the privilege level.
   - Set TRACE_MATCH_ECAUSE to enable ecause match mode:
     - Configure TRACE_MATCH_CHOICE_ECAUSE to specify the ecause value
   - Set TRACE_MATCH_INTERRUPT to enable interrupt match mode:
     - Configure TRACE_MATCH_VALUE_INTERRUPT to specify interrupt level
   - Set TRACE_MATCH_COMP to enable comparator match mode:
     - Configure the primary comparator:
       * Configure TRACE_P_INPUT to choose the input
       * Configure TRACE_P_FUNCTION to select the comparator function
       * Configure TRACE_FILTER_P_COMPARATOR_MATCH_REG to define match value
     - Configure the secondary comparator if needed:
       * Configure TRACE_S_INPUT to choose the input
       * Configure TRACE_S_FUNCTION to select the comparator function
       * Configure TRACE_FILTER_S_COMPARATOR_MATCH_REG to define match value
     - Choose which comparator to match via TRACE_MATCH_MODE:
       * 0: only the primary comparator matches, and the secondary comparator has no effect

* 1: both the primary comparator and the secondary comparator match ($P\&\&S$)
* 2: neither the primary comparator nor secondary comparator match $!(P\&\&S)$
* 3: start filtering when primary matches up until secondary matches

If TRACE_P_NOTIFY or TRACE_S_NOTIFY is set, the comparator will report the matched address as a notification, the instructions will not be filtered. For example:

- If the TRACE_MATCH_MODE is 0, and TRACE_P_NOTIFY is set, then all instructions will be traced, and the matched primary comparator will be notified.
- If the TRACE_MATCH_MODE is 0, and TRACE_S_NOTIFY is set, the instruction will be filtered by the primary comparator, and the instruction that matches the secondary comparator will be notified.

2. Set TRACE_FILTER_EN to enable filter unit.

3.8.3 Enable Encoder

- Configure the address space for the trace memory via TRACE_MEM_START_ADDR_REG and TRACE_MEM_END_ADDR_REG. The encoder can access L2 MEM and PSRAM:
  - L2MEM: 0x4FF00000 ~ 0x4FFBFFFF
  - PSRAM: 0x48000000 ~ 0x4BFFFFFF
- Update the value of TRACE_MEM_CURRENT_ADDR_REG to the value of TRACE_MEM_START_ADDR_REG by setting TRACE_MEM_CURRENT_ADDR_UPDATE
- (Optional) Configure the memory writing mode via the TRACE_MEM_LOOP bit of TRACE_TRIGGER_REG
  - 0: Non-loop mode
  - 1: Loop mode (default)
- Configure the synchronization mode via the TRACE_RESYNC_MODE bit of TRACE_RESYNC_PROLONGED_REG
  - 0: Disable the synchronization counter
  - 1: Invalid. No effect
  - 2: Synchronization counter counts by packet
  - 3: Synchronization counter counts by cycle
- (Optional) Configures the threshold for the synchronization counter (default value is 128) via TRACE_RESYNC_PROLONGED_REG
- (Optional) Enable Interrupt
  - Set the corresponding bit of TRACE_INTR_ENA_REG to enable the corresponding interrupt
  - Set the corresponding bit of TRACE_INTR_CLR_REG to clear the corresponding interrupt
  - Read TRACE_INTR_RAW_REG to know which interrupt occurs
- (Optional) Enable automatic restart by setting the TRACE_RESTART_ENA bit of TRACE_TRIGGER_REG. This function is enabled by default

- There are 2 ways to enable the trace encoder:
  - Setting the TRACE_TRIGGER_ON bit of TRACE_TRIGGER_REG
  - Through the Debug module Trace-on to start the encoder. When the trigger is matched and the action is 2, it will start the encoder. For trigger configurations, please refer to the RISC-V Debug Specification.

Once the encoder is enabled, it will keep tracing the HP CPU's instruction trace interface and writing packets to the trace memory.

3.8.4 Disable Encoder

There are two ways to end output trace packets:

1. Set TRACE_TRIGGER_OFF of TRACE_TRIGGER_REG to end the encoder
2. Through the Debug module Trace-off to end the encoder. When the trigger is matched and the action is 3, it will end the encoder. For trigger configurations, please refer to the RISC-V Debug Specification.

Due to the internal FIFO, the packets are not written to the memory immediately after the end of the trace. It is recommended to query the TRACE_FIFO_STATUS_REG to confirm that the data is all written to memory.

Table 3.8-2. Trace Status

Field	Description
TRACE_FIFO_EMPTY	If 1 indicates that the FIFO is empty, all data have been written to memory
TRACE_WORK_STATUS	Encoder's work status: • 0: idle state, the encoder is not started • 1: the encoder is normally working that output packets • 2: the encoder is not outputting packets, the CPU is halted or in reset • 3: the encoder is not outputting packets, it occurs while a packet is lost, and is waiting for FIFO empty to restart.

3.8.5 Notify

Users may want to report a special address, even if it is not the target of an uninferable discontinuity. There are two ways to notify and report a special address:

1. The debug trigger unit matched, and the action is 4
2. The filter comparator matched

When a notification is requested, the encoder will report a format 2 or format 1 with an address packet, determined by the branch_map. If branch_map is 0, report format 2, otherwise report format 1. The notify field shows whether the packet is reporting a notification address. If it is the target of an uninferable discontinuity, even if it is a notification, the notify field will not be asserted, otherwise the decoder cannot reconstruct the instruction stream correctly.

3.8.6 Decode Data Packets

- Find the first address to decode
  - Read the TRACE_MEM_FULL_INTR_RAW bit of the TRACE_INTR_RAW_REG register to know if the trace memory is full
    * if read 0, read the trace packets from TRACE_MEM_START_ADDR_REG
    * if read 1, and the loop mode is enabled, then the old trace packets are overwritten. In this case, read the TRACE_MEM_CURRENT_ADDR_REG to know the last writing address, and use this address as the first address to decode
- Use the decoder to decode data packets
  - The decoder reads all data packets starting from the first address, and reconstructs the data stream with the binary file
  - As mentioned in 3.6, the encoder writes 14 zero bytes to the memory partition boundary every time when 128 packets are transmitted. Given this fact, the first non-zero byte after 14 zero bytes should be the header of a new packet

3.8.7 AHB Configuration

ESP32-P4 encoder supports the AHB bus. There are some optional configurations to control the transmission bandwidth of the encoder.

If there are many DMA masters, users can increase the bandwidth of Trace by setting AHB burst TRACE_AHB_CONFIG_REG.

- The TRACE_HBURST supports the following configurations:
  - 0x0: SINGLE
  - 0x1: INCR (length not defined)
  - 0x2: INCR4
  - 0x4: INCR8
  - Others: reserved

  Burst transfer means that once arbitration is successful, multiple words can be transferred continuously. For SINGLE, the length is 1; for INCR, the length is undefined; for INCR4, the length is 4; for INCR8, the length is 8. The longer the burst length, the better the bandwidth.
- When configured as INCR transfer, since INCR is a burst of indeterminate length, in order to avoid Trace occupying the bus for a long time, it will end the INCR transfer after the transfer length up to TRACE_MAX_INCR, and restart bus arbitration.

This is the configuration used to adjust the bandwidth. When the bandwidth is sufficient, it's not required to do the configuration, and users can use the default configuration.

3.8.8 Software Retention

The TRACE_WORK_STATUS represents the state of the encoder. When the encoder is not in the IDLE state, it must backup and restore the configuration when the chip enters sleep.

3.9 Register Summary

The addresses in this section are relative to RISC-V Trace Encoder base address provided in Table 9.3-2 in Chapter 9 *System and Memory*.

The abbreviations given in Column **Access** are explained in Section *Access Types for Registers*.

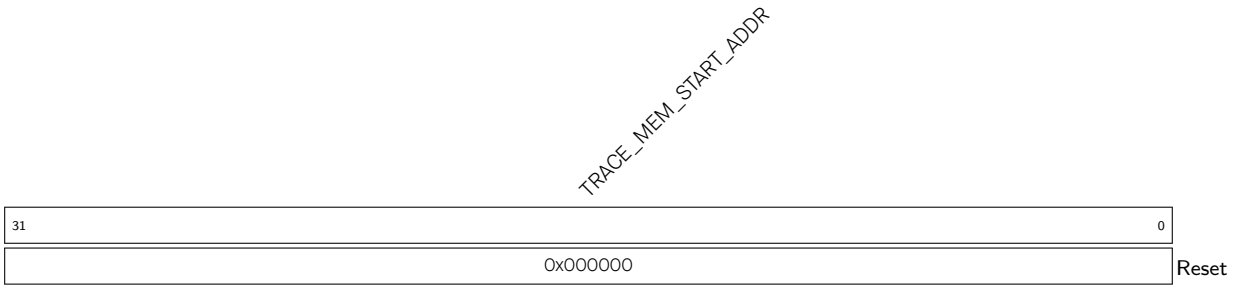
Name	Description	Address	Access
Memory configuration registers			
TRACE_MEM_START_ADDR_REG	Memory start address	0x0000	R/W
TRACE_MEM_END_ADDR_REG	Memory end address	0x0004	R/W
TRACE_MEM_CURRENT_ADDR_REG	Memory current address	0x0008	RO
TRACE_MEM_ADDR_UPDATE_REG	Memory address update	0x000C	WT
FIFO status register			
TRACE_FIFO_STATUS_REG	FIFO status register	0x0010	RO
Interrupt registers			
TRACE_INTR_ENA_REG	Interrupt enable register	0x0014	R/W
TRACE_INTR_RAW_REG	Interrupt raw status register	0x0018	RO
TRACE_INTR_CLR_REG	Interrupt clear register	0x001C	WT
Trace configuration registers			
TRACE_TRIGGER_REG	Trace trigger register	0x0020	varies
TRACE_CONFIG_REG	Trace configuration register	0x0024	R/W
TRACE_FILTER_CONTROL_REG	Filter control register	0x0028	R/W
TRACE_FILTER_MATCH_CONTROL_REG	Filter match control register	0x002C	R/W
TRACE_FILTER_COMPARATOR_CONTROL_REG	Filter comparator match control register	0x0030	R/W
TRACE_FILTER_P_COMPARATOR_MATCH_REG	Primary comparator match value register	0x0034	R/W
TRACE_FILTER_S_COMPARATOR_MATCH_REG	Secondary comparator match value register	0x0038	R/W
TRACE_RESYNC_PROLONGED_REG	Resynchronization configuration register	0x003C	R/W
TRACE_AHB_CONFIG_REG	AHB configuration register	0x0040	R/W
Clock gating control and configuration register			
TRACE_CLOCK_GATE_REG	Clock gating control register	0x0044	R/W
Version register			
TRACE_DATE_REG	Version control register	0x03FC	varies

3.10 Registers

The addresses in this section are relative to RISC-V Trace Encoder base address provided in Table 9.3-2 in Chapter 9 *System and Memory*.

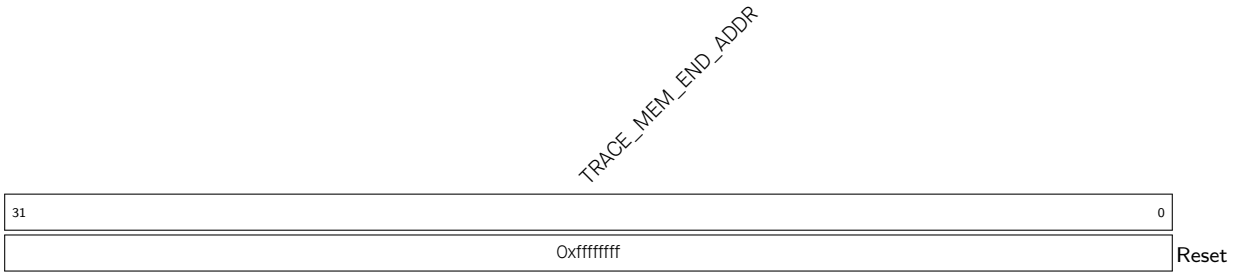
For how to program reserved fields, please refer to Section Programming Reserved Register Field.

Register 3.1. TRACE_MEM_START_ADDR_REG (0x0000)



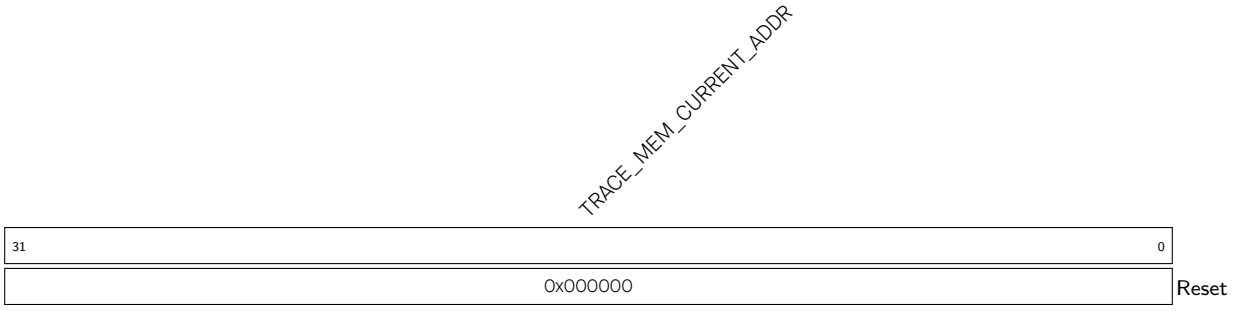
TRACE_MEM_START_ADDR Configures the start address of the trace memory. (R/W)

Register 3.2. TRACE_MEM_END_ADDR_REG (0x0004)



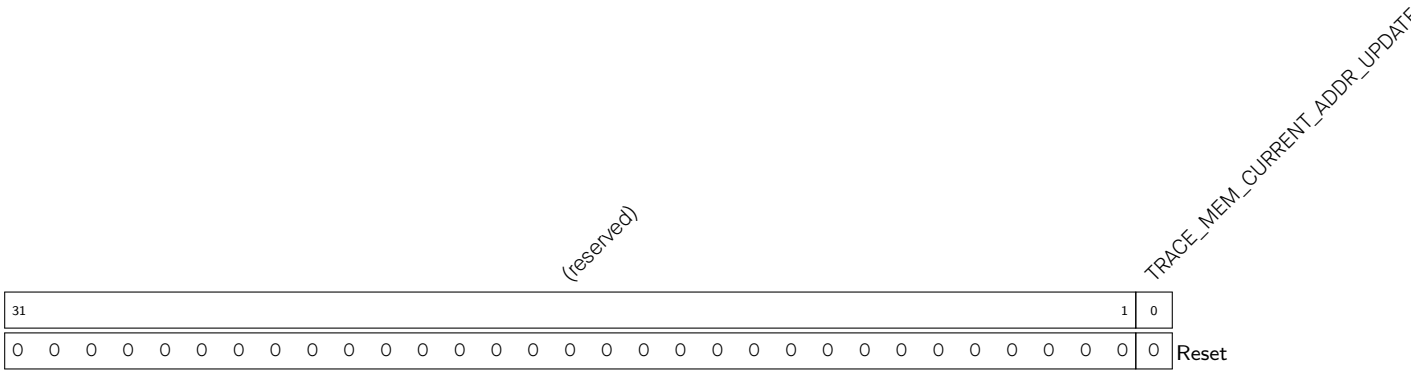
TRACE_MEM_END_ADDR Configures the end address of the trace memory. (R/W)

Register 3.3. TRACE_MEM_CURRENT_ADDR_REG (0x0008)



TRACE_MEM_CURRENT_ADDR Represents the current memory address for writing. (RO)

Register 3.4. TRACE_MEM_ADDR_UPDATE_REG (0x000C)



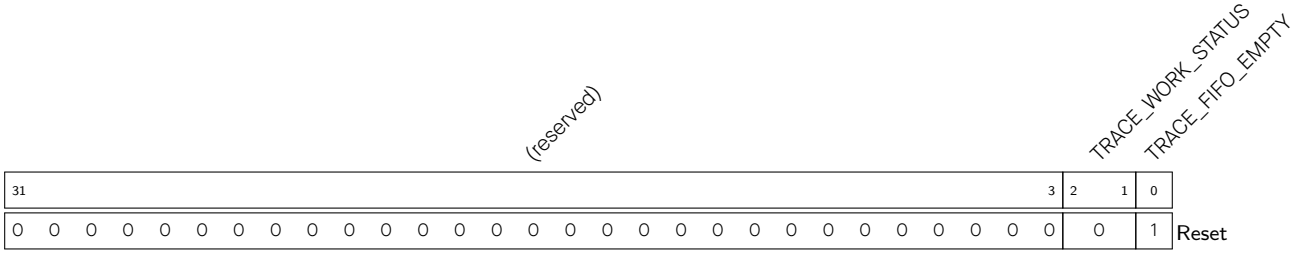
TRACE_MEM_CURRENT_ADDR_UPDATE Configures whether to update the value of TRACE_MEM_CURRENT_ADDR to TRACE_MEM_START_ADDR.

0: Not update

1: Update

(WT)

Register 3.5. TRACE_FIFO_STATUS_REG (0x0010)



TRACE_FIFO_EMPTY Represents whether the FIFO is empty.

1: Empty

0: Not empty (RO)

TRACE_WORK_STATUS Represents the state of the encoder:

0: Idle state

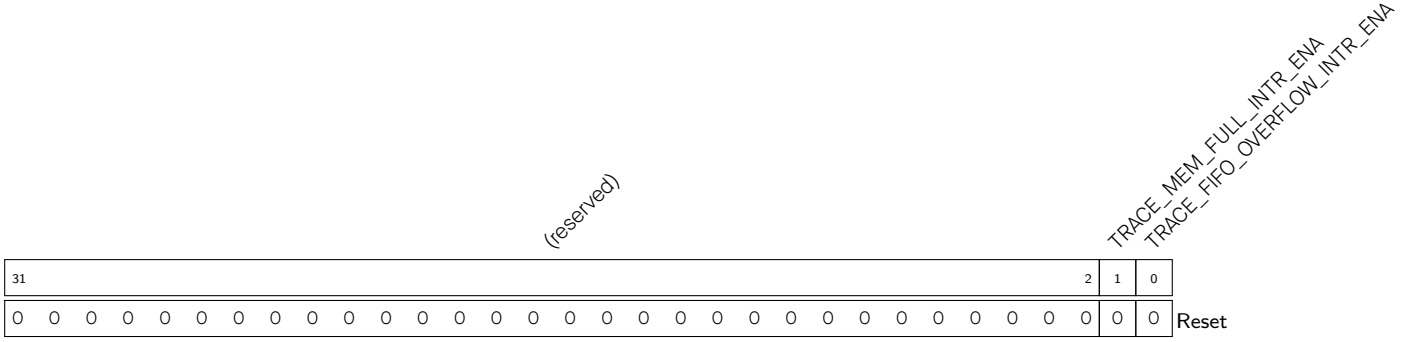
1: Working state

2: Wait state because the hart is halted or in reset

3: Lost state

(RO)

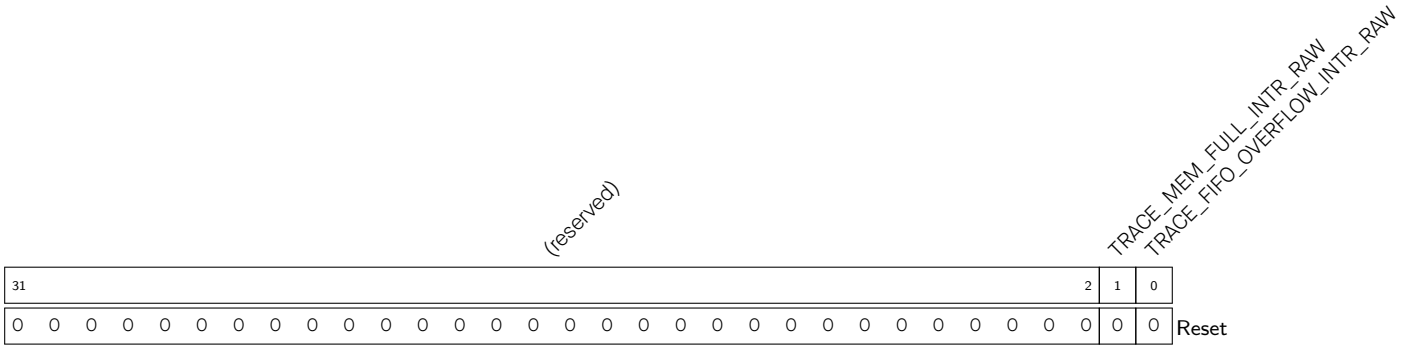
Register 3.6. TRACE_INTR_ENA_REG (0x0014)



TRACE_FIFO_OVERFLOW_INTR_ENA Write 1 to enable TRACE_FIFO_OVERFLOW_INTR. (R/W)

TRACE_MEM_FULL_INTR_ENA Write 1 to enable TRACE_MEM_FULL_INTR. (R/W)

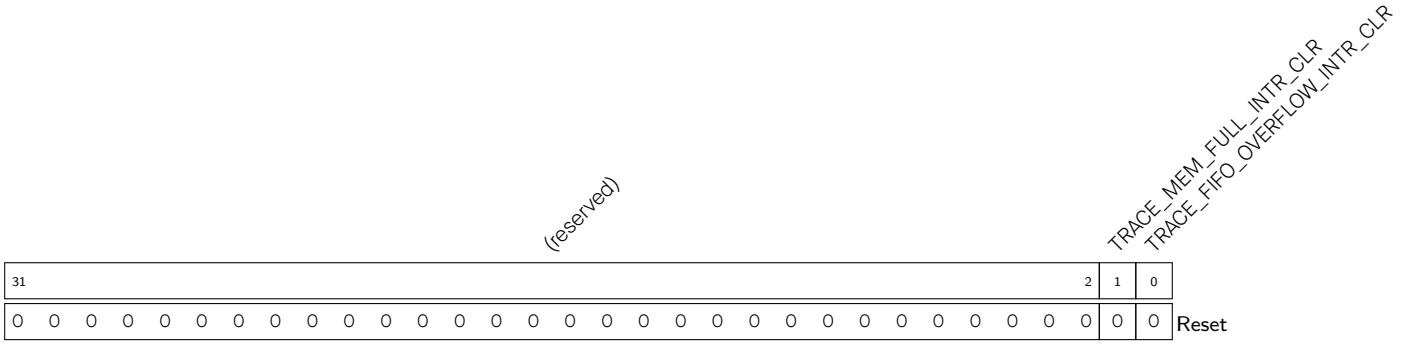
Register 3.7. TRACE_INTR_RAW_REG (0x0018)



TRACE_FIFO_OVERFLOW_INTR_RAW The raw interrupt status of TRACE_FIFO_OVERFLOW_INTR. (RO)

TRACE_MEM_FULL_INTR_RAW The raw interrupt status of TRACE_MEM_FULL_INTR. (RO)

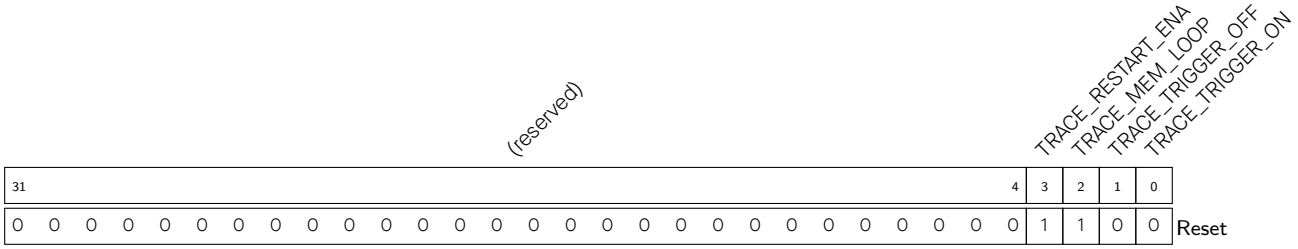
Register 3.8. TRACE_INTR_CLR_REG (0x001C)



TRACE_FIFO_OVERFLOW_INTR_CLR Write 1 to clear TRACE_FIFO_OVERFLOW_INTR. (WT)

TRACE_MEM_FULL_INTR_CLR Write 1 to clear TRACE_MEM_FULL_INTR. (WT)

Register 3.9. TRACE_TRIGGER_REG (0x0020)



TRACE_TRIGGER_ON Configures whether to enable the encoder.

0: Invalid

1: Enable

(WT)

TRACE_TRIGGER_OFF Configures whether to disable the encoder.

0: Invalid

1: Disable

(WT)

TRACE_MEM_LOOP Configures the memory writing mode.

0: Non-loop mode

1: Loop mode

(R/W)

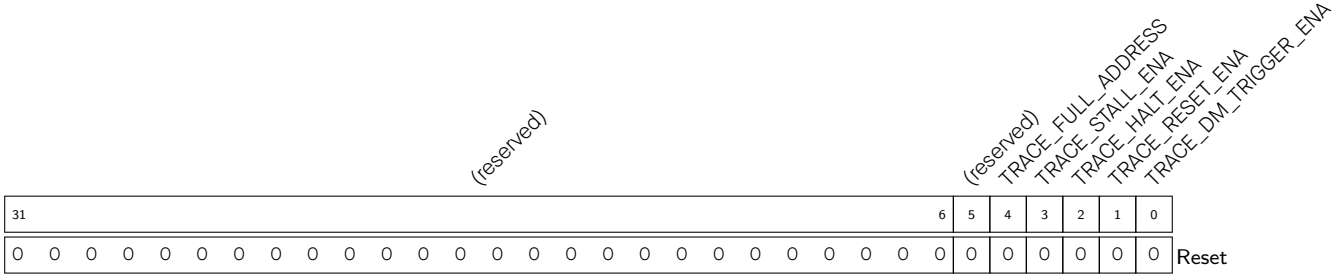
TRACE_RESTART_ENA Configures whether to enable automatic restart.

0: Disable

1: Enable

(R/W)

Register 3.10. TRACE_CONFIG_REG (0x0024)



TRACE_DM_TRIGGER_ENA Configure whether to enable the trigger signal.

0: Disable

1: Enable

(R/W)

TRACE_RESET_ENA Configures whether to enable the reset signal.

0: Disable

1: Enable

(R/W)

TRACE_HALT_ENA Configures whether to enable the halted signal.

0: Disable

1: Enable

(R/W)

TRACE_STALL_ENA Configures whether to enable the stall signal.

0: Disable

1: Enable

(R/W)

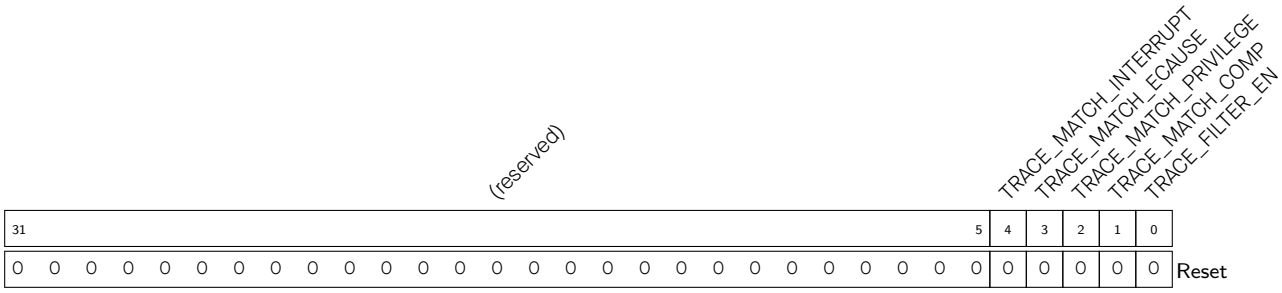
TRACE_FULL_ADDRESS Configure the address mode.

0: Delta address mode

1: Full address mode

(R/W)

Register 3.11. TRACE_FILTER_CONTROL_REG (0x0028)



TRACE_FILTER_EN Configure whether to enable filtering.

0: Disable

1: Enable

(R/W)

TRACE_MATCH_COMP Configures whether to enable the comparator match mode.

0: Disable

1: Enable

(R/W)

TRACE_MATCH_PRIVILEGE Configures whether to enable the privilege match mode.

0: Disable

1: Enable

(R/W)

TRACE_MATCH_ECAUSE Configures whether to enable the ecause match mode.

0: Disable

1: Enable

(R/W)

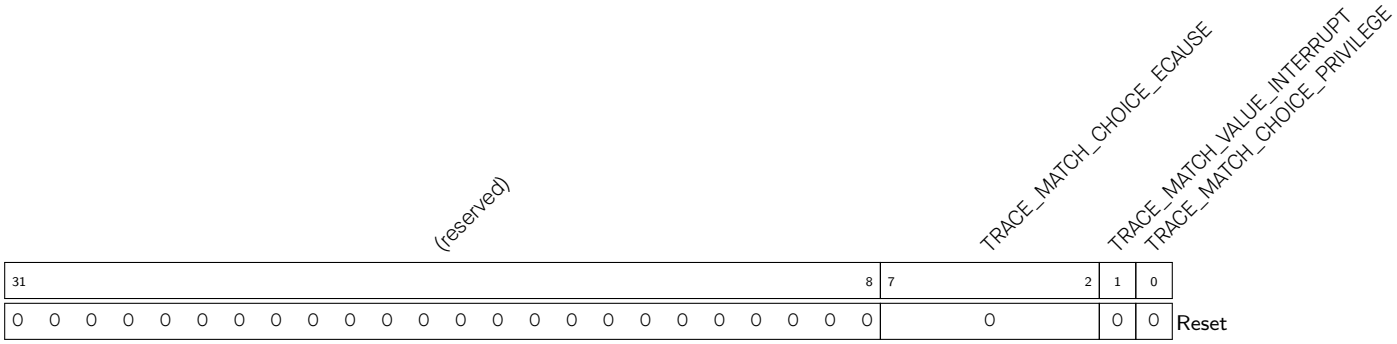
TRACE_MATCH_INTERRUPT Configures whether to enable the interrupt match mode.

0: Disable

1: Enable

(R/W)

Register 3.12. TRACE_FILTER_MATCH_CONTROL_REG (0x002C)



TRACE_MATCH_CHOICE_PRIVILEGE Configures the privilege level for matching.

0: User mode

1: Machine mode

(R/W)

TRACE_MATCH_VALUE_INTERRUPT Configures the interrupt level for matching. Valid only when TRACE_MATCH_INTERRUPT is set.

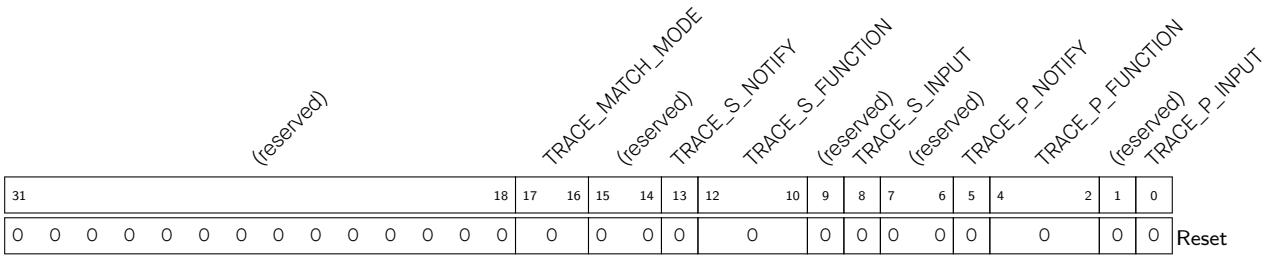
0: itype=1

1: itype=2

(R/W)

TRACE_MATCH_CHOICE_ECAUSE Configures the ecause code for matching. (R/W)

Register 3.13. TRACE_FILTER_COMPARATOR_CONTROL_REG (0x0030)


(reserved)
TRACE_MATCH_MODE
(reserved)
TRACE_S_NOTIFY
TRACE_S_FUNCTION
(reserved)
TRACE_S_INPUT
(reserved)
TRACE_P_NOTIFY
TRACE_P_FUNCTION
(reserved)
TRACE_P_INPUT
31 18 17 16 15 14 13 12 10 9 8 7 6 5 4 2 1 0
0 0 0

Register 3.13. TRACE_FILTER_COMPARATOR_CONTROL_REG (0x0030)

Continued from the previous page...

TRACE_S_NOTIFY Configures whether to explicitly report an instruction address matched against the secondary comparator.

0: Not report

1: Report

(R/W)

TRACE_MATCH_MODE Configures the comparator match condition.

0: Only the primary comparator matches

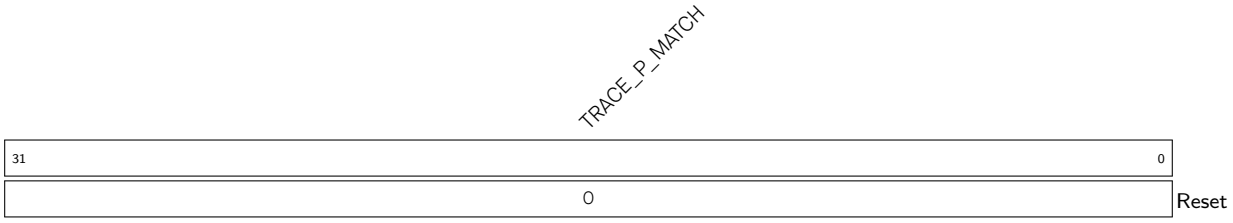
1: Both primary and secondary comparators match (P&&S)

2: Neither primary nor secondary comparator match !(P&&S)

3: Start filtering when the primary comparator matches and stop filtering when the secondary comparator matches

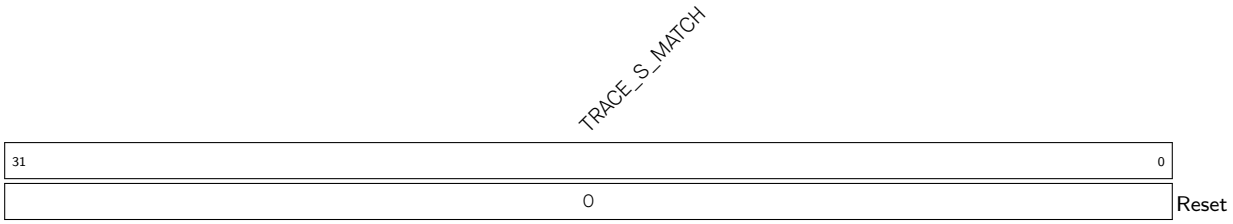
(R/W)

Register 3.14. TRACE_FILTER_P_COMPARATOR_MATCH_REG (0x0034)



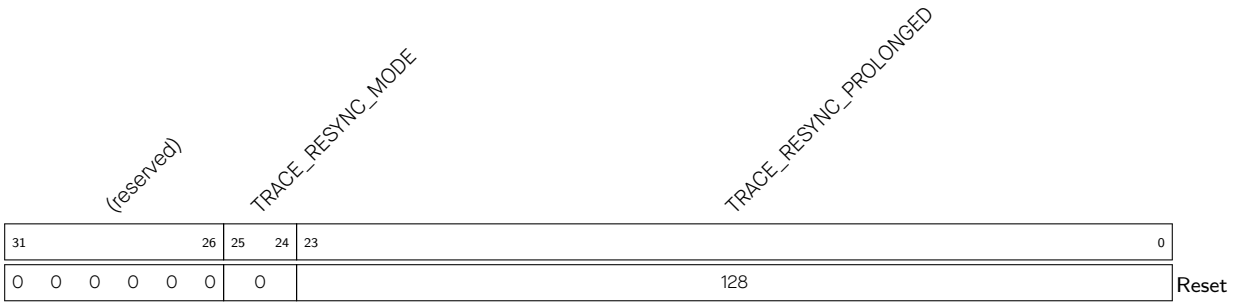
TRACE_P_MATCH Configures the match value for the primary comparator. (R/W)

Register 3.15. TRACE_FILTER_S_COMPARATOR_MATCH_REG (0x0038)



TRACE_S_MATCH Configures the match value for the secondary comparator. (R/W)

Register 3.16. TRACE_RESYNC_PROLONGED_REG (0x003C)



TRACE_RESYNC_PROLONGED Configures the threshold for the synchronization counter. (R/W)

TRACE_RESYNC_MODE Configures the synchronization mode.

0: Disable the synchronization counter

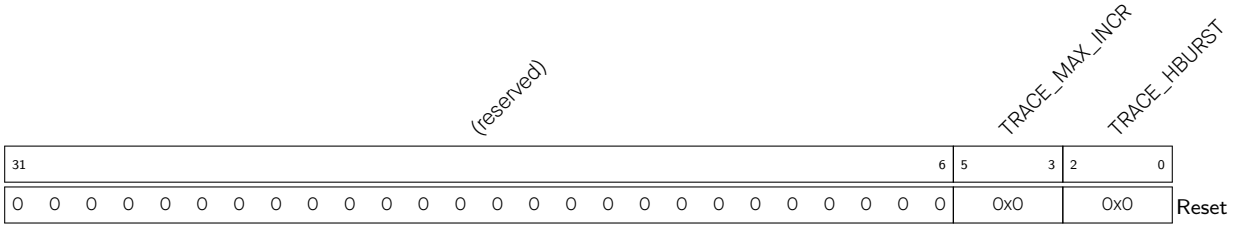
1: Invalid

2: Synchronization counter counts by packet

3: Synchronization counter counts by cycle

(R/W)

Register 3.17. TRACE_AHB_CONFIG_REG (0x0040)



TRACE_HBURST Configures the AHB burst mode.

0: SINGLE

1: INCR (length not defined)

2: INCR4

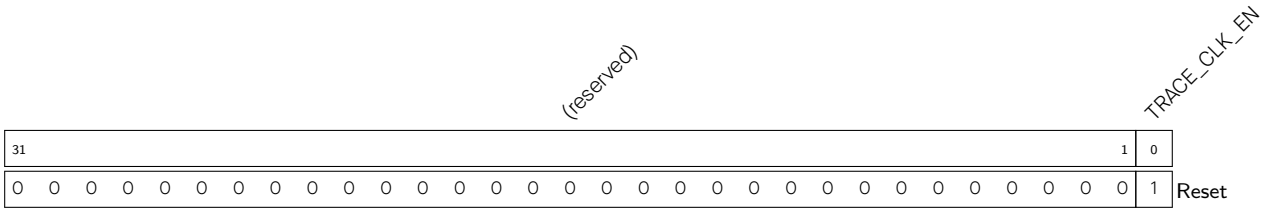
4: INCR8

Others: Invalid

(R/W)

TRACE_MAX_INCR Configures the maximum burst length for INCR mode. (R/W)

Register 3.18. TRACE_CLOCK_GATE_REG (0x0044)



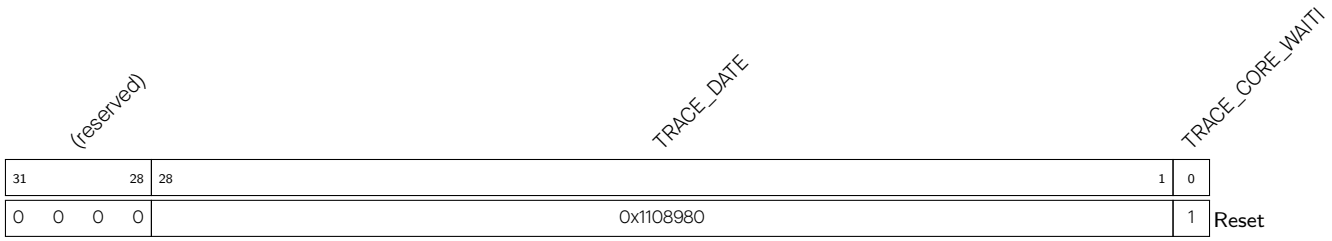
TRACE_CLK_EN Configures register clock gating. 0: Support clock only when the application writes registers to save power

1: Always force the clock on for registers

This bit doesn't affect register access

(R/W)

Register 3.19. TRACE_DATE_REG (0x03FC)



TRACE_DATE Version control register. (R/W)

TRACE_CORE_WAITI Represents whether the HP CPU cores are in the Wait-For-Interrupt (WFI) state.

0: Not in WFI

1: In WFI

(RO)

Chapter 4

Processor Instruction Extensions (PIE)

4.1 Overview

The ESP32-P4 HP 32-bit RISC-V dual-core processor supports the standard RV32IMAFCZcZb extensions. It also includes a custom extension named Xesploop (hardware loop), which executes "for" loops in hardware to improve performance. For more details, see the 2High-Performance CPU chapter. In addition, a custom AI and DSP extension, Xespv2p2, accelerates selected AI and DSP algorithms.

This chapter describes the Xespv2p2 extension, which is based on the SIMD (Single Instruction Multiple Data) architecture. It supports 8-bit, 16-bit, and 32-bit vector operations, which improves data throughput. Arithmetic instructions such as multiplication, shifting, and accumulation can also transfer data while performing computation, further improving execution efficiency.

4.2 Features

The Xespv2p2 extension has the following features:

- 128-bit general-purpose registers
- 128-bit vector operations, e.g., multiplication, addition, subtraction, accumulation, shifting, and comparison
- Integration of data transfer into arithmetic instructions
- Support for non-aligned 128-bit vector data
- Configurable rounding and saturation modes

4.3 Structure Overview

4.3.1 Internal Structure Diagram

This section helps you understand the available instructions, their capabilities, and their limits. It is not intended to describe hardware implementation details.

The figure below shows the internal structure of the Xespv2p2 extension for multiply-accumulate (MAC) instructions:

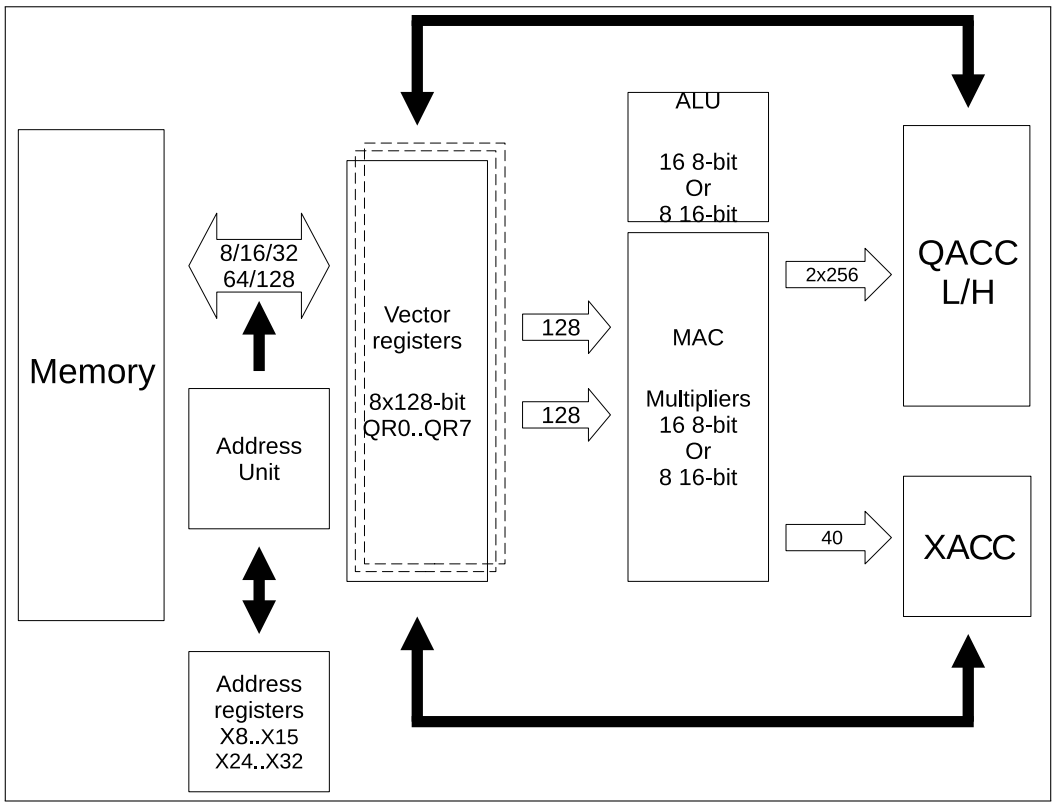


Figure 4.3-1. Xespv2p2 Internal Structure (MAC)

The diagram shows the data paths and Xespv2p2 components.

The Xespv2p2 unit contains:

- An address unit that reads 8/16/32/64/128-bit aligned data
- A bank of eight 128-bit vector QR registers
- An arithmetic logic unit (ALU) with
  - sixteen 8-bit multipliers
  - eight 16-bit multipliers
- QACC_H/QACC_L: two 256-bit accumulators
- XACC: a 40-bit accumulator

4.3.2 Vector Register Bank

Xespv2p2 provides eight 128-bit vector registers (QR). Each register can be viewed as an array of sixteen 8-bit elements, eight 16-bit elements, or four 32-bit elements. The element width depends on the instruction being executed.

4.3.3 Arithmetic Logic Unit (ALU)

The ALU can operate on 8-bit, 16-bit, or 32-bit data, matching the input width: an 8-bit ALU for 8-bit inputs, a 16-bit ALU for 16-bit inputs, and a 32-bit ALU for 32-bit inputs. It performs arithmetic operations such as addition and subtraction, and bitwise operations such as AND, OR, and shift.

ALU operands come from QR registers, and results can be written back to QR registers.

4.3.4 Multiply-and-Accumulate (MAC) Unit

The multiply-accumulate (MAC) unit performs MAC operations. It includes eight 16-bit multipliers and sixteen 8-bit multipliers. MAC results are stored in the special accumulator registers XACC and QACC.

4.3.4.1 QACC Accumulator Register

The QACC accumulator stores sixteen 8-bit MAC results or eight 16-bit MAC results. It consists of two parts: 256-bit QACC_H and 256-bit QACC_L. For 8-bit MAC operations, each 256-bit QACC register is treated as eight 32-bit elements. For 16-bit MAC operations, each is treated as four 64-bit elements.

To write an accumulator result back to a QR register, a 32-bit result can be reduced to 8 bits by right-shifting. For 16-bit MAC operations, a 64-bit result can be reduced to 16 bits by right-shifting.

Data can be loaded from memory into QACC, or QACC can be cleared to 0.

4.3.4.2 XACC Accumulator Register

XACC is a 40-bit accumulator that stores the sum of all multiplier outputs.
Data can be loaded from memory into XACC, or XACC can be cleared to 0.

4.3.5 Address Unit

Most Xespv2p2 instructions can load from or store to the 128-bit QR registers in parallel with computation. The address unit updates address registers in parallel, reducing the cost of explicit address updates.
Supported address updates include AR + signed immediate, ARx + ARy, and AR + 16.

4.4 Components of Extended Instruction Set

4.4.1 Registers

This section describes the registers used by the ESP32-P4 extended instruction set. See Table 4.4-1 for a register summary.

Table 4.4-1. Register List of ESP32-P4 Extended Instruction Set

Register	Quantity	Bit Width	Access	RW Instructions	Type
AR	16[1]	32	R/W	-	General-purpose register
QR	8	128	R/W	ESP.VLD.128.IP ESP.VST.128.IP	Customized vector register
SAR	1	6	R/W	ESP.MOVX.R.SAR ESP.MOVX.W.SAR	Special register
SAR_BYTE	1	4	R/W	ESP.MOVX.R.SAR.BYTES ESP.MOVX.W.SAR.BYTES	Customized special register
XACC	1	40	R/W	ESP.LD.XACC.IP ESP.ST.S.XACC.IP	Customized special register

Cont'd on next page

Table 4.4-1 – cont'd from previous page

Register	Quantity	Bit Width	Access	RW Instructions	Type
				ESP.ST.U.XACC.IP	
QACC_H	1	256	R/W	ESP.LD.QACC.H.L.128.IP	Customized special register
				ESP.LD.QACC.H.H.128.IP	
				ESP.ST.QACC.H.L.128.IP	
				ESP.ST.QACC.H.H.128.IP	
QACC_L	1	256	R/W	ESP.LD.QACC.L.L.128.IP	Customized special register
				ESP.LD.QACC.L.H.128.IP	
				ESP.ST.QACC.L.L.128.IP	
				ESP.ST.QACC.L.H.128.IP	
FFT_BIT_WIDTH	1	4	R/W	ESP.MOVX.R.FFT.BIT.WIDTH	Customized special register
				ESP.MOVX.W.FFT.BIT.WIDTH	
UA_STATE	1	128	R/W	ESP.LD.UA.STATE.IP	Customized special register
				ESP.ST.UA.STATE.IP	
CFG	1	32	R/W	ESP.MOVX.R.CFG	Customized special register
				ESP.MOVX.W.CFG	

4.4.1.1 General-Purpose Registers

4.4.1.1.1 AR

Although the HP core has 32 general-purpose registers, only 16 of them are available to Xespv2p2 extension instructions: x8–x15 and x24–x31 (s0–s1, a0–a5, s8–s11, t3–t6).

4.4.1.2 Vector Registers

4.4.1.2.1 QR

General-purpose registers are only 32 bits wide, while ESP32-P4 can access 128-bit data at a time, so they use only a quarter of the available data-bus bandwidth. For this reason, ESP32-P4 provides eight 128-bit vector registers (QR). QR registers mainly hold data transferred over the 128-bit bus for memory load and store, and temporary results of 128-bit vector operations.

During instruction execution, a QR register is treated as sixteen 8-bit, eight 16-bit, or four 32-bit operands, depending on the vector width defined by the instruction, so that one instruction can operate on multiple elements.

When a QR register is used as an instruction operand, you must specify its index explicitly. For example:

$$ESP.VADD.S8 \quad q2, q0, q1$$

This instruction uses QR registers q0 and q1 as inputs and stores the vector addition result in q2.

4.4.1.3 Special Registers

Unlike general-purpose registers, special registers are used implicitly by certain instructions. You neither need to nor can specify them as instruction operands. For example:

$$ESP.VMUL.S16 \quad q2, q0, q1$$

This vector multiplication uses q0 and q1 as inputs. Internally, each intermediate 32-bit product is right-shifted, and the lower 16 bits are retained to form the 128-bit result written to q2. The shift amount comes from the Shift Amount Register (SAR), which does not appear in the instruction operand list.

4.4.1.3.1 SAR

The Shift Amount Register (SAR) stores the shift amount in bits. Two classes of ESP32-P4 extension instructions use SAR: vector shift instructions, including ESP.VSR.S32, ESP.VSL.S32, and ESP.VSL.U32; and multiplication-related instructions, including ESP.VMUL.*, ESP.CMUL.*, ESP.FFT.AMS.*, and ESP.FFT.CMUL.*.

Vector shift instructions use the lower 5 bits of SAR as the right-shift or left-shift amount. Multiplication-related instructions use SAR to right-shift intermediate products, which affects the precision of the final result.

4.4.1.3.2 SAR_BYTE

SAR_BYTE stores the shift amount in bytes. It is used to handle misaligned 128-bit data (see Section 4.4.2). For vector arithmetic instructions, loads and stores are forced to 16-byte alignment, but application data addresses are not always 16-byte aligned.

While loading 128-bit data from memory, ESP.LD.128.USAR.IP and ESP.LD.128.USAR.XP write the lower 4 bits of the access address (the misalignment offset) to SAR_BYTE.

Two classes of ESP32-P4 extension instructions use SAR_BYTE. The first handles misaligned data in QR registers and includes ESP.SRCQ.* and ESP.SRC.Q*. These instructions load two 16-byte blocks from the aligned addresses before and after the misaligned address, concatenate them, and right-shift by the byte count in SAR_BYTE to obtain 128-bit data at the misaligned address. The second class handles misalignment during arithmetic and usually has the ".QUP" suffix.

4.4.1.3.3 FFT_BIT_WIDTH

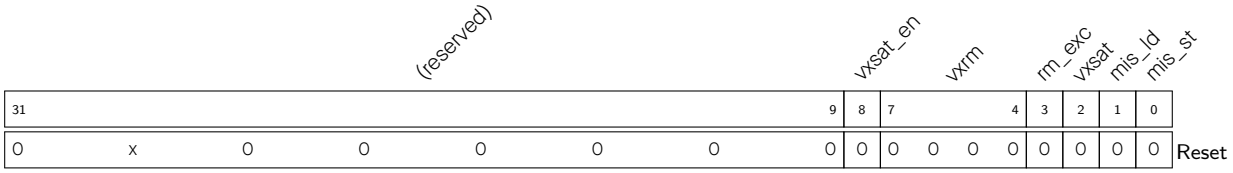
This special register is used only by ESP.FFT.BITREV. Its value selects the operating mode of ESP.FFT.BITREV: 0 through 7 correspond to 3-bit through 10-bit modes. For details, see ESP.FFT.BITREV.

4.4.1.3.4 UA_STATE

This special register is used only by ESP.FFT.AMS.S16.LD.INCP.UAUP. It holds misaligned 128-bit data previously read from memory. Each time the instruction runs, the contents of this register are concatenated with newly loaded misaligned data and then shifted to produce a 128-bit aligned result.

4.4.1.3.5 Core Configuration and Status Register

Register 4.1. CFG



vxsat_en Enables the saturation status flag.
(R/W)

vxrm Configures the rounding mode. See Table 4.4-3.
(R/W)

rm_exc Exception status for UNNECESSARY rounding (when both the instruction `rm` field and vxrm are DYNAMIC).
(RO)

vxsat Saturation status flag.
(RO)

mis_ld Enables hardware handling of misaligned loads.
(R/W)

mis_st Enables hardware handling of misaligned stores.
(R/W)

4.4.2 Data Format and Alignment

The extended instruction set supports 1-byte, 2-byte, 4-byte, 8-byte, and 16-byte data formats. QACC_H and QACC_L also use a 32-byte format, but there is no direct path between these special registers and memory. You can transfer QACC_H and QACC_L data through five 4-byte (AR) registers or two 16-byte (QR) registers.

Table 4.4-2 lists the bit length and alignment of common formats ('x' means the bit can be 0 or 1). In all formats, the HP core uses the byte as the smallest addressable unit in memory.

Table 4.4-2. Data Format and Alignment

Data Format	Length	Lower 4 Bits of Aligned Addresses in Memory
1-byte	8 bits	xxxx
2-byte	16 bits	xxx0
4-byte	32 bits	xx00
8-byte	64 bits	x000
16-byte	128 bits	0000

4.4.3 Misaligned Access Handling

Misaligned load and store for the ESP32-P4 HP core Xespv2p2 extension can be configured through mis_ld and mis_st:

- 0: Force-alignment mode
- 1: Hardware misaligned mode

4.4.3.1 Force-alignment mode

In this mode, every access address used by the extended instruction set is forced into alignment by clearing its lowest bits. For example, a 128-bit load from 0x3fc8_0024 forces the lowest 4 bits to 0, so the data at 0x3fc8_0020 is read. Likewise, the lowest 3 bits are cleared for 64-bit accesses, the lowest 2 bits for 32-bit accesses, and the lowest 1 bit for 16-bit accesses.

Therefore, access addresses must be correctly aligned; otherwise the returned data is not what you expect. In application code, declare the required alignment for variables or arrays. 16-byte alignment covers most use cases.

When the address of data used in an operation is not known to be aligned, the extension provides SAR_BYTE and related instructions such as ESP.LD.128.USAR.* and ESP.SRC.* to reconstruct misaligned 128-bit data.

Assume that register a8 holds the address of misaligned 128-bit data. The following sequence loads that data into QR register q2:

$ESP.LD.128.USAR.IP \quad q0, \quad a8, \quad 16$
$ESP.VLD.128.IP \quad q1, \quad a8, \quad 16$
$ESP.SRC.Q \quad q2, \quad q0, \quad q1$

4.4.3.2 Hardware misaligned mode

In this mode, hardware automatically handles misaligned accesses by splitting them into multiple aligned accesses. The results match the expected data.

4.4.4 Data Overflow and Saturation

Data overflow occurs when a result does not fit in the destination register. For example, in ESP.VMUL.S8, each product of two 8-bit values is 16 bits wide and remains 16 bits after a right shift, but the final storage width is 8 bits, so overflow is possible.

The ESP32-P4 extensions handle overflow in two ways: saturation and wraparound (retaining only the lower bits). Saturation clamps the result to the minimum or maximum value representable in the destination. Instructions that use saturation state this explicitly—for example, ESP.VADDS.* saturates addition results. Other internal overflow cases use wraparound: only the lower bits that fit the destination width are kept.

The 1-bit status flag vxsat indicates whether saturation occurred in the last operation (only when vxsat_en is set). Reading the CFG register clears this bit.

4.4.5 Data Rounding

Selected instructions include an `rm` field that selects the rounding mode for right shifts. Table 4.4-3 lists the eight modes and their encodings.

When `rm` is DYNAMIC (7), the instruction uses the 4-bit vxrm field in the CFG register. If both `rm` and vxrm are DYNAMIC, the operation is treated as UNNECESSARY: it behaves like FLOOR, and rm_exc is set when the right-shift result is not exact.

Table 4.4-3. Rounding Mode Configuration

Mode	Value of `rm`/vxrm	Description
FLOOR	0	Round toward negative infinity
CEILING	1	Round toward positive infinity
UP	2	Round away from zero
DOWN	3	Round toward zero
HALF_UP	4	Round to nearest; if equidistant, round up
HALF_DOWN	5	Round to nearest; if equidistant, round down
HALF_EVEN	6	Round to nearest; if equidistant, round to the even neighbor
DYNAMIC	7	Use the mode in vxrm. If vxrm is also DYNAMIC, use UNNECESSARY (FLOOR behavior with rm_exc reporting).

The following table shows example results for each rounding mode.

Table 4.4-4. Rounding Operation Results

Input number	FLOOR	CEILING	UP	DOWN	HALF_UP	HALF_DOWN	HALF_EVEN	UNNECESSARY
5.5	5	6	6	5	6	5	6	Exception
2.5	2	3	3	2	3	2	2	Exception
1.6	1	2	2	1	2	2	2	Exception
1.1	1	2	2	1	1	1	1	Exception
1.0	1	1	1	1	1	1	1	1
-1.0	-1	-1	-1	-1	-1	-1	-1	-1
-1.1	-2	-1	-2	-1	-1	-1	-1	Exception
-1.6	-2	-1	-2	-1	-2	-2	-2	Exception
-2.5	-3	-2	-3	-2	-3	-2	-2	Exception
-5.5	-6	-5	-6	-5	-6	-5	-6	Exception

4.5 Extended Instruction List

Table 4.5-1 lists the instruction categories in the extended instruction set. This section briefly introduces each category.

Table 4.5-1. Extended Instruction List

Instruction Type	Instruction[1]	Reference Section
Read instructions	ESP.VLD.128.[XP/IP]	4.5.1
	ESP.VLD.[H/L].64.[XP/IP]	
	ESP.VLDBC.[8/16/32].[XP/IP]	
	ESP.VLDHBC.16.INCP	
	ESP.LD.128.USAR.[XP/IP]	
	ESP.LDQA.[U/S][8/16].128.[XP/IP]	
	ESP.LD.QACC.[H/L].[H/L].128.IP	
	ESP.LD.XACC.IP	
	ESP.LD.UA.STATE.IP	
	ESP.LDXQ.32	
	ESP.VLDEXT.[U/S].[8/16].[IP/XP]	
Write instructions	ESP.VST.128.[XP/IP]	4.5.2
	ESP.VST.[H/L].64.[XP/IP]	
	ESP.ST.QACC.[H/L].[H/L].128.IP	
	ESP.ST.[U/S].XACC.IP	
	ESP.ST.UA.STATE.IP	
	ESP.STXQ.32	
Data exchange instructions	ESP.MOVI.[8/16/32].A	4.5.3
	ESP.MOVI.[8/16/32].Q	
	ESP.MOVX.[R/W].CFG	
	ESP.MOVX.[R/W].SAR.BYTES	
	ESP.MOVX.[R/W].SAR	
	ESP.MOVX.[R/W].FFT.BIT.WIDTH	
	ESP.MOVX.[R/W].PERF	
	ESP.MOVX.[R/W].XACC.[L/H]	
	ESP.VZIP.[8/16/32]	
	ESP.VUNZIP.[8/16/32]	
	ESP.VEXT.[U/S].[8/16]	
	ESP.ZERO.Q	
	ESP.ZERO.QACC	
	ESP.ZERO.XACC	
	ESP.MOV.S8.QACC	
	ESP.MOV.S16.QACC	
	ESP.MOV.U8.QACC	
	ESP.MOV.U16.QACC	

Con't on next page

Table4.5-1 – con't from previous page

Instruction Type	Instruction[1]	Reference Section
Arithmetic instructions	ESP.VADD.S[8/16/32].[-/LD.INCP/ST.INCP]	4.5.4
	ESP.VSADDS.S[8/16]	
	ESP.VSADDS.U[8/16]	
	ESP.VSUB.S[8/16/32].[-/LD.INCP/ST.INCP]	
	ESP.VSSUBS.S[8/16]	
	ESP.VSSUBS.U[8/16]	
	ESP.VMUL.[U/S][8/16].[-/LD.INCP/ST.INCP]	
	ESP.VMULS32.S16xS16	
	ESP.CMUL.S8.[-/LD.INCP/ST.INCP]	
	ESP.CMUL.S16.[-/LD.INCP/ST.INCP]	
	ESP.VMULAS.[U/S][8/16].XACC.[-/LD.IP/LD.XP]	
	ESP.VMULAS.S[8/16].QACC_[H/L]	
	ESP.VMULAS.[U/S][8/16].QACC.[-/LD.IP/LD.XP/LDBC.INCP]	
	ESP.VSMULAS.S[8/16].QACC.[-/LD.INCP]	
	ESP.SRCMB.[U/S][8/16].QACC	
	ESP.SRCMB.[U/S][8/16].Q.QACC	
	ESP.SRS.[S/U].XACC	
	ESP.VRELU.S[8/16]	
	ESP.VPRELU.S[8/16]	
	ESP.SAT	
	ESP.ADDX2	
	ESP.ADDX4	
	ESP.SUBX2	
	ESP.SUBX4	
	ESP.VABS.[8/16/32]	
Comparison instructions	ESP.VMAX.S[8/16/32].[-/LD.INCP/ST.INCP]	4.5.5
	ESP.MAX.S[16/32].A	
	ESP.VMIN.S[8/16/32].[-/LD.INCP/ST.INCP]	
	ESP.MIN.S[16/32].A	
	ESP.VCMP.[EQ/LT/GT].S[8/16/32]	
	ESP.VCLAMP.S16	
	ESP.VSAT.[U/S][8/16/32]	
Bitwise logic instructions	ESP.ORQ	4.5.6
	ESP.XORQ	
	ESP.ANDQ	
	ESP.NOTQ	

Con't on next page

Table4.5-1 – con't from previous page

Instruction Type	Instruction[1]	Reference Section
Shift instructions	ESP.SRC.Q	4.5.7
	ESP.SRC.Q.LD.[XP/IP]	
	ESP.SLCI.2Q	
	ESP.SLCXXP.2Q	
	ESP.SRCI.2Q	
	ESP.SRCXXP.2Q	
	ESP.SRCQ.128.ST.INCP	
	ESP.VSRD.[8/16/32]	
	ESP.VSLD.[8/16/32]	
	ESP.VSR.U32	
	ESP.VSR.S32	
	ESP.VSL.S32	
	ESP.VSL.U32	
FFT dedicated instructions	ESP.FFT.R2BF.S16.[-/ST.INCP]	4.5.8
	ESP.FFT.CMUL.S16.[LD.XP/ST.XP]	
	ESP.FFT.BITREV	
	ESP.FFT.AMS.S16.[LD.INCP.UAUP/LD.INCP/LD.R32.DECP/ST.INCP]	
	ESP.FFT.VST.R32.DECP	

[1] For more information about these instructions, please refer to Section 4.7.

4.5.1 Read Instructions

Read instructions use an AR register that holds the access address to issue a virtual-address memory access. Most read instructions load from memory first, then update the address. ESP.LDXQ.32 is an exception: it first selects a 16-bit element from a QR register with an immediate, adds that value to the address, and then performs the load.

The following table summarizes the memory behavior of read instructions. For details, see Section 4.7.

Table 4.5-2. Read Instructions

Instructions	Description
ESP.VLD.128.XP	Read the 16-byte data, then add the value stored in the AR register to the access address.
ESP.VLD.128.IP	Read the 16-byte data, then add the immediate value to the access address.
ESP.VLD.[H/L].64.XP	Read the 8-byte data, then add the value stored in the AR register to the access address.
ESP.VLD.[H/L].64.IP	Read the 8-byte data, then add the immediate value to the access address.
ESP.VLDBC.[8/16/32].XP	Read the 1-byte/2-byte/4-byte data, then add the value stored in the AR register to the access address.
ESP.VLDBC.[8/16/32].IP	Read the 1-byte/2-byte/4-byte data, then add the immediate value to the access address.
ESP.VLDHBC.16.INCP	Read the 16-byte data, then add 16 to the access address.
ESP.LD.128.USAR.XP	Read the 16-byte data, then add the value stored in the AR register to the access address.
ESP.LD.128.USAR.IP	Read the 16-byte data, then add the immediate value to the access address.
ESP.LDQA.U8.128.[XP/IP]	Load 16-byte data, slice by 1 byte, zero-extend each slice to 32 bits, write to QACC_H and QACC_L, then add an AR value or immediate to the access address.
ESP.LDQA.U16.128.[XP/IP]	Load 16-byte data, slice by 2 bytes, zero-extend each slice to 64 bits, write to QACC_H and QACC_L, then add an AR value or immediate to the access address.
ESP.LDQA.S8.128.[XP/IP]	Load 16-byte data, slice by 1 byte, sign-extend each slice to 32 bits, write to QACC_H and QACC_L, then add an AR value or immediate to the access address.
ESP.LDQA.S16.128.[XP/IP]	Load 16-byte data, slice by 2 bytes, sign-extend each slice to 64 bits, write to QACC_H and QACC_L, then add an AR value or immediate to the access address.
ESP.LD.QACC.[H/L].H.128.IP	Read the 16-byte data, then add the immediate value to the access address.
ESP.LD.QACC.[H/L].L.128.IP	Read the 16-byte data, then add the immediate value to the access address.
ESP.LD.XACC.IP	Read the 8-byte data, then add the immediate value to the access address.
ESP.LD.UA.STATE.IP	Read the 16-byte data, then add the immediate value to the access address.
ESP.LDXQ.32	Update access address first, then read the 4-byte data.
ESP.VLDEXT.[U/S][8/16].[IP/XP]	Vector zero-/sign-extend the loaded 8-bit/16-bit segments to 16-bit/32-bit segments.

4.5.2 Write Instructions

Write instructions use an AR register that holds the access address to issue a virtual-address memory access. Most write instructions store to memory first, then update the address. ESP.STXQ.32 is an exception: it first selects a 16-bit element from a QR register with an immediate, adds that value to the address, and then performs the store.

The following table summarizes the memory behavior of write instructions. For details, see Section 4.7.

Table 4.5-3. Write Instructions

Instructions	Description
ESP.VST.128.XP	Write the 16-byte data to memory, then add the value stored in the AR register to the access address.
ESP.VST.128.IP	Write the 16-byte data to memory, then add the immediate value to the access address.
ESP.VST.[H/L].64.XP	Write the 8-byte data to memory, then add the value stored in the AR register to the access address.
ESP.VST.[H/L].64.IP	Write the 8-byte data to memory, then add the immediate value to the access address.
ESP.ST.QACC.[H/L].H.128.IP	Write the 16-byte data to memory, then add the immediate value to the access address.
ESP.ST.QACC.[H/L].L.128.IP	Write the 16-byte data to memory, then add the immediate value to the access address.
ESP.ST.[U/S].XACC.IP	Zero-extend or signed-extend the value in the XACC register to 8-byte data and write it to memory, then add the immediate value to the access address.
ESP.ST.UA.STATE.IP	Write the 16-byte data to memory, then add the immediate value to the access address.
ESP.STXQ.32	Update the access address first, then write the 4-byte data to memory.

4.5.3 Data Exchange Instructions

Data exchange instructions transfer data between registers. Because register widths differ, immediate values select elements or fields, and both zero-extension and sign-extension instructions are provided.

For details, see Section 4.7.

Table 4.5-4. Data Exchange Instructions

Instructions	Description
ESP.MOVI.8.A	Assign a piece of 8-bit data from the QR register to the AR register.
ESP.MOVI.16.A	Assign a piece of 16-bit data from the QR register to the AR register.
ESP.MOVI.32.A	Assign a piece of 32-bit data from the QR register to the AR register.
ESP.MOVI.8.Q	Assign the data stored in the AR register to a piece of 8-bit data space in the QR register.
ESP.MOVI.16.Q	Assign the data stored in the AR register to a piece of 16-bit data space in the QR register.
ESP.MOVI.32.Q	Assign the data stored in the AR register to a piece of 32-bit data space in the QR register.
ESP.MOVX.R.CFG	Read CFG value to an AR register.
ESP.MOVX.W.CFG	Write CFG register.
ESP.MOVX.R.SAR.BYTES	Read SAR_BYTE value to an AR register.
ESP.MOVX.W.SAR.BYTES	Write SAR_BYTE register.
ESP.MOVX.R.SAR	Read SAR value to an AR register.

Instructions	Description
ESP.MOVX.W.SAR	Write SAR register.
ESP.MOVX.R.FFT.BIT.WIDTH	Read FFT_BIT_WIDTH value to an AR register.
ESP.MOVX.W.FFT.BIT.WIDTH	Write FFT_BIT_WIDTH register.
ESP.MOVX.R.PERF	Read performance counter value to an AR register.
ESP.MOVX.R.XACC.H	Read highest 8-bit value of XACC to an AR register.
ESP.MOVX.R.XACC.L	Read lowest 32-bit value of XACC to an AR register.
ESP.MOVX.W.XACC.H	Write highest 8-bit value of XACC.
ESP.MOVX.W.XACC.L	Write lowest 32-bit value of XACC.
ESP.VZIP.[8/16/32]	Encode the two QR registers in 1-byte/2-byte/4-byte unit.
ESP.VUNZIP.[8/16/32]	Decode the two QR registers in 1-byte/2-byte/4-byte unit.
ESP.VZIPT.[8/16]	Encode the 3 QR registers in 1-byte/2-byte unit.
ESP.VUNZIPT.[8/16]	Decode the 3 QR registers in 1-byte/2-byte unit.
ESP.VEXT.[U/S]8	Vector-extend the loaded 8-bit segments to 16-bit segments.
ESP.VEXT.[U/S]16	Vector-extend the loaded 16-bit segments to 32-bit segments.
ESP.ZERO.Q	Clear a specified QR register.
ESP.ZERO.QACC	Clear QACC_H and QACC_L registers.
ESP.ZERO.XACC	Clear a specified XACC register.
ESP.MOV.S8.QACC	Slice the QR register by 1-byte, and sign-extend it to 32-bit data, then assign this value to QACC_H and QACC_L registers.
ESP.MOV.S16.QACC	Slice the QR register by 2-byte, and sign-extend it to 64-bit data, then assign this value to QACC_H and QACC_L registers.
ESP.MOV.U8.QACC	Slice the QR register by 1-byte, and zero-extend it to 32-bit data, then assign this value to QACC_H and QACC_L registers.
ESP.MOV.U16.QACC	Slice the QR register by 2-byte, and zero-extend it to 64-bit data, then assign this value to QACC_H and QACC_L registers.

4.5.4 Arithmetic Instructions

Arithmetic instructions perform SIMD vector operations, including vector addition, multiplication, complex multiplication, multiply-accumulate, and vector–scalar multiply-accumulate.

4.5.4.1 Vector Addition Instructions

ESP32-P4 provides vector addition and subtraction instructions for 1-byte, 2-byte, and 4-byte elements.

Because vector operands are often stored in memory, some vector addition instructions combine the arithmetic with a 16-byte memory access and then add 16 to the address register so that it points to the next contiguous 16-byte block. Choose the instruction that matches your algorithm.

These instructions also saturate addition and subtraction results to preserve numeric range.

Table 4.5-5. Vector Addition Instructions

Instructions	Description
ESP.VADD.S[8/16/32]	Perform vector addition operation on 1-byte/2-byte/4-byte data.

Instructions	Description
ESP.VADD.S[8/16/32].LD.INCP	Perform vector addition operation on 1-byte/2-byte/4-byte data, and read 16-byte data from memory at the same time.
ESP.VADD.S[8/16/32].ST.INCP	Perform vector addition operation on 1-byte/2-byte/4-byte data, and write 16-byte data to memory at the same time.
ESP.VSUB.S[8/16/32]	Perform vector subtraction operation on 1-byte/2-byte/4-byte data.
ESP.VSUB.S[8/16/32].LD.INCP	Perform vector subtraction operation on 1-byte/2-byte/4-byte data, and read 16-byte data from memory at the same time.
ESP.VSUB.S[8/16/32].ST.INCP	Perform vector subtraction operation on 1-byte/2-byte/4-byte data, and write 16-byte data to memory at the same time.
ESP.VSSUBS.S[8/16/32]	Perform vector subtraction operation on 1-byte/2-byte/4-byte data.
ESP.VSSUBS.U[8/16/32]	Perform vector subtraction operation on 1-byte/2-byte/4-byte data.
ESP.VSADDS.S[8/16/32]	Perform vector addition operation on 1-byte/2-byte/4-byte data.
ESP.VSADDS.U[8/16/32]	Perform vector addition operation on 1-byte/2-byte/4-byte data.

4.5.4.2 Vector Multiplication Instructions

ESP32-P4 provides signed and unsigned vector multiplication for 1-byte and 2-byte elements.

Some vector multiplication instructions combine the multiply with a 16-byte memory access and then add 16 to the access address. Choose the instruction that matches your algorithm.

Table 4.5-6. Vector Multiplication Instructions

Instructions	Description
ESP.VMUL.U[8/16]	Perform vector multiplication operation on unsigned 1-byte/2-byte data.
ESP.VMUL.S[8/16]	Perform vector multiplication operation on signed 1-byte/2-byte data.
ESP.VMUL.U[8/16].LD.INCP	Perform vector multiplication operation on unsigned 1-byte/2-byte data, and read 16-byte data from memory at the same time.
ESP.VMUL.S[8/16].LD.INCP	Perform vector multiplication operation on signed 1-byte/2-byte data, and read 16-byte data from memory at the same time.
ESP.VMUL.U[8/16].ST.INCP	Perform vector multiplication operation on unsigned 1-byte/2-byte data, and write 16-byte data to memory at the same time.
ESP.VMUL.S[8/16].ST.INCP	Perform vector multiplication operation on signed 1-byte/2-byte data, and write 16-byte data to memory at the same time.
ESP.VMUL.S32.S16xS16	Perform 16bit vector multiplication operation and right shift the result.

4.5.4.3 Vector Complex Multiplication Instructions

ESP32-P4 provides vector complex multiplication for 1-byte and 2-byte signed data.

Some complex-multiplication instructions combine the multiply with a 16-byte memory access and then add 16 to the access address. Choose the instruction that matches your algorithm.

Table 4.5-7. Vector Complex Multiplication Instructions

Instructions	Description
ESP.CMUL.S16	Perform vector complex multiplication operation to 2-byte data.
ESP.CMUL.S16.LD.INCP	Perform vector complex multiplication operation to 2-byte data, and read 16-byte data from memory at the same time.
ESP.CMUL.S16.ST.INCP	Perform vector complex multiplication operation to 2-byte data, and write 16-byte data to memory at the same time.
ESP.CMUL.S8	Perform vector complex multiplication operation to 1-byte data.
ESP.CMUL.S8.LD.INCP	Perform vector complex multiplication operation to 1-byte data, and read 16-byte data from memory at the same time.
ESP.CMUL.S8.ST.INCP	Perform vector complex multiplication operation to 1-byte data, and write 16-byte data to memory at the same time.

4.5.4.4 Vector Multiplication Accumulation Instructions

ESP32-P4 provides two kinds of vector multiply-accumulate instructions: one accumulates into the 40-bit XACC register; the other accumulates into the corresponding segments of QACC_H and QACC_L. Both support 1-byte and 2-byte segments.

Some of these instructions also perform a 16-byte memory access in parallel and then update the address by an AR value or an immediate.

Table 4.5-8. Vector Multiplication Accumulation Instructions

Instructions	Description
ESP.VMULAS.[U/S][8/16].XACC	Perform vector multiplication accumulation to signed/unsigned data in 1-byte/2-byte segment, and store the result to the XACC register temporarily.
ESP.VMULAS.[U/S][8/16].XACC.LD.IP	Perform vector multiplication accumulation to signed/unsigned data in 1-byte/2-byte segment, and store the result to the XACC register temporarily, then read 16-byte data from memory. Add immediate to address register.
ESP.VMULAS.[U/S][8/16].XACC.LD.XP	Perform vector multiplication accumulation to signed/unsigned data in 1-byte/2-byte segment, and store the result to the XACC register temporarily, then read 16-byte data from memory. Add the value of AR register to address register.
ESP.VCMULAS.S[8/16].QACC.[H/L]	Perform vector multiplication operation on signed data in 1-byte/2-byte segment, and accumulate the results to corresponding bit segments in QACC_H or QACC_L registers.
ESP.VCMULAS.S[8/16].QACC.[H/L].LD.IP	Perform vector multiplication operation on signed data in 1-byte/2-byte segment, and accumulate the results to corresponding bit segments in QACC_H or QACC_L registers.
ESP.VCMULAS.S[8/16].QACC.[H/L].LD.XP	Perform vector multiplication operation on signed data in 1-byte/2-byte segment, and accumulate the results to corresponding bit segments in QACC_H or QACC_L registers.

Instructions	Description
ESP.VMULAS.[U/S][8/16].QACC	Perform vector multiplication operation on signed/unsigned data in 1-byte/2-byte segment, and accumulate the results to corresponding bit segments in QACC_H and QACC_L registers.
ESP.VMULAS.[U/S][8/16].QACC.LD.IP	Perform vector multiplication operation on signed/unsigned data in 1-byte/2-byte segment, and accumulate the results to corresponding bit segments in QACC_H and QACC_L registers, then read 16-byte data from memory. Add immediate to address register.
ESP.VMULAS.[U/S][8/16].QACC.LD.XP	Perform vector multiplication operation on signed/unsigned data in 1-byte/2-byte segment, and accumulate the results to corresponding bit segments in QACC_H and QACC_L registers, then read 16-byte data from memory. Add the value of AR register to address register.
ESP.VMULAS.[U/S][8/16].QACC.LDBC.INCP	Perform vector multiplication operation on signed/unsigned data in 1-byte/2-byte segment, and accumulate the results to corresponding bit segments in QACC_H and QACC_L registers. Then read 1-byte/2-byte data from memory and broadcast it to the 128-bit QR register. Add 16 to address register.

4.5.4.5 Vector and Scalar Multiplication Accumulation Instructions

These instructions are similar to QACC-based vector multiply-accumulate, except that one operand is a vector and the other is a scalar. Variants that access memory during the operation are also available.

Table 4.5-9. Vector and Scalar Multiplication Accumulation Instructions

Instructions	Description
ESP.VSMULAS.S[8/16].QACC	Perform vector and scalar multiplication accumulation on signed data in 1-byte/2-byte segment.
ESP.VSMULAS.S[8/16].QACC.LD.INCP	Perform vector and scalar multiplication accumulation on signed data in 1-byte/2-byte segment, and read 16-byte data from memory at the same time. Add 16 to address register.

4.5.4.6 Other Instructions

This group includes arithmetic right shifts of multiply-accumulate results in QACC_H, QACC_L, and XACC. Adjust the shift amount to obtain the desired precision. It also includes conditionally enabled vector multiplications.

Table 4.5-10. Other Instructions

Instructions	Description
ESP.SRCMB.S[8/16].QACC	Perform signed right-shifting on data in QACC_H and QACC_L registers in segment unit.
ESP.SRCMB.U[8/16].QACC	Perform unsigned right-shifting on data in QACC_H and QACC_L registers in segment unit.
ESP.SRCMB.S[8/16].Q.QACC	Perform signed right-shifting on data in QACC_H and QACC_L registers in segment unit.
ESP.SRCMB.U[8/16].Q.QACC	Perform unsigned right-shifting on data in QACC_H and QACC_L registers in segment unit.
ESP.SRS.S.XACC	Perform signed right-shifting on data in the XACC register.
ESP.SRS.U.XACC	Perform unsigned right-shifting on data in the XACC register.
ESP.VRELU.S[8/16]	Perform vector and scalar multiplication based on enabling conditions.
ESP.VPRELU.S[8/16]	Perform vector-to-vector multiplication based on enabling conditions.
ESP.SAT	Saturate 32-bit general register between rs1 and rs2.
ESP.ADDX2	Perform addition between rs1 and rs2.
ESP.ADDX4	Perform addition between rs1 and rs2.
ESP.SUBX2	Perform subtraction between rs1 and rs2.
ESP.SUBX4	Perform subtraction between rs1 and rs2.
ESP.VABS.[8/16/32]	Take the absolute value of the signed data.

4.5.5 Comparison Instructions

Vector comparison instructions operate on 1-byte, 2-byte, or 4-byte elements. They include finding the larger or smaller of two values, and generating an all-ones or all-zeros mask for equal-to, greater-than, and less-than comparisons.

Some comparison instructions also perform a 16-byte memory access and then add 16 to the address register. Choose the instruction that matches your algorithm.

Table 4.5-11. Comparison Instructions

Instructions	Description
ESP.VMAX.S[8/16/32]	Take the larger value between the two 1-byte/2-byte/4-byte values.
ESP.VMAX.S[8/16/32].LD.INCP	Take the larger value between the two 1-byte/2-byte/4-byte values, and read 16-byte data from memory at the same time.
ESP.VMAX.S[8/16/32].ST.INCP	Take the larger value between the two 1-byte/2-byte/4-byte values, and write 16-byte data to memory at the same time.
ESP.VMAX.S[16/32].A	Find the maximum signed value in all elements of a QR register.
ESP.VMIN.S[8/16/32]	Take the smaller value between the two 1-byte/2-byte/4-byte values.
ESP.VMIN.S[8/16/32].LD.INCP	Take the smaller value between the two 1-byte/2-byte/4-byte values, and read 16-byte data from memory at the same time.
ESP.VMIN.S[8/16/32].ST.INCP	Take the smaller value between the two 1-byte/2-byte/4-byte values, and write 16-byte data to memory at the same time.
ESP.VMIN.S[16/32].A	Find the minimum signed value in all elements of a QR register.

Instructions	Description
ESP.VCMP.EQ.S[8/16/32]	Compare two 1-byte/2-byte/4-byte values, set all bits to 1 when the two values are equal, or set all bits to 0 when they are not equal.
ESP.VCMP.LT.S[8/16/32]	Compare two 1-byte/2-byte/4-byte values, set all bits to 1 when the former value is smaller than the latter one, or set them to 0 otherwise.
ESP.VCMP.GT.S[8/16/32]	Compare two 1-byte/2-byte/4-byte values, set all bits to 1 when the former value is larger than the latter one, or set them to 0 otherwise.
ESP.VCLAMP.S16	Clamp all elements in a QR register to a specified range.
ESP.VSAT.[U/S][8/16/32]	Saturate all elements in a QR register.

4.5.6 Bitwise Logical Instructions

Bitwise logical instructions provide OR, AND, XOR, and NOT on QR registers.

Table 4.5-12. Bitwise Logical Instructions

Instructions	Description
ESP.ORQ	Bitwise logical OR $qa = qx \mid qy$
ESP.XORQ	Bitwise logical XOR $qa = qx \text{ ^ } qy$
ESP.ANDQ	Bitwise logical AND $qa = qx \& qy$
ESP.NOTQ	Bitwise NOT $qa = \sim qx$

4.5.7 Shift Instructions

Shift instructions include 4-byte vector left and right shifts, and shifts of concatenated 16-byte data. For vector shifts, the amount comes from SAR. For concatenated 16-byte shifts, the amount can come from SAR_BYTE, an immediate, or the lower bits of an AR register. Choose the instruction that fits your application. These shifts follow signed-shift semantics where applicable.

Table 4.5-13. Shift Instructions

Instructions	Description
ESP.SRC.Q	Perform logical right-shift on the spliced 16-byte data, and the shift value is determined by the SAR_BYTE register.
ESP.SRC.Q.LD.XP	Perform logical right-shift on the spliced 16-byte data, and the shift value is determined by the SAR_BYTE register. At the same time, read 16-byte data from memory and add a register value to the read address.
ESP.SRC.Q.LD.IP	Perform logical right-shift on the spliced 16-byte data, and the shift value is determined by the SAR_BYTE register. At the same time, read 16-byte data from memory and add an immediate number to the read address.
ESP.SLCI.2Q	Perform logical left-shift on the spliced 16-byte data, and the shift value is determined by the immediate value.
ESP.SLCXXP.2Q	Perform logical left-shift on the spliced 16-byte data, and the shift value is determined by the value in the AR register.
ESP.SRCI.2Q	Perform logical right-shift on the spliced 16-byte data, and the shift value is determined by the immediate value.

Instructions	Description
ESP.SRCXXP.2Q	Perform logical right-shift on the spliced 16-byte data, and the shift value is determined by the value in the AR register.
ESP.SRCQ.128.ST.INCP	Perform logical right-shift on the spliced 16-byte data, which will be written to memory after the shift.
ESP.VSR.U32	Perform vector arithmetic right-shift on the 4-byte data.
ESP.VSR.S32	Perform vector arithmetic right-shift on the 4-byte data.
ESP.VSL.S32 hline ESP.VSL.U32	Perform vector arithmetic left-shift on the 4-byte signed data. Perform vector arithmetic left-shift on the 4-byte unsigned data.
ESP.VSRD.[8/16/32]	Perform vector arithmetic right or left shift on the 1-byte/2-byte/4-byte data.
ESP.VSLD.[8/16/32]	Perform vector arithmetic right or left shift on the 1-byte/2-byte/4-byte data.

4.5.8 Fast Fourier Transform (FFT) Instructions

FFT instructions include butterfly operations, bit reversal, and real FFT helpers.

4.5.8.1 Butterfly Computation Instructions

These instructions support radix-2 butterfly computation.

Table 4.5-14. Butterfly Computation Instructions

Instructions	Description
ESP.FFT.R2BF.S16.	Perform radix-2 butterfly computation.
ESP.FFT.R2BF.S16.ST.INCP	Perform radix-2 butterfly computation, and write the 16-byte result to memory at the same time.
ESP.FFT.CMUL.S16.LD.XP	Perform radix-2 complex butterfly computation, and read 16-byte data from memory at the same time.
ESP.FFT.CMUL.S16.ST.XP	Perform radix-2 complex butterfly computation, and write the 16-byte data (consists of the result and partial data segments in QR register) to memory at the same time.

4.5.8.2 Bit Reverse Instruction

The bit-reversal width is set by FFT_BIT_WIDTH.

Table 4.5-15. Bit Reverse Instruction

Instruction	Description
ESP.FFT.BITREV	Bit-reverse indices using FFT_BIT_WIDTH.

4.5.8.3 Real FFT Instructions

A single real-FFT instruction can combine additions, multiplications, shifts, and related operations.

Table 4.5-16. Real Number FFT Instructions

Instructions	Description
ESP.FFT.AMS.S16.LD.INCP.UAUP	Perform complex calculations and read 16-byte data from memory at the same time, then output 16-byte aligned data.
ESP.FFT.AMS.S16.LD.INCP	Perform complex calculations and read 16-byte data from memory at the same time. Add 16 to address register.
ESP.FFT.AMS.S16.LD.R32.DECP	Perform complex calculations and read 16-byte data from memory at the same time. Reverse the word order of read data. Add 16 to address register.
ESP.FFT.AMS.S16.ST.INCP	Perform complex calculation and write 16-byte data (consists of the data in AR and partial data segments in QR) to memory at the same time.
ESP.FFT.VST.R32.DECP	Splice the QR register in 2-byte unit, shift the result and write this 16-byte data to memory.

4.6 Instruction Performance

On a pipelined processor, the ideal case is to issue one instruction per cycle. The ESP32-P4 processor uses a five-stage pipeline: F (fetch), D (decode), E (execute), M (memory), and W (write-back). Table 4.6-1 summarizes each stage.

Table 4.6-1. Five-Stage Pipeline of the ESP32-P4 Processor

Pipeline Stage	Number	Operation
F	-	Fetch instructions
D	0	Read general-purpose registers AR and QR Decode instructions, detect interlocks, and forward operands
E	1	For arithmetic instructions, run the ALU (add, subtract, multiply, and so on) For loads, generate the virtual address For branches, select the target address
M	2	Issue memory reads and writes
W	3	Write results and loaded data back to registers

An instruction can enter the pipeline only when its operands and required hardware resources are ready. In practice, the following hazards can stall the pipeline and delay instruction issue.

4.6.1 Data Hazard

If instruction A writes register X (an explicit general-purpose register or an implicit special register) and instruction B reads the same register, then B depends on A. If A produces the value at the end of stage SA and B needs it at the start of stage SB, A must be issued $D = max(SA - SB + 1, 0)$ cycles before B.
If B is fetched fewer than D cycles after A, the processor delays issuing B until D cycles have elapsed. That delay is called an interlock.
For example, if SA for A is W and SB for B is E, then $D = max(2 - 1 + 1, 0) = 2$, as shown in Figure 4.6-1.

When a result becomes available at the end of a stage, dependent instructions normally wait until it is written to a register. The processor also supports bypass (forwarding): it can forward a value from the producing stage to the consuming stage without waiting for write-back.

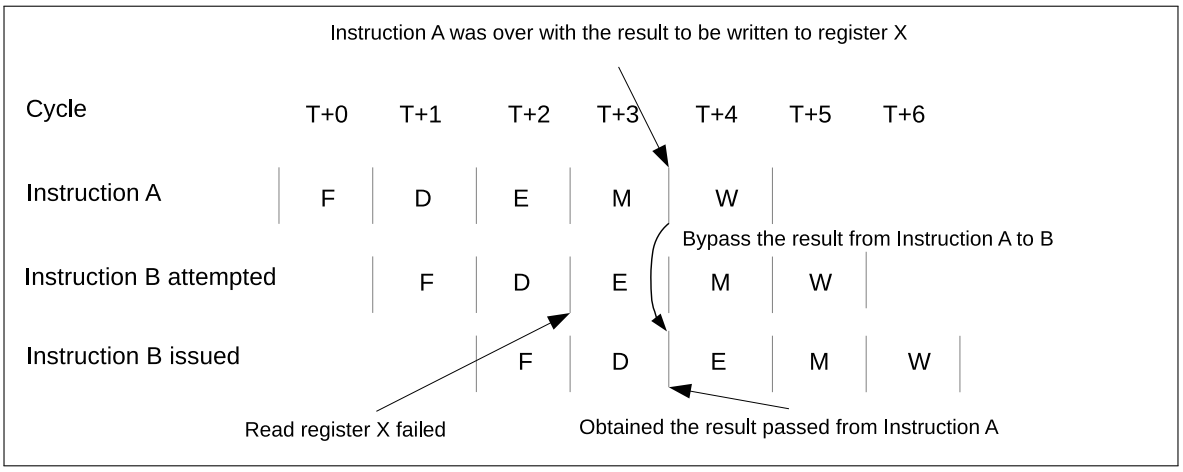


Figure 4.6-1. Interlock Caused by Operand Dependency

Data dependencies depend on which operands each instruction uses and at which stages they are read or written. Section 4.7 lists the use and def stages for each instruction operand.

4.6.2 Hardware Resource Hazard

If multiple instructions need the same hardware resource in the same cycle, only one may use it and the others are delayed. For example, the processor has eight 16-bit multipliers. If instruction C needs all eight in stage M and instruction D needs four in stage E, and C and D are issued in cycles T+0 and T+1 as in Figure 4.6-2, both contend for four multipliers in cycle T+3. The processor then delays D by one cycle to avoid the conflict.

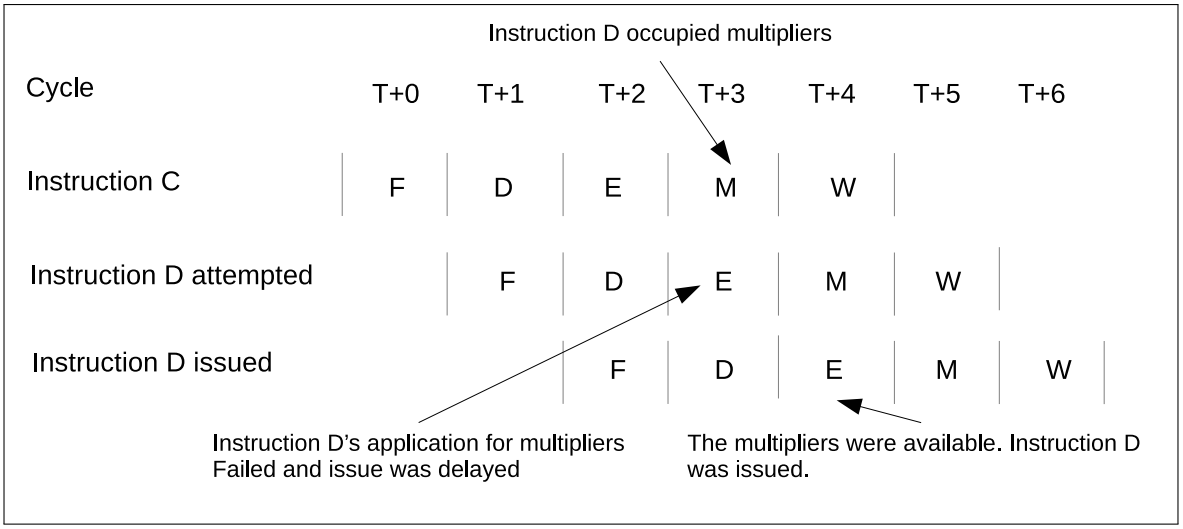


Figure 4.6-2. Hardware Resource Hazard

4.6.3 Control Hazard

Data and resource hazards can often be reduced by reordering code; control hazards are harder to hide. Conditional code becomes branch and jump instructions: if the condition is true, fetch continues from the target; otherwise sequential fetch continues. When a taken branch redirects fetch, as in Figure 4.6-3, instructions already in the F and D stages are discarded, which costs two pipeline cycles.

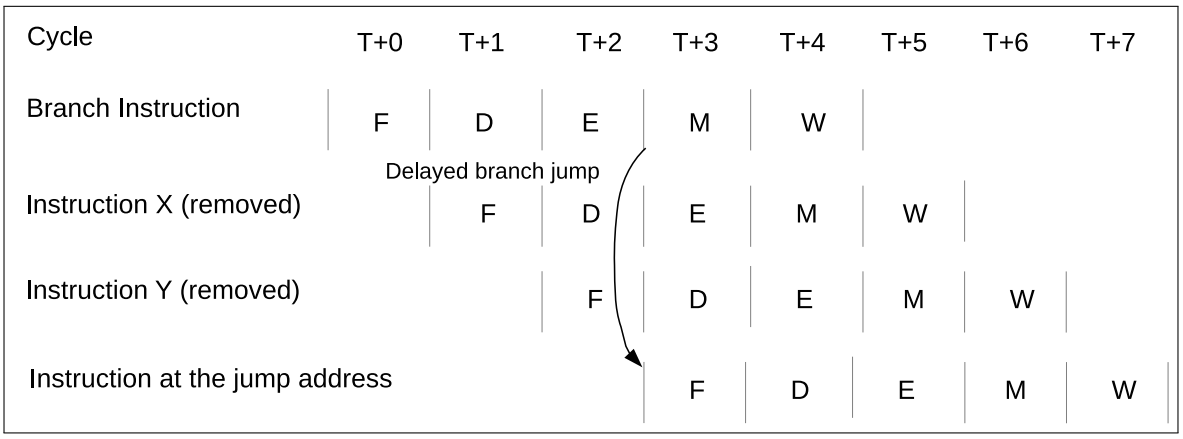


Figure 4.6-3. Control Hazard

4.7 Extended Instruction Functional Description

4.7.1 Instruction Symbols and Field Definitions

Table 4.7-1 describes the meaning of the symbols used in instruction descriptions. When reading the per-instruction functional descriptions below, use this table to understand operands and encoding fields.

`[N:M]` denotes a bit range within a field. Both the upper and lower bit positions N and M are inclusive, so the field width is `(N-M+1)`. For example, `Qx[2:0]` consists of 3 bits: bit0, bit1, and bit2; `Qx[1]` represents the value of bit1.

Table 4.7-1. Instruction Field Names and Descriptions

Name			Description
r*	32-bit RISC-V standard general-purpose registers	rs1	In type or In-out type. 32-bit general-purpose register which stores address and data value.
		rs2	In type. 32-bit general-purpose register which stores shift value, increment value, or calculation result.
		rd	Out type. 32-bit general-purpose register which stores the output value.
q*	128-bit general-purpose registers	qu	In/Out/Inout type. 128-bit general-purpose register which stores the load/store data or exchange data.
		qv	In/Out/Inout type. 128-bit general-purpose register.
		qx, qy, qw	In type. Stores data used for vector operations.
		qz	Out type. Stores results of vector operations.
sat			1-bit immediate value. Enables or disables saturation for the instruction result.
rm			3-bit immediate value ranging from 0 to 7. Selects the rounding mode for right-shift operations. Value 7 (DYNAMIC) uses `vxrm` from the CFG register.
sel2			1-bit immediate value ranging from 0 to 1. Used to select signals.
sel4,upd4			2-bit immediate value ranging from 0 to 3. Used to select signals.
sel8			3-bit immediate value ranging from 0 to 7. Used to select signals.
sel16			4-bit immediate value ranging from 0 to 15. Used to select signals.
sar2			1-bit immediate value ranging from 0 to 1. Represents shifting numbers.
sar4			2-bit immediate value ranging from 0 to 3. Represents shifting numbers.
sar16			4-bit immediate value ranging from 0 to 15. Represents shifting numbers.

Cont'd on next page

Table 4.7-1 – cont'd from previous page

Name	Description
imm1	7-bit unsigned immediate value ranging from 0 to 127 with an interval of 1. This is used to show the size of the updated read/write operation address value.
imm2	7-bit unsigned immediate value ranging from 0 to 254 with an interval of 2. This is used to show the size of the updated read/write operation address value.
imm4	8-bit signed immediate value ranging from -256 to 252 with an interval of 8. This is used to show the size of the updated read/write operation address value.
imm16	8-bit signed immediate value ranging from -2048 to 2032 with an interval of 16. This is used to show the size of the updated read/write operation address value.

4.7.2 List of Instructions

4.7.2.1 ESP.VCMULAS.S16.QACC.H

Syntax

ESP.VCMULAS.S16.QACC.H qx,qy,sat

Description

This instruction performs multiplication of 16-bit vectors and stores the result in `QACC_H`.

Operation

```
add_0[64:0] = QACC_H[ 63:  0] + (qx[ 79: 64]*qy[ 79: 64] - qx[95:80]*qy[95:80])
add_1[64:0] = QACC_H[127: 64] + (qx[ 95: 80]*qy[ 79: 64] + qx[79:64]*qy[95:80])
add_2[64:0] = QACC_H[191:128] + (qx[111:96]*qy[111:96] - qx[127:112]*qy[127:112])
add_3[64:0] = QACC_H[255:192] + (qx[127:112]*qy[111:96] - qx[111:96]*qy[127:112])
QACC_H[ 63:  0] = min(max(add_0[64:0], -2^{63}), 2^{63}-1)
QACC_H[127: 64] = min(max(add_1[64:0], -2^{63}), 2^{63}-1)
QACC_H[191:128] = min(max(add_2[64:0], -2^{63}), 2^{63}-1)
QACC_H[255:192] = min(max(add_3[64:0], -2^{63}), 2^{63}-1)
```

Schedule

Operands	Use	Def
qx	1	-
qy	1	-
QACC_H	2	2

Format

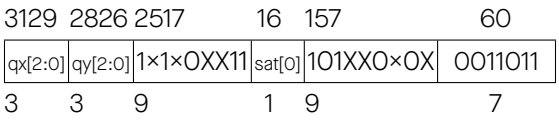


Figure 4.7-1. ESP.VCMULAS.S16.QACC.H

4.7.2.2 ESP.VCMULAS.S16.QACC.H.LD.IP

Syntax

ESP.VCMULAS.S16.QACC.H.LD.IP qu,rs1,imm16,qx,qy,sat

Description

This instruction performs multiplication of 16-bit vectors and stores the result in `QACC_H`.

During the operation, it loads the 16-byte data from the memory into the register `qu`. After the access is completed, the value in the register `rs1` is incremented by a 4-bit sign-extended constant in the instruction code segment, left-shifted by 4.

imm16 starts at -128, ends at 112, with a step of 16.

Operation

```
add_0[64:0] = QACC_H[63:0] + (qx[79:64]*qy[79:64] - qx[95:80]*qy[95:80])
add_1[64:0] = QACC_H[127:64] + (qx[95:80]*qy[79:64] + qx[79:64]*qy[95:80])
add_2[64:0] = QACC_H[191:128] + (qx[111:96]*qy[111:96] - qx[127:112]*qy[127:112])
add_3[64:0] = QACC_H[255:192] + (qx[127:112]*qy[111:96] - qx[111:96]*qy[127:112])

QACC_H[ 63:  0] = min(max(add_0[64:0], -2^{63}), 2^{63}-1)
QACC_H[127: 64] = min(max(add_1[64:0], -2^{63}), 2^{63}-1)
QACC_H[191:128] = min(max(add_2[64:0], -2^{63}), 2^{63}-1)
QACC_H[255:192] = min(max(add_3[64:0], -2^{63}), 2^{63}-1)
qu[127:0] = load128(rs1[31:0])
rs1[31:0] = rs1[31:0] + imm16
```

Schedule

Operands	Use	Def
qu	-	2
rs1	1	1
qx	1	-
qy	1	-
QACC_H	2	2

Format

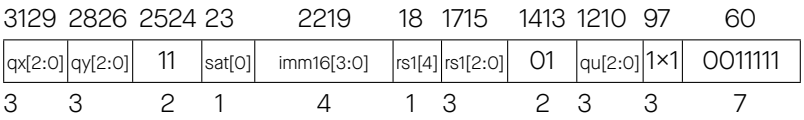


Figure 4.7-2. ESP.VCMULAS.S16.QACC.H.LD.IP

4.7.2.3 ESP.VCMULAS.S16.QACC.H.LD.XP

Syntax

ESP.VCMULAS.S16.QACC.H.LD.XP qu,rs1,rs2,qx,qy,sat

Description

This instruction performs multiplication of 16-bit vectors and stores the result in `QACC_H`.

During the operation, it loads the 16-byte data from the memory into the register `qu`. After the access is completed, the value in the register `rs1` is incremented by the value of the register `rs2`.

Operation

```
add_0[64:0] = QACC_H[63:0] + (qx[79:64]*qy[79:64] - qx[95:80]*qy[95:80])
add_1[64:0] = QACC_H[127:64] + (qx[95:80]*qy[79:64] + qx[79:64]*qy[95:80])
add_2[64:0] = QACC_H[191:128] + (qx[111:96]*qy[111:96] - qx[127:112]*qy[127:112])
add_3[64:0] = QACC_H[255:192] + (qx[127:112]*qy[111:96] - qx[111:96]*qy[127:112])

QACC_H[ 63:  0] = min(max(add_0[64:0], -2^{63}), 2^{63}-1)
QACC_H[127: 64] = min(max(add_1[64:0], -2^{63}), 2^{63}-1)
QACC_H[191:128] = min(max(add_2[64:0], -2^{63}), 2^{63}-1)
QACC_H[255:192] = min(max(add_3[64:0], -2^{63}), 2^{63}-1)

qu[127:0] = load128(rs1[31:0])
rs1[31:0] = rs1[31:0] + rs2[31:0]
```

Schedule

Operands	Use	Def
qu	-	2
rs1	1	1
rs2	1	-
qx	1	-
qy	1	-
QACC_H	2	2

Format

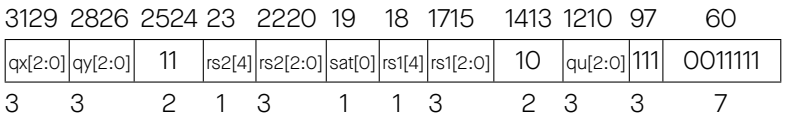


Figure 4.7-3. ESP.VCMULAS.S16.QACC.H.LD.XP

4.7.2.4 ESP.VCMULAS.S16.QACC.L

Syntax

ESP.VCMULAS.S16.QACC.L qx,qy,sat

Description

This instruction performs multiplication of 16-bit vectors and stores the result in `QACC_L`.

Operation

```
    add_0[64:0] = QACC_L[63:0] + (qx[15:0]*qy[15:0] - qx[31:16]*qy[31:16])
    add_1[64:0] = QACC_L[127:64] + (qx[31:16]*qy[15:0] + qx[15:0]*qy[31:16])
    add_2[64:0] = QACC_L[191:128] + (qx[47:32]*qy[47:32] - qx[63:48]*qy[63:48])
    add_3[64:0] = QACC_L[255:192] + (qx[63:48]*qy[47:32] - qx[47:32]*qy[63:48])

    QACC_L[ 63:  0] = min(max(add_0[64:0], -2^{63}), 2^{63}-1)
    QACC_L[127: 64] = min(max(add_1[64:0], -2^{63}), 2^{63}-1)
    QACC_L[191:128] = min(max(add_2[64:0], -2^{63}), 2^{63}-1)
    QACC_L[255:192] = min(max(add_3[64:0], -2^{63}), 2^{63}-1)
```

Schedule

Operands	Use	Def
qx	1	-
qy	1	-
QACC_L	2	2

Format

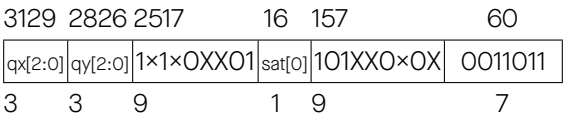


Figure 4.7-4. ESP.VCMULAS.S16.QACC.L

4.7.2.5 ESP.VCMULAS.S16.QACC.L.LD.IP

Syntax

ESP.VCMULAS.S16.QACC.L.LD.IP qu,rs1,imm16,qx,qy,sat

Description

This instruction performs multiplication of 16-bit vectors and stores the result in `QACC_L`.

During the operation, it loads the 16-byte data from the memory into the register `qu`. After the access is completed, the value in the register `rs1` is incremented by a 4-bit sign-extended constant in the instruction code segment, left-shifted by 4.

imm16 starts at -128, ends at 112, with a step of 16.

Operation

```
    add_0[64:0] = QACC_L[63:0] + (qx[15:0]*qy[15:0] - qx[31:16]*qy[31:16])
    add_1[64:0] = QACC_L[127:64] + (qx[31:16]*qy[15:0] + qx[15:0]*qy[31:16])
    add_2[64:0] = QACC_L[191:128] + (qx[47:32]*qy[47:32] - qx[63:48]*qy[63:48])
    add_3[64:0] = QACC_L[255:192] + (qx[63:48]*qy[47:32] - qx[47:32]*qy[63:48])

    QACC_L[ 63:  0] = min(max(add_0[64:0], -2^{63}), 2^{63}-1)
    QACC_L[127: 64] = min(max(add_1[64:0], -2^{63}), 2^{63}-1)
    QACC_L[191:128] = min(max(add_2[64:0], -2^{63}), 2^{63}-1)
    QACC_L[255:192] = min(max(add_3[64:0], -2^{63}), 2^{63}-1)

    qu[127:0] = load128(rs1[31:0])
    rs1[31:0] = rs1[31:0] + imm16
```

Schedule

Operands	Use	Def
qu	-	2
rs1	1	1
qx	1	-
qy	1	-
QACC_L	2	2

Format

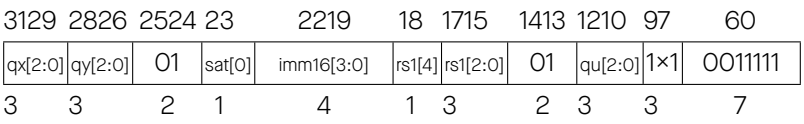


Figure 4.7-5. ESP.VCMULAS.S16.QACC.L.LD.IP

4.7.2.6 ESP.VCMULAS.S16.QACC.L.LD.XP

Syntax

ESP.VCMULAS.S16.QACC.L.LD.XP qu,rs1,rs2,qx,qy,sat

Description

This instruction performs multiplication of 16-bit vectors and stores the result in `QACC_L`.

During the operation, it loads the 16-byte data from the memory into the register `qu`. After the access is completed, the value in the register `rs1` is incremented by the value of the register `rs2`.

Operation

```
add_0[64:0] = QACC_L[63:0] + (qx[15:0]*qy[15:0] - qx[31:16]*qy[31:16])
add_1[64:0] = QACC_L[127:64] + (qx[31:16]*qy[15:0] + qx[15:0]*qy[31:16])
add_2[64:0] = QACC_L[191:128] + (qx[47:32]*qy[47:32] - qx[63:48]*qy[63:48])
add_3[64:0] = QACC_L[255:192] + (qx[63:48]*qy[47:32] - qx[47:32]*qy[63:48])

QACC_L[ 63:  0] = min(max(add_0[64:0], -2^{63}), 2^{63}-1)
QACC_L[127: 64] = min(max(add_1[64:0], -2^{63}), 2^{63}-1)
QACC_L[191:128] = min(max(add_2[64:0], -2^{63}), 2^{63}-1)
QACC_L[255:192] = min(max(add_3[64:0], -2^{63}), 2^{63}-1)

qu[127:0] = load128(rs1[31:0])
rs1[31:0] = rs1[31:0] + rs2[31:0]
```

Schedule

Operands	Use	Def
qu	-	2
rs1	1	1
rs1	1	-
qx	1	-
qy	1	-
QACC_L	2	2

Format

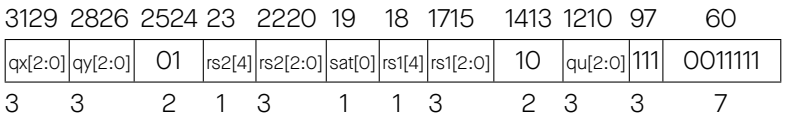


Figure 4.7-6. ESP.VCMULAS.S16.QACC.L.LD.XP

4.7.2.7 ESP.VCMULAS.S8.QACC.H

Syntax

ESP.VCMULAS.S8.QACC.H qx,qy,sat

Description

This instruction performs multiplication of 8-bit vectors and stores the result in `QACC_H`.

Operation

```
add_0[32:0] = QACC_H[ 31:  0] + (qx[ 71: 64]*qy[ 71: 64] - qx[ 79: 72]*qy[ 79: 72])
add_1[32:0] = QACC_H[ 63: 32] + (qx[ 79: 72]*qy[ 71: 64] - qx[ 71: 64]*qy[ 79: 72])

add_2[32:0] = QACC_H[ 95: 64] + (qx[ 87: 80]*qy[ 87: 80] - qx[ 95: 88]*qy[ 95: 88])
add_3[32:0] = QACC_H[127: 96] + (qx[ 95: 88]*qy[ 87: 80] - qx[ 87: 80]*qy[ 95: 88])

add_4[32:0] = QACC_H[159:128] + (qx[103: 96]*qy[103: 96] - qx[111:104]*qy[111:104])
add_5[32:0] = QACC_H[191:160] + (qx[111:104]*qy[103: 96] - qx[103: 96]*qy[111:104])

add_6[32:0] = QACC_H[223:192] + (qx[119:112]*qy[119:112] - qx[127:120]*qy[127:120])
add_7[32:0] = QACC_H[255:224] + (qx[127:120]*qy[119:112] - qx[119:112]*qy[127:120])

QACC_H[ 31:  0] = min(max(add_0[32:0], -2^{31}), 2^{31}-1)
QACC_H[ 63: 32] = min(max(add_1[32:0], -2^{31}), 2^{31}-1)
QACC_H[ 95: 64] = min(max(add_2[32:0], -2^{31}), 2^{31}-1)
QACC_H[127: 96] = min(max(add_3[32:0], -2^{31}), 2^{31}-1)
QACC_H[159:128] = min(max(add_4[32:0], -2^{31}), 2^{31}-1)
QACC_H[191:160] = min(max(add_5[32:0], -2^{31}), 2^{31}-1)
QACC_H[223:192] = min(max(add_6[32:0], -2^{31}), 2^{31}-1)
QACC_H[255:224] = min(max(add_7[32:0], -2^{31}), 2^{31}-1)
```

Schedule

Operands	Use	Def
qx	1	-
qy	1	-
QACC_H	2	2

Format

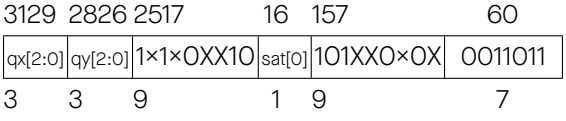


Figure 4.7-7. ESP.VCMULAS.S8.QACC.H

4.7.2.8 ESP.VCMULAS.S8.QACC.H.LD.IP

Syntax

ESP.VCMULAS.S8.QACC.H.LD.IP qu,rs1,imm16,qx,qy,sat

Description

This instruction performs multiplication of 8-bit vectors and stores the result in `QACC_H`.

During the operation, it loads the 16-byte data from the memory into the register `qu`. After the access is completed, the value in the register `rs1` is incremented by a 4-bit sign-extended constant in the instruction code segment, left-shifted by 4.

imm16 starts at -128, ends at 112, with a step of 16.

Operation

```
    add_0[32:0] = QACC_H[ 31:  0] + (qx[ 71: 64]*qy[ 71: 64] - qx[ 79: 72]*qy[ 79: 72])
    add_1[32:0] = QACC_H[ 63: 32] + (qx[ 79: 72]*qy[ 71: 64] - qx[ 71: 64]*qy[ 79: 72])

    add_2[32:0] = QACC_H[ 95: 64] + (qx[ 87: 80]*qy[ 87: 80] - qx[ 95: 88]*qy[ 95: 88])
    add_3[32:0] = QACC_H[127: 96] + (qx[ 95: 88]*qy[ 87: 80] - qx[ 87: 80]*qy[ 95: 88])

    add_4[32:0] = QACC_H[159:128] + (qx[103: 96]*qy[103: 96] - qx[111:104]*qy[111:104])
    add_5[32:0] = QACC_H[191:160] + (qx[111:104]*qy[103: 96] - qx[103: 96]*qy[111:104])

    add_6[32:0] = QACC_H[223:192] + (qx[119:112]*qy[119:112] - qx[127:120]*qy[127:120])
    add_7[32:0] = QACC_H[255:224] + (qx[127:120]*qy[119:112] - qx[119:112]*qy[127:120])

    QACC_H[ 31:  0] = min(max(add_0[32:0], -2^{31}), 2^{31}-1)
    QACC_H[ 63: 32] = min(max(add_1[32:0], -2^{31}), 2^{31}-1)
    QACC_H[ 95: 64] = min(max(add_2[32:0], -2^{31}), 2^{31}-1)
    QACC_H[127: 96] = min(max(add_3[32:0], -2^{31}), 2^{31}-1)
    QACC_H[159:128] = min(max(add_4[32:0], -2^{31}), 2^{31}-1)
    QACC_H[191:160] = min(max(add_5[32:0], -2^{31}), 2^{31}-1)
    QACC_H[223:192] = min(max(add_6[32:0], -2^{31}), 2^{31}-1)
    QACC_H[255:224] = min(max(add_7[32:0], -2^{31}), 2^{31}-1)

    qu[127:0] = load128(rs1[31:0])
    rs1[31:0] = rs1[31:0] + signed32(imm16)
```

Schedule

Operands	Use	Def
qu	-	2
rs1	1	1

Operands	Use	Def
qx	1	-
qy	1	-
QACC_H	2	2

Format

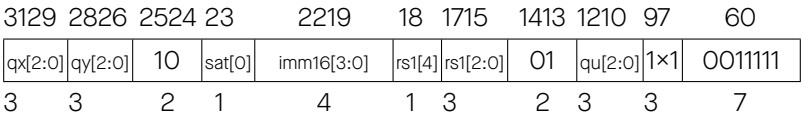


Figure 4.7-8. ESP.VCMULAS.S8.QACC.H.LD.IP

4.7.2.9 ESP.VCMULAS.S8.QACC.H.LD.XP

Syntax

ESP.VCMULAS.S8.QACC.H.LD.XP qu,rs1,rs2,qx,qy,sat

Description

This instruction performs multiplication of 8-bit vectors and stores the result in `QACC_H`.

During the operation, it loads the 16-byte data from the memory into the register `qu`. After the access is completed, the value in the register `rs1` is incremented by the value of the register `rs2`.

Operation

```
add_0[32:0] = QACC_H[ 31:  0] + (qx[ 71: 64]*qy[ 71: 64] - qx[ 79: 72]*qy[ 79: 72])
add_1[32:0] = QACC_H[ 63: 32] + (qx[ 79: 72]*qy[ 71: 64] - qx[ 71: 64]*qy[ 79: 72])

add_2[32:0] = QACC_H[ 95: 64] + (qx[ 87: 80]*qy[ 87: 80] - qx[ 95: 88]*qy[ 95: 88])
add_3[32:0] = QACC_H[127: 96] + (qx[ 95: 88]*qy[ 87: 80] - qx[ 87: 80]*qy[ 95: 88])

add_4[32:0] = QACC_H[159:128] + (qx[103: 96]*qy[103: 96] - qx[111:104]*qy[111:104])
add_5[32:0] = QACC_H[191:160] + (qx[111:104]*qy[103: 96] - qx[103: 96]*qy[111:104])

add_6[32:0] = QACC_H[223:192] + (qx[119:112]*qy[119:112] - qx[127:120]*qy[127:120])
add_7[32:0] = QACC_H[255:224] + (qx[127:120]*qy[119:112] - qx[119:112]*qy[127:120])

QACC_H[ 31:  0] = min(max(add_0[32:0], -2^{31}), 2^{31}-1)
QACC_H[ 63: 32] = min(max(add_1[32:0], -2^{31}), 2^{31}-1)
QACC_H[ 95: 64] = min(max(add_2[32:0], -2^{31}), 2^{31}-1)
QACC_H[127: 96] = min(max(add_3[32:0], -2^{31}), 2^{31}-1)
QACC_H[159:128] = min(max(add_4[32:0], -2^{31}), 2^{31}-1)
QACC_H[191:160] = min(max(add_5[32:0], -2^{31}), 2^{31}-1)
QACC_H[223:192] = min(max(add_6[32:0], -2^{31}), 2^{31}-1)
QACC_H[255:224] = min(max(add_7[32:0], -2^{31}), 2^{31}-1)

qu[127:0] = load128(rs1[31:0])
rs1[31:0] = rs1[31:0] + rs2[31:0]
```

Schedule

Operands	Use	Def
qu	-	2
rs1	1	1
rs2	1	-
qx	1	-

Operands	Use	Def
qy	1	-
QACC_H	2	2

Format

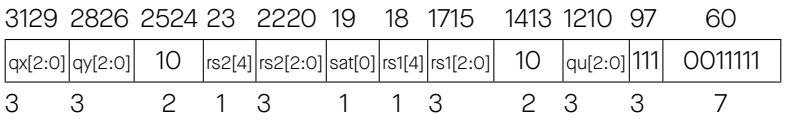


Figure 4.7-9. ESP.VCMULAS.S8.QACC.H.LD.XP

4.7.2.10 ESP.VCMULAS.S8.QACC.L

Syntax

ESP.VCMULAS.S8.QACC.L qx,qy,sat

Description

This instruction performs multiplication of 8-bit vectors and stores the result in `QACC_L`.

Operation

```
add_0[32:0] = QACC_L[ 31:  0] + (qx[  7:  0]*qy[  7:  0] - qx[ 15:  8]*qy[ 15:
    8])
add_1[32:0] = QACC_L[ 63: 32] + (qx[ 15:  8]*qy[  7:  0] - qx[  7:  0]*qy[ 15:
    8])

add_2[32:0] = QACC_L[ 95: 64] + (qx[ 23: 16]*qy[ 23: 16] - qx[ 31: 24]*qy[ 31:
    24])
add_3[32:0] = QACC_L[127: 96] + (qx[ 31: 24]*qy[ 23: 16] - qx[ 23: 16]*qy[ 31:
    24])

add_4[32:0] = QACC_L[159:128] + (qx[ 39: 32]*qy[ 39: 32] - qx[ 47: 40]*qy[ 47:
    40])
add_5[32:0] = QACC_L[191:160] + (qx[ 47: 40]*qy[ 39: 32] - qx[ 39: 32]*qy[ 47:
    40])

add_6[32:0] = QACC_L[223:192] + (qx[ 55: 48]*qy[ 55: 48] - qx[ 63: 56]*qy[ 63:
    56])
add_7[32:0] = QACC_L[255:224] + (qx[ 63: 56]*qy[ 55: 48] - qx[ 55: 48]*qy[ 63:
    56])

QACC_L[ 31:  0] = min(max(add_0[32:0], -2^{31}), 2^{31}-1)
QACC_L[ 63: 32] = min(max(add_1[32:0], -2^{31}), 2^{31}-1)
QACC_L[ 95: 64] = min(max(add_2[32:0], -2^{31}), 2^{31}-1)
QACC_L[127: 96] = min(max(add_3[32:0], -2^{31}), 2^{31}-1)
QACC_L[159:128] = min(max(add_4[32:0], -2^{31}), 2^{31}-1)
QACC_L[191:160] = min(max(add_5[32:0], -2^{31}), 2^{31}-1)
QACC_L[223:192] = min(max(add_6[32:0], -2^{31}), 2^{31}-1)
QACC_L[255:224] = min(max(add_7[32:0], -2^{31}), 2^{31}-1)
```

Schedule

Operands	Use	Def
qx	1	-
qy	1	-
QACC_L	2	2

Format

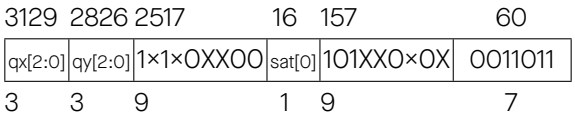


Figure 4.7-10. ESP.VCMULAS.S8.QACC.L

4.7.2.11 ESP.VCMULAS.S8.QACC.L.LD.IP

Syntax

ESP.VCMULAS.S8.QACC.L.LD.IP qu,rs1,imm16,qx,qy,sat

Description

This instruction performs multiplication of 8-bit vectors and stores the result in `QACC_L`.

During the operation, it loads the 16-byte data from the memory into the register `qu`. After the access is completed, the value in the register `rs1` is incremented by a 4-bit sign-extended constant in the instruction code segment, left-shifted by 4.

imm16 starts at -128, ends at 112, with a step of 16.

Operation

```
add_0[32:0] = QACC_L[ 31:  0] + (qx[  7:  0]*qy[  7:  0] - qx[ 15:  8]*qy[ 15:  8])
add_1[32:0] = QACC_L[ 63: 32] + (qx[ 15:  8]*qy[  7:  0] - qx[  7:  0]*qy[ 15:  8])

add_2[32:0] = QACC_L[ 95: 64] + (qx[ 23: 16]*qy[ 23: 16] - qx[ 31: 24]*qy[ 31: 24])
add_3[32:0] = QACC_L[127: 96] + (qx[ 31: 24]*qy[ 23: 16] - qx[ 23: 16]*qy[ 31: 24])

add_4[32:0] = QACC_L[159:128] + (qx[ 39: 32]*qy[ 39: 32] - qx[ 47: 40]*qy[ 47: 40])
add_5[32:0] = QACC_L[191:160] + (qx[ 47: 40]*qy[ 39: 32] - qx[ 39: 32]*qy[ 47: 40])

add_6[32:0] = QACC_L[223:192] + (qx[ 55: 48]*qy[ 55: 48] - qx[ 63: 56]*qy[ 63: 56])
add_7[32:0] = QACC_L[255:224] + (qx[ 63: 56]*qy[ 55: 48] - qx[ 55: 48]*qy[ 63: 56])

QACC_L[ 31:  0] = min(max(add_0[32:0], -2^{31}), 2^{31}-1)
QACC_L[ 63: 32] = min(max(add_1[32:0], -2^{31}), 2^{31}-1)
QACC_L[ 95: 64] = min(max(add_2[32:0], -2^{31}), 2^{31}-1)
QACC_L[127: 96] = min(max(add_3[32:0], -2^{31}), 2^{31}-1)
QACC_L[159:128] = min(max(add_4[32:0], -2^{31}), 2^{31}-1)
QACC_L[191:160] = min(max(add_5[32:0], -2^{31}), 2^{31}-1)
QACC_L[223:192] = min(max(add_6[32:0], -2^{31}), 2^{31}-1)
QACC_L[255:224] = min(max(add_7[32:0], -2^{31}), 2^{31}-1)

qu[127:0] = load128(rs1[31:0])
rs1[31:0] = rs1[31:0] + imm16
```

Schedule

Operands	Use	Def
qu	-	2
rs1	1	1

Operands	Use	Def
qx	1	-
qy	1	-
QACC_L	2	2

Format

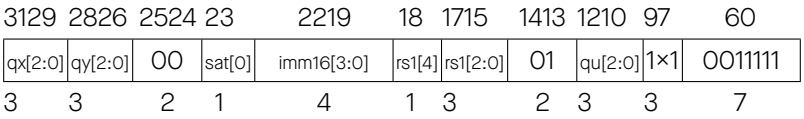


Figure 4.7-11. ESP.VCMULAS.S8.QACC.L.LD.IP

4.7.2.12 ESP.VCMULAS.S8.QACC.L.LD.XP

Syntax

ESP.VCMULAS.S8.QACC.L.LD.XP qu,rs1,rs2,qx,qy,sat

Description

This instruction performs multiplication of 8-bit vectors and stores the result in `QACC_L`.

During the operation, it loads the 16-byte data from the memory into the register `qu`. After the access is completed, the value in the register `rs1` is incremented by the value of the register `rs2`.

Operation

```
add_0[32:0] = QACC_L[ 31:  0] + (qx[  7:  0]*qy[  7:  0] - qx[ 15:  8]*qy[ 15:
    8])
add_1[32:0] = QACC_L[ 63: 32] + (qx[ 15:  8]*qy[  7:  0] - qx[  7:  0]*qy[ 15:
    8])

add_2[32:0] = QACC_L[ 95: 64] + (qx[ 23: 16]*qy[ 23: 16] - qx[ 31: 24]*qy[ 31:
    24])
add_3[32:0] = QACC_L[127: 96] + (qx[ 31: 24]*qy[ 23: 16] - qx[ 23: 16]*qy[ 31:
    24])

add_4[32:0] = QACC_L[159:128] + (qx[ 39: 32]*qy[ 39: 32] - qx[ 47: 40]*qy[ 47:
    40])
add_5[32:0] = QACC_L[191:160] + (qx[ 47: 40]*qy[ 39: 32] - qx[ 39: 32]*qy[ 47:
    40])

add_6[32:0] = QACC_L[223:192] + (qx[ 55: 48]*qy[ 55: 48] - qx[ 63: 56]*qy[ 63:
    56])
add_7[32:0] = QACC_L[255:224] + (qx[ 63: 56]*qy[ 55: 48] - qx[ 55: 48]*qy[ 63:
    56])

QACC_L[ 31:  0] = min(max(add_0[32:0], -2^{31}), 2^{31}-1)
QACC_L[ 63: 32] = min(max(add_1[32:0], -2^{31}), 2^{31}-1)
QACC_L[ 95: 64] = min(max(add_2[32:0], -2^{31}), 2^{31}-1)
QACC_L[127: 96] = min(max(add_3[32:0], -2^{31}), 2^{31}-1)
QACC_L[159:128] = min(max(add_4[32:0], -2^{31}), 2^{31}-1)
QACC_L[191:160] = min(max(add_5[32:0], -2^{31}), 2^{31}-1)
QACC_L[223:192] = min(max(add_6[32:0], -2^{31}), 2^{31}-1)
QACC_L[255:224] = min(max(add_7[32:0], -2^{31}), 2^{31}-1)

qu[127:0] = load128(rs1[31:0])
rs1[31:0] = rs1[31:0] + rs2[31:0]
```

Schedule

Operands	Use	Def
qu	-	2
rs1	1	1
rs2	1	-
qx	1	-

Operands	Use	Def
qy	1	-
QACC_L	2	2

Format

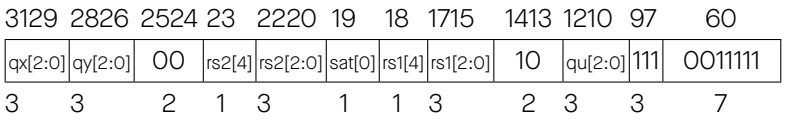


Figure 4.7-12. ESP.VCMULAS.S8.QACC.L.LD.XP

4.7.2.13 ESP.VMULAS.S16.QACC

Syntax

ESP.VMULAS.S16.QACC qx,qy,sat

Description

This instruction divides registers qx and qy into 8 data segments of 16 bits each. Then, the signed multiplication result of the 8 sets of segments is added to the corresponding 64-bit data segment in the special registers QACC_H and QACC_L respectively. The calculated result is saturated to a 64-bit signed number and then stored in the corresponding 64-bit data segment in QACC_H and QACC_L.

Operation

```
QACC_L[63 :  0] = min(max(QACC_L[63 :  0] + qx[15 :  0] * qy[15 :  0], -2^63),
    2^63-1)
QACC_L[127: 64] = min(max(QACC_L[127: 64] + qx[31 : 16] * qy[31 : 16], -2^63),
    2^63-1)
QACC_L[191:128] = min(max(QACC_L[191:128] + qx[47 : 32] * qy[47 : 32], -2^63),
    2^63-1)
QACC_L[255:192] = min(max(QACC_L[255:192] + qx[63 : 48] * qy[63 : 48], -2^63),
    2^63-1)
QACC_H[63 :  0] = min(max(QACC_H[63 :  0] + qx[79 : 64] * qy[79 : 64], -2^63),
    2^63-1)
QACC_H[127: 64] = min(max(QACC_H[127: 64] + qx[95 : 80] * qy[95 : 80], -2^63),
    2^63-1)
QACC_H[191:128] = min(max(QACC_H[191:128] + qx[111: 96] * qy[111: 96], -2^63),
    2^63-1)
QACC_H[255:192] = min(max(QACC_H[255:192] + qx[127:112] * qy[127:112], -2^63),
    2^63-1)
```

Schedule

Operands	Use	Def
qx	1	-
qy	1	-
QACC_H	2	2
QACC_L	2	2

Format

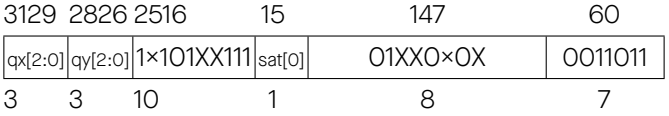


Figure 4.7-13. ESP.VMULAS.S16.QACC

4.7.2.14 ESP.VMULAS.S16.QACC.LD.IP

Syntax

ESP.VMULAS.S16.QACC.LD.IP qu,rs1,imm16,qx,qy,sat

Description

This instruction divides registers `qx` and `qy` into 8 data segments of 16 bits each. Then, the signed multiplication result of the 8 sets of segments is added to the corresponding 64-bit data segment in special registers `QACC_H` and `QACC_L` respectively. The calculated result is saturated to a 64-bit signed number and then stored in the corresponding 64-bit data segment in `QACC_H` and `QACC_L`.

During the operation, it loads the 16-byte data from the memory into the register `qu`. After the access is completed, the value in the register `rs1` is incremented by the 4-bit sign-extended constant in the instruction code segment, which is left-shifted by 4.

The value of imm starts at -128, ends at 112, and steps by 16.

Operation

```
QACC_L[63 :  0] = min(max(QACC_L[63 :  0] + qx[15 :  0] * qy[15 :  0], -2^63),
    2^63-1)
QACC_L[127: 64] = min(max(QACC_L[127: 64] + qx[31 : 16] * qy[31 : 16], -2^63),
    2^63-1)
QACC_L[191:128] = min(max(QACC_L[191:128] + qx[47 : 32] * qy[47 : 32], -2^63),
    2^63-1)
QACC_L[255:192] = min(max(QACC_L[255:192] + qx[63 : 48] * qy[63 : 48], -2^63),
    2^63-1)
QACC_H[63 :  0] = min(max(QACC_H[63 :  0] + qx[79 : 64] * qy[79 : 64], -2^63),
    2^63-1)
QACC_H[127: 64] = min(max(QACC_H[127: 64] + qx[95 : 80] * qy[95 : 80], -2^63),
    2^63-1)
QACC_H[191:128] = min(max(QACC_H[191:128] + qx[111: 96] * qy[111: 96], -2^63),
    2^63-1)
QACC_H[255:192] = min(max(QACC_H[255:192] + qx[127:112] * qy[127:112], -2^63),
    2^63-1)

qu[127:0] = load128(rs1[31:0])
rs1[31:0] = rs1[31:0] + imm
```

Schedule

Operands	Use	Def
qu	-	2
rs1	1	1
qx	1	-
qy	1	-
QACC_H	2	2
QACC_L	2	2

Format

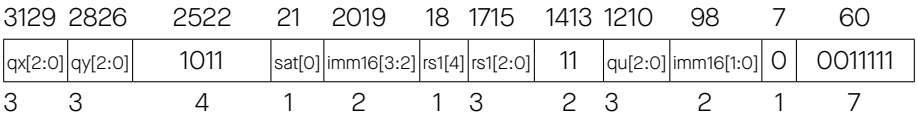


Figure 4.7-14. ESP.VMULAS.S16.QACC.LD.IP

4.7.2.15 ESP.VMULAS.S16.QACC.LD.XP

Syntax

ESP.VMULAS.S16.QACC.LD.XP qu,rs1,rs2,qx,qy,sat

Description

This instruction divides registers `qx` and `qy` into 8 data segments of 16 bits each. Then, the signed multiplication result of the 8 sets of segments is added to the corresponding 64-bit data segment in special registers `QACC_H` and `QACC_L` respectively. The calculated result is saturated to a 64-bit signed number and then stored in the corresponding 64-bit data segment in `QACC_H` and `QACC_L`.

During the operation, it loads the 16-byte data from the memory into the register `qu`. After the access is completed, the value in the register `rs1` is incremented by the value of the register `rs2`.

Operation

```
QACC_L[63 :  0] = min(max(QACC_L[63 :  0] + qx[15 :  0] * qy[15 :  0], -2^63),
    2^63-1)
QACC_L[127: 64] = min(max(QACC_L[127: 64] + qx[31 : 16] * qy[31 : 16], -2^63),
    2^63-1)
QACC_L[191:128] = min(max(QACC_L[191:128] + qx[47 : 32] * qy[47 : 32], -2^63),
    2^63-1)
QACC_L[255:192] = min(max(QACC_L[255:192] + qx[63 : 48] * qy[63 : 48], -2^63),
    2^63-1)
QACC_H[63 :  0] = min(max(QACC_H[63 :  0] + qx[79 : 64] * qy[79 : 64], -2^63),
    2^63-1)
QACC_H[127: 64] = min(max(QACC_H[127: 64] + qx[95 : 80] * qy[95 : 80], -2^63),
    2^63-1)
QACC_H[191:128] = min(max(QACC_H[191:128] + qx[111: 96] * qy[111: 96], -2^63),
    2^63-1)
QACC_H[255:192] = min(max(QACC_H[255:192] + qx[127:112] * qy[127:112], -2^63),
    2^63-1)

qu[127:0] = load128(rs1[31:0])
rs1[31:0] = rs1[31:0] + rs2[31:0]
```

Schedule

Operands	Use	Def
qu	-	2
rs1	1	1
rs2	1	-
qx	1	-
qy	1	-
QACC_H	2	2
QACC_L	2	2

Format

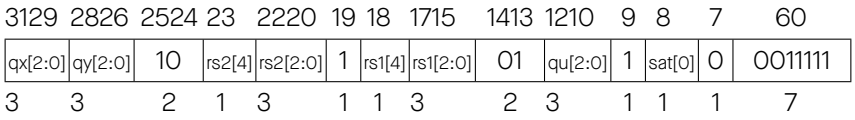


Figure 4.7-15. ESP.VMULAS.S16.QACC.LD.XP

4.7.2.16 ESP.VMULAS.S16.QACC.ST.IP

Syntax

ESP.VMULAS.S16.QACC.ST.IP qu,rs1,imm16,qx,qy,sat

Description

This instruction divides registers `qx` and `qy` into 8 data segments of 16 bits each. Then, the signed multiplication results of the 8 sets of segments are added to the corresponding 64-bit data segments in special registers `QACC_H` and `QACC_L` respectively. The calculated result is saturated to a 64-bit signed number and then stored in the corresponding 64-bit data segment in `QACC_H` and `QACC_L`.

During the operation, the value of the register `qu` is stored in memory. After the access is completed, the value in the register `rs1` is incremented by a 4-bit sign-extended constant in the instruction code segment, which is left-shifted by 4.

The imm starts at -128, ends at 112, and steps by 16.

Operation

```
QACC_L[63 :  0] = min(max(QACC_L[63 :  0] + qx[15 :  0] * qy[15 :  0], -2^63),
    2^63-1)
QACC_L[127: 64] = min(max(QACC_L[127: 64] + qx[31 : 16] * qy[31 : 16], -2^63),
    2^63-1)
QACC_L[191:128] = min(max(QACC_L[191:128] + qx[47 : 32] * qy[47 : 32], -2^63),
    2^63-1)
QACC_L[255:192] = min(max(QACC_L[255:192] + qx[63 : 48] * qy[63 : 48], -2^63),
    2^63-1)
QACC_H[63 :  0] = min(max(QACC_H[63 :  0] + qx[79 : 64] * qy[79 : 64], -2^63),
    2^63-1)
QACC_H[127: 64] = min(max(QACC_H[127: 64] + qx[95 : 80] * qy[95 : 80], -2^63),
    2^63-1)
QACC_H[191:128] = min(max(QACC_H[191:128] + qx[111: 96] * qy[111: 96], -2^63),
    2^63-1)
QACC_H[255:192] = min(max(QACC_H[255:192] + qx[127:112] * qy[127:112], -2^63),
    2^63-1)

qu[127:0] => store128(rs1[31:0])
rs1[31:0] = rs1[31:0] + imm
```

Schedule

Operands	Use	Def
qu	2	-
rs1	1	1
qx	1	-
qy	1	-
QACC_H	2	2
QACC_L	2	2

Format

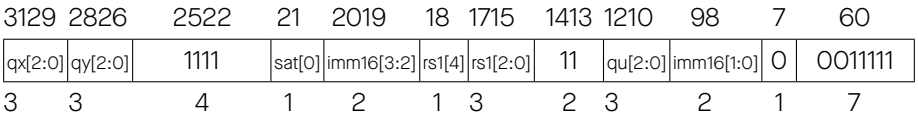


Figure 4.7-16. ESP.VMULAS.S16.QACC.ST.IP

4.7.2.17 ESP.VMULAS.S16.QACC.ST.XP

Syntax

ESP.VMULAS.S16.QACC.ST.XP qu,rs1,rs2,qx,qy,sat

Description

This instruction divides registers `qx` and `qy` into 8 data segments of 16 bits each. Then, the signed multiplication results of the 8 sets of segments are added to the corresponding 64-bit data segments in special registers `QACC_H` and `QACC_L` respectively. The calculated results are saturated to a 64-bit signed number and then stored in the corresponding 64-bit data segments in `QACC_H` and `QACC_L`.

During the operation, the value of the register `qu` is stored in memory. After the access is completed, the value in the register `rs1` is incremented by the value of the register `rs2`.

Operation

```
QACC_L[63 :  0] = min(max(QACC_L[63 :  0] + qx[15 :  0] * qy[15 :  0], -2^63),
    2^63-1)
QACC_L[127: 64] = min(max(QACC_L[127: 64] + qx[31 : 16] * qy[31 : 16], -2^63),
    2^63-1)
QACC_L[191:128] = min(max(QACC_L[191:128] + qx[47 : 32] * qy[47 : 32], -2^63),
    2^63-1)
QACC_L[255:192] = min(max(QACC_L[255:192] + qx[63 : 48] * qy[63 : 48], -2^63),
    2^63-1)
QACC_H[63 :  0] = min(max(QACC_H[63 :  0] + qx[79 : 64] * qy[79 : 64], -2^63),
    2^63-1)
QACC_H[127: 64] = min(max(QACC_H[127: 64] + qx[95 : 80] * qy[95 : 80], -2^63),
    2^63-1)
QACC_H[191:128] = min(max(QACC_H[191:128] + qx[111: 96] * qy[111: 96], -2^63),
    2^63-1)
QACC_H[255:192] = min(max(QACC_H[255:192] + qx[127:112] * qy[127:112], -2^63),
    2^63-1)

qu[127:0] => store128(rs1[31:0])
rs1[31:0] = rs1[31:0] + rs2[31:0]
```

Schedule

Operands	Use	Def
qu	2	-
rs1	1	1
rs2	1	-
qx	1	-
qy	1	-
QACC_H	2	2
QACC_L	2	2

Format

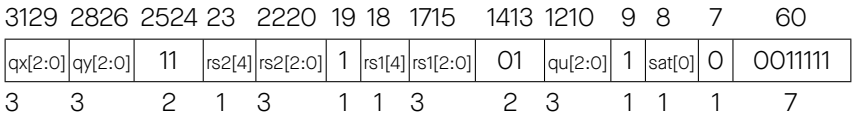


Figure 4.7-17. ESP.VMULAS.S16.QACC.ST.XP

4.7.2.18 ESP.VMULAS.S16.XACC

Syntax

ESP.VMULAS.S16.XACC qx,qy,sat

Description

This instruction divides registers qx and qy into 8 data segments of 16 bits each. Then, it performs a signed multiply-accumulate operation on each of the 8 sets of segments. The accumulated result is added to the value in the special register XACC. Then, the resulting sum is saturated and stored in XACC.

Operation

```
res[39 :  0] = qx[15 :  0] * qy[15 :  0] + qx[31 : 16] * qy[31 : 16] + … + qx
    [127:112] * qy[127:112]
XACC = min(max(XACC + res[39 :  0], -2^39), 2^39-1)
```

Schedule

Operands	Use	Def
qx	1	-
qy	1	-
XACC	3	3

Format

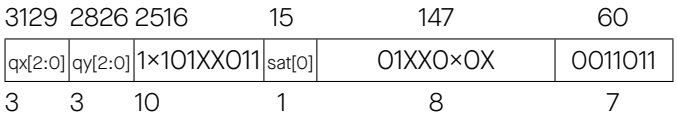


Figure 4.7-18. ESP.VMULAS.S16.XACC

4.7.2.19 ESP.VMULAS.S16.XACC.LD.IP

Syntax

ESP.VMULAS.S16.XACC.LD.IP qu,rs1,imm16,qx,qy,sat

Description

This instruction divides registers `qx` and `qy` into 8 data segments of 16 bits each. Then, it performs a signed multiply-accumulate operation on the 8 sets of segments respectively. The accumulated result is added to the value in the special register `XACC`. Then, the sum obtained is saturated and stored in `XACC`.

During the operation, the 16-byte data is loaded from the memory to the register `qu`. After the access is completed, the value in the register `rs1` is incremented by a 4-bit sign-extended constant in the instruction code segment, which is left-shifted by 4.

The value of imm starts at -128, ends at 112, and steps by 16.

Operation

```
res[39 :  0] = qx[15 :  0] * qy[15 :  0] + qx[31 : 16] * qy[31 : 16] + … + qx
    [127:112] * qy[127:112]
XACC = min(max(XACC + res[39 :  0], -2^39), 2^39-1)
qu[127:0] = load128(rs1[31:0])
rs1[31:0] = rs1[31:0] + imm
```

Schedule

Operands	Use	Def
qu	-	2
rs1	1	1
qx	1	-
qy	1	-
XACC	3	3

Format

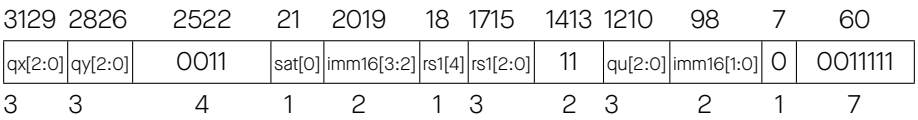


Figure 4.7-19. ESP.VMULAS.S16.XACC.LD.IP

4.7.2.20 ESP.VMULAS.S16.XACC.LD.XP

Syntax

ESP.VMULAS.S16.XACC.LD.XP qu,rs1,rs2,qx,qy,sat

Description

This instruction divides registers `qx` and `qy` into 8 data segments of 16 bits each. Then, it performs a signed multiply-accumulate operation on each of the 8 segment sets. The accumulated result is added to the value in the special register `XACC`. The resulting sum is then saturated and stored in `XACC`.

During the operation, the 16-byte data is loaded from the memory into the register `qu`. After the access is completed, the value in the register `rs1` is incremented by the value of the register `rs2`.

Operation

```
res[39 :  0] = qx[15 :  0] * qy[15 :  0] + qx[31 : 16] * qy[31 : 16] + … + qx
    [127:112] * qy[127:112]
XACC = min(max(XACC + res[39 :  0], -2^39), 2^39-1)
qu[127:0] = load128(rs1[31:0])
rs1[31:0] = rs1[31:0] + rs2[31:0]
```

Schedule

Operands	Use	Def
qu	-	2
rs1	1	1
rs2	1	-
qx	1	-
qy	1	-
XACC	3	3

Format

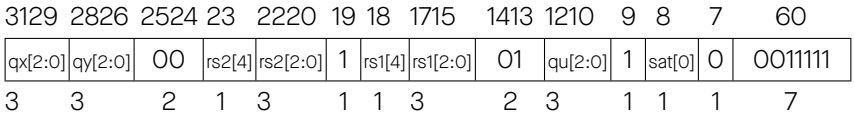


Figure 4.7-20. ESP.VMULAS.S16.XACC.LD.XP

4.7.2.21 ESP.VMULAS.S16.XACC.ST.IP

Syntax

ESP.VMULAS.S16.XACC.ST.IP qu,rs1,imm16,qx,qy,sat

Description

This instruction divides registers `qx` and `qy` into 8 data segments of 16 bits each. Then, it performs a signed multiply-accumulate operation on the 8 sets of segments respectively. The accumulated result is added to the value in the special register `XACC`. Then, the sum obtained is saturated and then stored in `XACC`.

During the operation, the value of the register `qu` is stored in the memory. After the access is completed, the value in the register `rs1` is incremented by a 4-bit sign-extended constant in the instruction code segment, which is left-shifted by 4.

The value of imm starts at -128, ends at 112, and steps by 16.

Operation

```
res[39 :  0] = qx[15 :  0] * qy[15 :  0] + qx[31 : 16] * qy[31 : 16] + … + qx
    [127:112] * qy[127:112]
XACC = min(max(XACC + res[39 :  0], -2^39), 2^39-1)
qu[127:0] => store128(rs1[31:0])
rs1[31:0] = rs1[31:0] + imm
```

Schedule

Operands	Use	Def
qu	2	-
rs1	1	1
qx	1	-
qy	1	-
XACC	3	3

Format

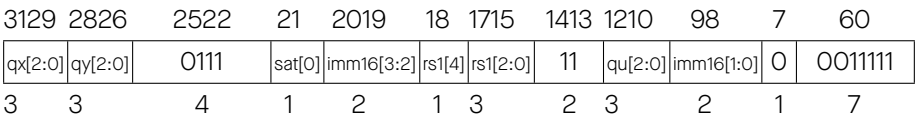


Figure 4.7-21. ESP.VMULAS.S16.XACC.ST.IP

4.7.2.22 ESP.VMULAS.S16.XACC.ST.XP

Syntax

ESP.VMULAS.S16.XACC.ST.XP qu,rs1,rs2,qx,qy,sat

Description

This instruction divides registers `qx` and `qy` into 8 data segments of 16 bits each. Then, it performs a signed multiply-accumulate operation on each of the 8 segment sets. The accumulated result is added to the value in the special register `XACC`. The resulting sum is then saturated and stored in `XACC`.

During the operation, the value of the register `qu` is stored in memory. After the access is completed, the value in the register `rs1` is incremented by the value of the register `rs2`.

Operation

```
res[39 :  0] = qx[15 :  0] * qy[15 :  0] + qx[31 : 16] * qy[31 : 16] + … + qx
    [127:112] * qy[127:112]
XACC = min(max(XACC + res[39 :  0], -2^39), 2^39-1)
qu[127:0] => store128(rs1[31:0])
rs1[31:0] = rs1[31:0] + rs2[31:0]
```

Schedule

Operands	Use	Def
qu	2	-
rs1	1	1
rs2	1	-
qx	1	-
qy	1	-
XACC	3	3

Format

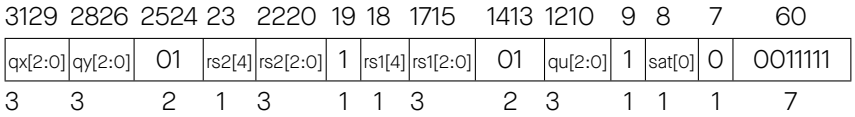


Figure 4.7-22. ESP.VMULAS.S16.XACC.ST.XP

4.7.2.23 ESP.VMULAS.S8.QACC

Syntax

ESP.VMULAS.S8.QACC qx,qy,sat

Description

This instruction divides registers qx and qy into 16 data segments of 8 bits each. Then, the signed multiplication results of the 16 sets of segments are added to the corresponding 32-bit data segments in special registers QACC_H and QACC_L respectively. The calculated results are saturated to 32-bit signed numbers and then stored in the corresponding 32-bit data segments in QACC_H and QACC_L.

Operation

```
QACC_L[31 :  0] = min(max(QACC_L[31 :  0] + qx[7  :  0] * qy[7  :  0], -2^31),
    2^31-1)
QACC_L[63 : 32] = min(max(QACC_L[63 : 32] + qx[15 :  8] * qy[15 :  8], -2^31),
    2^31-1)
QACC_L[95 : 64] = min(max(QACC_L[95 : 64] + qx[23 : 16] * qy[23 : 16], -2^31),
    2^31-1)
QACC_L[127: 96] = min(max(QACC_L[127: 96] + qx[31 : 24] * qy[31 : 24], -2^31),
    2^31-1)
QACC_L[159:128] = min(max(QACC_L[159:128] + qx[39 : 32] * qy[39 : 32], -2^31),
    2^31-1)
QACC_L[191:160] = min(max(QACC_L[191:160] + qx[47 : 40] * qy[47 : 40], -2^31),
    2^31-1)
QACC_L[223:192] = min(max(QACC_L[223:192] + qx[55 : 48] * qy[55 : 48], -2^31),
    2^31-1)
QACC_L[255:224] = min(max(QACC_L[255:224] + qx[63 : 56] * qy[63 : 56], -2^31),
    2^31-1)
QACC_H[31 :  0] = min(max(QACC_H[31 :  0] + qx[71 : 64] * qy[71 : 64], -2^31),
    2^31-1)
QACC_H[63 : 32] = min(max(QACC_H[63 : 32] + qx[79 : 72] * qy[79 : 72], -2^31),
    2^31-1)
QACC_H[95 : 64] = min(max(QACC_H[95 : 64] + qx[87 : 80] * qy[87 : 80], -2^31),
    2^31-1)
QACC_H[127: 96] = min(max(QACC_H[127: 96] + qx[95 : 88] * qy[95 : 88], -2^31),
    2^31-1)
QACC_H[159:128] = min(max(QACC_H[159:128] + qx[103: 96] * qy[103: 96], -2^31),
    2^31-1)
QACC_H[191:160] = min(max(QACC_H[191:160] + qx[111:104] * qy[111:104], -2^31),
    2^31-1)
QACC_H[223:192] = min(max(QACC_H[223:192] + qx[119:112] * qy[119:112], -2^31),
    2^31-1)
QACC_H[255:224] = min(max(QACC_H[255:224] + qx[127:120] * qy[127:120], -2^31),
    2^31-1)
```

Schedule

Operands	Use	Def
qx	1	-
qy	1	-
QACC_H	2	2

Operands	Use	Def
QACC_L	2	2

Format

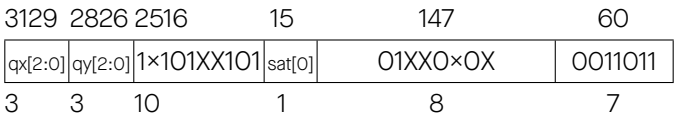


Figure 4.7-23. ESP.VMULAS.S8.QACC

4.7.2.24 ESP.VMULAS.S8.QACC.LD.IP

Syntax

ESP.VMULAS.S8.QACC.LD.IP qu,rs1,imm16,qx,qy,sat

Description

This instruction divides registers `qx` and `qy` into 16 data segments of 8 bits each. Then, the signed multiplication results of the 16 sets of segments are added to the corresponding 32-bit data segments in special registers `QACC_H` and `QACC_L` respectively. The calculated result is saturated to a 32-bit signed number and then stored in the corresponding 32-bit data segment in `QACC_H` and `QACC_L`.

During the operation, the 16-byte data is loaded from the memory to the register `qu`. After the access is completed, the value in the register `rs1` is incremented by the 4-bit sign-extended constant in the instruction code segment, which is left-shifted by 4.

imm starts at -128, ends at 112, with a step of 16.

Operation

```
QACC_L[31 :  0] = min(max(QACC_L[31 :  0] + qx[7  :  0] * qy[7  :  0], -2^31),
    2^31-1)
QACC_L[63 : 32] = min(max(QACC_L[63 : 32] + qx[15 :  8] * qy[15 :  8], -2^31),
    2^31-1)
QACC_L[95 : 64] = min(max(QACC_L[95 : 64] + qx[23 : 16] * qy[23 : 16], -2^31),
    2^31-1)
QACC_L[127: 96] = min(max(QACC_L[127: 96] + qx[31 : 24] * qy[31 : 24], -2^31),
    2^31-1)
QACC_L[159:128] = min(max(QACC_L[159:128] + qx[39 : 32] * qy[39 : 32], -2^31),
    2^31-1)
QACC_L[191:160] = min(max(QACC_L[191:160] + qx[47 : 40] * qy[47 : 40], -2^31),
    2^31-1)
QACC_L[223:192] = min(max(QACC_L[223:192] + qx[55 : 48] * qy[55 : 48], -2^31),
    2^31-1)
QACC_L[255:224] = min(max(QACC_L[255:224] + qx[63 : 56] * qy[63 : 56], -2^31),
    2^31-1)
QACC_H[31 :  0] = min(max(QACC_H[31 :  0] + qx[71 : 64] * qy[71 : 64], -2^31),
    2^31-1)
QACC_H[63 : 32] = min(max(QACC_H[63 : 32] + qx[79 : 72] * qy[79 : 72], -2^31),
    2^31-1)
QACC_H[95 : 64] = min(max(QACC_H[95 : 64] + qx[87 : 80] * qy[87 : 80], -2^31),
    2^31-1)
QACC_H[127: 96] = min(max(QACC_H[127: 96] + qx[95 : 88] * qy[95 : 88], -2^31),
    2^31-1)
QACC_H[159:128] = min(max(QACC_H[159:128] + qx[103: 96] * qy[103: 96], -2^31),
    2^31-1)
QACC_H[191:160] = min(max(QACC_H[191:160] + qx[111:104] * qy[111:104], -2^31),
    2^31-1)
QACC_H[223:192] = min(max(QACC_H[223:192] + qx[119:112] * qy[119:112], -2^31),
    2^31-1)
QACC_H[255:224] = min(max(QACC_H[255:224] + qx[127:120] * qy[127:120], -2^31),
    2^31-1)

qu[127:0] = load128(rs1[31:0])
```

```
rs1[31:0] = rs1[31:0] + imm
```

Schedule

Operands	Use	Def
qu	-	2
rs1	1	1
qx	1	-
qy	1	-
QACC_H	2	2
QACC_L	2	2

Format

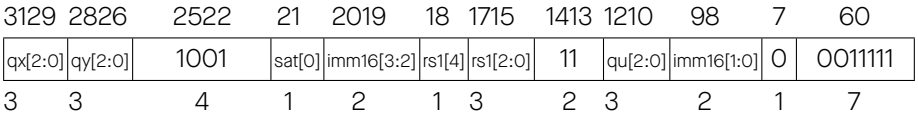


Figure 4.7-24. ESP.VMULAS.S8.QACC.LD.IP

4.7.2.25 ESP.VMULAS.S8.QACC.LD.XP

Syntax

ESP.VMULAS.S8.QACC.LD.XP qu,rs1,rs2,qx,qy,sat

Description

This instruction divides registers `qx` and `qy` into 16 data segments of 8 bits each. Then, the signed multiplication results of the 16 sets of segments are added to the corresponding 32-bit data segments in special registers `QACC_H` and `QACC_L` respectively. The calculated results are saturated to 32-bit signed numbers and then stored in the corresponding 32-bit data segments in `QACC_H` and `QACC_L`.

During the operation, the 16-byte data is loaded from the memory into the register `qu`. After the access is completed, the value in the register `rs1` is incremented by the value of the register `rs2`.

Operation

```
QACC_L[31 :  0] = min(max(QACC_L[31 :  0] + qx[7  :  0] * qy[7  :  0], -2^31),
    2^31-1)
QACC_L[63 : 32] = min(max(QACC_L[63 : 32] + qx[15 :  8] * qy[15 :  8], -2^31),
    2^31-1)
QACC_L[95 : 64] = min(max(QACC_L[95 : 64] + qx[23 : 16] * qy[23 : 16], -2^31),
    2^31-1)
QACC_L[127: 96] = min(max(QACC_L[127: 96] + qx[31 : 24] * qy[31 : 24], -2^31),
    2^31-1)
QACC_L[159:128] = min(max(QACC_L[159:128] + qx[39 : 32] * qy[39 : 32], -2^31),
    2^31-1)
QACC_L[191:160] = min(max(QACC_L[191:160] + qx[47 : 40] * qy[47 : 40], -2^31),
    2^31-1)
QACC_L[223:192] = min(max(QACC_L[223:192] + qx[55 : 48] * qy[55 : 48], -2^31),
    2^31-1)
QACC_L[255:224] = min(max(QACC_L[255:224] + qx[63 : 56] * qy[63 : 56], -2^31),
    2^31-1)
QACC_H[31 :  0] = min(max(QACC_H[31 :  0] + qx[71 : 64] * qy[71 : 64], -2^31),
    2^31-1)
QACC_H[63 : 32] = min(max(QACC_H[63 : 32] + qx[79 : 72] * qy[79 : 72], -2^31),
    2^31-1)
QACC_H[95 : 64] = min(max(QACC_H[95 : 64] + qx[87 : 80] * qy[87 : 80], -2^31),
    2^31-1)
QACC_H[127: 96] = min(max(QACC_H[127: 96] + qx[95 : 88] * qy[95 : 88], -2^31),
    2^31-1)
QACC_H[159:128] = min(max(QACC_H[159:128] + qx[103: 96] * qy[103: 96], -2^31),
    2^31-1)
QACC_H[191:160] = min(max(QACC_H[191:160] + qx[111:104] * qy[111:104], -2^31),
    2^31-1)
QACC_H[223:192] = min(max(QACC_H[223:192] + qx[119:112] * qy[119:112], -2^31),
    2^31-1)
QACC_H[255:224] = min(max(QACC_H[255:224] + qx[127:120] * qy[127:120], -2^31),
    2^31-1)

qu[127:0] = load128(rs1[31:0])
rs1[31:0] = rs1[31:0] + rs2[31:0]
```

Schedule

Operands	Use	Def
qu	-	2
rs1	1	1
rs2	1	-
qx	1	-
qy	1	-
QACC_H	2	2
QACC_L	2	2

Format

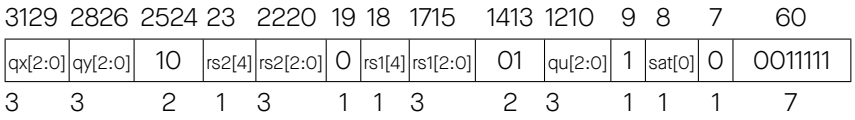


Figure 4.7-25. ESP.VMULAS.S8.QACC.LD.XP

4.7.2.26 ESP.VMULAS.S8.QACC.ST.IP

Syntax

ESP.VMULAS.S8.QACC.ST.IP qu,rs1,imm16,qx,qy,sat

Description

This instruction divides registers `qx` and `qy` into 16 data segments of 8 bits each. Then, the signed multiplication results of the 16 sets of segments are added to the corresponding 32-bit data segments in special registers `QACC_H` and `QACC_L` respectively. The calculated result is saturated to a 32-bit signed number and then stored in the corresponding 32-bit data segment in `QACC_H` and `QACC_L`.

During the operation, the value of the register `qu` is stored in memory. After the access is completed, the value in the register `rs1` is incremented by a 4-bit sign-extended constant in the instruction code segment, which is left-shifted by 4.

The value of imm starts at -128, ends at 112, and steps in increments of 16.

Operation

```
QACC_L[31 :  0] = min(max(QACC_L[31 :  0] + qx[7  :  0] * qy[7  :  0], -2^31),
    2^31-1)
QACC_L[63 : 32] = min(max(QACC_L[63 : 32] + qx[15 :  8] * qy[15 :  8], -2^31),
    2^31-1)
QACC_L[95 : 64] = min(max(QACC_L[95 : 64] + qx[23 : 16] * qy[23 : 16], -2^31),
    2^31-1)
QACC_L[127: 96] = min(max(QACC_L[127: 96] + qx[31 : 24] * qy[31 : 24], -2^31),
    2^31-1)
QACC_L[159:128] = min(max(QACC_L[159:128] + qx[39 : 32] * qy[39 : 32], -2^31),
    2^31-1)
QACC_L[191:160] = min(max(QACC_L[191:160] + qx[47 : 40] * qy[47 : 40], -2^31),
    2^31-1)
QACC_L[223:192] = min(max(QACC_L[223:192] + qx[55 : 48] * qy[55 : 48], -2^31),
    2^31-1)
QACC_L[255:224] = min(max(QACC_L[255:224] + qx[63 : 56] * qy[63 : 56], -2^31),
    2^31-1)
QACC_H[31 :  0] = min(max(QACC_H[31 :  0] + qx[71 : 64] * qy[71 : 64], -2^31),
    2^31-1)
QACC_H[63 : 32] = min(max(QACC_H[63 : 32] + qx[79 : 72] * qy[79 : 72], -2^31),
    2^31-1)
QACC_H[95 : 64] = min(max(QACC_H[95 : 64] + qx[87 : 80] * qy[87 : 80], -2^31),
    2^31-1)
QACC_H[127: 96] = min(max(QACC_H[127: 96] + qx[95 : 88] * qy[95 : 88], -2^31),
    2^31-1)
QACC_H[159:128] = min(max(QACC_H[159:128] + qx[103: 96] * qy[103: 96], -2^31),
    2^31-1)
QACC_H[191:160] = min(max(QACC_H[191:160] + qx[111:104] * qy[111:104], -2^31),
    2^31-1)
QACC_H[223:192] = min(max(QACC_H[223:192] + qx[119:112] * qy[119:112], -2^31),
    2^31-1)
QACC_H[255:224] = min(max(QACC_H[255:224] + qx[127:120] * qy[127:120], -2^31),
    2^31-1)

qu[127:0] => store128(rs1[31:0])
```

```
rs1[31:0] = rs1[31:0] + imm
```

Schedule

Operands	Use	Def
qu	2	-
rs2	1	-
qx	1	-
qy	1	-
QACC_H	2	2
QACC_L	2	2

Format

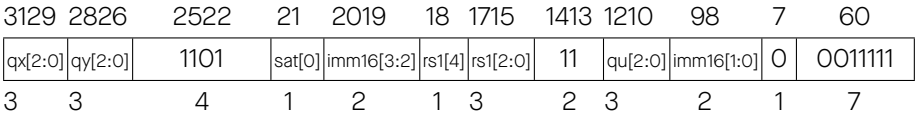


Figure 4.7-26. ESP.VMULAS.S8.QACC.ST.IP

4.7.2.27 ESP.VMULAS.S8.QACC.ST.XP

Syntax

ESP.VMULAS.S8.QACC.ST.XP qu,rs1,rs2,qx,qy,sat

Description

This instruction divides registers `qx` and `qy` into 16 data segments of 8 bits each. Then, the signed multiplication results of the 16 sets of segments are added to the corresponding 32-bit data segments in special registers `QACC_H` and `QACC_L` respectively. The calculated results are saturated to 32-bit signed numbers and then stored in the corresponding 32-bit data segments in `QACC_H` and `QACC_L`.

During the operation, the value of the register `qu` is stored in memory. After the access is completed, the value in the register `rs1` is incremented by the value of the register `rs2`.

Operation

```
QACC_L[31 :  0] = min(max(QACC_L[31 :  0] + qx[7  :  0] * qy[7  :  0], -2^31),
    2^31-1)
QACC_L[63 : 32] = min(max(QACC_L[63 : 32] + qx[15 :  8] * qy[15 :  8], -2^31),
    2^31-1)
QACC_L[95 : 64] = min(max(QACC_L[95 : 64] + qx[23 : 16] * qy[23 : 16], -2^31),
    2^31-1)
QACC_L[127: 96] = min(max(QACC_L[127: 96] + qx[31 : 24] * qy[31 : 24], -2^31),
    2^31-1)
QACC_L[159:128] = min(max(QACC_L[159:128] + qx[39 : 32] * qy[39 : 32], -2^31),
    2^31-1)
QACC_L[191:160] = min(max(QACC_L[191:160] + qx[47 : 40] * qy[47 : 40], -2^31),
    2^31-1)
QACC_L[223:192] = min(max(QACC_L[223:192] + qx[55 : 48] * qy[55 : 48], -2^31),
    2^31-1)
QACC_L[255:224] = min(max(QACC_L[255:224] + qx[63 : 56] * qy[63 : 56], -2^31),
    2^31-1)
QACC_H[31 :  0] = min(max(QACC_H[31 :  0] + qx[71 : 64] * qy[71 : 64], -2^31),
    2^31-1)
QACC_H[63 : 32] = min(max(QACC_H[63 : 32] + qx[79 : 72] * qy[79 : 72], -2^31),
    2^31-1)
QACC_H[95 : 64] = min(max(QACC_H[95 : 64] + qx[87 : 80] * qy[87 : 80], -2^31),
    2^31-1)
QACC_H[127: 96] = min(max(QACC_H[127: 96] + qx[95 : 88] * qy[95 : 88], -2^31),
    2^31-1)
QACC_H[159:128] = min(max(QACC_H[159:128] + qx[103: 96] * qy[103: 96], -2^31),
    2^31-1)
QACC_H[191:160] = min(max(QACC_H[191:160] + qx[111:104] * qy[111:104], -2^31),
    2^31-1)
QACC_H[223:192] = min(max(QACC_H[223:192] + qx[119:112] * qy[119:112], -2^31),
    2^31-1)
QACC_H[255:224] = min(max(QACC_H[255:224] + qx[127:120] * qy[127:120], -2^31),
    2^31-1)

qu[127:0] => store128(rs1[31:0])
rs1[31:0] = rs1[31:0] + rs2[31:0]
```

Schedule

Operands	Use	Def
qu	2	-
rs1	1	1
rs2	1	-
qx	1	-
qy	1	-
QACC_H	2	2
QACC_L	2	2

Format

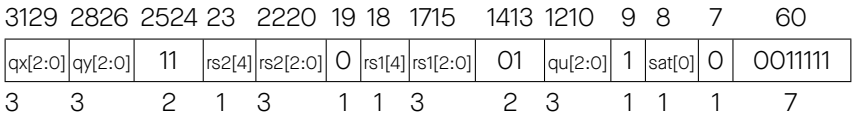


Figure 4.7-27. ESP.VMULAS.S8.QACC.ST.XP

4.7.2.28 ESP.VMULAS.S8.XACC

Syntax

ESP.VMULAS.S8.XACC qx,qy,sat

Description

This instruction divides registers qx and qy into 16 data segments of 8 bits each. Then, it performs a signed multiply-accumulate operation on each of the 16 sets of segments. The accumulated result is added to the value in the special register XACC. The resulting sum is then saturated and stored in XACC.

Operation

```
add0[15:0] = qx[  7:  0] * qy[  7:  0]
add1[15:0] = qx[ 15:  8] * qy[ 15:  8]
...
add15[15:0] = qx[127:120] * qy[127:120]
sum[40:0] = XACC[39:0] + add0[15:0] + add1[15:0] + ... + add15[15:0]

XACC[39:0] = min(max(sum[40:0], -2^{39}), 2^{39}-1)
```

Schedule

Operands	Use	Def
qx	1	-
qy	1	-
XACC	3	3

Format

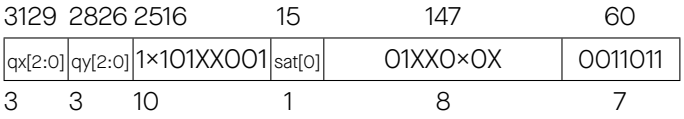


Figure 4.7-28. ESP.VMULAS.S8.XACC

4.7.2.29 ESP.VMULAS.S8.XACC.LD.IP

Syntax

ESP.VMULAS.S8.XACC.LD.IP qu,rs1,imm16,qx,qy,sat

Description

This instruction divides registers `qx` and `qy` into 16 data segments of 8 bits each. Then, it performs a signed multiply-accumulate operation on each of the 16 segment sets. The accumulated result is added to the value in the special register `XACC`. The resulting sum is then saturated and stored in `XACC`.

During the operation, the 16-byte data is loaded from the memory into the register `qu`. After the access is completed, the value in the register `rs1` is incremented by a 4-bit sign-extended constant from the instruction code segment, which is left-shifted by 4.

The imm starts at -128, ends at 112, and steps in increments of 16.

Operation

```
add0[15:0] = qx[  7:  0] * qy[  7:  0]
add1[15:0] = qx[ 15:  8] * qy[ 15:  8]
...
add15[15:0] = qx[127:120] * qy[127:120]
sum[40:0] = XACC[39:0] + add0[15:0] + add1[15:0] + ... + add15[15:0]

XACC[39:0] = min(max(sum[40:0], -2^{39}), 2^{39}-1)
qu[127:0] = load128(rs1[31:0])
rs1[31:0] = rs1[31:0] + imm
```

Schedule

Operands	Use	Def
qu	-	2
rs1	1	1
qx	1	-
qy	1	-
XACC	3	3

Format

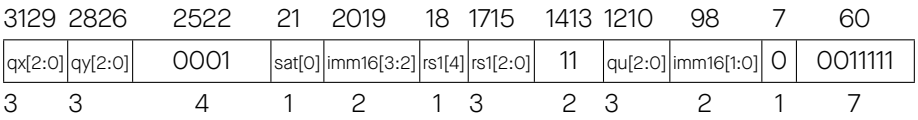


Figure 4.7-29. ESP.VMULAS.S8.XACC.LD.IP

4.7.2.30 ESP.VMULAS.S8.XACC.LD.XP

Syntax

ESP.VMULAS.S8.XACC.LD.XP qu,rs1,rs2,qx,qy,sat

Description

This instruction divides registers `qx` and `qy` into 16 data segments of 8 bits each. Then, it performs a signed multiply-accumulate operation on each of the 16 segments. The accumulated result is added to the value in the special register `XACC`. The resulting sum is then saturated and stored in `XACC`.

During the operation, the 16-byte data is loaded from the memory into the register `qu`. After the access is completed, the value in the register `rs1` is incremented by the value of the register `rs2`.

Operation

```
add0[15:0] = qx[  7:  0] * qy[  7:  0]
add1[15:0] = qx[ 15:  8] * qy[ 15:  8]
...
add15[15:0] = qx[127:120] * qy[127:120]
sum[40:0] = XACC[39:0] + add0[15:0] + add1[15:0] + ... + add15[15:0]

XACC[39:0] = min(max(sum[40:0], -2^{39}), 2^{39}-1)
qu[127:0] = load128(rs1[31:0])
rs1[31:0] = rs1[31:0] + rs2[31:0]
```

Schedule

Operands	Use	Def
qu	-	2
rs1	1	1
rs2	1	-
qx	1	-
qy	1	-
QACC_H	2	2
QACC_L	2	2

Format

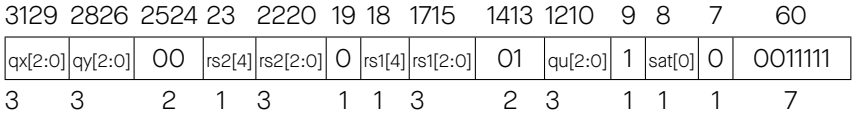


Figure 4.7-30. ESP.VMULAS.S8.XACC.LD.XP

4.7.2.31 ESP.VMULAS.S8.XACC.ST.IP

Syntax

ESP.VMULAS.S8.XACC.ST.IP qu,rs1,imm16,qx,qy,sat

Description

This instruction divides registers `qx` and `qy` into 16 data segments of 8 bits each. Then, it performs a signed multiply-accumulate operation on each of the 16 segments. The accumulated result is added to the value in the special register `XACC`. The resulting sum is then saturated and stored in `XACC`.

During the operation, the register `qu` is stored to memory. After the access is completed, the value in the register `rs1` is incremented by a 4-bit sign-extended constant from the instruction code segment, which is left-shifted by 4.

The imm starts at -128, ends at 112, and steps in increments of 16.

Operation

```
add0[15:0] = qx[  7:  0] * qy[  7:  0]
add1[15:0] = qx[ 15:  8] * qy[ 15:  8]
...
add15[15:0] = qx[127:120] * qy[127:120]
sum[40:0] = XACC[39:0] + add0[15:0] + add1[15:0] + ... + add15[15:0]

XACC[39:0] = min(max(sum[40:0], -2^{39}), 2^{39}-1)
qu[127:0] => store128(rs1[31:0])
rs1[31:0] = rs1[31:0] + imm
```

Schedule

Operands	Use	Def
qu	2	1
rs1	1	1
qx	1	-
qy	1	-
XACC	3	3

Format

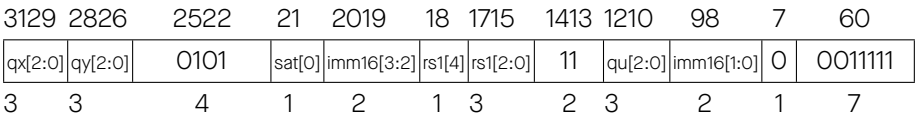


Figure 4.7-31. ESP.VMULAS.S8.XACC.ST.IP

4.7.2.32 ESP.VMULAS.S8.XACC.ST.XP

Syntax

ESP.VMULAS.S8.XACC.ST.XP qu,rs1,rs2,qx,qy,sat

Description

This instruction divides registers `qx` and `qy` into 16 data segments of 8 bits each. Then, it performs a signed multiply-accumulate operation on each of the 16 sets of segments. The accumulated result is added to the value in the special register `XACC`. The resulting sum is then saturated and stored in `XACC`.
During the operation, the value in the register `qu` is stored to memory. After the access is completed, the value in the register `rs1` is incremented by the value of the register `rs2`.

Operation

```
add0[15:0] = qx[  7:  0] * qy[  7:  0]
add1[15:0] = qx[ 15:  8] * qy[ 15:  8]
...
add15[15:0] = qx[127:120] * qy[127:120]
sum[40:0] = XACC[39:0] + add0[15:0] + add1[15:0] + ... + add15[15:0]

XACC[39:0] = min(max(sum[40:0], -2^{39}), 2^{39}-1)
qu[127:0] => store128(rs1[31:0])
rs1[31:0] = rs1[31:0] + rs2[31:0]
```

Schedule

Operands	Use	Def
qu	-	2
rs1	1	1
rs2	1	-
qx	1	-
qy	1	-
XACC	3	3

Format

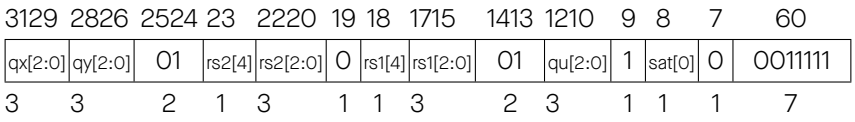


Figure 4.7-32. ESP.VMULAS.S8.XACC.ST.XP

4.7.2.33 ESP.VMULAS.U16.QACC

Syntax

ESP.VMULAS.U16.QACC qx,qy,sat

Description

This instruction divides registers qx and qy into 8 data segments of 16 bits each. Then, the unsigned multiplication results of the 8 sets of segments are added to the corresponding 64-bit data segments in special registers QACC_H and QACC_L respectively. The calculated results are saturated to 64-bit signed numbers and then stored in the corresponding 64-bit data segments in QACC_H and QACC_L.

Operation

```
QACC_L[63 :  0] = min(QACC_L[63 :  0] + qx[15 :  0] * qy[15 :  0], 2^63-1)
QACC_L[127: 64] = min(QACC_L[127: 64] + qx[31 : 16] * qy[31 : 16], 2^63-1)
QACC_L[191:128] = min(QACC_L[191:128] + qx[47 : 32] * qy[47 : 32], 2^63-1)
QACC_L[255:192] = min(QACC_L[255:192] + qx[63 : 48] * qy[63 : 48], 2^63-1)
QACC_H[63 :  0] = min(QACC_H[63 :  0] + qx[79 : 64] * qy[79 : 64], 2^63-1)
QACC_H[127: 64] = min(QACC_H[127: 64] + qx[95 : 80] * qy[95 : 80], 2^63-1)
QACC_H[191:128] = min(QACC_H[191:128] + qx[111: 96] * qy[111: 96], 2^63-1)
QACC_H[255:192] = min(QACC_H[255:192] + qx[127:112] * qy[127:112], 2^63-1)
```

Schedule

Operands	Use	Def
qx	1	-
qy	1	-
QACC_H	2	2
QACC_L	2	2

Format

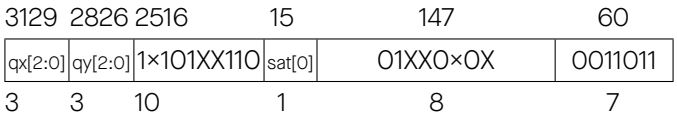


Figure 4.7-33. ESP.VMULAS.U16.QACC

4.7.2.34 ESP.VMULAS.U16.QACC.LD.IP

Syntax

ESP.VMULAS.U16.QACC.LD.IP qu,rs1,imm16,qx,qy,sat

Description

This instruction divides registers `qx` and `qy` into 8 data segments of 16 bits each. Then, the unsigned multiplication results of the 8 sets of segments are added to the corresponding 64-bit data segments in special registers `QACC_H` and `QACC_L` respectively. The calculated result is saturated to a 64-bit signed number and then stored in the corresponding 64-bit data segment in `QACC_H` and `QACC_L`.

During the operation, the 16-byte data is loaded from the memory into the register `qu`. After the access is completed, the value in the register `rs1` is incremented by the 4-bit sign-extended constant in the instruction code segment, which is left-shifted by 4.

The range of imm starts at -128, ends at 112, and steps by 16.

Operation

```
QACC_L[63 :  0] = min(QACC_L[63 :  0] + qx[15 :  0] * qy[15 :  0], 2^63-1)
QACC_L[127: 64] = min(QACC_L[127: 64] + qx[31 : 16] * qy[31 : 16], 2^63-1)
QACC_L[191:128] = min(QACC_L[191:128] + qx[47 : 32] * qy[47 : 32], 2^63-1)
QACC_L[255:192] = min(QACC_L[255:192] + qx[63 : 48] * qy[63 : 48], 2^63-1)
QACC_H[63 :  0] = min(QACC_H[63 :  0] + qx[79 : 64] * qy[79 : 64], 2^63-1)
QACC_H[127: 64] = min(QACC_H[127: 64] + qx[95 : 80] * qy[95 : 80], 2^63-1)
QACC_H[191:128] = min(QACC_H[191:128] + qx[111: 96] * qy[111: 96], 2^63-1)
QACC_H[255:192] = min(QACC_H[255:192] + qx[127:112] * qy[127:112], 2^63-1)

qu[127:0] = load128(rs1[31:0])
rs1[31:0] = rs1[31:0] + imm
```

Schedule

Operands	Use	Def
qu	-	2
rs1	1	1
qx	1	-
qy	1	-
QACC_H	2	2
QACC_L	2	2

Format

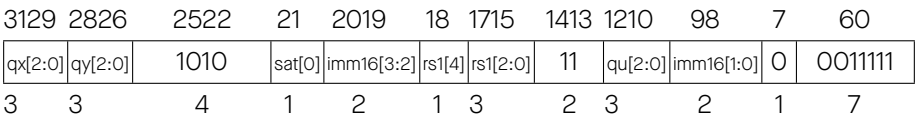


Figure 4.7-34. ESP.VMULAS.U16.QACC.LD.IP

4.7.2.35 ESP.VMULAS.U16.QACC.LD.XP

Syntax

ESP.VMULAS.U16.QACC.LD.XP qu,rs1,rs2,qx,qy,sat

Description

This instruction divides registers `qx` and `qy` into 8 data segments of 16 bits each. Then, it performs the unsigned multiplication of 8 sets of segments and adds the result to the corresponding 64-bit data segment in special registers `QACC_H` and `QACC_L` respectively. The calculated result is saturated to a 64-bit signed number and then stored in the corresponding 64-bit data segment in `QACC_H` and `QACC_L`.

During the operation, it loads the 16-byte data from the memory into the register `qu`. After the access is completed, the value in the register `rs1` is incremented by the value of the register `rs2`.

Operation

```
QACC_L[63 :  0] = min(QACC_L[63 :  0] + qx[15 :  0] * qy[15 :  0], 2^63-1)
QACC_L[127: 64] = min(QACC_L[127: 64] + qx[31 : 16] * qy[31 : 16], 2^63-1)
QACC_L[191:128] = min(QACC_L[191:128] + qx[47 : 32] * qy[47 : 32], 2^63-1)
QACC_L[255:192] = min(QACC_L[255:192] + qx[63 : 48] * qy[63 : 48], 2^63-1)
QACC_H[63 :  0] = min(QACC_H[63 :  0] + qx[79 : 64] * qy[79 : 64], 2^63-1)
QACC_H[127: 64] = min(QACC_H[127: 64] + qx[95 : 80] * qy[95 : 80], 2^63-1)
QACC_H[191:128] = min(QACC_H[191:128] + qx[111: 96] * qy[111: 96], 2^63-1)
QACC_H[255:192] = min(QACC_H[255:192] + qx[127:112] * qy[127:112], 2^63-1)
qu[127:0] = load128(rs1[31:0])
rs1[31:0] = rs1[31:0] + rs2[31:0]
```

Schedule

Operands	Use	Def
qu	-	2
rs1	1	1
rs2	1	-
qx	1	-
qy	1	-
QACC_H	2	2
QACC_L	2	2

Format

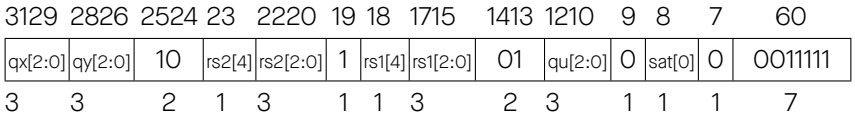


Figure 4.7-35. ESP.VMULAS.U16.QACC.LD.XP

4.7.2.36 ESP.VMULAS.U16.QACC.ST.IP

Syntax

ESP.VMULAS.U16.QACC.ST.IP qu,rs1,imm16,qx,qy,sat

Description

This instruction divides registers `qx` and `qy` into 8 data segments of 16 bits each. Then, the unsigned multiplication results of 8 sets of segments are added to the corresponding 64-bit data segments in special registers `QACC_H` and `QACC_L` respectively. The calculated result is saturated to a 64-bit signed number and then stored in the corresponding 64-bit data segments in `QACC_H` and `QACC_L`.

During the operation, the register `qu` is stored to memory. After the access is completed, the value in the register `rs1` is incremented by a 4-bit sign-extended constant in the instruction code segment, left-shifted by 4.

imm starts at -128, ends at 112, with a step of 16.

Operation

```
QACC_L[63 :  0] = min(QACC_L[63 :  0] + qx[15 :  0] * qy[15 :  0], 2^63-1)
QACC_L[127: 64] = min(QACC_L[127: 64] + qx[31 : 16] * qy[31 : 16], 2^63-1)
QACC_L[191:128] = min(QACC_L[191:128] + qx[47 : 32] * qy[47 : 32], 2^63-1)
QACC_L[255:192] = min(QACC_L[255:192] + qx[63 : 48] * qy[63 : 48], 2^63-1)
QACC_H[63 :  0] = min(QACC_H[63 :  0] + qx[79 : 64] * qy[79 : 64], 2^63-1)
QACC_H[127: 64] = min(QACC_H[127: 64] + qx[95 : 80] * qy[95 : 80], 2^63-1)
QACC_H[191:128] = min(QACC_H[191:128] + qx[111: 96] * qy[111: 96], 2^63-1)
QACC_H[255:192] = min(QACC_H[255:192] + qx[127:112] * qy[127:112], 2^63-1)

qu[127:0] => store128(rs1[31:0])
rs1[31:0] = rs1[31:0] + imm
```

Schedule

Operands	Use	Def
qu	2	-
rs1	1	1
qx	1	-
qy	1	-
QACC_H	2	2
QACC_L	2	2

Format

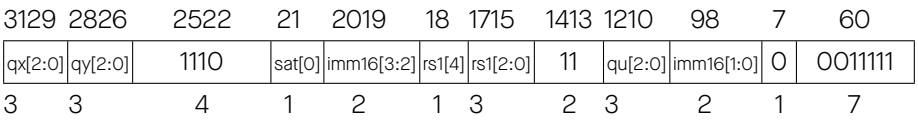


Figure 4.7-36. ESP.VMULAS.U16.QACC.ST.IP

4.7.2.37 ESP.VMULAS.U16.QACC.ST.XP

Syntax

ESP.VMULAS.U16.QACC.ST.XP qu,rs1,rs2,qx,qy,sat

Description

This instruction divides registers `qx` and `qy` into 8 data segments of 16 bits each. Then, the unsigned multiplication results of the 8 sets of segments are added to the corresponding 64-bit data segments in special registers `QACC_H` and `QACC_L` respectively. The calculated result is saturated to a 64-bit signed number and then stored in the corresponding 64-bit data segment in `QACC_H` and `QACC_L`.

During the operation, the register `qu` is stored to memory. After the access is completed, the value in the register `rs1` is incremented by the value of the register `rs2`.

Operation

```
QACC_L[63 :  0] = min(QACC_L[63 :  0] + qx[15 :  0] * qy[15 :  0], 2^63-1)
QACC_L[127: 64] = min(QACC_L[127: 64] + qx[31 : 16] * qy[31 : 16], 2^63-1)
QACC_L[191:128] = min(QACC_L[191:128] + qx[47 : 32] * qy[47 : 32], 2^63-1)
QACC_L[255:192] = min(QACC_L[255:192] + qx[63 : 48] * qy[63 : 48], 2^63-1)
QACC_H[63 :  0] = min(QACC_H[63 :  0] + qx[79 : 64] * qy[79 : 64], 2^63-1)
QACC_H[127: 64] = min(QACC_H[127: 64] + qx[95 : 80] * qy[95 : 80], 2^63-1)
QACC_H[191:128] = min(QACC_H[191:128] + qx[111: 96] * qy[111: 96], 2^63-1)
QACC_H[255:192] = min(QACC_H[255:192] + qx[127:112] * qy[127:112], 2^63-1)
qu[127:0] => store128(rs1[31:0])
rs1[31:0] = rs1[31:0] + rs2[31:0]
```

Schedule

Operands	Use	Def
qu	2	-
rs1	1	1
rs2	1	-
qx	1	-
qy	1	-
QACC_H	2	2
QACC_L	2	2

Format

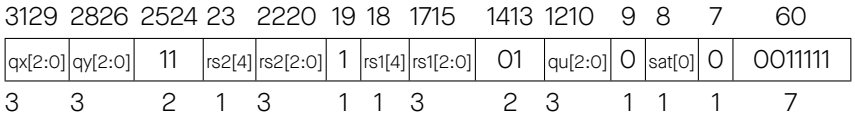


Figure 4.7-37. ESP.VMULAS.U16.QACC.ST.XP

4.7.2.38 ESP.VMULAS.U16.XACC

Syntax

ESP.VMULAS.U16.XACC qx,qy,sat

Description

This instruction divides registers qx and qy into 8 data segments of 16 bits each. Then, it performs an unsigned multiply-accumulate operation on each of the 8 sets of segments. The accumulated result is added to the value in the special register XACC. The resulting sum is then saturated and stored in XACC.

Operation

```
add0[31:0] = qx[ 15:  0] * qy[ 15:  0]
add1[31:0] = qx[ 31: 16] * qy[ 31: 16]
...
add7[31:0] = qx[127:112] * qy[127:112]
sum[40:0] = XACC[39:0] + add0[31:0] + add1[31:0] + ... + add7[31:0]

XACC[39:0] = min(max(sum[40:0], 0), 2^{40}-1)
```

Schedule

Operands	Use	Def
qx	1	-
qy	1	-
XACC	3	3

Format

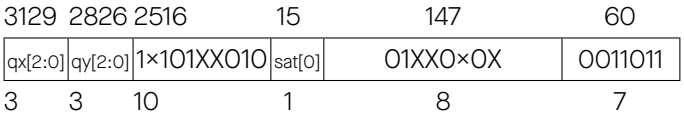


Figure 4.7-38. ESP.VMULAS.U16.XACC

4.7.2.39 ESP.VMULAS.U16.XACC.LD.IP

Syntax

ESP.VMULAS.U16.XACC.LD.IP qu,rs1,imm16,qx,qy,sat

Description

This instruction divides registers `qx` and `qy` into 8 data segments of 16 bits each. Then, it performs an unsigned multiply-accumulate operation on each of the 8 segment sets. The accumulated result is added to the value in the special register `XACC`. The resulting sum is then saturated and stored in `XACC`.
During the operation, the 16-byte data is loaded from the memory into the register `qu`. After the access is completed, the value in the register `rs1` is incremented by a 4-bit sign-extended constant from the instruction code segment, which is left-shifted by 4.
imm starts at -128, ends at 112, and steps by 16.

Operation

```
add0[31:0] = qx[ 15:  0] * qy[ 15:  0]
add1[31:0] = qx[ 31: 16] * qy[ 31: 16]
...
add7[31:0] = qx[127:112] * qy[127:112]
sum[40:0] = XACC[39:0] + add0[31:0] + add1[31:0] + ... + add7[31:0]
XACC[40:0] = min(max(sum[40:0], 0), 2^{40}-1)

qu[127:0] = load128(rs1[31:0])
rs1[31:0] = rs1[31:0] + imm
```

Schedule

Operands	Use	Def
qu	-	2
rs1	1	1
qx	1	-
qy	1	-
XACC	3	3

Format

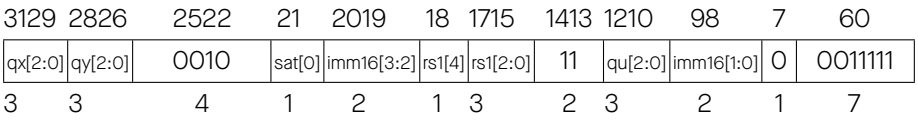


Figure 4.7-39. ESP.VMULAS.U16.XACC.LD.IP

4.7.2.40 ESP.VMULAS.U16.XACC.LD.XP

Syntax

ESP.VMULAS.U16.XACC.LD.XP qu,rs1,rs2,qx,qy,sat

Description

This instruction divides registers `qx` and `qy` into 8 data segments of 16 bits each. Then, it performs an unsigned multiply-accumulate operation on each of the 8 segment sets. The accumulated result is added to the value in the special register `XACC`. The resulting sum is then saturated and stored in `XACC`.
During the operation, it loads the 16-byte data from the memory into the register `qu`. After the access is completed, the value in the register `rs1` is incremented by the value of the register `rs2`.

Operation

```
add0[31:0] = qx[ 15:  0] * qy[ 15:  0]
add1[31:0] = qx[ 31: 16] * qy[ 31: 16]
...
add7[31:0] = qx[127:112] * qy[127:112]
sum[40:0] = XACC[39:0] + add0[31:0] + add1[31:0] + ... + add7[31:0]
XACC[40:0] = min(max(sum[40:0], 0), 2^{40}-1)

qu[127:0] = load128(rs1[31:0])
rs1[31:0] = rs1[31:0] + rs2[31:0]
```

Schedule

Operands	Use	Def
qu	-	2
rs1	1	1
rs2	1	-
qx	1	-
qy	1	-
XACC	3	3

Format

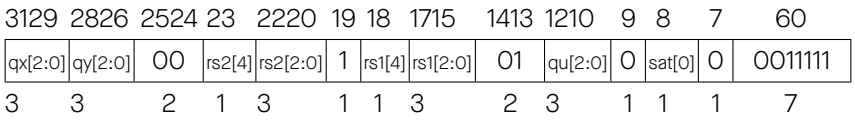


Figure 4.7-40. ESP.VMULAS.U16.XACC.LD.XP

4.7.2.41 ESP.VMULAS.U16.XACC.ST.IP

Syntax

ESP.VMULAS.U16.XACC.ST.IP qu,rs1,imm16,qx,qy,sat

Description

This instruction divides registers `qx` and `qy` into 8 data segments of 16 bits each. Then, it performs an unsigned multiply-accumulate operation on the 8 sets of segments respectively. The accumulated result is added to the value in the special register `XACC`. Then, the sum obtained is saturated and stored in `XACC`.

During the operation, the register `qu` is stored to memory. After the access is completed, the value in the register `rs1` is incremented by a 4-bit sign-extended constant in the instruction code segment, which is left-shifted by 4.

imm starts at -128, ends at 112, and steps by 16.

Operation

```
add0[31:0] = qx[ 15:  0] * qy[ 15:  0]
add1[31:0] = qx[ 31: 16] * qy[ 31: 16]
...
add7[31:0] = qx[127:112] * qy[127:112]
sum[40:0] = XACC[39:0] + add0[31:0] + add1[31:0] + ... + add7[31:0]
XACC[40:0] = min(max(sum[40:0], 0), 2^{40}-1)

qu[127:0] => store128(rs1[31:0])
rs1[31:0] = rs1[31:0] + imm
```

Schedule

Operands	Use	Def
qu	2	-
rs1	1	1
qx	1	-
qy	1	-
XACC	3	3

Format

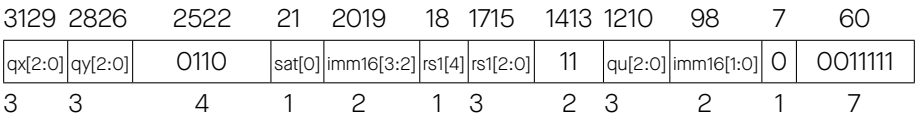


Figure 4.7-41. ESP.VMULAS.U16.XACC.ST.IP

4.7.2.42 ESP.VMULAS.U16.XACC.ST.XP

Syntax

ESP.VMULAS.U16.XACC.ST.XP qu,rs1,rs2,qx,qy,sat

Description

This instruction divides registers `qx` and `qy` into 8 data segments of 16 bits each. Then, it performs an unsigned multiply-accumulate operation on each of the 8 segment sets. The accumulated result is added to the value in the special register `XACC`. The resulting sum is then saturated and stored in `XACC`.
During the operation, the register `qu` is stored to memory. After the access is completed, the value in the register `rs1` is incremented by the value of the register `rs2`.

Operation

```
add0[31:0] = qx[ 15:  0] * qy[ 15:  0]
add1[31:0] = qx[ 31: 16] * qy[ 31: 16]
...
add7[31:0] = qx[127:112] * qy[127:112]
sum[40:0] = XACC[39:0] + add0[31:0] + add1[31:0] + ... + add7[31:0]
XACC[40:0] = min(max(sum[40:0], 0), 2^{40}-1)

qu[127:0] => store128(rs1[31:0])
rs1[31:0] = rs1[31:0] + rs2[31:0]
```

Schedule

Operands	Use	Def
qu	2	-
rs1	1	1
rs2	1	-
qx	1	-
qy	1	-
XACC	3	3

Format

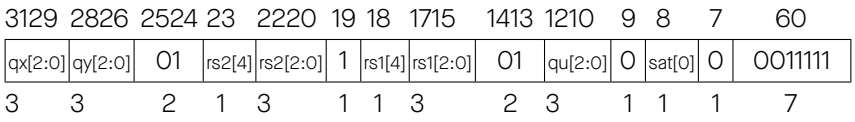


Figure 4.7-42. ESP.VMULAS.U16.XACC.ST.XP

4.7.2.43 ESP.VMULAS.U8.QACC

Syntax

ESP.VMULAS.U8.QACC qx,qy,sat

Description

This instruction divides registers qx and qy into 16 data segments of 8 bits each. Then, it performs the unsigned multiplication of the 16 sets of segments and adds the result to the corresponding 32-bit data segment in special registers QACC_H and QACC_L respectively. The calculated result is saturated to a 32-bit unsigned number and then stored in the corresponding 32-bit data segment in QACC_H and QACC_L.

Operation

```
QACC_L[31 :  0] = min(QACC_L[31 :  0] + qx[7  :  0] * qy[7  :  0], 2^31-1)
QACC_L[63 : 32] = min(QACC_L[63 : 32] + qx[15 :  8] * qy[15 :  8], 2^31-1)
QACC_L[95 : 64] = min(QACC_L[95 : 64] + qx[23 : 16] * qy[23 : 16], 2^31-1)
QACC_L[127: 96] = min(QACC_L[127: 96] + qx[31 : 24] * qy[31 : 24], 2^31-1)
QACC_L[159:128] = min(QACC_L[159:128] + qx[39 : 32] * qy[39 : 32], 2^31-1)
QACC_L[191:160] = min(QACC_L[191:160] + qx[47 : 40] * qy[47 : 40], 2^31-1)
QACC_L[223:192] = min(QACC_L[223:192] + qx[55 : 48] * qy[55 : 48], 2^31-1)
QACC_L[255:224] = min(QACC_L[255:224] + qx[63 : 56] * qy[63 : 56], 2^31-1)
QACC_H[31 :  0] = min(QACC_H[31 :  0] + qx[71 : 64] * qy[71 : 64], 2^31-1)
QACC_H[63 : 32] = min(QACC_H[63 : 32] + qx[79 : 72] * qy[79 : 72], 2^31-1)
QACC_H[95 : 64] = min(QACC_H[95 : 64] + qx[87 : 80] * qy[87 : 80], 2^31-1)
QACC_H[127: 96] = min(QACC_H[127: 96] + qx[95 : 88] * qy[95 : 88], 2^31-1)
QACC_H[159:128] = min(QACC_H[159:128] + qx[103: 96] * qy[103: 96], 2^31-1)
QACC_H[191:160] = min(QACC_H[191:160] + qx[111:104] * qy[111:104], 2^31-1)
QACC_H[223:192] = min(QACC_H[223:192] + qx[119:112] * qy[119:112], 2^31-1)
QACC_H[255:224] = min(QACC_H[255:224] + qx[127:120] * qy[127:120], 2^31-1)
```

Schedule

Operands	Use	Def
qx	1	-
qy	1	-
QACC_H	2	2
QACC_L	2	2

Format

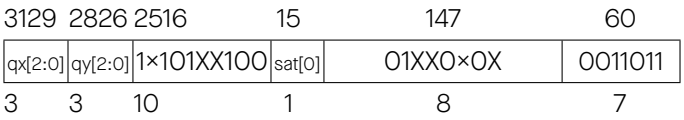


Figure 4.7-43. ESP.VMULAS.U8.QACC

4.7.2.44 ESP.VMULAS.U8.QACC.LD.IP

Syntax

ESP.VMULAS.U8.QACC.LD.IP qu,rs1,imm16,qx,qy,sat

Description

This instruction divides registers `qx` and `qy` into 16 data segments of 8 bits each. Then, the unsigned multiplication results of the 16 sets of segments are added to the corresponding 32-bit data segments in special registers `QACC_H` and `QACC_L` respectively. The calculated result is saturated to a 32-bit unsigned number and then stored in the corresponding 32-bit data segment in `QACC_H` and `QACC_L`.
During the operation, the 16-byte data is loaded from the memory to the register `qu`. After the access is completed, the value in the register `rs1` is incremented by the 4-bit sign-extended constant in the instruction code segment, which is left-shifted by 4.

The value of imm starts at -128, ends at 112, and steps by 16.

Operation

```
QACC_L[31 :  0] = min(QACC_L[31 :  0] + qx[7  :  0] * qy[7  :  0], 2^31-1)
QACC_L[63 : 32] = min(QACC_L[63 : 32] + qx[15 :  8] * qy[15 :  8], 2^31-1)
QACC_L[95 : 64] = min(QACC_L[95 : 64] + qx[23 : 16] * qy[23 : 16], 2^31-1)
QACC_L[127: 96] = min(QACC_L[127: 96] + qx[31 : 24] * qy[31 : 24], 2^31-1)
QACC_L[159:128] = min(QACC_L[159:128] + qx[39 : 32] * qy[39 : 32], 2^31-1)
QACC_L[191:160] = min(QACC_L[191:160] + qx[47 : 40] * qy[47 : 40], 2^31-1)
QACC_L[223:192] = min(QACC_L[223:192] + qx[55 : 48] * qy[55 : 48], 2^31-1)
QACC_L[255:224] = min(QACC_L[255:224] + qx[63 : 56] * qy[63 : 56], 2^31-1)
QACC_H[31 :  0] = min(QACC_H[31 :  0] + qx[71 : 64] * qy[71 : 64], 2^31-1)
QACC_H[63 : 32] = min(QACC_H[63 : 32] + qx[79 : 72] * qy[79 : 72], 2^31-1)
QACC_H[95 : 64] = min(QACC_H[95 : 64] + qx[87 : 80] * qy[87 : 80], 2^31-1)
QACC_H[127: 96] = min(QACC_H[127: 96] + qx[95 : 88] * qy[95 : 88], 2^31-1)
QACC_H[159:128] = min(QACC_H[159:128] + qx[103: 96] * qy[103: 96], 2^31-1)
QACC_H[191:160] = min(QACC_H[191:160] + qx[111:104] * qy[111:104], 2^31-1)
QACC_H[223:192] = min(QACC_H[223:192] + qx[119:112] * qy[119:112], 2^31-1)
QACC_H[255:224] = min(QACC_H[255:224] + qx[127:120] * qy[127:120], 2^31-1)

qu[127:0] = load128(rs1[31:0])
rs1[31:0] = rs1[31:0] + imm
```

Schedule

Operands	Use	Def
qu	-	2
rs1	1	1
qx	1	-
qy	1	-
QACC_H	2	2
QACC_L	2	2

Format

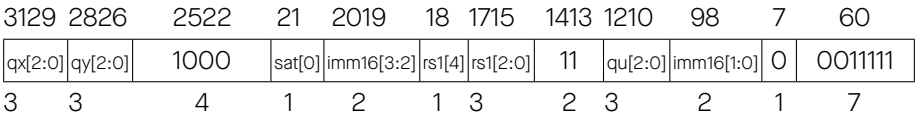


Figure 4.7-44. ESP.VMULAS.U8.QACC.LD.IP

4.7.2.45 ESP.VMULAS.U8.QACC.LD.XP

Syntax

ESP.VMULAS.U8.QACC.LD.XP qu,rs1,rs2,qx,qy,sat

Description

This instruction divides registers `qx` and `qy` into 16 data segments of 8 bits each. Then, the unsigned multiplication results of the 16 sets of segments are added to the corresponding 32-bit data segments in special registers `QACC_H` and `QACC_L` respectively. The calculated result is saturated to a 32-bit unsigned number and then stored in the corresponding 32-bit data segment in `QACC_H` and `QACC_L`.
During the operation, the 16-byte data is loaded from the memory to the register `qu`. After the access is completed, the value in the register `rs1` is incremented by the value of the register `rs2`.

Operation

```
QACC_L[31 :  0] = min(QACC_L[31 :  0] + qx[7  :  0] * qy[7  :  0], 2^31-1)
QACC_L[63 : 32] = min(QACC_L[63 : 32] + qx[15 :  8] * qy[15 :  8], 2^31-1)
QACC_L[95 : 64] = min(QACC_L[95 : 64] + qx[23 : 16] * qy[23 : 16], 2^31-1)
QACC_L[127: 96] = min(QACC_L[127: 96] + qx[31 : 24] * qy[31 : 24], 2^31-1)
QACC_L[159:128] = min(QACC_L[159:128] + qx[39 : 32] * qy[39 : 32], 2^31-1)
QACC_L[191:160] = min(QACC_L[191:160] + qx[47 : 40] * qy[47 : 40], 2^31-1)
QACC_L[223:192] = min(QACC_L[223:192] + qx[55 : 48] * qy[55 : 48], 2^31-1)
QACC_L[255:224] = min(QACC_L[255:224] + qx[63 : 56] * qy[63 : 56], 2^31-1)
QACC_H[31 :  0] = min(QACC_H[31 :  0] + qx[71 : 64] * qy[71 : 64], 2^31-1)
QACC_H[63 : 32] = min(QACC_H[63 : 32] + qx[79 : 72] * qy[79 : 72], 2^31-1)
QACC_H[95 : 64] = min(QACC_H[95 : 64] + qx[87 : 80] * qy[87 : 80], 2^31-1)
QACC_H[127: 96] = min(QACC_H[127: 96] + qx[95 : 88] * qy[95 : 88], 2^31-1)
QACC_H[159:128] = min(QACC_H[159:128] + qx[103: 96] * qy[103: 96], 2^31-1)
QACC_H[191:160] = min(QACC_H[191:160] + qx[111:104] * qy[111:104], 2^31-1)
QACC_H[223:192] = min(QACC_H[223:192] + qx[119:112] * qy[119:112], 2^31-1)
QACC_H[255:224] = min(QACC_H[255:224] + qx[127:120] * qy[127:120], 2^31-1)

qu[127:0] = load128(rs1[31:0])
rs1[31:0] = rs1[31:0] + rs2[31:0]
```

Schedule

Operands	Use	Def
qu	-	2
rs1	1	1
rs2	1	-
qx	1	-
qy	1	-
QACC_H	2	2
QACC_L	2	2

Format

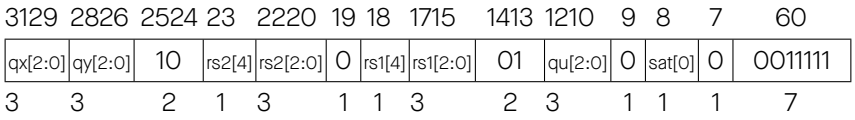


Figure 4.7-45. ESP.VMULAS.U8.QACC.LD.XP

4.7.2.46 ESP.VMULAS.U8.QACC.ST.IP

Syntax

ESP.VMULAS.U8.QACC.ST.IP qu,rs1,imm16,qx,qy,sat

Description

This instruction divides registers `qx` and `qy` into 16 data segments of 8 bits each. Then, the unsigned multiplication results of the 16 sets of segments are added to the corresponding 32-bit data segments in special registers `QACC_H` and `QACC_L` respectively. The calculated result is saturated to a 32-bit unsigned number and then stored in the corresponding 32-bit data segment in `QACC_H` and `QACC_L`.
During the operation, the register `qu` is stored to memory. After the access is completed, the value in the register `rs1` is incremented by the 4-bit sign-extended constant in the instruction code segment, which is left-shifted by 4.

The imm starts at -128, ends at 112, and steps by 16.

Operation

```
QACC_L[31 :  0] = min(QACC_L[31 :  0] + qx[7  :  0] * qy[7  :  0], 2^31-1)
QACC_L[63 : 32] = min(QACC_L[63 : 32] + qx[15 :  8] * qy[15 :  8], 2^31-1)
QACC_L[95 : 64] = min(QACC_L[95 : 64] + qx[23 : 16] * qy[23 : 16], 2^31-1)
QACC_L[127: 96] = min(QACC_L[127: 96] + qx[31 : 24] * qy[31 : 24], 2^31-1)
QACC_L[159:128] = min(QACC_L[159:128] + qx[39 : 32] * qy[39 : 32], 2^31-1)
QACC_L[191:160] = min(QACC_L[191:160] + qx[47 : 40] * qy[47 : 40], 2^31-1)
QACC_L[223:192] = min(QACC_L[223:192] + qx[55 : 48] * qy[55 : 48], 2^31-1)
QACC_L[255:224] = min(QACC_L[255:224] + qx[63 : 56] * qy[63 : 56], 2^31-1)
QACC_H[31 :  0] = min(QACC_H[31 :  0] + qx[71 : 64] * qy[71 : 64], 2^31-1)
QACC_H[63 : 32] = min(QACC_H[63 : 32] + qx[79 : 72] * qy[79 : 72], 2^31-1)
QACC_H[95 : 64] = min(QACC_H[95 : 64] + qx[87 : 80] * qy[87 : 80], 2^31-1)
QACC_H[127: 96] = min(QACC_H[127: 96] + qx[95 : 88] * qy[95 : 88], 2^31-1)
QACC_H[159:128] = min(QACC_H[159:128] + qx[103: 96] * qy[103: 96], 2^31-1)
QACC_H[191:160] = min(QACC_H[191:160] + qx[111:104] * qy[111:104], 2^31-1)
QACC_H[223:192] = min(QACC_H[223:192] + qx[119:112] * qy[119:112], 2^31-1)
QACC_H[255:224] = min(QACC_H[255:224] + qx[127:120] * qy[127:120], 2^31-1)

qu[127:0] => store128(rs1[31:0])
rs1[31:0] = rs1[31:0] + imm
```

Schedule

Operands	Use	Def
qu	2	-
rs1	1	1
qx	1	-
qy	1	-
QACC_H	2	2
QACC_L	2	2

Format

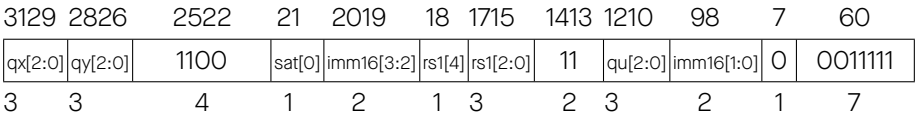


Figure 4.7-46. ESP.VMULAS.U8.QACC.ST.IP

4.7.2.47 ESP.VMULAS.U8.QACC.ST.XP

Syntax

ESP.VMULAS.U8.QACC.ST.XP qu,rs1,rs2,qx,qy,sat

Description

This instruction divides registers `qx` and `qy` into 16 data segments of 8 bits each. Then, the unsigned multiplication results of the 16 sets of segments are added to the corresponding 32-bit data segments in special registers `QACC_H` and `QACC_L` respectively. The calculated result is saturated to a 32-bit unsigned number and then stored in the corresponding 32-bit data segment in `QACC_H` and `QACC_L`.
During the operation, the register `qu` is stored to memory. After the access is completed, the value in the register `rs1` is incremented by the value of the register `rs2`.

Operation

```
QACC_L[31 :  0] = min(QACC_L[31 :  0] + qx[7  :  0] * qy[7  :  0], 2^31-1)
QACC_L[63 : 32] = min(QACC_L[63 : 32] + qx[15 :  8] * qy[15 :  8], 2^31-1)
QACC_L[95 : 64] = min(QACC_L[95 : 64] + qx[23 : 16] * qy[23 : 16], 2^31-1)
QACC_L[127: 96] = min(QACC_L[127: 96] + qx[31 : 24] * qy[31 : 24], 2^31-1)
QACC_L[159:128] = min(QACC_L[159:128] + qx[39 : 32] * qy[39 : 32], 2^31-1)
QACC_L[191:160] = min(QACC_L[191:160] + qx[47 : 40] * qy[47 : 40], 2^31-1)
QACC_L[223:192] = min(QACC_L[223:192] + qx[55 : 48] * qy[55 : 48], 2^31-1)
QACC_L[255:224] = min(QACC_L[255:224] + qx[63 : 56] * qy[63 : 56], 2^31-1)
QACC_H[31 :  0] = min(QACC_H[31 :  0] + qx[71 : 64] * qy[71 : 64], 2^31-1)
QACC_H[63 : 32] = min(QACC_H[63 : 32] + qx[79 : 72] * qy[79 : 72], 2^31-1)
QACC_H[95 : 64] = min(QACC_H[95 : 64] + qx[87 : 80] * qy[87 : 80], 2^31-1)
QACC_H[127: 96] = min(QACC_H[127: 96] + qx[95 : 88] * qy[95 : 88], 2^31-1)
QACC_H[159:128] = min(QACC_H[159:128] + qx[103: 96] * qy[103: 96], 2^31-1)
QACC_H[191:160] = min(QACC_H[191:160] + qx[111:104] * qy[111:104], 2^31-1)
QACC_H[223:192] = min(QACC_H[223:192] + qx[119:112] * qy[119:112], 2^31-1)
QACC_H[255:224] = min(QACC_H[255:224] + qx[127:120] * qy[127:120], 2^31-1)

qu[127:0] => store128(rs1[31:0])
rs1[31:0] = rs1[31:0] + rs2[31:0]
```

Schedule

Operands	Use	Def
qu	2	-
rs1	1	1
rs2	1	-
qx	1	-
qy	1	-
QACC_H	2	2
QACC_L	2	2

Format

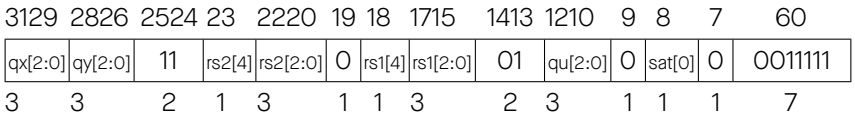


Figure 4.7-47. ESP.VMULAS.U8.QACC.ST.XP

4.7.2.48 ESP.VMULAS.U8.XACC

Syntax

ESP.VMULAS.U8.XACC qx,qy,sat

Description

This instruction divides registers qx and qy into 16 data segments of 8 bits each. Then, it performs an unsigned multiply-accumulate operation on each of the 16 sets of segments. The accumulated result is added to the value in the special register XACC. The resulting sum is then saturated and stored in XACC.

Operation

```
add0[15:0] = qx[  7:  0] * qy[  7:  0]
add1[15:0] = qx[ 15:  8] * qy[ 15:  8]
...
add15[15:0] = qx[127:120] * qy[127:120]
sum[40:0] = XACC[39:0] + add0[15:0] + add1[15:0] + ... + add15[15:0]

XACC[40:0] = min(max(sum[40:0], 0), 2^{40}-1)
```

Schedule

Operands	Use	Def
qx	1	-
qy	1	-
XACC	3	3

Format

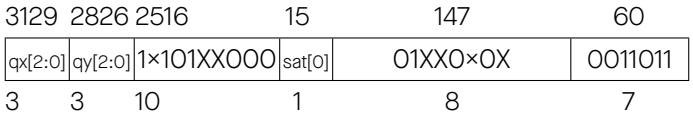


Figure 4.7-48. ESP.VMULAS.U8.XACC

4.7.2.49 ESP.VMULAS.U8.XACC.LD.IP

Syntax

ESP.VMULAS.U8.XACC.LD.IP qu,rs1,imm16,qx,qy,sat

Description

This instruction divides registers `qx` and `qy` into 16 data segments of 8 bits each. Then, it performs an unsigned multiply-accumulate operation on each of the 16 segment sets. The accumulated result is added to the value in the special register `XACC`. The resulting sum is then saturated and stored in `XACC`.
During the operation, the 16-byte data is loaded from the memory into the register `qu`. After the access is completed, the value in the register `rs1` is incremented by the 4-bit sign-extended constant in the instruction code segment, which is left-shifted by 4.
imm starts at -128, ends at 112, and steps by 16.

Operation

```
add0[15:0] = qx[  7:  0] * qy[  7:  0]
add1[15:0] = qx[ 15:  8] * qy[ 15:  8]
...
add15[15:0] = qx[127:120] * qy[127:120]
sum[40:0] = XACC[39:0] + add0[15:0] + add1[15:0] + ... + add15[15:0]
XACC[39:0] = min(max(sum[40:0], 0), 2^{40}-1)

qu[127:0] = load128(rs1[31:0])
rs1[31:0] = rs1[31:0] + imm
```

Schedule

Operands	Use	Def
qu	-	2
rs1	1	1
qx	1	-
qy	1	-
XACC	3	3

Format

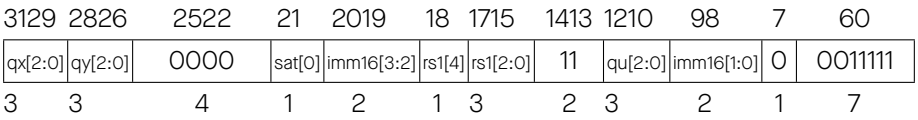


Figure 4.7-49. ESP.VMULAS.U8.XACC.LD.IP

4.7.2.50 ESP.VMULAS.U8.XACC.LD.XP

Syntax

ESP.VMULAS.U8.XACC.LD.XP qu,rs1,rs2,qx,qy,sat

Description

This instruction divides registers `qx` and `qy` into 16 data segments of 8 bits each. Then, it performs an unsigned multiply-accumulate operation on each of the 16 sets of segments. The accumulated result is added to the value in the special register `XACC`. The resulting sum is then saturated and stored in `XACC`.
During the operation, the 16-byte data is loaded from the memory into the register `qu`. After the access is completed, the value in the register `rs1` is incremented by the value in the register `rs2`.

Operation

```
add0[15:0] = qx[  7:  0] * qy[  7:  0]
add1[15:0] = qx[ 15:  8] * qy[ 15:  8]
...
add15[15:0] = qx[127:120] * qy[127:120]
sum[40:0] = XACC[39:0] + add0[15:0] + add1[15:0] + ... + add15[15:0]
XACC[39:0] = min(max(sum[40:0], 0), 2^{40}-1)

qu[127:0] = load128(rs1[31:0])
rs1[31:0] = rs1[31:0] + rs2[31:0]
```

Schedule

Operands	Use	Def
qu	-	2
rs1	1	1
rs2	1	-
qx	1	-
qy	1	-
XACC	3	3

Format

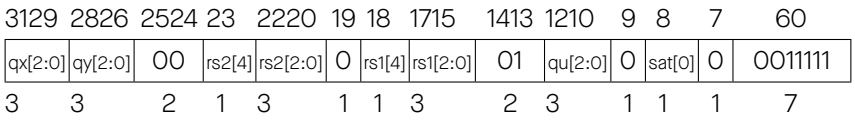


Figure 4.7-50. ESP.VMULAS.U8.XACC.LD.XP

4.7.2.51 ESP.VMULAS.U8.XACC.ST.IP

Syntax

ESP.VMULAS.U8.XACC.ST.IP qu,rs1,imm16,qx,qy,sat

Description

This instruction divides registers `qx` and `qy` into 16 data segments of 8 bits each. Then, it performs an unsigned multiply-accumulate operation on each of the 16 segment sets. The accumulated result is added to the value in the special register `XACC`. The resulting sum is then saturated and stored in `XACC`.
During the operation, the value of the register `qu` is stored to memory. After the access is completed, the value in the register `rs1` is incremented by a 4-bit sign-extended constant from the instruction code segment, which is left-shifted by 4.
imm starts at -128, ends at 112, and steps by 16.

Operation

```
add0[15:0] = qx[  7:  0] * qy[  7:  0]
add1[15:0] = qx[ 15:  8] * qy[ 15:  8]
...
add15[15:0] = qx[127:120] * qy[127:120]
sum[40:0] = XACC[39:0] + add0[15:0] + add1[15:0] + ... + add15[15:0]
XACC[39:0] = min(max(sum[40:0], 0), 2^{40}-1)

qu[127:0] => store128(rs1[31:0])
rs1[31:0] = rs1[31:0] + imm
```

Schedule

Operands	Use	Def
qu	2	-
rs1	1	1
qx	1	-
qy	1	-
XACC	3	3

Format

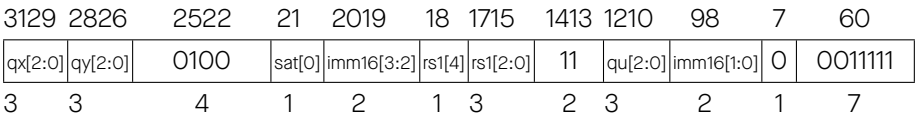


Figure 4.7-51. ESP.VMULAS.U8.XACC.ST.IP

4.7.2.52 ESP.VMULAS.U8.XACC.ST.XP

Syntax

ESP.VMULAS.U8.XACC.ST.XP qu,rs1,rs2,qx,qy,sat

Description

This instruction divides registers `qx` and `qy` into 16 data segments of 8 bits each. Then, it performs an unsigned multiply-accumulate operation on each of the 16 segments. The accumulated result is added to the value in the special register `XACC`. The resulting sum is then saturated and stored in `XACC`.

During the operation, the value of the register `qu` is stored to memory. After the access is completed, the value in the register `rs1` is incremented by the value of the register `rs2`.

Operation

```
add0[15:0] = qx[  7:  0] * qy[  7:  0]
add1[15:0] = qx[ 15:  8] * qy[ 15:  8]
...
add15[15:0] = qx[127:120] * qy[127:120]
sum[40:0] = XACC[39:0] + add0[15:0] + add1[15:0] + ... + add15[15:0]
XACC[39:0] = min(max(sum[40:0], 0), 2^{40}-1)

qu[127:0] => store128(rs1[31:0])
rs1[31:0] = rs1[31:0] + rs2[31:0]
```

Schedule

Operands	Use	Def
qu	2	-
rs1	1	1
rs2	1	-
qx	1	-
qy	1	-
XACC	3	3

Format

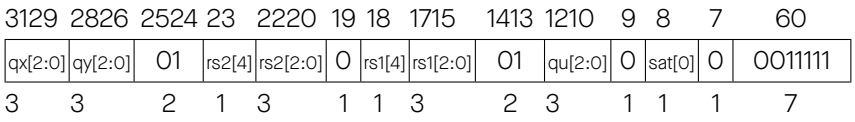


Figure 4.7-52. ESP.VMULAS.U8.XACC.ST.XP

4.7.2.53 ESP.VMULAS.S16.QACC.LDBC.INCP

Syntax

ESP.VMULAS.S16.QACC.LDBC.INCP qu,rs1,qx,qy,sat

Description

This instruction divides registers `qx` and `qy` into 8 data segments of 16 bits each. Then, the signed multiplication results of the 8 sets of segments are added to the corresponding 64-bit data segments in special registers `QACC_H` and `QACC_L` respectively. The calculated result is saturated to a 64-bit signed number and then stored in the corresponding 64-bit data segment in `QACC_H` and `QACC_L`.

During the operation, the 2-byte data is loaded from the memory and broadcast to the register `qu`. After the access is completed, the value in the register `rs1` is incremented by 2.

Operation

```
QACC_L[ 63:  0] = min(max(QACC_L[ 63:  0] + qx[ 15:  0] * qy[ 15:  0], -2^{63}),
    2^{63}-1)
QACC_L[127: 64] = min(max(QACC_L[127: 64] + qx[ 31: 16] * qy[ 31: 16], -2^{63}),
    2^{63}-1)
QACC_L[191:128] = min(max(QACC_L[191:128] + qx[ 47: 32] * qy[ 47: 32], -2^{63}),
    2^{63}-1)
QACC_L[255:192] = min(max(QACC_L[255:192] + qx[ 63: 48] * qy[ 63: 48], -2^{63}),
    2^{63}-1)
QACC_H[ 63:  0] = min(max(QACC_H[ 63:  0] + qx[ 79: 64] * qy[ 79: 64], -2^{63}),
    2^{63}-1)
QACC_H[127: 64] = min(max(QACC_H[127: 64] + qx[ 95: 80] * qy[ 95: 80], -2^{63}),
    2^{63}-1)
QACC_H[191:128] = min(max(QACC_H[191:128] + qx[111: 96] * qy[111: 96], -2^{63}),
    2^{63}-1)
QACC_H[255:192] = min(max(QACC_H[255:192] + qx[127:112] * qy[127:112], -2^{63}),
    2^{63}-1)

qu[127:0] = {8{load16(rs1[31:0])}}
rs1[31:0] = rs1[31:0] + 2
```

Schedule

Operands	Use	Def
qu	-	2
rs1	1	1
qx	1	-
qy	1	-
QACC_H	2	2
QACC_L	2	2

Format

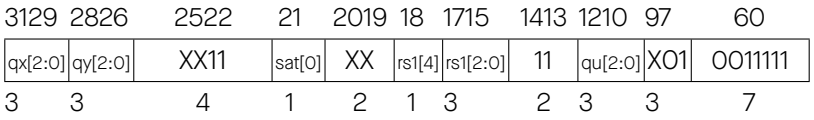


Figure 4.7-53. ESP.VMULAS.S16.QACC.LDBC.INCP

4.7.2.54 ESP.VMULAS.S8.QACC.LDBC.INCP

Syntax

ESP.VMULAS.S8.QACC.LDBC.INCP qu,rs1,qx,qy,sat

Description

This instruction divides registers `qx` and `qy` into 16 data segments of 8 bits each. Then, the signed multiplication results of the 16 sets of segments are added to the corresponding 32-bit data segments in special registers `QACC_H` and `QACC_L` respectively. The calculated result is saturated to a 32-bit signed number and then stored in the corresponding 32-bit data segment in `QACC_H` and `QACC_L`.

During the operation, the 1-byte data is loaded from the memory and broadcast to the register `qu`. After the access is completed, the value in the register `rs1` is incremented by 1.

Operation

```
QACC_L[ 31:  0] = min(max(QACC_L[ 31:  0] + qx[  7:  0] * qy[  7:  0], -2^{31}), 2^{31}-1)
QACC_L[ 63: 32] = min(max(QACC_L[ 63: 32] + qx[ 15:  8] * qy[ 15:  8], -2^{31}), 2^{31}-1)
QACC_L[ 95: 64] = min(max(QACC_L[ 95: 64] + qx[ 23: 16] * qy[ 23: 16], -2^{31}), 2^{31}-1)
QACC_L[127: 96] = min(max(QACC_L[127: 96] + qx[ 31: 24] * qy[ 31: 24], -2^{31}), 2^{31}-1)
QACC_L[159:128] = min(max(QACC_L[159:128] + qx[ 39: 32] * qy[ 39: 32], -2^{31}), 2^{31}-1)
QACC_L[191:160] = min(max(QACC_L[191:160] + qx[ 47: 40] * qy[ 47: 40], -2^{31}), 2^{31}-1)
QACC_L[223:192] = min(max(QACC_L[223:192] + qx[ 55: 48] * qy[ 55: 48], -2^{31}), 2^{31}-1)
QACC_L[255:224] = min(max(QACC_L[255:224] + qx[ 63: 56] * qy[ 63: 56], -2^{31}), 2^{31}-1)
QACC_H[ 31:  0] = min(max(QACC_H[ 31:  0] + qx[ 71: 64] * qy[ 71: 64], -2^{31}), 2^{31}-1)
QACC_H[ 63: 32] = min(max(QACC_H[ 63: 32] + qx[ 79: 72] * qy[ 79: 72], -2^{31}), 2^{31}-1)
QACC_H[ 95: 64] = min(max(QACC_H[ 95: 64] + qx[ 87: 80] * qy[ 87: 80], -2^{31}), 2^{31}-1)
QACC_H[127: 96] = min(max(QACC_H[127: 96] + qx[ 95: 88] * qy[ 95: 88], -2^{31}), 2^{31}-1)
QACC_H[159:128] = min(max(QACC_H[159:128] + qx[103: 96] * qy[103: 96], -2^{31}), 2^{31}-1)
QACC_H[191:160] = min(max(QACC_H[191:160] + qx[111:104] * qy[111:104], -2^{31}), 2^{31}-1)
QACC_H[223:192] = min(max(QACC_H[223:192] + qx[119:112] * qy[119:112], -2^{31}), 2^{31}-1)
QACC_H[255:224] = min(max(QACC_H[255:224] + qx[127:120] * qy[127:120], -2^{31}), 2^{31}-1)


qu[127:0] = {16{load8(rs1[31:0])}}
rs1[31:0] = rs1[31:0] + 1
```

Schedule

Operands	Use	Def
qu	-	2
rs1	1	1
qx	1	-
qy	1	-
QACC_H	2	2
QACC_L	2	2

Format

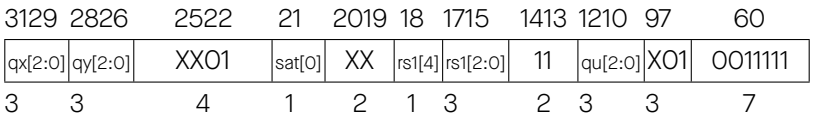


Figure 4.7-54. ESP.VMULAS.S8.QACC.LDBC.INCP

4.7.2.55 ESP.VMULAS.U16.QACC.LDBC.INCP

Syntax

ESP.VMULAS.U16.QACC.LDBC.INCP qu,rs1,qx,qy,sat

Description

This instruction divides registers `qx` and `qy` into 8 data segments of 16 bits each. Then, it performs the unsigned multiplication of 8 sets of segments and adds the result to the corresponding 64-bit data segment in special registers `QACC_H` and `QACC_L` respectively. The calculated result is saturated to a 64-bit unsigned number and then stored in the corresponding 64-bit data segment in `QACC_H` and `QACC_L`.

During the operation, it loads the 2-byte data from the memory and broadcasts it to the register `qu`. After the access is completed, the value in the register `rs1` is incremented by 2.

Operation

```
QACC_L[ 63:  0] = min(QACC_L[ 63:  0] + qx[ 15:  0] * qy[ 15:  0], 2^{64}-1)
QACC_L[127: 64] = min(QACC_L[127: 64] + qx[ 31: 16] * qy[ 31: 16], 2^{64}-1)
QACC_L[191:128] = min(QACC_L[191:128] + qx[ 47: 32] * qy[ 47: 32], 2^{64}-1)
QACC_L[255:192] = min(QACC_L[255:192] + qx[ 63: 48] * qy[ 63: 48], 2^{64}-1)
QACC_H[ 63:  0] = min(QACC_H[ 63:  0] + qx[ 79: 64] * qy[ 79: 64], 2^{64}-1)
QACC_H[127: 64] = min(QACC_H[127: 64] + qx[ 95: 80] * qy[ 95: 80], 2^{64}-1)
QACC_H[191:128] = min(QACC_H[191:128] + qx[111: 96] * qy[111: 96], 2^{64}-1)
QACC_H[255:192] = min(QACC_H[255:192] + qx[127:112] * qy[127:112], 2^{64}-1)

qu[127:0] = {8{load16(rs1[31:0])}}
rs1[31:0] = rs1[31:0] + 2
```

Schedule

Operands	Use	Def
qu	-	2
rs1	1	1
qx	1	-
qy	1	-
QACC_H	2	2
QACC_L	2	2

Format

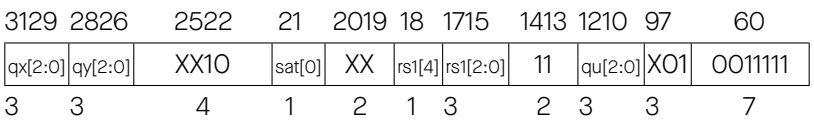


Figure 4.7-55. ESP.VMULAS.U16.QACC.LDBC.INCP

4.7.2.56 ESP.VMULAS.U8.QACC.LDBC.INCP

Syntax

ESP.VMULAS.U8.QACC.LDBC.INCP qu,rs1,qx,qy,sat

Description

This instruction divides registers `qx` and `qy` into 16 data segments of 8 bits each. Then, the unsigned multiplication results of the 16 sets of segments are added to the corresponding 32-bit data segments in special registers `QACC_H` and `QACC_L` respectively. The calculated result is saturated to a 32-bit unsigned number and then stored in the corresponding 32-bit data segment in `QACC_H` and `QACC_L`.

During the operation, the 1-byte data is loaded from the memory and broadcast to the register `qu`. After the access is completed, the value in the register `rs1` is incremented by 1.

Operation

```
QACC_L[ 31:  0] = min(QACC_L[ 31:  0] + qx[  7:  0] * qy[  7:  0], 2^{32}-1)
QACC_L[ 63: 32] = min(QACC_L[ 63: 32] + qx[ 15:  8] * qy[ 15:  8], 2^{32}-1)
QACC_L[ 95: 64] = min(QACC_L[ 95: 64] + qx[ 23: 16] * qy[ 23: 16], 2^{32}-1)
QACC_L[127: 96] = min(QACC_L[127: 96] + qx[ 31: 24] * qy[ 31: 24], 2^{32}-1)
QACC_L[159:128] = min(QACC_L[159:128] + qx[ 39: 32] * qy[ 39: 32], 2^{32}-1)
QACC_L[191:160] = min(QACC_L[191:160] + qx[ 47: 40] * qy[ 47: 40], 2^{32}-1)
QACC_L[223:192] = min(QACC_L[223:192] + qx[ 55: 48] * qy[ 55: 48], 2^{32}-1)
QACC_L[255:224] = min(QACC_L[255:224] + qx[ 63: 56] * qy[ 63: 56], 2^{32}-1)
QACC_H[ 31:  0] = min(QACC_H[ 31:  0] + qx[ 71: 64] * qy[ 71: 64], 2^{32}-1)
QACC_H[ 63: 32] = min(QACC_H[ 63: 32] + qx[ 79: 72] * qy[ 79: 72], 2^{32}-1)
QACC_H[ 95: 64] = min(QACC_H[ 95: 64] + qx[ 87: 80] * qy[ 87: 80], 2^{32}-1)
QACC_H[127: 96] = min(QACC_H[127: 96] + qx[ 95: 88] * qy[ 95: 88], 2^{32}-1)
QACC_H[159:128] = min(QACC_H[159:128] + qx[103: 96] * qy[103: 96], 2^{32}-1)
QACC_H[191:160] = min(QACC_H[191:160] + qx[111:104] * qy[111:104], 2^{32}-1)
QACC_H[223:192] = min(QACC_H[223:192] + qx[119:112] * qy[119:112], 2^{32}-1)
QACC_H[255:224] = min(QACC_H[255:224] + qx[127:120] * qy[127:120], 2^{32}-1)


qu[127:0] = {16{load8(rs1[31:0])}}
rs1[31:0] = rs1[31:0] + 1
```

Schedule

Operands	Use	Def
qu	-	2
rs1	1	1
qx	1	-
qy	1	-
QACC_H	2	2
QACC_L	2	2

Format

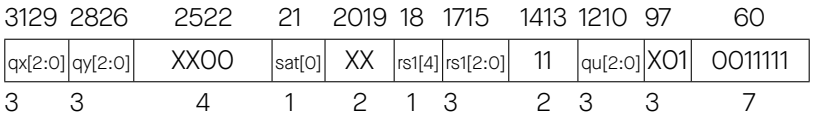


Figure 4.7-56. ESP.VMULAS.U8.QACC.LDBC.INCP

4.7.2.57 ESP.VSMULAS.S16.QACC

Syntax

ESP.VSMULAS.S16.QACC qx,qy,sel16,sat

Description

This instruction selects one out of the eight 16-bit data segments in the register `qy` according to the immediate number `sel16` and performs a signed multiplication on it and the eight 16-bit data segments in the register `qx` respectively. The eight results obtained are added to the corresponding 64-bit data segments in the special registers `QACC_H` and `QACC_L` respectively. Then, each result is saturated to a 64-bit signed number and then stored in the corresponding 64-bit data segment in `QACC_H` and `QACC_L`.

Operation

```
temp[15:0] = qy[sel16*16+15:sel16*16]
QACC_L[ 63:  0] = min(max(QACC_L[ 63:  0] + qx[ 15:  0] * temp[15:0], -2^{63}),
    2^{63}-1)
QACC_L[127: 64] = min(max(QACC_L[127: 64] + qx[ 31: 16] * temp[15:0], -2^{63}),
    2^{63}-1)
QACC_L[191:128] = min(max(QACC_L[191:128] + qx[ 47: 32] * temp[15:0], -2^{63}),
    2^{63}-1)
QACC_L[255:192] = min(max(QACC_L[255:192] + qx[ 63: 48] * temp[15:0], -2^{63}),
    2^{63}-1)
QACC_H[ 63:  0] = min(max(QACC_H[ 63:  0] + qx[ 79: 64] * temp[15:0], -2^{63}),
    2^{63}-1)
QACC_H[127: 64] = min(max(QACC_H[127: 64] + qx[ 95: 80] * temp[15:0], -2^{63}),
    2^{63}-1)
QACC_H[191:128] = min(max(QACC_H[191:128] + qx[111: 96] * temp[15:0], -2^{63}),
    2^{63}-1)
QACC_H[255:192] = min(max(QACC_H[255:192] + qx[127:112] * temp[15:0], -2^{63}),
    2^{63}-1)
```

Schedule

Operands	Use	Def
qx	1	-
qy	1	-
QACC_H	2	2
QACC_L	2	2

Format



Figure 4.7-57. ESP.VSMULAS.S16.QACC

4.7.2.58 ESP.VSMULAS.S16.QACC.LD.INCP

Syntax

ESP.VSMULAS.S16.QACC.LD.INCP qu,rs1,qx,qy,sel16,sat

Description

This instruction selects one out of the eight 16-bit data segments in the register `qy` according to the immediate number `sel16` and performs a signed multiplication on it and the eight 16-bit data segments in the register `qx` respectively. The eight results obtained are added to the corresponding 64-bit data segments in the special registers `QACC_H` and `QACC_L` respectively. Then, each result is saturated to a 64-bit signed number and then stored in the corresponding 64-bit data segment in `QACC_H` and `QACC_L`.
During the operation, the 16-byte data is loaded from the memory to the register `qu`. After the access is completed, the value in the register `rs1` is incremented by 16.

Operation

```
temp[15:0] = qy[sel16*16+15:sel16*16]
QACC_L[ 63:  0] = min(max(QACC_L[ 63:  0] + qx[ 15:  0] * temp[15:0], -2^{63}), 2^{63}-1)
QACC_L[127: 64] = min(max(QACC_L[127: 64] + qx[ 31: 16] * temp[15:0], -2^{63}), 2^{63}-1)
QACC_L[191:128] = min(max(QACC_L[191:128] + qx[ 47: 32] * temp[15:0], -2^{63}), 2^{63}-1)
QACC_L[255:192] = min(max(QACC_L[255:192] + qx[ 63: 48] * temp[15:0], -2^{63}), 2^{63}-1)
QACC_H[ 63:  0] = min(max(QACC_H[ 63:  0] + qx[ 79: 64] * temp[15:0], -2^{63}), 2^{63}-1)
QACC_H[127: 64] = min(max(QACC_H[127: 64] + qx[ 95: 80] * temp[15:0], -2^{63}), 2^{63}-1)
QACC_H[191:128] = min(max(QACC_H[191:128] + qx[111: 96] * temp[15:0], -2^{63}), 2^{63}-1)
QACC_H[255:192] = min(max(QACC_H[255:192] + qx[127:112] * temp[15:0], -2^{63}), 2^{63}-1)

qu[127:0] = load128(rs1[31:0])
rs1[31:0] = rs1[31:0] + 16
```

Schedule

Operands	Use	Def
qu	-	2
rs1	1	1
qx	1	-
qy	1	-
QACC_H	2	2
QACC_L	2	2

Format

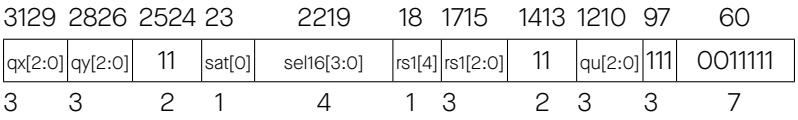


Figure 4.7-58. ESP.VSMULAS.S16.QACC.LD.INCP

4.7.2.59 ESP.VSMULAS.S8.QACC

Syntax

ESP.VSMULAS.S8.QACC qx,qy,sel16,sat

Description

This instruction selects one out of the 16 8-bit data segments in the register `qy` according to the immediate number `sel16` and performs a signed multiplication on it and the 16 8-bit data segments in the register `qx` respectively. The 16 results obtained are added to the corresponding 32-bit data segments in the special registers `QACC_H` and `QACC_L` respectively. Then, each result is saturated to a 32-bit signed number and then stored in the corresponding 32-bit data segment in `QACC_H` and `QACC_L`.

Operation

```
temp[7:0] = qy[sel16*8+7:sel16*8]
QACC_L[ 31:  0] = min(max(QACC_L[ 31:  0] + qx[  7:  0] * temp[7:0], -2^{31}), 2^{31}-1)
QACC_L[ 63: 32] = min(max(QACC_L[ 63: 32] + qx[ 15:  8] * temp[7:0], -2^{31}), 2^{31}-1)
QACC_L[ 95: 64] = min(max(QACC_L[ 95: 64] + qx[ 23: 16] * temp[7:0], -2^{31}), 2^{31}-1)
QACC_L[127: 96] = min(max(QACC_L[128: 96] + qx[ 31: 24] * temp[7:0], -2^{31}), 2^{31}-1)
QACC_L[159:128] = min(max(QACC_L[159:128] + qx[ 39: 32] * temp[7:0], -2^{31}), 2^{31}-1)
QACC_L[191:160] = min(max(QACC_L[191:160] + qx[ 47: 40] * temp[7:0], -2^{31}), 2^{31}-1)
QACC_L[223:192] = min(max(QACC_L[223:192] + qx[ 55: 48] * temp[7:0], -2^{31}), 2^{31}-1)
QACC_L[255:224] = min(max(QACC_L[255:224] + qx[ 63: 56] * temp[7:0], -2^{31}), 2^{31}-1)
QACC_H[ 31:  0] = min(max(QACC_H[ 31:  0] + qx[ 71: 64] * temp[7:0], -2^{31}), 2^{31}-1)
QACC_H[ 63: 32] = min(max(QACC_H[ 63: 32] + qx[ 79: 72] * temp[7:0], -2^{31}), 2^{31}-1)
QACC_H[ 95: 64] = min(max(QACC_H[ 95: 64] + qx[ 87: 80] * temp[7:0], -2^{31}), 2^{31}-1)
QACC_H[127: 96] = min(max(QACC_H[127: 96] + qx[ 95: 88] * temp[7:0], -2^{31}), 2^{31}-1)
QACC_H[159:128] = min(max(QACC_H[159:128] + qx[103: 96] * temp[7:0], -2^{31}), 2^{31}-1)
QACC_H[191:160] = min(max(QACC_H[191:160] + qx[111:104] * temp[7:0], -2^{31}), 2^{31}-1)
QACC_H[223:192] = min(max(QACC_H[223:192] + qx[119:112] * temp[7:0], -2^{31}), 2^{31}-1)
QACC_H[255:224] = min(max(QACC_H[255:224] + qx[127:120] * temp[7:0], -2^{31}),
```

```
    2^{31}-1)
```

Schedule

Operands	Use	Def
qx	1	-
qy	1	-
QACC_H	2	2
QACC_L	2	2

Format

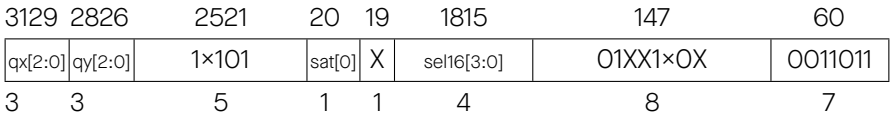


Figure 4.7-59. ESP.VSMULAS.S8.QACC

4.7.2.60 ESP.VSMULAS.S8.QACC.LD.INCP

Syntax

ESP.VSMULAS.S8.QACC.LD.INCP qu,rs1,qx,qy,sel16,sat

Description

This instruction selects one out of the 16 8-bit data segments in the register `qy` according to the immediate number `sel16` and performs a signed multiplication on it and the 16 8-bit data segments in the register `qx` respectively. The 16 results obtained are added to the corresponding 32-bit data segments in the special registers `QACC_H` and `QACC_L` respectively. Then, the result is saturated to a 32-bit signed number and stored in the corresponding 32-bit data segment in `QACC_H` and `QACC_L`.

During the operation, the 16-byte data is loaded from the memory into the register `qu`. After the access is completed, the value in the register `rs1` is incremented by 16.

Operation

```
temp[7:0] = qy[sel16*8+7:sel16*8]
QACC_L[ 31:  0] = min(max(QACC_L[ 31:  0] + qx[  7:  0] * temp[7:0], -2^{31}), 2^{31}-1)
QACC_L[ 63: 32] = min(max(QACC_L[ 63: 32] + qx[ 15:  8] * temp[7:0], -2^{31}), 2^{31}-1)
QACC_L[ 95: 64] = min(max(QACC_L[ 95: 64] + qx[ 23: 16] * temp[7:0], -2^{31}), 2^{31}-1)
QACC_L[127: 96] = min(max(QACC_L[128: 96] + qx[ 31: 24] * temp[7:0], -2^{31}), 2^{31}-1)
QACC_L[159:128] = min(max(QACC_L[159:128] + qx[ 39: 32] * temp[7:0], -2^{31}), 2^{31}-1)
QACC_L[191:160] = min(max(QACC_L[191:160] + qx[ 47: 40] * temp[7:0], -2^{31}), 2^{31}-1)
QACC_L[223:192] = min(max(QACC_L[223:192] + qx[ 55: 48] * temp[7:0], -2^{31}), 2^{31}-1)
QACC_L[255:224] = min(max(QACC_L[255:224] + qx[ 63: 56] * temp[7:0], -2^{31}), 2^{31}-1)
QACC_H[ 31:  0] = min(max(QACC_H[ 31:  0] + qx[ 71: 64] * temp[7:0], -2^{31}), 2^{31}-1)
QACC_H[ 63: 32] = min(max(QACC_H[ 63: 32] + qx[ 79: 72] * temp[7:0], -2^{31}), 2^{31}-1)
QACC_H[ 95: 64] = min(max(QACC_H[ 95: 64] + qx[ 87: 80] * temp[7:0], -2^{31}), 2^{31}-1)
QACC_H[127: 96] = min(max(QACC_H[127: 96] + qx[ 95: 88] * temp[7:0], -2^{31}), 2^{31}-1)
QACC_H[159:128] = min(max(QACC_H[159:128] + qx[103: 96] * temp[7:0], -2^{31}), 2^{31}-1)
QACC_H[191:160] = min(max(QACC_H[191:160] + qx[111:104] * temp[7:0], -2^{31}), 2^{31}-1)
QACC_H[223:192] = min(max(QACC_H[223:192] + qx[119:112] * temp[7:0], -2^{31}), 2^{31}-1)
QACC_H[255:224] = min(max(QACC_H[255:224] + qx[127:120] * temp[7:0], -2^{31}), 2^{31}-1)

qu[127:0] = load128(rs1[31:0])
rs1[31:0] = rs1[31:0] + 16
```

Schedule

Operands	Use	Def
qu	-	2
rs1	1	1
qx	1	-
qy	1	-
QACC_H	2	2
QACC_L	2	2

Format

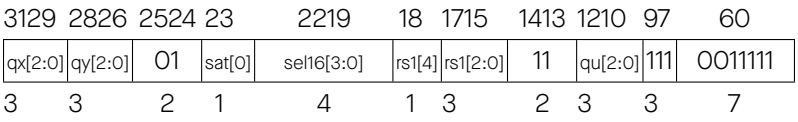


Figure 4.7-60. ESP.VSMULAS.S8.QACC.LD.INCP

4.7.2.61 ESP.VSMULAS.U16.QACC

Syntax

ESP.VSMULAS.U16.QACC qx,qy,sel16,sat

Description

This instruction selects one out of the eight 16-bit data segments in the register `qy` according to the immediate number `sel16` and performs an unsigned multiplication on it and the eight 16-bit data segments in the register `qx` respectively. The eight results obtained are added to the corresponding 64-bit data segments in the special registers `QACC_H` and `QACC_L` respectively. Then, each result is saturated to a 64-bit unsigned number and then stored in the corresponding 64-bit data segment in `QACC_H` and `QACC_L`.

Operation

```
temp[15:0] = qy[sel16*16+15:sel16*16]
QACC_L[ 63:  0] = min(QACC_L[ 63:  0] + qx[ 15:  0] * temp[15:0], 2^{64}-1)
QACC_L[127: 64] = min(QACC_L[127: 64] + qx[ 31: 16] * temp[15:0], 2^{64}-1)
QACC_L[191:128] = min(QACC_L[191:128] + qx[ 47: 32] * temp[15:0], 2^{64}-1)
QACC_L[255:192] = min(QACC_L[255:192] + qx[ 63: 48] * temp[15:0], 2^{64}-1)
QACC_H[ 63:  0] = min(QACC_H[ 63:  0] + qx[ 79: 64] * temp[15:0], 2^{64}-1)
QACC_H[127: 64] = min(QACC_H[127: 64] + qx[ 95: 80] * temp[15:0], 2^{64}-1)
QACC_H[191:128] = min(QACC_H[191:128] + qx[111: 96] * temp[15:0], 2^{64}-1)
QACC_H[255:192] = min(QACC_H[255:192] + qx[127:112] * temp[15:0], 2^{64}-1)
```

Schedule

Operands	Use	Def
qx	1	-
qy	1	-
QACC_H	2	2
QACC_L	2	2

Format

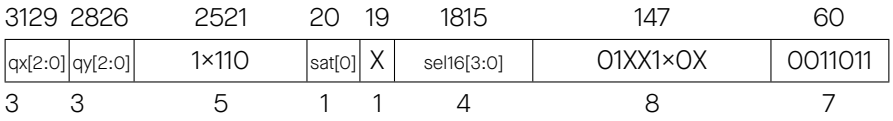


Figure 4.7-61. ESP.VSMULAS.U16.QACC

4.7.2.62 ESP.VSMULAS.U16.QACC.LD.INCP

Syntax

ESP.VSMULAS.U16.QACC.LD.INCP qu,rs1,qx,qy,sel16,sat

Description

This instruction selects one out of the eight 16-bit data segments in the register `qy` according to the immediate number `sel16` and performs an unsigned multiplication on it and the eight 16-bit data segments in the register `qx` respectively. The 8 results obtained are added to the corresponding 64-bit data segments in the special registers `QACC_H` and `QACC_L` respectively. Then, the result is saturated to a 64-bit unsigned number and stored in the corresponding 64-bit data segment in `QACC_H` and `QACC_L`.

During the operation, the 16-byte data is loaded from the memory to the register `qu`. After the access is completed, the value in the register `rs1` is incremented by 16.

Operation

```
temp[15:0] = qy[sel16*16+15:sel16*16]
QACC_L[ 63:  0] = min(QACC_L[ 63:  0] + qx[ 15:  0] * temp[15:0], 2^{64}-1)
QACC_L[127: 64] = min(QACC_L[127: 64] + qx[ 31: 16] * temp[15:0], 2^{64}-1)
QACC_L[191:128] = min(QACC_L[191:128] + qx[ 47: 32] * temp[15:0], 2^{64}-1)
QACC_L[255:192] = min(QACC_L[255:192] + qx[ 63: 48] * temp[15:0], 2^{64}-1)
QACC_H[ 63:  0] = min(QACC_H[ 63:  0] + qx[ 79: 64] * temp[15:0], 2^{64}-1)
QACC_H[127: 64] = min(QACC_H[127: 64] + qx[ 95: 80] * temp[15:0], 2^{64}-1)
QACC_H[191:128] = min(QACC_H[191:128] + qx[111: 96] * temp[15:0], 2^{64}-1)
QACC_H[255:192] = min(QACC_H[255:192] + qx[127:112] * temp[15:0], 2^{64}-1)

qu[127:0] = load128(rs1[31:0])
rs1[31:0] = rs1[31:0] + 16
```

Schedule

Operands	Use	Def
qu	-	2
rs1	1	1
qx	1	-
qy	1	-
QACC_H	2	2
QACC_L	2	2

Format

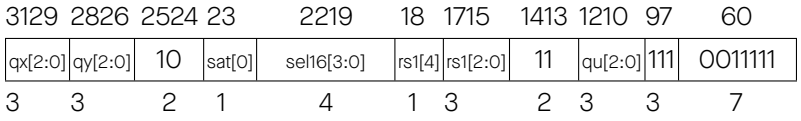


Figure 4.7-62. ESP.VSMULAS.U16.QACC.LD.INCP

4.7.2.63 ESP.VSMULAS.U8.QACC

Syntax

ESP.VSMULAS.U8.QACC qx,qy,sel16,sat

Description

This instruction selects one out of the 16 8-bit data segments in the register `qy` according to the immediate number `sel16` and performs an unsigned multiplication on it and the 16 8-bit data segments in the register `qx` respectively. The 16 results obtained are added to the corresponding 32-bit data segments in the special registers `QACC_H` and `QACC_L` respectively. Then, each result is saturated to a 32-bit unsigned number and then stored in the corresponding 32-bit data segment in `QACC_H` and `QACC_L`.

Operation

```
temp[7:0] = qy[sel16*8+7:sel16*8]
QACC_L[ 31:  0] = min(QACC_L[ 31:  0] + qx[  7:  0] * temp[7:0] , 2^{32}-1)
QACC_L[ 63: 32] = min(QACC_L[ 63: 32] + qx[ 15:  8] * temp[7:0] , 2^{32}-1)
QACC_L[ 95: 64] = min(QACC_L[ 95: 64] + qx[ 23: 16] * temp[7:0] , 2^{32}-1)
QACC_L[127: 96] = min(QACC_L[128: 96] + qx[ 31: 24] * temp[7:0] , 2^{32}-1)
QACC_L[159:128] = min(QACC_L[159:128] + qx[ 39: 32] * temp[7:0] , 2^{32}-1)
QACC_L[191:160] = min(QACC_L[191:160] + qx[ 47: 40] * temp[7:0] , 2^{32}-1)
QACC_L[223:192] = min(QACC_L[223:192] + qx[ 55: 48] * temp[7:0] , 2^{32}-1)
QACC_L[255:224] = min(QACC_L[255:224] + qx[ 63: 56] * temp[7:0] , 2^{32}-1)
QACC_H[ 31:  0] = min(QACC_H[ 31:  0] + qx[ 71: 64] * temp[7:0] , 2^{32}-1)
QACC_H[ 63: 32] = min(QACC_H[ 63: 32] + qx[ 79: 72] * temp[7:0] , 2^{32}-1)
QACC_H[ 95: 64] = min(QACC_H[ 95: 64] + qx[ 87: 80] * temp[7:0] , 2^{32}-1)
QACC_H[127: 96] = min(QACC_H[127: 96] + qx[ 95: 88] * temp[7:0] , 2^{32}-1)
QACC_H[159:128] = min(QACC_H[159:128] + qx[103: 96] * temp[7:0] , 2^{32}-1)
QACC_H[191:160] = min(QACC_H[191:160] + qx[111:104] * temp[7:0] , 2^{32}-1)
QACC_H[223:192] = min(QACC_H[223:192] + qx[119:112] * temp[7:0] , 2^{32}-1)
QACC_H[255:224] = min(QACC_H[255:224] + qx[127:120] * temp[7:0] , 2^{32}-1)
```

Schedule

Operands	Use	Def
qx	1	-
qy	1	-
QACC_H	2	2
QACC_L	2	2

Format

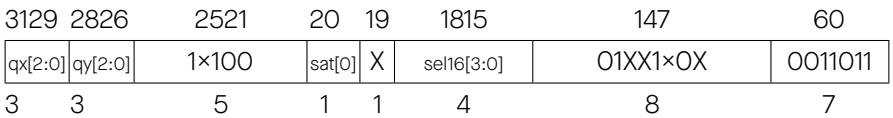


Figure 4.7-63. ESP.VSMULAS.U8.QACC

4.7.2.64 ESP.VSMULAS.U8.QACC.LD.INCP

Syntax

ESP.VSMULAS.U8.QACC.LD.INCP qu,rs1,qx,qy,sel16,sat

Description

This instruction selects one out of the 16 8-bit data segments in the register `qy` according to the immediate number `sel16` and performs an unsigned multiplication on it and the 16 8-bit data segments in the register `qx` respectively. The 16 results obtained are added to the corresponding 32-bit data segments in the special registers `QACC_H` and `QACC_L` respectively. Then, each result is saturated to a 32-bit unsigned number and then stored in the corresponding 32-bit data segment in `QACC_H` and `QACC_L`.

During the operation, the 16-byte data is loaded from the memory into the register `qu`. After the access is completed, the value in the register `rs1` is incremented by 16.

Operation

```
temp[7:0] = qy[sel16*8+7:sel16*8]
QACC_L[ 31:  0] = min(QACC_L[ 31:  0] + qx[  7:  0] * temp[7:0] , 2^{32}-1)
QACC_L[ 63: 32] = min(QACC_L[ 63: 32] + qx[ 15:  8] * temp[7:0] , 2^{32}-1)
QACC_L[ 95: 64] = min(QACC_L[ 95: 64] + qx[ 23: 16] * temp[7:0] , 2^{32}-1)
QACC_L[127: 96] = min(QACC_L[128: 96] + qx[ 31: 24] * temp[7:0] , 2^{32}-1)
QACC_L[159:128] = min(QACC_L[159:128] + qx[ 39: 32] * temp[7:0] , 2^{32}-1)
QACC_L[191:160] = min(QACC_L[191:160] + qx[ 47: 40] * temp[7:0] , 2^{32}-1)
QACC_L[223:192] = min(QACC_L[223:192] + qx[ 55: 48] * temp[7:0] , 2^{32}-1)
QACC_L[255:224] = min(QACC_L[255:224] + qx[ 63: 56] * temp[7:0] , 2^{32}-1)
QACC_H[ 31:  0] = min(QACC_H[ 31:  0] + qx[ 71: 64] * temp[7:0] , 2^{32}-1)
QACC_H[ 63: 32] = min(QACC_H[ 63: 32] + qx[ 79: 72] * temp[7:0] , 2^{32}-1)
QACC_H[ 95: 64] = min(QACC_H[ 95: 64] + qx[ 87: 80] * temp[7:0] , 2^{32}-1)
QACC_H[127: 96] = min(QACC_H[127: 96] + qx[ 95: 88] * temp[7:0] , 2^{32}-1)
QACC_H[159:128] = min(QACC_H[159:128] + qx[103: 96] * temp[7:0] , 2^{32}-1)
QACC_H[191:160] = min(QACC_H[191:160] + qx[111:104] * temp[7:0] , 2^{32}-1)
QACC_H[223:192] = min(QACC_H[223:192] + qx[119:112] * temp[7:0] , 2^{32}-1)
QACC_H[255:224] = min(QACC_H[255:224] + qx[127:120] * temp[7:0] , 2^{32}-1)

qu[127:0] = load128(rs1[31:0])
rs1[31:0] = rs1[31:0] + 16
```

Schedule

Operands	Use	Def
qu	-	2
rs1	1	1
qx	1	-
qy	1	-
QACC_H	2	2
QACC_L	2	2

Format

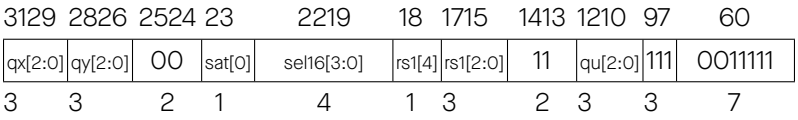


Figure 4.7-64. ESP.VSMULAS.U8.QACC.LD.INCP

4.7.2.65 ESP.CMUL.S16

Syntax

ESP.CMUL.S16 qz,qx,qy,sel4,sat,rm

Description

This instruction performs a 16-bit signed complex multiplication. The range of the immediate number `sel4` is 0 ~ 3, which specifies the 32 bits in the two QR registers `qx` and `qy` for complex multiplication. The real and imaginary parts of the complex numbers are stored in the upper 16 bits and lower 16 bits of the 32 bits, respectively. The calculated real part and imaginary part results are stored in the corresponding 32 bits of the register `qz`.

Operation

```
if sel4 == 0:
  qz[ 15:  0] = (qx[ 15:  0] * qy[ 15:  0] - qx[ 31: 16] * qy[ 31: 16]) >> SAR[5:0]
  qz[ 31: 16] = (qx[ 15:  0] * qy[ 31: 16] + qx[ 31: 16] * qy[ 15:  0]) >> SAR[5:0]
  qz[ 47: 32] = (qx[ 47: 32] * qy[ 47: 32] - qx[ 63: 48] * qy[ 63: 48]) >> SAR[5:0]
  qz[ 63: 48] = (qx[ 47: 32] * qy[ 63: 48] + qx[ 63: 48] * qy[ 47: 32]) >> SAR[5:0]
if sel4 == 1:
  qz[ 79: 64] = (qx[ 79: 64] * qy[ 79: 64] - qx[ 95: 80] * qy[ 95: 80]) >> SAR[5:0]
  qz[ 95: 80] = (qx[ 79: 64] * qy[ 95: 80] + qx[ 95: 80] * qy[ 79: 64]) >> SAR[5:0]
  qz[111: 96] = (qx[111: 96] * qy[111: 96] - qx[127:112] * qy[127:112]) >> SAR[5:0]
  qz[127:112] = (qx[111: 96] * qy[127:112] + qx[127:112] * qy[111: 96]) >> SAR[5:0]
if sel4 == 2:
  qz[ 15:  0] = (qx[ 15:  0] * qy[ 15:  0] + qx[ 31: 16] * qy[ 31: 16]) >> SAR[5:0]
  qz[ 31: 16] = (qx[ 15:  0] * qy[ 31: 16] - qx[ 31: 16] * qy[ 15:  0]) >> SAR[5:0]
  qz[ 47: 32] = (qx[ 47: 32] * qy[ 47: 32] + qx[ 63: 48] * qy[ 63: 48]) >> SAR[5:0]
  qz[ 63: 48] = (qx[ 47: 32] * qy[ 63: 48] - qx[ 63: 48] * qy[ 47: 32]) >> SAR[5:0]
if sel4 == 3:
  qz[ 79: 64] = (qx[ 79: 64] * qy[ 79: 64] + qx[ 95: 80] * qy[ 95: 80]) >> SAR[5:0]
  qz[ 95: 80] = (qx[ 79: 64] * qy[ 95: 80] - qx[ 95: 80] * qy[ 79: 64]) >> SAR[5:0]
  qz[111: 96] = (qx[111: 96] * qy[111: 96] + qx[127:112] * qy[127:112]) >> SAR[5:0]
  qz[127:112] = (qx[111: 96] * qy[127:112] - qx[127:112] * qy[111: 96]) >> SAR[5:0]
```

Schedule

Operands	Use	Def
qz	-	2
qx	1	-
qy	1	-

Format

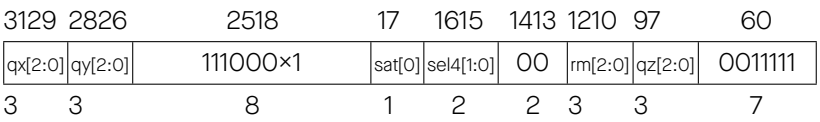


Figure 4.7-65. ESP.CMUL.S16

4.7.2.66 ESP.CMUL.S16.LD.INCP

Syntax

ESP.CMUL.S16.LD.INCP qu,rs1,qz,qx,qy,sel4,sat,rm

Description

This instruction performs a 16-bit signed complex multiplication. The range of the immediate number `sel4` is 0 ~ 3, which specifies the 32 bits in the two QR registers `qx` and `qy` for complex multiplication. The real and imaginary parts of complex numbers are stored in the upper 16 bits and lower 16 bits of the 32 bits respectively. The calculated real part and imaginary part results are stored in the corresponding 32 bits of the register `qz`.

Meanwhile, it also loads 16-byte data from the memory into the register `qu`. After the access, the value in the register `rs1` is incremented by 16.

Operation

```
if sel4 == 0:
  qz[ 15:  0] = (qx[ 15:  0] * qy[ 15:  0] - qx[ 31: 16] * qy[ 31: 16]) >> SAR[5:0]
  qz[ 31: 16] = (qx[ 15:  0] * qy[ 31: 16] + qx[ 31: 16] * qy[ 15:  0]) >> SAR[5:0]
  qz[ 47: 32] = (qx[ 47: 32] * qy[ 47: 32] - qx[ 63: 48] * qy[ 63: 48]) >> SAR[5:0]
  qz[ 63: 48] = (qx[ 47: 32] * qy[ 63: 48] + qx[ 63: 48] * qy[ 47: 32]) >> SAR[5:0]
if sel4 == 1:
  qz[ 79: 64] = (qx[ 79: 64] * qy[ 79: 64] - qx[ 95: 80] * qy[ 95: 80]) >> SAR[5:0]
  qz[ 95: 80] = (qx[ 79: 64] * qy[ 95: 80] + qx[ 95: 80] * qy[ 79: 64]) >> SAR[5:0]
  qz[111: 96] = (qx[111: 96] * qy[111: 96] - qx[127:112] * qy[127:112]) >> SAR[5:0]
  qz[127:112] = (qx[111: 96] * qy[127:112] + qx[127:112] * qy[111: 96]) >> SAR[5:0]
if sel4 == 2:
  qz[ 15:  0] = (qx[ 15:  0] * qy[ 15:  0] + qx[ 31: 16] * qy[ 31: 16]) >> SAR[5:0]
  qz[ 31: 16] = (qx[ 15:  0] * qy[ 31: 16] - qx[ 31: 16] * qy[ 15:  0]) >> SAR[5:0]
  qz[ 47: 32] = (qx[ 47: 32] * qy[ 47: 32] + qx[ 63: 48] * qy[ 63: 48]) >> SAR[5:0]
  qz[ 63: 48] = (qx[ 47: 32] * qy[ 63: 48] - qx[ 63: 48] * qy[ 47: 32]) >> SAR[5:0]
if sel4 == 3:
  qz[ 79: 64] = (qx[ 79: 64] * qy[ 79: 64] + qx[ 95: 80] * qy[ 95: 80]) >> SAR[5:0]
  qz[ 95: 80] = (qx[ 79: 64] * qy[ 95: 80] - qx[ 95: 80] * qy[ 79: 64]) >> SAR[5:0]
  qz[111: 96] = (qx[111: 96] * qy[111: 96] + qx[127:112] * qy[127:112]) >> SAR[5:0]
  qz[127:112] = (qx[111: 96] * qy[127:112] - qx[127:112] * qy[111: 96]) >> SAR[5:0]

  qu[127:0] = load128(rs1[31:0])
  rs1[31:0] = rs1[31:0] + 16
```

Schedule

Operands	Use	Def
qz	-	2
qx	1	-
qy	1	-
qu	-	2
rs1	1	1

Format

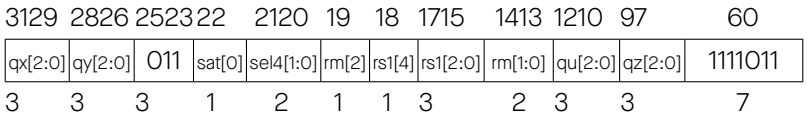


Figure 4.7-66. ESP.CMUL.S16.LD.INCP

4.7.2.67 ESP.CMUL.S16.ST.INCP

Syntax

ESP.CMUL.S16.ST.INCP qu,rs1,qz,qx,qy,sel4,sat,rm

Description

This instruction performs a 16-bit signed complex multiplication. The range of the immediate number `sel4` is 0 ~ 3, which specifies the 32 bits in the two QR registers `qx` and `qy` for complex multiplication. The real and imaginary parts of complex numbers are stored in the upper 16 bits and lower 16 bits of the 32 bits, respectively. The calculated real part and imaginary part results are stored in the corresponding 32 bits of the `qz` register.

Meanwhile, it also stores the 16-byte data of `qu` in memory. After the access, the value in the `rs1` register is incremented by 16.

Operation

```
if sel4 == 0:
  qz[ 15:  0] = (qx[ 15:  0] * qy[ 15:  0] - qx[ 31: 16] * qy[ 31: 16]) >> SAR[5:0]
  qz[ 31: 16] = (qx[ 15:  0] * qy[ 31: 16] + qx[ 31: 16] * qy[ 15:  0]) >> SAR[5:0]
  qz[ 47: 32] = (qx[ 47: 32] * qy[ 47: 32] - qx[ 63: 48] * qy[ 63: 48]) >> SAR[5:0]
  qz[ 63: 48] = (qx[ 47: 32] * qy[ 63: 48] + qx[ 63: 48] * qy[ 47: 32]) >> SAR[5:0]
if sel4 == 1:
  qz[ 79: 64] = (qx[ 79: 64] * qy[ 79: 64] - qx[ 95: 80] * qy[ 95: 80]) >> SAR[5:0]
  qz[ 95: 80] = (qx[ 79: 64] * qy[ 95: 80] + qx[ 95: 80] * qy[ 79: 64]) >> SAR[5:0]
  qz[111: 96] = (qx[111: 96] * qy[111: 96] - qx[127:112] * qy[127:112]) >> SAR[5:0]
  qz[127:112] = (qx[111: 96] * qy[127:112] + qx[127:112] * qy[111: 96]) >> SAR[5:0]
if sel4 == 2:
  qz[ 15:  0] = (qx[ 15:  0] * qy[ 15:  0] + qx[ 31: 16] * qy[ 31: 16]) >> SAR[5:0]
  qz[ 31: 16] = (qx[ 15:  0] * qy[ 31: 16] - qx[ 31: 16] * qy[ 15:  0]) >> SAR[5:0]
  qz[ 47: 32] = (qx[ 47: 32] * qy[ 47: 32] + qx[ 63: 48] * qy[ 63: 48]) >> SAR[5:0]
  qz[ 63: 48] = (qx[ 47: 32] * qy[ 63: 48] - qx[ 63: 48] * qy[ 47: 32]) >> SAR[5:0]
if sel4 == 3:
  qz[ 79: 64] = (qx[ 79: 64] * qy[ 79: 64] + qx[ 95: 80] * qy[ 95: 80]) >> SAR[5:0]
  qz[ 95: 80] = (qx[ 79: 64] * qy[ 95: 80] - qx[ 95: 80] * qy[ 79: 64]) >> SAR[5:0]
  qz[111: 96] = (qx[111: 96] * qy[111: 96] + qx[127:112] * qy[127:112]) >> SAR[5:0]
  qz[127:112] = (qx[111: 96] * qy[127:112] - qx[127:112] * qy[111: 96]) >> SAR[5:0]

  qu[127:0] => store128(rs1[31:0])
  rs1[31:0] = rs1[31:0] + 16
```

Schedule

Operands	Use	Def
qz	-	2
qx	1	-

Operands	Use	Def
qy	1	-
qu	2	-
rs1	1	1

Format

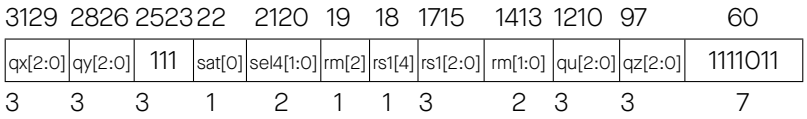


Figure 4.7-67. ESP.CMUL.S16.ST.INCP

4.7.2.68 ESP.CMUL.S8

Syntax

ESP.CMUL.S8 qz,qx,qy,sel4,sat,rm

Description

This instruction performs an 8-bit signed complex multiplication. The range of the immediate number `sel4` is 0 ~ 3, which specifies the 16 bits in the two QR registers `qx` and `qy` for complex multiplication. The real and imaginary parts of complex numbers are stored in the upper 8 bits and lower 8 bits of the 16 bits respectively. The calculated real part and imaginary part results are stored in the corresponding 16 bits of the register `qz`.

Operation

```
if sel4 == 0:
  qz[  7:  0] = (qx[  7:  0] * qy[  7:  0] - qx[ 15:  8] * qy[ 15:  8]) >> SAR[5:0]
  qz[ 15:  8] = (qx[  7:  0] * qy[ 15:  8] + qx[ 15:  8] * qy[  7:  0]) >> SAR[5:0]
  qz[ 23: 16] = (qx[ 23: 16] * qy[ 23: 16] - qx[ 31: 16] * qy[ 31: 24]) >> SAR[5:0]
  qz[ 31: 24] = (qx[ 23: 16] * qy[ 31: 24] + qx[ 31: 16] * qy[ 23: 16]) >> SAR[5:0]
  qz[ 39: 32] = (qx[ 39: 32] * qy[ 39: 32] - qx[ 47: 40] * qy[ 47: 40]) >> SAR[5:0]
  qz[47 : 40] = (qx[39 : 32] * qy[47 : 40] + qx[47 : 40] * qy[39 : 32]) >> SAR[5:0]
  qz[55 : 48] = (qx[55 : 48] * qy[55 : 48] - qx[63 : 56] * qy[63 : 56]) >> SAR[5:0]
  qz[63 : 56] = (qx[55 : 48] * qy[63 : 56] + qx[63 : 56] * qy[55 : 48]) >> SAR[5:0]
else if sel4 == 1:
  qz[71 : 64] = (qx[71 : 64] * qy[71 : 64] - qx[79 : 72] * qy[79 : 72]) >> SAR[5:0]
  qz[79 : 72] = (qx[71 : 64] * qy[79 : 72] + qx[79 : 72] * qy[71 : 64]) >> SAR[5:0]
  qz[87 : 80] = (qx[87 : 80] * qy[87 : 80] - qx[95 : 88] * qy[95 : 88]) >> SAR[5:0]
  qz[95 : 88] = (qx[87 : 80] * qy[95 : 88] + qx[95 : 88] * qy[87 : 80]) >> SAR[5:0]
  qz[103: 96] = (qx[103: 96] * qy[103: 96] - qx[111:104] * qy[111:104]) >> SAR[5:0]
  qz[111:104] = (qx[103: 96] * qy[111:104] + qx[111:104] * qy[103: 96]) >> SAR[5:0]
  qz[119:112] = (qx[119:112] * qy[119:112] - qx[127:120] * qy[127:120]) >> SAR[5:0]
  qz[127:120] = (qx[119:112] * qy[127:120] + qx[127:120] * qy[119:112]) >> SAR[5:0]
else if sel4 == 2:
  qz[7  :  0] = (qx[7  :  0] * qy[7  :  0] + qx[15 :  8] * qy[15 :  8]) >> SAR[5:0]
  qz[15 :  8] = (qx[7  :  0] * qy[15 :  8] - qx[15 :  8] * qy[7  :  0]) >> SAR[5:0]
  qz[23 : 16] = (qx[23 : 16] * qy[23 : 16] + qx[31 : 24] * qy[31 : 24]) >> SAR[5:0]
  qz[31 : 24] = (qx[23 : 16] * qy[31 : 24] - qx[31 : 24] * qy[23 : 16]) >> SAR[5:0]
  qz[39 : 32] = (qx[39 : 32] * qy[39 : 32] + qx[47 : 40] * qy[47 : 40]) >> SAR[5:0]
  qz[47 : 40] = (qx[39 : 32] * qy[47 : 40] - qx[47 : 40] * qy[39 : 32]) >> SAR[5:0]
  qz[55 : 48] = (qx[55 : 48] * qy[55 : 48] + qx[63 : 56] * qy[63 : 56]) >> SAR[5:0]
  qz[63 : 56] = (qx[55 : 48] * qy[63 : 56] - qx[63 : 56] * qy[55 : 48]) >> SAR[5:0]
else if sel4 == 3:
  qz[71 : 64] = (qx[71 : 64] * qy[71 : 64] + qx[79 : 72] * qy[79 : 72]) >> SAR[5:0]
  qz[79 : 72] = (qx[71 : 64] * qy[79 : 72] - qx[79 : 72] * qy[71 : 64]) >> SAR[5:0]
  qz[87 : 80] = (qx[87 : 80] * qy[87 : 80] + qx[95 : 88] * qy[95 : 88]) >> SAR[5:0]
  qz[95 : 88] = (qx[87 : 80] * qy[95 : 88] - qx[95 : 88] * qy[87 : 80]) >> SAR[5:0]
  qz[103: 96] = (qx[103: 96] * qy[103: 96] + qx[111:104] * qy[111:104]) >> SAR[5:0]
  qz[111:104] = (qx[103: 96] * qy[111:104] - qx[111:104] * qy[103: 96]) >> SAR[5:0]
  qz[119:112] = (qx[119:112] * qy[119:112] + qx[127:120] * qy[127:120]) >> SAR[5:0]
  qz[127:120] = (qx[119:112] * qy[127:120] - qx[127:120] * qy[119:112]) >> SAR[5:0]
```

Schedule

Operands	Use	Def
qz	-	2
qx	1	-
qy	1	-

Format

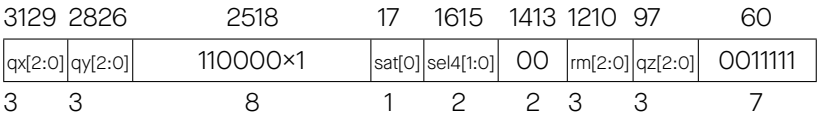


Figure 4.7-68. ESP.CMUL.S8

4.7.2.69 ESP.CMUL.S8.LD.INCP

Syntax

ESP.CMUL.S8.LD.INCP qu,rs1,qz,qx,qy,sel4,sat,rm

Description

This instruction performs an 8-bit signed complex multiplication. The range of the immediate number `sel4` is 0 ~ 3, which specifies the 16 bits in the two QR registers `qx` and `qy` for complex multiplication. The real and imaginary parts of complex numbers are stored in the upper 8 bits and lower 8 bits of the 16 bits respectively. The calculated real part and imaginary part results are stored in the corresponding 16 bits of the register `qz`.

Meanwhile, it also loads 16-byte data from the memory into the register `qu`. After the access, the value in the register `rs1` is incremented by 16.

Operation

```
if sel4 == 0:
  qz[  7:  0] = (qx[  7:  0] * qy[  7:  0] - qx[ 15:  8] * qy[ 15:  8]) >> SAR[5  :0]
  qz[ 15:  8] = (qx[  7:  0] * qy[ 15:  8] + qx[ 15:  8] * qy[  7:  0]) >> SAR[5  :0]
  qz[ 23: 16] = (qx[ 23: 16] * qy[ 23: 16] - qx[ 31: 16] * qy[ 31: 24]) >> SAR[5  :0]
  qz[ 31: 24] = (qx[ 23: 16] * qy[ 31: 24] + qx[ 31: 16] * qy[ 23: 16]) >> SAR[5  :0]
  qz[ 39: 32] = (qx[ 39: 32] * qy[ 39: 32] - qx[ 47: 40] * qy[ 47: 40]) >> SAR[5  :0]
  qz[47 : 40] = (qx[39 : 32] * qy[47 : 40] + qx[47 : 40] * qy[39 : 32]) >> SAR[5  :0]
  qz[55 : 48] = (qx[55 : 48] * qy[55 : 48] - qx[63 : 56] * qy[63 : 56]) >> SAR[5  :0]
  qz[63 : 56] = (qx[55 : 48] * qy[63 : 56] + qx[63 : 56] * qy[55 : 48]) >> SAR[5  :0]
else if sel4 == 1:
  qz[71 : 64] = (qx[71 : 64] * qy[71 : 64] - qx[79 : 72] * qy[79 : 72]) >> SAR[5  :0]
  qz[79 : 72] = (qx[71 : 64] * qy[79 : 72] + qx[79 : 72] * qy[71 : 64]) >> SAR[5  :0]
  qz[87 : 80] = (qx[87 : 80] * qy[87 : 80] - qx[95 : 88] * qy[95 : 88]) >> SAR[5  :0]
  qz[95 : 88] = (qx[87 : 80] * qy[95 : 88] + qx[95 : 88] * qy[87 : 80]) >> SAR[5  :0]
  qz[103: 96] = (qx[103: 96] * qy[103: 96] - qx[111:104] * qy[111:104]) >> SAR[5  :0]
  qz[111:104] = (qx[103: 96] * qy[111:104] + qx[111:104] * qy[103: 96]) >> SAR[5  :0]
  qz[119:112] = (qx[119:112] * qy[119:112] - qx[127:120] * qy[127:120]) >> SAR[5  :0]
  qz[127:120] = (qx[119:112] * qy[127:120] + qx[127:120] * qy[119:112]) >> SAR[5  :0]
else if sel4 == 2:
  qz[7  :  0] = (qx[7  :  0] * qy[7  :  0] + qx[15 :  8] * qy[15 :  8]) >> SAR[5  :
```

```
    0]
  qz[15 :  8] = (qx[7  :  0] * qy[15 :  8] - qx[15 :  8] * qy[7  :  0]) >> SAR[5  :
    0]
  qz[23 : 16] = (qx[23 : 16] * qy[23 : 16] + qx[31 : 24] * qy[31 : 24]) >> SAR[5  :
    0]
  qz[31 : 24] = (qx[23 : 16] * qy[31 : 24] - qx[31 : 24] * qy[23 : 16]) >> SAR[5  :
    0]
  qz[39 : 32] = (qx[39 : 32] * qy[39 : 32] + qx[47 : 40] * qy[47 : 40]) >> SAR[5  :
    0]
  qz[47 : 40] = (qx[39 : 32] * qy[47 : 40] - qx[47 : 40] * qy[39 : 32]) >> SAR[5  :
    0]
  qz[55 : 48] = (qx[55 : 48] * qy[55 : 48] + qx[63 : 56] * qy[63 : 56]) >> SAR[5  :
    0]
  qz[63 : 56] = (qx[55 : 48] * qy[63 : 56] - qx[63 : 56] * qy[55 : 48]) >> SAR[5  :
    0]
else if sel4 == 3:
  qz[71 : 64] = (qx[71 : 64] * qy[71 : 64] + qx[79 : 72] * qy[79 : 72]) >> SAR[5  :
    0]
  qz[79 : 72] = (qx[71 : 64] * qy[79 : 72] - qx[79 : 72] * qy[71 : 64]) >> SAR[5  :
    0]
  qz[87 : 80] = (qx[87 : 80] * qy[87 : 80] + qx[95 : 88] * qy[95 : 88]) >> SAR[5  :
    0]
  qz[95 : 88] = (qx[87 : 80] * qy[95 : 88] - qx[95 : 88] * qy[87 : 80]) >> SAR[5  :
    0]
  qz[103: 96] = (qx[103: 96] * qy[103: 96] + qx[111:104] * qy[111:104]) >> SAR[5  :
    0]
  qz[111:104] = (qx[103: 96] * qy[111:104] - qx[111:104] * qy[103: 96]) >> SAR[5  :
    0]
  qz[119:112] = (qx[119:112] * qy[119:112] + qx[127:120] * qy[127:120]) >> SAR[5  :
    0]
  qz[127:120] = (qx[119:112] * qy[127:120] - qx[127:120] * qy[119:112]) >> SAR[5  :
    0]

qu[127:0] = load128(rs1[31:0])
rs1[31:0] = rs1[31:0] + 16
```

Schedule

Operands	Use	Def
qz	-	2
qx	1	-
qy	1	-
qu	-	2
rs1	1	1

Format

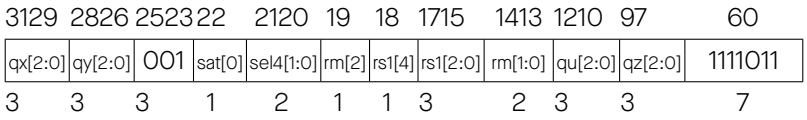


Figure 4.7-69. ESP.CMUL.S8.LD.INCP

4.7.2.70 ESP.CMUL.S8.ST.INCP

Syntax

ESP.CMUL.S8.ST.INCP qu,rs1,qz,qx,qy,sel4,sat,rm

Description

This instruction performs an 8-bit signed complex multiplication. The range of the immediate number `sel4` is 0 ~ 3, which specifies the 16 bits in the two QR registers `qx` and `qy` for complex multiplication. The real and imaginary parts of complex numbers are stored in the upper 8 bits and lower 8 bits of the 16 bits respectively. The calculated real part and imaginary part results are stored in the corresponding 16 bits of the register `qz`.

Meanwhile, it also stores the 16-byte data of `qu` in memory. After the access, the value in the register `rs1` is incremented by 16.

Operation

```
if sel4 == 0:
  qz[  7:  0] = (qx[  7:  0] * qy[  7:  0] - qx[ 15:  8] * qy[ 15:  8]) >> SAR[5  :0]
  qz[ 15:  8] = (qx[  7:  0] * qy[ 15:  8] + qx[ 15:  8] * qy[  7:  0]) >> SAR[5  :0]
  qz[ 23: 16] = (qx[ 23: 16] * qy[ 23: 16] - qx[ 31: 16] * qy[ 31: 24]) >> SAR[5  :0]
  qz[ 31: 24] = (qx[ 23: 16] * qy[ 31: 24] + qx[ 31: 16] * qy[ 23: 16]) >> SAR[5  :0]
  qz[ 39: 32] = (qx[ 39: 32] * qy[ 39: 32] - qx[ 47: 40] * qy[ 47: 40]) >> SAR[5  :0]
  qz[47 : 40] = (qx[39 : 32] * qy[47 : 40] + qx[47 : 40] * qy[39 : 32]) >> SAR[5  :0]
  qz[55 : 48] = (qx[55 : 48] * qy[55 : 48] - qx[63 : 56] * qy[63 : 56]) >> SAR[5  :0]
  qz[63 : 56] = (qx[55 : 48] * qy[63 : 56] + qx[63 : 56] * qy[55 : 48]) >> SAR[5  :0]
else if sel4 == 1:
  qz[71 : 64] = (qx[71 : 64] * qy[71 : 64] - qx[79 : 72] * qy[79 : 72]) >> SAR[5  :0]
  qz[79 : 72] = (qx[71 : 64] * qy[79 : 72] + qx[79 : 72] * qy[71 : 64]) >> SAR[5  :0]
  qz[87 : 80] = (qx[87 : 80] * qy[87 : 80] - qx[95 : 88] * qy[95 : 88]) >> SAR[5  :0]
  qz[95 : 88] = (qx[87 : 80] * qy[95 : 88] + qx[95 : 88] * qy[87 : 80]) >> SAR[5  :0]
  qz[103: 96] = (qx[103: 96] * qy[103: 96] - qx[111:104] * qy[111:104]) >> SAR[5  :0]
  qz[111:104] = (qx[103: 96] * qy[111:104] + qx[111:104] * qy[103: 96]) >> SAR[5  :
```

```
    0]
  qz[119:112] = (qx[119:112] * qy[119:112] - qx[127:120] * qy[127:120]) >> SAR[5  :
    0]
  qz[127:120] = (qx[119:112] * qy[127:120] + qx[127:120] * qy[119:112]) >> SAR[5  :
    0]
else if sel4 == 2:
  qz[7  :  0] = (qx[7  :  0] * qy[7  :  0] + qx[15 :  8] * qy[15 :  8]) >> SAR[5  :
    0]
  qz[15 :  8] = (qx[7  :  0] * qy[15 :  8] - qx[15 :  8] * qy[7  :  0]) >> SAR[5  :
    0]
  qz[23 : 16] = (qx[23 : 16] * qy[23 : 16] + qx[31 : 24] * qy[31 : 24]) >> SAR[5  :
    0]
  qz[31 : 24] = (qx[23 : 16] * qy[31 : 24] - qx[31 : 24] * qy[23 : 16]) >> SAR[5 :
    0]
  qz[39 : 32] = (qx[39 : 32] * qy[39 : 32] + qx[47 : 40] * qy[47 : 40]) >> SAR[5  :
    0]
  qz[47 : 40] = (qx[39 : 32] * qy[47 : 40] - qx[47 : 40] * qy[39 : 32]) >> SAR[5  :
    0]
  qz[55 : 48] = (qx[55 : 48] * qy[55 : 48] + qx[63 : 56] * qy[63 : 56]) >> SAR[5  :
    0]
  qz[63 : 56] = (qx[55 : 48] * qy[63 : 56] - qx[63 : 56] * qy[55 : 48]) >> SAR[5  :
    0]
else if sel4 == 3:
  qz[71 : 64] = (qx[71 : 64] * qy[71 : 64] + qx[79 : 72] * qy[79 : 72]) >> SAR[5  :
    0]
  qz[79 : 72] = (qx[71 : 64] * qy[79 : 72] - qx[79 : 72] * qy[71 : 64]) >> SAR[5  :
    0]
  qz[87 : 80] = (qx[87 : 80] * qy[87 : 80] + qx[95 : 88] * qy[95 : 88]) >> SAR[5  :
    0]
  qz[95 : 88] = (qx[87 : 80] * qy[95 : 88] - qx[95 : 88] * qy[87 : 80]) >> SAR[5  :
    0]
  qz[103: 96] = (qx[103: 96] * qy[103: 96] + qx[111:104] * qy[111:104]) >> SAR[5  :
    0]
  qz[111:104] = (qx[103: 96] * qy[111:104] - qx[111:104] * qy[103: 96]) >> SAR[5  :
    0]
  qz[119:112] = (qx[119:112] * qy[119:112] + qx[127:120] * qy[127:120]) >> SAR[5  :
    0]
  qz[127:120] = (qx[119:112] * qy[127:120] - qx[127:120] * qy[119:112]) >> SAR[5  :
    0]

  qu[127:0] => store128(rs1[31:0])
  rs1[31:0] = rs1[31:0] + 16
```

Schedule

Operands	Use	Def
qz	-	2
qx	1	-
qy	1	-
qu	2	-

Operands	Use	Def
rs1	1	1

Format

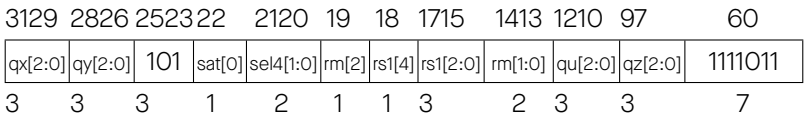


Figure 4.7-70. ESP.CMUL.S8.ST.INCP

4.7.2.71 ESP.CMUL.U16

Syntax

ESP.CMUL.U16 qz,qx,qy,sel4,sat,rm

Description

This instruction performs a 16-bit unsigned complex multiplication. The range of the immediate number `sel4` is 0 ~ 3, which specifies the 32 bits in the two QR registers `qx` and `qy` for complex multiplication. The real and imaginary parts of the complex numbers are stored in the upper 16 bits and lower 16 bits of the 32 bits, respectively. The calculated real part and imaginary part results are stored in the corresponding 32 bits of the register `qz`.

Operation

```
if sel4 == 0:
  qz[ 15:  0] = (qx[ 15:  0] * qy[ 15:  0] - qx[ 31: 16] * qy[ 31: 16]) >> SAR[5:0]
  qz[ 31: 16] = (qx[ 15:  0] * qy[ 31: 16] + qx[ 31: 16] * qy[ 15:  0]) >> SAR[5:0]
  qz[ 47: 32] = (qx[ 47: 32] * qy[ 47: 32] - qx[ 63: 48] * qy[ 63: 48]) >> SAR[5:0]
  qz[ 63: 48] = (qx[ 47: 32] * qy[ 63: 48] + qx[ 63: 48] * qy[ 47: 32]) >> SAR[5:0]
if sel4 == 1:
  qz[ 79: 64] = (qx[ 79: 64] * qy[ 79: 64] - qx[ 95: 80] * qy[ 95: 80]) >> SAR[5:0]
  qz[ 95: 80] = (qx[ 79: 64] * qy[ 95: 80] + qx[ 95: 80] * qy[ 79: 64]) >> SAR[5:0]
  qz[111: 96] = (qx[111: 96] * qy[111: 96] - qx[127:112] * qy[127:112]) >> SAR[5:0]
  qz[127:112] = (qx[111: 96] * qy[127:112] + qx[127:112] * qy[111: 96]) >> SAR[5:0]
if sel4 == 2:
  qz[ 15:  0] = (qx[ 15:  0] * qy[ 15:  0] + qx[ 31: 16] * qy[ 31: 16]) >> SAR[5:0]
  qz[ 31: 16] = (qx[ 15:  0] * qy[ 31: 16] - qx[ 31: 16] * qy[ 15:  0]) >> SAR[5:0]
  qz[ 47: 32] = (qx[ 47: 32] * qy[ 47: 32] + qx[ 63: 48] * qy[ 63: 48]) >> SAR[5:0]
  qz[ 63: 48] = (qx[ 47: 32] * qy[ 63: 48] - qx[ 63: 48] * qy[ 47: 32]) >> SAR[5:0]
if sel4 == 3:
  qz[ 79: 64] = (qx[ 79: 64] * qy[ 79: 64] + qx[ 95: 80] * qy[ 95: 80]) >> SAR[5:0]
  qz[ 95: 80] = (qx[ 79: 64] * qy[ 95: 80] - qx[ 95: 80] * qy[ 79: 64]) >> SAR[5:0]
  qz[111: 96] = (qx[111: 96] * qy[111: 96] + qx[127:112] * qy[127:112]) >> SAR[5:0]
  qz[127:112] = (qx[111: 96] * qy[127:112] - qx[127:112] * qy[111: 96]) >> SAR[5:0]
```

Schedule

Operands	Use	Def
qz	-	2
qx	1	-
qy	1	-

Format

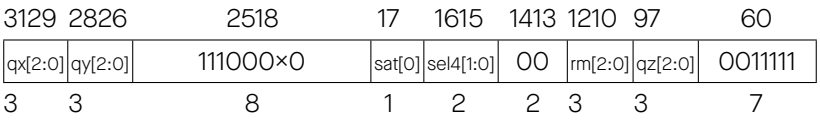


Figure 4.7-71. ESP.CMUL.U16

4.7.2.72 ESP.CMUL.U16.LD.INCP

Syntax

ESP.CMUL.U16.LD.INCP qu,rs1,qz,qx,qy,sel4,sat,rm

Description

This instruction performs a 16-bit unsigned complex multiplication. The range of the immediate number `sel4` is 0 ~ 3, which specifies the 32 bits in the two QR registers `qx` and `qy` for complex multiplication. The real and imaginary parts of complex numbers are stored in the upper 16 bits and lower 16 bits of the 32 bits respectively. The calculated real part and imaginary part results are stored in the corresponding 32 bits of the register `qz`.

Meanwhile, it also loads 16-byte data from the memory into the register `qu`. After the access, the value in the register `rs1` is incremented by 16.

Operation

```
if sel4 == 0:
  qz[ 15:  0] = (qx[ 15:  0] * qy[ 15:  0] - qx[ 31: 16] * qy[ 31: 16]) >> SAR[5:0]
  qz[ 31: 16] = (qx[ 15:  0] * qy[ 31: 16] + qx[ 31: 16] * qy[ 15:  0]) >> SAR[5:0]
  qz[ 47: 32] = (qx[ 47: 32] * qy[ 47: 32] - qx[ 63: 48] * qy[ 63: 48]) >> SAR[5:0]
  qz[ 63: 48] = (qx[ 47: 32] * qy[ 63: 48] + qx[ 63: 48] * qy[ 47: 32]) >> SAR[5:0]
if sel4 == 1:
  qz[ 79: 64] = (qx[ 79: 64] * qy[ 79: 64] - qx[ 95: 80] * qy[ 95: 80]) >> SAR[5:0]
  qz[ 95: 80] = (qx[ 79: 64] * qy[ 95: 80] + qx[ 95: 80] * qy[ 79: 64]) >> SAR[5:0]
  qz[111: 96] = (qx[111: 96] * qy[111: 96] - qx[127:112] * qy[127:112]) >> SAR[5:0]
  qz[127:112] = (qx[111: 96] * qy[127:112] + qx[127:112] * qy[111: 96]) >> SAR[5:0]
if sel4 == 2:
  qz[ 15:  0] = (qx[ 15:  0] * qy[ 15:  0] + qx[ 31: 16] * qy[ 31: 16]) >> SAR[5:0]
  qz[ 31: 16] = (qx[ 15:  0] * qy[ 31: 16] - qx[ 31: 16] * qy[ 15:  0]) >> SAR[5:0]
  qz[ 47: 32] = (qx[ 47: 32] * qy[ 47: 32] + qx[ 63: 48] * qy[ 63: 48]) >> SAR[5:0]
  qz[ 63: 48] = (qx[ 47: 32] * qy[ 63: 48] - qx[ 63: 48] * qy[ 47: 32]) >> SAR[5:0]
if sel4 == 3:
  qz[ 79: 64] = (qx[ 79: 64] * qy[ 79: 64] + qx[ 95: 80] * qy[ 95: 80]) >> SAR[5:0]
  qz[ 95: 80] = (qx[ 79: 64] * qy[ 95: 80] - qx[ 95: 80] * qy[ 79: 64]) >> SAR[5:0]
  qz[111: 96] = (qx[111: 96] * qy[111: 96] + qx[127:112] * qy[127:112]) >> SAR[5:0]
  qz[127:112] = (qx[111: 96] * qy[127:112] - qx[127:112] * qy[111: 96]) >> SAR[5:0]

  qu[127:0] = load128(rs1[31:0])
  rs1[31:0] = rs1[31:0] + 16
```

Schedule

Operands	Use	Def
qz	-	2
qx	1	-
qy	1	-
qu	-	2
rs1	1	1

Format

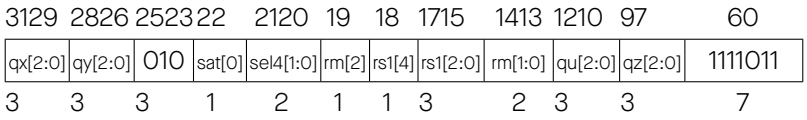


Figure 4.7-72. ESP.CMUL.U16.LD.INCP

4.7.2.73 ESP.CMUL.U16.ST.INCP

Syntax

ESP.CMUL.U16.ST.INCP qu,rs1,qz,qx,qy,sel4,sat,rm

Description

This instruction performs a 16-bit unsigned complex multiplication. The range of the immediate number `sel4` is 0 ~ 3, which specifies the 32 bits in the two QR registers `qx` and `qy` for complex multiplication. The real and imaginary parts of complex numbers are stored in the upper 16 bits and lower 16 bits of the 32 bits, respectively. The calculated real part and imaginary part results are stored in the corresponding 32 bits of the register `qz`.

Meanwhile, it also stores the 16-byte data of `qu` in memory. After the access, the value in the register `rs1` is incremented by 16.

Operation

```
if sel4 == 0:
  qz[ 15:  0] = (qx[ 15:  0] * qy[ 15:  0] - qx[ 31: 16] * qy[ 31: 16]) >> SAR[5:0]
  qz[ 31: 16] = (qx[ 15:  0] * qy[ 31: 16] + qx[ 31: 16] * qy[ 15:  0]) >> SAR[5:0]
  qz[ 47: 32] = (qx[ 47: 32] * qy[ 47: 32] - qx[ 63: 48] * qy[ 63: 48]) >> SAR[5:0]
  qz[ 63: 48] = (qx[ 47: 32] * qy[ 63: 48] + qx[ 63: 48] * qy[ 47: 32]) >> SAR[5:0]
if sel4 == 1:
  qz[ 79: 64] = (qx[ 79: 64] * qy[ 79: 64] - qx[ 95: 80] * qy[ 95: 80]) >> SAR[5:0]
  qz[ 95: 80] = (qx[ 79: 64] * qy[ 95: 80] + qx[ 95: 80] * qy[ 79: 64]) >> SAR[5:0]
  qz[111: 96] = (qx[111: 96] * qy[111: 96] - qx[127:112] * qy[127:112]) >> SAR[5:0]
  qz[127:112] = (qx[111: 96] * qy[127:112] + qx[127:112] * qy[111: 96]) >> SAR[5:0]
if sel4 == 2:
  qz[ 15:  0] = (qx[ 15:  0] * qy[ 15:  0] + qx[ 31: 16] * qy[ 31: 16]) >> SAR[5:0]
  qz[ 31: 16] = (qx[ 15:  0] * qy[ 31: 16] - qx[ 31: 16] * qy[ 15:  0]) >> SAR[5:0]
  qz[ 47: 32] = (qx[ 47: 32] * qy[ 47: 32] + qx[ 63: 48] * qy[ 63: 48]) >> SAR[5:0]
  qz[ 63: 48] = (qx[ 47: 32] * qy[ 63: 48] - qx[ 63: 48] * qy[ 47: 32]) >> SAR[5:0]
if sel4 == 3:
  qz[ 79: 64] = (qx[ 79: 64] * qy[ 79: 64] + qx[ 95: 80] * qy[ 95: 80]) >> SAR[5:0]
  qz[ 95: 80] = (qx[ 79: 64] * qy[ 95: 80] - qx[ 95: 80] * qy[ 79: 64]) >> SAR[5:0]
  qz[111: 96] = (qx[111: 96] * qy[111: 96] + qx[127:112] * qy[127:112]) >> SAR[5:0]
  qz[127:112] = (qx[111: 96] * qy[127:112] - qx[127:112] * qy[111: 96]) >> SAR[5:0]

  qu[127:0] => store128(rs1[31:0])
  rs1[31:0] = rs1[31:0] + 16
```

Schedule

Operands	Use	Def
qz	-	2
qx	1	-

Operands	Use	Def
qy	1	-
qu	2	-
rs1	1	1

Format

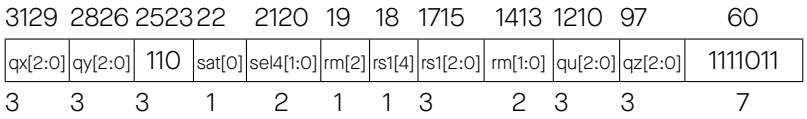


Figure 4.7-73. ESP.CMUL.U16.ST.INCP

4.7.2.74 ESP.CMUL.U8

Syntax

ESP.CMUL.U8 qz,qx,qy,sel4,sat,rm

Description

This instruction performs an 8-bit unsigned complex multiplication. The range of the immediate number `sel4` is 0 ~ 3, which specifies the 16 bits in the two QR registers `qx` and `qy` for complex multiplication. The real and imaginary parts of complex numbers are stored in the upper 8 bits and lower 8 bits of the 16 bits respectively. The calculated real part and imaginary part results are stored in the corresponding 16 bits of the register `qz`.

Operation

```
if sel4 == 0:
  qz[  7:  0] = (qx[  7:  0] * qy[  7:  0] - qx[ 15:  8] * qy[ 15:  8]) >> SAR[5  :
      0]
  qz[ 15:  8] = (qx[  7:  0] * qy[ 15:  8] + qx[ 15:  8] * qy[  7:  0]) >> SAR[5  :
      0]
  qz[ 23: 16] = (qx[ 23: 16] * qy[ 23: 16] - qx[ 31: 16] * qy[ 31: 24]) >> SAR[5  :
      0]
  qz[ 31: 24] = (qx[ 23: 16] * qy[ 31: 24] + qx[ 31: 16] * qy[ 23: 16]) >> SAR[5  :
      0]
  qz[ 39: 32] = (qx[ 39: 32] * qy[ 39: 32] - qx[ 47: 40] * qy[ 47: 40]) >> SAR[5  :
      0]
  qz[47 : 40] = (qx[39 : 32] * qy[47 : 40] + qx[47 : 40] * qy[39 : 32]) >> SAR[5  :
      0]
  qz[55 : 48] = (qx[55 : 48] * qy[55 : 48] - qx[63 : 56] * qy[63 : 56]) >> SAR[5  :
      0]
  qz[63 : 56] = (qx[55 : 48] * qy[63 : 56] + qx[63 : 56] * qy[55 : 48]) >> SAR[5  :
      0]
else if sel4 == 1:
  qz[71 : 64] = (qx[71 : 64] * qy[71 : 64] - qx[79 : 72] * qy[79 : 72]) >> SAR[5  :
      0]
  qz[79 : 72] = (qx[71 : 64] * qy[79 : 72] + qx[79 : 72] * qy[71 : 64]) >> SAR[5  :
      0]
  qz[87 : 80] = (qx[87 : 80] * qy[87 : 80] - qx[95 : 88] * qy[95 : 88]) >> SAR[5  :
      0]
  qz[95 : 88] = (qx[87 : 80] * qy[95 : 88] + qx[95 : 88] * qy[87 : 80]) >> SAR[5  :
      0]
  qz[103: 96] = (qx[103: 96] * qy[103: 96] - qx[111:104] * qy[111:104]) >> SAR[5  :
      0]
  qz[111:104] = (qx[103: 96] * qy[111:104] + qx[111:104] * qy[103: 96]) >> SAR[5  :
      0]
  qz[119:112] = (qx[119:112] * qy[119:112] - qx[127:120] * qy[127:120]) >> SAR[5  :
      0]
  qz[127:120] = (qx[119:112] * qy[127:120] + qx[127:120] * qy[119:112]) >> SAR[5  :
      0]
else if sel4 == 2:
  qz[7  :  0] = (qx[7  :  0] * qy[7  :  0] + qx[15 :  8] * qy[15 :  8]) >> SAR[5  :
      0]
  qz[15 :  8] = (qx[7  :  0] * qy[15 :  8] - qx[15 :  8] * qy[7  :  0]) >> SAR[5  :
```

```
    0]
  qz[23 : 16] = (qx[23 : 16] * qy[23 : 16] + qx[31 : 24] * qy[31 : 24]) >> SAR[5  :
    0]
  qz[31 : 24] = (qx[23 : 16] * qy[31 : 24] - qx[31 : 24] * qy[23 : 16]) >> SAR[5  :
    0]
  qz[39 : 32] = (qx[39 : 32] * qy[39 : 32] + qx[47 : 40] * qy[47 : 40]) >> SAR[5  :
    0]
  qz[47 : 40] = (qx[39 : 32] * qy[47 : 40] - qx[47 : 40] * qy[39 : 32]) >> SAR[5  :
    0]
  qz[55 : 48] = (qx[55 : 48] * qy[55 : 48] + qx[63 : 56] * qy[63 : 56]) >> SAR[5  :
    0]
  qz[63 : 56] = (qx[55 : 48] * qy[63 : 56] - qx[63 : 56] * qy[55 : 48]) >> SAR[5  :
    0]
else if sel4 == 3:
  qz[71 : 64] = (qx[71 : 64] * qy[71 : 64] + qx[79 : 72] * qy[79 : 72]) >> SAR[5  :
    0]
  qz[79 : 72] = (qx[71 : 64] * qy[79 : 72] - qx[79 : 72] * qy[71 : 64]) >> SAR[5  :
    0]
  qz[87 : 80] = (qx[87 : 80] * qy[87 : 80] + qx[95 : 88] * qy[95 : 88]) >> SAR[5  :
    0]
  qz[95 : 88] = (qx[87 : 80] * qy[95 : 88] - qx[95 : 88] * qy[87 : 80]) >> SAR[5  :
    0]
  qz[103: 96] = (qx[103: 96] * qy[103: 96] + qx[111:104] * qy[111:104]) >> SAR[5  :
    0]
  qz[111:104] = (qx[103: 96] * qy[111:104] - qx[111:104] * qy[103: 96]) >> SAR[5  :
    0]
  qz[119:112] = (qx[119:112] * qy[119:112] + qx[127:120] * qy[127:120]) >> SAR[5  :
    0]
  qz[127:120] = (qx[119:112] * qy[127:120] - qx[127:120] * qy[119:112]) >> SAR[5  :
    0]
```

Schedule

Operands	Use	Def
qz	-	2
qx	1	-
qy	1	-

Format

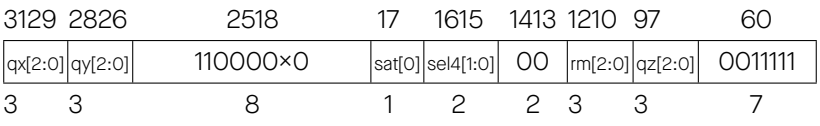


Figure 4.7-74. ESP.CMUL.U8

4.7.2.75 ESP.CMUL.U8.LD.INCP

Syntax

ESP.CMUL.U8.LD.INCP qu,rs1,qz,qx,qy,sel4,sat,rm

Description

This instruction performs an 8-bit unsigned complex multiplication. The range of the immediate number `sel4` is 0 ~ 3, which specifies the 16 bits in the two QR registers `qx` and `qy` for complex multiplication. The real and imaginary parts of complex numbers are stored in the upper 8 bits and lower 8 bits of the 16 bits respectively. The calculated real part and imaginary part results are stored in the corresponding 16 bits of the register `qz`.

Meanwhile, it also loads 16-byte data from the memory into the register `qu`. After the access, the value in the register `rs1` is incremented by 16.

Operation

```
if sel4 == 0:
  qz[  7:  0] = (qx[  7:  0] * qy[  7:  0] - qx[ 15:  8] * qy[ 15:  8]) >> SAR[5  :
      0]
  qz[ 15:  8] = (qx[  7:  0] * qy[ 15:  8] + qx[ 15:  8] * qy[  7:  0]) >> SAR[5  :
      0]
  qz[ 23: 16] = (qx[ 23: 16] * qy[ 23: 16] - qx[ 31: 16] * qy[ 31: 24]) >> SAR[5  :
      0]
  qz[ 31: 24] = (qx[ 23: 16] * qy[ 31: 24] + qx[ 31: 16] * qy[ 23: 16]) >> SAR[5  :
      0]
  qz[ 39: 32] = (qx[ 39: 32] * qy[ 39: 32] - qx[ 47: 40] * qy[ 47: 40]) >> SAR[5  :
      0]
  qz[47 : 40] = (qx[39 : 32] * qy[47 : 40] + qx[47 : 40] * qy[39 : 32]) >> SAR[5  :
      0]
  qz[55 : 48] = (qx[55 : 48] * qy[55 : 48] - qx[63 : 56] * qy[63 : 56]) >> SAR[5  :
      0]
  qz[63 : 56] = (qx[55 : 48] * qy[63 : 56] + qx[63 : 56] * qy[55 : 48]) >> SAR[5  :
      0]
else if sel4 == 1:
  qz[71 : 64] = (qx[71 : 64] * qy[71 : 64] - qx[79 : 72] * qy[79 : 72]) >> SAR[5  :
      0]
  qz[79 : 72] = (qx[71 : 64] * qy[79 : 72] + qx[79 : 72] * qy[71 : 64]) >> SAR[5  :
      0]
  qz[87 : 80] = (qx[87 : 80] * qy[87 : 80] - qx[95 : 88] * qy[95 : 88]) >> SAR[5  :
      0]
  qz[95 : 88] = (qx[87 : 80] * qy[95 : 88] + qx[95 : 88] * qy[87 : 80]) >> SAR[5  :
      0]
  qz[103: 96] = (qx[103: 96] * qy[103: 96] - qx[111:104] * qy[111:104]) >> SAR[5  :
      0]
  qz[111:104] = (qx[103: 96] * qy[111:104] + qx[111:104] * qy[103: 96]) >> SAR[5  :
      0]
  qz[119:112] = (qx[119:112] * qy[119:112] - qx[127:120] * qy[127:120]) >> SAR[5  :
      0]
  qz[127:120] = (qx[119:112] * qy[127:120] + qx[127:120] * qy[119:112]) >> SAR[5  :
      0]
else if sel4 == 2:
  qz[7  :  0] = (qx[7  :  0] * qy[7  :  0] + qx[15 :  8] * qy[15 :  8]) >> SAR[5  :
```

```
    0]
  qz[15 :  8] = (qx[7  :  0] * qy[15 :  8] - qx[15 :  8] * qy[7  :  0]) >> SAR[5  :
    0]
  qz[23 : 16] = (qx[23 : 16] * qy[23 : 16] + qx[31 : 24] * qy[31 : 24]) >> SAR[5  :
    0]
  qz[31 : 24] = (qx[23 : 16] * qy[31 : 24] - qx[31 : 24] * qy[23 : 16]) >> SAR[5  :
    0]
  qz[39 : 32] = (qx[39 : 32] * qy[39 : 32] + qx[47 : 40] * qy[47 : 40]) >> SAR[5  :
    0]
  qz[47 : 40] = (qx[39 : 32] * qy[47 : 40] - qx[47 : 40] * qy[39 : 32]) >> SAR[5  :
    0]
  qz[55 : 48] = (qx[55 : 48] * qy[55 : 48] + qx[63 : 56] * qy[63 : 56]) >> SAR[5  :
    0]
  qz[63 : 56] = (qx[55 : 48] * qy[63 : 56] - qx[63 : 56] * qy[55 : 48]) >> SAR[5  :
    0]
else if sel4 == 3:
  qz[71 : 64] = (qx[71 : 64] * qy[71 : 64] + qx[79 : 72] * qy[79 : 72]) >> SAR[5  :
    0]
  qz[79 : 72] = (qx[71 : 64] * qy[79 : 72] - qx[79 : 72] * qy[71 : 64]) >> SAR[5  :
    0]
  qz[87 : 80] = (qx[87 : 80] * qy[87 : 80] + qx[95 : 88] * qy[95 : 88]) >> SAR[5  :
    0]
  qz[95 : 88] = (qx[87 : 80] * qy[95 : 88] - qx[95 : 88] * qy[87 : 80]) >> SAR[5  :
    0]
  qz[103: 96] = (qx[103: 96] * qy[103: 96] + qx[111:104] * qy[111:104]) >> SAR[5  :
    0]
  qz[111:104] = (qx[103: 96] * qy[111:104] - qx[111:104] * qy[103: 96]) >> SAR[5  :
    0]
  qz[119:112] = (qx[119:112] * qy[119:112] + qx[127:120] * qy[127:120]) >> SAR[5  :
    0]
  qz[127:120] = (qx[119:112] * qy[127:120] - qx[127:120] * qy[119:112]) >> SAR[5  :
    0]

qu[127:0] = load128(rs1[31:0])
rs1[31:0] = rs1[31:0] + 16
```

Schedule

Operands	Use	Def
qz	-	2
qx	1	-
qy	1	-
qu	-	2
rs1	1	1

Format

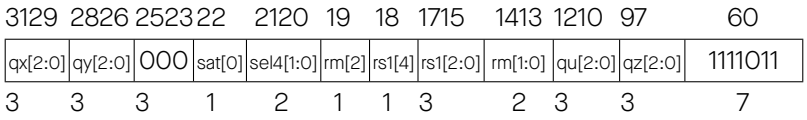


Figure 4.7-75. ESP.CMUL.U8.LD.INCP

4.7.2.76 ESP.CMUL.U8.ST.INCP

Syntax

ESP.CMUL.U8.ST.INCP qu,rs1,qz,qx,qy,sel4,sat,rm

Description

This instruction performs an 8-bit unsigned complex multiplication. The range of the immediate number `sel4` is 0 ~ 3, which specifies the 16 bits in the two QR registers `qx` and `qy` for complex multiplication. The real and imaginary parts of complex numbers are stored in the upper 8 bits and lower 8 bits of the 16 bits respectively. The calculated real part and imaginary part results are stored in the corresponding 16 bits of the register `qz`.

Meanwhile, it also stores the 16-byte data of `qu` in memory. After the access, the value in the register `rs1` is incremented by 16.

Operation

```
if sel4 == 0:
  qz[  7:  0] = (qx[  7:  0] * qy[  7:  0] - qx[ 15:  8] * qy[ 15:  8]) >> SAR[5  :
      0]
  qz[ 15:  8] = (qx[  7:  0] * qy[ 15:  8] + qx[ 15:  8] * qy[  7:  0]) >> SAR[5  :
      0]
  qz[ 23: 16] = (qx[ 23: 16] * qy[ 23: 16] - qx[ 31: 16] * qy[ 31: 24]) >> SAR[5  :
      0]
  qz[ 31: 24] = (qx[ 23: 16] * qy[ 31: 24] + qx[ 31: 16] * qy[ 23: 16]) >> SAR[5  :
      0]
  qz[ 39: 32] = (qx[ 39: 32] * qy[ 39: 32] - qx[ 47: 40] * qy[ 47: 40]) >> SAR[5  :
      0]
  qz[47 : 40] = (qx[39 : 32] * qy[47 : 40] + qx[47 : 40] * qy[39 : 32]) >> SAR[5  :
      0]
  qz[55 : 48] = (qx[55 : 48] * qy[55 : 48] - qx[63 : 56] * qy[63 : 56]) >> SAR[5  :
      0]
  qz[63 : 56] = (qx[55 : 48] * qy[63 : 56] + qx[63 : 56] * qy[55 : 48]) >> SAR[5  :
      0]
else if sel4 == 1:
  qz[71 : 64] = (qx[71 : 64] * qy[71 : 64] - qx[79 : 72] * qy[79 : 72]) >> SAR[5  :
      0]
  qz[79 : 72] = (qx[71 : 64] * qy[79 : 72] + qx[79 : 72] * qy[71 : 64]) >> SAR[5  :
      0]
  qz[87 : 80] = (qx[87 : 80] * qy[87 : 80] - qx[95 : 88] * qy[95 : 88]) >> SAR[5  :
      0]
  qz[95 : 88] = (qx[87 : 80] * qy[95 : 88] + qx[95 : 88] * qy[87 : 80]) >> SAR[5  :
      0]
  qz[103: 96] = (qx[103: 96] * qy[103: 96] - qx[111:104] * qy[111:104]) >> SAR[5  :
      0]
  qz[111:104] = (qx[103: 96] * qy[111:104] + qx[111:104] * qy[103: 96]) >> SAR[5  :
      0]
```

```
  qz[119:112] = (qx[119:112] * qy[119:112] - qx[127:120] * qy[127:120]) >> SAR[5  :
      0]
  qz[127:120] = (qx[119:112] * qy[127:120] + qx[127:120] * qy[119:112]) >> SAR[5  :
      0]
else if sel4 == 2:
  qz[7  :  0] = (qx[7  :  0] * qy[7  :  0] + qx[15 :  8] * qy[15 :  8]) >> SAR[5  :
      0]
  qz[15 :  8] = (qx[7  :  0] * qy[15 :  8] - qx[15 :  8] * qy[7  :  0]) >> SAR[5  :
      0]
  qz[23 : 16] = (qx[23 : 16] * qy[23 : 16] + qx[31 : 24] * qy[31 : 24]) >> SAR[5  :
      0]
  qz[31 : 24] = (qx[23 : 16] * qy[31 : 24] - qx[31 : 24] * qy[23 : 16]) >> SAR[5  :
      0]
  qz[39 : 32] = (qx[39 : 32] * qy[39 : 32] + qx[47 : 40] * qy[47 : 40]) >> SAR[5  :
      0]
  qz[47 : 40] = (qx[39 : 32] * qy[47 : 40] - qx[47 : 40] * qy[39 : 32]) >> SAR[5  :
      0]
  qz[55 : 48] = (qx[55 : 48] * qy[55 : 48] + qx[63 : 56] * qy[63 : 56]) >> SAR[5  :
      0]
  qz[63 : 56] = (qx[55 : 48] * qy[63 : 56] - qx[63 : 56] * qy[55 : 48]) >> SAR[5  :
      0]
else if sel4 == 3:
  qz[71 : 64] = (qx[71 : 64] * qy[71 : 64] + qx[79 : 72] * qy[79 : 72]) >> SAR[5  :
      0]
  qz[79 : 72] = (qx[71 : 64] * qy[79 : 72] - qx[79 : 72] * qy[71 : 64]) >> SAR[5  :
      0]
  qz[87 : 80] = (qx[87 : 80] * qy[87 : 80] + qx[95 : 88] * qy[95 : 88]) >> SAR[5  :
      0]
  qz[95 : 88] = (qx[87 : 80] * qy[95 : 88] - qx[95 : 88] * qy[87 : 80]) >> SAR[5  :
      0]
  qz[103: 96] = (qx[103: 96] * qy[103: 96] + qx[111:104] * qy[111:104]) >> SAR[5  :
      0]
  qz[111:104] = (qx[103: 96] * qy[111:104] - qx[111:104] * qy[103: 96]) >> SAR[5  :
      0]
  qz[119:112] = (qx[119:112] * qy[119:112] + qx[127:120] * qy[127:120]) >> SAR[5  :
      0]
  qz[127:120] = (qx[119:112] * qy[127:120] - qx[127:120] * qy[119:112]) >> SAR[5  :
      0]

  qu[127:0] => store128(rs1[31:0])
  rs1[31:0] = rs1[31:0] + 16
```

Schedule

Operands	Use	Def
qz	-	2
qx	1	-
qy	1	-
qu	2	-
rs1	1	1

Format

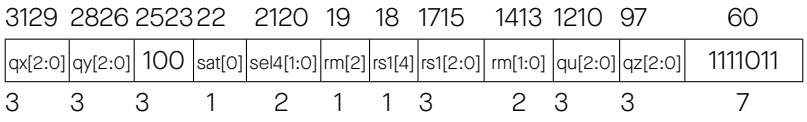


Figure 4.7-76. ESP.CMUL.U8.ST.INCP

4.7.2.77 ESP.MAX.S16.A

Syntax

ESP.MAX.S16.A qw,rd

Description

This instruction saves the maximum signed value of 8 elements of 16 bits from `qw` to `rd`.

Operation

```
    max_value[15 :  0] = max(qw[15 :  0], qw[31 : 16], ..., qw[127:112])
    rd[31 :  0] = {16{max_value[15]}, max_value[15 :  0]}
```

Schedule

Operands	Use	Def
rd	-	2
qw	1	-

Format

3126	2524	2320	19	1811	10	97	60
1100XX	qw[1:0]	00XX	qw[2]	0XXX01XX	rd[4]	rd[2:0]	0011011
6	2	4	1	8	1	3	7

Figure 4.7-77. ESP.MAX.S16.A

4.7.2.78 ESP.MAX.S32.A

Syntax

ESP.MAX.S32.A qw,rd

Description

This instruction saves the maximum signed value of 4 elements of 32 bits from `qw` to `rd`.

Operation

```
max_value[31 :  0] = max(qw[31 :  0], qw[63 : 32], ..., qw[127:96])
rd[31 :  0] = max_value[31 :  0]
```

Schedule

Operands	Use	Def
rd	-	2
qw	1	-

Format

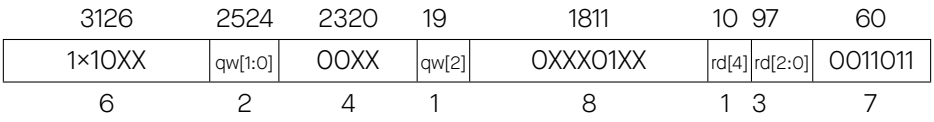


Figure 4.7-78. ESP.MAX.S32.A

4.7.2.79 ESP.MAX.S8.A

Syntax

ESP.MAX.S8.A qw,rd

Description

This instruction saves the maximum signed value of 16 elements of 8 bits from `qw` to `rd`.

Operation

```
max_value[7 :  0] = max(qw[7 :  0], qw[16 : 8], ..., qw[127:120])
rd[31 :  0] = {24{max_value[7]}, max_value[7 :  0]}
```

Schedule

Operands	Use	Def
rd	-	2
qw	1	-

Format

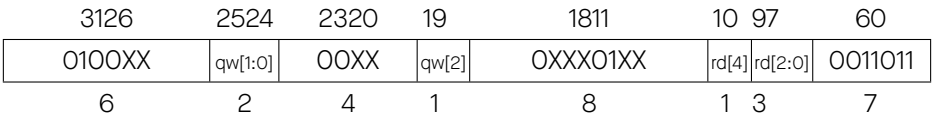


Figure 4.7-79. ESP.MAX.S8.A

4.7.2.80 ESP.MAX.U16.A

Syntax

ESP.MAX.U16.A qw,rd

Description

This instruction saves the maximum unsigned value of 8 elements of 16 bits from `qw` to `rd`.

Operation

```
max_value[15 :  0] = max(qw[15 :  0], qw[31 : 16], ..., qw[127:112])
rd[31 :  0] = {16'b0, max_value[15 :  0]}
```

Schedule

Operands	Use	Def
rd	-	2
qw	1	-

Format

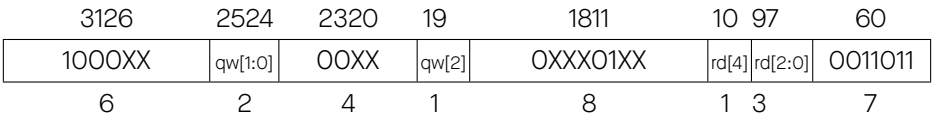


Figure 4.7-80. ESP.MAX.U16.A

4.7.2.81 ESP.MAX.U32.A

Syntax

ESP.MAX.U32.A qw,rd

Description

This instruction saves the maximum unsigned value of 4 elements of 32 bits from `qw` to `rd`.

Operation

```
max_value[31 :  0] = max(qw[31 :  0], qw[63 : 32], ..., qw[127:96])
rd[31 :  0] = max_value[31 :  0]
```

Schedule

Operands	Use	Def
rd	-	2
qw	1	-

Format

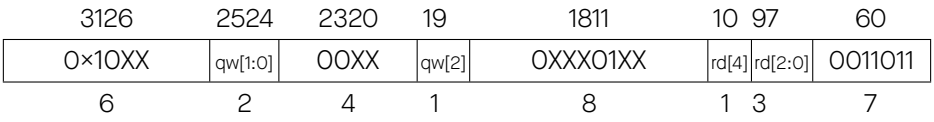


Figure 4.7-81. ESP.MAX.U32.A

4.7.2.82 ESP.MAX.U8.A

Syntax

ESP.MAX.U8.A qw,rd

Description

This instruction saves the maximum unsigned value of 16 elements of 8 bits from `qw` to `rd`.

Operation

```
max_value[7 :  0] = max(qw[7 :  0], qw[16 : 8], ..., qw[127:120])
rd[31 :  0] = {24{0}, max_value[7 :  0]}
```

Schedule

Operands	Use	Def
rd	-	2
qw	1	-

Format

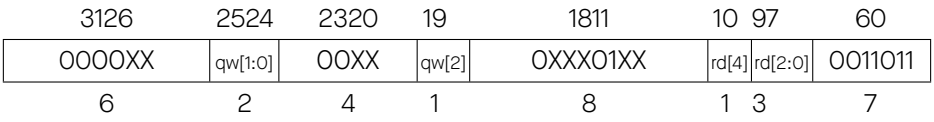


Figure 4.7-82. ESP.MAX.U8.A

4.7.2.83 ESP.MIN.S16.A

Syntax

ESP.MIN.S16.A qw,rd

Description

This instruction saves the minimum signed value of 8 elements of 16 bits from `qw` to `rd`.

Operation

```
    min_value[15 :  0] = min(qw[15 :  0], qw[31 : 16], ..., qw[127:112])
    rd[31 :  0] = {16{min_value[15]}, min_value[15 :  0]}
```

Schedule

Operands	Use	Def
rd	-	2
qw	1	-

Format

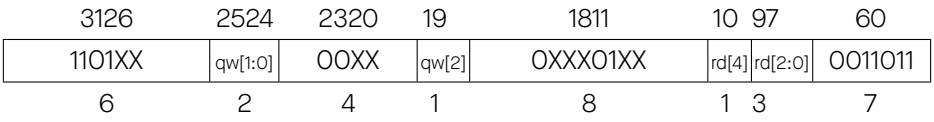


Figure 4.7-83. ESP.MIN.S16.A

4.7.2.84 ESP.MIN.S32.A

Syntax

ESP.MIN.S32.A qw,rd

Description

This instruction saves the minimum signed value of 4 elements of 32 bits of `qw` to `rd`.

Operation

```
min_value[31 :  0] = min(qw[31 :  0], qw[63 : 32], ..., qw[127:96])
rd[31 :  0] = min_value[31 :  0]
```

Schedule

Operands	Use	Def
rd	-	2
qw	1	-

Format

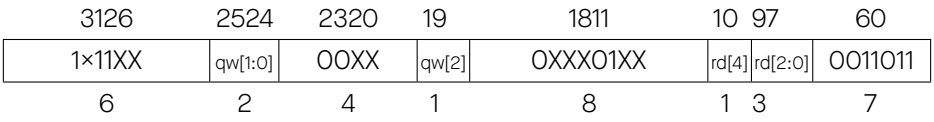


Figure 4.7-84. ESP.MIN.S32.A

4.7.2.85 ESP.MIN.S8.A

Syntax

ESP.MIN.S8.A qw,rd

Description

This instruction saves the minimum signed value of 16 elements of 8 bits from `qw` to `rd`.

Operation

```
    min_value[7 :  0] = min(qw[7 :  0], qw[16 : 8], ..., qw[127:120])
    rd[31 :  0] = {24{min_value[7]}, min_value[7 :  0]}
```

Schedule

Operands	Use	Def
rd	-	2
qw	1	-

Format

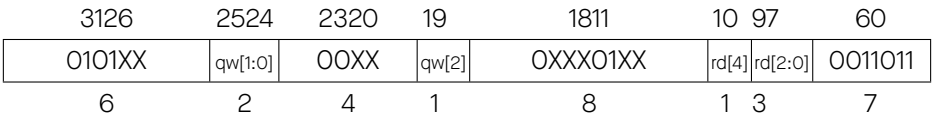


Figure 4.7-85. ESP.MIN.S8.A

4.7.2.86 ESP.MIN.U16.A

Syntax

ESP.MIN.U16.A qw,rd

Description

This instruction saves the minimum unsigned value of 8 elements of 16 bits from `qw` to `rd`.

Operation

```
min_value[15 :  0] = min(qw[15 :  0], qw[31 : 16], ..., qw[127:112])
rd[31 :  0] = {16'b0, min_value[15 :  0]}
```

Schedule

Operands	Use	Def
rd	-	2
qw	1	-

Format

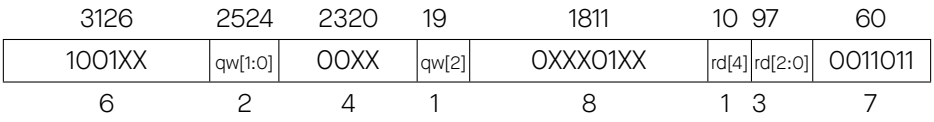


Figure 4.7-86. ESP.MIN.U16.A

4.7.2.87 ESP.MIN.U32.A

Syntax

ESP.MIN.U32.A qw,rd

Description

This instruction saves the minimum unsigned value of 4 elements of 32 bits from `qw` to `rd`.

Operation

```
min_value[31 :  0] = min(qw[31 :  0], qw[63 : 32], ..., qw[127:96])
rd[31 :  0] = min_value[31 :  0]
```

Schedule

Operands	Use	Def
rd	-	2
qw	1	-

Format

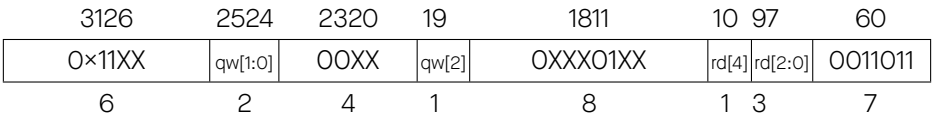


Figure 4.7-87. ESP.MIN.U32.A

4.7.2.88 ESP.MIN.U8.A

Syntax

ESP.MIN.U8.A qw,rd

Description

This instruction saves the minimum unsigned value of 16 elements of 8 bits from `qw` to `rd`.

Operation

```
    min_value[7 :  0] = min(qw[7 :  0], qw[16 : 8], ..., qw[127:120])
    rd[31 :  0] = {24{0}, min_value[7 :  0]}
```

Schedule

Operands	Use	Def
rd	-	2
qw	1	-

Format

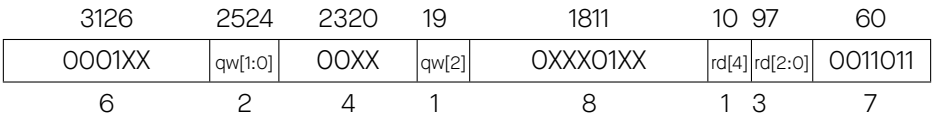


Figure 4.7-88. ESP.MIN.U8.A

4.7.2.89 ESP.VABS.16

Syntax

ESP.VABS.16 qv,qy

Description

This instruction takes the absolute value of the 8 16-bit data segments.

Operation

```
qv[ 15:  0] = (qy[ 7: 0] < 0) ? -qy[ 7: 0] : qy[ 7: 0]
qv[ 31: 16] = (qy[ 15: 16] < 0) ? -qy[ 15: 8] : qy[ 15: 8]
qv[ 47: 32] = (qy[ 47: 32] < 0) ? -qy[ 47: 16] : qy[ 47: 16]
qv[ 63: 48] = (qy[ 63: 48] < 0) ? -qy[ 63: 48] : qy[ 63: 48]
qv[ 79: 64] = (qy[ 79: 64] < 0) ? -qy[ 79: 64] : qy[ 79: 64]
qv[ 95: 80] = (qy[ 95: 80] < 0) ? -qy[ 95: 80] : qy[ 95: 80]
qv[111: 96] = (qy[111: 96] < 0) ? -qy[111: 96] : qy[111: 96]
qv[127:112] = (qy[127:112] < 0) ? -qy[127:112] : qy[127:112]
```

Schedule

Operands	Use	Def
qy	1	-
qv	-	1

Format

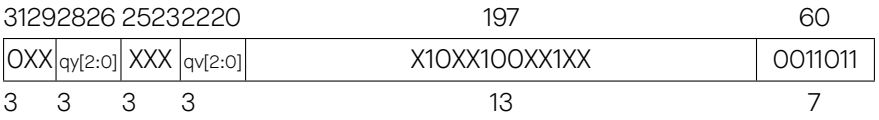


Figure 4.7-89. ESP.VABS.16

4.7.2.90 ESP.VABS.32

Syntax

ESP.VABS.32 qv,qy

Description

This instruction takes the absolute value of the 4 32-bit data segments.

Operation

```
qv[ 31:  0] = (qy[ 31: 0] < 0) ? -qy[ 31: 0] : qy[ 31: 0]
qv[ 63: 32] = (qy[ 63: 32] < 0) ? -qy[ 63: 32] : qy[ 63: 32]
qv[ 95: 64] = (qy[ 95: 64] < 0) ? -qy[ 95: 64] : qy[ 95: 64]
qv[127: 96] = (qy[127: 96] < 0) ? -qy[127: 96] : qy[127: 96]
```

Schedule

Operands	Use	Def
qy	1	-
qv	-	1

Format

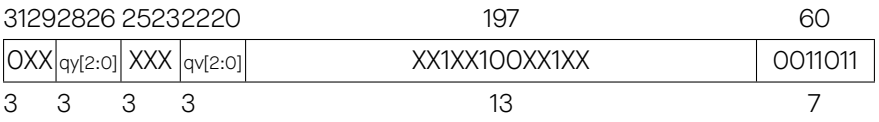


Figure 4.7-90. ESP.VABS.32

4.7.2.91 ESP.VABS.8

Syntax

ESP.VABS.8 qv,qy

Description

This instruction takes the absolute value of the 16 8-bit data segments.

Operation

```
qv[  7:  0] = (qy[ 7: 0] < 0) ? -qy[ 7: 0] : qy[ 7: 0]
qv[ 15:  8] = (qy[ 15: 8] < 0) ? -qy[ 15: 8] : qy[ 15: 8]
qv[ 23: 16] = (qy[ 23: 16] < 0) ? -qy[ 23: 16] : qy[ 23: 16]
qv[ 31: 24] = (qy[ 31: 24] < 0) ? -qy[ 31: 24] : qy[ 31: 24]
qv[ 39: 32] = (qy[ 39: 32] < 0) ? -qy[ 39: 32] : qy[ 39: 32]
qv[ 47: 40] = (qy[ 47: 40] < 0) ? -qy[ 47: 40] : qy[ 47: 40]
qv[ 55: 48] = (qy[ 55: 48] < 0) ? -qy[ 55: 48] : qy[ 55: 48]
qv[ 63: 56] = (qy[ 63: 56] < 0) ? -qy[ 63: 56] : qy[ 63: 56]
qv[ 71: 64] = (qy[ 71: 64] < 0) ? -qy[ 71: 64] : qy[ 71: 64]
qv[ 79: 72] = (qy[ 79: 72] < 0) ? -qy[ 79: 72] : qy[ 79: 72]
qv[ 87: 80] = (qy[ 87: 80] < 0) ? -qy[ 87: 80] : qy[ 87: 80]
qv[ 95: 88] = (qy[ 95: 88] < 0) ? -qy[ 95: 88] : qy[ 95: 88]
qv[103: 96] = (qy[103: 96] < 0) ? -qy[103: 96] : qy[103: 96]
qv[111:104] = (qy[111:104] < 0) ? -qy[111:104] : qy[111:104]
qv[119:112] = (qy[119:112] < 0) ? -qy[119:112] : qy[119:112]
qv[127:120] = (qy[127:120] < 0) ? -qy[127:120] : qy[127:120]
```

Schedule

Operands	Use	Def
qy	1	-
qv	-	1

Format

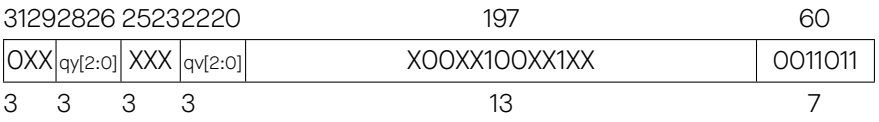


Figure 4.7-91. ESP.VABS.8

4.7.2.92 ESP.VADD.S16

Syntax

ESP.VADD.S16 qv,qx,qy,sat

Description

This instruction performs a vector signed addition on 16-bit data in the two registers qx and qy. Then, the 8 results obtained from the calculation are saturated, and the saturated results are written to the register qv.

Operation

```
qv[ 15:  0] = min(max(qx[ 15:  0] + qy[ 15:  0], -2^{15}), 2^{15}-1)
qv[ 31: 16] = min(max(qx[ 31: 16] + qy[ 31: 16], -2^{15}), 2^{15}-1)
...
qv[127:112] = min(max(qx[127:112] + qy[127:112], -2^{15}), 2^{15}-1)
```

Schedule

Operands	Use	Def
qx	1	-
qy	1	-
qv	-	1

Format

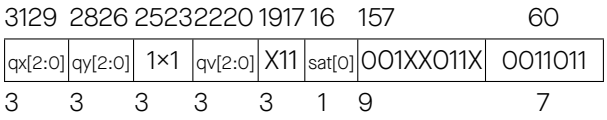


Figure 4.7-92. ESP.VADD.S16

4.7.2.93 ESP.VADD.S16.LD.INCP

Syntax

ESP.VADD.S16.LD.INCP qu,rs1,qv,qx,qy,sat

Description

This instruction performs a vector signed addition on 16-bit data in the two registers `qx` and `qy`. Then, the 8 results obtained from the calculation are saturated, and the saturated results are written to the register `qv`.

During the operation, 16 bytes are loaded from the memory to the register `qu`. After the access, the value in the register `rs1` is incremented by 16.

Operation

```
qv[ 15:  0] = min(max(qx[ 15:  0] + qy[ 15:  0], -2^{15}), 2^{15}-1)
qv[ 31: 16] = min(max(qx[ 31: 16] + qy[ 31: 16], -2^{15}), 2^{15}-1)
...

qu[127:0] = load128(rs1[31:0])
rs1[31:0] = rs1[31:0] + 16
```

Schedule

Operands	Use	Def
qx	1	-
qy	1	-
qv	-	1
qu	-	2
rs1	1	1

Format

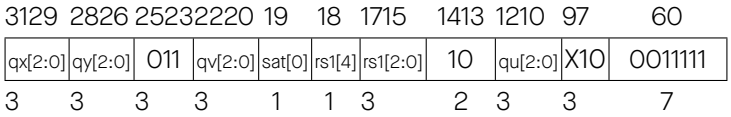


Figure 4.7-93. ESP.VADD.S16.LD.INCP

4.7.2.94 ESP.VADD.S16.ST.INCP

Syntax

ESP.VADD.S16.ST.INCP qu,rs1,qv,qx,qy,sat

Description

This instruction performs a vector signed addition on 16-bit data in the two registers `qx` and `qy`. Then, the 8 results obtained from the calculation are saturated, and the saturated results are written to the register `qv`.

During the operation, the value in the register `qu` is stored in memory. After the access, the value in the register `rs1` is incremented by 16.

Operation

```
qv[ 15:  0] = min(max(qx[ 15:  0] + qy[ 15:  0], -2^{15}), 2^{15}-1)
qv[ 31: 16] = min(max(qx[ 31: 16] + qy[ 31: 16], -2^{15}), 2^{15}-1)
...

qu[127:0] => store128(rs1[31:0])
rs1[31:0] = rs1[31:0] + 16
```

Schedule

Operands	Use	Def
qx	1	-
qy	1	-
qv	-	1
qu	2	-
rs1	1	1

Format

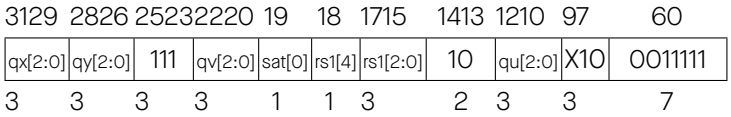


Figure 4.7-94. ESP.VADD.S16.ST.INCP

4.7.2.95 ESP.VADD.S32

Syntax

ESP.VADD.S32 qv,qx,qy,sat

Description

This instruction performs a vector signed addition on 32-bit data in the two registers qx and qy. Then, the 4 results obtained from the calculation are saturated, and the saturated results are written to the register qv.

Operation

```
qv[ 31:  0] = min(max(qx[ 31:  0] + qy[ 31:  0], -2^{31}), 2^{31}-1)
qv[ 63: 32] = min(max(qx[ 63: 32] + qy[ 63: 32], -2^{31}), 2^{31}-1)
...
```

Schedule

Operands	Use	Def
qx	1	-
qy	1	-
qv	-	1

Format

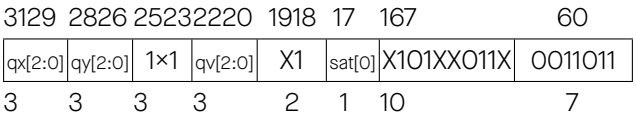


Figure 4.7-95. ESP.VADD.S32

4.7.2.96 ESP.VADD.S32.LD.INCP

Syntax

ESP.VADD.S32.LD.INCP qu,rs1,qv,qx,qy,sat

Description

This instruction performs a vector signed addition on 32-bit data in the two registers `qx` and `qy`. Then, the 4 results obtained from the calculation are saturated, and the saturated results are written to the register `qv`.

During the operation, 16 bytes are loaded from the memory to the register `qu`. After the access, the value in the register `rs1` is incremented by 16.

Operation

```
qv[ 31:  0] = min(max(qx[ 31:  0] + qy[ 31:  0], -2^{31}), 2^{31}-1)
qv[ 63: 32] = min(max(qx[ 63: 32] + qy[ 63: 32], -2^{31}), 2^{31}-1)
...

qu[127:0] = load128(rs1[31:0])
rs1[31:0] = rs1[31:0] + 16
```

Schedule

Operands	Use	Def
qx	1	-
qy	1	-
qv	-	1
qu	-	2
rs1	1	1

Format

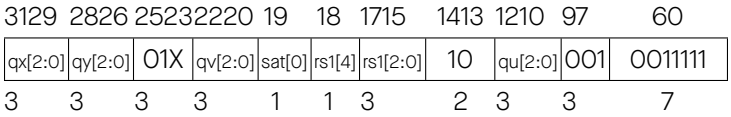


Figure 4.7-96. ESP.VADD.S32.LD.INCP

4.7.2.97 ESP.VADD.S32.ST.INCP

Syntax

ESP.VADD.S32.ST.INCP qu,rs1,qv,qx,qy,sat

Description

This instruction performs a vector signed addition on 32-bit data in the two registers `qx` and `qy`. Then, the 4 results obtained from the calculation are saturated, and the saturated results are written to the register `qu`.

During the operation, the value in the register `qu` is stored in memory. After the access, the value in the register `rs1` is incremented by 16.

Operation

```
qv[ 31:  0] = min(max(qx[ 31:  0] + qy[ 31:  0], -2^{31}), 2^{31}-1)
qv[ 63: 32] = min(max(qx[ 63: 32] + qy[ 63: 32], -2^{31}), 2^{31}-1)
...

qu[127:0] => store128(rs1[31:0])
rs1[31:0] = rs1[31:0] + 16
```

Schedule

Operands	Use	Def
qx	1	-
qy	1	-
qv	-	1
qu	2	-
rs1	1	1

Format

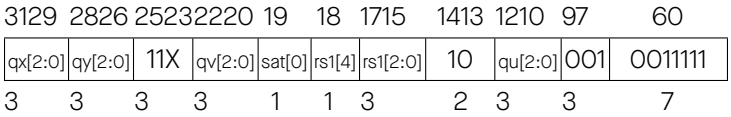


Figure 4.7-97. ESP.VADD.S32.ST.INCP

4.7.2.98 ESP.VADD.S8

Syntax

ESP.VADD.S8 qv,qx,qy,sat

Description

This instruction performs a vector signed addition on 8-bit data in the two registers `qx` and `qy`. Then, the 16 results obtained from the calculation are saturated, and the saturated results are written to the register `qv`.

Operation

```
qv[  7:  0] = min(max(qx[  7:  0] + qy[  7:  0], -2^{7}), 2^{7}-1)
qv[ 15:  8] = min(max(qx[ 15:  8] + qy[ 15:  8], -2^{7}), 2^{7}-1)
...
qv[127:120] = min(max(qx[127:120] + qy[127:120], -2^{7}), 2^{7}-1)
```

Schedule

Operands	Use	Def
qx	1	-
qy	1	-
qv	-	1

Format

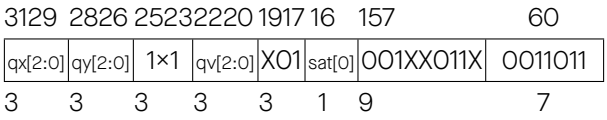


Figure 4.7-98. ESP.VADD.S8

4.7.2.99 ESP.VADD.S8.LD.INCP

Syntax

ESP.VADD.S8.LD.INCP qu,rs1,qv,qx,qy,sat

Description

This instruction performs a vector signed addition on 8-bit data in the two registers `qx` and `qy`. Then, the 16 results obtained from the calculation are saturated, and the saturated results are written to the register `qa`.

During the operation, 16 bytes are loaded from the memory to the register `qu`. After the access, the value in the register `rs1` is incremented by 16.

Operation

```
qv[  7:  0] = min(max(qx[  7:  0] + qy[  7:  0], -2^{7}), 2^{7}-1)
qv[ 15:  8] = min(max(qx[ 15:  8] + qy[ 15:  8], -2^{7}), 2^{7}-1)
...

qu[127:0] = load128(rs1[31:0])
rs1[31:0] = rs1[31:0] + 16
```

Schedule

Operands	Use	Def
qx	1	-
qy	1	-
qv	-	1
qu	-	2
rs1	1	1

Format

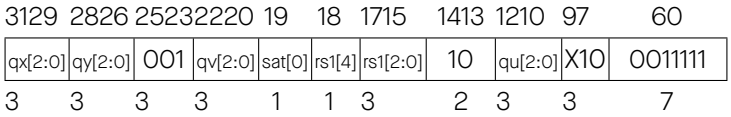


Figure 4.7-99. ESP.VADD.S8.LD.INCP

4.7.2.100 ESP.VADD.S8.ST.INCP

Syntax

ESP.VADD.S8.ST.INCP qu,rs1,qv,qx,qy,sat

Description

This instruction performs a vector signed addition on 8-bit data in the two registers `qx` and `qy`. Then, the 16 results obtained from the calculation are saturated, and the saturated results are written to the register `qa`.

During the operation, the value in the register `qu` is stored in memory. After the access, the value in the register `rs1` is incremented by 16.

Operation

```
qv[  7:  0] = min(max(qx[  7:  0] + qy[  7:  0], -2^{7}), 2^{7}-1)
qv[ 15:  8] = min(max(qx[ 15:  8] + qy[ 15:  8], -2^{7}), 2^{7}-1)
...

qu[127:0] => store128(rs1[31:0])
rs1 [31:0] = rs1[31:0] + 16
```

Schedule

Operands	Use	Def
qx	1	-
qy	1	-
qv	-	1
qu	2	-
rs1	1	1

Format

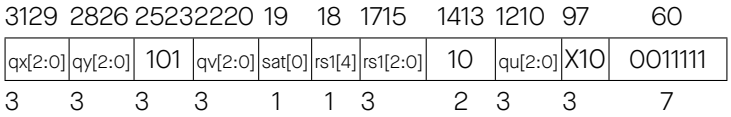


Figure 4.7-100. ESP.VADD.S8.ST.INCP

4.7.2.101 ESP.VADD.U16

Syntax

ESP.VADD.U16 qv,qx,qy,sat

Description

This instruction performs a vector unsigned addition on 16-bit data in the two registers qx and qy. Then, the 8 results obtained from the calculation are saturated, and the saturated results are written to the register qv.

Operation

```
qv[ 15:  0] = min(qx[ 15:  0] + qy[ 15:  0], 2^{16}-1)
qv[ 31: 16] = min(qx[ 31: 16] + qy[ 31: 16], 2^{16}-1)
...
qv[127:112] = min(qx[127:112] + qy[127:112], 2^{16}-1)
```

Schedule

Operands	Use	Def
qx	1	-
qy	1	-
qv	-	1

Format

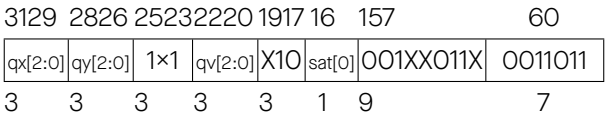


Figure 4.7-101. ESP.VADD.U16

4.7.2.102 ESP.VADD.U16.LD.INCP

Syntax

ESP.VADD.U16.LD.INCP qu,rs1,qv,qx,qy,sat

Description

This instruction performs a vector unsigned addition on 16-bit data in the two registers `qx` and `qy`. Then, the 8 results obtained from the calculation are saturated, and the saturated results are written to the register `qv`.

During the operation, the value in the register `qu` is stored in memory. After the access, the value in the register `rs1` is incremented by 16.

Operation

```
qv[ 15:  0] = min(qx[ 15:  0] + qy[ 15:  0], 2^{16}-1)
qv[ 31: 16] = min(qx[ 31: 16] + qy[ 31: 16], 2^{16}-1)
...
qv[127:112] = min(qx[127:112] + qy[127:112], 2^{16}-1)

 qu[127:0] = load128(rs1[31:0])
rs1[31:0] = rs1[31:0] + 16
```

Schedule

Operands	Use	Def
qx	1	-
qy	1	-
qv	-	1
qu	-	2
rs1	1	1

Format

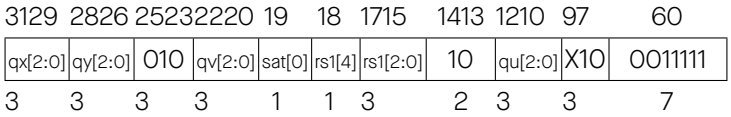


Figure 4.7-102. ESP.VADD.U16.LD.INCP

4.7.2.103 ESP.VADD.U16.ST.INCP

Syntax

ESP.VADD.U16.ST.INCP qu,rs1,qv,qx,qy,sat

Description

This instruction performs a vector unsigned addition on 16-bit data in the two registers `qx` and `qy`. Then, the 8 results obtained from the calculation are saturated, and the saturated results are written to the register `qv`.

During the operation, the value in the register `qu` is stored to memory. After the access, the value in the register `rs1` is incremented by 16.

Operation

```
qv[ 15:  0] = min(qx[ 15:  0] + qy[ 15:  0], 2^{16}-1)
qv[ 31: 16] = min(qx[ 31: 16] + qy[ 31: 16], 2^{16}-1)
...
qv[127:112] = min(qx[127:112] + qy[127:112], 2^{16}-1)

qu[127:0] => store128(rs1[31:0])
rs1[31:0] = rs1[31:0] + 16
```

Schedule

Operands	Use	Def
qx	1	-
qy	1	-
qv	-	1
qu	2	-
rs1	1	1

Format

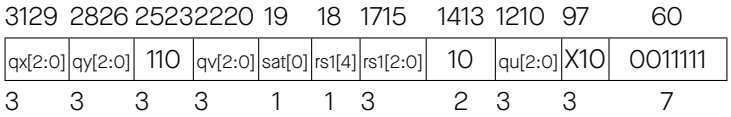


Figure 4.7-103. ESP.VADD.U16.ST.INCP

4.7.2.104 ESP.VADD.U32

Syntax

ESP.VADD.U32 qv,qx,qy,sat

Description

This instruction performs a vector unsigned addition on 32-bit data in the two registers qx and qy. Then, the 4 results obtained from the calculation are saturated, and the saturated results are written to the register qv.

Operation

```
qv[ 31:  0] = min(qx[ 31:  0] + qy[ 31:  0], 2^{32}-1)
qv[ 63: 32] = min(qx[ 63: 32] + qy[ 63: 32], 2^{32}-1)
...
```

Schedule

Operands	Use	Def
qx	1	-
qy	1	-
qv	-	1

Format

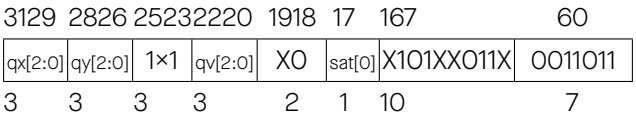


Figure 4.7-104. ESP.VADD.U32

4.7.2.105 ESP.VADD.U32.LD.INCP

Syntax

ESP.VADD.U32.LD.INCP qu,rs1,qv,qx,qy,sat

Description

This instruction performs a vector unsigned addition on 32-bit data in the two registers `qx` and `qy`. Then, the 4 results obtained from the calculation are saturated, and the saturated results are written to the register `qv`.

During the operation, 16 bytes are loaded from the memory to the register `qu`. After the access, the value in the register `rs1` is incremented by 16.

Operation

```
qv[ 31:  0] = min(qx[ 31:  0] + qy[ 31:  0], 2^{32}-1)
qv[ 63: 32] = min(qx[ 63: 32] + qy[ 63: 32], 2^{32}-1)
...

qu[127:0] = load128(rs1[31:0])
rs1[31:0] = rs1[31:0] + 16
```

Schedule

Operands	Use	Def
qx	1	-
qy	1	-
qv	-	1
qu	-	2
rs1	1	1

Format

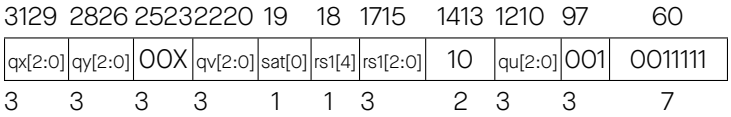


Figure 4.7-105. ESP.VADD.U32.LD.INCP

4.7.2.106 ESP.VADD.U32.ST.INCP

Syntax

ESP.VADD.U32.ST.INCP qu,rs1,qv,qx,qy,sat

Description

This instruction performs a vector unsigned addition on 32-bit data in the two registers `qx` and `qy`. Then, the 4 results obtained from the calculation are saturated, and the saturated results are written to the register `qv`.

During the operation, the value in the register `qu` is stored in memory. After the access, the value in the register `rs1` is incremented by 16.

Operation

```
qv[ 31:  0] = min(qx[ 31:  0] + qy[ 31:  0], 2^{32}-1)
qv[ 63: 32] = min(qx[ 63: 32] + qy[ 63: 32], 2^{32}-1)
...
qu[127:0] => store128(rs1[31:0])
rs1[31:0] = rs1[31:0] + 16
```

Schedule

Operands	Use	Def
qx	1	-
qy	1	-
qv	-	1
qu	2	-
rs1	1	1

Format

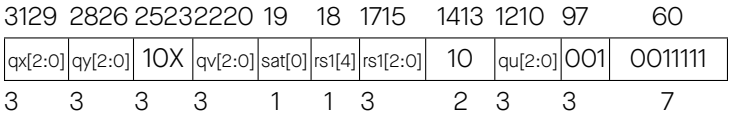


Figure 4.7-106. ESP.VADD.U32.ST.INCP

4.7.2.107 ESP.VADD.U8

Syntax

ESP.VADD.U8 qv,qx,qy,sat

Description

This instruction performs a vector unsigned addition on 8-bit data in the two registers `qx` and `qy`. Then, the 16 results obtained from the calculation are saturated, and the saturated results are written to the register `qv`.

Operation

```
qv[  7:  0] = min(qx[  7:  0] + qy[  7:  0], 2^{8}-1)
qv[ 15:  8] = min(qx[ 15:  8] + qy[ 15:  8], 2^{8}-1)
...
qv[127:120] = min(qx[127:120] + qy[127:120], 2^{8}-1)
```

Schedule

Operands	Use	Def
qx	1	-
qy	1	-
qv	-	1

Format

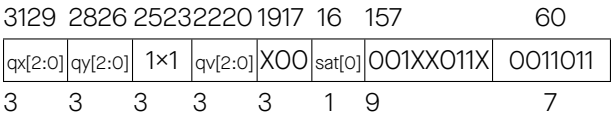


Figure 4.7-107. ESP.VADD.U8

4.7.2.108 ESP.VADD.U8.LD.INCP

Syntax

ESP.VADD.U8.LD.INCP qu,rs1,qv,qx,qy,sat

Description

This instruction performs a vector unsigned addition on 8-bit data in the two registers `qx` and `qy`. Then, the 16 results obtained from the calculation are saturated, and the saturated results are written to the register `qv`.

During the operation, 16 bytes are loaded from the memory to the register `qu`. After the access, the value in the register `rs1` is incremented by 16.

Operation

```
qv[  7:  0] = min(qx[  7:  0] + qy[  7:  0], 2^{8}-1)
qv[ 15:  8] = min(qx[ 15:  8] + qy[ 15:  8], 2^{8}-1)
...

qu[127:0] = load128(rs1[31:0])
rs1[31:0] = rs1[31:0] + 16
```

Schedule

Operands	Use	Def
qx	1	-
qy	1	-
qv	-	1
qu	-	2
rs1	1	1

Format

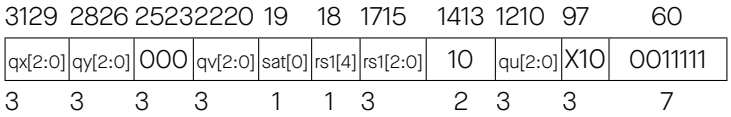


Figure 4.7-108. ESP.VADD.U8.LD.INCP

4.7.2.109 ESP.VADD.U8.ST.INCP

Syntax

ESP.VADD.U8.ST.INCP qu,rs1,qv,qx,qy,sat

Description

This instruction performs a vector unsigned addition on 8-bit data in the two registers `qx` and `qy`. Then, the 16 results obtained from the calculation are saturated, and the saturated results are written to the register `qv`.

During the operation, the value in the register `qu` is stored in memory. After the access, the value in the register `rs1` is incremented by 16.

Operation

```
qv[  7:  0] = min(qx[  7:  0] + qy[  7:  0], 2^{8}-1)
qv[ 15:  8] = min(qx[ 15:  8] + qy[ 15:  8], 2^{8}-1)
...

qu[127:0] => store128(rs1[31:0])
rs1 [31:0] = rs1[31:0] + 16
```

Schedule

Operands	Use	Def
qx	1	-
qy	1	-
qv	-	1
qu	2	-
rs1	1	1

Format

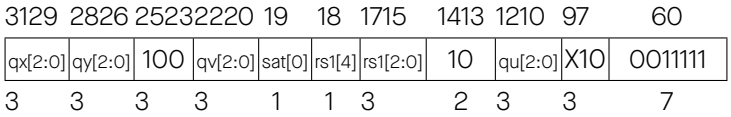


Figure 4.7-109. ESP.VADD.U8.ST.INCP

4.7.2.110 ESP.VCLAMP.S16

Syntax

ESP.VCLAMP.S16 qz,qx,sel16

Description

This instruction performs clamping for 16-bit signed data.

Operation

```
    qz[ 15:  0] = min( max(qx[ 15:  0], -2^{sel16}), 2^{sel16}-1 )
    qz[ 31: 16] = min( max(qx[ 31: 16], -2^{sel16}), 2^{sel16}-1 )
    qz[ 47: 32] = min( max(qx[ 47: 32], -2^{sel16}), 2^{sel16}-1 )
    qz[ 63: 48] = min( max(qx[ 63: 48], -2^{sel16}), 2^{sel16}-1 )
    qz[ 79: 64] = min( max(qx[ 79: 64], -2^{sel16}), 2^{sel16}-1 )
    qz[ 95: 80] = min( max(qx[ 95: 80], -2^{sel16}), 2^{sel16}-1 )
    qz[111: 96] = min( max(qx[111: 96], -2^{sel16}), 2^{sel16}-1 )
    qz[127:112] = min( max(qx[127:112], -2^{sel16}), 2^{sel16}-1 )
```

Schedule

Operands	Use	Def
qx	1	-
qy	1	-
qz	-	1

Format

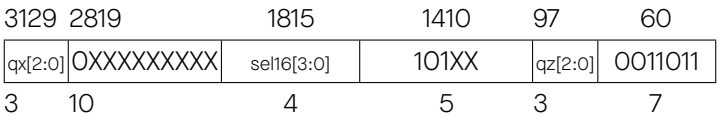


Figure 4.7-110. ESP.VCLAMP.S16

4.7.2.111 ESP.VMAX.S16

Syntax

ESP.VMAX.S16 qz,qx,qy

Description

This instruction compares the numerical values of the eight 16-bit signed vector data segments in the registers qx and qy. The data segment with the larger value is written into the corresponding 16-bit data segment in the register qz.

Operation

```
qz[ 15:  0] = (qx[ 15:  0]>=qy[ 15:  0]) ? qx[ 15:  0] : qy[ 15:  0]
qz[ 31: 16] = (qx[ 31: 16]>=qy[ 31: 16]) ? qx[ 31: 16] : qy[ 31: 16]
...
qz[127:112] = (qx[127:112]>=qy[127:112]) ? qx[127:112] : qy[127:112]
```

Schedule

Operands	Use	Def
qx	1	-
qy	1	-
qz	-	1

Format

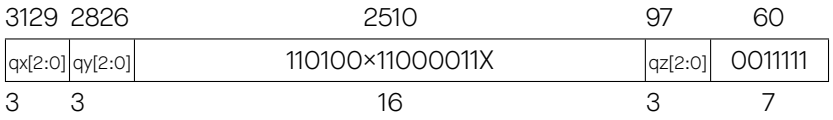


Figure 4.7-111. ESP.VMAX.S16

4.7.2.112 ESP.VMAX.S16.LD.INCP

Syntax

ESP.VMAX.S16.LD.INCP qu,rs1,qz,qx,qy

Description

This instruction compares the numerical values of the eight 16-bit signed vector data segments in the registers `qx` and `qy`. The data segment with the larger value is written into the corresponding 16-bit data segment in the register `qz`.

During the operation, 16 bytes are loaded from the memory into the register `qu`. After the access, the value in the register `rs1` is incremented by 16.

Operation

```
qz[ 15:  0] = (qx[ 15:  0]>=qy[ 15:  0]) ? qx[ 15:  0] : qy[ 15:  0]
qz[ 31: 16] = (qx[ 31: 16]>=qy[ 31: 16]) ? qx[ 31: 16] : qy[ 31: 16]
...
qz[127:112] = (qx[127:112]>=qy[127:112]) ? qx[127:112] : qy[127:112]

qu[127:0] = load128(rs1[31:0])
rs1[31:0] = rs1[31:0] + 16
```

Schedule

Operands	Use	Def
qx	1	-
qy	1	-
qz	-	1
qu	-	2
rs1	1	1

Format

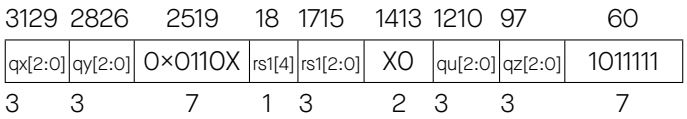


Figure 4.7-112. ESP.VMAX.S16.LD.INCP

4.7.2.113 ESP.VMAX.S16.ST.INCP

Syntax

ESP.VMAX.S16.ST.INCP qu,rs1,qz,qx,qy

Description

This instruction compares the numerical values of the eight 16-bit signed vector data segments in the registers `qx` and `qy`. The data segment with the larger value is written into the corresponding 16-bit data segment in the register `qz`.

During the operation, the value in the register `qu` is stored in memory. After the access, the value in the register `rs1` is incremented by 16.

Operation

```
qz[ 15:  0] = (qx[ 15:  0]>=qy[ 15:  0]) ? qx[ 15:  0] : qy[ 15:  0]
qz[ 31: 16] = (qx[ 31: 16]>=qy[ 31: 16]) ? qx[ 31: 16] : qy[ 31: 16]
...
qz[127:112] = (qx[127:112]>=qy[127:112]) ? qx[127:112] : qy[127:112]

qu[127:0] => store128(rs1[31:0])
rs1 [31:0] = rs1[31:0] + 16
```

Schedule

Operands	Use	Def
qx	1	-
qy	1	-
qz	-	1
qu	2	-
rs1	1	1

Format

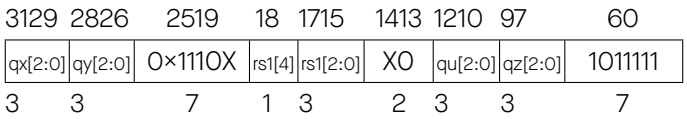


Figure 4.7-113. ESP.VMAX.S16.ST.INCP

4.7.2.114 ESP.VMAX.S32

Syntax

ESP.VMAX.S32 qz,qx,qy

Description

This instruction compares the numerical values of the four 32-bit signed vector data segments in the registers qx and qy. The data segment with the larger value is written into the corresponding 32-bit data segment in the register qz.

Operation

```
qz[ 31:  0] = (qx[ 31:  0]>=qy[ 31:  0]) ? qx[ 31:  0] : qy[ 31:  0]
qz[ 63: 32] = (qx[ 63: 32]>=qy[ 63: 32]) ? qx[ 63: 32] : qy[ 63: 32]
...
qz[127: 96] = (qx[127: 96]>=qy[127: 96]) ? qx[127: 96] : qy[127: 96]
```

Schedule

Operands	Use	Def
qx	1	-
qy	1	-
qz	-	1

Format

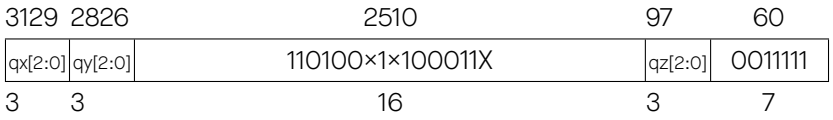


Figure 4.7-114. ESP.VMAX.S32

4.7.2.115 ESP.VMAX.S32.LD.INCP

Syntax

ESP.VMAX.S32.LD.INCP qu,rs1,qz,qx,qy

Description

This instruction compares the numerical values of the four 32-bit signed vector data segments in registers `qx` and `qy`. The data segment with the larger value is written into the corresponding 32-bit data segment in register `qz`.

During the operation, 16 bytes are loaded from the memory into register `qu`. After the access, the value in register `rs1` is incremented by 16.

Operation

```
qz[ 31:  0] = (qx[ 31:  0]>=qy[ 31:  0]) ? qx[ 31:  0] : qy[ 31:  0]
qz[ 63: 32] = (qx[ 63: 32]>=qy[ 63: 32]) ? qx[ 63: 32] : qy[ 63: 32]
...
qz[127: 96] = (qx[127: 96]>=qy[127: 96]) ? qx[127: 96] : qy[127: 96]

 qu[127:0] = load128(rs1[31:0])
rs1[31:0] = rs1[31:0] + 16
```

Schedule

Operands	Use	Def
qx	1	-
qy	1	-
qz	-	1
qu	-	2
rs1	1	1

Format

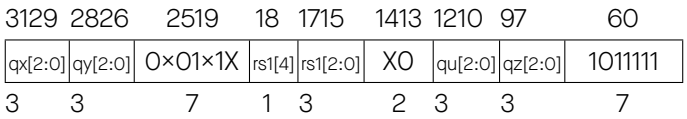


Figure 4.7-115. ESP.VMAX.S32.LD.INCP

4.7.2.116 ESP.VMAX.S32.ST.INCP

Syntax

ESP.VMAX.S32.ST.INCP qu,rs1,qz,qx,qy

Description

This instruction compares the numerical values of the four 32-bit signed vector data segments in the registers `qx` and `qy`. The data segment with the larger value is written into the corresponding 32-bit data segment in the register `qz`.

During the operation, the value in the register `qu` is stored in memory. After the access, the value in the register `rs1` is incremented by 16.

Operation

```
qz[ 31:  0] = (qx[ 31:  0]>=qy[ 31:  0]) ? qx[ 31:  0] : qy[ 31:  0]
qz[ 63: 32] = (qx[ 63: 32]>=qy[ 63: 32]) ? qx[ 63: 32] : qy[ 63: 32]
...
qz[127: 96] = (qx[127: 96]>=qy[127: 96]) ? qx[127: 96] : qy[127: 96]

qu[127:0] => store128(rs1[31:0])
rs1 [31:0] = rs1[31:0] + 16
```

Schedule

Operands	Use	Def
qx	1	-
qy	1	-
qz	-	1
qu	2	-
rs1	1	1

Format

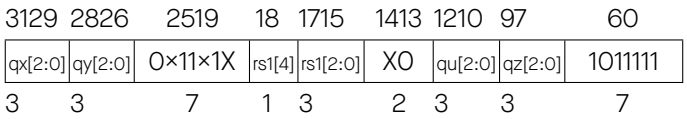


Figure 4.7-116. ESP.VMAX.S32.ST.INCP

4.7.2.117 ESP.VMAX.S8

Syntax

ESP.VMAX.S8 qz,qx,qy

Description

This instruction compares the numerical values of the 16 8-bit signed vector data segments in registers qx and qy. The data segment with the larger value is written into the corresponding 8-bit data segment in the register qz.

Operation

```
qz[  7:  0] = (qx[  7:  0]>=qy[  7:  0]) ? qx[  7:  0] : qy[  7:  0]
qz[ 15:  8] = (qx[ 15:  8]>=qy[ 15:  8]) ? qx[ 15:  8] : qy[ 15:  8]
...
qz[127:120] = (qx[127:120]>=qy[127:120]) ? qx[127:120] : qy[127:120]
```

Schedule

Operands	Use	Def
qx	1	-
qy	1	-
qz	-	1

Format

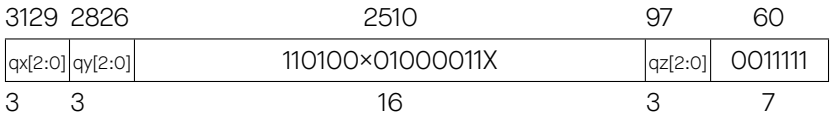


Figure 4.7-117. ESP.VMAX.S8

4.7.2.118 ESP.VMAX.S8.LD.INCP

Syntax

ESP.VMAX.S8.LD.INCP qu,rs1,qz,qx,qy

Description

This instruction compares the numerical values of the 16 8-bit signed vector data segments in registers `qx` and `qy`. The data segment with the larger value is written into the corresponding 8-bit data segment in register `qz`.

During the operation, 16 bytes are loaded from the memory into register `qu`. After the access, the value in register `rs1` is incremented by 16.

Operation

```
qz[  7:  0] = (qx[  7:  0]>=qy[  7:  0]) ? qx[  7:  0] : qy[  7:  0]
qz[ 15:  8] = (qx[ 15:  8]>=qy[ 15:  8]) ? qx[ 15:  8] : qy[ 15:  8]
...
qz[127:120] = (qx[127:120]>=qy[127:120]) ? qx[127:120] : qy[127:120]

qu[127:0] = load128(rs1[31:0])
rs1[31:0] = rs1[31:0] + 16
```

Schedule

Operands	Use	Def
qx	1	-
qy	1	-
qz	-	1
qu	-	2
rs1	1	1

Format

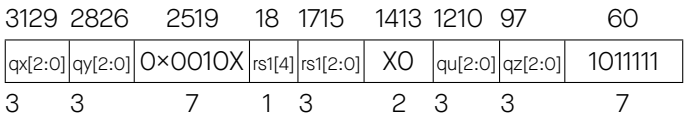


Figure 4.7-118. ESP.VMAX.S8.LD.INCP

4.7.2.119 ESP.VMAX.S8.ST.INCP

Syntax

ESP.VMAX.S8.ST.INCP qu,rs1,qz,qx,qy

Description

This instruction compares the numerical values of the 16 8-bit vector data segments in registers `qx` and `qy`. The data segment with the larger value is written into the corresponding 8-bit data segment in register `qz`.

During the operation, the value in register `qu` is stored to memory. After the access, the value in register `rs1` is incremented by 16.

Operation

```
qz[  7:  0] = (qx[  7:  0]>=qy[  7:  0]) ? qx[  7:  0] : qy[  7:  0]
qz[ 15:  8] = (qx[ 15:  8]>=qy[ 15:  8]) ? qx[ 15:  8] : qy[ 15:  8]
...
qz[127:120] = (qx[127:120]>=qy[127:120]) ? qx[127:120] : qy[127:120]

qu[127:0] => store128(rs1[31:0])
rs1 [31:0] = rs1[31:0] + 16
```

Schedule

Operands	Use	Def
qx	1	-
qy	1	-
qz	-	1
qu	2	-
rs1	1	1

Format

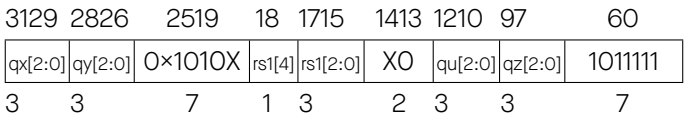


Figure 4.7-119. ESP.VMAX.S8.ST.INCP

4.7.2.120 ESP.VMAX.U16

Syntax

ESP.VMAX.U16 qz,qx,qy

Description

This instruction compares the numerical values of the eight 16-bit unsigned vector data segments in the registers qx and qy. The data segment with the larger value is written into the corresponding 16-bit data segment in the register qz.

Operation

```
qz[ 15:  0] = (qx[ 15:  0]>=qy[ 15:  0]) ? qx[ 15:  0] : qy[ 15:  0]
qz[ 31: 16] = (qx[ 31: 16]>=qy[ 31: 16]) ? qx[ 31: 16] : qy[ 31: 16]
...
qz[127:112] = (qx[127:112]>=qy[127:112]) ? qx[127:112] : qy[127:112]
```

Schedule

Operands	Use	Def
qx	1	-
qy	1	-
qz	-	1

Format

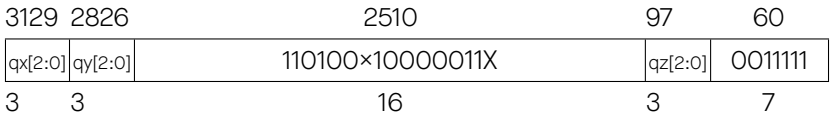


Figure 4.7-120. ESP.VMAX.U16

4.7.2.121 ESP.VMAX.U16.LD.INCP

Syntax

ESP.VMAX.U16.LD.INCP qu,rs1,qz,qx,qy

Description

This instruction compares the numerical values of the eight 16-bit unsigned vector data segments in the registers `qx` and `qy`. The data segment with the larger value is written into the corresponding 16-bit data segment in the register `qz`.

During the operation, 16 bytes are loaded from the memory into the register `qu`. After the access, the value in the register `rs1` is incremented by 16.

Operation

```
qz[ 15:  0] = (qx[ 15:  0]>=qy[ 15:  0]) ? qx[ 15:  0] : qy[ 15:  0]
qz[ 31: 16] = (qx[ 31: 16]>=qy[ 31: 16]) ? qx[ 31: 16] : qy[ 31: 16]
...
qz[127:112] = (qx[127:112]>=qy[127:112]) ? qx[127:112] : qy[127:112]

qu[127:0] = load128(rs1[31:0])
rs1[31:0] = rs1[31:0] + 16
```

Schedule

Operands	Use	Def
qx	1	-
qy	1	-
qz	-	1
qu	-	2
rs1	1	1

Format

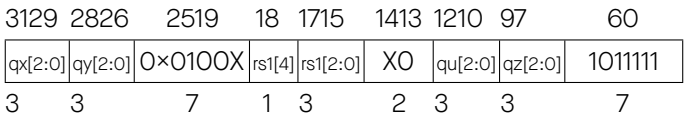


Figure 4.7-121. ESP.VMAX.U16.LD.INCP

4.7.2.122 ESP.VMAX.U16.ST.INCP

Syntax

ESP.VMAX.U16.ST.INCP qu,rs1,qz,qx,qy

Description

This instruction compares the numerical values of the eight 16-bit unsigned vector data segments in the registers `qx` and `qy`. The data segment with the larger value is written into the corresponding 16-bit data segment in the register `qz`.

During the operation, the value in the register `qu` is stored in memory. After the access, the value in the register `rs1` is incremented by 16.

Operation

```
qz[ 15:  0] = (qx[ 15:  0]>=qy[ 15:  0]) ? qx[ 15:  0] : qy[ 15:  0]
qz[ 31: 16] = (qx[ 31: 16]>=qy[ 31: 16]) ? qx[ 31: 16] : qy[ 31: 16]
...
qz[127:112] = (qx[127:112]>=qy[127:112]) ? qx[127:112] : qy[127:112]

qu[127:0] => store128(rs1[31:0])
rs1 [31:0] = rs1[31:0] + 16
```

Schedule

Operands	Use	Def
qx	1	-
qy	1	-
qz	-	1
qu	2	-
rs1	1	1

Format

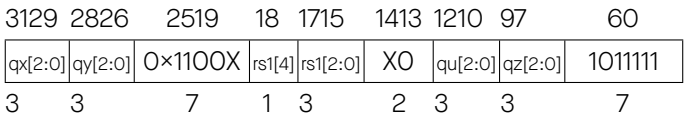


Figure 4.7-122. ESP.VMAX.U16.ST.INCP

4.7.2.123 ESP.VMAX.U32

Syntax

ESP.VMAX.U32 qz,qx,qy

Description

This instruction compares the numerical values of the four 32-bit unsigned vector data segments in the registers qx and qy. The data segment with the larger value is written into the corresponding 32-bit data segment in the register qz.

Operation

```
qz[ 31:  0] = (qx[ 31:  0]>=qy[ 31:  0]) ? qx[ 31:  0] : qy[ 31:  0]
qz[ 63: 32] = (qx[ 63: 32]>=qy[ 63: 32]) ? qx[ 63: 32] : qy[ 63: 32]
...
qz[127: 96] = (qx[127: 96]>=qy[127: 96]) ? qx[127: 96] : qy[127: 96]
```

Schedule

Operands	Use	Def
qx	1	-
qy	1	-
qz	-	1

Format

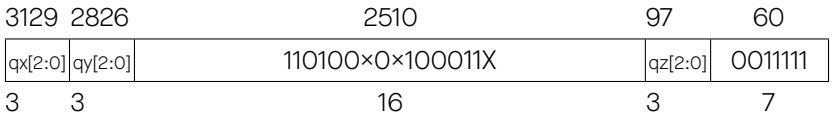


Figure 4.7-123. ESP.VMAX.U32

4.7.2.124 ESP.VMAX.U32.LD.INCP

Syntax

ESP.VMAX.U32.LD.INCP qu,rs1,qz,qx,qy

Description

This instruction compares the numerical values of the four 32-bit unsigned vector data segments in registers `qx` and `qy`. The data segment with the larger value is written into the corresponding 32-bit data segment in register `qz`.

During the operation, 16 bytes are loaded from the memory into register `qu`. After the access, the value in register `rs1` is incremented by 16.

Operation

```
qz[ 31:  0] = (qx[ 31:  0]>=qy[ 31:  0]) ? qx[ 31:  0] : qy[ 31:  0]
qz[ 63: 32] = (qx[ 63: 32]>=qy[ 63: 32]) ? qx[ 63: 32] : qy[ 63: 32]
...
qz[127: 96] = (qx[127: 96]>=qy[127: 96]) ? qx[127: 96] : qy[127: 96]

qu[127:0] = load128(rs1[31:0])
rs1[31:0] = rs1[31:0] + 16
```

Schedule

Operands	Use	Def
qx	1	-
qy	1	-
qz	-	1
qu	-	2
rs1	1	1

Format

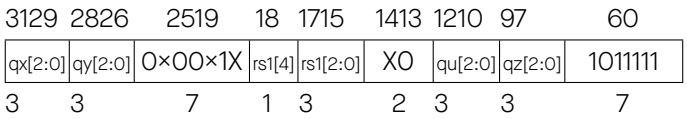


Figure 4.7-124. ESP.VMAX.U32.LD.INCP

4.7.2.125 ESP.VMAX.U32.ST.INCP

Syntax

ESP.VMAX.U32.ST.INCP qu,rs1,qz,qx,qy

Description

This instruction compares the numerical values of the four 32-bit unsigned vector data segments in the registers `qx` and `qy`. The data segment with the larger value is written into the corresponding 32-bit data segment in the register `qz`.

During the operation, the value in the register `qu` is stored in memory. After the access, the value in the register `rs1` is incremented by 16.

Operation

```
qz[ 31:  0] = (qx[ 31:  0]>=qy[ 31:  0]) ? qx[ 31:  0] : qy[ 31:  0]
qz[ 63: 32] = (qx[ 63: 32]>=qy[ 63: 32]) ? qx[ 63: 32] : qy[ 63: 32]
...
qz[127: 96] = (qx[127: 96]>=qy[127: 96]) ? qx[127: 96] : qy[127: 96]

qu[127:0] => store128(rs1[31:0])
rs1 [31:0] = rs1[31:0] + 16
```

Schedule

Operands	Use	Def
qx	1	-
qy	1	-
qz	-	1
qu	2	-
rs1	1	1

Format

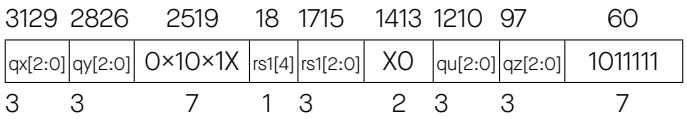


Figure 4.7-125. ESP.VMAX.U32.ST.INCP

4.7.2.126 ESP.VMAX.U8

Syntax

ESP.VMAX.U8 qz,qx,qy

Description

This instruction compares the numerical values of the 16 8-bit unsigned vector data segments in the registers qx and qy. The data segment with the larger value is written into the corresponding 8-bit data segment in the register qz.

Operation

```
qz[  7:  0] = (qx[  7:  0]>=qy[  7:  0]) ? qx[  7:  0] : qy[  7:  0]
qz[ 15:  8] = (qx[ 15:  8]>=qy[ 15:  8]) ? qx[ 15:  8] : qy[ 15:  8]
...
qz[127:120] = (qx[127:120]>=qy[127:120]) ? qx[127:120] : qy[127:120]
```

Schedule

Operands	Use	Def
qx	1	-
qy	1	-
qz	-	1

Format

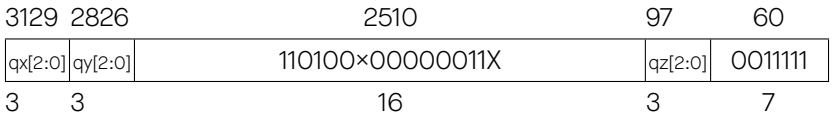


Figure 4.7-126. ESP.VMAX.U8

4.7.2.127 ESP.VMAX.U8.LD.INCP

Syntax

ESP.VMAX.U8.LD.INCP qu,rs1,qz,qx,qy

Description

This instruction compares the numerical values of the 16 8-bit unsigned vector data segments in registers `qx` and `qy`. The data segment with the larger value is written into the corresponding 8-bit data segment in register `qz`.

During the operation, 16 bytes are loaded from the memory into register `qu`. After the access, the value in register `rs1` is incremented by 16.

Operation

```
qz[  7:  0] = (qx[  7:  0]>=qy[  7:  0]) ? qx[  7:  0] : qy[  7:  0]
qz[ 15:  8] = (qx[ 15:  8]>=qy[ 15:  8]) ? qx[ 15:  8] : qy[ 15:  8]
...
qz[127:120] = (qx[127:120]>=qy[127:120]) ? qx[127:120] : qy[127:120]

qu[127:0] = load128(rs1[31:0])
rs1[31:0] = rs1[31:0] + 16
```

Schedule

Operands	Use	Def
qx	1	-
qy	1	-
qz	-	1
qu	-	2
rs1	1	1

Format

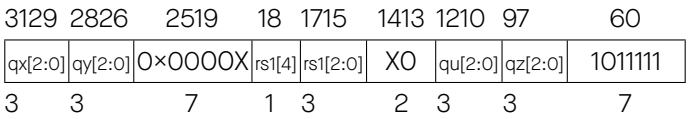


Figure 4.7-127. ESP.VMAX.U8.LD.INCP

4.7.2.128 ESP.VMAX.U8.ST.INCP

Syntax

ESP.VMAX.U8.ST.INCP qu,rs1,qz,qx,qy

Description

This instruction compares the numerical values of the 16 8-bit unsigned vector data segments in registers `qx` and `qy`. The data segment with the larger value is written into the corresponding 8-bit data segment in register `qz`.

During the operation, the value in register `qu` is stored in memory. After the access, the value in register `rs1` is incremented by 16.

Operation

```
qz[  7:  0] = (qx[  7:  0]>=qy[  7:  0]) ? qx[  7:  0] : qy[  7:  0]
qz[ 15:  8] = (qx[ 15:  8]>=qy[ 15:  8]) ? qx[ 15:  8] : qy[ 15:  8]
...
qz[127:120] = (qx[127:120]>=qy[127:120]) ? qx[127:120] : qy[127:120]

  qu[127:0] => store128(rs1[31:0])
rs1 [31:0] = rs1[31:0] + 16
```

Schedule

Operands	Use	Def
qx	1	-
qy	1	-
qz	-	1
qu	2	-
rs1	1	1

Format

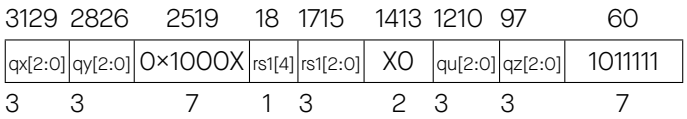


Figure 4.7-128. ESP.VMAX.U8.ST.INCP

4.7.2.129 ESP.VMIN.S16

Syntax

ESP.VMIN.S16 qz,qx,qy

Description

This instruction compares the numerical values of the eight 16-bit signed vector data segments in the registers qx and qy. The data segment with the smaller value is written into the corresponding 16-bit data segment in the register qz.

Operation

```
qz[ 15:  0] = (qx[ 15:  0]<=qy[ 15:  0]) ? qx[ 15:  0] : qy[ 15:  0]
qz[ 31: 16] = (qx[ 31: 16]<=qy[ 31: 16]) ? qx[ 31: 16] : qy[ 31: 16]
...
qz[127:112] = (qx[127:112]<=qy[127:112]) ? qx[127:112] : qy[127:112]
```

Schedule

Operands	Use	Def
qx	1	-
qy	1	-
qz	-	1

Format

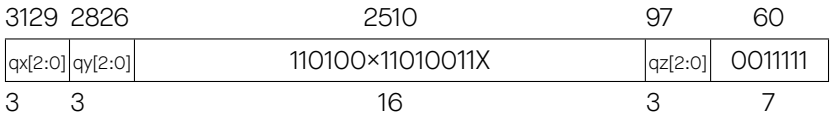


Figure 4.7-129. ESP.VMIN.S16

4.7.2.130 ESP.VMIN.S16.LD.INCP

Syntax

ESP.VMIN.S16.LD.INCP qu,rs1,qz,qx,qy

Description

This instruction compares the numerical values of the eight 16-bit signed vector data segments in the registers `qx` and `qy`. The data segment with the smaller value is written into the corresponding 16-bit data segment in the register `qz`.

During the operation, 16 bytes are loaded from the memory into the register `qu`. After the access, the value in the register `rs1` is incremented by 16.

Operation

```
qz[ 15:  0] = (qx[ 15:  0]<=qy[ 15:  0]) ? qx[ 15:  0] : qy[ 15:  0]
qz[ 31: 16] = (qx[ 31: 16]<=qy[ 31: 16]) ? qx[ 31: 16] : qy[ 31: 16]
...
qz[127:112] = (qx[127:112]<=qy[127:112]) ? qx[127:112] : qy[127:112]

qu[127:0] = load128(rs1[31:0])
rs1[31:0] = rs1[31:0] + 16
```

Schedule

Operands	Use	Def
qx	1	-
qy	1	-
qz	-	1
qu	-	2
rs1	1	1

Format

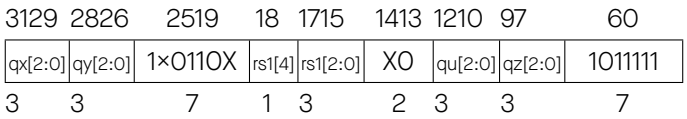


Figure 4.7-130. ESP.VMIN.S16.LD.INCP

4.7.2.131 ESP.VMIN.S16.ST.INCP

Syntax

ESP.VMIN.S16.ST.INCP qu,rs1,qz,qx,qy

Description

This instruction compares the numerical values of the eight 16-bit signed vector data segments in the registers `qx` and `qy`. The data segment with the smaller value is written into the corresponding 16-bit data segment in the register `qz`.

During the operation, the value in the register `qu` is stored in memory. After the access, the value in the register `rs1` is incremented by 16.

Operation

```
qz[ 15:  0] = (qx[ 15:  0]<=qy[ 15:  0]) ? qx[ 15:  0] : qy[ 15:  0]
qz[ 31: 16] = (qx[ 31: 16]<=qy[ 31: 16]) ? qx[ 31: 16] : qy[ 31: 16]
...
qz[127:112] = (qx[127:112]<=qy[127:112]) ? qx[127:112] : qy[127:112]

qu[127:0] => store128(rs1[31:0])
rs1[31:0] = rs1[31:0] + 16
```

Schedule

Operands	Use	Def
qx	1	-
qy	1	-
qz	-	1
qu	2	-
rs1	1	1

Format

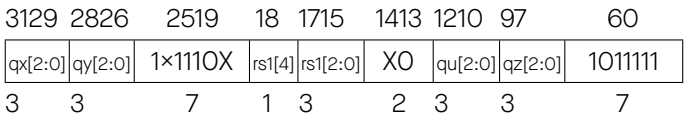


Figure 4.7-131. ESP.VMIN.S16.ST.INCP

4.7.2.132 ESP.VMIN.S32

Syntax

ESP.VMIN.S32 qz,qx,qy

Description

This instruction compares the numerical values of the four 32-bit signed vector data segments in the registers qx and qy. The data segment with the smaller value is written into the corresponding 32-bit data segment in the register qz.

Operation

```
qz[ 31:  0] = (qx[ 31:  0]<=qy[ 31:  0]) ? qx[ 31:  0] : qy[ 31:  0]
qz[ 63: 32] = (qx[ 63: 32]<=qy[ 63: 32]) ? qx[ 63: 32] : qy[ 63: 32]
...
qz[127: 96] = (qx[127: 96]<=qy[127: 96]) ? qx[127: 96] : qy[127: 96]
```

Schedule

Operands	Use	Def
qx	1	-
qy	1	-
qz	-	1

Format

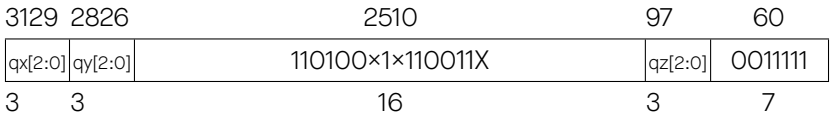


Figure 4.7-132. ESP.VMIN.S32

4.7.2.133 ESP.VMIN.S32.LD.INCP

Syntax

ESP.VMIN.S32.LD.INCP qu,rs1,qz,qx,qy

Description

This instruction compares the numerical values of the four 32-bit signed vector data segments in registers `qx` and `qy`. The data segment with the smaller value is written into the corresponding 32-bit data segment in register `qz`.

During the operation, 16 bytes are loaded from the memory into register `qu`. After the access, the value in register `rs1` is incremented by 16.

Operation

```
qz[ 31:  0] = (qx[ 31:  0]<=qy[ 31:  0]) ? qx[ 31:  0] : qy[ 31:  0]
qz[ 63: 32] = (qx[ 63: 32]<=qy[ 63: 32]) ? qx[ 63: 32] : qy[ 63: 32]
...
qz[127: 96] = (qx[127: 96]<=qy[127: 96]) ? qx[127: 96] : qy[127: 96]

qu[127:0] = load128(rs1[31:0])
rs1[31:0] = rs1[31:0] + 16
```

Schedule

Operands	Use	Def
qx	1	-
qy	1	-
qz	-	1
qu	-	2
rs1	1	1

Format

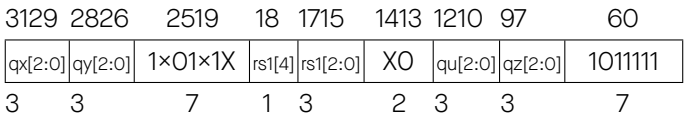


Figure 4.7-133. ESP.VMIN.S32.LD.INCP

4.7.2.134 ESP.VMIN.S32.ST.INCP

Syntax

ESP.VMIN.S32.ST.INCP qu,rs1,qz,qx,qy

Description

This instruction compares the numerical values of the four 32-bit signed vector data segments in the registers `qx` and `qy`. The data segment with the smaller value is written into the corresponding 32-bit data segment in the register `qz`.

During the operation, the value in the register `qu` is stored in memory. After the access, the value in the register `rs1` is incremented by 16.

Operation

```
qz[ 31:  0] = (qx[ 31:  0]<=qy[ 31:  0]) ? qx[ 31:  0] : qy[ 31:  0]
qz[ 63: 32] = (qx[ 63: 32]<=qy[ 63: 32]) ? qx[ 63: 32] : qy[ 63: 32]
...
qz[127: 96] = (qx[127: 96]<=qy[127: 96]) ? qx[127: 96] : qy[127: 96]

qu[127:0] = store128(rs1[31:0])
rs1[31:0] = rs1[31:0] + 16
```

Schedule

Operands	Use	Def
qx	1	-
qy	1	-
qz	-	1
qu	2	-
rs1	1	1

Format

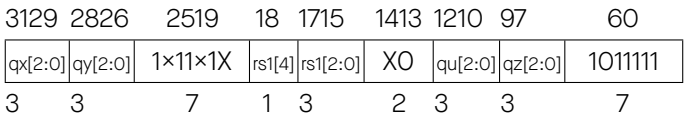


Figure 4.7-134. ESP.VMIN.S32.ST.INCP

4.7.2.135 ESP.VMIN.S8

Syntax

ESP.VMIN.S8 qz,qx,qy

Description

This instruction compares the numerical values of the 16 8-bit signed vector data segments in registers qx and qy. The data segment with the smaller value is written into the corresponding 8-bit data segment in the register qz.

Operation

```
qz[  7:  0] = (qx[  7:  0]<=qy[  7:  0]) ? qx[  7:  0] : qy[  7:  0]
qz[ 15:  8] = (qx[ 15:  8]<=qy[ 15:  8]) ? qx[ 15:  8] : qy[ 15:  8]
...
qz[127:120] = (qx[127:120]<=qy[127:120]) ? qx[127:120] : qy[127:120]
```

Schedule

Operands	Use	Def
qx	1	-
qy	1	-
qz	-	1

Format

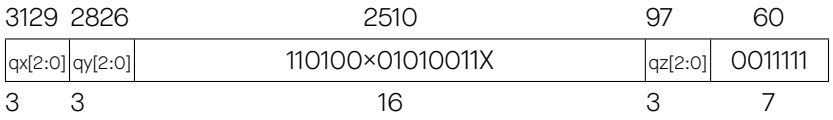


Figure 4.7-135. ESP.VMIN.S8

4.7.2.136 ESP.VMIN.S8.LD.INCP

Syntax

ESP.VMIN.S8.LD.INCP qu,rs1,qz,qx,qy

Description

This instruction compares the numerical values of the 16 8-bit signed vector data segments in registers `qx` and `qy`. The data segment with the smaller value is written into the corresponding 8-bit data segment in register `qz`.

During the operation, 16 bytes are loaded from the memory into register `qu`. After the access, the value in register `rs1` is incremented by 16.

Operation

```
qz[  7:  0] = (qx[  7:  0]<=qy[  7:  0]) ? qx[  7:  0] : qy[  7:  0]
qz[ 15:  8] = (qx[ 15:  8]<=qy[ 15:  8]) ? qx[ 15:  8] : qy[ 15:  8]
...
qz[127:120] = (qx[127:120]<=qy[127:120]) ? qx[127:120] : qy[127:120]

qu[127:0] = load128(rs1[31:0])
rs1[31:0] = rs1[31:0] + 16
```

Schedule

Operands	Use	Def
qx	1	-
qy	1	-
qz	-	1
qu	-	2
rs1	1	1

Format

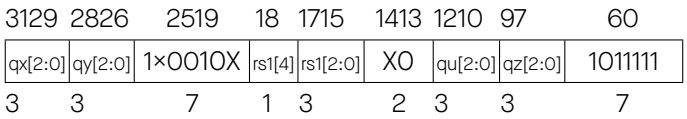


Figure 4.7-136. ESP.VMIN.S8.LD.INCP

4.7.2.137 ESP.VMIN.S8.ST.INCP

Syntax

ESP.VMIN.S8.ST.INCP qu,rs1,qz,qx,qy

Description

This instruction compares the numerical values of the 16 8-bit signed vector data segments in registers `qx` and `qy`. The data segment with the smaller value is written into the corresponding 8-bit data segment in register `qz`.

During the operation, the value in register `qu` is stored in memory. After the access, the value in register `rs1` is incremented by 16.

Operation

```
qz[  7:  0] = (qx[  7:  0]<=qy[  7:  0]) ? qx[  7:  0] : qy[  7:  0]
qz[ 15:  8] = (qx[ 15:  8]<=qy[ 15:  8]) ? qx[ 15:  8] : qy[ 15:  8]
...
qz[127:120] = (qx[127:120]<=qy[127:120]) ? qx[127:120] : qy[127:120]

qu[127:0] = store128(rs1[31:0])
rs1[31:0] = rs1[31:0] + 16
```

Schedule

Operands	Use	Def
qx	1	-
qy	1	-
qz	-	1
qu	2	-
rs1	1	1

Format

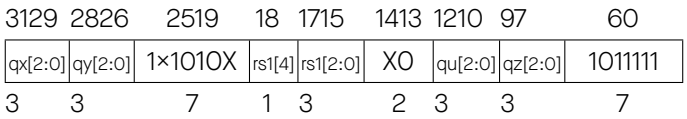


Figure 4.7-137. ESP.VMIN.S8.ST.INCP

4.7.2.138 ESP.VMIN.U16

Syntax

ESP.VMIN.U16 qz,qx,qy

Description

This instruction compares the numerical values of the eight 16-bit unsigned vector data segments in the registers qx and qy. The data segment with the smaller value is written into the corresponding 16-bit data segment in the register qz.

Operation

```
qz[ 15:  0] = (qx[ 15:  0]<=qy[ 15:  0]) ? qx[ 15:  0] : qy[ 15:  0]
qz[ 31: 16] = (qx[ 31: 16]<=qy[ 31: 16]) ? qx[ 31: 16] : qy[ 31: 16]
...
qz[127:112] = (qx[127:112]<=qy[127:112]) ? qx[127:112] : qy[127:112]
```

Schedule

Operands	Use	Def
qx	1	-
qy	1	-
qz	-	1

Format

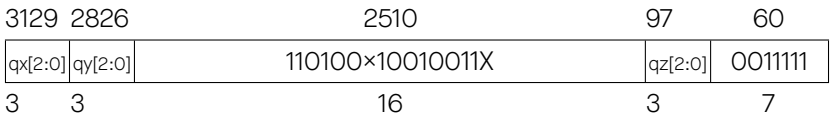


Figure 4.7-138. ESP.VMIN.U16

4.7.2.139 ESP.VMIN.U16.LD.INCP

Syntax

ESP.VMIN.U16.LD.INCP qu,rs1,qz,qx,qy

Description

This instruction compares the numerical values of the eight 16-bit unsigned vector data segments in registers `qx` and `qy`. The data segment with the smaller value is written into the corresponding 16-bit data segment in register `qz`.

During the operation, 16 bytes are loaded from the memory into register `qu`. After the access, the value in register `rs1` is incremented by 16.

Operation

```
qz[ 15:  0] = (qx[ 15:  0]<=qy[ 15:  0]) ? qx[ 15:  0] : qy[ 15:  0]
qz[ 31: 16] = (qx[ 31: 16]<=qy[ 31: 16]) ? qx[ 31: 16] : qy[ 31: 16]
...
qz[127:112] = (qx[127:112]<=qy[127:112]) ? qx[127:112] : qy[127:112]

qu[127:0] = load128(rs1[31:0])
rs1[31:0] = rs1[31:0] + 16
```

Schedule

Operands	Use	Def
qx	1	-
qy	1	-
qz	-	1
qu	-	2
rs1	1	1

Format

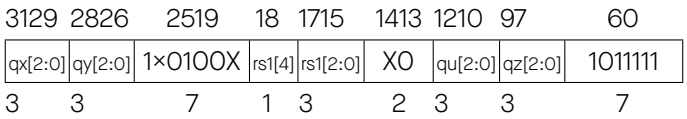


Figure 4.7-139. ESP.VMIN.U16.LD.INCP

4.7.2.140 ESP.VMIN.U16.ST.INCP

Syntax

ESP.VMIN.U16.ST.INCP qu,rs1,qz,qx,qy

Description

This instruction compares the numerical values of the eight 16-bit unsigned vector data segments in the registers `qx` and `qy`. The data segment with the smaller value is written into the corresponding 16-bit data segment in the register `qz`.

During the operation, the value in the register `qu` is stored in memory. After the access, the value in the register `rs1` is incremented by 16.

Operation

```
qz[ 15:  0] = (qx[ 15:  0]<=qy[ 15:  0]) ? qx[ 15:  0] : qy[ 15:  0]
qz[ 31: 16] = (qx[ 31: 16]<=qy[ 31: 16]) ? qx[ 31: 16] : qy[ 31: 16]
...
qz[127:112] = (qx[127:112]<=qy[127:112]) ? qx[127:112] : qy[127:112]

qu[127:0] = store128(rs1[31:0])
rs1[31:0] = rs1[31:0] + 16
```

Schedule

Operands	Use	Def
qx	1	-
qy	1	-
qz	-	1
qu	2	-
rs1	1	1

Format

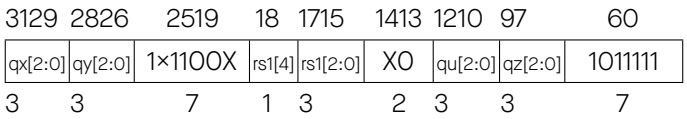


Figure 4.7-140. ESP.VMIN.U16.ST.INCP

4.7.2.141 ESP.VMIN.U32

Syntax

ESP.VMIN.U32 qz,qx,qy

Description

This instruction compares the numerical values of the four 32-bit unsigned vector data segments in the registers qx and qy. The data segment with the smaller value is written into the corresponding 32-bit data segment in the register qz.

Operation

```
qz[ 31:  0] = (qx[ 31:  0]<=qy[ 31:  0]) ? qx[ 31:  0] : qy[ 31:  0]
qz[ 63: 32] = (qx[ 63: 32]<=qy[ 63: 32]) ? qx[ 63: 32] : qy[ 63: 32]
...
qz[127: 96] = (qx[127: 96]<=qy[127: 96]) ? qx[127: 96] : qy[127: 96]
```

Schedule

Operands	Use	Def
qx	1	-
qy	1	-
qz	-	1

Format

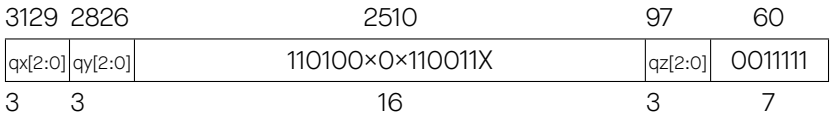


Figure 4.7-141. ESP.VMIN.U32

4.7.2.142 ESP.VMIN.U32.LD.INCP

Syntax

ESP.VMIN.U32.LD.INCP qu,rs1,qz,qx,qy

Description

This instruction compares the numerical values of the four 32-bit unsigned vector data segments in registers `qx` and `qy`. The data segment with the smaller value is written into the corresponding 32-bit data segment in register `qz`.

During the operation, 16 bytes are loaded from the memory into register `qu`. After the access, the value in register `rs1` is incremented by 16.

Operation

```
qz[ 31:  0] = (qx[ 31:  0]<=qy[ 31:  0]) ? qx[ 31:  0] : qy[ 31:  0]
qz[ 63: 32] = (qx[ 63: 32]<=qy[ 63: 32]) ? qx[ 63: 32] : qy[ 63: 32]
...
qz[127: 96] = (qx[127: 96]<=qy[127: 96]) ? qx[127: 96] : qy[127: 96]

qu[127:0] = load128(rs1[31:0])
rs1[31:0] = rs1[31:0] + 16
```

Schedule

Operands	Use	Def
qx	1	-
qy	1	-
qz	-	1
qu	-	2
rs1	1	1

Format

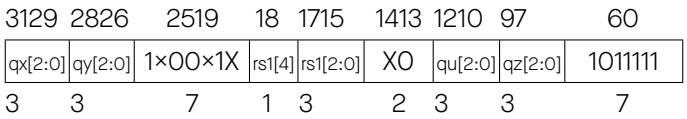


Figure 4.7-142. ESP.VMIN.U32.LD.INCP

4.7.2.143 ESP.VMIN.U32.ST.INCP

Syntax

ESP.VMIN.U32.ST.INCP qu,rs1,qz,qx,qy

Description

This instruction compares the numerical values of the four 32-bit unsigned vector data segments in the registers `qx` and `qy`. The data segment with the smaller value is written into the corresponding 32-bit data segment in the register `qz`.

During the operation, the value in the register `qu` is stored in memory. After the access, the value in the register `rs1` is incremented by 16.

Operation

```
qz[ 31:  0] = (qx[ 31:  0]<=qy[ 31:  0]) ? qx[ 31:  0] : qy[ 31:  0]
qz[ 63: 32] = (qx[ 63: 32]<=qy[ 63: 32]) ? qx[ 63: 32] : qy[ 63: 32]
...
qz[127: 96] = (qx[127: 96]<=qy[127: 96]) ? qx[127: 96] : qy[127: 96]

qu[127:0] = store128(rs1[31:0])
rs1[31:0] = rs1[31:0] + 16
```

Schedule

Operands	Use	Def
qx	1	-
qy	1	-
qz	-	1
qu	2	-
rs1	1	1

Format

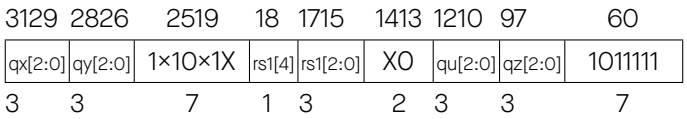


Figure 4.7-143. ESP.VMIN.U32.ST.INCP

4.7.2.144 ESP.VMIN.U8

Syntax

ESP.VMIN.U8 qz,qx,qy

Description

This instruction compares the numerical values of the 16 8-bit unsigned vector data segments in registers qx and qy. The data segment with the smaller value is written into the corresponding 8-bit data segment in the register qz.

Operation

```
qz[  7:  0] = (qx[  7:  0]<=qy[  7:  0]) ? qx[  7:  0] : qy[  7:  0]
qz[ 15:  8] = (qx[ 15:  8]<=qy[ 15:  8]) ? qx[ 15:  8] : qy[ 15:  8]
...
qz[127:120] = (qx[127:120]<=qy[127:120]) ? qx[127:120] : qy[127:120]
```

Schedule

Operands	Use	Def
qx	1	-
qy	1	-
qz	-	1

Format

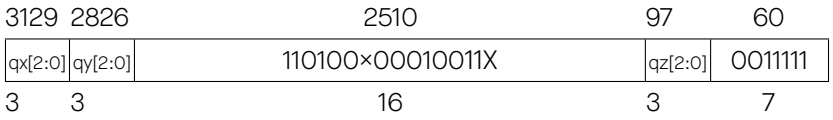


Figure 4.7-144. ESP.VMIN.U8

4.7.2.145 ESP.VMIN.U8.LD.INCP

Syntax

ESP.VMIN.U8.LD.INCP qu,rs1,qz,qx,qy

Description

This instruction compares the numerical values of the 16 8-bit unsigned vector data segments in registers `qx` and `qy`. The data segment with the smaller value is written into the corresponding 8-bit data segment in register `qz`.

During the operation, 16 bytes are loaded from the memory into register `qu`. After the access, the value in register `rs1` is incremented by 16.

Operation

```
qz[  7:  0] = (qx[  7:  0]<=qy[  7:  0]) ? qx[  7:  0] : qy[  7:  0]
qz[ 15:  8] = (qx[ 15:  8]<=qy[ 15:  8]) ? qx[ 15:  8] : qy[ 15:  8]
...
qz[127:120] = (qx[127:120]<=qy[127:120]) ? qx[127:120] : qy[127:120]

qu[127:0] = load128(rs1[31:0])
rs1[31:0] = rs1[31:0] + 16
```

Schedule

Operands	Use	Def
qx	1	-
qy	1	-
qz	-	1
qu	-	2
rs1	1	1

Format

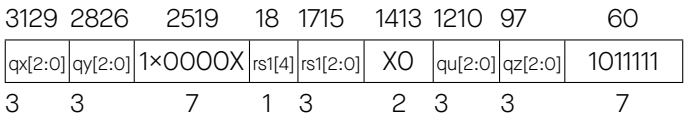


Figure 4.7-145. ESP.VMIN.U8.LD.INCP

4.7.2.146 ESP.VMIN.U8.ST.INCP

Syntax

ESP.VMIN.U8.ST.INCP qu,rs1,qz,qx,qy

Description

This instruction compares the numerical values of the 16 8-bit unsigned vector data segments in registers `qx` and `qy`. The data segment with the smaller value is written into the corresponding 8-bit data segment in register `qz`.

During the operation, the value in register `qu` is stored in memory. After the access, the value in register `rs1` is incremented by 16.

Operation

```
qz[  7:  0] = (qx[  7:  0]<=qy[  7:  0]) ? qx[  7:  0] : qy[  7:  0]
qz[ 15:  8] = (qx[ 15:  8]<=qy[ 15:  8]) ? qx[ 15:  8] : qy[ 15:  8]
...
qz[127:120] = (qx[127:120]<=qy[127:120]) ? qx[127:120] : qy[127:120]

qu[127:0] = store128(rs1[31:0])
rs1[31:0] = rs1[31:0] + 16
```

Schedule

Operands	Use	Def
qx	1	-
qy	1	-
qz	-	1
qu	2	-
rs1	1	1

Format

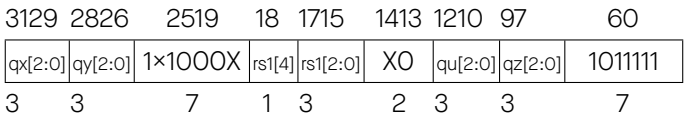


Figure 4.7-146. ESP.VMIN.U8.ST.INCP

4.7.2.147 ESP.VMUL.S16

Syntax

ESP.VMUL.S16 qz,qx,qy,sat,rm

Description

This instruction performs a signed vector multiplication on 16-bit data. Registers qx and qy are the multiplier and the multiplicand respectively. The eight 32-bit data results obtained from the calculation are arithmetically right-shifted by the value in the special register SAR. Then, the lower 16-bit data of the shift result are written into the corresponding segment of the register qz.

Operation

```
qz[ 15:  0] = (qx[ 15:  0] * qy[ 15:  0]) >> SAR[5:0]
qz[ 31: 16] = (qx[ 31: 16] * qy[ 31: 16]) >> SAR[5:0]
qz[ 47: 32] = (qx[ 47: 32] * qy[ 47: 32]) >> SAR[5:0]
qz[ 63: 48] = (qx[ 63: 48] * qy[ 63: 48]) >> SAR[5:0]
qz[ 79: 64] = (qx[ 79: 64] * qy[ 79: 64]) >> SAR[5:0]
qz[ 95: 80] = (qx[ 95: 80] * qy[ 95: 80]) >> SAR[5:0]
qz[111: 96] = (qx[111: 96] * qy[111: 96]) >> SAR[5:0]
qz[127:112] = (qx[127:112] * qy[127:112]) >> SAR[5:0]
```

Schedule

Operands	Use	Def
qx	1	-
qy	1	-
qz	-	2
SAR	1	-

Format

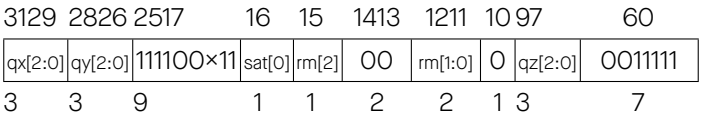


Figure 4.7-147. ESP.VMUL.S16

4.7.2.148 ESP.VMUL.S16.LD.INCP

Syntax

ESP.VMUL.S16.LD.INCP qu,rs1,qz,qx,qy,sat,rm

Description

This instruction performs a signed vector multiplication on 16-bit data. Registers `qx` and `qy` are the multiplier and the multiplicand respectively. The eight 32-bit data results obtained from the calculation are arithmetically right-shifted by the value in the special register `SAR`. Then, the lower 16-bit data of the shift result are written into the corresponding segment of the register `qz`.

During the operation, 16 bytes are loaded from the memory into the register `qu`. After the access, the value in the register `rs1` is incremented by 16.

Operation

```
qz[ 15:  0] = (qx[ 15:  0] * qy[ 15:  0]) >> SAR[5:0]
qz[ 31: 16] = (qx[ 31: 16] * qy[ 31: 16]) >> SAR[5:0]
qz[ 47: 32] = (qx[ 47: 32] * qy[ 47: 32]) >> SAR[5:0]
qz[ 63: 48] = (qx[ 63: 48] * qy[ 63: 48]) >> SAR[5:0]
qz[ 79: 64] = (qx[ 79: 64] * qy[ 79: 64]) >> SAR[5:0]
qz[ 95: 80] = (qx[ 95: 80] * qy[ 95: 80]) >> SAR[5:0]
qz[111: 96] = (qx[111: 96] * qy[111: 96]) >> SAR[5:0]
qz[127:112] = (qx[127:112] * qy[127:112]) >> SAR[5:0]

qu[127:0] = load128(rs1[31:0])
rs1[31:0] = rs1[31:0] + 16
```

Schedule

Operands	Use	Def
qx	1	-
qy	1	-
qz	-	2
SAR	1	-
qu	-	2
rs1	1	1

Format

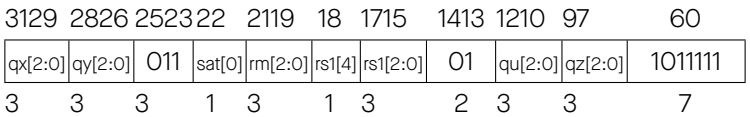


Figure 4.7-148. ESP.VMUL.S16.LD.INCP

4.7.2.149 ESP.VMUL.S16.S8XS8

Syntax

ESP.VMUL.S16.S8XS8 qz,qv,qx,qy,rm

Description

This instruction performs a signed vector multiplication on 8-bit data. Registers qx and qy are the multiplier and the multiplicand respectively. The 16 16-bit data results obtained from the calculation are arithmetically right-shifted by the value in the special register SAR. Then, the shift results are written into the corresponding segments of registers qz and qv.

Operation

```
qz[ 15:  0] = (qx[  7:  0] * qy[  7:  0]) >> SAR[5:0]
qz[ 31: 16] = (qx[ 15:  8] * qy[ 15:  8]) >> SAR[5:0]
qz[ 47: 32] = (qx[ 23: 16] * qy[ 23: 16]) >> SAR[5:0]
qz[ 63: 48] = (qx[ 31: 24] * qy[ 31: 24]) >> SAR[5:0]
qz[ 79: 64] = (qx[ 39: 32] * qy[ 39: 32]) >> SAR[5:0]
qz[ 95: 80] = (qx[ 47: 40] * qy[ 47: 40]) >> SAR[5:0]
qz[111: 96] = (qx[ 55: 48] * qy[ 55: 48]) >> SAR[5:0]
qz[127:112] = (qx[ 63: 56] * qy[ 63: 56]) >> SAR[5:0]

qv[ 15:  0] = (qx[ 71: 64] * qy[ 71: 64]) >> SAR[5:0]
qv[ 31: 16] = (qx[ 79: 72] * qy[ 79: 72]) >> SAR[5:0]
qv[ 47: 32] = (qx[ 87: 80] * qy[ 87: 80]) >> SAR[5:0]
qv[ 63: 48] = (qx[ 95: 88] * qy[ 95: 88]) >> SAR[5:0]
qv[ 79: 64] = (qx[103: 96] * qy[103: 96]) >> SAR[5:0]
qv[ 95: 80] = (qx[111:104] * qy[111:104]) >> SAR[5:0]
qv[111: 96] = (qx[119:112] * qy[119:112]) >> SAR[5:0]
qv[127:112] = (qx[127:120] * qy[127:120]) >> SAR[5:0]
```

Schedule

Operands	Use	Def
qx	1	-
qy	1	-
qz	-	2
qv	-	2
SAR	1	-

Format

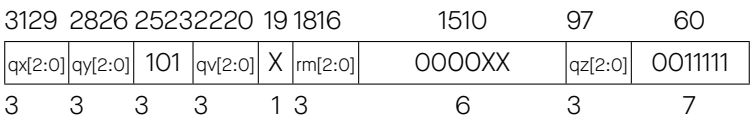


Figure 4.7-149. ESP.VMUL.S16.S8XS8

4.7.2.150 ESP.VMUL.S16.ST.INCP

Syntax

ESP.VMUL.S16.ST.INCP qu,rs1,qz,qx,qy,sat,rm

Description

This instruction performs a signed vector multiplication on 16-bit data. Registers `qx` and `qy` are the multiplier and the multiplicand respectively. The eight 32-bit data results obtained from the calculation are arithmetically right-shifted by the value in the special register `SAR`. Then, the lower 16-bit data of the shift result are written into the corresponding segment of the register `qz`.

During the operation, the value in the register `qu` is stored to memory. After the access, the value in the register `rs1` is incremented by 16.

Operation

```
qz[ 15:  0] = (qx[ 15:  0] * qy[ 15:  0]) >> SAR[5:0]
qz[ 31: 16] = (qx[ 31: 16] * qy[ 31: 16]) >> SAR[5:0]
qz[ 47: 32] = (qx[ 47: 32] * qy[ 47: 32]) >> SAR[5:0]
qz[ 63: 48] = (qx[ 63: 48] * qy[ 63: 48]) >> SAR[5:0]
qz[ 79: 64] = (qx[ 79: 64] * qy[ 79: 64]) >> SAR[5:0]
qz[ 95: 80] = (qx[ 95: 80] * qy[ 95: 80]) >> SAR[5:0]
qz[111: 96] = (qx[111: 96] * qy[111: 96]) >> SAR[5:0]
qz[127:112] = (qx[127:112] * qy[127:112]) >> SAR[5:0]

qu[127:0] => store128(rs1[31:0])
rs1[31:0] = rs1[31:0] + 16
```

Schedule

Operands	Use	Def
qx	1	-
qy	1	-
qz	-	2
SAR	1	□
qu	2	-
rs1	1	1

Format

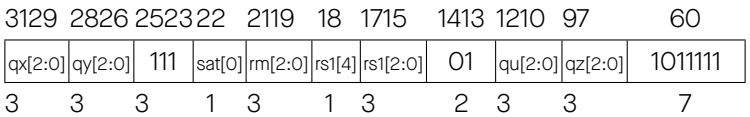


Figure 4.7-150. ESP.VMUL.S16.ST.INCP

4.7.2.151 ESP.VMUL.S32.S16XS16

Syntax

ESP.VMUL.S32.S16XS16 qz,qv,qx,qy,rm

Description

This instruction performs a signed vector multiplication on 16-bit data. Registers qx and qy are the multiplier and the multiplicand respectively. The eight 32-bit data results obtained from the calculation are arithmetically right-shifted by the value in the special register SAR. Then, the shift results are written into the corresponding segments of registers qz and qv.

Operation

```
qz[ 31:  0] = (qx[ 15:  0] * qy[ 15:  0]) >> SAR[5:0]
qz[ 63:  0] = (qx[ 31: 16] * qy[ 31: 16]) >> SAR[5:0]
qz[ 95: 64] = (qx[ 47: 32] * qy[ 47: 32]) >> SAR[5:0]
qz[127: 96] = (qx[ 63: 48] * qy[ 63: 48]) >> SAR[5:0]

qv[ 31:  0] = (qx[ 79: 64] * qy[ 79: 64]) >> SAR[5:0]
qv[ 63: 32] = (qx[ 95: 80] * qy[ 95: 80]) >> SAR[5:0]
qv[ 95: 64] = (qx[111: 96] * qy[111: 96]) >> SAR[5:0]
qv[127: 96] = (qx[127:112] * qy[127:112]) >> SAR[5:0]
```

Schedule

Operands	Use	Def
qx	1	-
qy	1	-
qz	-	2
qv	-	2
SAR	1	-

Format

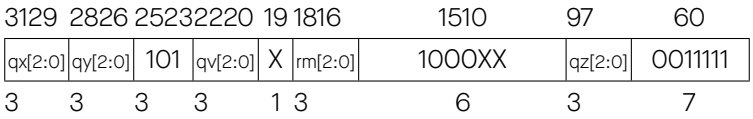


Figure 4.7-151. ESP.VMUL.S32.S16XS16

4.7.2.152 ESP.VMUL.S8

Syntax

ESP.VMUL.S8 qz,qx,qy,sat,rm

Description

This instruction performs a signed vector multiplication on 8-bit data. Registers `qx` and `qy` are the multiplier and the multiplicand respectively. The eight 16-bit data results obtained from the calculation are arithmetically right-shifted by the value in the special register `SAR`. Then, the lower 8-bit data of the shift result are written into the corresponding segment of the register `qz`.

Operation

```
qz[  7:  0] = (qx[  7:  0] * qy[  7:  0]) >> SAR[4:0]
qz[ 15:  8] = (qx[ 15:  8] * qy[ 15:  8]) >> SAR[4:0]
qz[ 23: 16] = (qx[ 23: 16] * qy[ 23: 16]) >> SAR[4:0]
qz[ 31: 24] = (qx[ 31: 24] * qy[ 31: 24]) >> SAR[4:0]
qz[ 39: 32] = (qx[ 39: 32] * qy[ 39: 32]) >> SAR[4:0]
qz[ 47: 40] = (qx[ 47: 40] * qy[ 47: 40]) >> SAR[4:0]
qz[ 55: 48] = (qx[ 55: 48] * qy[ 55: 48]) >> SAR[4:0]
qz[ 63: 56] = (qx[ 63: 56] * qy[ 63: 56]) >> SAR[4:0]
qz[ 71: 64] = (qx[ 71: 64] * qy[ 71: 64]) >> SAR[4:0]
qz[ 79: 71] = (qx[ 79: 72] * qy[ 79: 72]) >> SAR[4:0]
qz[ 87: 80] = (qx[ 87: 80] * qy[ 87: 80]) >> SAR[4:0]
qz[ 95: 88] = (qx[ 95: 88] * qy[ 95: 88]) >> SAR[4:0]
qz[103: 96] = (qx[103: 96] * qy[103: 96]) >> SAR[4:0]
qz[111:104] = (qx[111:104] * qy[111:104]) >> SAR[4:0]
qz[119:112] = (qx[119:112] * qy[119:112]) >> SAR[4:0]
qz[127:120] = (qx[127:120] * qy[127:120]) >> SAR[4:0]
```

Schedule

Operands	Use	Def
qx	1	-
qy	1	-
qz	-	2
SAR	1	-

Format

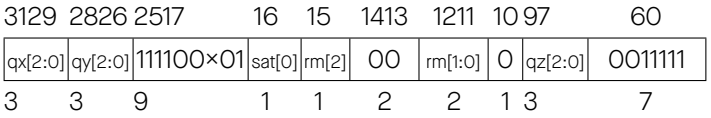


Figure 4.7-152. ESP.VMUL.S8

4.7.2.153 ESP.VMUL.S8.LD.INCP

Syntax

ESP.VMUL.S8.LD.INCP qu,rs1,qz,qx,qy,sat,rm

Description

This instruction performs a signed vector multiplication on 8-bit data. Registers `qx` and `qy` are the multiplier and the multiplicand respectively. The eight 16-bit data results obtained from the calculation are arithmetically right-shifted by the value in the special register `SAR`. Then, the lower 8-bit data of the shift result are written into the corresponding segment of the register `qz`.

During the operation, 16 bytes are loaded from the memory into the register `qu`. After the access, the value in the register `rs1` is incremented by 16.

Operation

```
qz[  7:  0] = (qx[  7:  0] * qy[  7:  0]) >> SAR[4:0]
qz[ 15:  8] = (qx[ 15:  8] * qy[ 15:  8]) >> SAR[4:0]
qz[ 23: 16] = (qx[ 23: 16] * qy[ 23: 16]) >> SAR[4:0]
qz[ 31: 24] = (qx[ 31: 24] * qy[ 31: 24]) >> SAR[4:0]
qz[ 39: 32] = (qx[ 39: 32] * qy[ 39: 32]) >> SAR[4:0]
qz[ 47: 40] = (qx[ 47: 40] * qy[ 47: 40]) >> SAR[4:0]
qz[ 55: 48] = (qx[ 55: 48] * qy[ 55: 48]) >> SAR[4:0]
qz[ 63: 56] = (qx[ 63: 56] * qy[ 63: 56]) >> SAR[4:0]
qz[ 71: 64] = (qx[ 71: 64] * qy[ 71: 64]) >> SAR[4:0]
qz[ 79: 71] = (qx[ 79: 72] * qy[ 79: 72]) >> SAR[4:0]
qz[ 87: 80] = (qx[ 87: 80] * qy[ 87: 80]) >> SAR[4:0]
qz[ 95: 88] = (qx[ 95: 88] * qy[ 95: 88]) >> SAR[4:0]
qz[103: 96] = (qx[103: 96] * qy[103: 96]) >> SAR[4:0]
qz[111:104] = (qx[111:104] * qy[111:104]) >> SAR[4:0]
qz[119:112] = (qx[119:112] * qy[119:112]) >> SAR[4:0]
qz[127:120] = (qx[127:120] * qy[127:120]) >> SAR[4:0]

qu[127:0] = load128(rs1[31:0])
rs1[31:0] = rs1[31:0] + 16
```

Schedule

Operands	Use	Def
qx	1	-
qy	1	-
qz	-	2
SAR	1	-
qu	-	2
rs1	1	1

Format

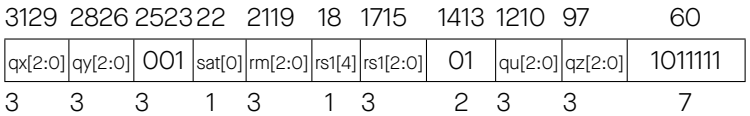


Figure 4.7-153. ESP.VMUL.S8.LD.INCP

4.7.2.154 ESP.VMUL.S8.ST.INCP

Syntax

ESP.VMUL.S8.ST.INCP qu,rs1,qz,qx,qy,sat,rm

Description

This instruction performs a signed vector multiplication on 8-bit data. Registers `qx` and `qy` are the multiplier and the multiplicand respectively. The eight 16-bit data results obtained from the calculation are arithmetically right-shifted by the value in the special register `SAR`. Then, the lower 8-bit data of the shift result are written into the corresponding segment of the register `qz`.

During the operation, the value in the register `qu` is stored to memory. After the access, the value in the register `rs1` is incremented by 16.

Operation

```
qz[  7:  0] = (qx[  7:  0] * qy[  7:  0]) >> SAR[4:0]
qz[ 15:  8] = (qx[ 15:  8] * qy[ 15:  8]) >> SAR[4:0]
qz[ 23: 16] = (qx[ 23: 16] * qy[ 23: 16]) >> SAR[4:0]
qz[ 31: 24] = (qx[ 31: 24] * qy[ 31: 24]) >> SAR[4:0]
qz[ 39: 32] = (qx[ 39: 32] * qy[ 39: 32]) >> SAR[4:0]
qz[ 47: 40] = (qx[ 47: 40] * qy[ 47: 40]) >> SAR[4:0]
qz[ 55: 48] = (qx[ 55: 48] * qy[ 55: 48]) >> SAR[4:0]
qz[ 63: 56] = (qx[ 63: 56] * qy[ 63: 56]) >> SAR[4:0]
qz[ 71: 64] = (qx[ 71: 64] * qy[ 71: 64]) >> SAR[4:0]
qz[ 79: 71] = (qx[ 79: 72] * qy[ 79: 72]) >> SAR[4:0]
qz[ 87: 80] = (qx[ 87: 80] * qy[ 87: 80]) >> SAR[4:0]
qz[ 95: 88] = (qx[ 95: 88] * qy[ 95: 88]) >> SAR[4:0]
qz[103: 96] = (qx[103: 96] * qy[103: 96]) >> SAR[4:0]
qz[111:104] = (qx[111:104] * qy[111:104]) >> SAR[4:0]
qz[119:112] = (qx[119:112] * qy[119:112]) >> SAR[4:0]
qz[127:120] = (qx[127:120] * qy[127:120]) >> SAR[4:0]

qu[127:0] => store128(rs1[31:0])
rs1[31:0] = rs1[31:0] + 16
```

Schedule

Operands	Use	Def
qx	1	-
qy	1	-
qz	-	2
SAR	1	-
qu	2	-
rs1	1	1

Format

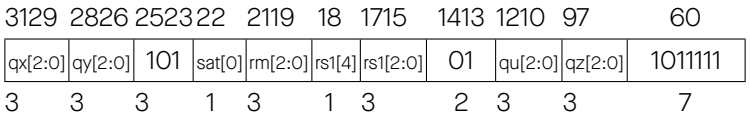


Figure 4.7-154. ESP.VMUL.S8.ST.INCP

4.7.2.155 ESP.VMUL.U16

Syntax

ESP.VMUL.U16 qz,qx,qy,sat,rm

Description

This instruction performs an unsigned vector multiplication on 16-bit data. Registers qx and qy are the multiplier and the multiplicand respectively. The eight 32-bit data results obtained from the calculation are logically right-shifted by the value in the special register SAR. Then, the lower 16-bit data of the shift result are written into the corresponding segment of the register qz.

Operation

```
qz[ 15:  0] = (qx[ 15:  0] * qy[ 15:  0]) >> SAR[5:0]
qz[ 31: 16] = (qx[ 31: 16] * qy[ 31: 16]) >> SAR[5:0]
qz[ 47: 32] = (qx[ 47: 32] * qy[ 47: 32]) >> SAR[5:0]
qz[ 63: 48] = (qx[ 63: 48] * qy[ 63: 48]) >> SAR[5:0]
qz[ 79: 64] = (qx[ 79: 64] * qy[ 79: 64]) >> SAR[5:0]
qz[ 95: 80] = (qx[ 95: 80] * qy[ 95: 80]) >> SAR[5:0]
qz[111: 96] = (qx[111: 96] * qy[111: 96]) >> SAR[5:0]
qz[127:112] = (qx[127:112] * qy[127:112]) >> SAR[5:0]
```

Schedule

Operands	Use	Def
qx	1	-
qy	1	-
qz	-	2
SAR	1	-

Format

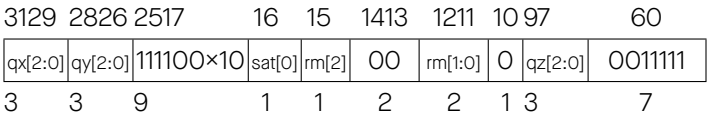


Figure 4.7-155. ESP.VMUL.U16

4.7.2.156 ESP.VMUL.U16.LD.INCP

Syntax

ESP.VMUL.U16.LD.INCP qu,rs1,qz,qx,qy,sat,rm

Description

This instruction performs an unsigned vector multiplication on 16-bit data. Registers `qx` and `qy` are the multiplier and the multiplicand respectively. The eight 32-bit data results obtained from the calculation are logically right-shifted by the value in the special register `SAR`. Then, the lower 16-bit data of the shift result are written into the corresponding segment of the register `qz`.

During the operation, the 16-byte data are loaded from the memory to the register `qu`. After the access, the value in the register `rs1` is incremented by 16.

Operation

```
qz[ 15:  0] = (qx[ 15:  0] * qy[ 15:  0]) >> SAR[5:0]
qz[ 31: 16] = (qx[ 31: 16] * qy[ 31: 16]) >> SAR[5:0]
qz[ 47: 32] = (qx[ 47: 32] * qy[ 47: 32]) >> SAR[5:0]
qz[ 63: 48] = (qx[ 63: 48] * qy[ 63: 48]) >> SAR[5:0]
qz[ 79: 64] = (qx[ 79: 64] * qy[ 79: 64]) >> SAR[5:0]
qz[ 95: 80] = (qx[ 95: 80] * qy[ 95: 80]) >> SAR[5:0]
qz[111: 96] = (qx[111: 96] * qy[111: 96]) >> SAR[5:0]
qz[127:112] = (qx[127:112] * qy[127:112]) >> SAR[5:0]

qu[127:0] = load128(rs1[31:0])
rs1[31:0] = rs1[31:0] + 16
```

Schedule

Operands	Use	Def
qx	1	-
qy	1	-
qz	-	2
SAR	1	-
qu	-	2
rs1	1	1

Format

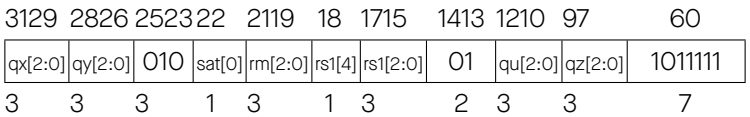


Figure 4.7-156. ESP.VMUL.U16.LD.INCP

4.7.2.157 ESP.VMUL.U16.ST.INCP

Syntax

ESP.VMUL.U16.ST.INCP qu,rs1,qz,qx,qy,sat,rm

Description

This instruction performs an unsigned vector multiplication on 16-bit data. Registers `qx` and `qy` are the multiplier and the multiplicand respectively. The eight 32-bit data results obtained from the calculation are logically right-shifted by the value in the special register `SAR`. Then, the lower 16-bit data of the shift result are written into the corresponding segment of the register `qz`.

During the operation, the value in the register `qu` is stored to memory. After the access, the value in the register `rs1` is incremented by 16.

Operation

```
qz[ 15:  0] = (qx[ 15:  0] * qy[ 15:  0]) >> SAR[5:0]
qz[ 31: 16] = (qx[ 31: 16] * qy[ 31: 16]) >> SAR[5:0]
qz[ 47: 32] = (qx[ 47: 32] * qy[ 47: 32]) >> SAR[5:0]
qz[ 63: 48] = (qx[ 63: 48] * qy[ 63: 48]) >> SAR[5:0]
qz[ 79: 64] = (qx[ 79: 64] * qy[ 79: 64]) >> SAR[5:0]
qz[ 95: 80] = (qx[ 95: 80] * qy[ 95: 80]) >> SAR[5:0]
qz[111: 96] = (qx[111: 96] * qy[111: 96]) >> SAR[5:0]
qz[127:112] = (qx[127:112] * qy[127:112]) >> SAR[5:0]

qu[127:0] => store128(rs1[31:0])
rs1[31:0] = rs1[31:0] + 16
```

Schedule

Operands	Use	Def
qx	1	-
qy	1	-
qz	-	2
SAR	1	-
qu	2	-
rs1	1	1

Format

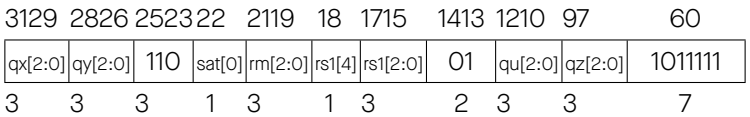


Figure 4.7-157. ESP.VMUL.U16.ST.INCP

4.7.2.158 ESP.VMUL.U8

Syntax

ESP.VMUL.U8 qz,qx,qy,sat,rm

Description

This instruction performs an unsigned vector multiplication on 8-bit data. Registers `qx` and `qy` are the multiplier and the multiplicand respectively. The 16 16-bit data results obtained from the calculation are logically right-shifted by the value in the special register `SAR`. Then, the lower 8-bit data of the shift result are written into the corresponding segment of the register `qz`.

Operation

```
qz[  7:  0] = (qx[  7:  0] * qy[  7:  0]) >> SAR[5:0]
qz[ 15:  8] = (qx[ 15:  8] * qy[ 15:  8]) >> SAR[5:0]
qz[ 23: 16] = (qx[ 23: 16] * qy[ 23: 16]) >> SAR[5:0]
qz[ 31: 24] = (qx[ 31: 24] * qy[ 31: 24]) >> SAR[5:0]
qz[ 39: 32] = (qx[ 39: 32] * qy[ 39: 32]) >> SAR[5:0]
qz[ 47: 40] = (qx[ 47: 40] * qy[ 47: 40]) >> SAR[5:0]
qz[ 55: 48] = (qx[ 55: 48] * qy[ 55: 48]) >> SAR[5:0]
qz[ 63: 56] = (qx[ 63: 56] * qy[ 63: 56]) >> SAR[5:0]
qz[ 71: 64] = (qx[ 71: 64] * qy[ 71: 64]) >> SAR[5:0]
qz[ 79: 72] = (qx[ 79: 72] * qy[ 79: 72]) >> SAR[5:0]
qz[ 87: 80] = (qx[ 87: 80] * qy[ 87: 80]) >> SAR[5:0]
qz[ 95: 88] = (qx[ 95: 88] * qy[ 95: 88]) >> SAR[5:0]
qz[103: 96] = (qx[103: 96] * qy[103: 96]) >> SAR[5:0]
qz[111:104] = (qx[111:104] * qy[111:104]) >> SAR[5:0]
qz[119:112] = (qx[119:112] * qy[119:112]) >> SAR[5:0]
qz[127:120] = (qx[127:120] * qy[127:120]) >> SAR[5:0]
```

Schedule

Operands	Use	Def
qx	1	-
qy	1	-
qz	-	2
SAR	1	-
qu	2	-
rs1	1	1

Format

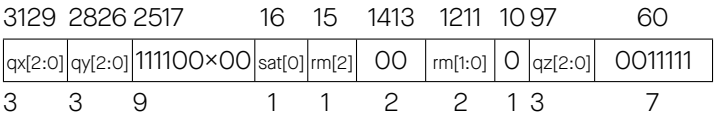


Figure 4.7-158. ESP.VMUL.U8

4.7.2.159 ESP.VMUL.U8.LD.INCP

Syntax

ESP.VMUL.U8.LD.INCP qu,rs1,qz,qx,qy,sat,rm

Description

This instruction performs an unsigned vector multiplication on 8-bit data. Registers `qx` and `qy` are the multiplier and the multiplicand respectively. The 16 32-bit data results obtained from the calculation are logically right-shifted by the value in the special register `SAR`. Then, the lower 8-bit data of the shift result are written into the corresponding segment of the register `qz`.

During the operation, the 16-byte data are loaded from the memory to the register `qu`. After the access, the value in the register `rs1` is incremented by 16.

Operation

```
qz[  7:  0] = (qx[  7:  0] * qy[  7:  0]) >> SAR[5:0]
qz[ 15:  8] = (qx[ 15:  8] * qy[ 15:  8]) >> SAR[5:0]
qz[ 23: 16] = (qx[ 23: 16] * qy[ 23: 16]) >> SAR[5:0]
qz[ 31: 24] = (qx[ 31: 24] * qy[ 31: 24]) >> SAR[5:0]
qz[ 39: 32] = (qx[ 39: 32] * qy[ 39: 32]) >> SAR[5:0]
qz[ 47: 40] = (qx[ 47: 40] * qy[ 47: 40]) >> SAR[5:0]
qz[ 55: 48] = (qx[ 55: 48] * qy[ 55: 48]) >> SAR[5:0]
qz[ 63: 56] = (qx[ 63: 56] * qy[ 63: 56]) >> SAR[5:0]
qz[ 71: 64] = (qx[ 71: 64] * qy[ 71: 64]) >> SAR[5:0]
qz[ 79: 72] = (qx[ 79: 72] * qy[ 79: 72]) >> SAR[5:0]
qz[ 87: 80] = (qx[ 87: 80] * qy[ 87: 80]) >> SAR[5:0]
qz[ 95: 88] = (qx[ 95: 88] * qy[ 95: 88]) >> SAR[5:0]
qz[103: 96] = (qx[103: 96] * qy[103: 96]) >> SAR[5:0]
qz[111:104] = (qx[111:104] * qy[111:104]) >> SAR[5:0]
qz[119:112] = (qx[119:112] * qy[119:112]) >> SAR[5:0]
qz[127:120] = (qx[127:120] * qy[127:120]) >> SAR[5:0]

qu[127:0] = load128(rs1[31:0])
rs1[31:0] = rs1[31:0] + 16
```

Schedule

Operands	Use	Def
qx	1	-
qy	1	-
qz	-	2
SAR	1	-
qu	-	2
rs1	1	1

Format

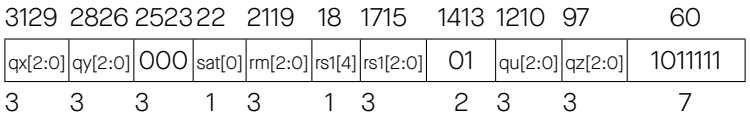


Figure 4.7-159. ESP.VMUL.U8.LD.INCP

4.7.2.160 ESP.VMUL.U8.ST.INCP

Syntax

ESP.VMUL.U8.ST.INCP qu,rs1,qz,qx,qy,sat,rm

Description

This instruction performs an unsigned vector multiplication on 8-bit data. Registers `qx` and `qy` are the multiplier and the multiplicand respectively. The 16 16-bit data results obtained from the calculation are logically right-shifted by the value in the special register `SAR`. Then, the lower 8-bit data of the shift result are written into the corresponding segment of the register `qz`.

During the operation, the value in the register `qu` is stored to memory. After the access, the value in the register `rs1` is incremented by 16.

Operation

```
qz[  7:  0] = (qx[  7:  0] * qy[  7:  0]) >> SAR[5:0]
qz[ 15:  8] = (qx[ 15:  8] * qy[ 15:  8]) >> SAR[5:0]
qz[ 23: 16] = (qx[ 23: 16] * qy[ 23: 16]) >> SAR[5:0]
qz[ 31: 24] = (qx[ 31: 24] * qy[ 31: 24]) >> SAR[5:0]
qz[ 39: 32] = (qx[ 39: 32] * qy[ 39: 32]) >> SAR[5:0]
qz[ 47: 40] = (qx[ 47: 40] * qy[ 47: 40]) >> SAR[5:0]
qz[ 55: 48] = (qx[ 55: 48] * qy[ 55: 48]) >> SAR[5:0]
qz[ 63: 56] = (qx[ 63: 56] * qy[ 63: 56]) >> SAR[5:0]
qz[ 71: 64] = (qx[ 71: 64] * qy[ 71: 64]) >> SAR[5:0]
qz[ 79: 72] = (qx[ 79: 72] * qy[ 79: 72]) >> SAR[5:0]
qz[ 87: 80] = (qx[ 87: 80] * qy[ 87: 80]) >> SAR[5:0]
qz[ 95: 88] = (qx[ 95: 88] * qy[ 95: 88]) >> SAR[5:0]
qz[103: 96] = (qx[103: 96] * qy[103: 96]) >> SAR[5:0]
qz[111:104] = (qx[111:104] * qy[111:104]) >> SAR[5:0]
qz[119:112] = (qx[119:112] * qy[119:112]) >> SAR[5:0]
qz[127:120] = (qx[127:120] * qy[127:120]) >> SAR[5:0]

qu[127:0] => store128(rs1[31:0])
rs1[31:0] = rs1[31:0] + 16
```

Schedule

Operands	Use	Def
qx	1	-
qy	1	-
qz	-	2
SAR	1	-
qu	2	-
rs1	1	1

Format

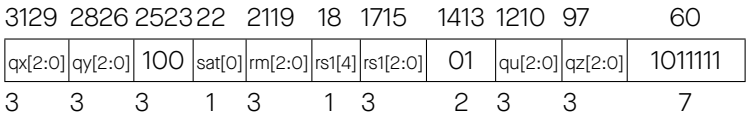


Figure 4.7-160. ESP.VMUL.U8.ST.INCP

4.7.2.161 ESP.VPRELU.S16

Syntax

ESP.VPRELU.S16 qz,qy,qx,rs1,sat,rm

Description

This instruction divides the register `qx` into 8 data segments of 16 bits each. If the value of a segment is not greater than 0, it performs multiplication with the value of the corresponding 16-bit segment in the register `qy`, right-shifted by the lower 6-bit value in the register `rs1`. The result obtained is then assigned to the corresponding 16-bit data segment in the register `qz`. Otherwise, the value of the segment in `qx` is assigned to `qz`. The data is signed.

Operation

```
qz[ 15:  0] = (qx[ 15:  0]<=0) ? (qx[ 15:  0] * qy[ 15:  0]) >> rs1[5:0] : qx[ 15:  0]
qz[ 31: 16] = (qx[ 31: 16]<=0) ? (qx[ 31: 16] * qy[ 31: 16]) >> rs1[5:0] : qx[ 31: 16]
...
qz[127:112] = (qx[127:112]<=0) ? (qx[127:112] * qy[127:112]) >> rs1[5:0] : qx[127:112]
```

Schedule

Operands	Use	Def
qx	1	-
qy	1	-
qz	-	2
rs1	1	-

Format

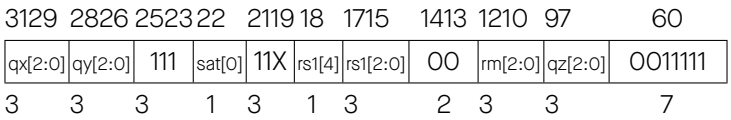


Figure 4.7-161. ESP.VPRELU.S16

4.7.2.162 ESP.VPRELU.S8

Syntax

ESP.VPRELU.S8 qz,qy,qx,rs1,sat,rm

Description

This instruction divides the register `qx` into 16 data segments of 8 bits each. If the value of a segment is not greater than 0, it performs multiplication with the value of the corresponding 8-bit segment in the register `qy`, right-shifts by the lower 5-bit value in the register `rs1`, and then assigns the calculated result to the corresponding 8-bit data segment in the register `qz`. Otherwise, the value of the segment in `qx` is assigned to `qz`.

Operation

```
qz[  7:  0] = (qx[  7:  0]<=0) ? (qx[  7:  0] * qy[  7:  0]) >> rs1[4:0] : qx[  7:  0]
qz[ 15:  8] = (qx[ 15:  8]<=0) ? (qx[ 15:  8] * qy[ 15:  8]) >> rs1[4:0] : qx[ 15:  8]
...
qz[127:120] = (qx[127:120]<=0) ? (qx[127:120] * qy[127:120]) >> rs1[4:0] : qx[127:120]
```

Schedule

Operands	Use	Def
qx	1	-
qy	1	-
qz	-	2
rs1	1	-

Format

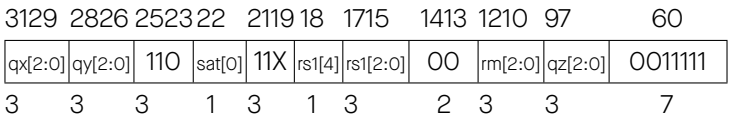


Figure 4.7-162. ESP.VPRELU.S8

4.7.2.163 ESP.VRELU.S16

Syntax

ESP.VRELU.S16 qy,rs2,rs1,sat,rm

Description

This instruction divides register `qy` into 8 data segments by 16 bits. If the value of the segment is not greater than 0, it will be multiplied by the value of the lower 16 bits in register `rs2` and right-shifted by the value of the lower 6 bits in register `rs1`, and then the result obtained will overwrite the value of the segment. Otherwise, the value of the segment will remain unchanged.

Operation

```
qy[ 15:  0] = (qy[ 15:  0]<=0) ? (qy[ 15:  0] * rs2[15:0]) >> rs1[5:0] : qy[ 15:
    0]
qy[ 31: 16] = (qy[ 31: 16]<=0) ? (qy[ 31: 16] * rs2[15:0]) >> rs1[5:0] : qy[ 31:
    16]
...
qy[127:112] = (qy[127:112]<=0) ? (qy[127:112] * rs2[15:0]) >> rs1[5:0] : qy
    [127:112]
```

Schedule

Operands	Use	Def
qy	1	2
rs2	1	-
rs1	1	-

Format

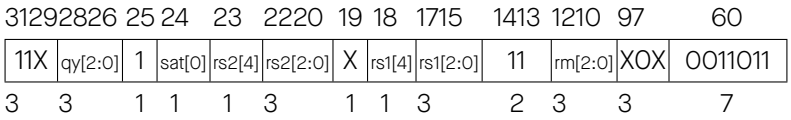


Figure 4.7-163. ESP.VRELU.S16

4.7.2.164 ESP.VRELU.S8

Syntax

ESP.VRELU.S8 qy,rs2,rs1,sat,rm

Description

This instruction divides register `qy` into 16 data segments by 8 bits. If the value of the segment is not greater than 0, it will be multiplied by the value of the lower 8 bits in register `rs2` and right-shifted by the value of the lower 5 bits in register `rs1`, and then the result obtained will overwrite the value of the segment. Otherwise, the value of the segment will remain unchanged.

Operation

```
qy[  7:  0] = (qy[  7:  0]<=0) ? (qy[  7:  0] * rs2[7:0]) >> rs1[4:0] : qy[  7:  0]
qy[ 15:  8] = (qy[ 15:  8]<=0) ? (qy[ 15:  8] * rs2[7:0]) >> rs1[4:0] : qy[ 15:  8]
...
qy[127:120] = (qy[127:120]<=0) ? (qy[127:120] * rs2[7:0]) >> rs1[4:0] : qy[127:120]
```

Schedule

Operands	Use	Def
qy	1	2
rs2	1	-
rs1	1	-

Format

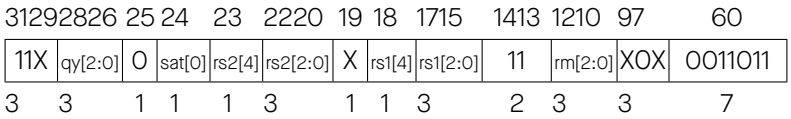


Figure 4.7-164. ESP.VRELU.S8

4.7.2.165 ESP.VSADDS.S16

Syntax

ESP.VSADDS.S16 qv,qx,rs1,sat

Description

This instruction performs a signed vector addition on 16-bit data.

Operation

```
qv[ 15:  0] = min( max( qx[ 15:  0] + rs1[15:0], -2^{15} ), 2^{15}-1 )
qv[ 31: 16] = min( max( qx[ 31: 16] + rs1[15:0], -2^{15} ), 2^{15}-1 )
qv[ 47: 32] = min( max( qx[ 47: 32] + rs1[15:0], -2^{15} ), 2^{15}-1 )
qv[ 63: 48] = min( max( qx[ 63: 48] + rs1[15:0], -2^{15} ), 2^{15}-1 )
qv[ 79: 64] = min( max( qx[ 79: 64] + rs1[15:0], -2^{15} ), 2^{15}-1 )
qv[ 95: 80] = min( max( qx[ 95: 80] + rs1[15:0], -2^{15} ), 2^{15}-1 )
qv[111: 96] = min( max( qx[111: 96] + rs1[15:0], -2^{15} ), 2^{15}-1 )
qv[127:112] = min( max( qx[127:112] + rs1[15:0], -2^{15} ), 2^{15}-1 )
```

Schedule

Operands	Use	Def
qv	-	2
qx	1	-
rs1	1	-

Format

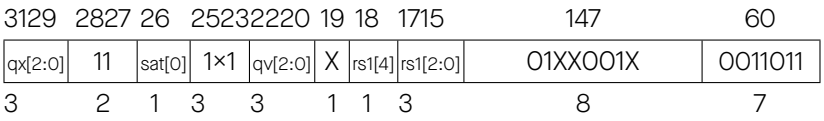


Figure 4.7-165. ESP.VSADDS.S16

4.7.2.166 ESP.VSADDS.S8

Syntax

ESP.VSADDS.S8 qv,qx,rs1,sat

Description

This instruction performs a signed vector addition on 8-bit data.

Operation

```
qv[  7:  0] = min( max( qx[  7:  0] + rs1[7:0], -2^{7} ), 2^{7}-1 )
qv[ 15:  8] = min( max( qx[ 15:  8] + rs1[7:0], -2^{7} ), 2^{7}-1 )
qv[ 23: 16] = min( max( qx[ 23: 16] + rs1[7:0], -2^{7} ), 2^{7}-1 )
qv[ 31: 24] = min( max( qx[ 31: 24] + rs1[7:0], -2^{7} ), 2^{7}-1 )
qv[ 39: 32] = min( max( qx[ 39: 32] + rs1[7:0], -2^{7} ), 2^{7}-1 )
qv[ 47: 40] = min( max( qx[ 47: 40] + rs1[7:0], -2^{7} ), 2^{7}-1 )
qv[ 55: 48] = min( max( qx[ 55: 48] + rs1[7:0], -2^{7} ), 2^{7}-1 )
qv[ 63: 56] = min( max( qx[ 63: 56] + rs1[7:0], -2^{7} ), 2^{7}-1 )
qv[ 71: 64] = min( max( qx[ 71: 64] + rs1[7:0], -2^{7} ), 2^{7}-1 )
qv[ 79: 72] = min( max( qx[ 79: 72] + rs1[7:0], -2^{7} ), 2^{7}-1 )
qv[ 87: 80] = min( max( qx[ 87: 80] + rs1[7:0], -2^{7} ), 2^{7}-1 )
qv[ 95: 88] = min( max( qx[ 95: 88] + rs1[7:0], -2^{7} ), 2^{7}-1 )
qv[103: 96] = min( max( qx[103: 96] + rs1[7:0], -2^{7} ), 2^{7}-1 )
qv[111:104] = min( max( qx[111:104] + rs1[7:0], -2^{7} ), 2^{7}-1 )
qv[119:112] = min( max( qx[119:112] + rs1[7:0], -2^{7} ), 2^{7}-1 )
qv[127:120] = min( max( qx[127:120] + rs1[7:0], -2^{7} ), 2^{7}-1 )
```

Schedule

Operands	Use	Def
qv	-	2
qx	1	-
rs1	1	-

Format

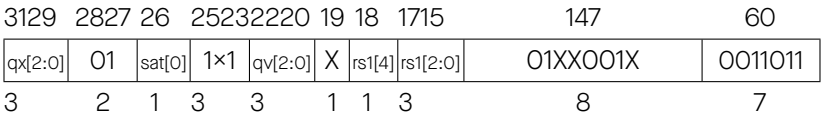


Figure 4.7-166. ESP.VSADDS.S8

4.7.2.167 ESP.VSADDS.U16

Syntax

ESP.VSADDS.U16 qv,qx,rs1,sat

Description

This instruction performs an unsigned vector addition on 16-bit data.

Operation

```
qv[ 15:  0] = min( qx[ 15:  0] + rs1[15:0], 2^{16}-1 )
qv[ 31: 16] = min( qx[ 31: 16] + rs1[15:0], 2^{16}-1 )
qv[ 47: 32] = min( qx[ 47: 32] + rs1[15:0], 2^{16}-1 )
qv[ 63: 48] = min( qx[ 63: 48] + rs1[15:0], 2^{16}-1 )
qv[ 79: 64] = min( qx[ 79: 64] + rs1[15:0], 2^{16}-1 )
qv[ 95: 80] = min( qx[ 95: 80] + rs1[15:0], 2^{16}-1 )
qv[111: 96] = min( qx[111: 96] + rs1[15:0], 2^{16}-1 )
qv[127:112] = min( qx[127:112] + rs1[15:0], 2^{16}-1 )
```

Schedule

Operands	Use	Def
qv	-	2
qx	1	-
rs1	1	-

Format

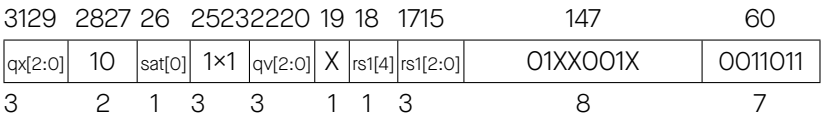


Figure 4.7-167. ESP.VSADDS.U16

4.7.2.168 ESP.VSADDS.U8

Syntax

ESP.VSADDS.U8 qv,qx,rs1,sat

Description

This instruction performs an unsigned vector addition on 8-bit data.

Operation

```
qv[  7:  0] = min( qx[  7:  0] + rs1[7:0], 2^{8}-1 )
qv[ 15:  8] = min( qx[ 15:  8] + rs1[7:0], 2^{8}-1 )
qv[ 23: 16] = min( qx[ 23: 16] + rs1[7:0], 2^{8}-1 )
qv[ 31: 24] = min( qx[ 31: 24] + rs1[7:0], 2^{8}-1 )
qv[ 39: 32] = min( qx[ 39: 32] + rs1[7:0], 2^{8}-1 )
qv[ 47: 40] = min( qx[ 47: 40] + rs1[7:0], 2^{8}-1 )
qv[ 55: 48] = min( qx[ 55: 48] + rs1[7:0], 2^{8}-1 )
qv[ 63: 56] = min( qx[ 63: 56] + rs1[7:0], 2^{8}-1 )
qv[ 71: 64] = min( qx[ 71: 64] + rs1[7:0], 2^{8}-1 )
qv[ 79: 72] = min( qx[ 79: 72] + rs1[7:0], 2^{8}-1 )
qv[ 87: 80] = min( qx[ 87: 80] + rs1[7:0], 2^{8}-1 )
qv[ 95: 88] = min( qx[ 95: 88] + rs1[7:0], 2^{8}-1 )
qv[103: 96] = min( qx[103: 96] + rs1[7:0], 2^{8}-1 )
qv[111:104] = min( qx[111:104] + rs1[7:0], 2^{8}-1 )
qv[119:112] = min( qx[119:112] + rs1[7:0], 2^{8}-1 )
qv[127:120] = min( qx[127:120] + rs1[7:0], 2^{8}-1 )
```

Schedule

Operands	Use	Def
qv	-	2
qx	1	-
rs1	1	-

Format

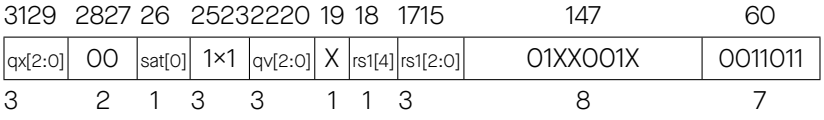


Figure 4.7-168. ESP.VSADDS.U8

4.7.2.169 ESP.VSAT.S16

Syntax

ESP.VSAT.S16 qz,qx,rs1,rs2

Description

This instruction performs vector saturation for 16-bit signed segments.

Operation

```
    min_t[15:0] = max( rs1[15:0], rs2[15:0] )
    max_t[15:0] = min( rs1[15:0], rs2[15:0] )

    qz[ 15:  0] = max( min( qx[ 15:  0], max_t[15:0]), min_t[15:0] )
    qz[ 31: 16] = max( min( qx[ 31: 16], maxt_[15:0]), min_t[15:0] )
    qz[ 47: 32] = max( min( qx[ 47: 32], max_t[15:0]), min_t[15:0] )
    qz[ 63: 48] = max( min( qx[ 63: 48], max_t[15:0]), min_t[15:0] )
    qz[ 79: 64] = max( min( qx[ 79: 64], max_t[15:0]), min_t[15:0] )
    qz[ 95: 80] = max( min( qx[ 95: 80], max_t[15:0]), min_t[15:0] )
    qz[111: 96] = max( min( qx[111: 96], max_t[15:0]), min_t[15:0] )
    qz[127:112] = max( min( qx[127:112], max_t[15:0]), min_t[15:0] )
```

Schedule

Operands	Use	Def
qz	-	1
qx	1	-
rs1	1	-
rs2	1	-

Format

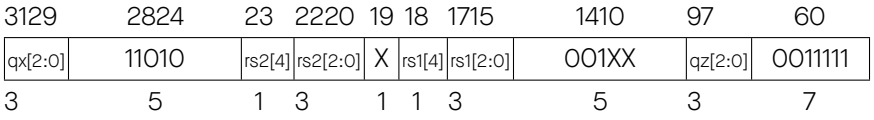


Figure 4.7-169. ESP.VSAT.S16

4.7.2.170 ESP.VSAT.S32

Syntax

ESP.VSAT.S32 qz,qx,rs1,rs2

Description

This instruction performs vector saturation for 32-bit signed segments.

Operation

```
    min_t[31:0] = max( rs1[31;0], rs2[31:0] )
    max_t[31:0] = min( rs1[31:0], rs2[31:0] )

    qz[ 31:  0] = max( min( qx[ 31:  0], max_t[31:0]), min_t[31:0] )
    qz[ 63: 32] = max( min( qx[ 63: 32], max_t[31:0]), min_t[31:0] )
    qz[ 95: 64] = max( min( qx[ 95: 64], max_t[31:0]), min_t[31:0] )
    qz[127: 96] = max( min( qx[127: 96], maxt_[31:0]), min_t[31:0] )
```

Schedule

Operands	Use	Def
qz	-	1
qx	1	-
rs1	1	-
rs2	1	-

Format

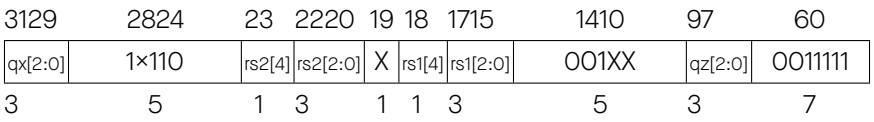


Figure 4.7-170. ESP.VSAT.S32

4.7.2.171 ESP.VSAT.S8

Syntax

ESP.VSAT.S8 qz,qx,rs1,rs2

Description

This instruction performs vector saturation for 8-bit signed segments.

Operation

```
min_t[7:0] = max( rs1[7:0], rs2[7:0] )
max_t[7:0] = min( rs1[7:0], rs2[7:0] )

qz[  7:  0] = max( min( qx[  7:  0], max_t[7:0]), min_t[7:0] )
qz[ 15:  8] = max( min( qx[ 15:  8], max_t[7:0]), min_t[7:0] )
qz[ 23: 16] = max( min( qx[ 23: 16], max_t[7:0]), min_t[7:0] )
qz[ 31: 24] = max( min( qx[ 31: 24], maxt_[7:0]), min_t[7:0] )
qz[ 39: 32] = max( min( qx[ 39: 32], max_t[7:0]), min_t[7:0] )
qz[ 47: 40] = max( min( qx[ 47: 40], max_t[7:0]), min_t[7:0] )
qz[ 55: 48] = max( min( qx[ 55: 48], max_t[7:0]), min_t[7:0] )
qz[ 63: 56] = max( min( qx[ 63: 56], max_t[7:0]), min_t[7:0] )
qz[ 71: 64] = max( min( qx[ 71: 64], max_t[7:0]), min_t[7:0] )
qz[ 79: 72] = max( min( qx[ 79: 72], max_t[7:0]), min_t[7:0] )
qz[ 87: 80] = max( min( qx[ 87: 80], max_t[7:0]), min_t[7:0] )
qz[ 95: 88] = max( min( qx[ 95: 88], max_t[7:0]), min_t[7:0] )
qz[103: 96] = max( min( qx[103: 96], max_t[7:0]), min_t[7:0] )
qz[111:104] = max( min( qx[111:104], max_t[7:0]), min_t[7:0] )
qz[119:112] = max( min( qx[119:112], max_t[7:0]), min_t[7:0] )
qz[127:120] = max( min( qx[127:120], max_t[7:0]), min_t[7:0] )
```

Schedule

Operands	Use	Def
qz	-	1
qx	1	-
rs1	1	-
rs2	1	-

Format

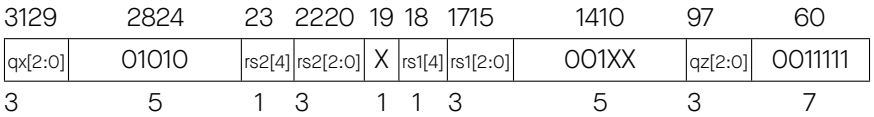


Figure 4.7-171. ESP.VSAT.S8

4.7.2.172 ESP.VSAT.U16

Syntax

ESP.VSAT.U16 qz,qx,rs1,rs2

Description

This instruction performs vector saturation for 16-bit unsigned segments.

Operation

```
    min_t[15:0] = max( rs1[15:0], rs2[15:0] )
    max_t[15:0] = min( rs1[15:0], rs2[15:0] )

    qz[ 15:  0] = max( min( qx[ 15:  0], max_t[15:0]), min_t[15:0] )
    qz[ 31: 16] = max( min( qx[ 31: 16], maxt_[15:0]), min_t[15:0] )
    qz[ 47: 32] = max( min( qx[ 47: 32], max_t[15:0]), min_t[15:0] )
    qz[ 63: 48] = max( min( qx[ 63: 48], max_t[15:0]), min_t[15:0] )
    qz[ 79: 64] = max( min( qx[ 79: 64], max_t[15:0]), min_t[15:0] )
    qz[ 95: 80] = max( min( qx[ 95: 80], max_t[15:0]), min_t[15:0] )
    qz[111: 96] = max( min( qx[111: 96], max_t[15:0]), min_t[15:0] )
    qz[127:112] = max( min( qx[127:112], max_t[15:0]), min_t[15:0] )
```

Schedule

Operands	Use	Def
qz	-	1
qx	1	-
rs1	1	-
rs2	1	-

Format

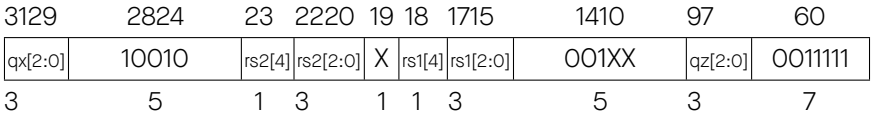


Figure 4.7-172. ESP.VSAT.U16

4.7.2.173 ESP.VSAT.U32

Syntax

ESP.VSAT.U32 qz,qx,rs1,rs2

Description

This instruction performs vector saturation for 32-bit unsigned segments.

Operation

```
    min_t[31:0] = max( rs1[31;0], rs2[31:0] )
    max_t[31:0] = min( rs1[31:0], rs2[31:0] )

    qz[ 31:  0] = max( min( qx[ 31:  0], max_t[31:0]), min_t[31:0] )
    qz[ 63: 32] = max( min( qx[ 63: 32], max_t[31:0]), min_t[31:0] )
    qz[ 95: 64] = max( min( qx[ 95: 64], max_t[31:0]), min_t[31:0] )
    qz[127: 96] = max( min( qx[127: 96], maxt_[31:0]), min_t[31:0] )
```

Schedule

Operands	Use	Def
qz	-	1
qx	1	-
rs1	1	-
rs2	1	-

Format

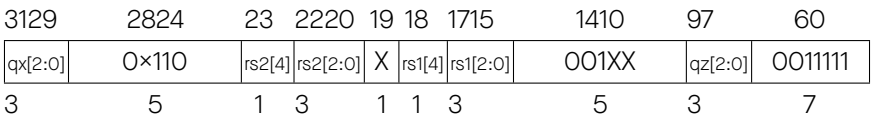


Figure 4.7-173. ESP.VSAT.U32

4.7.2.174 ESP.VSAT.U8

Syntax

ESP.VSAT.U8 qz,qx,rs1,rs2

Description

This instruction performs vector saturation for 8-bit unsigned segments.

Operation

```
    min_t[7:0] = max( rs1[7:0], rs2[7:0] )
    max_t[7:0] = min( rs1[7:0], rs2[7:0] )

    qz[  7:  0] = max( min( qx[  7:  0], max_t[7:0]), min_t[7:0] )
    qz[ 15:  8] = max( min( qx[ 15:  8], max_t[7:0]), min_t[7:0] )
    qz[ 23: 16] = max( min( qx[ 23: 16], max_t[7:0]), min_t[7:0] )
    qz[ 31: 24] = max( min( qx[ 31: 24], maxt_[7:0]), min_t[7:0] )
    qz[ 39: 32] = max( min( qx[ 39: 32], max_t[7:0]), min_t[7:0] )
    qz[ 47: 40] = max( min( qx[ 47: 40], max_t[7:0]), min_t[7:0] )
    qz[ 55: 48] = max( min( qx[ 55: 48], max_t[7:0]), min_t[7:0] )
    qz[ 63: 56] = max( min( qx[ 63: 56], max_t[7:0]), min_t[7:0] )
    qz[ 71: 64] = max( min( qx[ 71: 64], max_t[7:0]), min_t[7:0] )
    qz[ 79: 72] = max( min( qx[ 79: 72], max_t[7:0]), min_t[7:0] )
    qz[ 87: 80] = max( min( qx[ 87: 80], max_t[7:0]), min_t[7:0] )
    qz[ 95: 88] = max( min( qx[ 95: 88], max_t[7:0]), min_t[7:0] )
    qz[103: 96] = max( min( qx[103: 96], max_t[7:0]), min_t[7:0] )
    qz[111:104] = max( min( qx[111:104], max_t[7:0]), min_t[7:0] )
    qz[119:112] = max( min( qx[119:112], max_t[7:0]), min_t[7:0] )
    qz[127:120] = max( min( qx[127:120], max_t[7:0]), min_t[7:0] )
```

Schedule

Operands	Use	Def
qz	-	1
qx	1	-
rs1	1	-
rs2	1	-

Format

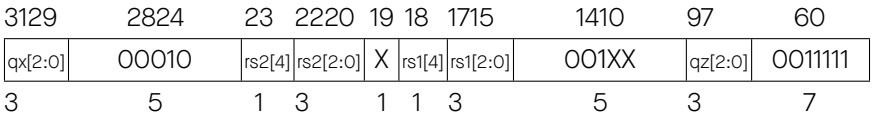


Figure 4.7-174. ESP.VSAT.U8

4.7.2.175 ESP.VSSUBS.S16

Syntax

ESP.VSSUBS.S16 qv,qx,rs1,sat

Description

This instruction performs a signed vector subtraction on 16-bit data.

Operation

```
qv[ 15:  0] = min( max( qx[ 15:  0] − rs1[15:0], -2^{15} ), 2^{15}-1 )
qv[ 31: 16] = min( max( qx[ 31: 16] − rs1[15:0], -2^{15} ), 2^{15}-1 )
qv[ 47: 32] = min( max( qx[ 47: 32] − rs1[15:0], -2^{15} ), 2^{15}-1 )
qv[ 63: 48] = min( max( qx[ 63: 48] − rs1[15:0], -2^{15} ), 2^{15}-1 )
qv[ 79: 64] = min( max( qx[ 79: 64] − rs1[15:0], -2^{15} ), 2^{15}-1 )
qv[ 95: 80] = min( max( qx[ 95: 80] − rs1[15:0], -2^{15} ), 2^{15}-1 )
qv[111: 96] = min( max( qx[111: 96] − rs1[15:0], -2^{15} ), 2^{15}-1 )
qv[127:112] = min( max( qx[127:112] − rs1[15:0], -2^{15} ), 2^{15}-1 )
```

Schedule

Operands	Use	Def
qv	-	1
qx	1	-
rs1	1	-

Format

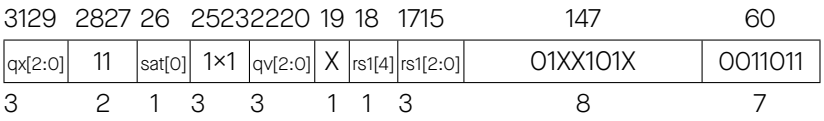


Figure 4.7-175. ESP.VSSUBS.S16

4.7.2.176 ESP.VSSUBS.S8

Syntax

ESP.VSSUBS.S8 qv,qx,rs1,sat

Description

This instruction performs a signed vector subtraction on 8-bit data.

Operation

```
qv[  7:  0] = min( max( qx[  7:  0] □ rs1[7:0], -2^{7} ), 2^{7}-1 )
qv[ 15:  8] = min( max( qx[ 15:  8] □ rs1[7:0], -2^{7} ), 2^{7}-1 )
qv[ 23: 16] = min( max( qx[ 23: 16] □ rs1[7:0], -2^{7} ), 2^{7}-1 )
qv[ 31: 24] = min( max( qx[ 31: 24] □ rs1[7:0], -2^{7} ), 2^{7}-1 )
qv[ 39: 32] = min( max( qx[ 39: 32] □ rs1[7:0], -2^{7} ), 2^{7}-1 )
qv[ 47: 40] = min( max( qx[ 47: 40] □ rs1[7:0], -2^{7} ), 2^{7}-1 )
qv[ 55: 48] = min( max( qx[ 55: 48] □ rs1[7:0], -2^{7} ), 2^{7}-1 )
qv[ 63: 56] = min( max( qx[ 63: 56] □ rs1[7:0], -2^{7} ), 2^{7}-1 )
qv[ 71: 64] = min( max( qx[ 71: 64] □ rs1[7:0], -2^{7} ), 2^{7}-1 )
qv[ 79: 72] = min( max( qx[ 79: 72] □ rs1[7:0], -2^{7} ), 2^{7}-1 )
qv[ 87: 80] = min( max( qx[ 87: 80] □ rs1[7:0], -2^{7} ), 2^{7}-1 )
qv[ 95: 88] = min( max( qx[ 95: 88] □ rs1[7:0], -2^{7} ), 2^{7}-1 )
qv[103: 96] = min( max( qx[103: 96] □ rs1[7:0], -2^{7} ), 2^{7}-1 )
qv[111:104] = min( max( qx[111:104] □ rs1[7:0], -2^{7} ), 2^{7}-1 )
qv[119:112] = min( max( qx[119:112] □ rs1[7:0], -2^{7} ), 2^{7}-1 )
qv[127:120] = min( max( qx[127:120] □ rs1[7:0], -2^{7} ), 2^{7}-1 )
```

Schedule

Operands	Use	Def
qv	-	1
qx	1	-
rs1	1	-

Format

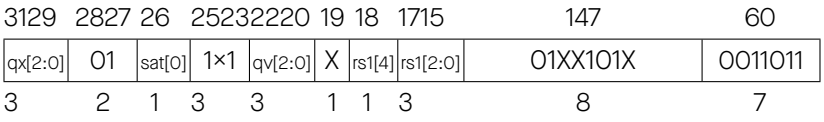


Figure 4.7-176. ESP.VSSUBS.S8

4.7.2.177 ESP.VSSUBS.U16

Syntax

ESP.VSSUBS.U16 qv,qx,rs1,sat

Description

This instruction performs an unsigned vector subtraction on 16-bit data.

Operation

```
qv[ 15:  0] = min( qx[ 15:  0] □ rs1[15:0], 2^{16}-1 )
qv[ 31: 16] = min( qx[ 31: 16] □ rs1[15:0], 2^{16}-1 )
qv[ 47: 32] = min( qx[ 47: 32] □ rs1[15:0], 2^{16}-1 )
qv[ 63: 48] = min( qx[ 63: 48] □ rs1[15:0], 2^{16}-1 )
qv[ 79: 64] = min( qx[ 79: 64] □ rs1[15:0], 2^{16}-1 )
qv[ 95: 80] = min( qx[ 95: 80] □ rs1[15:0], 2^{16}-1 )
qv[111: 96] = min( qx[111: 96] □ rs1[15:0], 2^{16}-1 )
qv[127:112] = min( qx[127:112] □ rs1[15:0], 2^{16}-1 )
```

Schedule

Operands	Use	Def
qv	-	1
qx	1	-
rs1	1	-

Format

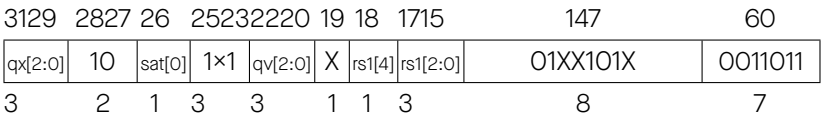


Figure 4.7-177. ESP.VSSUBS.U16

4.7.2.178 ESP.VSSUBS.U8

Syntax

ESP.VSSUBS.U8 qv,qx,rs1,sat

Description

This instruction performs a signed vector subtraction on 8-bit data.

Operation

```
qv[  7:  0] = min( qx[  7:  0] - rs1[7:0], 2^{8}-1 )
qv[ 15:  8] = min( qx[ 15:  8] - rs1[7:0], 2^{8}-1 )
qv[ 23: 16] = min( qx[ 23: 16] - rs1[7:0], 2^{8}-1 )
qv[ 31: 24] = min( qx[ 31: 24] - rs1[7:0], 2^{8}-1 )
qv[ 39: 32] = min( qx[ 39: 32] - rs1[7:0], 2^{8}-1 )
qv[ 47: 40] = min( qx[ 47: 40] - rs1[7:0], 2^{8}-1 )
qv[ 55: 48] = min( qx[ 55: 48] - rs1[7:0], 2^{8}-1 )
qv[ 63: 56] = min( qx[ 63: 56] - rs1[7:0], 2^{8}-1 )
qv[ 71: 64] = min( qx[ 71: 64] - rs1[7:0], 2^{8}-1 )
qv[ 79: 72] = min( qx[ 79: 72] - rs1[7:0], 2^{8}-1 )
qv[ 87: 80] = min( qx[ 87: 80] - rs1[7:0], 2^{8}-1 )
qv[ 95: 88] = min( qx[ 95: 88] - rs1[7:0], 2^{8}-1 )
qv[103: 96] = min( qx[103: 96] - rs1[7:0], 2^{8}-1 )
qv[111:104] = min( qx[111:104] - rs1[7:0], 2^{8}-1 )
qv[119:112] = min( qx[119:112] - rs1[7:0], 2^{8}-1 )
qv[127:120] = min( qx[127:120] - rs1[7:0], 2^{8}-1 )
```

Schedule

Operands	Use	Def
qv	-	1
qx	1	-
rs1	1	-

Format

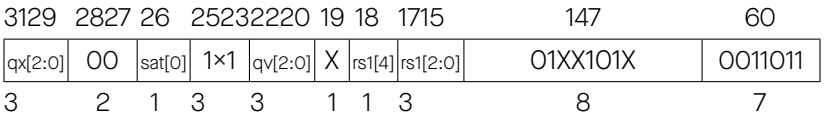


Figure 4.7-178. ESP.VSSUBS.U8

4.7.2.179 ESP.VSUB.S16

Syntax

ESP.VSUB.S16 qv,qx,qy,sat

Description

This instruction performs a vector subtraction on 16-bit data. Registers qx and qy are the minuend and the subtrahend respectively. Then, the 8 results obtained from the calculation are saturated and then written into the qv register.

Operation

```
qv[ 15:  0] = min(max(qx[ 15:  0] - qy[ 15:  0], -2^{15}), 2^{15}-1)
qv[ 31: 16] = min(max(qx[ 31: 16] - qy[ 31: 16], -2^{15}), 2^{15}-1)
...
```

Schedule

Operands	Use	Def
qv	-	1
qx	1	-
qy	1	-

Format

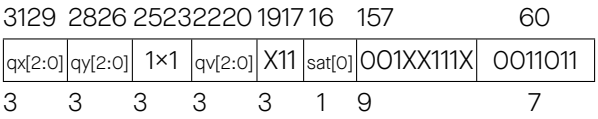


Figure 4.7-179. ESP.VSUB.S16

4.7.2.180 ESP.VSUB.S16.LD.INCP

Syntax

ESP.VSUB.S16.LD.INCP qu,rs1,qv,qx,qy,sat

Description

This instruction performs a vector subtraction on 16-bit data. Registers `qx` and `qy` are the subtrahend and the minuend, respectively. Then, the 8 results obtained from the calculation are saturated and then written into the register `qv`.

During the operation, the 16-byte data is loaded from the memory into the register `qu`. After the access, the value in the register `rs1` is incremented by 16.

Operation

```
qv[ 15:  0] = min(max(qx[ 15:  0] - qy[ 15:  0], -2^{15}), 2^{15}-1)
qv[ 31: 16] = min(max(qx[ 31: 16] - qy[ 31: 16], -2^{15}), 2^{15}-1)
...

qu[127:0] = load128(rs1[31:0])
rs1[31:0] = rs1[31:0] + 16
```

Schedule

Operands	Use	Def
qv	-	1
qx	1	-
qy	1	-
qu	-	2
rs1	1	1

Format

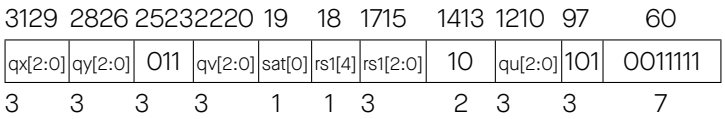


Figure 4.7-180. ESP.VSUB.S16.LD.INCP

4.7.2.181 ESP.VSUB.S16.ST.INCP

Syntax

ESP.VSUB.S16.ST.INCP qu,rs1,qv,qx,qy,sat

Description

This instruction performs a vector subtraction on 16-bit data. Registers `qx` and `qy` are the subtrahend and the minuend respectively. Then, the 8 results obtained from the calculation are saturated and then written into the register `qv`.

During the operation, the value in the register `qu` is stored to memory. After the access, the value in the register `rs1` is incremented by 16.

Operation

```
qv[ 15:  0] = min(max(qx[ 15:  0] - qy[ 15:  0], -2^{15}), 2^{15}-1)
qv[ 31: 16] = min(max(qx[ 31: 16] - qy[ 31: 16], -2^{15}), 2^{15}-1)
...

qu[127:0] => store128(rs1[31:0])
rs1[31:0] = rs1[31:0] + 16
```

Schedule

Operands	Use	Def
qv	-	1
qx	1	-
qy	1	-
qu	2	-
rs1	1	1

Format

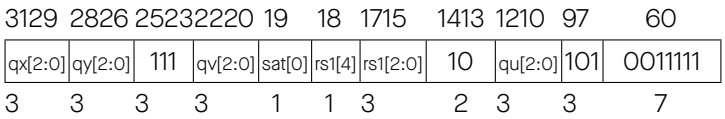


Figure 4.7-181. ESP.VSUB.S16.ST.INCP

4.7.2.182 ESP.VSUB.S32

Syntax

ESP.VSUB.S32 qv,qx,qy,sat

Description

This instruction performs a vector subtraction on 32-bit data. Registers qx and qy are the minuend and the subtrahend respectively. Then, the 4 results obtained from the calculation are saturated and then written into the register qv.

Operation

```
qv[ 31:  0] = min(max(qx[ 31:  0] - qy[ 31:  0], -2^{31}), 2^{31}-1)
qv[ 63: 32] = min(max(qx[ 63: 32] - qy[ 63: 32], -2^{31}), 2^{31}-1)
...
```

Schedule

Operands	Use	Def
qv	-	1
qx	1	-
qy	1	-

Format

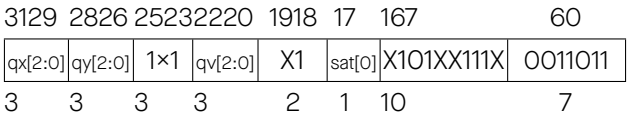


Figure 4.7-182. ESP.VSUB.S32

4.7.2.183 ESP.VSUB.S32.LD.INCP

Syntax

ESP.VSUB.S32.LD.INCP qu,rs1,qv,qx,qy,sat

Description

This instruction performs a vector subtraction on 32-bit data. Registers `qx` and `qy` are the subtrahend and the minuend, respectively. Then, the 4 results obtained from the calculation are saturated and then written into the register `qv`.

During the operation, the 16-byte data is loaded from the memory into the register `qu`. After the access, the value in the register `rs1` is incremented by 16.

Operation

```
qv[31: 0] = min(max(qx[31: 0] - qy[31: 0], -2^{31}), 2^{31}-1)
qv[63:32] = min(max(qx[63:32] - qy[63:32], -2^{31}), 2^{31}-1)
...

qu[127:0] = load128(rs1[31:0])
rs1[31:0] = rs1[31:0] + 16
```

Schedule

Operands	Use	Def
qv	-	1
qx	1	-
qy	1	-
qu	-	2
rs1	1	1

Format

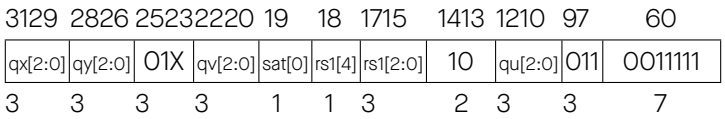


Figure 4.7-183. ESP.VSUB.S32.LD.INCP

4.7.2.184 ESP.VSUB.S32.ST.INCP

Syntax

ESP.VSUB.S32.ST.INCP qu,rs1,qv,qx,qy,sat

Description

This instruction performs a vector subtraction on 32-bit data. Registers `qx` and `qy` are the subtrahend and the minuend, respectively. Then, the 4 results obtained from the calculation are saturated and then written into the register `qv`.

During the operation, the value in the register `qu` is stored in memory. After the access, the value in the register `rs1` is incremented by 16.

Operation

```
qv[ 31:  0] = min(max(qx[ 31:  0] - qy[ 31:  0], -2^{31}), 2^{31}-1)
qv[ 63: 32] = min(max(qx[ 63: 32] - qy[ 63: 32], -2^{31}), 2^{31}-1)
...

qu[127:0] => store128(rs1[31:0])
rs1[31:0] = rs1[31:0] + 16
```

Schedule

Operands	Use	Def
qv	-	1
qx	1	-
qy	1	-
qu	2	-
rs1	1	1

Format

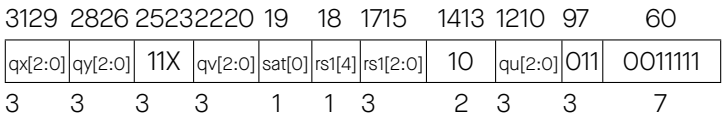


Figure 4.7-184. ESP.VSUB.S32.ST.INCP

4.7.2.185 ESP.VSUB.S8

Syntax

ESP.VSUB.S8 qv,qx,qy,sat

Description

This instruction performs a vector subtraction on 8-bit data. Registers qx and qy are the minuend and the subtrahend respectively. Then, the 16 results obtained from the calculation are saturated and then written into the register qv.

Operation

```
qv[  7:  0] = min(max(qx[  7:  0] - qy[  7:  0], -2^{7}), 2^{7}-1)
qv[ 15:  8] = min(max(qx[ 15:  8] - qy[ 15:  8], -2^{7}), 2^{7}-1)
...
```

Schedule

Operands	Use	Def
qv	-	1
qx	1	-
qy	1	-

Format

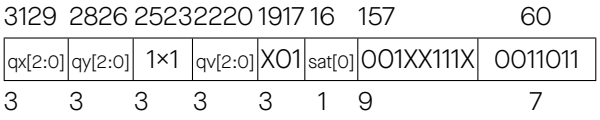


Figure 4.7-185. ESP.VSUB.S8

4.7.2.186 ESP.VSUB.S8.LD.INCP

Syntax

ESP.VSUB.S8.LD.INCP qu,rs1,qv,qx,qy,sat

Description

This instruction performs a vector subtraction on 8-bit data. Registers `qx` and `qy` are the subtrahend and the minuend, respectively. Then, the 16 results obtained from the calculation are saturated and then written into the register `qv`.

During the operation, the 16-byte data is loaded from the memory into the register `qu`. After the access, the value in the register `rs1` is incremented by 16.

Operation

```
qv[  7:  0] = min(max(qx[  7:  0] - qy[  7:  0], -2^{7}), 2^{7}-1)
qv[ 15:  8] = min(max(qx[ 15:  8] - qy[ 15:  8], -2^{7}), 2^{7}-1)
...

qu[127:0] = load128(rs1[31:0])
rs1[31:0] = rs1[31:0] + 16
```

Schedule

Operands	Use	Def
qv	-	1
qx	1	-
qy	1	-
qu	-	2
rs1	1	1

Format

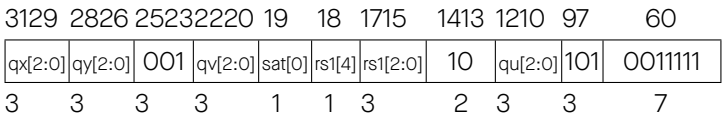


Figure 4.7-186. ESP.VSUB.S8.LD.INCP

4.7.2.187 ESP.VSUB.S8.ST.INCP

Syntax

ESP.VSUB.S8.ST.INCP qu,rs1,qv,qx,qy,sat

Description

This instruction performs a vector subtraction on 8-bit data. Registers `qx` and `qy` are the subtrahend and the minuend respectively. Then, the 16 results obtained from the calculation are saturated and then written into the register `qv`.

During the operation, the value in the register `qu` is stored to memory. After the access, the value in the register `rs1` is incremented by 16.

Operation

```
qv[  7:  0] = min(max(qx[  7:  0] - qy[  7:  0], -2^{7}), 2^{7}-1)
qv[ 15:  8] = min(max(qx[ 15:  8] - qy[ 15:  8], -2^{7}), 2^{7}-1)
...

qu[127:0] => store128(rs1[31:0])
rs1[31:0] = rs1[31:0] + 16
```

Schedule

Operands	Use	Def
qv	-	1
qx	1	-
qy	1	-
qu	2	-
rs1	1	1

Format

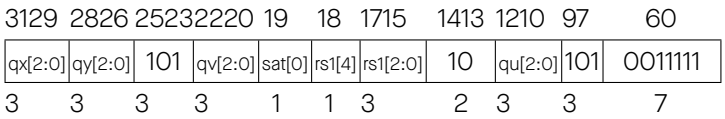


Figure 4.7-187. ESP.VSUB.S8.ST.INCP

4.7.2.188 ESP.VSUB.U16

Syntax

ESP.VSUB.U16 qv,qx,qy,sat

Description

This instruction performs a vector subtraction on 16-bit unsigned data. Registers qx and qy are the minuend and the subtrahend respectively. Then, the 8 results obtained from the calculation are saturated and then written into the qv register.

Operation

```
qv[ 15:  0] = min( qx[ 15:  0] - qy[ 15:  0], 2^{16}-1)
qv[ 31: 16] = min( qx[ 31: 16] - qy[ 31: 16], 2^{16}-1)
...
```

Schedule

Operands	Use	Def
qv	-	1
qx	1	-
qy	1	-

Format

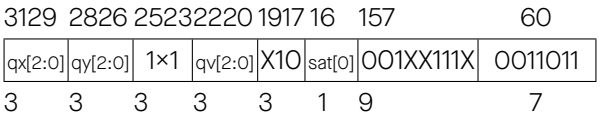


Figure 4.7-188. ESP.VSUB.U16

4.7.2.189 ESP.VSUB.U16.LD.INCP

Syntax

ESP.VSUB.U16.LD.INCP qu,rs1,qv,qx,qy,sat

Description

This instruction performs a vector subtraction on 16-bit unsigned data. Registers `qx` and `qy` are the subtrahend and the minuend, respectively. Then, the 8 results obtained from the calculation are saturated and then written into the register `qv`.

During the operation, the 16-byte data is loaded from the memory into the register `qu`. After the access, the value in the register `rs1` is incremented by 16.

Operation

```
qv[ 15:  0] = min(qx[ 15:  0] - qy[ 15:  0], 2^{16}-1)
qv[ 31: 16] = min(qx[ 31: 16] - qy[ 31: 16], 2^{16}-1)
...

qu[127:0] = load128(rs1[31:0])
rs1[31:0] = rs1[31:0] + 16
```

Schedule

Operands	Use	Def
qv	-	1
qx	1	-
qy	1	-
qu	-	2
rs1	1	1

Format

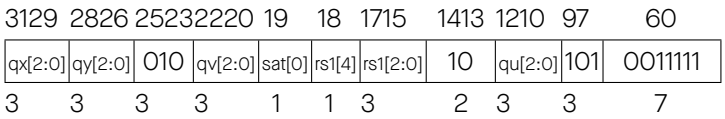


Figure 4.7-189. ESP.VSUB.U16.LD.INCP

4.7.2.190 ESP.VSUB.U16.ST.INCP

Syntax

ESP.VSUB.U16.ST.INCP qu,rs1,qv,qx,qy,sat

Description

This instruction performs a vector subtraction on 16-bit unsigned data. Registers `qx` and `qy` are the subtrahend and the minuend, respectively. Then, the 8 results obtained from the calculation are saturated and then written into the register `qv`.

During the operation, the value in the register `qu` is stored to memory. After the access, the value in the register `rs1` is incremented by 16.

Operation

```
qv[ 15:  0] = min(qx[ 15:  0] - qy[ 15:  0], 2^{16}-1)
qv[ 31: 16] = min(qx[ 31: 16] - qy[ 31: 16], 2^{16}-1)
...

qu[127:0] => store128(rs1[31:0])
rs1[31:0] = rs1[31:0] + 16
```

Schedule

Operands	Use	Def
qv	-	1
qx	1	-
qy	1	-
qu	2	-
rs1	1	1

Format

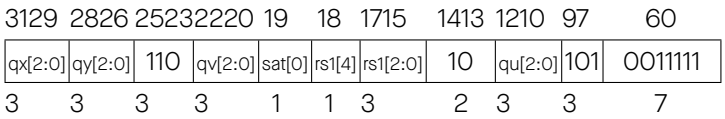


Figure 4.7-190. ESP.VSUB.U16.ST.INCP

4.7.2.191 ESP.VSUB.U32

Syntax

ESP.VSUB.U32 qv,qx,qy,sat

Description

This instruction performs a vector subtraction on 32-bit unsigned data. Registers qx and qy are the minuend and the subtrahend respectively. Then, the 4 results obtained from the calculation are saturated and then written into the register qv.

Operation

```
qv[ 31:  0] = min(qx[ 31:  0] - qy[ 31:  0], 2^{32}-1)
qv[ 63: 32] = min(qx[ 63: 32] - qy[ 63: 32], 2^{32}-1)
...
```

Schedule

Operands	Use	Def
qv	-	1
qx	1	-
qy	1	-

Format

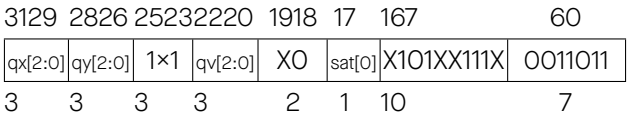


Figure 4.7-191. ESP.VSUB.U32

4.7.2.192 ESP.VSUB.U32.LD.INCP

Syntax

ESP.VSUB.U32.LD.INCP qu,rs1,qv,qx,qy,sat

Description

This instruction performs a vector subtraction on 32-bit unsigned data. Registers `qx` and `qy` are the subtrahend and the minuend, respectively. Then, the 4 results obtained from the calculation are saturated and then written into the register `qv`.

During the operation, the 16-byte data is loaded from the memory into the register `qu`. After the access, the value in the register `rs1` is incremented by 16.

Operation

```
qv[31: 0] = min(qx[31: 0] - qy[31: 0], 2^{32}-1)
qv[63:32] = min(qx[63:32] - qy[63:32], 2^{32}-1)
...

qu[127:0] = load128(rs1[31:0])
rs1[31:0] = rs1[31:0] + 16
```

Schedule

Operands	Use	Def
qv	-	1
qx	1	-
qy	1	-
qu	-	2
rs1	1	1

Format

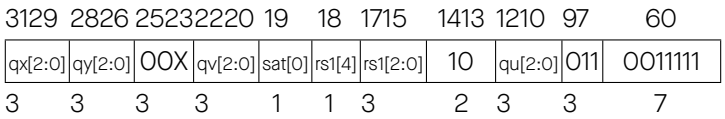


Figure 4.7-192. ESP.VSUB.U32.LD.INCP

4.7.2.193 ESP.VSUB.U32.ST.INCP

Syntax

ESP.VSUB.U32.ST.INCP qu,rs1,qv,qx,qy,sat

Description

This instruction performs a vector subtraction on 32-bit unsigned data. Registers `qx` and `qy` are the subtrahend and the minuend, respectively. Then, the 4 results obtained from the calculation are saturated and then written into the register `qv`.

During the operation, the value in the register `qu` is stored to memory. After the access, the value in the register `rs1` is incremented by 16.

Operation

```
qv[ 31:  0] = min(qx[ 31:  0] - qy[ 31:  0], 2^{32}-1)
qv[ 63: 32] = min(qx[ 63: 32] - qy[ 63: 32], 2^{32}-1)
...

qu[127:0] => store128(rs1[31:0])
rs1[31:0] = rs1[31:0] + 16
```

Schedule

Operands	Use	Def
qv	-	1
qx	1	-
qy	1	-
qu	2	-
rs1	1	1

Format

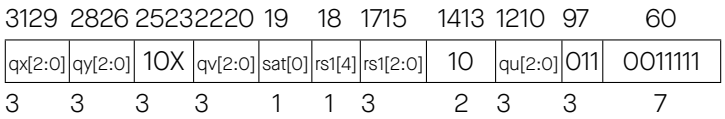


Figure 4.7-193. ESP.VSUB.U32.ST.INCP

4.7.2.194 ESP.VSUB.U8

Syntax

ESP.VSUB.U8 qv,qx,qy,sat

Description

This instruction performs a vector subtraction on 8-bit unsigned data. Registers `qx` and `qy` are the minuend and the subtrahend respectively. Then, the 16 results obtained from the calculation are saturated and then written into the register `qv`.

Operation

```
qv[  7:  0] = min(qx[  7:  0] - qy[  7:  0], 2^{8}-1)
qv[ 15:  8] = min(qx[ 15:  8] - qy[ 15:  8], 2^{8}-1)
...
```

Schedule

Operands	Use	Def
qv	-	1
qx	1	-
qy	1	-

Format

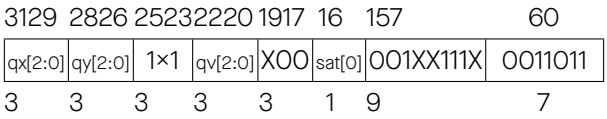


Figure 4.7-194. ESP.VSUB.U8

4.7.2.195 ESP.VSUB.U8.LD.INCP

Syntax

ESP.VSUB.U8.LD.INCP qu,rs1,qv,qx,qy,sat

Description

This instruction performs a vector subtraction on 8-bit unsigned data. Registers `qx` and `qy` are the subtrahend and the minuend, respectively. Then, the 16 results obtained from the calculation are saturated and then written into register `qv`.

During the operation, the 16-byte data is loaded from the memory into register `qu`. After the access, the value in register `rs1` is incremented by 16.

Operation

```
qv[  7:  0] = min(qx[  7:  0] - qy[  7:  0], 2^{8}-1)
qv[ 15:  8] = min(qx[ 15:  8] - qy[ 15:  8], 2^{8}-1)
...

qu[127:0] = load128(rs1[31:0])
rs1[31:0] = rs1[31:0] + 16
```

Schedule

Operands	Use	Def
qv	-	1
qx	1	-
qy	1	-
qu	-	2
rs1	1	1

Format

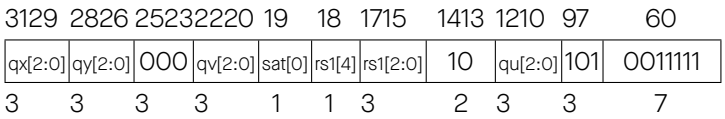


Figure 4.7-195. ESP.VSUB.U8.LD.INCP

4.7.2.196 ESP.VSUB.U8.ST.INCP

Syntax

ESP.VSUB.U8.ST.INCP qu,rs1,qv,qx,qy,sat

Description

This instruction performs a vector subtraction on 8-bit unsigned data. Registers `qx` and `qy` are the subtrahend and the minuend, respectively. Then, the 16 results obtained from the calculation are saturated and then written into the register `qv`.

During the operation, the value in the register `qu` is stored to memory. After the access, the value in the register `rs1` is incremented by 16.

Operation

```
qv[  7:  0] = min(qx[  7:  0] - qy[  7:  0], 2^{8}-1)
qv[ 15:  8] = min(qx[ 15:  8] - qy[ 15:  8], 2^{8}-1)
...

qu[127:0] => store128(rs1[31:0])
rs1[31:0] = rs1[31:0] + 16
```

Schedule

Operands	Use	Def
qv	-	1
qx	1	-
qy	1	-
qu	2	-
rs1	1	1

Format

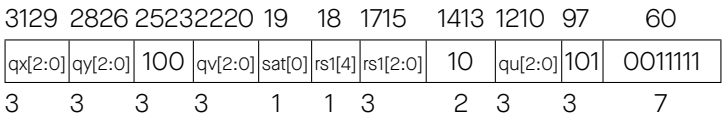


Figure 4.7-196. ESP.VSUB.U8.ST.INCP

4.7.2.197 ESP.ADDX2

Syntax

ESP.ADDX2 rd,rs1,rs2

Description

This instruction shifts `rs2` to the left by 1 and adds it to `rs1`.

Operation

```
rd = rs1 + rs2<<1
```

Schedule

Operands	Use	Def
rd	-	1
rs1	1	-
rs2	1	-

Format

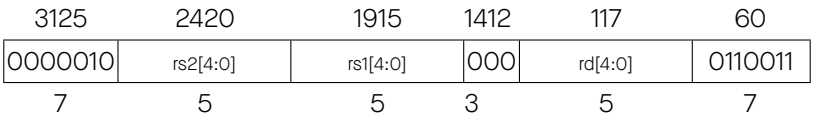


Figure 4.7-197. ESP.ADDX2

4.7.2.198 ESP.ADDX4

Syntax

ESP.ADDX4 rd,rs1,rs2

Description

This instruction shifts `rs2` to the left by 2 and adds it to `rs1`.

Operation

```
rd = rs1 + rs2<<2
```

Schedule

Operands	Use	Def
rd	-	1
rs1	1	-
rs2	1	-

Format

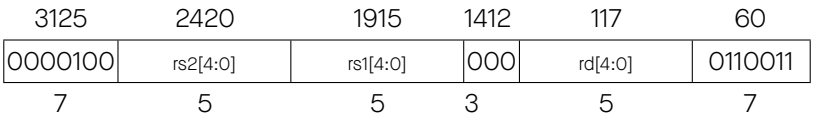


Figure 4.7-198. ESP.ADDX4

4.7.2.199 ESP.SAT

Syntax

ESP.SAT rsd,rs0,rs1

Description

This instruction performs saturation for `rsd`.

Operation

```
    min_t[31:0] = min( rs1[31:0], rs0[31:0] )
    max_t[31:0] = max( rs1[31:0], rs0[31:0] )

    rsd[31:0] = max( min( rsd[31:0], max_t[31:0]), min_t[31:0] )
```

Schedule

Operands	Use	Def
rsd	1	1
rs0	1	-
rs1	1	-

Format

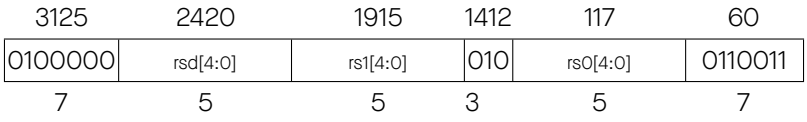


Figure 4.7-199. ESP.SAT

4.7.2.200 ESP.SUBX2

Syntax

ESP.SUBX2 rd,rs1,rs2

Description

This instruction shifts `rs2` to the left by 1 and subtracts it from `rs1`.

Operation

```
rd = rs1 - rs2<<1
```

Schedule

Operands	Use	Def
rd	-	1
rs1	1	-
rs2	1	-

Format

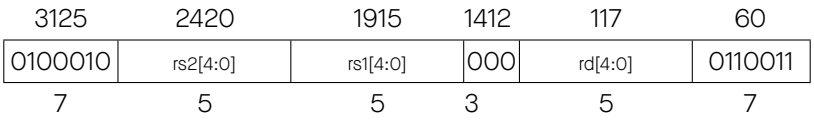


Figure 4.7-200. ESP.SUBX2

4.7.2.201 ESP.SUBX4

Syntax

ESP.SUBX4 rd,rs1,rs2

Description

This instruction shifts `rs2` to the left by 2 and subtracts it from `rs1`.

Operation

```
rd = rs1 - rs2<<2
```

Schedule

Operands	Use	Def
rd	-	1
rs1	1	-
rs2	1	-

Format

3125	2420	1915	1412	117	60
0100100	rs2[4:0]	rs1[4:0]	000	rd[4:0]	0110011
7	5	5	3	5	7

Figure 4.7-201. ESP.SUBX4

4.7.2.202 ESP.ANDQ

Syntax

ESP.ANDQ qz,qx,qy

Description

This instruction performs a bitwise AND operation on registers qx and qy, and writes the result of the logical operation to the register qz.

Operation

```
qz = qx & qy
```

Schedule

Operands	Use	Def
qz	-	1
qx	1	-
qy	1	-

Format

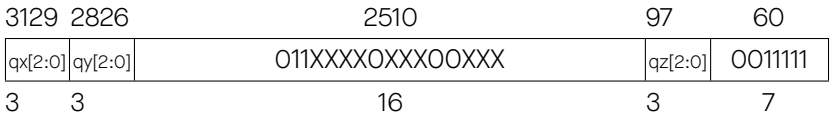


Figure 4.7-202. ESP.ANDQ

4.7.2.203 ESP.NOTQ

Syntax

ESP.NOTQ qz,qx

Description

This instruction performs a bitwise NOT operation on the register qx and writes the result to the register qz.

Operation

```
qz = ~qx
```

Schedule

Operands	Use	Def
qz	-	1
qx	1	-

Format

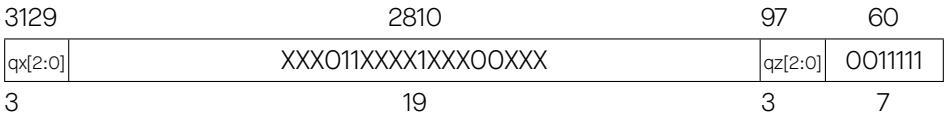


Figure 4.7-203. ESP.NOTQ

4.7.2.204 ESP.ORQ

Syntax

ESP.ORQ qz,qx,qy

Description

This instruction performs a bitwise or operation on registers qx and qy, and writes the result of the logical operation to the register qz.

Operation

```
qz = qx | qy
```

Schedule

Operands	Use	Def
qz	-	1
qx	1	-
qy	1	-

Format

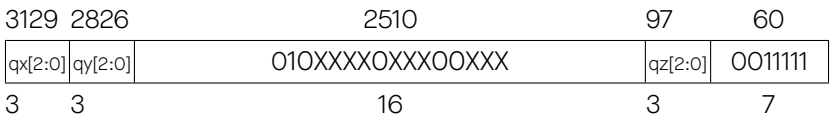


Figure 4.7-204. ESP.ORQ

4.7.2.205 ESP.XORQ

Syntax

ESP.XORQ qz,qx,qy

Description

This instruction performs a bitwise xor operation on registers `qx` and `qy`, and writes the result of the logical operation to the register `qz`.

Operation

```
qz = qx ^ qy
```

Schedule

Operands	Use	Def
qz	-	1
qx	1	-
qy	1	-

Format

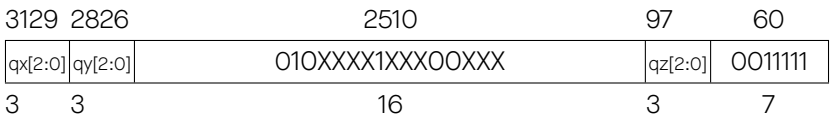


Figure 4.7-205. ESP.XORQ

4.7.2.206 ESP.VCMP.EQ.S16

Syntax

ESP.VCMP.EQ.S16 qz,qx,qy

Description

This instruction compares 16-bit vector data. It compares the numerical values of the eight 16-bit data segments in registers `qx` and `qy`. If the values are equal, it writes 0xFFFF into the corresponding 16-bit data segment in register `qz`. Otherwise, it writes 0 into the segment.

Operation

```
qz[ 15:  0] = (qx[ 15:  0]==qy[ 15:  0]) ? 0xFFFF : 0
qz[ 31: 16] = (qx[ 31: 16]==qy[ 31: 16]) ? 0xFFFF : 0
...
qz[127:112] = (qx[127:112]==qy[127:112]) ? 0xFFFF : 0
```

Schedule

Operands	Use	Def
qz	-	1
qx	1	-
qy	1	-

Format

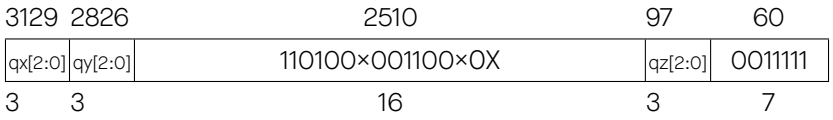


Figure 4.7-206. ESP.VCMP.EQ.S16

4.7.2.207 ESP.VCMP.EQ.S32

Syntax

ESP.VCMP.EQ.S32 qz,qx,qy

Description

This instruction compares 32-bit vector data. It compares the numerical values of the four 32-bit data segments in registers qx and qy. If the values are equal, it writes 0xFFFFFFFF into the corresponding 32-bit data segment in the qz register. Otherwise, it writes 0 into the segment.

Operation

```
qz[ 31:  0] = (qx[ 31:  0]==qy[ 31:  0]) ? 0xFFFFFFFF : 0
qz[ 63: 32] = (qx[ 63: 32]==qy[ 63: 32]) ? 0xFFFFFFFF : 0
...
qz[127: 96] = (qx[127: 96]==qy[127: 96]) ? 0xFFFFFFFF : 0
```

Schedule

Operands	Use	Def
qz	-	1
qx	1	-
qy	1	-

Format

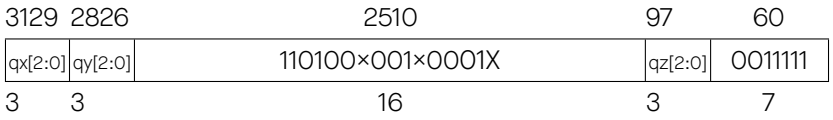


Figure 4.7-207. ESP.VCMP.EQ.S32

4.7.2.208 ESP.VCMP.EQ.S8

Syntax

ESP.VCMP.EQ.S8 qz,qx,qy

Description

This instruction compares 8-bit vector data. It compares the numerical values of the 16 8-bit data segments in registers qx and qy. If the values are equal, it writes 0xFF into the corresponding 8-bit data segment in register qz. Otherwise, it writes 0 into the segment.

Operation

```
qz[  7:  0] = (qx[  7:  0]==qy[  7:  0]) ? 0xFF : 0
qz[ 15:  8] = (qx[ 15:  8]==qy[ 15:  8]) ? 0xFF : 0
...
qz[127:120] = (qx[127:120]==qy[127:120]) ? 0xFF : 0
```

Schedule

Operands	Use	Def
qz	-	1
qx	1	-
qy	1	-

Format

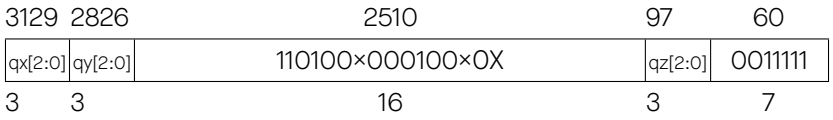


Figure 4.7-208. ESP.VCMP.EQ.S8

4.7.2.209 ESP.VCMP.EQ.U16

Syntax

ESP.VCMP.EQ.U16 qz,qx,qy

Description

This instruction compares 16-bit vector data. It compares the numerical values of the eight 16-bit data segments in the registers `qx` and `qy`. If the values are equal, it writes 0xFFFF into the corresponding 16-bit data segment in the register `qz`. Otherwise, it writes 0 into the segment.

Operation

```
qz[ 15:  0] = (qx[ 15:  0]==qy[ 15:  0]) ? 0xFFFF : 0
qz[ 31: 16] = (qx[ 31: 16]==qy[ 31: 16]) ? 0xFFFF : 0
...
qz[127:112] = (qx[127:112]==qy[127:112]) ? 0xFFFF : 0
```

Schedule

Operands	Use	Def
qz	-	1
qx	1	-
qy	1	-

Format

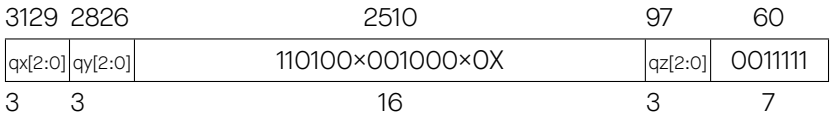


Figure 4.7-209. ESP.VCMP.EQ.U16

4.7.2.210 ESP.VCMP.EQ.U32

Syntax

ESP.VCMP.EQ.U32 qz,qx,qy

Description

This instruction compares 32-bit vector data. It compares the numerical values of the four 32-bit data segments in the registers qx and qy. If the values are equal, it writes 0xFFFFFFFF into the corresponding 32-bit data segment in the register qz. Otherwise, it writes 0 into the segment.

Operation

```
qz[ 31:  0] = (qx[ 31:  0]==qy[ 31:  0]) ? 0xFFFFFFFF : 0
qz[ 63: 32] = (qx[ 63: 32]==qy[ 63: 32]) ? 0xFFFFFFFF : 0
...
qz[127: 96] = (qx[127: 96]==qy[127: 96]) ? 0xFFFFFFFF : 0
```

Schedule

Operands	Use	Def
qz	-	1
qx	1	-
qy	1	-

Format

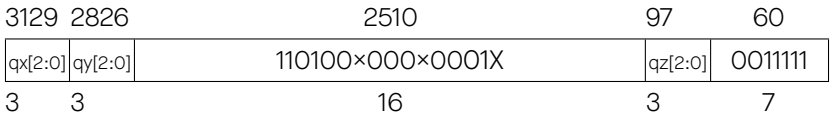


Figure 4.7-210. ESP.VCMP.EQ.U32

4.7.2.211 ESP.VCMP.EQ.U8

Syntax

ESP.VCMP.EQ.U8 qz,qx,qy

Description

This instruction compares 8-bit vector data. It compares the numerical values of the 16 8-bit data segments in registers qx and qy. If the values are equal, it writes 0xFF into the corresponding 8-bit data segment in the qz register. Otherwise, it writes 0 into the segment.

Operation

```
qz[  7:  0] = (qx[  7:  0]==qy[  7:  0]) ? 0xFF : 0
qz[ 15:  8] = (qx[ 15:  8]==qy[ 15:  8]) ? 0xFF : 0
...
qz[127:120] = (qx[127:120]==qy[127:120]) ? 0xFF : 0
```

Schedule

Operands	Use	Def
qz	-	1
qx	1	-
qy	1	-

Format

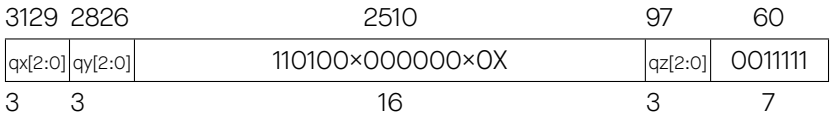


Figure 4.7-211. ESP.VCMP.EQ.U8

4.7.2.212 ESP.VCMP.GT.S16

Syntax

ESP.VCMP.GT.S16 qz,qx,qy

Description

This instruction compares 16-bit vector data. It compares the numerical values of the eight 16-bit data segments in registers qx and qy. If the former is larger than the latter, it writes 0xFFFF into the corresponding 16-bit data segment in register qz. Otherwise, it writes 0 into the segment.

Operation

```
qz[ 15:  0] = (qx[ 15:  0]>qy[ 15:  0]) ? 0xFFFF : 0
qz[ 31: 16] = (qx[ 31: 16]>qy[ 31: 16]) ? 0xFFFF : 0
...
qz[127:112] = (qx[127:112]>qy[127:112]) ? 0xFFFF : 0
```

Schedule

Operands	Use	Def
qz	-	1
qx	1	-
qy	1	-

Format

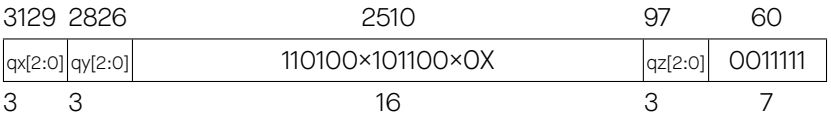


Figure 4.7-212. ESP.VCMP.GT.S16

4.7.2.213 ESP.VCMP.GT.S32

Syntax

ESP.VCMP.GT.S32 qz,qx,qy

Description

This instruction compares 32-bit signed vector data. It compares the numerical values of the four 32-bit data segments in the registers `qx` and `qy`. If the former is larger than the latter, it writes 0xFFFFFFFF into the corresponding 32-bit data segment in the register `qz`. Otherwise, it writes 0 into the segment.

Operation

```
qz[ 31:  0] = (qx[ 31:  0]>qy[ 31:  0]) ? 0xFFFFFFFF : 0
qz[ 63: 32] = (qx[ 63: 32]>qy[ 63: 32]) ? 0xFFFFFFFF : 0
...
qz[127: 96] = (qx[127: 96]>qy[127: 96]) ? 0xFFFFFFFF : 0
```

Schedule

Operands	Use	Def
qz	-	1
qx	1	-
qy	1	-

Format

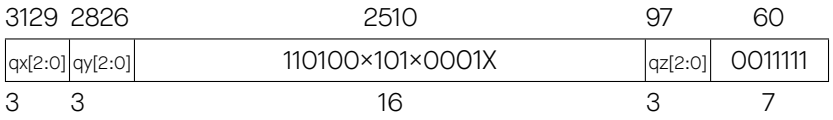


Figure 4.7-213. ESP.VCMP.GT.S32

4.7.2.214 ESP.VCMP.GT.S8

Syntax

ESP.VCMP.GT.S8 qz,qx,qy

Description

This instruction compares 8-bit signed vector data. It compares the numerical values of the 16 8-bit data segments in the registers `qx` and `qy`. If the former is larger than the latter, it writes 0xFF into the corresponding 8-bit data segment in the register `qz`. Otherwise, it writes 0 into the segment.

Operation

```
qz[  7:  0] = (qx[  7:  0]>qy[  7:  0]) ? 0xFF : 0
qz[ 15:  8] = (qx[ 15:  8]>qy[ 15:  8]) ? 0xFF : 0
...
qz[127:120] = (qx[127:120]>qy[127:120]) ? 0xFF : 0
```

Schedule

Operands	Use	Def
qz	-	1
qx	1	-
qy	1	-

Format

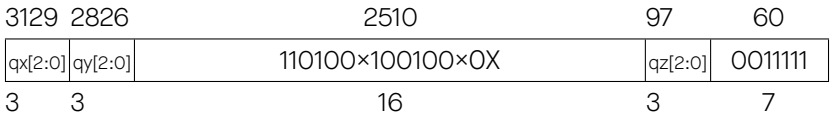


Figure 4.7-214. ESP.VCMP.GT.S8

4.7.2.215 ESP.VCMP.GT.U16

Syntax

ESP.VCMP.GT.U16 qz,qx,qy

Description

This instruction compares 16-bit unsigned vector data. It compares the numerical values of the eight 16-bit data segments in the registers `qx` and `qy`. If the former is larger than the latter, it writes 0xFFFF into the corresponding 16-bit data segment in the register `qz`. Otherwise, it writes 0 into the segment.

Operation

```
qz[ 15:  0] = (qx[ 15:  0]>qy[ 15:  0]) ? 0xFFFF : 0
qz[ 31: 16] = (qx[ 31: 16]>qy[ 31: 16]) ? 0xFFFF : 0
...
qz[127:112] = (qx[127:112]>qy[127:112]) ? 0xFFFF : 0
```

Schedule

Operands	Use	Def
qz	-	1
qx	1	-
qy	1	-

Format

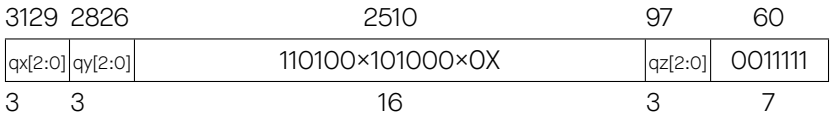


Figure 4.7-215. ESP.VCMP.GT.U16

4.7.2.216 ESP.VCMP.GT.U32

Syntax

ESP.VCMP.GT.U32 qz,qx,qy

Description

This instruction compares 32-bit unsigned vector data. It compares the numerical values of the four 32-bit data segments in registers `qx` and `qy`. If the former is larger than the latter, it writes 0xFFFFFFFF into the corresponding 32-bit data segment in the register `qz`. Otherwise, it writes 0 into the segment.

Operation

```
qz[ 31:  0] = (qx[ 31:  0]>qy[ 31:  0]) ? 0xFFFFFFFF : 0
qz[ 63: 32] = (qx[ 63: 32]>qy[ 63: 32]) ? 0xFFFFFFFF : 0
...
qz[127: 96] = (qx[127: 96]>qy[127: 96]) ? 0xFFFFFFFF : 0
```

Schedule

Operands	Use	Def
qz	-	1
qx	1	-
qy	1	-

Format

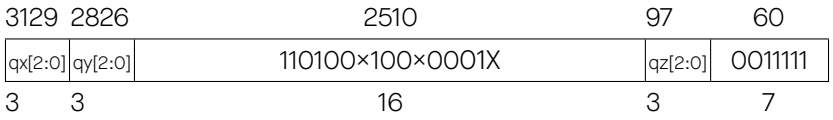


Figure 4.7-216. ESP.VCMP.GT.U32

4.7.2.217 ESP.VCMP.GT.U8

Syntax

ESP.VCMP.GT.U8 qz,qx,qy

Description

This instruction compares 8-bit unsigned vector data. It compares the numerical values of the 16 8-bit data segments in registers `qx` and `qy`. If the former is larger than the latter, it writes 0xFF into the corresponding 8-bit data segment in the register `qz`. Otherwise, it writes 0 into the segment.

Operation

```
qz[  7:  0] = (qx[  7:  0]>qy[  7:  0]) ? 0xFF : 0
qz[ 15:  8] = (qx[ 15:  8]>qy[ 15:  8]) ? 0xFF : 0
...
qz[127:120] = (qx[127:120]>qy[127:120]) ? 0xFF : 0
```

Schedule

Operands	Use	Def
qz	-	1
qx	1	-
qy	1	-

Format

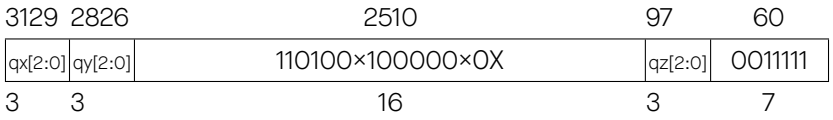


Figure 4.7-217. ESP.VCMP.GT.U8

4.7.2.218 ESP.VCMP.LT.S16

Syntax

ESP.VCMP.LT.S16 qz,qx,qy

Description

This instruction compares 16-bit signed vector data. It compares the numerical values of the eight 16-bit data segments in registers qx and qy. If the former is smaller than the latter, it writes 0xFFFF into the corresponding 16-bit data segment in register qz. Otherwise, it writes 0 into the segment.

Operation

```
qz[ 15:  0] = (qx[ 15:  0]<qy[ 15:  0]) ? 0xFFFF : 0
qz[ 31: 16] = (qx[ 31: 16]<qy[ 31: 16]) ? 0xFFFF : 0
...
qz[127:112] = (qx[127:112]<qy[127:112]) ? 0xFFFF : 0
```

Schedule

Operands	Use	Def
qz	-	1
qx	1	-
qy	1	-

Format

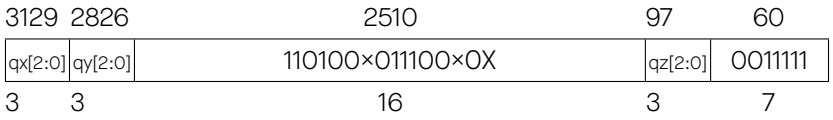


Figure 4.7-218. ESP.VCMP.LT.S16

4.7.2.219 ESP.VCMP.LT.S32

Syntax

ESP.VCMP.LT.S32 qz,qx,qy

Description

This instruction compares 32-bit signed vector data. It performs a comparison of the numerical values of the four 32-bit data segments in registers qx and qy. If the former is smaller than the latter, it writes 0xFFFFFFFF into the corresponding 32-bit data segment in register qz. Otherwise, it writes 0 into the segment.

Operation

```
qz[ 31:  0] = (qx[ 31:  0]<qy[ 31:  0]) ? 0xFFFFFFFF : 0
qz[ 63: 32] = (qx[ 63: 32]<qy[ 63: 32]) ? 0xFFFFFFFF : 0
...
qz[127: 96] = (qx[127: 96]<qy[127: 96]) ? 0xFFFFFFFF : 0
```

Schedule

Operands	Use	Def
qz	-	1
qx	1	-
qy	1	-

Format

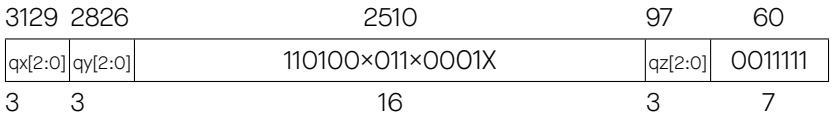


Figure 4.7-219. ESP.VCMP.LT.S32

4.7.2.220 ESP.VCMP.LT.S8

Syntax

ESP.VCMP.LT.S8 qz,qx,qy

Description

This instruction compares 8-bit signed vector data. It compares the numerical values of the 16 8-bit data segments in registers `qx` and `qy`. If the former is smaller than the latter, it writes 0xFF into the corresponding 8-bit data segment in register `qz`. Otherwise, it writes 0 into the segment.

Operation

```
qz[  7:  0] = (qx[  7:  0]<qy[  7:  0]) ? 0xFF : 0
qz[ 15:  8] = (qx[ 15:  8]<qy[ 15:  8]) ? 0xFF : 0
...
qz[127:120] = (qx[127:120]<qy[127:120]) ? 0xFF : 0
```

Schedule

Operands	Use	Def
qz	-	1
qx	1	-
qy	1	-

Format

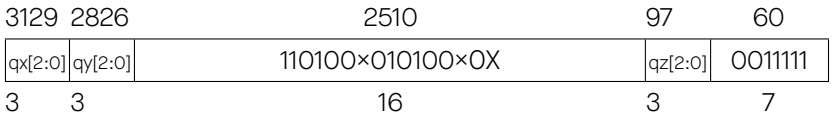


Figure 4.7-220. ESP.VCMP.LT.S8

4.7.2.221 ESP.VCMP.LT.U16

Syntax

ESP.VCMP.LT.U16 qz,qx,qy

Description

This instruction compares 16-bit unsigned vector data. It compares the numerical values of the eight 16-bit data segments in the registers `qx` and `qy`. If the former is smaller than the latter, it writes 0xFFFF into the corresponding 16-bit data segment in the register `qz`. Otherwise, it writes 0 into the segment.

Operation

```
qz[ 15:  0] = (qx[ 15:  0]<qy[ 15:  0]) ? 0xFFFF : 0
qz[ 31: 16] = (qx[ 31: 16]<qy[ 31: 16]) ? 0xFFFF : 0
...
qz[127:112] = (qx[127:112]<qy[127:112]) ? 0xFFFF : 0
```

Schedule

Operands	Use	Def
qz	-	1
qx	1	-
qy	1	-

Format

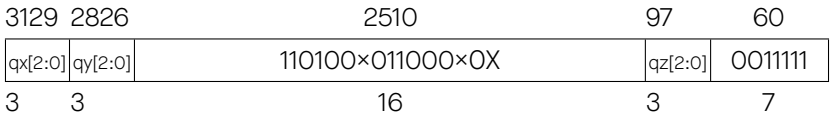


Figure 4.7-221. ESP.VCMP.LT.U16

4.7.2.222 ESP.VCMP.LT.U32

Syntax

ESP.VCMP.LT.U32 qz,qx,qy

Description

This instruction compares 32-bit unsigned vector data. It compares the numerical values of the four 32-bit data segments in registers `qx` and `qy`. If the former is smaller than the latter, it writes 0xFFFFFFFF into the corresponding 32-bit data segment in register `qz`. Otherwise, it writes 0 into the segment.

Operation

```
qz[ 31:  0] = (qx[ 31:  0]<qy[ 31:  0]) ? 0xFFFFFFFF : 0
qz[ 63: 32] = (qx[ 63: 32]<qy[ 63: 32]) ? 0xFFFFFFFF : 0
...
qz[127: 96] = (qx[127: 96]<qy[127: 96]) ? 0xFFFFFFFF : 0
```

Schedule

Operands	Use	Def
qz	-	1
qx	1	-
qy	1	-

Format

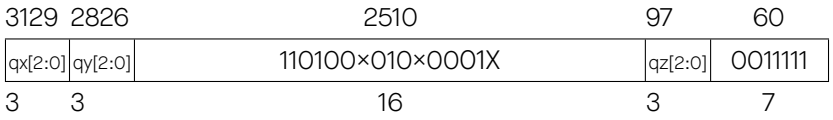


Figure 4.7-222. ESP.VCMP.LT.U32

4.7.2.223 ESP.VCMP.LT.U8

Syntax

ESP.VCMP.LT.U8 qz,qx,qy

Description

This instruction compares 8-bit unsigned vector data. It compares the numerical values of the 16 8-bit data segments in registers qx and qy. If the former is smaller than the latter, it writes 0xFF into the corresponding 8-bit data segment in register qz. Otherwise, it writes 0 into the segment.

Operation

```
qz[  7:  0] = (qx[  7:  0]<qy[  7:  0]) ? 0xFF : 0
qz[ 15:  8] = (qx[ 15:  8]<qy[ 15:  8]) ? 0xFF : 0
...
qz[127:120] = (qx[127:120]<qy[127:120]) ? 0xFF : 0
```

Schedule

Operands	Use	Def
qz	-	1
qx	1	-
qy	1	-

Format

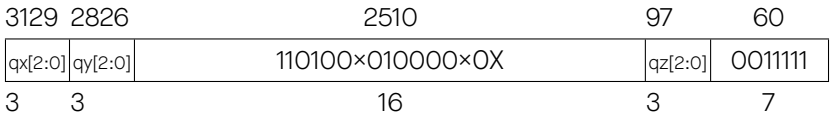


Figure 4.7-223. ESP.VCMP.LT.U8

4.7.2.224 ESP.MOV.S16.QACC

Syntax

ESP.MOV.S16.QACC qu

Description

This instruction signed-extends the 8 segments of 16-bit data in the register `qs` to 64 bits and writes the results to the special registers `QACC_H` and `QACC_L`.

Operation

```
QACC_L[ 63:  0] = {48{qu[15]}, qu[ 15:  0]}
QACC_L[127: 64] = {48{qu[31]}, qu[ 31: 16]}
QACC_L[191:128] = {48{qu[47]}, qu[ 47: 32]}
QACC_L[255:192] = {48{qu[63]}, qu[ 63: 48]}
QACC_H[ 39:  0] = {48{qu[79]}, qu[ 79: 64]}
QACC_H[ 79: 40] = {48{qu[95]}, qu[ 95: 80]}
QACC_H[119: 80] = {48{qu[111]}, qu[111: 96]}
QACC_H[159:120] = {48{qu[127]}, qu[127:112]}
```

Schedule

Operands	Use	Def
qu	2	-
QACC_H	-	2
QACC_L	-	2

Format



Figure 4.7-224. ESP.MOV.S16.QACC

4.7.2.225 ESP.MOV.S8.QACC

Syntax

ESP.MOV.S8.QACC qu

Description

This instruction sign-extends the 16 segments of 8-bit data in the register `qs` to 32 bits and writes the result to the special registers `QACC_H` and `QACC_L`.

Operation

```
QACC_L[ 31:  0] = {24{qu[7]}, qu[  7:  0]}
QACC_L[ 63: 32] = {24{qu[15]}, qu[ 15:  8]}
QACC_L[ 95: 64] = {24{qu[23]}, qu[ 23: 16]}
...
QACC_L[255:224] = {24{qu[63]}, qu[ 63: 56]}
QACC_H[ 31:  0] = {24{qu[71]}, qu[ 71: 64]}
QACC_H[ 63: 32] = {24{qu[79]}, qu[ 79: 72]}
QACC_H[ 95: 64] = {24{qu[87]}, qu[ 87: 80]}
...
QACC_H[255:224] = {24{qu[127]}, qu[127:120]}
```

Schedule

Operands	Use	Def
qu	2	-
QACC_H	-	2
QACC_L	-	2

Format



Figure 4.7-225. ESP.MOV.S8.QACC

4.7.2.226 ESP.MOV.U16.QACC

Syntax

ESP.MOV.U16.QACC qu

Description

This instruction zero-extends the 8 segments of 16-bit data in the register `qs` to 64 bits and writes the results to the special registers `QACC_H` and `QACC_L`.

Operation

```
QACC_L[ 63:  0] = {48{0}, qu[ 15:  0]}
QACC_L[127: 64] = {48{0}, qu[ 31: 16]}
QACC_L[191:128] = {48{0}, qu[ 47: 32]}
QACC_L[255:192] = {48{0}, qu[ 63: 48]}
QACC_H[ 39:  0] = {48{0}, qu[ 79: 64]}
QACC_H[ 79: 40] = {48{0}, qu[ 95: 80]}
QACC_H[119: 80] = {48{0}, qu[111: 96]}
QACC_H[159:120] = {48{0}, qu[127:112]}
```

Schedule

Operands	Use	Def
qu	2	-
QACC_H	-	2
QACC_L	-	2

Format



Figure 4.7-226. ESP.MOV.U16.QACC

4.7.2.227 ESP.MOV.U8.QACC

Syntax

ESP.MOV.U8.QACC qu

Description

This instruction zero-extends the 16 segments of 8-bit data in the register `qs` to 32 bits and writes the result to the special registers `QACC_H` and `QACC_L`.

Operation

```
QACC_L[ 31:  0] = {24{0}, qu[  7:  0]}
QACC_L[ 63: 32] = {24{0}, qu[ 15:  8]}
QACC_L[ 95: 64] = {24{0}, qu[ 23: 16]}
...
QACC_L[255:224] = {24{0}, qu[ 63: 56]}
QACC_H[ 31:  0] = {24{0}, qu[ 71: 64]}
QACC_H[ 63: 32] = {24{0}, qu[ 79: 72]}
QACC_H[ 95: 64] = {24{0}, qu[ 87: 80]}
...
QACC_H[255:224] = {24{0}, qu[127:120]}
```

Schedule

Operands	Use	Def
qu	2	-
QACC_H	-	2
QACC_L	-	2

Format



Figure 4.7-227. ESP.MOV.U8.QACC

4.7.2.228 ESP.MOVI.16.A

Syntax

ESP.MOVI.16.A qy,rd,sel16

Description

This instruction extracts 16 bits from the sel index of qy to the `rd` register.

Operation

```
if sel == 0:    rd[31 :  0] = { '16b0, qy[15 :  0] }
else if sel==1: rd[31 :  0] = { '16b0, qy[31 : 16] }
else if sel==2: rd[31 :  0] = { '16b0, qy[47 : 32] }
else if sel==3: rd[31 :  0] = { '16b0, qy[63 : 48] }
else if sel==4: rd[31 :  0] = { '16b0, qy[79 : 64] }
else if sel==5: rd[31 :  0] = { '16b0, qy[95 : 80] }
else if sel==6: rd[31 :  0] = { '16b0, qy[111: 96] }
else if sel==7: rd[31 :  0] = { '16b0, qy[127:112] }
```

Schedule

Operands	Use	Def
rd	-	2
qy	1	-

Format

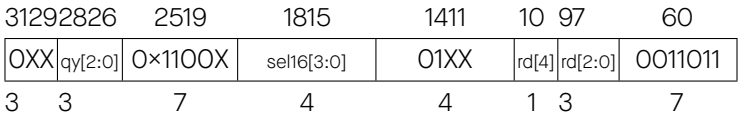


Figure 4.7-228. ESP.MOVI.16.A

4.7.2.229 ESP.MOVI.16.Q

Syntax

ESP.MOVI.16.Q qy,rs1,sel16

Description

This instruction inserts 16 bits into the sel index of `qy` from the `rs1` register.

Operation

```
if sel == 0:        qy[31 :  0] = { qy[31 : 16], rs1[15:0] }
else if sel == 1:   qy[31 :  0] = { rs1[15:0], qy[15 :  0] }
else if sel == 2:   qy[63 : 32] = { qy[63 : 48], rs1[15:0] }
else if sel == 3:   qy[63 : 48] = { rs1[15:0], qy[47 : 32] }
else if sel == 4:   qy[95 : 64] = { qy[95 : 80], rs1[15:0] }
else if sel == 5:   qy[95 : 64] = { rs1[15:0], qy[79 : 64] }
else if sel == 6:   qy[127: 96] = { qy[127:112], rs1[15:0] }
else if sel == 7:   qy[127: 96] = { rs1[15:0], qy[111: 96] }
```

Schedule

Operands	Use	Def
qy	1	1
rs1	1	-

Format

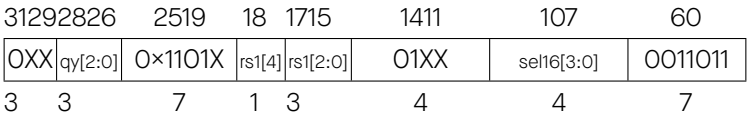


Figure 4.7-229. ESP.MOVI.16.Q

4.7.2.230 ESP.MOVI.32.A

Syntax

ESP.MOVI.32.A qy,rd,sel4

Description

This instruction extracts 32 bits from the sel index of qv to the `rd` register.

Operation

```
    if sel == 0:    rd[31 :  0] = qy[31 :  0]
    else if sel==1: rd[31 :  0] = qy[63 : 32]
    else if sel==2: rd[31 :  0] = qy[95 : 64]
    else if sel==3: rd[31 :  0] = qy[127: 96]
```

Schedule

Operands	Use	Def
rd	-	2
qy	1	-

Format

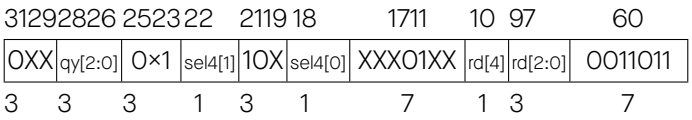


Figure 4.7-230. ESP.MOVI.32.A

4.7.2.231 ESP.MOVI.32.Q

Syntax

ESP.MOVI.32.Q qy,rs1,sel4

Description

This instruction inserts 32 bits into the sel index of `qy` from the `rs1` register.

Operation

```
    if sel == 0:        qy[31 :  0] = rs1[31:0]
    else if sel == 1:   qy[63 : 32] = rs1[31:0]
    else if sel == 2:   qy[95 : 64] = rs1[31:0]
    else if sel == 3:   qy[127: 96] = rs1[31:0]
```

Schedule

Operands	Use	Def
qy	1	1
rs1	1	-

Format

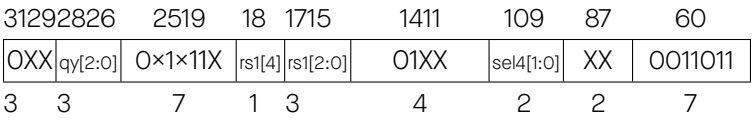


Figure 4.7-231. ESP.MOVI.32.Q

4.7.2.232 ESP.MOVI.8.A

Syntax

ESP.MOVI.8.A qy,rd,sel16

Description

This instruction extracts 8 bits from the sel index of qv to the `rd` register.

Operation

```
    if sel == 0:    rd[31 :  0] = { '24b0, qy[7  :  0] }
    else if sel==1: rd[31 :  0] = { '24b0, qy[15 :  8] }
    else if sel==2: rd[31 :  0] = { '24b0, qy[23 : 16] }
    else if sel==3: rd[31 :  0] = { '24b0, qy[31 : 24] }
    else if sel==4: rd[31 :  0] = { '24b0, qy[39 : 32] }
    else if sel==5: rd[31 :  0] = { '24b0, qy[47 : 40] }
    else if sel==6: rd[31 :  0] = { '24b0, qy[55 : 48] }
    else if sel==7: rd[31 :  0] = { '24b0, qy[63 : 56] }
    else if sel==8: rd[31 :  0] = { '24b0, qy[71 : 64] }
    else if sel==9: rd[31 :  0] = { '24b0, qy[79 : 72] }
    else if sel==10:rd[31 :  0] = { '24b0, qy[87 : 80] }
    else if sel==11:rd[31 :  0] = { '24b0, qy[95 : 88] }
    else if sel==12:rd[31 :  0] = { '24b0, qy[103: 96] }
    else if sel==13:rd[31 :  0] = { '24b0, qy[111:104] }
    else if sel==14:rd[31 :  0] = { '24b0, qy[119:112] }
    else if sel==15:rd[31 :  0] = { '24b0, qy[127:120] }
```

Schedule

Operands	Use	Def
rd	-	2
qy	1	-

Format

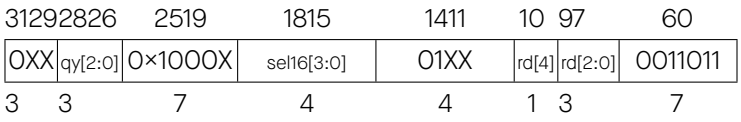


Figure 4.7-232. ESP.MOVI.8.A

4.7.2.233 ESP.MOVI.8.Q

Syntax

ESP.MOVI.8.Q qy,rs1,sel16

Description

This instruction inserts 8 bits into the sel index of `qy` from the `rs1` register.

Operation

```
if sel == 0:        qy[31 :  0] = { qy[31 :  8], rs1[7:0] }
else if sel == 1:   qy[31 :  0] = { qy[31 : 16], rs1[7:0], qy[7  :  0] }
else if sel == 2:   qy[31 :  0] = { qy[31 :  0], rs1[7:0], qy[15 :  0] }
else if sel == 3:   qy[31 : 24] = { rs1[7:0], qy[23 :  0] }
else if sel == 4:   qy[63 : 32] = { qy[63 : 40], rs1[7:0] }
else if sel == 5:   qy[63 : 32] = { qy[63 : 48], rs1[7:0], qy[39 : 32] }
else if sel == 6:   qy[63 : 32] = { qy[63 : 56], rs1[7:0], qy[47 : 32] }
else if sel == 7:   qy[63 : 32] = { rs1[7:0], qy[55 : 32] }
else if sel == 8:   qy[95 : 64] = { qy[95 : 72], rs1[7:0] }
else if sel == 9:   qy[95 : 64] = { qy[95 : 80], rs1[7:0], qy[71 : 64] }
else if sel == 10:  qy[95 : 64] = { qy[95 : 88], rs1[7:0], qy[79 : 64] }
else if sel == 11:  qy[95 : 64] = { rs1[7:0], qy[87 : 64] }
else if sel == 12:  qy[127: 96] = { qy[127:104], rs1[7:0] }
else if sel == 13:  qy[127: 96] = { qy[127:112], rs1[7:0], qy[103: 96] }
else if sel == 14:  qy[127: 96] = { qy[127:120], rs1[7:0], qy[111: 96] }
else if sel == 15:  qy[127: 96] = { rs1[7:0], qy[119: 96] }
```

Schedule

Operands	Use	Def
qy	1	1
rs1	1	-

Format

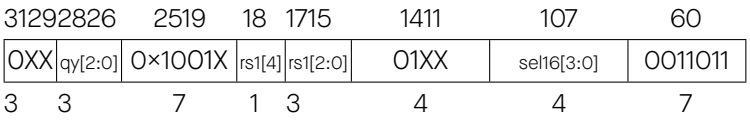


Figure 4.7-233. ESP.MOVI.8.Q

4.7.2.234 ESP.MOVX.R.CFG

Syntax

ESP.MOVX.R.CFG rd

Description

This instruction moves data from the special register `CFG` to the `rd` register.

Operation

```
    rd[31 :  0] = CFG[31:0]
```

Schedule

Operands	Use	Def
rd	-	2
CFG	1	-

Format

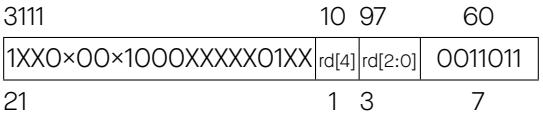


Figure 4.7-234. ESP.MOVX.R.CFG

4.7.2.235 ESP.MOVX.R.FFT.BIT.WIDTH

Syntax

ESP.MOVX.R.FFT.BIT.WIDTH rd

Description

This instruction moves data from the special register `FFT_BIT_WIDTH` to the `rd` register.

Operation

```
    rd[31 :  0] = {28{0}, FFT_BIT_WIDTH[3  :  0]}
```

Schedule

Operands	Use	Def
rd	-	2
FFT_BIT_WIDTH	1	-

Format

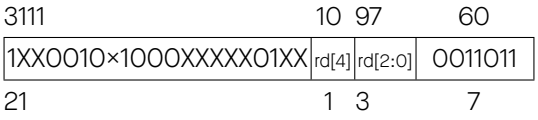


Figure 4.7-235. ESP.MOVX.R.FFT.BIT.WIDTH

4.7.2.236 ESP.MOVX.R.PERF

Syntax

ESP.MOVX.R.PERF rd,rs1

Description

This instruction moves data from the special register `PERF` to the `rd` register.

Operation

```
rd[31 :  0] = PERF[31 :  0]
```

Schedule

Operands	Use	Def
rd	-	2
PERF	1	-

Format

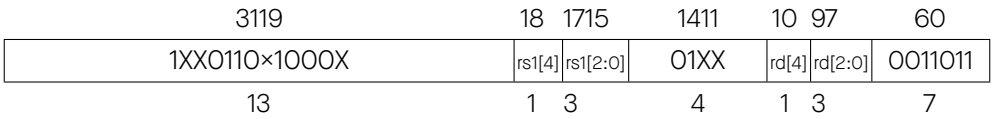


Figure 4.7-236. ESP.MOVX.R.PERF

4.7.2.237 ESP.MOVX.R.SAR

Syntax

ESP.MOVX.R.SAR rd

Description

This instruction moves data from the special register SAR to the `rd` register.

Operation

```
rd[31 :  0] = '{26b0, SAR[5:0]}
```

Schedule

Operands	Use	Def
rd	-	2
SAR	1	-

Format

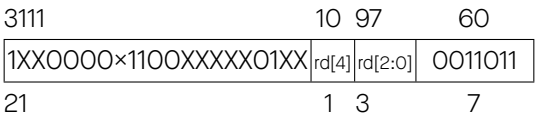


Figure 4.7-237. ESP.MOVX.R.SAR

4.7.2.238 ESP.MOVX.R.SAR.BYTES

Syntax

ESP.MOVX.R.SAR.BYTES rd

Description

This instruction moves data from the special register `SAR_BYTES` to the `rd` register.

Operation

```
rd[31 :  0] = '{28b0, SAR_BYTES[3:0]}
```

Schedule

Operands	Use	Def
rd	-	2
SAR_BYTES	1	-

Format

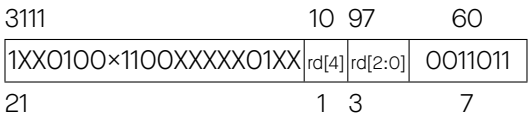


Figure 4.7-238. ESP.MOVX.R.SAR.BYTES

4.7.2.239 ESP.MOVX.R.XACC.H

Syntax

ESP.MOVX.R.XACC.H rd

Description

This instruction moves 8-bit data from the special register `XACC` to the `rd` register.

Operation

```
rd[31 :  0] = '{24b0, XACC[39 : 32]}
```

Schedule

Operands	Use	Def
rd	-	2
XACC	2	-

Format

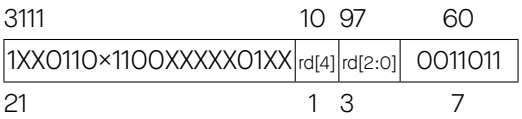


Figure 4.7-239. ESP.MOVX.R.XACC.H

4.7.2.240 ESP.MOVX.R.XACC.L

Syntax

ESP.MOVX.R.XACC.L rd

Description

This instruction moves 32 bits of the special register `XACC` to the `rd` register.

Operation

```
rd[31 :  0] = XACC[31 :  0]
```

Schedule

Operands	Use	Def
rd	-	2
XACC	2	-

Format

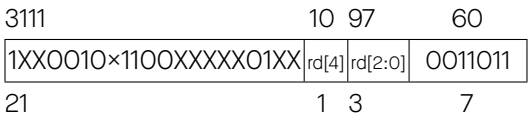


Figure 4.7-240. ESP.MOVX.R.XACC.L

4.7.2.241 ESP.MOVX.W.CFG

Syntax

ESP.MOVX.W.CFG rs1

Description

This instruction writes to the special register `CFG` with the value of the register `rs1`.

Operation

```
CFG[31:0] = rs1[31:0]
```

Schedule

Operands	Use	Def
rs1	1	-
CFG	-	1

Format

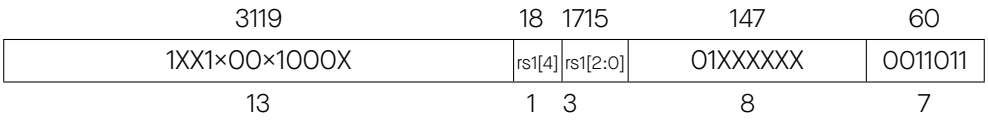


Figure 4.7-241. ESP.MOVX.W.CFG

4.7.2.242 ESP.MOVX.W.FFT.BIT.WIDTH

Syntax

ESP.MOVX.W.FFT.BIT.WIDTH rs1

Description

This instruction writes the `rs1` to `FFT_BIT_WIDTH`.

Operation

```
FFT_BIT_WIDTH[3 : 0] = rs1[3:0]
```

Schedule

Operands	Use	Def
rs1	1	-
FFT_BIT_WIDTH	-	1

Format

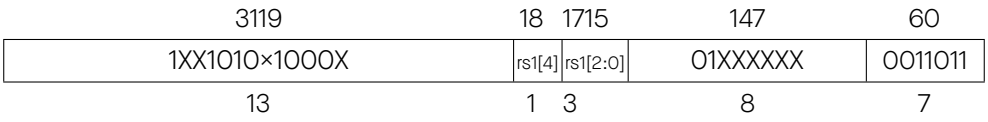


Figure 4.7-242. ESP.MOVX.W.FFT.BIT.WIDTH

4.7.2.243 ESP.MOVX.W.PERF

Syntax

ESP.MOVX.W.PERF rs1

Description

This instruction writes to the special register `PERF` with the value of the register `rs1`.

Operation

```
PERF[31:0] = rs1[31:0]
```

Schedule

Operands	Use	Def
rs1	1	-
PERF	-	1

Format

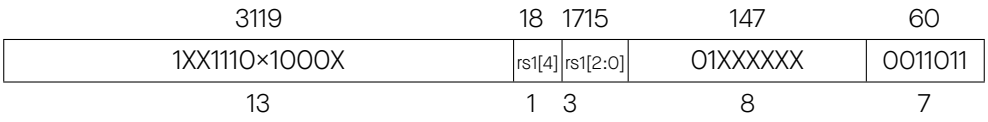


Figure 4.7-243. ESP.MOVX.W.PERF

4.7.2.244 ESP.MOVX.W.SAR

Syntax

ESP.MOVX.W.SAR rs1

Description

This instruction moves data from `rs1` to the special register `SAR`.

Operation

```
SAR[5 : 0] = rs1[5:0]
```

Schedule

Operands	Use	Def
rs1	1	-
SAR	-	1

Format

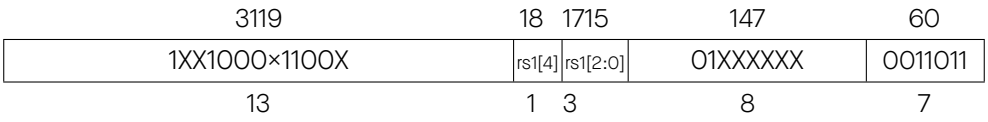


Figure 4.7-244. ESP.MOVX.W.SAR

4.7.2.245 ESP.MOVX.W.SAR.BYTES

Syntax

ESP.MOVX.W.SAR.BYTES rs1

Description

This instruction writes the `rs1` to `SAR_BYTES`.

Operation

```
SAR_BYTES[3 : 0] = rs1[3:0]
```

Schedule

Operands	Use	Def
rs1	1	-
SAR_BYTES	-	1

Format

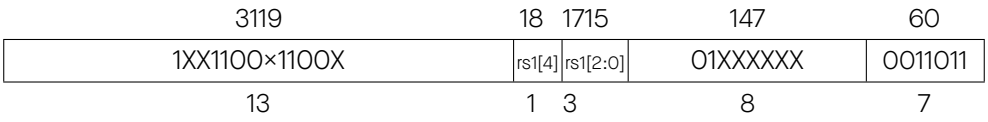


Figure 4.7-245. ESP.MOVX.W.SAR.BYTES

4.7.2.246 ESP.MOVX.W.XACC.H

Syntax

ESP.MOVX.W.XACC.H rs1

Description

This instruction writes to the special register `XACC[39:32]` with the value of `rs1`.

Operation

```
XACC[39:32] = rs1[7:0]
```

Schedule

Operands	Use	Def
rs1	1	-
XACC	-	2

Format

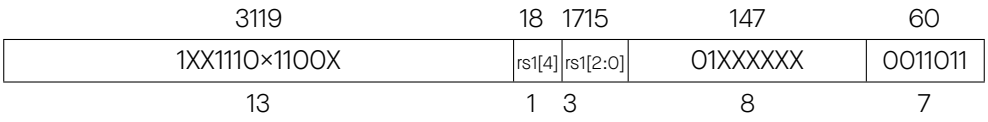


Figure 4.7-246. ESP.MOVX.W.XACC.H

4.7.2.247 ESP.MOVX.W.XACC.L

Syntax

ESP.MOVX.W.XACC.L rs1

Description

This instruction writes to the special register `XACC[31:0]` with the value of `rs1`.

Operation

```
XACC[31:0] = rs1[31:0]
```

Schedule

Operands	Use	Def
rs1	1	-
XACC	-	2

Format

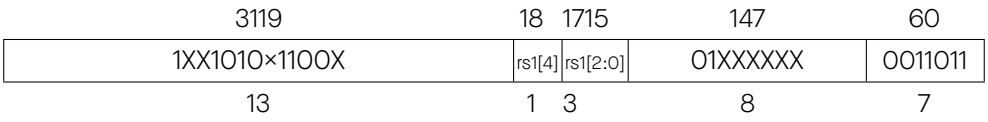


Figure 4.7-247. ESP.MOVX.W.XACC.L

4.7.2.248 ESP.VEXT.S16

Syntax

ESP.VEXT.S16 qz,qv,qw

Description

This instruction signs-extends the load data from a 16-bit segment to a 32-bit segment.

Operation

```
qz[31 :  0] = { {16{qw[ 15]}}, qw[15 :  0] }
qz[63 : 32] = { {16{qw[ 31]}}, qw[31 : 16] }
qz[95 : 64] = { {16{qw[ 47]}}, qw[47 : 32] }
qz[127: 96] = { {16{qw[ 63]}}, qw[63 : 48] }
qv[31 :  0] = { {16{qw[ 79]}}, qw[79 : 64] }
qv[63 : 32] = { {16{qw[ 95]}}, qw[95 : 80] }
qv[95 : 64] = { {16{qw[111]}}, qw[111: 96] }
qv[127: 96] = { {16{qw[127]}}, qw[127:112] }
```

Schedule

Operands	Use	Def
qz	-	1
qv	-	1
qw	1	-

Format

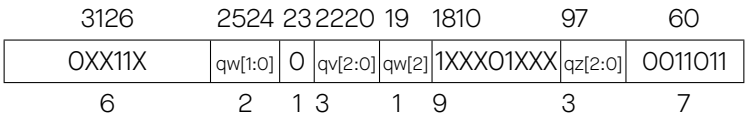


Figure 4.7-248. ESP.VEXT.S16

4.7.2.249 ESP.VEXT.S8

Syntax

ESP.VEXT.S8 qz,qv,qw

Description

This instruction signs-extends the load data from an 8-bit segment to a 16-bit segment.

Operation

```
qz[15 :  0] = { {8{qw[  7]}}, qw[7  :  0] }
qz[31 : 16] = { {8{qw[ 15]}}, qw[15 :  8] }
qz[47 : 32] = { {8{qw[ 23]}}, qw[23 : 16] }
qz[63 : 48] = { {8{qw[ 31]}}, qw[31 : 24] }
qz[79 : 64] = { {8{qw[ 39]}}, qw[39 : 32] }
qz[95 : 80] = { {8{qw[ 47]}}, qw[47 : 40] }
qz[111: 96] = { {8{qw[ 55]}}, qw[55 : 48] }
qz[127:112] = { {8{qw[ 63]}}, qw[63 : 56] }
qv[15 :  0] = { {8{qw[ 71]}}, qw[71 : 64] }
qv[31 : 16] = { {8{qw[ 79]}}, qw[79 : 72] }
qv[47 : 32] = { {8{qw[ 87]}}, qw[87 : 80] }
qv[63 : 48] = { {8{qw[ 95]}}, qw[95 : 88] }
qv[79 : 64] = { {8{qw[103]}}, qw[103: 96] }
qv[95 : 80] = { {8{qw[111]}}, qw[111:104] }
qv[111: 96] = { {8{qw[119]}}, qw[119:112] }
qv[127:112] = { {8{qw[127]}}, qw[127:120] }
```

Schedule

Operands	Use	Def
qz	-	1
qv	-	1
qw	1	-

Format

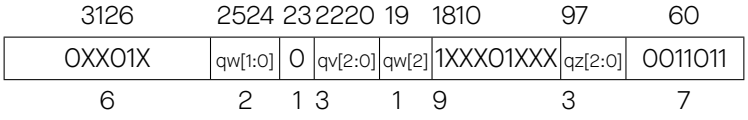


Figure 4.7-249. ESP.VEXT.S8

4.7.2.250 ESP.VEXT.U16

Syntax

ESP.VEXT.U16 qz,qv,qw

Description

This instruction zero-extends the load data from a 16-bit segment to a 32-bit segment.

Operation

```
qz[31 :  0] = { {16{0}}, qw[15 :  0] }
qz[63 : 32] = { {16{0}}, qw[31 : 16] }
qz[95 : 64] = { {16{0}}, qw[47 : 32] }
qz[127: 96] = { {16{0}}, qw[63 : 48] }
qv[31 :  0] = { {16{0}}, qw[79 : 64] }
qv[63 : 32] = { {16{0}}, qw[95 : 80] }
qv[95 : 64] = { {16{0}}, qw[111: 96] }
qv[127: 96] = { {16{0}}, qw[127:112] }
```

Schedule

Operands	Use	Def
qz	-	1
qv	-	1
qw	1	-

Format

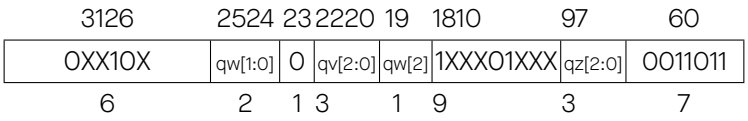


Figure 4.7-250. ESP.VEXT.U16

4.7.2.251 ESP.VEXT.U8

Syntax

ESP.VEXT.U8 qz,qv,qw

Description

This instruction zero-extends the load data from an 8-bit segment to a 16-bit segment.

Operation

```
qz[15 :  0] = { {8{0}}, qw[7  :  0] }
qz[31 : 16] = { {8{0}}, qw[15 :  8] }
qz[47 : 32] = { {8{0}}, qw[23 : 16] }
qz[63 : 48] = { {8{0}}, qw[31 : 24] }
qz[79 : 64] = { {8{0}}, qw[39 : 32] }
qz[95 : 80] = { {8{0}}, qw[47 : 40] }
qz[111: 96] = { {8{0}}, qw[55 : 48] }
qz[127:112] = { {8{0}}, qw[63 : 56] }
qv[15 :  0] = { {8{0}}, qw[71 : 64] }
qv[31 : 16] = { {8{0}}, qw[79 : 72] }
qv[47 : 32] = { {8{0}}, qw[87 : 80] }
qv[63 : 48] = { {8{0}}, qw[95 : 88] }
qv[79 : 64] = { {8{0}}, qw[103: 96] }
qv[95 : 80] = { {8{0}}, qw[111:104] }
qv[111: 96] = { {8{0}}, qw[119:112] }
qv[127:112] = { {8{0}}, qw[127:120] }
```

Schedule

Operands	Use	Def
qz	-	1
qv	-	1
qw	1	-

Format

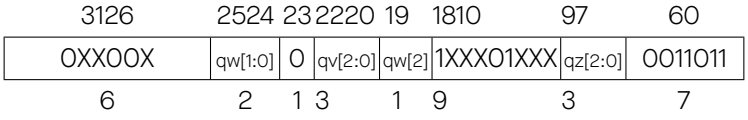


Figure 4.7-251. ESP.VEXT.U8

4.7.2.252 ESP.VUNZIP.16

Syntax

ESP.VUNZIP.16 qx,qy

Description

This instruction implements the unzip algorithm on 16-bit vector data.

Operation

```
qx[ 15:  0] = qx[ 15:  0]
qx[ 31: 16] = qx[ 47: 32]
qx[ 47: 32] = qx[ 79: 64]
qx[ 63: 48] = qx[111: 96]
qx[ 79: 64] = qy[ 15:  0]
qx[ 95: 80] = qy[ 47: 32]
qx[111: 96] = qy[ 79: 64]
qx[127:112] = qy[111: 96]
qy[ 15:  0] = qx[ 31: 16]
qy[ 31: 16] = qx[ 63: 48]
qy[ 47: 32] = qx[ 95: 80]
qy[ 63: 48] = qx[127:112]
qy[ 79: 64] = qy[ 31: 16]
qy[ 95: 80] = qy[ 63: 48]
qy[111: 96] = qy[ 95: 80]
qy[127:112] = qy[127:112]
```

Schedule

Operands	Use	Def
qx	1	1
qy	1	1

Format

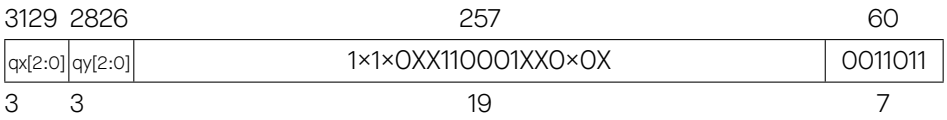


Figure 4.7-252. ESP.VUNZIP.16

4.7.2.253 ESP.VUNZIP.32

Syntax

ESP.VUNZIP.32 qx,qy

Description

This instruction implements the unzip algorithm on 32-bit vector data.

Operation

```
qx[ 31:  0] = qx[ 31:  0]
qx[ 63: 32] = qx[ 95: 64]
qx[ 95: 64] = qy[ 31:  0]
qx[127: 96] = qy[ 95: 64]
qy[ 31:  0] = qx[ 63: 32]
qy[ 63: 32] = qx[127: 96]
qy[ 95: 64] = qy[ 63: 32]
qy[127: 96] = qy[127: 96]
```

Schedule

Operands	Use	Def
qx	1	1
qy	1	1

Format

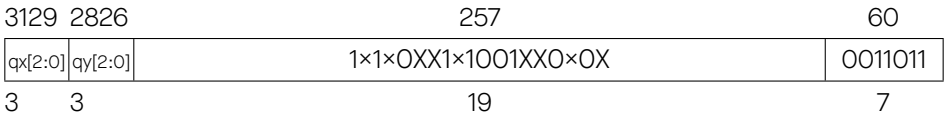


Figure 4.7-253. ESP.VUNZIP.32

4.7.2.254 ESP.VUNZIP.8

Syntax

ESP.VUNZIP.8 qx,qy

Description

This instruction implements the unzip algorithm on 8-bit vector data.

Operation

```
qx[  7:  0] = qx[  7:  0]
qx[ 15:  8] = qx[ 23: 16]
qx[ 23: 16] = qx[ 39: 32]
qx[ 31: 24] = qx[ 55: 48]
qx[ 39: 32] = qx[ 71: 64]
qx[ 47: 40] = qx[ 87: 80]
qx[ 55: 48] = qx[103: 96]
qx[ 63: 56] = qx[119:112]
qx[ 71: 64] = qy[  7:  0]
qx[ 79: 72] = qy[ 23: 16]
qx[ 87: 80] = qy[ 39: 32]
qx[ 95: 88] = qy[ 55: 48]
qx[103: 96] = qy[ 71: 64]
qx[111:104] = qy[ 87: 80]
qx[119:112] = qy[103: 96]
qx[127:120] = qy[119:112]
qy[  7:  0] = qx[ 15:  8]
qy[ 15:  8] = qx[ 31: 24]
qy[ 23: 16] = qx[ 47: 40]
qy[ 31: 24] = qx[ 63: 56]
qy[ 39: 32] = qx[ 79: 72]
qy[ 47: 40] = qx[ 95: 88]
qy[ 55: 48] = qx[111:104]
qy[ 63: 56] = qx[127:120]
qy[ 71: 64] = qy[ 15:  8]
qy[ 79: 72] = qy[ 31: 24]
qy[ 87: 80] = qy[ 47: 40]
qy[ 95: 88] = qy[ 63: 56]
qy[103: 96] = qy[ 79: 72]
qy[111:104] = qy[ 95: 88]
qy[119:112] = qy[111:104]
qy[127:120] = qy[127:120]
```

Schedule

Operands	Use	Def
qx	1	1
qy	1	1

Format

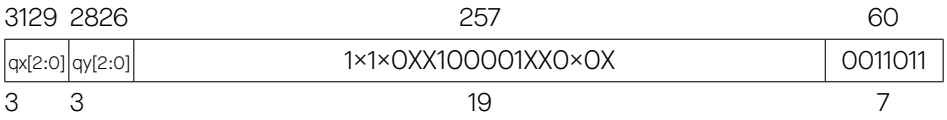


Figure 4.7-254. ESP.VUNZIP.8

4.7.2.255 ESP.VUNZIPT.16

Syntax

ESP.VUNZIPT.16 qx,qy,qw

Description

This instruction exchanges the values of `qx`, `qy`, and `qw`; it can be used to convert RGB to YUV.

Operation

```
temp0[ 15:  0] = qx[ 15:  0]
temp1[ 15:  0] = qx[ 31: 16]
temp2[ 15:  0] = qx[ 47: 32]
temp0[ 31: 16] = qx[ 63: 48]
temp1[ 31: 16] = qx[ 79: 64]
temp2[ 31: 16] = qx[ 95: 80]
temp0[ 47: 32] = qx[111: 96]
temp1[ 47: 32] = qx[127:112]

temp2[ 47: 32] = qy[ 15:  0]
temp0[ 63: 48] = qy[ 31: 16]
temp1[ 63: 48] = qy[ 47: 32]
temp2[ 63: 48] = qy[ 63: 48]
temp0[ 79: 64] = qy[ 79: 64]
temp1[ 79: 64] = qy[ 95: 80]
temp2[ 79: 64] = qy[111: 96]
temp0[ 95: 80] = qy[127:112]

temp1[ 95: 80] = qw[ 15:  0]
temp2[ 95: 80] = qw[ 31: 16]
temp0[111: 96] = qw[ 47: 32]
temp1[111: 96] = qw[ 63: 48]
temp2[111: 96] = qw[ 79: 64]
temp0[127:112] = qw[ 95: 80]
temp1[127:112] = qw[111: 96]
temp2[127:112] = qw[127:112]

qx[127:0] = temp0[127:0]
qy[127:0] = temp1[127:0]
qw[127:0] = temp2[127:0]
```

Schedule

Operands	Use	Def
qx	1	1
qy	1	1

Operands	Use	Def
qw	1	1

Format

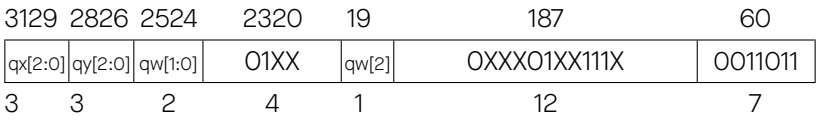


Figure 4.7-255. ESP.VUNZIPT.16

4.7.2.256 ESP.VUNZIPT.8

Syntax

ESP.VUNZIPT.8 qx,qy,qw

Description

This instruction exchanges the values of `qx`, `qy`, and `qw`; it can be used to convert RGB to YUV.

Operation

```
temp0[  7:  0] = qx[  7:  0]
temp1[  7:  0] = qx[ 15:  8]
temp2[  7:  0] = qx[ 23: 16]
temp0[ 15:  8] = qx[ 31: 24]
temp1[ 15:  8] = qx[ 39: 32]
temp2[ 15:  8] = qx[ 47: 40]
temp0[ 23: 16] = qx[ 55: 48]
temp1[ 23: 16] = qx[ 63: 56]
temp2[ 23: 16] = qx[ 71: 64]
temp0[ 31: 24] = qx[ 79: 72]
temp1[ 31: 24] = qx[ 87: 80]
temp2[ 31: 24] = qx[ 95: 88]
temp0[ 39: 32] = qx[103: 96]
temp1[ 39: 32] = qx[111:104]
temp2[ 39: 32] = qx[119:112]
temp0[ 47: 40] = qx[127:120]

temp1[ 47: 40] = qy[  7:  0]
temp2[ 47: 40] = qy[ 15:  8]
temp0[ 55: 48] = qy[ 23: 16]
temp1[ 55: 48] = qy[ 31: 24]
temp2[ 55: 48] = qy[ 39: 32]
temp0[ 63: 56] = qy[ 47: 40]
temp1[ 63: 56] = qy[ 55: 48]
temp2[ 63: 56] = qy[ 63: 56]
temp0[ 71: 64] = qy[ 71: 64]
temp1[ 71: 64] = qy[ 79: 72]
temp2[ 71: 64] = qy[ 87: 80]
temp0[ 79: 72] = qy[ 95: 88]
temp1[ 79: 72] = qy[103: 96]
temp2[ 79: 72] = qy[111:104]
temp0[ 87: 80] = qy[119:112]
temp1[ 87: 80] = qy[127:120]

temp2[ 87: 80] = qw[  7:  0]
temp0[ 95: 88] = qw[ 15:  8]
temp1[ 95: 88] = qw[ 23: 16]
temp2[ 95: 88] = qw[ 31: 24]
temp0[103: 96] = qw[ 39: 32]
temp1[103: 96] = qw[ 47: 40]
temp2[103: 96] = qw[ 55: 48]
temp0[111:104] = qw[ 63: 56]
```

```
temp1[111:104] = qw[ 71: 64]
temp2[111:104] = qw[ 79: 72]
temp0[119:112] = qw[ 87: 80]
temp1[119:112] = qw[ 95: 88]
temp2[119:112] = qw[103: 96]
temp0[127:120] = qw[111:104]
temp1[127:120] = qw[119:112]
temp2[127:120] = qw[127:120]

qx[127:0] = temp0[127:0]
qy[127:0] = temp1[127:0]
qw[127:0] = temp2[127:0]
```

Schedule

Operands	Use	Def
qx	1	1
qy	1	1
qw	1	1

Format

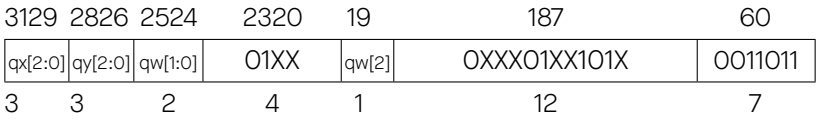


Figure 4.7-256. ESP.VUNZIPT.8

4.7.2.257 ESP.VZIP.16

Syntax

ESP.VZIP.16 qx,qy

Description

This instruction interleaves the values of qx and qy.

Operation

```
qx[ 15:  0] = qx[ 15:  0]
qx[ 31: 16] = qy[ 15:  0]
qx[ 47: 32] = qx[ 31: 16]
qx[ 63: 48] = qy[ 31: 16]
qx[ 79: 64] = qx[ 47: 32]
qx[ 95: 80] = qy[ 47: 32]
qx[111: 96] = qx[ 63: 48]
qx[127:112] = qy[ 63: 48]
qy[ 15:  0] = qx[ 79: 64]
qy[ 31: 16] = qy[ 79: 64]
qy[ 47: 32] = qx[ 95: 80]
qy[ 63: 48] = qy[ 95: 80]
qy[ 79: 64] = qx[111: 96]
qy[ 95: 80] = qy[111: 96]
qy[111: 96] = qx[127:112]
qy[127:112] = qy[127:112]
```

Schedule

Operands	Use	Def
qx	1	1
qy	1	1

Format

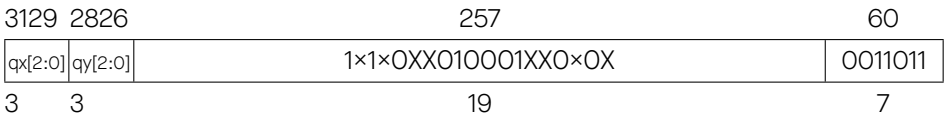


Figure 4.7-257. ESP.VZIP.16

4.7.2.258 ESP.VZIP.32

Syntax

ESP.VZIP.32 qx,qy

Description

This instruction interleaves the values of `qx` and `qy`.

Operation

```
qx[ 31:  0] = qx[ 31:  0]
qx[ 63: 32] = qy[ 31:  0]
qx[ 95: 64] = qx[ 63: 32]
qx[127: 96] = qy[ 63: 32]
qy[ 31:  0] = qx[ 95: 64]
qy[ 63: 32] = qy[ 95: 64]
qy[ 95: 64] = qx[127: 96]
qy[127: 96] = qy[127: 96]
```

Schedule

Operands	Use	Def
qx	1	1
qy	1	1

Format

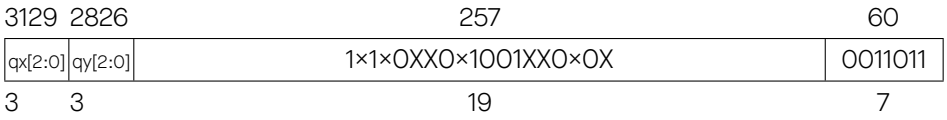


Figure 4.7-258. ESP.VZIP.32

4.7.2.259 ESP.VZIP.8

Syntax

ESP.VZIP.8 qx,qy

Description

This instruction interleaves the values of `qx` and `qy`.

Operation

```
qx[  7:  0] = qx[  7:  0]
qx[ 15:  8] = qy[  7:  0]
qx[ 23: 16] = qx[ 15:  8]
qx[ 31: 24] = qy[ 15:  8]
qx[ 39: 32] = qx[ 23: 16]
qx[ 47: 40] = qy[ 23: 16]
qx[ 55: 48] = qx[ 31: 24]
qx[ 63: 56] = qy[ 31: 24]
qx[ 71: 64] = qx[ 39: 32]
qx[ 79: 72] = qy[ 39: 32]
qx[ 87: 80] = qx[ 47: 40]
qx[ 95: 88] = qy[ 47: 40]
qx[103: 96] = qx[ 55: 48]
qx[111:104] = qy[ 55: 48]
qx[119:112] = qx[ 63: 56]
qx[127:120] = qy[ 63: 56]
qy[  7:  0] = qx[ 71: 64]
qy[ 15:  8] = qy[ 71: 64]
qy[ 23: 16] = qx[ 79: 72]
qy[ 31: 24] = qy[ 79: 72]
qy[ 39: 32] = qx[ 87: 80]
qy[ 47: 40] = qy[ 87: 80]
qy[ 55: 48] = qx[ 95: 88]
qy[ 63: 56] = qy[ 95: 88]
qy[ 71: 64] = qx[103: 96]
qy[ 79: 72] = qy[103: 96]
qy[ 87: 80] = qx[111:104]
qy[ 95: 88] = qy[111:104]
qy[103: 96] = qx[119:112]
qy[111:104] = qy[119:112]
qy[119:112] = qx[127:120]
qy[127:120] = qy[127:120]
```

Schedule

Operands	Use	Def
qx	1	1
qy	1	1

Format

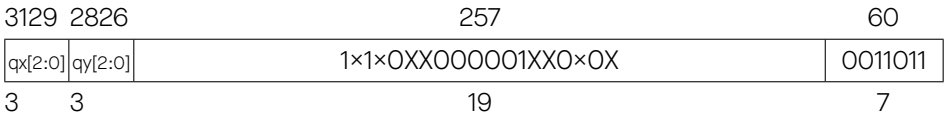


Figure 4.7-259. ESP.VZIP.8

4.7.2.260 ESP.VZIPT.16

Syntax

ESP.VZIPT.16 qx,qy,qw

Description

This instruction interleaves the values of `qx`, `qy`, and `qw`. It can be used for image color transmission, for example, converting YUV to RGB.

Operation

```
temp0[ 15:  0] = qx[ 15:  0]
temp0[ 31: 16] = qy[ 15:  0]
temp0[ 47: 32] = qw[ 15:  0]
temp0[ 63: 48] = qx[ 31: 16]
temp0[ 79: 64] = qy[ 31: 16]
temp0[ 95: 80] = qw[ 31: 16]
temp0[111: 96] = qx[ 47: 32]
temp0[127:112] = qy[ 47: 32]

temp1[ 15:  0] = qw[ 47: 32]
temp1[ 31: 16] = qx[ 63: 48]
temp1[ 47: 32] = qy[ 63: 48]
temp1[ 63: 48] = qw[ 63: 48]
temp1[ 79: 64] = qx[ 79: 64]
temp1[ 95: 80] = qy[ 79: 64]
temp1[111: 96] = qw[ 79: 64]
temp1[127:112] = qx[ 95: 80]

temp2[ 15:  0] = qy[ 95: 80]
temp2[ 31: 16] = qw[ 95: 80]
temp2[ 47: 32] = qx[111: 96]
temp2[ 63: 48] = qy[111: 96]
temp2[ 79: 64] = qw[111: 96]
temp2[ 95: 80] = qx[127:112]
temp2[111: 96] = qy[127:112]
temp2[127:112] = qw[127:112]

qx[127:0] = temp0[127:0]
qy[127:0] = temp1[127:0]
qw[127:0] = temp2[127:0]
```

Schedule

Operands	Use	Def
qx	1	1

Operands	Use	Def
qy	1	1
qw	1	1

Format

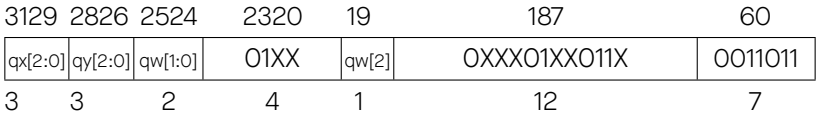


Figure 4.7-260. ESP.VZIPT.16

4.7.2.261 ESP.VZIPT.8

Syntax

ESP.VZIPT.8 qx,qy,qw

Description

This instruction interleaves the values of `qx`, `qy`, and `qw`. It can be used for image color transmission, for example, converting YUV to RGB.

Operation

```
temp0[  7:  0] = qx[  7:  0]
temp0[ 15:  8] = qy[  7:  0]
temp0[ 23: 16] = qw[  7:  0]
temp0[ 31: 24] = qx[ 15:  8]
temp0[ 39: 32] = qy[ 15:  8]
temp0[ 47: 40] = qw[ 15:  8]
temp0[ 55: 48] = qx[ 23: 16]
temp0[ 63: 56] = qy[ 23: 16]
temp0[ 71: 64] = qw[ 23: 16]
temp0[ 79: 72] = qx[ 31: 24]
temp0[ 87: 80] = qy[ 31: 24]
temp0[ 95: 88] = qw[ 31: 24]
temp0[103: 96] = qx[ 39: 32]
temp0[111:104] = qy[ 39: 32]
temp0[119:112] = qw[ 39: 32]
temp0[127:120] = qx[ 47: 40]

temp1[ 15:  0] = qy[ 47: 40]
temp1[ 31: 16] = qw[ 47: 40]
temp1[ 47: 32] = qx[ 55: 48]
temp1[ 63: 48] = qy[ 55: 48]
temp1[ 79: 64] = qw[ 55: 48]
temp1[ 95: 80] = qx[ 63: 56]
temp1[111: 96] = qy[ 63: 56]
temp1[127:112] = qw[ 63: 56]
temp1[ 15:  0] = qx[ 71: 64]
temp1[ 31: 16] = qy[ 71: 64]
temp1[ 47: 32] = qw[ 71: 64]
temp1[ 63: 48] = qx[ 79: 72]
temp1[ 79: 64] = qy[ 79: 72]
temp1[ 95: 80] = qw[ 79: 72]
temp1[111: 96] = qx[ 87: 80]
temp1[127:112] = qy[ 87: 80]

temp2[ 15:  0] = qw[ 87: 80]
temp2[ 31: 16] = qx[ 95: 88]
temp2[ 47: 32] = qy[ 95: 88]
temp2[ 63: 48] = qw[ 95: 88]
temp2[ 79: 64] = qx[103: 96]
temp2[ 95: 80] = qy[103: 96]
temp2[111: 96] = qw[103: 96]
```

```
temp2[127:112] = qx[111:104]
temp2[ 15:  0] = qy[111:104]
temp2[ 31: 16] = qw[111:104]
temp2[ 47: 32] = qx[119:112]
temp2[ 63: 48] = qy[119:112]
temp2[ 79: 64] = qw[119:112]
temp2[ 95: 80] = qx[127:120]
temp2[111: 96] = qy[127:120]
temp2[127:112] = qw[127:120]

qx[127:0] = temp0[127:0]
qy[127:0] = temp1[127:0]
qw[127:0] = temp2[127:0]
```

Schedule

Operands	Use	Def
qx	1	1
qy	1	1
qw	1	1

Format

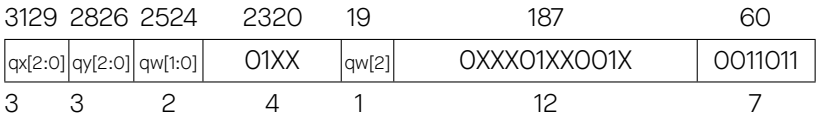


Figure 4.7-261. ESP.VZIPT.8

4.7.2.262 ESP.ZERO.Q

Syntax

ESP.ZERO.Q qz

Description

This instruction sets the register qz to zero.

Operation

```
qz[127:0] = 0
```

Schedule

Operands	Use	Def
qz	-	1

Format



Figure 4.7-262. ESP.ZERO.Q

4.7.2.263 ESP.ZERO.QACC

Syntax

ESP.ZERO.QACC

Description

This instruction sets `QACC_L` and `QACC_H` to zero.

Operation

```
QACC_L = 0
QACC_H = 0
```

Schedule

Operands	Use	Def
QACC_H	-	2
QACC_L	-	2

Format

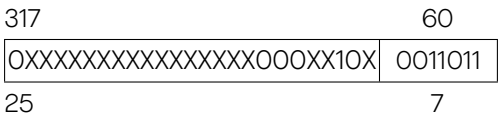


Figure 4.7-263. ESP.ZERO.QACC

4.7.2.264 ESP.ZERO.XACC

Syntax

ESP.ZERO.XACC

Description

This instruction sets XACC to zero.

Operation

```
XACC = 0
```

Schedule

Operands	Use	Def
XACC	-	2

Format

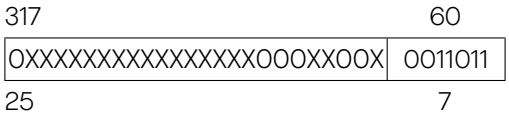


Figure 4.7-264. ESP.ZERO.XACC

4.7.2.265 ESP.FFT.AMS.S16.LD.INCP

Syntax

ESP.FFT.AMS.S16.LD.INCP qu,rs1,qz,qv,qx,qw,qy,sel2,sat

Description

It is a dedicated FFT instruction to perform addition, subtraction, multiplication, addition and subtraction, and shift operations on 16-bit data segments.

During the operation, the 16-byte data is loaded from the memory to register `qu`. After the access, the value in register `rs1` is incremented by 16.

Operation

```
  temp0[16:0] = qx[ 47: 32] + qw[ 47: 32]
  temp1[16:0] = qx[ 63: 48] - qw[ 63: 48]

if sel2==0:
  temp2[33:0] = ((qx[ 47: 32] - qw[ 47: 32]) * qy[ 47: 32] - (qx[ 63: 48] + qw[ 63: 48]) * qy[ 63: 48]) >> SAR
  temp3[33:0] = ((qx[ 47: 32] - qw[ 47: 32]) * qy[ 63: 48] + (qx[ 63: 48] + qw[ 63: 48]) * qy[ 47: 32]) >> SAR
if sel2==1:
  temp2[33:0] = ((qx[ 63: 48] + qw[ 63: 48]) * qy[ 63: 48] + (qx[ 47: 32] - qw[ 47: 32]) * qy[ 47: 32]) >> SAR
  temp3[33:0] = ((qx[ 63: 48] + qw[ 63: 48]) * qy[ 47: 32] - (qx[ 47: 32] - qw[ 47: 32]) * qy[ 63: 48]) >> SAR

  add_res0[34: 0] = temp0[16:0] + temp2[33:0]
  add_res1[34: 0] = temp1[16:0] + temp3[33:0]
  sub_res0[34: 0] = temp0[16:0] - temp2[33:0]
  sub_res1[34: 0] = temp3[33:0] - temp1[16:0]

  qz[15:0] = sat_s16(add_res0[34:0])
  qz[31:16] = sat_s16(add_res1[34:0])
  qv[15:0] = sat_s16(sub_res0[34:0])
  qv[31:16] = sat_s16(sub_res1[34:0])

  qu = load128(as[31:0])
  rs1[31:0] = rs1[31:0] + 16
```

Schedule

Operands	Use	Def
qu	-	2
rs1	1	1
qz	-	2
qv	-	2
qx	1	-
qw	1	-
qy	1	-
SAR	1	-

Format

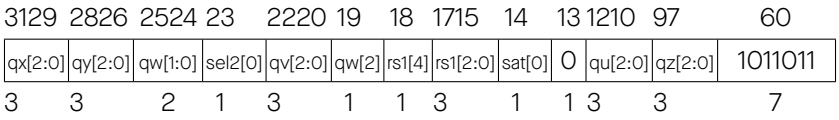


Figure 4.7-265. ESP.FFT.AMS.S16.LD.INCP

4.7.2.266 ESP.FFT.AMS.S16.LD.INCP.UAUP

Syntax

ESP.FFT.AMS.S16.LD.INCP.UAUP qu,rs1,qz,qv,qx,qw,qy,sel2,sat

Description

This is a dedicated FFT instruction designed to perform addition, subtraction, multiplication, addition and subtraction, and shift operations on 16-bit data segments.

During the operation, the 16-byte data are loaded from the memory. The instruction combines the loaded data and the data in the special register `UA_STATE` into 32-byte data, right-shifts the 32-byte data by the result of the `SAR_BYTE` value multiplied by 8, and assigns the lower 128 bits of the shifted result to the register `qu`. Meanwhile, the register `UA_STATE` is updated with the loaded 16-byte data. After the access, the value in the register `rs1` is incremented by 16.

Operation

```
  temp0[16:0] = qx[ 15:  0] + qw[ 15:  0]
  temp1[16:0] = qx[ 31: 16] - qw[ 31: 16]

if sel2==0:
  temp2[33:0] = ((qx[ 15:  0] - qw[ 15:  0]) * qy[ 15:  0] - (qx[ 31: 16] + qw[ 31: 16]) * qy[ 31: 16]) >> SAR
  temp3[33:0] = ((qx[ 15:  0] - qw[ 15:  0]) * qy[ 31: 16] + (qx[ 31: 16] + qw[ 31: 16]) * qy[ 15:  0]) >> SAR
if sel2==1:
  temp2[33:0] = ((qx[ 31: 16] + qw[ 31: 16]) * qy[ 31: 16] + (qx[ 15:  0] - qw[ 15:  0]) * qy[ 15:  0]) >> SAR
  temp3[33:0] = ((qx[ 31: 16] + qw[ 31: 16]) * qy[ 15:  0] - (qx[ 15:  0] - qw[ 15:  0]) * qy[ 31: 16]) >> SAR

add_res0[34: 0] = temp0[16:0] + temp2[33:0]
add_res1[34: 0] = temp1[16:0] + temp3[33:0]
sub_res0[34: 0] = temp0[16:0] - temp2[33:0]
sub_res1[34: 0] = temp3[33:0] - temp1[16:0]

qz[15:0] = sat_s16(add_res0[34:0])
qz[31:16] = sat_s16(add_res1[34:0])
qv[15:0] = sat_s16(sub_res0[34:0])
qv[31:16] = sat_s16(sub_res1[34:0])

  dataIn[127:0] = load128(as[31:0])

  qu[127: 0] = {dataIn[127:0], UA_STATE[127:0]} >> {SAR_BYTE[3:0] << 3}
  UA_STATE[127:0] = dataIn[127:0]
  rs1[31:0] = rs1[31:0] + 16
```

Schedule

Operands	Use	Def
qu	-	2
rs1	1	1
qz	-	2

Operands	Use	Def
qv	-	2
qx	1	-
qw	1	-
qy	1	-
SAR	1	-
SAR_BYTES	1	-

Format

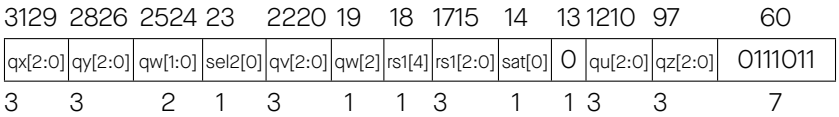


Figure 4.7-266. ESP.FFT.AMS.S16.LD.INCP.UAUP

4.7.2.267 ESP.FFT.AMS.S16.LD.R32.DECP

Syntax

ESP.FFT.AMS.S16.LD.R32.DECP qu,rs1,qz,qv,qx,qw,qy,sel2,sat

Description

It is a dedicated FFT instruction to perform addition, subtraction, multiplication, addition and subtraction, and shift operations on 16-bit data segments.

During the operation, the instruction forces the lower 4 bits of the access address in register `rs1` to 0 and loads the 16-byte data from the memory to register `qu` in the big-endian word order, namely, loads the segment [127: 96] of the data to [31:0] of `qu`. After the access is completed, the value in register `rs1` is decreased by 16.

Operation

```
  temp0[15:0] = qx[ 79: 64] + qy[ 79: 64]
  temp1[15:0] = qx[ 95: 80] - qy[ 95: 80]

if sel2==0:
  temp2[15:0] = ((qx[ 79: 64] - qw[ 79: 64]) * qy[ 79: 64] - (qx[ 95: 80] + qw[ 95:
      80]) * qy[ 95: 80]) >> SAR
  temp3[15:0] = ((qx[ 79: 64] - qw[ 79: 64]) * qy[ 95: 80] + (qx[ 95: 80] + qw[ 95:
      80]) * qy[ 79: 64]) >> SAR
if sel2==1:
  temp2[15:0] = ((qx[ 95: 80] + qw[ 95: 80]) * qy[ 95: 80] + (qx[ 79: 64] - qw[ 79:
      64]) * qy[ 79: 64]) >> SAR
  temp3[15:0] = ((qx[ 95: 80] + qw[ 95: 80]) * qy[ 79: 64] - (qx[ 79: 64] - qw[ 79:
      64]) * qy[ 95: 80]) >> SAR

  qz[79: 64] = temp0[15:0] + temp2[15:0]
  qz[95: 80] = temp1[15:0] + temp3[15:0]
  qv[79:64] = temp0[15:0] - temp2[15:0]
  qv[95:80] = temp3[15:0] - temp1[15:0]
  {qu[31: 0], qu[63: 32], qu[95: 64], qu[127: 96]} = load128(rs1[31:0])
  rs1[31:0] = rs1[31:0] - 16
```

Schedule

Operands	Use	Def
qu	-	2
rs1	1	1
qz	-	2
qv	-	2
qx	1	-
qw	1	-
qy	1	-
SAR	1	-

Format

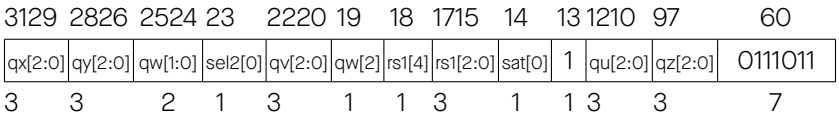


Figure 4.7-267. ESP.FFT.AMS.S16.LD.R32.DECP

4.7.2.268 ESP.FFT.AMS.S16.ST.INCP

Syntax

ESP.FFT.AMS.S16.ST.INCP qu,qz,rs2,rs1,qx,qw,qy,sel2,sat

Description

It is a dedicated FFT instruction to perform addition, subtraction, multiplication, addition and subtraction, and shift operations on 16-bit data segments.

During the operation, splices the values in registers `qu` and `rs2` into 16-byte data, and stores the spliced result to memory. After the access is completed, the value in register `rs1` is incremented by 16. Besides, the operation result is updated to register `rs2`.

Operation

```
  temp0[15:0] = qx[111: 96] + qw[111: 96]
  temp1[15:0] = qx[127:112] - qw[127:112]

if sel2==0:
  temp2[15:0] = ((qx[111: 96] - qw[111: 96]) * qy[111: 96] - (qx[127:112] + qw
      [127:112]) * qy[127:112]) >> SAR[5:0]
  temp3[15:0] = ((qx[111: 96] - qw[111: 96]) * qy[127:112] + (qx[127:112] + qw
      [127:112]) * qy[111: 96]) >> SAR[5:0]
  {qv[ 95: 80] >> 1, qv[ 79: 64] >> 1, qv[ 63: 48] >> 1, qv[ 47: 32] >> 1, qv[ 31:
      16] >> 1, qv[ 15:  0] >> 1, rs2[31: 16] >> 1, rs2[15:0] >> 1} => store128(rs
      2[31:0])
if sel2==1:
  temp2[15:0] = ((qx[127:112] + qw[127:112]) * qy[127:112] + (qx[111: 96] - qw[111:
      96]) * qy[111: 96]) >> SAR[5:0]
  temp3[15:0] = ((qx[127:112] + qw[127:112]) * qy[111: 96] - (qx[111: 96] - qw[111:
      96]) * qy[127:112]) >> SAR[5:0]
  {qu[ 95: 64], qu[ 63: 32], qu[ 31:  0], rs2[31:0]} => store128(rs1[31:0])

  temp4[16:0] = temp1[15:0] + temp3[15:0]
  temp5[16:0] = temp0[15:0] + temp2[15:0]
  qz[111: 96] = temp0[15:0] - temp2[15:0]
  qz[127:112] = temp3[15:0] - temp1[15:0]
  rs2 = {temp4[15:0], temp5[15:0]}
  rs1[31:0] = rs1[31:0] +16
```

Schedule

Operands	Use	Def
qu	2	-
rs1	1	1
qz	-	2
qx	1	-

Operands	Use	Def
qw	1	-
qy	1	-
SAR	1	-

Format

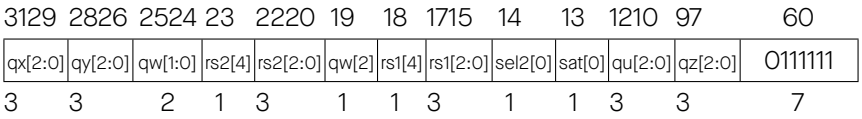


Figure 4.7-268. ESP.FFT.AMS.S16.ST.INCP

4.7.2.269 ESP.FFT.CMUL.S16.LD.XP

Syntax

ESP.FFT.CMUL.S16.LD.XP qu,rs1,rs2,qz,qy,qx,sel8,sat

Description

This instruction performs a 16-bit signed complex multiplication. It is similar to ESP.CMUL.S16, except that the order of registers `qx` and `qy` is reversed.

During the operation, the 16-byte data are loaded from the memory to the register `qu`. After the access is completed, the value in the register `rs1` is incremented by the value in the register `rs2`.

Operation

```
if sel8 == 0:
  qz[ 15:  0] = (qy[ 15:  0] * qx[ 15:  0] + qy[ 31: 16] * qx[ 31: 16]) >> SAR[5:0]
  qz[ 31: 16] = (qy[ 31: 16] * qx[ 15:  0] - qy[ 15:  0] * qx[ 31: 16]) >> SAR[5:0]
if sel8 == 1:
  qz[ 15:  0] = (qy[ 15:  0] * qx[ 15:  0] - qy[ 31: 16] * qx[ 31: 16]) >> SAR[5:0]
  qz[ 31: 16] = (qy[ 31: 16] * qx[ 15:  0] + qy[ 15:  0] * qx[ 31: 16]) >> SAR[5:0]
if sel8 == 2:
  qz[ 47: 32] = (qy[ 47: 32] * qx[ 47: 32] + qy[ 63: 48] * qx[ 63: 48]) >> SAR[5:0]
  qz[ 63: 48] = (qy[ 63: 48] * qx[ 47: 32] - qy[ 47: 32] * qx[ 63: 48]) >> SAR[5:0]
if sel8 == 3:
  qz[ 47: 32] = (qy[ 47: 32] * qx[ 47: 32] - qy[ 63: 48] * qx[ 63: 48]) >> SAR[5:0]
  qz[ 63: 48] = (qy[ 63: 48] * qx[ 47: 32] + qy[ 47: 32] * qx[ 63: 48]) >> SAR[5:0]
if sel8 == 4:
  qz[ 79: 64] = (qy[ 79: 64] * qx[ 79: 64] + qy[ 95: 80] * qx[ 95: 80]) >> SAR[5:0]
  qz[ 95: 80] = (qy[ 95: 80] * qx[ 79: 64] - qy[ 79: 64] * qx[ 95: 80]) >> SAR[5:0]
if sel8 == 5:
  qz[ 79: 64] = (qy[ 79: 64] * qx[ 79: 64] - qy[ 95: 80] * qx[ 95: 80]) >> SAR[5:0]
  qz[ 95: 80] = (qy[ 95: 80] * qx[ 79: 64] + qy[ 79: 64] * qx[ 95: 80]) >> SAR[5:0]

  qu[127:0] = load128(rs1[31:0])
  rs1[31:0] = rs1[31:0] + rs2[31:0]
```

Schedule

Operands	Use	Def
qu	-	2
rs1	1	1
rs2	1	-
qz	-	2
qx	1	-
qy	1	-
SAR	1	-

Format

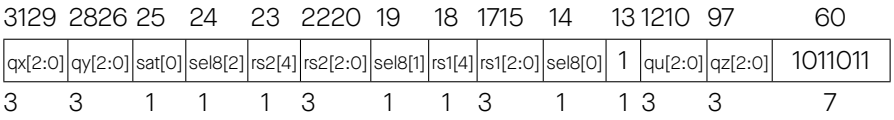


Figure 4.7-269. ESP.FFT.CMUL.S16.LD.XP

4.7.2.270 ESP.FFT.CMUL.S16.ST.XP

Syntax

ESP.FFT.CMUL.S16.ST.XP qy,qx,qu,rs1,rs2,sel8,upd4,sel4,sat

Description

This instruction performs a 16-bit signed complex multiplication. It is similar to ESP.CMUL.S16, except that the order of registers `qx` and `qy` is reversed.

The result of the operation and the data segments in registers `qx` and `qu`, specified by the immediate data `upd4`, are concatenated into 128 bits. These are then written into memory. After the access is completed, the value in register `rs1` is incremented by the value in register `rs2`.

Operation

```
if sel8 == 6:
  temp[15: 0] = (qy[111: 96] * qx[111: 96] + qy[127:112] * qx[127:112]) >> SAR[5:0]
  temp[31:16] = (qy[127:112] * qx[111: 96] - qy[111: 96] * qx[127:112]) >> SAR[5:0]
if sel8 == 7:
  temp[15: 0] = (qy[111: 96] * qx[111: 96] - qy[127:112] * qx[127:112]) >> SAR[5:0]
  temp[31:16] = (qy[127:112] * qx[111: 96] + qy[111: 96] * qx[127:112]) >> SAR[5:0]

if upd4 == 0:  // normal radix 2
  {temp[31:0], qu[ 95:  0]} => store128(rs1[31:0])
if upd4 == 1:  // radix2 last second stage
  {
       temp[31:0],
       qu[ 95: 64],
       qy[ 63: 48] >> sel4,
       qy[ 47: 32] >> sel4,
       qy[ 31: 16] >> sel4,
       qy[ 15:  0] >> sel4
  } => store128(rs1[31:0])
if upd4 == 2:  // radix2 last stage
  {
       temp[31:0],
       qy[ 63: 48] >> sel4,
       qy[ 47: 32] >> sel4,
       qu[ 95: 64],
       qy[ 31: 16] >> sel4,
       qy[ 15:  0] >> sel4
  } => store128(rs1[31:0])

  rs1[31:0] = rs1[31:0] + rs2[31:0]
```

Schedule

Operands	Use	Def
qu	2	-
rs1	1	1
rs2	1	-
qx	1	-
qy	1	-
SAR	1	-

Format

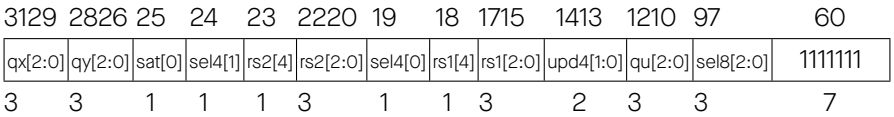


Figure 4.7-270. ESP.FFT.CMUL.S16.ST.XP

4.7.2.271 ESP.FFT.R2BF.S16

Syntax

ESP.FFT.R2BF.S16 qz,qv,qx,qy,sel2,sat

Description

This instruction performs the radix-2 butterfly operation on the 16-byte values in registers qx and qy, with a data bit width of 16 bits. Some of the calculation results are written to the register qz, while others are written to the register qv.

Operation

```
if sel2==0:
  op_a[127:  0] = {qy[ 63:  0], qx[ 63:  0]}
  op_b[127:  0] = {qy[127: 64], qx[127: 64]}
if sel2==1:
  op_a[127:  0] = {qy[ 95: 64], qy[ 31:  0], qx[ 95: 64], qx[ 31:  0]}
  op_b[127:  0] = {qy[127: 96], qy[ 63: 32], qx[127: 96], qx[ 63: 32]}

  qz[ 15:  0] = op_a[ 15:  0] + op_b[ 15:  0]
  qz[ 31: 16] = op_a[ 31: 16] + op_b[ 31: 16]
  qz[ 47: 32] = op_a[ 47: 32] + op_b[ 47: 32]
  qz[ 63: 48] = op_a[ 63: 48] + op_b[ 63: 48]
  qz[ 79: 64] = op_a[ 15:  0] - op_b[ 15:  0]
  qz[ 95: 80] = op_a[ 31: 16] - op_b[ 31: 16]
  qz[111: 96] = op_a[ 47: 32] - op_b[ 47: 32]
  qz[127:112] = op_a[ 63: 48] - op_b[ 63: 48]

  qv[ 15:  0] = op_a[ 79: 64] + op_b[ 79: 64]
  qv[ 31: 16] = op_a[ 95: 80] + op_b[ 95: 80]
  qv[ 47: 32] = op_a[111: 96] + op_b[111: 96]
  qv[ 63: 48] = op_a[127:112] + op_b[127:112]
  qv[ 79: 64] = op_a[ 79: 64] - op_b[ 79: 64]
  qv[ 95: 80] = op_a[ 95: 80] - op_b[ 95: 80]
  qv[111: 96] = op_a[111: 96] - op_b[111: 96]
  qv[127:112] = op_a[127:112] - op_b[127:112]
```

Schedule

Operands	Use	Def
qz	-	1
qv	-	1
qx	1	-
qy	1	-

Format

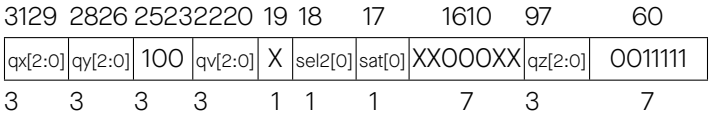


Figure 4.7-271. ESP.FFT.R2BF.S16

4.7.2.272 ESP.FFT.R2BF.S16.ST.INCP

Syntax

ESP.FFT.R2BF.S16.ST.INCP qz,qx,qy,rs1,sel4,sat

Description

This instruction performs the radix-2 butterfly operation on the 16-byte values in registers `qx` and `qy`, and the data bit width is 16 bits. Some of the calculation results are written to the register `qz`, while others perform an arithmetic shift and are written into the memory address indicated by `rs1`. After the access is completed, the value in the register `rs1` is incremented by 16.

Operation

```
qz[ 15:  0] = qx[ 15:  0] - qy[ 15:  0]
qz[ 31: 16] = qx[ 31: 16] - qy[ 31: 16]
qz[ 47: 32] = qx[ 47: 32] - qy[ 47: 32]
qz[ 63: 48] = qx[ 63: 48] - qy[ 63: 48]
qz[ 79: 64] = qx[ 79: 64] - qy[ 79: 64]
qz[ 95: 80] = qx[ 95: 80] - qy[ 95: 80]
qz[111: 96] = qx[111: 96] - qy[111: 96]
qz[127:112] = qx[127:112] - qy[127:112]

{
  (qx[127:112] + qy[127:112]) >> sel4,
  (qx[111: 96] + qy[111: 96]) >> sel4,
  (qx[ 95: 80] + qy[ 95: 80]) >> sel4,
  (qx[ 79: 64] + qy[ 79: 64]) >> sel4,
  (qx[ 63: 48] + qy[ 63: 48]) >> sel4,
  (qx[ 47: 32] + qy[ 47: 32]) >> sel4,
  (qx[ 31: 16] + qy[ 31: 16]) >> sel4,
  (qx[ 15:  0] + qy[ 15:  0]) >> sel4
} => store128(rs1[31:0])
rs1[31:0] = rs1[31:0] + 16
```

Schedule

Operands	Use	Def
qz	-	1
rs1	1	1
qx	1	-
qy	1	-

Format

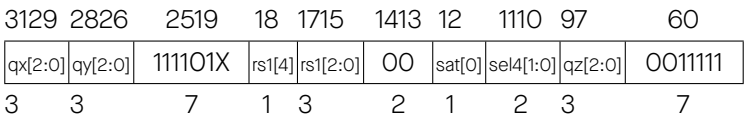


Figure 4.7-272. ESP.FFT.R2BF.S16.ST.INCP

4.7.2.273 ESP.FFT.VST.R32.DECP

Syntax

ESP.FFT.VST.R32.DECP qu,rs1,sel2

Description

It is a dedicated FFT instruction. This instruction divides data in register `qu` into 8 segments of 16-bit data and performs an arithmetic right shift on them by 0 or 1 depending on the immediate number `sel2`, and finally writes the result to the memory address indicated by register `rs1` in word big-endian order. After the access is completed, the value in register `rs1` is decreased by 16.

Operation

```
{
  qu[ 31: 16] >> sel2,
  qu[ 15:  0] >> sel2,
  qu[ 63: 48] >> sel2,
  qu[ 47: 32] >> sel2,
  qu[ 95: 80] >> sel2,
  qu[ 79: 64] >> sel2,
  qu[127:112] >> sel2,
  qu[111: 96] >> sel2
} => store128(rs1[31:0])
rs1[31:0] = rs1[31:0] - 16
```

Schedule

Operands	Use	Def
qu	2	-
rs1	1	1

Format

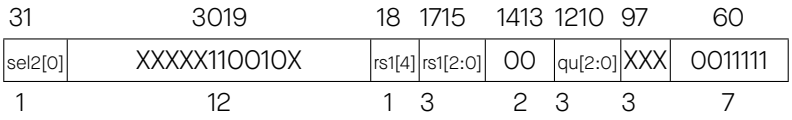


Figure 4.7-273. ESP.FFT.VST.R32.DECP

4.7.2.274 ESP.LD.128.USAR.IP

Syntax

ESP.LD.128.USAR.IP qu,rs1,imm16

Description

This instruction loads 16-byte data from memory to the register `qu`. Meanwhile, it saves the value of the lower 4 bits in the register `rs1` to the special register `SAR_BYTE`. After the access is completed, the value in the register `rs1` is incremented by an 8-bit sign-extended constant in the instruction code segment, left-shifted by 4.

imm starts at -2048, ends at 2032, and steps by 16.

Operation

```
qu[127:0] = load128(rs1[31:0])
SAR_BYTE = rs1[3:0]
rs1[31:0] = rs1[31:0] + imm
```

Schedule

Operands	Use	Def
qu	-	2
rs1	1	1
SAR_BYTE	-	1

Format

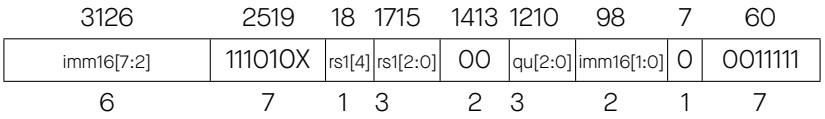


Figure 4.7-274. ESP.LD.128.USAR.IP

4.7.2.275 ESP.LD.128.USAR.XP

Syntax

ESP.LD.128.USAR.XP qu,rs1,rs2

Description

This instruction loads 16-byte data from memory into the register `qu`. Meanwhile, it saves the value of the lower 4 bits in `rs1` into the special register `SAR_BYTE`. After the access is completed, the value in the register `rs1` is incremented by the value in the register `rs2`.

Operation

```
qu[127:0] = load128(rs1[31:0])
SAR_BYTE = rs1[3:0]
rs1[31:0] = rs1[31:0] + rs2[31:0]
```

Schedule

Operands	Use	Def
qu	-	2
rs1	1	1
rs2	1	-
SAR_BYTE	-	1

Format

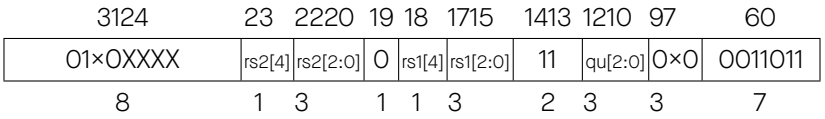


Figure 4.7-275. ESP.LD.128.USAR.XP

4.7.2.276 ESP.LD.XACC.IP

Syntax

ESP.LD.XACC.IP rs1,imm8

Description

Loads 64-bit data from memory, and saves its lower 40 bits to the special register `XACC`. After the access is completed, the value in the register `rs1` is incremented by an 8-bit sign-extended constant in the instruction code segment left-shifted by 3.

imm starts at -1024, ends at 1016, and the step is 8.

Operation

```
DataIn[63:0] = load64(rs1[31:0])
XACC[39:0] = DataIn[39:0]
rs1[31:0] = rs1[31:0] + imm
```

Schedule

Operands	Use	Def
XACC	-	2
rs1	1	1

Format

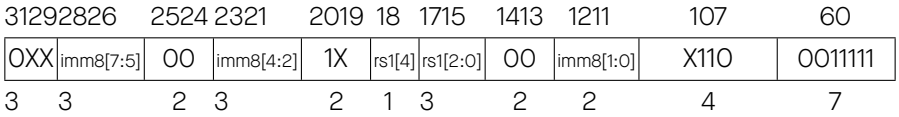


Figure 4.7-276. ESP.LD.XACC.IP

4.7.2.277 ESP.LDQA.S16.128.IP

Syntax

ESP.LDQA.S16.128.IP rs1,imm16

Description

This instruction loads 16-byte data from memory, divides it into 8 segments of 16 bits, sign-extends each segment to 64 bits, and then stores the results to the 256-bit special registers `QACC_L` and `QACC_H` respectively. After the access is completed, the value in the register `rs1` is incremented by an 8-bit sign-extended constant in the instruction code segment, left-shifted by 4.
imm starts at -2048, ends at 2032, with a step of 16.

Operation

```
dataIn[127:0] = load128(rs1[31:0])
QACC_L[ 63:  0] = {48{dataIn[15]}, dataIn[ 15:  0]}
QACC_L[127: 64] = {48{dataIn[31]}, dataIn[ 31: 16]}
QACC_L[191:128] = {48{dataIn[47]}, dataIn[ 47: 32]}
QACC_L[255:192] = {48{dataIn[63]}, dataIn[ 63: 48]}

QACC_H[ 63:  0] = {48{dataIn[79]}, dataIn[ 79: 64]}
QACC_H[127: 64] = {48{dataIn[95]}, dataIn[ 95: 80]}
QACC_H[191:128] = {48{dataIn[111]}, dataIn[111: 96]}
QACC_H[159:192] = {48{dataIn[127]}, dataIn[127:112]}

rs1[31:0] = rs1[31:0] + imm
```

Schedule

Operands	Use	Def
QACC_H	-	2
QACC_L	-	2
rs1	1	1

Format

3127	26	2524	2321	2019	18	1715	1413	129	87	60
0XX11	imm16[7]	00	imm16[6:4]	1X	rs1[4]	rs1[2:0]	00	imm16[3:0]	01	0011111
5	1	2	3	2	1	3	2	4	2	7

Figure 4.7-277. ESP.LDQA.S16.128.IP

4.7.2.278 ESP.LDQA.S16.128.XP

Syntax

ESP.LDQA.S16.128.XP rs1,rs2

Description

This instruction loads 16-byte data from memory, divides it into 8 segments of 16 bits, sign-extends each segment to 64 bits, and then stores the results in the 256-bit special registers `QACC_L` and `QACC_H` respectively. After the access is completed, the value in the register `rs1` is incremented by the value in the register `rs2`.

Operation

```
dataIn[127:0] = load128(rs1[31:0])
QACC_L[ 63:  0] = {48{dataIn[15]}, dataIn[ 15:  0]}
QACC_L[127: 64] = {48{dataIn[31]}, dataIn[ 31: 16]}
QACC_L[191:128] = {48{dataIn[47]}, dataIn[ 47: 32]}
QACC_L[255:192] = {48{dataIn[63]}, dataIn[ 63: 48]}

QACC_H[ 63:  0] = {48{dataIn[79]}, dataIn[ 79: 64]}
QACC_H[127: 64] = {48{dataIn[95]}, dataIn[ 95: 80]}
QACC_H[191:128] = {48{dataIn[111]}, dataIn[111: 96]}
QACC_H[159:192] = {48{dataIn[127]}, dataIn[127:112]}

rs1[31:0] = rs1[31:0] + rs2[31:0]
```

Schedule

Operands	Use	Def
QACC_H	-	2
QACC_L	-	2
rs1	1	1
rs2	1	-

Format

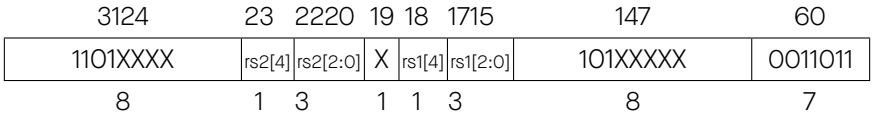


Figure 4.7-278. ESP.LDQA.S16.128.XP

4.7.2.279 ESP.LDQA.S8.128.IP

Syntax

ESP.LDQA.S8.128.IP rs1,imm16

Description

This instruction loads 16-byte data from memory, divides it into 16 segments of 8 bits, sign-extends each segment to 32 bits, and then stores the results in the 256-bit special registers `QACC_L` and `QACC_H` respectively. After the access is completed, the value in the register `rs1` is incremented by the 8-bit sign-extended constant in the instruction code segment, left-shifted by 4.
imm starts at -2048, ends at 2032, with a step of 16.

Operation

```
DataIn[127:0] = load128(rs1[31:0])
QACC_L[31 :  0] = {24{DataIn[ 7]}, DataIn[7  :  0]}
QACC_L[63 : 32] = {24{DataIn[15]}, DataIn[15 :  8]}
QACC_L[95 : 64] = {24{DataIn[23]}, DataIn[23 : 16]}
...
QACC_L[255:224] = {24{DataIn[63]}, DataIn[63 : 56]}
QACC_H[31 :  0] = {24{DataIn[71]}, DataIn[71 : 64]}
QACC_H[63 : 32] = {24{DataIn[79]}, DataIn[79 : 72]}
QACC_H[95 : 64] = {24{DataIn[87]}, DataIn[87 : 80]}
...
QACC_H[255:224] = {24{DataIn[127]}, DataIn[127:120]}

rs1[31:0] = rs1[31:0] + imm
```

Schedule

Operands	Use	Def
QACC_H	-	2
QACC_L	-	2
rs1	1	1

Format

31..27	26	25..24	23..21	20..19	18	17..15	14..13	12..9	8..7	6..0
0XX01	imm16[7]	00	imm16[6:4]	1X	rs1[4]	rs1[2:0]	00	imm16[3:0]	01	0011111
5	1	2	3	2	1	3	2	4	2	7

Figure 4.7-279. ESP.LDQA.S8.128.IP

4.7.2.280 ESP.LDQA.S8.128.XP

Syntax

ESP.LDQA.S8.128.XP rs1,rs2

Description

This instruction loads 16-byte data from memory, divides it into 16 segments of 8 bits, sign-extends each segment to 32 bits, and then stores the results in the 256-bit special registers `QACC_L` and `QACC_H` respectively. After the access is completed, the value in the register `rs1` is incremented by the value in the register `rs2`.

Operation

```
DataIn[127:0] = load128(rs1[31:0])
QACC_L[31 :  0] = {24{DataIn[ 7]}, DataIn[7  :  0]}
QACC_L[63 : 32] = {24{DataIn[15]}, DataIn[15 :  8]}
QACC_L[95 : 64] = {24{DataIn[23]}, DataIn[23 : 16]}
...
QACC_L[255:224] = {24{DataIn[63]}, DataIn[63 : 56]}
QACC_H[31 :  0] = {24{DataIn[71]}, DataIn[71 : 64]}
QACC_H[63 : 32] = {24{DataIn[79]}, DataIn[79 : 72]}
QACC_H[95 : 64] = {24{DataIn[87]}, DataIn[87 : 80]}
...
QACC_H[255:224] = {24{DataIn[127]}, DataIn[127:120]}

rs1[31:0] = rs1[31:0] + rs2[31:0]
```

Schedule

Operands	Use	Def
QACC_H	-	2
QACC_L	-	2
rs1	1	1
rs2	1	-

Format

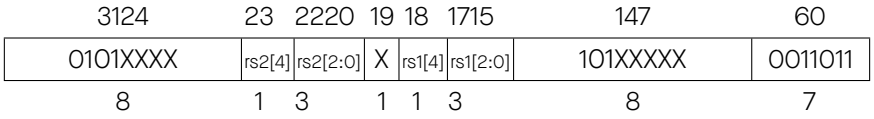


Figure 4.7-280. ESP.LDQA.S8.128.XP

4.7.2.281 ESP.LDQA.U16.128.IP

Syntax

ESP.LDQA.U16.128.IP rs1,imm16

Description

This instruction loads 16-byte data from memory, divides it into 8 segments of 16 bits, zero-extends each segment to 64 bits, and then stores the results in the 256-bit special registers `QACC_L` and `QACC_H` respectively. After the access is completed, the value in the register `rs1` is incremented by an 8-bit sign-extended constant in the instruction code segment, left-shifted by 4.
imm starts at -2048, ends at 2032, with a step of 16.

Operation

```
dataIn[127:0] = load128(rs1[31:0])
QACC_L[ 63:  0] = {48{0}, dataIn[ 15:  0]}
QACC_L[127: 64] = {48{0}, dataIn[ 31: 16]}
QACC_L[191:128] = {48{0}, dataIn[ 47: 32]}
QACC_L[255:192] = {48{0}, dataIn[ 63: 48]}

QACC_H[ 63:  0] = {48{0}, dataIn[ 79: 64]}
QACC_H[127: 64] = {48{0}, dataIn[ 95: 80]}
QACC_H[191:128] = {48{0}, dataIn[111: 96]}
QACC_H[159:192] = {48{0}, dataIn[127:112]}

rs1[31:0] = rs1[31:0] + imm
```

Schedule

Operands	Use	Def
QACC_H	-	2
QACC_L	-	2
rs1	1	1

Format

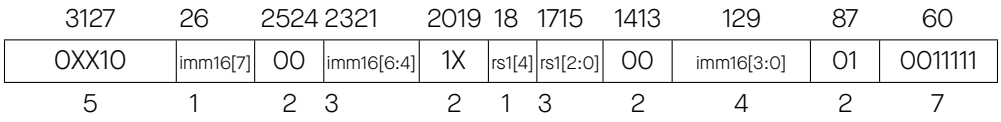


Figure 4.7-281. ESP.LDQA.U16.128.IP

4.7.2.282 ESP.LDQA.U16.128.XP

Syntax

ESP.LDQA.U16.128.XP rs1,rs2

Description

This instruction loads 16-byte data from memory, divides it into 8 segments of 16 bits, zero-extends each segment to 64 bits, and then stores the results in the 256-bit special registers `QACC_L` and `QACC_H` respectively. After the access is completed, the value in the register `rs1` is incremented by the value in the register `rs2`.

Operation

```
dataIn[127:0] = load128(rs1[31:0])
QACC_L[ 63:  0] = {48{0}, dataIn[ 15:  0]}
QACC_L[127: 64] = {48{0}, dataIn[ 31: 16]}
QACC_L[191:128] = {48{0}, dataIn[ 47: 32]}
QACC_L[255:192] = {48{0}, dataIn[ 63: 48]}

QACC_H[ 63:  0] = {48{0}, dataIn[ 79: 64]}
QACC_H[127: 64] = {48{0}, dataIn[ 95: 80]}
QACC_H[191:128] = {48{0}, dataIn[111: 96]}
QACC_H[159:192] = {48{0}, dataIn[127:112]}

rs1[31:0] = rs1[31:0] + rs2[31:0]
```

Schedule

Operands	Use	Def
QACC_H	-	2
QACC_L	-	2
rs1	1	1
rs2	1	-

Format

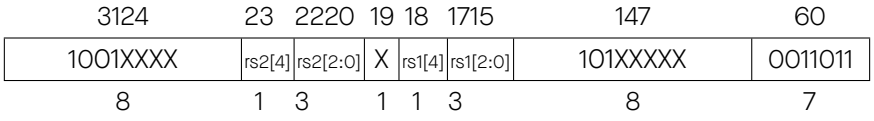


Figure 4.7-282. ESP.LDQA.U16.128.XP

4.7.2.283 ESP.LDQA.U8.128.IP

Syntax

ESP.LDQA.U8.128.IP rs1,imm16

Description

This instruction loads 16-byte data from memory, divides it into 16 segments of 8 bits, zero-extends each segment to 32 bits, and then stores the results in the 256-bit special registers `QACC_L` and `QACC_H` respectively. After the access is completed, the value in the register `rs1` is incremented by the 8-bit sign-extended constant in the instruction code segment, which is left-shifted by 4.
imm starts at -2048, ends at 2032, and steps by 16.

Operation

```
DataIn[127:0] = load128(rs1[31:0])
QACC_L[31 :  0] = {24{0}, DataIn[7  :  0]}
QACC_L[63 : 32] = {24{0}, DataIn[15 :  8]}
QACC_L[95 : 64] = {24{0}, DataIn[23 : 16]}
...
QACC_L[255:224] = {24{0}, DataIn[63 : 56]}
QACC_H[31 :  0] = {24{0}, DataIn[71 : 64]}
QACC_H[63 : 32] = {24{0}, DataIn[79 : 72]}
QACC_H[95 : 64] = {24{0}, DataIn[87 : 80]}
...
QACC_H[255:224] = {24{0}, DataIn[127:120]}

rs1[31:0] = rs1[31:0] + imm
```

Schedule

Operands	Use	Def
QACC_H	-	2
QACC_L	-	2
rs1	1	1

Format

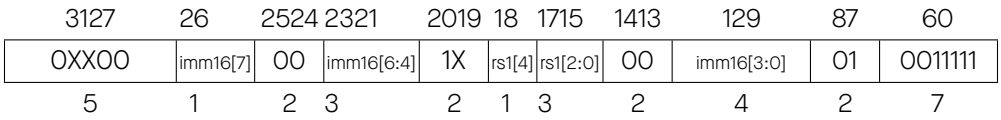


Figure 4.7-283. ESP.LDQA.U8.128.IP

4.7.2.284 ESP.LDQA.U8.128.XP

Syntax

ESP.LDQA.U8.128.XP rs1,rs2

Description

This instruction loads 16-byte data from memory, divides it into 16 segments of 8 bits, zero-extends each segment to 32 bits, and then stores the results in the 256-bit special registers `QACC_L` and `QACC_H` respectively. After the access is completed, the value in the register `rs1` is incremented by the value in the register `rs2`.

Operation

```
DataIn[127:0] = load128(rs1[31:0])
QACC_L[31 :  0] = {24{0}, DataIn[7  :  0]}
QACC_L[63 : 32] = {24{0}, DataIn[15 :  8]}
QACC_L[95 : 64] = {24{0}, DataIn[23 : 16]}
...
QACC_L[255:224] = {24{0}, DataIn[63 : 56]}
QACC_H[31 :  0] = {24{0}, DataIn[71 : 64]}
QACC_H[63 : 32] = {24{0}, DataIn[79 : 72]}
QACC_H[95 : 64] = {24{0}, DataIn[87 : 80]}
...
QACC_H[255:224] = {24{0}, DataIn[127:120]}

rs1[31:0] = rs1[31:0] + rs2[31:0]
```

Schedule

Operands	Use	Def
QACC_H	-	2
QACC_L	-	2
rs1	1	1
rs2	1	-

Format

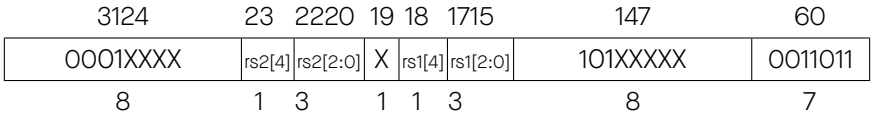


Figure 4.7-284. ESP.LDQA.U8.128.XP

4.7.2.285 ESP.VLDBC.16.IP

Syntax

ESP.VLDBC.16.IP qu,rs1,imm2

Description

These instructions load 16-bit data from memory and broadcast the data to register qu, with the pointer updated post-loading by an immediate value.

imm starts at -256, ends at 254, and steps by 2.

Operation

```
qu[127:0] = {8{load16({rs1[31:0]})}}
rs1[31:0] = rs1[31:0] + imm
```

Schedule

Operands	Use	Def
qu	-	2
rs1	1	1

Format

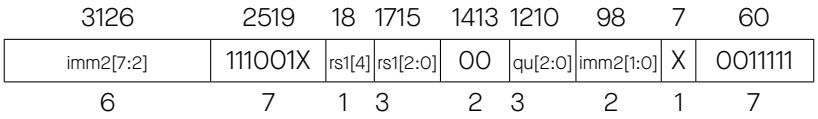


Figure 4.7-285. ESP.VLDBC.16.IP

4.7.2.286 ESP.VLDBC.16.XP

Syntax

ESP.VLDBC.16.XP qu,rs1,rs2

Description

These instructions load 16-bit data from memory and broadcast to the register `qu`, with the pointer updated post-loading by the value in the register `rs2`.

Operation

```
qu[127:0] = {8{load16({rs1[31:0]})}}
rs1[31:0] = rs1[31:0] + rs2[31:0]
```

Schedule

Operands	Use	Def
qu	-	2
rs1	1	1
rs2	1	-

Format

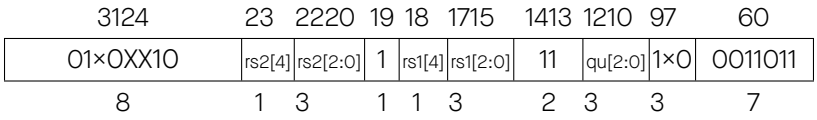


Figure 4.7-286. ESP.VLDBC.16.XP

4.7.2.287 ESP.VLDBC.32.IP

Syntax

ESP.VLDBC.32.IP qu,rs1,imm4

Description

These instructions load 32-bit data from memory and broadcast to the register qu, with the pointer post updated by an immediate value.

imm starts at -512, ends at 508, and steps by 4.

Operation

```
    qu[127:0] = {4{load32({rs1[31:0]})}}
    rs1[31:0] = rs1[31:0] + imm
```

Schedule

Operands	Use	Def
qu	-	2
rs1	1	1

Format

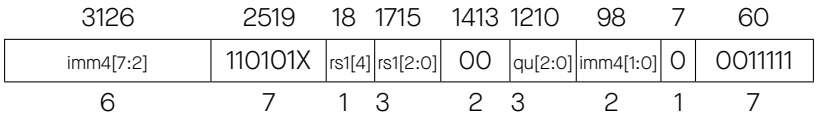


Figure 4.7-287. ESP.VLDBC.32.IP

4.7.2.288 ESP.VLDBC.32.XP

Syntax

ESP.VLDBC.32.XP qu,rs1,rs2

Description

These instructions load 32 bits of data from memory and broadcast to the register `qu`, with the pointer updated post-loading by the value in the register `rs2`.

Operation

```
qu[127:0] = {4{load32({rs1[31:0]})}}
rs1[31:0] = rs1[31:0] + rs2[31:0]
```

Schedule

Operands	Use	Def
qu	-	2
rs1	1	1
rs2	1	-

Format

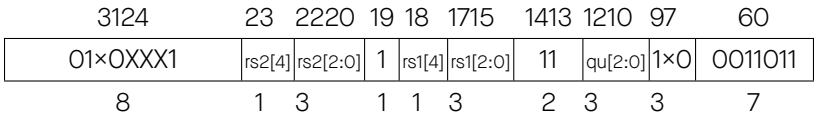


Figure 4.7-288. ESP.VLDBC.32.XP

4.7.2.289 ESP.VLDBC.8.IP

Syntax

ESP.VLDBC.8.IP qu,rs1,imm1

Description

These instructions load 8 bits of data from memory and broadcast to the register qu with a pointer post updated by an immediate value.

imm starts at -128, ends at 127, and the step is 1.

Operation

```
qu[127:0] = {16{load8({rs1[31:0]})}}
rs1[31:0] = rs1[31:0] + imm
```

Schedule

Operands	Use	Def
qu	-	2
rs1	1	1

Format

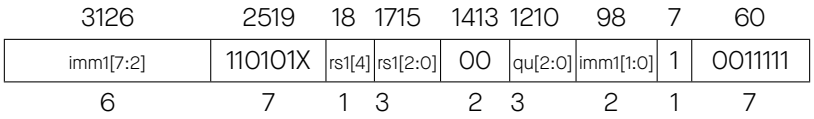


Figure 4.7-289. ESP.VLDBC.8.IP

4.7.2.290 ESP.VLDBC.8.XP

Syntax

ESP.VLDBC.8.XP qu,rs1,rs2

Description

These instructions load 8 bits of data from memory and broadcast to the register `qu`, with the pointer updated post-loading by the value in the register `rs2`.

Operation

```
qu[127:0] = {16{load8({rs1[31:0]})}}
rs1[31:0] = rs1[31:0] + rs2[31:0]
```

Schedule

Operands	Use	Def
qu	-	2
rs1	1	1
rs2	1	-

Format

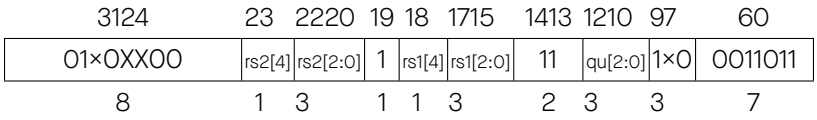


Figure 4.7-290. ESP.VLDBC.8.XP

4.7.2.291 ESP.VLDEXT.S16.IP

Syntax

ESP.VLDEXT.S16.IP qu,qz,rs1,imm16

Description

This instruction loads 128 bit of data from memory, sign-extends each 16-bit segment to 32 bits, with pointer post-incremented by an immediate value.

imm is start at -2048, end at 2032, step is 16.

Operation

```
    DataIn[127:0] = load128(rs1[31:0])
    qu[ 31:  0] = { {16{DataIn[ 15]}}, DataIn[ 15:  0] }
    qu[ 63: 32] = { {16{DataIn[ 31]}}, DataIn[ 31: 16] }
    qu[ 95: 64] = { {16{DataIn[ 47]}}, DataIn[ 47: 32] }
    qu[127: 96] = { {16{DataIn[ 63]}}, DataIn[ 63: 48] }
    qz[ 31:  0] = { {16{DataIn[ 79]}}, DataIn[ 79: 64] }
    qz[ 63: 32] = { {16{DataIn[ 95]}}, DataIn[ 95: 80] }
    qz[ 95: 64] = { {16{DataIn[111]}}, DataIn[111: 96] }
    qz[127: 96] = { {16{DataIn[127]}}, DataIn[127:112] }
    rs1[31:0] = rs1[31:0] + imm
```

Schedule

Operands	Use	Def
qu	-	2
qv	-	2
rs1	1	1

Format

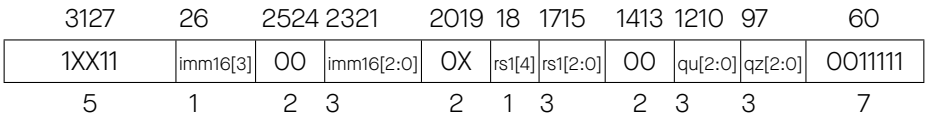


Figure 4.7-291. ESP.VLDEXT.S16.IP

4.7.2.292 ESP.VLDEXT.S16.XP

Syntax

ESP.VLDEXT.S16.XP qu,qz,rs1,rs2

Description

This instruction loads 128-bit data from memory and sign-extends the data from a 16-bit segment to a 32-bit segment.

Operation

```
DataIn[127:0] = load128(rs1[31:0])
qu[ 31:  0] = { {16{DataIn[ 15]}}, DataIn[ 15:  0] }
qu[ 63: 32] = { {16{DataIn[ 31]}}, DataIn[ 31: 16] }
qu[ 95: 64] = { {16{DataIn[ 47]}}, DataIn[ 47: 32] }
qu[127: 96] = { {16{DataIn[ 63]}}, DataIn[ 63: 48] }
qz[ 31:  0] = { {16{DataIn[ 79]}}, DataIn[ 79: 64] }
qz[ 63: 32] = { {16{DataIn[ 95]}}, DataIn[ 95: 80] }
qz[ 95: 64] = { {16{DataIn[111]}}, DataIn[111: 96] }
qz[127: 96] = { {16{DataIn[127]}}, DataIn[127:112] }
rs1[31:0] = rs1[31:0] + rs2[31:0]
```

Schedule

Operands	Use	Def
qu	-	2
qv	-	2
rs1	1	1
rs2	1	-

Format

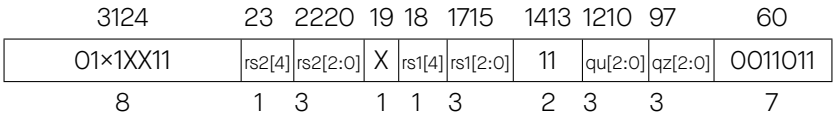


Figure 4.7-292. ESP.VLDEXT.S16.XP

4.7.2.293 ESP.VLDEXT.S8.IP

Syntax

ESP.VLDEXT.S8.IP qu,qz,rs1,imm16

Description

This instruction loads 128 bit data from memory, and sign-extend the data as 8-bit segment to 16-bit, with pointer post update by immediate value.

imm starts at -2048, ends at 2032, with a step of step 16.

Operation

```
DataIn[127:0] = load128(rs1[31:0])
qu[ 15:  0] = { {8{DataIn[  7]}}, DataIn[  7:  0] }
qu[ 31: 16] = { {8{DataIn[ 15]}}, DataIn[ 15:  8] }
qu[ 47: 32] = { {8{DataIn[ 23]}}, DataIn[ 23: 16] }
qu[ 63: 48] = { {8{DataIn[ 31]}}, DataIn[ 31: 24] }
qu[ 79: 64] = { {8{DataIn[ 39]}}, DataIn[ 39: 32] }
qu[ 95: 80] = { {8{DataIn[ 47]}}, DataIn[ 47: 40] }
qu[111: 96] = { {8{DataIn[ 55]}}, DataIn[ 55: 48] }
qu[127:112] = { {8{DataIn[ 63]}}, DataIn[ 63: 56] }
qz[ 15:  0] = { {8{DataIn[ 71]}}, DataIn[ 71: 64] }
qz[ 31: 16] = { {8{DataIn[ 79]}}, DataIn[ 79: 72] }
qz[ 47: 32] = { {8{DataIn[ 87]}}, DataIn[ 87: 80] }
qz[ 63: 48] = { {8{DataIn[ 95]}}, DataIn[ 95: 88] }
qz[ 79: 64] = { {8{DataIn[103]}}, DataIn[103: 96] }
qz[ 95: 80] = { {8{DataIn[111]}}, DataIn[111:104] }
qz[111: 96] = { {8{DataIn[119]}}, DataIn[119:112] }
qz[127:112] = { {8{DataIn[127]}}, DataIn[127:120] }
rs1[31:0] = rs1[31:0] + imm
```

Schedule

Operands	Use	Def
qu	-	2
qv	-	2
rs1	1	1

Format

3127	26	2524	2321	2019	18	1715	1413	1210	97	60
1XX01	imm16[3]	00	imm16[2:0]	0X	rs1[4]	rs1[2:0]	00	qu[2:0]	qz[2:0]	0011111
5	1	2	3	2	1	3	2	3	3	7

Figure 4.7-293. ESP.VLDEXT.S8.IP

4.7.2.294 ESP.VLDEXT.S8.XP

Syntax

ESP.VLDEXT.S8.XP qu,qz,rs1,rs2

Description

This instruction loads 128-bit data from memory and sign-extends the data from an 8-bit segment to a 16-bit segment.

Operation

```
DataIn[127:0] = load128(rs1[31:0])
qu[ 15:  0] = { {8{DataIn[  7]}}, DataIn[  7:  0] }
qu[ 31: 16] = { {8{DataIn[ 15]}}, DataIn[ 15:  8] }
qu[ 47: 32] = { {8{DataIn[ 23]}}, DataIn[ 23: 16] }
qu[ 63: 48] = { {8{DataIn[ 31]}}, DataIn[ 31: 24] }
qu[ 79: 64] = { {8{DataIn[ 39]}}, DataIn[ 39: 32] }
qu[ 95: 80] = { {8{DataIn[ 47]}}, DataIn[ 47: 40] }
qu[111: 96] = { {8{DataIn[ 55]}}, DataIn[ 55: 48] }
qu[127:112] = { {8{DataIn[ 63]}}, DataIn[ 63: 56] }
qv[ 15:  0] = { {8{DataIn[ 71]}}, DataIn[ 71: 64] }
qv[ 31: 16] = { {8{DataIn[ 79]}}, DataIn[ 79: 72] }
qv[ 47: 32] = { {8{DataIn[ 87]}}, DataIn[ 87: 80] }
qv[ 63: 48] = { {8{DataIn[ 95]}}, DataIn[ 95: 88] }
qv[ 79: 64] = { {8{DataIn[103]}}, DataIn[103: 96] }
qv[ 95: 80] = { {8{DataIn[111]}}, DataIn[111:104] }
qv[111: 96] = { {8{DataIn[119]}}, DataIn[119:112] }
qv[127:112] = { {8{DataIn[127]}}, DataIn[127:120] }
rs1[31:0] = rs1[31:0] + rs2[31:0]
```

Schedule

Operands	Use	Def
qu	-	2
qv	-	2
rs1	1	1
rs2	1	-

Format

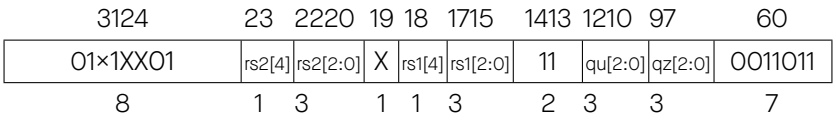


Figure 4.7-294. ESP.VLDEXT.S8.XP

4.7.2.295 ESP.VLDEXT.U16.IP

Syntax

ESP.VLDEXT.U16.IP qu,qz,rs1,imm16

Description

This instruction loads 128-bit data from memory and zero-extends the data from a 16-bit segment to a 32-bit segment.

`imm` starts at -2048, ends at 2032, and the step is 16.

Operation

```
    DataIn[127:0] = load128(rs1[31:0])
    qu[ 31:  0] = { {16{0}}, DataIn[ 15:  0] }
    qu[ 63: 32] = { {16{0}}, DataIn[ 31: 16] }
    qu[ 95: 64] = { {16{0}}, DataIn[ 47: 32] }
    qu[127: 96] = { {16{0}}, DataIn[ 63: 48] }
    qz[ 31:  0] = { {16{0}}, DataIn[ 79: 64] }
    qz[ 63: 32] = { {16{0}}, DataIn[ 95: 80] }
    qz[ 95: 64] = { {16{0}}, DataIn[111: 96] }
    qz[127: 96] = { {16{0}}, DataIn[127:112] }
    rs1[31:0] = rs1[31:0] + imm
```

Schedule

Operands	Use	Def
qu	-	2
qv	-	2
rs1	1	1

Format

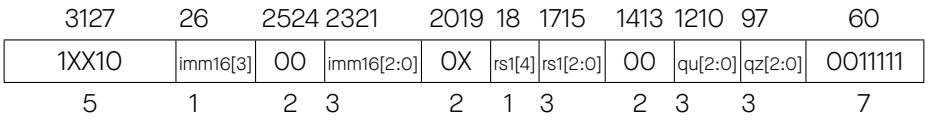


Figure 4.7-295. ESP.VLDEXT.U16.IP

4.7.2.296 ESP.VLDEXT.U16.XP

Syntax

ESP.VLDEXT.U16.XP qu,qz,rs1,rs2

Description

This instruction loads 128-bit data from memory and zero-extends the data from a 16-bit segment to a 32-bit segment.

Operation

```
DataIn[127:0] = load128(rs1[31:0])
qu[ 31:  0] = { {16{0}}, DataIn[ 15:  0] }
qu[ 63: 32] = { {16{0}}, DataIn[ 31: 16] }
qu[ 95: 64] = { {16{0}}, DataIn[ 47: 32] }
qu[127: 96] = { {16{0}}, DataIn[ 63: 48] }
qz[ 31:  0] = { {16{0}}, DataIn[ 79: 64] }
qz[ 63: 32] = { {16{0}}, DataIn[ 95: 80] }
qz[ 95: 64] = { {16{0}}, DataIn[111: 96] }
qz[127: 96] = { {16{0}}, DataIn[127:112] }
rs1[31:0] = rs1[31:0] + rs2[31:0]
```

Schedule

Operands	Use	Def
qu	-	2
qv	-	2
rs1	1	1
rs2	1	-

Format

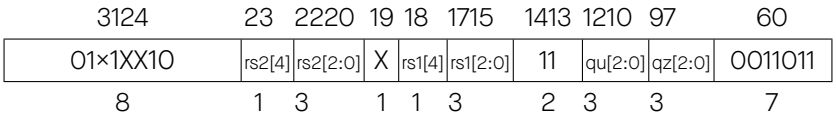


Figure 4.7-296. ESP.VLDEXT.U16.XP

4.7.2.297 ESP.VLDEXT.U8.IP

Syntax

ESP.VLDEXT.U8.IP qu,qz,rs1,imm16

Description

This instruction loads 128-bit data from memory and zero-extends the data from an 8-bit segment to a 16-bit segment.

`imm` starts at -2048, ends at 2032, and steps in increments of 16.

Operation

```
DataIn[127:0] = load128(rs1[31:0])
qu[ 15:  0] = { {8{0}}, DataIn[  7:  0] }
qu[ 31: 16] = { {8{0}}, DataIn[ 15:  8] }
qu[ 47: 32] = { {8{0}}, DataIn[ 23: 16] }
qu[ 63: 48] = { {8{0}}, DataIn[ 31: 24] }
qu[ 79: 64] = { {8{0}}, DataIn[ 39: 32] }
qu[ 95: 80] = { {8{0}}, DataIn[ 47: 40] }
qu[111: 96] = { {8{0}}, DataIn[ 55: 48] }
qu[127:112] = { {8{0}}, DataIn[ 63: 56] }
qz[ 15:  0] = { {8{0}}, DataIn[ 71: 64] }
qz[ 31: 16] = { {8{0}}, DataIn[ 79: 72] }
qz[ 47: 32] = { {8{0}}, DataIn[ 87: 80] }
qz[ 63: 48] = { {8{0}}, DataIn[ 95: 88] }
qz[ 79: 64] = { {8{0}}, DataIn[103: 96] }
qz[ 95: 80] = { {8{0}}, DataIn[111:104] }
qz[111: 96] = { {8{0}}, DataIn[119:112] }
qz[127:112] = { {8{0}}, DataIn[127:120] }
rs1[31:0] = rs1[31:0] + imm
```

Schedule

Operands	Use	Def
qu	-	2
qv	-	2
rs1	1	1

Format

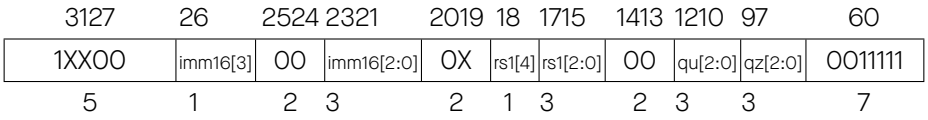


Figure 4.7-297. ESP.VLDEXT.U8.IP

4.7.2.298 ESP.VLDEXT.U8.XP

Syntax

ESP.VLDEXT.U8.XP qu,qz,rs1,rs2

Description

This instruction loads 128-bit data from memory and zero-extends the data from an 8-bit segment to a 16-bit segment.

Operation

```
DataIn[127:0] = load128(rs1[31:0])
qu[ 15:  0] = { {8{0}}, DataIn[  7:  0] }
qu[ 31: 16] = { {8{0}}, DataIn[ 15:  8] }
qu[ 47: 32] = { {8{0}}, DataIn[ 23: 16] }
qu[ 63: 48] = { {8{0}}, DataIn[ 31: 24] }
qu[ 79: 64] = { {8{0}}, DataIn[ 39: 32] }
qu[ 95: 80] = { {8{0}}, DataIn[ 47: 40] }
qu[111: 96] = { {8{0}}, DataIn[ 55: 48] }
qu[127:112] = { {8{0}}, DataIn[ 63: 56] }
qz[ 15:  0] = { {8{0}}, DataIn[ 71: 64] }
qz[ 31: 16] = { {8{0}}, DataIn[ 79: 72] }
qz[ 47: 32] = { {8{0}}, DataIn[ 87: 80] }
qz[ 63: 48] = { {8{0}}, DataIn[ 95: 88] }
qz[ 79: 64] = { {8{0}}, DataIn[103: 96] }
qz[ 95: 80] = { {8{0}}, DataIn[111:104] }
qz[111: 96] = { {8{0}}, DataIn[119:112] }
qz[127:112] = { {8{0}}, DataIn[127:120] }
rs1[31:0] = rs1[31:0] + rs2[31:0]
```

Schedule

Operands	Use	Def
qu	-	2
qv	-	2
rs1	1	1
rs2	1	-

Format

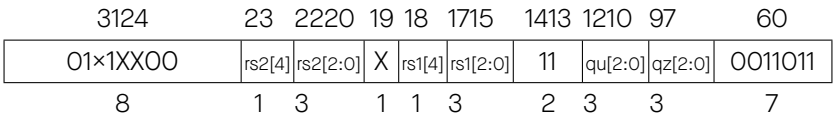


Figure 4.7-298. ESP.VLDEXT.U8.XP

4.7.2.299 ESP.VLDHBC.16.INCP

Syntax

ESP.VLDHBC.16.INCP qu,qz,rs1

Description

This instruction loads 16-byte data from memory, extends it to 256 bits in the following way, and assigns the result to the registers `qu` and `qz`. After the access, the value in the register `rs1` is incremented by 16.

Operation

```
dataIn[127:0] = load128(rs1[31:0])
qu  = {2{dataIn[ 63: 48]}, 2{dataIn[ 47: 32]}, 2{dataIn[ 31: 16]}, 2{dataIn[ 15: 0]} }
qz = {2{dataIn[127:112]}, 2{dataIn[111: 96]}, 2{dataIn[ 95: 80]}, 2{dataIn[ 79: 64]} }
rs1[31:0] = rs1[31:0] + 16
```

Schedule

Operands	Use	Def
qu	-	2
qz	-	2
rs1	1	1

Format

31..19	18	17..15	14..13	12..10	9..7	6..0
1XXXXXOOXXX1X	rs1[4]	rs1[2:0]	00	qu[2:0]	qz[2:0]	0011111
13	1	3	2	3	3	7

Figure 4.7-299. ESP.VLDHBC.16.INCP

4.7.2.300 ESP.LD.QACC.H.H.128.IP

Syntax

ESP.LD.QACC.H.H.128.IP rs1,imm16

Description

This instruction loads 16-byte data from memory into the special register `QACC_H[255:128]`. After the access is completed, the value in the register `rs1` is incremented by an 8-bit sign-extended constant in the instruction code segment, left-shifted by 4.
imm starts at -2048, ends at 2032, and steps by 16.

Operation

```
QACC_H[255:  128] = load128(rs1[31:0])
rs1[31:0] = rs1[31:0] + imm
```

Schedule

Operands	Use	Def
QACC_H	-	2
rs1	1	1

Format

31–27	26	25–24	23–21	20–19	18	17–15	14–13	12–9	8–7	6–0
0XX01	imm16[7]	00	imm16[6:4]	0X	rs1[4]	rs1[2:0]	00	imm16[3:0]	XX	0011111
5	1	2	3	2	1	3	2	4	2	7

Figure 4.7-300. ESP.LD.QACC.H.H.128.IP

4.7.2.301 ESP.LD.QACC.H.L.128.IP

Syntax

ESP.LD.QACC.H.L.128.IP rs1,imm16

Description

This instruction loads 16-byte data from memory into the special register `QACC_H[127:0]`. After the access is completed, the value in the register `rs1` is incremented by an 8-bit sign-extended constant in the instruction code segment, left-shifted by 4.
imm starts at -2048, ends at 2032, and steps by 16.

Operation

```
QACC_H[127:  0] = load128(rs1[31:0])
rs1[31:0] = rs1[31:0] + imm
```

Schedule

Operands	Use	Def
QACC_H	-	2
rs1	1	1

Format

31–27	26	25–24	23–21	20–19	18	17–15	14–13	12–9	8–7	6–0
0XX01	imm16[7]	00	imm16[6:4]	1X	rs1[4]	rs1[2:0]	00	imm16[3:0]	00	0011111
5	1	2	3	2	1	3	2	4	2	7

Figure 4.7-301. ESP.LD.QACC.H.L.128.IP

4.7.2.302 ESP.LD.QACC.L.H.128.IP

Syntax

ESP.LD.QACC.L.H.128.IP rs1,imm16

Description

This instruction loads 16-byte data from memory into the special register `QACC_L[255:128]`. After the access is completed, the value in the register `rs1` is incremented by an 8-bit sign-extended constant in the instruction code segment, left-shifted by 4.
imm starts at -2048, ends at 2032, and steps by 16.

Operation

```
QACC_L[255:  128] = load128(rs1[31:0])
rs1[31:0] = rs1[31:0] + imm
```

Schedule

Operands	Use	Def
QACC_L	-	2
rs1	1	1

Format

31–27	26	25–24	23–21	20–19	18	17–15	14–13	12–9	8–7	6–0
0XX00	imm16[7]	00	imm16[6:4]	0X	rs1[4]	rs1[2:0]	00	imm16[3:0]	XX	0011111
5	1	2	3	2	1	3	2	4	2	7

Figure 4.7-302. ESP.LD.QACC.L.H.128.IP

4.7.2.303 ESP.LD.QACC.L.L.128.IP

Syntax

ESP.LD.QACC.L.L.128.IP rs1,imm16

Description

This instruction loads 16-byte data from memory into the special register `QACC_L[127:0]`. After the access is completed, the value in the register `rs1` is incremented by an 8-bit sign-extended constant in the instruction code segment, left-shifted by 4.
imm starts at -2048, ends at 2032, and steps by 16.

Operation

```
QACC_L[127:  0] = load128(rs1[31:0])
rs1[31:0] = rs1[31:0] + imm
```

Schedule

Operands	Use	Def
QACC_L	-	2
rs1	1	1

Format

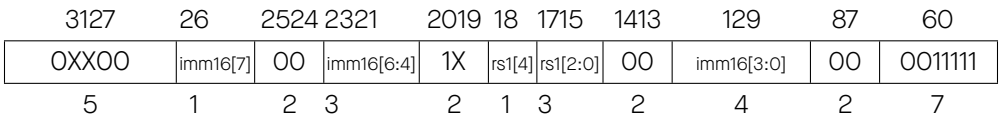


Figure 4.7-303. ESP.LD.QACC.L.L.128.IP

4.7.2.304 ESP.LD.UA.STATE.IP

Syntax

ESP.LD.UA.STATE.IP rs1,imm16

Description

This instruction loads 16-byte data from memory into the special register `UA_STATE[127:0]`. After the access is completed, the value in the register `rs1` is incremented by an 8-bit sign-extended constant in the instruction code segment, left-shifted by 4.
imm starts at -2048, ends at 2032, and steps by 16.

Operation

```
UA_STATE[127:  0] = load128(rs1[31:0])
rs1[31:0] = rs1[31:0] + imm
```

Schedule

Operands	Use	Def
UA_STATE	-	2
rs1	1	1

Format

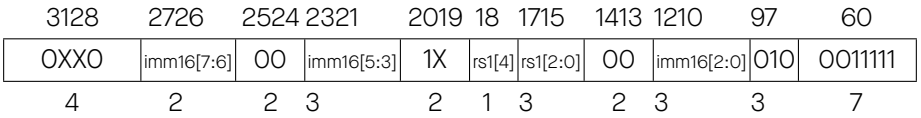


Figure 4.7-304. ESP.LD.UA.STATE.IP

4.7.2.305 ESP.LDXQ.32

Syntax

ESP.LDXQ.32 qu,qw,rs1,sel4,sel8

Description

This instruction selects one of the 8 segments of 16-bit data in `qw` as the addend according to the immediate number `sel8`. It performs the addition of the addend left-shifted by 2 bits to the value of the address register `rs1`, uses the result as the access address, and stores the loaded data in a 32-bit data segment in the register `qu` according to the value of the immediate number `sel4`.

Operation

```
  vaddr0[31:0] = rs1[31:0] + qw[ 15:  0] * 4
  vaddr1[31:0] = rs1[31:0] + qw[ 31: 16] * 4
  vaddr2[31:0] = rs1[31:0] + qw[ 47: 32] * 4
  vaddr3[31:0] = rs1[31:0] + qw[ 63: 47] * 4
  vaddr4[31:0] = rs1[31:0] + qw[ 79: 64] * 4
  vaddr5[31:0] = rs1[31:0] + qw[ 95: 80] * 4
  vaddr6[31:0] = rs1[31:0] + qw[111: 96] * 4
  vaddr7[31:0] = rs1[31:0] + qw[127:112] * 4

if sel8 == 0:
  dataIn[31:0] = load32(vaddr0[31:0])
if sel8 == 1:
  dataIn[31:0] = load32(vaddr1[31:0])
if sel8 == 2:
  dataIn[31:0] = load32(vaddr2[31:0])
if sel8 == 3:
  dataIn[31:0] = load32(vaddr3[31:0])
if sel8 == 4:
  dataIn[31:0] = load32(vaddr4[31:0])
if sel8 == 5:
  dataIn[31:0] = load32(vaddr5[31:0])
if sel8 == 6:
  dataIn[31:0] = load32(vaddr6[31:0])
if sel8 == 7:
  dataIn[31:0] = load32(vaddr7[31:0])

  qu[32*sel4+31:32*sel4] = dataIn[31:0]
```

Schedule

Operands	Use	Def
qu	-	2
qw	0	-
rs1	1	-

Format

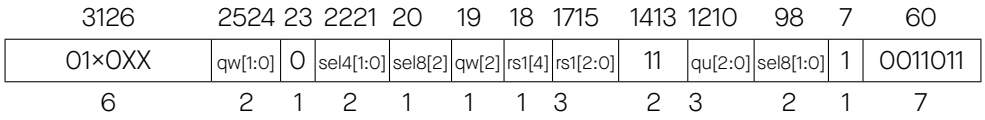


Figure 4.7-305. ESP.LDXQ.32

4.7.2.306 ESP.ST.QACC.H.H.128.IP

Syntax

ESP.ST.QACC.H.H.128.IP rs1,imm16

Description

This instruction stores the upper 128 bits in the special register `QACC_H` to memory. After the access is completed, the value in the register `rs1` is incremented by an 8-bit sign-extended constant in the instruction code segment, left-shifted by 4.
imm starts at -2048, ends at 2032, and steps by 16.

Operation

```
MEM(rs1[31:0]) = QACC_H[255:128]
rs1[31:0] = rs1[31:0] + imm
```

Schedule

Operands	Use	Def
QACC_H	2	-
rs1	1	1

Format

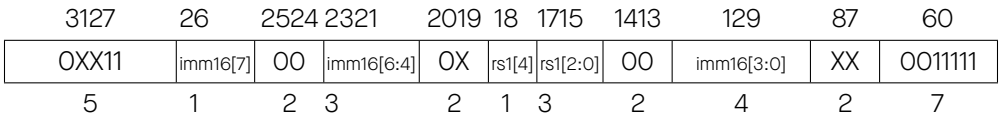


Figure 4.7-306. ESP.ST.QACC.H.H.128.IP

4.7.2.307 ESP.ST.QACC.H.L.128.IP

Syntax

ESP.ST.QACC.H.L.128.IP rs1,imm16

Description

This instruction stores the lower 128 bits in the special register `QACC_H` to memory. After the access is completed, the value in the register `rs1` is incremented by an 8-bit sign-extended constant in the instruction code segment, left-shifted by 4.
imm starts at -2048, ends at 2032, and steps by 16.

Operation

```
MEM(rs1[31:0]) = QACC_H[127:0]
rs1[31:0] = rs1[31:0] + imm
```

Schedule

Operands	Use	Def
QACC_H	2	-
rs1	1	1

Format

31–27	26	25–24	23–21	20–19	18	17–15	14–13	12–9	8–7	6–0
0XX11	imm16[7]	00	imm16[6:4]	1X	rs1[4]	rs1[2:0]	00	imm16[3:0]	00	0011111
5	1	2	3	2	1	3	2	4	2	7

Figure 4.7-307. ESP.ST.QACC.H.L.128.IP

4.7.2.308 ESP.ST.QACC.L.H.128.IP

Syntax

ESP.ST.QACC.L.H.128.IP rs1,imm16

Description

This instruction stores the upper 128 bits in the special register `QACC_L` to memory. After the access is completed, the value in the register `rs1` is incremented by an 8-bit sign-extended constant in the instruction code segment, left-shifted by 4.
imm starts at -2048, ends at 2032, and steps by 16.

Operation

```
MEM(rs1[31:0]) = QACC_L[255:128]
rs1[31:0] = rs1[31:0] + imm
```

Schedule

Operands	Use	Def
QACC_L	2	-
rs1	1	1

Format

31–27	26	25–24	23–21	20–19	18	17–15	14–13	12–9	8–7	6–0
0XX10	imm16[7]	00	imm16[6:4]	0X	rs1[4]	rs1[2:0]	00	imm16[3:0]	XX	0011111
5	1	2	3	2	1	3	2	4	2	7

Figure 4.7-308. ESP.ST.QACC.L.H.128.IP

4.7.2.309 ESP.ST.QACC.L.L.128.IP

Syntax

ESP.ST.QACC.L.L.128.IP rs1,imm16

Description

This instruction stores the lower 128 bits in the special register `QACC_L` to memory. After the access is completed, the value in the register `rs1` is incremented by an 8-bit sign-extended constant in the instruction code segment, left-shifted by 4.
imm starts at -2048, ends at 2032, and steps by 16.

Operation

```
MEM(rs1[31:0]) = QACC_L[127:0]
rs1[31:0] = rs1[31:0] + imm
```

Schedule

Operands	Use	Def
QACC_L	2	-
rs1	1	1

Format

31–27	26	25–24	23–21	20–19	18	17–15	14–13	12–9	8–7	6–0
0XX10	imm16[7]	00	imm16[6:4]	1X	rs1[4]	rs1[2:0]	00	imm16[3:0]	00	0011111
5	1	2	3	2	1	3	2	4	2	7

Figure 4.7-309. ESP.ST.QACC.L.L.128.IP

4.7.2.310 ESP.ST.UA.STATE.IP

Syntax

ESP.ST.UA.STATE.IP rs1,imm16

Description

This instruction stores the 128-bit data in the special register `UA_STATE` to memory. After the access is completed, the value in the register `rs1` is incremented by an 8-bit sign-extended constant in the instruction code segment, left-shifted by 4.
imm starts at -2048, ends at 2032, and steps by 16.

Operation

```
MEM(rs1[31:0]) = UA_STATE[127:0]
rs1[31:0] = rs1[31:0] + imm
```

Schedule

Operands	Use	Def
UA_STATE	2	-
rs1	1	1

Format

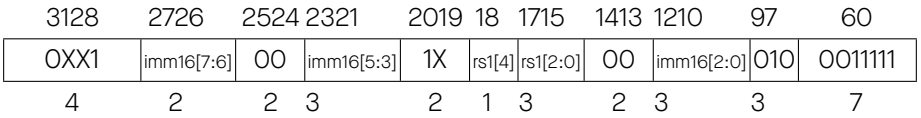


Figure 4.7-310. ESP.ST.UA.STATE.IP

4.7.2.311 ESP.STXQ.32

Syntax

ESP.STXQ.32 qu,qw,rs1,sel4,sel8

Description

This instruction selects one of the four 32-bit data segments in the register `qw` according to the immediate number `sel4`. Then, it selects an addend from the eight 16-bit data segments in the register `qw` according to the immediate number `sel8`, adds the addend left-shifted by 2 bits to the value of the address register `rs1`, and uses the result as the write address.

Operation

```
  vaddr0[31:0] = rs1[31:0] + qw[ 15:  0] * 4
  vaddr1[31:0] = rs1[31:0] + qw[ 31: 16] * 4
  vaddr2[31:0] = rs1[31:0] + qw[ 47: 32] * 4
  vaddr3[31:0] = rs1[31:0] + qw[ 63: 47] * 4
  vaddr4[31:0] = rs1[31:0] + qw[ 79: 64] * 4
  vaddr5[31:0] = rs1[31:0] + qw[ 95: 80] * 4
  vaddr6[31:0] = rs1[31:0] + qw[111: 96] * 4
  vaddr7[31:0] = rs1[31:0] + qw[127:112] * 4

if sel8 == 0:
  qu[32*sel4+31:32*sel4] =>store32(vaddr0[31:0])
if sel8 == 1:
  qu[32*sel4+31:32*sel4] =>store32(vaddr1[31:0])
if sel8 == 2:
  qu[32*sel4+31:32*sel4] =>store32(vaddr2[31:0])
if sel8 == 3:
  qu[32*sel4+31:32*sel4] =>store32(vaddr3[31:0])
if sel8 == 4:
  qu[32*sel4+31:32*sel4] =>store32(vaddr4[31:0])
if sel8 == 5:
  qu[32*sel4+31:32*sel4] =>store32(vaddr5[31:0])
if sel8 == 6:
  qu[32*sel4+31:32*sel4] =>store32(vaddr6[31:0])
if sel8 == 7:
  qu[32*sel4+31:32*sel4] =>store32(vaddr7[31:0])
```

Schedule

Operands	Use	Def
qu	2	-
qw	0	-
rs1	1	1

Format

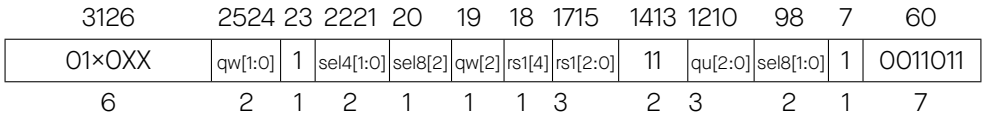


Figure 4.7-311. ESP.STXQ.32

4.7.2.312 ESP.VLD.128.IP

Syntax

ESP.VLD.128.IP qu,rs1,imm16

Description

This instruction loads 128-bit data from memory to the register qu, with the pointer incremented by an immediate.

imm starts at -2048, ends at 2032, and steps by 16.

Operation

```
qu[127:  0] = load128(rs1[31:0])
rs1[31:0] = rs1[31:0] + imm
```

Schedule

Operands	Use	Def
qu	-	2
rs1	1	1

Format

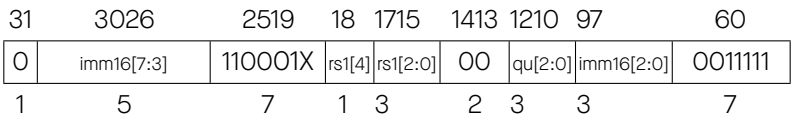


Figure 4.7-312. ESP.VLD.128.IP

4.7.2.313 ESP.VLD.128.XP

Syntax

ESP.VLD.128.XP qu,rs1,rs2

Description

This instruction loads 128-bit data from memory into the register `qu`, with the pointer incremented by the value of the register `rs2`.

Operation

```
qu[127:  0] = load128(rs1[31:0])
rs1[31:0] = rs1[31:0] + rs2[31:0]
```

Schedule

Operands	Use	Def
qu	-	2
rs1	1	1
rs2	1	-

Format

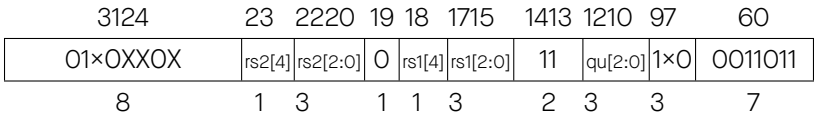


Figure 4.7-313. ESP.VLD.128.XP

4.7.2.314 ESP.VLD.H.64.IP

Syntax

ESP.VLD.H.64.IP qu,rs1,imm8

Description

This instruction loads 64-bit data from memory to the upper 64-bit segment in the register qu, with the pointer incremented by an immediate.
imm starts at -1024, ends at 1016, and steps by 8.

Operation

```
qu[ 127:  64] = load64(rs1[31:0])
rs1[31:0] = rs1[31:0] + imm
```

Schedule

Operands	Use	Def
qu	-	2
rs1	1	1

Format

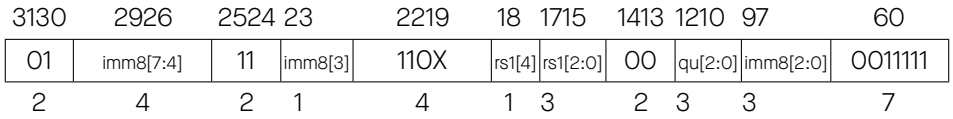


Figure 4.7-314. ESP.VLD.H.64.IP

4.7.2.315 ESP.VLD.H.64.XP

Syntax

ESP.VLD.H.64.XP qu,rs1,rs2

Description

This instruction loads 64-bit data from memory into the upper 64-bit segment of the register `qu`, with the pointer incremented by the value in the register `rs2`.

Operation

```
qu[ 127:  64] = load64(rs1[31:0])
rs1[31:0] = rs1[31:0] + rs2[31:0]
```

Schedule

Operands	Use	Def
qu	-	2
rs1	1	1
rs2	1	-

Format

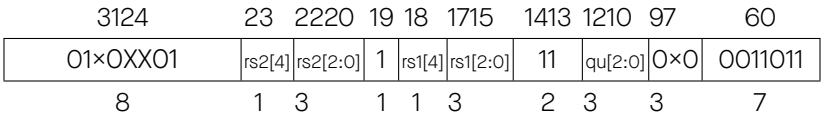


Figure 4.7-315. ESP.VLD.H.64.XP

4.7.2.316 ESP.VLD.L.64.IP

Syntax

ESP.VLD.L.64.IP qu,rs1,imm8

Description

This instruction loads 64-bit data from memory into the lower 64-bit segment of the register qu, with the pointer incremented by an immediate.
imm starts at -1024, ends at 1016, and steps by 8.

Operation

```
qu[ 63:  0] = load64(rs1[31:0])
rs1[31:0] = rs1[31:0] + imm
```

Schedule

Operands	Use	Def
qu	-	2
rs1	1	1

Format

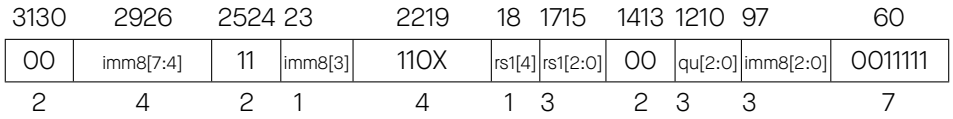


Figure 4.7-316. ESP.VLD.L.64.IP

4.7.2.317 ESP.VLD.L.64.XP

Syntax

ESP.VLD.L.64.XP qu,rs1,rs2

Description

This instruction loads 64-bit data from memory into the lower 64-bit segment of the `qu` register, with the pointer incremented by the value in the `rs2` register.

Operation

```
qu[ 63:  0] = load64(rs1[31:0])
rs1[31:0] = rs1[31:0] + rs2[31:0]
```

Schedule

Operands	Use	Def
qu	-	2
rs1	1	1
rs2	1	-

Format

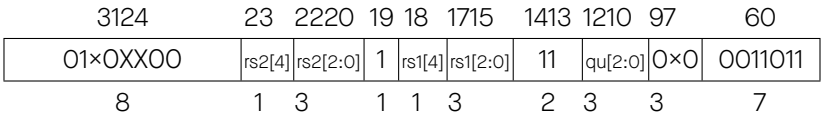


Figure 4.7-317. ESP.VLD.L.64.XP

4.7.2.318 ESP.VST.128.IP

Syntax

ESP.VST.128.IP qu,rs1,imm16

Description

This instruction stores the 128 bits in the register qu to memory, with the pointer incremented by an immediate.

imm starts at -2048, ends at 2032, and steps in increments of 16.

Operation

```
MEM(rs1[31:0]) = qu[127:  0]
rs1[31:0] = rs1[31:0] +imm
```

Schedule

Operands	Use	Def
qu	2	-
rs1	1	1

Format

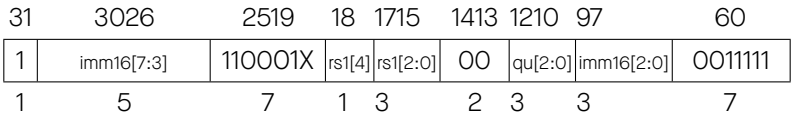


Figure 4.7-318. ESP.VST.128.IP

4.7.2.319 ESP.VST.128.XP

Syntax

ESP.VST.128.XP qu,rs1,rs2

Description

This instruction stores the 128 bits in the register `qu` to memory, with the pointer incremented by the value in the register `rs2`.

Operation

```
MEM(rs1[31:0]) = qu[127:  0]
rs1[31:0] = rs1[31:0] + rs2[31:0]
```

Schedule

Operands	Use	Def
qu	2	-
rs1	1	1

Format

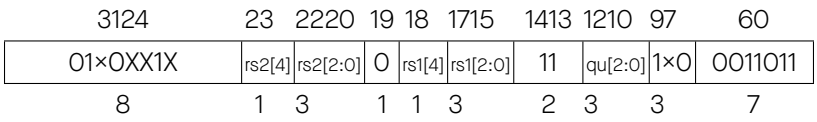


Figure 4.7-319. ESP.VST.128.XP

4.7.2.320 ESP.VST.H.64.IP

Syntax

ESP.VST.H.64.IP qu,rs1,imm8

Description

This instruction stores the 64 bits in the upper 64 bits of the register qu to memory, with the pointer incremented by an immediate.
imm starts at -1024, ends at 1016, and steps by 8.

Operation

```
MEM(rs1[31:0]) = qu[127:  64]
rs1[31:0] = rs1[31:0] + imm
```

Schedule

Operands	Use	Def
qu	2	-
rs1	1	1

Format

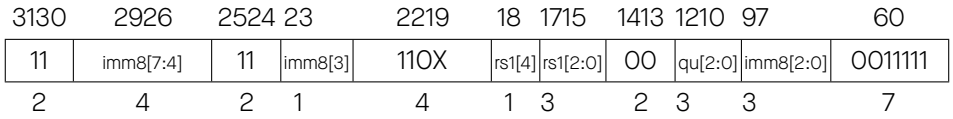


Figure 4.7-320. ESP.VST.H.64.IP

4.7.2.321 ESP.VST.H.64.XP

Syntax

ESP.VST.H.64.XP qu,rs1,rs2

Description

This instruction stores the 64 bits in the upper 64 bits of the register `qu` to memory, with the pointer incremented by the value in the register `rs2`.

Operation

```
MEM(rs1[31:0]) = qu[127:  64]
rs1[31:0] = rs1[31:0] + rs2[31:0]
```

Schedule

Operands	Use	Def
qu	2	-
rs1	1	1
rs2	1	-

Format

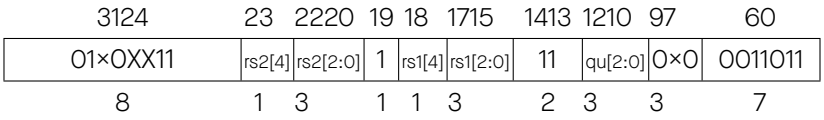


Figure 4.7-321. ESP.VST.H.64.XP

4.7.2.322 ESP.VST.L.64.IP

Syntax

ESP.VST.L.64.IP qu,rs1,imm8

Description

This instruction stores the 64 bits in the lower 64 bits of the register qu to memory, with the pointer incremented by an immediate.
imm starts at -1024, ends at 1016, and steps by 8.

Operation

```
MEM(rs1[31:0]) = qu[63:  0]
rs1[31:0] = rs1[31:0] + imm
```

Schedule

Operands	Use	Def
qu	2	-
rs1	1	1

Format

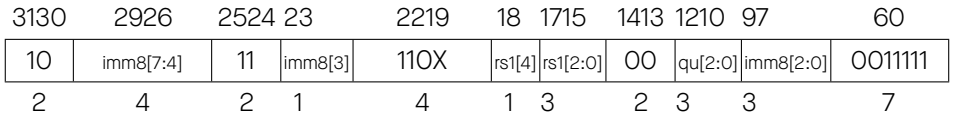


Figure 4.7-322. ESP.VST.L.64.IP

4.7.2.323 ESP.VST.L.64.XP

Syntax

ESP.VST.L.64.XP qu,rs1,rs2

Description

This instruction stores the 64 bits in the lower 64 bits of the register `qu` to memory, with the pointer incremented by the value in the register `rs2`.

Operation

```
MEM(rs1[31:0]) = qu[63:  0]
rs1[31:0] = rs1[31:0] + rs2[31:0]
```

Schedule

Operands	Use	Def
qu	2	-
rs1	1	1
rs2	1	-

Format

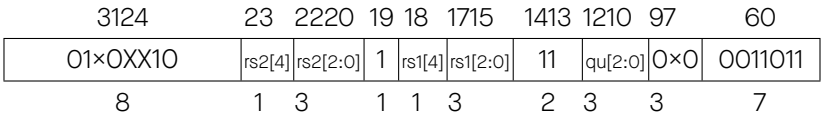


Figure 4.7-323. ESP.VST.L.64.XP

4.7.2.324 ESP.SLCI.2Q

Syntax

ESP.SLCI.2Q qy,qw,sel16

Description

This instruction performs a left shift on the 32-byte concatenation of registers `qw` and `qy`, and pads the lower bits with 0. The upper 128 bits of the shift result are written to register `qy`, and the lower 128 bits are written to `qw`. The left shift amount is 8 times the sum of `sel16` and 1.

Operation

```
{qy[127:  0], qw[127:  0]} = {qy[127:  0], qw[127:  0]} << ((sel16[3:0]+1)*8)
```

Schedule

Operands	Use	Def
qy	1	1
qw	1	1

Format

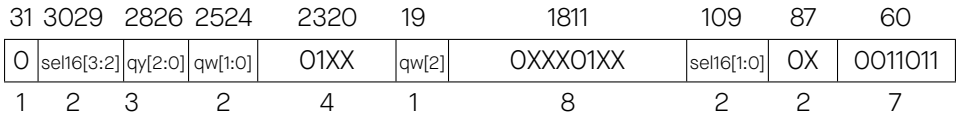


Figure 4.7-324. ESP.SLCI.2Q

4.7.2.325 ESP.SLCXXP.2Q

Syntax

ESP.SLCXXP.2Q qy,qw,rs1,rs2

Description

This instruction performs a left shift on the 32-byte concatenation of registers `qy` and `qw`, and pads the lower bits with 0. The upper 128 bits of the shift result are written to register `qy`, and the lower 128 bits are written to `qw`. The left shift amount is 8, performed by the sum of 1 plus the lower 4-bit value of register `rs1`. After the above operations, the value in `rs1` is incremented by the value in `rs2`.

Operation

```
{qy[127:  0], qw[127:  0]} = {qy[127:  0], qw[127:  0]} << ((rs1[3:0]+1)*8)
rs1[31:0] = rs1[31:0] + rs2[31:0]
```

Schedule

Operands	Use	Def
qy	1	1
qw	1	1
rs1	1	1
rs2	1	-

Format

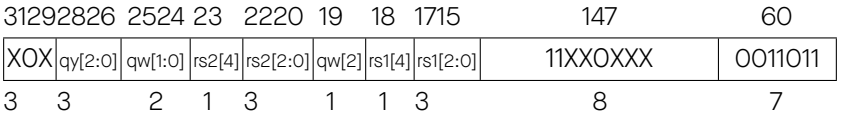


Figure 4.7-325. ESP.SLCXXP.2Q

4.7.2.326 ESP.SRC.Q

Syntax

ESP.SRC.Q qz,qw,qy

Description

This instruction performs an arithmetic right shift on the 32-byte concatenation of registers qy and qw, which hold the loaded data from two consecutive aligned addresses. In this way, you can obtain unaligned 16-byte data, which will then be written to the register qw. The right shift amount is SAR_BYTE performed by 8.

Operation

```
qz[127:0] = {qy[127: 0], qw[127: 0]} >> {SAR_BYTE[3:0] << 3}
```

Schedule

Operands	Use	Def
qz	-	1
qw	1	-
qy	1	-
SAR_BYTES	1	-

Format

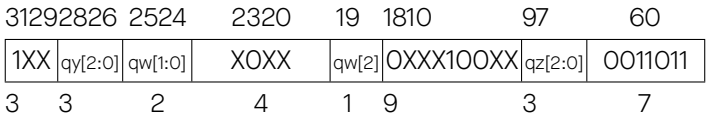


Figure 4.7-326. ESP.SRC.Q

4.7.2.327 ESP.SRC.Q.LD.IP

Syntax

ESP.SRC.Q.LD.IP qu,rs1,imm16,qw,qy

Description

This instruction performs an arithmetic right shift on the 32-byte concatenation of registers `qy` and `qw`, which hold the loaded data from two consecutive aligned addresses. In this way, you can obtain unaligned 16-byte data, which will be written to the register `qw`. The right shift amount is `SAR_BYTE` performed by 8. Simultaneously, the 16-byte data is loaded from the memory into the register `qu`. After the access is completed, the value in the register `rs1` is incremented by `imm`.
`imm` starts at -2048, ends at 2032, with a step of 16.

Operation

```
qw[127:0] = {qy[127: 0], qw[127: 0]} >> {SAR_BYTE[3:0] << 3}
qu[127:0] = load128(rs1[31:0])
rs1[31:0] = rs1[31:0] + imm
```

Schedule

Operands	Use	Def
qz	-	1
qw	1	-
qy	1	-
SAR_BYTES	1	-
qu	-	2
rs1	1	1

Format

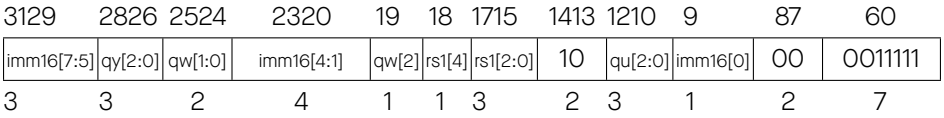


Figure 4.7-327. ESP.SRC.Q.LD.IP

4.7.2.328 ESP.SRC.Q.LD.XP

Syntax

ESP.SRC.Q.LD.XP qu,rs1,rs2,qw,qy

Description

This instruction performs an arithmetic right shift on the 32-byte concatenation of registers `qy` and `qw`, which hold the loaded data from two consecutive aligned addresses. In this way, you can obtain unaligned 16-byte data, which will be written to the register `qw`. The right shift amount is the product of `SAR_BYTE` and 8. Simultaneously, the 16-byte data is loaded from memory into the register `qu`. After the access is completed, the value in the register `rs1` is incremented by the value in the register `rs2`.

Operation

```
qw[127:0] = {qy[127: 0], qw[127: 0]} >> {SAR_BYTE[3:0] << 3}
qu[127:0] = load128(rs1[31:0])
rs1[31:0] = rs1[31:0] + rs2[31:0]
```

Schedule

Operands	Use	Def
qz	-	1
qw	1	-
qy	1	-
SAR_BYTES	1	-
qu	-	2
rs1	1	1
rs2	1	-

Format

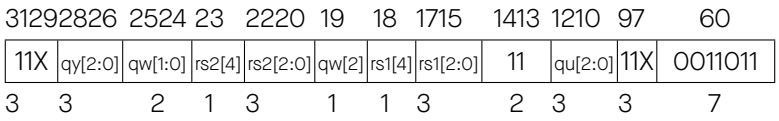


Figure 4.7-328. ESP.SRC.Q.LD.XP

4.7.2.329 ESP.SRCI.2Q

Syntax

ESP.SRCI.2Q qy,qw,sel16

Description

This instruction performs a logical right shift on the 32-byte concatenation of registers `qw` and `qy`, and pads the higher bits with 0. The upper 128 bits of the shift result are written to the register `qy`, and the lower 128 bits are written to `qw`. The right shift amount is 8 times the sum of `sel16` and 1.

Operation

```
{qy[127:  0], qw[127:  0]} = {qy[127:  0], qw[127:  0]} >> ((sel16+1)*8)
qy[127:127-8*sel16] = 0
```

Schedule

Operands	Use	Def
qz	1	1
qw	1	1

Format

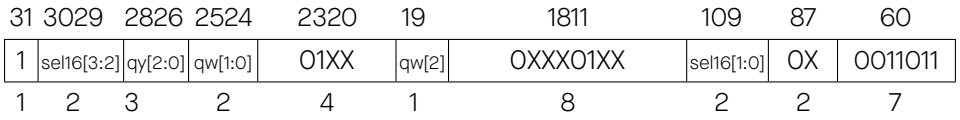


Figure 4.7-329. ESP.SRCI.2Q

4.7.2.330 ESP.SRCMB.S16.Q.QACC

Syntax

ESP.SRCMB.S16.Q.QACC qu,qw,sat,rm

Description

This instruction extracts 8 data segments of 64 bits from special registers QACC_H and QACC_L and performs arithmetic right shift operations respectively, the shift amounts are from qw. While writing the shift results back to QACC_H and QACC_L, the instruction saturates the results to 16-bit signed numbers and writes the 8 16-bit data obtained after saturation into the register qu.

Operation

```
shf_value0[5:0] = (qw[ 15: 0] >= 0x3f) ? 0x3f : qw[ 15: 0]
shf_value1[5:0] = (qw[ 31: 16] >= 0x3f) ? 0x3f : qw[ 31: 16]
shf_value2[5:0] = (qw[ 47: 32] >= 0x3f) ? 0x3f : qw[ 47: 32]
shf_value3[5:0] = (qw[ 63: 48] >= 0x3f) ? 0x3f : qw[ 63: 48]
shf_value4[5:0] = (qw[ 79: 64] >= 0x3f) ? 0x3f : qw[ 79: 64]
shf_value5[5:0] = (qw[ 95: 80] >= 0x3f) ? 0x3f : qw[ 95: 80]
shf_value6[5:0] = (qw[111: 96] >= 0x3f) ? 0x3f : qw[111: 96]
shf_value7[5:0] = (qw[127:112] >= 0x3f) ? 0x3f : qw[127:112]

DATA_L[ 63:  0] = QACC_L[ 63:  0] >> shf_value0[5:0]
DATA_L[127: 64] = QACC_L[127: 64] >> shf_value1[5:0]
DATA_L[191:128] = QACC_L[191:128] >> shf_value2[5:0]
DATA_L[255:192] = QACC_L[255:192] >> shf_value3[5:0]
DATA_H[ 63:  0] = QACC_H[ 63:  0] >> shf_value4[5:0]
DATA_H[127: 64] = QACC_H[127: 64] >> shf_value5[5:0]
DATA_H[191:128] = QACC_H[191:128] >> shf_value6[5:0]
DATA_H[255:192] = QACC_H[255:192] >> shf_value7[5:0]

QACC_L[255:0] = DATA_L[255:0]
QACC_H[255:0] = DATA_H[255:0]

if sel2 == 0:
  qu[15: 0] = DATA_L[15:0]
  qu[31:16] = DATA_L[79:64]
  qu[47:32] = DATA_L[143:128]
  qu[63:48] = DATA_L[207:192]
  qu[79:64] = DATA_H[15:0]
  qu[95:80] = DATA_H[79:64]
  qu[111:96] = DATA_H[143:128]
  qu[127:112] = DATA_H[207:192]
else if sel2 == 1:
  qu[ 15:  0] = min(max(DATA_L[63:0], -2^{15}), 2^{15}-1)
  qu[ 31: 16] = min(max(DATA_L[127:64], -2^{15}), 2^{15}-1)
  qu[ 47: 32] = min(max(DATA_L[191:128], -2^{15}), 2^{15}-1)
  qu[ 63: 48] = min(max(DATA_L[255:192], -2^{15}), 2^{15}-1)
  qu[ 79: 64] = min(max(DATA_H[63:0], -2^{15}), 2^{15}-1)
  qu[ 95: 80] = min(max(DATA_H[127:64], -2^{15}), 2^{15}-1)
  qu[111: 96] = min(max(DATA_H[191:128], -2^{15}), 2^{15}-1)
  qu[127:112] = min(max(DATA_H[255:192], -2^{15}), 2^{15}-1)
```

Schedule

Operands	Use	Def
qu	-	2
rs1	1	-
QACC_H	1	2
QACC_L	1	2

Format

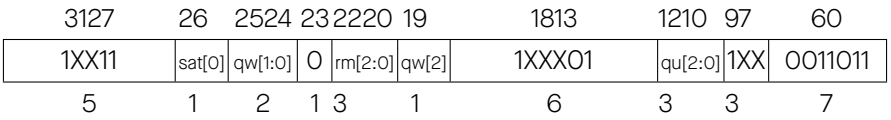


Figure 4.7-330. ESP.SRCMB.S16.Q.QACC

4.7.2.331 ESP.SRCMB.S16.QACC

Syntax

ESP.SRCMB.S16.QACC qu,rs1,sat,rm

Description

This instruction extracts 8 data segments of 64 bits from special registers `QACC_H` and `QACC_L` and performs arithmetic right shift operations respectively. While writing the shift results back to `QACC_H` and `QACC_L`, the instruction saturates the results to 16-bit signed numbers and writes the 8 16-bit data segments obtained after saturation into the `qu` register.

Operation

```
temp0[63:0] = QACC_L[ 63:  0]
...
temp3[63:0] = QACC_L[255:192]
temp4[63:0] = QACC_H[ 63:  0]
...
temp7[63:0] = QACC_H[255:192]

temp_shf0[63:0] = temp0[63:0] >> rs1[5:0]
temp_shf1[63:0] = temp1[63:0] >> rs1[5:0]
...
temp_shf7[63:0] = temp7[63:0] >> rs1[5:0]

QACC_L[ 63:  0] = temp_shf0[63:0]
...
QACC_L[255:192] = temp_shf3[63:0]
QACC_H[ 63:  0] = temp_shf4[63:0]
...
QACC_H[255:192] = temp_shf7[63:0]

if sel2 == 0:
  qu[ 15:  0] = temp0[15:0]
  qu[ 31: 16] = temp1[15:0]
  qu[ 47: 32] = temp2[15:0]
  qu[ 63: 48] = temp3[15:0]
  qu[ 79: 64] = temp4[15:0]
  qu[ 95: 80] = temp5[15:0]
  qu[111: 96] = temp6[15:0]
  qu[127:112] = temp7[15:0]
else if sel2==1:
  qu[ 15:  0] = min(max(temp_shf0[63:0], -2^{15}), 2^{15}-1)
  ...
  qu[ 63: 48] = min(max(temp_shf3[63:0], -2^{15}), 2^{15}-1)
  qu[ 79: 64] = min(max(temp_shf4[63:0], -2^{15}), 2^{15}-1)
  ...
  qu[127:112] = min(max(temp_shf7[63:0], -2^{15}), 2^{15}-1)
```

Schedule

Operands	Use	Def
qu	-	2
rs1	1	-
QACC_H	1	2
QACC_L	1	2

Format

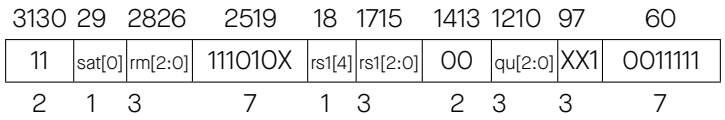


Figure 4.7-331. ESP.SRCMB.S16.QACC

4.7.2.332 ESP.SRCMB.S8.Q.QACC

Syntax

ESP.SRCMB.S8.Q.QACC qu,qw,sat,rm

Description

This instruction extracts 16 data segments of 32 bits from special registers QACC_H and QACC_L and performs arithmetic right shift operations respectively, the shift amounts are from qw. While writing the shift results back to QACC_H and QACC_L, the instruction saturates the results to 8-bit signed numbers and writes the 16 8-bit data obtained after saturation into the register qu.

Operation

```
shf_value0[5:0] = (qw[ 7: 0] >= 0x1f) ? 0x1f : qw[ 7: 0]
shf_value1[5:0] = (qw[ 15: 8] >= 0x1f) ? 0x1f : qw[ 15: 8]
shf_value2[5:0] = (qw[ 23: 16] >= 0x1f) ? 0x1f : qw[ 23: 16]
shf_value3[5:0] = (qw[ 31: 24] >= 0x1f) ? 0x1f : qw[ 31: 24]
shf_value4[5:0] = (qw[ 39: 32] >= 0x1f) ? 0x1f : qw[ 39: 32]
shf_value5[5:0] = (qw[ 47: 40] >= 0x1f) ? 0x1f : qw[ 47: 40]
shf_value6[5:0] = (qw[ 55: 48] >= 0x1f) ? 0x1f : qw[ 55: 48]
shf_value7[5:0] = (qw[ 63: 56] >= 0x1f) ? 0x1f : qw[ 63: 56]
shf_value8[5:0] = (qw[ 71: 64] >= 0x1f) ? 0x1f : qw[ 71: 64]
shf_value9[5:0] = (qw[ 79: 72] >= 0x1f) ? 0x1f : qw[ 79: 72]
shf_value10[5:0] = (qw[ 87: 80] >= 0x1f) ? 0x1f : qw[ 87: 80]
shf_value11[5:0] = (qw[ 95: 88] >= 0x1f) ? 0x1f : qw[ 95: 88]
shf_value12[5:0] = (qw[103: 96] >= 0x1f) ? 0x1f : qw[103: 96]
shf_value13[5:0] = (qw[111:104] >= 0x1f) ? 0x1f : qw[111:104]
shf_value14[5:0] = (qw[119:112] >= 0x1f) ? 0x1f : qw[119:112]
shf_value15[5:0] = (qw[127:120] >= 0x1f) ? 0x1f : qw[127:120]

DATA_L[ 31:  0] = QACC_L[ 31:  0] >> shf_value0[4:0]
DATA_L[ 63: 32] = QACC_L[ 63: 32] >> shf_value1[4:0]
DATA_L[ 95: 64] = QACC_L[ 95: 64] >> shf_value2[4:0]
DATA_L[127: 96] = QACC_L[127: 96] >> shf_value3[4:0]
DATA_L[159:128] = QACC_L[159:128] >> shf_value4[4:0]
DATA_L[191:160] = QACC_L[191:160] >> shf_value5[4:0]
DATA_L[223:192] = QACC_L[223:192] >> shf_value6[4:0]
DATA_L[255:224] = QACC_L[255:224] >> shf_value7[4:0]
DATA_H[ 31:  0] = QACC_H[ 31:  0] >> shf_value8[4:0]
DATA_H[ 63: 32] = QACC_H[ 63: 32] >> shf_value9[4:0]
DATA_H[ 95: 64] = QACC_H[ 95: 64] >> shf_value10[4:0]
DATA_H[127: 96] = QACC_H[127: 96] >> shf_value11[4:0]
DATA_H[159:128] = QACC_H[159:128] >> shf_value12[4:0]
DATA_H[191:160] = QACC_H[191:160] >> shf_value13[4:0]
DATA_H[223:192] = QACC_H[223:192] >> shf_value14[4:0]
DATA_H[255:224] = QACC_H[255:224] >> shf_value15[4:0]


QACC_L[255:0] = DATA_L[255:0]
QACC_H[255:0] = DATA_H[255:0]

if sel2 == 0:
```

```
    qu[  7:  0] = DATA_L[  7:  0]
    qu[ 15:  8] = DATA_L[ 39: 32]
    qu[ 23: 16] = DATA_L[ 71: 64]
    qu[ 31: 24] = DATA_L[103: 96]
    qu[ 39: 32] = DATA_L[135:128]
    qu[ 47: 40] = DATA_L[167:160]
    qu[ 55: 48] = DATA_L[199:192]
    qu[ 63: 56] = DATA_L[231:224]
    qu[ 71: 64] = DATA_H[  7:  0]
    qu[ 79: 72] = DATA_H[ 39: 32]
    qu[ 87: 80] = DATA_H[ 71: 64]
    qu[ 95: 88] = DATA_H[103: 96]
    qu[103: 96] = DATA_H[135:128]
    qu[111:104] = DATA_H[167:160]
    qu[119:112] = DATA_H[199:192]
    qu[127:120] = DATA_H[231:224]
  else if sel2 == 1:
    qu[  7:  0] = min(max(DATA_L[ 31:  0], -2^{7}), 2^{7}-1)
    qu[ 15:  8] = min(max(DATA_L[ 63: 32], -2^{7}), 2^{7}-1)
    qu[ 23: 16] = min(max(DATA_L[ 95: 64], -2^{7}), 2^{7}-1)
    qu[ 31: 24] = min(max(DATA_L[127: 96], -2^{7}), 2^{7}-1)
    qu[ 39: 32] = min(max(DATA_L[159:128], -2^{7}), 2^{7}-1)
    qu[ 47: 40] = min(max(DATA_L[191:160], -2^{7}), 2^{7}-1)
    qu[ 55: 48] = min(max(DATA_L[223:192], -2^{7}), 2^{7}-1)
    qu[ 63: 56] = min(max(DATA_L[255:224], -2^{7}), 2^{7}-1)
    qu[ 71: 64] = min(max(DATA_L[ 31:  0], -2^{7}), 2^{7}-1)
    qu[ 79: 72] = min(max(DATA_H[ 63: 32], -2^{7}), 2^{7}-1)
    qu[ 87: 80] = min(max(DATA_H[ 95: 64], -2^{7}), 2^{7}-1)
    qu[ 95: 88] = min(max(DATA_H[127: 96], -2^{7}), 2^{7}-1)
    qu[103: 96] = min(max(DATA_H[159:128], -2^{7}), 2^{7}-1)
    qu[111:104] = min(max(DATA_H[191:160], -2^{7}), 2^{7}-1)
    qu[119:112] = min(max(DATA_H[223:192], -2^{7}), 2^{7}-1)
    qu[127:120] = min(max(DATA_H[255:224], -2^{7}), 2^{7}-1)
```

Schedule

Operands	Use	Def
qu	-	2
rs1	1	-
QACC_H	1	2
QACC_L	1	2

Format

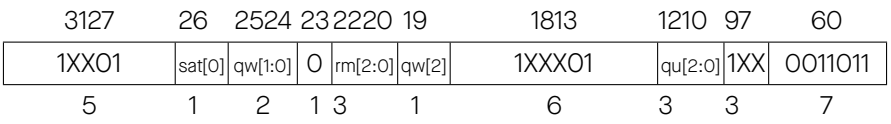


Figure 4.7-332. ESP.SRCMB.S8.Q.QACC

4.7.2.333 ESP.SRCMB.S8.QACC

Syntax

ESP.SRCMB.S8.QACC qu,rs1,sat,rm

Description

This instruction extracts 16 data segments of 32 bits from special registers `QACC_H` and `QACC_L` and performs arithmetic right shift operations respectively. While writing the shift result back to `QACC_H` and `QACC_L`, the instruction saturates the result to 8-bit signed numbers and writes the 16 8-bit data obtained after saturation into the register `qu`.

Operation

```
DATA_L[ 31:  0] = QACC_L[ 31:  0] >> rs1[4:0]
DATA_L[ 63: 32] = QACC_L[ 63: 32] >> rs1[4:0]
DATA_L[ 95: 64] = QACC_L[ 95: 64] >> rs1[4:0]
DATA_L[127: 96] = QACC_L[127: 96] >> rs1[4:0]
DATA_L[159:128] = QACC_L[159:128] >> rs1[4:0]
DATA_L[191:160] = QACC_L[191:160] >> rs1[4:0]
DATA_L[223:192] = QACC_L[223:192] >> rs1[4:0]
DATA_L[255:224] = QACC_L[255:224] >> rs1[4:0]
DATA_H[ 31:  0] = QACC_H[ 31:  0] >> rs1[4:0]
DATA_H[ 63: 32] = QACC_H[ 63: 32] >> rs1[4:0]
DATA_H[ 95: 64] = QACC_H[ 95: 64] >> rs1[4:0]
DATA_H[127: 96] = QACC_H[127: 96] >> rs1[4:0]
DATA_H[159:128] = QACC_H[159:128] >> rs1[4:0]
DATA_H[191:160] = QACC_H[191:160] >> rs1[4:0]
DATA_H[223:192] = QACC_H[223:192] >> rs1[4:0]
DATA_H[255:224] = QACC_H[255:224] >> rs1[4:0]

QACC_L[255:0] = DATA_L[255:0]
QACC_H[255:0] = DATA_H[255:0]

if sel2 == 0:
  qu[  7:  0] = DATA_L[  7:  0]
  qu[ 15:  8] = DATA_L[ 39: 32]
  qu[ 23: 16] = DATA_L[ 71: 64]
  qu[ 31: 24] = DATA_L[103: 96]
  qu[ 39: 32] = DATA_L[135:128]
  qu[ 47: 40] = DATA_L[167:160]
  qu[ 55: 48] = DATA_L[199:192]
  qu[ 63: 56] = DATA_L[231:224]
  qu[ 71: 64] = DATA_H[  7:  0]
  qu[ 79: 72] = DATA_H[ 39: 32]
  qu[ 87: 80] = DATA_H[ 71: 64]
  qu[ 95: 88] = DATA_H[103: 96]
  qu[103: 96] = DATA_H[135:128]
  qu[111:104] = DATA_H[167:160]
  qu[119:112] = DATA_H[199:192]
  qu[127:120] = DATA_H[231:224]
else if sel2 == 1:
```

```
    qu[  7:  0] = min(max(DATA_L[ 31:  0], -2^{7}), 2^{7}-1)
    qu[ 15:  8] = min(max(DATA_L[ 63: 32], -2^{7}), 2^{7}-1)
    qu[ 23: 16] = min(max(DATA_L[ 95: 64], -2^{7}), 2^{7}-1)
    qu[ 31: 24] = min(max(DATA_L[127: 96], -2^{7}), 2^{7}-1)
    qu[ 39: 32] = min(max(DATA_L[159:128], -2^{7}), 2^{7}-1)
    qu[ 47: 40] = min(max(DATA_L[191:160], -2^{7}), 2^{7}-1)
    qu[ 55: 48] = min(max(DATA_L[223:192], -2^{7}), 2^{7}-1)
    qu[ 63: 56] = min(max(DATA_L[255:224], -2^{7}), 2^{7}-1)
    qu[ 71: 64] = min(max(DATA_L[ 31:  0], -2^{7}), 2^{7}-1)
    qu[ 79: 72] = min(max(DATA_H[ 63: 32], -2^{7}), 2^{7}-1)
    qu[ 87: 80] = min(max(DATA_H[ 95: 64], -2^{7}), 2^{7}-1)
    qu[ 95: 88] = min(max(DATA_H[127: 96], -2^{7}), 2^{7}-1)
    qu[103: 96] = min(max(DATA_H[159:128], -2^{7}), 2^{7}-1)
    qu[111:104] = min(max(DATA_H[191:160], -2^{7}), 2^{7}-1)
    qu[119:112] = min(max(DATA_H[223:192], -2^{7}), 2^{7}-1)
    qu[127:120] = min(max(DATA_H[255:224], -2^{7}), 2^{7}-1)
```

Schedule

Operands	Use	Def
qu	-	2
rs1	1	-
QACC_H	1	2
QACC_L	1	2

Format

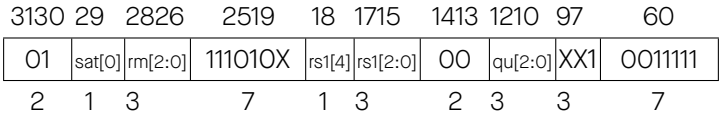


Figure 4.7-333. ESP.SRCMB.S8.QACC

4.7.2.334 ESP.SRCMB.U16.Q.QACC

Syntax

ESP.SRCMB.U16.Q.QACC qu,qw,sat,rm

Description

This instruction extracts 8 data segments of 64 bits from special registers QACC_H and QACC_L and performs arithmetic right shift operations respectively, the shift amounts are from qw. While writing the shift results back to QACC_H and QACC_L, the instruction saturates the results to 16-bit unsigned numbers and writes the 8 16-bit data obtained after saturation into the register qu.

Operation

```
shf_value0[5:0] = (qw[ 15: 0] >= 0x3f) ? 0x3f : qw[ 15: 0]
shf_value1[5:0] = (qw[ 31: 16] >= 0x3f) ? 0x3f : qw[ 31: 16]
shf_value2[5:0] = (qw[ 47: 32] >= 0x3f) ? 0x3f : qw[ 47: 32]
shf_value3[5:0] = (qw[ 63: 48] >= 0x3f) ? 0x3f : qw[ 63: 48]
shf_value4[5:0] = (qw[ 79: 64] >= 0x3f) ? 0x3f : qw[ 79: 64]
shf_value5[5:0] = (qw[ 95: 80] >= 0x3f) ? 0x3f : qw[ 95: 80]
shf_value6[5:0] = (qw[111: 96] >= 0x3f) ? 0x3f : qw[111: 96]
shf_value7[5:0] = (qw[127:112] >= 0x3f) ? 0x3f : qw[127:112]

DATA_L[ 63:  0] = QACC_L[ 63:  0] >> shf_value0[5:0]
DATA_L[127: 64] = QACC_L[127: 64] >> shf_value1[5:0]
DATA_L[191:128] = QACC_L[191:128] >> shf_value2[5:0]
DATA_L[255:192] = QACC_L[255:192] >> shf_value3[5:0]
DATA_H[ 63:  0] = QACC_H[ 63:  0] >> shf_value4[5:0]
DATA_H[127: 64] = QACC_H[127: 64] >> shf_value5[5:0]
DATA_H[191:128] = QACC_H[191:128] >> shf_value6[5:0]
DATA_H[255:192] = QACC_H[255:192] >> shf_value7[5:0]

QACC_L[255:0] = DATA_L[255:0]
QACC_H[255:0] = DATA_H[255:0]

if sel2 == 0:
  qu[15: 0] = DATA_L[15:0]
  qu[31:16] = DATA_L[79:64]
  qu[47:32] = DATA_L[143:128]
  qu[63:48] = DATA_L[207:192]
  qu[79:64] = DATA_H[15:0]
  qu[95:80] = DATA_H[79:64]
  qu[111:96] = DATA_H[143:128]
  qu[127:112] = DATA_H[207:192]
else if sel2 == 1:
  qu[ 15:  0] = min(DATA_L[ 63:  0], 2^{16}-1)
  qu[ 31: 16] = min(DATA_L[127: 64], 2^{16}-1)
  qu[ 47: 32] = min(DATA_L[191:128], 2^{16}-1)
  qu[ 63: 48] = min(DATA_L[255:192], 2^{16}-1)
  qu[ 79: 64] = min(DATA_H[ 63:  0], 2^{16}-1)
  qu[ 95: 80] = min(DATA_H[127: 64], 2^{16}-1)
  qu[111: 96] = min(DATA_H[191:128], 2^{16}-1)
  qu[127:112] = min(DATA_H[255:192], 2^{16}-1)
```

Schedule

Operands	Use	Def
qu	-	2
rs1	1	-
QACC_H	1	2
QACC_L	1	2

Format

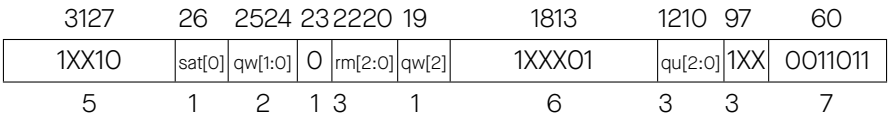


Figure 4.7-334. ESP.SRCMB.U16.Q.QACC

4.7.2.335 ESP.SRCMB.U16.QACC

Syntax

ESP.SRCMB.U16.QACC qu,rs1,sat,rm

Description

This instruction extracts 8 data segments of 64 bits from special registers `QACC_H` and `QACC_L` and performs arithmetic right shift operations respectively. While writing the shift results back to `QACC_H` and `QACC_L`, the instruction saturates the results to 16-bit unsigned numbers and writes the 8 16-bit data segments obtained after saturation into the `qu` register.

Operation

```
temp0[63:0] = QACC_L[ 63:  0]
...
temp3[63:0] = QACC_L[255:192]
temp4[63:0] = QACC_H[ 63:  0]
...
temp7[63:0] = QACC_H[255:192]

temp_shf0[63:0] = temp0[63:0] >> rs1[5:0]
temp_shf1[63:0] = temp1[63:0] >> rs1[5:0]
...
temp_shf7[63:0] = temp7[63:0] >> rs1[5:0]

QACC_L[ 63:  0] = temp_shf0[63:0]
...
QACC_L[255:192] = temp_shf3[63:0]
QACC_H[ 63:  0] = temp_shf4[63:0]
...
QACC_H[255:192] = temp_shf7[63:0]

if sel2 == 0:
  qu[ 15:  0] = temp0[15:0]
  qu[ 31: 16] = temp1[15:0]
  qu[ 47: 32] = temp2[15:0]
  qu[ 63: 48] = temp3[15:0]
  qu[ 79: 64] = temp4[15:0]
  qu[ 95: 80] = temp5[15:0]
  qu[111: 96] = temp6[15:0]
  qu[127:112] = temp7[15:0]
else if sel2==1:
  qu[ 15:  0] = min(temp_shf0[63:0], 2^{16}-1)
  ...
  qu[ 63: 48] = min(temp_shf3[63:0], 2^{16}-1)
  qu[ 79: 64] = min(temp_shf4[63:0], 2^{16}-1)
  ...
  qu[127:112] = min(temp_shf7[63:0], 2^{16}-1)
```

Schedule

Operands	Use	Def
qu	-	2
rs1	1	-
QACC_H	1	2
QACC_L	1	2

Format

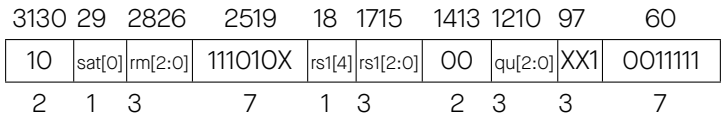


Figure 4.7-335. ESP.SRCMB.U16.QACC

4.7.2.336 ESP.SRCMB.U8.Q.QACC

Syntax

ESP.SRCMB.U8.Q.QACC qu,qw,sat,rm

Description

This instruction extracts 16 data segments of 32 bits from special registers QACC_H and QACC_L and performs arithmetic right shift operations respectively, the shift amounts are from qw. While writing the shift results back to QACC_H and QACC_L, the instruction saturates the results to 8-bit unsigned numbers and writes the 16 8-bit data obtained after saturation into the register qu.

Operation

```
shf_value0[5:0] = (qw[ 7: 0] >= 0x1f) ? 0x1f : qw[ 7: 0]
shf_value1[5:0] = (qw[ 15: 8] >= 0x1f) ? 0x1f : qw[ 15: 8]
shf_value2[5:0] = (qw[ 23: 16] >= 0x1f) ? 0x1f : qw[ 23: 16]
shf_value3[5:0] = (qw[ 31: 24] >= 0x1f) ? 0x1f : qw[ 31: 24]
shf_value4[5:0] = (qw[ 39: 32] >= 0x1f) ? 0x1f : qw[ 39: 32]
shf_value5[5:0] = (qw[ 47: 40] >= 0x1f) ? 0x1f : qw[ 47: 40]
shf_value6[5:0] = (qw[ 55: 48] >= 0x1f) ? 0x1f : qw[ 55: 48]
shf_value7[5:0] = (qw[ 63: 56] >= 0x1f) ? 0x1f : qw[ 63: 56]
shf_value8[5:0] = (qw[ 71: 64] >= 0x1f) ? 0x1f : qw[ 71: 64]
shf_value9[5:0] = (qw[ 79: 72] >= 0x1f) ? 0x1f : qw[ 79: 72]
shf_value10[5:0] = (qw[ 87: 80] >= 0x1f) ? 0x1f : qw[ 87: 80]
shf_value11[5:0] = (qw[ 95: 88] >= 0x1f) ? 0x1f : qw[ 95: 88]
shf_value12[5:0] = (qw[103: 96] >= 0x1f) ? 0x1f : qw[103: 96]
shf_value13[5:0] = (qw[111:104] >= 0x1f) ? 0x1f : qw[111:104]
shf_value14[5:0] = (qw[119:112] >= 0x1f) ? 0x1f : qw[119:112]
shf_value15[5:0] = (qw[127:120] >= 0x1f) ? 0x1f : qw[127:120]

DATA_L[ 31:  0] = QACC_L[ 31:  0] >> shf_value0[4:0]
DATA_L[ 63: 32] = QACC_L[ 63: 32] >> shf_value1[4:0]
DATA_L[ 95: 64] = QACC_L[ 95: 64] >> shf_value2[4:0]
DATA_L[127: 96] = QACC_L[127: 96] >> shf_value3[4:0]
DATA_L[159:128] = QACC_L[159:128] >> shf_value4[4:0]
DATA_L[191:160] = QACC_L[191:160] >> shf_value5[4:0]
DATA_L[223:192] = QACC_L[223:192] >> shf_value6[4:0]
DATA_L[255:224] = QACC_L[255:224] >> shf_value7[4:0]
DATA_H[ 31:  0] = QACC_H[ 31:  0] >> shf_value8[4:0]
DATA_H[ 63: 32] = QACC_H[ 63: 32] >> shf_value9[4:0]
DATA_H[ 95: 64] = QACC_H[ 95: 64] >> shf_value10[4:0]
DATA_H[127: 96] = QACC_H[127: 96] >> shf_value11[4:0]
DATA_H[159:128] = QACC_H[159:128] >> shf_value12[4:0]
DATA_H[191:160] = QACC_H[191:160] >> shf_value13[4:0]
DATA_H[223:192] = QACC_H[223:192] >> shf_value14[4:0]
DATA_H[255:224] = QACC_H[255:224] >> shf_value15[4:0]

QACC_L[255:0] = DATA_L[255:0]
QACC_H[255:0] = DATA_H[255:0]

if sel2 == 0:
  qu[  7:  0] = DATA_L[  7:  0]
```

```
    qu[ 15:  8] = DATA_L[ 39: 32]
    qu[ 23: 16] = DATA_L[ 71: 64]
    qu[ 31: 24] = DATA_L[103: 96]
    qu[ 39: 32] = DATA_L[135:128]
    qu[ 47: 40] = DATA_L[167:160]
    qu[ 55: 48] = DATA_L[199:192]
    qu[ 63: 56] = DATA_L[231:224]
    qu[ 71: 64] = DATA_H[  7:  0]
    qu[ 79: 72] = DATA_H[ 39: 32]
    qu[ 87: 80] = DATA_H[ 71: 64]
    qu[ 95: 88] = DATA_H[103: 96]
    qu[103: 96] = DATA_H[135:128]
    qu[111:104] = DATA_H[167:160]
    qu[119:112] = DATA_H[199:192]
    qu[127:120] = DATA_H[231:224]
  else if sel2 == 1:
    qu[  7:  0] = min(DATA_L[ 31:  0], 2^{8}-1)
    qu[ 15:  8] = min(DATA_L[ 63: 32], 2^{8}-1)
    qu[ 23: 16] = min(DATA_L[ 95: 64], 2^{8}-1)
    qu[ 31: 24] = min(DATA_L[127: 96], 2^{8}-1)
    qu[ 39: 32] = min(DATA_L[159:128], 2^{8}-1)
    qu[ 47: 40] = min(DATA_L[191:160], 2^{8}-1)
    qu[ 55: 48] = min(DATA_L[223:192], 2^{8}-1)
    qu[ 63: 56] = min(DATA_L[255:224], 2^{8}-1)
    qu[ 71: 64] = min(DATA_L[ 31:  0], 2^{8}-1)
    qu[ 79: 72] = min(DATA_H[ 63: 32], 2^{8}-1)
    qu[ 87: 80] = min(DATA_H[ 95: 64], 2^{8}-1)
    qu[ 95: 88] = min(DATA_H[127: 96], 2^{8}-1)
    qu[103: 96] = min(DATA_H[159:128], 2^{8}-1)
    qu[111:104] = min(DATA_H[191:160], 2^{8}-1)
    qu[119:112] = min(DATA_H[223:192], 2^{8}-1)
    qu[127:120] = min(DATA_H[255:224], 2^{8}-1)
```

Schedule

Operands	Use	Def
qu	-	2
rs1	1	-
QACC_H	1	2
QACC_L	1	2

Format

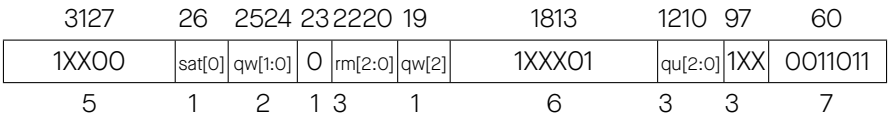


Figure 4.7-336. ESP.SRCMB.U8.Q.QACC

4.7.2.337 ESP.SRCMB.U8.QACC

Syntax

ESP.SRCMB.U8.QACC qu,rs1,sat,rm

Description

This instruction extracts 16 data segments of 32 bits from special registers `QACC_H` and `QACC_L` and performs arithmetic right shift operations respectively. While writing the shift result back to `QACC_H` and `QACC_L`, the instruction saturates the result to 8-bit unsigned numbers and writes the 16 8-bit data obtained after saturation into the register `qu`.

Operation

```
DATA_L[ 31:  0] = QACC_L[ 31:  0] >> rs1[4:0]
DATA_L[ 63: 32] = QACC_L[ 63: 32] >> rs1[4:0]
DATA_L[ 95: 64] = QACC_L[ 95: 64] >> rs1[4:0]
DATA_L[127: 96] = QACC_L[127: 96] >> rs1[4:0]
DATA_L[159:128] = QACC_L[159:128] >> rs1[4:0]
DATA_L[191:160] = QACC_L[191:160] >> rs1[4:0]
DATA_L[223:192] = QACC_L[223:192] >> rs1[4:0]
DATA_L[255:224] = QACC_L[255:224] >> rs1[4:0]
DATA_H[ 31:  0] = QACC_H[ 31:  0] >> rs1[4:0]
DATA_H[ 63: 32] = QACC_H[ 63: 32] >> rs1[4:0]
DATA_H[ 95: 64] = QACC_H[ 95: 64] >> rs1[4:0]
DATA_H[127: 96] = QACC_H[127: 96] >> rs1[4:0]
DATA_H[159:128] = QACC_H[159:128] >> rs1[4:0]
DATA_H[191:160] = QACC_H[191:160] >> rs1[4:0]
DATA_H[223:192] = QACC_H[223:192] >> rs1[4:0]
DATA_H[255:224] = QACC_H[255:224] >> rs1[4:0]

QACC_L[255:0] = DATA_L[255:0]
QACC_H[255:0] = DATA_H[255:0]

if sel2 == 0:
  qu[  7:  0] = DATA_L[  7:  0]
  qu[ 15:  8] = DATA_L[ 39: 32]
  qu[ 23: 16] = DATA_L[ 71: 64]
  qu[ 31: 24] = DATA_L[103: 96]
  qu[ 39: 32] = DATA_L[135:128]
  qu[ 47: 40] = DATA_L[167:160]
  qu[ 55: 48] = DATA_L[199:192]
  qu[ 63: 56] = DATA_L[231:224]
  qu[ 71: 64] = DATA_H[  7:  0]
  qu[ 79: 72] = DATA_H[ 39: 32]
  qu[ 87: 80] = DATA_H[ 71: 64]
  qu[ 95: 88] = DATA_H[103: 96]
  qu[103: 96] = DATA_H[135:128]
  qu[111:104] = DATA_H[167:160]
  qu[119:112] = DATA_H[199:192]
  qu[127:120] = DATA_H[231:224]
else if sel2 == 1:
```

```
    qu[  7:  0] = min(DATA_L[ 31:  0], 2^{8}-1)
    qu[ 15:  8] = min(DATA_L[ 63: 32], 2^{8}-1)
    qu[ 23: 16] = min(DATA_L[ 95: 64], 2^{8}-1)
    qu[ 31: 24] = min(DATA_L[127: 96], 2^{8}-1)
    qu[ 39: 32] = min(DATA_L[159:128], 2^{8}-1)
    qu[ 47: 40] = min(DATA_L[191:160], 2^{8}-1)
    qu[ 55: 48] = min(DATA_L[223:192], 2^{8}-1)
    qu[ 63: 56] = min(DATA_L[255:224], 2^{8}-1)
    qu[ 71: 64] = min(DATA_L[ 31:  0], 2^{8}-1)
    qu[ 79: 72] = min(DATA_H[ 63: 32], 2^{8}-1)
    qu[ 87: 80] = min(DATA_H[ 95: 64], 2^{8}-1)
    qu[ 95: 88] = min(DATA_H[127: 96], 2^{8}-1)
    qu[103: 96] = min(DATA_H[159:128], 2^{8}-1)
    qu[111:104] = min(DATA_H[191:160], 2^{8}-1)
    qu[119:112] = min(DATA_H[223:192], 2^{8}-1)
    qu[127:120] = min(DATA_H[255:224], 2^{8}-1)
```

Schedule

Operands	Use	Def
qu	-	2
rs1	1	-
QACC_H	1	2
QACC_L	1	2

Format

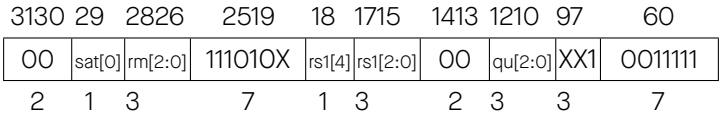


Figure 4.7-337. ESP.SRCMB.U8.QACC

4.7.2.338 ESP.SRCQ.128.ST.INCP

Syntax

ESP.SRCQ.128.ST.INCP qw,qy,rs1

Description

This instruction performs an arithmetic right shift on the 32-byte concatenation of registers `qw` and `qy`. Then, it writes the lower 128 bits of the shift result to memory. After the access, the value in the register `rs1` is incremented by 16.

Operation

```
{qy[127:  0], qw[127:  0]} >> {SAR_BYTE[3:0] << 3}  => store128(rs1[31:0])
rs1[31:0] = rs1[31:0] + 16
```

Schedule

Operands	Use	Def
qz	-	1
qu	1	1
qv	1	2
rs1	1	1
SAR_BYTES	1	-

Format

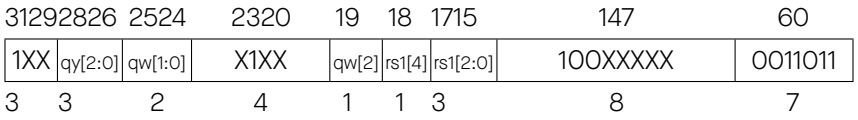


Figure 4.7-338. ESP.SRCQ.128.ST.INCP

4.7.2.339 ESP.SRCXXP.2Q

Syntax

ESP.SRCXXP.2Q qy,qw,rs1,rs2

Description

This instruction performs a logical right shift on the 32-byte concatenation of registers `qw` and `qy`, and pads the higher bits with 0. The upper 128 bits of the shift result are written to register `qy`, and the lower 128 bits are written to `qw`. The right shift amount is 8 times the sum of 1 plus the lower 4-bit value of register `rs1`. After the above operations, the value in `rs1` is incremented by the value in `rs2`.

Operation

```
{qy[127:  0], qw[127:  0]} = {qy[127:  0], qw[127:  0]} >> ((rs1[3:0]+1)*8)
qy[127:127-8*rs1[3:  0]] = 0
rs1[31:0] = rs1[31:0] + rs2[31:0]
```

Schedule

Operands	Use	Def
qy	1	1
qw	1	1
rs1	1	1
rs2	1	-

Format

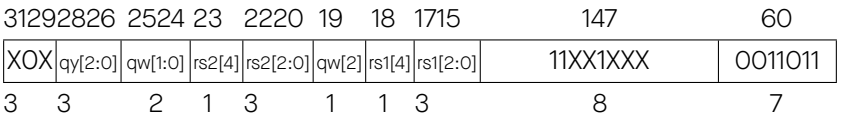


Figure 4.7-339. ESP.SRCXXP.2Q

4.7.2.340 ESP.SRS.S.XACC

Syntax

ESP.SRS.S.XACC rd,rs1,sat,rm

Description

This instruction performs an arithmetic right shift on the special register `XACC`. While writing the shift result back to `XACC`, the instruction saturates the shift result to a 32-bit signed number and writes the saturated result into the register `rd`.

Operation

```
temp_shf[39:0] = XACC[39:0] >> rs1[5:0]
XACC = temp_shf[39:0]
rd = min(max(temp_shf[39:0], -2^{31}), 2^{31}-1)
```

Schedule

Operands	Use	Def
rd	-	2
rs1	1	-
XACC	1	2

Format

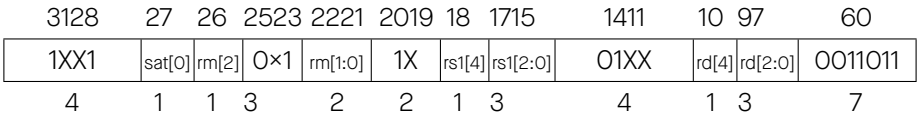


Figure 4.7-340. ESP.SRS.S.XACC

4.7.2.341 ESP.SRS.U.XACC

Syntax

ESP.SRS.U.XACC rd,rs1,sat,rm

Description

This instruction performs an arithmetic right shift on the special register `XACC`. While writing the shift result back to `XACC`, the instruction saturates the shift result to a 32-bit unsigned number and writes the saturated result into the register `rd`.

Operation

```
temp_shf[39:0] = XACC[39:0] >> rs1[5:0]
XACC = temp_shf[39:0]
rd = min(temp_shf[39:0], 2^{32}-1)
```

Schedule

Operands	Use	Def
rd	-	2
rs1	1	-
XACC	1	2

Format

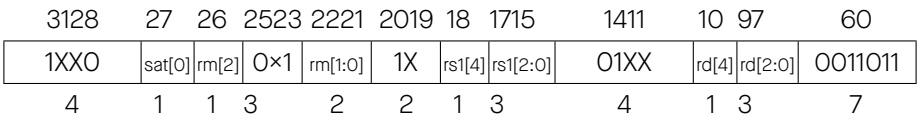


Figure 4.7-341. ESP.SRS.U.XACC

4.7.2.342 ESP.VSLD.16

Syntax

ESP.VSLD.16 qu,qy,qw,sat,rm

Description

This instruction performs a 16-bit vector architecture shift and the shift amount comes from segments of qw. If the shift amount is a negative number, it performs a right shift; otherwise, it performs a left shift.

Operation

```
qu[ 15:  0] = (qw[ 4: 0] < 0) ? (qy[ 15: 0] » (-qw[ 4: 0]) ) : (qy[ 15: 0] « qw[ 4: 0])
qu[ 31: 16] = (qw[ 20: 16] < 0) ? (qy[ 31: 16] » (-qw[ 20: 16]) ) : (qy[ 31: 16] « qw[ 20: 16])
qu[ 47: 32] = (qw[ 36: 32] < 0) ? (qy[ 47: 32] » (-qw[ 36: 32]) ) : (qy[ 47: 32] « qw[ 36: 32])
qu[ 63: 48] = (qw[ 52: 48] < 0) ? (qy[ 63: 48] » (-qw[ 52: 48]) ) : (qy[ 63: 48] « qw[ 52: 48])
qu[ 79: 64] = (qw[ 68: 64] < 0) ? (qy[ 79: 64] » (-qw[ 68: 64]) ) : (qy[ 79: 64] « qw[ 68: 64])
qu[ 95: 80] = (qw[ 84: 80] < 0) ? (qy[ 95: 80] » (-qw[ 84: 80]) ) : (qy[ 95: 80] « qw[ 84: 80])
qu[111: 96] = (qw[100: 96] < 0) ? (qy[111: 96] » (-qw[100: 96]) ) : (qy[111: 96] « qw[100: 96])
qu[127:112] = (qw[116:112] < 0) ? (qy[127:112] » (-qw[116:112]) ) : (qy[127:112] « qw[116:112])
```

Schedule

Operands	Use	Def
qu	-	1
qy	1	-
qw	1	-

Format

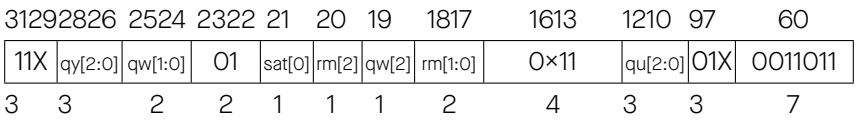


Figure 4.7-342. ESP.VSLD.16

4.7.2.343 ESP.VSLD.32

Syntax

ESP.VSLD.32 qu,qy,qw,sat,rm

Description

This instruction performs a 32-bit vector architecture shift and the shift amount comes from segments of qw. If the shift amount is a negative number, it performs a right shift; otherwise, it performs a left shift.

Operation

```
qu[ 31:  0] = (qw[ 5: 0] < 0) ? (qy[ 31: 0] » (-qw[ 5: 0]) ) : (qy[ 31: 0] « qw[ 5: 0])
qu[ 63: 32] = (qw[ 37: 32] < 0) ? (qy[ 63: 32] » (-qw[ 37: 32]) ) : (qy[ 63: 32] « qw[ 37: 32])
qu[ 95: 64] = (qw[ 69: 64] < 0) ? (qy[ 95: 64] » (-qw[ 69: 64]) ) : (qy[ 95: 64] « qw[ 69: 64])
qu[127: 96] = (qw[101: 96] < 0) ? (qy[127: 96] » (-qw[101: 96]) ) : (qy[127: 96] « qw[101: 96])
```

Schedule

Operands	Use	Def
qu	-	1
qy	1	-
qw	1	-

Format

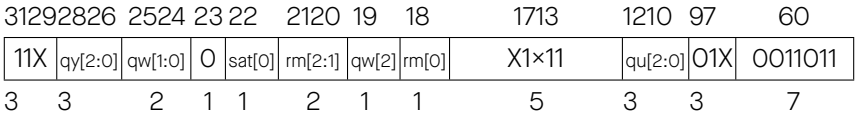


Figure 4.7-343. ESP.VSLD.32

4.7.2.344 ESP.VSLD.8

Syntax

ESP.VSLD.8 qu,qy,qw,sat,rm

Description

This instruction performs an 8-bit vector architecture shift and the shift amount comes from segments of qw. If the shift amount is a negative number, it performs a right shift; otherwise, it performs a left shift.

Operation

```
qu[  7:  0] = (qw[ 3: 0] < 0) ? (qy[ 7: 0] » (-qw[ 3: 0]) ) : (qy[ 7: 0] « qw[ 3: 0])
qu[ 15:  8] = (qw[ 11: 8] < 0) ? (qy[ 15: 8] » (-qw[ 11: 8]) ) : (qy[ 15: 8] « qw[ 11: 8])
qu[ 23: 16] = (qw[ 19: 16] < 0) ? (qy[ 23: 16] » (-qw[ 19: 16]) ) : (qy[ 23: 16] « qw[ 19: 16])
qu[ 31: 24] = (qw[ 27: 24] < 0) ? (qy[ 31: 24] » (-qw[ 27: 24]) ) : (qy[ 31: 24] « qw[ 27: 24])
qu[ 39: 32] = (qw[ 35: 32] < 0) ? (qy[ 39: 32] » (-qw[ 35: 32]) ) : (qy[ 39: 32] « qw[ 35: 32])
qu[ 47: 40] = (qw[ 43: 40] < 0) ? (qy[ 47: 40] » (-qw[ 43: 40]) ) : (qy[ 47: 40] « qw[ 43: 40])
qu[ 55: 48] = (qw[ 51: 48] < 0) ? (qy[ 55: 48] » (-qw[ 51: 48]) ) : (qy[ 55: 48] « qw[ 51: 48])
qu[ 63: 56] = (qw[ 59: 56] < 0) ? (qy[ 63: 56] » (-qw[ 59: 56]) ) : (qy[ 63: 56] « qw[ 59: 56])
qu[ 71: 64] = (qw[ 67: 64] < 0) ? (qy[ 71: 64] » (-qw[ 67: 64]) ) : (qy[ 71: 64] « qw[ 67: 64])
qu[ 79: 72] = (qw[ 75: 72] < 0) ? (qy[ 79: 72] » (-qw[ 75: 72]) ) : (qy[ 79: 72] « qw[ 75: 72])
qu[ 87: 80] = (qw[ 83: 80] < 0) ? (qy[ 87: 80] » (-qw[ 83: 80]) ) : (qy[ 87: 80] « qw[ 83: 80])
qu[ 95: 88] = (qw[ 91: 88] < 0) ? (qy[ 95: 88] » (-qw[ 91: 88]) ) : (qy[ 95: 88] « qw[ 91: 88])
qu[103: 96] = (qw[ 99: 96] < 0) ? (qy[103: 96] » (-qw[ 99: 96]) ) : (qy[103: 96] « qw[ 99: 96])
qu[111:104] = (qw[107:104] < 0) ? (qy[111:104] » (-qw[107:104]) ) : (qy[111:104] « qw[107:104])
qu[119:112] = (qw[115:112] < 0) ? (qy[119:112] » (-qw[115:112]) ) : (qy[119:112] « qw[115:112])
qu[127:120] = (qw[123:120] < 0) ? (qy[127:120] » (-qw[123:120]) ) : (qy[127:120] « qw[123:120])
```

Schedule

Operands	Use	Def
qu	-	1
qy	1	-
qw	1	-

Format

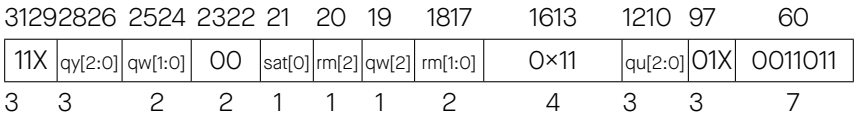


Figure 4.7-344. ESP.VSLD.8

4.7.2.345 ESP.VSR.S32

Syntax

ESP.VSR.S32 qu,qy,rm

Description

This instruction performs an arithmetic right shift on the four 32-bit data segments in the register qy respectively. The shift amount is the value in the 6-bit special register SAR. During the shift process, the higher bits are padded with the signed bit. The lower 32 bits of the shift result are stored in the corresponding data segment in the register qu.

Operation

```
qu[ 31:  0] = (qy[ 31:  0] >> SAR[5:0])
qu[ 63: 32] = (qy[ 63: 32] >> SAR[5:0])
qu[ 95: 64] = (qy[ 95: 64] >> SAR[5:0])
qu[127: 96] = (qy[127: 96] >> SAR[5:0])
```

Schedule

Operands	Use	Def
qu	-	1
qy	1	-
SAR	1	-

Format

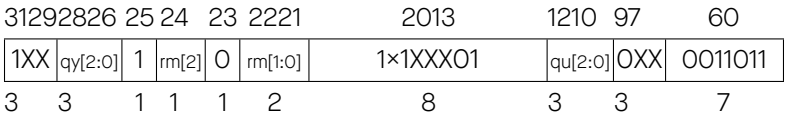


Figure 4.7-345. ESP.VSR.S32

4.7.2.346 ESP.VSR.U32

Syntax

ESP.VSR.U32 qu,qy,rm

Description

This instruction performs an arithmetic right shift on the four 32-bit unsigned data segments in the register qy respectively. The shift amount is the value in the 6-bit special register SAR. During the shift process, the higher bits are padded with the signed bit. The lower 32 bits of the shift result are stored in the corresponding data segment in the register qu.

Operation

```
qu[ 31:  0] = (qy[ 31:  0] >> SAR[5:0])
qu[ 63: 32] = (qy[ 63: 32] >> SAR[5:0])
qu[ 95: 64] = (qy[ 95: 64] >> SAR[5:0])
qu[127: 96] = (qy[127: 96] >> SAR[5:0])
```

Schedule

Operands	Use	Def
qu	-	1
qy	1	-
SAR	1	-

Format

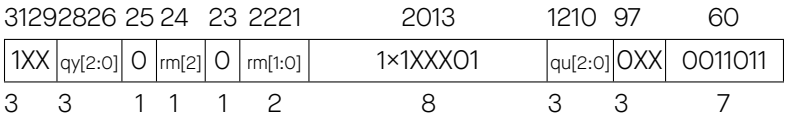


Figure 4.7-346. ESP.VSR.U32

4.7.2.347 ESP.VSRD.16

Syntax

ESP.VSRD.16 qu,qy,qw,sat,rm

Description

This instruction performs a 16-bit vector architecture shift and the shift amount comes from segments of qw. If the shift amount is a negative number, it performs a left shift; otherwise, it performs a right shift.

Operation

```
qu[ 15:  0] = (qw[ 4: 0] < 0) ? (qy[ 15: 0] « (-qw[ 4: 0]) ) : (qy[ 15: 0] » qw[ 4: 0])
qu[ 31: 16] = (qw[ 20: 16] < 0) ? (qy[ 31: 16] « (-qw[ 20: 16]) ) : (qy[ 31: 16] » qw[ 20: 16])
qu[ 47: 32] = (qw[ 36: 32] < 0) ? (qy[ 47: 32] « (-qw[ 36: 32]) ) : (qy[ 47: 32] » qw[ 36: 32])
qu[ 63: 48] = (qw[ 52: 48] < 0) ? (qy[ 63: 48] « (-qw[ 52: 48]) ) : (qy[ 63: 48] » qw[ 52: 48])
qu[ 79: 64] = (qw[ 68: 64] < 0) ? (qy[ 79: 64] « (-qw[ 68: 64]) ) : (qy[ 79: 64] » qw[ 68: 64])
qu[ 95: 80] = (qw[ 84: 80] < 0) ? (qy[ 95: 80] « (-qw[ 84: 80]) ) : (qy[ 95: 80] » qw[ 84: 80])
qu[111: 96] = (qw[100: 96] < 0) ? (qy[111: 96] « (-qw[100: 96]) ) : (qy[111: 96] » qw[100: 96])
qu[127:112] = (qw[116:112] < 0) ? (qy[127:112] « (-qw[116:112]) ) : (qy[127:112] » qw[116:112])
```

Schedule

Operands	Use	Def
qu	-	1
qy	1	-
qw	1	-

Format

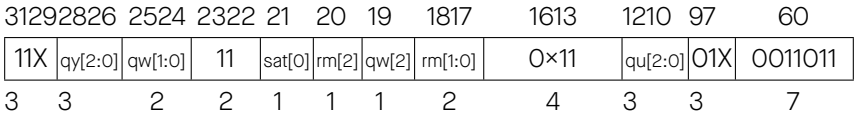


Figure 4.7-347. ESP.VSRD.16

4.7.2.348 ESP.VSRD.32

Syntax

ESP.VSRD.32 qu,qy,qw,sat,rm

Description

This instruction performs a 32-bit vector architecture shift and the shift amount comes from segments of qw. If the shift amount is a negative number, it performs a left shift; otherwise, it performs a right shift.

Operation

```
qu[ 31:  0] = (qw[ 5: 0] < 0) ? (qy[ 31: 0] « (-qw[ 5: 0]) ) : (qy[ 31: 0] » qw[ 5: 0])
qu[ 63: 32] = (qw[ 37: 32] < 0) ? (qy[ 63: 32] « (-qw[ 37: 32]) ) : (qy[ 63: 32] » qw[ 37: 32])
qu[ 95: 64] = (qw[ 69: 64] < 0) ? (qy[ 95: 64] « (-qw[ 69: 64]) ) : (qy[ 95: 64] » qw[ 69: 64])
qu[127: 96] = (qw[101: 96] < 0) ? (qy[127: 96] « (-qw[101: 96]) ) : (qy[127: 96] » qw[101: 96])
```

Schedule

Operands	Use	Def
qu	-	1
qy	1	-
qw	1	-

Format

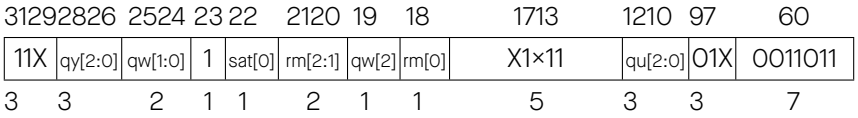


Figure 4.7-348. ESP.VSRD.32

4.7.2.349 ESP.VSRD.8

Syntax

ESP.VSRD.8 qu,qy,qw,sat,rm

Description

This instruction performs an 8-bit vector architecture shift, and the shift amount comes from segments of qw. If the shift amount is a negative number, it performs a left shift; otherwise, it performs a right shift.

Operation

```
qu[  7:  0] = (qw[ 3: 0] < 0) ? (qy[ 7: 0] « (-qw[ 3: 0]) ) : (qy[ 7: 0] » qw[ 3: 0])
qu[ 15:  8] = (qw[ 11: 8] < 0) ? (qy[ 15: 8] « (-qw[ 11: 8]) ) : (qy[ 15: 8] » qw[ 11: 8])
qu[ 23: 16] = (qw[ 19: 16] < 0) ? (qy[ 23: 16] « (-qw[ 19: 16]) ) : (qy[ 23: 16] » qw[ 19: 16])
qu[ 31: 24] = (qw[ 27: 24] < 0) ? (qy[ 31: 24] « (-qw[ 27: 24]) ) : (qy[ 31: 24] » qw[ 27: 24])
qu[ 39: 32] = (qw[ 35: 32] < 0) ? (qy[ 39: 32] « (-qw[ 35: 32]) ) : (qy[ 39: 32] » qw[ 35: 32])
qu[ 47: 40] = (qw[ 43: 40] < 0) ? (qy[ 47: 40] « (-qw[ 43: 40]) ) : (qy[ 47: 40] » qw[ 43: 40])
qu[ 55: 48] = (qw[ 51: 48] < 0) ? (qy[ 55: 48] « (-qw[ 51: 48]) ) : (qy[ 55: 48] » qw[ 51: 48])
qu[ 63: 56] = (qw[ 59: 56] < 0) ? (qy[ 63: 56] « (-qw[ 59: 56]) ) : (qy[ 63: 56] » qw[ 59: 56])
qu[ 71: 64] = (qw[ 67: 64] < 0) ? (qy[ 71: 64] « (-qw[ 67: 64]) ) : (qy[ 71: 64] » qw[ 67: 64])
qu[ 79: 72] = (qw[ 75: 72] < 0) ? (qy[ 79: 72] « (-qw[ 75: 72]) ) : (qy[ 79: 72] » qw[ 75: 72])
qu[ 87: 80] = (qw[ 83: 80] < 0) ? (qy[ 87: 80] « (-qw[ 83: 80]) ) : (qy[ 87: 80] » qw[ 83: 80])
qu[ 95: 88] = (qw[ 91: 88] < 0) ? (qy[ 95: 88] « (-qw[ 91: 88]) ) : (qy[ 95: 88] » qw[ 91: 88])
qu[103: 96] = (qw[ 99: 96] < 0) ? (qy[103: 96] « (-qw[ 99: 96]) ) : (qy[103: 96] » qw[ 99: 96])
qu[111:104] = (qw[107:104] < 0) ? (qy[111:104] « (-qw[107:104]) ) : (qy[111:104] » qw[107:104])
qu[119:112] = (qw[115:112] < 0) ? (qy[119:112] « (-qw[115:112]) ) : (qy[119:112] » qw[115:112])
qu[127:120] = (qw[123:120] < 0) ? (qy[127:120] « (-qw[123:120]) ) : (qy[127:120] » qw[123:120])
```

Operands	Use	Def
qu	-	1
qy	1	-
qw	1	-

Format

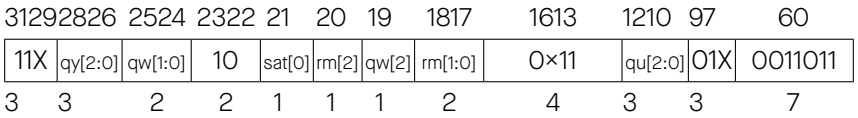


Figure 4.7-349. ESP.VSRD.8

4.7.2.350 ESP.ST.S.XACC.IP

Syntax

ESP.ST.S.XACC.IP rs1,imm8

Description

This instruction sign-extends the special register XACC to 64 bits and stores the result in memory. After the access is completed, the value in the register `rs1` is incremented by an 8-bit sign-extended constant in the instruction code segment, left-shifted by 3.
imm starts at -1024, ends at 1016, and steps by 8.

Operation

```
MEM(rs1[31:0]) = {24{XACC[39]}, XACC[39:0]}
rs1[31:0] = rs1[31:0] + imm
```

Schedule

Operands	Use	Def
XACC	2	-
rs1	1	1

Format

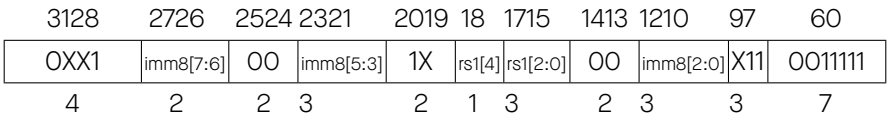


Figure 4.7-350. ESP.ST.S.XACC.IP

4.7.2.351 ESP.ST.U.XACC.IP

Syntax

ESP.ST.U.XACC.IP rs1,imm8

Description

This instruction zero-extends the special register `XACC` to 64 bits and stores the result in memory. After the access is completed, the value in the register `rs1` is incremented by an 8-bit sign-extended constant in the instruction code segment, left-shifted by 3.
imm starts at -1024, ends at 1016, and steps by 8.

Operation

```
MEM(rs1[31:0]) = {24{0}, XACC[39:0]}
rs1[31:0] = rs1[31:0] + imm
```

Schedule

Operands	Use	Def
XACC	2	-
rs1	1	1

Format

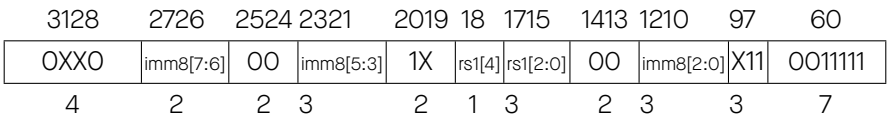


Figure 4.7-351. ESP.ST.U.XACC.IP

4.7.2.352 ESP.FFT.BITREV

Syntax

ESP.FFT.BITREV qv,rs1

Description

This instruction swaps the bit order of data of different bit widths according to the value of the special register `FFT_BIT_WIDTH`. Then, it compares the data before and after the swap, takes the larger value, pads the higher bits with 0 until it reaches 16 bits, and writes the result into the corresponding data segment of the register `qa`. In the following, Switchx is a function that represents the bit inversion of the x-bit. Switch5(0b10100) = 0b00101. Here, 0b10100 represents 5-bit binary data.

Operation

```
  temp0[15:0] = rs1[15:0]
  temp1[15:0] = rs1[15:0] + 1
  temp2[15:0] = rs1[15:0] + 2
  temp3[15:0] = rs1[15:0] + 3
  temp4[15:0] = rs1[15:0] + 4
  temp5[15:0] = rs1[15:0] + 5
  temp6[15:0] = rs1[15:0] + 6
  temp7[15:0] = rs1[15:0] + 7
if FFT_BIT_WIDTH==3:
  Switch3(X[2:0]) = X[0:2]
  qv[ 15:  0] = {13'h0, max(tmp0[2:0], Switch3(tmp0[2:0]))}
  qv[ 31: 16] = {13'h0, max(tmp1[2:0], Switch3(tmp1[2:0]))}
  qv[ 47: 32] = {13'h0, max(tmp2[2:0], Switch3(tmp2[2:0]))}
  qv[ 63: 48] = {13'h0, max(tmp3[2:0], Switch3(tmp3[2:0]))}
  qv[ 79: 64] = {13'h0, max(tmp4[2:0], Switch3(tmp4[2:0]))}
  qv[ 95: 80] = {13'h0, max(tmp5[2:0], Switch3(tmp5[2:0]))}
  qv[127: 96] = 0
if FFT_BIT_WIDTH==4:
  Switch4(X[3:0]) = X[0:3]
  qv[ 15:  0] = {12'h0, max(tmp0[3:0], Switch4(tmp0[3:0]))}
  qv[ 31: 16] = {12'h0, max(tmp1[3:0], Switch4(tmp1[3:0]))}
  qv[ 47: 32] = {12'h0, max(tmp2[3:0], Switch4(tmp2[3:0]))}
  qv[ 63: 48] = {12'h0, max(tmp3[3:0], Switch4(tmp3[3:0]))}
  qv[ 79: 64] = {12'h0, max(tmp4[3:0], Switch4(tmp4[3:0]))}
  qv[ 95: 80] = {12'h0, max(tmp5[3:0], Switch4(tmp5[3:0]))}
  qv[111: 96] = {12'h0, max(tmp6[3:0], Switch4(tmp6[3:0]))}
  qv[127:112] = {12'h0, max(tmp7[3:0], Switch4(tmp7[3:0]))}
  ...
if FFT_BIT_WIDTH==10:
  Switch10(X[9:0]) = X[0:9]
  qv[ 15:  0] = {6'h0, max(tmp0[9:0], Switch10(tmp0[9:0]))}
  qv[ 31: 16] = {6'h0, max(tmp1[9:0], Switch10(tmp1[9:0]))}
  qv[ 47: 32] = {6'h0, max(tmp2[9:0], Switch10(tmp2[9:0]))}
  qv[ 63: 48] = {6'h0, max(tmp3[9:0], Switch10(tmp3[9:0]))}
  qv[ 79: 64] = {6'h0, max(tmp4[9:0], Switch10(tmp4[9:0]))}
  qv[ 95: 80] = {6'h0, max(tmp5[9:0], Switch10(tmp5[9:0]))}
  qv[111: 96] = {6'h0, max(tmp6[9:0], Switch10(tmp6[9:0]))}
  qv[127:112] = {6'h0, max(tmp7[9:0], Switch10(tmp7[9:0]))}
```

```
39
40  rs1[31:0] = rs1[31:0] + 8
```

Schedule

Operands	Use	Def
qv	-	1
rs1	1	1

Format

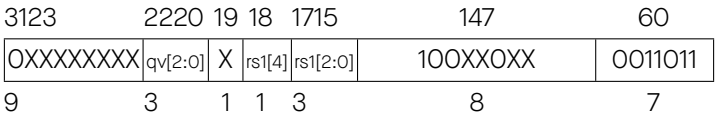


Figure 4.7-352. ESP.FFT.BITREV

4.7.2.353 ESP.SRC.Q.QUP

Syntax

ESP.SRC.Q.QUP qz,qw,qy

Description

This instruction performs an arithmetic right shift on the 32-byte concatenation of registers `qw` and `qy`, which hold the loaded data from two consecutive aligned addresses. In this way, it is possible to obtain unaligned 16-byte data, which will then be written to the register `qz`. The right shift amount is `SAR_BYTE` performed by 8. Simultaneously, the value of the register `qy` is updated to `qw`.

Operation

```
qz[127:  0] = {qy[127:  0], qw[127:  0]} >> {SAR_BYTE[3:0] << 3}
qw = qy
```

Schedule

Operands	Use	Def
qz	-	1
qw	1	-
qy	1	-
SAR_BYTES	1	-

Format

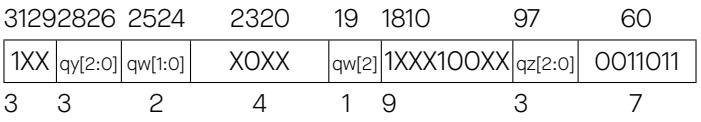


Figure 4.7-353. ESP.SRC.Q.QUP

4.7.2.354 ESP.VSL.S32

Syntax

ESP.VSL.S32 qu,qy,sat

Description

This instruction performs a left shift on the four 32-bit signed data segments in the register qy respectively. The left shift amount is the value in the 6-bit special register SAR. During the shift process, the lower bits are padded with 0. The lower 32 bits of the shift result are stored in the corresponding data segment in the register qu.

Operation

```
qu[ 31:  0] = (qy[ 31:  0] << SAR[5:0])
qu[ 63: 32] = (qy[ 63: 32] << SAR[5:0])
qu[ 95: 64] = (qy[ 95: 64] << SAR[5:0])
qu[127: 96] = (qy[127: 96] << SAR[5:0])
```

Schedule

Operands	Use	Def
qu	-	1
qy	1	-
SAR	1	-

Format

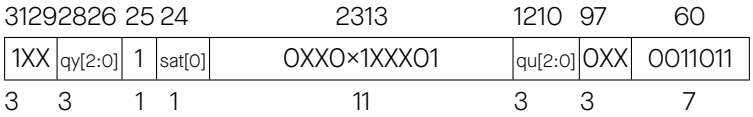


Figure 4.7-354. ESP.VSL.S32

4.7.2.355 ESP.VSL.U32

Syntax

ESP.VSL.U32 qu,qy,sat

Description

This instruction performs a left shift on the four 32-bit unsigned data segments in the register qy respectively. The left shift amount is the value in the 6-bit special register SAR. During the shift process, the lower bits are padded with 0. The lower 32 bits of the shift result are stored in the corresponding data segment in the register qu.

Operation

```
qu[ 31:  0] = (qy[ 31:  0] << SAR[5:0])
qu[ 63: 32] = (qy[ 63: 32] << SAR[5:0])
qu[ 95: 64] = (qy[ 95: 64] << SAR[5:0])
qu[127: 96] = (qy[127: 96] << SAR[5:0])
```

Schedule

Operands	Use	Def
qu	-	1
qy	1	-
SAR	1	-

Format

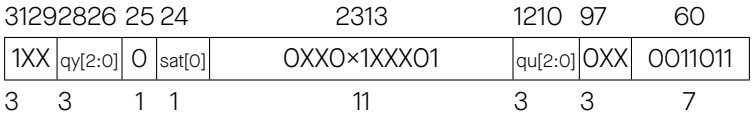


Figure 4.7-355. ESP.VSL.U32

Chapter 5

Low-Power CPU

5.1 Overview

The ESP32-P4 Low-Power CPU (LP CPU) is a 32-bit processor based upon RISC-V ISA comprising integer (I), multiplication/division (M), atomic (A), and compressed (C) standard extensions. It features ultra-low power consumption and has a 2-stage, in-order, and scalar pipeline. The LP CPU core complex has an interrupt controller (INTC), a debug module (DM), and system bus (SYS BUS) interfaces for memory and peripheral access.

The LP CPU is in sleep mode by default (see Section 5.10). It can stay powered on when the chip enters Deep-sleep mode (see Chapter 15 *Low-Power Management* for details) and can access most peripherals and memories (see Chapter 9 *System and Memory* for details). It has two application scenarios:

- Power insensitive scenario: When the High-Performance CPU (HP CPU) is active, the LP CPU can assist the it with some speed- and efficiency-insensitive controls and computations.
- Power sensitive scenario: When the HP CPU is in the power-down state to save power, the LP CPU can be woken up to handle some external wake-up events.

Figure 5.1-1 shows the resources accessible to the HP CPU and the LP CPU.

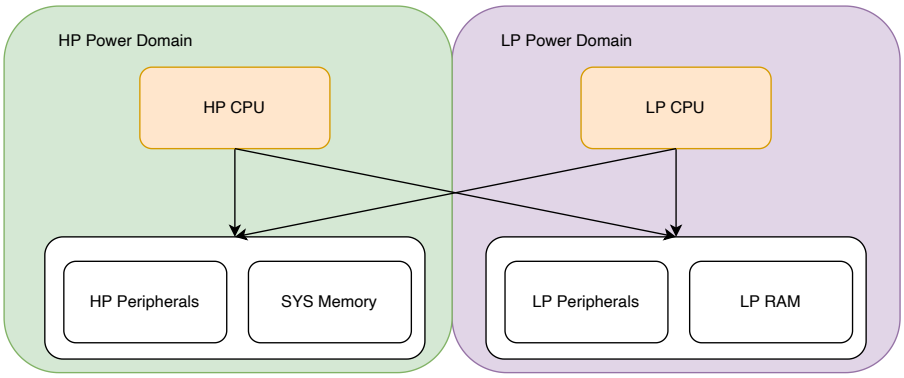


Figure 5.1-1. LP CPU Overview

In the figure above, the HP Power Domain and LP Power Domain can be powered on and off independently. When one domain is powered down, the CPU of the other domain cannot access the resources of that domain. In other words, only when the domain is powered on can its resources be accessed. For more information about power, please refer to the chapter 15 *Low-Power Management*.

5.2 Features

The LP CPU has the following features:

- Operating clock frequency of up to 40 MHz
- 18 vector interrupts
- Debug module compliant with RISC-V External Debug Support Version 0.13 with external debugger support over an industry-standard JTAG/USB port
- Hardware trigger compliant with RISC-V External Debug Support Version 0.13 with up to 2 breakpoints/watchpoints
- Core performance metric events
- Wake-up interrupt for HP CPU
- Access to HP memory and LP memory
- Access to the entire peripheral address space

5.3 Configuration and Status Registers (CSRs)

5.3.1 Register Summary

Below is a list of CSRs that are supported by the LP CPU. Except for the custom performance counter CSRs, all the implemented CSRs follow the standard mapping of bit fields as described in the RISC-V Instruction Set Manual, Volume II: Privileged Architecture, Version 1.10. It must be noted that even among the standard CSRs, not all bit fields have been implemented, limited by the subset of features implemented in the CPU. For a detailed description of the subset of fields implemented under each of these CSRs, please refer to Section 5.3.2.

The abbreviations given in Column **Access** are explained in Section *Access Types for Registers*.

Name	Description	Address	Access
Machine Information CSR			
mhartid	Machine Hart ID	0xF14	RO
Machine Trap Setup CSRs			
mstatus	Machine Mode Status	0x300	R/W
misa [1]	Machine ISA	0x301	R/W
mie	Machine Interrupt Enable	0x304	R/W
mtvec [2]	Machine Trap Vector	0x305	R/W
Machine Trap Handling CSRs			
mscratch	Machine Scratch	0x340	R/W
mepc	Machine Trap Program Counter	0x341	R/W
mcause [3]	Machine Trap Cause	0x342	R/W
mtval	Machine Trap Value	0x343	R/W
mip	Machine Interrupt Pending	0x344	R/W
Trigger Module CSRs (shared with Debug Mode)			
tselect	Trigger Select Register	0x7A0	R/W

[1] Although misa is specified as having both read and write access (R/W), its fields are hardwired and thus write has no effect. This is what would be termed WARL (Write Any Read Legal) in RISC-V terminology.

[2] mtvec only provides configuration for trap handling in vectored mode with the base address aligned to 256 bytes.

[3] External interrupt IDs reflected in mcause include even those IDs which have been reserved by RISC-V standard for core internal sources.

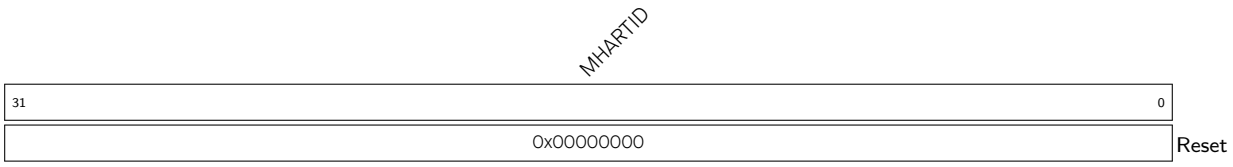
Name	Description	Address	Access
tdata1	Trigger Abstract Data 1	0x7A1	R/W
tdata2	Trigger Abstract Data 2	0x7A2	R/W
Debug Mode CSRs			
dcsr	Debug Control and Status	0x7B0	R/W
dpc	Debug PC	0x7B1	R/W
dscratch0	Debug Scratch Register 0	0x7B2	R/W
dscratch1	Debug Scratch Register 1	0x7B3	R/W
Machine Counter/Timer CSRs			
mcycle	Machine Clock Cycle Counter	0xB00	R/W
minstret	Machine Retired Instruction Counter	0xB02	R/W
mhpmcounter*n*(*n*:3-12)	Machine Performance Monitor Counter	0xB00+*n*	R/W
mcycleh	The higher 32 bits of mcycle	0xB80	R/W
minstreth	The higher 32 bits of minstret	0xB82	R/W
mhpmcounter*n*h(*n*:3-12)	The higher 32 bits of mhpmcounter*n*(*n*:3-12)	0xB80+*n*	R/W
Machine Counter Setup CSR			
mcountinhibit	Machine Counter Control	0x320	R/W

Note that if write, set, or clear operation is attempted on any of the read-only (RO) CSRs indicated in the above table, the CPU will generate an illegal instruction exception.

5.3.2 Registers

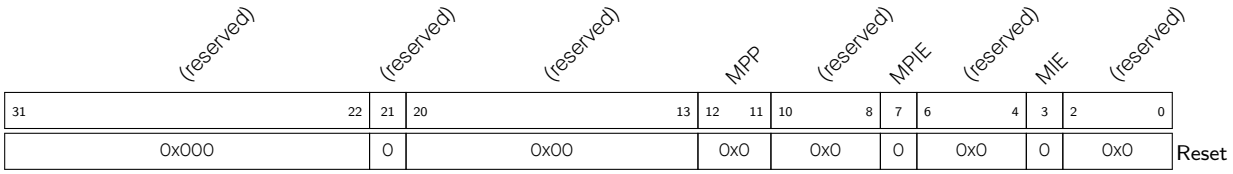
For how to program reserved fields, please refer to Section *Programming Reserved Register Field*.

Register 5.1. mhartid (0xF14)



MHARTID Represents hart ID. The LP CPU hart ID is 0. (RO)

Register 5.2. mstatus (0x300)



MIE Write 1 to enable the global machine mode interrupt. (R/W)

MPIE Write 1 to enable the machine previous interrupt (before trap). (R/W)

MPP Configures machine previous privilege mode (before trap).

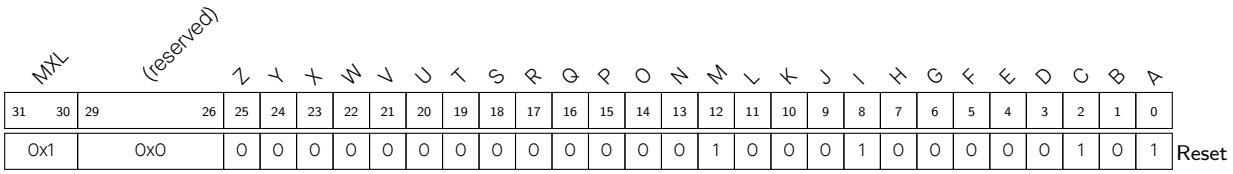
0x3: Machine mode

Other values: Invalid

Note: Only the lower bit is writable. Any write to the higher bit is ignored as it is directly tied to the lower bit.

(R/W)

Register 5.3. misa (0x301)



MXL	(reserved)	Z	Y	X	W	V	U	T	S	R	Q	P	O	N	M	L	K	J	I	H	G	F	E	D	C	B	A
31 30	29 26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x1	0x0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	0	0	0	0	0	1	0	1

Reset

MXL Machine XLEN = 1 (32-bit). (RO)

Z Reserved = 0. (RO)

Y Reserved = 0. (RO)

X Non-standard extensions present = 0. (RO)

W Reserved = 0. (RO)

V Reserved = 0. (RO)

U User mode implemented = 0. (RO)

T Reserved = 0. (RO)

S Supervisor mode implemented = 0. (RO)

R Reserved = 0. (RO)

Q Quad-precision floating-point extension = 0. (RO)

P Reserved = 0. (RO)

O Reserved = 0. (RO)

N User-level interrupts supported = 0. (RO)

M Integer Multiply/Divide extension = 1. (RO)

L Reserved = 0. (RO)

K Reserved = 0. (RO)

J Reserved = 0. (RO)

I RV32I base ISA = 1. (RO)

H Hypervisor extension = 0. (RO)

G Additional standard extensions present = 0. (RO)

F Single-precision floating-point extension = 0. (RO)

E RV32E base ISA = 0. (RO)

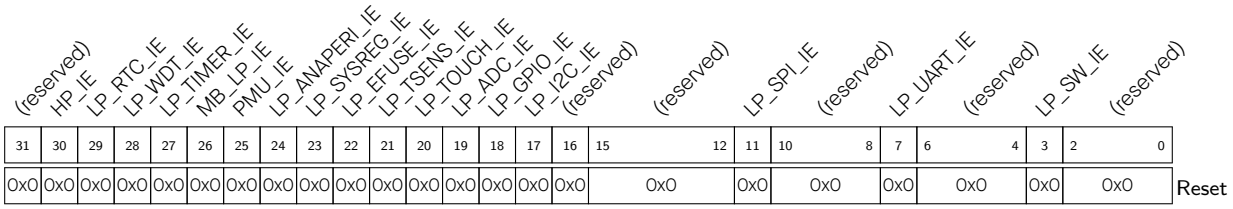
D Double-precision floating-point extension = 0. (RO)

C Compressed Extension = 1. (RO)

B Reserved = 0. (RO)

A Atomic Standard Extension = 1. (RO)

Register 5.4. mie (0x304)



LP_SW_IE Write 1 to enable LP software interrupt. (R/W)

LP_UART_IE Write 1 to enable LP UART interrupt. (R/W)

LP_SPI_IE Write 1 to enable LP SPI interrupt. (R/W)

LP_I2C_IE Write 1 to enable I2C interrupt. (R/W)

LP_GPIO_IE Write 1 to enable LP GPIO interrupt. (R/W)

LP_ADC_IE Write 1 to enable LP ADC interrupt. (R/W)

LP_TOUCH_IE Write 1 to enable LP TOUCH interrupt. (R/W)

LP_TSENS_IE Write 1 to enable LP TSENS interrupt. (R/W)

LP_EFUSE_IE Write 1 to enable LP eFuse interrupt. (R/W)

LP_SYSREG_IE Write 1 to enable SYSREG interrupt. (R/W)

LP_ANAPERI_IE Write 1 to enable ANAPERI interrupt. (R/W)

LP_PMU_IE Write 1 to enable PMU interrupt. (R/W)

LP_MB_IE Write 1 to enable Mailbox interrupt. (R/W)

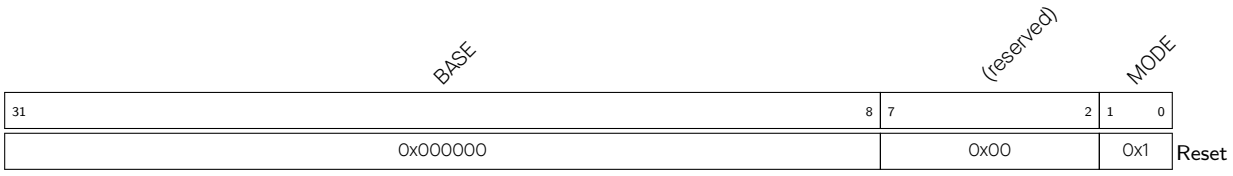
LP_TIMER_IE Write 1 to enable LP Timer interrupt. (R/W)

LP_WDT_IE Write 1 to enable WDT interrupt. (R/W)

LP_RTC_IE Write 1 to enable RTC interrupt. (R/W)

HP_IE Write 1 to enable HP interrupt. (R/W)

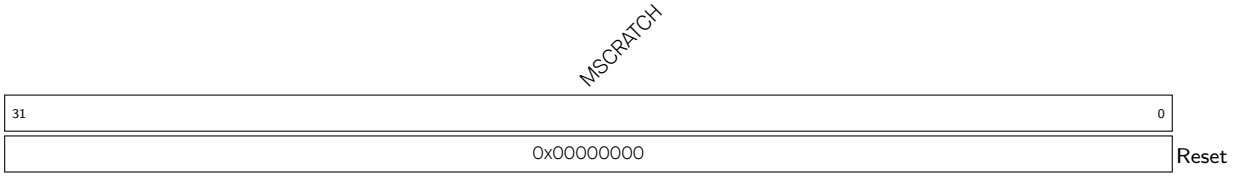
Register 5.5. mtvec (0x305)



MODE Represents whether machine mode interrupts are vectored. Only vectored mode **0x1** is available. (RO)

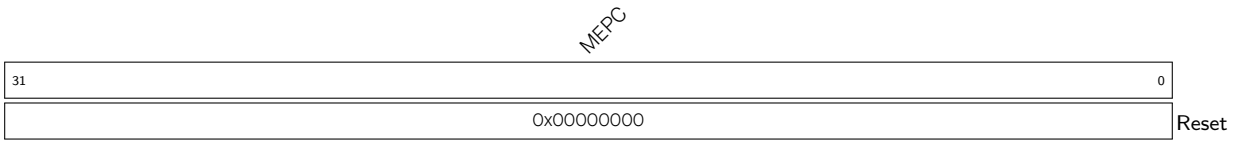
BASE Configures the higher 24 bits of trap vector base address aligned to 256 bytes. (R/W)

Register 5.6. mscratch (0x340)



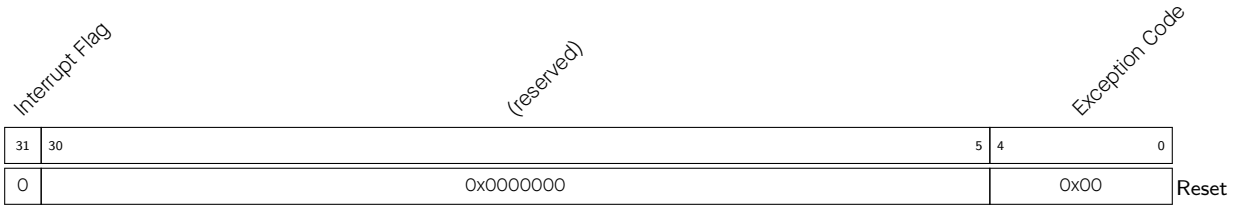
MSCRATCH Contains machine scratch information for custom use. (R/W)

Register 5.7. mepc (0x341)



MEPC Configures the machine trap/exception program counter. This is automatically updated with address of the instruction which was about to be executed while CPU encountered the most recent trap. (R/W)

Register 5.8. mcause (0x342)



Exception Code This field is automatically updated with unique ID of the most recent exception or interrupt due to which CPU entered trap. Possible exception IDs are:

0x2: Illegal instruction

0x3: Hardware breakpoint/watchpoint or EBREAK

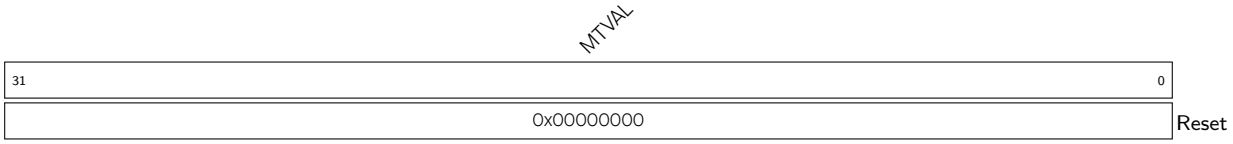
0x6: Misaligned atomic instructions

Note: Exception ID 0x0 (instruction access misaligned) is not present because CPU always masks the lowest bit of the address during instruction fetch.

(R/W)

Interrupt Flag This flag is automatically updated when CPU enters trap. If this is found to be set, it indicates that the latest trap occurred due to an interrupt. For exceptions it remains unset. (R/W)

Register 5.9. mtval (0x343)



MTVAL Configures machine trap value. This is automatically updated with an exception dependent data which may be useful for handling that exception. Data is to be interpreted depending upon exception IDs:

0x1: Faulting address of instruction

0x2: Faulting instruction opcode

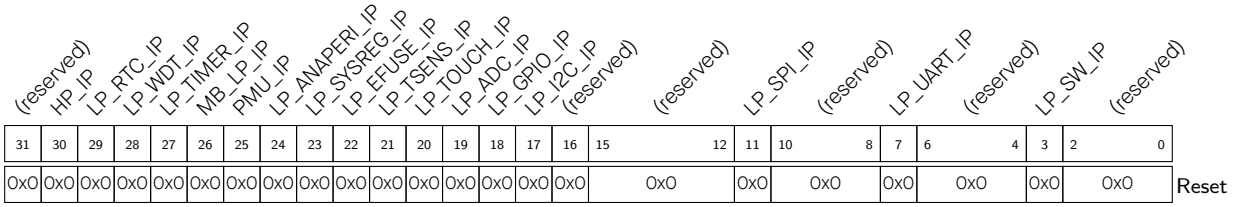
0x5: Faulting data address of load operation

0x7: Faulting data address of store operation

Note: The value of this register is not valid for other exception IDs and interrupts.

(R/W)

Register 5.10. mip (0x344)



LP_SW_IP Configures the pending status of the LP software interrupt.

0□Not pending

1□Pending

(R/W)

LP_UART_IP Configures the pending status of the LP UART interrupt.

0□Not pending

1□Pending

(R/W)

LP_SPI_IP Configures the pending status of the LP SPI interrupt.

0□Not pending

1□Pending

(R/W)

LP_I2C_IP Configures the pending status of the LP I2C interrupt.

0□Not pending

1□Pending

(R/W)

LP_GPIO_IP Configures the pending status of the LP GPIO interrupt.

0□Not pending

1□Pending

(R/W)

LP_ADC_IP Configures the pending status of the LP ADC interrupt.

0□Not pending

1□Pending

(R/W)

LP_TOUCH_IP Configures the pending status of the LP TOUCH interrupt.

0□Not pending

1□Pending

(R/W)

LP_TSENS_IP Configures the pending status of the LP TSENS interrupt.

0□Not pending

1□Pending

(R/W)

LP_EFUSE_IP Configures the pending status of the LP eFuse interrupt.

0□Not pending

1□Pending

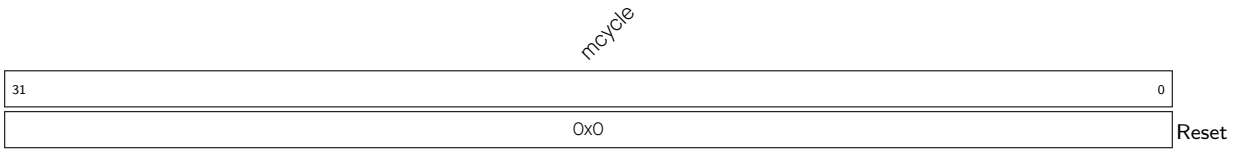
(R/W)

LP_SYSREG_IP Configures the pending status of the LP SYSREG interrupt.

0□Not pending

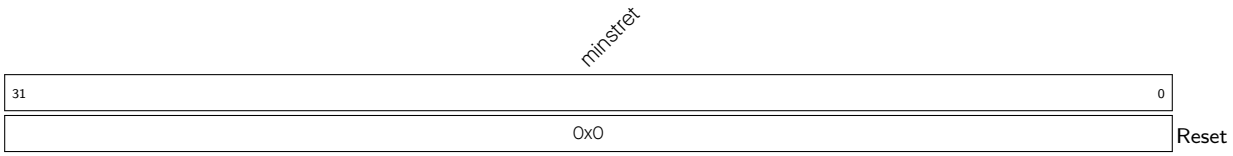
1□Pending

Register 5.11. mcycle (0xB00)



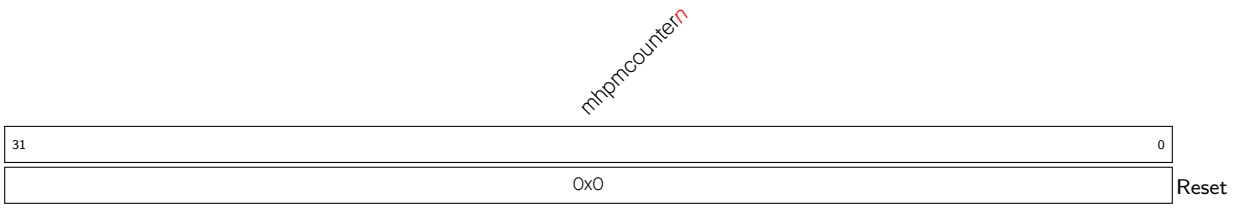
MCYCLE Configures the lower 32 bits of the clock cycle counter. (R/W)

Register 5.12. minstret (0xB02)



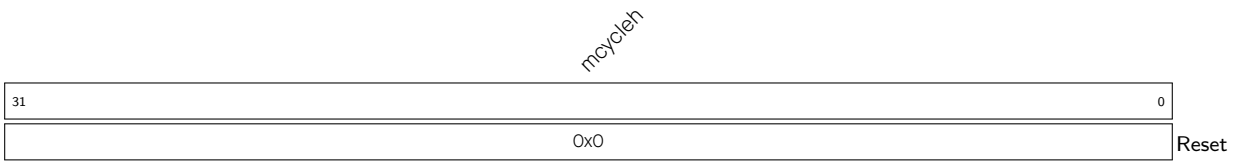
MINSTRET Configures the lower 32 bits of the instruction counter. (R/W)

Register 5.13. mhpmcounter*n*(*n*: 3-12) (0xB00+*n*)



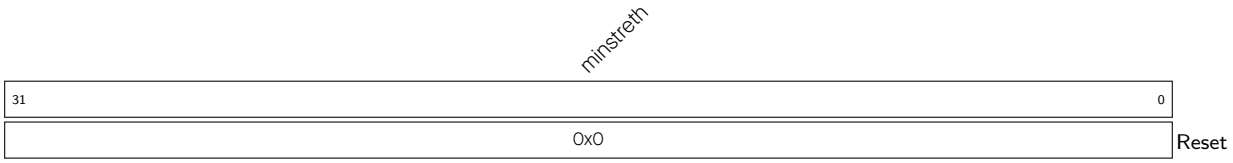
MHPMCOUNTER*n* Configures the lower 32 bits of the performance counter *n*. (R/W)

Register 5.14. mcycleh (0xB80)



MCYCLEH Configures the higher 32 bits of the clock cycle counter. (R/W)

Register 5.15. minstreth (0xB82)



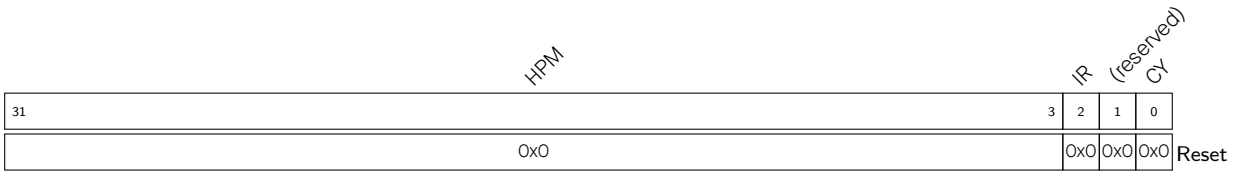
MINSTRETH Configures the higher 32 bits of the instruction counter. (R/W)

Register 5.16. mhpmcounter*n*(*n*: 3-12)h (0xB80+*n*)



MHPMCOUNTER*n*h Configures the higher 32 bits of the performance counter *n*. (R/W)

Register 5.17. mcountinhibit (0x320)



HPM Configures whether the performance counter *n*(*n*:3-12) increments.

0: The counter does not count

1: The counter increments

(R/W)

IR Configure whether the instruction counter increments.

0: The counter does not count

1: The counter increments

(R/W)

CY Configure whether the clock cycle counter increments.

0: The counter does not count

1: The counter increments

(R/W)

5.4 Interrupts and Exceptions

The LP CPU handles interrupts and exceptions according to RISC-V Instruction Set Manual, Volume II: Privileged Architecture, Version 1.10. After entering an interrupt/exception handler, the CPU:

- Saves the current program counter (PC) value to the mepc CSR
- Copies the state of MIE of mstatus to MPIE of mstatus
- Saves the current privileged mode to MPP of mstatus
- Clears MIE of mstatus
- Toggles the privileged mode to machine mode (M mode)
- Jumps to the handler address
  - For exceptions, the handler address is the base address of the vector table in the mtvec CSR
  - For interrupts, the handler address is $mtvec + 4 * ID$, where ID is the interrupt ID. For more information, see Section 5.4.1
- After the mret instruction is executed, the core jumps to the PC saved in the mepc CSR, restores the value of MPIE of mstatus to MIE of mstatus, and restores the privileged mode to that indicated by MPP of mstatus

When the core starts up, the base address of the vector table is initialized to the boot address 0x50000000. After startup, the base address can be changed by writing to the mtvec CSR. For more information about CSRs, see Section 5.3.1.

After a reset, the core starts to fetch instructions from the memory address specified by the LP_SYSTEM_LP_CORE_BOOT_ADDR_REG register with an offset of 0x80. The default value for this register is 0x50100000, which corresponds to the starting address of the LP ROM.

5.4.1 Interrupts

The LP CPU in the ESP32-P4 supports 18 interrupt inputs. Each interrupt corresponds to an ID, which is denoted as n (where n = 3, 7, 11, 16 to 30), and has a specific entry address of $mtvec + 4 * n$. These interrupts are associated with specific peripheral sources. For the mapping between peripheral sources and interrupt IDs, please refer to 5.4-1.

Table 5.4-1. LP CPU Interrupt Mapping

Interrupt ID	Peripheral Source
3	LP_SW_INTR
7	LP_UART_INTR
11	LP_SPI_INTR
17	LP_I2C_INTR
18	LP_GPIO_INTR
19	LP_ADC_INTR
20	LP_TOUCH_INTR
21	LP_TSENS_INTR
22	LP_EFUSE_INTR

Interrupt ID	Peripheral Source
23	LP_SYSREG_INTR
24	LP_ANAPERI_INTR
25	PMU_REG_0_INTR, PMU_REG_1_INTR
26	MB_HP_INTR, MB_LP_INTR
27	LP_TIMER_REG_0_INTR, LP_TIMER_REG_1_INTR
28	LP_WDT_INTR
29	LP_RTC_INTR
30	HP_INTR

The read-only register LPINTR_STATUS_REG indicates the current status of the LP CPU external interrupt sources. For the mapping between this register and the interrupt sources, please refer to the description of this register.

5.4.2 Interrupt Handling

By default, interrupts are disabled globally because the MIE bit in mstatus has a reset value of 0. Software must set this bit to enable global interrupts.

Assuming the interrupt to enable is Interrupt n:

1. Enable Interrupt n:
   - To enable interrupts globally, write 1 to the MIE bit of mstatus.
   - To enable Interrupt n, write 1 to the nth bit of mie CSR.
2. After Interrupt n is enabled, the LP CPU can respond to it. Meanwhile, the corresponding peripheral interrupt should be configured to send an interrupt signal to the LP CPU.
3. After the interrupt is triggered, the LP CPU jumps to the $mtvec + 4 * n$ address.
4. To clear Interrupt n, the peripheral needs to clear the interrupt signal.

5.4.3 Exceptions

The LP CPU supports the RISC-V standard exceptions and can trigger the following exceptions:

Table 5.4-2. LP CPU Exception Causes

Exception ID	Description
1	Instruction access fault
2	Illegal instructions
3	Breakpoints (EBREAK)
5	Load access fault
6	Misaligned atomic instructions
7	Store access fault

5.5 Debugging

This section describes how to debug and test the LP CPU. Debug support is provided through standard JTAG pins and complies with RISC-V External Debug Support Version 0.13.

For ESP32-P4 system debugging overview, please refer to Chapter 2 *High-Performance CPU* > Figure 2.11-1 *Debug System Overview*.

The user interacts with the Debug Host (e.g., laptop), which is running a debugger (e.g., gdb). The debugger communicates with a Debug Translator (e.g., OpenOCD, which may include a hardware driver) to communicate with Debug Transport Hardware (e.g., ESP-Prog adapter). The Debug Transport Hardware connects the Debug Host to the CPU's Debug Transport Module (DTM) through a standard JTAG interface. The DTM provides access to the debug module (DM) using the Debug Module Interface (DMI).

ESP32-P4 features two DTMs interconnected in a daisy-chain configuration. Specifically, TAP0 connects to the HP CPU, while TAP1 is dedicated to the LP CPU.

The HP DM supports multicore debugging according to the specification RISC-V External Debug Support Version 0.13. It can simultaneously control both cores of the HP CPU. In contrast, the LP DM is designed for debugging the single-core LP CPU.

The LP CPU implements four registers for core debugging: dcsr, dpc, dscratch0, and dscratch1. All of those registers can only be accessed from debug mode. If software attempts to access them when the LP CPU is not in debug mode, an illegal instruction exception will be triggered.

5.5.1 Features

The Low-Power CPU debug module has the following debugging features:

- Access to necessary information about the implementation
- CPU core halt and resume
- Read and write of CPU core registers (including CSRs)
- CPU core reset
- CPU halted on software breakpoint (planted breakpoint instruction)
- Hardware single-stepping
- Two hardware triggers (which can be used as breakpoints/watchpoints). See Section 5.6 for details

5.5.2 Functional Description

The debugging mechanism adheres to the specification RISC-V External Debug Support Version 0.13. For a detailed description of the debugging features, refer to the specification.

According to the specification, a hart has the following states: nonexistent, unavail, running, and halted. By default, the LP CPU is in the unavail state. To connect the LP CPU for debugging, users need to clear that state by configuring the LPPERI_CPU_REG register.

5.5.3 Register Summary

The following table lists the debug CSRs supported for the LP CPU.

The abbreviations given in Column **Access** are explained in Section *Access Types for Registers*.

Name	Description	Address	Access
dcsr	Debug Control and Status	0x7B0	R/W
dpc	Debug PC	0x7B1	R/W
dscratch0	Debug Scratch Register 0	0x7B2	R/W
dscratch1	Debug Scratch Register 1	0x7B3	R/W

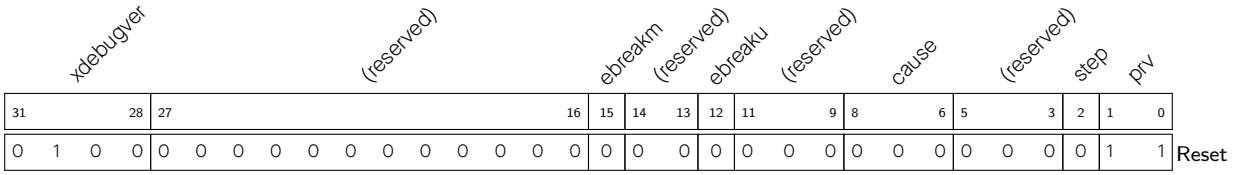
All debug module registers are implemented in accordance with the specification RISC-V External Debug Support Version 0.13. For more information, refer to the specification.

5.5.4 Registers

The following is a detailed description of the debug CSR supported by the LP CPU.

For how to program reserved fields, please refer to Section *Programming Reserved Register Field*.

Register 5.18. dcsr (0x7B0)



xdebugver Represents the debug version.

4: External debug support exists

(RO)

ebreakm Configures execution of the EBREAK instruction in machine mode.

0: Trigger an exception with mcause = 3

1: Enter debug mode

(R/W)

ebreaku Configures execution of the EBREAK instruction in user mode.

0: Trigger an exception with mcause = 3 as described in privileged mode

1: Enter debug mode

(R/W)

cause Represents the reason why debug mode was entered. When there are multiple reasons to enter debug mode in a single cycle, the cause with the highest priority number is the one written.

1: An EBREAK instruction was executed. (priority 3)

2: The Trigger Module caused a halt. (priority 4)

3: haltreq was set. (priority 2)

4: The CPU core single stepped because step was set. (priority 1)

Other values: reserved for future use

(RO)

step When set and not in Debug Mode, the core will only execute a single instruction and then enter Debug Mode.

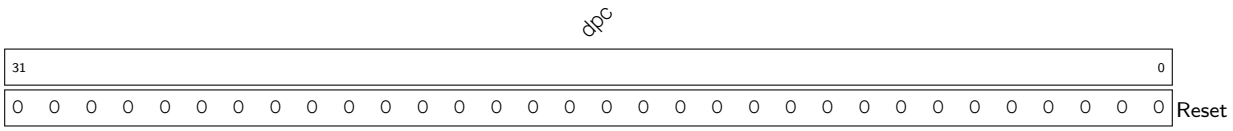
If the instruction does not complete due to an exception, the core will immediately enter Debug Mode before executing the trap handler, with appropriate exception registers set.

Setting this bit does not mask interrupts. This is a deviation from the RISC-V External Debug Support Specification Version 0.13.

(R/W)

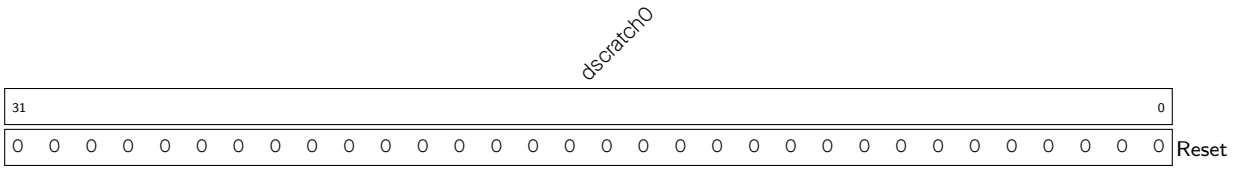
prv Contains the privilege level the core is operating in when debug mode is entered. A debugger can change this value to change the core's privilege level when exiting debug mode. Only **0x3** (machine mode) and **0x0** (user mode) are supported. (RO)

Register 5.19. dpc (0x7B1)



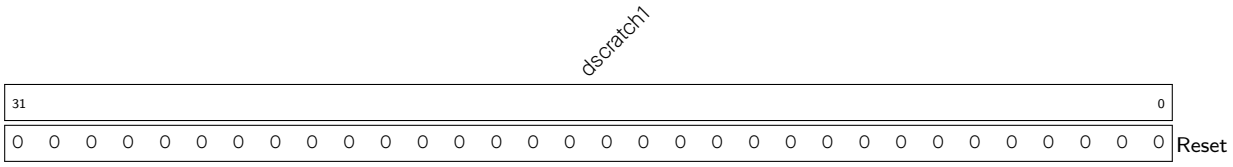
dpc Upon entry to debug mode, dpc is written with the address of the next instruction that will be executed. When resuming, the CPU core's PC is updated to the address stored in dpc. In debug mode, dpc can be modified. This field can be accessed in debug mode. (R/W)

Register 5.20. dscratch0 (0x7B2)



dscratch0 Used by the debug module internally. (R/W)

Register 5.21. dscratch1 (0x7B3)



dscratch1 Used by the debug module internally. (R/W)

5.6 Hardware Trigger

5.6.1 Features

Hardware Trigger module provides breakpoint and watchpoint capability for debugging. It has the following features:

- Two independent trigger units
- Configurable unit to match the address of the program counter
- Able to halt execution and transfer control to the debugger

5.6.2 Functional Description

The hardware trigger module provides three CSRs. See Section 5.6 for details. Among them, tdata1 and tdata2 are abstract CSRs, which means they are shadow registers for accessing internal registers in the trigger units, one at a time.

To select a specific trigger unit, the corresponding number (0-1) needs to be written to the tselect CSR. When a valid value is written, the abstract CSRs, tdata1 and tdata2, automatically match the internal registers of the trigger unit. Each trigger unit has two internal registers, namely mcontrol and maddress, which are mapped to tdata1 and tdata2, respectively.

Writing a value more than 1 to tselect will set tselect to 1.

Since software or debugger may need to know the type of the selected trigger to correctly interpret tdata1 and tdata2, the 4-bit field (31-28) of tdata1 encodes the type of the selected trigger. This type field is read-only and always provides a value of 0x2 for every trigger, which stands for support for address and data matching. Hence, it is inferred that tdata1 and tdata2 are to be interpreted as fields of mcontrol and maddress, respectively. The specification RISC-V External Debug Support Version 0.13 provides information on other possible values, but the trigger module only supports the 0x2 type.

Once a trigger unit has been chosen by writing its index to tselect, it will become possible to configure it by setting the appropriate bits in mcontrol CSR (tdata1) and writing the target address to maddress CSR (tdata2).

5.6.3 Trigger Execution Flow

When a hart is halted and enters debug mode due to the firing of a trigger (action = 1):

- dpc is set to the current PC in the decoding phase
- The cause field in dcsr is set to 2, which means halt due to trigger

5.6.4 Register Summary

Below is a list of Trigger Module CSRs supported by the CPU. These are only accessible from machine mode.

The abbreviations given in Column **Access** are explained in Section *Access Types for Registers*.

Name	Description	Address	Access
tselect	Trigger Select Register	0x7A0	R/W
tdata1	Trigger Abstract Data 1	0x7A1	R/W
mcontrol	tdata1 Shadow Register	0x7A1	R/W
tdata2	Trigger Abstract Data 2	0x7A2	R/W
maddress	tdata2 Shadow Register	0x7A2	R/W

5.6.5 Registers

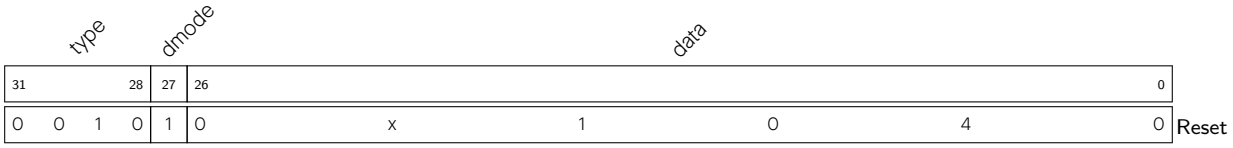
For how to program reserved fields, please refer to Section *Programming Reserved Register Field*.

Register 5.22. tselect (0x7A0)



tselect Configures the index of the selected trigger unit. (R/W)

Register 5.23. tdata1 (0x7A1)



type Represents the trigger type. This field is reserved since only match type (0x2) triggers are supported. (RO)

dmode This is set to 1 if a trigger is being used by the debugger. This field is reserved since it is only supported in debug mode. (RO)

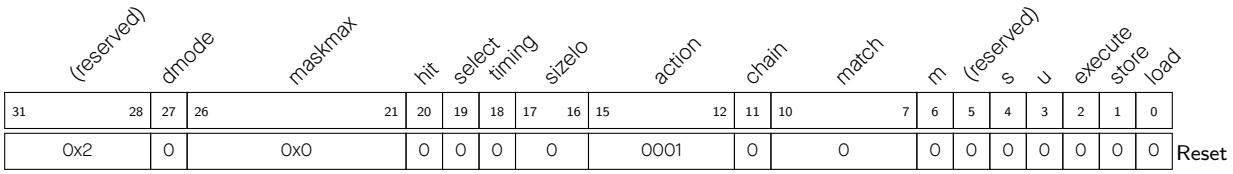
data Configures the abstract tdata1 content. This will always be interpreted as fields of mcontrol since only match type (0x2) triggers are supported. (R/W)

Register 5.24. tdata2 (0x7A2)



tdata2 Configures the abstract tdata2 content. This will always be interpreted as maddress since only match type (0x2) triggers are supported. (R/W)

Register 5.25. mcontrol (0x7A1)



dmode Same as dmode in tdata1. (RO)

maskmax Represents the maximum NAPOT range.

0: A byte. Only exact match is supported.

Other values: Not supported.

(RO)

hit Not implemented in hardware. This field remains 0. (RO)

select Configures to select between an address match or a data match.

0: Perform a match on the virtual address

1: Perform a match on the data value loaded or stored, or the instruction executed

Note: Only address match is implemented. This field remains 0.

(RO)

timing Configures when the trigger will take action.

0: Take action before the instruction is executed

1: Take action after the instruction is executed

Note: The field remains 0.

(RO)

sizelo Only match of any size is supported. This field remains 0. (RO)

action Configure action of the selected trigger after it is triggered.

0x0: Cause a breakpoint exception

0x1: Enter debug mode (Valid only when dmode = 1)

Note: Only entering debug mode is supported. This field remains 1.

(RO)

CHAIN Not implemented in hardware. This field remains 0. (RO)

match Configures the trigger to perform the matching operation of the lower data/instruction address.

0x0: Exact match. Namely, the address corresponding to a certain byte during the access must exactly match the value of maddress.

0x1: NAPOT match. Namely, at least one byte during the access is in the NAPOT region specified in maddress.

Note: Only exact byte match is supported. This field remains 0.

(R/W)

m Set this field to make the selected trigger operate in machine mode. (RO)

s Set this field to make the selected trigger operate in supervisor mode. Operation in supervisor mode is not supported. This field is always 0. (RO)

Continued on the next page...

Register 5.25. mcontrol (0x7A1)

Continued from the previous page...

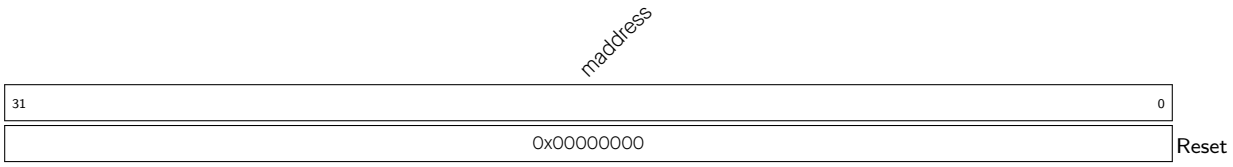
u Set this field to make the selected trigger operate in user mode. Operation in user mode is not supported. This field is always 0. (RO)

execute Configures whether to enable the selected trigger to match the virtual address of instructions.
0: Not enable
1: Enable
(R/W)

store Set this field to make the selected trigger match the virtual address of the memory write operation. Not supported by hardware. This field is always 0. (RO)

load Set this field to make the selected trigger match the virtual address of a memory read operation. Not supported by hardware. This field is always 0. (RO)

Register 5.26. maddress (0x7A2)



maddress Configures the address used by the selected trigger when performing match operation. (R/W)

5.7 Performance Counter

The LP CPU implements one clock cycle counter mcycle(h), one instruction counter minstret(h), and ten event counters mhpmcounter*n*(*n*:3-12). The clock cycle counter and instruction counter are always available and each is 64-bit wide. The performance counters are 40-bit wide each.

By default, all counters are enabled after reset. A counter can be enabled or disabled individually via the corresponding bit in the mcountinhibit CSR.

As shown in Table 5.7-1, each counter is dedicated to counting a particular event.

Table 5.7-1. Performance Counter

Counter	Counted Event
mcycle	Clock cycles
minstret	The number of instructions
mhpmcounter3	Wait cycles for memory access

Counter	Counted Event
mhpmcounter4	Wait cycles for fetching instructions
mhpmcounter5	The number of memory read operations. An unaligned read is counted as two.
mhpmcounter6	The number of memory write operations. An unaligned write is counted as two.
mhpmcounter7	The number of unconditional jump instructions (jal, jr, jalr)
mhpmcounter8	The number of branch instructions
mhpmcounter9	The number of taken branch instructions
mhpmcounter10	The number of compressed instructions
mhpmcounter11	Wait cycles for multiplication instructions
mhpmcounter12	Wait cycles for division instructions

5.8 System Access

The LP CPU has access to both memory and peripherals. When accessing an illegal address, i.e., outside the memory's or peripherals' address space, it will trigger the corresponding access error exception based on the type of access.

5.8.1 Memory Access

The ESP32-P4 LP CPU can access LP ROM, LP SRAM, and L2 SRAM. For more information, please refer to Section 9 *System and Memory*.

- LP ROM: 16 KB region starting from 0x5010_0000 to 0x5010_3FFF. It is used for instruction fetch, data read, etc.
- LP SRAM: 32 KB region starting from 0x5010_8000 ~ 0x5010_FFFF. It is used for instruction fetch, data read, data write, etc.
- L2 SRAM: 768 KB region starting from 0x4FF0_0000 to 0x4FFB_FFFF. It is used for instruction fetch, data read, data write, etc.

Note:
The LP CPU experiences significant latency when accessing L2 SRAM, with each access taking approximately 20 LP CPU clock cycles. However, it can access LP ROM and LP SRAM with no latency.

The LP CPU supports the atomic instruction set. Both the LP CPU and the HP CPU can access memory through atomic instructions, thus achieving atomicity of memory access. For details on the atomic instruction set, please refer to RISC-V Instruction Set Manual Volume I: Unprivileged ISA, Version 2.2.

Note that only L2 SRAM region supports atomic access from HP CPU and LP CPU.

5.8.2 Peripheral Access

Table 9.3-2 in Chapter 9 *System and Memory* lists the peripherals accessible by the LP CPU and their base addresses.

5.9 Event Task Matrix Feature

The LP CPU on ESP32-P4 supports the Event Task Matrix (ETM) function, which allows LP CPU's ETM tasks to be triggered by any peripherals' ETM events, or LP CPU's ETM events to trigger any peripherals' ETM tasks. This section introduces the ETM tasks and events related to the LP CPU. For more information, please refer to Chapter 14 Event Task Matrix (ETM).

LP CPU can receive the following ETM task:

- ULP_TASK_WAKEUP_CPU: Wakes up the LP CPU.
- ULP_TASK_INT_CPU: Triggers an LP CPU interrupt.

LP CPU can generate the following ETM events:

- ULP_EVT_ERR_INTR: Indicates that an LP CPU exception occurs.
- ULP_EVT_START_INTR: Indicates that the LP CPU clock is turned on.
- ULP_EVT_HALT: Indicates that the LP CPU enters the sleep state

5.10 Sleep and Wake-Up Process

5.10.1 Features

Here are the sleep and wake-up features supported by the LP CPU.

- Independent operation, sleeping, and waking-up in the low-power system when the HP CPU is sleeping
- Actively configuring registers to enter the sleep state based on software operating status
- Multiple wake-up sources
- Sleep entry rejection by peripherals

5.10.2 Process

The LP CPU is in sleep by default and its wake-up module follows the process below to wake it up for work and make it sleep.

To configure wake-up sources, please refer to Table 5.10-1.

The first startup of the LP CPU after power-up depends on the wake-up enable and wake-up source configuration by the HP CPU.

- Initialization of the LP CPU
  - Initialize the LP memory.
  - Start the LP CPU. Since the startup of the LP CPU depends on the wake-up process, it is recommended to use the PMU_HP_TRIGGER_LP register to start the initialization of the LP CPU in the following way:
    * Set PMU_LP_CPU_WAKEUP_EN to 0x400000
    * Set PMU_HP_TRIGGER_LP to 0x1

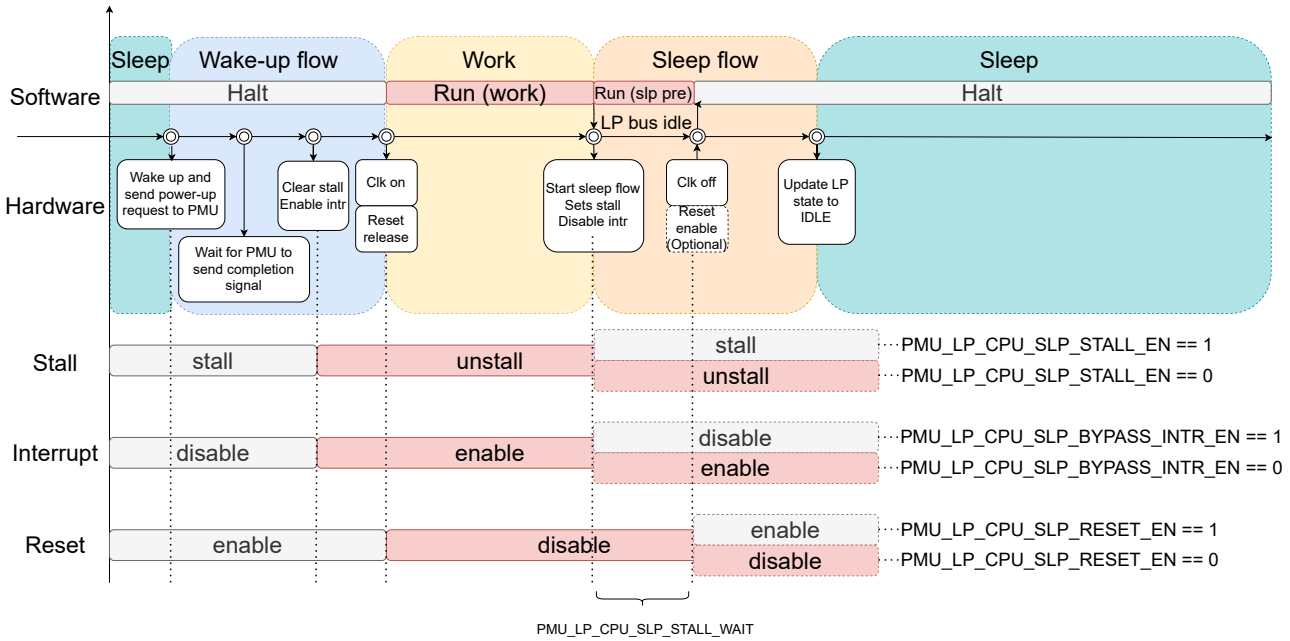


Figure 5.10-1. Wake-Up and Sleep Flow of LP CPU

* The LP CPU will go through the wake-up process to start running

- Wake-up process:
  - The wake-up module receives a wake-up signal and sends a power-up request to the PMU.
  - If the current power consumption state (clock, power supply, etc.) meets the requirements of the LP CPU, the PMU will immediately reply with the completion signal. Otherwise, it will adjust the power consumption state before replying with the completion signal.
  - The wake-up module disables the STALL state of the LP CPU and enables interrupt receiving.
  - The wake-up module starts the clock, releases reset (ignore this step if reset is not enabled for sleep), and starts working.
- Sleep process:
  - The LP CPU configures the PMU_LP_CPU_SLEEP_REQ register to enable the wake-up module to start the sleep process.
  - If PMU_LP_CPU_SLP_STALL_EN is 1, the wake-up module enables the STALL state of the LP CPU. If it is 0, the module does not enable that state. If PMU_LP_CPU_SLP_BYPASS_INTR_EN is 1, the module masks all the LP CPU's interrupts. If it is 0, the module does not mask them.
  - The wake-up module waits for PMU_LP_CPU_SLP_STALL_WAIT LP CPU clock cycles, and then turns off the LP CPU clock. If PMU_LP_CPU_SLP_RESET_EN is 1, the module enables reset of the LP CPU.
  - The wake-up module changes the state of the LP CPU to IDLE and the PMU can enter a deeper level of sleep state.

5.10.3 Wake-Up Sources

Table 5.10-1. Wake Sources

Register Value [1]	Wake-Up Source	Description
BIT(9)	LP IO	The LP CPU can be waken up by the LP IO interrupt status register signal. For more information, please refer to Chapter 10 *GPIO Matrix and IO MUX*.
BIT(10)	LP UART	The LP CPU can be waken up when the LP UART receives a certain number of RX pulses. REG_UART_WAKEUP_EN needs to be enabled. For more information, please refer to Chapter 45 *UART Controller (UART)*.
BIT(13)	RTC timer	The LP CPU can be waken up by the RTC timer target 0 timeout interrupt. For more information, please refer to Chapter 15 *Low-Power Management*.
BIT(14)	Brownout detection	The LP CPU can be waken up when a brownout occurs. For more information, please refer to Chapter 25 *Brown-out Detector*.
BIT(17)	ETM	The LP CPU can be waken up by the ETM task. For more information, please refer to Chapter 14 *Event Task Matrix (ETM)*.
BIT(18)	RTC timer	The LP CPU can be waken up by the RTC timer target 0 timeout interrupt. For more information, please refer to Chapter 15 *Low-Power Management*.
BIT(19)	LP I2S audio	The LP CPU can be waken up when the predefined audio is received via I2S. For more details, please refer to Chapter 49 *I2S Controller (I2S)*.
BIT(22)	PMU_HP_TRIGGER_LP register	The HP CPU sets the register PMU_HP_TRIGGER_LP to wake up the LP CPU, and sets REG_HP_SW_TRIGGER_INT_CLR to clear this wake-up source.

[1] Value of the PMU_LP_CPU_WAKEUP_EN register

5.10.4 Sleep Rejection

When the LP CPU is preparing to enter the sleep state, it can be rejected by peripheral events. In other words, peripheral events have the capability to prevent the LP CPU from entering sleep mode. Below is the configuration process for the rejection:

1. The LP CPU configures PMU_SLP_REJECT_EN to enable the functionality that allows peripherals to reject the LP CPU from entering the sleep mode.
2. The LP CPU configures PMU_SLEEP_REJECT_ENA to enable the corresponding peripheral event to prevent the LP CPU from entering the sleep state.

The peripheral sources that can prevent the LP CPU from sleeping are the same as the wake-up sources.

Part II

System DMA

Focused on managing and transferring data within the system, this part explores controllers for Direct Memory Access (DMA), providing guidelines for efficient data movement supporting various peripherals and devices.

Chapter 6

GDMA Controller (GDMA-AHB, GDMA-AXI)

6.1 Overview

General Direct Memory Access (GDMA) is a feature that allows peripheral-to-memory, memory-to-peripheral, and memory-to-memory data transfer at high speed. The CPU is not involved in the GDMA transfer and therefore is more efficient with less workload.

ESP32-P4 has two types of general-purpose DMA controllers, namely GDMA-AHB and GDMA-AXI, to directly access the AHB bus or the AXI bus respectively. Both GDMA-AHB and GDMA-AXI have six independent channels, i.e., three transmit channels and three receive channels.

The GDMA-AHB channels are shared and can be assigned to I3C, UHCI, I2S, ADC, or RMT to access internal and external memory.

The GDMA-AXI channels are also shared and can be assigned to LCD, CAM, two general-purpose SPIs, PARLIO, AES, or SHA to access internal and external memory.

GDMA-AHB and GDMA-AXI use configurable priority and weight arbitration schemes to manage peripherals' needs for bandwidth.

Unless otherwise specified, the term "GDMA controller" refers to both GDMA-AHB and GDMA-AXI in the following text.

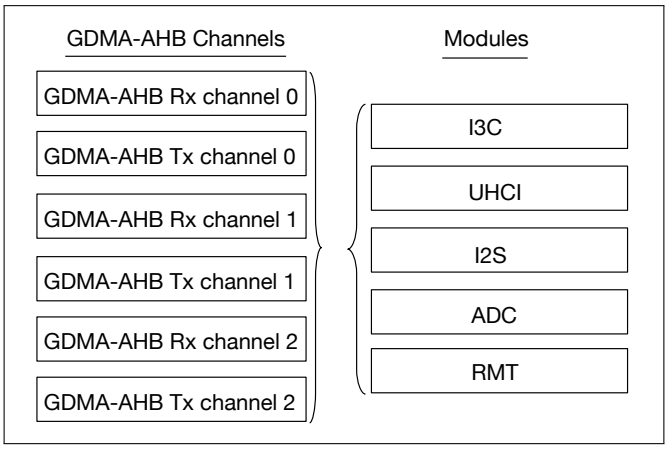


Figure 6.1-1. Modules that Share GDMA-AHB Channels

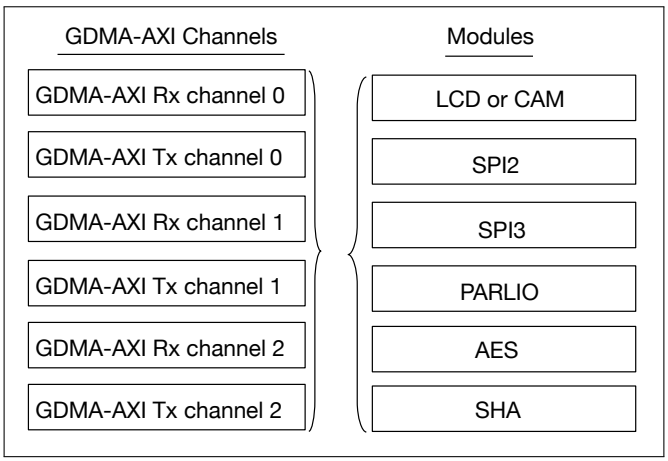


Figure 6.1-2. Modules that Share GDMA-AXI Channels

6.2 Features

GDMA-AHB and GDMA-AXI have the following features:

- Architecture:
  - GDMA-AHB: AHB bus architecture
  - GDMA-AXI: AXI bus architecture, which gives the possibility to complete up to 8 transactions out of order and up to 8 outstanding transactions
- Programmable length of data to be transferred in bytes
- Access via any address and size
- Alignment:
  - GDMA-AHB:
    * Descriptor address: 1-word aligned
    * Data address and length:
      · Internal memory and non-encrypted external memory address space: no requirements
      · Encrypted external memory address space: 16-byte aligned
  - GDMA-AXI:
    * Descriptor address: 2-word aligned
    * Data address and length:
      · Internal memory and non-encrypted external memory address space: no requirements
      · Encrypted external memory address space: 16-byte aligned
- Linked list of descriptors
- When accessing memory, GDMA-AHB supports INCR4, INCR8, or INCR16 burst transfer, and GDMA-AXI supports INCR burst transfer
- Three transmit channels and three receive channels for each controller

- Software-configurable selection of peripheral requesting its service
- Configurable channel priority and weight arbitration
- Support for memory transfer
- CRC calculation of data
- Interrupt response mechanism for GDMA-AXI linked list switching

6.3 Architecture

In ESP32-P4, all modules that need high-speed data transfer support GDMA. The GDMA controller and CPU data bus have access to the same address space in memory. Figure 6.3-1 shows the basic architecture of the GDMA controller.

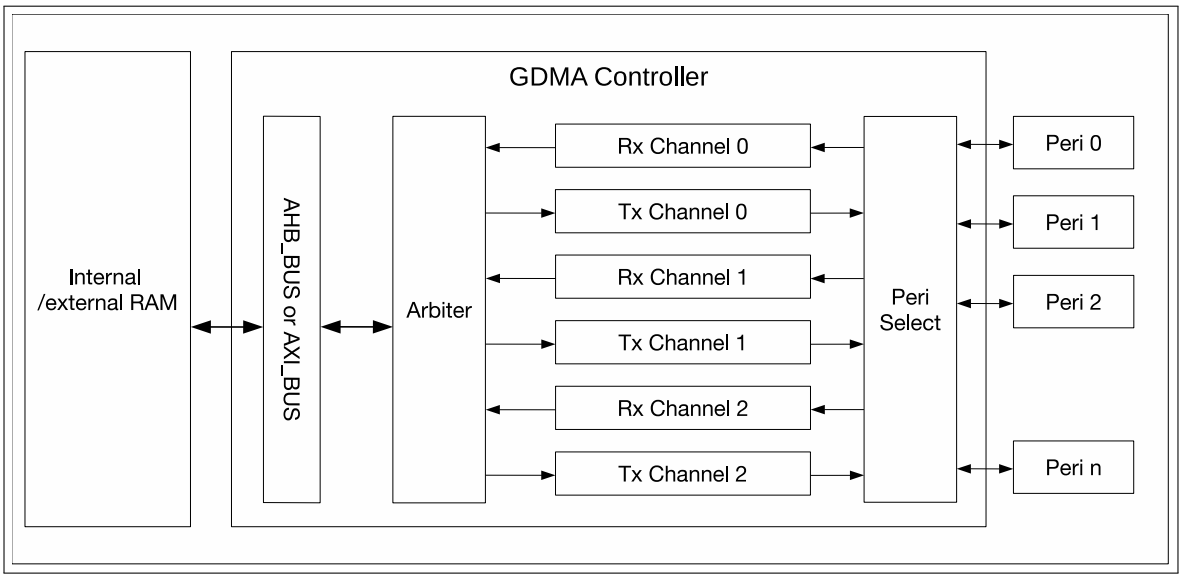


Figure 6.3-1. GDMA controller Architecture

GDMA-AHB and GDMA-AXI have six independent channels respectively, i.e., three transmit channels and three receive channels. Every channel can be connected to different peripherals. In other words, channels are general-purpose, and shared by peripherals.

GDMA-AHB and GDMA-AXI read data from or write data to internal and external memory via AHB_BUS or AXI_BUS respectively. Before this, GDMA uses configurable arbitration schemes for channels requesting read or write access. For the available address range of internal and external memory, please see Chapter 9 *System and Memory*.

Software can use the GDMA through linked lists. GDMA-AHB linked lists are stored in internal memory, while GDMA-AXI linked lists are stored in internal or external memory. These linked lists consist of outlink*n* and inlink*n*, where *n* indicates the channel number (ranging from 0 to 2). The GDMA reads an outlink*n* (i.e., a linked list of transmit descriptors) from memory and transmit data in corresponding memory according to the outlink*n*, or read an inlink*n* (i.e., a linked list of receive descriptors) and store received data into specific address space in memory according to the inlink*n*.

6.4 Functional Description

6.4.1 Linked List

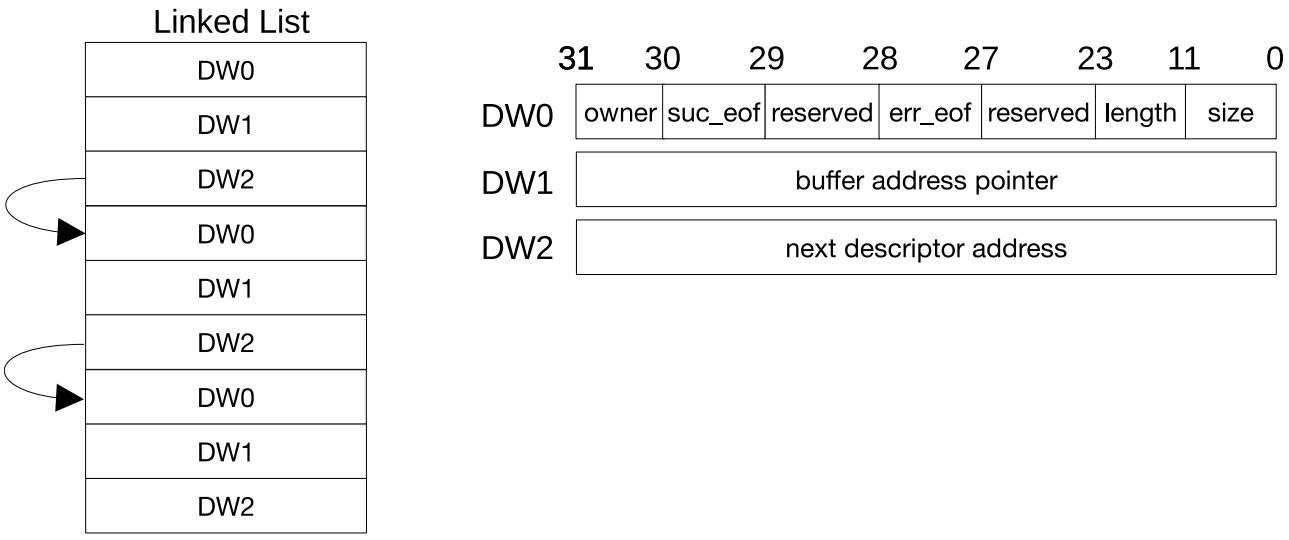


Figure 6.4-1. Structure of a Linked List

Figure 6.4-1 shows the structure of a linked list. An outlink and an inlink have the same structure. A linked list is formed by one or more descriptors, and each descriptor consists of three words. Linked lists should be stored in the memory for the GDMA to be able to use them. The meanings of a descriptor's fields are as follows:

- owner (DW0) [31]: Specifies who is allowed to access the buffer that this descriptor points to.
  0: CPU can access the buffer.
  1: The GDMA controller can access the buffer.
  When the GDMA controller stops using the buffer, this bit in a receive descriptor is automatically cleared by hardware, while this bit in a transmit descriptor can only be automatically cleared by hardware if AHB/AXI_DMA_OUT_AUTO_WRBACK_CH*n* is set to 1. Software can disable automatic clearing by hardware by setting the AHB/AXI_DMA_OUT_LOOP_TEST_CH*n* or AHB/AXI_DMA_IN_LOOP_TEST_CH*n* bit. When software loads a linked list, this bit should be set to 1.
  Note: AHB/AXI_DMA_OUT is the prefix of transmit channel registers, and AHB/AXI_DMA_IN is the prefix of receive channel registers.
- suc_eof (DW0) [30]: Specifies whether the AHB/AXI_DMA_IN_SUC_EOF_CH*n*_INT or AHB/AXI_DMA_OUT_EOF_CH*n*_INT interrupt will be triggered when the data corresponding to this descriptor has been received or transmitted.
  0: No interrupt will be triggered after the current descriptor's successful transfer;
  1: An interrupt will be triggered after the current descriptor's successful transfer.
  For receive descriptors, software needs to clear this bit to 0, and hardware will set it to 1 after receiving data containing the EOF flag.
  For transmit descriptors, software needs to set this bit to 1 as needed.
  If software configures this bit to 1 in a descriptor, the GDMA will include the EOF flag in the data sent to the corresponding peripheral, indicating to the peripheral that this data segment marks the end of one

transfer phase.

- reserved (DW0) [29]: Reserved. The value of this bit does not matter.
- err_eof (DW0) [28]: Specifies whether the received data has errors.
  0: The received data does not have errors.
  1: The received data has errors.
  This bit is used only when UHCI or PARLIO uses the GDMA to receive data. When an error is detected in the received data segment corresponding to a descriptor, this bit in the receive descriptor is set to 1 by hardware.
- reserved (DW0) [27:24]: Reserved.
- length (DW0) [23:12]: Specifies the number of valid bytes in the buffer that this descriptor points to. This field in a transmit descriptor is written by software and indicates how many bytes can be read from the buffer; this field in a receive descriptor is written by hardware automatically and indicates how many valid bytes have been stored in the buffer.
- size (DW0) [11:0]: Specifies the size of the buffer that this descriptor points to.
- buffer address pointer (DW1): Address of the buffer.
- next descriptor address (DW2): Address of the next descriptor. If the current descriptor is the last one, this value is 0. For GDMA-AHB, this field can only point to the internal memory space; for GDMA-AXI, this field can point to the internal or external memory space.

> **Note:**
> Please note that addresses of GDMA-AHB descriptors should be 4-byte aligned, and addresses of GDMA-AXI descriptors should be 8-byte aligned.

If the length of data received is smaller than the size of the buffer, the GDMA controller will not use the available space of the buffer in the next transaction.

6.4.2 Peripheral-to-Memory and Memory-to-Peripheral Data Transfer

The GDMA controller can transfer data from memory to peripheral (transmit) and from peripheral to memory (receive). A transmit channel transfers data in the specified memory location to a peripheral's transmitter via an outlink*n*, whereas a receive channel transfers data received by a peripheral to the specified memory location via an inlink*n*.

Every transmit and receive channel can be connected to any peripheral with the GDMA feature. Table 6.4-1 and Table 6.4-2 illustrate how to select the peripheral to be connected via registers. "Dummy-*n*" corresponds to register values for memory-to-memory data transfer. When a channel is connected to a peripheral, the rest of the channels cannot be connected to that peripheral.

Table 6.4-1. GDMA-AHB Selecting Peripherals via Register Configuration

AHB_DMA_PERI_IN_SEL_CH*n* AHB_DMA_PERI_OUT_SEL_CH*n*	Peripheral
0	I3C
1	Dummy-1

2	UHCI
3	I2S0
4	I2S1
5	I2S2
6 ~ 7	Dummy-6 ~ 7
8	ADC
9	Dummy-9
10	RMT
11 ~ 15	Dummy-11 ~ 15
16 ~ 63	Invalid

Table 6.4-2. GDMA-AXI Selecting Peripherals via Register Configuration

AXI_DMA_PERI_IN_SEL_CH*n* AXI_DMA_PERI_OUT_SEL_CH*n*	Peripheral
0	LCD or CAM
1	SPI2
2	SPI3
3	PARLIO
4	AES
5	SHA
6 ~ 15	Dummy-6 ~ 15
16 ~ 63	Invalid

6.4.3 Memory-to-Memory Data Transfer

The GDMA controller also allows memory-to-memory data transfer. Such data transfer can be enabled by setting AHB/AXI_DMA_MEM_TRANS_EN_CH*n*, which connects the output of transmit channel *n* to the input of receive channel *n*. Note that a transmit channel is only connected to the receive channel with the same number (*n*), and AHB/AXI_DMA_PERI_IN_SEL_CH*n* and AHB/AXI_DMA_PERI_OUT_SEL_CH*n* should be configured to the same value corresponding to "Dummy".

GDMA-AHB only supports memory-to-memory data transfer on channels 1 and 2. The AHB_DMA_PERI_IN_SEL_CH*n* and AHB_DMA_PERI_OUT_SEL_CH*n* (*n* = 1 ~2) fields need to be configured to the same value corresponding to "Dummy", but the ID captured by AHB_DMA_AHBINF_RESP_ERR_ID is a fixed value. When using GDMA-AHB channel 1, the ID is 11; when using GDMA-AHB channel 2, the ID is 12.

6.4.4 Enabling GDMA

The software uses the GDMA controller through linked lists. When the GDMA controller receives data, software loads an inlink, configures the AHB/AXI_DMA_INLINK_ADDR_CH*n* field with the address of the first receive descriptor, and sets the AHB/AXI_DMA_INLINK_START_CH*n* bit to enable GDMA. When the GDMA controller transmits data, software loads an outlink, prepares data to be transmitted, configures the AHB/AXI_DMA_OUTLINK_ADDR_CH*n* field with address of the first transmit descriptor, and sets the AHB/AXI_DMA_OUTLINK_START_CH*n* bit to enable GDMA. The AHB/AXI_DMA_INLINK_START_CH*n* bit and AHB/AXI_DMA_OUTLINK_START_CH*n* bit are cleared automatically by hardware.

In some cases, you may want to append more descriptors to a DMA transfer that is already started. Naively, it would seem to be possible to do this by clearing the EOF bit of the final descriptor in the existing list and setting its next descriptor address pointer field (DW2) to the first descriptor of the to-be-added list. However, this strategy fails if the existing DMA transfer is almost or entirely finished. Instead, the GDMA controller has specialized logic to make sure a DMA transfer can be continued or restarted: if the transfer is ongoing, the controller will make sure to take the appended descriptors into account; if the transfer has already finished, the controller will restart with the new descriptors. This is implemented by the Restart function.

When using the Restart function, software needs to rewrite the address of the first descriptor in the new list to DW2 of the last descriptor in the loaded list, and set AHB/AXI_DMA_INLINK_RESTART_CH*n* bit or AHB/AXI_DMA_OUTLINK_RESTART_CH*n* bit (these two bits are cleared automatically by hardware). As shown in Figure 6.4-2, by doing so hardware can obtain the address of the first descriptor in the new list when reading the last descriptor in the loaded list, and then read the new list.

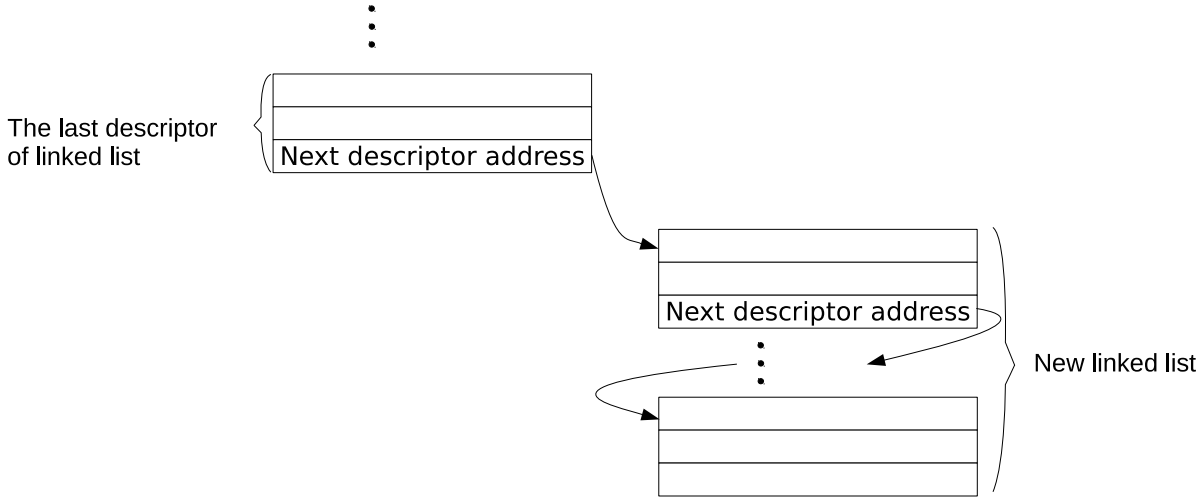


Figure 6.4-2. Relationship among Linked Lists

6.4.5 Linked List Reading Process

Once configured and enabled by software, the GDMA controller starts to read the linked list from memory. The GDMA performs checks on descriptors in the linked list. Only if descriptors pass the checks, the corresponding GDMA channel will start data transfer. If the descriptors fail any of the checks, hardware will trigger descriptor error interrupt (either AHB/AXI_DMA_IN_DSCR_ERR_CH*n*_INT or AHB/AXI_DMA_OUT_DSCR_ERR_CH*n*_INT), and the channel will halt.

The checks performed on descriptors are:

- Owner bit check when AHB/AXI_DMA_IN_CHECK_OWNER_CH*n* or AHB/AXI_DMA_OUT_CHECK_OWNER_CH*n* is set to 1. If the owner bit is 0, the buffer is accessed by the CPU. In this case, the owner bit fails the check. The owner bit will not be checked if AHB/AXI_DMA_IN_CHECK_OWNER_CH*n* or AHB/AXI_DMA_OUT_CHECK_OWNER_CH*n* is 0.
- Descriptor address check, which checks if the descriptor address is located in configured memory space. If a GDMA-AHB descriptor points to AHB_DMA_ACCESS_INTR_MEM_START_ADDR ~ AHB_DMA_ACCESS_INTR_MEM_END_ADDR, it passes the check. If a GDMA-AXI descriptor points to AXI_DMA_ACCESS_INTR_MEM_START_ADDR ~ AXI_DMA_ACCESS_INTR_MEM_END_ADDR, or AXI_DMA_ACCESS_EXTR_MEM_START_ADDR ~ AXI_DMA_ACCESS_EXTR_MEM_END_ADDR, it passes

the check. For details, please refer to Section 6.4.8.

After the software detects a descriptor error interrupt, it must reset the corresponding channel, and enable GDMA by setting AHB/AXI_DMA_OUTLINK_START_CH*n* or AHB/AXI_DMA_INLINK_START_CH*n* bit.

6.4.6 Interrupt Response Mechanism for GDMA-AXI Linked List Switching

GDMA-AXI supports generating an interrupt when a linked list switching is completed. Software can set the AXI_DMA_LINK_SWITCH_STATE_CH*n* bit to 1 to enable the linked list switching. Once the linked list switching is completed, if the AXI_DMA_OUT_LINK_SWITCH_CH*n*_INT interrupt is enabled, GDMA-AXI will generate this interrupt, and clear the AXI_DMA_LINK_SWITCH_STATE_CH*n* bit, indicating that the linked list has been switched.

This feature serves only as a status indication and does not affect normal GDMA-AXI data transfers.

6.4.7 EOF

Note: In this chapter, EOF of transmit descriptors refers to suc_eof (i.e., bit 30 of DW0), while EOF of receive descriptors refers to both suc_eof and err_eof (i.e., bit 28 of DW0).

The GDMA controller uses EOF (end of frame) flags to indicate the end of data segment transfer corresponding to a specific descriptor.

For data transmission, the GDMA generates two types of EOF interrupts:

- AHB/AXI_DMA_OUT_EOF_CH*n*_INT, generated when the suc_eof bit of any descriptor in the linked list is set, and the data corresponding to this descriptor has been transmitted. This interrupt is enabled by setting the AHB/AXI_DMA_OUT_EOF_CH*n*_INT_ENA bit.
- AHB/AXI_DMA_OUT_TOTAL_EOF_CH*n*_INT, generated when the suc_eof bit of the last descriptor in the linked list is set, and the data corresponding to the last descriptor has been transmitted. This interrupt is enabled by setting the AHB/AXI_DMA_OUT_TOTAL_EOF_CH*n*_INT_ENA bit.

For data reception, the GDMA also generates two types of EOF interrupts:

- AHB/AXI_DMA_IN_SUC_EOF_CH*n*_INT, generated when a data segment with an EOF flag has been received. This interrupt is enabled by setting the AHB/AXI_DMA_IN_SUC_EOF_CH*n*_INT_ENA bit.
- (UHCI and PARLIO only) AHB/AXI_DMA_IN_ERR_EOF_CH*n*_INT, generated when a data segment corresponding to a descriptor has been received with errors. This interrupt is enabled by setting the AHB/AXI_DMA_IN_ERR_EOF_CH*n*_INT_ENA bit, and is valid only when the channel is connected to UHCI or PARLIO.

When detecting an AHB/AXI_DMA_OUT_TOTAL_EOF_CH*n*_INT or AHB/AXI_DMA_IN_SUC_EOF_CH*n*_INT interrupt, software can read the value of the AHB/AXI_DMA_OUT_EOF_DES_ADDR_CH*n* or AHB/AXI_DMA_IN_SUC_EOF_DES_ADDR_CH*n* field, which stores the address of the finished descriptor. Therefore, the software can tell which descriptors have been used and reclaim them as needed. For RX, the address of the descriptor can also be stored to the AHB/AXI_DMA_IN_ERR_EOF_DES_ADDR_CH*n* field when an AHB/AXI_DMA_IN_ERR_EOF_CH*n*_INT interrupt is triggered.

6.4.8 Accessing Memory

The transmit and receive channels of GDMA-AHB can access the internal and external memory space configured by AHB_DMA_ACCESS_INTR_MEM_START_ADDR and AHB_DMA_ACCESS_INTR_MEM_END_ADDR.

The transmit and receive channels of GDMA-AXI can access:

- Internal memory space configured by AXI_DMA_ACCESS_INTR_MEM_START_ADDR and AXI_DMA_ACCESS_INTR_MEM_END_ADDR.
- External memory space configured by AXI_DMA_ACCESS_EXTR_MEM_START_ADDR and AXI_DMA_ACCESS_EXTR_MEM_END_ADDR.

To improve data transfer efficiency, GDMA can send data in burst mode. For GDMA-AHB, this mode is disabled by default, and can be enabled for receive channels by setting AHB_DMA_IN_DATA_BURST_EN_CH*n*, and enabled for transmit channels by setting AHB_DMA_OUT_DATA_BURST_EN_CH*n*. For GDMA-AXI, this mode is enabled by default, and cannot be disabled. Users can configure the number of data bytes in a single AXI burst (burst length) to 8, 16, 32, 64, or 128 bytes via AXI_DMA_IN_BURST_SIZE_SEL_CH*n* and AXI_DMA_OUT_BURST_SIZE_SEL_CH*n*.

When GDMA-AHB and GDMA-AXI access the configured internal memory and non-encrypted external memory space, there are no requirements for descriptor field alignment; when they access the encrypted external memory space (see Chapter 34 *External Memory Encryption and Decryption (XTS_AES)*), buffer address pointer and data should be 16-byte aligned, and other fields are not required to be aligned.

6.4.9 Arbitration

To ensure timely response to peripherals running at a high speed with low latency (such as SPI), the GDMA controller implements two arbitration schemes, based on channel priority and channel weight respectively.

- Priority arbitration: Each channel can be assigned a priority from 0 ~ 5 (in total 6 levels). The larger the number, the higher the priority. When several channels perform data transfers at the same time, the GDMA would respond to the transfer requests according to priority levels, and the channel with higher priority would get a response more timely.
- Weight arbitration:
  - Each channel is assigned a weight (i.e., the number of tokens) from 0 ~ 15. The GDMA divides the AHB or AXI bus clock period into multiple time slots, and in each time slot, the number of transfers performed by a channel is determined by the number of its tokens. Every time a channel performs a data transfer, one of its tokens is spent. If all of its tokens have been spent in a time slot, then this channel's transfer request will no longer be responded to until the time slot expires, or until tokens of all channels have been spent and the new time slot starts.
  - If the number of tokens assigned to a channel is not zero, the GDMA waits for this channel's tokens to be spent even if it does not have data transfer requests, and will not exit from this time slot ahead of expiration. This leads to a waste of bandwidth. Therefore, the GDMA provides weight arbitration optimization. When this feature is enabled, a channel without data transfer requests will not be involved in the arbitration and its tokens will be ignored. When all channels no longer have transfer

requests, the GDMA arbiter exits from the current time slot before expiration. This way, the arbitration response is accelerated and the transfer efficiency is improved.

Note:

- If channels have the same weight but different priorities, on condition that the number of bytes to be transferred is the same and not large, channels with higher priorities would finish data transfers first.
- If channels have different priorities and weights, on condition that the number of bytes to be transferred is the same, channels with higher weights would be allocated more bandwidth and finish data transfers first.

6.4.10 Bus Error Response Information Logging

During GDMA-AHB data transfers, if a slave device on the bus returns an error response, the data transfer will not be interrupted; GDMA-AHB will continue to transfer data according to the existing linked list. As a result, the actual data received after the transfer may not match the expected data. To facilitate troubleshooting, if an error response is received during GDMA-AHB data transfer, the following internal registers can be used to store relevant transfer information:

- AHB_DMA_AHBINF_RESP_ERR_ADDR: Records the 32-bit address corresponding to the current bus transfer.
- AHB_DMA_AHBINF_RESP_ERR_CH_ID: Records the GDMA-AHB channel used for the current bus transfer.
- AHB_DMA_AHBINF_RESP_ERR_ID: Records the transfer ID corresponding to the current peripheral-to-memory, memory-to-peripheral, or memory-to-memory transfer on the bus.
- AHB_DMA_AHBINF_RESP_ERR_WR: Records the type of transfer for the current bus transfer.

When GDMA-AHB initiates an AHB bus transfer as the master and receives an error response, it will also generate an error response interrupt to notify the CPU. After the interrupt is generated, GDMA-AHB will not interrupt the data transfer. In addition, GDMA-AHB does not latch data into registers for the first AHB bus transfer that triggers an error response. If GDMA-AHB receives multiple consecutive error responses, the error response logging registers always reflect the most recent error; only the last error response prior to reading the registers is retained.

6.4.11 Automatic Clock Gating

When GDMA-AHB is operating, not all channels are needed in certain scenarios; in this case, the clocks for unused channels can be disabled to reduce power consumption. Similarly, the internal arbitration and bus interface modules only need to operate during data transfer, and their clocks can also be disabled during idle periods to save power. Currently, GDMA-AHB has the following internal modules:

- Bus interface processing module
- Arbitration module
- RX data processing module
- RX descriptor processing module
- TX data processing module

- TX descriptor processing module
- Register configuration synchronization module

Clocks of above modules are automatically disabled when idle to further minimize power consumption. These clocks can also be forced to enable by setting corresponding register bits. For example, setting AHB_DMA_AHBINF_CLK_EN will keep the bus interface processing module clock always on; the configurations for keeping other module clocks always on are similar.

6.4.12 CRC Calculation

The GDMA has a CRC calculation module for both TX and RX. This module supports arbitrary polynomials and 8-bit parallel computation, namely 8 bits at a time. The CRC result is up to 32-bit wide for GDMA-AHB, and up to 16-bit for GDMA-AXI. It is calculated for each linked list descriptor by descriptor and becomes invalid when the Restart function is triggered during transfer.

For easier understanding of register configuration rules, assume:

- The 8-bit data to calculate each time is data[7:0].
- The required CRC result is 5-bit wide, and the polynomial used is $crc[4:0] = x \oplus 5 + x \oplus 2 + 1$;
- The initial CRC value is 0x1F, and the intermediate result calculated in each clock cycle is crc_tmp[4:0].

Taking as an example the GDMA reading data from memory and transmitting the data to peripherals, the following fields are provided by hardware for CRC computation:

- AHB/AXI_DMA_OUT_CRC_INIT_DATA_CH*n*: Configures the initial CRC value, i.e., 0x1F in the above example. AHB_DMA_OUT_CRC_INIT_DATA_CH*n* can be configured with a 32-bit value at the maximum, while AXI_DMA_OUT_CRC_INIT_DATA_CH*n* can be configured with a 16-bit value at the maximum, meaning that the CRC result could be 32 bits or 16 bits at most.
- AHB/AXI_DMA_OUT_CRC_FINAL_RESULT_CH*n*: Stores the computation result of all data in a DMA transfer (until data pointed by the last descriptor in the linked list has been transferred). The computation result could be 32 bits at most.
- AHB/AXI_DMA_TX_CRC_WIDTH_CH*n*: Configures the CRC result length.
- AHB/AXI_DMA_TX_CRC_LAUTCH_FLGA_CH*n*, which latches the CRC result.
- AHB/AXI_DMA_TX_CRC_EN_WR_DATA_CH*n*: Indicates which bits in crc_tmp[4:0] are used to compute crc[x] (x is configured via AHB/AXI_DMA_TX_CRC_EN_ADDR_CH*n*). If AHB/AXI_DMA_TX_CRC_EN_WR_DATA_CH*n* is configured as 0xF, it means that only crc_tmp[3], crc_tmp[2], crc_tmp[1], and crc_tmp[0] are used for computation, and other bits are not used (in this case crc_tmp[4]).
- AHB/AXI_DMA_TX_CRC_DATA_EN_WR_DATA_CH*n*: Indicates which bits in data[7:0] are used to compute crc[x] (x is configured via AHB/AXI_DMA_TX_CRC_DATA_EN_ADDR_CH*n*). GDMA calculates the CRC in the unit of byte, so only the lower 8 bits of this field needs to be configured. If AHB/AXI_DMA_TX_CRC_DATA_EN_WR_DATA_CH*n* is configured as 0xF, it means that only data[3], data[2], data[1], and data[0] are used for computation.

CRC calculation is achieved in the GDMA by configuring a parallel calculation matrix derived from the polynomial. If the expected CRC result is n bits, then the matrix has n rows and $n + 8$ columns.

Based on the assumptions at the beginning of this section, the calculation matrix would be 5 rows and 13 columns, and the actual hardware CRC calculation formulas would be as follows ($\oplus$represents the XOR operation):

$$crc[0] = crc_tmp[0] \oplus crc_tmp[2] \oplus crc_tmp[3] \oplus data[0] \oplus data[3] \oplus data[5] \oplus data[6]$$
$$crc[1] = crc_tmp[1] \oplus crc_tmp[3] \oplus crc_tmp[4] \oplus data[1] \oplus data[4] \oplus data[6] \oplus data[7]$$
$$crc[2] = crc_tmp[0] \oplus crc_tmp[3] \oplus crc_tmp[4] \oplus data[0] \oplus data[2] \oplus data[3] \oplus data[6] \oplus data[7]$$
$$crc[3] = crc_tmp[0] \oplus crc_tmp[1] \oplus crc_tmp[4] \oplus data[1] \oplus data[4] \oplus data[3] \oplus data[7]$$
$$crc[4] = crc_tmp[1] \oplus crc_tmp[2] \oplus data[2] \oplus data[4] \oplus data[5]$$

The above calculation formula can be abstracted as a switch matrix shown in Figure 6.4-3:

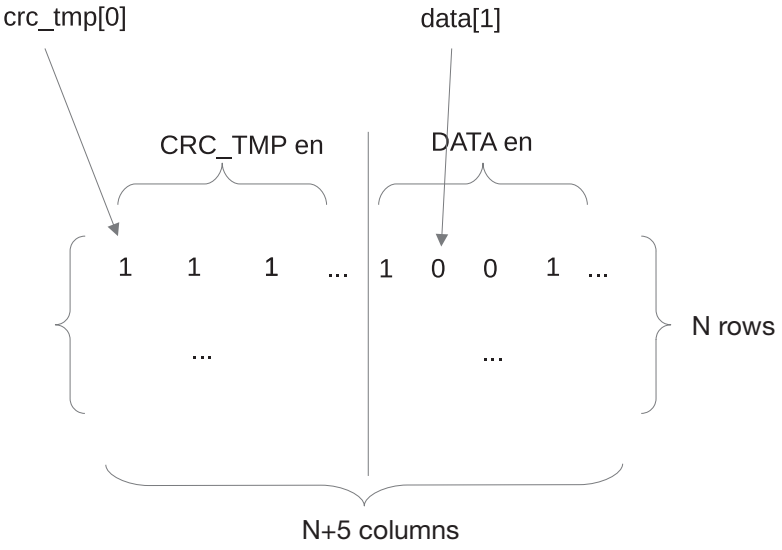


Figure 6.4-3. CRC Calculation Matrix

Note:
The CRC calculation described above is a Linear Feedback Shift Register (LFSR) implementation of the bytewise CRC algorithm. The formulas can be generated with some online tools according to the polynomial and the data width for parallel computation (always 8 bits for ESP32-P4).

6.5 Event Task Matrix Feature

The GDMA controller on ESP32-P4 supports the Event Task Matrix (ETM) function, which allows GDMA's ETM tasks to be triggered by any peripherals' ETM events, or GDMA's ETM events to trigger any peripherals' ETM tasks. This section introduces the ETM tasks and events related to GDMA. For more information, please refer to Chapter 14 Event Task Matrix (ETM).

GDMA can receive the following ETM tasks:

- GDMA_AHB/AXI_TASK_IN_START_CH*n*: Enables the corresponding RX channel *n* for data transfer.
- GDMA_AHB/AXI_TASK_OUT_START_CH*n*: Enables the corresponding TX channel *n* for data transfer.

Note:
Above ETM tasks can achieve the same functions as CPU configuring AHB/AXI_DMA_INLINK_START_CH*n* and AHB/AXI_DMA_OUTLINK_START_CH*n*. When AHB/AXI_DMA_IN_ETM_EN_CH*n* or AHB/AXI_DMA_OUT_ETM_EN_CH*n* is 1, only ETM tasks can be used to configure the transfer direction and enable the corresponding GDMA channel. When AHB/AXI_DMA_IN_ETM_EN_CH*n* or AHB_DMA_OUT_ETM_EN_CH*n* is 0, only CPU can be used to enable the corresponding GDMA channel.

GDMA can generate the following ETM events:

- GDMA_AHB/AXI_EVT_IN_DONE_CH*n*: Indicates that the data has been received according to the receive descriptor via channel *n*.
- GDMA_AHB/AXI_EVT_IN_SUC_EOF_CH*n*: Indicates that the data corresponding to a receive descriptor has been received via channel *n* and the EOF bit of this descriptor is 1.
- GDMA_AHB/AXI_EVT_IN_FIFO_EMPTY_CH*n*: Indicates that the RX FIFO has become empty.
- GDMA_AHB/AXI_EVT_IN_FIFO_FULL_CH*n*: Indicates that the RX FIFO has become full.
- GDMA_AHB/AXI_EVT_OUT_DONE_CH*n*: Indicates that the data has been transmitted according to the transmit descriptor via channel *n*.
- GDMA_AHB/AXI_EVT_OUT_SUC_EOF_CH*n*: Indicates that the data corresponding to a transmit descriptor has been transmitted or received via channel *n* and the EOF bit of this descriptor is 1.
- GDMA_AHB/AXI_EVT_OUT_TOTAL_EOF_CH*n*: Indicates that the data corresponding to the last transmit descriptors has been sent via transmit channel *n* and the EOF bit of this descriptor is 1.
- GDMA_AHB/AXI_EVT_OUT_FIFO_EMPTY_CH*n*: Indicates that the TX FIFO has become empty.
- GDMA_AHB/AXI_EVT_OUT_FIFO_FULL_CH*n*: Indicates that the TX FIFO has become full.

In practical applications, GDMA's ETM events can trigger its own ETM tasks. For example, the GDMA_AHB/AXI_EVT_OUT_TOTAL_EOF_CH0 event can trigger the GDMA_AHB/AXI_TASK_IN_START_CH1 task, and in this way trigger a new round of GDMA operations.

6.6 Interrupts

ESP32-P4's GDMA module can generate the following interrupt signals that will be sent to the Interrupt Matrix.

- AHB/AXI_PDMA_IN_CH0_INTR
- AHB/AXI_PDMA_IN_CH1_INTR
- AHB/AXI_PDMA_IN_CH2_INTR
- AHB/AXI_PDMA_OUT_CH0_INTR
- AHB/AXI_PDMA_OUT_CH1_INTR
- AHB/AXI_PDMA_OUT_CH2_INTR

There are several internal interrupt sources from GDMA that can generate the above interrupt signals.

The AHB/AXI_PDMA_IN_CH*n*_INTR (*n* is 0 ~ 2) signals are generated by the following interrupt sources:

- AHB/AXI_DMA_IN_DONE_CH*n*_INT: Triggered when all data corresponding to a receive descriptor has been received via receive channel *n*.
- AHB/AXI_DMA_IN_SUC_EOF_CH*n*_INT: Triggered when the suc_eof bit in a receive descriptor is 1 and the data corresponding to this receive descriptor has been received via receive channel *n*.
- AHB/AXI_DMA_IN_ERR_EOF_CH*n*_INT: Triggered when an error is detected in the data segment corresponding to a descriptor received via receive channel *n*. This interrupt is used only for UHCI (UART0 or UART1) or PARLIO.
- AHB/AXI_DMA_IN_DSCR_EMPTY_CH*n*_INT: Triggered when the size of the buffer pointed by receive descriptors is smaller than the length of data to be received via receive channel *n*.
- AHB/AXI_DMA_IN_DSCR_ERR_CH*n*_INT: Triggered when a receive descriptor on receive channel *n* fails any of the two descriptor checks.
- AHB_DMA_INFIFO_OVF_CH*n*_INT or AXI_DMA_INFIFO_L1/L2/L3_OVF_CH*n*_INT: Triggered when the RX FIFO of GDMA-AHB or the L1/L2/L3 RX FIFO of GDMA-AXI overflows.
- AHB_DMA_INFIFO_UDF_CH*n*_INT or AXI_DMA_INFIFO_L1/L2/L3_UDF_CH*n*_INT: Triggered when the RX FIFO of GDMA-AHB or the L1/L2/L3 RX FIFO of GDMA-AXI underflows.
- AHB_DMA_IN_AHBINF_RESP_ERR_CH*n*_INT: Triggered when GDMA-AHB receive channel *n* performs a data transfer on the AHB bus and an error response is received during the transfer.

The AHB/AXI_PDMA_OUT_CH*n*_INTR (*n* is 0 ~ 2) signals are generated by the following interrupt sources:

- AHB/AXI_DMA_OUT_DONE_CH*n*_INT: Triggered when all data corresponding to a transmit descriptor has been sent via transmit channel *n*.
- AHB/AXI_DMA_OUT_EOF_CH*n*_INT: Triggered when the suc_eof bit in a transmit descriptor is 1 and data corresponding to this descriptor has been sent via transmit channel *n*. If AHB/AXI_DMA_OUT_EOF_MODE_CH*n* is 0, this interrupt will be triggered when the last byte of data corresponding to this descriptor enters GDMA's transmit channel; if AHB/AXI_DMA_OUT_EOF_MODE_CH*n* is 1, this interrupt is triggered when the last byte of data is taken from GDMA's transmit channel.
- AHB/AXI_DMA_OUT_DSCR_ERR_CH*n*_INT: Triggered when a transmit descriptor on transmit channel *n* fails any of the two descriptor checks.
- AHB/AXI_DMA_OUT_TOTAL_EOF_CH*n*_INT: Triggered when all data corresponding to a linked list (including multiple descriptors) has been sent via transmit channel *n*.
- AHB_DMA_OUTFIFO_OVF_CH*n*_INT or AXI_DMA_OUTFIFO_L1/L2/L3_OVF_CH*n*_INT: Triggered when the TX FIFO of GDMA-AHB or the L1/L2/L3 TX FIFO of GDMA-AXI overflows.
- AHB_DMA_OUTFIFO_UDF_CH*n*_INT or AXI_DMA_OUTFIFO_L1/L2/L3_UDF_CH*n*_INT: Triggered when the TX FIFO of GDMA-AHB or the L1/L2/L3 TX FIFO of GDMA-AXI underflows.
- AXI_DMA_OUT_LINK_SWITCH_CH*n*_INT: Triggered when GDMA-AXI has switched to a new linked list on transmit channel *n*.
- AHB_DMA_OUT_AHBINF_RESP_ERR_CH*n*_INT: Triggered when GDMA-AHB transmit channel *n* performs a data transfer on the AHB bus and an error response is received during the transfer.

6.7 Programming Procedures

The clock gating for GDMA can be configured via HP_SYS_CLKRST_AHB/AXI_PDMA_SYS_CLK_EN, and is enabled by default. GDMA can be reset by configuring HP_SYS_CLKRST_RST_EN_AHB/AXI_PDMA.

6.7.1 Programming Procedures for GDMA's Transmit Channel

To transmit data, GDMA's transmit channel should be configured by software as follows:

1. Set AHB/AXI_DMA_OUT_RST_CH*n* first to 1 and then to 0, to reset the state machine of GDMA's transmit channel and FIFO pointer.
2. Load an outlink, and configure AHB/AXI_DMA_OUTLINK_ADDR_CH*n* with address of the first transmit descriptor.
3. Configure AHB/AXI_DMA_PERI_OUT_SEL_CH*n* with the value corresponding to the peripheral to be connected, as shown in Table 6.4-1 and Table 6.4-2.
4. Set AHB/AXI_DMA_OUTLINK_START_CH*n* to enable GDMA's transmit channel for data transfer.
5. Configure and enable the corresponding peripheral (SPI2, UHCI (UART0 or UART1), I2S, AES, SHA, and ADC). See details in individual chapters of these peripherals.
6. Wait for the AHB/AXI_DMA_OUT_TOTAL_EOF_CH*n*_INT interrupt, which indicates the completion of data transfer.

6.7.2 Programming Procedures for GDMA's Receive Channel

To receive data, GDMA's receive channel should be configured by software as follows:

1. Set AHB/AXI_DMA_IN_RST_CH*n* first to 1 and then to 0, to reset the state machine of GDMA's receive channel and FIFO pointer.
2. Load an inlink, and configure AHB/AXI_DMA_INLINK_ADDR_CH*n* with address of the first receive descriptor.
3. Configure AHB/AXI_DMA_PERI_IN_SEL_CH*n* with the value corresponding to the peripheral to be connected, as shown in Table 6.4-1 and Table 6.4-2.
4. Set AHB/AXI_DMA_INLINK_START_CH*n* to enable GDMA's receive channel for data transfer.
5. Configure and enable the corresponding peripheral (SPI2, UHCI (UART0 or UART1), I2S, AES, SHA, and ADC). See details in individual chapters of these peripherals.

6.7.3 Programming Procedures for Memory-to-Memory Transfer

To transfer data from one memory location to another, GDMA should be configured by software as follows:

1. Set AHB/AXI_DMA_OUT_RST_CH*n* first to 1 and then to 0, to reset the state machine of GDMA's transmit channel and FIFO pointer.
2. Set AHB/AXI_DMA_IN_RST_CH*n* first to 1 and then to 0, to reset the state machine of GDMA's receive channel and FIFO pointer.

3. Load an outlink, and configure AHB/AXI_DMA_OUTLINK_ADDR_CH*n* with address of the first transmit descriptor.
4. Load an inlink, and configure AHB/AXI_DMA_INLINK_ADDR_CH*n* with address of the first receive descriptor.
5. Set AHB/AXI_DMA_MEM_TRANS_EN_CH*n* to enable memory-to-memory transfer.
6. Set AHB/AXI_DMA_OUTLINK_START_CH*n* to enable GDMA's transmit channel for data transfer.
7. Set AHB/AXI_DMA_INLINK_START_CH*n* to enable GDMA's receive channel for data transfer.
8. If the suc_eof bit is set in a transmit descriptor, an AHB/AXI_DMA_IN_SUC_EOF_CH*n*_INT interrupt will be triggered when the data segment corresponding to this descriptor has been transmitted.

6.7.4 Programming Procedures for Channel Priority and Weight

The priority arbitration can be configured as follows:

1. Configure the channel priority for TX and RX respectively via AHB/AXI_DMA_TX_PRI_CH*n* and AHB/AXI_DMA_RX_PRI_CH*n*.

The weight arbitration can be configured as follows:

1. Configure the time slot
   - GDMA-AHB: TX and RX share the same time slot configured by AHB_DMA_ARB_TIMEOUT.
   - GDMA-AXI: TX and RX use independent time slots configured by AXI_DMA_ARB_TIMEOUT_TX and AXI_DMA_ARB_TIMEOUT_RX.
2. Configure the number of tokens for TX and RX respectively via AHB/AXI_DMA_TX_CH_ARB_WEIGH_CH*n* and AHB/AXI_DMA_RX_CH_ARB_WEIGH_CH*n*.
3. Enable weight arbitration optimization for TX and RX respectively by clearing AHB/AXI_DMA_TX_ARB_WEIGH_OPT_DIR_CH*n* and AHB/AXI_DMA_RX_ARB_WEIGH_OPT_DIR_CH*n*.
4. Enable the arbitration:
   - GDMA-AHB: Enable the arbitration for both TX and RX by setting AHB_DMA_WEIGHT_EN.
   - GDMA-AXI: Enable the arbitration for TX and RX respectively by setting AXI_DMA_WEIGHT_EN_TX and AXI_DMA_WEIGHT_EN_RX.

6.7.5 Programming Procedures for CRC Calculation

The CRC calculation registers can be configured as follows:

1. Configure which bits in crc_tmp and data are used in the matrix via AHB/AXI_DMA_TX_CRC_EN_ADDR_CH*n*, AHB/AXI_DMA_TX_CRC_DATA_EN_ADDR_CH*n*, AHB/AXI_DMA_TX_CRC_EN_WR_DATA_CH*n*, and AHB/AXI_DMA_TX_CRC_DATA_EN_WR_DATA_CH*n*.
2. Latch the values of the above four fields by first writing 1 and then 0 to AHB/AXI_DMA_TX_CRC_LAUTCH_FLGA_CH*n*. Then the configuration for a row is complete.
3. Repeat Step 1 and Step 2 until all rows in the matrix are set up, and then start DMA transfer.

6.8 Register Summary

6.8.1 GDMA-AHB Register Summary

The addresses in this section are relative to GDMA-AHB base address provided in Table 9.3-2 in Chapter 9 *System and Memory*.

The abbreviations given in Column **Access** are explained in Section *Access Types for Registers*.

Name	Description	Address	Access
Interrupt Registers			
AHB_DMA_IN_INT_RAW_CH0_REG	RX channel 0 raw interrupt status register	0x0000	R/WTC/SS
AHB_DMA_IN_INT_ST_CH0_REG	RX channel 0 masked interrupt status register	0x0004	RO
AHB_DMA_IN_INT_ENA_CH0_REG	RX channel 0 interrupt enable register	0x0008	R/W
AHB_DMA_IN_INT_CLR_CH0_REG	RX channel 0 interrupt clear register	0x000C	WT
AHB_DMA_IN_INT_RAW_CH1_REG	RX channel 1 raw interrupt status register	0x0010	R/WTC/SS
AHB_DMA_IN_INT_ST_CH1_REG	RX channel 1 masked interrupt status register	0x0014	RO
AHB_DMA_IN_INT_ENA_CH1_REG	RX channel 1 interrupt enable register	0x0018	R/W
AHB_DMA_IN_INT_CLR_CH1_REG	RX channel 1 interrupt clear register	0x001C	WT
AHB_DMA_IN_INT_RAW_CH2_REG	RX channel 2 raw interrupt status register	0x0020	R/WTC/SS
AHB_DMA_IN_INT_ST_CH2_REG	RX channel 2 masked interrupt status register	0x0024	RO
AHB_DMA_IN_INT_ENA_CH2_REG	RX channel 2 interrupt enable register	0x0028	R/W
AHB_DMA_IN_INT_CLR_CH2_REG	RX channel 2 interrupt clear register	0x002C	WT
AHB_DMA_OUT_INT_RAW_CH0_REG	TX channel 0 raw interrupt status register	0x0030	R/WTC/SS
AHB_DMA_OUT_INT_ST_CH0_REG	TX channel 0 masked interrupt status register	0x0034	RO
AHB_DMA_OUT_INT_ENA_CH0_REG	TX channel 0 interrupt enable register	0x0038	R/W
AHB_DMA_OUT_INT_CLR_CH0_REG	TX channel 0 interrupt clear register	0x003C	WT
AHB_DMA_OUT_INT_RAW_CH1_REG	TX channel 1 raw interrupt status register	0x0040	R/WTC/SS
AHB_DMA_OUT_INT_ST_CH1_REG	TX channel 1 masked interrupt status register	0x0044	RO
AHB_DMA_OUT_INT_ENA_CH1_REG	TX channel 1 interrupt enable register	0x0048	R/W
AHB_DMA_OUT_INT_CLR_CH1_REG	TX channel 1 interrupt clear register	0x004C	WT
AHB_DMA_OUT_INT_RAW_CH2_REG	TX channel 2 raw interrupt status register	0x0050	R/WTC/SS

GoBack

Name	Description	Address	Access
AHB_DMA_OUT_INT_ST_CH2_REG	TX channel 2 masked interrupt status register	0x0054	RO
AHB_DMA_OUT_INT_ENA_CH2_REG	TX channel 2 interrupt enable register	0x0058	R/W
AHB_DMA_OUT_INT_CLR_CH2_REG	TX channel 2 interrupt clear register	0x005C	WT
Debug Registers			
AHB_DMA_AHB_TEST_REG	Reserved	0x0060	R/W
Configuration Registers			
AHB_DMA_MISC_CONF_REG	Miscellaneous register	0x0064	R/W
AHB_DMA_IN_CONF0_CH0_REG	Configuration register 0 of RX channel 0	0x0070	R/W
AHB_DMA_IN_CONF1_CH0_REG	Configuration register 1 of Rx channel 0	0x0074	R/W
AHB_DMA_IN_POP_CH0_REG	Pop control register of RX channel 0	0x007C	varies
AHB_DMA_IN_LINK_CH0_REG	Linked list descriptor configuration and control register of RX channel 0	0x0080	varies
AHB_DMA_OUT_CONF0_CH0_REG	Configuration register 0 of TX channel 0	0x00D0	R/W
AHB_DMA_OUT_CONF1_CH0_REG	Configuration register 1 of TX channel 0	0x00D4	R/W
AHB_DMA_OUT_PUSH_CH0_REG	Push control register of TX channel 0	0x00DC	varies
AHB_DMA_OUT_LINK_CH0_REG	Linked list descriptor configuration and control register of TX channel 0	0x00E0	varies
AHB_DMA_IN_CONF0_CH1_REG	Configuration register 0 of RX channel 1	0x0130	R/W
AHB_DMA_IN_CONF1_CH1_REG	Configuration register 1 of RX channel 1	0x0134	R/W
AHB_DMA_IN_POP_CH1_REG	Pop control register of RX channel 1	0x013C	varies
AHB_DMA_IN_LINK_CH1_REG	Linked list descriptor configuration and control register of RX channel 1	0x0140	varies
AHB_DMA_OUT_CONF0_CH1_REG	Configuration register 0 of TX channel 1	0x0190	R/W
AHB_DMA_OUT_CONF1_CH1_REG	Configuration register 1 of TX channel 1	0x0194	R/W
AHB_DMA_OUT_PUSH_CH1_REG	Push control register of TX channel 1	0x019C	varies
AHB_DMA_OUT_LINK_CH1_REG	Linked list descriptor configuration and control register of TX channel 1	0x01A0	varies
AHB_DMA_IN_CONF0_CH2_REG	Configuration register 0 of RX channel 2	0x01F0	R/W
AHB_DMA_IN_CONF1_CH2_REG	Configuration register 1 of RX channel 2	0x01F4	R/W

Name	Description	Address	Access
AHB_DMA_IN_POP_CH2_REG	Pop control register of RX channel 2	0x01FC	varies
AHB_DMA_IN_LINK_CH2_REG	Linked list descriptor configuration and control register of RX channel 2	0x0200	varies
AHB_DMA_OUT_CONF0_CH2_REG	Configuration register 0 of TX channel 2	0x0250	R/W
AHB_DMA_OUT_CONF1_CH2_REG	Configuration register 1 of TX channel 2	0x0254	R/W
AHB_DMA_OUT_PUSH_CH2_REG	Push control register of TX channel 2	0x025C	varies
AHB_DMA_OUT_LINK_CH2_REG	Linked list descriptor configuration and control register of TX channel 2	0x0260	varies
AHB_DMA_OUT_CRC_INIT_DATA_CH0_REG	TX channel 0 CRC initial value configuration register	0x02BC	R/W
AHB_DMA_TX_CRC_WIDTH_CH0_REG	TX channel 0 CRC result width configuration register	0x02C0	R/W
AHB_DMA_OUT_CRC_CLEAR_CH0_REG	TX channel 0 CRC result clear register	0x02C4	R/W
AHB_DMA_OUT_CRC_FINAL_RESULT_CH0_REG	TX channel 0 CRC result register	0x02C8	RO
AHB_DMA_TX_CRC_EN_WR_DATA_CH0_REG	CRC TX channel 0 CRC intermediate result mask register	0x02CC	R/W
AHB_DMA_TX_CRC_EN_ADDR_CH0_REG	TX channel 0 CRC intermediate result mask target register	0x02D0	R/W
AHB_DMA_TX_CRC_DATA_EN_WR_DATA_CH0_REG	TX channel 0 CRC data input mask register	0x02D4	R/W
AHB_DMA_TX_CRC_DATA_EN_ADDR_CH0_REG	TX channel 0 CRC data input mask target register	0x02D8	R/W
AHB_DMA_TX_CH_ARB_WEIGH_CH0_REG	TX channel 0 arbitration weight configuration register	0x02DC	R/W
AHB_DMA_TX_ARB_WEIGH_OPT_DIR_CH0_REG	TX channel 0 weight arbitration optimization enable register	0x02E0	R/W
AHB_DMA_OUT_CRC_INIT_DATA_CH1_REG	TX channel 1 CRC initial value configuration register	0x02E4	R/W
AHB_DMA_TX_CRC_WIDTH_CH1_REG	TX channel 1 CRC result width configuration register	0x02E8	R/W
AHB_DMA_OUT_CRC_CLEAR_CH1_REG	TX channel 1 CRC result clear register	0x02EC	R/W
AHB_DMA_OUT_CRC_FINAL_RESULT_CH1_REG	TX channel 1 CRC result register	0x02F0	RO
AHB_DMA_TX_CRC_EN_WR_DATA_CH1_REG	TX channel 1 CRC intermediate result mask register	0x02F4	R/W
AHB_DMA_TX_CRC_EN_ADDR_CH1_REG	TX channel 1 CRC intermediate result mask target register	0x02F8	R/W
AHB_DMA_TX_CRC_DATA_EN_WR_DATA_CH1_REG	TX channel 1 CRC data input mask register	0x02FC	R/W
AHB_DMA_TX_CRC_DATA_EN_ADDR_CH1_REG	TX channel 1 CRC data input mask target register	0x0300	R/W
AHB_DMA_TX_CH_ARB_WEIGH_CH1_REG	TX channel 1 arbitration weight configuration register	0x0304	R/W
AHB_DMA_TX_ARB_WEIGH_OPT_DIR_CH1_REG	TX channel 1 weight arbitration optimization enable register	0x0308	R/W
AHB_DMA_OUT_CRC_INIT_DATA_CH2_REG	TX channel 2 CRC initial value configuration register	0x030C	R/W

Name	Description	Address	Access
AHB_DMA_TX_CRC_WIDTH_CH2_REG	TX channel 2 CRC result width configuration register	0x0310	R/W
AHB_DMA_OUT_CRC_CLEAR_CH2_REG	TX channel 2 CRC result clear register	0x0314	R/W
AHB_DMA_OUT_CRC_FINAL_RESULT_CH2_REG	TX channel 2 CRC result register	0x0318	RO
AHB_DMA_TX_CRC_EN_WR_DATA_CH2_REG	TX channel 2 CRC intermediate result mask register	0x031C	R/W
AHB_DMA_TX_CRC_EN_ADDR_CH2_REG	TX channel 2 CRC intermediate result mask target register	0x0320	R/W
AHB_DMA_TX_CRC_DATA_EN_WR_DATA_CH2_REG	TX channel 2 CRC data input mask register	0x0324	R/W
AHB_DMA_TX_CRC_DATA_EN_ADDR_CH2_REG	TX channel 2 CRC data input mask target register	0x0328	R/W
AHB_DMA_TX_CH_ARB_WEIGH_CH2_REG	TX channel 2 arbitration weight configuration register	0x032C	R/W
AHB_DMA_TX_ARB_WEIGH_OPT_DIR_CH2_REG	TX channel 2 weight arbitration optimization enable register	0x0330	R/W
AHB_DMA_IN_CRC_INIT_DATA_CH0_REG	RX channel 0 CRC initial value configuration register	0x0334	R/W
AHB_DMA_RX_CRC_WIDTH_CH0_REG	RX channel 0 CRC result width configuration register	0x0338	R/W
AHB_DMA_IN_CRC_CLEAR_CH0_REG	RX channel 0 CRC result clear register	0x033C	R/W
AHB_DMA_IN_CRC_FINAL_RESULT_CH0_REG	RX channel 0 CRC result register	0x0340	RO
AHB_DMA_RX_CRC_EN_WR_DATA_CH0_REG	RX channel 0 CRC intermediate result mask register	0x0344	R/W
AHB_DMA_RX_CRC_EN_ADDR_CH0_REG	RX channel 0 CRC data input mask target register	0x0348	R/W
AHB_DMA_RX_CRC_DATA_EN_WR_DATA_CH0_REG	RX channel 0 CRC data input mask register	0x034C	R/W
AHB_DMA_RX_CRC_DATA_EN_ADDR_CH0_REG	RX channel 0 CRC data input mask target register	0x0350	R/W
AHB_DMA_RX_CH_ARB_WEIGH_CH0_REG	RX channel 0 arbitration weight configuration register	0x0354	R/W
AHB_DMA_RX_ARB_WEIGH_OPT_DIR_CH0_REG	RX channel 0 weight arbitration optimization enable register	0x0358	R/W
AHB_DMA_IN_CRC_INIT_DATA_CH1_REG	RX channel 1 CRC initial value configuration register	0x035C	R/W
AHB_DMA_RX_CRC_WIDTH_CH1_REG	RX channel 1 CRC result width configuration register	0x0360	R/W
AHB_DMA_IN_CRC_CLEAR_CH1_REG	RX channel 1 CRC result clear register	0x0364	R/W
AHB_DMA_IN_CRC_FINAL_RESULT_CH1_REG	RX channel 1 CRC result register	0x0368	RO
AHB_DMA_RX_CRC_EN_WR_DATA_CH1_REG	RX channel 1 CRC intermediate result mask register	0x036C	R/W
AHB_DMA_RX_CRC_EN_ADDR_CH1_REG	RX channel 1 CRC data input mask target register	0x0370	R/W
AHB_DMA_RX_CRC_DATA_EN_WR_DATA_CH1_REG	RX channel 1 CRC data input mask register	0x0374	R/W
AHB_DMA_RX_CRC_DATA_EN_ADDR_CH1_REG	RX channel 1 CRC data input mask target register	0x0378	R/W
AHB_DMA_RX_CH_ARB_WEIGH_CH1_REG	RX channel 1 arbitration weight configuration register	0x037C	R/W
AHB_DMA_RX_ARB_WEIGH_OPT_DIR_CH1_REG	RX channel 1 weight arbitration optimization enable register	0x0380	R/W

Name	Description	Address	Access
AHB_DMA_IN_CRC_INIT_DATA_CH2_REG	RX channel 2 CRC initial value configuration register	0x0384	R/W
AHB_DMA_RX_CRC_WIDTH_CH2_REG	RX channel 2 CRC result width configuration register	0x0388	R/W
AHB_DMA_IN_CRC_CLEAR_CH2_REG	RX channel 2 CRC result clear register	0x038C	R/W
AHB_DMA_IN_CRC_FINAL_RESULT_CH2_REG	RX channel 2 CRC result register	0x0390	RO
AHB_DMA_RX_CRC_EN_WR_DATA_CH2_REG	RX channel 2 CRC intermediate result mask register	0x0394	R/W
AHB_DMA_RX_CRC_EN_ADDR_CH2_REG	RX channel 2 CRC data input mask target register	0x0398	R/W
AHB_DMA_RX_CRC_DATA_EN_WR_DATA_CH2_REG	RX channel 2 CRC data input mask register	0x039C	R/W
AHB_DMA_RX_CRC_DATA_EN_ADDR_CH2_REG	RX channel 2 CRC data input mask target register	0x03A0	R/W
AHB_DMA_RX_CH_ARB_WEIGH_CH2_REG	RX channel 2 arbitration weight configuration register	0x03A4	R/W
AHB_DMA_RX_ARB_WEIGH_OPT_DIR_CH2_REG	RX channel 2 weight arbitration optimization enable register	0x03A8	R/W
AHB_DMA_IN_LINK_ADDR_CH0_REG	Linked list descriptor configuration register of RX channel 0	0x03AC	R/W
AHB_DMA_IN_LINK_ADDR_CH1_REG	Linked list descriptor configuration register of RX channel 1	0x03B0	R/W
AHB_DMA_IN_LINK_ADDR_CH2_REG	Linked list descriptor configuration register of RX channel 2	0x03B4	R/W
AHB_DMA_OUT_LINK_ADDR_CH0_REG	Linked list descriptor configuration register of TX channel 0	0x03B8	R/W
AHB_DMA_OUT_LINK_ADDR_CH1_REG	Linked list descriptor configuration register of TX channel 1	0x03BC	R/W
AHB_DMA_OUT_LINK_ADDR_CH2_REG	Linked list descriptor configuration register of TX channel 2	0x03C0	R/W
AHB_DMA_INTR_MEM_START_ADDR_REG	Accessible address space start address configuration register	0x03C4	R/W
AHB_DMA_INTR_MEM_END_ADDR_REG	Accessible address space end address configuration register	0x03C8	R/W
AHB_DMA_ARB_TIMEOUT_REG	Weight arbitration timeout configuration register	0x03DC	R/W
AHB_DMA_WEIGHT_EN_REG	Weight arbitration enable register	0x0400	R/W
AHB_DMA_MODULE_CLK_EN_REG	Force clock enable register for all internal AHB DMA modules	0x0404	R/W
Version Register			
AHB_DMA_DATE_REG	Version control register	0x0068	R/W
Status Registers			
AHB_DMA_INFIFO_STATUS_CH0_REG	RX channel 0 FIFO status	0x0078	RO
AHB_DMA_IN_STATE_CH0_REG	RX channel 0 status	0x0084	RO
AHB_DMA_IN_SUC_EOF_DES_ADDR_CH0_REG	Receive descriptor address when EOF occurs on RX channel 0	0x0088	RO
AHB_DMA_IN_ERR_EOF_DES_ADDR_CH0_REG	Receive descriptor address when errors occur on RX channel 0	0x008C	RO

Name	Description	Address	Access
AHB_DMA_IN_DSCR_CH0_REG	Address of the next receive descriptor pointed by the current pre-read receive descriptor on RX channel 0	0x0090	RO
AHB_DMA_IN_DSCR_BF0_CH0_REG	Address of the current pre-read receive descriptor on RX channel 0	0x0094	RO
AHB_DMA_IN_DSCR_BF1_CH0_REG	Address of the previous pre-read receive descriptor on RX channel 0	0x0098	RO
AHB_DMA_OUTFIFO_STATUS_CH0_REG	TX channel 0 FIFO status	0x00D8	RO
AHB_DMA_OUT_STATE_CH0_REG	TX channel 0 status	0x00E4	RO
AHB_DMA_OUT_EOF_DES_ADDR_CH0_REG	Transmit descriptor address when EOF occurs on TX channel 0	0x00E8	RO
AHB_DMA_OUT_EOF_BFR_DES_ADDR_CH0_REG	The last transmit descriptor address when EOF occurs on TX channel 0	0x00EC	RO
AHB_DMA_OUT_DSCR_CH0_REG	Address of the next transmit descriptor pointed by the current pre-read transmit descriptor on TX channel 0	0x00F0	RO
AHB_DMA_OUT_DSCR_BF0_CH0_REG	Address of the current pre-read transmit descriptor on TX channel 0	0x00F4	RO
AHB_DMA_OUT_DSCR_BF1_CH0_REG	Address of the previous pre-read transmit descriptor on TX channel 0	0x00F8	RO
AHB_DMA_INFIFO_STATUS_CH1_REG	RX channel 1 FIFO status	0x0138	RO
AHB_DMA_IN_STATE_CH1_REG	RX channel 1 status	0x0144	RO
AHB_DMA_IN_SUC_EOF_DES_ADDR_CH1_REG	Receive descriptor address when EOF occurs on RX channel 1	0x0148	RO
AHB_DMA_IN_ERR_EOF_DES_ADDR_CH1_REG	Receive descriptor address when errors occur on RX channel 1	0x014C	RO
AHB_DMA_IN_DSCR_CH1_REG	Address of the next receive descriptor pointed by the current pre-read receive descriptor on RX channel 1	0x0150	RO
AHB_DMA_IN_DSCR_BF0_CH1_REG	Address of the current pre-read receive descriptor on RX channel 1	0x0154	RO
AHB_DMA_IN_DSCR_BF1_CH1_REG	Address of the previous pre-read receive descriptor on RX channel 1	0x0158	RO
AHB_DMA_OUTFIFO_STATUS_CH1_REG	TX channel 1 FIFO status	0x0198	RO
AHB_DMA_OUT_STATE_CH1_REG	TX channel 1 status	0x01A4	RO

Name	Description	Address	Access
AHB_DMA_OUT_EOF_DES_ADDR_CH1_REG	Transmit descriptor address when EOF occurs on TX channel 1	0x01A8	RO
AHB_DMA_OUT_EOF_BFR_DES_ADDR_CH1_REG	The last transmit descriptor address when EOF occurs on TX channel 1	0x01AC	RO
AHB_DMA_OUT_DSCR_CH1_REG	Address of the next transmit descriptor pointed by the current pre-read transmit descriptor on TX channel 1	0x01B0	RO
AHB_DMA_OUT_DSCR_BF0_CH1_REG	Address of the current pre-read transmit descriptor on TX channel 1	0x01B4	RO
AHB_DMA_OUT_DSCR_BF1_CH1_REG	Address of the previous pre-read transmit descriptor on TX channel 1	0x01B8	RO
AHB_DMA_INFIFO_STATUS_CH2_REG	RX channel 2 FIFO status	0x01F8	RO
AHB_DMA_IN_STATE_CH2_REG	RX channel 2 status	0x0204	RO
AHB_DMA_IN_SUC_EOF_DES_ADDR_CH2_REG	Receive descriptor address when EOF occurs on RX channel 2	0x0208	RO
AHB_DMA_IN_ERR_EOF_DES_ADDR_CH2_REG	Receive descriptor address when errors occur on RX channel 2	0x020C	RO
AHB_DMA_IN_DSCR_CH2_REG	Address of the next receive descriptor pointed by the current pre-read receive descriptor on RX channel 2	0x0210	RO
AHB_DMA_IN_DSCR_BF0_CH2_REG	Address of the current pre-read receive descriptor on RX channel 2	0x0214	RO
AHB_DMA_IN_DSCR_BF1_CH2_REG	Address of the previous pre-read receive descriptor on RX channel 2	0x0218	RO
AHB_DMA_OUTFIFO_STATUS_CH2_REG	TX channel 2 FIFO status	0x0258	RO
AHB_DMA_OUT_STATE_CH2_REG	TX channel 2 status	0x0264	RO
AHB_DMA_OUT_EOF_DES_ADDR_CH2_REG	Transmit descriptor address when EOF occurs on TX channel 2	0x0268	RO
AHB_DMA_OUT_EOF_BFR_DES_ADDR_CH2_REG	The last transmit descriptor address when EOF occurs on TX channel 2	0x026C	RO
AHB_DMA_OUT_DSCR_CH2_REG	Address of the next transmit descriptor pointed by the current pre-read transmit descriptor on TX channel 2	0x0270	RO
AHB_DMA_OUT_DSCR_BF0_CH2_REG	Address of the current pre-read transmit descriptor on TX channel 2	0x0274	RO

Name	Description	Address	Access
AHB_DMA_OUT_DSCR_BF1_CH2_REG	Address of the previous pre-read transmit descriptor on TX channel 2	0x0278	RO
AHB_DMA_AHBINF_RESP_ERR_STATUS0_REG	Address of the transfer for which AHB DMA received an AHB bus error response	0x0408	RO
AHB_DMA_AHBINF_RESP_ERR_STATUS1_REG	Transfer type, ID, and channel ID of the transfer for which AHB DMA received an AHB bus error response	0x040C	RO
AHB_DMA_IN_DONE_DES_ADDR_CH0_REG	Address of the completed descriptor for RX channel 0	0x0410	RO
AHB_DMA_OUT_DONE_DES_ADDR_CH0_REG	Address of the completed descriptor for TX channel 0	0x0414	RO
AHB_DMA_IN_DONE_DES_ADDR_CH1_REG	Address of the completed descriptor for RX channel 1	0x0418	RO
AHB_DMA_OUT_DONE_DES_ADDR_CH1_REG	Address of the completed descriptor for TX channel 1	0x041C	RO
AHB_DMA_IN_DONE_DES_ADDR_CH2_REG	Address of the completed descriptor for RX channel 2	0x0420	RO
AHB_DMA_OUT_DONE_DES_ADDR_CH2_REG	Address of the completed descriptor for TX channel 2	0x0424	RO
Priority Registers			
AHB_DMA_IN_PRI_CH0_REG	Priority register of RX channel 0	0x009C	R/W
AHB_DMA_OUT_PRI_CH0_REG	Priority register of TX channel 0	0x00FC	R/W
AHB_DMA_IN_PRI_CH1_REG	Priority register of RX channel 1	0x015C	R/W
AHB_DMA_OUT_PRI_CH1_REG	Priority register of TX channel 1	0x01BC	R/W
AHB_DMA_IN_PRI_CH2_REG	Priority register of RX channel 2	0x021C	R/W
AHB_DMA_OUT_PRI_CH2_REG	Priority register of TX channel 2	0x027C	R/W
Peripheral Select Registers			
AHB_DMA_IN_PERI_SEL_CH0_REG	Peripheral selection register of RX channel 0	0x00A0	R/W
AHB_DMA_OUT_PERI_SEL_CH0_REG	Peripheral selection register of TX channel 0	0x0100	R/W
AHB_DMA_IN_PERI_SEL_CH1_REG	Peripheral selection register of RX channel 1	0x0160	R/W
AHB_DMA_OUT_PERI_SEL_CH1_REG	Peripheral selection register of TX channel 1	0x01C0	R/W
AHB_DMA_IN_PERI_SEL_CH2_REG	Peripheral selection register of RX channel 2	0x0220	R/W
AHB_DMA_OUT_PERI_SEL_CH2_REG	Peripheral selection register of TX channel 2	0x0280	R/W

6.8.2 GDMA-AXI Register Summary

The addresses in this section are relative to GDMA-AXI base address provided in Table 9.3-2 in Chapter 9 *System and Memory*.

The abbreviations given in Column **Access** are explained in Section *Access Types for Registers*.

Name	Description	Address	Access
Interrupt Registers			
AXI_DMA_IN_INT_RAW_CH0_REG	RX channel 0 raw interrupt status register	0x0000	R/WTC/SS
AXI_DMA_IN_INT_ST_CH0_REG	RX channel 0 masked interrupt status register	0x0004	RO
AXI_DMA_IN_INT_ENA_CH0_REG	RX channel 0 interrupt enable register	0x0008	R/W
AXI_DMA_IN_INT_CLR_CH0_REG	RX channel 0 interrupt clear register	0x000C	WT
AXI_DMA_IN_INT_RAW_CH1_REG	RX channel 1 raw interrupt status register	0x0068	R/WTC/SS
AXI_DMA_IN_INT_ST_CH1_REG	RX channel 1 masked interrupt status register	0x006C	RO
AXI_DMA_IN_INT_ENA_CH1_REG	RX channel 1 interrupt enable register	0x0070	R/W
AXI_DMA_IN_INT_CLR_CH1_REG	RX channel 1 interrupt clear register	0x0074	WT
AXI_DMA_IN_INT_RAW_CH2_REG	RX channel 2 raw interrupt status register	0x00D0	R/WTC/SS
AXI_DMA_IN_INT_ST_CH2_REG	RX channel 2 masked interrupt status register	0x00D4	RO
AXI_DMA_IN_INT_ENA_CH2_REG	RX channel 2 interrupt enable register	0x00D8	R/W
AXI_DMA_IN_INT_CLR_CH2_REG	RX channel 2 interrupt clear register	0x00DC	WT
AXI_DMA_OUT_INT_RAW_CH0_REG	TX channel 0 raw interrupt status register	0x0138	R/WTC/SS
AXI_DMA_OUT_INT_ST_CH0_REG	TX channel 0 masked interrupt status register	0x013C	RO
AXI_DMA_OUT_INT_ENA_CH0_REG	TX channel 0 interrupt enable register	0x0140	R/W
AXI_DMA_OUT_INT_CLR_CH0_REG	TX channel 0 interrupt clear register	0x0144	WT
AXI_DMA_OUT_INT_RAW_CH1_REG	TX channel 1 raw interrupt status register	0x01A0	R/WTC/SS
AXI_DMA_OUT_INT_ST_CH1_REG	TX channel 1 masked interrupt status register	0x01A4	RO
AXI_DMA_OUT_INT_ENA_CH1_REG	TX channel 1 interrupt enable register	0x01A8	R/W
AXI_DMA_OUT_INT_CLR_CH1_REG	TX channel 1 interrupt clear register	0x01AC	WT
AXI_DMA_OUT_INT_RAW_CH2_REG	TX channel 2 raw interrupt status register	0x0208	R/WTC/SS
AXI_DMA_OUT_INT_ST_CH2_REG	TX channel 2 masked interrupt status register	0x020C	RO
AXI_DMA_OUT_INT_ENA_CH2_REG	TX channel 2 interrupt enable register	0x0210	R/W

GoBack

Name	Description	Address	Access
AXI_DMA_OUT_INT_CLR_CH2_REG	TX channel 2 interrupt clear register	0x0214	WT
Configuration Registers			
AXI_DMA_IN_CONF0_CH0_REG	Configuration register 0 of RX channel 0	0x0010	R/W
AXI_DMA_IN_CONF1_CH0_REG	Configuration register 1 of RX channel 0	0x0014	R/W
AXI_DMA_IN_POP_CH0_REG	Pop control register of RX channel 0	0x001C	varies
AXI_DMA_IN_LINK1_CH0_REG	Linked list descriptor configuration and control register 1 of RX channel 0	0x0020	varies
AXI_DMA_IN_LINK2_CH0_REG	Linked list descriptor configuration and control register 2 of RX channel 0	0x0024	R/W
AXI_DMA_IN_CRC_INIT_DATA_CH0_REG	RX channel 0 CRC initial value configuration register	0x0048	R/W
AXI_DMA_RX_CRC_WIDTH_CH0_REG	RX channel 0 CRC result width configuration register	0x004C	R/W
AXI_DMA_IN_CRC_CLEAR_CH0_REG	RX channel 0 CRC result clear register	0x0050	R/W
AXI_DMA_IN_CRC_FINAL_RESULT_CH0_REG	RX channel 0 CRC result register	0x0054	RO
AXI_DMA_RX_CRC_EN_ADDR_CH0_REG	TX channel 0 CRC intermediate result mask target register	0x005C	R/W
AXI_DMA_RX_CRC_DATA_EN_WR_DATA_CH0_REG	CRC RX channel 0 CRC intermediate result mask register	0x0060	R/W
AXI_DMA_RX_CRC_DATA_EN_ADDR_CH0_REG	RX channel 0 CRC data input mask target register	0x0064	R/W
AXI_DMA_IN_CONF0_CH1_REG	Configuration register 0 of RX channel 1	0x0078	R/W
AXI_DMA_IN_CONF1_CH1_REG	Configuration register 1 of RX channel 1	0x007C	R/W
AXI_DMA_IN_POP_CH1_REG	Pop control register of RX channel 1	0x0084	varies
AXI_DMA_IN_LINK1_CH1_REG	Linked list descriptor configuration and control register 1 of RX channel 1	0x0088	varies
AXI_DMA_IN_LINK2_CH1_REG	Linked list descriptor configuration and control register 2 of RX channel 1	0x008C	R/W
AXI_DMA_IN_CRC_INIT_DATA_CH1_REG	RX channel 1 CRC initial value configuration register	0x00B0	R/W
AXI_DMA_RX_CRC_WIDTH_CH1_REG	RX channel 1 CRC result width configuration register	0x00B4	R/W
AXI_DMA_IN_CRC_CLEAR_CH1_REG	RX channel 1 CRC result clear register	0x00B8	R/W
AXI_DMA_IN_CRC_FINAL_RESULT_CH1_REG	RX channel 1 CRC result register	0x00BC	RO
AXI_DMA_RX_CRC_EN_ADDR_CH1_REG	RX channel 1 CRC intermediate result mask target register	0x00C4	R/W
AXI_DMA_RX_CRC_DATA_EN_WR_DATA_CH1_REG	CRC RX channel 1 CRC intermediate result mask register	0x00C8	R/W

GoBack

Name	Description	Address	Access
AXI_DMA_RX_CRC_DATA_EN_ADDR_CH1_REG	RX channel 1 CRC data input mask target register	0x00CC	R/W
AXI_DMA_IN_CONF0_CH2_REG	Configuration register 0 of RX channel 2	0x00E0	R/W
AXI_DMA_IN_CONF1_CH2_REG	Configuration register 1 of RX channel 2	0x00E4	R/W
AXI_DMA_IN_POP_CH2_REG	Pop control register of RX channel 2	0x00EC	varies
AXI_DMA_IN_LINK1_CH2_REG	Linked list descriptor configuration and control register 1 of RX channel 2	0x00F0	varies
AXI_DMA_IN_LINK2_CH2_REG	Linked list descriptor configuration and control register 2 of RX channel 2	0x00F4	R/W
AXI_DMA_IN_CRC_INIT_DATA_CH2_REG	RX channel 2 CRC initial value configuration register	0x0118	R/W
AXI_DMA_RX_CRC_WIDTH_CH2_REG	RX channel 2 CRC result width configuration register	0x011C	R/W
AXI_DMA_IN_CRC_CLEAR_CH2_REG	RX channel 2 CRC result clear register	0x0120	R/W
AXI_DMA_IN_CRC_FINAL_RESULT_CH2_REG	RX channel 2 CRC result register	0x0124	RO
AXI_DMA_RX_CRC_EN_ADDR_CH2_REG	RX channel 2 CRC intermediate result mask target register	0x012C	R/W
AXI_DMA_RX_CRC_DATA_EN_WR_DATA_CH2_REG	CRC RX channel 2 CRC intermediate result mask register	0x0130	R/W
AXI_DMA_RX_CRC_DATA_EN_ADDR_CH2_REG	RX channel 2 CRC data input mask target register	0x0134	R/W
AXI_DMA_OUT_CONF0_CH0_REG	Configuration register 0 of TX channel 0	0x0148	R/W
AXI_DMA_OUT_CONF1_CH0_REG	Configuration register 1 of TX channel 0	0x014C	R/W
AXI_DMA_OUT_PUSH_CH0_REG	Push control register of TX channel 0	0x0154	varies
AXI_DMA_OUT_LINK1_CH0_REG	Linked list descriptor configuration and control register 1 of TX channel 0	0x0158	varies
AXI_DMA_OUT_LINK2_CH0_REG	Linked list descriptor configuration and control register 2 of TX channel 0	0x015C	R/W
AXI_DMA_OUT_CRC_INIT_DATA_CH0_REG	TX channel 0 CRC initial value configuration register	0x0180	R/W
AXI_DMA_TX_CRC_WIDTH_CH0_REG	TX channel 0 CRC result width configuration register	0x0184	R/W
AXI_DMA_OUT_CRC_CLEAR_CH0_REG	TX channel 0 CRC result clear register	0x0188	R/W
AXI_DMA_OUT_CRC_FINAL_RESULT_CH0_REG	TX channel 0 CRC result register	0x018C	RO
AXI_DMA_TX_CRC_EN_WR_DATA_CH0_REG	CRC TX channel 0 CRC intermediate result mask register	0x0190	R/W
AXI_DMA_TX_CRC_EN_ADDR_CH0_REG	TX channel 0 CRC intermediate result mask target register	0x0194	R/W
AXI_DMA_TX_CRC_DATA_EN_WR_DATA_CH0_REG	TX channel 0 CRC data input mask register	0x0198	R/W

Name	Description	Address	Access
AXI_DMA_TX_CRC_DATA_EN_ADDR_CH0_REG	TX channel 0 CRC data input mask target register	0x019C	R/W
AXI_DMA_OUT_CONF0_CH1_REG	Configuration register 0 of TX channel 1	0x01B0	R/W
AXI_DMA_OUT_CONF1_CH1_REG	Configuration register 1 of TX channel 1	0x01B4	R/W
AXI_DMA_OUT_PUSH_CH1_REG	Push control register of TX channel 1	0x01BC	varies
AXI_DMA_OUT_LINK1_CH1_REG	Linked list descriptor configuration and control register 1 of TX channel 1	0x01C0	varies
AXI_DMA_OUT_LINK2_CH1_REG	Linked list descriptor configuration and control register 2 of TX channel 1	0x01C4	R/W
AXI_DMA_OUT_CRC_INIT_DATA_CH1_REG	TX channel 1 CRC initial value configuration register	0x01E8	R/W
AXI_DMA_TX_CRC_WIDTH_CH1_REG	TX channel 1 CRC result width configuration register	0x01EC	R/W
AXI_DMA_OUT_CRC_CLEAR_CH1_REG	TX channel 1 CRC result clear register	0x01F0	R/W
AXI_DMA_OUT_CRC_FINAL_RESULT_CH1_REG	TX channel 1 CRC result register	0x01F4	RO
AXI_DMA_TX_CRC_EN_WR_DATA_CH1_REG	CRC TX channel 1 CRC intermediate result mask register	0x01F8	R/W
AXI_DMA_TX_CRC_EN_ADDR_CH1_REG	TX channel 1 CRC intermediate result mask target register	0x01FC	R/W
AXI_DMA_TX_CRC_DATA_EN_WR_DATA_CH1_REG	TX channel 1 CRC data input mask target register	0x0200	R/W
AXI_DMA_TX_CRC_DATA_EN_ADDR_CH1_REG	TX channel 1 CRC data input mask target register	0x0204	R/W
AXI_DMA_OUT_CONF0_CH2_REG	Configuration register 0 of TX channel 2	0x0218	R/W
AXI_DMA_OUT_CONF1_CH2_REG	Configuration register 1 of TX channel 2	0x021C	R/W
AXI_DMA_OUT_PUSH_CH2_REG	Push control register of TX channel 2	0x0224	varies
AXI_DMA_OUT_LINK1_CH2_REG	Linked list descriptor configuration and control register of TX channel 2	0x0228	varies
AXI_DMA_OUT_LINK2_CH2_REG	Linked list descriptor configuration and control register 2 1of TX channel 2	0x022C	R/W
AXI_DMA_OUT_CRC_INIT_DATA_CH2_REG	TX channel 2 CRC initial value configuration register	0x0250	R/W
AXI_DMA_TX_CRC_WIDTH_CH2_REG	TX channel 2 CRC result width configuration register	0x0254	R/W
AXI_DMA_OUT_CRC_CLEAR_CH2_REG	TX channel 2 CRC result clear register	0x0258	R/W
AXI_DMA_OUT_CRC_FINAL_RESULT_CH2_REG	TX channel 2 CRC result register	0x025C	RO
AXI_DMA_TX_CRC_EN_WR_DATA_CH2_REG	CRC TX channel 2 CRC intermediate result mask register	0x0260	R/W
AXI_DMA_TX_CRC_EN_ADDR_CH2_REG	TX channel 2 CRC intermediate result mask target register	0x0264	R/W

Name	Description	Address	Access
AXI_DMA_TX_CRC_DATA_EN_WR_DATA_CH2_REG	TX channel 2 CRC data input mask target register	0x0268	R/W
AXI_DMA_TX_CRC_DATA_EN_ADDR_CH2_REG	TX channel 2 CRC data input mask target register	0x026C	R/W
AXI_DMA_ARB_TIMEOUT_REG	Arbitration timeout configuration register	0x0270	R/W
AXI_DMA_WEIGHT_EN_REG	Weight arbitration enable register	0x0274	R/W
AXI_DMA_IN_MEM_CONF_REG	Internal memory power configuration register for RX channel	0x0278	R/W
AXI_DMA_INTR_MEM_START_ADDR_REG	Accessible internal memory start address configuration register	0x027C	R/W
AXI_DMA_INTR_MEM_END_ADDR_REG	Accessible internal memory end address configuration register	0x0280	R/W
AXI_DMA_EXTR_MEM_START_ADDR_REG	Accessible external memory start address configuration register	0x0284	R/W
AXI_DMA_EXTR_MEM_END_ADDR_REG	Accessible external memory end address configuration register	0x0288	R/W
AXI_DMA_MISC_CONF_REG	Miscellaneous register	0x02A8	R/W
AXI_DMA_LINK_SWITCH_STATE_REG	Linked list switching configuration register	0x02DC	R/W
AXI_DMA_RX_CRC_EN_WR_DATA_CH0_REG	CRC RX channel 0 CRC intermediate result mask register	0x0058	R/W
AXI_DMA_RX_CRC_EN_WR_DATA_CH1_REG	CRC RX channel 1 CRC intermediate result mask register	0x00C0	R/W
AXI_DMA_RX_CRC_EN_WR_DATA_CH2_REG	CRC RX channel 2 CRC intermediate result mask register	0x0128	R/W
Status Registers			
AXI_DMA_INFIFO_STATUS_CH0_REG	RX channel 0 FIFO status register	0x0018	RO
AXI_DMA_IN_STATE_CH0_REG	RX channel 0 status register	0x0028	RO
AXI_DMA_IN_SUC_EOF_DES_ADDR_CH0_REG	Receive descriptor address when EOF occurs on RX channel 0	0x002C	RO
AXI_DMA_IN_ERR_EOF_DES_ADDR_CH0_REG	Receive descriptor address when errors occur on RX channel 0	0x0030	RO
AXI_DMA_IN_DSCR_CH0_REG	Address of the next receive descriptor pointed by the current pre-read receive descriptor on RX channel 0	0x0034	RO
AXI_DMA_IN_DSCR_BF0_CH0_REG	Address of the current pre-read receive descriptor on RX channel 0	0x0038	RO
AXI_DMA_IN_DSCR_BF1_CH0_REG	Address of the previous pre-read receive descriptor on RX channel 0	0x003C	RO
AXI_DMA_INFIFO_STATUS_CH1_REG	RX channel 1 FIFO status register	0x0080	RO
AXI_DMA_IN_STATE_CH1_REG	RX channel 1 status register	0x0090	RO
AXI_DMA_IN_SUC_EOF_DES_ADDR_CH1_REG	Receive descriptor address when EOF occurs on RX channel 1	0x0094	RO
AXI_DMA_IN_ERR_EOF_DES_ADDR_CH1_REG	Receive descriptor address when errors occur on RX channel 1	0x0098	RO

Name	Description	Address	Access
AXI_DMA_IN_DSCR_CH1_REG	Address of the next receive descriptor pointed by the current pre-read receive descriptor on RX channel 1	0x009C	RO
AXI_DMA_IN_DSCR_BF0_CH1_REG	Address of the current pre-read receive descriptor on RX channel 1	0x00A0	RO
AXI_DMA_IN_DSCR_BF1_CH1_REG	Address of the previous pre-read receive descriptor on RX channel 1	0x00A4	RO
AXI_DMA_INFIFO_STATUS_CH2_REG	RX channel 2 FIFO status register	0x00E8	RO
AXI_DMA_IN_STATE_CH2_REG	RX channel 2 status register	0x00F8	RO
AXI_DMA_IN_SUC_EOF_DES_ADDR_CH2_REG	Receive descriptor address when EOF occurs on RX channel 2	0x00FC	RO
AXI_DMA_IN_ERR_EOF_DES_ADDR_CH2_REG	Receive descriptor address when errors occur on RX channel 2	0x0100	RO
AXI_DMA_IN_DSCR_CH2_REG	Address of the next receive descriptor pointed by the current pre-read receive descriptor on RX channel 2	0x0104	RO
AXI_DMA_IN_DSCR_BF0_CH2_REG	Address of the current pre-read receive descriptor on RX channel 2	0x0108	RO
AXI_DMA_IN_DSCR_BF1_CH2_REG	Address of the previous pre-read receive descriptor on RX channel 2	0x010C	RO
AXI_DMA_OUTFIFO_STATUS_CH0_REG	TX channel 0 FIFO status	0x0150	RO
AXI_DMA_OUT_STATE_CH0_REG	TX channel 0 status	0x0160	RO
AXI_DMA_OUT_EOF_DES_ADDR_CH0_REG	Transmit descriptor address when EOF occurs on TX channel 0	0x0164	RO
AXI_DMA_OUT_EOF_BFR_DES_ADDR_CH0_REG	The last transmit descriptor address when EOF occurs on TX channel 0	0x0168	RO
AXI_DMA_OUT_DSCR_CH0_REG	Address of the next transmit descriptor pointed by the current pre-read transmit descriptor on TX channel 0	0x016C	RO
AXI_DMA_OUT_DSCR_BF0_CH0_REG	Address of the current pre-read transmit descriptor on TX channel 0	0x0170	RO
AXI_DMA_OUT_DSCR_BF1_CH0_REG	Address of the previous pre-read transmit descriptor on TX channel 0	0x0174	RO
AXI_DMA_OUTFIFO_STATUS_CH1_REG	TX channel 0 FIFO status	0x01B8	RO
AXI_DMA_OUT_STATE_CH1_REG	TX channel 1 status	0x01C8	RO

Name	Description	Address	Access
AXI_DMA_OUT_EOF_DES_ADDR_CH1_REG	Transmit descriptor address when EOF occurs on TX channel 1	0x01CC	RO
AXI_DMA_OUT_EOF_BFR_DES_ADDR_CH1_REG	The last transmit descriptor address when EOF occurs on TX channel 1	0x01D0	RO
AXI_DMA_OUT_DSCR_CH1_REG	Address of the next transmit descriptor pointed by the current pre-read transmit descriptor on TX channel 1	0x01D4	RO
AXI_DMA_OUT_DSCR_BF0_CH1_REG	Address of the current pre-read transmit descriptor on TX channel 1	0x01D8	RO
AXI_DMA_OUT_DSCR_BF1_CH1_REG	Address of the previous pre-read transmit descriptor on TX channel 1	0x01DC	RO
AXI_DMA_OUTFIFO_STATUS_CH2_REG	TX channel 2 FIFO status	0x0220	RO
AXI_DMA_OUT_STATE_CH2_REG	TX channel 2 status	0x0230	RO
AXI_DMA_OUT_EOF_DES_ADDR_CH2_REG	Transmit descriptor address when EOF occurs on TX channel 2	0x0234	RO
AXI_DMA_OUT_EOF_BFR_DES_ADDR_CH2_REG	The last transmit descriptor address when EOF occurs on TX channel 2	0x0238	RO
AXI_DMA_OUT_DSCR_CH2_REG	Address of the next transmit descriptor pointed by the current pre-read transmit descriptor on TX channel 2	0x023C	RO
AXI_DMA_OUT_DSCR_BF0_CH2_REG	Address of the current pre-read transmit descriptor on TX channel 2	0x0240	RO
AXI_DMA_OUT_DSCR_BF1_CH2_REG	Address of the previous pre-read transmit descriptor on TX channel 2	0x0244	RO
AXI_DMA_IN_RESET_AVAIL_CH0_REG	RX channel 0 reset status register	0x028C	RO
AXI_DMA_IN_RESET_AVAIL_CH1_REG	RX channel 1 reset status register	0x0290	RO
AXI_DMA_IN_RESET_AVAIL_CH2_REG	RX channel 2 reset status register	0x0294	RO
AXI_DMA_OUT_RESET_AVAIL_CH0_REG	TX channel 0 reset status register	0x0298	RO
AXI_DMA_OUT_RESET_AVAIL_CH1_REG	TX channel 1 reset status register	0x029C	RO
AXI_DMA_OUT_RESET_AVAIL_CH2_REG	TX channel 2 reset status register	0x02A0	RO
Priority Registers			
AXI_DMA_IN_PRI_CH0_REG	Priority register of RX channel 0	0x0040	R/W
AXI_DMA_IN_PRI_CH1_REG	Priority register of RX channel 1	0x00A8	R/W

Name	Description	Address	Access
AXI_DMA_IN_PRI_CH2_REG	Priority register of RX channel 2	0x0110	R/W
AXI_DMA_OUT_PRI_CH0_REG	Priority register of TX channel 0	0x0178	R/W
AXI_DMA_OUT_PRI_CH1_REG	Priority register of TX channel 1	0x01E0	R/W
AXI_DMA_OUT_PRI_CH2_REG	Priority register of TX channel 2	0x0248	R/W
Peripheral Select Registers			
AXI_DMA_IN_PERI_SEL_CH0_REG	Peripheral selection register of RX channel 0	0x0044	R/W
AXI_DMA_IN_PERI_SEL_CH1_REG	Peripheral selection register of RX channel 1	0x00AC	R/W
AXI_DMA_IN_PERI_SEL_CH2_REG	Peripheral selection register of RX channel 2	0x0114	R/W
AXI_DMA_OUT_PERI_SEL_CH0_REG	Peripheral selection register of TX channel 0	0x017C	R/W
AXI_DMA_OUT_PERI_SEL_CH1_REG	Peripheral selection register of TX channel 1	0x01E4	R/W
AXI_DMA_OUT_PERI_SEL_CH2_REG	Peripheral selection register of TX channel 2	0x024C	R/W
Version Register			
AXI_DMA_DATE_REG	Version control register	0x02D8	R/W

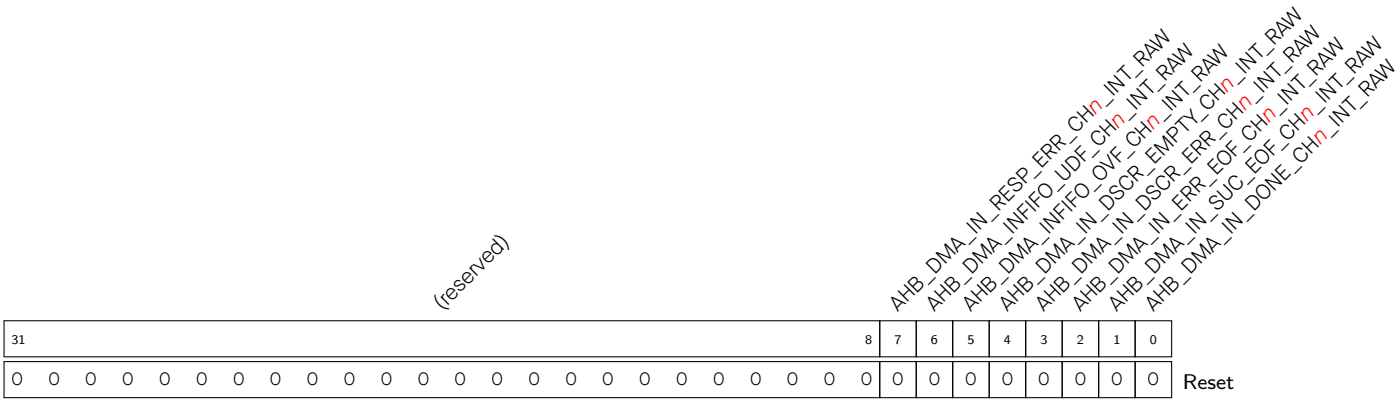
6.9 Registers

6.9.1 GDMA-AHB Registers

The addresses in this section are relative to GDMA-AHB base address provided in Table 9.3-2 in Chapter 9 *System and Memory*.

For how to program reserved fields, please refer to Section Programming Reserved Register Field.

Register 6.1. AHB_DMA_IN_INT_RAW_CH*n*_REG (*n*: 0-2) (0x0000+0x10n*)**



AHB_DMA_IN_DONE_CH*n*_INT_RAW The raw interrupt status of AHB_DMA_IN_DONE_CH*n*_INT. (R/WTC/SS)

AHB_DMA_IN_SUC_EOF_CH*n*_INT_RAW The raw interrupt status of AHB_DMA_IN_SUC_EOF_CH*n*_INT. For UHCI, this bit turns to 1 when the last data byte pointed by one receive descriptor has been received and no data error is detected for RX channel *n*. (R/WTC/SS)

AHB_DMA_IN_ERR_EOF_CH*n*_INT_RAW The raw interrupt status of AHB_DMA_IN_ERR_EOF_CH*n*_INT. Valid only for UHCI. (R/WTC/SS)

AHB_DMA_IN_DSCR_ERR_CH*n*_INT_RAW The raw interrupt status of AHB_DMA_IN_DSCR_ERR_CH*n*_INT. (R/WTC/SS)

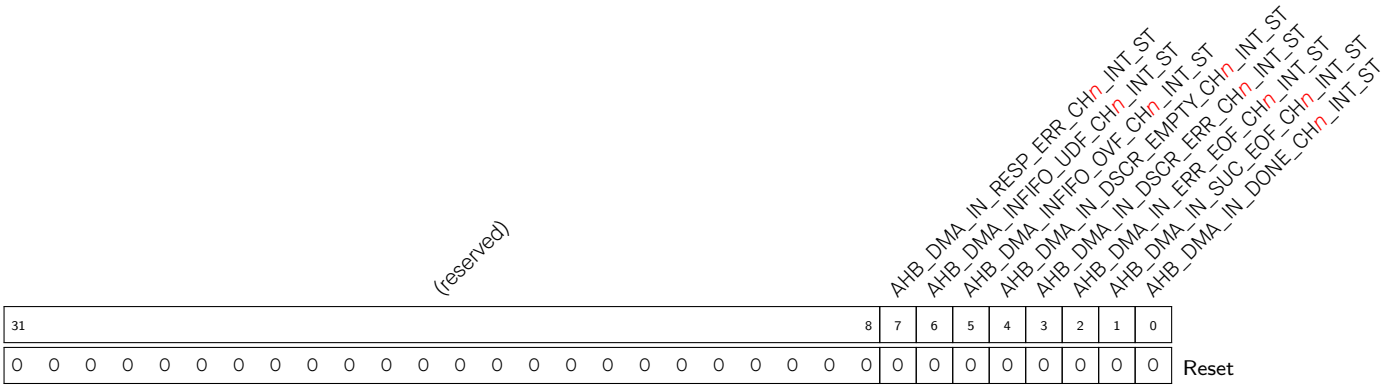
AHB_DMA_IN_DSCR_EMPTY_CH*n*_INT_RAW The raw interrupt status of AHB_DMA_IN_DSCR_EMPTY_CH*n*_INT. (R/WTC/SS)

AHB_DMA_INFIFO_OVF_CH*n*_INT_RAW The raw interrupt status of AHB_DMA_INFIFO_OVF_CH*n*_INT. (R/WTC/SS)

AHB_DMA_INFIFO_UDF_CH*n*_INT_RAW The raw interrupt status of AHB_DMA_INFIFO_UDF_CH*n*_INT. (R/WTC/SS)

AHB_DMA_IN_RESP_ERR_CH*n*_INT_RAW The raw interrupt status of AHB_DMA_IN_RESP_ERR_CH*n*_INT. (R/WTC/SS)

Register 6.2. AHB_DMA_IN_INT_ST_CH*n*_REG (*n*: 0-2) (0x0004+0x10***n*)



AHB_DMA_IN_DONE_CH*n*_INT_ST The masked interrupt status of AHB_DMA_IN_DONE_CH*n*_INT. (RO)

AHB_DMA_IN_SUC_EOF_CH*n*_INT_ST The masked interrupt status of AHB_DMA_IN_SUC_EOF_CH*n*_INT. (RO)

AHB_DMA_IN_ERR_EOF_CH*n*_INT_ST The masked interrupt status of AHB_DMA_IN_ERR_EOF_CH*n*_INT. (RO)

AHB_DMA_IN_DSCR_ERR_CH*n*_INT_ST The masked interrupt status of AHB_DMA_IN_DSCR_ERR_CH*n*_INT. (RO)

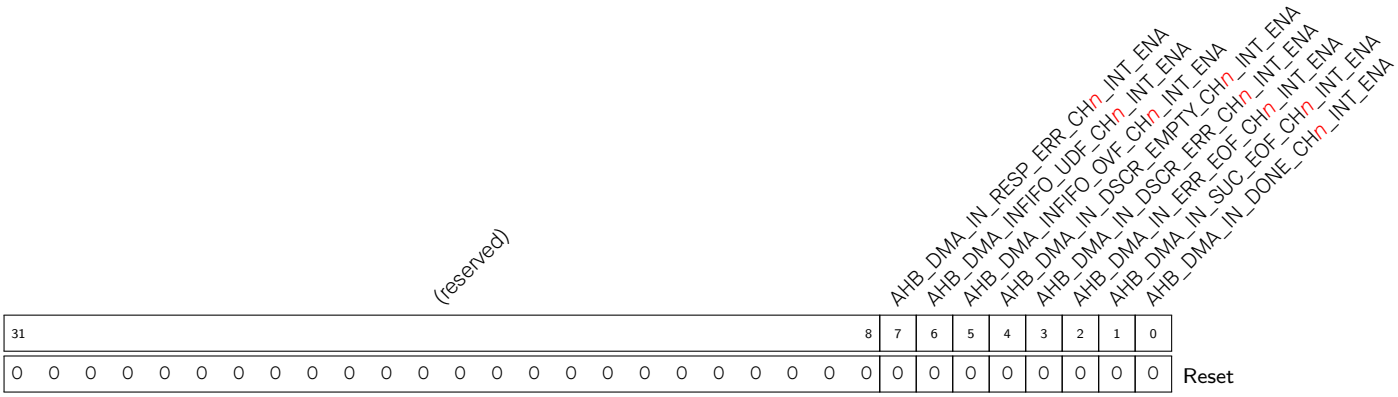
AHB_DMA_IN_DSCR_EMPTY_CH*n*_INT_ST The masked interrupt status of AHB_DMA_IN_DSCR_EMPTY_CH*n*_INT. (RO)

AHB_DMA_INFIFO_OVF_CH*n*_INT_ST The masked interrupt status of AHB_DMA_INFIFO_OVF_CH*n*_INT. (RO)

AHB_DMA_INFIFO_UDF_CH*n*_INT_ST The masked interrupt status of AHB_DMA_INFIFO_UDF_CH*n*_INT. (RO)

AHB_DMA_IN_RESP_ERR_CH*n*_INT_ST The masked interrupt status of AHB_DMA_IN_RESP_ERR_CH*n*_INT. (RO)

Register 6.3. AHB_DMA_IN_INT_ENA_CH*n*_REG (*n*: 0-2) (0x0008+0x10n*)**



AHB_DMA_IN_DONE_CH*n*_INT_ENA Write 1 to enable AHB_DMA_IN_DONE_CH*n*_INT. (R/W)

AHB_DMA_IN_SUC_EOF_CH*n*_INT_ENA Write 1 to enable AHB_DMA_IN_SUC_EOF_CH*n*_INT. (R/W)

AHB_DMA_IN_ERR_EOF_CH*n*_INT_ENA Write 1 to enable AHB_DMA_IN_ERR_EOF_CH*n*_INT. (R/W)

AHB_DMA_IN_DSCR_ERR_CH*n*_INT_ENA Write 1 to enable AHB_DMA_IN_DSCR_ERR_CH*n*_INT. (R/W)

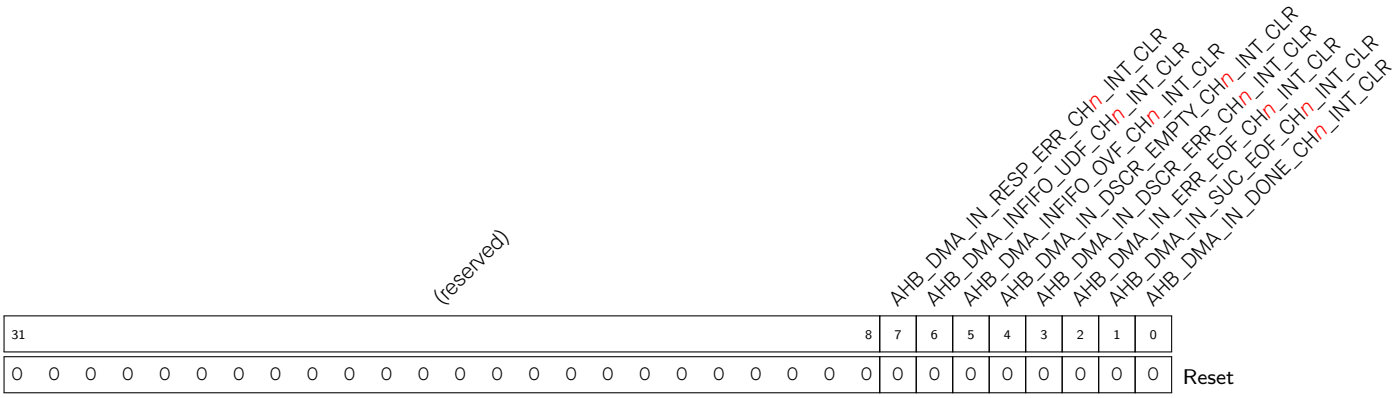
AHB_DMA_IN_DSCR_EMPTY_CH*n*_INT_ENA Write 1 to enable AHB_DMA_IN_DSCR_EMPTY_CH*n*_INT. (R/W)

AHB_DMA_INFIFO_OVF_CH*n*_INT_ENA Write 1 to enable AHB_DMA_INFIFO_OVF_CH*n*_INT. (R/W)

AHB_DMA_INFIFO_UDF_CH*n*_INT_ENA Write 1 to enable AHB_DMA_INFIFO_UDF_CH*n*_INT. (R/W)

AHB_DMA_IN_RESP_ERR_CH*n*_INT_ENA Write 1 to enable AHB_DMA_IN_RESP_ERR_CH*n*_INT. (R/W)

Register 6.4. AHB_DMA_IN_INT_CLR_CH*n*_REG (*n*: 0-2) (0x000C+0x10n*)**



AHB_DMA_IN_DONE_CH*n*_INT_CLR Write 1 to clear AHB_DMA_IN_DONE_CH*n*_INT. (WT)

AHB_DMA_IN_SUC_EOF_CH*n*_INT_CLR Write 1 to clear AHB_DMA_IN_SUC_EOF_CH*n*_INT. (WT)

AHB_DMA_IN_ERR_EOF_CH*n*_INT_CLR Write 1 to clear AHB_DMA_IN_ERR_EOF_CH*n*_INT. (WT)

AHB_DMA_IN_DSCR_ERR_CH*n*_INT_CLR Write 1 to clear AHB_DMA_IN_DSCR_ERR_CH*n*_INT. (WT)

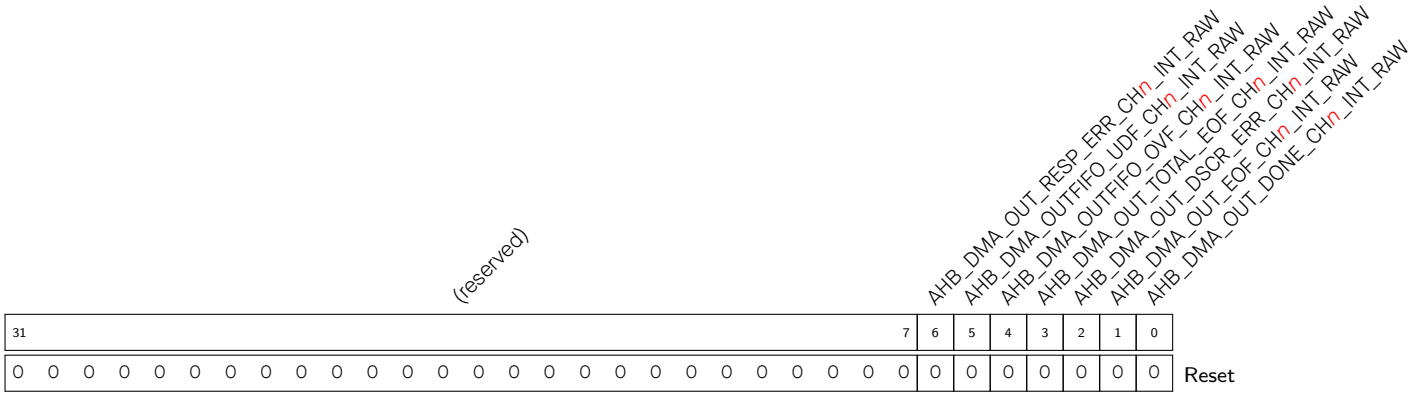
AHB_DMA_IN_DSCR_EMPTY_CH*n*_INT_CLR Write 1 to clear AHB_DMA_IN_DSCR_EMPTY_CH*n*_INT. (WT)

AHB_DMA_INFIFO_OVF_CH*n*_INT_CLR Write 1 to clear AHB_DMA_INFIFO_OVF_CH*n*_INT. (WT)

AHB_DMA_INFIFO_UDF_CH*n*_INT_CLR Write 1 to clear AHB_DMA_INFIFO_UDF_CH*n*_INT. (WT)

AHB_DMA_IN_RESP_ERR_CH*n*_INT_CLR Write 1 to clear AHB_DMA_IN_RESP_ERR_CH*n*_INT. (WT)

Register 6.5. AHB_DMA_OUT_INT_RAW_CH*n*_REG (*n*: 0-2) (0x0030+0x10n*)**



AHB_DMA_OUT_DONE_CH*n*_INT_RAW The raw interrupt status of AHB_DMA_OUT_DONE_CH*n*_INT. (R/WTC/SS)

AHB_DMA_OUT_EOF_CH*n*_INT_RAW The raw interrupt status of AHB_DMA_OUT_EOF_CH*n*_INT. (R/WTC/SS)

AHB_DMA_OUT_DSCR_ERR_CH*n*_INT_RAW The raw interrupt status of AHB_DMA_OUT_DSCR_ERR_CH*n*_INT. (R/WTC/SS)

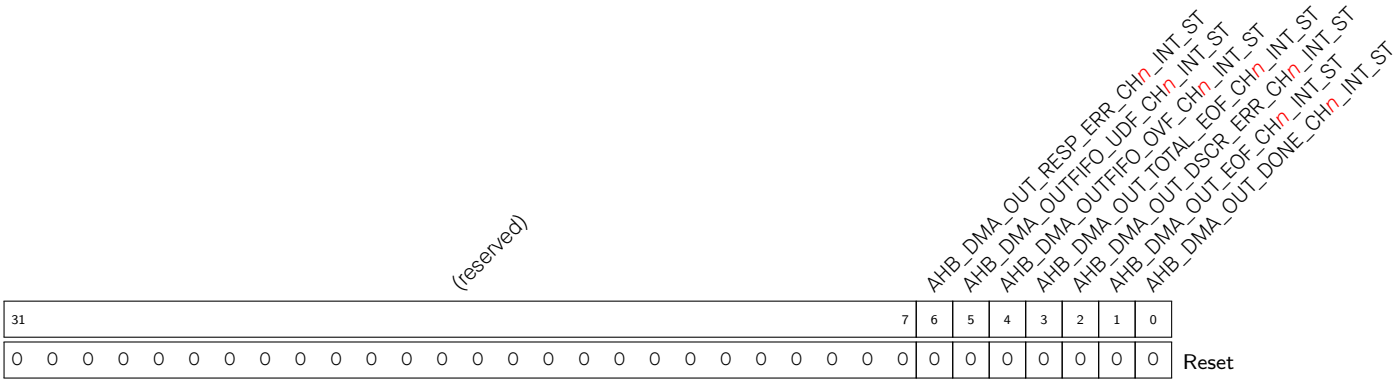
AHB_DMA_OUT_TOTAL_EOF_CH*n*_INT_RAW The raw interrupt status of AHB_DMA_OUT_TOTAL_EOF_CH*n*_INT. (R/WTC/SS)

AHB_DMA_OUTFIFO_OVF_CH*n*_INT_RAW The raw interrupt status of AHB_DMA_OUTFIFO_OVF_CH*n*_INT. (R/WTC/SS)

AHB_DMA_OUTFIFO_UDF_CH*n*_INT_RAW The raw interrupt status of AHB_DMA_OUTFIFO_UDF_CH*n*_INT. (R/WTC/SS)

AHB_DMA_OUT_RESP_ERR_CH*n*_INT_RAW The raw interrupt status of AHB_DMA_OUT_RESP_ERR_CH*n*_INT. (R/WTC/SS)

Register 6.6. AHB_DMA_OUT_INT_ST_CH*n*_REG (*n*: 0-2) (0x0034+0x10n*)**



AHB_DMA_OUT_DONE_CH*n*_INT_ST The masked interrupt status of AHB_DMA_OUT_DONE_CH*n*_INT. (RO)

AHB_DMA_OUT_EOF_CH*n*_INT_ST The masked interrupt status of AHB_DMA_OUT_EOF_CH*n*_INT. (RO)

AHB_DMA_OUT_DSCR_ERR_CH*n*_INT_ST The masked interrupt status of AHB_DMA_OUT_DSCR_ERR_CH*n*_INT. (RO)

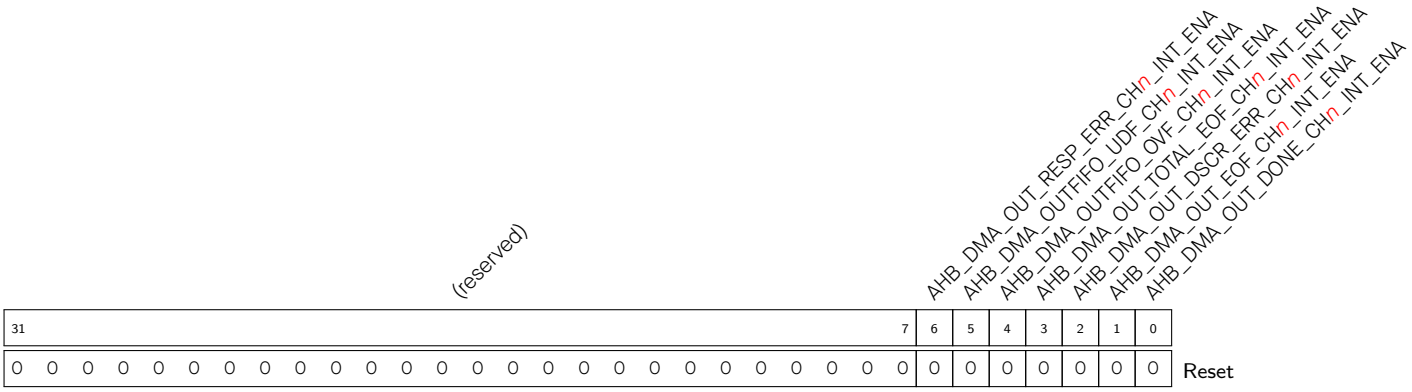
AHB_DMA_OUT_TOTAL_EOF_CH*n*_INT_ST The masked interrupt status of AHB_DMA_OUT_TOTAL_EOF_CH*n*_INT. (RO)

AHB_DMA_OUTFIFO_OVF_CH*n*_INT_ST The masked interrupt status of AHB_DMA_OUTFIFO_OVF_CH*n*_INT. (RO)

AHB_DMA_OUTFIFO_UDF_CH*n*_INT_ST The masked interrupt status of AHB_DMA_OUTFIFO_UDF_CH*n*_INT. (RO)

AHB_DMA_OUT_RESP_ERR_CH*n*_INT_ST The masked interrupt status of AHB_DMA_OUT_RESP_ERR_CH*n*_INT. (RO)

Register 6.7. AHB_DMA_OUT_INT_ENA_CH*n*_REG (*n*: 0-2) (0x0038+0x10**n*)



AHB_DMA_OUT_DONE_CH*n*_INT_ENA Write 1 to enable AHB_DMA_OUT_DONE_CH*n*_INT. (R/W)

AHB_DMA_OUT_EOF_CH*n*_INT_ENA Write 1 to enable AHB_DMA_OUT_EOF_CH*n*_INT. (R/W)

AHB_DMA_OUT_DSCR_ERR_CH*n*_INT_ENA Write 1 to enable AHB_DMA_OUT_DSCR_ERR_CH*n*_INT. (R/W)

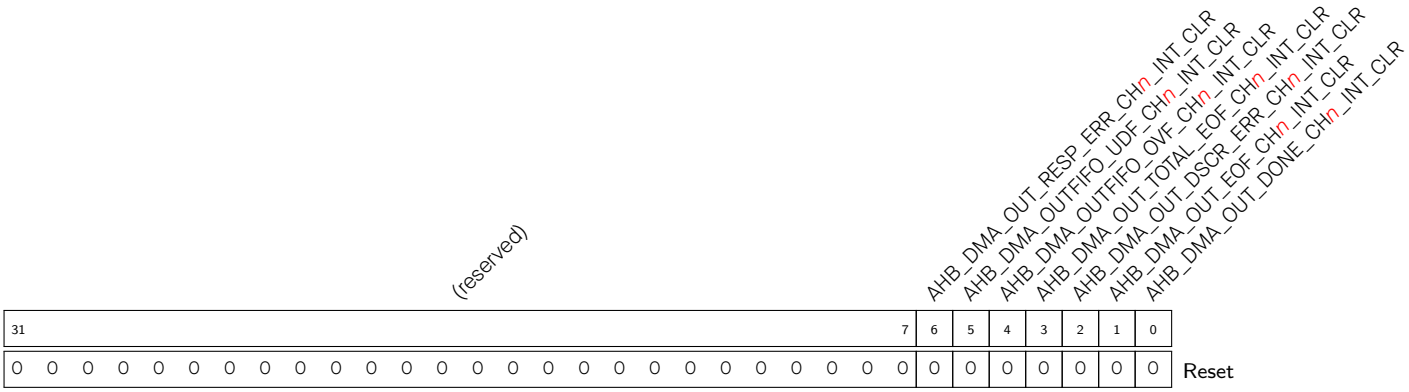
AHB_DMA_OUT_TOTAL_EOF_CH*n*_INT_ENA Write 1 to enable AHB_DMA_OUT_TOTAL_EOF_CH*n*_INT. (R/W)

AHB_DMA_OUTFIFO_OVF_CH*n*_INT_ENA Write 1 to enable AHB_DMA_OUTFIFO_OVF_CH*n*_INT. (R/W)

AHB_DMA_OUTFIFO_UDF_CH*n*_INT_ENA Write 1 to enable AHB_DMA_OUTFIFO_UDF_CH*n*_INT. (R/W)

AHB_DMA_OUT_RESP_ERR_CH*n*_INT_ENA Write 1 to enable AHB_DMA_OUT_RESP_ERR_CH*n*_INT. (R/W)

Register 6.8. AHB_DMA_OUT_INT_CLR_CH*n*_REG (*n*: 0-2) (0x003C+0x10**n*)



AHB_DMA_OUT_DONE_CH*n*_INT_CLR Write 1 to clear AHB_DMA_OUT_DONE_CH*n*_INT. (WT)

AHB_DMA_OUT_EOF_CH*n*_INT_CLR Write 1 to clear AHB_DMA_OUT_EOF_CH*n*_INT. (WT)

AHB_DMA_OUT_DSCR_ERR_CH*n*_INT_CLR Write 1 to clear AHB_DMA_OUT_DSCR_ERR_CH*n*_INT. (WT)

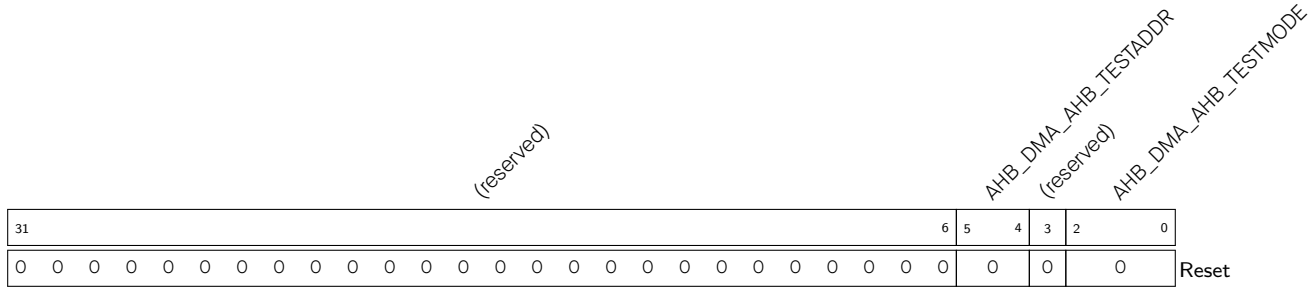
AHB_DMA_OUT_TOTAL_EOF_CH*n*_INT_CLR Write 1 to clear AHB_DMA_OUT_TOTAL_EOF_CH*n*_INT. (WT)

AHB_DMA_OUTFIFO_OVF_CH*n*_INT_CLR Write 1 to clear AHB_DMA_OUTFIFO_OVF_CH*n*_INT. (WT)

AHB_DMA_OUTFIFO_UDF_CH*n*_INT_CLR Write 1 to clear AHB_DMA_OUTFIFO_UDF_CH*n*_INT. (WT)

AHB_DMA_OUT_RESP_ERR_CH*n*_INT_CLR Write 1 to clear AHB_DMA_OUT_RESP_ERR_CH*n*_INT. (WT)

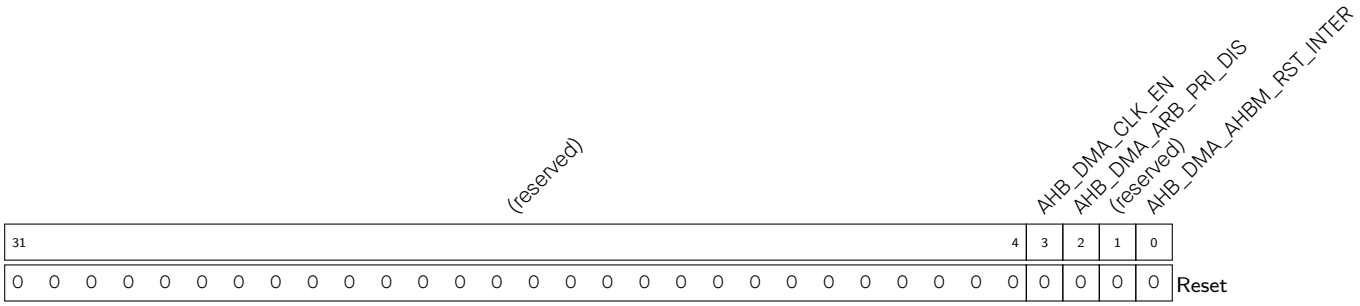
Register 6.9. AHB_DMA_AHB_TEST_REG (0x0060)



AHB_DMA_AHB_TESTMODE Reserved. (R/W)

AHB_DMA_AHB_TESTADDR Reserved. (R/W)

Register 6.10. AHB_DMA_MISC_CONF_REG (0x0064)



AHB_DMA_AHBM_RST_INTER Write 1 and then 0 to reset the internal AHB FSM. (R/W)

AHB_DMA_ARB_PRI_DIS Configures whether to disable the priority arbitration.

0: Enable

1: Disable

(R/W)

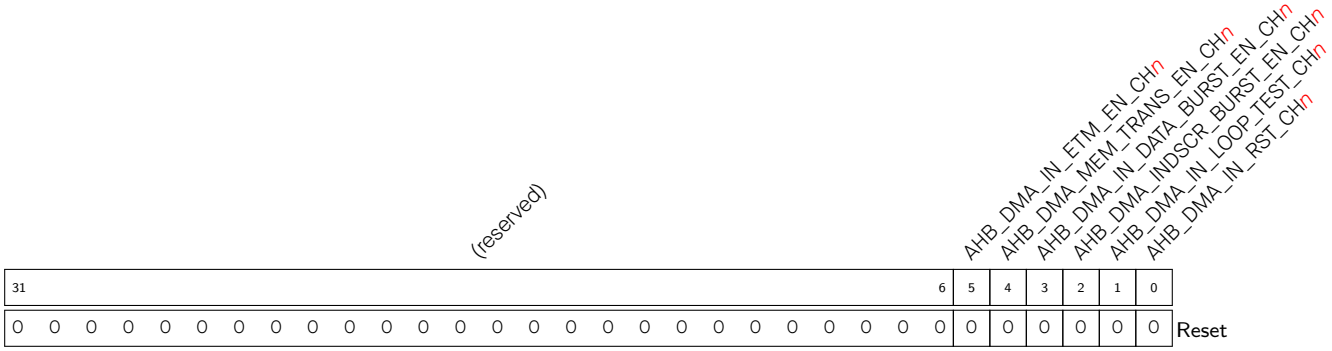
AHB_DMA_CLK_EN Configures AHB DMA clock gating.

0: Support clock only when the application writes registers

1: Always force the clock on for registers

(R/W)

Register 6.11. AHB_DMA_IN_CONF0_CH*n*_REG (*n*: 0-2) (0x0070+0xC0*n*)



AHB_DMA_IN_RST_CH*n* Write 1 and then 0 to reset RX channel *n* FSM and RX FIFO pointer.(R/W)

AHB_DMA_IN_LOOP_TEST_CH*n* Reserved. (R/W)

AHB_DMA_INDSCR_BURST_EN_CH*n* Configures whether to enable INCR burst transfer for RX channel *n* to read descriptors.

0: Disable

1: Enable

(R/W)

AHB_DMA_IN_DATA_BURST_EN_CH*n* Configures whether to enable INCR burst transfer for RX channel *n*.

0: Disable

1: Enable

(R/W)

AHB_DMA_MEM_TRANS_EN_CH*n* (*n* = 1 ~2) Configures whether to enable memory-to-memory data transfer. Channel 0 does not support this feature.

0: Disable

1: Enable

(R/W)

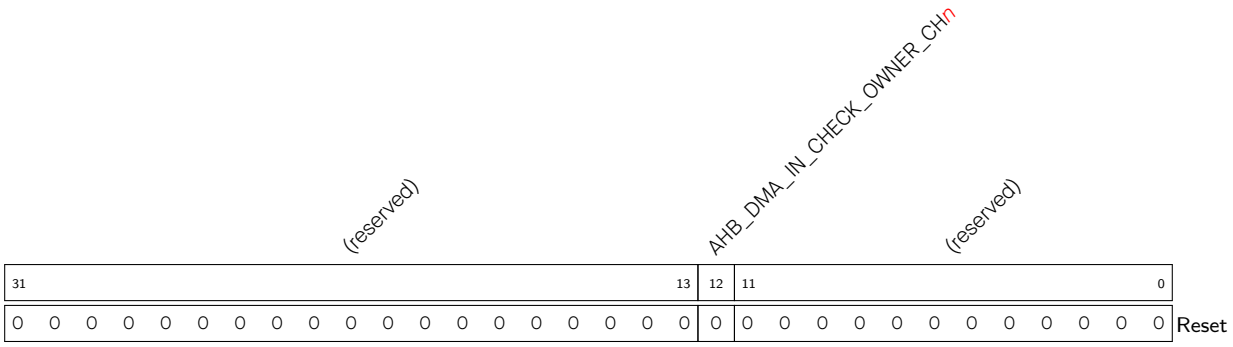
AHB_DMA_IN_ETM_EN_CH*n* Configures whether to enable ETM control for RX channel*n*.

0: Disable

1: Enable

(R/W)

Register 6.12. AHB_DMA_IN_CONF1_CH*n*_REG (*n*: 0-2) (0x0074+0xC0*n*)



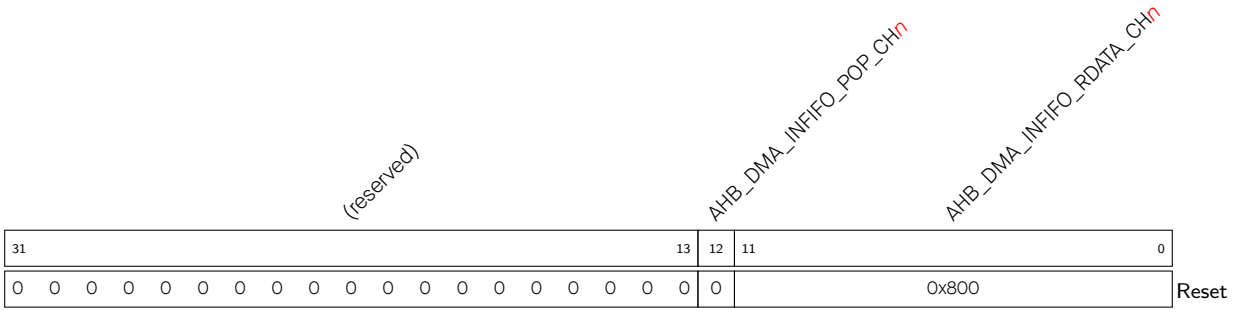
AHB_DMA_IN_CHECK_OWNER_CH*n* Configures whether to enable owner bit check for RX channel *n*.

0: Disable

1: Enable

(R/W)

Register 6.13. AHB_DMA_IN_POP_CH*n*_REG (*n*: 0-2) (0x007C+0xC0*n*)



AHB_DMA_INFIFO_RDATA_CH*n* Represents the data popped from AHB DMA RX FIFO. (RO)

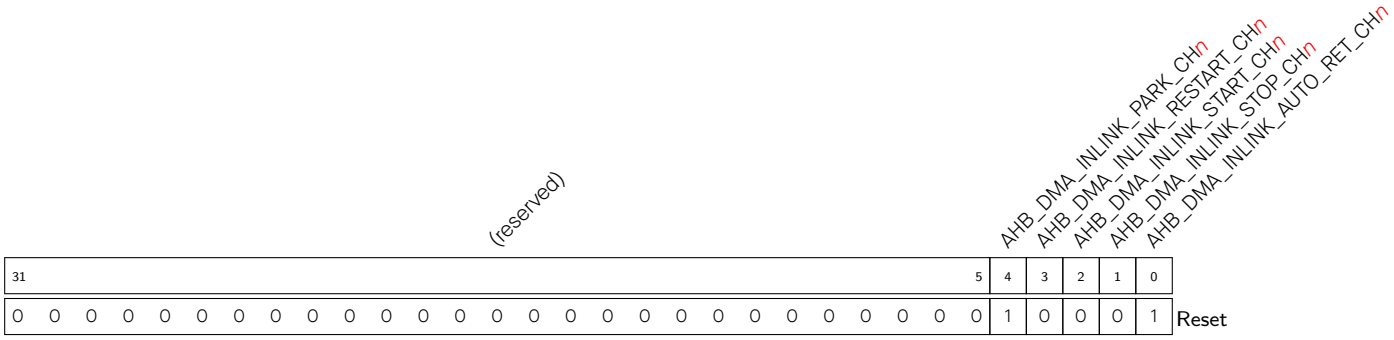
AHB_DMA_INFIFO_POP_CH*n* Configures whether to pop data from AHB DMA RX FIFO.

0: Invalid. No effect

1: Pop

(WT)

Register 6.14. AHB_DMA_IN_LINK_CH*n*_REG (*n*: 0-2) (0x0080+0xC0n*)**



AHB_DMA_INLINK_AUTO_RET_CH*n* Configures whether to return to the current receive descriptor's address when there are some errors in current receiving data.

0: Not return

1: Return

(R/W)

AHB_DMA_INLINK_STOP_CH*n* Configures whether to stop RX channel *n* from receiving data.

0: Invalid. No effect

1: Stop

(WT)

AHB_DMA_INLINK_START_CH*n* Configures whether to enable RX channel *n* for data transfer.

0: Disable

1: Enable

(WT)

AHB_DMA_INLINK_RESTART_CH*n* Configures whether to restart RX channel *n* for AHB DMA transfer.

0: Invalid. No effect

1: Restart

(WT)

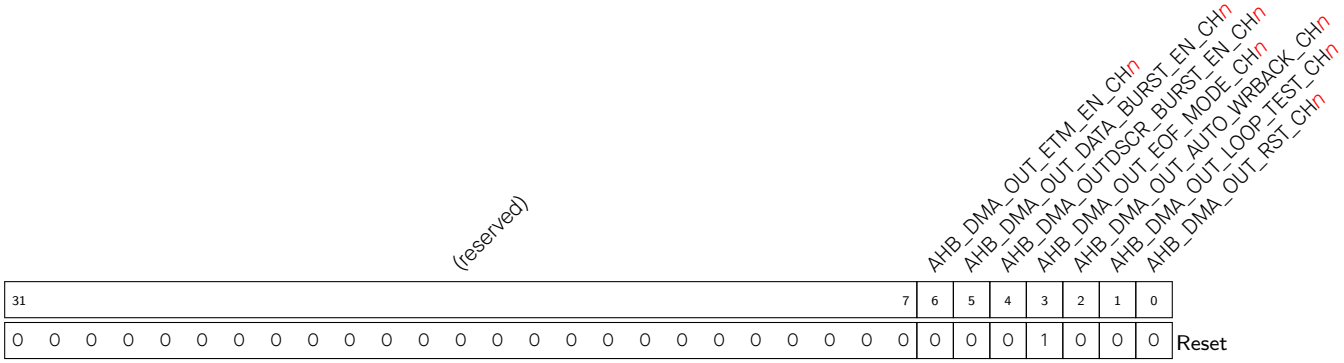
AHB_DMA_INLINK_PARK_CH*n* Represents the status of the receive descriptor's FSM.

0: Running

1: Idle

(RO)

Register 6.15. AHB_DMA_OUT_CONF0_CH*n*_REG (*n*: 0-2) (0x00D0+0xC0n*)**



AHB_DMA_OUT_RST_CH*n* Configures the reset state of TX channel *n* FSM and TX FIFO pointer.

0: Release reset

1: Reset

(R/W)

AHB_DMA_OUT_LOOP_TEST_CH*n* Reserved. (R/W)

AHB_DMA_OUT_AUTO_WRBACK_CH*n* Configures whether to enable automatic outlink write-back when all the data in TX FIFO has been transmitted.

0: Disable

1: Enable

(R/W)

AHB_DMA_OUT_EOF_MODE_CH*n* Configures when to generate EOF flag.

0: EOF flag for TX channel *n* is generated when data to be transmitted has been pushed into FIFO in AHB DMA.

1: EOF flag for TX channel *n* is generated when data to be transmitted has been popped from FIFO in AHB DMA.

(R/W)

AHB_DMA_OUTDSCR_BURST_EN_CH*n* Configures whether to enable INCR burst transfer for TX channel *n* reading descriptors.

0: Disable

1: Enable

(R/W)

AHB_DMA_OUT_DATA_BURST_EN_CH*n* Configures whether to enable INCR burst transfer for TX channel *n*.

0: Disable

1: Enable

(R/W)

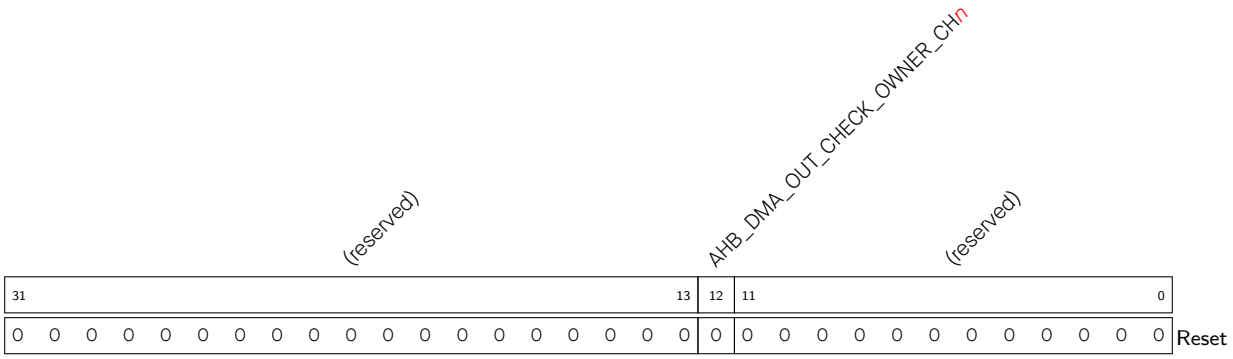
AHB_DMA_OUT_ETM_EN_CH*n* Configures whether to enable ETM control for TX channel *n*.

0: Disable

1: Enable

(R/W)

Register 6.16. AHB_DMA_OUT_CONF1_CH*n*_REG (*n*: 0-2) (0x00D4+0xC0*n*)



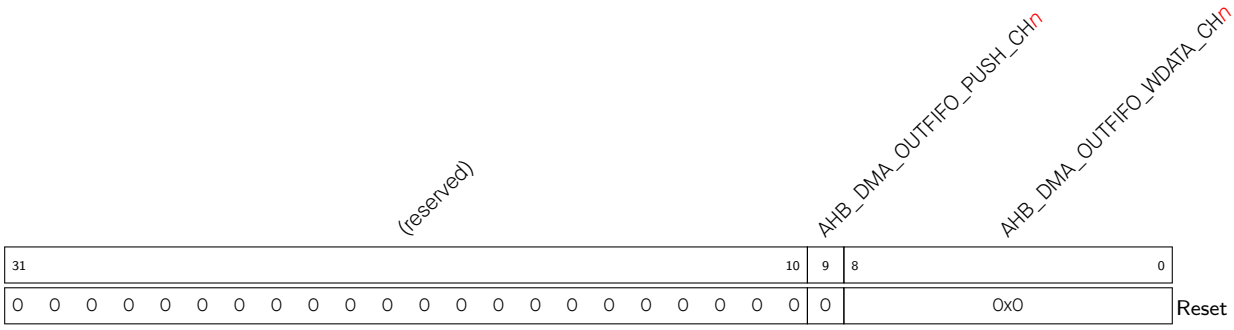
AHB_DMA_OUT_CHECK_OWNER_CH*n* Configures whether to enable owner bit check for TX channel *n*.

0: Disable

1: Enable

(R/W)

Register 6.17. AHB_DMA_OUT_PUSH_CH*n*_REG (*n*: 0-2) (0x00DC+0xC0*n*)



AHB_DMA_OUTFIFO_WDATA_CH*n* Represents the data that need to be pushed into AHB DMA TX FIFO. (R/W)

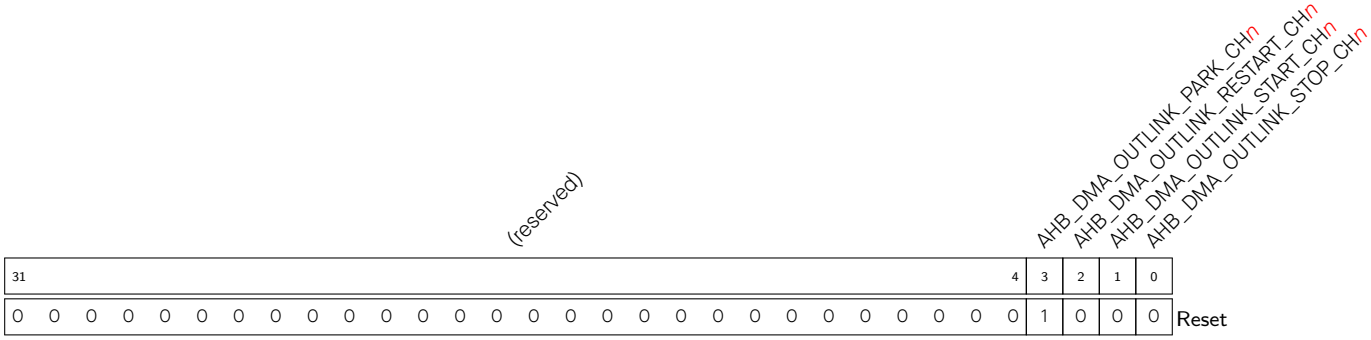
AHB_DMA_OUTFIFO_PUSH_CH*n* Configures whether to push data into AHB DMA TX FIFO.

0: Invalid. No effect

1: Push

(WT)

Register 6.18. AHB_DMA_OUT_LINK_CH*n*_REG (*n*: 0-2) (0x00E0+0xC0**n*)



AHB_DMA_OUTLINK_STOP_CH*n* Configures whether to stop TX channel *n* from transmitting data.

0: Invalid. No effect

1: Stop

(WT)

AHB_DMA_OUTLINK_START_CH*n* Configures whether to enable TX channel *n* for data transfer.

0: Disable

1: Enable

(WT)

AHB_DMA_OUTLINK_RESTART_CH*n* Configures whether to restart TX channel *n* for AHB DMA transfer.

0: Invalid. No effect

1: Restart

(WT)

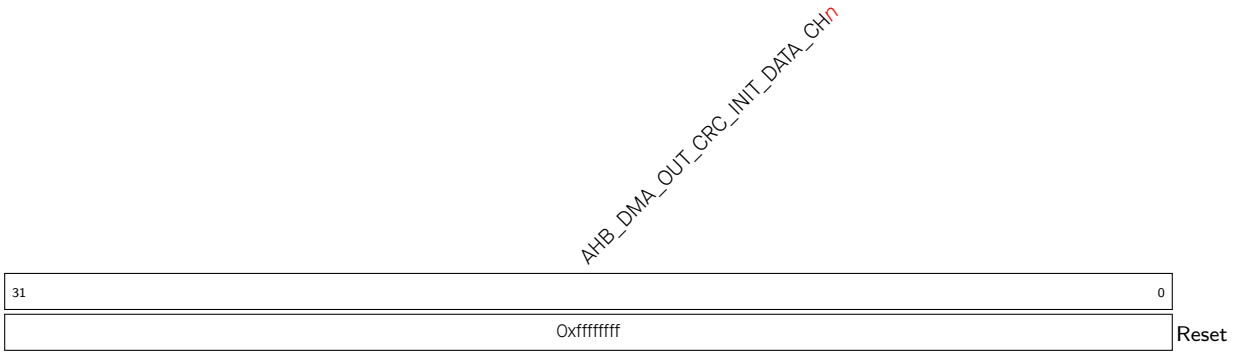
AHB_DMA_OUTLINK_PARK_CH*n* Represents the status of the transmit descriptor's FSM.

0: Running

1: Idle

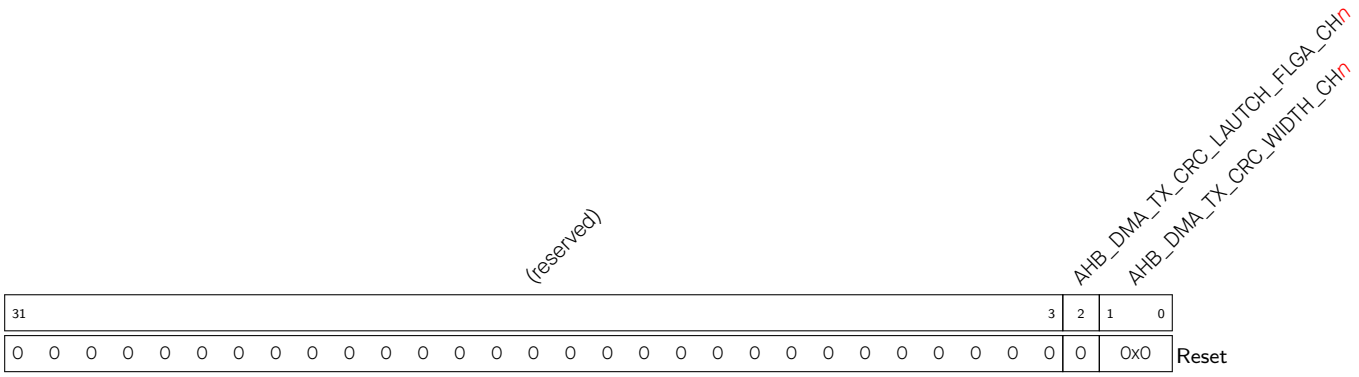
(RO)

Register 6.19. AHB_DMA_OUT_CRC_INIT_DATA_CH*n*_REG (*n*: 0-2) (0x02BC+0x28**n*)



AHB_DMA_OUT_CRC_INIT_DATA_CH*n* Configures the CRC initial value for TX channel *n*. (R/W)

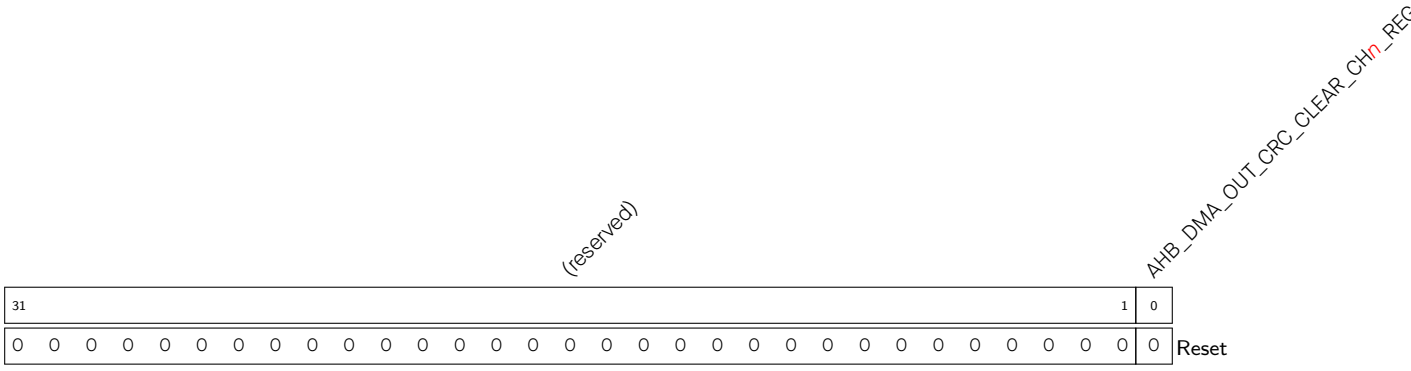
Register 6.20. AHB_DMA_TX_CRC_WIDTH_CH*n*_REG (*n*: 0-2) (0x02C0+0x28**n*)



AHB_DMA_TX_CRC_WIDTH_CH*n* Configures the CRC result width for TX channel *n*. 0: $\leq$ 8 bits
1: $\leq$ 16 bits
2: $\leq$ 24 bits
3: $\leq$ 32 bits
(R/W)

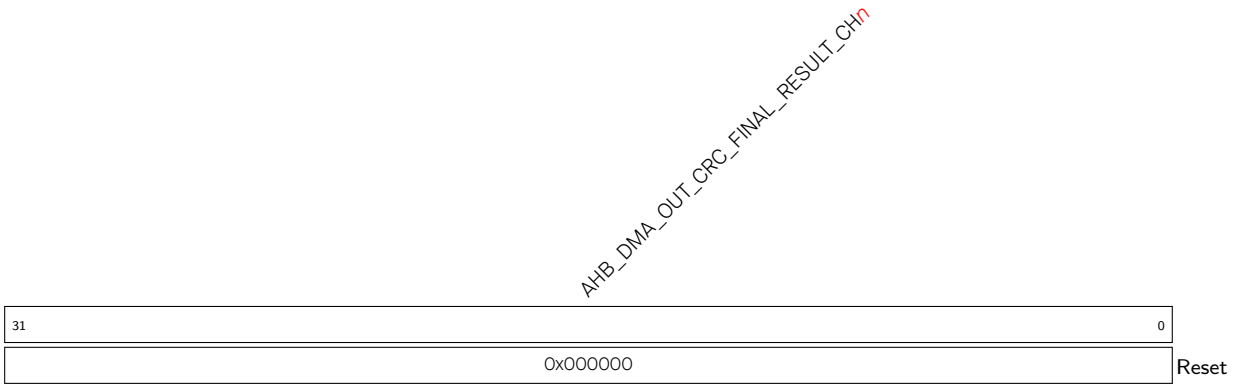
AHB_DMA_TX_CRC_LAUTCH_FLGA_CH*n* Write 1 and then 0 to latch the values of AHB_DMA_TX_CRC_EN_ADDR_CH*n*, AHB_DMA_TX_CRC_DATA_EN_ADDR_CH*n*, AHB_DMA_TX_CRC_EN_WR_DATA_CH*n*, and AHB_DMA_TX_CRC_DATA_EN_WR_DATA_CH*n*. (R/W)

Register 6.21. AHB_DMA_OUT_CRC_CLEAR_CH*n*_REG (*n*: 0-2) (0x02C4+0x28**n*)



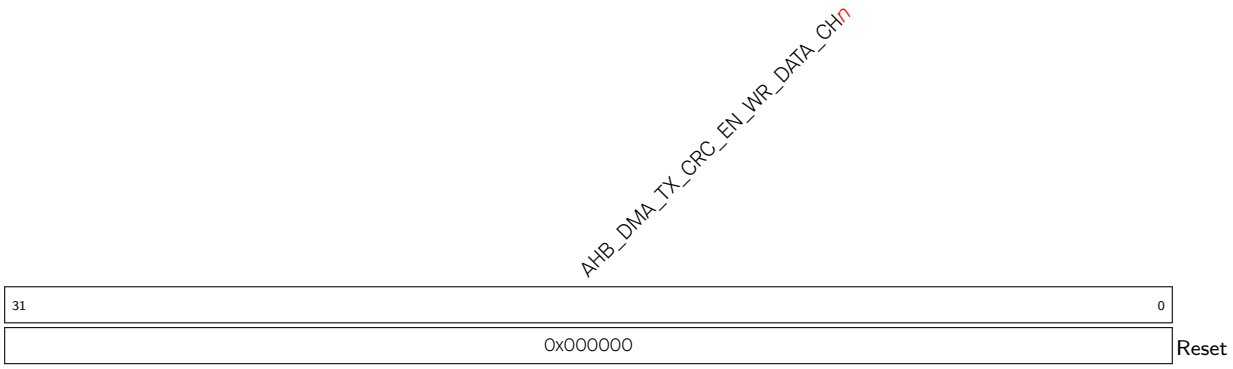
AHB_DMA_OUT_CRC_CLEAR_CH*n*_REG Reserved. (R/W)

Register 6.22. AHB_DMA_OUT_CRC_FINAL_RESULT_CH*n*_REG (*n*: 0-2) (0x02C8+0x28*n)



AHB_DMA_OUT_CRC_FINAL_RESULT_CH*n* Represents the CRC result for TX channel *n*. (RO)

Register 6.23. AHB_DMA_TX_CRC_EN_WR_DATA_CH*n*_REG (*n*: 0-2) (0x02CC+0x28*n)



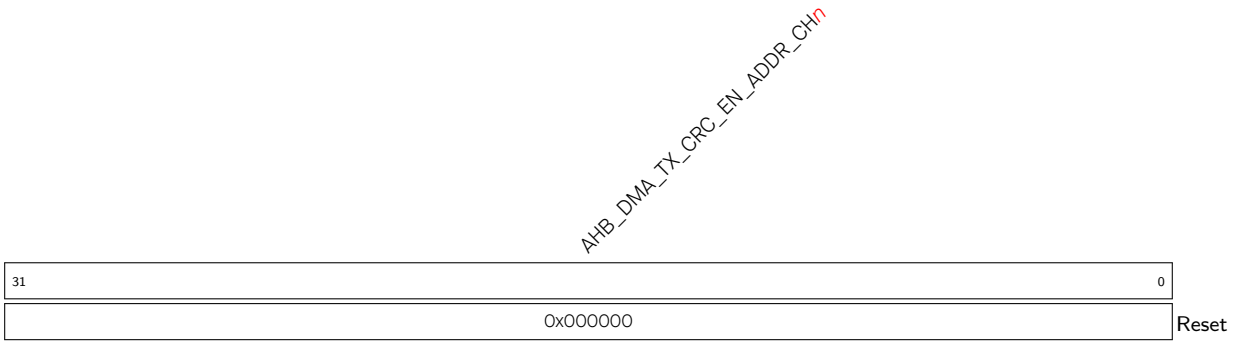
AHB_DMA_TX_CRC_EN_WR_DATA_CH*n* Configures whether to include each bit of the intermediate result in the CRC calculation matrix for TX channel *n*.

0: Not include

1: Include

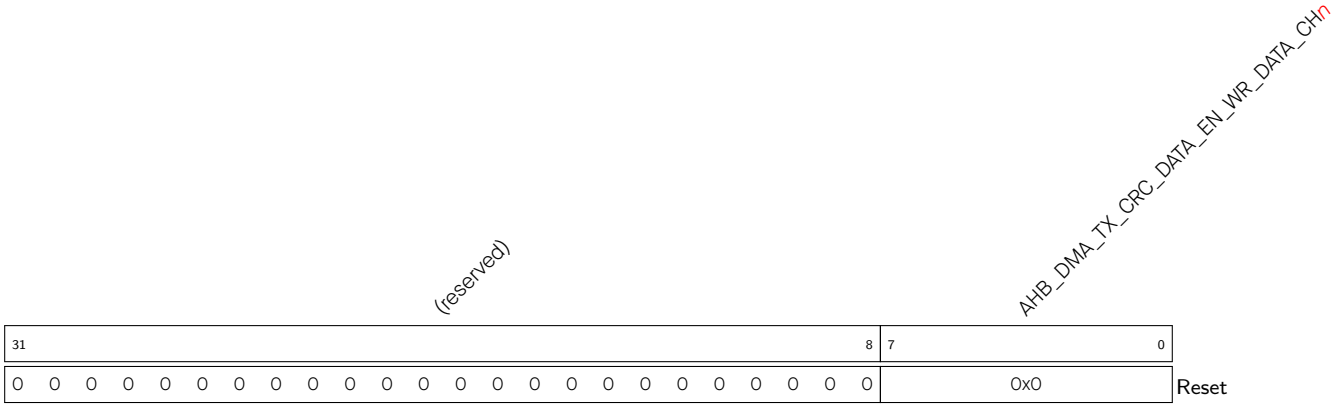
(R/W)

Register 6.24. AHB_DMA_TX_CRC_EN_ADDR_CH*n*_REG (*n*: 0-2) (0x02D0+0x28n*)**



AHB_DMA_TX_CRC_EN_ADDR_CH*n* Configures at which bit of the CRC result the AHB_DMA_TX_CRC_EN_WR_DATA_CH*n*_REG register targets for TX channel *n*. (R/W)

Register 6.25. AHB_DMA_TX_CRC_DATA_EN_WR_DATA_CH*n*_REG (*n*: 0-2) (0x02D4+0x28n*)**



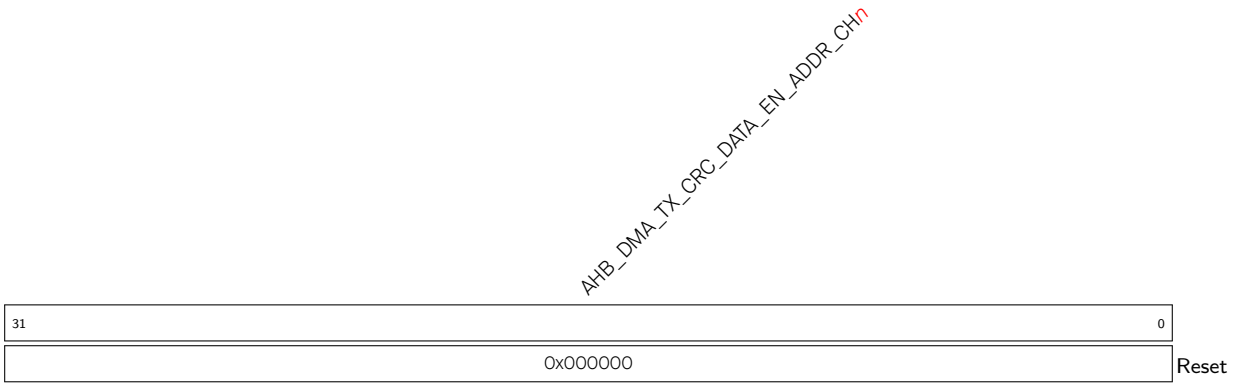
AHB_DMA_TX_CRC_DATA_EN_WR_DATA_CH*n* Configures whether to include each bit of the data in the CRC calculation matrix for TX channel *n*.

0: Not include

1: Include

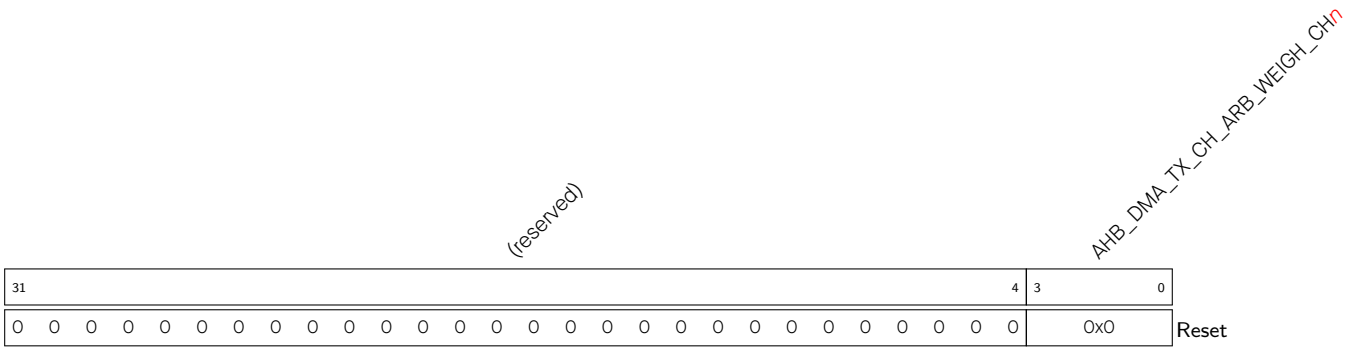
(R/W)

Register 6.26. AHB_DMA_TX_CRC_DATA_EN_ADDR_CH*n*_REG (*n*: 0-2) (0x02D8+0x28*n)



AHB_DMA_TX_CRC_DATA_EN_ADDR_CH*n* Configures at which bit of the CRC result the AHB_DMA_TX_CRC_DATA_EN_WR_DATA_CH*n*_REG register targets for TX channel *n*. (R/W)

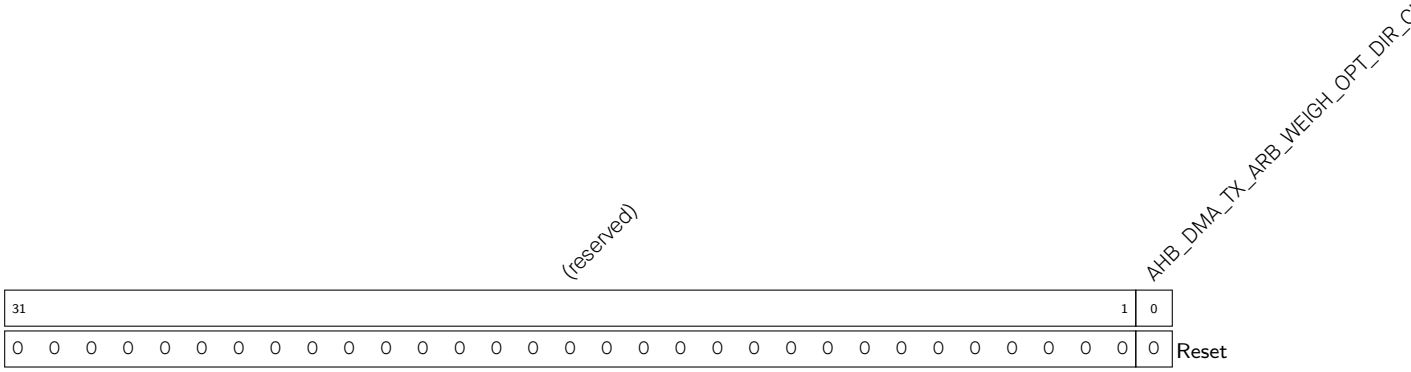
Register 6.27. AHB_DMA_TX_CH_ARB_WEIGH_CH*n*_REG (*n*: 0-2) (0x02DC+0x28*n)



AHB_DMA_TX_CH_ARB_WEIGH_CH*n* Configures the weight (i.e the number of tokens) of TX channel *n*.

Value range: 0 ~ 15. (R/W)

Register 6.28. AHB_DMA_TX_ARB_WEIGH_OPT_DIR_CH*n*_REG (*n*: 0-2) (0x02E0+0x28**n*)



AHB_DMA_TX_ARB_WEIGH_OPT_DIR_CH*n* Configures whether to enable weight optimization for TX channel *n*.

0: Enable

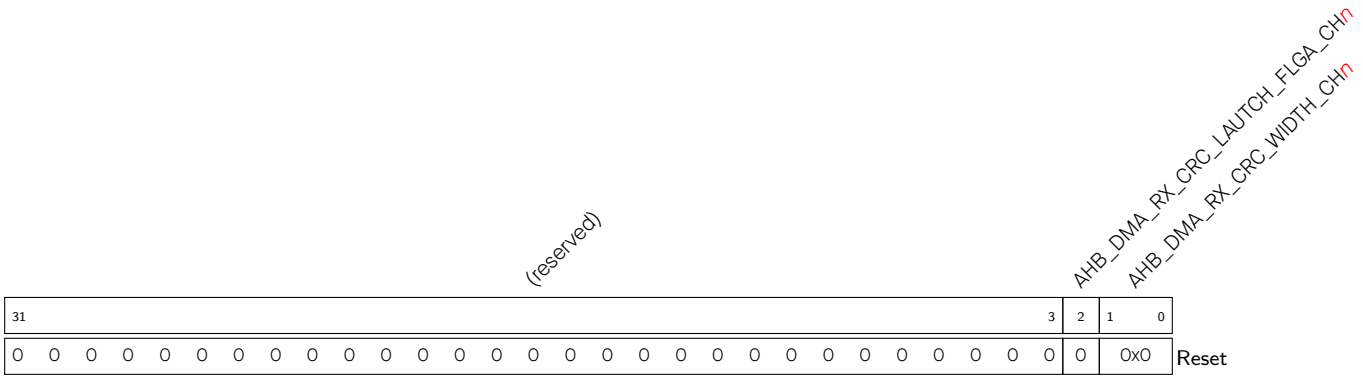
1: Disable (R/W)

Register 6.29. AHB_DMA_IN_CRC_INIT_DATA_CH*n*_REG (*n*: 0-2) (0x0334+0x28**n*)



AHB_DMA_IN_CRC_INIT_DATA_CH*n* Configures the CRC initial value for RX channel *n*. (R/W)

Register 6.30. AHB_DMA_RX_CRC_WIDTH_CH*n*_REG (*n*: 0-2) (0x0338+0x28**n*)



AHB_DMA_RX_CRC_WIDTH_CH*n* Configures the CRC result width for RX channel *n*.

0: $\leq$ 8 bits

1: $\leq$ 16 bits

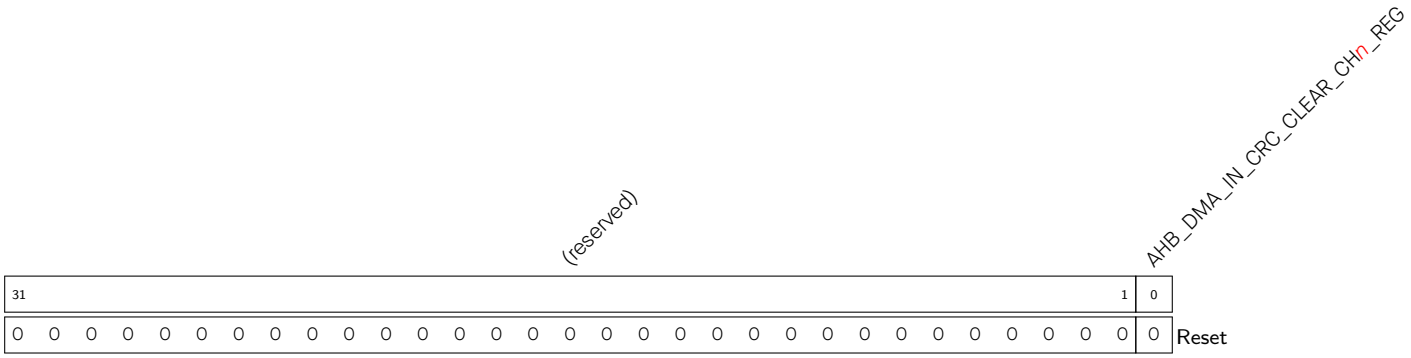
2: $\leq$ 24 bits

3: $\leq$ 32 bits

(R/W)

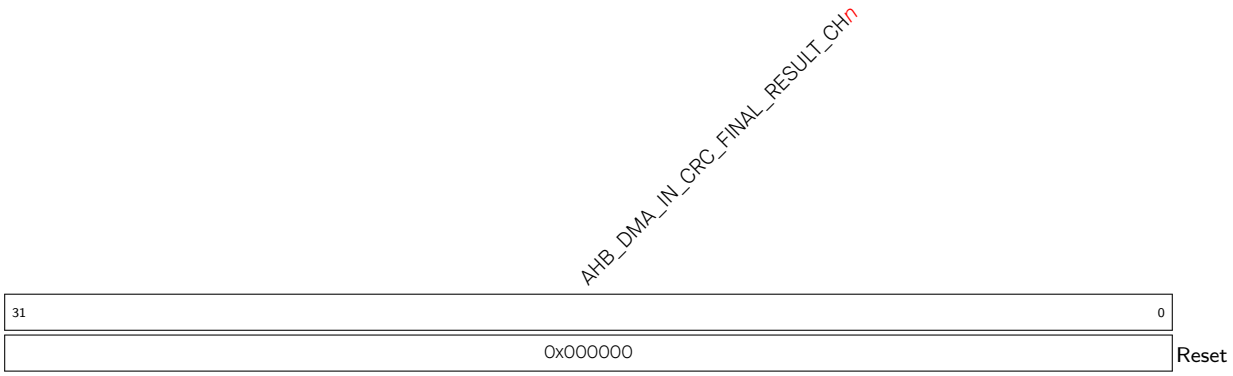
AHB_DMA_RX_CRC_LAUTCH_FLGA_CH*n* Write 1 and then 0 to latch the values of AHB_DMA_RX_CRC_EN_ADDR_CH*n*, AHB_DMA_RX_CRC_DATA_EN_ADDR_CH*n*, AHB_DMA_RX_CRC_EN_WR_DATA_CH*n*, and AHB_DMA_RX_CRC_DATA_EN_WR_DATA_CH*n*. (R/W)

Register 6.31. AHB_DMA_IN_CRC_CLEAR_CH*n*_REG (*n*: 0-2) (0x033C+0x28**n*)



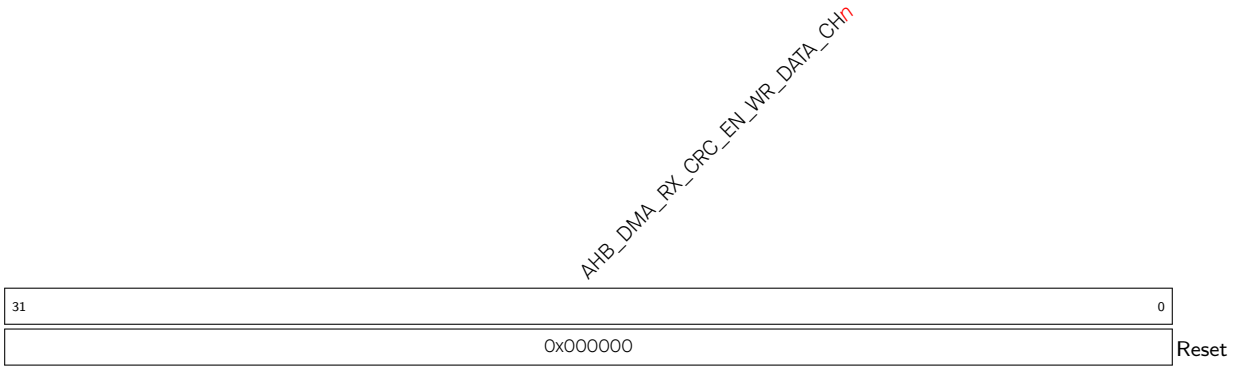
AHB_DMA_IN_CRC_CLEAR_CH*n*_REG Reserved. (R/W)

Register 6.32. AHB_DMA_IN_CRC_FINAL_RESULT_CH*n*_REG (*n*: 0-2) (0x0340+0x28**n*)



AHB_DMA_IN_CRC_FINAL_RESULT_CH*n* Represents the CRC result for RX channel *n*. (RO)

Register 6.33. AHB_DMA_RX_CRC_EN_WR_DATA_CH*n*_REG (*n*: 0-2) (0x0344+0x28**n*)



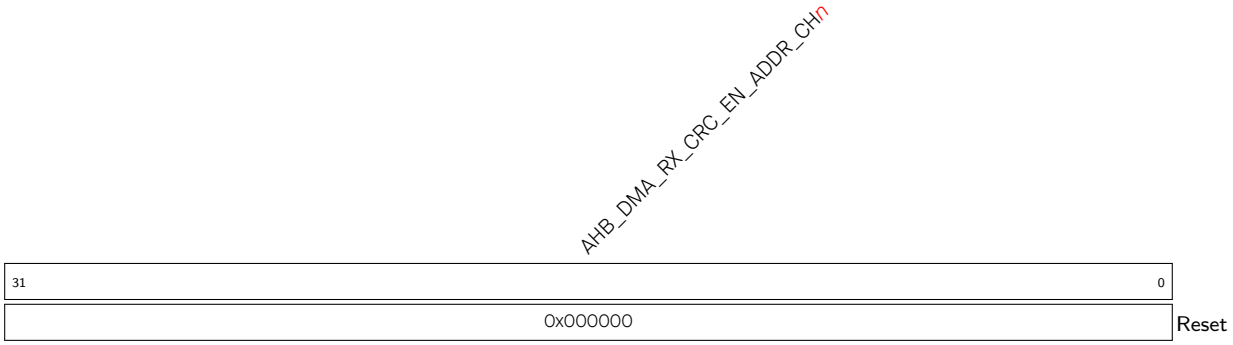
AHB_DMA_RX_CRC_EN_WR_DATA_CH*n* Configures whether to include each bit of the intermediate result in the CRC calculation matrix for RX channel *n*.

0: Not include

1: Include

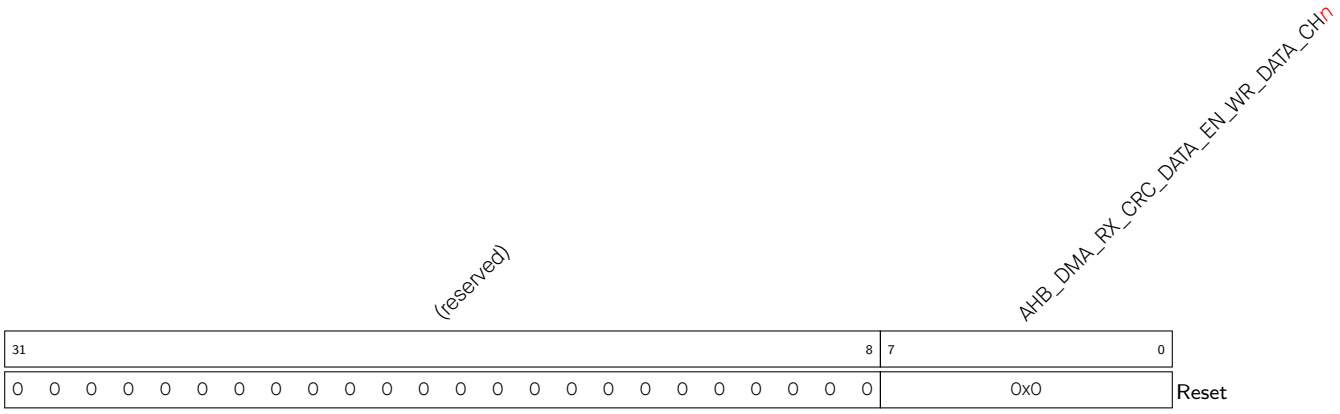
(R/W)

Register 6.34. AHB_DMA_RX_CRC_EN_ADDR_CH*n*_REG (*n*: 0-2) (0x0348+0x28n*)**



AHB_DMA_RX_CRC_EN_ADDR_CH*n* Configures at which bit of the CRC result the AHB_DMA_RX_CRC_EN_WR_DATA_CH*n*_REG register targets for RX channel *n*. (R/W)

Register 6.35. AHB_DMA_RX_CRC_DATA_EN_WR_DATA_CH*n*_REG (*n*: 0-2) (0x034C+0x28n*)**



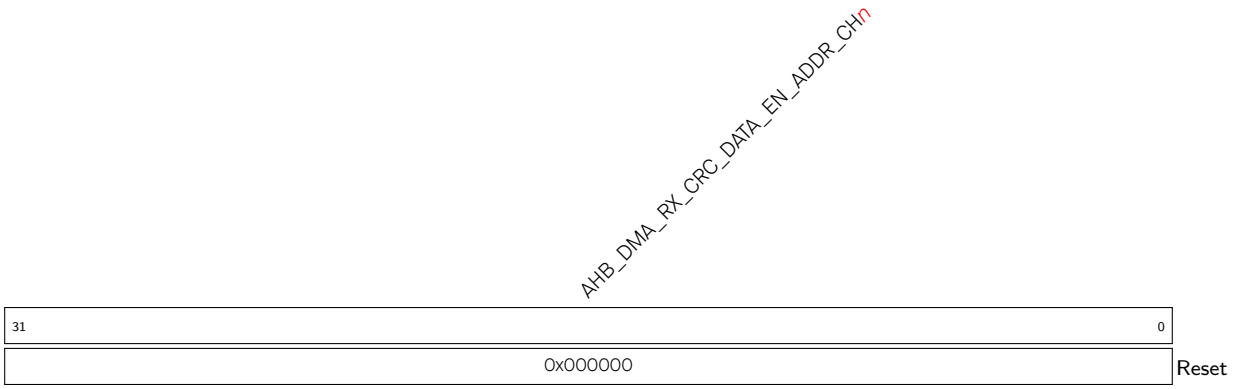
AHB_DMA_RX_CRC_DATA_EN_WR_DATA_CH*n* Configures whether to include each bit of the data in the CRC calculation matrix for RX channel *n*.

0: Not include

1: Include

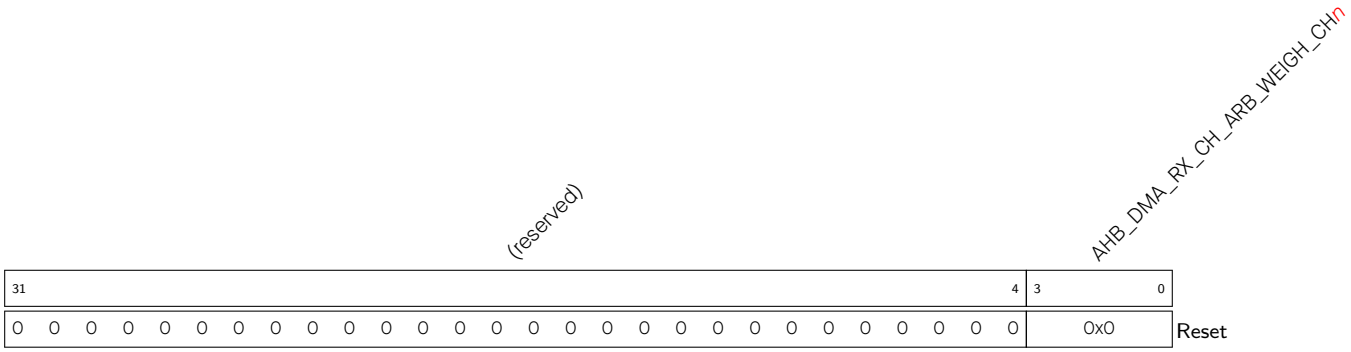
(R/W)

Register 6.36. AHB_DMA_RX_CRC_DATA_EN_ADDR_CH*n*_REG (*n*: 0-2) (0x0350+0x28**n*)



AHB_DMA_RX_CRC_DATA_EN_ADDR_CH*n* Configures at which bit of the CRC result the AHB_DMA_RX_CRC_DATA_EN_WR_DATA_CH*n*_REG register targets for RX channel *n*. (R/W)

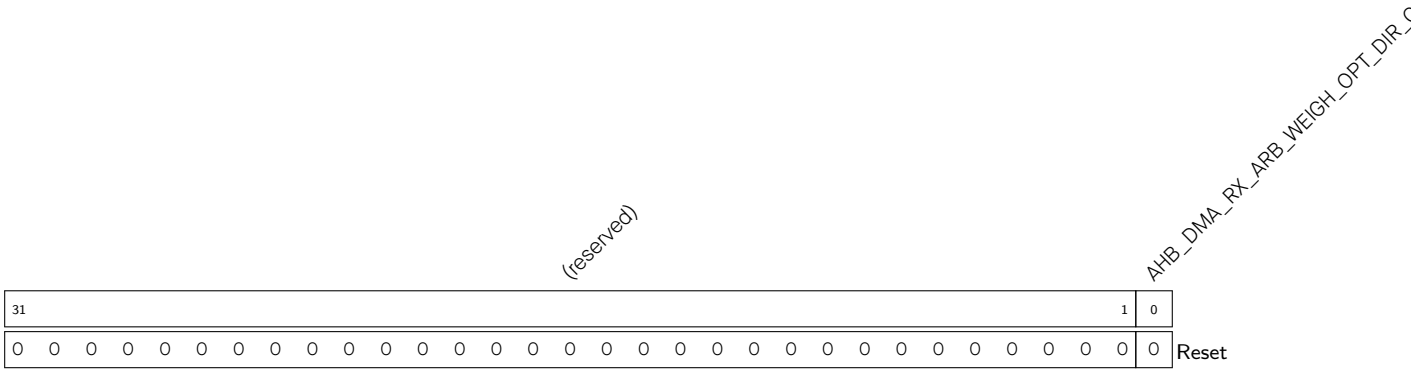
Register 6.37. AHB_DMA_RX_CH_ARB_WEIGH_CH*n*_REG (*n*: 0-2) (0x0354+0x28**n*)



AHB_DMA_RX_CH_ARB_WEIGH_CH*n* Configures the weight (i.e the number of tokens) of RX channel *n*.

Value range: 0 ~ 15. (R/W)

Register 6.38. AHB_DMA_RX_ARB_WEIGH_OPT_DIR_CH*n*_REG (*n*: 0-2) (0x0358+0x28**n*)

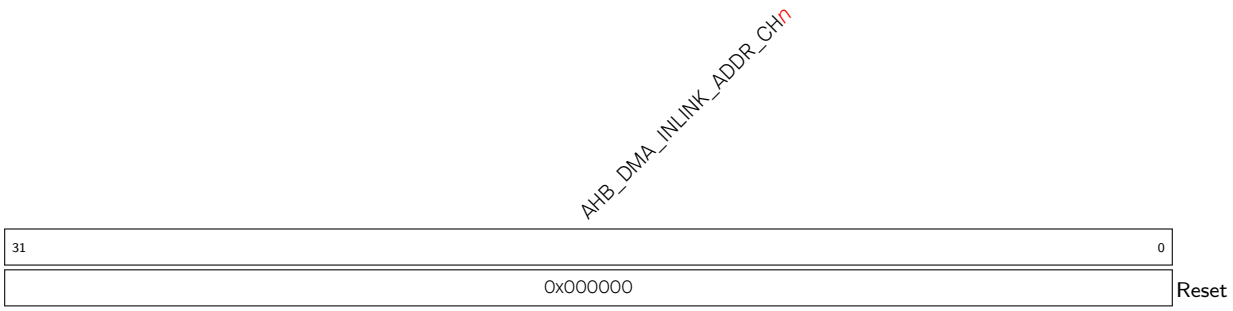


AHB_DMA_RX_ARB_WEIGH_OPT_DIR_CH*n* Configures whether to enable weight optimization for RX channel *n*.

0: Enable

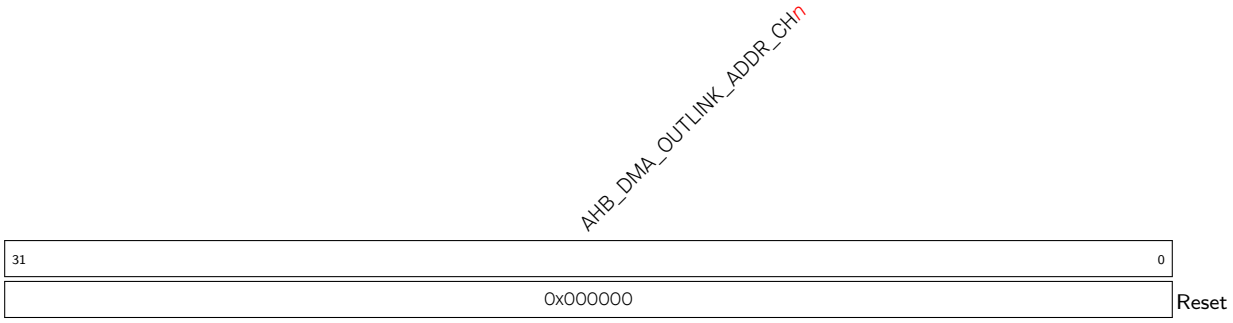
1: Disable (R/W)

Register 6.39. AHB_DMA_IN_LINK_ADDR_CH*n*_REG (*n*: 0-2) (0x03AC+0x4**n*)



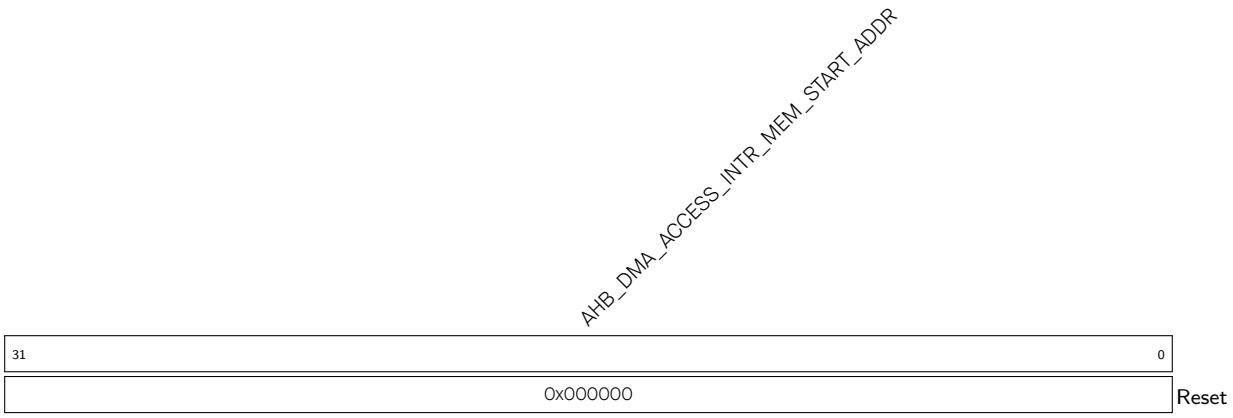
AHB_DMA_INLINK_ADDR_CH*n* Represents the first receive descriptor's address. (R/W)

Register 6.40. AHB_DMA_OUT_LINK_ADDR_CH*n*_REG (*n*: 0-2) (0x03B8+0x4**n*)



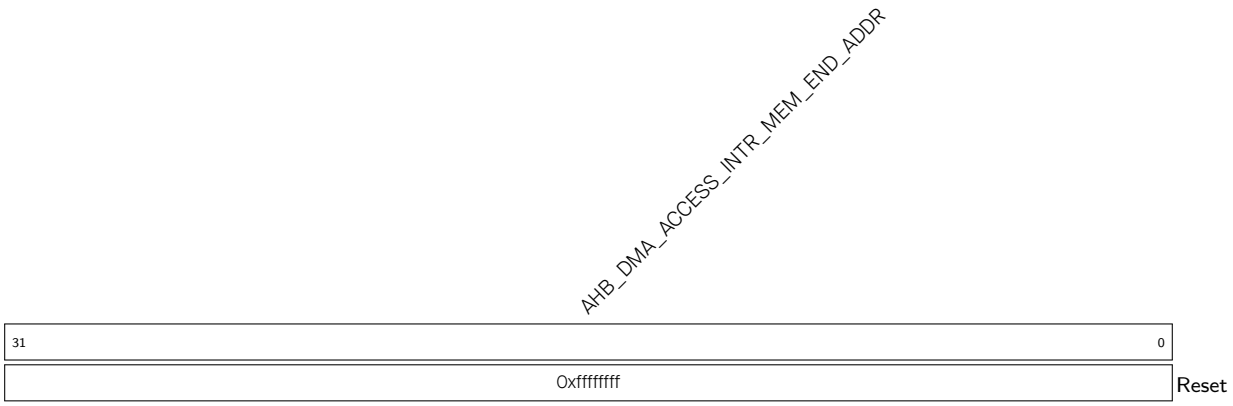
AHB_DMA_OUTLINK_ADDR_CH*n* Represents the first transmit descriptor's address. (R/W)

Register 6.41. AHB_DMA_INTR_MEM_START_ADDR_REG (0x03C4)



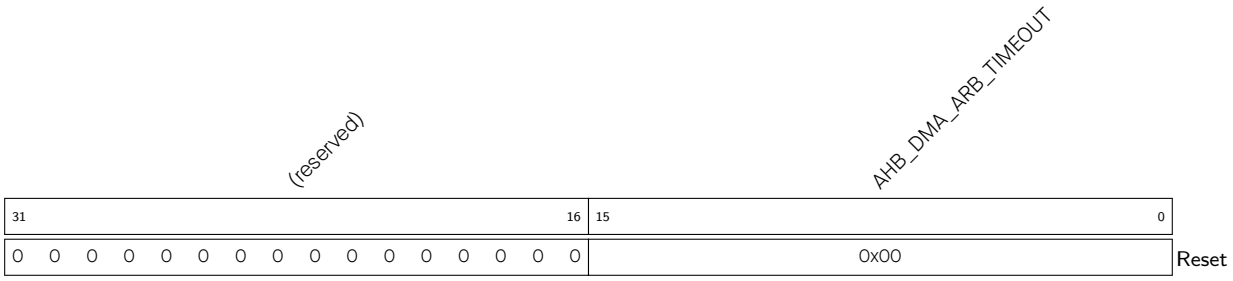
AHB_DMA_ACCESS_INTR_MEM_START_ADDR Configures the start address of accessible address space. (R/W)

Register 6.42. AHB_DMA_INTR_MEM_END_ADDR_REG (0x03C8)



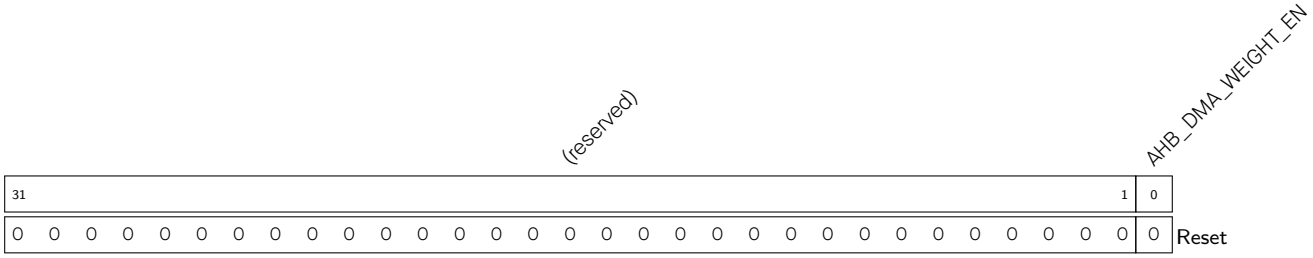
AHB_DMA_ACCESS_INTR_MEM_END_ADDR Configures the end address of accessible address space. (R/W)

Register 6.43. AHB_DMA_ARB_TIMEOUT_REG (0x03DC)



AHB_DMA_ARB_TIMEOUT Configures the time slot of weight arbitration. Measurement unit: AHB bus clock cycle. (R/W)

Register 6.44. AHB_DMA_WEIGHT_EN_REG (0x0400)



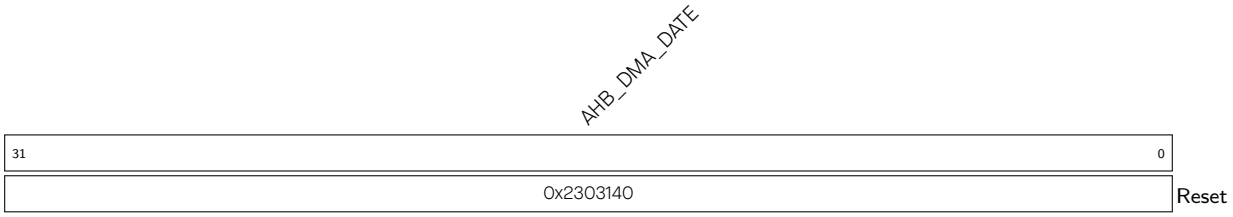
AHB_DMA_WEIGHT_EN Configures whether to enable weight arbitration.

0: Disable

1: Enable

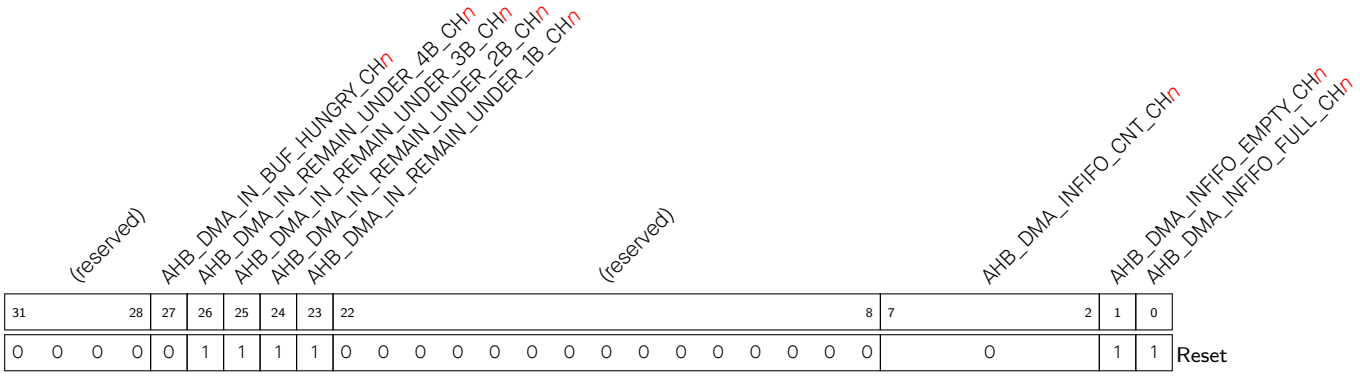
(R/W)

Register 6.45. AHB_DMA_DATE_REG (0x0068)



AHB_DMA_DATE Version control register. (R/W)

Register 6.46. AHB_DMA_INFIFO_STATUS_CH*n*_REG (*n*: 0-2) (0x0078+0xC0n*)**



AHB_DMA_INFIFO_FULL_CH*n* Represents whether L1 RX FIFO is full.

0: Not Full

1: Full

(RO)

AHB_DMA_INFIFO_EMPTY_CH*n* Represents whether L1 RX FIFO is empty.

0: Not empty

1: Empty

(RO)

AHB_DMA_INFIFO_CNT_CH*n* Represents the number of data bytes in L1 RX FIFO for RX channel *n*.

(RO)

AHB_DMA_IN_REMAIN_UNDER_1B_CH*n* Reserved. (RO)

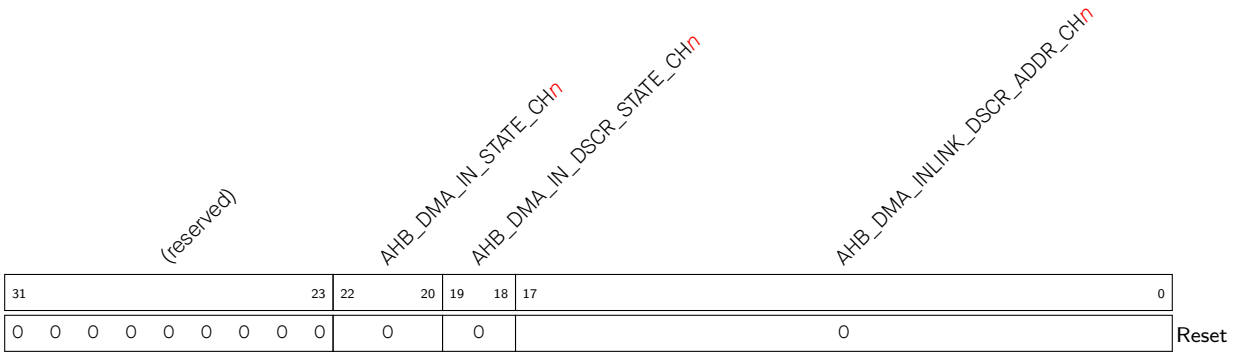
AHB_DMA_IN_REMAIN_UNDER_2B_CH*n* Reserved. (RO)

AHB_DMA_IN_REMAIN_UNDER_3B_CH*n* Reserved. (RO)

AHB_DMA_IN_REMAIN_UNDER_4B_CH*n* Reserved. (RO)

AHB_DMA_IN_BUF_HUNGRY_CH*n* Reserved. (RO)

Register 6.47. AHB_DMA_IN_STATE_CH*n*_REG (*n*: 0-2) (0x0084+0xC0*n)

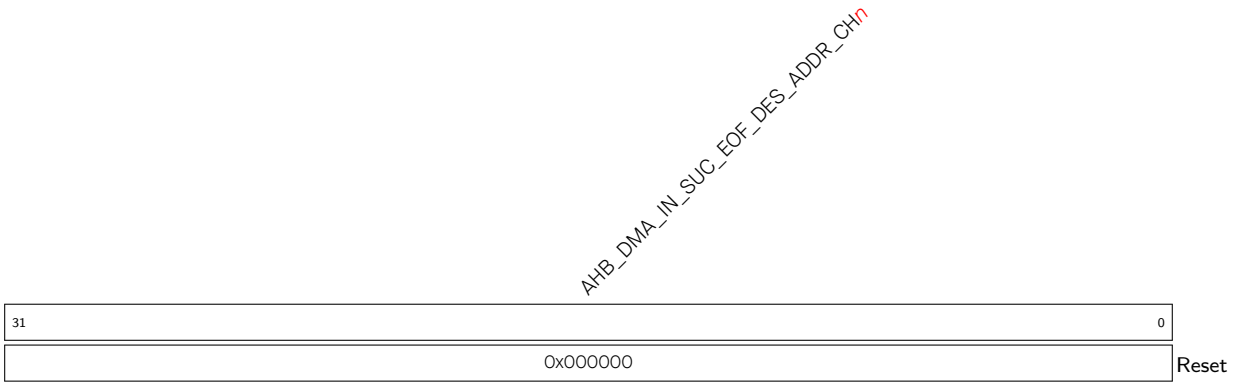


AHB_DMA_INLINK_DSCR_ADDR_CH*n* Represents the lower 18 bits of the next receive descriptor address that is pre-read (but not processed yet). If the current receive descriptor is the last descriptor, then this field represents the address of the current receive descriptor. (RO)

AHB_DMA_IN_DSCR_STATE_CH*n* Reserved. (RO)

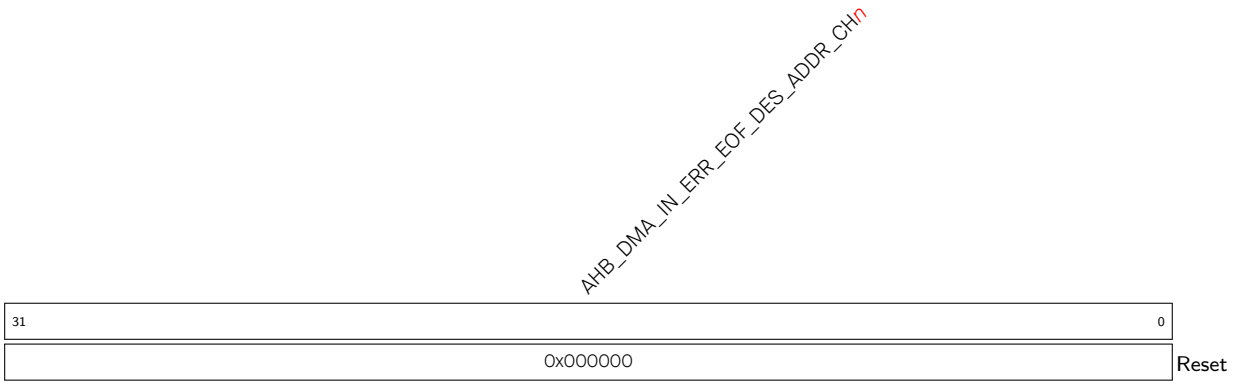
AHB_DMA_IN_STATE_CH*n* Reserved. (RO)

Register 6.48. AHB_DMA_IN_SUC_EOF_DES_ADDR_CH*n*_REG (*n*: 0-2) (0x0088+0xC0*n)



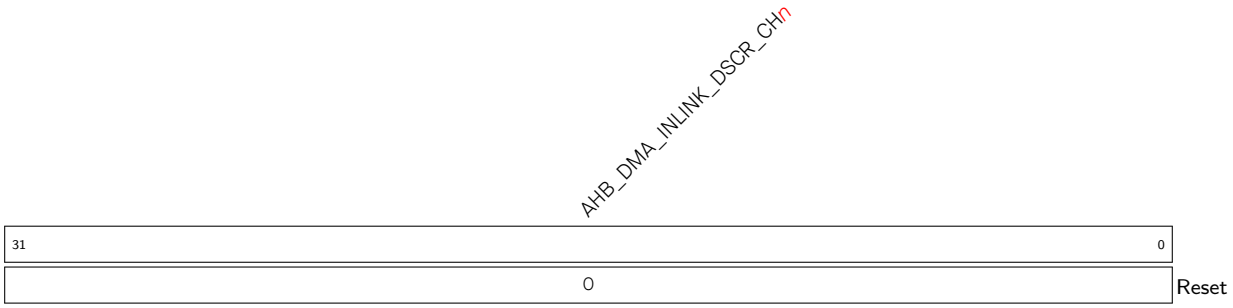
AHB_DMA_IN_SUC_EOF_DES_ADDR_CH*n* Represents the address of the receive descriptor when the EOF bit in this descriptor is 1. (RO)

Register 6.49. AHB_DMA_IN_ERR_EOF_DES_ADDR_CH*n*_REG (*n*: 0-2) (0x008C+0xC0*n*)



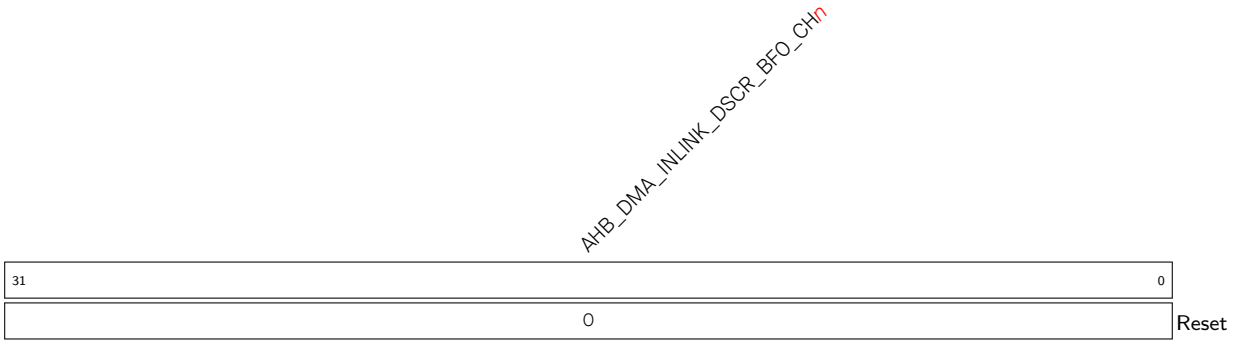
AHB_DMA_IN_ERR_EOF_DES_ADDR_CH*n* Represents the address of the receive descriptor when there are some errors in the currently received data. Valid only for UHCI. (RO)

Register 6.50. AHB_DMA_IN_DSCR_CH*n*_REG (*n*: 0-2) (0x0090+0xC0*n*)



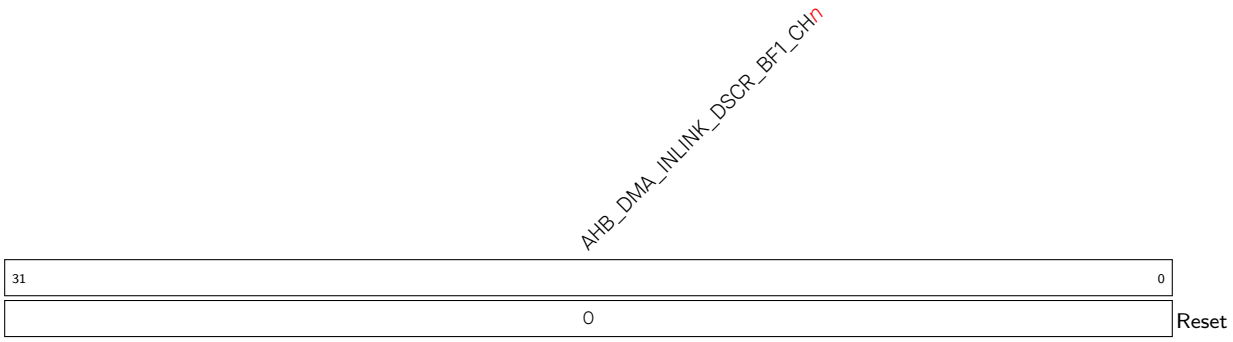
AHB_DMA_INLINK_DSCR_CH*n* Represents the address of the next receive descriptor x+1 pointed by the current receive descriptor that is pre-read. (RO)

Register 6.51. AHB_DMA_IN_DSCR_BF0_CH*n*_REG (*n*: 0-2) (0x0094+0xC0*n*)



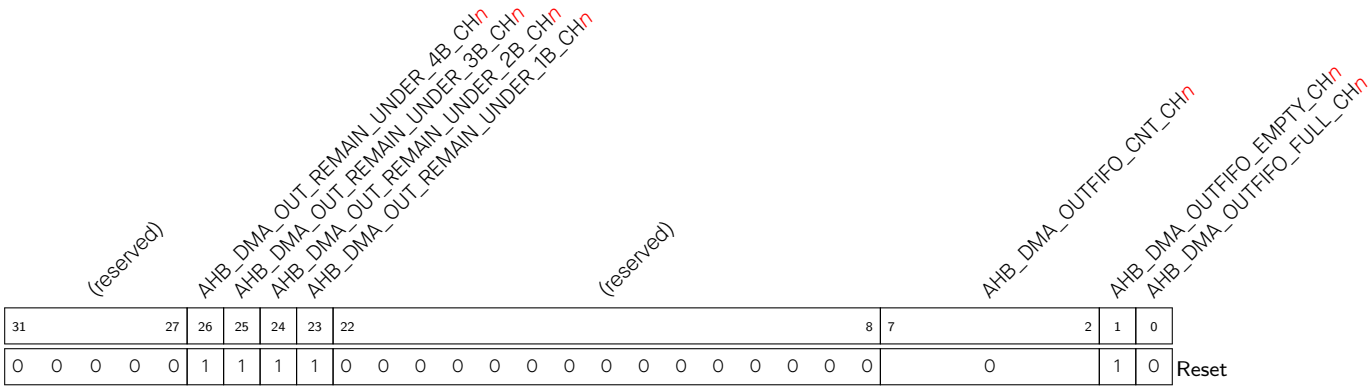
AHB_DMA_INLINK_DSCR_BF0_CH*n* Represents the address of the current receive descriptor x that is pre-read. (RO)

Register 6.52. AHB_DMA_IN_DSCR_BF1_CH*n*_REG (*n*: 0-2) (0x0098+0xC0*n*)



AHB_DMA_INLINK_DSCR_BF1_CH*n* Represents the address of the previous receive descriptor x-1 that is pre-read. (RO)

Register 6.53. AHB_DMA_OUTFIFO_STATUS_CH*n*_REG (*n*: 0-2) (0x00D8+0xC0*n*)



AHB_DMA_OUTFIFO_FULL_CH*n* Represents whether L1 TX FIFO is full.

0: Not Full

1: Full

(RO)

AHB_DMA_OUTFIFO_EMPTY_CH*n* Represents whether L1 TX FIFO is empty.

0: Not empty

1: Empty

(RO)

AHB_DMA_OUTFIFO_CNT_CH*n* Represents the number of data bytes in L1 TX FIFO for TX channel *n*. (RO)

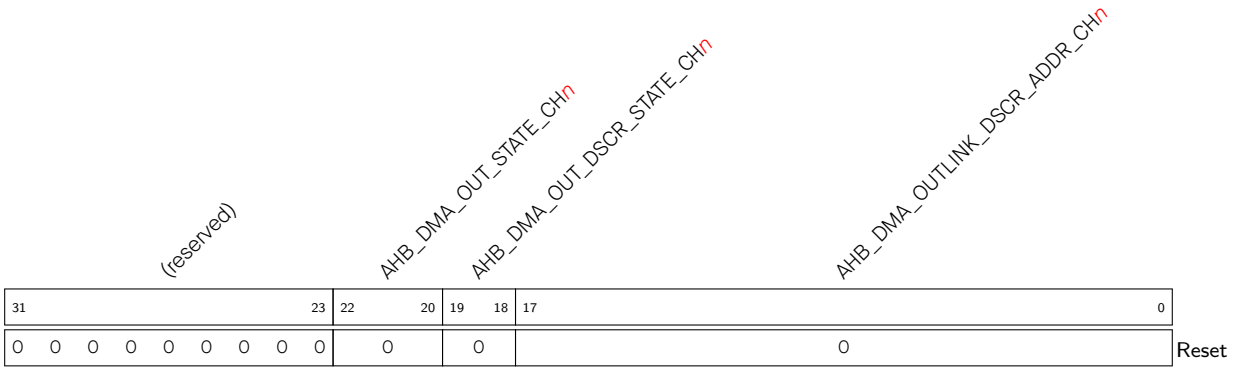
AHB_DMA_OUT_REMAIN_UNDER_1B_CH*n* Reserved. (RO)

AHB_DMA_OUT_REMAIN_UNDER_2B_CH*n* Reserved. (RO)

AHB_DMA_OUT_REMAIN_UNDER_3B_CH*n* Reserved. (RO)

AHB_DMA_OUT_REMAIN_UNDER_4B_CH*n* Reserved. (RO)

Register 6.54. AHB_DMA_OUT_STATE_CH*n*_REG (*n*: 0-2) (0x00E4+0xC0*n*)

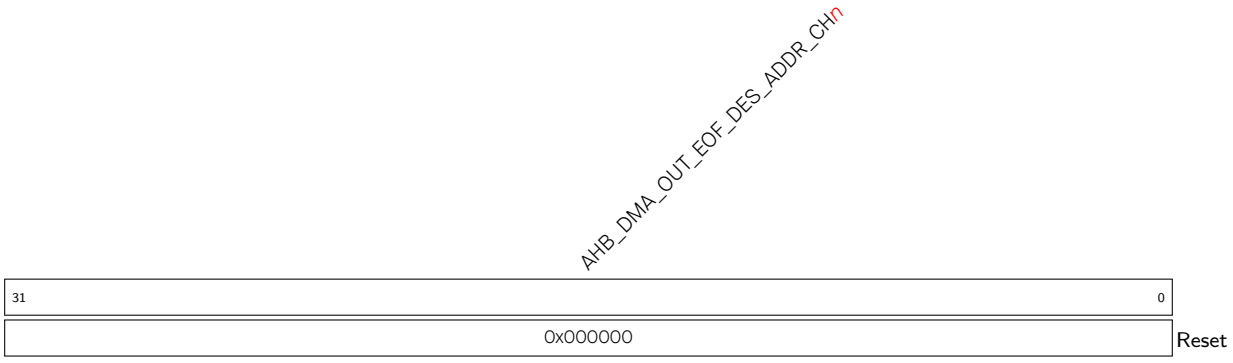


AHB_DMA_OUTLINK_DSCR_ADDR_CH*n* Represents the lower 18 bits of the next transmit descriptor address that is pre-read (but not processed yet). If the current transmit descriptor is the last descriptor, then this field represents the address of the current transmit descriptor. (RO)

AHB_DMA_OUT_DSCR_STATE_CH*n* Reserved. (RO)

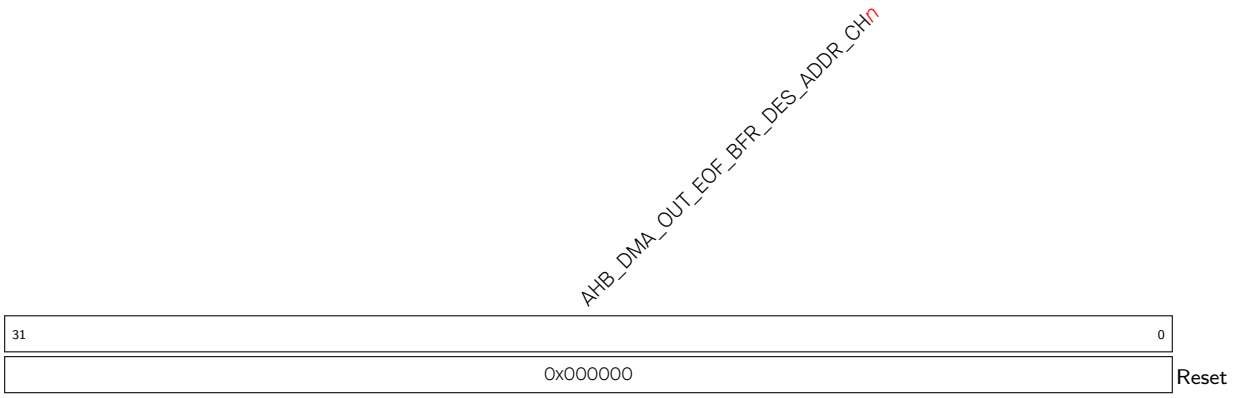
AHB_DMA_OUT_STATE_CH*n* Reserved. (RO)

Register 6.55. AHB_DMA_OUT_EOF_DES_ADDR_CH*n*_REG (*n*: 0-2) (0x00E8+0xC0*n*)



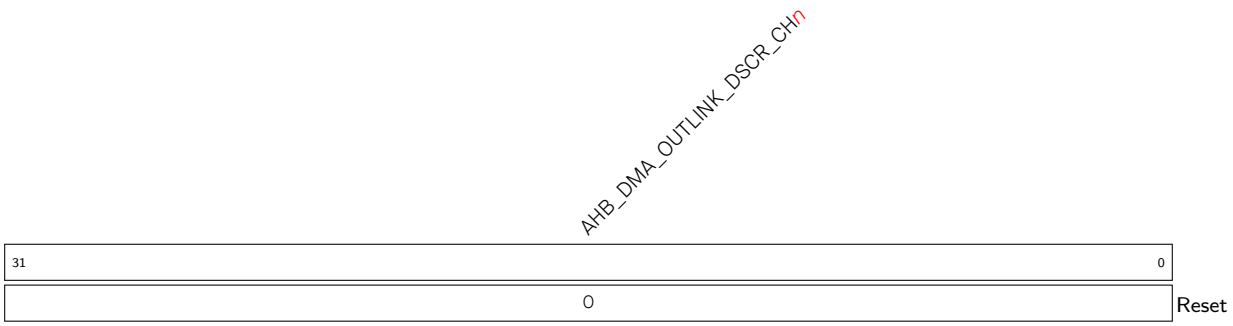
AHB_DMA_OUT_EOF_DES_ADDR_CH*n* Represents the address of the transmit descriptor when the EOF bit in this descriptor is 1. (RO)

Register 6.56. AHB_DMA_OUT_EOF_BFR_DES_ADDR_CH*n*_REG (*n*: 0-2) (0x00EC+0xC0*n*)



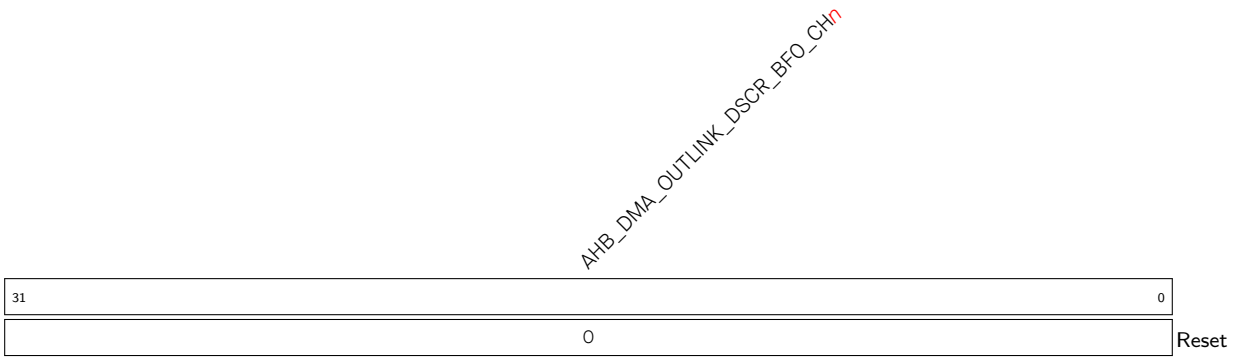
AHB_DMA_OUT_EOF_BFR_DES_ADDR_CH*n* Represents the address of the transmit descriptor before the last transmit descriptor. (RO)

Register 6.57. AHB_DMA_OUT_DSCR_CH*n*_REG (*n*: 0-2) (0x00F0+0xC0*n*)



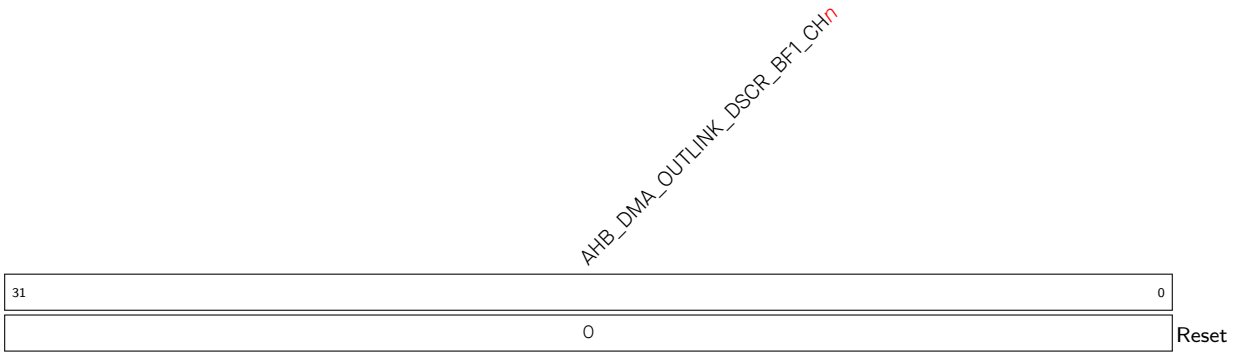
AHB_DMA_OUTLINK_DSCR_CH*n* Represents the address of the next transmit descriptor y+1 pointed by the current transmit descriptor that is pre-read. (RO)

Register 6.58. AHB_DMA_OUT_DSCR_BF0_CH*n*_REG (*n*: 0-2) (0x00F4+0xC0*n*)



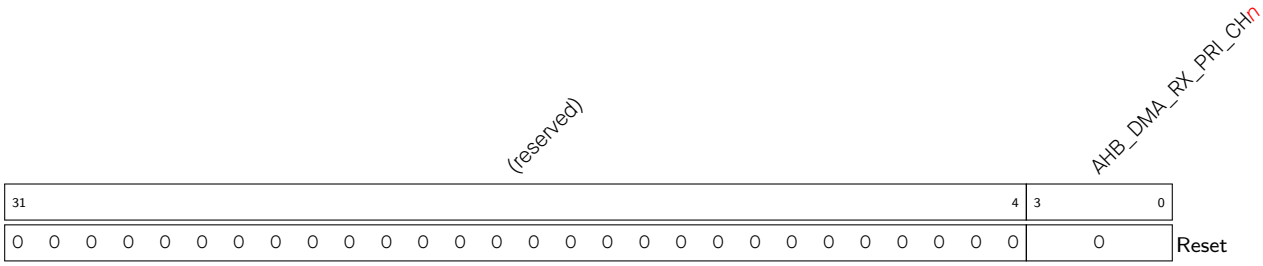
AHB_DMA_OUTLINK_DSCR_BF0_CH*n* Represents the address of the current transmit descriptor y that is pre-read. (RO)

Register 6.59. AHB_DMA_OUT_DSCR_BF1_CH*n*_REG (*n*: 0-2) (0x00F8+0xC0*n*)



AHB_DMA_OUTLINK_DSCR_BF1_CH*n* Represents the address of the previous transmit descriptor y-1 that is pre-read. (RO)

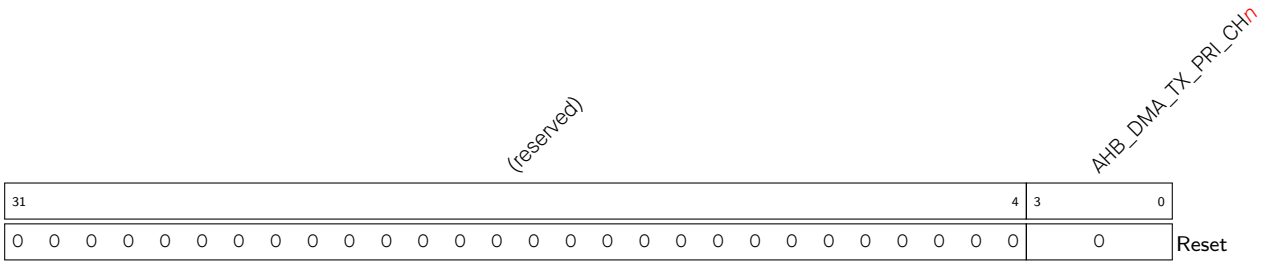
Register 6.60. AHB_DMA_IN_PRI_CH*n*_REG (*n*: 0-2) (0x009C+0xC0*n*)



AHB_DMA_RX_PRI_CH*n* Configures the priority of RX channel *n*. The larger the value, the higher the priority.

Value range: 0 ~ 5 (R/W)

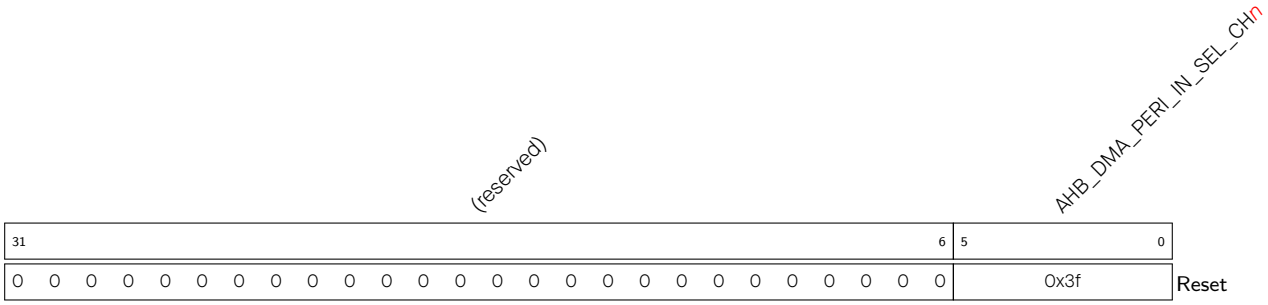
Register 6.61. AHB_DMA_OUT_PRI_CH*n*_REG (*n*: 0-2) (0x00FC+0xC0*n*)



AHB_DMA_TX_PRI_CH*n* Configures the priority of TX channel *n*. The larger the value, the higher the priority.

Value range: 0 ~ 5 (R/W)

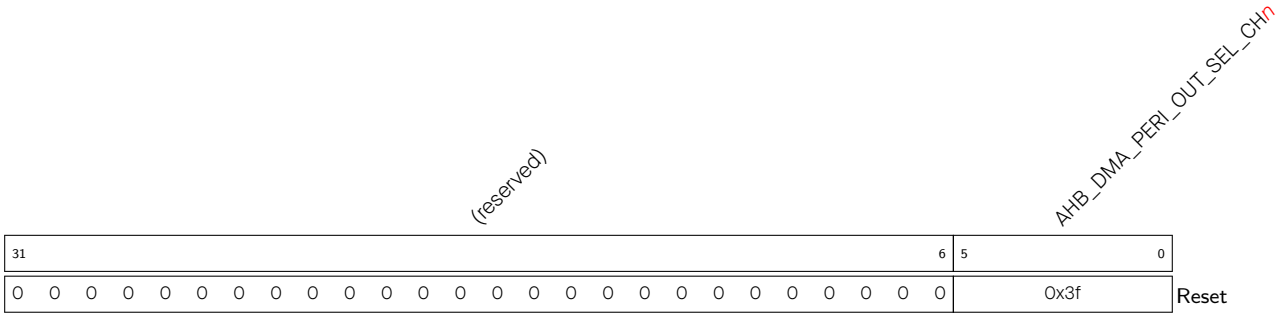
Register 6.62. AHB_DMA_IN_PERI_SEL_CH*n*_REG (*n*: 0-2) (0x00A0+0xC0n*)**



AHB_DMA_PERI_IN_SEL_CH*n* Configures the peripheral connected to RX channel *n*.

0: I3C
1: Dummy-1
2: UHCI
3: I2S0
4: I2S1
5: I2S2
6: Dummy-6
7: Dummy-7
8: ADC
9: Dummy-9
10: RMT
11 ~ 15: Dummy-11 ~ Dummy-15
16 ~ 63: Invalid
(R/W)

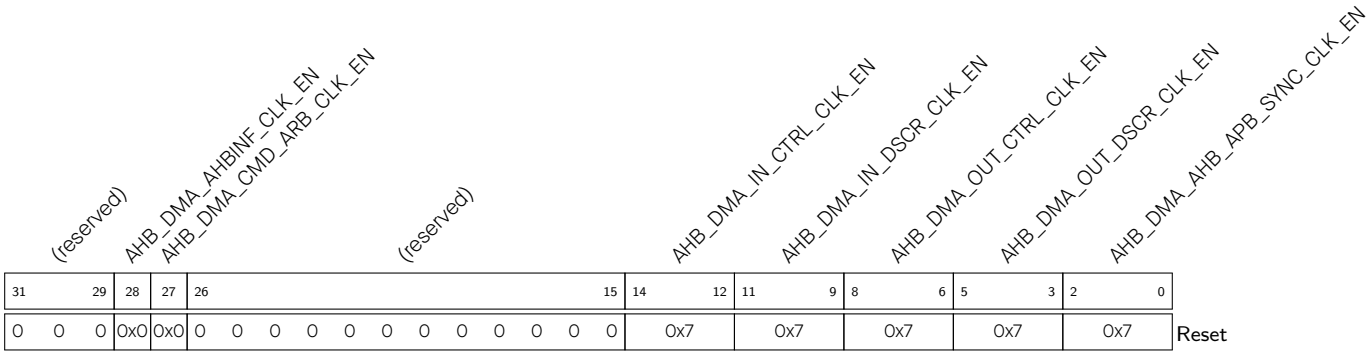
Register 6.63. AHB_DMA_OUT_PERI_SEL_CH*n*_REG (*n*: 0-2) (0x0100+0xC0*n*)



AHB_DMA_PERI_OUT_SEL_CH*n* Configures the peripheral connected to TX channel *n*.

0: I3C
1: Dummy-1
2: UHCI
3: I2S0
4: I2S1
5: I2S2
6: Dummy-6
7: Dummy-7
8: ADC
9: Dummy-9
10: RMT
11 ~ 15: Dummy-11 ~ Dummy-15
16 ~ 63: Invalid
(R/W)

Register 6.64. AHB_DMA_MODULE_CLK_EN_REG (0x0404)



AHB_DMA_AHB_APB_SYNC_CLK_EN Configures whether to forcibly enable the clock for the AHB DMA register configuration synchronization module.

Bit 0 corresponds to TX and RX channel 0, bit 1 corresponds to channel TX and RX 1, and so on.

0: Not forcibly enable

1: Forcibly enable

(R/W)

AHB_DMA_OUT_DSCR_CLK_EN Configures whether to forcibly enable the clock for the AHB DMA TX descriptor processing module.

Bit 0 corresponds to TX channel 0, bit 1 corresponds to TX channel 1, and so on.

0: Not forcibly enable

1: Forcibly enable

(R/W)

AHB_DMA_OUT_CTRL_CLK_EN Configures whether to forcibly enable the clock for the AHB DMA TX data processing module.

Bit 0 corresponds to TX channel 0, bit 1 corresponds to TX channel 1, and so on.

0: Not forcibly enable

1: Forcibly enable

(R/W)

AHB_DMA_IN_DSCR_CLK_EN Configures whether to forcibly enable the clock for the AHB DMA RX descriptor processing module.

Bit 0 corresponds to RX channel 0, bit 1 corresponds to RX channel 1, and so on.

0: Not forcibly enable

1: Forcibly enable

(R/W)

AHB_DMA_IN_CTRL_CLK_EN Configures whether to forcibly enable the clock for the AHB DMA RX data processing module.

Bit 0 corresponds to RX channel 0, bit 1 corresponds to RX channel 1, and so on.

0: Not forcibly enable

1: Forcibly enable

(R/W)

Continued on the next page...

Register 6.64. AHB_DMA_MODULE_CLK_EN_REG (0x0404)

Continued from the previous page...

AHB_DMA_CMD_ARB_CLK_EN Configures whether to forcibly enable the clock for the AHB DMA arbitration module.
0: Not forcibly enable
1: Forcibly enable
(R/W)

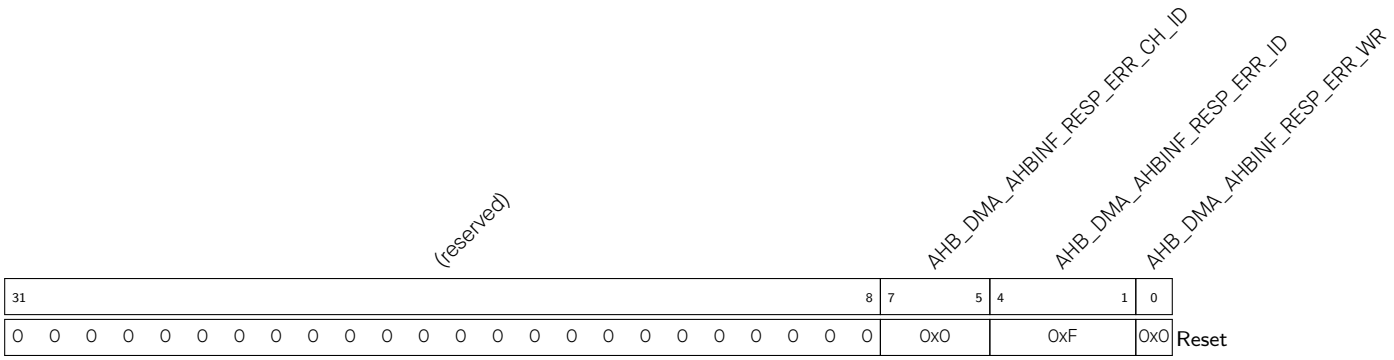
AHB_DMA_AHBINF_CLK_EN Configures whether to forcibly enable the clock for the AHB DMA bus interface processing module.
0: Not forcibly enable
1: Forcibly enable
(R/W)

Register 6.65. AHB_DMA_AHBINF_RESP_ERR_STATUS0_REG (0x0408)



AHB_DMA_AHBINF_RESP_ERR_ADDR Represents the address of the current AHB bus transfer that triggers an error response. (RO)

Register 6.66. AHB_DMA_AHBINF_RESP_ERR_STATUS1_REG (0x040C)



AHB_DMA_AHBINF_RESP_ERR_CH_ID Represents the channel ID of the transfer that triggers the AHB bus error response.

Bits[0:1]: Channel number. Value range: 0 ~2

Bit[2]: Channel direction. 0: RX

1: TX

(RO)

AHB_DMA_AHBINF_RESP_ERR_ID Represents the ID of the transfer that triggers the AHB bus error response.

0: Transfer between I3C and memory

2: Transfer between UHCI and memory

3: Transfer between I2S0 and memory

4: Transfer between I2S1 and memory

5: Transfer between I2S2 and memory

6: Transfer between ADC and memory

10: Transfer between RMT and memory

11 + 15: Dummy-11 + Dummy-15

16 + 63: Invalid

Other values are invalid

(RO)

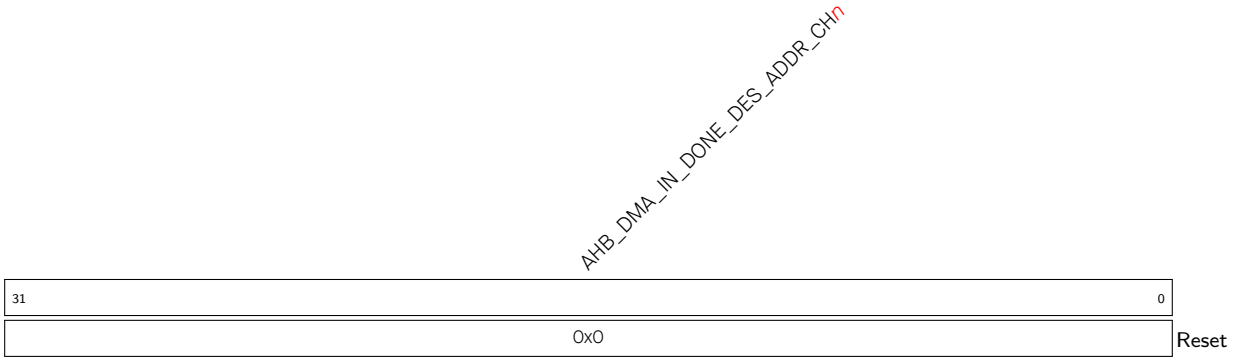
AHB_DMA_AHBINF_RESP_ERR_WR Represents the type of the transfer that triggered the AHB bus error response received by the AHB DMA.

0: Read

1: Write

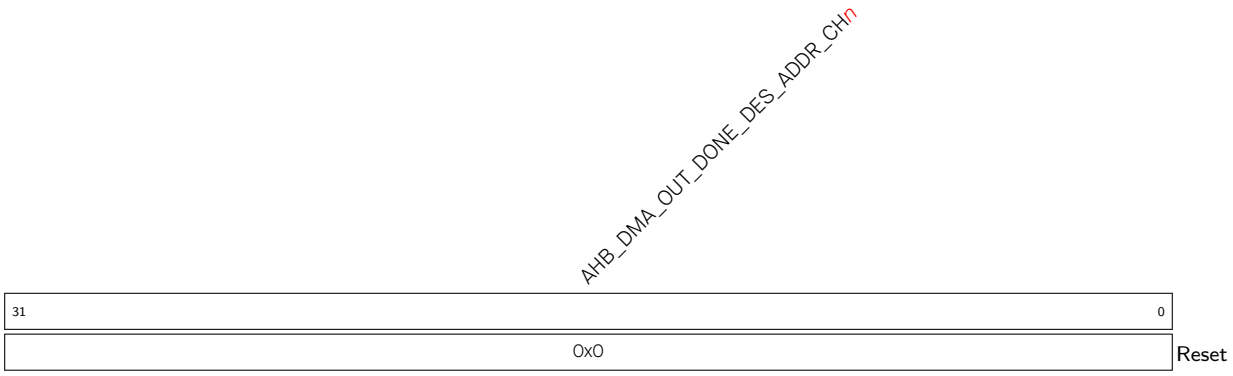
(RO)

Register 6.67. AHB_DMA_IN_DONE_DES_ADDR_CH*n*_REG(*n*: 0-2) (0x0410+0x08n*)**



AHB_DMA_IN_DONE_DES_ADDR_CH*n* Represents the address of the descriptor corresponding to the completed RX transfer. (RO)

Register 6.68. AHB_DMA_OUT_DONE_DES_ADDR_CH*n*_REG(*n*: 0-2) (0x0414+0x08n*)**



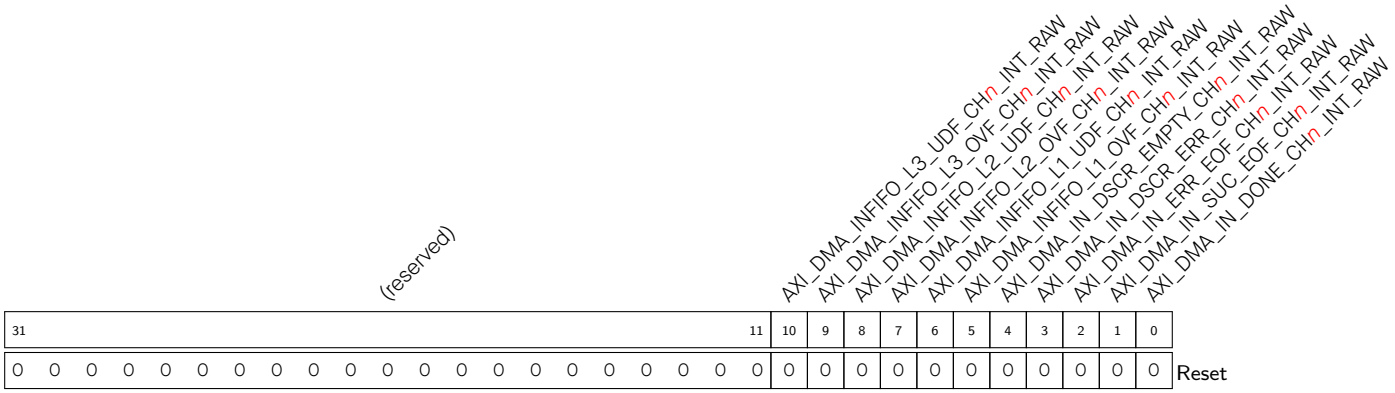
AHB_DMA_OUT_DONE_DES_ADDR_CH*n* Represents the address of the descriptor corresponding to the completed TX transfer. (RO)

6.9.2 GDMA-AXI Registers

The addresses in this section are relative to GDMA-AXI base address provided in Table 9.3-2 in Chapter 9 *System and Memory*.

For how to program reserved fields, please refer to Section Programming Reserved Register Field.

Register 6.69. AXI_DMA_IN_INT_RAW_CH*n*_REG (*n*: 0-2) (0x0000+0x68**n*)



AXI_DMA_IN_DONE_CH*n*_INT_RAW The raw interrupt status of AXI_DMA_IN_DONE_CH*n*_INT. (R/WTC/SS)

AXI_DMA_IN_SUC_EOF_CH*n*_INT_RAW The raw interrupt status of AXI_DMA_IN_SUC_EOF_CH*n*_INT. For UHCI this bit turns to 1 when the last data byte pointed by one receive descriptor has been received and no data error is detected for RX channel 0. (R/WTC/SS)

AXI_DMA_IN_ERR_EOF_CH*n*_INT_RAW The raw interrupt status of AXI_DMA_IN_ERR_EOF_CH*n*_INT. Valid only for UHCI. (R/WTC/SS)

AXI_DMA_IN_DSCR_ERR_CH*n*_INT_RAW The raw interrupt status of AXI_DMA_IN_DSCR_ERR_CH*n*_INT. (R/WTC/SS)

AXI_DMA_IN_DSCR_EMPTY_CH*n*_INT_RAW The raw interrupt status of AXI_DMA_IN_DSCR_EMPTY_CH*n*_INT. (R/WTC/SS)

AXI_DMA_INFIFO_L1_OVF_CH*n*_INT_RAW The raw interrupt status of AXI_DMA_INFIFO_L1_OVF_CH*n*_INT. (R/WTC/SS)

AXI_DMA_INFIFO_L1_UDF_CH*n*_INT_RAW The raw interrupt status of AXI_DMA_INFIFO_L1_UDF_CH*n*_INT. (R/WTC/SS)

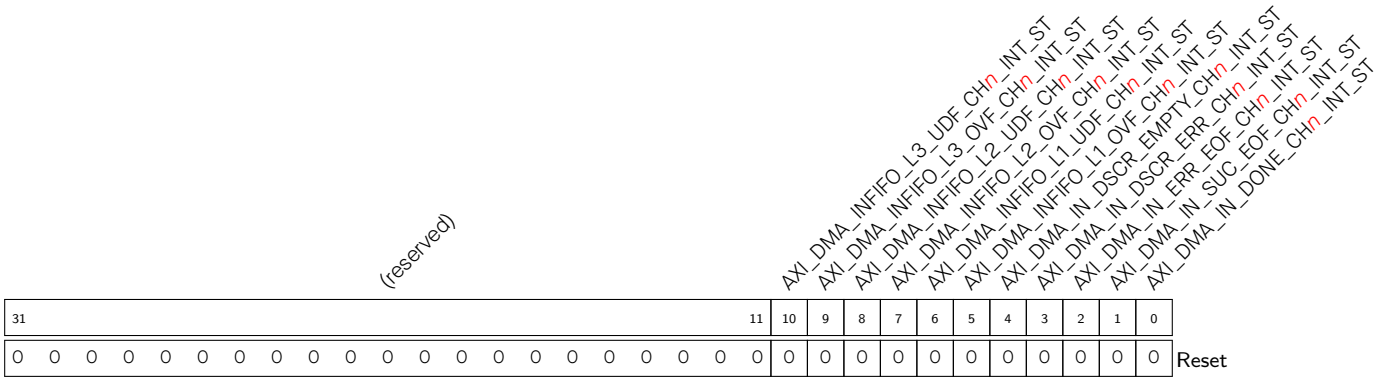
AXI_DMA_INFIFO_L2_OVF_CH*n*_INT_RAW The raw interrupt status of AXI_DMA_INFIFO_L2_OVF_CH*n*_INT. (R/WTC/SS)

AXI_DMA_INFIFO_L2_UDF_CH*n*_INT_RAW The raw interrupt status of AXI_DMA_INFIFO_L2_UDF_CH*n*_INT. (R/WTC/SS)

AXI_DMA_INFIFO_L3_OVF_CH*n*_INT_RAW The raw interrupt status of AXI_DMA_INFIFO_L3_OVF_CH*n*_INT. (R/WTC/SS)

AXI_DMA_INFIFO_L3_UDF_CH*n*_INT_RAW The raw interrupt status of AXI_DMA_INFIFO_L3_UDF_CH*n*_INT. (R/WTC/SS)

Register 6.70. AXI_DMA_IN_INT_ST_CH*n*_REG (*n*: 0-2) (0x0004+0x68**n*)



AXI_DMA_IN_DONE_CH*n*_INT_ST The masked interrupt status of AXI_DMA_IN_DONE_CH*n*_INT. (RO)

AXI_DMA_IN_SUC_EOF_CH*n*_INT_ST The masked interrupt status of AXI_DMA_IN_SUC_EOF_CH*n*_INT. (RO)

AXI_DMA_IN_ERR_EOF_CH*n*_INT_ST The masked interrupt status of AXI_DMA_IN_ERR_EOF_CH*n*_INT. (RO)

AXI_DMA_IN_DSCR_ERR_CH*n*_INT_ST The masked interrupt status of AXI_DMA_IN_DSCR_ERR_CH*n*_INT. (RO)

AXI_DMA_IN_DSCR_EMPTY_CH*n*_INT_ST The masked interrupt status of AXI_DMA_IN_DSCR_EMPTY_CH*n*_INT. (RO)

AXI_DMA_INFIFO_L2_OVF_CH*n*_INT_ST The masked interrupt status of AXI_DMA_INFIFO_L2_OVF_CH*n*_INT. (RO)

AXI_DMA_INFIFO_L2_UDF_CH*n*_INT_ST The masked interrupt status of AXI_DMA_INFIFO_L2_UDF_CH*n*_INT. (RO)

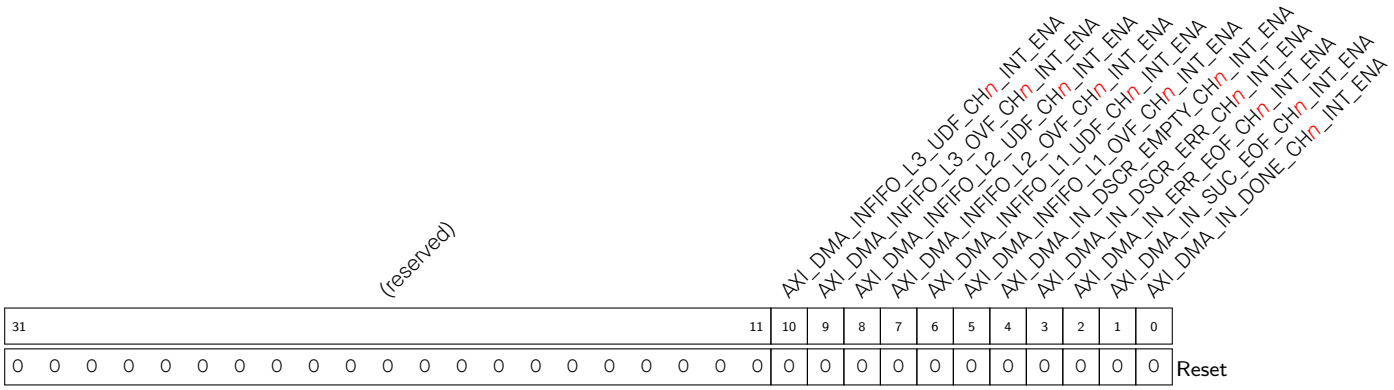
AXI_DMA_INFIFO_L1_OVF_CH*n*_INT_ST The masked interrupt status of AXI_DMA_INFIFO_L1_OVF_CH*n*_INT. (RO)

AXI_DMA_INFIFO_L1_UDF_CH*n*_INT_ST The masked interrupt status of AXI_DMA_INFIFO_L1_UDF_CH*n*_INT. (RO)

AXI_DMA_INFIFO_L3_OVF_CH*n*_INT_ST The masked interrupt status of AXI_DMA_INFIFO_L3_OVF_CH*n*_INT. (RO)

AXI_DMA_INFIFO_L3_UDF_CH*n*_INT_ST The masked interrupt status of AXI_DMA_INFIFO_L3_UDF_CH*n*_INT. (RO)

Register 6.71. AXI_DMA_IN_INT_ENA_CH*n*_REG (*n*: 0-2) (0x0008+0x68**n*)



AXI_DMA_IN_DONE_CH*n*_INT_ENA Write 1 to enable AXI_DMA_IN_DONE_CH*n*_INT. (R/W)

AXI_DMA_IN_SUC_EOF_CH*n*_INT_ENA Write 1 to enable AXI_DMA_IN_SUC_EOF_CH*n*_INT. (R/W)

AXI_DMA_IN_ERR_EOF_CH*n*_INT_ENA Write 1 to enable AXI_DMA_IN_ERR_EOF_CH*n*_INT. (R/W)

AXI_DMA_IN_DSCR_ERR_CH*n*_INT_ENA Write 1 to enable AXI_DMA_IN_DSCR_ERR_CH*n*_INT. (R/W)

AXI_DMA_IN_DSCR_EMPTY_CH*n*_INT_ENA Write 1 to enable AXI_DMA_IN_DSCR_EMPTY_CH*n*_INT. (R/W)

AXI_DMA_INFIFO_L1_OVF_CH*n*_INT_ENA Write 1 to enable AXI_DMA_INFIFO_L1_OVF_CH*n*_INT. (R/W)

AXI_DMA_INFIFO_L1_UDF_CH*n*_INT_ENA Write 1 to enable AXI_DMA_INFIFO_L1_UDF_CH*n*_INT. (R/W)

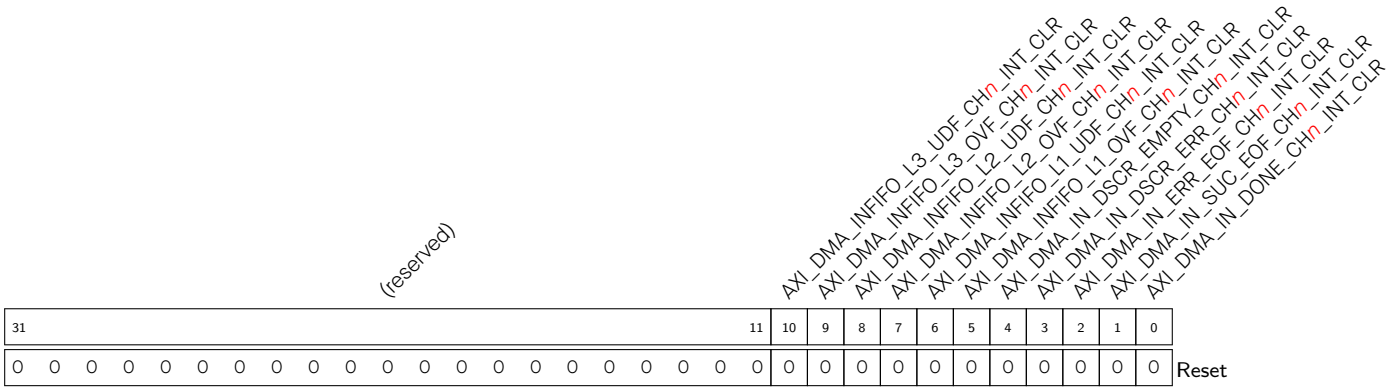
AXI_DMA_INFIFO_L2_OVF_CH*n*_INT_ENA Write 1 to enable AXI_DMA_INFIFO_L2_OVF_CH*n*_INT. (R/W)

AXI_DMA_INFIFO_L2_UDF_CH*n*_INT_ENA Write 1 to enable AXI_DMA_INFIFO_L2_UDF_CH*n*_INT. (R/W)

AXI_DMA_INFIFO_L3_OVF_CH*n*_INT_ENA Write 1 to enable AXI_DMA_INFIFO_L3_OVF_CH*n*_INT. (R/W)

AXI_DMA_INFIFO_L3_UDF_CH*n*_INT_ENA Write 1 to enable AXI_DMA_INFIFO_L3_UDF_CH*n*_INT. (R/W)

Register 6.72. AXI_DMA_IN_INT_CLR_CH*n*_REG (*n*: 0-2) (0x000C+0x68**n*)



AXI_DMA_IN_DONE_CH*n*_INT_CLR Write 1 to clear AXI_DMA_IN_DONE_CH*n*_INT. (WT)

AXI_DMA_IN_SUC_EOF_CH*n*_INT_CLR Write 1 to clear AXI_DMA_IN_SUC_EOF_CH*n*_INT. (WT)

AXI_DMA_IN_ERR_EOF_CH*n*_INT_CLR Write 1 to clear AXI_DMA_IN_ERR_EOF_CH*n*_INT. (WT)

AXI_DMA_IN_DSCR_ERR_CH*n*_INT_CLR Write 1 to clear AXI_DMA_IN_DSCR_ERR_CH*n*_INT. (WT)

AXI_DMA_IN_DSCR_EMPTY_CH*n*_INT_CLR Write 1 to clear AXI_DMA_IN_DSCR_EMPTY_CH*n*_INT. (WT)

AXI_DMA_INFIFO_L1_OVF_CH*n*_INT_CLR Write 1 to clear AXI_DMA_INFIFO_L1_OVF_CH*n*_INT. (WT)

AXI_DMA_INFIFO_L1_UDF_CH*n*_INT_CLR Write 1 to clear AXI_DMA_INFIFO_L1_UDF_CH*n*_INT. (WT)

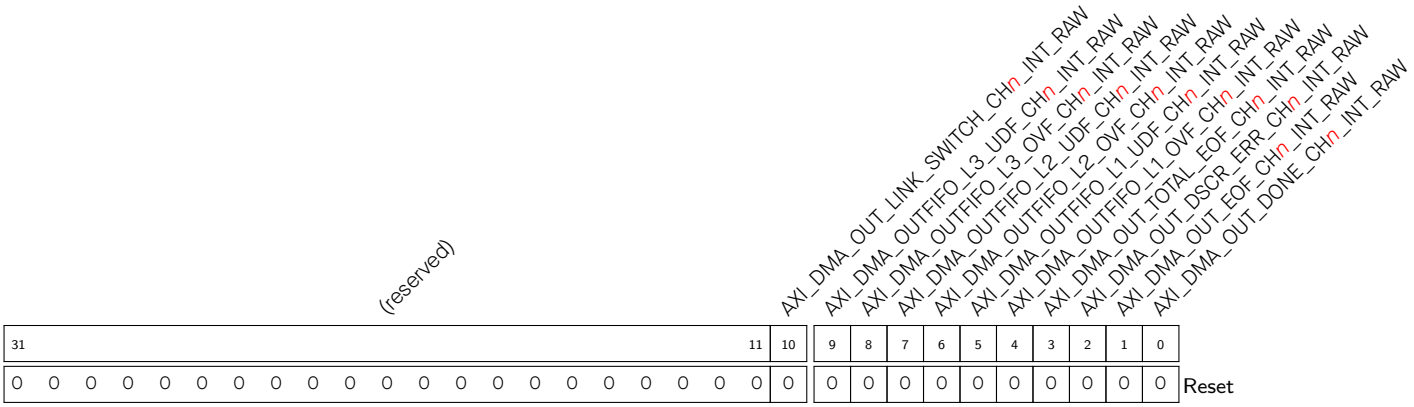
AXI_DMA_INFIFO_L2_OVF_CH*n*_INT_CLR Write 1 to clear AXI_DMA_INFIFO_L2_OVF_CH*n*_INT. (WT)

AXI_DMA_INFIFO_L2_UDF_CH*n*_INT_CLR Write 1 to clear AXI_DMA_INFIFO_L2_UDF_CH*n*_INT. (WT)

AXI_DMA_INFIFO_L3_OVF_CH*n*_INT_CLR Write 1 to clear AXI_DMA_INFIFO_L3_OVF_CH*n*_INT. (WT)

AXI_DMA_INFIFO_L3_UDF_CH*n*_INT_CLR Write 1 to clear AXI_DMA_INFIFO_L3_UDF_CH*n*_INT. (WT)

Register 6.73. AXI_DMA_OUT_INT_RAW_CH*n*_REG (*n*: 0-2) (0x0138+0x68n*)**



AXI_DMA_OUT_DONE_CH*n*_INT_RAW The raw interrupt status of AXI_DMA_OUT_DONE_CH*n*_INT. (R/WTC/SS)

AXI_DMA_OUT_EOF_CH*n*_INT_RAW The raw interrupt status of AXI_DMA_OUT_EOF_CH*n*_INT. (R/WTC/SS)

AXI_DMA_OUT_DSCR_ERR_CH*n*_INT_RAW The raw interrupt status of AXI_DMA_OUT_DSCR_ERR_CH*n*_INT. (R/WTC/SS)

AXI_DMA_OUT_TOTAL_EOF_CH*n*_INT_RAW The raw interrupt status of AXI_DMA_OUT_TOTAL_EOF_CH*n*_INT. (R/WTC/SS)

AXI_DMA_OUTFIFO_L1_OVF_CH*n*_INT_RAW The raw interrupt status of AXI_DMA_OUTFIFO_L1_OVF_CH*n*_INT. (R/WTC/SS)

AXI_DMA_OUTFIFO_L1_UDF_CH*n*_INT_RAW The raw interrupt status of AXI_DMA_OUTFIFO_L1_UDF_CH*n*_INT. (R/WTC/SS)

AXI_DMA_OUTFIFO_L2_OVF_CH*n*_INT_RAW The raw interrupt status of AXI_DMA_OUTFIFO_L2_OVF_CH*n*_INT. (R/WTC/SS)

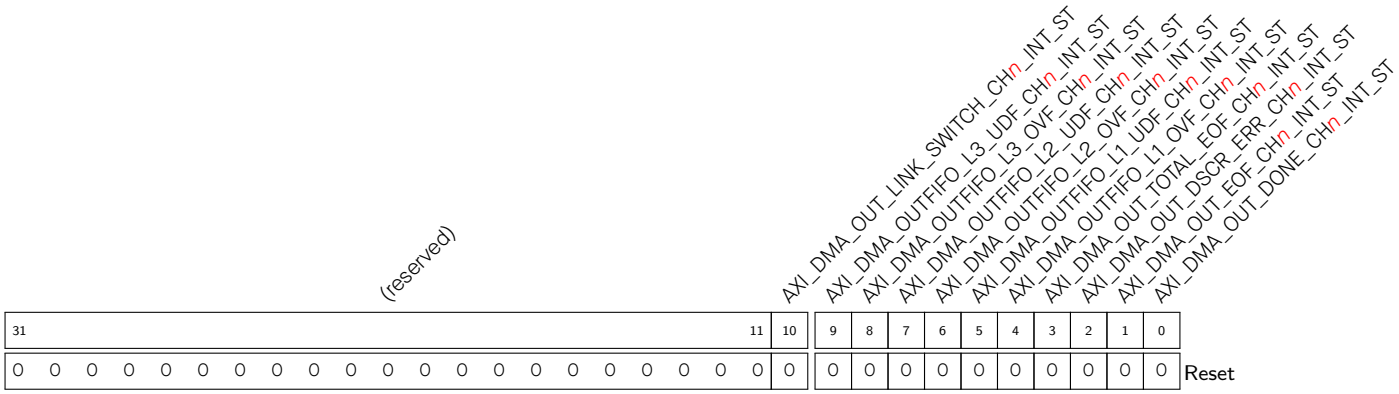
AXI_DMA_OUTFIFO_L2_UDF_CH*n*_INT_RAW The raw interrupt status of AXI_DMA_OUTFIFO_L2_UDF_CH*n*_INT. (R/WTC/SS)

AXI_DMA_OUTFIFO_L3_OVF_CH*n*_INT_RAW The raw interrupt status of AXI_DMA_OUTFIFO_L3_OVF_CH*n*_INT. (R/WTC/SS)

AXI_DMA_OUTFIFO_L3_UDF_CH*n*_INT_RAW The raw interrupt status of AXI_DMA_OUTFIFO_L3_UDF_CH*n*_INT. (R/WTC/SS)

AXI_DMA_OUT_LINK_SWITCH_CH*n*_INT_RAW The raw interrupt status of AXI_DMA_OUT_LINK_SWITCH_CH*n*_INT. (R/WTC/SS)

Register 6.74. AXI_DMA_OUT_INT_ST_CH*n*_REG (*n*: 0-2) (0x013C+0x68n*)**



AXI_DMA_OUT_DONE_CH*n*_INT_ST The masked interrupt status of AXI_DMA_OUT_DONE_CH*n*_INT. (RO)

AXI_DMA_OUT_EOF_CH*n*_INT_ST The masked interrupt status of AXI_DMA_OUT_EOF_CH*n*_INT. (RO)

AXI_DMA_OUT_DSCR_ERR_CH*n*_INT_ST The masked interrupt status of AXI_DMA_OUT_DSCR_ERR_CH*n*_INT. (RO)

AXI_DMA_OUT_TOTAL_EOF_CH*n*_INT_ST The masked interrupt status of AXI_DMA_OUT_TOTAL_EOF_CH*n*_INT. (RO)

AXI_DMA_OUTFIFO_L1_OVF_CH*n*_INT_ST The masked interrupt status of AXI_DMA_OUTFIFO_L1_OVF_CH*n*_INT. (RO)

AXI_DMA_OUTFIFO_L1_UDF_CH*n*_INT_ST The masked interrupt status of AXI_DMA_OUTFIFO_L1_UDF_CH*n*_INT. (RO)

AXI_DMA_OUTFIFO_L2_OVF_CH*n*_INT_ST The masked interrupt status of AXI_DMA_OUTFIFO_L2_OVF_CH*n*_INT. (RO)

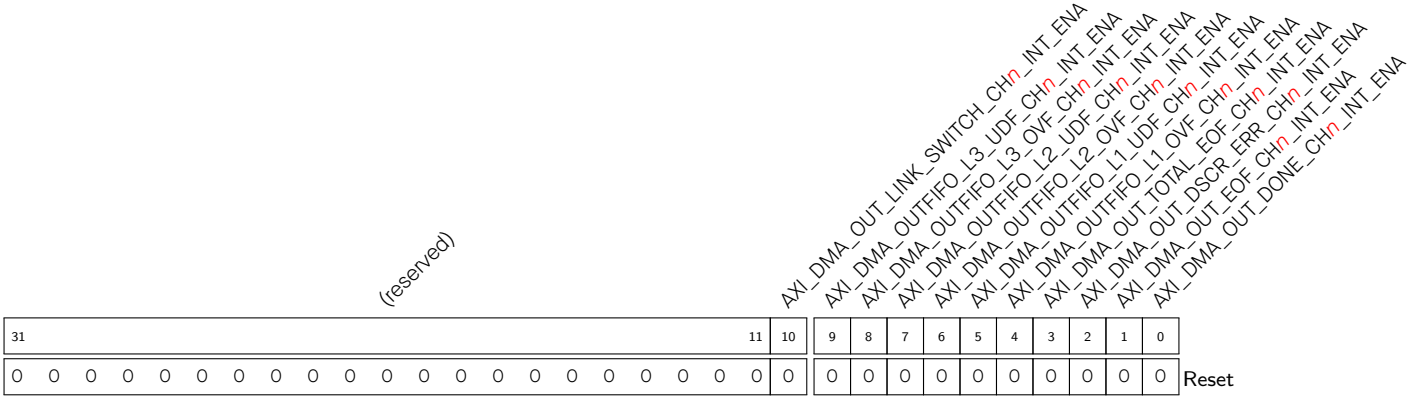
AXI_DMA_OUTFIFO_L2_UDF_CH*n*_INT_ST The masked interrupt status of AXI_DMA_OUTFIFO_L2_UDF_CH*n*_INT. (RO)

AXI_DMA_OUTFIFO_L3_OVF_CH*n*_INT_ST The masked interrupt status of AXI_DMA_OUTFIFO_L3_OVF_CH*n*_INT. (RO)

AXI_DMA_OUTFIFO_L3_UDF_CH*n*_INT_ST The masked interrupt status of AXI_DMA_OUTFIFO_L3_UDF_CH*n*_INT. (RO)

AXI_DMA_OUT_LINK_SWITCH_CH*n*_INT_ST The masked interrupt status of AXI_DMA_OUT_LINK_SWITCH_CH*n*_INT. (RO)

Register 6.75. AXI_DMA_OUT_INT_ENA_CH*n*_REG (*n*: 0-2) (0x0140+0x68**n*)



AXI_DMA_OUT_DONE_CH*n*_INT_ENA Write 1 to enable AXI_DMA_OUT_DONE_CH*n*_INT. (R/W)

AXI_DMA_OUT_EOF_CH*n*_INT_ENA Write 1 to enable AXI_DMA_OUT_EOF_CH*n*_INT. (R/W)

AXI_DMA_OUT_DSCR_ERR_CH*n*_INT_ENA Write 1 to enable AXI_DMA_OUT_DSCR_ERR_CH*n*_INT. (R/W)

AXI_DMA_OUT_TOTAL_EOF_CH*n*_INT_ENA Write 1 to enable AXI_DMA_OUT_TOTAL_EOF_CH*n*_INT. (R/W)

AXI_DMA_OUTFIFO_L1_OVF_CH*n*_INT_ENA Write 1 to enable AXI_DMA_OUTFIFO_L1_OVF_CH*n*_INT. (R/W)

AXI_DMA_OUTFIFO_L1_UDF_CH*n*_INT_ENA Write 1 to enable AXI_DMA_OUTFIFO_L1_UDF_CH*n*_INT. (R/W)

AXI_DMA_OUTFIFO_L2_OVF_CH*n*_INT_ENA Write 1 to enable AXI_DMA_OUTFIFO_L2_OVF_CH*n*_INT. (R/W)

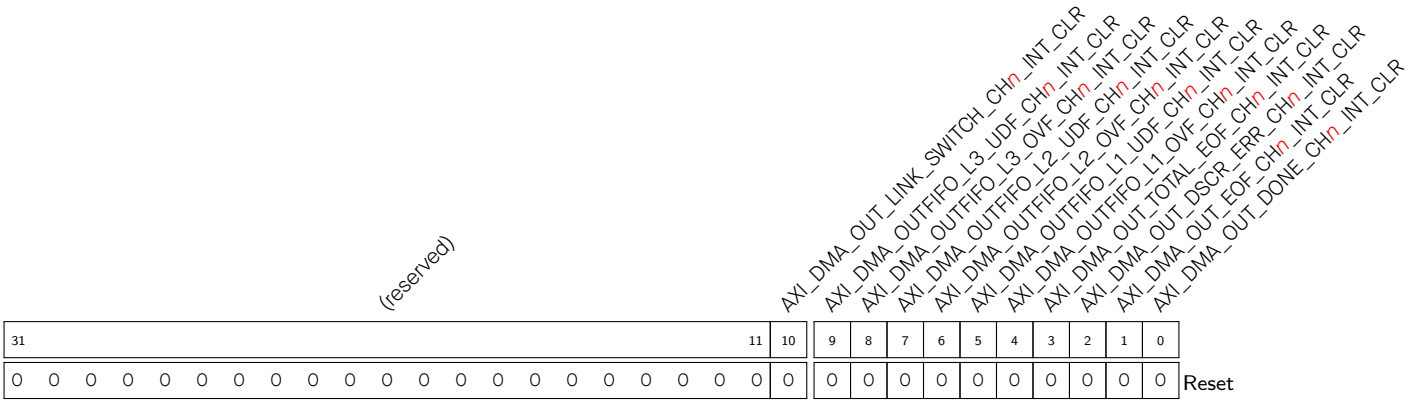
AXI_DMA_OUTFIFO_L2_UDF_CH*n*_INT_ENA Write 1 to enable AXI_DMA_OUTFIFO_L2_UDF_CH*n*_INT. (R/W)

AXI_DMA_OUTFIFO_L3_OVF_CH*n*_INT_ENA Write 1 to enable AXI_DMA_OUTFIFO_L3_OVF_CH*n*_INT. (R/W)

AXI_DMA_OUTFIFO_L3_UDF_CH*n*_INT_ENA Write 1 to enable AXI_DMA_OUTFIFO_L3_UDF_CH*n*_INT. (R/W)

AXI_DMA_OUT_LINK_SWITCH_CH*n*_INT_ENA Write 1 to enable AXI_DMA_OUT_LINK_SWITCH_CH*n*_INT. (RO)

Register 6.76. AXI_DMA_OUT_INT_CLR_CH*n*_REG (*n*: 0-2) (0x0144+0x68***n*)



AXI_DMA_OUT_DONE_CH*n*_INT_CLR Write 1 to clear AXI_DMA_OUT_DONE_CH*n*_INT. (WT)

AXI_DMA_OUT_EOF_CH*n*_INT_CLR Write 1 to clear AXI_DMA_OUT_EOF_CH*n*_INT. (WT)

AXI_DMA_OUT_DSCR_ERR_CH*n*_INT_CLR Write 1 to clear AXI_DMA_OUT_DSCR_ERR_CH*n*_INT. (WT)

AXI_DMA_OUT_TOTAL_EOF_CH*n*_INT_CLR Write 1 to clear AXI_DMA_OUT_TOTAL_EOF_CH*n*_INT. (WT)

AXI_DMA_OUTFIFO_L1_OVF_CH*n*_INT_CLR Write 1 to clear AXI_DMA_OUTFIFO_L1_OVF_CH*n*_INT. (WT)

AXI_DMA_OUTFIFO_L1_UDF_CH*n*_INT_CLR Write 1 to clear AXI_DMA_OUTFIFO_L1_UDF_CH*n*_INT. (WT)

AXI_DMA_OUTFIFO_L2_OVF_CH*n*_INT_CLR Write 1 to clear AXI_DMA_OUTFIFO_L2_OVF_CH*n*_INT. (WT)

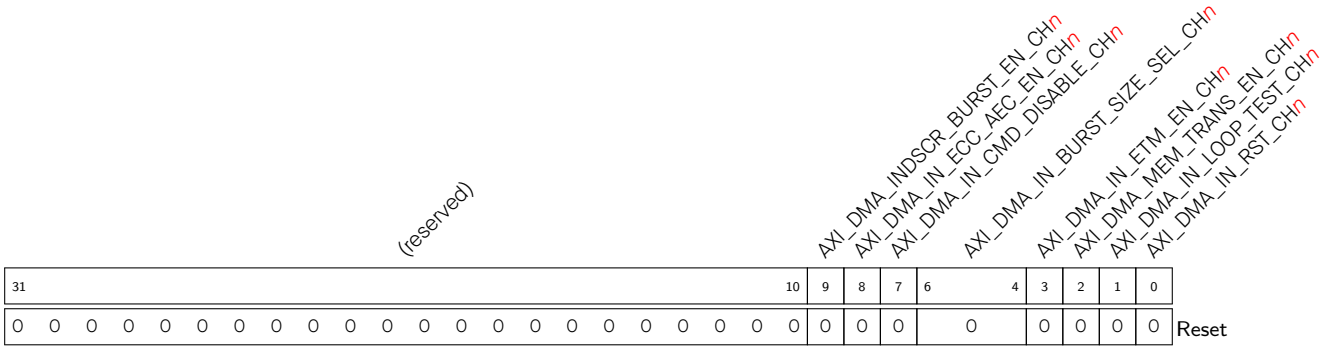
AXI_DMA_OUTFIFO_L2_UDF_CH*n*_INT_CLR Write 1 to clear AXI_DMA_OUTFIFO_L2_UDF_CH*n*_INT. (WT)

AXI_DMA_OUTFIFO_L3_OVF_CH*n*_INT_CLR Write 1 to clear AXI_DMA_OUTFIFO_L3_OVF_CH*n*_INT. (WT)

AXI_DMA_OUTFIFO_L3_UDF_CH*n*_INT_CLR Write 1 to clear AXI_DMA_OUTFIFO_L3_UDF_CH*n*_INT. (WT)

AXI_DMA_OUT_LINK_SWITCH_CH*n*_INT_CLR Write 1 to clear AXI_DMA_OUT_LINK_SWITCH_CH*n*_INT. (RO)

Register 6.77. AXI_DMA_IN_CONF0_CH*n*_REG (*n*: 0-2) (0x0010+0x68n*)**



AXI_DMA_IN_RST_CH*n* Write 1 and then 0 to reset RX channel *n* FSM and RX FIFO pointer.(R/W)

AXI_DMA_IN_LOOP_TEST_CH*n* Reserved. (R/W)

AXI_DMA_MEM_TRANS_EN_CH*n* Configures whether to enable memory-to-memory data transfer.

0: Disable

1: Enable

(R/W)

AXI_DMA_IN_ETM_EN_CH*n* Configures whether to enable ETM control for RX channel *n*.

0: Disable

1: Enable

(R/W)

AXI_DMA_IN_BURST_SIZE_SEL_CH*n* Configures the burst length for RX channel *n*.

0: 8 bytes

1: 16 bytes

2: 32 bytes

3: 64 bytes

4: 128 bytes

5 ~ 7: Invalid

(R/W)

AXI_DMA_IN_CMD_DISABLE_CH*n* Configures whether to disable command on RX channel *n*.

0: Enable

1: Disable

(R/W)

AXI_DMA_IN_ECC_AEC_EN_CH*n* Configures whether AXI DMA can access external memory space for ECC and AES via RX channel *n*.

0: Not access

1: Access

(R/W)

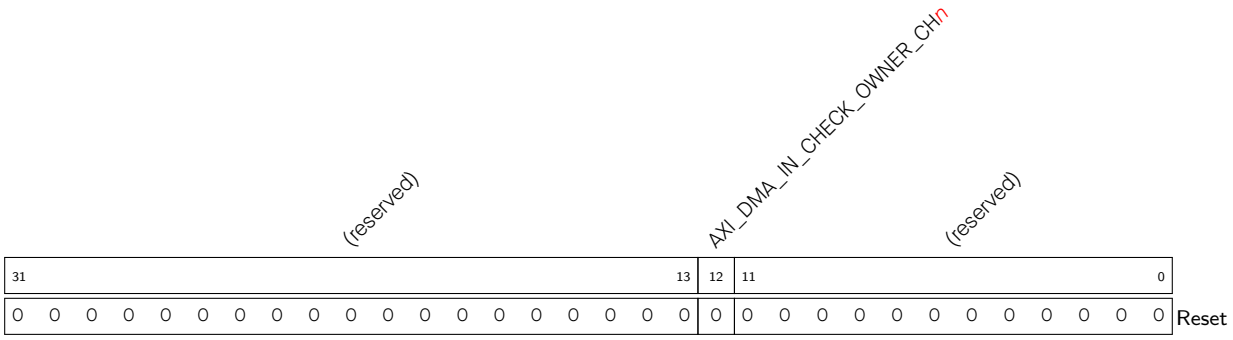
AXI_DMA_INDSCR_BURST_EN_CH*n* Configures whether to enable INCR burst transfer for RX channel *n* to read descriptors when accessing internal memory.

0: Disable

1: Enable

(R/W)

Register 6.78. AXI_DMA_IN_CONF1_CH*n*_REG (*n*: 0-2) (0x0014+0x68**n*)



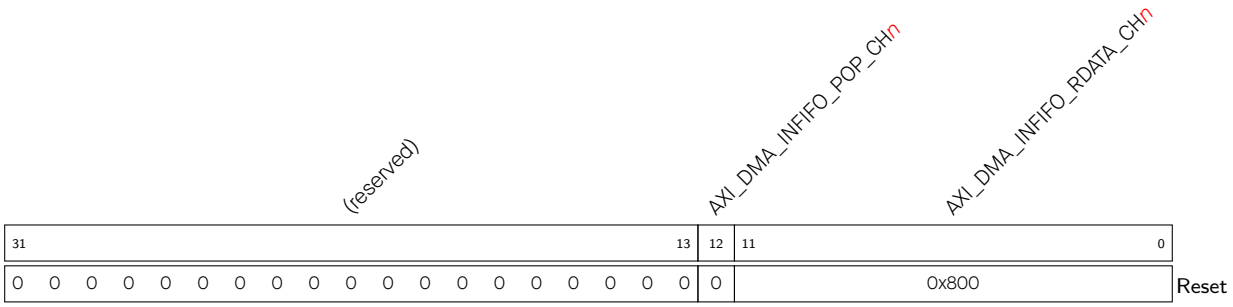
AXI_DMA_IN_CHECK_OWNER_CH*n* Configures whether to enable owner bit check for RX channel *n*.

0: Disable

1: Enable

(R/W)

Register 6.79. AXI_DMA_IN_POP_CH*n*_REG (*n*: 0-2) (0x001C+0x68**n*)



AXI_DMA_INFIFO_RDATA_CH*n* Represents the data popped from AXI DMA RX FIFO. (RO)

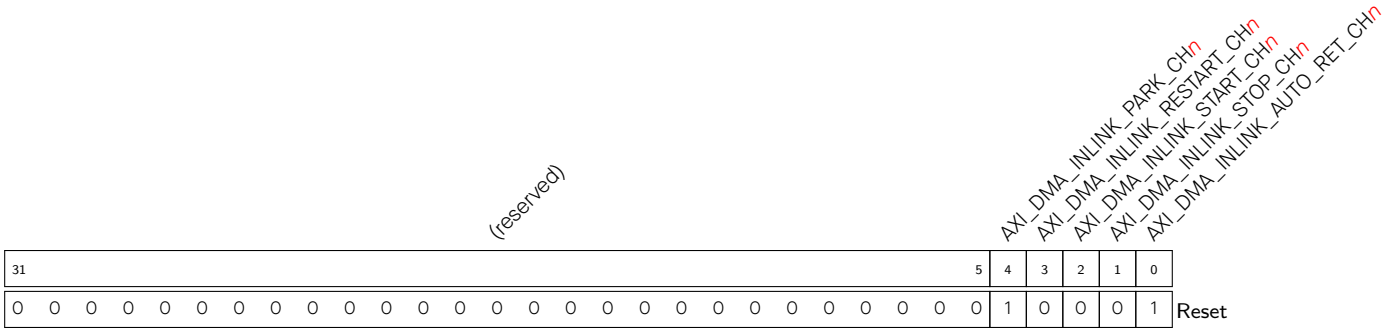
AXI_DMA_INFIFO_POP_CH*n* Configures whether to pop data from AXI DMA RX FIFO.

0: Invalid. No effect

1: Pop

(WT)

Register 6.80. AXI_DMA_IN_LINK1_CH*n*_REG (*n*: 0-2) (0x0020+0x68*n*)**



AXI_DMA_INLINK_AUTO_RET_CH*n* Configures whether to return to the current receive descriptor's address when there are some errors in current receiving data.

0: Not return

1: Return

(R/W)

AXI_DMA_INLINK_STOP_CH*n* Configures whether to stop RX channel *n* from receiving data.

0: Invalid. No effect

1: Stop

(WT)

AXI_DMA_INLINK_START_CH*n* Configures whether to enable RX channel *n* for data transfer.

0: Disable

1: Enable

(WT)

AXI_DMA_INLINK_RESTART_CH*n* Configures whether to restart RX channel *n* for AXI DMA transfer.

0: Invalid. No effect

1: Restart

(WT)

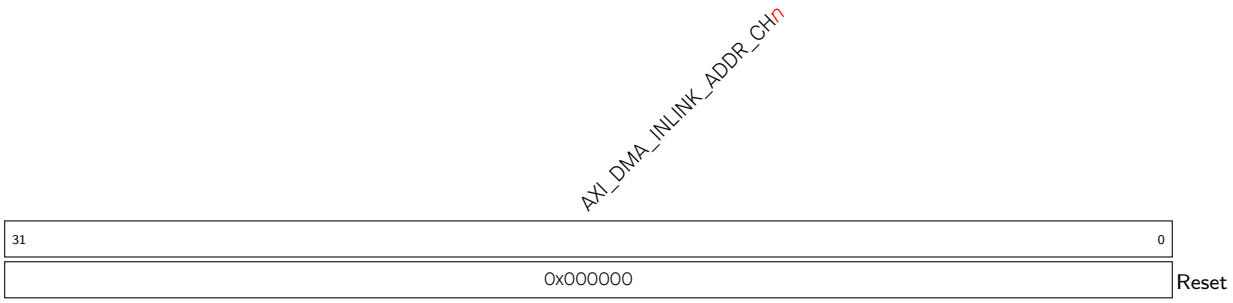
AXI_DMA_INLINK_PARK_CH*n* Represents the status of the receive descriptor's FSM.

0: Running

1: Idle

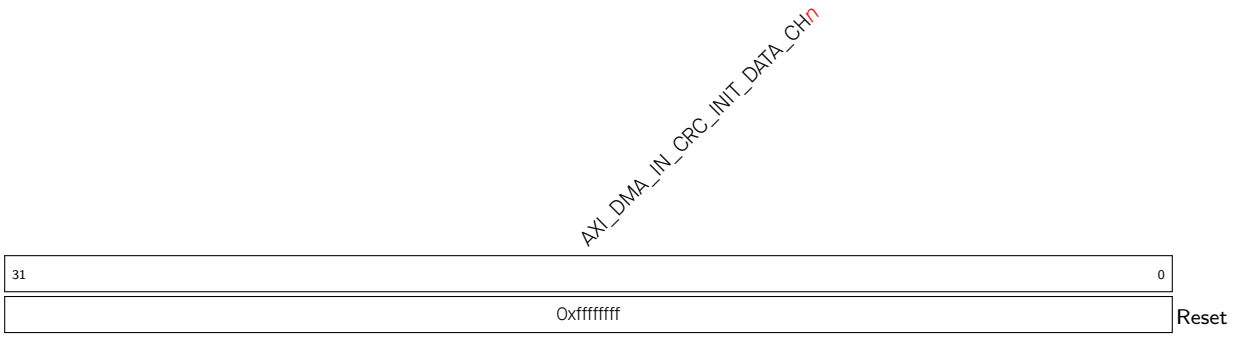
(RO)

Register 6.81. AXI_DMA_IN_LINK2_CH*n*_REG (*n*: 0-2) (0x0024+0x68**n*)



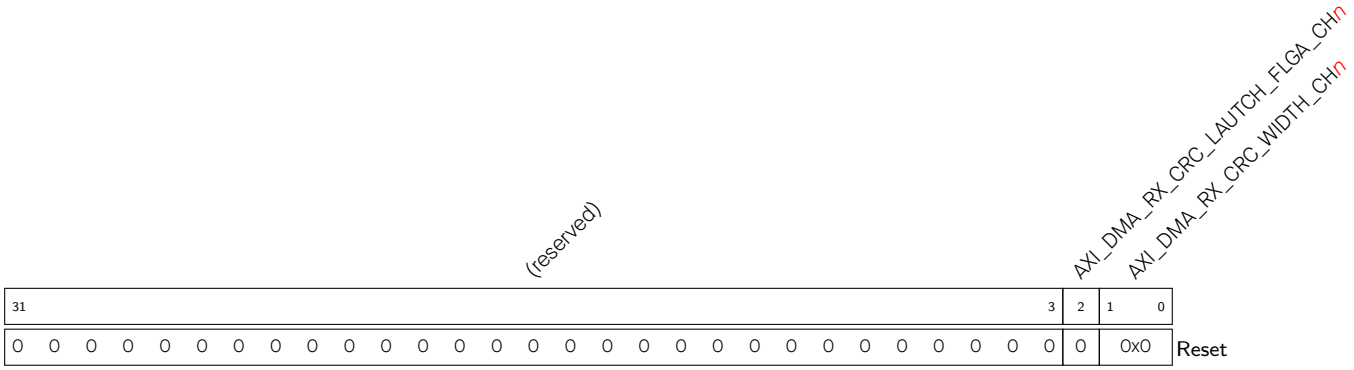
AXI_DMA_INLINK_ADDR_CH*n* Represents the first receive descriptor's address. (R/W)

Register 6.82. AXI_DMA_IN_CRC_INIT_DATA_CH*n*_REG (*n*: 0-2) (0x0048+0x68**n*)



AXI_DMA_IN_CRC_INIT_DATA_CH*n* Configures the CRC initial value for RX channel *n*. (R/W)

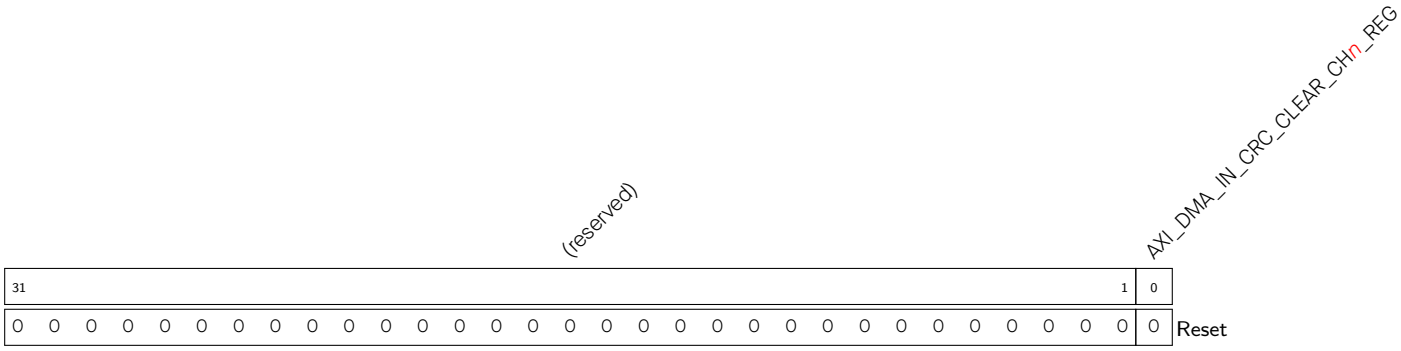
Register 6.83. AXI_DMA_RX_CRC_WIDTH_CH*n*_REG (*n*: 0-2) (0x004C+0x68n*)**



AXI_DMA_RX_CRC_WIDTH_CH*n* Configures the CRC result width for RX channel *n*. 0: $\leq$ 8 bits
1: $\leq$ 16 bits
2: $\leq$ 24 bits
3: $\leq$ 32 bits
(R/W)

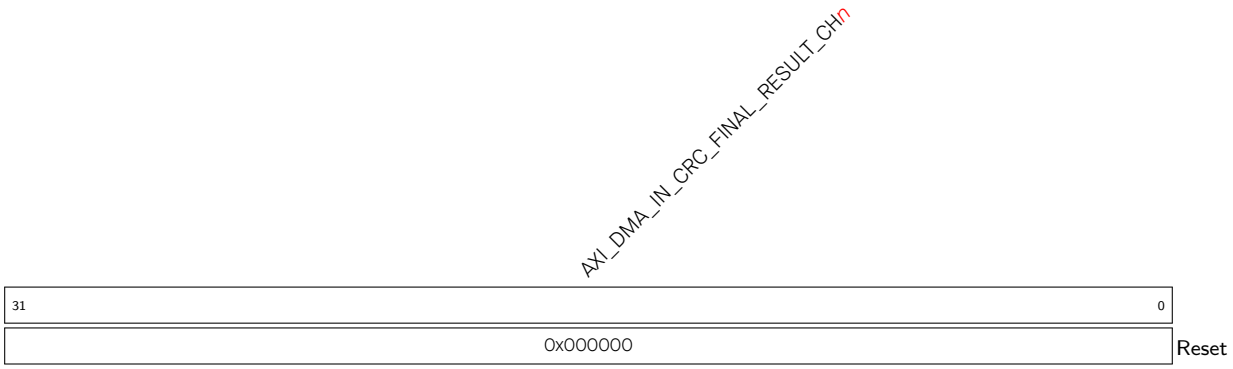
AXI_DMA_RX_CRC_LAUTCH_FLGA_CH*n* Write 1 and then 0 to latch the values of AXI_DMA_RX_CRC_EN_ADDR_CH*n*, AXI_DMA_RX_CRC_DATA_EN_ADDR_CH*n*, AXI_DMA_RX_CRC_EN_WR_DATA_CH*n*, and AXI_DMA_RX_CRC_DATA_EN_WR_DATA_CH*n*. (R/W)

Register 6.84. AXI_DMA_IN_CRC_CLEAR_CH*n*_REG (*n*: 0-2) (0x0050+0x68n*)**



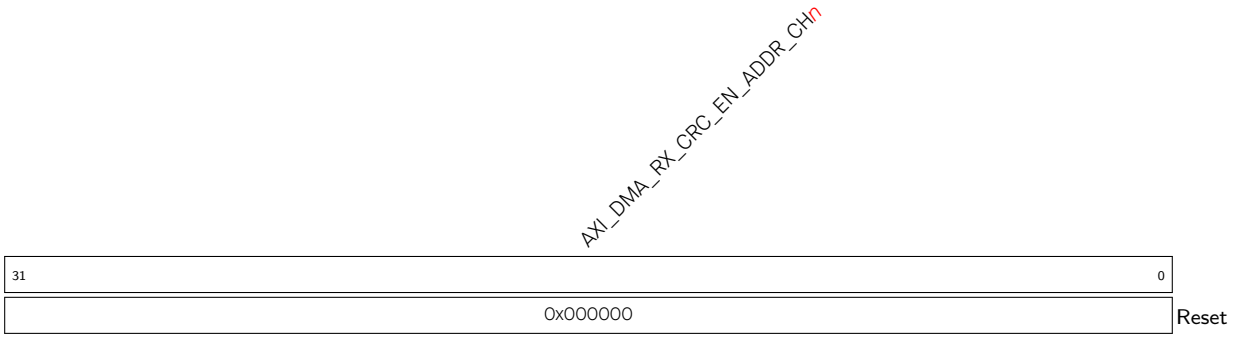
AXI_DMA_IN_CRC_CLEAR_CH*n*_REG Configures whether to clear the CRC result for RX channel *n*.
0: Not clear
1: Clear
(R/W)

Register 6.85. AXI_DMA_IN_CRC_FINAL_RESULT_CH*n*_REG (*n*: 0-2) (0x0054+0x68n*)**



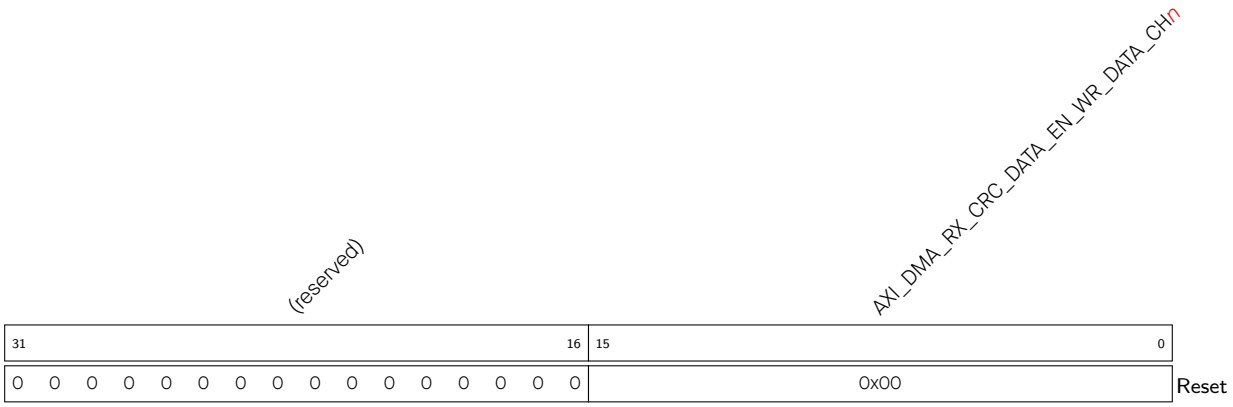
AXI_DMA_IN_CRC_FINAL_RESULT_CH*n* Represents the CRC result for RX channel *n*. (RO)

Register 6.86. AXI_DMA_RX_CRC_EN_ADDR_CH*n*_REG (*n*: 0-2) (0x005C+0x68n*)**



AXI_DMA_RX_CRC_EN_ADDR_CH*n* Configures at which bit of the CRC result the AXI_DMA_RX_CRC_EN_WR_DATA_CH*n*_REG register targets for TX channel *n*. (R/W)

Register 6.87. AXI_DMA_RX_CRC_DATA_EN_WR_DATA_CH*n*_REG (*n*: 0-2) (0x0060+0x68n*)**



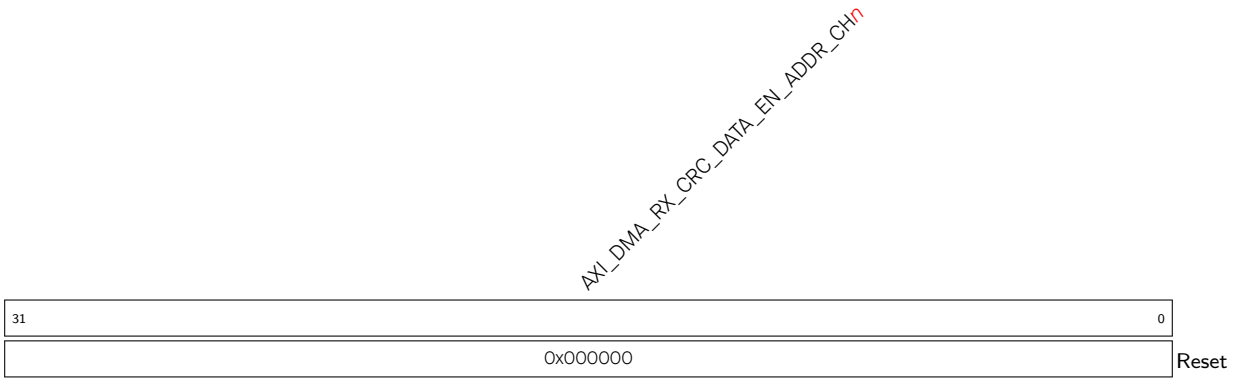
AXI_DMA_RX_CRC_DATA_EN_WR_DATA_CH*n* Configures whether to include each bit of the data in the CRC calculation matrix for TX channel *n*.

0: Not include

1: Include

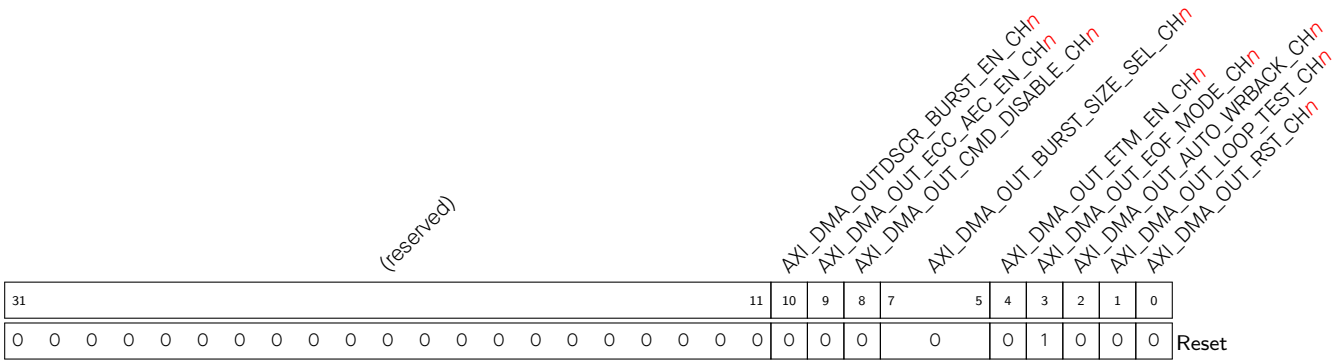
(R/W)

Register 6.88. AXI_DMA_RX_CRC_DATA_EN_ADDR_CH*n*_REG (*n*: 0-2) (0x0064+0x68n*)**



AXI_DMA_RX_CRC_DATA_EN_ADDR_CH*n* Configures at which bit of the CRC result the AXI_DMA_RX_CRC_DATA_EN_WR_DATA_CH*n*_REG register targets for TX channel *n*. (R/W)

Register 6.89. AXI_DMA_OUT_CONFO_CHn_REG(n: 0-2) (0x0148+0x68*n)


(reserved)
AXI_DMA_OUTDSCR_BURST_EN_CHn
AXI_DMA_OUT_ECC_AEC_EN_CHn
AXI_DMA_OUT_CMD_DISABLE_CHn
AXI_DMA_OUT_BURST_SIZE_SEL_CHn
AXI_DMA_OUT_ETM_EN_CHn
AXI_DMA_OUT_EOF_MODE_CHn
AXI_DMA_OUT_AUTO_WRBACK_CHn
AXI_DMA_OUT_LOOP_TEST_CHn
AXI_DMA_OUT_RST_CHn
31 11 10 9 8 7 5 4 3 2 1 0
0 0 0

Register 6.89. AXI_DMA_OUT_CONF0_CH*n*_REG(*n*: 0-2) (0x0148+0x68**n*)

Continued from the previous page...

AXI_DMA_OUT_ECC_AEC_EN_CH*n* Configures whether AXI DMA can access external memory space for ECC and AES via TX channel *n*.

0: Not access

1: Access

(R/W)

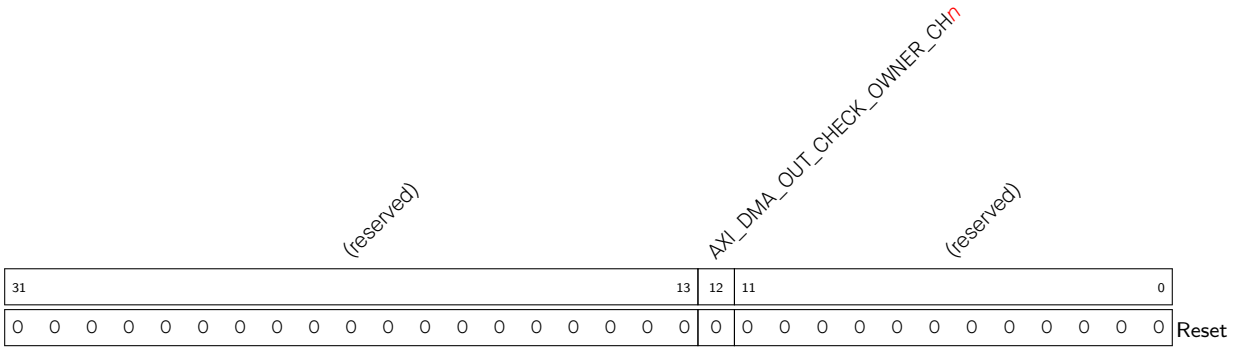
AXI_DMA_OUTDSCR_BURST_EN_CH*n* Configures whether to enable INCR burst transfer for TX channel *n* reading descriptors when accessing internal memory.

0: Disable

1: Enable

(R/W)

Register 6.90. AXI_DMA_OUT_CONF1_CH*n*_REG (*n*: 0-2) (0x014C+0x68**n*)



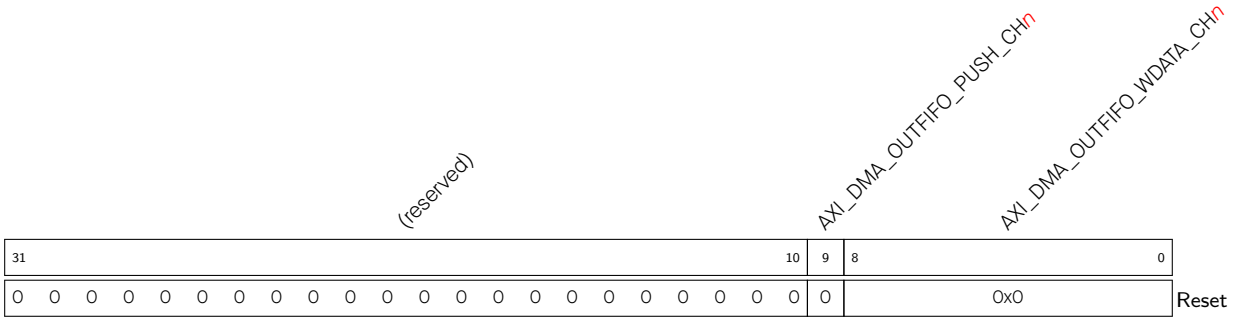
AXI_DMA_OUT_CHECK_OWNER_CH*n* Configures whether to enable owner bit check for TX channel *n*.

0: Disable

1: Enable

(R/W)

Register 6.91. AXI_DMA_OUT_PUSH_CH*n*_REG (*n*: 0-2) (0x0154+0x68*n*)**



AXI_DMA_OUTFIFO_WDATA_CH*n* Represents the data that need to be pushed into AXI DMA TX FIFO. (R/W)

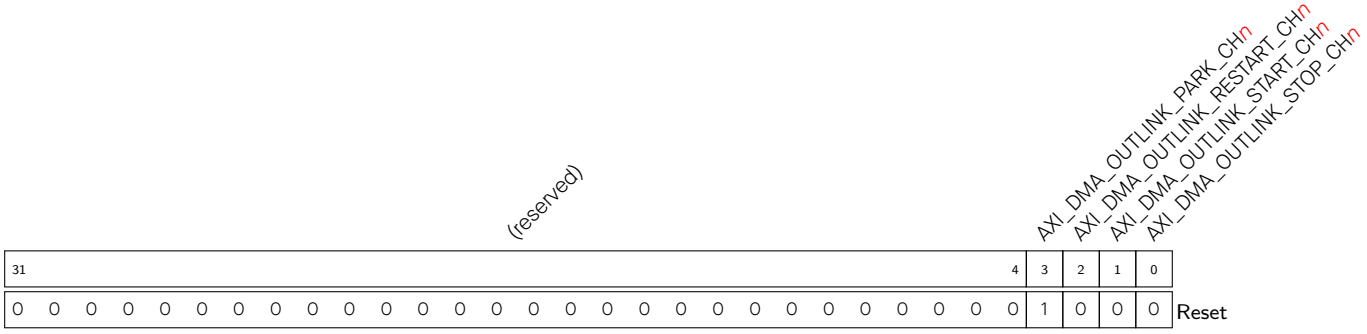
AXI_DMA_OUTFIFO_PUSH_CH*n* Configures whether to push data into AXI DMA TX FIFO.

0: Invalid. No effect

1: Push

(WT)

Register 6.92. AXI_DMA_OUT_LINK1_CH*n*_REG (*n*: 0-2) (0x0158+0x68n*)**



AXI_DMA_OUTLINK_STOP_CH*n* Configures whether to stop TX channel *n* from transmitting data.

0: Invalid. No effect

1: Stop

(WT)

AXI_DMA_OUTLINK_START_CH*n* Configures whether to enable TX channel *n* for data transfer.

0: Disable

1: Enable

(WT)

AXI_DMA_OUTLINK_RESTART_CH*n* Configures whether to restart TX channel *n* for AXI DMA transfer.

0: Invalid. No effect

1: Restart

(WT)

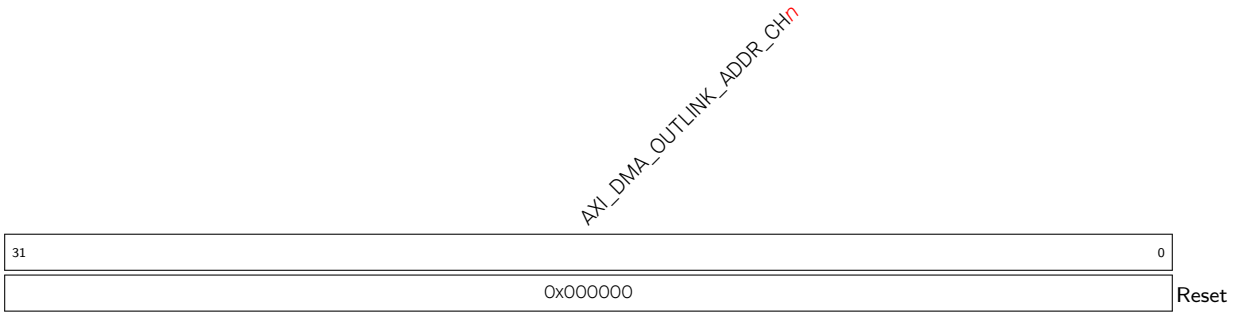
AXI_DMA_OUTLINK_PARK_CH*n* Represents the status of the transmit descriptor's FSM.

0: Running

1: Idle

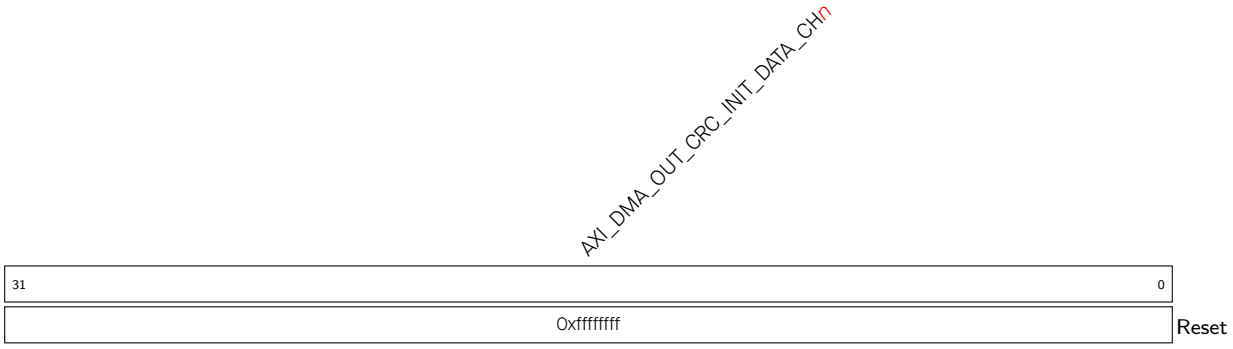
(RO)

Register 6.93. AXI_DMA_OUT_LINK2_CH*n*_REG (*n*: 0-2) (0x015C+0x68n*)**



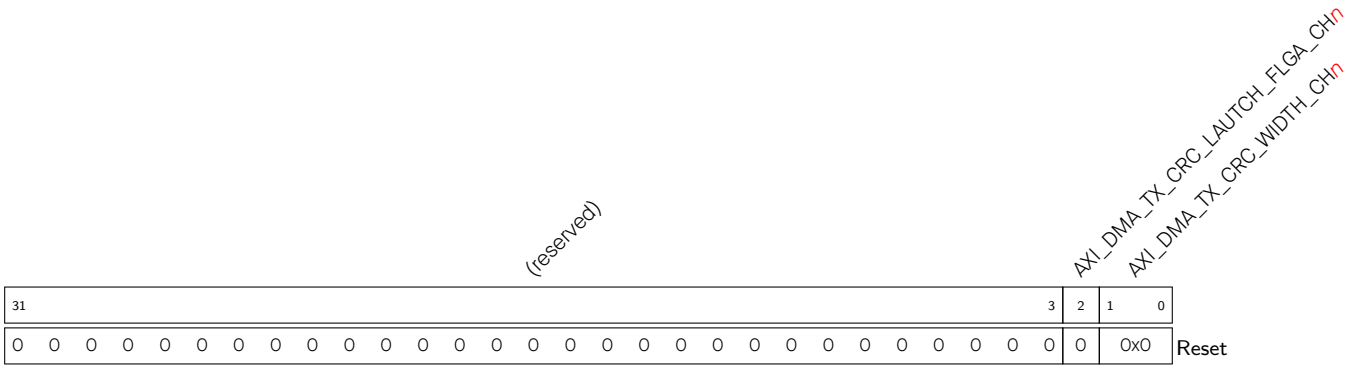
AXI_DMA_OUTLINK_ADDR_CH*n* Represents the first transmit descriptor's address. (R/W)

Register 6.94. AXI_DMA_OUT_CRC_INIT_DATA_CH*n*_REG (*n*: 0-2) (0x0180+0x68n*)**



AXI_DMA_OUT_CRC_INIT_DATA_CH*n* Configures the CRC initial value for TX channel *n*. (R/W)

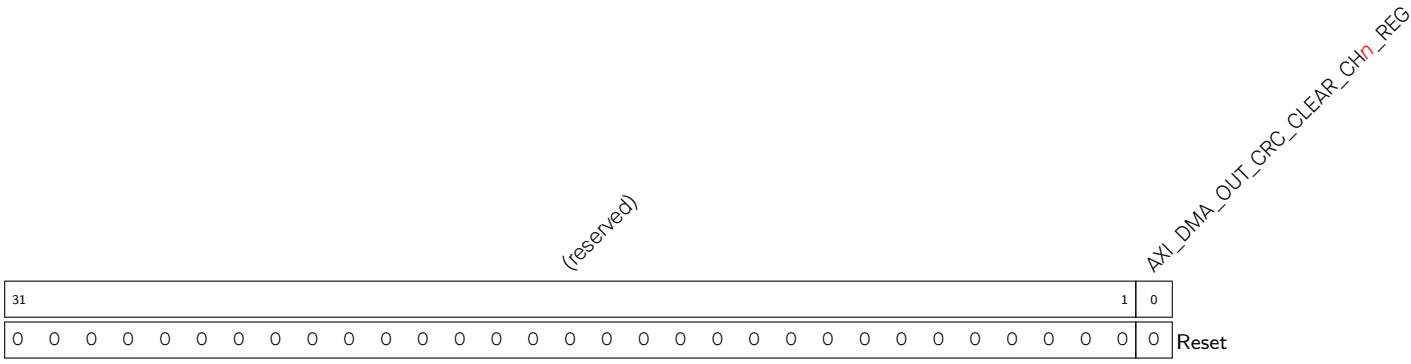
Register 6.95. AXI_DMA_TX_CRC_WIDTH_CH*n*_REG (*n*: 0-2) (0x0184+0x68n*)**



AXI_DMA_TX_CRC_WIDTH_CH*n* Configures the CRC result width for TX channel *n*. 0: $\leq$ 8 bits
1: $\leq$ 16 bits
2: $\leq$ 24 bits
3: $\leq$ 32 bits
(R/W)

AXI_DMA_TX_CRC_LAUTCH_FLGA_CH*n* Write 1 and then 0 to latch the values of AXI_DMA_TX_CRC_EN_ADDR_CH*n*, AXI_DMA_TX_CRC_DATA_EN_ADDR_CH*n*, AXI_DMA_TX_CRC_EN_WR_DATA_CH*n*, and AXI_DMA_TX_CRC_DATA_EN_WR_DATA_CH*n*. (R/W)

Register 6.96. AXI_DMA_OUT_CRC_CLEAR_CH*n*_REG (*n*: 0-2) (0x0188+0x68n*)**



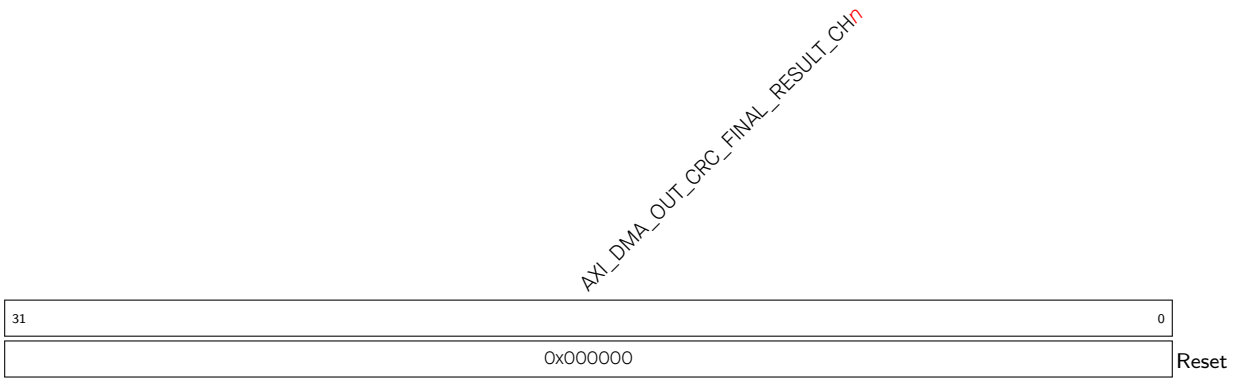
AXI_DMA_OUT_CRC_CLEAR_CH*n*_REG Configures whether to clear the CRC result for TX channel *n*.

0: Not clear

1: Clear

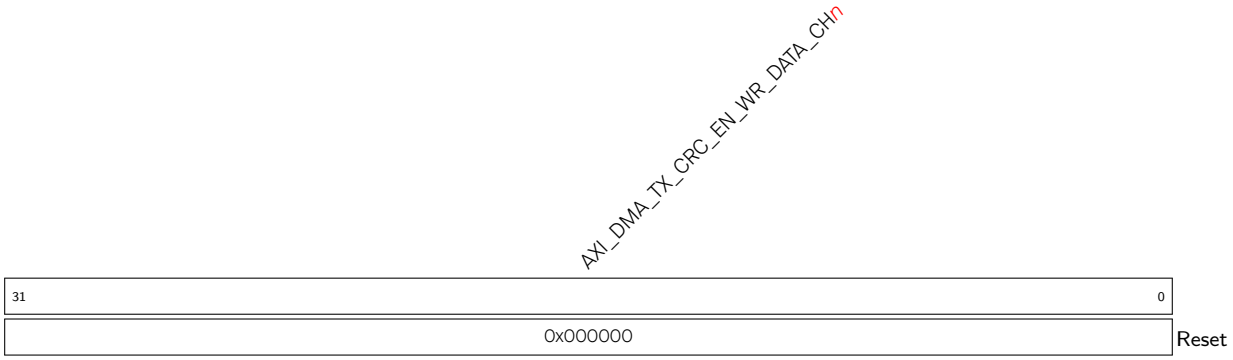
(R/W)

Register 6.97. AXI_DMA_OUT_CRC_FINAL_RESULT_CH*n*_REG (*n*: 0-2) (0x018C+0x68n*)**



AXI_DMA_OUT_CRC_FINAL_RESULT_CH*n* Represents the CRC result for TX channel *n*. (RO)

Register 6.98. AXI_DMA_TX_CRC_EN_WR_DATA_CH*n*_REG (*n*: 0-2) (0x0190+0x68**n*)



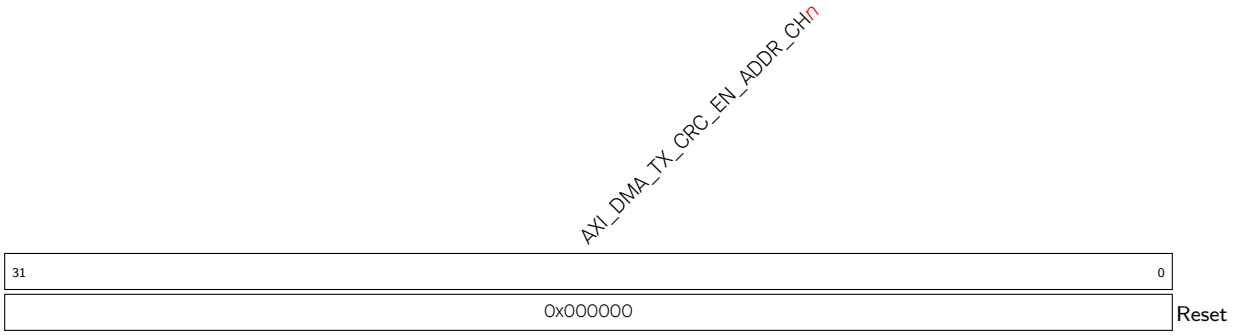
AXI_DMA_TX_CRC_EN_WR_DATA_CH*n* Configures whether to include each bit of the intermediate result in the CRC calculation matrix for TX channel *n*.

0: Not include

1: Include

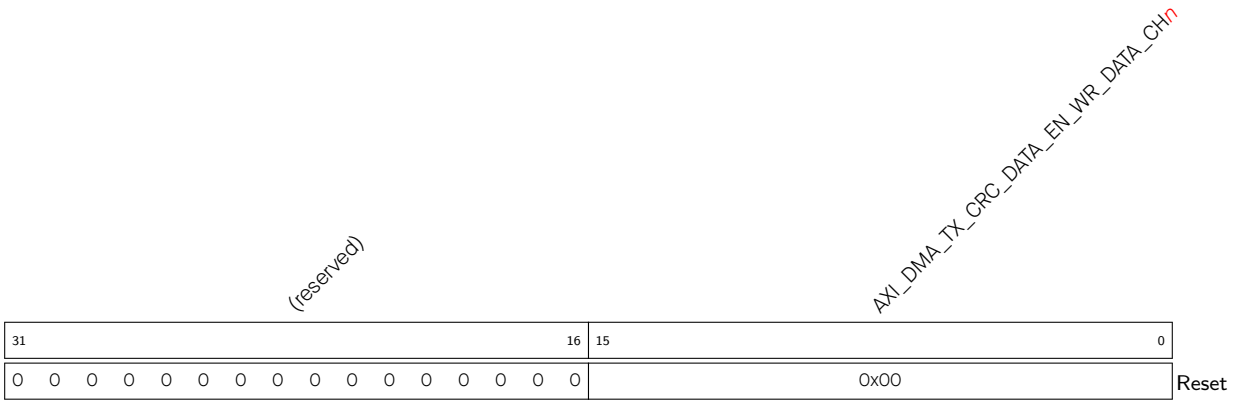
(R/W)

Register 6.99. AXI_DMA_TX_CRC_EN_ADDR_CH*n*_REG (*n*: 0-2) (0x0194+0x68**n*)



AXI_DMA_TX_CRC_EN_ADDR_CH*n* Configures at which bit of the CRC result the AXI_DMA_TX_CRC_EN_WR_DATA_CH*n*_REG register targets for TX channel *n*. (R/W)

Register 6.100. AXI_DMA_TX_CRC_DATA_EN_WR_DATA_CH*n*_REG (*n*: 0-2) (0x0198+0x68**n*)



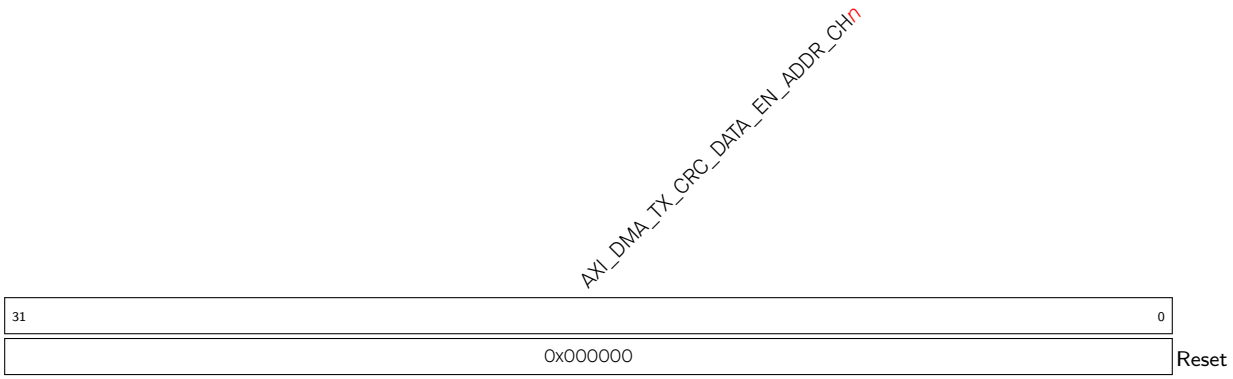
AXI_DMA_TX_CRC_DATA_EN_WR_DATA_CH*n* Configures whether to include each bit of the input data in the CRC calculation matrix for TX channel *n*.

0: Not include

1: Include

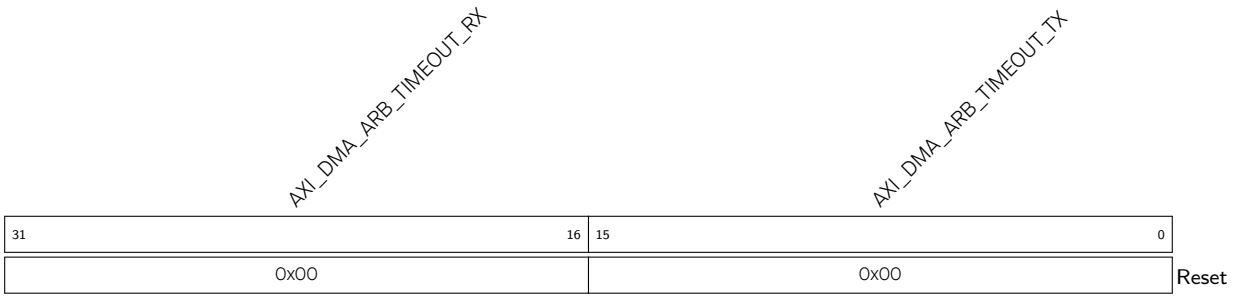
(R/W)

Register 6.101. AXI_DMA_TX_CRC_DATA_EN_ADDR_CH*n*_REG (*n*: 0-2) (0x019C+0x68**n*)



AXI_DMA_TX_CRC_DATA_EN_ADDR_CH*n* Configures at which bit of the CRC result the AAXI_DMA_TX_CRC_DATA_EN_WR_DATA_CH*n*_REG register targets for TX channel *n*.(R/W)

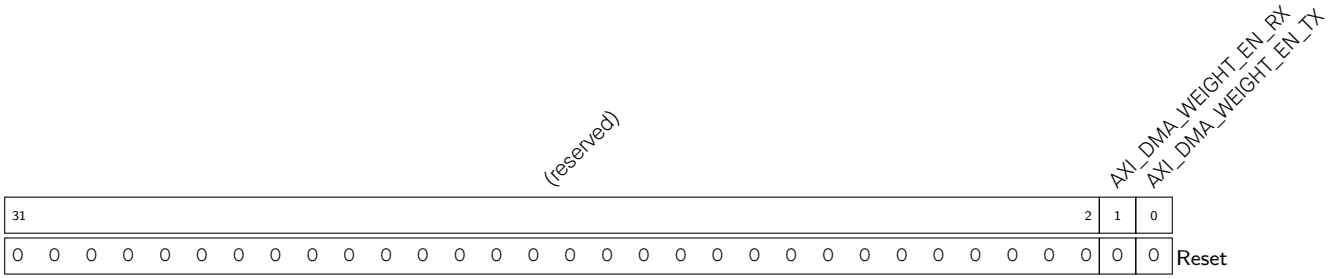
Register 6.102. AXI_DMA_ARB_TIMEOUT_REG (0x0270)



AXI_DMA_ARB_TIMEOUT_TX Configures the time slot for TX. Measurement unit: AXI bus clock cycle. (R/W)

AXI_DMA_ARB_TIMEOUT_RX Configures the time slot for RX. Measurement unit: AXI bus clock cycle. (R/W)

Register 6.103. AXI_DMA_WEIGHT_EN_REG (0x0274)



AXI_DMA_WEIGHT_EN_TX Configures whether to enable weight arbitration for TX.

0: Disable

1: Enable

(R/W)

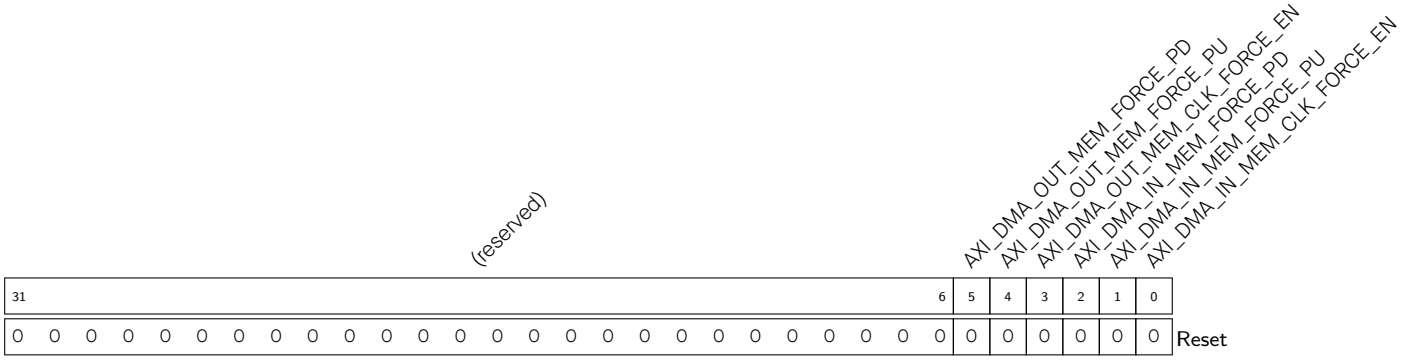
AXI_DMA_WEIGHT_EN_RX Configures whether to enable weight arbitration for RX.

0: Disable

1: Enable

(R/W)

Register 6.104. AXI_DMA_IN_MEM_CONF_REG (0x0278)



AXI_DMA_IN_MEM_CLK_FORCE_EN Configures memory clock gating for RX.

0: Support clock only when AXI DMA accesses memory

1: Always force the clock on

(R/W)

AXI_DMA_IN_MEM_FORCE_PU Configures whether to force power up memory for RX.

0: Not force power up

1: Force power up

(R/W)

AXI_DMA_IN_MEM_FORCE_PD Configures whether to force power down memory for RX.

0: Not force power down

1: Force power down

(R/W)

AXI_DMA_OUT_MEM_CLK_FORCE_EN Configures memory clock gating for TX.

0: Support clock only when AXI DMA accesses memory

1: Always force the clock on

(R/W)

AXI_DMA_OUT_MEM_FORCE_PU Configures whether to force power up memory for TX.

0: Not force power up

1: Force power up

(R/W)

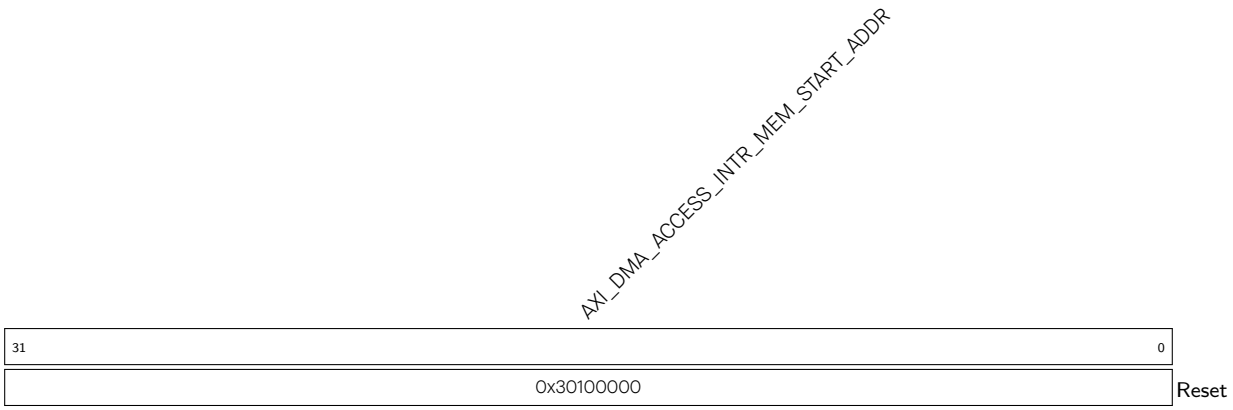
AXI_DMA_OUT_MEM_FORCE_PD Configures whether to force power down memory for TX.

0: Not force power down

1: Force power down

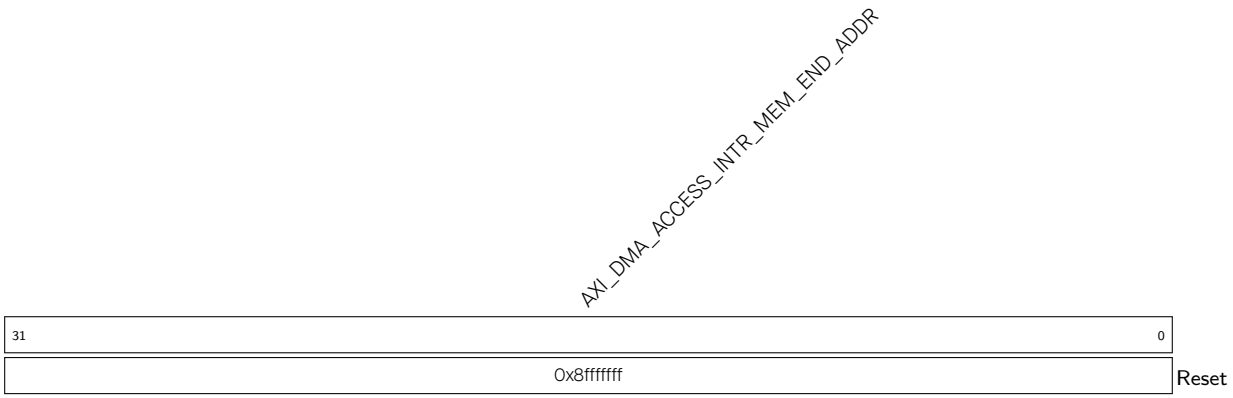
(R/W)

Register 6.105. AXI_DMA_INTR_MEM_START_ADDR_REG (0x027C)



AXI_DMA_ACCESS_INTR_MEM_START_ADDR Configures the start address of the accessible internal memory address space. (R/W)

Register 6.106. AXI_DMA_INTR_MEM_END_ADDR_REG (0x0280)



AXI_DMA_ACCESS_INTR_MEM_END_ADDR Configures the end address of the accessible internal memory address space. (R/W)

Register 6.107. AXI_DMA_EXTR_MEM_START_ADDR_REG (0x0284)



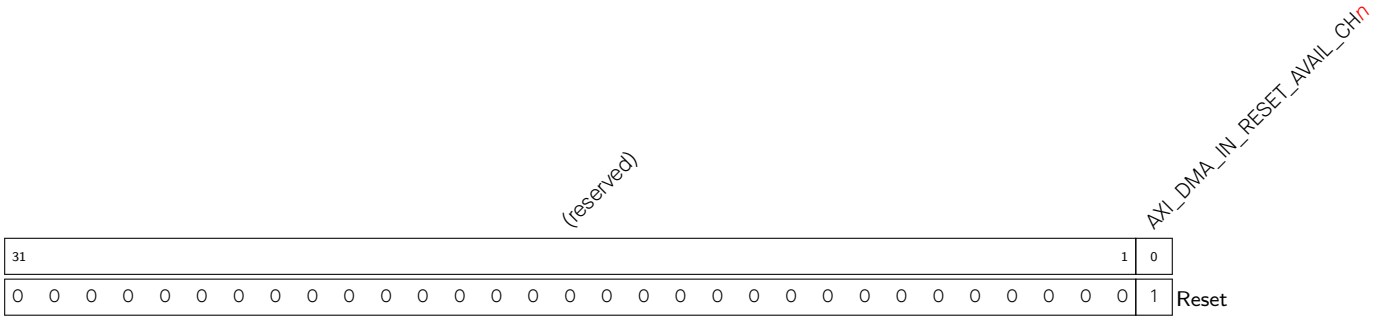
AXI_DMA_ACCESS_EXTR_MEM_START_ADDR Configures the start address of the accessible external memory address space. (R/W)

Register 6.108. AXI_DMA_EXTR_MEM_END_ADDR_REG (0x0288)



AXI_DMA_ACCESS_EXTR_MEM_END_ADDR Configures the end address of the accessible external memory address space. (R/W)

Register 6.109. AXI_DMA_IN_RESET_AVAIL_CH*n*_REG (*n*: 0-2) (0x028C+0x4n*)**



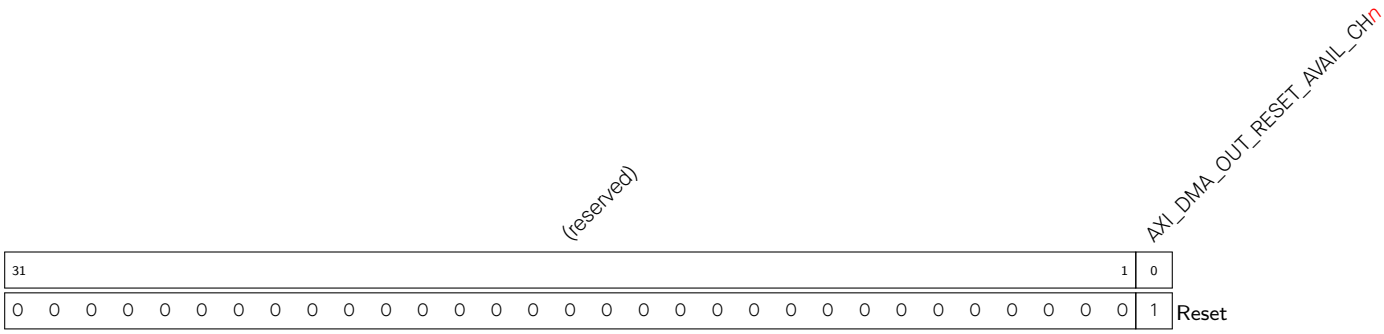
AXI_DMA_IN_RESET_AVAIL_CH*n* Represents whether RX channel *n* can be reset.

0: Cannot be reset by software

1: Can be reset

(RO)

Register 6.110. AXI_DMA_OUT_RESET_AVAIL_CH*n*_REG (*n*: 0-2) (0x0298+0x4n*)**



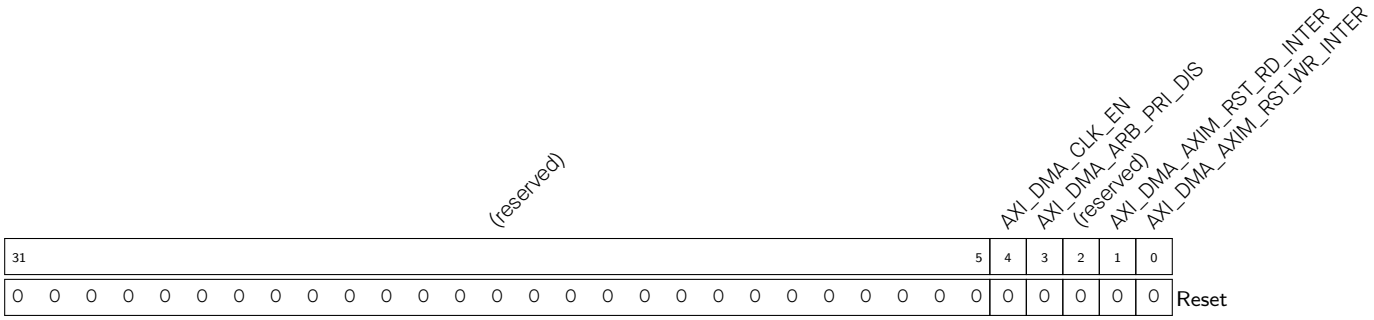
AXI_DMA_OUT_RESET_AVAIL_CH*n* Represents whether TX channel *n* can be reset.

0: Cannot be reset by software

1: Can be reset

. (RO)

Register 6.111. AXI_DMA_MISC_CONF_REG (0x02A8)



AXI_DMA_AXIM_RST_WR_INTER Write 1 and then 0 to reset the internal AXI write FSM. (R/W)

AXI_DMA_AXIM_RST_RD_INTER Write 1 and then 0 to reset the internal AXI read FSM. (R/W)

AXI_DMA_ARB_PRI_DIS Configures whether to disable the priority arbitration.

0: Enable

1: Disable

(R/W)

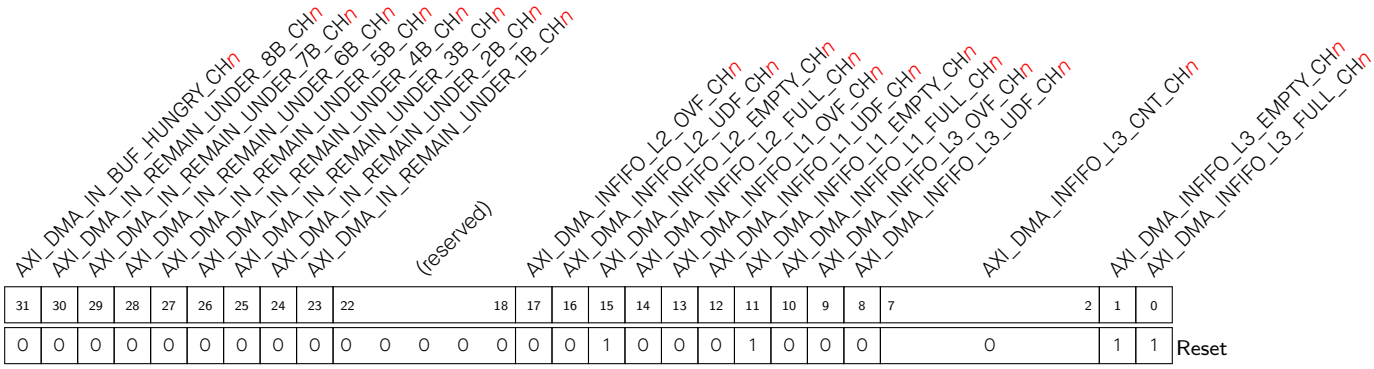
AXI_DMA_CLK_EN Configures AXI DMA clock gating.

0: Support clock only when the application writes registers.

1: Always force the clock on for registers.

(R/W)

Register 6.112. AXI_DMA_INFIFO_STATUS_CH*n*_REG (*n*: 0-2) (0x0018+0x68**n*)



AXI_DMA_INFIFO_L3_FULL_CH*n* Represents whether L3 RX FIFO is full.

0: Not Full

1: Full

(RO)

AXI_DMA_INFIFO_L3_EMPTY_CH*n* Represents whether L3 RX FIFO is empty.

0: Not empty

1: Empty

(RO)

AXI_DMA_INFIFO_L3_CNT_CH*n* Represents the number of data bytes in L3 RX FIFO for RX channel *n*. (RO)

AXI_DMA_INFIFO_L3_UDF_CH*n* Represents whether L3 RX FIFO underflows.

0: No underflow

1: Underflow

(RO)

AXI_DMA_INFIFO_L3_OVF_CH*n* Represents whether L3 RX FIFO overflows.

0: No overflow

1: Overflow

(RO)

AXI_DMA_INFIFO_L1_FULL_CH*n* Represents whether L1 RX FIFO is full.

0: Not Full

1: Full

(RO)

AXI_DMA_INFIFO_L1_EMPTY_CH*n* Represents whether L1 RX FIFO is empty.

0: Not empty

1: Empty

(RO)

AXI_DMA_INFIFO_L1_UDF_CH*n* Represents whether L1 RX FIFO underflows.

0: No underflow

1: Underflow

(RO)

Continued on the next page...

Register 6.112. AXI_DMA_INFIFO_STATUS_CH*n*_REG (*n*: 0-2) (0x0018+0x68**n*)

Continued from the previous page...

AXI_DMA_INFIFO_L1_OVF_CH*n* Represents whether L1 RX FIFO overflows.
0: No overflow
1: Overflow
(RO)

AXI_DMA_INFIFO_L2_FULL_CH*n* Represents whether L2 RX FIFO is full.
0: Not Full
1: Full
(RO)

AXI_DMA_INFIFO_L2_EMPTY_CH*n* Represents whether L1 RX FIFO is empty.
0: Not empty
1: Empty
(RO)

AXI_DMA_INFIFO_L2_UDF_CH*n* Represents whether L2 RX FIFO underflows.
0: No underflow
1: Underflow
(RO)

AXI_DMA_INFIFO_L2_OVF_CH*n* Represents whether L2 RX FIFO overflows.
0: No overflow
1: Overflow
(RO)

AXI_DMA_IN_REMAIN_UNDER_1B_CH*n* Reserved.(RO)

AXI_DMA_IN_REMAIN_UNDER_2B_CH*n* Reserved.(RO)

AXI_DMA_IN_REMAIN_UNDER_3B_CH*n* Reserved.(RO)

AXI_DMA_IN_REMAIN_UNDER_4B_CH*n* Reserved.(RO)

AXI_DMA_IN_REMAIN_UNDER_5B_CH*n* Reserved.(RO)

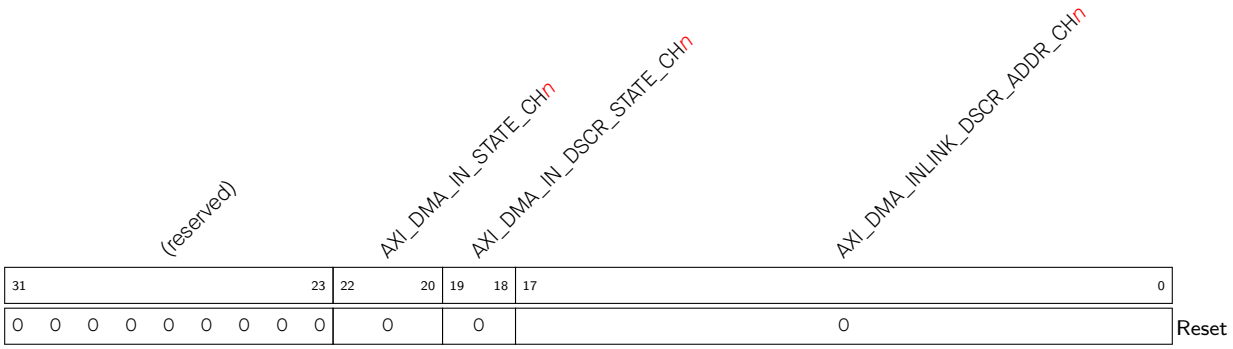
AXI_DMA_IN_REMAIN_UNDER_6B_CH*n* Reserved.(RO)

AXI_DMA_IN_REMAIN_UNDER_7B_CH*n* Reserved.(RO)

AXI_DMA_IN_REMAIN_UNDER_8B_CH*n* Reserved.(RO)

AXI_DMA_IN_BUF_HUNGRY_CH*n* Reserved.(RO)

Register 6.113. AXI_DMA_IN_STATE_CH*n*_REG (*n*: 0-2) (0x0028+0x68**n*)

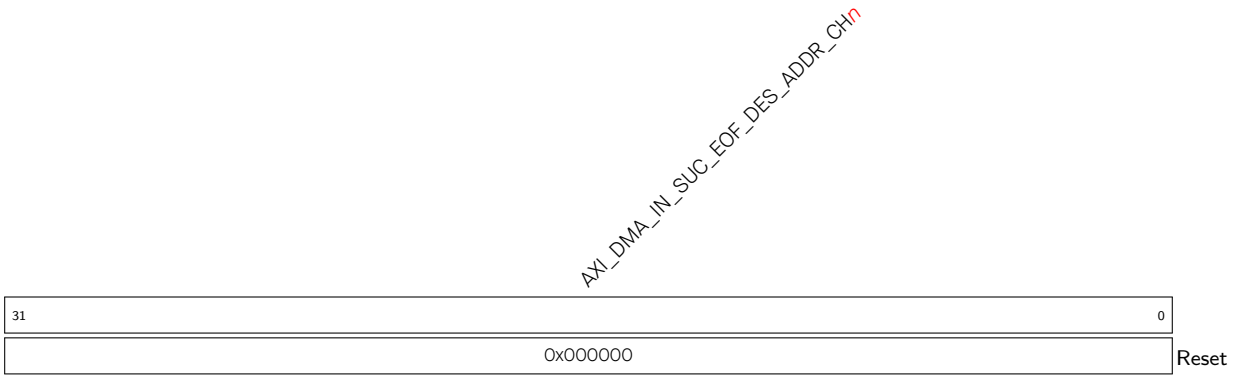


AXI_DMA_INLINK_DSCR_ADDR_CH*n* Represents the lower 18 bits of the next receive descriptor address that is pre-read (but not processed yet). If the current receive descriptor is the last descriptor, then this field represents the address of the current receive descriptor. (RO)

AXI_DMA_IN_DSCR_STATE_CH*n* Reserved. (RO)

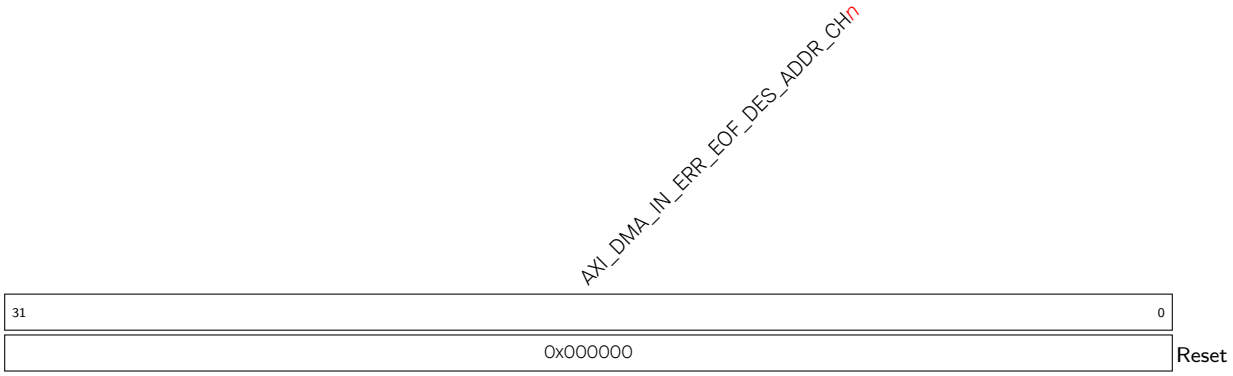
AXI_DMA_IN_STATE_CH*n* Reserved. (RO)

Register 6.114. AXI_DMA_IN_SUC_EOF_DES_ADDR_CH*n*_REG (*n*: 0-2) (0x002C+0x68**n*)



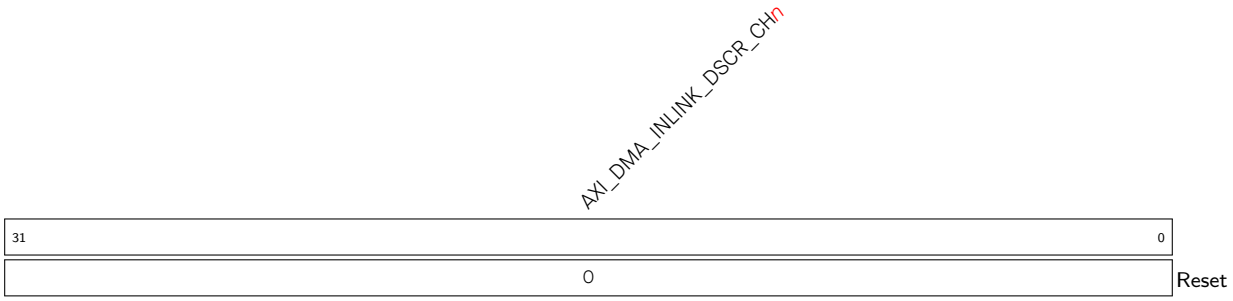
AXI_DMA_IN_SUC_EOF_DES_ADDR_CH*n* Represents the address of the receive descriptor when the EOF bit in this descriptor is 1. (RO)

Register 6.115. AXI_DMA_IN_ERR_EOF_DES_ADDR_CH*n*_REG (*n*: 0-2) (0x0030+0x68**n*)



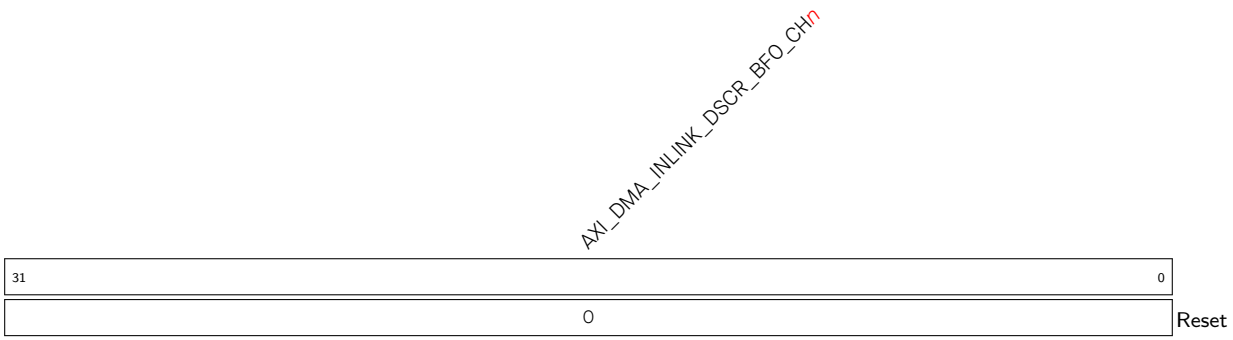
AXI_DMA_IN_ERR_EOF_DES_ADDR_CH*n* Represents the address of the receive descriptor when there are some errors in the currently received data. Valid only for UHCI. (RO)

Register 6.116. AXI_DMA_IN_DSCR_CH*n*_REG (*n*: 0-2) (0x0034+0x68**n*)



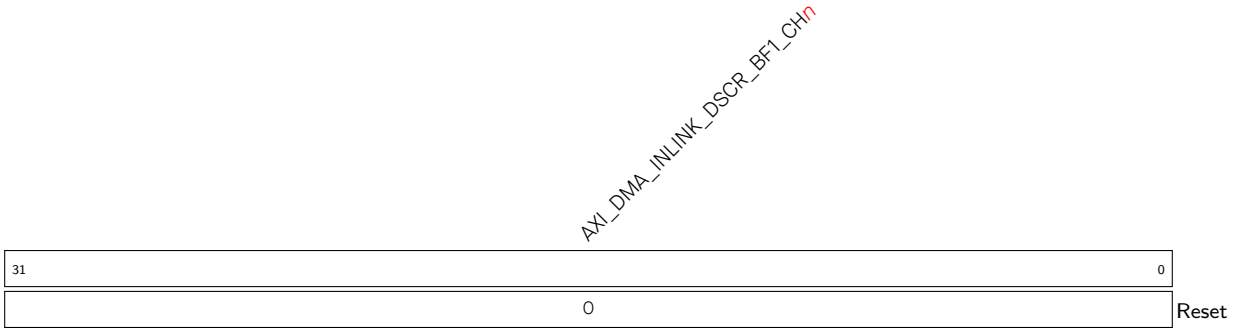
AXI_DMA_INLINK_DSCR_CH*n* Represents the address of the next receive descriptor x+1 pointed by the current receive descriptor that is pre-read. (RO)

Register 6.117. AXI_DMA_IN_DSCR_BF0_CH*n*_REG (*n*: 0-2) (0x0038+0x68**n*)



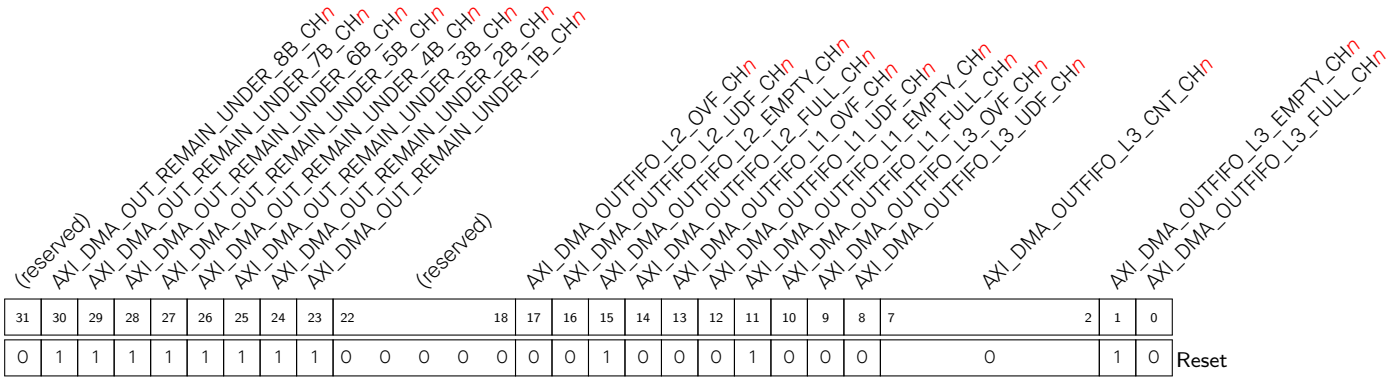
AXI_DMA_INLINK_DSCR_BF0_CH*n* Represents the address of the current receive descriptor x that is pre-read. (RO)

Register 6.118. AXI_DMA_IN_DSCR_BF1_CH*n*_REG (*n*: 0-2) (0x003C+0x68*n*)**



AXI_DMA_INLINK_DSCR_BF1_CH*n* Represents the address of the previous receive descriptor x-1 that is pre-read. (RO)

Register 6.119. AXI_DMA_OUTFIFO_STATUS_CH*n*_REG (*n*: 0-2) (0x0150+0x68**n*)



AXI_DMA_OUTFIFO_L3_FULL_CH*n* Represents whether L3 TX FIFO is full.

0: Not Full

1: Full

(RO)

AXI_DMA_OUTFIFO_L3_EMPTY_CH*n* Represents whether L3 TX FIFO is empty.

0: Not empty

1: Empty

(RO)

AXI_DMA_OUTFIFO_L3_CNT_CH*n* Represents the number of data bytes in L3 TX FIFO for TX channel *n*. (RO)

AXI_DMA_OUTFIFO_L3_UDF_CH*n* Represents whether L3 TX FIFO underflows.

0: No underflow

1: Underflow

(RO)

AXI_DMA_OUTFIFO_L3_OVF_CH*n* Represents whether L3 TX FIFO overflows.

0: No overflow

1: Overflow

(RO)

AXI_DMA_OUTFIFO_L1_FULL_CH*n* Represents whether L1 TX FIFO is full.

0: Not Full

1: Full

(RO)

AXI_DMA_OUTFIFO_L1_EMPTY_CH*n* Represents whether L1 TX FIFO is empty.

0: Not empty

1: Empty

(RO)

AXI_DMA_OUTFIFO_L1_UDF_CH*n* Represents whether L1 TX FIFO underflows.

0: No underflow

1: Underflow

(RO)

Continued on the next page...

Register 6.119. AXI_DMA_OUTFIFO_STATUS_CH*n*_REG (*n*: 0-2) (0x0150+0x68**n*)

Continued from the previous page...

AXI_DMA_OUTFIFO_L1_OVF_CH*n* Represents whether L1 TX FIFO overflows.
0: No overflow
1: Overflow
(RO)

AXI_DMA_OUTFIFO_L2_FULL_CH*n* Represents whether L2 TX FIFO is full.
0: Not Full
1: Full
(RO)

AXI_DMA_OUTFIFO_L2_EMPTY_CH*n* Represents whether L2 TX FIFO is empty.
0: Not empty
1: Empty
(RO)

AXI_DMA_OUTFIFO_L2_UDF_CH*n* Represents whether L2 TX FIFO underflows.
0: No underflow
1: Underflow
(RO)

AXI_DMA_OUTFIFO_L2_OVF_CH*n* Represents whether L2 TX FIFO overflows.
0: No overflow
1: Overflow
(RO)

AXI_DMA_OUT_REMAIN_UNDER_1B_CH*n* Reserved.(RO)

AXI_DMA_OUT_REMAIN_UNDER_2B_CH*n* Reserved.(RO)

AXI_DMA_OUT_REMAIN_UNDER_3B_CH*n* Reserved.(RO)

AXI_DMA_OUT_REMAIN_UNDER_4B_CH*n* Reserved.(RO)

AXI_DMA_OUT_REMAIN_UNDER_5B_CH*n* Reserved.(RO)

AXI_DMA_OUT_REMAIN_UNDER_6B_CH*n* Reserved.(RO)

AXI_DMA_OUT_REMAIN_UNDER_7B_CH*n* Reserved.(RO)

AXI_DMA_OUT_REMAIN_UNDER_8B_CH*n* Reserved.(RO)

Register 6.120. AXI_DMA_OUT_STATE_CH*n*_REG (*n*: 0-2) (0x0160+0x68**n*)

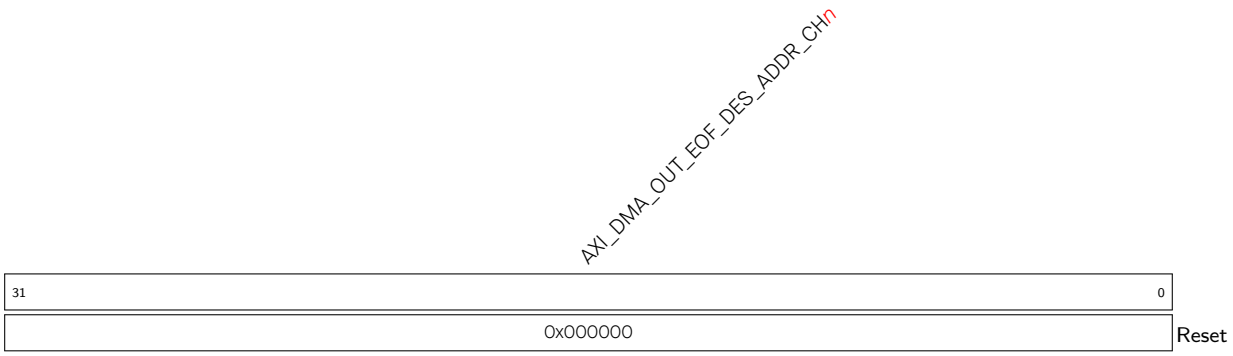


AXI_DMA_OUTLINK_DSCR_ADDR_CH*n* Represents the lower 18 bits of the next transmit descriptor address that is pre-read (but not processed yet). If the current transmit descriptor is the last descriptor, then this field represents the address of the current transmit descriptor. (RO)

AXI_DMA_OUT_DSCR_STATE_CH*n* Reserved. (RO)

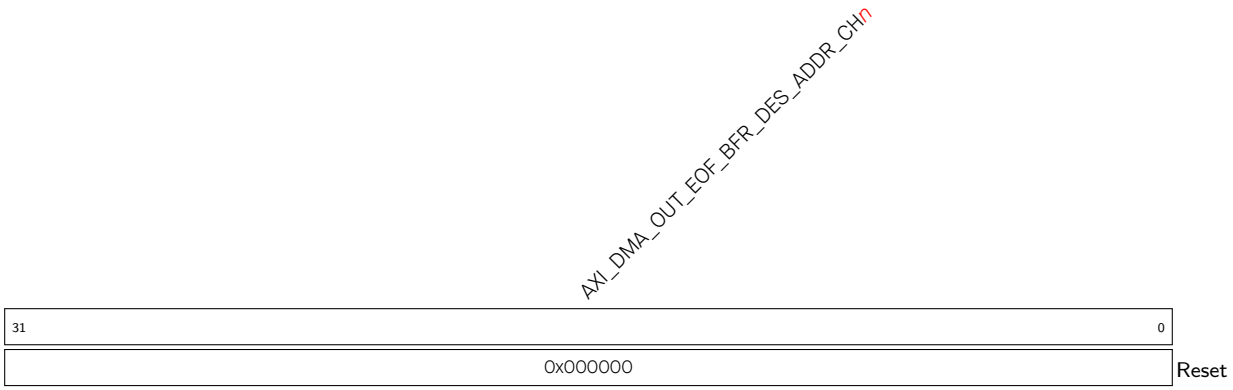
AXI_DMA_OUT_STATE_CH*n* Reserved. (RO)

Register 6.121. AXI_DMA_OUT_EOF_DES_ADDR_CH*n*_REG (*n*: 0-2) (0x0164+0x68**n*)



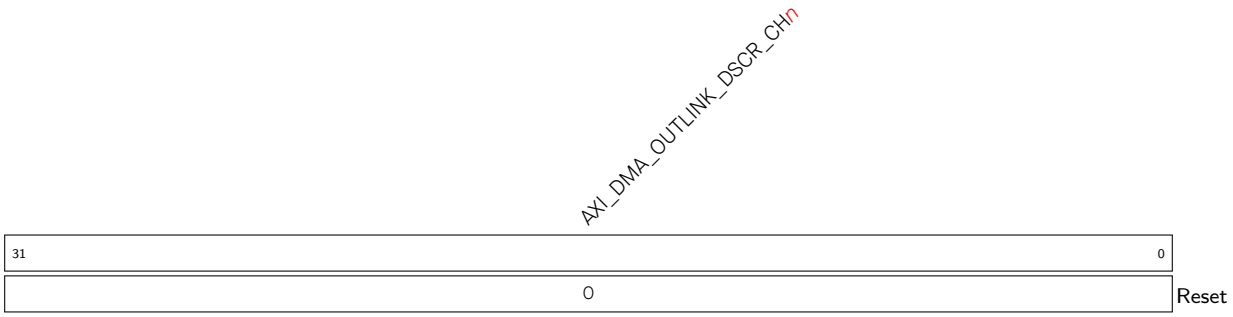
AXI_DMA_OUT_EOF_DES_ADDR_CH*n* Represents the address of the transmit descriptor when the EOF bit in this descriptor is 1. (RO)

Register 6.122. AXI_DMA_OUT_EOF_BFR_DES_ADDR_CH*n*_REG (*n*: 0-2) (0x0168+0x68**n*)



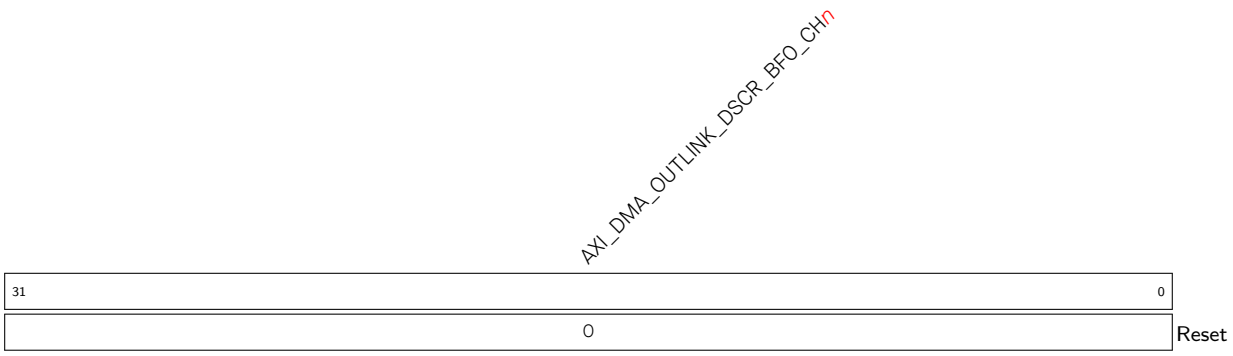
AXI_DMA_OUT_EOF_BFR_DES_ADDR_CH*n* Represents the address of the transmit descriptor before the last transmit descriptor. (RO)

Register 6.123. AXI_DMA_OUT_DSCR_CH*n*_REG (*n*: 0-2) (0x016C+0x68**n*)



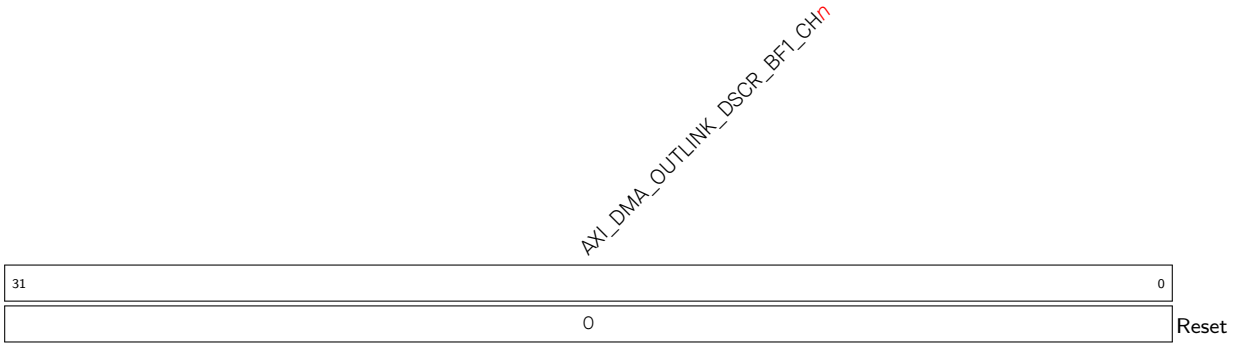
AXI_DMA_OUTLINK_DSCR_CH*n* Represents the address of the next transmit descriptor y+1 pointed by the current transmit descriptor that is pre-read. (RO)

Register 6.124. AXI_DMA_OUT_DSCR_BF0_CH*n*_REG (*n*: 0-2) (0x0170+0x68**n*)



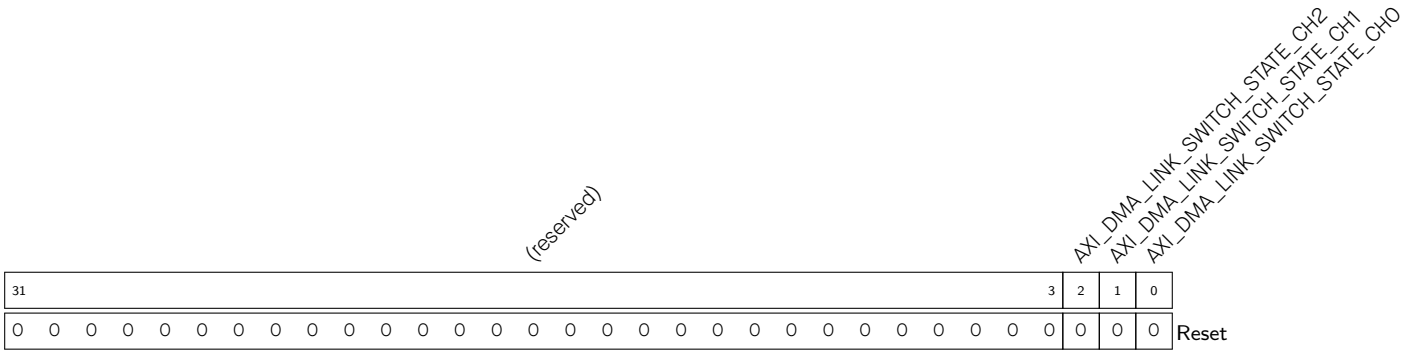
AXI_DMA_OUTLINK_DSCR_BF0_CH*n* Represents the address of the current transmit descriptor y that is pre-read. (RO)

Register 6.125. AXI_DMA_OUT_DSCR_BF1_CH*n*_REG (*n*: 0-2) (0x0174+0x68n*)**



AXI_DMA_OUTLINK_DSCR_BF1_CH*n* Represents the address of the previous transmit descriptor y-1 that is pre-read. (RO)

Register 6.126. AXI_DMA_LINK_SWITCH_STATE_REG (0x02DC)



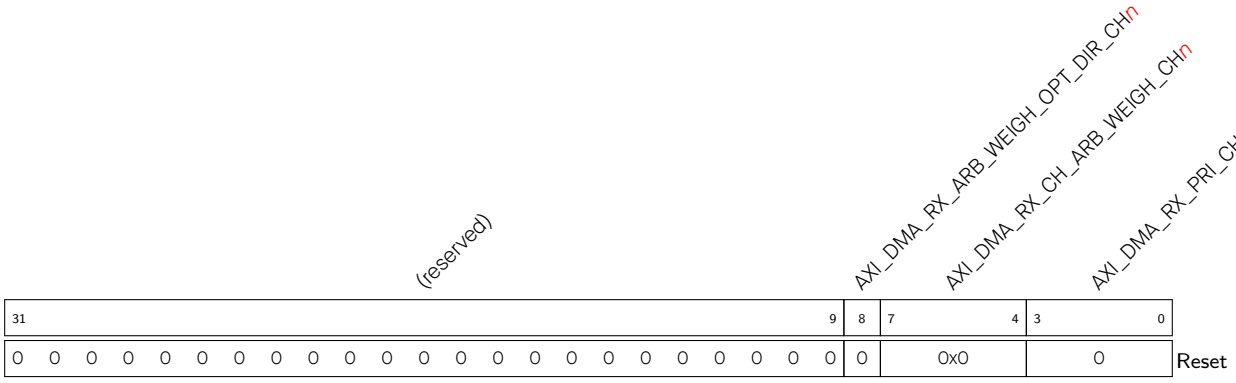
AXI_DMA_LINK_SWITCH_STATE_CH*n* Configures whether to switch the linked lists read by AXI DMA.

0: Not switch

1: Switch

(R/W)

Register 6.127. AXI_DMA_IN_PRI_CH*n*_REG (*n*: 0-2) (0x0040+0x68n*)**



AXI_DMA_RX_PRI_CH*n* Configures the priority of RX channel *n*. The larger the value, the higher the priority.

Value range: 0 ~ 5. (R/W)

AXI_DMA_RX_CH_ARB_WEIGH_CH*n* Configures the weight (i.e the number of tokens) of RX channel *n*.

Value range: 0 ~ 15. (R/W)

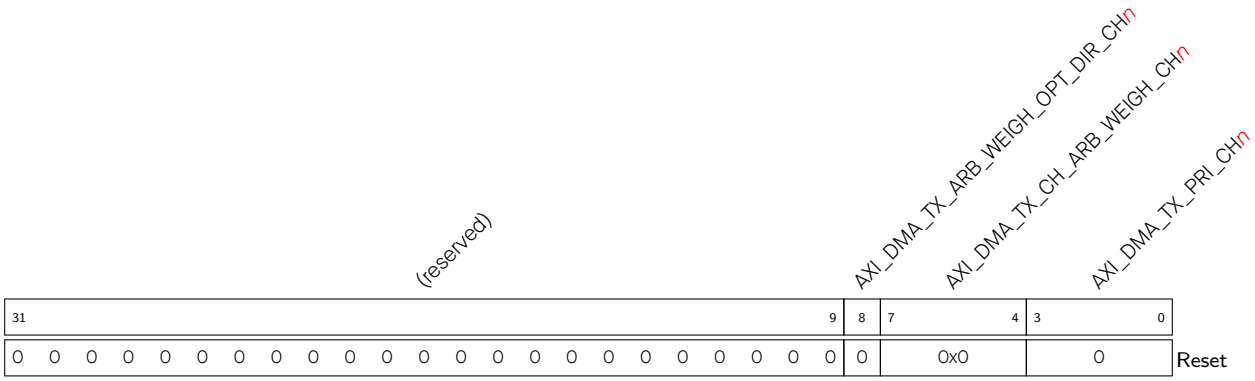
AXI_DMA_RX_ARB_WEIGH_OPT_DIR_CH*n* Configures whether to enable weight optimization for RX channel *n*.

0: Enable

1: Disable

(R/W)

Register 6.128. AXI_DMA_OUT_PRI_CH*n*_REG (*n*: 0-2) (0x0178+0x68**n*)



AXI_DMA_TX_PRI_CH*n* Configures the priority of RX channel *n*. The larger the value, the higher the priority.

Value range: 0 ~ 5. (R/W)

AXI_DMA_TX_CH_ARB_WEIGH_CH*n* Configures the weight (i.e the number of tokens) of TX channel *n*.

Value range: 0 ~ 15. (R/W)

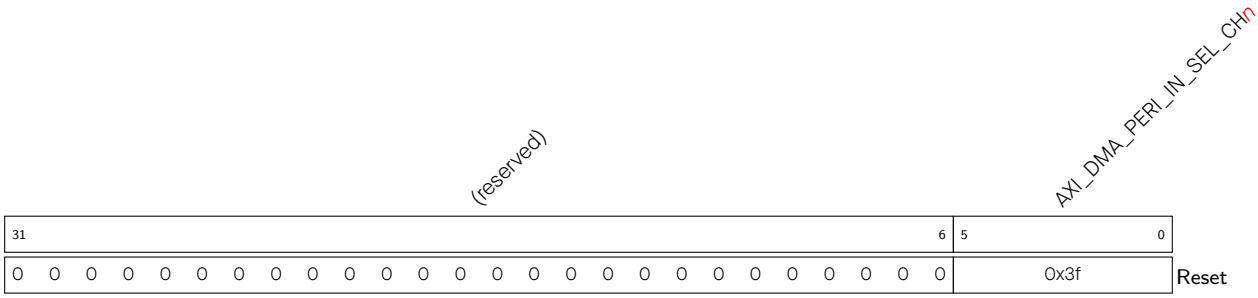
AXI_DMA_TX_ARB_WEIGH_OPT_DIR_CH*n* Configures whether to enable weight optimization for RX channel *n*.

0: Enable

1: Disable

(R/W)

Register 6.129. AXI_DMA_IN_PERI_SEL_CH*n*_REG (*n*: 0-2) (0x0044+0x68**n*)



AXI_DMA_PERI_IN_SEL_CH*n* Configures the peripheral connected to RX channel *n*.

0: LCD_CAM

1: SPI2

2: SPI3

3: PARLIO

4: AES

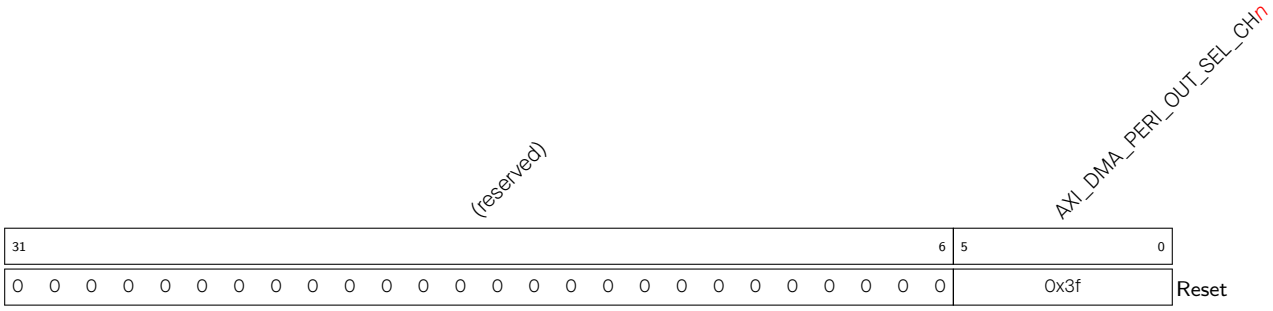
5: SHA

6 ~ 15: Dummy-6 ~ Dummy-15

16 ~ 63: Invalid

(R/W)

Register 6.130. AXI_DMA_OUT_PERI_SEL_CH*n*_REG (*n*: 0-2) (0x017C+0x68n*)**



AXI_DMA_PERI_OUT_SEL_CH*n* Configures the peripheral connected to TX channel *n*.

0: LCD_CAM

1: SPI2

2: SPI3

3: PARLIO

4: AES

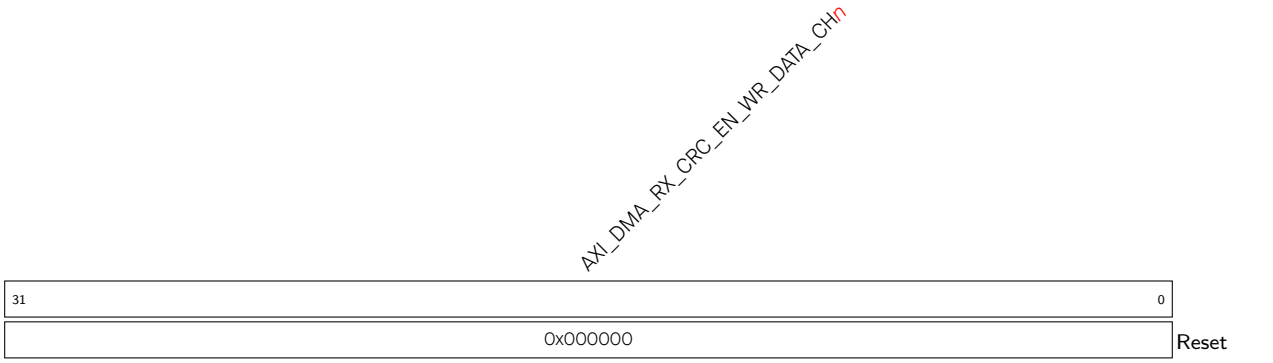
5: SHA

6 ~ 15: Dummy-6 ~ Dummy-15

16 ~ 63: Invalid

(R/W)

Register 6.131. AXI_DMA_RX_CRC_EN_WR_DATA_CH*n*_REG (*n*: 0-2) (0x0058+0x68n*)**



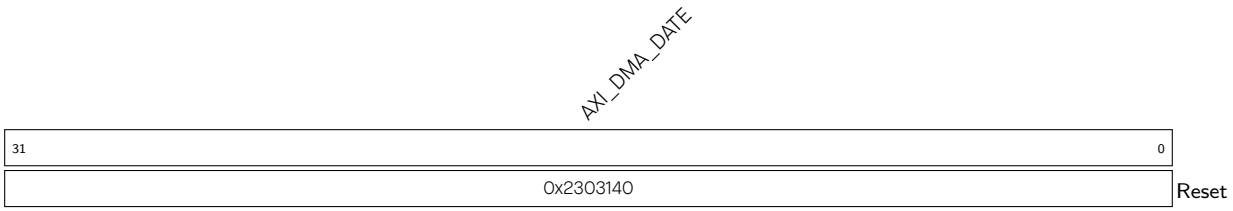
AXI_DMA_RX_CRC_EN_WR_DATA_CH*n* Configures whether to include each bit of the intermediate result in the CRC calculation matrix for RX channel *n*.

0: Not include

1: Include

(R/W)

Register 6.132. AXI_DMA_DATE_REG (0x02D8)



AXI_DMA_DATE Version control register. (R/W)

Chapter 7

VDMA Controller (VDMA)

7.1 Overview

DMA (Direct Memory Access) enables direct access to system memory or peripherals without CPU involvement. The VDMA controller on ESP32-P4 is a general-purpose DMA that performs high-speed data transfer from memory to memory, from memory to peripheral, and from peripheral to memory. The VDMA complies with the AXI3 protocol and includes two AXI master interfaces. This design allows users to select between the two interfaces for data transfer dynamically.

7.2 Terminology

The following terms are defined in the context of the VDMA controller on ESP32-P4:

- **AXI**: Advanced eXtensible Interface, a part of AMBA 3 (Advanced Microcontroller Bus Architecture) specification, designed to facilitate high bandwidth and low latency communication between memory and peripheral, and between memory and memory.
- **Peripheral**: A system component that has a handshake interface with VDMA, as listed in Table 7.4-1.
- **Memory**: A system component that is always ready for DMA transfer, which means there is no handshaking interface between VDMA and memory. The memory that supports VDMA data transfer includes HP L2MEM, external flash, and external RAM.
- **Source**: Source memory or peripheral from which VDMA reads data via the AXI interface. VDMA then stores data in channel FIFO.
- **Source transfer**: Transfer from the source to VDMA.
- **Destination**: Destination peripheral or memory to which VDMA writes the data stored in channel FIFO.
- **Destination transfer**: Transfer from VDMA to the destination.
- **Channel**: Data path between source and destination.
- **Master interface**: Refers to the two AXI master interfaces of VDMA.
- **Slave interface**: The APB3 slave interface for programming registers.
- **Handshaking interface**: The handshaking interface performs a handshake between VDMA and the source or destination peripheral. The handshaking interface requests, acknowledges, and controls an VDMA transfer.
- **Flow controller**: The flow controller defines the length of a DMA block transfer and terminates the transfer. The flow controller can be VDMA, a source peripheral, or a destination peripheral, but it cannot be memory.

7.3 Features

VDMA supports the following features:

- Four channels for unidirectional data transfer from source to destination
- Two AXI master interfaces
- Handshake with MIPI DSI (Display Serial Interface) and ISP (Image Signal Processor)
- Memory-to-memory, ISP-to-memory, and MIPI DSI-to-memory transfer types
- Multiple levels of DMA transfer hierarchy
- Configurable transfer type, transfer length, and transfer size for each channel
- Single-block transfer
- Multi-block transfer based on contiguous address, automatic reloading register configuration, shadow registers, and linked lists
- Independent configuration of multi-block transfer type for source transfer and destination transfer
- Channel disabling without data loss
- Channel suspension, resume, and abortion
- Configurable priorities among arbitration channels
- Flow control using VDMA or peripherals
- Programmable mapping between peripherals and channels

7.4 Architectural Overview

Figure 7.4-1 shows the architecture of VDMA and its connection with other system components.

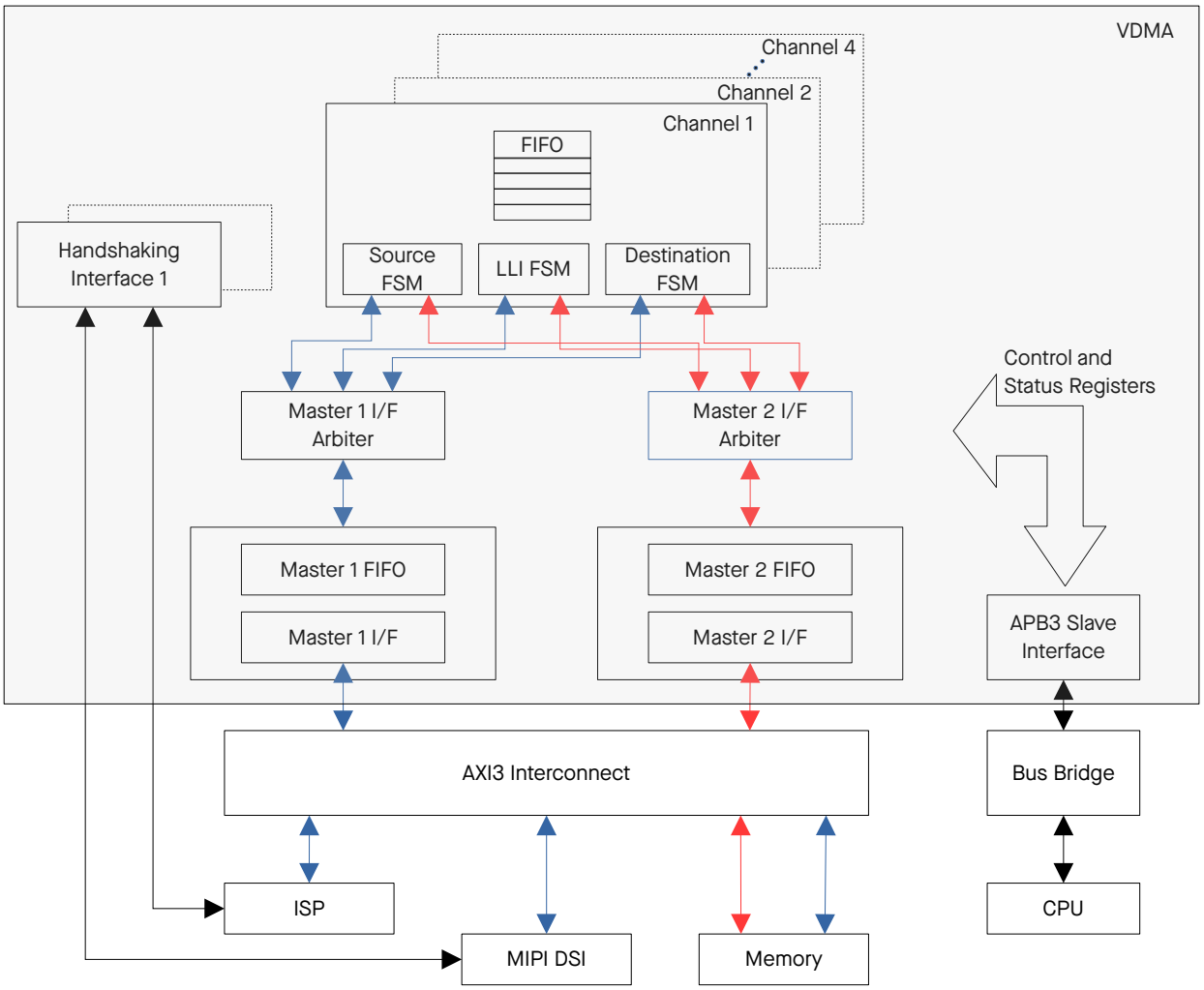


Figure 7.4-1. VDMA Architecture

It can be seen from the figure that:

- The CPU accesses VDMA registers via the APB3 slave interface.
- VDMA has four channels (Channel 1 to Channel 4). Each channel has a data FIFO with a depth of 64 and a width of 64 bits.
- VDMA has two AXI master interfaces (Master 1 and Master 2).
  - Master 1 can access ISP, MIPI DSI, and memory, as blue arrows indicate.
  - Master 2 can only access memory, as red arrows indicate.
  - Master 1 and Master 2 each have an arbiter that arbitrates read and write requests on all channels.
- VDMA has three hardware handshaking interfaces. Table 7.4-1 shows the mapping between the handshaking interfaces and peripherals.

Table 7.4-1. Hardware Handshaking Interface and Peripherals Mapping Table

Handshaking Interface Number	Peripheral
0	MIPI DSI

1	ISP (ISP-to-memory)
2	ISP (memory-to-ISP)

For more information about ISP and MIPI DSI, please refer to Chapter 38 *Image Signal Processor (ISP)* and Chapter 43 *MIPI DSI*.

7.5 Functional Description

This section provides a detailed description of the key features and operations of the module.

7.5.1 Transfer Hierarchy

Transfers are organized into a maximum of four levels: DMA transfer level, block transfer level, transaction level, and AXI transfer level.

Figure 7.5-1 shows four transfer levels between VDMA and peripherals.

Figure 7.5-2 shows three transfer levels between VDMA and memory. There is no transaction level because memory is assumed to be always ready for data transfer.

The following descriptions of DMA, block, and AXI transfers apply to the transfer between VDMA and memory, VDMA and peripherals. **The description of the transaction level applies only to transfers between VDMA and peripherals.**

- **DMA transfer**: Software defines the number of blocks in a DMA transfer. Once a DMA transfer has finished, VDMA disables the channel and generates an interrupt. The channel can then be reprogrammed for a new DMA transfer. A DMA transfer can consist of either a single-block or a multi-block transfer.
- **Block transfer**: A block transfer moves a specified data block through the VDMA. The flow controller controls the block length. There are two types of block transfer: single-block transfer and multi-block transfer. As shown in Figure 7.5-1, a block transfer may consist of multiple transactions.
- **Transaction**: A peripheral request may trigger either a single or burst transaction, corresponding to an AXI single transfer or multiple AXI burst transfers, respectively.
  - Single transaction: The length of a single transaction is always 1, in the unit of DMAC_CH*n*_SRC_TR_WIDTH.
  - Burst transaction: The length of a burst transaction is programmed into the VDMA through DMAC_CH*n*_SRC_MSIZE for the source or DMAC_CH*n*_DST_MSIZE for the destination. The burst length typically relates to the FIFO sizes in the source or destination peripherals. For more information, please refer to the chapter on each peripheral.
- **AXI transfer**: Refers to the AXI protocol transfer, which is divided into AXI single transfer and AXI burst transfer.

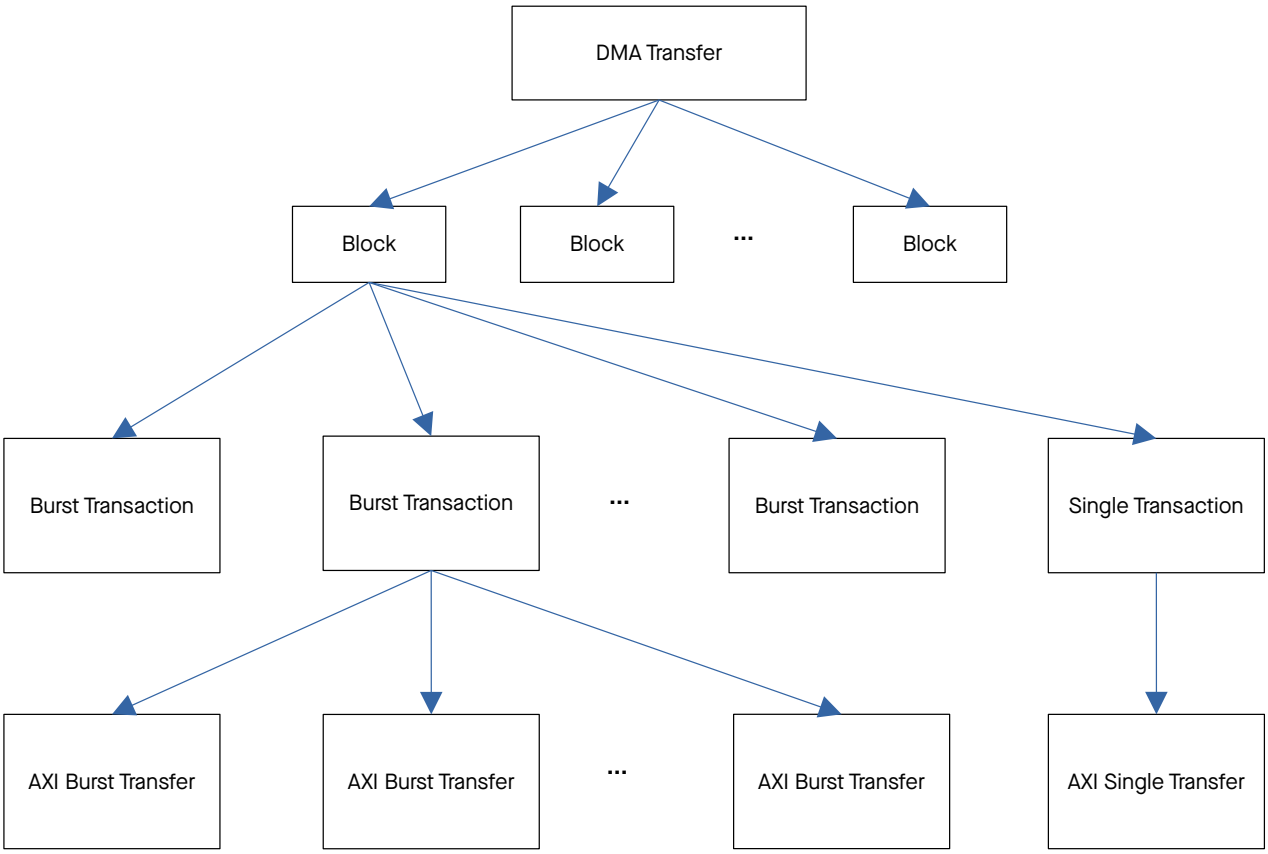


Figure 7.5-1. VDMA Transfer Hierarchy for Peripherals

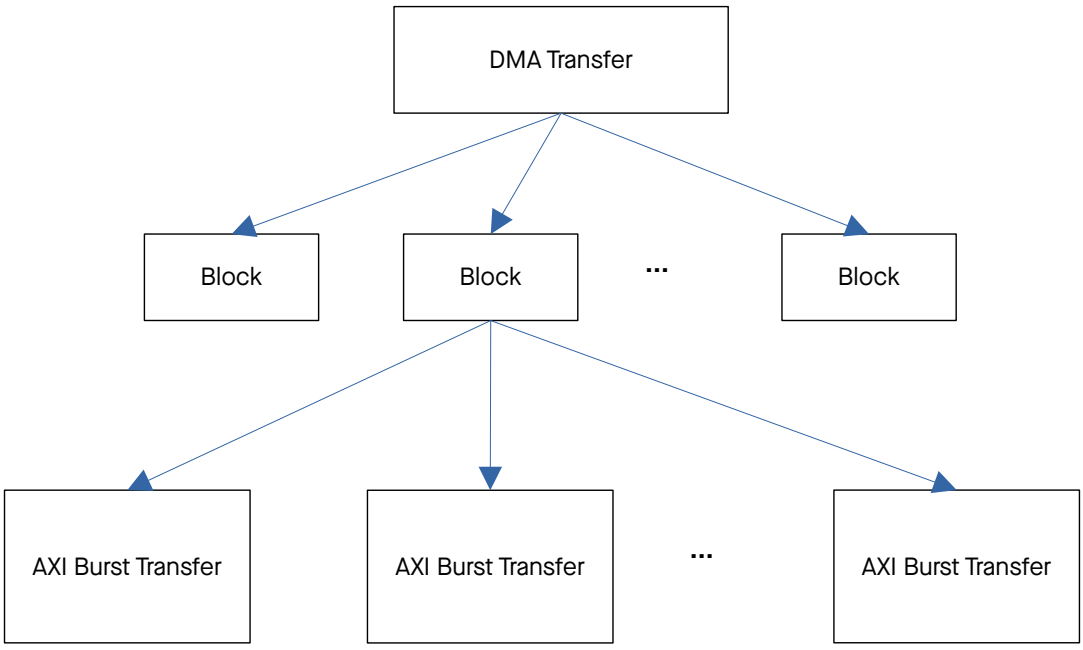


Figure 7.5-2. VDMA Transfer Hierarchy for Memory

7.5.2 Arbitration Scheme

The arbitration mechanism arbitrates read and write requests for all channels.

VDMA uses a priority and fair-among-equals arbitration mechanism. The priority for each channel is defined in the channel configuration register DMAC_CH*n*_CFG1_REG.

A priority of 3 is the highest, while 0 is the lowest. Multiple channels can share the same priority level.

The arbitration mechanism works as follows:

- The arbitration mechanism determines which request client to grant the authorization signal based on the priority of the request. The request with the highest priority will be granted access to the channel.
- When two or more requests have the same priority, request clients will gain access randomly.

7.5.2.1 Read Arbiter

The read arbiter arbitrates read requests, including:

- VDMA's read requests to obtain data or status from the source
- VDMA's read requests to obtain status from the destination
- VDMA's read requests to obtain LLI descriptors

The block diagram of the read arbiter is shown in Figure 7.5-3:

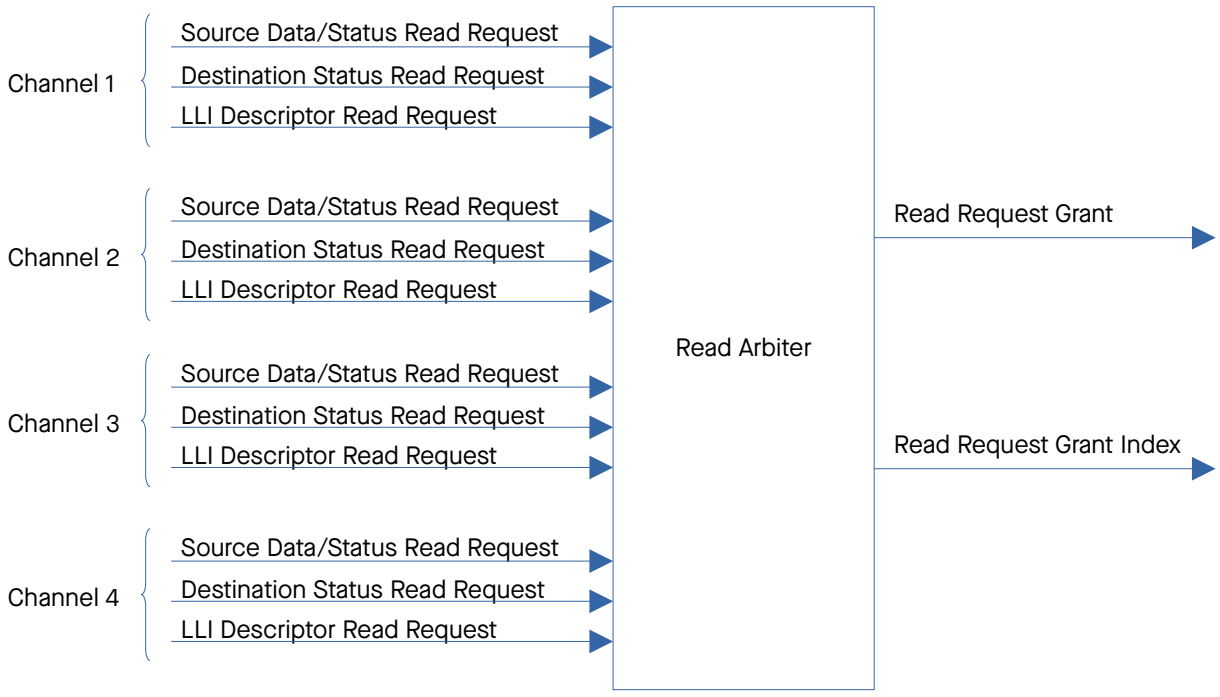


Figure 7.5-3. VDMA Read Arbiter

Read requests to obtain linked list descriptors will be granted the highest priority, regardless of the channels' priority settings.

7.5.2.2 Write Arbiter

The write arbiter arbitrates write requests, including:

- Destination Data Write Request
- LLI Descriptor Write-Back Request

The block diagram of the write arbiter is shown in Figure 7.5-4:

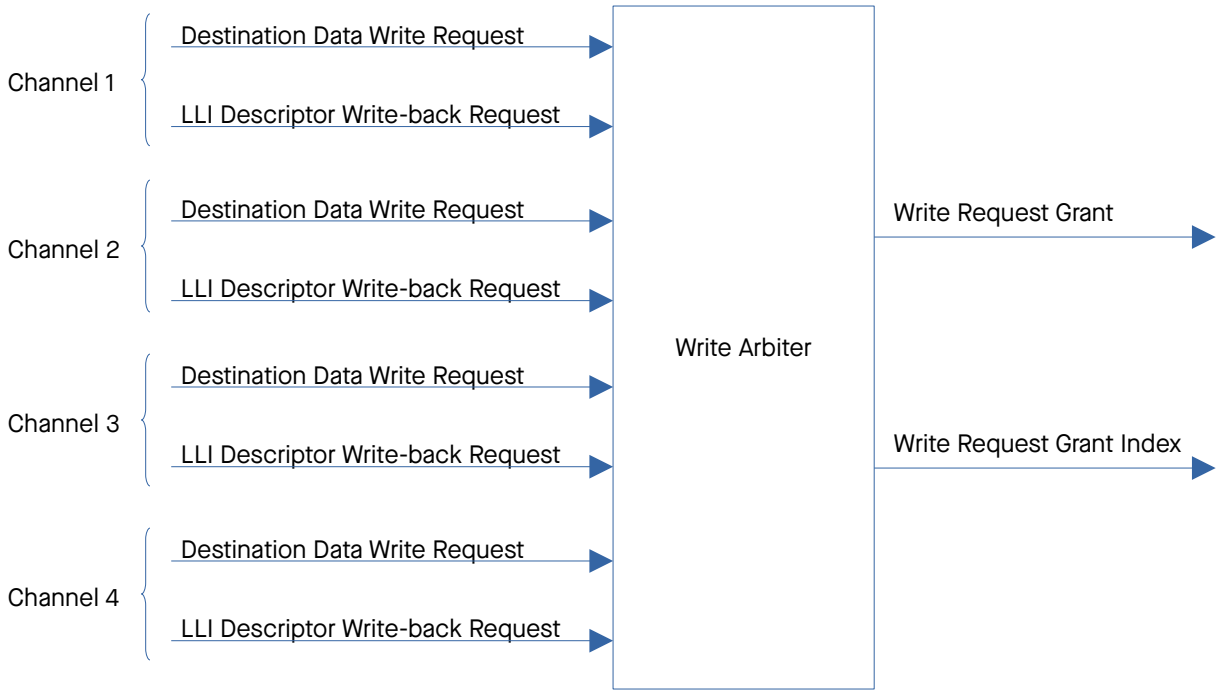


Figure 7.5-4. VDMA Write Arbiter

Within the same channel, Destination Data Write Request has higher priority than LLI Descriptor Write-Back Request. If different channels have the same priority, then Data Write Requests have the highest priority.

7.5.3 Handshaking Interface

A peripheral uses a handshaking interface to inform VDMA that it is ready to transmit or receive data via AXI bus. The operation of the handshaking interface depends on whether the flow controller is the peripheral or VDMA.

VDMA performs a single transaction or a burst transaction per handshake. In some cases, a block transfer cannot be completed using only burst transactions. This situation typically occurs when the block size is not a multiple of the burst transaction length. In this case, the block transfer uses burst transactions until the remaining size of the block is less than the amount of data in the burst transaction. At this point, a single transaction is used to complete the block transfer.

A source's burst transaction size is determined by DMAC_CH*n*_SRC_MSIZE. A destination's burst transaction size is determined by DMAC_CH*n*_DST_MSIZE. The transaction size also corresponds to the burst transaction size used for handshaking. The value of DMAC_CH*n*_SRC_MSIZE or DMAC_CH*n*_DST_MSIZE should match the peripheral configuration. For detailed information, please refer to the chapter on related peripherals.

VDMA has three hardware handshaking interfaces with MIPI DSI, ISP (ISP-to-memory), and ISP (memory-to-ISP). For the mapping relationship please see Table 7.4-1.

7.5.4 Transfer Control

Transfer control logic facilitates the data transfer from a source to a destination. Data from the source is temporarily stored in the channel FIFO before being sent to the destination. If source and destination peripherals use different transfer sizes, VDMA will pack and unpack the data to fit the FIFO configuration.

For a specific DMA transfer, the transfer type and flow control configurations are determined by DMAC_CH*n*_TT_FC.

7.5.4.1 Single-Block Transfer

If a DMA transfer consists of a single block, the software can set DMAC_CH*n*_SRC_MULTBLK_TYPE and DMAC_CH*n*_DST_MULTBLK_TYPE to 0 to enable contiguous-address-based single-block transfer. In this case, VDMA disables the channel once the block transfer of size DMAC_CH*n*_BLOCK_TS is completed.

> **Note:**
> Single-block transfer is a special case of multi-block transfer, and there is no strict distinction between the two. In this chapter, single-block transfer refers to the transfer consisting of only one block, while multi-block transfer refers to the transfer consisting of at least two blocks.

7.5.4.2 Multi-Block Transfer

If a DMA transfer consists of multiple blocks, the software can configure DMAC_CH*n*_SRC_MULTBLK_TYPE and DMAC_CH*n*_DST_MULTBLK_TYPE to choose a multi-block transfer type. There are four types of multi-block transfers, depending on how the transfer control registers (DMAC_CH*n*_SARO_REG, DMAC_CH*n*_DARO_REG, DMAC_CH*n*_BLOCK_TS, DMAC_CH*n*_CTLO/1_REG) are updated:

1. **Contiguous address**
2. **Auto reloading**: Register values reload from initial values
3. **Shadow register**: Register values load from shadow registers
4. **Linked list** Register values load from the next linked list (LLI)

Table 7.5-1 lists all the cases of register update methods for multi-block transfer.

Table 7.5-1. Register Update Methods for Multi-Block Transfer

DMAC_CH*n*_CFG0_REG		Registers Update Method				Comment
DMAC_CH*n*_SRC_MULTBLK_TYPE	DMAC_CH*n*_DST_MULTBLK_TYPE	DMAC_CH*n*_SARO_REG	DMAC_CH*n*_DARO_REG	DMAC_CH*n*_CTLO_REG /CTL1_REG	DMAC_CH*n*_BLOCK_TS	
00	00	Contiguous address	Contiguous address	No update	No update	Single block or last block of multi-block
00	01	Contiguous address	Reload from initial value	Reload from initial value	Reload from initial value	-
00	10	Contiguous address	Load from shadow register	Load from shadow register	Load from shadow register	-
01	00	Reload from initial value	Contiguous address	Reload from initial value	Reload from initial value	-
01	01	Reload from initial value	Reload from initial value	Reload from initial value	Reload from initial value	-
01	10	Reload from initial value	Load from shadow register	Load from shadow register	Load from shadow register	-
10	00	Load from shadow register	Contiguous address	Load from shadow register	Load from shadow register	-
10	01	Load from shadow register	Reload from initial value	Load from shadow register	Load from shadow register	-
10	10	Load from shadow register	Load from shadow register	Load from shadow register	Load from shadow register	-
00	11	Contiguous address	Load from next LLI	Load from next LLI	Load from next LLI	-
01	11	Reload from initial value	Load from next LLI	Load from next LLI	Load from next LLI	-
11	00	Load from next LLI	Contiguous address	Load from next LLI	Load from next LLI	-
11	01	Load from next LLI	Reload from initial value	Load from next LLI	Load from next LLI	-
11	11	Load from next LLI	Load from next LLI	Load from next LLI	Load from next LLI	-
10	11	-	-	-	-	Invalid. Interrupt* is generated.
11	10	-	-	-	-	Invalid. Interrupt* is generated.

*Note: The interrupt is DMAC_CH*n*_SLVIF_MULTIBLKTYPE_ERR_INT.

Contiguous Address

In this scenario, the address for each subsequent block continues from the end of the preceding block. To achieve this contiguity, configure DMAC_CH*n*_SRC_MULTBLK_TYPE or DMAC_CH*n*_DST_MULTBLK_TYPE to 0 for the source address or destination address to remain continuous across the blocks.

In the case of multi-block transfer, the values of DMAC_CH*n*_SRC_MULTBLK_TYPE and DMAC_CH*n*_DST_MULTBLK_TYPE cannot both be selected as contiguous addresses, meaning they cannot both be configured to 0 at the same time. If both are configured to 0 during a multi-block transfer, then the current block will be considered as the last block.

During a multi-block transfer, if DMAC_CH*n*_SARO_REG and DMAC_CH*n*_DARO_REG need to have contiguous addresses between blocks, you can indirectly achieve this functionality using linked-list or shadow-register-based multi-block transfer.

Note:
If you have enabled contiguous-address-based multi-block transfer (i.e., DMAC_CH*n*_SRC_MULTBLK_TYPE or DMAC_CH*n*_DST_MULTBLK_TYPE is 0), ensure there are at least two blocks in the DMA transfer. Otherwise, it will result in unpredictable behavior.

Auto Reloading

In this scenario, DMAC_CH*n*_SARO_REG and DMAC_CH*n*_DARO_REG are reloaded with their initial values at the end of each block. VDMA does not proceed to the next block transfer until software clears the corresponding channel's block transfer complete interrupt by writing 1 to DMAC_CH*n*_CLEAR_SRC_TRANSCOMP_INTSTAT and DMAC_CH*n*_CLEAR_DST_TRANSCOMP_INTSTAT.

Note:
If you have enabled auto-reloading-based multi-block transfer, please ensure there are at least two blocks in the DMA transfer. Otherwise, it will result in unpredictable behavior.

Shadow Register

In this scenario, DMAC_CH*n*_SARO_REG, DMAC_CH*n*_DARO_REG, DMAC_CH*n*_BLOCK_TS, DMAC_CH*n*_CTLO_REG, and DMAC_CH*n*_CTL1_REG synchronize with their corresponding shadow registers when each block finishes. The values of the shadow registers are then used for the subsequent block.

Software writes the block configurations to the corresponding shadow registers. VDMA copies the shadow register contents to DMAC_CH*n*_SARO_REG, DMAC_CH*n*_DARO_REG, DMAC_CH*n*_BLOCK_TS, DMAC_CH*n*_CTLO_REG, and DMAC_CH*n*_CTL1_REG before starting the subsequent block transfer.

Read operations to the DMAC_CH*n*_SARO_REG, DMAC_CH*n*_DARO_REG, DMAC_CH*n*_BLOCK_TS, DMAC_CH*n*_CTLO_REG, and DMAC_CH*n*_CTL1_REG always return the data corresponding to the current block transfer, not the shadow register contents that correspond to the next block.

DMAC_CH*n*_SHADOWREG_OR_LLI_VALID indicates whether the shadow register contents are valid (1) or not (0). If this bit reads as 0 during a shadow register fetch phase, VDMA discards the shadow register contents and generates a DMAC_CH*n*_SHADOWREG_OR_LLI_INVALID_ERR_INT interrupt. VDMA waits until the software

writes 1 to DMAC_CH*n*_BLK_TFR_RESUMEREQ, indicating that the shadow registers are available. After this, VDMA attempts to read the shadow registers again and continues with the next block transfer.

Note:
In a shadow-register-based multi-block transfer, the values in DMAC_CH*n*_SARO_REG, DMAC_CH*n*_DARO_REG, DMAC_CH*n*_BLOCK_TS, DMAC_CH*n*_CTLO_REG, and DMAC_CH*n*_CTL1_REG correspond to the current transfer configurations. When software writes to these registers, it is actually writing to the corresponding shadow registers.

Linked List

In this scenario, VDMA retrieves a block descriptor for a specific block from memory to update DMAC_CH*n*_SARO_REG, DMAC_CH*n*_DARO_REG, DMAC_CH*n*_BLOCK_TS, DMAC_CH*n*_CTLO_REG, and DMAC_CH*n*_CTL1_REG, and then initiates the block transfer. This process is called LLI (Linked List Item) update. The block chaining feature of VDMA uses the linked list pointer register DMAC_CH*n*_LOCO to store the address of the next linked list item in memory, thereby achieving seamless continuity of data transfer.

Figure 7.5-5 shows the VDMA linked list item:

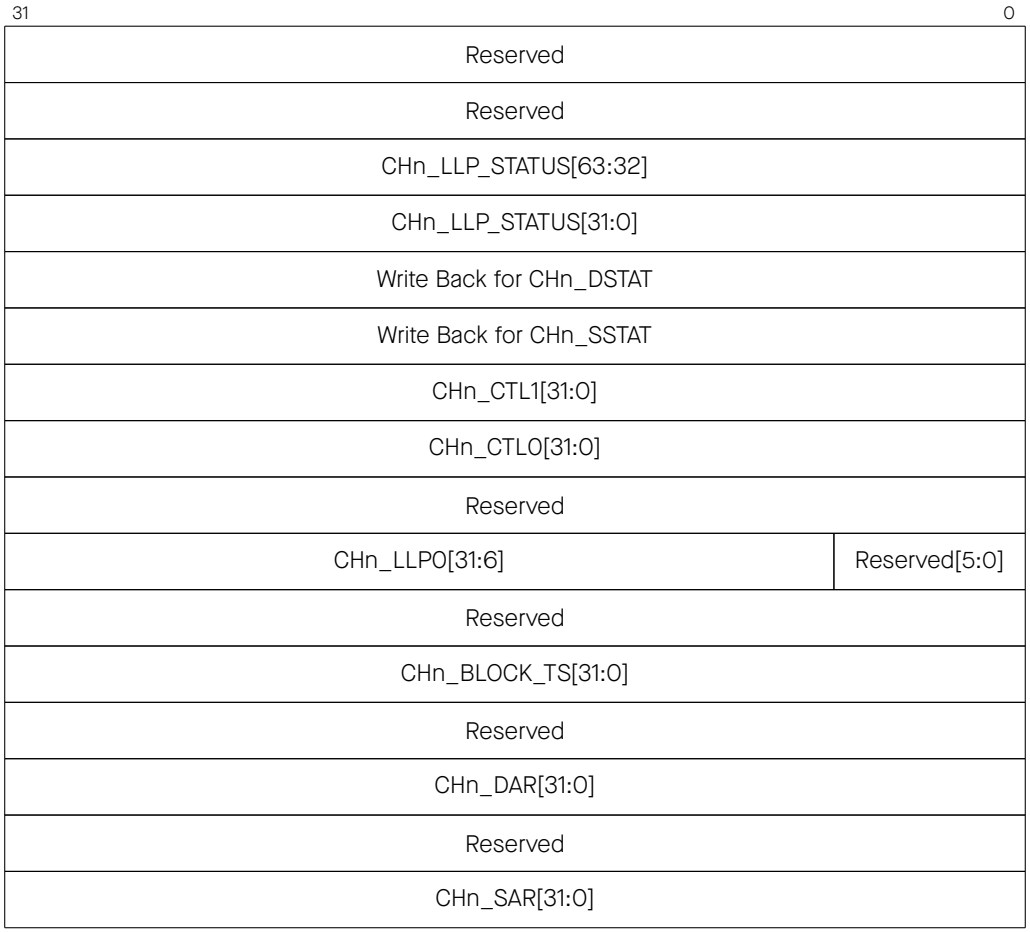


Figure 7.5-5. VDMA Linked List Item (Descriptor)

VDMA can dynamically extend linked lists, which means you do not need to pre-create the entire linked list in system memory. To enable this feature, set DMAC_CH*n*_SHADOWREG_OR_LLI_VALID to 1. If the linked list item is the last one, then also configure DMAC_CH*n*_SHADOWREG_OR_LLI_LAST to 1.

DMAC_CH*n*_SHADOWREG_OR_LLI_VALID indicates whether the linked list item fetched from memory is valid (1) or not (0). If the LLI is invalid, VDMA discards the LLI and generates a DMAC_CH*n*_SHADOWREG_OR_LLI_INVALID_ERR_INT interrupt. This interrupt causes VDMA to halt the corresponding channel and wait until the software writes 1 to DMAC_CH*n*_BLK_TFR_RESUMEREQ to indicate that the LLI is available. After this, VDMA will attempt to read the LLI again.

Note:
For pre-fetched LLIs, if DMAC_CH*n*_SHADOWREG_OR_LLI_VALID is 0, then the DMAC_CH*n*_SHADOWREG_OR_LLI_INVALID_ERR_INTSTAT interrupt will not be generated. VDMA re-attempts to fetch the LLI again after completing the current block transfer and generates a DMAC_CH*n*_SHADOWREG_OR_LLI_INVALID_ERR_INTSTAT interrupt only if DMAC_CH*n*_SHADOWREG_OR_LLI_VALID still reads as 0.

The following fields in LLI are prepared by software:

- CH*n*_SAR: The field definition is the same as DMAC_CH*n*_SARO_REG
- CH*n*_DAR: The field definition is the same as DMAC_CH*n*_DARO_REG
- CH*n*_BLOCK_TS: The field definition is the same as DMAC_CH*n*_BLOCK_TS
- CH*n*_LLPO[31:6]: The field definition is the same as DMAC_CH*n*_LLPO_REG[31:6]
- CH*n*_CTLO: The field definition is the same as DMAC_CH*n*_CTLO_REG
- CH*n*_CTL1: The field definition is the same as DMAC_CH*n*_CTL1_REG

If the status write-back option is enabled, VDMA writes back CH*n*_CTLO, CH*n*_CTL1, CH*n*_LLP_STATUS, CH*n*_SSTAT, and CH*n*_DSTAT information to the location defined for this field, which is from address [CH*n*_LLP] + 0x20 to [CH*n*_LLP] + 0x34. The write-back of CH*n*_SSTAT and CH*n*_DSTAT can be independently enabled or disabled by programming DMAC_CH*n*_SRC_STAT_EN and DMAC_CH*n*_DST_STAT_EN.

Note:
- If CH*n*_SSTAT and CH*n*_DSTAT write-back is not enabled, do not use PSRAM to store LLI.
- DMAC_CH*n*_SHADOWREG_OR_LLI_VALID will be 0 after the LLI write-back operation.

Figure 7.5-6 shows CH*n*_LLP_STATUS write-back field of LLI:



Note: * denotes Status Indication (CHn_IntStatus[1:0])

Figure 7.5-6. CH*n*_LLP_STATUS Write-Back Field of LLI

CH*n*_LLP_STATUS[63] and CH*n*_LLP_STATUS[62] indicate DMA_TFR_DONE and BLOCK_TFR_DONE status respectively. These two fields are the last to be updated during LLI write-back. Software should ensure that BLOCK_TFR_DONE bit is set to 1 before using CH*n*_SSTAT and CH*n*_DSTAT information. DMA_TFR_DONE and BLOCK_TFR_DONE bits will be set to 1 after transferring the last block.

CH*n*_STATUS[21:0] corresponds to DMAC_CH*n*_CMPLTD_BLK_TFR_SIZE, and CH*n*_STATUS[46:32] corresponds to DMAC_CH*n*_DATA_LEFT_IN_FIFO[14:0].

Suspension of Transfers Between Blocks

At the end of every block transfer, a block transfer completion interrupt DMAC_CH*n*_BLOCK_TFR_DONE_INT is generated if:

- Global interrupt is enabled (DMAC_INT_EN = 1)
- The channel block transfer completion interrupt is enabled (DMAC_CH*n*_ENABLE_BLOCK_TFR_DONE_INTSTAT = 1, DMAC_CH*n*_ENABLE_BLOCK_TFR_DONE_INTSIGNAL = 1, DMAC_CH*n*_IOC_BLKTFR = 1)

For contiguous-address and auto-reloading-based multi-block transfers (neither source nor destination peripheral uses shadow-register or linked-list-based multi-block transfers), the DMA transfer automatically stalls after the DMAC_CH*n*_BLOCK_TFR_DONE_INT interrupt is generated. VDMA does not proceed to the next block transfer until software writes 1 to DMAC_CH*n*_CLEAR_BLOCK_TFR_DONE_INTSTAT to clear the interrupt.

Channel suspension between blocks ensures that the block transfer done ISR (Interrupt Service Routine) of the next-to-last block is serviced before the final block transfer starts. This ensures that the ISR has cleared DMAC_CH*n*_SRC_MULTBLK_TYPE and DMAC_CH*n*_DST_MULTBLK_TYPE before the final block transfer is completed.

End of Multi-Block Transfers

If either source or destination peripheral uses shadow-register or linked-list-based multi-block transfers, then DMAC_CH*n*_SHADOWREG_OR_LLI_LAST indicates whether the current block is the last in the transfer. If this bit is 1, VDMA recognizes that the current block is the final block in the transfer and completes the DMA transfer operation at the end of the current block transfer.

For contiguous-address and auto-reloading-based multi-block transfers (when neither source nor destination peripheral uses shadow-register or linked-list-based multi-block transfers), if the corresponding multi-block type selection fields DMAC_CH*n*_SRC_MULTBLK_TYPE and/or DMAC_CH*n*_DST_MULTBLK_TYPE are 0 at the end of a block transfer, VDMA understands that the previous block was the final block in the transfer and completes the DMA transfer operation.

7.5.5 Flow Controller

The flow controller determines the length of the DMA block transfer. The flow controller can be VDMA, the source peripheral, or the destination peripheral; memory cannot serve as a flow controller.

If the block length is known before enabling the channel, configure VDMA as the flow controller. If the block length is unknown before enabling the channel, use the source or destination peripheral as the flow controller to determine when to terminate the DMA block transfer.

7.5.6 Channel Suspend and Resume

Software can suspend and resume a channel during DMA transfer. To suspend a channel during DMA transfer:

1. Software writes 1 to DMAC_CH*n*_SUSP.
2. VDMA halts all transfers from the source, after completing all AXI transfers initiated from the source.
3. VDMA sets DMAC_CH*n*_CH_SRC_SUSPENDED_INT to 1 to indicate that source transfer is suspended.
4. VDMA transfers all the data in channel FIFO to the destination. If DMAC_CH*n*_SRC_TR_WIDTH < DMAC_CH*n*_DST_TR_WIDTH and DMAC_CH*n*_SUSP is 1, there may still be data in the channel FIFO, but not enough to form a single transfer of DMAC_CH*n*_DST_TR_WIDTH. The remaining data in the channel FIFO will be transferred to the destination if the channel is resumed.
5. VDMA sets DMAC_CH*n*_CH_SUSPENDED_INT to 1 to indicate that the channel is suspended.

After a channel suspend, software can resume the channel by setting DMAC_CH*n*_SUSP to 0. Then, VDMA resumes the DMA transfer from the point where it was suspended.

7.5.7 Channel Disable

Under normal operation, software enables a channel by writing 1 to DMAC_CH*n*_EN, and VDMA clears DMAC_CH*n*_EN after channel transfer completion. Software can disable a channel before a transfer completes.

7.5.7.1 Disabling a Suspended Channel Before Transfer Completion

To disable a suspended channel before transfer completion:

- Write 0 to DMAC_CH*n*_EN after DMAC_CH*n*_CH_SUSPENDED_INTSTAT is set to 1. If DMAC_CH*n*_SRC_TR_WIDTH < DMAC_CH*n*_DST_TR_WIDTH and DMAC_CH*n*_SUSP is 1, there may still be data in the channel FIFO, but not enough to form a single transfer of DMAC_CH*n*_DST_TR_WIDTH. In this case, once the channel is disabled, the remaining data in the channel FIFO is not transferred to the destination peripheral and is lost.
- VDMA sets DMAC_CH*n*_CH_DISABLED_INTSTAT to 1 to indicate that the channel is disabled and generates an interrupt.

7.5.7.2 Disabling a Non-suspended Channel Before Transfer Completion

To disable a non-suspended channel before transfer completion:

- Software writes 0 to DMAC_CH*n*_EN during DMA transfer.
- VDMA halts all transfers from the source, after completing all AXI transfers initiated on the source.
- VDMA transfers all the data in the channel FIFO to the destination. When DMAC_CH*n*_SRC_TR_WIDTH < DMAC_CH*n*_DST_TR_WIDTH and DMAC_CH*n*_EN is 0, there may still be data in the channel FIFO, but not enough to form a single transfer of DMAC_CH*n*_DST_TR_WIDTH. In this scenario, once the channel is disabled, the remaining data in the channel FIFO is not transferred to the destination peripheral and is lost.
- VDMA sets DMAC_CH*n*_CH_DISABLED_INTSTAT to 1 to indicate that the channel is disabled and generates an interrupt.

Note:

Once software initiates a channel disable procedure by writing 0 to DMAC_CH*n*_EN, before DMAC_CH*n*_CH_DISABLED_INT is set to 1, you cannot re-enable the channel by writing 1 to DMAC_CH*n*_EN. VDMA will ignore this write operation.

7.5.8 Low-Power Technique

VDMA implements a low-power technique that intelligently monitors idle periods across multiple DMA channels, the slave bus interface, and AXI master interface channels in various scenarios. By detecting inactivity, this feature can turn off the clock for these modules, significantly reducing overall power consumption.

When activity is detected, the logic quickly restores the clocks to these modules, ensuring that performance remains uncompromised.

Configure DMAC_LOWPOWER_CFG0_REG to enable the low-power feature and manage the low-power counters.

7.5.8.1 Low-Power Technique for DMA Channels

DMA channels occupy a significant portion of the chip area in the VDMA module, resulting in higher power consumption compared to other components. Therefore, optimizing power usage in this module is essential.

The low-power technique for DMA channels employs a **DMA channel low-power state machine** to monitor activities. When the state machine detects a low-power condition, it activates a **DMA channel low-power delay counter**. This counter runs until it expires. Once the counter expires, the DMA channels enter a low-power state by gating the AXI_DMAC_CORE_CLK.

The DMA channels will remain in this low-power state until any activity is detected. If activity is detected while the delay counter is still running, the counter is reset and paused until the low-power condition is detected again.

7.5.8.2 Low-Power Technique for Slave Bus Interface

The slave bus interface unit (SBIU) accesses the internal registers of VDMA via the APB3 slave interface. The low-power technique can optimize the power consumption in SBIU.

The low-power technique for the slave bus interface employs a **SBIU state machine** to monitor activities. When a low-power condition is detected (i.e., when there are no ongoing APB reads or writes and the SBIU state machine is idle), the **SBIU low power delay counter** is activated. Once the delay counter expires, SBIU enters the low-power state by gating PCLK or AXI_DMAC_CORE_CLK (based on the clock mode configuration).

SBIU remains in the low-power state until an active read or write transaction is detected on the slave bus interface. If an active transaction is detected while the SBIU low-power delay counter running, the delay counter is reset and paused until the low-power condition is detected again.

If a read or write transaction occurs while SBIU is in a low-power state, the low-power logic immediately exits the low-power state by un-gating PCLK or AXI_DMAC_CORE_CLK.

7.5.8.3 Low-Power Technique for AXI Master Interface Channels

The AXI master interface module implements the AXI bus to transfer data between memory and peripheral.

Whenever a low-power condition is detected on the AXI master interface channels, the **AXI master interface channel low-power delay counter** starts. The AXI low-power logic waits until the delay counter expires. After the delay counter expires, the AXI master interface channel enters the low-power state by gating the clock of AXI master interface and AXI_DMAC_CORE_CLK. It remains in this state until any activity is detected. If an activity is detected while the AXI master interface channel low-power delay counter is running, the delay counter is reset and paused until the low-power condition is detected again.

7.5.8.4 Global Low-Power Technique

The global low-power technique implements the low-power option based on the idle condition of the DMA channel, slave bus interface, and AXI master interface channels. If all the mentioned modules indicate that the VDMA is idle, the **global low-power delay counter** is activated.

The global low-power logic waits until the low-power delay counter expires. After the counter expires, VDMA enters the low-power state by gating the clocks of all modules. It remains in the low-power state until any activity is detected on any of the modules. If an activity is detected while the global low-power delay counter is running, the global delay counter is reset and paused until the low-power condition is again detected.

7.6 Interrupts

ESP32-P4's VDMA can generate the interrupt signal AXI_DMAC_INTR that will be sent to the Interrupt Matrix.

The interrupt signal is generated by the internal interrupt sources from VDMA.

- Common interrupts:
  - DMAC_SLVIF_UNDEFINEDREG_DEC_ERR_INT: Slave interface undefined register decode error interrupt
  - DMAC_SLVIF_COMMONREG_RD2WO_ERR_INT: Slave interface common register read to write-only error interrupt
  - DMAC_SLVIF_COMMONREG_WR2RO_ERR_INT: Slave interface common register write to read-only error Interrupt
  - DMAC_SLVIF_COMMONREG_DEC_ERR_INT: Slave interface common register decode error interrupt
- Channel interrupts (n = 1 ~ 4):
  - DMAC_CHn_CH_ABORTED_INT: Channel aborted interrupt
  - DMAC_CHn_CH_DISABLED_INT: Channel disabled interrupt
  - DMAC_CHn_CH_SUSPENDED_INT: Channel suspended interrupt
  - DMAC_CHn_CH_SRC_SUSPENDED_INT: Channel source suspended interrupt
  - DMAC_CHn_SLVIF_SHADOWREG_WRON_VALID_ERR_INT: Shadow register write on valid error interrupt

- DMAC_CH*n*_SLVIF_WRONCHEN_ERR_INT: Slave interface write on enabled channel error interrupt
- DMAC_CH*n*_SLVIF_RD2RWO_ERR_INT: Slave interface read to write-only error interrupt
- DMAC_CH*n*_SLVIF_WR2RO_ERR_INT: Slave interface write to read-only error interrupt
- DMAC_CH*n*_SLVIF_DEC_ERR_INT: Slave interface decode error interrupt
- DMAC_CH*n*_SLVIF_MULTIBLKTYPE_ERR_INT: Slave interface multi-block type error interrupt
- DMAC_CH*n*_SHADOWREG_OR_LLI_INVALID_ERR_INT: Shadow register or LLI invalid error interrupt
- DMAC_CH*n*_LLI_WR_SLV_ERR_INT: LLI write slave error interrupt
- DMAC_CH*n*_LLI_RD_SLV_ERR_INT: LLI read slave error interrupt
- DMAC_CH*n*_LLI_WR_DEC_ERR_INT: LLI write decode error interrupt
- DMAC_CH*n*_LLI_RD_DEC_ERR_INT: LLI read decode error interrupt
- DMAC_CH*n*_DST_SLV_ERR_INT: Destination slave error interrupt
- DMAC_CH*n*_SRC_SLV_ERR_INT: Source slave error interrupt
- DMAC_CH*n*_DST_DEC_ERR_INT: Destination decode error interrupt
- DMAC_CH*n*_SRC_DEC_ERR_INT: Source decode error interrupt
- DMAC_CH*n*_DST_TRANSCOMP_INT: Destination transfer completed interrupt
- DMAC_CH*n*_SRC_TRANSCOMP_INT: Source transfer completed interrupt
- DMAC_CH*n*_DMA_TFR_DONE_INT: DMA transfer done interrupt
- DMAC_CH*n*_BLOCK_TFR_DONE_INT: Block transfer done interrupt

7.7 Programming Procedures

This section outlines the programming steps for different types of transfers.

7.7.1 Common Programming Procedures

The following two steps are common and should be performed before other programming procedures.

1. Set HP_SYS_CLKRST_GDMA_SYS_CLK_EN to 1 to enable VDMA bus clock.
2. Configure TEE_DMA_GDMA_CH*n*_W_PMS and TEE_DMA_GDMA_CH*n*_R_PMS to define the access permission of VDMA. For more details please refer to Chapter 20 *Permission Control (PMS)*.

7.7.2 Programming Procedures for Shadow-Register-Based Multi-Block Transfer

1. Read DMAC_CH*n*_EN to select an available (unused) channel.
2. Configure DMAC_CH*n*_SRC_MULTBLK_TYPE and/or DMAC_CH*n*_DST_MULTBLK_TYPE to 2 to enable shadow-register-based multi-block transfer for source transfer and/or destination transfer.

3. Configure DMAC_CH*n*_SARO_REG and/or DMAC_CH*n*_DARO_REG, DMAC_CH*n*_BLOCK_TS, DMAC_CH*n*_CTLO_REG, and DMAC_CH*n*_CTL1_REG for the first block. VDMA loads the configured values to the corresponding shadow registers.

   Notice: The DMAC_CH*n*_SHADOWREG_OR_LLI_VALID field in DMAC_CH*n*_CTL1_REG must be the last one to be set to 1.

4. Set DMAC_CH*n*_EN to 1 to enable the corresponding channel.
5. VDMA initiates the DMA block transfer operation based on the configurations.
   - The block transfer might start immediately or after the handshaking request, depending on DMAC_CH*n*_TT_FC. Specifically,
     - When the source is memory, the source transfer starts immediately; when the destination is memory, the destination transfer starts immediately.
     - When the source is a peripheral, obtaining data from the source requires waiting for a handshake with the source. When the destination is a peripheral, writing data to the destination requires waiting for a handshake with the destination.
   - VDMA checks DMAC_CH*n*_SHADOWREG_OR_LLI_VALID and if it is seen as 0, VDMA waits until software writes 1 to DMAC_CH*n*_BLK_TFR_RESUMEREQ to indicate valid shadow registers availability. Then VDMA re-attempts shadow register fetch operation. In this case, VDMA may generate a DMAC_CH*n*_SHADOWREG_OR_LLI_INVALID_ERR_INT interrupt.
   - VDMA checks DMAC_CH*n*_SHADOWREG_OR_LLI_VALID and if it is seen as 1, VDMA copies the shadow register contents to the registers used for executing the DMA block transfer (i.e., DMAC_CH*n*_SARO_REG and/or DMAC_CH*n*_DARO_REG, DMAC_CH*n*_BLOCK_TS, DMAC_CH*n*_CTLO_REG, and DMAC_CH*n*_CTL1_REG) and clears DMAC_CH*n*_SHADOWREG_OR_LLI_VALID.
     - If VDMA sees DMAC_CH*n*_SHADOWREG_OR_LLI_LAST as 1, it understands that the current block is the final block in the transfer and completes the DMA transfer operation at the end of current block transfer.
     - If VDMA sees DMAC_CH*n*_SHADOWREG_OR_LLI_LAST as 0, it understands that there are one or more blocks to be transferred and checks DMAC_CH*n*_SHADOWREG_OR_LLI_VALID again at the end of current block transfer.
6. Software waits DMAC_CH*n*_SHADOWREG_OR_LLI_VALID to be cleared to 0.
7. Configure DMAC_CH*n*_SARO_REG and/or DMAC_CH*n*_DARO_REG, DMAC_CH*n*_BLOCK_TS, DMAC_CH*n*_CTLO_REG, and DMAC_CH*n*_CTL1_REG for the next block. If the next block is the last block, set DMAC_CH*n*_SHADOWREG_OR_LLI_LAST to 1.

   Notice: DMAC_CH*n*_SHADOWREG_OR_LLI_VALID must be the lase to be set to 1.

8. Software waits for a DMAC_CH*n*_BLOCK_TFR_DONE_INT interrupt or polls DMAC_CH*n*_BLOCK_TFR_DONE_INTSTAT until it reads as 1 and goes back to Step 6.

Note:
In cases where a DMAC_CH*n*_SHADOWREG_OR_LLI_INVALID_ERR_INT interrupt is generated, follow the recommended flow to resume transfer:

- Configure DMAC_CH*n*_SARO_REG and/or DMAC_CH*n*_DARO_REG, DMAC_CH*n*_BLOCK_TS, DMAC_CH*n*_CTLO_REG, and DMAC_CH*n*_CTL1_REG for the next block.
- Set DMAC_CH*n*_CLEAR_SHADOWREG_OR_LLI_INVALID_ERR_INTSTAT to 1 to clear the interrupt.
- Set DMAC_CH*n*_BLK_TFR_RESUMEREQ to 1 to request for resuming block transfer.

7.7.3 Programming Procedures for Linked-List-Based Multi-Block Transfer

1. Reads DMAC_CH*n*_EN to select an available (unused) channel.
2. Configure DMAC_CH*n*_SRC_MULTBLK_TYPE and/or DMAC_CH*n*_DST_MULTBLK_TYPE to 3 to enable linked-list-based transfer for source transfer and/or destination transfer.
3. Configure DMAC_CH*n*_LLPO_REG to define the base address of the first linked list item and the master interface on which the linked list item is available.
4. Create one or more linked list items in the system memory. You can create the entire linked list item in advance or dynamically extend it using DMAC_CH*n*_SHADOWREG_OR_LLI_VALID and DMAC_CH*n*_SHADOWREG_OR_LLI_LAST fields of the LLI. Follow the steps below to dynamically extend the linked list.
   - During the DMA transfer process, when the linked list is not ready, configure DMAC_CH*n*_SHADOWREG_OR_LLI_VALID in the linked list to 0 and wait until the linked list is prepared. Then set DMAC_CH*n*_SHADOWREG_OR_LLI_VALID to 1 to extend the linked list.
   - When it is time to end the DMA transfer, set DMAC_CH*n*_SHADOWREG_OR_LLI_LAST of the last linked list to 1 to indicate the end of the transfer.
5. Set DMAC_CH*n*_EN to 1 to enable the channel.

   Note:
   You can swap the sequence of Step 4 and Step 5. However, if Step 5 is performed before Step 4, or if the linked list item for the next block transfer is not available in system memory during the multi-block transfer (i.e., DMAC_CH*n*_SHADOWREG_OR_LLI_VALID of the fetched LLI is 0), VDMA may generate a DMAC_CH*n*_SHADOWREG_OR_LLI_INVALID_ERR_INT interrupt.

6. VDMA initiates the DMA block transfer operation based on the configurations. The block transfer might start immediately or after the handshaking request, depending on DMAC_CH*n*_TT_FC. VDMA copies the linked list contents to the registers used for executing the DMA block transfer (i.e., DMAC_CH*n*_SARO_REG and/or DMAC_CH*n*_DARO_REG, DMAC_CH*n*_BLOCK_TS, DMAC_CH*n*_CTLO_REG, and DMAC_CH*n*_CTL1_REG) and initiates the DMA block transfer.
7. During the linked list fetch phase:
   - If VDMA reads DMAC_CH*n*_SHADOWREG_OR_LLI_LAST as 1, it understands that the current block is the final block in the transfer and completes the DMA transfer operation at the end of the current block transfer.
   - If VDMA reads DMAC_CH*n*_SHADOWREG_OR_LLI_LAST as 0, it understands that there are one or more blocks to be transferred and goes to Step 6.

- If VDMA reads DMAC_CH*n*_SHADOWREG_OR_LLI_VALID as 0, it may generate a DMAC_CH*n*_SHADOWREG_OR_LLI_INVALID_ERR_INT interrupt. VDMA waits until software writes 1 to DMAC_CH*n*_BLK_TFR_RESUMEREQ to indicate valid LLI availability before re-attempting the LLI read operation.

7.7.4 Programming Procedures for Single-Block Transfer

1. Read DMAC_CH*n*_EN to choose a free (unused) channel.
2. Configure DMAC_CH*n*_SRC_MULTBLK_TYPE and DMAC_CH*n*_DST_MULTBLK_TYPE to 0 to enable contiguous-address-based single-block transfer.
3. Configure DMAC_CH*n*_SARO_REG and/or DMAC_CH*n*_DARO_REG, DMAC_CH*n*_BLOCK_TS, DMAC_CH*n*_CTLO_REG, and DMAC_CH*n*_CTL1_REG for the block.
4. Set DMAC_CH*n*_EN to 1 to enable the channel.
5. Software waits for the DMAC_CH*n*_BLOCK_TFR_DONE_INT interrupt or polls DMAC_CH*n*_BLOCK_TFR_DONE_INTSTAT until it is 1.

7.8 Register Summary

The addresses in this section are relative to the VDMA base address provided in Table 9.3-2 in Chapter 9 *System and Memory*.

The abbreviations given in Column **Access** are explained in Section *Access Types for Registers*.

Name	Description	Address	Access
Version Control Registers			
DMAC_IDO_REG	VDMA ID number	0x0000	RO
DMAC_COMPVERO_REG	VDMA version number	0x0008	RO
Configuration Registers			
DMAC_CFGO_REG	VDMA Configuration register	0x0010	R/W
DMAC_CHENO_REG	VDMA channel enable register	0x0018	varies
DMAC_RESETO_REG	VDMA reset control register	0x0058	R/W
DMAC_LOWPOWER_CFGO_REG	VDMA low power configuration register 0	0x0060	R/W
DMAC_LOWPOWER_CFG1_REG	VDMA low power configuration register 1	0x0064	R/W
DMAC_CH1_SARO_REG	VDMA channel 1 source address register	0x0100	R/W
DMAC_CH1_DARO_REG	VDMA channel 1 destination address register	0x0108	R/W
DMAC_CH1_BLOCK_TSO_REG	VDMA channel 1 block transfer size register	0x0110	R/W
DMAC_CH1_CTLO_REG	VDMA channel 1 control register 0	0x0118	R/W
DMAC_CH1_CTL1_REG	VDMA channel 1 control register 1	0x011C	R/W
DMAC_CH1_CFGO_REG	VDMA channel 1 configuration register 0	0x0120	varies
DMAC_CH1_CFG1_REG	VDMA channel 1 configuration register 1	0x0124	varies
DMAC_CH1_LLPO_REG	VDMA channel 1 linked list pointer register	0x0128	R/W
DMAC_CH1_BLK_TFR_RESUMEREQO_REG	VDMA channel 1 block transfer resume request register	0x0148	WO
DMAC_CH1_AXI_IDO_REG	VDMA channel 1 AXI ID register	0x0150	R/W
DMAC_CH2_SARO_REG	VDMA channel 2 source address register	0x0200	R/W
DMAC_CH2_DARO_REG	VDMA channel 2 destination address register	0x0208	R/W
DMAC_CH2_BLOCK_TSO_REG	VDMA channel 2 block transfer size register	0x0210	R/W
DMAC_CH2_CTLO_REG	VDMA channel 2 control register 0	0x0218	R/W
DMAC_CH2_CTL1_REG	VDMA channel 2 control register 1	0x021C	R/W

Name	Description	Address	Access
DMAC_CH2_CFG0_REG	VDMA channel 2 configuration register 0	0x0220	varies
DMAC_CH2_CFG1_REG	VDMA channel 2 configuration register 1	0x0224	varies
DMAC_CH2_LLP0_REG	VDMA channel 2 linked list pointer register	0x0228	R/W
DMAC_CH2_BLK_TFR_RESUMEREQ0_REG	VDMA channel 2 block transfer resume request register	0x0248	WO
DMAC_CH2_AXI_ID0_REG	VDMA channel 2 AXI ID register	0x0250	R/W
DMAC_CH3_SAR0_REG	VDMA channel 3 source address register	0x0300	R/W
DMAC_CH3_DAR0_REG	VDMA channel 3 destination address register	0x0308	R/W
DMAC_CH3_BLOCK_TS0_REG	VDMA channel 3 block transfer size register	0x0310	R/W
DMAC_CH3_CTL0_REG	VDMA channel 3 control register 0	0x0318	R/W
DMAC_CH3_CTL1_REG	VDMA channel 3 control register 1	0x031C	R/W
DMAC_CH3_CFG0_REG	VDMA channel 3 configuration register 0	0x0320	varies
DMAC_CH3_CFG1_REG	VDMA channel 3 configuration register 1	0x0324	varies
DMAC_CH3_LLP0_REG	VDMA channel 3 linked list pointer register	0x0328	R/W
DMAC_CH3_BLK_TFR_RESUMEREQ0_REG	VDMA channel 3 block transfer resume request register	0x0348	WO
DMAC_CH3_AXI_ID0_REG	VDMA channel 3 AXI ID register	0x0350	R/W
DMAC_CH4_SAR0_REG	VDMA channel 4 source address register	0x0400	R/W
DMAC_CH4_DAR0_REG	VDMA channel 4 destination address register	0x0408	R/W
DMAC_CH4_BLOCK_TS0_REG	VDMA channel 4 block transfer size register	0x0410	R/W
DMAC_CH4_CTL0_REG	VDMA channel 4 control register 0	0x0418	R/W
DMAC_CH4_CTL1_REG	VDMA channel 4 control register 1	0x041C	R/W
DMAC_CH4_CFG0_REG	VDMA channel 4 configuration register 0	0x0420	varies
DMAC_CH4_CFG1_REG	VDMA channel 4 configuration register 1	0x0424	varies
DMAC_CH4_LLP0_REG	VDMA channel 4 linked list pointer register	0x0428	R/W
DMAC_CH4_BLK_TFR_RESUMEREQ0_REG	VDMA channel 4 block transfer resume request register	0x0448	WO
DMAC_CH4_AXI_ID0_REG	VDMA channel 4 AXI ID register	0x0450	R/W
Interrupt Registers			

Name	Description	Address	Access
DMAC_INTSTATUSO_REG	VDMA interrupt status register	0x0030	RO
DMAC_COMMONREG_INTCLEARO_REG	VDMA common interrupt clear register	0x0038	WO
DMAC_COMMONREG_INTSTATUS_ENABLEO_REG	VDMA common interrupt status enable register	0x0040	varies
DMAC_COMMONREG_INTSIGNAL_ENABLEO_REG	VDMA common interrupt signal enable register	0x0048	varies
DMAC_COMMONREG_INTSTATUSO_REG	VDMA common interrupt status register	0x0050	RO
DMAC_CH1_INTSTATUS_ENABLEO_REG	VDMA channel 1 interrupt status enable register	0x0180	varies
DMAC_CH1_INTSTATUSO_REG	VDMA channel 1 interrupt status register	0x0188	RO
DMAC_CH1_INTSIGNAL_ENABLEO_REG	VDMA channel 1 interrupt signal enable register	0x0190	varies
DMAC_CH1_INTCLEARO_REG	VDMA channel 1 interrupt clear register	0x0198	WO
DMAC_CH2_INTSTATUS_ENABLEO_REG	VDMA channel 2 interrupt status enable register	0x0280	varies
DMAC_CH2_INTSTATUSO_REG	VDMA channel 2 interrupt status register	0x0288	RO
DMAC_CH2_INTSIGNAL_ENABLEO_REG	VDMA channel 2 interrupt signal enable register	0x0290	varies
DMAC_CH2_INTCLEARO_REG	VDMA channel 2 interrupt clear register	0x0298	WO
DMAC_CH3_INTSTATUS_ENABLEO_REG	VDMA channel 3 interrupt status enable register	0x0380	varies
DMAC_CH3_INTSTATUSO_REG	VDMA channel 3 interrupt status register	0x0388	RO
DMAC_CH3_INTSIGNAL_ENABLEO_REG	VDMA channel 3 interrupt signal enable register	0x0390	varies
DMAC_CH3_INTCLEARO_REG	VDMA channel 3 interrupt clear register	0x0398	WO
DMAC_CH4_INTSTATUS_ENABLEO_REG	VDMA channel 4 interrupt status enable register	0x0480	varies
DMAC_CH4_INTSTATUSO_REG	VDMA channel 4 interrupt status register	0x0488	RO
DMAC_CH4_INTSIGNAL_ENABLEO_REG	VDMA channel 4 interrupt signal enable register	0x0490	varies
DMAC_CH4_INTCLEARO_REG	VDMA channel 4 interrupt clear register	0x0498	WO
status registers			
DMAC_CH1_STATUSO_REG	VDMA channel 1 status register 0	0x0130	RO
DMAC_CH1_STATUS1_REG	VDMA channel 1 status register 1	0x0134	RO
DMAC_CH1_SSTATO_REG	VDMA channel 1 source status register	0x0160	RO
DMAC_CH1_DSTATO_REG	VDMA channel 1 destination status register	0x0168	RO
DMAC_CH1_SSTATARO_REG	VDMA channel 1 source status address register	0x0170	R/W

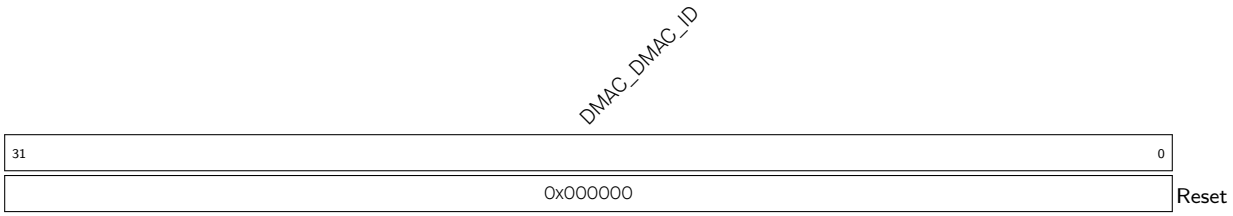
Name	Description	Address	Access
DMAC_CH1_DSTATARO_REG	VDMA channel 1 destination status address register	0x0178	R/W
DMAC_CH2_STATUSO_REG	VDMA channel 2 status register 0	0x0230	RO
DMAC_CH2_STATUS1_REG	VDMA channel 2 status register 1	0x0234	RO
DMAC_CH2_SSTATO_REG	VDMA channel 2 source status register	0x0260	RO
DMAC_CH2_DSTATO_REG	VDMA channel 2 destination status register	0x0268	RO
DMAC_CH2_SSTATARO_REG	VDMA channel 2 source status address register	0x0270	R/W
DMAC_CH2_DSTATARO_REG	VDMA channel 2 destination status address register	0x0278	R/W
DMAC_CH3_STATUSO_REG	VDMA channel 3 status register 0	0x0330	RO
DMAC_CH3_STATUS1_REG	VDMA channel 3 status register 1	0x0334	RO
DMAC_CH3_SSTATO_REG	VDMA channel 3 source status register	0x0360	RO
DMAC_CH3_DSTATO_REG	VDMA channel 3 destination status register	0x0368	RO
DMAC_CH3_SSTATARO_REG	VDMA channel 3 source status address register	0x0370	R/W
DMAC_CH3_DSTATARO_REG	VDMA channel 3 destination status address register	0x0378	R/W
DMAC_CH4_STATUSO_REG	VDMA channel 4 status register 0	0x0430	RO
DMAC_CH4_STATUS1_REG	VDMA channel 4 status register 1	0x0434	RO
DMAC_CH4_SSTATO_REG	VDMA channel 4 source status register	0x0460	RO
DMAC_CH4_DSTATO_REG	VDMA channel 4 destination status register	0x0468	RO
DMAC_CH4_SSTATARO_REG	VDMA channel 4 source status address register	0x0470	R/W
DMAC_CH4_DSTATARO_REG	VDMA channel 4 destination status address register	0x0478	R/W

7.9 Registers

The addresses in this section are relative to the VDMA base address provided in Table 9.3-2 in Chapter 9 *System and Memory*.

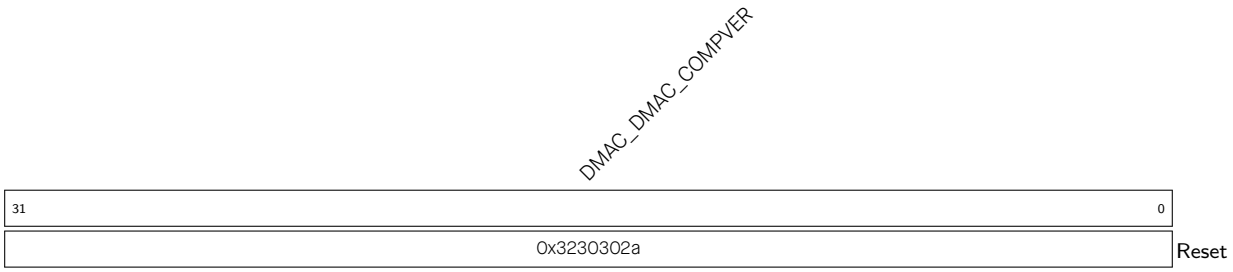
For how to program reserved fields, please refer to Section Programming Reserved Register Field.

Register 7.1. DMAC_IDO_REG (0x0000)



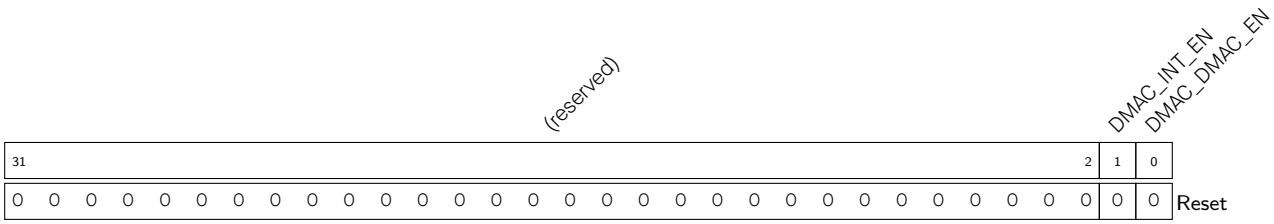
DMAC_DMAC_ID Represents VDMA ID number. (RO)

Register 7.2. DMAC_COMPVER0_REG (0x0008)



DMAC_DMAC_COMPVER Represents VDMA version number. (RO)

Register 7.3. DMAC_CFG0_REG (0x0010)



DMAC_DMAC_EN Configures whether to enable VDMA.

0: Disable

1: Enable

Note: If this field is cleared but a channel is still active, this field will still return 1. When hardware has terminated activity on all channels, this field will return 0.

(R/W)

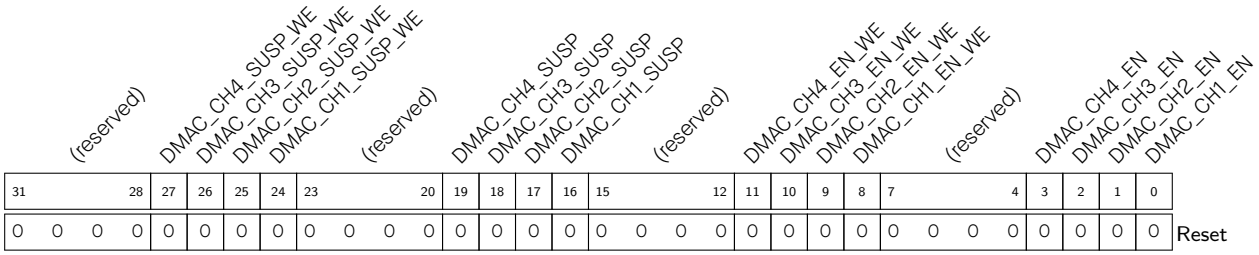
DMAC_INT_EN Configures whether to enable global interrupt.

0: Disable

1: Enable

(R/W)

Register 7.4. DMAC_CHENO_REG (0x0018)



DMAC_CH*n*_EN (*n*: 1-4) Configures whether to enable VDMA channel *n*. Hardware automatically clears this bit after the last AXI transfer of the DMA transfer has been completed. Software can poll this field to determine when this channel is free for a new DMA transfer.

0: Disable channel *n*

1: Enable channel *n*

(R/W)

DMAC_CH*n*_EN_WE (*n*: 1-4) Configures whether to enable write to DMAC_CH*n*_EN. This field always reads as 0.

0: Disable write to DMAC_CH*n*_EN

1: Enable write to DMAC_CH*n*_EN

(WO)

DMAC_CH*n*_SUSP (*n*: 1-4) Configures whether to suspend channel *n*.

0: Do not suspend channel *n*

1: Suspend channel *n*

Software can clear this field to 0 after VDMA sets DMAC_CH*n*_CH_SUSPENDED_INTSTAT to 1, to exit the channel suspend mode.

Note: This field is cleared when channel *n* is disabled.

(R/W)

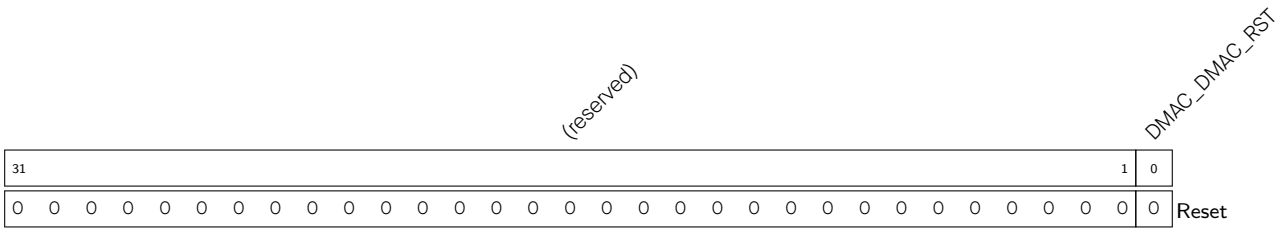
DMAC_CH*n*_SUSP_WE (*n*: 1-4) Configures whether to enable write to DMAC_CH*n*_SUSP. This field always reads as 0.

0: Disable write to DMAC_CH*n*_SUSP

1: Enable write to DMAC_CH*n*_SUSP

(WO)

Register 7.5. DMAC_RESET0_REG (0x0058)

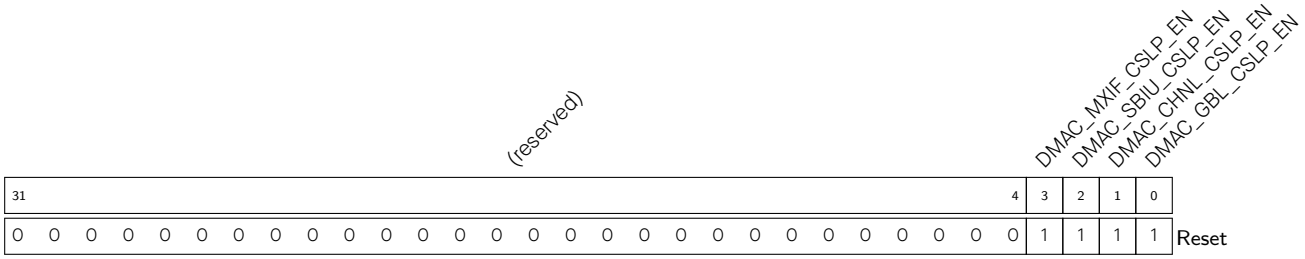


DMAC_DMAC_RST Configures whether to reset VDMA. Software writes 1 to this field to reset VDMA and polls this field to see if it is 0. VDMA resets all the modules except the APB slave interface module and clears this field to 0.

Note: Software is not allowed to write 0 to this field.

(R/W)

Register 7.6. DMAC_LOWPOWER_CFG0_REG (0x0060)



DMAC_GBL_CSLP_EN Configures whether to enable global low-power feature.

0: Disable

1: Enable

(R/W)

DMAC_CHNL_CSLP_EN Configures whether to enable low-power feature for DMA channels.

0: Disable

1: Enable

(R/W)

DMAC_SBIU_CSLP_EN SBIU Configures whether to enable low-power feature for slave bus interface.

0: Disable

1: Enable

(R/W)

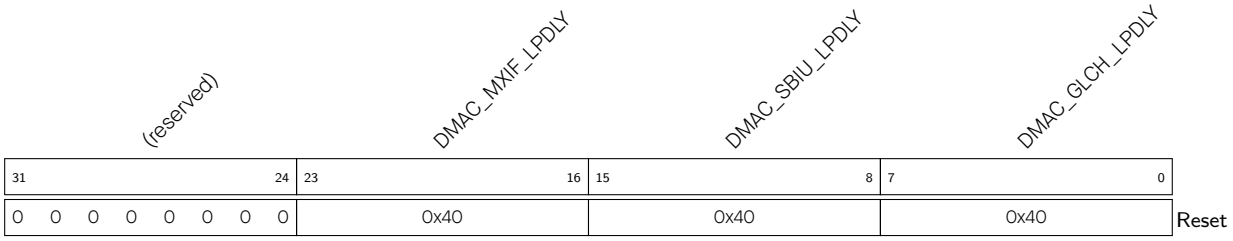
DMAC_MXIF_CSLP_EN Configures whether to enable low-power feature for AXI master interface.

0: Disable

1: Enable

(R/W)

Register 7.7. DMAC_LOWPOWER_CFG1_REG (0x0064)

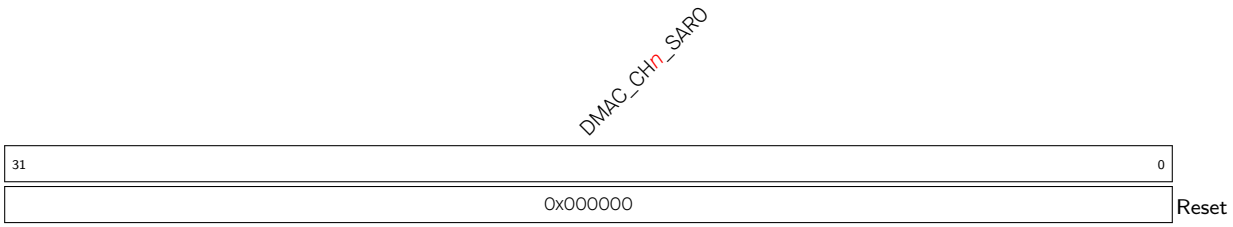


DMAC_GLCH_LPDLY Configures the value of the AXI master interface low-power delay counter. The programmed value must be greater than or equal to 0x4. If it is less than 0x4, the field will reset to 0x40. (R/W)

DMAC_SBIU_LPDLY Configures the value of the SBIU low-power delay counter. The programmed value must be greater than or equal to 0x4. If it is less than 0x4, the field will reset to 0x40. (R/W)

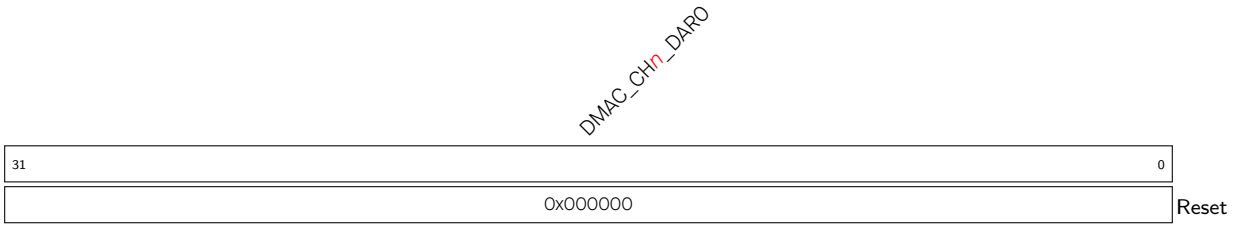
DMAC_MXIF_LPDLY Configures the value of the global low-power delay counter and the DMA channel low-power delay counter. The programmed value must be greater than or equal to 0x4. If it is less than 0x4, the field will reset to 0x40. (R/W)

Register 7.8. DMAC_CH*n*_SARO_REG (*n*: 1-4) (0x0100*n*)



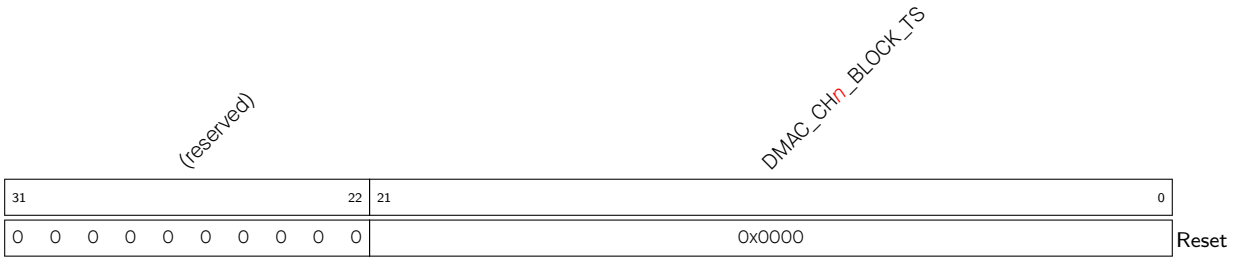
DMAC_CH*n*_SARO Configures the source address of DMA transfer. This field is updated after each source transfer. DMAC_CH*n*_SINC determines whether the address increments or remains unchanged on every source transfer throughout the block transfer. (R/W)

Register 7.9. DMAC_CH*n*_DARO_REG (*n*: 1-4) (0x0100*n* + 0x0008)



DMAC_CH*n*_DARO Configures the destination address of DMA transfer. This field is updated after each source transfer. DMAC_CH*n*_DINC determines whether the address increments or remains unchanged on every destination transfer throughout the block transfer. (R/W)

Register 7.10. DMAC_CH*n*_BLOCK_TSO_REG (*n*: 1-4) (0x0100*n* + 0x0010)

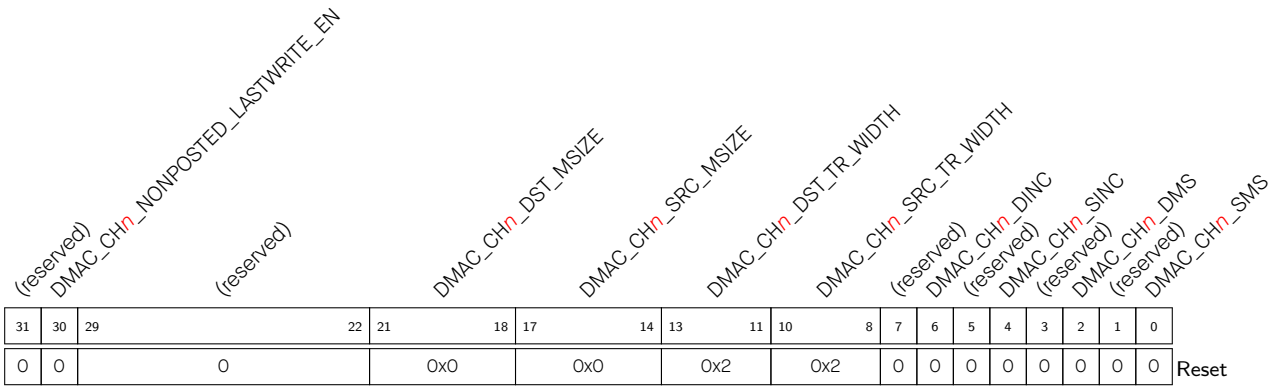


DMAC_CH*n*_BLOCK_TS Configures the total number of DMAC_CH*n*_SRC_TR_WIDTH in a DMA transfer. The programming range of this field is 0 ~ 4,194,303.

Block transfer size = Programmed value + 1

(R/W)

Register 7.11. DMAC_CH*n*_CTL0_REG (*n*: 1-4) (0x0100*n* + 0x0018)



DMAC_CH*n*_SMS Configures which AXI master to access the source.

0: AXI Master 1

1: AXI Master 2

(R/W)

DMAC_CH*n*_DMS Configures which AXI master to access the destination.

0: AXI Master 1

1: AXI Master 2

(R/W)

DMAC_CH*n*_SINC Configures whether to increment the source address on every source transfer. If VDMA is fetching data from a source peripheral's FIFO at a fixed address, then set this field to 1.

0: Increment

1: No change

(R/W)

DMAC_CH*n*_DINC Configures whether to increment the destination address on every destination transfer. If VDMA is writing data to a source peripheral's FIFO at a fixed address, then set this field to 1.

0: Increment

1: No change

(R/W)

DMAC_CH*n*_SRC_TR_WIDTH Configures source transfer width.

0x0: 8 bits

0x1: 16 bits

0x2: 32 bits

0x3: 64 bits

(R/W)

Continued on the next page...

Register 7.11. DMAC_CH*n*_CTL0_REG (*n*: 1-4) (0x0100**n* + 0x0018)

Continued from the previous page...

DMAC_CH*n*_DST_TR_WIDTH Configures destination transfer width.

0x0: 8 bits
0x1: 16 bits
0x2: 32 bits
0x3: 64 bits
(R/W)

DMAC_CH*n*_SRC_MSIZE Configures the length of the source burst transaction. Every time the handshaking interface requests for a source burst transaction, the read value of this field from the source is the number of data items, each of width DMAC_CH*n*_SRC_TR_WIDTH.

0x0: 1 data item
0x1: 4 data items
0x2: 8 data items
0x3: 16 data items
0x4: 32 data items
0x5: 64 data items
0x6: 128 data items
0x7: 256 data items
0x8: 512 data items
0x9: 1024 data items
(R/W)

DMAC_CH*n*_DST_MSIZE Configures the length of the destination burst transaction. Every time the handshaking interface requests for a destination burst transaction, the read value of this field from the destination is the number of data items, each of width DMAC_CH*n*_SRC_TR_WIDTH.

0x0: 1 data item
0x1: 4 data items
0x2: 8 data items
0x3: 16 data items
0x4: 32 data items
0x5: 64 data items
0x6: 128 data items
0x7: 256 data items
0x8: 512 data items
0x9: 1024 data items
(R/W)

Continued on the next page...

Register 7.11. DMAC_CH*n*_CTL0_REG (*n*: 1-4) (0x0100**n* + 0x0018)

Continued from the previous page...

DMAC_CH*n*_NONPOSTED_LASTWRITE_EN Configures whether to enable non-posted writes throughout the block transfer.

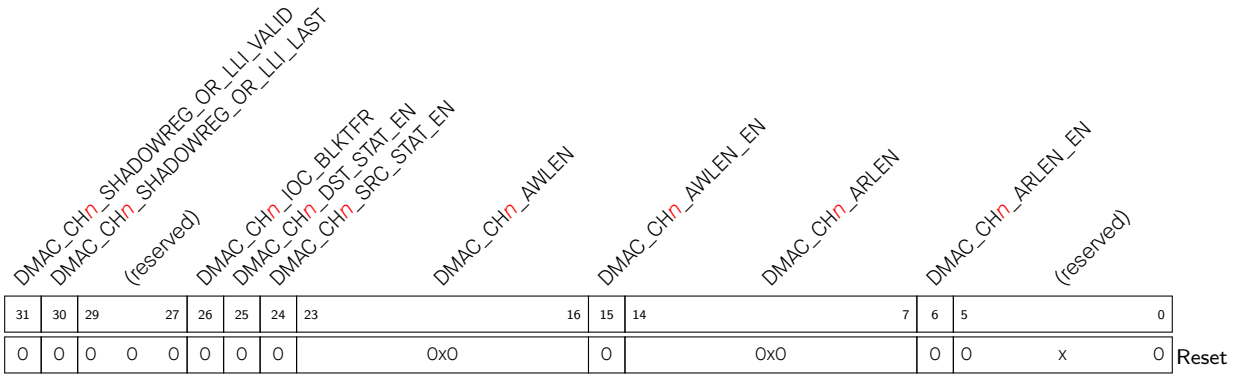
Posted writes refers to sending data without the need to write the data into the destination (the data may still be in transit); VDMA can consider this write request to be complete. Non-posted writes refers to sending data that needs to be written into the destination and requires a response, only then can VDMA consider this write request to be complete.

0: Posted writes can be used throughout the block transfer.

1: Posted writes can be used at the end of the block transfer (within the block). The last write must be non-posted to ensure synchronization between the generation of the block completion interrupt and the final written data reaching the destination.

(R/W)

Register 7.12. DMAC_CH*n*_CTL1_REG (*n*: 1-4) (0x0100*n* + 0x001C)



DMAC_CH*n*_ARLEN_EN Configures whether to enable the length of the source burst transfer.

0: VDMA uses any possible value that is less than or equal to 16 as the length of the source burst transfer.

1: VDMA uses the value of DMAC_CH*n*_ARLEN as the length of the source burst transfer till the extent possible. The remaining transfers use the maximum possible burst transfer length. For example, if the data size is 192 bytes and the source burst transfer length DMAC_CH*n*_ARLEN is set to 128 bytes, then after transmitting 128 bytes, the remaining 64 bytes is the maximum possible burst transfer length.

(R/W)

DMAC_CH*n*_ARLEN Configures the length of the source burst transfer. The specified burst length is used for source data transfer till the extent possible. The remaining transfers use the maximum possible value that is less than or equal to 16.

(R/W)

DMAC_CH*n*_AWLEN_EN Configures whether to enable the length of the destination burst transfer.

0: VDMA uses any possible value that is less than or equal to 16 as the length of the destination burst transfer.

1: VDMA uses the value of DMAC_CH*n*_AWLEN as the length of the destination burst transfer till the extent possible. The remaining transfers use the maximum possible burst transfer length.

(R/W)

DMAC_CH*n*_AWLEN Configures the length of the destination burst transfer. The specified burst length is used for destination data transfer till the extent possible. The remaining transfers use the maximum possible value that is less than or equal to 16.

(R/W)

DMAC_CH*n*_SRC_STAT_EN Configures whether to enable fetch of source status.

This logic enables the fetch of the status from the source peripheral of channel *n* pointed to by DMAC_CH*n*_SSTATARO. The value is subsequently stored in DMAC_CH*n*_SSTAT. At the end of each block transfer, the status value is written back to the DMAC_CH*n*_SSTAT register of the linked list, if either source or destination peripheral uses linked-list-based multi-block transfer.

0: Do not fetch source status

1: Fetch source status and store the value in DMAC_CH*n*_SSTAT

(R/W)

Continued on the next page...

Register 7.12. DMAC_CH*n*_CTL1_REG (*n*: 1-4) (0x0100**n* + 0x001C)

Continued from the previous page...

DMAC_CH*n*_DST_STAT_EN Configures whether to enable fetch of destination status.

This logic enables the fetch of the status from the destination peripheral of channel *n* pointed to by DMAC_CH*n*_DSTATARO. The value is subsequently stored in DMAC_CH*n*_DSTAT. At the end of each block transfer, the status value is written back to the DMAC_CH*n*_DSTAT register of the linked list, if either source or destination peripheral uses linked-list-based multi-block transfer.

0: Do not fetch destination status

1: Fetch destination status and store the value in DMAC_CH*n*_DSTAT

(R/W)

DMAC_CH*n*_IOC_BLKTFR Configures whether to enable the interrupt on completion of multi-block transfer based on shadow register or linked list.

Note: If the source and destination uses contiguous-address or auto-reloading-based multi-block transfer, this field cannot be modified on a block by block basis. In addition, the programmed value before enabling the channel will be used for all the blocks in DMA transfer.

0: Disable DMAC_CH*n*_BLOCK_TFR_DONE_INTSTAT

1: Enable DMAC_CH*n*_BLOCK_TFR_DONE_INTSTAT, valid when DMAC_CH*n*_ENABLE_BLOCK_TFR_DONE_INTSTAT is 1. To output a DMAC_CH*n*_BLOCK_TFR_DONE_INT interrupt, set DMAC_CH*n*_ENABLE_BLOCK_TFR_DONE_INTSIGNAL to 1.

(R/W)

DMAC_CH*n*_SHADOWREG_OR_LLI_LAST Configures whether the shadow register or linked list item is the last one.

0: Not the last one

1: The last one

(R/W)

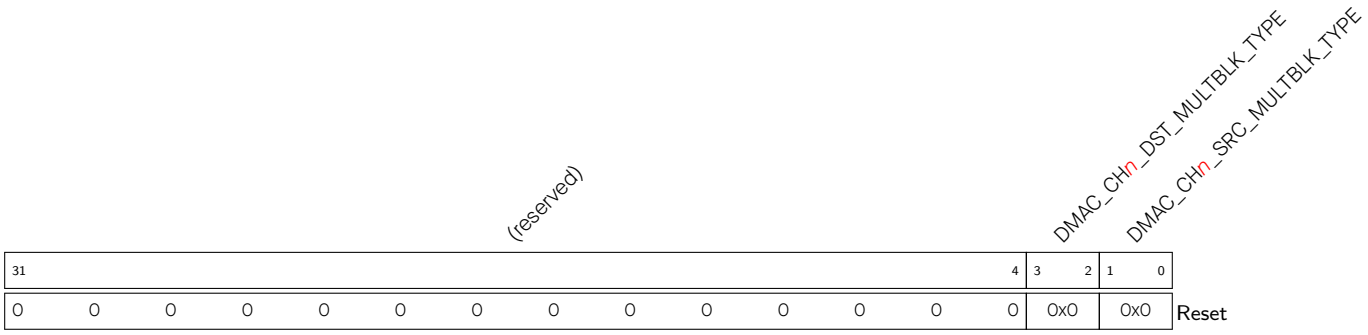
DMAC_CH*n*_SHADOWREG_OR_LLI_VALID Configures whether the contents of shadow register or the linked list item are valid.

0: Invalid

1: Valid

(R/W)

Register 7.13. DMAC_CH*n*_CFG0_REG (*n*: 1-4) (0x0100*n* + 0x0120)



DMAC_CH*n*_SRC_MULTBLK_TYPE Configures the source multi-block transfer type.

0: Contiguous address

1: Auto reloading

2: Shadow register

3: Linked list

(R/W)

DMAC_CH*n*_DST_MULTBLK_TYPE Configures the destination multi-block transfer type.

0: Contiguous address

1: Auto reloading

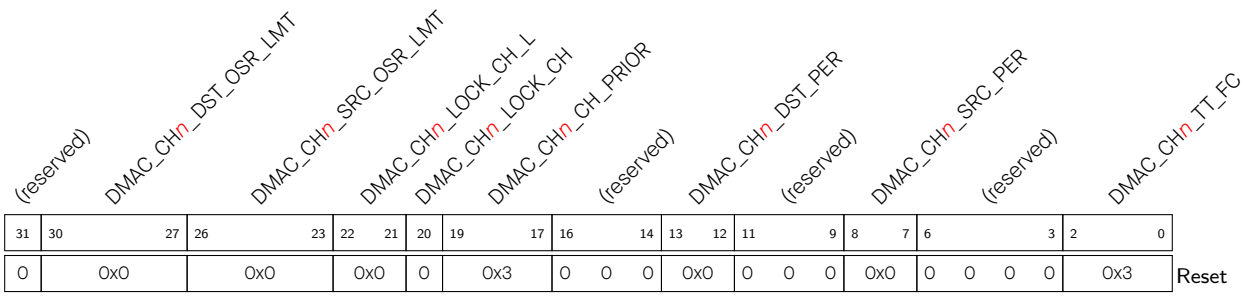
2: Shadow register

3: Linked list

If both DMAC_CH*n*_SRC_MULTBLK_TYPE and DMAC_CH*n*_DST_MULTBLK_TYPE are 0, then the transfer type is single-block transfer based on contiguous address.

(R/W)

Register 7.14. DMAC_CH*n*_CFG1_REG (*n*: 1-4) (0x0100*n* + 0x0124)



DMAC_CH*n*_TT_FC Configures transfer type and flow controller.

0x0: Transfer type is memory to memory and flow controller is VDMA

0x1: Transfer type is memory to peripheral and flow controller is VDMA

0x2: Transfer type is peripheral to memory and flow controller is VDMA

0x3: Invalid

0x4: Transfer type is peripheral to memory and flow controller is source peripheral

0x5: Invalid

0x6: Transfer type is memory to peripheral and flow controller is destination peripheral

0x7: Invalid

(R/W)

DMAC_CH*n*_SRC_PER Assigns a handshaking interface (0 ~ 2) to the source of channel *n*. (R/W)

DMAC_CH*n*_DST_PER Assigns a handshaking interface (0 ~ 2) to the destination of channel *n*. (R/W)

DMAC_CH*n*_CH_PRIOR Assigns channel priority (0 ~ 3). A priority of 3 is the highest priority, and 0 is the lowest. A programmed value outside this range will cause erroneous behavior. (R/W)

DMAC_CH*n*_LOCK_CH VDMA does not support lock feature. Reads of this field always return 0. (RO)

DMAC_CH*n*_LOCK_CH_L VDMA does not support lock feature. Reads of this field always return 0. (RO)

DMAC_CH*n*_SRC_OSR_LMT Configures the limit of the source AXI Outstanding request. The maximum number of AXI Outstanding request is 16.

Source AXI Outstanding request limit = Programmed value + 1.

AXI Outstanding is a feature in the AXI protocol that allows to send request before the previous request response is received, thereby increasing the bandwidth of the bus interface.

(R/W)

DMAC_CH*n*_DST_OSR_LMT Configures the limit of the destination AXI Outstanding request. The maximum number of AXI Outstanding request is 16.

Destination AXI Outstanding request limit = Programmed value + 1.

(R/W)

Register 7.15. DMAC_CH*n*_LLP0_REG (*n*: 1-4) (0x0100*n* + 0x0128)



DMAC_CH*n*_LMS Selects the AXI master for accessing LLI.

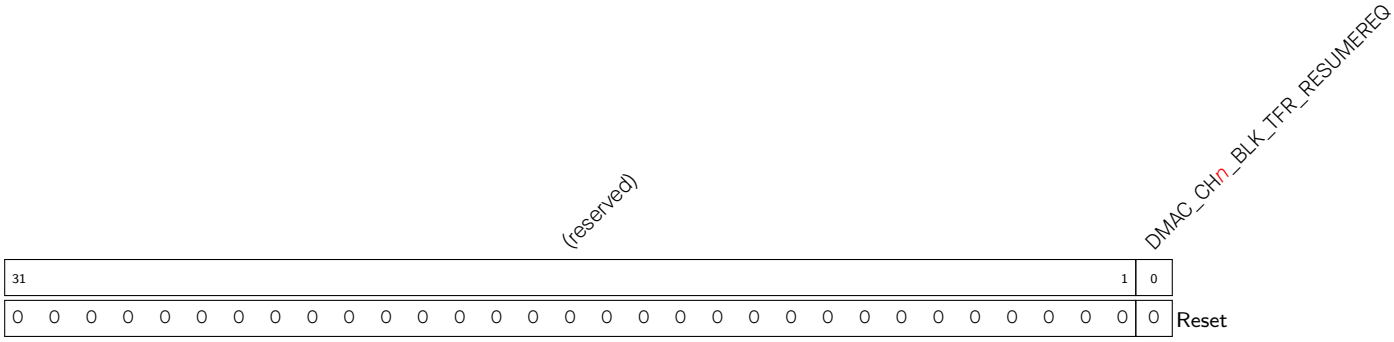
0: AXI Master 1

1: AXI Master 2

(R/W)

DMAC_CH*n*_LOC0 Configures the starting address of the first linked list item in memory. The six least significant bits (LSB) of the starting address are not stored because it is assumed that the address is 64-byte aligned. (R/W)

Register 7.16. DMAC_CH*n*_BLK_TFR_RESUMEREQ0_REG (*n*: 1-4) (0x0100*n* + 0x0148)



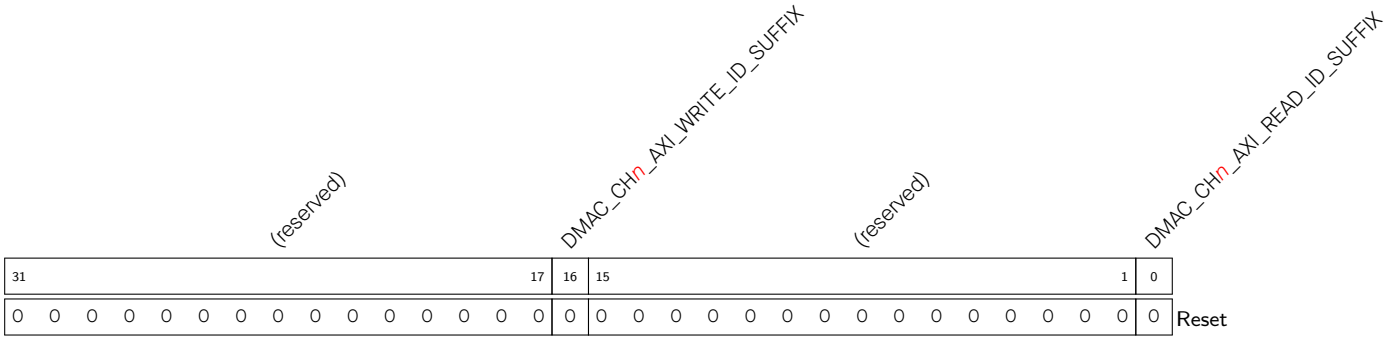
DMAC_CH*n*_BLK_TFR_RESUMEREQ Configures whether to request to resume block transfer during linked-list or shadow-register-based multi-block transfer.

0: Do not request to resume block transfer

1: Request to resume block transfer

(WO)

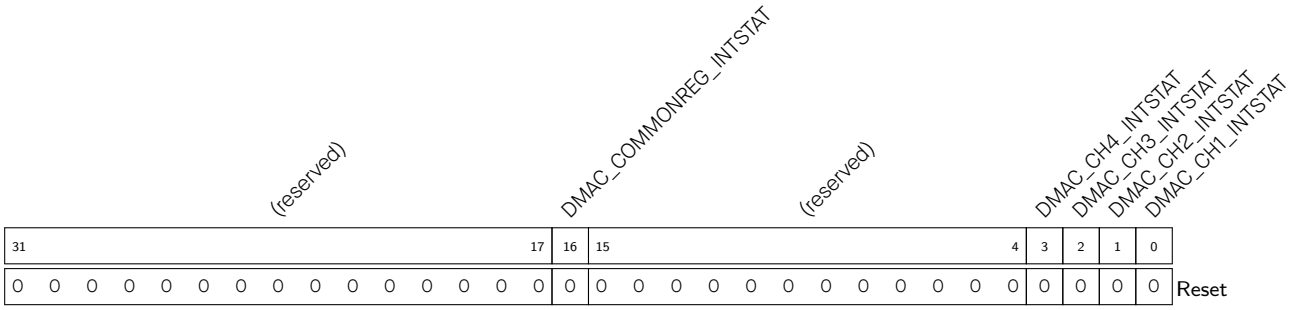
Register 7.17. DMAC_CH*n*_AXI_IDO_REG (0x0150)



DMAC_CH*n*_AXI_READ_ID_SUFFIX Configure the least significant bit of AXI read ID. (R/W)

DMAC_CH*n*_AXI_WRITE_ID_SUFFIX Configure the least significant bit of AXI write ID. (R/W)

Register 7.18. DMAC_INTSTATUSO_REG (0x0030)



DMAC_CH*n*_INTSTAT (*n*: 1-4) Indicates the interrupt status for channel *n*.

1: Interrupt is active

0: Interrupt is inactive

(RO)

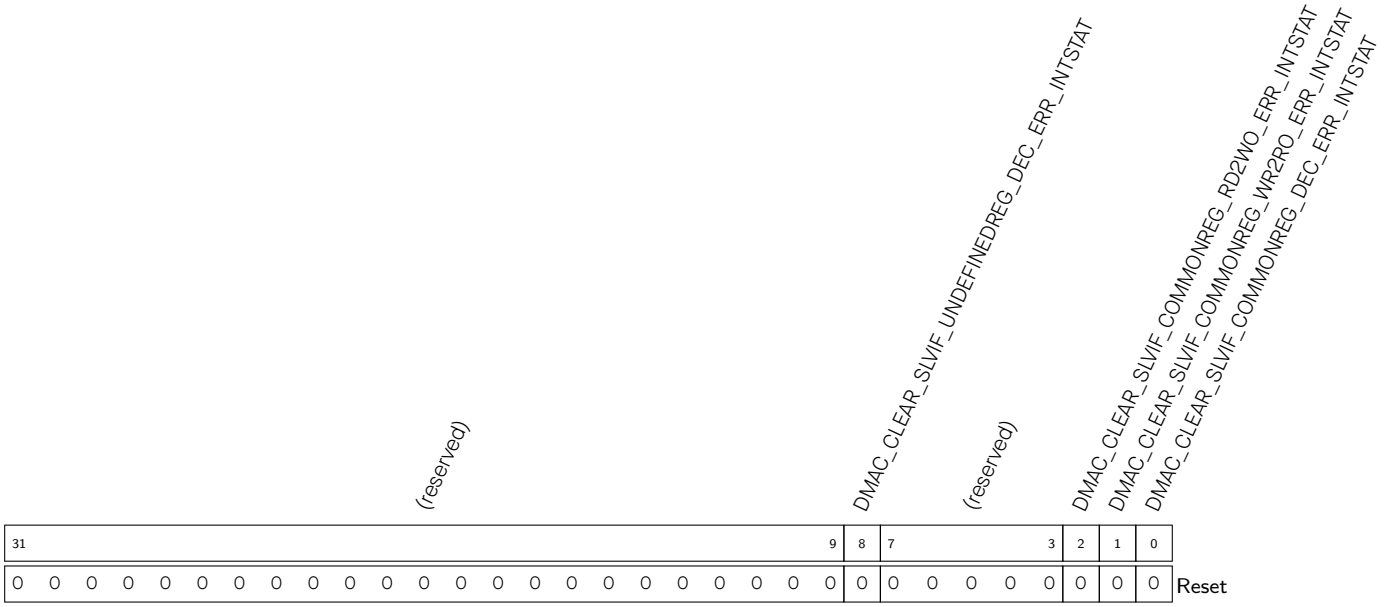
DMAC_COMMONREG_INTSTAT Indicates the common register interrupt status.

1: Interrupt is active

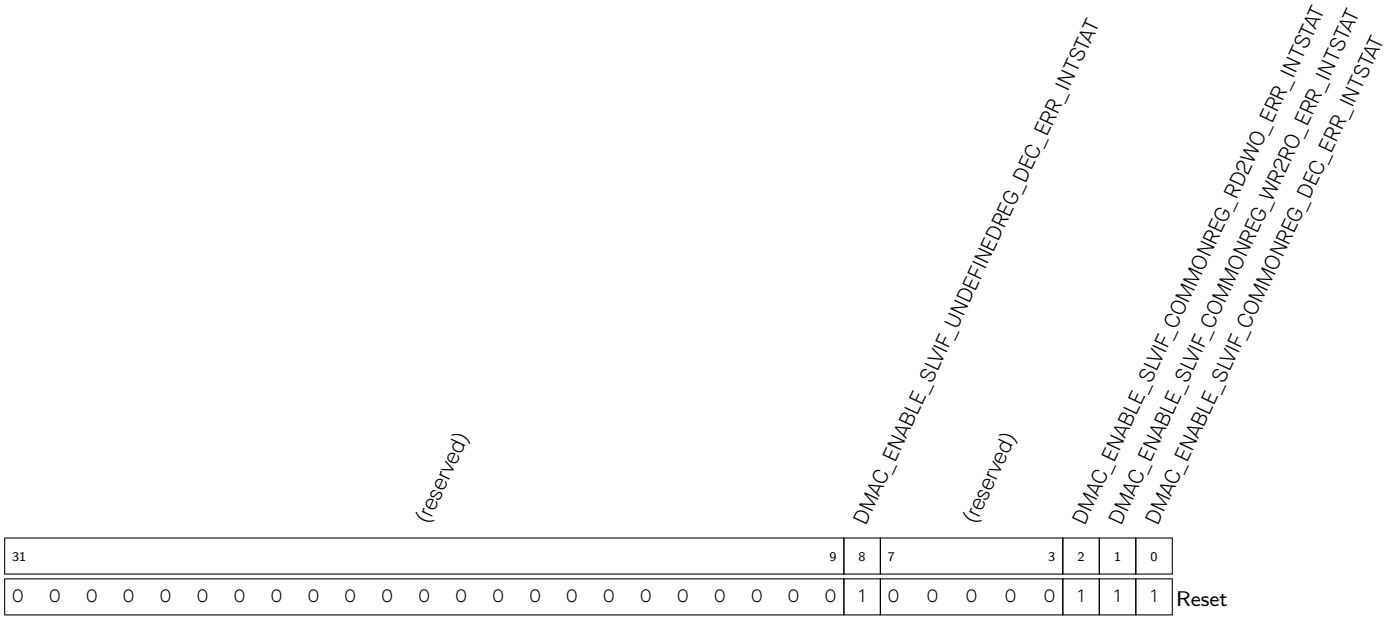
0: Interrupt is inactive

(RO)

Register 7.19. DMAC_COMMONREG_INTCLEAR0_REG (0x0038)


(reserved)
DMAC_CLEAR_SLVIF_UNDEFINEDREG_DEC_ERR_INTSTAT
(reserved)
DMAC_CLEAR_SLVIF_COMMONREG_RD2WO_ERR_INTSTAT
DMAC_CLEAR_SLVIF_COMMONREG_WR2RO_ERR_INTSTAT
DMAC_CLEAR_SLVIF_COMMONREG_DEC_ERR_INTSTAT
31 9 8 7 3 2 1 0
0 0 0

Register 7.20. DMAC_COMMONREG_INTSTATUS_ENABLEO_REG (0x0040)



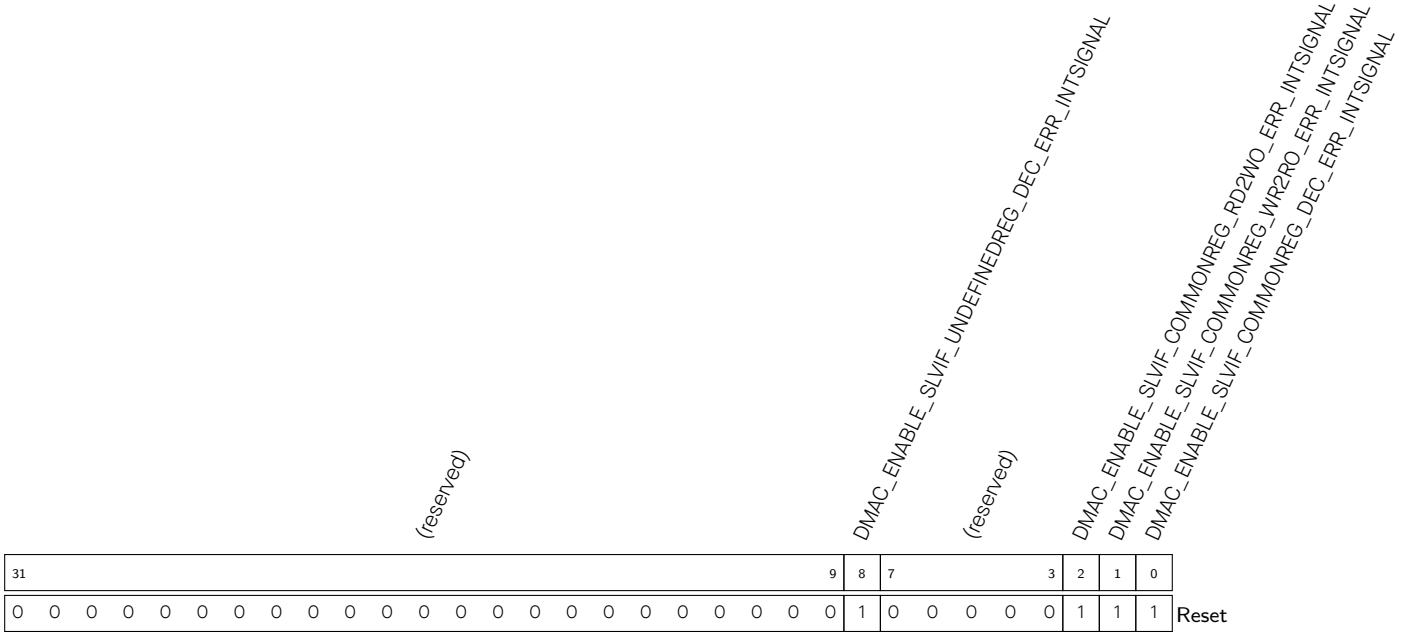
DMAC_ENABLE_SLVIF_COMMONREG_DEC_ERR_INTSTAT Write 1 to enable DMAC_SLVIF_COMMONREG_DEC_ERR_INTSTAT. (R/W)

DMAC_ENABLE_SLVIF_COMMONREG_WR2RO_ERR_INTSTAT Write 1 to enable DMAC_SLVIF_COMMONREG_WR2RO_ERR_INTSTAT. (R/W)

DMAC_ENABLE_SLVIF_COMMONREG_RD2WO_ERR_INTSTAT Write 1 to enable DMAC_SLVIF_COMMONREG_RD2WO_ERR_INTSTAT. (R/W)

DMAC_ENABLE_SLVIF_UNDEFINEDREG_DEC_ERR_INTSTAT Write 1 to enable DMAC_SLVIF_UNDEFINEDREG_DEC_ERR_INTSTAT. (R/W)

Register 7.21. DMAC_COMMONREG_INTSIGNAL_ENABLEO_REG (0x0048)



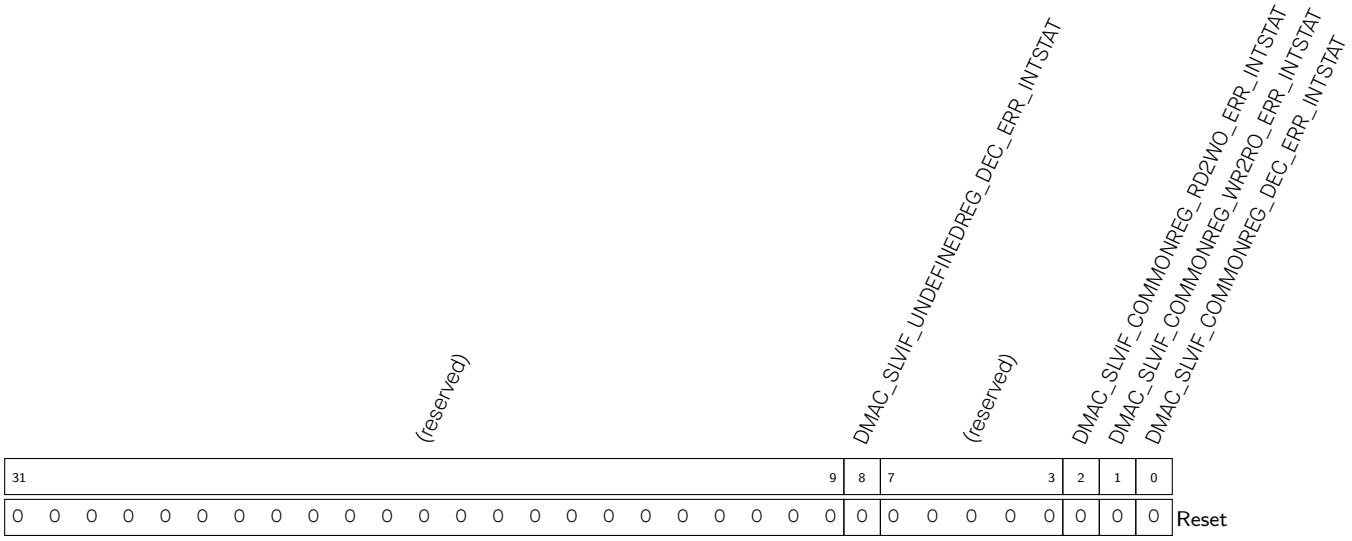
DMAC_ENABLE_SLVIF_COMMONREG_DEC_ERR_INTSIGNAL Write 1 to enable DMAC_SLVIF_COMMONREG_DEC_ERR_INT at port level. (R/W)

DMAC_ENABLE_SLVIF_COMMONREG_WR2RO_ERR_INTSIGNAL Write 1 to enable DMAC_SLVIF_COMMONREG_WR2RO_ERR_INT at port level. (R/W)

DMAC_ENABLE_SLVIF_COMMONREG_RD2WO_ERR_INTSIGNAL Write 1 to enable DMAC_SLVIF_COMMONREG_RD2WO_ERR_INT at port level. (R/W)

DMAC_ENABLE_SLVIF_UNDEFINEDREG_DEC_ERR_INTSIGNAL Write 1 to enable DMAC_SLVIF_UNDEFINEDREG_DEC_ERR_INT at port level. (R/W)

Register 7.22. DMAC_COMMONREG_INTSTATUS0_REG (0x0050)



DMAC_SLVIF_COMMONREG_DEC_ERR_INTSTAT Slave interface common register decode error interrupt status bit. This error occurs when VDMA accesses an invalid address in the common register space (0x000 to 0x0FF).

0: No error occurs

1: Error occurs

(RO)

DMAC_SLVIF_COMMONREG_WR2RO_ERR_INTSTAT Slave interface common register write to read-only error interrupt status bit. This error occurs when a write operation is performed to a read-only register in the common register space (0x000 to 0x0FF).

0: No error occurs

1: Error occurs

(RO)

DMAC_SLVIF_COMMONREG_RD2WO_ERR_INTSTAT Slave interface common register read to write-only error interrupt status bit. This error occurs when a read operation is performed to a write-only register in the common register space (0x000 to 0x0FF).

0: No error occurs

1: Error occurs

(RO)

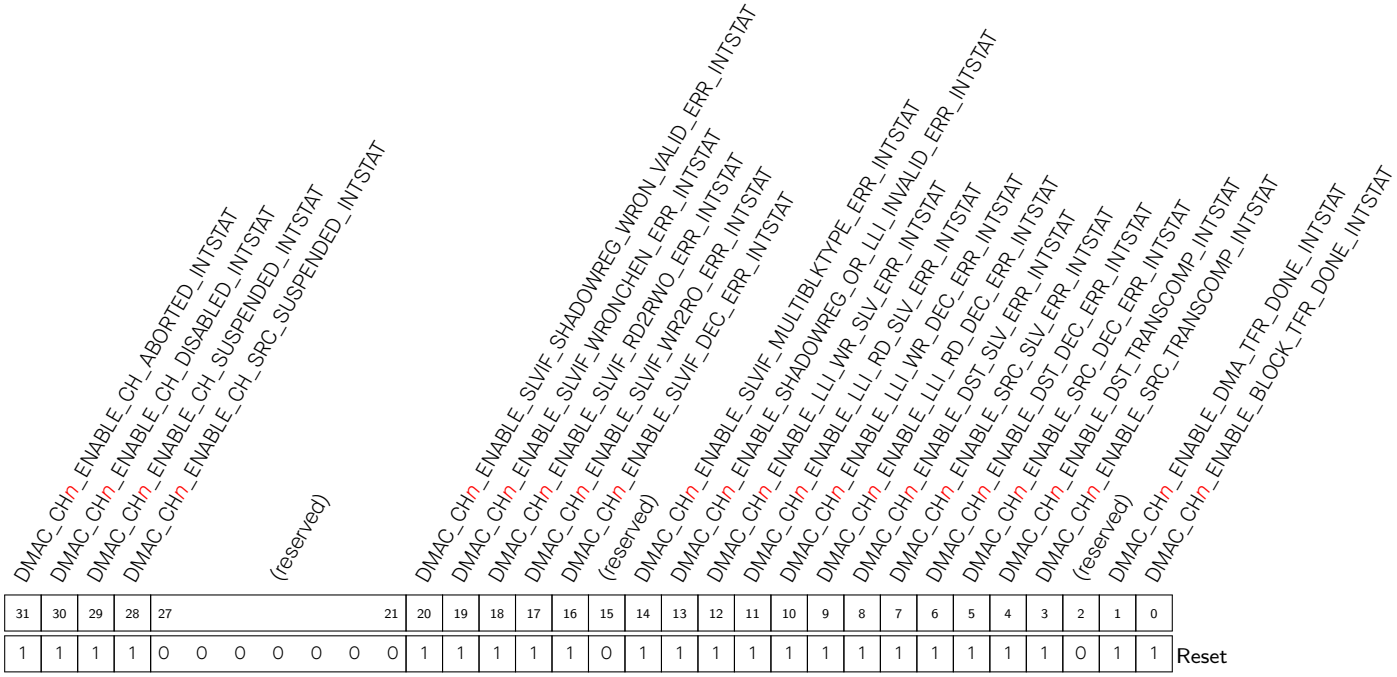
DMAC_SLVIF_UNDEFINEDREG_DEC_ERR_INTSTAT Slave interface undefined register decode error interrupt signal enable bit. This error occurs when the register access is to undefined address range (>0x4FF).

0: No error occurs

1: Error occurs

(RO)

Register 7.23. DMAC_CH*n*_INTSTATUS_ENABLEO_REG (*n*: 1-4) (0x0100*n* + 0x0180)



DMAC_CH*n*_ENABLE_BLOCK_TFR_DONE_INTSTAT Write 1 to enable DMAC_CH*n*_BLOCK_TFR_DONE_INTSTAT. (R/W)

DMAC_CH*n*_ENABLE_DMA_TFR_DONE_INTSTAT Write 1 to enable DMAC_CH*n*_DMA_TFR_DONE_INTSTAT. (R/W)

DMAC_CH*n*_ENABLE_SRC_TRANSCOMP_INTSTAT Write 1 to enable DMAC_CH*n*_SRC_TRANSCOMP_INTSTAT. (R/W)

DMAC_CH*n*_ENABLE_DST_TRANSCOMP_INTSTAT Write 1 to enable DMAC_CH*n*_DST_TRANSCOMP_INTSTAT. (R/W)

DMAC_CH*n*_ENABLE_SRC_DEC_ERR_INTSTAT Write 1 to enable DMAC_CH*n*_SRC_DEC_ERR_INTSTAT. (R/W)

DMAC_CH*n*_ENABLE_DST_DEC_ERR_INTSTAT Write 1 to enable DMAC_CH*n*_DST_DEC_ERR_INTSTAT. (R/W)

DMAC_CH*n*_ENABLE_SRC_SLV_ERR_INTSTAT Write 1 to enable DMAC_CH*n*_SRC_SLV_ERR_INTSTAT. (R/W)

DMAC_CH*n*_ENABLE_DST_SLV_ERR_INTSTAT Write 1 to enable DMAC_CH*n*_DST_SLV_ERR_INTSTAT. (R/W)

DMAC_CH*n*_ENABLE_LLI_RD_DEC_ERR_INTSTAT Write 1 to enable DMAC_CH*n*_LLI_RD_DEC_ERR_INTSTAT. (R/W)

DMAC_CH*n*_ENABLE_LLI_WR_DEC_ERR_INTSTAT Write 1 to enable DMAC_CH*n*_LLI_WR_DEC_ERR_INTSTAT. (R/W)

Continued on the next page...

Register 7.23. DMAC_CH*n*_INTSTATUS_ENABLEO_REG (*n*: 1-4) (0x0100*n* + 0x0180)

Continued from the previous page...

DMAC_CH*n*_ENABLE_LLI_RD_SLV_ERR_INTSTAT Write 1 to enable DMAC_CH*n*_LLI_RD_SLV_ERR_INTSTAT. (R/W)

DMAC_CH*n*_ENABLE_LLI_WR_SLV_ERR_INTSTAT Write 1 to enable DMAC_CH*n*_LLI_WR_SLV_ERR_INTSTAT. (R/W)

DMAC_CH*n*_ENABLE_SHADOWREG_OR_LLI_INVALID_ERR_INTSTAT Write 1 to enable DMAC_CH*n*_SHADOWREG_OR_LLI_INVALID_ERR_INTSTAT. (R/W)

DMAC_CH*n*_ENABLE_SLVIF_MULTIBLKTYPE_ERR_INTSTAT Write 1 to enable DMAC_CH*n*_SLVIF_MULTIBLKTYPE_ERR_INTSTAT. (R/W)

DMAC_CH*n*_ENABLE_SLVIF_DEC_ERR_INTSTAT Write 1 to enable DMAC_CH*n*_SLVIF_DEC_ERR_INTSTAT. (R/W)

DMAC_CH*n*_ENABLE_SLVIF_WR2RO_ERR_INTSTAT Write 1 to enable DMAC_CH*n*_SLVIF_WR2RO_ERR_INTSTAT. (R/W)

DMAC_CH*n*_ENABLE_SLVIF_RD2RWO_ERR_INTSTAT Write 1 to enable DMAC_CH*n*_SLVIF_RD2RWO_ERR_INTSTAT. (R/W)

DMAC_CH*n*_ENABLE_SLVIF_WRONCHEN_ERR_INTSTAT Write 1 to enable DMAC_CH*n*_SLVIF_WRONCHEN_ERR_INTSTAT. (R/W)

DMAC_CH*n*_ENABLE_SLVIF_SHADOWREG_WRON_VALID_ERR_INTSTAT Write 1 to enable DMAC_CH*n*_SLVIF_SHADOWREG_WRON_VALID_ERR_INTSTAT. (R/W)

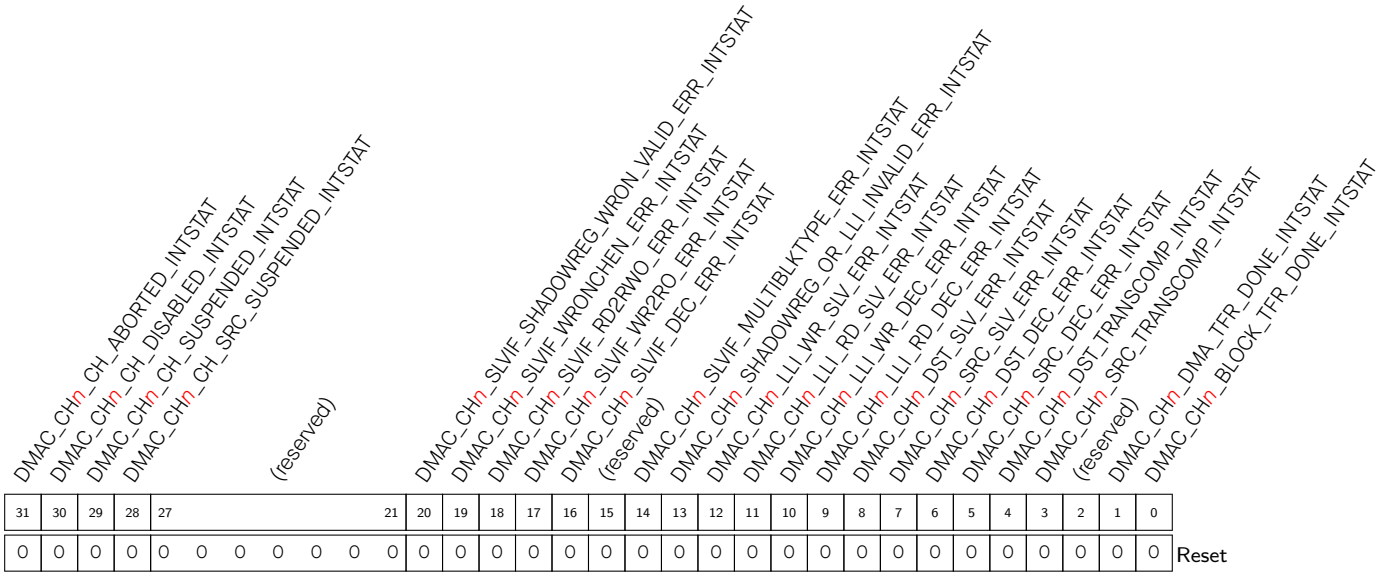
DMAC_CH*n*_ENABLE_CH_SRC_SUSPENDED_INTSTAT Write 1 to enable DMAC_CH*n*_CH_SRC_SUSPENDED_INTSTAT. (R/W)

DMAC_CH*n*_ENABLE_CH_SUSPENDED_INTSTAT Write 1 to enable DMAC_CH*n*_CH_SUSPENDED_INTSTAT. (R/W)

DMAC_CH*n*_ENABLE_CH_DISABLED_INTSTAT Write 1 to enable DMAC_CH*n*_CH_DISABLED_INTSTAT. (R/W)

DMAC_CH*n*_ENABLE_CH_ABORTED_INTSTAT Write 1 to enable DMAC_CH*n*_CH_ABORTED_INTSTAT. (R/W)

Register 7.24. DMAC_CH*n*_INTSTATUSO_REG (*n*: 1-4) (0x0100*n* + 0x0188)



DMAC_CH*n*_BLOCK_TFR_DONE_INTSTAT Block transfer done status bit. (RO)

DMAC_CH*n*_DMA_TFR_DONE_INTSTAT DMA transfer done status bit. (RO)

DMAC_CH*n*_SRC_TRANSCOMP_INTSTAT Source DMA transfer completed status bit. (RO)

DMAC_CH*n*_DST_TRANSCOMP_INTSTAT Destination DMA transfer completed status bit. (RO)

DMAC_CH*n*_SRC_DEC_ERR_INTSTAT Source decode error status bit. The decode error is detected by the master interface during source transfer. (RO)

DMAC_CH*n*_DST_DEC_ERR_INTSTAT Destination decode error status bit. The decode error is detected by the master interface during destination transfer. (RO)

DMAC_CH*n*_SRC_SLV_ERR_INTSTAT Source slave error status bit. The slave error is detected by the master interface during source transfer. This error occurs if the slave interface that reads data issues a slave error. (RO)

DMAC_CH*n*_DST_SLV_ERR_INTSTAT Destination slave error status bit. The slave error is detected by the master interface during source transfer. This error occurs if the slave interface that writes data issues a slave error. (RO)

DMAC_CH*n*_LLI_RD_DEC_ERR_INTSTAT LLI read decode error status bit. The decode error is detected by the master interface during LLI read operation. (RO)

DMAC_CH*n*_LLI_WR_DEC_ERR_INTSTAT LLI write decode error status bit. The decode error is detected by the master interface during LLI write-back operation. (RO)

DMAC_CH*n*_LLI_RD_SLV_ERR_INTSTAT LLI read slave error status bit. The slave error is detected by the master interface during LLI read operation. (RO)

DMAC_CH*n*_LLI_WR_SLV_ERR_INTSTAT LLI write slave error status bit. The slave error is detected by the master interface during LLI write-back operation. (RO)

Continued on the next page...

Register 7.24. DMAC_CH*n*_INTSTATUSO_REG (*n*: 1-4) (0x0100*n* + 0x0188)

Continued from the previous page...

DMAC_CH*n*_SHADOWREG_OR_LLI_INVALID_ERR_INTSTAT Shadow register or LLI invalid error status bit. This error occurs if DMAC_CH*n*_SHADOWREG_OR_LLI_VALID is 0 during VDMA shadow register/LLI fetch phase. (RO)

DMAC_CH*n*_SLVIF_MULTIBLKTYPE_ERR_INTSTAT Slave interface multi-block type error status bit. This error occurs if multi-block transfer type programmed in DMAC_CH*n*_SRC_MULTBLK_TYPE and DMAC_CH*n*_DST_MULTBLK_TYPE is invalid. (RO)

DMAC_CH*n*_SLVIF_DEC_ERR_INTSTAT Slave interface decode error status bit. The decode error is generated by VDMA during register access. This error occurs if the register access is to invalid address in channel *n*. (RO)

DMAC_CH*n*_SLVIF_WR2RO_ERR_INTSTAT Slave interface write to read-only error status bit. This error occurs if write operation is performed to a read-only register. (RO)

DMAC_CH*n*_SLVIF_RD2RWO_ERR_INTSTAT Slave interface read to write-only error status bit. This error occurs if read operation is performed to a write-only register. (RO)

DMAC_CH*n*_SLVIF_WRONCHEN_ERR_INTSTAT Slave interface write to enable channel error status bit. This error occurs if an illegal write operation is performed on a register as per the VDMA specification. (RO)

DMAC_CH*n*_SLVIF_SHADOWREG_WRON_VALID_ERR_INTSTAT Shadow register write on valid error status bit. This error occurs if shadow register based multi-block transfer is enabled and software tries to write to the shadow register when DMAC_CH*n*_SHADOWREG_OR_LLI_VALID bit is 1. (RO)

DMAC_CH*n*_CH_SRC_SUSPENDED_INTSTAT Channel source suspended status bit.

0: Channel source is not suspended.

1: Channel Source is suspended.

(RO)

DMAC_CH*n*_CH_SUSPENDED_INTSTAT Channel suspended status bit.

0: Channel is not suspended.

1: Channel is suspended.

(RO)

DMAC_CH*n*_CH_DISABLED_INTSTAT Channel disabled status bit.

0: Channel is not disabled.

1: Channel is disabled.

(RO)

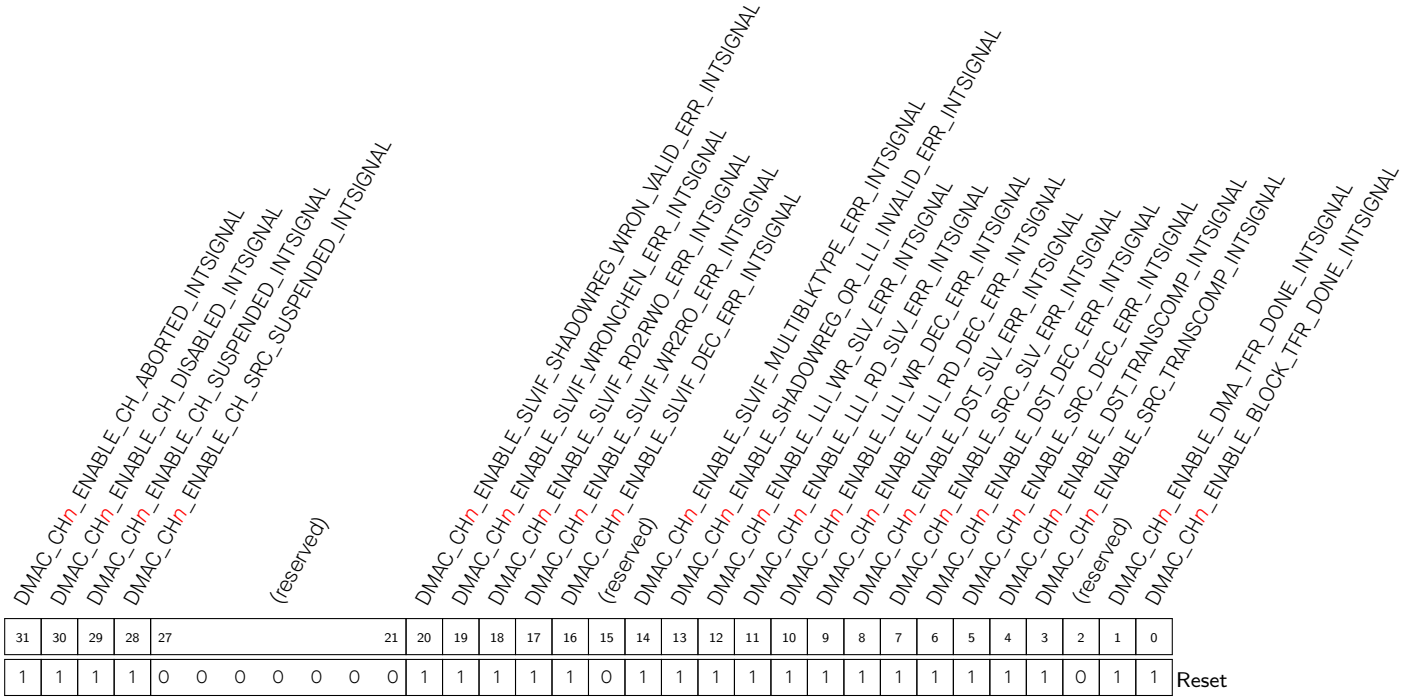
DMAC_CH*n*_CH_ABORTED_INTSTAT Channel aborted status bit.

0: Channel is not aborted.

1: Channel is aborted.

(RO)

Register 7.25. DMAC_CH*n*_INTSIGNAL_ENABLEO_REG (*n*: 1-4) (0x0100*n* + 0x0190)



DMAC_CH*n*_ENABLE_BLOCK_TFR_DONE_INTSIGNAL Write 1 to enable DMAC_CH*n*_BLOCK_TFR_DONE_INT at port level. (R/W)

DMAC_CH*n*_ENABLE_DMA_TFR_DONE_INTSIGNAL Write 1 to enable DMAC_CH*n*_DMA_TFR_DONE_INT at port level. (R/W)

DMAC_CH*n*_ENABLE_SRC_TRANSCOMP_INTSIGNAL Write 1 to enable DMAC_CH*n*_SRC_TRANSCOMP_INT at port level. (R/W)

DMAC_CH*n*_ENABLE_DST_TRANSCOMP_INTSIGNAL Write 1 to enable DMAC_CH*n*_DST_TRANSCOMP_INT at port level. (R/W)

DMAC_CH*n*_ENABLE_SRC_DEC_ERR_INTSIGNAL Write 1 to enable DMAC_CH*n*_SRC_DEC_ERR_INT at port level. (R/W)

DMAC_CH*n*_ENABLE_DST_DEC_ERR_INTSIGNAL Write 1 to enable DMAC_CH*n*_DST_DEC_ERR_INT at port level. (R/W)

DMAC_CH*n*_ENABLE_SRC_SLV_ERR_INTSIGNAL Write 1 to enable DMAC_CH*n*_SRC_SLV_ERR_INT at port level. (R/W)

DMAC_CH*n*_ENABLE_DST_SLV_ERR_INTSIGNAL Write 1 to enable DMAC_CH*n*_DST_SLV_ERR_INT at port level. (R/W)

DMAC_CH*n*_ENABLE_LLI_RD_DEC_ERR_INTSIGNAL Write 1 to enable DMAC_CH*n*_LLI_RD_DEC_ERR_INT at port level. (R/W)

DMAC_CH*n*_ENABLE_LLI_WR_DEC_ERR_INTSIGNAL Write 1 to enable DMAC_CH*n*_LLI_WR_DEC_ERR_INT at port level. (R/W)

Continued on the next page...

Register 7.25. DMAC_CH*n*_INTSIGNAL_ENABLEO_REG (*n*: 1-4) (0x0100*n* + 0x0190)

Continued from the previous page...

DMAC_CH*n*_ENABLE_LLI_RD_SLV_ERR_INTSIGNAL Write 1 to enable DMAC_CH*n*_LLI_RD_SLV_ERR_INT at port level. (R/W)

DMAC_CH*n*_ENABLE_LLI_WR_SLV_ERR_INTSIGNAL Write 1 to enable DMAC_CH*n*_LLI_WR_SLV_ERR_INT at port level. (R/W)

DMAC_CH*n*_ENABLE_SHADOWREG_OR_LLI_INVALID_ERR_INTSIGNAL Write 1 to enable DMAC_CH*n*_SHADOWREG_OR_LLI_INVALID_ERR_INT at port level. (R/W)

DMAC_CH*n*_ENABLE_SLVIF_MULTIBLKTYPE_ERR_INTSIGNAL Write 1 to enable DMAC_CH*n*_SLVIF_MULTIBLKTYPE_ERR_INT at port level. (R/W)

DMAC_CH*n*_ENABLE_SLVIF_DEC_ERR_INTSIGNAL Write 1 to enable DMAC_CH*n*_SLVIF_DEC_ERR_INT at port level. (R/W)

DMAC_CH*n*_ENABLE_SLVIF_WR2RO_ERR_INTSIGNAL Write 1 to enable DMAC_CH*n*_SLVIF_WR2RO_ERR_INT at port level. (R/W)

DMAC_CH*n*_ENABLE_SLVIF_RD2RWO_ERR_INTSIGNAL Write 1 to enable DMAC_CH*n*_SLVIF_RD2RWO_ERR_INT at port level. (R/W)

DMAC_CH*n*_ENABLE_SLVIF_WRONCHEN_ERR_INTSIGNAL Write 1 to enable DMAC_CH*n*_SLVIF_WRONCHEN_ERR_INT at port level. (R/W)

DMAC_CH*n*_ENABLE_SLVIF_SHADOWREG_WRON_VALID_ERR_INTSIGNAL Write 1 to enable DMAC_CH*n*_SLVIF_SHADOWREG_WRON_VALID_ERR_INT at port level. (R/W)

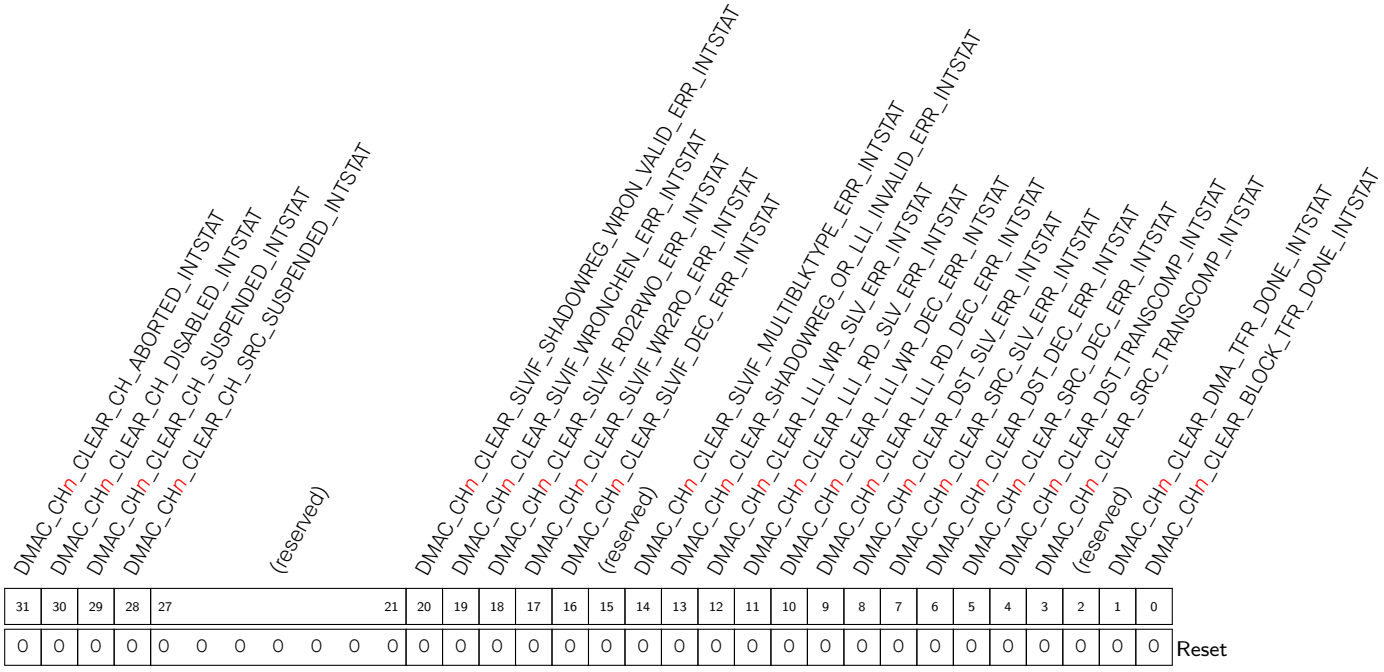
DMAC_CH*n*_ENABLE_CH_SRC_SUSPENDED_INTSIGNAL Write 1 to enable DMAC_CH*n*_CH_SRC_SUSPENDED_INT at port level. (R/W)

DMAC_CH*n*_ENABLE_CH_SUSPENDED_INTSIGNAL Write 1 to enable DMAC_CH*n*_CH_SUSPENDED_INT at port level. (R/W)

DMAC_CH*n*_ENABLE_CH_DISABLED_INTSIGNAL Write 1 to enable DMAC_CH*n*_CH_DISABLED_INT at port level. (R/W)

DMAC_CH*n*_ENABLE_CH_ABORTED_INTSIGNAL Write 1 to enable DMAC_CH*n*_CH_ABORTED_INT at port level. (R/W)

Register 7.26. DMAC_CH*n*_INTCLEAR0_REG (*n*: 1-4) (0x0100*n* + 0x0198)



DMAC_CH*n*_CLEAR_BLOCK_TFR_DONE_INTSTAT Write 1 to clear DMAC_CH*n*_BLOCK_TFR_DONE_INTSTAT. (WO)

DMAC_CH*n*_CLEAR_DMA_TFR_DONE_INTSTAT Write 1 to clear DMAC_CH*n*_DMA_TFR_DONE_INTSTAT. (WO)

DMAC_CH*n*_CLEAR_SRC_TRANSCOMP_INTSTAT Write 1 to clear DMAC_CH*n*_SRC_TRANSCOMP_INTSTAT. (WO)

DMAC_CH*n*_CLEAR_DST_TRANSCOMP_INTSTAT Write 1 to clear DMAC_CH*n*_DST_TRANSCOMP_INTSTAT. (WO)

DMAC_CH*n*_CLEAR_SRC_DEC_ERR_INTSTAT Write 1 to clear DMAC_CH*n*_SRC_DEC_ERR_INTSTAT. (WO)

DMAC_CH*n*_CLEAR_DST_DEC_ERR_INTSTAT Write 1 to clear DMAC_CH*n*_DST_DEC_ERR_INTSTAT. (WO)

DMAC_CH*n*_CLEAR_SRC_SLV_ERR_INTSTAT Write 1 to clear DMAC_CH*n*_SRC_SLV_ERR_INTSTAT. (WO)

DMAC_CH*n*_CLEAR_DST_SLV_ERR_INTSTAT Write 1 to clear DMAC_CH*n*_DST_SLV_ERR_INTSTAT. (WO)

DMAC_CH*n*_CLEAR_LLI_RD_DEC_ERR_INTSTAT Write 1 to clear DMAC_CH*n*_LLI_RD_DEC_ERR_INTSTAT. (WO)

DMAC_CH*n*_CLEAR_LLI_WR_DEC_ERR_INTSTAT Write 1 to clear DMAC_CH*n*_LLI_WR_DEC_ERR_INTSTAT. (WO)

Continued on the next page...

Register 7.26. DMAC_CH*n*_INTCLEAR0_REG (*n*: 1-4) (0x0100**n* + 0x0198)

Continued from the previous page...

DMAC_CH*n*_CLEAR_LLI_RD_SLV_ERR_INTSTAT Write 1 to clear DMAC_CH*n*_LLI_RD_SLV_ERR_INTSTAT. (WO)

DMAC_CH*n*_CLEAR_LLI_WR_SLV_ERR_INTSTAT Write 1 to clear DMAC_CH*n*_LLI_WR_SLV_ERR_INTSTAT. (WO)

DMAC_CH*n*_CLEAR_SHADOWREG_OR_LLI_INVALID_ERR_INTSTAT Write 1 to clear DMAC_CH*n*_SHADOWREG_OR_LLI_INVALID_ERR_INTSTAT. (WO)

DMAC_CH*n*_CLEAR_SLVIF_MULTIBLKTYPE_ERR_INTSTAT Write 1 to clear DMAC_CH*n*_SLVIF_MULTIBLKTYPE_ERR_INTSTAT. (WO)

DMAC_CH*n*_CLEAR_SLVIF_DEC_ERR_INTSTAT Write 1 to clear DMAC_CH*n*_SLVIF_DEC_ERR_INTSTAT. (WO)

DMAC_CH*n*_CLEAR_SLVIF_WR2RO_ERR_INTSTAT Write 1 to clear DMAC_CH*n*_SLVIF_WR2RO_ERR_INTSTAT. (WO)

DMAC_CH*n*_CLEAR_SLVIF_RD2RWO_ERR_INTSTAT Write 1 to clear DMAC_CH*n*_SLVIF_RD2RWO_ERR_INTSTAT. (WO)

DMAC_CH*n*_CLEAR_SLVIF_WRONCHEN_ERR_INTSTAT Write 1 to clear DMAC_CH*n*_SLVIF_WRONCHEN_ERR_INTSTAT. (WO)

DMAC_CH*n*_CLEAR_SLVIF_SHADOWREG_WRON_VALID_ERR_INTSTAT Write 1 to clear DMAC_CH*n*_SLVIF_SHADOWREG_WRON_VALID_ERR_INTSTAT. (WO)

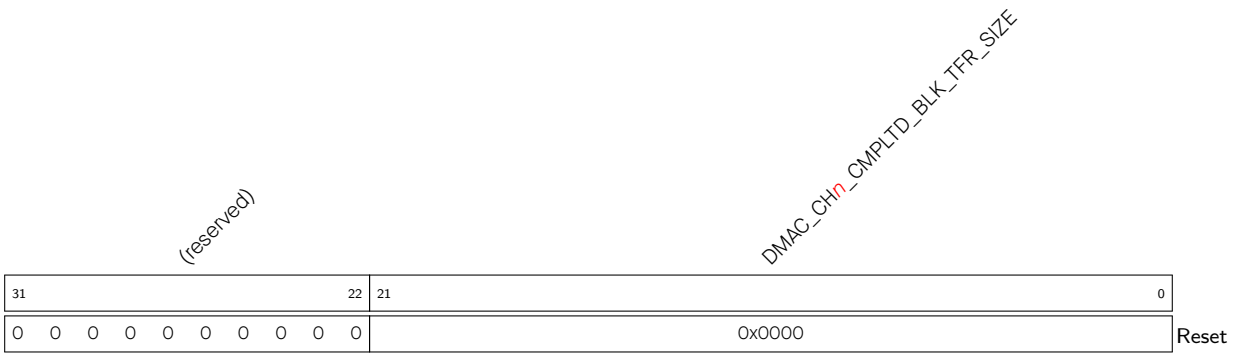
DMAC_CH*n*_CLEAR_CH_SRC_SUSPENDED_INTSTAT Write 1 to clear DMAC_CH*n*_CH_SRC_SUSPENDED_INTSTAT. (WO)

DMAC_CH*n*_CLEAR_CH_SUSPENDED_INTSTAT Write 1 to clear DMAC_CH*n*_CH_SUSPENDED_INTSTAT. (WO)

DMAC_CH*n*_CLEAR_CH_DISABLED_INTSTAT Write 1 to clear DMAC_CH*n*_CH_DISABLED_INTSTAT. (WO)

DMAC_CH*n*_CLEAR_CH_ABORTED_INTSTAT Write 1 to clear DMAC_CH*n*_CH_ABORTED_INTSTAT. (WO)

Register 7.27. DMAC_CH*n*_STATUS0_REG (*n*: 1-4) (0x0100*n* + 0x0130)



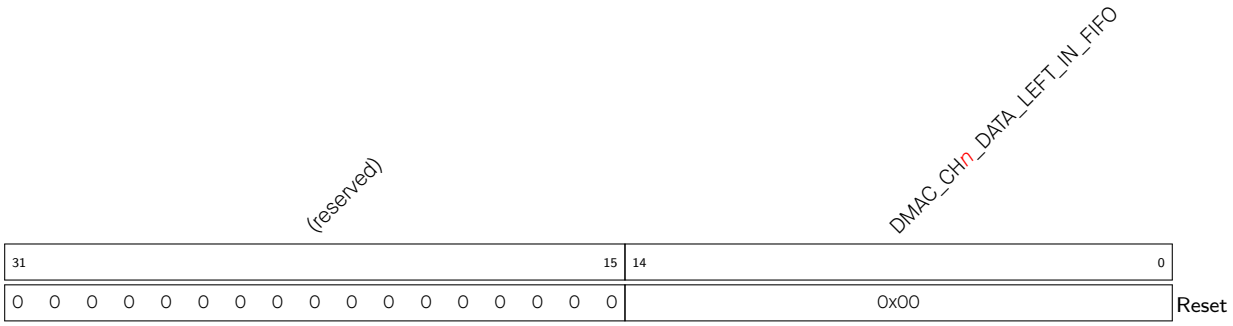
DMAC_CH*n*_CMPLTD_BLK_TFR_SIZE Indicates the total number of data of width DMAC_CH*n*_SRC_TR_WIDTH transferred during the previous block transfer.

If any error occurs during DMA transfer, the block transfer might be terminated early. In such case, the value of this field indicates the actual data transferred without error in the current block.

This field is cleared to 0 when the channel is enabled.

(RO)

Register 7.28. DMAC_CH*n*_STATUS1_REG (*n*: 1-4) (0x0100*n* + 0x0134)



DMAC_CH*n*_DATA_LEFT_IN_FIFO Indicates the total number of remaining data in the VDMA channel FIFO after completing the current block transfer.

The data width in the channel FIFO is DMAC_CH*n*_SRC_TR_WIDTH.

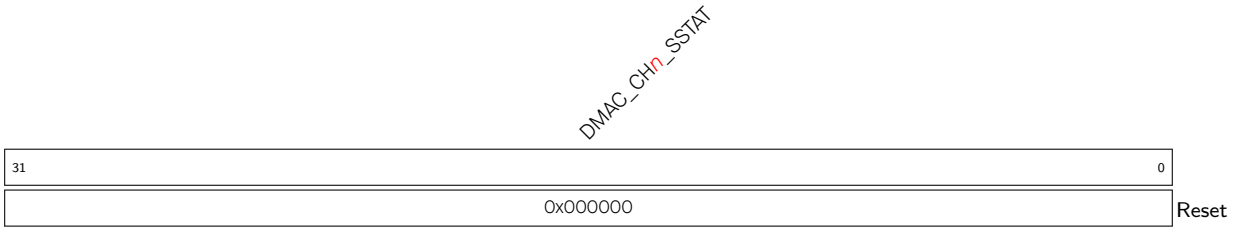
For a normal block transfer completion without errors, the value of this field is 0. If any errors occur during DMA transfer, the block transfer may be terminated early; in this case, this field indicates the remaining data in the channel FIFO that cannot be transferred to the target peripheral.

This field will be cleared when the channel is enabled.

If DMAC_CH*n*_DST_TR_WIDTH > DMAC_CH*n*_SRC_TR_WIDTH, there may be remaining data in the FIFO that is not enough to form a single transfer of DMAC_CH*n*_SRC_TR_WIDTH width; in this case, this field will return 0.

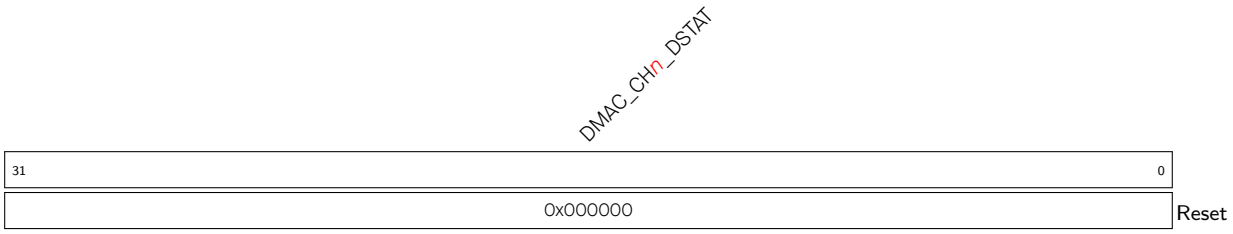
(RO)

Register 7.29. DMAC_CH*n*_SSTAT0_REG (*n*: 1-4) (0x0100**n* + 0x0160)



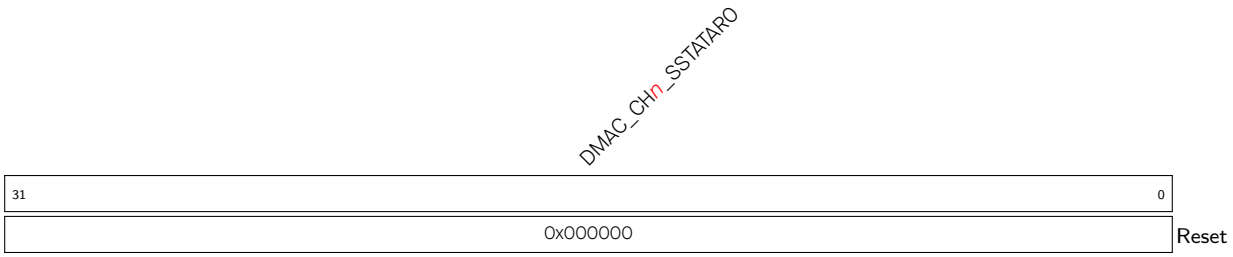
DMAC_CH*n*_SSTAT Indicates the source status information retrieved by hardware from the address pointed to by DMAC_CH*n*_SSTATARO_REG. (RO)

Register 7.30. DMAC_CH*n*_DSTAT0_REG (*n*: 1-4) (0x0100**n* + 0x0168)



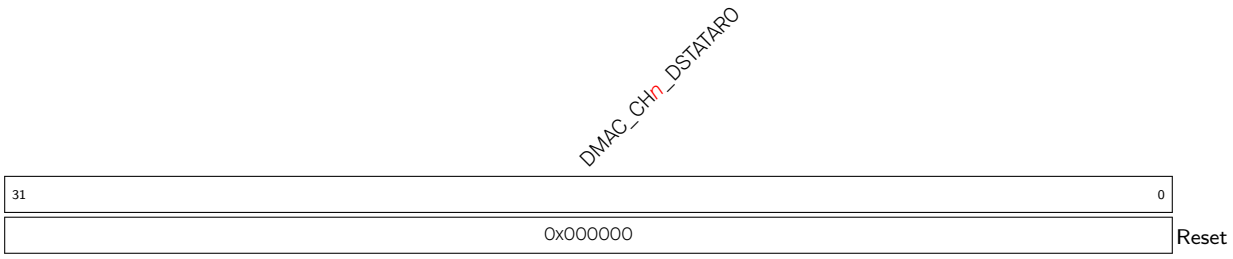
DMAC_CH*n*_DSTAT Indicates the destination status information retrieved by hardware from the address pointed to by DMAC_CH*n*_DSTATARO_REG.
(RO)

Register 7.31. DMAC_CH*n*_SSTATARO_REG (*n*: 1-4) (0x0100**n* + 0x0170)



DMAC_CH*n*_SSTATARO Configures the address from where the hardware obtains the source status information. (R/W)

Register 7.32. DMAC_CH*n*_DSTATARO_REG (*n*: 1-4) (0x0100*n* + 0x0178)



DMAC_CH*n*_DSTATARO Configures the address from where the hardware obtains the destination status information. (R/W)

Chapter 8

2D-DMA Controller (2D-DMA)

8.1 Overview

The 2D-DMA controller is a DMA (Direct Memory Access) dedicated to two-dimensional image processing. In addition to all the features of GDMA-AXI, it includes support for macroblock reordering and color space conversion (CSC) to better meet the data transfer requirements from JPEG and PPA. Notably, the 2D-DMA facilitates memory-to-memory transfers, enabling the movement of macroblocks between different segments of memory address space while concurrently performing color space conversion.

8.2 Features

- One AXI master interface
- Data transfer with unaligned starting addresses
- Memory-to-memory, peripheral-to-memory (RX), and memory-to-peripheral (TX) data transfer
- Four memory-to-peripheral channels, and three peripheral-to-memory channels
- Support for PPA and JPEG Codec
- Macroblock reordering
- Color space conversion
- Configurable channel priority and weight

8.3 Architecture

In ESP32-P4, only JPEG Codec and PPA support 2D-DMA. The 2D-DMA controller and CPU data bus have access to the same address space in memory. Figure 8.3-1 shows the basic architecture of the 2D-DMA controller.

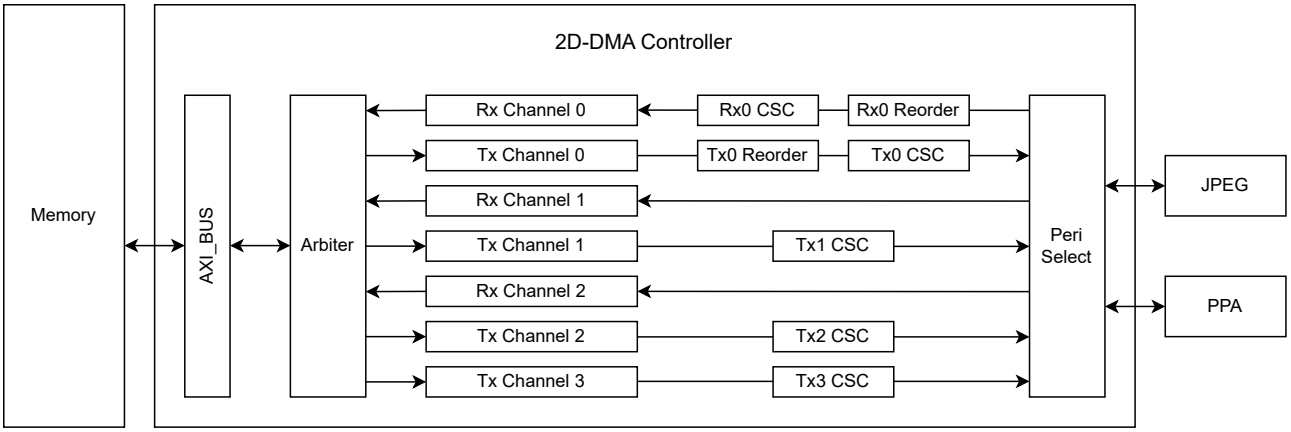


Figure 8.3-1. 2D-DMA Architecture

The 2D-DMA has seven independent channels: four transmit channels and three receive channels. Every channel can be connected to different peripherals, or in other words shared by peripherals.

The macroblock reordering and color space conversion features of the 2D-DMA are not supported by all channels. Among the channels, transmit channel 0 and receive channel 0 support macroblock reordering, while transmit channel 0 ~ 3 and receive channel 0 support color space conversion.

The 2D-DMA reads data from or writes data to the internal or external memory via AXI_BUS. Before the data transfer, the 2D-DMA uses configurable arbitration schemes for channels requesting read or write access. For the available address range of internal and external memory, please see Chapter 9 *System and Memory*.

The software can use the 2D-DMA through linked lists stored in internal or external memory. These linked lists consist of outlink*n* and inlink*n*, where *n* indicates the channel number. For outlink*n*, *n* ranges from 0 to 3; for inlink*n*, *n* ranges from 0 to 2. The 2D-DMA reads an outlink*n* (i.e., a linked list of transmit descriptors) from memory and transmit data in corresponding memory according to the outlink*n*, or read an inlink*n* (i.e., a linked list of receive descriptors) and store received data into specific address space in memory according to the inlink*n*.

8.4 Functional Description

8.4.1 Transfer Mode

The 2D-DMA supports four transfer modes:

- 1D mode: enabled by setting the 2DEN field of DW0 and the mod field of DW2 to 0. In this mode, the 2D-DMA has the same functionalities as GDMA-AXI: it performs memory read/write operations based on the data length and memory space configured in the linked list, and the operation addresses are contiguous.
- 2D-MOD0 mode: enabled by setting the 2DEN field of DW0 to 1 and the mod field of DW2 to 0. In this mode, the 2D-DMA reads or writes a hb*vb macroblock starting from the (X, Y) coordinates of the HA*VA image. After the last data in each row has been read or written, the address jumps to the address of the first data in the next row to be read or written.

- 2D-MOD1 mode: enabled by setting the 2DEN field of DW0 to 1 and the mod field of DW2 to 1. In this mode, the 2D-DMA reads or writes multiple macroblocks from the starting address of the HA*VA image until the HA*VA image has been fully read or written. For JPEG, it is necessary to configure DMA2D_IN_MACRO_BLOCK_SIZE_CH*n* or DMA2D_OUT_MACRO_BLOCK_SIZE_CH*n* to choose a macroblock size out of the three available options (horizontal × vertical): 8 pixels × 8 pixels, 8 pixels × 16 pixels, and 16 pixels × 16 pixels. For more information about macroblock size, see Chapter 37 *JPEG Codec*. For correspondence between macroblock size, hb, and vb of descriptors, see Table 8.4-4 and Table 8.4-5.
- DSCR-PORT mode: enabled by setting the 2DEN field of DW0 to 1, the mod field of DW2 to 0, and the DMA2D_OUT_DSCR_PORT_EN_CH*n* bit or the DMA2D_IN_DSCR_PORT_EN_CH*n* bit to 1. This mode is specifically designed for PPA. For details, see Chapter 39 *Pixel-Processing Accelerator (PPA)*.

1D Mode, 2D-MOD0 Mode, 2D-MOD1 Mode, and DSCR-PORT support both peripheral-to-memory and memory-to-peripheral data transfers. Only 1D Mode and 2D-MOD0 Mode support memory-to-memory data transfers.

8.4.2 Linked List

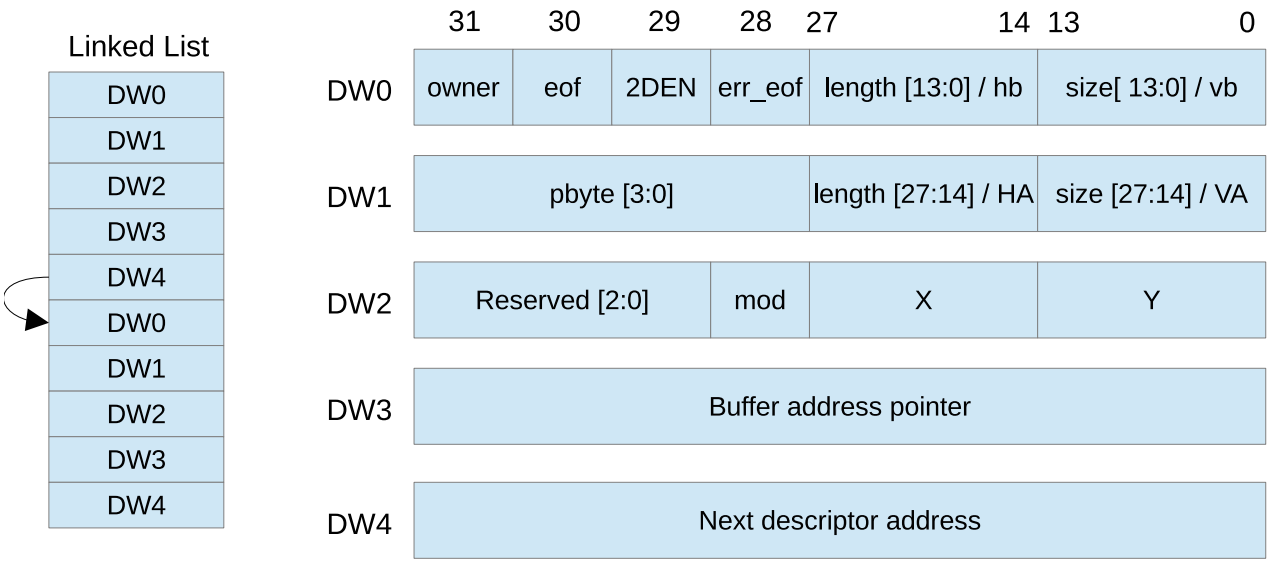


Figure 8.4-1. Structure of a Linked List

Figure 8.4-1 shows the structure of a linked list. An outlink and an inlink have the same structure. A linked list is formed by one or more descriptors, and each descriptor consists of five words. Linked lists should be stored in the memory for the 2D-DMA to be able to use them. The meanings of a descriptor's fields are shown in Table 8.4-1.

Note: DMA2D_OUT is the prefix of transmit channel registers, and DMA2D_IN is the prefix of receive channel registers.

Table 8.4-1. Descriptor Field Description

Bits	Name	Description
DW0[31]	owner	Specifies who is allowed to access the buffer that this descriptor points to. • 0: CPU can access the buffer • 1: The 2D-DMA controller can access the buffer When the 2D-DMA controller stops using the buffer, this bit in a receive descriptor is automatically cleared by hardware, while this bit in a transmit descriptor can only be automatically cleared by hardware if DMA2D_OUT_AUTO_WRBACK_CH*n* is set to 1. When software loads a linked list, this bit should be set to 1.
DW0[30]	eof	Specifies the end of frame. • 0: The current descriptor is not the last descriptor of an image • 1: The current descriptor is the last descriptor of an image For receive descriptors, software needs to write 0 to this bit first, and hardware would set this bit to 1 after receiving a complete image. For transmit descriptors, software should set this bit in the last descriptor of an image to 1.
DW0[29]	2DEN	Specifies whether to enable 2D functions. • 0: 2D-DMA operates in 1D mode • 1: 2D-DMA operates in 2D mode
DW0[28]	err_eof	Specifies whether the received data has errors (JPEG only). • 0: The received data does not have errors • 1: The received data has errors For receive descriptors, the hardware sets this bit to 1 after receiving a complete image and detecting errors in the received data.
DW0[27:14]	length[13:0]/hb	**hb** in 2D mode – Specifies the horizontal width of the macroblocks to be moved in the unit of pixels. **length[13:0]** in 1D mode – Specifies the lower 14 bits of length, which is the number of valid bytes in the buffer that this descriptor points to. This field in a transmit descriptor is written by software and indicates how many bytes can be read from the buffer; this field in a receive descriptor is written by hardware automatically and indicates how many valid bytes have been stored in the buffer.
DW0[13:0]	size[13:0]/vb	**vb** in 2D mode – Specifies the vertical height of the macroblocks to be moved in the unit of pixels. **size[13:0]** in 1D mode – Specifies the lower 14 bits of size, which is the capacity of the buffer that this descriptor points to in the unit of bytes.

Bits	Name	Description
DW1[31:28]	pbyte	In 2D mode, specifies the number of bytes per pixel of the image to be transferred to/from the memory. (If color space conversion is applied, this field represents the number of bytes per pixel of the image stored in the memory.) • 0: 0.5 byte/pixel • 1: 1 byte/pixel • 2: 1.5 bytes/pixel • 3: 2 bytes/pixel • 4: 3 bytes/pixel • 5: 4 bytes/pixel In 1D mode, this field is not used and can be any value.
DW1[27:14]	length[27:14]/HA	**HA** in 2D mode – Specifies the horizontal width of the original image corresponding to the macroblocks to be moved in the unit of pixels. **length[27:14]** in 1D mode – Specifies the higher 14 bits of length.
DW1[13:0]	size[27:14]/VA	**VA** in 2D mode – Specifies the vertical height of the original image corresponding to the macroblocks to be moved in the unit of pixels. **size[27:14]** in 1D mode – Specifies the higher 14 bits of size.
DW2[31:29]	Reserved	Reserved. Irrelevant.
DW2[28]	mod	Specifies the macroblock read mode. • 0: 2D-DMA reads or writes a single macroblock with horizontal width hb and vertical height vb only once • 1: 2D-DMA reads or writes multiple macroblocks with horizontal width hb and vertical height vb until the image data with horizontal width HA and vertical height VA is fully read or written In 1D mode, this bit must be 0. **Note:** hb/vb/HA/VA are in the unit of pixels.
DW2[27:14]	X	In 2D mode, specifies the X coordinate of the hb*vb image block within the HA*VA image in the unit of pixels. In 1D mode, this field is not used and can be any value.
DW2[13:0]	Y	In 2D mode, specifies the Y coordinate of the hb*vb image block within the HA*VA image in the unit of pixels. In 1D mode, this field is not used and can be any value.
DW3	Buffer address pointer	In 2D mode, specifies the address of the HA*VA image block origin in the buffer (the address of the original image's first byte in the memory). In 1D mode, specifies the starting address of the buffer.

Bits	Name	Description
DW4	Next descriptor address	Address of the next descriptor. If the current descriptor is the last descriptor in the linked list, this value can be 0. This address can point to the address space of internal or external memory. **Note:** The starting address for storing 2D-DMA descriptors must be 8-byte aligned.

8.4.3 Padding in 2D-MOD1 Mode and DSCR-PORT Mode

This section only applies to the TX direction of 2D-DMA transfers in 2D-MOD1 and DSCR-PORT mode.

For JPEG decoding, the basic unit of image data input is a block of 8 × 8, 8 × 16, or 16 × 16 (pixels). In 2D-MOD1 mode, when the length and width of the image are not integer multiples of the basic unit, the image needs to pad out to integer multiples of the basic unit based on the previous or the current macroblock.

For PPA, the basic unit for transmit channels in DSCR-PORT mode is 36 × 36 (YUV420), 36 × 34 (YUV422) and 34 × 34 (horizontal × vertical). The width and height of a basic unit for different transmit channels are configured using DMA2D_OUT_DSCR_PORT_BLK_H_CH*n* and DMA2D_OUT_DSCR_PORT_BLK_V_CH*n*. When the length and width of the image are not integer multiples of the basic unit, the image needs to pad out.

Below are the padding rules rules:

- Horizontally, the padding depends on the pbyte field of DW1.
  - When pbyte is 0 (0.5 byte/pixel), the remaining pixels must be even, and the padding is the last 0.5 byte.
  - When pbyte is 1 (1 byte/pixel), the padding is the last 1 byte.
  - When pbyte is 2 (1.5 bytes/pixel), the remaining pixels must be even, and the padding is the last 3 bytes.
  - When pbyte is 3 (2 bytes/pixel), the padding is the last 2 bytes.
  - When pbyte is 4 (3 bytes/pixel), the padding is the last 3 bytes.
  - When pbyte is 5 (4 bytes/pixel), the padding is the last 4 bytes.
- Vertically, the padding is the last row.

Figure 8.4-2 and Figure 8.4-3 illustrate the padding method.

In Figure 8.4-2, a blue grid represents a basic unit (a block of 8 × 8, 8 × 16, or 16 × 16 pixels), and a yellow box (incomplete basic unit) along with a green box (padding) forms a basic unit.

In Figure 8.4-3, a grid represents one pixel, and the numbers inside each grid represent pixel values.

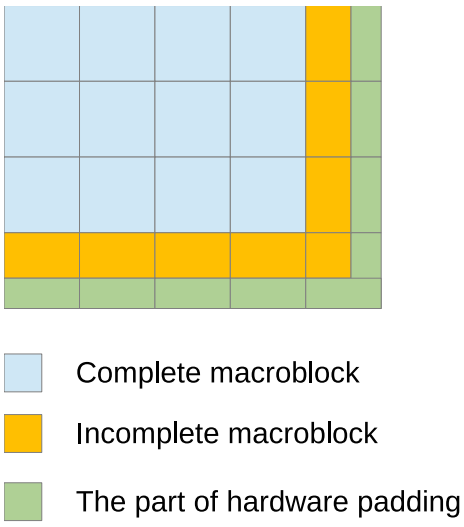


Figure 8.4-2. Padding Illustration (Overall)

1	2	3	3	3	3	3	3
4	5	6	6	6	6	6	6
7	8	9	9	9	9	9	9
7	8	9	9	9	9	9	9
7	8	9	9	9	9	9	9
7	8	9	9	9	9	9	9
7	8	9	9	9	9	9	9
7	8	9	9	9	9	9	9

Figure 8.4-3. Padding Illustration (Detail)

8.4.4 Peripheral-to-Memory and Memory-to-Peripheral Data Transfer

When DMA2D_MEM_TRANS_EN_CH*n* is 0, the 2D-DMA controller enters peripheral transfer mode. The transmit channel reads data from the memory and transmits it to peripherals, while the receive channel receives data from peripherals and stores it into the memory.

The transmit direction refers to data transfer from memory to peripheral, and the receive direction refers data transfer from peripheral to memory. A transmit channel transfers data in the specified memory location to a peripheral's transmitter via an outlink*n*, whereas a receive channel transfers data received by a peripheral to the specified memory location via an inlink*n*.

Every transmit and receive channel can be connected to any peripheral that supports 2D-DMA. Table 8.4-2 and Table 8.4-3 illustrate how to select the peripheral to be connected via registers. "Dummy-*n*" corresponds to register values for memory-to-memory data transfer. When a channel is connected to a peripheral, the rest

of the channels cannot be connected to that peripheral.

Table 8.4-2. Selecting Peripherals via Register Configuration in TX Direction

DMA2D_PERI_OUT_SEL_CH*n*	Peripheral
0	JPEG
1	PPA's SRM module
2	PPA's BLEND0 module
3	PPA's BLEND1 module
4 ~ 7	Dummy-4 ~ 7

Table 8.4-3. Selecting Peripherals via Register Configuration in RX Direction

DMA2D_PERI_IN_SEL_CH*n*	Peripheral
0	JPEG
1	PPA's SRM module
2	PPA's BLEND channel
3 ~ 7	Dummy-3 ~ 7

8.4.5 Memory-to-Memory Data Transfer

The 2D-DMA controller allows memory-to-memory data transfer only in 1D mode and 2D-MOD0 mode. Such data transfer can be enabled by setting DMA2D_IN_MEM_TRANS_EN_CH*n*, which connects the output of transmit channel *n* to the input of receive channel *n*. Note that a transmit channel can only be connected to the receive channel with the same number (*n*), and DMA2D_IN_PERI_SEL_CH*n* and DMA2D_OUT_PERI_SEL_CH*n* should be configured to any value corresponding to "Dummy".

Memory-to-memory data transfer can be used in combination with color space conversion so that a macroblock can be moved from one segment of memory address space to another with its color space converted.

8.4.6 Macroblock Reordering

8.4.6.1 Macroblock Reordering in TX Direction

When the 2D-DMA accesses external memory via SPI, in order to improve the bandwidth utilization of SPI, it is necessary to initiate AXI transfers with a larger burst length. In 2D-MOD1 mode, the 2D-DMA can increase the burst length of AXI transfers by fetching multiple macroblocks at once. For example, the 2D-DMA can fetch five macroblocks of 8x8 pixels in one operation, and in this case, the hb field in the descriptor should be 40 and vb should be 8. However, the JPEG requires the 2D-DMA to send data in the order of macroblocks. This means that the 2D-DMA needs to reorder the fetched macroblocks to meet the requirements of JPEG.

Recommended configurations for macroblock reordering are listed in Table 8.4-4.

Table 8.4-4. Recommended Configurations for Macroblock Reordering in TX Direction

Byte/Pixel	Macroblock Size (Pixel)	Optimal Macroblock Count	hb	vb
3	8× 8	5	40	8
3	16 × 8	2	32	8
3	16 × 16	2	32	16
2	8 × 8	8	64	8
2	16 × 8	4	64	8
2	16 × 16	3	48	16
1	8 × 8	16	128	8

The macroblock size in TX direction should be configured via the DMA2D_OUT_MACRO_BLOCK_SIZE_CH*n* field.

8.4.6.2 Macroblock Reordering in RX Direction

Currently, in RX direction the 2D-DMA can only reorder the data decoded by JPEG (the bitstream is 1 scan, see Chapter 37 *JPEG Codec*). The 2D-DMA breaks down color component data decoded by JPEG into pixels, assembles them into larger macroblocks, and writes these blocks into the memory.

The data formats decoded by JPEG include GRAY, YUV444, YUV422, and YUV420. Recommended configurations for macroblock reordering are listed in Table 8.4-5.

Table 8.4-5. Recommended Configuration Table for Macroblock Reordering in 2D-DMA Receive Direction (Without Color Format Conversion)

Format	Macroblock Size (pixels)	Optimal Macroblock Count	hb	vb
GRAY	8 × 8	12	96	8
YUV444	8 × 8	5	40	8
YUV422	16 × 8	4	64	8
YUV420	16 × 16	3	48	16

The macroblock size in TX direction should be configured via the DMA2D_IN_MACRO_BLOCK_SIZE_CH*n* field.

Figure 8.4-4 shows the layout of macroblocks outputted by the JPEG decoder. Each blue grid in this figure represents 8x8 bytes of continuous data, and for each color format the figure displays four macroblocks of 8 × 8 pixels.

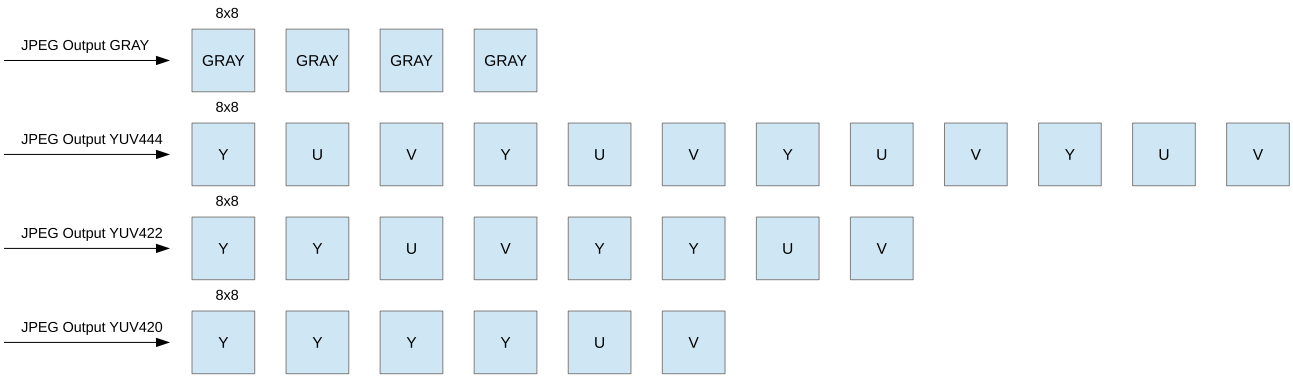


Figure 8.4-4. Macroblock Layout of JPEG Decoder Output Data

8.4.7 Color Space Conversion

On the transmit side, the 2D-DMA converts the color space after assembling the data read from memory into pixels. The 2D-DMA supports the following conversion except YUV420:

- RGB888 to RGB565
- RGB565 to RGB888
- RGB888 to YUV444
- RGB888 to YUV422 (the number of pixels per row in RGB format must be even)
- YUV444 to RGB888
- YUV422 to RGB888

On the receive side, only pixels received from JPEG requires color space conversion. The 2D-DMA supports the following conversion:

- YUV444 to RGB888/RGB565/YUV422/YUV420
- YUV422 to YUV444 to RGB888/RGB565/YUV420
- YUV420 to YUV444 to RGB888/RGB565

Note:

- For JPEG decoding, i.e., the receive side of 2D-DMA, the hardware will forcibly convert YUV422 and YUV420 to the YUV444 format. Software can optionally further convert the format to RGB888/RGB565 with the color space conversion feature.

Figure 8.4-5 shows the structure of the color space conversion module.

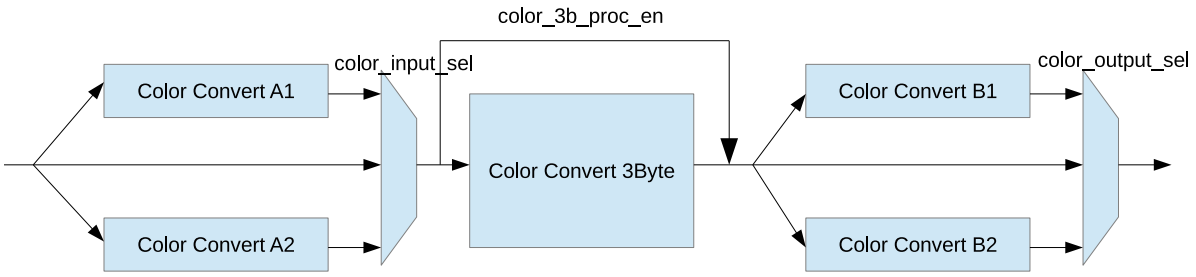


Figure 8.4-5. Color Space Conversion

Color space conversion is performed in 3 stages. The first stage converts a specific 16-bit color format (2 bytes/pixel) to a 24-bit color format (3 bytes/pixel). The source color formats can be configured via the DMA2D_OUT_COLOR_INPUT_SEL_CH*n* or DMA2D_IN_COLOR_INPUT_SEL_CH0 field. (See the supported color format conversion in the first stage by clicking the fields)

The second stage further converts the output data of the first stage using the following formula:

$$256 * Q = A[9:0] * x + B[10:0] * y + C[9:0] * z + D[17:0]$$

In this formula, Q is the output data; x, y, and z are the most significant, middle, and the least significant byte of the input data respectively; A, B, C, and D are the corresponding multiplication or addition coefficients, which are signed numbers. At most 3 bytes of data can be processed in one go, and therefore 3 * 4 = 12 coefficients need to be configured via the following fields:

- DMA2D_OUT_COLOR_PARAM_H0_CH*n*/DMA2D_IN_COLOR_PARAM_H0_CH0
- DMA2D_OUT_COLOR_PARAM_H1_CH*n*/DMA2D_IN_COLOR_PARAM_H1_CH0
- DMA2D_OUT_COLOR_PARAM_M0_CH*n*/DMA2D_IN_COLOR_PARAM_M0_CH0
- DMA2D_OUT_COLOR_PARAM_M1_CH*n*/DMA2D_IN_COLOR_PARAM_M1_CH0
- DMA2D_OUT_COLOR_PARAM_L0_CH*n*/DMA2D_IN_COLOR_PARAM_L0_CH0
- DMA2D_OUT_COLOR_PARAM_L1_CH*n*/DMA2D_IN_COLOR_PARAM_L1_CH0

PARAM_H*x*, PARAM_M*x*, and PARAM_L*x* (*x* is 0 or 1) in field names represent the parameters for the three bytes of data respectively. Take PARAM_H*x* as examples, the lower 10 bits of the 21-bit PARAM_H0 is coefficient A, the higher 11 bits of PARAM_H0 is coefficient B, the lower 10 bits of the 28-bit PARAM_H1 is coefficient C, and the higher 18 bits of PARAM_H1 is coefficient D.

The third stage, similar to the first stage, outputs the color format converted in the second stage directly, or further converts the 24-bit color format (3 bytes/pixel) to a 16-bit color format. This is configured using DMA2D_OUT_COLOR_OUTPUT_SEL_CH*n* or DMA2D_IN_COLOR_OUTPUT_SEL_CH0.

The following table lists the parameter configurations for commonly used color space conversions:

Table 8.4-6. Parameter Configuration for Color Space Conversion in TX Direction

Conversion	input_sel	proc_en	output_sel	param_h0	param_h1	param_m0	param_m1	param_l0	param_l1
No conversion	7	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
Only scramble order	2/3	0	2	N/A	N/A	N/A	N/A	N/A	N/A

Cont'd on next page

Table 8.4-6 – cont'd from previous page

Conversion	input_sel	proc_en	output_sel	param_h0	param_h1	param_m0	param_m1	param_l0	param_l1
RGB888 → RGB565	3	0	0	N/A	N/A	N/A	N/A	N/A	N/A
RGB565 → RGB888	0	0	2	N/A	N/A	N/A	N/A	N/A	N/A
RGB888 → (601) YUV444	3	1	2	20442	400019	1EDBDA	2000070	1E8870	20003EE
RGB888 → (709) YUV444	3	1	2	2742F	400010	1EABE6	2000070	1E6870	20003F6
RGB888 → (601) YUV422-MIPI	3	1	1	20442	400019	1EDBDA	2000070	1E8870	20003EE
RGB888 → (709) YUV422-MIPI	3	1	1	2742F	400010	1EABE6	2000070	1E6870	20003F6
YUV444 → (601) RGB888	3	1	2	12A	C86D999	1E712A	21E4F30	8112A	BAD3000
YUV444 → (709) RGB888	3	1	2	12A	C21E5CB	1F252A	1338778	8752A	B7D0800
YUV422-MIPI → (601) RGB888	1	1	2	12A	C86D999	1E712A	21E4F30	8112A	BAD3000
YUV422-MIPI → (709) RGB888	1	1	2	12A	C21E5CB	1F252A	1338778	8752A	B7D0800

Table 8.4-7. Parameter Configuration for Color Space Conversion in RX Direction

Conversion	input_sel	proc_en	output_sel	param_h0	param_h1	param_m0	param_m1	param_l0	param_l1
No conversion	7	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
Only scramble order	1/2	0	1	N/A	N/A	N/A	N/A	N/A	N/A
YUV422 → YUV444	0	0	1	N/A	N/A	N/A	N/A	N/A	N/A
YUV420 → YUV444	0	0	1	N/A	N/A	N/A	N/A	N/A	N/A
YUV444 → YUV422	0	0	2	N/A	N/A	N/A	N/A	N/A	N/A
YUV444 → YUV420	0	0	3	N/A	N/A	N/A	N/A	N/A	N/A
YUV422 → YUV420	0	0	3	N/A	N/A	N/A	N/A	N/A	N/A
YUV420 → (601) RGB888/RGB565	0	1	1/0	12A	C86D999	1E712A	21E4F30	8112A	BAD3000
YUV422 → (601) RGB888/RGB565	0	1	1/0	12A	C86D999	1E712A	21E4F30	8112A	BAD3000
YUV444 → (601) RGB888/RGB565	2	1	1/0	12A	C86D999	1E712A	21E4F30	8112A	BAD3000
YUV420 → (709) RGB888/RGB565	0	1	1/0	12A	C21E5CB	1F252A	1338778	8752A	B7D0800

Cont'd on next page

Table 8.4-7 – cont'd from previous page

Conversion	input_sel	proc_en	output_sel	param_h0	param_h1	param_m0	param_m1	param_l0	param_l1
YUV422 → (709) RGB888/RGB565	0	1	1/0	12A	C21E5CB	1F252A	1338778	8752A	B7D0800
YUV444 → (709) RGB888/RGB565	2	1	1/0	12A	C21E5CB	1F252A	1338778	8752A	B7D0800

Additionally, the color space conversion module can scramble the order of at most three bytes per pixel. This can be done before the color space conversion in TX direction, and before or after the color space conversion in RX direction. Table 8.4-8, Table 8.4-9, and Table 8.4-10 show the byte order per pixel required by the color space conversion module. For the JPEG decoder, if the byte order per pixel is not scrambled after the macroblock reordering in RX direction, it will conform to the requirements of color space conversion. Assuming that the order of an input pixel is represented from MSB to LSD as 2, 1, 0, the order can be scrambled and outputted as 210 (default), 201, 102, 120, 021, or 012. In RX direction, this feature can be configured before or after the color space conversion via DMA2D_IN_SCRAMBLE_SEL_PRE_CH0 orDMA2D_IN_SCRAMBLE_SEL_POST_CH0 respectively; in TX rection, this feature can be configured before the color space conversion via DMA2D_OUT_SCRAMBLE_SEL_PRE_CH*n*.

Table 8.4-8. RGBB888 Storage Format

	LSB								MSB
LINE	B1	G1	R1	B2	G2	R2	B3	G3	R3

Table 8.4-9. YUV444 Storage Format

	LSB								MSB
LINE	V1	U1	Y1	V2	U2	Y2	V3	U3	Y3

Table 8.4-10. YUV422 Storage Format

	LSB							MSB
LINE	U1	Y1	V1	Y2	U3	Y3	V3	Y4

8.4.8 Enabling 2D-DMA

The software uses the 2D-DMA controller through linked lists. For receive channels, software loads an inlink, configures the DMA2D_INLINK_ADDR_CH*n* field with the address of the first receive descriptor, and sets the DMA2D_INLINK_START_CH*n* bit to enable the 2D-DMA. For transmit channels, software loads an outlink, prepares data to be transmitted, configures the DMA2D_OUTLINK_ADDR_CH*n* field with address of the first transmit descriptor, and sets the DMA2D_OUTLINK_START_CH*n* bit to enable the 2D-DMA. The DMA2D_INLINK_START_CH*n* bit and DMA2D_OUTLINK_START_CH*n* bit are cleared automatically by hardware.

In some cases, you may want to append more descriptors to a 2D-DMA transfer that is already started. Naively, it would seem to be possible to do this by clearing the eof bit of the final descriptor in the existing list and setting its next descriptor address pointer field (DW4) to the first descriptor of the to-be-added list. However, this strategy fails if the existing 2D-DMA transfer is almost or entirely finished. Instead, the 2D-DMA controller

has specialized logic to make sure a transfer can be continued or restarted: if the transfer is ongoing, the controller will make sure to take the appended descriptors into account; if the transfer has already finished, the controller will restart with the new descriptors. This is implemented by the Restart function.

When using the Restart function, software needs to

1. Set DMA2D_INLINK_STOP_CH*n* or DMA2D_OUTLINK_STOP_CH*n* to stop the 2D-DMA when it finishes processing the current descriptor
2. Rewrite the address of the first descriptor in the new list to DW4 of the last descriptor in the loaded list
3. Set DMA2D_INLINK_RESTART_CH*n* or DMA2D_OUTLINK_RESTART_CH*n* (these two bits are cleared automatically by hardware)

By doing so, hardware can obtain the address of the first descriptor in the new list when reading the last descriptor in the loaded list, and then read the new list.

8.4.9 Linked List Reading Process

Once configured and enabled by software, the 2D-DMA controller starts to read the linked list from the memory. The 2D-DMA performs checks on descriptors in the linked list. Only if descriptors pass the checks, the corresponding 2D-DMA channel will start data transfer. If the descriptors fail any of the checks, hardware will trigger descriptor error interrupt (either DMA2D_IN_DSCR_ERR_CH*n*_INT or DMA2D_OUT_DSCR_ERR_CH*n*_INT), and the channel will remain idle. This channel can be restarted by setting DMA2D_OUTLINK_START_CH*n* or DMA2D_INLINK_START_CH*n* again.

The checks performed on descriptors are:

- Owner bit check when DMA2D_IN_CHECK_OWNER_CH*n* or DMA2D_OUT_CHECK_OWNER_CH*n* is set to 1. If the owner bit is 0, the buffer is accessed by the CPU. In this case, the owner bit fails the check. The owner bit will not be checked if DMA2D_IN_CHECK_OWNER_CH*n* or DMA2D_OUT_CHECK_OWNER_CH*n* is 0.
- Descriptor address check, which checks if the descriptor address is located in configured memory space. If a 2D-DMA descriptor points to DMA2D_ACCESS_INTR_MEM_START_ADDR ~ DMA2D_ACCESS_INTR_MEM_END_ADDR (the value should be located in internal memory) or DMA2D_ACCESS_EXTR_MEM_START_ADDR ~ DMA2D_ACCESS_EXTR_MEM_END_ADDR (the value should be located in external memory), it passes the check. For details, please refer to Section 8.4.11).
- HA/VA/hb/vb/SIZE check, which checks if any of these fields is 0. If any field is 0, it fails the check. HA/VA/hb/vb fields are not checked in 1D mode, while SIZE is checked only in 1D mode.

After the software detects a descriptor error interrupt, it must reset the corresponding channel, and restart the 2D-DMA DMA by setting the DMA2D_OUTLINK_START_CH*n* or DMA2D_INLINK_START_CH*n* bit.

To quickly obtain receive or transmit descriptors from the memory, AXI burst transfers for 2D-DMA can be enabled by setting DMA2D_INDSCR_BURST_EN_CH*n* or DMA2D_OUTDSCR_BURST_EN_CH*n*.

8.4.10 EOF

Note: In this chapter, EOF of transmit descriptors refers to eof (i.e., bit 30 of DW0), while EOF of receive descriptors refers to both eof and err_eof (i.e., bit 28 of DW0).

The 2D-DMA controller uses EOF (end of frame) flags to indicate the end of the image transmission.

For data transmission, the 2D-DMA generates two types of EOF interrupts:

- DMA2D_OUT_EOF_CH*n*_INT, generated when the eof bit of any descriptor in the linked list is set, and the data corresponding to this descriptor has been transmitted. Usually, the eof bit is set in the last descriptor for the transmission of an image. This interrupt is enabled by setting the DMA2D_OUT_EOF_CH*n*_INT_ENA bit.
- DMA2D_OUT_TOTAL_EOF_CH*n*_INT, generated when the eof bit of the last descriptor in the linked list is set, and the data corresponding to the last descriptor has been transmitted. This interrupt is enabled by setting the DMA2D_OUT_TOTAL_EOF_CH*n*_INT_ENA bit.

For data reception, the 2D-DMA also generates two types of EOF interrupts:

- DMA2D_IN_SUC_EOF_CH*n*_INT, generated when an image has been received. This interrupt is enabled by setting the DMA2D_IN_SUC_EOF_CH*n*_INT_ENA bit.
- (JPEG only) DMA2D_IN_ERR_CH*n*_EOF_INT, generated when an image has been received with errors. This interrupt is enabled by setting the DMA2D_IN_ERR_EOF_CH*n*_INT_ENA bit, and is valid only when the channel is connected to JPEG.

When detecting an DMA2D_OUT_TOTAL_EOF_CH*n*_INT or DMA2D_IN_SUC_EOF_CH*n*_INT interrupt, software can read the value of the DMA2D_OUT_EOF_DES_ADDR_CH*n* or DMA2D_IN_SUC_EOF_DES_ADDR_CH*n* field, which stores the address of the finished descriptor. Therefore, the software can tell which descriptors have been used and reclaim them as needed. In RX direction, the address of the descriptor can also be stored to the DMA2D_IN_ERR_EOF_DES_ADDR_CH*n* field when an DMA2D_IN_ERR_EOF_CH*n*_INT interrupt is triggered.

8.4.11 Accessing Memory

Any transmit and receive channels of the 2D-DMA can access the internal memory address space configured by DMA2D_ACCESS_INTR_MEM_START_ADDR and DMA2D_ACCESS_INTR_MEM_END_ADDR, and the external memory address space configured by DMA2D_ACCESS_EXTR_MEM_START_ADDR and DMA2D_ACCESS_EXTR_MEM_END_ADDR.

The 2D-DMA can send data in the internal memory in burst mode via the AXI bus. Burst transfer on the AXI bus is enabled by default, and cannot be disabled. Users can configure the number of data bytes in a single AXI burst (burst length) to 8, 16, 32, 64, or 128 bytes via DMA2D_IN_MEM_BURST_LENGTH_CH*n* and DMA2D_OUT_MEM_BURST_LENGTH_CH*n*.

When the 2D-DMA accesses the encrypted external memory space (see Chapter 34 *External Memory Encryption and Decryption (XTS_AES)*), buffer address pointer and corresponding data should be 16-byte aligned; when it accesses the configured internal memory and non-encrypted external memory space, there are no requirements for descriptor field alignment.

8.4.12 Arbitration

To ensure timely response to peripherals running at a high speed with low latency, the 2D-DMA controller implements two arbitration schemes, based on channel priority and channel weight respectively.

- Priority arbitration: Each channel can be assigned a priority from 0 ~15 (in total 16 levels). The larger the number, the higher the priority. When several channels perform data transfers at the same time, the

2D-DMA would respond to the transfer requests according to priority levels, and the channel with higher priority would get a response more timely.

- Weight arbitration:
  - Each channel is assigned a weight (i.e., the number of tokens) from 0 ~ 15. The 2D-DMA divides the AXI bus clock period into multiple time slots, and in each time slot, the number of transfers performed by a channel is determined by the number of its tokens. Every time a channel performs a data transfer, one of its tokens is spent. If all of its tokens have been spent in a time slot, then this channel's transfer request will no longer be responded to until the time slot expires, or until tokens of all channels have been spent and the new time slot starts.
  - If the number of tokens assigned to a channel is not zero, the 2D-DMA waits for this channel's tokens to be spent even if it does not have data transfer requests, and will not exit from this time slot ahead of expiration. This leads to a waste of bandwidth. Therefore, the 2D-DMA provides weight arbitration optimization. When this feature is enabled, a channel without data transfer requests will not be involved in the arbitration and its tokens will be ignored. When all channels no longer have transfer requests, the 2D-DMA arbiter exits from the current time slot before expiration. This way, the arbitration response is accelerated and the transfer efficiency is improved.

Note:

- If channels have the same weight but different priorities, on condition that the number of bytes to be transferred is the same and not large, channels with higher priorities would finish data transfers first.
- If channels have different priorities and weights, on condition that the number of bytes to be transferred is the same, channels with higher weights would be allocated more bandwidth and finish data transfers first.

8.5 Event Task Matrix Feature

The 2D-DMA controller on ESP32-P4 supports the Event Task Matrix (ETM) function, which allows 2D-DMA's ETM tasks to be triggered by any peripherals' ETM events, or 2D-DMA's ETM events to trigger any peripherals' ETM tasks. This section introduces the ETM tasks and events related to 2D-DMA. For more information, please refer to Chapter 14 Event Task Matrix (ETM).

2D-DMA can receive the following ETM tasks:

- DMA2D_TASK_IN_START_CH*n*: Enables the corresponding receive channel *n* for data transfer.
- DMA2D_TASK_OUT_START_CH*n*: Enables the corresponding transmit channel *n* for data transfer.

  Note:
  Above ETM tasks can achieve the same functions as CPU configuring DMA2D_INLINK_START_CH*n* and DMA2D_OUTLINK_START_CH*n*. When DMA2D_IN_ETM_EN_CH*n* or DMA2D_OUT_ETM_EN_CH*n* is 1, only ETM tasks can be used to configure the transfer direction and enable the corresponding 2D-DMA channel. When DMA2D_IN_ETM_EN_CH*n* or DMA2D_OUT_ETM_EN_CH*n* is 0, only CPU can be used to enable the corresponding 2D-DMA channel.

- DMA2D_TASK_IN_DSCR_READY_CH*n*: Allows receive channel *n* to process the next descriptor.

- DMA2D_TASK_OUT_DSCR_READY_CH*n*: Allows transmit channel *n* to process the next descriptor.

> **Note:**
> In normal conditions, after finishing with the current descriptor if the next descriptor is available (i.e., the next descriptor address field is not zero), the 2D-DMA would process the next descriptor directly and reads/writes to the memory according to the descriptor's configuration until all descriptors are processed. When DMA2D_IN_ETM_LOOP_EN_CH*n* or DMA2D_OUT_ETM_LOOP_EN_CH*n* is set, after finishing with the current descriptor, the 2D-DMA would not start processing the next descriptor until receiving the DMA2D_TASK_IN_DSCR_READY_CH*n* or
> DMA2D_TASK_OUT_DSCR_READY_CH*n* task.
> The DMA2D_TASK_IN_DSCR_READY_CH*n* or DMA2D_TASK_OUT_DSCR_READY_CH*n* task can be stored in the task buffer, which can cache at most four tasks. When the 2D-DMA receives this task but the current descriptor is not yet processed, it can cache the task. The depth of the task buffer is determined by DMA2D_IN_DSCR_TASK_MAX_CH*n* or DMA2D_OUT_DSCR_TASK_MAX_CH*n*.
> When the task buffer exceeds its maximum capacity, it will trigger an DMA2D_IN_DSCR_TASK_OVF_CH*n*_INT or DMA2D_OUT_DSCR_TASK_OVF_CH*n*_INT interrupt.

2D-DMA can generate the following ETM events:

- DMA2D_EVT_IN_DONE_CH*n*: Indicates that the data has been received according to the receive descriptor via channel *n*.
- DMA2D_EVT_IN_SUC_EOF_CH*n*: Indicates that the data corresponding to a receive descriptor has been received via channel *n* and the Eof bit of this descriptor is 1.
- DMA2D_EVT_OUT_DONE_CH*n*: Indicates that the data has been transmitted according to the transmit descriptor via channel *n*.
- DMA2D_EVT_OUT_SUC_EOF_CH*n*: Indicates that the data corresponding to a transmit descriptor has been transmitted or received via channel *n* and the eof bit of this descriptor is 1.
- DMA2D_EVT_OUT_TOTAL_EOF_CH*n*: Indicates that the data corresponding to the last transmit descriptors has been sent via transmit channel *n* and the eof bit of this descriptor is 1.

In practical applications, 2D-DMA's ETM events can trigger its own ETM tasks. For example, the DMA2D_EVT_OUT_TOTAL_EOF_CH0 event can trigger the DMA2D_TASK_IN_START_CH1 task, and in this way trigger a new round of 2D-DMA operations.

8.6 Interrupts

ESP32-P4's 2D-DMA module can generate the following interrupt signals that will be sent to the Interrupt Matrix.

- DMA2D_IN_CH0_INTR
- DMA2D_IN_CH1_INTR
- DMA2D_IN_CH2_INTR
- DMA2D_OUT_CH0_INTR
- DMA2D_OUT_CH1_INTR

- DMA2D_OUT_CH2_INTR
- DMA2D_OUT_CH3_INTR

There are several internal interrupt sources from 2D-DMA that can generate the above interrupt signals.

The DMA2D_IN_CH*n*_INTR (*n* is 0 ~ 2) signals are generated by the following interrupt sources:

- DMA2D_IN_DONE_CH*n*_INT: Triggered when all data corresponding to a receive descriptor has been received via receive channel *n*.
- DMA2D_IN_SUC_EOF_CH*n*_INT: Triggered when the eof bit in a receive descriptor is 1 and the data corresponding to this receive descriptor has been received (i.e., when an image corresponding to an inlink has been received) via receive channel *n*.
- DMA2D_IN_ERR_EOF_CH*n*_INT: Triggered when an error is detected in the image received via receive channel *n*. This interrupt is used only for JPEG.
- DMA2D_IN_DSCR_EMPTY_CH*n*_INT: Triggered when the size of the buffer pointed by receive descriptors is smaller than the length of data to be received via receive channel *n*.
- DMA2D_IN_DSCR_ERR_CH*n*_INT: Triggered when a receive descriptor on receive channel *n* fails any of the two descriptor checks.
- DMA2D_INFIFO_OVF_CH*n*_INT: Triggered when the RX FIFO of 2D-DMA overflows.
- DMA2D_INFIFO_UDF_CH*n*_INT or AXI_DMA_INFIFO_L1/L2/L3_UDF_CH*n*_INT: Triggered when the RX FIFO of 2D-DMA underflows.
- DMA2D_IN_DSCR_TASK_OVF_CH*n*_INT: Triggered when the number of cached DMA2D_TASK_IN_DSCR_READY_CH*n* task exceeds the maximum capacity of the task buffer.
- DMA2D_INFIFO_RO_OVF_CH0_INT (only channel 0): Triggered when the buffer for macroblock reordering in RX direction overflows.
- DMA2D_INFIFO_RO_UDF_CH0_INT (only channel 0): Triggered when the buffer for macroblock reordering in RX direction underflows.

The DMA2D_OUT_CH*n*_INTR (*n* is 0 ~ 3) signals are generated by the following interrupt sources:

- DMA2D_OUT_DONE_CH*n*_INT: Triggered when all data corresponding to a transmit descriptor has been sent via transmit channel *n*.
- DMA2D_OUT_EOF_CH*n*_INT: Triggered when the eof bit in a transmit descriptor is 1 and data corresponding to this descriptor has been sent via transmit channel *n*. If DMA2D_OUT_EOF_MODE_CH*n* is 0, this interrupt will be triggered when the last byte of data corresponding to this descriptor enters 2D-DMA's transmit channel via the bus; if DMA2D_OUT_EOF_MODE_CH*n* is 1, this interrupt is triggered when the last byte of data is taken from 2D-DMA's transmit channel.
- DMA2D_OUT_DSCR_ERR_CH*n*_INT: Triggered when a transmit descriptor on transmit channel *n* fails any of the two descriptor checks.
- DMA2D_OUT_TOTAL_EOF_CH*n*_INT: Triggered when all data corresponding to a linked list (including multiple descriptors) has been sent via transmit channel *n*.
- DMA2D_OUTFIFO_L1/L2/L3_OVF_CH*n*_INT: Triggered when the TX FIFO of 2D-DMA overflows.
- DMA2D_OUTFIFO_L1/L2/L3_UDF_CH*n*_INT: Triggered when the TX FIFO of 2D-DMA underflows.

- DMA2D_OUT_DSCR_TASK_OVF_CH*n*_INT: Triggered when the number of cached DMA2D_TASK_OUT_DSCR_READY_CH*n* task exceeds the maximum capacity of the task buffer.
- DMA2D_OUTFIFO_RO_OVF_CH0_INT (only channel 0): Triggered when the buffer for macroblock reordering or color space conversion in TX direction overflows.
- DMA2D_OUTFIFO_RO_UDF_CH0_INT (only channel 0): Triggered when the buffer for macroblock reordering or color space conversion in TX direction underflows.

8.7 Programming Procedures

The clock gating for the 2D-DMA can be configured via HP_SYS_CLKRST_DMA2D_SYS_CLK_EN, and is enabled by default. The 2D-DMA can be reset globally by configuring HP_SYS_CLKRST_RST_EN_AHB/AXI_PDMA.

8.7.1 General Configurations for 2D-DMA

Configure the accessible address space of internal and external memory and burst length according to Section 8.4.11.

8.7.2 Mode-Specific Configurations for 2D-DMA

- 1D Mode: No specific configuration required.
- 2D-MOD0 Mode: No specific configuration required.
- 2D-MOD1 Mode: For JPEG, configure DMA2D_IN_MACRO_BLOCK_SIZE_CH*n* or DMA2D_OUT_MACRO_BLOCK_SIZE_CH*n* to choose a macroblock size out of the three available options (horizontal × vertical): 8 pixels × 8 pixels, 8 pixels × 16 pixels, and 16 pixels × 16 pixels. For more information about macroblock size, see Chapter 37 *JPEG Codec*. For correspondence between macroblock size, hb, and vb of descriptors, see Table 8.4-4 and Table 8.4-5.
- DSCR-PORT Mode: Set DMA2D_OUT_DSCR_PORT_EN_CH*n* to enter this mode. This mode is specifically designed for PPA. For details, see Chapter 39 *Pixel-Processing Accelerator (PPA)*.

8.7.3 Configurations for 2D-DMA's Transmit Channel

To transmit data, 2D-DMA's transmit channel should be configured by software as follows:

1. Set DMA2D_OUT_RST_CH*n* first to 1 and then to 0, to reset the state machine of 2D-DMA's transmit channel and FIFO pointer.
2. Load an outlink, and configure DMA2D_OUTLINK_ADDR_CH*n* with address of the first transmit descriptor.
3. Configure DMA2D_OUT_PERI_SEL_CH*n* with the value corresponding to the peripheral to be connected, as shown in Table 8.4-2.
4. Set DMA2D_OUTLINK_START_CH*n* to enable 2D-DMA's transmit channel for data transfer.
5. Configure and enable the corresponding peripheral. See details in individual chapters of these peripherals.
6. Wait for the DMA2D_OUT_EOF_CH*n*_INT interrupt, which indicates the completion of data transfer.

8.7.4 Configurations for 2D-DMA's Receive Channel

To receive data, 2D-DMA's receive channel should be configured by software as follows:

1. Set DMA2D_IN_RST_CH*n* first to 1 and then to 0, to reset the state machine of 2D-DMA's receive channel and FIFO pointer.
2. Load an inlink, and configure DMA2D_INLINK_ADDR_CH*n* with address of the first receive descriptor.
3. Configure DMA2D_IN_PERI_SEL_CH*n* with the value corresponding to the peripheral to be connected, as shown in Table 8.4-3.
4. Set DMA2D_INLINK_START_CH*n* to enable 2D-DMA's receive channel for data transfer.
5. Configure and enable the corresponding peripheral. See details in individual chapters of these peripherals.
6. Wait for DMA2D_IN_SUC_EOF_CH*n*_INT interrupt, which indicates that an image has been received.

8.7.5 Configurations for Memory-to-Memory Transfer

To transfer data from one memory location to another, 2D-DMA should be configured by software as follows:

1. Set DMA2D_OUT_RST_CH*n* first to 1 and then to 0, to reset the state machine of 2D-DMA's transmit channel and FIFO pointer.
2. Set DMA2D_IN_RST_CH*n* first to 1 and then to 0, to reset the state machine of 2D-DMA's receive channel and FIFO pointer.
3. Load an outlink, and configure DMA2D_OUTLINK_ADDR_CH*n* with address of the first transmit descriptor.
4. Load an inlink, and configure DMA2D_INLINK_ADDR_CH*n* with address of the first receive descriptor.
5. Configure DMA2D_PERI_OUT_SEL_CH*n* with any value corresponding to Dummy-*n* (see Table 8.4-2).
6. Configure DMA2D_PERI_IN_SEL_CH*n* with any value corresponding to Dummy-*n* (see Table 8.4-3).
7. Set DMA2D_IN_MEM_TRANS_EN_CH*n* to enable memory-to-memory transfer.
8. Set DMA2D_OUTLINK_START_CH*n* to enable 2D-DMA's transmit channel for data transfer.
9. Set DMA2D_INLINK_START_CH*n* to enable 2D-DMA's receive channel for data transfer.
10. Wait for DMA2D_IN_SUC_EOF_CH*n*_INT interrupt, which indicates that a data transaction has been completed.

8.7.6 Configurations for Channel Priority and Weight

The priority arbitration can be configured as follows:

1. Configure the channel priority for TX and RX respectively via DMA2D_OUT_ARB_PRIORITY_CH*n* and DMA2D_IN_ARB_PRIORITY_CH*n*.

The weight arbitration can be configured as follows:

1. Configure the time slot for TX and RX respectively via DMA2D_OUT_ARB_TIMEOUT_NUM and DMA2D_IN_ARB_TIMEOUT_NUM.

2. Configure the number of tokens for TX and RX respectively via DMA2D_OUT_ARB_TOKEN_NUM_CH*n* and DMA2D_IN_ARB_TOKEN_NUM_CH*n*.
3. Enable weight arbitration optimization for TX and RX respectively by clearing DMA2D_OUT_ARB_WEIGHT_OPT_DIS_CH_CH*n* and DMA2D_IN_ARB_WEIGHT_OPT_DIS_CH*n*.
4. Enable the arbitration for TX and RX respectively via DMA2D_OUT_WEIGHT_EN and DMA2D_IN_WEIGHT_EN.

8.7.7 Resetting 2D-DMA While it Runs

When the 2D-DMA is running, namely when it is still processing descriptors, it must be stopped first before being reset fully via HP_SYS_CLKRST_RST_EN_DMA2D or partially (only reset AXI functions) via DMA2D_AXIM_RD_RST and DMA2D_AXIM_WR_RST. The procedures are as follows:

1. Set DMA2D_IN_CMD_DISABLE_CH*n* and DMA2D_OUT_CMD_DISABLE_CH*n* to stop the 2D-DMA.
2. Reset the 2D-DMA when DMA2D_OUT_RESET_AVAIL_CH*n* and DMA2D_IN_RESET_AVAIL_CH*n* turn to 1.
3. Clear DMA2D_IN_CMD_DISABLE_CH*n* and DMA2D_OUT_CMD_DISABLE_CH*n* to 0 after the 2D-DMA is reset.

8.8 Register Summary

The addresses in this section are relative to 2D-DMA base address provided in Table 9.3-2 in Chapter 9 *System and Memory*.

The abbreviations given in Column **Access** are explained in Section *Access Types for Registers*.

Name	Description	Address	Access
Configuration Registers			
DMA2D_OUT_CONF0_CH0_REG	Configures the TX channel 0	0x0000	R/W
DMA2D_OUT_PUSH_CH0_REG	Configures the FIFO of TX channel 0	0x0018	varies
DMA2D_OUT_LINK_CONF_CH0_REG	Configures the transmit descriptor operations of TX channel 0	0x001C	varies
DMA2D_OUT_LINK_ADDR_CH0_REG	Configures the transmit descriptor address of TX channel 0	0x0020	R/W
DMA2D_OUT_ARB_CH0_REG	Configures the arbitration of TX channel 0	0x003C	R/W
DMA2D_OUT_RO_PD_CONF_CH0_REG	Configures the macroblock reordering memory of TX channel 0	0x0044	R/W
DMA2D_OUT_COLOR_CONVERT_CH0_REG	Configures the color space conversion of TX channel 0	0x0048	R/W
DMA2D_OUT_SCRAMBLE_CH0_REG	Configures the byte scrambling of TX channel 0	0x004C	R/W
DMA2D_OUT_COLOR_PARAM0_CH0_REG	Configures the color space conversion parameter of TX channel 0	0x0050	R/W
DMA2D_OUT_COLOR_PARAM1_CH0_REG	Configures the color space conversion parameter of TX channel 0	0x0054	R/W
DMA2D_OUT_COLOR_PARAM2_CH0_REG	Configures the color space conversion parameter of TX channel 0	0x0058	R/W
DMA2D_OUT_COLOR_PARAM3_CH0_REG	Configures the color space conversion parameter of TX channel 0	0x005C	R/W
DMA2D_OUT_COLOR_PARAM4_CH0_REG	Configures the color space conversion parameter of TX channel 0	0x0060	R/W
DMA2D_OUT_COLOR_PARAM5_CH0_REG	Configures the color space conversion parameter of TX channel 0	0x0064	R/W
DMA2D_OUT_ETM_CONF_CH0_REG	Configures the ETM of TX channel 0	0x0068	R/W
DMA2D_OUT_DSCR_PORT_BLK_CH0_REG	Configures the block size for TX channel 0 in DSCR-PORT mode	0x006C	R/W
DMA2D_OUT_CONF0_CH1_REG	Configures the TX channel 1	0x0100	R/W
DMA2D_OUT_PUSH_CH1_REG	Configures the FIFO of TX channel 1	0x0118	varies
DMA2D_OUT_LINK_CONF_CH1_REG	Configures the transmit descriptor operations of TX channel 1	0x011C	varies
DMA2D_OUT_LINK_ADDR_CH1_REG	Configures the transmit descriptor address of TX channel 1	0x0120	R/W
DMA2D_OUT_ARB_CH1_REG	Configures the arbitration of TX channel 1	0x013C	R/W
DMA2D_OUT_COLOR_CONVERT_CH1_REG	Configures the color space conversion of TX channel 1	0x0148	R/W
DMA2D_OUT_SCRAMBLE_CH1_REG	Configures the byte scrambling of TX channel 1	0x014C	R/W

Name	Description	Address	Access
DMA2D_OUT_COLOR_PARAM0_CH1_REG	Configures the color space conversion parameter of TX channel 1	0x0150	R/W
DMA2D_OUT_COLOR_PARAM1_CH1_REG	Configures the color space conversion parameter of TX channel 1	0x0154	R/W
DMA2D_OUT_COLOR_PARAM2_CH1_REG	Configures the color space conversion parameter of TX channel 1	0x0158	R/W
DMA2D_OUT_COLOR_PARAM3_CH1_REG	Configures the color space conversion parameter of TX channel 1	0x015C	R/W
DMA2D_OUT_COLOR_PARAM4_CH1_REG	Configures the color space conversion parameter of TX channel 1	0x0160	R/W
DMA2D_OUT_COLOR_PARAM5_CH1_REG	Configures the color space conversion parameter of TX channel 1	0x0164	R/W
DMA2D_OUT_ETM_CONF_CH1_REG	Configures the ETM of TX channel 1	0x0168	R/W
DMA2D_OUT_DSCR_PORT_BLK_CH1_REG	Configures the block size for TX channel 1 in DSCR-PORT mode	0x016C	R/W
DMA2D_OUT_CONF0_CH2_REG	Configures the TX channel 2	0x0200	R/W
DMA2D_OUT_PUSH_CH2_REG	Configures the FIFO of TX channel 2	0x0218	varies
DMA2D_OUT_LINK_CONF_CH2_REG	Configures the transmit descriptor operations of TX channel 2	0x021C	varies
DMA2D_OUT_LINK_ADDR_CH2_REG	Configures the transmit descriptor address of TX channel 2	0x0220	R/W
DMA2D_OUT_ARB_CH2_REG	Configures the arbitration of TX channel 2	0x023C	R/W
DMA2D_OUT_COLOR_CONVERT_CH2_REG	Configures the color space conversion of TX channel 2	0x0248	R/W
DMA2D_OUT_SCRAMBLE_CH2_REG	Configures the byte scrambling of TX channel 2	0x024C	R/W
DMA2D_OUT_COLOR_PARAM0_CH2_REG	Configures the color space conversion parameter of TX channel 2	0x0250	R/W
DMA2D_OUT_COLOR_PARAM1_CH2_REG	Configures the color space conversion parameter of TX channel 2	0x0254	R/W
DMA2D_OUT_COLOR_PARAM2_CH2_REG	Configures the color space conversion parameter of TX channel 2	0x0258	R/W
DMA2D_OUT_COLOR_PARAM3_CH2_REG	Configures the color space conversion parameter of TX channel 2	0x025C	R/W
DMA2D_OUT_COLOR_PARAM4_CH2_REG	Configures the color space conversion parameter of TX channel 2	0x0260	R/W
DMA2D_OUT_COLOR_PARAM5_CH2_REG	Configures the color space conversion parameter of TX channel 2	0x0264	R/W
DMA2D_OUT_ETM_CONF_CH2_REG	Configures the ETM of TX channel 2	0x0268	R/W
DMA2D_OUT_DSCR_PORT_BLK_CH2_REG	Configures the block size for TX channel 2 in DSCR-PORT mode	0x026C	R/W
DMA2D_OUT_CONF0_CH3_REG	Configures the TX channel 3	0x0300	R/W
DMA2D_OUT_PUSH_CH3_REG	Configures the FIFO of TX channel 3	0x0318	varies
DMA2D_OUT_LINK_CONF_CH3_REG	Configures the transmit descriptor operations of TX channel 3	0x031C	varies
DMA2D_OUT_LINK_ADDR_CH3_REG	Configures the transmit descriptor address of TX channel 3	0x0320	R/W
DMA2D_OUT_ARB_CH3_REG	Configures the arbitration of TX channel 3	0x033C	R/W
DMA2D_OUT_COLOR_CONVERT_CH3_REG	Configures the color space conversion of TX channel 3	0x0348	R/W

Name	Description	Address	Access
DMA2D_OUT_SCRAMBLE_CH3_REG	Configures the byte scrambling of TX channel 3	0x034C	R/W
DMA2D_OUT_COLOR_PARAM0_CH3_REG	Configures the color space conversion parameter of TX channel 3	0x0350	R/W
DMA2D_OUT_COLOR_PARAM1_CH3_REG	Configures the color space conversion parameter of TX channel 3	0x0354	R/W
DMA2D_OUT_COLOR_PARAM2_CH3_REG	Configures the color space conversion parameter of TX channel 3	0x0358	R/W
DMA2D_OUT_COLOR_PARAM3_CH3_REG	Configures the color space conversion parameter of TX channel 3	0x035C	R/W
DMA2D_OUT_COLOR_PARAM4_CH3_REG	Configures the color space conversion parameter of TX channel 3	0x0360	R/W
DMA2D_OUT_COLOR_PARAM5_CH3_REG	Configures the color space conversion parameter of TX channel 3	0x0364	R/W
DMA2D_OUT_ETM_CONF_CH3_REG	Configures the ETM of TX channel 3	0x0368	R/W
DMA2D_OUT_DSCR_PORT_BLK_CH3_REG	Configures the block size for TX channel 3 in DSCR-PORT mode	0x036C	R/W
DMA2D_IN_CONF0_CH0_REG	Configures the RX channel 0	0x0500	R/W
DMA2D_IN_POP_CH0_REG	Configures the FIFO of RX channel 0	0x0518	varies
DMA2D_IN_LINK_CONF_CH0_REG	Configures the receive descriptor operations of RX channel 0	0x051C	varies
DMA2D_IN_LINK_ADDR_CH0_REG	Configures the receive descriptor address of RX channel 0	0x0520	R/W
DMA2D_IN_ARB_CH0_REG	Configures the arbitration of RX channel 0	0x0540	R/W
DMA2D_IN_RO_PD_CONF_CH0_REG	Configures the macroblock reordering memory of RX channel 0	0x0548	R/W
DMA2D_IN_COLOR_CONVERT_CH0_REG	Configures the color space conversion of RX channel 0	0x054C	R/W
DMA2D_IN_SCRAMBLE_CH0_REG	Configures the byte scrambling of RX channel 0	0x0550	R/W
DMA2D_IN_COLOR_PARAM0_CH0_REG	Configures the color space conversion parameter of RX channel 0	0x0554	R/W
DMA2D_IN_COLOR_PARAM1_CH0_REG	Configures the color space conversion parameter of RX channel 0	0x0558	R/W
DMA2D_IN_COLOR_PARAM2_CH0_REG	Configures the color space conversion parameter of RX channel 0	0x055C	R/W
DMA2D_IN_COLOR_PARAM3_CH0_REG	Configures the color space conversion parameter of RX channel 0	0x0560	R/W
DMA2D_IN_COLOR_PARAM4_CH0_REG	Configures the color space conversion parameter of RX channel 0	0x0564	R/W
DMA2D_IN_COLOR_PARAM5_CH0_REG	Configures the color space conversion parameter of RX channel 0	0x0568	R/W
DMA2D_IN_ETM_CONF_CH0_REG	Configures the ETM of RX channel 0	0x056C	R/W
DMA2D_IN_CONF0_CH1_REG	Configures the RX channel 1	0x0600	R/W
DMA2D_IN_POP_CH1_REG	Configures the FIFO of RX channel 1	0x0618	varies
DMA2D_IN_LINK_CONF_CH1_REG	Configures the receive descriptor operations of RX channel 1	0x061C	varies
DMA2D_IN_LINK_ADDR_CH1_REG	Configures the receive descriptor address of RX channel 1	0x0620	R/W
DMA2D_IN_ARB_CH1_REG	Configures the arbitration of RX channel 1	0x0640	R/W

Name	Description	Address	Access
DMA2D_IN_ETM_CONF_CH1_REG	Configures the ETM of RX channel 1	0x0648	R/W
DMA2D_IN_CONF0_CH2_REG	Configures the RX channel 2	0x0700	R/W
DMA2D_IN_POP_CH2_REG	Configures the FIFO of RX channel 2	0x0718	varies
DMA2D_IN_LINK_CONF_CH2_REG	Configures the receive descriptor operations of RX channel 2	0x071C	varies
DMA2D_IN_LINK_ADDR_CH2_REG	Configures the receive descriptor address of RX channel 2	0x0720	R/W
DMA2D_IN_ARB_CH2_REG	Configures the arbitration of RX channel 2	0x0740	R/W
DMA2D_IN_ETM_CONF_CH2_REG	Configures the ETM of RX channel 2	0x0748	R/W
DMA2D_RST_CONF_REG	Configures the reset of AXI	0x0A04	R/W
DMA2D_INTR_MEM_START_ADDR_REG	The start address of accessible internal memory address space	0x0A08	R/W
DMA2D_INTR_MEM_END_ADDR_REG	The end address of accessible internal memory address space	0x0A0C	R/W
DMA2D_EXTR_MEM_START_ADDR_REG	The start address of accessible external memory address space	0x0A10	R/W
DMA2D_EXTR_MEM_END_ADDR_REG	The end address of accessible external memory address space	0x0A14	R/W
DMA2D_OUT_ARB_CONFIG_REG	Configures the arbitration in TX direction	0x0A18	R/W
DMA2D_IN_ARB_CONFIG_REG	Configures the arbitration in RX direction	0x0A1C	R/W
Interrupt Registers			
DMA2D_OUT_INT_RAW_CH0_REG	Raw interrupt status of TX channel 0	0x0004	R/WTC/SS
DMA2D_OUT_INT_ENA_CH0_REG	Interrupt enable bits of TX channel 0	0x0008	R/W
DMA2D_OUT_INT_ST_CH0_REG	Masked interrupt status of TX channel 0	0x000C	RO
DMA2D_OUT_INT_CLR_CH0_REG	Interrupt clear bits of TX channel 0	0x0010	WT
DMA2D_OUT_INT_RAW_CH1_REG	Raw interrupt status of TX channel 1	0x0104	R/WTC/SS
DMA2D_OUT_INT_ENA_CH1_REG	Interrupt enable bits of TX channel 1	0x0108	R/W
DMA2D_OUT_INT_ST_CH1_REG	Masked interrupt status of TX channel 1	0x010C	RO
DMA2D_OUT_INT_CLR_CH1_REG	Interrupt clear bits of TX channel 1	0x0110	WT
DMA2D_OUT_INT_RAW_CH2_REG	Raw interrupt status of TX channel 2	0x0204	R/WTC/SS
DMA2D_OUT_INT_ENA_CH2_REG	Interrupt enable bits of TX channel 2	0x0208	R/W
DMA2D_OUT_INT_ST_CH2_REG	Masked interrupt status of TX channel 2	0x020C	RO
DMA2D_OUT_INT_CLR_CH2_REG	Interrupt clear bits of TX channel 2	0x0210	WT
DMA2D_OUT_INT_RAW_CH3_REG	Raw interrupt status of TX channel 3	0x0304	R/WTC/SS
DMA2D_OUT_INT_ENA_CH3_REG	Interrupt enable bits of TX channel 3	0x0308	R/W

Name	Description	Address	Access
DMA2D_OUT_INT_ST_CH3_REG	Masked interrupt status of TX channel 3	0x030C	RO
DMA2D_OUT_INT_CLR_CH3_REG	Interrupt clear bits of TX channel 3	0x0310	WT
DMA2D_IN_INT_RAW_CH0_REG	Raw interrupt status of RX channel 0	0x0504	R/WTC/SS
DMA2D_IN_INT_ENA_CH0_REG	Interrupt enable bits of RX channel 0	0x0508	R/W
DMA2D_IN_INT_ST_CH0_REG	Masked interrupt status of RX channel 0	0x050C	RO
DMA2D_IN_INT_CLR_CH0_REG	Interrupt clear bits of RX channel 0	0x0510	WT
DMA2D_IN_INT_RAW_CH1_REG	Raw interrupt status of RX channel 1	0x0604	R/WTC/SS
DMA2D_IN_INT_ENA_CH1_REG	Interrupt enable bits of RX channel 1	0x0608	R/W
DMA2D_IN_INT_ST_CH1_REG	Masked interrupt status of RX channel 1	0x060C	RO
DMA2D_IN_INT_CLR_CH1_REG	Interrupt clear bits of RX channel 1	0x0610	WT
DMA2D_IN_INT_RAW_CH2_REG	Raw interrupt status of RX channel 2	0x0704	R/WTC/SS
DMA2D_IN_INT_ENA_CH2_REG	Interrupt enable bits of RX channel 2	0x0708	R/W
DMA2D_IN_INT_ST_CH2_REG	Masked interrupt status of RX channel 2	0x070C	RO
DMA2D_IN_INT_CLR_CH2_REG	Interrupt clear bits of RX channel 2	0x0710	WT
Status Registers			
DMA2D_OUTFIFO_STATUS_CH0_REG	Represents the status of the FIFO of TX channel 0	0x0014	RO
DMA2D_OUT_STATE_CH0_REG	Represents the working status of the transmit descriptor of TX channel 0	0x0024	RO
DMA2D_OUT_EOF_DES_ADDR_CH0_REG	Represents the transmit descriptor address when EOF occurs on TX channel 0	0x0028	RO
DMA2D_OUT_DSCR_CH0_REG	Represents the address of the next transmit descriptor pointed by the current pre-read transmit descriptor on TX channel 0	0x002C	RO
DMA2D_OUT_DSCR_BF0_CH0_REG	Represents the address of the current pre-read transmit descriptor on TX channel 0	0x0030	RO
DMA2D_OUT_DSCR_BF1_CH0_REG	Represents the address of the previous pre-read transmit descriptor on TX channel 0	0x0034	RO
DMA2D_OUTFIFO_STATUS_CH1_REG	Represents the status of the FIFO of TX channel 1	0x0114	RO
DMA2D_OUT_STATE_CH1_REG	Represents the working status of the transmit descriptor of TX channel 1	0x0124	RO
DMA2D_OUT_EOF_DES_ADDR_CH1_REG	Represents the transmit descriptor address when EOF occurs on TX channel 1	0x0128	RO

Name	Description	Address	Access
DMA2D_OUT_DSCR_CH1_REG	Represents the address of the next transmit descriptor pointed by the current pre-read transmit descriptor on TX channel 1	0x012C	RO
DMA2D_OUT_DSCR_BF0_CH1_REG	Represents the address of the current pre-read transmit descriptor on TX channel 1	0x0130	RO
DMA2D_OUT_DSCR_BF1_CH1_REG	Represents the address of the previous pre-read transmit descriptor on TX channel 1	0x0134	RO
DMA2D_OUTFIFO_STATUS_CH2_REG	Represents the status of the FIFO of TX channel 2	0x0214	RO
DMA2D_OUT_STATE_CH2_REG	Represents the working status of the transmit descriptor of TX channel 2	0x0224	RO
DMA2D_OUT_EOF_DES_ADDR_CH2_REG	Represents the transmit descriptor address when EOF occurs on TX channel 2	0x0228	RO
DMA2D_OUT_DSCR_CH2_REG	Represents the address of the next transmit descriptor pointed by the current pre-read transmit descriptor on TX channel 2	0x022C	RO
DMA2D_OUT_DSCR_BF0_CH2_REG	Represents the address of the current pre-read transmit descriptor on TX channel 2	0x0230	RO
DMA2D_OUT_DSCR_BF1_CH2_REG	Represents the address of the previous pre-read transmit descriptor on TX channel 2	0x0234	RO
DMA2D_INFIFO_STATUS_CH0_REG	Represents the status of the FIFO of RX channel 0	0x0514	RO
DMA2D_IN_STATE_CH0_REG	Represents the working status of the receive descriptor of RX channel 0	0x0524	RO
DMA2D_IN_SUC_EOF_DES_ADDR_CH0_REG	Represents the eceive descriptor address when EOF occurs on RX channel 0	0x0528	RO
DMA2D_IN_ERR_EOF_DES_ADDR_CH0_REG	Represents the receive descriptor address when errors occur on RX channel 0	0x052C	RO
DMA2D_IN_DSCR_CH0_REG	Represents the address of the next receive descriptor pointed by the current pre-read receive descriptor on RX channel 0	0x0530	RO
DMA2D_IN_DSCR_BF0_CH0_REG	Represents address of the current pre-read receive descriptor on RX channel 0	0x0534	RO
DMA2D_IN_DSCR_BF1_CH0_REG	Represents the address of the previous pre-read receive descriptor on RX channel 0	0x0538	RO
DMA2D_INFIFO_STATUS_CH1_REG	Represents the status of the FIFO of RX channel 1	0x0614	RO

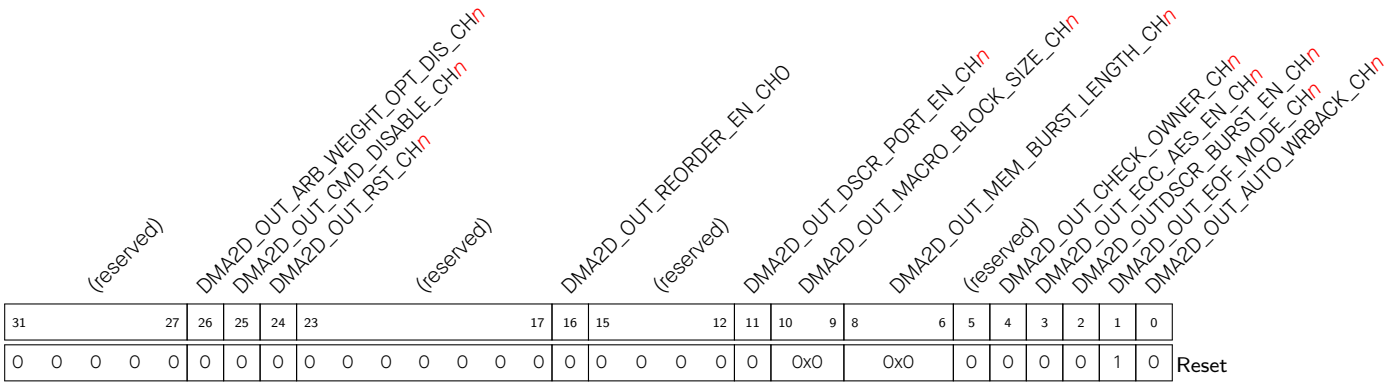
Name	Description	Address	Access
DMA2D_IN_STATE_CH1_REG	Represents the working status of the receive descriptor of RX channel 1	0x0624	RO
DMA2D_IN_SUC_EOF_DES_ADDR_CH1_REG	Represents the eceive descriptor address when EOF occurs on RX channel 1	0x0628	RO
DMA2D_IN_ERR_EOF_DES_ADDR_CH1_REG	Represents the receive descriptor address when errors occur on RX channel 1	0x062C	RO
DMA2D_IN_DSCR_CH1_REG	Represents the address of the next receive descriptor pointed by the current pre-read receive descriptor on RX channel 1	0x0630	RO
DMA2D_IN_DSCR_BF0_CH1_REG	Represents address of the current pre-read receive descriptor on RX channel 1	0x0634	RO
DMA2D_IN_DSCR_BF1_CH1_REG	Represents the address of the previous pre-read receive descriptor on RX channel 1	0x0638	RO
DMA2D_AXI_ERR_REG	Represents the status of the AXI bus	0x0A00	RO
DMA2D_DATE_REG	Version register	0x0A2C	R/W
Peripheral Select Registers			
DMA2D_OUT_PERI_SEL_CH0_REG	Configures the peripheral connected to TX channel 0	0x0038	R/W
DMA2D_OUT_PERI_SEL_CH1_REG	Configures the peripheral connected to TX channel 1	0x0138	R/W
DMA2D_OUT_PERI_SEL_CH2_REG	Configures the peripheral connected to TX channel 2	0x0238	R/W
DMA2D_OUT_PERI_SEL_CH3_REG	Configures the peripheral connected to TX channel 3	0x0338	R/W
DMA2D_IN_PERI_SEL_CH0_REG	Configures the peripheral connected to RX channel 0	0x053C	R/W
DMA2D_IN_PERI_SEL_CH1_REG	Configures the peripheral connected to RX channel 1	0x063C	R/W
DMA2D_IN_PERI_SEL_CH2_REG	Configures the peripheral connected to RX channel 2	0x073C	R/W

8.9 Registers

The addresses in this section are relative to 2D-DMA base address provided in Table 9.3-2 in Chapter 9 *System and Memory*.

For how to program reserved fields, please refer to Section *Programming Reserved Register Field*.

Register 8.1. DMA2D_OUT_CONF0_CH*n*_REG (*n*: 0-3) (0x0000+0x100n*)**


(reserved)
DMA2D_OUT_ARB_WEIGHT_OPT_DIS_CHn
DMA2D_OUT_CMD_DISABLE_CHn
DMA2D_OUT_RST_CHn
(reserved)
DMA2D_OUT_REORDER_EN_CH0
(reserved)
DMA2D_OUT_DSCR_PORT_EN_CHn
DMA2D_OUT_MACRO_BLOCK_SIZE_CHn
DMA2D_OUT_MEM_BURST_LENGTH_CHn
(reserved)
DMA2D_OUT_CHECK_OWNER_CHn
DMA2D_OUT_ECC_AES_EN_CHn
DMA2D_OUTDSCR_BURST_EN_CHn
DMA2D_OUT_EOF_MODE_CHn
DMA2D_OUT_AUTO_WRBACK_CHn
31 27 26 25 24 23 17 16 15 12 11 10 9 8 6 5 4 3 2 1 0
0 0 0

Register 8.1. DMA2D_OUT_CONF0_CH*n*_REG (*n*: 0-3) (0x0000+0x100**n*)

Continued from the previous page...

DMA2D_OUT_CHECK_OWNER_CH*n* Configures whether to enable the owner bit check for TX channel *n*.
0: Disable
1: Enable
(R/W)

DMA2D_OUT_MEM_BURST_LENGTH_CH*n* Configures the burst length for TX channel *n*.
0: 8 bytes
1: 16 bytes
2: 32 bytes
3: 64 bytes
4: 128 bytes
Others: Invalid
(R/W)

DMA2D_OUT_MACRO_BLOCK_SIZE_CH*n* Configures macroblock size for TX channel *n*. Used only in 2D-MOD1 mode.
0: 8 * 8 pixels
1: 8 * 16 pixels
2: 16 *16 pixels
3: No macroblock
(R/W)

DMA2D_OUT_DSCR_PORT_EN_CH*n* Configures whether to enable the DSCR-PORT mode for TX channel *n*.
0: Disable
1: Enable
(R/W)

DMA2D_OUT_REORDER_EN_CH0 Configures whether to enable macroblock reordering for TX channel 0.
0: Disable
1: Enable
(R/W)

DMA2D_OUT_RST_CH*n* Configures whether to reset TX channel *n*.
0: Reset release
1: Reset
(R/W)

Continued on the next page...

Register 8.1. DMA2D_OUT_CONFO_CH*n*_REG (*n*: 0-3) (0x0000+0x100**n*)

Continued from the previous page...

DMA2D_OUT_CMD_DISABLE_CH*n* Configures reset command for TX channel *n*.

0: Set 0 after reset release

1: Pause transfers before reset

(R/W)

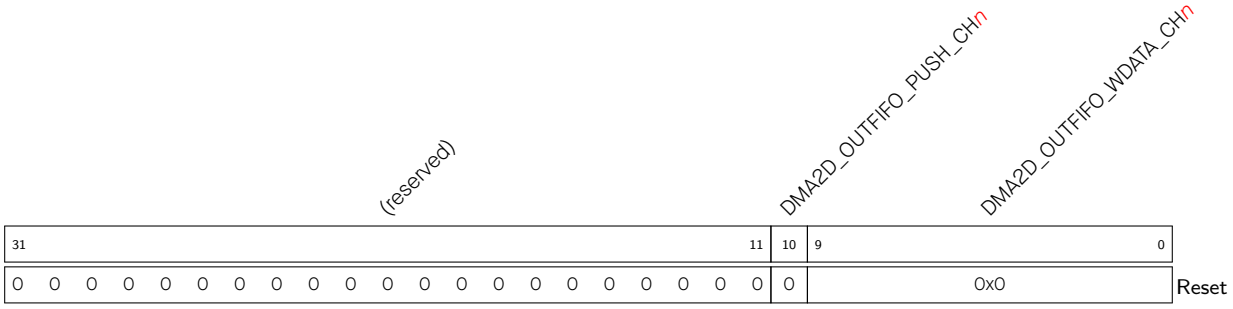
DMA2D_OUT_ARB_WEIGHT_OPT_DIS_CH*n* Configures whether to enable weight optimization for TX channel *n*.

0: Enable

1: Disable

(R/W)

Register 8.2. DMA2D_OUT_PUSH_CH*n*_REG (*n*: 0-3) (0x0018+0x100**n*)



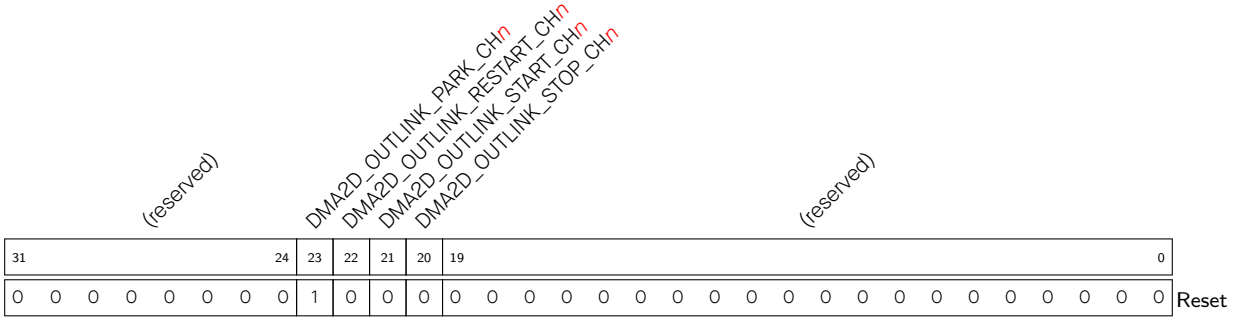
DMA2D_OUTFIFO_WDATA_CH*n* Configures the data to be pushed into the 2D-DMA TX FIFO. (R/W)

DMA2D_OUTFIFO_PUSH_CH*n* Configures whether to push data into the 2D-DMA TX FIFO.

0: Not push 1: Push

(R/W/SC)

Register 8.3. DMA2D_OUT_LINK_CONF_CH*n*_REG (*n*: 0-3) (0x001C+0x100**n*)



DMA2D_OUTLINK_STOP_CH*n* Configures whether to stop TX channel *n* from transmitting data.

0: Invalid. No effect

1: Stop

(R/W/SC)

DMA2D_OUTLINK_START_CH*n* Configures whether to enable TX channel *n* for data transfer.

0: Disable

1: Enable

(R/W/SC)

DMA2D_OUTLINK_RESTART_CH*n* Configures whether to restart TX channel *n* for AXI DMA transfer.

0: Invalid. No effect

1: Restart

(R/W/SC)

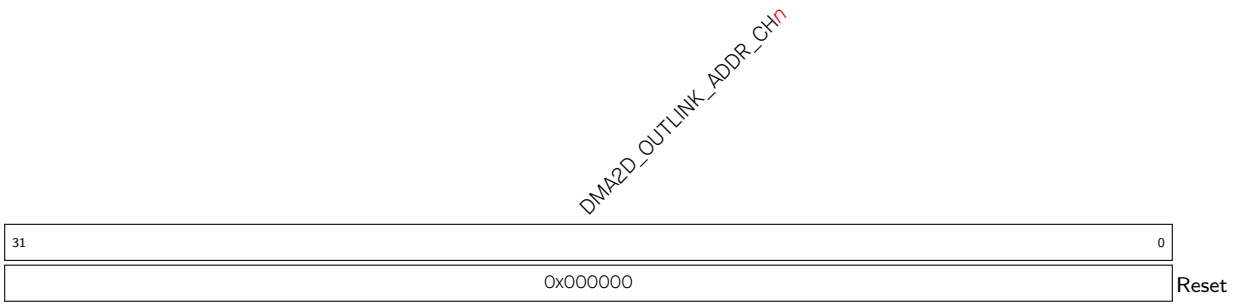
DMA2D_OUTLINK_PARK_CH*n* Represents the status of the transmit descriptor's FSM.

0: Running

1: Idle

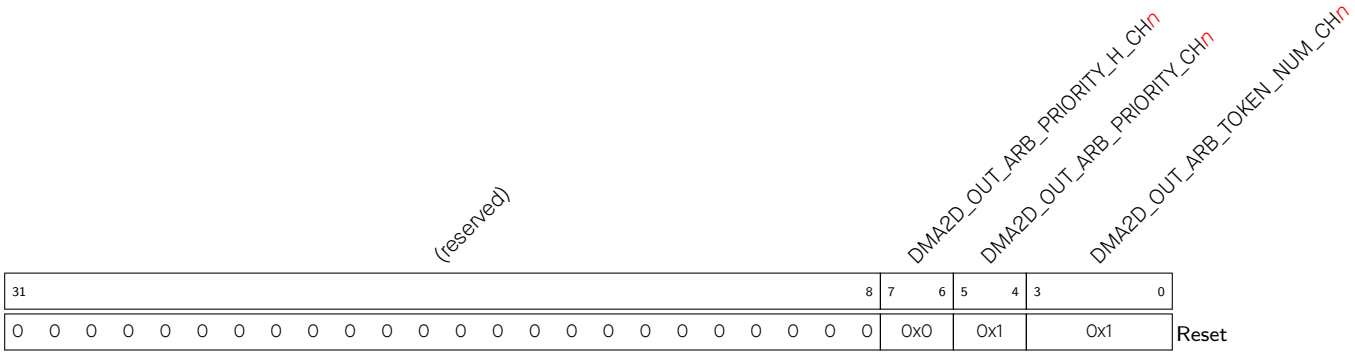
(RO)

Register 8.4. DMA2D_OUT_LINK_ADDR_CH*n*_REG (*n*: 0-3) (0x0020+0x100**n*)



DMA2D_OUTLINK_ADDR_CH*n* Represents the first transmit descriptor's address. (R/W)

Register 8.5. DMA2D_OUT_ARB_CH*n*_REG (*n*: 0-3) (0x003C+0x100**n*)



DMA2D_OUT_ARB_TOKEN_NUM_CH*n* Configures the weight (i.e the number of tokens) of TX channel *n*.

Value range: 0 ~ 15. (R/W)

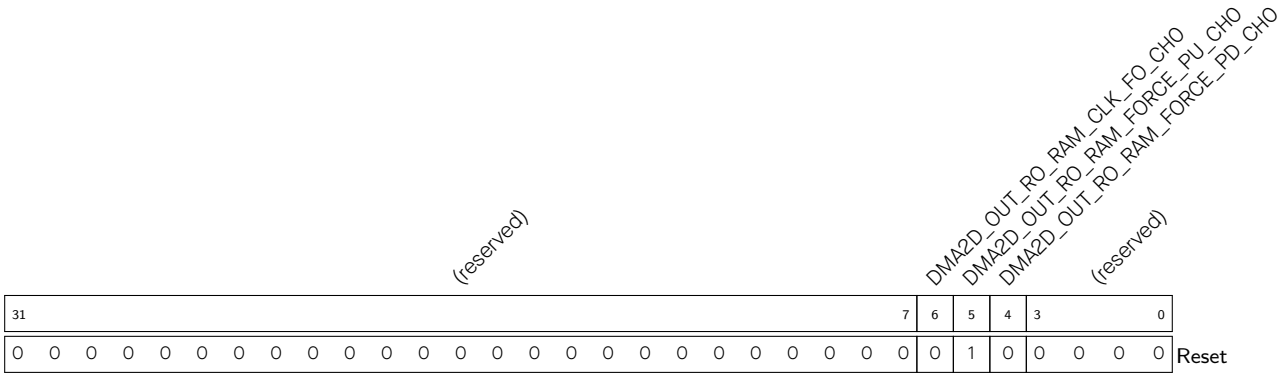
DMA2D_OUT_ARB_PRIORITY_CH*n* Configures the lower two bits of TX channel *n* priority.

Together with DMA2D_OUT_ARB_PRIORITY_H_CH*n*, this field forms a 4-bit priority value for TX channel *n*, ranging from 0 (lowest) to 15 (highest). (R/W)

DMA2D_OUT_ARB_PRIORITY_H_CH*n* Configures the higher two bits of TX channel *n* priority.

Together with DMA2D_OUT_ARB_PRIORITY_CH*n*, this field forms a 4-bit priority value for TX channel *n*, ranging from 0 (lowest) to 15 (highest). (R/W)

Register 8.6. DMA2D_OUT_RO_PD_CONF_CH0_REG (0x0044)



DMA2D_OUT_RO_RAM_FORCE_PD_CH0 Configures whether to force power down the macroblock reordering RAM for TX channel 0.

0: Not force power down

1: Force power down

(R/W)

DMA2D_OUT_RO_RAM_FORCE_PU_CH0 Configures whether to force power up the macroblock reordering RAM for TX channel 0.

0: Not force power up

1: Force power up

(R/W)

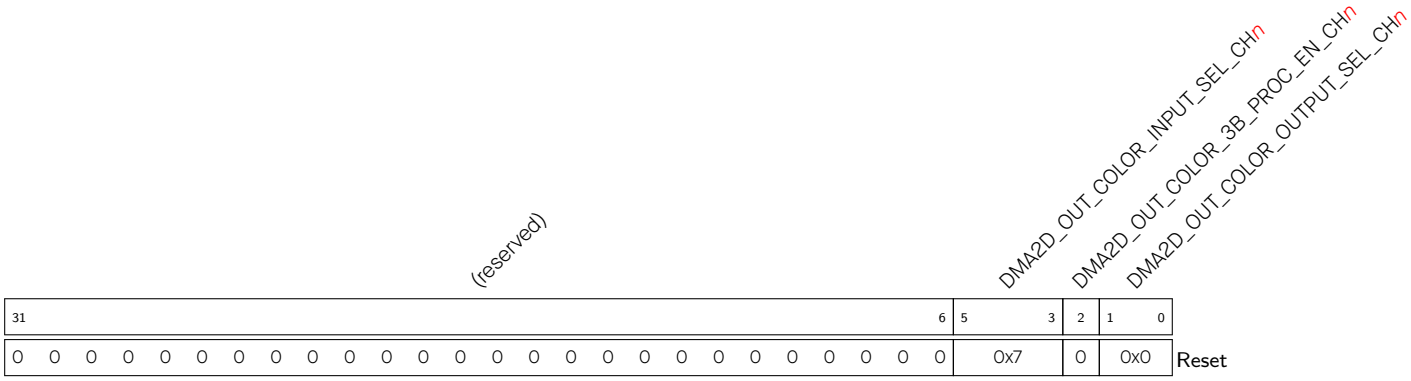
DMA2D_OUT_RO_RAM_CLK_FO_CH0 Configures the clock gating when TX channel 0 accessing the 2D-DMA RAM.

0: Use the clock gate

1: Force the clock on and bypass the clock gate

(R/W)

Register 8.7. DMA2D_OUT_COLOR_CONVERT_CH*n*_REG (*n*: 0-3) (0x0048+0x100n*)**



DMA2D_OUT_COLOR_OUTPUT_SEL_CH*n* Configures the third stage output of color space conversion for TX channel *n*.

0: RGB888 to RGB565

1: YUV444 to YUV422

2: Output directly

Others: Invalid

(R/W)

DMA2D_OUT_COLOR_3B_PROC_EN_CH*n* Configures whether to enable the second stage of color space conversion for TX channel *n*.

0: Disable

1: Enable

(R/W)

DMA2D_OUT_COLOR_INPUT_SEL_CH*n* Configures the first stage output of color space conversion for TX channel *n*.

0: RGB565 to RGB888

1: YUV422 to YUV444

2: Other 2 bytes/pixel color format

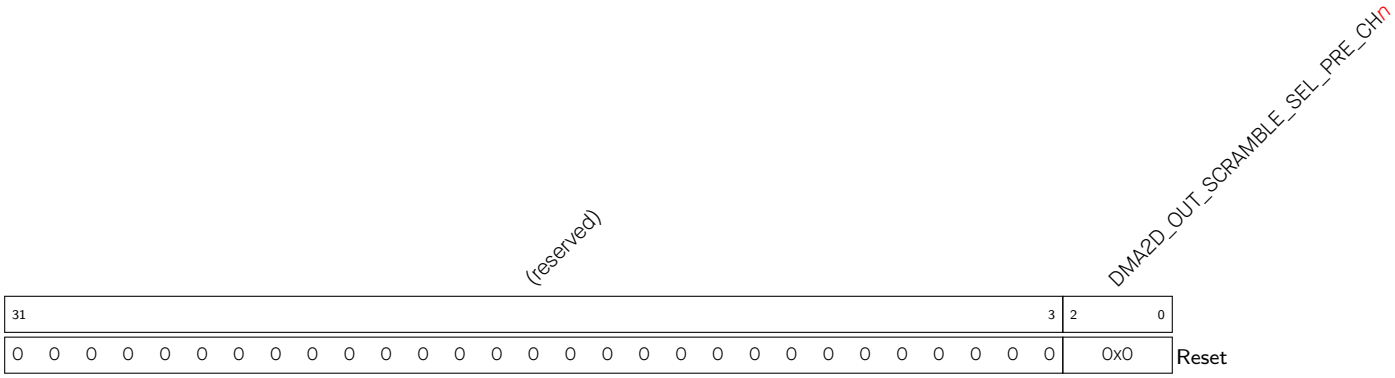
3: Other 3 bytes/pixel color format

7: Disable color space conversion

Others: Invalid

(R/W)

Register 8.8. DMA2D_OUT_SCRAMBLE_CH*n*_REG (*n*: 0-3) (0x004C+0x100**n*)



DMA2D_OUT_SCRAMBLE_SEL_PRE_CH*n* Configures the byte order scrambling before the color space conversion for TX channel *n*.

0: BYTE2-1-0

1: BYTE2-0-1

2: BYTE1-0-2

3: BYTE1-2-0

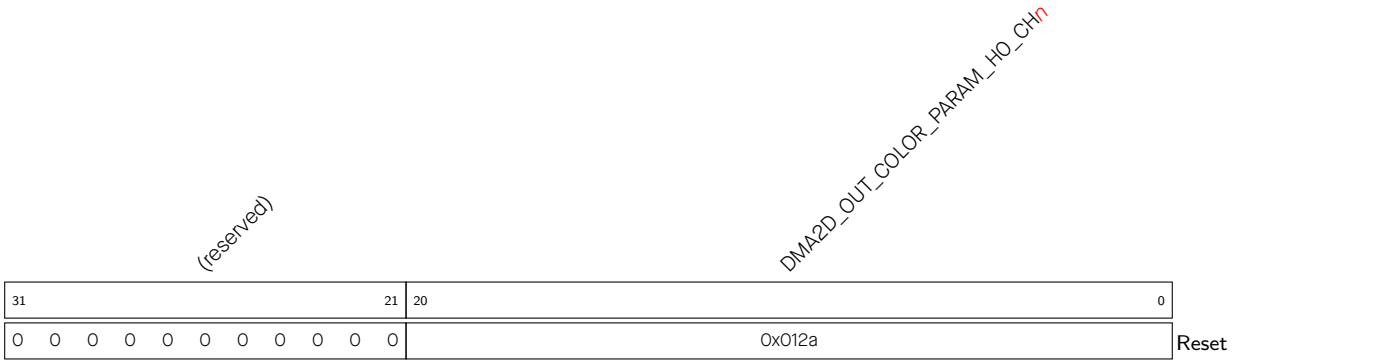
4: BYTE0-2-1

5: BYTE0-1-2

Others: Invalid

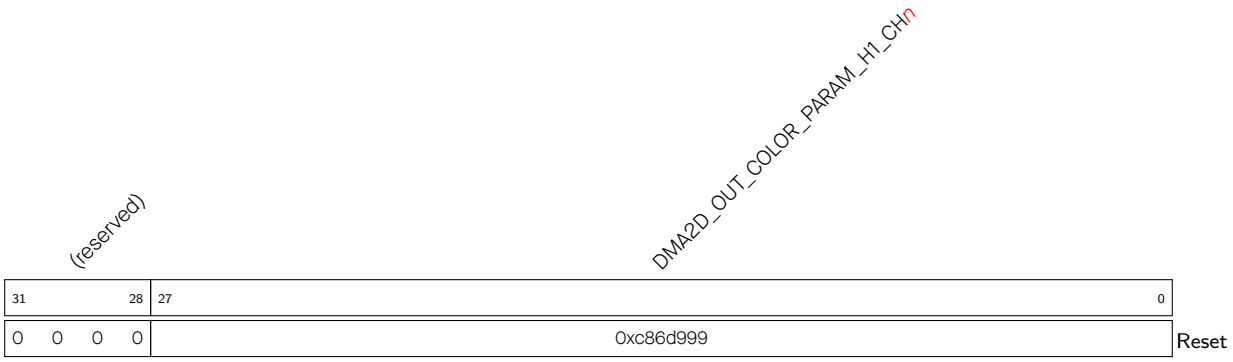
(R/W)

Register 8.9. DMA2D_OUT_COLOR_PARAM0_CH*n*_REG (*n*: 0-3) (0x0050+0x100**n*)



DMA2D_OUT_COLOR_PARAM_H0_CH*n* Configures the coefficient A and B for the most significant byte of the input of color space conversion's second stage for TX channel *n*. (R/W)

Register 8.10. DMA2D_OUT_COLOR_PARAM1_CH*n*_REG (*n*: 0-3) (0x0054+0x100**n*)



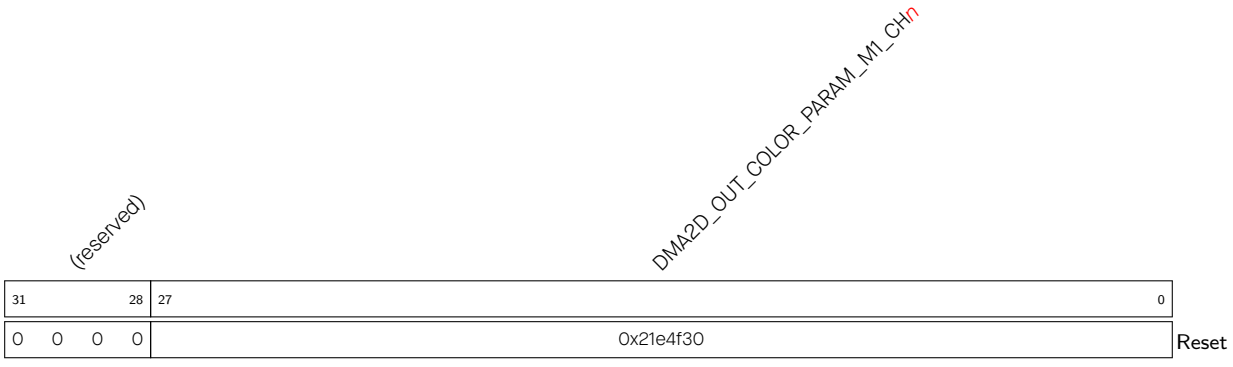
DMA2D_OUT_COLOR_PARAM_H1_CH*n* Configures the coefficient C and D for the most significant byte of the input of color space conversion's second stage for TX channel *n*. (R/W)

Register 8.11. DMA2D_OUT_COLOR_PARAM2_CH*n*_REG (*n*: 0-3) (0x0058+0x100**n*)



DMA2D_OUT_COLOR_PARAM_M0_CH*n* Configures the coefficient A and B for the middle byte of the input of color space conversion's second stage for TX channel *n*. (R/W)

Register 8.12. DMA2D_OUT_COLOR_PARAM3_CH*n*_REG (*n*: 0-3) (0x005C+0x100**n*)



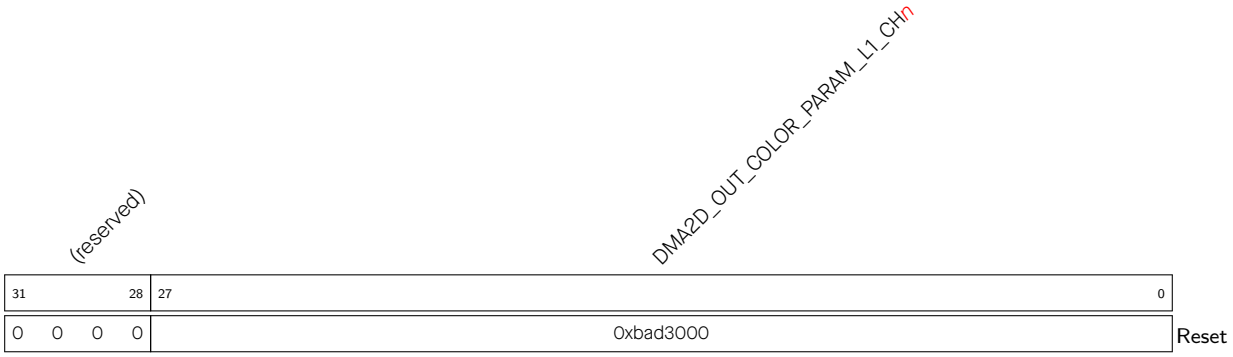
DMA2D_OUT_COLOR_PARAM_M1_CH*n* Configures the coefficient C and D for the middle byte of the input of color space conversion's second stage for TX channel *n*. (R/W)

Register 8.13. DMA2D_OUT_COLOR_PARAM4_CH*n*_REG (*n*: 0-3) (0x0060+0x100n*)**



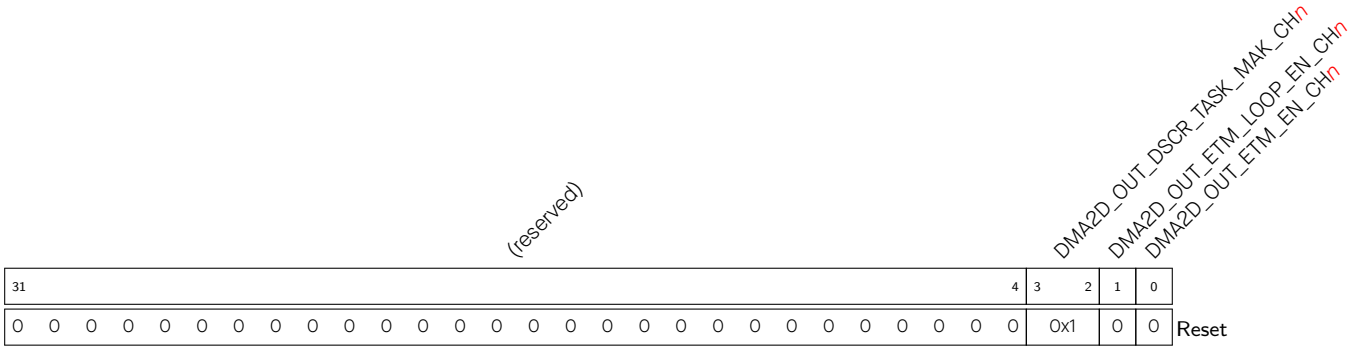
DMA2D_OUT_COLOR_PARAM_LO_CH*n* Configures the coefficient A and B for the least significant byte of the input of color space conversion's second stage for TX channel *n*. (R/W)

Register 8.14. DMA2D_OUT_COLOR_PARAM5_CH*n*_REG (*n*: 0-3) (0x0064+0x100n*)**



DMA2D_OUT_COLOR_PARAM_L1_CH*n* Configures the coefficient C and D for the least significant byte of the input of color space conversion's second stage for TX channel *n*. (R/W)

Register 8.15. DMA2D_OUT_ETM_CONF_CH*n*_REG (*n*: 0-3) (0x0068+0x100n*)**



DMA2D_OUT_ETM_EN_CH*n* Configures whether to enable the ETM function for TX channel *n*.

0: Disable

1: Enable

(R/W)

DMA2D_OUT_ETM_LOOP_EN_CH*n* Configures whether to use the ETM task to indicate the processing of the next descriptor for TX channel *n*.

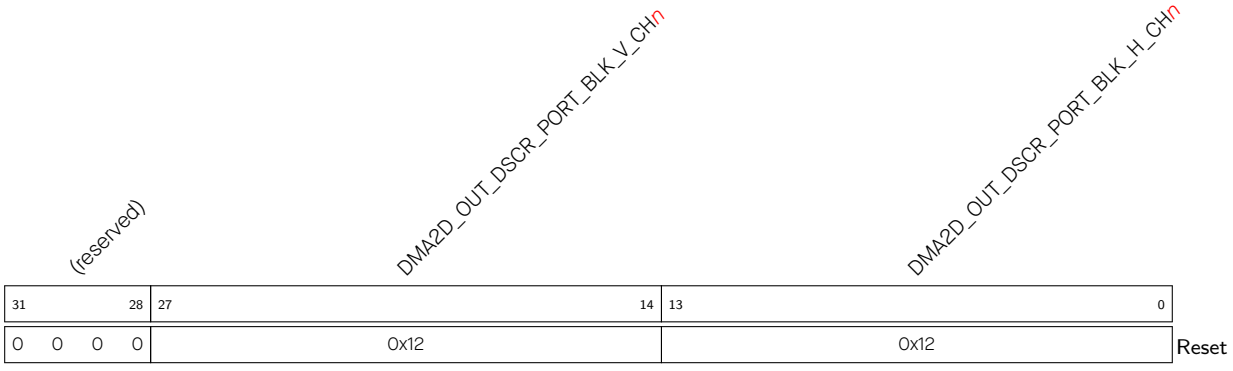
0: Not use ETM task

1: Use ETM task

(R/W)

DMA2D_OUT_DSCR_TASK_MAK_CH*n* Configures the maximum number of tasks that can be cached for TX channel *n*. (R/W)

Register 8.16. DMA2D_OUT_DSCR_PORT_BLK_CH*n*_REG (*n*: 0-3) (0x006C+0x100n*)**



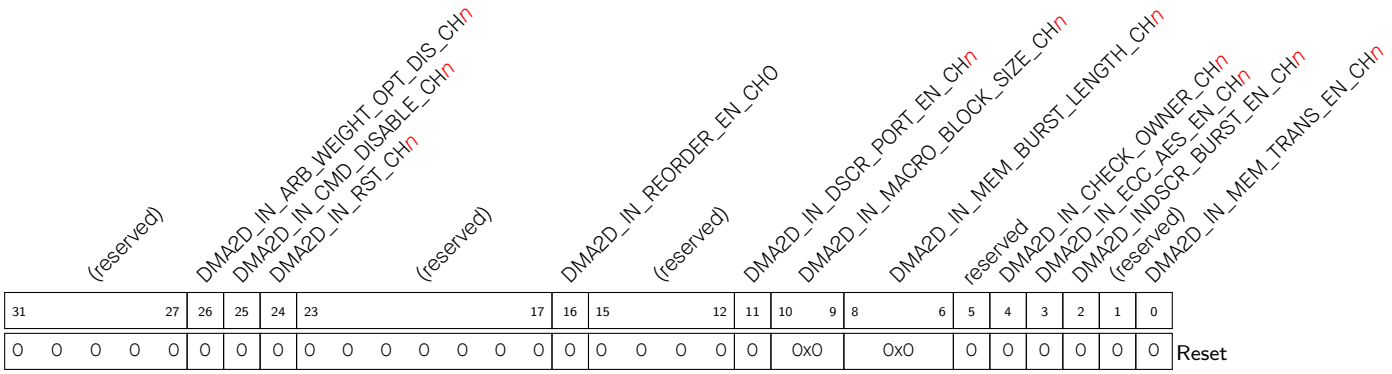
DMA2D_OUT_DSCR_PORT_BLK_H_CH*n* Configures the horizontal width of the basic unit for TX channel *n* in DSCR-PORT mode.

Measurement unit: pixels. (R/W)

DMA2D_OUT_DSCR_PORT_BLK_V_CH*n* Configures the vertical height of the basic unit for TX channel *n* in DSCR-PORT mode.

Measurement unit: pixels. (R/W)

Register 8.17. DMA2D_IN_CONF0_CH*n*_REG (*n*: 0-2) (0x0500+0x100**n*)


(reserved)
DMA2D_IN_ARB_WEIGHT_OPT_DIS_CHn
DMA2D_IN_CMD_DISABLE_CHn
DMA2D_IN_RST_CHn
(reserved)
DMA2D_IN_REORDER_EN_CH0
(reserved)
DMA2D_IN_DSCR_PORT_EN_CHn
DMA2D_IN_MACRO_BLOCK_SIZE_CHn
DMA2D_IN_MEM_BURST_LENGTH_CHn
reserved
DMA2D_IN_CHECK_OWNER_CHn
DMA2D_IN_ECC_AES_EN_CHn
DMA2D_INDSCR_BURST_EN_CHn
(reserved)
DMA2D_IN_MEM_TRANS_EN_CHn
31 27 26 25 24 23 17 16 15 12 11 10 9 8 6 5 4 3 2 1 0
0 0 0

Register 8.17. DMA2D_IN_CONF0_CH*n*_REG (*n*: 0-2) (0x0500+0x100**n*)

Continued from the previous page...

DMA2D_IN_MACRO_BLOCK_SIZE_CH*n* Configures macroblock size for RX channel *n*. Used only in 2D-MOD1 mode.
0: 8 * 8 pixels
1: 8 * 16 pixels
2: 16 * 16 pixels
3: No macroblock
(R/W)

DMA2D_IN_DSCR_PORT_EN_CH*n* Configures whether to enable the DSCR-PORT mode for RX channel *n*.
0: Disable
1: Enable
(R/W)

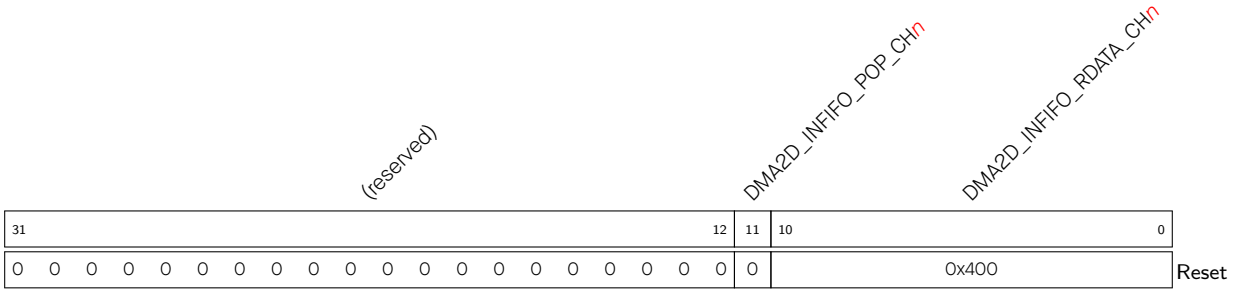
DMA2D_IN_REORDER_EN_CH0 Configures whether to enable macroblock reordering for RX channel 0.
0: Disable
1: Enable
(R/W)

DMA2D_IN_RST_CH*n* Configures whether to reset RX channel *n*.
0: Invalid
1: Reset
(R/W)

DMA2D_IN_CMD_DISABLE_CH*n* Configures reset command for RX channel *n*.
0: Set 0 after reset release
1: Pause transfers before reset
(R/W)

DMA2D_IN_ARB_WEIGHT_OPT_DIS_CH*n* Configures whether to enable weight optimization for RX channel *n*.
0: Enable
1: Disable
(R/W)

Register 8.18. DMA2D_IN_POP_CH*n*_REG (*n*: 0-2) (0x0518+0x100n*)**



DMA2D_INFIFO_RDATA_CH*n* Represents the data popped from 2D-DMA RX FIFO. (RO)

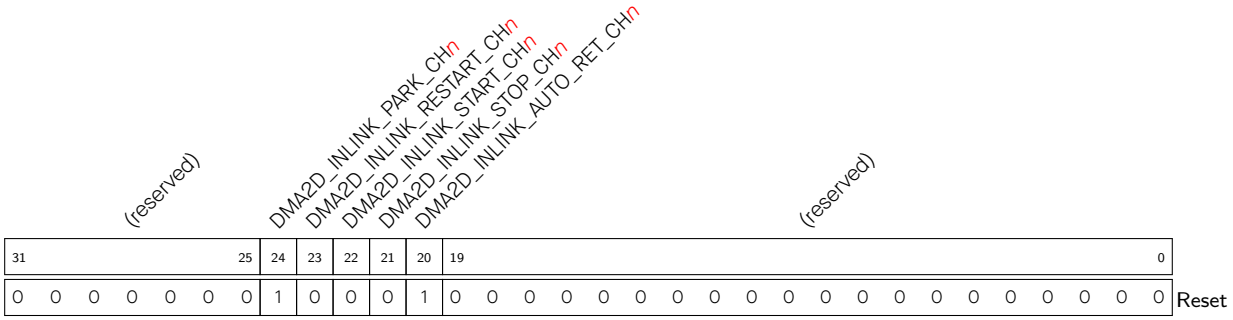
DMA2D_INFIFO_POP_CH*n* Configures whether to pop data from 2D-DMA RX FIFO.

0: Not pop

1: Pop

(R/W/SC)

Register 8.19. DMA2D_IN_LINK_CONF_CH*n*_REG (*n*: 0-2) (0x051C+0x100**n*)



DMA2D_INLINK_AUTO_RET_CH*n* Configure the value of the owner bit written back to the receive descriptor.

0: Write back 0

1: Write back 1

(R/W)

DMA2D_INLINK_STOP_CH*n* Configures whether to stop RX channel *n* from receiving data.

0: Invalid. No effect

1: Stop

(R/W/SC)

DMA2D_INLINK_START_CH*n* Configures whether to enable RX channel *n* for data transfer.

0: Disable

1: Enable

(R/W/SC)

DMA2D_INLINK_RESTART_CH*n* Configures whether to restart RX channel *n* for 2D-DMA transfer.

0: Invalid. No effect

1: Restart

(R/W/SC)

DMA2D_INLINK_PARK_CH*n* Represents the status of the receive descriptor's FSM.

0: Running

1: Idle

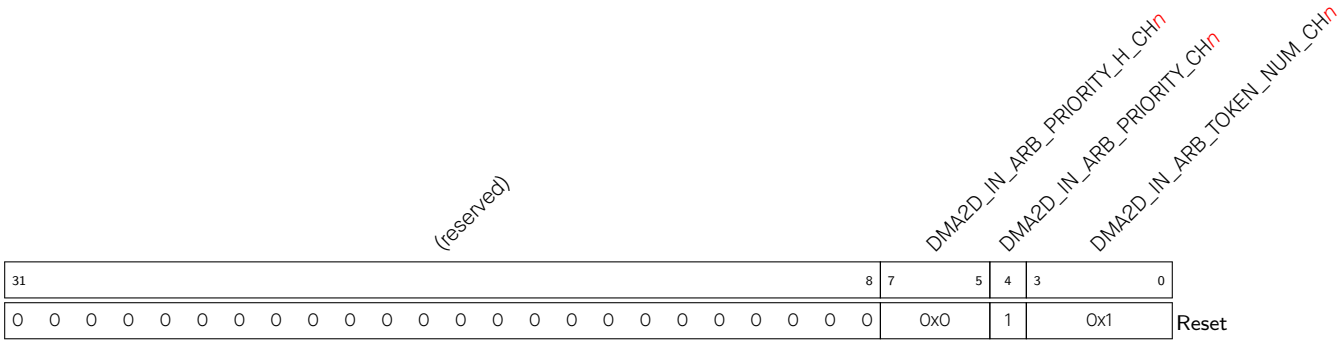
(RO)

Register 8.20. DMA2D_IN_LINK_ADDR_CH*n*_REG (*n*: 0-2) (0x0520+0x100n*)**



DMA2D_INLINK_ADDR_CH*n* Represents the first receive descriptor's address. (R/W)

Register 8.21. DMA2D_IN_ARB_CH*n*_REG (*n*: 0-2) (0x0540+0x100n*)**



DMA2D_IN_ARB_TOKEN_NUM_CH*n* Configures the weight (i.e the number of tokens) of RX channel *n*.

Value range: 0 ~ 15. (R/W)

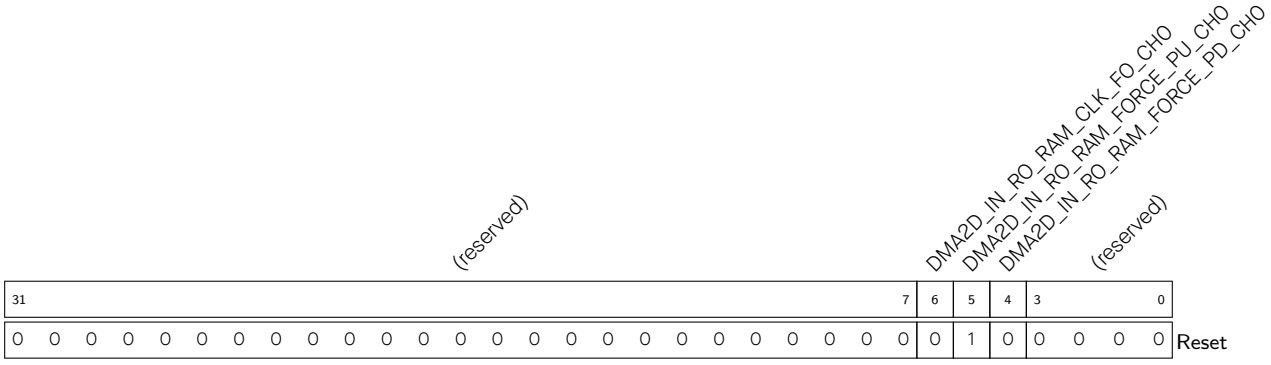
DMA2D_IN_ARB_PRIORITY_CH*n* Configures the lower two bits of RX channel *n* priority.

Together with DMA2D_IN_ARB_PRIORITY_H_CH*n*, this field forms a 4-bit priority value for TX channel *n*, ranging from 0 (lowest) to 15 (highest). (R/W)

DMA2D_IN_ARB_PRIORITY_H_CH*n* Configures the higher two bits of TX channel *n* priority.

Together with DMA2D_IN_ARB_PRIORITY_CH*n*, this field forms a 4-bit priority value for TX channel *n*, ranging from 0 (lowest) to 15 (highest). (R/W)

Register 8.22. DMA2D_IN_RO_PD_CONF_CH0_REG (0x0548)



DMA2D_IN_RO_RAM_FORCE_PD_CH0 Configures whether to force power down the macroblock reordering RAM for RX channel 0.

0: Not force power down

1: Force power down

(R/W)

DMA2D_IN_RO_RAM_FORCE_PU_CH0 Configures whether to force power up the macroblock reordering RAM for RX channel 0.

0: Not force power up

1: Force power up

(R/W)

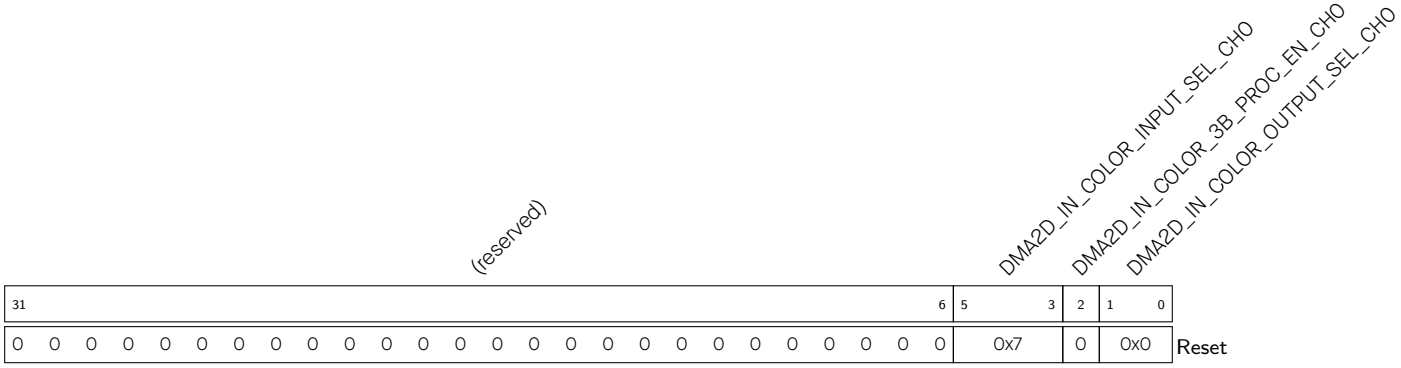
DMA2D_IN_RO_RAM_CLK_FO_CH0 Configures the clock gating when RX channel 0 accessing the 2D-DMA RAM.

0: Use the clock gate

1: Force the clock on and bypass the clock gate

(R/W)

Register 8.23. DMA2D_IN_COLOR_CONVERT_CH0_REG (0x054C)



DMA2D_IN_COLOR_OUTPUT_SEL_CH0 Configures the third stage output of color space conversion for RX channel 0.

0: RGB888 to RGB565

1: Output directly

(R/W)

DMA2D_IN_COLOR_3B_PROC_EN_CH0 Configures whether to enable the second stage of color space conversion for RX channel 0.

0: Disable

1: Enable

(R/W)

DMA2D_IN_COLOR_INPUT_SEL_CH0 Configures the first stage output of color space conversion for RX channel 0.

0: YUV422/420 to YUV444

1: YUV422

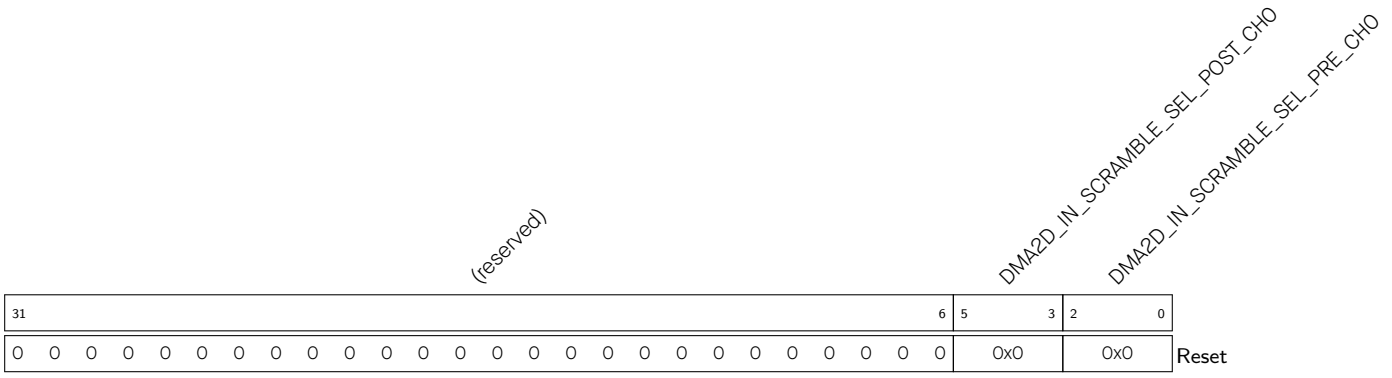
2: YUV444/420

7: Disable color space conversion

Others: Invalid

(R/W)

Register 8.24. DMA2D_IN_SCRAMBLE_CH0_REG (0x0550)



DMA2D_IN_SCRAMBLE_SEL_PRE_CH0 Configures the byte order scrambling before the color space conversion for RX channel 0.

0: BYTE2-1-0

1: BYTE2-0-1

2: BYTE1-0-2

3: BYTE1-2-0

4: BYTE0-2-1

5: BYTE0-1-2

Others: Invalid

(R/W)

DMA2D_IN_SCRAMBLE_SEL_POST_CH0 Configures the byte order scrambling after the color space conversion for RX channel 0.

0: BYTE2-1-0

1: BYTE2-0-1

2: BYTE1-0-2

3: BYTE1-2-0

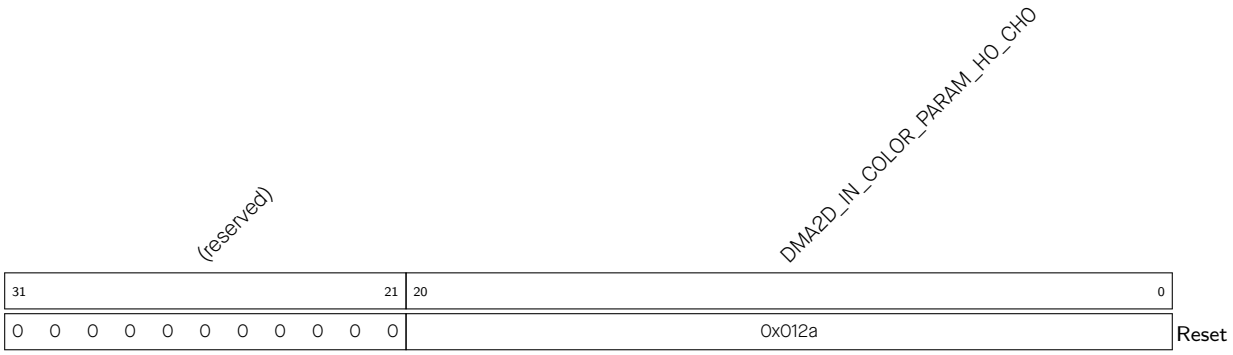
4: BYTE0-2-1

5: BYTE0-1-2

Others: Invalid

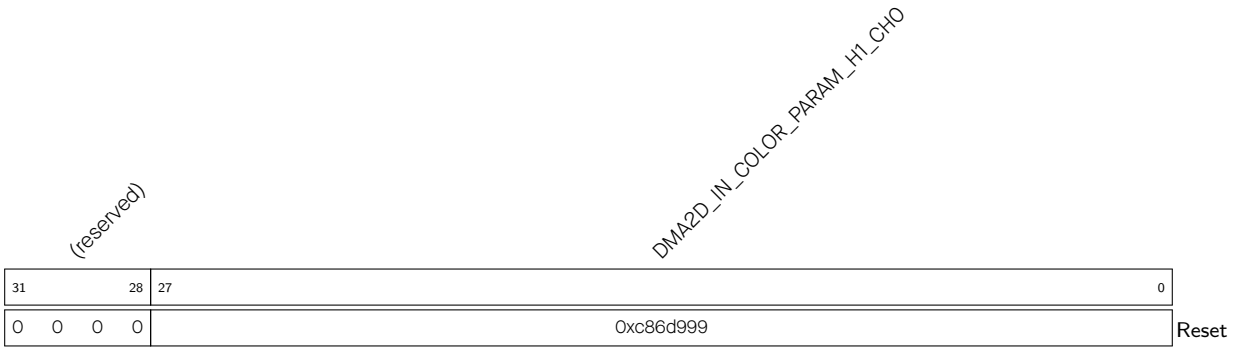
(R/W)

Register 8.25. DMA2D_IN_COLOR_PARAM0_CH0_REG (0x0554)



DMA2D_IN_COLOR_PARAM_H0_CH0 Configures the coefficient A and B for the most significant byte of the input of color space conversion's second stage for RX channel 0. (R/W)

Register 8.26. DMA2D_IN_COLOR_PARAM1_CH0_REG (0x0558)



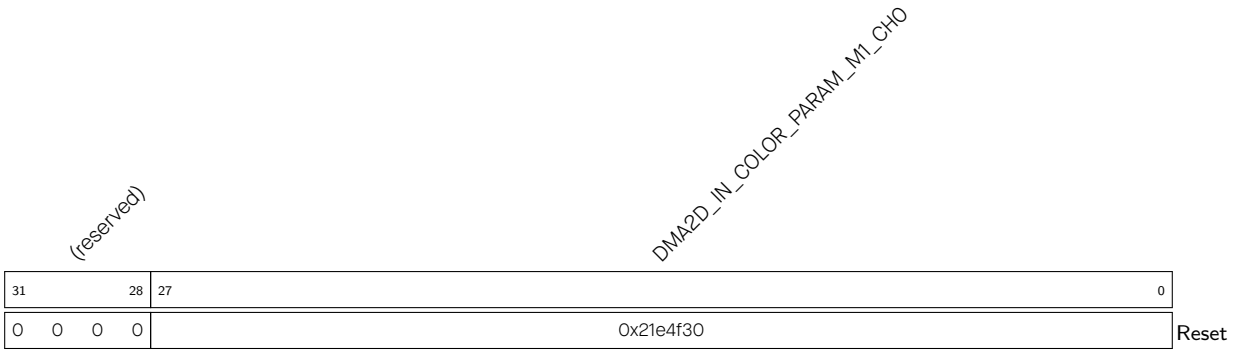
DMA2D_IN_COLOR_PARAM_H1_CH0 Configures the coefficient C and D for the most significant byte of the input of color space conversion's second stage for RX channel 0. (R/W)

Register 8.27. DMA2D_IN_COLOR_PARAM2_CH0_REG (0x055C)



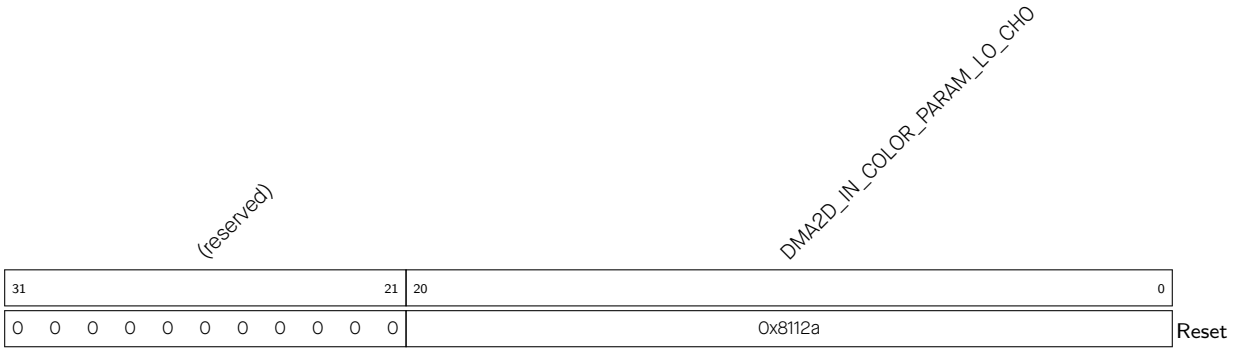
DMA2D_IN_COLOR_PARAM_M0_CH0 Configures the coefficient A and B for the middle byte of the input of color space conversion's second stage for RX channel 0. (R/W)

Register 8.28. DMA2D_IN_COLOR_PARAM3_CH0_REG (0x0560)



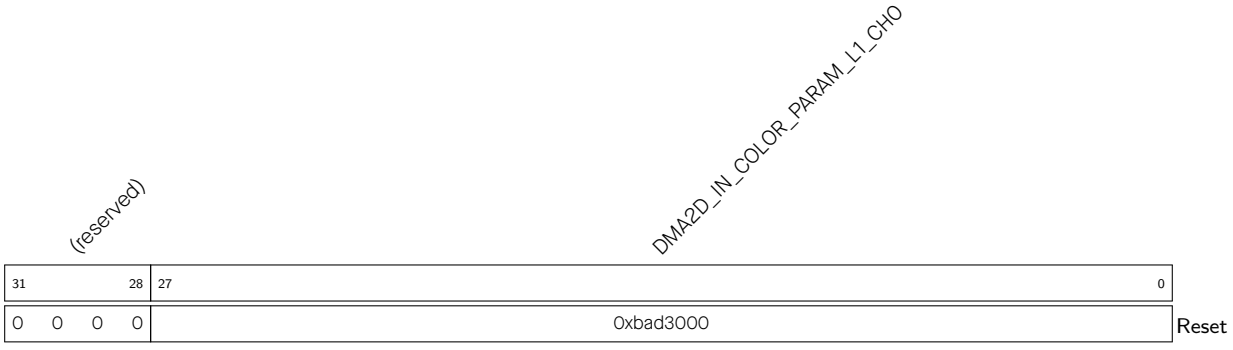
DMA2D_IN_COLOR_PARAM_M1_CH0 Configures the coefficient C and D for the middle byte of the input of color space conversion's second stage for RX channel 0. (R/W)

Register 8.29. DMA2D_IN_COLOR_PARAM4_CH0_REG (0x0564)



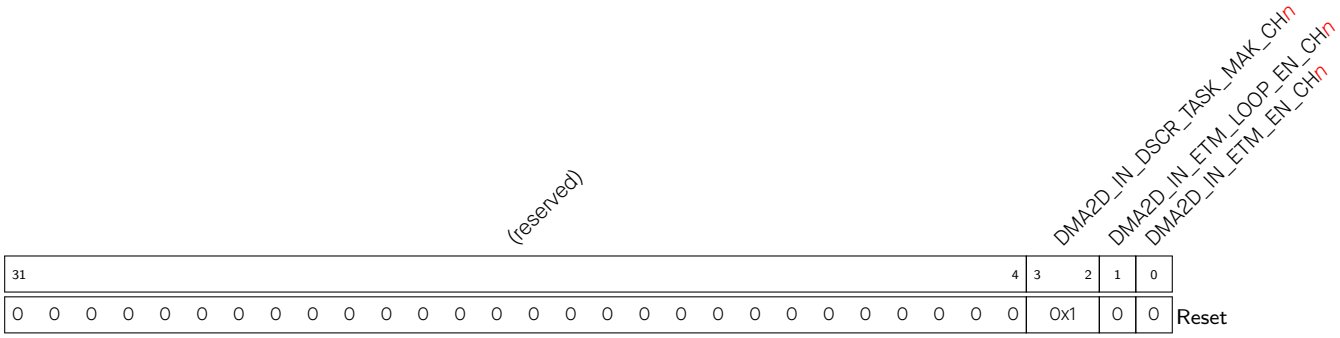
DMA2D_IN_COLOR_PARAM_L0_CH0 Configures the coefficient A and B for the least significant byte of the input of color space conversion's second stage for RX channel 0. (R/W)

Register 8.30. DMA2D_IN_COLOR_PARAM5_CH0_REG (0x0568)



DMA2D_IN_COLOR_PARAM_L1_CH0 Configures the coefficient C and D for the least significant byte of the input of color space conversion's second stage for RX channel 0. (R/W)

Register 8.31. DMA2D_IN_ETM_CONF_CH*n*_REG (*n*: 0-2) (0x056C+0x100n*)**



DMA2D_IN_ETM_EN_CH*n* Configures whether to enable the ETM function for RX channel *n*.

0: Disable

1: Enable

(R/W)

DMA2D_IN_ETM_LOOP_EN_CH*n* Configures whether to use the ETM task to indicate the processing of the next descriptor for RX channel *n*.

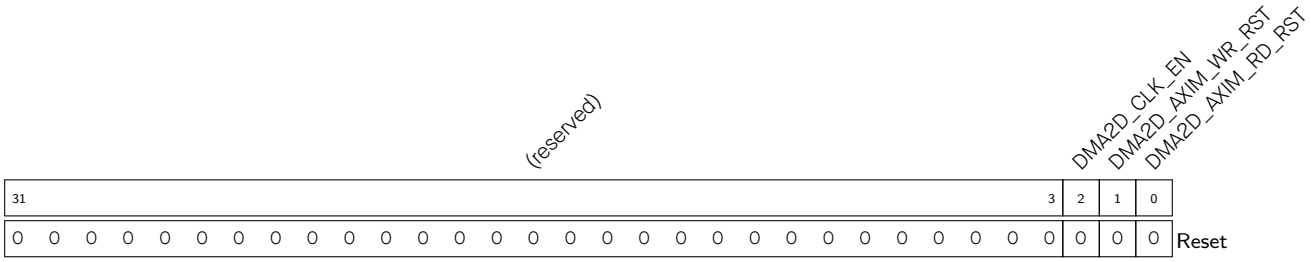
0: Not use ETM task

1: Use ETM task

(R/W)

DMA2D_IN_DSCR_TASK_MAK_CH*n* Configures the maximum number of tasks that can be cached for RX channel *n*. (R/W)

Register 8.32. DMA2D_RST_CONF_REG (0x0A04)



DMA2D_AXIM_RD_RST Configures whether to reset the AXI master read data FIFO.

0: No effect

1: Reset

(R/W)

DMA2D_AXIM_WR_RST Configures whether to reset the AXI master write data FIFO.

0: No effect

1: Reset

(R/W)

DMA2D_CLK_EN Configures the clock for registers.

0: Support clock only when application writes registers

1: Force clock on for register

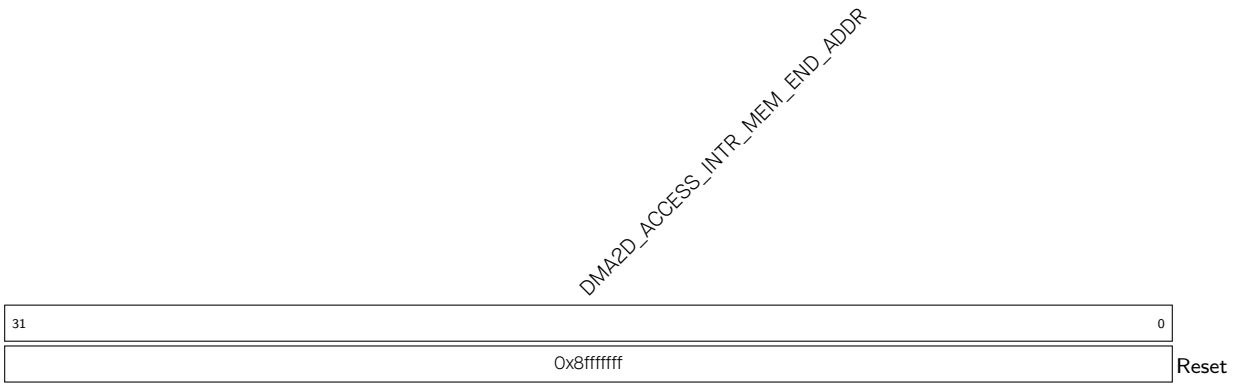
(R/W)

Register 8.33. DMA2D_INTR_MEM_START_ADDR_REG (0x0A08)



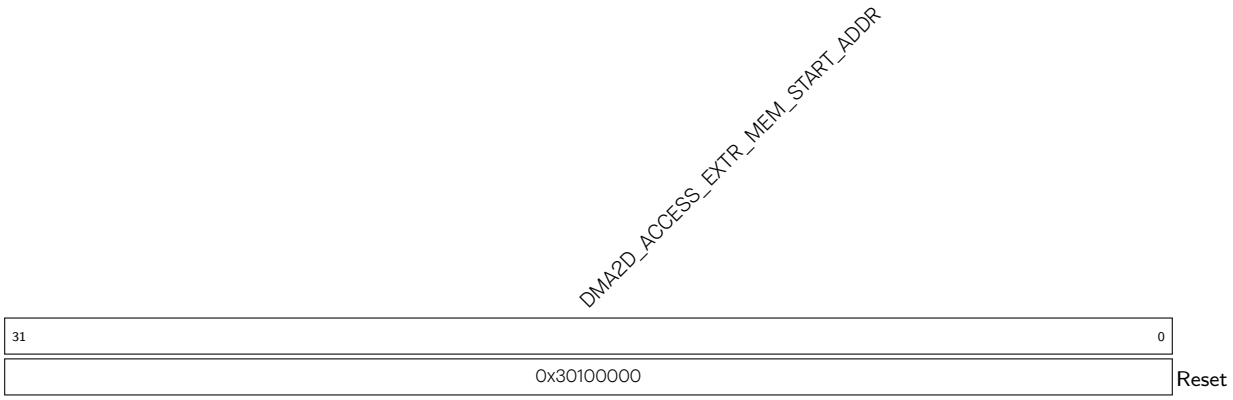
DMA2D_ACCESS_INTR_MEM_START_ADDR Configures the start address of the accessible internal address space. (R/W)

Register 8.34. DMA2D_INTR_MEM_END_ADDR_REG (0x0A0C)



DMA2D_ACCESS_INTR_MEM_END_ADDR Configures the end address of the accessible internal address space. Accessing an address beyond this range will result in a descriptor error. (R/W)

Register 8.35. DMA2D_EXTR_MEM_START_ADDR_REG (0x0A10)



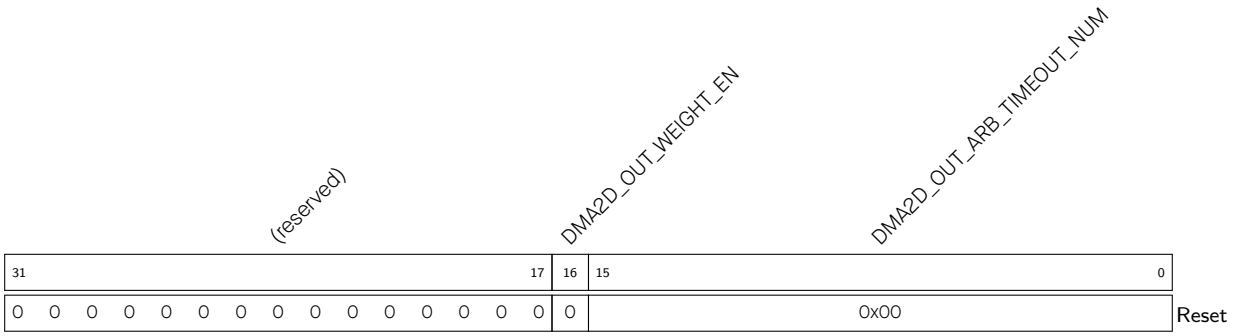
DMA2D_ACCESS_EXTR_MEM_START_ADDR Configures the start address of the accessible external address space. (R/W)

Register 8.36. DMA2D_EXTR_MEM_END_ADDR_REG (0x0A14)



DMA2D_ACCESS_EXTR_MEM_END_ADDR Configures the end address of the accessible external address space. Accessing an address beyond this range will result in a descriptor error. (R/W)

Register 8.37. DMA2D_OUT_ARB_CONFIG_REG (0x0A18)



DMA2D_OUT_ARB_TIMEOUT_NUM Configures the time slot for TX. Measurement unit: AXI bus clock cycle. (R/W)

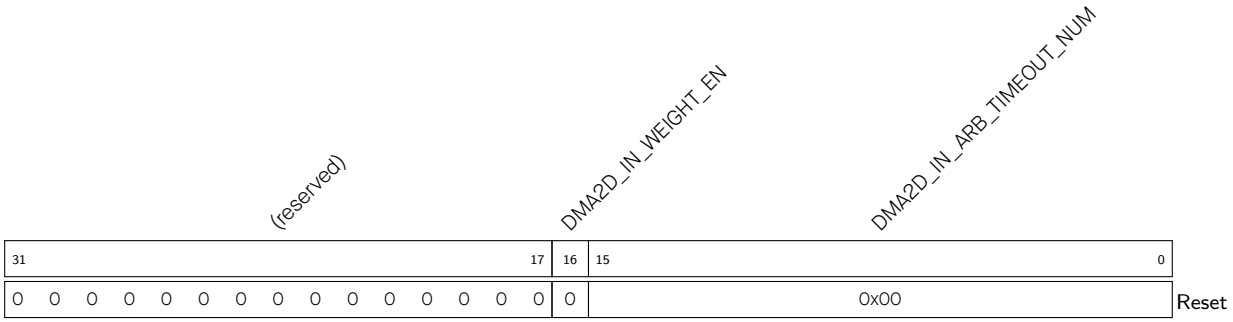
DMA2D_OUT_WEIGHT_EN Configures whether to enable weight arbitration for TX.

0: Disable

1: Enable

(R/W)

Register 8.38. DMA2D_IN_ARB_CONFIG_REG (0x0A1C)



DMA2D_IN_ARB_TIMEOUT_NUM Configures the time slot for RX. Measurement unit: AXI bus clock cycle. (R/W)

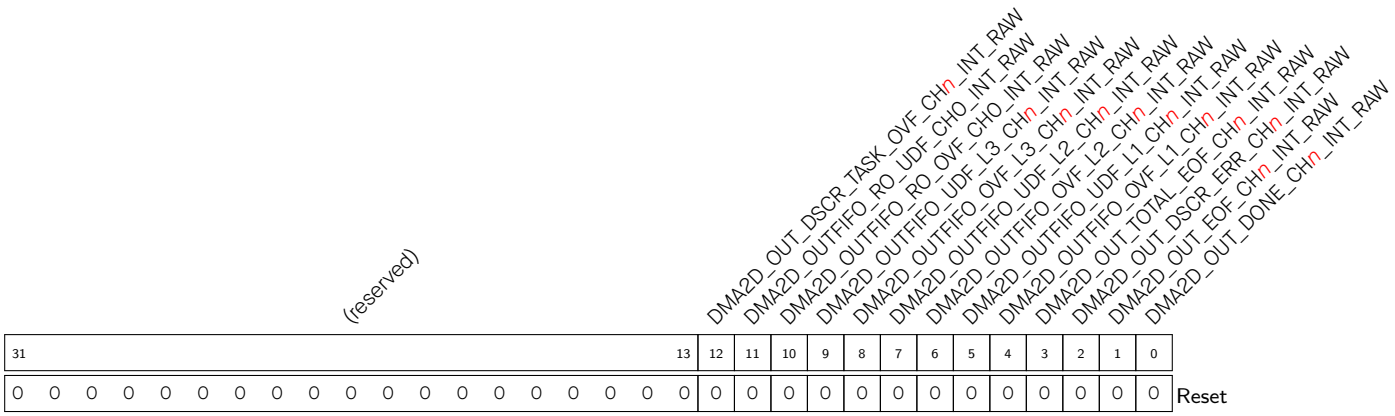
DMA2D_IN_WEIGHT_EN Configures whether to enable weight arbitration for RX.

0: Disable

1: Enable

(R/W)

Register 8.39. DMA2D_OUT_INT_RAW_CH*n*_REG (*n*: 0-3) (0x0004+0x100n*)**



DMA2D_OUT_DONE_CH*n*_INT_RAW The raw interrupt status bit of DMA2D_OUT_DONE_CH*n*_INT. (R/WTC/SS)

DMA2D_OUT_EOF_CH*n*_INT_RAW The raw interrupt status of DMA2D_OUT_EOF_CH*n*_INT. (R/WTC/SS)

DMA2D_OUT_DSCR_ERR_CH*n*_INT_RAW The raw interrupt status of DMA2D_OUT_DSCR_ERR_CH*n*_INT. (R/WTC/SS)

DMA2D_OUT_TOTAL_EOF_CH*n*_INT_RAW The raw interrupt status of DMA2D_OUT_TOTAL_EOF_CH*n*_INT. (R/WTC/SS)

DMA2D_OUTFIFO_OVF_L1_CH*n*_INT_RAW The raw interrupt status of DMA2D_OUTFIFO_OVF_L1_CH*n*_INT. (R/WTC/SS)

DMA2D_OUTFIFO_UDF_L1_CH*n*_INT_RAW The raw interrupt status of DMA2D_OUTFIFO_UDF_L1_CH*n*_INT. (R/WTC/SS)

DMA2D_OUTFIFO_OVF_L2_CH*n*_INT_RAW The raw interrupt status of DMA2D_OUTFIFO_OVF_L2_CH*n*_INT. (R/WTC/SS)

DMA2D_OUTFIFO_UDF_L2_CH*n*_INT_RAW The raw interrupt status of DMA2D_OUTFIFO_UDF_L2_CH*n*_INT. (R/WTC/SS)

DMA2D_OUTFIFO_OVF_L3_CH*n*_INT_RAW The raw interrupt status of DMA2D_OUTFIFO_OVF_L3_CH*n*_INT. (R/WTC/SS)

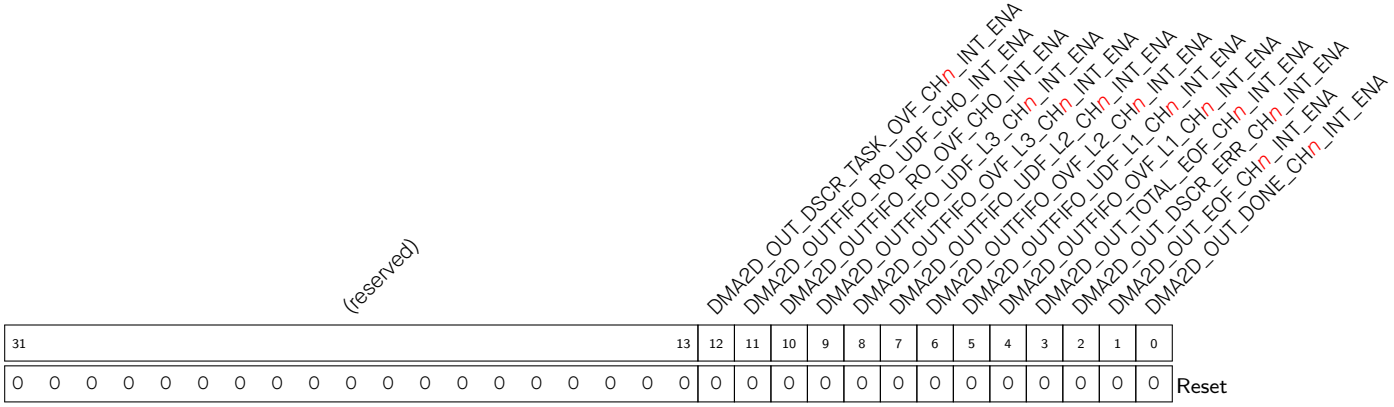
DMA2D_OUTFIFO_UDF_L3_CH*n*_INT_RAW The raw interrupt status of DMA2D_OUTFIFO_UDF_L3_CH*n*_INT. (R/WTC/SS)

DMA2D_OUTFIFO_RO_OVF_CHO_INT_RAW The raw interrupt status of DMA2D_OUTFIFO_RO_OVF_CHO_INT. (R/WTC/SS)

DMA2D_OUTFIFO_RO_UDF_CHO_INT_RAW The raw interrupt status of DMA2D_OUTFIFO_RO_UDF_CHO_INT. (R/WTC/SS)

DMA2D_OUT_DSCR_TASK_OVF_CH*n*_INT_RAW The raw interrupt status of DMA2D_OUT_DSCR_TASK_OVF_CH*n*_INT. (R/WTC/SS)

Register 8.40. DMA2D_OUT_INT_ENA_CH*n*_REG (*n*: 0-3) (0x0008+0x100**n*)



DMA2D_OUT_DONE_CH*n*_INT_ENA Write 1 to enable the DMA2D_OUT_DONE_CH*n*_INT. (R/W)

DMA2D_OUT_EOF_CH*n*_INT_ENA Write 1 to enable the DMA2D_OUT_EOF_CH*n*_INT. (R/W)

DMA2D_OUT_DSCR_ERR_CH*n*_INT_ENA Write 1 to enable the DMA2D_OUT_DSCR_ERR_CH*n*_INT. (R/W)

DMA2D_OUT_TOTAL_EOF_CH*n*_INT_ENA Write 1 to enable the DMA2D_OUT_TOTAL_EOF_CH*n*_INT. (R/W)

DMA2D_OUTFIFO_OVF_L1_CH*n*_INT_ENA Write 1 to enable the DMA2D_OUTFIFO_OVF_L1_CH*n*_INT. (R/W)

DMA2D_OUTFIFO_UDF_L1_CH*n*_INT_ENA Write 1 to enable the DMA2D_OUTFIFO_UDF_L1_CH*n*_INT. (R/W)

DMA2D_OUTFIFO_OVF_L2_CH*n*_INT_ENA Write 1 to enable the DMA2D_OUTFIFO_OVF_L2_CH*n*_INT. (R/W)

DMA2D_OUTFIFO_UDF_L2_CH*n*_INT_ENA Write 1 to enable the DMA2D_OUTFIFO_UDF_L2_CH*n*_INT. (R/W)

DMA2D_OUTFIFO_OVF_L3_CH*n*_INT_ENA Write 1 to enable the DMA2D_OUTFIFO_OVF_L3_CH*n*_INT. (R/W)

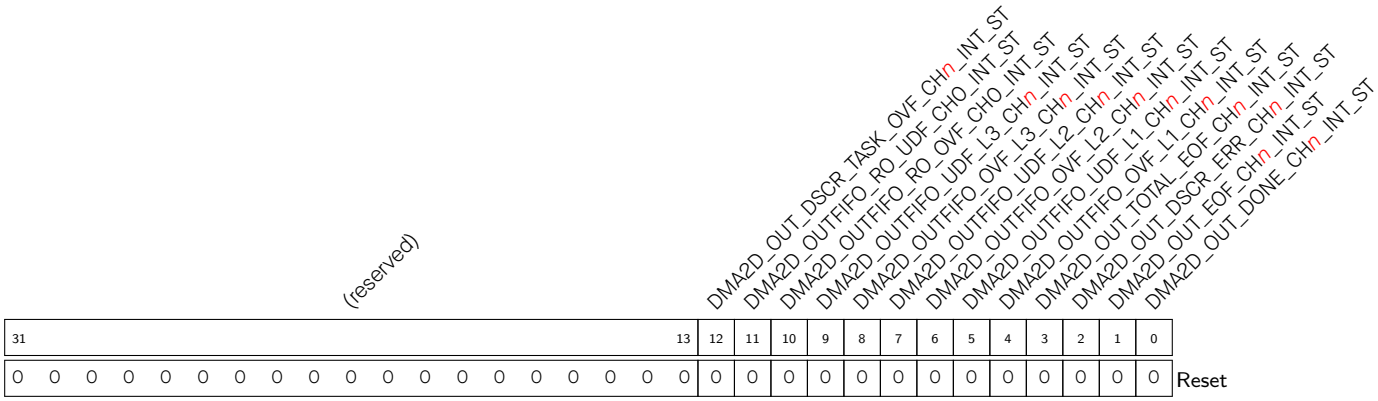
DMA2D_OUTFIFO_UDF_L3_CH*n*_INT_ENA Write 1 to enable the DMA2D_OUTFIFO_UDF_L3_CH*n*_INT. (R/W)

DMA2D_OUTFIFO_RO_OVF_CHO_INT_ENA Write 1 to enable the DMA2D_OUTFIFO_RO_OVF_CHO_INT. (R/W)

DMA2D_OUTFIFO_RO_UDF_CHO_INT_ENA Write 1 to enable the DMA2D_OUTFIFO_RO_UDF_CHO_INT. (R/W)

DMA2D_OUT_DSCR_TASK_OVF_CH*n*_INT_ENA Write 1 to enable the DMA2D_OUT_DSCR_TASK_OVF_CH*n*_INT. (R/W)

Register 8.41. DMA2D_OUT_INT_ST_CH*n*_REG (*n*: 0-3) (0x000C+0x100**n*)



DMA2D_OUT_DONE_CH*n*_INT_ST The masked interrupt status of DMA2D_OUT_DONE_CH*n*_INT. (RO)

DMA2D_OUT_EOF_CH*n*_INT_ST The masked interrupt status of DMA2D_OUT_EOF_CH*n*_INT. (RO)

DMA2D_OUT_DSCR_ERR_CH*n*_INT_ST The masked interrupt status of DMA2D_OUT_DSCR_ERR_CH*n*_INT. (RO)

DMA2D_OUT_TOTAL_EOF_CH*n*_INT_ST The masked interrupt status of DMA2D_OUT_TOTAL_EOF_CH*n*_INT. (RO)

DMA2D_OUTFIFO_OVF_L1_CH*n*_INT_ST The masked interrupt status of DMA2D_OUTFIFO_OVF_L1_CH*n*_INT. (RO)

DMA2D_OUTFIFO_UDF_L1_CH*n*_INT_ST The masked interrupt status of DMA2D_OUTFIFO_UDF_L1_CH*n*_INT. (RO)

DMA2D_OUTFIFO_OVF_L2_CH*n*_INT_ST The masked interrupt status of DMA2D_OUTFIFO_OVF_L2_CH*n*_INT. (RO)

DMA2D_OUTFIFO_UDF_L2_CH*n*_INT_ST The masked interrupt status of DMA2D_OUTFIFO_UDF_L2_CH*n*_INT. (RO)

DMA2D_OUTFIFO_OVF_L3_CH*n*_INT_ST The masked interrupt status of DMA2D_OUTFIFO_OVF_L3_CH*n*_INT. (RO)

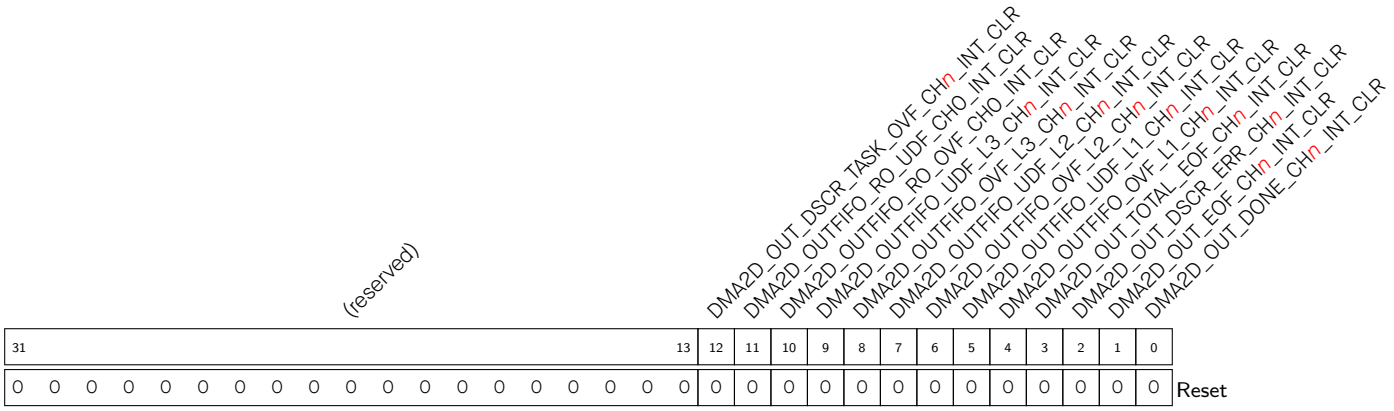
DMA2D_OUTFIFO_UDF_L3_CH*n*_INT_ST The masked interrupt status of DMA2D_OUTFIFO_UDF_L3_CH*n*_INT. (RO)

DMA2D_OUTFIFO_RO_OVF_CHO_INT_ST The masked interrupt status of DMA2D_OUTFIFO_RO_OVF_CHO_INT. (RO)

DMA2D_OUTFIFO_RO_UDF_CHO_INT_ST The masked interrupt status of DMA2D_OUTFIFO_RO_UDF_CHO_INT. (RO)

DMA2D_OUT_DSCR_TASK_OVF_CH*n*_INT_ST The masked interrupt status of DMA2D_OUT_DSCR_TASK_OVF_CH*n*_INT. (RO)

Register 8.42. DMA2D_OUT_INT_CLR_CH*n*_REG (*n*: 0-3) (0x0010+0x100***n*)



DMA2D_OUT_DONE_CH*n*_INT_CLR Write 1 to clear the DMA2D_OUT_DONE_CH*n*_INT. (WT)

DMA2D_OUT_EOF_CH*n*_INT_CLR Write 1 to clear the DMA2D_OUT_EOF_CH*n*_INT. (WT)

DMA2D_OUT_DSCR_ERR_CH*n*_INT_CLR Write 1 to clear the DMA2D_OUT_DSCR_ERR_CH*n*_INT. (WT)

DMA2D_OUT_TOTAL_EOF_CH*n*_INT_CLR Write 1 to clear the DMA2D_OUT_TOTAL_EOF_CH*n*_INT. (WT)

DMA2D_OUTFIFO_OVF_L1_CH*n*_INT_CLR Write 1 to clear the DMA2D_OUTFIFO_OVF_L1_CH*n*_INT. (WT)

DMA2D_OUTFIFO_UDF_L1_CH*n*_INT_CLR Write 1 to clear the DMA2D_OUTFIFO_UDF_L1_CH*n*_INT. (WT)

DMA2D_OUTFIFO_OVF_L2_CH*n*_INT_CLR Write 1 to clear the DMA2D_OUTFIFO_OVF_L2_CH*n*_INT. (WT)

DMA2D_OUTFIFO_UDF_L2_CH*n*_INT_CLR Write 1 to clear the DMA2D_OUTFIFO_UDF_L2_CH*n*_INT. (WT)

DMA2D_OUTFIFO_OVF_L3_CH*n*_INT_CLR Write 1 to clear the DMA2D_OUTFIFO_OVF_L3_CH*n*_INT. (WT)

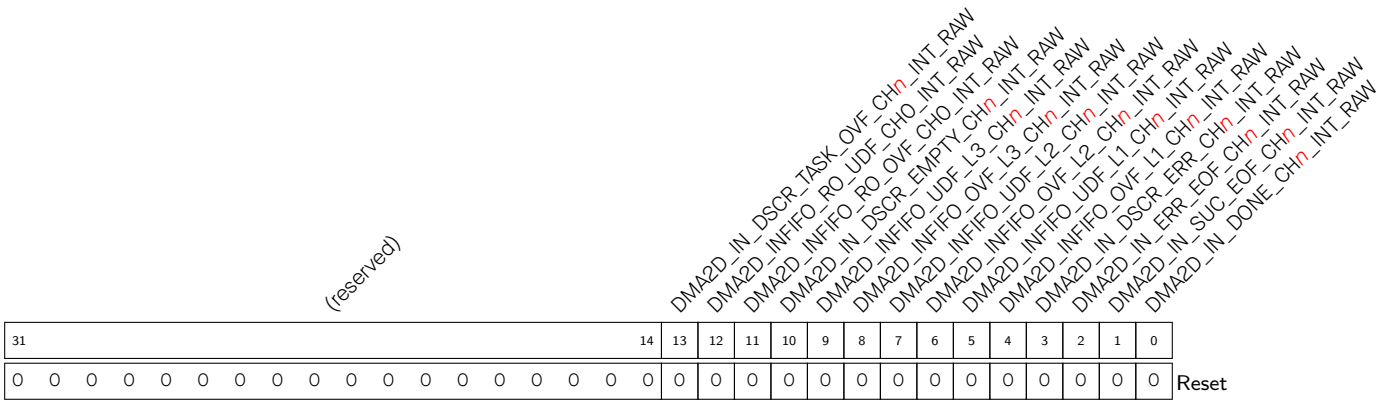
DMA2D_OUTFIFO_UDF_L3_CH*n*_INT_CLR Write 1 to clear the DMA2D_OUTFIFO_UDF_L3_CH*n*_INT. (WT)

DMA2D_OUTFIFO_RO_OVF_CHO_INT_CLR Write 1 to clear the DMA2D_OUTFIFO_RO_OVF_CHO_INT. (WT)

DMA2D_OUTFIFO_RO_UDF_CHO_INT_CLR Write 1 to clear the DMA2D_OUTFIFO_RO_UDF_CHO_INT. (WT)

DMA2D_OUT_DSCR_TASK_OVF_CH*n*_INT_CLR Write 1 to clear the DMA2D_OUT_DSCR_TASK_OVF_CH*n*_INT. (WT)

Register 8.43. DMA2D_IN_INT_RAW_CH*n*_REG (*n*: 0-2) (0x0504+0x100n*)**



DMA2D_IN_DONE_CH*n*_INT_RAW The raw interrupt status of DMA2D_IN_DONE_CH*n*_INT. (R/WTC/SS)

DMA2D_IN_SUC_EOF_CH*n*_INT_RAW The raw interrupt status of DMA2D_IN_SUC_EOF_CH*n*_INT. (R/WTC/SS)

DMA2D_IN_ERR_EOF_CH*n*_INT_RAW The raw interrupt status of DMA2D_IN_ERR_EOF_CH*n*_INT. (R/WTC/SS)

DMA2D_IN_DSCR_ERR_CH*n*_INT_RAW The raw interrupt status of DMA2D_IN_DSCR_ERR_CH*n*_INT. (R/WTC/SS)

DMA2D_INFIFO_OVF_L1_CH*n*_INT_RAW The raw interrupt status of DMA2D_INFIFO_OVF_L1_CH*n*_INT. (R/WTC/SS)

DMA2D_INFIFO_UDF_L1_CH*n*_INT_RAW The raw interrupt status of DMA2D_INFIFO_UDF_L1_CH*n*_INT. (R/WTC/SS)

DMA2D_INFIFO_OVF_L2_CH*n*_INT_RAW The raw interrupt status of DMA2D_INFIFO_OVF_L2_CH*n*_INT. (R/WTC/SS)

DMA2D_INFIFO_UDF_L2_CH*n*_INT_RAW The raw interrupt status of DMA2D_INFIFO_UDF_L2_CH*n*_INT. (R/WTC/SS)

DMA2D_INFIFO_OVF_L3_CH*n*_INT_RAW The raw interrupt status of DMA2D_INFIFO_OVF_L3_CH*n*_INT. (R/WTC/SS)

DMA2D_INFIFO_UDF_L3_CH*n*_INT_RAW The raw interrupt status of DMA2D_INFIFO_UDF_L3_CH*n*_INT. (R/WTC/SS)

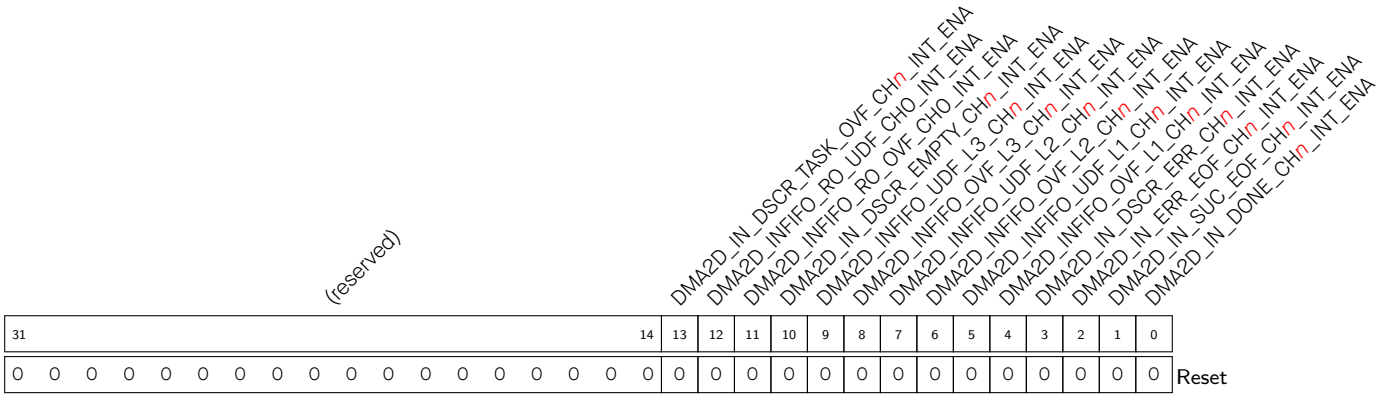
DMA2D_IN_DSCR_EMPTY_CH*n*_INT_RAW The raw interrupt status of DMA2D_IN_DSCR_EMPTY_CH*n*_INT. (R/WTC/SS)

DMA2D_INFIFO_RO_OVF_CHO_INT_RAW The raw interrupt status of DMA2D_INFIFO_RO_OVF_CHO_INT. (R/WTC/SS)

DMA2D_INFIFO_RO_UDF_CHO_INT_RAW The raw interrupt status of DMA2D_INFIFO_RO_UDF_CHO_INT. (R/WTC/SS)

DMA2D_IN_DSCR_TASK_OVF_CH*n*_INT_RAW The raw interrupt status of DMA2D_IN_DSCR_TASK_OVF_CH*n*_INT. (R/WTC/SS)

Register 8.44. DMA2D_IN_INT_ENA_CH*n*_REG (*n*: 0-2) (0x0508+0x100**n*)



DMA2D_IN_DONE_CH*n*_INT_ENA Write 1 to enable the DMA2D_IN_DONE_CH*n*_INT. (R/W)

DMA2D_IN_SUC_EOF_CH*n*_INT_ENA Write 1 to enable the DMA2D_IN_SUC_EOF_CH*n*_INT. (R/W)

DMA2D_IN_ERR_EOF_CH*n*_INT_ENA Write 1 to enable the DMA2D_IN_ERR_EOF_CH*n*_INT. (R/W)

DMA2D_IN_DSCR_ERR_CH*n*_INT_ENA Write 1 to enable the DMA2D_IN_DSCR_ERR_CH*n*_INT. (R/W)

DMA2D_INFIFO_OVF_L1_CH*n*_INT_ENA Write 1 to enable the DMA2D_INFIFO_OVF_L1_CH*n*_INT. (R/W)

DMA2D_INFIFO_UDF_L1_CH*n*_INT_ENA Write 1 to enable the DMA2D_INFIFO_UDF_L1_CH*n*_INT. (R/W)

DMA2D_INFIFO_OVF_L2_CH*n*_INT_ENA Write 1 to enable the DMA2D_INFIFO_OVF_L2_CH*n*_INT. (R/W)

DMA2D_INFIFO_UDF_L2_CH*n*_INT_ENA Write 1 to enable the DMA2D_INFIFO_UDF_L2_CH*n*_INT. (R/W)

DMA2D_INFIFO_OVF_L3_CH*n*_INT_ENA Write 1 to enable the DMA2D_INFIFO_OVF_L3_CH*n*_INT. (R/W)

DMA2D_INFIFO_UDF_L3_CH*n*_INT_ENA Write 1 to enable the DMA2D_INFIFO_UDF_L3_CH*n*_INT. (R/W)

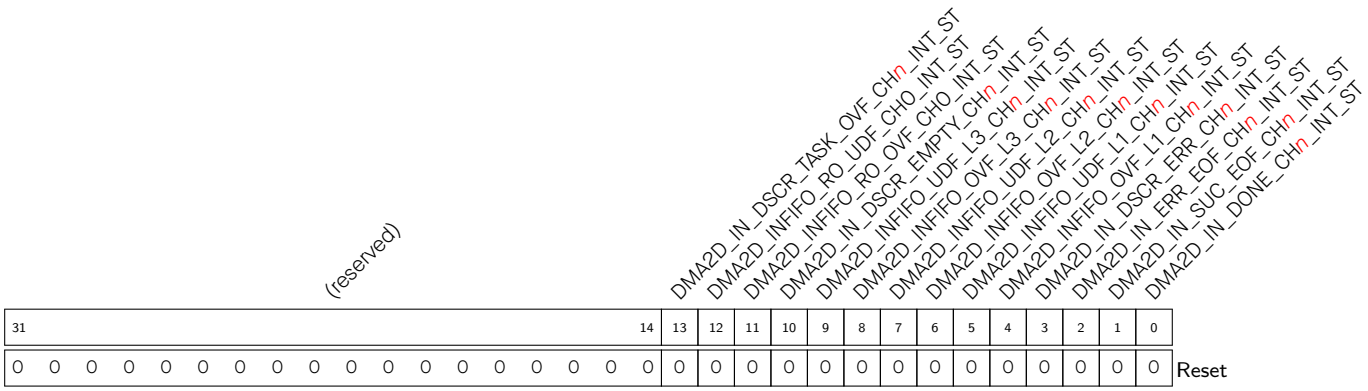
DMA2D_IN_DSCR_EMPTY_CH*n*_INT_ENA Write 1 to enable the DMA2D_IN_DSCR_EMPTY_CH*n*_INT. (R/W)

DMA2D_INFIFO_RO_OVF_CH0_INT_ENA Write 1 to enable the DMA2D_INFIFO_RO_OVF_CH0_INT. (R/W)

DMA2D_INFIFO_RO_UDF_CH0_INT_ENA Write 1 to enable the DMA2D_INFIFO_RO_UDF_CH0_INT. (R/W)

DMA2D_IN_DSCR_TASK_OVF_CH*n*_INT_ENA Write 1 to enable the DMA2D_IN_DSCR_TASK_OVF_CH*n*_INT. (R/W)

Register 8.45. DMA2D_IN_INT_ST_CH*n*_REG (*n*: 0-2) (0x050C+0x100*n)



DMA2D_IN_DONE_CH*n*_INT_ST The masked interrupt status of the DMA2D_IN_DONE_CH*n*_INT. (RO)

DMA2D_IN_SUC_EOF_CH*n*_INT_ST The masked interrupt status of the DMA2D_IN_SUC_EOF_CH*n*_INT. (RO)

DMA2D_IN_ERR_EOF_CH*n*_INT_ST The masked interrupt status of the DMA2D_IN_ERR_EOF_CH*n*_INT. (RO)

DMA2D_IN_DSCR_ERR_CH*n*_INT_ST The masked interrupt status of the DMA2D_IN_DSCR_ERR_CH*n*_INT. (RO)

DMA2D_INFIFO_OVF_L1_CH*n*_INT_ST The masked interrupt status of the DMA2D_INFIFO_OVF_L1_CH*n*_INT. (RO)

DMA2D_INFIFO_UDF_L1_CH*n*_INT_ST The masked interrupt status of the DMA2D_INFIFO_UDF_L1_CH*n*_INT. (RO)

DMA2D_INFIFO_OVF_L2_CH*n*_INT_ST The masked interrupt status of the DMA2D_INFIFO_OVF_L2_CH*n*_INT. (RO)

DMA2D_INFIFO_UDF_L2_CH*n*_INT_ST The masked interrupt status of the DMA2D_INFIFO_UDF_L2_CH*n*_INT. (RO)

DMA2D_INFIFO_OVF_L3_CH*n*_INT_ST The masked interrupt status of the DMA2D_INFIFO_OVF_L3_CH*n*_INT. (RO)

DMA2D_INFIFO_UDF_L3_CH*n*_INT_ST The masked interrupt status of the DMA2D_INFIFO_UDF_L3_CH*n*_INT. (RO)

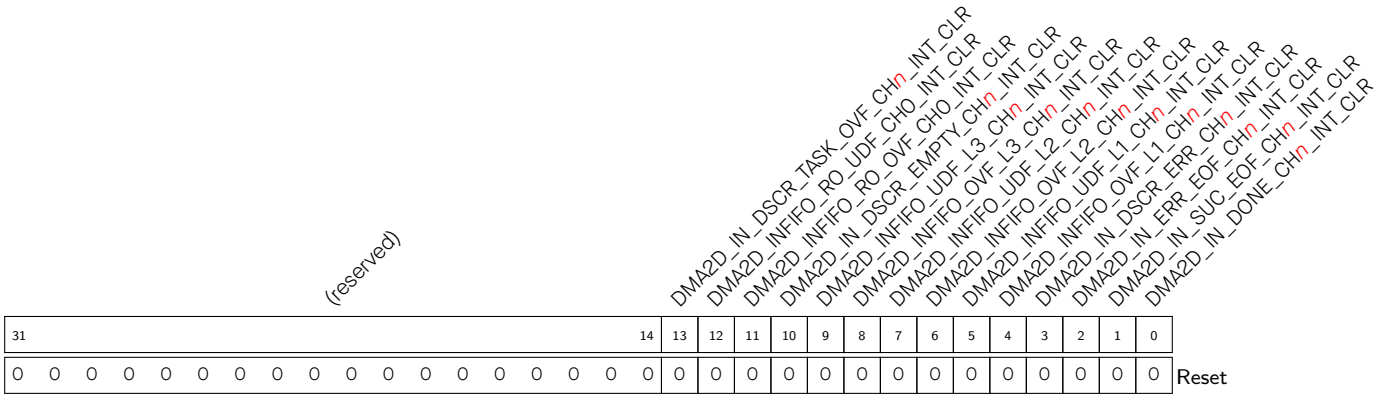
DMA2D_IN_DSCR_EMPTY_CH*n*_INT_ST The masked interrupt status of the DMA2D_IN_DSCR_EMPTY_CH*n*_INT. (RO)

DMA2D_INFIFO_RO_OVF_CHO_INT_ST The masked interrupt status of the DMA2D_INFIFO_RO_OVF_CHO_INT. (RO)

DMA2D_INFIFO_RO_UDF_CHO_INT_ST The masked interrupt status of the DMA2D_INFIFO_RO_UDF_CHO_INT. (RO)

DMA2D_IN_DSCR_TASK_OVF_CH*n*_INT_ST The masked interrupt status of the DMA2D_IN_DSCR_TASK_OVF_CH*n*_INT. (RO)

Register 8.46. DMA2D_IN_INT_CLR_CHn_REG (n: 0-2) (0x0510+0x100*n)



DMA2D_IN_DONE_CHn_INT_CLR Write 1 to clear the DMA2D_IN_DONE_CHn_INT. (WT)

DMA2D_IN_SUC_EOF_CHn_INT_CLR Write 1 to clear the DMA2D_IN_SUC_EOF_CHn_INT. (WT)

DMA2D_IN_ERR_EOF_CHn_INT_CLR Write 1 to clear the DMA2D_IN_ERR_EOF_CHn_INT. (WT)

DMA2D_IN_DSCR_ERR_CHn_INT_CLR Write 1 to clear the DMA2D_IN_DSCR_ERR_CHn_INT. (WT)

DMA2D_INFIFO_OVF_L1_CHn_INT_CLR Write 1 to clear the DMA2D_INFIFO_OVF_L1_CHn_INT. (WT)

DMA2D_INFIFO_UDF_L1_CHn_INT_CLR Write 1 to clear the DMA2D_INFIFO_UDF_L1_CHn_INT. (WT)

DMA2D_INFIFO_OVF_L2_CHn_INT_CLR Write 1 to clear the DMA2D_INFIFO_OVF_L2_CHn_INT. (WT)

DMA2D_INFIFO_UDF_L2_CHn_INT_CLR Write 1 to clear the DMA2D_INFIFO_UDF_L2_CHn_INT. (WT)

DMA2D_INFIFO_OVF_L3_CHn_INT_CLR Write 1 to clear the DMA2D_INFIFO_OVF_L3_CHn_INT. (WT)

DMA2D_INFIFO_UDF_L3_CHn_INT_CLR Write 1 to clear the DMA2D_INFIFO_UDF_L3_CHn_INT. (WT)

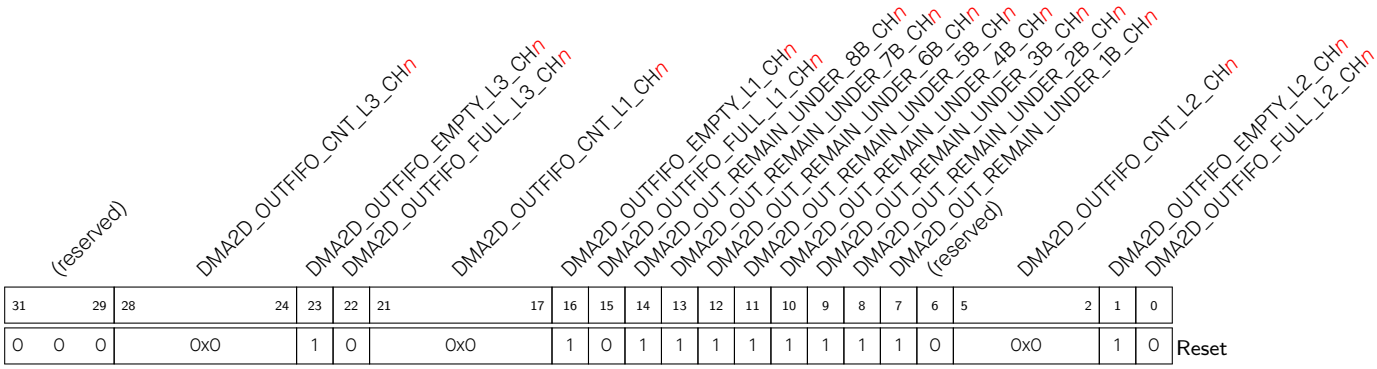
DMA2D_IN_DSCR_EMPTY_CHn_INT_CLR Write 1 to clear the DMA2D_IN_DSCR_EMPTY_CHn_INT. (WT)

DMA2D_INFIFO_RO_OVF_CHO_INT_CLR Write 1 to clear the DMA2D_INFIFO_RO_OVF_CHO_INT. (WT)

DMA2D_INFIFO_RO_UDF_CHO_INT_CLR Write 1 to clear the DMA2D_INFIFO_RO_UDF_CHO_INT. (WT)

DMA2D_IN_DSCR_TASK_OVF_CHn_INT_CLR Write 1 to clear the DMA2D_IN_DSCR_TASK_OVF_CHn_INT. (WT)

Register 8.47. DMA2D_OUTFIFO_STATUS_CH*n*_REG (*n*: 0-3) (0x0014+0x100n*)**



DMA2D_OUTFIFO_FULL_L2_CH*n* Represents whether the L2 TX FIFO for TX channel *n* is full.

0: Not full

1: Full

(RO)

DMA2D_OUTFIFO_EMPTY_L2_CH*n* Represents whether the L2 TX FIFO for TX channel *n* is empty.

0: Not empty

1: Empty

(RO)

DMA2D_OUTFIFO_CNT_L2_CH*n* Represents the number of data bytes in L2 TX FIFO for TX channel *n*. (RO)

DMA2D_OUT_REMAIN_UNDER_1B_CH*n* Reserved. (RO)

DMA2D_OUT_REMAIN_UNDER_2B_CH*n* Reserved. (RO)

DMA2D_OUT_REMAIN_UNDER_3B_CH*n* Reserved. (RO)

DMA2D_OUT_REMAIN_UNDER_4B_CH*n* Reserved. (RO)

DMA2D_OUT_REMAIN_UNDER_5B_CH*n* Reserved. (RO)

DMA2D_OUT_REMAIN_UNDER_6B_CH*n* Reserved. (RO)

DMA2D_OUT_REMAIN_UNDER_7B_CH*n* Reserved. (RO)

DMA2D_OUT_REMAIN_UNDER_8B_CH*n* Reserved. (RO)

DMA2D_OUTFIFO_FULL_L1_CH*n* Represents whether the L1 TX FIFO for TX channel *n* is full.

0: Not full

1: Full

(RO)

DMA2D_OUTFIFO_EMPTY_L1_CH*n* Represents whether the L1 TX FIFO for TX channel *n* is empty.

0: Not empty

1: Empty

(RO)

DMA2D_OUTFIFO_CNT_L1_CH*n* Represents the number of data bytes in L1 TX FIFO for TX channel *n*. (RO)

Continued on the next page...

Register 8.47. DMA2D_OUTFIFO_STATUS_CH*n*_REG (*n*: 0-3) (0x0014+0x100**n*)

Continued from the previous page...

DMA2D_OUTFIFO_FULL_L3_CH*n* Represents whether the L3 TX FIFO for TX channel *n* is full.

0: Not full

1: Full

(RO)

DMA2D_OUTFIFO_EMPTY_L3_CH*n* Represents whether the L3 TX FIFO for TX channel *n* is empty.

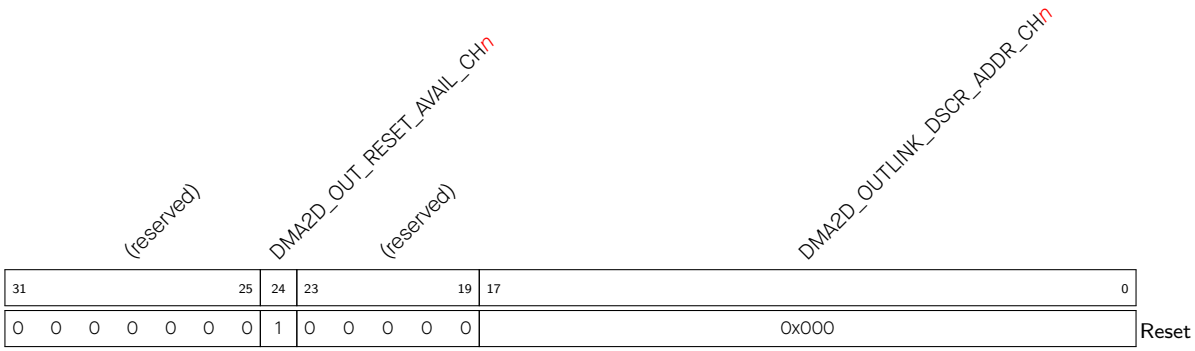
0: Not empty

1: Empty

(RO)

DMA2D_OUTFIFO_CNT_L3_CH*n* Represents the number of data bytes in L3 TX FIFO for TX channel *n*. (RO)

Register 8.48. DMA2D_OUT_STATE_CH*n*_REG (*n*: 0-3) (0x0024+0x100**n*)



DMA2D_OUTLINK_DSCR_ADDR_CH*n* Represents the lower 18 bits of the next transmit descriptor address that is pre-read (but not processed yet). If the current transmit descriptor is the last descriptor, then this field represents the address of the current transmit descriptor. (RO)

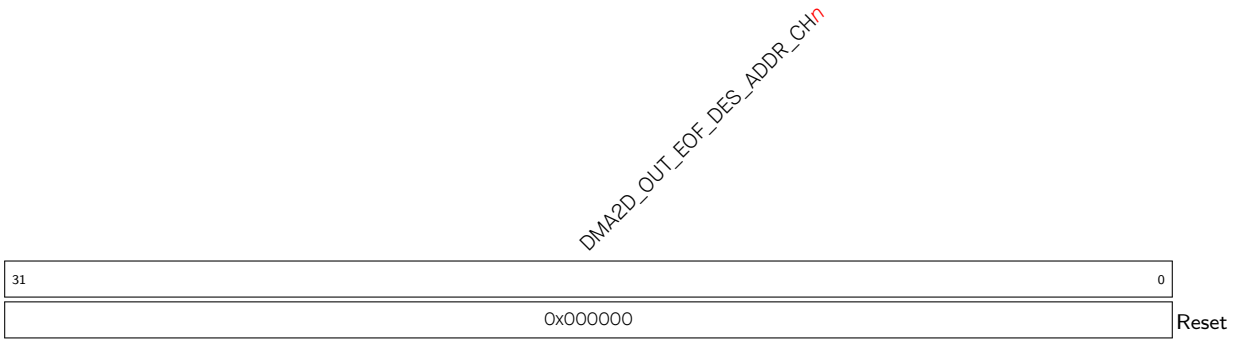
DMA2D_OUT_RESET_AVAIL_CH*n* Represents whether it is safe to reset TX channel *n*.

0: Not safe

1: Safe

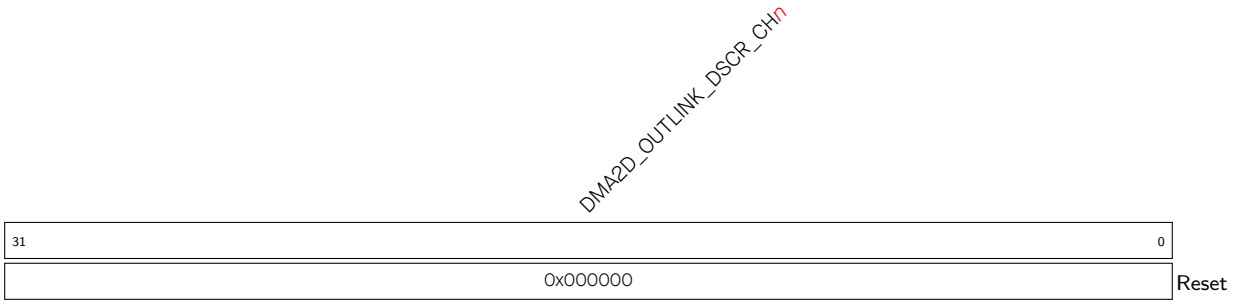
(RO)

Register 8.49. DMA2D_OUT_EOF_DES_ADDR_CH*n*_REG (*n*: 0-3) (0x0028+0x100**n*)



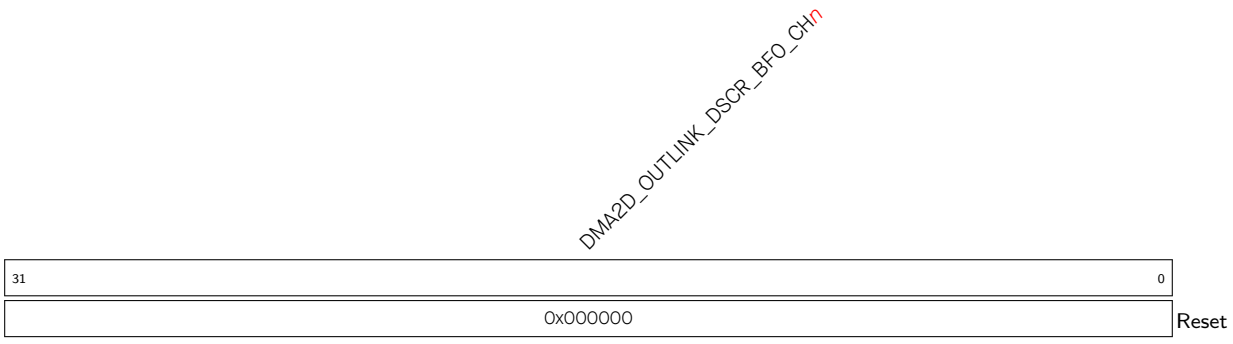
DMA2D_OUT_EOF_DES_ADDR_CH*n* Represents the address of the transmit descriptor when the eof bit in this descriptor is 1. (RO)

Register 8.50. DMA2D_OUT_DSCR_CH*n*_REG (*n*: 0-3) (0x002C+0x100**n*)



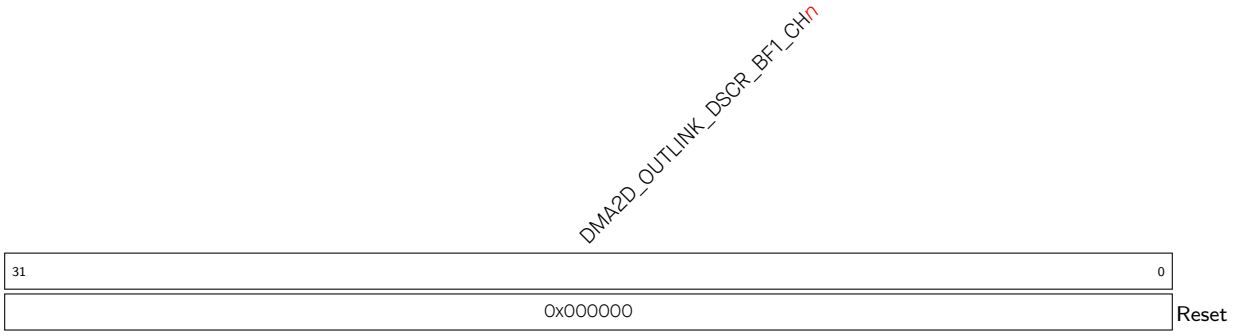
DMA2D_OUTLINK_DSCR_CH*n* Represents the address of the next transmit descriptor pointed by the current transmit descriptor that is pre-read. (RO)

Register 8.51. DMA2D_OUT_DSCR_BF0_CH*n*_REG (*n*: 0-3) (0x0030+0x100**n*)



DMA2D_OUTLINK_DSCR_BF0_CH*n* Represents the address of the current transmit descriptor that is pre-read. (RO)

Register 8.52. DMA2D_OUT_DSCR_BF1_CH*n*_REG (*n*: 0-3) (0x0034+0x100n*)**



DMA2D_OUTLINK_DSCR_BF1_CH*n* Represents the address of the previous transmit descriptor that is pre-read. (RO)

Register 8.53. DMA2D_INFIFO_STATUS_CH*n*_REG (*n*: 0-2) (0x0514+0x100**n*)



DMA2D_INFIFO_FULL_L2_CH*n* Represents whether the L2 RX FIFO for RX channel *n* is full.

0: Not full

1: Full

(RO)

DMA2D_INFIFO_EMPTY_L2_CH*n* Represents whether the L2 RX FIFO for RX channel *n* is empty.

0: Not empty

1: Empty

(RO)

DMA2D_INFIFO_CNT_L2_CH*n* Represents the number of data bytes in the L2 RX FIFO for RX channel *n*. (RO)

DMA2D_IN_REMAIN_UNDER_1B_CH*n* Reserved. (RO)

DMA2D_IN_REMAIN_UNDER_2B_CH*n* Reserved. (RO)

DMA2D_IN_REMAIN_UNDER_3B_CH*n* Reserved. (RO)

DMA2D_IN_REMAIN_UNDER_4B_CH*n* Reserved. (RO)

DMA2D_IN_REMAIN_UNDER_5B_CH*n* Reserved. (RO)

DMA2D_IN_REMAIN_UNDER_6B_CH*n* Reserved. (RO)

DMA2D_IN_REMAIN_UNDER_7B_CH*n* Reserved. (RO)

DMA2D_IN_REMAIN_UNDER_8B_CH*n* Reserved. (RO)

DMA2D_INFIFO_FULL_L1_CH*n* Represents whether the L1 RX FIFO for RX channel *n* is full.

0: Not full

1: Full

(RO)

DMA2D_INFIFO_EMPTY_L1_CH*n* Represents whether the L1 RX FIFO for RX channel *n* is empty.

0: Not empty

1: Empty

(RO)

DMA2D_INFIFO_CNT_L1_CH*n* Represents the number of data bytes in the L1 RX FIFO for RX channel *n*. (RO)

Continued on the next page...

Register 8.53. DMA2D_INFIFO_STATUS_CH*n*_REG (*n*: 0-2) (0x0514+0x100**n*)

Continued from the previous page...

DMA2D_INFIFO_FULL_L3_CH*n* Represents whether the L3 RX FIFO for RX channel *n* is full.

0: Not full

1: Full

(RO)

DMA2D_INFIFO_EMPTY_L3_CH*n* Represents whether the L3 RX FIFO for RX channel *n* is empty.

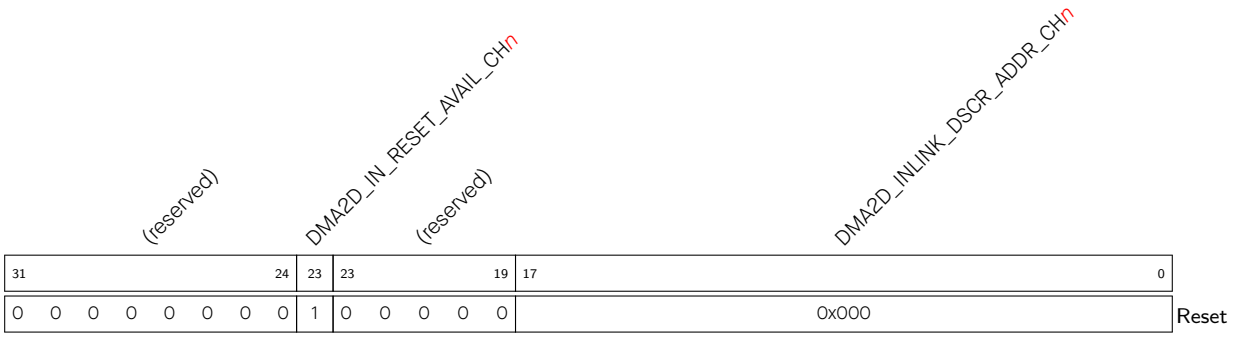
0: Not empty

1: Empty

(RO)

DMA2D_INFIFO_CNT_L3_CH*n* Represents the number of data bytes in the L3 RX FIFO for RX channel *n*. (RO)

Register 8.54. DMA2D_IN_STATE_CH*n*_REG (*n*: 0-2) (0x0524+0x100**n*)



DMA2D_INLINK_DSCR_ADDR_CH*n* Represents the lower 18 bits of the next receive descriptor address that is pre-read (but not processed yet). If the current receive descriptor is the last descriptor, then this field represents the address of the current receive descriptor. (RO)

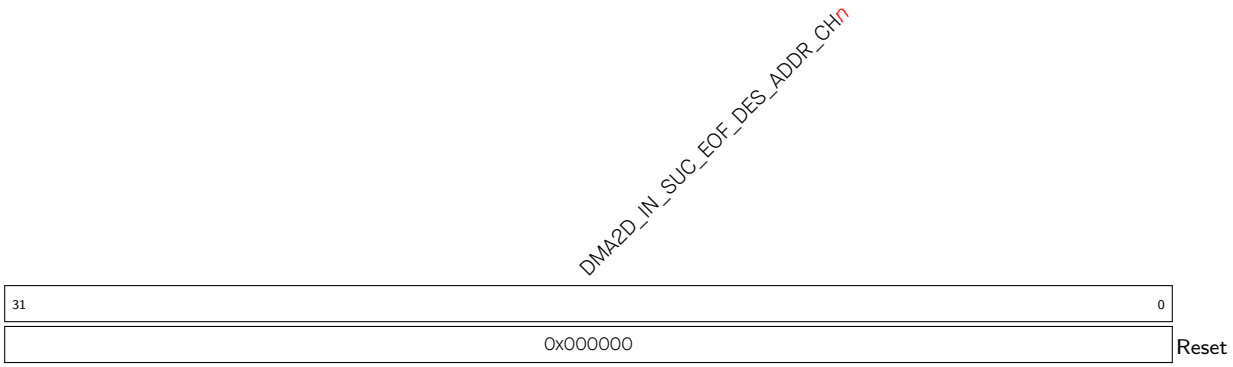
DMA2D_IN_RESET_AVAIL_CH*n* Represents whether it is safe to reset RX channel *n*.

0: Not safe

1: Safe

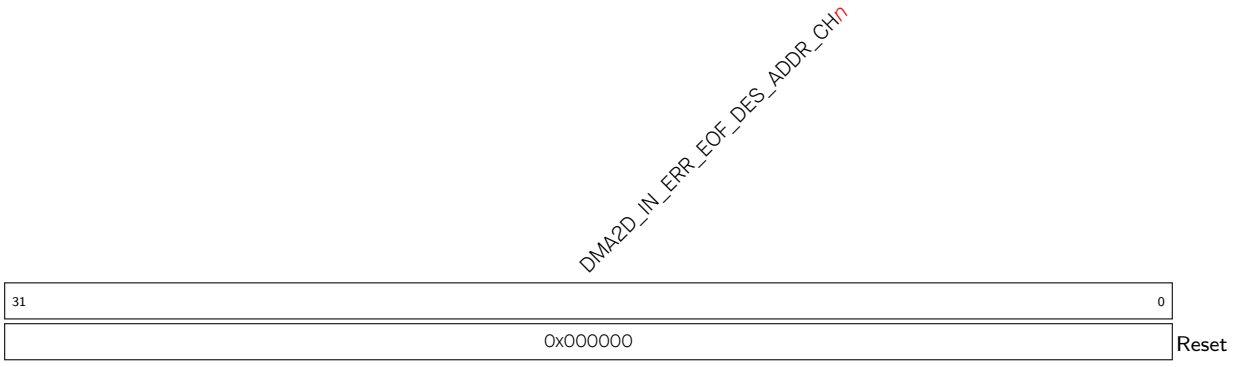
(RO)

Register 8.55. DMA2D_IN_SUC_EOF_DES_ADDR_CH*n*_REG (*n*: 0-2) (0x0528+0x100*n)



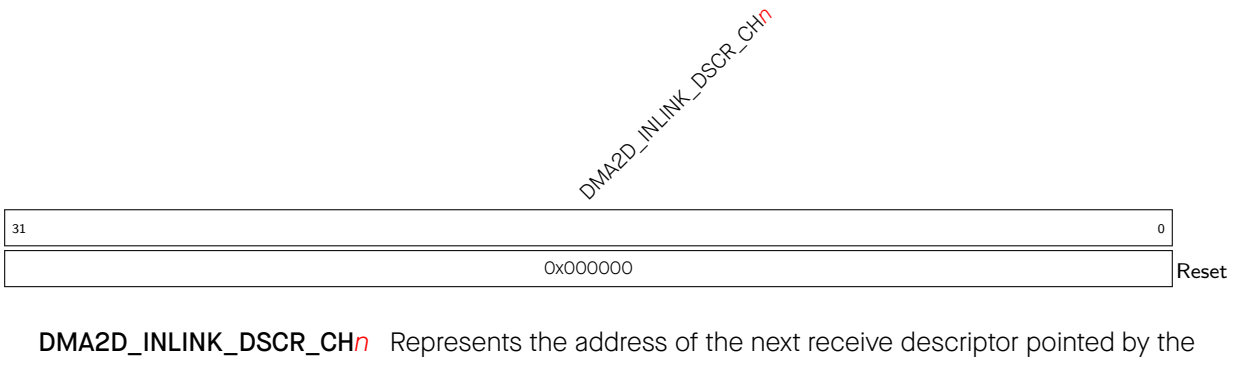
DMA2D_IN_SUC_EOF_DES_ADDR_CH*n* Represents the address of the receive descriptor when the eof bit in this descriptor is 1. (RO)

Register 8.56. DMA2D_IN_ERR_EOF_DES_ADDR_CH*n*_REG (*n*: 0-2) (0x052C+0x100*n)



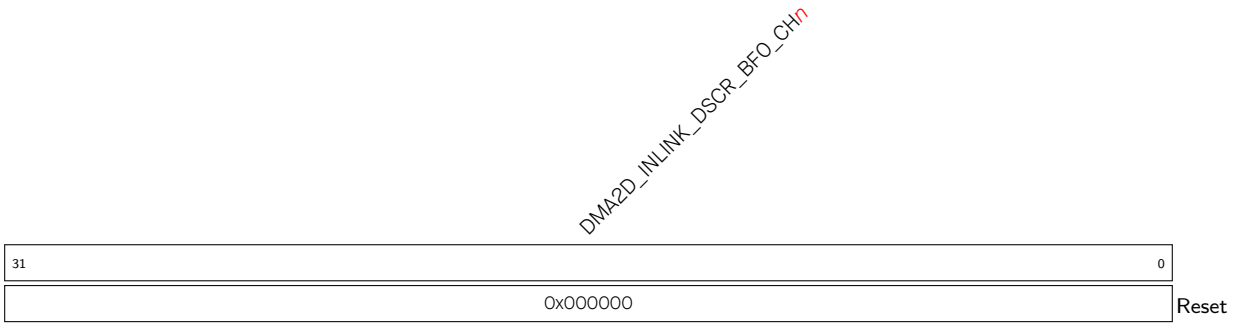
DMA2D_IN_ERR_EOF_DES_ADDR_CH*n* Represents the address of the receive descriptor when there are some errors in the currently received data. Valid only for JPEG. (RO)

Register 8.57. DMA2D_IN_DSCR_CH*n*_REG (*n*: 0-2) (0x0530+0x100*n)



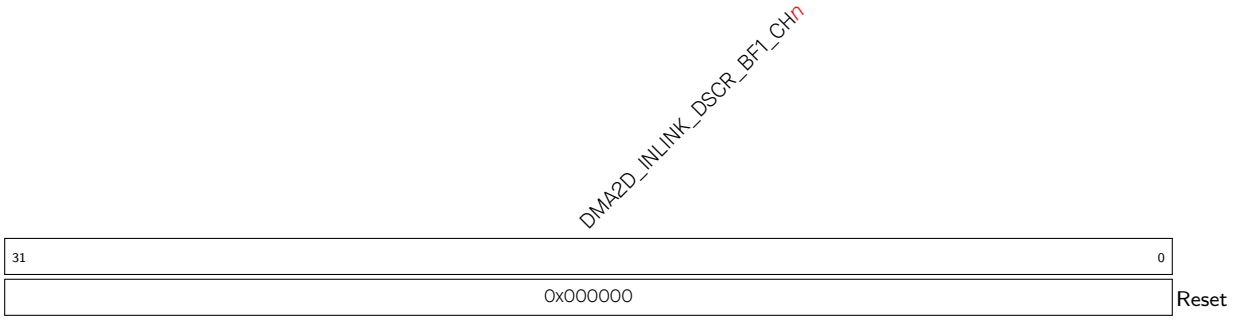
DMA2D_INLINK_DSCR_CH*n* Represents the address of the next receive descriptor pointed by the current receive descriptor that is pre-read. (RO)

Register 8.58. DMA2D_IN_DSCR_BF0_CH*n*_REG (*n*: 0-2) (0x0534+0x100**n*)



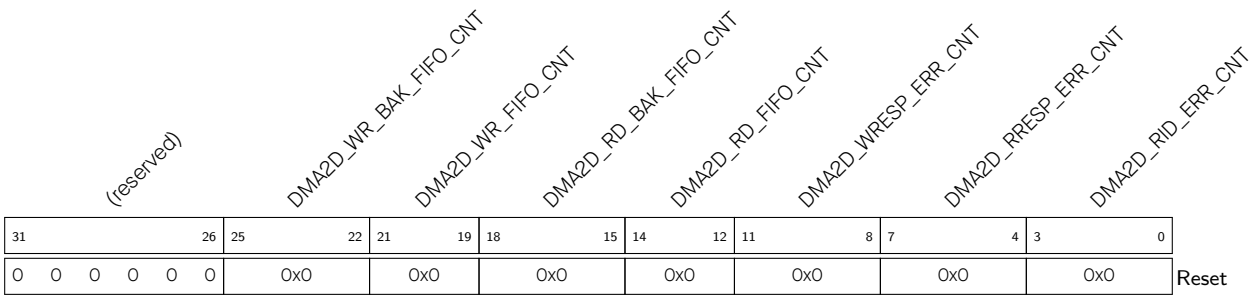
DMA2D_INLINK_DSCR_BF0_CH*n* Represents the address of the current receive descriptor that is pre-read. (RO)

Register 8.59. DMA2D_IN_DSCR_BF1_CH*n*_REG (*n*: 0-2) (0x0538+0x100**n*)



DMA2D_INLINK_DSCR_BF1_CH*n* Represents the address of the previous receive descriptor that is pre-read. (RO)

Register 8.60. DMA2D_AXI_ERR_REG (0x0A00)



DMA2D_RID_ERR_CNT Represents the number of AXI read ID errors. (RO)

DMA2D_RRESP_ERR_CNT Represents the number of AXI read response errors. (RO)

DMA2D_WRESP_ERR_CNT Represents the number of AXI write response errors. (RO)

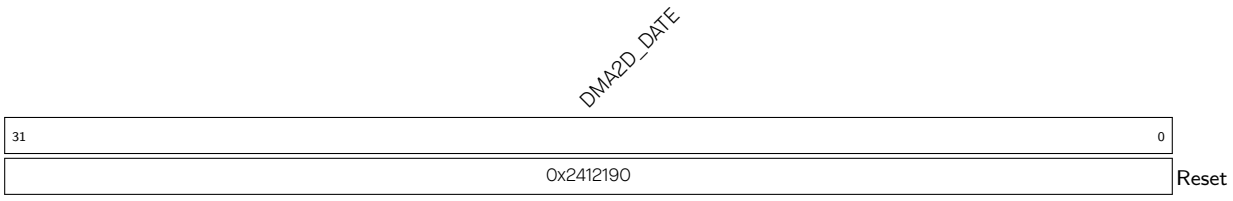
DMA2D_RD_FIFO_CNT Represents the number of remaining commands in the AXI read command FIFO. (RO)

DMA2D_RD_BAK_FIFO_CNT Represents the number of remaining commands in the AXI read backup command FIFO. (RO)

DMA2D_WR_FIFO_CNT Represents the number of remaining commands in the AXI write command FIFO. (RO)

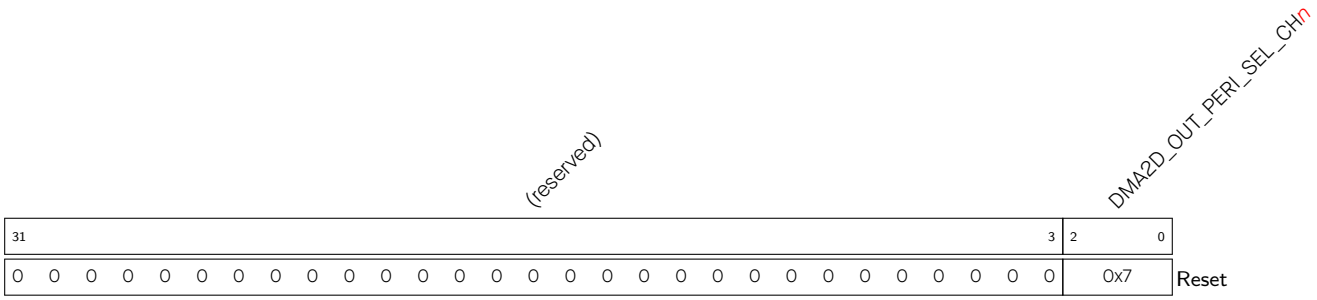
DMA2D_WR_BAK_FIFO_CNT Represents the number of remaining commands in the AXI write backup command FIFO. (RO)

Register 8.61. DMA2D_DATE_REG (0x0A2C)



DMA2D_DATE Version control register. (R/W)

Register 8.62. DMA2D_OUT_PERI_SEL_CH*n*_REG (*n*: 0-3) (0x0038+0x100n*)**



DMA2D_OUT_PERI_SEL_CH*n* Configures the peripheral connected to TX channel *n*.

0: JPEG

1: SRM module of PPA

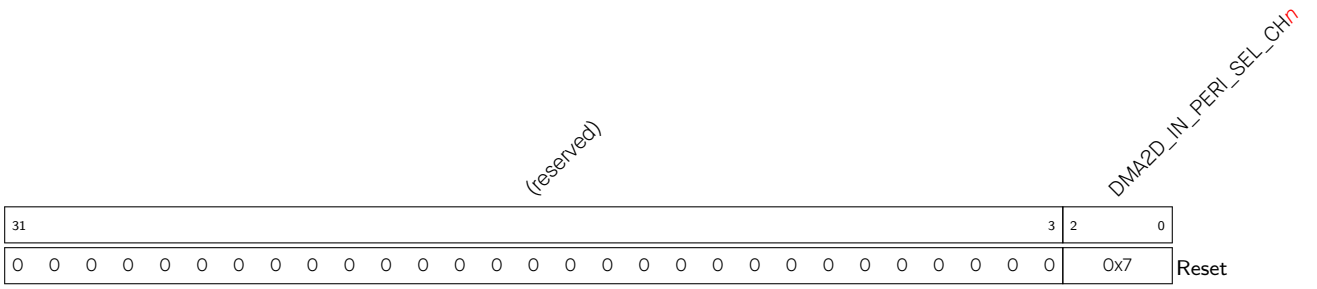
2: BLEND0 channel of PPA

3: BLEND1 channel of PPA

4 ~ 7: Dummy 4 ~ 7

(R/W)

Register 8.63. DMA2D_IN_PERI_SEL_CH*n*_REG (*n*: 0-2) (0x053C+0x100n*)**



DMA2D_IN_PERI_SEL_CH*n* Configures the peripheral connected to RX channel *n*.

0: JPEG

1: SRM module of PPA

2: BLEND module of PPA

3 ~ 7: Dummy 3 ~ 7

(R/W)

Part III

Memory Organization

This part provides insights into the system's memory structure, discussing the organization and mapping of RAM, ROM, eFuse, and external memories, offering a framework for understanding memory-related subsystems.

Chapter 9

System and Memory

9.1 Overview

ESP32-P4 integrates two processors:

- a high-performance 32-bit RISC-V dual-core processor (HP CPU), five-stage pipeline, clock frequency up to 400 MHz.
- a low-power 32-bit RISC-V single-core processor (LP CPU), two-stage pipeline, clock frequency up to 40 MHz.

All internal memory, external memory, and peripherals are located on the HP CPU and LP CPU buses.

9.2 Features

- **Address Space**
  - 896 KB of HP internal memory address space, accessible by the instruction bus or data bus of HP CPU via cache
  - 8 KB of HP internal memory address space, accessible by the instruction bus or data bus of HP CPU
  - 48 KB of LP internal memory address space, accessible by the instruction bus or data bus of LP CPU with zero latency
  - 1256 KB of peripheral address space
  - 64 MB of external flash virtual address space, accessible by the instruction bus or data bus
  - 64 MB of external RAM virtual address space, accessible by the instruction bus or data bus
  - 768 KB of internal DMA address space
  - 128 MB of external DMA address space
- **Internal Memory**
  - 128 KB of HP ROM
  - 768 KB of HP L2MEM
  - 8 KB of HP SPM (Scratchpad Memory)
  - 16 KB of LP ROM
  - 32 KB of LP SRAM
- **External Memory**

  - Supports up to 64 MB of external flash and RAM

- **Peripheral Space**
  - 96 modules/peripherals in total
- **GDMA**
  - 7 GDMA-AHB supported modules/peripherals
  - 6 GDMA-AXI supported modules/peripherals

9.3 Functional Description

9.3.1 Address Mapping

Figure 9.3-1 illustrates the system structure and address mapping. All the non-reserved addresses are accessible via both the instruction bus and the data bus, meaning that the instruction bus and the data bus share the same address space.

Both the data bus and instruction bus of the HP CPU and LP CPU are little-endian. However, the HP CPU's data bus (DBUS) has a 128-bit data width, while other buses maintain a 32-bit data width.

The HP CPU can access data via the data bus using single-byte, double-byte, 4-byte alignment, and, in case of AI Instruction, up to 16-byte alignment. The LP CPU can access data via the data bus using single-byte, double-byte, and 4-byte alignment.

The HP CPU has the following access capabilities:

- Direct access to HP SPM via both the data bus and instruction bus.
- Access to internal memory and external memory mapped into the address space via cache, including:
  - 128 KB HP ROM (0x4FC0_0000 ~ 0x4FC1_FFFF)
  - 768 KB HP L2MEM (0x4FF0_0000 ~ 0x4FFB_FFFF)
  - 64 MB external flash (0x4000_0000 ~ 0x43FF_FFFF)
  - 64 MB external RAM (0x4800_0000 ~ 0x4BFF_FFFF)

  Note:
  Software can use the addresses starting with "0x4" in two ways: cache-able access or noncache-able access, depending on the control of CPU Physical Memory Attribution (PMA). Cache-able access is done via cache. Noncache-able access is done without cache, which is similar to the direct access to the addresses starting with "0x8", however, such direct access to "0x8" addresses is usually for debugging.

- Direct access to internal memory and external memory mapped into the following address space without cache. This access is slower than that via cache.
  - 128 KB HP ROM (0x8FC0_0000 ~ 0x8FC1_FFFF)
  - 768 KB HP L2MEM (0x8FF0_0000 ~ 0x8FFB_FFFF)
  - 32 KB LP SRAM (0x5010_8000 ~ 0x5010_FFFF)

  - 64 MB external flash (0x8000_0000 ~ 0x83FF_FFFF)
  - 64 MB external RAM (0x8800_0000 ~ 0x8BFF_FFFF)
- Direct access to modules/peripherals via the data bus, including:
  - HP CPU Peripherals
  - HP Peripherals
  - LP Peripherals

The LP CPU's access capabilities include:

- Direct access to LP SRAM and LP ROM via both the data bus and instruction bus.
- Access to internal memory and external memory, including:
  - 128 KB HP ROM (0x4FC0_0000 ~ 0x4FC1_FFFF)
  - 768 KB HP L2MEM (0x4FF0_0000 ~ 0x4FFB_FFFF)
  - 64 MB external flash (0x4000_0000 ~ 0x43FF_FFFF)
  - 64 MB external RAM (0x4800_0000 ~ 0x4BFF_FFFF)
- Direct access to modules/peripherals via the data bus, including:
  - HP CPU Peripherals
  - HP Peripherals
  - LP Peripherals

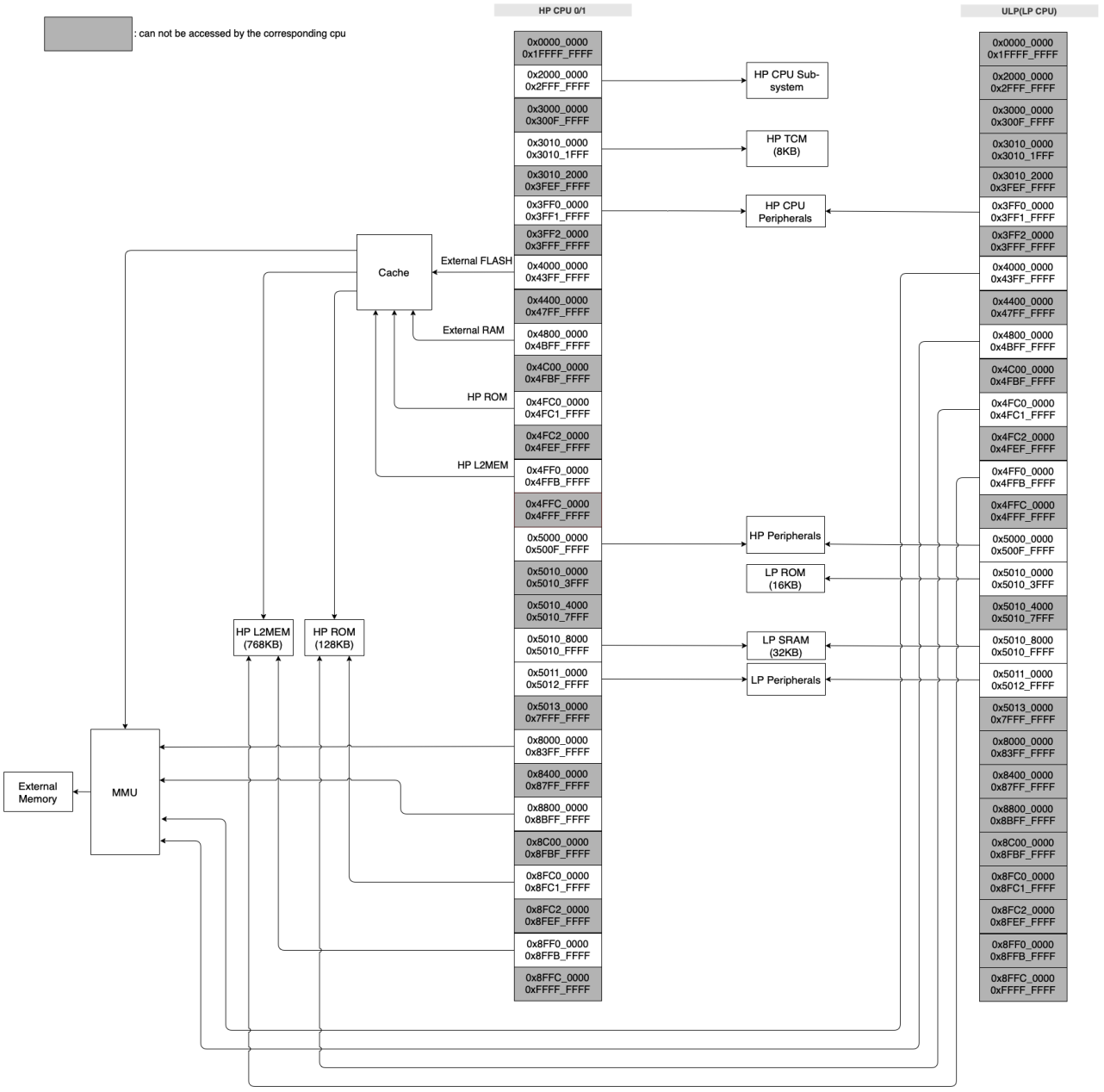


Figure 9.3-1. System Structure and Address Mapping

- The range of addresses available in the address space may be larger than the actual available memory of a particular type.
- For HP CPU sub-system, please refer to Chapter *2 High-Performance CPU*.

Table 9.3-1 lists the address ranges on the data bus and instruction bus and their corresponding target memories.

Table 9.3-1. Memory Address Mapping

Bus Type	Boundary Address		Size	Target*
	Low Address	High Address		
	0x0000_0000	0x300F_FFFF		Reserved

Cont'd on next page

Table 9.3-1 – cont'd from previous page

Bus Type	Boundary Address		Size	Target*
	Low Address	High Address		
Data/Instruction bus	0x3010_0000	0x3010_1FFF	8 KB	HP SPM
	0x3010_2000	0x3FEF_FFFF		Reserved
Data/Instruction bus	0x3FF0_0000	0x3FF1_FFFF	128 KB	HP CPU peripherals
	0x3FF2_0000	0x3FFF_FFFF		Reserved
Data/Instruction bus	0x4000_0000	0x43FF_FFFF	64 MB	External flash
	0x4400_0000	0x47FF_FFFF		Reserved
Data/Instruction bus	0x4800_0000	0x4BFF_FFFF	64 MB	External RAM
	0x4C00_000	0x4FBF_FFFF		Reserved
Data/Instruction bus	0x4FC0_0000	0x4FC1_FFFF	128 KB	HP ROM
	0x4FC2_0000	0x4FEF_FFFF		Reserved
Data/Instruction bus	0x4FF0_0000	0x4FFB_FFFF	768 KB	HP L2MEM
	0x4FFC_0000	0x4FFF_FFFF		Reserved
Data/Instruction bus	0x5000_0000	0x500F_FFFF	1 MB	HP peripherals
Data/Instruction bus	0x5010_0000	0x5010_3FFF	16 KB	LP ROM
	0x5010_4000	0x5010_7FFF		Reserved
Data/Instruction bus	0x5010_8000	0x5010_FFFF	32 KB	LP SRAM
Data/Instruction bus	0x5011_0000	0x5012_FFFF	128 KB	LP peripherals
	0x5013_0000	0x7FFF_FFFF		Reserved
Data/Instruction bus	0x8000_0000	0x83FF_FFFF	64 MB	External flash (direct access by CPU)
	0x8400_0000	0x87FF_FFFF		Reserved
Data/Instruction bus	0x8800_0000	0x8BFF_FFFF	64 MB	External RAM (direct access by CPU)
	0x8C00_000	0x8FBF_FFFF		Reserved
Data/Instruction bus	0x8FC0_0000	0x8FC1_FFFF	128 KB	HP ROM (direct access by CPU)
	0x8FC2_0000	0x8FEF_FFFF		Reserved
Data/Instruction bus	0x8FF0_0000	0x8FFB_FFFF	768 KB	HP L2MEM (direct access by CPU)
	0x8FFC_0000	0xFFFF_FFFF		Reserved

* Address starting with 0x4xxx_xxxx can be configured as noncache-able or cache-able, depending on CPU PMA, whereas address starting with 0x8xxx_xxxx is accessed directly by CPU.

* For the targets accessible by HP CPU or LP CPU, see Figure 9.3-1.

9.3.2 Internal Memory

ESP32-P4 has various types of internal memory:

- **HP ROM (128 KB):** This read-only memory is dedicated to the HP system and is not programmable. It contains the ROM code and read-only data of some low-level system software. The HP ROM is accessed by the HP CPU through the l2 cache at half the frequency of HP CPU.
- **HP L2MEM (768 KB):** A volatile memory accessible by the HP CPU, LP CPU, or DMA peripherals at half the frequency of the HP CPU. HP L2MEM can be configured to retain power during Light-sleep mode, making it suitable for data retention or register backup.
- **LP ROM (16 KB):** This read-only memory serves the LP system, containing the code for booting the LP CPU and some basic system functions. The LP ROM is accessed by the LP CPU with zero latency.

- **LP SRAM (32 KB):** A volatile memory accessible by both the HP CPU and LP CPU, operating at the same frequency as the LP CPU.
- **HP SPM (8 KB):** A volatile memory accessed by the HP CPU, finishing one access in two cycles.

1. Details for HP ROM

The 128 KB HP ROM is a read-only memory accessed by the HP CPU through the instruction bus or data bus via the following addresses as shown in Table 9.3-1:

- 0x4FC0_0000 (cache-able or noncache-able, configuration-dependent) ~ 0x4FC1_FFFF (cache-able or noncache-able, configuration-dependent).
- 0x8FC0_0000 (direct access by CPU) ~ 0x8FC1_FFFF (direct access by CPU).

2. Details for HP L2MEM

This 768 KB HP L2MEM is a read-and-write memory accessed by the HP CPU or LP CPU through the instruction bus or data bus, or by DMA master through AHB matrix or AXI matrix via the following addresses as shown in Table 9.3-1:

- 0x4FF0_0000 (cache-able or noncache-able, configuration-dependent) ~ 0x4FFB_FFFF (cache-able or noncache-able, configuration-dependent).
- 0x8FF0_0000 (direct access by CPU) ~ 0x8FFB_FFFF (direct access by CPU).

L2MEM supports multiple capacity split configurations: L2 RAM and L2 cache share a total capacity of 768 KB, and one of the following four configurations can be selected:

- Config 0: L2 RAM 256 KB, L2 cache 512 KB
- Config 1: L2 RAM 512 KB, L2 cache 256 KB
- Config 2: L2 RAM 640 KB, L2 cache 128 KB
- Config 3: L2 RAM 768 KB, L2 cache 0 KB

The address layout of HP L2MEM for the above four configurations is shown in Figure 9.3-2.

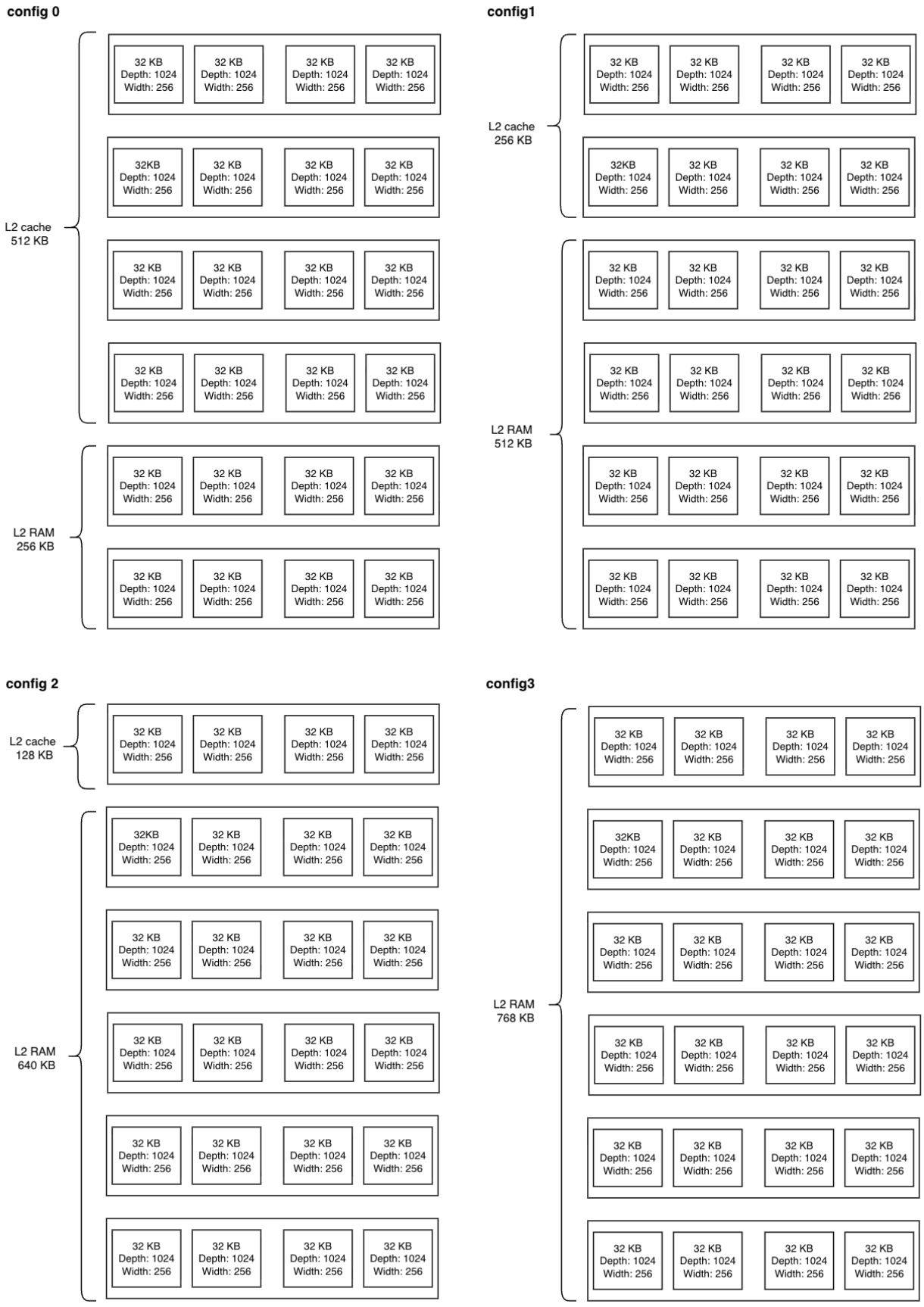


Figure 9.3-2. HP L2MEM Address Layout

HP L2MEM can be accessed by various peripherals:

- by USB 2.0 High-Speed OTG, USB 2.0 Full-Speed OTG, GMAC, SDMMC, TRACE, GDMA-AHB, and SYSMON via AHB matrix.
- by GDMA, GDMA-AXI, 2D-DMA, and H264 Encoder via AXI matrix.

The HP L2MEM supports Error Correction Code (ECC) check to correct one-bit memory flip error. In the event of a one-bit flip error, the data in HP L2MEM still remains valid. Before ECC check, HP L2MEM itself must be initialized first.

HP L2MEM consists of six memory units, each occupying 128 KB. The initialization process allows individual configuration of each unit for power considerations. It is recommended to initialize memory units sequentially rather than simultaneously. This sequential approach helps prevent excessive power consumption and avoids triggering a brownout condition when initializing the entire 768 KB HP L2MEM.

Detailed Configuration Steps:
Take unit0 as an example, other units follow the same configuration steps.

- Set HP_SYSTEM_L2_MEM_UNIT0_REFRESH_EN and clear HP_SYSTEM_L2_MEM_REFRESH_CNT_RESET to start initializing HP L2MEM.
- Read the value of HP_SYSTEM_L2_MEM_UNIT0_REFRESH_DONE.
- Once HP_SYSTEM_L2_MEM_UNIT0_REFRESH_DONE reaches a value of 1, set HP_SYSTEM_L2_RAM_UNIT0_ECC_EN to start ECC check.
- (Recommended) clear HP_SYSTEM_L2_MEM_UNIT0_REFRESH_EN and set HP_SYSTEM_L2_MEM_REFRESH_CNT_RESET after HP_SYSTEM_L2_MEM_UNIT0_REFRESH_DONE reaches 1 to complete the initialization process.

During the initialization process, avoid any access to HP L2MEM unit being initialized.

3. Details for LP ROM

This 16 KB LP ROM is a read-only memory accessed by LP CPU through the instruction bus or data bus via their shared address 0x5010_0000 ~ 0x5010_3FFF as shown in Table 9.3-1.

4. Details for LP SRAM

This 32 KB LP SRAM is a read-and-write memory accessed by the HP CPU via AHB matrix and by the LP CPU through the instruction bus or data bus via their shared address 0x5010_8000 ~ 0x5010_FFFF as shown in Table 9.3-1. LP SRAM supports atomic operation.

5. Details for HP SPM

This 8 KB HP SPM is a read-and-write memory accessed by the HP CPU through the instruction bus or data bus via their shared address 0x3010_0000 ~ 0x3010_1FFF as shown in Table 9.3-1. Additionally, HP SPM supports atomic operation of the HP CPU.

HP SPM supports parity check to detect one-bit memory flip error. Before conducting the parity check, HP SPM itself must be initialized first.

Detailed Configuration Steps:

- Set HP_SYSTEM_SPM_INIT_EN and clear HP_SYSTEM_SPM_INIT_CNT_RESET to start initializing HP SPM.

- Read the value of HP_SYSTEM_SPM_INIT_DONE.
- Once HP_SYSTEM_SPM_INIT_DONE reaches a value of 1, set HP_SYSTEM_HP_SPM_PARITY_CHECK_EN to start parity check.
- (Recommended) clear HP_SYSTEM_SPM_INIT_EN and set HP_SYSTEM_SPM_INIT_CNT_RESET after HP_SYSTEM_SPM_INIT_CNT_RESET reaches 1 to complete the initialization process.

During the above process, avoid any access to HP SPM.

9.3.3 External Memory

ESP32-P4 supports SPI, dual SPI, quad SPI and QPI interfaces for connecting to external flash, as well as OPI and HPI interfaces for connecting to external RAM.

ESP32-P4 provides the following security features to protect users' data in external flash and RAM:

- hardware manual encryption and automatic decryption based on XTS-AES algorithm to protect users' programs and data stored in the external flash.
- automatic encryption and decryption based on XTS-AES algorithm to protect users' data in the external RAM.

9.3.3.1 External Memory Address Mapping

HP CPU can access external flash or RAM via 0x4000_0000 ~ 0x43FF_FFFF or via 0x4800_0000 ~ 0x4BFF_FFFF using cache-able or noncache-able methods. HP CPU can also access external flash or RAM directly via 0x8000_0000 ~ 0x83FF_FFFF or via 0x8800_0000 ~ 0x8BFF_FFFF, which is slower than the access via cache and is therefore commonly used for debugging.
When the HP CPU accesses external memory, it maps the addresses to physical addresses based on the information in the Memory Management Unit (MMU).

- Range (0x4000_0000 ~ 0x43FF_FFFF) is mapped to the physical address of the external flash accessed by HP CPU via cache.
- Range (0x4800_0000 ~ 0x4BFF_FFFF) is mapped to the physical address of the external RAM accessed by HP CPU via cache.
- Range (0x8000_0000 ~ 0x83FF_FFFF) is mapped to the physical address of the external flash accessed by HP CPU directly.
- Range (0x8800_0000 ~ 0x8BFF_FFFF) is mapped to the physical address of the external RAM accessed by HP CPU directly.

This mapping allows ESP32-P4 to address up to 64 MB of external flash and 64 MB of external RAM. **Note that** the instruction bus shares the same address space (64 MB) with the data bus for accessing external memory.

9.3.3.2 Cache

As shown in Figure 9.3-3, ESP32-P4 has a two-level cache system, with the following features:

- L1 cache, including:
  - 16 KB of instruction cache (icache) with a 64 B block size, four-way set associative

  - 64 KB of data cache (dcache) with a 64 B block size, two-way set associative, supporting two writing strategies: write-through and write-back
- L2 cache: 128 KB/256 KB/512 KB with 64 B/128 B block sizes, eight-way set associative
- Cache-able and noncache-able access, depending on CPU PMA
- Preload operation
- Lock operation
- Critical word first and early restart

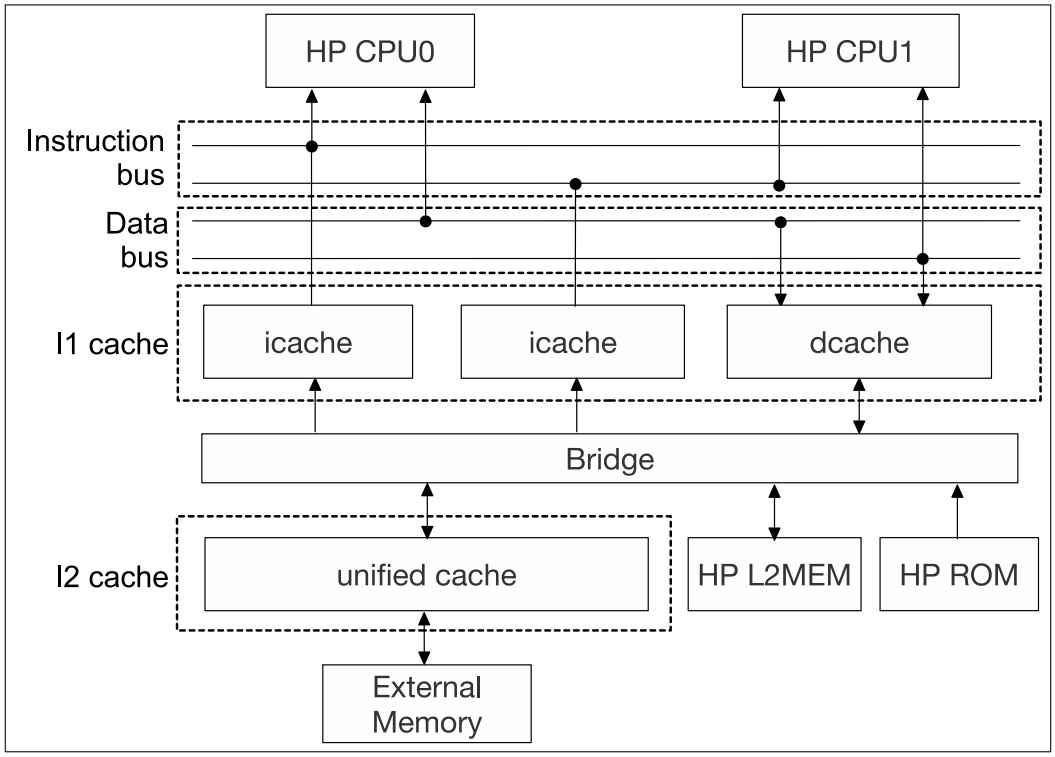


Figure 9.3-3. Cache Structure

9.3.3.3 Cache Operations

ESP32-P4 I1 cache and I2 cache supports the following operations:

1. **Write-back** (for dcache and I2 cache only):
   - Clears dirty bits in the tag memory and updates new data to external memory.
   - After the write-back operation, the new data is updated to external memory.
   - Users can choose whether to invalidate the data in the cache.
   - If the data is not invalidated, CPU will read/write data directly from/to the cache where data missing will not occur.
2. **Clean** (for dcache and I2 cache only):
   - Clears dirty bits in the tag memory without updating data to external memory.
   - After the clean operation, old data remains in external memory, while the cache holds the new one (unaware of it).

- CPU can then read/write the data directly from/to the cache, where data missing will not occur.

3. **Invalidate**:
   - Removes valid data from the cache.
   - Once this operation is done, the deleted data is stored only in external memory.
   - HP CPU needs to access external memory if it wants to access the data again.
   - Two types of invalidate operation:
     - Manual-Invalidate: is performed only on data in the specified cache area.
     - Invalidate-All: is performed on all data in the cache.
4. **Preload**:
   - Loads instructions and data into the cache in advance.
   - Minimum unit of preload-operation is one block.
   - Two types of preload operation:
     - Manual-Preload: involves the hardware prefetching continuous data based on the virtual address specified by the software.
     - Auto-Preload: involves the hardware prefetching continuous data based on the current address where the cache hits or misses (configuration-dependent).
5. **Lock/Unlock**:
   - Lock operation prevents easily replacing data in the cache.
   - Two types of lock:
     - Prelock: locks data in the specified area when filling missing data to cache memory, leaving the data outside the specified area unlocked.
     - Manual Lock: checks data in the cache memory and locks data only if it falls in the specified area, leaving data outside the specified area unlocked.
   - When there is missing data, the cache replaces the data in the unlocked way first, ensuring that the data in the locked way remains in the cache and will not be replaced.
   - When all ways within the cache are locked, the cache will replace data, as if it was not locked.
   - Unlocking is the reverse of locking and can only be done manually.
   - Note: Manual-Invalidate operation only works on the unlocked data. If you plan to perform this operation on the locked data, please unlock them first.

9.3.4 DMA Address Space

ESP32-P4 includes AHB General Direct Memory Access (GDMA-AHB), AXI General Direct Memory Access (GDMA-AXI), and VDMA, all providing general direct memory access services.

In ESP32-P4, masters like USB 2.0 High-Speed OTG, USB 2.0 Full-Speed OTG, GDMAC, Trace0/1, VDMA, 2D-DMA, and H264 Encoder have built-in DMA function modules, providing direct memory access services.

All DMA-enabled master devices mentioned above can access:

- HP L2MEM (0x4FF0_0000~0x4FFB_FFFF).
- external flash (0x4000_0000 ~ 0x43FF_FFFF).
- external RAM (0x4800_0000 ~ 0x4BFF_FFFF).

Additionally, VDMA Master 1 can also access:

- MIPI CSI internal memory (0x5010_4000 ~ 0x5010_4FFF).
- MIPI DSI internal memory (0x5010_5000 ~ 0x5010_5FFF).

Note:
When accessing a memory via DMA, a corresponding access permission is needed, otherwise this access may fail. For more information about permission control, please refer to Chapter 20 *Permission Control (PMS)*.

9.3.5 Modules/Peripherals Address Mapping

Table 9.3-2 lists all the modules/peripherals and their respective address ranges. Note that the address space of specific modules/peripherals is defined by "Boundary Address" (including both Low Address and High Address).

Table 9.3-2. Module/Peripheral Address Mapping

Target	Boundary Address		Size (KB)
	Low Address	High Address	
HP CPU Peripherals (HP CPU PERI)			
Reserved	0x3FF0_0000	0x3FF0_3FFF	
RISC-V Trace Encoder 0	0x3FF0_4000	0x3FF0_4FFF	4
RISC-V Trace Encoder 1	0x3FF0_5000	0x3FF0_5FFF	4
Bus Monitor	0x3FF0_6000	0x3FF0_6FFF	4
Reserved	0x3FF0_7000	0x3FF0_DFFF	
L2MEM Monitor	0x3FF0_E000	0x3FF0_EFFF	4
SPM Monitor	0x3FF0_F000	0x3FF0_FFFF	4
HP Peripherals 0 (HP PERI0)			
USB 2.0 High-Speed OTG	0x5000_0000	0x5003_FFFF	256
USB 2.0 Full-Speed OTG	0x5004_0000	0x5007_FFFF	256
USB 2.0 Full-Speed OTG PHY	0x5008_0000	0x5008_0FFF	4
VDMA Controller	0x5008_1000	0x5008_1FFF	4
Reserved	0x5008_2000	0x5008_2FFF	
SD/MMC Host Controller	0x5008_3000	0x5008_3FFF	4
H264 Encoder	0x5008_4000	0x5008_4FFF	4
GDMA-AHB	0x5008_5000	0x5008_5FFF	4
JPEG Codec	0x5008_6000	0x5008_6FFF	4
Pixel-Processing Accelerator (PPA)	0x5008_7000	0x5008_7FFF	4
2D-DMA Controller	0x5008_8000	0x5008_8FFF	4

Cont'd on next page

Table 9.3-2 – cont'd from previous page

Target	Boundary Address		Size (KB)
	Low Address	High Address	
Key Manager	0x5008_9000	0x5008_9FFF	4
GDMA-AXI	0x5008_A000	0x5008_BFFF	8
AES Accelerator (AES)	0x5009_0000	0x5009_0FFF	4
SHA Accelerator (SHA)	0x5009_1000	0x5009_1FFF	4
RSA Accelerator (RSA)	0x5009_2000	0x5009_2FFF	4
RSA Digital Signature Peripheral (RSA_DS)	0x5009_3000	0x5009_3FFF	4
HMAC Accelerator (HMAC)	0x5009_4000	0x5009_4FFF	4
ECC Accelerator (ECC)	0x5009_5000	0x5009_5FFF	4
ECDSA_DS	0x5009_6000	0x5009_6FFF	4
Reserved	0x5009_7000	0x5009_7FFF	
GMAC	0x5009_8000	0x5009_BFFF	16
USB 2.0 High-Speed OTG PHY	0x5009_C000	0x5009_CFFF	4
Reserved	0x5009_D000	0x5009_DFFF	
CSI Host	0x5009_F000	0x500F_FFFF	4
DSI Host	0x500A_0000	0x500A_0FFF	4
Image Signal Processor (ISP)	0x500A_1000	0x500A_1FFF	4
Remote Control Peripheral (RMT)	0x500A_2000	0x500A_2FFF	4
Bit-scrambler	0x500A_3000	0x500A_3FFF	4
AXI ICM	0x500A_4000	0x500A_4FFF	4
HP Peripheral Permission (HP_PERI_PMS)	0x500A_5000	0x5009_57FF	2
LP2HP Peripheral Permission (LP2HP_PERI_PMS)	0x500A_5800	0x500A_5FFF	2
HP DMA Permission (HP_DMA_PMS)	0x500A_6000	0x500A_6FFF	4
H264 DMA	0x500A_7000	0x500A_7FFF	4
Reserved	0x500A_8000	0x500B_FFFF	
HP Peripherals 1 (HP PERI1)			
Motor Control PWM0 (MCPWM0)	0x500C_0000	0x500C_0FFF	4
Motor Control PWM1 (MCPWM1)	0x500C_1000	0x500C_1FFF	4
Timer Group 0 (TIMG0)	0x500C_2000	0x500C_2FFF	4
Timer Group 1 (TIMG1)	0x500C_3000	0x500C_3FFF	4
I2C Controller0 (I2C0)	0x500C_4000	0x500C_4FFF	4
I2C Controller1 (I2C1)	0x500C_5000	0x500C_5FFF	4
I2S Controller0 (I2S0)	0x500C_6000	0x500C_6FFF	4
I2S Controller1 (I2S1)	0x500C_7000	0x500C_7FFF	4
I2S Controller0 (I2S2)	0x500C_8000	0x500C_8FFF	4
Pulse Count Controller (PCNT)	0x500C_9000	0x500C_9FFF	4
UART Controller 0 (UART0)	0x500C_A000	0x500C_AFFF	4
UART Controller 1 (UART1)	0x500C_B000	0x500C_BFFF	4
UART Controller 2 (UART2)	0x500C_C000	0x500C_CFFF	4
UART Controller 3 (UART3)	0x500C_D000	0x500C_DFFF	4
UART Controller 4 (UART4)	0x500C_E000	0x500C_EFFF	4

Cont'd on next page

Table 9.3-2 – cont'd from previous page

Target	Boundary Address		Size (KB)
	Low Address	High Address	
Parallel IO Controller (PARL_IO)	0x500C_F000	0x500C_FFFF	4
General Purpose SPI2 (GP-SPI2)	0x500D_0000	0x500D_0FFF	4
General Purpose SPI3 (GP-SPI3)	0x500D_1000	0x500D_1FFF	4
USB Serial/JTAG Controller	0x500D_2000	0x500D_2FFF	4
LED PWM Controller (LEDC)	0x500D_3000	0x500D_3FFF	4
Reserved	0x500D_4000	0x500D_4FFF	
Event Task Matrix (SOC_ETM)	0x500D_5000	0x500D_5FFF	4
Interrupt Matrix (INTMTX)	0x500D_6000	0x500D_6FFF	4
Two-wire Automotive Interface 0 (TWAI0)	0x500D_7000	0x500D_7FFF	4
Two-wire Automotive Interface 1 (TWAI1)	0x500D_8000	0x500D_8FFF	4
Two-wire Automotive Interface 2 (TWAI2)	0x500D_9000	0x500D_9FFF	4
I3C Master Controller	0x500D_A000	0x500D_AFFF	4
I3C Slave Controller	0x500D_B000	0x500D_BFFF	4
LCD and Camera Controller (LCD_CAM)	0x500D_C000	0x500D_CFFF	4
Reserved	0x500D_D000	0x500D_DFFF	
ADC Controller	0x500D_E000	0x500D_EFFF	4
UHCI Controller (UHCI)	0x500D_F000	0x500D_FFFF	4
GPIO Matrix	0x500E_0000	0x500E_0FFF	4
IO MUX	0x500E_1000	0x500E_1FFF	4
System Timer (SYSTIMER)	0x500E_2000	0x500E_2FFF	4
Reserved	0x500E_3000	0x500E_4FFF	
HP System Register (SYSREG)	0x500E_5000	0x500D_5FFF	4
Reset and Clock	0x500E_6000	0x500E_6FFF	4
Reserved	0x500E_7000	0x500F_FFFF	
MIPI Camera and LCD Memory			
MIPI Camera Memory	0x5010_4000	0x5010_4FFF	4
MIPI LCD Memory	0x5010_5000	0x5010_5FFF	4
LP Always On Peripherals (LP AON PERI)			
LP System Register	0x5011_0000	0x5011_0FFF	4
LP Always-on Clock and Reset	0x5011_1000	0x5011_1FFF	4
LP Timer	0x5011_2000	0x5011_2FFF	4
LP Analog peripherals (LP ANAPERI)	0x5011_3000	0x5011_3FFF	4
LP HUK	0x5011_4000	0x5011_4FFF	4
Power Management Unit (PMU)	0x5011_5000	0x5011_5FFF	4
LP Watch Dog Timer (LP WDT)	0x5011_6000	0x5011_6FFF	4
Reserved	0x5011_7000	0x5011_7FFF	
LP Mailbox (LP MB)	0x5011_8000	0x5011_8FFF	4
Reserved	0x5011_A000	0x5011_FFFF	
LP Peripherals (LP PERI)			
LP Peripheral Clock and Reset (LP PERI-CLKRST)	0x5012_0000	0x5012_0FFF	4

Cont'd on next page

Table 9.3-2 – cont'd from previous page

Target	Boundary Address		Size (KB)
	Low Address	High Address	
LP UART	0x5012_1000	0x5012_1FFF	4
LP I2C	0x5012_2000	0x5012_2FFF	4
LP SPI	0x5012_3000	0x5012_3FFF	4
LP Analog I2C	0x5012_4000	0x5012_4FFF	4
LP I2S	0x5012_5000	0x5012_5FFF	4
Reserved	0x5012_6000	0x5012_6FFF	4
LP ADC	0x5012_7000	0x5012_7FFF	4
Reserved	0x5012_8000	0x5012_9FFF	
LP GPIO Matrix	0x5012_A000	0x5012_AFFF	4
LP IO MUX	0x5012_B000	0x5012_BFFF	4
LP Interrupt (LP INTR)	0x5012_C000	0x5012_CFFF	4
LP eFuse	0x5012_D000	0x5012_DFFF	4
LP Peripheral Permission (LP_PERI_PMS)	0x5012_E000	0x5012_E7FF	2
HP2LP Peripheral Permission (HP2LP_PERI_PMS)	0x5012_E800	0x5012_EFFF	2
LP Temperature Sensor	0x5012_F000	0x5012_FFFF	4

Note:
As shown in Figure 9.3-1, HP CPU and LP CPU can access HP CPU PERI, HP PERI0, HP PERI1, LP AON PERI and LP PERI listed in Table 9.3-2.

9.4 eFuse Controller (EFUSE)

9.4.1 Overview

ESP32-P4 contains a 4096-bit eFuse memory to store parameters and user data. The parameters include control parameters for some hardware modules, system data parameters and keys used for the encryption/decryption module. Once an eFuse bit is programmed to 1, it can never be reverted to 0. The eFuse controller programs individual bits of parameters in eFuse according to user configurations. From outside the chip, eFuse data can only be read via the eFuse controller. For some data, such as some keys stored in eFuse for internal use by hardware cryptography modules (e.g., digital signature, HMAC), if read protection is not enabled, the data can be read from outside the chip; if read protection is enabled, the data cannot be read from outside the chip.

9.4.2 Features

- 4096-bit one-time programmable memory (including up to 1792 bits reserved for custom use, depending on whether all EFUSE_KEY_PURPOSE_*n* parameters are set to 0. For more information, see Table 9.4-2)
- Configurable write protection
- Configurable read protection
- Multiple hardware encoding schemes against data corruption

9.4.3 Functional Description

9.4.3.1 Structure

The eFuse system consists of the eFuse controller and eFuse memory. Data flow in this system is shown in Figure 9.4-1.

Users can program bits in the eFuse memory via the eFuse controller by writing the data to be programmed to the programming register and executing the programming instruction. For detailed programming steps, please refer to Section 9.4.3.2.

Users cannot directly read the data programmed in the eFuse memory, so they need to read the programmed data into the Reading Data Register of the corresponding address segment through the eFuse controller. During the reading process, if the data is inconsistent with that in the eFuse memory, the eFuse controller can automatically correct it through the hardware encoding mechanism (see Section 9.4.3.1.3 for details), and send the error message to the error report register. For detailed steps to read parameters, please refer to the Section 9.4.3.3.

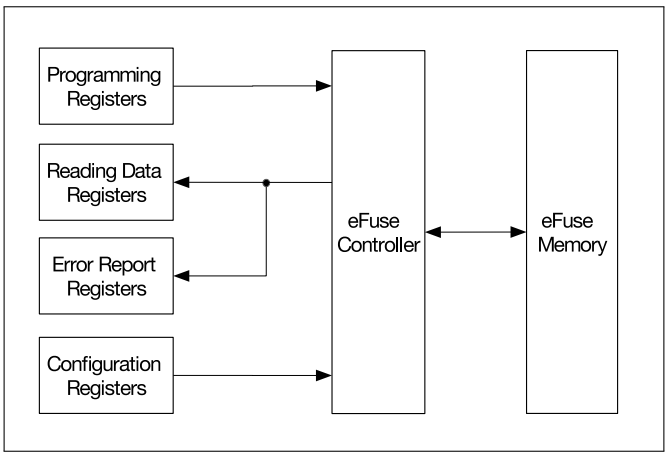


Figure 9.4-1. Data Flow in eFuse

Data in eFuse memory is organized in 11 blocks (BLOCK0 ~ BLOCK10).

BLOCK0 holds most parameters for software and hardware uses.

Table 9.4-1 lists the name, bit width, hardware accessibility, write-protection bit, and brief function description of every parameter in BLOCK0 that is accessible to users (readable and usable). For more information on each parameter, follow the link in the table.

The **EFUSE_WR_DIS** parameter controls write protection for other parameters. **EFUSE_RD_DIS** controls read protection for BLOCK4 ~ BLOCK10. For more information on these two parameters, please see Section 9.4.3.1.1 and Section 9.4.3.1.2.

Table 9.4-1. Parameters in eFuse BLOCK0

Parameters	Bit Width	Accessible by Hardware	Write Protection by EFUSE_WR_DIS Bit Number	Description
EFUSE_WR_DIS	32	Y	N/A	Represents whether writing of eFuse bits by eFuse controller is disabled.
EFUSE_RD_DIS	7	Y	0	Represents whether reading data from BLOCK4 ~ BLOCK10 in eFuse memory by users is disabled.
EFUSE_RECOVERY_BOOTLOADER_FLASH_SECTOR_0_1	2	N	N/A	Represents the starting flash sector (flash sector size is 0x1000) of the recovery bootloader used by the ROM bootloader if the primary bootloader fails.
EFUSE_DIS_USB_JTAG	1	Y	2	Represents whether the USB-to-JTAG function in the USB device module is disabled.
EFUSE_RECOVERY_BOOTLOADER_FLASH_SECTOR_2_2	1	N	N/A	Represents the starting flash sector (flash sector size is 0x1000) of the recovery bootloader used by the ROM bootloader if the primary bootloader fails.
EFUSE_DIS_FORCE_DOWNLOAD	1	Y	2	Set this bit to disable the function that forces chip into download mode.
EFUSE_SPI_DOWNLOAD_MSPI_DIS	1	Y	2	Set this bit to disable accessing MSPI flash/MSPI RAM by SYS AXI matrix during boot_mode_download.
EFUSE_DIS_TWAI	1	Y	2	Set this bit to disable TWAI function.
EFUSE_JTAG_SEL_ENABLE	1	Y	2	Set this bit to enable selection between usb_to_jtag and pad_to_jtag through strapping gpio25 when both EFUSE_DIS_PAD_JTAG and EFUSE_DIS_USB_JTAG are equal to 0.
EFUSE_SOFT_DIS_JTAG	3	Y	31	Represents whether JTAG is soft-disabled. It can be restarted via HMAC. Odd count of bits with a value of 1: Disabled, Even count of bits with a value of 1: Enabled.
EFUSE_DIS_PAD_JTAG	1	Y	2	Set this bit to hard-disable JTAG. (Permanent disable)

Cont'd on next page

Table 9.4-1 – cont'd from previous page

Parameters	Bit Width	Accessible by Hardware	Write Protection by EFUSE_WR_DIS Bit Number	Description
EFUSE_DIS_DOWNLOAD_MANUAL_ENCRYPT	1	Y	2	Set this bit to disable flash manual encrypt function (except in SPI boot mode).
EFUSE_RECOVERY_BOOTLOADER_FLASH_SECTOR_3_6	4	N	N/A	Represents the starting flash sector (flash sector size is 0x1000) of the recovery bootloader used by the ROM bootloader if the primary bootloader fails.
EFUSE_USB_PHY_SEL	1	Y	30	Configures the corresponding relationship between the two PHYs and the USB interfaces.
EFUSE_HUK_GEN_STATE	5	Y	19	Set the bits to control validation of HUK generate mode. The value is valid when the count of 1 bits is even, and invalid when odd.
EFUSE_RECOVERY_BOOTLOADER_FLASH_SECTOR_7_7	1	N	N/A	Represents the starting flash sector (flash sector size is 0x1000) of the recovery bootloader used by the ROM bootloader if the primary bootloader fails.
EFUSE_RECOVERY_BOOTLOADER_FLASH_SECTOR_8_10	3	N	N/A	Represents the starting flash sector (flash sector size is 0x1000) of the recovery bootloader used by the ROM bootloader if the primary bootloader fails.
EFUSE_RECOVERY_BOOTLOADER_FLASH_SECTOR_11_11	1	N	N/A	Represents the starting flash sector (flash sector size is 0x1000) of the recovery bootloader used by the ROM bootloader if the primary bootloader fails.
EFUSE_KM_RND_SWITCH_CYCLE	1	Y	1	Set the bits to control key manager random number switch cycle.

Cont'd on next page

Table 9.4-1 – cont'd from previous page

Parameters	Bit Width	Accessible by Hardware	Write Protection by EFUSE_WR_DIS Bit Number	Description
EFUSE_KM_DEPLOY_ONLY_ONCE	4	Y	1	This field, together with EFUSE_KM_DEPLOY_ONLY_ONCE_H form one field: EFUSE_KM_DEPLOY_ONLY_ONCE_H, EFUSE_KM_DEPLOY_ONLY_ONCE[3:0]. Set each bit to control whether corresponding key can only be deployed once. 1 is true, 0 is false. bit 0: ecsda, bit 1: xts, bit2: hmac, bit3: ds, bit4: psram
EFUSE_FORCE_USE_KEY_MANAGER_KEY	4	Y	1	This field, together with EFUSE_FORCE_USE_KEY_MANAGER_KEY_H, forms EFUSE_FORCE_USE_KEY_MANAGER_KEY_H, EFUSE_FORCE_USE_KEY_MANAGER_KEY[3:0]. Each bit determines whether the corresponding key must be sourced from the key manager.
EFUSE_FORCE_DISABLE_SW_INIT_KEY	1	Y	1	Set this bit to disable software written init key, and force use efuse_init_key.
EFUSE_KM_XTS_KEY_LENGTH_256	1	N	1	Set this bit to use XTS-512 key for flash encryption, otherwise XTS-256 key is used when the key manager is enabled.
EFUSE_ECC_FORCE_CONST_TIME	1	Y	14	Set this bit to permanently turn on ECC const-time mode.
EFUSE_WDT_DELAY_SEL	1	Y	2	Select LP WDT timeout threshold.
EFUSE_SPI_BOOT_CRYPT_CNT	3	Y	4	Set this bit to enable SPI boot encrypt/decrypt.
EFUSE_SECURE_BOOT_KEY_REVOKE0	1	N	5	Set this bit to enable revoking first secure boot key.
EFUSE_SECURE_BOOT_KEY_REVOKE1	1	N	6	Set this bit to enable revoking second secure boot key.
EFUSE_SECURE_BOOT_KEY_REVOKE2	1	N	7	Set this bit to enable revoking third secure boot key.
EFUSE_KEY_PURPOSE_0	4	Y	8	Purpose of Key0.
EFUSE_KEY_PURPOSE_1	4	Y	9	Purpose of Key1.
EFUSE_KEY_PURPOSE_2	4	Y	10	Purpose of Key2.
EFUSE_KEY_PURPOSE_3	4	Y	11	Purpose of Key3.

Cont'd on next page

GoBack

Table 9.4-1 – cont'd from previous page

Parameters	Bit Width	Accessible by Hardware	Write Protection by EFUSE_WR_DIS Bit Number	Description
EFUSE_KEY_PURPOSE_4	4	Y	12	Purpose of Key4.
EFUSE_KEY_PURPOSE_5	4	Y	13	Purpose of Key5.
EFUSE_SEC_DPA_LEVEL	2	Y	14	Configures the clock random divide mode to determine the DPA secure level.
EFUSE_XTS_DPA_CLK_ENABLE	1	Y	14	Sets this bit to enable xts clock anti-DPA attack function.
EFUSE_SECURE_BOOT_EN	1	N	15	Set this bit to enable secure boot.
EFUSE_SECURE_BOOT_AGGRESSIVE_REVOKE	1	N	16	Set this bit to enable revoking aggressive secure boot.
EFUSE_KM_DEPLOY_ONLY_ONCE_H	1	Y	1	This field, together with EFUSE_KM_DEPLOY_ONLY_ONCE, forms EFUSE_KM_DEPLOY_ONLY_ONCE_H, EFUSE_KM_DEPLOY_ONLY_ONCE[3:0]. Each bit controls whether the corresponding key can be deployed only once.
EFUSE_FORCE_USE_KEY_MANAGER_KEY_H	1	Y	1	Form one field together with EFUSE_FORCE_USE_KEY_MANAGER_KEY: EFUSE_FORCE_USE_KEY_MANAGER_KEY_H, EFUSE_FORCE_USE_KEY_MANAGER_KEY[3:0]. Set each bit to control whether corresponding key must come from key manager.
EFUSE_FLASH_ECC_EN	1	N	18	Set this bit to enable ECC for flash boot.
EFUSE_DIS_USB_OTG_DOWNLOAD_MODE	1	N	18	Set this bit to disable download via USB-OTG.
EFUSE_FLASH_TPUW	4	N	18	Configures flash waiting time after power-up, in unit of ms.
EFUSE_DIS_DOWNLOAD_MODE	1	N	18	Set this bit to disable download mode (boot_mode[3:0] = 0, 1, 2, 4, 5, 6, 7).
EFUSE_DIS_DIRECT_BOOT	1	N	18	Set this bit to disable direct boot mode.
EFUSE_DIS_USB_SERIAL_JTAG_ROM_PRINT	1	N	18	Set this bit to disable USB-Serial-JTAG print during rom boot.

Cont'd on next page

GoBack

Table 9.4-1 – cont'd from previous page

Parameters	Bit Width	Accessible by Hardware	Write Protection by EFUSE_WR_DIS Bit Number	Description
EFUSE_LOCK_KM_KEY	1	N	1	set this bit to lock the key manager key after deploy.
EFUSE_DIS_USB_SERIAL_JTAG_DOWNLOAD_MODE	1	N	18	Set this bit to disable the USB-Serial-JTAG download function.
EFUSE_ENABLE_SECURITY_DOWNLOAD	1	N	18	Set this bit to enable security download mode.
EFUSE_UART_PRINT_CONTROL	2	N	18	Set the type of UART printing.
EFUSE_FORCE_SEND_RESUME	1	N	18	Set this bit to force ROM code to send a resume command during SPI boot.
EFUSE_SECURE_VERSION	16	N	18	Secure version used by ESP-IDF anti-rollback feature.
EFUSE_SECURE_BOOT_DISABLE_FAST_WAKE	1	N	18	Represents whether FAST VERIFY ON WAKE is disabled when Secure Boot is enabled.
EFUSE_HYS_EN_PAD	1	Y	2	Set bits to enable hysteresis function of PAD0 27.
EFUSE_KEY_PURPOSE_0_H	1	Y	8	Purpose of Key0 (5th bit).
EFUSE_KEY_PURPOSE_1_H	1	Y	9	Purpose of Key1 (5th bit).
EFUSE_KEY_PURPOSE_2_H	1	Y	10	Purpose of Key2. (5th bit).
EFUSE_KEY_PURPOSE_3_H	1	Y	11	Purpose of Key3. (5th bit).
EFUSE_KEY_PURPOSE_4_H	1	Y	12	Purpose of Key4. (5th bit).
EFUSE_OPXA_TIEH_SEL_0	2	Y	2	Output LDO VO0 tieh source select.
EFUSE_PVT_GLITCH_EN	1	Y	3	Represents whether to enable PVT power glitch monitor function.
EFUSE_KEY_PURPOSE_5_H	1	Y	13	Purpose of Key5. (5th bit).
EFUSE_KM_DISABLE_DEPLOY_MODE_H	1	Y	1	EFUSE_KM_DISABLE_DEPLOY_MODE and EFUSE_KM_DISABLE_DEPLOY_MODE_H together form one field: EFUSE_KM_DISABLE_DEPLOY_MODE_H, EFUSE_KM_DISABLE_DEPLOY_MODE[3:0]. Set each bit to control whether corresponding key's deploy mode of new value deployment is disabled.

Cont'd on next page

Table 9.4-1 – cont'd from previous page

Parameters	Bit Width	Accessible by Hardware	Write Protection by EFUSE_WR_DIS Bit Number	Description
EFUSE_KM_DISABLE_DEPLOY_MODE	4	Y	1	This field, together with EFUSE_KM_DISABLE_DEPLOY_MODE_H, forms a complete field EFUSE_KM_DISABLE_DEPLOY_MODE_H, EFUSE_KM_DISABLE_DEPLOY_MODE[3:0]. Each bit controls whether the deploy mode for new value deployment of the corresponding key is disabled.
EFUSE_XTS_DPA_PSEUDO_LEVEL	2	Y	14	Sets this bit to control the xts pseudo-round anti-dpa attack function.
EFUSE_HP_PWR_SRC_SEL	1	Y	17	HP system power source select.
EFUSE_SECURE_BOOT_SHA384_EN	1	N	N/A	Represents whether secure boot using SHA-384 is enabled.
EFUSE_DIS_WDT	1	Y	2	Set this bit to disable watch dog.
EFUSE_DIS_SWD	1	Y	2	Set bit to disable super-watchdog.
EFUSE_PVT_GLITCH_MODE	2	Y	3	Use to configure glitch mode.

Table 9.4-2 lists all key purposes and their values. Setting the eFuse parameter EFUSE_KEY_PURPOSE_*n* declares the purpose of KEY*n* (*n*: 0 ~ 5).

Table 9.4-2. Secure Key Purpose Values

Key Purpose Values	Purposes
0	Designated for user use
1	Designated for ECDSA_KEY
2	Designated for XTS_AES_256_KEY_1
3	Designated for XTS_AES_256_KEY_2
4	Designated for XTS_AES_128_KEY (for flash/SRAM encryption and decryption)
5	Designated for HMAC Downstream mode (JTAG and Digital Signature)
6	Designated for JTAG in HMAC Downstream mode
7	Designated for Digital Signature in HMAC Downstream mode
8	Designated for HMAC Upstream mode
9	Designated for SECURE_BOOT_DIGEST0 (secure boot key digest)
10	Designated for SECURE_BOOT_DIGEST1 (secure boot key digest)
11	Designated for SECURE_BOOT_DIGEST2 (secure boot key digest)
12	Designated for KM_INIT_KEY

Table 9.4-3 provides the details of parameters in BLOCK1 ~ BLOCK10.

Table 9.4-3. Parameters in BLOCK1 to BLOCK10

BLOCK	Parameters	Bit Width	Accessible by Hardware	Write Protection by EFUSE_WR_DIS Bit Number	Read Protection by EFUSE_RD_DIS Bit Number	Description
BLOCK1	EFUSE_MAC	48	Y	20	N/A	Represents the MAC address (lower 48 bits).
	EFUSE_SYS_DATA_PART0	78	N	20	N/A	Represents the first 78 bits of Part 0 of system data.
BLOCK2	EFUSE_SYS_DATA_PART1	256	N	21	N/A	System data
BLOCK3	EFUSE_USR_DATA	256	N	22	N/A	User data
BLOCK4	EFUSE_KEY0_DATA	256	Y	23	0	KEY0 or user data
BLOCK5	EFUSE_KEY1_DATA	256	Y	24	1	KEY1 or user data
BLOCK6	EFUSE_KEY2_DATA	256	Y	25	2	KEY2 or user data
BLOCK7	EFUSE_KEY3_DATA	256	Y	26	3	KEY3 or user data
BLOCK8	EFUSE_KEY4_DATA	256	Y	27	4	KEY4 or user data
BLOCK9	EFUSE_KEY5_DATA	256	Y	28	5	KEY5 or user data
BLOCK10	EFUSE_SYS_DATA_PART2	192	N	29	6	Represents the first 32 bits of Part 2 of system data.
	EFUSE_USB_DEVICE_EXCHG_PINS	1	Y	29	6	Enables swapping of the D+ and D- pins for USB-Serial-JTAG.
	EFUSE_USB_OTG11_EXCHG_PINS	1	Y	29	6	Enables swapping of the D+ and D- pins for USB OTG11.

Among these blocks, BLOCK4 ~ 9 can be used to store KEY0 ~ 5. Up to six 256-bit keys can be written into eFuse. Whenever a key is written, its purpose value should also be written (see table 9.4-2). For example, when a key for the JTAG function in HMAC Downstream mode is written to KEY3 (i.e., BLOCK7), its key purpose value 6 should also be written to EFUSE_KEY_PURPOSE_3.

Note:
Do not program the XTS-AES key into the KEY5 block, i.e., BLOCK9. Otherwise, the key may be unreadable. Instead, program it into the preceding blocks, i.e., BLOCK4 ~ BLOCK8. The last block, BLOCK9, is used to program other keys.

BLOCK1 ~ BLOCK10 use the RS coding scheme, so there are some limitations on writing to these parameters. For more detailed information, please refer to Section 9.4.3.1.3 and Section 9.4.3.2.

9.4.3.1.1 EFUSE_WR_DIS

Parameter EFUSE_WR_DIS determines whether individual eFuse parameters are write-protected. After EFUSE_WR_DIS has been programmed, execute an eFuse read operation so the new values would take effect.

Column "Write Protection by EFUSE_WR_DIS Bit Number" in Table 9.4-1 and Table 9.4-3 list the specific bits in EFUSE_WR_DIS that disable writing.

When the write protection bit of a parameter is set to 0, it means that this parameter is not write-protected and can be programmed, unless it has been programmed before.

When the write protection bit of a parameter is set to 1, it means that this parameter is write-protected and none of its bits can be modified, with non-programmed bits always remaining 0 and programmed bits always remaining 1. That is to say, if a parameter is write-protected, it will always remain in this state and cannot be changed.

9.4.3.1.2 EFUSE_RD_DIS

Among all parameters, only those in BLOCK4 ~ BLOCK10 can be configured for user read protection, i.e., the entries in Table 9.4-3 for which column "Read Protection by EFUSE_RD_DIS Bit Number" is not N/A. After EFUSE_RD_DIS has been programmed, execute an eFuse read operation so the new values would take effect.

If the corresponding EFUSE_RD_DIS bit is 0, the parameter controlled by this bit is not read-protected from users. If it is 1, the parameter controlled by it is read-protected from users.

Other parameters that are not in BLOCK4 ~ BLOCK10 can always be read by users.

When BLOCK4 ~ BLOCK10 are set to be read-protected, the data in them can still be read by hardware cryptography modules if the EFUSE_KEY_PURPOSE_*n* bit is set accordingly.

9.4.3.1.3 Data Storage

Internally, eFuse uses the hardware encoding scheme to protect data from corruption. The scheme and the encoding process are invisible to users.

All BLOCK0 parameters except for EFUSE_WR_DIS are stored with four backups, meaning each bit is stored four times. This backup scheme is not visible to users.

In BLOCK0, EFUSE_WR_DIS occupies 32 bits, and each of the other parameters occupies 152 bits. Therefore, BLOCK0 occupies $32 + 152 \times 4 = 640$ bits in eFuse memory.

BLOCK1 ~ BLOCK10 use RS (44, 32) coding scheme that supports up to 6 bytes of automatic error correction. The primitive polynomial of RS (44, 32) is $p(x) = x^8 + x^4 + x^3 + x^2 + 1$.

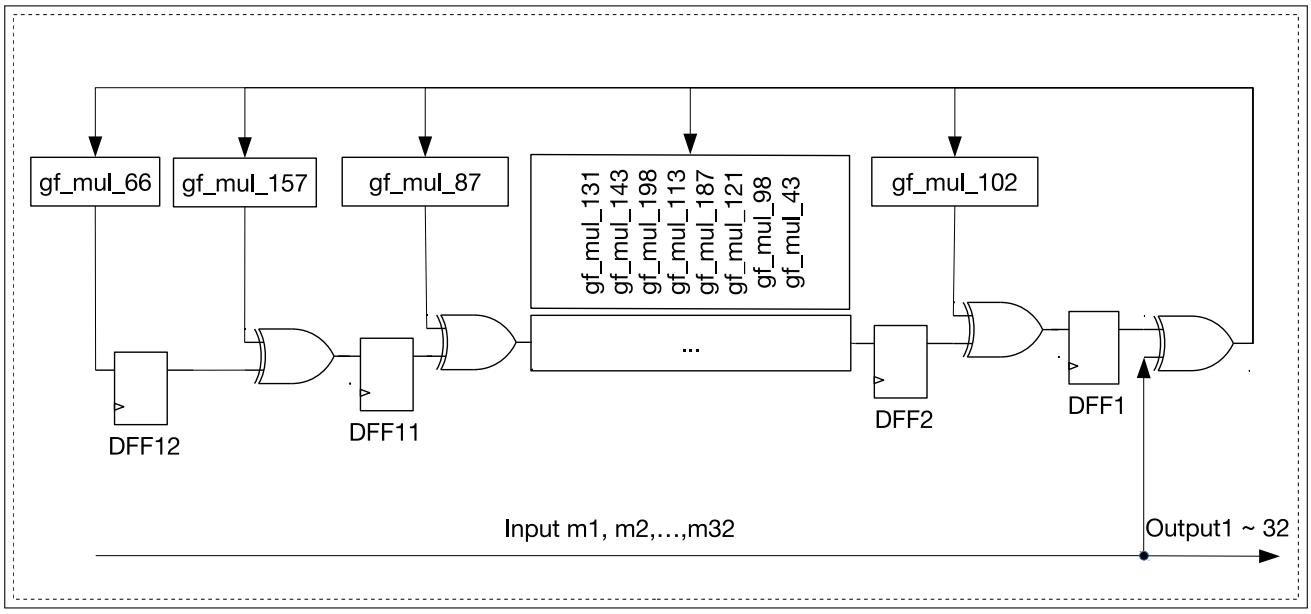


Figure 9.4-2. Shift Register Circuit (first 32 bytes)

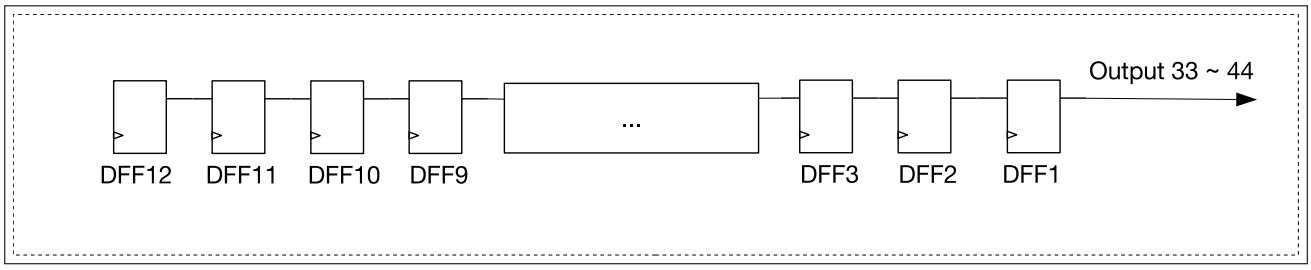


Figure 9.4-3. Shift Register Circuit (last 12 bytes)

The shift register circuit shown in Figure 9.4-2 and 9.4-3 processes 32 data bytes using RS (44, 32). This coding scheme encodes 32 bytes of data into 44 bytes:

- Bytes [0:31] are the data bytes themselves
- Bytes [32:43] are the encoded parity bytes stored in 8-bit flip-flops DFF1, DFF2, ..., DFF12 (gf_mul_n is the result of multiplying a byte of data in $GF(2^8)$ by α^n, where n is an integer).

After that, the hardware programs into eFuse the 44-byte codeword consisting of the data bytes and the parity bytes. When the eFuse block is read, the eFuse controller automatically decodes the codeword and applies error correction if needed.

Because the RS check codes are generated on the entire 32-byte eFuse block, each block can only be written once.

Since the size of BLOCK1 is less than 32 bytes, the unused bits will be treated as 0 by hardware during the RS (44, 32) encoding. Thus, the final coding result will not be affected.

Among blocks using the RS (44, 32) coding scheme, the parameters in BLOCK1 is 24 bytes, and the RS check code is 12 bytes, so BLOCK1 occupies 24 + 12 = 36 bytes in eFuse memory.

In each of the other blocks (BLOCK2~BLOCK10), the parameter data is 32 bytes and the RS check code is 12 bytes, so those nine blocks occupy $(32 + 12) \times 9 = 396$ bytes in eFuse memory.

9.4.3.2 Programming of Parameters

The eFuse controller can only program eFuse parameters in one block at a time. BLOCK0 ~ BLOCK10 share the same address range to store the parameters to be programmed. Configure parameter EFUSE_BLK_NUM to indicate which block should be programmed.

Since there is a one-to-one correspondence between the reading data registers and the programming data registers (see table 9.4-4 for details), users can find out where the data to be programmed is located in programming registers by checking the parameter description and the parameter location in the corresponding read registers.

For example, if the user wants to program the parameter EFUSE_USB_DEVICE_EXCHG_PINS in BLOCK0 to 1, they can first search the reading data registers EFUSE_RD_REPEAT_DATA0 ~ 4_REG in BLOCK0 for where the parameter is located, namely, the 8th bit in EFUSE_RD_REPEAT_DATA0_REG. So, the user can set the 8th bit of EFUSE_PGM_DATA1_REG to 1 and follow the programming steps below. After the steps are completed, the corresponding bit in the eFuse memory will be programmed to 1.

Programming preparation

- Programming BLOCK0
  1. Set EFUSE_BLK_NUM to 0.
  2. Write into EFUSE_PGM_DATA0_REG ~ EFUSE_PGM_DATA5_REG the data to be programmed to BLOCK0.
     The data in EFUSE_PGM_DATA6_REG ~ EFUSE_PGM_DATA7_REG and EFUSE_PGM_CHECK_VALUE0_REG ~ EFUSE_PGM_CHECK_VALUE2_REG does not affect the programming of BLOCK0.
- BLOCK1 cannot be programmed by users as it has been programmed at manufacturing.
- Programming BLOCK2 ~ 10
  1. Set EFUSE_BLK_NUM to the block number.
  2. Write into EFUSE_PGM_DATA0_REG ~ EFUSE_PGM_DATA7_REG the data to be programmed. Write into EFUSE_PGM_CHECK_VALUE0_REG ~ EFUSE_PGM_CHECK_VALUE2_REG the corresponding RS code.

Programming process

The process of programming parameters is as follows:

1. Configure the value of parameter EFUSE_BLK_NUM to determine the block to be programmed.
2. Write parameters to be programmed to registers EFUSE_PGM_DATA0_REG ~ EFUSE_PGM_DATA7_REG and EFUSE_PGM_CHECK_VALUE0_REG ~ EFUSE_PGM_CHECK_VALUE2_REG.
3. Make sure the eFuse programming voltage VDDQ is configured correctly as described in Section 9.4.3.4.
4. Configure the field EFUSE_OP_CODE of register EFUSE_CONF_REG to 0x5A5A.

5. Configure the field EFUSE_PGM_CMD of register EFUSE_CMD_REG to 1.
6. Poll register EFUSE_CMD_REG until it is 0x0, or wait for a PGM_DONE interrupt. For more information on how to identify a PGM_DONE or READ_DONE interrupt, please see the end of Section 9.4.3.3.
7. Clear the parameters in EFUSE_PGM_DATA0_REG ~ EFUSE_PGM_DATA7_REG and EFUSE_PGM_CHECK_VALUE0_REG ~ EFUSE_PGM_CHECK_VALUE2_REG.
8. Trigger an eFuse read operation (see Section 9.4.3.3) to update eFuse registers with the new values.
9. Check error record registers. If the values read in error record registers are not 0, run the programming process again following steps 1 ~ 7 above. This can resolve cases where error record registers are non-zero due to insufficient programming. The error record registers to check for each eFuse block are as follows:
    - BLOCK0: EFUSE_RD_REPEAT_DATA_ERR0_REG ~ EFUSE_RD_REPEAT_DATA_ERR4_REG
    - BLOCK1: EFUSE_RD_MAC_SYS_ERR_NUM, EFUSE_RD_MAC_SYS_FAIL
    - BLOCK2: EFUSE_RD_SYS_PART1_DATA_ERR_NUM, EFUSE_RD_SYS_PART1_DATA_FAIL
    - BLOCK3: EFUSE_RD_USR_DATA_ERR_NUM, EFUSE_RD_USR_DATA_FAIL
    - BLOCK4: EFUSE_RD_KEY0_DATA_ERR_NUM, EFUSE_RD_KEY0_DATA_FAIL
    - BLOCK5: EFUSE_RD_KEY1_DATA_ERR_NUM, EFUSE_RD_KEY1_DATA_FAIL
    - BLOCK6: EFUSE_RD_KEY2_DATA_ERR_NUM, EFUSE_RD_KEY2_DATA_FAIL
    - BLOCK7: EFUSE_RD_KEY3_DATA_ERR_NUM, EFUSE_RD_KEY3_DATA_FAIL
    - BLOCK8: EFUSE_RD_KEY4_DATA_ERR_NUM, EFUSE_RD_KEY4_DATA_FAIL
    - BLOCK9: EFUSE_RD_KEY5_DATA_ERR_NUM, EFUSE_RD_KEY5_DATA_FAIL
    - BLOCK10: EFUSE_RD_SYS_PART2_DATA_ERR_NUM, EFUSE_RD_SYS_PART2_DATA_FAIL

Limitations

In BLOCK0, different parameters, or even different bits within the same parameter, may be programmed across multiple programming cycles. However, this is not recommended; the number of programming cycles should be minimized. We recommend programming all bits of a parameter that need to be set in a single programming action. In addition, after all parameters controlled by a given bit of EFUSE_WR_DIS are programmed, that bit should be programmed immediately. The parameters controlled by that bit and the write-protection bit itself may even be programmed together in one action.

BLOCK1 cannot be programmed by users as it has been programmed at manufacturing.

BLOCK2 ~ 10 can only be programmed once. Repeated programming is not allowed.

9.4.3.3 Reading of Parameters by Users

Users cannot read eFuse bits directly. The eFuse controller hardware reads all eFuse bits and stores the results to their corresponding registers in its memory space. Then, users can read eFuse bits by reading the registers that start with EFUSE_RD_. Details are provided in Table 9.4-4.

Table 9.4-4. User Read Registers and Programming Registers

BLOCK	Read Registers	Registers When Programming This Block
0	EFUSE_RD_WR_DIS_REG	EFUSE_PGM_DATA0_REG
0	EFUSE_RD_REPEAT_DATA0 ~ 4_REG	EFUSE_PGM_DATA1 ~ 5_REG
1	EFUSE_RD_MAC_SYS_0 ~ 5_REG	EFUSE_PGM_DATA0 ~ 5_REG
2	EFUSE_RD_SYS_PART1_DATA0 ~ 7_REG	EFUSE_PGM_DATA0 ~ 7_REG
3	EFUSE_RD_USR_DATA0 ~ 7_REG	EFUSE_PGM_DATA0 ~ 7_REG
4-9	EFUSE_RD_KEY*n*_DATA0 ~ 7_REG (*n*: 0 ~ 5)	EFUSE_PGM_DATA0 ~ 7_REG
10	EFUSE_RD_SYS_PART2_DATA0 ~ 7_REG	EFUSE_PGM_DATA0 ~ 7_REG

Updating reading data registers

The eFuse controller reads eFuse memory to update corresponding registers. This read operation happens at system reset and can also be triggered manually by users as needed (e.g., if new eFuse values have been programmed). The process of triggering a read operation by users is as follows:

1. Configure the field EFUSE_OP_CODE in register EFUSE_CONF_REG to 0x5AA5.
2. Configure the field EFUSE_READ_CMD in register EFUSE_CMD_REG to 1.
3. Poll register EFUSE_CMD_REG until it is 0x0, or wait for a READ_DONE interrupt. Information on how to identify a PGM_DONE or READ_DONE interrupt is provided below in this section.
4. Read the values of each parameter from eFuse memory.

The eFuse read registers will hold all values until the next read operation.

Programming error detection

The programming error record registers allow users to check the integrity of parameters stored in the eFuse memory. For instance, they can help detect whether the four-backup parameters are consistent and whether parameters protected by RS encoding are decoded successfully.

Registers EFUSE_RD_REPEAT_ERR0 ~ 3_REG indicate if there are any errors in programming parameters (except EFUSE_WR_DIS) to BLOCK0. The value 1 indicates an error is detected in programming the corresponding bit. The value 0 indicates no error.

Registers EFUSE_RD_RS_ERR0 ~ 1_REG store the number of corrected bytes as well as the result of RS decoding when eFuse controller reads BLOCK1 ~ BLOCK10.

The values of the above registers will be updated every time the reading data registers of eFuse controller have been updated.

Identifying completion of program or read operations

The methods to identify the completion of a program/read operation are described below. Please note that bit 1 corresponds to a program operation, and bit 0 corresponds to a read operation.

- Method one: Poll bit 1/0 in register EFUSE_INT_RAW_REG until it becomes 1, which represents the completion of a program/read operation.
- Method two:
  1. Set bit 1/0 in register EFUSE_INT_ENA_REG to 1 to enable the eFuse controller to post a PGM_DONE or READ_DONE interrupt.

2. Configure the Interrupt Matrix to enable the CPU to respond to eFuse interrupt signals. See Chapter 13 *Interrupt Matrix*.
3. Wait for the PGM_DONE or READ_DONE interrupt.
4. Set bit 1/0 in register EFUSE_INT_CLR_REG to 1 to clear the PGM_DONE or READ_DONE interrupt.

Note

When eFuse controller is updating its registers, it will use EFUSE_PGM_DATA*n*_REG (n=0, 1, ... ,7) again to store data. So please do not write important data into these registers before this updating process is initiated.
During the chip boot process, eFuse controller will automatically update data from eFuse memory into the registers that can be accessed by users. Users can get programmed eFuse data by reading corresponding registers. Thus, there is no need to update the reading data registers in such case.

9.4.3.4 eFuse VDDQ Timing

The eFuse controller operates at the clock frequency of 20 MHz, and its programming voltage VDDQ should be configured as follows:

- EFUSE_DAC_NUM (the rising period of VDDQ): The default value of VDDQ is 2.5 V and the voltage increases by 0.01 V in each clock cycle. The default value of this parameter is 255.
- EFUSE_DAC_CLK_DIV (the clock divisor of VDDQ): The clock period to program VDDQ should be larger than 1 μs.
- EFUSE_PWR_ON_NUM (the power-on time for VDDQ): The programming voltage should be stabilized after this time, which means the value of this parameter should be configured to exceed the result of EFUSE_DAC_CLK_DIV × EFUSE_DAC_NUM.
- EFUSE_PWR_OFF_NUM (the power-off time for VDDQ): The value of this parameter should be larger than 10 μs.

Table 9.4-5. Configuration of Default VDDQ Timing Parameters

EFUSE_DAC_NUM	EFUSE_DAC_CLK_DIV	EFUSE_PWR_ON_NUM	EFUSE_PWR_OFF_NUM
0xFF	0x28	0x3000	0x190

9.4.3.5 Parameters Used by Hardware Modules

Some hardware modules are directly connected to the eFuse peripheral in order to use the parameters that are marked with "Y" in columns "Accessible by Hardware" of Table 9.4-1 and Table 9.4-3. Users cannot intervene in this process.

9.4.4 Interrupts

ESP32-P4's eFuse controller can generate the following interrupt signal that will be sent to the Interrupt Matrix.

- EFUSE_INT

There are several internal interrupt sources from eFuse controller that can generate the above interrupt signal. The interrupt sources from eFuse controller are listed with their trigger conditions and the resulted interrupt signal in Table 9.4-6.

Table 9.4-6. eFuse's Internal Interrupt Sources

Internal Interrupt Source	Trigger Condition	Interrupt Signal
EFUSE_PGM_DONE_INT	Programming of eFuse completes	EFUSE_INT
EFUSE_READ_DONE_INT	Reading of eFuse completes	

Note:
For definitions of *interrupt*, *interrupt signal*, *interrupt source*, and their correlations, please refer to Chapter 13 *Interrupt Matrix* > Section 13.2 *Interrupt Terminology in ESP32-P4*.

Each interrupt source can be configured by a common set of registers that are described in Section *Interrupt Configuration Registers*. The specific registers can be found in Section 9.4.5 *Register Summary*.

9.4.5 Register Summary

The addresses in this section are relative to eFuse controller base address provided in Table 9.3-2 in Chapter 9 *System and Memory*.

The abbreviations given in Column **Access** are explained in Section *Access Types for Registers*.

Name	Description	Address	Access
Programming Data Registers			
EFUSE_PGM_DATA0_REG	Represents pgm_data0	0x0000	R/W
EFUSE_PGM_DATA1_REG	Represents pgm_data1	0x0004	R/W
EFUSE_PGM_DATA2_REG	Represents pgm_data2	0x0008	R/W
EFUSE_PGM_DATA3_REG	Represents pgm_data3	0x000C	R/W
EFUSE_PGM_DATA4_REG	Represents pgm_data4	0x0010	R/W
EFUSE_PGM_DATA5_REG	Represents pgm_data5	0x0014	R/W
EFUSE_PGM_DATA6_REG	Represents pgm_data6	0x0018	R/W
EFUSE_PGM_DATA7_REG	Represents pgm_data7	0x001C	R/W
EFUSE_PGM_CHECK_VALUE0_REG	Represents pgm_check_value0	0x0020	R/W
EFUSE_PGM_CHECK_VALUE1_REG	Represents pgm_check_value1	0x0024	R/W
EFUSE_PGM_CHECK_VALUE2_REG	Represents pgm_check_value2	0x0028	R/W
Block 0 Registers			
EFUSE_RD_WR_DIS_REG	Represents rd_wr_dis	0x002C	RO
EFUSE_RD_REPEAT_DATA0_REG	Represents rd_repeat_data	0x0030	RO
EFUSE_RD_REPEAT_DATA1_REG	Represents rd_repeat_data	0x0034	RO
EFUSE_RD_REPEAT_DATA2_REG	Represents rd_repeat_data	0x0038	RO
EFUSE_RD_REPEAT_DATA3_REG	Represents rd_repeat_data	0x003C	RO
EFUSE_RD_REPEAT_DATA4_REG	Represents rd_repeat_data	0x0040	RO
Block 1 Registers			
EFUSE_RD_MAC_SYS0_REG	Represents rd_mac_sys	0x0044	RO
EFUSE_RD_MAC_SYS1_REG	Represents rd_mac_sys	0x0048	RO
EFUSE_RD_MAC_SYS3_REG	Represents rd_mac_sys	0x0050	RO
EFUSE_RD_MAC_SYS4_REG	Represents rd_mac_sys	0x0054	RO
EFUSE_RD_MAC_SYS5_REG	Represents rd_mac_sys	0x0058	RO
Block 2 Registers			
EFUSE_RD_SYS_PART1_DATA0_REG	Represents rd_sys_part1_data0	0x005C	RO
EFUSE_RD_SYS_PART1_DATA1_REG	Represents rd_sys_part1_data1	0x0060	RO
EFUSE_RD_SYS_PART1_DATA2_REG	Represents rd_sys_part1_data2	0x0064	RO
EFUSE_RD_SYS_PART1_DATA3_REG	Represents rd_sys_part1_data3	0x0068	RO
EFUSE_RD_SYS_PART1_DATA4_REG	Represents rd_sys_part1_data4	0x006C	RO
EFUSE_RD_SYS_PART1_DATA5_REG	Represents rd_sys_part1_data5	0x0070	RO
EFUSE_RD_SYS_PART1_DATA6_REG	Represents rd_sys_part1_data6	0x0074	RO
EFUSE_RD_SYS_PART1_DATA7_REG	Represents rd_sys_part1_data7	0x0078	RO
Block 3 Registers			
EFUSE_RD_USR_DATA0_REG	Represents rd_usr_data0	0x007C	RO
EFUSE_RD_USR_DATA1_REG	Represents rd_usr_data1	0x0080	RO
EFUSE_RD_USR_DATA2_REG	Represents rd_usr_data2	0x0084	RO

Name	Description	Address	Access
EFUSE_RD_USR_DATA3_REG	Represents rd_usr_data3	0x0088	RO
EFUSE_RD_USR_DATA4_REG	Represents rd_usr_data4	0x008C	RO
EFUSE_RD_USR_DATA5_REG	Represents rd_usr_data5	0x0090	RO
EFUSE_RD_USR_DATA6_REG	Represents rd_usr_data6	0x0094	RO
EFUSE_RD_USR_DATA7_REG	Represents rd_usr_data7	0x0098	RO
Block 4 Registers			
EFUSE_RD_KEY0_DATA0_REG	Represents rd_key0_data0	0x009C	RO
EFUSE_RD_KEY0_DATA1_REG	Represents rd_key0_data1	0x00A0	RO
EFUSE_RD_KEY0_DATA2_REG	Represents rd_key0_data2	0x00A4	RO
EFUSE_RD_KEY0_DATA3_REG	Represents rd_key0_data3	0x00A8	RO
EFUSE_RD_KEY0_DATA4_REG	Represents rd_key0_data4	0x00AC	RO
EFUSE_RD_KEY0_DATA5_REG	Represents rd_key0_data5	0x00B0	RO
EFUSE_RD_KEY0_DATA6_REG	Represents rd_key0_data6	0x00B4	RO
EFUSE_RD_KEY0_DATA7_REG	Represents rd_key0_data7	0x00B8	RO
Block 5 Registers			
EFUSE_RD_KEY1_DATA0_REG	Represents rd_key1_data0	0x00BC	RO
EFUSE_RD_KEY1_DATA1_REG	Represents rd_key1_data1	0x00C0	RO
EFUSE_RD_KEY1_DATA2_REG	Represents rd_key1_data2	0x00C4	RO
EFUSE_RD_KEY1_DATA3_REG	Represents rd_key1_data3	0x00C8	RO
EFUSE_RD_KEY1_DATA4_REG	Represents rd_key1_data4	0x00CC	RO
EFUSE_RD_KEY1_DATA5_REG	Represents rd_key1_data5	0x00D0	RO
EFUSE_RD_KEY1_DATA6_REG	Represents rd_key1_data6	0x00D4	RO
EFUSE_RD_KEY1_DATA7_REG	Represents rd_key1_data7	0x00D8	RO
Block 6 Registers			
EFUSE_RD_KEY2_DATA0_REG	Represents rd_key2_data0	0x00DC	RO
EFUSE_RD_KEY2_DATA1_REG	Represents rd_key2_data1	0x00E0	RO
EFUSE_RD_KEY2_DATA2_REG	Represents rd_key2_data2	0x00E4	RO
EFUSE_RD_KEY2_DATA3_REG	Represents rd_key2_data3	0x00E8	RO
EFUSE_RD_KEY2_DATA4_REG	Represents rd_key2_data4	0x00EC	RO
EFUSE_RD_KEY2_DATA5_REG	Represents rd_key2_data5	0x00F0	RO
EFUSE_RD_KEY2_DATA6_REG	Represents rd_key2_data6	0x00F4	RO
EFUSE_RD_KEY2_DATA7_REG	Represents rd_key2_data7	0x00F8	RO
Block 7 Registers			
EFUSE_RD_KEY3_DATA0_REG	Represents rd_key3_data0	0x00FC	RO
EFUSE_RD_KEY3_DATA1_REG	Represents rd_key3_data1	0x0100	RO
EFUSE_RD_KEY3_DATA2_REG	Represents rd_key3_data2	0x0104	RO
EFUSE_RD_KEY3_DATA3_REG	Represents rd_key3_data3	0x0108	RO
EFUSE_RD_KEY3_DATA4_REG	Represents rd_key3_data4	0x010C	RO
EFUSE_RD_KEY3_DATA5_REG	Represents rd_key3_data5	0x0110	RO
EFUSE_RD_KEY3_DATA6_REG	Represents rd_key3_data6	0x0114	RO
EFUSE_RD_KEY3_DATA7_REG	Represents rd_key3_data7	0x0118	RO
Block 8 Registers			
EFUSE_RD_KEY4_DATA0_REG	Represents rd_key4_data0	0x011C	RO

Name	Description	Address	Access
EFUSE_RD_KEY4_DATA1_REG	Represents rd_key4_data1	0x0120	RO
EFUSE_RD_KEY4_DATA2_REG	Represents rd_key4_data2	0x0124	RO
EFUSE_RD_KEY4_DATA3_REG	Represents rd_key4_data3	0x0128	RO
EFUSE_RD_KEY4_DATA4_REG	Represents rd_key4_data4	0x012C	RO
EFUSE_RD_KEY4_DATA5_REG	Represents rd_key4_data5	0x0130	RO
EFUSE_RD_KEY4_DATA6_REG	Represents rd_key4_data6	0x0134	RO
EFUSE_RD_KEY4_DATA7_REG	Represents rd_key4_data7	0x0138	RO
Block 9 Registers			
EFUSE_RD_KEY5_DATA0_REG	Represents rd_key5_data0	0x013C	RO
EFUSE_RD_KEY5_DATA1_REG	Represents rd_key5_data1	0x0140	RO
EFUSE_RD_KEY5_DATA2_REG	Represents rd_key5_data2	0x0144	RO
EFUSE_RD_KEY5_DATA3_REG	Represents rd_key5_data3	0x0148	RO
EFUSE_RD_KEY5_DATA4_REG	Represents rd_key5_data4	0x014C	RO
EFUSE_RD_KEY5_DATA5_REG	Represents rd_key5_data5	0x0150	RO
EFUSE_RD_KEY5_DATA6_REG	Represents rd_key5_data6	0x0154	RO
EFUSE_RD_KEY5_DATA7_REG	Represents rd_key5_data7	0x0158	RO
Block 10 Registers			
EFUSE_RD_SYS_PART2_DATA0_REG	Represents rd_sys_part2_data	0x015C	RO
EFUSE_RD_SYS_PART2_DATA1_REG	Represents rd_sys_part2_data	0x0160	RO
EFUSE_RD_SYS_PART2_DATA2_REG	Represents rd_sys_part2_data	0x0164	RO
EFUSE_RD_SYS_PART2_DATA3_REG	Represents rd_sys_part2_data	0x0168	RO
EFUSE_RD_SYS_PART2_DATA4_REG	Represents rd_sys_part2_data	0x016C	RO
EFUSE_RD_SYS_PART2_DATA5_REG	Represents rd_sys_part2_data	0x0170	RO
EFUSE_RD_SYS_PART2_DATA7_REG	Represents rd_sys_part2_data	0x0178	RO
Block 0 Error Report Registers			
EFUSE_RD_REPEAT_DATA_ERR0_REG	Represents rd_repeat_data_err	0x017C	RO
EFUSE_RD_REPEAT_DATA_ERR1_REG	Represents rd_repeat_data_err	0x0180	RO
EFUSE_RD_REPEAT_DATA_ERR2_REG	Represents rd_repeat_data_err	0x0184	RO
EFUSE_RD_REPEAT_DATA_ERR3_REG	Represents rd_repeat_data_err	0x0188	RO
EFUSE_RD_REPEAT_DATA_ERR4_REG	Represents rd_repeat_data_err	0x018C	RO
EFUSE ECDSA Configuration Register			
EFUSE_ECDSA_REG	eFuse ECDSA configuration register	0x01B0	varies
RS Block Error Report Registers			
EFUSE_RD_RS_DATA_ERR0_REG	Represents rd_rs_data_err	0x01C0	RO
EFUSE_RD_RS_DATA_ERR1_REG	Represents rd_rs_data_err	0x01C4	RO
EFUSE_CLK_REG	eFuse clock configuration register	0x01C8	R/W
EFUSE_CONF_REG	eFuse operation mode configuration register	0x01CC	R/W
EFUSE_STATUS_REG	eFuse status register	0x01D0	RO
EFUSE_CMD_REG	eFuse command register	0x01D4	varies
EFUSE_INT_RAW_REG	eFuse raw interrupt register	0x01D8	R/SS/WTC
EFUSE_INT_ST_REG	eFuse interrupt status register	0x01DC	RO
EFUSE_INT_ENA_REG	eFuse interrupt enable register	0x01E0	R/W

Name	Description	Address	Access
EFUSE_INT_CLR_REG	eFuse interrupt clear register	0x01E4	WT
EFUSE_DAC_CONF_REG	Controls the eFuse programming voltage	0x01E8	R/W
EFUSE_RD_TIM_CONF_REG	Configures read timing parameters	0x01EC	R/W
EFUSE_WR_TIM_CONF1_REG	Configuration register 1 of eFuse programming timing parameters	0x01F0	R/W
EFUSE_WR_TIM_CONF2_REG	Configuration register 2 of eFuse programming timing parameters	0x01F4	R/W
EFUSE_WR_TIM_CONF0_RS_BYPASS_REG	Configuration register 0 of eFuse programming timing parameters and RS bypass operation	0x01F8	varies
EFUSE Version Register			
EFUSE_DATE_REG	eFuse version register	0x01FC	R/W
EFUSE_APB2OTP Block 0 Registers			
EFUSE_APB2OTP_WR_DIS_REG	eFuse APB2OTP Block 0 data register1	0x0800	RO
EFUSE_APB2OTP_BLK0_BACKUP1_W1_REG	eFuse APB2OTP Block 0 data register2	0x0804	RO
EFUSE_APB2OTP_BLK0_BACKUP1_W2_REG	eFuse APB2OTP Block 0 data register3	0x0808	RO
EFUSE_APB2OTP_BLK0_BACKUP1_W3_REG	eFuse APB2OTP Block 0 data register4	0x080C	RO
EFUSE_APB2OTP_BLK0_BACKUP1_W4_REG	eFuse APB2OTP Block 0 data register5	0x0810	RO
EFUSE_APB2OTP_BLK0_BACKUP1_W5_REG	eFuse APB2OTP Block 0 data register6	0x0814	RO
EFUSE_APB2OTP_BLK0_BACKUP2_W1_REG	eFuse APB2OTP Block 0 data register7	0x0818	RO
EFUSE_APB2OTP_BLK0_BACKUP2_W2_REG	eFuse APB2OTP Block 0 data register8	0x081C	RO
EFUSE_APB2OTP_BLK0_BACKUP2_W3_REG	eFuse APB2OTP Block 0 data register9	0x0820	RO
EFUSE_APB2OTP_BLK0_BACKUP2_W4_REG	eFuse APB2OTP Block 0 data register10	0x0824	RO
EFUSE_APB2OTP_BLK0_BACKUP2_W5_REG	eFuse APB2OTP Block 0 data register11	0x0828	RO
EFUSE_APB2OTP_BLK0_BACKUP3_W1_REG	eFuse APB2OTP Block 0 data register12	0x082C	RO
EFUSE_APB2OTP_BLK0_BACKUP3_W2_REG	eFuse APB2OTP Block 0 data register13	0x0830	RO
EFUSE_APB2OTP_BLK0_BACKUP3_W3_REG	eFuse APB2OTP Block 0 data register14	0x0834	RO
EFUSE_APB2OTP_BLK0_BACKUP3_W4_REG	eFuse APB2OTP Block 0 data register15	0x0838	RO
EFUSE_APB2OTP_BLK0_BACKUP3_W5_REG	eFuse APB2OTP Block 0 data register16	0x083C	RO
EFUSE_APB2OTP_BLK0_BACKUP4_W1_REG	eFuse APB2OTP Block 0 data register17	0x0840	RO
EFUSE_APB2OTP_BLK0_BACKUP4_W2_REG	eFuse APB2OTP Block 0 data register18	0x0844	RO
EFUSE_APB2OTP_BLK0_BACKUP4_W3_REG	eFuse APB2OTP Block 0 data register19	0x0848	RO
EFUSE_APB2OTP_BLK0_BACKUP4_W4_REG	eFuse APB2OTP Block 0 data register20	0x084C	RO
EFUSE_APB2OTP_BLK0_BACKUP4_W5_REG	eFuse APB2OTP Block 0 data register21	0x0850	RO
EFUSE_APB2OTP Block 1 Registers			
EFUSE_APB2OTP_BLK1_W1_REG	eFuse APB2OTP Block 1 data register1	0x0854	RO
EFUSE_APB2OTP_BLK1_W2_REG	eFuse APB2OTP Block 1 data register2	0x0858	RO
EFUSE_APB2OTP_BLK1_W3_REG	eFuse APB2OTP Block 1 data register3	0x085C	RO
EFUSE_APB2OTP_BLK1_W4_REG	eFuse APB2OTP Block 1 data register4	0x0860	RO
EFUSE_APB2OTP_BLK1_W5_REG	eFuse APB2OTP Block 1 data register5	0x0864	RO
EFUSE_APB2OTP_BLK1_W6_REG	eFuse APB2OTP Block 1 data register6	0x0868	RO

Name	Description	Address	Access
EFUSE_APB2OTP_BLK1_W7_REG	eFuse APB2OTP Block 1 data register7	0x086C	RO
EFUSE_APB2OTP_BLK1_W8_REG	eFuse APB2OTP Block 1 data register8	0x0870	RO
EFUSE_APB2OTP_BLK1_W9_REG	eFuse APB2OTP Block 1 data register9	0x0874	RO
EFUSE_APB2OTP Block 2 Registers			
EFUSE_APB2OTP_BLK2_W1_REG	eFuse APB2OTP Block 2 data register1	0x0878	RO
EFUSE_APB2OTP_BLK2_W2_REG	eFuse APB2OTP Block 2 data register2	0x087C	RO
EFUSE_APB2OTP_BLK2_W3_REG	eFuse APB2OTP Block 2 data register3	0x0880	RO
EFUSE_APB2OTP_BLK2_W4_REG	eFuse APB2OTP Block 2 data register4	0x0884	RO
EFUSE_APB2OTP_BLK2_W5_REG	eFuse APB2OTP Block 2 data register5	0x0888	RO
EFUSE_APB2OTP_BLK2_W6_REG	eFuse APB2OTP Block 2 data register6	0x088C	RO
EFUSE_APB2OTP_BLK2_W7_REG	eFuse APB2OTP Block 2 data register7	0x0890	RO
EFUSE_APB2OTP_BLK2_W8_REG	eFuse APB2OTP Block 2 data register8	0x0894	RO
EFUSE_APB2OTP_BLK2_W9_REG	eFuse APB2OTP Block 2 data register9	0x0898	RO
EFUSE_APB2OTP_BLK2_W10_REG	eFuse APB2OTP Block 2 data register10	0x089C	RO
EFUSE_APB2OTP_BLK2_W11_REG	eFuse APB2OTP Block 2 data register11	0x08A0	RO
EFUSE_APB2OTP Block 3 Registers			
EFUSE_APB2OTP_BLK3_W1_REG	eFuse APB2OTP Block 3 data register1	0x08A4	RO
EFUSE_APB2OTP_BLK3_W2_REG	eFuse APB2OTP Block 3 data register2	0x08A8	RO
EFUSE_APB2OTP_BLK3_W3_REG	eFuse APB2OTP Block 3 data register3	0x08AC	RO
EFUSE_APB2OTP_BLK3_W4_REG	eFuse APB2OTP Block 3 data register4	0x08B0	RO
EFUSE_APB2OTP_BLK3_W5_REG	eFuse APB2OTP Block 3 data register5	0x08B4	RO
EFUSE_APB2OTP_BLK3_W6_REG	eFuse APB2OTP Block 3 data register6	0x08B8	RO
EFUSE_APB2OTP_BLK3_W7_REG	eFuse APB2OTP Block 3 data register7	0x08BC	RO
EFUSE_APB2OTP_BLK3_W8_REG	eFuse APB2OTP Block 3 data register8	0x08C0	RO
EFUSE_APB2OTP_BLK3_W9_REG	eFuse APB2OTP Block 3 data register9	0x08C4	RO
EFUSE_APB2OTP_BLK3_W10_REG	eFuse APB2OTP Block 3 data register10	0x08C8	RO
EFUSE_APB2OTP_BLK3_W11_REG	eFuse APB2OTP Block 3 data register11	0x08CC	RO
EFUSE_APB2OTP Block 4 Registers			
EFUSE_APB2OTP_BLK4_W1_REG	eFuse APB2OTP Block 4 data register1	0x08D0	RO
EFUSE_APB2OTP_BLK4_W2_REG	eFuse APB2OTP Block 4 data register2	0x08D4	RO
EFUSE_APB2OTP_BLK4_W3_REG	eFuse APB2OTP Block 4 data register3	0x08D8	RO
EFUSE_APB2OTP_BLK4_W4_REG	eFuse APB2OTP Block 4 data register4	0x08DC	RO
EFUSE_APB2OTP_BLK4_W5_REG	eFuse APB2OTP Block 4 data register5	0x08E0	RO
EFUSE_APB2OTP_BLK4_W6_REG	eFuse APB2OTP Block 4 data register6	0x08E4	RO
EFUSE_APB2OTP_BLK4_W7_REG	eFuse APB2OTP Block 4 data register7	0x08E8	RO
EFUSE_APB2OTP_BLK4_W8_REG	eFuse APB2OTP Block 4 data register8	0x08EC	RO
EFUSE_APB2OTP_BLK4_W9_REG	eFuse APB2OTP Block 4 data register9	0x08F0	RO
EFUSE_APB2OTP_BLK4_W10_REG	eFuse APB2OTP Block 4 data register10	0x08F4	RO
EFUSE_APB2OTP_BLK4_W11_REG	eFuse APB2OTP Block 4 data register11	0x08F8	RO
EFUSE_APB2OTP Block 5 Registers			
EFUSE_APB2OTP_BLK5_W1_REG	eFuse APB2OTP Block 5 data register1	0x08FC	RO
EFUSE_APB2OTP_BLK5_W2_REG	eFuse APB2OTP Block 5 data register2	0x0900	RO
EFUSE_APB2OTP_BLK5_W3_REG	eFuse APB2OTP Block 5 data register3	0x0904	RO

Name	Description	Address	Access
EFUSE_APB2OTP_BLK5_W4_REG	eFuse APB2OTP Block 5 data register4	0x0908	RO
EFUSE_APB2OTP_BLK5_W5_REG	eFuse APB2OTP Block 5 data register5	0x090C	RO
EFUSE_APB2OTP_BLK5_W6_REG	eFuse APB2OTP Block 5 data register6	0x0910	RO
EFUSE_APB2OTP_BLK5_W7_REG	eFuse APB2OTP Block 5 data register7	0x0914	RO
EFUSE_APB2OTP_BLK5_W8_REG	eFuse APB2OTP Block 5 data register8	0x0918	RO
EFUSE_APB2OTP_BLK5_W9_REG	eFuse APB2OTP Block 5 data register9	0x091C	RO
EFUSE_APB2OTP_BLK5_W10_REG	eFuse APB2OTP Block 5 data register10	0x0920	RO
EFUSE_APB2OTP_BLK5_W11_REG	eFuse APB2OTP Block 5 data register11	0x0924	RO
EFUSE_APB2OTP Block 6 Registers			
EFUSE_APB2OTP_BLK6_W1_REG	eFuse APB2OTP Block 6 data register1	0x0928	RO
EFUSE_APB2OTP_BLK6_W2_REG	eFuse APB2OTP Block 6 data register2	0x092C	RO
EFUSE_APB2OTP_BLK6_W3_REG	eFuse APB2OTP Block 6 data register3	0x0930	RO
EFUSE_APB2OTP_BLK6_W4_REG	eFuse APB2OTP Block 6 data register4	0x0934	RO
EFUSE_APB2OTP_BLK6_W5_REG	eFuse APB2OTP Block 6 data register5	0x0938	RO
EFUSE_APB2OTP_BLK6_W6_REG	eFuse APB2OTP Block 6 data register6	0x093C	RO
EFUSE_APB2OTP_BLK6_W7_REG	eFuse APB2OTP Block 6 data register7	0x0940	RO
EFUSE_APB2OTP_BLK6_W8_REG	eFuse APB2OTP Block 6 data register8	0x0944	RO
EFUSE_APB2OTP_BLK6_W9_REG	eFuse APB2OTP Block 6 data register9	0x0948	RO
EFUSE_APB2OTP_BLK6_W10_REG	eFuse APB2OTP Block 6 data register10	0x094C	RO
EFUSE_APB2OTP_BLK6_W11_REG	eFuse APB2OTP Block 6 data register11	0x0950	RO
EFUSE_APB2OTP Block 7 Registers			
EFUSE_APB2OTP_BLK7_W1_REG	eFuse APB2OTP Block 7 data register1	0x0954	RO
EFUSE_APB2OTP_BLK7_W2_REG	eFuse APB2OTP Block 7 data register2	0x0958	RO
EFUSE_APB2OTP_BLK7_W3_REG	eFuse APB2OTP Block 7 data register3	0x095C	RO
EFUSE_APB2OTP_BLK7_W4_REG	eFuse APB2OTP Block 7 data register4	0x0960	RO
EFUSE_APB2OTP_BLK7_W5_REG	eFuse APB2OTP Block 7 data register5	0x0964	RO
EFUSE_APB2OTP_BLK7_W6_REG	eFuse APB2OTP Block 7 data register6	0x0968	RO
EFUSE_APB2OTP_BLK7_W7_REG	eFuse APB2OTP Block 7 data register7	0x096C	RO
EFUSE_APB2OTP_BLK7_W8_REG	eFuse APB2OTP Block 7 data register8	0x0970	RO
EFUSE_APB2OTP_BLK7_W9_REG	eFuse APB2OTP Block 7 data register9	0x0974	RO
EFUSE_APB2OTP_BLK7_W10_REG	eFuse APB2OTP Block 7 data register10	0x0978	RO
EFUSE_APB2OTP_BLK7_W11_REG	eFuse APB2OTP Block 7 data register11	0x097C	RO
EFUSE_APB2OTP Block 8 Registers			
EFUSE_APB2OTP_BLK8_W1_REG	eFuse APB2OTP Block 8 data register1	0x0980	RO
EFUSE_APB2OTP_BLK8_W2_REG	eFuse APB2OTP Block 8 data register2	0x0984	RO
EFUSE_APB2OTP_BLK8_W3_REG	eFuse APB2OTP Block 8 data register3	0x0988	RO
EFUSE_APB2OTP_BLK8_W4_REG	eFuse APB2OTP Block 8 data register4	0x098C	RO
EFUSE_APB2OTP_BLK8_W5_REG	eFuse APB2OTP Block 8 data register5	0x0990	RO
EFUSE_APB2OTP_BLK8_W6_REG	eFuse APB2OTP Block 8 data register6	0x0994	RO
EFUSE_APB2OTP_BLK8_W7_REG	eFuse APB2OTP Block 8 data register7	0x0998	RO
EFUSE_APB2OTP_BLK8_W8_REG	eFuse APB2OTP Block 8 data register8	0x099C	RO
EFUSE_APB2OTP_BLK8_W9_REG	eFuse APB2OTP Block 8 data register9	0x09A0	RO
EFUSE_APB2OTP_BLK8_W10_REG	eFuse APB2OTP Block 8 data register10	0x09A4	RO

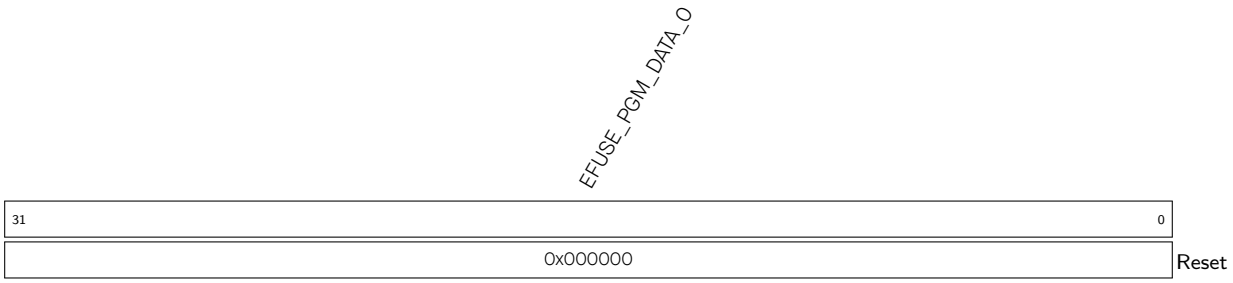
Name	Description	Address	Access
EFUSE_APB2OTP_BLK8_W11_REG	eFuse APB2OTP Block 8 data register11	0x09A8	RO
EFUSE_APB2OTP Block 9 Registers			
EFUSE_APB2OTP_BLK9_W1_REG	eFuse APB2OTP Block 9 data register1	0x09AC	RO
EFUSE_APB2OTP_BLK9_W2_REG	eFuse APB2OTP Block 9 data register2	0x09B0	RO
EFUSE_APB2OTP_BLK9_W3_REG	eFuse APB2OTP Block 9 data register3	0x09B4	RO
EFUSE_APB2OTP_BLK9_W4_REG	eFuse APB2OTP Block 9 data register4	0x09B8	RO
EFUSE_APB2OTP_BLK9_W5_REG	eFuse APB2OTP Block 9 data register5	0x09BC	RO
EFUSE_APB2OTP_BLK9_W6_REG	eFuse APB2OTP Block 9 data register6	0x09C0	RO
EFUSE_APB2OTP_BLK9_W7_REG	eFuse APB2OTP Block 9 data register7	0x09C4	RO
EFUSE_APB2OTP_BLK9_W8_REG	eFuse APB2OTP Block 9 data register8	0x09C8	RO
EFUSE_APB2OTP_BLK9_W9_REG	eFuse APB2OTP Block 9 data register9	0x09CC	RO
EFUSE_APB2OTP_BLK9_W10_REG	eFuse APB2OTP Block 9 data register10	0x09D0	RO
EFUSE_APB2OTP_BLK9_W11_REG	eFuse APB2OTP Block 9 data register11	0x09D4	RO
EFUSE_APB2OTP Block 10 Registers			
EFUSE_APB2OTP_BLK10_W1_REG	eFuse APB2OTP Block 10 data register1	0x09D8	RO
EFUSE_APB2OTP_BLK10_W2_REG	eFuse APB2OTP Block 10 data register2	0x09DC	RO
EFUSE_APB2OTP_BLK10_W3_REG	eFuse APB2OTP Block 10 data register3	0x09E0	RO
EFUSE_APB2OTP_BLK10_W4_REG	eFuse APB2OTP Block 10 data register4	0x09E4	RO
EFUSE_APB2OTP_BLK10_W5_REG	eFuse APB2OTP Block 10 data register5	0x09E8	RO
EFUSE_APB2OTP_BLK10_W6_REG	eFuse APB2OTP Block 10 data register6	0x09EC	RO
EFUSE_APB2OTP_BLK10_W7_REG	eFuse APB2OTP Block 10 data register7	0x09F0	RO
EFUSE_APB2OTP_BLK10_W8_REG	eFuse APB2OTP Block 10 data register8	0x09F4	RO
EFUSE_APB2OTP_BLK10_W9_REG	eFuse APB2OTP Block 10 data register9	0x09F8	RO
EFUSE_APB2OTP_BLK10_W10_REG	eFuse APB2OTP Block 10 data register10	0x09FC	RO
EFUSE_APB2OTP_BLK10_W11_REG	eFuse APB2OTP Block 10 data register11	0x0A00	RO
EFUSE_APB2OTP Function Enable Register			
EFUSE_APB2OTP_EN_REG	eFuse apb2otp enable configuration register	0x0A08	R/W

9.4.6 Registers

The addresses in this section are relative to eFuse controller base address provided in Table 9.3-2 in Chapter 9 *System and Memory*.

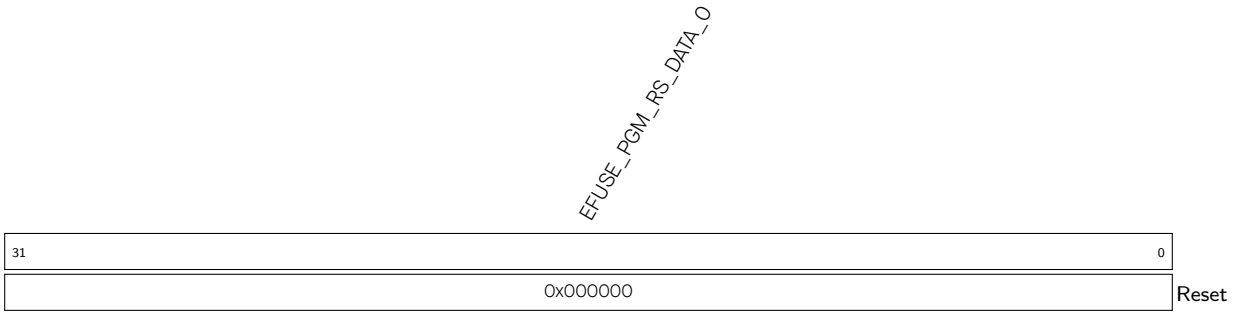
For how to program reserved fields, please refer to Section *Programming Reserved Register Field*.

Register 9.1. EFUSE_PGM_DATA*n*_REG (*n*: 0-7) (0x0000+0x4n*)**



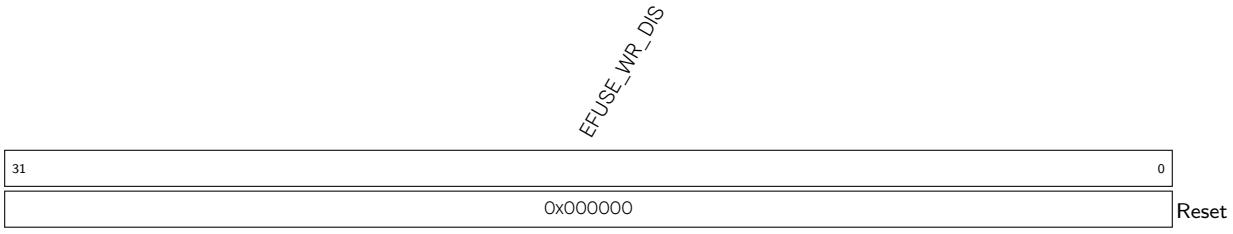
EFUSE_PGM_DATA_*n* Configures the *n*th 32-bit data to be programmed. (R/W)

Register 9.2. EFUSE_PGM_CHECK_VALUE*n*_REG (*n*: 0-2) (0x0020+0x4n*)**



EFUSE_PGM_RS_DATA_*n* Configures the *n*th RS code to be programmed. (R/W)

Register 9.3. EFUSE_RD_WR_DIS_REG (0x002C)

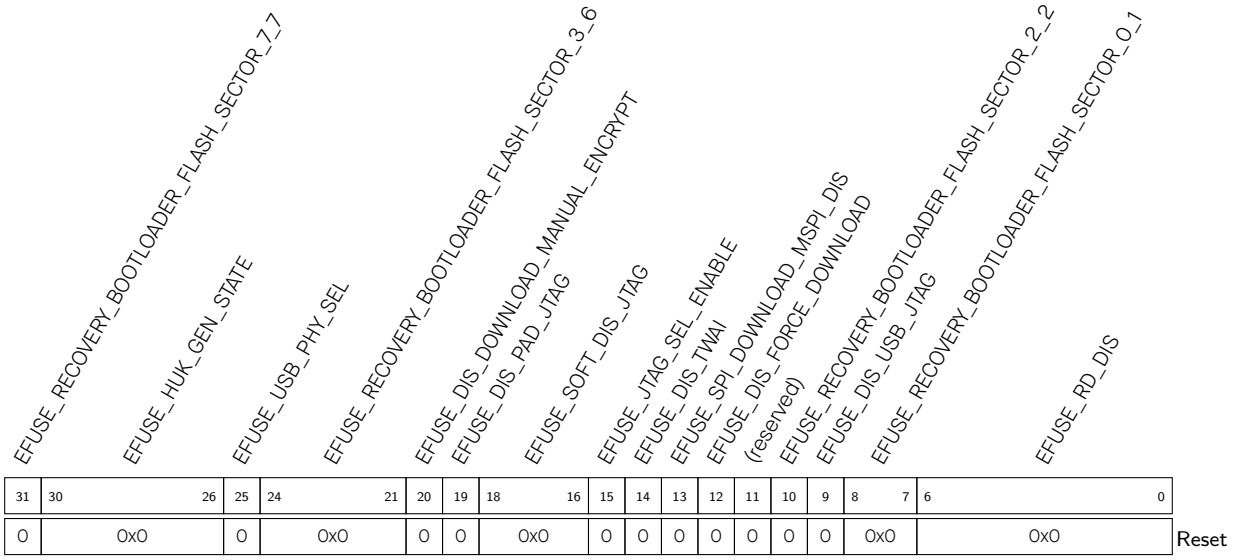


EFUSE_WR_DIS Represents whether programming of eFuse memory bits is disabled. For mapping between the bits of this field and the eFuse memory bits, please refer to Table 9.4-1 and Table 9.4-3. (RO)

1: Disabled

0: Enabled

Register 9.4. EFUSE_RD_REPEAT_DATA0_REG (0x0030)



EFUSE_RD_DIS Represents whether reading of individual eFuse block (block4 block10) is disabled or enabled. (RO)

1: disabled

0: enabled

EFUSE_RECOVERY_BOOTLOADER_FLASH_SECTOR_0_1 Represents the starting flash sector (flash sector size is 0x1000) of the recovery bootloader used by the ROM bootloader if the primary bootloader fails. (RO)

0 and 0xFFF: this feature is disabled

EFUSE_DIS_USB_JTAG Represents whether the USB to JTAG switch in the USB device module is disabled. (RO)

EFUSE_RECOVERY_BOOTLOADER_FLASH_SECTOR_2_2 Represents the starting flash sector (flash sector size is 0x1000) of the recovery bootloader used by the ROM bootloader if the primary bootloader fails. (RO)

0 and 0xFFF: this feature is disabled

EFUSE_DIS_FORCE_DOWNLOAD Represents whether the function that forces chip into download mode is disabled. (RO)

EFUSE_SPI_DOWNLOAD_MSPI_DIS Represents whether accessing MSPI flash/MSPI RAM by SYS AXI matrix during boot_mode_download is disabled. (RO)

EFUSE_DIS_TWAI Represents whether the TWAI function is disabled. (RO)

EFUSE_JTAG_SEL_ENABLE Represents whether selection between usb_to_jtag and pad_to_jtag through strapping GPIO25 is enabled when both EFUSE_DIS_PAD_JTAG and EFUSE_DIS_USB_JTAG are equal to 0. (RO)

Continued on the next page...

Register 9.4. EFUSE_RD_REPEAT_DATA0_REG (0x0030)

Continued from the previous page...

EFUSE_SOFT_DIS_JTAG Represents whether JTAG is disabled in the soft way or not. JTAG can be restarted in the HMAC module. (RO)
Odd count of bits with a value of 1: Disabled
Even count of bits with a value of 1: Enabled.

EFUSE_DIS_PAD_JTAG Represents whether JTAG is disabled in the hard way (permanently). (RO)

EFUSE_DIS_DOWNLOAD_MANUAL_ENCRYPT Represents whether the flash manual encrypt function is disabled (except in SPI boot mode). (RO)

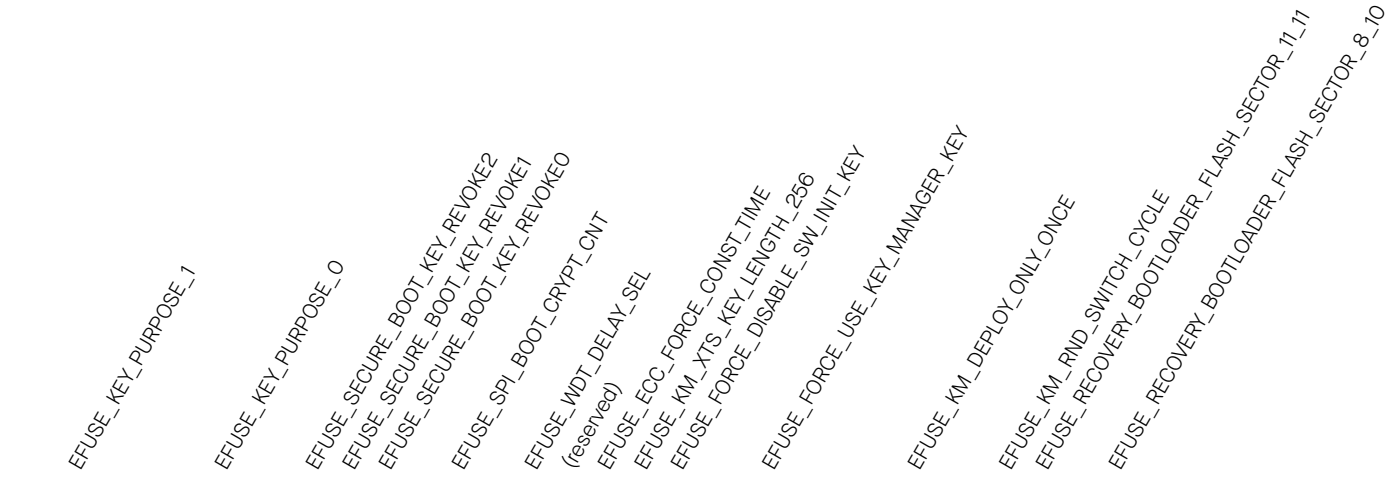
EFUSE_RECOVERY_BOOTLOADER_FLASH_SECTOR_3_6 Represents the starting flash sector (flash sector size is 0x1000) of the recovery bootloader used by the ROM bootloader if the primary bootloader fails. (RO)
0 and 0xFFF: this feature is disabled

EFUSE_USB_PHY_SEL Configures the corresponding relationship between the two PHYs and the USB interfaces. (RO)
0: The PHY on GPIO24/GPIO25 is connected to usb_device, while the PHY on GPIO26/GPIO27 is connected to usb_otg11
1: the PHY on GPIO26/GPIO27 is connected to usb_device, while the PHY on GPIO24/GPIO25 is connected to usb_otg11

EFUSE_HUK_GEN_STATE Represents the validation status of the HUK generation mode. (RO)
Odd count of bits with a value of 1: invalid
Even count of bits with a value of 1: valid

EFUSE_RECOVERY_BOOTLOADER_FLASH_SECTOR_7_7 Represents the starting flash sector of the recovery bootloader (flash sector size is 0x1000). When the primary bootloader fails, the ROM bootloader loads and executes the recovery bootloader from this sector. (RO)
0 and 0xFFF: The recovery bootloader feature is disabled

Register 9.5. EFUSE_RD_REPEAT_DATA1_REG (0x0034)



EFUSE_RECOVERY_BOOTLOADER_FLASH_SECTOR_8_10 Represents the starting flash sector of the recovery bootloader (flash sector size is 0x1000). When the primary bootloader fails, the ROM bootloader loads and executes the recovery bootloader from this sector. (RO)
0 and 0xFFF: The recovery bootloader feature is disabled

EFUSE_RECOVERY_BOOTLOADER_FLASH_SECTOR_11_11 Represents the starting flash sector of the recovery bootloader (flash sector size is 0x1000). When the primary bootloader fails, the ROM bootloader loads and executes the recovery bootloader from this sector. (RO)
0 and 0xFFF: The recovery bootloader feature is disabled

EFUSE_KM_RND_SWITCH_CYCLE Represents the key manager random number switch cycle. (RO)
0: control by register
1: 8 km clk cycles
2: 16 km cycles
3: 32 km cycles

EFUSE_KM_DEPLOY_ONLY_ONCE EFUSE_KM_DEPLOY_ONLY_ONCE and EFUSE_KM_DEPLOY_ONLY_ONCE_H together form one field: EFUSE_KM_DEPLOY_ONLY_ONCE_H, EFUSE_KM_DEPLOY_ONLY_ONCE[3:0]. Represents whether each corresponding key can only be deployed once. (RO)
1: true
0: false
bit 0: ecsda
bit 1: xts
bit 2: hmac
bit 3: ds
bit 4: psram

Continued on the next page...

Register 9.5. EFUSE_RD_REPEAT_DATA1_REG (0x0034)

Continued from the previous page...

EFUSE_FORCE_USE_KEY_MANAGER_KEY EFUSE_FORCE_USE_KEY_MANAGER_KEY and EFUSE_FORCE_USE_KEY_MANAGER_KEY_H together form one field: EFUSE_FORCE_USE_KEY_MANAGER_KEY_H, EFUSE_FORCE_USE_KEY_MANAGER_KEY[3:0]. Represents whether each corresponding key must come from key manager. (RO)
1: true
0: false
bit 0: ecsda
bit 1: xts
bit 2: hmac
bit 3: ds
bit 4: psram

EFUSE_FORCE_DISABLE_SW_INIT_KEY Represents whether software written init key is disabled, and efuse_init_key is forced to be used. (RO)

EFUSE_KM_XTS_KEY_LENGTH_256 Represents whether flash encryption uses XTS-512 key. Otherwise, the key manager will use the XTS-256 key. (RO)

EFUSE_ECC_FORCE_CONST_TIME Represents whether ECC const-time mode is permanently turned on. (RO)

EFUSE_WDT_DELAY_SEL Represents the LP WDT timeout threshold selection. The relationship is

$$\text{startup_timeout} = \text{initial_timeout_value} \times 2^{(\text{EFUSE_WDT_DELAY_SEL}+1)}$$

(RO)

EFUSE_SPI_BOOT_CRYPT_CNT Represents whether SPI boot encrypt/decrypt is enabled. (RO)
Odd number of 1: enabled
Even number of 1: disabled

EFUSE_SECURE_BOOT_KEY_REVOKE0 Represents whether revoking the first secure boot key is enabled. (RO)

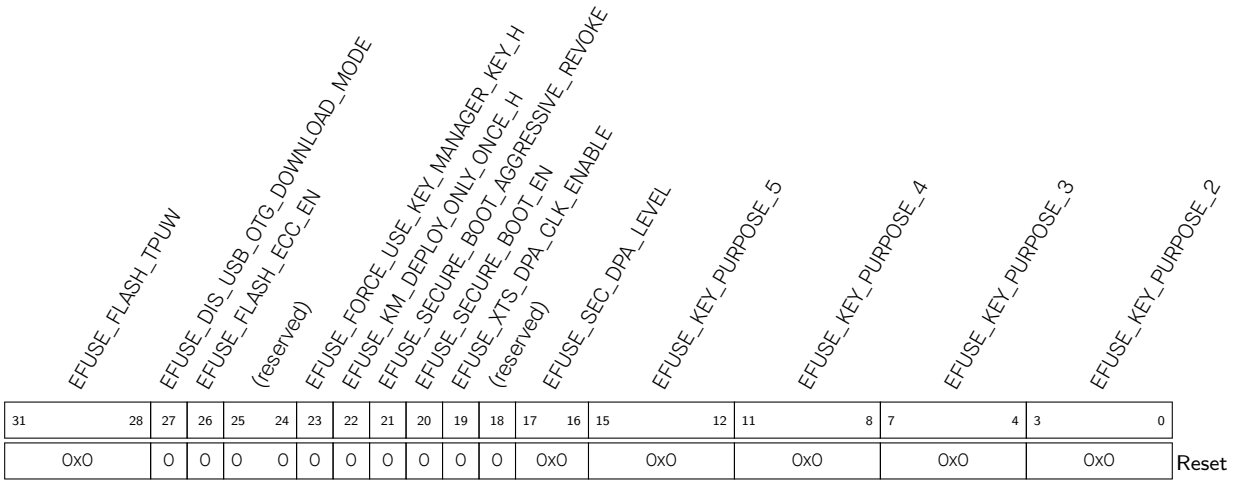
EFUSE_SECURE_BOOT_KEY_REVOKE1 Represents whether revoking the second secure boot key is enabled. (RO)

EFUSE_SECURE_BOOT_KEY_REVOKE2 Represents whether revoking the third secure boot key is enabled. (RO)

EFUSE_KEY_PURPOSE_0 Represents the purpose of Key0. For more information, see Table 9.4-2. (RO)

EFUSE_KEY_PURPOSE_1 Represents the purpose of Key1. For more information, see Table 9.4-2. (RO)

Register 9.6. EFUSE_RD_REPEAT_DATA2_REG (0x0038)



EFUSE_KEY_PURPOSE_2 Represents the purpose of Key2. For more information, see Table 9.4-2. (RO)

EFUSE_KEY_PURPOSE_3 Represents the purpose of Key3. For more information, see Table 9.4-2. (RO)

EFUSE_KEY_PURPOSE_4 Represents the purpose of Key4. For more information, see Table 9.4-2. (RO)

EFUSE_KEY_PURPOSE_5 Represents the purpose of Key5. For more information, see Table 9.4-2. (RO)

EFUSE_SEC_DPA_LEVEL Represents the DPA security level. This field effectively sets the protection level by configuring the clock random divide mode. (RO)

EFUSE_XTS_DPA_CLK_ENABLE Represents whether the XTS clock anti-DPA attack function is enabled. (RO)

EFUSE_SECURE_BOOT_EN Represents whether secure boot is enabled. (RO)

EFUSE_SECURE_BOOT_AGGRESSIVE_REVOKE Represents whether revoking aggressive secure boot is enabled. (RO)

EFUSE_KM_DEPLOY_ONLY_ONCE_H EFUSE_KM_DEPLOY_ONLY_ONCE and EFUSE_KM_DEPLOY_ONLY_ONCE_H together form one field: EFUSE_KM_DEPLOY_ONLY_ONCE_H & EFUSE_KM_DEPLOY_ONLY_ONCE[3:0]. Represents whether each corresponding key can only be deployed once. (RO)

1: true

0: false

bit 0: ecsda

bit 1: xts

bit 2: hmac

bit 3: ds

bit 4: psram

Continued on the next page...

Register 9.6. EFUSE_RD_REPEAT_DATA2_REG (0x0038)

Continued from the previous page...

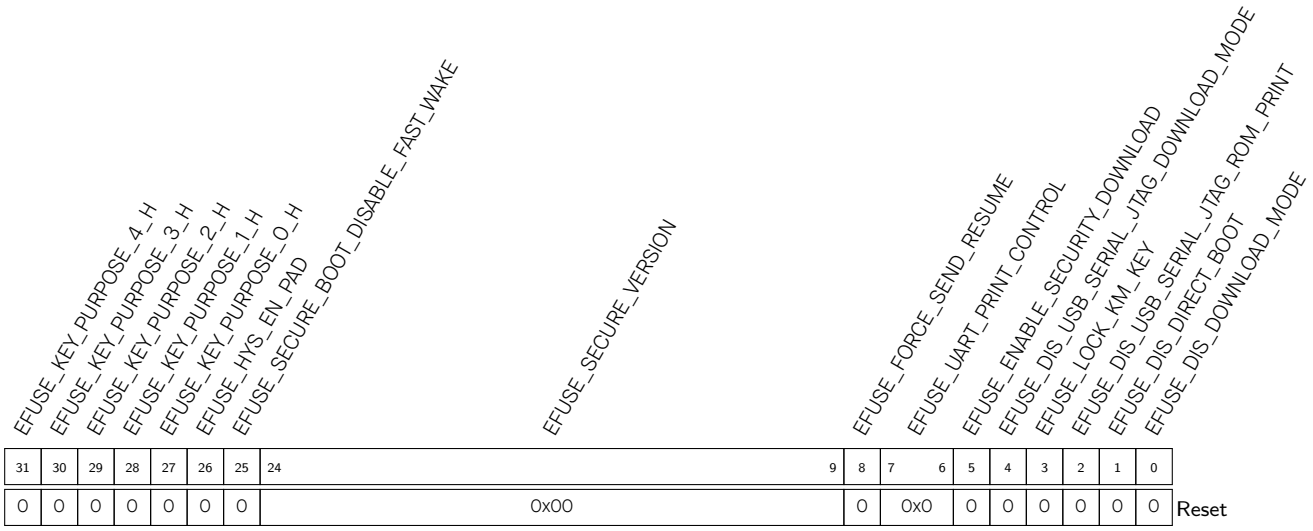
EFUSE_FORCE_USE_KEY_MANAGER_KEY_H EFUSE_FORCE_USE_KEY_MANAGER_KEY and EFUSE_FORCE_USE_KEY_MANAGER_KEY_H together form one field: EFUSE_FORCE_USE_KEY_MANAGER_KEY_H & EFUSE_FORCE_USE_KEY_MANAGER_KEY[3:0]. Represents whether each corresponding key must come from key manager. (RO)
1: true
0: false
bit 0: ecsda
bit 1: xts
bit 2: hmac
bit 3: ds
bit 4: psram

EFUSE_FLASH_ECC_EN Represents whether ECC for flash boot is enabled. (RO)

EFUSE_DIS_USB_OTG_DOWNLOAD_MODE Represents whether download via USB-OTG is disabled. (RO)

EFUSE_FLASH_TPUW Represents the flash waiting time after power-up, in unit of ms. (RO)
When the value is less than 15, the waiting time is the configurable value.
Otherwise, the waiting time is 30.

Register 9.7. EFUSE_RD_REPEAT_DATA3_REG (0x003C)



EFUSE_DIS_DOWNLOAD_MODE Represents whether all Download modes are disabled. (RO)

EFUSE_DIS_DIRECT_BOOT Represents whether direct boot mode is disabled. (RO)

EFUSE_DIS_USB_SERIAL_JTAG_ROM_PRINT Represents whether USB-Serial-JTAG print during ROM boot is disabled. (RO)

EFUSE_LOCK_KM_KEY Represents whether the key manager key is locked after deploy. (RO)

EFUSE_DIS_USB_SERIAL_JTAG_DOWNLOAD_MODE Represents whether the USB-Serial-JTAG download function is disabled. (RO)

EFUSE_ENABLE_SECURITY_DOWNLOAD Represents whether security download mode is enabled. (RO)

EFUSE_UART_PRINT_CONTROL Represents the type of UART printing. (RO)

00: force enable printing

01: enable printing when GPIO8 is reset at low level

10: enable printing when GPIO8 is reset at high level

11: force disable printing

EFUSE_FORCE_SEND_RESUME Represents whether ROM code is forced to send a resume command during SPI boot. (RO)

EFUSE_SECURE_VERSION Represents the secure version used by the ESP-IDF anti-rollback feature. (RO)

EFUSE_SECURE_BOOT_DISABLE_FAST_WAKE When secure boot is enabled, represents whether FAST VERIFY ON WAKE is disabled. (RO)

1: disabled

0: enabled

Continued on the next page...

Register 9.7. EFUSE_RD_REPEAT_DATA3_REG (0x003C)

Continued from the previous page...

EFUSE_HYS_EN_PAD Represents whether hysteresis function of PAD0 27 is enabled. (RO)

EFUSE_KEY_PURPOSE_0_H Represents the purpose of Key0. The 5th bit. (RO)

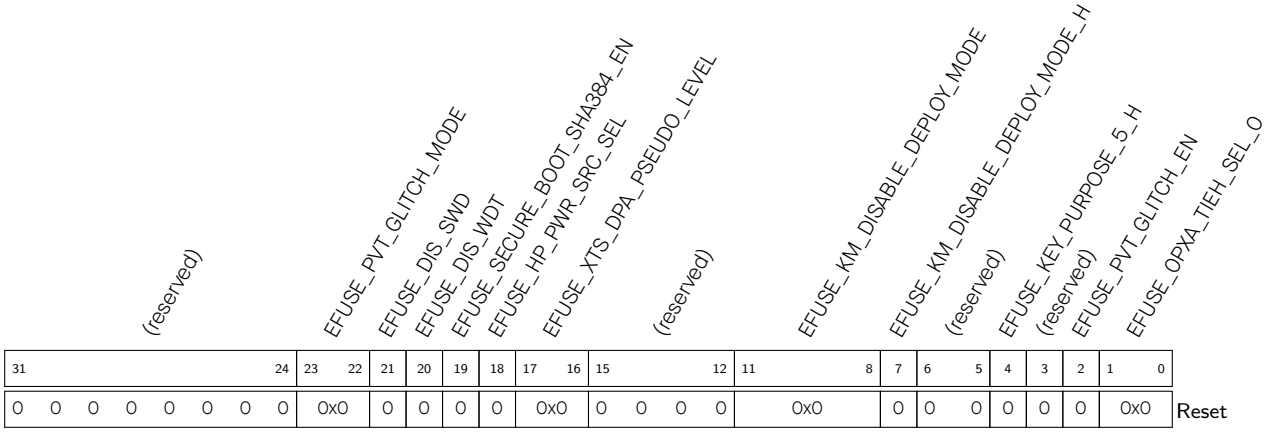
EFUSE_KEY_PURPOSE_1_H Represents the purpose of Key1. The 5th bit. (RO)

EFUSE_KEY_PURPOSE_2_H Represents the purpose of Key2. The 5th bit. (RO)

EFUSE_KEY_PURPOSE_3_H Represents the purpose of Key3. The 5th bit. (RO)

EFUSE_KEY_PURPOSE_4_H Represents the purpose of Key4. The 5th bit. (RO)

Register 9.8. EFUSE_RD_REPEAT_DATA4_REG (0x0040)



EFUSE_OPXA_TIEH_SEL_0 Represents the control selection for the LDO VO0 pull-high (tieh) source. (RO)

0: tied high

1: SDMMC1 peripheral

2: register field

3: SDMMC0 peripheral

EFUSE_PVT_GLITCH_EN Represents whether PVT power glitch monitor function is enabled. (RO)

1: Enable

0: Disable

EFUSE_KEY_PURPOSE_5_H Represents the purpose of Key5. The 5th bit. (RO)

EFUSE_KM_DISABLE_DEPLOY_MODE_H EFUSE_KM_DISABLE_DEPLOY_MODE and EFUSE_KM_-DISABLE_DEPLOY_MODE_H together form one field: EFUSE_KM_DISABLE_DEPLOY_MODE_H & EFUSE_KM_DISABLE_DEPLOY_MODE[3:0]. Represents whether each corresponding key's deploy mode of new value deployment is disabled. (RO)

1: true

0: false

bit 0: ecsda

bit 1: xts

bit 2: hmac

bit 3: ds

bit 4: psram

Continued on the next page...

Register 9.8. EFUSE_RD_REPEAT_DATA4_REG (0x0040)

Continued from the previous page...

EFUSE_KM_DISABLE_DEPLOY_MODE EFUSE_KM_DISABLE_DEPLOY_MODE and EFUSE_KM_-DISABLE_DEPLOY_MODE_H together form one field: EFUSE_KM_DISABLE_DEPLOY_MODE_H & EFUSE_KM_DISABLE_DEPLOY_MODE[3:0]. Represents whether each corresponding key's deploy mode of new value deployment is disabled. (RO)

1: true

0: false

bit 0: ecsda

bit 1: xts

bit 2: hmac

bit 3: ds

bit 4: psram

EFUSE_XTS_DPA_PSEUDO_LEVEL Represents the XTS pseudo-round anti-DPA attack function control. (RO)

0: controlled by register

1-3: the higher the value is, the more pseudo-rounds are inserted to the XTS-AES calculation

EFUSE_HP_PWR_SRC_SEL Represents the HP system power source selection. (RO)

0: LDO

1: DCDC

EFUSE_SECURE_BOOT_SHA384_EN Represents whether secure boot using SHA-384 is enabled. (RO)

0: disable

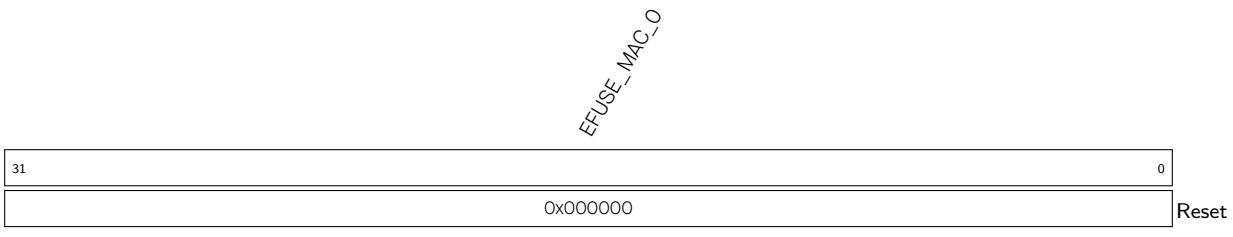
1: enable

EFUSE_DIS_WDT Represents whether the watchdog is disabled. (RO)

EFUSE_DIS_SWD Represents whether the super-watchdog is disabled. (RO)

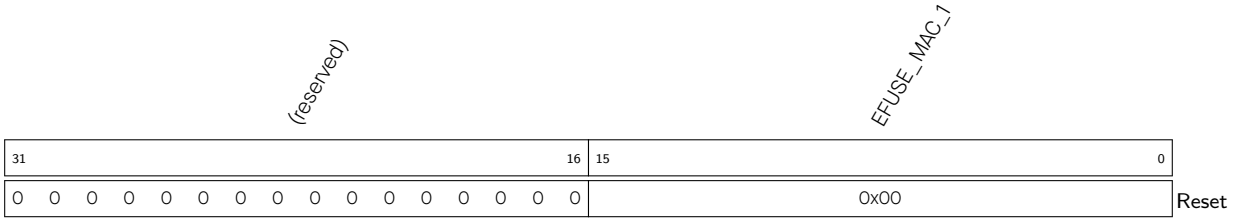
EFUSE_PVT_GLITCH_MODE Configures PVT glitch handling mode. (RO)

Register 9.9. EFUSE_RD_MAC_SYS0_REG (0x0044)



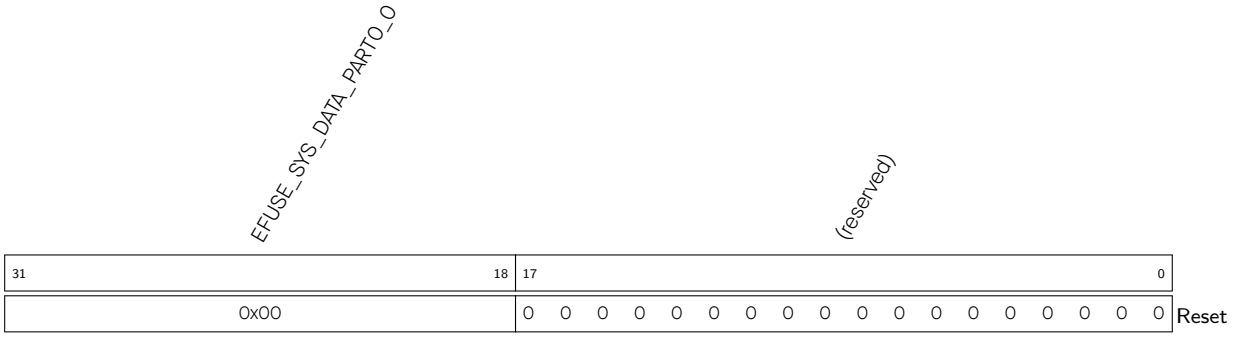
EFUSE_MAC_0 Represents the MAC address (lower 32 bits). (RO)

Register 9.10. EFUSE_RD_MAC_SYS1_REG (0x0048)



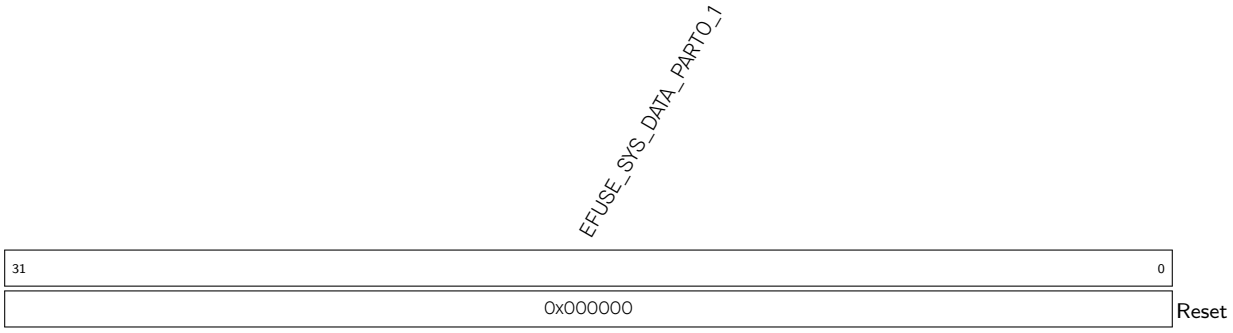
EFUSE_MAC_1 Represents the MAC address (upper 16 bits). (RO)

Register 9.11. EFUSE_RD_MAC_SYS3_REG (0x0050)



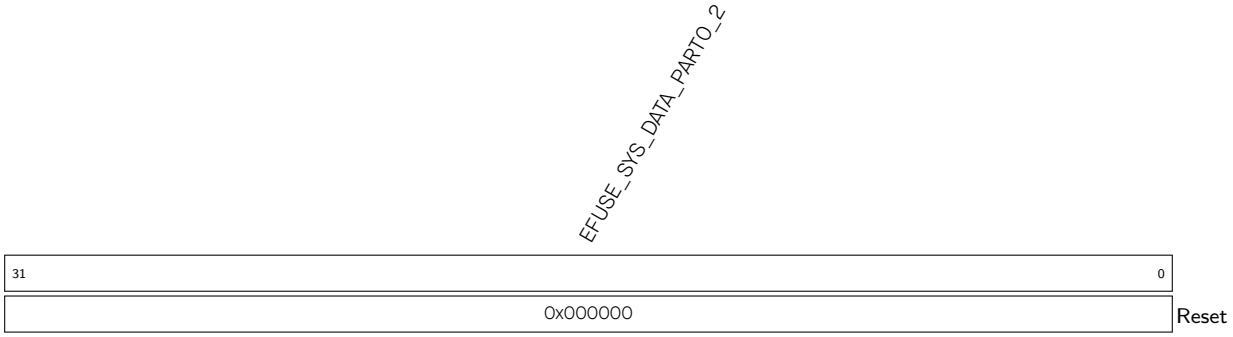
EFUSE_SYS_DATA_PART0_0 Represents the first 14 bits of Part 0 of system data. (RO)

Register 9.12. EFUSE_RD_MAC_SYS4_REG (0x0054)



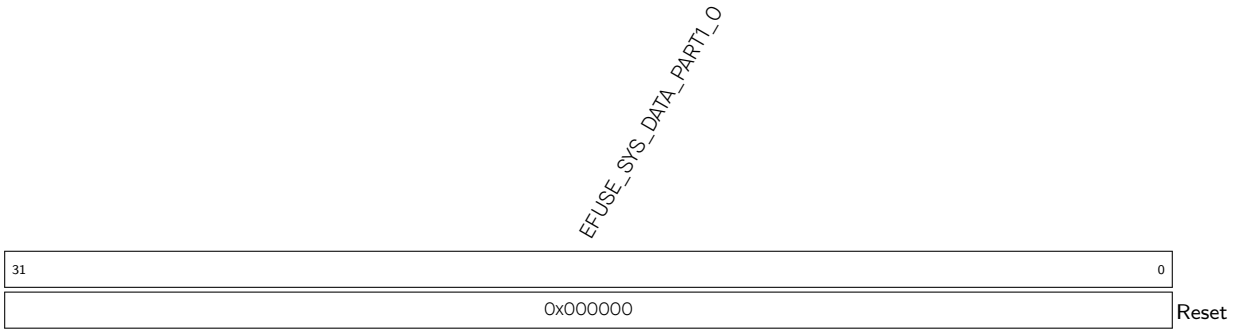
EFUSE_SYS_DATA_PART0_1 Represents the first 32-bit field of Part 0 of system data. (RO)

Register 9.13. EFUSE_RD_MAC_SYS5_REG (0x0058)



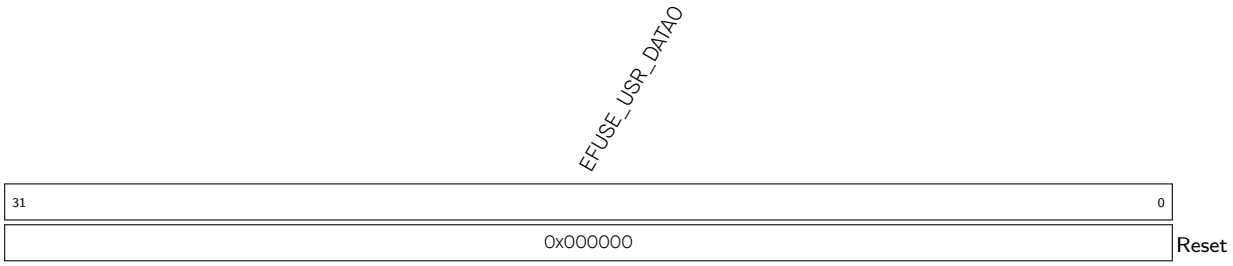
EFUSE_SYS_DATA_PART0_2 Represents the second 32-bit field of Part 0 of system data. (RO)

Register 9.14. EFUSE_RD_SYS_PART1_DATA*n*_REG (*n*: 0-7) (0x005C+0x4**n*)



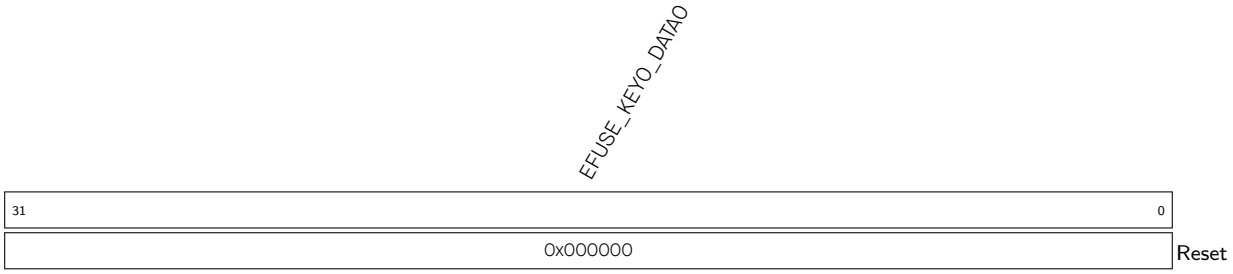
EFUSE_SYS_DATA_PART1_*n* Represents 32-bit word *n* of Part 1 of system data. (RO)

Register 9.15. EFUSE_RD_USR_DATA*n*_REG (*n*: 0-7) (0x007C+0x4**n*)



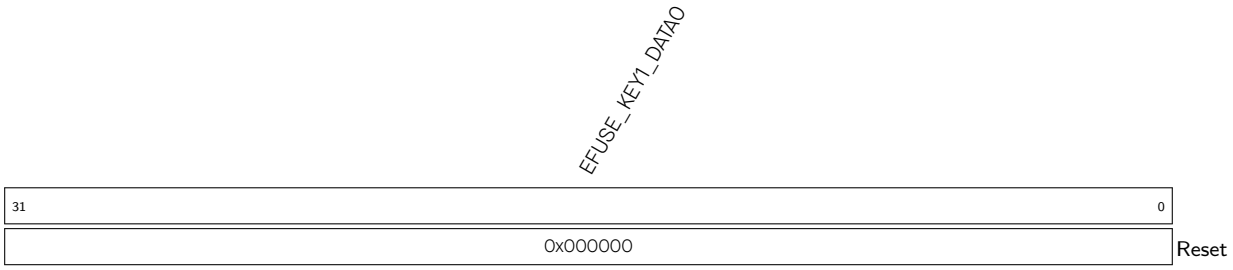
EFUSE_USR_DATA*n* Represents 32-bit word *n* of BLOCK3 (user data). (RO)

Register 9.16. EFUSE_RD_KEY0_DATA*n*_REG (*n*: 0-7) (0x009C+0x4**n*)



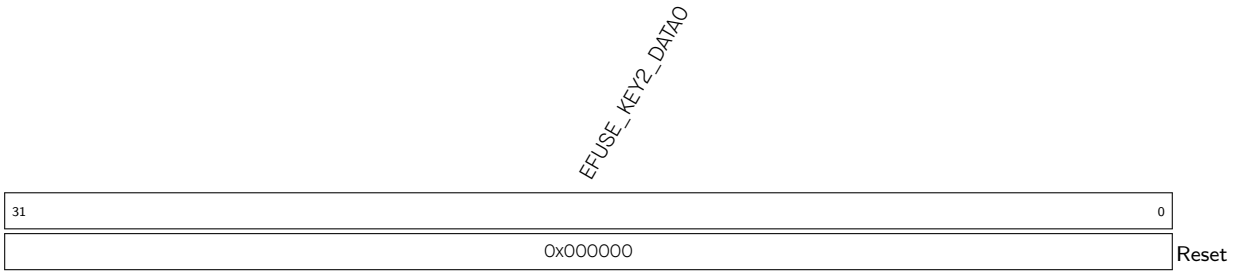
EFUSE_KEY0_DATA*n* Represents 32-bit word *n* of KEY0. (RO)

Register 9.17. EFUSE_RD_KEY1_DATA*n*_REG (*n*: 0-7) (0x00BC+0x4**n*)



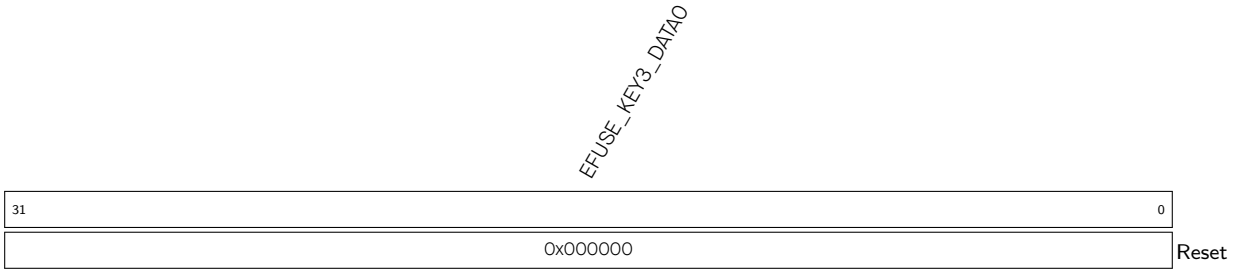
EFUSE_KEY1_DATA*n* Represents 32-bit word *n* of KEY1. (RO)

Register 9.18. EFUSE_RD_KEY2_DATA*n*_REG (*n*: 0-7) (0x00DC+0x4**n*)



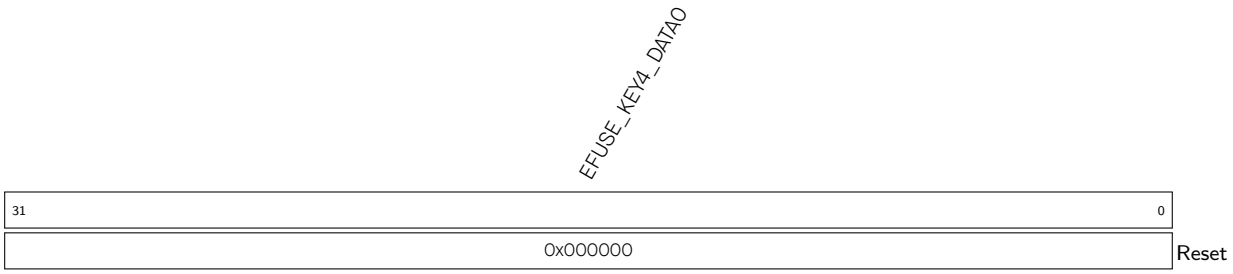
EFUSE_KEY2_DATA*n* Represents 32-bit word *n* of KEY2. (RO)

Register 9.19. EFUSE_RD_KEY3_DATA*n*_REG (*n*: 0-7) (0x00FC+0x4**n*)



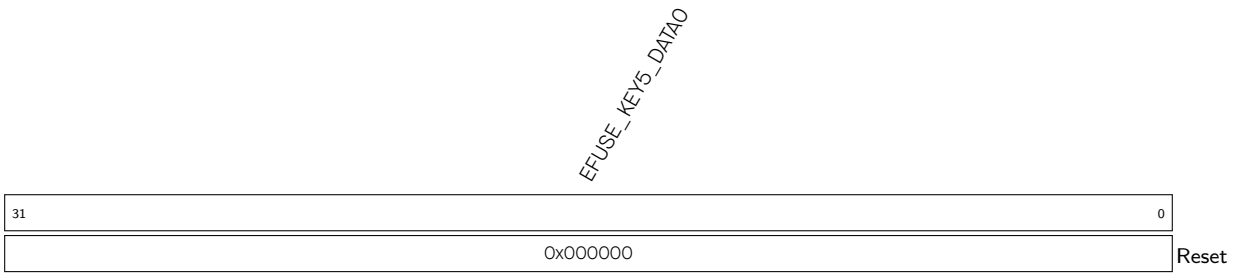
EFUSE_KEY3_DATA*n* Represents 32-bit word *n* of KEY3. (RO)

Register 9.20. EFUSE_RD_KEY4_DATA*n*_REG (*n*: 0-7) (0x011C+0x4**n*)



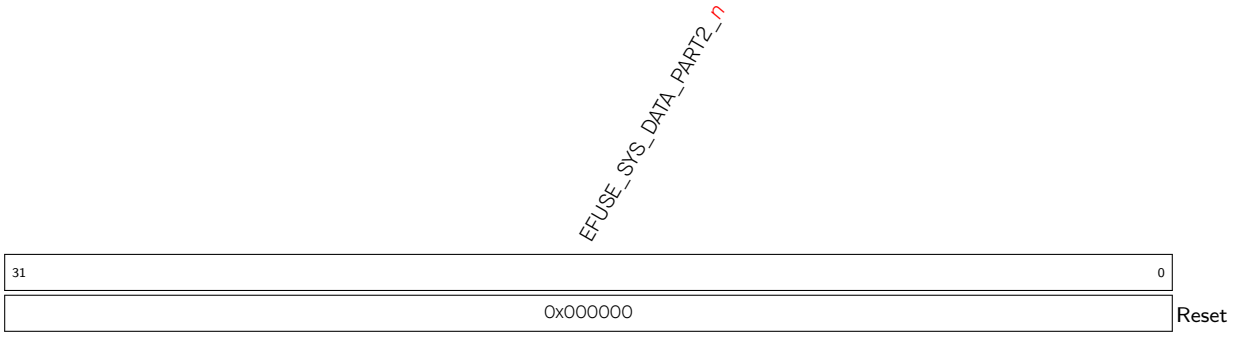
EFUSE_KEY4_DATA*n* Represents 32-bit word *n* of KEY4. (RO)

Register 9.21. EFUSE_RD_KEY5_DATA*n*_REG (*n*: 0-7) (0x013C+0x4**n*)



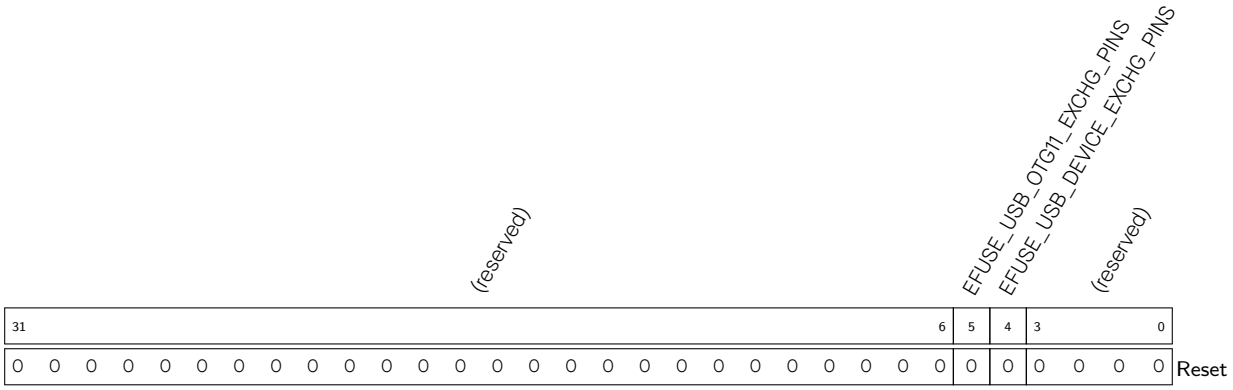
EFUSE_KEY5_DATA*n* Represents 32-bit word *n* of KEY5. (RO)

Register 9.22. EFUSE_RD_SYS_PART2_DATA*n*_REG (*n*: 0-5) (0x015C+0x4**n*)



EFUSE_SYS_DATA_PART2_*n* Represents 32-bit word *n* of the second part of system data. (RO)

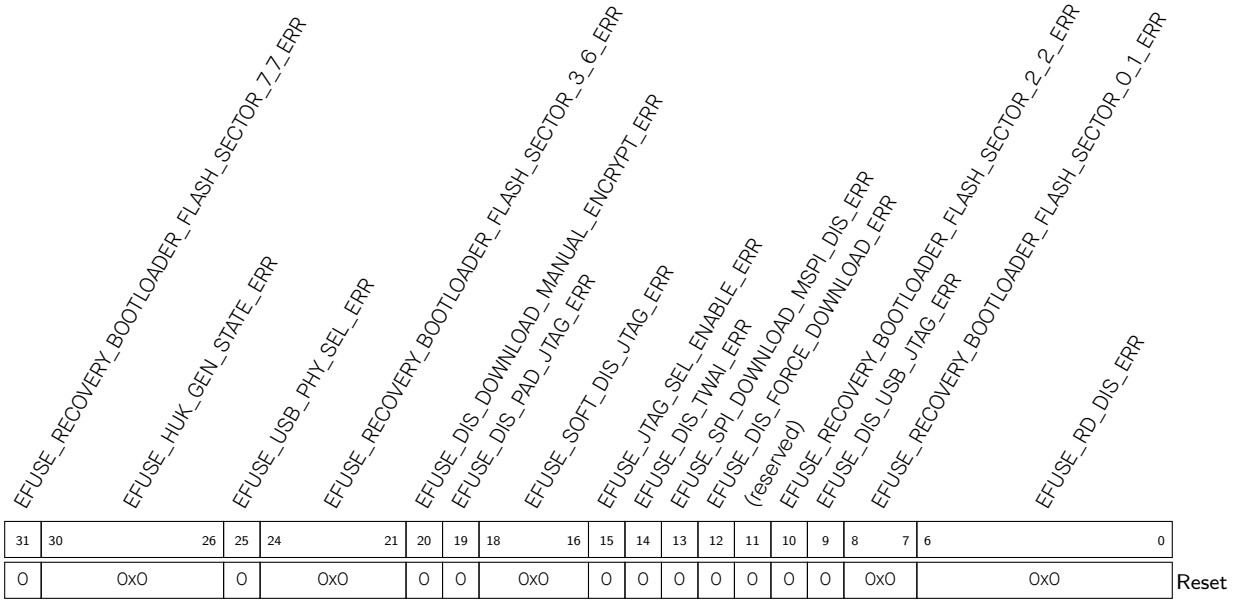
Register 9.23. EFUSE_RD_SYS_PART2_DATA7_REG (0x0178)



EFUSE_USB_DEVICE_EXCHG_PINS Represents whether USB device exchange pins of D+ and D- are enabled. (RO)

EFUSE_USB_OTG11_EXCHG_PINS Represents whether USB OTG11 exchange pins of D+ and D- are enabled. (RO)

Register 9.24. EFUSE_RD_REPEAT_DATA_ERR0_REG (0x017C)



EFUSE_RD_DIS_ERR Represents the programming error of EFUSE_RD_DIS (RO)

EFUSE_RECOVERY_BOOTLOADER_FLASH_SECTOR_0_1_ERR Represents the programming error of EFUSE_RECOVERY_BOOTLOADER_FLASH_SECTOR_0_1 (RO)

EFUSE_DIS_USB_JTAG_ERR Represents the programming error of EFUSE_DIS_USB_JTAG (RO)

EFUSE_RECOVERY_BOOTLOADER_FLASH_SECTOR_2_2_ERR Represents the programming error of EFUSE_RECOVERY_BOOTLOADER_FLASH_SECTOR_2_2 (RO)

EFUSE_DIS_FORCE_DOWNLOAD_ERR Represents the programming error of EFUSE_DIS_FORCE_-DOWNLOAD (RO)

EFUSE_SPI_DOWNLOAD_MSPI_DIS_ERR Represents the programming error of EFUSE_SPI_-DOWNLOAD_MSPI_DIS (RO)

EFUSE_DIS_TWAI_ERR Represents the programming error of EFUSE_DIS_TWAI (RO)

EFUSE_JTAG_SEL_ENABLE_ERR Represents the programming error of EFUSE_JTAG_SEL_-ENABLE (RO)

EFUSE_SOFT_DIS_JTAG_ERR Represents the programming error of EFUSE_SOFT_DIS_JTAG (RO)

EFUSE_DIS_PAD_JTAG_ERR Represents the programming error of EFUSE_DIS_PAD_JTAG (RO)

EFUSE_DIS_DOWNLOAD_MANUAL_ENCRYPT_ERR Represents the programming error of EFUSE_-DIS_DOWNLOAD_MANUAL_ENCRYPT (RO)

EFUSE_RECOVERY_BOOTLOADER_FLASH_SECTOR_3_6_ERR Represents the programming error of EFUSE_RECOVERY_BOOTLOADER_FLASH_SECTOR_3_6 (RO)

EFUSE_USB_PHY_SEL_ERR Represents the programming error of EFUSE_USB_PHY_SEL (RO)

Continued on the next page...

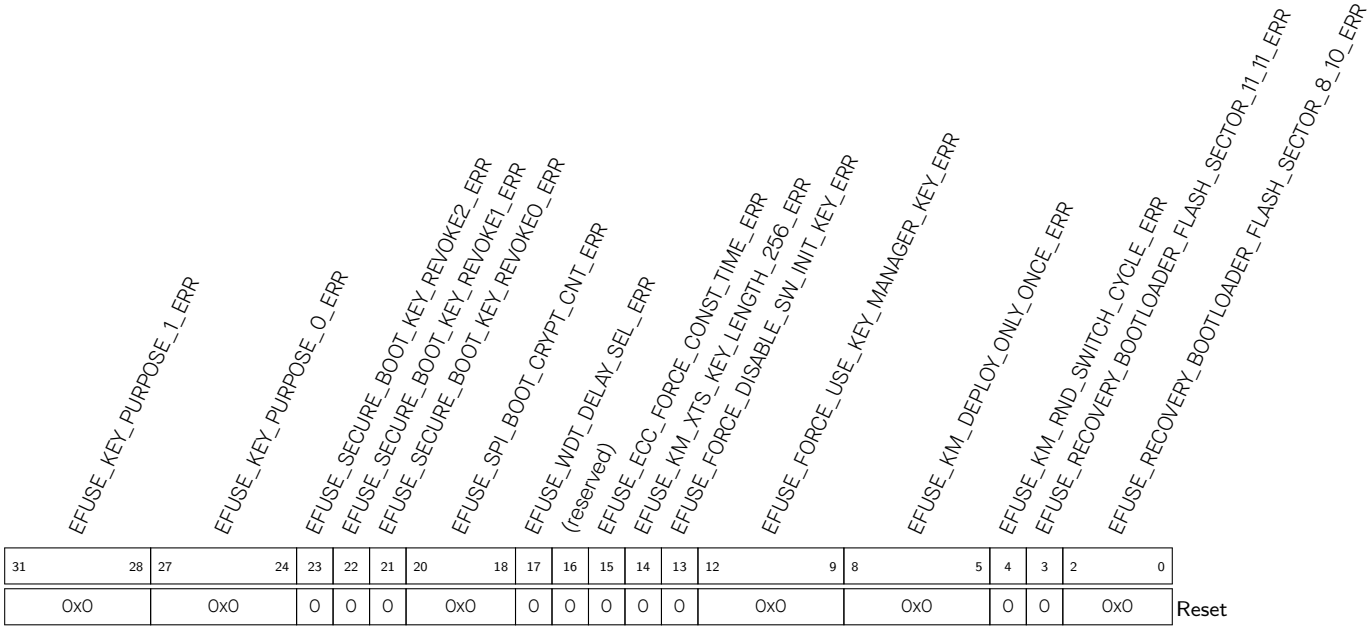
Register 9.24. EFUSE_RD_REPEAT_DATA_ERR0_REG (0x017C)

Continued from the previous page...

EFUSE_HUK_GEN_STATE_ERR Represents the programming error of EFUSE_HUK_GEN_STATE (RO)

EFUSE_RECOVERY_BOOTLOADER_FLASH_SECTOR_7_7_ERR Represents the programming error of EFUSE_RECOVERY_BOOTLOADER_FLASH_SECTOR_7_7 (RO)

Register 9.25. EFUSE_RD_REPEAT_DATA_ERR1_REG (0x0180)



EFUSE_RECOVERY_BOOTLOADER_FLASH_SECTOR_8_10_ERR Represents the programming error of EFUSE_RECOVERY_BOOTLOADER_FLASH_SECTOR_8_10 (RO)

EFUSE_RECOVERY_BOOTLOADER_FLASH_SECTOR_11_11_ERR Represents the programming error of EFUSE_RECOVERY_BOOTLOADER_FLASH_SECTOR_11_11 (RO)

EFUSE_KM_RND_SWITCH_CYCLE_ERR Represents the programming error of EFUSE_KM_RND_SWITCH_CYCLE (RO)

EFUSE_KM_DEPLOY_ONLY_ONCE_ERR Represents the programming error of EFUSE_KM_DEPLOY_ONLY_ONCE (RO)

EFUSE_FORCE_USE_KEY_MANAGER_KEY_ERR Represents the programming error of EFUSE_FORCE_USE_KEY_MANAGER_KEY (RO)

EFUSE_FORCE_DISABLE_SW_INIT_KEY_ERR Represents the programming error of EFUSE_FORCE_DISABLE_SW_INIT_KEY (RO)

EFUSE_KM_XTS_KEY_LENGTH_256_ERR Represents the programming error of EFUSE_KM_XTS_KEY_LENGTH_256 (RO)

EFUSE_ECC_FORCE_CONST_TIME_ERR Represents the programming error of EFUSE_ECC_FORCE_CONST_TIME (RO)

EFUSE_WDT_DELAY_SEL_ERR Represents the programming error of EFUSE_WDT_DELAY_SEL (RO)

EFUSE_SPI_BOOT_CRYPT_CNT_ERR Represents the programming error of EFUSE_SPI_BOOT_CRYPT_CNT (RO)

EFUSE_SECURE_BOOT_KEY_REVOKE0_ERR Represents the programming error of EFUSE_SECURE_BOOT_KEY_REVOKE0 (RO)

Continued on the next page...

Register 9.25. EFUSE_RD_REPEAT_DATA_ERR1_REG (0x0180)

Continued from the previous page...

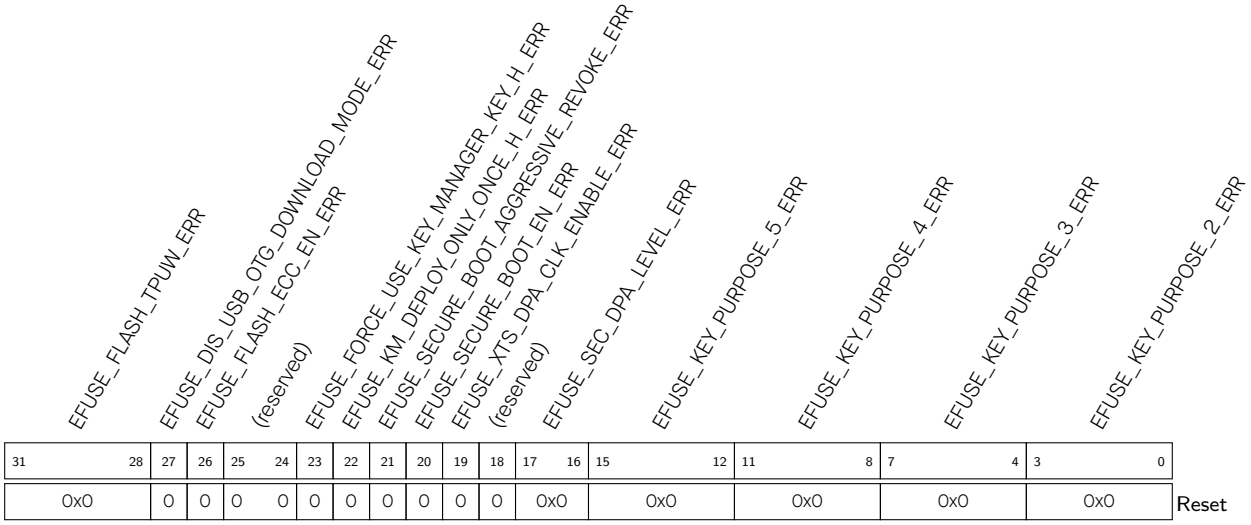
EFUSE_SECURE_BOOT_KEY_REVOKE1_ERR Represents the programming error of EFUSE_SECURE_BOOT_KEY_REVOKE1 (RO)

EFUSE_SECURE_BOOT_KEY_REVOKE2_ERR Represents the programming error of EFUSE_SECURE_BOOT_KEY_REVOKE2 (RO)

EFUSE_KEY_PURPOSE_0_ERR Represents the programming error of EFUSE_KEY_PURPOSE_0 (RO)

EFUSE_KEY_PURPOSE_1_ERR Represents the programming error of EFUSE_KEY_PURPOSE_1 (RO)

Register 9.26. EFUSE_RD_REPEAT_DATA_ERR2_REG (0x0184)



EFUSE_KEY_PURPOSE_2_ERR Represents the programming error of EFUSE_KEY_PURPOSE_2 (RO)

EFUSE_KEY_PURPOSE_3_ERR Represents the programming error of EFUSE_KEY_PURPOSE_3 (RO)

EFUSE_KEY_PURPOSE_4_ERR Represents the programming error of EFUSE_KEY_PURPOSE_4 (RO)

EFUSE_KEY_PURPOSE_5_ERR Represents the programming error of EFUSE_KEY_PURPOSE_5 (RO)

EFUSE_SEC_DPA_LEVEL_ERR Represents the programming error of EFUSE_SEC_DPA_LEVEL (RO)

EFUSE_XTS_DPA_CLK_ENABLE_ERR Represents the programming error of EFUSE_XTS_DPA_CLK_ENABLE (RO)

EFUSE_SECURE_BOOT_EN_ERR Represents the programming error of EFUSE_SECURE_BOOT_EN (RO)

EFUSE_SECURE_BOOT_AGGRESSIVE_REVOKE_ERR Represents the programming error of EFUSE_SECURE_BOOT_AGGRESSIVE_REVOKE (RO)

EFUSE_KM_DEPLOY_ONLY_ONCE_H_ERR Represents the programming error of EFUSE_KM_DEPLOY_ONLY_ONCE_H (RO)

EFUSE_FORCE_USE_KEY_MANAGER_KEY_H_ERR Represents the programming error of EFUSE_FORCE_USE_KEY_MANAGER_KEY_H (RO)

EFUSE_FLASH_ECC_EN_ERR Represents the programming error of EFUSE_FLASH_ECC_EN (RO)

EFUSE_DIS_USB_OTG_DOWNLOAD_MODE_ERR Represents the programming error of EFUSE_DIS_USB_OTG_DOWNLOAD_MODE (RO)

EFUSE_FLASH_TPUW_ERR Represents the programming error of EFUSE_FLASH_TPUW (RO)

Register 9.27. EFUSE_RD_REPEAT_DATA_ERR3_REG (0x0188)



EFUSE_DIS_DOWNLOAD_MODE_ERR Represents the programming error of EFUSE_DIS_DOWNLOAD_MODE (RO)

EFUSE_DIS_DIRECT_BOOT_ERR Represents the programming error of EFUSE_DIS_DIRECT_BOOT (RO)

EFUSE_DIS_USB_SERIAL_JTAG_ROM_PRINT_ERR Represents the programming error of EFUSE_DIS_USB_SERIAL_JTAG_ROM_PRINT (RO)

EFUSE_LOCK_KM_KEY_ERR Represents the programming error of EFUSE_LOCK_KM_KEY (RO)

EFUSE_DIS_USB_SERIAL_JTAG_DOWNLOAD_MODE_ERR Represents the programming error of EFUSE_DIS_USB_SERIAL_JTAG_DOWNLOAD_MODE (RO)

EFUSE_ENABLE_SECURITY_DOWNLOAD_ERR Represents the programming error of EFUSE_ENABLE_SECURITY_DOWNLOAD (RO)

EFUSE_UART_PRINT_CONTROL_ERR Represents the programming error of EFUSE_UART_PRINT_CONTROL (RO)

EFUSE_FORCE_SEND_RESUME_ERR Represents the programming error of EFUSE_FORCE_SEND_RESUME (RO)

EFUSE_SECURE_VERSION_ERR Represents the programming error of EFUSE_SECURE_VERSION (RO)

EFUSE_SECURE_BOOT_DISABLE_FAST_WAKE_ERR Represents the programming error of EFUSE_SECURE_BOOT_DISABLE_FAST_WAKE (RO)

EFUSE_HYS_EN_PAD_ERR Represents the programming error of EFUSE_HYS_EN_PAD (RO)

EFUSE_KEY_PURPOSE_0_H_ERR Represents the programming error of EFUSE_KEY_PURPOSE_0_H (RO)

Continued on the next page...

Register 9.27. EFUSE_RD_REPEAT_DATA_ERR3_REG (0x0188)

Continued from the previous page...

EFUSE_KEY_PURPOSE_1_H_ERR Represents the programming error of EFUSE_KEY_PURPOSE_1_H (RO)

EFUSE_KEY_PURPOSE_2_H_ERR Represents the programming error of EFUSE_KEY_PURPOSE_2_H (RO)

EFUSE_KEY_PURPOSE_3_H_ERR Represents the programming error of EFUSE_KEY_PURPOSE_3_H (RO)

EFUSE_KEY_PURPOSE_4_H_ERR Represents the programming error of EFUSE_KEY_PURPOSE_4_H (RO)

Register 9.28. EFUSE_RD_REPEAT_DATA_ERR4_REG (0x018C)



EFUSE_OPXA_TIEH_SEL_0_ERR Represents the programming error of EFUSE_OPXA_TIEH_SEL_0 (RO)

EFUSE_PVT_GLITCH_EN_ERR Represents the programming error of EFUSE_PVT_GLITCH_EN (RO)

EFUSE_KEY_PURPOSE_5_H_ERR Represents the programming error of EFUSE_KEY_PURPOSE_5_H (RO)

EFUSE_KM_DISABLE_DEPLOY_MODE_H_ERR Represents the programming error of EFUSE_KM_DISABLE_DEPLOY_MODE_H (RO)

EFUSE_KM_DISABLE_DEPLOY_MODE_ERR Represents the programming error of EFUSE_KM_DISABLE_DEPLOY_MODE (RO)

EFUSE_XTS_DPA_PSEUDO_LEVEL_ERR Represents the programming error of EFUSE_XTS_DPA_PSEUDO_LEVEL (RO)

EFUSE_HP_PWR_SRC_SEL_ERR Represents the programming error of EFUSE_HP_PWR_SRC_SEL (RO)

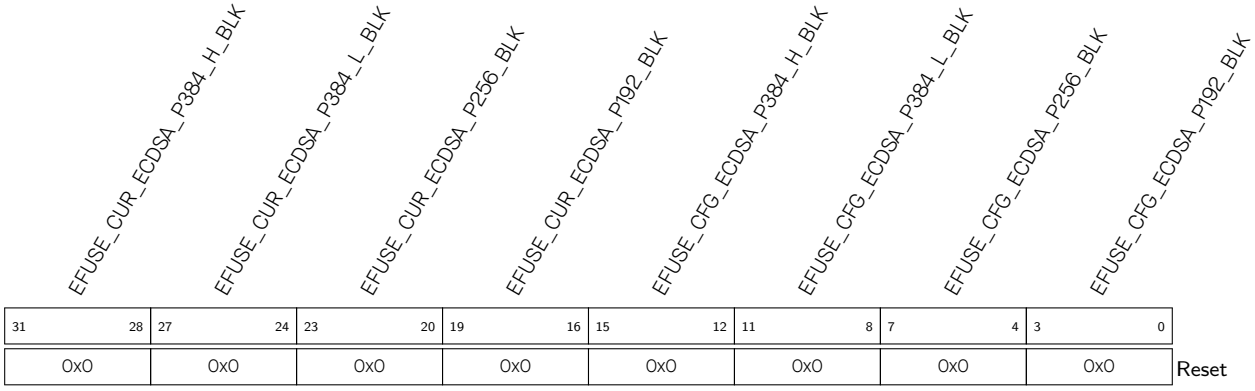
EFUSE_SECURE_BOOT_SHA384_EN_ERR Represents the programming error of EFUSE_SECURE_BOOT_SHA384_EN (RO)

EFUSE_DIS_WDT_ERR Represents the programming error of EFUSE_DIS_WDT (RO)

EFUSE_DIS_SWD_ERR Represents the programming error of EFUSE_DIS_SWD (RO)

EFUSE_PVT_GLITCH_MODE_ERR Represents the programming error of EFUSE_PVT_GLITCH_MODE (RO)

Register 9.29. EFUSE_ECDSA_REG (0x01B0)



EFUSE_CFG_ECDSA_P192_BLK Configures which block to use for ECDSA P192 key output. (R/W)

EFUSE_CFG_ECDSA_P256_BLK Configures which block to use for ECDSA P256 key output. (R/W)

EFUSE_CFG_ECDSA_P384_L_BLK Configures which block to use for ECDSA P384 key low part output. (R/W)

EFUSE_CFG_ECDSA_P384_H_BLK Configures which block to use for ECDSA P384 key high part output. (R/W)

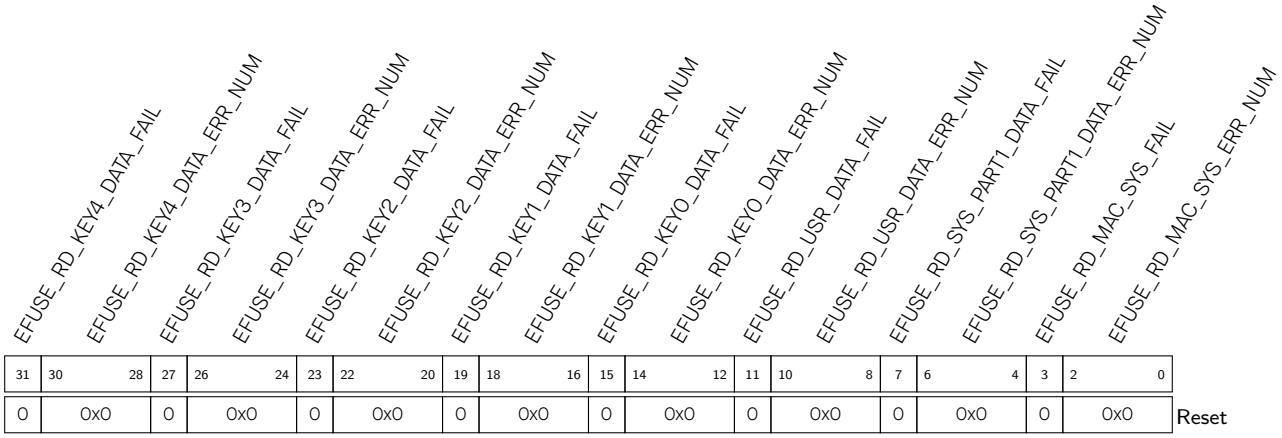
EFUSE_CUR_ECDSA_P192_BLK Represents which block is used for ECDSA P192 key output. (RO)

EFUSE_CUR_ECDSA_P256_BLK Represents which block is used for ECDSA P256 key output. (RO)

EFUSE_CUR_ECDSA_P384_L_BLK Represents which block is used for ECDSA P384 key low part output. (RO)

EFUSE_CUR_ECDSA_P384_H_BLK Represents which block is used for ECDSA P384 key high part output. (RO)

Register 9.30. EFUSE_RD_RS_DATA_ERR0_REG (0x01C0)



EFUSE_RD_MAC_SYS_ERR_NUM Represents the error number of registers. (RO)

The value of this signal means the number of error bytes in rd_mac_sys.

EFUSE_RD_MAC_SYS_FAIL Represents the error status of register. (RO)

0: means no failure and that the data of rd_mac_sys is reliable

1: means that programming rd_mac_sys failed and the number of error bytes is over 6

EFUSE_RD_SYS_PART1_DATA_ERR_NUM Represents the error number of registers. (RO)

The value of this signal means the number of error bytes in rd_sys_part1_data (RO)

EFUSE_RD_SYS_PART1_DATA_FAIL Represents the error status of register. (RO)

0: means no failure and that the data of rd_sys_part1_data is reliable

1: means that programming rd_sys_part1_data failed and the number of error bytes is over 6.

EFUSE_RD_USR_DATA_ERR_NUM Represents the error number of registers. (RO)

The value of this signal means the number of error bytes in rd_usr_data (RO)

EFUSE_RD_USR_DATA_FAIL Represents the error status of register. (RO)

0: means no failure and that the data of rd_usr_data is reliable

1: means that programming rd_usr_data failed and the number of error bytes is over 6.

EFUSE_RD_KEY0_DATA_ERR_NUM Represents the error number of registers. (RO)

The value of this signal means the number of error bytes in rd_key0_data (RO)

EFUSE_RD_KEY0_DATA_FAIL Represents the error status of register. (RO)

0: means no failure and that the data of rd_key0_data is reliable

1: means that programming rd_key0_data failed and the number of error bytes is over 6.

EFUSE_RD_KEY1_DATA_ERR_NUM Represents the error number of registers. (RO)

The value of this signal means the number of error bytes in rd_key1_data (RO)

EFUSE_RD_KEY1_DATA_FAIL Represents the error status of register. (RO)

0: means no failure and that the data of rd_key1_data is reliable

1: means that programming rd_key1_data failed and the number of error bytes is over 6.

EFUSE_RD_KEY2_DATA_ERR_NUM Represents the error number of registers. (RO)

The value of this signal means the number of error bytes in rd_key2_data (RO)

Continued on the next page...

Register 9.30. EFUSE_RD_RS_DATA_ERR0_REG (0x01C0)

Continued from the previous page...

EFUSE_RD_KEY2_DATA_FAIL Represents the error status of register. (RO)

0: means no failure and that the data of rd_key2_data is reliable

1: means that programming rd_key2_data failed and the number of error bytes is over 6.

EFUSE_RD_KEY3_DATA_ERR_NUM Represents the error number of registers. (RO)

The value of this signal means the number of error bytes in rd_key3_data (RO)

EFUSE_RD_KEY3_DATA_FAIL Represents the error status of register. (RO)

0: means no failure and that the data of rd_key3_data is reliable

1: means that programming rd_key3_data failed and the number of error bytes is over 6.

EFUSE_RD_KEY4_DATA_ERR_NUM Represents the error number of registers. (RO)

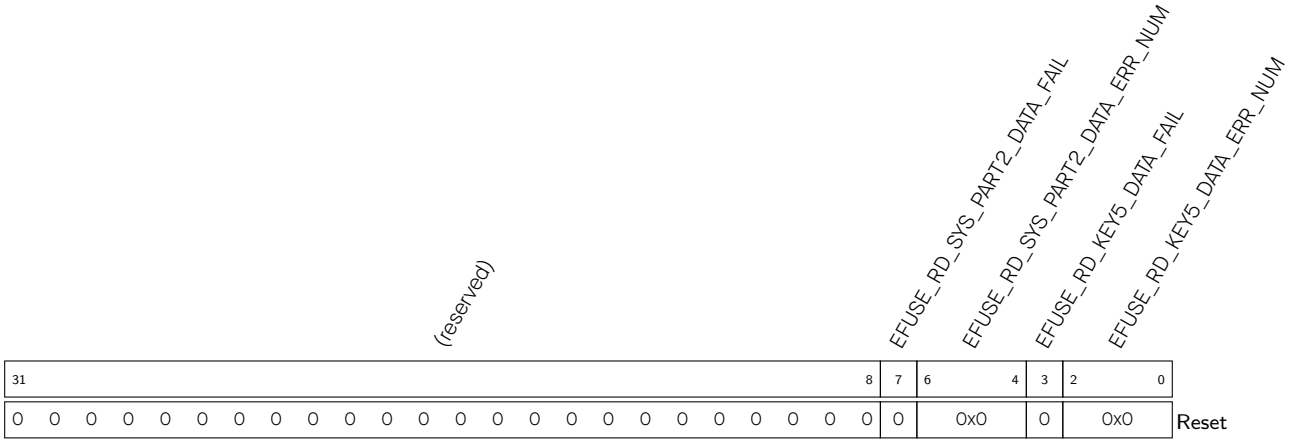
The value of this signal means the number of error bytes in rd_key4_data.

EFUSE_RD_KEY4_DATA_FAIL Represents the error status of register. (RO)

0: means no failure and that the data of rd_key4_data is reliable

1: means that programming rd_key4_data failed and the number of error bytes is over 6.

Register 9.31. EFUSE_RD_RS_DATA_ERR1_REG (0x01C4)



EFUSE_RD_KEY5_DATA_ERR_NUM Represents the error number of registers. (RO)

The value of this signal means the number of error bytes in rd_key5_data (RO)

EFUSE_RD_KEY5_DATA_FAIL Represents the error status of register. (RO)

0: means no failure and that the data of rd_key5_data is reliable

1: means that programming rd_key5_data failed and the number of error bytes is over 6

EFUSE_RD_SYS_PART2_DATA_ERR_NUM Represents the error number of registers. (RO)

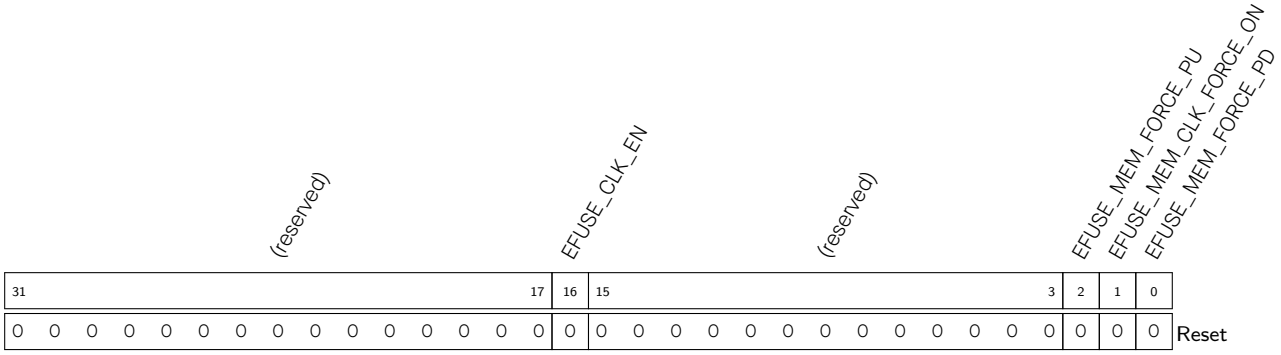
The value of this signal means the number of error bytes in rd_sys_part2_data (RO)

EFUSE_RD_SYS_PART2_DATA_FAIL Represents the error status of register. (RO)

0: means no failure and that the data of rd_sys_part2_data is reliable

1: means that programming rd_sys_part2_data failed and the number of error bytes is over 6

Register 9.32. EFUSE_CLK_REG (0x01C8)



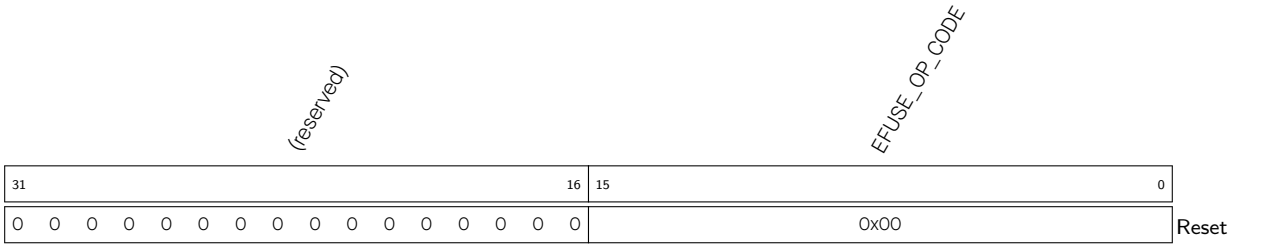
EFUSE_MEM_FORCE_PD Configures whether to force eFuse SRAM into power-saving mode. (R/W)

EFUSE_MEM_CLK_FORCE_ON Configures whether to force activate clock signal of eFuse SRAM. (R/W)

EFUSE_MEM_FORCE_PU Configures whether to force eFuse SRAM into working mode. (R/W)

EFUSE_CLK_EN Configures whether to force enable eFuse register configuration clock signal. (R/W)

Register 9.33. EFUSE_CONF_REG (0x01CC)

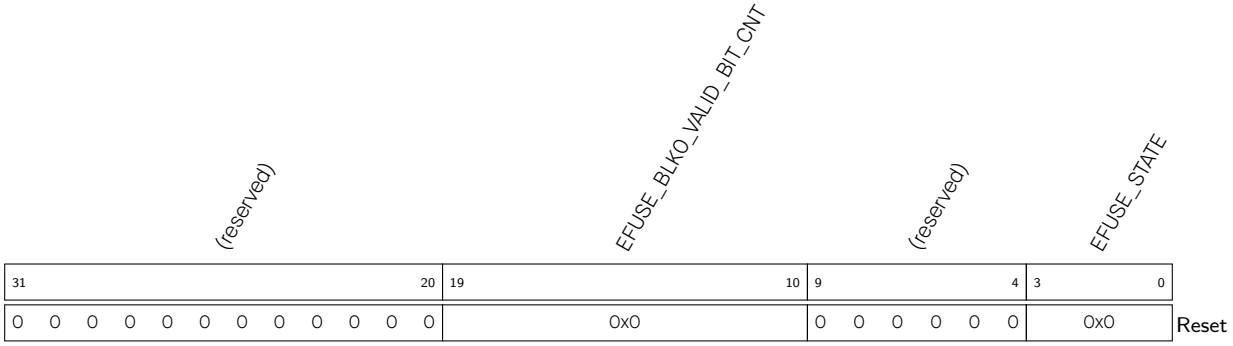


EFUSE_OP_CODE Configures the operation command. (R/W)

0x5A5A: programming operation command

0x5AA5: read operation command

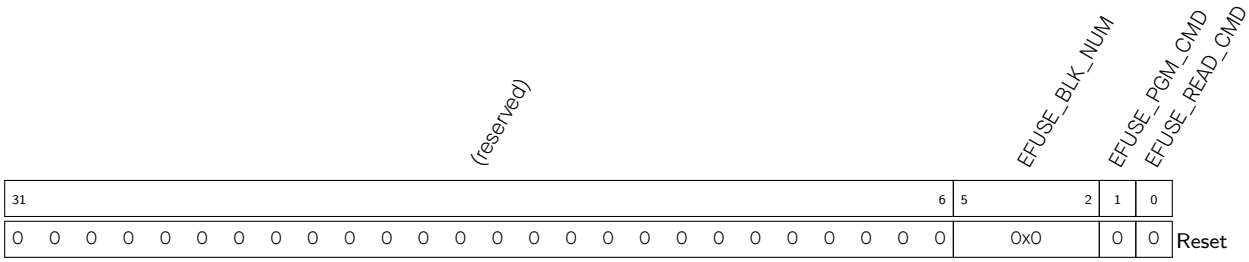
Register 9.34. EFUSE_STATUS_REG (0x01D0)



EFUSE_STATE Represents the state of the eFuse state machine. (RO)

EFUSE_BLK0_VALID_BIT_CNT Represents the number of block valid bits. (RO)

Register 9.35. EFUSE_CMD_REG (0x01D4)

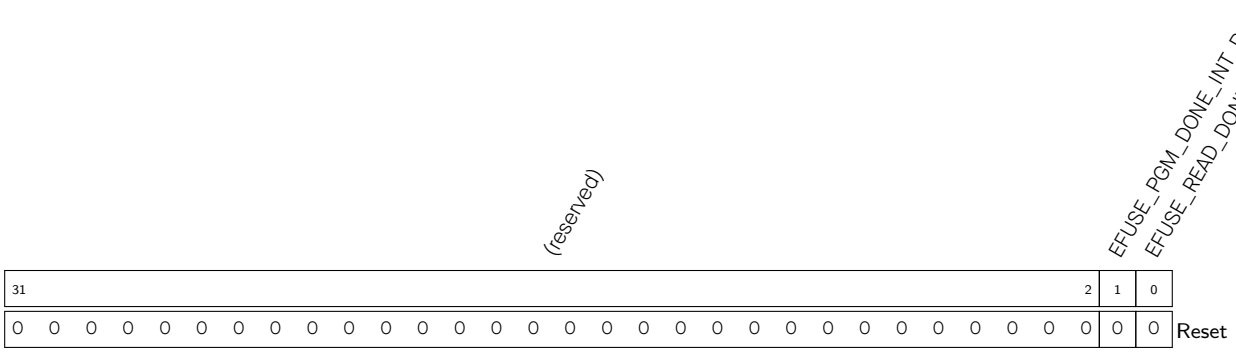


EFUSE_READ_CMD Configures whether to send read command. (R/W/SC)

EFUSE_PGM_CMD Configures whether to send programming command. (R/W/SC)

EFUSE_BLK_NUM Configures the serial number of the block to be programmed. (R/W)
Value 0-10 corresponds to block number 0-10, respectively.

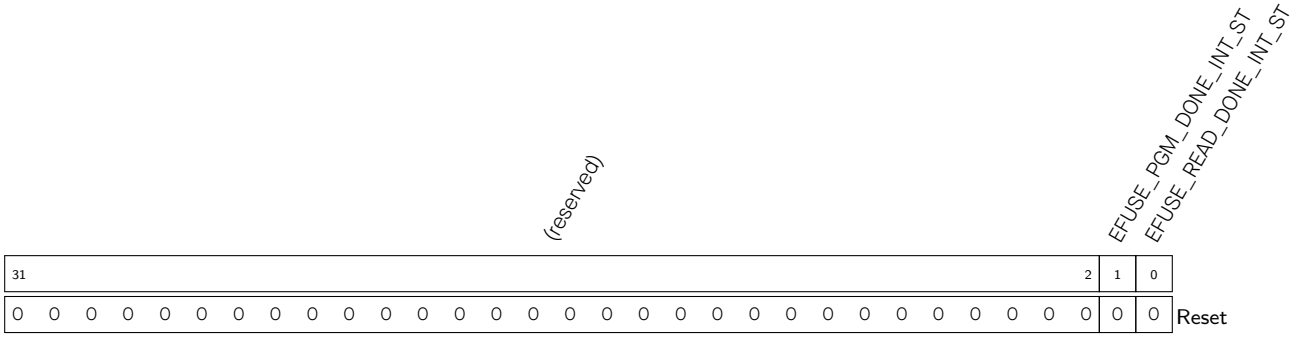
Register 9.36. EFUSE_INT_RAW_REG (0x01D8)



EFUSE_READ_DONE_INT_RAW The raw interrupt status of read_done interrupt. (R/SS/WTC)

EFUSE_PGM_DONE_INT_RAW The raw interrupt status of pgm_done interrupt. (R/SS/WTC)

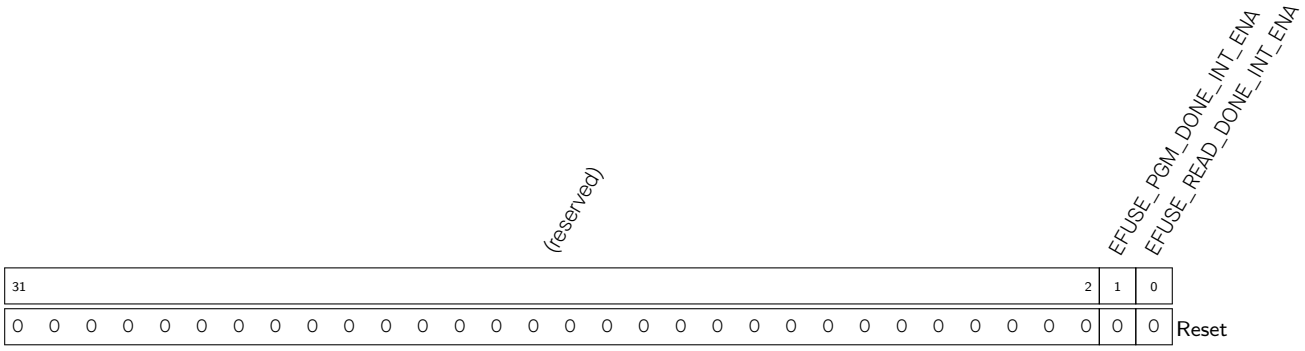
Register 9.37. EFUSE_INT_ST_REG (0x01DC)



EFUSE_READ_DONE_INT_ST The interrupt status of read_done interrupt. (RO)

EFUSE_PGM_DONE_INT_ST The interrupt status of pgm_done interrupt. (RO)

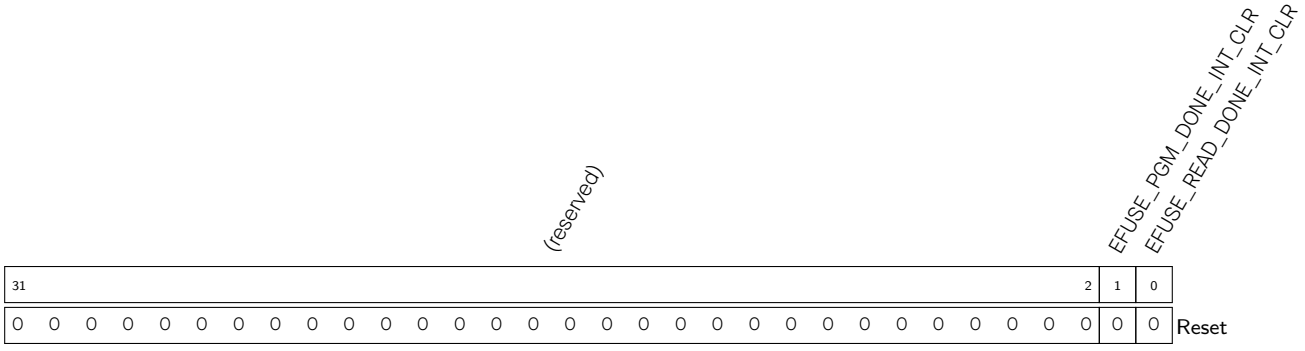
Register 9.38. EFUSE_INT_ENA_REG (0x01E0)



EFUSE_READ_DONE_INT_ENA Write 1 to enable read_done interrupt. (R/W)

EFUSE_PGM_DONE_INT_ENA Write 1 to enable pgm_done interrupt. (R/W)

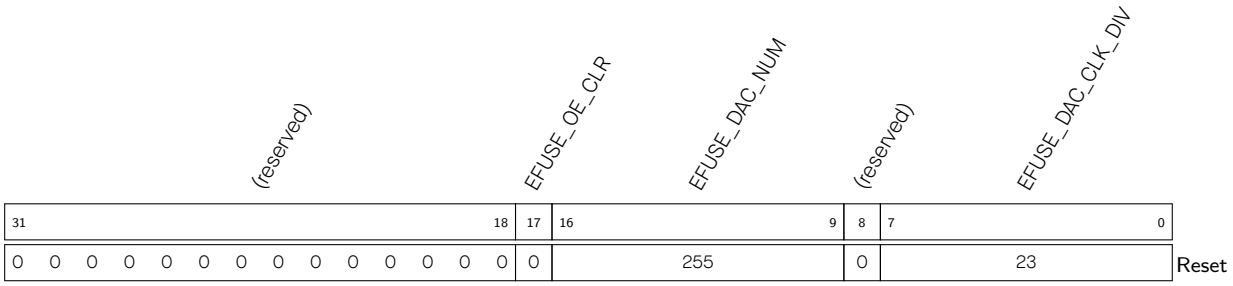
Register 9.39. EFUSE_INT_CLR_REG (0x01E4)



EFUSE_READ_DONE_INT_CLR Write 1 to clear read_done interrupt. (WT)

EFUSE_PGM_DONE_INT_CLR Write 1 to clear pgm_done interrupt. (WT)

Register 9.40. EFUSE_DAC_CONF_REG (0x01E8)

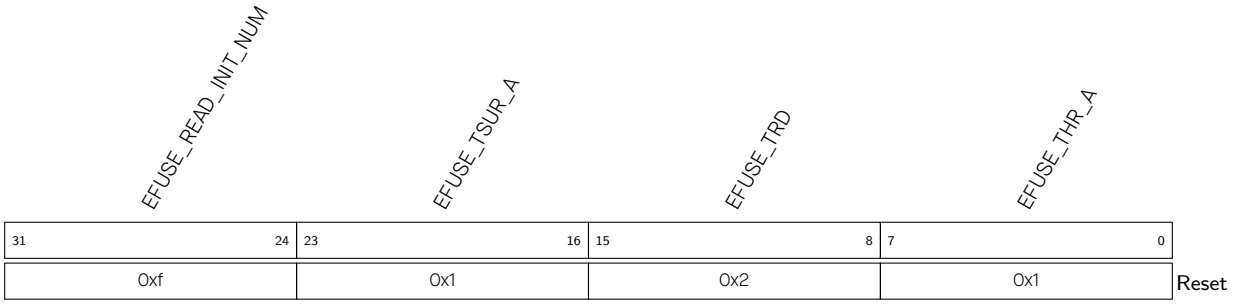


EFUSE_DAC_CLK_DIV Configures the division factor of the rising clock of the programming voltage. (R/W)

EFUSE_DAC_NUM Configures the rising period of the programming voltage. (R/W)

EFUSE_OE_CLR Configures whether to reduce the power supply of the programming voltage. (R/W)

Register 9.41. EFUSE_RD_TIM_CONF_REG (0x01EC)



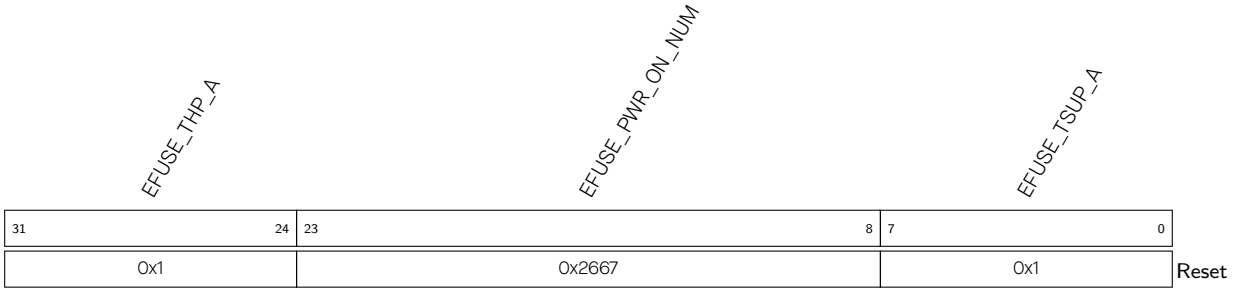
EFUSE_THR_A Configures the read hold time. (R/W)

EFUSE_TRD Configures the read time. (R/W)

EFUSE_TSUR_A Configures the read setup time. (R/W)

EFUSE_READ_INIT_NUM Configures the waiting time of reading eFuse memory. (R/W)

Register 9.42. EFUSE_WR_TIM_CONF1_REG (0x01F0)

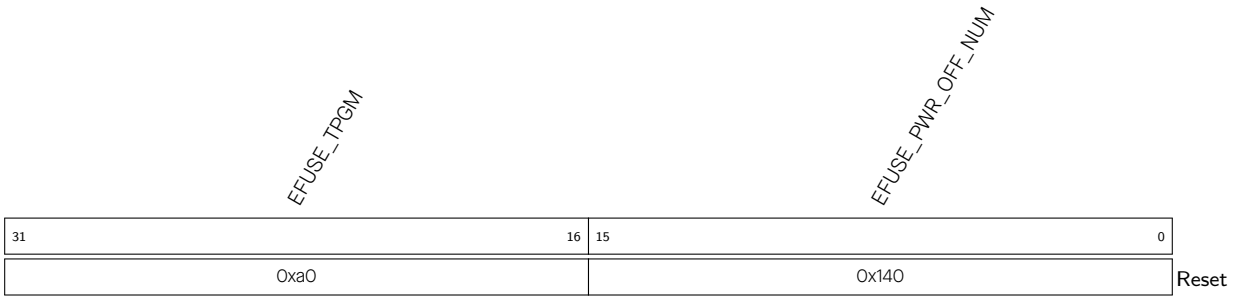


EFUSE_TSUP_A Configures the programming setup time. (R/W)

EFUSE_PWR_ON_NUM Configures the power up time for VDDQ. (R/W)

EFUSE_THP_A Configures the programming hold time. (R/W)

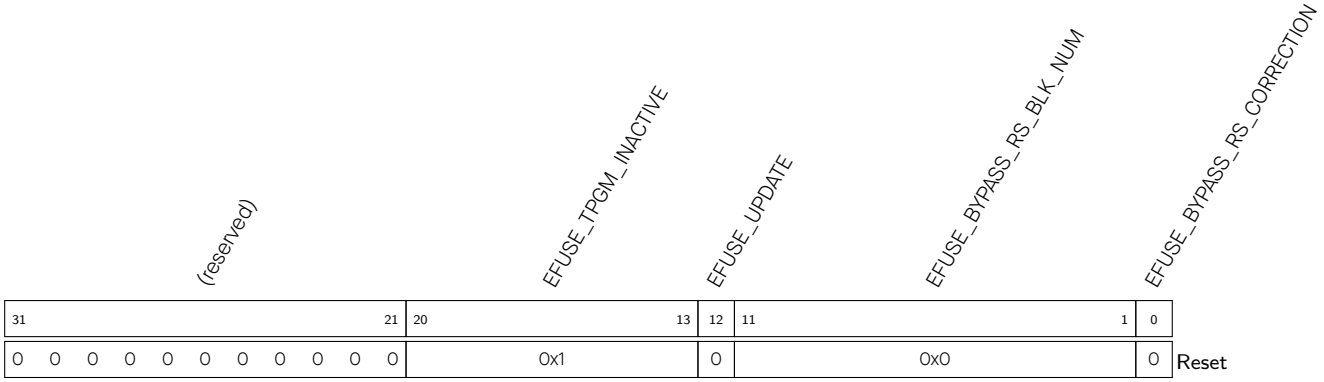
Register 9.43. EFUSE_WR_TIM_CONF2_REG (0x01F4)



EFUSE_PWR_OFF_NUM Configures the power outage time for VDDQ. (R/W)

EFUSE_TPGM Configures the active programming time. (R/W)

Register 9.44. EFUSE_WR_TIM_CONF0_RS_BYPASS_REG (0x01F8)



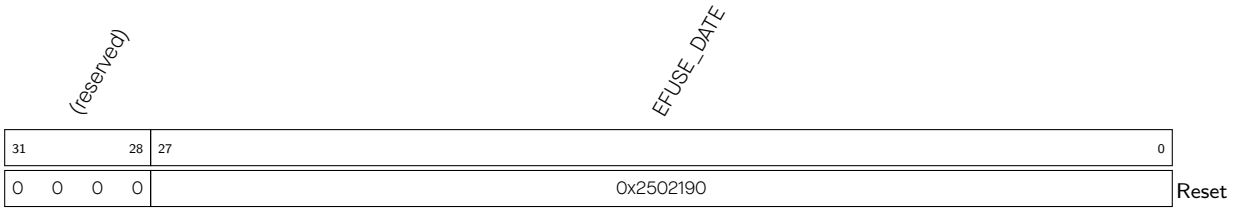
EFUSE_BYPASS_RS_CORRECTION Configures whether to bypass Reed-Solomon correction step. (R/W)

EFUSE_BYPASS_RS_BLK_NUM Configures block number of programming twice operation. (R/W)

EFUSE_UPDATE Write 1 to update multi-bit register signals. (WT)

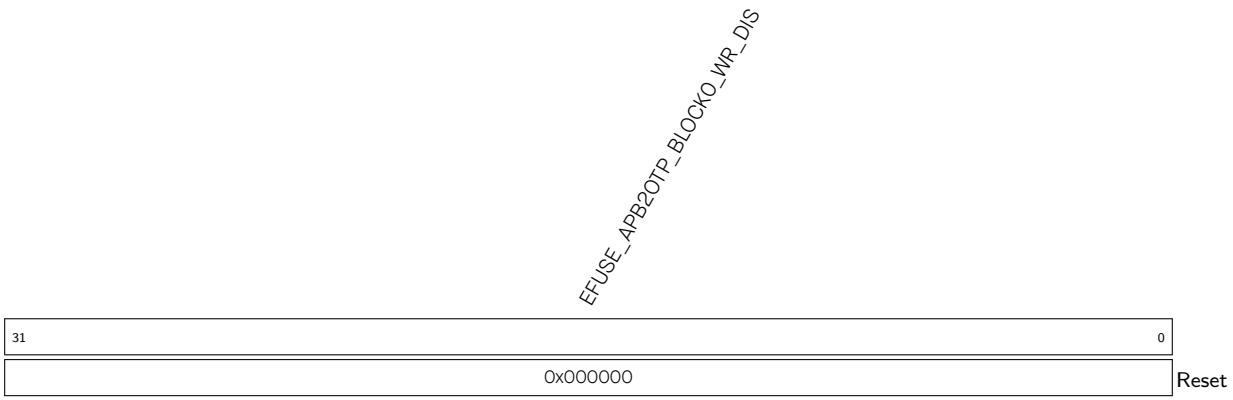
EFUSE_TPGM_INACTIVE Configures the inactive programming time. (R/W)

Register 9.45. EFUSE_DATE_REG (0x01FC)



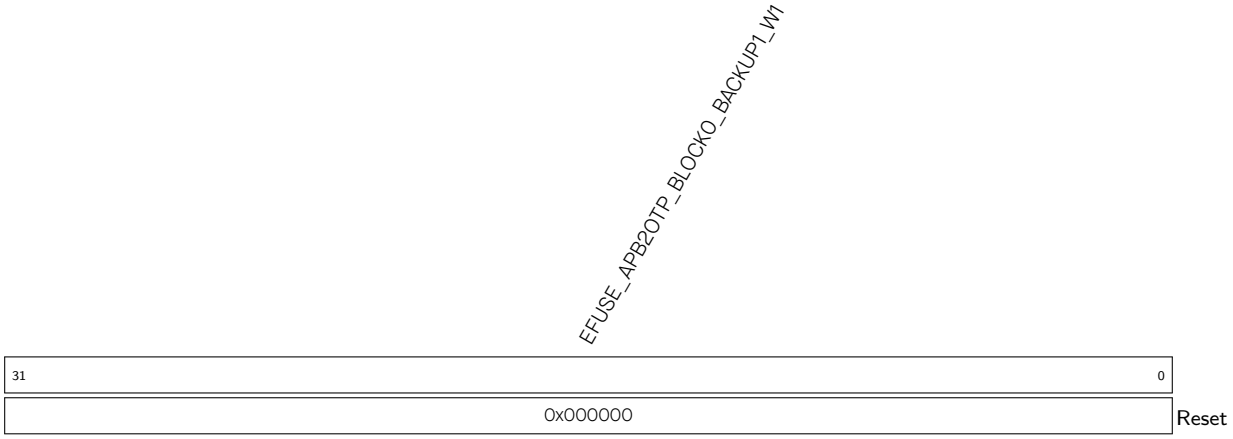
EFUSE_DATE Configures the eFuse version. (R/W)

Register 9.46. EFUSE_APB2OTP_WR_DIS_REG (0x0800)



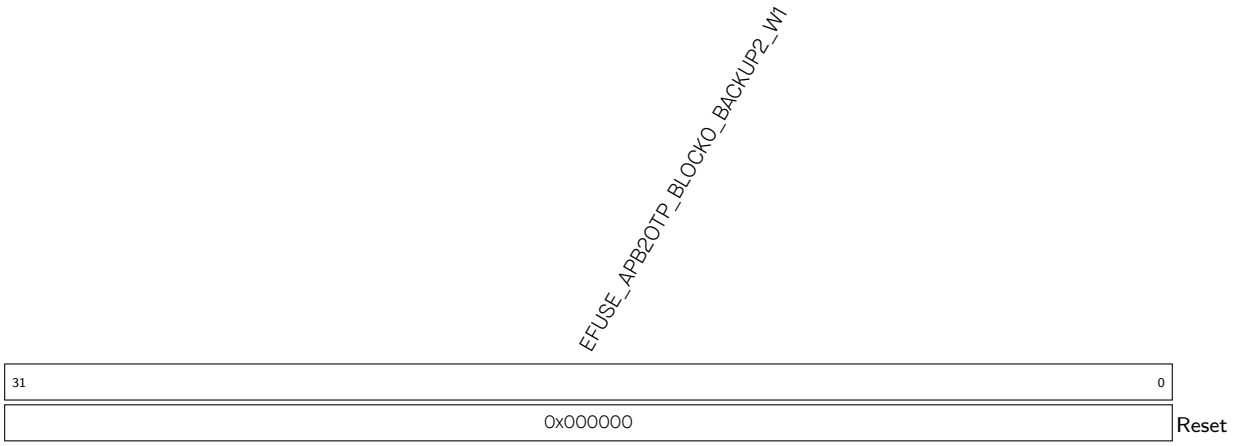
EFUSE_APB2OTP_BLOCK0_WR_DIS Represents OTP block0 write disable data. (RO)

Register 9.47. EFUSE_APB2OTP_BLK0_BACKUP1_W*n*_REG (*n*: 1-5) (0x0800+0x4**n*)



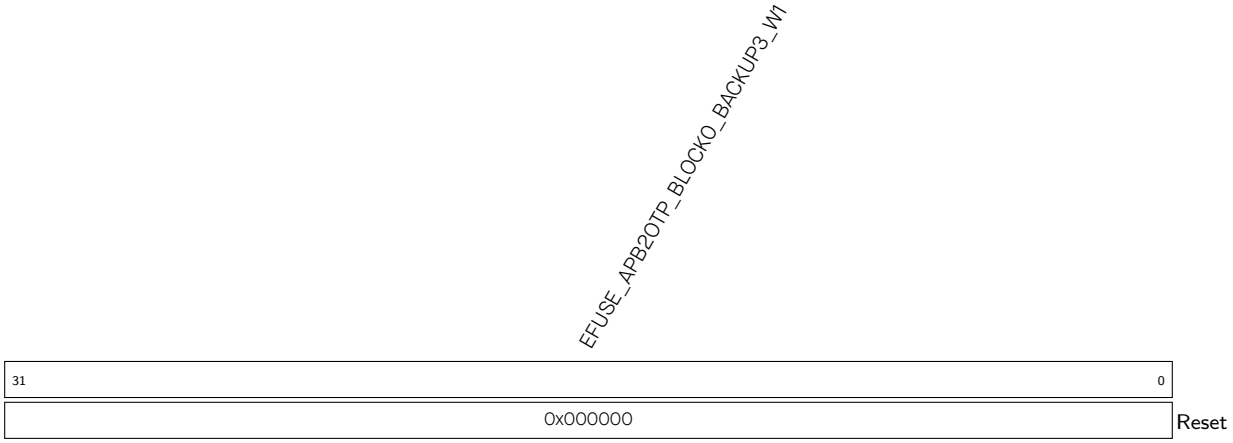
EFUSE_APB2OTP_BLOCK0_BACKUP1_W*n* Represents 32-bit word *n* of OTP block0 backup1. (RO)

Register 9.48. EFUSE_APB2OTP_BLK0_BACKUP2_W*n*_REG (*n*: 1-5) (0x0814+0x4**n*)



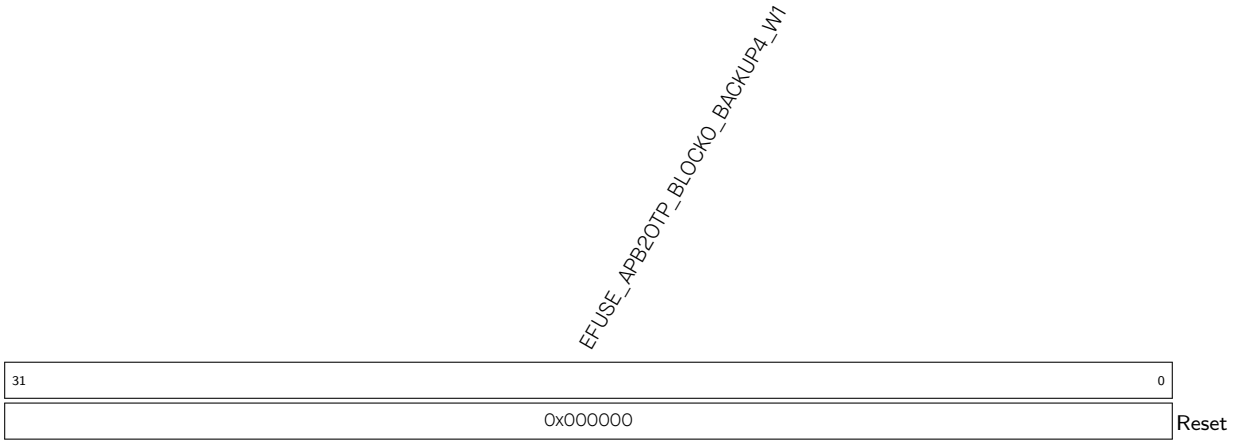
EFUSE_APB2OTP_BLOCK0_BACKUP2_W*n* Represents 32-bit word *n* of OTP block0 backup2. (RO)

Register 9.49. EFUSE_APB2OTP_BLK0_BACKUP3_W*n*_REG (*n*: 1-5) (0x0828+0x4**n*)



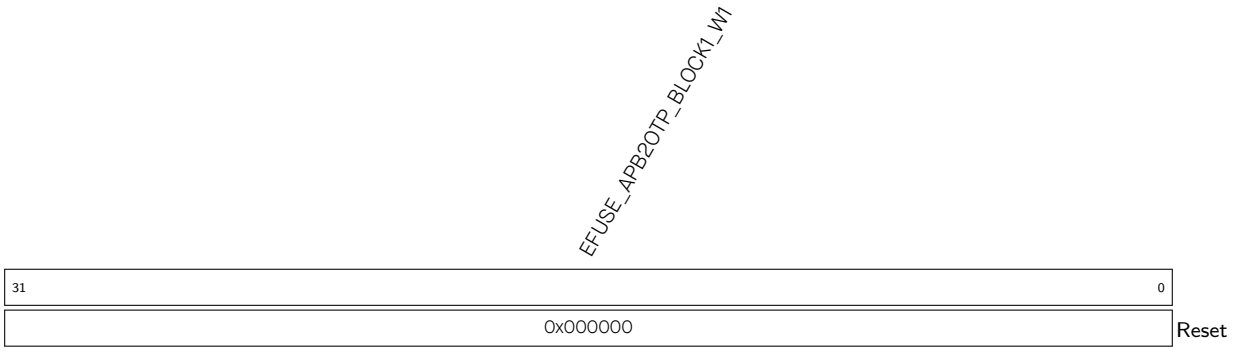
EFUSE_APB2OTP_BLOCK0_BACKUP3_W*n* Represents 32-bit word *n* of OTP block0 backup3. (RO)

Register 9.50. EFUSE_APB2OTP_BLK0_BACKUP4_W*n*_REG (*n*: 1-5) (0x083C+0x4**n*)



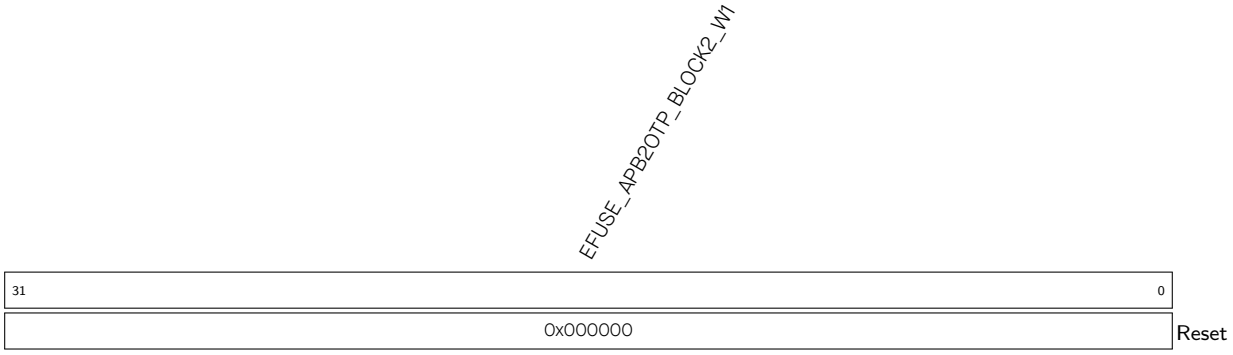
EFUSE_APB2OTP_BLOCK0_BACKUP4_W*n* Represents 32-bit word *n* of OTP block0 backup4. (RO)

Register 9.51. EFUSE_APB2OTP_BLK1_W*n*_REG (*n*: 1-9) (0x0850+0x4**n*)



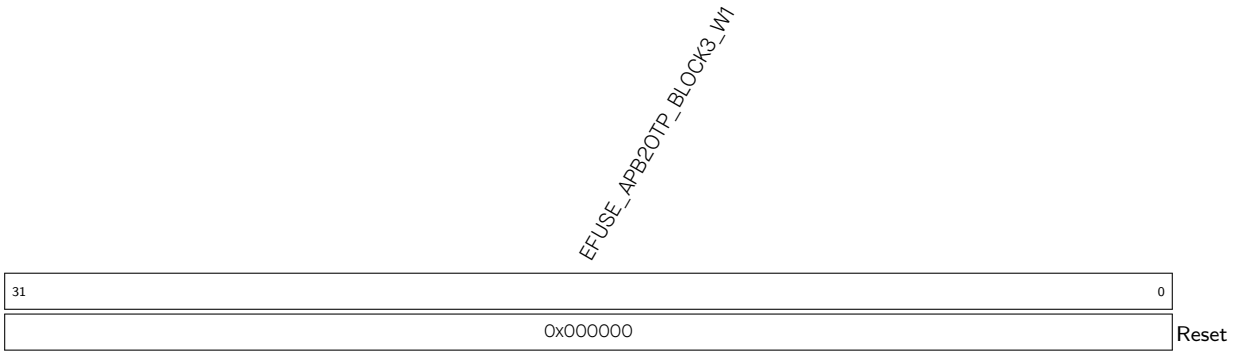
EFUSE_APB2OTP_BLOCK1_W*n* Represents 32-bit word *n* of OTP block1. (RO)

Register 9.52. EFUSE_APB2OTP_BLK2_W*n*_REG (*n*: 1-11) (0x0874+0x4**n*)



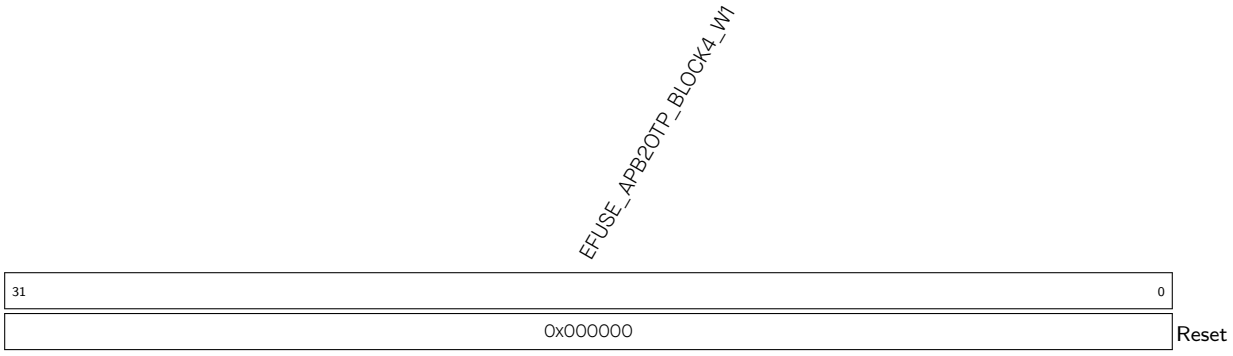
EFUSE_APB2OTP_BLOCK2_W*n* Represents 32-bit word *n* of OTP block2. (RO)

Register 9.53. EFUSE_APB2OTP_BLK3_W*n*_REG (*n*: 1-11) (0x08A0+0x4**n*)



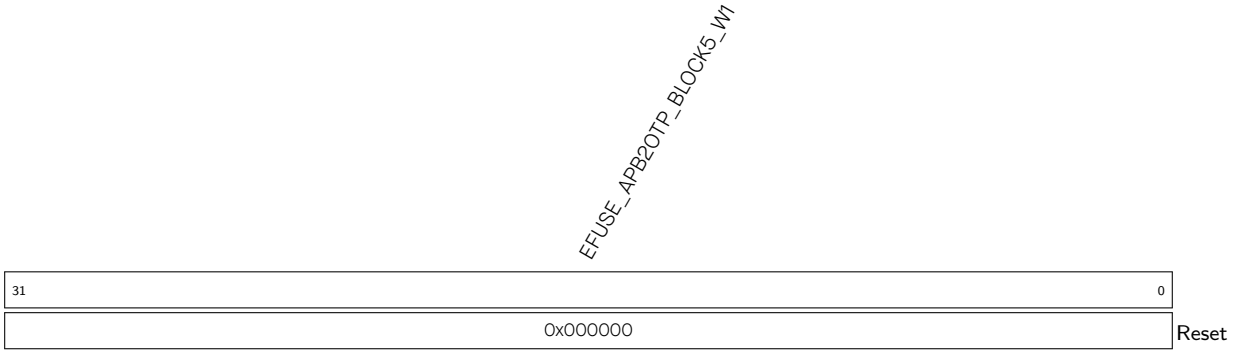
EFUSE_APB2OTP_BLOCK3_W*n* Represents 32-bit word *n* of OTP block3. (RO)

Register 9.54. EFUSE_APB2OTP_BLK4_W*n*_REG (*n*: 1-11) (0x08CC+0x4**n*)



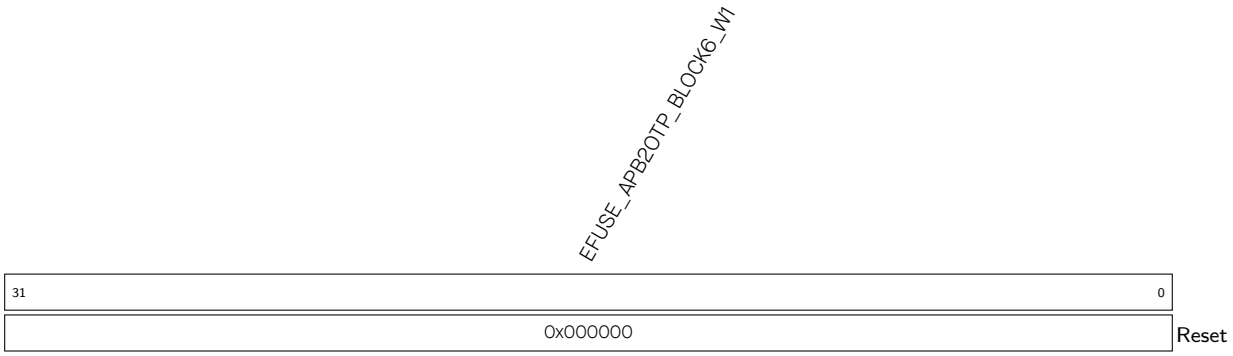
EFUSE_APB2OTP_BLOCK4_W*n* Represents 32-bit word *n* of OTP block4. (RO)

Register 9.55. EFUSE_APB2OTP_BLK5_W*n*_REG (*n*: 1-11) (0x08F8+0x4**n*)



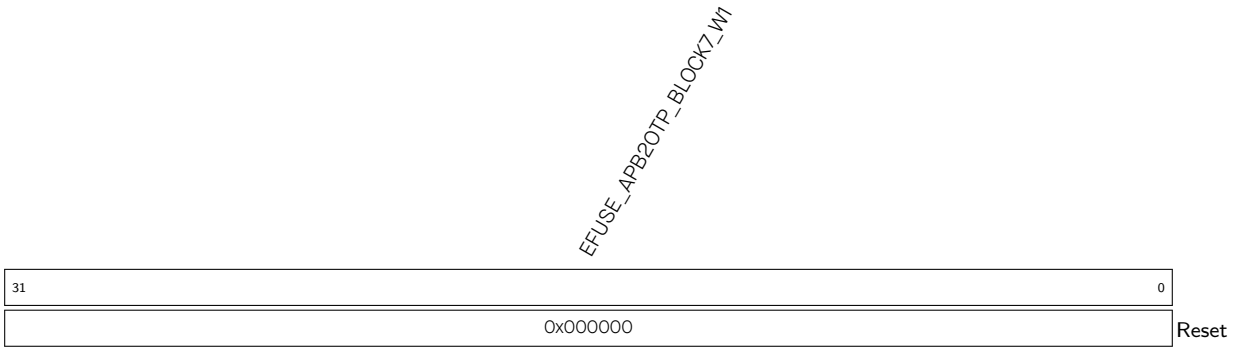
EFUSE_APB2OTP_BLOCK5_W*n* Represents 32-bit word *n* of OTP block5. (RO)

Register 9.56. EFUSE_APB2OTP_BLK6_W*n*_REG (*n*: 1-11) (0x0924+0x4**n*)



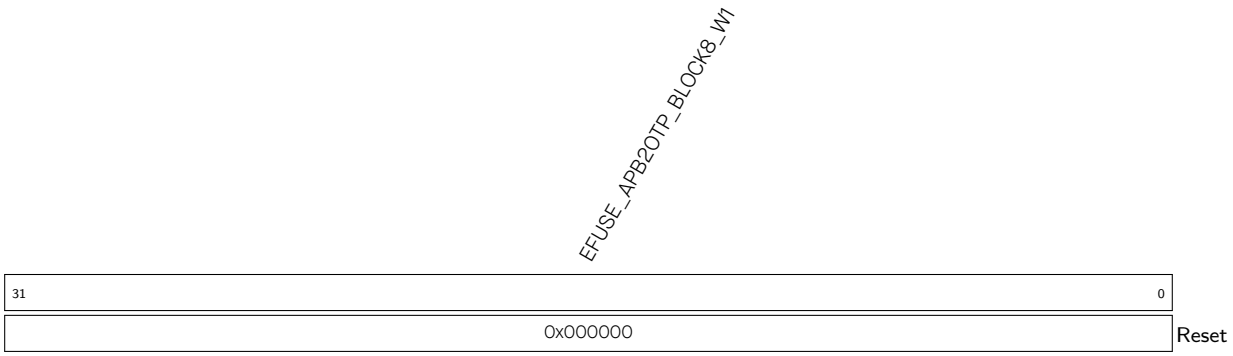
EFUSE_APB2OTP_BLOCK6_W*n* Represents 32-bit word *n* of OTP block6. (RO)

Register 9.57. EFUSE_APB2OTP_BLK7_W*n*_REG (*n*: 1-11) (0x0950+0x4**n*)



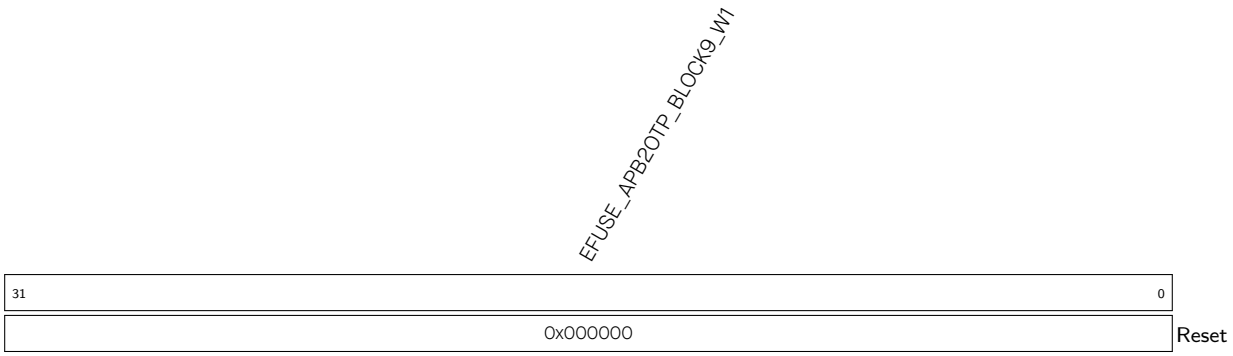
EFUSE_APB2OTP_BLOCK7_W*n* Represents 32-bit word *n* of OTP block7. (RO)

Register 9.58. EFUSE_APB2OTP_BLK8_W*n*_REG (*n*: 1-11) (0x097C+0x4**n*)



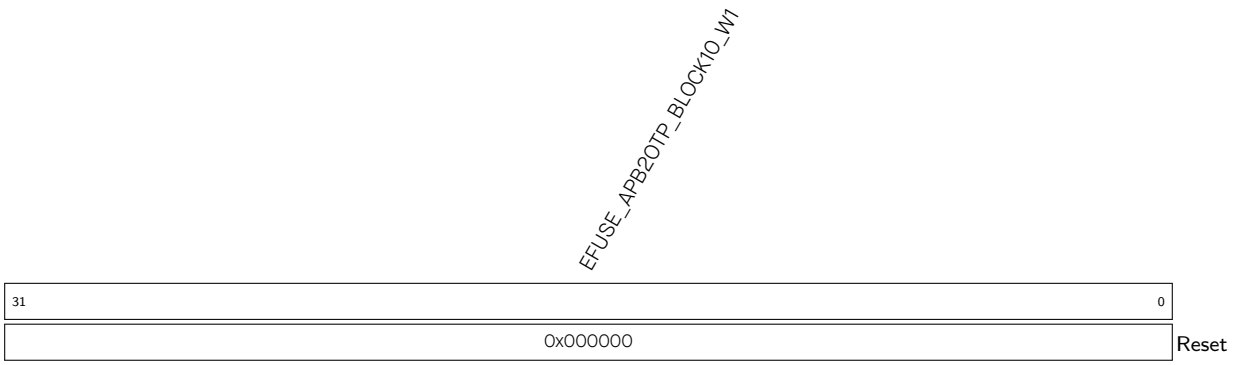
EFUSE_APB2OTP_BLOCK8_W*n* Represents 32-bit word *n* of OTP block8. (RO)

Register 9.59. EFUSE_APB2OTP_BLK9_W*n*_REG (*n*: 1-11) (0x09A8+0x4**n*)



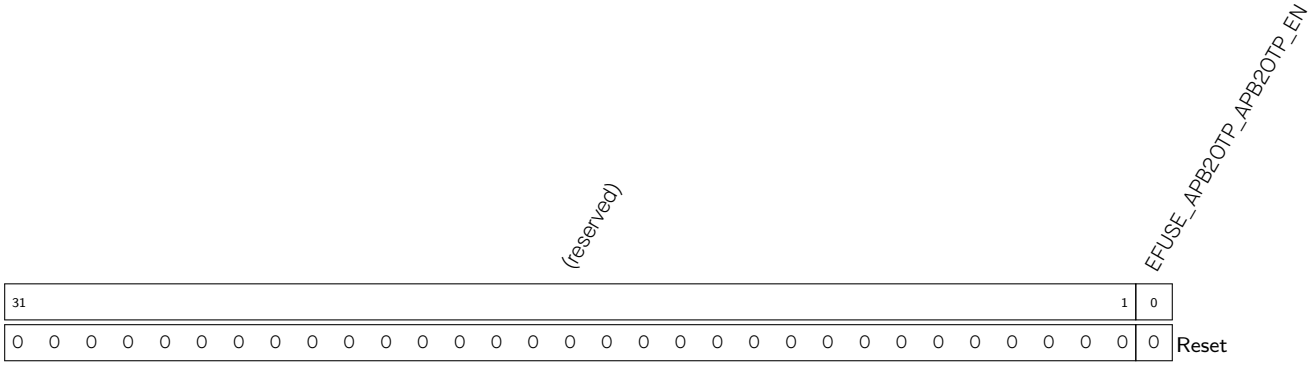
EFUSE_APB2OTP_BLOCK9_W*n* Represents 32-bit word *n* of OTP block9. (RO)

Register 9.60. EFUSE_APB2OTP_BLK10_W*n*_REG (*n*: 1-11) (0x09D4+0x4**n*)



EFUSE_APB2OTP_BLOCK10_W*n* Represents 32-bit word *n* of OTP block10. (RO)

Register 9.61. EFUSE_APB2OTP_EN_REG (0x0A08)



EFUSE_APB2OTP_APB2OTP_EN Configures whether APB2OTP mode is enabled. (R/W)

Part IV

System Component

Encompassing a range of system-level functionalities, this part describes components related to system boot, clocks, GPIO, timers, watchdogs, debug assistance, event and interrupt handling, low-power management, and various system registers.

Chapter 10

GPIO Matrix and IO MUX

10.1 Overview

GPIO matrix is a hardware structure that allows dynamic remapping between the GPIO pins and peripheral input/output signals to provide greater flexibility. IO MUX (Input/output multiplexing) is a technology that allows the same pin to perform different functions at different times, allowing dynamic switching of pin functions through configuration. ESP32-P4 provides two groups of IO MUX and GPIO matrix:

- HP GPIO matrix and HP IO MUX
- LP GPIO matrix and LP IO MUX

The ESP32-P4 chip features 55 GPIO pins, including 16 low-power (LP) GPIO pins and 39 high-performance (HP) GPIO pins. Each pin can be used as a general-purpose I/O, or be connected to an internal peripheral signal.

- Through HP GPIO matrix and HP IO MUX, HP peripheral input signals can be from any GPIO pins, and HP peripheral output signals can be routed to any GPIO pins.
- Through LP GPIO matrix and LP IO MUX, LP peripheral input signals can be from any LP GPIO pins, and LP peripheral output signals can be routed to any LP GPIO pins.

Together these modules provide highly configurable I/O. The 55 GPIO pins are numbered from GPIO0 to GPIO54.

- LP GPIO pins (GPIO0 ~ GPIO15) can be used by either HP or LP peripherals.
- HP GPIO pins (GPIO16 ~ GPIO54) can be used only by HP peripherals.

The ESP32-P4 chip has dedicated pins for external flash and in-package PSRAM. Such pins can not be used for other purpose. See *ESP32-P4 Datasheet* > Section *Pin Mapping Between Chip and Flash/PSRAM*.

Unless otherwise specified, GPIO matrix refers to both LP GPIO matrix and HP GPIO matrix, so as to IO MUX.

10.2 Features

10.2.1 HP GPIO Matrix and HP IO MUX

HP GPIO matrix has the following features:

- A full-switching matrix between HP peripheral input/output signals and the GPIO pins
- 222 HP peripheral input signals sourced from the input of any GPIO pins
- 232 HP peripheral output signals routed to the output of any GPIO pins

- Signal synchronization for HP peripheral inputs based on **HP IO MUX operating clock**
- GPIO Filter hardware for input signal filtering
- Glitch Filter hardware for second-time filtering on input signal
- Sigma delta modulated (SDM) output
- GPIO simple input and output
- HP GPIO Wakeup

HP IO MUX has the following features:

- Control of 55 GPIOs (GPIO0 ~ GPIO54) for HP peripherals.
- A configuration register IO_MUX_GPIO*n*_REG provided for each GPIO pin, to control the pin's input/output, pull-up/pull-down, drive strength, and function selection.
- Better high-frequency digital performance achieved by routing some digital signals (SPI, EMAC) directly from HP IO MUX to peripherals.

10.2.2 LP GPIO Matrix and LP IO MUX

LP GPIO matrix has the following features:

- A full-switching matrix between the LP peripheral input/output signals and the LP GPIO pins
- 14 LP peripheral input signals sourced from the input of any LP GPIO pins
- 14 LP peripheral output signals routed to the output of any LP GPIO pins
- GPIO Filter hardware for input signal filtering
- GPIO simple input and output
- LP GPIO Wakeup

LP IO MUX has the following feature:

- Control of 16 LP GPIO pins (GPIO0 ~ GPIO15) for LP peripherals.
- A configuration register LP_IOMUX_PAD*n*_REG provided for each LP GPIO pin, to control the pin's input/output, pull-up/pull-down, drive strength, function selection, and IO MUX selection.

10.3 Architectural Overview

Figure 10.3-1 shows in details how HP GPIO matrix, HP IO MUX, LP GPIO matrix, and LP IO MUX route signals from pins to peripherals, and from peripherals to pins.

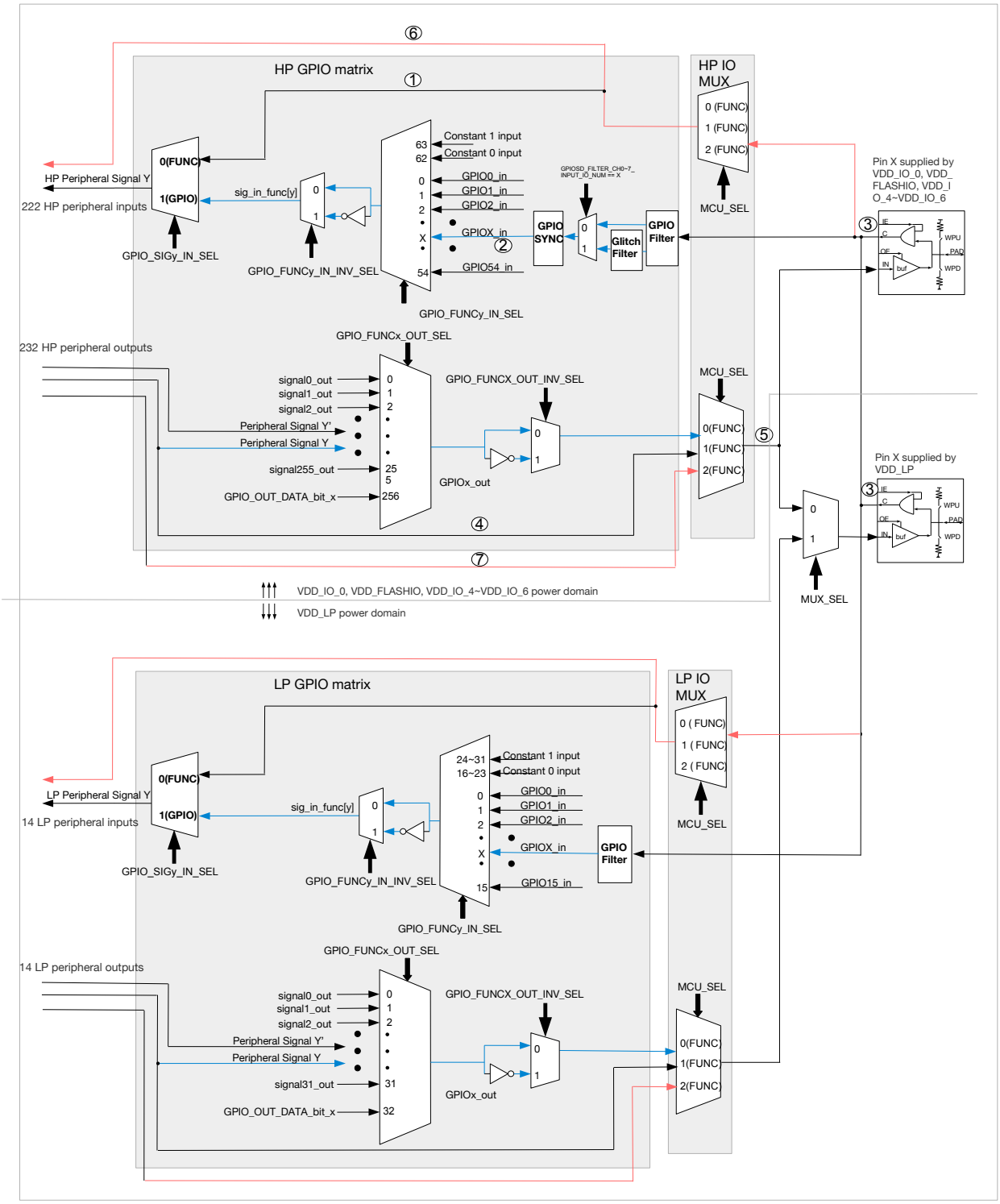


Figure 10.3-1. Architecture of HP GPIO Matrix, HP IO MUX, LP GPIO Matrix, and LP IO MUX

The following points explain the areas marked numerically in the figure above, taking the HP system as an example:

- ① Part of peripheral input signals (marked “yes” in column “Direct input through HP IO MUX” in Table

10.12-1) can be routed to the peripherals via HP IO MUX, or via HP GPIO matrix, while the other input signals can only be routed to the peripherals via HP GPIO matrix.

- ② There are only 55 inputs from GPIO SYNC to HP GPIO matrix, since ESP32-P4 provides 55 GPIO pins in total.
- ③ The pins supplied by VDD_IO_0, VDD_FLASHIO, VDD_IO_4 ~ VDD_IO_6, and VDD_LP are controlled by the signals: IE, OE, HYS, WPU, and WPD.
- ④ Part of peripheral outputs (marked "yes" in column "Direct output through HP IO MUX" in Table 10.12-1) can be routed to pins via HP IO MUX or via HP GPIO matrix, while the other output signals can only be routed to pins via HP GPIO matrix.
- ⑤ There are 55 outputs (corresponding to GPIO pin *X*: 0 ~ 54) from HP GPIO matrix to HP IO MUX.
- ⑥ There are peripheral inputs only through HP IO MUX.
- ⑦ There are peripheral outputs only through HP IO MUX.

Figure 10.3-2 shows the internal structure of a pad, which is an electrical interface between the chip logic and the GPIO pin. The structure is applicable to all 55 GPIO pins and can be controlled using IE, OE, WPU, and WPD signals. For the configuration of these signals, see IO_MUX_GPIO*x*_REG orLP_IOMUX_PAD*x*_REG.

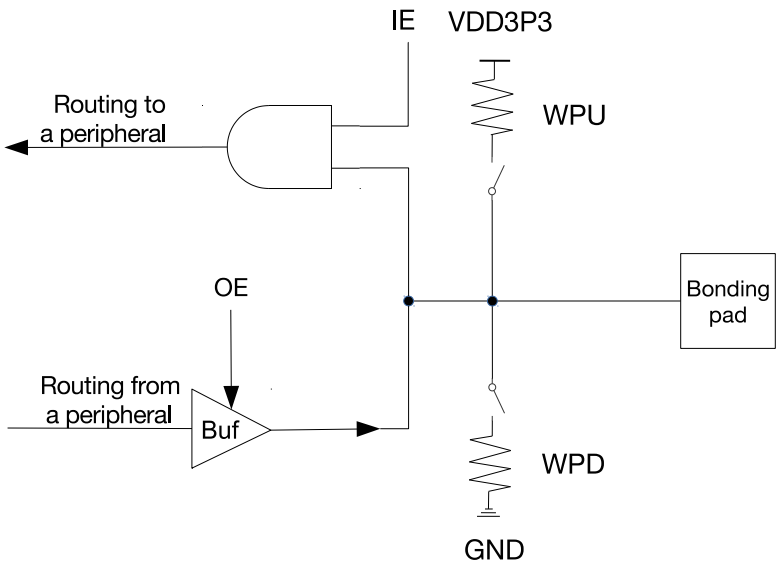


Figure 10.3-2. Internal Structure of a Pad

- IE: input enable
- OE: output enable
- WPU: internal weak pull-up resistor
- WPD: internal weak pull-down resistor
- Bonding pad: a terminal point of the chip logic used to make a physical connection from the chip die to GPIO pin in the chip package

10.4 Peripheral Input via GPIO Matrix

10.4.1 Overview

To receive a peripheral input signal via HP GPIO matrix,

- configure the matrix to source the peripheral input signal from one of the 55 GPIOs (0 ~ 54), see Table 10.12-1.
- configure the peripheral signal to receive input signal via HP GPIO matrix.
- configure the GPIO pin to be controlled by HP IO MUX.

For detailed configuration, see Figure 10.3-1 and Section 10.4.7.

To receive a peripheral input signal via LP GPIO matrix,

- configure the matrix to source the peripheral input signal from one of the 16 GPIOs (0 ~ 15), see Table 10.13-1.
- configure the peripheral signal to receive input signal via LP GPIO matrix.
- configure the LP GPIO pin to be controlled by LP IO MUX.

For detailed configuration, see Figure 10.3-1 and Section 10.4.7.

As shown in Figure 10.3-1, when GPIO matrix is used to input a signal from the pin, all external input signals are sourced from the GPIO pins and then filtered by the GPIO Filter, as shown in Step 2 in Section 10.4.7.

The Glitch Filter is only available in HP GPIO matrix. The Glitch Filter hardware can filter eight of the output signals from the GPIO Filter, and the other unselected signals go directly to the GPIO SYNC hardware, as shown in Step 3 in Section 10.4.7.

All signals filtered by the GPIO Filter hardware or the Glitch Filter hardware are synchronized by the GPIO SYNC hardware to IO MUX operating clock and then enter the GPIO matrix, see Section 10.4.2. Such signal filtering and synchronization features apply to all GPIO matrix signals but do not apply when using the IO MUX.

10.4.2 Signal Synchronization

Only HP GPIO matrix supports this signal synchronization function. Figure 10.4-1 shows the functionality of GPIO SYNC. In the figure, negative sync and positive sync mean GPIO input is synchronized on falling edge and on rising edge of HP IO MUX operating clock respectively.

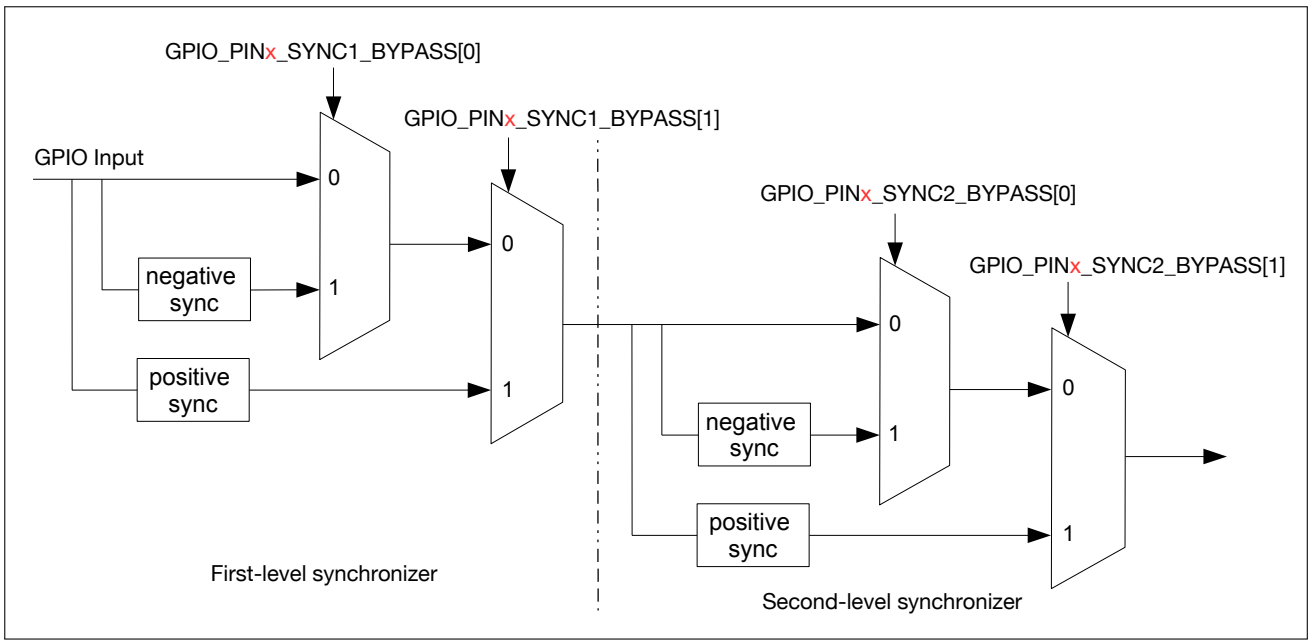


Figure 10.4-1. GPIO Input Synchronized on Rising Edge or on Falling Edge of HP IO MUX Operating Clock

The synchronization function is disabled by default by the synchronizer. But when an asynchronous peripheral signal is connected to the pin, the signal should be synchronized by the two-level synchronizer (i.e., the first-level synchronizer and the second-level synchronizer as shown in Figure 10.4-1) to lower the probability of causing metastability.

10.4.3 GPIO Filter

Both the HP GPIO matrix and LP GPIO matrix support GPIO Filter function. When this function is enabled, only signals with a valid width of more than two clock cycles can be sampled, as illustrated in Figure 10.4-2.

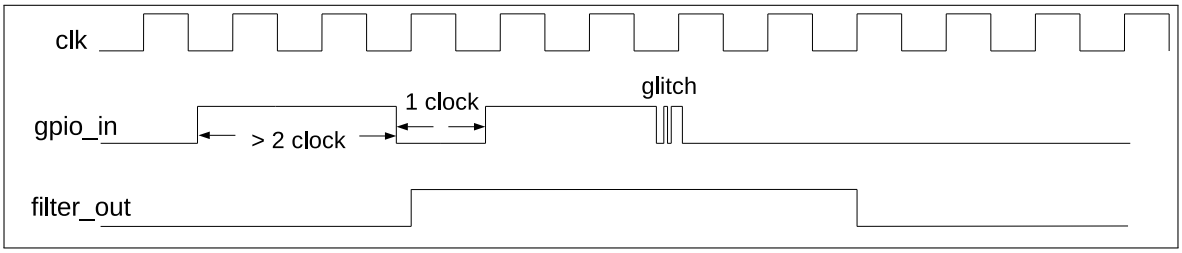


Figure 10.4-2. GPIO Filter Timing of GPIO Input Signals

10.4.4 Glitch Filter

The Glitch Filter function is exclusive to the HP GPIO matrix. The Glitch Filter hardware supports eight channels, each of which selects one signal from the 55 (0 ~ 54) output signals generated by the GPIO Filter hardware. It then conducts a second round of filtering on the selected signal. This Glitch Filter hardware can be used to filter slow-speed signals. For more information, see Step 3 in Section 10.4.7.1.

10.4.5 Simple GPIO Input

Both the HP GPIO matrix and LP GPIO matrix support the Simple GPIO Input function. Enabling this function allows the direct reading of the input value of a GPIO pin at any time, without the need to route the GPIO input to any peripherals.

For HP GPIO matrix, to implement simple GPIO input, follow the steps below:

- Set IO_MUX_GPIO*x*_MCU_IE in IO_MUX_GPIO*x*_REG, to enable pin input.
- Read the GPIO input from GPIO_IN_REG[*x*].

For LP GPIO matrix, to implement simple GPIO input, follow the steps below:

- Set LP_IOMUX_PAD*x*_FUN_IE in LP_IO_MUX_PAD*x*_REG, to enable pin input.
- Read the GPIO input from LP_GPIO_IN_REG[*x*].

10.4.6 GPIO Wakeup

10.4.6.1 HP GPIO Wakeup

The HP GPIO wakeup feature is designed to awaken the system from Light-sleep when the HP system is in clock gating mode. HP GPIO wakeup is not available when the HP system is powered down.

GPIO0 ~ GPIO54 can be used to generate HP GPIO wakeup by enabling GPIO_PIN*x*_WAKEUP_ENABLE (*x*: 0 ~ 54).

HP GPIO wakeup supports both high-voltage sensitive and low-voltage sensitive wakeups, with specific conditions based on the GPIO configuration.

Table 10.4-1. HP GPIO Wakeup Signal Trigger and Clear Conditions

GPIO_PIN*x*_INT_TYPE[1,2]	Wakeup is generated when	Wakeup is cleared when
5	the input GPIO is high-level.	the input GPIO is low-level.
4	the input GPIO is low-level.	the input GPIO is high-level.

[1] *x* ranges from 0 to 54.
[2] If the register is configured to any other value, HP GPIO wakeup feature is disabled.

10.4.6.2 LP GPIO Wakeup

The LP GPIO wakeup feature is designed to awaken the system from Deep-sleep when the LP peripherals are not in power gating mode. LP GPIO wakeup is not available when the LP peripherals are powered down.

GPIO0 ~ GPIO15 can be used to generate LP GPIO wakeup by enabling LP_GPIO_PIN*n*_WAKEUP_ENABLE (*n*: 0 ~ 15).

LP GPIO wakeup supports posedge sensitive, negedge sensitive, both posedge and negedge sensitive, high-voltage sensitive and low-voltage sensitive wakeups, with specific conditions based on the GPIO configuration.

Table 10.4-2. LP GPIO Wakeup Signal Trigger and Clear Conditions

LP_GPIO_PIN*x*_INT_TYPE[1,2]	Wakeup is generated when	Wakeup is cleared when
1	the input GPIO toggles from low-level to high-level.	LP_GPIO_PIN*x*_EDGE_WAKEUP_CLR is set.
2	the input GPIO toggles from high-level to low-level.	LP_GPIO_PIN*x*_EDGE_WAKEUP_CLR is set.
3	the input GPIO toggles from high-level to low-level, or vice versa.	LP_GPIO_PIN*x*_EDGE_WAKEUP_CLR is set.
4	the input GPIO is low-level.	the input GPIO is high-level.
5	the input GPIO is high-level.	the input GPIO is low-level.

[1] *x* ranges from 0 to 15.

[2] If the register is configured to any other value, LP GPIO wakeup feature is disabled.

10.4.7 Programming Procedure

10.4.7.1 HP GPIO Matrix

To read GPIO pin *X*[1] into HP peripheral signal *Y*, follow the steps below:

1. Configure GPIO_FUNC*y*_IN_SEL_CFG_REG corresponding to HP peripheral signal *Y* in HP GPIO matrix:
   - Set GPIO_SIG*y*_IN_SEL to enable peripheral signal input via HP GPIO matrix.
   - Set GPIO_FUNC*y*_IN_SEL to the desired GPIO pin, i.e., *X* here.

   Note that some peripheral signals have no valid GPIO_SIG*y*_IN_SEL bit, namely, these peripherals can only receive input signals via HP GPIO matrix.
2. Optionally enable the GPIO Filter for pin input signals by setting IO_MUX_GPIO*n*_FILTER_EN.
3. Enable Glitch Filter feature as follows:
   - Configure GPIO_EXT_FILTER_CH*n*_INPUT_IO_NUM to *m*. *n* (0 ~ 7) represents the channel number. *m* (0 ~ 54) represents the GPIO pin number.
   - Configure GPIO_EXT_FILTER_CH*n*_WINDOW_WIDTH to **VALUE1** and GPIO_EXT_FILTER_CH*n*_WINDOW_THRES to **VALUE2**. During **VALUE1** + 1 cycles, if there are **VALUE2** + 1 input signals that do not match the current output signal value, the Glitch Filter hardware inverts the output signal. GPIO_EXT_FILTER_CH*n*_WINDOW_WIDTH and GPIO_EXT_FILTER_CH*n*_WINDOW_THRES can be configured to the same value **VALUE3**, then only signals with a width greater than **VALUE3** + 1 clock cycles will be sampled.
   - Set GPIO_EXT_FILTER_CH*n*_EN to enable channel *n*.

   An example is shown in Figure 10.4-3, where GPIO_EXT_FILTER_CH*x*_WINDOW_WIDTH is configured to 3 and GPIO_EXT_FILTER_CH*x*_WINDOW_THRES to 2. The output signal value (signal_out) keeps as "0" in the four clock cycles before T1. The input signal value (signal_in) has been "1" for three clock cycles in the same period, then the output signal is inverted to "1" after T1.

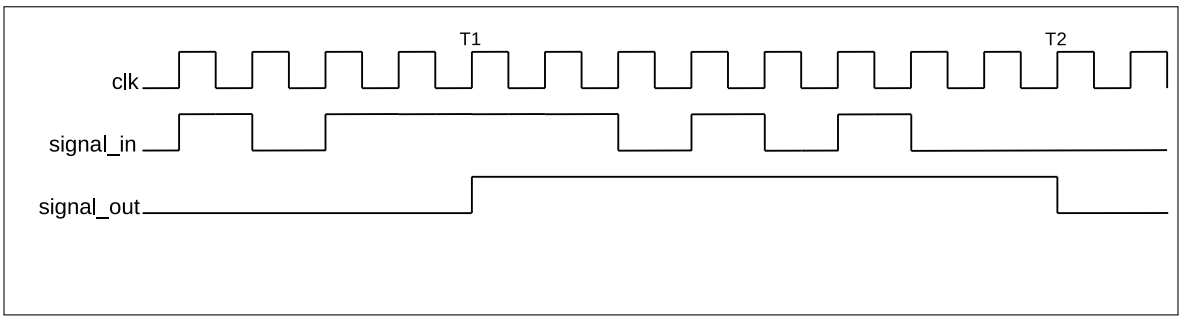


Figure 10.4-3. Glitch Filter Timing Example

4. Synchronize GPIO input signals. To do so, please set GPIO_PIN*x*_REG corresponding to GPIO pin *X* as follows:
   - Set GPIO_PIN*x*_SYNC1_BYPASS to enable input signal synchronized on rising edge or on falling edge in the first-level synchronization, see Figure 10.4-1.
   - Set GPIO_PIN*x*_SYNC2_BYPASS to enable input signal synchronized on rising edge or on falling edge in the second-level synchronization, see Figure 10.4-1.
5. Configure HP IO MUX register to enable pin input. For this end, please set IO_MUX_GPIO*x*_REG corresponding to GPIO pin *X* as follows:
   - Set IO_MUX_GPIO*x*_FUN_IE to enable input[2].
   - Set or clear IO_MUX_GPIO*x*_FUN_WPU and IO_MUX_GPIO*x*_FUN_WPD as desired to enable or disable pull-up and pull-down resistors.

For example, to connect UART0 TXD input signal[3] (uart0_rxd_pad_in, signal index 10) to GPIO7, please follow the steps below.

1. Set GPIO_SIG10_IN_SEL in GPIO_FUNC10_IN_SEL_CFG_REG to enable peripheral signal input via HP GPIO matrix.
2. Set GPIO_FUNC10_IN_SEL in GPIO_FUNC10_IN_SEL_CFG_REG to 7, i.e., select GPIO7.
3. Set IO_MUX_GPIO7_FUN_IE in IO_MUX_GPIO7_REG to enable pin input.

Note:

1. One input pin can be connected to multiple peripheral input signals.
2. The input signal can be inverted by configuring GPIO_FUNC*y*_IN_INV_SEL.
3. It is possible to have a HP peripheral read a constantly low or constantly high input value without connecting this input to a pin. This can be done by selecting a special GPIO_FUNC*y*_IN_SEL input, instead of a GPIO number:
   - When GPIO_FUNC*y*_IN_SEL is set to 62, input signal is always 0.
   - When GPIO_FUNC*y*_IN_SEL is set to 63, input signal is always 1.

10.4.7.2 LP GPIO Matrix

The programming procedure for peripheral input via the LP GPIO matrix resembles that of Section 10.4.7.1. However, it's worth noting that:

- the LP GPIO matrix lacks a Glitch Filter function.
- The control and status registers must use LP GPIO related registers. See Section 10.19.4 and Section 10.19.5.

Note:
It is possible to have a LP peripheral read a constantly low or constantly high input value without connecting this input to a pin. This can be done by selecting a special LP_GPIO_SIG*y*_IN_SEL input, instead of a GPIO number:

- When LP_GPIO_SIG*y*_IN_SEL is set to 16, input signal is always 0.
- When LP_GPIO_SIG*y*_IN_SEL is set to 24, input signal is always 1.

10.5 Peripheral Output via GPIO Matrix

10.5.1 Overview

To output a signal from a peripheral via GPIO matrix,

- configure the HP GPIO matrix to route HP peripheral output signals (only signals with a name assigned in the column "Output signal" in Table 10.12-1) to one of the 55 GPIOs (0 ~ 54).
- Or configure the LP GPIO matrix to route LP peripherals (only signals with a name assigned in the column "Output signal" in Table 10.13-1) to one of the 16 LP GPIOs (0 ~ 15).
- The output signal is routed from the peripheral into GPIO matrix and then into IO MUX. IO MUX must be configured to set the chosen pin to GPIO function. This enables the GPIO output signal to be connected to the pin.

Note:
There is a range of peripheral output signals (250 ~ 255 in Table 10.12-1) which are not connected to any peripheral, but to the input signals (250 ~ 255) directly.

10.5.2 Simple GPIO Output

GPIO matrix can also be used for simple GPIO output. For this case, one GPIO pin can be configured to directly output the desired value, without routing any peripheral output to this pin.

Follow the steps below to configure HP GPIO matrix for simple GPIO output:

- Set GPIO_FUNC*x*_OUT_SEL with a special peripheral index 256 (0x100).
- Configure the corresponding bit in GPIO_OUT_REG/GPIO_OUT1_REG to the desired GPIO output value.

Note:

- GPIO_OUT_REG[0] ~ GPIO_OUT_REG[31] correspond to GPIO0 ~ GPIO31, and GPIO_OUT1_REG[0] ~ GPIO_OUT1_REG[22] to GPIO32 ~ GPIO54.
- Recommended operation: use GPIO_OUT/OUT1_W1TS and GPIO_OUT/OUT1_W1TC to set or clear GPIO_OUT/OUT1_REG.

Follow the steps below to configure LP GPIO matrix for simple GPIO output:

- Set LP_GPIO_FUNC*x*_OUT_SEL with a special peripheral index 32 (0x20).
- Configure the corresponding bit in LP_GPIO_OUT_REG to the desired GPIO output value.

Note:
Recommended operation: use LP_GPIO_OUT_DATA_W1TS and LP_GPIO_OUT_DATA_W1TC to set or clear LP_GPIO_OUT_REG.

10.5.3 Sigma Delta Modulated Output (SDM)

10.5.3.1 Functional Description

Eight HP peripheral output signals (index: 72 ~ 79 in Table 10.12-1) support 1-bit second-order sigma delta modulation. By default the output is enabled for these eight channels. This Sigma Delta modulator can also output PDM (pulse density modulation) signal with configurable duty cycle. The function is:

$$H(z) = \mathrm{X(z)z^{-1} + E(z)(1\text{-}z^{-1})^2}$$

E(z) is quantization error and X(z) is the input.
This modulator supports scaling down of IO MUX operating clock by divider 1 ~ 256:

- Set GPIO_EXT_SD_FUNCTION_CLK_EN to enable the modulator clock.
- Configure GPIO_EXT_SD*n*_PRESCALE (*n* = 0 ~ 7 for the eight channels).

After scaling, the clock cycle is equal to one pulse output cycle from the modulator.

GPIO_EXT_SD*n*_IN is a signed number with a range of [-128, 127] and is used to control the duty cycle[1] of PDM output signal.

- GPIO_EXT_SD*n*_IN = -128, the duty cycle of the output signal is 0%.
- GPIO_EXT_SD*n*_IN = 0, the duty cycle of the output signal is near 50%.
- GPIO_EXT_SD*n*_IN = 127, the duty cycle of the output signal is near 100%.

The formula for calculating PDM signal duty cycle is shown as below:

$$Duty_Cycle = \frac{GPIO_EXT_SDn_IN + 128}{256}$$

Note:
For PDM signals, duty cycle refers to the percentage of high level cycles to the whole statistical period (several pulse cycles, for example, 256 pulse cycles).

10.5.3.2 SDM Configuration

The configuration of SDM is shown below:

- Route one of SDM outputs to a pin via HP GPIO matrix, see Section 10.5.4.1.

- Enable the modulator clock by setting GPIO_EXT_SD_FUNCTION_CLK_EN.
- Configure the divider value by setting GPIO_EXT_SD*n*_PRESCALE.
- Configure the duty cycle of SDM output signal by setting GPIO_EXT_SD*n*_IN.

10.5.4 Programming Procedure

10.5.4.1 HP GPIO Matrix

The output signals with a name assigned in the column "Output signal" in Table 10.12-1 can be set to go through the HP GPIO matrix into HP IO MUX and then to a pin. Figure 10.3-1 illustrates the configuration.

To output peripheral signal *Y* to a particular GPIO pin *X*, follow the steps below:

1. Configure GPIO_FUNC*x*_OUT_SEL_CFG_REG and GPIO_ENABLE_REG[*x*] corresponding to GPIO pin *X* in HP GPIO matrix. Recommended operation: use corresponding W1TS (write 1 to set) and W1TC (write 1 to clear) registers to set or clear GPIO_ENABLE_REG.
   - Set the GPIO_FUNC*x*_OUT_SEL field in register GPIO_FUNC*x*_OUT_SEL_CFG_REG to the index of the desired peripheral output signal *Y*.
   - If the signal should always be enabled as an output, set the GPIO_FUNC*x*_OE_SEL bit in register GPIO_FUNC*x*_OUT_SEL_CFG_REG and the bit in register GPIO_ENABLE_W1TS_REG, corresponding to GPIO pin *X*. To have the output enable signal decided by internal logic (for example, the spi2_dqs_pad_oe in column "Output enable signal when GPIO_FUNC*n*_OE_SEL = 0" in Table 10.12-1), clear the GPIO_FUNC*x*_OE_SEL bit instead.
   - Set the corresponding bit in register GPIO_ENABLE_W1TC_REG to disable the output from the GPIO pin.
2. For an open drain output, set the GPIO_PIN*x*_PAD_DRIVER bit in register GPIO_PIN*x*_REG corresponding to GPIO pin *X*.
3. Configure HP IO MUX register to enable output via HP GPIO matrix. Set IO_MUX_GPIO*x*_REG corresponding to GPIO pin *X* as follows:
   - Set the field IO_MUX_GPIO*x*_MCU_SEL to desired HP IO MUX function corresponding to GPIO pin *X*. This is Function 1 (GPIO function), numeric value 1, for all pins.
   - Set the IO_MUX_GPIO*x*_FUN_DRV field to the desired value for output strength (0 ~ 3). The higher the drive strength, the more current can be sourced/sunk from the pin.
   - If using open drain mode, set/clear the IO_MUX_GPIO*x*_FUN_WPU and IO_MUX_GPIO*x*_FUN_WPD bits to enable/disable the internal pull-up/pull-down resistors.
4. Enable hysteresis function:
   - GPIO0 ~ GPIO15: set LP_IOMUX_LP_GPIO_HYS[*x*] (*x*: 0 ~ 15, corresponding to GPIO0 ~ GPIO15).
   - GPIO16 ~ GPIO47: set HP_SYSTEM_GPIO_0_HYS_LOW[*x*] (*x*: 0 ~ 31, corresponding to GPIO16 ~ GPIO47).
   - GPIO48 ~ GPIO54: set HP_SYSTEM_GPIO_0_HYS_HIGH[*x*] (*x*: 0 ~ 6, corresponding to GPIO48 ~ GPIO54).

Note:

- The output signal from a single peripheral can be sent to multiple pins simultaneously.
- The output signal can be inverted by setting GPIO_FUNC*x*_OUT_INV_SEL.

10.5.4.2 LP GPIO Matrix

The programming procedure for peripheral input via the LP GPIO matrix resembles that of 10.5.4.1. However, it's worth noting that the control and status registers must use LP GPIO matrix registers. See Section 10.19.4 and Section 10.19.5.

10.6 Direct Input and Output via IO MUX

10.6.1 Overview

Some digital signals (SDMMC, EMAC, etc.) can bypass GPIO matrix for better high-frequency digital performance. In this case, IO MUX is used to connect these pins directly to peripherals. This option is less flexible than routing signals via GPIO matrix, as the IO MUX register for each GPIO pin can only select from a limited number of functions, but high-frequency digital performance can be improved.

ESP32-P4 provides 16 GPIO pins with low power (LP) capabilities. These pins can be controlled by either HP IO MUX or LP IO MUX. If controlled by LP IO MUX, these pins will bypass HP IO MUX and HP GPIO matrix for the use by peripherals in LP system.

When configured as LP GPIOs, the pins can still be controlled by the peripherals in LP system during chip Deep-sleep, and wake up the chip from Deep-sleep.

10.6.2 Functional Description

The pins with LP functions (GPIO0 ~ GPIO15) are controlled by LP_IOMUX_PAD*n*_MUX_SEL (*n*: 0 ~ 15) in register LP_IOMUX_PAD*n*_REG. By default, all bits in these registers are set to 0, routing all input/output signals via HP IO MUX.

If LP_IOMUX_PAD*n*_MUX_SEL is set, then input/output signals are controlled by LP IO MUX. In this mode, LP_IOMUX_PAD*n*_REG is used to control the LP GPIO pins. See 10.15-1 for the LP functions of each LP GPIO pin. Note that LP_IOMUX_PAD*n*_REG applies the LP GPIO pin numbering, not the HP GPIO pin numbering.

10.6.2.1 HP IO MUX

Two fields must be configured in order to bypass HP GPIO matrix for HP peripheral input signals:

1. IO_MUX_GPIO*n*_MCU_SEL for the GPIO pin must be set to the required pin function. For the list of pin functions, please refer to Table 10.14-1.
2. Clear GPIO_SIG*n*_IN_SEL to route the input directly to the peripheral.

To bypass HP GPIO matrix for HP peripheral output signals, IO_MUX_GPIO*n*_MCU_SEL for the GPIO pin must be set to the required pin function.

10.6.2.2 LP IO MUX

Two fields must be configured in order to bypass LP GPIO matrix for LP peripheral input signals:

1. LP_IOMUX_PAD*n*_MUX_SEL for the GPIO pin must be set to the required pin function. For the list of pin functions, please refer to Table 10.15-1.
2. Clear LP_GPIO_GPIO_SIG*n*_IN_SEL to route the input directly to the peripheral.

To bypass LP GPIO matrix for LP peripheral output signals, LP_IOMUX_PAD*n*_MUX_SEL for the GPIO pin must be set to the required pin function.

Note:
Not all signals can be directly connected to peripheral via IO MUX. Some input/output signals can only be connected to peripheral via GPIO matrix.

10.7 Analog Functions

10.7.1 Overview

ESP32-P4 provides 34 GPIO pins with analog functions, including 16 LP GPIO pins and 18 HP GPIO pins. See Table 10.16-1.

10.7.2 Analog Functions

When the pin is used for analog purpose, make sure this pin is left floating by configuring registers. By such way, the external analog signal is directly connected to internal analog signal via GPIO pin. The configuration is as follows:

- Clear IO_MUX_GPIO*n*_FUN_IE, IO_MUX_GPIO*n*_FUN_WPU, and IO_MUX_GPIO*n*_FUN_WPD, to set the pin floating.
- Write 1 to the corresponding bit in GPIO_ENABLE_W1TC and GPIO_ENABLE1_W1TC, to clear output enable.

To use these functions listed in Table 10.16-1, please refer to the following related chapters:

- For XTAL_32K related functions, see Chapter 15 *Low-Power Management*.
- For TOUCH related functions, see Chapter 63 *Touch Sensor (TOUCH)*.
- For ADC related functions, see Chapter 65 *ADC Controller (ADC)*.
- For COMP related functions, see Chapter 66 *Analog Voltage Comparator*.

Note:
GPIO51 ~ GPIO54 each has two different analog functions, which can be used simultaneously.

10.8 Pin Functions in Light-sleep

Pins may provide different functions when ESP32-P4 is in Light-sleep mode. If IO_MUX_GPIO*n*_SLP_SEL in register IO_MUX_GPIO*n*_REG for a GPIO pin is set to 1, a different set of bits will be used to control the pin when the chip is in Light-sleep mode.

Table 10.8-1. Bit Used to Control IO MUX Functions in Light-sleep Mode

IO MUX Function	Normal Execution OR IO_MUX_GPIO*n*_SLP_SEL = 0	Light-sleep Mode AND IO_MUX_GPIO*n*_SLP_SEL = 1
Output Drive Strength	IO_MUX_GPIO*n*_FUN_DRV	IO_MUX_GPIO*n*_MCU_DRV
Pull-up Resistor	IO_MUX_GPIO*n*_FUN_WPU	IO_MUX_GPIO*n*_MCU_WPU
Pull-down Resistor	IO_MUX_GPIO*n*_FUN_WPD	IO_MUX_GPIO*n*_MCU_WPD
Input Enable	IO_MUX_GPIO*n*_FUN_IE	IO_MUX_GPIO*n*_MCU_IE
Output Enable	OE_SEL from GPIO matrix*	IO_MUX_GPIO*n*_MCU_OE

Note:
If IO_MUX_GPIO*n*_SLP_SEL is set to 0, pin functions remain the same in both normal execution and in Light-sleep mode. Please refer to Section 10.5.4.1 for how to enable output in normal execution.

10.9 Pin Hold Feature

Each GPIO pin has an individual hold function controlled by Power Management Unit (PMU) or registers. When the pin is set to hold, the state is latched at that moment and will not change no matter how the internal signals change or how the IO MUX/GPIO configuration is modified. Users can use the hold function for the pins to retain the pin state through a core reset triggered by watchdog time-out or Deep-sleep events.

HP Pins (GPIO16 ~ GPIO54):

The Hold state of each HP pin is controlled by the result of OR operation of the pin's Hold enable signal and the global Hold enable signal.

- Each pin's Hold enable signal is controlled by the following registers:
  - In Deep-sleep mode:
    * LP_SYSTEM_REG_PAD_RTC_HOLD_CTRL0[*n*] (*n* = 16~31), controls the Hold signal of each pin of GPIO16~GPIO31.
    * LP_SYSTEM_REG_PAD_RTC_HOLD_CTRL1[*n*] (*n* = 0~22), controls the Hold signal of each pin of GPIO32~GPIO54.
  - In non-Deep-sleep mode:
    * HP_SYSTEM_GPIO_O_HOLD_LOW[*n*] (*n* = 0~31), controls the Hold signal of each pin of GPIO16~GPIO47.
    * HP_SYSTEM_GPIO_O_HOLD_HIGH[*n*](*n* = 0~6), controls the Hold signal of each pin of GPIO48~GPIO54.

> **Note:**
> In non-Deep-sleep mode, besides the above HP_SYSTEM registers, the LP_SYSTEM registers (LP_SYSTEM_REG_PAD_RTC_HOLD_CTRL0/1) can also control the Hold function for corresponding pins, with the same effect as the HP_SYSTEM registers.

- PMU_TIE_HIGH_HP_PAD_HOLD_ALL, controls the global Hold signal of all HP pins.

To use this feature, follow the steps below:

- To maintain the pin's input/output status in Deep-sleep, set LP_SYSTEM_REG_PAD_RTC_HOLD_CTRL0[*n*] (*n* = 16∼31) or LP_SYSTEM_REG_PAD_RTC_HOLD_CTRL1[*n*] (*n* = 0∼22) before powering down. To disable the hold function after waking up, clear both registers.
- To maintain the pin's input/output status in non-Deep-sleep mode, set HP_SYSTEM_GPIO_O_HOLD_LOW[*n*] (*n* = 0∼31) or HP_SYSTEM_GPIO_O_HOLD_HIGH[*n*] (*n* = 0∼6) before powering down. To disable the hold function after waking up, clear both registers. You can also use the LP_SYSTEM registers (LP_SYSTEM_REG_PAD_RTC_HOLD_CTRL0/1) as described above.
- Alternatively, set PMU_TIE_HIGH_HP_PAD_HOLD_ALL to maintain the input/output status of all HP pins and set PMU_TIE_LOW_HP_PAD_HOLD_ALL to disable the hold function for all HP pins.
- Alternatively, configure the PMU task to hold the HP pins before Deep-sleep. In Deep-sleep, the HP pins would be hold automatically by the PMU.

LP Pins (GPIO0 ∼ GPIO15):

The Hold state of each LP pin is controlled by the result of OR operation of the pin's Hold enable signal and the global Hold enable signal.

- Each pin's Hold enable signal is controlled by the following registers:
  - In Deep-sleep mode, LP_SYSTEM_REG_PAD_RTC_HOLD_CTRL0[*n*] (*n* = 0∼15) controls the Hold signal of each pin of GPIO0∼GPIO15.
  - In non-Deep-sleep mode, LP_IOMUX_LP_PAD_HOLD_REG[*n*] (*n* = 0∼15) controls the Hold signal of each pin of GPIO0∼GPIO15.

> **Note:**
> In non-Deep-sleep mode, besides the above LP_IOMUX register, the LP_SYSTEM register (LP_SYSTEM_REG_PAD_RTC_HOLD_CTRL0) can also control the Hold function for corresponding pins, with the same effect as the LP_IOMUX register.

- PMU_TIE_HIGH_LP_PAD_HOLD_ALL controls the global Hold signal of all LP pins.

To use this feature, follow the steps below:

- To maintain the pin's input/output status in Deep-sleep, set LP_SYSTEM_REG_PAD_RTC_HOLD_CTRL0[*n*] (*n* = 0∼15) to hold the value of GPIO*n*, or clear LP_SYSTEM_REG_PAD_RTC_HOLD_CTRL0[*n*] (*n* = 0∼15) to disable the hold function of GPIO*n*.
- To maintain the pin's input/output status in non-Deep-sleep mode, set LP_IOMUX_LP_PAD_HOLD_REG[*n*] (*n* = 0∼15) to hold the value of GPIO*n*, or clear

LP_IOMUX_LP_PAD_HOLD_REG[*n*] (*n* = 0~15) to disable the hold function of GPIO*n*. You can also use the LP_SYSTEM register (LP_SYSTEM_REG_PAD_RTC_HOLD_CTRL0) as described above.

- Alternatively, set PMU_TIE_HIGH_LP_PAD_HOLD_ALL to hold the values of all LP pins, and set PMU_TIE_LOW_LP_PAD_HOLD_ALL to disable the hold function of all LP pins.
- Alternatively, configure the PMU task to hold LP pins before Deep-sleep. In Deep-sleep, LP pins will be hold automatically by the PMU.

10.10 Hysteresis Characteristics of GPIO Pins

Each GPIO pin has hysteresis functionality. When hysteresis is not enabled, as shown in Figure 10.10-1, the level flip of the signal (C) input to the chip from the PAD has only one threshold (Vt, about 1.7 V). When the voltage on the PAD is higher than Vt, the level on the C is high. Otherwise, it is low. However, noise on the PAD may affect the signal on C.

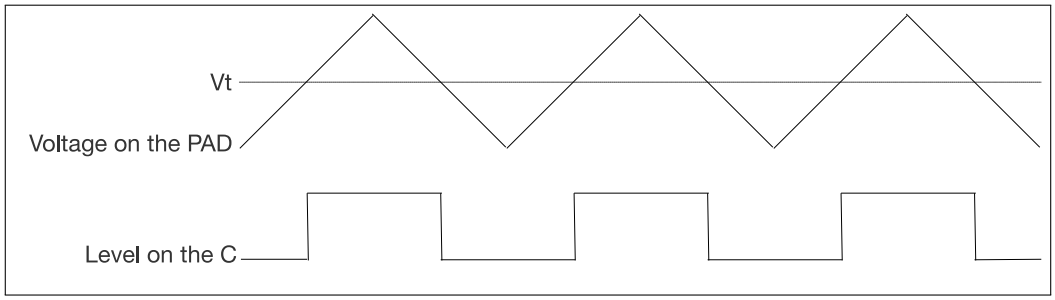


Figure 10.10-1. Example of Level Flip on the Chip Pad — Hysteresis Function Not Enabled

When hysteresis is enabled, as shown in Figure 10.10-2, the level flip of C has two thresholds, high-level threshold (Vth, about 1.7 V) and low-level threshold (Vtl, about 1.4 V). When the voltage of pad goes from low to high, if the voltage is higher than Vth, the level of C is high. When the voltage of PAD goes from high to low, if the voltage is lower than Vtl, the level of C is low. When the voltage of PAD is between Vth and Vtl, the level of C does not change. The hysteresis function plays an anti-interference role by mitigating the impact of noise, consequently reducing the level flip time of C.

To enable the hysteresis function, follow the steps below:

- Set LP_IOMUX_LP_GPIO_HYS[*n*] (*n* ranges from 0 ~ 15, corresponding to GPIO0 ~ GPIO15), HP_SYS_GPIO0_HYS_LOW[*n*] (*n* ranges from 0 ~ 31, corresponding to GPIO16 ~ GPIO47) or HP_SYS_GPIO0_HYS_HIGH[*n*] (*n* ranges from 0 ~ 6, corresponding to GPIO48 ~ GPIO54) to 1 to enable the hysteresis function for GPIO*n*.
- Set LP_IOMUX_LP_GPIO_HYS[*n*] (*n* ranges from 0 ~ 15, corresponding to GPIO0 ~ GPIO15), HP_SYS_GPIO0_HYS_LOW[*n*] (*n* ranges from 0 ~ 31, corresponding to GPIO16 ~ GPIO47) or HP_SYS_GPIO0_HYS_HIGH[*n*] (*n* ranges from 0 ~ 6, corresponding to GPIO48 ~ GPIO54) to 0 to disable the hysteresis function for GPIO*n*.

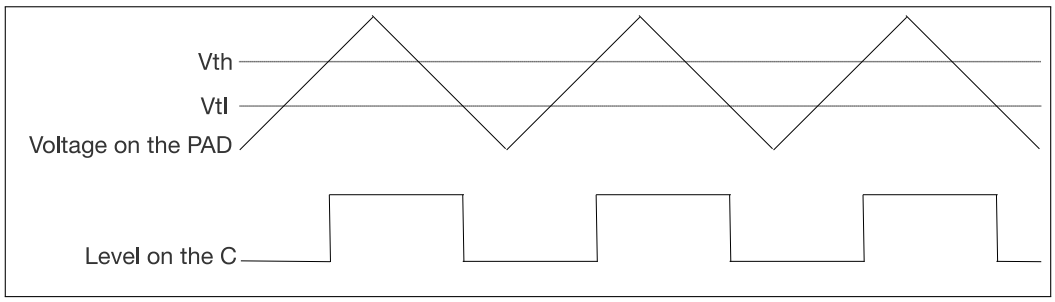


Figure 10.10-2. Example of Level Flip on the Chip Pad — Hysteresis Function Enabled

10.11 Power Supplies and Management of GPIO Pins

10.11.1 Power Supplies of GPIO Pins

For more information on the power supply for GPIO pins, please refer to Pin Definition in *ESP32-P4 Datasheet*. All the pins can be used to wake up the chip from Light-sleep, but only the pins (GPIO0 ~ GPIO15) in VDD_LP domain can be used to wake up the chip from Deep-sleep.

10.11.2 Power Supply Management

Each ESP32-P4 pin is connected to one of the following power domains.

- VDD_LP: the input power supply for LP GPIO pins
- VDD_IO_0, VDD_FLASHIO, VDD_IO_4 ~ VDD_IO_6: the input power supply for HP GPIO pins

10.12 HP Peripheral Signal List

Table 10.12-1 shows the peripheral input/output signals via HP GPIO matrix.

Please pay attention to the configuration of the bit GPIO_FUNC*n*_OE_SEL:

- GPIO_FUNC*n*_OE_SEL = 1: the output enable is controlled by the corresponding bit *n* of GPIO_ENABLE/ENABLE1_REG:
  - GPIO_ENABLE/ENABLE1_REG = 0: output is disabled.
  - GPIO_ENABLE/ENABLE1_REG = 1: output is enabled.
- GPIO_FUNC*n*_OE_SEL = 0: use the output enable signal from peripheral, for example spi2_dqs_pad_oe in the column "Output enable signal when GPIO_FUNC*n*_OE_SEL = 0" of Table 10.12-1. Note that the signals such as spi2_dqs_pad_oe can be 1 (1'd1) or 0 (1'd0), depending on the configuration of corresponding peripherals. If it's 1'd1 in column "Output enable signal when GPIO_FUNC*n*_OE_SEL = 0", it indicates that once GPIO_FUNC*n*_OE_SEL is cleared, the output signal is always enabled by default.

Note:

Signals are numbered consecutively, but not all signals are valid.

- Only the signals with a name assigned in the column "Input signal" in Table 10.12-1 are valid input signals.
- Only the signals with a name assigned in the column "Output signal" in Table 10.12-1 are valid output signals.

Table 10.12-1. Peripheral Signals via HP GPIO Matrix

Signal No.	Input Signal	Default Value	Direct Input via HP IO MUX	Output Signal	Output Enable Signal when GPIO_FUNCn_OE_SEL = 0	Direct Output via HP IO MUX
0	-	-	-	sd_card_cclk_2_pad_out	1'd1	no
1	sd_card_ccmd_2_pad_in	1	no	sd_card_ccmd_2_pad_out	sd_card_ccmd_2_pad_oe	no
2	sd_card_cdata0_2_pad_in	1	no	sd_card_cdata0_2_pad_out	sd_card_cdata0_2_pad_oe	no
3	sd_card_cdata1_2_pad_in	1	no	sd_card_cdata1_2_pad_out	sd_card_cdata1_2_pad_oe	no
4	sd_card_cdata2_2_pad_in	1	no	sd_card_cdata2_2_pad_out	sd_card_cdata2_2_pad_oe	no
5	sd_card_cdata3_2_pad_in	1	no	sd_card_cdata3_2_pad_out	sd_card_cdata3_2_pad_oe	no
6	sd_card_cdata4_2_pad_in	1	no	sd_card_cdata4_2_pad_out	sd_card_cdata4_2_pad_oe	no
7	sd_card_cdata5_2_pad_in	1	no	sd_card_cdata5_2_pad_out	sd_card_cdata5_2_pad_oe	no
8	sd_card_cdata6_2_pad_in	1	no	sd_card_cdata6_2_pad_out	sd_card_cdata6_2_pad_oe	no
9	sd_card_cdata7_2_pad_in	1	no	sd_card_cdata7_2_pad_out	sd_card_cdata7_2_pad_oe	no
10	uart0_rxd_pad_in	0	yes	uart0_txd_pad_out	1'd1	yes
11	uart0_cts_pad_in	0	yes	uart0_rts_pad_out	1'd1	yes
12	uart0_dsr_pad_in	0	no	uart0_dtr_pad_out	1'd1	no
13	uart1_rxd_pad_in	0	yes	uart1_txd_pad_out	1'd1	yes
14	uart1_cts_pad_in	0	yes	uart1_rts_pad_out	1'd1	yes
15	uart1_dsr_pad_in	0	no	uart1_dtr_pad_out	1'd1	no
16	uart2_rxd_pad_in	0	no	uart2_txd_pad_out	1'd1	no
17	uart2_cts_pad_in	0	no	uart2_rts_pad_out	1'd1	no
18	uart2_dsr_pad_in	0	no	uart2_dtr_pad_out	1'd1	no
19	uart3_rxd_pad_in	0	no	uart3_txd_pad_out	1'd1	no
20	uart3_cts_pad_in	0	no	uart3_rts_pad_out	1'd1	no
21	uart3_dsr_pad_in	0	no	uart3_dtr_pad_out	1'd1	no
22	uart4_rxd_pad_in	0	no	uart4_txd_pad_out	1'd1	no
23	uart4_cts_pad_in	0	no	uart4_rts_pad_out	1'd1	no
24	uart4_dsr_pad_in	0	no	uart4_dtr_pad_out	1'd1	no
25	i2s0_o_bck_pad_in	0	no	i2s0_o_bck_pad_out	1'd1	no
26	i2s0_mclk_pad_in	0	no	i2s0_mclk_pad_out	1'd1	no

Signal No.	Input Signal	Default Value	Direct Input via HP IO MUX	Output Signal	Output Enable Signal when GPIO_FUNCn_OE_SEL = 0	Direct Output via HP IO MUX
27	i2s0_o_ws_pad_in	0	no	i2s0_o_ws_pad_out	1'd1	no
28	i2s0_i_sd_pad_in	0	no	i2s0_o_sd_pad_out	1'd1	no
29	i2s0_i_bck_pad_in	0	no	i2s0_i_bck_pad_out	1'd1	no
30	i2s0_i_ws_pad_in	0	no	i2s0_i_ws_pad_out	1'd1	no
31	i2s1_o_bck_pad_in	0	no	i2s1_o_bck_pad_out	1'd1	no
32	i2s1_mclk_pad_in	0	no	i2s1_mclk_pad_out	1'd1	no
33	i2s1_o_ws_pad_in	0	no	i2s1_o_ws_pad_out	1'd1	no
34	i2s1_i_sd_pad_in	0	no	i2s1_o_sd_pad_out	1'd1	no
35	i2s1_i_bck_pad_in	0	no	i2s1_i_bck_pad_out	1'd1	no
36	i2s1_i_ws_pad_in	0	no	i2s1_i_ws_pad_out	1'd1	no
37	i2s2_o_bck_pad_in	0	no	i2s2_o_bck_pad_out	1'd1	no
38	i2s2_mclk_pad_in	0	no	i2s2_mclk_pad_out	1'd1	no
39	i2s2_o_ws_pad_in	0	no	i2s2_o_ws_pad_out	1'd1	no
40	i2s2_i_sd_pad_in	0	no	i2s2_o_sd_pad_out	1'd1	no
41	i2s2_i_bck_pad_in	0	no	i2s2_i_bck_pad_out	1'd1	no
42	i2s2_i_ws_pad_in	0	no	i2s2_i_ws_pad_out	1'd1	no
43	i2s0_i_sd1_pad_in	0	no	i2s0_o_sd1_pad_out	1'd1	no
44	i2s0_i_sd2_pad_in	0	no	spi2_dqs_pad_out	spi2_dqs_pad_oe	yes
45	i2s0_i_sd3_pad_in	0	no	spi3_cs2_pad_out	spi3_cs2_pad_oe	no
46	-	-	-	spi3_cs1_pad_out	spi3_cs1_pad_oe	no
47	spi3_ck_pad_in	0	no	spi3_ck_pad_out	spi3_ck_pad_oe	no
48	spi3_q_pad_in	0	no	spi3_qo_pad_out	spi3_q_pad_oe	no
49	spi3_d_pad_in	0	no	spi3_d_pad_out	spi3_d_pad_oe	no
50	spi3_hold_pad_in	0	no	spi3_hold_pad_out	spi3_hold_pad_oe	no
51	spi3_wp_pad_in	0	no	spi3_wp_pad_out	spi3_wp_pad_oe	no
52	spi3_cs_pad_in	0	no	spi3_cs_pad_out	spi3_cs_pad_oe	no
53	spi2_ck_pad_in	0	yes	spi2_ck_pad_out	spi2_ck_pad_oe	yes
54	spi2_q_pad_in	0	yes	spi2_q_pad_out	spi2_q_pad_oe	yes

GoBack

Signal No.	Input Signal	Default Value	Direct Input via HP IO MUX	Output Signal	Output Enable Signal when GPIO_FUNCn_OE_SEL = 0	Direct Output via HP IO MUX
55	spi2_d_pad_in	0	yes	spi2_d_pad_out	spi2_d_pad_oe	yes
56	spi2_hold_pad_in	0	yes	spi2_hold_pad_out	spi2_hold_pad_oe	yes
57	spi2_wp_pad_in	0	yes	spi2_wp_pad_out	spi2_wp_pad_oe	yes
58	spi2_io4_pad_in	0	yes	spi2_io4_pad_out	spi2_io4_pad_oe	yes
59	spi2_io5_pad_in	0	yes	spi2_io5_pad_out	spi2_io5_pad_oe	yes
60	spi2_io6_pad_in	0	yes	spi2_io6_pad_out	spi2_io6_pad_oe	yes
61	spi2_io7_pad_in	0	yes	spi2_io7_pad_out	spi2_io7_pad_oe	yes
62	spi2_cs_pad_in	0	yes	spi2_cs_pad_out	spi2_cs_pad_oe	yes
63	pcnt_rst_pad_in0	0	no	spi2_cs1_pad_out	spi2_cs1_pad_oe	no
64	pcnt_rst_pad_in1	0	no	spi2_cs2_pad_out	spi2_cs2_pad_oe	no
65	pcnt_rst_pad_in2	0	no	spi2_cs3_pad_out	spi2_cs3_pad_oe	no
66	pcnt_rst_pad_in3	0	no	spi2_cs4_pad_out	spi2_cs4_pad_oe	no
67	-	-	-	spi2_cs5_pad_out	spi2_cs5_pad_oe	no
68	i2c0_scl_pad_in	1	no	i2c0_scl_pad_out	i2c0_scl_oe_pad_out	no
69	i2c0_sda_pad_in	1	no	i2c0_sda_pad_out	i2c0_sda_oe_pad_out	no
70	i2c1_scl_pad_in	1	no	i2c1_scl_pad_out	i2c1_scl_oe_pad_out	no
71	i2c1_sda_pad_in	1	no	i2c1_sda_pad_out	i2c1_sda_oe_pad_out	no
72	-	-	-	gpio_sd0_out	1'd1	no
73	-	-	-	gpio_sd1_out	1'd1	no
74	uart0_slp_clk_pad_in	0	no	gpio_sd2_out	1'd1	no
75	uart1_slp_clk_pad_in	0	no	gpio_sd3_out	1'd1	no
76	uart2_slp_clk_pad_in	0	no	gpio_sd4_out	1'd1	no
77	uart3_slp_clk_pad_in	0	no	gpio_sd5_out	1'd1	no
78	uart4_slp_clk_pad_in	0	no	gpio_sd6_out	1'd1	no
79	-	-	-	gpio_sd7_out	1'd1	no
80	twai0_rx_pad_in	1	no	twai0_tx_pad_out	1'd1	no
81	-	-	-	twai0_bus_off_on_pad_out	1'd1	no
82	-	-	-	twai0_clkout_pad_out	1'd1	no

Signal No.	Input Signal	Default Value	Direct Input via HP IO MUX	Output Signal	Output Enable Signal when GPIO_FUNCn_OE_SEL = 0	Direct Output via HP IO MUX
83	twai1_rx_pad_in	1	no	twai1_tx_pad_out	1'd1	no
84	-	-	-	twai1_bus_off_on_pad_out	1'd1	no
85	-	-	-	twai1_clkout_pad_out	1'd1	no
86	twai2_rx_pad_in	1	no	twai2_tx_pad_out	1'd1	no
87	-	-	-	twai2_bus_off_on_pad_out	1'd1	no
88	-	-	-	twai2_clkout_pad_out	1'd1	no
89	pwm0_sync0_pad_in	0	no	pwm0_ch0_a_pad_out	1'd1	no
90	pwm0_sync1_pad_in	0	no	pwm0_ch0_b_pad_out	1'd1	no
91	pwm0_sync2_pad_in	0	no	pwm0_ch1_a_pad_out	1'd1	no
92	pwm0_f0_pad_in	0	no	pwm0_ch1_b_pad_out	1'd1	no
93	pwm0_f1_pad_in	0	no	pwm0_ch2_a_pad_out	1'd1	no
94	pwm0_f2_pad_in	0	no	pwm0_ch2_b_pad_out	1'd1	no
95	pwm0_cap0_pad_in	0	no	pwm1_ch0_a_pad_out	1'd1	no
96	pwm0_cap1_pad_in	0	no	pwm1_ch0_b_pad_out	1'd1	no
97	pwm0_cap2_pad_in	0	no	pwm1_ch1_a_pad_out	1'd1	no
98	pwm1_sync0_pad_in	0	no	pwm1_ch1_b_pad_out	1'd1	no
99	pwm1_sync1_pad_in	0	no	pwm1_ch2_a_pad_out	1'd1	no
100	pwm1_sync2_pad_in	0	no	pwm1_ch2_b_pad_out	1'd1	no
101	pwm1_f0_pad_in	0	no	-	-	-
102	pwm1_f1_pad_in	0	no	-	-	-
103	pwm1_f2_pad_in	0	no	-	-	-
104	pwm1_cap0_pad_in	0	no	-	-	-
105	pwm1_cap1_pad_in	0	no	twai0_standby_pad_out	1'd1	no
106	pwm1_cap2_pad_in	0	no	twai1_standby_pad_out	1'd1	no
107	gmii_mdi_pad_in	0	no	twai2_standby_pad_out	1'd1	no
108	gmac_phy_col_pad_in	0	no	gmii_mdc_pad_out	1'd1	no
109	gmac_phy_crs_pad_in	0	no	gmii_mdo_pad_out	gmii_mdo_oe_pad_out	no
110	usb_otg11_iddig_pad_in	0	no	usb_srp_dischrgvbus_pad_out	1'd1	no

Signal No.	Input Signal	Default Value	Direct Input via HP IO MUX	Output Signal	Output Enable Signal when GPIO_FUNCn_OE_SEL = 0	Direct Output via HP IO MUX
111	usb_otg11_avalid_pad_in	0	no	usb_otg11_idpullup_pad_out	1'd1	no
112	usb_srp_bvalid_pad_in	0	no	usb_otg11_dppulldown_pad_out	1'd1	no
113	usb_otg11_vbusvalid_pad_in	0	no	usb_otg11_dmpulldown_pad_out	1'd1	no
114	usb_srp_sessend_pad_in	0	no	usb_otg11_drvvbus_pad_out	1'd1	no
115	-	-	-	usb_srp_chrgvbus_pad_out	1'd1	no
116	-	-	-	-	-	-
117	ulpi_clk_pad_in	0	no	-	-	-
118	usb_hsphy_refclk_in	0	no	-	-	-
119	-	-	-	-	-	-
120	-	-	-	-	-	-
121	-	-	-	-	-	-
122	-	-	-	-	-	-
123	-	-	-	-	-	-
124	-	-	-	-	-	-
125	-	-	-	-	-	-
126	sd_card_detect_n_1_pad_in	0	no	ledc_ls_sig_out_pad_out0	1'd1	no
127	sd_card_detect_n_2_pad_in	0	no	ledc_ls_sig_out_pad_out1	1'd1	no
128	sd_card_int_n_1_pad_in	1	no	ledc_ls_sig_out_pad_out2	1'd1	no
129	sd_card_int_n_2_pad_in	1	no	ledc_ls_sig_out_pad_out3	1'd1	no
130	sd_card_write_prt_1_pad_in	0	no	ledc_ls_sig_out_pad_out4	1'd1	no
131	sd_card_write_prt_2_pad_in	0	no	ledc_ls_sig_out_pad_out5	1'd1	no
132	sd_data_strobe_1_pad_in	0	no	ledc_ls_sig_out_pad_out6	1'd1	no
133	sd_data_strobe_2_pad_in	0	no	ledc_ls_sig_out_pad_out7	1'd1	no
134	i3c_mst_scl_pad_in	1	no	i3c_mst_scl_pad_out	i3c_mst_scl_oe_pad_out	no
135	i3c_mst_sda_pad_in	1	no	i3c_mst_sda_pad_out	i3c_mst_sda_oe_pad_out	no
136	i3c_slv_scl_pad_in	1	no	i3c_slv_scl_pad_out	i3c_slv_scl_oe_pad_out	no
137	i3c_slv_sda_pad_in	1	no	i3c_slv_sda_pad_out	i3c_slv_sda_oe_pad_out	no

GoBack

Signal No.	Input Signal	Default Value	Direct Input via HP IO MUX	Output Signal	Output Enable Signal when GPIO_FUNCn_OE_SEL = 0	Direct Output via HP IO MUX
138	-	-	-	i3c_mst_scl_pullup_en_pad_out	1'd1	no
139	-	-	-	i3c_mst_sda_pullup_en_pad_out	1'd1	no
140	usb_jtag_tdo_bridge_pad_in	0	no	usb_jtag_tdi_bridge_pad_out	1'd1	no
141	pcnt_sig_ch0_pad_in0	0	no	usb_jtag_tms_bridge_pad_out	1'd1	no
142	pcnt_sig_ch0_pad_in1	0	no	usb_jtag_tck_bridge_pad_out	1'd1	no
143	pcnt_sig_ch0_pad_in2	0	no	usb_jtag_trst_bridge_pad_out	1'd1	no
144	pcnt_sig_ch0_pad_in3	0	no	lcd_cs_pad_out	1'd1	no
145	pcnt_sig_ch1_pad_in0	0	no	lcd_dc_pad_out	1'd1	no
146	pcnt_sig_ch1_pad_in1	0	no	sd_rst_n_1_pad_out	1'd1	no
147	pcnt_sig_ch1_pad_in2	0	no	sd_rst_n_2_pad_out	1'd1	no
148	pcnt_sig_ch1_pad_in3	0	no	sd_ccmd_od_pullup_en_n_pad_out	1'd1	no
149	pcnt_ctrl_ch0_pad_in0	0	no	lcd_pclk_pad_out	1'd1	no
150	pcnt_ctrl_ch0_pad_in1	0	no	cam_clk_pad_out	1'd1	no
151	pcnt_ctrl_ch0_pad_in2	0	no	lcd_h_enable_pad_out	1'd1	no
152	pcnt_ctrl_ch0_pad_in3	0	no	lcd_h_sync_pad_out	1'd1	no
153	pcnt_ctrl_ch1_pad_in0	0	no	lcd_v_sync_pad_out	1'd1	no
154	pcnt_ctrl_ch1_pad_in1	0	no	lcd_data_out_pad_out0	1'd1	no
155	pcnt_ctrl_ch1_pad_in2	0	no	lcd_data_out_pad_out1	1'd1	no
156	pcnt_ctrl_ch1_pad_in3	0	no	lcd_data_out_pad_out2	1'd1	no
157	-	-	-	lcd_data_out_pad_out3	1'd1	no
158	cam_pclk_pad_in	0	no	lcd_data_out_pad_out4	1'd1	no
159	cam_h_enable_pad_in	0	no	lcd_data_out_pad_out5	1'd1	no
160	cam_h_sync_pad_in	0	no	lcd_data_out_pad_out6	1'd1	no
161	cam_v_sync_pad_in	0	no	lcd_data_out_pad_out7	1'd1	no
162	cam_data_in_pad_in0	0	no	lcd_data_out_pad_out8	1'd1	no
163	cam_data_in_pad_in1	0	no	lcd_data_out_pad_out9	1'd1	no

GoBack

Signal No.	Input Signal	Default Value	Direct Input via HP IO MUX	Output Signal	Output Enable Signal when GPIO_FUNCn_OE_SEL = 0	Direct Output via HP IO MUX
164	cam_data_in_pad_in2	0	no	lcd_data_out_pad_out10	1'd1	no
165	cam_data_in_pad_in3	0	no	lcd_data_out_pad_out11	1'd1	no
166	cam_data_in_pad_in4	0	no	lcd_data_out_pad_out12	1'd1	no
167	cam_data_in_pad_in5	0	no	lcd_data_out_pad_out13	1'd1	no
168	cam_data_in_pad_in6	0	no	lcd_data_out_pad_out14	1'd1	no
169	cam_data_in_pad_in7	0	no	lcd_data_out_pad_out15	1'd1	no
170	cam_data_in_pad_in8	0	no	lcd_data_out_pad_out16	1'd1	no
171	cam_data_in_pad_in9	0	no	lcd_data_out_pad_out17	1'd1	no
172	cam_data_in_pad_in10	0	no	lcd_data_out_pad_out18	1'd1	no
173	cam_data_in_pad_in11	0	no	lcd_data_out_pad_out19	1'd1	no
174	cam_data_in_pad_in12	0	no	lcd_data_out_pad_out20	1'd1	no
175	cam_data_in_pad_in13	0	no	lcd_data_out_pad_out21	1'd1	no
176	cam_data_in_pad_in14	0	no	lcd_data_out_pad_out22	1'd1	no
177	cam_data_in_pad_in15	0	no	lcd_data_out_pad_out23	1'd1	no
178	gmac_phy_rxdv_pad_in	0	yes	gmac_phy_txen_pad_out	1'd1	yes
179	gmac_phy_rxd0_pad_in	0	yes	gmac_phy_txd0_pad_out	1'd1	yes
180	gmac_phy_rxd1_pad_in	0	yes	gmac_phy_txd1_pad_out	1'd1	yes
181	gmac_phy_rxd2_pad_in	0	no	gmac_phy_txd2_pad_out	1'd1	no
182	gmac_phy_rxd3_pad_in	0	no	gmac_phy_txd3_pad_out	1'd1	no
183	gmac_phy_rxer_pad_in	0	yes	gmac_phy_txer_pad_out	1'd1	yes
184	gmac_rx_clk_pad_in	0	no	-	-	-
185	gmac_tx_clk_pad_in	0	no	-	-	-
186	parlio_rx_clk_pad_in	0	no	parlio_rx_clk_pad_out	1'd1	no
187	parlio_tx_clk_pad_in	0	no	parlio_tx_clk_pad_out	1'd1	no
188	parlio_rx_data0_pad_in	0	no	parlio_tx_data0_pad_out	1'd1	no
189	parlio_rx_data1_pad_in	0	no	parlio_tx_data1_pad_out	1'd1	no
190	parlio_rx_data2_pad_in	0	no	parlio_tx_data2_pad_out	1'd1	no
191	parlio_rx_data3_pad_in	0	no	parlio_tx_data3_pad_out	1'd1	no

Signal No.	Input Signal	Default Value	Direct Input via HP IO MUX	Output Signal	Output Enable Signal when GPIO_FUNCn_OE_SEL = 0	Direct Output via HP IO MUX
192	parlio_rx_data4_pad_in	0	no	parlio_tx_data4_pad_out	1'd1	no
193	parlio_rx_data5_pad_in	0	no	parlio_tx_data5_pad_out	1'd1	no
194	parlio_rx_data6_pad_in	0	no	parlio_tx_data6_pad_out	1'd1	no
195	parlio_rx_data7_pad_in	0	no	parlio_tx_data7_pad_out	1'd1	no
196	parlio_rx_data8_pad_in	0	no	parlio_tx_data8_pad_out	1'd1	no
197	parlio_rx_data9_pad_in	0	no	parlio_tx_data9_pad_out	1'd1	no
198	parlio_rx_data10_pad_in	0	no	parlio_tx_data10_pad_out	1'd1	no
199	parlio_rx_data11_pad_in	0	no	parlio_tx_data11_pad_out	1'd1	no
200	parlio_rx_data12_pad_in	0	no	parlio_tx_data12_pad_out	1'd1	no
201	parlio_rx_data13_pad_in	0	no	parlio_tx_data13_pad_out	1'd1	no
202	parlio_rx_data14_pad_in	0	no	parlio_tx_data14_pad_out	1'd1	no
203	parlio_rx_data15_pad_in	0	no	parlio_tx_data15_pad_out	1'd1	no
204	-	-	-	-	-	-
205	-	-	-	-	-	-
206	-	-	-	-	-	-
207	-	-	-	-	-	-
208	-	-	-	-	-	-
209	-	-	-	-	-	-
210	-	-	-	-	-	-
211	-	-	-	-	-	-
212	-	-	-	constant0_pad_out	1'd1	no
213	-	-	-	constant1_pad_out	1'd1	no
214	core_gpio_in_pad_in0	0	no	core_gpio_out_pad_out0	core_gpio_oe_pad_out0	no
215	core_gpio_in_pad_in1	0	no	core_gpio_out_pad_out1	core_gpio_oe_pad_out1	no
216	core_gpio_in_pad_in2	0	no	core_gpio_out_pad_out2	core_gpio_oe_pad_out2	no
217	core_gpio_in_pad_in3	0	no	core_gpio_out_pad_out3	core_gpio_oe_pad_out3	no
218	core_gpio_in_pad_in4	0	no	core_gpio_out_pad_out4	core_gpio_oe_pad_out4	no
219	core_gpio_in_pad_in5	0	no	core_gpio_out_pad_out5	core_gpio_oe_pad_out5	no

Signal No.	Input Signal	Default Value	Direct Input via HP IO MUX	Output Signal	Output Enable Signal when GPIO_FUNCn_OE_SEL = 0	Direct Output via HP IO MUX
220	core_gpio_in_pad_in6	0	no	core_gpio_out_pad_out6	core_gpio_oe_pad_out6	no
221	core_gpio_in_pad_in7	0	no	core_gpio_out_pad_out7	core_gpio_oe_pad_out7	no
222	core_gpio_in_pad_in8	0	no	core_gpio_out_pad_out8	core_gpio_oe_pad_out8	no
223	core_gpio_in_pad_in9	0	no	core_gpio_out_pad_out9	core_gpio_oe_pad_out9	no
224	core_gpio_in_pad_in10	0	no	core_gpio_out_pad_out10	core_gpio_oe_pad_out10	no
225	core_gpio_in_pad_in11	0	no	core_gpio_out_pad_out11	core_gpio_oe_pad_out11	no
226	core_gpio_in_pad_in12	0	no	core_gpio_out_pad_out12	core_gpio_oe_pad_out12	no
227	core_gpio_in_pad_in13	0	no	core_gpio_out_pad_out13	core_gpio_oe_pad_out13	no
228	core_gpio_in_pad_in14	0	no	core_gpio_out_pad_out14	core_gpio_oe_pad_out14	no
229	core_gpio_in_pad_in15	0	no	core_gpio_out_pad_out15	core_gpio_oe_pad_out15	no
230	core_gpio_in_pad_in16	0	no	core_gpio_out_pad_out16	core_gpio_oe_pad_out16	no
231	core_gpio_in_pad_in17	0	no	core_gpio_out_pad_out17	core_gpio_oe_pad_out17	no
232	core_gpio_in_pad_in18	0	no	core_gpio_out_pad_out18	core_gpio_oe_pad_out18	no
233	core_gpio_in_pad_in19	0	no	core_gpio_out_pad_out19	core_gpio_oe_pad_out19	no
234	core_gpio_in_pad_in20	0	no	core_gpio_out_pad_out20	core_gpio_oe_pad_out20	no
235	core_gpio_in_pad_in21	0	no	core_gpio_out_pad_out21	core_gpio_oe_pad_out21	no
236	core_gpio_in_pad_in22	0	no	core_gpio_out_pad_out22	core_gpio_oe_pad_out22	no
237	core_gpio_in_pad_in23	0	no	core_gpio_out_pad_out23	core_gpio_oe_pad_out23	no
238	core_gpio_in_pad_in24	0	no	core_gpio_out_pad_out24	core_gpio_oe_pad_out24	no
239	core_gpio_in_pad_in25	0	no	core_gpio_out_pad_out25	core_gpio_oe_pad_out25	no
240	core_gpio_in_pad_in26	0	no	core_gpio_out_pad_out26	core_gpio_oe_pad_out26	no
241	core_gpio_in_pad_in27	0	no	core_gpio_out_pad_out27	core_gpio_oe_pad_out27	no
242	core_gpio_in_pad_in28	0	no	parlio_tx_cs_pad_out	1'd1	no
243	core_gpio_in_pad_in29	0	no	emac_ptp_pps_pad_out	1'd1	no
244	core_gpio_in_pad_in30	0	no	ana_comp0_out	1'd1	no
245	core_gpio_in_pad_in31	0	no	ana_comp1_out	1'd1	no
246	rmt_sig_pad_in0	0	no	rmt_sig_pad_out0	1'd1	no
247	rmt_sig_pad_in1	0	no	rmt_sig_pad_out1	1'd1	no

Signal No.	Input Signal	Default Value	Direct Input via HP IO MUX	Output Signal	Output Enable Signal when GPIO_FUNCn_OE_SEL = 0	Direct Output via HP IO MUX
248	rmt_sig_pad_in2	0	no	rmt_sig_pad_out2	1'd1	no
249	rmt_sig_pad_in3	0	no	rmt_sig_pad_out3	1'd1	no
250	sig_in_func250	0	no	sig_in_func250	1'd1	no
251	sig_in_func251	0	no	sig_in_func251	1'd1	no
252	sig_in_func252	0	no	sig_in_func252	1'd1	no
253	sig_in_func253	0	no	sig_in_func253	1'd1	no
254	sig_in_func254	0	no	sig_in_func254	1'd1	no
255	sig_in_func255	0	no	sig_in_func255	1'd1	no

10.13 LP Peripheral Signal List

Table 10.13-1 shows the peripheral input/output signals via LP GPIO matrix.

Please pay attention to the configuration of the bit LP_GPIO_FUNC*n*_OE_SEL:

- LP_GPIO_FUNC*n*_OE_SEL = 1: the output enable is controlled by the corresponding bit*n* of LP_GPIO_ENABLE_REG:
  - LP_GPIO_ENABLE_REG = 0: output is disabled.
  - LP_GPIO_ENABLE_REG = 1: output is enabled.
- LP_GPIO_FUNC*n*_OE_SEL = 0: use the output enable signal from peripheral, for example lp_spi_clk_pad_oe in the column "Output enable signal when LP_GPIO_FUNC*n*_OE_SEL = 0" of Table 10.13-1. Note that the signals such as lp_spi_clk_pad_oe can be 1 (1'd1) or 0 (1'd0), depending on the configuration of corresponding peripherals. If it's 1'd1 in column "Output enable signal when LP_GPIO_FUNC*n*_OE_SEL = 0", it indicates that once LP_GPIO_FUNC*n*_OE_SEL is cleared, the output signal is always enabled by default.

Note:

Signals are numbered consecutively, but not all signals are valid.

- Only the signals with a name assigned in the column "Input signal" in Table 10.13-1 are valid input signals.
- Only the signals with a name assigned in the column "Output signal" in Table 10.13-1 are valid output signals.

Table 10.13-1. Peripheral Signals via LP GPIO Matrix

Signal No.	Input Signal	Default value	Direct Input via LP IO MUX	Output Signal	Output enable signal when LP_GPIO_FUNCn_OE_SEL = 0	Direct Output via LP IO MUX
0	lp_i2c_scl_pad_in	1	no	lp_i2c_scl_pad_out	lp_i2c_scl_pad_oe	no
1	lp_i2c_sda_pad_in	1	no	lp_i2c_sda_pad_out	lp_i2c_sda_pad_oe	no
2	lp_uart_rxd_pad_in	0	yes	lp_uart_txd_pad_out	1'd1	yes
3	lp_uart_ctsn_pad_in	1	no	lp_uart_rtsn_pad_out	1'd1	no
4	lp_uart_dsrn_pad_in	1	no	lp_uart_dtrn_pad_out	1'd1	no
5	lp_spi_ck_pad_in	0	no	lp_spi_ck_pad_out	lp_spi_ck_pad_oe	no
6	lp_spi_cs_pad_in	0	no	lp_spi_cs_pad_out	lp_spi_cs_pad_oe	no
7	lp_spi_d_pad_in	0	no	lp_spi_d_pad_out	lp_spi_d_pad_oe	no
8	lp_spi_q_pad_in	0	no	lp_spi_q_pad_out	lp_spi_q_pad_oe	no
9	lp_i2s_i_bck_pad_in	0	no	lp_i2s_i_bck_pad_out	1'd1	no
10	lp_i2s_i_sd_pad_in	0	no	lp_i2s_o_sd_pad_out	1'd1	no
11	lp_i2s_i_ws_pad_in	0	no	lp_i2s_i_ws_pad_out	1'd1	no
12	lp_i2s_o_bck_pad_in	0	no	lp_i2s_o_bck_pad_out	1'd1	no
13	lp_i2s_o_ws_pad_in	0	no	lp_i2s_o_ws_pad_out	1'd1	no
14	-	-	-	-	-	-
15	-	-	-	-	-	-
16	-	-	-	-	-	-
17	-	-	-	-	-	-
18	-	-	-	-	-	-
19	-	-	-	-	-	-
20	-	-	-	-	-	-
21	-	-	-	-	-	-
22	-	-	-	-	-	-
23	-	-	-	-	-	-

Signal No.	Input Signal	Default value	Direct Input via LP IO MUX	Output Signal	Output enable signal when LP_GPIO_FUNCn_OE_SEL = 0	Direct Output via LP IO MUX
24	-	-	-	-	-	-
25	-	-	-	-	-	-
26	-	-	-	-	-	-
27	-	-	-	-	-	-
28	-	-	-	-	-	-
29	-	-	-	-	-	-
30	-	-	-	-	-	-
31	-	-	-	-	-	-

10.14 HP IO MUX Functions List

Table 10.14-1 shows the HP IO MUX functions and default states of each HP GPIO pin.

Table 10.14-1. HP IO MUX Pin Functions

Name	Function 0	Function 1	Function 2	Function 3	DRV	Reset	Notes
GPIO0	GPIO0	GPIO0	-	-	2	0	R
GPIO1	GPIO1	GPIO1	-	-	2	0	R
GPIO2	MTCK	GPIO2	-	-	2	1*	R
GPIO3	MTDI	GPIO3	-	-	2	1	R
GPIO4	MTMS	GPIO4	-	-	2	1	R
GPIO5	MTDO	GPIO5	-	-	2	0	R
GPIO6	GPIO6	GPIO6	-	SPI2_HOLD_PAD	2	0	R
GPIO7	GPIO7	GPIO7	-	SPI2_CS_PAD	2	0	R
GPIO8	GPIO8	GPIO8	UART0_RTS_PAD	SPI2_D_PAD	2	0	R

Cont'd on next page

Table 10.14-1 – cont'd from previous page

Name	Function 0	Function 1	Function 2	Function 3	DRV	Reset	Notes
GPIO9	GPIO9	GPIO9	UART0_CTS_PAD	SPI2_CK_PAD	2	0	R
GPIO10	GPIO10	GPIO10	UART1_TXD_PAD	SPI2_Q_PAD	2	0	R
GPIO11	GPIO11	GPIO11	UART1_RXD_PAD	SPI2_WP_PAD	2	0	R
GPIO12	GPIO12	GPIO12	UART1_RTS_PAD	-	2	0	R
GPIO13	GPIO13	GPIO13	UART1_CTS_PAD	-	2	0	R
GPIO14	GPIO14	GPIO14	—	-	2	0	R
GPIO15	GPIO15	GPIO15	—	-	2	0	R
GPIO16	GPIO16	GPIO16	—	-	2	0	R
GPIO17	GPIO17	GPIO17	—	-	2	0	R
GPIO18	GPIO18	GPIO18	—	-	2	0	R
GPIO19	GPIO19	GPIO19	—	-	2	0	R
GPIO20	GPIO20	GPIO20	—	-	2	0	R
GPIO21	GPIO21	GPIO21	—	-	2	0	R
GPIO22	GPIO22	GPIO22	—	-	2	0	R
GPIO23	GPIO23	GPIO23	—	REF_50M_CLK_PAD	2	0	R
GPIO24	GPIO24	GPIO24	—	-	3	0	R, USB
GPIO25	GPIO25	GPIO25	—	-	3	3*	R, USB
GPIO26	GPIO26	GPIO26	—	-	2	0	R, USB
GPIO27	GPIO27	GPIO27	—	-	2	0	R, USB
GPIO28	GPIO28	GPIO28	SPI2_CS_PAD	GMAC_PHY_RXDV_PAD	2	0	—
GPIO29	GPIO29	GPIO29	SPI2_D_PAD	GMAC_PHY_RXD0_PAD	2	0	—
GPIO30	GPIO30	GPIO30	SPI2_CK_PAD	GMAC_PHY_RXD1_PAD	2	0	—
GPIO31	GPIO31	GPIO31	SPI2_Q_PAD	GMAC_PHY_RXER_PAD	2	0	—
GPIO32	GPIO32	GPIO32	SPI2_HOLD_PAD	GMAC_RMII_CLK_PAD	2	0	—
GPIO33	GPIO33	GPIO33	SPI2_WP_PAD	GMAC_PHY_TXEN_PAD	2	0	—
GPIO34	GPIO34	GPIO34	SPI2_IO4_PAD	GMAC_PHY_TXD0_PAD	2	0	—
GPIO35	GPIO35	GPIO35	SPI2_IO5_PAD	GMAC_PHY_TXD1_PAD	2	0	—

Cont'd on next page

GoBack

Table 10.14-1 – cont'd from previous page

Name	Function 0	Function 1	Function 2	Function 3	DRV	Reset	Notes
GPIO36	GPIO36	GPIO36	SPI2_IO6_PAD	GMAC_PHY_TXER_PAD	2	0	—
GPIO37	UART0_TXD_PAD	GPIO37	SPI2_IO7_PAD	-	2	5	—
GPIO38	UART0_RXD_PAD	GPIO38	SPI2_DQS_PAD	-	2	0	—
GPIO39	SD1_CDATA0_PAD	GPIO39	—	REF_50M_CLK_PAD	2	0	—
GPIO40	SD1_CDATA1_PAD	GPIO40	—	GMAC_PHY_TXEN_PAD	2	0	—
GPIO41	SD1_CDATA2_PAD	GPIO41	—	GMAC_PHY_TXD0_PAD	2	0	—
GPIO42	SD1_CDATA3_PAD	GPIO42	—	GMAC_PHY_TXD1_PAD	2	0	—
GPIO43	SD1_CCLK_PAD	GPIO43	—	GMAC_PHY_TXER_PAD	2	0	—
GPIO44	SD1_CCMD_PAD	GPIO44	—	GMAC_RMII_CLK_PAD	2	0	—
GPIO45	SD1_CDATA4_PAD	GPIO45	—	GMAC_PHY_RXDV_PAD	2	0	—
GPIO46	SD1_CDATA5_PAD	GPIO46	—	GMAC_PHY_RXD0_PAD	2	0	—
GPIO47	SD1_CDATA6_PAD	GPIO47	—	GMAC_PHY_RXD1_PAD	2	0	—
GPIO48	SD1_CDATA7_PAD	GPIO48	—	GMAC_PHY_RXER_PAD	2	0	—
GPIO49	GPIO49	GPIO49	—	GMAC_PHY_TXEN_PAD	2	0	R
GPIO50	GPIO50	GPIO50	—	GMAC_RMII_CLK_PAD	2	0	R
GPIO51	GPIO51	GPIO51	—	GMAC_PHY_RXDV_PAD	2	0	R
GPIO52	GPIO52	GPIO52	—	GMAC_PHY_RXD0_PAD	2	0	R
GPIO53	GPIO53	GPIO53	—	GMAC_PHY_RXD1_PAD	2	0	R
GPIO54	GPIO54	GPIO54	—	GMAC_PHY_RXER_PAD	2	0	R

Drive Strength

"DRV" column shows the drive strength of each pin after reset:

- **0** - Drive current = ~5 mA
- **1** - Drive current = ~10 mA
- **2** - Drive current = ~20 mA
- **3** - Drive current = ~40 mA

Reset

The default configuration of each pin after reset:

- **0** - input disabled, in high impedance state (IE = 0)
- **1** - input enabled, in high impedance state (IE = 1)
- **1*** - When the value of eFuse bit EFUSE_DIS_PAD_JTAG is
  0 (default), input enabled, pull-up resistor enabled (IE = 1, WPU = 1)
  1, input disabled, in high impedance state (IE = 1)
- **3*** - input disabled, pull-up resistor enabled (IE = 0, USB_PU = 1). See details in Notes.
- **5** - input disabled (IE = 0). Output is controlled by the peripheral of Function 0, and the default output is 1.

We recommend pulling high or low GPIO pins in high impedance state to avoid unnecessary power consumption. You may add pull-up and pull-down resistors in your PCB design, or enable internal pull-up and pull-down resistors during software initialization.

Notes

- **R** - These pins have analog functions.
- **USB** - The pull-up value of a USB pin is controlled by the pin's pull-up value together with the USB pull-up value. If any of the two pull-up values is 1, the pin's pull-up resistor will be enabled.

Notice:
In HP IO MUX, unused pins must be configured to GPIO function.

10.15 LP IO MUX Functions List

Table 10.15-1 shows the LP IO MUX functions of each GPIO pin. GPIO pins are default controlled by HP IO MUX, so Table 10.15-1 only show functions of LP IO MUX.

Table 10.15-1. LP IO MUX Pin Functions

Name	Function 0	Function 1
GPIO0	LP_GPIO0	LP_GPIO0
GPIO1	LP_GPIO1	LP_GPIO1
GPIO2	LP_GPIO2	LP_GPIO2

Name	Function 0	Function 1
GPIO3	LP_GPIO3	LP_GPIO3
GPIO4	LP_GPIO4	LP_GPIO4
GPIO5	LP_GPIO5	LP_GPIO5
GPIO6	LP_GPIO6	LP_GPIO6
GPIO7	LP_GPIO7	LP_GPIO7
GPIO8	LP_GPIO8	LP_GPIO8
GPIO9	LP_GPIO9	LP_GPIO9
GPIO10	LP_GPIO10	LP_GPIO10
GPIO11	LP_GPIO11	LP_GPIO11
GPIO12	LP_GPIO12	LP_GPIO12
GPIO13	LP_GPIO13	LP_GPIO13
GPIO14	LP_UART_TXD_PAD	LP_GPIO14
GPIO15	LP_UART_RXD_PAD	LP_GPIO15

Notice:
In LP IO MUX, unused pins must be configured to LP_GPIO function.

10.16 GPIO Pin Analog Functions List

Table 10.16-1 shows the GPIO pins and their correspond analog functions.

Table 10.16-1. GPIO Pin Analog Functions

Name	Analog Function 0	Analog Function 1
GPIO0	XTAL_32K_N	—
GPIO1	XTAL_32K_P	—
GPIO2	TOUCH_CHANNEL0	—
GPIO3	TOUCH_CHANNEL1	—
GPIO4	TOUCH_CHANNEL2	—
GPIO5	TOUCH_CHANNEL3	—
GPIO6	TOUCH_CHANNEL4	—
GPIO7	TOUCH_CHANNEL5	—
GPIO8	TOUCH_CHANNEL6	—
GPIO9	TOUCH_CHANNEL7	—
GPIO10	TOUCH_CHANNEL8	—
GPIO11	TOUCH_CHANNEL9	—
GPIO12	TOUCH_CHANNEL10	—
GPIO13	TOUCH_CHANNEL11	—
GPIO14	TOUCH_CHANNEL12	—
GPIO15	TOUCH_CHANNEL13	—
GPIO16	ADC1_CHANNEL0	—
GPIO17	ADC1_CHANNEL1	—

Name	Analog Function 0	Analog Function 1
GPIO18	ADC1_CHANNEL2	—
GPIO19	ADC1_CHANNEL3	—
GPIO20	ADC1_CHANNEL4	—
GPIO21	ADC1_CHANNEL5	—
GPIO22	ADC1_CHANNEL6	—
GPIO23	ADC1_CHANNEL7	—
GPIO24	USB1P1_N0	—
GPIO25	USB1P1_P0	—
GPIO26	USB1P1_N1	—
GPIO27	USB1P1_P1	—
GPIO49	ADC2_CHANNEL0	—
GPIO50	ADC2_CHANNEL1	—
GPIO51	ADC2_CHANNEL2	ANA_COMP0
GPIO52	ADC2_CHANNEL3	ANA_COMP0
GPIO53	ADC2_CHANNEL4	ANA_COMP1
GPIO54	ADC2_CHANNEL5	ANA_COMP1

10.17 Event Task Matrix Function

In ESP32-P4, GPIO supports ETM function, that is, the ETM task of GPIO can be triggered by the ETM event of any peripheral, or the ETM task of any peripheral can be triggered by the ETM event of GPIO. For more details about ETM, please refer to Chapter 14 *Event Task Matrix (ETM)*. Only ETM tasks and ETM events related to GPIO are introduced here.

The GPIO ETM provides eight task channels *x* (0 ~ 7). The ETM tasks that each task channel can receive are:

- GPIO_TASK_CH*x*_SET: GPIO goes high when triggered.
- GPIO_TASK_CH*x*_CLEAR: GPIO goes low when triggered.
- GPIO_TASK_CH*x*_TOGGLE: GPIO toggles level when triggered.

Below is an example to configure task channel *x* to control GPIO*y*:

- Configure IO_MUX_GPIO*y*_MCU_SEL to 1, to select Function 1 listed in Table 10.14-1.
- Configure GPIO_ENABLE_REG[*y*] to 1.
- Configure GPIO_EXT_ETM_TASK_GPIO*y*_SEL to *x*.
- Set GPIO_EXT_ETM_TASK_GPIO*y*_EN, to enable ETM task channel *x* to control GPIO*y*.

Note:

- One task channel can be selected by one or more GPIOs.
- When two or three of the signals GPIO_TASK_CH*x*_SET, GPIO_TASK_CH*x*_CLEAR, and GPIO_TASK_CH*x*_TOGGLE of the task channel *x* selected by GPIO*y* are valid at the same time, then GPIO_TASK_CH*x*_SET has the highest priority, GPIO_TASK_CH*x*_CLEAR takes the second higher priority, and GPIO_TASK_CH*x*_TOGGLE has the

lowest priority.

- When GPIO*y* is controlled by ETM task channel, the values of GPIO_OUT_REG, GPIO_FUNC*n*_OUT_INV_SEL, and GPIO_FUNC*n*_OUT_SEL may be modified by the hardware. For such reason, it's recommended to reconfigure these registers when the GPIO is free from the control of ETM task channel.

GPIO has eight event channels, and the ETM events that each event channel can generate are:

- GPIO_EVT_CH*x*_RISE_EDGE: Indicates that the output signal of the corresponding GPIO Filter (see Figure 10.3-1) has a rising edge.
- GPIO_EVT_CH*x*_FALL_EDGE: Indicates that the output signal of the corresponding GPIO Filter (see Figure 10.3-1) has a falling edge.
- GPIO_EVT_CH*x*_ANY_EDGE: Indicates that the output signal of the corresponding GPIO Filter (see Figure 10.3-1) is reversed.

The specific configuration of the event channel is as follows:

- Set GPIO_EXT_ETM_CH*x*_EVENT_EN to enable event channel *x* (0 ~ 7).
- Configure GPIO_EXT_ETM_CH*x*_EVENT_SEL to *y* (0 ~ 54), i.e., select one from the 55 GPIOs.

Note:
One GPIO can be selected by one or more event channels.

In specific applications, GPIO ETM events can be used to trigger GPIO ETM tasks. For example, event channel 0 selects GPIO0, GPIO1 selects task channel 0, and the GPIO_EVT_CH*0*_RISE_EDGE event is used to trigger the GPIO_TASK_CH*0*_TOGGLE task. When a square wave signal is input to the chip through GPIO0, the chip outputs a square wave signal with a frequency divided by 2 through GPIO1.

10.18 Interrupts

ESP32-P4's HP IO MUX and HP GPIO matrix can generate the following interrupt signals that will be sent to the Interrupt Matrix.

- GPIO_INTR0
- GPIO_INTR1
- GPIO_INTR2
- GPIO_INTR3
- GPIO_PAD_COMP_INTR

There are several internal interrupt sources from HP IO MUX and HP GPIO matrix that can generate the above interrupt signals. The interrupt sources from HP IO MUX and HP GPIO matrix are listed with their trigger conditions and the resulted interrupt signals in Table 10.18-1. For more detailed information about GPIO_PAD_COMP_INTR, please refer to Chapter 66 *Analog Voltage Comparator*.

Table 10.18-1. HP IO MUX and HP GPIO Matrix's Internal Interrupt Sources

Internal Interrupt Source	Trigger Condition	Interrupt Signal

GPIO_INTR0	GPIO_STATUS_INTERRUPT[*n*] & GPIO_PIN*n*_INT_ENA[0] (n: 0 ~ 31)	GPIO_INTR0
GPIO_INTR0	GPIO_STATUS1_INTERRUPT[*n*] & GPIO_PIN*n+32*_INT_ENA[0] (n: 0 ~ 22)	GPIO_INTR0
GPIO_INTR1	GPIO_STATUS_INTERRUPT[*n*] & GPIO_PIN*n*_INT_ENA[1] (n: 0 ~ 31)	GPIO_INTR1
GPIO_INTR1	GPIO_STATUS1_INTERRUPT[*n*] & GPIO_PIN*n+32*_INT_ENA[1] (n: 0 ~ 22)	GPIO_INTR1
GPIO_INTR2	GPIO_STATUS_INTERRUPT[*n*] & GPIO_PIN*n*_INT_ENA[3] (n: 0 ~ 31)	GPIO_INTR2
GPIO_INTR2	GPIO_STATUS1_INTERRUPT[*n*] & GPIO_PIN*n+32*_INT_ENA[3] (n: 0 ~ 22)	GPIO_INTR2
GPIO_INTR3	GPIO_STATUS_INTERRUPT[*n*] & GPIO_PIN*n*_INT_ENA[4] (n: 0 ~ 31)	GPIO_INTR3
GPIO_INTR3	GPIO_STATUS1_INTERRUPT[*n*] & GPIO_PIN*n+32*_INT_ENA[4] (n: 0 ~ 22)	GPIO_INTR3

ESP32-P4's LP IO MUX and GPIO matrix can generate the following interrupt signals that will be sent to the Interrupt Matrix.

- LP_GPIO_INTR0

The interrupt source from LP IO MUX and LP GPIO matrix are listed with their trigger conditions and the resulted interrupt signals in Table 10.18-2.

Table 10.18-2. LP IO MUX and LP GPIO Matrix's Internal Interrupt Sources

Internal Interrupt Source	Trigger Condition	Interrupt Signal
LP_GPIO_INTR	LP_GPIO_STATUS_DATA[*n*] (n: 0 ~ 15)	LP_GPIO_INTR

10.19 Register Summary

10.19.1 HP GPIO Matrix Register Summary

The addresses in this section are relative to HP GPIO matrix base address provided in Table 9.3-2 in Chapter 9 *System and Memory*.

The abbreviations given in Column **Access** are explained in Section *Access Types for Registers*.

Name	Description	Address	Access
Configuration Registers			
GPIO_OUT_REG	GPIO0 ~ GPIO31 output register	0x0004	R/W/SC/WTC
GPIO_OUT_W1TS_REG	GPIO0 ~ GPIO31 output set register	0x0008	WT
GPIO_OUT_W1TC_REG	GPIO0 ~ GPIO31 output clear register	0x000C	WT
GPIO_OUT1_REG	GPIO32 ~ GPIO56 output register	0x0010	R/W/SC/WTC
GPIO_OUT1_W1TS_REG	GPIO32 ~ GPIO56 output set register	0x0014	WT

Name	Description	Address	Access
GPIO_OUT1_W1TC_REG	GPIO32 ~ GPIO56 output clear register	0x0018	WT
GPIO_ENABLE_REG	GPIO0 ~ GPIO31 output enable register	0x0020	R/W/WTC
GPIO_ENABLE_W1TS_REG	GPIO0 ~ GPIO31 output enable set register	0x0024	WT
GPIO_ENABLE_W1TC_REG	GPIO0 ~ GPIO31 output enable clear register	0x0028	WT
GPIO_ENABLE1_REG	GPIO32 ~ GPIO56 output enable register	0x002C	R/W/WTC
GPIO_ENABLE1_W1TS_REG	GPIO32 ~ GPIO56 output enable set register	0x0030	WT
GPIO_ENABLE1_W1TC_REG	GPIO32 ~ GPIO56 output enable clear register	0x0034	WT
GPIO_STRAP_REG	Strapping pin register	0x0038	RO
GPIO_IN_REG	GPIO0 ~ GPIO31 input register	0x003C	RO
GPIO_IN1_REG	GPIO32 ~ GPIO56 input register	0x0040	RO
GPIO_STATUS_REG	GPIO0 ~ GPIO31 interrupt status register	0x0044	R/W/WTC
GPIO_STATUS_W1TS_REG	GPIO0 ~ GPIO31 interrupt status set register	0x0048	WT
GPIO_STATUS_W1TC_REG	GPIO0 ~ GPIO31 interrupt status clear register	0x004C	WT
GPIO_STATUS1_REG	GPIO32 ~ GPIO56 interrupt status register	0x0050	R/W/WTC
GPIO_STATUS1_W1TS_REG	GPIO32 ~ GPIO56 interrupt status set register	0x0054	WT
GPIO_STATUS1_W1TC_REG	GPIO32 ~ GPIO56 interrupt status clear register	0x0058	WT
GPIO_INTR_0_REG	GPIO0 ~ GPIO31 interrupt 0 status register	0x005C	RO
GPIO_INTR1_0_REG	GPIO32 ~ GPIO56 interrupt 0 status register	0x0060	RO
GPIO_INTR_1_REG	GPIO0 ~ GPIO31 interrupt 1 status register	0x0064	RO
GPIO_INTR1_1_REG	GPIO32 ~ GPIO56 interrupt 1 status register	0x0068	RO
GPIO_STATUS_NEXT_REG	GPIO0 ~ GPIO31 interrupt source register	0x006C	RO
GPIO_STATUS_NEXT1_REG	GPIO32 ~ GPIO56 interrupt source register	0x0070	RO
GPIO_PIN0_REG	GPIO pin0 configuration register	0x0074	R/W
GPIO_PIN1_REG	GPIO pin1 configuration register	0x0078	R/W
GPIO_PIN2_REG	GPIO pin2 configuration register	0x007C	R/W
GPIO_PIN3_REG	GPIO pin3 configuration register	0x0080	R/W
GPIO_PIN4_REG	GPIO pin4 configuration register	0x0084	R/W
...	...	...	...
GPIO_PIN54_REG	GPIO pin54 configuration register	0x014C	R/W
GPIO_PIN55_REG	GPIO pin55 configuration register	0x0150	R/W
GPIO_PIN56_REG	GPIO pin56 configuration register	0x0154	R/W
GPIO_FUNC1_IN_SEL_CFG_REG	Input signal 1 configuration register	0x015C	R/W
GPIO_FUNC2_IN_SEL_CFG_REG	Input signal 2 configuration register	0x0160	R/W
GPIO_FUNC3_IN_SEL_CFG_REG	Input signal 3 configuration register	0x0164	R/W
...	...	...	...
GPIO_FUNC253_IN_SEL_CFG_REG	Input signal 253 configuration register	0x054C	R/W
GPIO_FUNC254_IN_SEL_CFG_REG	Input signal 254 configuration register	0x0550	R/W
GPIO_FUNC255_IN_SEL_CFG_REG	Input signal 255 configuration register	0x0554	R/W
GPIO_FUNC0_OUT_SEL_CFG_REG	GPIO0 output configuration register	0x0558	varies
GPIO_FUNC1_OUT_SEL_CFG_REG	GPIO1 output configuration register	0x055C	varies
GPIO_FUNC2_OUT_SEL_CFG_REG	GPIO2 output configuration register	0x0560	varies
GPIO_FUNC3_OUT_SEL_CFG_REG	GPIO3 output configuration register	0x0564	varies
...	...	...	...

Name	Description	Address	Access
GPIO_FUNC53_OUT_SEL_CFG_REG	GPIO53 output configuration register	0x062C	varies
GPIO_FUNC54_OUT_SEL_CFG_REG	GPIO54 output configuration register	0x0630	varies
GPIO_FUNC55_OUT_SEL_CFG_REG	GPIO55 output configuration register	0x0634	varies
GPIO_FUNC56_OUT_SEL_CFG_REG	GPIO56 output configuration register	0x0638	varies
GPIO_INTR_2_REG	GPIO0 ~ GPIO31 interrupt 2 status register	0x063C	RO
GPIO_INTR1_2_REG	GPIO32 ~ GPIO56 interrupt 2 status register	0x0640	RO
GPIO_INTR_3_REG	GPIO0 ~ GPIO31 interrupt 3 status register	0x0644	RO
GPIO_INTR1_3_REG	GPIO32 ~ GPIO56 interrupt 3 status register	0x0648	RO
GPIO_CLOCK_GATE_REG	GPIO clock gate register	0x064C	R/W
GPIO_ZERO_DETO_FILTER_CNT_REG	GPIO analog comparator zero detect filter count	0x0710	R/W
GPIO_ZERO_DET1_FILTER_CNT_REG	GPIO analog comparator zero detect filter count	0x0714	R/W
GPIO Interrupt Registers			
GPIO_INT_RAW_REG	Analog comparator interrupt raw register	0x0700	R/WTC/SS
GPIO_INT_ST_REG	Analog comparator interrupt status register	0x0704	RO
GPIO_INT_ENA_REG	Analog comparator interrupt enable register	0x0708	R/W
GPIO_INT_CLR_REG	Analog comparator interrupt clear register	0x070C	WT
Version Register			
GPIO_DATE_REG	Version control register	0x07FC	R/W

10.19.2 HP IO MUX Register Summary

The addresses in this section are relative to the HP IO MUX base address provided in Table 9.3-2 in Chapter 9 *System and Memory*.

The abbreviations given in Column **Access** are explained in Section *Access Types for Registers*.

Name	Description	Address	Access
IO MUX Configuration Registers			
IO_MUX_GPIO0_REG	IO MUX configuration register for GPIO0	0x0004	R/W
IO_MUX_GPIO1_REG	IO MUX configuration register for GPIO1	0x0008	R/W
IO_MUX_GPIO2_REG	IO MUX configuration register for GPIO2	0x000C	R/W
IO_MUX_GPIO3_REG	IO MUX configuration register for GPIO3	0x0010	R/W
IO_MUX_GPIO4_REG	IO MUX configuration register for GPIO4	0x0014	R/W
IO_MUX_GPIO5_REG	IO MUX configuration register for GPIO5	0x0018	R/W
IO_MUX_GPIO6_REG	IO MUX configuration register for GPIO6	0x001C	R/W
IO_MUX_GPIO7_REG	IO MUX configuration register for GPIO7	0x0020	R/W
IO_MUX_GPIO8_REG	IO MUX configuration register for GPIO8	0x0024	R/W
IO_MUX_GPIO9_REG	IO MUX configuration register for GPIO9	0x0028	R/W
IO_MUX_GPIO10_REG	IO MUX configuration register for GPIO10	0x002C	R/W
IO_MUX_GPIO11_REG	IO MUX configuration register for GPIO11	0x0030	R/W
IO_MUX_GPIO12_REG	IO MUX configuration register for GPIO12	0x0034	R/W
IO_MUX_GPIO13_REG	IO MUX configuration register for GPIO13	0x0038	R/W
IO_MUX_GPIO14_REG	IO MUX configuration register for GPIO14	0x003C	R/W
IO_MUX_GPIO15_REG	IO MUX configuration register for GPIO15	0x0040	R/W

Name	Description	Address	Access
IO_MUX_GPIO16_REG	IO MUX configuration register for GPIO16	0x0044	R/W
IO_MUX_GPIO17_REG	IO MUX configuration register for GPIO17	0x0048	R/W
IO_MUX_GPIO18_REG	IO MUX configuration register for GPIO18	0x004C	R/W
IO_MUX_GPIO19_REG	IO MUX configuration register for GPIO19	0x0050	R/W
IO_MUX_GPIO20_REG	IO MUX configuration register for GPIO20	0x0054	R/W
IO_MUX_GPIO21_REG	IO MUX configuration register for GPIO21	0x0058	R/W
IO_MUX_GPIO22_REG	IO MUX configuration register for GPIO22	0x005C	R/W
IO_MUX_GPIO23_REG	IO MUX configuration register for GPIO23	0x0060	R/W
IO_MUX_GPIO24_REG	IO MUX configuration register for GPIO24	0x0064	R/W
IO_MUX_GPIO25_REG	IO MUX configuration register for GPIO25	0x0068	R/W
IO_MUX_GPIO26_REG	IO MUX configuration register for GPIO26	0x006C	R/W
IO_MUX_GPIO27_REG	IO MUX configuration register for GPIO27	0x0070	R/W
IO_MUX_GPIO28_REG	IO MUX configuration register for GPIO28	0x0074	R/W
IO_MUX_GPIO29_REG	IO MUX configuration register for GPIO29	0x0078	R/W
IO_MUX_GPIO30_REG	IO MUX configuration register for GPIO30	0x007C	R/W
IO_MUX_GPIO31_REG	IO MUX configuration register for GPIO31	0x0080	R/W
IO_MUX_GPIO32_REG	IO MUX configuration register for GPIO32	0x0084	R/W
IO_MUX_GPIO33_REG	IO MUX configuration register for GPIO33	0x0088	R/W
IO_MUX_GPIO34_REG	IO MUX configuration register for GPIO34	0x008C	R/W
IO_MUX_GPIO35_REG	IO MUX configuration register for GPIO35	0x0090	R/W
IO_MUX_GPIO36_REG	IO MUX configuration register for GPIO36	0x0094	R/W
IO_MUX_GPIO37_REG	IO MUX configuration register for GPIO37	0x0098	R/W
IO_MUX_GPIO38_REG	IO MUX configuration register for GPIO38	0x009C	R/W
IO_MUX_GPIO39_REG	IO MUX configuration register for GPIO39	0x00A0	R/W
IO_MUX_GPIO40_REG	IO MUX configuration register for GPIO40	0x00A4	R/W
IO_MUX_GPIO41_REG	IO MUX configuration register for GPIO41	0x00A8	R/W
IO_MUX_GPIO42_REG	IO MUX configuration register for GPIO42	0x00AC	R/W
IO_MUX_GPIO43_REG	IO MUX configuration register for GPIO43	0x00B0	R/W
IO_MUX_GPIO44_REG	IO MUX configuration register for GPIO44	0x00B4	R/W
IO_MUX_GPIO45_REG	IO MUX configuration register for GPIO45	0x00B8	R/W
IO_MUX_GPIO46_REG	IO MUX configuration register for GPIO46	0x00BC	R/W
IO_MUX_GPIO47_REG	IO MUX configuration register for GPIO47	0x00C0	R/W
IO_MUX_GPIO48_REG	IO MUX configuration register for GPIO48	0x00C4	R/W
IO_MUX_GPIO49_REG	IO MUX configuration register for GPIO49	0x00C8	R/W
IO_MUX_GPIO50_REG	IO MUX configuration register for GPIO50	0x00CC	R/W
IO_MUX_GPIO51_REG	IO MUX configuration register for GPIO51	0x00D0	R/W
IO_MUX_GPIO52_REG	IO MUX configuration register for GPIO52	0x00D4	R/W
IO_MUX_GPIO53_REG	IO MUX configuration register for GPIO53	0x00D8	R/W
IO_MUX_GPIO54_REG	IO MUX configuration register for GPIO54	0x00DC	R/W
Version Register			
IO_MUX_DATE_REG	Version control register	0x0104	R/W

10.19.3 GPIO EXT Register Summary

GPIO EXT registers consist of SDM registers, Glitch Filter registers, and ETM registers.

The addresses in this section are relative to (HP GPIO matrix base address + 0x0F00). GPIO base address is provided in Table 9.3-2 in Chapter 9 *System and Memory*.

The abbreviations given in Column **Access** are explained in Section *Access Types for Registers*.

Name	Description	Address	Access
SDM Configuration Registers			
GPIO_EXT_SIGMADELTA0_REG	SDM channel 0 duty cycle configuration register	0x0000	R/W
GPIO_EXT_SIGMADELTA1_REG	SDM channel 1 duty cycle configuration register	0x0004	R/W
GPIO_EXT_SIGMADELTA2_REG	SDM channel 2 duty cycle configuration register	0x0008	R/W
GPIO_EXT_SIGMADELTA3_REG	SDM channel 3 duty cycle configuration register	0x000C	R/W
GPIO_EXT_SIGMADELTA4_REG	SDM channel 4 duty cycle configuration register	0x0010	R/W
GPIO_EXT_SIGMADELTA5_REG	SDM channel 5 duty cycle configuration register	0x0014	R/W
GPIO_EXT_SIGMADELTA6_REG	SDM channel 6 duty cycle configuration register	0x0018	R/W
GPIO_EXT_SIGMADELTA7_REG	SDM channel 7 duty cycle configuration register	0x001C	R/W
GPIO_EXT_SIGMADELTA_MISC_REG	MISC register	0x0024	R/W
Glitch Filter Configuration Registers			
GPIO_EXT_GLITCH_FILTER_CH0_REG	Glitch Filter configuration register for channel 0	0x0030	R/W
GPIO_EXT_GLITCH_FILTER_CH1_REG	Glitch Filter configuration register for channel 1	0x0034	R/W
GPIO_EXT_GLITCH_FILTER_CH2_REG	Glitch Filter configuration register for channel 2	0x0038	R/W
GPIO_EXT_GLITCH_FILTER_CH3_REG	Glitch Filter configuration register for channel 3	0x003C	R/W
GPIO_EXT_GLITCH_FILTER_CH4_REG	Glitch Filter configuration register for channel 4	0x0040	R/W
GPIO_EXT_GLITCH_FILTER_CH5_REG	Glitch Filter configuration register for channel 5	0x0044	R/W
GPIO_EXT_GLITCH_FILTER_CH6_REG	Glitch Filter configuration register for channel 6	0x0048	R/W
GPIO_EXT_GLITCH_FILTER_CH7_REG	Glitch Filter configuration register for channel 7	0x004C	R/W
ETM Configuration Registers			
GPIO_EXT_ETM_EVENT_CH0_CFG_REG	ETM configuration register for channel 0	0x0060	R/W

Name	Description	Address	Access
GPIO_EXT_ETM_EVENT_CH1_CFG_REG	ETM configuration register for channel 1	0x0064	R/W
GPIO_EXT_ETM_EVENT_CH2_CFG_REG	ETM configuration register for channel 2	0x0068	R/W
GPIO_EXT_ETM_EVENT_CH3_CFG_REG	ETM configuration register for channel 3	0x006C	R/W
GPIO_EXT_ETM_EVENT_CH4_CFG_REG	ETM configuration register for channel 4	0x0070	R/W
GPIO_EXT_ETM_EVENT_CH5_CFG_REG	ETM configuration register for channel 5	0x0074	R/W
GPIO_EXT_ETM_EVENT_CH6_CFG_REG	ETM configuration register for channel 6	0x0078	R/W
GPIO_EXT_ETM_EVENT_CH7_CFG_REG	ETM configuration register for channel 7	0x007C	R/W
GPIO_EXT_ETM_TASK_P0_CFG_REG	GPIO selection register 0 for ETM	0x00A0	R/W
GPIO_EXT_ETM_TASK_P1_CFG_REG	GPIO selection register 1 for ETM	0x00A4	R/W
GPIO_EXT_ETM_TASK_P2_CFG_REG	GPIO selection register 2 for ETM	0x00A8	R/W
GPIO_EXT_ETM_TASK_P3_CFG_REG	GPIO selection register 3 for ETM	0x00AC	R/W
GPIO_EXT_ETM_TASK_P4_CFG_REG	GPIO selection register 4 for ETM	0x00B0	R/W
GPIO_EXT_ETM_TASK_P5_CFG_REG	GPIO selection register 5 for ETM	0x00B4	R/W
GPIO_EXT_ETM_TASK_P6_CFG_REG	GPIO selection register 6 for ETM	0x00B8	R/W
GPIO_EXT_ETM_TASK_P7_CFG_REG	GPIO selection register 7 for ETM	0x00BC	R/W
GPIO_EXT_ETM_TASK_P8_CFG_REG	GPIO selection register 8 for ETM	0x00C0	R/W
GPIO_EXT_ETM_TASK_P9_CFG_REG	GPIO selection register 9 for ETM	0x00C4	R/W
GPIO_EXT_ETM_TASK_P10_CFG_REG	GPIO selection register 10 for ETM	0x00C8	R/W
GPIO_EXT_ETM_TASK_P11_CFG_REG	GPIO selection register 11 for ETM	0x00CC	R/W
GPIO_EXT_ETM_TASK_P12_CFG_REG	GPIO selection register 12 for ETM	0x00D0	R/W
GPIO_EXT_ETM_TASK_P13_CFG_REG	GPIO selection register 13 for ETM	0x00D4	R/W
Version Register			
GPIO_EXT_VERSION_REG	Version control register	0x00FC	R/W

10.19.4 LP GPIO Matrix Register Summary

The addresses in this section are relative to LP GPIO matrix base address provided in Table 9.3-2 in Chapter 9 *System and Memory*.

The abbreviations given in Column **Access** are explained in Section *Access Types for Registers*.

Name	Description	Address	Access
Clock Enable Register			
LP_GPIO_CLK_EN_REG	LP GPIO clock gate register	0x0000	R/W
Configuration Registers			
LP_GPIO_OUT_REG	LP GPIO output register	0x0008	R/W/WTC
LP_GPIO_OUT_W1TS_REG	LP GPIO output set register	0x000C	WT
LP_GPIO_OUT_W1TC_REG	LP GPIO output clear register	0x0010	WT
LP_GPIO_ENABLE_REG	LP GPIO output enable register	0x0014	R/W/WTC
LP_GPIO_ENABLE_W1TS_REG	LP GPIO output enable set register	0x0018	WT
LP_GPIO_ENABLE_W1TC_REG	LP GPIO output enable clear register	0x001C	WT
LP_GPIO_STATUS_REG	LP GPIO interrupt status register	0x0020	R/W/WTC
LP_GPIO_STATUS_W1TS_REG	LP GPIO interrupt status set register	0x0024	WT
LP_GPIO_STATUS_W1TC_REG	LP GPIO interrupt status clear register	0x0028	WT

Name	Description	Address	Access
LP_GPIO_STATUS_NEXT_REG	LP GPIO interrupt source register	0x002C	RO
LP_GPIO_IN_REG	LP GPIO input register	0x0030	RO
LP_GPIO_PIN0_REG	LP GPIO0 configuration register	0x0034	varies
LP_GPIO_PIN1_REG	LP GPIO1 configuration register	0x0038	varies
LP_GPIO_PIN2_REG	LP GPIO2 configuration register	0x003C	varies
LP_GPIO_PIN3_REG	LP GPIO3 configuration register	0x0040	varies
LP_GPIO_PIN4_REG	LP GPIO4 configuration register	0x0044	varies
LP_GPIO_PIN5_REG	LP GPIO5 configuration register	0x0048	varies
LP_GPIO_PIN6_REG	LP GPIO6 configuration register	0x004C	varies
LP_GPIO_PIN7_REG	LP GPIO7 configuration register	0x0050	varies
LP_GPIO_PIN8_REG	LP GPIO8 configuration register	0x0054	varies
LP_GPIO_PIN9_REG	LP GPIO9 configuration register	0x0058	varies
LP_GPIO_PIN10_REG	LP GPIO10 configuration register	0x005C	varies
LP_GPIO_PIN11_REG	LP GPIO11 configuration register	0x0060	varies
LP_GPIO_PIN12_REG	LP GPIO12 configuration register	0x0064	varies
LP_GPIO_PIN13_REG	LP GPIO13 configuration register	0x0068	varies
LP_GPIO_PIN14_REG	LP GPIO14 configuration register	0x006C	varies
LP_GPIO_PIN15_REG	LP GPIO15 configuration register	0x0070	varies
LP_GPIO_FUNC0_IN_SEL_CFG_REG	Configuration register for input signal 0	0x0074	R/W
LP_GPIO_FUNC1_IN_SEL_CFG_REG	Configuration register for input signal 1	0x0078	R/W
LP_GPIO_FUNC2_IN_SEL_CFG_REG	Configuration register for input signal 2	0x007C	R/W
LP_GPIO_FUNC3_IN_SEL_CFG_REG	Configuration register for input signal 3	0x0080	R/W
LP_GPIO_FUNC4_IN_SEL_CFG_REG	Configuration register for input signal 4	0x0084	R/W
LP_GPIO_FUNC5_IN_SEL_CFG_REG	Configuration register for input signal 5	0x0088	R/W
LP_GPIO_FUNC6_IN_SEL_CFG_REG	Configuration register for input signal 6	0x008C	R/W
LP_GPIO_FUNC7_IN_SEL_CFG_REG	Configuration register for input signal 7	0x0090	R/W
LP_GPIO_FUNC8_IN_SEL_CFG_REG	Configuration register for input signal 8	0x0094	R/W
LP_GPIO_FUNC9_IN_SEL_CFG_REG	Configuration register for input signal 9	0x0098	R/W
LP_GPIO_FUNC10_IN_SEL_CFG_REG	Configuration register for input signal 10	0x009C	R/W
LP_GPIO_FUNC11_IN_SEL_CFG_REG	Configuration register for input signal 11	0x00A0	R/W
LP_GPIO_FUNC12_IN_SEL_CFG_REG	Configuration register for input signal 12	0x00A4	R/W
LP_GPIO_FUNC13_IN_SEL_CFG_REG	Configuration register for input signal 13	0x00A8	R/W
LP_GPIO_FUNC0_OUT_SEL_CFG_REG	Configuration register for LP GPIO0 output	0x00F4	R/W
LP_GPIO_FUNC1_OUT_SEL_CFG_REG	Configuration register for LP GPIO1 output	0x00F8	R/W
LP_GPIO_FUNC2_OUT_SEL_CFG_REG	Configuration register for LP GPIO2 output	0x00FC	R/W
LP_GPIO_FUNC3_OUT_SEL_CFG_REG	Configuration register for LP GPIO3 output	0x0100	R/W
LP_GPIO_FUNC4_OUT_SEL_CFG_REG	Configuration register for LP GPIO4 output	0x0104	R/W
LP_GPIO_FUNC5_OUT_SEL_CFG_REG	Configuration register for LP GPIO5 output	0x0108	R/W
LP_GPIO_FUNC6_OUT_SEL_CFG_REG	Configuration register for LP GPIO6 output	0x010C	R/W
LP_GPIO_FUNC7_OUT_SEL_CFG_REG	Configuration register for LP GPIO7 output	0x0110	R/W
LP_GPIO_FUNC8_OUT_SEL_CFG_REG	Configuration register for LP GPIO8 output	0x0114	R/W
LP_GPIO_FUNC9_OUT_SEL_CFG_REG	Configuration register for LP GPIO9 output	0x0118	R/W
LP_GPIO_FUNC10_OUT_SEL_CFG_REG	Configuration register for LP GPIO10 output	0x011C	R/W

Name	Description	Address	Access
LP_GPIO_FUNC11_OUT_SEL_CFG_REG	Configuration register for LP GPIO11 output	0x0120	R/W
LP_GPIO_FUNC12_OUT_SEL_CFG_REG	Configuration register for LP GPIO12 output	0x0124	R/W
LP_GPIO_FUNC13_OUT_SEL_CFG_REG	Configuration register for LP GPIO13 output	0x0128	R/W
LP_GPIO_FUNC14_OUT_SEL_CFG_REG	Configuration register for LP GPIO14 output	0x012C	R/W
LP_GPIO_FUNC15_OUT_SEL_CFG_REG	Configuration register for LP GPIO15 output	0x0130	R/W
Version Register			
LP_GPIO_VER_DATE_REG	Version control register	0x0004	R/W

10.19.5 LP IO MUX Register Summary

The addresses in this section are relative to LP IO MUX base address provided in Table 9.3-2 in Chapter 9 *System and Memory*.

The abbreviations given in Column **Access** are explained in Section *Access Types for Registers*.

Name	Description	Address	Access
Clock Enable Register			
LP_IOMUX_CLK_EN_REG	LP IO MUX clock gate register	0x0000	R/W
Configuration Registers			
LP_IOMUX_PAD0_REG	LP IO MUX configuration register for GPIO0	0x0008	R/W
LP_IOMUX_PAD1_REG	LP IO MUX configuration register for GPIO1	0x000C	R/W
LP_IOMUX_PAD2_REG	LP IO MUX configuration register for GPIO2	0x0010	R/W
LP_IOMUX_PAD3_REG	LP IO MUX configuration register for GPIO3	0x0014	R/W
LP_IOMUX_PAD4_REG	LP IO MUX configuration register for GPIO4	0x0018	R/W
LP_IOMUX_PAD5_REG	LP IO MUX configuration register for GPIO5	0x001C	R/W
LP_IOMUX_PAD6_REG	LP IO MUX configuration register for GPIO6	0x0020	R/W
LP_IOMUX_PAD7_REG	LP IO MUX configuration register for GPIO7	0x0024	R/W
LP_IOMUX_PAD8_REG	LP IO MUX configuration register for GPIO8	0x0028	R/W
LP_IOMUX_PAD9_REG	LP IO MUX configuration register for GPIO9	0x002C	R/W
LP_IOMUX_PAD10_REG	LP IO MUX configuration register for GPIO10	0x0030	R/W
LP_IOMUX_PAD11_REG	LP IO MUX configuration register for GPIO11	0x0034	R/W
LP_IOMUX_PAD12_REG	LP IO MUX configuration register for GPIO12	0x0038	R/W
LP_IOMUX_PAD13_REG	LP IO MUX configuration register for GPIO13	0x003C	R/W
LP_IOMUX_PAD14_REG	LP IO MUX configuration register for GPIO14	0x0040	R/W
LP_IOMUX_PAD15_REG	LP IO MUX configuration register for GPIO15	0x0044	R/W
LP_IOMUX_LP_PAD_HOLD_REG	LP GPIO PAD hold control register	0x004C	R/W
LP_IOMUX_LP_PAD_HYS_REG	LP GPIO PAD hysteresis control register	0x0050	R/W
Version Register			
LP_IOMUX_VER_DATE_REG	Version control register	0x0004	R/W

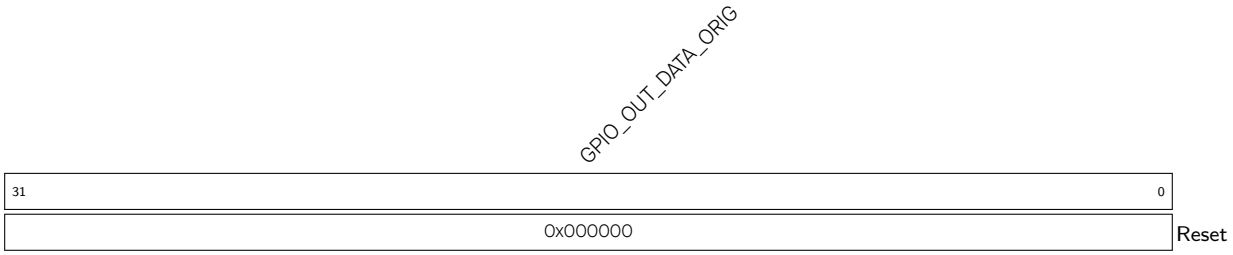
10.20 Registers

10.20.1 HP GPIO Matrix Registers

The addresses in this section are relative to HP GPIO matrix base address provided in Table 9.3-2 in Chapter 9 *System and Memory*.

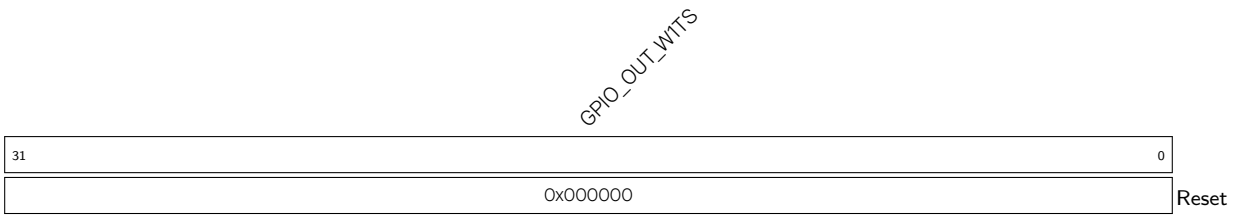
For how to program reserved fields, please refer to Section IX .

Register 10.1. GPIO_OUT_REG (0x0004)



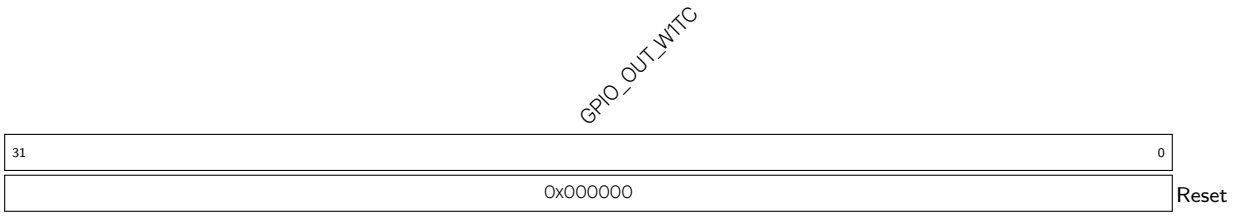
GPIO_OUT_DATA_ORIG Configures the output value of GPIO0 ~ GPIO31 output in simple GPIO output mode.
0: Low level
1: High level
The value of bit0 ~ bit31 correspond to the output value of GPIO0 ~ GPIO31 respectively. (R/W/SC/WTC)

Register 10.2. GPIO_OUT_W1TS_REG (0x0008)



GPIO_OUT_W1TS Configures whether or not to set the output register GPIO_OUT_REG of GPIO0 ~ GPIO31.
0: Not set
1: The corresponding bit in GPIO_OUT_REG will be set to 1
Bit0 ~ bit31 are corresponding to GPIO0 ~ GPIO31. Recommended operation: use this register to set GPIO_OUT_REG.
(WT)

Register 10.3. GPIO_OUT_W1TC_REG (0x000C)



GPIO_OUT_W1TC Configures whether or not to clear the output register GPIO_OUT_REG of GPIO0 ~ GPIO31 output.

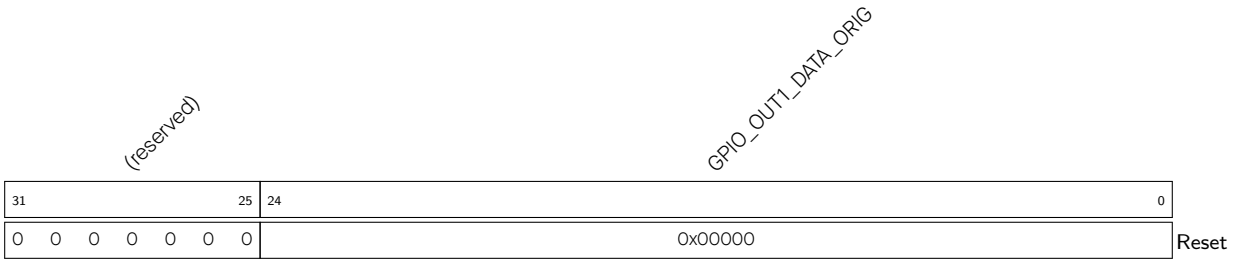
0: Not clear

1: The corresponding bit in GPIO_OUT_REG will be cleared.

Bit0 ~ bit31 are corresponding to GPIO0 ~ GPIO31. Recommended operation: use this register to clear GPIO_OUT_REG.

(WT)

Register 10.4. GPIO_OUT1_REG (0x0010)



GPIO_OUT1_DATA_ORIG Configures the output value of GPIO32 ~ GPIO54 output in simple GPIO output mode.

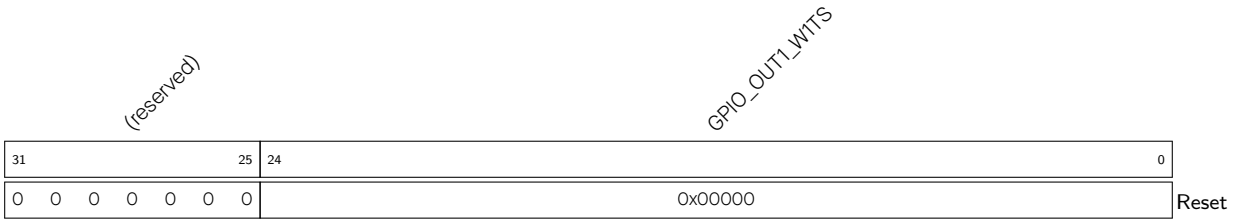
0: Low level

1: High level

The value of bit0 ~ bit22 correspond to the output value of GPIO32 ~ GPIO54 respectively.

(R/W/SC/WTC)

Register 10.5. GPIO_OUT1_W1TS_REG (0x0014)



GPIO_OUT1_W1TS Configures whether or not to set the output register GPIO_OUT1_REG of GPIO32 ~ GPIO54.

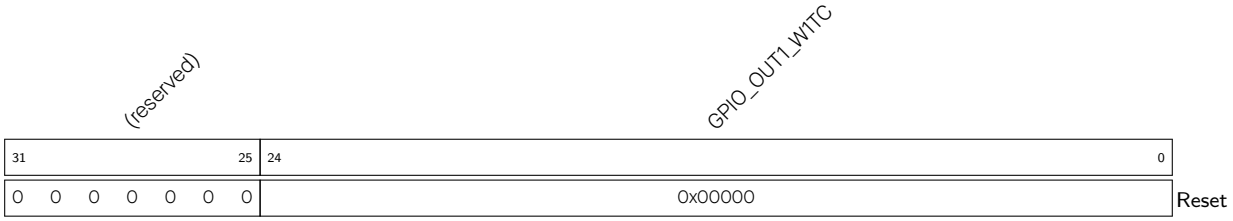
0: Not set

1: The corresponding bit in GPIO_OUT1_REG will be set to 1

Bit0 ~ bit22 are corresponding to GPIO32 ~ GPIO54. Recommended operation: use this register to set GPIO_OUT1_REG.

(WT)

Register 10.6. GPIO_OUT1_W1TC_REG (0x0018)



GPIO_OUT1_W1TC Configures whether or not to clear the output register GPIO_OUT1_REG of GPIO32 ~ GPIO54 output.

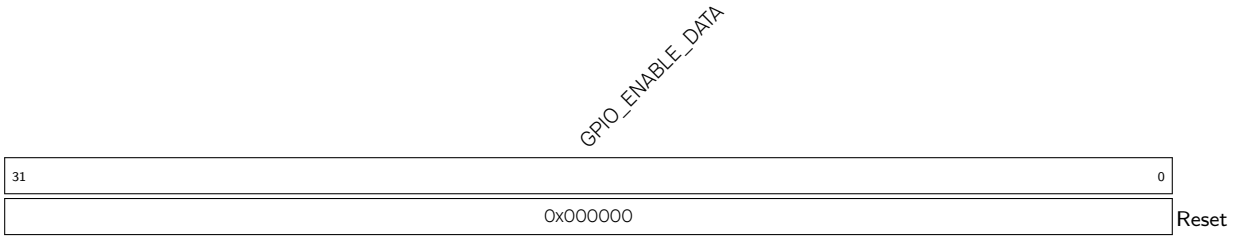
0: Not clear

1: The corresponding bit in GPIO_OUT1_REG will be cleared.

Bit0 ~ bit22 are corresponding to GPIO32 ~ GPIO54. Recommended operation: use this register to clear GPIO_OUT_REG.

(WT)

Register 10.7. GPIO_ENABLE_REG (0x0020)



GPIO_ENABLE_DATA Configures whether or not to enable the output of GPIO0 ~ GPIO31.

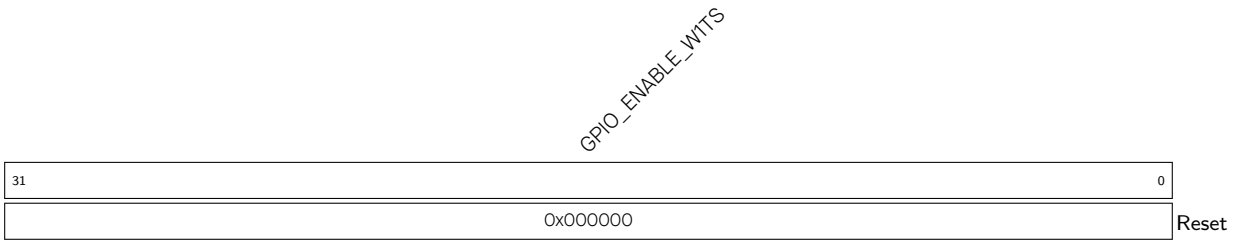
0: Not enable

1: Enable

Bit0 ~ bit31 are corresponding to GPIO0 ~ GPIO31

(R/W/WTC)

Register 10.8. GPIO_ENABLE_W1TS_REG (0x0024)



GPIO_ENABLE_W1TS Configures whether or not to set the output enable register GPIO_ENABLE_REG of GPIO0 ~ GPIO31.

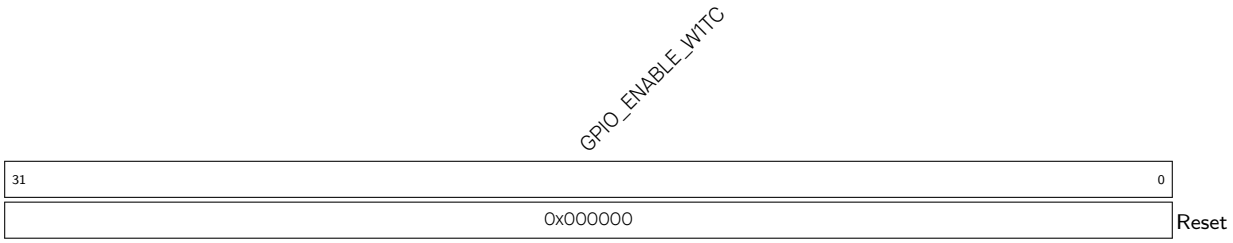
0: Not set

1: The corresponding bit in GPIO_ENABLE_REG will be set to 1

Bit0 ~ bit31 are corresponding to GPIO0 ~ GPIO31. Recommended operation: use this register to set GPIO_ENABLE_REG.

(WT)

Register 10.9. GPIO_ENABLE_W1TC_REG (0x0028)



GPIO_ENABLE_W1TC Configures whether or not to clear the output enable register GPIO_ENABLE_REG of GPIO0 ~ GPIO31.

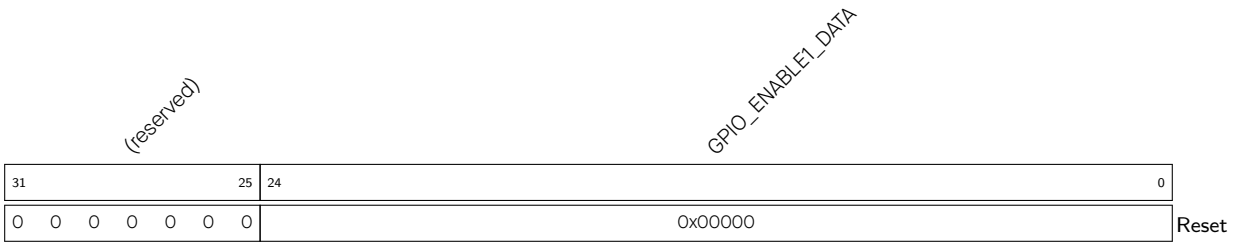
0: Not clear

1: The corresponding bit in GPIO_ENABLE_REG will be cleared

Bit0 ~ bit31 are corresponding to GPIO0 ~ 31. Recommended operation: use this register to clear GPIO_ENABLE_REG.

(WT)

Register 10.10. GPIO_ENABLE1_REG (0x002C)



GPIO_ENABLE1_DATA Configures whether or not to enable the output of GPIO32 ~ GPIO54.

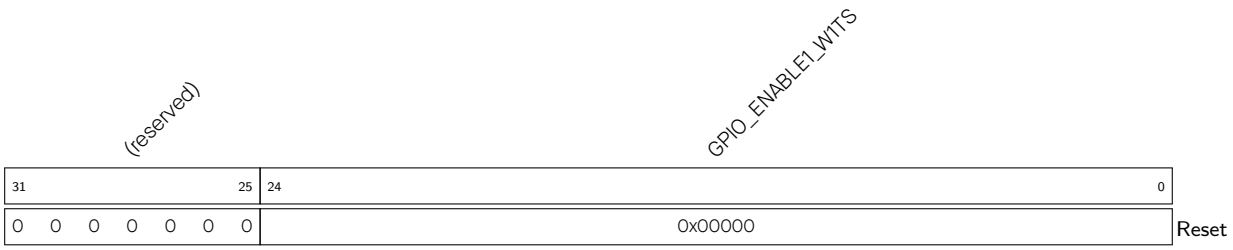
0: Not enable

1: Enable

Bit0 ~ bit22 are corresponding to GPIO32 ~ GPIO54.

(R/W/WTC)

Register 10.11. GPIO_ENABLE1_W1TS_REG (0x0030)



GPIO_ENABLE1_W1TS Configures whether or not to set the output enable register GPIO_ENABLE1_REG of GPIO32 ~ GPIO54.

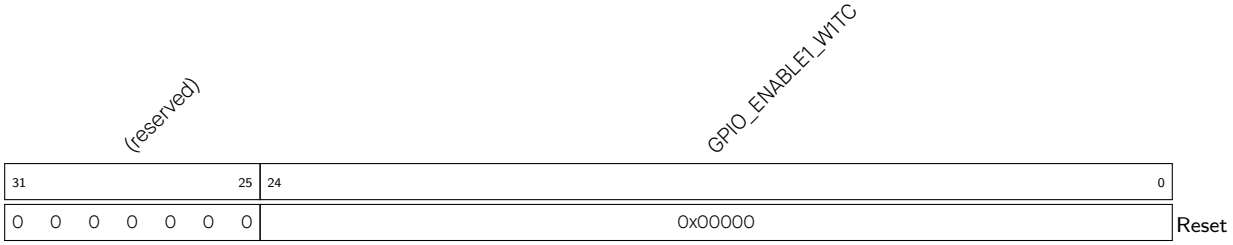
0: Not set

1: The corresponding bit in GPIO_ENABLE1_REG will be set to 1

Bit0 ~ bit22 are corresponding to GPIO32 ~ GPIO54. Recommended operation: use this register to set GPIO_ENABLE1_REG.

(WT)

Register 10.12. GPIO_ENABLE1_W1TC_REG (0x0034)



GPIO_ENABLE1_W1TC Configures whether or not to clear the output enable register GPIO_ENABLE1_REG of GPIO32 ~ GPIO54.

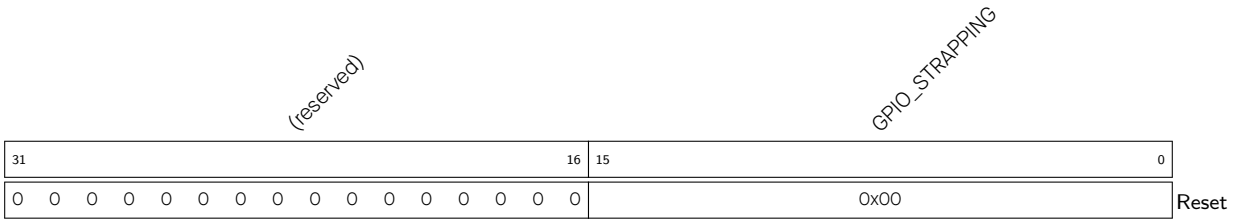
0: Not clear

1: The corresponding bit in GPIO_ENABLE1_REG will be cleared

Bit0 ~ bit22 are corresponding to GPIO32 ~ 54. Recommended operation: use this register to clear GPIO_ENABLE1_REG.

(WT)

Register 10.13. GPIO_STRAP_REG (0x0038)



GPIO_STRAPPING Represents the values of GPIO strapping pins.

bit0: GPIO38

bit1: GPIO37

bit2: GPIO36

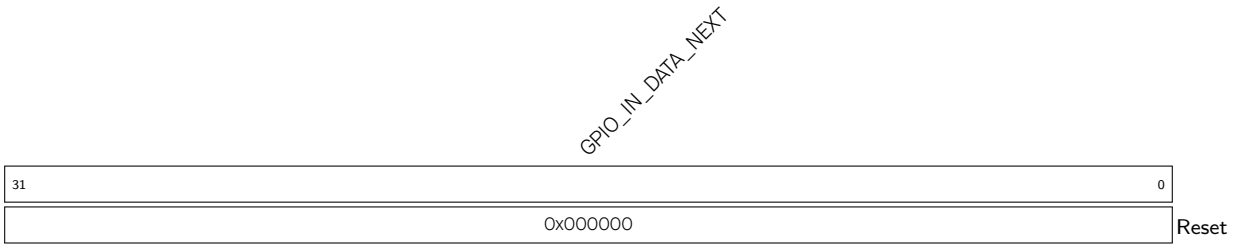
bit3: GPIO35

bit4: GPIO34

bit5 ~ bit15: invalid

For more information about the functions controlled by strapping pins, see *ESP32-P4 Datasheet* > Section *Strapping Pins*. (RO)

Register 10.14. GPIO_IN_REG (0x003C)



GPIO_IN_DATA_NEXT Represents the input value of GPIO0 ~ GPIO31. Each bit represents a pin input value:

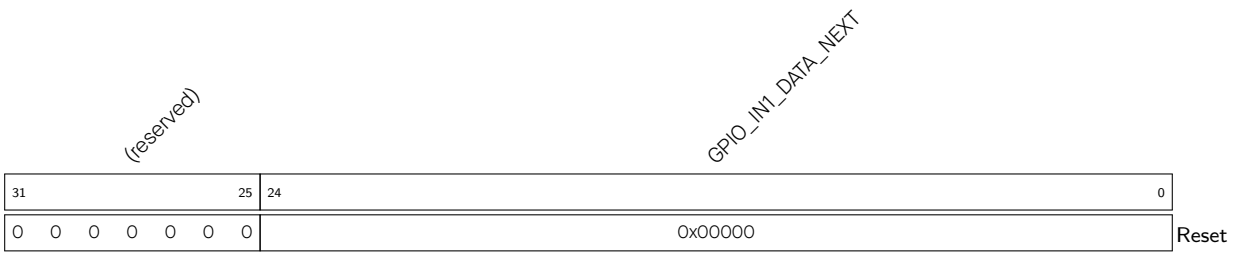
0: Low level

1: High level

Bit0 ~ bit31 are corresponding to GPIO0 ~ GPIO31.

(RO)

Register 10.15. GPIO_IN1_REG (0x0040)



GPIO_IN1_DATA_NEXT Represents the input value of GPIO32 ~ GPIO54. Each bit represents a pin input value:

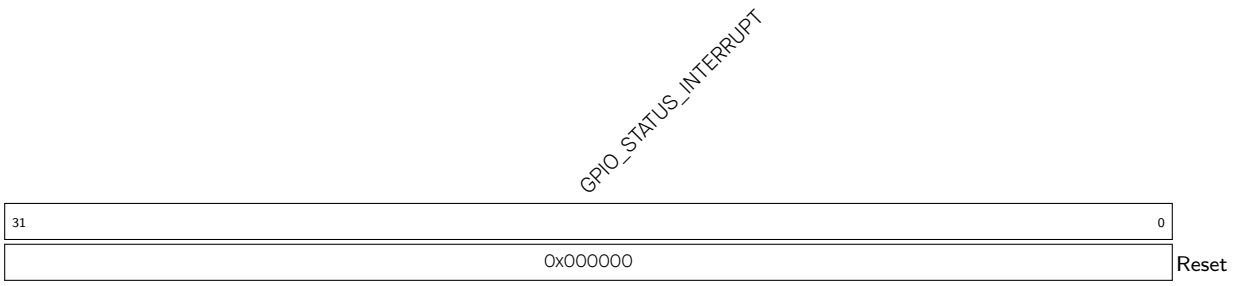
0: Low level

1: High level

Bit0 ~ bit22 are corresponding to GPIO32 ~ GPIO54.

(RO)

Register 10.16. GPIO_STATUS_REG (0x0044)



GPIO_STATUS_INTERRUPT The interrupt status of GPIO0 ~ GPIO31, can be configured by the software.

Bit0 ~ bit31 are corresponding to GPIO0 ~ GPIO31.

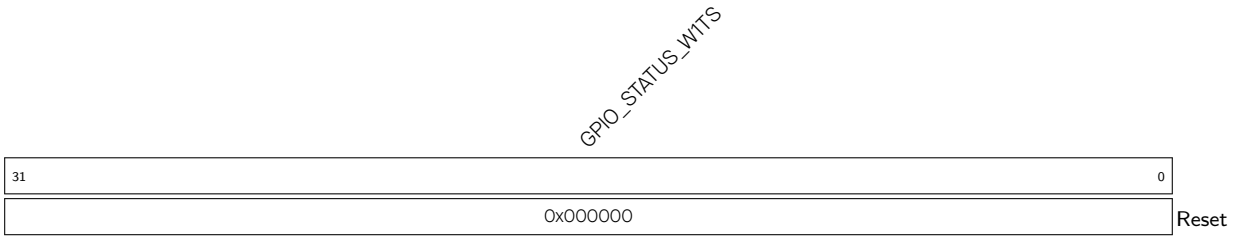
Each bit represents the status of its corresponding GPIO:

0: Represents the GPIO does not generate the interrupt configured by GPIO_PIN*n*_INT_TYPE, or this bit is configured to 0 by the software.

1: Represents the GPIO generates the interrupt configured by GPIO_PIN*n*_INT_TYPE, or this bit is configured to 1 by the software.

(R/W/WTC)

Register 10.17. GPIO_STATUS_W1TS_REG (0x0048)



GPIO_STATUS_W1TS Configures whether or not to set the interrupt status register GPIO_STATUS_INTERRUPT of GPIO0 ~ GPIO31.

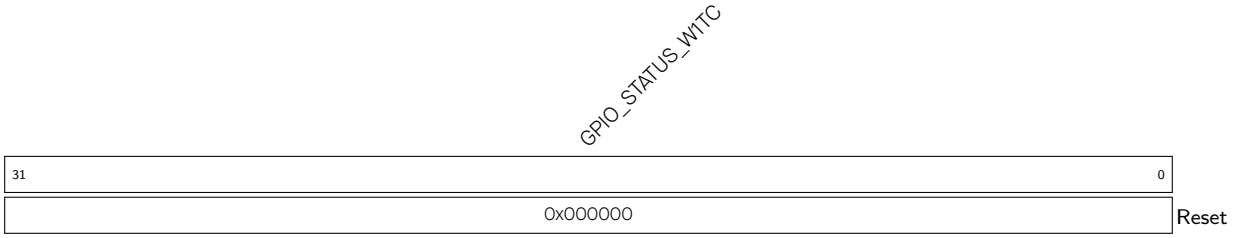
Bit0 ~ bit31 are corresponding to GPIO0 ~ GPIO31.

If the value 1 is written to a bit here, the corresponding bit in GPIO_STATUS_INTERRUPT will be set to 1.

Recommended operation: use this register to set GPIO_STATUS_INTERRUPT.

(WT)

Register 10.18. GPIO_STATUS_W1TC_REG (0x004C)



GPIO_STATUS_W1TC Configures whether or not to clear the interrupt status register GPIO_STATUS_INTERRUPT of GPIO0 ~ GPIO31.

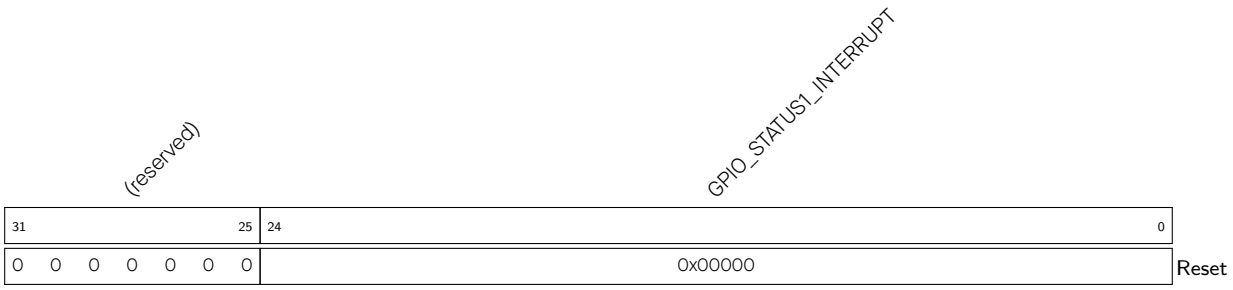
Bit0 ~ bit31 are corresponding to GPIO0 ~ GPIO31.

If the value 1 is written to a bit here, the corresponding bit in GPIO_STATUS_INTERRUPT will be cleared.

Recommended operation: use this register to clear GPIO_STATUS_INTERRUPT.

(WT)

Register 10.19. GPIO_STATUS1_REG (0x0050)



GPIO_STATUS1_INTERRUPT The interrupt status of GPIO32 ~ GPIO54, can be configured by the software.

Bit0 ~ bit22 are corresponding to GPIO32 ~ GPIO54.

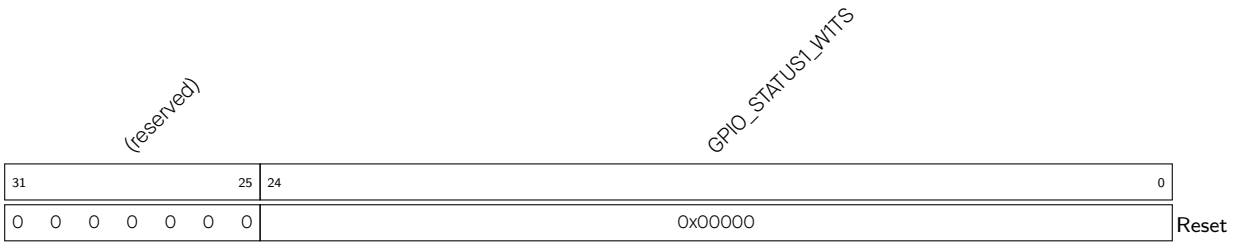
Each bit represents the status of its corresponding GPIO:

0: Represents the GPIO does not generate the interrupt configured by GPIO_PIN*n*_INT_TYPE, or this bit is configured to 0 by the software.

1: Represents the GPIO generates the interrupt configured by GPIO_PIN*n*_INT_TYPE, or this bit is configured to 1 by the software.

(R/W/WTC)

Register 10.20. GPIO_STATUS1_W1TS_REG (0x0054)



GPIO_STATUS1_W1TS Configures whether or not to set the interrupt status register GPIO_STATUS1_INTERRUPT of GPIO32 ~ GPIO54.

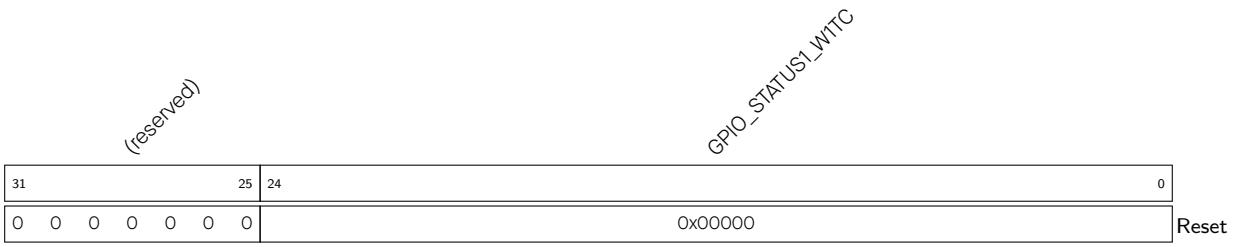
Bit0 ~ bit22 are corresponding to GPIO32 ~ GPIO54.

If the value 1 is written to a bit here, the corresponding bit in GPIO_STATUS1_INTERRUPT will be set to 1.

Recommended operation: use this register to set GPIO_STATUS1_INTERRUPT.

(WT)

Register 10.21. GPIO_STATUS1_W1TC_REG (0x0058)



GPIO_STATUS1_W1TC Configures whether or not to clear the interrupt status register GPIO_STATUS1_INTERRUPT of GPIO32 ~ GPIO54.

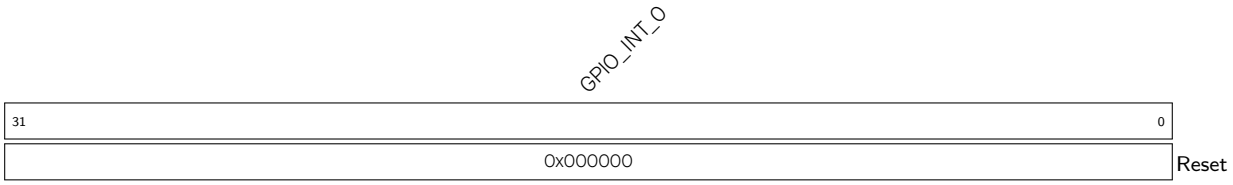
Bit0 ~ bit22 are corresponding to GPIO32 ~ GPIO54.

If the value 1 is written to a bit here, the corresponding bit in GPIO_STATUS1_INTERRUPT will be cleared.

Recommended operation: use this register to clear GPIO_STATUS1_INTERRUPT.

(WT)

Register 10.22. GPIO_INTR_0_REG (0x005C)



GPIO_INT_0 Represents the CPU interrupt 0 status of GPIO0 ~ GPIO31. Each bit represents:

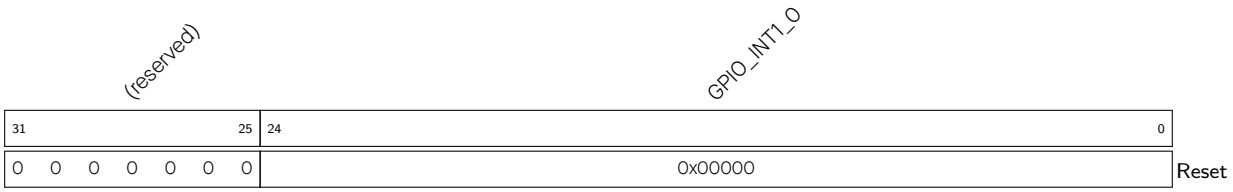
0: Represents CPU interrupt is not enabled, or the GPIO does not generate the interrupt configured by GPIO_PIN*n*_INT_TYPE.

1: Represents the GPIO generates an interrupt configured by GPIO_PIN*n*_INT_TYPE after the CPU interrupt is enabled.

Bit0 ~ bit31 are corresponding to GPIO0 ~ GPIO31. This interrupt status is corresponding to the bit in GPIO_STATUS_REG when assert (high) enable signal (bit13 of GPIO_PIN*n*_REG).

(RO)

Register 10.23. GPIO_INTR1_0_REG (0x0060)



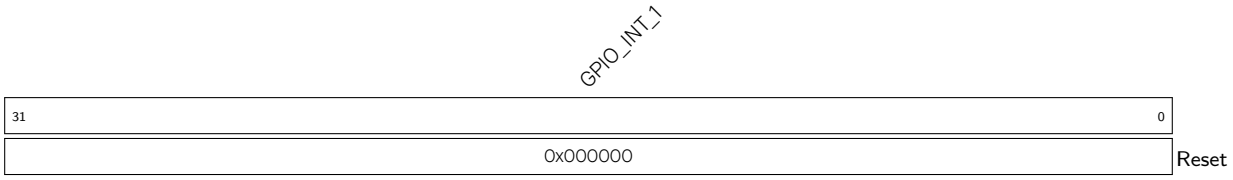
GPIO_INT1_0 Represents the CPU interrupt 0 status of GPIO32 ~ GPIO54. Each bit represents:

0: Represents CPU interrupt is not enabled, or the GPIO does not generate the interrupt configured by GPIO_PIN*n*_INT_TYPE.

1: Represents the GPIO generates an interrupt configured by GPIO_PIN*n*_INT_TYPE after the CPU interrupt is enabled.

Bit0 ~ bit22 are corresponding to GPIO32 ~ GPIO54. This interrupt status is corresponding to the bit in GPIO_STATUS1_REG when assert (high) enable signal (bit13 of GPIO_PIN*n*_REG). (RO)

Register 10.24. GPIO_INTR_1_REG (0x0064)



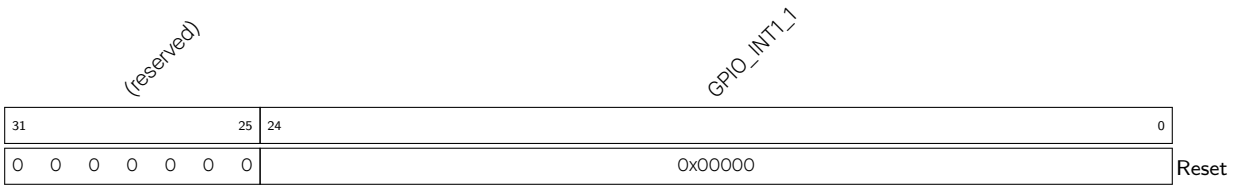
GPIO_INT_1 Represents the CPU interrupt 1 status of GPIO0 ~ GPIO31. Each bit represents:

0: Represents CPU interrupt is not enabled, or the GPIO does not generate the interrupt configured by GPIO_PIN*n*_INT_TYPE.

1: Represents the GPIO generates an interrupt configured by GPIO_PIN*n*_INT_TYPE after the CPU interrupt is enabled.

Bit0 ~ bit31 are corresponding to GPIO0 ~ GPIO31. This interrupt status is corresponding to the bit in GPIO_STATUS_REG when assert (high) enable signal (bit14 of GPIO_PIN*n*_REG). (RO)

Register 10.25. GPIO_INTR1_1_REG (0x0068)



GPIO_INT1_1 Represents the CPU interrupt 1 status of GPIO32 ~ GPIO54. Each bit represents:

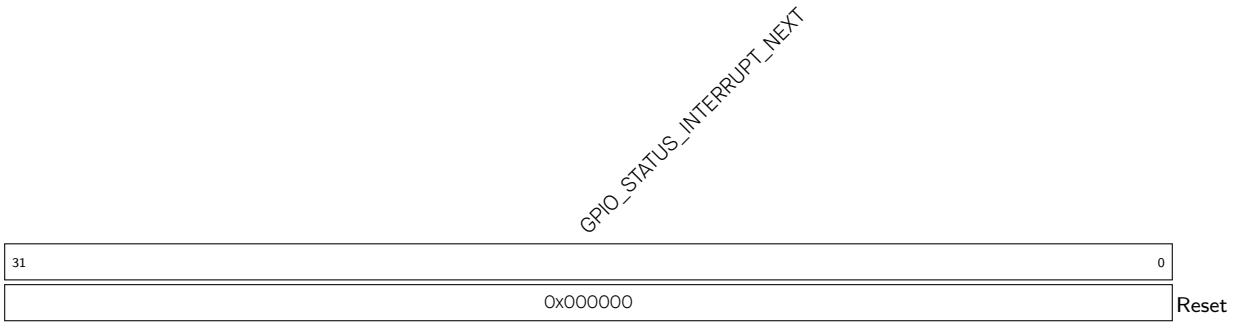
0: Represents CPU interrupt is not enabled, or the GPIO does not generate the interrupt configured by GPIO_PIN*n*_INT_TYPE.

1: Represents the GPIO generates an interrupt configured by GPIO_PIN*n*_INT_TYPE after the CPU interrupt is enabled.

Bit0 ~ bit22 are corresponding to GPIO32 ~ GPIO54. This interrupt status is corresponding to the bit in GPIO_STATUS1_REG when assert (high) enable signal (bit14 of GPIO_PIN*n*_REG).

(RO)

Register 10.26. GPIO_STATUS_NEXT_REG (0x006C)



GPIO_STATUS_INTERRUPT_NEXT Represents the interrupt source signal of GPIO0 ~ GPIO31.

Bit0 ~ bit31 are corresponding to GPIO0 ~ GPIO31. Each bit represents:

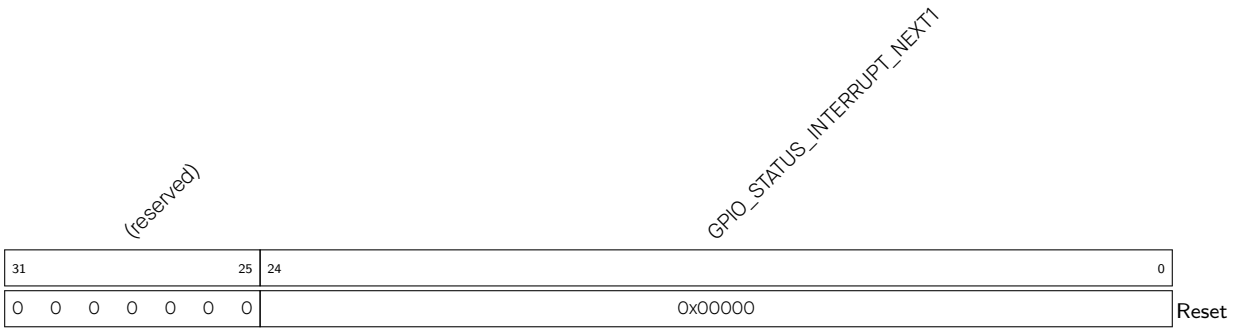
0: The GPIO does not generate the interrupt configured by GPIO_PIN*n*_INT_TYPE.

1: The GPIO generates an interrupt configured by GPIO_PIN*n*_INT_TYPE.

The interrupt could be rising edge interrupt, falling edge interrupt, level sensitive interrupt and any edge interrupt.

(RO)

Register 10.27. GPIO_STATUS_NEXT1_REG (0x0070)



GPIO_STATUS_INTERRUPT_NEXT1 Represents the interrupt source signal of GPIO32 ~ GPIO54. Bit0 ~ bit22 are corresponding to GPIO32 ~ GPIO54. Each bit represents:

0: The GPIO does not generate the interrupt configured by GPIO_PIN*n*_INT_TYPE.

1: The GPIO generates an interrupt configured by GPIO_PIN*n*_INT_TYPE.

The interrupt could be rising edge interrupt, falling edge interrupt, level sensitive interrupt and any edge interrupt.

(RO)

Register 10.28. GPIO_PIN*n*_REG (*n*: 0 - 54) (0x0074+0x4**n*)



GPIO_PIN*n*_SYNC2_BYPASS Configures whether or not to synchronize GPIO input data on either edge of HP IO MUX operating clock for the second-level synchronization.

0: Not synchronize

1: Synchronize on falling edge

2: Synchronize on rising edge

3: Synchronize on rising edge

(R/W)

GPIO_PIN*n*_PAD_DRIVER Configures to select pin drive mode.

0: Normal output

1: Open drain output

(R/W)

GPIO_PIN*n*_SYNC1_BYPASS Configures whether or not to synchronize GPIO input data on either edge of HP IO MUX operating clock for the first-level synchronization.

0: Not synchronize

1: Synchronize on falling edge

2: Synchronize on rising edge

3: Synchronize on rising edge

(R/W)

GPIO_PIN*n*_INT_TYPE Configures GPIO interrupt type.

0: GPIO interrupt disabled

1: Rising edge trigger

2: Falling edge trigger

3: Any edge trigger

4: Low level trigger

5: High level trigger

(R/W)

GPIO_PIN*n*_WAKEUP_ENABLE Configures whether or not to enable GPIO wake-up function.

0: Disable

1: Enable

This function only wakes up the CPU from Light-sleep.

(R/W)

Continued on the next page...

Register 10.28. GPIO_PIN*n*_REG (*n*: 0 - 54) (0x0074+0x4**n*)

Continued from the previous page...

GPIO_PIN*n*_INT_ENA Configures whether or not to enable CPU interrupt or CPU non-maskable interrupt.

bit13: Configures whether or not to enable CPU interrupt 0:

0: Disable

1: Enable

bit14: Configures whether or not to enable CPU interrupt 1:

0: Disable

1: Enable

bit15: invalid

bit16: Configures whether or not to enable CPU interrupt 2:

0: Disable

1: Enable

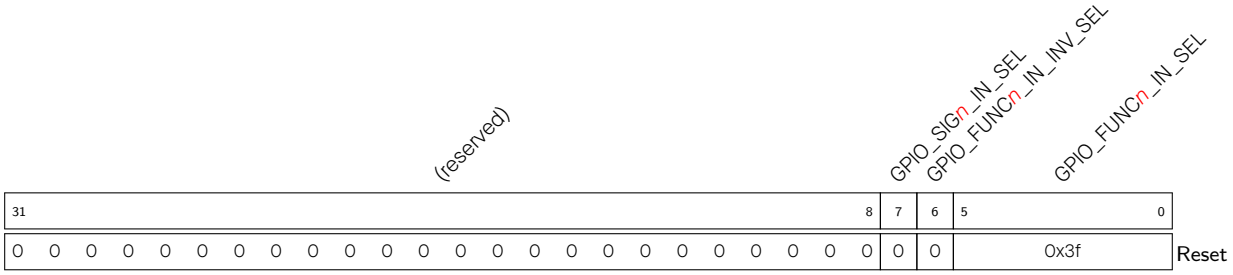
bit17: Configures whether or not to enable CPU interrupt 3:

0: Disable

1: Enable

(R/W)

Register 10.29. GPIO_FUNC*n*_IN_SEL_CFG_REG (*n*: 0 - 255) (0x015C+4n*)**



GPIO_FUNC*n*_IN_SEL Configures to select a pin from the 55 GPIO pins to connect the input signal *n*.

0: Select GPIO0

1: Select GPIO1

.....

53: Select GPIO53

54: Select GPIO54

55 ~ 61: invalid

Or

62: A constantly low input

63: A constantly high input

(R/W)

GPIO_FUNC*n*_IN_INV_SEL Configures whether or not to invert the input value.

0: Not invert

1: Invert

(R/W)

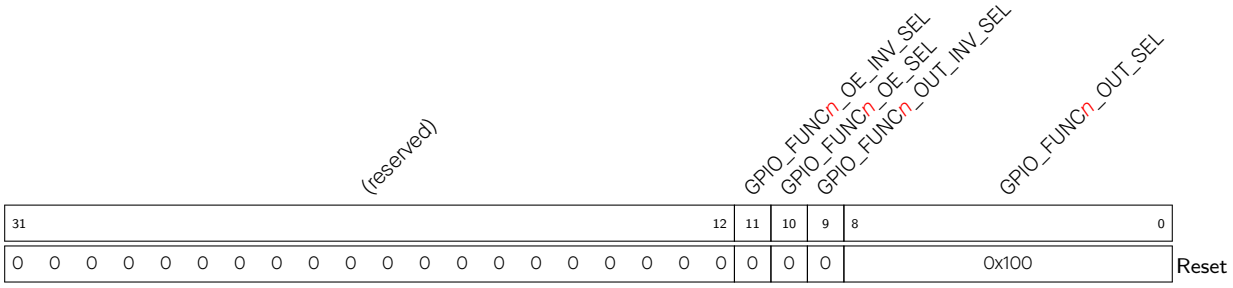
GPIO_SIG*n*_IN_SEL Configures whether or not to route signals via HP GPIO matrix.

0: Bypass HP GPIO matrix, i.e., connect signals directly to peripheral configured in HP IO MUX.

1: Route signals via HP GPIO matrix.

(R/W)

Register 10.30. GPIO_FUNC*n*_OUT_SEL_CFG_REG (*n*: 0 - 54) (0x0558+0x4**n*)



GPIO_FUNC*n*_OUT_SEL Configures to select a signal *Y* (0 <= *Y* < 256) from 256 peripheral signals to be output to GPIO*n*.

0: Select signal 0

1: Select signal 1

.....

254: Select signal 254

255: Select signal 255

Or

256: Bit*n* of GPIO_OUT_REG and GPIO_OUT1_REG are selected as the output value, GPIO_ENABLE_REG and GPIO_ENABLE1_REG are selected as the output enable.

257 ~ 511: invalid

For the detailed signal list, see Table 10.12-1.

(R/W/SC)

GPIO_FUNC*n*_OUT_INV_SEL Configures whether or not to invert the output value.

0: Not invert

1: Invert

(R/W/SC)

GPIO_FUNC*n*_OE_SEL Configures to select the source of output enable signal.

0: Use output enable signal from peripheral.

1: Force the output enable signal to be sourced from bit*n* (*n*: 0 ~ 31) of GPIO_ENABLE_REG and bit*n*-32 (*n*: 32 ~ 54) of GPIO_ENABLE1_REG.

(R/W)

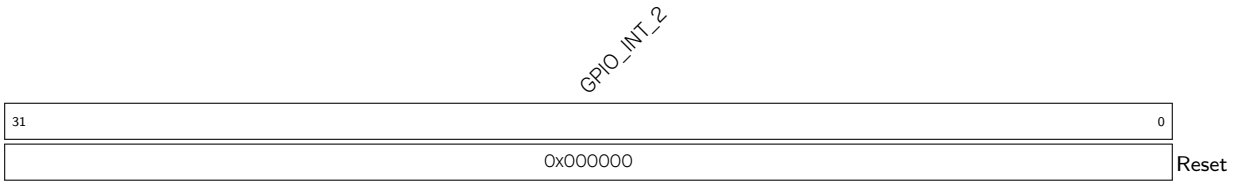
GPIO_FUNC*n*_OE_INV_SEL Configures whether or not to invert the output enable signal.

0: Not invert

1: Invert

(R/W)

Register 10.31. GPIO_INTR_2_REG (0x063C)



GPIO_INT_2 Represents the CPU interrupt 2 status of GPIO0 ~ GPIO31. Each bit represents:

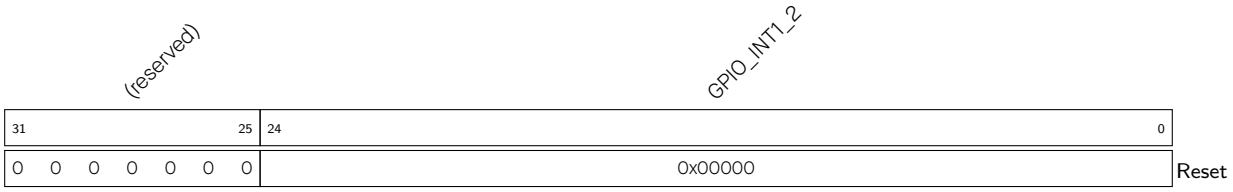
0: Represents CPU interrupt is not enabled, or the GPIO does not generate the interrupt configured by GPIO_PIN*n*_INT_TYPE.

1: Represents the GPIO generates an interrupt configured by GPIO_PIN*n*_INT_TYPE after the CPU interrupt is enabled.

Bit0 ~ bit31 are corresponding to GPIO0 ~ GPIO31. This interrupt status is corresponding to the bit in GPIO_STATUS_REG when assert (high) enable signal (bit16 of GPIO_PIN*n*_REG).

(RO)

Register 10.32. GPIO_INTR1_2_REG (0x0640)



GPIO_INT1_2 Represents the CPU interrupt 2 status of GPIO32 ~ GPIO54. Each bit represents:

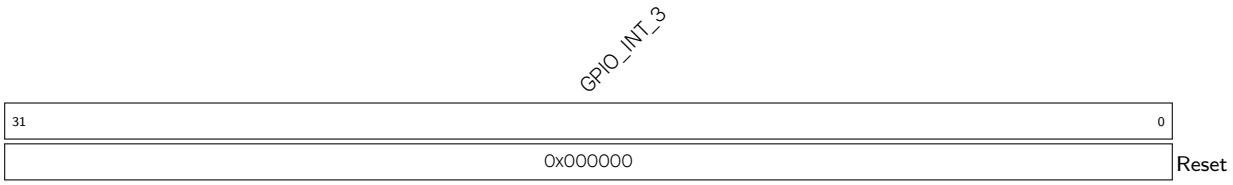
0: Represents CPU interrupt is not enabled, or the GPIO does not generate the interrupt configured by GPIO_PIN*n*_INT_TYPE.

1: Represents the GPIO generates an interrupt configured by GPIO_PIN*n*_INT_TYPE after the CPU interrupt is enabled.

Bit0 ~ bit22 are corresponding to GPIO32 ~ GPIO54. This interrupt status is corresponding to the bit in GPIO_STATUS1_REG when assert (high) enable signal (bit16 of GPIO_PIN*n*_REG).

(RO)

Register 10.33. GPIO_INTR_3_REG (0x0644)



GPIO_INT_3 Represents the CPU interrupt 3 status of GPIO0 ~ GPIO31. Each bit represents:

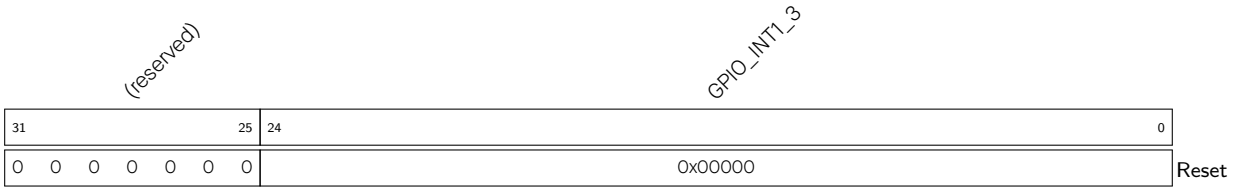
0: Represents CPU interrupt is not enabled, or the GPIO does not generate the interrupt configured by GPIO_PIN*n*_INT_TYPE.

1: Represents the GPIO generates an interrupt configured by GPIO_PIN*n*_INT_TYPE after the CPU interrupt is enabled.

Bit0 ~ bit31 are corresponding to GPIO0 ~ GPIO31. This interrupt status is corresponding to the bit in GPIO_STATUS_REG when assert (high) enable signal (bit17 of GPIO_PIN*n*_REG).

(RO)

Register 10.34. GPIO_INTR1_3_REG (0x0648)



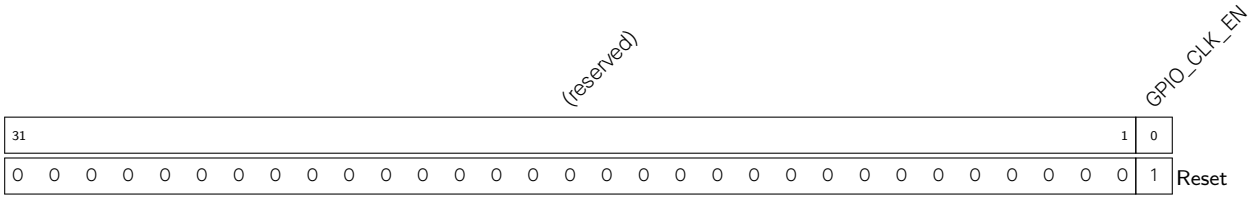
GPIO_INT1_3 Represents the CPU interrupt 3 status of GPIO32 ~ GPIO54. Each bit represents:

0: Represents CPU interrupt is not enabled, or the GPIO does not generate the interrupt configured by GPIO_PIN*n*_INT_TYPE.

1: Represents the GPIO generates an interrupt configured by GPIO_PIN*n*_INT_TYPE after the CPU interrupt is enabled.

Bit0 ~ bit22 are corresponding to GPIO32 ~ GPIO54. This interrupt status is corresponding to the bit in GPIO_STATUS1_REG when assert (high) enable signal (bit17 of GPIO_PIN*n*_REG).

Register 10.35. GPIO_CLOCK_GATE_REG (0x064C)



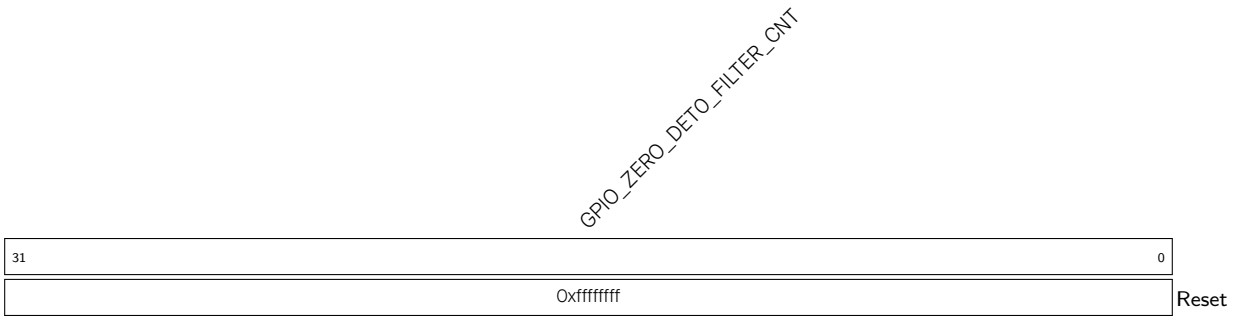
GPIO_CLK_EN Configures whether or not to enable clock gate.

0: Not enable

1: Enable, the clock is free running.

(R/W)

Register 10.36. GPIO_ZERO_DET0_FILTER_CNT_REG (0x0710)



GPIO_ZERO_DET0_FILTER_CNT Configures to change GPIO analog comparator0 zero detect filter's threshold.

Interrupts in GPIO_ZERO_DET0_FILTER_CNT iomux_core_clk cycles after one interrupt will be fillterred, and not be seen by CPU. (R/W)

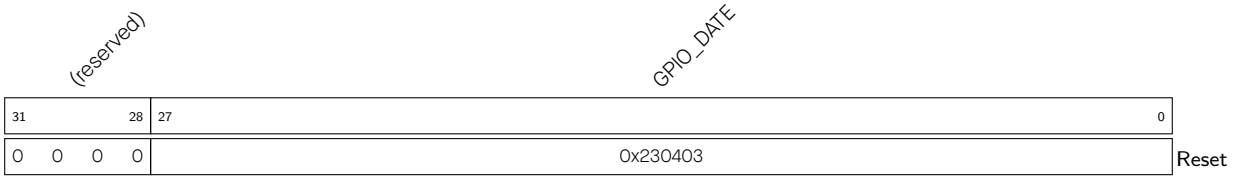
Register 10.37. GPIO_ZERO_DET1_FILTER_CNT_REG (0x0714)



GPIO_ZERO_DET1_FILTER_CNT Configures to change GPIO analog comparator1 zero detect filter's threshold.

Interrupts in GPIO_ZERO_DET0_FILTER_CNT iomux_core_clk cycles after one interrupt will be fillterred, and not be seen by CPU. (R/W)

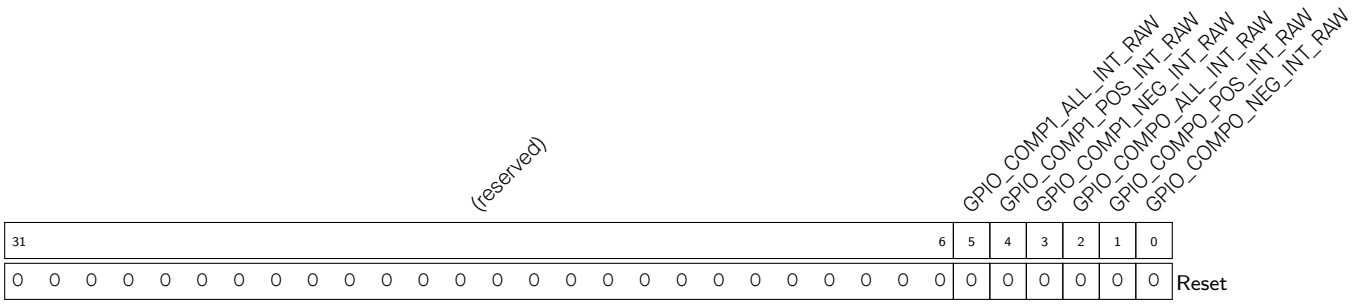
Register 10.38. GPIO_DATE_REG (0x07FC)



GPIO_DATE Version control register.

(R/W)

Register 10.39. GPIO_INT_RAW_REG (0x0700)



GPIO_COMPO_NEG_INT_RAW The raw interrupt status of GPIO_COMPO_NEG_INT. (R/WTC/SS)

GPIO_COMPO_POS_INT_RAW The raw interrupt status of GPIO_COMPO_POS_INT. (R/WTC/SS)

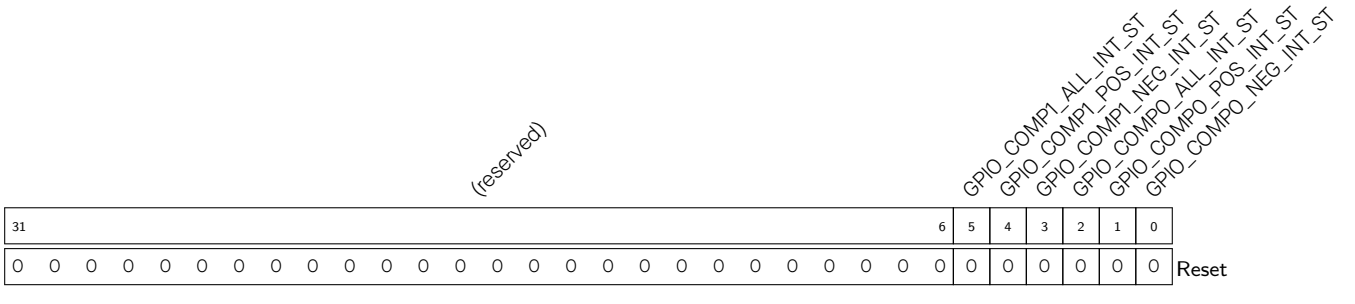
GPIO_COMPO_ALL_INT_RAW The raw interrupt status of GPIO_COMPO_ALL_INT. (R/WTC/SS)

GPIO_COMP1_NEG_INT_RAW The raw interrupt status of GPIO_COMP1_NEG_INT. (R/WTC/SS)

GPIO_COMP1_POS_INT_RAW The raw interrupt status of GPIO_COMP1_POS_INT. (R/WTC/SS)

GPIO_COMP1_ALL_INT_RAW The raw interrupt status of GPIO_COMP1_ALL_INT. (R/WTC/SS)

Register 10.40. GPIO_INT_ST_REG (0x0704)



GPIO_COMPO_NEG_INT_ST The masked interrupt status of GPIO_COMPO_NEG_INT. (RO)

GPIO_COMPO_POS_INT_ST The masked interrupt status of GPIO_COMPO_POS_INT. (RO)

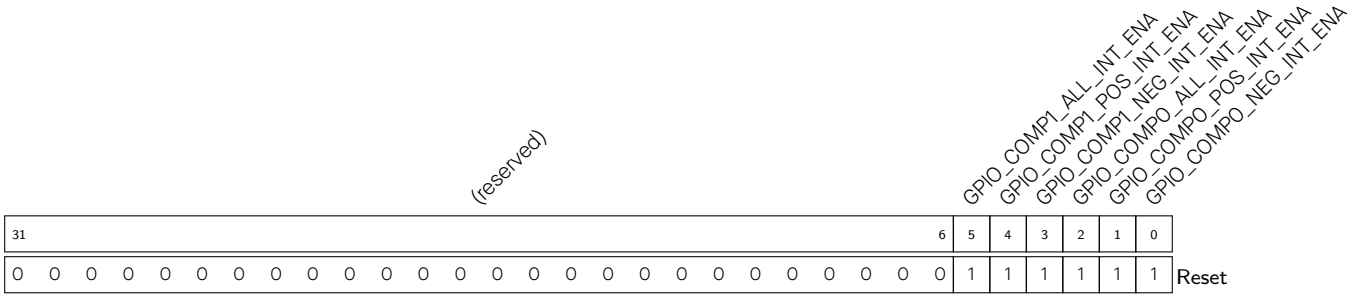
GPIO_COMPO_ALL_INT_ST The masked interrupt status of GPIO_COMPO_ALL_INT. (RO)

GPIO_COMP1_NEG_INT_ST The masked interrupt status of GPIO_COMP1_NEG_INT. (RO)

GPIO_COMP1_POS_INT_ST The masked interrupt status of GPIO_COMP1_POS_INT. (RO)

GPIO_COMP1_ALL_INT_ST The masked interrupt status of GPIO_COMP1_ALL_INT. (RO)

Register 10.41. GPIO_INT_ENA_REG (0x0708)



GPIO_COMP0_NEG_INT_ENA Write 1 to enable GPIO_COMP0_NEG_INT. (R/W)

GPIO_COMP0_POS_INT_ENA Write 1 to enable GPIO_COMP0_POS_INT. (R/W)

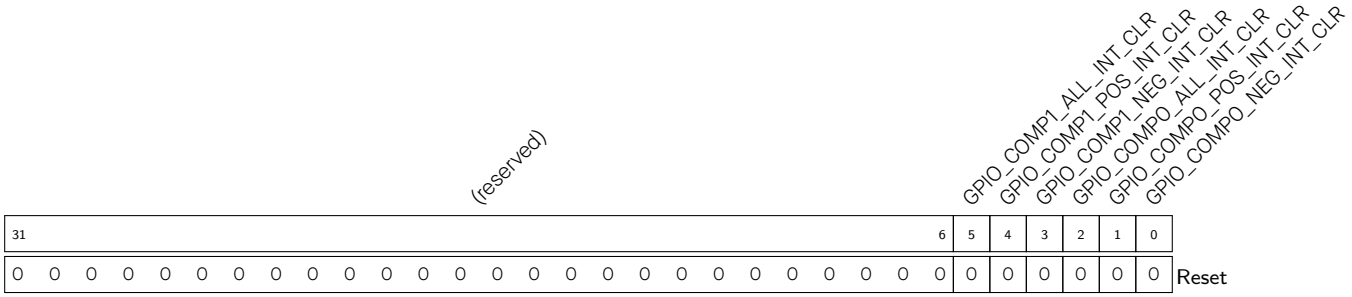
GPIO_COMP0_ALL_INT_ENA Write 1 to enable GPIO_COMP0_ALL_INT. (R/W)

GPIO_COMP1_NEG_INT_ENA Write 1 to enable GPIO_COMP1_NEG_INT. (R/W)

GPIO_COMP1_POS_INT_ENA Write 1 to enable GPIO_COMP1_POS_INT. (R/W)

GPIO_COMP1_ALL_INT_ENA Write 1 to enable GPIO_COMP1_ALL_INT. (R/W)

Register 10.42. GPIO_INT_CLR_REG (0x070C)



GPIO_COMP0_NEG_INT_CLR Write 1 to clear GPIO_COMP0_NEG_INT. (WT)

GPIO_COMP0_POS_INT_CLR Write 1 to clear GPIO_COMP0_POS_INT. (WT)

GPIO_COMP0_ALL_INT_CLR Write 1 to clear GPIO_COMP0_ALL_INT. (WT)

GPIO_COMP1_NEG_INT_CLR Write 1 to clear GPIO_COMP1_NEG_INT. (WT)

GPIO_COMP1_POS_INT_CLR Write 1 to clear GPIO_COMP1_POS_INT. (WT)

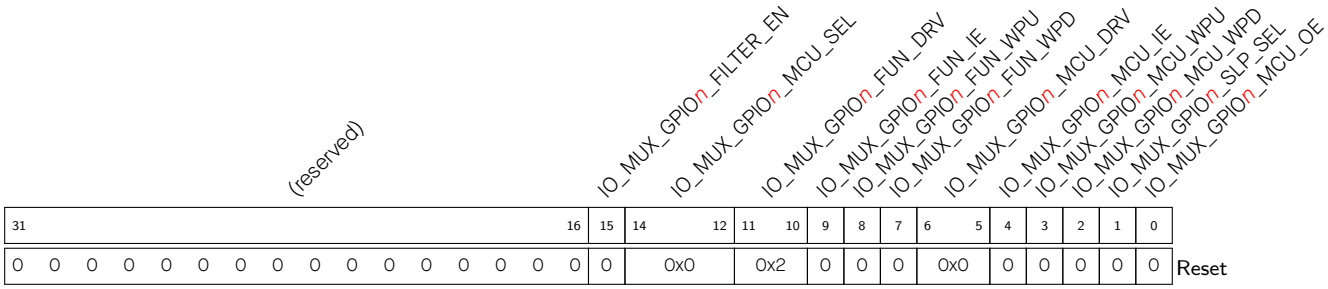
GPIO_COMP1_ALL_INT_CLR Write 1 to clear GPIO_COMP1_ALL_INT. (WT)

10.20.2 HP IO MUX Registers

The addresses in this section are relative to the HP IO MUX base address provided in Table 9.3-2 in Chapter 9 *System and Memory*.

For how to program reserved fields, please refer to Section IX .

Register 10.43. IO_MUX_GPIO*n*_REG (*n*: 0 - 54) (0x0004+4**n*)



IO_MUX_GPIO*n*_MCU_OE Configures whether or not to enable the output of GPIO*n* in sleep mode.

0: Disable

1: Enable

(R/W)

IO_MUX_GPIO*n*_SLP_SEL Configures whether or not to enter sleep mode for GPIO*n*.

0: Not enter

1: Enter

(R/W)

IO_MUX_GPIO*n*_MCU_WPD Configure whether or not to enable pull-down resistor of GPIO*n* during sleep mode.

0: Disable

1: Enable

(R/W)

IO_MUX_GPIO*n*_MCU_WPU Configures whether or not to enable pull-up resistor of GPIO*n* during sleep mode.

0: Disable

1: Enable

(R/W)

IO_MUX_GPIO*n*_MCU_IE Configures whether or not to enable the input of GPIO*n* during sleep mode.

0: Disable

1: Enable

(R/W)

IO_MUX_GPIO*n*_MCU_DRV Configures the drive strength of GPIO*n* during sleep mode.

0: ~5 mA

1: ~10 mA

2: ~20 mA

3: ~40 mA

(R/W)

Continued on the next page...

Register 10.43. IO_MUX_GPIO*n*_REG (*n*: 0 - 54) (0x0004+4**n*)

Continued from the previous page...

IO_MUX_GPIO*n*_FUN_WPD Configures whether or not to enable pull-down resistor of GPIO*n*.

0: Disable

1: Enable

(R/W)

IO_MUX_GPIO*n*_FUN_WPU Configures whether or not enable pull-up resistor of GPIO*n*.

0: Disable

1: Enable

(R/W)

IO_MUX_GPIO*n*_FUN_IE Configures whether or not to enable input of GPIO*n*.

0: Disable

1: Enable

(R/W)

IO_MUX_GPIO*n*_FUN_DRV Configures the drive strength of GPIO*n*.

0: ~5 mA

1: ~10 mA

2: ~20 mA

3: ~40 mA

(R/W)

IO_MUX_GPIO*n*_MCU_SEL Configures to select IO MUX function for this pin.

0: Select Function 0

1: Select Function 1

.....

(R/W)

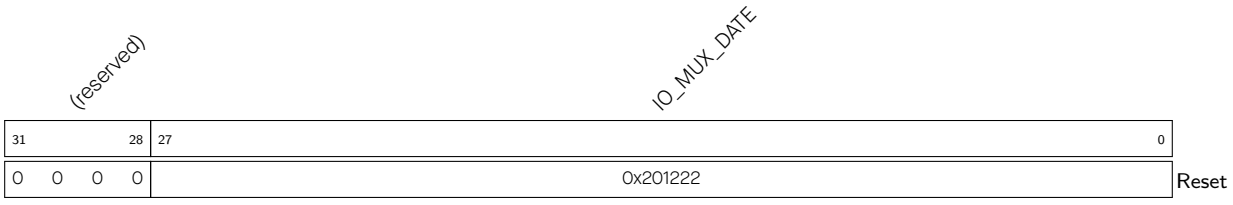
IO_MUX_GPIO*n*_FILTER_EN Configures whether or not to enable filter for pin input signals.

0: Disable

1: Enable

(R/W)

Register 10.44. IO_MUX_DATE_REG (0x0104)



IO_MUX_DATE Version control register.

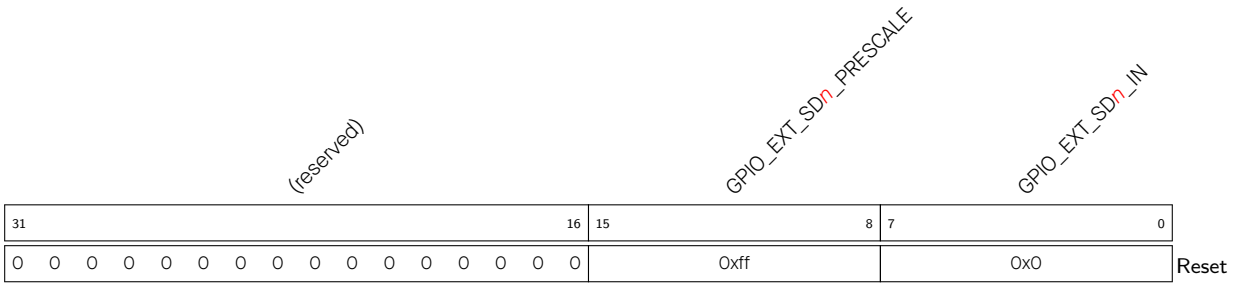
(R/W)

10.20.3 GPIO EXT Registers

The addresses in this section are relative to (HP GPIO matrix base address + 0x0F00). GPIO base address is provided in Table 9.3-2 in Chapter 9 *System and Memory*.

For how to program reserved fields, please refer to Section IX .

Register 10.45. GPIO_EXT_SIGMADELTA*n*_REG (*n*: 0 - 7) (0x0000+0x4n*)**



GPIO_EXT_SD*n*_IN (*n*: 0 - 7) Configures the duty cycle of sigma delta modulation output. (R/W)

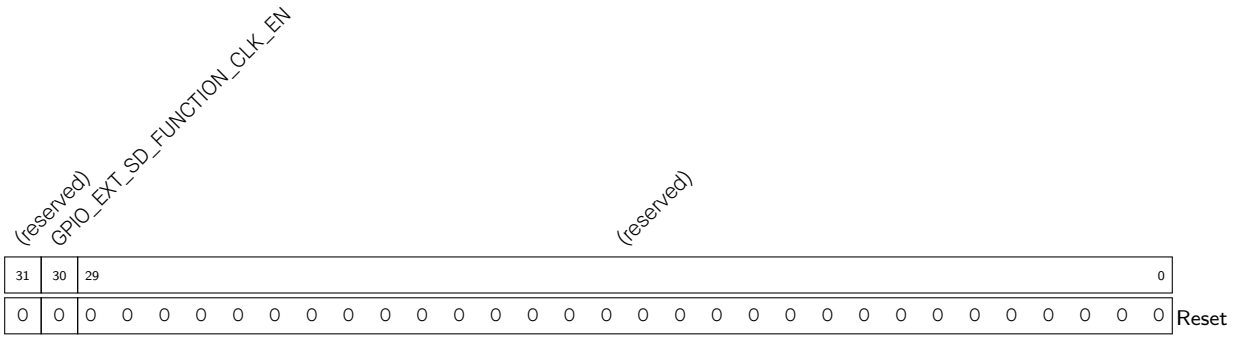
GPIO_EXT_SD*n*_PRESCALE (*n*: 0 - 7) Configures the divider value to divide HP IO MUX operating clock. Can be 0 ~ 255.

0: Do not divide

1 ~ 255: The clock is divided by 2 ~ 256

(R/W)

Register 10.46. GPIO_EXT_SIGMADELTA_MISC_REG (0x0024)



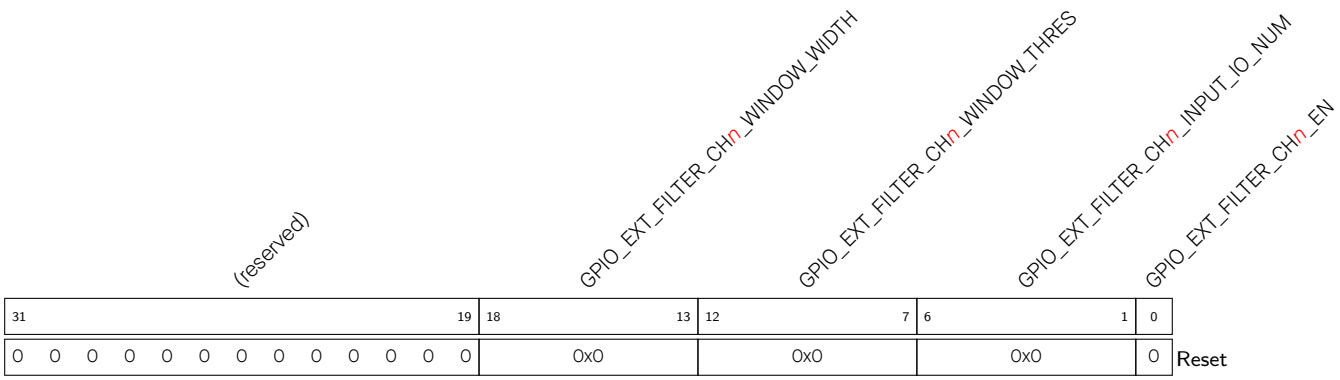
GPIO_EXT_SD_FUNCTION_CLK_EN Configures whether or not to enable the clock for sigma delta modulation.

0: Not enable

1: Enable

(R/W)

Register 10.47. GPIO_EXT_GLITCH_FILTER_CH*n*_REG (*n*: 0 - 7) (0x0030+0x4n*)**



GPIO_EXT_FILTER_CH*n*_EN Configures whether or not to enable channel *n* of Glitch Filter.

0: Not enable

1: Enable

(R/W)

GPIO_EXT_FILTER_CH*n*_INPUT_IO_NUM Configures to select the input GPIO for Glitch Filter.

0: Select GPIO0

1: Select GPIO1

.....

53: Select GPIO53

54: Select GPIO54

(R/W)

GPIO_EXT_FILTER_CH*n*_WINDOW_THRES Configures the window threshold for Glitch Filter. The window threshold should be less than or equal to GPIO_EXT_FILTER_CH*n*_WINDOW_WIDTH.

Measurement unit: HP IO MUX operating clock cycle

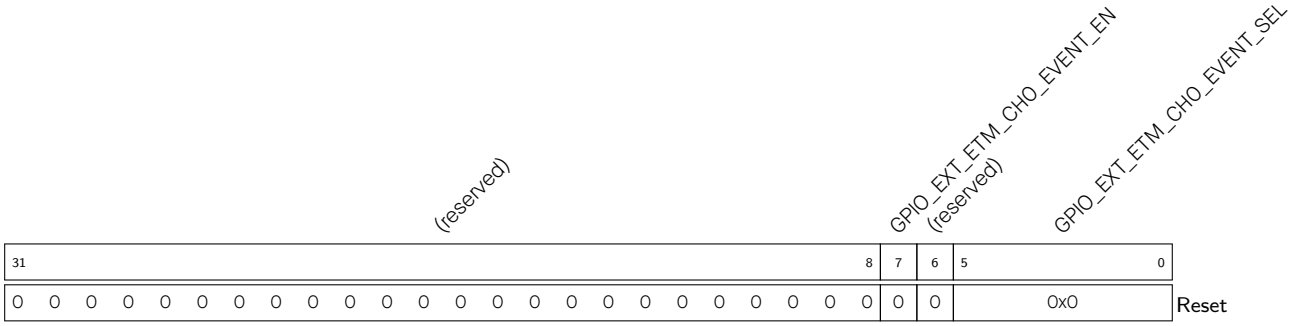
(R/W)

GPIO_EXT_FILTER_CH*n*_WINDOW_WIDTH Configures the window width for Glitch Filter. The effective value of window width is 0 ~ 62. 63 is a reserved value and can not be used.

Measurement unit: HP IO MUX operating clock cycle

(R/W)

Register 10.48. GPIO_EXT_ETM_EVENT_CH*n*_CFG_REG (*n*: 0 - 7) (0x0060+0x4n*)**



GPIO_EXT_ETM_CH*n*_EVENT_SEL Configures to select GPIO for ETM event channel.

0: Select GPIO0

1: Select GPIO1

.....

53: Select GPIO53

54: Select GPIO54

(R/W)

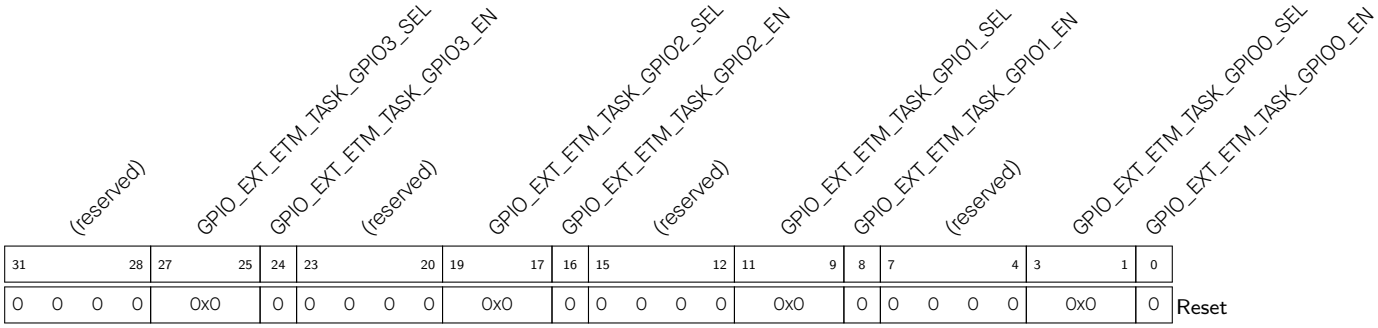
GPIO_EXT_ETM_CH*n*_EVENT_EN Configures whether or not to enable ETM event send.

0: Not enable

1: Enable

(R/W)

Register 10.49. GPIO_EXT_ETM_TASK_P0_CFG_REG (0x00A0)



GPIO_EXT_ETM_TASK_GPIO*n*_EN (*n*: 0 - 3) Configures whether or not to enable GPIO*n* to response ETM task.

0: Not enable

1: Enable

(R/W)

GPIO_EXT_ETM_TASK_GPIO*n*_SEL (*n*: 0 - 3) Configures to select an ETM task channel for GPIO*n*.

0: Select channel 0

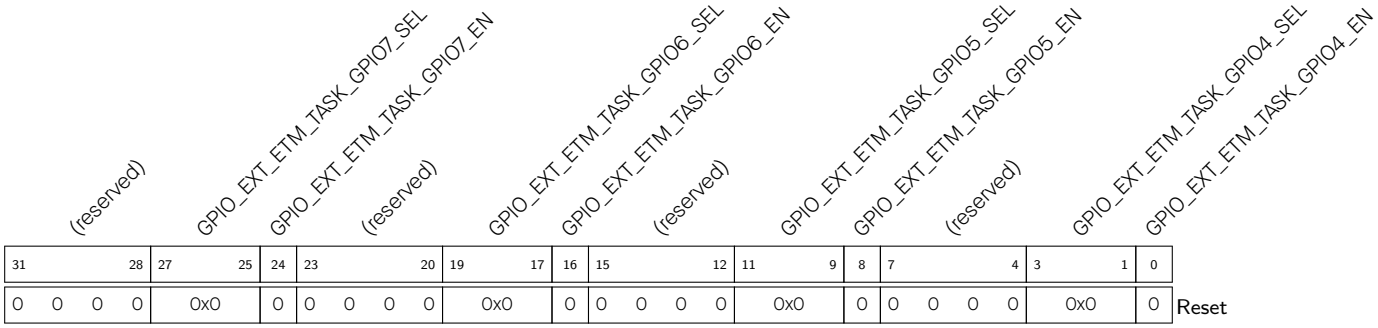
1: Select channel 1

.....

7: Select channel 7

(R/W)

Register 10.50. GPIO_EXT_ETM_TASK_P1_CFG_REG (0x00A4)



GPIO_EXT_ETM_TASK_GPIO*n*_EN (*n*: 4 - 7) Configures whether or not to enable GPIO*n* to response ETM task.

0: Not enable

1: Enable

(R/W)

GPIO_EXT_ETM_TASK_GPIO*n*_SEL (*n*: 4 - 7) Configures to select an ETM task channel for GPIO*n*.

0: Select channel 0

1: Select channel 1

.....

7: Select channel 7

(R/W)

Register 10.51. GPIO_EXT_ETM_TASK_P2_CFG_REG (0x00A8)



GPIO_EXT_ETM_TASK_GPIO*n*_EN (*n*: 8 - 11) Configures whether or not to enable GPIO*n* to response ETM task.

0: Not enable

1: Enable

(R/W)

GPIO_EXT_ETM_TASK_GPIO*n*_SEL (*n*: 8 - 11) Configures to select an ETM task channel for GPIO*n*.

0: Select channel 0

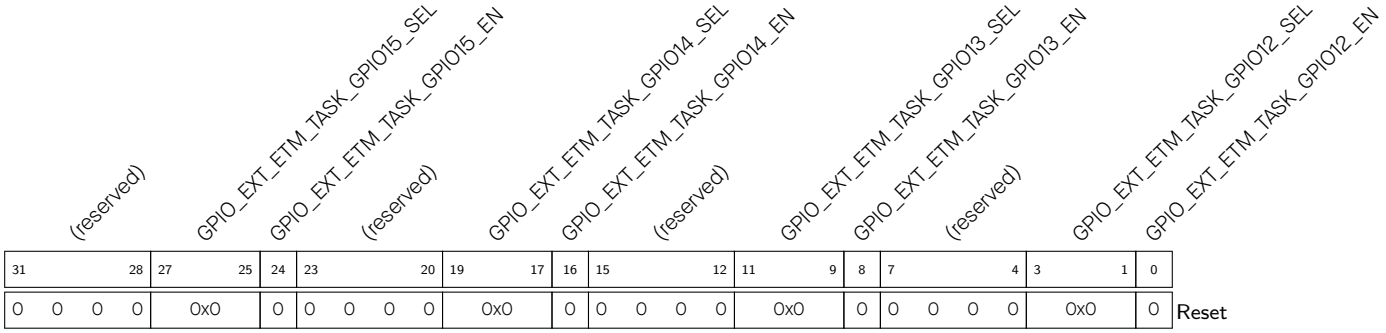
1: Select channel 1

.....

7: Select channel 7

(R/W)

Register 10.52. GPIO_EXT_ETM_TASK_P3_CFG_REG (0x00AC)



GPIO_EXT_ETM_TASK_GPIO*n*_EN (*n*: 12 - 15) Configures whether or not to enable GPIO*n* to response ETM task.

0: Not enable

1: Enable

(R/W)

GPIO_EXT_ETM_TASK_GPIO*n*_SEL (*n*: 12 - 15) Configures to select an ETM task channel for GPIO*n*.

0: Select channel 0

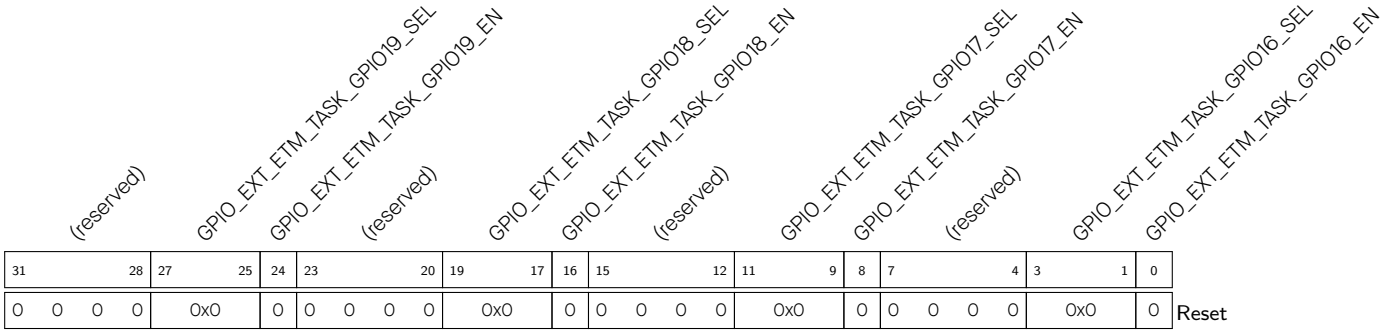
1: Select channel 1

.....

7: Select channel 7

(R/W)

Register 10.53. GPIO_EXT_ETM_TASK_P4_CFG_REG (0x00B0)



GPIO_EXT_ETM_TASK_GPIO*n*_EN (*n*: 16 - 19) Configures whether or not to enable GPIO*n* to response ETM task.

0: Not enable

1: Enable

(R/W)

GPIO_EXT_ETM_TASK_GPIO*n*_SEL (*n*: 16 - 19) Configures to select an ETM task channel for GPIO*n*.

0: Select channel 0

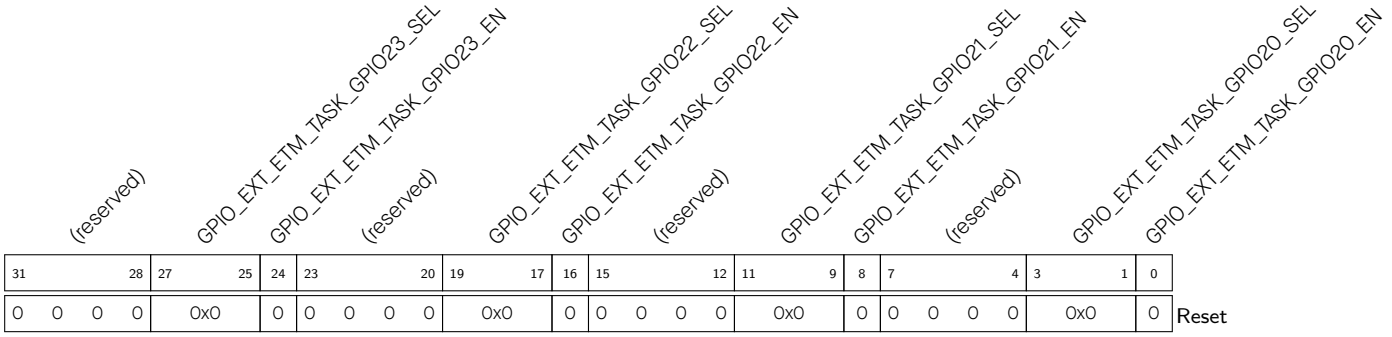
1: Select channel 1

.....

7: Select channel 7

(R/W)

Register 10.54. GPIO_EXT_ETM_TASK_P5_CFG_REG (0x00B4)



GPIO_EXT_ETM_TASK_GPIO*n*_EN (*n*: 20 - 23) Configures whether or not to enable GPIO*n* to response ETM task.

0: Not enable

1: Enable

(R/W)

GPIO_EXT_ETM_TASK_GPIO*n*_SEL (*n*: 20 - 23) Configures to select an ETM task channel for GPIO*n*.

0: Select channel 0

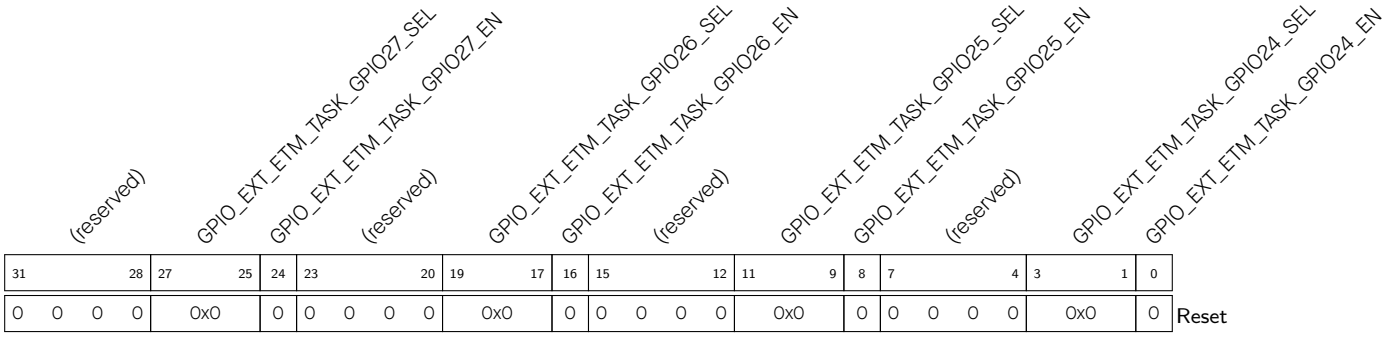
1: Select channel 1

.....

7: Select channel 7

(R/W)

Register 10.55. GPIO_EXT_ETM_TASK_P6_CFG_REG (0x00B8)



GPIO_EXT_ETM_TASK_GPIO*n*_EN (*n*: 24 - 27) Configures whether or not to enable GPIO*n* to response ETM task.

0: Not enable

1: Enable

(R/W)

GPIO_EXT_ETM_TASK_GPIO*n*_SEL (*n*: 24 - 27) Configures to select an ETM task channel for GPIO*n*.

0: Select channel 0

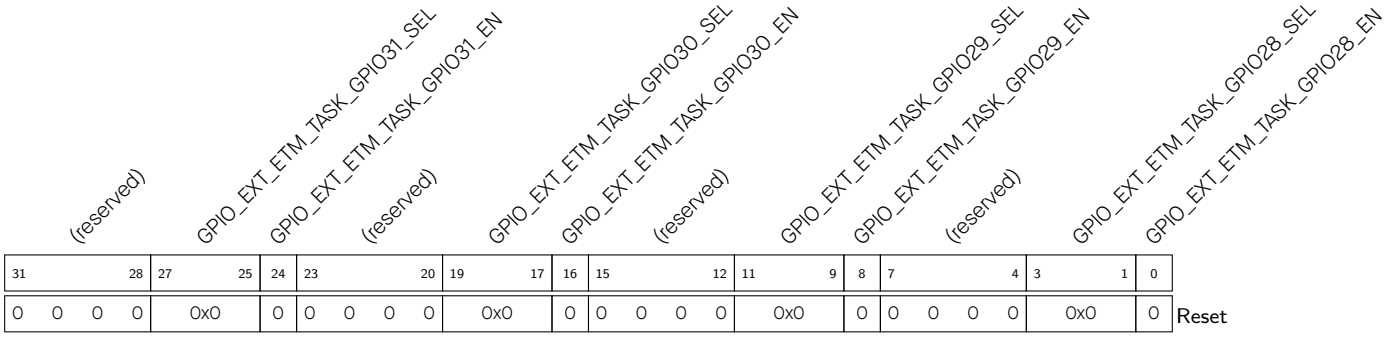
1: Select channel 1

.....

7: Select channel 7

(R/W)

Register 10.56. GPIO_EXT_ETM_TASK_P7_CFG_REG (0x00BC)



GPIO_EXT_ETM_TASK_GPIO*n*_EN (*n*: 28 - 31) Configures whether or not to enable GPIO*n* to response ETM task.

0: Not enable

1: Enable

(R/W)

GPIO_EXT_ETM_TASK_GPIO*n*_SEL (*n*: 28 - 31) Configures to select an ETM task channel for GPIO*n*.

0: Select channel 0

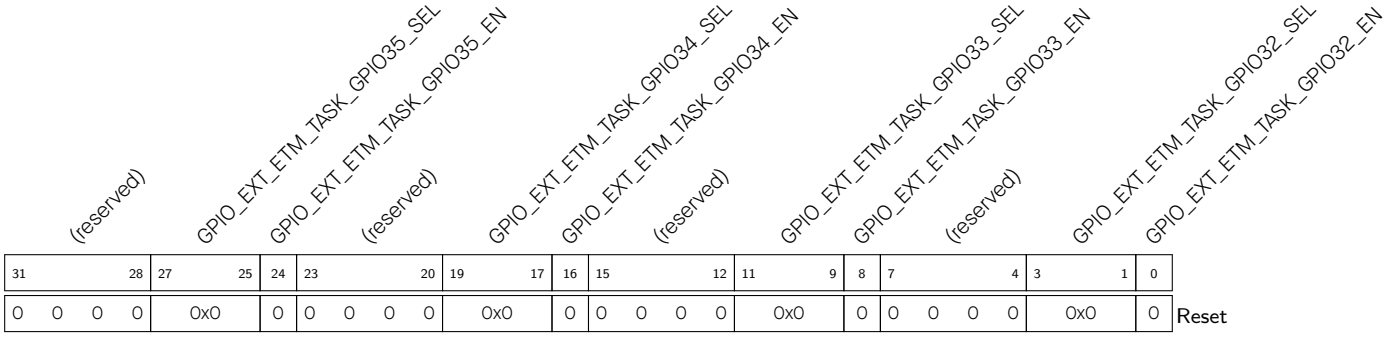
1: Select channel 1

.....

7: Select channel 7

(R/W)

Register 10.57. GPIO_EXT_ETM_TASK_P8_CFG_REG (0x00C0)



GPIO_EXT_ETM_TASK_GPIO*n*_EN (*n*: 32 - 35) Configures whether or not to enable GPIO*n* to response ETM task.

0: Not enable

1: Enable

(R/W)

GPIO_EXT_ETM_TASK_GPIO*n*_SEL (*n*: 32 - 35) Configures to select an ETM task channel for GPIO*n*.

0: Select channel 0

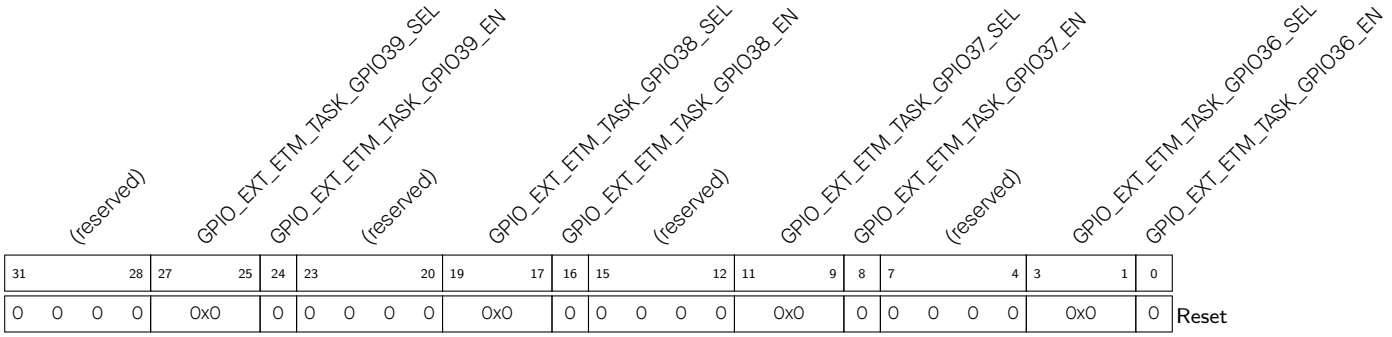
1: Select channel 1

.....

7: Select channel 7

(R/W)

Register 10.58. GPIO_EXT_ETM_TASK_P9_CFG_REG (0x00C4)



GPIO_EXT_ETM_TASK_GPIO*n*_EN (*n*: 36 - 39) Configures whether or not to enable GPIO*n* to response ETM task.

0: Not enable

1: Enable

(R/W)

GPIO_EXT_ETM_TASK_GPIO*n*_SEL (*n*: 36 - 39) Configures to select an ETM task channel for GPIO*n*.

0: Select channel 0

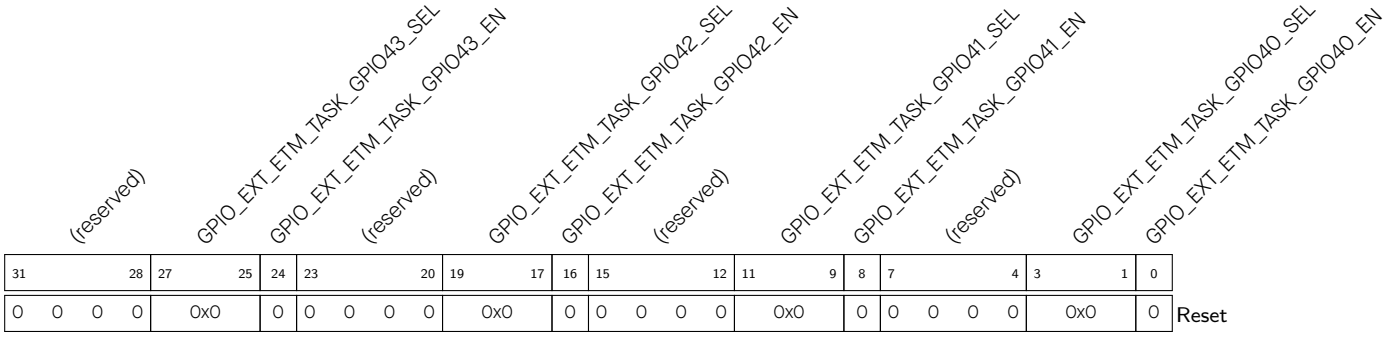
1: Select channel 1

.....

7: Select channel 7

(R/W)

Register 10.59. GPIO_EXT_ETM_TASK_P10_CFG_REG (0x00C8)



GPIO_EXT_ETM_TASK_GPIO*n*_EN (*n*: 40 - 43) Configures whether or not to enable GPIO*n* to response ETM task.

0: Not enable

1: Enable

(R/W)

GPIO_EXT_ETM_TASK_GPIO*n*_SEL (*n*: 40 - 43) Configures to select an ETM task channel for GPIO*n*.

0: Select channel 0

1: Select channel 1

.....

7: Select channel 7

(R/W)

Register 10.60. GPIO_EXT_ETM_TASK_P11_CFG_REG (0x00CC)



GPIO_EXT_ETM_TASK_GPIO*n*_EN (*n*: 44 - 47) Configures whether or not to enable GPIO*n* to response ETM task.

0: Not enable

1: Enable

(R/W)

GPIO_EXT_ETM_TASK_GPIO*n*_SEL (*n*: 44 - 47) Configures to select an ETM task channel for GPIO*n*.

0: Select channel 0

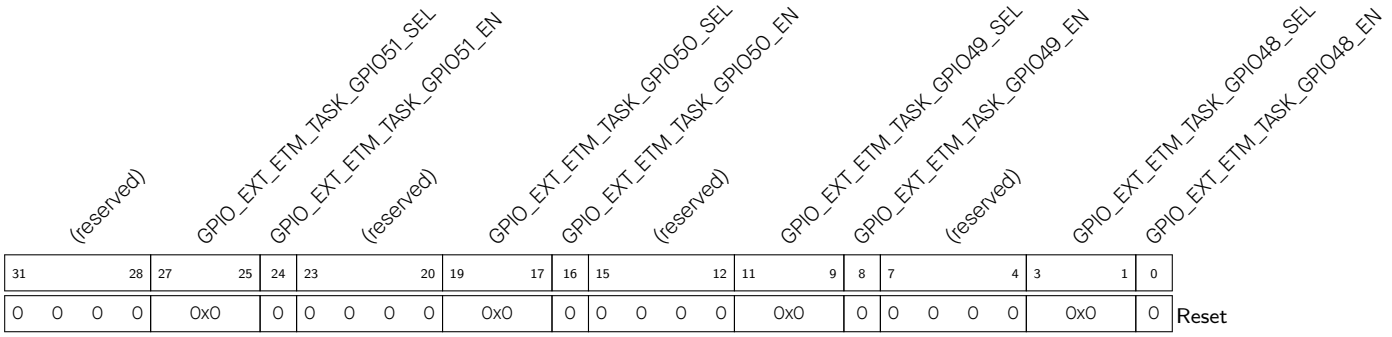
1: Select channel 1

.....

7: Select channel 7

(R/W)

Register 10.61. GPIO_EXT_ETM_TASK_P12_CFG_REG (0x00D0)



GPIO_EXT_ETM_TASK_GPIO*n*_EN (*n*: 48 - 51) Configures whether or not to enable GPIO*n* to response ETM task.

0: Not enable

1: Enable

(R/W)

GPIO_EXT_ETM_TASK_GPIO*n*_SEL (*n*: 48 - 51) Configures to select an ETM task channel for GPIO*n*.

0: Select channel 0

1: Select channel 1

.....

7: Select channel 7

(R/W)

Register 10.62. GPIO_EXT_ETM_TASK_P13_CFG_REG (0x00D4)



GPIO_EXT_ETM_TASK_GPIO*n*_EN (*n*: 52 - 54) Configures whether or not to enable GPIO*n* to response ETM task.

0: Not enable

1: Enable

(R/W)

GPIO_EXT_ETM_TASK_GPIO*n*_SEL (*n*: 52 - 54) Configures to select an ETM task channel for GPIO*n*.

0: Select channel 0

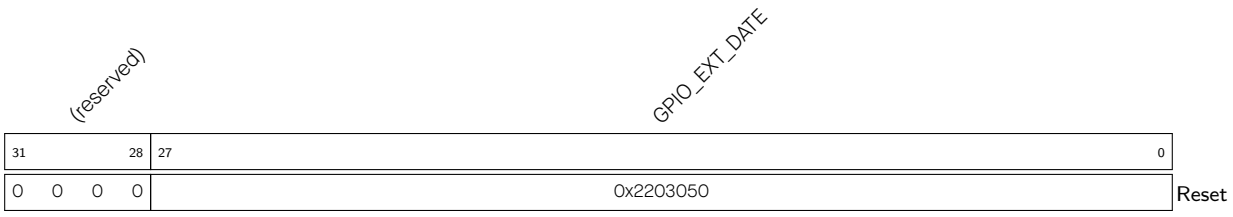
1: Select channel 1

.....

7: Select channel 7

(R/W)

Register 10.63. GPIO_EXT_VERSION_REG (0x00FC)



GPIO_EXT_DATE Version control register.

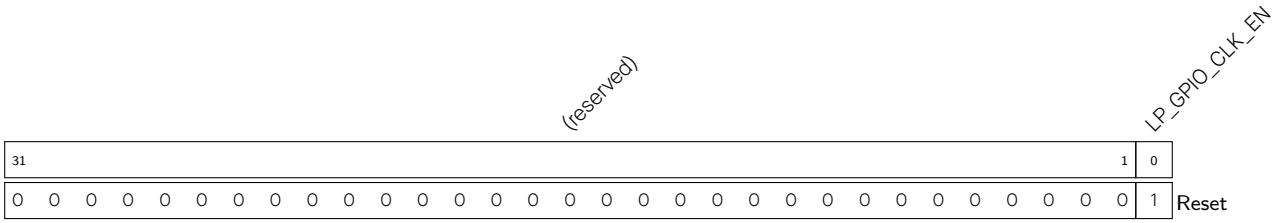
(R/W)

10.20.4 LP GPIO Matrix Registers

The addresses in this section are relative to LP GPIO matrix base address provided in Table 9.3-2 in Chapter 9 *System and Memory*.

For how to program reserved fields, please refer to Section IX .

Register 10.64. LP_GPIO_CLK_EN_REG (0x0000)



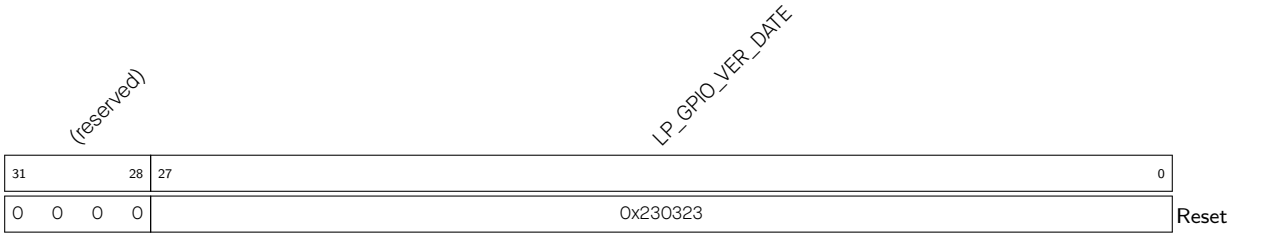
LP_GPIO_CLK_EN Configures whether or not to enable clock gate.

0: Not enable

1: Enable, the clock is free running.

(R/W)

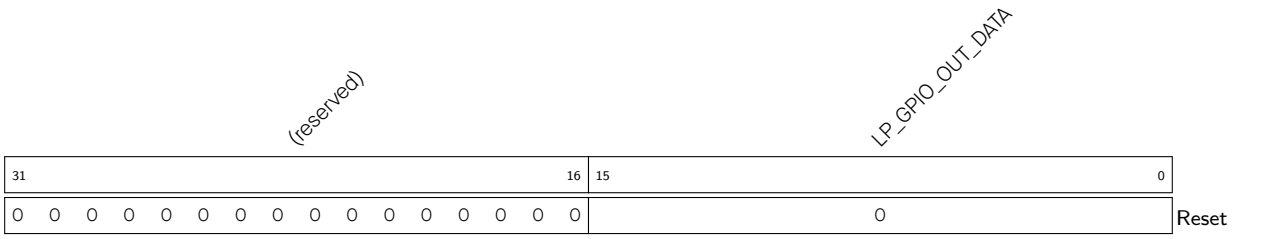
Register 10.65. LP_GPIO_VER_DATE_REG (0x0004)



LP_GPIO_VER_DATE Version control register.

(R/W)

Register 10.66. LP_GPIO_OUT_REG (0x0008)



LP_GPIO_OUT_DATA Configures the output value of GPIO0 ~ GPIO15 output in simple GPIO output mode.

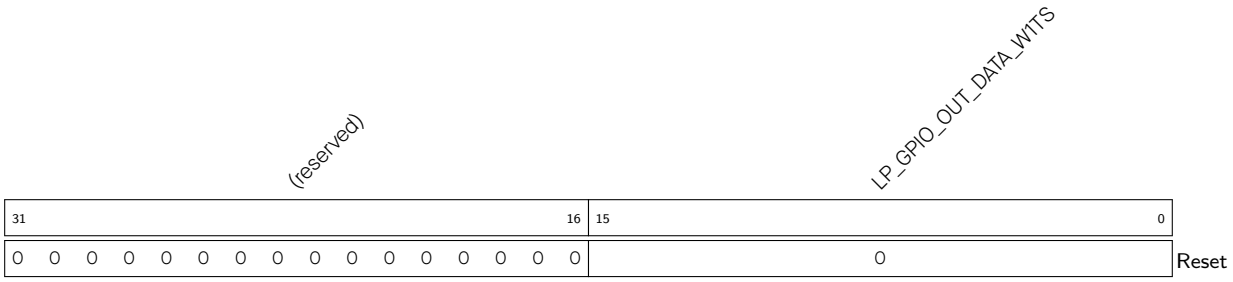
0: Low level

1: High level

The value of bit0 ~ bit15 correspond to the output value of GPIO0 ~ GPIO15 respectively. Bit16 ~ bit31 are invalid.

(R/W/WTC)

Register 10.67. LP_GPIO_OUT_W1TS_REG (0x000C)



LP_GPIO_OUT_DATA_W1TS Configures whether or not to set the output register LP_GPIO_OUT_REG of GPIO0 ~ GPIO15.

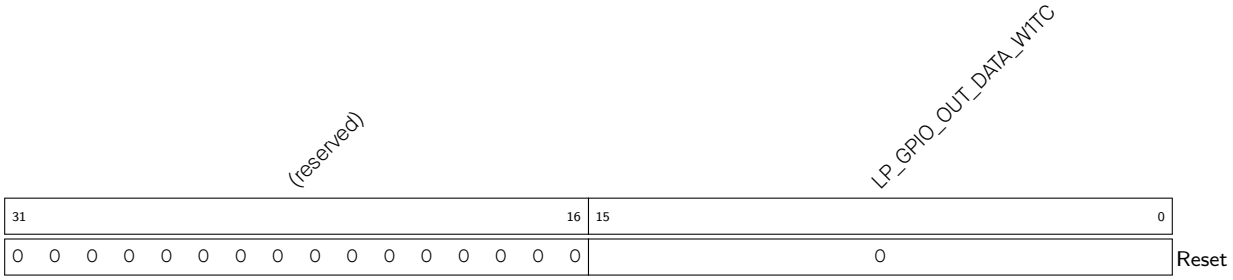
0: Not set

1: The corresponding bit in LP_GPIO_OUT_REG will be set to 1

Bit0 ~ bit15 are corresponding to GPIO0 ~ GPIO15. Bit16 ~ bit31 are invalid. Recommended operation: use this register to set LP_GPIO_OUT_REG.

(WT)

Register 10.68. LP_GPIO_OUT_W1TC_REG (0x0010)



LP_GPIO_OUT_DATA_W1TC Configures whether or not to clear the output register LP_GPIO_OUT_REG of GPIO0 ~ GPIO15 output.

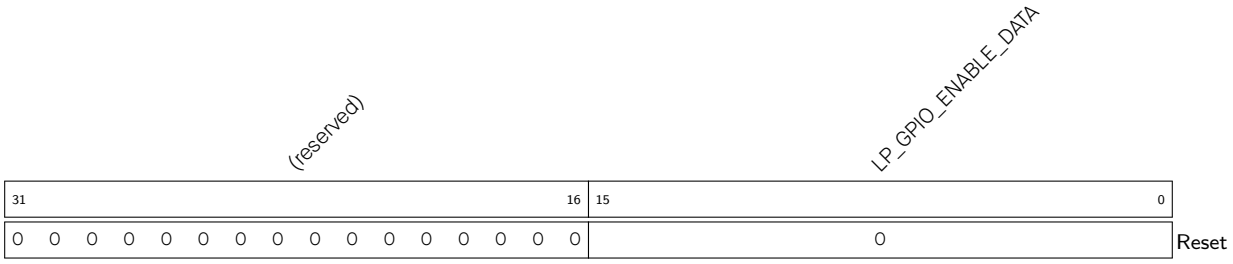
0: Not clear

1: The corresponding bit in LP_GPIO_OUT_REG will be cleared.

Bit0 ~ bit15 are corresponding to GPIO0 ~ GPIO15. Bit16 ~ bit31 are invalid. Recommended operation: use this register to clear LP_GPIO_OUT_REG.

(WT)

Register 10.69. LP_GPIO_ENABLE_REG (0x0014)



LP_GPIO_ENABLE_DATA Configures whether or not to enable the output of GPIO0 ~ GPIO15.

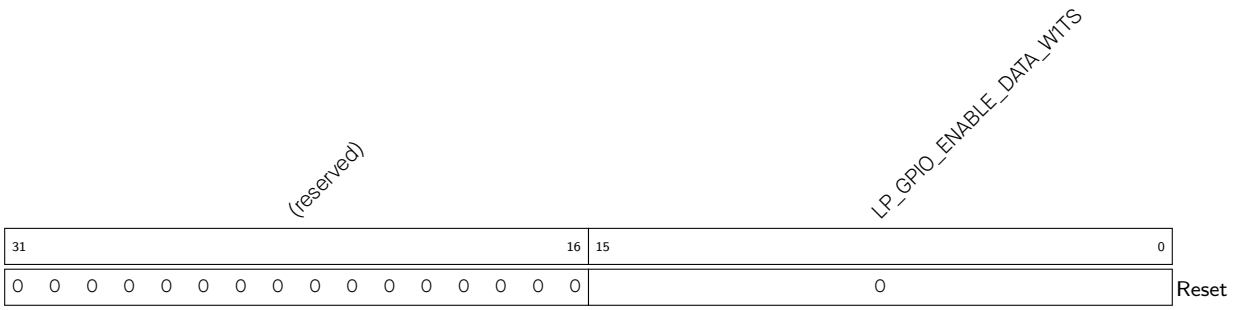
0: Not enable

1: Enable

Bit0 ~ bit15 are corresponding to GPIO0 ~ GPIO15. Bit16 ~ bit31 are invalid.

(R/W/WTC)

Register 10.70. LP_GPIO_ENABLE_W1TS_REG (0x0018)



LP_GPIO_ENABLE_DATA_W1TS Configures whether or not to set the output enable register LP_GPIO_ENABLE_REG of GPIO0 ~ GPIO15.

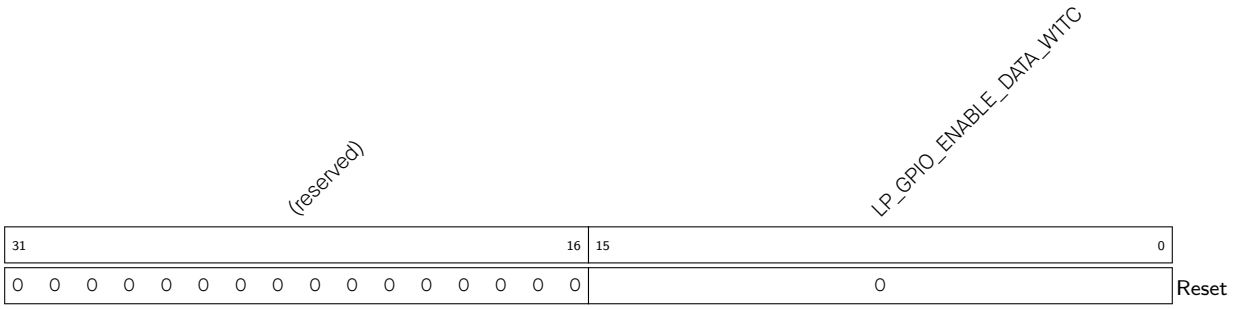
0: Not set

1: The corresponding bit in LP_GPIO_ENABLE_REG will be set to 1

Bit0 ~ bit15 are corresponding to GPIO0 ~ GPIO15. Bit16 ~ bit31 are invalid. Recommended operation: use this register to set LP_GPIO_ENABLE_REG.

(WT)

Register 10.71. LP_GPIO_ENABLE_W1TC_REG (0x001C)



LP_GPIO_ENABLE_DATA_W1TC Configures whether or not to clear the output enable register LP_GPIO_ENABLE_REG of GPIO0 ~ GPIO15.

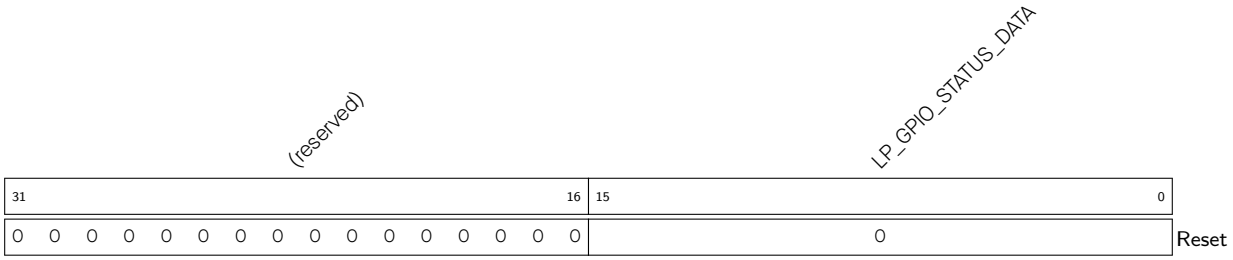
0: Not clear

1: The corresponding bit in LP_GPIO_ENABLE_REG will be cleared

Bit0 ~ bit15 are corresponding to GPIO0 ~ 15. Bit16 ~ bit31 are invalid. Recommended operation: use this register to clear LP_GPIO_ENABLE_REG.

(WT)

Register 10.72. LP_GPIO_STATUS_REG (0x0020)



LP_GPIO_STATUS_DATA The interrupt status of GPIO0 ~ GPIO15, can be configured by the software.

Bit0 ~ bit15 are corresponding to GPIO0 ~ GPIO15. Bit16 ~ bit31 are invalid.

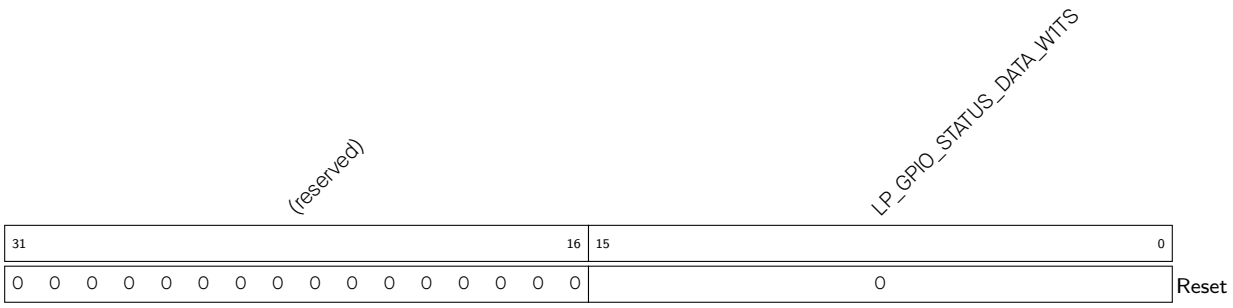
Each bit represents the status of its corresponding GPIO:

0: Represents the GPIO does not generate the interrupt configured by LP_GPIO_PIN*n*_INT_TYPE, or this bit is configured to 0 by the software.

1: Represents the GPIO generates the interrupt configured by LP_GPIO_PIN*n*_INT_TYPE, or this bit is configured to 1 by the software.

(R/W/WTC)

Register 10.73. LP_GPIO_STATUS_W1TS_REG (0x0024)



LP_GPIO_STATUS_DATA_W1TS Configures whether or not to set the interrupt status register LP_GPIO_STATUS_DATA of GPIO0 ~ GPIO15.

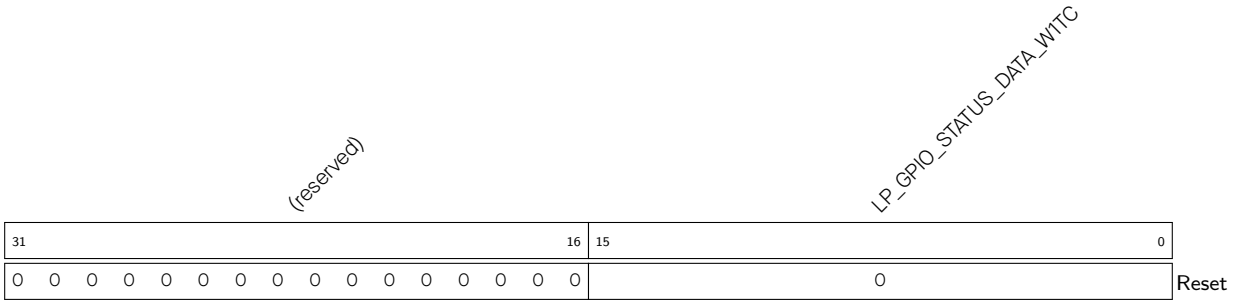
Bit0 ~ bit15 are corresponding to GPIO0 ~ GPIO15. Bit16 ~ bit31 are invalid.

If the value 1 is written to a bit here, the corresponding bit in LP_GPIO_STATUS_DATA will be set to 1.

Recommended operation: use this register to set LP_GPIO_STATUS_DATA.

(WT)

Register 10.74. LP_GPIO_STATUS_W1TC_REG (0x0028)



LP_GPIO_STATUS_DATA_W1TC Configures whether or not to clear the interrupt status register LP_GPIO_STATUS_DATA of GPIO0 ~ GPIO15.

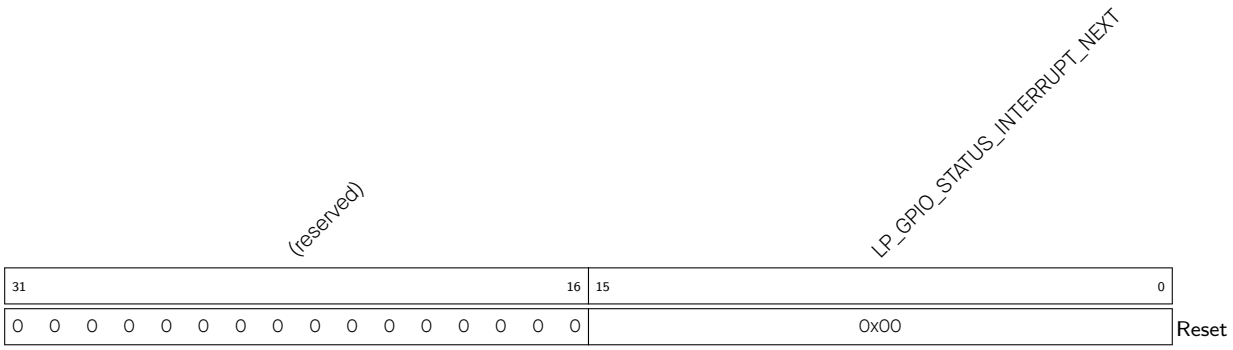
Bit0 ~ bit15 are corresponding to GPIO0 ~ GPIO15. Bit16 ~ bit31 are invalid.

If the value 1 is written to a bit here, the corresponding bit in LP_GPIO_STATUS_DATA will be cleared.

Recommended operation: use this register to clear LP_GPIO_STATUS_DATA.

(WT)

Register 10.75. LP_GPIO_STATUS_NEXT_REG (0x002C)



LP_GPIO_STATUS_INTERRUPT_NEXT Represents the interrupt source signal of GPIO0 ~ GPIO15. Bit0 ~ bit15 are corresponding to GPIO0 ~ 15. Bit16 ~ bit31 are invalid. Each bit represents:

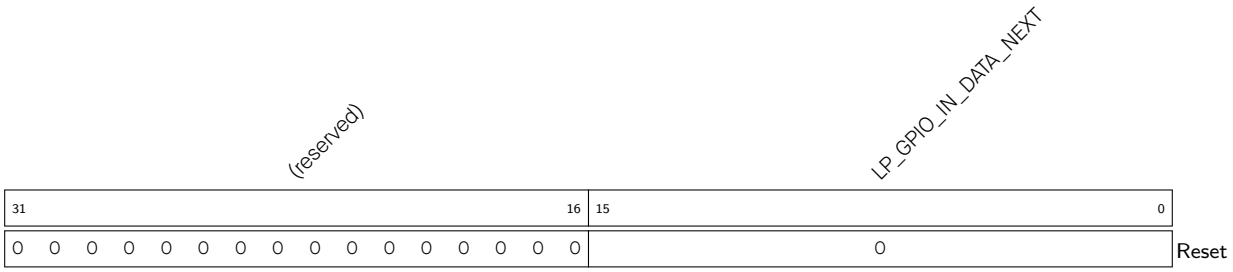
0: The GPIO does not generate the interrupt configured by LP_GPIO_PIN*n*_INT_TYPE.

1: The GPIO generates an interrupt configured by LP_GPIO_PIN*n*_INT_TYPE.

The interrupt could be rising edge interrupt, falling edge interrupt, level sensitive interrupt and any edge interrupt.

(RO)

Register 10.76. LP_GPIO_IN_REG (0x0030)



LP_GPIO_IN_DATA_NEXT Represents the input value of GPIO0 ~ GPIO15. Each bit represents a pin input value:

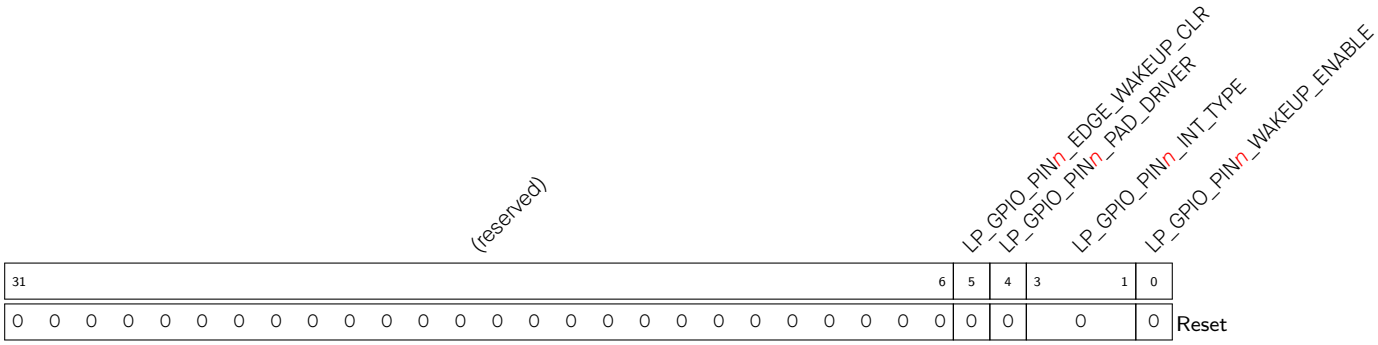
0: Low level

1: High level

Bit0 ~ bit15 are corresponding to GPIO0 ~ GPIO15. Bit16 ~ bit31 are invalid.

(RO)

Register 10.77. LP_GPIO_PIN*n*_REG (*n*: 0 - 15) (0x0034+4n*)**



LP_GPIO_PIN*n*_WAKEUP_ENABLE Configures whether or not to enable GPIO wake-up function.

0: Disable

1: Enable

This function only wakes up system from Deep-sleep.

(R/W)

LP_GPIO_PIN*n*_INT_TYPE Configures GPIO interrupt type.

0: GPIO interrupt disabled

1: Rising edge trigger

2: Falling edge trigger

3: Any edge trigger

4: Low level trigger

5: High level trigger

(R/W)

LP_GPIO_PIN*n*_PAD_DRIVER Configures to select pin drive mode.

0: Normal output

1: Open drain output

(R/W)

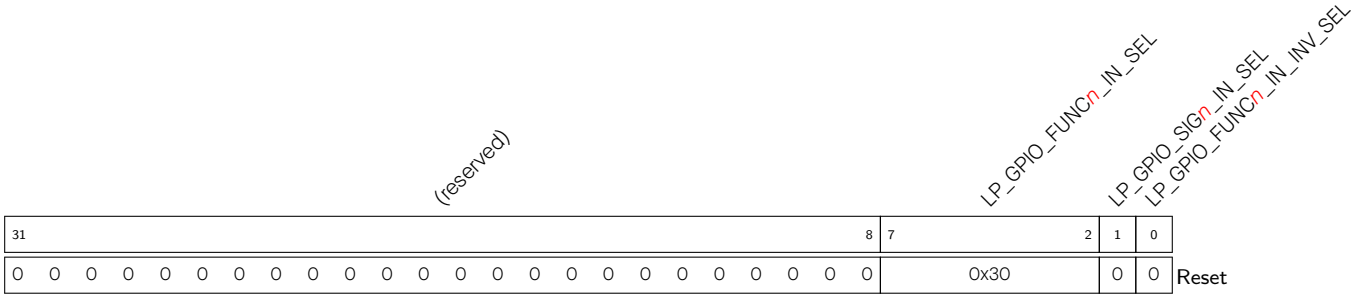
LP_GPIO_PIN*n*_EDGE_WAKEUP_CLR Configures to clear edge-triggered GPIO wake-up source in PMU.

1: The corresponding bit in LP_PMU_xxx will be cleared.

0: No effect.

(WT)

Register 10.78. LP_GPIO_FUNC*n*_IN_SEL_CFG_REG (*n*: 0 - 15) (0x0074+4n*)**



LP_GPIO_FUNC*n*_IN_INV_SEL Configures whether or not to invert the input value.

0: Not invert

1: Invert

(R/W)

LP_GPIO_SIG*n*_IN_SEL Configures whether or not to route signals via LP GPIO matrix.

0: Bypass LP GPIO matrix, i.e., connect signals directly to peripheral configured in LP IO MUX.

1: Route signals via LP GPIO matrix.

(R/W)

LP_GPIO_FUNC*n*_IN_SEL Configures to select a pin from the 16 LP GPIO pins to connect the input signal *n*.

0: Select GPIO0

1: Select GPIO1

.....

14: Select GPIO14

15: Select GPIO15

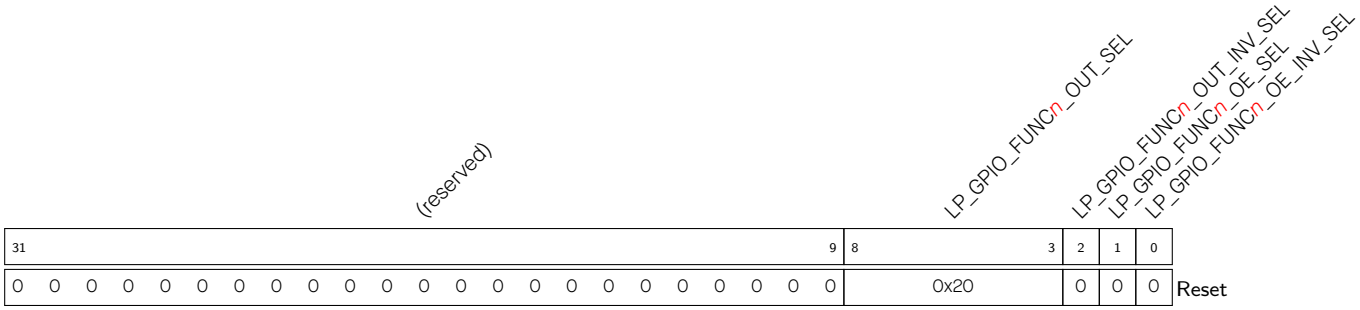
16 ~ 31: Invalid

32 ~ 47: A constantly low input

48 ~ 63: A constantly high input

(R/W)

Register 10.79. LP_GPIO_FUNC*n*_OUT_SEL_CFG_REG (*n*: 0 - 15) (0x00F4+4**n*)



LP_GPIO_FUNC*n*_OE_INV_SEL Configures whether or not to invert the output enable signal.

0: Not invert

1: Invert

(R/W)

LP_GPIO_FUNC*n*_OE_SEL Configures to select the source of output enable signal.

0: Use output enable signal from peripheral.

1: Force the output enable signal to be sourced from bit*n* of LP_GPIO_ENABLE_REG.

(R/W)

LP_GPIO_FUNC*n*_OUT_INV_SEL Configures whether or not to invert the output value.

0: Not invert

1: Invert

(R/W)

LP_GPIO_FUNC*n*_OUT_SEL Configures to select a signal *Y* (0 <= *Y* < 32) from 31 peripheral signals to be output to GPIO*n*.

0: Select signal 0

1: Select signal 1

.....

30: Select signal 30

31: Select signal 31

Or

32: Bit*n* of LP_GPIO_OUT_REG and LP_GPIO_ENABLE_REG are selected as the output value and output enable.

For the detailed signal list, see Table 10.13-1.

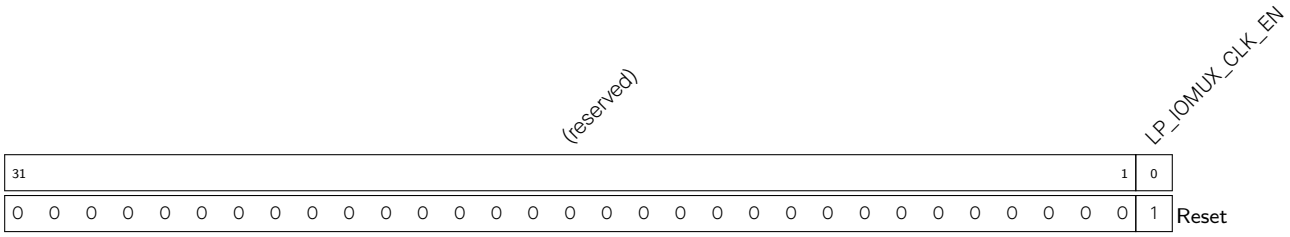
(R/W)

10.20.5 LP IO MUX Registers

The addresses in this section are relative to LP IO MUX base address provided in Table 9.3-2 in Chapter 9 *System and Memory*.

For how to program reserved fields, please refer to Section IX .

Register 10.80. LP_IOMUX_CLK_EN_REG (0x0000)



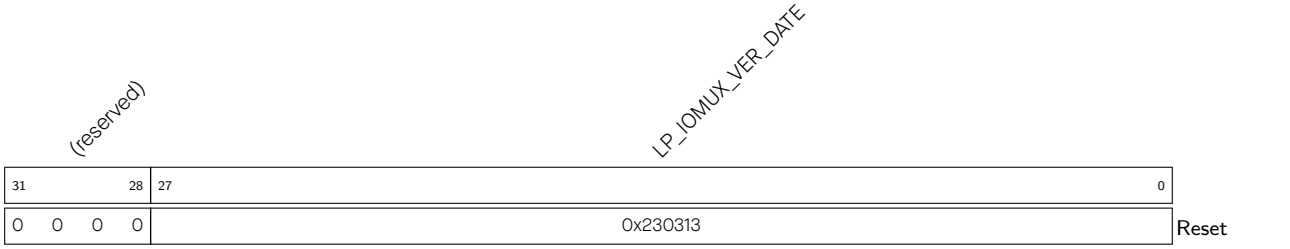
LP_IOMUX_CLK_EN Configures whether or not to enable clock gate.

0: Not enable

1: Enable, the clock is free running.

(R/W)

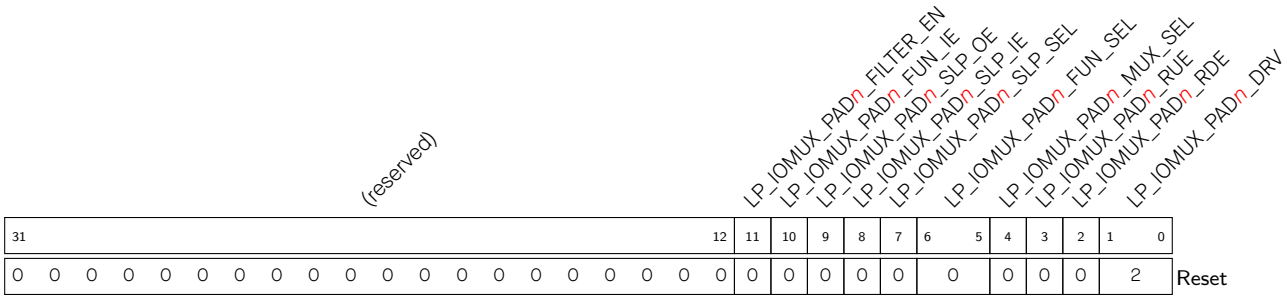
Register 10.81. LP_IOMUX_VER_DATE_REG (0x0004)



LP_IOMUX_VER_DATE Version control register.

(R/W)

Register 10.82. LP_IOMUX_PADn_REG (n = 0 - 15) (0x0008+4*n)



LP_IOMUX_PADn_DRV Configures the drive strength of GPIOn.

0: ~5 mA

1: ~10 mA

2: ~20 mA

3: ~40 mA

(R/W)

LP_IOMUX_PADn_RDE Configures whether or not enable pull-down resistor of GPIOn.

0: Disable

1: Enable

(R/W)

LP_IOMUX_PADn_RUE Configures whether or not to enable pull-up resistor of GPIOn.

0: Disable

1: Enable

(R/W)

LP_IOMUX_PADn_MUX_SEL Configures to decide GPIOn is used by HP IO MUX or LP IO MUX.

0: GPIOn is used by HP IO MUX

1: GPIOn is used by LP IO MUX

(R/W)

LP_IOMUX_PADn_FUN_SEL Configures to select LP IO MUX function for this signal.

0: Select Function 0

1: Select Function 1

.....

(R/W)

LP_IOMUX_PADn_SLP_SEL Configures whether or not to enter sleep mode for GPIOn.

0: Not enter

1: Enter

(R/W)

LP_IOMUX_PADn_SLP_IE Configures whether or not to enable the input of GPIOn during sleep mode.

0: Disable

1: Enable

(R/W)

Continued on the next page...

Register 10.83. LP_IOMUX_PAD*n*_REG (*n* = 0 - 15) (0x0008+4**n*)

Continued from the previous page...

LP_IOMUX_PAD*n*_SLP_OE Configures whether or not to enable the output of GPIO*n* in sleep mode.

0: Disable

1: Enable

(R/W)

LP_IOMUX_PAD*n*_FUN_IE Configures whether or not to enable input of GPIO*n*.

0: Disable

1: Enable

(R/W)

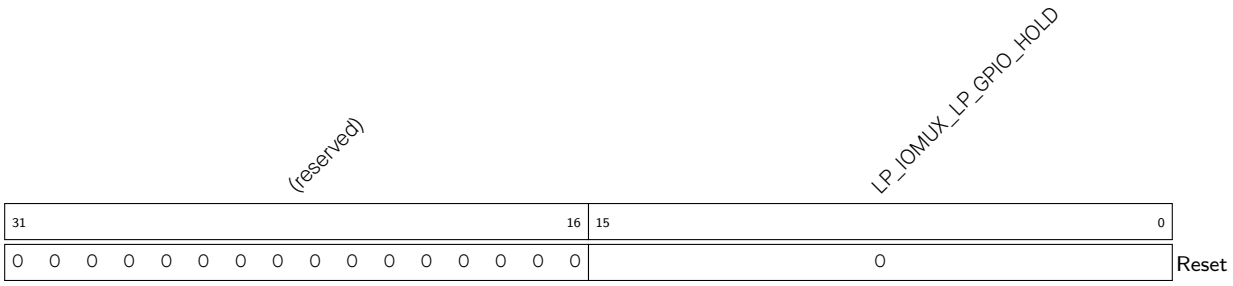
LP_IOMUX_PAD*n*_FILTER_EN Configures whether or not to enable filter for pin input signals.

0: Disable

1: Enable

(R/W)

Register 10.84. LP_IOMUX_LP_PAD_HOLD_REG (0x004C)



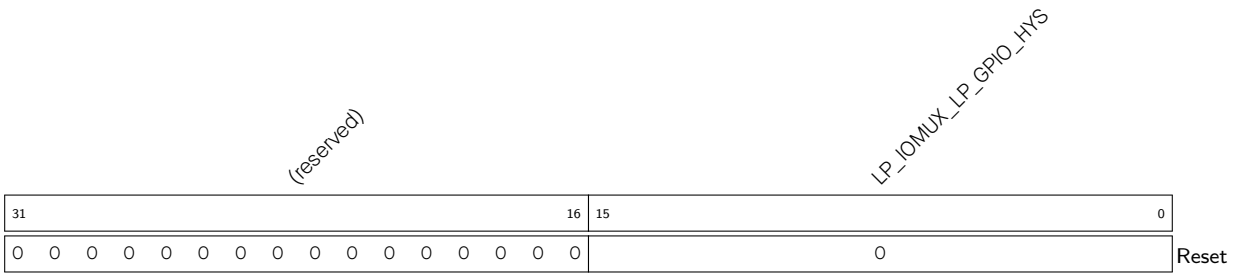
LP_IOMUX_LP_GPIO_HOLD Configures whether or not to enable hold function of GPIO*n*.

0: Disable

1: Enable

(R/W)

Register 10.85. LP_IOMUX_LP_PAD_HYS_REG (0x0050)



LP_IOMUX_LP_GPIO_HYS Configures whether or not to enable hysteresis function of GPIO*n*.

0: Disable

1: Enable

Bit0 bit15 are corresponding to GPIO0 GPIO15.

(R/W)

Chapter 11

Reset and Clock

11.1 Reset

11.1.1 Overview

ESP32-P4 provides four types of reset that occur at different levels, namely CPU Reset, Core Reset, System Reset, and Chip Reset. All reset types mentioned above (except Chip Reset) preserve the data stored in internal memory. Figure 11.1-1 shows the scopes of affected subsystems by each type of reset.

11.1.2 Architectural Overview

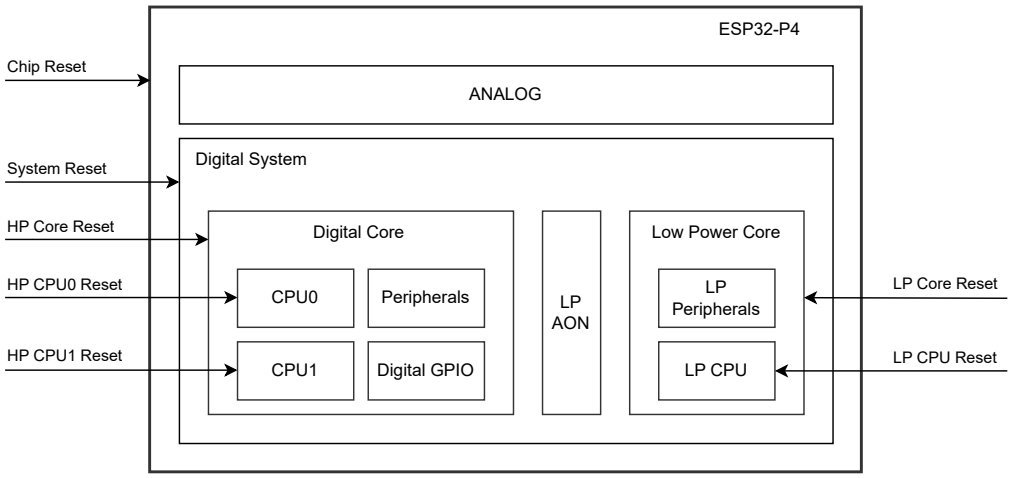


Figure 11.1-1. Reset Types

ESP32-P4's Digital System consists of High Performance System (HP system) that includes Digital Core and HP memory, and Low Power System (LP system) that includes the Low Power Always On (LP AON) system, LP Core, and LP memory. See Figure 11.1-1 for details (note that HP memory and LP memory would not be reset, so they are not shown in the figure).

11.1.3 Features

- Four reset types:
  - CPU Reset: resets CPU core. HP CPU0, HP CPU1, and LP CPU can be reset independently:
    * HP CPU0 will be automatically released from reset after chip power-up, and start execution from CPU Reset Vector (0x30100000).

    * HP CPU1 is at reset by default after chip power-up, and needs to be manually released from reset by writing 0 to HP_SYS_CLKRST_RST_EN_CORE1_GLOBAL. Once released from reset, HP CPU1 will start execution from CPU Reset Vector (0x30100000).
    * LP CPU is at reset after chip power-up, and needs to be manually released from reset by configuring the power management unit (PMU). Once released from reset, LP CPU will start execution from LP_SYSTEM_LP_CPU_BOOT_ADDR (0x50100000 by default). For details, see Chapter 15 *Low-Power Management*.
  - Core Reset: resets the whole digital system except for LP AON. HP core and LP core can be reset independently: HP Core Reset resets HP CPU0, HP CPI1, HP peripherals, HP GPIO, etc., and LP Core Reset resets LP CPU and LP peripherals.
  - System Reset: resets the whole digital system, including the LP system.
  - Chip Reset: resets the whole chip.
- Software reset and hardware reset:
  - Software Reset: triggered via software by configuring the corresponding registers of CPU, see Chapter 15 *Low-Power Management*.
  - Hardware Reset: triggered directly by the hardware.

11.1.4 Functional Description

CPU will be reset immediately when any type of reset above occurs. After the reset is released, users can retrieve reset source codes by reading the following fields:

- HP CPU0: LP_CLKRST_HPCORE0_RESET_CAUSE
- HP CPU1: LP_CLKRST_HPCORE1_RESET_CAUSE
- LP CPU: LP_CLKRST_LPCORE_RESET_CAUSE

Table 11.1-1 and Table 11.1-2 list possible reset sources and the types of reset they trigger.

Table 11.1-1. HP CPU Reset Source

Code	Source	Reset Type	Note
0x01	Chip reset[1]	Chip Reset	—
0x03	Software system reset	HP Core Reset	Triggered by configuring LP_SYSTEM_SYS_SW_RST
0x05	PMU core reset	HP Core Reset	See Chapter 15 *Low-Power Management*
0x07	MWDT core reset[2]	HP Core Reset	See Chapter 18 *Watchdog Timers (WDT)*
0x09	RWDT core reset	HP Core Reset	See Chapter 18 *Watchdog Timers (WDT)*
0x0B	MWDT CPU reset	HP CPU0/1 Reset	See Chapter 18 *Watchdog Timers (WDT)*
0x0C	Software CPU reset	HP CPU0/1 Reset	Triggered by configuring LP_CLKRST_HPCORE0_SW_RESET or LP_CLKRST_HPCORE1_SW_RESET
0x0D	RWDT CPU reset	HP CPU0/1 Reset	See Chapter 18 *Watchdog Timers (WDT)*

Cont'd on next page

Table 11.1-1 – cont'd from previous page

Code	Source	Reset Type	Note
0x0F	Brown-out system reset	Chip Reset or System Reset	Triggered by the brown-out detector. Once brown-out status is detected, the detector will trigger System Reset or Chip Reset, depending on the register configuration. See Chapter 15 *Low-Power Management*
0x10	RWDT system reset	System Reset	See Chapter 18 *Watchdog Timers (WDT)*
0x12	Super Watchdog reset	System Reset	See Chapter 18 *Watchdog Timers (WDT)*
0x13	Power glitch reset	System Reset	See Chapter 25 *Brown-out Detector*
0x1B	PSDET reset	System Reset	See Chapter 26 *Power Supply Detector (PSDET)*
0x14	eFuse reset	HP Core Reset	Triggered by eFuse CRC error
0x16	USB (JTAG) reset	HP Core Reset	Triggered when external USB host sends a specific command to the JTAG interface of USB Serial/JTAG Controller. See Chapter 54 *USB Serial/JTAG Controller (USB_SERIAL_JTAG)*
0x17	USB (UART) reset	HP Core Reset	Triggered when external USB host sends a specific command to the serial interface of USB Serial/JTAG Controller. See Chapter 54 *USB Serial/JTAG Controller (USB_SERIAL_JTAG)*
0x18	JTAG CPU reset	HP CPU0/1 Reset	Triggered when a "GDB Resetting CPU" instruction is received
0x1A	Lockup reset	HP CPU0/1 Reset	Triggered when the CPU enters lockup. HP CPU0 and HP CPU1 can be reset in lockup state by configuring LP_CLKRST_HPCOREx_LOCKUP_RESET_EN

- Chip Reset can be triggered by the following sources:
  - Chip power-up
  - Brown-out detector
- The HP system contains two watchdog timers called Main System Watchdog Timers (MWDT), and the MWDT that can trigger reset can be configured via HP_SYS_CLKRST_HPCORE0_WDT_RESET_SOURCE_SEL and HP_SYS_CLKRST_HPCORE0_WDT_RESET_SOURCE_SEL for HP CPU0 and HP CPU1 respectively.

Table 11.1-2. LP CPU Reset Source

Code	Source	Reset Type	Note
0x01	Chip reset	Chip Reset	—
0x09	PMU LP Peripheral reset	LP Core Reset	See Chapter 15 *Low-Power Management*
0x0A	PMU LP CPU reset[1]	LP CPU Reset	See Chapter 15 *Low-Power Management*
0x0F	Brown-out system reset	Chip Reset or System Reset	Triggered by the brown-out detector
0x10	RWDT system reset	System Reset	See Chapter 18 *Watchdog Timers (WDT)*
0x12	Super Watchdog reset	System Reset	See Chapter 18 *Watchdog Timers (WDT)*

Cont'd on next page

Table 11.1-2 – cont'd from previous page

Code	Source	Reset Type	Note
0x13	Power glitch reset	System Reset	See Chapter 25 *Brown-out Detector*
0x14	Software LP CPU reset	LP CPU Reset	Triggered by configuring LPPERI_RST_EN_LP_CORE

[1] LP CPU is reset by PMU after chip power-up. To retrieve the correct reset source code (0x01) when the LP CPU runs for the first time, the source code for PMU LP CPU reset (0x0A) will be masked. After the CPU's first run, clear LP_CLKRST_LPCORE_RESET_CAUSE_PMU_LP_CPU_MASK to unmask the 0x0A source code.

11.1.5 Peripheral Reset

Peripherals can be reset individually by configuring corresponding registers, or globally by Core Reset, System Reset, or Chip Reset.

ESP32-P4 has three groups of peripheral reset registers, prefixed with HP_SYS_CLKRST, LP_CLKRST, and LPPERI. See Section 11.4 *Register Summary* for detailed information.

11.2 Clock

11.2.1 Overview

ESP32-P4 clocks are mainly sourced from oscillator (OSC, including Resistor-Capacitor circuit), crystal (XTAL), and PLL circuit, and then processed by the dividers or selectors, which allows most functional modules to select their working clock according to their power consumption and performance requirements. Figure 11.2-1 shows the HP and LP system clock structure, including the main clock path and typical peripheral clock generator circuit. ESP32-P4 has a large number of peripherals, and limited by space, not all peripheral clocks are shown in the figure.

11.2.2 Architectural Overview

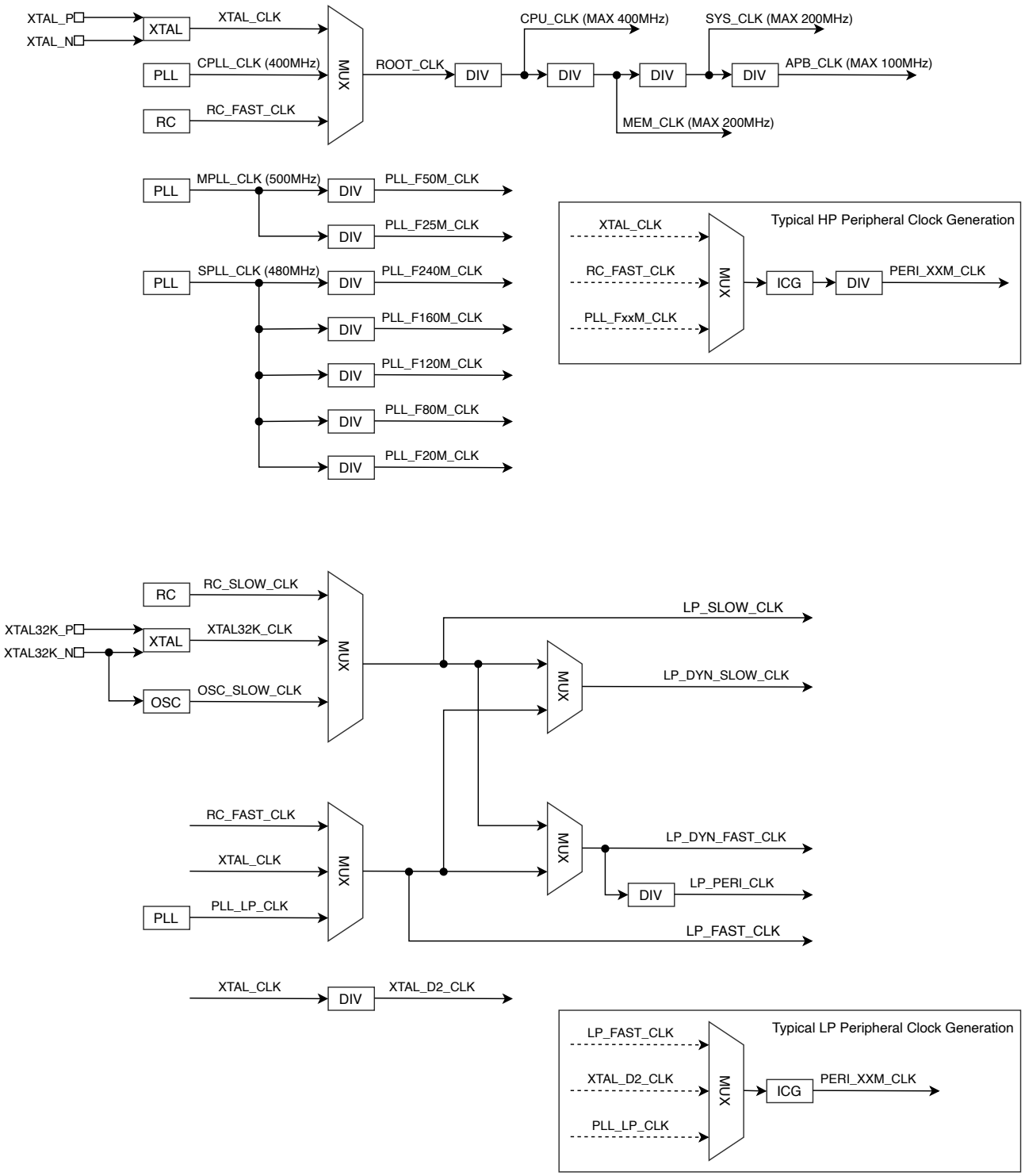


Figure 11.2-1. HP and LP System Clock

11.2.3 Features

ESP32-P4 clocks can be classified into two types depending on their frequencies:

- High speed clocks for devices working at a higher frequency, such as HP CPU0/1 and digital peripherals

  - CPLL_CLK: internal 360 MHz PLL clock. Its reference clock is XTAL_CLK
  - MPLL_CLK: internal 500 MHz PLL clock. Its reference clock is XTAL_CLK
  - SPLL_CLK: internal 480 MHz PLL clock. Its reference clock is XTAL_CLK
- Slow speed clocks for LP system and some peripherals working in low-power mode
  - XTAL32K_CLK: external 32 kHz crystal clock
  - RC_SLOW_CLK: internal 150 kHz slow RC oscillator
  - OSC_SLOW_CLK: external slow clock input through XTAL_32K_N, with a frequency of 32 kHz by default. After configuring this GPIO, also configure the Hold function (see Chapter 10 *GPIO Matrix and IO MUX* > 10.9 *Pin Hold Feature*)
  - XTAL_CLK: 40 MHz external crystal clock
  - RC_FAST_CLK: internal fast RC oscillator with adjustable frequency (20 MHz by default)
  - PLL_LP_CLK: internal PLL clock with a frequency of 8 MHz by default. Its reference clock can be XTAL32K_CLK

11.2.4 Functional Description

11.2.4.1 HP System Clock

As Figure 11.2-1 shows, XTAL_CLK, CPLL_CLK, and RC_FAST_CLK are the multiplexer inputs to generate ROOT_CLK. For selecting which input, see Table 11.2-1.

Table 11.2-1. ROOT_CLK Clock Source

LP_CLKRST_HP_ROOT_CLK_SRC_SEL	Clock Source
0	XTAL_CLK
1	CPLL_CLK
2	RC_FAST_CLK

ROOT_CLK is then divided by a series of dividers, and sequentially derives:

- CPU_CLK, which drives HP CPUs and their logic
- MEM_CLK, which drives the internal memories (L2 Cache, L2MEM, ROM) and their logic
- SYS_CLK, which is the HP high-speed bus clock that drives the AXI and AHB bus logic
- APB_CLK, which is the HP low-speed bus clock that drives the APB bus logic

The above four derived clocks are synchronous SoC clocks, asynchronous to peripheral clocks described in Section 11.2.4.3 *Peripheral Clocks*. Peripheral clocks are asynchronous with each other unless otherwise stated.

The maximum frequencies allowed for these SoC clocks are as follows. Software should ensure that the frequencies of these clocks do not exceed the maximum frequency.

- CPU_CLK: 360 MHz
- MEM_CLK: 200 MHz

- SYS_CLK: 200 MHz
- APB_CLK: 100 MHz

Each module has an individual clock gate for the SoC clock it uses. For example, users can enable the clock for HP CPU0 and disable the clock for HP CPU1 by setting the HP_SYS_CLKRST_CORE0_CPU_CLK_EN bit and clearing the HP_SYS_CLKRST_CORE1_CPU_CLK_EN bit. "CPU_CLK" in bit names indicates that the corresponding clock belongs to the CPU_CLK group shown in Figure 11.2-1.

The internal PLL clocks of ESP32-P4, namely MPLL_CLK and SPLL_CLK, generate reference clocks for peripherals through a set of parallel clock dividers.

Besides these two PLL clocks, there are two additional PLL clocks dedicated to specific modules:

- SDIO_PLL, which generates three clocks:
  - SDIO_PLL0_CLK, used for SDIO's high-speed SDIO_SLF_CLK (internal signal clock) and the source of SDIO's low-speed clock
  - SDIO_PLL1_CLK, users for SDIO's high-speed SDIO_DRV_CLK (output signal driving clock)
  - SDIO_PLL2_CLK, used for SDIO's high-speed SDIO_SAM_CLK (input signal sampling clock)
- APLL_CLK, which is dedicated to I2S

A typical peripheral clock generation circuit consists of a clock selector, a clock gating unit, and a divider. In high-performance mode, the peripheral can select a high-speed clock source without division for higher processing speed. In low-power mode, the peripheral can select a low-speed clock source or divide it by a large divisor to lower power consumption. When the peripheral is disabled, its clock can be directly disabled.

Note:
Updates to HP_SYS_CLKRST_ROOT_CLK_CTRL0/1/2/3_REG will take effect only after HP_SYS_CLKRST_SOC_CLK_DIV_UPDATE is set.

11.2.4.2 LP System Clock

The LP system can operate when most other clocks are disabled. LP system clocks include LP_SLOW_CLK, LP_FAST_CLK, LP_DYN_SLOW_CLK, LP_DYN_FAST_CLK, and XTAL_D2_CLK.

The clock sources for LP_SLOW_CLK and LP_FAST_CLK are low-frequency clocks:

- LP_SLOW_CLK has four clock sources, see Table 11.2-2:

Table 11.2-2. LP_SLOW_CLK Clock Source Selection

LP_CLKRST_SLOW_CLK_SEL	Clock Source
0	RC_SLOW_CLK
1	XTAL32K_CLK
2	Invalid
3	OSC_SLOW_CLK

- LP_FAST_CLK has three clock sources, see Table 11.2-3:

Table 11.2-3. LP_FAST_CLK Clock Source Selection

LP_CLKRST_FAST_CLK_SEL	Clock Source
0	RC_FAST_CLK
1	XTAL_CLK
2	PLL_LP_CLK

LP_DYN_SLOW_CLK and LP_DYN_FAST_CLK can be derived from LP_SLOW_CLK or LP_FAST_CLK depending on the chip's power mode (see Chapter 15 *Low-Power Management*), and should be able to operate in all power modes to drive LP AON modules. These two clocks are always synchronous with each other.

- In Active mode, select LP_FAST_CLK as the clock source. In this case, LP_DYN_SLOW_CLK has the same frequency with LP_SLOW_CLK, and the same phase with LP_FAST_CLK;
- In Deep-sleep mode, select the clock source according to PMU configurations. If the LP CPU is off, LP_SLOW_CLK is the clock source, and has the same frequency and phase as LP_DYN_FAST_CLK in this case; if the LP CPU is on, LP_FAST_CLK is the clock source.

LP_PERI_CLK is derived from LP_DYN_FAST_CLK and drives all the buses for LP peripherals.

XTAL_D2_CLK is derived from XTAL_CLK divided by 2.

11.2.4.3 Peripheral Clocks

Table 11.2-4, Table 11.2-5, Table 11.2-6, and Table 11.2-7 list the derived HP/LP clock sources and HP clocks/LP clocks for each peripheral.

Table 11.2-4. Derived HP Clock sources

Derived Clock	Source Clock							Derived Clock						
	RC_FAST_CLK	XTAL_CLK	CPLL_CLK	MPLL_CLK	SPLL_CLK	APLL_CLK	SDIO_PLL0_CLK	ROOT_CLK	CPU_CLK	MEM_CLK	SYS_CLK	PLL_F80M_CLK	PLL_F160M_CLK	PLL_F240M_CLK
ROOT_CLK	Y	Y	Y											
CPU_CLK								Y						
MEM_CLK									Y					
SYS_CLK										Y				
APB_CLK											Y			
PLL_F50M_CLK				Y										
PLL_F25M_CLK				Y										
PLL_F240M_CLK					Y									
PLL_F160M_CLK					Y									
PLL_F120M_CLK					Y									
PLL_F80M_CLK					Y									
PLL_F20M_CLK					Y									
PLL_F12M_CLK					Y									
PLL_F48M_CLK					Y									
FLASH_CLK		Y	Y		Y									
PSRAM_CLK		Y	Y	Y	Y									
GPSPI_CLK	Y	Y			Y	Y	Y							
ADC_CLK	Y	Y										Y		
UART_CLK	Y	Y										Y		
CRYPTO_CLK	Y	Y											Y	Y

Table 11.2-5. HP Peripheral Clocks

Peripheral	Clock	Clock Source														Derived Clock						Clock from IO
		RC_FAST_CLK	XTAL_CLK	APLL_CLK	CPLL_CLK	SPLL_CLK	MPLL_CLK	SDIO_PLL0_CLK	PLL_F12M_CLK	PLL_F20M_CLK	PLL_F25M_CLK	PLL_F48M_CLK	PLL_F80M_CLK	PLL_F120M_CLK	PLL_F160M_CLK	PLL_F240M_CLK	FLASH_CLK	PSRAM_CLK	GPSPI_CLK	ADC_CLK	UART_CLK	CRYPTO_CLK	
SPI0/1 (flash)	FLASH_PLL_CLK																Y						
	FLASH_CORE_CLK																Y						
SPI0/1 (PSRAM)	PSRAM_PLL_CLK																	Y					
	PSRAM_CORE_CLK																	Y					
I2S	I2S_RX_CLK		Y	Y											Y								PAD_I2S_MCLK
	I2S_TX_CLK		Y	Y											Y								PAD_I2S_MCLK
I2C	I2C_CLK	Y	Y																				
MCPWM	MCPWM_CLK	Y	Y												Y								
TIMG	TIMERGRP_T0_CLK	Y	Y										Y										
	TIMERGRP_T1_CLK	Y	Y										Y										
MWDT	TIMERGRP_WDT_CLK	Y	Y										Y										
TWAI	TWAI_CLK	Y	Y																				
GP-SPI2/GP-SPI3	GPSPI_HS_CLK																		Y				
	GPSPI_MST_CLK																		Y				
EMAC	EMAC_RMII_CLK																						PAD_EMAC_TXRX_CLK PAD_EMAC_RX_CLK PAD_EMAC_TX_CLK
	EMAC_RX_CLK																						PAD_EMAC_TXRX_CLK PAD_EMAC_RX_CLK
	EMAC_TX_CLK																						PAD_EMAC_TXRX_CLK PAD_EMAC_TX_CLK
	EMAC_PTP_REF_CLK		Y										Y										
CAM OUT	CAM_CLK		Y	Y											Y								
IO MUX	IOMUX_CLK		Y										Y										
LCD	LCD_CLK		Y	Y											Y								

Cont'd on next page

GoBack

Table 11.2-5 – cont'd from previous page

Peripheral	Clock	Clock Source														Derived Clock						Clock from IO
		RC_FAST_CLK	XTAL_CLK	APLL_CLK	CPLL_CLK	SPLL_CLK	MPLL_CLK	SDIO_PLL0_CLK	PLL_F12M_CLK	PLL_F20M_CLK	PLL_F25M_CLK	PLL_F48M_CLK	PLL_F80M_CLK	PLL_F120M_CLK	PLL_F160M_CLK	PLL_F240M_CLK	FLASH_CLK	PSRAM_CLK	GPSPI_CLK	ADC_CLK	UART_CLK	CRYPTO_CLK	
LEDC	LEDC_CLK	Y	Y										Y										
ADC	HPADC_CLK																			Y			
	HPADC1_SARCLK																			Y			
	HPADC2_SARCLK																			Y			
AES SHA RSA ECC RSA_DS HMAC Key Manager ECDSA_DS	AES_CLK SHA_CLK RSA_CLK ECC_CLK DSA_CLK HMAC_CLK KM_CLK ECDSA_DS_CLK SEC_CLK																					Y	
RMT	RMT_CLK	Y	Y										Y										
System Timer	SYSTIMER_CLK	Y	Y																				
UART	UART_FCLK																				Y		
	UART_SCLK																				Y		
	UART_SLP_CLK																						PAD_UART_SLP_CLK
PARLIO	PARLIO_RX_CLK	Y	Y												Y								PAD_PARLIO_RX_CLK
	PARLIO_TX_CLK	Y	Y												Y								PAD_PARLIO_TX_CLK
I3S	I3C_MST_CLK		Y											Y	Y								
ISP	ISP_CLK		Y												Y	Y							
MIPI DSI	DSI_DPHY_CFG_CLK	Y								Y	Y												
	DSI_DPHY_PLL_REFCLK		Y	Y	Y	Y	Y																
	DSI_DPICLK		Y	Y											Y	Y							
MIPI CSI	CSI_DPHY_CFG_CLK	Y								Y	Y												
H264	H264_CLK		Y													Y							
USB_SERIAL_JTAG	USB_SERIAL_JTAG_CLK											Y											
USB 2.0 Full-Speed OTG	USB_OTG11_CLK											Y											
USB 2.0 High-Speed OTG	USB_HS_PHY_REFCLK								Y	Y													PAD_USB_HSPHY_REFCLK
SD/MMC Host Controller (SDHOST)	SDIO_DRV_CLK (low-speed) SDIO_SLF_CLK (low-speed) SDIO_SAM_CLK (low-speed)							Y							Y								

Table 11.2-6. Derived LP Clock Sources

Derived clock	Source clock						Derived clock					Clock from IO
	RC_SLOW_CLK 150 kHz	XTAL32K_CLK 32 kHz	OSC_SLOW_CLK 32 kHz	RC_FAST_CLK 20 MHz	XTAL_CLK 40 MHz	PLL_LP_CLK 8 MHz	LP_SLOW_CLK	LP_FAST_CLK	LP_DYN_SLOW_CLK	LP_DYN_FAST_CLK	XTAL_D2_CLK	
LP_FAST_CLK				Y	Y	Y						
LP_SLOW_CLK	Y	Y	Y									
LP_DYN_FAST_CLK							Y	Y				
LP_DYN_SLOW_CLK							Y	Y				
LP_PERI_CLK										Y		
XTAL_D2_CLK					Y							

Table 11.2-7. LP Peripheral Clocks

Derived clock	Source clock						Derived clock						Clock from IO
	RC_SLOW_CLK 150 kHz	XTAL32K_CLK 32 kHz	OSC_SLOW_CLK 32 kHz	RC_FAST_CLK 20 MHz	XTAL_CLK 40 MHz	PLL_LP_CLK 8 MHz	LP_SLOW_CLK	LP_FAST_CLK	LP_DYN_SLOW_CLK	LP_DYN_FAST_CLK	LP_PERI_CLK	XTAL_D2_CLK	
eFuse Controller (EFUSE)												Y	
RTC Watchdog Timer (RWDT)	Y								Y	Y			
RTC Timer									Y	Y			
Brown-out Detector										Y			
Power Management Unit (PMU)	Y							Y	Y	Y			
LP UART						Y				Y		Y	PAD_LP_UART_SLP_CK
LP SPI													PAD_LP_SPI_CK
LP I2C						Y				Y		Y	
Analog I2C Controller								Y					
LP I2S Controller						Y				Y		Y	
LP ADC Controller											Y		

Note:
To protect the encryption and decryption peripherals from DPA (Differential Power Analysis) attacks, a random divider strategy is implemented for its function clock. Four security levels are available, depending on the range of the random divider. Users can select the security level by configuring HP_SYS_CLKRST_SEC_DPA_CFG_SEL.
If HP_SYS_CLKRST_SEC_DPA_CFG_SEL is set to 1, the security level is determined by the configuration of HP_SYS_CLKRST_SEC_DPA_LEVEL. Otherwise, the security level is determined by the value of EFUSE_SEC_DPA_LEVEL.

11.3 Programming Procedures

11.3.1 HP System Clock Configuration

- Configure the clock divisor for CPU_CLK: HP_SYS_CLKRST_CPU_CLK_DIV_NUM for the integral part, HP_SYS_CLKRST_CPU_CLK_DIV_NUMERATOR for the numerator of the fractional part, and HP_SYS_CLKRST_CPU_CLK_DIV_DENOMINATOR for the denominator of the fractional part.
  Write 1 to HP_SYS_CLKRST_SOC_CLK_DIV_UPDATE to apply the frequency divisor configurations for CPU_CLK.
- Configure the clock divisor for MEM_CLK: HP_SYS_CLKRST_MEM_CLK_DIV_NUM for the integral part, HP_SYS_CLKRST_MEM_CLK_DIV_NUMERATOR for the numerator of the fractional part, and HP_SYS_CLKRST_MEM_CLK_DIV_DENOMINATOR for the denominator of the fractional part.
  Write 1 to HP_SYS_CLKRST_SOC_CLK_DIV_UPDATE to apply the frequency divisor configurations for MEM_CLK.
- Configure the clock divisor for SYS_CLK: HP_SYS_CLKRST_SYS_CLK_DIV_NUM for the integral part, HP_SYS_CLKRST_SYS_CLK_DIV_NUMERATOR for the numerator of the fractional part, and HP_SYS_CLKRST_SYS_CLK_DIV_DENOMINATOR for the denominator of the fractional part.
  Write 1 to HP_SYS_CLKRST_SOC_CLK_DIV_UPDATE to apply the frequency divisor configurations for SYS_CLK.
- Configure the clock divisor for APB_CLK: HP_SYS_CLKRST_APB_CLK_DIV_NUM for the integral part, HP_SYS_CLKRST_APB_CLK_DIV_NUMERATOR for the numerator of the fractional part, and HP_SYS_CLKRST_APB_CLK_DIV_DENOMINATOR for the denominator of the fractional part.
  Write 1 to HP_SYS_CLKRST_SOC_CLK_DIV_UPDATE to apply the frequency divisor configurations for APB_CLK.
- Configure the clock source for ROOT_CLK via LP_CLKRST_HP_ROOT_CLK_SRC_SEL.

Notice:

1. It is recommended to first configure the frequency divisors for the SoC clocks and apply the configurations, and then configure the clock source for ROOT_CLK.
2. Before changing the clock source for ROOT_CLK, software needs to ensure that the configured clock divisors do not result in clock frequencies exceeding the allowed maximum frequencies.
3. Writing 1 to HP_SYS_CLKRST_SOC_CLK_DIV_UPDATE will apply the frequency divisor configurations for CPU_CLK, MEM_CLK, SYS_CLK, and APB_CLK at one go, but it is not recommended to do so. It would be better to apply the configuration for each clock individually.
4. When switching from a high divisor to a low divisor, the recommended configuration order is APB_CLK

→ SYS_CLK → MEM_CLK → CPU_CLK. When switching from a low divisor to a high divisor, the recommended configuration order is CPU_CLK → MEM_CLK → SYS_CLK → APB_CLK. This helps to prevent high-frequency clock glitches.

11.3.2 LP System Clock Configuration

- Configure the clock source of LP_SLOW_CLK via LP_CLKRST_SLOW_CLK_SEL
- Configure the clock source of LP_FAST_CLK via LP_CLKRST_FAST_CLK_SEL
- Configure the clock divisor of LP_PERI_CLK via LP_CLKRST_LP_PERI_DIV_NUM

11.3.3 Peripheral Clock Reset and Configuration

Notice:

ESP32-P4 features low power consumption. This is why some peripheral clocks are gated (disabled) by default. Before using any of these peripherals, it is mandatory to enable the clock for the given peripheral by setting the corresponding CLK_EN bit to 1, and release the peripheral from the reset state to make it operational by setting the RST_EN bit to 0.

The clocks of most peripherals can be classified into two types:

- Bus clock: used to configure peripheral registers.
- Function clock: such as UART's reference clock, used by peripherals to operate.

The function clock of most peripherals can be selected from multiple clock sources. In the description of the gating registers, it will be stated whether a register is a bus clock (SYS_CLK, APB_CLK) gating register or a function clock gating register.

Bus clock enable/disable registers, function clock enable/disable registers, clock source selection registers, and clock frequency division registers are grouped into the HP_SYS_CLKRST, LP_CLKRST, and LPPERI modules. Except for EMA, SDMMAC, USB Serial/JTAG, USB 2.0 Full-Speed OTG, and USB 2.0 High-Speed OTG, whose reset configuration registers are in the LP_CLKRST module, all the other HP clock and reset registers are in the HP_SYS_CLKRST module. For more information, see Section 11.4 *Register Summary*.

When changing the clock configuration, it is recommended to turn off the corresponding clock gate first, update the clock divider configurations, clock sources, and other configurations, and then turn it on again.

When a peripheral is not working, users can turn off its function clock by configuring related registers. Turning off the peripheral's function clock does not affect the rest of the system.

Take I2C clock configuration as an example.

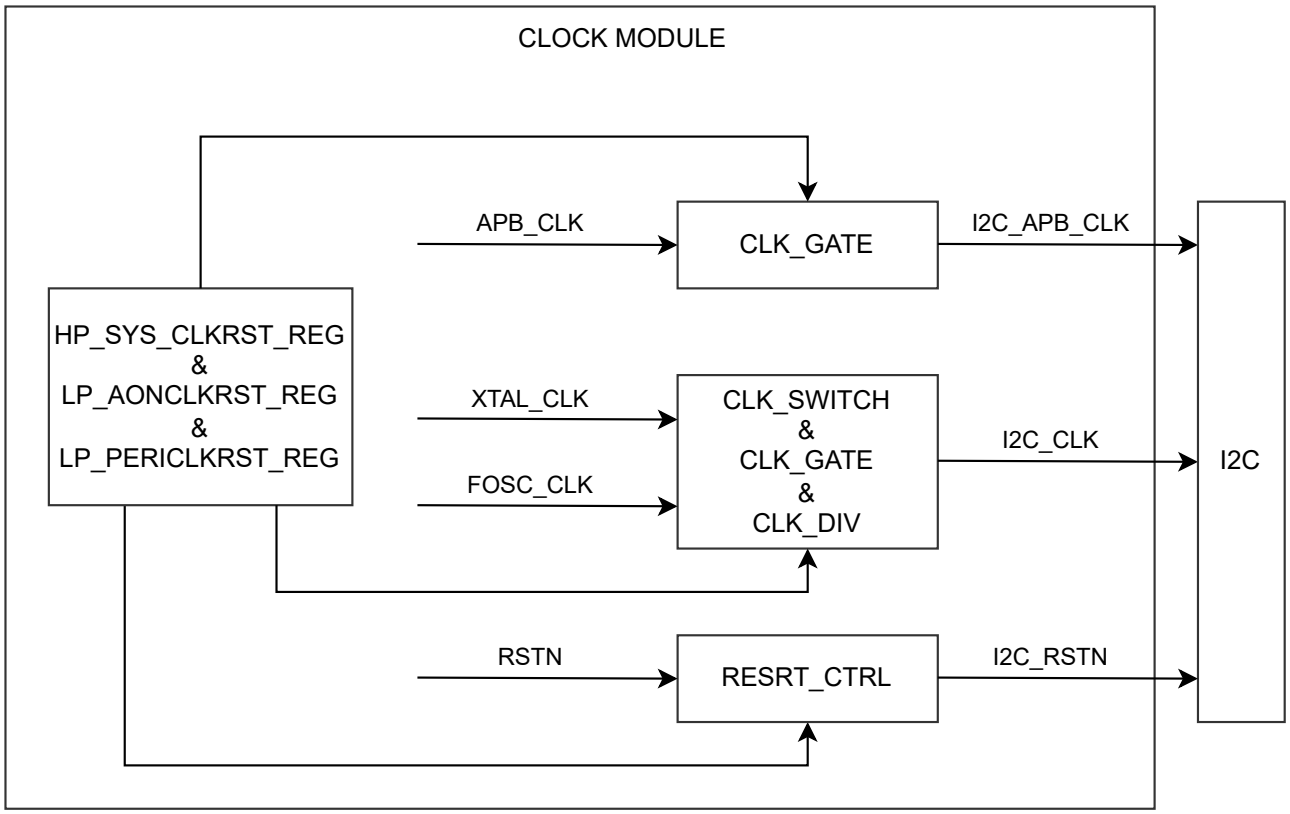


Figure 11.3-1. Clock Configuration Example

Figure 11.3-1 shows the clock structure of I2C. The clock structure of other peripherals is similar to this one. CLK_SWITCH is used to select a clock output and CLK_GATE to turn on/off the clock.

In scenarios that require low power consumption, when the peripheral is not in use, in addition to turning off the function clock, the bus clock of the peripheral can also be turned off to further lower power consumption.

Note that if you turn off the bus clock first, the functional clock may continue working. Therefore, when turning off clocks, it is recommended to turn off the functional clock first and then the bus clock; when turning on clocks, it is recommended to turn on the bus clock first and then the functional clock.

Note:
In this chapter, all divisor configuration registers are configured with the actual divisor minus 1.

11.4 Register Summary

11.4.1 Reset and Clock (HP_SYS_CLKRST) Register Summary

The addresses in this section are relative to HP_SYS_CLKRST base address provided in Table 9.3-2 in Chapter 9 *System and Memory*.

The abbreviations given in Column **Access** are explained in Section *Access Types for Registers*.

Name	Description	Address	Access
Clock gate enable register			
HP_SYS_CLKRST_CLK_EN0_REG	Register clock gate enable register	0x0000	R/W
HP_SYS_CLKRST_CLK_FORCE_ON_CTRL0_REG	System and peripheral clock gate configuration register	0x00B4	R/W
Root clock configuration register			
HP_SYS_CLKRST_ROOT_CLK_CTRL0_REG	Root clock configuration register 0	0x0004	varies
HP_SYS_CLKRST_ROOT_CLK_CTRL1_REG	Root clock configuration register 1	0x0008	R/W
HP_SYS_CLKRST_ROOT_CLK_CTRL2_REG	Root clock configuration register 2	0x000C	R/W
HP_SYS_CLKRST_ROOT_CLK_CTRL3_REG	Root clock configuration register 3	0x0010	R/W
Bus clock configuration register			
HP_SYS_CLKRST_SOC_CLK_CTRL0_REG	Bus clock configuration register 0	0x0014	R/W
HP_SYS_CLKRST_SOC_CLK_CTRL1_REG	Bus clock configuration register 1	0x0018	R/W
HP_SYS_CLKRST_SOC_CLK_CTRL2_REG	Bus clock configuration register 2	0x001C	R/W
HP_SYS_CLKRST_SOC_CLK_CTRL3_REG	Bus clock configuration register 3	0x0020	R/W
Reference clock configuration register			
HP_SYS_CLKRST_REF_CLK_CTRL0_REG	Reference clock configuration register 0	0x0024	R/W
HP_SYS_CLKRST_REF_CLK_CTRL1_REG	Reference clock configuration register 1	0x0028	R/W
HP_SYS_CLKRST_REF_CLK_CTRL2_REG	Reference clock configuration register 2	0x002C	R/W
Peripheral clock configuration register			
HP_SYS_CLKRST_PERI_CLK_CTRL00_REG	Peripheral clock configuration register	0x0030	R/W
HP_SYS_CLKRST_PERI_CLK_CTRL01_REG	Peripheral clock configuration register	0x0034	R/W
HP_SYS_CLKRST_PERI_CLK_CTRL02_REG	Peripheral clock configuration register	0x0038	varies
HP_SYS_CLKRST_PERI_CLK_CTRL03_REG	Peripheral clock configuration register	0x003C	R/W
HP_SYS_CLKRST_PERI_CLK_CTRL10_REG	Peripheral clock configuration register	0x0040	R/W
HP_SYS_CLKRST_PERI_CLK_CTRL11_REG	Peripheral clock configuration register	0x0044	R/W
HP_SYS_CLKRST_PERI_CLK_CTRL12_REG	Peripheral clock configuration register	0x0048	R/W
HP_SYS_CLKRST_PERI_CLK_CTRL13_REG	Peripheral clock configuration register	0x004C	R/W
HP_SYS_CLKRST_PERI_CLK_CTRL14_REG	Peripheral clock configuration register	0x0050	R/W
HP_SYS_CLKRST_PERI_CLK_CTRL15_REG	Peripheral clock configuration register	0x0054	R/W
HP_SYS_CLKRST_PERI_CLK_CTRL16_REG	Peripheral clock configuration register	0x0058	R/W
HP_SYS_CLKRST_PERI_CLK_CTRL17_REG	Peripheral clock configuration register	0x005C	R/W
HP_SYS_CLKRST_PERI_CLK_CTRL18_REG	Peripheral clock configuration register	0x0060	R/W
HP_SYS_CLKRST_PERI_CLK_CTRL19_REG	Peripheral clock configuration register	0x0064	R/W
HP_SYS_CLKRST_PERI_CLK_CTRL110_REG	Peripheral clock configuration register	0x0068	R/W
HP_SYS_CLKRST_PERI_CLK_CTRL111_REG	Peripheral clock configuration register	0x006C	R/W

Name	Description	Address	Access
HP_SYS_CLKRST_PERI_CLK_CTRL112_REG	Peripheral clock configuration register	0x0070	R/W
HP_SYS_CLKRST_PERI_CLK_CTRL113_REG	Peripheral clock configuration register	0x0074	R/W
HP_SYS_CLKRST_PERI_CLK_CTRL114_REG	Peripheral clock configuration register	0x0078	R/W
HP_SYS_CLKRST_PERI_CLK_CTRL115_REG	Peripheral clock configuration register	0x007C	R/W
HP_SYS_CLKRST_PERI_CLK_CTRL116_REG	Peripheral clock configuration register	0x0080	R/W
HP_SYS_CLKRST_PERI_CLK_CTRL117_REG	Peripheral clock configuration register	0x0084	R/W
HP_SYS_CLKRST_PERI_CLK_CTRL118_REG	Peripheral clock configuration register	0x0088	R/W
HP_SYS_CLKRST_PERI_CLK_CTRL119_REG	Peripheral clock configuration register	0x008C	R/W
HP_SYS_CLKRST_PERI_CLK_CTRL120_REG	Peripheral clock configuration register	0x0090	R/W
HP_SYS_CLKRST_PERI_CLK_CTRL20_REG	Peripheral clock configuration register	0x0094	R/W
HP_SYS_CLKRST_PERI_CLK_CTRL21_REG	Peripheral clock configuration register	0x0098	R/W
HP_SYS_CLKRST_PERI_CLK_CTRL22_REG	Peripheral clock configuration register	0x009C	R/W
HP_SYS_CLKRST_PERI_CLK_CTRL23_REG	Peripheral clock configuration register	0x00A0	R/W
HP_SYS_CLKRST_PERI_CLK_CTRL24_REG	Peripheral clock configuration register	0x00A4	R/W
HP_SYS_CLKRST_PERI_CLK_CTRL25_REG	Peripheral clock configuration register	0x00A8	R/W
HP_SYS_CLKRST_PERI_CLK_CTRL26_REG	Peripheral clock configuration register	0x00AC	R/W
DPA clock configuration register			
HP_SYS_CLKRST_DPA_CTRL0_REG	DPA clock configuration register	0x00B8	R/W
Analog PLL calibration configuration register			
HP_SYS_CLKRST_ANA_PLL_CTRL0_REG	Analog PLL calibration configuration register	0x00BC	varies
Peripheral reset register			
HP_SYS_CLKRST_HP_RST_EN0_REG	Peripheral reset register 0	0x00C0	R/W
HP_SYS_CLKRST_HP_RST_EN1_REG	Peripheral reset register 1	0x00C4	R/W
HP_SYS_CLKRST_HP_RST_EN2_REG	Peripheral reset register 2	0x00C8	R/W
HP_SYS_CLKRST_HP_FORCE_NORST0_REG	Peripheral reset disable register register 0	0x00CC	R/W
HP_SYS_CLKRST_HP_FORCE_NORST1_REG	Peripheral reset disable register register 1	0x00D0	R/W
HP CPU WDT reset configuration register			
HP_SYS_CLKRST_HPWDT_CORE0_RST_CTRL0_REG	HP CPU0 WDT reset configuration register	0x00D4	R/W
HP_SYS_CLKRST_HPWDT_CORE1_RST_CTRL0_REG	HP CPU1 WDT reset configuration register	0x00D8	R/W
HP_SYS_CLKRST_HPCORE_WDT_RESET_SOURCE0_REG	HP CPU WDT reset source configuration register	0x00EC	R/W
CPU clock status register			
HP_SYS_CLKRST_CPU_SRC_FREQ0_REG	CPU source clock frequency register	0x00DC	RO
HP_SYS_CLKRST_CPU_CLK_STATUS0_REG	CPU clock status register	0x00E0	RO
WAITI Signal Control Register			
HP_SYS_CLKRST_CPU_WAITI_CTRL0_REG	WAITI signal control resister	0x00F4	R/W

11.4.2 LP Always on Clock and Reset (LP_CLKRST) Register Summary

The addresses in this section are relative to LP_CLKRST base address provided in Table 9.3-2 in Chapter 9 *System and Memory*.

The abbreviations given in Column **Access** are explained in Section *Access Types for Registers*.

Name	Description	Address	Access
Configuration registers			
LP_CLKRST_LP_CLK_CONF_REG	LP root clock configuration register	0x0000	R/W
LP_CLKRST_LP_CLK_EN_REG	LP always on peripheral clock configuration register	0x0008	R/W
LP_CLKRST_LP_RST_EN_REG	LP always on peripheral reset configuration register	0x000C	R/W
LP_CLKRST_RESET_CAUSE_REG	HP CPU and LP CPU reset cause register	0x0010	varies
LP_CLKRST_HPCPU_RESET_CTRL0_REG	HP CPU reset configuration register 0	0x0014	varies
LP_CLKRST_HPCPU_RESET_CTRL1_REG	HP CPU reset configuration register 1	0x0018	R/W
LP_CLKRST_FOSC_CNTL_REG	RC_FAST_CLK frequency configuration register	0x001C	R/W
LP_CLKRST_XTAL32K_REG	XTAL32K_CLK configuration register	0x0030	R/W
LP_CLKRST_HP_CLK_CTRL_REG	HP clock control register	0x0040	R/W
LP_CLKRST_HP_USB_CLKRST_CTRL0_REG	USB clock and reset control register 0	0x0044	R/W
LP_CLKRST_HP_USB_CLKRST_CTRL1_REG	USB clock and reset control register 1	0x0048	R/W
LP_CLKRST_HP_SDMMC_EMAC_RST_CTRL_REG	SDMMC and EMAC reset control register	0x004C	R/W
Version register			
LP_CLKRST_DATE_REG	Version control register	0x03FC	R/W

11.4.3 LP Peripheral Clock and Reset (LPPERI) Register Summary

The addresses in this section are relative to LPPERI base address provided in Table 9.3-2 in Chapter 9 *System and Memory*.

The abbreviations given in Column **Access** are explained in Section *Access Types for Registers*.

Name	Description	Address	Access
Configuration registers			
LPPERI_CLK_EN_REG	LP peripheral bus clock configuration register	0x0000	R/W
LPPERI_CORE_CLK_SEL_REG	LP peripheral function clock configuration register	0x0004	R/W
LPPERI_RESET_EN_REG	LP peripheral reset configuration register	0x0008	varies
LPPERI_CPU_REG	LP CPU access configuration register	0x000C	R/W
LPPERI_MEM_CTRL_REG	LP UART miscellaneous control register	0x0028	varies
LPPERI_ADC_CTRL_REG	LP ADC clock configuration register	0x002C	R/W
LPPERI_LP_I2S_RXCLK_DIV_NUM_REG	LP I2S RX clock configuration register	0x0030	R/W
LPPERI_LP_I2S_RXCLK_DIV_XYZ_REG	LP I2S RX clock configuration register	0x0034	R/W
Version register			

Name	Description	Address	Access
LPPERI_DATE_REG	Version control register	0x03FC	R/W

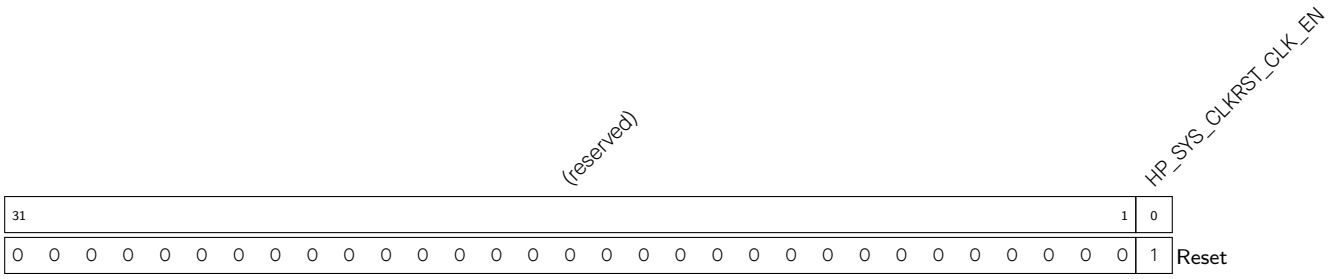
11.5 Registers

11.5.1 Reset and Clock (HP_SYS_CLKRST) Registers

The addresses in this section are relative to HP_SYS_CLKRST base address provided in Table 9.3-2 in Chapter 9 *System and Memory*.

For how to program reserved fields, please refer to Section Programming Reserved Register Field.

Register 11.1. HP_SYS_CLKRST_CLK_EN0_REG (0x0000)



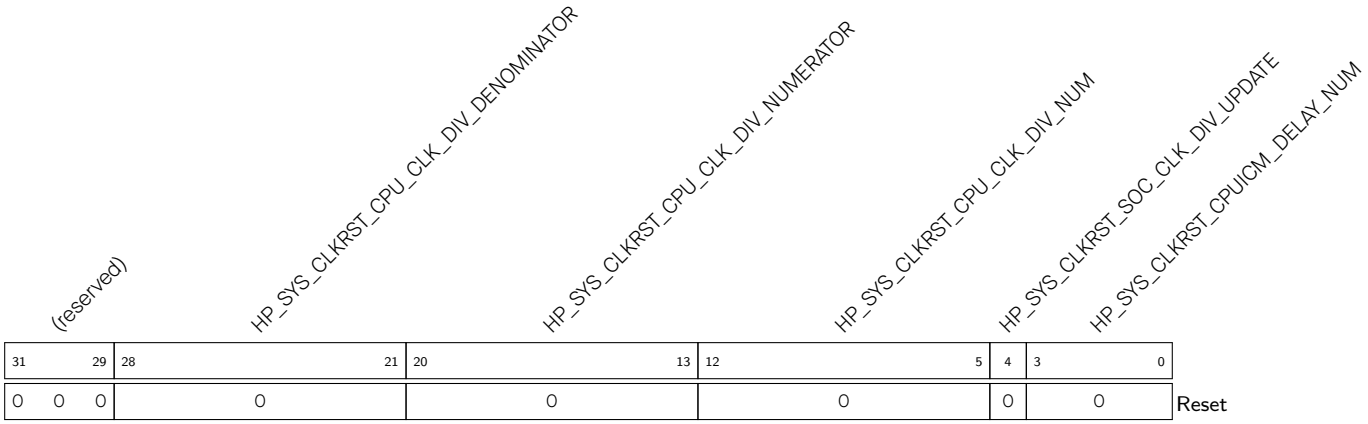
HP_SYS_CLKRST_CLK_EN Configures resister clock gating.

0: Support clock only when application writes registers

1: Force on clock gating for registers

(R/W)

Register 11.2. HP_SYS_CLKRST_ROOT_CLK_CTRL0_REG (0x0004)



HP_SYS_CLKRST_CPUICM_DELAY_NUM Configures the time required from entering WFI mode to the actual shutdown of the CPU clock.

Measurement unit: clock cycles. (R/W)

HP_SYS_CLKRST_SOC_CLK_DIV_UPDATE Configures whether to update the divisors for CPU_CLK, MEM_CLK, SYS_CLK, APB_CLK.

0: Not update

1: Update

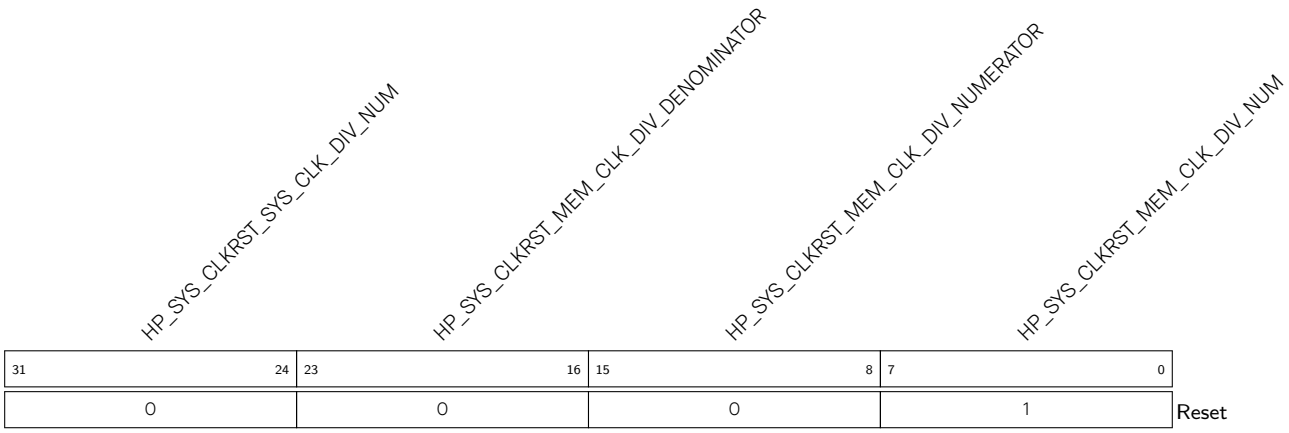
(WT)

HP_SYS_CLKRST_CPU_CLK_DIV_NUM Configures the integer part of the CPU_CLK clock divisor. (R/W)

HP_SYS_CLKRST_CPU_CLK_DIV_NUMERATOR Configures the numerator of the divisor's fractional part for CPU_CLK. (R/W)

HP_SYS_CLKRST_CPU_CLK_DIV_DENOMINATOR Configures the denominator of the divisor's fractional part for CPU_CLK. (R/W)

Register 11.3. HP_SYS_CLKRST_ROOT_CLK_CTRL1_REG (0x0008)



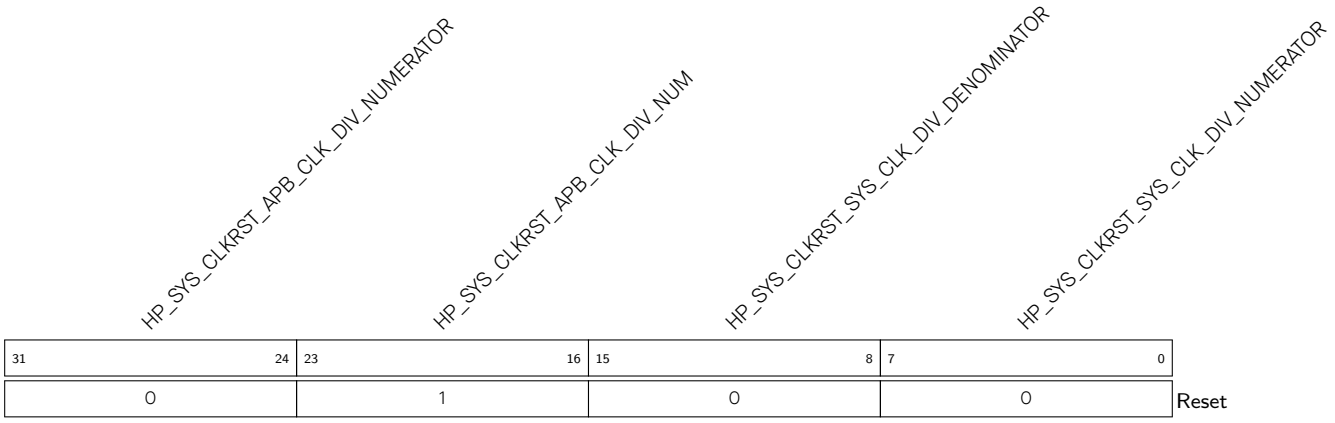
HP_SYS_CLKRST_MEM_CLK_DIV_NUM Configures the integer part of the MEM_CLK clock divisor. (R/W)

HP_SYS_CLKRST_MEM_CLK_DIV_NUMERATOR Configures the numerator of the divisor's fractional part for MEM_CLK. (R/W)

HP_SYS_CLKRST_MEM_CLK_DIV_DENOMINATOR Configures the denominator of the divisor's fractional part for MEM_CLK . (R/W)

HP_SYS_CLKRST_SYS_CLK_DIV_NUM Configures the integer part of the SYS_CLK clock divisor. (R/W)

Register 11.4. HP_SYS_CLKRST_ROOT_CLK_CTRL2_REG (0x000C)



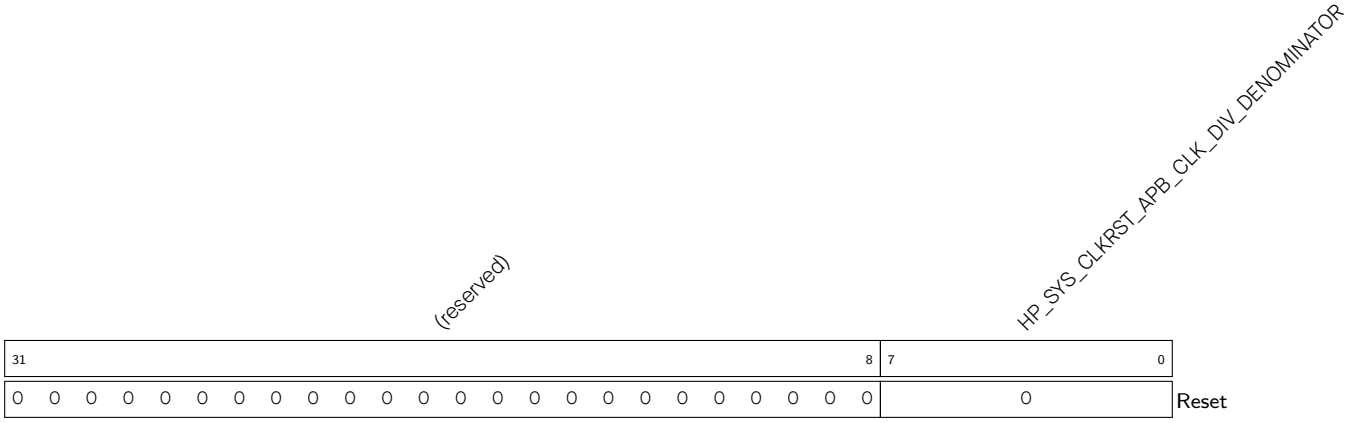
HP_SYS_CLKRST_SYS_CLK_DIV_NUMERATOR Configures the numerator of the divisor's fractional part for SYS_CLK. (R/W)

HP_SYS_CLKRST_SYS_CLK_DIV_DENOMINATOR Configures the denominator of the divisor's fractional part for SYS_CLK. (R/W)

HP_SYS_CLKRST_APB_CLK_DIV_NUM Configures the integer part of the APB_CLK fclock divisor. (R/W)

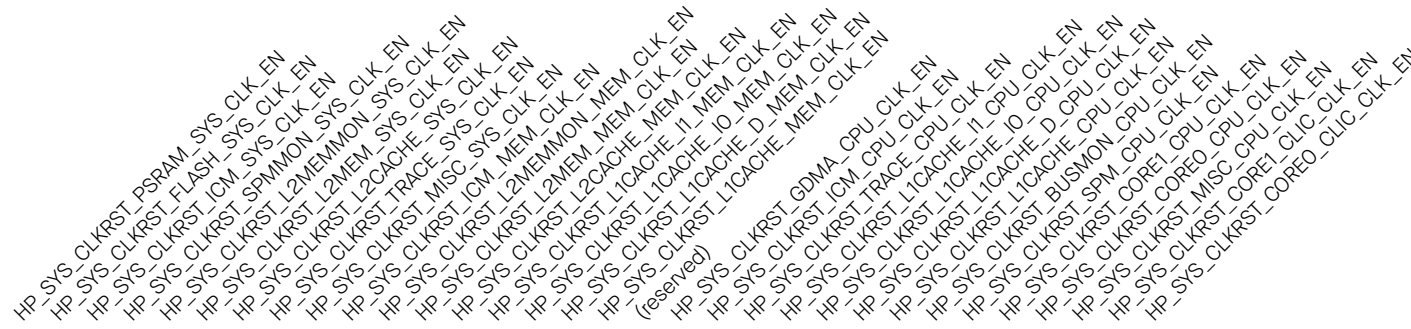
HP_SYS_CLKRST_APB_CLK_DIV_NUMERATOR Configures the numerator of the divisor's fractional part for APB_CLK. (R/W)

Register 11.5. HP_SYS_CLKRST_ROOT_CLK_CTRL3_REG (0x0010)



HP_SYS_CLKRST_APB_CLK_DIV_DENOMINATOR Configures the denominator of the divisor's fractional part for APB_CLK. (R/W)

Register 11.6. HP_SYS_CLKRST_SOC_CLK_CTRL0_REG (0x0014)



31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
1	1	1	0	0	1	1	0	1	1	0	1	1	1	1	1	1	0	0	1	0	1	1	1	1	0	1	0	1	1	1	1	Reset

HP_SYS_CLKRST_COREO_CLIC_CLK_EN Configures whether to enable the bus clock of HP CPU0 CLIC.

0: Disable

1: Enable

(R/W)

HP_SYS_CLKRST_CORE1_CLIC_CLK_EN Configures whether to enable the bus clock of HP CPU1 CLIC.

0: Disable

1: Enable

(R/W)

HP_SYS_CLKRST_MISC_CPU_CLK_EN Configures whether to enable the bus clock of miscellaneous modules (CPU_CLK clock domain).

0: Disable

1: Enable

(R/W)

HP_SYS_CLKRST_COREO_CPU_CLK_EN Configures whether to enable the bus clock of HP CPU0 (CPU_CLK clock domain).

0: Disable

1: Enable

(R/W)

HP_SYS_CLKRST_CORE1_CPU_CLK_EN Configures whether to enable the bus clock of HP CPU1 (CPU_CLK clock domain).

0: Disable

1: Enable

(R/W)

HP_SYS_CLKRST_SPM_CPU_CLK_EN Configures whether to enable the bus clock of SPM monitor (CPU_CLK clock domain).

0: Disable

1: Enable

(R/W)

Continued on the next page...

Register 11.6. HP_SYS_CLKRST_SOC_CLK_CTRL0_REG (0x0014)

Continued from the previous page...

HP_SYS_CLKRST_BUSMON_CPU_CLK_EN Configures whether to enable the bus clock of bus monitor (CPU_CLK clock domain).
0: Disable
1: Enable
(R/W)

HP_SYS_CLKRST_L1CACHE_CPU_CLK_EN Configures whether to enable the bus clock of L1 Cache (CPU_CLK clock domain).
0: Disable
1: Enable
(R/W)

HP_SYS_CLKRST_L1CACHE_D_CPU_CLK_EN Configures whether to enable the bus clock of L1 D-Cache (CPU_CLK clock domain).
0: Disable
1: Enable
(R/W)

HP_SYS_CLKRST_L1CACHE_I0_CPU_CLK_EN Configures whether to enable the bus clock of L1 I-Cache 0 (CPU_CLK clock domain).
0: Disable
1: Enable
(R/W)

HP_SYS_CLKRST_L1CACHE_I1_CPU_CLK_EN Configures whether to enable the bus clock of L1 I-Cache 1 (CPU_CLK clock domain).
0: Disable
1: Enable
(R/W)

HP_SYS_CLKRST_TRACE_CPU_CLK_EN Configures whether to enable the bus clock of RISC-V Trace Encoder (CPU_CLK clock domain).
0: Disable
1: Enable
(R/W)

HP_SYS_CLKRST_ICM_CPU_CLK_EN Configures the bus clock of the bus module (CPU_CLK clock domain).
0: Disable
1: Enable
(R/W)

Continued on the next page...

Register 11.6. HP_SYS_CLKRST_SOC_CLK_CTRL0_REG (0x0014)

Continued from the previous page...

HP_SYS_CLKRST_GDMA_CPU_CLK_EN Configures whether to enable the bus clock of VDMA (CPU_CLK clock domain).
0: Disable
1: Enable
(R/W)

HP_SYS_CLKRST_L1CACHE_MEM_CLK_EN Configures whether to enable the bus clock of L1 Cache (MEM_CLK clock domain).
0: Disable
1: Enable
(R/W)

HP_SYS_CLKRST_L1CACHE_D_MEM_CLK_EN Configures whether to enable the bus clock of L1 D-Cache (MEM_CLK clock domain).
0: Disable
1: Enable
(R/W)

HP_SYS_CLKRST_L1CACHE_I0_MEM_CLK_EN Configures whether to enable the bus clock of L1 I-Cache0 (MEM_CLK clock domain).
0: Disable
1: Enable
(R/W)

HP_SYS_CLKRST_L1CACHE_I1_MEM_CLK_EN Configures whether to enable the bus clock of L1 I-Cache1 (MEM_CLK clock domain).
0: Disable
1: Enable
(R/W)

HP_SYS_CLKRST_L2CACHE_MEM_CLK_EN Configures whether to enable the bus clock of L2 Cache (MEM_CLK clock domain).
0: Disable
1: Enable
(R/W)

Continued on the next page...

Register 11.6. HP_SYS_CLKRST_SOC_CLK_CTRL0_REG (0x0014)

Continued from the previous page...

HP_SYS_CLKRST_L2MEM_MEM_CLK_EN Configures whether to enable the bus clock of L2 memory (MEM_CLK clock domain).
0: Disable
1: Enable
(R/W)

HP_SYS_CLKRST_L2MEMMON_MEM_CLK_EN Configures whether to enable the bus clock of L1 memory monitor (MEM_CLK clock domain).
0: Disable
1: Enable
(R/W)

HP_SYS_CLKRST_ICM_MEM_CLK_EN Configures the bus clock of the bus module (MEM_CLK clock domain).
0: Disable
1: Enable
(R/W)

HP_SYS_CLKRST_MISC_SYS_CLK_EN Configures whether to enable the bus clock of miscellaneous modules (SYS_CLK clock domain).
0: Disable
1: Enable
(R/W)

HP_SYS_CLKRST_TRACE_SYS_CLK_EN Configures whether to enable the bus clock of RISC-V Trace Encoder (SYS_CLK clock domain).
0: Disable
1: Enable
(R/W)

HP_SYS_CLKRST_L2CACHE_SYS_CLK_EN Configures whether to enable the bus clock of L2 Cache (SYS_CLK clock domain).
0: Disable
1: Enable
(R/W)

HP_SYS_CLKRST_L2MEM_SYS_CLK_EN Configures whether to enable the bus clock of L2 memory (SYS_CLK clock domain).
0: Disable
1: Enable
(R/W)

Continued on the next page...

Register 11.6. HP_SYS_CLKRST_SOC_CLK_CTRL0_REG (0x0014)

Continued from the previous page...

HP_SYS_CLKRST_L2MEMMON_SYS_CLK_EN Configures whether to enable the bus clock of L2 memory monitor (SYS_CLK clock domain).
0: Disable
1: Enable
(R/W)

HP_SYS_CLKRST_SPMMON_SYS_CLK_EN Configures whether to enable the bus clock of L2 SPM monitor (SYS_CLK clock domain).
0: Disable
1: Enable
(R/W)

HP_SYS_CLKRST_ICM_SYS_CLK_EN Configures the bus clock of the bus module (SYS_CLK clock domain).
0: Disable
1: Enable
(R/W)

HP_SYS_CLKRST_FLASH_SYS_CLK_EN Configures whether to enable the bus clock of flash (SYS_CLK clock domain).
0: Disable
1: Enable
(R/W)

HP_SYS_CLKRST_PSRAM_SYS_CLK_EN Configures whether to enable the bus clock of PSRAM (SYS_CLK clock domain).
0: Disable
1: Enable
(R/W)

Register 11.7. HP_SYS_CLKRST_SOC_CLK_CTRL1_REG (0x0018)



31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
0	1	1	1	1	1	0	0	0	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	1	1	0	1	1	Reset

HP_SYS_CLKRST_GPSPI2_SYS_CLK_EN Configures whether to enable the bus clock of GP-SPI2 (SYS_CLK clock domain).

0: Disable

1: Enable

(R/W)

HP_SYS_CLKRST_GPSPI3_SYS_CLK_EN Configures whether to enable the bus clock of GP-SPI3 (SYS_CLK clock domain).

0: Disable

1: Enable

(R/W)

HP_SYS_CLKRST_AHB_PDMA_SYS_CLK_EN Configures whether to enable the bus clock of GDMA-AHB (SYS_CLK clock domain).

0: Disable

1: Enable

(R/W)

HP_SYS_CLKRST_AXI_PDMA_SYS_CLK_EN Configures whether to enable the bus clock of GDMA-AXI (SYS_CLK clock domain).

0: Disable

1: Enable

(R/W)

HP_SYS_CLKRST_GDMA_SYS_CLK_EN Configures whether to enable the bus clock of VDMA (SYS_CLK clock domain).

0: Disable

1: Enable

(R/W)

Continued on the next page...

Register 11.7. HP_SYS_CLKRST_SOC_CLK_CTRL1_REG (0x0018)

Continued from the previous page...

HP_SYS_CLKRST_DMA2D_SYS_CLK_EN Configures whether to enable the bus clock of 2DDMA (SYS_CLK clock domain).
0: Disable
1: Enable
(R/W)

HP_SYS_CLKRST_JPEG_SYS_CLK_EN Configures whether to enable the bus clock of JPEG (SYS_CLK clock domain).
0: Disable
1: Enable
(R/W)

HP_SYS_CLKRST_PPA_SYS_CLK_EN Configures whether to enable the bus clock of PPA (SYS_CLK clock domain).
0: Disable
1: Enable
(R/W)

HP_SYS_CLKRST_CSI_BRG_SYS_CLK_EN Configures whether to enable the bus clock of CSI Bridge and ISP (SYS_CLK clock domain).
0: Disable
1: Enable
(R/W)

HP_SYS_CLKRST_CSI_HOST_SYS_CLK_EN Configures whether to enable the bus clock of CSI Host (SYS_CLK clock domain).
0: Disable
1: Enable
(R/W)

HP_SYS_CLKRST_DSI_SYS_CLK_EN Configures whether to enable the bus clock of DSI (SYS_CLK clock domain).
0: Disable
1: Enable
(R/W)

Continued on the next page...

Register 11.7. HP_SYS_CLKRST_SOC_CLK_CTRL1_REG (0x0018)

Continued from the previous page...

HP_SYS_CLKRST_EMAC_SYS_CLK_EN Configures whether to enable the bus clock of EMAC (SYS_CLK clock domain).
0: Disable
1: Enable
(R/W)

HP_SYS_CLKRST_SDMMC_SYS_CLK_EN Configures whether to enable the bus clock of SDMMC (SYS_CLK clock domain).
0: Disable
1: Enable
(R/W)

HP_SYS_CLKRST_USB_OTG11_SYS_CLK_EN Configures whether to enable the bus clock of USB 2.0 Full-Speed OTG (SYS_CLK clock domain).
0: Disable
1: Enable
(R/W)

HP_SYS_CLKRST_USB_OTG20_SYS_CLK_EN Configures whether to enable the bus clock of USB 2.0 High-Speed OTG (SYS_CLK clock domain).
0: Disable
1: Enable
(R/W)

HP_SYS_CLKRST_UHCI_SYS_CLK_EN Configures whether to enable the bus clock of UHCI (SYS_CLK clock domain).
0: Disable
1: Enable
(R/W)

HP_SYS_CLKRST_UART0_SYS_CLK_EN Configures whether to enable the bus clock of UART0 (SYS_CLK clock domain).
0: Disable
1: Enable
(R/W)

HP_SYS_CLKRST_UART1_SYS_CLK_EN Configures whether to enable the bus clock of UART1 (SYS_CLK clock domain).
0: Disable
1: Enable
(R/W)

Continued on the next page...

Register 11.7. HP_SYS_CLKRST_SOC_CLK_CTRL1_REG (0x0018)

Continued from the previous page...

HP_SYS_CLKRST_UART2_SYS_CLK_EN Configures whether to enable the bus clock of UART2 (SYS_CLK clock domain).
0: Disable
1: Enable
(R/W)

HP_SYS_CLKRST_UART3_SYS_CLK_EN Configures whether to enable the bus clock of UART3 (SYS_CLK clock domain).
0: Disable
1: Enable
(R/W)

HP_SYS_CLKRST_UART4_SYS_CLK_EN Configures whether to enable the bus clock of UART4 (SYS_CLK clock domain).
0: Disable
1: Enable
(R/W)

HP_SYS_CLKRST_PARLIO_SYS_CLK_EN Configures whether to enable the bus clock of PARLIO (SYS_CLK clock domain).
0: Disable
1: Enable
(R/W)

HP_SYS_CLKRST_ETM_SYS_CLK_EN Configures whether to enable the bus clock of ETM (SYS_CLK clock domain).
0: Disable
1: Enable
(R/W) 0: Disable
1: Enable
(R/W)

HP_SYS_CLKRST_CRYPTO_SYS_CLK_EN Configures whether to enable the bus clock of the crypto modules (SYS_CLK clock domain).
0: Disable
1: Enable
(R/W)

Continued on the next page...

Register 11.7. HP_SYS_CLKRST_SOC_CLK_CTRL1_REG (0x0018)

Continued from the previous page...

HP_SYS_CLKRST_KEY_MANAGER_SYS_CLK_EN Configures whether to enable the bus clock of the Key Manager (SYS_CLK clock domain).
0: Disable
1: Enable
(R/W)

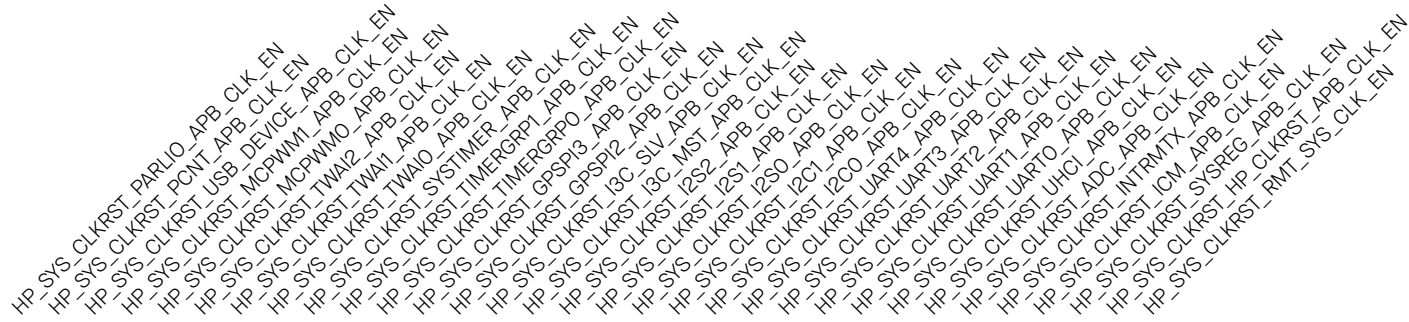
HP_SYS_CLKRST_BITSRAMBLER_SYS_CLK_EN Configures whether to enable the bus clock of the Bit Scrambler (SYS_CLK clock domain).
0: Disable
1: Enable
(R/W)

HP_SYS_CLKRST_BITSRAMBLER_RX_SYS_CLK_EN Configures whether to enable the bus clock of the Bit Scrambler RX (SYS_CLK clock domain).
0: Disable
1: Enable
(R/W)

HP_SYS_CLKRST_BITSRAMBLER_TX_SYS_CLK_EN Configures whether to enable the bus clock of the Bit Scrambler TX (SYS_CLK clock domain).
0: Disable
1: Enable
(R/W)

HP_SYS_CLKRST_H264_SYS_CLK_EN Configures whether to enable the bus clock of H264 (SYS_CLK clock domain).
0: Disable
1: Enable
(R/W)

Register 11.8. HP_SYS_CLKRST_SOC_CLK_CTRL2_REG (0x001C)



31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
0	0	1	0	0	0	0	0	1	1	1	1	1	0	0	0	0	0	0	0	1	1	1	1	1	1	0	1	1	1	1	0	Reset

HP_SYS_CLKRST_RMT_SYS_CLK_EN Configures whether to enable the bus clock of RMT (SYS_CLK clock domain).

0: Disable

1: Enable

(R/W)

HP_SYS_CLKRST_HP_CLKRST_APB_CLK_EN Configures whether to enable the bus clock of the Reset and Clock module (APB_CLK clock domain).

0: Disable

1: Enable

(R/W)

HP_SYS_CLKRST_SYSREG_APB_CLK_EN Configures whether to enable the bus clock of the HP System Registers (APB_CLK clock domain).

0: Disable

1: Enable

(R/W)

HP_SYS_CLKRST_ICM_APB_CLK_EN Configures the bus clock of the bus module (APB_CLK clock domain).

0: Disable

1: Enable

(R/W)

HP_SYS_CLKRST_INTRMTX_APB_CLK_EN Configures whether to enable the bus clock of the Interrupt Matrix (APB_CLK clock domain).

0: Disable

1: Enable

(R/W)

HP_SYS_CLKRST_ADC_APB_CLK_EN Configures whether to enable the bus clock of the ADC (APB_CLK clock domain).

0: Disable

1: Enable

(R/W)

Continued on the next page...

Register 11.8. HP_SYS_CLKRST_SOC_CLK_CTRL2_REG (0x001C)

Continued from the previous page...

HP_SYS_CLKRST_UHCI_APB_CLK_EN Configures whether to enable the bus clock of the UHCI (APB_CLK clock domain).
0: Disable
1: Enable
(R/W)

HP_SYS_CLKRST_UART0_APB_CLK_EN Configures whether to enable the bus clock of UART0 (APB_CLK clock domain).
0: Disable
1: Enable
(R/W)

HP_SYS_CLKRST_UART1_APB_CLK_EN Configures whether to enable the bus clock of UART1 (APB_CLK clock domain).
0: Disable
1: Enable
(R/W)

HP_SYS_CLKRST_UART2_APB_CLK_EN Configures whether to enable the bus clock of UART2 (APB_CLK clock domain).
0: Disable
1: Enable
(R/W)

HP_SYS_CLKRST_UART3_APB_CLK_EN Configures whether to enable the bus clock of UART3 (APB_CLK clock domain).
0: Disable
1: Enable
(R/W)

HP_SYS_CLKRST_UART4_APB_CLK_EN Configures whether to enable the bus clock of UART4 (APB_CLK clock domain).
0: Disable
1: Enable
(R/W)

HP_SYS_CLKRST_I2C0_APB_CLK_EN Configures whether to enable the bus clock of I2C0 (APB_CLK clock domain).
0: Disable
1: Enable
(R/W)

Continued on the next page...

Register 11.8. HP_SYS_CLKRST_SOC_CLK_CTRL2_REG (0x001C)

Continued from the previous page...

HP_SYS_CLKRST_I2C1_APB_CLK_EN Configures whether to enable the bus clock of I2C1 (APB_CLK clock domain).
0: Disable
1: Enable
(R/W)

HP_SYS_CLKRST_I2S0_APB_CLK_EN Configures whether to enable the bus clock of I2S0 (APB_CLK clock domain).
0: Disable
1: Enable
(R/W)

HP_SYS_CLKRST_I2S1_APB_CLK_EN Configures whether to enable the bus clock of I2S1 (APB_CLK clock domain).
0: Disable
1: Enable
(R/W)

HP_SYS_CLKRST_I2S2_APB_CLK_EN Configures whether to enable the bus clock of I2S2 (APB_CLK clock domain).
0: Disable
1: Enable
(R/W)

HP_SYS_CLKRST_I3C_MST_APB_CLK_EN Configures whether to enable the bus clock of the I3C Master (APB_CLK clock domain).
0: Disable
1: Enable
(R/W)

HP_SYS_CLKRST_I3C_SLV_APB_CLK_EN Configures whether to enable the bus clock of the I3C Slave (APB_CLK clock domain).
0: Disable
1: Enable
(R/W)

HP_SYS_CLKRST_GPSPI2_APB_CLK_EN Configures whether to enable the bus clock of GP-SPI2 (APB_CLK clock domain).
0: Disable
1: Enable
(R/W)

Continued on the next page...

Register 11.8. HP_SYS_CLKRST_SOC_CLK_CTRL2_REG (0x001C)

Continued from the previous page...

HP_SYS_CLKRST_GPSPI3_APB_CLK_EN Configures whether to enable the bus clock of GP-SPI3 (APB_CLK clock domain).
0: Disable
1: Enable
(R/W)

HP_SYS_CLKRST_TIMERGRP0_APB_CLK_EN Configures whether to enable the bus clock of Timer Group 0 (APB_CLK clock domain).
0: Disable
1: Enable
(R/W)

HP_SYS_CLKRST_TIMERGRP1_APB_CLK_EN Configures whether to enable the bus clock of Timer Group 1 (APB_CLK clock domain).
0: Disable
1: Enable
(R/W)

HP_SYS_CLKRST_SYSTIMER_APB_CLK_EN Configures whether to enable the bus clock of the System Timer (APB_CLK clock domain).
0: Disable
1: Enable
(R/W)

HP_SYS_CLKRST_TWAI0_APB_CLK_EN Configures whether to enable the bus clock of TWAI0 (APB_CLK clock domain).
0: Disable
1: Enable
(R/W)

HP_SYS_CLKRST_TWAI1_APB_CLK_EN Configures whether to enable the bus clock of TWAI1 (APB_CLK clock domain).
0: Disable
1: Enable
(R/W)

HP_SYS_CLKRST_TWAI2_APB_CLK_EN Configures whether to enable the bus clock of TWAI2 (APB_CLK clock domain).
0: Disable
1: Enable
(R/W)

Continued on the next page...

Register 11.8. HP_SYS_CLKRST_SOC_CLK_CTRL2_REG (0x001C)

Continued from the previous page...

HP_SYS_CLKRST_MCPWM0_APB_CLK_EN Configures whether to enable the bus clock of MCPWM0 (APB_CLK clock domain).
0: Disable
1: Enable
(R/W)

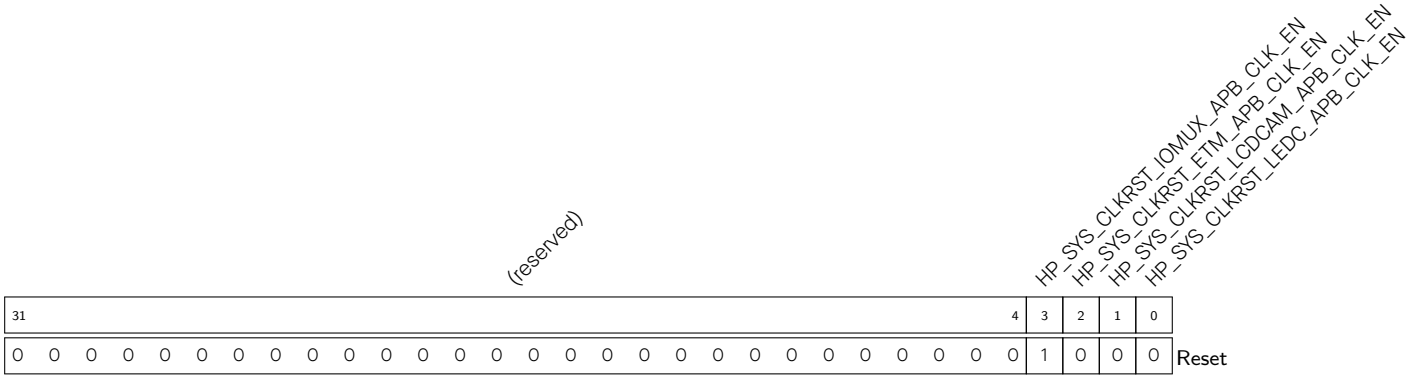
HP_SYS_CLKRST_MCPWM1_APB_CLK_EN Configures whether to enable the bus clock of MCPWM1 (APB_CLK clock domain).
0: Disable
1: Enable
(R/W)

HP_SYS_CLKRST_USB_DEVICE_APB_CLK_EN Configures whether to enable the bus clock of the USB Serial/JTAG (APB_CLK clock domain).
0: Disable
1: Enable
(R/W)

HP_SYS_CLKRST_PCNT_APB_CLK_EN Configures whether to enable the bus clock of the PCNT (APB_CLK clock domain).
0: Disable
1: Enable
(R/W)

HP_SYS_CLKRST_PARLIO_APB_CLK_EN Configures whether to enable the bus clock of the PARLIO (APB_CLK clock domain).
0: Disable
1: Enable
(R/W)

Register 11.9. HP_SYS_CLKRST_SOC_CLK_CTRL3_REG (0x0020)



HP_SYS_CLKRST_LEDC_APB_CLK_EN Configures whether to enable the bus clock of LEDC (APB_CLK clock domain).

0: Disable

1: Enable

(R/W)

HP_SYS_CLKRST_LCDCAM_APB_CLK_EN Configures whether to enable the bus clock of the LCD_CAM (APB_CLK clock domain).

0: Disable

1: Enable

(R/W)

HP_SYS_CLKRST_ETM_APB_CLK_EN Configures whether to enable the bus clock of ETM (APB_CLK clock domain).

0: Disable

1: Enable

(R/W)

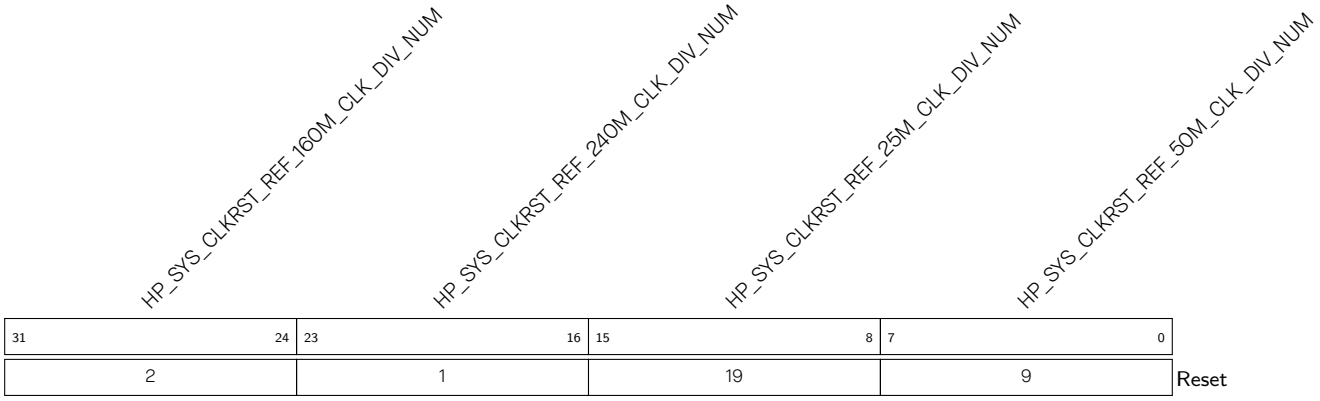
HP_SYS_CLKRST_IOMUX_APB_CLK_EN Configures whether to enable the bus clock of IO MUX (APB_CLK clock domain).

0: Disable

1: Enable

(R/W)

Register 11.10. HP_SYS_CLKRST_REF_CLK_CTRL0_REG (0x0024)



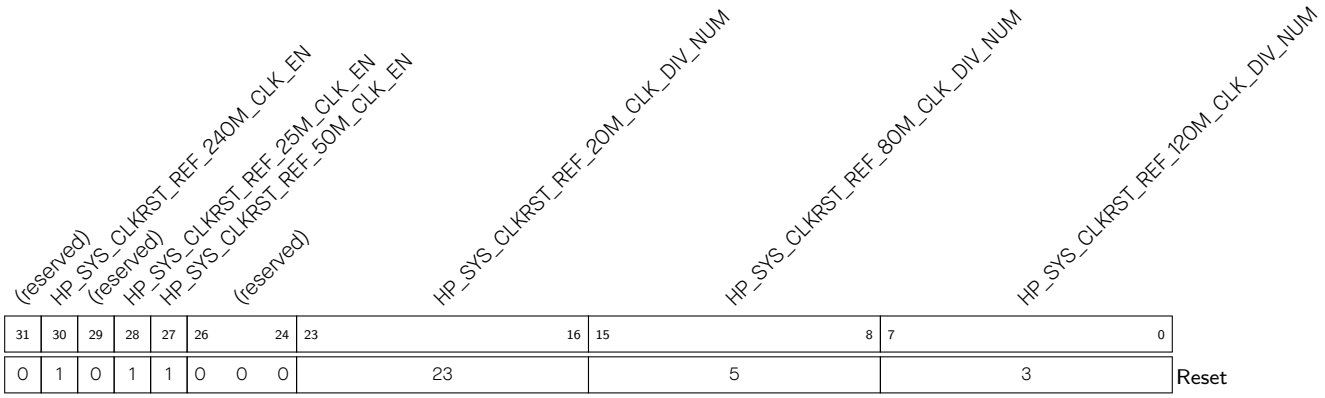
HP_SYS_CLKRST_REF_50M_CLK_DIV_NUM Configures the divisor of PLL_F50M_CLK. It is not recommended for users to change. (R/W)

HP_SYS_CLKRST_REF_25M_CLK_DIV_NUM Configures the divisor of PLL_F25M_CLK. It is not recommended for users to change. (R/W)

HP_SYS_CLKRST_REF_240M_CLK_DIV_NUM Configures the divisor of PLL_F240M_CLK. It is not recommended for users to change. (R/W)

HP_SYS_CLKRST_REF_160M_CLK_DIV_NUM Configures the divisor of PLL_F160M_CLK. It is not recommended for users to change. (R/W)

Register 11.11. HP_SYS_CLKRST_REF_CLK_CTRL1_REG (0x0028)



HP_SYS_CLKRST_REF_120M_CLK_DIV_NUM Configures the divisor of PLL_F120M_CLK. (R/W)

HP_SYS_CLKRST_REF_80M_CLK_DIV_NUM Configures the divisor of PLL_F80M_CLK. (R/W)

HP_SYS_CLKRST_REF_20M_CLK_DIV_NUM Configures the divisor of PLL_F20M_CLK. (R/W)

HP_SYS_CLKRST_REF_50M_CLK_EN Configures whether to enable PLL_F50M_CLK.

0: Disable

1: Enable

(R/W)

HP_SYS_CLKRST_REF_25M_CLK_EN Configures whether to enable PLL_F25M_CLK.

0: Disable

1: Enable

(R/W)

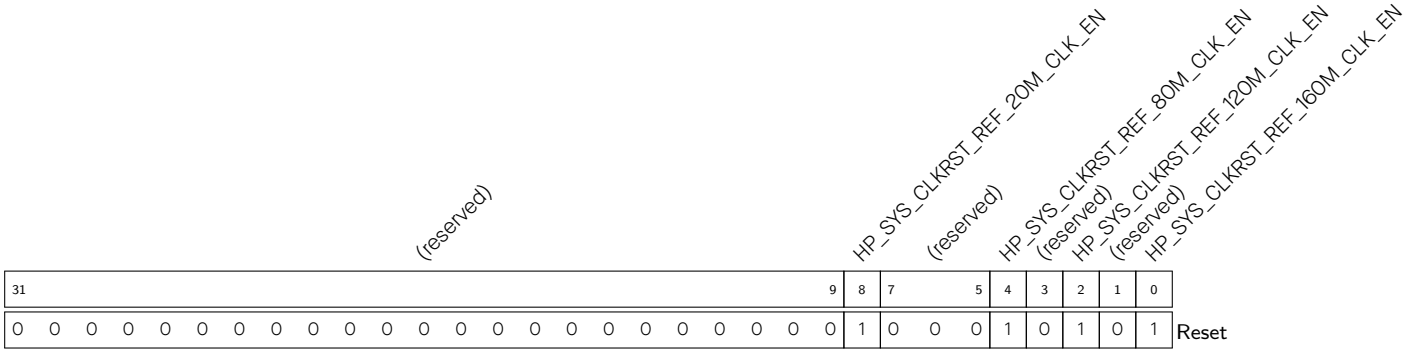
HP_SYS_CLKRST_REF_240M_CLK_EN Configures whether to enable PLL_F240M_CLK.

0: Disable

1: Enable

(R/W)

Register 11.12. HP_SYS_CLKRST_REF_CLK_CTRL2_REG (0x002C)



HP_SYS_CLKRST_REF_160M_CLK_EN Configures whether to enable PLL_F160M_CLK.

0: Disable

1: Enable

User modification is not recommended. (R/W)

HP_SYS_CLKRST_REF_120M_CLK_EN Configures whether to enable PLL_F120M_CLK.

0: Disable

1: Enable

User modification is not recommended. (R/W)

HP_SYS_CLKRST_REF_80M_CLK_EN Configures whether to enable PLL_F80M_CLK.

0: Disable

1: Enable

User modification is not recommended. (R/W)

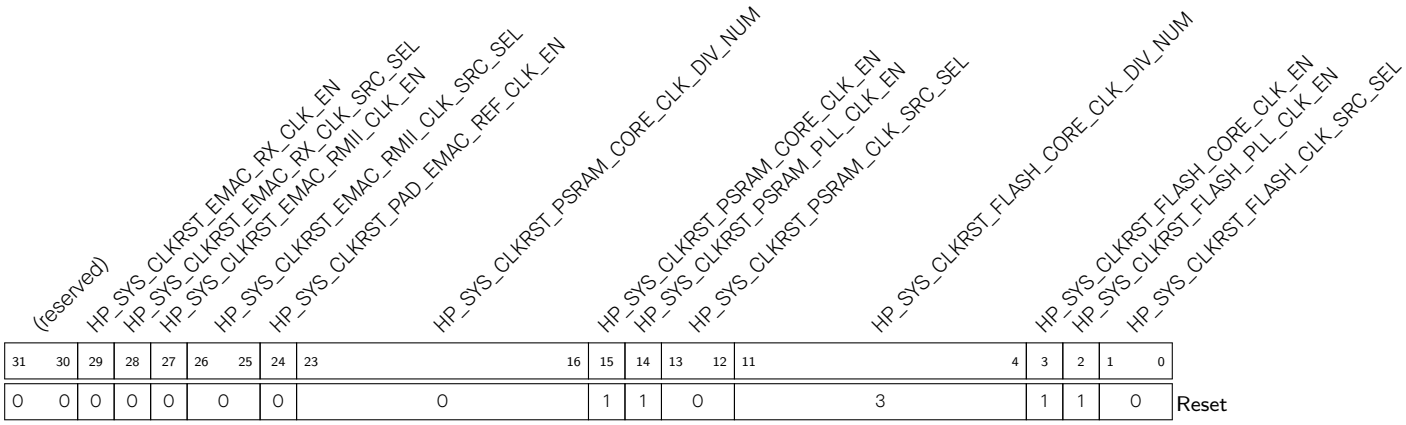
HP_SYS_CLKRST_REF_20M_CLK_EN Configures whether to enable PLL_F20M_CLK.

0: Disable

1: Enable

User modification is not recommended. (R/W)

Register 11.13. HP_SYS_CLKRST_PERI_CLK_CTRL00_REG (0x0030)



HP_SYS_CLKRST_FLASH_CLK_SRC_SEL Configures the clock source for FLASH_CLK.

0: XTAL_CLK

1: SPLL_CLK (480 MHz)

2: CPLL_CLK (360 MHz)

3: Invalid

(R/W)

HP_SYS_CLKRST_FLASH_PLL_CLK_EN Configures whether to enable FLASH_PLL_CLK.

0: Disable

1: Enable

(R/W)

HP_SYS_CLKRST_FLASH_CORE_CLK_EN Configures whether to enable FLASH_CORE_CLK.

0: Disable

1: Enable

(R/W)

HP_SYS_CLKRST_FLASH_CORE_CLK_DIV_NUM Configures the clock divisor of FLASH_CORE_CLK. (R/W)

HP_SYS_CLKRST_PSRAM_CLK_SRC_SEL Configures the clock source for PSRAM_CLK.

0: XTAL_CLK

1: MPLL_CLK

2: SPLL_CLK (480 MHz)

3: CPLL_CLK (360 MHz)

(R/W)

HP_SYS_CLKRST_PSRAM_PLL_CLK_EN Configures whether to enable PSRAM_PLL_CLK.

0: Disable

1: Enable

(R/W)

HP_SYS_CLKRST_PSRAM_CORE_CLK_EN Configures whether to enable PSRAM_CORE_CLK.

0: Disable

1: Enable

(R/W)

Continued on the next page...

Register 11.13. HP_SYS_CLKRST_PERI_CLK_CTRL00_REG (0x0030)

Continued from the previous page...

HP_SYS_CLKRST_PSRAM_CORE_CLK_DIV_NUM Configures the clock divisor of PSRAM_CORE_CLK. (R/W)

HP_SYS_CLKRST_PAD_EMAC_REF_CLK_EN Configures whether to enable PAD_EMAC_REF_CLK.
0: Disable
1: Enable
(R/W)

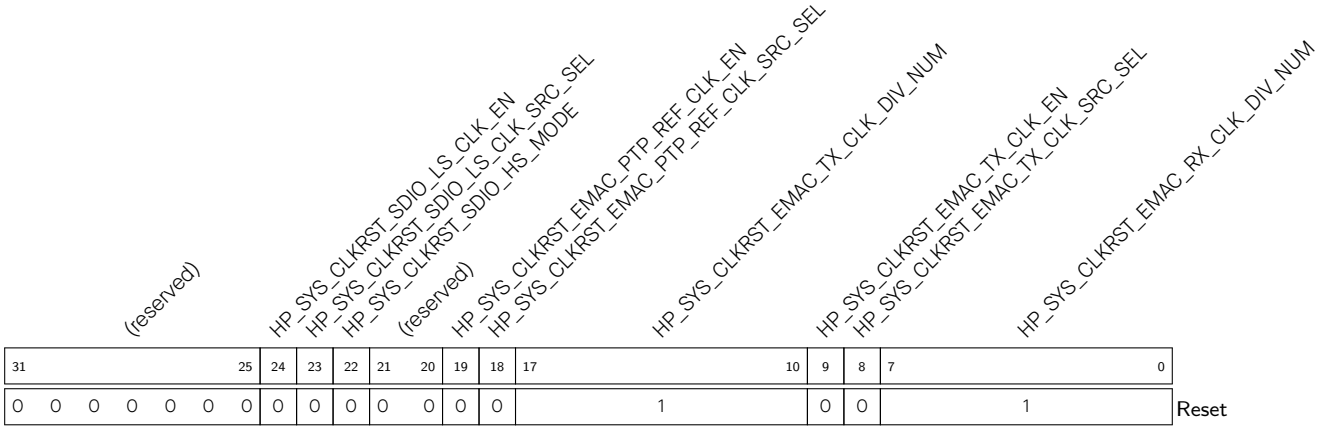
HP_SYS_CLKRST_EMAC_RMII_CLK_SRC_SEL Configures the clock source for EMAC_RMII_CLK.
0: PAD_EMAC_TXRX_CLK
1: PAD_EMAC_RX_CLK
2: PAD_EMAC_TX_CLK
3: Invalid
(R/W)

HP_SYS_CLKRST_EMAC_RMII_CLK_EN Configures whether to enable EMAC_RMII_CLK.
0: Disable
1: Enable
(R/W)

HP_SYS_CLKRST_EMAC_RX_CLK_SRC_SEL Configures the clock source for EMAC_RX_CLK.
0: PAD_EMAC_TXRX_CLK
1: PAD_EMAC_RX_CLK
(R/W)

HP_SYS_CLKRST_EMAC_RX_CLK_EN Configures whether to enable EMAC_RX_CLK.
0: Disable
1: Enable
(R/W)

Register 11.14. HP_SYS_CLKRST_PERI_CLK_CTRL01_REG (0x0034)



HP_SYS_CLKRST_EMAC_RX_CLK_DIV_NUM Configures the clock divisor for EMAC_RX_CLK. (R/W)

HP_SYS_CLKRST_EMAC_TX_CLK_SRC_SEL Configures the clock source for EMAC_TX_CLK.

0: PAD_EMAC_TXRX_CLK

1: PAD_EMAC_TX_CLK

(R/W)

HP_SYS_CLKRST_EMAC_TX_CLK_EN Configures whether to enable EMAC_TX_CLK.

0: Disable

1: Enable

(R/W)

HP_SYS_CLKRST_EMAC_TX_CLK_DIV_NUM Configures the clock divisor for EMAC_TX_CLK. (R/W)

HP_SYS_CLKRST_EMAC_PTP_REF_CLK_SRC_SEL Configures the clock source for EMAC_PTP_REF_CLK.

0: XTAL_CLK

1: PLL_F80M_CLK

(R/W)

HP_SYS_CLKRST_EMAC_PTP_REF_CLK_EN Configures whether to enable EMAC_PTP_REF_CLK.

0: Disable

1: Enable

(R/W)

Continued on the next page...

Register 11.14. HP_SYS_CLKRST_PERI_CLK_CTRL01_REG (0x0034)

Continued from the previous page...

HP_SYS_CLKRST_SDIO_HS_MODE Configures the mode of SDIO clock.

0: Low-speed mode

1: High-speed mode

(R/W)

HP_SYS_CLKRST_SDIO_LS_CLK_SRC_SEL Configures the low-speed clock source for SDIO.

0: PLL_F160M_CLK

1: APLL_CLK

(R/W)

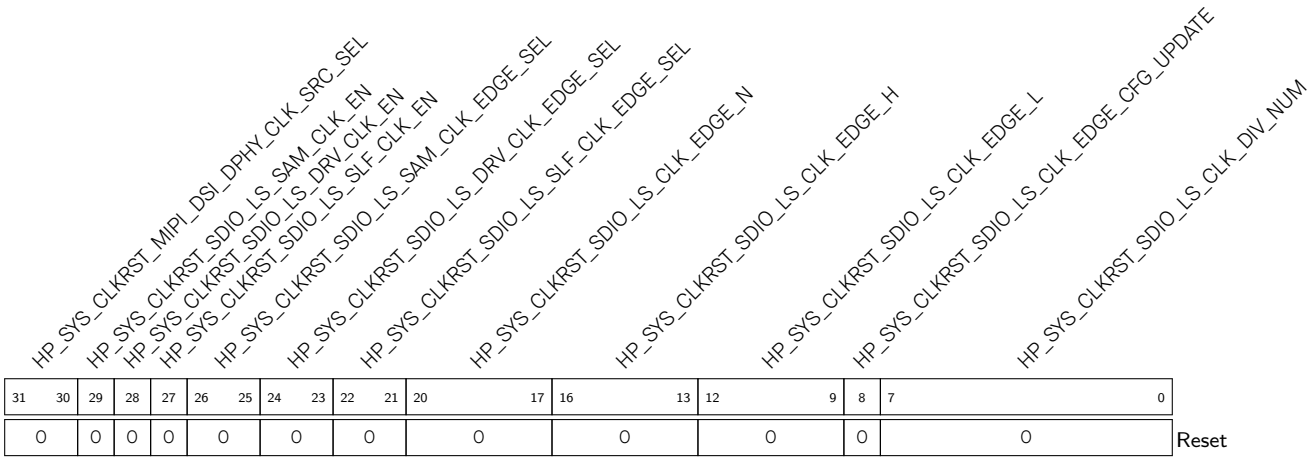
HP_SYS_CLKRST_SDIO_LS_CLK_EN Configures whether to enable SDIO low-speed clock.

0: Disable

1: Enable

(R/W)

Register 11.15. HP_SYS_CLKRST_PERI_CLK_CTRL02_REG (0x0038)



HP_SYS_CLKRST_SDIO_LS_CLK_DIV_NUM Configures the divisor of the SDIO low-speed clock. (R/W)

HP_SYS_CLKRST_SDIO_LS_CLK_EDGE_CFG_UPDATE Configures whether to update the SDIO low-speed clock phase configuration.

0: Not update

1: Update (WT)

HP_SYS_CLKRST_SDIO_LS_CLK_EDGE_L Configures the threshold for the SDIO low-speed clock signal to switch to the low level. (R/W)

HP_SYS_CLKRST_SDIO_LS_CLK_EDGE_H Configures the threshold for the SDIO low-speed clock signal to switch to the high level. (R/W)

HP_SYS_CLKRST_SDIO_LS_CLK_EDGE_N Configures the threshold for clearing the count value of the SDIO low-speed clock. (R/W)

HP_SYS_CLKRST_SDIO_LS_SLF_CLK_EDGE_SEL Configures SDIO_SLF_CLK to select the SDIO low-speed clock phase. (R/W)

HP_SYS_CLKRST_SDIO_LS_DRV_CLK_EDGE_SEL Configures SDIO_DRV_CLK to select the SDIO low-speed clock phase. (R/W)

HP_SYS_CLKRST_SDIO_LS_SAM_CLK_EDGE_SEL Configures SDIO_SAM_CLK to select the SDIO low-speed clock phase. (R/W)

HP_SYS_CLKRST_SDIO_LS_SLF_CLK_EN Configures whether to enable SDIO_SLF_CLK.

0: Disable

1: Enable

(R/W)

Continued on the next page...

Register 11.15. HP_SYS_CLKRST_PERI_CLK_CTRL02_REG (0x0038)

Continued from the previous page...

HP_SYS_CLKRST_SDIO_LS_DRV_CLK_EN Configures whether to enable SDIO_DRV_CLK.

0: Disable

1: Enable

(R/W)

HP_SYS_CLKRST_SDIO_LS_SAM_CLK_EN Configures whether to enable SDIO_SAM_CLK.

0: Disable

1: Enable

(R/W)

HP_SYS_CLKRST_MIPI_DSI_DPHY_CLK_SRC_SEL Configures the clock source for DSI_DPHY_CFG_CLK and DSI_DPHY_PLL_REFCLK.

0: PLL_F20M_CLK

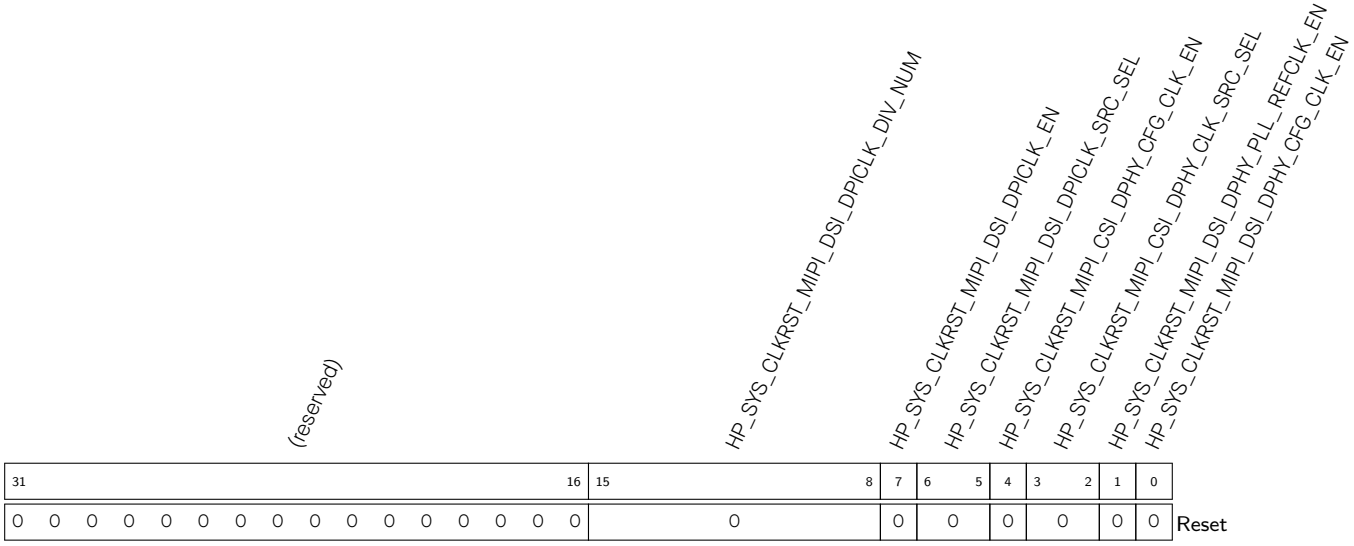
1: RC_FAST_CLK

2: PLL_F25M_CLK

3: Invalid

(R/W)

Register 11.16. HP_SYS_CLKRST_PERI_CLK_CTRL03_REG (0x003C)



HP_SYS_CLKRST_MIPI_DSI_DPHY_CFG_CLK_EN Configures whether to enable DSI_DPHY_CFG_CLK clock.

0: Disable

1: Enable

(R/W)

HP_SYS_CLKRST_MIPI_DSI_DPHY_PLL_REFCLK_EN Configures whether to enable DSI_DPHY_PLL_REFCLK clock.

0: Disable

1: Enable

(R/W)

HP_SYS_CLKRST_MIPI_CSI_DPHY_CLK_SRC_SEL Configures the clock source for CSI_DPHY_CFG_CLK.

0: PLL_F20M_CLK

1: RC_FAST_CLK

2: PLL_F25M_CLK

3: Invalid

(R/W)

HP_SYS_CLKRST_MIPI_CSI_DPHY_CFG_CLK_EN Configures whether to enable CSI_DPHY_CFG_CLK clock.

0: Disable

1: Enable

(R/W)

HP_SYS_CLKRST_MIPI_DSI_DPICLK_SRC_SEL Configures the clock source for DSI_DPI_CLK.

0: XTAL_CLK

1: PLL_F240M_CLK

2: PLL_F160M_CLK

3: Invalid

(R/W)

Continued on the next page...

Register 11.16. HP_SYS_CLKRST_PERI_CLK_CTRL03_REG (0x003C)

Continued from the previous page...

HP_SYS_CLKRST_MIPI_DSI_DPICLK_EN Configures whether to enable DSI_DPI_CLK clock.

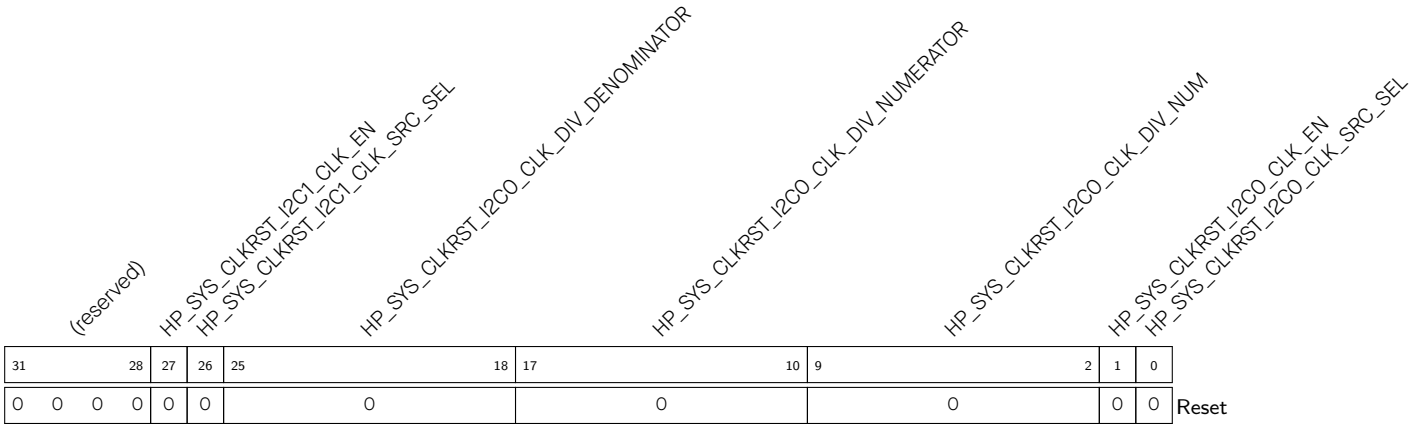
0: Disable

1: Enable

(R/W)

HP_SYS_CLKRST_MIPI_DSI_DPICLK_DIV_NUM Configures the clock divisor of DSI_DPI_CLK. (R/W)

Register 11.17. HP_SYS_CLKRST_PERI_CLK_CTRL10_REG (0x0040)



HP_SYS_CLKRST_I2C0_CLK_SRC_SEL Configures the clock source for I2C0_CLK.

0: XTAL_CLK

1: RC_FAST_CLK

(R/W)

HP_SYS_CLKRST_I2C0_CLK_EN Configures whether to enable I2C0_CLK.

0: Disable

1: Enable

(R/W)

HP_SYS_CLKRST_I2C0_CLK_DIV_NUM Configures the integer part of the clock divisor of I2C0_CLK. (R/W)

HP_SYS_CLKRST_I2C0_CLK_DIV_NUMERATOR Configures the numerator of the divisor's fractional part for clock divisor of I2C0_CLK. (R/W)

HP_SYS_CLKRST_I2C0_CLK_DIV_DENOMINATOR Configures the denominator of the divisor's fractional part for clock divisor of I2C0_CLK. (R/W)

HP_SYS_CLKRST_I2C1_CLK_SRC_SEL Configures the clock source for I2C1_CLK.

0: XTAL_CLK

1: RC_FAST_CLK

(R/W)

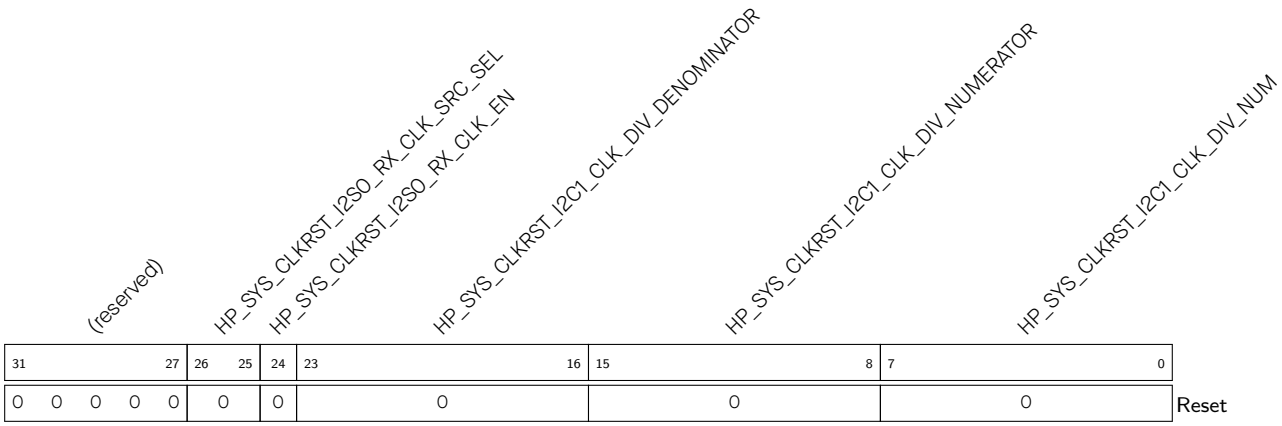
HP_SYS_CLKRST_I2C1_CLK_EN Configures whether to enable I2C1_CLK.

0: Disable

1: Enable

(R/W)

Register 11.18. HP_SYS_CLKRST_PERI_CLK_CTRL11_REG (0x0044)



HP_SYS_CLKRST_I2C1_CLK_DIV_NUM Configures the integer part of the I2C1_CLK clock divisor. (R/W)

HP_SYS_CLKRST_I2C1_CLK_DIV_NUMERATOR Configures the numerator of the divisor's fractional part for I2C1_CLK clock divisor. (R/W)

HP_SYS_CLKRST_I2C1_CLK_DIV_DENOMINATOR Configures the denominator of the divisor's fractional part for I2C1_CLK clock divisor. (R/W)

HP_SYS_CLKRST_I2S0_RX_CLK_EN Configures whether to enable I2S0_RX_CLK.

0: Disable

1: Enable

(R/W)

HP_SYS_CLKRST_I2S0_RX_CLK_SRC_SEL Configures the clock source for I2S0_RX_CLK.

0: XTAL_CLK

1: APLL_CLK

2: PAD_I2S0_MCLK

3: Invalid

(R/W)

Register 11.19. HP_SYS_CLKRST_PERI_CLK_CTRL12_REG (0x0048)

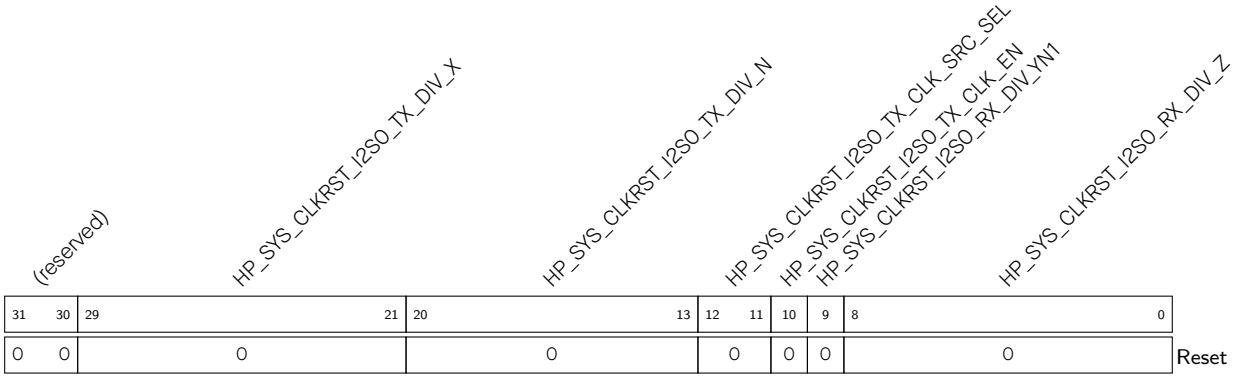


HP_SYS_CLKRST_I2S0_RX_DIV_N Configures the integer part of the I2S0_RX_CLK clock divisor. (R/W)

HP_SYS_CLKRST_I2S0_RX_DIV_X Configures the coefficient x of the I2S0_RX_CLK clock divisor. (R/W)

HP_SYS_CLKRST_I2S0_RX_DIV_Y Configures the coefficient y of the I2S0_RX_CLK clock divisor. (R/W)

Register 11.20. HP_SYS_CLKRST_PERI_CLK_CTRL13_REG (0x004C)



HP_SYS_CLKRST_I2S0_RX_DIV_Z Configures the coefficient z of I2S0_RX_CLK clock divisor. (R/W)

HP_SYS_CLKRST_I2S0_RX_DIV_YN1 Configures the coefficient yn1 of I2S0_RX_CLK clock divisor. (R/W)

HP_SYS_CLKRST_I2S0_TX_CLK_EN Configures whether to enable I2S0_TX_CLK.

0: Disable

1: Enable

(R/W)

HP_SYS_CLKRST_I2S0_TX_CLK_SRC_SEL Configures the clock source for I2S0_TX_CLK.

0: XTAL_CLK

1: APLL_CLK

2: PAD_I2S0_MCLK

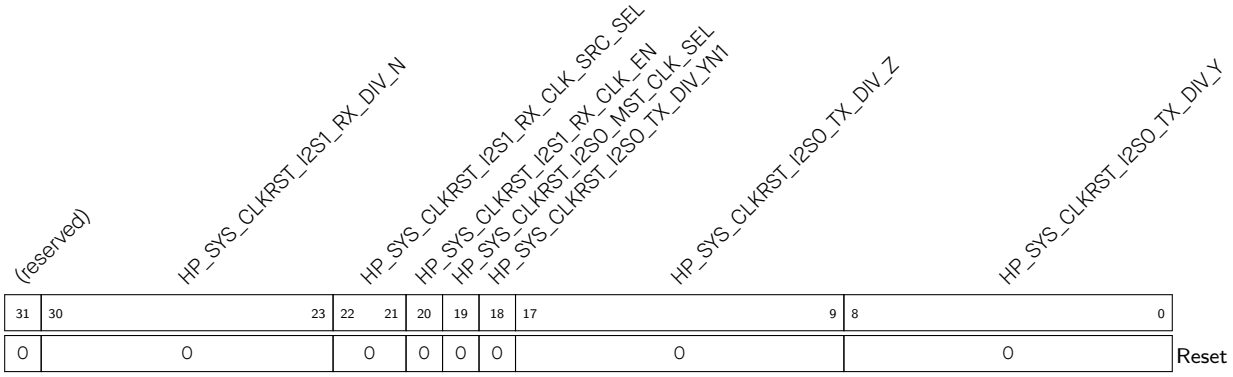
3: Invalid

(R/W)

HP_SYS_CLKRST_I2S0_TX_DIV_N Configures the integer part of the clock divisor of I2S0_TX_CLK. (R/W)

HP_SYS_CLKRST_I2S0_TX_DIV_X Configures the coefficient x of I2S0_TX_CLK clock divisor. (R/W)

Register 11.21. HP_SYS_CLKRST_PERI_CLK_CTRL14_REG (0x0050)



HP_SYS_CLKRST_I2S0_TX_DIV_Y Configures the coefficient y for I2S0_TX_CLK clock divisor. (R/W)

HP_SYS_CLKRST_I2S0_TX_DIV_Z Configures the coefficient z for I2S0_TX_CLK clock divisor. (R/W)

HP_SYS_CLKRST_I2S0_TX_DIV_YN1 Configures the coefficient yn1 for I2S0_TX_CLK clock divisor. (R/W)

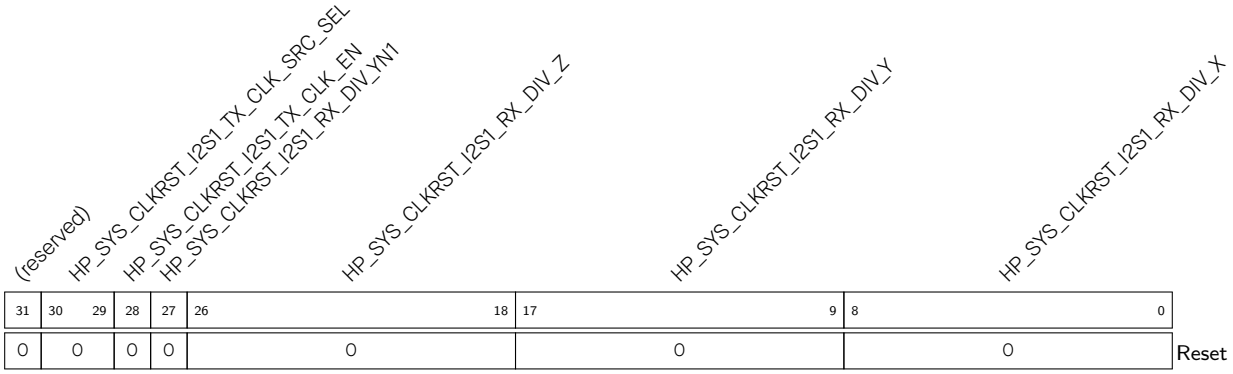
HP_SYS_CLKRST_I2S0_MST_CLK_SEL Configures the clock source for the output clock PAD_I2S0_MST_CLK.
0: I2S0_RX_CLK
1: I2S0_TX_CLK
(R/W)

HP_SYS_CLKRST_I2S1_RX_CLK_EN Configures whether to enable I2S1_RX_CLK.
0: Disable
1: Enable
(R/W)

HP_SYS_CLKRST_I2S1_RX_CLK_SRC_SEL Configures the clock source for I2S1_RX_CLK.
0: XTAL_CLK
1: APLL_CLK
2: PAD_I2S1_MCLK
3: Invalid
(R/W)

HP_SYS_CLKRST_I2S1_RX_DIV_N Configures the integer part of the clock divisor for I2S1_RX_CLK. (R/W)

Register 11.22. HP_SYS_CLKRST_PERI_CLK_CTRL15_REG (0x0054)



HP_SYS_CLKRST_I2S1_RX_DIV_X Configures the coefficient x for I2S1_RX_CLK clock divisor. (R/W)

HP_SYS_CLKRST_I2S1_RX_DIV_Y Configures the coefficient y for I2S1_RX_CLK clock divisor. (R/W)

HP_SYS_CLKRST_I2S1_RX_DIV_Z Configures the coefficient z for I2S1_RX_CLK clock divisor. (R/W)

HP_SYS_CLKRST_I2S1_RX_DIV_YN1 Configures the coefficient yn1 for I2S1_RX_CLK clock divisor. (R/W)

HP_SYS_CLKRST_I2S1_TX_CLK_EN Configures whether to enable PI2S1_TX_CLK.

0: Disable

1: Enable

(R/W)

HP_SYS_CLKRST_I2S1_TX_CLK_SRC_SEL Configures the clock source for I2S1_TX_CLK.

0: XTAL_CLK

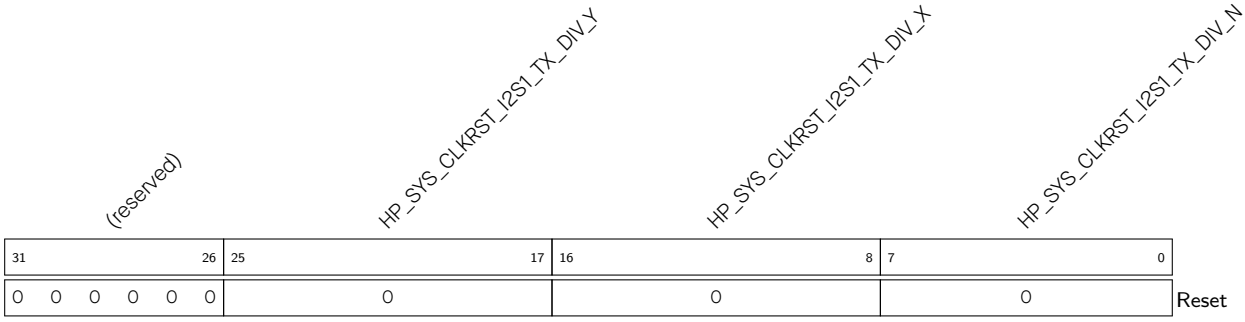
1: APLL_CLK

2: PAD_I2S1_MCLK

3: Invalid

(R/W)

Register 11.23. HP_SYS_CLKRST_PERI_CLK_CTRL16_REG (0x0058)

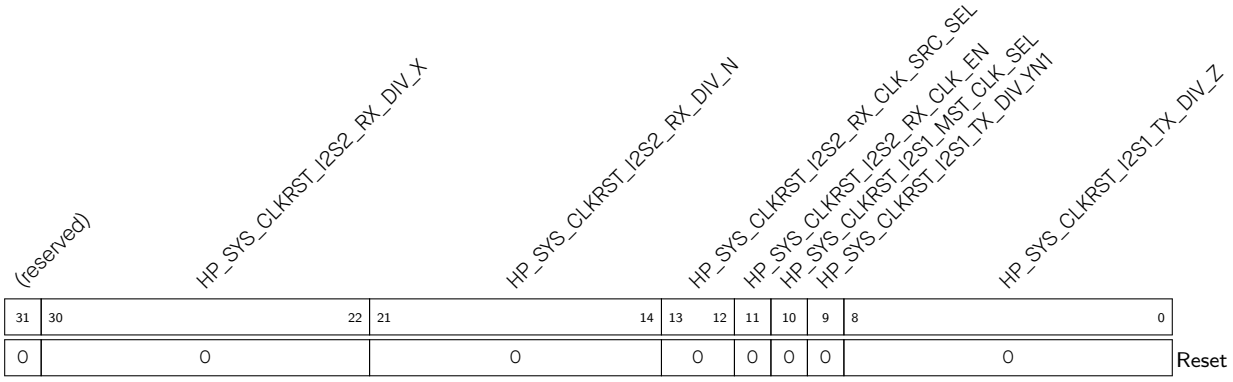


HP_SYS_CLKRST_I2S1_TX_DIV_N Configures the integer part of the clock divisor for I2S1_TX_CLK. (R/W)

HP_SYS_CLKRST_I2S1_TX_DIV_X Configures the coefficient x for I2S1_TX_CLK clock divisor. (R/W)

HP_SYS_CLKRST_I2S1_TX_DIV_Y Configures the coefficient y for I2S1_TX_CLK clock divisor. (R/W)

Register 11.24. HP_SYS_CLKRST_PERI_CLK_CTRL17_REG (0x005C)



HP_SYS_CLKRST_I2S1_TX_DIV_Z Configures the coefficient z for I2S1_TX_CLK clock divisor. (R/W)

HP_SYS_CLKRST_I2S1_TX_DIV_YN1 Configures the coefficient yn1 for I2S1_TX_CLK clock divisor. (R/W)

HP_SYS_CLKRST_I2S1_MST_CLK_SEL Configures the clock source for the output clock PAD_I2S1_MST_CLK.
0: I2S1_RX_CLK
1: I2S1_TX_CLK
(R/W)

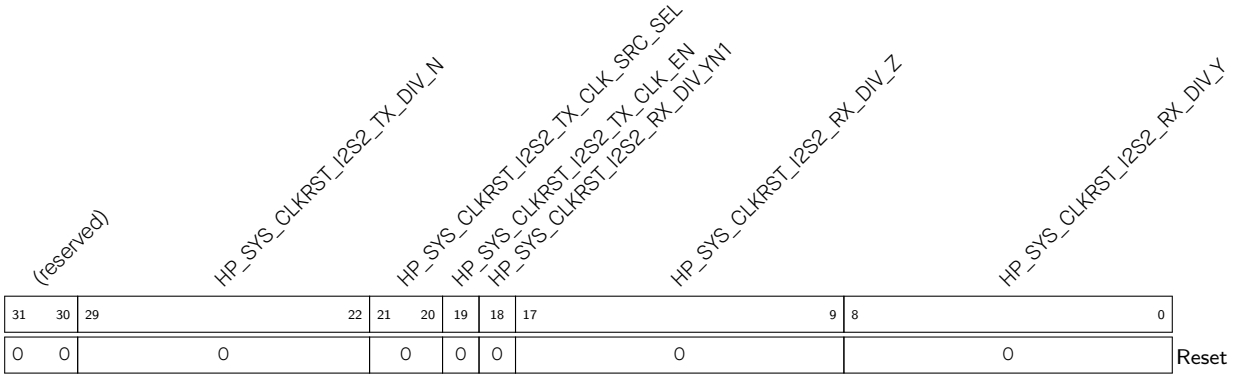
HP_SYS_CLKRST_I2S2_RX_CLK_EN Configures whether to enable I2S2_RX_CLK.
0: Disable
1: Enable
(R/W)

HP_SYS_CLKRST_I2S2_RX_CLK_SRC_SEL Configures the clock source for I2S2_RX_CLK.
0: XTAL_CLK
1: APLL_CLK
2: PAD_I2S2_MCLK
3: Invalid
(R/W)

HP_SYS_CLKRST_I2S2_RX_DIV_N Configures the integer part of the clock divisor for I2S2_RX_CLK. (R/W)

HP_SYS_CLKRST_I2S2_RX_DIV_X Configures the coefficient x for I2S2_RX_CLK clock divisor. (R/W)

Register 11.25. HP_SYS_CLKRST_PERI_CLK_CTRL18_REG (0x0060)



HP_SYS_CLKRST_I2S2_RX_DIV_Y Configures the coefficient y for I2S2_RX_CLK clock divisor. (R/W)

HP_SYS_CLKRST_I2S2_RX_DIV_Z Configures the coefficient z for I2S2_RX_CLK clock divisor. (R/W)

HP_SYS_CLKRST_I2S2_RX_DIV_YN1 Configures the coefficient yn1 for I2S2_RX_CLK clock divisor. (R/W)

HP_SYS_CLKRST_I2S2_TX_CLK_EN Configures whether to enable I2S2_TX_CLK.

0: Disable

1: Enable

(R/W)

HP_SYS_CLKRST_I2S2_TX_CLK_SRC_SEL Configures the clock source for I2S2_TX_CLK.

0: XTAL_CLK

1: APLL_CLK

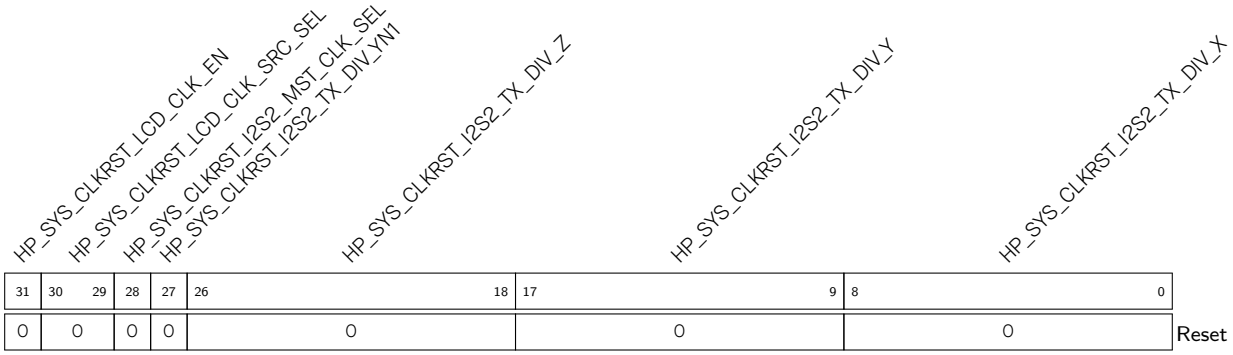
2: PAD_I2S2_MCLK

3: Invalid

(R/W)

HP_SYS_CLKRST_I2S2_TX_DIV_N Configures the integer part for I2S2_TX_CLK clock divisor. (R/W)

Register 11.26. HP_SYS_CLKRST_PERI_CLK_CTRL19_REG (0x0064)



HP_SYS_CLKRST_I2S2_TX_DIV_X Configures the coefficient x of I2S2_TX_CLK clock divisor. (R/W)

HP_SYS_CLKRST_I2S2_TX_DIV_Y Configures the coefficient y of I2S2_TX_CLK clock divisor. (R/W)

HP_SYS_CLKRST_I2S2_TX_DIV_Z Configures the coefficient z of I2S2_TX_CLK clock divisor. (R/W)

HP_SYS_CLKRST_I2S2_TX_DIV_YN1 Configures the coefficient yn1 of I2S2_TX_CLK clock divisor. (R/W)

HP_SYS_CLKRST_I2S2_MST_CLK_SEL Configures the clock source for output clock PAD_I2S2_MST_CLK.

0: I2S2_RX_CLK

1: I2S2_TX_CLK

(R/W)

HP_SYS_CLKRST_LCD_CLK_SRC_SEL Configures the clock source for LCD_CLK.

0: XTAL_CLK

1: PLL_F160M_CLK

2: APLL_CLK

3: Invalid

(R/W)

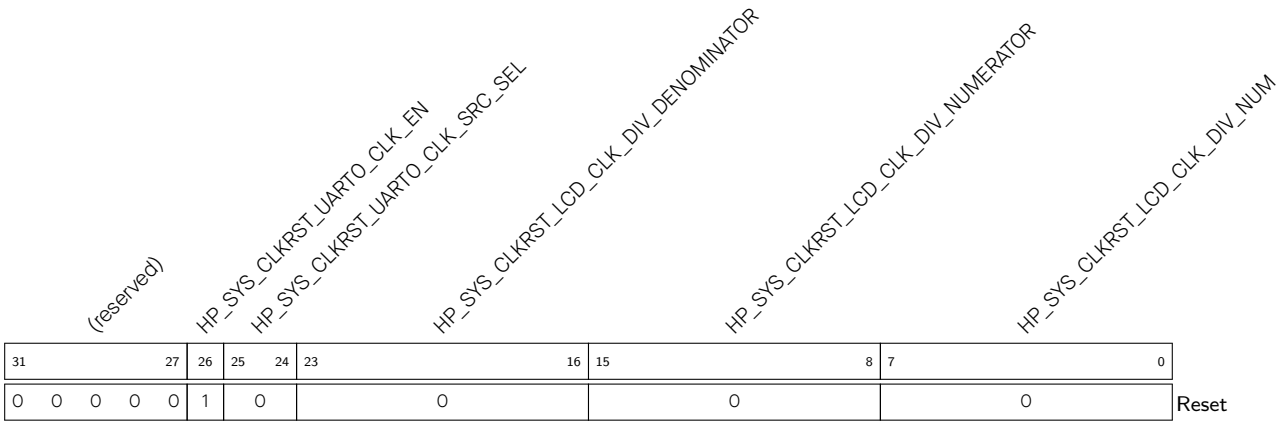
HP_SYS_CLKRST_LCD_CLK_EN Configures whether to enable LCD_CLK.

0: Disable

1: Enable

(R/W)

Register 11.27. HP_SYS_CLKRST_PERI_CLK_CTRL110_REG (0x0068)



HP_SYS_CLKRST_LCD_CLK_DIV_NUM Configures the integer part of the LCD_CLK clock divisor. (R/W)

HP_SYS_CLKRST_LCD_CLK_DIV_NUMERATOR Configures the numerator of the divisor's fractional part for LCD_CLK. (R/W)

HP_SYS_CLKRST_LCD_CLK_DIV_DENOMINATOR Configures the denominator of the divisor's fractional part for LCD_CLK. (R/W)

HP_SYS_CLKRST_UART0_CLK_SRC_SEL Configures the clock source for UART0_CLK.

0: XTAL_CLK

1: RC_FAST_CLK

2: PLL_F80M_CLK

3: Invalid

(R/W)

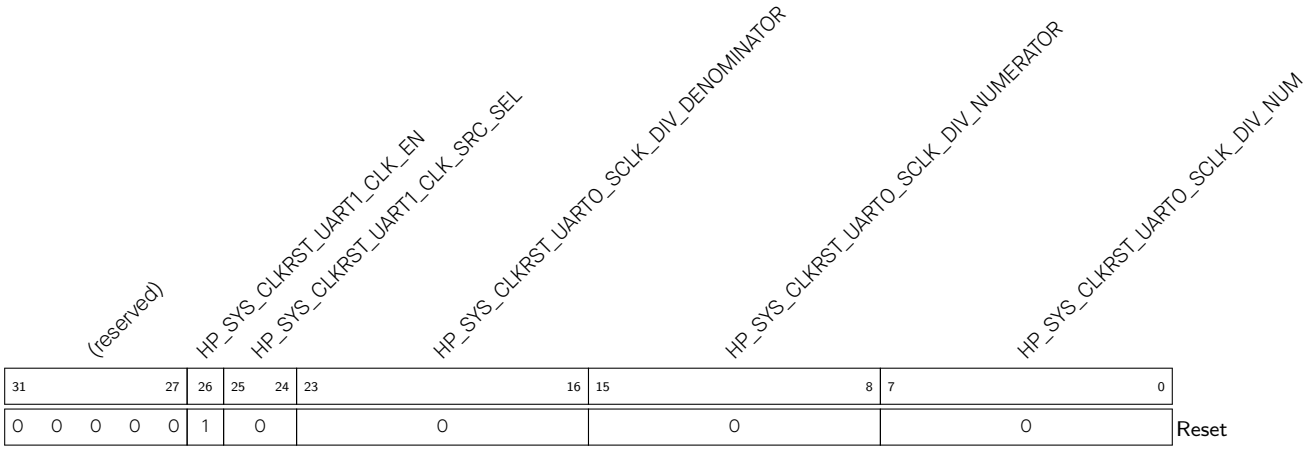
HP_SYS_CLKRST_UART0_CLK_EN Configures whether to enable UART0_CLK.

0: Disable

1: Enable

(R/W)

Register 11.28. HP_SYS_CLKRST_PERI_CLK_CTRL111_REG (0x006C)



HP_SYS_CLKRST_UART0_SCLK_DIV_NUM Configures the integer part of the UART0_SCLK clock divisor. (R/W)

HP_SYS_CLKRST_UART0_SCLK_DIV_NUMERATOR Configures the numerator of the divisor's fractional part for UART0_SCLK. (R/W)

HP_SYS_CLKRST_UART0_SCLK_DIV_DENOMINATOR Configures the denominator of the divisor's fractional part for UART0_SCLK. (R/W)

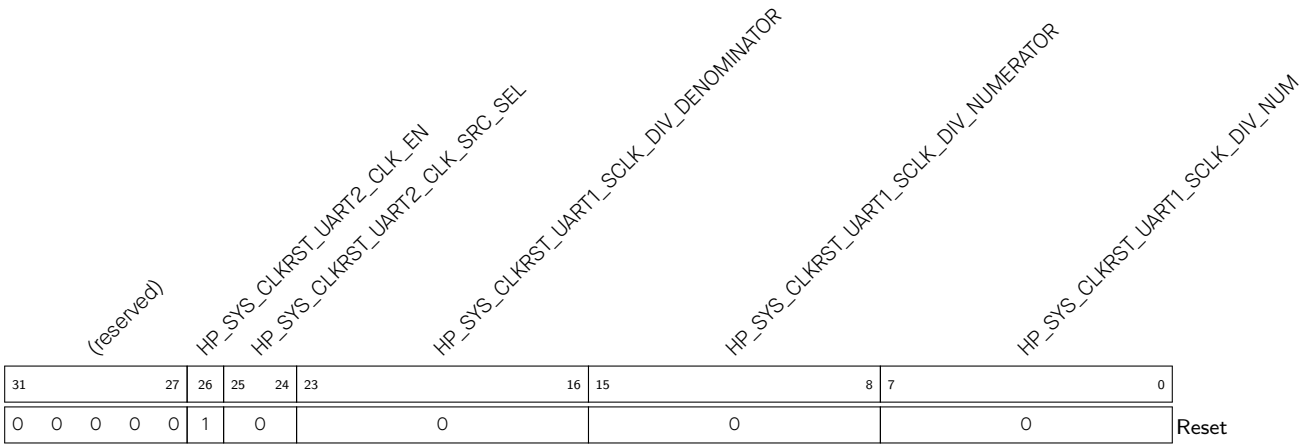
HP_SYS_CLKRST_UART1_CLK_SRC_SEL Configures the clock source for UART1_CLK.

0: XTAL_CLK
1: RC_FAST_CLK
2: PLL_F80M_CLK
3: Invalid
(R/W)

HP_SYS_CLKRST_UART1_CLK_EN Configures whether to enable UART1_CLK.

0: Disable
1: Enable
(R/W)

Register 11.29. HP_SYS_CLKRST_PERI_CLK_CTRL112_REG (0x0070)



HP_SYS_CLKRST_UART1_SCLK_DIV_NUM Configures the integer part of the UART1_SCLK clock divisor. (R/W)

HP_SYS_CLKRST_UART1_SCLK_DIV_NUMERATOR Configures the numerator of the divisor's fractional part for UART1_SCLK. (R/W)

HP_SYS_CLKRST_UART1_SCLK_DIV_DENOMINATOR Configures the denominator of the divisor's fractional part for UART1_SCLK. (R/W)

HP_SYS_CLKRST_UART2_CLK_SRC_SEL Configures the clock source for UART2_CLK.

0: XTAL_CLK

1: RC_FAST_CLK

2: PLL_F80M_CLK

3: Invalid

(R/W)

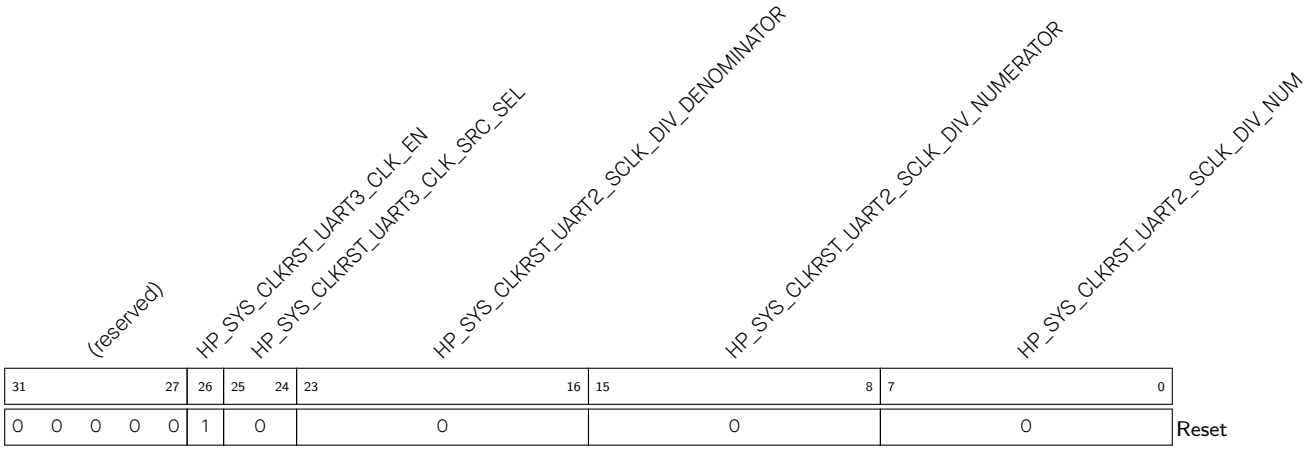
HP_SYS_CLKRST_UART2_CLK_EN Configures whether to enable the UART2_CLK clock.

0: Disable

1: Enable

(R/W)

Register 11.30. HP_SYS_CLKRST_PERI_CLK_CTRL113_REG (0x0074)



HP_SYS_CLKRST_UART2_SCLK_DIV_NUM Configures the integer part of the UART2_SCLK clock divisor. (R/W)

HP_SYS_CLKRST_UART2_SCLK_DIV_NUMERATOR Configures the numerator of the fractional part of the UART2_SCLK. (R/W)

HP_SYS_CLKRST_UART2_SCLK_DIV_DENOMINATOR Configures the denominator of the fractional part of the UART2_SCLK. (R/W)

HP_SYS_CLKRST_UART3_CLK_SRC_SEL Configures the clock source for UART3_CLK.

0: XTAL_CLK

1: RC_FAST_CLK

2: PLL_F80M_CLK

3: Invalid

(R/W)

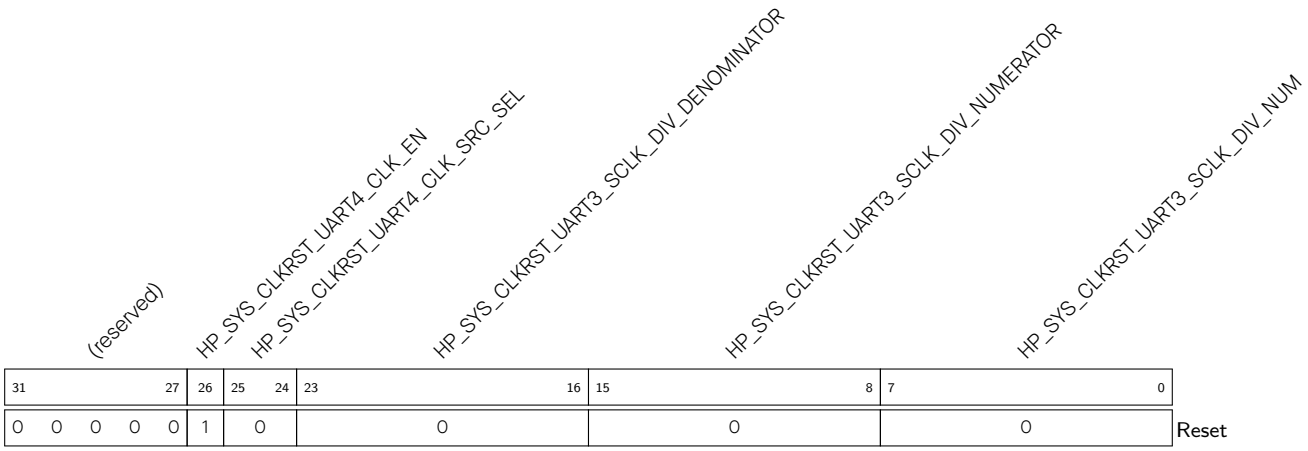
HP_SYS_CLKRST_UART3_CLK_EN Configures whether to enable the UART3_CLK clock.

0: Disable

1: Enable

(R/W)

Register 11.31. HP_SYS_CLKRST_PERI_CLK_CTRL114_REG (0x0078)



HP_SYS_CLKRST_UART3_SCLK_DIV_NUM Configures the integer part of the UART3_SCLK clock divisor. (R/W)

HP_SYS_CLKRST_UART3_SCLK_DIV_NUMERATOR Configures the numerator of the fractional part of the UART3_SCLK. (R/W)

HP_SYS_CLKRST_UART3_SCLK_DIV_DENOMINATOR Configures the denominator of the fractional part of the UART3_SCLK. (R/W)

HP_SYS_CLKRST_UART4_CLK_SRC_SEL Configures the clock source for UART4_CLK.

0: XTAL_CLK

1: RC_FAST_CLK

2: PLL_F80M_CLK

3: Invalid

(R/W)

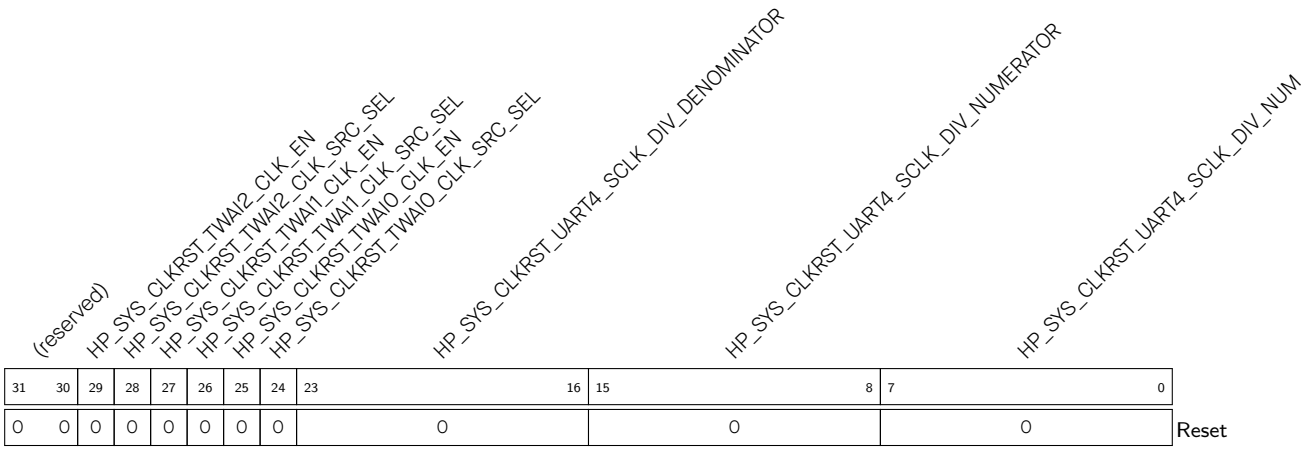
HP_SYS_CLKRST_UART4_CLK_EN Configures whether to enable the UART4_CLK clock.

0: Disable

1: Enable

(R/W)

Register 11.32. HP_SYS_CLKRST_PERI_CLK_CTRL115_REG (0x007C)



HP_SYS_CLKRST_UART4_SCLK_DIV_NUM Configures the integer part of the UART4_SCLK clock divisor. (R/W)

HP_SYS_CLKRST_UART4_SCLK_DIV_NUMERATOR Configures the numerator of the fractional part of the UART4_SCLK clock divisor. (R/W)

HP_SYS_CLKRST_UART4_SCLK_DIV_DENOMINATOR Configures the denominator of the fractional part of the UART4_SCLK clock divisor. (R/W)

HP_SYS_CLKRST_TWAI0_CLK_SRC_SEL Configures the clock source for TWAI0_CLK.

0: XTAL_CLK

1: RC_FAST_CLK

(R/W)

HP_SYS_CLKRST_TWAI0_CLK_EN Configures whether to enable TWAI0_CLK.

0: Disable

1: Enable

(R/W)

HP_SYS_CLKRST_TWAI1_CLK_SRC_SEL Configures the clock source for TWAI1_CLK.

0: XTAL_CLK

1: RC_FAST_CLK

(R/W)

HP_SYS_CLKRST_TWAI1_CLK_EN Configures whether to enable TWAI1_CLK.

0: Disable

1: Enable

(R/W)

HP_SYS_CLKRST_TWAI2_CLK_SRC_SEL Configures the clock source for TWAI2_CLK.

0: XTAL_CLK

1: RC_FAST_CLK

(R/W)

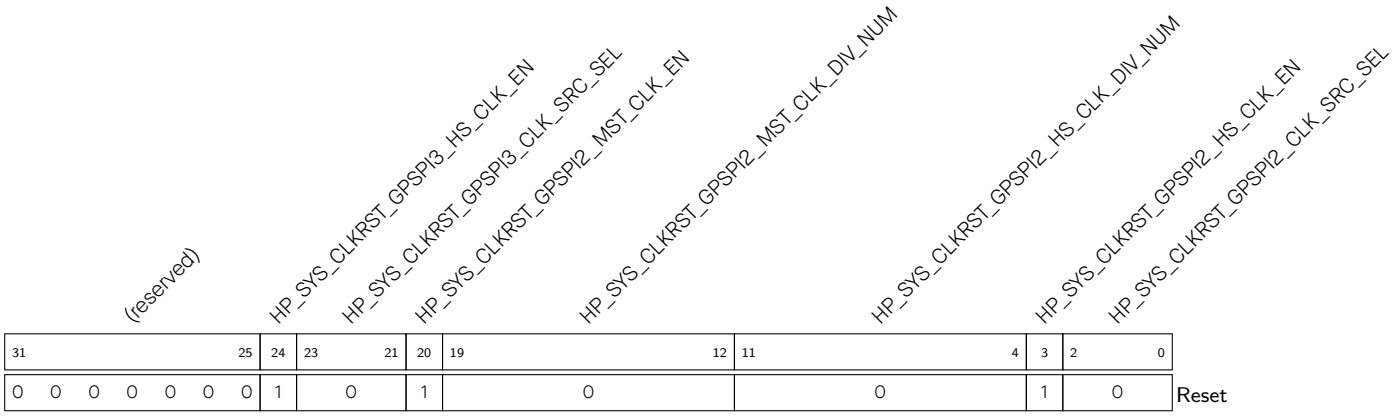
HP_SYS_CLKRST_TWAI2_CLK_EN Configures whether to enable TWAI2_CLK.

0: Disable

1: Enable

(R/W)

Register 11.33. HP_SYS_CLKRST_PERI_CLK_CTRL116_REG (0x0080)



HP_SYS_CLKRST_GPSPI2_CLK_SRC_SEL Configures the clock source for GPSPI2_CLK.

0: XTAL_CLK

1: RC_FAST_CLK

2: SDIO_PLL0_CLK

3: APLL_CLK

4: SPLL_CLK (480 MHz)

5-7: Invalid

(R/W)

HP_SYS_CLKRST_GPSPI2_HS_CLK_EN Configures whether to enable GPSPI2_HS_CLK.

0: Disable

1: Enable

(R/W)

HP_SYS_CLKRST_GPSPI2_HS_CLK_DIV_NUM Configures the integer part of the clock divisor of GPSPI2_HS_CLK. (R/W)

HP_SYS_CLKRST_GPSPI2_MST_CLK_DIV_NUM Configures the integer part of the clock divisor of GPSPI2_MST_CLK. (R/W)

HP_SYS_CLKRST_GPSPI2_MST_CLK_EN Configures whether to enable GPSPI2_MST_CLK.

0: Disable

1: Enable

(R/W)

HP_SYS_CLKRST_GPSPI3_CLK_SRC_SEL Configures the clock source for GPSPI3_CLK.

0: XTAL_CLK

1: RC_FAST_CLK

2: SDIO_PLL0_CLK

3: APLL_CLK

4: SPLL_CLK (480 MHz)

5-7: Invalid

(R/W)

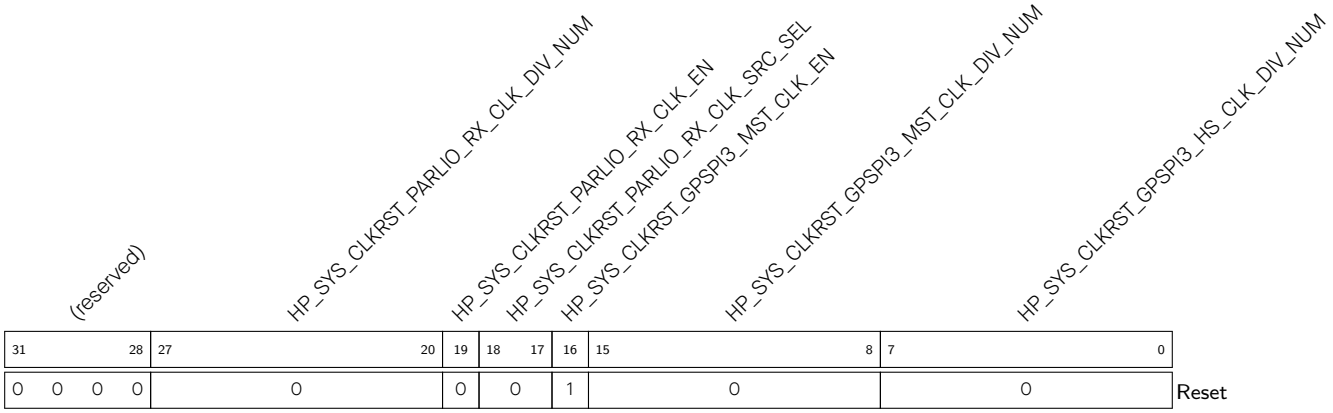
HP_SYS_CLKRST_GPSPI3_HS_CLK_EN Configures whether to enable GPSPI3_HS_CLK.

0: Disable

1: Enable

(R/W)

Register 11.34. HP_SYS_CLKRST_PERI_CLK_CTRL117_REG (0x0084)



HP_SYS_CLKRST_GPSPI3_HS_CLK_DIV_NUM Configures the integer part of the clock divisor for GPSPI3_HS_CLK. (R/W)

HP_SYS_CLKRST_GPSPI3_MST_CLK_DIV_NUM Configures the integer part of the clock divisor for GPSPI3_MST_CLK. (R/W)

HP_SYS_CLKRST_GPSPI3_MST_CLK_EN Configures whether to enable GPSPI3_MST_CLK.

0: Disable

1: Enable

(R/W)

HP_SYS_CLKRST_PARLIO_RX_CLK_SRC_SEL Configures the clock source for PARLIO_RX_CLK.

0: XTAL_CLK

1: RC_FAST_CLK

2: PLL_F160M_CLK

3: PAD_PARLIO_RX_CLK

(R/W)

HP_SYS_CLKRST_PARLIO_RX_CLK_EN Configures whether to enable PARLIO_RX_CLK.

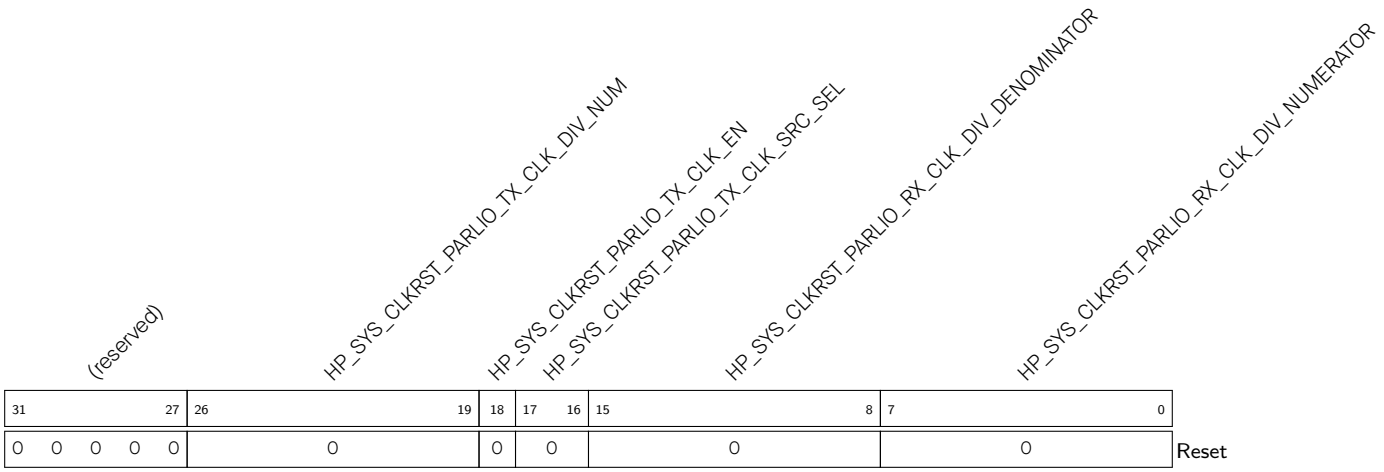
0: Disable

1: Enable

(R/W)

HP_SYS_CLKRST_PARLIO_RX_CLK_DIV_NUM Configures the integer part of the clock divisor for PARLIO_RX_CLK. (R/W)

Register 11.35. HP_SYS_CLKRST_PERI_CLK_CTRL118_REG (0x0088)



HP_SYS_CLKRST_PARLIO_RX_CLK_DIV_NUMERATOR Configures the numerator of the divisor's fractional part for clock divisor for PARLIO_RX_CLK. (R/W)

HP_SYS_CLKRST_PARLIO_RX_CLK_DIV_DENOMINATOR Configures the denominator of the divisor's fractional part for clock divisor for PARLIO_RX_CLK. (R/W)

HP_SYS_CLKRST_PARLIO_TX_CLK_SRC_SEL Configures the clock source for PARLIO_TX_CLK.

0: XTAL_CLK

1: RC_FAST_CLK

2: PLL_F160M_CLK

3: PAD_PARLIO_TX_CLK

(R/W)

HP_SYS_CLKRST_PARLIO_TX_CLK_EN Configures whether to enable PARLIO_TX_CLK.

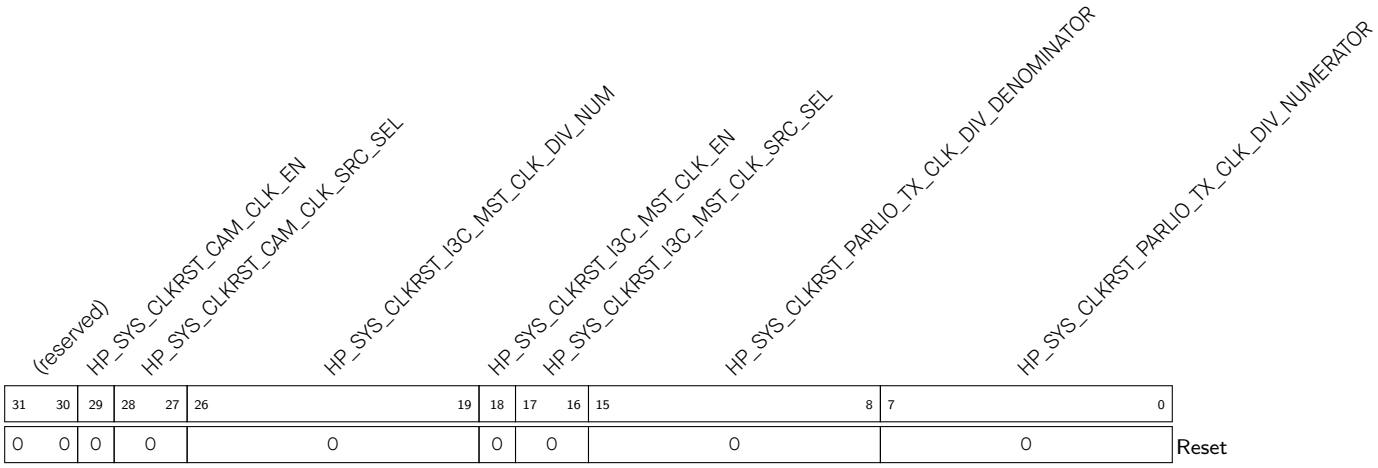
0: Disable

1: Enable

(R/W)

HP_SYS_CLKRST_PARLIO_TX_CLK_DIV_NUM Configures the integer part of the clock divisor for PARLIO_TX_CLK. (R/W)

Register 11.36. HP_SYS_CLKRST_PERI_CLK_CTRL119_REG (0x008C)



HP_SYS_CLKRST_PARLIO_TX_CLK_DIV_NUMERATOR Configures the numerator of the divisor's fractional part for PARLIO_TX_CLK. (R/W)

HP_SYS_CLKRST_PARLIO_TX_CLK_DIV_DENOMINATOR Configures the denominator of the divisor's fractional part for PARLIO_TX_CLK. (R/W)

HP_SYS_CLKRST_I3C_MST_CLK_SRC_SEL Configures the clock source for I3C_MST_CLK.

0: XTAL_CLK
1: PLL_F160M_CLK
2: PLL_F120M_CLK
3: Invalid
(R/W)

HP_SYS_CLKRST_I3C_MST_CLK_EN Configures whether to enable I3C_MST_CLK.

0: Disable
1: Enable
(R/W)

HP_SYS_CLKRST_I3C_MST_CLK_DIV_NUM Configures the integer part of the I3C_MST_CLK clock divisor. (R/W)

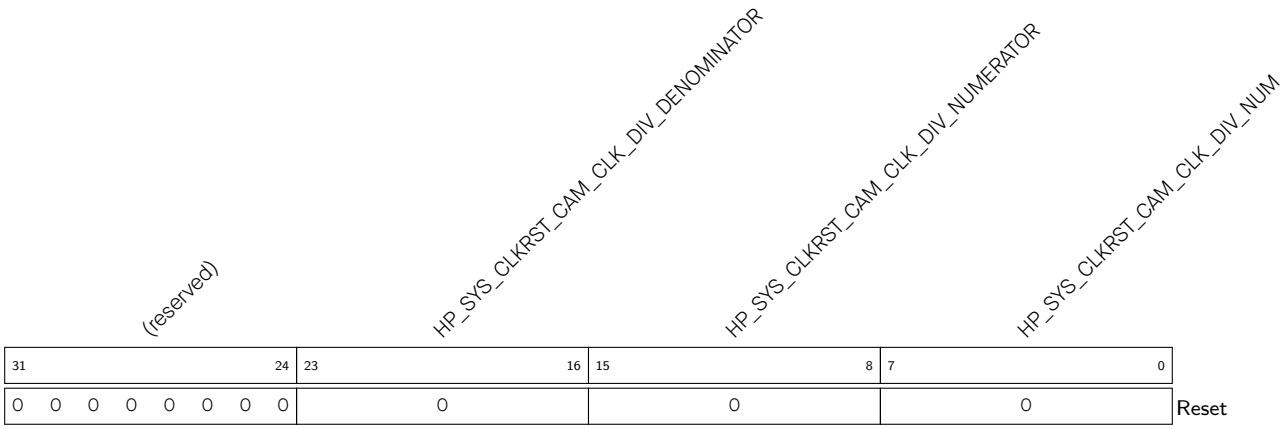
HP_SYS_CLKRST_CAM_CLK_SRC_SEL Configures the clock source for the output clock CAM_CLK.

0: XTAL_CLK
1: PLL_F160M_CLK
2: APLL_CLK
3: Invalid
(R/W)

HP_SYS_CLKRST_CAM_CLK_EN Configures whether to enable the output clock CAM_CLK.

0: Disable
1: Enable
(R/W)

Register 11.37. HP_SYS_CLKRST_PERI_CLK_CTRL120_REG (0x0090)

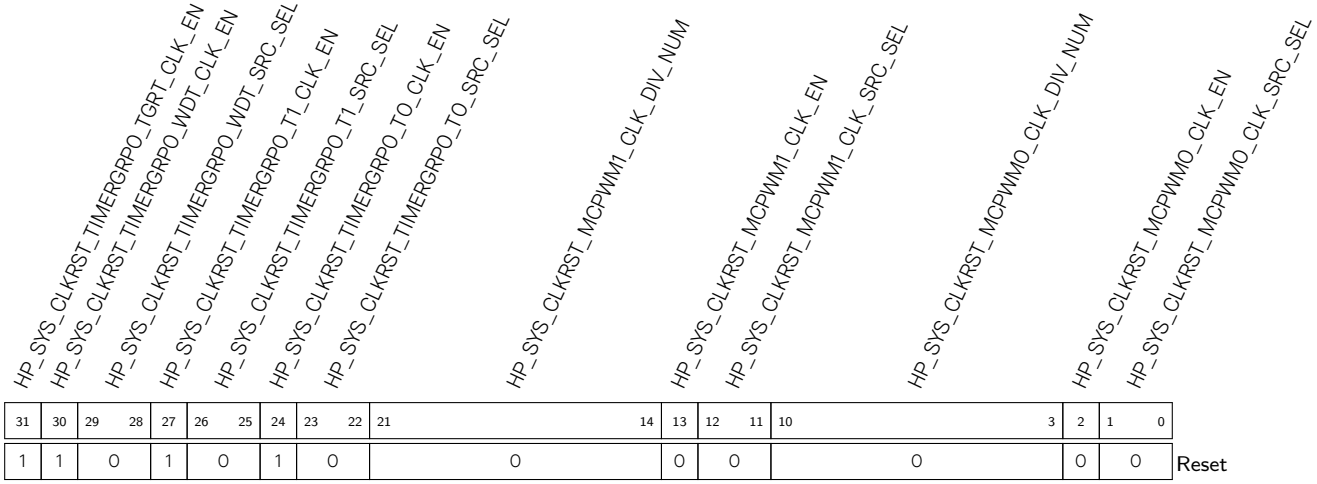


HP_SYS_CLKRST_CAM_CLK_DIV_NUM Configures the integer part of the CAM_CLK clock divisor. (R/W)

HP_SYS_CLKRST_CAM_CLK_DIV_NUMERATOR Configures the numerator of the divisor's fractional part for CAM_CLK. (R/W)

HP_SYS_CLKRST_CAM_CLK_DIV_DENOMINATOR Configures the denominator of the divisor's fractional part for CAM_CLK. (R/W)

Register 11.38. HP_SYS_CLKRST_PERI_CLK_CTRL20_REG (0x0094)



HP_SYS_CLKRST_MCPWM0_CLK_SRC_SEL Configures the clock source for MCPWM0_CLK.

0: XTAL_CLK

1: RC_FAST_CLK

2: PLL_F160_CLK

3: Invalid

(R/W)

HP_SYS_CLKRST_MCPWM0_CLK_EN Configures whether to enable MCPWM0_CLK.

0: Disable

1: Enable

(R/W)

HP_SYS_CLKRST_MCPWM0_CLK_DIV_NUM Configures the clock divisor of MCPWM0_CLK. (R/W)

HP_SYS_CLKRST_MCPWM1_CLK_SRC_SEL Configures the clock source for MCPWM1_CLK.

0: XTAL_CLK

1: RC_FAST_CLK

2: PLL_F160_CLK

3: Invalid

(R/W)

HP_SYS_CLKRST_MCPWM1_CLK_EN Configures whether to enable MCPWM1_CLK.

0: Disable

1: Enable

(R/W)

HP_SYS_CLKRST_MCPWM1_CLK_DIV_NUM Configures the clock divisor of MCPWM1_CLK. (R/W)

HP_SYS_CLKRST_TIMERGRP0_T0_SRC_SEL Configures the clock source for TIMERGRP0_T0_CLK.

0: XTAL_CLK

1: RC_FAST_CLK

2: PLL_F80M_CLK

3: Invalid

(R/W)

Register 11.38. HP_SYS_CLKRST_PERI_CLK_CTRL20_REG (0x0094)

Continued from the previous page...

HP_SYS_CLKRST_TIMERGRP0_TO_CLK_EN Configures whether to enable TIMERGRP0_TO_CLK.
0: Disable
1: Enable
(R/W)

HP_SYS_CLKRST_TIMERGRP0_T1_SRC_SEL Configures the clock source for TIMERGRP0_T1_CLK.
0: XTAL_CLK
1: RC_FAST_CLK
2: PLL_F80M_CLK
3: Invalid
(R/W)

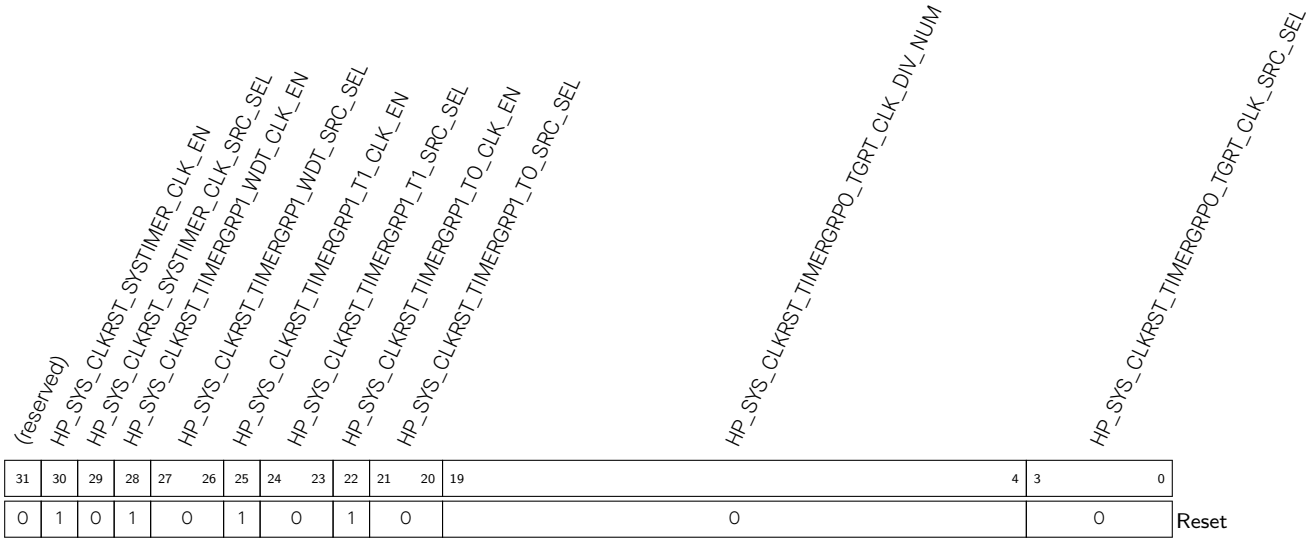
HP_SYS_CLKRST_TIMERGRP0_T1_CLK_EN Configures whether to enable TIMERGRP0_T1_CLK.
0: Disable
1: Enable
(R/W)

HP_SYS_CLKRST_TIMERGRP0_WDT_SRC_SEL Configures the clock source for TIMERGRP0_WDT_CLK.
0: XTAL_CLK
1: RC_FAST_CLK
2: PLL_F80M_CLK
3: Invalid
(R/W)

HP_SYS_CLKRST_TIMERGRP0_WDT_CLK_EN Configures whether to enable TIMERGRP0_WDT_CLK.
0: Disable
1: Enable
(R/W)

HP_SYS_CLKRST_TIMERGRP0_TGRT_CLK_EN Configures whether to enable TIMERGRP0_TGRT_CLK.
0: Disable
1: Enable
(R/W)

Register 11.39. HP_SYS_CLKRST_PERI_CLK_CTRL21_REG (0x0098)



HP_SYS_CLKRST_TIMERGRP0_TGRT_CLK_SRC_SEL Configures the clock source for TIMERGRP0_TGRT_CLK.

0: MPLL_CLK (500 MHz)
1: SPLL_CLK (480 MHz)
2: CPLL_CLK (360 MHz)
3: APLL_CLK
4: SDIO_PLL0_CLK
5: SDIO_PLL1_CLK
6: SDIO_PLL2_CLK
7: RC_FAST_CLK
8: RC_SLOW_CLK
9: Invalid
10: XTAL32K_CLK
11: PLL_LP_CLK
12-15: Clock tie 0
(R/W)

HP_SYS_CLKRST_TIMERGRP0_TGRT_CLK_DIV_NUM Configures the clock divisor of TIMERGRP0_TGRT_CLK. (R/W)

HP_SYS_CLKRST_TIMERGRP1_T0_SRC_SEL Configures the clock source for TIMERGRP1_T0_CLK.

0: XTAL_CLK
1: RC_FAST_CLK
2: PLL_F80M_CLK
3: Invalid
(R/W)

HP_SYS_CLKRST_TIMERGRP1_T0_CLK_EN Configures whether to enable TIMERGRP1_T0_CLK.

0: Disable
1: Enable
(R/W)

Continued on the next page...

Register 11.39. HP_SYS_CLKRST_PERI_CLK_CTRL21_REG (0x0098)

Continued from the previous page...

HP_SYS_CLKRST_TIMERGRP1_T1_SRC_SEL Configures the clock source for TIMERGRP1_T1_CLK.

0: XTAL_CLK

1: RC_FAST_CLK

2: PLL_F80M_CLK

3: Invalid

(R/W)

HP_SYS_CLKRST_TIMERGRP1_T1_CLK_EN Configures whether to enable TIMERGRP1_T1_CLK.

0: Disable

1: Enable

(R/W)

HP_SYS_CLKRST_TIMERGRP1_WDT_SRC_SEL Configures the clock source for TIMERGRP1_WDT_CLK.

0: XTAL_CLK

1: RC_FAST_CLK

2: PLL_F80M_CLK

3: Invalid

(R/W)

HP_SYS_CLKRST_TIMERGRP1_WDT_CLK_EN Configures whether to enable TIMERGRP1_WDT_CLK.

0: Disable

1: Enable

(R/W)

HP_SYS_CLKRST_SYSTIMER_CLK_SRC_SEL Configures the clock source for SYSTIMER_CLK.

0: XTAL_CLK

1: RC_FAST_CLK

(R/W)

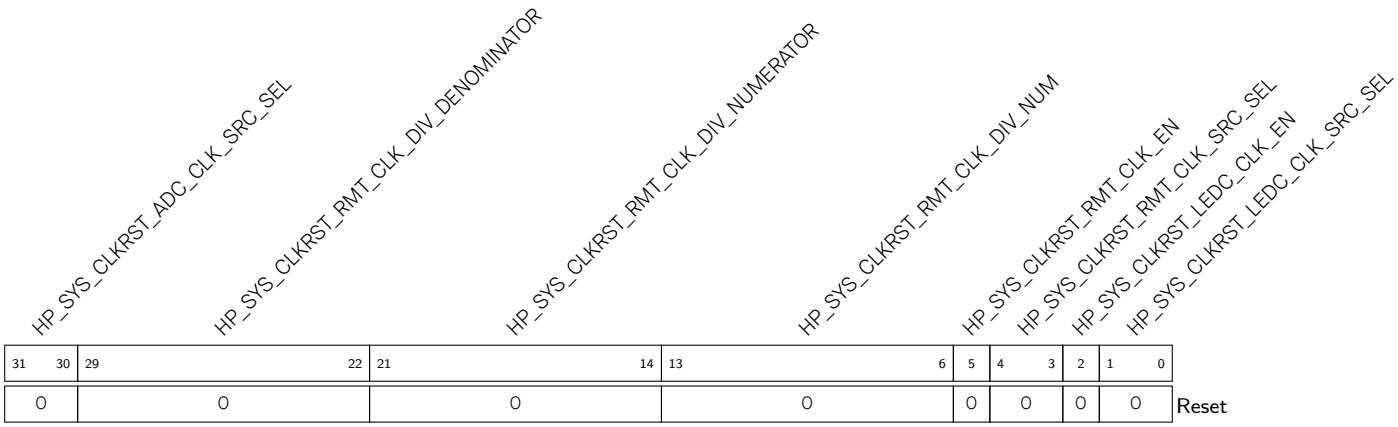
HP_SYS_CLKRST_SYSTIMER_CLK_EN Configures whether to enable SYSTIMER_CLK.

0: Disable

1: Enable

(R/W)

Register 11.40. HP_SYS_CLKRST_PERI_CLK_CTRL22_REG (0x009C)



HP_SYS_CLKRST_LEDC_CLK_SRC_SEL Configures the clock source for LEDC_CLK.

0: XTAL_CLK

1: RC_FAST_CLK

2: PLL_F80M_CLK

3: Invalid

(R/W)

HP_SYS_CLKRST_LEDC_CLK_EN Configures whether to enable LEDC_CLK.

0: Disable

1: Enable

(R/W)

HP_SYS_CLKRST_RMT_CLK_SRC_SEL Configures the clock source for RMT_CLK.

0: XTAL_CLK

1: RC_FAST_CLK

2: PLL_F80M_CLK

3: Invalid

(R/W)

HP_SYS_CLKRST_RMT_CLK_EN Configures whether to enable RMT_CLK.

0: Disable

1: Enable

(R/W)

HP_SYS_CLKRST_RMT_CLK_DIV_NUM Configures the integer part of the RMT_CLK clock divisor. (R/W)

HP_SYS_CLKRST_RMT_CLK_DIV_NUMERATOR Configures the numerator of the divisor's fractional part for RMT_CLK. (R/W)

HP_SYS_CLKRST_RMT_CLK_DIV_DENOMINATOR Configures the denominator of the divisor's fractional part for RMT_CLK. (R/W)

Continued on the next page...

Register 11.40. HP_SYS_CLKRST_PERI_CLK_CTRL22_REG (0x009C)

Continued from the previous page...

HP_SYS_CLKRST_ADC_CLK_SRC_SEL Configures the clock source for HPADC_CLK.

0: XTAL_CLK

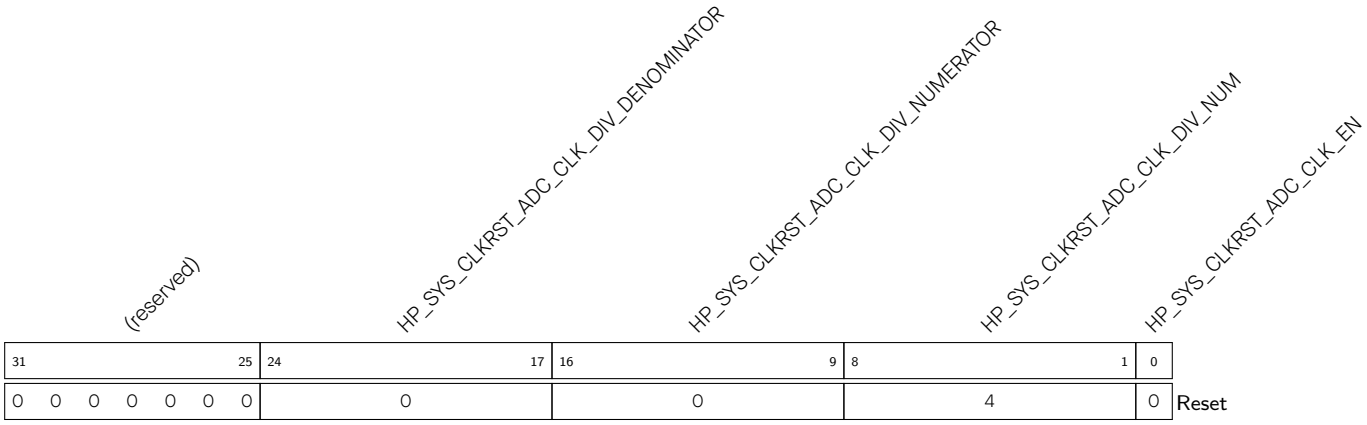
1: RC_FAST_CLK

2: PLL_F80M_CLK

3: Invalid

(R/W)

Register 11.41. HP_SYS_CLKRST_PERI_CLK_CTRL23_REG (0x00A0)



HP_SYS_CLKRST_ADC_CLK_EN Configures whether to enable HPADC_CLK.

0: Disable

1: Enable

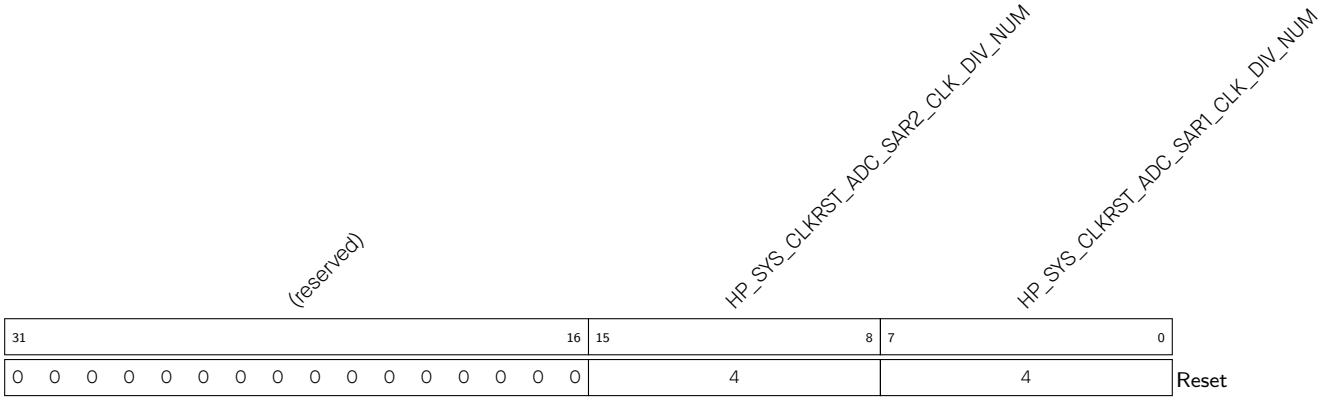
(R/W)

HP_SYS_CLKRST_ADC_CLK_DIV_NUM Configures the integer part of the HPADC_CLK clock divisor. (R/W)

HP_SYS_CLKRST_ADC_CLK_DIV_NUMERATOR Configures the numerator of the divisor's fractional part for HPADC_CLK. (R/W)

HP_SYS_CLKRST_ADC_CLK_DIV_DENOMINATOR Configures the denominator of the divisor's fractional part for HPADC_CLK. (R/W)

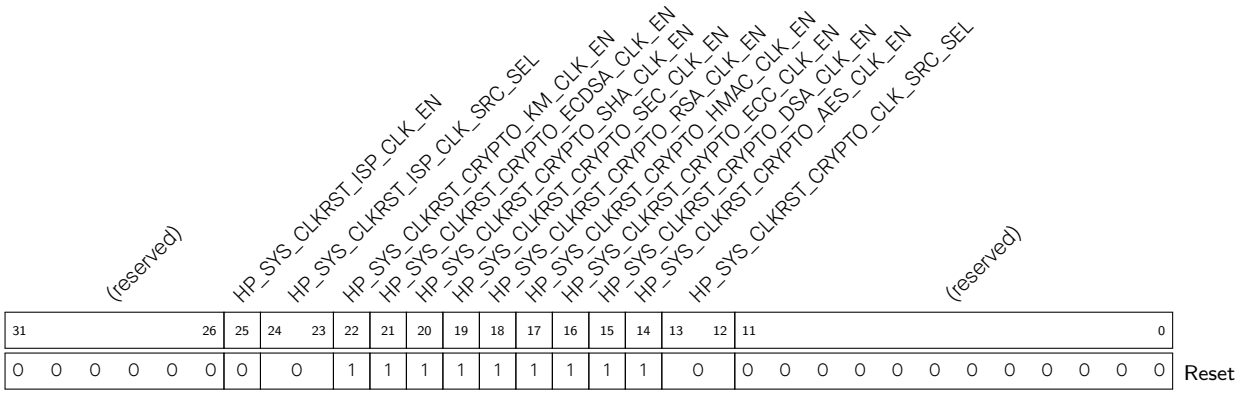
Register 11.42. HP_SYS_CLKRST_PERI_CLK_CTRL24_REG (0x00A4)



HP_SYS_CLKRST_ADC_SAR1_CLK_DIV_NUM Configures the integer part of the clock divisor for ADC_SAR1_CLK. (R/W)

HP_SYS_CLKRST_ADC_SAR2_CLK_DIV_NUM Configures the integer part of the clock divisor for ADC_SAR2_CLK. (R/W)

Register 11.43. HP_SYS_CLKRST_PERI_CLK_CTRL25_REG (0x00A8)



HP_SYS_CLKRST_CRYPTO_CLK_SRC_SEL Configures the clock source for CRYPTO_CLK.

0: XTAL_CLK

1: RC_FAST_CLK

2: PLL_F240M_CLK

3: PLL_F160M_CLK

(R/W)

HP_SYS_CLKRST_CRYPTO_AES_CLK_EN Configures whether to enable AES_CLK.

0: Disable

1: Enable

(R/W)

HP_SYS_CLKRST_CRYPTO_DSA_CLK_EN Configures whether to enable the RSA_DS clock.

0: Disable

1: Enable

(R/W)

HP_SYS_CLKRST_CRYPTO_ECC_CLK_EN Configures whether to enable ECC_CLK.

0: Disable

1: Enable

(R/W)

HP_SYS_CLKRST_CRYPTO_HMAC_CLK_EN Configures whether to enable HMAC_CLK.

0: Disable

1: Enable

(R/W)

HP_SYS_CLKRST_CRYPTO_RSA_CLK_EN Configures whether to enable RSA_CLK.

0: Disable

1: Enable

(R/W)

Continued on the next page...

Register 11.43. HP_SYS_CLKRST_PERI_CLK_CTRL25_REG (0x00A8)

Continued from the previous page...

HP_SYS_CLKRST_CRYPTO_SEC_CLK_EN Configures whether to enable SEC_CLK.

0: Disable

1: Enable

(R/W)

HP_SYS_CLKRST_CRYPTO_SHA_CLK_EN Configures whether to enable SHA_CLK.

0: Disable

1: Enable

(R/W)

HP_SYS_CLKRST_CRYPTO_ECDSA_CLK_EN Configures whether to enable the ECDSA_DS clock (ECDSA_DS_CLK).

0: Disable

1: Enable

(R/W)

HP_SYS_CLKRST_CRYPTO_KM_CLK_EN Configures whether to enable KM_CLK.

0: Disable

1: Enable

(R/W)

HP_SYS_CLKRST_ISP_CLK_SRC_SEL Configures the clock source for ISP_CLK.

0: XTAL_CLK

1: PLL_F240M_CLK

2: PLL_F160M_CLK

3: Invalid

(R/W)

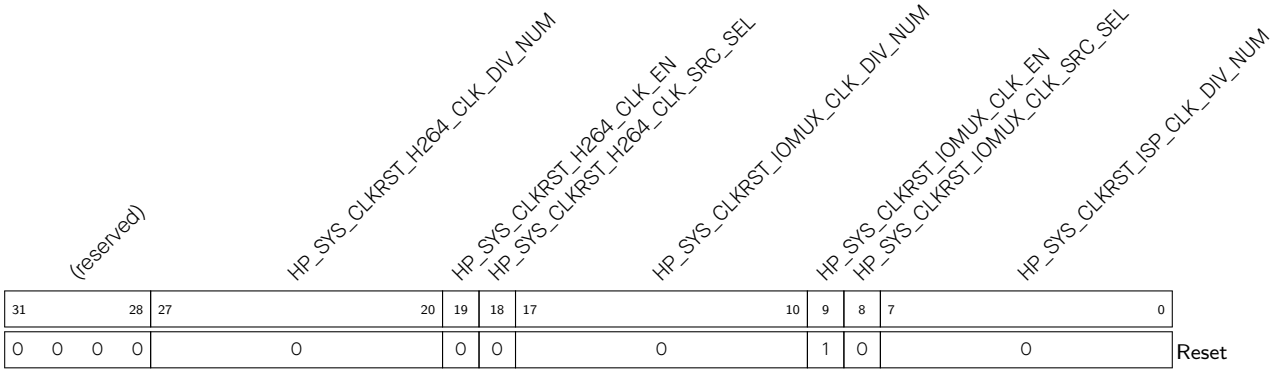
HP_SYS_CLKRST_ISP_CLK_EN Configures whether to enable ISP_CLK.

0: Disable

1: Enable

(R/W)

Register 11.44. HP_SYS_CLKRST_PERI_CLK_CTRL26_REG (0x00AC)



HP_SYS_CLKRST_ISP_CLK_DIV_NUM Configures the clock divisor of ISP_CLK. (R/W)

HP_SYS_CLKRST_IOMUX_CLK_SRC_SEL Configures the clock source for IOMUX_CLK.

0: XTAL_CLK

1: PLL_F80M_CLK

(R/W)

HP_SYS_CLKRST_IOMUX_CLK_EN Configures whether to enable the IOMUX_CLK clock.

0: Disable

1: Enable

(R/W)

HP_SYS_CLKRST_IOMUX_CLK_DIV_NUM Configures the clock divisor of IOMUX_CLK. (R/W)

HP_SYS_CLKRST_H264_CLK_SRC_SEL Configures the clock source for H264_CLK.

0: XTAL_CLK

1: PLL_F240M_CLK

(R/W)

HP_SYS_CLKRST_H264_CLK_EN Configures whether to enable the H264_CLK clock.

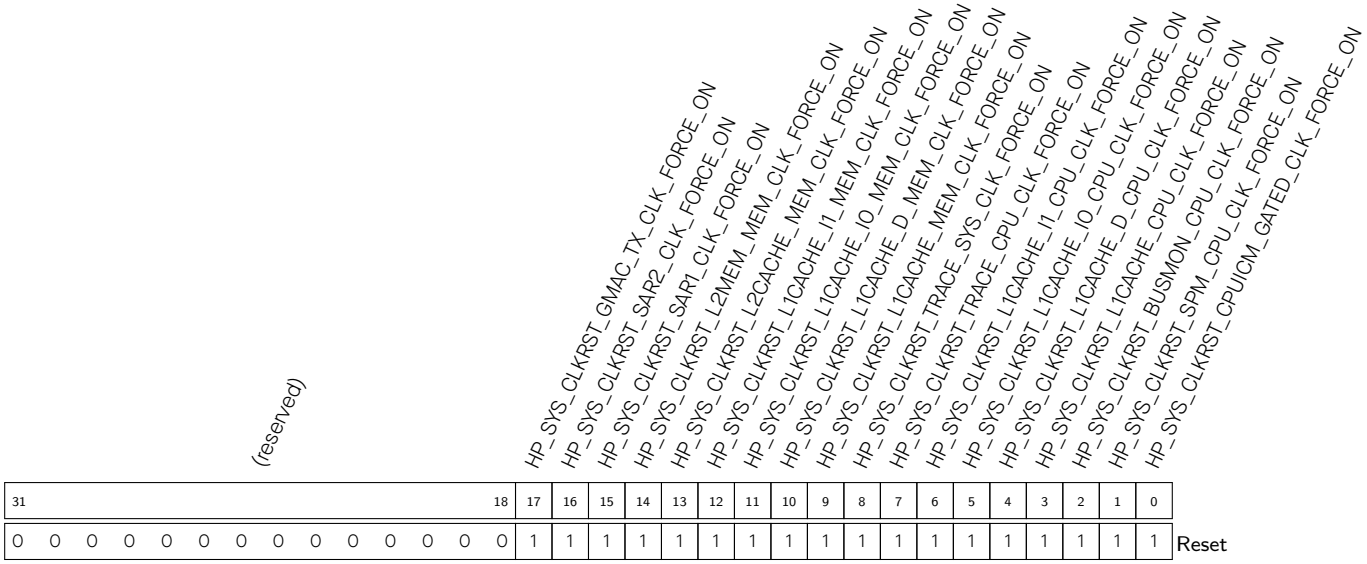
0: Disable

1: Enable

(R/W)

HP_SYS_CLKRST_H264_CLK_DIV_NUM Configures the clock divisor of H264_CLK. (R/W)

Register 11.45. HP_SYS_CLKRST_CLK_FORCE_ON_CTRL0_REG (0x00B4)



HP_SYS_CLKRST_CPUICM_GATED_CLK_FORCE_ON Configures the CPUICM_CLK clock gating path.

0: Clock gating is enabled

1: Force on the clock, bypassing the clock gating

(R/W)

HP_SYS_CLKRST_SPM_CPU_CLK_FORCE_ON Configures the SPM_CPU_CLK clock gating path.

0: Clock gating is enabled

1: Force on the clock, bypassing the clock gating

(R/W)

HP_SYS_CLKRST_BUSMON_CPU_CLK_FORCE_ON Configures the BUSMON_CPU_CLK clock gating path.

0: Clock gating is enabled

1: Force on the clock, bypassing the clock gating

(R/W)

HP_SYS_CLKRST_L1CACHE_CPU_CLK_FORCE_ON Configures the L1CACHE_CPU_CLK clock gating path.

0: Clock gating is enabled

1: Force on the clock, bypassing the clock gating

(R/W)

HP_SYS_CLKRST_L1CACHE_D_CPU_CLK_FORCE_ON Configures the L1DCACHE_CPU_CLK clock gating path.

0: Clock gating is enabled

1: Force on the clock, bypassing the clock gating

(R/W)

Continued on the next page...

Register 11.45. HP_SYS_CLKRST_CLK_FORCE_ON_CTRL0_REG (0x00B4)

Continued from the previous page...

HP_SYS_CLKRST_L1CACHE_I0_CPU_CLK_FORCE_ON Configures the L1I0CACHE_CPU_CLK clock gating path.
0: Clock gating is enabled
1: Force on the clock, bypassing the clock gating
(R/W)

HP_SYS_CLKRST_L1CACHE_I1_CPU_CLK_FORCE_ON Configures the L1I1CACHE_CPU_CLK clock gating path.
(R/W)

HP_SYS_CLKRST_TRACE_CPU_CLK_FORCE_ON Configures the TRACE_CPU_CLK clock gating path.
0: Clock gating is enabled
1: Force on the clock, bypassing the clock gating
(R/W)

HP_SYS_CLKRST_TRACE_SYS_CLK_FORCE_ON Configures the TRACE_SYS_CLK clock gating path.
0: Clock gating is enabled
1: Force on the clock, bypassing the clock gating
(R/W)

HP_SYS_CLKRST_L1CACHE_MEM_CLK_FORCE_ON Configures the L1CACHE_MEM_CLK clock gating path.
0: Clock gating is enabled
1: Force on the clock, bypassing the clock gating
(R/W)

HP_SYS_CLKRST_L1CACHE_D_MEM_CLK_FORCE_ON Configures the L1DCACHE_MEM_CLK clock gating path.
0: Clock gating is enabled
1: Force on the clock, bypassing the clock gating
(R/W)

HP_SYS_CLKRST_L1CACHE_I0_MEM_CLK_FORCE_ON Configures the L1I0CACHE_MEM_CLK clock gating path.
0: Clock gating is enabled
1: Force on the clock, bypassing the clock gating
(R/W)

HP_SYS_CLKRST_L1CACHE_I1_MEM_CLK_FORCE_ON Configures the L1I1CACHE_MEM_CLK clock gating path.
0: Clock gating is enabled
1: Force on the clock, bypassing the clock gating
(R/W)

Continued on the next page...

Register 11.45. HP_SYS_CLKRST_CLK_FORCE_ON_CTRL0_REG (0x00B4)

Continued from the previous page...

HP_SYS_CLKRST_L2CACHE_MEM_CLK_FORCE_ON Configures the L2CACHE_MEM_CLK clock gating path.
0: Clock gating is enabled
1: Force on the clock, bypassing the clock gating
(R/W)

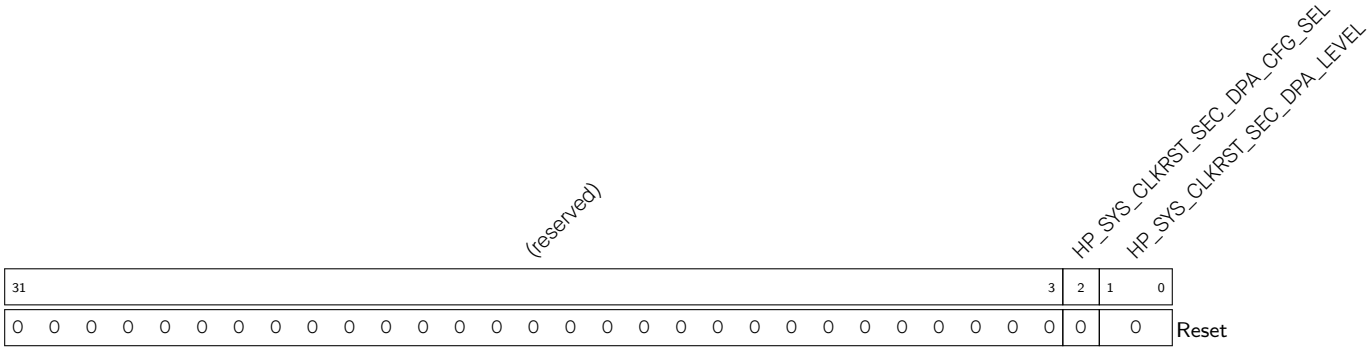
HP_SYS_CLKRST_L2MEM_MEM_CLK_FORCE_ON Configures the L2MEM_MEM_CLK clock gating path.
0: Clock gating is enabled
1: Force on the clock, bypassing the clock gating
(R/W)

HP_SYS_CLKRST_SAR1_CLK_FORCE_ON Configures the SAR1_CLK clock gating path.
0: Clock gating is enabled
1: Force on the clock, bypassing the clock gating
(R/W)

HP_SYS_CLKRST_SAR2_CLK_FORCE_ON Configures the SAR2_CLK clock gating path.
0: Clock gating is enabled
1: Force on the clock, bypassing the clock gating
(R/W)

HP_SYS_CLKRST_GMAC_TX_CLK_FORCE_ON Configures the GMAC_TX_CLK clock gating path.
0: Clock gating is enabled
1: Force on the clock, bypassing the clock gating
(R/W)

Register 11.46. HP_SYS_CLKRST_DPA_CTRL0_REG (0x00B8)



HP_SYS_CLKRST_SEC_DPA_LEVEL Configures the security level of CRYPTO_CLK against DPA attacks.

0: Low security level

1: High security level

(R/W)

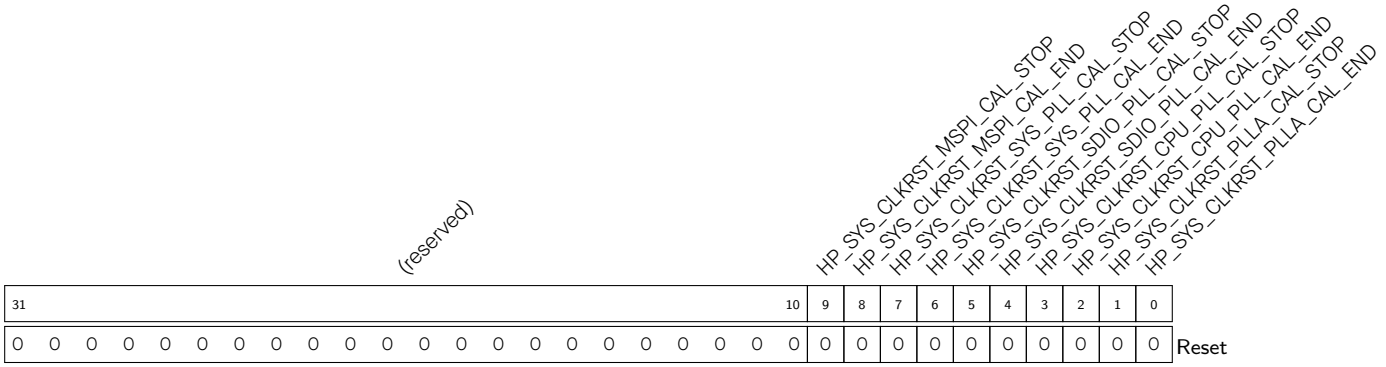
HP_SYS_CLKRST_SEC_DPA_CFG_SEL Configures the source of the security level of CRYPTO_CLK against DPA attacks.

0: The security level is determined by EFUSE_SEC_DPA_LEVEL

1: The security level is determined by HP_SYS_CLKRST_SEC_DPA_LEVEL

(R/W)

Register 11.47. HP_SYS_CLKRST_ANA_PLL_CTRL0_REG (0x00BC)



HP_SYS_CLKRST_PLLA_CAL_END Represents whether the APLL_CLK calibration is finished.

0: Calibration in progress

1: Calibration done

(RO)

HP_SYS_CLKRST_PLLA_CAL_STOP Configures whether to start or stop the APLL_CLK calibration.

0: Start calibration

1: Stop calibration

(R/W)

HP_SYS_CLKRST_CPU_PLL_CAL_END Represents whether the CPLL_CLK calibration is finished.

0: Calibration in progress

1: Calibration done

(RO)

HP_SYS_CLKRST_CPU_PLL_CAL_STOP Configures whether to start or stop the CPLL_CLK calibration.

0: Start calibration

1: Stop calibration

(R/W)

HP_SYS_CLKRST_SDIO_PLL_CAL_END Represents whether the SDIO_PLL calibration is finished.

0: Calibration in progress

1: Calibration done

(RO)

HP_SYS_CLKRST_SDIO_PLL_CAL_STOP Configures whether to start or stop the SDIO_PLL calibration.

0: Start calibration

1: Stop calibration

(R/W)

Continued on the next page...

Register 11.47. HP_SYS_CLKRST_ANA_PLL_CTRL0_REG (0x00BC)

Continued from the previous page...

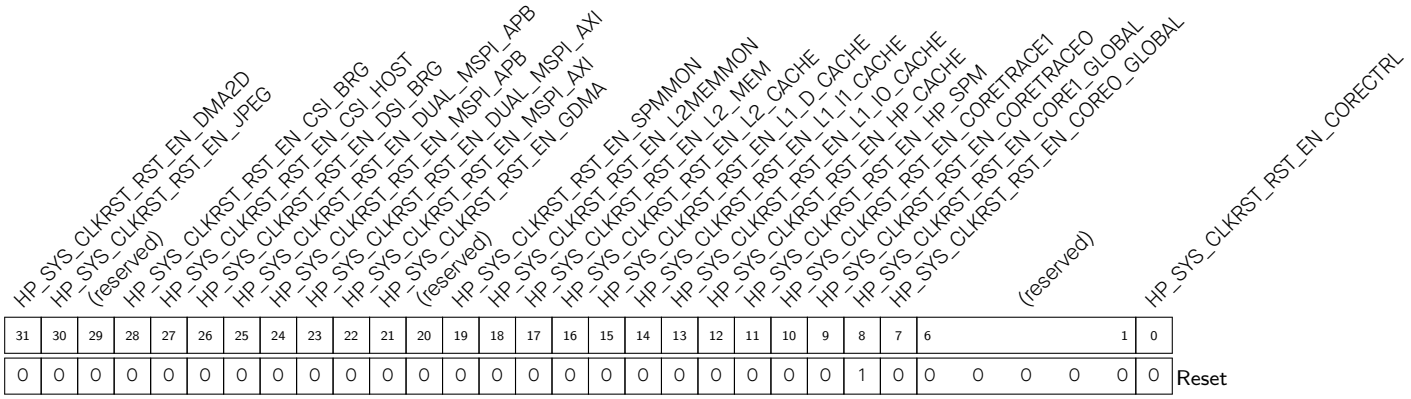
HP_SYS_CLKRST_SYS_PLL_CAL_END Represents whether the SPLL_CLK calibration is finished.
0: Calibration in progress
1: Calibration done
(RO)

HP_SYS_CLKRST_SYS_PLL_CAL_STOP Configures whether to start or stop the SPLL_CLK calibration.
0: Start calibration
1: Stop calibration
(R/W)

HP_SYS_CLKRST_MSPI_CAL_END Represents whether the MPLL_CLK calibration is finished.
0: Calibration in progress
1: Calibration done
(RO)

HP_SYS_CLKRST_MSPI_CAL_STOP Configures whether to start or stop the MPLL_CLK calibration.
0: Start calibration
1: Stop calibration
(R/W)

Register 11.48. HP_SYS_CLKRST_HP_RST_EN0_REG (0x00C0)



HP_SYS_CLKRST_RST_EN_CORECTRL Configures whether to reset CORECTRL.

0: Release from reset

1: Reset

(R/W)

HP_SYS_CLKRST_RST_EN_COREO_GLOBAL Configures whether to reset HP CPU0.

0: Release from reset

1: Reset

(R/W)

HP_SYS_CLKRST_RST_EN_CORE1_GLOBAL Configures whether to reset HP CPU1.

0: Release from reset

1: Reset

(R/W)

HP_SYS_CLKRST_RST_EN_CORETRACE0 Configures whether to reset RISC-V Trace Encoder0.

0: Release from reset

1: Reset

(R/W)

HP_SYS_CLKRST_RST_EN_CORETRACE1 Configures whether to reset RISC-V Trace Encoder1.

0: Release from reset

1: Reset

(R/W)

HP_SYS_CLKRST_RST_EN_HP_SPM Configures whether to reset SPM.

0: Release from reset

1: Reset

(R/W)

Continued on the next page...

Register 11.48. HP_SYS_CLKRST_HP_RST_EN0_REG (0x00C0)

Continued from the previous page...

HP_SYS_CLKRST_RST_EN_HP_CACHE Configures whether to reset L1 I-Cache0, L1 I-Cache1, L1 D-Cache, L2 Cache.
0: Release from reset
1: Reset
(R/W)

HP_SYS_CLKRST_RST_EN_L1_I0_CACHE Configures whether to reset L1 I-Cache0.
0: Release from reset
1: Reset
(R/W)

HP_SYS_CLKRST_RST_EN_L1_I1_CACHE Configures whether to reset L1 I-Cache1.
0: Release from reset
1: Reset
(R/W)

HP_SYS_CLKRST_RST_EN_L1_D_CACHE Configures whether to reset L1 D-Cache.
0: Release from reset
1: Reset
(R/W)

HP_SYS_CLKRST_RST_EN_L2_CACHE Configures whether to reset L2 Cache.
0: Release from reset
1: Reset
(R/W)

HP_SYS_CLKRST_RST_EN_L2_MEM Configures whether to reset L2 MEM.
0: Release from reset
1: Reset
(R/W)

HP_SYS_CLKRST_RST_EN_L2MEMMON Configures whether to reset L2 MEM monitor.
0: Release from reset
1: Reset
(R/W)

HP_SYS_CLKRST_RST_EN_SPMMON Configures whether to reset SPM monitor.
0: Release from reset
1: Reset
(R/W)

HP_SYS_CLKRST_RST_EN_GDMA Configures whether to reset VDMA.
0: Release from reset
1: Reset
(R/W)

Continued on the next page...

Register 11.48. HP_SYS_CLKRST_HP_RST_EN0_REG (0x00C0)

Continued from the previous page...

HP_SYS_CLKRST_RST_EN_MSPI_AXI Configures whether to reset FLASH MSPI AXI.

0: Release from reset

1: Reset

(R/W)

HP_SYS_CLKRST_RST_EN_DUAL_MSPI_AXI Configures whether to reset PSRAM MSPI AXI.

0: Release from reset

1: Reset

(R/W)

HP_SYS_CLKRST_RST_EN_MSPI_APB Configures whether to reset FLASH MSPI APB.

0: Release from reset

1: Reset

(R/W)

HP_SYS_CLKRST_RST_EN_DUAL_MSPI_APB Configures whether to reset PSRAM MSPI APB.

0: Release from reset

1: Reset

(R/W)

HP_SYS_CLKRST_RST_EN_DSI_BRG Configures whether to reset DSI Bridge.

0: Release from reset

1: Reset

(R/W)

HP_SYS_CLKRST_RST_EN_CSI_HOST Configures whether to reset CSI Host.

0: Release from reset

1: Reset

(R/W)

HP_SYS_CLKRST_RST_EN_CSI_BRG Configures whether to reset CSI Bridge and ISP.

0: Release from reset

1: Reset

(R/W)

HP_SYS_CLKRST_RST_EN_JPEG Configures whether to reset JPEG.

0: Release from reset

1: Reset

(R/W)

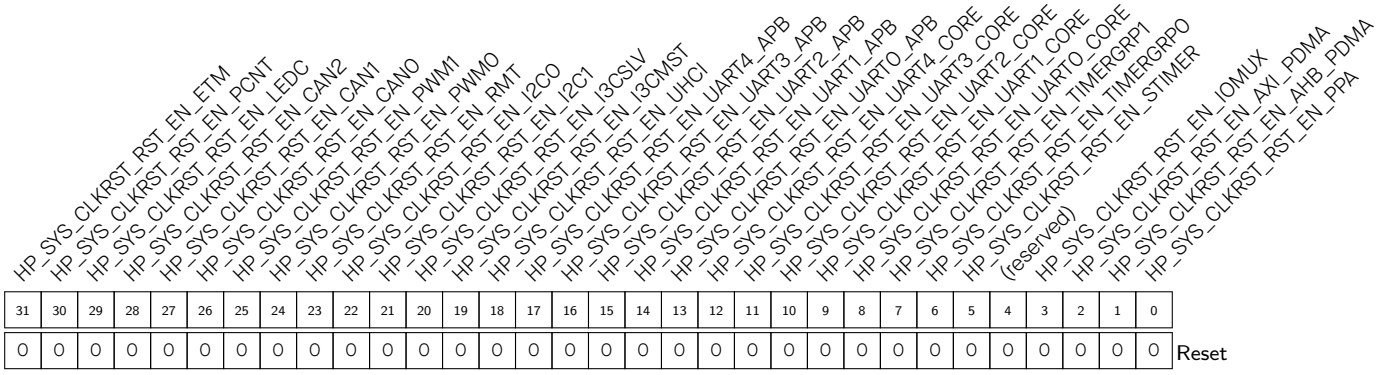
HP_SYS_CLKRST_RST_EN_DMA2D Configures whether to reset 2DDMA.

0: Release from reset

1: Reset

(R/W)

Register 11.49. HP_SYS_CLKRST_HP_RST_EN1_REG (0x00C4)



31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Reset

HP_SYS_CLKRST_RST_EN_PPA Configures whether to reset PPA.

0: Release from reset

1: Reset

(R/W)

HP_SYS_CLKRST_RST_EN_AHB_PDMA Configures whether to reset GDMA-AHB.

0: Release from reset

1: Reset

(R/W)

HP_SYS_CLKRST_RST_EN_AXI_PDMA Configures whether to reset GDMA-AXI.

0: Release from reset

1: Reset

(R/W)

HP_SYS_CLKRST_RST_EN_IOMUX Configures whether to reset IO MUX.

0: Release from reset

1: Reset

(R/W)

HP_SYS_CLKRST_RST_EN_STIMER Configures whether to reset the System Timer.

0: Release from reset

1: Reset

(R/W)

HP_SYS_CLKRST_RST_EN_TIMERGRP0 Configures whether to reset Timer Group 0.

0: Release from reset

1: Reset

(R/W)

HP_SYS_CLKRST_RST_EN_TIMERGRP1 Configures whether to reset Timer Group 1.

0: Release from reset

1: Reset

(R/W)

Continued on the next page...

Register 11.49. HP_SYS_CLKRST_HP_RST_EN1_REG (0x00C4)

Continued from the previous page...

HP_SYS_CLKRST_RST_EN_UART0_CORE Configures whether to reset UART0.

0: Release from reset

1: Reset

(R/W)

HP_SYS_CLKRST_RST_EN_UART1_CORE Configures whether to reset UART1.

0: Release from reset

1: Reset

(R/W)

HP_SYS_CLKRST_RST_EN_UART2_CORE Configures whether to reset UART2.

0: Release from reset

1: Reset

(R/W)

HP_SYS_CLKRST_RST_EN_UART3_CORE Configures whether to reset UART3.

0: Release from reset

1: Reset

(R/W)

HP_SYS_CLKRST_RST_EN_UART4_CORE Configures whether to reset UART4.

0: Release from reset

1: Reset

(R/W)

HP_SYS_CLKRST_RST_EN_UART0_APB Configures whether to reset UART0 APB.

0: Release from reset

1: Reset

(R/W)

HP_SYS_CLKRST_RST_EN_UART1_APB Configures whether to reset UART1 APB.

0: Release from reset

1: Reset

(R/W)

HP_SYS_CLKRST_RST_EN_UART2_APB Configures whether to reset UART2 APB.

0: Release from reset

1: Reset

(R/W)

Continued on the next page...

Register 11.49. HP_SYS_CLKRST_HP_RST_EN1_REG (0x00C4)

Continued from the previous page...

HP_SYS_CLKRST_RST_EN_UART3_APB Configures whether to reset UART3 APB.

0: Release from reset

1: Reset

(R/W)

HP_SYS_CLKRST_RST_EN_UART4_APB Configures whether to reset UART4 APB.

0: Release from reset

1: Reset

(R/W)

HP_SYS_CLKRST_RST_EN_UHCI Configures whether to reset UHCI.

0: Release from reset

1: Reset

(R/W)

HP_SYS_CLKRST_RST_EN_I3CMST Configures whether to reset I3C Master.

0: Release from reset

1: Reset

(R/W)

HP_SYS_CLKRST_RST_EN_I3CSLV Configures whether to reset I3C Slave.

0: Release from reset

1: Reset

(R/W)

HP_SYS_CLKRST_RST_EN_I2C1 Configures whether to reset I2C1.

0: Release from reset

1: Reset

(R/W)

HP_SYS_CLKRST_RST_EN_I2C0 Configures whether to reset I2C0.

0: Release from reset

1: Reset

(R/W)

HP_SYS_CLKRST_RST_EN_RMT Configures whether to reset RMT.

0: Release from reset

1: Reset

(R/W)

HP_SYS_CLKRST_RST_EN_PWM0 Configures whether to reset MCPMW0.

0: Release from reset

1: Reset

(R/W)

Continued on the next page...

Register 11.49. HP_SYS_CLKRST_HP_RST_EN1_REG (0x00C4)

Continued from the previous page...

HP_SYS_CLKRST_RST_EN_PWM1 Configures whether to reset MCPWM1.

0: Release from reset

1: Reset

(R/W)

HP_SYS_CLKRST_RST_EN_CAN0 Configures whether to reset TWAI0.

0: Release from reset

1: Reset

(R/W)

HP_SYS_CLKRST_RST_EN_CAN1 Configures whether to reset TWAI.

0: Release from reset

1: Reset

(R/W)

HP_SYS_CLKRST_RST_EN_CAN2 Configures whether to reset TWAI.

0: Release from reset

1: Reset

(R/W)

HP_SYS_CLKRST_RST_EN_LEDC Configures whether to reset LEDC.

0: Release from reset

1: Reset

(R/W)

HP_SYS_CLKRST_RST_EN_PCNT Configures whether to reset PCNT.

0: Release from reset

1: Reset

(R/W)

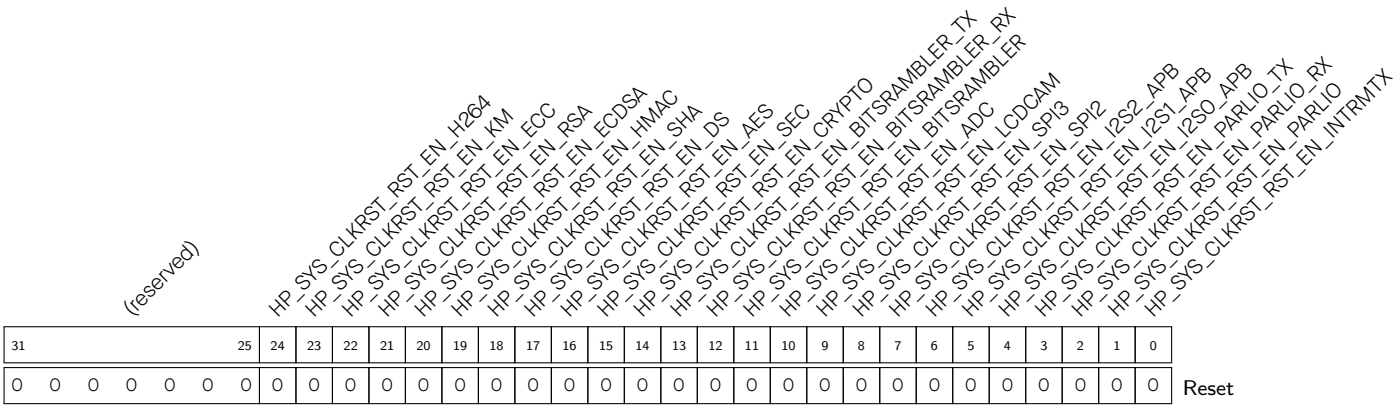
HP_SYS_CLKRST_RST_EN_ETM Configures whether to reset ETM.

0: Release from reset

1: Reset

(R/W)

Register 11.50. HP_SYS_CLKRST_HP_RST_EN2_REG (0x00C8)



HP_SYS_CLKRST_RST_EN_INTRMTX Configures whether to reset the Interrupt Matrix.

0: Release from reset

1: Reset

(R/W)

HP_SYS_CLKRST_RST_EN_PARLIO Configures whether to reset PARLIO.

0: Release from reset

1: Reset

(R/W)

HP_SYS_CLKRST_RST_EN_PARLIO_RX Configures whether to reset PARLIO RX.

0: Release from reset

1: Reset

(R/W)

HP_SYS_CLKRST_RST_EN_PARLIO_TX Configures whether to reset PARLIO TX.

0: Release from reset

1: Reset

(R/W)

HP_SYS_CLKRST_RST_EN_I2S0_APB Configures whether to reset I2S0.

0: Release from reset

1: Reset

(R/W)

HP_SYS_CLKRST_RST_EN_I2S1_APB Configures whether to reset I2S1.

0: Release from reset

1: Reset

(R/W)

HP_SYS_CLKRST_RST_EN_I2S2_APB Configures whether to reset I2S2.

0: Release from reset

1: Reset

(R/W)

Continued on the next page...

Register 11.50. HP_SYS_CLKRST_HP_RST_EN2_REG (0x00C8)

Continued from the previous page...

HP_SYS_CLKRST_RST_EN_SPI2 Configures whether to reset SPI2.

0: Release from reset

1: Reset

(R/W)

HP_SYS_CLKRST_RST_EN_SPI3 Configures whether to reset SPI3.

0: Release from reset

1: Reset

(R/W)

HP_SYS_CLKRST_RST_EN_LCDCAM Configures whether to reset the LCD_CAM.

0: Release from reset

1: Reset

(R/W)

HP_SYS_CLKRST_RST_EN_ADC Configures whether to reset ADC.

0: Release from reset

1: Reset

(R/W)

HP_SYS_CLKRST_RST_EN_BITSRAMBLER Configures whether to reset the Bit Scrambler.

0: Release from reset

1: Reset

(R/W)

HP_SYS_CLKRST_RST_EN_BITSRAMBLER_RX Configures whether to reset the Bit Scrambler RX.

0: Release from reset

1: Reset

(R/W)

HP_SYS_CLKRST_RST_EN_BITSRAMBLER_TX Configures whether to reset the Bit Scrambler TX.

0: Release from reset

1: Reset

(R/W)

HP_SYS_CLKRST_RST_EN_CRYPTO Configures whether to reset the crypto modules.

0: Release from reset

1: Reset

(R/W)

HP_SYS_CLKRST_RST_EN_SEC Configures whether to reset SEC.

0: Release from reset

1: Reset

(R/W)

Continued on the next page...

Register 11.50. HP_SYS_CLKRST_HP_RST_EN2_REG (0x00C8)

Continued from the previous page...

HP_SYS_CLKRST_RST_EN_AES Configures whether to reset AES.

0: Release from reset

1: Reset

(R/W)

HP_SYS_CLKRST_RST_EN_DS Configures whether to reset the RSA_DS peripheral.

0: Release from reset

1: Reset

(R/W)

HP_SYS_CLKRST_RST_EN_SHA Configures whether to reset SHA.

0: Release from reset

1: Reset

(R/W)

HP_SYS_CLKRST_RST_EN_HMAC Configures whether to reset HMAC.

0: Release from reset

1: Reset

(R/W)

HP_SYS_CLKRST_RST_EN_ECDSA Configures whether to reset ECDSA_DS.

0: Release from reset

1: Reset

(R/W)

HP_SYS_CLKRST_RST_EN_RSA Configures whether to reset RSA.

0: Release from reset

1: Reset

(R/W)

HP_SYS_CLKRST_RST_EN_ECC Configures whether to reset ECC.

0: Release from reset

1: Reset

(R/W)

HP_SYS_CLKRST_RST_EN_KM Configures whether to reset the Key Manager.

0: Release from reset

1: Reset

(R/W)

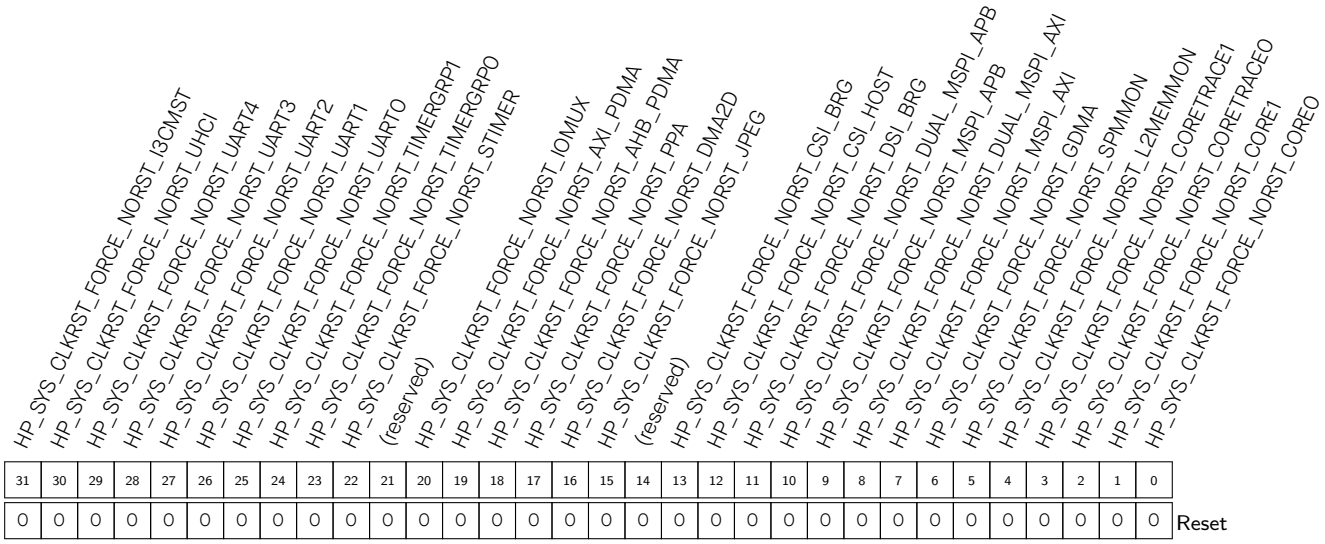
HP_SYS_CLKRST_RST_EN_H264 Configures whether to reset H264.

0: Release from reset

1: Reset

(R/W)

Register 11.51. HP_SYS_CLKRST_HP_FORCE_NORST0_REG (0x00CC)



HP_SYS_CLKRST_FORCE_NORST_CORE0 Configures whether HP CPU0 can be reset.

0: Can be reset

1: Force not to be reset

(R/W)

HP_SYS_CLKRST_FORCE_NORST_CORE1 Configures whether HP CPU1 can be reset.

0: Can be reset

1: Force not to be reset

(R/W)

HP_SYS_CLKRST_FORCE_NORST_CORETRACE0 Configures whether RISC-V Trace Encoder 0 can be reset.

0: Can be reset

1: Force not to be reset

(R/W)

HP_SYS_CLKRST_FORCE_NORST_CORETRACE1 Configures whether RISC-V Trace Encoder 1 can be reset.

0: Can be reset

1: Force not to be reset

(R/W)

HP_SYS_CLKRST_FORCE_NORST_L2MEMMON Configures whether L2 MEM monitor can be reset.

0: Can be reset

1: Force not to be reset

(R/W)

HP_SYS_CLKRST_FORCE_NORST_SPMMON Configures whether SPM monitor can be reset.

0: Can be reset

1: Force not to be reset

(R/W)

Continued on the next page...

Register 11.51. HP_SYS_CLKRST_HP_FORCE_NORST0_REG (0x00CC)

Continued from the previous page...

HP_SYS_CLKRST_FORCE_NORST_GDMA Configures whether VDMA can be reset.

0: Can be reset

1: Force not to be reset

(R/W)

HP_SYS_CLKRST_FORCE_NORST_MSPI_AXI Configures whether FLASH MSPI can be reset.

0: Can be reset

1: Force not to be reset

(R/W)

HP_SYS_CLKRST_FORCE_NORST_DUAL_MSPI_AXI Configures whether PSRAM MSPI can be reset.

0: Can be reset

1: Force not to be reset

(R/W)

HP_SYS_CLKRST_FORCE_NORST_MSPI_APB Configures whether FLASH MSPI APB can be reset.

0: Can be reset

1: Force not to be reset

(R/W)

HP_SYS_CLKRST_FORCE_NORST_DUAL_MSPI_APB Configures whether PSRAM MSPI APB can be reset.

0: Can be reset

1: Force not to be reset

(R/W)

HP_SYS_CLKRST_FORCE_NORST_DSI_BRG Configures whether DSI Bridge can be reset.

0: Can be reset

1: Force not to be reset

(R/W)

HP_SYS_CLKRST_FORCE_NORST_CSI_HOST Configures whether CSI Host can be reset.

0: Can be reset

1: Force not to be reset

(R/W)

HP_SYS_CLKRST_FORCE_NORST_CSI_BRG Configures whether CSI Bridge, ISP can be reset.

0: Can be reset

1: Force not to be reset

(R/W)

Continued on the next page...

Register 11.51. HP_SYS_CLKRST_HP_FORCE_NORST0_REG (0x00CC)

Continued from the previous page...

HP_SYS_CLKRST_FORCE_NORST_JPEG Configures whether JPEG can be reset.

0: Can be reset

1: Force not to be reset

(R/W)

HP_SYS_CLKRST_FORCE_NORST_DMA2D Configures whether 2DDMA can be reset.

0: Can be reset

1: Force not to be reset

(R/W)

HP_SYS_CLKRST_FORCE_NORST_PPA Configures whether PPA can be reset.

0: Can be reset

1: Force not to be reset

(R/W)

HP_SYS_CLKRST_FORCE_NORST_AHB_PDMA Configures whether GDMA-AHB can be reset.

0: Can be reset

1: Force not to be reset

(R/W)

HP_SYS_CLKRST_FORCE_NORST_AXI_PDMA Configures whether GDMA-AXI can be reset.

0: Can be reset

1: Force not to be reset

(R/W)

HP_SYS_CLKRST_FORCE_NORST_IOMUX Configures whether IO MUX can be reset.

0: Can be reset

1: Force not to be reset

(R/W)

HP_SYS_CLKRST_FORCE_NORST_STIMER Configures whether the System Timer can be reset.

0: Can be reset

1: Force not to be reset

(R/W)

HP_SYS_CLKRST_FORCE_NORST_TIMERGRP0 Configures whether Timer Group 0 can be reset.

0: Can be reset

1: Force not to be reset

(R/W)

HP_SYS_CLKRST_FORCE_NORST_TIMERGRP1 Configures whether Timer Group 1 can be reset.

0: Can be reset

1: Force not to be reset

(R/W)

Continued on the next page...

Register 11.51. HP_SYS_CLKRST_HP_FORCE_NORST0_REG (0x00CC)

Continued from the previous page...

HP_SYS_CLKRST_FORCE_NORST_UART0 Configures whether UART0 can be reset.

0: Can be reset

1: Force not to be reset

(R/W)

HP_SYS_CLKRST_FORCE_NORST_UART1 Configures whether UART1 can be reset.

0: Can be reset

1: Force not to be reset

(R/W)

HP_SYS_CLKRST_FORCE_NORST_UART2 Configures whether UART2 can be reset.

0: Can be reset

1: Force not to be reset

(R/W)

HP_SYS_CLKRST_FORCE_NORST_UART3 Configures whether UART3 can be reset.

0: Can be reset

1: Force not to be reset

(R/W)

HP_SYS_CLKRST_FORCE_NORST_UART4 Configures whether UART4 can be reset.

0: Can be reset

1: Force not to be reset

(R/W)

HP_SYS_CLKRST_FORCE_NORST_UHCI Configures whether UHCI can be reset.

0: Can be reset

1: Force not to be reset

(R/W)

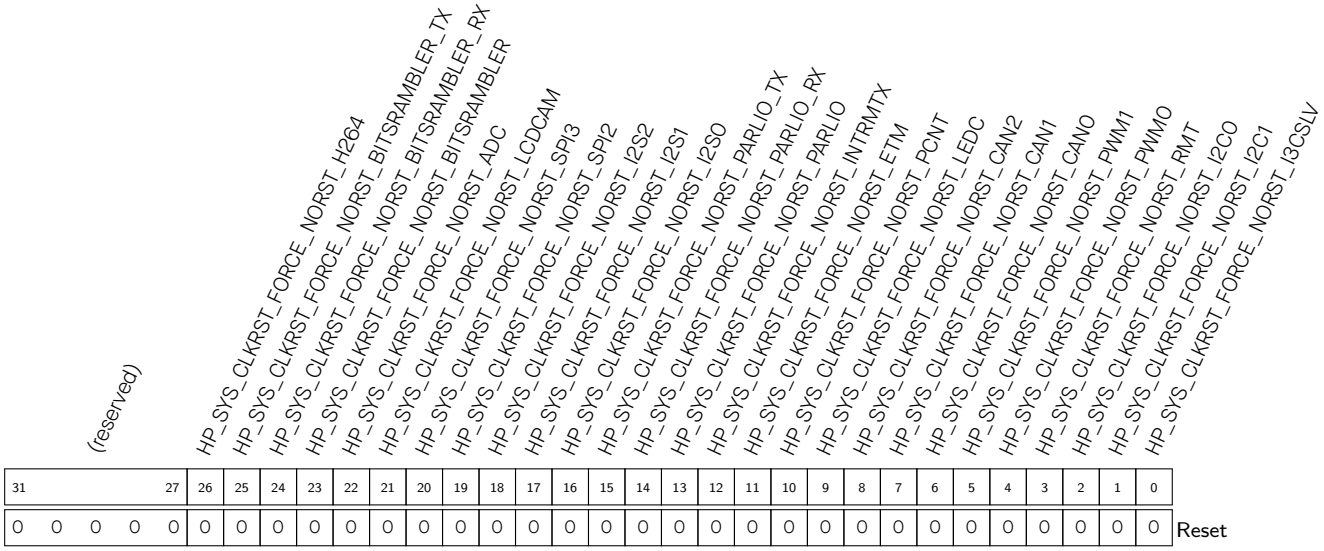
HP_SYS_CLKRST_FORCE_NORST_I3CMST Configures whether I3C Master can be reset.

0: Can be reset

1: Force not to be reset

(R/W)

Register 11.52. HP_SYS_CLKRST_HP_FORCE_NORST1_REG (0x00D0)



HP_SYS_CLKRST_FORCE_NORST_I3CSLV Configures whether I3C Slave can be reset.

0: Can be reset

1: Force not to be reset

(R/W)

HP_SYS_CLKRST_FORCE_NORST_I2C1 Configures whether I2C1 can be reset.

0: Can be reset

1: Force not to be reset

(R/W)

HP_SYS_CLKRST_FORCE_NORST_I2C0 Configures whether I2C0 can be reset.

0: Can be reset

1: Force not to be reset

(R/W)

HP_SYS_CLKRST_FORCE_NORST_RMT Configures whether RMT can be reset.

0: Can be reset

1: Force not to be reset

(R/W)

HP_SYS_CLKRST_FORCE_NORST_PWM0 Configures whether MCPWM0 can be reset.

0: Can be reset

1: Force not to be reset

(R/W)

HP_SYS_CLKRST_FORCE_NORST_PWM1 Configures whether MCPWM1 can be reset.

0: Can be reset

1: Force not to be reset

(R/W)

HP_SYS_CLKRST_FORCE_NORST_CAN0 Configures whether TWAI0 can be reset.

0: Can be reset

1: Force not to be reset

(R/W)

Continued on the next page...

Register 11.52. HP_SYS_CLKRST_HP_FORCE_NORST1_REG (0x00D0)

Continued from the previous page...

HP_SYS_CLKRST_FORCE_NORST_CAN1 Configures whether TWAI1 can be reset.

0: Can be reset

1: Force not to be reset

(R/W)

HP_SYS_CLKRST_FORCE_NORST_CAN2 Configures whether TWAI2 can be reset.

0: Can be reset

1: Force not to be reset

(R/W)

HP_SYS_CLKRST_FORCE_NORST_LEDC Configures whether LEDC can be reset.

0: Can be reset

1: Force not to be reset

(R/W)

HP_SYS_CLKRST_FORCE_NORST_PCNT Configures whether PCNT can be reset.

0: Can be reset

1: Force not to be reset

(R/W)

HP_SYS_CLKRST_FORCE_NORST_ETM Configures whether ETM can be reset.

0: Can be reset

1: Force not to be reset

(R/W)

HP_SYS_CLKRST_FORCE_NORST_INTRMTX Configures whether the interrupt matrix can be reset.

0: Can be reset

1: Force not to be reset

(R/W)

HP_SYS_CLKRST_FORCE_NORST_PARLIO Configures whether PARLIO can be reset.

0: Can be reset

1: Force not to be reset

(R/W)

HP_SYS_CLKRST_FORCE_NORST_PARLIO_RX Configures whether PARLIO RX can be reset.

0: Can be reset

1: Force not to be reset

(R/W)

HP_SYS_CLKRST_FORCE_NORST_PARLIO_TX Configures whether PARLIO TX can be reset.

0: Can be reset

1: Force not to be reset

(R/W)

Continued on the next page...

Register 11.52. HP_SYS_CLKRST_HP_FORCE_NORST1_REG (0x00D0)

Continued from the previous page...

HP_SYS_CLKRST_FORCE_NORST_I2S0 Configures whether I2S0 can be reset.

0: Can be reset

1: Force not to be reset

(R/W)

HP_SYS_CLKRST_FORCE_NORST_I2S1 Configures whether I2S1 can be reset.

0: Can be reset

1: Force not to be reset

(R/W)

HP_SYS_CLKRST_FORCE_NORST_I2S2 Configures whether I2S2 can be reset.

0: Can be reset

1: Force not to be reset

(R/W)

HP_SYS_CLKRST_FORCE_NORST_SPI2 Configures whether SPI2 can be reset.

0: Can be reset

1: Force not to be reset

(R/W)

HP_SYS_CLKRST_FORCE_NORST_SPI3 Configures whether SPI3 can be reset.

0: Can be reset

1: Force not to be reset

(R/W)

HP_SYS_CLKRST_FORCE_NORST_LCDCAM Configures whether the LCD_CAM can be reset.

0: Can be reset

1: Force not to be reset

(R/W)

HP_SYS_CLKRST_FORCE_NORST_ADC Configures whether ADC can be reset.

0: Can be reset

1: Force not to be reset

(R/W)

Continued on the next page...

Register 11.52. HP_SYS_CLKRST_HP_FORCE_NORST1_REG (0x00D0)

Continued from the previous page...

HP_SYS_CLKRST_FORCE_NORST_BITSRAMBLER Configures whether the Bit Scrambler can be reset.

0: Can be reset

1: Force not to be reset

(R/W)

HP_SYS_CLKRST_FORCE_NORST_BITSRAMBLER_RX Configures whether the Bit Scrambler RX can be reset.

0: Can be reset

1: Force not to be reset

(R/W)

HP_SYS_CLKRST_FORCE_NORST_BITSRAMBLER_TX Configures whether the Bit Scrambler TX can be reset.

0: Can be reset

1: Force not to be reset

(R/W)

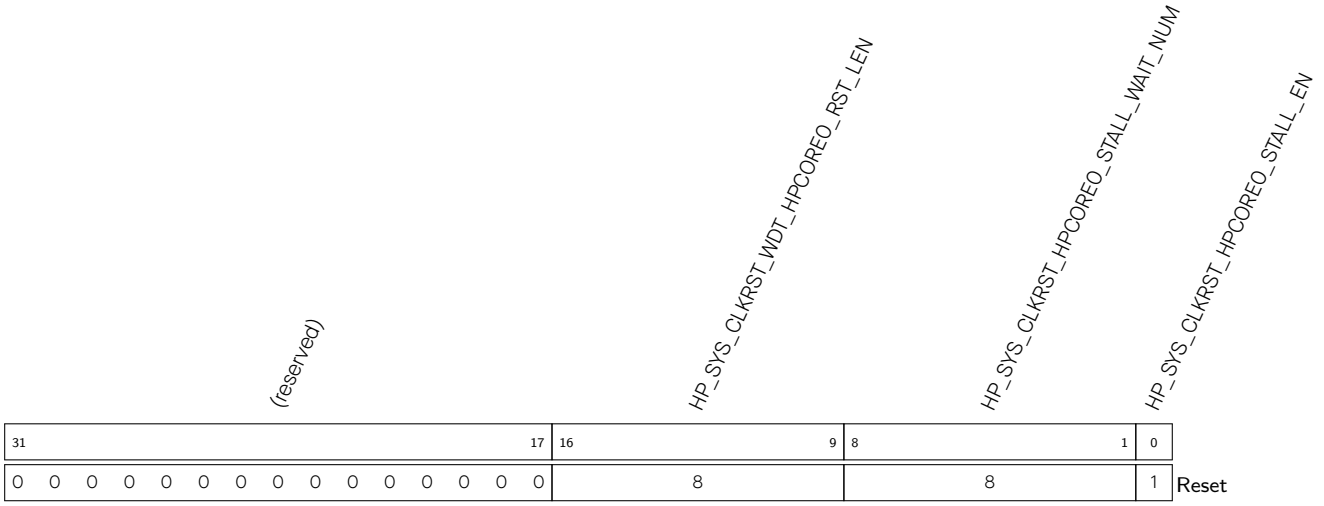
HP_SYS_CLKRST_FORCE_NORST_H264 Configures whether H264 can be reset.

0: Can be reset

1: Force not to be reset

(R/W)

Register 11.53. HP_SYS_CLKRST_HPWDT_CORE0_RST_CTRL0_REG (0x00D4)



HP_SYS_CLKRST_HPCORE0_STALL_EN Configures the behavior when MWDT triggers HP CPU0 reset.

0: Immediately reset HP CPU0

1: Stall HP CPU0 first

(R/W)

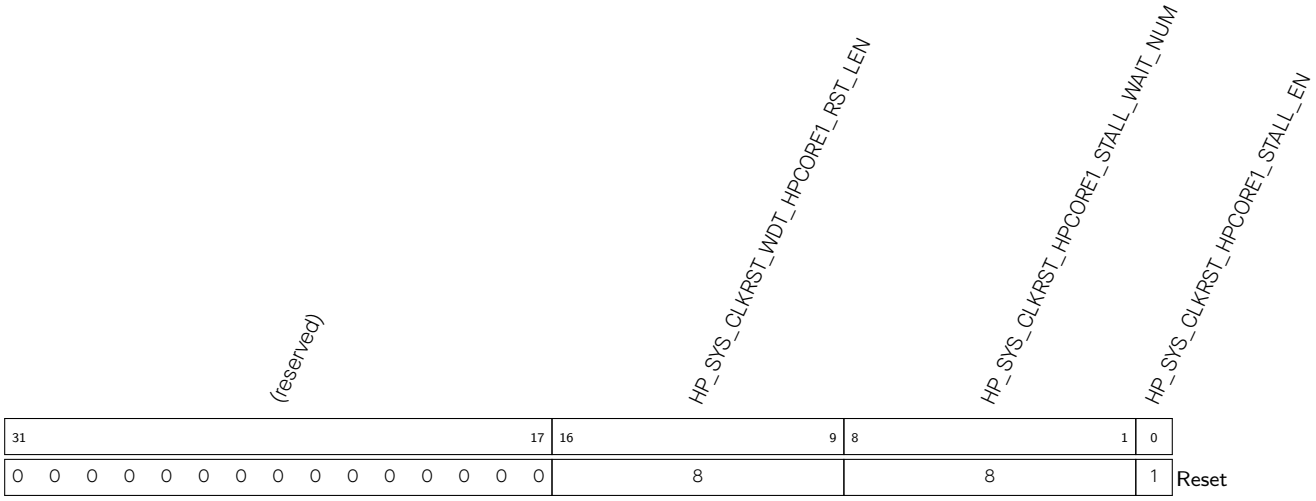
HP_SYS_CLKRST_HPCORE0_STALL_WAIT_NUM Configures the duration of HP CPU0 stall when MWDT triggers HP CPU0 reset and HP_SYS_CLKRST_HPCORE0_STALL_EN is set to 1.

Measurement unit: Clock cycles. (R/W)

HP_SYS_CLKRST_WDT_HPCORE0_RST_LEN Configures the duration of the reset when MWDT triggers HP CPU0 reset.

Measurement unit: Clock cycles. (R/W)

Register 11.54. HP_SYS_CLKRST_HPWDT_CORE1_RST_CTRL0_REG (0x00D8)



HP_SYS_CLKRST_HPCORE1_STALL_EN Configures the behavior when MWDT triggers HP CPU1 reset.

0: Immediately reset HP CPU1

1: Stall HP CPU1 first

(R/W)

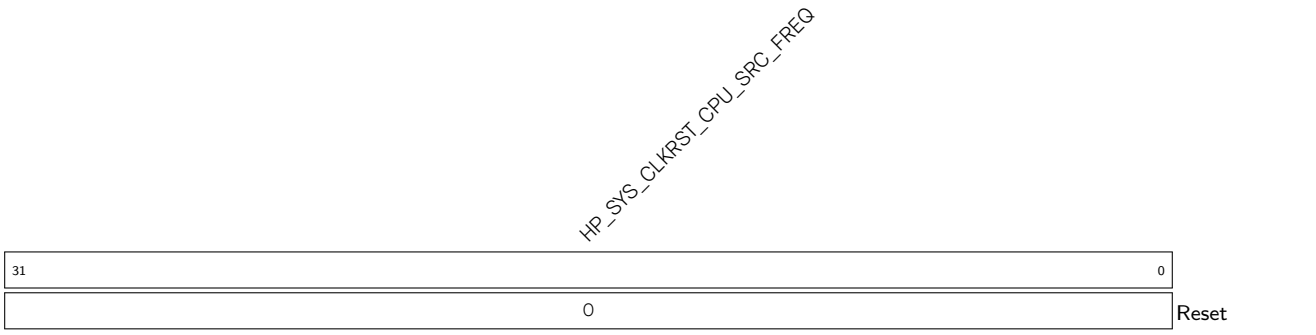
HP_SYS_CLKRST_HPCORE1_STALL_WAIT_NUM Configures the duration of HP CPU1 stall when MWDT triggers HP CPU1 reset and HP_SYS_CLKRST_HPCORE1_STALL_EN is set to 1.

Measurement unit: Clock cycles. (R/W)

HP_SYS_CLKRST_WDT_HPCORE1_RST_LEN Configures the duration of the reset when MWDT triggers HP CPU1 reset.

Measurement unit: Clock cycles. (R/W)

Register 11.55. HP_SYS_CLKRST_CPU_SRC_FREQ0_REG (0x00DC)



HP_SYS_CLKRST_CPU_SRC_FREQ Represents the CPU source clock frequency.

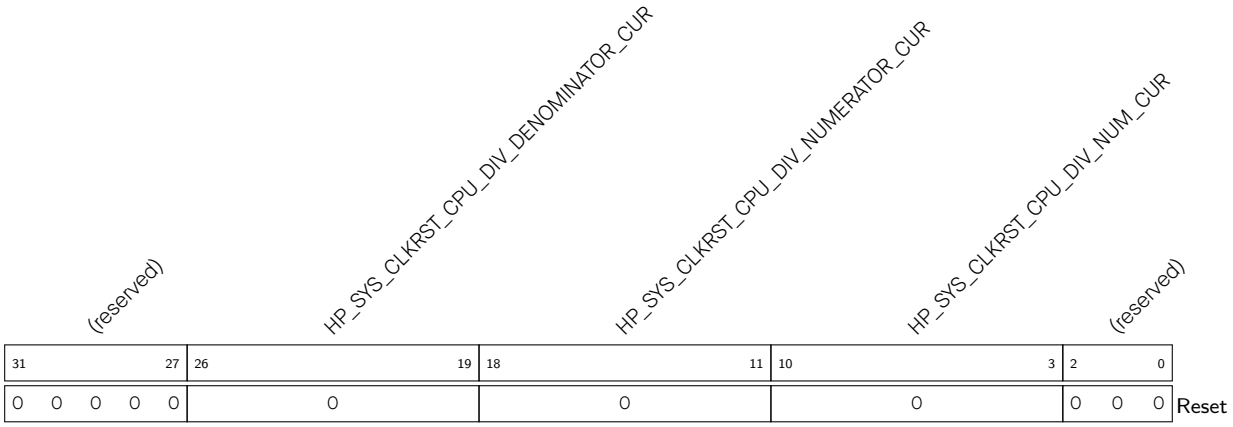
80: RC_FAST_CLK

160: XTAL_CLK

1600: CPLL_CLK

(RO)

Register 11.56. HP_SYS_CLKRST_CPU_CLK_STATUS0_REG (0x00E0)

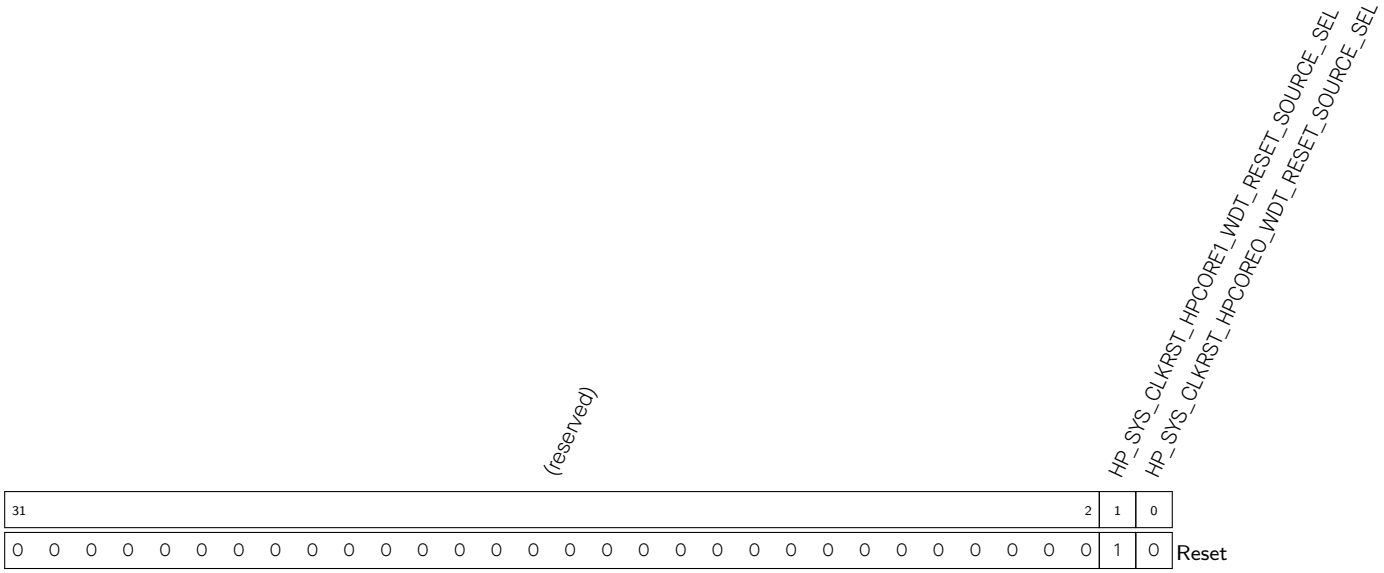


HP_SYS_CLKRST_CPU_DIV_NUM_CUR Represents the integer part of the current CPU_CLK clock divisor. (RO)

HP_SYS_CLKRST_CPU_DIV_NUMERATOR_CUR Represents the current numerator of the divisor's fractional part for the CPU_CLK. (RO)

HP_SYS_CLKRST_CPU_DIV_DENOMINATOR_CUR Represents the current denominator of the divisor's fractional part for the CPU_CLK. (RO)

Register 11.57. HP_SYS_CLKRST_HPCORE_WDT_RESET_SOURCE0_REG (0x00EC)



HP_SYS_CLKRST_HPCORE0_WDT_RESET_SOURCE_SEL Configures the MWDT used to trigger the HP CPU0 reset.

0: MWDT0

1: MWDT1

(R/W)

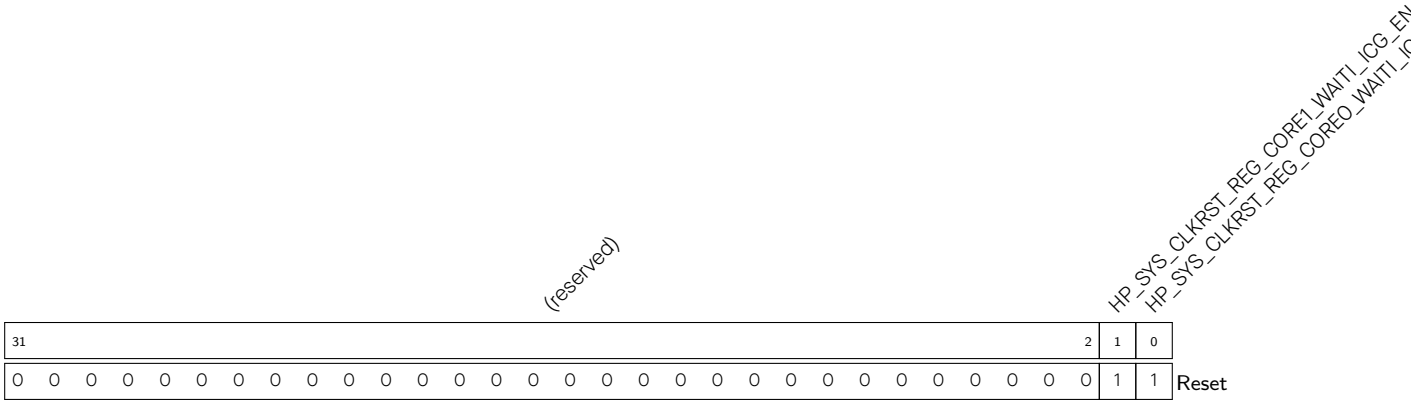
HP_SYS_CLKRST_HPCORE1_WDT_RESET_SOURCE_SEL Configures the MWDT used to trigger the HP CPU1 reset.

0: MWDT0

1: MWDT1

(R/W)

Register 11.58. HP_SYS_CLKRST_CPU_WAITI_CTRL0_REG (0x00F4)



HP_SYS_CLKRST_REG_CORE0_WAITI_ICG_EN Configures whether CPU Core0's WAITI signal is used to control the clock gate. 0: Do not use the WAITI signal to control the clock gate. 1: Use the WAITI signal to control the clock gate. If this bit is set for both Core0 and Core1, the related clock is gated only when both Core0's and Core1's WAITI signals are active. (R/W)

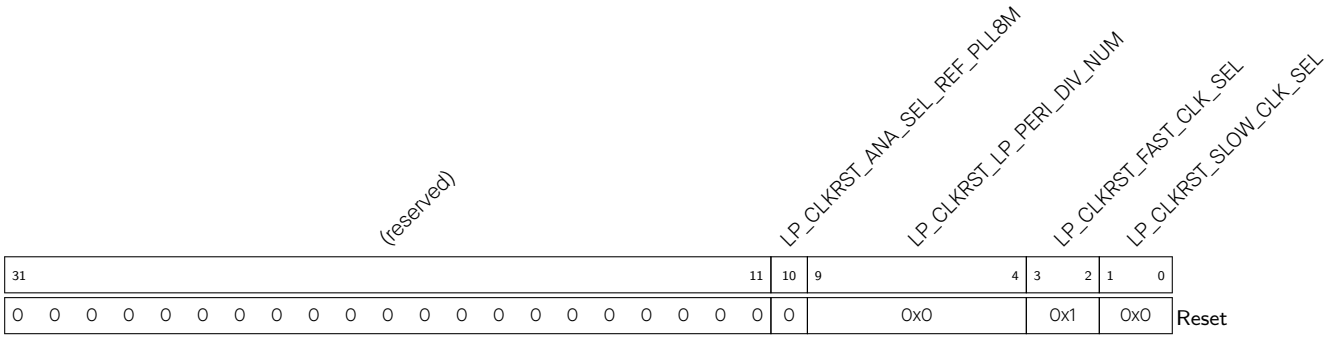
HP_SYS_CLKRST_REG_CORE1_WAITI_ICG_EN Configures whether CPU Core1's WAITI signal is used to control the clock gate. 0: Do not use the WAITI signal to control the clock gate. 1: Use the WAITI signal to control the clock gate. If this bit is set for both Core0 and Core1, the related clock is gated only when both Core0's and Core1's WAITI signals are active. (R/W)

11.5.2 LP Always on Clock and Reset (LP_CLKRST) Registers

The addresses in this section are relative to LP_CLKRST base address provided in Table 9.3-2 in Chapter 9 *System and Memory*.

For how to program reserved fields, please refer to Section Programming Reserved Register Field.

Register 11.59. LP_CLKRST_LP_CLK_CONF_REG (0x0000)



LP_CLKRST_SLOW_CLK_SEL Configures the clock source of LP_SLOW_CLK.

0: RC_SLOW_CLK

1: XTAL32K_CLK

2: Invalid

3: OSC_SLOW_CLK

(R/W)

LP_CLKRST_FAST_CLK_SEL Configures the clock source of LP_FAST_CLK.

0: RC_FAST_CLK

1: XTAL_CLK

2: PLL_LP_CLK

3: Invalid

(R/W)

LP_CLKRST_LP_PERI_DIV_NUM Configures the clock divisor for LP_PERI_CLK. (R/W)

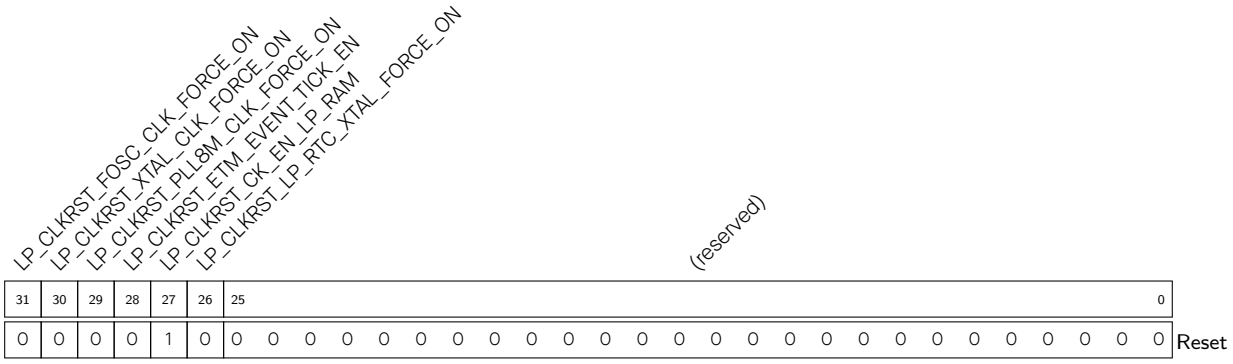
LP_CLKRST_ANA_SEL_REF_PLL8M Configures the reference clock of PLL_LP_CLK.

0: Invalid

1: XTAL32K_CLK

(R/W)

Register 11.60. LP_CLKRST_LP_CLK_EN_REG (0x0008)



LP_CLKRST_LP_RTC_XTAL_FORCE_ON Configures whether to force on the XTAL_CLK control of LP CALI.

0: Controlled by hardware

1: Force on, bypassing hardware control

(R/W)

LP_CLKRST_CK_EN_LP_RAM Configurewhether to enable the clock of LP RAM.

0: Disable

1: Enable

(R/W)

LP_CLKRST_ETM_EVENT_TICK_EN Configures whether to enable the ETM event timer.

0: Disable

1: Enable

(R/W)

LP_CLKRST_PLL8M_CLK_FORCE_ON Configures whether to force on PLL_LP_CLK as LP_FAST_CLK source.

0: Controlled by hardware

1: Force on, bypassing hardware control

(R/W)

LP_CLKRST_XTAL_CLK_FORCE_ON Configures whether to force on XTAL_CLK as LP_FAST_CLK source.

0: Controlled by hardware

1: Force on, bypassing hardware control

(R/W)

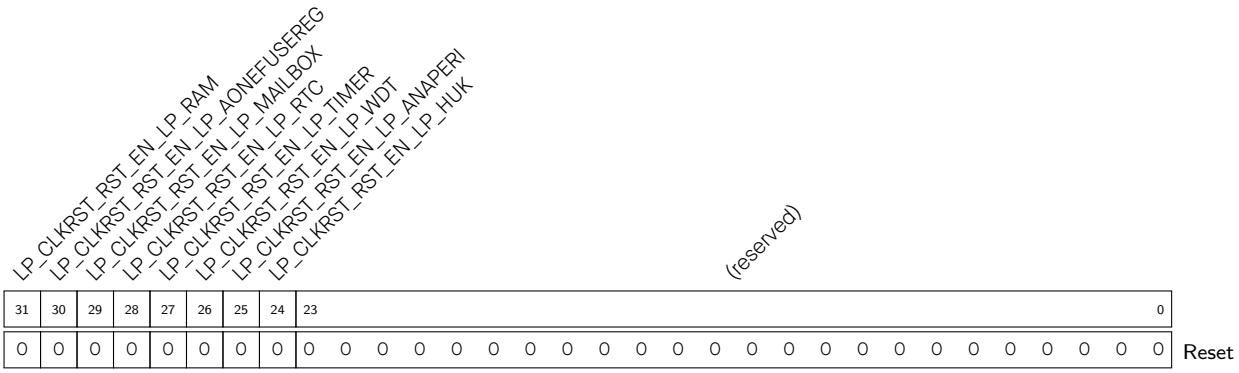
LP_CLKRST_FOSC_CLK_FORCE_ON Configures whether to force on RC_FAST_CLK as LP_FAST_CLK source.

0: Controlled by hardware

1: Force on, bypassing hardware control

(R/W)

Register 11.61. LP_CLKRST_LP_RST_EN_REG (0x000C)



LP_CLKRST_RST_EN_LP_HUK Configures whether to reset LP HUK.

0: Release from reset

1: Reset

(R/W)

LP_CLKRST_RST_EN_LP_ANAPERI Configures whether to reset LP analog peripherals.

0: Release from reset

1: Reset

(R/W)

LP_CLKRST_RST_EN_LP_WDT Configures whether to reset RWDT.

0: Release from reset

1: Reset

(R/W)

LP_CLKRST_RST_EN_LP_TIMER Configures whether to reset LP timer.

0: Release from reset

1: Reset

(R/W)

LP_CLKRST_RST_EN_LP_RTC Configures whether to reset the LP clock calibration module.

0: Release from reset

1: Reset

(R/W)

LP_CLKRST_RST_EN_LP_MAILBOX Configures whether to reset LP Mailbox.

0: Release from reset

1: Reset

(R/W)

LP_CLKRST_RST_EN_LP_AONEFUSEREG Configures whether to reset LP eFuse.

0: Release from reset

1: Reset

(R/W)

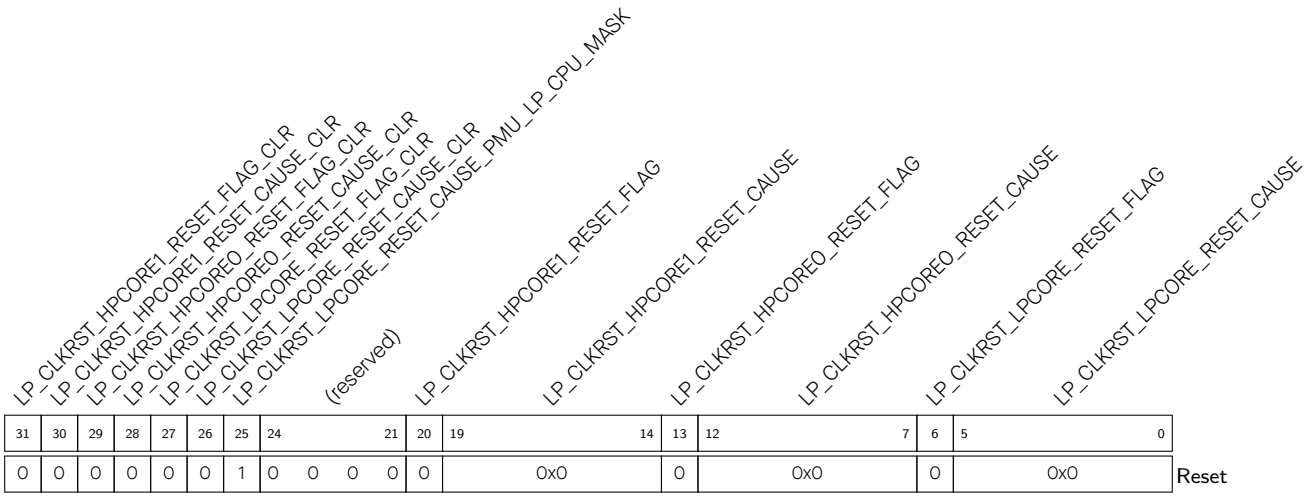
LP_CLKRST_RST_EN_LP_RAM Configures whether to reset LP RAM.

0: Release from reset

1: Reset

(R/W)

Register 11.62. LP_CLKRST_RESET_CAUSE_REG (0x0010)



LP_CLKRST_LPCORE_RESET_CAUSE Represents the reset source of LP CPU.

0x01: Chip reset
0x09: PMU LP Peripheral reset
0x0A: PMU LP CPU reset
0x0F: Brown-out system reset
0x10: RWDT system reset
0x12: Super watchdog reset
0x13: Power glitch reset
0x14: Software LP CPU reset
(RO)

LP_CLKRST_LPCORE_RESET_FLAG Represents the reset flag of LP CPU.

0: Not reset
1: Reset
(RO)

Continued on the next page...

Register 11.62. LP_CLKRST_RESET_CAUSE_REG (0x0010)

Continued from the previous page...

LP_CLKRST_HPCORE0_RESET_CAUSE Represents the reset source of HP CPU0.

0x01: Chip reset
0x03: Software system reset
0x05: PMU core reset
0x07: MWDT core reset
0x09: RWDT core reset
0x0B: MWDT CPU reset
0x0C: Software CPU reset
0x0D: RWDT CPU reset
0x0F: Brown-out system reset
0x10: RWDT system reset
0x12: Super watchdog reset
0x13: Power glitch reset
0x14: eFuse reset
0x16: USB (JTAG) reset
0x17: USB (UART) reset
0x18: JTAG CPU reset
0x1A: Lockup reset
0x1B: PSDET reset
(RO)

LP_CLKRST_HPCORE0_RESET_FLAG Represents the reset flag of HP CPU0.

0: Not reset
1: Reset
(RO)

Continued on the next page...

Register 11.62. LP_CLKRST_RESET_CAUSE_REG (0x0010)

Continued from the previous page...

LP_CLKRST_HPCORE1_RESET_CAUSE Represents the reset source of HP CPU1.

0x01: Chip reset
0x03: Software system reset
0x05: PMU core reset
0x07: MWDT core reset
0x09: RWDT core reset
0x0B: MWDT CPU reset
0x0C: Software CPU reset
0x0D: RWDT CPU reset
0x0F: Brown-out system reset
0x10: RWDT system reset
0x12: Super watchdog reset
0x13: Power glitch reset
0x14: eFuse reset
0x16: USB (JTAG) reset
0x17: USB (UART) reset
0x18: JTAG CPU reset
0x1A: Lockup reset
(RO)

LP_CLKRST_HPCORE1_RESET_FLAG Represents the reset flag of HP CPU1.

0: Not reset
1: Reset
(RO)

LP_CLKRST_LPCORE_RESET_CAUSE_PMU_LP_CPU_MASK Configures whether to mask the reset source code 0x0A of PMU LP CPU reset.

0: Not mask
1: Mask
(R/W)

Continued on the next page...

Register 11.62. LP_CLKRST_RESET_CAUSE_REG (0x0010)

Continued from the previous page...

LP_CLKRST_LPCORE_RESET_CAUSE_CLR Write 1 to clear LP_CLKRST_LPCORE_RESET_CAUSE. (WT)

LP_CLKRST_LPCORE_RESET_FLAG_CLR Write 1 to clear LP_CLKRST_LPCORE_RESET_FLAG. (WT)

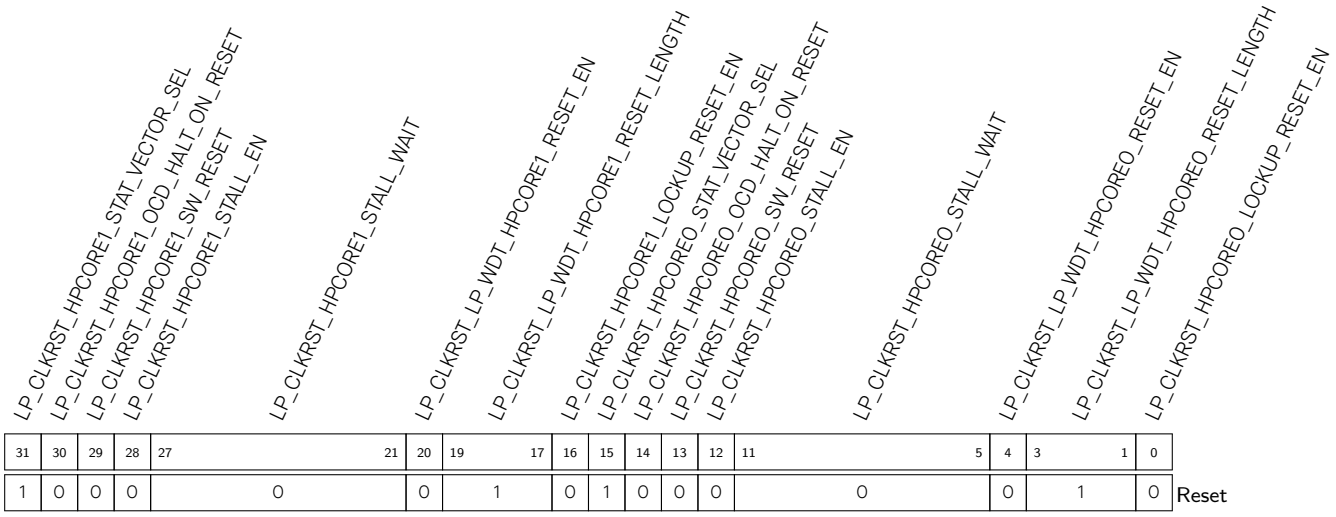
LP_CLKRST_HPCORE0_RESET_CAUSE_CLR Write 1 to clear LP_CLKRST_HPCORE0_RESET_CAUSE. (WT)

LP_CLKRST_HPCORE0_RESET_FLAG_CLR Write 1 to clear LP_CLKRST_HPCORE0_RESET_FLAG. (WT)

LP_CLKRST_HPCORE1_RESET_CAUSE_CLR Write 1 to clear LP_CLKRST_HPCORE1_RESET_CAUSE. (WT)

LP_CLKRST_HPCORE1_RESET_FLAG_CLR Write 1 to clear LP_CLKRST_HPCORE1_RESET_FLAG. (WT)

Register 11.63. LP_CLKRST_HPCPU_RESET_CTRL0_REG (0x0014)



LP_CLKRST_HPCORE0_LOCKUP_RESET_EN Configures whether to enable Lockup to trigger HP CPU0 reset.
0: Disable
1: Enable
(R/W)

LP_CLKRST_LP_WDT_HPCORE0_RESET_LENGTH Configures the duration of the reset when RWDT and software trigger HP CPU0 reset.
Measurement unit: Clock cycles. (R/W)

LP_CLKRST_LP_WDT_HPCORE0_RESET_EN Configures whether to enable RWDT to trigger HP CPU0 reset.
0: Disable
1: Enable
(R/W)

LP_CLKRST_HPCORE0_STALL_WAIT Configures the time HP CPU0 stalls when RWDT and software trigger HP CPU0 reset.
Measurement unit: Clock cycles. (R/W)

LP_CLKRST_HPCORE0_STALL_EN Configures the behavior when RWDT and software trigger HP CPU0 reset.
0: Immediately reset HP CPU0
1: Stall HP CPU0 first
(R/W)

LP_CLKRST_HPCORE0_SW_RESET Write 1 to trigger HP CPU0 software reset. (WT)

LP_CLKRST_HPCORE0_OCD_HALT_ON_RESET Configures whether HP CPU0 halts at the first instruction after released from reset.
0: No halt
1: Halt
(R/W)

Continued on the next page...

Register 11.63. LP_CLKRST_HPCPU_RESET_CTRL0_REG (0x0014)

Continued from the previous page...

LP_CLKRST_HPCORE0_STAT_VECTOR_SEL Configures the address from which the HP CPU0 starts.
0: From LP SPM RAM (0x50108000)
1: From HP SPM ROM (0x4FC00000)
(R/W)

LP_CLKRST_HPCORE1_LOCKUP_RESET_EN Configures whether to enable Lockup to trigger HP CPU1 reset.
0: Disable
1: Enable
(R/W)

LP_CLKRST_LP_WDT_HPCORE1_RESET_LENGTH Configures the duration of the reset when RWDT and software trigger HP CPU1 reset.
Measurement unit: Clock cycles. (R/W)

LP_CLKRST_LP_WDT_HPCORE1_RESET_EN Configures whether to enable RWDT to trigger HP CPU1 reset.
0: Disable
1: Enable
(R/W)

LP_CLKRST_HPCORE1_STALL_WAIT Configures the time HP CPU1 stalls when RWDT and software trigger HP CPU1 reset.
Measurement unit: Clock cycles. (R/W)

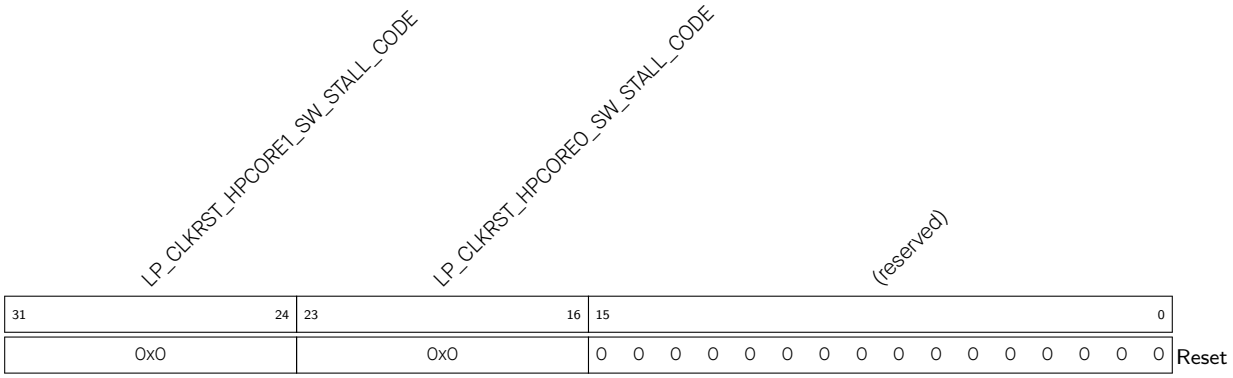
LP_CLKRST_HPCORE1_STALL_EN Configures the behavior when RWDT and software trigger HP CPU1 reset.
0: Immediately reset HP CPU1
1: Stall HP CPU1 first
(R/W)

LP_CLKRST_HPCORE1_SW_RESET Write 1 to trigger HP CPU1 software reset. (WT)

LP_CLKRST_HPCORE1_OCD_HALT_ON_RESET Configures whether HP CPU1 halts at the first instruction after released from reset.
0: No halt
1: Halt
(R/W)

LP_CLKRST_HPCORE1_STAT_VECTOR_SEL Configures the address from which the HP CPU1 starts.
0: From LP SPM RAM (0x50108000)
1: From HP SPM ROM (0x4FC00000)
(R/W)

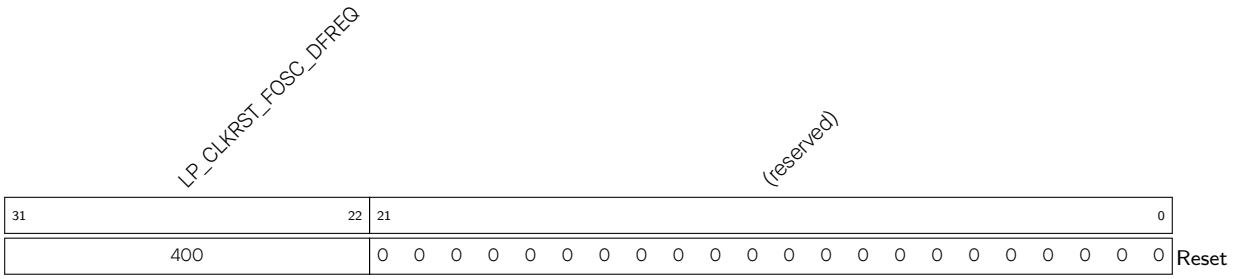
Register 11.64. LP_CLKRST_HPCPU_RESET_CTRL1_REG (0x0018)



LP_CLKRST_HPCORE0_SW_STALL_CODE Configures the threshold for HP CPU0 stall triggered by software. (R/W)

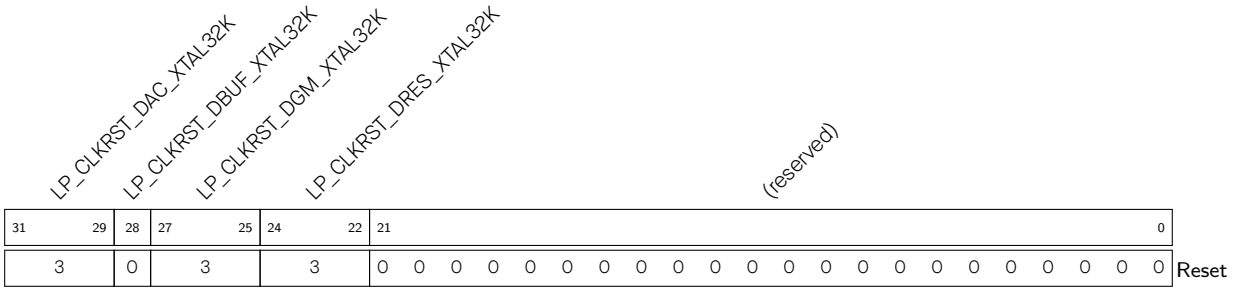
LP_CLKRST_HPCORE1_SW_STALL_CODE Configures the threshold for HP CPU1 stall triggered by software. (R/W)

Register 11.65. LP_CLKRST_FOSC_CNTL_REG (0x001C)

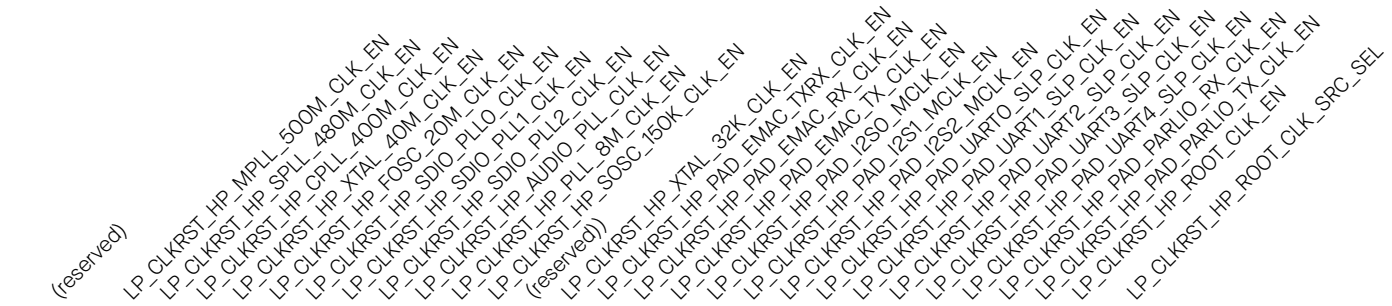


LP_CLKRST_FOSC_DFREQ Configures the frequency of RC_FAST_CLK. (R/W)

Register 11.66. LP_CLKRST_XTAL32K_REG (0x0030)


LP_CLKRST_DAC_XTAL32K
LP_CLKRST_DBUF_XTAL32K
LP_CLKRST_DGM_XTAL32K
LP_CLKRST_DRES_XTAL32K
(reserved)
31 29 28 27 25 24 22 21 0
3 0 3 3 0 0 0

Register 11.67. LP_CLKRST_HP_CLK_CTRL_REG (0x0040)



31 29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1 0
0 0 0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0

Reset

LP_CLKRST_HP_ROOT_CLK_SRC_SEL Configures the clock source of ROOT_CLK.

0: XTAL_CLK

1: CPLL_CLK (360 MHz)

2: RC_FAST_CLK

3: Invalid

(R/W)

LP_CLKRST_HP_ROOT_CLK_EN Configures whether to enable ROOT_CLK.

0: Disable

1: Enable

(R/W)

LP_CLKRST_HP_PAD_PARLIO_TX_CLK_EN Configures whether to enable PAD_PARLIO_TX_CLK from IO pin.

0: Disable

1: Enable

(R/W)

LP_CLKRST_HP_PAD_PARLIO_RX_CLK_EN Configures whether to enable PAD_PARLIO_RX_CLK from IO pin.

0: Disable

1: Enable

(R/W)

LP_CLKRST_HP_PAD_UART4_SLP_CLK_EN Configures whether to enable PAD_UART4_SLP_CLK from IO pin.

0: Disable

1: Enable

(R/W)

LP_CLKRST_HP_PAD_UART3_SLP_CLK_EN Configures whether to enable PAD_UART3_SLP_CLK from IO pin.

0: Disable

1: Enable

(R/W)

Continued on the next page...

Register 11.67. LP_CLKRST_HP_CLK_CTRL_REG (0x0040)

Continued from the previous page...

LP_CLKRST_HP_PAD_UART2_SLP_CLK_EN Configures whether to enable PAD_UART2_SLP_CLK from IO pin.
0: Disable
1: Enable
(R/W)

LP_CLKRST_HP_PAD_UART1_SLP_CLK_EN Configures whether to enable PAD_UART1_SLP_CLK from IO pin.
0: Disable
1: Enable
(R/W)

LP_CLKRST_HP_PAD_UART0_SLP_CLK_EN Configures whether to enable PAD_UART0_SLP_CLK from IO pin.
0: Disable
1: Enable
(R/W)

LP_CLKRST_HP_PAD_I2S2_MCLK_EN Configures whether to enable PAD_I2S2_MCLK from IO pin.
0: Disable
1: Enable
(R/W)

LP_CLKRST_HP_PAD_I2S1_MCLK_EN Configures whether to enable PAD_I2S1_MCLK from IO pin.
0: Disable
1: Enable
(R/W)

LP_CLKRST_HP_PAD_I2S0_MCLK_EN Configures whether to enable PAD_I2S0_MCLK from IO pin.
0: Disable
1: Enable
(R/W)

LP_CLKRST_HP_PAD_EMAC_TX_CLK_EN Configures whether to enable PAD_EMAC_RX_CLK from IO pin.
0: Disable
1: Enable
(R/W)

Continued on the next page...

Register 11.67. LP_CLKRST_HP_CLK_CTRL_REG (0x0040)

Continued from the previous page...

LP_CLKRST_HP_PAD_EMAC_RX_CLK_EN Configures whether to enable PAD_EMAC_TX_CLK from IO pin.
0: Disable
1: Enable
(R/W)

LP_CLKRST_HP_PAD_EMAC_TXRX_CLK_EN Configures whether to enable PAD_EMAC_TXRX_CLK from IO pin.
0: Disable
1: Enable
(R/W)

LP_CLKRST_HP_XTAL_32K_CLK_EN Configures whether to enable XTAL32K_CLK.
0: Disable
1: Enable
(R/W)

LP_CLKRST_HP_SOSC_150K_CLK_EN Configures whether to enable RC_SLOW_CLK.
0: Disable
1: Enable
(R/W)

LP_CLKRST_HP_PLL_8M_CLK_EN Configures whether to enable PLL_LP_CLK.
0: Disable
1: Enable
(R/W)

LP_CLKRST_HP_AUDIO_PLL_CLK_EN Configures whether to enable APLL_CLK.
0: Disable
1: Enable
(R/W)

LP_CLKRST_HP_SDIO_PLL2_CLK_EN Configures whether to enable SDIO_PLL2_CLK.
0: Disable
1: Enable
(R/W)

Continued on the next page...

Register 11.67. LP_CLKRST_HP_CLK_CTRL_REG (0x0040)

Continued from the previous page...

LP_CLKRST_HP_SDIO_PLL1_CLK_EN Configures whether to enable SDIO_PLL1_CLK.

0: Disable

1: Enable

(R/W)

LP_CLKRST_HP_SDIO_PLL0_CLK_EN Configures whether to enable SDIO_PLL0_CLK.

0: Disable

1: Enable

(R/W)

LP_CLKRST_HP_FOSC_20M_CLK_EN Configures whether to enable RC_FAST_CLK.

0: Disable

1: Enable

(R/W)

LP_CLKRST_HP_XTAL_40M_CLK_EN Configures whether to enable XTAL_CLK.

0: Disable

1: Enable

(R/W)

LP_CLKRST_HP_CPLL_400M_CLK_EN Configures whether to enable CPLL_CLK.

0: Disable

1: Enable

(R/W)

LP_CLKRST_HP_SPLL_480M_CLK_EN Configures whether to enable SPLL_CLK.

0: Disable

1: Enable

(R/W)

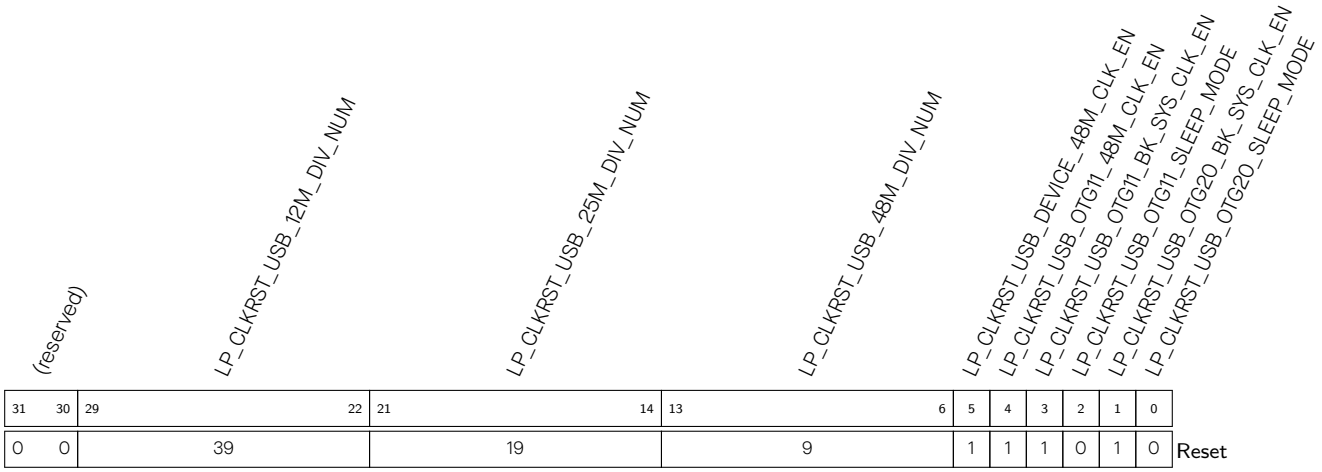
LP_CLKRST_HP_MPLL_500M_CLK_EN Configures whether to enable MPLL_CLK.

0: Disable

1: Enable

(R/W)

Register 11.68. LP_CLKRST_HP_USB_CLKRST_CTRL0_REG (0x0044)



LP_CLKRST_USB_OTG20_SLEEP_MODE Not used. (R/W)

LP_CLKRST_USB_OTG20_BK_SYS_CLK_EN Not used. (R/W)

LP_CLKRST_USB_OTG11_SLEEP_MODE Not used. (R/W)

LP_CLKRST_USB_OTG11_BK_SYS_CLK_EN Not used. (R/W)

LP_CLKRST_USB_OTG11_48M_CLK_EN Configures whether to enable the USB 2.0 Full-Speed OTG PHY clock.
0: Disable
1: Enable
(R/W)

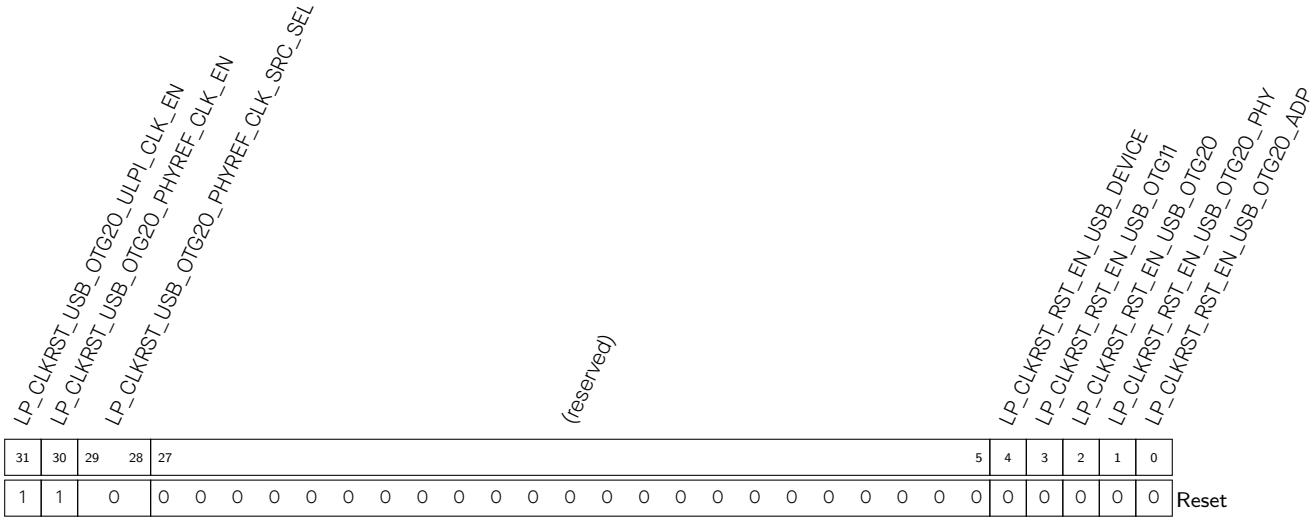
LP_CLKRST_USB_DEVICE_48M_CLK_EN Configures whether to enable the USB Serial/JTAG PHY clock.
0: Disable
1: Enable
(R/W)

LP_CLKRST_USB_48M_DIV_NUM Configures the division number from USB 480M to 25M. (R/W)

LP_CLKRST_USB_25M_DIV_NUM Configures the division number from USB 500M to 25M. (R/W)

LP_CLKRST_USB_12M_DIV_NUM Configures the division number from USB 480M to 12M. (R/W)

Register 11.69. LP_CLKRST_HP_USB_CLKRST_CTRL1_REG (0x0048)



LP_CLKRST_RST_EN_USB_OTG20_ADP Configures whether to reset USB 2.0 High-Speed OTG ADP.

0: Release from reset

1: Reset

(R/W)

LP_CLKRST_RST_EN_USB_OTG20_PHY Configures whether to reset USB 2.0 High-Speed OTG PHY.

0: Release from reset

1: Reset

(R/W)

LP_CLKRST_RST_EN_USB_OTG20 Configures whether to reset USB 2.0 High-Speed OTG.

0: Release from reset

1: Reset

(R/W)

LP_CLKRST_RST_EN_USB_OTG11 Configures whether to reset USB 2.0 Full-Speed OTG.

0: Release from reset

1: Reset

(R/W)

LP_CLKRST_RST_EN_USB_DEVICE Configures whether to reset USB Serial/JTAG.

0: Release from reset

1: Reset

(R/W)

Continued on the next page...

Register 11.69. LP_CLKRST_HP_USB_CLKRST_CTRL1_REG (0x0048)

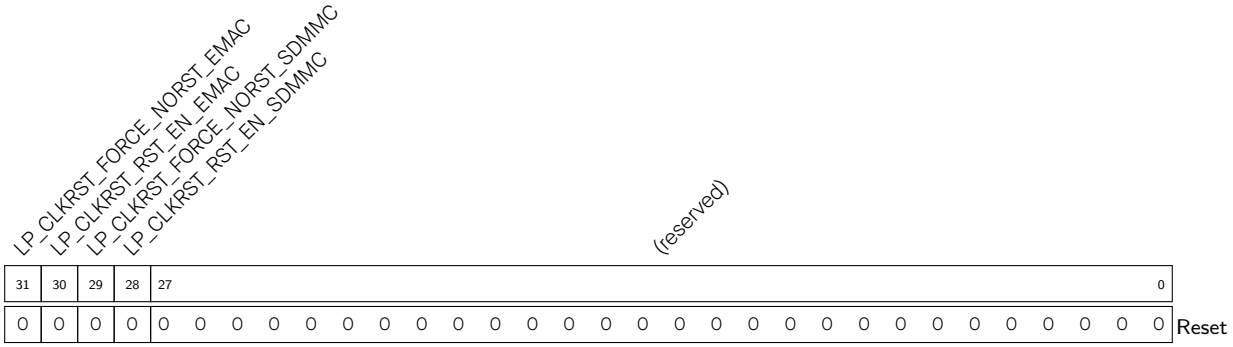
Continued from the previous page...

LP_CLKRST_USB_OTG20_PHYREF_CLK_SRC_SEL Configures the clock source of USB 2.0 High-Speed OTG PHY.
0: PLL_F12M_CLK
1: PLL_F25M_CLK
2: PAD_USB_HSPHY_REFCLK
3: Invalid
(R/W)

LP_CLKRST_USB_OTG20_PHYREF_CLK_EN Configures whether to enable USB 2.0 High-Speed OTG PHY REFCLK.
0: Disable
1: Enable
(R/W)

LP_CLKRST_USB_OTG20_ULPI_CLK_EN Configures whether to enable USB 2.0 High-Speed OTG ULPI clock.
0: Disable
1: Enable
(R/W)

Register 11.70. LP_CLKRST_HP_SDMMC_EMAC_RST_CTRL_REG (0x004C)



LP_CLKRST_RST_EN_SDMMC Configures whether to reset SDMMC.

0: Release from reset

1: Reset

(R/W)

LP_CLKRST_FORCE_NORST_SDMMC Configures whether SDMMC can be reset.

0: Can be reset

1: Force not to be reset

(R/W)

LP_CLKRST_RST_EN_EMAC Configures whether to reset EMAC.

0: Release from reset

1: Reset

(R/W)

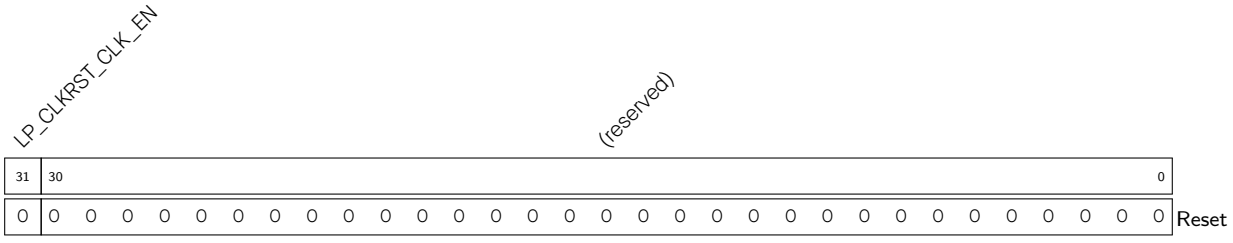
LP_CLKRST_FORCE_NORST_EMAC Configures whether EMAC can be reset.

0: Can be reset

1: Force not to be reset

(R/W)

Register 11.71. LP_CLKRST_DATE_REG (0x03FC)



LP_CLKRST_CLK_EN Configures resister clock gating.

0: Support clock only when application writes registers

1: Force on clock gating for registers

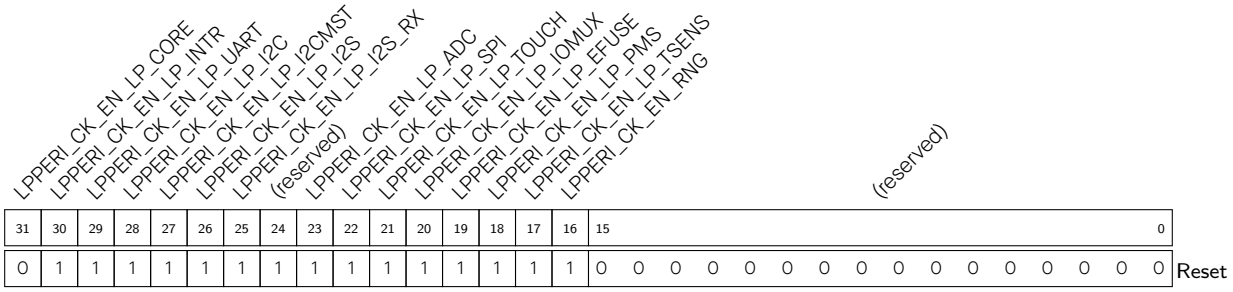
(R/W)

11.5.3 LP Peripheral Clock and Reset (LPPERI) Registers

The addresses in this section are relative to LPPERI base address provided in Table 9.3-2 in Chapter 9 *System and Memory*.

For how to program reserved fields, please refer to Section Programming Reserved Register Field.

Register 11.72. LPPERI_CLK_EN_REG (0x0000)



LPPERI_CK_EN_RNG Configures whether to enable LP RNG clock.

0: Disable

1: Enable

(R/W)

LPPERI_CK_EN_LP_TSENS Configures whether to enable LP temperature sensor clock.

0: Disable

1: Enable

(R/W)

LPPERI_CK_EN_LP_PMS Configures whether to enable LP PMS clock.

0: Disable

1: Enable

(R/W)

LPPERI_CK_EN_LP_EFUSE Configures whether to enable LP eFuse clock.

0: Disable

1: Enable

(R/W)

LPPERI_CK_EN_LP_IOMUX Configures whether to enable LP IO MUX clock.

0: Disable

1: Enable

(R/W)

LPPERI_CK_EN_LP_TOUCH Configures whether to enable LP touch sensor clock.

0: Disable

1: Enable

(R/W)

LPPERI_CK_EN_LP_SPI Configures whether to enable LP SPI clock.

0: Disable

1: Enable

(R/W)

Continued on the next page...

Register 11.72. LPPERI_CLK_EN_REG (0x0000)

Continued from the previous page...

LPPERI_CK_EN_LP_ADC Configures whether to enable LP ADC clock.

0: Disable

1: Enable

(R/W)

LPPERI_CK_EN_LP_I2S_RX Configures whether to enable LP I2S RX clock.

0: Disable

1: Enable

(R/W)

LPPERI_CK_EN_LP_I2S Configures whether to enable LP I2S clock.

0: Disable

1: Enable

(R/W)

LPPERI_CK_EN_LP_I2CMST Configures whether to enable LP I2C master clock.

0: Disable

1: Enable

(R/W)

LPPERI_CK_EN_LP_I2C Configures whether to enable LP I2C clock.

0: Disable

1: Enable

(R/W)

LPPERI_CK_EN_LP_UART Configures whether to enable LP UART clock.

0: Disable

1: Enable

(R/W)

LPPERI_CK_EN_LP_INTR Configures whether to enable LP interrupt matrix clock.

0: Disable

1: Enable

(R/W)

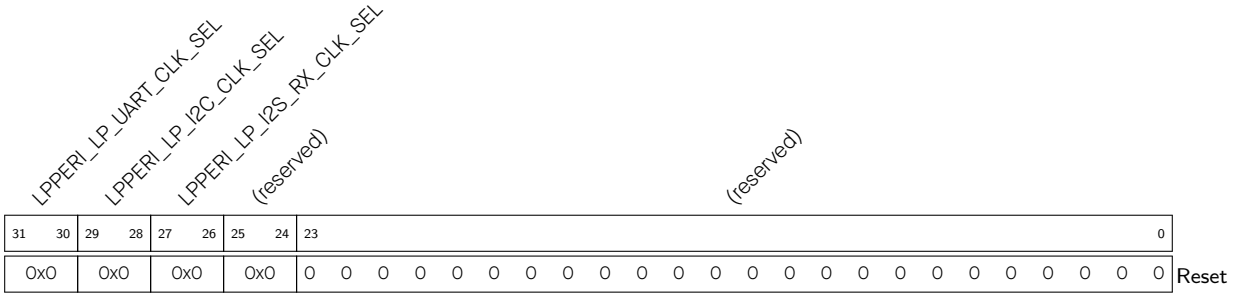
LPPERI_CK_EN_LP_CORE Configures whether to force on LP CPU clock.

0: Controlled by hardware

1: Force on, bypassing hardware control

(R/W)

Register 11.73. LPPERI_CORE_CLK_SEL_REG (0x0004)



LPPERI_LP_I2S_RX_CLK_SEL Configures LP I2S RX clock source.

0: LP_FAST_CLK

1: XTAL_D2_CLK

2: PLL_LP_CLK

3: Invalid

(R/W)

LPPERI_LP_I2C_CLK_SEL Configures LP I2C clock source.

0: LP_FAST_CLK

1: XTAL_D2_CLK

2: PLL_LP_CLK

3: Invalid

(R/W)

LPPERI_LP_UART_CLK_SEL Configures LP UART clock source.

0: RC_FAST_CLK

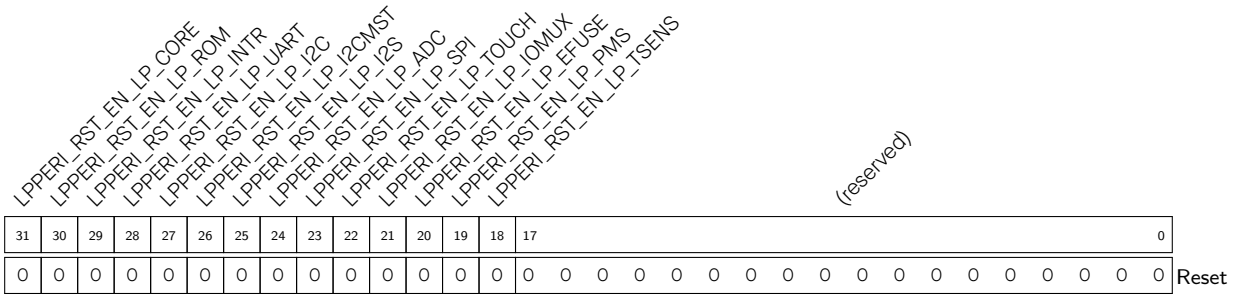
1: XTAL_D2_CLK

2: PLL_LP_CLK

3: Invalid

(R/W)

Register 11.74. LPPERI_RESET_EN_REG (0x0008)



LPPERI_RST_EN_LP_TSENS Configures whether to reset LP temperature sensor.

0: Release from reset

1: Reset

(R/W)

LPPERI_RST_EN_LP_PMS Configures whether to reset LP PMS.

0: Release from reset

1: Reset

(R/W)

LPPERI_RST_EN_LP_EFUSE Configures whether to reset LP eFuse.

0: Release from reset

1: Reset

(R/W)

LPPERI_RST_EN_LP_IOMUX Configures whether to reset LP IO MUX.

0: Release from reset

1: Reset

(R/W)

LPPERI_RST_EN_LP_TOUCH Configures whether to reset LP touch sensor.

0: Release from reset

1: Reset

(R/W)

LPPERI_RST_EN_LP_SPI Configures whether to reset LP SPI.

0: Release from reset

1: Reset

(R/W)

LPPERI_RST_EN_LP_ADC Configures whether to reset LP ADC.

0: Release from reset

1: Reset

(R/W)

Continued on the next page...

Register 11.74. LPPERI_RESET_EN_REG (0x0008)

Continued from the previous page...

LPPERI_RST_EN_LP_I2S Configures whether to reset LP I2S.

0: Release from reset

1: Reset

(R/W)

LPPERI_RST_EN_LP_I2CMST Configures whether to reset LP I2C master.

0: Release from reset

1: Reset

(R/W)

LPPERI_RST_EN_LP_I2C Configures whether to reset LP I2C.

0: Release from reset

1: Reset

(R/W)

LPPERI_RST_EN_LP_UART Configures whether to reset LP UART.

0: Release from reset

1: Reset

(R/W)

LPPERI_RST_EN_LP_INTR Configures whether to reset LP interrupt matrix.

0: Release from reset

1: Reset

(R/W)

LPPERI_RST_EN_LP_ROM Configures whether to reset LP ROM.

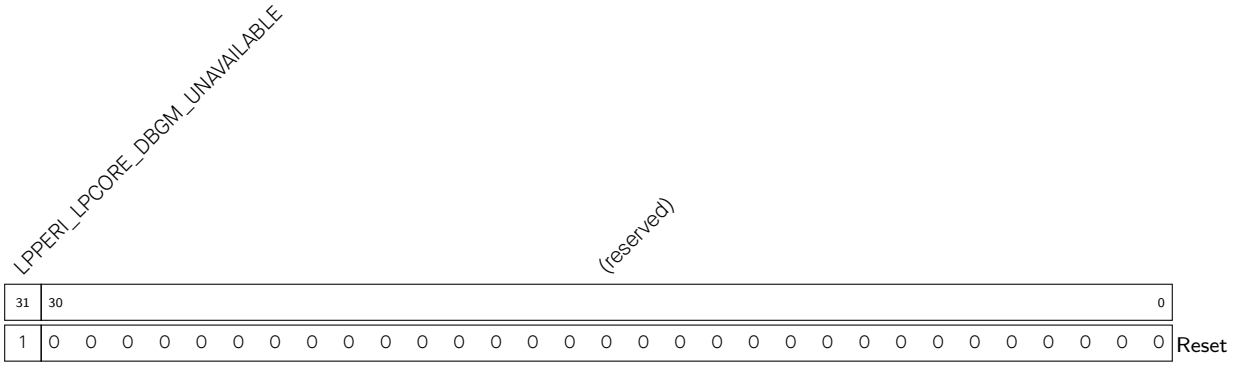
0: Release from reset

1: Reset

(R/W)

LPPERI_RST_EN_LP_CORE Writing 1 triggers LP CPU software reset. (WT)

Register 11.75. LPPERI_CPU_REG (0x000C)



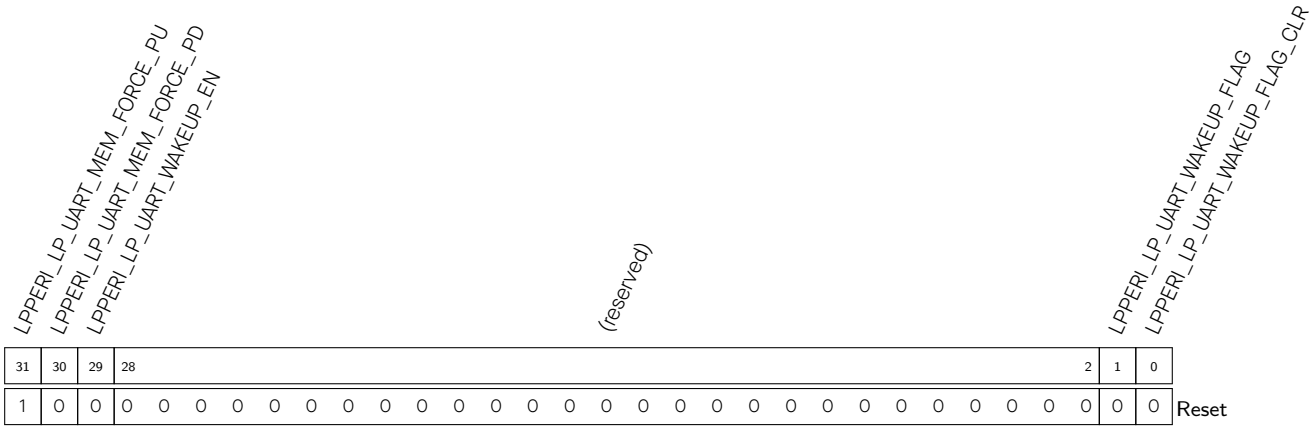
LPPERI_LPCORE_DBGM_UNAVAILABLE Configures whether the LP CPU can be accessed by the HP CPU.

0: Not accessible

1: Accessible

(R/W)

Register 11.76. LPPERI_MEM_CTRL_REG (0x0028)



LPPERI_LP_UART_WAKEUP_FLAG_CLR Write 1 to clear LPPERI_LP_UART_WAKEUP_FLAG. (WT)

LPPERI_LP_UART_WAKEUP_FLAG Represents whether an LP UART wakeup has occurred.

0: No LP UART wakeup occurred

1: LP UART wakeup occurred

(R/WTC/SS)

LPPERI_LP_UART_WAKEUP_EN Configures whether to enable the LP UART wakeup function.

0: Enable

1: Disable

(R/W)

LPPERI_LP_UART_MEM_FORCE_PD Configures whether to force the LP UART memory to power down.

0: Do not force power down

1: Force power down

(R/W)

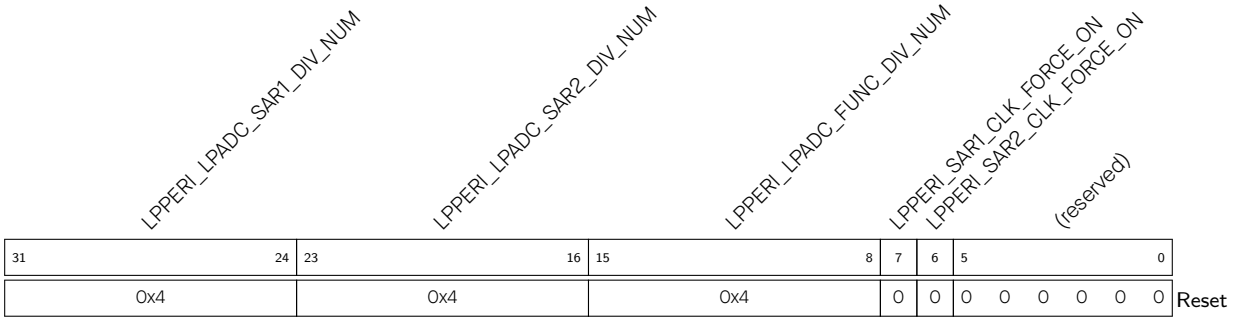
LPPERI_LP_UART_MEM_FORCE_PU Configures whether to force the LP UART memory to power up.

0: Do not force power up

1: Force power up

(R/W)

Register 11.77. LPPERI_ADC_CTRL_REG (0x002C)



LPPERI_SAR2_CLK_FORCE_ON Configures whether to force on the LP ADC SAR1 clock.

0: Controlled by hardware

1: Force on, bypassing hardware control

(R/W)

LPPERI_SAR1_CLK_FORCE_ON Configures whether to force on the LP ADC SAR2 clock.

0: Controlled by hardware

1: Force on, bypassing hardware control

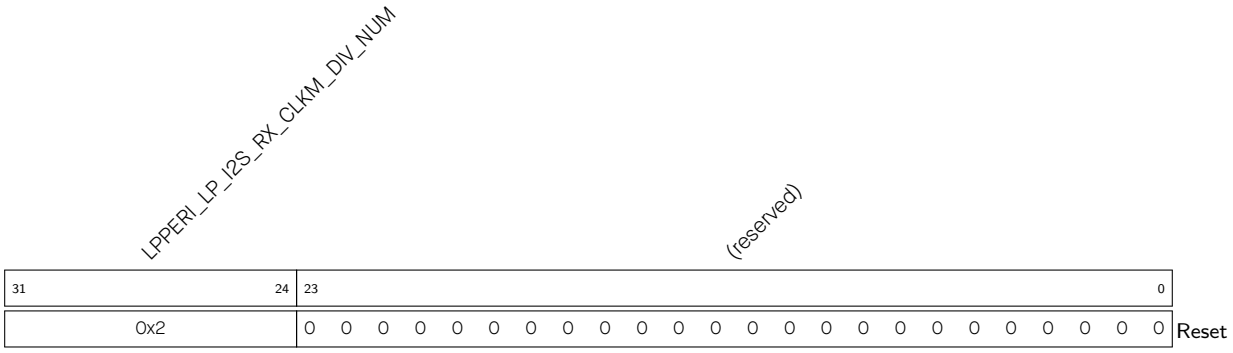
(R/W)

LPPERI_LPADC_FUNC_DIV_NUM Configures the LP ADC function clock divisor. (R/W)

LPPERI_LPADC_SAR2_DIV_NUM Configures the LP ADC SAR1 clock divisor.(R/W)

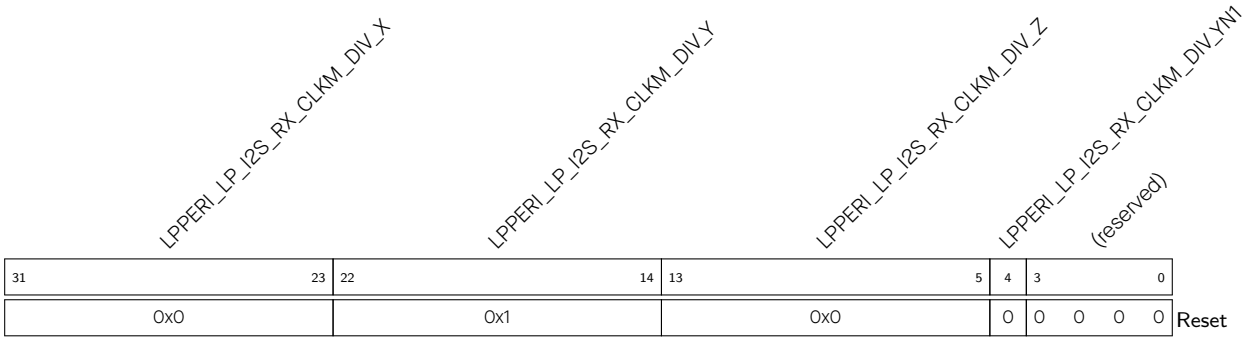
LPPERI_LPADC_SAR1_DIV_NUM Configures the LP ADC SAR2 clock divisor.(R/W)

Register 11.78. LPPERI_LP_I2S_RXCLK_DIV_NUM_REG (0x0030)



LPPERI_LP_I2S_RX_CLKM_DIV_NUM Configures the integer part of the LP I2S RX clock divisor. (R/W)

Register 11.79. LPPERI_LP_I2S_RXCLK_DIV_XYZ_REG (0x0034)



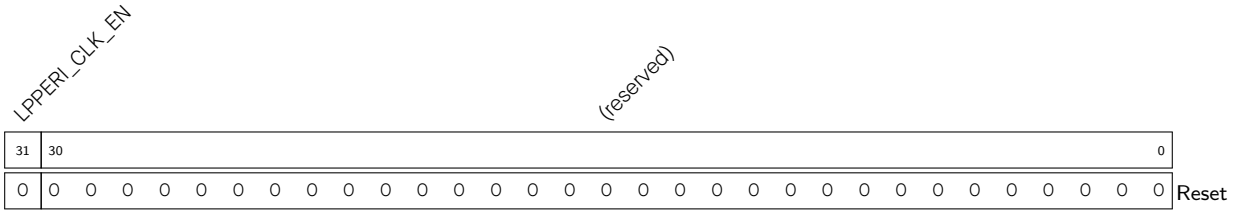
LPPERI_LP_I2S_RX_CLKM_DIV_YN1 Configures the yn1 coefficient of the LP I2S RX clock divisor. (R/W)

LPPERI_LP_I2S_RX_CLKM_DIV_Z Configures the z coefficient of the LP I2S RX clock divisor. (R/W)

LPPERI_LP_I2S_RX_CLKM_DIV_Y Configures the y coefficient of the LP I2S RX clock divisor. (R/W)

LPPERI_LP_I2S_RX_CLKM_DIV_X Configures the x coefficient of the LP I2S RX clock divisor. (R/W)

Register 11.80. LPPERI_DATE_REG (0x03FC)



LPPERI_CLK_EN Configures resister clock gating.

0: Support clock only when application writes registers

1: Force on clock gating for registers

(R/W)

Chapter 12

Chip Boot Control

12.1 Overview

Chip boot process and some chip functions are determined on power-on or hardware reset using strapping pins and eFuse bits. The following functionality can be determined:

- chip boot mode
- enable or disable of ROM messages printing
- source of JTAG signals

ESP32-P4 has five strapping pins:

- GPIO34
- GPIO35
- GPIO36
- GPIO37
- GPIO38

During Chip Reset (see Chapter 11 *Reset and Clock*), hardware samples and stores the voltage level of strapping pins as strapping bit of "0" or "1" in latches, and holds these bits until the chip is powered down. Software can read the latch status (strapping value) from GPIO_STRAPPING.

12.2 Functional Description

This section provides description of the chip functions and the patterns of the strapping pins and eFuse values to invoke each function.

Notice:

Only documented patterns should be used. If an undocumented pattern is used, it may trigger unexpected behaviors.

12.2.1 Default Configuration

By default, GPIO35 is connected to the chip's internal pull-up resistor. If GPIO35 is not connected or is connected to an external high-impedance circuit, the internal weak pull-up determines the default input level of this strapping pin (see Table 12.2-1).

Table 12.2-1. Default Configuration of Strapping Pins

Strapping Pin	Default Configuration
GPIO34	Floating
GPIO35	Pull-up
GPIO36	Floating
GPIO37	Floating
GPIO38	Floating

To change the strapping bit values, users can apply external pull-down/pull-up resistors, or use host MCU GPIOs to control the voltage level of these pins when powering on ESP32-P4. After the reset is released, the strapping pins work as normal-function pins.

12.2.2 Boot Mode Control

The values of GPIO35, GPIO36, GPIO37 and GPIO38 at reset determine the boot mode after the reset is released. Table 12.2-2 shows the strapping pin values of GPIO35, GPIO36, GPIO37 and GPIO38 and the associated boot modes.

Table 12.2-2. Boot Mode Control

Boot Mode	GPIO35	GPIO36	GPIO37	GPIO38
SPI Boot mode (default)	1	x[1]	x	x
Joint Download Boot mode[2]	0	1	x	x
SPI Download Boot mode[3]	0	0	0	1
Invalid Combination[4]	0	0	1	x
	0	0	0	0

[1] x: values that have no effect on the result and can therefore be ignored.

[2] Joint Download Boot mode: Joint Download Boot mode supports the following download methods:

- USB-Serial-JTAG Download Boot
- UART Download Boot
- SPI Slave Download Boot
- USB 2.0 OTG Download Boot

[3] SPI Download Boot mode: GPIO37 and GPIO38 need to be reserved only when using SPI Download Boot mode. GPIO37 and GPIO38 are floating by default and are in a high-impedance state at reset.

[4] Invalid Combination: This combination can trigger unexpected behavior and should be avoided.

In SPI Boot mode, the ROM bootloader loads and executes the program from SPI flash to boot the system. SPI Boot mode can be further classified as follows:

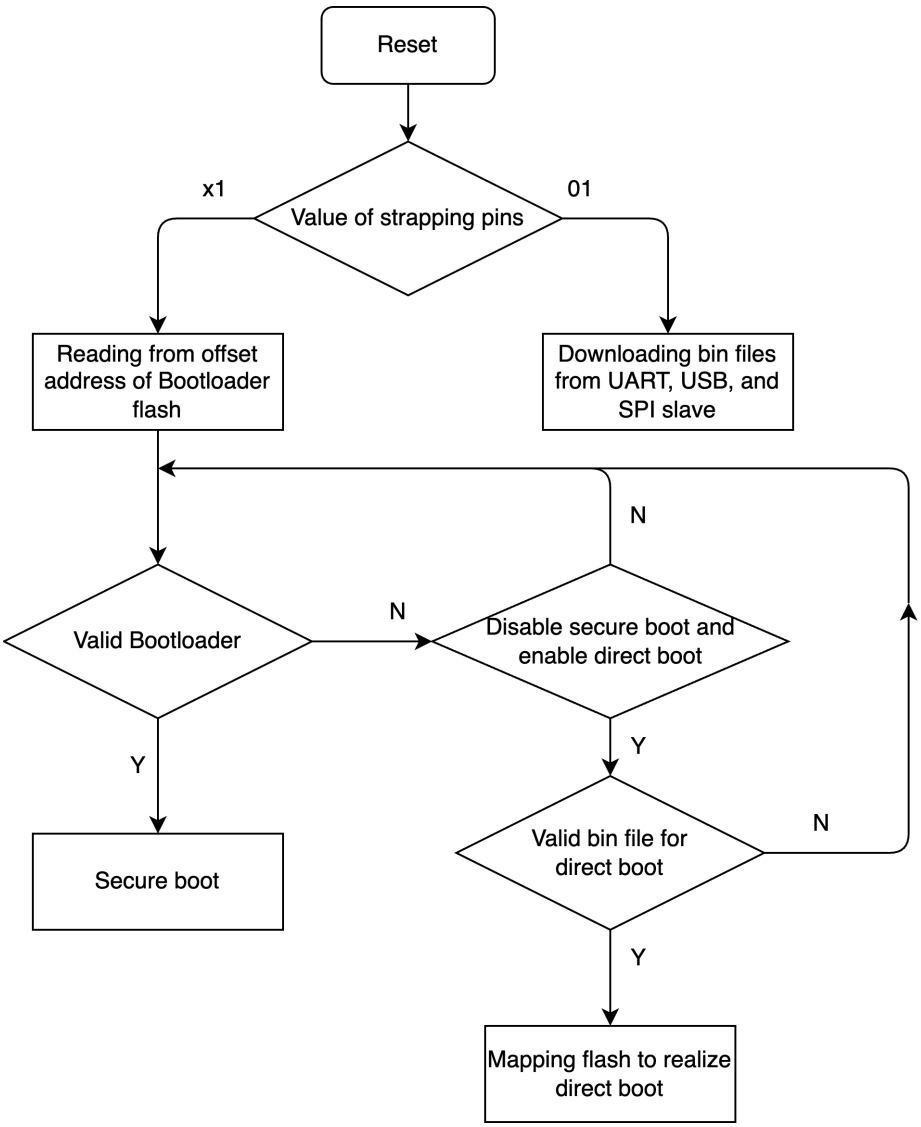
- Normal flash Boot: supports Secure Boot. The ROM bootloader loads the program from flash into L2MEM and executes it. In most practical scenarios, this program is the 2nd stage bootloader, which later boots the target application.

- Direct Boot: does not support Secure Boot and programs run directly from flash. To enable this mode, make sure that the first two words of the bin file downloaded to flash are 0xaedb041d. For more detailed process, see Figure 12.2-1.

In Joint Download Boot mode, users can download binary files into flash using UART0, SPI slave, USB 2.0 OTG or USB Serial/JTAG interface. It is also possible to download binary files into L2MEM and execute it from L2MEM.

In SPI Download Boot mode, users can download binary files into flash using SPI interface. It is also possible to download binary files into L2MEM and execute it from L2MEM.

Figure 12.2-1 shows the detailed boot flow of the chip.



Note: "x1" and "01" are combination values of strapping pins GPIO35 and GPIO36, see Table 12.2-2.

Figure 12.2-1. Chip Boot Flow

The following eFuse bits allows controlling boot mode behaviors:

- EFUSE_DIS_FORCE_DOWNLOAD

  If this eFuse is 0 (default), software can force switch the chip from SPI Boot mode to Joint Download

Boot mode by setting register LPSYSREG_FORCE_DOWNLOAD_BOOT and triggering a CPU reset. If this eFuse is 1, LPSYSREG_FORCE_DOWNLOAD_BOOT is disabled.

- EFUSE_DIS_DOWNLOAD_MODE

  If this eFuse is 1, Joint Download Boot mode is permanently disabled.

- EFUSE_ENABLE_SECURITY_DOWNLOAD

  If this eFuse is 1, Joint Download Boot mode only allows reading, writing, and erasing plaintext flash and does not support any L2MEM or register operations. Ignore this eFuse if Download Boot mode is disabled.

- EFUSE_DIS_DIRECT_BOOT

  If this eFuse is 1, Direct Boot would be disabled in SPI Boot Mode.

USB Serial/JTAG Controller can also force switch the chip to Joint Download Boot mode from SPI Boot mode, and vice versa. For detailed information, please refer to Chapter *54 USB Serial/JTAG Controller (USB_SERIAL_JTAG)*.

You can set eFuse bit EFUSE_DIS_USB_SERIAL_JTAG_DOWNLOAD_MODE to disable USB Serial/JTAG Controller from force switching to Joint Download Boot mode.

12.2.3 ROM Messages Printing Control

During early SPI Boot process, the messages by the ROM code can be printed to:

- **(Default) UART0 and USB Serial/JTAG controller**
- **UART0**
- **USB Serial/JTAG controller**

EFUSE_UART_PRINT_CONTROL and GPIO36 control ROM messages printing to **UART0** as shown in Table 12.2-3 ROM Message Printing Control.

Table 12.2-3. ROM Message Printing Control

eFuse[1]	GPIO36	ROM Code Printing
0	x[2]	Always enabled
1	0	Enabled
	1	Disabled
2	0	Disabled
	1	Enabled
3	x	Always disabled

[1] eFuse: EFUSE_UART_PRINT_CONTROL

[2] x: values that have no effect on the result and can therefore be ignored.

EFUSE_DIS_USB_SERIAL_JTAG_ROM_PRINT controls the printing to **USB Serial/JTAG controller**. When this bit is 1, printing to USB Serial/JTAG controller is disabled. When this bit is 0, and USB Serial/JTAG controller is enabled via EFUSE_DIS_USB_SERIAL_JTAG, ROM messages can be printed to USB Serial/JTAG controller.

Note that if EFUSE_DIS_USB_SERIAL_JTAG_ROM_PRINT is set to 0 to print to USB, but USB Serial/JTAG Controller has been disabled, then ROM messages will not be printed to USB Serial/JTAG Controller.

12.2.4 JTAG Signal Source Control

GPIO34 controls the source of JTAG signals during the early boot process. This GPIO is used together with EFUSE_DIS_PAD_JTAG, EFUSE_DIS_USB_JTAG, and EFUSE_JTAG_SEL_ENABLE. See Table 12.2-4.

Table 12.2-4. JTAG Signal Source Control

eFuse 1[a]	eFuse 2[b]	eFuse 3[c]	GPIO34	Signal Source
0	0	0	x[d]	JTAG signals come from USB Serial/JTAG Controller.
		1	0	JTAG signals come from corresponding pins MTDI, MTCK, MTMS, and MTDO.
			1	JTAG signals come from USB Serial/JTAG Controller.
0	1	x	x	JTAG signals come from corresponding pins MTDI, MTCK, MTMS, and MTDO.
1	0	x	x	JTAG signals come from USB Serial/JTAG Controller.
1	1	x	x	JTAG is disabled.

[a] eFuse 1: EFUSE_DIS_PAD_JTAG

[b] eFuse 2: EFUSE_DIS_USB_JTAG

[c] eFuse 3: EFUSE_JTAG_SEL_ENABLE

[d] x: values that have no effect on the result and can therefore be ignored.

Chapter 13

Interrupt Matrix

13.1 Overview

The interrupt matrix embedded in ESP32-P4 independently routes peripheral interrupt sources to the ESP-RISC-V CPU's peripheral interrupts to timely inform HP CPU0 or HP CPU1 to process the coming interrupts.

Peripheral interrupt sources must be routed to HP CPU0 or HP CPU1 through the interrupt matrix for the following reasons:

- Each HP CPU has 32 interrupt inputs, but the ESP32-P4 provides 128 peripheral interrupt sources. To connect all sources to the CPUs, the interrupt matrix is required.
- The interrupt matrix allows a single peripheral interrupt source to be mapped to any HP CPU0 or HP CPU1 interrupt input, depending on application requirements.

Note:
This chapter focuses on how to map peripheral interrupt sources to HP CPU0/HP CPU1 interrupts. For more details about interrupt configuration, interrupt vectors, and interrupt handling operations recommended by the ISA, please refer to Chapter 2 *High-Performance CPU*.

13.2 Interrupt Terminology in ESP32-P4

The following terms related to interrupts are defined in the context of the *ESP32-P4 Technical Reference Manual* to help readers better understand this document:

13.2.1 Interrupt

An interrupt is an event or condition that temporarily suspends the CPU's current execution to execute an interrupt service routine (ISR). This mechanism enables the CPU to respond immediately to important events.

The *ESP32-P4 Technical Reference Manual* uses the term "interrupt" in a broader sense to refer to both interrupt signal and interrupt source.

13.2.2 Interrupt Signal/Interrupt Source

Both interrupt signal and interrupt source describe the same signal from different points in the system.

From the perspective of peripherals, interrupt signals are generated by the peripheral's internal interrupt sources and are sent to the interrupt matrix.

From the perspective of the interrupt matrix, it receives the interrupt signals sent from peripherals and considers them interrupt sources. The interrupt matrix then outputs CPU peripheral interrupt signals to the CPU.

From the perspective of the CPU, the interrupt signals from the interrupt matrix become interrupt sources and are sent to the CPU core together with the core local interrupt sources.

13.2.3 Interrupt Flow in ESP32-P4

Figure 13.2-1 shows the interrupt flow in ESP32-P4.

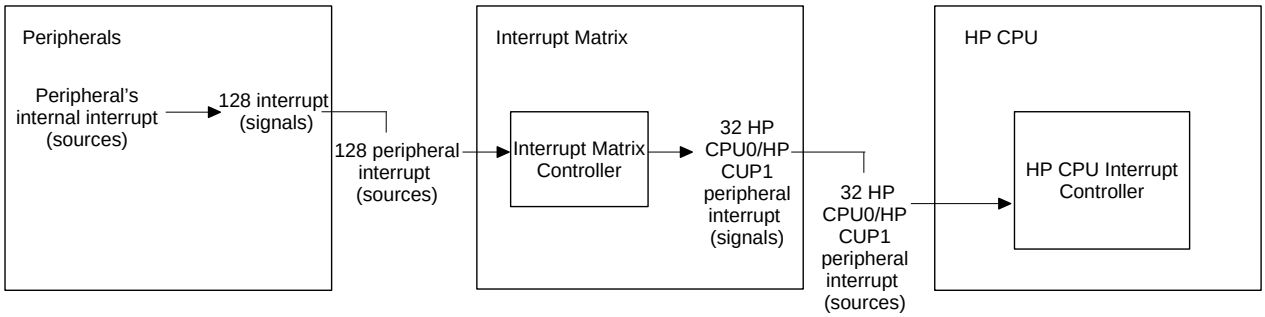


Figure 13.2-1. Interrupt Flow in ESP32-P4

13.3 Features

The interrupt matrix embedded in the ESP32-P4 has the following features:

- 128 peripheral interrupt sources accepted as input
- 32 HP CPU0 peripheral interrupts and 32 HP CPU1 peripheral interrupts generated to HP CPU as output
- Current interrupt status query for peripheral interrupt sources
- Multiple interrupt sources mapped to a single HP CPU0 or HP CPU1 interrupt (i.e., shared interrupts)
- Remapping of HP CPU0 or HP CPU1 User Mode interrupts to Machine Mode interrupts

Figure 13.3-1 shows the structure of the interrupt matrix.

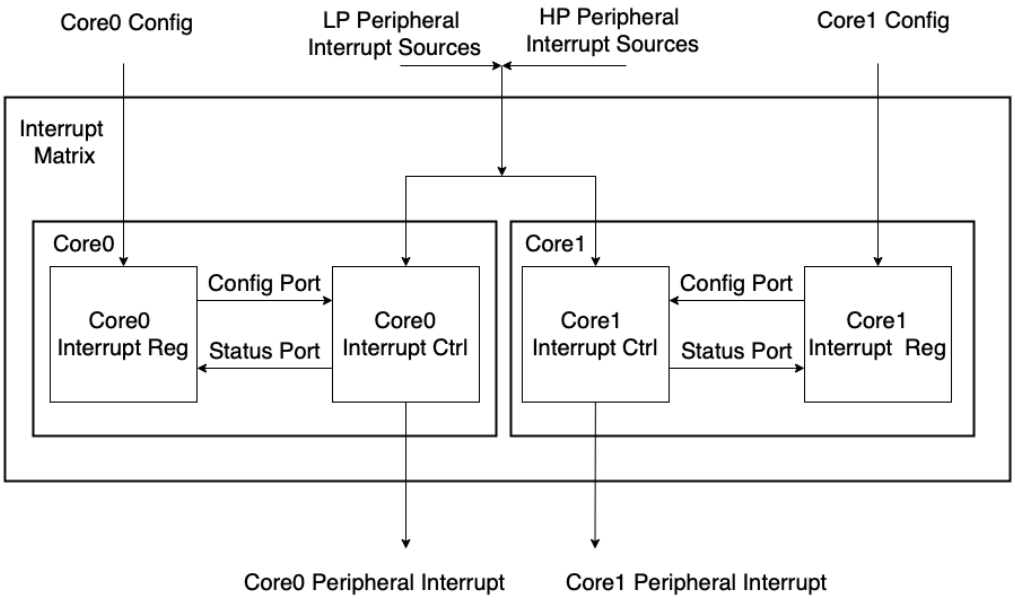


Figure 13.3-1. Interrupt Matrix Structure

All interrupts generated by the peripheral interrupt sources can be handled by HP CPU0 or HP CPU1. Users can configure HP CPU0 interrupt registers ("Core0 Interrupt Reg" module in Figure 13.3-1) to assign peripheral interrupt sources to HP CPU0, or configure HP CPU1 interrupt registers ("Core1 Interrupt Reg" module in Figure 13.3-1) to assign peripheral interrupt sources to HP CPU1. Peripheral interrupt sources can be mapped to both HP CPU0 and HP CPU1 at the same time, allowing both to receive the interrupts.

13.4 Functional Description

13.4.1 Peripheral Interrupt Sources

ESP32-P4 has 128 peripheral interrupt sources in total. Table 13.4-1 lists all these sources and their mapping/status registers.

- Column "No.": Peripheral interrupt source number, which can be 0 ~ 82, 84 ~ 98, 100 ~ 111, 113 ~ 130 (83, 99 and 112 are reserved).
- Column "Chapter": The chapter in which the interrupt source is described in detail.
- Column "Interrupt Source": Name of the peripheral interrupt source.
- Column "Interrupt Source Mapping Register": Registers used for routing the peripheral interrupt sources to HP CPU peripheral interrupts.
- Column "Interrupt Status Register": Registers used for indicating the interrupt status of peripheral interrupt sources.
  - Column "Interrupt Status Register - Bit": Bit position in status register, indicating the interrupt status.
  - Column "Interrupt Status Register - Name": Name of status registers.

The registers in column "Interrupt Source Mapping Register" and column "Interrupt Status Register" correspond to the peripheral interrupt source in column "Interrupt Source". For example, the interrupt source

mapping register for interrupt source RMT_INTR is CORE*x*_RMT_INT_MAP_REG, and its status register is CORE*x*_INTR_STATUS_REG_1_REG.

Note that CORE*x* in the table can be CORE0 (HP CPU0) or CORE1 (HP CPU1).

Table 13.4-1. CPU Peripheral Interrupt Source Mapping/Status Registers and Peripheral Interrupt Sources

No.	Chapter	Interrupt Source	Interrupt Source Mapping Register	Interrupt Status Register	
				Bit	Name
0	*RTC Timer*	LP_RTC_INTR	COREx LP_RTC_INT_MAP_REG	0	COREx_INTR_STATUS_REG_0_REG
1	*Watchdog Timers (WDT)*	LP_WDT_INTR	COREx_LP_WDT_INT_MAP_REG	1	
2	*RTC Timer*	LP_TIMER_REG_0_INTR	COREx_LP_TIMER_REG_0_INT_MAP_REG	2	
3	*RTC Timer*	LP_TIMER_REG_1_INTR	COREx_LP_TIMER_REG_1_INT_MAP_REG	3	
4	*LP Mailbox*	MB_LP_INTR	COREx_MB_LP_INT_MAP_REG	4	
5	*LP Mailbox*	MB_HP_INTR	COREx_MB_LP_INT_MAP_REG	5	
6	*Low-Power Management*	PMU_REG_0_INTR	COREx_PMU_REG_0_INT_MAP_REG	6	
7	*Low-Power Management*	PMU_REG_1_INTR	COREx_PMU_REG_1_INT_MAP_REG	7	
8	*Brown-out Detector*	LP_ANAPERI_INTR	COREx_PMU_REG_0_INT_MAP_REG	8	
9	*ADC Controller (ADC)*	LP_ADC_INTR	COREx_PMU_REG_0_INT_MAP_REG	9	
10	*GPIO Matrix and IO MUX*	LP_GPIO_INTR	COREx_LP_GPIO_INT_MAP_REG	10	
11	*I2C Controller (I2C)*	LP_I2C_INTR	COREx_LP_I2C_INT_MAP_REG	11	
12	*I2S Controller (I2S)*	LP_I2S_INTR	COREx_LP_I2S_INT_MAP_REG	12	
13	*SPI Controller (SPI)*	LP_SPI_INTR	COREx_LP_SPI_INT_MAP_REG	13	
14	*Touch Sensor (TOUCH)*	LP_TOUCH_INTR	COREx_LP_TOUCH_INT_MAP_REG	14	
15	*Temperature Sensor (TSENS)*	LP_TSENS_INTR	COREx_LP_TSENS_INT_MAP_REG	15	
16	*UART Controller (UART)*	LP_UART_INTR	COREx_LP_UART_INT_MAP_REG	16	
17	*eFuse Controller (EFUSE)*	LP_EFUSE_INTR	COREx_LP_EFUSE_INT_MAP_REG	17	
18	*Low-Power CPU*	LP_SW_INTR	COREx_LP_SW_INT_MAP_REG	18	
19	*System Registers (SYSREG)*	LP_SYSREG_INTR	COREx_LP_SYSREG_INT_MAP_REG	19	
20	*Key Manager*	LP_HUK_INTR	COREx_LP_HUK_INT_MAP_REG	20	
21	*System Registers (SYSREG)*	SYS_ICM_INTR	COREx_SYS_ICM_INT_MAP_REG	21	
22	*USB Serial/JTAG Controller (USB_SERIAL_JTAG)*	USB_DEVICE_INTR	COREx_USB_DEVICE_INT_MAP_REG	22	
23	*SD/MMC Host Controller (SDHOST)*	SDIO_HOST_INTR	COREx_SDIO_HOST_INT_MAP_REG	23	
24	*VDMA Controller (VDMA)*	GDMA_INTR	COREx_GDMA_INT_MAP_REG	24	
25	*SPI Controller (SPI)*	SPI2_INTR	COREx_SPI2_INT_MAP_REG	25	
26	*SPI Controller (SPI)*	SPI3_INTR	COREx_SPI3_INT_MAP_REG	26	
27	*I2S Controller (I2S)*	I2S0_INTR	COREx_I2S0_INT_MAP_REG	27	
28	*I2S Controller (I2S)*	I2S1_INTR	COREx_I2S1_INT_MAP_REG	28	
29	*I2S Controller (I2S)*	I2S2_INTR_0	COREx_I2S2_INT_MAP_REG	29	
30	*UART Controller (UART)*	UHCI0_INTR	COREx_UHCI0_INT_MAP_REG	30	
31	*UART Controller (UART)*	UART0_INTR	COREx_UART0_INT_MAP_REG	31	

No.	Chapter	Interrupt Source	Interrupt Source Mapping Register	Interrupt Status Register	
				Bit	Name
32	*UART Controller (UART)*	UART1_INTR	CORE*x*_UART1_INT_MAP_REG	0	CORE*x*_INTR_STATUS_REG_1_REG
33	*UART Controller (UART)*	UART2_INTR	CORE*x*_UART2_INT_MAP_REG	1	
34	*UART Controller (UART)*	UART3_INTR	CORE*x*_UART3_INT_MAP_REG	2	
35	*UART Controller (UART)*	UART4_INTR	CORE*x*_UART4_INT_MAP_REG	3	
36	*LCD and Camera Controller (LCD_CAM)*	LCD_CAM_INTR	CORE*x*_LCD_CAM_INT_MAP_REG	4	
37	*ADC Controller (ADC)*	ADC_INTR	CORE*x*_ADC_INT_MAP_REG	5	
38	*Motor Control PWM (MCPWM)*	PWM0_INTR	CORE*x*_PWM0_INT_MAP_REG	6	
39	*Motor Control PWM (MCPWM)*	PWM1_INTR	CORE*x*_PWM1_INT_MAP_REG	7	
40	*Two-Wire Automotive Interface (TWAI)*	TWAI0_INTR	CORE*x*_TWAI0_INT_MAP_REG	8	
41	*Two-Wire Automotive Interface (TWAI)*	TWAI1_INTR	CORE*x*_TWAI1_INT_MAP_REG	9	
42	*Two-Wire Automotive Interface (TWAI)*	TWAI2_INTR	CORE*x*_TWAI2_INT_MAP_REG	10	
43	*Remote Control Peripheral (RMT)*	RMT_INTR	CORE*x*_RMT_INT_MAP_REG	11	
44	*I2C Controller (I2C)*	I2C0_INTR	CORE*x*_I2C0_INT_MAP_REG	12	
45	*I2C Controller (I2C)*	I2C1_INTR	CORE*x*_I2C1_INT_MAP_REG	13	
46	*Timer Group (TIMG)*	TIMERGRP0_T0_INTR	CORE*x*_TIMERTGRP0_T0_INT_MAP_REG	14	
47	*Timer Group (TIMG)*	TIMERGRP0_T1_INTR	CORE*x*_TIMERGRP0_T1_INT_MAP_REG	15	
48	*Timer Group (TIMG)*	TIMERGRP0_WDT_INTR	CORE*x*_TIMERGRP0_WDT_INT_MAP_REG	16	
49	*Timer Group (TIMG)*	TIMERGRP1_T0_INTR	CORE*x*_TIMRTGRP1_T0_INT_MAP_REG	17	
50	*Timer Group (TIMG)*	TIMERGRP1_T1_INTR	CORE*x*_TIMERGRP1_T1_INT_MAP_REG	18	
51	*Timer Group (TIMG)*	TIMERGRP1_WDT_INTR	CORE*x*_TIMERGRP1_WDT_INT_MAP_REG	19	
52	*LED PWM Controller (LEDC)*	LEDC_INTR	CORE*x*_LEDC_INT_MAP_REG	20	
53	*System Timer*	SYSTIMER_TARGET0_INTR	CORE*x*_SYSTIMER_TARGET0_INT_MAP_REG	21	
54	*System Timer*	SYSTIMER_TARGET1_INTR	CORE*x*_SYSTIMER_TARGET1_INT_MAP_REG	22	
55	*System Timer*	SYSTIMER_TARGET2_INTR	CORE*x*_SYSTIMER_TARGET2_INT_MAP_REG	23	
56	*GDMA Controller (GDMA-AHB, GDMA-AXI)*	AHB_PDMA_IN_CH0_INTR	CORE*x*_AHB_PDMA_IN_CH0_INT_MAP_REG	24	
57	*GDMA Controller (GDMA-AHB, GDMA-AXI)*	AHB_PDMA_IN_CH1_INTR	CORE*x*_AHB_PDMA_IN_CH1_INT_MAP_REG	25	
58	*GDMA Controller (GDMA-AHB, GDMA-AXI)*	AHB_PDMA_IN_CH2_INTR	CORE*x*_AHB_PDMA_IN_CH2_INT_MAP_REG	26	
59	*GDMA Controller (GDMA-AHB, GDMA-AXI)*	AHB_PDMA_OUT_CH0_INTR	CORE*x*_AHB_PDMA_OUT_CH0_INT_MAP_REG	27	
60	*GDMA Controller (GDMA-AHB, GDMA-AXI)*	AHB_PDMA_OUT_CH1_INTR	CORE*x*_AHB_PDMA_OUT_CH1_INT_MAP_REG	28	
61	*GDMA Controller (GDMA-AHB, GDMA-AXI)*	AHB_PDMA_OUT_CH2_INTR	CORE*x*_AHB_PDMA_OUT_CH2_INT_MAP_REG	29	
62	*GDMA Controller (GDMA-AHB, GDMA-AXI)*	AXI_PDMA_IN_CH0_INTR	CORE*x*_AXI_PDMA_IN_CH0_INT_MAP_REG	30	
63	*GDMA Controller (GDMA-AHB, GDMA-AXI)*	AXI_PDMA_IN_CH1_INTR	CORE*x*_AXI_PDMA_IN_CH1_INT_MAP_REG	31	

GoBack

No.	Chapter	Interrupt Source	Interrupt Source Mapping Register	Interrupt Status Register	
				Bit	Name
64	*GDMA Controller (GDMA-AHB, GDMA-AXI)*	AXI_PDMA_IN_CH2_INTR	COREx_AXI_PDMA_IN_CH2_INT_MAP_REG	0	
65	*GDMA Controller (GDMA-AHB, GDMA-AXI)*	AXI_PDMA_OUT_CH0_INTR	COREx_AXI_PDMA_OUT_CH0_INT_MAP_REG	1	
66	*GDMA Controller (GDMA-AHB, GDMA-AXI)*	AXI_PDMA_OUT_CH1_INTR	COREx_AXI_PDMA_OUT_CH1_INT_MAP_REG	2	
67	*GDMA Controller (GDMA-AHB, GDMA-AXI)*	AXI_PDMA_OUT_CH2_INTR	COREx_AXI_PDMA_OUT_CH2_INT_MAP_REG	3	
68	*RSA Accelerator (RSA)*	RSA_INTR	COREx_RSA_INT_MAP_REG	4	
69	*AES Accelerator (AES)*	AES_INTR	COREx_AES_INT_MAP_REG	5	
70	*SHA Accelerator (SHA)*	SHA_INTR	COREx_SHA_INT_MAP_REG	6	COREx_INTR_STATUS_REG_2_REG
71	*SHA Accelerator (SHA)*	ECC_INTR	COREx_ECC_INT_MAP_REG	7	
72	*ECDSA Digital Signature Peripheral (ECDSA_DS)*	ECDSA_INTR	COREx_ECDSA_INT_MAP_REG	8	
73	*Key Manager*	KM_INTR	COREx_KM_INT_MAP_REG	9	
74	*GPIO Matrix and IO MUX*	GPIO_INTR0	COREx_GPIO_INTR0_MAP_REG	10	
75	*GPIO Matrix and IO MUX*	GPIO_INTR1	COREx_GPIO_INTR1_MAP_REG	11	
76	*GPIO Matrix and IO MUX*	GPIO_INTR2	COREx_GPIO_INTR2_MAP_REG	12	
77	*GPIO Matrix and IO MUX*	GPIO_INTR3	COREx_GPIO_INTR3_MAP_REG	13	
78	*GPIO Matrix and IO MUX*	GPIO_PAD_COMP_INTR	COREx_GPIO_PAD_COMP_INT_MAP_REG	14	
79	*System Registers (SYSREG)*	CPU_INTR_FROM_CPU_0	COREx_CPU_INTR_FROM_CPU_0_MAP_REG	15	
80	*System Registers (SYSREG)*	CPU_INTR_FROM_CPU_1	COREx_CPU_INTR_FROM_CPU_1_MAP_REG	16	
81	*System Registers (SYSREG)*	CPU_INTR_FROM_CPU_2	COREx_CPU_INTR_FROM_CPU_2_MAP_REG	17	
82	*System Registers (SYSREG)*	CPU_INTR_FROM_CPU_3	COREx_CPU_INTR_FROM_CPU_3_MAP_REG	18	
83	n/a	reserved	reserved	19	
84	*SPI Controller (SPI)*	FLASH_MSPI_INTR	COREx_FLASH_MSPI_INT_MAP_REG	20	
85	*MIPI CSI*	CSI_BRIDGE_INTR	COREx_CSI_BRIDGE_INT_MAP_REG	21	
86	*MIPI DSI*	DSI_BRIDGE_INTR	COREx_DSI_BRIDGE_INT_MAP_REG	22	
87	*MIPI CSI*	CSI_INTR	COREx_CSI_INT_MAP_REG	23	
88	*MIPI DSI*	DSI_INTR	COREx_DSI_INT_MAP_REG	24	
89	*Ethernet Media Access Controller (EMAC)*	GMII_PHY_INTR	COREx_GMII_PHY_INT_MAP_REG	25	
90	*Ethernet Media Access Controller (EMAC)*	LPI_INTR	COREx_LPI_INT_MAP_REG	26	
91	*Ethernet Media Access Controller (EMAC)*	PMT_INTR	COREx_PMT_INT_MAP_REG	27	
92	*Ethernet Media Access Controller (EMAC)*	ETH_MAC_INTR	COREx_ETH_MAC_INT_MAP_REG	28	
93	*USB 2.0 High-Speed OTG*	USB_OTG_INTR	COREx_USB_OTG_INT_MAP_REG	29	
94	*USB 2.0 High-Speed OTG*	USB_OTG_ENDP_MULTI_PROC_INTR	COREx_USB_OTG_ENDP_MULTI_PROC_INT_MAP_REG	30	
95	*JPEG Codec*	JPEG_INTR	COREx_JPEG_INT_MAP_REG	31	

GoBack

No.	Chapter	Interrupt Source	Interrupt Source Mapping Register	Interrupt Status Register	
				Bit	Name
96	*Pixel-Processing Accelerator (PPA)*	PPA_INTR	COREx_PPA_INT_MAP_REG	0	
97	*RISC-V Trace Encoder (TRACE)*	CORE0_TRACE_INTR	COREx_CORE0_TRACE_INT_MAP_REG	1	
98	*RISC-V Trace Encoder (TRACE)*	CORE1_TRACE_INTR	COREx_CORE1_TRACE_INT_MAP_REG	2	
99	n/a	reserved	reserved	3	
100	*Image Signal Processor (ISP)*	ISP_INTR	COREx_ISP_INT_MAP_REG	4	
101	*I3C Controller [to be added later]*	I3C_MST_INTR	COREx_I3C_MST_INT_MAP_REG	5	
102	*I3C Controller [to be added later]*	I3C_SLV_INTR	COREx_I3C_SLV_INT_MAP_REG	6	
103	*USB 2.0 Full-Speed OTG*	USB_OTG11_INTR	COREx_USB_OTG11_INT_MAP_REG	7	
104	*2D-DMA Controller (2D-DMA)*	DMA2D_IN_CH0_INTR	COREx_DMA2D_IN_CH0_INT_MAP_REG	8	
105	*2D-DMA Controller (2D-DMA)*	DMA2D_IN_CH1_INTR	COREx_DMA2D_IN_CH1_INT_MAP_REG	9	
106	*2D-DMA Controller (2D-DMA)*	DMA2D_OUT_CH0_INTR	COREx_DMA2D_OUT_CH0_INT_MAP_REG	10	
107	*2D-DMA Controller (2D-DMA)*	DMA2D_OUT_CH1_INTR	COREx_DMA2D_OUT_CH1_INT_MAP_REG	11	
108	*2D-DMA Controller (2D-DMA)*	DMA2D_OUT_CH2_INTR	COREx_DMA2D_OUT_CH2_INT_MAP_REG	12	
109	*SPI Controller (SPI)*	PSRAM_MSPI_INTR	COREx_PSRAM_MSPI_INT_MAP_REG	13	
110	*System Registers (SYSREG)*	HP_SYSREG_INTR	COREx_HP_SYSREG_INT_MAP_REG	14	
111	*Pulse Count Controller (PCNT)*	PCNT_INTR	COREx_PCNT_INT_MAP_REG	15	COREx_INTR_STATUS_REG_3_REG
112	n/a	reserved	reserved	16	
113	*Parallel IO Controller (PARLIO)*	HP_PARLIO_RX_INTR	COREx_HP_PARLIO_RX_INT_MAP_REG	17	
114	*Parallel IO Controller (PARLIO)*	HP_PARLIO_TX_INTR	COREx_HP_PARLIO_TX_INT_MAP_REG	18	
115	*H264 Encoder*	H264_DMA_OUT_CH0_INTR	COREx_H264_DMA_OUT_CH0_INT_MAP_REG	19	
116	*H264 Encoder*	H264_DMA_OUT_CH1_INTR	COREx_H264_DMA_OUT_CH1_INT_MAP_REG	20	
117	*H264 Encoder*	H264_DMA_OUT_CH2_INTR	COREx_H264_DMA_OUT_CH2_INT_MAP_REG	21	
118	*H264 Encoder*	H264_DMA_OUT_CH3_INTR	COREx_H264_DMA_OUT_CH3_INT_MAP_REG	22	
119	*H264 Encoder*	H264_DMA_OUT_CH4_INTR	COREx_H264_DMA_OUT_CH4_INT_MAP_REG	23	
120	*H264 Encoder*	H264_DMA_IN_CH0_INTR	COREx_H264_DMA_IN_CH0_INT_MAP_REG	24	
121	*H264 Encoder*	H264_DMA_IN_CH1_INTR	COREx_H264_DMA_IN_CH1_INT_MAP_REG	25	
122	*H264 Encoder*	H264_DMA_IN_CH2_INTR	COREx_H264_DMA_IN_CH2_INT_MAP_REG	26	
123	*H264 Encoder*	H264_DMA_IN_CH3_INTR	COREx_H264_DMA_IN_CH3_INT_MAP_REG	27	
124	*H264 Encoder*	H264_DMA_IN_CH4_INTR	COREx_H264_DMA_IN_CH4_INT_MAP_REG	28	
125	*H264 Encoder*	H264_DMA_IN_CH5_INTR	COREx_H264_DMA_IN_CH5_INT_MAP_REG	29	
126	*H264 Encoder*	H264_REG_INTR	COREx_H264_REG_INT_MAP_REG	30	
127	*Debug Assistant*	ASSIST_DEBUG_INTR	COREx_ASSIST_DEBUG_INT_MAP_REG	31	

No.	Chapter	Interrupt Source	Interrupt Source Mapping Register	Interrupt Status Register	
				Bit	Name
128	*2D-DMA Controller (2D-DMA)*	DMA2D_IN_CH2_INTR	COREx_DMA2D_IN_CH2_INT_MAP_REG	0	COREx_INTR_STATUS_REG_4_REG
129	*2D-DMA Controller (2D-DMA)*	DMA2D_OUT_CH3_INTR	COREx_DMA2D_OUT_CH3_INT_MAP_REG	1	
130	*AXI Performance Monitor*	AXI_PERF_MON_INTR	COREx_AXI_PERF_MON_INT_MAP_REG	3	

13.4.2 Assign Peripheral Interrupt Source to HP CPU Interrupt

In this section, the following terms are used to describe the operation of the interrupt matrix.

- *SOURCE*: Stands for a peripheral interrupt source in Table 13.4-1.
- CORE*x*_*SOURCE*_MAP: Stands for an interrupt source mapping register for the peripheral interrupt source (*SOURCE*) of HP CPU*x*.
- Num_P: The index of HP CPU interrupts, which can be 16 ~ 47.
- Interrupt_P: Stands for the HP CPU interrupt numbered as Num_P.

13.4.2.1 Assign One Peripheral Interrupt Source (*SOURCE*) to HP CPU*x*

Setting the corresponding source mapping register CORE*x*_*SOURCE*_MAP of *SOURCE* to Num_P assigns this interrupt source to Interrupt_P of HP CPU*x*.

13.4.2.2 Assign Multiple Peripheral Interrupt Sources (*SOURCE*) to HP CPU*x*

Setting the corresponding source mapping register CORE*x*_*SOURCE*_MAP of each interrupt source to the same Num_P assigns multiple sources to the same Interrupt_P of HP CPU*x*. Any of these sources can trigger CPU Interrupt_P of HP CPU*x*. In other words, the CPU Interrupt_P will be shared between these interrupt sources. When an interrupt signal is generated, HP CPU*x* should check the interrupt status registers to determine which peripheral generated the interrupt. For more information, see Chapter 2 *High-Performance CPU*.

13.4.2.3 Disable HP CPU*x* Peripheral Interrupt Source (*SOURCE*)

Writing 0 to the CORE*x*_*SOURCE*_MAP register disables the corresponding interrupt source.

13.4.3 Query Current Interrupt Status of HP CPU*x* Peripheral Interrupt Source

After enabling peripheral interrupt sources, users can query the current interrupt status of HP CPU*x* peripheral interrupt sources by reading the bit value in CORE*x*_INTR_STATUS_*n* (read only). For the mapping between CORE*x*_INTR_STATUS_*n* and peripheral interrupt sources, please refer to Table 13.4-1.

13.4.4 Interrupt Remapping

The RISC-V HP CPU*x* (x can be 0 or 1) of ESP32-P4 supports Machine Mode and User Mode.

HP CPU*x* provides a total of 32 peripheral interrupts. Each peripheral interrupt can be delegated by HP CPU*x* as either a Machine-mode interrupt or a User-mode interrupt.

Under normal conditions, when HP CPU*x* is running in Machine Mode, it can respond only to Machine-mode interrupts and cannot respond to User-mode interrupts. When HP CPU*x* is running in User Mode, it can respond to both Machine-mode and User-mode interrupts.

To allow HP CPU*x* to indirectly respond to User-mode interrupts while operating in Machine Mode, the system introduces the Interrupt Remap mechanism.

Interrupt remapping allows software to redirect HP CPU*x* interrupts originally assigned to User Mode to a single specified Machine-mode interrupt. In this way, HP CPU*x* can indirectly respond to User-mode interrupts while running in Machine Mode.

Interrupt remapping features:

- Enable interrupt remapping: Each peripheral interrupt source can be independently configured to participate in interrupt remapping. For example, setting CORE*x*_*SOURCE*_SRC_PASS_IN_SEC to 1 enables interrupt remapping for interrupt matrix source *SOURCE*.
- Specify the target Machine-mode interrupt: All remapped User-mode interrupt signals are uniformly mapped to the same specified Machine-mode interrupt. For example, writing a value n to CORE*x*_INTR_SIG_IDX_ASSERT_IN_SEC_REG selects the Machine-mode interrupt with interrupt number n as the remapping target.
- Access restrictions for remapping-related registers: the following registers and fields related to interrupt remapping are writable only when HP CPU*x* is operating in Machine Mode:
  - CORE*x*_INTR_SIG_IDX_ASSERT_IN_SEC_REG
  - CORE*x*_*SOURCE*_SRC_PASS_IN_SEC
  - CORE*x*_*SOURCE*_SRC_IN_SEC_FLAG
  - When CORE*x*_*SOURCE*_SRC_IN_SEC_FLAG is set to 1, CORE*x*_*SOURCE*_MAP is also writable only in Machine Mode.

When interrupt source *SOURCE* has been mapped to the peripheral interrupt signal *n* of CORE*x*, interrupt remapping for source *SOURCE* occurs only when all of the following conditions are met:

- CORE*x*_*SOURCE*_SRC_PASS_IN_SEC is set to 1
- CORE*x* is currently operating in Machine Mode
- The peripheral interrupt signal *n* of CORE*x* is internally delegated to User Mode by the CPU
- The target interrupt signal specified by CORE*x*_INTR_SIG_IDX_ASSERT_IN_SEC_REG is internally delegated to Machine Mode by the CPU

13.5 Register Summary

The addresses in this section are relative to the interrupt matrix HP CPU0/HP CPU1 base address provided in Table 9.3-2 in Chapter 9 *System and Memory*.

The abbreviations given in Column **Access** are explained in Section *Access Types for Registers*.

13.5.1 HP CPU0 Interrupt Matrix Register Summary

Name	Description	Address	Access
COREO_LP_RTC_INT_MAP_REG	LP_RTC_INTR mapping register	0x0000	R/W
COREO_LP_WDT_INT_MAP_REG	LP_WDT_INTR mapping register	0x0004	R/W
COREO_LP_TIMER_REG_0_INT_MAP_REG	LP_TIMER_REG_0_INTR mapping register	0x0008	R/W
COREO_LP_TIMER_REG_1_INT_MAP_REG	LP_TIMER_REG_1_INTR mapping register	0x000C	R/W
COREO_MB_HP_INT_MAP_REG	MB_HP_INTR mapping register	0x0010	R/W
COREO_MB_LP_INT_MAP_REG	MB_LP_INTR mapping register	0x0014	R/W
COREO_PMU_REG_0_INT_MAP_REG	PMU_REG_0_INTR mapping register	0x0018	R/W
COREO_PMU_REG_1_INT_MAP_REG	PMU_REG_1_INTR mapping register	0x001C	R/W
COREO_LP_ANAPERI_INT_MAP_REG	LP_ANAPERI_INTR mapping register	0x0020	R/W
COREO_LP_ADC_INT_MAP_REG	LP_ADC_INTR mapping register	0x0024	R/W
COREO_LP_GPIO_INT_MAP_REG	LP_GPIO_INTR mapping register	0x0028	R/W
COREO_LP_I2C_INT_MAP_REG	LP_I2C_INTR mapping register	0x002C	R/W
COREO_LP_I2S_INT_MAP_REG	LP_I2S_INTR mapping register	0x0030	R/W
COREO_LP_SPI_INT_MAP_REG	LP_SPI_INTR mapping register	0x0034	R/W
COREO_LP_TOUCH_INT_MAP_REG	LP_TOUCH_INTR mapping register	0x0038	R/W
COREO_LP_TSENS_INT_MAP_REG	LP_TSENS_INTR mapping register	0x003C	R/W
COREO_LP_UART_INT_MAP_REG	LP_UART_INTR mapping register	0x0040	R/W
COREO_LP_EFUSE_INT_MAP_REG	LP_EFUSE_INTR mapping register	0x0044	R/W
COREO_LP_SW_INT_MAP_REG	LP_SW_INTR mapping register	0x0048	R/W
COREO_LP_SYSREG_INT_MAP_REG	LP_SYSREG_INTR mapping register	0x004C	R/W
COREO_LP_HUK_INT_MAP_REG	LP_HUK_INTR mapping register	0x0050	R/W

GoBack

Name	Description	Address	Access
COREO_SYS_ICM_INT_MAP_REG	SYS_ICM_INTR mapping register	0x0054	R/W
COREO_USB_DEVICE_INT_MAP_REG	USB_DEVICE_INTR mapping register	0x0058	R/W
COREO_SDIO_HOST_INT_MAP_REG	SDIO_HOST_INTR mapping register	0x005C	R/W
COREO_GDMA_INT_MAP_REG	GDMA_INTR mapping register	0x0060	R/W
COREO_SPI2_INT_MAP_REG	SPI2_INTR mapping register	0x0064	R/W
COREO_SPI3_INT_MAP_REG	SPI3_INTR mapping register	0x0068	R/W
COREO_I2S0_INT_MAP_REG	I2S0_INTR mapping register	0x006C	R/W
COREO_I2S1_INT_MAP_REG	I2S1_INTR mapping register	0x0070	R/W
COREO_I2S2_INT_MAP_REG	I2S2_INTR mapping register	0x0074	R/W
COREO_UHCI0_INT_MAP_REG	UHCI0_INTR mapping register	0x0078	R/W
COREO_UART0_INT_MAP_REG	UART0_INTR mapping register	0x007C	R/W
COREO_UART1_INT_MAP_REG	UART1_INTR mapping register	0x0080	R/W
COREO_UART2_INT_MAP_REG	UART2_INTR mapping register	0x0084	R/W
COREO_UART3_INT_MAP_REG	UART3_INTR mapping register	0x0088	R/W
COREO_UART4_INT_MAP_REG	UART4_INTR mapping register	0x008C	R/W
COREO_LCD_CAM_INT_MAP_REG	LCD_CAM_INTR mapping register	0x0090	R/W
COREO_ADC_INT_MAP_REG	ADC_INTR mapping register	0x0094	R/W
COREO_PWM0_INT_MAP_REG	PWM0_INTR mapping register	0x0098	R/W
COREO_PWM1_INT_MAP_REG	PWM1_INTR mapping register	0x009C	R/W
COREO_TWAI0_INT_MAP_REG	TWAI0_INTR mapping register	0x00A0	R/W
COREO_TWAI1_INT_MAP_REG	TWAI1_INTR mapping register	0x00A4	R/W
COREO_TWAI2_INT_MAP_REG	TWAI2_INTR mapping register	0x00A8	R/W
COREO_RMT_INT_MAP_REG	RMT_INTR mapping register	0x00AC	R/W
COREO_I2C0_INT_MAP_REG	I2C0_INTR mapping register	0x00B0	R/W
COREO_I2C1_INT_MAP_REG	I2C1_INTR mapping register	0x00B4	R/W
COREO_TIMERGRP0_T0_INT_MAP_REG	TIMERGRP0_T0_INTR mapping register	0x00B8	R/W
COREO_TIMERGRP0_T1_INT_MAP_REG	TIMERGRP0_T1_INTR mapping register	0x00BC	R/W
COREO_TIMERGRP0_WDT_INT_MAP_REG	TIMERGRP0_WDT_INTR mapping register	0x00C0	R/W
COREO_TIMERGRP1_T0_INT_MAP_REG	TIMERGRP1_T0_INTR mapping register	0x00C4	R/W

Name	Description	Address	Access
COREO_TIMERGRP1_T1_INT_MAP_REG	TIMERGRP1_T1_INTR mapping register	0x00C8	R/W
COREO_TIMERGRP1_WDT_INT_MAP_REG	TIMERGRP1_WDT_INTR mapping register	0x00CC	R/W
COREO_LEDC_INT_MAP_REG	LEDC_INTR mapping register	0x00D0	R/W
COREO_SYSTIMER_TARGET0_INT_MAP_REG	SYSTIMER_TARGET0_INTR mapping register	0x00D4	R/W
COREO_SYSTIMER_TARGET1_INT_MAP_REG	SYSTIMER_TARGET1_INTR mapping register	0x00D8	R/W
COREO_SYSTIMER_TARGET2_INT_MAP_REG	SYSTIMER_TARGET2_INTR mapping register	0x00DC	R/W
COREO_AHB_PDMA_IN_CH0_INT_MAP_REG	AHB_PDMA_IN_CH0_INTR mapping register	0x00E0	R/W
COREO_AHB_PDMA_IN_CH1_INT_MAP_REG	AHB_PDMA_IN_CH1_INTR mapping register	0x00E4	R/W
COREO_AHB_PDMA_IN_CH2_INT_MAP_REG	AHB_PDMA_IN_CH2_INTR mapping register	0x00E8	R/W
COREO_AHB_PDMA_OUT_CH0_INT_MAP_REG	AHB_PDMA_OUT_CH0_INTR mapping register	0x00EC	R/W
COREO_AHB_PDMA_OUT_CH1_INT_MAP_REG	AHB_PDMA_OUT_CH1_INTR mapping register	0x00F0	R/W
COREO_AHB_PDMA_OUT_CH2_INT_MAP_REG	AHB_PDMA_OUT_CH2_INTR mapping register	0x00F4	R/W
COREO_AXI_PDMA_IN_CH0_INT_MAP_REG	AXI_PDMA_IN_CH0_INTR mapping register	0x00F8	R/W
COREO_AXI_PDMA_IN_CH1_INT_MAP_REG	AXI_PDMA_IN_CH1_INTR mapping register	0x00FC	R/W
COREO_AXI_PDMA_IN_CH2_INT_MAP_REG	AXI_PDMA_IN_CH2_INTR mapping register	0x0100	R/W
COREO_AXI_PDMA_OUT_CH0_INT_MAP_REG	AXI_PDMA_OUT_CH0_INTR mapping register	0x0104	R/W
COREO_AXI_PDMA_OUT_CH1_INT_MAP_REG	AXI_PDMA_OUT_CH1_INTR mapping register	0x0108	R/W
COREO_AXI_PDMA_OUT_CH2_INT_MAP_REG	AXI_PDMA_OUT_CH2_INTR mapping register	0x010C	R/W
COREO_RSA_INT_MAP_REG	RSA_INTR mapping register	0x0110	R/W
COREO_AES_INT_MAP_REG	AES_INTR mapping register	0x0114	R/W
COREO_SHA_INT_MAP_REG	SHA_INTR mapping register	0x0118	R/W
COREO_ECC_INT_MAP_REG	ECC_INTR mapping register	0x011C	R/W
COREO_ECDSA_INT_MAP_REG	ECDSA_INTR mapping register	0x0120	R/W
COREO_KM_INT_MAP_REG	KM_INTR mapping register	0x0124	R/W
COREO_GPIO_INT0_MAP_REG	GPIO_INTR0 mapping register	0x0128	R/W
COREO_GPIO_INT1_MAP_REG	GPIO_INTR1 mapping register	0x012C	R/W
COREO_GPIO_INT2_MAP_REG	GPIO_INTR2 mapping register	0x0130	R/W
COREO_GPIO_INT3_MAP_REG	GPIO_INTR3 mapping register	0x0134	R/W
COREO_GPIO_PAD_COMP_INT_MAP_REG	GPIO_PAD_COMP_INTR mapping register	0x0138	R/W

GoBack

Name	Description	Address	Access
COREO_CPU_INT_FROM_CPU_0_MAP_REG	CPU_INTR_FROM_CPU_0 mapping register	0x013C	R/W
COREO_CPU_INT_FROM_CPU_1_MAP_REG	CPU_INTR_FROM_CPU_1 mapping register	0x0140	R/W
COREO_CPU_INT_FROM_CPU_2_MAP_REG	CPU_INTR_FROM_CPU_2 mapping register	0x0144	R/W
COREO_CPU_INT_FROM_CPU_3_MAP_REG	CPU_INTR_FROM_CPU_3 mapping register	0x0148	R/W
COREO_FLASH_MSPI_INT_MAP_REG	FLASH_MSPI_INTR mapping register	0x0150	R/W
COREO_CSI_BRIDGE_INT_MAP_REG	CSI_BRIDGE_INTR mapping register	0x0154	R/W
COREO_DSI_BRIDGE_INT_MAP_REG	DSI_BRIDGE_INTR mapping register	0x0158	R/W
COREO_CSI_INT_MAP_REG	CSI_INTR mapping register	0x015C	R/W
COREO_DSI_INT_MAP_REG	DSI_INTR mapping register	0x0160	R/W
COREO_GMII_PHY_INT_MAP_REG	GMII_PHY_INTR mapping register	0x0164	R/W
COREO_LPI_INT_MAP_REG	LPI_INTR mapping register	0x0168	R/W
COREO_PMT_INT_MAP_REG	PMT_INTR mapping register	0x016C	R/W
COREO_ETH_MAC_INT_MAP_REG	ETH_MAC_INTR mapping register	0x0170	R/W
COREO_USB_OTG_INT_MAP_REG	USB_OTG_INTR mapping register	0x0174	R/W
COREO_USB_OTG_ENDP_MULTI_PROC_INT_MAP_REG	USB_OTG_ENDP_MULTI_PROC_INTR mapping register	0x0178	R/W
COREO_JPEG_INT_MAP_REG	JPEG_INTR mapping register	0x017C	R/W
COREO_PPA_INT_MAP_REG	PPA_INTR mapping register	0x0180	R/W
COREO_COREO_TRACE_INT_MAP_REG	COREO_TRACE_INTR mapping register	0x0184	R/W
COREO_CORE1_TRACE_INT_MAP_REG	CORE1_TRACE_INTR mapping register	0x0188	R/W
COREO_ISP_INT_MAP_REG	ISP_INTR mapping register	0x0190	R/W
COREO_I3C_MST_INT_MAP_REG	I3C_MST_INTR mapping register	0x0194	R/W
COREO_I3C_SLV_INT_MAP_REG	I3C_SLV_INTR mapping register	0x0198	R/W
COREO_USB_OTG11_INT_MAP_REG	USB_OTG11_INTR mapping register	0x019C	R/W
COREO_DMA2D_IN_CH0_INT_MAP_REG	DMA2D_IN_CH0_INTR mapping register	0x01A0	R/W
COREO_DMA2D_IN_CH1_INT_MAP_REG	DMA2D_IN_CH1_INTR mapping register	0x01A4	R/W
COREO_DMA2D_OUT_CH0_INT_MAP_REG	DMA2D_OUT_CH0_INTR mapping register	0x01A8	R/W
COREO_DMA2D_OUT_CH1_INT_MAP_REG	DMA2D_OUT_CH1_INTR mapping register	0x01AC	R/W
COREO_DMA2D_OUT_CH2_INT_MAP_REG	DMA2D_OUT_CH2_INTR mapping register	0x01B0	R/W

Name	Description	Address	Access
CORE0_PSRAM_MSPI_INT_MAP_REG	PSRAM_MSPI_INTR mapping register	0x01B4	R/W
CORE0_HP_SYSREG_INT_MAP_REG	HP_SYSREG_INTR mapping register	0x01B8	R/W
CORE0_PCNT_INT_MAP_REG	PCNT_INTR mapping register	0x01BC	R/W
CORE0_HP_PARLIO_RX_INT_MAP_REG	HP_PARLIO_RX_INTR mapping register	0x01C4	R/W
CORE0_HP_PARLIO_TX_INT_MAP_REG	HP_PARLIO_TX_INTR mapping register	0x01C8	R/W
CORE0_H264_DMA2D_OUT_CH0_INT_MAP_REG	H264_DMA2D_OUT_CH0_INTR mapping register	0x01CC	R/W
CORE0_H264_DMA2D_OUT_CH1_INT_MAP_REG	H264_DMA2D_OUT_CH1_INTR mapping register	0x01D0	R/W
CORE0_H264_DMA2D_OUT_CH2_INT_MAP_REG	H264_DMA2D_OUT_CH2_INTR mapping register	0x01D4	R/W
CORE0_H264_DMA2D_OUT_CH3_INT_MAP_REG	H264_DMA2D_OUT_CH3_INTR mapping register	0x01D8	R/W
CORE0_H264_DMA2D_OUT_CH4_INT_MAP_REG	H264_DMA2D_OUT_CH4_INTR mapping register	0x01DC	R/W
CORE0_H264_DMA2D_IN_CH0_INT_MAP_REG	H264_DMA2D_IN_CH0_INTR mapping register	0x01E0	R/W
CORE0_H264_DMA2D_IN_CH1_INT_MAP_REG	H264_DMA2D_IN_CH1_INTR mapping register	0x01E4	R/W
CORE0_H264_DMA2D_IN_CH2_INT_MAP_REG	H264_DMA2D_IN_CH2_INTR mapping register	0x01E8	R/W
CORE0_H264_DMA2D_IN_CH3_INT_MAP_REG	H264_DMA2D_IN_CH3_INTR mapping register	0x01EC	R/W
CORE0_H264_DMA2D_IN_CH4_INT_MAP_REG	H264_DMA2D_IN_CH4_INTR mapping register	0x01F0	R/W
CORE0_H264_DMA2D_IN_CH5_INT_MAP_REG	H264_DMA2D_IN_CH5_INTR mapping register	0x01F4	R/W
CORE0_H264_REG_INT_MAP_REG	H264_REG_INTR mapping register	0x01F8	R/W
CORE0_ASSIST_DEBUG_INT_MAP_REG	ASSIST_DEBUG_INTR mapping register	0x01FC	R/W
CORE0_INTR_STATUS_REG_0_REG	Status register for interrupt sources 0 ~ 31	0x0200	RO
CORE0_INTR_STATUS_REG_1_REG	Status register for interrupt sources 32 ~ 63	0x0204	RO
CORE0_INTR_STATUS_REG_2_REG	Status register for interrupt sources 64 ~ 95	0x0208	RO
CORE0_INTR_STATUS_REG_3_REG	Status register for interrupt sources 96 ~ 127	0x020C	RO
CORE0_CLOCK_GATE_REG	Clock gating register	0x0210	R/W
CORE0_DMA2D_IN_CH2_INT_MAP_REG	DMA2D_IN_CH2_INTR mapping register	0x214	R/W

Name	Description	Address	Access
COREO_DMA2D_OUT_CH3_INT_MAP_REG	DMA2D_OUT_CH3_INTR mapping register	0x218	R/W
COREO_AXI_PERF_MON_INT_MAP_REG	AXI_PERF_MON_INTR mapping register	0x21C	R/W
COREO_INTR_STATUS_REG_4_REG	Status register for interrupt sources 128 ~ 130	0x0220	RO
COREO_INTR_SIG_IDX_ASSERT_IN_SEC_REG	Interrupt delegation configuration register for COREO	0x0228	R/W
COREO_INTR_SEC_STATUS_REG	Interrupt delegation status register for COREO	0x022C	RO
COREO_INTR_SRC_PAS_IN_SEC_STATUS_0_REG	Interrupt delegation status register for interrupt sources 0 ~ 31	0x0230	RO
COREO_INTR_SRC_PAS_IN_SEC_STATUS_1_REG	Interrupt delegation status register for interrupt sources 32 ~ 63	0x0234	RO
COREO_INTR_SRC_PAS_IN_SEC_STATUS_2_REG	Interrupt delegation status register for interrupt sources 64 ~ 95	0x0238	RO
COREO_INTR_SRC_PAS_IN_SEC_STATUS_3_REG	Interrupt delegation status register for interrupt sources 96 ~ 127	0x023C	RO
COREO_INTR_SRC_PAS_IN_SEC_STATUS_4_REG	Interrupt delegation status register for interrupt sources 128 ~ 130	0x0240	RO
COREO_INTERRUPT_REG_DATE_REG	Version control register	0x03FC	R/W

13.5.2 HP CPU1 Interrupt Matrix Register Summary

Name	Description	Address	Access
CORE1_LP_RTC_INT_MAP_REG	LP_RTC_INTR mapping register	0x0000	R/W
CORE1_LP_WDT_INT_MAP_REG	LP_WDT_INTR mapping register	0x0004	R/W
CORE1_LP_TIMER_REG_0_INT_MAP_REG	LP_TIMER_REG_0_INTR mapping register	0x0008	R/W
CORE1_LP_TIMER_REG_1_INT_MAP_REG	LP_TIMER_REG_1_INTR mapping register	0x000C	R/W
CORE1_MB_HP_INT_MAP_REG	MB_HP_INTR mapping register	0x0010	R/W
CORE1_MB_LP_INT_MAP_REG	MB_LP_INTR mapping register	0x0014	R/W
CORE1_PMU_REG_0_INT_MAP_REG	PMU_REG_0_INTR mapping register	0x0018	R/W

Name	Description	Address	Access
CORE1_PMU_REG_1_INT_MAP_REG	PMU_REG_1_INTR mapping register	0x001C	R/W
CORE1_LP_ANAPERI_INT_MAP_REG	LP_ANAPERI_INTR mapping register	0x0020	R/W
CORE1_LP_ADC_INT_MAP_REG	LP_ADC_INTR mapping register	0x0024	R/W
CORE1_LP_GPIO_INT_MAP_REG	LP_GPIO_INTR mapping register	0x0028	R/W
CORE1_LP_I2C_INT_MAP_REG	LP_I2C_INTR mapping register	0x002C	R/W
CORE1_LP_I2S_INT_MAP_REG	LP_I2S_INTR mapping register	0x0030	R/W
CORE1_LP_SPI_INT_MAP_REG	LP_SPI_INTR mapping register	0x0034	R/W
CORE1_LP_TOUCH_INT_MAP_REG	LP_TOUCH_INTR mapping register	0x0038	R/W
CORE1_LP_TSENS_INT_MAP_REG	LP_TSENS_INTR mapping register	0x003C	R/W
CORE1_LP_UART_INT_MAP_REG	LP_UART_INTR mapping register	0x0040	R/W
CORE1_LP_EFUSE_INT_MAP_REG	LP_EFUSE_INTR mapping register	0x0044	R/W
CORE1_LP_SW_INT_MAP_REG	LP_SW_INTR mapping register	0x0048	R/W
CORE1_LP_SYSREG_INT_MAP_REG	LP_SYSREG_INTR mapping register	0x004C	R/W
CORE1_LP_HUK_INT_MAP_REG	LP_HUK_INTR mapping register	0x0050	R/W
CORE1_SYS_ICM_INT_MAP_REG	SYS_ICM_INTR mapping register	0x0054	R/W
CORE1_USB_DEVICE_INT_MAP_REG	USB_DEVICE_INTR mapping register	0x0058	R/W
CORE1_SDIO_HOST_INT_MAP_REG	SDIO_HOST_INTR mapping register	0x005C	R/W
CORE1_GDMA_INT_MAP_REG	GDMA_INTR mapping register	0x0060	R/W
CORE1_SPI2_INT_MAP_REG	SPI2_INTR mapping register	0x0064	R/W
CORE1_SPI3_INT_MAP_REG	SPI3_INTR mapping register	0x0068	R/W
CORE1_I2S0_INT_MAP_REG	I2S0_INTR mapping register	0x006C	R/W
CORE1_I2S1_INT_MAP_REG	I2S1_INTR mapping register	0x0070	R/W
CORE1_I2S2_INT_MAP_REG	I2S2_INTR mapping register	0x0074	R/W
CORE1_UHCI0_INT_MAP_REG	UHCI0_INTR mapping register	0x0078	R/W
CORE1_UART0_INT_MAP_REG	UART0_INTR mapping register	0x007C	R/W
CORE1_UART1_INT_MAP_REG	UART1_INTR mapping register	0x0080	R/W
CORE1_UART2_INT_MAP_REG	UART2_INTR mapping register	0x0084	R/W
CORE1_UART3_INT_MAP_REG	UART3_INTR mapping register	0x0088	R/W
CORE1_UART4_INT_MAP_REG	UART4_INTR mapping register	0x008C	R/W

Name	Description	Address	Access
CORE1_LCD_CAM_INT_MAP_REG	LCD_CAM_INTR mapping register	0x0090	R/W
CORE1_ADC_INT_MAP_REG	ADC_INTR mapping register	0x0094	R/W
CORE1_PWM0_INT_MAP_REG	PWM0_INTR mapping register	0x0098	R/W
CORE1_PWM1_INT_MAP_REG	PWM1_INTR mapping register	0x009C	R/W
CORE1_TWAI0_INT_MAP_REG	TWAI0_INTR mapping register	0x00A0	R/W
CORE1_TWAI1_INT_MAP_REG	TWAI1_INTR mapping register	0x00A4	R/W
CORE1_TWAI2_INT_MAP_REG	TWAI2_INTR mapping register	0x00A8	R/W
CORE1_RMT_INT_MAP_REG	RMT_INTR mapping register	0x00AC	R/W
CORE1_I2C0_INT_MAP_REG	I2C0_INTR mapping register	0x00B0	R/W
CORE1_I2C1_INT_MAP_REG	I2C1_INTR mapping register	0x00B4	R/W
CORE1_TIMERGRP0_T0_INT_MAP_REG	TIMERGRP0_T0_INTR mapping register	0x00B8	R/W
CORE1_TIMERGRP0_T1_INT_MAP_REG	TIMERGRP0_T1_INTR mapping register	0x00BC	R/W
CORE1_TIMERGRP0_WDT_INT_MAP_REG	TIMERGRP0_WDT_INTR mapping register	0x00C0	R/W
CORE1_TIMERGRP1_T0_INT_MAP_REG	TIMERGRP1_T0_INTR mapping register	0x00C4	R/W
CORE1_TIMERGRP1_T1_INT_MAP_REG	TIMERGRP1_T1_INTR mapping register	0x00C8	R/W
CORE1_TIMERGRP1_WDT_INT_MAP_REG	TIMERGRP1_WDT_INTR mapping register	0x00CC	R/W
CORE1_LEDC_INT_MAP_REG	LEDC_INTR mapping register	0x00D0	R/W
CORE1_SYSTIMER_TARGET0_INT_MAP_REG	SYSTIMER_TARGET0_INTR mapping register	0x00D4	R/W
CORE1_SYSTIMER_TARGET1_INT_MAP_REG	SYSTIMER_TARGET1_INTR mapping register	0x00D8	R/W
CORE1_SYSTIMER_TARGET2_INT_MAP_REG	SYSTIMER_TARGET2_INTR mapping register	0x00DC	R/W
CORE1_AHB_PDMA_IN_CH0_INT_MAP_REG	AHB_PDMA_IN_CH0_INTR mapping register	0x00E0	R/W
CORE1_AHB_PDMA_IN_CH1_INT_MAP_REG	AHB_PDMA_IN_CH1_INTR mapping register	0x00E4	R/W
CORE1_AHB_PDMA_IN_CH2_INT_MAP_REG	AHB_PDMA_IN_CH2_INTR mapping register	0x00E8	R/W
CORE1_AHB_PDMA_OUT_CH0_INT_MAP_REG	AHB_PDMA_OUT_CH0_INTR mapping register	0x00EC	R/W
CORE1_AHB_PDMA_OUT_CH1_INT_MAP_REG	AHB_PDMA_OUT_CH1_INTR mapping register	0x00F0	R/W
CORE1_AHB_PDMA_OUT_CH2_INT_MAP_REG	AHB_PDMA_OUT_CH2_INTR mapping register	0x00F4	R/W
CORE1_AXI_PDMA_IN_CH0_INT_MAP_REG	AXI_PDMA_IN_CH0_INTR mapping register	0x00F8	R/W
CORE1_AXI_PDMA_IN_CH1_INT_MAP_REG	AXI_PDMA_IN_CH1_INTR mapping register	0x00FC	R/W
CORE1_AXI_PDMA_IN_CH2_INT_MAP_REG	AXI_PDMA_IN_CH2_INTR mapping register	0x0100	R/W

Name	Description	Address	Access
CORE1_AXI_PDMA_OUT_CH0_INT_MAP_REG	AXI_PDMA_OUT_CH0_INTR mapping register	0x0104	R/W
CORE1_AXI_PDMA_OUT_CH1_INT_MAP_REG	AXI_PDMA_OUT_CH1_INTR mapping register	0x0108	R/W
CORE1_AXI_PDMA_OUT_CH2_INT_MAP_REG	AXI_PDMA_OUT_CH2_INTR mapping register	0x010C	R/W
CORE1_RSA_INT_MAP_REG	RSA_INTR mapping register	0x0110	R/W
CORE1_AES_INT_MAP_REG	AES_INTR mapping register	0x0114	R/W
CORE1_SHA_INT_MAP_REG	SHA_INTR mapping register	0x0118	R/W
CORE1_ECC_INT_MAP_REG	ECC_INTR mapping register	0x011C	R/W
CORE1_ECDSA_INT_MAP_REG	ECDSA_INTR mapping register	0x0120	R/W
CORE1_KM_INT_MAP_REG	KM_INTR mapping register	0x0124	R/W
CORE1_GPIO_INT0_MAP_REG	GPIO_INTR0 mapping register	0x0128	R/W
CORE1_GPIO_INT1_MAP_REG	GPIO_INTR1 mapping register	0x012C	R/W
CORE1_GPIO_INT2_MAP_REG	GPIO_INTR2 mapping register	0x0130	R/W
CORE1_GPIO_INT3_MAP_REG	GPIO_INTR3 mapping register	0x0134	R/W
CORE1_GPIO_PAD_COMP_INT_MAP_REG	GPIO_PAD_COMP_INTR mapping register	0x0138	R/W
CORE1_CPU_INT_FROM_CPU_0_MAP_REG	CPU_INTR_FROM_CPU_0 mapping register	0x013C	R/W
CORE1_CPU_INT_FROM_CPU_1_MAP_REG	CPU_INTR_FROM_CPU_1 mapping register	0x0140	R/W
CORE1_CPU_INT_FROM_CPU_2_MAP_REG	CPU_INTR_FROM_CPU_2 mapping register	0x0144	R/W
CORE1_CPU_INT_FROM_CPU_3_MAP_REG	CPU_INTR_FROM_CPU_3 mapping register	0x0148	R/W
CORE1_FLASH_MSPI_INT_MAP_REG	FLASH_MSPI_INTR mapping register	0x0150	R/W
CORE1_CSI_BRIDGE_INT_MAP_REG	CSI_BRIDGE_INTR mapping register	0x0154	R/W
CORE1_DSI_BRIDGE_INT_MAP_REG	DSI_BRIDGE_INTR mapping register	0x0158	R/W
CORE1_CSI_INT_MAP_REG	CSI_INTR mapping register	0x015C	R/W
CORE1_DSI_INT_MAP_REG	DSI_INTR mapping register	0x0160	R/W
CORE1_GMII_PHY_INT_MAP_REG	GMII_PHY_INTR mapping register	0x0164	R/W
CORE1_LPI_INT_MAP_REG	LPI_INTR mapping register	0x0168	R/W
CORE1_PMT_INT_MAP_REG	PMT_INTR mapping register	0x016C	R/W
CORE1_ETH_MAC_INT_MAP_REG	ETH_MAC_INTR mapping register	0x0170	R/W
CORE1_USB_OTG_INT_MAP_REG	USB_OTG_INTR mapping register	0x0174	R/W

Name	Description	Address	Access
CORE1_USB_OTG_ENDP_MULTI_PROC_INT_MAP_REG	USB_OTG_ENDP_MULTI_PROC_INTR mapping register	0x0178	R/W
CORE1_JPEG_INT_MAP_REG	JPEG_INTR mapping register	0x017C	R/W
CORE1_PPA_INT_MAP_REG	PPA_INTR mapping register	0x0180	R/W
CORE1_COREO_TRACE_INT_MAP_REG	COREO_TRACE_INTR mapping register	0x0184	R/W
CORE1_CORE1_TRACE_INT_MAP_REG	CORE1_TRACE_INTR mapping register	0x0188	R/W
CORE1_ISP_INT_MAP_REG	ISP_INTR mapping register	0x0190	R/W
CORE1_I3C_MST_INT_MAP_REG	I3C_MST_INTR mapping register	0x0194	R/W
CORE1_I3C_SLV_INT_MAP_REG	I3C_SLV_INTR mapping register	0x0198	R/W
CORE1_USB_OTG11_INT_MAP_REG	USB_OTG11_INTR mapping register	0x019C	R/W
CORE1_DMA2D_IN_CHO_INT_MAP_REG	DMA2D_IN_CHO_INTR mapping register	0x01A0	R/W
CORE1_DMA2D_IN_CH1_INT_MAP_REG	DMA2D_IN_CH1_INTR mapping register	0x01A4	R/W
CORE1_DMA2D_OUT_CHO_INT_MAP_REG	DMA2D_OUT_CHO_INTR mapping register	0x01A8	R/W
CORE1_DMA2D_OUT_CH1_INT_MAP_REG	DMA2D_OUT_CH1_INTR mapping register	0x01AC	R/W
CORE1_DMA2D_OUT_CH2_INT_MAP_REG	DMA2D_OUT_CH2_INTR mapping register	0x01B0	R/W
CORE1_PSRAM_MSPI_INT_MAP_REG	PSRAM_MSPI_INTR mapping register	0x01B4	R/W
CORE1_HP_SYSREG_INT_MAP_REG	HP_SYSREG_INTR mapping register	0x01B8	R/W
CORE1_PCNT_INT_MAP_REG	PCNT_INTR mapping register	0x01BC	R/W
CORE1_HP_PARLIO_RX_INT_MAP_REG	HP_PARLIO_RX_INTR mapping register	0x01C4	R/W
CORE1_HP_PARLIO_TX_INT_MAP_REG	HP_PARLIO_TX_INTR mapping register	0x01C8	R/W
CORE1_H264_DMA2D_OUT_CHO_INT_MAP_REG	H264_DMA2D_OUT_CHO_INTR mapping register	0x01CC	R/W
CORE1_H264_DMA2D_OUT_CH1_INT_MAP_REG	H264_DMA2D_OUT_CH1_INTR mapping register	0x01D0	R/W
CORE1_H264_DMA2D_OUT_CH2_INT_MAP_REG	H264_DMA2D_OUT_CH2_INTR mapping register	0x01D4	R/W
CORE1_H264_DMA2D_OUT_CH3_INT_MAP_REG	H264_DMA2D_OUT_CH3_INTR mapping register	0x01D8	R/W

Name	Description	Address	Access
CORE1_H264_DMA2D_OUT_CH4_INT_MAP_REG	H264_DMA2D_OUT_CH4_INTR mapping register	0x01DC	R/W
CORE1_H264_DMA2D_IN_CH0_INT_MAP_REG	H264_DMA2D_IN_CH0_INTR mapping register	0x01E0	R/W
CORE1_H264_DMA2D_IN_CH1_INT_MAP_REG	H264_DMA2D_IN_CH1_INTR mapping register	0x01E4	R/W
CORE1_H264_DMA2D_IN_CH2_INT_MAP_REG	H264_DMA2D_IN_CH2_INTR mapping register	0x01E8	R/W
CORE1_H264_DMA2D_IN_CH3_INT_MAP_REG	H264_DMA2D_IN_CH3_INTR mapping register	0x01EC	R/W
CORE1_H264_DMA2D_IN_CH4_INT_MAP_REG	H264_DMA2D_IN_CH4_INTR mapping register	0x01F0	R/W
CORE1_H264_DMA2D_IN_CH5_INT_MAP_REG	H264_DMA2D_IN_CH5_INTR mapping register	0x01F4	R/W
CORE1_H264_REG_INT_MAP_REG	H264_REG_INTR mapping register	0x01F8	R/W
CORE1_ASSIST_DEBUG_INT_MAP_REG	ASSIST_DEBUG_INTR mapping register	0x01FC	R/W
CORE1_INTR_STATUS_REG_0_REG	Status register for interrupt sources 0 ~ 31	0x0200	RO
CORE1_INTR_STATUS_REG_1_REG	Status register for interrupt sources 32 ~ 63	0x0204	RO
CORE1_INTR_STATUS_REG_2_REG	Status register for interrupt sources 64 ~ 95	0x0208	RO
CORE1_INTR_STATUS_REG_3_REG	Status register for interrupt sources 96 ~ 127	0x020C	RO
CORE1_CLOCK_GATE_REG	Clock gating register	0x0210	R/W
CORE1_DMA2D_IN_CH2_INT_MAP_REG	DMA2D_IN_CH2_INTR mapping register	0x214	R/W
CORE1_DMA2D_OUT_CH3_INT_MAP_REG	DMA2D_OUT_CH3_INTR mapping register	0x218	R/W
CORE1_AXI_PERF_MON_INT_MAP_REG	AXI_PERF_MON_INTR mapping register	0x21C	R/W
CORE1_INTR_STATUS_REG_4_REG	Status register for interrupt sources 128 ~ 130	0x0220	RO
CORE1_INTR_SIG_IDX_ASSERT_IN_SEC_REG	CORE1 interrupt delegation configuration register	0x0228	R/W
CORE1_INTR_SEC_STATUS_REG	CORE1 interrupt delegation status register	0x022C	RO
CORE1_INTR_SRC_PASS_IN_SEC_STATUS_0_REG	Interrupt delegation status register for interrupt sources 0 ~ 31	0x0230	RO
CORE1_INTR_SRC_PASS_IN_SEC_STATUS_1_REG	Interrupt delegation status register for interrupt sources 32 ~ 63	0x0234	RO
CORE1_INTR_SRC_PASS_IN_SEC_STATUS_2_REG	Interrupt delegation status register for interrupt sources 64 ~ 95	0x0238	RO

Name	Description	Address	Access
CORE1_INTR_SRC_PASS_IN_SEC_STATUS_3_REG	Interrupt delegation status register for interrupt sources 96 ~ 127	0x023C	RO
CORE1_INTR_SRC_PASS_IN_SEC_STATUS_4_REG	Interrupt delegation status register for interrupt sources 128 ~ 130	0x0240	RO
CORE1_INTERRUPT_REG_DATE_REG	Version control register	0x03FC	R/W

13.6 Registers

The addresses in this section are relative to the interrupt matrix base address provided in Table 9.3-2 in Chapter 9 *System and Memory*.

For how to program reserved fields, please refer to Section *Programming Reserved Register Field*.

13.6.1 HP CPU0 Interrupt Matrix Registers

Register 13.1. COREO_*LP_RTC_INT*_MAP_REG (0x0000)

Register 13.2. COREO_*LP_WDT_INT*_MAP_REG (0x0004)

Register 13.3. COREO_*LP_TIMER_REG_O_INT*_MAP_REG (0x0008)

Register 13.4. COREO_*LP_TIMER_REG_1_INT*_MAP_REG (0x000C)

Register 13.5. COREO_*MB_HP_INT*_MAP_REG (0x0010)

Register 13.6. COREO_*MB_LP_INT*_MAP_REG (0x0014)

Register 13.7. COREO_*PMU_REG_O_INT*_MAP_REG (0x0018)

Register 13.8. COREO_*PMU_REG_1_INT*_MAP_REG (0x001C)

Register 13.9. COREO_*LP_ANAPERI_INT*_MAP_REG (0x0020)

Register 13.10. COREO_*LP_ADC_INT*_MAP_REG (0x0024)

Register 13.11. COREO_*LP_GPIO_INT*_MAP_REG (0x0028)

Register 13.12. COREO_*LP_I2C_INT*_MAP_REG (0x002C)

Register 13.13. COREO_*LP_I2S_INT*_MAP_REG (0x0030)

Register 13.14. COREO_*LP_SPI_INT*_MAP_REG (0x0034)

Register 13.15. COREO_*LP_TOUCH_INT*_MAP_REG (0x0038)

Register 13.16. COREO_*LP_TSENS_INT*_MAP_REG (0x003C)

Register 13.17. COREO_*LP_UART_INT*_MAP_REG (0x0040)

Register 13.18. COREO_*LP_EFUSE_INT*_MAP_REG (0x0044)

Register 13.19. COREO_*LP_SW_INT*_MAP_REG (0x0048)

Register 13.20. COREO_*LP_SYSREG_INT*_MAP_REG (0x004C)

Register 13.21. COREO_*LP_HUK_INT*_MAP_REG (0x0050)

Register 13.22. COREO_*SYS_ICM_INT*_MAP_REG (0x0054)

Register 13.23. COREO_*USB_DEVICE_INT*_MAP_REG (0x0058)

Register 13.24. COREO_*SDIO_HOST_INT*_MAP_REG (0x005C)

Register 13.25. COREO_*GDMA_INT*_MAP_REG (0x0060)

Register 13.26. COREO_*SPI2_INT*_MAP_REG (0x0064)

Register 13.27. COREO_*SPI3_INT*_MAP_REG (0x0068)

Register 13.28. COREO_*I2SO_INT*_MAP_REG (0x006C)

Register 13.29. COREO_*I2S1_INT*_MAP_REG (0x0070)

Register 13.30. COREO_*I2S2_INT*_MAP_REG (0x0074)

Register 13.31. COREO_*UHCI0_INT*_MAP_REG (0x0078)

Register 13.32. COREO_*UART0_INT*_MAP_REG (0x007C)

Register 13.33. COREO_*UART1_INT*_MAP_REG (0x0080)

Register 13.34. COREO_*UART2_INT*_MAP_REG (0x0084)

Register 13.35. COREO_*UART3_INT*_MAP_REG (0x0088)

Register 13.36. COREO_*UART4_INT*_MAP_REG (0x008C)

Register 13.37. COREO_*LCD_CAM_INT*_MAP_REG (0x0090)

Register 13.38. COREO_*ADC_INT*_MAP_REG (0x0094)

Register 13.39. COREO_*PWM0_INT*_MAP_REG (0x0098)

Register 13.40. COREO_*PWM1_INT*_MAP_REG (0x009C)

Register 13.41. COREO_*TWAI0_INT*_MAP_REG (0x00A0)

Register 13.42. COREO_*TWAI1_INT*_MAP_REG (0x00A4)

Register 13.43. COREO_*TWAI2_INT*_MAP_REG (0x00A8)

Register 13.44. COREO_*RMT_INT*_MAP_REG (0x00AC)

Register 13.45. COREO_*I2C0_INT*_MAP_REG (0x00B0)

Register 13.46. COREO_*I2C1_INT*_MAP_REG (0x00B4)

Register 13.47. COREO_*TIMERGRP0_T0_INT*_MAP_REG (0x00B8)

Register 13.48. COREO_*TIMERGRP0_T1_INT*_MAP_REG (0x00BC)

Register 13.49. COREO_*TIMERGRP0_WDT_INT*_MAP_REG (0x00C0)

Register 13.50. COREO_*TIMERGRP1_T0_INT*_MAP_REG (0x00C4)

Register 13.51. COREO_*TIMERGRP1_T1_INT*_MAP_REG (0x00C8)

Register 13.52. COREO_*TIMERGRP1_WDT_INT*_MAP_REG (0x00CC)

Register 13.53. COREO_*LEDC_INT*_MAP_REG (0x00D0)

Register 13.54. COREO_*SYSTIMER_TARGET0_INT*_MAP_REG (0x00D4)

Register 13.55. COREO_*SYSTIMER_TARGET1_INT*_MAP_REG (0x00D8)

Register 13.56. COREO_*SYSTIMER_TARGET2_INT*_MAP_REG (0x00DC)

Register 13.57. COREO_*AHB_PDMA_IN_CH0_INT*_MAP_REG (0x00E0)

Register 13.58. COREO_*AHB_PDMA_IN_CH1_INT*_MAP_REG (0x00E4)

Register 13.59. COREO_*AHB_PDMA_IN_CH2_INT*_MAP_REG (0x00E8)

Register 13.60. COREO_*AHB_PDMA_OUT_CH0_INT*_MAP_REG (0x00EC)

Register 13.61. COREO_*AHB_PDMA_OUT_CH1_INT*_MAP_REG (0x00F0)

Register 13.62. COREO_*AHB_PDMA_OUT_CH2_INT*_MAP_REG (0x00F4)

Register 13.63. COREO_*AXI_PDMA_IN_CH0_INT*_MAP_REG (0x00F8)

Register 13.64. COREO_*AXI_PDMA_IN_CH1_INT*_MAP_REG (0x00FC)

Register 13.65. COREO_*AXI_PDMA_IN_CH2_INT*_MAP_REG (0x0100)

Register 13.66. COREO_*AXI_PDMA_OUT_CHO_INT*_MAP_REG (0x0104)

Register 13.67. COREO_*AXI_PDMA_OUT_CH1_INT*_MAP_REG (0x0108)

Register 13.68. COREO_*AXI_PDMA_OUT_CH2_INT*_MAP_REG (0x010C)

Register 13.69. COREO_*RSA_INT*_MAP_REG (0x0110)

Register 13.70. COREO_*AES_INT*_MAP_REG (0x0114)

Register 13.71. COREO_*SHA_INT*_MAP_REG (0x0118)

Register 13.72. COREO_*ECC_INT*_MAP_REG (0x011C)

Register 13.73. COREO_*ECDSA_INT*_MAP_REG (0x0120)

Register 13.74. COREO_*KM_INT*_MAP_REG (0x0124)

Register 13.75. COREO_*GPIO_INTO*_MAP_REG (0x0128)

Register 13.76. COREO_*GPIO_INT1*_MAP_REG (0x012C)

Register 13.77. COREO_*GPIO_INT2*_MAP_REG (0x0130)

Register 13.78. COREO_*GPIO_INT3*_MAP_REG (0x0134)

Register 13.79. COREO_*GPIO_PAD_COMP_INT*_MAP_REG (0x0138)

Register 13.80. COREO_*CPU_INT_FROM_CPU_O*_MAP_REG (0x013C)

Register 13.81. COREO_*CPU_INT_FROM_CPU_1*_MAP_REG (0x0140)

Register 13.82. COREO_*CPU_INT_FROM_CPU_2*_MAP_REG (0x0144)

Register 13.83. COREO_*CPU_INT_FROM_CPU_3*_MAP_REG (0x0148)

Register 13.84. COREO_*FLASH_MSPI_INT*_MAP_REG (0x0150)

Register 13.85. COREO_*CSI_BRIDGE_INT*_MAP_REG (0x0154)

Register 13.86. COREO_*DSI_BRIDGE_INT*_MAP_REG (0x0158)

Register 13.87. COREO_*CSI_INT*_MAP_REG (0x015C)

Register 13.88. COREO_*DSI_INT*_MAP_REG (0x0160)

Register 13.89. COREO_*GMII_PHY_INT*_MAP_REG (0x0164)

Register 13.90. COREO_*LPI_INT*_MAP_REG (0x0168)

Register 13.91. COREO_*PMT_INT*_MAP_REG (0x016C)

Register 13.92. COREO_*ETH_MAC_INT*_MAP_REG (0x0170)

Register 13.93. COREO_*USB_OTG_INT*_MAP_REG (0x0174)

Register 13.94. COREO_*USB_OTG_ENDP_MULTI_PROC_INT*_MAP_REG (0x0178)

Register 13.95. COREO_*JPEG_INT*_MAP_REG (0x017C)

Register 13.96. COREO_*PPA_INT*_MAP_REG (0x0180)

Register 13.97. COREO_*COREO_TRACE_INT*_MAP_REG (0x0184)

Register 13.98. COREO_*CORE1_TRACE_INT*_MAP_REG (0x0188)

Register 13.99. COREO_*ISP_INT*_MAP_REG (0x0190)

Register 13.100. COREO_*I3C_MST_INT*_MAP_REG (0x0194)

Register 13.101. COREO_*I3C_SLV_INT*_MAP_REG (0x0198)

Register 13.102. COREO_*USB_OTG11_INT*_MAP_REG (0x019C)

Register 13.103. COREO_*DMA2D_IN_CH0_INT*_MAP_REG (0x01A0)

Register 13.104. COREO_*DMA2D_IN_CH1_INT*_MAP_REG (0x01A4)

Register 13.105. COREO_*DMA2D_OUT_CH0_INT*_MAP_REG (0x01A8)

Register 13.106. COREO_*DMA2D_OUT_CH1_INT*_MAP_REG (0x01AC)

Register 13.107. COREO_*DMA2D_OUT_CH2_INT*_MAP_REG (0x01B0)

Register 13.108. COREO_*PSRAM_MSPI_INT*_MAP_REG (0x01B4)

Register 13.109. COREO_*HP_SYSREG_INT*_MAP_REG (0x01B8)

Register 13.110. COREO_*PCNT_INT*_MAP_REG (0x01BC)

Register 13.111. COREO_*HP_PARLIO_RX_INT*_MAP_REG (0x01C4)

Register 13.112. COREO_*HP_PARLIO_TX_INT*_MAP_REG (0x01C8)

Register 13.113. COREO_*H264_DMA2D_OUT_CH0_INT*_MAP_REG (0x01CC)

Register 13.114. COREO_*H264_DMA2D_OUT_CH1_INT*_MAP_REG (0x01D0)

Register 13.115. COREO_*H264_DMA2D_OUT_CH2_INT*_MAP_REG (0x01D4)

Register 13.116. COREO_*H264_DMA2D_OUT_CH3_INT*_MAP_REG (0x01D8)

Register 13.117. COREO_*H264_DMA2D_OUT_CH4_INT*_MAP_REG (0x01DC)

Register 13.118. COREO_*H264_DMA2D_IN_CH0_INT*_MAP_REG (0x01E0)

Register 13.119. COREO_*H264_DMA2D_IN_CH1_INT*_MAP_REG (0x01E4)

Register 13.120. COREO_*H264_DMA2D_IN_CH2_INT*_MAP_REG (0x01E8)

Register 13.121. COREO_*H264_DMA2D_IN_CH3_INT*_MAP_REG (0x01EC)

Register 13.122. COREO_*H264_DMA2D_IN_CH4_INT*_MAP_REG (0x01F0)

Register 13.123. COREO_*H264_DMA2D_IN_CH5_INT*_MAP_REG (0x01F4)

Register 13.124. COREO_*H264_REG_INT*_MAP_REG (0x01F8)

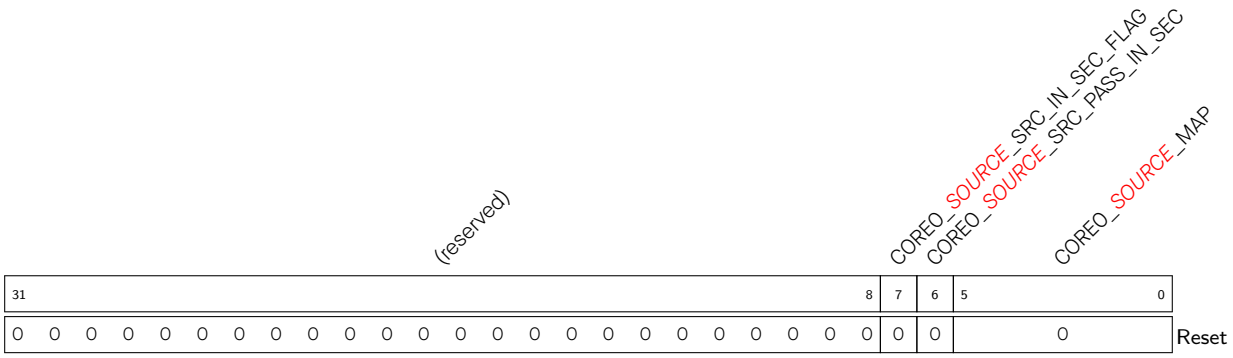
Register 13.125. COREO_*ASSIST_DEBUG_INT*_MAP_REG (0x01FC)

Register 13.126. COREO_*DMA2D_IN_CH2_INT*_MAP_REG (0x0214)

Register 13.127. COREO_*DMA2D_OUT_CH3_INT*_MAP_REG (0x0218)

Register 13.128. COREO_*AXI_PERF_MON_INT*_MAP_REG (0x021C)

Register 13.129. COREO_*SOURCE*_MAP_REG (0x0000 - 0x01FC, 0x0214 - 0x021C)



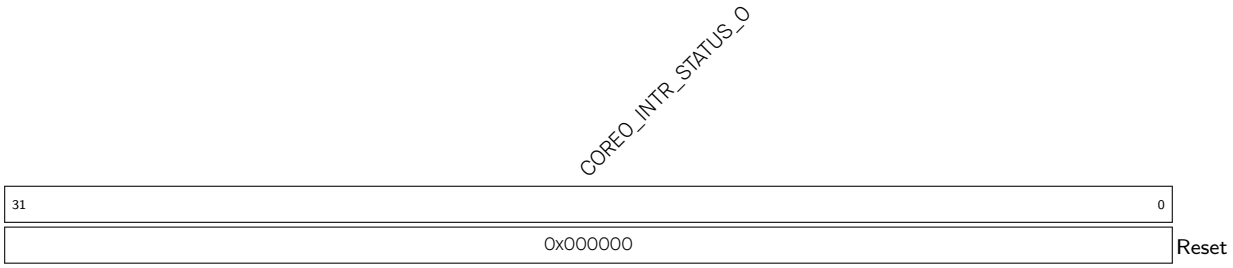
COREO_*SOURCE*_MAP When COREO_*SOURCE*_SRC_IN_SEC_FLAG is set to 1, this register can only be written by CPU while in Machine Mode.

Configure this register to map the interrupt signal from the peripheral interrupt sources *Source* to one of CPU0's peripheral interrupts. The valid configuration range is 16 ~ 47.
When set to 0 ~ 15, the corresponding peripheral interrupt source is disabled (recommended value: 0).
Other values are invalid.
For the list of interrupt sources *Source*, see Table 13.4-1. (R/W)

COREO_*SOURCE*_SRC_PASS_IN_SEC Write 1 to allow the interrupt signal from interrupt source *Source* to be delegated to CPU0's Machine Mode interrupt. Writable only by the CPU in Machine Mode. (R/W)

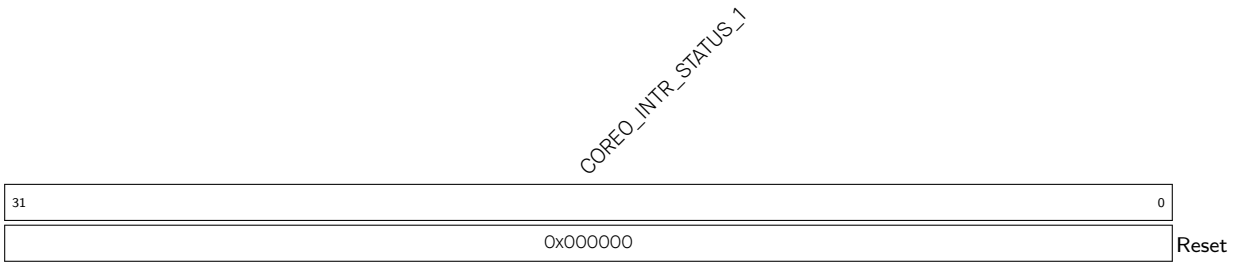
COREO_*SOURCE*_SRC_IN_SEC_FLAG Write 1 to configure the interrupt signal from interrupt source *Source* as a secure interrupt source (that is, its interrupt mapping can be configured only by the CPU running in Machine Mode). Writable only by the CPU in Machine Mode. (R/W)

Register 13.130. COREO_INTR_STATUS_REG_0_REG (0x0200)



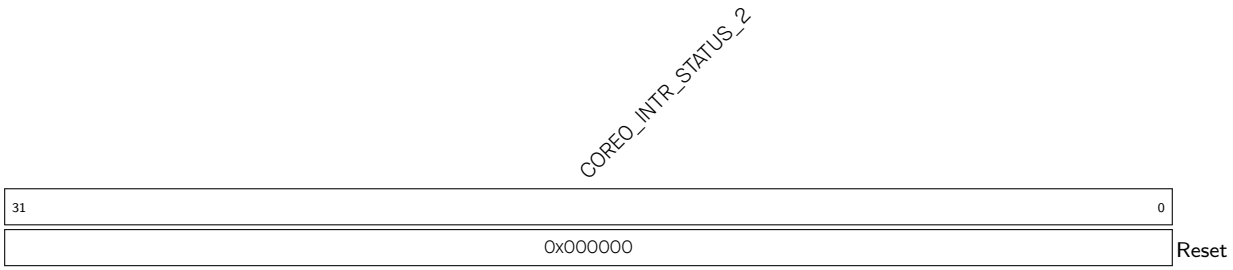
COREO_INTR_STATUS_0 Represents the status of the interrupt sources from 0 ~ 31. Each bit corresponds to one interrupt source. (RO)

Register 13.131. COREO_INTR_STATUS_REG_1_REG (0x0204)



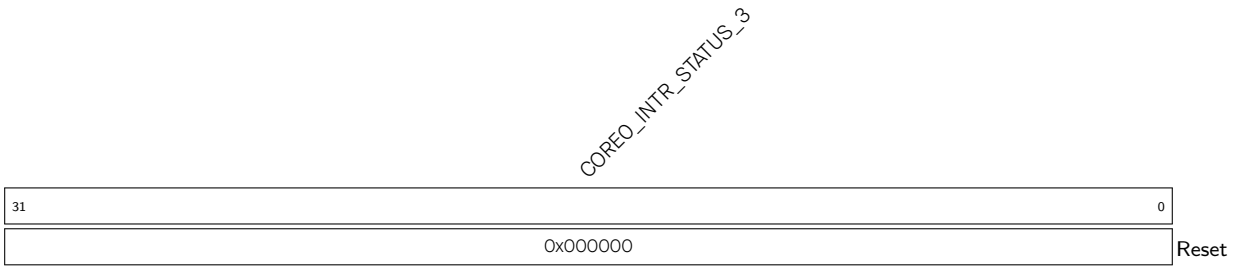
COREO_INTR_STATUS_1 Represents the status of the interrupt sources from 32 ~ 63. Each bit corresponds to one interrupt source. (RO)

Register 13.132. COREO_INTR_STATUS_REG_2_REG (0x0208)



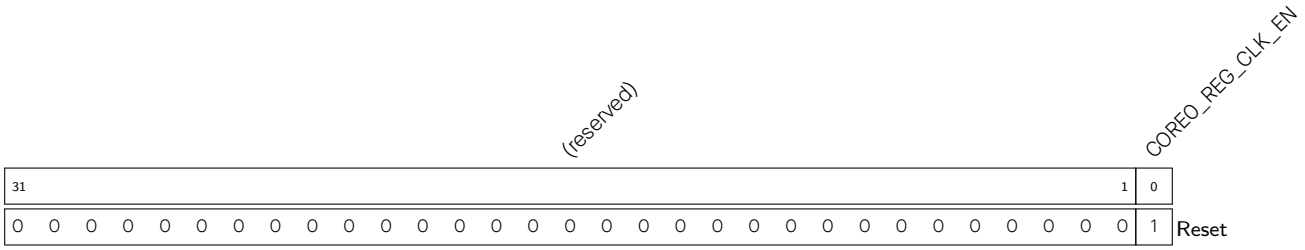
COREO_INTR_STATUS_2 Represents the status of the interrupt sources from 64 ~ 95. Each bit corresponds to one interrupt source. (RO)

Register 13.133. COREO_INTR_STATUS_REG_3_REG (0x020C)



COREO_INTR_STATUS_3 Represents the status of the interrupt sources numbered from 96 ~ 127 (99 and 112 are reserved). Each bit corresponds to one interrupt source. (RO)

Register 13.134. COREO_CLOCK_GATE_REG (0x0210)



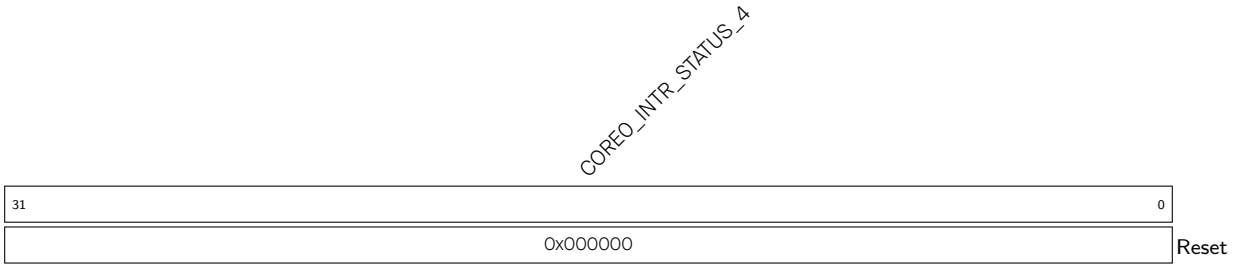
COREO_REG_CLK_EN Configures whether to force interrupt register clock-gate on.

0: No effect

1: Force interrupt register clock-gate on

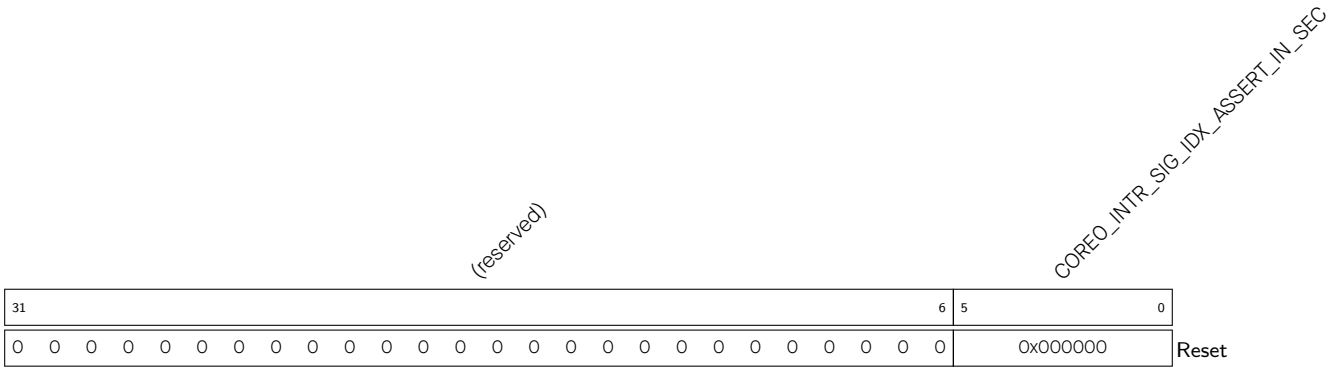
(R/W)

Register 13.135. COREO_INTR_STATUS_REG_4_REG (0x0220)



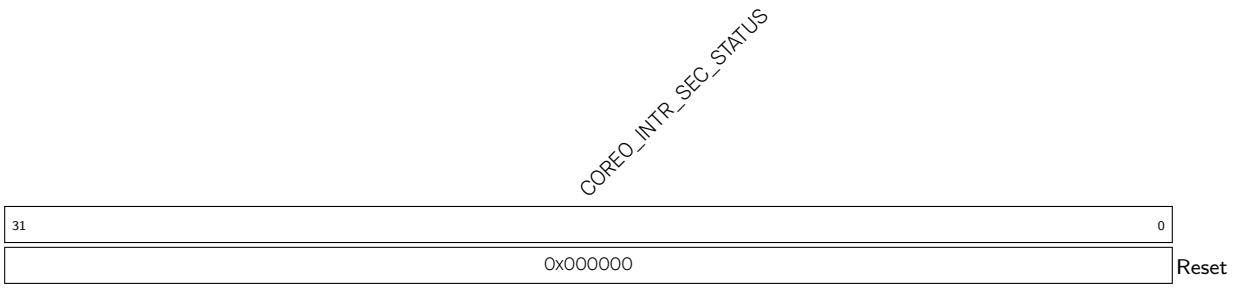
COREO_INTR_STATUS_4 Represents the status of the interrupt sources from 128 ~ 130. Each bit corresponds to one interrupt source. (RO)

Register 13.136. COREO_INTR_SIG_IDX_ASSERT_IN_SEC_REG (0x0228)



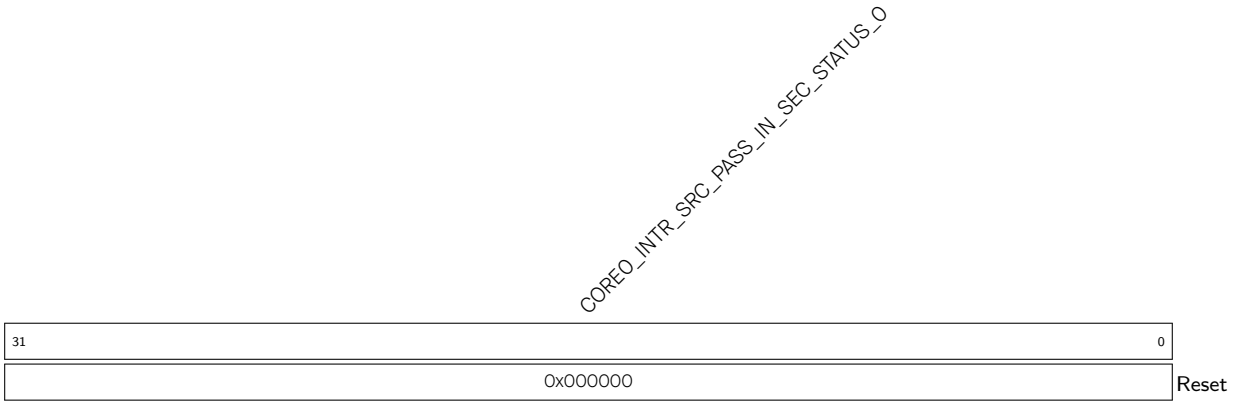
COREO_INTR_SIG_IDX_ASSERT_IN_SEC Configures which peripheral machine mode interrupt of CPU0 is used as the target for the interrupt delegation function. Valid values range from 16 ~ 47. (R/W)

Register 13.137. COREO_INTR_SEC_STATUS_REG (0x022C)



COREO_INTR_SEC_STATUS Represents status of the user mode interrupt signals that have been delegated to the machine mode interrupt when COREO is in machine mode. (RO)

Register 13.138. COREO_INTR_SRC_PASS_IN_SEC_STATUS_0_REG (0x0230)



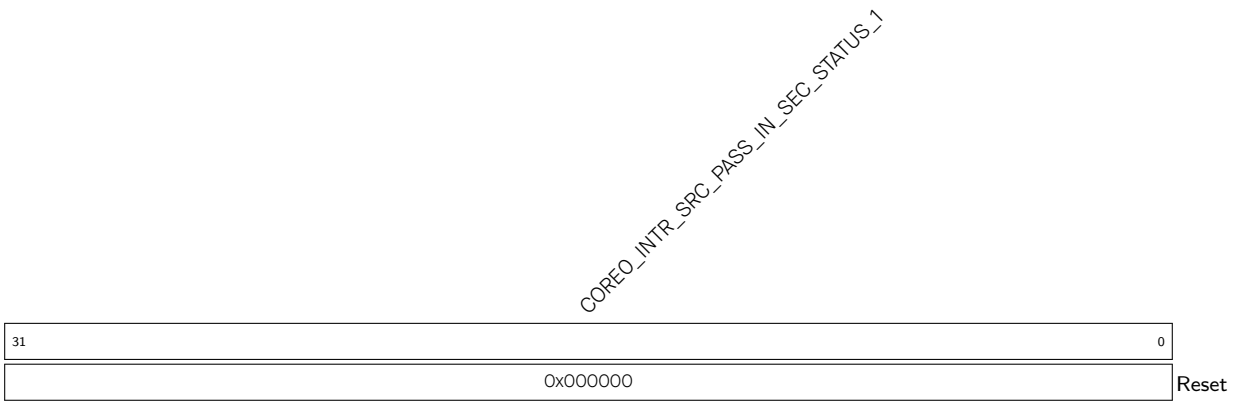
COREO_INTR_SRC_PASS_IN_SEC_STATUS_0 Represents whether interrupt source 0 ~ 31 has enabled the interrupt delegation function. Each bit corresponds to the enable status of one interrupt source.

0: Corresponding interrupt source has not enabled the interrupt delegation function.

1: Corresponding interrupt source has enabled the interrupt delegation function.

(RO)

Register 13.139. COREO_INTR_SRC_PASS_IN_SEC_STATUS_1_REG (0x0234)



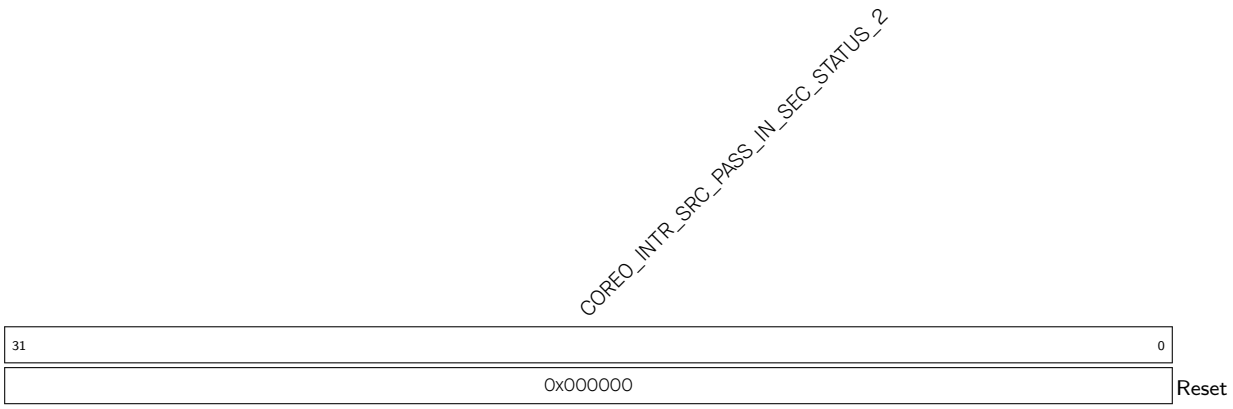
COREO_INTR_SRC_PASS_IN_SEC_STATUS_1 Represents whether interrupt sources 32 ~ 63 have enabled the interrupt delegation function. Each bit corresponds to the enable status of one interrupt source.

0: Corresponding interrupt source has not enabled the interrupt delegation function.

1: Corresponding interrupt source has enabled the interrupt delegation function.

(RO)

Register 13.140. COREO_INTR_SRC_PASS_IN_SEC_STATUS_2_REG (0x0238)



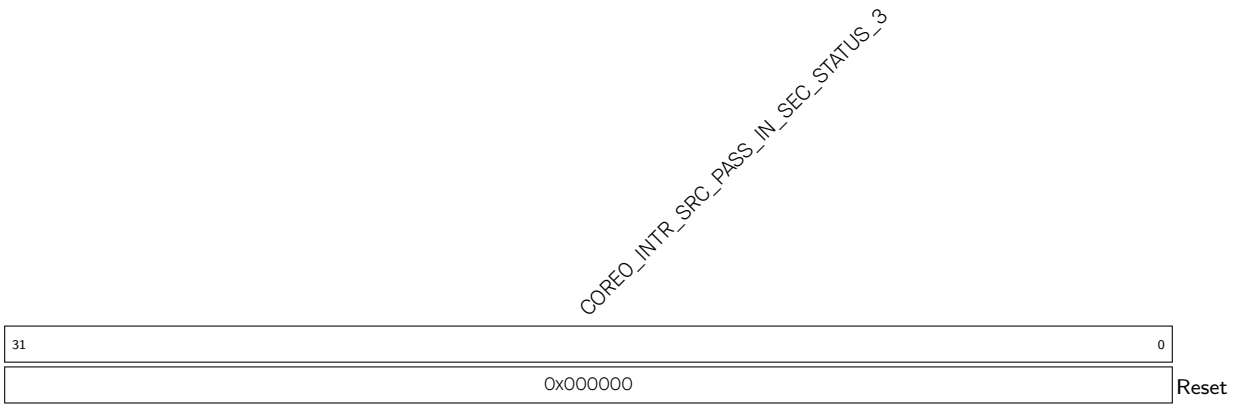
COREO_INTR_SRC_PASS_IN_SEC_STATUS_2 Represents whether interrupt source 64 ~ 95 has enabled the interrupt delegation function. Each bit corresponds to the enable status of one interrupt source.

0: Corresponding interrupt source has not enabled the interrupt delegation function.

1: Corresponding interrupt source has enabled the interrupt delegation function.

(RO)

Register 13.141. COREO_INTR_SRC_PASS_IN_SEC_STATUS_3_REG (0x023C)



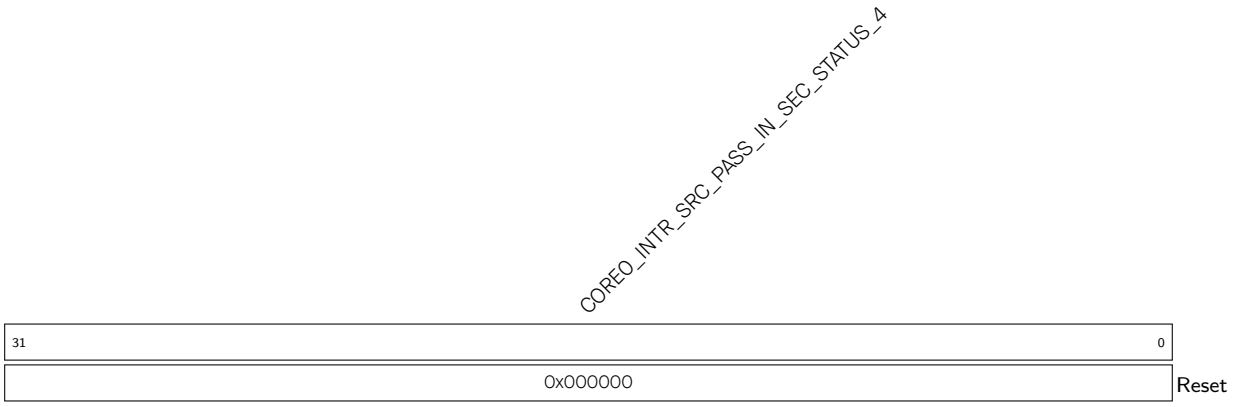
COREO_INTR_SRC_PASS_IN_SEC_STATUS_3 Represents whether interrupt source 96 ~ 127 has enabled the interrupt delegation function. Each bit corresponds to the enable status of one interrupt source.

0: Corresponding interrupt source has not enabled the interrupt delegation function.

1: Corresponding interrupt source has enabled the interrupt delegation function.

(RO)

Register 13.142. COREO_INTR_SRC_PASS_IN_SEC_STATUS_4_REG (0x0240)



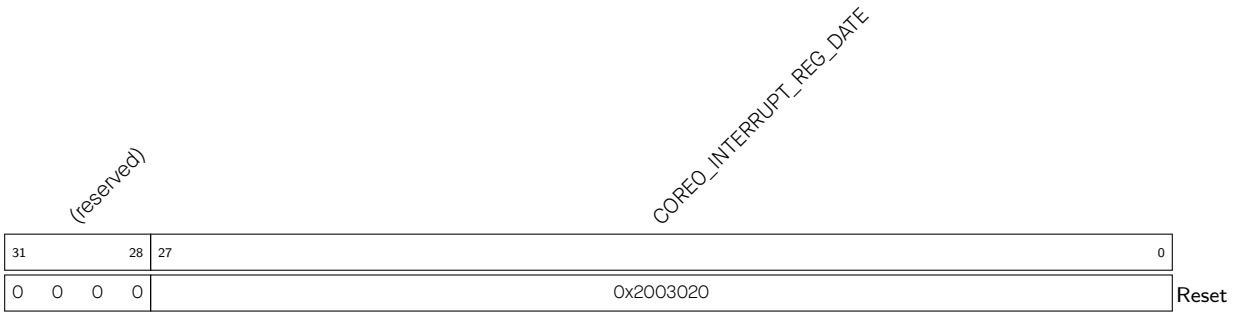
COREO_INTR_SRC_PASS_IN_SEC_STATUS_4 Represents whether interrupt source 128 ~ 130 has enabled the interrupt delegation function. Each bit corresponds to the enable status of one interrupt source.

0: Corresponding interrupt source has not enabled the interrupt delegation function.

1: Corresponding interrupt source has enabled the interrupt delegation function.

(RO)

Register 13.143. COREO_INTERRUPT_REG_DATE_REG (0x03FC)



COREO_INTERRUPT_REG_DATE Version control register. (R/W)

13.6.2 HP CPU1 Interrupt Matrix Registers

Register 13.144. CORE1_*LP_RTC_INT*_MAP_REG (0x0000)

Register 13.145. CORE1_*LP_WDT_INT*_MAP_REG (0x0004)

Register 13.146. CORE1_*LP_TIMER_REG_0_INT*_MAP_REG (0x0008)

Register 13.147. CORE1_*LP_TIMER_REG_1_INT*_MAP_REG (0x000C)

Register 13.148. CORE1_*MB_HP_INT*_MAP_REG (0x0010)

Register 13.149. CORE1_*MB_LP_INT*_MAP_REG (0x0014)

Register 13.150. CORE1_*PMU_REG_0_INT*_MAP_REG (0x0018)

Register 13.151. CORE1_*PMU_REG_1_INT*_MAP_REG (0x001C)

Register 13.152. CORE1_*LP_ANAPERI_INT*_MAP_REG (0x0020)

Register 13.153. CORE1_*LP_ADC_INT*_MAP_REG (0x0024)

Register 13.154. CORE1_*LP_GPIO_INT*_MAP_REG (0x0028)

Register 13.155. CORE1_*LP_I2C_INT*_MAP_REG (0x002C)

Register 13.156. CORE1_*LP_I2S_INT*_MAP_REG (0x0030)

Register 13.157. CORE1_*LP_SPI_INT*_MAP_REG (0x0034)

Register 13.158. CORE1_*LP_TOUCH_INT*_MAP_REG (0x0038)

Register 13.159. CORE1_*LP_TSENS_INT*_MAP_REG (0x003C)

Register 13.160. CORE1_*LP_UART_INT*_MAP_REG (0x0040)

Register 13.161. CORE1_*LP_EFUSE_INT*_MAP_REG (0x0044)

Register 13.162. CORE1_*LP_SW_INT*_MAP_REG (0x0048)

Register 13.163. CORE1_*LP_SYSREG_INT*_MAP_REG (0x004C)

Register 13.164. CORE1_*LP_HUK_INT*_MAP_REG (0x0050)

Register 13.165. CORE1_*SYS_ICM_INT*_MAP_REG (0x0054)

Register 13.166. CORE1_*USB_DEVICE_INT*_MAP_REG (0x0058)

Register 13.167. CORE1_SDIO_HOST_INT_MAP_REG (0x005C)

Register 13.168. CORE1_GDMA_INT_MAP_REG (0x0060)

Register 13.169. CORE1_SPI2_INT_MAP_REG (0x0064)

Register 13.170. CORE1_SPI3_INT_MAP_REG (0x0068)

Register 13.171. CORE1_I2S0_INT_MAP_REG (0x006C)

Register 13.172. CORE1_I2S1_INT_MAP_REG (0x0070)

Register 13.173. CORE1_I2S2_INT_MAP_REG (0x0074)

Register 13.174. CORE1_UHCI0_INT_MAP_REG (0x0078)

Register 13.175. CORE1_UART0_INT_MAP_REG (0x007C)

Register 13.176. CORE1_UART1_INT_MAP_REG (0x0080)

Register 13.177. CORE1_UART2_INT_MAP_REG (0x0084)

Register 13.178. CORE1_UART3_INT_MAP_REG (0x0088)

Register 13.179. CORE1_UART4_INT_MAP_REG (0x008C)

Register 13.180. CORE1_LCD_CAM_INT_MAP_REG (0x0090)

Register 13.181. CORE1_ADC_INT_MAP_REG (0x0094)

Register 13.182. CORE1_PWM0_INT_MAP_REG (0x0098)

Register 13.183. CORE1_PWM1_INT_MAP_REG (0x009C)

Register 13.184. CORE1_TWAI0_INT_MAP_REG (0x00A0)

Register 13.185. CORE1_TWAI1_INT_MAP_REG (0x00A4)

Register 13.186. CORE1_TWAI2_INT_MAP_REG (0x00A8)

Register 13.187. CORE1_RMT_INT_MAP_REG (0x00AC)

Register 13.188. CORE1_I2C0_INT_MAP_REG (0x00B0)

Register 13.189. CORE1_I2C1_INT_MAP_REG (0x00B4)

Register 13.190. CORE1_TIMERGRP0_T0_INT_MAP_REG (0x00B8)

Register 13.191. CORE1_TIMERGRP0_T1_INT_MAP_REG (0x00BC)

Register 13.192. CORE1_TIMERGRP0_WDT_INT_MAP_REG (0x00C0)

Register 13.193. CORE1_TIMERGRP1_T0_INT_MAP_REG (0x00C4)

Register 13.194. CORE1_TIMERGRP1_T1_INT_MAP_REG (0x00C8)

Register 13.195. CORE1_TIMERGRP1_WDT_INT_MAP_REG (0x00CC)

Register 13.196. CORE1_LEDC_INT_MAP_REG (0x00D0)

Register 13.197. CORE1_SYSTIMER_TARGET0_INT_MAP_REG (0x00D4)

Register 13.198. CORE1_SYSTIMER_TARGET1_INT_MAP_REG (0x00D8)

Register 13.199. CORE1_SYSTIMER_TARGET2_INT_MAP_REG (0x00DC)

Register 13.200. CORE1_AHB_PDMA_IN_CH0_INT_MAP_REG (0x00E0)

Register 13.201. CORE1_AHB_PDMA_IN_CH1_INT_MAP_REG (0x00E4)

Register 13.202. CORE1_*AHB_PDMA_IN_CH2_INT*_MAP_REG (0x00E8)

Register 13.203. CORE1_*AHB_PDMA_OUT_CH0_INT*_MAP_REG (0x00EC)

Register 13.204. CORE1_*AHB_PDMA_OUT_CH1_INT*_MAP_REG (0x00F0)

Register 13.205. CORE1_*AHB_PDMA_OUT_CH2_INT*_MAP_REG (0x00F4)

Register 13.206. CORE1_*AXI_PDMA_IN_CH0_INT*_MAP_REG (0x00F8)

Register 13.207. CORE1_*AXI_PDMA_IN_CH1_INT*_MAP_REG (0x00FC)

Register 13.208. CORE1_*AXI_PDMA_IN_CH2_INT*_MAP_REG (0x0100)

Register 13.209. CORE1_*AXI_PDMA_OUT_CH0_INT*_MAP_REG (0x0104)

Register 13.210. CORE1_*AXI_PDMA_OUT_CH1_INT*_MAP_REG (0x0108)

Register 13.211. CORE1_*AXI_PDMA_OUT_CH2_INT*_MAP_REG (0x010C)

Register 13.212. CORE1_*RSA_INT*_MAP_REG (0x0110)

Register 13.213. CORE1_*AES_INT*_MAP_REG (0x0114)

Register 13.214. CORE1_*SHA_INT*_MAP_REG (0x0118)

Register 13.215. CORE1_*ECC_INT*_MAP_REG (0x011C)

Register 13.216. CORE1_*ECDSA_INT*_MAP_REG (0x0120)

Register 13.217. CORE1_*KM_INT*_MAP_REG (0x0124)

Register 13.218. CORE1_*GPIO_INT0*_MAP_REG (0x0128)

Register 13.219. CORE1_*GPIO_INT1*_MAP_REG (0x012C)

Register 13.220. CORE1_*GPIO_INT2*_MAP_REG (0x0130)

Register 13.221. CORE1_*GPIO_INT3*_MAP_REG (0x0134)

Register 13.222. CORE1_*GPIO_PAD_COMP_INT*_MAP_REG (0x0138)

Register 13.223. CORE1_*CPU_INT_FROM_CPU_0*_MAP_REG (0x013C)

Register 13.224. CORE1_*CPU_INT_FROM_CPU_1*_MAP_REG (0x0140)

Register 13.225. CORE1_*CPU_INT_FROM_CPU_2*_MAP_REG (0x0144)

Register 13.226. CORE1_*CPU_INT_FROM_CPU_3*_MAP_REG (0x0148)

Register 13.227. CORE1_*FLASH_MSPI_INT*_MAP_REG (0x0150)

Register 13.228. CORE1_*CSI_BRIDGE_INT*_MAP_REG (0x0154)

Register 13.229. CORE1_*DSI_BRIDGE_INT*_MAP_REG (0x0158)

Register 13.230. CORE1_*CSI_INT*_MAP_REG (0x015C)

Register 13.231. CORE1_*DSI_INT*_MAP_REG (0x0160)

Register 13.232. CORE1_*GMII_PHY_INT*_MAP_REG (0x0164)

Register 13.233. CORE1_*LPI_INT*_MAP_REG (0x0168)

Register 13.234. CORE1_*PMT_INT*_MAP_REG (0x016C)

Register 13.235. CORE1_*ETH_MAC_INT*_MAP_REG (0x0170)

Register 13.236. CORE1_*USB_OTG_INT*_MAP_REG (0x0174)

Register 13.237. CORE1_*USB_OTG_ENDP_MULTI_PROC_INT*_MAP_REG (0x0178)

Register 13.238. CORE1_*JPEG_INT*_MAP_REG (0x017C)

Register 13.239. CORE1_*PPA_INT*_MAP_REG (0x0180)

Register 13.240. CORE1_*COREO_TRACE_INT*_MAP_REG (0x0184)

Register 13.241. CORE1_*CORE1_TRACE_INT*_MAP_REG (0x0188)

Register 13.242. CORE1_*ISP_INT*_MAP_REG (0x0190)

Register 13.243. CORE1_*I3C_MST_INT*_MAP_REG (0x0194)

Register 13.244. CORE1_*I3C_SLV_INT*_MAP_REG (0x0198)

Register 13.245. CORE1_*USB_OTG11_INT*_MAP_REG (0x019C)

Register 13.246. CORE1_*DMA2D_IN_CHO_INT*_MAP_REG (0x01A0)

Register 13.247. CORE1_*DMA2D_IN_CH1_INT*_MAP_REG (0x01A4)

Register 13.248. CORE1_*DMA2D_OUT_CHO_INT*_MAP_REG (0x01A8)

Register 13.249. CORE1_*DMA2D_OUT_CH1_INT*_MAP_REG (0x01AC)

Register 13.250. CORE1_*DMA2D_OUT_CH2_INT*_MAP_REG (0x01B0)

Register 13.251. CORE1_*PSRAM_MSPI_INT*_MAP_REG (0x01B4)

Register 13.252. CORE1_*HP_SYSREG_INT*_MAP_REG (0x01B8)

Register 13.253. CORE1_*PCNT_INT*_MAP_REG (0x01BC)

Register 13.254. CORE1_*HP_PAU_INT*_MAP_REG (0x01C0)

Register 13.255. CORE1_*HP_PARLIO_RX_INT*_MAP_REG (0x01C4)

Register 13.256. CORE1_*HP_PARLIO_TX_INT*_MAP_REG (0x01C8)

Register 13.257. CORE1_*H264_DMA2D_OUT_CHO_INT*_MAP_REG (0x01CC)

Register 13.258. CORE1_*H264_DMA2D_OUT_CH1_INT*_MAP_REG (0x01D0)

Register 13.259. CORE1_*H264_DMA2D_OUT_CH2_INT*_MAP_REG (0x01D4)

Register 13.260. CORE1_*H264_DMA2D_OUT_CH3_INT*_MAP_REG (0x01D8)

Register 13.261. CORE1_*H264_DMA2D_OUT_CH4_INT*_MAP_REG (0x01DC)

Register 13.262. CORE1_*H264_DMA2D_IN_CHO_INT*_MAP_REG (0x01E0)

Register 13.263. CORE1_*H264_DMA2D_IN_CH1_INT*_MAP_REG (0x01E4)

Register 13.264. CORE1_*H264_DMA2D_IN_CH2_INT*_MAP_REG (0x01E8)

Register 13.265. CORE1_*H264_DMA2D_IN_CH3_INT*_MAP_REG (0x01EC)

Register 13.266. CORE1_*H264_DMA2D_IN_CH4_INT*_MAP_REG (0x01F0)

Register 13.267. CORE1_*H264_DMA2D_IN_CH5_INT*_MAP_REG (0x01F4)

Register 13.268. CORE1_*H264_REG_INT*_MAP_REG (0x01F8)

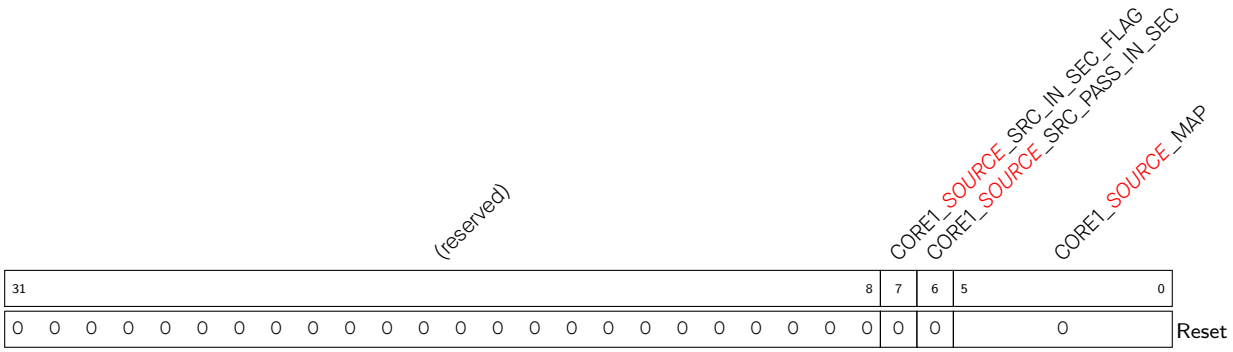
Register 13.269. CORE1_*ASSIST_DEBUG_INT*_MAP_REG (0x01FC)

Register 13.270. CORE1_*DMA2D_IN_CH2_INT*_MAP_REG (0x0214)

Register 13.271. CORE1_*DMA2D_OUT_CH3_INT*_MAP_REG (0x0218)

Register 13.272. CORE1_*AXI_PERF_MON_INT*_MAP_REG (0x021C)

Register 13.273. CORE1_*SOURCE*_MAP_REG (0x0000 - 0x01FC, 0x0214 - 0x021C)



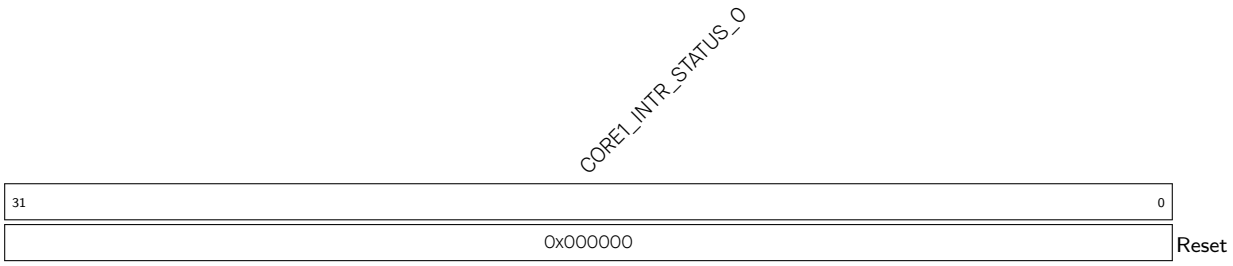
CORE1_*SOURCE*_MAP When CORE1_*SOURCE*_SRC_IN_SEC_FLAG is set to 1, this register can only be written by CPU while in Machine Mode.

Configures this register to map the interrupt signal from the peripheral interrupt sources *Source* to one of CPU1's peripheral interrupts. The valid configuration range is 16 ~ 47.
When set to 0 ~ 15, the corresponding peripheral interrupt source is disabled (recommended value: 0).
Other values are invalid.
For the list of interrupt sources *Source*, see Table 13.4-1. (R/W)

CORE1_*SOURCE*_SRC_PASS_IN_SEC Write 1 to allow the interrupt signal from interrupt source *Source* to be delegated to CPU1's Machine Mode interrupt. Writable only by the CPU in Machine Mode. (R/W)

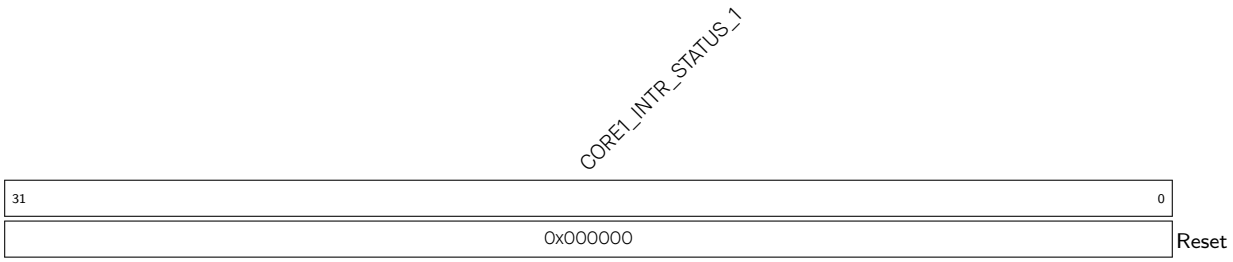
CORE1_*SOURCE*_SRC_IN_SEC_FLAG Write 1 to configure the interrupt signal from interrupt source *Source* as a secure interrupt source (that is, its interrupt mapping can be configured only by the CPU running in Machine Mode). Writable only by the CPU in Machine Mode. (R/W)

Register 13.274. CORE1_INTR_STATUS_REG_0_REG (0x0200)



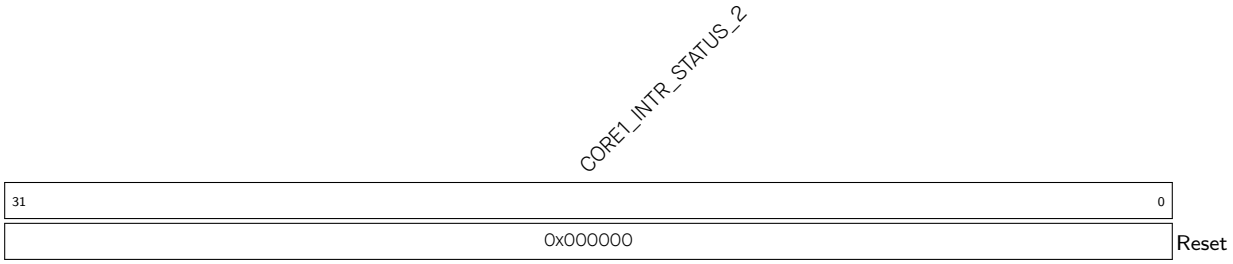
CORE1_INTR_STATUS_0 Represents the status of the interrupt sources from 0 ~ 31. Each bit corresponds to one interrupt source. (RO)

Register 13.275. CORE1_INTR_STATUS_REG_1_REG (0x0204)



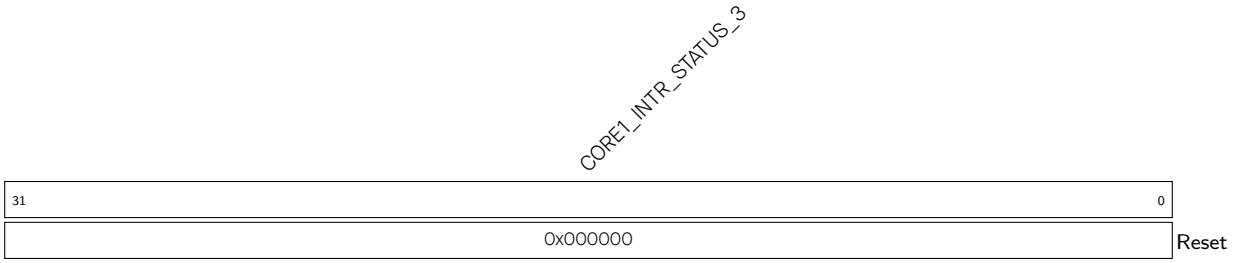
CORE1_INTR_STATUS_1 Represents the status of the interrupt sources from 32 ~ 63. Each bit corresponds to one interrupt source. (RO)

Register 13.276. CORE1_INTR_STATUS_REG_2_REG (0x0208)



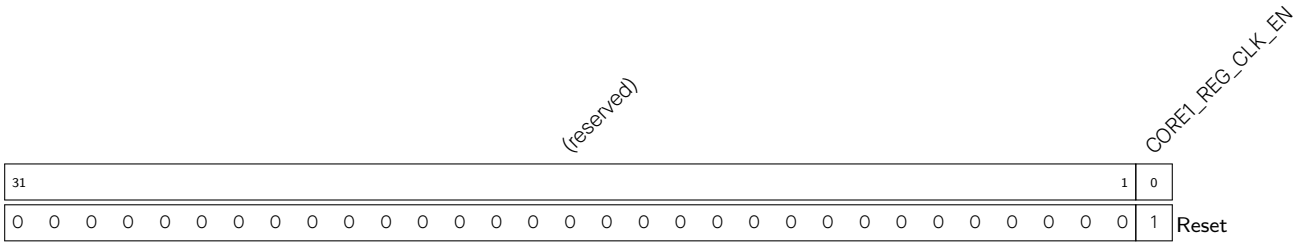
CORE1_INTR_STATUS_2 Represents the status register for interrupt sources from 64 ~ 95. Each bit corresponds to one interrupt source. (RO)

Register 13.277. CORE1_INTR_STATUS_REG_3_REG (0x020C)



CORE1_INTR_STATUS_3 Represents the status of the interrupt sources from 96 ~ 127. Each bit corresponds to one interrupt source. (RO)

Register 13.278. CORE1_CLOCK_GATE_REG (0x0210)



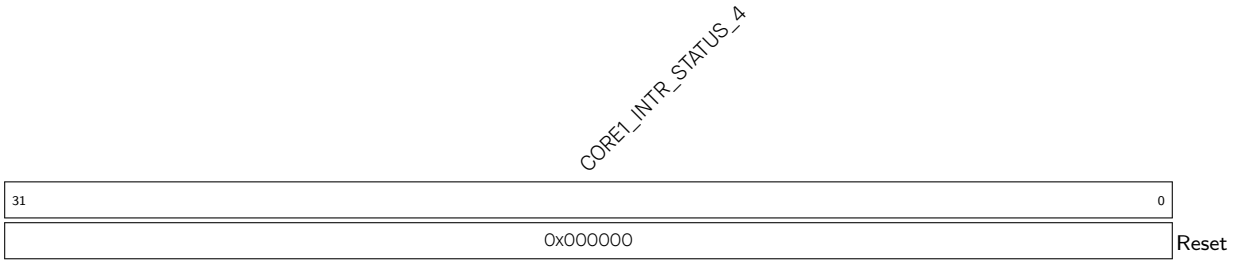
CORE1_REG_CLK_EN Configures whether to force interrupt register clock-gate on.

0: No effect

1: Force interrupt register clock-gate on

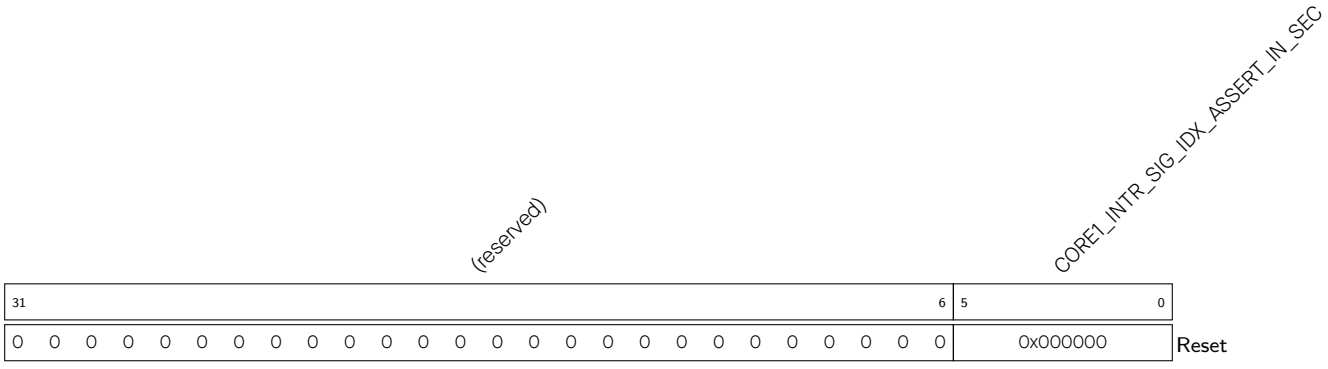
(R/W)

Register 13.279. CORE1_INTR_STATUS_REG_4_REG (0x0220)



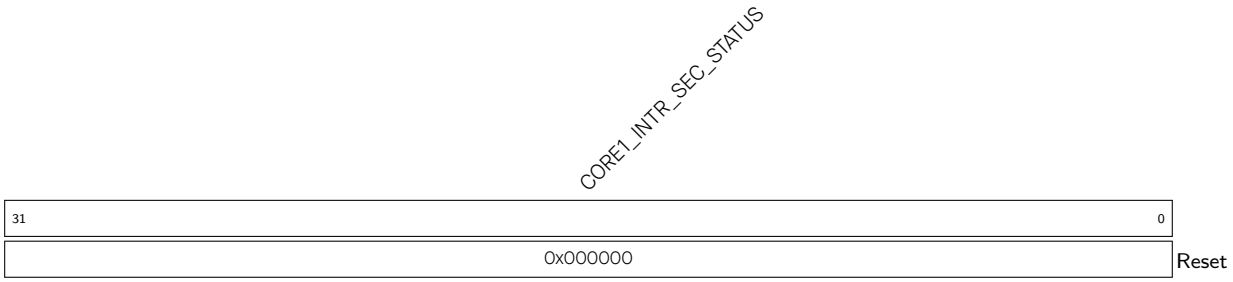
CORE1_INTR_STATUS_4 Represents the status of the interrupt sources from 128 ~ 130. Each bit corresponds to one interrupt source. (RO)

Register 13.280. CORE1_INTR_SIG_IDX_ASSERT_IN_SEC_REG (0x0228)



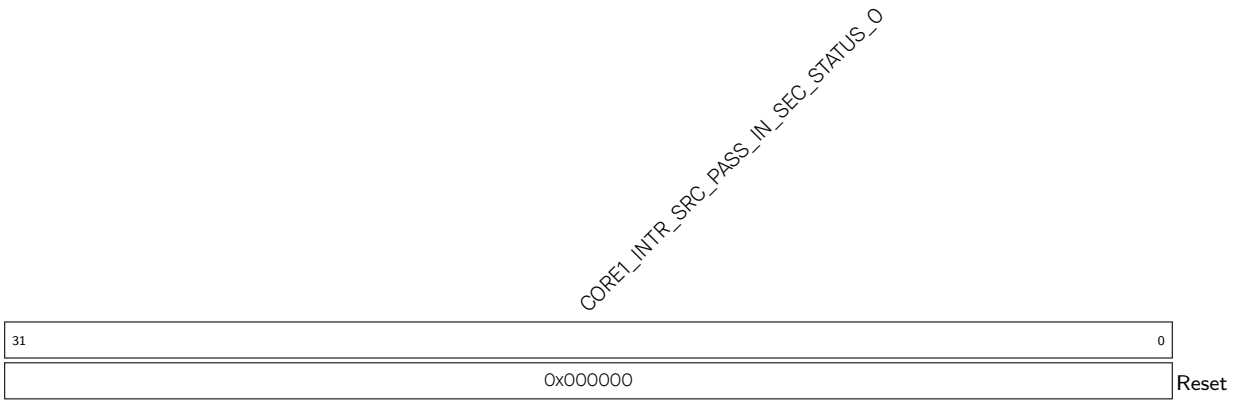
CORE1_INTR_SIG_IDX_ASSERT_IN_SEC Configures which peripheral machine mode interrupt of CPU1 is used as the target for the interrupt delegation function. Valid values range from 16 ~ 47. (R/W)

Register 13.281. CORE1_INTR_SEC_STATUS_REG (0x022C)



CORE1_INTR_SEC_STATUS Represents status of the user mode interrupt signals that have been delegated to the machine mode interrupt when CORE1 is in machine mode. (RO)

Register 13.282. CORE1_INTR_SRC_PASS_IN_SEC_STATUS_0_REG (0x0230)



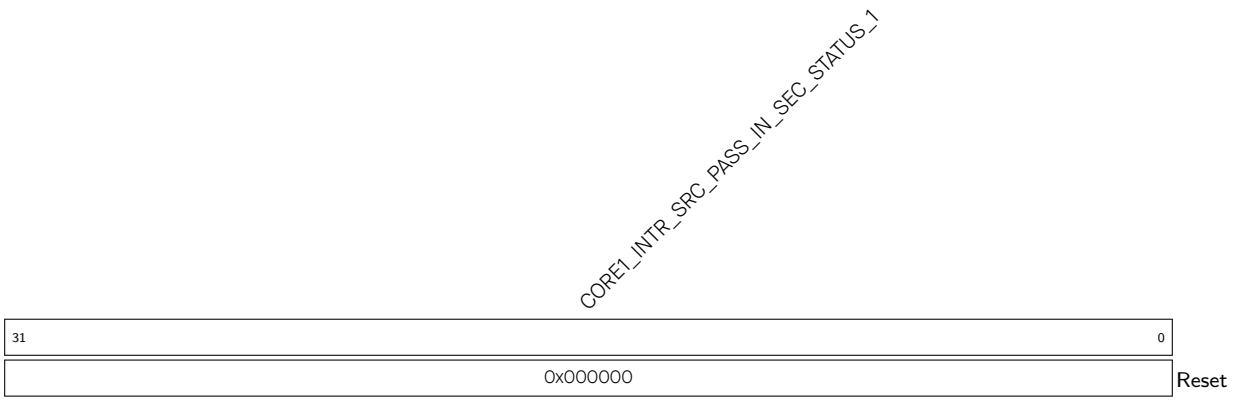
CORE1_INTR_SRC_PASS_IN_SEC_STATUS_0 Represents whether interrupt source 0 ~ 31 has enabled the interrupt delegation function. Each bit corresponds to the enable status of one interrupt source.

0: Corresponding interrupt source has not enabled the interrupt delegation function.

1: Corresponding interrupt source has enabled the interrupt delegation function.

(RO)

Register 13.283. CORE1_INTR_SRC_PASS_IN_SEC_STATUS_1_REG (0x0234)



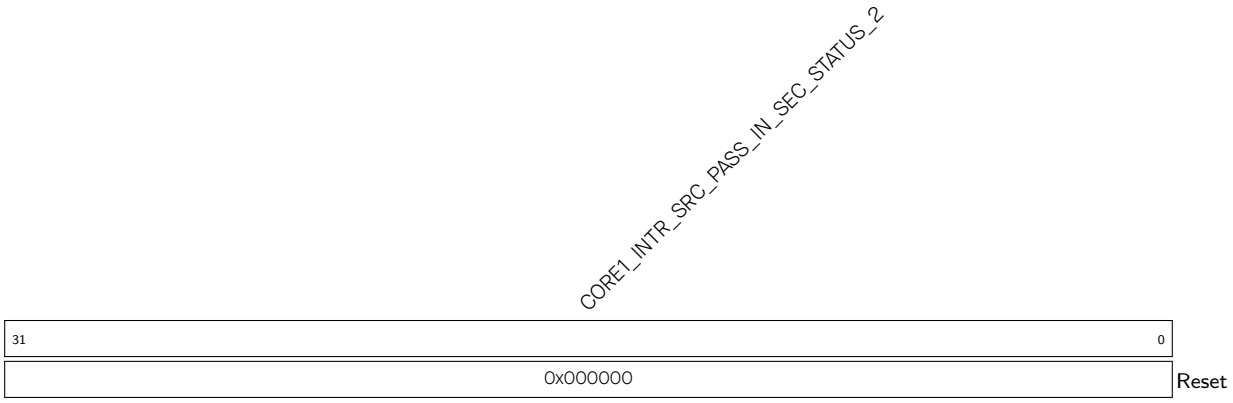
CORE1_INTR_SRC_PASS_IN_SEC_STATUS_1 Represents whether interrupt source 32 ~ 63 has enabled the interrupt delegation function. Each bit corresponds to the enable status of one interrupt source.

0: Corresponding interrupt source has not enabled the interrupt delegation function.

1: Corresponding interrupt source has enabled the interrupt delegation function.

(RO)

Register 13.284. CORE1_INTR_SRC_PASS_IN_SEC_STATUS_2_REG (0x0238)



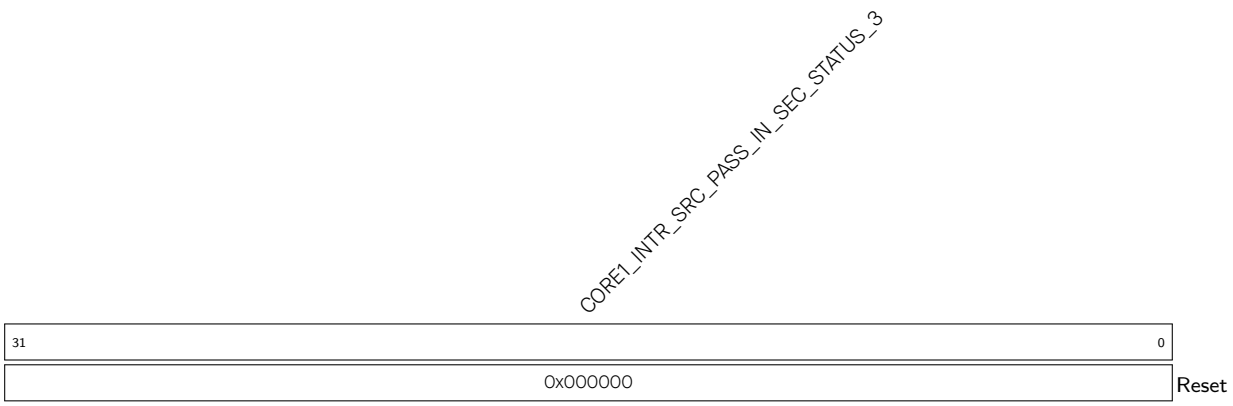
CORE1_INTR_SRC_PASS_IN_SEC_STATUS_2 Represents whether interrupt source 64 ~ 95 has enabled the interrupt delegation function. Each bit corresponds to the enable status of one interrupt source.

0: Corresponding interrupt source has not enabled the interrupt delegation function.

1: Corresponding interrupt source has enabled the interrupt delegation function.

(RO)

Register 13.285. CORE1_INTR_SRC_PASS_IN_SEC_STATUS_3_REG (0x023C)



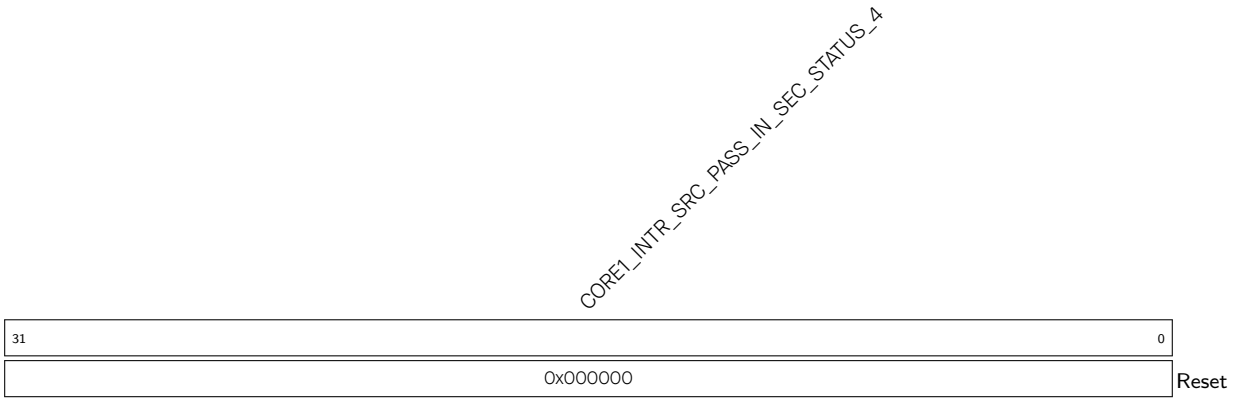
CORE1_INTR_SRC_PASS_IN_SEC_STATUS_3 Represents whether interrupt source 96 ~ 127 has enabled the interrupt delegation function. Each bit corresponds to the enable status of one interrupt source.

0: Corresponding interrupt source has not enabled the interrupt delegation function.

1: Corresponding interrupt source has enabled the interrupt delegation function.

(RO)

Register 13.286. CORE1_INTR_SRC_PASS_IN_SEC_STATUS_4_REG (0x0240)



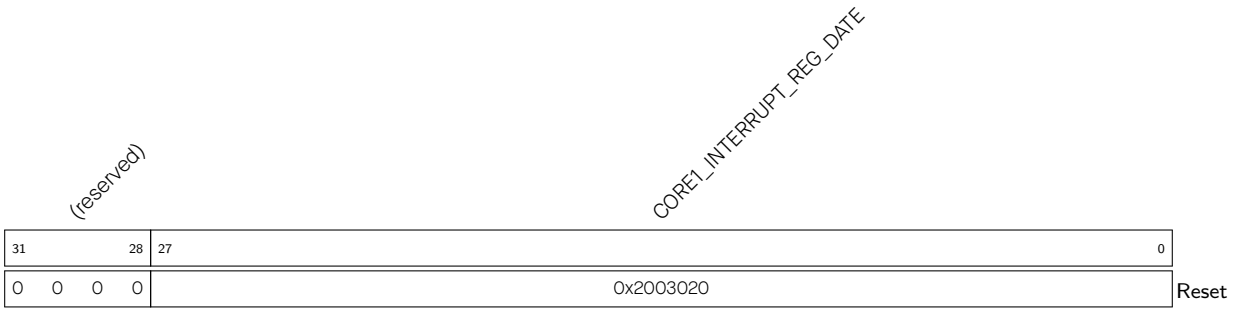
CORE1_INTR_SRC_PASS_IN_SEC_STATUS_4 Represents whether interrupt source 128 ~ 130 has enabled the interrupt delegation function. Each bit corresponds to the enable status of one interrupt source.

0: Corresponding interrupt source has not enabled the interrupt delegation function.

1: Corresponding interrupt source has enabled the interrupt delegation function.

(RO)

Register 13.287. CORE1_INTERRUPT_REG_DATE_REG (0x03FC)



CORE1_INTERRUPT_REG_DATE Version control register. (R/W)

Chapter 14

Event Task Matrix (ETM)

14.1 Overview

The Event Task Matrix (ETM) peripheral contains 50 configurable channels. Each channel can map an event of any specified peripheral to a task of any specified peripheral. In this way, peripherals can be triggered to execute specified tasks without CPU intervention.

14.2 Features

The Event Task Matrix has the following features:

- Receive various events from multiple peripherals
- Generate various tasks for multiple peripherals
- 50 independently configurable ETM channels
- An ETM channel can be set up to receive any event, and map it to any task
- Each ETM channel can be enabled independently. If not enabled, the channel will not respond to the configured event and generate the task mapped to that event
- Support for checking event and task status
- Peripherals supporting ETM include GPIO, LED PWM, general-purpose timers, RTC Timer, system timer, MCPWM, temperature sensor, ADC, I2S, LP CPU, GDMA-AHB, GDMA-AXI, 2D DMA, and PMU

Note that the 50 ETM channels are identical regarding their features and operations. Thus, in the following sections, ETM channels are collectively referred to as channel*n* (where *n* ranges from 0 to 49).

14.3 Functional Description

14.3.1 Architecture

The Event Task Matrix has 50 independent channels. A channel can choose any event as input, and map the event to any task as output (see Section 14.3.2 and Section 14.3.3 respectively). Each channel has an individual enable bit (see Section 14.3.6).

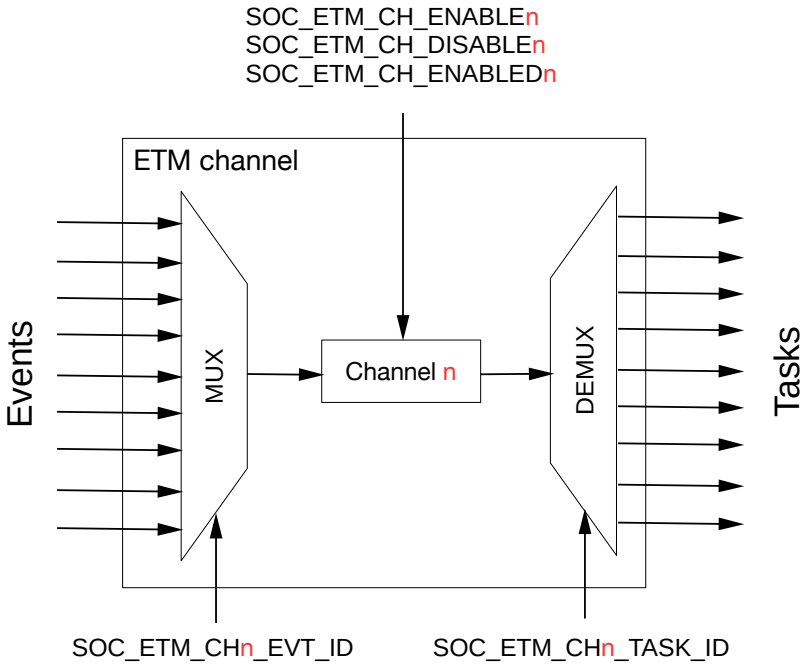


Figure 14.3-1. ETM Channel*n* Architecture

Figure 14.3-1 illustrates the structure of an ETM channel. The SOC_ETM_CH*n*_EVT_ID field configures the MUX (multiplexer) to select one of the events as the input of channel*n*. The SOC_ETM_CH*n*_TASK_ID field configures the DEMUX (demultiplexer) to map the event selected by channel*n* to one of the tasks. SOC_ETM_CH_ENABLE*n* and SOC_ETM_CH_DISABLE*n* are used to enable or disable channel*n*. SOC_ETM_CH_ENABLED*n* is used to indicate the status of the channel*n*.

14.3.2 Events

An ETM channel can be set up to choose which event to receive by configuring the SOC_ETM_CH*n*_EVT_ID field. Table 14.3-1 shows the configuration values of SOC_ETM_CH*n*_EVT_ID and their corresponding events.

Table 14.3-1. Selectable Events for ETM Channel*n*

SOC_ETM_CH*n*_EVT_ID	Selected Event	Peripheral Generating This Event
1	GPIO_EVT_CH0_RISE_EDGE	GPIO
2	GPIO_EVT_CH1_RISE_EDGE	
3	GPIO_EVT_CH2_RISE_EDGE	
4	GPIO_EVT_CH3_RISE_EDGE	
5	GPIO_EVT_CH4_RISE_EDGE	
6	GPIO_EVT_CH5_RISE_EDGE	
7	GPIO_EVT_CH6_RISE_EDGE	
8	GPIO_EVT_CH7_RISE_EDGE	
9	GPIO_EVT_CH0_FALL_EDGE	
10	GPIO_EVT_CH1_FALL_EDGE	

SOC_ETM_CH*n*_EVT_ID	Selected Event	Peripheral Generating This Event
11	GPIO_EVT_CH2_FALL_EDGE	
12	GPIO_EVT_CH3_FALL_EDGE	
13	GPIO_EVT_CH4_FALL_EDGE	
14	GPIO_EVT_CH5_FALL_EDGE	
15	GPIO_EVT_CH6_FALL_EDGE	
16	GPIO_EVT_CH7_FALL_EDGE	
17	GPIO_EVT_CH0_ANY_EDGE	
18	GPIO_EVT_CH1_ANY_EDGE	
19	GPIO_EVT_CH2_ANY_EDGE	
20	GPIO_EVT_CH3_ANY_EDGE	
21	GPIO_EVT_CH4_ANY_EDGE	
22	GPIO_EVT_CH5_ANY_EDGE	
23	GPIO_EVT_CH6_ANY_EDGE	
24	GPIO_EVT_CH7_ANY_EDGE	
25	GPIO_EVT_ZERO_DET_POS0	
26	GPIO_EVT_ZERO_DET_NEG0	
27	GPIO_EVT_ZERO_DET_POS1	
28	GPIO_EVT_ZERO_DET_NEG1	
29	LEDC_EVT_DUTY_CHNG_END_CH0	LED PWM Controller (LEDC)
30	LEDC_EVT_DUTY_CHNG_END_CH1	
31	LEDC_EVT_DUTY_CHNG_END_CH2	
32	LEDC_EVT_DUTY_CHNG_END_CH3	
33	LEDC_EVT_DUTY_CHNG_END_CH4	
34	LEDC_EVT_DUTY_CHNG_END_CH5	
35	LEDC_EVT_DUTY_CHNG_END_CH6	
36	LEDC_EVT_DUTY_CHNG_END_CH7	
37	LEDC_EVT_OVF_CNT_PLS_CH0	
38	LEDC_EVT_OVF_CNT_PLS_CH1	
39	LEDC_EVT_OVF_CNT_PLS_CH2	
40	LEDC_EVT_OVF_CNT_PLS_CH3	
41	LEDC_EVT_OVF_CNT_PLS_CH4	
42	LEDC_EVT_OVF_CNT_PLS_CH5	
43	LEDC_EVT_OVF_CNT_PLS_CH6	
44	LEDC_EVT_OVF_CNT_PLS_CH7	
45	LEDC_EVT_TIME_OVF_TIMER0	
46	LEDC_EVT_TIME_OVF_TIMER1	
47	LEDC_EVT_TIME_OVF_TIMER2	
48	LEDC_EVT_TIME_OVF_TIMER3	
49	LEDC_EVT_TIMER0_CMP	
50	LEDC_EVT_TIMER1_CMP	
51	LEDC_EVT_TIMER2_CMP	
52	LEDC_EVT_TIMER3_CMP	
53	TG0_EVT_CNT_CMP_TIMER0	General-purpose timer group 0
55	TG1_EVT_CNT_CMP_TIMER0	General-purpose timer group 1
57	SYSTIMER_EVT_CNT_CMP0	System Timer

SOC_ETM_CH*n*_EVT_ID	Selected Event	Peripheral Generating This Event
58	SYSTIMER_EVT_CNT_CMP1	
59	SYSTIMER_EVT_CNT_CMP2	
60	MCPWM0_EVT_TIMER0_STOP	MCPWM0
61	MCPWM0_EVT_TIMER1_STOP	
62	MCPWM0_EVT_TIMER2_STOP	
63	MCPWM0_EVT_TIMER0_TEZ	
64	MCPWM0_EVT_TIMER1_TEZ	
65	MCPWM0_EVT_TIMER2_TEZ	
66	MCPWM0_EVT_TIMER0_TEP	
67	MCPWM0_EVT_TIMER1_TEP	
68	MCPWM0_EVT_TIMER2_TEP	
69	MCPWM0_EVT_OP0_TEA	
70	MCPWM0_EVT_OP1_TEA	
71	MCPWM0_EVT_OP2_TEA	
72	MCPWM0_EVT_OP0_TEB	
73	MCPWM0_EVT_OP1_TEB	
74	MCPWM0_EVT_OP2_TEB	
75	MCPWM0_EVT_F0	
76	MCPWM0_EVT_F1	
77	MCPWM0_EVT_F2	
78	MCPWM0_EVT_F0_CLR	
79	MCPWM0_EVT_F1_CLR	
80	MCPWM0_EVT_F2_CLR	
81	MCPWM0_EVT_TZ0_CBC	
82	MCPWM0_EVT_TZ1_CBC	
83	MCPWM0_EVT_TZ2_CBC	
84	MCPWM0_EVT_TZ0_OST	
85	MCPWM0_EVT_TZ1_OST	
86	MCPWM0_EVT_TZ2_OST	
87	MCPWM0_EVT_CAP0	
88	MCPWM0_EVT_CAP1	
89	MCPWM0_EVT_CAP2	
90	MCPWM0_EVT_OP0_TEE1	
91	MCPWM0_EVT_OP1_TEE1	
92	MCPWM0_EVT_OP2_TEE1	
93	MCPWM0_EVT_OP0_TEE2	
94	MCPWM0_EVT_OP1_TEE2	
95	MCPWM0_EVT_OP2_TEE2	
96	MCPWM1_EVT_TIMER0_STOP	MCPWM1
97	MCPWM1_EVT_TIMER1_STOP	
98	MCPWM1_EVT_TIMER2_STOP	
99	MCPWM1_EVT_TIMER0_TEZ	
100	MCPWM1_EVT_TIMER1_TEZ	
101	MCPWM1_EVT_TIMER2_TEZ	
102	MCPWM1_EVT_TIMER0_TEP	

SOC_ETM_CH*n*_EVT_ID	Selected Event	Peripheral Generating This Event
103	MCPWM1_EVT_TIMER1_TEP	
104	MCPWM1_EVT_TIMER2_TEP	
105	MCPWM1_EVT_OP0_TEA	
106	MCPWM1_EVT_OP1_TEA	
107	MCPWM1_EVT_OP2_TEA	
108	MCPWM1_EVT_OP0_TEB	
109	MCPWM1_EVT_OP1_TEB	
110	MCPWM1_EVT_OP2_TEB	
111	MCPWM1_EVT_F0	
112	MCPWM1_EVT_F1	
113	MCPWM1_EVT_F2	
114	MCPWM1_EVT_F0_CLR	
115	MCPWM1_EVT_F1_CLR	
116	MCPWM1_EVT_F2_CLR	
117	MCPWM1_EVT_TZ0_CBC	
118	MCPWM1_EVT_TZ1_CBC	
119	MCPWM1_EVT_TZ2_CBC	
120	MCPWM1_EVT_TZ0_OST	
121	MCPWM1_EVT_TZ1_OST	
122	MCPWM1_EVT_TZ2_OST	
123	MCPWM1_EVT_CAP0	
124	MCPWM1_EVT_CAP1	
125	MCPWM1_EVT_CAP2	
126	MCPWM1_EVT_OP0_TEE1	
127	MCPWM1_EVT_OP1_TEE1	
128	MCPWM1_EVT_OP2_TEE1	
129	MCPWM1_EVT_OP0_TEE2	
130	MCPWM1_EVT_OP1_TEE2	
131	MCPWM1_EVT_OP2_TEE2	
132	ADC_EVT_CONV_CMPLT0	ADC Controller (ADC)
133	ADC_EVT_EQ_ABOVE_THRESH0	
134	ADC_EVT_EQ_ABOVE_THRESH1	
135	ADC_EVT_EQ_BELOW_THRESH0	
136	ADC_EVT_EQ_BELOW_THRESH1	
138	ADC_EVT_STOPPED0	
139	ADC_EVT_STARTED0	
148	TMPSNSR_EVT_OVER_LIMIT	Temperature Sensor (TSENS)
149	I2S0_EVT_RX_DONE	I2S0
150	I2S0_EVT_TX_DONE	
151	I2S0_EVT_X_WORDS_RECEIVED	
152	I2S0_EVT_X_WORDS_SENT	
153	I2S1_EVT_RX_DONE	I2S1
154	I2S1_EVT_TX_DONE	
155	I2S1_EVT_X_WORDS_RECEIVED	

SOC_ETM_CH*n*_EVT_ID	Selected Event	Peripheral Generating This Event
156	I2S1_EVT_X_WORDS_SENT	
157	I2S2_EVT_RX_DONE	I2S2
158	I2S2_EVT_TX_DONE	
159	I2S2_EVT_X_WORDS_RECEIVED	
160	I2S2_EVT_X_WORDS_SENT	
161	ULP_EVT_ERR_INTR	Low-Power CPU
162	ULP_EVT_HALT	
163	ULP_EVT_START_INTR	
164	RTC_EVT_TICK	RTC Timer
165	RTC_EVT_OVF	
166	RTC_EVT_CMP	
167	GDMA_AHB_EVT_IN_DONE_CH0	GDMA-AHB
168	GDMA_AHB_EVT_IN_DONE_CH1	
169	GDMA_AHB_EVT_IN_DONE_CH2	
170	GDMA_AHB_EVT_IN_SUC_EOF_CH0	
171	GDMA_AHB_EVT_IN_SUC_EOF_CH1	
172	GDMA_AHB_EVT_IN_SUC_EOF_CH2	
173	GDMA_AHB_EVT_IN_FIFO_EMPTY_CH0	
174	GDMA_AHB_EVT_IN_FIFO_EMPTY_CH1	
175	GDMA_AHB_EVT_IN_FIFO_EMPTY_CH2	
176	GDMA_AHB_EVT_IN_FIFO_FULL_CH0	
177	GDMA_AHB_EVT_IN_FIFO_FULL_CH1	
178	GDMA_AHB_EVT_IN_FIFO_FULL_CH2	
179	GDMA_AHB_EVT_OUT_DONE_CH0	
180	GDMA_AHB_EVT_OUT_DONE_CH1	
181	GDMA_AHB_EVT_OUT_DONE_CH2	
182	GDMA_AHB_EVT_OUT_EOF_CH0	
183	GDMA_AHB_EVT_OUT_EOF_CH1	
184	GDMA_AHB_EVT_OUT_EOF_CH2	
185	GDMA_AHB_EVT_OUT_TOTAL_EOF_CH0	
186	GDMA_AHB_EVT_OUT_TOTAL_EOF_CH1	
187	GDMA_AHB_EVT_OUT_TOTAL_EOF_CH2	
188	GDMA_AHB_EVT_OUT_FIFO_EMPTY_CH0	
189	GDMA_AHB_EVT_OUT_FIFO_EMPTY_CH1	
190	GDMA_AHB_EVT_OUT_FIFO_EMPTY_CH2	
191	GDMA_AHB_EVT_OUT_FIFO_FULL_CH0	
192	GDMA_AHB_EVT_OUT_FIFO_FULL_CH1	
193	GDMA_AHB_EVT_OUT_FIFO_FULL_CH2	
194	GDMA_AXI_EVT_IN_DONE_CH0	GDMA-AXI
195	GDMA_AXI_EVT_IN_DONE_CH1	
196	GDMA_AXI_EVT_IN_DONE_CH2	
197	GDMA_AXI_EVT_IN_SUC_EOF_CH0	
198	GDMA_AXI_EVT_IN_SUC_EOF_CH1	
199	GDMA_AXI_EVT_IN_SUC_EOF_CH2	
200	GDMA_AXI_EVT_IN_FIFO_EMPTY_CH0	

SOC_ETM_CH*n*_EVT_ID	Selected Event	Peripheral Generating This Event
201	GDMA_AXI_EVT_IN_FIFO_EMPTY_CH1	
202	GDMA_AXI_EVT_IN_FIFO_EMPTY_CH2	
203	GDMA_AXI_EVT_IN_FIFO_FULL_CH0	
204	GDMA_AXI_EVT_IN_FIFO_FULL_CH1	
205	GDMA_AXI_EVT_IN_FIFO_FULL_CH2	
206	GDMA_AXI_EVT_OUT_DONE_CH0	
207	GDMA_AXI_EVT_OUT_DONE_CH1	
208	GDMA_AXI_EVT_OUT_DONE_CH2	
209	GDMA_AXI_EVT_OUT_EOF_CH0	
210	GDMA_AXI_EVT_OUT_EOF_CH1	
211	GDMA_AXI_EVT_OUT_EOF_CH2	
212	GDMA_AXI_EVT_OUT_TOTAL_EOF_CH0	
213	GDMA_AXI_EVT_OUT_TOTAL_EOF_CH1	
214	GDMA_AXI_EVT_OUT_TOTAL_EOF_CH2	
215	GDMA_AXI_EVT_OUT_FIFO_EMPTY_CH0	
216	GDMA_AXI_EVT_OUT_FIFO_EMPTY_CH1	
217	GDMA_AXI_EVT_OUT_FIFO_EMPTY_CH2	
218	GDMA_AXI_EVT_OUT_FIFO_FULL_CH0	
219	GDMA_AXI_EVT_OUT_FIFO_FULL_CH1	
220	GDMA_AXI_EVT_OUT_FIFO_FULL_CH2	
221	PMU_EVT_SLEEP_WEEKUP	PMU
222	DMA2D_EVT_IN_DONE_CH0	2D-DMA Controller (2D-DMA)
223	DMA2D_EVT_IN_DONE_CH1	
224	DMA2D_EVT_IN_SUC_EOF_CH0	
225	DMA2D_EVT_IN_SUC_EOF_CH1	
226	DMA2D_EVT_OUT_DONE_CH0	
227	DMA2D_EVT_OUT_DONE_CH1	
228	DMA2D_EVT_OUT_DONE_CH2	
229	DMA2D_EVT_OUT_EOF_CH0	
230	DMA2D_EVT_OUT_EOF_CH1	
231	DMA2D_EVT_OUT_EOF_CH2	
232	DMA2D_EVT_OUT_TOTAL_EOF_CH0	
233	DMA2D_EVT_OUT_TOTAL_EOF_CH1	
234	DMA2D_EVT_OUT_TOTAL_EOF_CH2	

Whenever any of these events occurs, the corresponding peripheral generates a pulse signal. As soon as the pulse signal is high, the event is considered as being received.

For more detailed descriptions of an event, please refer to the chapter for the peripheral generating this event.

14.3.3 Tasks

An ETM channel can be set up to map its event to one of the tasks by configuring the SOC_ETM_CH*n*_TASK_ID field. Table14.3-2 shows the configuration values of SOC_ETM_CH*n*_TASK_ID and

their corresponding tasks.

Table 14.3-2. Mappable Tasks for ETM Channel*n*

SOC_ETM_CH*n*_TASK_ID	Mapped Task	Peripheral Receiving This Task
1	GPIO_TASK_CH0_SET	GPIO
2	GPIO_TASK_CH1_SET	
3	GPIO_TASK_CH2_SET	
4	GPIO_TASK_CH3_SET	
5	GPIO_TASK_CH4_SET	
6	GPIO_TASK_CH5_SET	
7	GPIO_TASK_CH6_SET	
8	GPIO_TASK_CH7_SET	
9	GPIO_TASK_CH0_CLEAR	
10	GPIO_TASK_CH1_CLEAR	
11	GPIO_TASK_CH2_CLEAR	
12	GPIO_TASK_CH3_CLEAR	
13	GPIO_TASK_CH4_CLEAR	
14	GPIO_TASK_CH5_CLEAR	
15	GPIO_TASK_CH6_CLEAR	
16	GPIO_TASK_CH7_CLEAR	
17	GPIO_TASK_CH0_TOGGLE	
18	GPIO_TASK_CH1_TOGGLE	
19	GPIO_TASK_CH2_TOGGLE	
20	GPIO_TASK_CH3_TOGGLE	
21	GPIO_TASK_CH4_TOGGLE	
22	GPIO_TASK_CH5_TOGGLE	
23	GPIO_TASK_CH6_TOGGLE	
24	GPIO_TASK_CH7_TOGGLE	
25	LEDC_TASK_TIMER0_RES_UPDATE	LED PWM Controller (LEDC)
26	LEDC_TASK_TIMER1_RES_UPDATE	
27	LEDC_TASK_TIMER2_RES_UPDATE	
28	LEDC_TASK_TIMER3_RES_UPDATE	
29	LEDC_TASK_DUTY_SCALE_UPDATE_CH0	
30	LEDC_TASK_DUTY_SCALE_UPDATE_CH1	
31	LEDC_TASK_DUTY_SCALE_UPDATE_CH2	
32	LEDC_TASK_DUTY_SCALE_UPDATE_CH3	
33	LEDC_TASK_DUTY_SCALE_UPDATE_CH4	
34	LEDC_TASK_DUTY_SCALE_UPDATE_CH5	
35	LEDC_TASK_DUTY_SCALE_UPDATE_CH6	
36	LEDC_TASK_DUTY_SCALE_UPDATE_CH7	
37	LEDC_TASK_TIMER0_CAP	
38	LEDC_TASK_TIMER1_CAP	
39	LEDC_TASK_TIMER2_CAP	
40	LEDC_TASK_TIMER3_CAP	
41	LEDC_TASK_SIG_OUT_DIS_CH0	

SOC_ETM_CH*n*_TASK_ID	Mapped Task	Peripheral Receiving This Task
42	LEDC_TASK_SIG_OUT_DIS_CH1	
43	LEDC_TASK_SIG_OUT_DIS_CH2	
44	LEDC_TASK_SIG_OUT_DIS_CH3	
45	LEDC_TASK_SIG_OUT_DIS_CH4	
46	LEDC_TASK_SIG_OUT_DIS_CH5	
47	LEDC_TASK_SIG_OUT_DIS_CH6	
48	LEDC_TASK_SIG_OUT_DIS_CH7	
49	LEDC_TASK_OVF_CNT_RST_CH0	
50	LEDC_TASK_OVF_CNT_RST_CH1	
51	LEDC_TASK_OVF_CNT_RST_CH2	
52	LEDC_TASK_OVF_CNT_RST_CH3	
53	LEDC_TASK_OVF_CNT_RST_CH4	
54	LEDC_TASK_OVF_CNT_RST_CH5	
55	LEDC_TASK_OVF_CNT_RST_CH6	
56	LEDC_TASK_OVF_CNT_RST_CH7	
57	LEDC_TASK_TIMER0_RST	
58	LEDC_TASK_TIMER1_RST	
59	LEDC_TASK_TIMER2_RST	
60	LEDC_TASK_TIMER3_RST	
61	LEDC_TASK_TIMER0_RESUME	
62	LEDC_TASK_TIMER1_RESUME	
63	LEDC_TASK_TIMER2_RESUME	
64	LEDC_TASK_TIMER3_RESUME	
65	LEDC_TASK_TIMER0_PAUSE	
66	LEDC_TASK_TIMER1_PAUSE	
67	LEDC_TASK_TIMER2_PAUSE	
68	LEDC_TASK_TIMER3_PAUSE	
69	LEDC_TASK_GAMMA_RESTART_CH0	
70	LEDC_TASK_GAMMA_RESTART_CH1	
71	LEDC_TASK_GAMMA_RESTART_CH2	
72	LEDC_TASK_GAMMA_RESTART_CH3	
73	LEDC_TASK_GAMMA_RESTART_CH4	
74	LEDC_TASK_GAMMA_RESTART_CH5	
75	LEDC_TASK_GAMMA_RESTART_CH6	
76	LEDC_TASK_GAMMA_RESTART_CH7	
77	LEDC_TASK_GAMMA_PAUSE_CH0	
78	LEDC_TASK_GAMMA_PAUSE_CH1	
79	LEDC_TASK_GAMMA_PAUSE_CH2	
80	LEDC_TASK_GAMMA_PAUSE_CH3	
81	LEDC_TASK_GAMMA_PAUSE_CH4	
82	LEDC_TASK_GAMMA_PAUSE_CH5	
83	LEDC_TASK_GAMMA_PAUSE_CH6	
84	LEDC_TASK_GAMMA_PAUSE_CH7	
85	LEDC_TASK_GAMMA_RESUME_CH0	
86	LEDC_TASK_GAMMA_RESUME_CH1	

SOC_ETM_CH*n*_TASK_ID	Mapped Task	Peripheral Receiving This Task
87	LEDC_TASK_GAMMA_RESUME_CH2	
88	LEDC_TASK_GAMMA_RESUME_CH3	
89	LEDC_TASK_GAMMA_RESUME_CH4	
90	LEDC_TASK_GAMMA_RESUME_CH5	
91	LEDC_TASK_GAMMA_RESUME_CH6	
92	LEDC_TASK_GAMMA_RESUME_CH7	
93	TG0_TASK_CNT_START_TIMER0	General-purpose timer group 0
94	TG0_TASK_ALARM_START_TIMER0	
95	TG0_TASK_CNT_STOP_TIMER0	
96	TG0_TASK_CNT_RELOAD_TIMER0	
97	TG0_TASK_CNT_CAP_TIMER0	
103	TG1_TASK_CNT_START_TIMER0	General-purpose timer group 1
104	TG1_TASK_ALARM_START_TIMER0	
105	TG1_TASK_CNT_STOP_TIMER0	
106	TG1_TASK_CNT_RELOAD_TIMER0	
107	TG1_TASK_CNT_CAP_TIMER0	
113	MCPWM0_TASK_CMPR0_A_UP	MCPWM0
114	MCPWM0_TASK_CMPR1_A_UP	
115	MCPWM0_TASK_CMPR2_A_UP	
116	MCPWM0_TASK_CMPR0_B_UP	
117	MCPWM0_TASK_CMPR1_B_UP	
118	MCPWM0_TASK_CMPR2_B_UP	
119	MCPWM0_TASK_GEN_STOP	
120	MCPWM0_TASK_TIMER0_SYN	
121	MCPWM0_TASK_TIMER1_SYN	
122	MCPWM0_TASK_TIMER2_SYN	
123	MCPWM0_TASK_TIMER0_PERIOD_UP	
124	MCPWM0_TASK_TIMER1_PERIOD_UP	
125	MCPWM0_TASK_TIMER2_PERIOD_UP	
126	MCPWM0_TASK_TZ0_OST	
127	MCPWM0_TASK_TZ1_OST	
128	MCPWM0_TASK_TZ2_OST	
129	MCPWM0_TASK_CLR0_OST	
130	MCPWM0_TASK_CLR1_OST	
131	MCPWM0_TASK_CLR2_OST	
132	MCPWM0_TASK_CAP0	
133	MCPWM0_TASK_CAP1	
134	MCPWM0_TASK_CAP2	
135	MCPWM1_TASK_CMPR0_A_UP	MCPWM1
136	MCPWM1_TASK_CMPR1_A_UP	
137	MCPWM1_TASK_CMPR2_A_UP	
138	MCPWM1_TASK_CMPR0_B_UP	
139	MCPWM1_TASK_CMPR1_B_UP	
140	MCPWM1_TASK_CMPR2_B_UP	
141	MCPWM1_TASK_GEN_STOP	

SOC_ETM_CH*n*_TASK_ID	Mapped Task	Peripheral Receiving This Task
142	MCPWM1_TASK_TIMER0_SYN	
143	MCPWM1_TASK_TIMER1_SYN	
144	MCPWM1_TASK_TIMER2_SYN	
145	MCPWM1_TASK_TIMER0_PERIOD_UP	
146	MCPWM1_TASK_TIMER1_PERIOD_UP	
147	MCPWM1_TASK_TIMER2_PERIOD_UP	
148	MCPWM1_TASK_TZ0_OST	
149	MCPWM1_TASK_TZ1_OST	
150	MCPWM1_TASK_TZ2_OST	
151	MCPWM1_TASK_CLR0_OST	
152	MCPWM1_TASK_CLR1_OST	
153	MCPWM1_TASK_CLR2_OST	
154	MCPWM1_TASK_CAP0	
155	MCPWM1_TASK_CAP1	
156	MCPWM1_TASK_CAP2	
157	ADC_TASK_SAMPLE0	ADC Controller (ADC)
159	ADC_TASK_START0	
160	ADC_TASK_STOP0	
165	TMPSNSR_TASK_START_SAMPLE	Temperature Sensor (TSENS)
166	TMPSNSR_TASK_STOP_SAMPLE	
167	I2S0_TASK_START_RX	I2S0
168	I2S0_TASK_START_TX	
169	I2S0_TASK_STOP_RX	
170	I2S0_TASK_STOP_TX	
171	I2S1_TASK_START_RX	I2S1
172	I2S1_TASK_START_TX	
173	I2S1_TASK_STOP_RX	
174	I2S1_TASK_STOP_TX	
175	I2S2_TASK_START_RX	I2S2
176	I2S2_TASK_START_TX	
177	I2S2_TASK_STOP_RX	
178	I2S2_TASK_STOP_TX	
179	ULP_TASK_WEAKUP_CPU	Low-Power CPU
180	ULP_TASK_INT_CPU	
185	GDMA_AHB_TASK_IN_START_CH0	GDMA-AHB
186	GDMA_AHB_TASK_IN_START_CH1	
187	GDMA_AHB_TASK_IN_START_CH2	
188	GDMA_AHB_TASK_OUT_START_CH0	
189	GDMA_AHB_TASK_OUT_START_CH1	
190	GDMA_AHB_TASK_OUT_START_CH2	
191	GDMA_AXI_TASK_IN_START_CH0	GDMA-AXI
192	GDMA_AXI_TASK_IN_START_CH1	
193	GDMA_AXI_TASK_IN_START_CH2	
194	GDMA_AXI_TASK_OUT_START_CH0	
195	GDMA_AXI_TASK_OUT_START_CH1	

SOC_ETM_CH*n*_TASK_ID	Mapped Task	Peripheral Receiving This Task
196	GDMA_AXI_TASK_OUT_START_CH2	
197	PMU_TASK_SLEEP_REQ	PMU
198	DMA2D_TASK_IN_START_CH0	2D-DMA Controller (2D-DMA)
199	DMA2D_TASK_IN_START_CH1	
200	DMA2D_TASK_IN_DSCR_READY_CH0	
201	DMA2D_TASK_IN_DSCR_READY_CH1	
202	DMA2D_TASK_OUT_START_CH0	
203	DMA2D_TASK_OUT_START_CH1	
204	DMA2D_TASK_OUT_START_CH2	
205	DMA2D_TASK_OUT_DSCR_READY_CH0	
206	DMA2D_TASK_OUT_DSCR_READY_CH1	
207	DMA2D_TASK_OUT_DSCR_READY_CH2	

When a channel receives a valid event pulse signal, it generates the mapped task pulse signal.

For more detailed descriptions of a task, please refer to the chapter for the peripheral receiving this task.

Events from different channels can be optionally mapped to the same task. For example, field SOC_ETM_CH*n*_TASK_ID of multiple channels can be configured with the same value, and field SOC_ETM_CH*n*_EVT_ID can be configured with the same or different values. In this case, when the event received by any of the channels is valid, the task is generated. If events received by multiple channels are valid at the same time, the task will be generated only once.

14.3.4 Event and Task Status

The ETM module supports checking the status of events and tasks by reading registers as follows:

- Event status can be checked by reading the SOC_ETM_EVT_ST*n*_REG register. If the field corresponding to an event is 1, it indicates that the event has been received by ETM. Otherwise, it indicates that the event has not been received by ETM. The fields in the SOC_ETM_EVT_ST*n*_REG register can be cleared by writing 1 to the corresponding fields in the SOC_ETM_EVT_ST*n*_CLR_REG register.
- Task status can be checked by reading the SOC_ETM_TASK_ST*n*_REG register. If the field corresponding to a task is 1, it indicates that the task has been generated by ETM. Otherwise, it indicates that the task has not been generated by ETM. The fields in the SOC_ETM_TASK_ST*n*_REG register can be cleared by writing 1 to the corresponding fields in the SOC_ETM_TASK_ST*n*_CLR_REG register.

14.3.5 Timing Considerations

Figure 14.3-2 shows the structure of clocks that drive received events, sent tasks, and ETM channels.

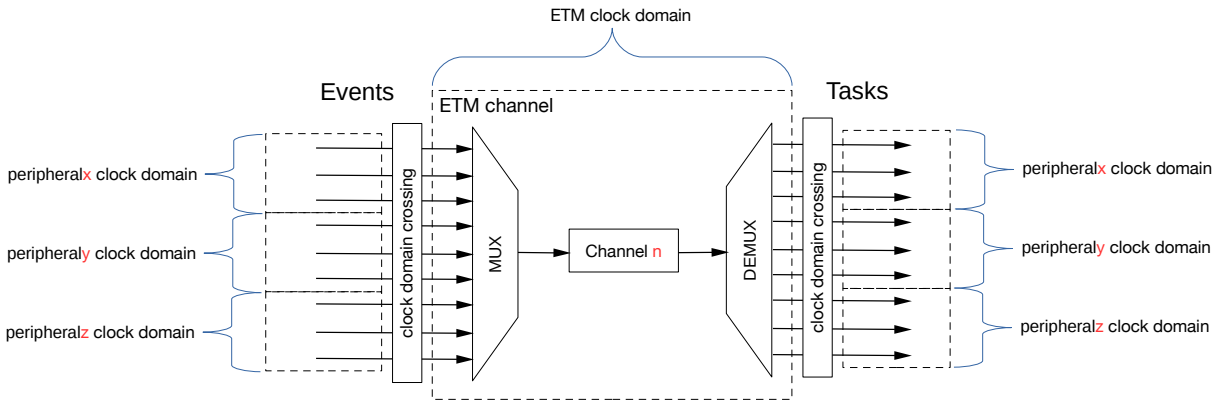


Figure 14.3-2. Event Task Matrix Clock Architecture

ETM is running at the SYS_CLK domain (see Chapter 11 *Reset and Clock*). Each event corresponds to a pulse signal generated by the corresponding peripheral in its clock domain, while each task is mapped by the ETM to a pulse signal under its corresponding peripheral clock domain. The peripherals generating events, the Event Task Matrix, and peripherals receiving tasks are not necessarily running off the same clock and as such need to be synchronized. Therefore, to avoid event loss, there should be a delay between two consecutive event pulses which is dependent on the clock of the peripheral generating the event, on the ETM's clock, and on the clock of the peripheral executing the task.

To make sure the Event Task Matrix receives every event successfully, for peripherals generating event pulses, the interval between two consecutive pulses must be greater than one ETM clock cycle, namely $ceil(\frac{peripheral_clock_frequency}{ETM_clock_frequency})$ in the unit of peripheral clock cycles.

For example, assuming that peripheral A is in the 80 MHz clock domain (REF_80M_CLK), and the ETM runs in the 40 MHz clock domain (SYS_CLK). Peripheral A generates event 1, which should be received by ETM. To receive each event 1 successfully, the interval between two consecutive event 1 must be greater than two peripheral A clock cycles (i.e., one ETM clock cycle).

Likewise, to make sure the Event Task Matrix maps the received event (i.e., event synchronized to the ETM's clock domain) successfully to a task, the interval between two consecutive event pulses in the ETM clock domain must be greater than one peripheral clock cycle, namely $ceil(\frac{ETM_clock_frequency}{peripheral_clock_frequency})$ in the unit of ETM clock cycles.

For example, assuming that peripheral B is in the 20 MHz clock domain (FOSC_20M_CLK), and the ETM runs in the 40 MHz clock domain (SYS_CLK). ETM maps an event to task 1, which should be received by peripheral B. To map each received event successfully to task 1, the interval between two consecutive events must be greater than two ETM clock cycles (i.e., one peripheral B clock cycle).

As a result, to map two consecutive events generated by peripheral A to peripheral B, the interval between these two events must be $ceil(\frac{peripheral_A_clock_frequency}{ETM_clock_frequency}) * ceil(\frac{ETM_clock_frequency}{peripheral_B_clock_frequency})$ in the unit of peripheral A clock cycles.

For example, assuming that peripheral A is in the 80 MHz clock domain (REF_80M_CLK), peripheral B is in the 20 MHz clock domain (FOSC_20M_CLK), and the ETM runs in the 40 MHz clock domain (SYS_CLK). ETM maps event 1 generated by peripheral A to task 1 received by peripheral B. To successfully map each event 1 to task 1, the interval between two consecutive event 1 must be greater than $2 * 2 = 4$ peripheral A clock

cycles.

14.3.6 Channel Control

Each ETM channel can be independently configured to be enabled or disabled. When channel*n* is enabled and receives the event configured via SOC_ETM_CH*n*_EVT_ID, it maps the event to the task configured via SOC_ETM_CH*n*_TASK_ID. When channel*n* is disabled, even if it receives the event configured via SOC_ETM_CH*n*_EVT_ID, no task will be generated.

To enable ETM channel*n*, write 1 to SOC_ETM_CH_ENABLE*n*. To disable ETM channel*n*, Write 1 to SOC_ETM_CH_DISABLE*n*.

The status of ETM channel*n* can be obtained by reading SOC_ETM_CH_ENABLED*n*. 1 indicates that channel*n* has been enabled, and 0 indicates disabled.

If SOC_ETM_CH*n*_EVT_ID or SOC_ETM_CH*n*_TASK_ID is configured to 0, ETM channel*n* will also be disabled.

The complete procedure to configure ETM channel*n* is as follows:

1. Enable the ETM's clock by writing 1 to HP_SYS_CLKRST_ETM_SYS_CLK_EN
2. Select the event to be received by channel*n* via SOC_ETM_CH*n*_EVT_ID
3. Select the task mapped to the received event via SOC_ETM_CH*n*_TASK_ID
4. Enable channel*n* by setting SOC_ETM_CH_ENABLE*n*
5. When channel*n* no longer needs to map the selected event to the selected task, disable channel*n* by setting SOC_ETM_CH_DISABLE*n*. To configure a new event and task mapping, repeat Steps 2 to 4. If no configurations, channel*n* will remain disabled
6. The whole ETM module (i.e., all ETM channels) can be reset by writing 1 and then 0 to the HP_SYS_CLKRST_RST_EN_ETM field

14.4 Register Summary

The addresses in this section are relative to Event Task Matrix base address provided in Table 9.3-2 in Chapter 9 *System and Memory*.

The abbreviations given in Column **Access** are explained in Section *Access Types for Registers*.

Name	Description	Address	Access
Status register			
SOC_ETM_CH_ENA_AD0_REG	Channel status register	0x0000	R/WTC/WTS
SOC_ETM_CH_ENA_AD1_REG	Channel status register	0x000C	R/WTC/WTS
SOC_ETM_EVT_ST0_REG	Event status register	0x01A8	R/WTC/SS
SOC_ETM_EVT_ST1_REG	Event status register	0x01B0	R/WTC/SS
SOC_ETM_EVT_ST2_REG	Event status register	0x01B8	R/WTC/SS
SOC_ETM_EVT_ST3_REG	Event status register	0x01C0	R/WTC/SS
SOC_ETM_EVT_ST4_REG	Event status register	0x01C8	R/WTC/SS
SOC_ETM_EVT_ST5_REG	Event status register	0x01D0	R/WTC/SS
SOC_ETM_EVT_ST6_REG	Event status register	0x01D8	R/WTC/SS
SOC_ETM_EVT_ST7_REG	Event status register	0x01E0	R/WTC/SS
SOC_ETM_TASK_ST0_REG	Task status register	0x01E8	R/WTC/SS
SOC_ETM_TASK_ST1_REG	Task status register	0x01F0	R/WTC/SS
SOC_ETM_TASK_ST2_REG	Task status register	0x01F8	R/WTC/SS
SOC_ETM_TASK_ST3_REG	Task status register	0x0200	R/WTC/SS
SOC_ETM_TASK_ST4_REG	Task status register	0x0208	R/WTC/SS
SOC_ETM_TASK_ST5_REG	Task status register	0x0210	R/WTC/SS
SOC_ETM_TASK_ST6_REG	Task status register	0x0218	R/WTC/SS
Configuration Register			
SOC_ETM_CH_ENA_AD0_SET_REG	Channel enable register	0x0004	WT
SOC_ETM_CH_ENA_AD0_CLR_REG	Channel disable register	0x0008	WT
SOC_ETM_CH_ENA_AD1_SET_REG	Channel enable register	0x0010	WT
SOC_ETM_CH_ENA_AD1_CLR_REG	Channel disable register	0x0014	WT
SOC_ETM_CH0_EVT_ID_REG	Channel0 event ID register	0x0018	R/W
SOC_ETM_CH0_TASK_ID_REG	Channel0 task ID register	0x001C	R/W
SOC_ETM_CH1_EVT_ID_REG	Channel1 event ID register	0x0020	R/W
SOC_ETM_CH1_TASK_ID_REG	Channel1 task ID register	0x0024	R/W
SOC_ETM_CH2_EVT_ID_REG	Channel2 event ID register	0x0028	R/W
SOC_ETM_CH2_TASK_ID_REG	Channel2 task ID register	0x002C	R/W
SOC_ETM_CH3_EVT_ID_REG	Channel3 event ID register	0x0030	R/W
SOC_ETM_CH3_TASK_ID_REG	Channel3 task ID register	0x0034	R/W
SOC_ETM_CH4_EVT_ID_REG	Channel4 event ID register	0x0038	R/W
SOC_ETM_CH4_TASK_ID_REG	Channel4 task ID register	0x003C	R/W
SOC_ETM_CH5_EVT_ID_REG	Channel5 event ID register	0x0040	R/W
SOC_ETM_CH5_TASK_ID_REG	Channel5 task ID register	0x0044	R/W
SOC_ETM_CH6_EVT_ID_REG	Channel6 event ID register	0x0048	R/W
SOC_ETM_CH6_TASK_ID_REG	Channel6 task ID register	0x004C	R/W

Name	Description	Address	Access
SOC_ETM_CH7_EVT_ID_REG	Channel7 event ID register	0x0050	R/W
SOC_ETM_CH7_TASK_ID_REG	Channel7 task ID register	0x0054	R/W
SOC_ETM_CH8_EVT_ID_REG	Channel8 event ID register	0x0058	R/W
SOC_ETM_CH8_TASK_ID_REG	Channel8 task ID register	0x005C	R/W
SOC_ETM_CH9_EVT_ID_REG	Channel9 event ID register	0x0060	R/W
SOC_ETM_CH9_TASK_ID_REG	Channel9 task ID register	0x0064	R/W
SOC_ETM_CH10_EVT_ID_REG	Channel10 event ID register	0x0068	R/W
SOC_ETM_CH10_TASK_ID_REG	Channel10 task ID register	0x006C	R/W
SOC_ETM_CH11_EVT_ID_REG	Channel11 event ID register	0x0070	R/W
SOC_ETM_CH11_TASK_ID_REG	Channel11 task ID register	0x0074	R/W
SOC_ETM_CH12_EVT_ID_REG	Channel12 event ID register	0x0078	R/W
SOC_ETM_CH12_TASK_ID_REG	Channel12 task ID register	0x007C	R/W
SOC_ETM_CH13_EVT_ID_REG	Channel13 event ID register	0x0080	R/W
SOC_ETM_CH13_TASK_ID_REG	Channel13 task ID register	0x0084	R/W
SOC_ETM_CH14_EVT_ID_REG	Channel14 event ID register	0x0088	R/W
SOC_ETM_CH14_TASK_ID_REG	Channel14 task ID register	0x008C	R/W
SOC_ETM_CH15_EVT_ID_REG	Channel15 event ID register	0x0090	R/W
SOC_ETM_CH15_TASK_ID_REG	Channel15 task ID register	0x0094	R/W
SOC_ETM_CH16_EVT_ID_REG	Channel16 event ID register	0x0098	R/W
SOC_ETM_CH16_TASK_ID_REG	Channel16 task ID register	0x009C	R/W
SOC_ETM_CH17_EVT_ID_REG	Channel17 event ID register	0x00A0	R/W
SOC_ETM_CH17_TASK_ID_REG	Channel17 task ID register	0x00A4	R/W
SOC_ETM_CH18_EVT_ID_REG	Channel18 event ID register	0x00A8	R/W
SOC_ETM_CH18_TASK_ID_REG	Channel18 task ID register	0x00AC	R/W
SOC_ETM_CH19_EVT_ID_REG	Channel19 event ID register	0x00B0	R/W
SOC_ETM_CH19_TASK_ID_REG	Channel19 task ID register	0x00B4	R/W
SOC_ETM_CH20_EVT_ID_REG	Channel20 event ID register	0x00B8	R/W
SOC_ETM_CH20_TASK_ID_REG	Channel20 task ID register	0x00BC	R/W
SOC_ETM_CH21_EVT_ID_REG	Channel21 event ID register	0x00C0	R/W
SOC_ETM_CH21_TASK_ID_REG	Channel21 task ID register	0x00C4	R/W
SOC_ETM_CH22_EVT_ID_REG	Channel22 event ID register	0x00C8	R/W
SOC_ETM_CH22_TASK_ID_REG	Channel22 task ID register	0x00CC	R/W
SOC_ETM_CH23_EVT_ID_REG	Channel23 event ID register	0x00D0	R/W
SOC_ETM_CH23_TASK_ID_REG	Channel23 task ID register	0x00D4	R/W
SOC_ETM_CH24_EVT_ID_REG	Channel24 event ID register	0x00D8	R/W
SOC_ETM_CH24_TASK_ID_REG	Channel24 task ID register	0x00DC	R/W
SOC_ETM_CH25_EVT_ID_REG	Channel25 event ID register	0x00E0	R/W
SOC_ETM_CH25_TASK_ID_REG	Channel25 task ID register	0x00E4	R/W
SOC_ETM_CH26_EVT_ID_REG	Channel26 event ID register	0x00E8	R/W
SOC_ETM_CH26_TASK_ID_REG	Channel26 task ID register	0x00EC	R/W
SOC_ETM_CH27_EVT_ID_REG	Channel27 event ID register	0x00F0	R/W
SOC_ETM_CH27_TASK_ID_REG	Channel27 task ID register	0x00F4	R/W
SOC_ETM_CH28_EVT_ID_REG	Channel28 event ID register	0x00F8	R/W

Name	Description	Address	Access
SOC_ETM_CH28_TASK_ID_REG	Channel28 task ID register	0x00FC	R/W
SOC_ETM_CH29_EVT_ID_REG	Channel29 event ID register	0x0100	R/W
SOC_ETM_CH29_TASK_ID_REG	Channel29 task ID register	0x0104	R/W
SOC_ETM_CH30_EVT_ID_REG	Channel30 event ID register	0x0108	R/W
SOC_ETM_CH30_TASK_ID_REG	Channel30 task ID register	0x010C	R/W
SOC_ETM_CH31_EVT_ID_REG	Channel31 event ID register	0x0110	R/W
SOC_ETM_CH31_TASK_ID_REG	Channel31 task ID register	0x0114	R/W
SOC_ETM_CH32_EVT_ID_REG	Channel32 event ID register	0x0118	R/W
SOC_ETM_CH32_TASK_ID_REG	Channel32 task ID register	0x011C	R/W
SOC_ETM_CH33_EVT_ID_REG	Channel33 event ID register	0x0120	R/W
SOC_ETM_CH33_TASK_ID_REG	Channel33 task ID register	0x0124	R/W
SOC_ETM_CH34_EVT_ID_REG	Channel34 event ID register	0x0128	R/W
SOC_ETM_CH34_TASK_ID_REG	Channel34 task ID register	0x012C	R/W
SOC_ETM_CH35_EVT_ID_REG	Channel35 event ID register	0x0130	R/W
SOC_ETM_CH35_TASK_ID_REG	Channel35 task ID register	0x0134	R/W
SOC_ETM_CH36_EVT_ID_REG	Channel36 event ID register	0x0138	R/W
SOC_ETM_CH36_TASK_ID_REG	Channel36 task ID register	0x013C	R/W
SOC_ETM_CH37_EVT_ID_REG	Channel37 event ID register	0x0140	R/W
SOC_ETM_CH37_TASK_ID_REG	Channel37 task ID register	0x0144	R/W
SOC_ETM_CH38_EVT_ID_REG	Channel38 event ID register	0x0148	R/W
SOC_ETM_CH38_TASK_ID_REG	Channel38 task ID register	0x014C	R/W
SOC_ETM_CH39_EVT_ID_REG	Channel39 event ID register	0x0150	R/W
SOC_ETM_CH39_TASK_ID_REG	Channel39 task ID register	0x0154	R/W
SOC_ETM_CH40_EVT_ID_REG	Channel40 event ID register	0x0158	R/W
SOC_ETM_CH40_TASK_ID_REG	Channel40 task ID register	0x015C	R/W
SOC_ETM_CH41_EVT_ID_REG	Channel41 event ID register	0x0160	R/W
SOC_ETM_CH41_TASK_ID_REG	Channel41 task ID register	0x0164	R/W
SOC_ETM_CH42_EVT_ID_REG	Channel42 event ID register	0x0168	R/W
SOC_ETM_CH42_TASK_ID_REG	Channel42 task ID register	0x016C	R/W
SOC_ETM_CH43_EVT_ID_REG	Channel43 event ID register	0x0170	R/W
SOC_ETM_CH43_TASK_ID_REG	Channel43 task ID register	0x0174	R/W
SOC_ETM_CH44_EVT_ID_REG	Channel44 event ID register	0x0178	R/W
SOC_ETM_CH44_TASK_ID_REG	Channel44 task ID register	0x017C	R/W
SOC_ETM_CH45_EVT_ID_REG	Channel45 event ID register	0x0180	R/W
SOC_ETM_CH45_TASK_ID_REG	Channel45 task ID register	0x0184	R/W
SOC_ETM_CH46_EVT_ID_REG	Channel46 event ID register	0x0188	R/W
SOC_ETM_CH46_TASK_ID_REG	Channel46 task ID register	0x018C	R/W
SOC_ETM_CH47_EVT_ID_REG	Channel47 event ID register	0x0190	R/W
SOC_ETM_CH47_TASK_ID_REG	Channel47 task ID register	0x0194	R/W
SOC_ETM_CH48_EVT_ID_REG	Channel48 event ID register	0x0198	R/W
SOC_ETM_CH48_TASK_ID_REG	Channel48 task ID register	0x019C	R/W
SOC_ETM_CH49_EVT_ID_REG	Channel49 event ID register	0x01A0	R/W
SOC_ETM_CH49_TASK_ID_REG	Channel49 task ID register	0x01A4	R/W

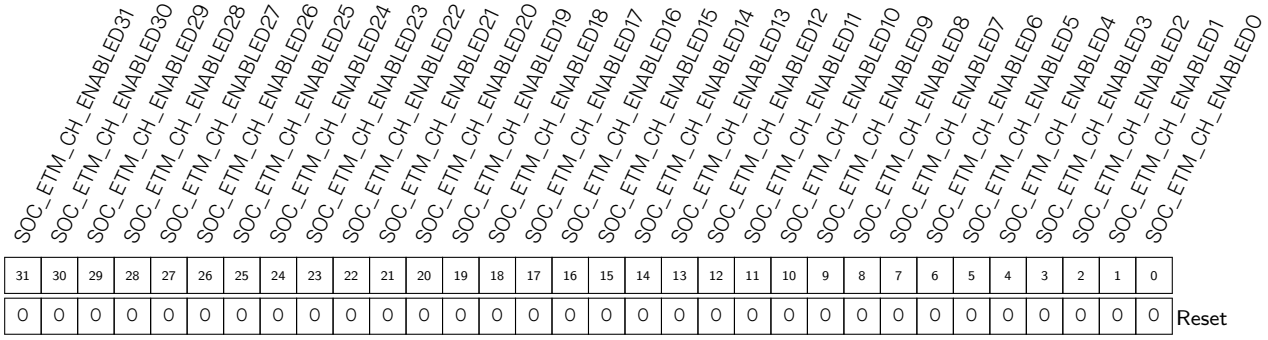
Name	Description	Address	Access
SOC_ETM_EVT_ST0_CLR_REG	Event status clear register	0x01AC	WT
SOC_ETM_EVT_ST1_CLR_REG	Event status clear register	0x01B4	WT
SOC_ETM_EVT_ST2_CLR_REG	Event status clear register	0x01BC	WT
SOC_ETM_EVT_ST3_CLR_REG	Event status clear register	0x01C4	WT
SOC_ETM_EVT_ST4_CLR_REG	Event status clear register	0x01CC	WT
SOC_ETM_EVT_ST5_CLR_REG	Event status clear register	0x01D4	WT
SOC_ETM_EVT_ST6_CLR_REG	Event status clear register	0x01DC	WT
SOC_ETM_EVT_ST7_CLR_REG	Event status clear register	0x01E4	WT
SOC_ETM_TASK_ST0_CLR_REG	Task status clear register	0x01EC	WT
SOC_ETM_TASK_ST1_CLR_REG	Task status clear register	0x01F4	WT
SOC_ETM_TASK_ST2_CLR_REG	Task status clear register	0x01FC	WT
SOC_ETM_TASK_ST3_CLR_REG	Task status clear register	0x0204	WT
SOC_ETM_TASK_ST4_CLR_REG	Task status clear register	0x020C	WT
SOC_ETM_TASK_ST5_CLR_REG	Task status clear register	0x0214	WT
SOC_ETM_TASK_ST6_CLR_REG	Task status clear register	0x021C	WT
SOC_ETM_CLK_EN_REG	ETM clock enable register	0x0220	R/W
Version Register			
SOC_ETM_DATE_REG	Version control register	0x0224	R/W

14.5 Registers

The addresses in this section are relative to Event Task Matrix base address provided in Table 9.3-2 in Chapter 9 *System and Memory*.

For how to program reserved fields, please refer to Section Programming Reserved Register Field.

Register 14.1. SOC_ETM_CH_ENA_AD0_REG (0x0000)



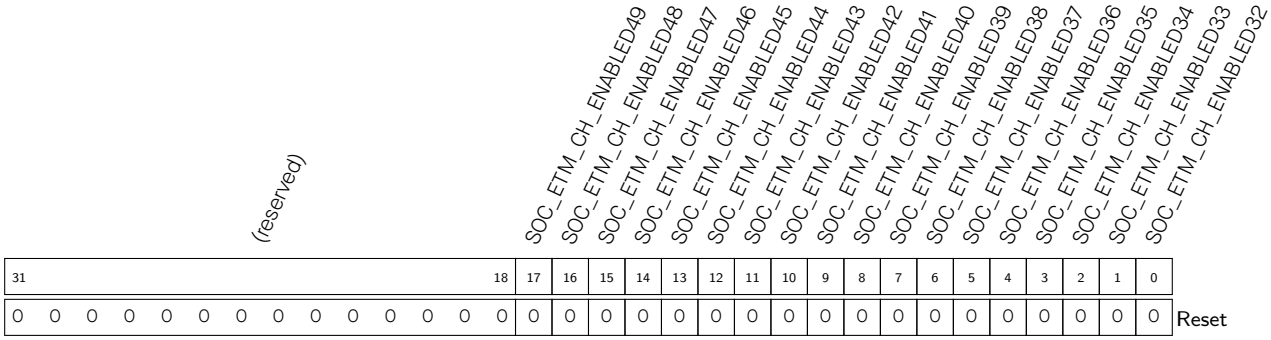
SOC_ETM_CH_ENABLED*n* (*n*: 0-31) Represents the status of channel*n*.

0: Disabled

1: Enabled

(R/WTC/SS)

Register 14.2. SOC_ETM_CH_ENA_AD1_REG (0x000C)



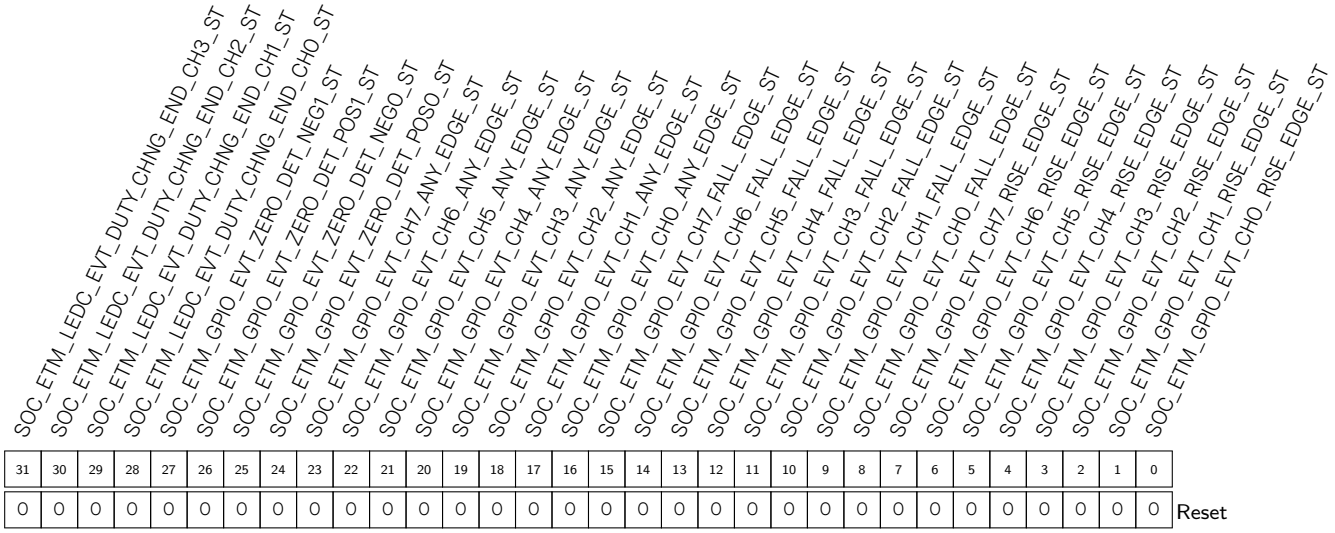
SOC_ETM_CH_ENABLED*n* (*n*: 32-49) Represents the status of channel*n*.

0: Disabled

1: Enabled

(R/WTC/SS)

Register 14.3. SOC_ETM_EVT_ST0_REG (0x01A8)



SOC_ETM_GPIO_EVT_CH0_RISE_EDGE_ST Represents the status of GPIO_EVT_CH0_RISE_EDGE.

0: Not received

1: Received

(R/WTC/SS)

SOC_ETM_GPIO_EVT_CH1_RISE_EDGE_ST Represents the status of GPIO_EVT_CH1_RISE_EDGE.

0: Not received

1: Received

(R/WTC/SS)

SOC_ETM_GPIO_EVT_CH2_RISE_EDGE_ST Represents the status of GPIO_EVT_CH2_RISE_EDGE.

0: Not received

1: Received

(R/WTC/SS)

SOC_ETM_GPIO_EVT_CH3_RISE_EDGE_ST Represents the status of GPIO_EVT_CH3_RISE_EDGE.

0: Not received

1: Received

(R/WTC/SS)

SOC_ETM_GPIO_EVT_CH4_RISE_EDGE_ST Represents the status of GPIO_EVT_CH4_RISE_EDGE.

0: Not received

1: Received

(R/WTC/SS)

SOC_ETM_GPIO_EVT_CH5_RISE_EDGE_ST Represents the status of GPIO_EVT_CH5_RISE_EDGE.

0: Not received

1: Received

(R/WTC/SS)

Continued on the next page...

Register 14.3. SOC_ETM_EVT_ST0_REG (0x01A8)

Continued from the previous page...

SOC_ETM_GPIO_EVT_CH6_RISE_EDGE_ST Represents the status of GPIO_EVT_CH6_RISE_EDGE.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_GPIO_EVT_CH7_RISE_EDGE_ST Represents the status of GPIO_EVT_CH7_RISE_EDGE.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_GPIO_EVT_CH0_FALL_EDGE_ST Represents the status of GPIO_EVT_CH0_FALL_EDGE.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_GPIO_EVT_CH1_FALL_EDGE_ST Represents the status of GPIO_EVT_CH1_FALL_EDGE.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_GPIO_EVT_CH2_FALL_EDGE_ST Represents the status of GPIO_EVT_CH2_FALL_EDGE.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_GPIO_EVT_CH3_FALL_EDGE_ST Represents the status of GPIO_EVT_CH3_FALL_EDGE.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_GPIO_EVT_CH4_FALL_EDGE_ST Represents the status of GPIO_EVT_CH4_FALL_EDGE.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_GPIO_EVT_CH5_FALL_EDGE_ST Represents the status of GPIO_EVT_CH5_FALL_EDGE.
0: Not received
1: Received
(R/WTC/SS)

Continued on the next page...

Register 14.3. SOC_ETM_EVT_ST0_REG (0x01A8)

Continued from the previous page...

SOC_ETM_GPIO_EVT_CH6_FALL_EDGE_ST Represents the status of GPIO_EVT_CH6_FALL_EDGE.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_GPIO_EVT_CH7_FALL_EDGE_ST Represents the status of GPIO_EVT_CH7_FALL_EDGE.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_GPIO_EVT_CH0_ANY_EDGE_ST Represents the status of GPIO_EVT_CH0_ANY_EDGE.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_GPIO_EVT_CH1_ANY_EDGE_ST Represents the status of GPIO_EVT_CH1_ANY_EDGE.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_GPIO_EVT_CH2_ANY_EDGE_ST Represents the status of GPIO_EVT_CH2_ANY_EDGE.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_GPIO_EVT_CH3_ANY_EDGE_ST Represents the status of GPIO_EVT_CH3_ANY_EDGE.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_GPIO_EVT_CH4_ANY_EDGE_ST Represents the status of GPIO_EVT_CH4_ANY_EDGE.
0: Not received
1: Received
(R/WTC/SS)

Continued on the next page...

Register 14.3. SOC_ETM_EVT_ST0_REG (0x01A8)

Continued from the previous page...

SOC_ETM_GPIO_EVT_CH5_ANY_EDGE_ST Represents the status of GPIO_EVT_CH5_ANY_EDGE.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_GPIO_EVT_CH6_ANY_EDGE_ST Represents the status of GPIO_EVT_CH6_ANY_EDGE.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_GPIO_EVT_CH7_ANY_EDGE_ST Represents the status of GPIO_EVT_CH7_ANY_EDGE.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_GPIO_EVT_ZERO_DET_POS0_ST Represents the status of GPIO_EVT_ZERO_DET_POS0.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_GPIO_EVT_ZERO_DET_NEG0_ST Represents the status of GPIO_EVT_ZERO_DET_NEG0.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_GPIO_EVT_ZERO_DET_POS1_ST Represents the status of GPIO_EVT_ZERO_DET_POS1.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_GPIO_EVT_ZERO_DET_NEG1_ST Represents the status of GPIO_EVT_ZERO_DET_NEG1.
0: Not received
1: Received
(R/WTC/SS)

Continued on the next page...

Register 14.3. SOC_ETM_EVT_ST0_REG (0x01A8)

Continued from the previous page...

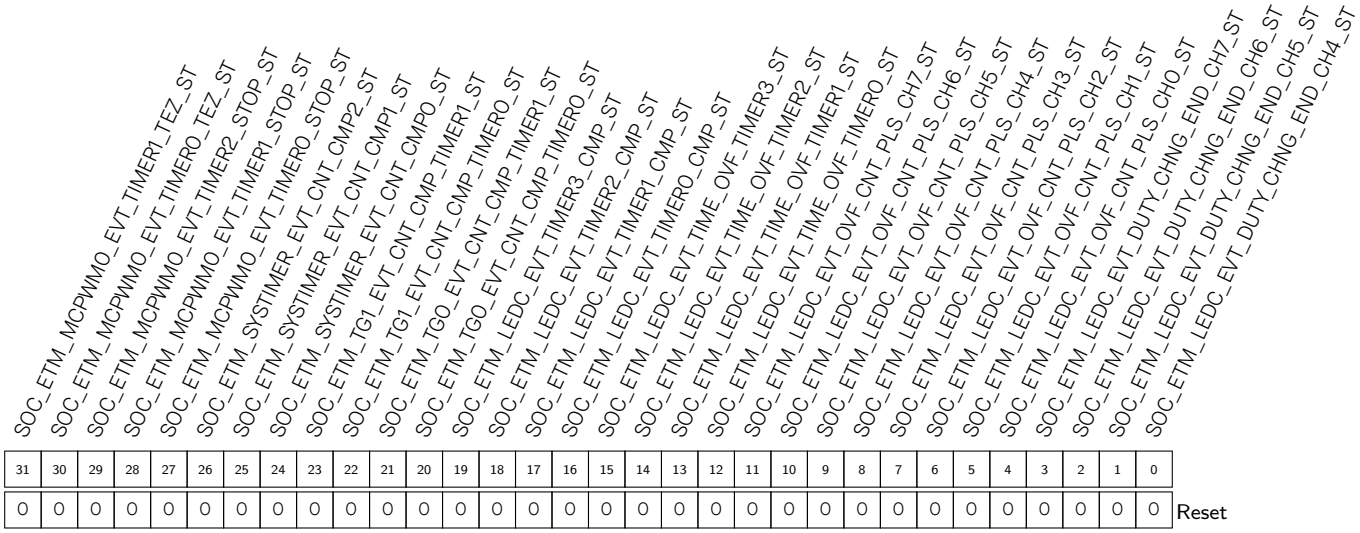
SOC_ETM_LEDC_EVT_DUTY_CHNG_END_CH0_ST Represents the status of LEDC_EVT_DUTY_CHNG_END_CH0.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_LEDC_EVT_DUTY_CHNG_END_CH1_ST Represents the status of LEDC_EVT_DUTY_CHNG_END_CH1.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_LEDC_EVT_DUTY_CHNG_END_CH2_ST Represents the status of LEDC_EVT_DUTY_CHNG_END_CH2.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_LEDC_EVT_DUTY_CHNG_END_CH3_ST Represents the status of LEDC_EVT_DUTY_CHNG_END_CH3.
0: Not received
1: Received
(R/WTC/SS)

Register 14.4. SOC_ETM_EVT_ST1_REG (0x01B0)



SOC_ETM_LEDC_EVT_DUTY_CHNG_END_CH4_ST Represents the status of LEDC_EVT_DUTY_CHNG_END_CH4.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_LEDC_EVT_DUTY_CHNG_END_CH5_ST Represents the status of LEDC_EVT_DUTY_CHNG_END_CH5.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_LEDC_EVT_DUTY_CHNG_END_CH6_ST Represents the status of LEDC_EVT_DUTY_CHNG_END_CH6.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_LEDC_EVT_DUTY_CHNG_END_CH7_ST Represents the status of LEDC_EVT_DUTY_CHNG_END_CH7.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_LEDC_EVT_OVF_CNT_PLS_CH0_ST Represents the status of LEDC_EVT_OVF_CNT_PLS_CH0.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_LEDC_EVT_OVF_CNT_PLS_CH1_ST Represents the status of LEDC_EVT_OVF_CNT_PLS_CH1.
0: Not received
1: Received
(R/WTC/SS)

Continued on the next page...

Register 14.4. SOC_ETM_EVT_ST1_REG (0x01B0)

Continued from the previous page...

SOC_ETM_LEDC_EVT_OVF_CNT_PLS_CH2_ST Represents the status of LEDC_EVT_OVF_CNT_PLS_CH2.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_LEDC_EVT_OVF_CNT_PLS_CH3_ST Represents the status of LEDC_EVT_OVF_CNT_PLS_CH3.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_LEDC_EVT_OVF_CNT_PLS_CH4_ST Represents the status of LEDC_EVT_OVF_CNT_PLS_CH4.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_LEDC_EVT_OVF_CNT_PLS_CH5_ST Represents the status of LEDC_EVT_OVF_CNT_PLS_CH5.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_LEDC_EVT_OVF_CNT_PLS_CH6_ST Represents the status of LEDC_EVT_OVF_CNT_PLS_CH6.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_LEDC_EVT_OVF_CNT_PLS_CH7_ST Represents the status of LEDC_EVT_OVF_CNT_PLS_CH7.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_LEDC_EVT_TIME_OVF_TIMER0_ST Represents the status of LEDC_EVT_TIME_OVF_TIMER0.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_LEDC_EVT_TIME_OVF_TIMER1_ST Represents the status of LEDC_EVT_TIME_OVF_TIMER1.
0: Not received
1: Received
(R/WTC/SS)

Continued on the next page...

Register 14.4. SOC_ETM_EVT_ST1_REG (0x01B0)

Continued from the previous page...

SOC_ETM_LEDC_EVT_TIME_OVF_TIMER2_ST Represents the status of LEDC_EVT_TIME_OVF_TIMER2.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_LEDC_EVT_TIME_OVF_TIMER3_ST Represents the status of LEDC_EVT_TIME_OVF_TIMER3.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_LEDC_EVT_TIMER0_CMP_ST Represents the status of LEDC_EVT_TIMER0_CMP.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_LEDC_EVT_TIMER1_CMP_ST Represents the status of LEDC_EVT_TIMER1_CMP.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_LEDC_EVT_TIMER2_CMP_ST Represents the status of LEDC_EVT_TIMER2_CMP.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_LEDC_EVT_TIMER3_CMP_ST Represents the status of LEDC_EVT_TIMER3_CMP.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_TG0_EVT_CNT_CMP_TIMER0_ST Represents the status of TG0_EVT_CNT_CMP_TIMER0.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_TG0_EVT_CNT_CMP_TIMER1_ST Represents the status of TG0_EVT_CNT_CMP_TIMER1.
0: Not received
1: Received
(R/WTC/SS)

Continued on the next page...

Register 14.4. SOC_ETM_EVT_ST1_REG (0x01B0)

Continued from the previous page...

SOC_ETM_TG1_EVT_CNT_CMP_TIMER0_ST Represents the status of TG1_EVT_CNT_CMP_TIMER0.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_TG1_EVT_CNT_CMP_TIMER1_ST Represents the status of TG1_EVT_CNT_CMP_TIMER1.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_SYSTIMER_EVT_CNT_CMP0_ST Represents the status of SYSTIMER_EVT_CNT_CMP0.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_SYSTIMER_EVT_CNT_CMP1_ST Represents the status of SYSTIMER_EVT_CNT_CMP1.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_SYSTIMER_EVT_CNT_CMP2_ST Represents the status of SYSTIMER_EVT_CNT_CMP2.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_MCPWM0_EVT_TIMER0_STOP_ST Represents the status of MCPWM0_EVT_TIMER0_STOP.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_MCPWM0_EVT_TIMER1_STOP_ST Represents the status of MCPWM0_EVT_TIMER1_STOP.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_MCPWM0_EVT_TIMER2_STOP_ST Represents the status of MCPWM0_EVT_TIMER2_STOP.
0: Not received
1: Received
(R/WTC/SS)

Continued on the next page...

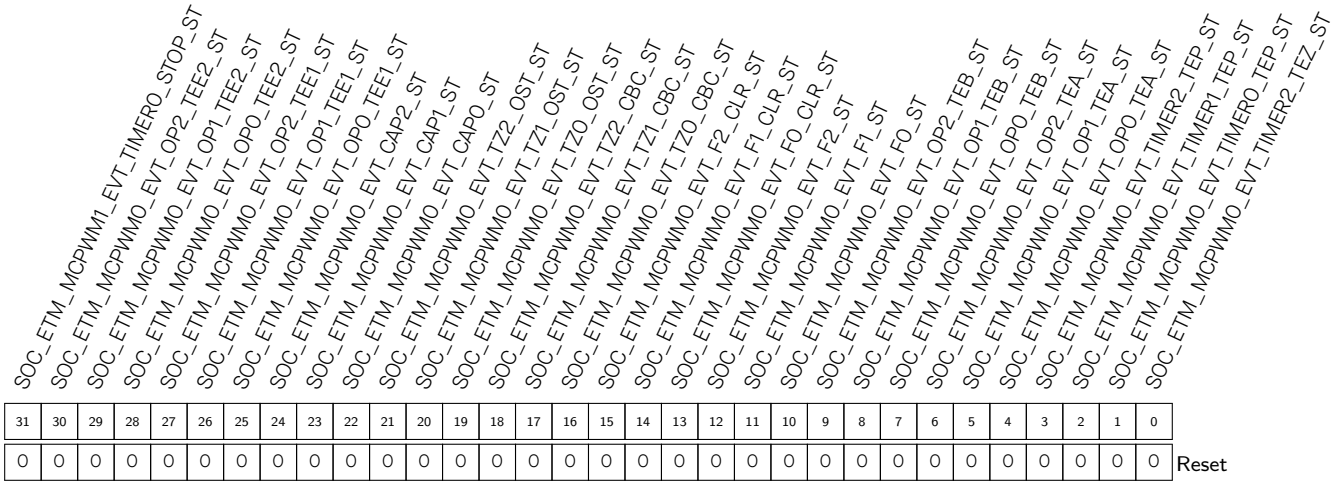
Register 14.4. SOC_ETM_EVT_ST1_REG (0x01B0)

Continued from the previous page...

SOC_ETM_MCPWM0_EVT_TIMER0_TEZ_ST Represents the status of MCPWM0_EVT_TIMER0_TEZ.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_MCPWM0_EVT_TIMER1_TEZ_ST Represents the status of MCPWM0_EVT_TIMER1_TEZ.
0: Not received
1: Received
(R/WTC/SS)

Register 14.5. SOC_ETM_EVT_ST2_REG (0x01B8)



31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Reset

SOC_ETM_MCPWM0_EVT_TIMER2_TEZ_ST Represents the status of MCPWM0_EVT_TIMER2_TEZ.

0: Not received

1: Received

(R/WTC/SS)

SOC_ETM_MCPWM0_EVT_TIMER0_TEP_ST Represents the status of MCPWM0_EVT_TIMER0_TEP.

0: Not received

1: Received

(R/WTC/SS)

SOC_ETM_MCPWM0_EVT_TIMER1_TEP_ST Represents the status of MCPWM0_EVT_TIMER1_TEP.

0: Not received

1: Received

(R/WTC/SS)

SOC_ETM_MCPWM0_EVT_TIMER2_TEP_ST Represents the status of MCPWM0_EVT_TIMER2_TEP.

0: Not received

1: Received

(R/WTC/SS)

SOC_ETM_MCPWM0_EVT_OP0_TEA_ST Represents the status of MCPWM0_EVT_OP0_TEA.

0: Not received

1: Received

(R/WTC/SS)

SOC_ETM_MCPWM0_EVT_OP1_TEA_ST Represents the status of MCPWM0_EVT_OP1_TEA.

0: Not received

1: Received

(R/WTC/SS)

Continued on the next page...

Register 14.5. SOC_ETM_EVT_ST2_REG (0x01B8)

Continued from the previous page...

SOC_ETM_MCPWM0_EVT_OP2_TEA_ST Represents the status of MCPWM0_EVT_OP2_TEA.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_MCPWM0_EVT_OP0_TEB_ST Represents the status of MCPWM0_EVT_OP0_TEB.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_MCPWM0_EVT_OP1_TEB_ST Represents the status of MCPWM0_EVT_OP1_TEB.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_MCPWM0_EVT_OP2_TEB_ST Represents the status of MCPWM0_EVT_OP2_TEB.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_MCPWM0_EVT_F0_ST Represents the status of MCPWM0_EVT_F0.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_MCPWM0_EVT_F1_ST Represents the status of MCPWM0_EVT_F1.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_MCPWM0_EVT_F2_ST Represents the status of MCPWM0_EVT_F2.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_MCPWM0_EVT_F0_CLR_ST Represents the status of MCPWM0_EVT_F0_CLR.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_MCPWM0_EVT_F1_CLR_ST Represents the status of MCPWM0_EVT_F1_CLR.
0: Not received
1: Received
(R/WTC/SS)

Continued on the next page...

Register 14.5. SOC_ETM_EVT_ST2_REG (0x01B8)

Continued from the previous page...

SOC_ETM_MCPWM0_EVT_F2_CLR_ST Represents the status of MCPWM0_EVT_F2_CLR.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_MCPWM0_EVT_TZ0_CBC_ST Represents the status of MCPWM0_EVT_TZ0_CBC.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_MCPWM0_EVT_TZ1_CBC_ST Represents the status of MCPWM0_EVT_TZ1_CBC.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_MCPWM0_EVT_TZ2_CBC_ST Represents the status of MCPWM0_EVT_TZ2_CBC.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_MCPWM0_EVT_TZ0_OST_ST Represents the status of MCPWM0_EVT_TZ0_OST.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_MCPWM0_EVT_TZ1_OST_ST Represents the status of MCPWM0_EVT_TZ1_OST.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_MCPWM0_EVT_TZ2_OST_ST Represents the status of MCPWM0_EVT_TZ2_OST.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_MCPWM0_EVT_CAP0_ST Represents the status of MCPWM0_EVT_CAP0.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_MCPWM0_EVT_CAP1_ST Represents the status of MCPWM0_EVT_CAP1.
0: Not received
1: Received
(R/WTC/SS)

Continued on the next page...

Register 14.5. SOC_ETM_EVT_ST2_REG (0x01B8)

Continued from the previous page...

SOC_ETM_MCPWM0_EVT_CAP2_ST Represents the status of MCPWM0_EVT_CAP2.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_MCPWM0_EVT_OP0_TEE1_ST Represents the status of MCPWM0_EVT_OP0_TEE1.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_MCPWM0_EVT_OP1_TEE1_ST Represents the status of MCPWM0_EVT_OP1_TEE1.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_MCPWM0_EVT_OP2_TEE1_ST Represents the status of MCPWM0_EVT_OP2_TEE1.
0: Not received
1: Received
(R/WTC/SS)

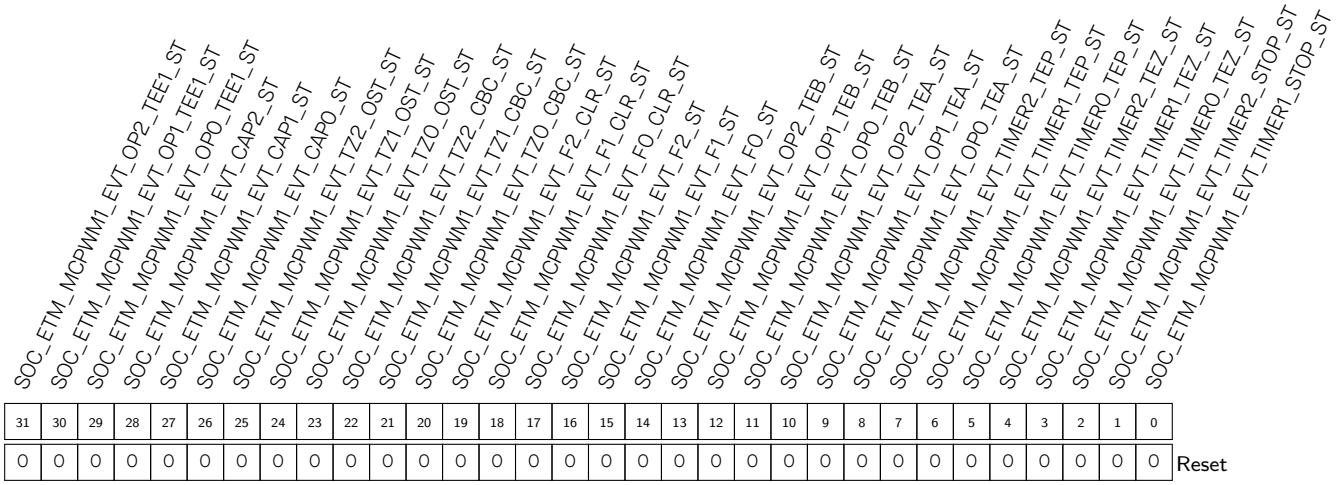
SOC_ETM_MCPWM0_EVT_OP0_TEE2_ST Represents the status of MCPWM0_EVT_OP0_TEE2.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_MCPWM0_EVT_OP1_TEE2_ST Represents the status of MCPWM0_EVT_OP1_TEE2.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_MCPWM0_EVT_OP2_TEE2_ST Represents the status of MCPWM0_EVT_OP2_TEE2.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_MCPWM1_EVT_TIMER0_STOP_ST Represents the status of MCPWM1_EVT_TIMER0_STOP.
0: Not received
1: Received
(R/WTC/SS)

Register 14.6. SOC_ETM_EVT_ST3_REG (0x01C0)



SOC_ETM_MCPWM1_EVT_TIMER1_STOP_ST Represents the status of MCPWM1_EVT_TIMER1_STOP.

0: Not received

1: Received

(R/WTC/SS)

SOC_ETM_MCPWM1_EVT_TIMER2_STOP_ST Represents the status of MCPWM1_EVT_TIMER2_STOP.

0: Not received

1: Received

(R/WTC/SS)

SOC_ETM_MCPWM1_EVT_TIMERO_TEZ_ST Represents the status of MCPWM1_EVT_TIMERO_TEZ.

0: Not received

1: Received

(R/WTC/SS)

SOC_ETM_MCPWM1_EVT_TIMER1_TEZ_ST Represents the status of MCPWM1_EVT_TIMER1_TEZ.

0: Not received

1: Received

(R/WTC/SS)

SOC_ETM_MCPWM1_EVT_TIMER2_TEZ_ST Represents the status of MCPWM1_EVT_TIMER2_TEZ.

0: Not received

1: Received

(R/WTC/SS)

SOC_ETM_MCPWM1_EVT_TIMERO_TEP_ST Represents the status of MCPWM1_EVT_TIMERO_TEP.

0: Not received

1: Received

(R/WTC/SS)

Continued on the next page...

Register 14.6. SOC_ETM_EVT_ST3_REG (0x01C0)

Continued from the previous page...

SOC_ETM_MCPWM1_EVT_TIMER1_TEP_ST Represents the status of MCPWM1_EVT_TIMER1_TEP.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_MCPWM1_EVT_TIMER2_TEP_ST Represents the status of MCPWM1_EVT_TIMER2_TEP.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_MCPWM1_EVT_OP0_TEA_ST Represents the status of MCPWM1_EVT_OP0_TEA.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_MCPWM1_EVT_OP1_TEA_ST Represents the status of MCPWM1_EVT_OP1_TEA.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_MCPWM1_EVT_OP2_TEA_ST Represents the status of MCPWM1_EVT_OP2_TEA.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_MCPWM1_EVT_OP0_TEB_ST Represents the status of MCPWM1_EVT_OP0_TEB.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_MCPWM1_EVT_OP1_TEB_ST Represents the status of MCPWM1_EVT_OP1_TEB.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_MCPWM1_EVT_OP2_TEB_ST Represents the status of MCPWM1_EVT_OP2_TEB.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_MCPWM1_EVT_F0_ST Represents the status of MCPWM1_EVT_F0.
0: Not received
1: Received
(R/WTC/SS)

Continued on the next page...

Register 14.6. SOC_ETM_EVT_ST3_REG (0x01C0)

Continued from the previous page...

SOC_ETM_MCPWM1_EVT_F1_ST Represents the status of MCPWM1_EVT_F1.

0: Not received

1: Received

(R/WTC/SS)

SOC_ETM_MCPWM1_EVT_F2_ST Represents the status of MCPWM1_EVT_F2.

0: Not received

1: Received

(R/WTC/SS)

SOC_ETM_MCPWM1_EVT_F0_CLR_ST Represents the status of MCPWM1_EVT_F0_CLR.

0: Not received

1: Received

(R/WTC/SS)

SOC_ETM_MCPWM1_EVT_F1_CLR_ST Represents the status of MCPWM1_EVT_F1_CLR.

0: Not received

1: Received

(R/WTC/SS)

SOC_ETM_MCPWM1_EVT_F2_CLR_ST Represents the status of MCPWM1_EVT_F2_CLR.

0: Not received

1: Received

(R/WTC/SS)

SOC_ETM_MCPWM1_EVT_TZ0_CBC_ST Represents the status of MCPWM1_EVT_TZ0_CBC.

0: Not received

1: Received

(R/WTC/SS)

SOC_ETM_MCPWM1_EVT_TZ1_CBC_ST Represents the status of MCPWM1_EVT_TZ1_CBC.

0: Not received

1: Received

(R/WTC/SS)

SOC_ETM_MCPWM1_EVT_TZ2_CBC_ST Represents the status of MCPWM1_EVT_TZ2_CBC.

0: Not received

1: Received

(R/WTC/SS)

SOC_ETM_MCPWM1_EVT_TZ0_OST_ST Represents the status of MCPWM1_EVT_TZ0_OST.

0: Not received

1: Received

(R/WTC/SS)

Continued on the next page...

Register 14.6. SOC_ETM_EVT_ST3_REG (0x01C0)

Continued from the previous page...

SOC_ETM_MCPWM1_EVT_TZ1_OST_ST Represents the status of MCPWM1_EVT_TZ1_OST.

0: Not received

1: Received

(R/WTC/SS)

SOC_ETM_MCPWM1_EVT_TZ2_OST_ST Represents the status of MCPWM1_EVT_TZ2_OST.

0: Not received

1: Received

(R/WTC/SS)

SOC_ETM_MCPWM1_EVT_CAP0_ST Represents the status of MCPWM1_EVT_CAP0.

0: Not received

1: Received

(R/WTC/SS)

SOC_ETM_MCPWM1_EVT_CAP1_ST Represents the status of MCPWM1_EVT_CAP1.

0: Not received

1: Received

(R/WTC/SS)

SOC_ETM_MCPWM1_EVT_CAP2_ST Represents the status of MCPWM1_EVT_CAP2.

0: Not received

1: Received

(R/WTC/SS)

SOC_ETM_MCPWM1_EVT_OP0_TEE1_ST Represents the status of MCPWM1_EVT_OP0_TEE1.

0: Not received

1: Received

(R/WTC/SS)

SOC_ETM_MCPWM1_EVT_OP1_TEE1_ST Represents the status of MCPWM1_EVT_OP1_TEE1.

0: Not received

1: Received

(R/WTC/SS)

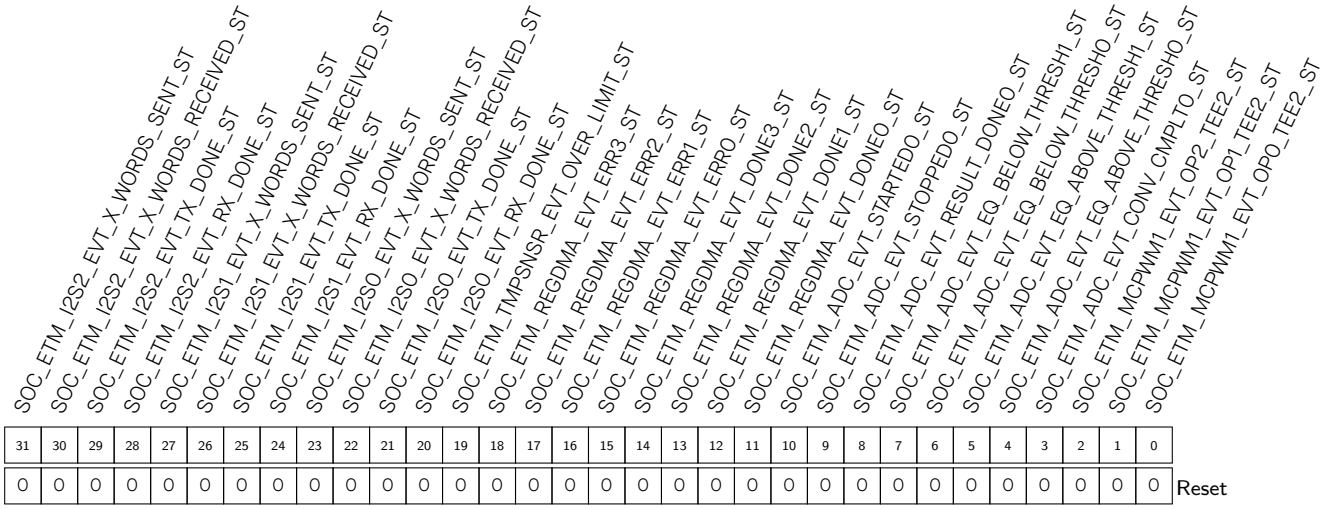
SOC_ETM_MCPWM1_EVT_OP2_TEE1_ST Represents the status of MCPWM1_EVT_OP2_TEE1.

0: Not received

1: Received

(R/WTC/SS)

Register 14.7. SOC_ETM_EVT_ST4_REG (0x01C8)



SOC_ETM_MCPWM1_EVT_OP0_TEE2_ST Represents the status of MCPWM1_EVT_OP0_TEE2.

0: Not received

1: Received

(R/WTC/SS)

SOC_ETM_MCPWM1_EVT_OP1_TEE2_ST Represents the status of MCPWM1_EVT_OP1_TEE2.

0: Not received

1: Received

(R/WTC/SS)

SOC_ETM_MCPWM1_EVT_OP2_TEE2_ST Represents the status of MCPWM1_EVT_OP2_TEE2.

0: Not received

1: Received

(R/WTC/SS)

SOC_ETM_ADC_EVT_CONV_CMPLTO_ST Represents the status of ADC_EVT_CONV_CMPLTO.

0: Not received

1: Received

(R/WTC/SS)

SOC_ETM_ADC_EVT_EQ_ABOVE_THRESH0_ST Represents the status of ADC_EVT_EQ_ABOVE_THRESH0.

0: Not received

1: Received

(R/WTC/SS)

SOC_ETM_ADC_EVT_EQ_ABOVE_THRESH1_ST Represents the status of ADC_EVT_EQ_ABOVE_THRESH1.

0: Not received

1: Received

(R/WTC/SS)

Continued on the next page...

Register 14.7. SOC_ETM_EVT_ST4_REG (0x01C8)

Continued from the previous page...

SOC_ETM_ADC_EVT_EQ_BELOW_THRESH0_ST Represents the status of ADC_EVT_EQ_BELOW_THRESH0.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_ADC_EVT_EQ_BELOW_THRESH1_ST Represents the status of ADC_EVT_EQ_BELOW_THRESH1.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_ADC_EVT_RESULT_DONE0_ST Represents the status of ADC_EVT_RESULT_DONE0.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_ADC_EVT_STOPPED0_ST Represents the status of ADC_EVT_STOPPED0.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_ADC_EVT_STARTED0_ST Represents the status of ADC_EVT_STARTED0.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_REGDMA_EVT_DONE0_ST Represents the status of REGDMA_EVT_DONE0.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_REGDMA_EVT_DONE1_ST Represents the status of REGDMA_EVT_DONE1.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_REGDMA_EVT_DONE2_ST Represents the status of REGDMA_EVT_DONE2.
0: Not received
1: Received
(R/WTC/SS)

Continued on the next page...

Register 14.7. SOC_ETM_EVT_ST4_REG (0x01C8)

Continued from the previous page...

SOC_ETM_REGDMA_EVT_DONE3_ST Represents the status of REGDMA_EVT_DONE3.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_REGDMA_EVT_ERR0_ST Represents the status of REGDMA_EVT_ERR0.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_REGDMA_EVT_ERR1_ST Represents the status of REGDMA_EVT_ERR1.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_REGDMA_EVT_ERR2_ST Represents the status of REGDMA_EVT_ERR2.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_REGDMA_EVT_ERR3_ST Represents the status of REGDMA_EVT_ERR3.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_TMPSNSR_EVT_OVER_LIMIT_ST Represents the status of TMPSNSR_EVT_OVER_LIMIT.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_I2S0_EVT_RX_DONE_ST Represents the status of I2S0_EVT_RX_DONE.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_I2S0_EVT_TX_DONE_ST Represents the status of I2S0_EVT_TX_DONE.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_I2S0_EVT_X_WORDS_RECEIVED_ST Represents the status of I2S0_EVT_X_WORDS_RECEIVED.
0: Not received
1: Received
(R/WTC/SS)

Continued on the next page...

Register 14.7. SOC_ETM_EVT_ST4_REG (0x01C8)

Continued from the previous page...

SOC_ETM_I2S0_EVT_X_WORDS_SENT_ST Represents the status of I2S0_EVT_X_WORDS_SENT.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_I2S1_EVT_RX_DONE_ST Represents the status of I2S1_EVT_RX_DONE.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_I2S1_EVT_TX_DONE_ST Represents the status of I2S1_EVT_TX_DONE.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_I2S1_EVT_X_WORDS_RECEIVED_ST Represents the status of I2S1_EVT_X_WORDS_RECEIVED.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_I2S1_EVT_X_WORDS_SENT_ST Represents the status of I2S1_EVT_X_WORDS_SENT.
0: Not received
1: Received
(R/WTC/SS)

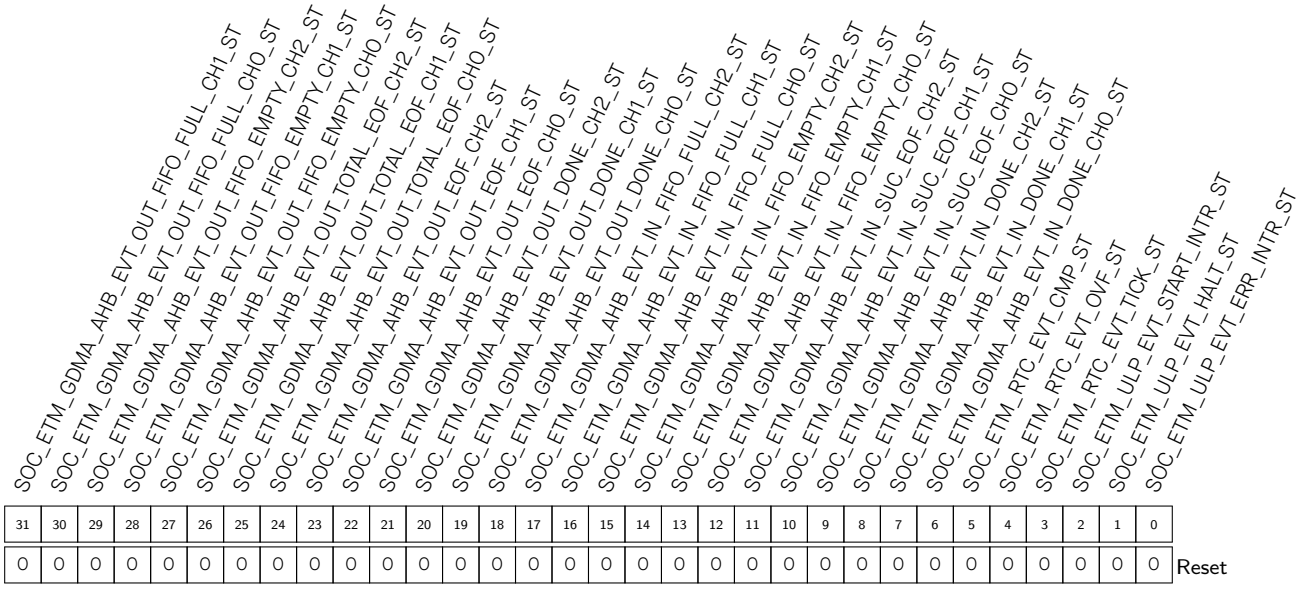
SOC_ETM_I2S2_EVT_RX_DONE_ST Represents the status of I2S2_EVT_RX_DONE.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_I2S2_EVT_TX_DONE_ST Represents the status of I2S2_EVT_TX_DONE.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_I2S2_EVT_X_WORDS_RECEIVED_ST Represents the status of I2S2_EVT_X_WORDS_RECEIVED.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_I2S2_EVT_X_WORDS_SENT_ST Represents the status of I2S2_EVT_X_WORDS_SENT.
0: Not received
1: Received
(R/WTC/SS)

Register 14.8. SOC_ETM_EVT_ST5_REG (0x01D0)



SOC_ETM_ULP_EVT_ERR_INTR_ST Represents the status of ULP_EVT_ERR_INTR.

0: Not received

1: Received

(R/WTC/SS)

SOC_ETM_ULP_EVT_HALT_ST Represents the status of ULP_EVT_HALT.

0: Not received

1: Received

(R/WTC/SS)

SOC_ETM_ULP_EVT_START_INTR_ST Represents the status of ULP_EVT_START_INTR.

0: Not received

1: Received

(R/WTC/SS)

SOC_ETM_RTC_EVT_TICK_ST Represents the status of RTC_EVT_TICK.

0: Not received

1: Received

(R/WTC/SS)

SOC_ETM_RTC_EVT_OVF_ST Represents the status of RTC_EVT_OVF.

0: Not received

1: Received

(R/WTC/SS)

SOC_ETM_RTC_EVT_CMP_ST Represents the status of RTC_EVT_CMP.

0: Not received

1: Received

(R/WTC/SS)

Continued on the next page...

Register 14.8. SOC_ETM_EVT_ST5_REG (0x01D0)

Continued from the previous page...

SOC_ETM_GDMA_AHB_EVT_IN_DONE_CH0_ST Represents the status of GDMA_AHB_EVT_IN_DONE_CH0.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_GDMA_AHB_EVT_IN_DONE_CH1_ST Represents the status of GDMA_AHB_EVT_IN_DONE_CH1.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_GDMA_AHB_EVT_IN_DONE_CH2_ST Represents the status of GDMA_AHB_EVT_IN_DONE_CH2.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_GDMA_AHB_EVT_IN_SUC_EOF_CH0_ST Represents the status of GDMA_AHB_EVT_IN_SUC_EOF_CH0.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_GDMA_AHB_EVT_IN_SUC_EOF_CH1_ST Represents the status of GDMA_AHB_EVT_IN_SUC_EOF_CH1.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_GDMA_AHB_EVT_IN_SUC_EOF_CH2_ST Represents the status of GDMA_AHB_EVT_IN_SUC_EOF_CH2.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_GDMA_AHB_EVT_IN_FIFO_EMPTY_CH0_ST Represents the status of GDMA_AHB_EVT_IN_FIFO_EMPTY_CH0.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_GDMA_AHB_EVT_IN_FIFO_EMPTY_CH1_ST Represents the status of GDMA_AHB_EVT_IN_FIFO_EMPTY_CH1.
0: Not received
1: Received
(R/WTC/SS)

Continued on the next page...

Register 14.8. SOC_ETM_EVT_ST5_REG (0x01D0)

Continued from the previous page...

SOC_ETM_GDMA_AHB_EVT_IN_FIFO_EMPTY_CH2_ST Represents the status of GDMA_AHB_EVT_IN_FIFO_EMPTY_CH2.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_GDMA_AHB_EVT_IN_FIFO_FULL_CH0_ST Represents the status of GDMA_AHB_EVT_IN_FIFO_FULL_CH0.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_GDMA_AHB_EVT_IN_FIFO_FULL_CH1_ST Represents the status of GDMA_AHB_EVT_IN_FIFO_FULL_CH1.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_GDMA_AHB_EVT_IN_FIFO_FULL_CH2_ST Represents the status of GDMA_AHB_EVT_IN_FIFO_FULL_CH2.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_GDMA_AHB_EVT_OUT_DONE_CH0_ST Represents the status of GDMA_AHB_EVT_OUT_DONE_CH0.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_GDMA_AHB_EVT_OUT_DONE_CH1_ST Represents the status of GDMA_AHB_EVT_OUT_DONE_CH1.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_GDMA_AHB_EVT_OUT_DONE_CH2_ST Represents the status of GDMA_AHB_EVT_OUT_DONE_CH2.
0: Not received
1: Received
(R/WTC/SS)

Continued on the next page...

Register 14.8. SOC_ETM_EVT_ST5_REG (0x01D0)

Continued from the previous page...

SOC_ETM_GDMA_AHB_EVT_OUT_EOF_CH0_ST Represents the status of GDMA_AHB_EVT_OUT_EOF_CH0.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_GDMA_AHB_EVT_OUT_EOF_CH1_ST Represents the status of GDMA_AHB_EVT_OUT_EOF_CH1.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_GDMA_AHB_EVT_OUT_EOF_CH2_ST Represents the status of GDMA_AHB_EVT_OUT_EOF_CH2.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_GDMA_AHB_EVT_OUT_TOTAL_EOF_CH0_ST Represents the status of GDMA_AHB_EVT_OUT_TOTAL_EOF_CH0.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_GDMA_AHB_EVT_OUT_TOTAL_EOF_CH1_ST Represents the status of GDMA_AHB_EVT_OUT_TOTAL_EOF_CH1.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_GDMA_AHB_EVT_OUT_TOTAL_EOF_CH2_ST Represents the status of GDMA_AHB_EVT_OUT_TOTAL_EOF_CH2.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_GDMA_AHB_EVT_OUT_FIFO_EMPTY_CH0_ST Represents the status of GDMA_AHB_EVT_OUT_FIFO_EMPTY_CH0.
0: Not received
1: Received
(R/WTC/SS)

Continued on the next page...

Register 14.8. SOC_ETM_EVT_ST5_REG (0x01D0)

Continued from the previous page...

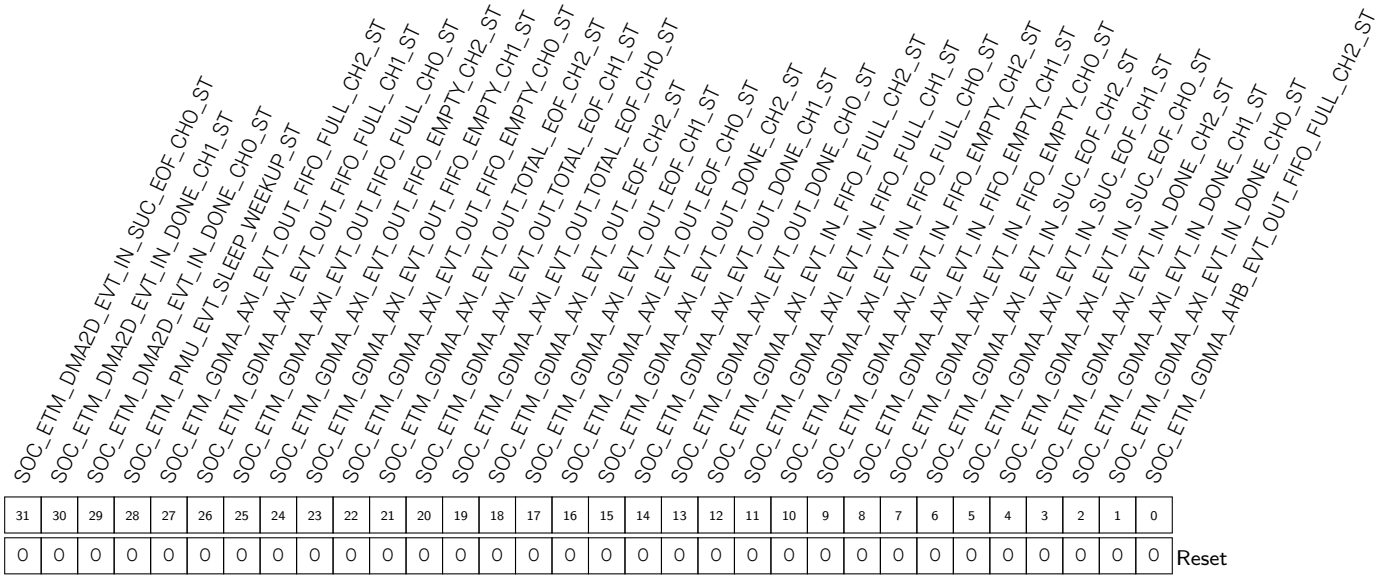
SOC_ETM_GDMA_AHB_EVT_OUT_FIFO_EMPTY_CH1_ST Represents the status of GDMA_AHB_EVT_OUT_FIFO_EMPTY_CH1.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_GDMA_AHB_EVT_OUT_FIFO_EMPTY_CH2_ST Represents the status of GDMA_AHB_EVT_OUT_FIFO_EMPTY_CH2.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_GDMA_AHB_EVT_OUT_FIFO_FULL_CH0_ST Represents the status of GDMA_AHB_EVT_OUT_FIFO_FULL_CH0.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_GDMA_AHB_EVT_OUT_FIFO_FULL_CH1_ST Represents the status of GDMA_AHB_EVT_OUT_FIFO_FULL_CH1.
0: Not received
1: Received
(R/WTC/SS)

Register 14.9. SOC_ETM_EVT_ST6_REG (0x01D8)



SOC_ETM_GDMA_AHB_EVT_OUT_FIFO_FULL_CH2_ST Represents the status of GDMA_AHB_EVT_OUT_FIFO_FULL_CH2.

0: Not received

1: Received

(R/WTC/SS)

SOC_ETM_GDMA_AXI_EVT_IN_DONE_CH0_ST Represents the status of GDMA_AXI_EVT_IN_DONE_CH0.

0: Not received

1: Received

(R/WTC/SS)

SOC_ETM_GDMA_AXI_EVT_IN_DONE_CH1_ST Represents the status of GDMA_AXI_EVT_IN_DONE_CH1.

0: Not received

1: Received

(R/WTC/SS)

SOC_ETM_GDMA_AXI_EVT_IN_DONE_CH2_ST Represents the status of GDMA_AXI_EVT_IN_DONE_CH2.

0: Not received

1: Received

(R/WTC/SS)

SOC_ETM_GDMA_AXI_EVT_IN_SUC_EOF_CH0_ST Represents the status of GDMA_AXI_EVT_IN_SUC_EOF_CH0.

0: Not received

1: Received

(R/WTC/SS)

Continued on the next page...

Register 14.9. SOC_ETM_EVT_ST6_REG (0x01D8)

Continued from the previous page...

SOC_ETM_GDMA_AXI_EVT_IN_SUC_EOF_CH1_ST Represents the status of GDMA_AXI_EVT_IN_SUC_EOF_CH1.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_GDMA_AXI_EVT_IN_SUC_EOF_CH2_ST Represents the status of GDMA_AXI_EVT_IN_SUC_EOF_CH2.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_GDMA_AXI_EVT_IN_FIFO_EMPTY_CH0_ST Represents the status of GDMA_AXI_EVT_IN_FIFO_EMPTY_CH0.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_GDMA_AXI_EVT_IN_FIFO_EMPTY_CH1_ST Represents the status of GDMA_AXI_EVT_IN_FIFO_EMPTY_CH1.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_GDMA_AXI_EVT_IN_FIFO_EMPTY_CH2_ST Represents the status of GDMA_AXI_EVT_IN_FIFO_EMPTY_CH2.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_GDMA_AXI_EVT_IN_FIFO_FULL_CH0_ST Represents the status of GDMA_AXI_EVT_IN_FIFO_FULL_CH0.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_GDMA_AXI_EVT_IN_FIFO_FULL_CH1_ST Represents the status of GDMA_AXI_EVT_IN_FIFO_FULL_CH1.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_GDMA_AXI_EVT_IN_FIFO_FULL_CH2_ST Represents the status of GDMA_AXI_EVT_IN_FIFO_FULL_CH2.
0: Not received
1: Received
(R/WTC/SS)

Continued on the next page...

Register 14.9. SOC_ETM_EVT_ST6_REG (0x01D8)

Continued from the previous page...

SOC_ETM_GDMA_AXI_EVT_OUT_DONE_CH0_ST Represents the status of GDMA_AXI_EVT_OUT_DONE_CH0.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_GDMA_AXI_EVT_OUT_DONE_CH1_ST Represents the status of GDMA_AXI_EVT_OUT_DONE_CH1.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_GDMA_AXI_EVT_OUT_DONE_CH2_ST Represents the status of GDMA_AXI_EVT_OUT_DONE_CH2.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_GDMA_AXI_EVT_OUT_EOF_CH0_ST Represents the status of GDMA_AXI_EVT_OUT_EOF_CH0.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_GDMA_AXI_EVT_OUT_EOF_CH1_ST Represents the status of GDMA_AXI_EVT_OUT_EOF_CH1.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_GDMA_AXI_EVT_OUT_EOF_CH2_ST Represents the status of GDMA_AXI_EVT_OUT_EOF_CH2.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_GDMA_AXI_EVT_OUT_TOTAL_EOF_CH0_ST Represents the status of GDMA_AXI_EVT_OUT_TOTAL_EOF_CH0.
0: Not received
1: Received
(R/WTC/SS)

Continued on the next page...

Register 14.9. SOC_ETM_EVT_ST6_REG (0x01D8)

Continued from the previous page...

SOC_ETM_GDMA_AXI_EVT_OUT_TOTAL_EOF_CH1_ST Represents the status of GDMA_AXI_EVT_OUT_TOTAL_EOF_CH1.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_GDMA_AXI_EVT_OUT_TOTAL_EOF_CH2_ST Represents the status of GDMA_AXI_EVT_OUT_TOTAL_EOF_CH2.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_GDMA_AXI_EVT_OUT_FIFO_EMPTY_CH0_ST Represents the status of GDMA_AXI_EVT_OUT_FIFO_EMPTY_CH0.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_GDMA_AXI_EVT_OUT_FIFO_EMPTY_CH1_ST Represents the status of GDMA_AXI_EVT_OUT_FIFO_EMPTY_CH1.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_GDMA_AXI_EVT_OUT_FIFO_EMPTY_CH2_ST Represents the status of GDMA_AXI_EVT_OUT_FIFO_EMPTY_CH2.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_GDMA_AXI_EVT_OUT_FIFO_FULL_CH0_ST Represents the status of GDMA_AXI_EVT_OUT_FIFO_FULL_CH0.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_GDMA_AXI_EVT_OUT_FIFO_FULL_CH1_ST Represents the status of GDMA_AXI_EVT_OUT_FIFO_FULL_CH1.
0: Not received
1: Received
(R/WTC/SS)

Continued on the next page...

Register 14.9. SOC_ETM_EVT_ST6_REG (0x01D8)

Continued from the previous page...

SOC_ETM_GDMA_AXI_EVT_OUT_FIFO_FULL_CH2_ST Represents the status of GDMA_AXI_EVT_OUT_FIFO_FULL_CH2.
0: Not received
1: Received
(R/WTC/SS)

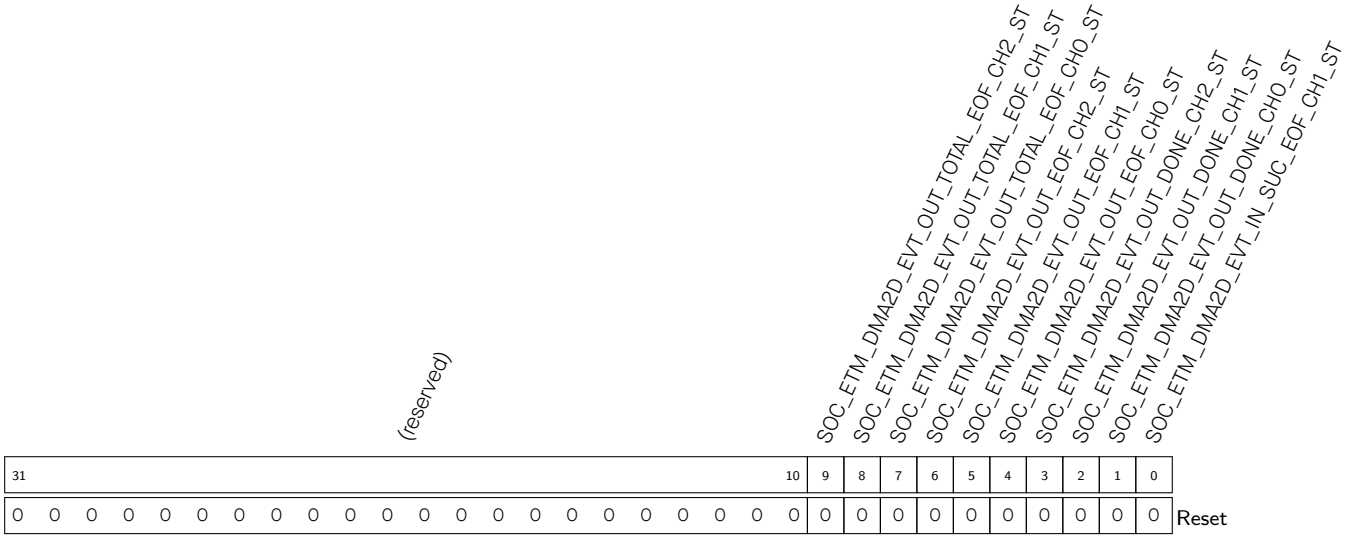
SOC_ETM_PMU_EVT_SLEEP_WEEKUP_ST Represents the status of PMU_EVT_SLEEP_WEEKUP.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_DMA2D_EVT_IN_DONE_CH0_ST Represents the status of DMA2D_EVT_IN_DONE_CH0.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_DMA2D_EVT_IN_DONE_CH1_ST Represents the status of DMA2D_EVT_IN_DONE_CH1.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_DMA2D_EVT_IN_SUC_EOF_CH0_ST Represents the status of DMA2D_EVT_IN_SUC_EOF_CH0.
0: Not received
1: Received
(R/WTC/SS)

Register 14.10. SOC_ETM_EVT_ST7_REG (0x01E0)



SOC_ETM_DMA2D_EVT_IN_SUC_EOF_CH1_ST Represents the status of DMA2D_EVT_IN_SUC_EOF_CH1.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_DMA2D_EVT_OUT_DONE_CH0_ST Represents the status of DMA2D_EVT_OUT_DONE_CH0.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_DMA2D_EVT_OUT_DONE_CH1_ST Represents the status of DMA2D_EVT_OUT_DONE_CH1.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_DMA2D_EVT_OUT_DONE_CH2_ST Represents the status of DMA2D_EVT_OUT_DONE_CH2.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_DMA2D_EVT_OUT_EOF_CH0_ST Represents the status of DMA2D_EVT_OUT_EOF_CH0.
0: Not received
1: Received
(R/WTC/SS)

Continued on the next page...

Register 14.10. SOC_ETM_EVT_ST7_REG (0x01E0)

Continued from the previous page...

SOC_ETM_DMA2D_EVT_OUT_EOF_CH1_ST Represents the status of DMA2D_EVT_OUT_EOF_CH1.
0: Not received
1: Received
(R/WTC/SS)

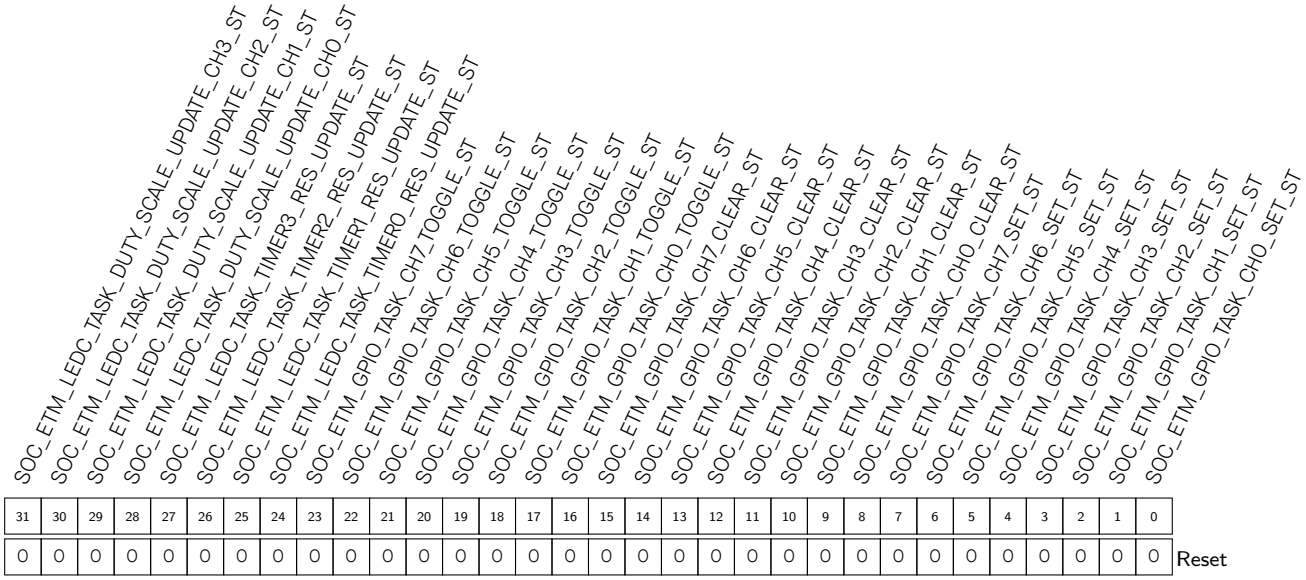
SOC_ETM_DMA2D_EVT_OUT_EOF_CH2_ST Represents the status of DMA2D_EVT_OUT_EOF_CH2.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_DMA2D_EVT_OUT_TOTAL_EOF_CH0_ST Represents the status of DMA2D_EVT_OUT_TOTAL_EOF_CH0.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_DMA2D_EVT_OUT_TOTAL_EOF_CH1_ST Represents the status of DMA2D_EVT_OUT_TOTAL_EOF_CH1.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_DMA2D_EVT_OUT_TOTAL_EOF_CH2_ST Represents the status of DMA2D_EVT_OUT_TOTAL_EOF_CH2.
0: Not received
1: Received
(R/WTC/SS)

Register 14.11. SOC_ETM_TASK_ST0_REG (0x01E8)



SOC_ETM_GPIO_TASK_CH0_SET_ST Represents the status of GPIO_TASK_CH0_SET.

0: Not generated

1: Generated

(R/WTC/SS)

SOC_ETM_GPIO_TASK_CH1_SET_ST Represents the status of GPIO_TASK_CH1_SET.

0: Not generated

1: Generated

(R/WTC/SS)

SOC_ETM_GPIO_TASK_CH2_SET_ST Represents the status of GPIO_TASK_CH2_SET.

0: Not generated

1: Generated

(R/WTC/SS)

SOC_ETM_GPIO_TASK_CH3_SET_ST Represents the status of GPIO_TASK_CH3_SET.

0: Not generated

1: Generated

(R/WTC/SS)

SOC_ETM_GPIO_TASK_CH4_SET_ST Represents the status of GPIO_TASK_CH4_SET.

0: Not generated

1: Generated

(R/WTC/SS)

SOC_ETM_GPIO_TASK_CH5_SET_ST Represents the status of GPIO_TASK_CH5_SET.

0: Not generated

1: Generated

(R/WTC/SS)

SOC_ETM_GPIO_TASK_CH6_SET_ST Represents the status of GPIO_TASK_CH6_SET.

0: Not generated

1: Generated

(R/WTC/SS)

Continued on the next page...

Register 14.11. SOC_ETM_TASK_ST0_REG (0x01E8)

Continued from the previous page...

SOC_ETM_GPIO_TASK_CH7_SET_ST Represents the status of GPIO_TASK_CH7_SET.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_GPIO_TASK_CH0_CLEAR_ST Represents the status of GPIO_TASK_CH0_CLEAR.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_GPIO_TASK_CH1_CLEAR_ST Represents the status of GPIO_TASK_CH1_CLEAR.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_GPIO_TASK_CH2_CLEAR_ST Represents the status of GPIO_TASK_CH2_CLEAR.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_GPIO_TASK_CH3_CLEAR_ST Represents the status of GPIO_TASK_CH3_CLEAR.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_GPIO_TASK_CH4_CLEAR_ST Represents the status of GPIO_TASK_CH4_CLEAR.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_GPIO_TASK_CH5_CLEAR_ST Represents the status of GPIO_TASK_CH5_CLEAR.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_GPIO_TASK_CH6_CLEAR_ST Represents the status of GPIO_TASK_CH6_CLEAR.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_GPIO_TASK_CH7_CLEAR_ST Represents the status of GPIO_TASK_CH7_CLEAR.
0: Not generated
1: Generated
(R/WTC/SS)

Continued on the next page...

Register 14.11. SOC_ETM_TASK_ST0_REG (0x01E8)

Continued from the previous page...

SOC_ETM_GPIO_TASK_CH0_TOGGLE_ST Represents the status of GPIO_TASK_CH0_TOGGLE.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_GPIO_TASK_CH1_TOGGLE_ST Represents the status of GPIO_TASK_CH1_TOGGLE.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_GPIO_TASK_CH2_TOGGLE_ST Represents the status of GPIO_TASK_CH2_TOGGLE.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_GPIO_TASK_CH3_TOGGLE_ST Represents the status of GPIO_TASK_CH3_TOGGLE.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_GPIO_TASK_CH4_TOGGLE_ST Represents the status of GPIO_TASK_CH4_TOGGLE.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_GPIO_TASK_CH5_TOGGLE_ST Represents the status of GPIO_TASK_CH5_TOGGLE.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_GPIO_TASK_CH6_TOGGLE_ST Represents the status of GPIO_TASK_CH6_TOGGLE.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_GPIO_TASK_CH7_TOGGLE_ST Represents the status of GPIO_TASK_CH7_TOGGLE.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_LEDC_TASK_TIMER0_RES_UPDATE_ST Represents the status of LEDC_TASK_TIMER0_RES_UPDATE.
0: Not generated
1: Generated
(R/WTC/SS)

Continued on the next page...

Register 14.11. SOC_ETM_TASK_ST0_REG (0x01E8)

Continued from the previous page...

SOC_ETM_LEDC_TASK_TIMER1_RES_UPDATE_ST Represents the status of LEDC_TASK_TIMER1_RES_UPDATE.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_LEDC_TASK_TIMER2_RES_UPDATE_ST Represents the status of LEDC_TASK_TIMER2_RES_UPDATE.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_LEDC_TASK_TIMER3_RES_UPDATE_ST Represents the status of LEDC_TASK_TIMER3_RES_UPDATE.
0: Not generated
1: Generated
(R/WTC/SS)

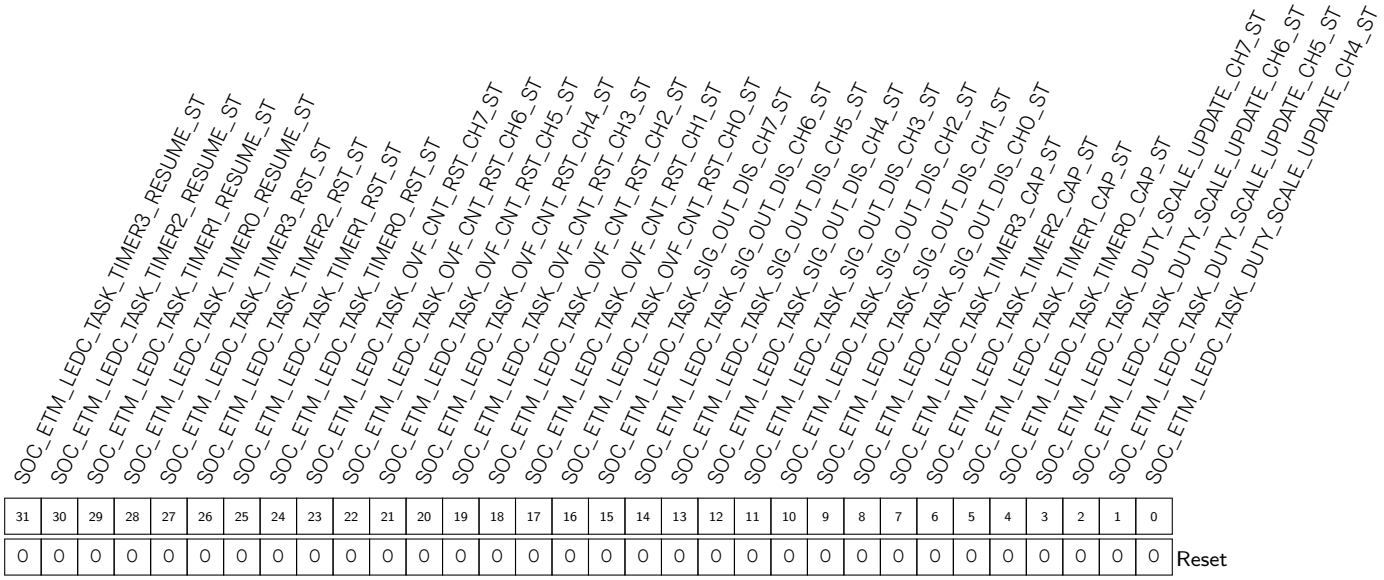
SOC_ETM_LEDC_TASK_DUTY_SCALE_UPDATE_CH0_ST Represents the status of LEDC_TASK_DUTY_SCALE_UPDATE_CH0.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_LEDC_TASK_DUTY_SCALE_UPDATE_CH1_ST Represents the status of LEDC_TASK_DUTY_SCALE_UPDATE_CH1.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_LEDC_TASK_DUTY_SCALE_UPDATE_CH2_ST Represents the status of LEDC_TASK_DUTY_SCALE_UPDATE_CH2.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_LEDC_TASK_DUTY_SCALE_UPDATE_CH3_ST Represents the status of LEDC_TASK_DUTY_SCALE_UPDATE_CH3.
0: Not generated
1: Generated
(R/WTC/SS)

Register 14.12. SOC_ETM_TASK_ST1_REG (0x01F0)



31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Reset

SOC_ETM_LEDC_TASK_DUTY_SCALE_UPDATE_CH4_ST Represents the status of LEDC_TASK_DUTY_SCALE_UPDATE_CH4.

0: Not generated

1: Generated

(R/WTC/SS)

SOC_ETM_LEDC_TASK_DUTY_SCALE_UPDATE_CH5_ST Represents the status of LEDC_TASK_DUTY_SCALE_UPDATE_CH5.

0: Not generated

1: Generated

(R/WTC/SS)

SOC_ETM_LEDC_TASK_DUTY_SCALE_UPDATE_CH6_ST Represents the status of LEDC_TASK_DUTY_SCALE_UPDATE_CH6.

0: Not generated

1: Generated

(R/WTC/SS)

SOC_ETM_LEDC_TASK_DUTY_SCALE_UPDATE_CH7_ST Represents the status of LEDC_TASK_DUTY_SCALE_UPDATE_CH7.

0: Not generated

1: Generated

(R/WTC/SS)

SOC_ETM_LEDC_TASK_TIMER0_CAP_ST Represents the status of LEDC_TASK_TIMER0_CAP.

0: Not generated

1: Generated

(R/WTC/SS)

SOC_ETM_LEDC_TASK_TIMER1_CAP_ST Represents the status of LEDC_TASK_TIMER1_CAP.

0: Not generated

1: Generated

(R/WTC/SS)

Continued on the next page...

Register 14.12. SOC_ETM_TASK_ST1_REG (0x01F0)

Continued from the previous page...

SOC_ETM_LEDC_TASK_TIMER2_CAP_ST Represents the status of LEDC_TASK_TIMER2_CAP.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_LEDC_TASK_TIMER3_CAP_ST Represents the status of LEDC_TASK_TIMER3_CAP.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_LEDC_TASK_SIG_OUT_DIS_CH0_ST Represents the status of LEDC_TASK_SIG_OUT_DIS_CH0.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_LEDC_TASK_SIG_OUT_DIS_CH1_ST Represents the status of LEDC_TASK_SIG_OUT_DIS_CH1.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_LEDC_TASK_SIG_OUT_DIS_CH2_ST Represents the status of LEDC_TASK_SIG_OUT_DIS_CH2.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_LEDC_TASK_SIG_OUT_DIS_CH3_ST Represents the status of LEDC_TASK_SIG_OUT_DIS_CH3.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_LEDC_TASK_SIG_OUT_DIS_CH4_ST Represents the status of LEDC_TASK_SIG_OUT_DIS_CH4.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_LEDC_TASK_SIG_OUT_DIS_CH5_ST Represents the status of LEDC_TASK_SIG_OUT_DIS_CH5.
0: Not generated
1: Generated
(R/WTC/SS)

Continued on the next page...

Register 14.12. SOC_ETM_TASK_ST1_REG (0x01F0)

Continued from the previous page...

SOC_ETM_LEDC_TASK_SIG_OUT_DIS_CH6_ST Represents the status of LEDC_TASK_SIG_OUT_DIS_CH6.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_LEDC_TASK_SIG_OUT_DIS_CH7_ST Represents the status of LEDC_TASK_SIG_OUT_DIS_CH7.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_LEDC_TASK_OVF_CNT_RST_CH0_ST Represents the status of LEDC_TASK_OVF_CNT_RST_CH0.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_LEDC_TASK_OVF_CNT_RST_CH1_ST Represents the status of LEDC_TASK_OVF_CNT_RST_CH1.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_LEDC_TASK_OVF_CNT_RST_CH2_ST Represents the status of LEDC_TASK_OVF_CNT_RST_CH2.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_LEDC_TASK_OVF_CNT_RST_CH3_ST Represents the status of LEDC_TASK_OVF_CNT_RST_CH3.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_LEDC_TASK_OVF_CNT_RST_CH4_ST Represents the status of LEDC_TASK_OVF_CNT_RST_CH4.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_LEDC_TASK_OVF_CNT_RST_CH5_ST Represents the status of LEDC_TASK_OVF_CNT_RST_CH5.
0: Not generated
1: Generated
(R/WTC/SS)

Continued on the next page...

Register 14.12. SOC_ETM_TASK_ST1_REG (0x01F0)

Continued from the previous page...

SOC_ETM_LEDC_TASK_OVF_CNT_RST_CH6_ST Represents the status of LEDC_TASK_OVF_CNT_RST_CH6.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_LEDC_TASK_OVF_CNT_RST_CH7_ST Represents the status of LEDC_TASK_OVF_CNT_RST_CH7.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_LEDC_TASK_TIMER0_RST_ST Represents the status of LEDC_TASK_TIMER0_RST.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_LEDC_TASK_TIMER1_RST_ST Represents the status of LEDC_TASK_TIMER1_RST.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_LEDC_TASK_TIMER2_RST_ST Represents the status of LEDC_TASK_TIMER2_RST.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_LEDC_TASK_TIMER3_RST_ST Represents the status of LEDC_TASK_TIMER3_RST.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_LEDC_TASK_TIMER0_RESUME_ST Represents the status of LEDC_TASK_TIMER0_RESUME.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_LEDC_TASK_TIMER1_RESUME_ST Represents the status of LEDC_TASK_TIMER1_RESUME.
0: Not generated
1: Generated
(R/WTC/SS)

Continued on the next page...

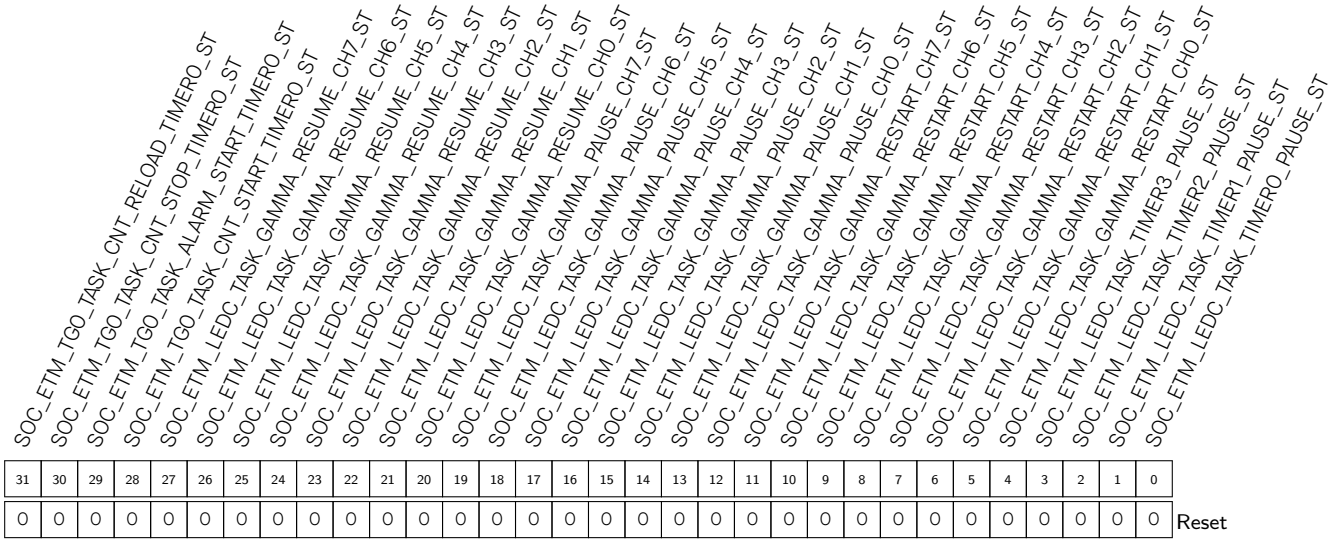
Register 14.12. SOC_ETM_TASK_ST1_REG (0x01F0)

Continued from the previous page...

SOC_ETM_LEDC_TASK_TIMER2_RESUME_ST Represents the status of LEDC_TASK_TIMER2_RESUME.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_LEDC_TASK_TIMER3_RESUME_ST Represents the status of LEDC_TASK_TIMER3_RESUME.
0: Not generated
1: Generated
(R/WTC/SS)

Register 14.13. SOC_ETM_TASK_ST2_REG (0x01F8)



31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Reset

SOC_ETM_LEDC_TASK_TIMER0_PAUSE_ST Represents the status of LEDC_TASK_TIMER0_PAUSE.

0: Not generated

1: Generated

(R/WTC/SS)

SOC_ETM_LEDC_TASK_TIMER1_PAUSE_ST Represents the status of LEDC_TASK_TIMER1_PAUSE.

0: Not generated

1: Generated

(R/WTC/SS)

SOC_ETM_LEDC_TASK_TIMER2_PAUSE_ST Represents the status of LEDC_TASK_TIMER2_PAUSE.

0: Not generated

1: Generated

(R/WTC/SS)

SOC_ETM_LEDC_TASK_TIMER3_PAUSE_ST Represents the status of LEDC_TASK_TIMER3_PAUSE.

0: Not generated

1: Generated

(R/WTC/SS)

SOC_ETM_LEDC_TASK_GAMMA_RESTART_CH0_ST Represents the status of LEDC_TASK_GAMMA_RESTART_CH0.

0: Not generated

1: Generated

(R/WTC/SS)

Continued on the next page...

Register 14.13. SOC_ETM_TASK_ST2_REG (0x01F8)

Continued from the previous page...

SOC_ETM_LEDC_TASK_GAMMA_RESTART_CH1_ST Represents the status of LEDC_TASK_GAMMA_RESTART_CH1.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_LEDC_TASK_GAMMA_RESTART_CH2_ST Represents the status of LEDC_TASK_GAMMA_RESTART_CH2.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_LEDC_TASK_GAMMA_RESTART_CH3_ST Represents the status of LEDC_TASK_GAMMA_RESTART_CH3.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_LEDC_TASK_GAMMA_RESTART_CH4_ST Represents the status of LEDC_TASK_GAMMA_RESTART_CH4.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_LEDC_TASK_GAMMA_RESTART_CH5_ST Represents the status of LEDC_TASK_GAMMA_RESTART_CH5.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_LEDC_TASK_GAMMA_RESTART_CH6_ST Represents the status of LEDC_TASK_GAMMA_RESTART_CH6.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_LEDC_TASK_GAMMA_RESTART_CH7_ST Represents the status of LEDC_TASK_GAMMA_RESTART_CH7.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_LEDC_TASK_GAMMA_PAUSE_CH0_ST Represents the status of LEDC_TASK_GAMMA_PAUSE_CH0.
0: Not generated
1: Generated
(R/WTC/SS)

Continued on the next page...

Register 14.13. SOC_ETM_TASK_ST2_REG (0x01F8)

Continued from the previous page...

SOC_ETM_LEDC_TASK_GAMMA_PAUSE_CH1_ST Represents the status of LEDC_TASK_GAMMA_PAUSE_CH1.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_LEDC_TASK_GAMMA_PAUSE_CH2_ST Represents the status of LEDC_TASK_GAMMA_PAUSE_CH2.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_LEDC_TASK_GAMMA_PAUSE_CH3_ST Represents the status of LEDC_TASK_GAMMA_PAUSE_CH3.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_LEDC_TASK_GAMMA_PAUSE_CH4_ST Represents the status of LEDC_TASK_GAMMA_PAUSE_CH4.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_LEDC_TASK_GAMMA_PAUSE_CH5_ST Represents the status of LEDC_TASK_GAMMA_PAUSE_CH5.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_LEDC_TASK_GAMMA_PAUSE_CH6_ST Represents the status of LEDC_TASK_GAMMA_PAUSE_CH6.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_LEDC_TASK_GAMMA_PAUSE_CH7_ST Represents the status of LEDC_TASK_GAMMA_PAUSE_CH7.
0: Not generated
1: Generated
(R/WTC/SS)

Continued on the next page...

Register 14.13. SOC_ETM_TASK_ST2_REG (0x01F8)

Continued from the previous page...

SOC_ETM_LEDC_TASK_GAMMA_RESUME_CH0_ST Represents the status of LEDC_TASK_GAMMA_RESUME_CH0.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_LEDC_TASK_GAMMA_RESUME_CH1_ST Represents the status of LEDC_TASK_GAMMA_RESUME_CH1.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_LEDC_TASK_GAMMA_RESUME_CH2_ST Represents the status of LEDC_TASK_GAMMA_RESUME_CH2.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_LEDC_TASK_GAMMA_RESUME_CH3_ST Represents the status of LEDC_TASK_GAMMA_RESUME_CH3.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_LEDC_TASK_GAMMA_RESUME_CH4_ST Represents the status of LEDC_TASK_GAMMA_RESUME_CH4.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_LEDC_TASK_GAMMA_RESUME_CH5_ST Represents the status of LEDC_TASK_GAMMA_RESUME_CH5.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_LEDC_TASK_GAMMA_RESUME_CH6_ST Represents the status of LEDC_TASK_GAMMA_RESUME_CH6.
0: Not generated
1: Generated
(R/WTC/SS)

Continued on the next page...

Register 14.13. SOC_ETM_TASK_ST2_REG (0x01F8)

Continued from the previous page...

SOC_ETM_LEDC_TASK_GAMMA_RESUME_CH7_ST Represents the status of LEDC_TASK_GAMMA_RESUME_CH7.
0: Not generated
1: Generated
(R/WTC/SS)

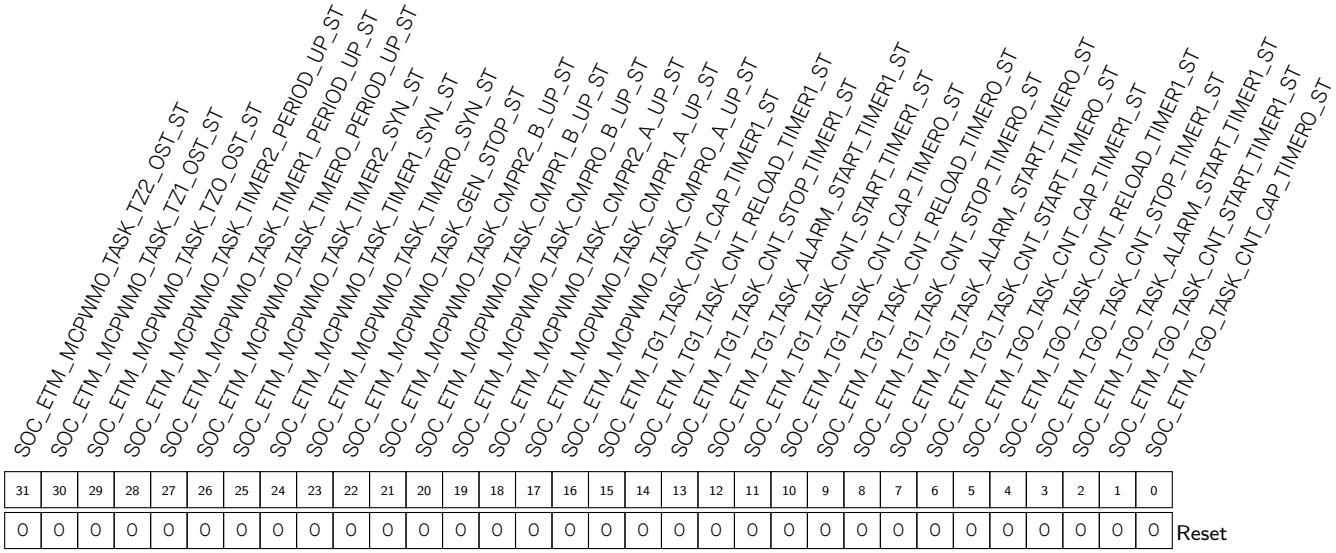
SOC_ETM_TGO_TASK_CNT_START_TIMER0_ST Represents the status of TGO_TASK_CNT_START_TIMER0.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_TGO_TASK_ALARM_START_TIMER0_ST Represents the status of TGO_TASK_ALARM_START_TIMER0.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_TGO_TASK_CNT_STOP_TIMER0_ST Represents the status of TGO_TASK_CNT_STOP_TIMER0.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_TGO_TASK_CNT_RELOAD_TIMER0_ST Represents the status of TGO_TASK_CNT_RELOAD_TIMER0.
0: Not generated
1: Generated
(R/WTC/SS)

Register 14.14. SOC_ETM_TASK_ST3_REG (0x0200)



SOC_ETM_TGO_TASK_CNT_CAP_TIMER0_ST Represents the status of TGO_TASK_CNT_CAP_TIMER0.

0: Not generated

1: Generated

(R/WTC/SS)

SOC_ETM_TGO_TASK_CNT_START_TIMER1_ST Represents the status of TGO_TASK_CNT_START_TIMER1.

0: Not generated

1: Generated

(R/WTC/SS)

SOC_ETM_TGO_TASK_ALARM_START_TIMER1_ST Represents the status of TGO_TASK_ALARM_START_TIMER1.

0: Not generated

1: Generated

(R/WTC/SS)

SOC_ETM_TGO_TASK_CNT_STOP_TIMER1_ST Represents the status of TGO_TASK_CNT_STOP_TIMER1.

0: Not generated

1: Generated

(R/WTC/SS)

SOC_ETM_TGO_TASK_CNT_RELOAD_TIMER1_ST Represents the status of TGO_TASK_CNT_RELOAD_TIMER1.

0: Not generated

1: Generated

(R/WTC/SS)

Continued on the next page...

Register 14.14. SOC_ETM_TASK_ST3_REG (0x0200)

Continued from the previous page...

SOC_ETM_TG0_TASK_CNT_CAP_TIMER1_ST Represents the status of TG0_TASK_CNT_CAP_TIMER1.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_TG1_TASK_CNT_START_TIMER0_ST Represents the status of TG1_TASK_CNT_START_TIMER0.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_TG1_TASK_ALARM_START_TIMER0_ST Represents the status of TG1_TASK_ALARM_START_TIMER0.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_TG1_TASK_CNT_STOP_TIMER0_ST Represents the status of TG1_TASK_CNT_STOP_TIMER0.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_TG1_TASK_CNT_RELOAD_TIMER0_ST Represents the status of TG1_TASK_CNT_RELOAD_TIMER0.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_TG1_TASK_CNT_CAP_TIMER0_ST Represents the status of TG1_TASK_CNT_CAP_TIMER0.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_TG1_TASK_CNT_START_TIMER1_ST Represents the status of TG1_TASK_CNT_START_TIMER1.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_TG1_TASK_ALARM_START_TIMER1_ST Represents the status of TG1_TASK_ALARM_START_TIMER1.
0: Not generated
1: Generated
(R/WTC/SS)

Continued on the next page...

Register 14.14. SOC_ETM_TASK_ST3_REG (0x0200)

Continued from the previous page...

SOC_ETM_TG1_TASK_CNT_STOP_TIMER1_ST Represents the status of TG1_TASK_CNT_STOP_TIMER1.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_TG1_TASK_CNT_RELOAD_TIMER1_ST Represents the status of TG1_TASK_CNT_RELOAD_TIMER1.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_TG1_TASK_CNT_CAP_TIMER1_ST Represents the status of TG1_TASK_CNT_CAP_TIMER1.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_MCPWM0_TASK_CMPR0_A_UP_ST Represents the status of MCPWM0_TASK_CMPR0_A_UP.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_MCPWM0_TASK_CMPR1_A_UP_ST Represents the status of MCPWM0_TASK_CMPR1_A_UP.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_MCPWM0_TASK_CMPR2_A_UP_ST Represents the status of MCPWM0_TASK_CMPR2_A_UP.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_MCPWM0_TASK_CMPR0_B_UP_ST Represents the status of MCPWM0_TASK_CMPR0_B_UP.
0: Not generated
1: Generated
(R/WTC/SS)

Continued on the next page...

Register 14.14. SOC_ETM_TASK_ST3_REG (0x0200)

Continued from the previous page...

SOC_ETM_MCPWM0_TASK_CMPR1_B_UP_ST Represents the status of MCPWM0_TASK_CMPR1_B_UP.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_MCPWM0_TASK_CMPR2_B_UP_ST Represents the status of MCPWM0_TASK_CMPR2_B_UP.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_MCPWM0_TASK_GEN_STOP_ST Represents the status of MCPWM0_TASK_GEN_STOP.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_MCPWM0_TASK_TIMER0_SYN_ST Represents the status of MCPWM0_TASK_TIMER0_SYN.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_MCPWM0_TASK_TIMER1_SYN_ST Represents the status of MCPWM0_TASK_TIMER1_SYN.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_MCPWM0_TASK_TIMER2_SYN_ST Represents the status of MCPWM0_TASK_TIMER2_SYN.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_MCPWM0_TASK_TIMER0_PERIOD_UP_ST Represents the status of MCPWM0_TASK_TIMER0_PERIOD_UP.
0: Not generated
1: Generated
(R/WTC/SS)

Continued on the next page...

Register 14.14. SOC_ETM_TASK_ST3_REG (0x0200)

Continued from the previous page...

SOC_ETM_MCPWM0_TASK_TIMER1_PERIOD_UP_ST Represents the status of MCPWM0_TASK_TIMER1_PERIOD_UP.
0: Not generated
1: Generated
(R/WTC/SS)

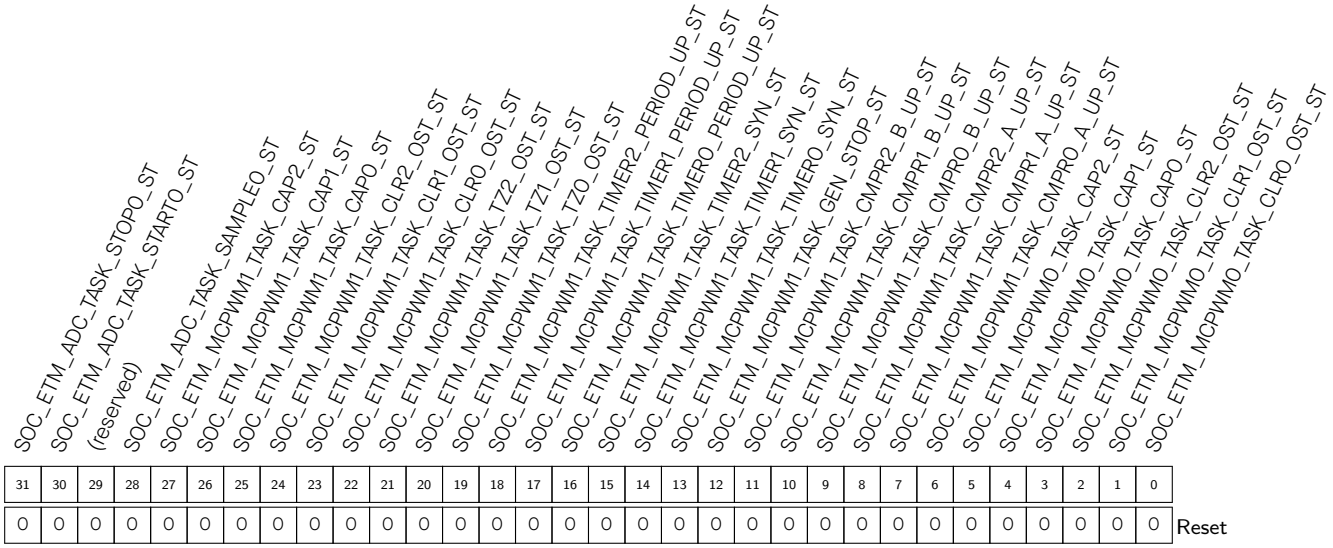
SOC_ETM_MCPWM0_TASK_TIMER2_PERIOD_UP_ST Represents the status of MCPWM0_TASK_TIMER2_PERIOD_UP.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_MCPWM0_TASK_TZ0_OST_ST Represents the status of MCPWM0_TASK_TZ0_OST.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_MCPWM0_TASK_TZ1_OST_ST Represents the status of MCPWM0_TASK_TZ1_OST.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_MCPWM0_TASK_TZ2_OST_ST Represents the status of MCPWM0_TASK_TZ2_OST.
0: Not generated
1: Generated
(R/WTC/SS)

Register 14.15. SOC_ETM_TASK_ST4_REG (0x0208)



SOC_ETM_MCPWM0_TASK_CLR0_OST_ST Represents the status of MCPWM0_TASK_CLR0_OST.

0: Not generated

1: Generated

(R/WTC/SS)

SOC_ETM_MCPWM0_TASK_CLR1_OST_ST Represents the status of MCPWM0_TASK_CLR1_OST.

0: Not generated

1: Generated

(R/WTC/SS)

SOC_ETM_MCPWM0_TASK_CLR2_OST_ST Represents the status of MCPWM0_TASK_CLR2_OST.

0: Not generated

1: Generated

(R/WTC/SS)

SOC_ETM_MCPWM0_TASK_CAP0_ST Represents the status of MCPWM0_TASK_CAP0.

0: Not generated

1: Generated

(R/WTC/SS)

SOC_ETM_MCPWM0_TASK_CAP1_ST Represents the status of MCPWM0_TASK_CAP1.

0: Not generated

1: Generated

(R/WTC/SS)

SOC_ETM_MCPWM0_TASK_CAP2_ST Represents the status of MCPWM0_TASK_CAP2.

0: Not generated

1: Generated

(R/WTC/SS)

Continued on the next page...

Register 14.15. SOC_ETM_TASK_ST4_REG (0x0208)

Continued from the previous page...

SOC_ETM_MCPWM1_TASK_CMPR0_A_UP_ST Represents the status of MCPWM1_TASK_CMPR0_A_UP.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_MCPWM1_TASK_CMPR1_A_UP_ST Represents the status of MCPWM1_TASK_CMPR1_A_UP.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_MCPWM1_TASK_CMPR2_A_UP_ST Represents the status of MCPWM1_TASK_CMPR2_A_UP.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_MCPWM1_TASK_CMPR0_B_UP_ST Represents the status of MCPWM1_TASK_CMPR0_B_UP.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_MCPWM1_TASK_CMPR1_B_UP_ST Represents the status of MCPWM1_TASK_CMPR1_B_UP.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_MCPWM1_TASK_CMPR2_B_UP_ST Represents the status of MCPWM1_TASK_CMPR2_B_UP.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_MCPWM1_TASK_GEN_STOP_ST Represents the status of MCPWM1_TASK_GEN_STOP.
0: Not generated
1: Generated
(R/WTC/SS)

Continued on the next page...

Register 14.15. SOC_ETM_TASK_ST4_REG (0x0208)

Continued from the previous page...

SOC_ETM_MCPWM1_TASK_TIMER0_SYN_ST Represents the status of MCPWM1_TASK_TIMER0_SYN.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_MCPWM1_TASK_TIMER1_SYN_ST Represents the status of MCPWM1_TASK_TIMER1_SYN.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_MCPWM1_TASK_TIMER2_SYN_ST Represents the status of MCPWM1_TASK_TIMER2_SYN.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_MCPWM1_TASK_TIMER0_PERIOD_UP_ST Represents the status of MCPWM1_TASK_TIMER0_PERIOD_UP.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_MCPWM1_TASK_TIMER1_PERIOD_UP_ST Represents the status of MCPWM1_TASK_TIMER1_PERIOD_UP.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_MCPWM1_TASK_TIMER2_PERIOD_UP_ST Represents the status of MCPWM1_TASK_TIMER2_PERIOD_UP.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_MCPWM1_TASK_TZ0_OST_ST Represents the status of MCPWM1_TASK_TZ0_OST.
0: Not generated
1: Generated
(R/WTC/SS)

Continued on the next page...

Register 14.15. SOC_ETM_TASK_ST4_REG (0x0208)

Continued from the previous page...

SOC_ETM_MCPWM1_TASK_TZ1_OST_ST Represents the status of MCPWM1_TASK_TZ1_OST.

0: Not generated

1: Generated

(R/WTC/SS)

SOC_ETM_MCPWM1_TASK_TZ2_OST_ST Represents the status of MCPWM1_TASK_TZ2_OST.

0: Not generated

1: Generated

(R/WTC/SS)

SOC_ETM_MCPWM1_TASK_CLR0_OST_ST Represents the status of MCPWM1_TASK_CLR0_OST.

0: Not generated

1: Generated

(R/WTC/SS)

SOC_ETM_MCPWM1_TASK_CLR1_OST_ST Represents the status of MCPWM1_TASK_CLR1_OST.

0: Not generated

1: Generated

(R/WTC/SS)

SOC_ETM_MCPWM1_TASK_CLR2_OST_ST Represents the status of MCPWM1_TASK_CLR2_OST.

0: Not generated

1: Generated

(R/WTC/SS)

SOC_ETM_MCPWM1_TASK_CAP0_ST Represents the status of MCPWM1_TASK_CAP0.

0: Not generated

1: Generated

(R/WTC/SS)

SOC_ETM_MCPWM1_TASK_CAP1_ST Represents the status of MCPWM1_TASK_CAP1.

0: Not generated

1: Generated

(R/WTC/SS)

Continued on the next page...

Register 14.15. SOC_ETM_TASK_ST4_REG (0x0208)

Continued from the previous page...

SOC_ETM_MCPWM1_TASK_CAP2_ST Represents the status of MCPWM1_TASK_CAP2.

0: Not generated

1: Generated

(R/WTC/SS)

SOC_ETM_ADC_TASK_SAMPLE0_ST Represents the status of ADC_TASK_SAMPLE0.

0: Not generated

1: Generated

(R/WTC/SS)

SOC_ETM_ADC_TASK_START0_ST Represents the status of ADC_TASK_START0.

0: Not generated

1: Generated

(R/WTC/SS)

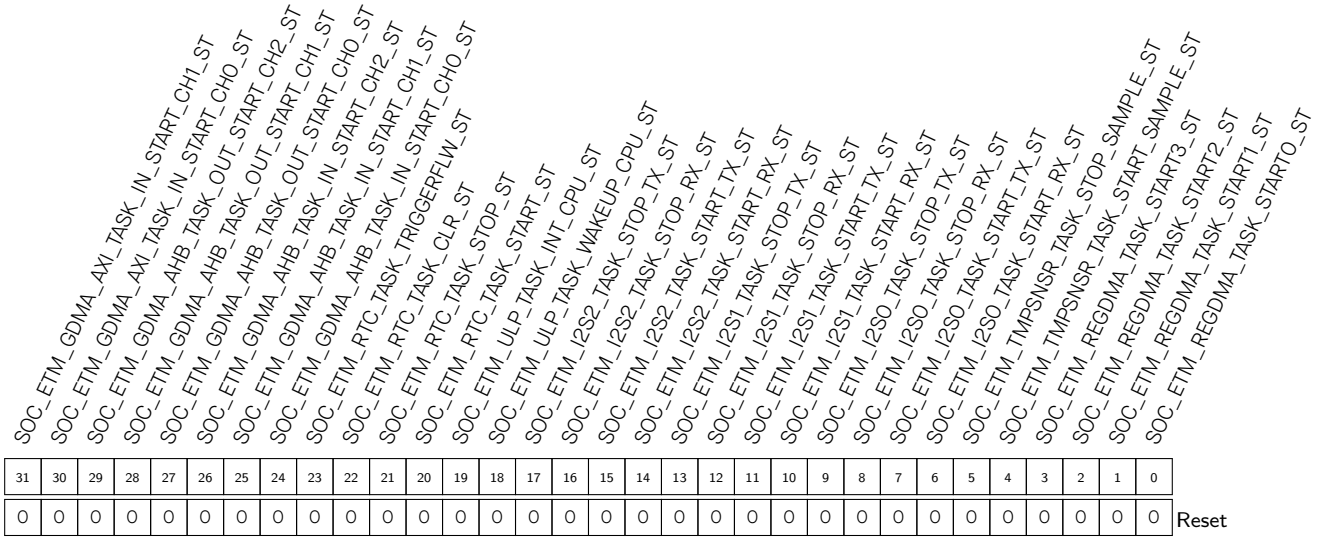
SOC_ETM_ADC_TASK_STOP0_ST Represents the status of ADC_TASK_STOP0.

0: Not generated

1: Generated

(R/WTC/SS)

Register 14.16. SOC_ETM_TASK_ST5_REG (0x0210)



SOC_ETM_REGDMA_TASK_START0_ST Represents the status of REGDMA_TASK_START0.

0: Not generated

1: Generated

(R/WTC/SS)

SOC_ETM_REGDMA_TASK_START1_ST Represents the status of REGDMA_TASK_START1.

0: Not generated

1: Generated

(R/WTC/SS)

SOC_ETM_REGDMA_TASK_START2_ST Represents the status of REGDMA_TASK_START2.

0: Not generated

1: Generated

(R/WTC/SS)

SOC_ETM_REGDMA_TASK_START3_ST Represents the status of REGDMA_TASK_START3.

0: Not generated

1: Generated

(R/WTC/SS)

SOC_ETM_TMPSNSR_TASK_START_SAMPLE_ST Represents the status of TMPSNSR_TASK_START_SAMPLE.

0: Not generated

1: Generated

(R/WTC/SS)

SOC_ETM_TMPSNSR_TASK_STOP_SAMPLE_ST Represents the status of TMPSNSR_TASK_STOP_SAMPLE.

0: Not generated

1: Generated

(R/WTC/SS)

Continued on the next page...

Register 14.16. SOC_ETM_TASK_ST5_REG (0x0210)

Continued from the previous page...

SOC_ETM_I2S0_TASK_START_RX_ST Represents the status of I2S0_TASK_START_RX.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_I2S0_TASK_START_TX_ST Represents the status of I2S0_TASK_START_TX.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_I2S0_TASK_STOP_RX_ST Represents the status of I2S0_TASK_STOP_RX.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_I2S0_TASK_STOP_TX_ST Represents the status of I2S0_TASK_STOP_TX.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_I2S1_TASK_START_RX_ST Represents the status of I2S1_TASK_START_RX.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_I2S1_TASK_START_TX_ST Represents the status of I2S1_TASK_START_TX.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_I2S1_TASK_STOP_RX_ST Represents the status of I2S1_TASK_STOP_RX.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_I2S1_TASK_STOP_TX_ST Represents the status of I2S1_TASK_STOP_TX.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_I2S2_TASK_START_RX_ST Represents the status of I2S2_TASK_START_RX.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_I2S2_TASK_START_TX_ST Represents the status of I2S2_TASK_START_TX.
0: Not generated
1: Generated
(R/WTC/SS)

Continued on the next page...

Register 14.16. SOC_ETM_TASK_ST5_REG (0x0210)

Continued from the previous page...

SOC_ETM_I2S2_TASK_STOP_RX_ST Represents the status of I2S2_TASK_STOP_RX.

0: Not generated

1: Generated

(R/WTC/SS)

SOC_ETM_I2S2_TASK_STOP_TX_ST Represents the status of I2S2_TASK_STOP_TX.

0: Not generated

1: Generated

(R/WTC/SS)

SOC_ETM_ULP_TASK_WAKEUP_CPU_ST Represents the status of ULP_TASK_WAKEUP_CPU.

0: Not generated

1: Generated

(R/WTC/SS)

SOC_ETM_ULP_TASK_INT_CPU_ST Represents the status of ULP_TASK_INT_CPU.

0: Not generated

1: Generated

(R/WTC/SS)

SOC_ETM_RTC_TASK_START_ST Represents the status of RTC_TASK_START.

0: Not generated

1: Generated

(R/WTC/SS)

SOC_ETM_RTC_TASK_STOP_ST Represents the status of RTC_TASK_STOP.

0: Not generated

1: Generated

(R/WTC/SS)

SOC_ETM_RTC_TASK_CLR_ST Represents the status of RTC_TASK_CLR.

0: Not generated

1: Generated

(R/WTC/SS)

SOC_ETM_RTC_TASK_TRIGGERFLW_ST Represents the status of RTC_TASK_TRIGGERFLW.

0: Not generated

1: Generated

(R/WTC/SS)

Continued on the next page...

Register 14.16. SOC_ETM_TASK_ST5_REG (0x0210)

Continued from the previous page...

SOC_ETM_GDMA_AHB_TASK_IN_START_CH0_ST Represents the status of GDMA_AHB_TASK_IN_START_CH0.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_GDMA_AHB_TASK_IN_START_CH1_ST Represents the status of GDMA_AHB_TASK_IN_START_CH1.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_GDMA_AHB_TASK_IN_START_CH2_ST Represents the status of GDMA_AHB_TASK_IN_START_CH2.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_GDMA_AHB_TASK_OUT_START_CH0_ST Represents the status of GDMA_AHB_TASK_OUT_START_CH0.
0: Not generated
1: Generated
(R/WTC/SS)

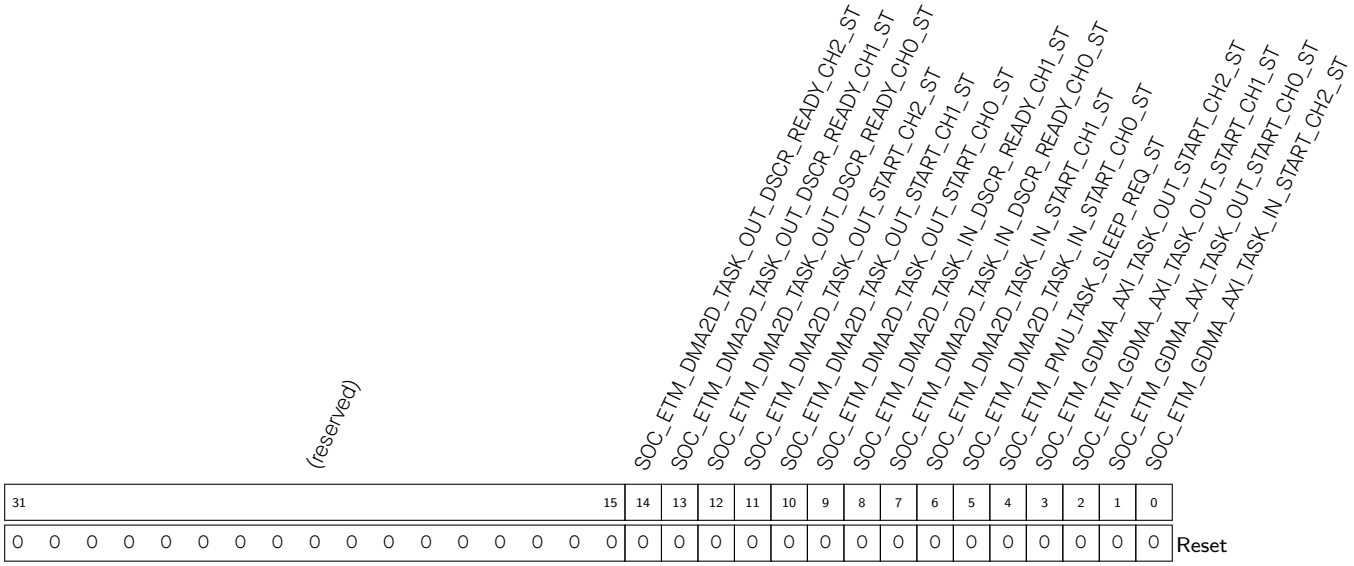
SOC_ETM_GDMA_AHB_TASK_OUT_START_CH1_ST Represents the status of GDMA_AHB_TASK_OUT_START_CH1.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_GDMA_AHB_TASK_OUT_START_CH2_ST Represents the status of GDMA_AHB_TASK_OUT_START_CH2.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_GDMA_AXI_TASK_IN_START_CH0_ST Represents the status of GDMA_AXI_TASK_IN_START_CH0.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_GDMA_AXI_TASK_IN_START_CH1_ST Represents the status of GDMA_AXI_TASK_IN_START_CH1.
0: Not generated
1: Generated
(R/WTC/SS)

Register 14.17. SOC_ETM_TASK_ST6_REG (0x0218)



SOC_ETM_GDMA_AXI_TASK_IN_START_CH2_ST Represents the status of GDMA_AXI_TASK_IN_START_CH2.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_GDMA_AXI_TASK_OUT_START_CH0_ST Represents the status of GDMA_AXI_TASK_OUT_START_CH0.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_GDMA_AXI_TASK_OUT_START_CH1_ST Represents the status of GDMA_AXI_TASK_OUT_START_CH1.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_GDMA_AXI_TASK_OUT_START_CH2_ST Represents the status of GDMA_AXI_TASK_OUT_START_CH2.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_PMU_TASK_SLEEP_REQ_ST Represents the status of PMU_TASK_SLEEP_REQ.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_DMA2D_TASK_IN_START_CH0_ST Represents the status of DMA2D_TASK_IN_START_CH0.
0: Not generated
1: Generated
(R/WTC/SS)

Continued on the next page...

Register 14.17. SOC_ETM_TASK_ST6_REG (0x0218)

Continued from the previous page...

SOC_ETM_DMA2D_TASK_IN_START_CH1_ST Represents the status of DMA2D_TASK_IN_START_CH1.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_DMA2D_TASK_IN_DSCR_READY_CH0_ST Represents the status of DMA2D_TASK_IN_DSCR_READY_CH0.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_DMA2D_TASK_IN_DSCR_READY_CH1_ST Represents the status of DMA2D_TASK_IN_DSCR_READY_CH1.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_DMA2D_TASK_OUT_START_CH0_ST Represents the status of DMA2D_TASK_OUT_START_CH0.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_DMA2D_TASK_OUT_START_CH1_ST Represents the status of DMA2D_TASK_OUT_START_CH1.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_DMA2D_TASK_OUT_START_CH2_ST Represents the status of DMA2D_TASK_OUT_START_CH2.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_DMA2D_TASK_OUT_DSCR_READY_CH0_ST Represents the status of DMA2D_TASK_OUT_DSCR_READY_CH0.
0: Not generated
1: Generated
(R/WTC/SS)

Continued on the next page...

Register 14.17. SOC_ETM_TASK_ST6_REG (0x0218)

Continued from the previous page...

SOC_ETM_DMA2D_TASK_OUT_DSCR_READY_CH1_ST Represents the status of DMA2D_TASK_OUT_DSCR_READY_CH1.

0: Not generated

1: Generated

(R/WTC/SS)

SOC_ETM_DMA2D_TASK_OUT_DSCR_READY_CH2_ST Represents the status of DMA2D_TASK_OUT_DSCR_READY_CH2.

0: Not generated

1: Generated

(R/WTC/SS)

Register 14.18. SOC_ETM_CH_ENA_AD0_SET_REG (0x0004)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
SOC_ETM_CH_ENABLE31	SOC_ETM_CH_ENABLE30	SOC_ETM_CH_ENABLE29	SOC_ETM_CH_ENABLE28	SOC_ETM_CH_ENABLE27	SOC_ETM_CH_ENABLE26	SOC_ETM_CH_ENABLE25	SOC_ETM_CH_ENABLE24	SOC_ETM_CH_ENABLE23	SOC_ETM_CH_ENABLE22	SOC_ETM_CH_ENABLE21	SOC_ETM_CH_ENABLE20	SOC_ETM_CH_ENABLE19	SOC_ETM_CH_ENABLE18	SOC_ETM_CH_ENABLE17	SOC_ETM_CH_ENABLE16	SOC_ETM_CH_ENABLE15	SOC_ETM_CH_ENABLE14	SOC_ETM_CH_ENABLE13	SOC_ETM_CH_ENABLE12	SOC_ETM_CH_ENABLE11	SOC_ETM_CH_ENABLE10	SOC_ETM_CH_ENABLE9	SOC_ETM_CH_ENABLE8	SOC_ETM_CH_ENABLE7	SOC_ETM_CH_ENABLE6	SOC_ETM_CH_ENABLE5	SOC_ETM_CH_ENABLE4	SOC_ETM_CH_ENABLE3	SOC_ETM_CH_ENABLE2	SOC_ETM_CH_ENABLE1	SOC_ETM_CH_ENABLE0	
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Reset

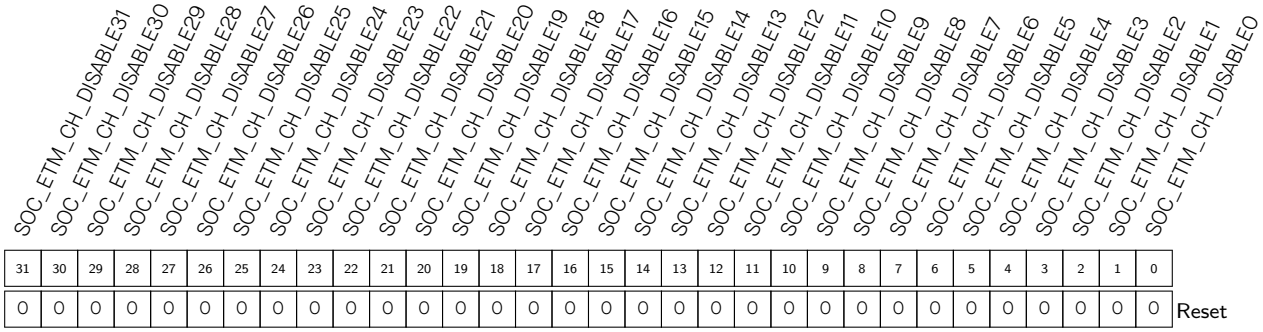
SOC_ETM_CH_ENABLE*n* (*n*: 0-31) Configures whether to enable channel*n*.

0: Invalid. No effect

1: Enable

(WT)

Register 14.19. SOC_ETM_CH_ENA_AD0_CLR_REG (0x0008)



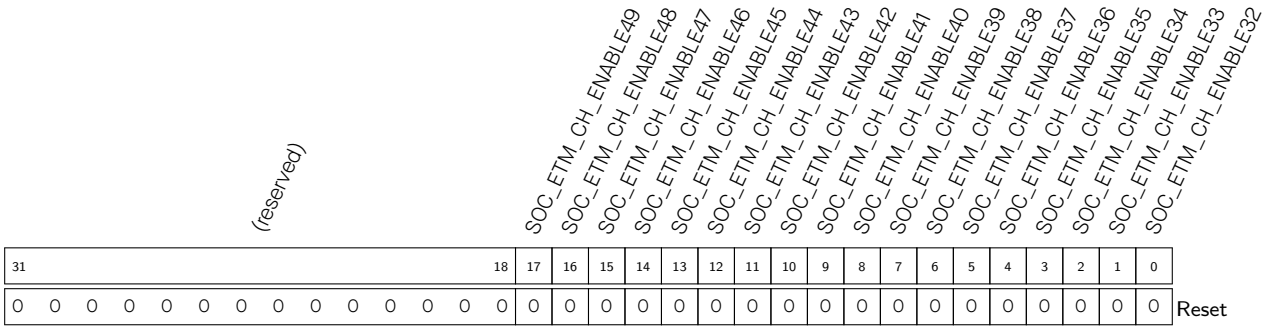
SOC_ETM_CH_DISABLE*n* (*n*: 0-31) Configures whether to disable channel 0.

0: Invalid. No effect

1: Disable

(WT)

Register 14.20. SOC_ETM_CH_ENA_AD1_SET_REG (0x0010)



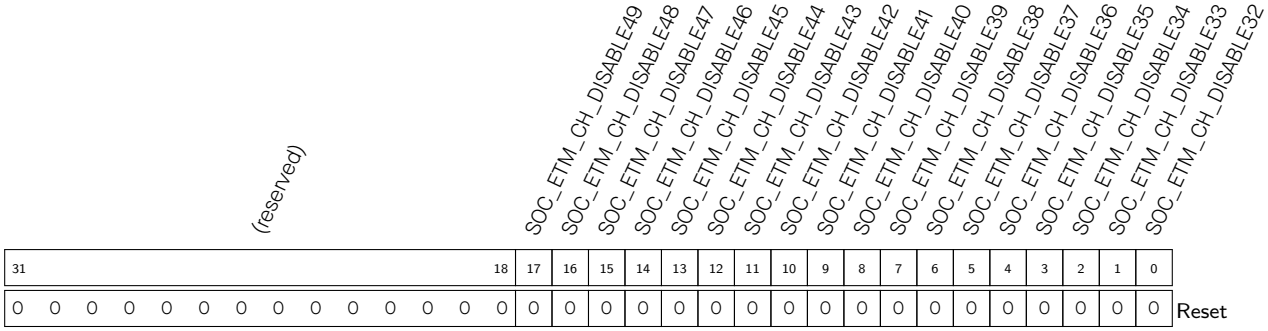
SOC_ETM_CH_ENABLE*n* (*n*: 32-49) Configures whether to enable channel*n*.

0: Invalid. No effect

1: Enable

(WT)

Register 14.21. SOC_ETM_CH_ENA_AD1_CLR_REG (0x0014)



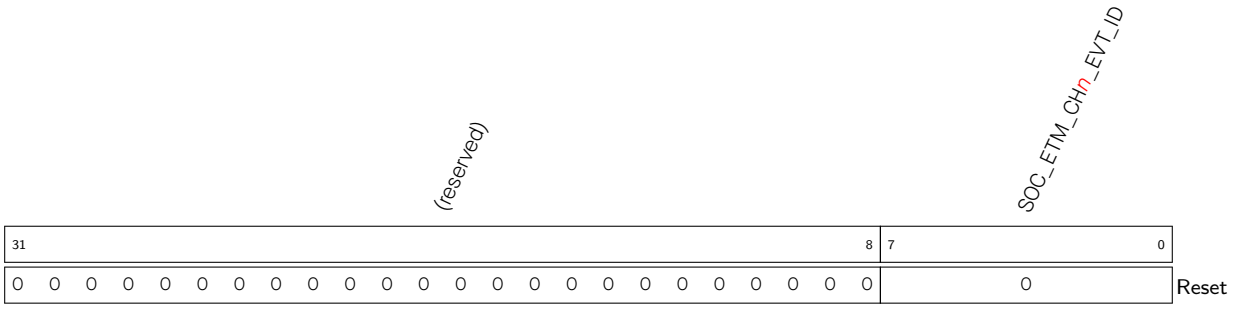
SOC_ETM_CH_DISABLE*n* (*n*: 32-49) Configures whether to disable channel*n*.

0: Invalid. No effect

1: Disable

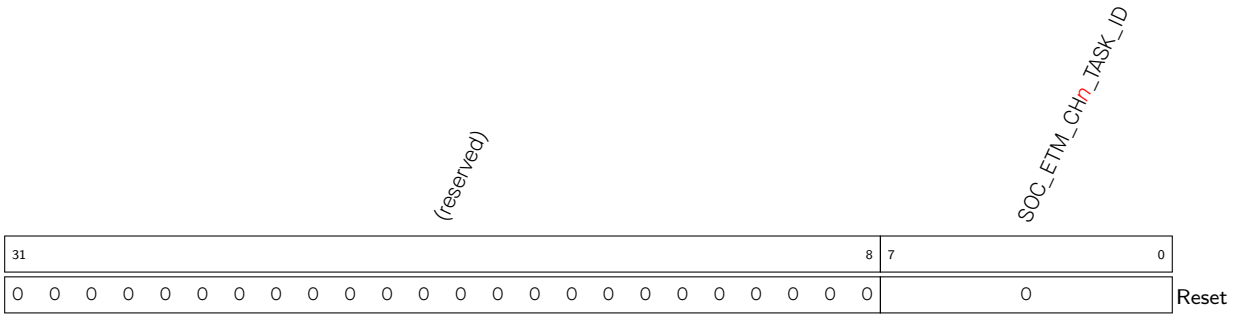
(WT)

Register 14.22. SOC_ETM_CH*n*_EVT_ID_REG (*n*: 0-49) (0x0018+0x8**n*)



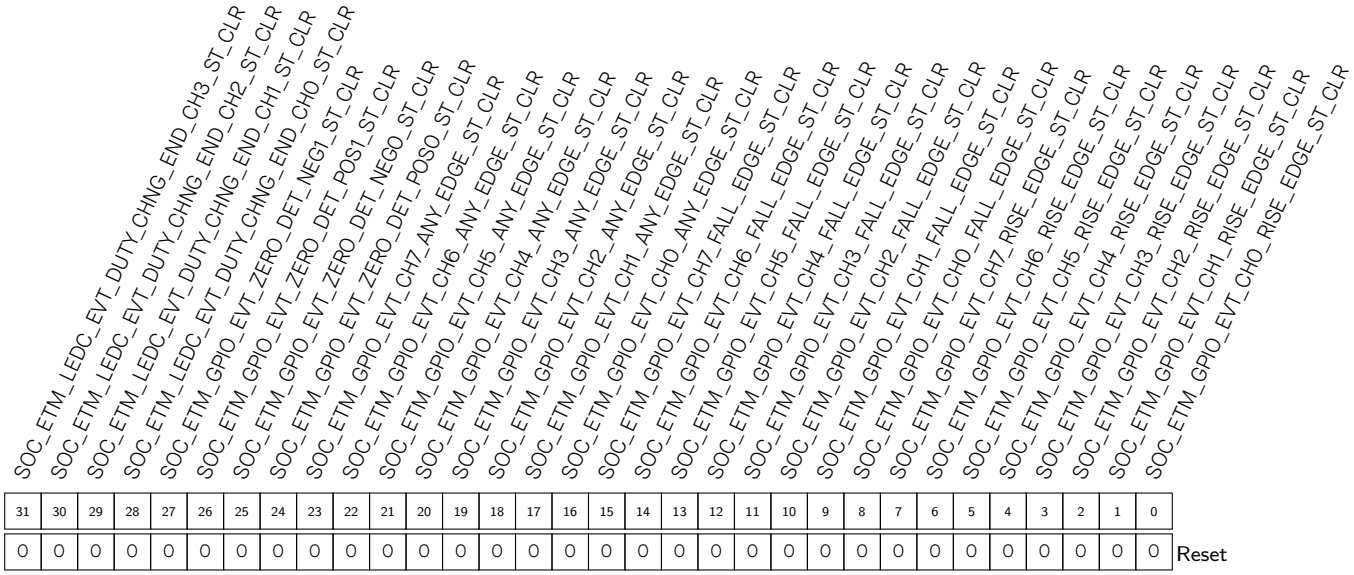
SOC_ETM_CH*n*_EVT_ID (*n*: 0-49) Configures the event ID of channel*n*. See Table 14.3-1. (R/W)

Register 14.23. SOC_ETM_CH*n*_TASK_ID_REG (*n*: 0-49) (0x001C+0x8**n*)



SOC_ETM_CH*n*_TASK_ID (*n*: 0-49) Configures the task ID of channel*n*. See Table 14.3-2. (R/W)

Register 14.24. SOC_ETM_EVT_ST0_CLR_REG (0x01AC)



SOC_ETM_GPIO_EVT_CH0_RISE_EDGE_ST_CLR Configures whether to clear GPIO_EVT_CH0_RISE_EDGE.

0: Invalid. No effect

1: Clear

(WT)

SOC_ETM_GPIO_EVT_CH1_RISE_EDGE_ST_CLR Configures whether to clear GPIO_EVT_CH1_RISE_EDGE.

0: Invalid. No effect

1: Clear

(WT)

SOC_ETM_GPIO_EVT_CH2_RISE_EDGE_ST_CLR Configures whether to clear GPIO_EVT_CH2_RISE_EDGE.

0: Invalid. No effect

1: Clear

(WT)

SOC_ETM_GPIO_EVT_CH3_RISE_EDGE_ST_CLR Configures whether to clear GPIO_EVT_CH3_RISE_EDGE.

0: Invalid. No effect

1: Clear

(WT)

SOC_ETM_GPIO_EVT_CH4_RISE_EDGE_ST_CLR Configures whether to clear GPIO_EVT_CH4_RISE_EDGE.

0: Invalid. No effect

1: Clear

(WT)

Continued on the next page...

Register 14.24. SOC_ETM_EVT_ST0_CLR_REG (0x01AC)

Continued from the previous page...

SOC_ETM_GPIO_EVT_CH5_RISE_EDGE_ST_CLR Configures whether to clear GPIO_EVT_CH5_RISE_EDGE.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_GPIO_EVT_CH6_RISE_EDGE_ST_CLR Configures whether to clear GPIO_EVT_CH6_RISE_EDGE.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_GPIO_EVT_CH7_RISE_EDGE_ST_CLR Configures whether to clear GPIO_EVT_CH7_RISE_EDGE.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_GPIO_EVT_CH0_FALL_EDGE_ST_CLR Configures whether to clear GPIO_EVT_CH0_FALL_EDGE.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_GPIO_EVT_CH1_FALL_EDGE_ST_CLR Configures whether to clear GPIO_EVT_CH1_FALL_EDGE.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_GPIO_EVT_CH2_FALL_EDGE_ST_CLR Configures whether to clear GPIO_EVT_CH2_FALL_EDGE.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_GPIO_EVT_CH3_FALL_EDGE_ST_CLR Configures whether to clear GPIO_EVT_CH3_FALL_EDGE.
0: Invalid. No effect
1: Clear
(WT)

Continued on the next page...

Register 14.24. SOC_ETM_EVT_ST0_CLR_REG (0x01AC)

Continued from the previous page...

SOC_ETM_GPIO_EVT_CH4_FALL_EDGE_ST_CLR Configures whether to clear GPIO_EVT_CH4_FALL_EDGE.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_GPIO_EVT_CH5_FALL_EDGE_ST_CLR Configures whether to clear GPIO_EVT_CH5_FALL_EDGE.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_GPIO_EVT_CH6_FALL_EDGE_ST_CLR Configures whether to clear GPIO_EVT_CH6_FALL_EDGE.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_GPIO_EVT_CH7_FALL_EDGE_ST_CLR Configures whether to clear GPIO_EVT_CH7_FALL_EDGE.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_GPIO_EVT_CH0_ANY_EDGE_ST_CLR Configures whether to clear GPIO_EVT_CH0_ANY_EDGE.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_GPIO_EVT_CH1_ANY_EDGE_ST_CLR Configures whether to clear GPIO_EVT_CH1_ANY_EDGE.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_GPIO_EVT_CH2_ANY_EDGE_ST_CLR Configures whether to clear GPIO_EVT_CH2_ANY_EDGE.
0: Invalid. No effect
1: Clear
(WT)

Continued on the next page...

Register 14.24. SOC_ETM_EVT_ST0_CLR_REG (0x01AC)

Continued from the previous page...

SOC_ETM_GPIO_EVT_CH3_ANY_EDGE_ST_CLR Configures whether to clear GPIO_EVT_CH3_ANY_EDGE.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_GPIO_EVT_CH4_ANY_EDGE_ST_CLR Configures whether to clear GPIO_EVT_CH4_ANY_EDGE.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_GPIO_EVT_CH5_ANY_EDGE_ST_CLR Configures whether to clear GPIO_EVT_CH5_ANY_EDGE.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_GPIO_EVT_CH6_ANY_EDGE_ST_CLR Configures whether to clear GPIO_EVT_CH6_ANY_EDGE.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_GPIO_EVT_CH7_ANY_EDGE_ST_CLR Configures whether to clear GPIO_EVT_CH7_ANY_EDGE.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_GPIO_EVT_ZERO_DET_POS0_ST_CLR Configures whether to clear GPIO_EVT_ZERO_DET_POS0.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_GPIO_EVT_ZERO_DET_NEG0_ST_CLR Configures whether to clear GPIO_EVT_ZERO_DET_NEG0.
0: Invalid. No effect
1: Clear
(WT)

Continued on the next page...

Register 14.24. SOC_ETM_EVT_ST0_CLR_REG (0x01AC)

Continued from the previous page...

SOC_ETM_GPIO_EVT_ZERO_DET_POS1_ST_CLR Configures whether to clear GPIO_EVT_ZERO_DET_POS1.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_GPIO_EVT_ZERO_DET_NEG1_ST_CLR Configures whether to clear GPIO_EVT_ZERO_DET_NEG1.
0: Invalid. No effect
1: Clear
(WT)

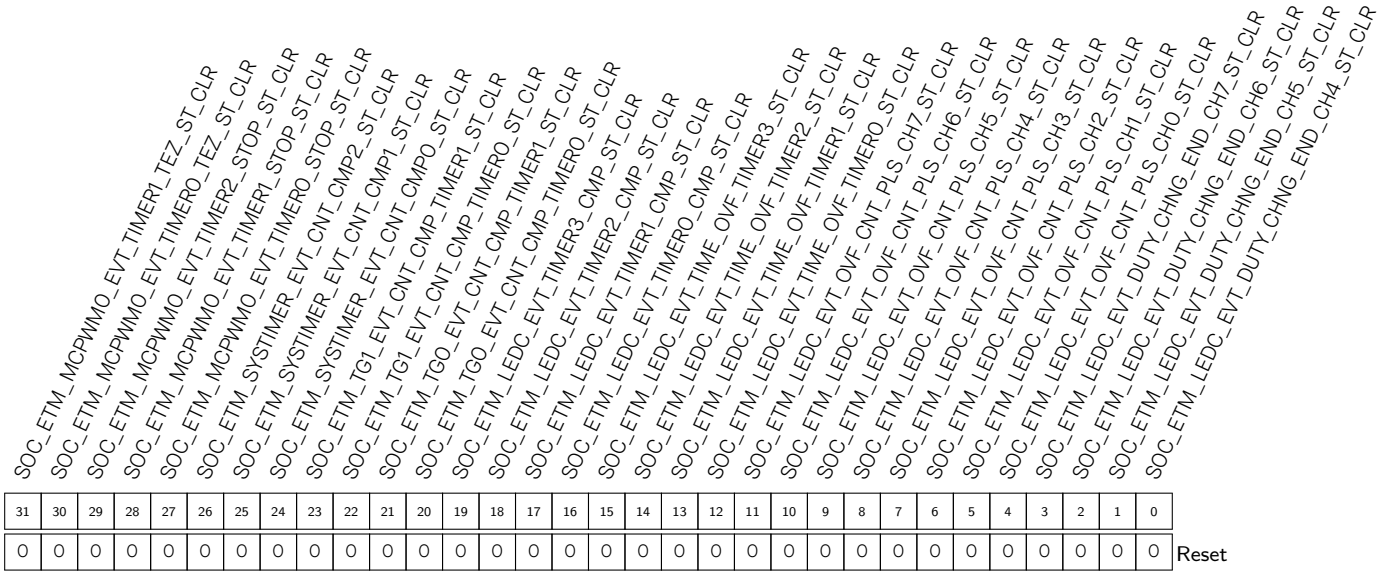
SOC_ETM_LEDC_EVT_DUTY_CHNG_END_CH0_ST_CLR Configures whether to clear LEDC_EVT_DUTY_CHNG_END_CH0.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_LEDC_EVT_DUTY_CHNG_END_CH1_ST_CLR Configures whether to clear LEDC_EVT_DUTY_CHNG_END_CH1.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_LEDC_EVT_DUTY_CHNG_END_CH2_ST_CLR Configures whether to clear LEDC_EVT_DUTY_CHNG_END_CH2.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_LEDC_EVT_DUTY_CHNG_END_CH3_ST_CLR Configures whether to clear LEDC_EVT_DUTY_CHNG_END_CH3.
0: Invalid. No effect
1: Clear
(WT)

Register 14.25. SOC_ETM_EVT_ST1_CLR_REG (0x01B4)



SOC_ETM_LEDC_EVT_DUTY_CHNG_END_CH4_ST_CLR Configures whether to clear LEDC_EVT_DUTY_CHNG_END_CH4.

0: Invalid. No effect

1: Clear

(WT)

SOC_ETM_LEDC_EVT_DUTY_CHNG_END_CH5_ST_CLR Configures whether to clear LEDC_EVT_DUTY_CHNG_END_CH5.

0: Invalid. No effect

1: Clear

(WT)

SOC_ETM_LEDC_EVT_DUTY_CHNG_END_CH6_ST_CLR Configures whether to clear LEDC_EVT_DUTY_CHNG_END_CH6.

0: Invalid. No effect

1: Clear

(WT)

SOC_ETM_LEDC_EVT_DUTY_CHNG_END_CH7_ST_CLR Configures whether to clear LEDC_EVT_DUTY_CHNG_END_CH7.

0: Invalid. No effect

1: Clear

(WT)

SOC_ETM_LEDC_EVT_OVF_CNT_PLS_CH0_ST_CLR Configures whether to clear LEDC_EVT_OVF_CNT_PLS_CH0.

0: Invalid. No effect

1: Clear

(WT)

Continued on the next page...

Register 14.25. SOC_ETM_EVT_ST1_CLR_REG (0x01B4)

Continued from the previous page...

SOC_ETM_LEDC_EVT_OVF_CNT_PLS_CH1_ST_CLR Configures whether to clear LEDC_EVT_OVF_CNT_PLS_CH1.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_LEDC_EVT_OVF_CNT_PLS_CH2_ST_CLR Configures whether to clear LEDC_EVT_OVF_CNT_PLS_CH2.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_LEDC_EVT_OVF_CNT_PLS_CH3_ST_CLR Configures whether to clear LEDC_EVT_OVF_CNT_PLS_CH3.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_LEDC_EVT_OVF_CNT_PLS_CH4_ST_CLR Configures whether to clear LEDC_EVT_OVF_CNT_PLS_CH4.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_LEDC_EVT_OVF_CNT_PLS_CH5_ST_CLR Configures whether to clear LEDC_EVT_OVF_CNT_PLS_CH5.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_LEDC_EVT_OVF_CNT_PLS_CH6_ST_CLR Configures whether to clear LEDC_EVT_OVF_CNT_PLS_CH6.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_LEDC_EVT_OVF_CNT_PLS_CH7_ST_CLR Configures whether to clear LEDC_EVT_OVF_CNT_PLS_CH7.
0: Invalid. No effect
1: Clear
(WT)

Continued on the next page...

Register 14.25. SOC_ETM_EVT_ST1_CLR_REG (0x01B4)

Continued from the previous page...

SOC_ETM_LEDC_EVT_TIME_OVF_TIMER0_ST_CLR Configures whether to clear LEDC_EVT_TIME_OVF_TIMER0.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_LEDC_EVT_TIME_OVF_TIMER1_ST_CLR Configures whether to clear LEDC_EVT_TIME_OVF_TIMER1.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_LEDC_EVT_TIME_OVF_TIMER2_ST_CLR Configures whether to clear LEDC_EVT_TIME_OVF_TIMER2.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_LEDC_EVT_TIME_OVF_TIMER3_ST_CLR Configures whether to clear LEDC_EVT_TIME_OVF_TIMER3.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_LEDC_EVT_TIMER0_CMP_ST_CLR Configures whether to clear LEDC_EVT_TIMER0_CMP.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_LEDC_EVT_TIMER1_CMP_ST_CLR Configures whether to clear LEDC_EVT_TIMER1_CMP.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_LEDC_EVT_TIMER2_CMP_ST_CLR Configures whether to clear LEDC_EVT_TIMER2_CMP.
0: Invalid. No effect
1: Clear
(WT)

Continued on the next page...

Register 14.25. SOC_ETM_EVT_ST1_CLR_REG (0x01B4)

Continued from the previous page...

SOC_ETM_LEDC_EVT_TIMER3_CMP_ST_CLR Configures whether to clear LEDC_EVT_TIMER3_CMP.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_TG0_EVT_CNT_CMP_TIMER0_ST_CLR Configures whether to clear TG0_EVT_CNT_CMP_TIMER0.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_TG0_EVT_CNT_CMP_TIMER1_ST_CLR Configures whether to clear TG0_EVT_CNT_CMP_TIMER1.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_TG1_EVT_CNT_CMP_TIMER0_ST_CLR Configures whether to clear TG1_EVT_CNT_CMP_TIMER0.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_TG1_EVT_CNT_CMP_TIMER1_ST_CLR Configures whether to clear TG1_EVT_CNT_CMP_TIMER1.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_SYSTIMER_EVT_CNT_CMP0_ST_CLR Configures whether to clear SYSTIMER_EVT_CNT_CMP0.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_SYSTIMER_EVT_CNT_CMP1_ST_CLR Configures whether to clear SYSTIMER_EVT_CNT_CMP1.
0: Invalid. No effect
1: Clear
(WT)

Continued on the next page...

Register 14.25. SOC_ETM_EVT_ST1_CLR_REG (0x01B4)

Continued from the previous page...

SOC_ETM_SYSTIMER_EVT_CNT_CMP2_ST_CLR Configures whether to clear SYSTIMER_EVT_CNT_CMP2.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_MCPWM0_EVT_TIMER0_STOP_ST_CLR Configures whether to clear MCPWM0_EVT_TIMER0_STOP.
0: Invalid. No effect
1: Clear
(WT)

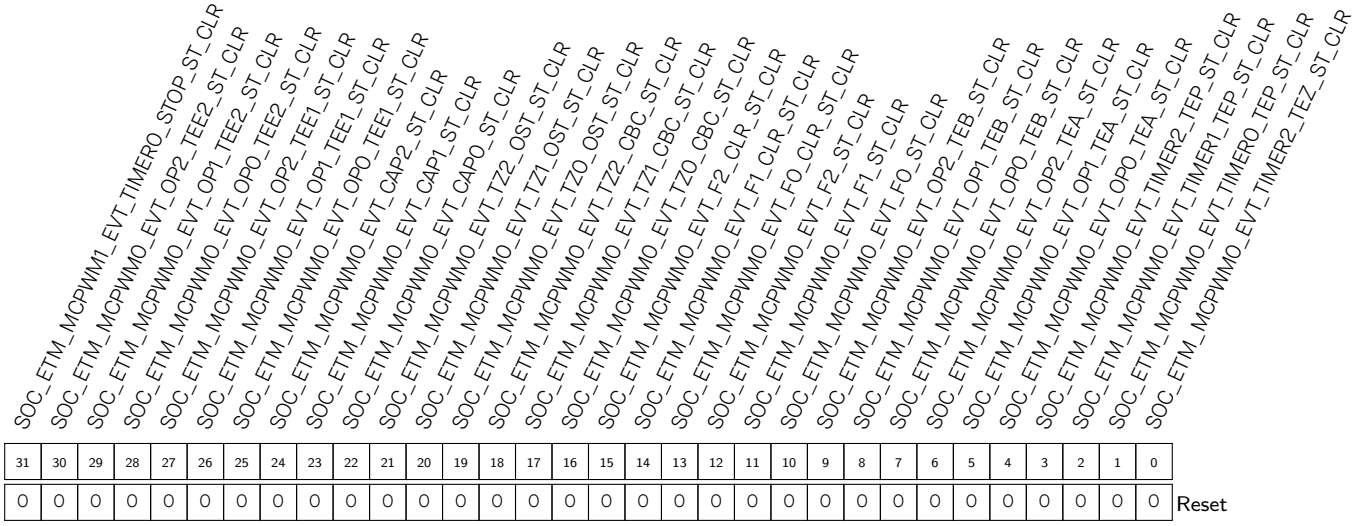
SOC_ETM_MCPWM0_EVT_TIMER1_STOP_ST_CLR Configures whether to clear MCPWM0_EVT_TIMER1_STOP.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_MCPWM0_EVT_TIMER2_STOP_ST_CLR Configures whether to clear MCPWM0_EVT_TIMER2_STOP.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_MCPWM0_EVT_TIMER0_TEZ_ST_CLR Configures whether to clear MCPWM0_EVT_TIMER0_TEZ.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_MCPWM0_EVT_TIMER1_TEZ_ST_CLR Configures whether to clear MCPWM0_EVT_TIMER1_TEZ.
0: Invalid. No effect
1: Clear
(WT)

Register 14.26. SOC_ETM_EVT_ST2_CLR_REG (0x01BC)



SOC_ETM_MCPWM0_EVT_TIMER2_TEZ_ST_CLR Configures whether to clear MCPWM0_EVT_TIMER2_TEZ.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_MCPWM0_EVT_TIMER0_TEP_ST_CLR Configures whether to clear MCPWM0_EVT_TIMER0_TEP.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_MCPWM0_EVT_TIMER1_TEP_ST_CLR Configures whether to clear MCPWM0_EVT_TIMER1_TEP.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_MCPWM0_EVT_TIMER2_TEP_ST_CLR Configures whether to clear MCPWM0_EVT_TIMER2_TEP.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_MCPWM0_EVT_OP0_TEA_ST_CLR Configures whether to clear MCPWM0_EVT_OP0_TEA.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_MCPWM0_EVT_OP1_TEA_ST_CLR Configures whether to clear MCPWM0_EVT_OP1_TEA.
0: Invalid. No effect
1: Clear
(WT)

Continued on the next page...

Register 14.26. SOC_ETM_EVT_ST2_CLR_REG (0x01BC)

Continued from the previous page...

SOC_ETM_MCPWM0_EVT_OP2_TEA_ST_CLR Configures whether to clear MCPWM0_EVT_OP2_TEA.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_MCPWM0_EVT_OP0_TEB_ST_CLR Configures whether to clear MCPWM0_EVT_OP0_TEB.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_MCPWM0_EVT_OP1_TEB_ST_CLR Configures whether to clear MCPWM0_EVT_OP1_TEB.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_MCPWM0_EVT_OP2_TEB_ST_CLR Configures whether to clear MCPWM0_EVT_OP2_TEB.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_MCPWM0_EVT_F0_ST_CLR Configures whether to clear MCPWM0_EVT_F0.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_MCPWM0_EVT_F1_ST_CLR Configures whether to clear MCPWM0_EVT_F1.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_MCPWM0_EVT_F2_ST_CLR Configures whether to clear MCPWM0_EVT_F2.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_MCPWM0_EVT_F0_CLR_ST_CLR Configures whether to clear MCPWM0_EVT_F0_CLR.
0: Invalid. No effect
1: Clear
(WT)

Continued on the next page...

Register 14.26. SOC_ETM_EVT_ST2_CLR_REG (0x01BC)

Continued from the previous page...

SOC_ETM_MCPWM0_EVT_F1_CLR_ST_CLR Configures whether to clear MCPWM0_EVT_F1_CLR.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_MCPWM0_EVT_F2_CLR_ST_CLR Configures whether to clear MCPWM0_EVT_F2_CLR.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_MCPWM0_EVT_TZ0_CBC_ST_CLR Configures whether to clear MCPWM0_EVT_TZ0_CBC.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_MCPWM0_EVT_TZ1_CBC_ST_CLR Configures whether to clear MCPWM0_EVT_TZ1_CBC.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_MCPWM0_EVT_TZ2_CBC_ST_CLR Configures whether to clear MCPWM0_EVT_TZ2_CBC.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_MCPWM0_EVT_TZ0_OST_ST_CLR Configures whether to clear MCPWM0_EVT_TZ0_OST.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_MCPWM0_EVT_TZ1_OST_ST_CLR Configures whether to clear MCPWM0_EVT_TZ1_OST.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_MCPWM0_EVT_TZ2_OST_ST_CLR Configures whether to clear MCPWM0_EVT_TZ2_OST.
0: Invalid. No effect
1: Clear
(WT)

Continued on the next page...

Register 14.26. SOC_ETM_EVT_ST2_CLR_REG (0x01BC)

Continued from the previous page...

SOC_ETM_MCPWM0_EVT_CAP0_ST_CLR Configures whether to clear MCPWM0_EVT_CAP0.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_MCPWM0_EVT_CAP1_ST_CLR Configures whether to clear MCPWM0_EVT_CAP1.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_MCPWM0_EVT_CAP2_ST_CLR Configures whether to clear MCPWM0_EVT_CAP2.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_MCPWM0_EVT_OP0_TEE1_ST_CLR Configures whether to clear MCPWM0_EVT_OP0_TEE1.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_MCPWM0_EVT_OP1_TEE1_ST_CLR Configures whether to clear MCPWM0_EVT_OP1_TEE1.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_MCPWM0_EVT_OP2_TEE1_ST_CLR Configures whether to clear MCPWM0_EVT_OP2_TEE1.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_MCPWM0_EVT_OP0_TEE2_ST_CLR Configures whether to clear MCPWM0_EVT_OP0_TEE2.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_MCPWM0_EVT_OP1_TEE2_ST_CLR Configures whether to clear MCPWM0_EVT_OP1_TEE2.
0: Invalid. No effect
1: Clear
(WT)

Continued on the next page...

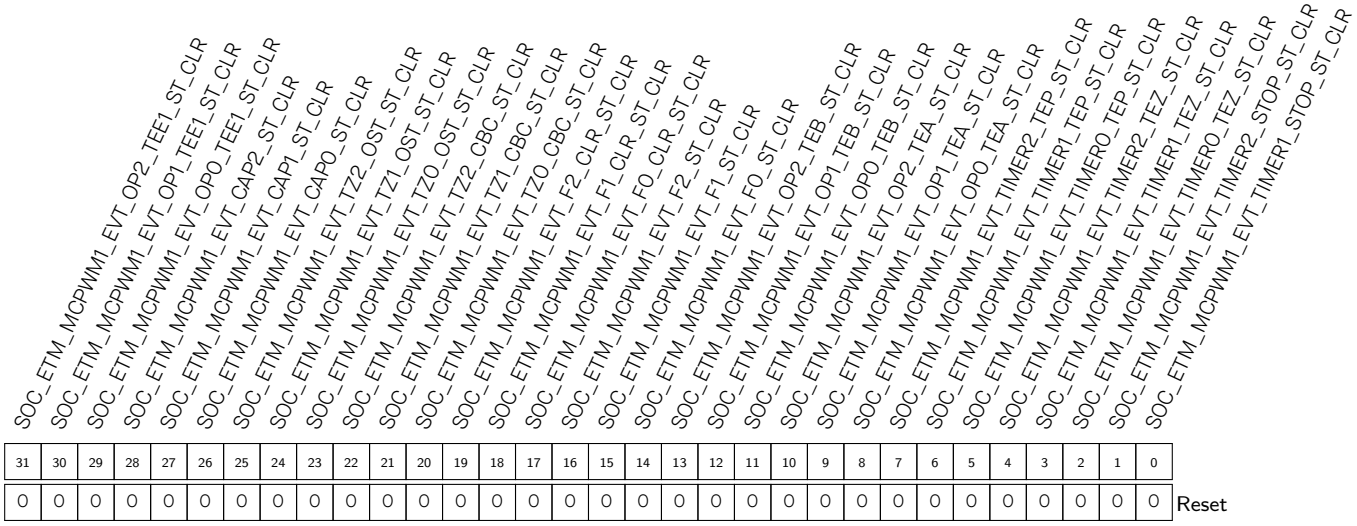
Register 14.26. SOC_ETM_EVT_ST2_CLR_REG (0x01BC)

Continued from the previous page...

SOC_ETM_MCPWM0_EVT_OP2_TEE2_ST_CLR Configures whether to clear MCPWM0_EVT_OP2_TEE2.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_MCPWM1_EVT_TIMER0_STOP_ST_CLR Configures whether to clear MCPWM1_EVT_TIMER0_STOP.
0: Invalid. No effect
1: Clear
(WT)

Register 14.27. SOC_ETM_EVT_ST3_CLR_REG (0x01C4)



SOC_ETM_MCPWM1_EVT_TIMER1_STOP_ST_CLR Configures whether to clear MCPWM1_EVT_TIMER1_STOP.

0: Invalid. No effect

1: Clear

(WT)

SOC_ETM_MCPWM1_EVT_TIMER2_STOP_ST_CLR Configures whether to clear MCPWM1_EVT_TIMER2_STOP.

0: Invalid. No effect

1: Clear

(WT)

SOC_ETM_MCPWM1_EVT_TIMER0_TEZ_ST_CLR Configures whether to clear MCPWM1_EVT_TIMER0_TEZ.

0: Invalid. No effect

1: Clear

(WT)

SOC_ETM_MCPWM1_EVT_TIMER1_TEZ_ST_CLR Configures whether to clear MCPWM1_EVT_TIMER1_TEZ.

0: Invalid. No effect

1: Clear

(WT)

SOC_ETM_MCPWM1_EVT_TIMER2_TEZ_ST_CLR Configures whether to clear MCPWM1_EVT_TIMER2_TEZ.

0: Invalid. No effect

1: Clear

(WT)

SOC_ETM_MCPWM1_EVT_TIMER0_TEP_ST_CLR Configures whether to clear MCPWM1_EVT_TIMER0_TEP.

0: Invalid. No effect

1: Clear

(WT)

Continued on the next page...

Register 14.27. SOC_ETM_EVT_ST3_CLR_REG (0x01C4)

Continued from the previous page...

SOC_ETM_MCPWM1_EVT_TIMER1_TEP_ST_CLR Configures whether to clear MCPWM1_EVT_TIMER1_TEP.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_MCPWM1_EVT_TIMER2_TEP_ST_CLR Configures whether to clear MCPWM1_EVT_TIMER2_TEP.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_MCPWM1_EVT_OP0_TEA_ST_CLR Configures whether to clear MCPWM1_EVT_OP0_TEA.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_MCPWM1_EVT_OP1_TEA_ST_CLR Configures whether to clear MCPWM1_EVT_OP1_TEA.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_MCPWM1_EVT_OP2_TEA_ST_CLR Configures whether to clear MCPWM1_EVT_OP2_TEA.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_MCPWM1_EVT_OP0_TEB_ST_CLR Configures whether to clear MCPWM1_EVT_OP0_TEB.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_MCPWM1_EVT_OP1_TEB_ST_CLR Configures whether to clear MCPWM1_EVT_OP1_TEB.
0: Invalid. No effect
1: Clear
(WT)

Continued on the next page...

Register 14.27. SOC_ETM_EVT_ST3_CLR_REG (0x01C4)

Continued from the previous page...

SOC_ETM_MCPWM1_EVT_OP2_TEB_ST_CLR Configures whether to clear MCPWM1_EVT_OP2_TEB.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_MCPWM1_EVT_F0_ST_CLR Configures whether to clear MCPWM1_EVT_F0.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_MCPWM1_EVT_F1_ST_CLR Configures whether to clear MCPWM1_EVT_F1.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_MCPWM1_EVT_F2_ST_CLR Configures whether to clear MCPWM1_EVT_F2.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_MCPWM1_EVT_F0_CLR_ST_CLR Configures whether to clear MCPWM1_EVT_F0_CLR.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_MCPWM1_EVT_F1_CLR_ST_CLR Configures whether to clear MCPWM1_EVT_F1_CLR.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_MCPWM1_EVT_F2_CLR_ST_CLR Configures whether to clear MCPWM1_EVT_F2_CLR.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_MCPWM1_EVT_TZ0_CBC_ST_CLR Configures whether to clear MCPWM1_EVT_TZ0_CBC.
0: Invalid. No effect
1: Clear
(WT)

Continued on the next page...

Register 14.27. SOC_ETM_EVT_ST3_CLR_REG (0x01C4)

Continued from the previous page...

SOC_ETM_MCPWM1_EVT_TZ1_CBC_ST_CLR Configures whether to clear MCPWM1_EVT_TZ1_CBC.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_MCPWM1_EVT_TZ2_CBC_ST_CLR Configures whether to clear MCPWM1_EVT_TZ2_CBC.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_MCPWM1_EVT_TZ0_OST_ST_CLR Configures whether to clear MCPWM1_EVT_TZ0_OST.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_MCPWM1_EVT_TZ1_OST_ST_CLR Configures whether to clear MCPWM1_EVT_TZ1_OST.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_MCPWM1_EVT_TZ2_OST_ST_CLR Configures whether to clear MCPWM1_EVT_TZ2_OST.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_MCPWM1_EVT_CAP0_ST_CLR Configures whether to clear MCPWM1_EVT_CAP0.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_MCPWM1_EVT_CAP1_ST_CLR Configures whether to clear MCPWM1_EVT_CAP1.
0: Invalid. No effect
1: Clear
(WT)

Continued on the next page...

Register 14.27. SOC_ETM_EVT_ST3_CLR_REG (0x01C4)

Continued from the previous page...

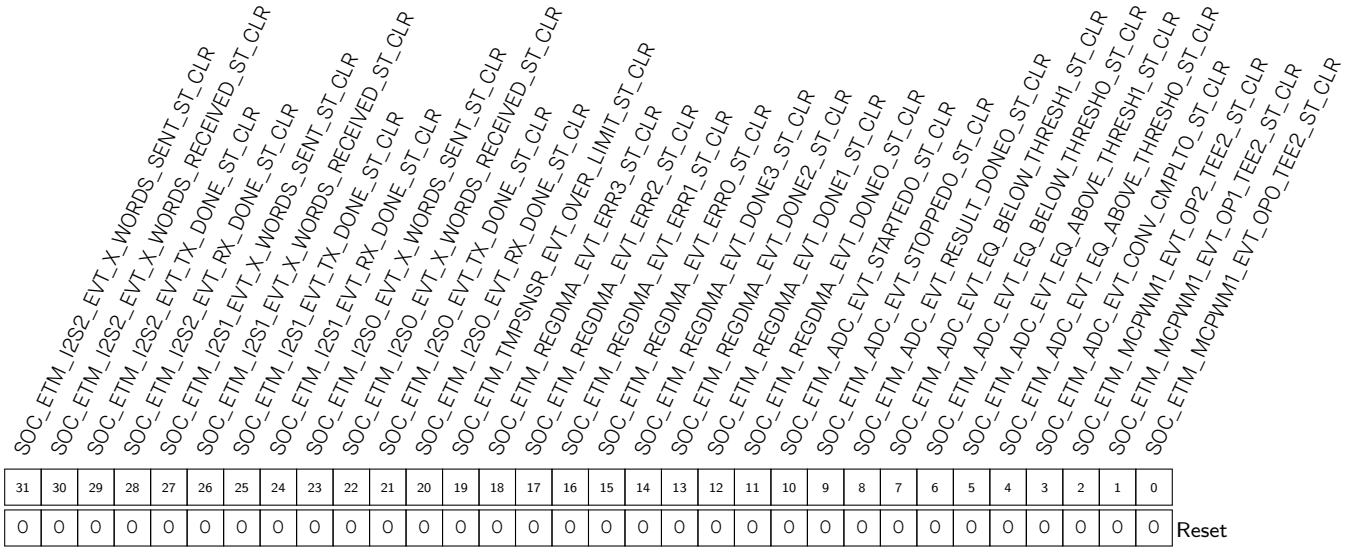
SOC_ETM_MCPWM1_EVT_CAP2_ST_CLR Configures whether to clear MCPWM1_EVT_CAP2.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_MCPWM1_EVT_OP0_TEE1_ST_CLR Configures whether to clear MCPWM1_EVT_OP0_TEE1.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_MCPWM1_EVT_OP1_TEE1_ST_CLR Configures whether to clear MCPWM1_EVT_OP1_TEE1.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_MCPWM1_EVT_OP2_TEE1_ST_CLR Configures whether to clear MCPWM1_EVT_OP2_TEE1.
0: Invalid. No effect
1: Clear
(WT)

Register 14.28. SOC_ETM_EVT_ST4_CLR_REG (0x01CC)



SOC_ETM_MCPWM1_EVT_OP0_TEE2_ST_CLR Configures whether to clear MCPWM1_EVT_OP0_TEE2.

0: Invalid. No effect

1: Clear

(WT)

SOC_ETM_MCPWM1_EVT_OP1_TEE2_ST_CLR Configures whether to clear MCPWM1_EVT_OP1_TEE2.

0: Invalid. No effect

1: Clear

(WT)

SOC_ETM_MCPWM1_EVT_OP2_TEE2_ST_CLR Configures whether to clear MCPWM1_EVT_OP2_TEE2.

0: Invalid. No effect

1: Clear

(WT)

SOC_ETM_ADC_EVT_CONV_CMPLTO_ST_CLR Configures whether to clear ADC_EVT_CONV_CMPLTO.

0: Invalid. No effect

1: Clear

(WT)

SOC_ETM_ADC_EVT_EQ_ABOVE_THRESH0_ST_CLR Configures whether to clear ADC_EVT_EQ_ABOVE_THRESH0.

0: Invalid. No effect

1: Clear

(WT)

Continued on the next page...

Register 14.28. SOC_ETM_EVT_ST4_CLR_REG (0x01CC)

Continued from the previous page...

SOC_ETM_ADC_EVT_EQ_ABOVE_THRESH1_ST_CLR Configures whether to clear ADC_EVT_EQ_ABOVE_THRESH1.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_ADC_EVT_EQ_BELOW_THRESH0_ST_CLR Configures whether to clear ADC_EVT_EQ_BELOW_THRESH0.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_ADC_EVT_EQ_BELOW_THRESH1_ST_CLR Configures whether to clear ADC_EVT_EQ_BELOW_THRESH1.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_ADC_EVT_RESULT_DONE0_ST_CLR Configures whether to clear ADC_EVT_RESULT_DONE0.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_ADC_EVT_STOPPED0_ST_CLR Configures whether to clear ADC_EVT_STOPPED0.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_ADC_EVT_STARTED0_ST_CLR Configures whether to clear ADC_EVT_STARTED0.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_REGDMA_EVT_DONE0_ST_CLR Configures whether to clear REGDMA_EVT_DONE0.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_REGDMA_EVT_DONE1_ST_CLR Configures whether to clear REGDMA_EVT_DONE1.
0: Invalid. No effect
1: Clear
(WT)

Continued on the next page...

Register 14.28. SOC_ETM_EVT_ST4_CLR_REG (0x01CC)

Continued from the previous page...

SOC_ETM_REGDMA_EVT_DONE2_ST_CLR Configures whether to clear REGDMA_EVT_DONE2.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_REGDMA_EVT_DONE3_ST_CLR Configures whether to clear REGDMA_EVT_DONE3.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_REGDMA_EVT_ERR0_ST_CLR Configures whether to clear REGDMA_EVT_ERR0.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_REGDMA_EVT_ERR1_ST_CLR Configures whether to clear REGDMA_EVT_ERR1.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_REGDMA_EVT_ERR2_ST_CLR Configures whether to clear REGDMA_EVT_ERR2.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_REGDMA_EVT_ERR3_ST_CLR Configures whether to clear REGDMA_EVT_ERR3.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_TMPSNSR_EVT_OVER_LIMIT_ST_CLR Configures whether to clear TMPSNSR_EVT_OVER_LIMIT.
0: Invalid. No effect
1: Clear
(WT)

Continued on the next page...

Register 14.28. SOC_ETM_EVT_ST4_CLR_REG (0x01CC)

Continued from the previous page...

SOC_ETM_I2S0_EVT_RX_DONE_ST_CLR Configures whether to clear I2S0_EVT_RX_DONE.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_I2S0_EVT_TX_DONE_ST_CLR Configures whether to clear I2S0_EVT_TX_DONE.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_I2S0_EVT_X_WORDS_RECEIVED_ST_CLR Configures whether to clear I2S0_EVT_X_WORDS_RECEIVED.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_I2S0_EVT_X_WORDS_SENT_ST_CLR Configures whether to clear I2S0_EVT_X_WORDS_SENT.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_I2S1_EVT_RX_DONE_ST_CLR Configures whether to clear I2S1_EVT_RX_DONE.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_I2S1_EVT_TX_DONE_ST_CLR Configures whether to clear I2S1_EVT_TX_DONE.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_I2S1_EVT_X_WORDS_RECEIVED_ST_CLR Configures whether to clear I2S1_EVT_X_WORDS_RECEIVED.
0: Invalid. No effect
1: Clear
(WT)

Continued on the next page...

Register 14.28. SOC_ETM_EVT_ST4_CLR_REG (0x01CC)

Continued from the previous page...

SOC_ETM_I2S1_EVT_X_WORDS_SENT_ST_CLR Configures whether to clear I2S1_EVT_X_WORDS_SENT.
0: Invalid. No effect
1: Clear
(WT)

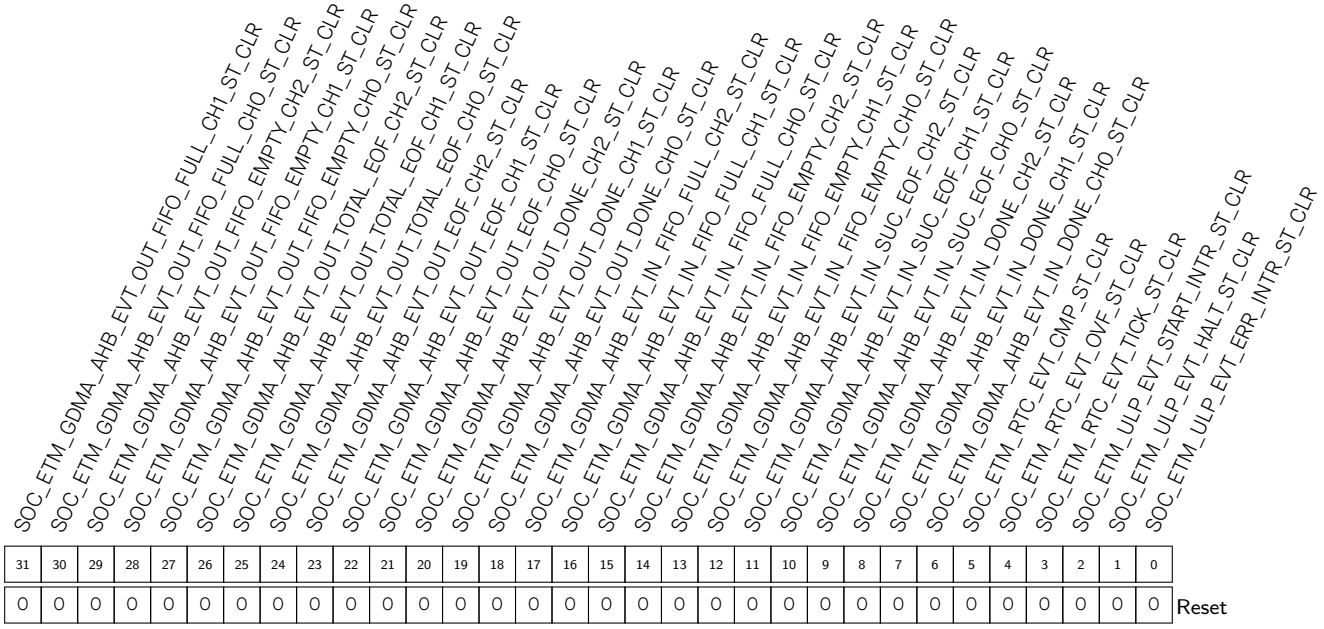
SOC_ETM_I2S2_EVT_RX_DONE_ST_CLR Configures whether to clear I2S2_EVT_RX_DONE.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_I2S2_EVT_TX_DONE_ST_CLR Configures whether to clear I2S2_EVT_TX_DONE.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_I2S2_EVT_X_WORDS_RECEIVED_ST_CLR Configures whether to clear I2S2_EVT_X_WORDS_RECEIVED.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_I2S2_EVT_X_WORDS_SENT_ST_CLR Configures whether to clear I2S2_EVT_X_WORDS_SENT.
0: Invalid. No effect
1: Clear
(WT)

Register 14.29. SOC_ETM_EVT_ST5_CLR_REG (0x01D4)



SOC_ETM_ULP_EVT_ERR_INTR_ST_CLR Configures whether to clear ULP_EVT_ERR_INTR.

0: Invalid. No effect

1: Clear

(WT)

SOC_ETM_ULP_EVT_HALT_ST_CLR Configures whether to clear ULP_EVT_HALT.

0: Invalid. No effect

1: Clear

(WT)

SOC_ETM_ULP_EVT_START_INTR_ST_CLR Configures whether to clear ULP_EVT_START_INTR.

0: Invalid. No effect

1: Clear

(WT)

SOC_ETM_RTC_EVT_TICK_ST_CLR Configures whether to clear RTC_EVT_TICK.

0: Invalid. No effect

1: Clear

(WT)

SOC_ETM_RTC_EVT_OVF_ST_CLR Configures whether to clear RTC_EVT_OVF.

0: Invalid. No effect

1: Clear

(WT)

SOC_ETM_RTC_EVT_CMP_ST_CLR Configures whether to clear RTC_EVT_CMP.

0: Invalid. No effect

1: Clear

(WT)

Continued on the next page...

Register 14.29. SOC_ETM_EVT_ST5_CLR_REG (0x01D4)

Continued from the previous page...

SOC_ETM_GDMA_AHB_EVT_IN_DONE_CH0_ST_CLR Configures whether to clear GDMA_AHB_EVT_IN_DONE_CH0.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_GDMA_AHB_EVT_IN_DONE_CH1_ST_CLR Configures whether to clear GDMA_AHB_EVT_IN_DONE_CH1.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_GDMA_AHB_EVT_IN_DONE_CH2_ST_CLR Configures whether to clear GDMA_AHB_EVT_IN_DONE_CH2.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_GDMA_AHB_EVT_IN_SUC_EOF_CH0_ST_CLR Configures whether to clear GDMA_AHB_EVT_IN_SUC_EOF_CH0.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_GDMA_AHB_EVT_IN_SUC_EOF_CH1_ST_CLR Configures whether to clear GDMA_AHB_EVT_IN_SUC_EOF_CH1.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_GDMA_AHB_EVT_IN_SUC_EOF_CH2_ST_CLR Configures whether to clear GDMA_AHB_EVT_IN_SUC_EOF_CH2.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_GDMA_AHB_EVT_IN_FIFO_EMPTY_CH0_ST_CLR Configures whether to clear GDMA_AHB_EVT_IN_FIFO_EMPTY_CH0.
0: Invalid. No effect
1: Clear
(WT)

Continued on the next page...

Register 14.29. SOC_ETM_EVT_ST5_CLR_REG (0x01D4)

Continued from the previous page...

SOC_ETM_GDMA_AHB_EVT_IN_FIFO_EMPTY_CH1_ST_CLR Configures whether to clear GDMA_AHB_EVT_IN_FIFO_EMPTY_CH1.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_GDMA_AHB_EVT_IN_FIFO_EMPTY_CH2_ST_CLR Configures whether to clear GDMA_AHB_EVT_IN_FIFO_EMPTY_CH2.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_GDMA_AHB_EVT_IN_FIFO_FULL_CH0_ST_CLR Configures whether to clear GDMA_AHB_EVT_IN_FIFO_FULL_CH0.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_GDMA_AHB_EVT_IN_FIFO_FULL_CH1_ST_CLR Configures whether to clear GDMA_AHB_EVT_IN_FIFO_FULL_CH1.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_GDMA_AHB_EVT_IN_FIFO_FULL_CH2_ST_CLR Configures whether to clear GDMA_AHB_EVT_IN_FIFO_FULL_CH2.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_GDMA_AHB_EVT_OUT_DONE_CH0_ST_CLR Configures whether to clear GDMA_AHB_EVT_OUT_DONE_CH0.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_GDMA_AHB_EVT_OUT_DONE_CH1_ST_CLR Configures whether to clear GDMA_AHB_EVT_OUT_DONE_CH1.
0: Invalid. No effect
1: Clear
(WT)

Continued on the next page...

Register 14.29. SOC_ETM_EVT_ST5_CLR_REG (0x01D4)

Continued from the previous page...

SOC_ETM_GDMA_AHB_EVT_OUT_DONE_CH2_ST_CLR Configures whether to clear GDMA_AHB_EVT_OUT_DONE_CH2.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_GDMA_AHB_EVT_OUT_EOF_CH0_ST_CLR Configures whether to clear GDMA_AHB_EVT_OUT_EOF_CH0.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_GDMA_AHB_EVT_OUT_EOF_CH1_ST_CLR Configures whether to clear GDMA_AHB_EVT_OUT_EOF_CH1.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_GDMA_AHB_EVT_OUT_EOF_CH2_ST_CLR Configures whether to clear GDMA_AHB_EVT_OUT_EOF_CH2.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_GDMA_AHB_EVT_OUT_TOTAL_EOF_CH0_ST_CLR Configures whether to clear GDMA_AHB_EVT_OUT_TOTAL_EOF_CH0.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_GDMA_AHB_EVT_OUT_TOTAL_EOF_CH1_ST_CLR Configures whether to clear GDMA_AHB_EVT_OUT_TOTAL_EOF_CH1.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_GDMA_AHB_EVT_OUT_TOTAL_EOF_CH2_ST_CLR Configures whether to clear GDMA_AHB_EVT_OUT_TOTAL_EOF_CH2.
0: Invalid. No effect
1: Clear
(WT)

Continued on the next page...

Register 14.29. SOC_ETM_EVT_ST5_CLR_REG (0x01D4)

Continued from the previous page...

SOC_ETM_GDMA_AHB_EVT_OUT_FIFO_EMPTY_CH0_ST_CLR Configures whether to clear GDMA_AHB_EVT_OUT_FIFO_EMPTY_CH0.
0: Invalid. No effect
1: Clear
(WT)

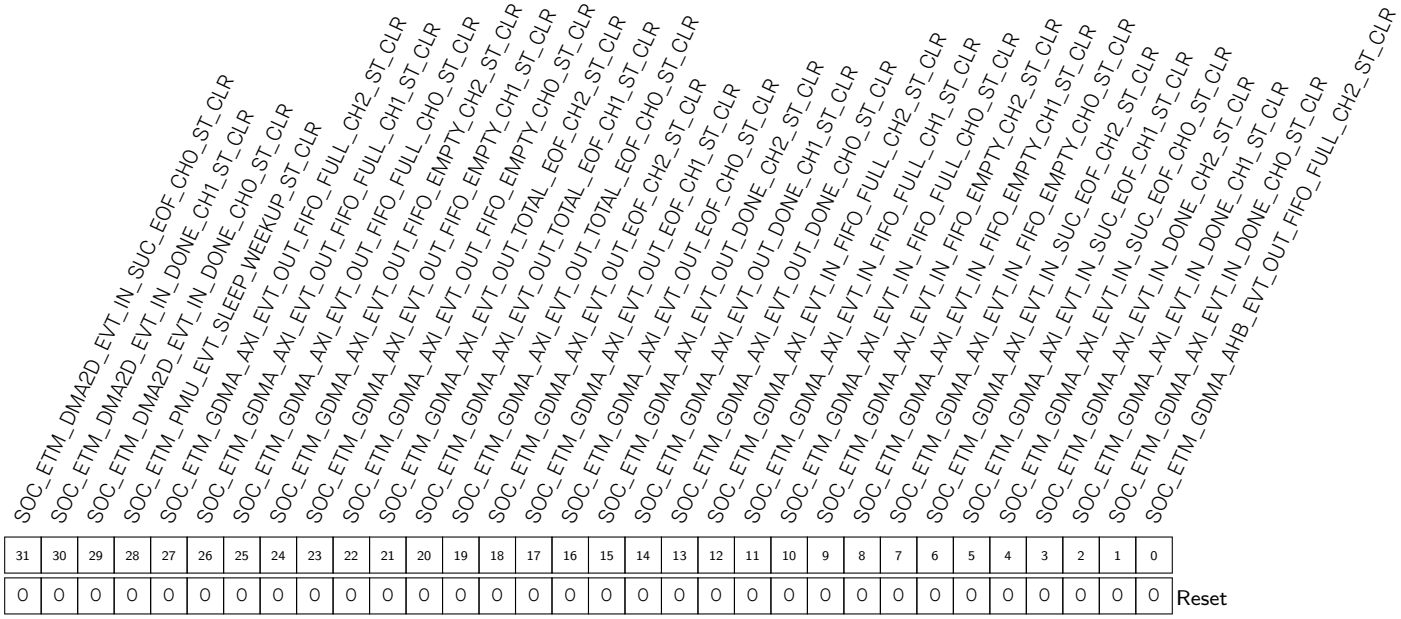
SOC_ETM_GDMA_AHB_EVT_OUT_FIFO_EMPTY_CH1_ST_CLR Configures whether to clear GDMA_AHB_EVT_OUT_FIFO_EMPTY_CH1.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_GDMA_AHB_EVT_OUT_FIFO_EMPTY_CH2_ST_CLR Configures whether to clear GDMA_AHB_EVT_OUT_FIFO_EMPTY_CH2.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_GDMA_AHB_EVT_OUT_FIFO_FULL_CH0_ST_CLR Configures whether to clear GDMA_AHB_EVT_OUT_FIFO_FULL_CH0.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_GDMA_AHB_EVT_OUT_FIFO_FULL_CH1_ST_CLR Configures whether to clear GDMA_AHB_EVT_OUT_FIFO_FULL_CH1.
0: Invalid. No effect
1: Clear
(WT)

Register 14.30. SOC_ETM_EVT_ST6_CLR_REG (0x01DC)



31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Reset

SOC_ETM_GDMA_AHB_EVT_OUT_FIFO_FULL_CH2_ST_CLR Configures whether to clear GDMA_AHB_EVT_OUT_FIFO_FULL_CH2.

0: Invalid. No effect

1: Clear

(WT)

SOC_ETM_GDMA_AXI_EVT_IN_DONE_CH0_ST_CLR Configures whether to clear GDMA_AXI_EVT_IN_DONE_CH0.

0: Invalid. No effect

1: Clear

(WT)

SOC_ETM_GDMA_AXI_EVT_IN_DONE_CH1_ST_CLR Configures whether to clear GDMA_AXI_EVT_IN_DONE_CH1.

0: Invalid. No effect

1: Clear

(WT)

SOC_ETM_GDMA_AXI_EVT_IN_DONE_CH2_ST_CLR Configures whether to clear GDMA_AXI_EVT_IN_DONE_CH2.

0: Invalid. No effect

1: Clear

(WT)

SOC_ETM_GDMA_AXI_EVT_IN_SUC_EOF_CH0_ST_CLR Configures whether to clear GDMA_AXI_EVT_IN_SUC_EOF_CH0.

0: Invalid. No effect

1: Clear

(WT)

Continued on the next page...

Register 14.30. SOC_ETM_EVT_ST6_CLR_REG (0x01DC)

Continued from the previous page...

SOC_ETM_GDMA_AXI_EVT_IN_SUC_EOF_CH1_ST_CLR Configures whether to clear GDMA_AXI_EVT_IN_SUC_EOF_CH1.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_GDMA_AXI_EVT_IN_SUC_EOF_CH2_ST_CLR Configures whether to clear GDMA_AXI_EVT_IN_SUC_EOF_CH2.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_GDMA_AXI_EVT_IN_FIFO_EMPTY_CH0_ST_CLR Configures whether to clear GDMA_AXI_EVT_IN_FIFO_EMPTY_CH0.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_GDMA_AXI_EVT_IN_FIFO_EMPTY_CH1_ST_CLR Configures whether to clear GDMA_AXI_EVT_IN_FIFO_EMPTY_CH1.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_GDMA_AXI_EVT_IN_FIFO_EMPTY_CH2_ST_CLR Configures whether to clear GDMA_AXI_EVT_IN_FIFO_EMPTY_CH2.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_GDMA_AXI_EVT_IN_FIFO_FULL_CH0_ST_CLR Configures whether to clear GDMA_AXI_EVT_IN_FIFO_FULL_CH0.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_GDMA_AXI_EVT_IN_FIFO_FULL_CH1_ST_CLR Configures whether to clear GDMA_AXI_EVT_IN_FIFO_FULL_CH1.
0: Invalid. No effect
1: Clear
(WT)

Continued on the next page...

Register 14.30. SOC_ETM_EVT_ST6_CLR_REG (0x01DC)

Continued from the previous page...

SOC_ETM_GDMA_AXI_EVT_IN_FIFO_FULL_CH2_ST_CLR Configures whether to clear GDMA_AXI_EVT_IN_FIFO_FULL_CH2.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_GDMA_AXI_EVT_OUT_DONE_CH0_ST_CLR Configures whether to clear GDMA_AXI_EVT_OUT_DONE_CH0.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_GDMA_AXI_EVT_OUT_DONE_CH1_ST_CLR Configures whether to clear GDMA_AXI_EVT_OUT_DONE_CH1.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_GDMA_AXI_EVT_OUT_DONE_CH2_ST_CLR Configures whether to clear GDMA_AXI_EVT_OUT_DONE_CH2.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_GDMA_AXI_EVT_OUT_EOF_CH0_ST_CLR Configures whether to clear GDMA_AXI_EVT_OUT_EOF_CH0.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_GDMA_AXI_EVT_OUT_EOF_CH1_ST_CLR Configures whether to clear GDMA_AXI_EVT_OUT_EOF_CH1.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_GDMA_AXI_EVT_OUT_EOF_CH2_ST_CLR Configures whether to clear GDMA_AXI_EVT_OUT_EOF_CH2.
0: Invalid. No effect
1: Clear
(WT)

Continued on the next page...

Register 14.30. SOC_ETM_EVT_ST6_CLR_REG (0x01DC)

Continued from the previous page...

SOC_ETM_GDMA_AXI_EVT_OUT_TOTAL_EOF_CH0_ST_CLR Configures whether to clear GDMA_AXI_EVT_OUT_TOTAL_EOF_CH0.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_GDMA_AXI_EVT_OUT_TOTAL_EOF_CH1_ST_CLR Configures whether to clear GDMA_AXI_EVT_OUT_TOTAL_EOF_CH1.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_GDMA_AXI_EVT_OUT_TOTAL_EOF_CH2_ST_CLR Configures whether to clear GDMA_AXI_EVT_OUT_TOTAL_EOF_CH2.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_GDMA_AXI_EVT_OUT_FIFO_EMPTY_CH0_ST_CLR Configures whether to clear GDMA_AXI_EVT_OUT_FIFO_EMPTY_CH0.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_GDMA_AXI_EVT_OUT_FIFO_EMPTY_CH1_ST_CLR Configures whether to clear GDMA_AXI_EVT_OUT_FIFO_EMPTY_CH1.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_GDMA_AXI_EVT_OUT_FIFO_EMPTY_CH2_ST_CLR Configures whether to clear GDMA_AXI_EVT_OUT_FIFO_EMPTY_CH2.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_GDMA_AXI_EVT_OUT_FIFO_FULL_CH0_ST_CLR Configures whether to clear GDMA_AXI_EVT_OUT_FIFO_FULL_CH0.
0: Invalid. No effect
1: Clear
(WT)

Continued on the next page...

Register 14.30. SOC_ETM_EVT_ST6_CLR_REG (0x01DC)

Continued from the previous page...

SOC_ETM_GDMA_AXI_EVT_OUT_FIFO_FULL_CH1_ST_CLR Configures whether to clear GDMA_AXI_EVT_OUT_FIFO_FULL_CH1.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_GDMA_AXI_EVT_OUT_FIFO_FULL_CH2_ST_CLR Configures whether to clear GDMA_AXI_EVT_OUT_FIFO_FULL_CH2.
0: Invalid. No effect
1: Clear
(WT)

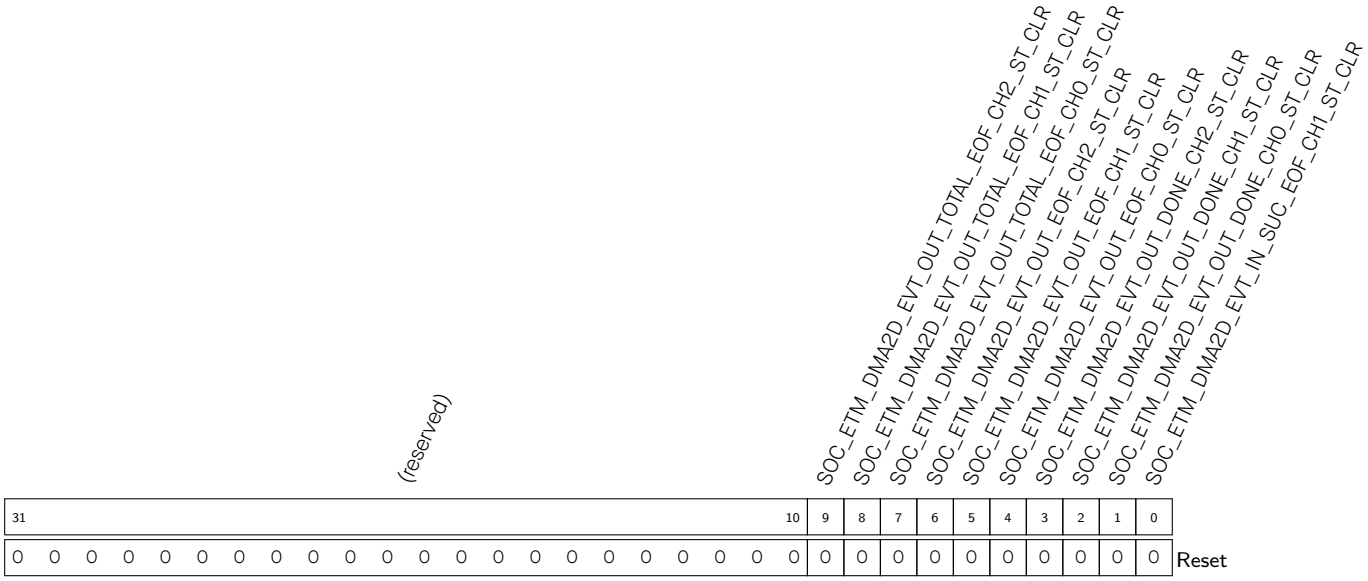
SOC_ETM_PMU_EVT_SLEEP_WEEKUP_ST_CLR Configures whether to clear PMU_EVT_SLEEP_WEEKUP.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_DMA2D_EVT_IN_DONE_CH0_ST_CLR Configures whether to clear DMA2D_EVT_IN_DONE_CH0.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_DMA2D_EVT_IN_DONE_CH1_ST_CLR Configures whether to clear DMA2D_EVT_IN_DONE_CH1.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_DMA2D_EVT_IN_SUC_EOF_CH0_ST_CLR Configures whether to clear DMA2D_EVT_IN_SUC_EOF_CH0.
0: Invalid. No effect
1: Clear
(WT)

Register 14.31. SOC_ETM_EVT_ST7_CLR_REG (0x01E4)



SOC_ETM_DMA2D_EVT_IN_SUC_EOF_CH1_ST_CLR Configures whether to clear DMA2D_EVT_IN_SUC_EOF_CH1.

0: Invalid. No effect

1: Clear

(WT)

SOC_ETM_DMA2D_EVT_OUT_DONE_CH0_ST_CLR Configures whether to clear DMA2D_EVT_OUT_DONE_CH0.

0: Invalid. No effect

1: Clear

(WT)

SOC_ETM_DMA2D_EVT_OUT_DONE_CH1_ST_CLR Configures whether to clear DMA2D_EVT_OUT_DONE_CH1.

0: Invalid. No effect

1: Clear

(WT)

SOC_ETM_DMA2D_EVT_OUT_DONE_CH2_ST_CLR Configures whether to clear DMA2D_EVT_OUT_DONE_CH2.

0: Invalid. No effect

1: Clear

(WT)

SOC_ETM_DMA2D_EVT_OUT_EOF_CH0_ST_CLR Configures whether to clear DMA2D_EVT_OUT_EOF_CH0.

0: Invalid. No effect

1: Clear

(WT)

Continued on the next page...

Register 14.31. SOC_ETM_EVT_ST7_CLR_REG (0x01E4)

Continued from the previous page...

SOC_ETM_DMA2D_EVT_OUT_EOF_CH1_ST_CLR Configures whether to clear DMA2D_EVT_OUT_EOF_CH1.
0: Invalid. No effect
1: Clear
(WT)

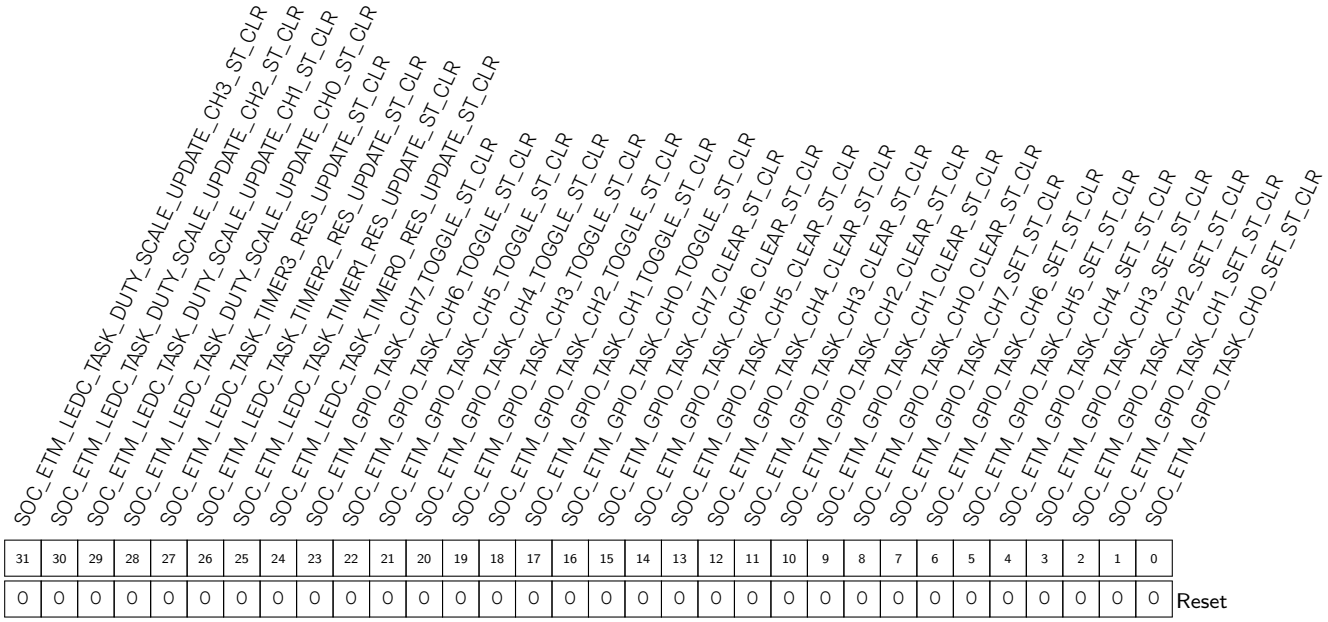
SOC_ETM_DMA2D_EVT_OUT_EOF_CH2_ST_CLR Configures whether to clear DMA2D_EVT_OUT_EOF_CH2.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_DMA2D_EVT_OUT_TOTAL_EOF_CH0_ST_CLR Configures whether to clear DMA2D_EVT_OUT_TOTAL_EOF_CH0.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_DMA2D_EVT_OUT_TOTAL_EOF_CH1_ST_CLR Configures whether to clear DMA2D_EVT_OUT_TOTAL_EOF_CH1.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_DMA2D_EVT_OUT_TOTAL_EOF_CH2_ST_CLR Configures whether to clear DMA2D_EVT_OUT_TOTAL_EOF_CH2.
0: Invalid. No effect
1: Clear
(WT)

Register 14.32. SOC_ETM_TASK_ST0_CLR_REG (0x01EC)



SOC_ETM_GPIO_TASK_CH0_SET_ST_CLR Configures whether to clear GPIO_TASK_CH0_SET.

0: Invalid. No effect

1: Clear

(WT)

SOC_ETM_GPIO_TASK_CH1_SET_ST_CLR Configures whether to clear GPIO_TASK_CH1_SET.

0: Invalid. No effect

1: Clear

(WT)

SOC_ETM_GPIO_TASK_CH2_SET_ST_CLR Configures whether to clear GPIO_TASK_CH2_SET.

0: Invalid. No effect

1: Clear

(WT)

SOC_ETM_GPIO_TASK_CH3_SET_ST_CLR Configures whether to clear GPIO_TASK_CH3_SET.

0: Invalid. No effect

1: Clear

(WT)

SOC_ETM_GPIO_TASK_CH4_SET_ST_CLR Configures whether to clear GPIO_TASK_CH4_SET.

0: Invalid. No effect

1: Clear

(WT)

SOC_ETM_GPIO_TASK_CH5_SET_ST_CLR Configures whether to clear GPIO_TASK_CH5_SET.

0: Invalid. No effect

1: Clear

(WT)

Continued on the next page...

Register 14.32. SOC_ETM_TASK_ST0_CLR_REG (0x01EC)

Continued from the previous page...

SOC_ETM_GPIO_TASK_CH6_SET_ST_CLR Configures whether to clear GPIO_TASK_CH6_SET.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_GPIO_TASK_CH7_SET_ST_CLR Configures whether to clear GPIO_TASK_CH7_SET.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_GPIO_TASK_CH0_CLEAR_ST_CLR Configures whether to clear GPIO_TASK_CH0_CLEAR.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_GPIO_TASK_CH1_CLEAR_ST_CLR Configures whether to clear GPIO_TASK_CH1_CLEAR.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_GPIO_TASK_CH2_CLEAR_ST_CLR Configures whether to clear GPIO_TASK_CH2_CLEAR.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_GPIO_TASK_CH3_CLEAR_ST_CLR Configures whether to clear GPIO_TASK_CH3_CLEAR.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_GPIO_TASK_CH4_CLEAR_ST_CLR Configures whether to clear GPIO_TASK_CH4_CLEAR.
0: Invalid. No effect
1: Clear
(WT)

Continued on the next page...

Register 14.32. SOC_ETM_TASK_ST0_CLR_REG (0x01EC)

Continued from the previous page...

SOC_ETM_GPIO_TASK_CH5_CLEAR_ST_CLR Configures whether to clear GPIO_TASK_CH5_CLEAR.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_GPIO_TASK_CH6_CLEAR_ST_CLR Configures whether to clear GPIO_TASK_CH6_CLEAR.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_GPIO_TASK_CH7_CLEAR_ST_CLR Configures whether to clear GPIO_TASK_CH7_CLEAR.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_GPIO_TASK_CH0_TOGGLE_ST_CLR Configures whether to clear GPIO_TASK_CH0_TOGGLE.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_GPIO_TASK_CH1_TOGGLE_ST_CLR Configures whether to clear GPIO_TASK_CH1_TOGGLE.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_GPIO_TASK_CH2_TOGGLE_ST_CLR Configures whether to clear GPIO_TASK_CH2_TOGGLE.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_GPIO_TASK_CH3_TOGGLE_ST_CLR Configures whether to clear GPIO_TASK_CH3_TOGGLE.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_GPIO_TASK_CH4_TOGGLE_ST_CLR Configures whether to clear GPIO_TASK_CH4_TOGGLE.
0: Invalid. No effect
1: Clear
(WT)

Continued on the next page...

Register 14.32. SOC_ETM_TASK_ST0_CLR_REG (0x01EC)

Continued from the previous page...

SOC_ETM_GPIO_TASK_CH5_TOGGLE_ST_CLR Configures whether to clear GPIO_TASK_CH5_TOGGLE.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_GPIO_TASK_CH6_TOGGLE_ST_CLR Configures whether to clear GPIO_TASK_CH6_TOGGLE.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_GPIO_TASK_CH7_TOGGLE_ST_CLR Configures whether to clear GPIO_TASK_CH7_TOGGLE.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_LEDC_TASK_TIMER0_RES_UPDATE_ST_CLR Configures whether to clear LEDC_TASK_TIMER0_RES_UPDATE.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_LEDC_TASK_TIMER1_RES_UPDATE_ST_CLR Configures whether to clear LEDC_TASK_TIMER1_RES_UPDATE.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_LEDC_TASK_TIMER2_RES_UPDATE_ST_CLR Configures whether to clear LEDC_TASK_TIMER2_RES_UPDATE.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_LEDC_TASK_TIMER3_RES_UPDATE_ST_CLR Configures whether to clear LEDC_TASK_TIMER3_RES_UPDATE.
0: Invalid. No effect
1: Clear
(WT)

Continued on the next page...

Register 14.32. SOC_ETM_TASK_ST0_CLR_REG (0x01EC)

Continued from the previous page...

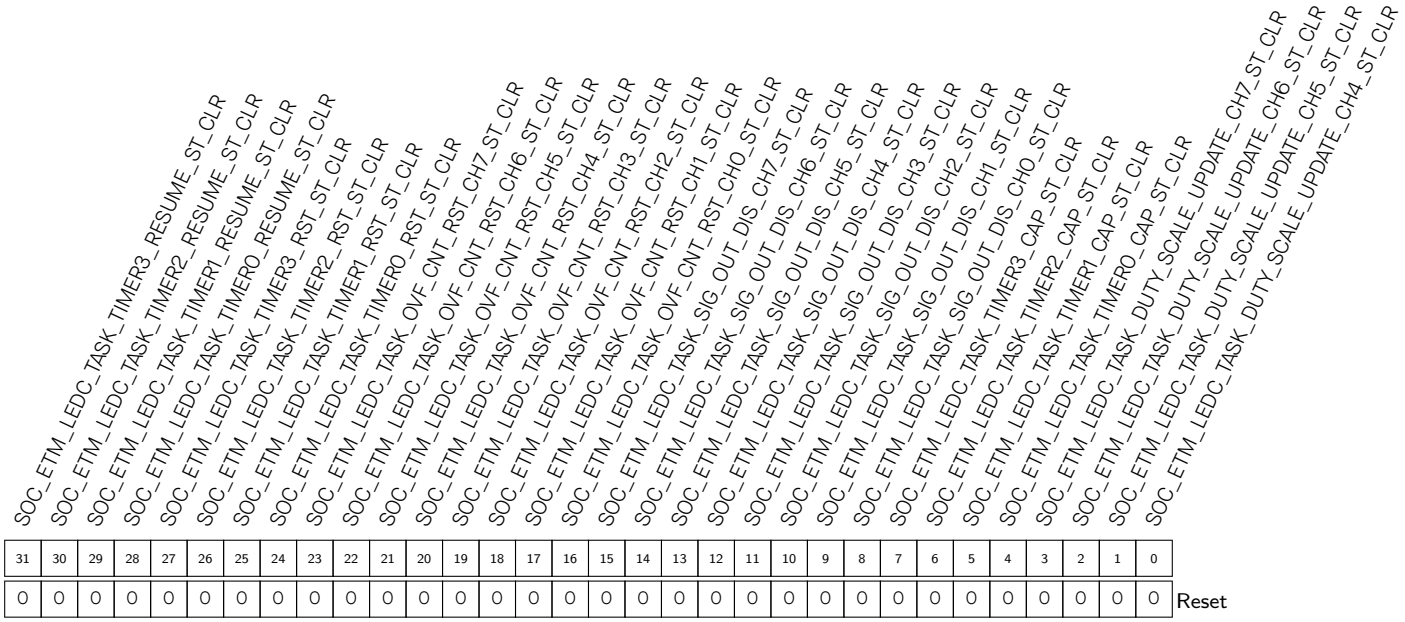
SOC_ETM_LEDC_TASK_DUTY_SCALE_UPDATE_CH0_ST_CLR Configures whether to clear LEDC_TASK_DUTY_SCALE_UPDATE_CH0.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_LEDC_TASK_DUTY_SCALE_UPDATE_CH1_ST_CLR Configures whether to clear LEDC_TASK_DUTY_SCALE_UPDATE_CH1.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_LEDC_TASK_DUTY_SCALE_UPDATE_CH2_ST_CLR Configures whether to clear LEDC_TASK_DUTY_SCALE_UPDATE_CH2.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_LEDC_TASK_DUTY_SCALE_UPDATE_CH3_ST_CLR Configures whether to clear LEDC_TASK_DUTY_SCALE_UPDATE_CH3.
0: Invalid. No effect
1: Clear
(WT)

Register 14.33. SOC_ETM_TASK_ST1_CLR_REG (0x01F4)



31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Reset

SOC_ETM_LEDC_TASK_DUTY_SCALE_UPDATE_CH4_ST_CLR Configures whether to clear LEDC_TASK_DUTY_SCALE_UPDATE_CH4.

0: Invalid. No effect

1: Clear

(WT)

SOC_ETM_LEDC_TASK_DUTY_SCALE_UPDATE_CH5_ST_CLR Configures whether to clear LEDC_TASK_DUTY_SCALE_UPDATE_CH5.

0: Invalid. No effect

1: Clear

(WT)

SOC_ETM_LEDC_TASK_DUTY_SCALE_UPDATE_CH6_ST_CLR Configures whether to clear LEDC_TASK_DUTY_SCALE_UPDATE_CH6.

0: Invalid. No effect

1: Clear

(WT)

SOC_ETM_LEDC_TASK_DUTY_SCALE_UPDATE_CH7_ST_CLR Configures whether to clear LEDC_TASK_DUTY_SCALE_UPDATE_CH7.

0: Invalid. No effect

1: Clear

(WT)

SOC_ETM_LEDC_TASK_TIMER0_CAP_ST_CLR Configures whether to clear LEDC_TASK_TIMER0_CAP.

0: Invalid. No effect

1: Clear

(WT)

Continued on the next page...

Register 14.33. SOC_ETM_TASK_ST1_CLR_REG (0x01F4)

Continued from the previous page...

SOC_ETM_LEDC_TASK_TIMER1_CAP_ST_CLR Configures whether to clear LEDC_TASK_TIMER1_CAP.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_LEDC_TASK_TIMER2_CAP_ST_CLR Configures whether to clear LEDC_TASK_TIMER2_CAP.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_LEDC_TASK_TIMER3_CAP_ST_CLR Configures whether to clear LEDC_TASK_TIMER3_CAP.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_LEDC_TASK_SIG_OUT_DIS_CH0_ST_CLR Configures whether to clear LEDC_TASK_SIG_OUT_DIS_CH0.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_LEDC_TASK_SIG_OUT_DIS_CH1_ST_CLR Configures whether to clear LEDC_TASK_SIG_OUT_DIS_CH1.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_LEDC_TASK_SIG_OUT_DIS_CH2_ST_CLR Configures whether to clear LEDC_TASK_SIG_OUT_DIS_CH2.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_LEDC_TASK_SIG_OUT_DIS_CH3_ST_CLR Configures whether to clear LEDC_TASK_SIG_OUT_DIS_CH3.
0: Invalid. No effect
1: Clear
(WT)

Continued on the next page...

Register 14.33. SOC_ETM_TASK_ST1_CLR_REG (0x01F4)

Continued from the previous page...

SOC_ETM_LEDC_TASK_SIG_OUT_DIS_CH4_ST_CLR Configures whether to clear LEDC_TASK_SIG_OUT_DIS_CH4.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_LEDC_TASK_SIG_OUT_DIS_CH5_ST_CLR Configures whether to clear LEDC_TASK_SIG_OUT_DIS_CH5.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_LEDC_TASK_SIG_OUT_DIS_CH6_ST_CLR Configures whether to clear LEDC_TASK_SIG_OUT_DIS_CH6.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_LEDC_TASK_SIG_OUT_DIS_CH7_ST_CLR Configures whether to clear LEDC_TASK_SIG_OUT_DIS_CH7.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_LEDC_TASK_OVF_CNT_RST_CH0_ST_CLR Configures whether to clear LEDC_TASK_OVF_CNT_RST_CH0.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_LEDC_TASK_OVF_CNT_RST_CH1_ST_CLR Configures whether to clear LEDC_TASK_OVF_CNT_RST_CH1.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_LEDC_TASK_OVF_CNT_RST_CH2_ST_CLR Configures whether to clear LEDC_TASK_OVF_CNT_RST_CH2.
0: Invalid. No effect
1: Clear
(WT)

Continued on the next page...

Register 14.33. SOC_ETM_TASK_ST1_CLR_REG (0x01F4)

Continued from the previous page...

SOC_ETM_LEDC_TASK_OVF_CNT_RST_CH3_ST_CLR Configures whether to clear LEDC_TASK_OVF_CNT_RST_CH3.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_LEDC_TASK_OVF_CNT_RST_CH4_ST_CLR Configures whether to clear LEDC_TASK_OVF_CNT_RST_CH4.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_LEDC_TASK_OVF_CNT_RST_CH5_ST_CLR Configures whether to clear LEDC_TASK_OVF_CNT_RST_CH5.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_LEDC_TASK_OVF_CNT_RST_CH6_ST_CLR Configures whether to clear LEDC_TASK_OVF_CNT_RST_CH6.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_LEDC_TASK_OVF_CNT_RST_CH7_ST_CLR Configures whether to clear LEDC_TASK_OVF_CNT_RST_CH7.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_LEDC_TASK_TIMER0_RST_ST_CLR Configures whether to clear LEDC_TASK_TIMER0_RST.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_LEDC_TASK_TIMER1_RST_ST_CLR Configures whether to clear LEDC_TASK_TIMER1_RST.
0: Invalid. No effect
1: Clear
(WT)

Continued on the next page...

Register 14.33. SOC_ETM_TASK_ST1_CLR_REG (0x01F4)

Continued from the previous page...

SOC_ETM_LEDC_TASK_TIMER2_RST_ST_CLR Configures whether to clear LEDC_TASK_TIMER2_RST.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_LEDC_TASK_TIMER3_RST_ST_CLR Configures whether to clear LEDC_TASK_TIMER3_RST.
0: Invalid. No effect
1: Clear
(WT)

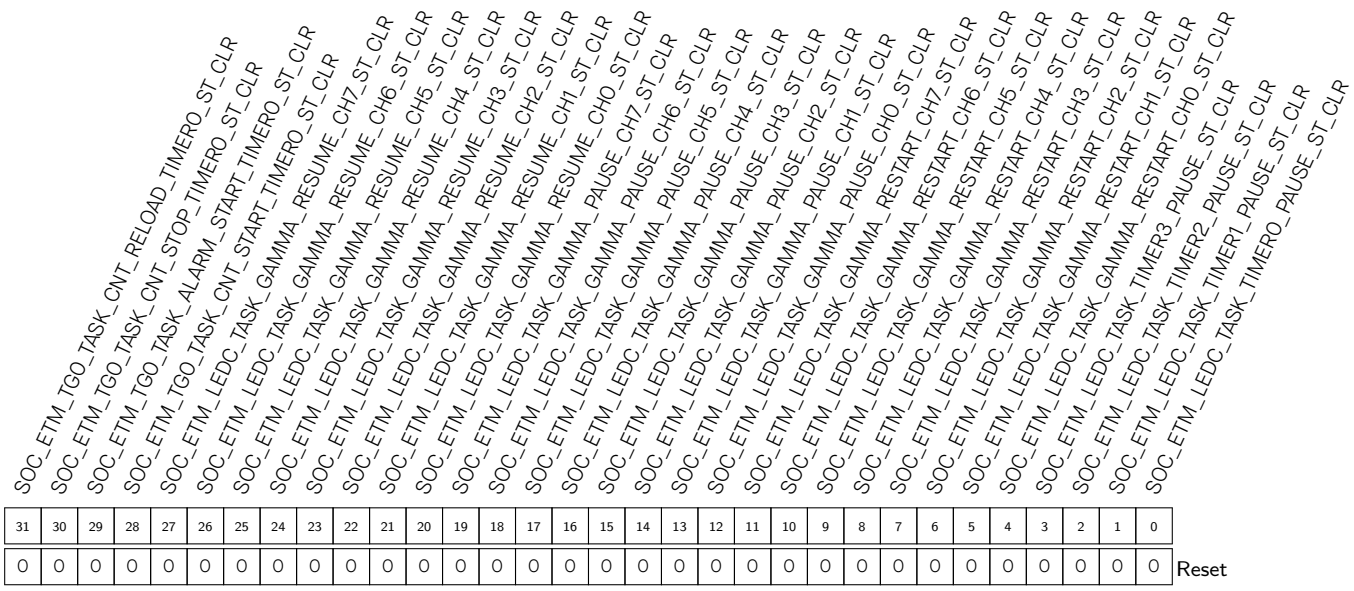
SOC_ETM_LEDC_TASK_TIMER0_RESUME_ST_CLR Configures whether to clear LEDC_TASK_TIMER0_RESUME.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_LEDC_TASK_TIMER1_RESUME_ST_CLR Configures whether to clear LEDC_TASK_TIMER1_RESUME.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_LEDC_TASK_TIMER2_RESUME_ST_CLR Configures whether to clear LEDC_TASK_TIMER2_RESUME.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_LEDC_TASK_TIMER3_RESUME_ST_CLR Configures whether to clear LEDC_TASK_TIMER3_RESUME.
0: Invalid. No effect
1: Clear
(WT)

Register 14.34. SOC_ETM_TASK_ST2_CLR_REG (0x01FC)



SOC_ETM_LEDC_TASK_TIMER0_PAUSE_ST_CLR Configures whether to clear LEDC_TASK_TIMER0_PAUSE.

0: Invalid. No effect

1: Clear

(WT)

SOC_ETM_LEDC_TASK_TIMER1_PAUSE_ST_CLR Configures whether to clear LEDC_TASK_TIMER1_PAUSE.

0: Invalid. No effect

1: Clear

(WT)

SOC_ETM_LEDC_TASK_TIMER2_PAUSE_ST_CLR Configures whether to clear LEDC_TASK_TIMER2_PAUSE.

0: Invalid. No effect

1: Clear

(WT)

SOC_ETM_LEDC_TASK_TIMER3_PAUSE_ST_CLR Configures whether to clear LEDC_TASK_TIMER3_PAUSE.

0: Invalid. No effect

1: Clear

(WT)

SOC_ETM_LEDC_TASK_GAMMA_RESTART_CH0_ST_CLR Configures whether to clear LEDC_TASK_GAMMA_RESTART_CH0.

0: Invalid. No effect

1: Clear

(WT)

Continued on the next page...

Register 14.34. SOC_ETM_TASK_ST2_CLR_REG (0x01FC)

Continued from the previous page...

SOC_ETM_LEDC_TASK_GAMMA_RESTART_CH1_ST_CLR Configures whether to clear LEDC_TASK_GAMMA_RESTART_CH1.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_LEDC_TASK_GAMMA_RESTART_CH2_ST_CLR Configures whether to clear LEDC_TASK_GAMMA_RESTART_CH2.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_LEDC_TASK_GAMMA_RESTART_CH3_ST_CLR Configures whether to clear LEDC_TASK_GAMMA_RESTART_CH3.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_LEDC_TASK_GAMMA_RESTART_CH4_ST_CLR Configures whether to clear LEDC_TASK_GAMMA_RESTART_CH4.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_LEDC_TASK_GAMMA_RESTART_CH5_ST_CLR Configures whether to clear LEDC_TASK_GAMMA_RESTART_CH5.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_LEDC_TASK_GAMMA_RESTART_CH6_ST_CLR Configures whether to clear LEDC_TASK_GAMMA_RESTART_CH6.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_LEDC_TASK_GAMMA_RESTART_CH7_ST_CLR Configures whether to clear LEDC_TASK_GAMMA_RESTART_CH7.
0: Invalid. No effect
1: Clear
(WT)

Continued on the next page...

Register 14.34. SOC_ETM_TASK_ST2_CLR_REG (0x01FC)

Continued from the previous page...

SOC_ETM_LEDC_TASK_GAMMA_PAUSE_CH0_ST_CLR Configures whether to clear LEDC_TASK_GAMMA_PAUSE_CH0.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_LEDC_TASK_GAMMA_PAUSE_CH1_ST_CLR Configures whether to clear LEDC_TASK_GAMMA_PAUSE_CH1.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_LEDC_TASK_GAMMA_PAUSE_CH2_ST_CLR Configures whether to clear LEDC_TASK_GAMMA_PAUSE_CH2.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_LEDC_TASK_GAMMA_PAUSE_CH3_ST_CLR Configures whether to clear LEDC_TASK_GAMMA_PAUSE_CH3.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_LEDC_TASK_GAMMA_PAUSE_CH4_ST_CLR Configures whether to clear LEDC_TASK_GAMMA_PAUSE_CH4.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_LEDC_TASK_GAMMA_PAUSE_CH5_ST_CLR Configures whether to clear LEDC_TASK_GAMMA_PAUSE_CH5.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_LEDC_TASK_GAMMA_PAUSE_CH6_ST_CLR Configures whether to clear LEDC_TASK_GAMMA_PAUSE_CH6.
0: Invalid. No effect
1: Clear
(WT)

Continued on the next page...

Register 14.34. SOC_ETM_TASK_ST2_CLR_REG (0x01FC)

Continued from the previous page...

SOC_ETM_LEDC_TASK_GAMMA_PAUSE_CH7_ST_CLR Configures whether to clear LEDC_TASK_GAMMA_PAUSE_CH7.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_LEDC_TASK_GAMMA_RESUME_CH0_ST_CLR Configures whether to clear LEDC_TASK_GAMMA_RESUME_CH0.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_LEDC_TASK_GAMMA_RESUME_CH1_ST_CLR Configures whether to clear LEDC_TASK_GAMMA_RESUME_CH1.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_LEDC_TASK_GAMMA_RESUME_CH2_ST_CLR Configures whether to clear LEDC_TASK_GAMMA_RESUME_CH2.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_LEDC_TASK_GAMMA_RESUME_CH3_ST_CLR Configures whether to clear LEDC_TASK_GAMMA_RESUME_CH3.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_LEDC_TASK_GAMMA_RESUME_CH4_ST_CLR Configures whether to clear LEDC_TASK_GAMMA_RESUME_CH4.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_LEDC_TASK_GAMMA_RESUME_CH5_ST_CLR Configures whether to clear LEDC_TASK_GAMMA_RESUME_CH5.
0: Invalid. No effect
1: Clear
(WT)

Continued on the next page...

Register 14.34. SOC_ETM_TASK_ST2_CLR_REG (0x01FC)

Continued from the previous page...

SOC_ETM_LEDC_TASK_GAMMA_RESUME_CH6_ST_CLR Configures whether to clear LEDC_TASK_GAMMA_RESUME_CH6.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_LEDC_TASK_GAMMA_RESUME_CH7_ST_CLR Configures whether to clear LEDC_TASK_GAMMA_RESUME_CH7.
0: Invalid. No effect
1: Clear
(WT)

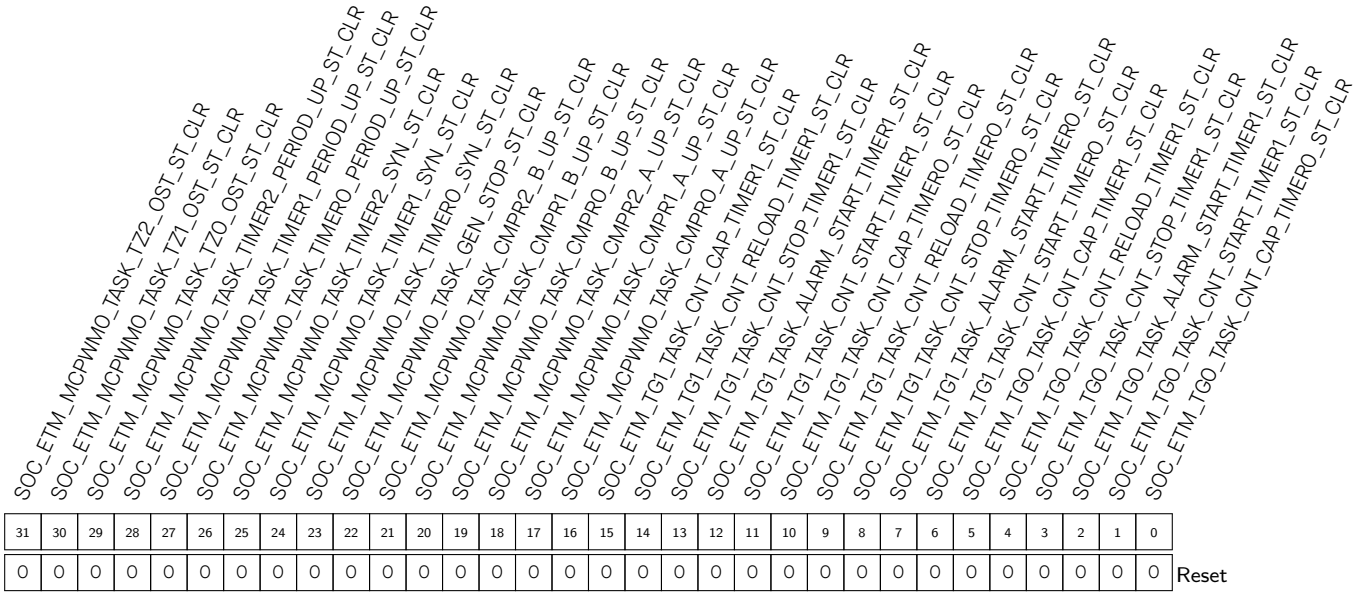
SOC_ETM_TGO_TASK_CNT_START_TIMER0_ST_CLR Configures whether to clear TGO_TASK_CNT_START_TIMER0.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_TGO_TASK_ALARM_START_TIMER0_ST_CLR Configures whether to clear TGO_TASK_ALARM_START_TIMER0.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_TGO_TASK_CNT_STOP_TIMER0_ST_CLR Configures whether to clear TGO_TASK_CNT_STOP_TIMER0.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_TGO_TASK_CNT_RELOAD_TIMER0_ST_CLR Configures whether to clear TGO_TASK_CNT_RELOAD_TIMER0.
0: Invalid. No effect
1: Clear
(WT)

Register 14.35. SOC_ETM_TASK_ST3_CLR_REG (0x0204)



31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Reset

SOC_ETM_TG0_TASK_CNT_CAP_TIMER0_ST_CLR Configures whether to clear TG0_TASK_CNT_CAP_TIMER0.

0: Invalid. No effect

1: Clear

(WT)

SOC_ETM_TG0_TASK_CNT_START_TIMER1_ST_CLR Configures whether to clear TG0_TASK_CNT_START_TIMER1.

0: Invalid. No effect

1: Clear

(WT)

SOC_ETM_TG0_TASK_ALARM_START_TIMER1_ST_CLR Configures whether to clear TG0_TASK_ALARM_START_TIMER1.

0: Invalid. No effect

1: Clear

(WT)

SOC_ETM_TG0_TASK_CNT_STOP_TIMER1_ST_CLR Configures whether to clear TG0_TASK_CNT_STOP_TIMER1.

0: Invalid. No effect

1: Clear

(WT)

SOC_ETM_TG0_TASK_CNT_RELOAD_TIMER1_ST_CLR Configures whether to clear TG0_TASK_CNT_RELOAD_TIMER1.

0: Invalid. No effect

1: Clear

(WT)

Continued on the next page...

Register 14.35. SOC_ETM_TASK_ST3_CLR_REG (0x0204)

Continued from the previous page...

SOC_ETM_TG0_TASK_CNT_CAP_TIMER1_ST_CLR Configures whether to clear TG0_TASK_CNT_CAP_TIMER1.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_TG1_TASK_CNT_START_TIMER0_ST_CLR Configures whether to clear TG1_TASK_CNT_START_TIMER0.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_TG1_TASK_ALARM_START_TIMER0_ST_CLR Configures whether to clear TG1_TASK_ALARM_START_TIMER0.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_TG1_TASK_CNT_STOP_TIMER0_ST_CLR Configures whether to clear TG1_TASK_CNT_STOP_TIMER0.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_TG1_TASK_CNT_RELOAD_TIMER0_ST_CLR Configures whether to clear TG1_TASK_CNT_RELOAD_TIMER0.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_TG1_TASK_CNT_CAP_TIMER0_ST_CLR Configures whether to clear TG1_TASK_CNT_CAP_TIMER0.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_TG1_TASK_CNT_START_TIMER1_ST_CLR Configures whether to clear TG1_TASK_CNT_START_TIMER1.
0: Invalid. No effect
1: Clear
(WT)

Continued on the next page...

Register 14.35. SOC_ETM_TASK_ST3_CLR_REG (0x0204)

Continued from the previous page...

SOC_ETM_TG1_TASK_ALARM_START_TIMER1_ST_CLR Configures whether to clear TG1_TASK_ALARM_START_TIMER1.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_TG1_TASK_CNT_STOP_TIMER1_ST_CLR Configures whether to clear TG1_TASK_CNT_STOP_TIMER1.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_TG1_TASK_CNT_RELOAD_TIMER1_ST_CLR Configures whether to clear TG1_TASK_CNT_RELOAD_TIMER1.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_TG1_TASK_CNT_CAP_TIMER1_ST_CLR Configures whether to clear TG1_TASK_CNT_CAP_TIMER1.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_MCPWM0_TASK_CMPR0_A_UP_ST_CLR Configures whether to clear MCPWM0_TASK_CMPR0_A_UP.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_MCPWM0_TASK_CMPR1_A_UP_ST_CLR Configures whether to clear MCPWM0_TASK_CMPR1_A_UP.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_MCPWM0_TASK_CMPR2_A_UP_ST_CLR Configures whether to clear MCPWM0_TASK_CMPR2_A_UP.
0: Invalid. No effect
1: Clear
(WT)

Continued on the next page...

Register 14.35. SOC_ETM_TASK_ST3_CLR_REG (0x0204)

Continued from the previous page...

SOC_ETM_MCPWM0_TASK_CMPR0_B_UP_ST_CLR Configures whether to clear MCPWM0_TASK_CMPR0_B_UP.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_MCPWM0_TASK_CMPR1_B_UP_ST_CLR Configures whether to clear MCPWM0_TASK_CMPR1_B_UP.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_MCPWM0_TASK_CMPR2_B_UP_ST_CLR Configures whether to clear MCPWM0_TASK_CMPR2_B_UP.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_MCPWM0_TASK_GEN_STOP_ST_CLR Configures whether to clear MCPWM0_TASK_GEN_STOP.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_MCPWM0_TASK_TIMER0_SYN_ST_CLR Configures whether to clear MCPWM0_TASK_TIMER0_SYN.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_MCPWM0_TASK_TIMER1_SYN_ST_CLR Configures whether to clear MCPWM0_TASK_TIMER1_SYN.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_MCPWM0_TASK_TIMER2_SYN_ST_CLR Configures whether to clear MCPWM0_TASK_TIMER2_SYN.
0: Invalid. No effect
1: Clear
(WT)

Continued on the next page...

Register 14.35. SOC_ETM_TASK_ST3_CLR_REG (0x0204)

Continued from the previous page...

SOC_ETM_MCPWM0_TASK_TIMER0_PERIOD_UP_ST_CLR Configures whether to clear MCPWM0_TASK_TIMER0_PERIOD_UP.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_MCPWM0_TASK_TIMER1_PERIOD_UP_ST_CLR Configures whether to clear MCPWM0_TASK_TIMER1_PERIOD_UP.
0: Invalid. No effect
1: Clear
(WT)

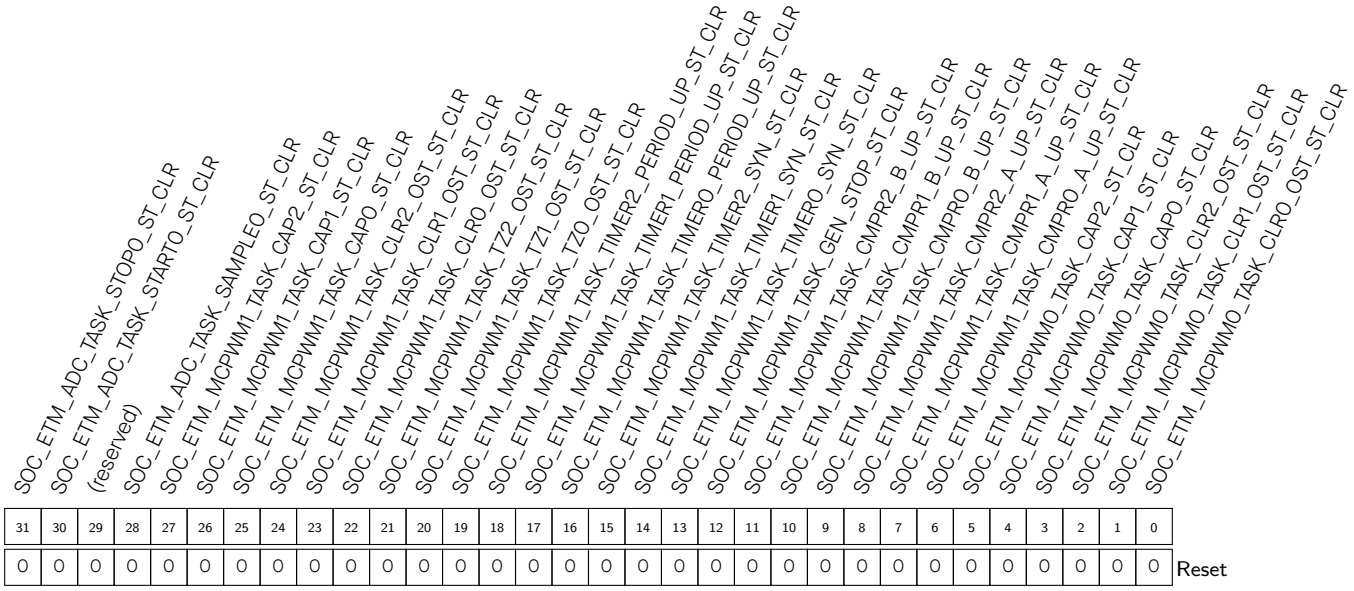
SOC_ETM_MCPWM0_TASK_TIMER2_PERIOD_UP_ST_CLR Configures whether to clear MCPWM0_TASK_TIMER2_PERIOD_UP.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_MCPWM0_TASK_TZ0_OST_ST_CLR Configures whether to clear MCPWM0_TASK_TZ0_OST.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_MCPWM0_TASK_TZ1_OST_ST_CLR Configures whether to clear MCPWM0_TASK_TZ1_OST.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_MCPWM0_TASK_TZ2_OST_ST_CLR Configures whether to clear MCPWM0_TASK_TZ2_OST.
0: Invalid. No effect
1: Clear
(WT)

Register 14.36. SOC_ETM_TASK_ST4_CLR_REG (0x020C)



SOC_ETM_MCPWM0_TASK_CLR0_OST_ST_CLR Configures whether to clear MCPWM0_TASK_CLR0_OST.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_MCPWM0_TASK_CLR1_OST_ST_CLR Configures whether to clear MCPWM0_TASK_CLR1_OST.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_MCPWM0_TASK_CLR2_OST_ST_CLR Configures whether to clear MCPWM0_TASK_CLR2_OST.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_MCPWM0_TASK_CAP0_ST_CLR Configures whether to clear MCPWM0_TASK_CAP0.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_MCPWM0_TASK_CAP1_ST_CLR Configures whether to clear MCPWM0_TASK_CAP1.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_MCPWM0_TASK_CAP2_ST_CLR Configures whether to clear MCPWM0_TASK_CAP2.
0: Invalid. No effect
1: Clear
(WT)

Continued on the next page...

Register 14.36. SOC_ETM_TASK_ST4_CLR_REG (0x020C)

Continued from the previous page...

SOC_ETM_MCPWM1_TASK_CMPR0_A_UP_ST_CLR Configures whether to clear MCPWM1_TASK_CMPR0_A_UP.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_MCPWM1_TASK_CMPR1_A_UP_ST_CLR Configures whether to clear MCPWM1_TASK_CMPR1_A_UP.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_MCPWM1_TASK_CMPR2_A_UP_ST_CLR Configures whether to clear MCPWM1_TASK_CMPR2_A_UP.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_MCPWM1_TASK_CMPR0_B_UP_ST_CLR Configures whether to clear MCPWM1_TASK_CMPR0_B_UP.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_MCPWM1_TASK_CMPR1_B_UP_ST_CLR Configures whether to clear MCPWM1_TASK_CMPR1_B_UP.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_MCPWM1_TASK_CMPR2_B_UP_ST_CLR Configures whether to clear MCPWM1_TASK_CMPR2_B_UP.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_MCPWM1_TASK_GEN_STOP_ST_CLR Configures whether to clear MCPWM1_TASK_GEN_STOP.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_MCPWM1_TASK_TIMER0_SYN_ST_CLR Configures whether to clear MCPWM1_TASK_TIMER0_SYN.
0: Invalid. No effect
1: Clear
(WT)

Continued on the next page...

Register 14.36. SOC_ETM_TASK_ST4_CLR_REG (0x020C)

Continued from the previous page...

SOC_ETM_MCPWM1_TASK_TIMER1_SYN_ST_CLR Configures whether to clear MCPWM1_TASK_TIMER1_SYN.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_MCPWM1_TASK_TIMER2_SYN_ST_CLR Configures whether to clear MCPWM1_TASK_TIMER2_SYN.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_MCPWM1_TASK_TIMER0_PERIOD_UP_ST_CLR Configures whether to clear MCPWM1_TASK_TIMER0_PERIOD_UP.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_MCPWM1_TASK_TIMER1_PERIOD_UP_ST_CLR Configures whether to clear MCPWM1_TASK_TIMER1_PERIOD_UP.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_MCPWM1_TASK_TIMER2_PERIOD_UP_ST_CLR Configures whether to clear MCPWM1_TASK_TIMER2_PERIOD_UP.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_MCPWM1_TASK_TZ0_OST_ST_CLR Configures whether to clear MCPWM1_TASK_TZ0_OST.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_MCPWM1_TASK_TZ1_OST_ST_CLR Configures whether to clear MCPWM1_TASK_TZ1_OST.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_MCPWM1_TASK_TZ2_OST_ST_CLR Configures whether to clear MCPWM1_TASK_TZ2_OST.
0: Invalid. No effect
1: Clear
(WT)

Continued on the next page...

Register 14.36. SOC_ETM_TASK_ST4_CLR_REG (0x020C)

Continued from the previous page...

SOC_ETM_MCPWM1_TASK_CLR0_OST_ST_CLR Configures whether to clear MCPWM1_TASK_CLR0_OST.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_MCPWM1_TASK_CLR1_OST_ST_CLR Configures whether to clear MCPWM1_TASK_CLR1_OST.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_MCPWM1_TASK_CLR2_OST_ST_CLR Configures whether to clear MCPWM1_TASK_CLR2_OST.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_MCPWM1_TASK_CAP0_ST_CLR Configures whether to clear MCPWM1_TASK_CAP0.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_MCPWM1_TASK_CAP1_ST_CLR Configures whether to clear MCPWM1_TASK_CAP1.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_MCPWM1_TASK_CAP2_ST_CLR Configures whether to clear MCPWM1_TASK_CAP2.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_ADC_TASK_SAMPLE0_ST_CLR Configures whether to clear ADC_TASK_SAMPLE0.
0: Invalid. No effect
1: Clear
(WT)

Continued on the next page...

Register 14.36. SOC_ETM_TASK_ST4_CLR_REG (0x020C)

Continued from the previous page...

SOC_ETM_ADC_TASK_START0_ST_CLR Configures whether to clear ADC_TASK_START0.

0: Invalid. No effect

1: Clear

(WT)

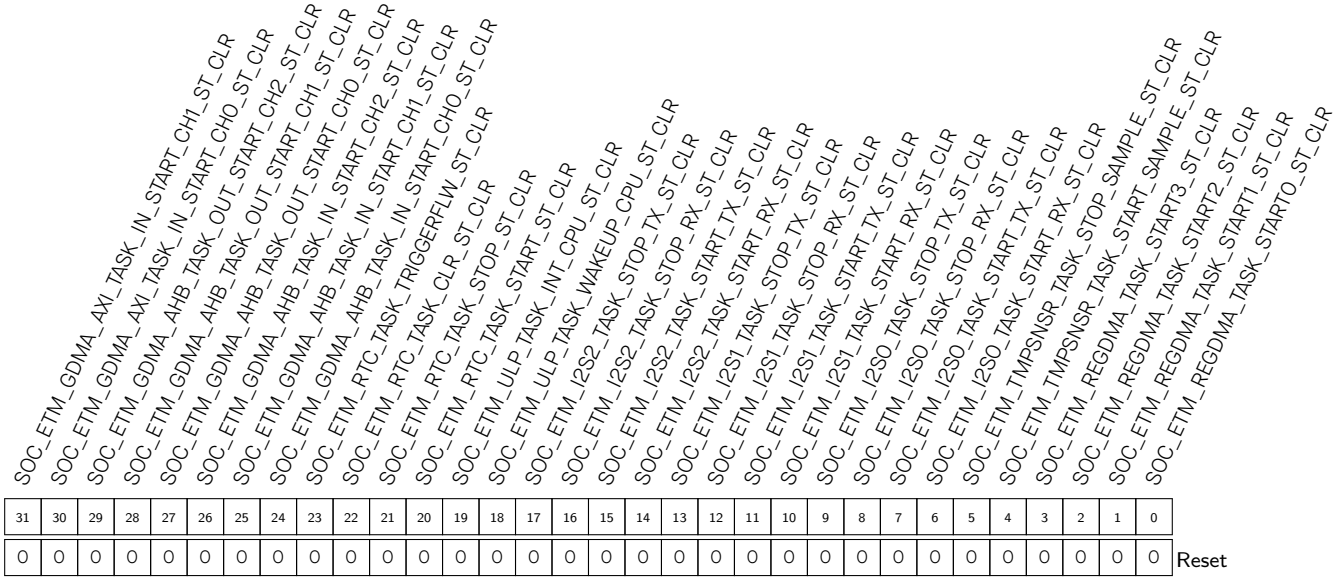
SOC_ETM_ADC_TASK_STOP0_ST_CLR Configures whether to clear ADC_TASK_STOP0.

0: Invalid. No effect

1: Clear

(WT)

Register 14.37. SOC_ETM_TASK_ST5_CLR_REG (0x0214)



SOC_ETM_REGDMA_TASK_START0_ST_CLR Configures whether to clear REGDMA_TASK_START0.

0: Invalid. No effect

1: Clear

(WT)

SOC_ETM_REGDMA_TASK_START1_ST_CLR Configures whether to clear REGDMA_TASK_START1.

0: Invalid. No effect

1: Clear

(WT)

SOC_ETM_REGDMA_TASK_START2_ST_CLR Configures whether to clear REGDMA_TASK_START2.

0: Invalid. No effect

1: Clear

(WT)

SOC_ETM_REGDMA_TASK_START3_ST_CLR Configures whether to clear REGDMA_TASK_START3.

0: Invalid. No effect

1: Clear

(WT)

SOC_ETM_TMPSNSR_TASK_START_SAMPLE_ST_CLR Configures whether to clear TMPSNSR_TASK_START_SAMPLE.

0: Invalid. No effect

1: Clear

(WT)

Continued on the next page...

Register 14.37. SOC_ETM_TASK_ST5_CLR_REG (0x0214)

Continued from the previous page...

SOC_ETM_TMPSNSR_TASK_STOP_SAMPLE_ST_CLR Configures whether to clear TMPSNSR_TASK_STOP_SAMPLE.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_I2S0_TASK_START_RX_ST_CLR Configures whether to clear I2S0_TASK_START_RX.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_I2S0_TASK_START_TX_ST_CLR Configures whether to clear I2S0_TASK_START_TX.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_I2S0_TASK_STOP_RX_ST_CLR Configures whether to clear I2S0_TASK_STOP_RX.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_I2S0_TASK_STOP_TX_ST_CLR Configures whether to clear I2S0_TASK_STOP_TX.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_I2S1_TASK_START_RX_ST_CLR Configures whether to clear I2S1_TASK_START_RX.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_I2S1_TASK_START_TX_ST_CLR Configures whether to clear I2S1_TASK_START_TX.
0: Invalid. No effect
1: Clear
(WT)

Continued on the next page...

Register 14.37. SOC_ETM_TASK_ST5_CLR_REG (0x0214)

Continued from the previous page...

SOC_ETM_I2S1_TASK_STOP_RX_ST_CLR Configures whether to clear I2S1_TASK_STOP_RX.

0: Invalid. No effect

1: Clear

(WT)

SOC_ETM_I2S1_TASK_STOP_TX_ST_CLR Configures whether to clear I2S1_TASK_STOP_TX.

0: Invalid. No effect

1: Clear

(WT)

SOC_ETM_I2S2_TASK_START_RX_ST_CLR Configures whether to clear I2S2_TASK_START_RX.

0: Invalid. No effect

1: Clear

(WT)

SOC_ETM_I2S2_TASK_START_TX_ST_CLR Configures whether to clear I2S2_TASK_START_TX.

0: Invalid. No effect

1: Clear

(WT)

SOC_ETM_I2S2_TASK_STOP_RX_ST_CLR Configures whether to clear I2S2_TASK_STOP_RX.

0: Invalid. No effect

1: Clear

(WT)

SOC_ETM_I2S2_TASK_STOP_TX_ST_CLR Configures whether to clear I2S2_TASK_STOP_TX.

0: Invalid. No effect

1: Clear

(WT)

SOC_ETM_ULP_TASK_WAKEUP_CPU_ST_CLR Configures whether to clear ULP_TASK_WAKEUP_CPU.

0: Invalid. No effect

1: Clear

(WT)

Continued on the next page...

Register 14.37. SOC_ETM_TASK_ST5_CLR_REG (0x0214)

Continued from the previous page...

SOC_ETM_ULP_TASK_INT_CPU_ST_CLR Configures whether to clear ULP_TASK_INT_CPU.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_RTC_TASK_START_ST_CLR Configures whether to clear RTC_TASK_START.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_RTC_TASK_STOP_ST_CLR Configures whether to clear RTC_TASK_STOP.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_RTC_TASK_CLR_ST_CLR Configures whether to clear RTC_TASK_CLR.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_RTC_TASK_TRIGGERFLW_ST_CLR Configures whether to clear RTC_TASK_TRIGGERFLW.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_GDMA_AHB_TASK_IN_START_CH0_ST_CLR Configures whether to clear GDMA_AHB_TASK_IN_START_CH0.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_GDMA_AHB_TASK_IN_START_CH1_ST_CLR Configures whether to clear GDMA_AHB_TASK_IN_START_CH1.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_GDMA_AHB_TASK_IN_START_CH2_ST_CLR Configures whether to clear GDMA_AHB_TASK_IN_START_CH2.
0: Invalid. No effect
1: Clear
(WT)

Continued on the next page...

Register 14.37. SOC_ETM_TASK_ST5_CLR_REG (0x0214)

Continued from the previous page...

SOC_ETM_GDMA_AHB_TASK_OUT_START_CH0_ST_CLR Configures whether to clear GDMA_AHB_TASK_OUT_START_CH0.
0: Invalid. No effect
1: Clear
(WT)

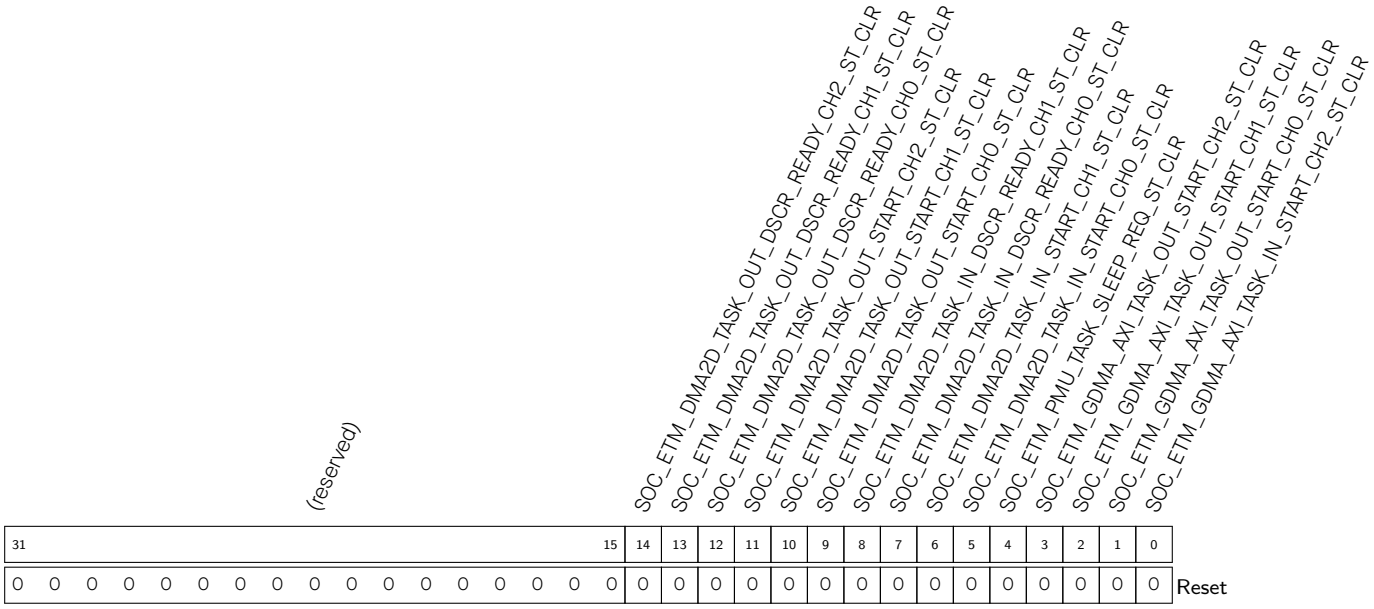
SOC_ETM_GDMA_AHB_TASK_OUT_START_CH1_ST_CLR Configures whether to clear GDMA_AHB_TASK_OUT_START_CH1.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_GDMA_AHB_TASK_OUT_START_CH2_ST_CLR Configures whether to clear GDMA_AHB_TASK_OUT_START_CH2.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_GDMA_AXI_TASK_IN_START_CH0_ST_CLR Configures whether to clear GDMA_AXI_TASK_IN_START_CH0.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_GDMA_AXI_TASK_IN_START_CH1_ST_CLR Configures whether to clear GDMA_AXI_TASK_IN_START_CH1.
0: Invalid. No effect
1: Clear
(WT)

Register 14.38. SOC_ETM_TASK_ST6_CLR_REG (0x021C)



SOC_ETM_GDMA_AXI_TASK_IN_START_CH2_ST_CLR Configures whether to clear GDMA_AXI_TASK_IN_START_CH2.

0: Invalid. No effect

1: Clear

(WT)

SOC_ETM_GDMA_AXI_TASK_OUT_START_CH0_ST_CLR Configures whether to clear GDMA_AXI_TASK_OUT_START_CH0.

0: Invalid. No effect

1: Clear

(WT)

SOC_ETM_GDMA_AXI_TASK_OUT_START_CH1_ST_CLR Configures whether to clear GDMA_AXI_TASK_OUT_START_CH1.

0: Invalid. No effect

1: Clear

(WT)

SOC_ETM_GDMA_AXI_TASK_OUT_START_CH2_ST_CLR Configures whether to clear GDMA_AXI_TASK_OUT_START_CH2.

0: Invalid. No effect

1: Clear

(WT)

SOC_ETM_PMU_TASK_SLEEP_REQ_ST_CLR Configures whether to clear PMU_TASK_SLEEP_REQ.

0: Invalid. No effect

1: Clear

(WT)

Continued on the next page...

Register 14.38. SOC_ETM_TASK_ST6_CLR_REG (0x021C)

Continued from the previous page...

SOC_ETM_DMA2D_TASK_IN_START_CH0_ST_CLR Configures whether to clear DMA2D_TASK_IN_START_CH0.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_DMA2D_TASK_IN_START_CH1_ST_CLR Configures whether to clear DMA2D_TASK_IN_START_CH1.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_DMA2D_TASK_IN_DSCR_READY_CH0_ST_CLR Configures whether to clear DMA2D_TASK_IN_DSCR_READY_CH0.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_DMA2D_TASK_IN_DSCR_READY_CH1_ST_CLR Configures whether to clear DMA2D_TASK_IN_DSCR_READY_CH1.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_DMA2D_TASK_OUT_START_CH0_ST_CLR Configures whether to clear DMA2D_TASK_OUT_START_CH0.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_DMA2D_TASK_OUT_START_CH1_ST_CLR Configures whether to clear DMA2D_TASK_OUT_START_CH1.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_DMA2D_TASK_OUT_START_CH2_ST_CLR Configures whether to clear DMA2D_TASK_OUT_START_CH2.
0: Invalid. No effect
1: Clear
(WT)

Continued on the next page...

Register 14.38. SOC_ETM_TASK_ST6_CLR_REG (0x021C)

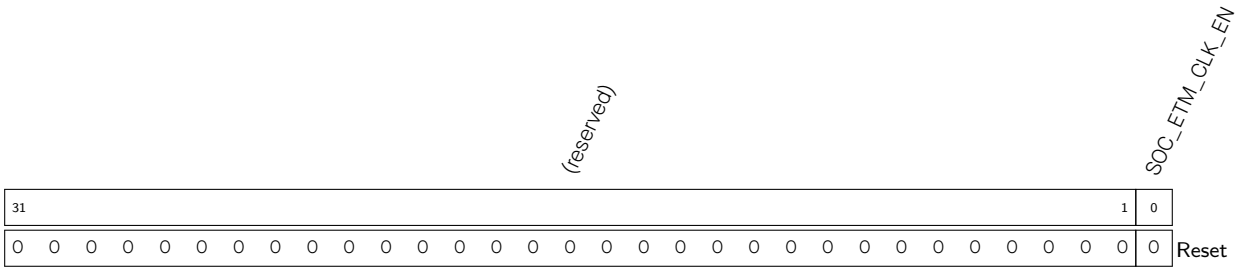
Continued from the previous page...

SOC_ETM_DMA2D_TASK_OUT_DSCR_READY_CH0_ST_CLR Configures whether to clear DMA2D_TASK_OUT_DSCR_READY_CH0.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_DMA2D_TASK_OUT_DSCR_READY_CH1_ST_CLR Configures whether to clear DMA2D_TASK_OUT_DSCR_READY_CH1.
0: Invalid. No effect
1: Clear
(WT)

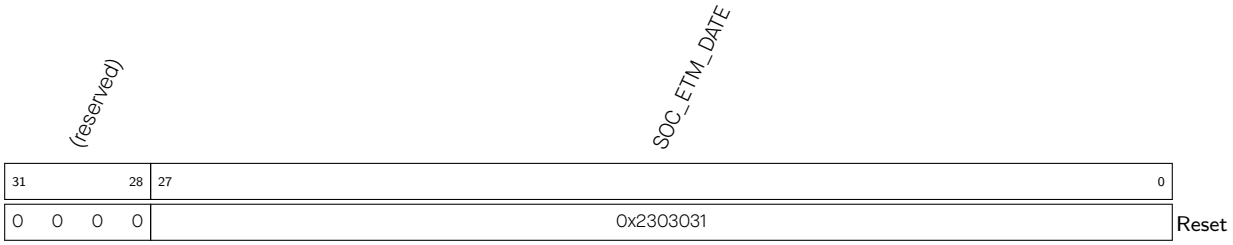
SOC_ETM_DMA2D_TASK_OUT_DSCR_READY_CH2_ST_CLR Configures whether to clear DMA2D_TASK_OUT_DSCR_READY_CH2.
0: Invalid. No effect
1: Clear
(WT)

Register 14.39. SOC_ETM_CLK_EN_REG (0x0220)



SOC_ETM_CLK_EN Configures resister clock gating.
0: Support clock only when application writes registers
1: Force on clock gating for registers
(R/W)

Register 14.40. SOC_ETM_DATE_REG (0x0224)



SOC_ETM_DATE Version control register. (R/W)

Chapter 15

Low-Power Management

15.1 Overview

ESP32-P4 features an advanced low-power management system that optimizes the chip's power consumption while maintaining its high performance.

The low-power management system employs various power-saving techniques such as sleep modes, dynamic voltage and frequency scaling, and peripheral power gating to minimize the chip's power consumption.

At the core of this low-power management system is the power management unit (PMU) hardware component. The PMU powers up and down different power domains of the chip to achieve the best balance between chip performance, power consumption, and wake-up latency.

15.2 Terminology

The following terms related to low-power management are defined in the context of the *ESP32-P4 Technical Reference Manual* to help readers better understand this document:

Low-power management	Refers to the whole system that manages the chip's power consumption.
Power management unit (PMU)	Refers to the specific hardware module that controls power up and down for power domains, clocks, and power-related logic.
Power domain	Refers to the smallest unit that can be independently powered up or down. A power domain can contain one or multiple modules within the chip.
PMU states	Refers to the states of the PMU's state machine. Users can configure the clock gating and power gating of a power domain in each of the states.
Power modes	Refers to the preset power modes that power up different domains for typical application scenarios.

15.3 Features

The PMU has the following features:

- Supports three configurable PMU states. Software can flexibly configure them according to the needs.
  - HP_ACTIVE
  - HP_SLEEP

  - LP_SLEEP

- Supports five preset power modes that suit various typical usage scenarios:
  - Active
  - Light-sleep0 (This mode does not power down the peripherals, but only the CPU. The clocks are not turned off. The power consumption is high, and the wake-up latency is short.)
  - Light-sleep1 (This mode does not power down the peripherals, but only turns off the clocks. The power consumption is relatively higher and the wake-up latency shorter.)
  - Light-sleep2 (This mode powers down the peripherals and needs data backup and restoration, resulting in lower power consumption and a longer wake-up latency.)
  - Deep-sleep
- LP SRAM (32 KB): Keep power up in all modes, can store HP CPU's non-volatile data, and LP CPU's instructions and data.
- Supports a programmable retention DMA that backs up and restores the CPU and peripherals status when the chip switches between PMU states.
- Supports a power controller that controls the power and clocks depending on the power modes.

15.4 Functional Description

ESP32-P4's low-power management involves the following components:

- **Power scheme**: The power scheme of ESP32-P4 includes power regulators, digital power domains, analog power domains, etc.
- **PMU controller**: It is the core part of the PMU, controlling the power up and down of power domains, clocks, etc.
- **One RTC timer**: For more information, please refer to Chapter 19 *RTC Timer*.
- **16 LP GPIO pins (GPIO0–GPIO15)**: These pins are always powered up and are not affected by any low-power modes, which makes them suitable for working as wake-up sources when the chip is in low-power modes. These pins can also work as regular GPIOs. For more information on LP GPIOs, refer to Chapter 10 *GPIO Matrix and IO MUX*.
- **32 KB SRAM**: The 32 KB SRAM is accessible to both HP CPU and LP CPU. It works under the HP CPU clock when accessed by the HP CPU and under the LP CPU clock when accessed by the LP CPU.
- **Brownout detector**: Monitors the power of the voltage supply pins, ensuring stable chip operation and preventing the SoC from potential malfunction if it is subject to glitches or under-voltage. For more information, refer to Chapter 25 *Brown-out Detector*.

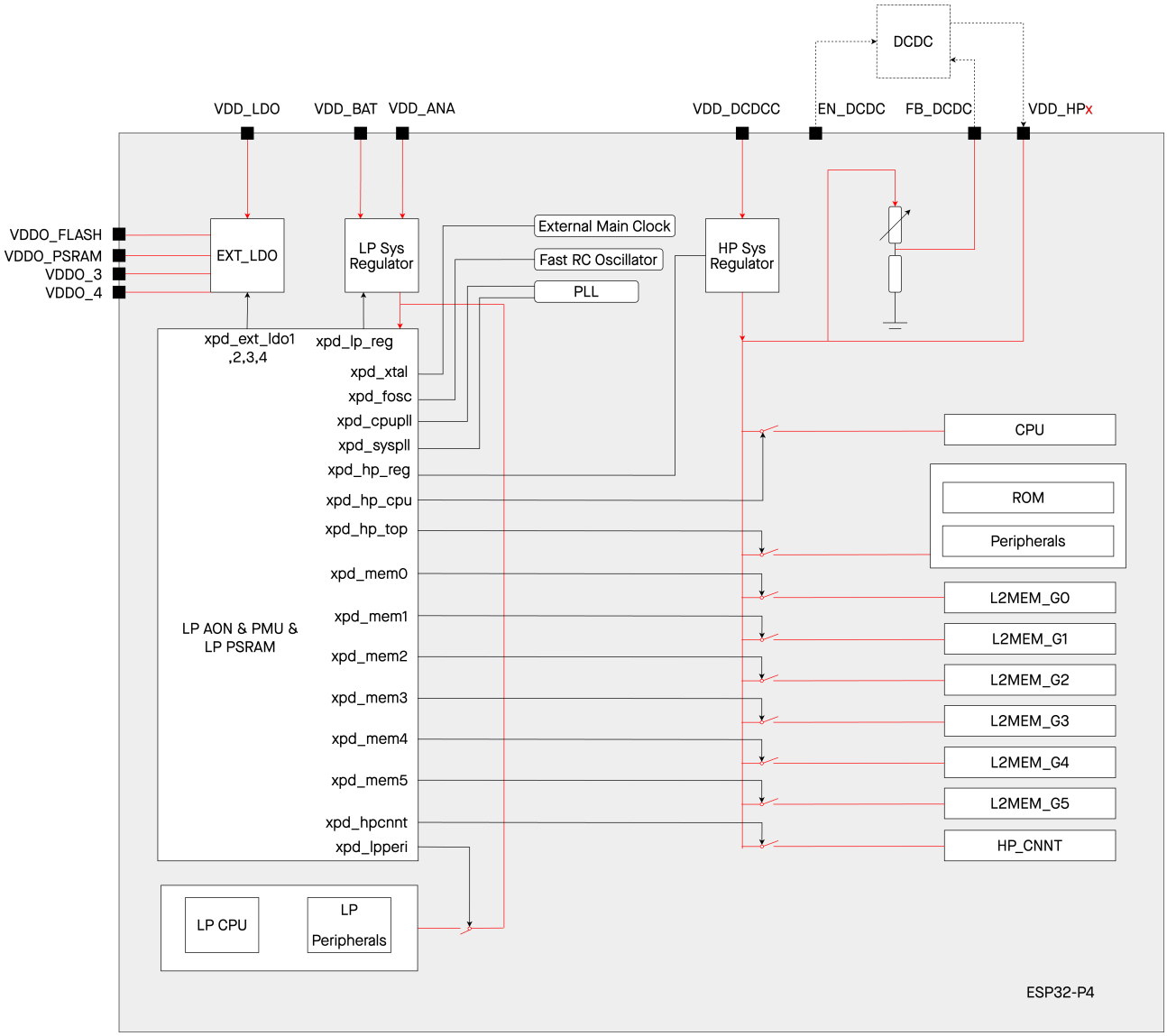
The following sections provide a detailed description of the components mentioned above.

15.4.1 Power Scheme

Figure 15.4-1 shows the power scheme of ESP32-P4 that mainly includes:

- A DCDC voltage regulation feedback system

- Two internal regulators
- Four output regulators (EXT_LDO)
- Analog power domains
- Digital power domains
- Battery power domain



–→: Control signals ––: Power lines

Figure 15.4-1. ESP32-P4 Power Scheme

15.4.1.1 Regulators

As shown in Figure 15.4-1, the analog section of ESP32-P4 includes one external DCDC with feedback regulation and two internal HP/LP system regulators, responsible for regulating the power supply for different power domains.

- The **DCDC voltage regulation feedback system** adjusts the voltage of the external DCDC.

- The **HP (high-performance) system regulator** has strong drive capability, higher power consumption, and adjustable output voltage. It is only suitable for power-up and sleep modes.
- The **LP (low-power) system regulator** regulates the power supply of low-power modules. Its output voltage is also adjustable.

The DCDC voltage regulation feedback system and the HP system regulator are responsible for powering the HP system. The external DCDC can adjust its output voltage through the internal voltage feedback system. By default, the chip is powered by the HP system regulator when powered up. After power-up is complete, it is recommended to switch to the DCDC power supply for better efficiency and load capacity.

15.4.1.2 Digital Power Domains

- The HP system contains the following digital power domains:
  - **CPU (PD_HP_CPU)**: Mainly includes the CPU and peripherals that support its operation.
  - **Peripherals + ROM (PD_TOP)**: Mainly includes the bus and HP peripherals.
  - **L2MEM_G0–G5**: It is divided into six sub-power domains, as shown in Figure 15.4-1.
  - **HP_CNNT (PD_HP_CNNT)**: Mainly includes USB and SDIO modules.

  There is an independent power switch between the regulator and each power domain, enabling up/down control of the digital power domain.
- The LP system contains the following digital power domains:
  - **LP PD peripherals (PD_LP_PERI)**: Mainly includes LP CPU and LP peripherals.
  - **LP always-on (PD_AON)**: It mainly includes LP always-on peripherals (e.g., RTC timer) and PMU controller. This power domain keeps powered up all the time.

15.4.1.3 Analog Power Domains

- The HP system contains the following analog power domains:
  - CPLL_CLK
  - SPLL_CLK
  - Audio PLL
  - SDIO PLL
- The LP system contains the following analog power domains:
  - External Main Clock
  - Fast RC Oscillator

15.4.1.4 Battery Power Domain

The battery power domain (VDD_BAT) serves as a backup power domain. When all external power supplies (VDD_ANA) are turned off, the low-power system regulators, LP_SLOW_CLK, and LP_FAST_CLK can all be powered by the VDD_BAT supply.

15.4.2 PMU

The PMU of ESP32-P4 controls power consumption-related components of each power domain, such as power and clock. PMU consists of the following major parts:

- **PMU main state machine**: It records and switches the PMU states.
- **Sleep/wake-up controller**: It sends sleep or wake-up requests to the PMU main state machine.
- **Power controllers**: They control the power and clock signals depending on the power modes of the chip. The power controllers include:
  - **Digital power controller**: It powers up or down the digital power domains.
  - **Analog power controller**: It enables the analog modules, such as the regulators, analog clocks, etc.
  - **Clock controller**: It manages the clock gating of peripherals and selects analog clock sources for digital clocks.
  - **Data backup controller**: It controls the data backup and restore process when the chip switches between PMU states.
  - **System controller**: It controls some system-level modules, such as suspending watchdog functionality in sleep mode (when the CPU is unavailable).

The PMU workflow involves the following steps:

- The sleep/wake-up controller sends sleep or wake-up requests to the PMU main state machine.
- The PMU main state machine generates power gating, clock gating, and reset signals.
- The power controllers and clock controllers power up or down different power domains and clocks based on the signals generated by the PMU main state machine, allowing the chip to enter or exit different low-power modes.

The PMU workflow is shown in Figure 15.4-2. The definitions of HP_SWITCH and LP_SWITCH are as follows:

- HP_SWITCH: The intermediate state for the HP system transitioning between HP_ACTIVE and HP_SLEEP. The hardware completes the control switching of various controllers in this state.
- LP_SWITCH: The intermediate state for the LP system transitioning between HP_SLEEP and LP_SLEEP. The hardware completes the control switching of various controllers in this state.

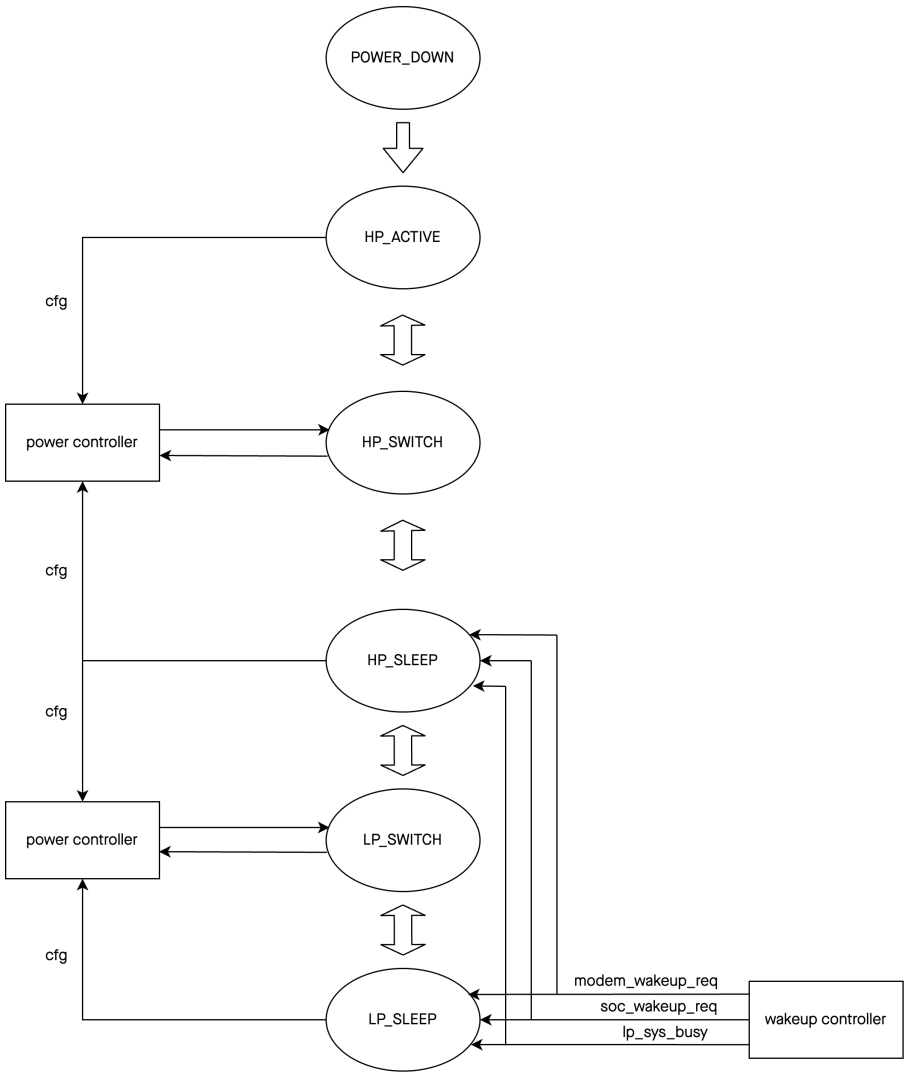


Figure 15.4-2. PMU Workflow

The following sections describe the main parts of PMU.

15.4.2.1 PMU Main State Machine

The PMU main state machine receives sleep and wake-up signals, changes the state of power and clock through the power controllers, thereby switching PMU states, and achieving a balance between performance and power consumption of the chip.

The PMU main state machine supports three PMU states, each controlled by different sleep and wake-up signals, supporting software customization of power and clocks. These PMU states allow the software to expand power modes for various application scenarios. The three PMU states are:

- **HP_ACTIVE**: The circuits on the chip are powered up to a maximum, supporting the HP system and LP system operation.
- **HP_SLEEP**: The HP system is in sleep, supporting LP peripherals operation.
- **LP_SLEEP**: The HP system and LP peripherals are in sleep, while the always-on circuits remain operational.

Note:
The division of the HP and LP systems is as follows:

- LP system (peripherals): LP RISC-V 32-bit Microprocessor, LP GPIO, LP UART, LP I2C, LP I2S, touch sensor, temperature sensor, LP ADC, LP SPI
- LP system (always-on circuits): PMU, RTC Watchdog Timer, Super Watchdog
- HP system: All peripherals except those belonging to the LP system.

For more details, please refer to *ESP32-P4 Datasheet* > *Functional Block Diagram*.

HP_ACTIVE is the operating state of the HP system, HP_SLEEP is the sleep state of the HP system, LP_SLEEP is the sleep state of the LP system. The operating state of the LP system reuses the HP_SLEEP state and is therefore also referred to as HP_SLEEP.

- If a module belongs to the HP system, its power up/down can be configured in the HP_ACTIVE/HP_SLEEP states. It reuses the HP_SLEEP state in the LP_SLEEP state.
- If a module belongs to the LP system, its power up/down can be configured in the HP_SLEEP/LP_SLEEP state. It reuses the HP_SLEEP state in the HP_ACTIVE state.

Take the HP CPU as an example. The HP CPU belongs to the HP system, so it can be configured to power up/down in the HP_ACTIVE/HP_SLEEP states through the following registers and reuse the HP_SLEEP configurations in LP_SLEEP.

- HP_ACTIVE: PMU_HP_ACTIVE_PD_HP_CPU_PD_EN
- HP_SLEEP: PMU_HP_SLEEP_PD_HP_CPU_PD_EN

Similarly, users can configure the power up and down for other power domains in different PMU states. For the configuration registers, please refer to Section 15.8.

Note:
In the following text, all such configuration registers are collectively referred to as PMU_*n1*_PD_*POWERDOMAIN*_PD_EN, where *n1* represents the three PMU states.

Once the configuration is done, the PMU uses various controllers to make these configurations effective, as described in the following sections.

15.4.2.2 Sleep/Wake-up Controller

The sleep/wake-up controller initiates sleep and wake-up requests to the PMU main state machine. ESP32-P4 supports multiple wake-up sources to wake the CPU from different power modes. Table 15.4-1 lists all the wake-up sources.

All the wake-up sources in Table 15.4-1 can wake up both the HP CPU and the LP CPU. Users can configure PMU_WAKEUP_ENA to set the target to the HP CPU, and configure PMU_LP_CPU_WAKEUP_EN to set the target to the LP CPU. Configuring both registers simultaneously can wake up both the HP CPU and the LP CPU.

Table 15.4-1. Wake-up Sources

Bit in PMU_WAKEUP_ENA/PMU_LP_CPU_WAKEUP_EN	Wake-up Sources	Source Power Domains
0	HP_SDIO	PD_HP_CNNT
1	Software wakeup	PD_LP_PERI
2	HP GPIO	PD_TOP
3	HP USB	PD_HP_CNNT
4	HP UART4	PD_TOP
5	HP UART3	PD_TOP
6	HP UART2	PD_TOP
7	HP UART1	PD_TOP
8	HP UART0	PD_TOP
9	LP GPIO	PD_LP_PERI
10	LP UART	PD_LP_PERI
11	LP TOUCH	PD_LP_PERI
12	EXT IO	PD_LP_PERI
13	RTC_TIMER_TAR0	PD_AON
14	BROWN OUT	PD_AON
15	VBAT_UNDERVOLTAGE	PD_AON
16	LP CORE EXCEPTION	PD_LP_PERI
17	ETM	PD_TOP
18	RTC_TIMER_TAR1	PD_AON
19	LP I2S	PD_LP_PERI
20	GMAC PMT	PD_HP_CNNT
21	GMAC LPI	PD_HP_CNNT
22	LP CPU	PD_AON
23	HP CPU	PD_TOP

ESP32-P4 provides a hardware mechanism that can reject sleep, meaning if some peripherals are in an uninterruptible working state and the HP CPU tries to sleep, the peripherals will send a wake-up signal to prevent the HP CPU from sleeping, thus ensuring the peripherals work normally.

The wake-up sources in Table 15.4-1 can all be configured as events to reject sleep. Users can configure the following registers to implement sleep rejection. The configuration values of PMU_SLEEP_REJECT_ENA and PMU_SLP_REJECT_CAUSE_REG and the corresponding wake-up sources are the same as shown in Table 15.4-1.

- Enable sleep rejection feature:
  - Set PMU_SLP_REJECT_EN to 1 to enable the sleep rejection feature.
  - Configure PMU_SLEEP_REJECT_ENA to select the sleep rejection signal source.
- Read PMU_SLP_REJECT_CAUSE_REG for the source of sleep rejection event.

15.4.2.3 Analog Power Controller

The analog power controller controls the power up and down of the analog circuits (including voltage regulators, high-speed clocks, and slow-speed clocks) in the PMU states.

The configuration of the regulators is as follows:

- Configure PMU_*n1*_HP_REGULATOR_XPD or PMU_*n1*_LP_REGULATOR_XPD to enable or disable the output voltage of the HP/LP sys regulator in the target PMU state. Turning off the LP sys regulator is not recommended, as it may cause the PMU itself to power down, resulting in chip malfunction.

The configuration of the high-speed clocks (XTAL_CLK and PLL clocks) is as follows:

- XTAL_CLK: Configure PMU_HP_SLEEP_XPD_XTAL to 1 to enable XTAL_CLK when the chip switches the PMU state to HP_SLEEP.

  Note: To avoid the instability in XTAL_CLK during startup, users have the option to configure PMU_WAIT_XTL_STABLE to delay the gate opening for XTAL_CLK. This delay ensures that the gate opening is enabled after PMU_WAIT_XTL_STABLE CLK_DYN_FAST_CLK cycles following the power-up of XTAL_CLK.

- CPLL_CLK: PMU can enable CPLL_CLK in different PMU states by configuring PMU_*n1*_XPD_PLL[0] to 1. For example, configuring PMU_HP_ACTIVE_XPD_PLL[0] to 1 enables the CPLL_CLK clock when the chip is in HP_ACTIVE state.
- SPLL_CLK: PMU can enable SPLL_CLK in different PMU states by configuring PMU_*n1*_XPD_PLL[1] to 1. For example, configuring PMU_HP_ACTIVE_XPD_PLL[1] to 1 enables the SPLL_CLK clock when the chip is in HP_ACTIVE state.
- Audio PLL: PMU can enable Audio PLL in different PMU states by configuring PMU_*n1*_XPD_PLL[2] to 1. For example, configuring PMU_HP_ACTIVE_XPD_PLL[2] to 1 enables the Audio PLL clock when the chip is in HP_ACTIVE state.
- SDIO PLL: PMU can enable SDIO PLL in different PMU states by configuring PMU_*n1*_XPD_PLL[3] to 1. For example, configuring PMU_HP_ACTIVE_XPD_PLL[3] to 1 enables the SDIO PLL clock when the chip is in HP_ACTIVE state.

  Note:

  - Before enabling the PLL clocks please ensure that XTAL_CLK is stable.
  - To avoid the instability in the PLL clocks during startup, users have the option to configure PMU_WAIT_PLL_STABLE to delay the gate opening for the PLL clocks. This delay ensures that the gate opening is enabled after PMU_WAIT_PLL_STABLE CLK_DYN_FAST_CLK cycles following the power-up of the PLL clocks.

Slow-speed clocks (RC_FAST_CLK and XTAL32K_CLK) operate with low power. The power up and down of the slow-speed clocks in HP_ACTIVE and HP_SLEEP states are controlled by PMU_HP_SLEEP_XPD_FOSC_CLK. In LP_SLEEP state, the power up and down of the slow-speed clocks are controlled by PMU_LP_SLEEP_XPD_FOSC_CLK.

15.4.2.4 Digital Power Controller

The digital power controller controls the power up and down of digital power domains in different PMU states. Unlike the analog power controller, the digital power controller does not directly control the regulator but instead controls the power switch connected to the regulator to power up and down the digital power domains.

The LP PD Peripherals domain can only be powered up and down while the PMU state switches between

HP_SLEEP and LP_SLEEP. The other digital power domains can only be powered up and down while the PMU state switches between HP_SLEEP and HP_ACTIVE.

When the chip switches between PMU states, if the power configuration of a power domain in the current PMU state does not match the power configuration it is about to switch to, then the power up-down process will be activated. Take the power up-to-down process of the CPU power domain as an example. Configure PMU_HP_SLEEP_PD_TOP_PD_EN to 1 or 0 to power down or up the CPU power domain in HP_SLEEP. PMU will perform the following configurations:

- Enable the digital isolation unit to ensure that the powered-down modules do not output unstable voltage levels to the powered-up modules. When a power domain loses power, the output of this module will be clamped to a fixed value.
- Enable reset. When the CPU power domain loses power, its global reset signal is set to a reset state, which persists for a period after the CPU power domain is re-powered. This mechanism guarantees a reset-to-release process for the CPU power domain during power-up, effectively mitigating any instability caused by power up-down transitions.

The following explains the power up and down of each digital power domain:

- L2MEM_G0–G5

  Configure PMU_*n1*_PD_HP_MEM_PD_EN to power up or down L2MEM_G0–G5.

- HP_CNNT

  Configure PMU_*n1*_PD_CNNT_PD_EN to power up or down HP_CNNT.

- PD_HP_CPU

  Configure PMU_*n1*_PD_HP_CPU_PD_EN to power up or down PD_HP_CPU.

- Peripherals + ROM (PD_TOP)

  Configure PMU_*n1*_PD_TOP_PD_EN to power or down PD_TOP.

  When the Peripherals domain is powered down, the following features are configurable:

  - Powering down the Peripherals domain may cause instability in GPIOs. This can be addressed by maintaining the state of the GPIOs (excluding the LP GPIOs) through PMU. For example, configuring PMU_HP_SLEEP_HP_PAD_HOLD_ALL can keep GPIOs in the same state as before Peripherals was powered down in HP_SLEEP.
  - In the HP_ACTIVE state, Memory Deep-sleep mode is supported under a standard power supply. The memory cannot be read or written in this mode, but data can be retained. Configure PMU_*n1*_HP_MEM_DSLP to enable this mode.

- LP PD Peripherals

  The LP PD Peripherals power domain remains powered up when the chip is in HP_ACTIVE state. The power state of LP PD Peripherals is configurable only in HP_SLEEP and LP_SLEEP states. The LP PD Peripherals power domain has an independent digital power switch, and its power control is not dependent on the power state of other digital power domains.

15.4.2.5 Clock Controller

The clock controller mainly controls the high-performance system clocks and LP system clocks when the chip switches between PMU states.

The high-performance system clocks include ROOT_CLK and high-performance system peripherals clocks. When the chip switches between HP_ACTIVE and HP_SLEEP states, PMU can switch, power up/down, and divide the frequency of ROOT_CLK, as well as power up/down high-performance system peripherals clocks.

- ROOT_CLK can be controlled as follows:
  - Configure PMU_*n1*_SYS_CLK_SLP_SEL to 1 to indicate that when the chip enters the corresponding PMU state, the clock source is controlled by PMU.
  - Configure PMU_*n1*_ICG_SYS_CLOCK_EN to 0 to disable ROOT_CLK in the corresponding PMU state.
  - Configure PMU_*n1*_DIG_SYS_CLK_SEL to select the clock source after the chip enters the corresponding PMU state. For details, please refer to Chapter 11 *Reset and Clock* > Table 11.2-1.
- High-performance system peripheral clocks can be controlled as follows:
  - Configure PMU_*n1*_ICG_SLP_SEL to 1 so that the clock gating in the target state is controlled by PMU. Configure this register to 0 so that the clock gating is controlled by PCR registers.
  - Configure PMU_*n1*_DIG_ICG_FUNC_EN to power up/down the function clock in the target PMU state. For detailed configuration please see Table 15.4-2.

Table 15.4-2. HP System Peripherals' Function Clocks

Bit in PMU_*n1*_DIG_ICG_FUNC_EN	Clock
bit 0	HP_CPU0_CLK
bit 1	HP_CPU1_CLK
bit 2	HP_CPU0_CLIC_CLK
bit 3	HP_CPU1_CLIC_CLK
bit 4	MISC_CPU_CLK
bit 5	MISC_SYS_CLK
bit 6	ICM_CPU_CLK
bit 7	ICM_SYS_CLK
bit 8	ICM_MEM_CLK
bit 9	ICM_APB_CLK
bit 10	SPM_CPU_CLK
bit 11	L2MEM_MEM_CLK
bit 12	L2MEM_SYS_CLK
bit 13	L1CACHE_CPU_CLK
bit 14	L1CACHE_D_CPU_CLK
bit 15	L1CACHE_I0_CPU_CLK
bit 16	L1CACHE_I1_CPU_CLK
bit 17	L1CACHE_MEM_CLK
bit 18	L1CACHE_D_MEM_CLK
bit 19	L1CACHE_I0_MEM_CLK

PMU_*n1*_DIG_ICG_FUNC_EN Bit	Clock
bit 20	L1CACHE_I1_MEM_CLK
bit 21	L2CACHE_MEM_CLK
bit 22	L2CACHE_SYS_CLK
bit 23	REGDMA_SYS_CLK
bit 24	HP_CLKRST_APB_CLK
bit 25	SYSREG_APB_CLK
bit 26	INTRMTX_CLK
bit 27	N/A
bit 28	N/A
bit 29	N/A
bit 30	N/A
bit 31	N/A

The LP system clocks are mainly used in the low-power system and include the following clocks:

- LP_SLOW_CLK
- LP_FAST_CLK
- LP_DYN_SLOW_CLK
- LP_DYN_FAST_CLK
- XTAL_D2_CLK

The clock frequency of LP_DYN_FAST_CLK is controlled by hardware as follows, depending on the PMU state (and cannot be changed by the user):

- LP_SLEEP: The LP_DYN_FAST_CLK frequency is the same as LP_SLOW_CLK.
- HP_ACTIVE, HP_SLEEP: The LP_DYN_FAST_CLK frequency is the same as LP_FAST_CLK.

15.4.2.6 Backup Controller

ESP32-P4 has a Retention DMA module that can transfer data between memory and peripherals when the chip switches between PMU states. In this way, the data is backed up when the power domain is powered down and restored when the power domain is powered up again.

Data transfer is implemented in the Peripherals power domain. PMU only generates relevant control signals. It is important to note that the data transfer control registers are directional, as unlike other control registers, these control behaviors are determined by both the original PMU state and the target PMU state.

For example, the control registers for transitioning from HP_ACTIVE to HP_SLEEP and from HP_SLEEP to HP_ACTIVE are different. The possible PMU state switches are listed below, collectively represented by *n2* in the register names:

- HP_SLEEP2ACTIVE
- HP_ACTIVE2SLEEP

The following section introduces how PMU controls the Retention DMA:

- Enable data transfer: Configure PMU_*n2*_BACKUP_EN to 1 to enable data transfer when the corresponding PMU state switch is performed.
- Enable corresponding clocks: Before data transfer starts, configure PMU_*n2*_BACKUP_CLK_SEL to select the clock source of the Retention DMA, and configure PMU_*n1*_BACKUP_ICG_FUNC_EN to enable the clock.

  After the data transfer is completed, the value of PMU_*n1*_BACKUP_ICG_FUNC_EN is determined by the configuration of PMU_*n1*_DIG_ICG_FUNC_EN in the target PMU state.
- Configure data transfer direction: Configure the highest bit of PMU_*n2*_BACKUP_MODE:
  - 1: From peripheral to memory
  - 0: From memory to peripheral
- Select linked list pointer: Configure the lower two bits of PMU_*n2*_BACKUP_MODE to select the linked list pointer, specifically:
  - 0: PAU_LINK_ADDR_0
  - 1: PAU_LINK_ADDR_1
  - 2: PAU_LINK_ADDR_2
  - 3: PAU_LINK_ADDR_3

15.4.2.7 System Controller

The system controller controls some functional modules when the chip switches PMU states, to achieve stable and low-power chip performance. Specifically, the system controller supports:

- **Pausing the watchdog function**: Configuring PMU_*n1*_DIG_PAUSE_WDT to 1 disables the RTC watchdog timer (RWDT) function when the chip switches to the corresponding target PMU state. Note that if this register is configured to 0 in any sleep mode, the watchdog function is not disabled, and RWDT will reset as the CPU does not feed the watchdog.
- **Switching GPIO to sleep mode**, where the configuration of the GPIO holds. Consider a GPIO pin working in low-drive mode as an input and a wake-up pin. Setting PMU_*n1*_HP_PAD_HOLD_ALL to 1 latches the current configuration of the GPIO pin as the sleep configuration when the chip transitions to the corresponding PMU state. For more information on the hold function of GPIO, please refer to Chapter 10 *GPIO Matrix and IO MUX* > Section 10.9.
- **Disabling UART wake-up function**: Configuring PMU_*n1*_UART_WAKEUP_EN to 0 disables the four UART wake-up modes in the corresponding PMU state. For more information on UART wake-up modes, please refer to Chapter 45 *UART Controller (UART)*.
- **Pausing CPU**: Setting PMU_*n1*_DIG_CPU_STALL to 1 suspends the CPU in the corresponding PMU state.

15.4.2.8 Battery Power Controller

As shown in Figure 15.4-1, the low-power system regulator can be powered by VDD_BAT (battery) or VDD_ANA (analog). PMU_*n1*_VDDBAT_MODE controls whether to enable the battery power. When PMU_*n1*_VDDBAT_MODE is 0, the battery power is not enabled, and the low-power system regulator is powered by VDD_ANA and can be used in Active mode or Light-sleep0/1 modes. When

PMU_*n1*_VDDBAT_MODE is 1, the battery power is enabled and can be used in Deep-sleep mode, where the main power, including VDD_ANA, is cut off, and only the LP AON & PMU power domains are powered up.

15.4.2.9 Output Regulator Control

As shown in Figure 15.4-1, ESP32-P4 has four output regulators:

- VDDO_FLASH (VO1), typically used for on-chip flash voltage regulation
- VDDO_PSRAM (VO2), typically used for on-chip PSRAM voltage regulation
- VDDO_3 (VO3) and VDDO_4 (VO4), which convert the input VDD_LDO (typically 3.3 V) into the voltages needed by peripherals such as flash, PSRAM, MIPI, and SDIO.

15.4.2.9.1 Regulator Features

The regulators support the following features:

- **Supports LDO mode and bypass mode**. LDO mode supports an output of 0.5 V – 2.7 V, while bypass mode outputs the voltage of VDD_LDO. Mode selection is controlled by a hardware state machine or a register, with the control authority managed by a register. The flash regulator (VO1) also supports eFuse control.
  - The mode control for VO1 is shown in Figure 15.4-3. In the figure, TIEH_SEL on the right being 0 selects LDO mode; being 1 selects bypass mode.

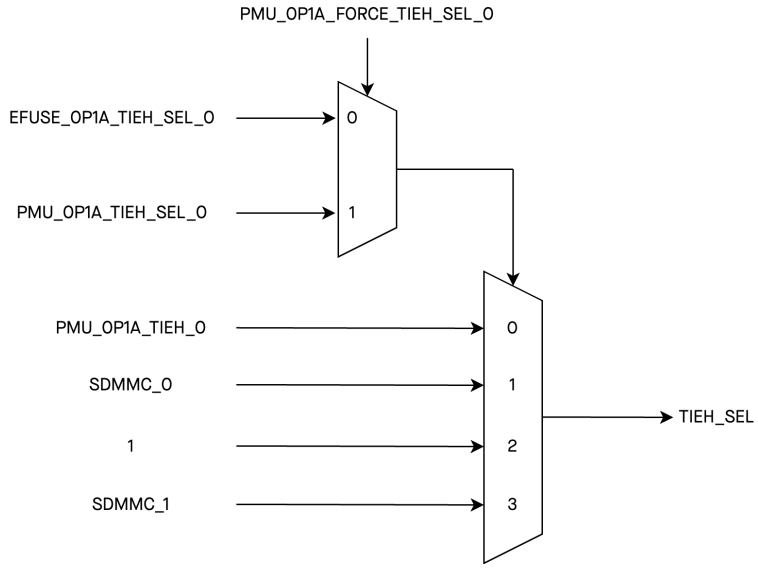


Figure 15.4-3. VO1 Mode Control

  - The mode control for VO2/3/4 is shown in Figure 15.4-4. In the figure, TIEH_SEL on the right being 0 selects LDO mode; being 1 selects bypass mode.

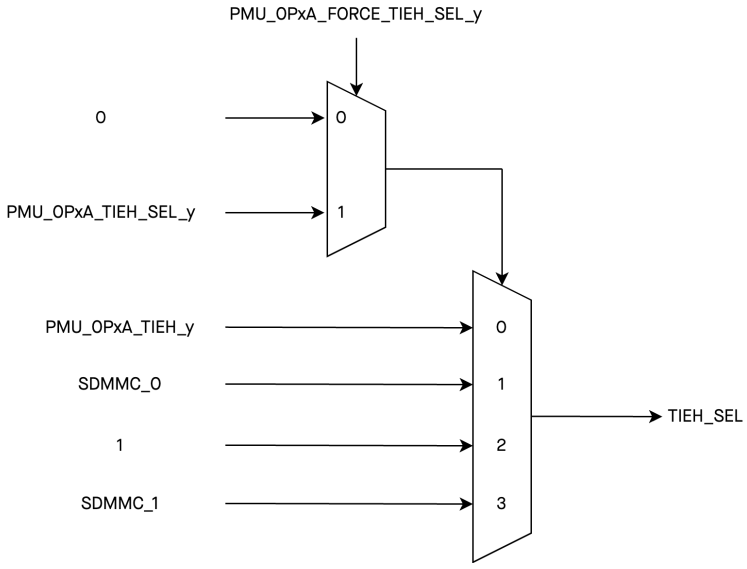


Figure 15.4-4. VO2/3/4 Mode Control

Table 15.4-3. VO2/3/4 Registers

Regulators	PMU_0PxA_FORCE_TIEH_SEL_y	PMU_0PxA_TIEH_SEL_y	PMU_0PxA_TIEH_y
VO2	PMU_0P1A_FORCE_TIEH_SEL_1	PMU_0P1A_TIEH_SEL_1	PMU_0P1A_TIEH_1
VO3	PMU_0P2A_FORCE_TIEH_SEL_0	PMU_0P2A_TIEH_SEL_0	PMU_0P2A_TIEH_0
VO4	PMU_0P2A_FORCE_TIEH_SEL_1	PMU_0P2A_TIEH_SEL_1	PMU_0P2A_TIEH_1

- **Supports LDO output voltage setting**: In LDO mode, the output voltage is configured by analog registers PMU_ANA_0PxA_DREF_y (sets Vref) and PMU_ANA_0PxA_MUL_y (sets MUL), following $V_{OUT} = \text{MUL} \times V_{REF}$. Vref supports 0.5 V–1.6 V, and MUL supports 1.0–2.75 (step 0.25). For details, see field descriptions in PMU_EXT_LDO_P0_0P1A_ANA_REG–PMU_EXT_LDO_P0_0P2A_ANA_REG.
- **Supports switch control**. Only the flash voltage regulator (VO1) supports power control by the PMU state machine, the other regulators only support power control by software. The regulator is enabled when the register in Table 15.4-4 is set to 1; otherwise, the regulator is disabled.

Table 15.4-4. Regulator Power Control

Regulator	Power Control
VO1	PMU_*n1*_REGULATOR_VO1_XPD
VO2	PMU_0P1A_XPD_1
VO3	PMU_0P2A_XPD_0
VO4	PMU_0P2A_XPD_1

- **Over-current protection**: Before powering up the regulators, users have the option to enable the over-current protection function to prevent excessive transient current that may damage the capacitor. After the capacitor charging is complete, the over-current protection should be disabled. The capacitor charging time depends on the load capacitance and voltage, following $T_{ms} = C_F \times V_V / 50$. Table 15.4-5 provides the over-current protection enable registers; when the register is set to 1, the function is enabled; otherwise, it is disabled.

Table 15.4-5. Over-Current Protection Control

Regulators	Over-Current Protection Control Registers
VO1	FSM or PMU_ANA_0P1A_EN_CUR_LIM_0
VO2	PMU_ANA_0P1A_EN_CUR_LIM_1
VO3	PMU_ANA_0P2A_EN_CUR_LIM_0
VO4	PMU_ANA_0P2A_EN_CUR_LIM_1

Due to the long power-up time, a waiting counter is provided for each regulator. Once the regulator is enabled, the counter is triggered and starts counting. The counter has two stages (stage 1/stage 2, corresponding to registers TARGET0/TARGET1), and each stage triggers different interrupts. Typically, set stage 1 (TARGET0) to over-current protection duration, and stage 2 (TARGET1) to voltage stabilization waiting time. The configuration of the counter is shown in the table below:

Table 15.4-6. Regulator Waiting Counter

Regulators	Counter Target 1	Counter Target 2
VO1	PMU_0P1A_TARGET0_0[6:0]	PMU_0P1A_TARGET1_0[7:0]
VO2	PMU_0P1A_TARGET0_1[7:0]	PMU_0P1A_TARGET1_1[7:0]
VO3	PMU_0P2A_TARGET0_0[7:0]	PMU_0P2A_TARGET1_0[7:0]
VO4	PMU_0P2A_TARGET0_1[7:0]	PMU_0P2A_TARGET1_1[7:0]

Note:

1. The bit 7 of PMU_0P1A_TARGET0_0 clears the counter. Setting bit 7 to 1 resets the counter.
2. The counter clock is derived from LP_DYN_SLOW_CLK, which is divided by a factor of 16.

15.4.2.9.2 Regulator Configuration Examples

- **Example 1: VO3 powering MIPI D-PHY (2.5 V, LDO mode)**
  On the board, the VO3 pin is connected to VDD_MIPI_DPHY that requires 2.5 V operating voltage. In this scenario, select LDO mode and set $V_{OUT} = 2.5\,\text{V}$. One valid combination is $V_{REF} = 1.0\,\text{V}$ (DREF=9) and MUL $= 2.5$ (MUL=6), i.e., PMU_ANA_0P2A_DREF_0=9, PMU_ANA_0P2A_MUL_0=6, and set PMU_0P2A_TIEH_0=0 (LDO).
- **Example 2: VO4 powering VDD_IO_5 (SD card voltage switching)**
  In reference designs, VO4 may be connected to VDD_IO_5 to supply some GPIO/SDIO pins.
  – 3.3 V output: use bypass mode (for example, set PMU_0P2A_TIEH_1=1).
  – 1.8 V output: use LDO mode and set $V_{OUT} = 1.8\,\text{V}$. One valid combination is $V_{REF} = 0.9\,\text{V}$ (DREF=8) and MUL $= 2.0$ (MUL=4), i.e., PMU_ANA_0P2A_DREF_1=8, PMU_ANA_0P2A_MUL_1=4, and set PMU_0P2A_TIEH_1=0 (LDO).

15.4.2.9.3 Software Configuration Steps

The recommended software configuration steps (example) are as follows:

1. Configure output mode: set PMU_0PxA_FORCE_TIEH_SEL_y so the register controls mode selection, then choose LDO or bypass mode via PMU_0PxA_TIEH_SEL_y/PMU_0PxA_TIEH_y (see Table 15.4-3).

2. If using LDO mode, configure output voltage: configure PMU_ANA_0PxA_DREF_y and PMU_ANA_0PxA_MUL_y to satisfy $V_{OUT} = \text{MUL} \times V_{REF}$.
3. Configure waiting counter: set stage 1 (TARGET0) to the desired over-current protection duration and stage 2 (TARGET1) to the voltage stabilization waiting time (see Table 15.4-6).
4. Enable over-current protection before power-up: set PMU_ANA_0PxA_EN_CUR_LIM_y=1 (see Table 15.4-5).
5. Enable the regulator: set PMU_0PxA_XPD_y=1 (see Table 15.4-4), then wait for the stage 1 timeout interrupt (e.g., VO3 uses PMU_0P2A_CNT_TARGET0_REACH_0_INT, VO4 uses PMU_0P2A_CNT_TARGET0_REACH_1_INT).
6. After stage 1 interrupt, disable over-current protection: clear PMU_ANA_0PxA_EN_CUR_LIM_y. If a voltage stabilization waiting time is required, wait for the stage 2 timeout interrupt (...CNT_TARGET1_REACH..._INT).

15.5 Power Modes

ESP32-P4 has three configurable PMU states. Based on the PMU states, the chip has defined five power modes for common application scenarios, as shown in Table 15.5-1.

Table 15.5-1. Preset Power Modes

Power Modes	Power Domains							
	LP always-on	LP PD peripherals	HP_CPU	TOP	HPCNNT	RC_FAST_CLK	XTAL_CLK	PLL
Active	ON	ON	ON	ON	ON	ON	ON	ON
Light-sleep0	ON	ON	OFF	ON	ON	ON	ON	ON
Light-sleep1	ON	ON	OFF	ON	ON	ON/OFF	ON/OFF	ON/OFF
Light-sleep2	ON	ON/OFF	OFF	OFF	ON	ON/OFF	ON/OFF	ON/OFF
Deep-sleep	ON	OFF	OFF	OFF	OFF	OFF	OFF	OFF

Note:

1. For power consumption data, please refer to *ESP32-P4 Datasheet*.
2. For supported wake-up sources, please refer to Table 15.4-1.

15.6 Event Task Matrix Feature

The low-power management system on ESP32-P4 supports the Event Task Matrix (ETM) function, which allows the low-power management system's ETM tasks to be triggered by any peripherals' ETM events, or the low-power management system's ETM events to trigger any peripherals' ETM tasks. This section introduces the ETM tasks and events related to the low-power management system. For more information, please refer to Chapter 14 Event Task Matrix (ETM).

The low-power management system can receive the following ETM tasks:

- PMU_TASK_SLEEP_REQ: Triggers PMU's sleep process.

The low-power management system can generate the following ETM events:

- PMU_EVT_SLEEP_WAKEUP: Indicates that PMU is woken up to the HP_ACTIVE state.

15.7 Interrupts

ESP32-P4's low-power management system can generate the following interrupt signals:

- PMU_INTR
- PMU_LP_INT

Among these interrupt signals, PMU_INTR is sent to the Interrupt Matrix, while PMU_LP_INT is sent to the LP CPU.

The interrupt signals are generated by the internal interrupt sources of each module, specifically:

PMU_INTR:

- PMU_SOC_WAKEUP_INT: Triggered when the chip is woken up to the HP_ACTIVE state.
- PMU_SOC_SLEEP_REJECT_INT: Triggered when a sleep-rejection source rejects a sleep request.
- PMU_SW_INT: Triggered when LP CPU is used as a wake-up source and wakes up the chip to HP_ACTIVE state.
- PMU_SDIO_IDLE_INT: Triggered when SDIO is in idle state.
- PMU_LP_CPU_EXC_INT: Triggered when there are LP CPU exceptions.
- PMU_0P2A_CNT_TARGET1_REACH_1_INT: Triggered when VO4 regulator's waiting counter target 1 reaches timeout.
- PMU_0P2A_CNT_TARGET0_REACH_1_INT: Triggered when VO4 regulator's waiting counter target 0 reaches timeout.
- PMU_0P2A_CNT_TARGET1_REACH_0_INT: Triggered when VO3 regulator's waiting counter target 1 reaches timeout.
- PMU_0P2A_CNT_TARGET0_REACH_0_INT: Triggered when VO3 regulator's waiting counter target 0 reaches timeout.
- PMU_0P1A_CNT_TARGET1_REACH_1_INT: Triggered when VO2 regulator's waiting counter target 1 reaches timeout.
- PMU_0P1A_CNT_TARGET0_REACH_1_INT: Triggered when VO2 regulator's waiting counter target 0 reaches timeout.
- PMU_0P1A_CNT_TARGET1_REACH_0_INT: Triggered when VO1 regulator's waiting counter target 1 reaches timeout.
- PMU_0P1A_CNT_TARGET0_REACH_0_INT: Triggered when VO1 regulator's waiting counter target 0 reaches timeout.

PMU_LP_INT:

- PMU_LP_CPU_SLEEP_REJECT_INT: Triggered when the LP sleep request is rejected.

- PMU_HP_SW_TRIGGER_INT: Triggered when HP CPU wakes up LP CPU.
- PMU_ACTIVE_SWITCH_SLEEP_START_INT: Triggered when the PMU state starts to switch from HP_ACTIVE to HP_SLEEP.
- PMU_SLEEP_SWITCH_ACTIVE_START_INT: Triggered when the PMU state starts to switch from HP_SLEEP to HP_ACTIVE.
- PMU_ACTIVE_SWITCH_SLEEP_END_INT: Triggered when the PMU state finishes switching from HP_ACTIVE to HP_SLEEP.
- PMU_SLEEP_SWITCH_ACTIVE_END_INT: Triggered when the PMU state finishes switching from HP_SLEEP to HP_ACTIVE.
- PMU_LP_CPU_WAKEUP_INT: Triggered when LP CPU is woken up.
- PMU_0P2A_CNT_TARGET1_REACH_1_LP_INT: Triggered when VO4 regulator's waiting counter target 1 reaches timeout.
- PMU_0P2A_CNT_TARGET0_REACH_1_LP_INT: Triggered when VO4 regulator's waiting counter target 0 reaches timeout.
- PMU_0P2A_CNT_TARGET1_REACH_0_LP_INT: Triggered when VO3 regulator's waiting counter target 1 reaches timeout.
- PMU_0P2A_CNT_TARGET0_REACH_0_LP_INT: Triggered when VO3 regulator's waiting counter target 0 reaches timeout.
- PMU_0P1A_CNT_TARGET1_REACH_1_LP_INT: Triggered when VO2 regulator's waiting counter target 1 reaches timeout.
- PMU_0P1A_CNT_TARGET0_REACH_1_LP_INT: Triggered when VO2 regulator's waiting counter target 0 reaches timeout.
- PMU_0P1A_CNT_TARGET1_REACH_0_LP_INT: Triggered when VO1 regulator's waiting counter target 1 reaches timeout.
- PMU_0P1A_CNT_TARGET0_REACH_0_LP_INT: Triggered when VO1 regulator's waiting counter target 0 reaches timeout.

Each interrupt source can be configured by a common set of registers that are described in Section *Interrupt Configuration Registers*. The specific registers can be found in Section 15.8 *Register Summary*.

15.8 Register Summary

The addresses in this section are relative to the PMU base address provided in Table 9.3-2 in Chapter 9 *System and Memory*.

The abbreviations given in Column **Access** are explained in Section *Access Types for Registers*.

Name	Description	Address	Access
Configuration Registers			
PMU_HP_ACTIVE_DIG_POWER_REG	Digital power domain control register in HP_ACTIVE state	0x0000	R/W
PMU_HP_ACTIVE_ICG_HP_FUNC_REG	HP system peripheral's function clock control register in HP_ACTIVE state	0x0004	R/W
PMU_HP_ACTIVE_HP_SYS_CNTL_REG	System control register in HP_ACTIVE state	0x0010	R/W
PMU_HP_ACTIVE_HP_CK_POWER_REG	Clock source power control register in HP_ACTIVE state	0x0014	R/W
PMU_HP_ACTIVE_BIAS_REG	BIAS control register in HP_ACTIVE state	0x0018	R/W
PMU_HP_ACTIVE_BACKUP_REG	Backup module control register in HP_ACTIVE state	0x001C	R/W
PMU_HP_ACTIVE_BACKUP_CLK_REG	Backup module's function clock control register in HP_ACTIVE state	0x0020	R/W
PMU_HP_ACTIVE_SYSCLK_REG	System clock control register in HP_ACTIVE state	0x0024	R/W
PMU_HP_ACTIVE_HP_REGULATOR0_REG	Regulator power control register in HP_ACTIVE state	0x0028	varies
PMU_HP_ACTIVE_XTAL_REG	XTAL_CLK power control register in HP_ACTIVE state	0x0030	R/W
PMU_HP_SLEEP_DIG_POWER_REG	Digital power domain control register in HP_SLEEP state	0x0068	R/W
PMU_HP_SLEEP_ICG_HP_FUNC_REG	HP system peripheral's function clock control register in HP_SLEEP state	0x006C	R/W
PMU_HP_SLEEP_HP_SYS_CNTL_REG	System control register in HP_SLEEP state	0x0078	R/W
PMU_HP_SLEEP_HP_CK_POWER_REG	Clock source power control register in HP_SLEEP state	0x007C	R/W
PMU_HP_SLEEP_BIAS_REG	BIAS control register in HP_SLEEP state	0x0080	R/W
PMU_HP_SLEEP_BACKUP_REG	Backup module control register in HP_SLEEP state	0x0084	R/W
PMU_HP_SLEEP_BACKUP_CLK_REG	Backup module's function clock control register in HP_SLEEP state	0x0088	R/W
PMU_HP_SLEEP_SYSCLK_REG	System clock control register in HP_SLEEP state	0x008C	R/W
PMU_HP_SLEEP_HP_REGULATOR0_REG	Regulator power control register in HP_SLEEP state	0x0090	R/W

Name	Description	Address	Access
PMU_HP_SLEEP_XTAL_REG	XTAL_CLK power control register in HP_SLEEP state	0x0098	R/W
PMU_HP_SLEEP_LP_REGULATOR0_REG	LP sys regulator control register in HP_SLEEP state	0x009C	R/W
PMU_HP_SLEEP_LP_DIG_POWER_REG	LP system digital power domains control register in HP_SLEEP state	0x00A8	R/W
PMU_HP_SLEEP_LP_CK_POWER_REG	Low-speed clock power control register in HP_ACTIVE/HP_SLEEP state	0x00AC	R/W
PMU_LP_SLEEP_LP_REGULATOR0_REG	LP sys regulator control register in HP_SLEEP state	0x00B4	R/W
PMU_LP_SLEEP_XTAL_REG	XTAL_CLK power control register in LP_SLEEP state	0x00BC	R/W
PMU_LP_SLEEP_LP_DIG_POWER_REG	Digital power domain control register in LP_SLEEP state	0x00C0	R/W
PMU_LP_SLEEP_LP_CK_POWER_REG	Low-speed clock power control register in LP_SLEEP state	0x00C4	R/W
PMU_IMM_HP_CK_POWER_REG	Software control register for clock power	0x00CC	WT
PMU_IMM_SLEEP_SYSCLK_REG	Software control register for system clock	0x00D0	WT
PMU_IMM_HP_FUNC_ICG_REG	Software control register for peripheral clock	0x00D4	WT
PMU_IMM_HP_APB_ICG_REG	Software control register for APB clock	0x00D8	WT
PMU_IMM_PAD_HOLD_ALL_REG	Software control register for PAD hold function	0x00E4	varies
PMU_POWER_CK_WAIT_CNTL_REG	Clock gating wait time configuration register	0x011C	R/W
PMU_SLP_WAKEUP_CNTL0_REG	Sleep request register	0x0120	WT
PMU_SLP_WAKEUP_CNTL1_REG	Sleep reject register	0x0124	R/W
PMU_SLP_WAKEUP_CNTL2_REG	Wake up source enable register	0x0128	R/W
PMU_SLP_WAKEUP_CNTL3_REG	Minimum sleep time control register	0x012C	R/W
PMU_SLP_WAKEUP_CNTL4_REG	Sleep reject cause clear register	0x0130	WT
PMU_SLP_WAKEUP_STATUS0_REG	Wake up cause register	0x0144	RO
PMU_SLP_WAKEUP_STATUS1_REG	Reset reject cause register	0x0148	RO
PMU_HP_CK_CNTL_REG	HP system clock control register	0x0154	R/W
PMU_RF_PWC_REG	SAR ADC power up register	0x015C	R/W
PMU_INT_RAW_REG	PMU sleep/wake-up raw interrupt	0x0164	R/WTC/SS
PMU_HP_INT_ST_REG	PMU sleep/wake-up state interrupt	0x0168	RO
PMU_HP_INT_ENA_REG	PMU sleep/wake-up interrupt enable register	0x016C	R/W
PMU_HP_INT_CLR_REG	PMU sleep/wake-up interrupt clear register	0x0170	WT
PMU_LP_INT_RAW_REG	PMU sleep/wake-up raw interrupt	0x0174	R/WTC/SS
PMU_LP_INT_ST_REG	PMU sleep/wake-up state interrupt	0x0178	RO

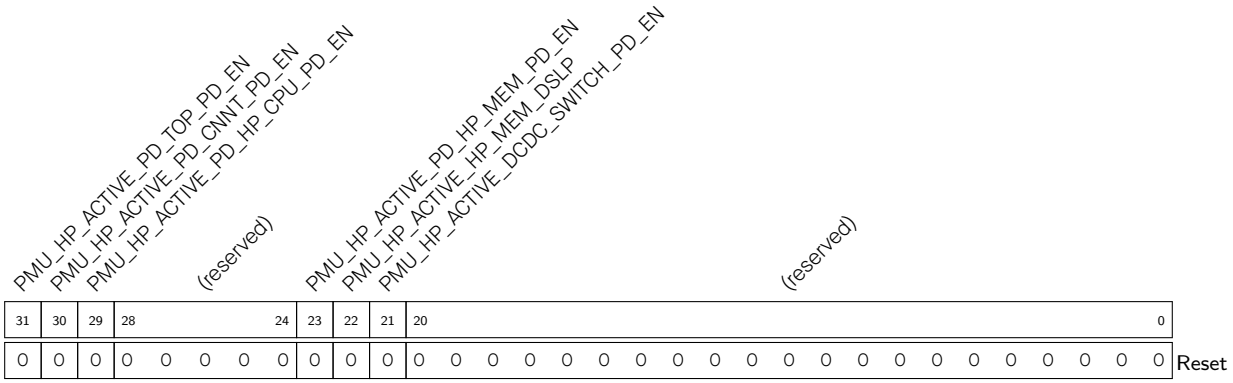
Name	Description	Address	Access
PMU_LP_INT_ENA_REG	PMU sleep/wake-up interrupt enable register	0x017C	R/W
PMU_LP_INT_CLR_REG	PMU sleep/wake-up interrupt clear register	0x0180	varies
PMU_LP_CPU_PWR0_REG	LP CPU sleep/wake-up control register	0x0184	varies
PMU_LP_CPU_PWR1_REG	LP CPU sleep/wake-up control register	0x0188	WT
PMU_LP_CPU_PWR2_REG	LP CPU sleep/wake-up control register	0x018C	R/W
PMU_LP_CPU_PWR3_REG	LP CPU sleep/wake-up control register	0x0190	RO
PMU_LP_CPU_PWR4_REG	LP CPU sleep/wake-up control register	0x0194	R/W
PMU_LP_CPU_PWR5_REG	LP CPU sleep/wake-up control register	0x0198	RO
PMU_HP_LP_CPU_COMM_REG	LP CPU sleep/wake-up control register	0x019C	WT
PMU_EXT_LDO_P0_OP1A_REG	VO1 regulator control register	0x01B8	R/W
PMU_EXT_LDO_P0_OP1A_ANA_REG	VO1 regulator control register	0x01BC	R/W
PMU_EXT_LDO_P0_OP2A_REG	VO3 regulator control register	0x01C0	R/W
PMU_EXT_LDO_P0_OP2A_ANA_REG	VO3 regulator control register	0x01C4	R/W
PMU_EXT_LDO_P1_OP1A_REG	VO2 regulator control register	0x01D0	R/W
PMU_EXT_LDO_P1_OP1A_ANA_REG	VO2 regulator control register	0x01D4	R/W
PMU_EXT_LDO_P1_OP2A_REG	VO4 regulator control register	0x01D8	R/W
PMU_EXT_LDO_P1_OP2A_ANA_REG	VO4 regulator control register	0x01DC	R/W
PMU_EXT_WAKEUP_LV_REG	EXT wake-up level control register	0x01E8	R/W
PMU_EXT_WAKEUP_SEL_REG	EXT wake-up IO control register	0x01EC	R/W
PMU_EXT_WAKEUP_ST_REG	EXT wake-up status register	0x01F0	RO
PMU_EXT_WAKEUP_CNTL_REG	EXT wake-up control register	0x01F4	R/W
PMU_SDIO_WAKEUP_CNTL_REG	SDIO wake-up control register	0x01F8	R/W
PMU_TOUCH_PWR_CNTL_REG	TOUCH sleep/wake-up control register	0x0210	R/W
PMU_DATE_REG	Version control register	0x03FC	R/W

15.9 Registers

The addresses in this section are relative to the PMU base address provided in Table 9.3-2 in Chapter 9 *System and Memory*.

For how to program reserved fields, please refer to Section *Programming Reserved Register Field*.

Register 15.1. PMU_HP_ACTIVE_DIG_POWER_REG (0x0000)



PMU_HP_ACTIVE_DCDC_SWITCH_PD_EN Configures whether to cut off the power from external DCDC in HP_ACTIVE state.

0: Do not cut off

1: Cut off

(R/W)

PMU_HP_ACTIVE_HP_MEM_DSLP Configures whether to put L2MEM_G0–G5 into Deep-sleep mode in HP_ACTIVE state.

0: Do not put L2MEM_G0–G5 into Deep-sleep

1: Put L2MEM_G0–G5 into Deep-sleep

(R/W)

PMU_HP_ACTIVE_PD_HP_MEM_PD_EN Configures whether to power down L2MEM_G0–G5 in HP_ACTIVE state.

0: Power up

1: Power down

(R/W)

PMU_HP_ACTIVE_PD_HP_CPU_PD_EN Configures whether to power down the HP_CPU power domain in HP_ACTIVE state.

0: Power up

1: Power down

(R/W)

PMU_HP_ACTIVE_PD_CNNT_PD_EN Configures whether to power down HP_CNNT in HP_ACTIVE state.

0: Power up

1: Power down

(R/W)

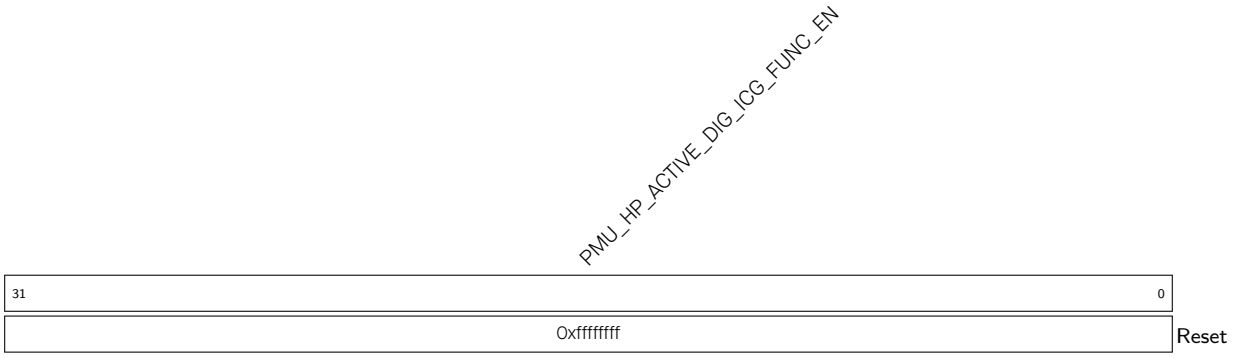
PMU_HP_ACTIVE_PD_TOP_PD_EN Configures whether to power down PD_TOP power domain in HP_ACTIVE state.

0: Power up

1: Power down

(R/W)

Register 15.2. PMU_HP_ACTIVE_ICG_HP_FUNC_REG (0x0004)



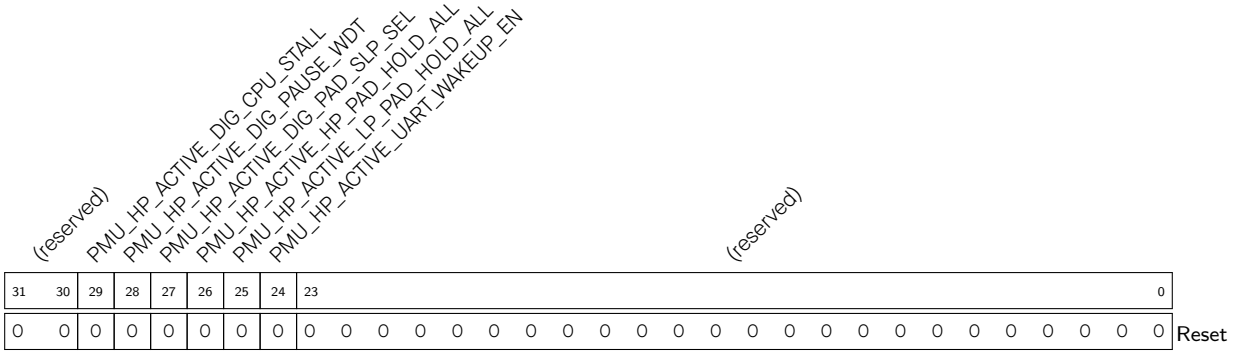
PMU_HP_ACTIVE_DIG_ICG_FUNC_EN Configures whether to enable HP system peripheral's function clock in HP_ACTIVE state. For detailed configuration please see Table 15.4-2.

0: Disable

1: Enable

(R/W)

Register 15.3. PMU_HP_ACTIVE_HP_SYS_CNTL_REG (0x0010)



PMU_HP_ACTIVE_UART_WAKEUP_EN Configures whether to enable UART wake up function in HP_ACTIVE state.

0: Disable wake-up function

1: Enable wake-up function

(R/W)

PMU_HP_ACTIVE_LP_PAD_HOLD_ALL Configures whether to hold LP GPIO's configuration in HP_ACTIVE state.

0: Do not hold

1: Hold

(R/W)

PMU_HP_ACTIVE_HP_PAD_HOLD_ALL Configures whether to hold GPIO's configuration in HP_ACTIVE state.

0: Do not hold

1: Hold

(R/W)

PMU_HP_ACTIVE_DIG_PAD_SLP_SEL Configures whether to use Light-sleep mode configuration for GPIO in HP_ACTIVE state.

0: Use normal configuration

1: Use Light-sleep mode configuration. For details please refer to Chapter 10 *GPIO Matrix and IO MUX* > Section 10.8 *Pin Functions in Light-sleep*.

(R/W)

PMU_HP_ACTIVE_DIG_PAUSE_WDT Configures whether to pause watchdog in HP_ACTIVE state.

0: Do not pause

1: Pause

(R/W)

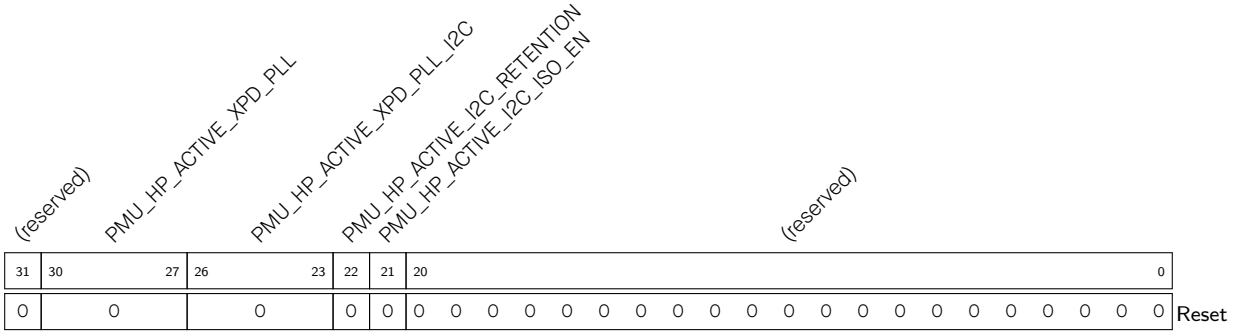
PMU_HP_ACTIVE_DIG_CPU_STALL Configures whether to stall the CPU in HP_ACTIVE state.

0: Do not stall

1: Stall

(R/W)

Register 15.4. PMU_HP_ACTIVE_HP_CK_POWER_REG (0x0014)



PMU_HP_ACTIVE_I2C_ISO_EN Configure whether to power up ANALOG_I2C in HP_ACTIVE state.

0: Power down

1: Power up

(R/W)

PMU_HP_ACTIVE_I2C_RETENTION Configures whether to enable ANALOG_I2C retention status in HP_ACTIVE state.

0: Disable

1: Enable

(R/W)

PMU_HP_ACTIVE_XPD_PLL_I2C Configures whether to enable PLL I2C controllers in HP_ACTIVE state. Each bit controls a PLL I2C register group.

0: Disable

1: Enable

(R/W)

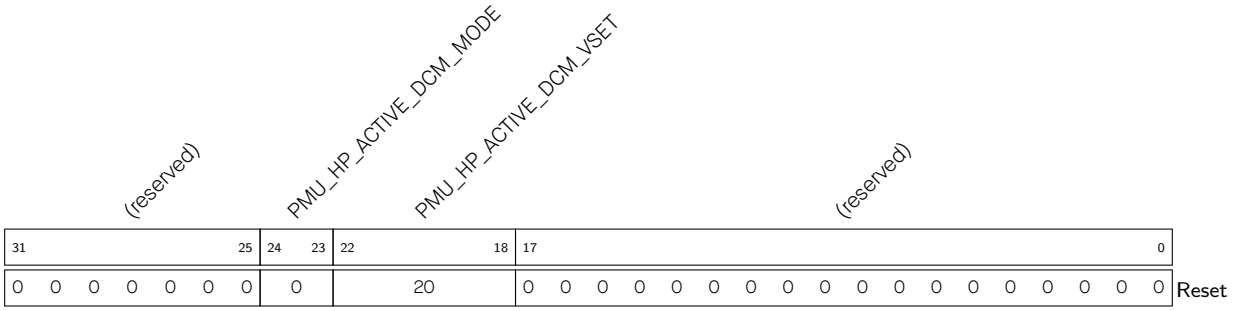
PMU_HP_ACTIVE_XPD_PLL Configures whether to enable the analog power domains in HP system in HP_ACTIVE state. Bit 0 controls CPLL_CLK, bit 1 controls SPLL_CLK, bit 2 controls Audio PLL, bit 3 controls SDIO PLL.

0: Disable

1: Enable

(R/W)

Register 15.5. PMU_HP_ACTIVE_BIAS_REG (0x0018)



PMU_HP_ACTIVE_DCM_VSET Regulates the DCDC voltage in HP_ACTIVE state. (R/W)

PMU_HP_ACTIVE_DCM_MODE Configures whether to enable DCDC status in HP_ACTIVE state.

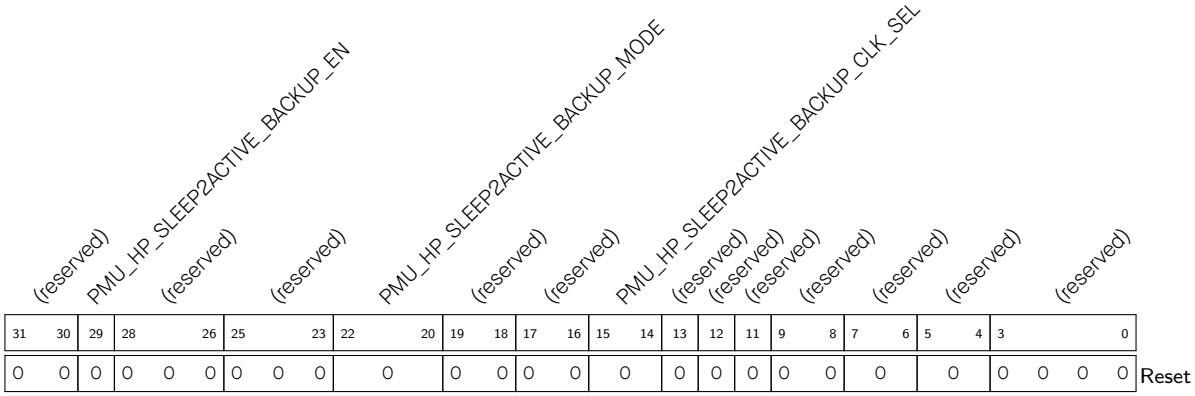
0: Disable

1: Enable

2, 3: Reserved

(R/W)

Register 15.6. PMU_HP_ACTIVE_BACKUP_REG (0x001C)



PMU_HP_SLEEP2ACTIVE_BACKUP_CLK_SEL Configures the backup module's function clock source when PMU state switches from HP_SLEEP to HP_ACTIVE.

0: Select XTAL

1: Select PLL_CLK

2: Select RC_FAST_CLK

3: Invalid value

(R/W)

PMU_HP_SLEEP2ACTIVE_BACKUP_MODE Configures the backup direction and link list when PMU state switches switch from HP_SLEEP to HP_ACTIVE.

Highest bit:

0: From peripheral to memory

1: From memory to peripheral

Lower two bits:

0: PAU_LINK_ADDR_0

1: PAU_LINK_ADDR_1

2: PAU_LINK_ADDR_2

3: PAU_LINK_ADDR_3

(R/W)

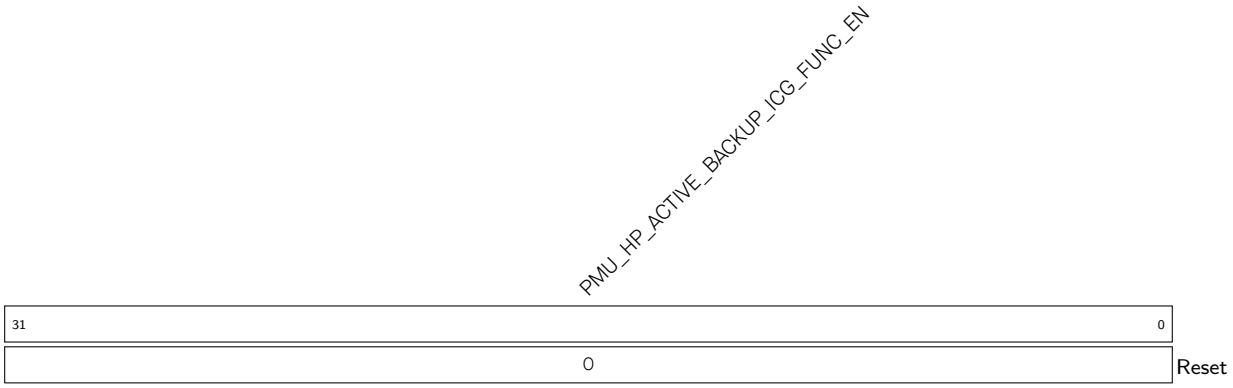
PMU_HP_SLEEP2ACTIVE_BACKUP_EN Configures whether to enable the backup flow when PMU state switches from HP_SLEEP to HP_ACTIVE.

0: Disable backup

1: Enable backup

(R/W)

Register 15.7. PMU_HP_ACTIVE_BACKUP_CLK_REG (0x0020)



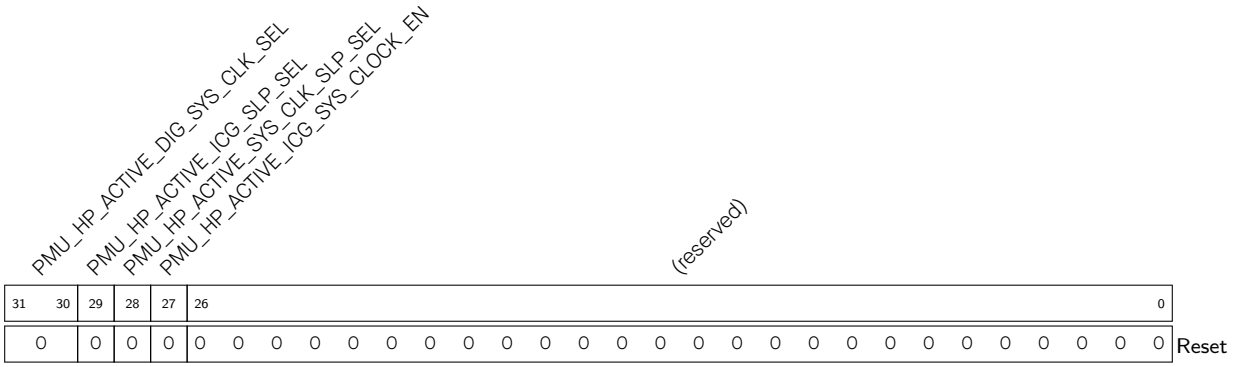
PMU_HP_ACTIVE_BACKUP_ICG_FUNC_EN Configures whether to enable each peripheral's function clock when the target state is HP_ACTIVE. For details, please refer to Table 15.4-2.

0: Disable

1: Enable

(R/W)

Register 15.8. PMU_HP_ACTIVE_SYSCLK_REG (0x0024)



PMU_HP_ACTIVE_ICG_SYS_CLOCK_EN Configures whether to enable ROOT_CLK in HP_ACTIVE state.

0: Disable

1: Enable

(R/W)

PMU_HP_ACTIVE_SYS_CLK_SLP_SEL Configures whether to allow PMU to control clock source in HP_ACTIVE state.

0: Controlled by PCR registers

1: Controlled by PMU

(R/W)

PMU_HP_ACTIVE_ICG_SLP_SEL Configures whether to allow PMU to control the clock gating in HP_ACTIVE state.

0: Controlled by PCR registers

1: Controlled by PMU

(R/W)

PMU_HP_ACTIVE_DIG_SYS_CLK_SEL Configures the source of ROOT_CLK in HP_ACTIVE state.

0: XTAL

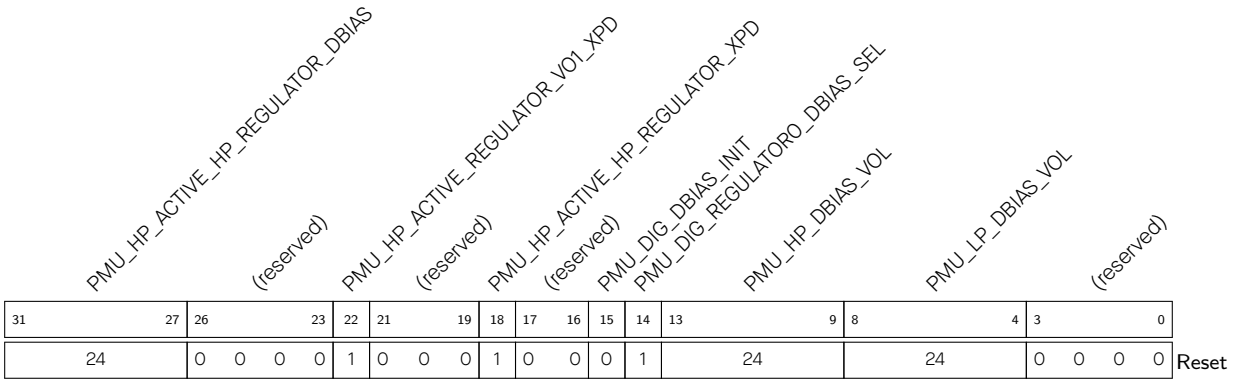
1: PLL_CLK

2: RC_FAST_CLK

3: Invalid value

(R/W)

Register 15.9. PMU_HP_ACTIVE_HP_REGULATOR0_REG (0x0028)



PMU_LP_DBIAS_VOL Indicates the current voltage of the LP system regulator. (RO)

PMU_HP_DBIAS_VOL Indicates the current voltage of the HP system regulator. (RO)

PMU_DIG_REGULATOR0_DBIAS_SEL Configures which of the following regulates the HP system regulator voltage.
0: Regulated by Hardware automatically
1: Regulated by Software
(R/W)

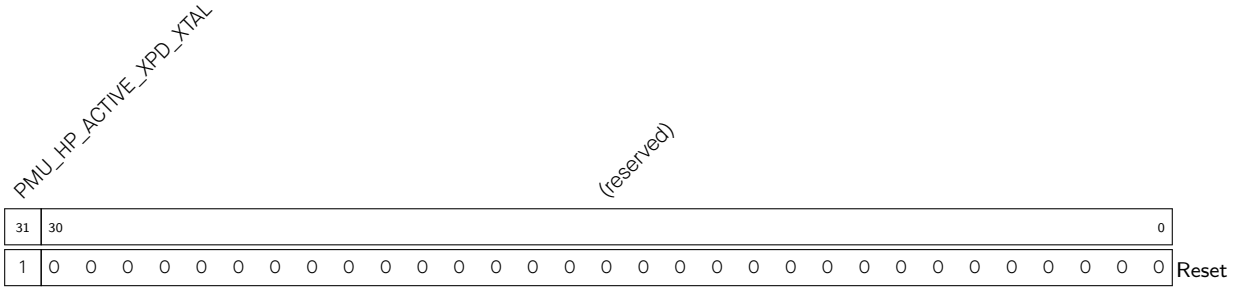
PMU_DIG_DBIAS_INIT Initializes the PVT voltage configurations. (WT)

PMU_HP_ACTIVE_HP_REGULATOR_XPD Configures whether to enable the HP sys regulator in HP_ACTIVE state.
0: Disable the HP sys regulator
1: Enable the HP sys regulator
(R/W)

PMU_HP_ACTIVE_REGULATOR_VO1_XPD Configures whether to enable VO1 regulator in HP_ACTIVE state.
0: Disable
1: Enable
(R/W)

PMU_HP_ACTIVE_HP_REGULATOR_DBIAS Regulates the voltage of the HP sys regulator in HP_ACTIVE state. The higher the value, the higher the voltage. (R/W)

Register 15.10. PMU_HP_ACTIVE_XTAL_REG (0x0030)



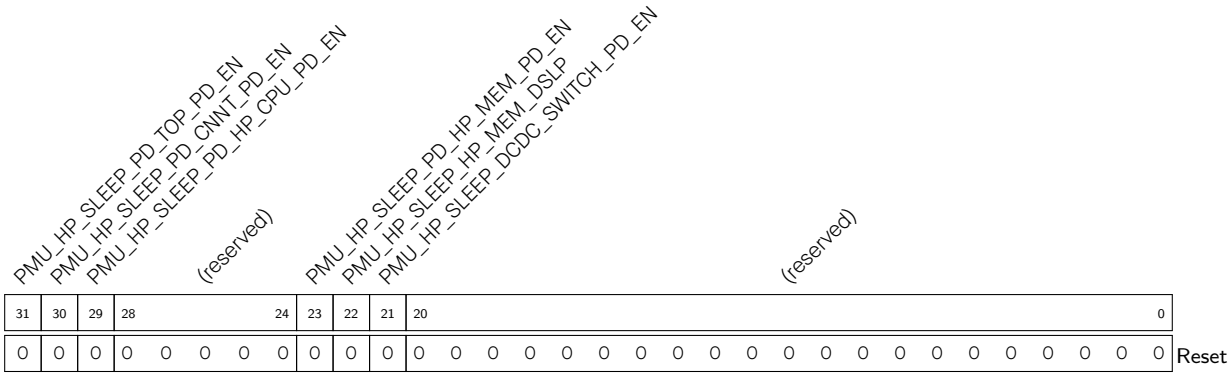
PMU_HP_ACTIVE_XPD_XTAL Configures whether to enable XTAL_CLK analog source in HP_ACTIVE state.

0: Disable

1: Enable

(R/W)

Register 15.11. PMU_HP_SLEEP_DIG_POWER_REG (0x0068)



PMU_HP_SLEEP_DCDC_SWITCH_PD_EN Configures whether to cut off the power from external DCDC in HP_SLEEP state.

0: Do not cut off

1: Cut off

(R/W)

PMU_HP_SLEEP_HP_MEM_DSLP Configures whether to put L2MEM_G0–G5 into Deep-sleep mode in HP_SLEEP state.

0: Do not put L2MEM_G0–G5 into Deep-sleep

1: Put L2MEM_G0–G5 into Deep-sleep

(R/W)

PMU_HP_SLEEP_PD_HP_MEM_PD_EN Configures whether to power down L2MEM_G0–G5 in HP_SLEEP state.

0: Power up

1: Power down

(R/W)

PMU_HP_SLEEP_PD_HP_CPU_PD_EN Configures whether to power down the HP_CPU power domain in HP_SLEEP state.

0: Power up

1: Power down

(R/W)

PMU_HP_SLEEP_PD_CNNT_PD_EN Configures whether to power down HP_CNNT in HP_SLEEP state.

0: Power up

1: Power down

(R/W)

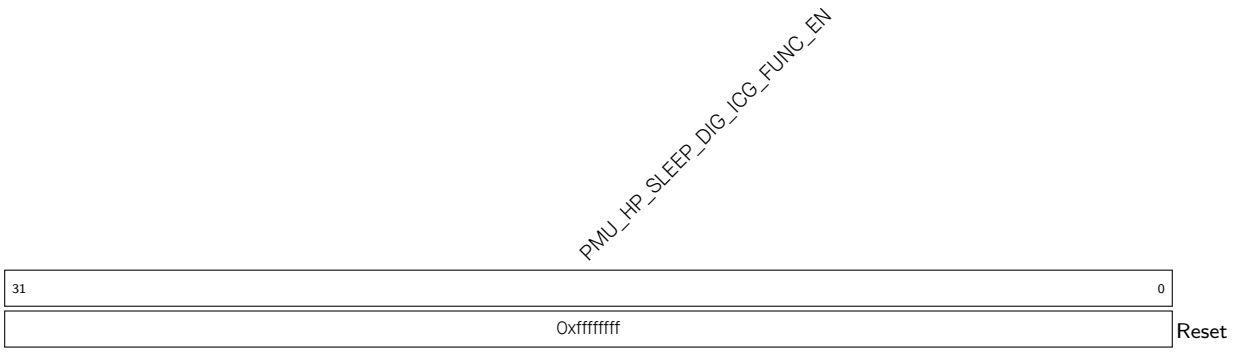
PMU_HP_SLEEP_PD_TOP_PD_EN Configures whether to power down PD_TOP power domain in HP_SLEEP state.

0: Power up

1: Power down

(R/W)

Register 15.12. PMU_HP_SLEEP_ICG_HP_FUNC_REG (0x006C)



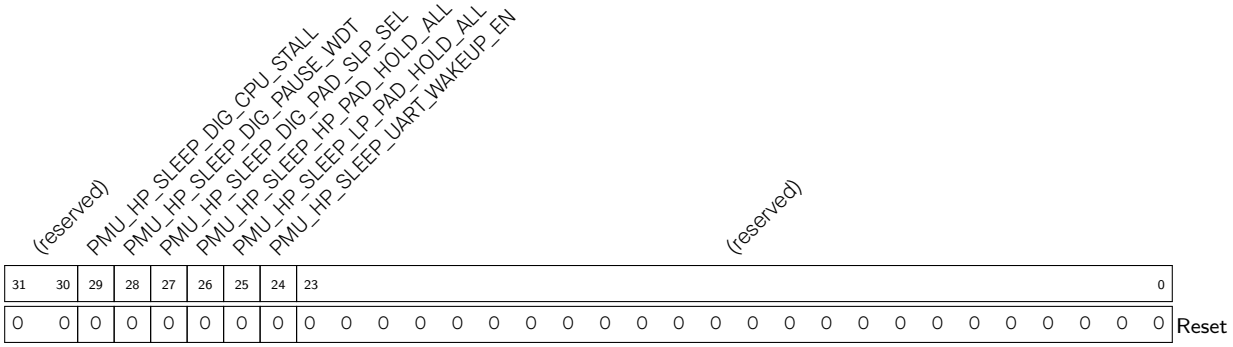
PMU_HP_SLEEP_DIG_ICG_FUNC_EN Configures whether to enable HP system peripheral's function clock in HP_SLEEP state. For detailed configuration please see Table 15.4-2.

0: Disable

1: Enable

(R/W)

Register 15.13. PMU_HP_SLEEP_HP_SYS_CNTL_REG (0x0078)



PMU_HP_SLEEP_UART_WAKEUP_EN Configures whether to enable UART wake up function in HP_SLEEP state.

0: Disable wake-up function

1: Enable wake-up function

(R/W)

PMU_HP_SLEEP_LP_PAD_HOLD_ALL Configures whether to hold LP GPIO's configuration in HP_SLEEP state.

0: Do not hold

1: Hold

(R/W)

PMU_HP_SLEEP_HP_PAD_HOLD_ALL Configures whether to hold GPIO's configuration in HP_SLEEP state.

0: Do not hold

1: Hold

(R/W)

PMU_HP_SLEEP_DIG_PAD_SLP_SEL Configures whether to use Light-sleep mode configuration for GPIO in HP_SLEEP state.

0: Use normal configuration

1: Use Light-sleep mode configuration. For details please refer to Chapter 10 *GPIO Matrix and IO MUX* > Section 10.8 *Pin Functions in Light-sleep*.

(R/W)

PMU_HP_SLEEP_DIG_PAUSE_WDT Configures whether to pause watchdog in HP_SLEEP state.

0: Do not pause

1: Pause

(R/W)

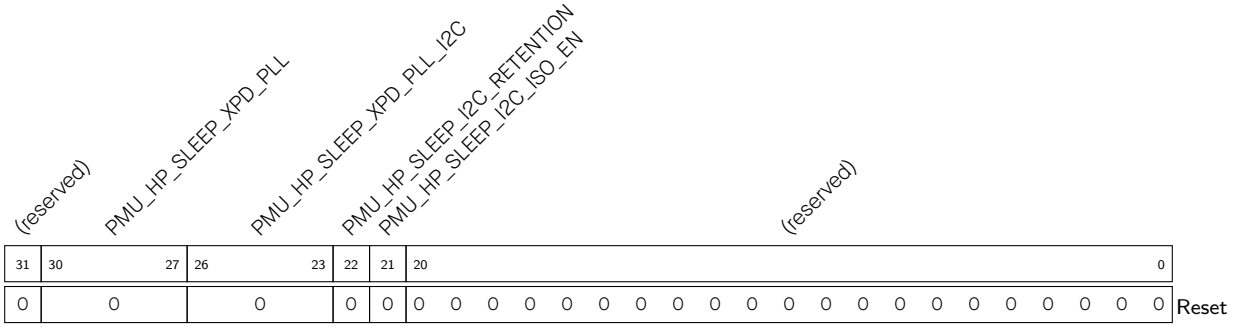
PMU_HP_SLEEP_DIG_CPU_STALL Configures whether to stall the CPU in HP_SLEEP state.

0: Do not stall

1: Stall

(R/W)

Register 15.14. PMU_HP_SLEEP_HP_CK_POWER_REG (0x007C)



PMU_HP_SLEEP_I2C_ISO_EN Configure whether to power up ANALOG_I2C in HP_SLEEP state.

0: Power down

1: Power up

(R/W)

PMU_HP_SLEEP_I2C_RETENTION Configures whether to enable ANALOG_I2C retention status in HP_SLEEP state.

0: Disable

1: Enable

(R/W)

PMU_HP_SLEEP_XPD_PLL_I2C Configures whether to enable PLL I2C controllers in HP_SLEEP state. Each bit controls a PLL I2C register group.

0: Disable

1: Enable

(R/W)

PMU_HP_SLEEP_XPD_PLL Configures whether to enable the analog power domains in HP system in HP_SLEEP state. Bit 0 controls CPLL_CLK, bit 1 controls SPLL_CLK, bit 2 controls Audio PLL, bit 3 controls SDIO PLL.

0: Disable

1: Enable

(R/W)

Register 15.15. PMU_HP_SLEEP_BIAS_REG (0x0080)



PMU_HP_SLEEP_DCM_VSET Regulates the DCDC voltage in HP_SLEEP state. (R/W)

PMU_HP_SLEEP_DCM_MODE Configures whether to enable DCDC status in HP_SLEEP state.

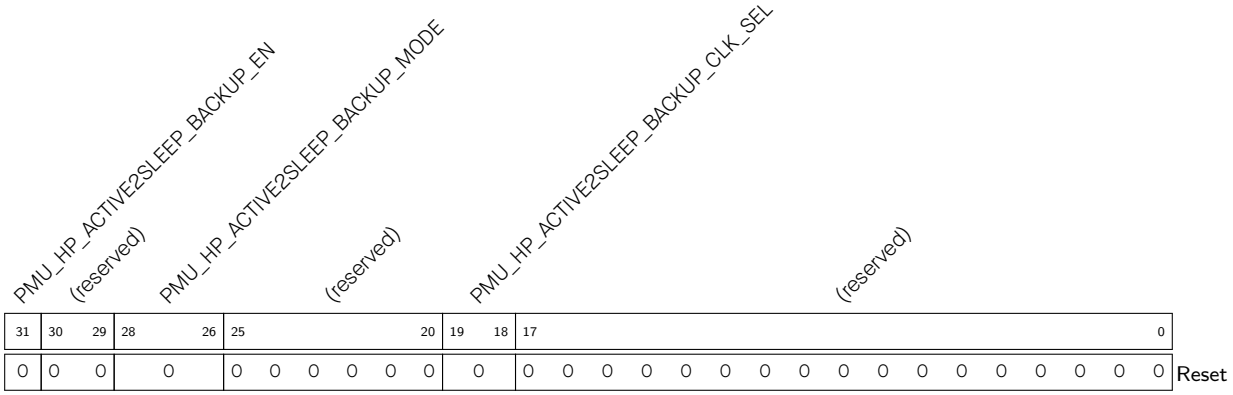
0: Disable

1: Enable

2, 3: Reserved

(R/W)

Register 15.16. PMU_HP_SLEEP_BACKUP_REG (0x0084)



PMU_HP_ACTIVE2SLEEP_BACKUP_CLK_SEL Configures the backup module function clock source when PMU state switches from HP_ACTIVE to HP_SLEEP.

0: Select XTAL

1: Select PLL_CLK

2: Select RC_FAST_CLK

3: Invalid value

(R/W)

PMU_HP_ACTIVE2SLEEP_BACKUP_MODE Configures the backup direction and link list when PMU state switches from HP_ACTIVE to HP_SLEEP.

Highest bit:

0: From memory to peripheral

1: From peripheral to memory

Lower two bits:

0: PAU_LINK_ADDR_0

1: PAU_LINK_ADDR_1

2: PAU_LINK_ADDR_2

3: PAU_LINK_ADDR_3

(R/W)

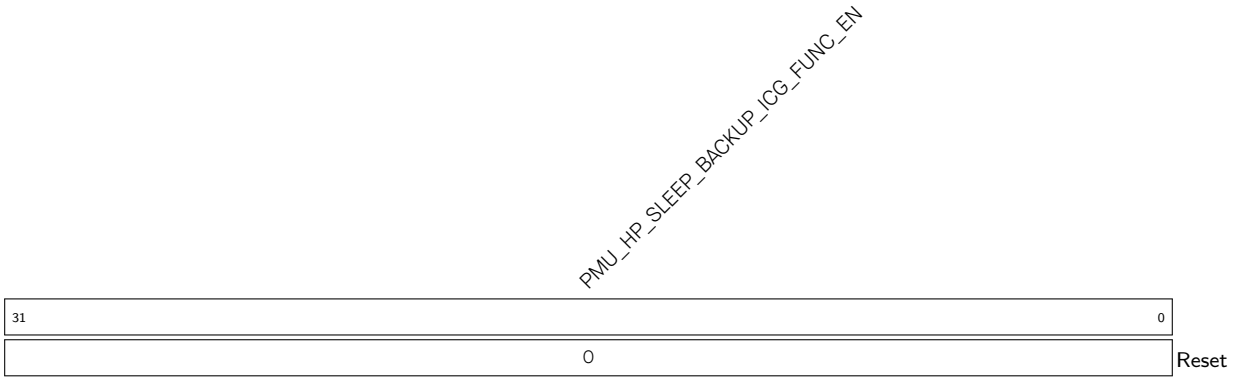
PMU_HP_ACTIVE2SLEEP_BACKUP_EN Configures whether to enable the backup flow when PMU state switches from HP_ACTIVE to HP_SLEEP.

0: Disable backup

1: Enable backup

(R/W)

Register 15.17. PMU_HP_SLEEP_BACKUP_CLK_REG (0x0088)



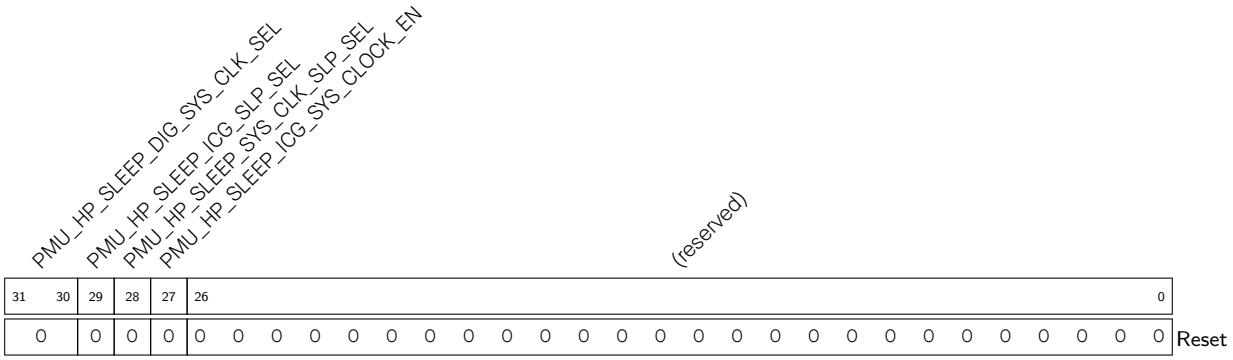
PMU_HP_SLEEP_BACKUP_ICG_FUNC_EN Configures whether to enable each peripheral's function clock when the target state is HP_SLEEP. For details, please refer to Table 15.4-2.

0: Disable

1: Enable

(R/W)

Register 15.18. PMU_HP_SLEEP_SYSCLK_REG (0x008C)



PMU_HP_SLEEP_ICG_SYS_CLOCK_EN Configures whether to enable ROOT_CLK in HP_SLEEP state.

0: Disable

1: Enable

(R/W)

PMU_HP_SLEEP_SYS_CLK_SLP_SEL Configures whether to allow PMU to control the clock source in HP_SLEEP state.

0: Controlled by PCR registers

1: Controlled by PMU

(R/W)

PMU_HP_SLEEP_ICG_SLP_SEL Configures whether to allow PMU to control the clock gating in HP_SLEEP state.

0: Controlled by PCR registers

1: Controlled by PMU

(R/W)

PMU_HP_SLEEP_DIG_SYS_CLK_SEL Configures the source of ROOT_CLK in HP_SLEEP state.

0: XTAL

1: PLL_CLK

2: RC_FAST_CLK

3: Invalid value

(R/W)

Register 15.19. PMU_HP_SLEEP_HP_REGULATOR0_REG (0x0090)



PMU_HP_SLEEP_HP_REGULATOR_XPD Configures whether to enable the HP sys regulator in HP_SLEEP state.

0: Disable

1: Enable

(R/W)

PMU_HP_SLEEP_REGULATOR_VO1_XPD Configures whether to enable VO1 regulator in HP_SLEEP state.

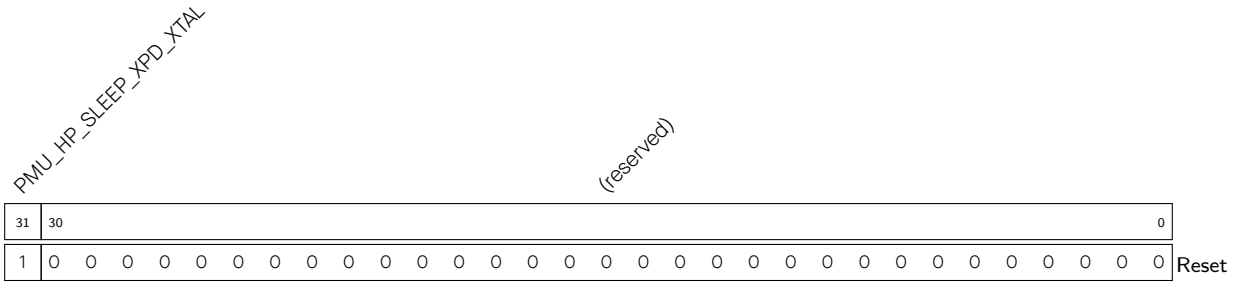
0: Disable

1: Enable

(R/W)

PMU_HP_SLEEP_HP_REGULATOR_DBIAS Regulates the voltage of the HP sys regulator in HP_SLEEP state. The higher the value, the higher the voltage. (R/W)

Register 15.20. PMU_HP_SLEEP_XTAL_REG (0x0098)



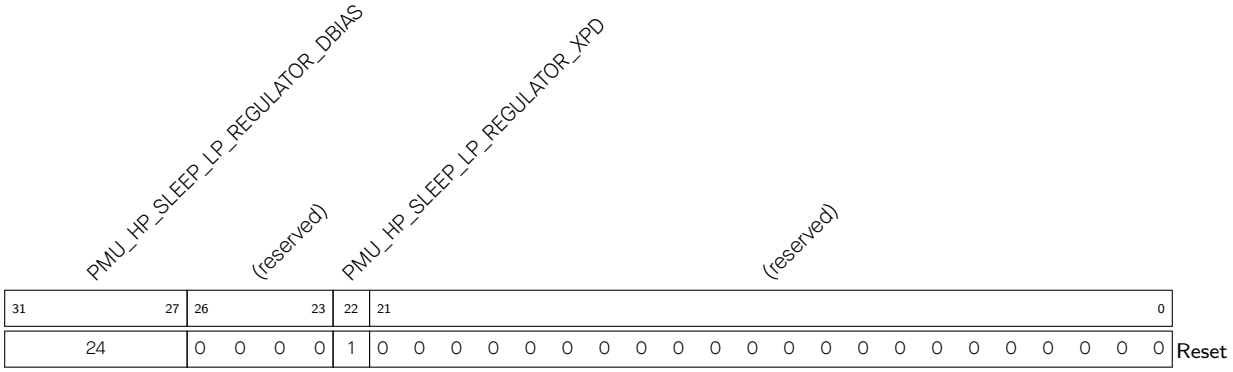
PMU_HP_SLEEP_XPD_XTAL Configures whether to enable XTAL_CLK analog source in HP_SLEEP state.

0: Disable

1: Enable

(R/W)

Register 15.21. PMU_HP_SLEEP_LP_REGULATOR0_REG (0x009C)



PMU_HP_SLEEP_LP_REGULATOR_XPD Configures whether to enable LP sys regulator in HP_ACTIVE/HP_SLEEP state.

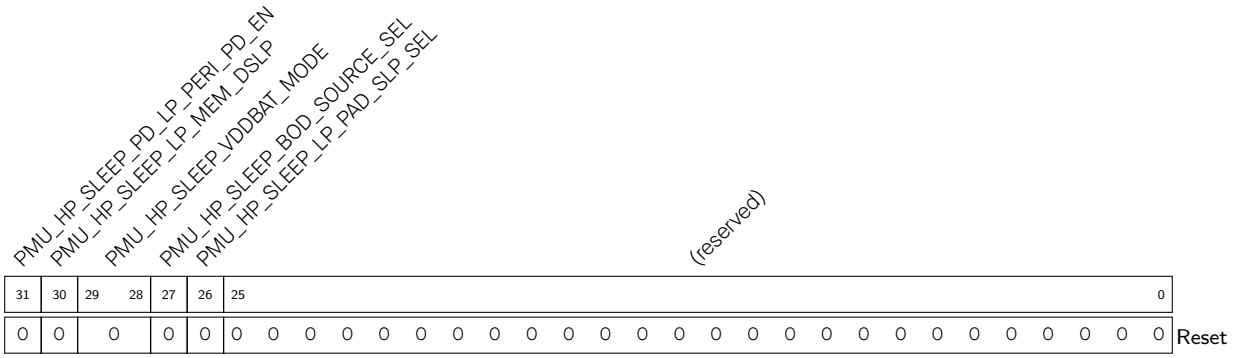
0: Disable

1: Enable

(R/W)

PMU_HP_SLEEP_LP_REGULATOR_DBIAS Regulates the voltage of the LP sys regulator in HP_ACTIVE/HP_SLEEP state. The higher the value, the higher the voltage. (R/W)

Register 15.22. PMU_HP_SLEEP_LP_DIG_POWER_REG (0x00A8)



PMU_HP_SLEEP_LP_PAD_SLP_SEL Configures whether to enable LP GPIO function in HP_ACTIVE/HP_SLEEP state.

0: Disable

1: Enable

(R/W)

PMU_HP_SLEEP_BOD_SOURCE_SEL Configures whether to enable brown-out detection in HP_ACTIVE/HP_SLEEP. (R/W)

PMU_HP_SLEEP_VDDBAT_MODE Configures whether to enable VBAT power in HP_ACTIVE/HP_SLEEP state.

0: Disable

1: Enable

(R/W)

PMU_HP_SLEEP_LP_MEM_DSLP Configures whether to enable Memory Deep-sleep mode for the LP SRAM in HP_ACTIVE/HP_SLEEP state.

0: Disable

1: Enable

(R/W)

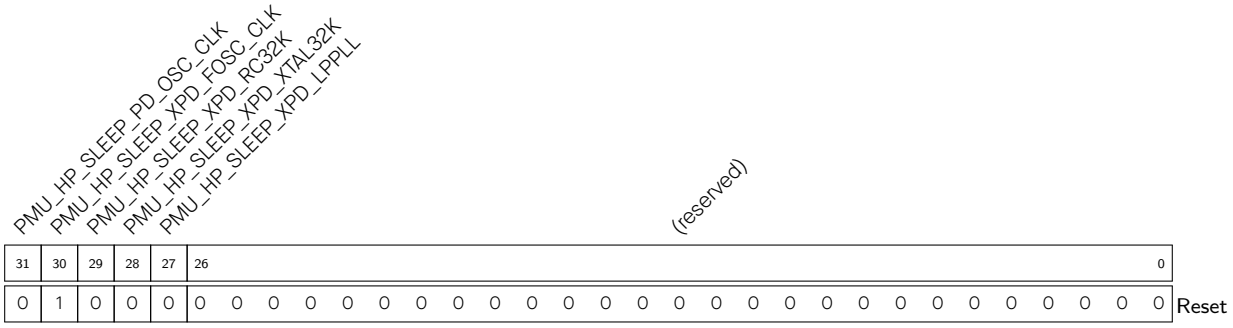
PMU_HP_SLEEP_PD_LP_PERI_PD_EN Configures whether to power down LP PD Peripherals in HP_ACTIVE/HP_SLEEP state.

0: Power up

1: Power down

(R/W)

Register 15.23. PMU_HP_SLEEP_LP_CK_POWER_REG (0x00AC)



PMU_HP_SLEEP_XPD_LPPLL Configures whether to enable PLL_LP_CLK in HP_ACTIVE/HP_SLEEP state.

0: Disable

1: Enable

(R/W)

PMU_HP_SLEEP_XPD_XTAL32K Configures whether to enable XTAL32K_CLK in HP_ACTIVE/HP_SLEEP state.

0: Disable

1: Enable

(R/W)

PMU_HP_SLEEP_XPD_RC32K Configures whether to enable RC32K_CLK in HP_ACTIVE/HP_SLEEP state.

0: Disable

1: Enable

(R/W)

PMU_HP_SLEEP_XPD_FOSC_CLK Configures whether to enable RC_FAST_CLK in HP_ACTIVE/HP_SLEEP state.

0: Disable

1: Enable

(R/W)

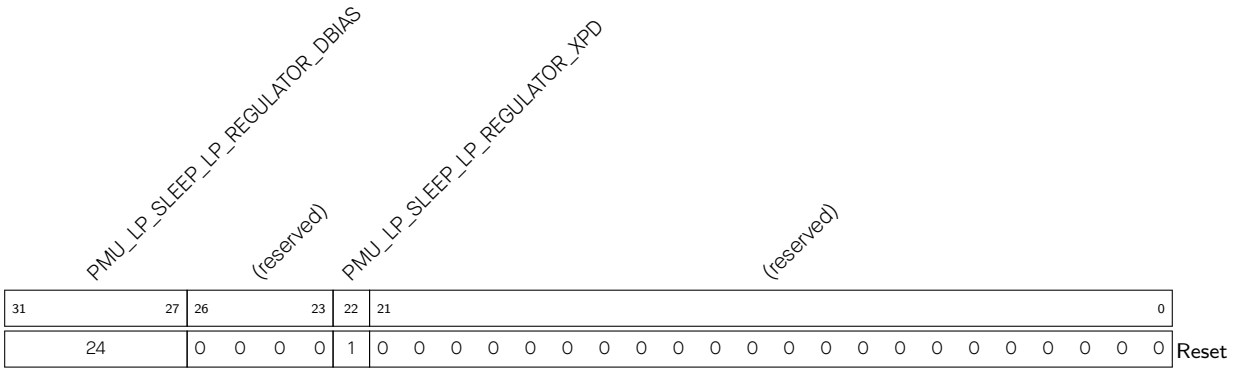
PMU_HP_SLEEP_PD_OSC_CLK Configures whether to enable RC_SLOW_CLK in HP_ACTIVE/HP_SLEEP state.

0: Disable

1: Enable

(R/W)

Register 15.24. PMU_LP_SLEEP_LP_REGULATOR0_REG (0x00B4)



PMU_LP_SLEEP_LP_REGULATOR_XPD Configures whether to enable the LP sys regulator in LP_SLEEP state.

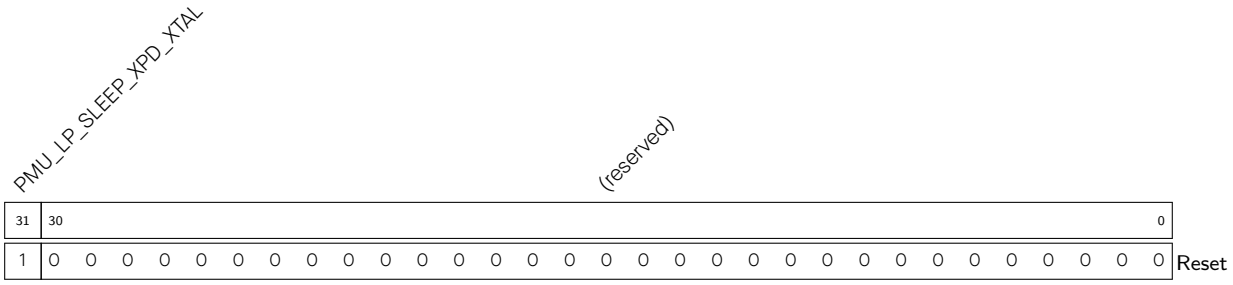
0: Disable

1: Enable

(R/W)

PMU_LP_SLEEP_LP_REGULATOR_DBIAS Regulates the voltage of the LP sys regulator in LP_SLEEP state. The higher the value, the higher the voltage. (R/W)

Register 15.25. PMU_LP_SLEEP_XTAL_REG (0x00BC)



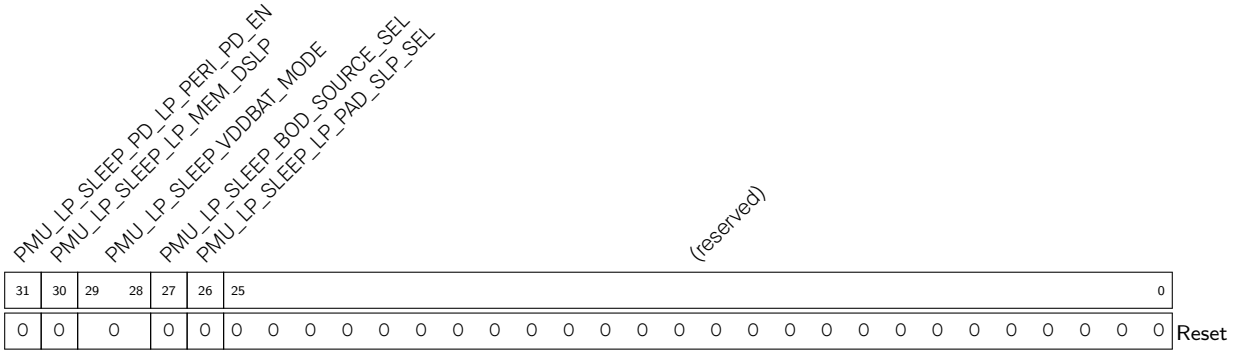
PMU_LP_SLEEP_XPD_XTAL Configures whether to enable XTAL_CLK analog source in LP_SLEEP state.

0: Disable

1: Enable

(R/W)

Register 15.26. PMU_LP_SLEEP_LP_DIG_POWER_REG (0x00C0)



PMU_LP_SLEEP_LP_PAD_SLP_SEL Configures whether to enable LP GPIO function in LP_SLEEP state.

0: Disable

1: Enable

(R/W)

PMU_LP_SLEEP_BOD_SOURCE_SEL Configures whether to enable brown-out detection in LP_SLEEP state. (R/W)

PMU_LP_SLEEP_VDDBAT_MODE Configures whether to enable VBAT power in LP_SLEEP state.

0: Disable

1: Enable

(R/W)

PMU_LP_SLEEP_LP_MEM_DSLP Configures whether to enable Memory Deep-sleep mode for the LP SRAM in LP_SLEEP state.

0: Disable

1: Enable

(R/W)

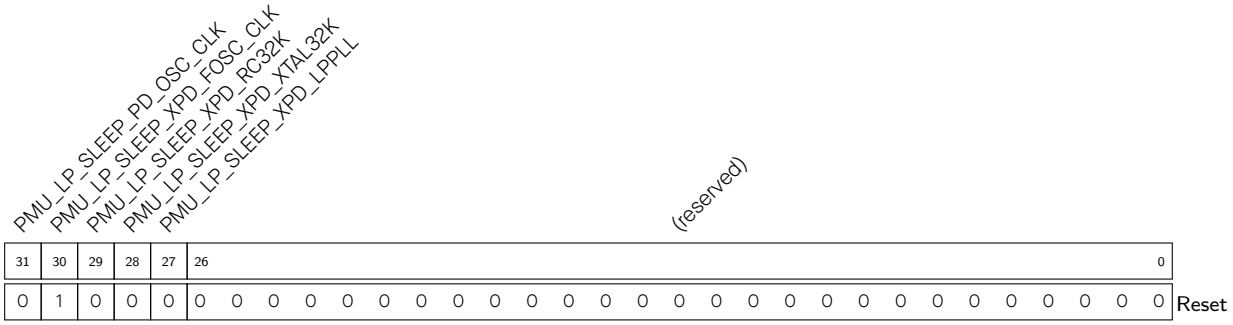
PMU_LP_SLEEP_PD_LP_PERI_PD_EN Configures whether to power down LP PD Peripherals in LP_SLEEP state.

0: Power up

1: Power down

(R/W)

Register 15.27. PMU_LP_SLEEP_LP_CK_POWER_REG (0x00C4)



PMU_LP_SLEEP_XPD_LPPLL Configures whether to enable PLL_LP_CLK in LP_SLEEP state.

0: Disable

1: Enable

(R/W)

PMU_LP_SLEEP_XPD_XTAL32K Configures whether to enable XTAL32K_CLK in LP_SLEEP state.

0: Disable

1: Enable

(R/W)

PMU_LP_SLEEP_XPD_RC32K Configures whether to enable RC32K_CLK in LP_SLEEP state.

0: Disable

1: Enable

(R/W)

PMU_LP_SLEEP_XPD_FOSC_CLK Configures whether to enable RC_FAST_CLK in LP_SLEEP state.

0: Disable

1: Enable

(R/W)

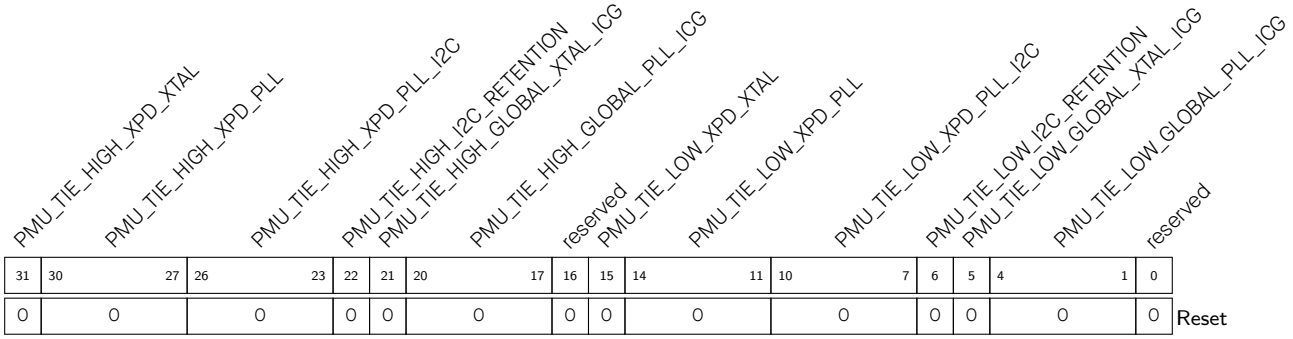
PMU_LP_SLEEP_PD_OSC_CLK Configures whether to enable RC_SLOW_CLK in LP_SLEEP state.

0: Disable

1: Enable

(R/W)

Register 15.28. PMU_IMM_HP_CK_POWER_REG (0x00CC)



PMU_TIE_LOW_GLOBAL_PLL_ICG Configures whether to power down global PLL clock gating bitmap.

0: No effect

1: Power down

(WT)

PMU_TIE_LOW_GLOBAL_XTAL_ICG Configures whether to power down XTAL clock.

0: No effect

1: Power down

(WT)

PMU_TIE_LOW_I2C_RETENTION Configures whether to power down ANALOG I2C retention.

0: No effect

1: Power down

(WT)

PMU_TIE_LOW_XPD_PLL_I2C Configures whether to power down global PLL I2C clock power bitmap.

0: No effect

1: Power down

(WT)

PMU_TIE_LOW_XPD_PLL Configures whether to power down global PLL clock power bitmap.

0: No effect

1: Power down

(WT)

PMU_TIE_LOW_XPD_XTAL Configures whether to power down XTAL clock power.

0: No effect

1: Power down

(WT)

PMU_TIE_HIGH_GLOBAL_PLL_ICG Configures whether to enable PLL clock gating bitmap.

0: No effect

1: Enable

(WT)

Continued on the next page...

Register 15.28. PMU_IMM_HP_CK_POWER_REG (0x00CC)

Continued from the previous page...

PMU_TIE_HIGH_GLOBAL_XTAL_ICG Configures whether to enable global XTAL clock.

0: No effect

1: Enable

(WT)

PMU_TIE_HIGH_I2C_RETENTION Configures whether to enable ANALOG I2C retention.

0: No effect

1: Enable

(WT)

PMU_TIE_HIGH_XPD_PLL_I2C Configures whether to enable PLL I2C clock power bitmap.

0: No effect

1: Enable

(WT)

PMU_TIE_HIGH_XPD_PLL Configures whether to enable global PLL clock power bitmap.

0: No effect

1: Enable

(WT)

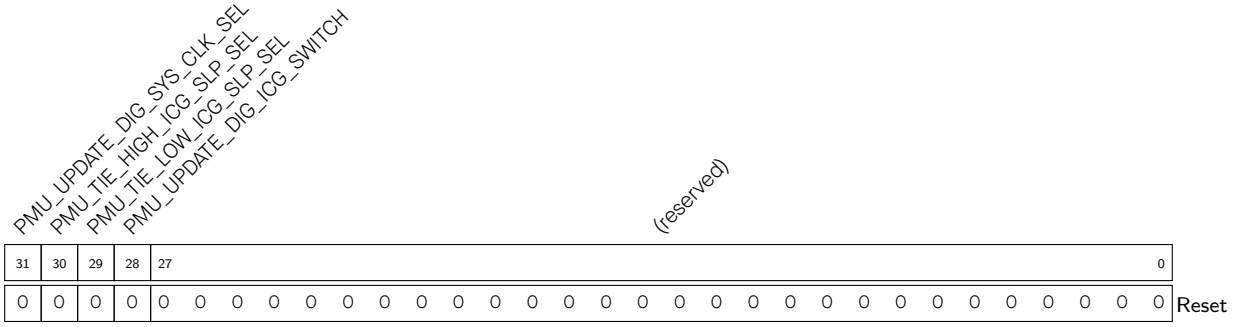
PMU_TIE_HIGH_XPD_XTAL Configures whether to enable global XTAL clock power.

0: No effect

1: Enable

(WT)

Register 15.29. PMU_IMM_SLEEP_SYSCLK_REG (0x00D0)



PMU_UPDATE_DIG_ICG_SWITCH Immediately updates function clock gating status. (WT)

PMU_TIE_LOW_ICG_SLP_SEL Configures whether to disable PMU control of function clock.

0: No effect

1: Disable

(WT)

PMU_TIE_HIGH_ICG_SLP_SEL Configures whether to enable PMU control of function clock.

0: No effect

1: Enable

(WT)

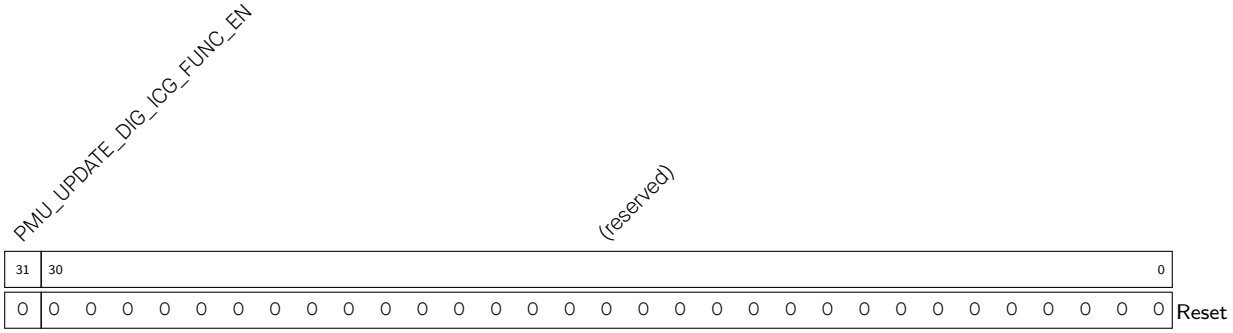
PMU_UPDATE_DIG_SYS_CLK_SEL Immediately updates system clock.

0: No effect

1: Updates the configurations of SYS_CLK_SEL in the current PMU state to the application side.

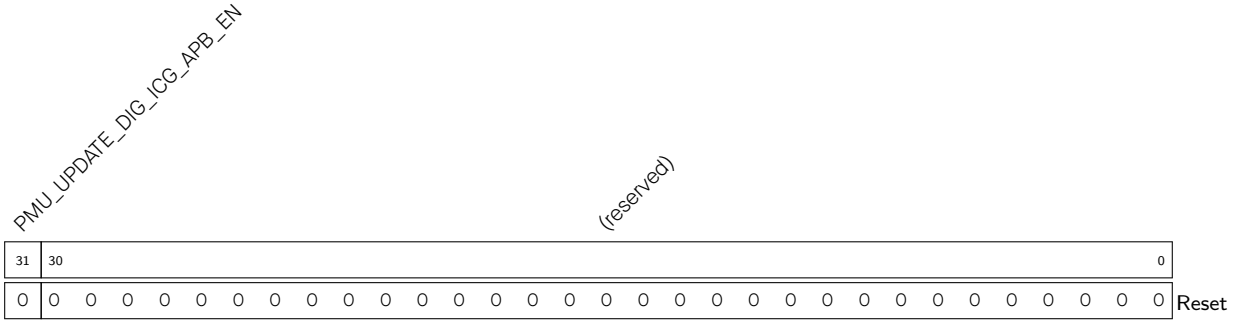
(WT)

Register 15.30. PMU_IMM_HP_FUNC_ICG_REG (0x00D4)



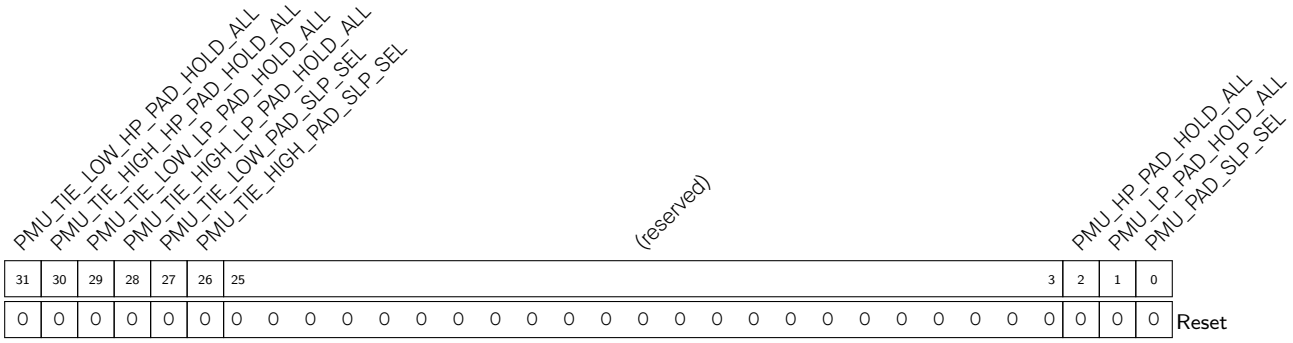
PMU_UPDATE_DIG_ICG_FUNC_EN Immediately updates function clock gating bitmap. (WT)

Register 15.31. PMU_IMM_HP_APB_ICG_REG (0x00D8)



PMU_UPDATE_DIG_ICG_APB_EN Immediately updates APB clock gating bitmap. (WT)

Register 15.32. PMU_IMM_PAD_HOLD_ALL_REG (0x00E4)



PMU_PAD_SLP_SEL Indicates whether the current GPIO function uses sleep configurations.

0: No

1: Yes

(RO)

PMU_LP_PAD_HOLD_ALL Indicates whether the LP PAD is in the HOLD status.

0: No

1: Yes

(RO)

PMU_HP_PAD_HOLD_ALL Represents whether the HP PAD is in the HOLD status.

0: No

1: Yes

(RO)

PMU_TIE_HIGH_PAD_SLP_SEL Immediately configures GPIO function to use sleep configurations. (WT)

PMU_TIE_LOW_PAD_SLP_SEL Immediately releases GPIO function from using sleep configuration. (WT)

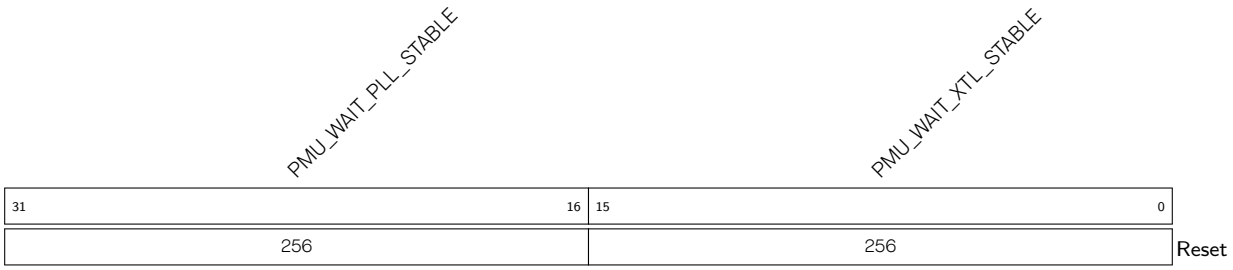
PMU_TIE_HIGH_LP_PAD_HOLD_ALL Immediately configure LP PAD to enter HOLD state. (WT)

PMU_TIE_LOW_LP_PAD_HOLD_ALL Immediately releases LP PAD from HOLD state. (WT)

PMU_TIE_HIGH_HP_PAD_HOLD_ALL Immediately configures HP PAD to enter HOLD state. (WT)

PMU_TIE_LOW_HP_PAD_HOLD_ALL Immediately releases HP PAD from HOLD state. (WT)

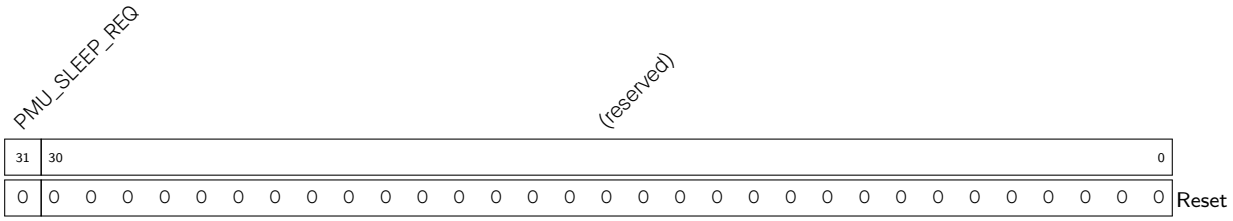
Register 15.33. PMU_POWER_CK_WAIT_CNTL_REG (0x011C)



PMU_WAIT_XTL_STABLE Configures the wait time for enabling XTAL global clock gating after the XTAL power supply is enabled. The unit is LP_DYN_FAST_CLK. (R/W)

PMU_WAIT_PLL_STABLE Configures the wait time for enabling PLL global clock gating after the PLL power supply is enabled. The unit is LP_DYN_FAST_CLK. (R/W)

Register 15.34. PMU_SLP_WAKEUP_CNTL0_REG (0x0120)



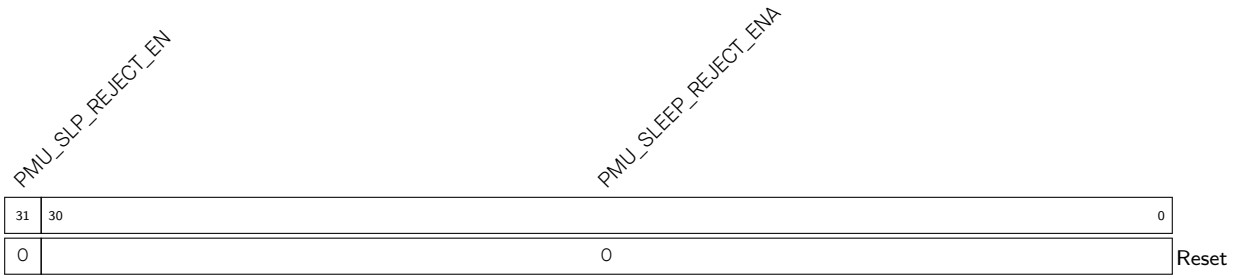
PMU_SLEEP_REQ Configures whether to switch the PMU state to HP_SLEEP or LP_SLEEP.

0: Do not switch

1: Switch to HP_SLEEP or LP_SLEEP, depending on the state of the LP CPU.

(WT)

Register 15.35. PMU_SLP_WAKEUP_CNTL1_REG (0x0124)



PMU_SLEEP_REJECT_ENA Configures the sleep rejection source. For the mapping between values and sources please refer to Table 15.4-1. (R/W)

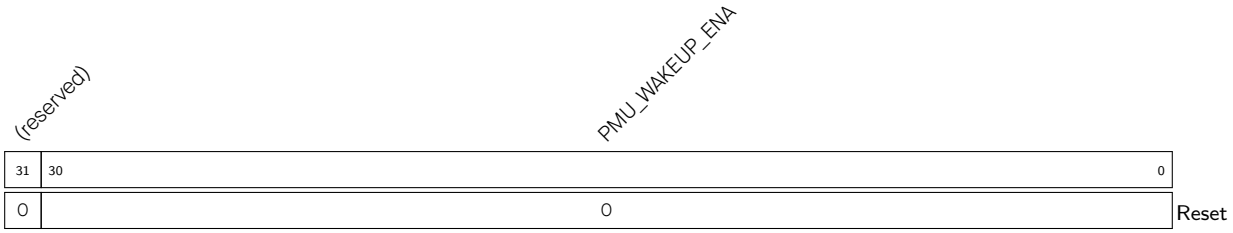
PMU_SLP_REJECT_EN Configures whether to enable sleep rejection function.

0: Disable

1: Enable

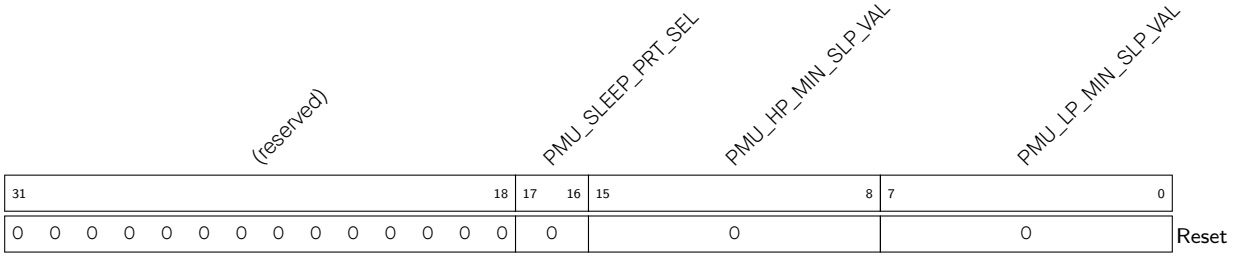
(R/W)

Register 15.36. PMU_SLP_WAKEUP_CNTL2_REG (0x0128)



PMU_WAKEUP_ENA Configures wake-up source. For the mapping between values and sources please refer to Table 15.4-1. (R/W)

Register 15.37. PMU_SLP_WAKEUP_CNTL3_REG (0x012C)



PMU_LP_MIN_SLP_VAL Configures the minimum sleep time to enter LP_SLEEP. The unit is LP_DYN_SLOW_CLK. (R/W)

PMU_HP_MIN_SLP_VAL Configures the minimum sleep time to enter HP_SLEEP. The unit is LP_DYN_SLOW_CLK. (R/W)

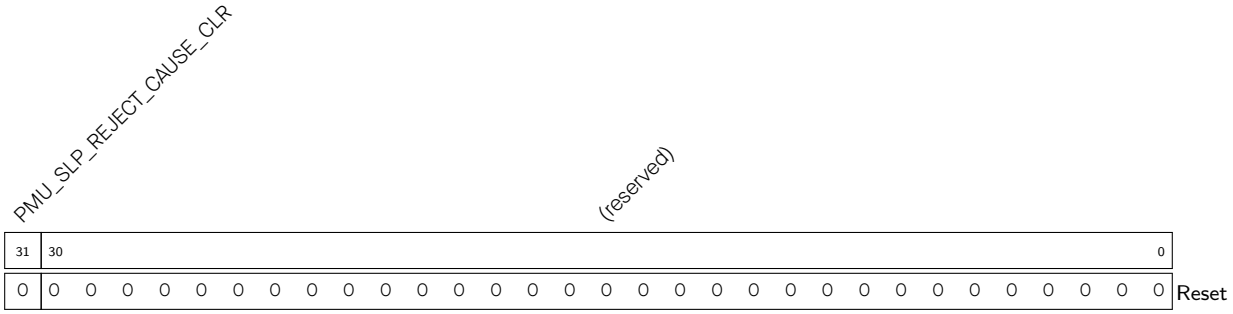
PMU_SLEEP_PRT_SEL Configures the minimum sleep time mode.

0, 1, 3: Reserved

2: Protects both LP_SLEEP and HP_SLEEP simultaneously

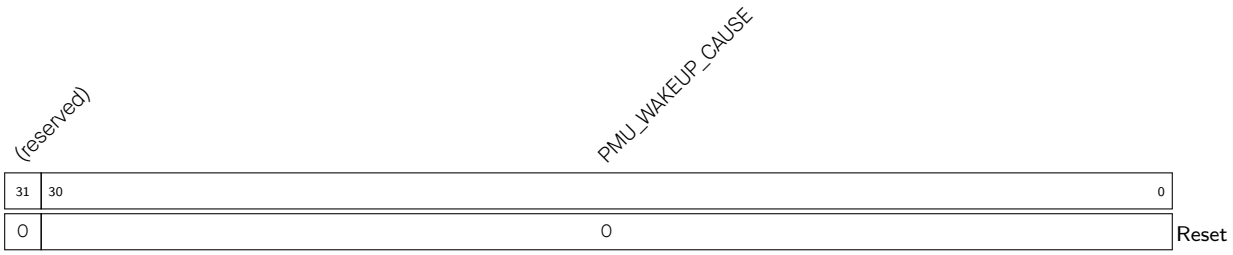
(R/W)

Register 15.38. PMU_SLP_WAKEUP_CNTL4_REG (0x0130)



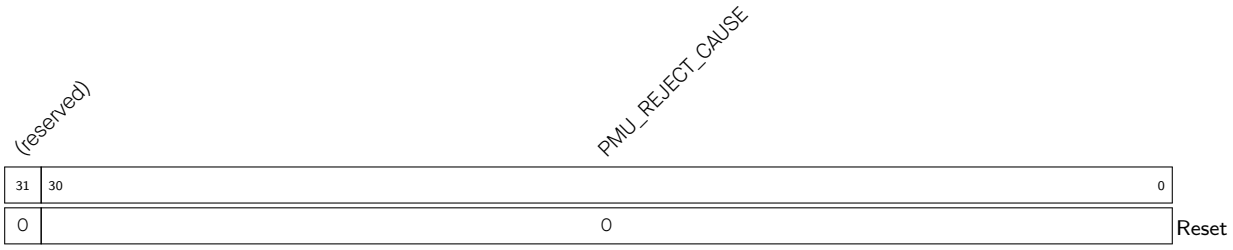
PMU_SLP_REJECT_CAUSE_CLR Write 1 to clear PMU_REJECT_CAUSE. (WT)

Register 15.39. PMU_SLP_WAKEUP_STATUS0_REG (0x0144)



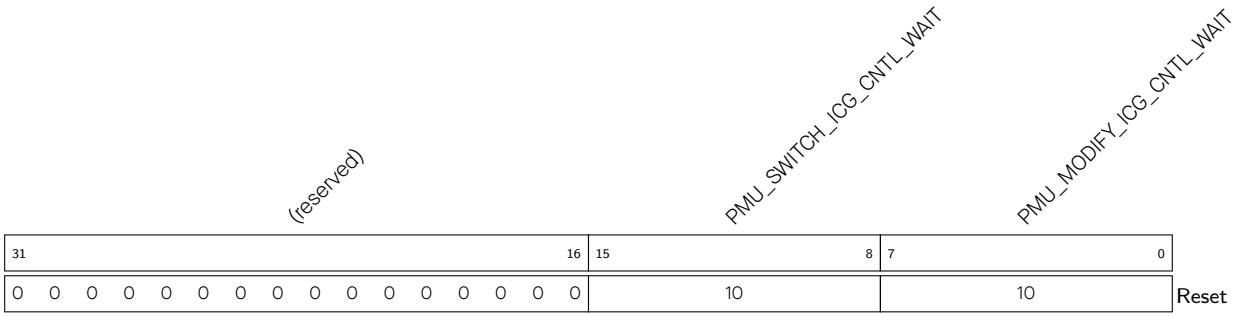
PMU_WAKEUP_CAUSE Indicates the wake-up source. For the mapping between values and sources please refer to Table 15.4-1. (RO)

Register 15.40. PMU_SLP_WAKEUP_STATUS1_REG (0x0148)



PMU_REJECT_CAUSE Indicates the wake-up rejection source. For the mapping between values and sources please refer to Table 15.4-1. (RO)

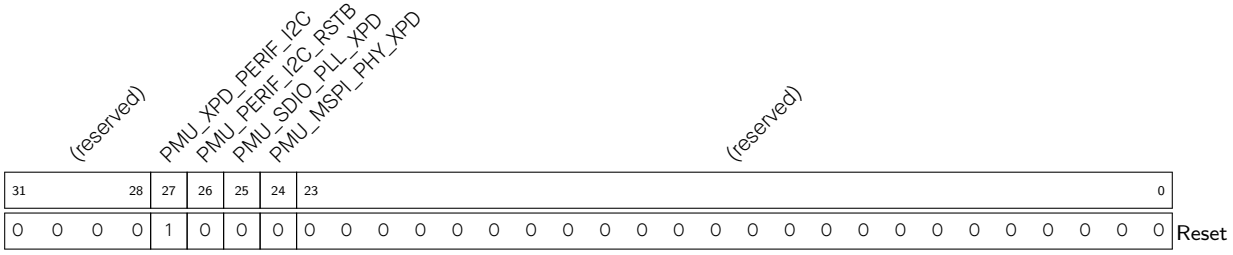
Register 15.41. PMU_HP_CK_CNTL_REG (0x0154)



PMU_MODIFY_ICG_CNTL_WAIT Configures the wait time for modifying the functional module clock gating. The unit is LP_DYN_FAST_CLK. (R/W)

PMU_SWITCH_ICG_CNTL_WAIT Configures the wait time for switching the functional module clock gating. The unit is LP_DYN_FAST_CLK. (R/W)

Register 15.42. PMU_RF_PWC_REG (0x015C)



PMU_MSPI_PHY_XPD Configures whether to enable MSPI PHY.

0: Disable

1: Enable

(R/W)

PMU_SDIO_PLL_XPD Configures whether to enable SDIO PLL.

0: Disable

1: Enable

(R/W)

PMU_PERIF_I2C_RSTB Configures whether to enable reset for PERI I2C.

0: Reset

1: Release

(R/W)

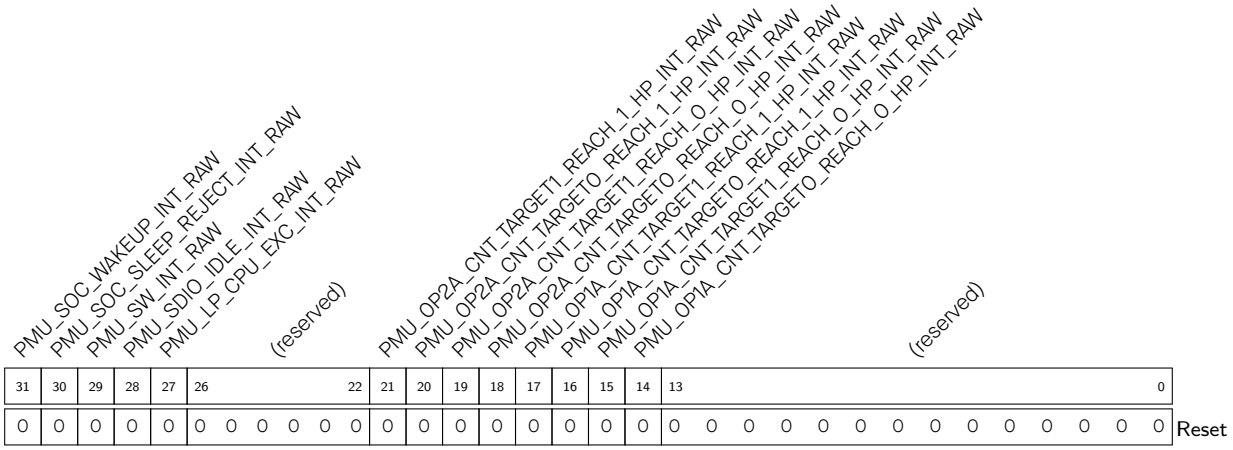
PMU_XPD_PERIF_I2C Configures whether to enable the power for PERI ADC.

0: Disable

1: Enable

(R/W)

Register 15.43. PMU_INT_RAW_REG (0x0164)



PMU_OP1A_CNT_TARGET0_REACH_0_HP_INT_RAW The raw interrupt status of PMU_OP1A_CNT_TARGET0_REACH_0_INT. (R/WTC/SS)

PMU_OP1A_CNT_TARGET1_REACH_0_HP_INT_RAW The raw interrupt status of PMU_OP1A_CNT_TARGET1_REACH_0_INT. (R/WTC/SS)

PMU_OP1A_CNT_TARGET0_REACH_1_HP_INT_RAW The raw interrupt status of PMU_OP1A_CNT_TARGET0_REACH_1_INT. (R/WTC/SS)

PMU_OP1A_CNT_TARGET1_REACH_1_HP_INT_RAW The raw interrupt status of PMU_OP1A_CNT_TARGET1_REACH_1_INT. (R/WTC/SS)

PMU_OP2A_CNT_TARGET0_REACH_0_HP_INT_RAW The raw interrupt status of PMU_OP2A_CNT_TARGET0_REACH_0_INT. (R/WTC/SS)

PMU_OP2A_CNT_TARGET1_REACH_0_HP_INT_RAW The raw interrupt status of PMU_OP2A_CNT_TARGET1_REACH_0_INT. (R/WTC/SS)

PMU_OP2A_CNT_TARGET0_REACH_1_HP_INT_RAW The raw interrupt status of PMU_OP2A_CNT_TARGET0_REACH_1_INT. (R/WTC/SS)

PMU_OP2A_CNT_TARGET1_REACH_1_HP_INT_RAW The raw interrupt status of PMU_OP2A_CNT_TARGET1_REACH_1_INT. (R/WTC/SS)

PMU_LP_CPU_EXC_INT_RAW The raw interrupt status of PMU_LP_CPU_EXC_INT. (R/WTC/SS)

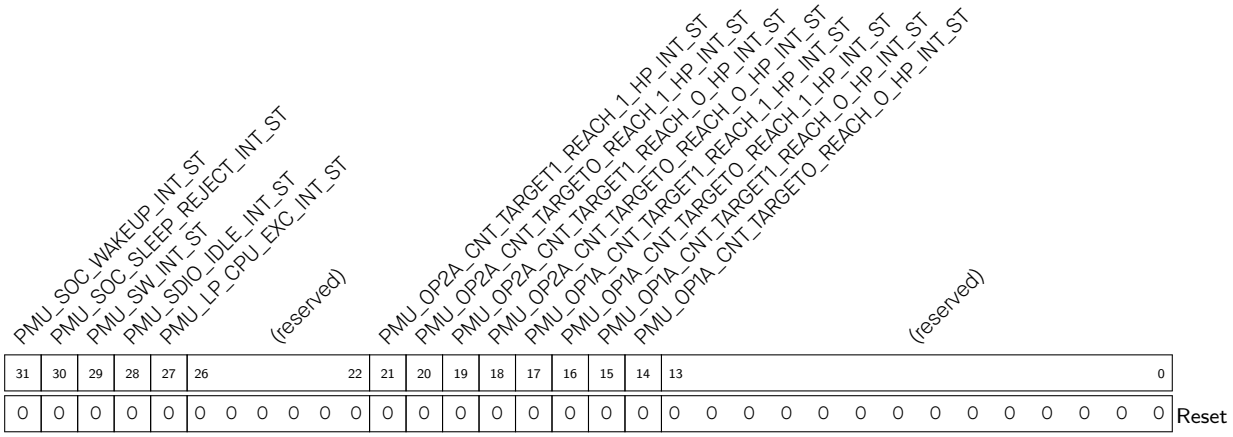
PMU_SDIO_IDLE_INT_RAW The raw interrupt status of PMU_SDIO_IDLE_INT. (R/WTC/SS)

PMU_SW_INT_RAW The raw interrupt status of PMU_SW_INT. (R/WTC/SS)

PMU_SOC_SLEEP_REJECT_INT_RAW The raw interrupt status of PMU_SOC_SLEEP_REJECT_INT. (R/WTC/SS)

PMU_SOC_WAKEUP_INT_RAW The raw interrupt status of PMU_SOC_WAKEUP_INT. (R/WTC/SS)

Register 15.44. PMU_HP_INT_ST_REG (0x0168)



PMU_OP1A_CNT_TARGET0_REACH_0_HP_INT_ST The masked interrupt status of PMU_OP1A_CNT_TARGET0_REACH_0_INT. (RO)

PMU_OP1A_CNT_TARGET1_REACH_0_HP_INT_ST The masked interrupt status of PMU_OP1A_CNT_TARGET1_REACH_0_INT. (RO)

PMU_OP1A_CNT_TARGET0_REACH_1_HP_INT_ST The masked interrupt status of PMU_OP1A_CNT_TARGET0_REACH_1_INT. (RO)

PMU_OP1A_CNT_TARGET1_REACH_1_HP_INT_ST The masked interrupt status of PMU_OP1A_CNT_TARGET1_REACH_1_INT. (RO)

PMU_OP2A_CNT_TARGET0_REACH_0_HP_INT_ST The masked interrupt status of PMU_OP2A_CNT_TARGET0_REACH_0_INT. (RO)

PMU_OP2A_CNT_TARGET1_REACH_0_HP_INT_ST The masked interrupt status of PMU_OP2A_CNT_TARGET1_REACH_0_INT. (RO)

PMU_OP2A_CNT_TARGET0_REACH_1_HP_INT_ST The masked interrupt status of PMU_OP2A_CNT_TARGET0_REACH_1_INT. (RO)

PMU_OP2A_CNT_TARGET1_REACH_1_HP_INT_ST The masked interrupt status of PMU_OP2A_CNT_TARGET1_REACH_1_INT. (RO)

PMU_LP_CPU_EXC_INT_ST The masked interrupt status of PMU_LP_CPU_EXC_INT. (RO)

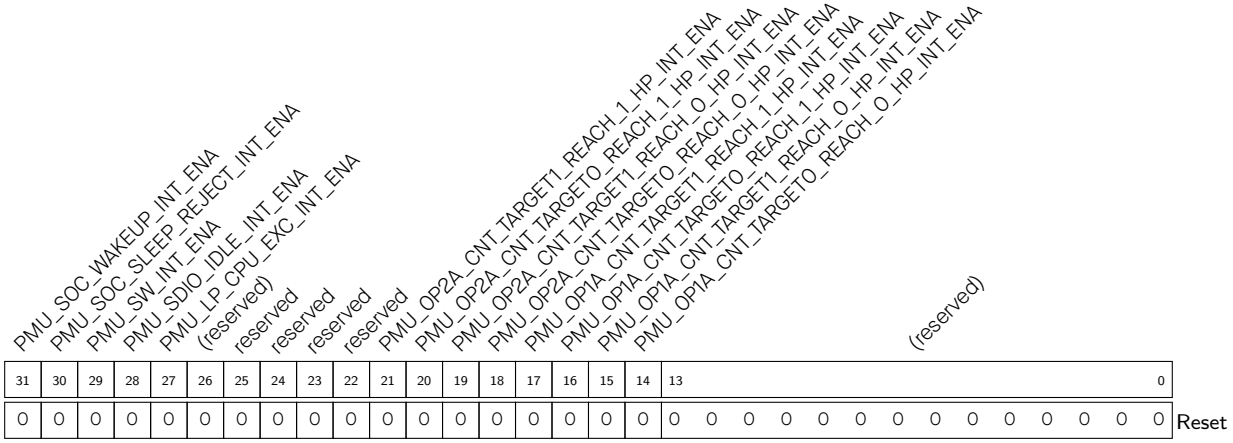
PMU_SDIO_IDLE_INT_ST The masked interrupt status of PMU_SDIO_IDLE_INT. (RO)

PMU_SW_INT_ST The masked interrupt status of PMU_SW_INT. (RO)

PMU_SOC_SLEEP_REJECT_INT_ST The masked interrupt status of PMU_SOC_SLEEP_REJECT_INT. (RO)

PMU_SOC_WAKEUP_INT_ST The masked interrupt status of PMU_SOC_WAKEUP_INT. (RO)

Register 15.45. PMU_HP_INT_ENA_REG (0x016C)



PMU_OP1A_CNT_TARGET0_REACH_0_HP_INT_ENA Write 1 to enable PMU_OP1A_CNT_TARGET0_REACH_0_INT. (R/W)

PMU_OP1A_CNT_TARGET1_REACH_0_HP_INT_ENA Write 1 to enable PMU_OP1A_CNT_TARGET1_REACH_0_INT. (R/W)

PMU_OP1A_CNT_TARGET0_REACH_1_HP_INT_ENA Write 1 to enable PMU_OP1A_CNT_TARGET0_REACH_1_INT. (R/W)

PMU_OP1A_CNT_TARGET1_REACH_1_HP_INT_ENA Write 1 to enable PMU_OP1A_CNT_TARGET1_REACH_1_INT. (R/W)

PMU_OP2A_CNT_TARGET0_REACH_0_HP_INT_ENA Write 1 to enable PMU_OP2A_CNT_TARGET0_REACH_0_INT. (R/W)

PMU_OP2A_CNT_TARGET1_REACH_0_HP_INT_ENA Write 1 to enable PMU_OP2A_CNT_TARGET1_REACH_0_INT. (R/W)

PMU_OP2A_CNT_TARGET0_REACH_1_HP_INT_ENA Write 1 to enable PMU_OP2A_CNT_TARGET0_REACH_1_INT. (R/W)

PMU_OP2A_CNT_TARGET1_REACH_1_HP_INT_ENA Write 1 to enable PMU_OP2A_CNT_TARGET1_REACH_1_INT. (R/W)

PMU_LP_CPU_EXC_INT_ENA Write 1 to enable PMU_LP_CPU_EXC_INT. (R/W)

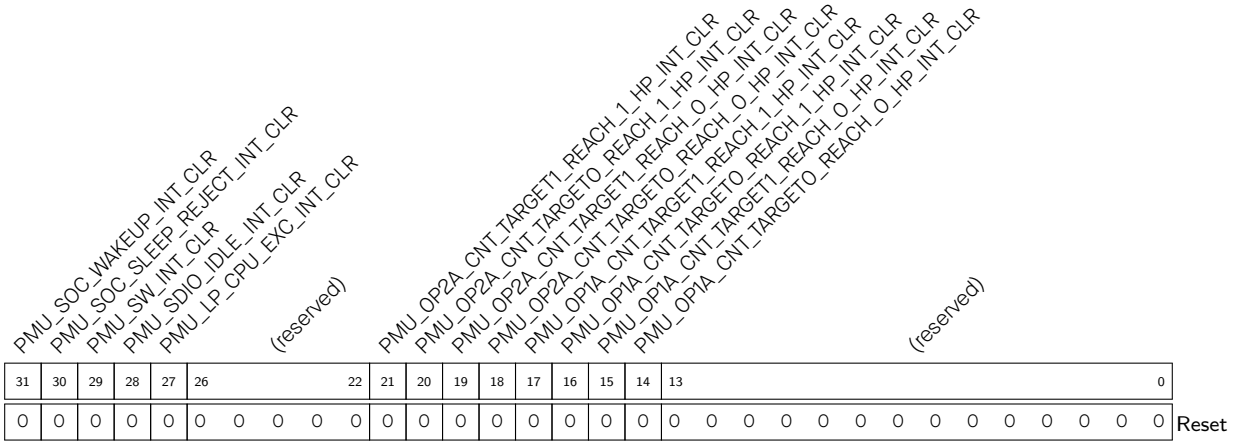
PMU_SDIO_IDLE_INT_ENA Write 1 to enable PMU_SDIO_IDLE_INT. (R/W)

PMU_SW_INT_ENA Write 1 to enable PMU_SW_INT. (R/W)

PMU_SOC_SLEEP_REJECT_INT_ENA Write 1 to enable PMU_SOC_SLEEP_REJECT_INT. (R/W)

PMU_SOC_WAKEUP_INT_ENA Write 1 to enable PMU_SOC_WAKEUP_INT. (R/W)

Register 15.46. PMU_HP_INT_CLR_REG (0x0170)



PMU_OP1A_CNT_TARGET0_REACH_0_HP_INT_CLR Write 1 to clear PMU_OP1A_CNT_TARGET0_REACH_0_INT. (WT)

PMU_OP1A_CNT_TARGET1_REACH_0_HP_INT_CLR Write 1 to clear PMU_OP1A_CNT_TARGET1_REACH_0_INT. (WT)

PMU_OP1A_CNT_TARGET0_REACH_1_HP_INT_CLR Write 1 to clear PMU_OP1A_CNT_TARGET0_REACH_1_INT. (WT)

PMU_OP1A_CNT_TARGET1_REACH_1_HP_INT_CLR Write 1 to clear PMU_OP1A_CNT_TARGET1_REACH_1_INT. (WT)

PMU_OP2A_CNT_TARGET0_REACH_0_HP_INT_CLR Write 1 to clear VPMU_OP2A_CNT_TARGET0_REACH_0_INT. (WT)

PMU_OP2A_CNT_TARGET1_REACH_0_HP_INT_CLR Write 1 to clear PMU_OP2A_CNT_TARGET1_REACH_0_INT. (WT)

PMU_OP2A_CNT_TARGET0_REACH_1_HP_INT_CLR Write 1 to clear PMU_OP2A_CNT_TARGET0_REACH_1_INT. (WT)

PMU_OP2A_CNT_TARGET1_REACH_1_HP_INT_CLR Write 1 to clear PMU_OP2A_CNT_TARGET1_REACH_1_INT. (WT)

PMU_LP_CPU_EXC_INT_CLR Write 1 to clear PMU_LP_CPU_EXC_INT. (WT)

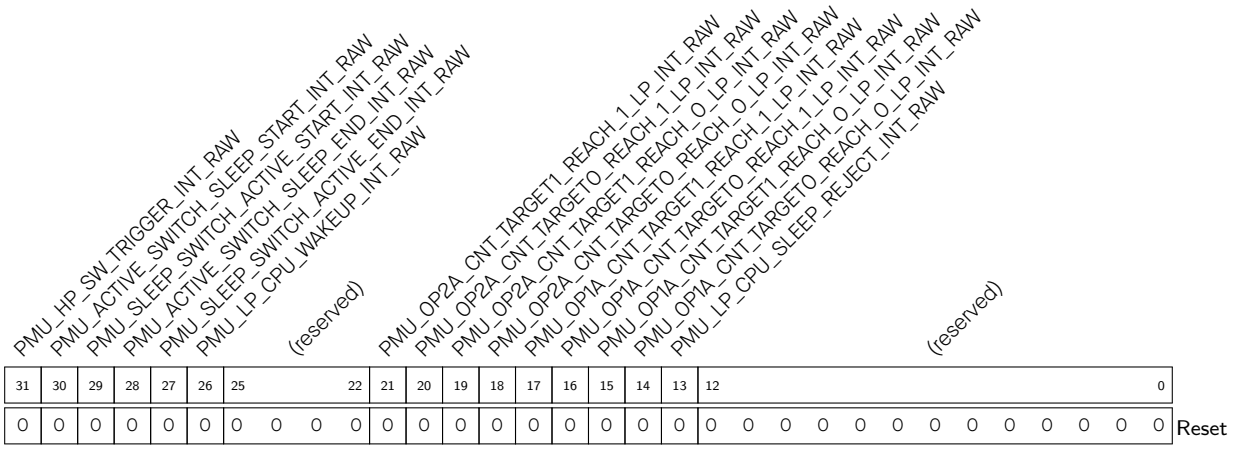
PMU_SDIO_IDLE_INT_CLR Write 1 to clear PMU_SDIO_IDLE_INT. (WT)

PMU_SW_INT_CLR Write 1 to clear PMU_SW_INT. (WT)

PMU_SOC_SLEEP_REJECT_INT_CLR Write 1 to clear PMU_SOC_SLEEP_REJECT_INT. (WT)

PMU_SOC_WAKEUP_INT_CLR Write 1 to clear PMU_SOC_WAKEUP_INT. (WT)

Register 15.47. PMU_LP_INT_RAW_REG (0x0174)



PMU_LP_CPU_SLEEP_REJECT_INT_RAW The raw interrupt status of PMU_LP_CPU_SLEEP_REJECT_INT. (R/WTC/SS)

PMU_OP1A_CNT_TARGET0_REACH_0_LP_INT_RAW The raw interrupt status of PMU_OP1A_CNT_TARGET0_REACH_0_LP_INT. (R/WTC/SS)

PMU_OP1A_CNT_TARGET1_REACH_0_LP_INT_RAW The raw interrupt status of PMU_OP1A_CNT_TARGET1_REACH_0_LP_INT. (R/WTC/SS)

PMU_OP1A_CNT_TARGET0_REACH_1_LP_INT_RAW The raw interrupt status of PMU_OP1A_CNT_TARGET0_REACH_1_LP_INT. (R/WTC/SS)

PMU_OP1A_CNT_TARGET1_REACH_1_LP_INT_RAW The raw interrupt status of PMU_OP1A_CNT_TARGET1_REACH_1_LP_INT. (R/WTC/SS)

PMU_OP2A_CNT_TARGET0_REACH_0_LP_INT_RAW The raw interrupt status of PMU_OP2A_CNT_TARGET0_REACH_0_LP_INT. (R/WTC/SS)

PMU_OP2A_CNT_TARGET1_REACH_0_LP_INT_RAW The raw interrupt status of PMU_OP2A_CNT_TARGET1_REACH_0_LP_INT. (R/WTC/SS)

PMU_OP2A_CNT_TARGET0_REACH_1_LP_INT_RAW The raw interrupt status of PMU_OP2A_CNT_TARGET0_REACH_1_LP_INT. (R/WTC/SS)

PMU_OP2A_CNT_TARGET1_REACH_1_LP_INT_RAW The raw interrupt status of PMU_OP2A_CNT_TARGET1_REACH_1_LP_INT. (R/WTC/SS)

PMU_LP_CPU_WAKEUP_INT_RAW The raw interrupt status of PMU_LP_CPU_WAKEUP_INT. (R/WTC/SS)

PMU_SLEEP_SWITCH_ACTIVE_END_INT_RAW The raw interrupt status of PMU_SLEEP_SWITCH_ACTIVE_END_INT. (R/WTC/SS)

PMU_ACTIVE_SWITCH_SLEEP_END_INT_RAW The raw interrupt status of PMU_ACTIVE_SWITCH_SLEEP_END_INT. (R/WTC/SS)

PMU_SLEEP_SWITCH_ACTIVE_START_INT_RAW The raw interrupt status of PMU_SLEEP_SWITCH_ACTIVE_START_INT. (R/WTC/SS)

Continued on the next page...

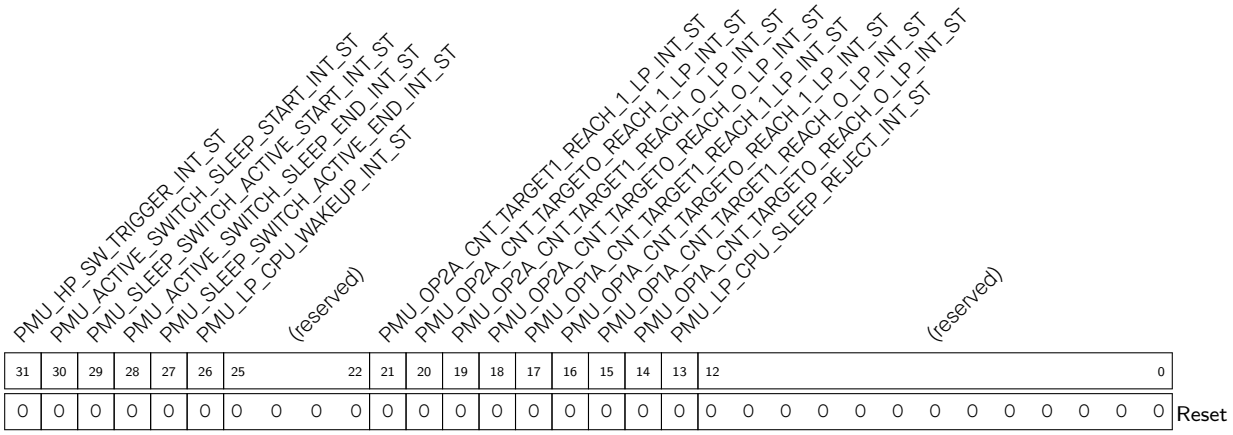
Register 15.47. PMU_LP_INT_RAW_REG (0x0174)

Continued from the previous page...

PMU_ACTIVE_SWITCH_SLEEP_START_INT_RAW The raw interrupt status of PMU_ACTIVE_SWITCH_SLEEP_START_INT. (R/WTC/SS)

PMU_HP_SW_TRIGGER_INT_RAW The raw interrupt status of PMU_HP_SW_TRIGGER_INT. (R/WTC/SS)

Register 15.48. PMU_LP_INT_ST_REG (0x0178)



PMU_LP_CPU_SLEEP_REJECT_INT_ST The masked interrupt status of PMU_LP_CPU_SLEEP_REJECT_INT. (RO)

PMU_OP1A_CNT_TARGET0_REACH_0_LP_INT_ST The masked interrupt status of PMU_OP1A_CNT_TARGET0_REACH_0_LP_INT. (RO)

PMU_OP1A_CNT_TARGET1_REACH_0_LP_INT_ST The masked interrupt status of PMU_OP1A_CNT_TARGET1_REACH_0_LP_INT. (RO)

PMU_OP1A_CNT_TARGET0_REACH_1_LP_INT_ST The masked interrupt status of PMU_OP1A_CNT_TARGET0_REACH_1_LP_INT. (RO)

PMU_OP1A_CNT_TARGET1_REACH_1_LP_INT_ST The masked interrupt status of PMU_OP1A_CNT_TARGET1_REACH_1_LP_INT. (RO)

PMU_OP2A_CNT_TARGET0_REACH_0_LP_INT_ST The masked interrupt status of PMU_OP2A_CNT_TARGET0_REACH_0_LP_INT. (RO)

PMU_OP2A_CNT_TARGET1_REACH_0_LP_INT_ST The masked interrupt status of PMU_OP2A_CNT_TARGET1_REACH_0_LP_INT. (RO)

PMU_OP2A_CNT_TARGET0_REACH_1_LP_INT_ST The masked interrupt status of PMU_OP2A_CNT_TARGET0_REACH_1_LP_INT. (RO)

PMU_OP2A_CNT_TARGET1_REACH_1_LP_INT_ST The masked interrupt status of PMU_OP2A_CNT_TARGET1_REACH_1_LP_INT. (RO)

PMU_LP_CPU_WAKEUP_INT_ST The masked interrupt status of PMU_LP_CPU_WAKEUP_INT. (RO)

PMU_SLEEP_SWITCH_ACTIVE_END_INT_ST The masked interrupt status of PMU_SLEEP_SWITCH_ACTIVE_END_INT. (RO)

PMU_ACTIVE_SWITCH_SLEEP_END_INT_ST The masked interrupt status of PMU_ACTIVE_SWITCH_SLEEP_END_INT. (RO)

PMU_SLEEP_SWITCH_ACTIVE_START_INT_ST The masked interrupt status of PMU_SLEEP_SWITCH_ACTIVE_START_INT. (RO)

Continued on the next page...

Register 15.48. PMU_LP_INT_ST_REG (0x0178)

Continued from the previous page...

PMU_ACTIVE_SWITCH_SLEEP_START_INT_ST The masked interrupt status of PMU_ACTIVE_SWITCH_SLEEP_START_INT. (RO)

PMU_HP_SW_TRIGGER_INT_ST The masked interrupt status of PMU_HP_SW_TRIGGER_INT. (RO)

Register 15.49. PMU_LP_INT_ENA_REG (0x017C)



PMU_LP_CPU_SLEEP_REJECT_INT_ENA Write 1 to enable PMU_LP_CPU_SLEEP_REJECT_INT. (R/W)

PMU_OP1A_CNT_TARGET0_REACH_0_LP_INT_ENA Write 1 to enable PMU_OP1A_CNT_TARGET0_REACH_0_LP_INT. (R/W)

PMU_OP1A_CNT_TARGET1_REACH_0_LP_INT_ENA Write 1 to enable PMU_OP1A_CNT_TARGET1_REACH_0_LP_INT. (R/W)

PMU_OP1A_CNT_TARGET0_REACH_1_LP_INT_ENA Write 1 to enable PMU_OP1A_CNT_TARGET0_REACH_1_LP_INT. (R/W)

PMU_OP1A_CNT_TARGET1_REACH_1_LP_INT_ENA Write 1 to enable PMU_OP1A_CNT_TARGET1_REACH_1_LP_INT. (R/W)

PMU_OP2A_CNT_TARGET0_REACH_0_LP_INT_ENA Write 1 to enable PMU_OP2A_CNT_TARGET0_REACH_0_LP_INT. (R/W)

PMU_OP2A_CNT_TARGET1_REACH_0_LP_INT_ENA Write 1 to enable PMU_OP2A_CNT_TARGET1_REACH_0_LP_INT. (R/W)

PMU_OP2A_CNT_TARGET0_REACH_1_LP_INT_ENA Write 1 to enable PMU_OP2A_CNT_TARGET0_REACH_1_LP_INT. (R/W)

PMU_OP2A_CNT_TARGET1_REACH_1_LP_INT_ENA Write 1 to enable PMU_OP2A_CNT_TARGET1_REACH_1_LP_INT. (R/W)

PMU_LP_CPU_WAKEUP_INT_ENA Write 1 to enable PMU_LP_CPU_WAKEUP_INT. (R/W)

PMU_SLEEP_SWITCH_ACTIVE_END_INT_ENA Write 1 to enable PMU_SLEEP_SWITCH_ACTIVE_END_INT. (R/W)

PMU_ACTIVE_SWITCH_SLEEP_END_INT_ENA Write 1 to enable PMU_ACTIVE_SWITCH_SLEEP_END_INT. (R/W)

PMU_SLEEP_SWITCH_ACTIVE_START_INT_ENA Write 1 to enable PMU_SLEEP_SWITCH_ACTIVE_START_INT. (R/W)

Continued on the next page...

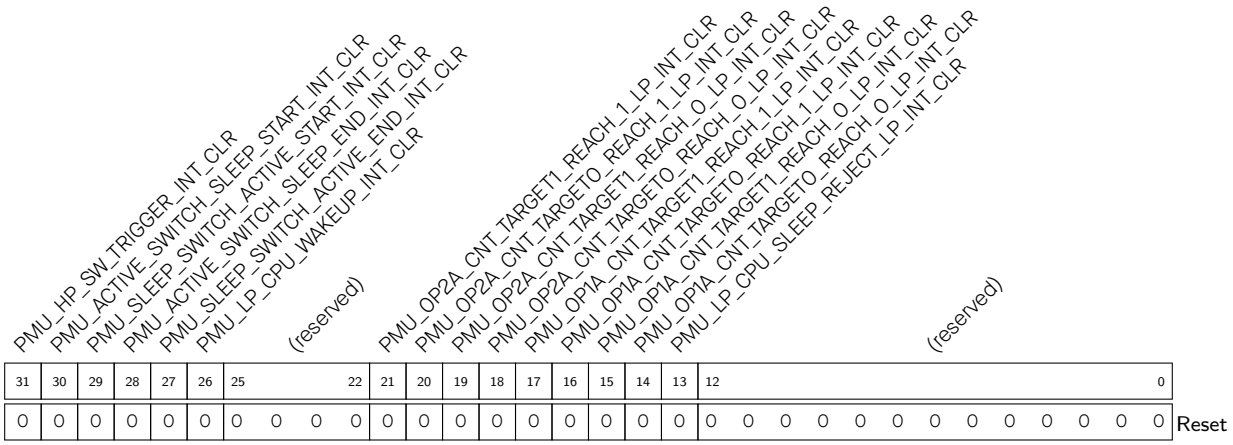
Register 15.49. PMU_LP_INT_ENA_REG (0x017C)

Continued from the previous page...

PMU_ACTIVE_SWITCH_SLEEP_START_INT_ENA Write 1 to enable PMU_ACTIVE_SWITCH_SLEEP_START_INT. (R/W)

PMU_HP_SW_TRIGGER_INT_ENA Write 1 to enable PMU_HP_SW_TRIGGER_INT. (R/W)

Register 15.50. PMU_LP_INT_CLR_REG (0x0180)



PMU_LP_CPU_SLEEP_REJECT_LP_INT_CLR Write 1 to clear PMU_LP_CPU_SLEEP_REJECT_INT. (R/W)

PMU_OP1A_CNT_TARGET0_REACH_0_LP_INT_CLR Write 1 to clear PMU_OP1A_CNT_TARGET0_REACH_0_LP_INT. (WT)

PMU_OP1A_CNT_TARGET1_REACH_0_LP_INT_CLR Write 1 to clear PMU_OP1A_CNT_TARGET1_REACH_0_LP_INT. (WT)

PMU_OP1A_CNT_TARGET0_REACH_1_LP_INT_CLR Write 1 to clear PMU_OP1A_CNT_TARGET0_REACH_1_LP_INT. (WT)

PMU_OP1A_CNT_TARGET1_REACH_1_LP_INT_CLR Write 1 to clear PMU_OP1A_CNT_TARGET1_REACH_1_LP_INT. (WT)

PMU_OP2A_CNT_TARGET0_REACH_0_LP_INT_CLR Write 1 to clear PMU_OP2A_CNT_TARGET0_REACH_0_LP_INT. (WT)

PMU_OP2A_CNT_TARGET1_REACH_0_LP_INT_CLR Write 1 to clear PMU_OP2A_CNT_TARGET1_REACH_0_LP_INT. (WT)

PMU_OP2A_CNT_TARGET0_REACH_1_LP_INT_CLR Write 1 to clear PMU_OP2A_CNT_TARGET0_REACH_1_LP_INT. (WT)

PMU_OP2A_CNT_TARGET1_REACH_1_LP_INT_CLR Write 1 to clear PMU_OP2A_CNT_TARGET1_REACH_1_LP_INT. (WT)

PMU_LP_CPU_WAKEUP_INT_CLR Write 1 to clear PMU_LP_CPU_WAKEUP_INT. (WT)

PMU_SLEEP_SWITCH_ACTIVE_END_INT_CLR Write 1 to clear PMU_SLEEP_SWITCH_ACTIVE_END_INT. (WT)

PMU_ACTIVE_SWITCH_SLEEP_END_INT_CLR Write 1 to clear PMU_ACTIVE_SWITCH_SLEEP_END_INT. (WT)

PMU_SLEEP_SWITCH_ACTIVE_START_INT_CLR Write 1 to clear PMU_SLEEP_SWITCH_ACTIVE_START_INT. (WT)

Continued on the next page...

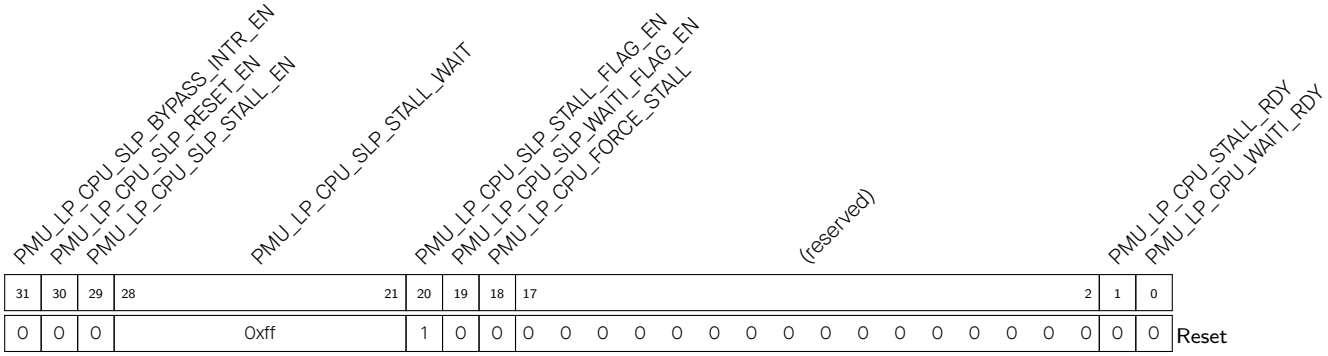
Register 15.50. PMU_LP_INT_CLR_REG (0x0180)

Continued from the previous page...

PMU_ACTIVE_SWITCH_SLEEP_START_INT_CLR Write 1 to clear PMU_ACTIVE_SWITCH_SLEEP_START_INT. (WT)

PMU_HP_SW_TRIGGER_INT_CLR Write 1 to clear PMU_HP_SW_TRIGGER_INT. (WT)

Register 15.51. PMU_LP_CPU_PWR0_REG (0x0184)



PMU_LP_CPU_WAITI_RDY Indicates whether the LP CPU is in the waiti state.

0: Does not enter the state

1: Enters the state

(RO)

PMU_LP_CPU_STALL_RDY Indicates whether the LP CPU is in the stall state.

0: Does not enter the state

1: Enters the state

(RO)

PMU_LP_CPU_FORCE_STALL Configures the LP CPU to enter the stall state.

0: Does not enter the state

1: Enters the state

(R/W)

PMU_LP_CPU_SLP_WAITI_FLAG_EN Configures whether the LP CPU needs to enter the waiti state when entering sleep mode.

0: Does not enter the state

1: Enters the state

(R/W)

PMU_LP_CPU_SLP_STALL_FLAG_EN Configures whether the LP CPU needs to enter the stall state when entering sleep mode.

0: Does not enter the state

1: Enters the state

(R/W)

PMU_LP_CPU_SLP_STALL_WAIT Configures the time to wait for the stall to take effect after enabling stall state when the LP CPU enters sleep mode. The unit is LP_DYN_FAST_CLK. (R/W)

PMU_LP_CPU_SLP_STALL_EN Configures whether to pause the LP CPU in sleep mode.

0: Do not pause

1: Pause

(R/W)

Continued on the next page...

Register 15.51. PMU_LP_CPU_PWR0_REG (0x0184)

Continued from the previous page...

PMU_LP_CPU_SLP_RESET_EN Configures whether to reset the LP CPU in sleep mode.

0: Do not reset

1: Reset

(R/W)

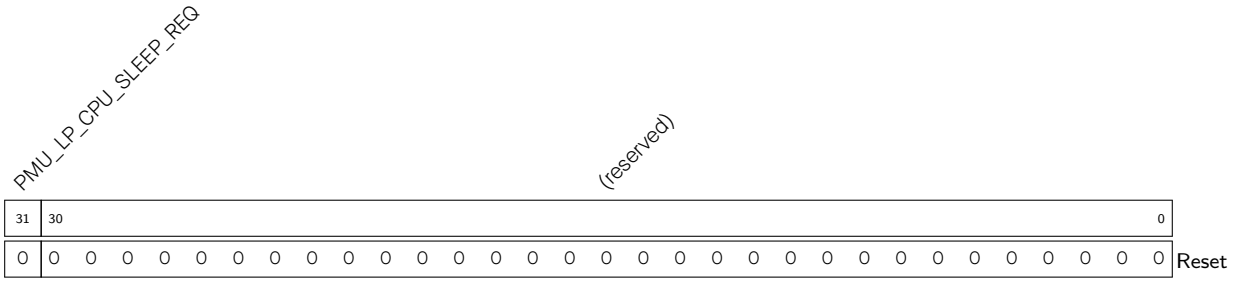
PMU_LP_CPU_SLP_BYPASS_INTR_EN Configures whether to enable interrupt signals for the LP CPU in sleep mode.

0: Disable interrupt signals

1: Enable interrupt signals

(R/W)

Register 15.52. PMU_LP_CPU_PWR1_REG (0x0188)



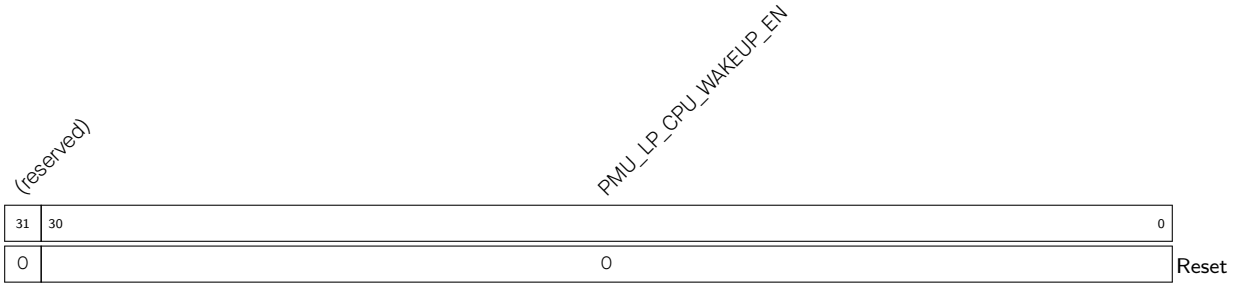
PMU_LP_CPU_SLEEP_REQ Configures whether the LP CPU enters sleep.

0: Does not enter sleep

1: Enters sleep

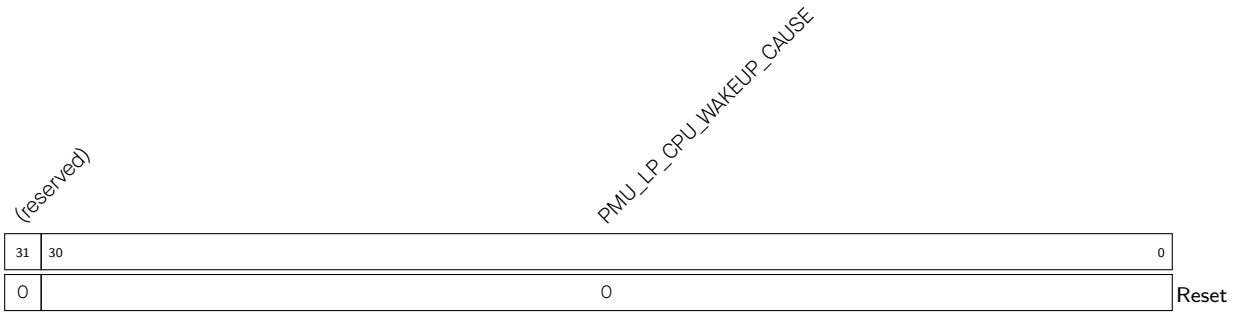
(WT)

Register 15.53. PMU_LP_CPU_PWR2_REG (0x018C)



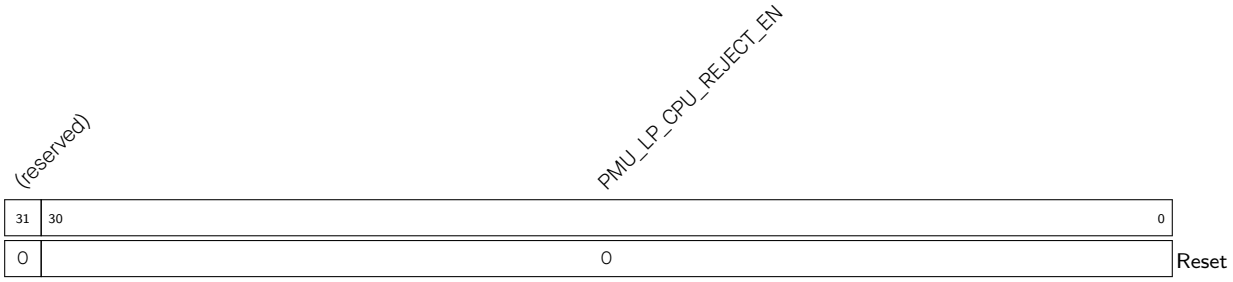
PMU_LP_CPU_WAKEUP_EN Configures the wake-up source for LP CPU. For details please refer to Table 15.4-1 *Wake-up Sources*. (R/W)

Register 15.54. PMU_LP_CPU_PWR3_REG (0x0190)



PMU_LP_CPU_WAKEUP_CAUSE Indicates the wake-up cause of LP CPU. (RO)

Register 15.55. PMU_LP_CPU_PWR4_REG (0x0194)



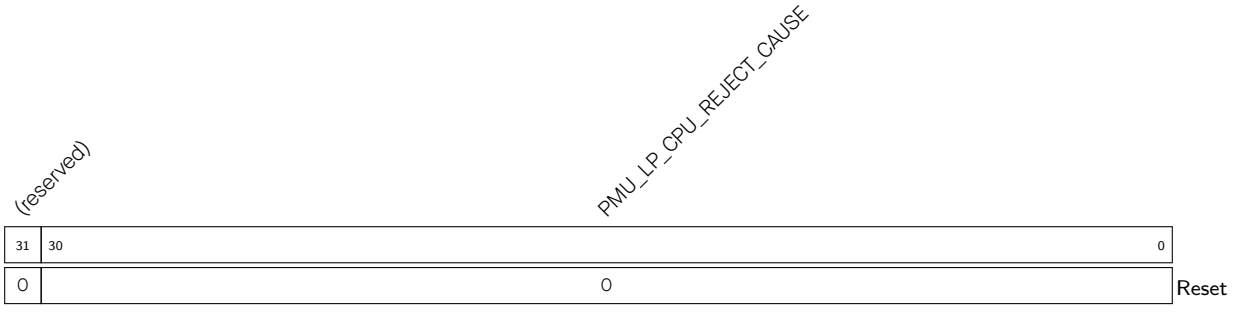
PMU_LP_CPU_REJECT_EN Configures whether to enable sleep rejection function for the LP CPU.

0: Disable

1: Enable

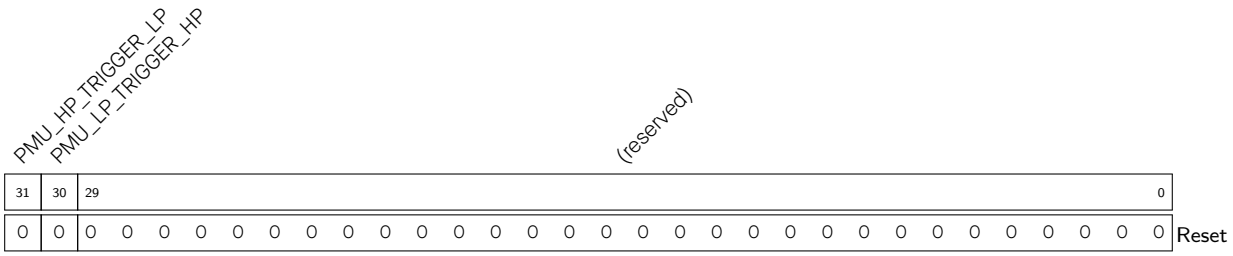
(R/W)

Register 15.56. PMU_LP_CPU_PWR5_REG (0x0198)



PMU_LP_CPU_REJECT_CAUSE Indicates the sleep rejection cause of LP CPU. (RO)

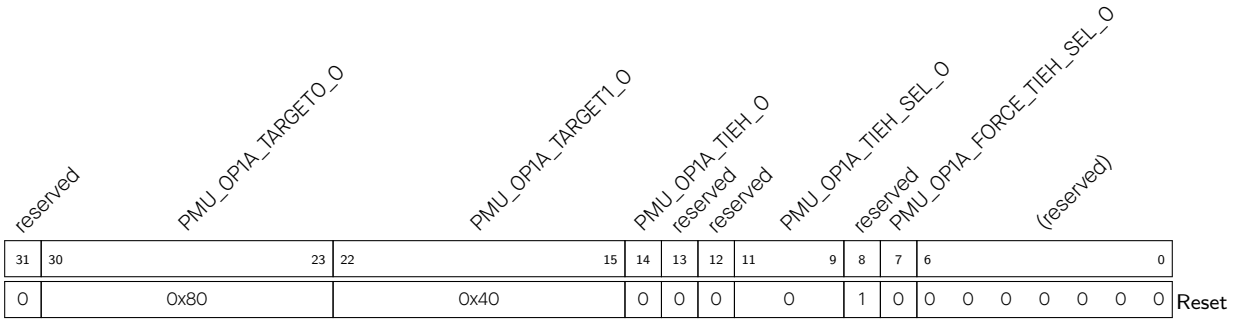
Register 15.57. PMU_HP_LP_CPU_COMM_REG (0x019C)



PMU_LP_TRIGGER_HP When the LP CPU sets this register to 1, the chip is woken up. (WT)

PMU_HP_TRIGGER_LP When the HP CPU sets this register to 1, the LP CPU is woken up. (WT)

Register 15.58. PMU_EXT_LDO_P0_0P1A_REG (0x01B8)



PMU_0P1A_FORCE_TIEH_SEL_0 Configures which of the following controls the mode selection for VO1 regulator.

1: Mode selection is controlled by PMU_0P1A_TIEH_SEL_0

0: Mode selection is controlled by eFuse

(R/W)

PMU_0P1A_TIEH_SEL_0 Configures mode for VO1.

0: Mode is controlled by PMU_0P1A_TIEH_0.

1: Mode is controlled by SDMMC_0

2: Bypass mode

3: Mode is controlled by SDMMC_1

(R/W)

PMU_0P1A_TIEH_0 Configures mode for VO1.

0: LDO mode

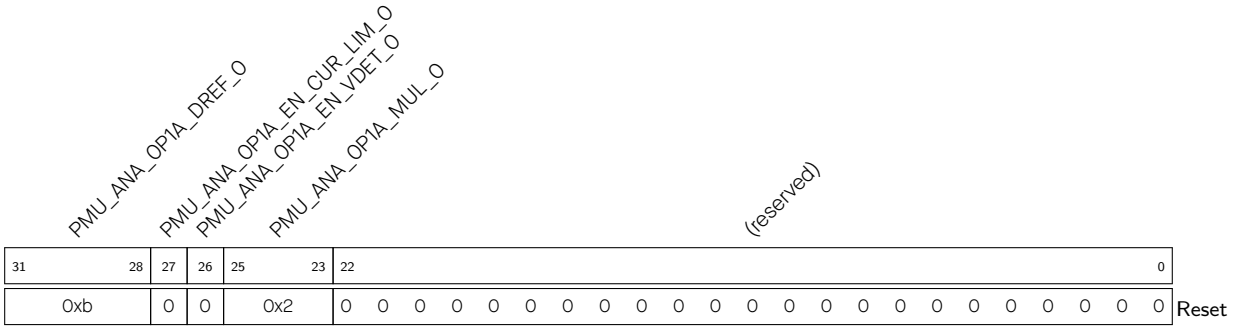
1: Bypass mode

(R/W)

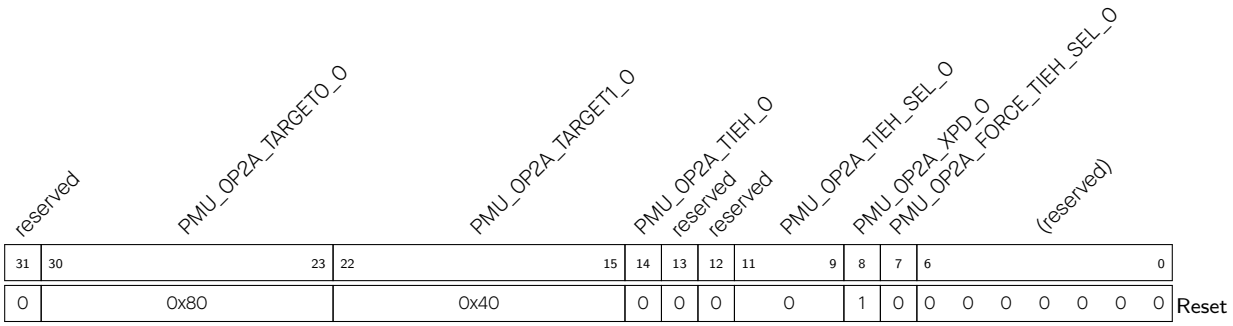
PMU_0P1A_TARGET1_0 Configures the timeout for stage 2 of the VO1 regulator wait counter, with the unit being a 16 division of LP_DYN_SLOW_CLK. (R/W)

PMU_0P1A_TARGET0_0 Configures the timeout for stage 1 of the VO1 regulator wait counter, with the unit being a 16 division of LP_DYN_SLOW_CLK. (R/W)

Register 15.59. PMU_EXT_LDO_P0_0P1A_ANA_REG (0x01BC)


PMU_ANA_0P1A_DREF_0
PMU_ANA_0P1A_EN_CUR_LIM_0
PMU_ANA_0P1A_EN_VDET_0
PMU_ANA_0P1A_MUL_0
(reserved)
31 28 27 26 25 23 22 0
0xb 0 0 0x2 0 0 0

Register 15.60. PMU_EXT_LDO_P0_0P2A_REG (0x01C0)



PMU_0P2A_FORCE_TIEH_SEL_0 Configures which of the following controls the mode selection for VO3 regulator.

0: Mode selection is controlled by PMU_0P2A_TIEH_SEL_0

1: Mode selection is controlled by SDMMC_0

2: Bypass mode

3: Mode selection is controlled by SDMMC_1

(R/W)

PMU_0P2A_XPD_0 Configures whether software enables VO3 regulator.

0: Disable

1: Enable

(R/W)

PMU_0P2A_TIEH_SEL_0 Configures mode for VO3.

0: Mode is controlled by PMU_0P2A_TIEH_0.

1: Mode is controlled by SDMMC_0

2: Bypass mode

3: Mode is controlled by SDMMC_1

(R/W)

PMU_0P2A_TIEH_0 Configures mode for VO3.

0: LDO mode

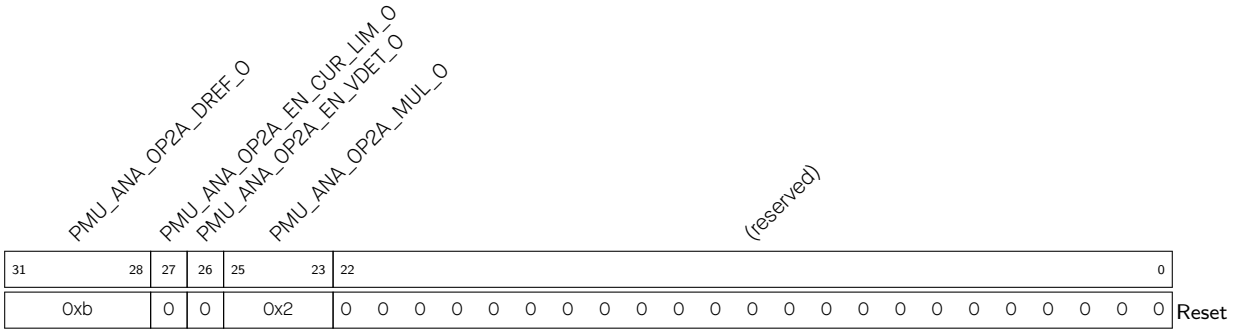
1: Bypass mode

(R/W)

PMU_0P2A_TARGET1_0 Configures the timeout for stage 2 of the VO3 regulator wait counter, with the unit being a 16 division of LP_DYN_SLOW_CLK. (R/W)

PMU_0P2A_TARGET0_0 Configures the timeout for stage 1 of the VO3 regulator wait counter, with the unit being a 16 division of LP_DYN_SLOW_CLK. (R/W)

Register 15.61. PMU_EXT_LDO_P0_0P2A_ANA_REG (0x01C4)



PMU_ANA_0P2A_MUL_0 Configures the MUL, where the output voltage of the regulator is MUL × Vref.

0-7: MUL ranges from 1 to 2.75, with a step of 0.25.

(R/W)

PMU_ANA_0P2A_EN_VDET_0 Configures whether to enable VO3 voltage detection.

0: Do not enable

1: Enable

(R/W)

PMU_ANA_0P2A_EN_CUR_LIM_0 Configures whether to enable over current protection for VO3.

0: Do not enable

1: Enable

(R/W)

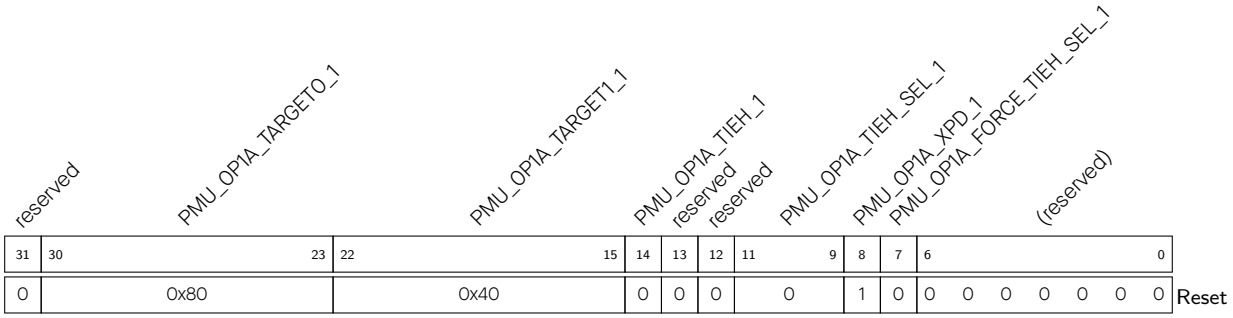
PMU_ANA_0P2A_DREF_0 Configures the LDO Vref, where the output voltage of the regulator is MUL × Vref.

0-8: Vref = 0.5 V - 0.9 V, with a step of 50 mV

9-15: Vref = 1.0 V - 1.6 V, with a step of 100 mV

(R/W)

Register 15.62. PMU_EXT_LDO_P1_0P1A_REG (0x01D0)



PMU_0P1A_FORCE_TIEH_SEL_1 Configures which of the following controls the mode selection for VO2 regulator.

0: Mode selection is controlled by PMU_0P1A_TIEH_SEL_1

1: Mode selection is controlled by SDMMC_0

2: Bypass mode

3: Mode selection is controlled by SDMMC_1

(R/W)

PMU_0P1A_XPD_1 Configures whether software enables VO2 regulator.

0: Disable

1: Enable

(R/W)

PMU_0P1A_TIEH_SEL_1 Configures mode for VO2.

0: Mode is controlled by PMU_0P1A_TIEH_1.

1: Mode is controlled by SDMMC_0

2: Bypass mode

3: Mode is controlled by SDMMC_1

(R/W)

PMU_0P1A_TIEH_1 Configures mode for VO2.

0: LDO mode

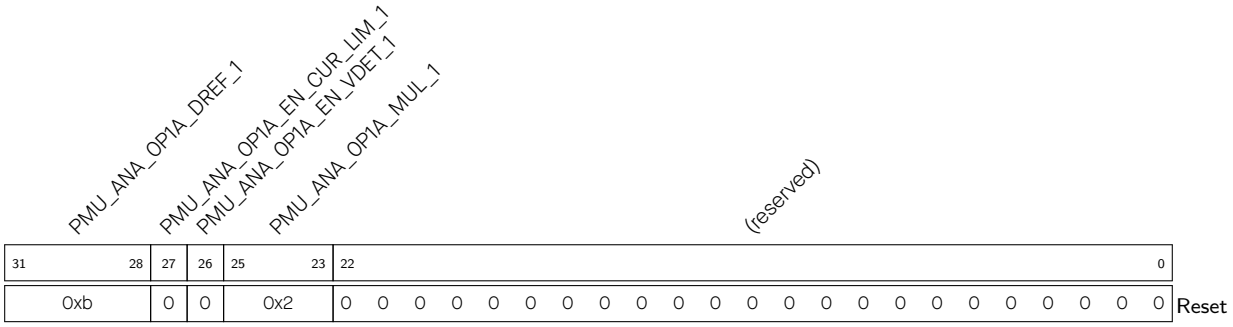
1: Bypass mode

(R/W)

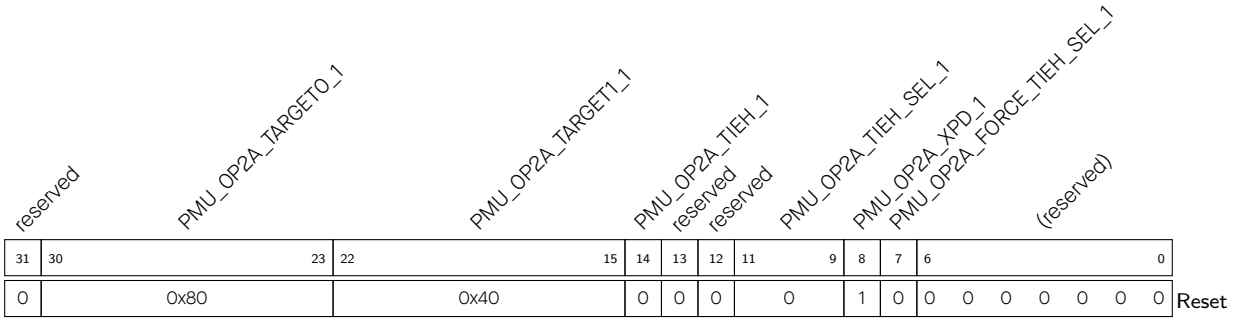
PMU_0P1A_TARGET1_1 Configures the timeout for stage 2 of the VO2 regulator wait counter, with the unit being a 16 division of LP_DYN_SLOW_CLK. (R/W)

PMU_0P1A_TARGET0_1 Configures the timeout for stage 1 of the VO2 regulator wait counter, with the unit being a 16 division of LP_DYN_SLOW_CLK. (R/W)

Register 15.63. PMU_EXT_LDO_P1_0P1A_ANA_REG (0x01D4)


PMU_ANA_0P1A_DREF_1
PMU_ANA_0P1A_EN_CUR_LIM_1
PMU_ANA_0P1A_EN_VDET_1
PMU_ANA_0P1A_MUL_1
(reserved)
31 28 27 26 25 23 22 0
0xb 0 0 0x2 0 0 0

Register 15.64. PMU_EXT_LDO_P1_0P2A_REG (0x01D8)



PMU_0P2A_FORCE_TIEH_SEL_1 Configures which of the following controls the mode selection for VO4 regulator.

1: Mode selection is controlled by PMU_0P2A_TIEH_SEL_1

1: Mode selection is controlled by SDMMC_0

2: Bypass mode

3: Mode selection is controlled by SDMMC_1

(R/W)

PMU_0P2A_XPD_1 Configures whether software enables VO4 regulator.

0: Disable

1: Enable

(R/W)

PMU_0P2A_TIEH_SEL_1 Configures mode for VO4.

0: Mode is controlled by PMU_0P2A_TIEH_1.

1: Mode is controlled by SDMMC_0

2: Bypass mode

3: Mode is controlled by SDMMC_1

(R/W)

PMU_0P2A_TIEH_1 Configures mode for VO4.

0: LDO mode

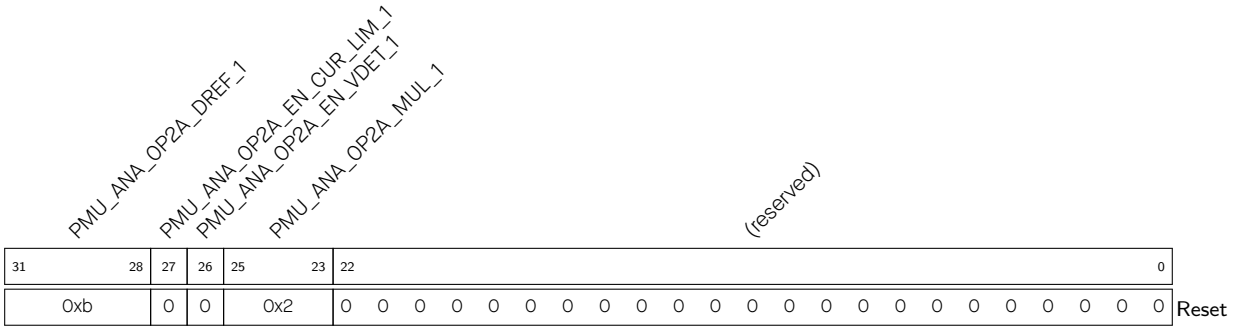
1: Bypass mode

(R/W)

PMU_0P2A_TARGET1_1 Configures the timeout for stage 2 of the VO4 regulator wait counter, with the unit being a 16 division of LP_DYN_SLOW_CLK. (R/W)

PMU_0P2A_TARGET0_1 Configures the timeout for stage 1 of the VO4 regulator wait counter, with the unit being a 16 division of LP_DYN_SLOW_CLK. (R/W)

Register 15.65. PMU_EXT_LDO_P1_0P2A_ANA_REG (0x01DC)



PMU_ANA_0P2A_MUL_1 Configures the MUL, where the output voltage of the regulator is MUL × Vref.

0-7: MUL ranges from 1 to 2.75, with a step of 0.25.

(R/W)

PMU_ANA_0P2A_EN_VDET_1 Configures whether to enable VO4 voltage detection.

0: Do not enable

1: Enable

(R/W)

PMU_ANA_0P2A_EN_CUR_LIM_1 Configures whether to enable over current protection for VO4.

0: Do not enable

1: Enable

(R/W)

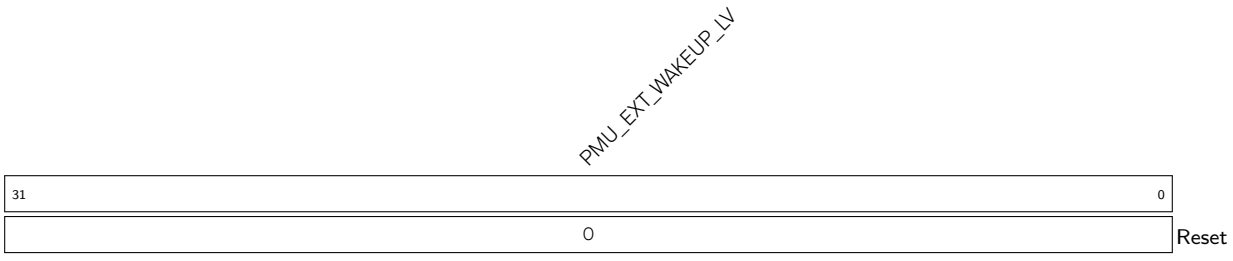
PMU_ANA_0P2A_DREF_1 Configures the LDO Vref, where the output voltage of the regulator is MUL × Vref.

0-8: Vref = 0.5 V - 0.9 V, with a step of 50 mV

9-15: Vref = 1.0 V - 1.6 V, with a step of 100 mV

(R/W)

Register 15.66. PMU_EXT_WAKEUP_LV_REG (0x01E8)



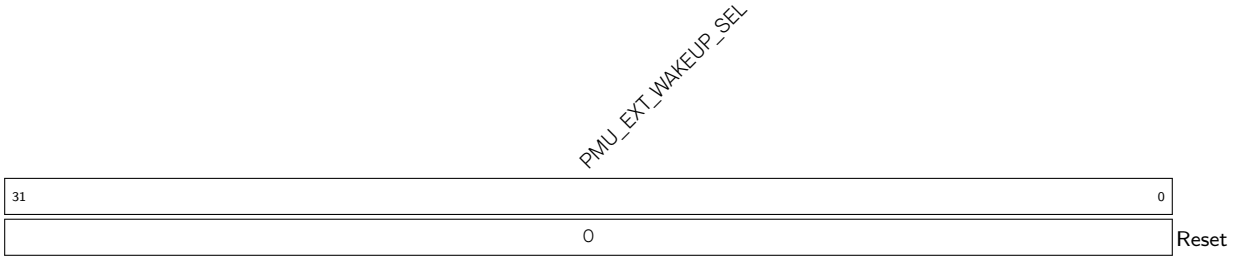
PMU_EXT_WAKEUP_LV Configures the wake level for EXT wake-up.

0: Low level wake-up

1: High level wake-up

(R/W)

Register 15.67. PMU_EXT_WAKEUP_SEL_REG (0x01EC)



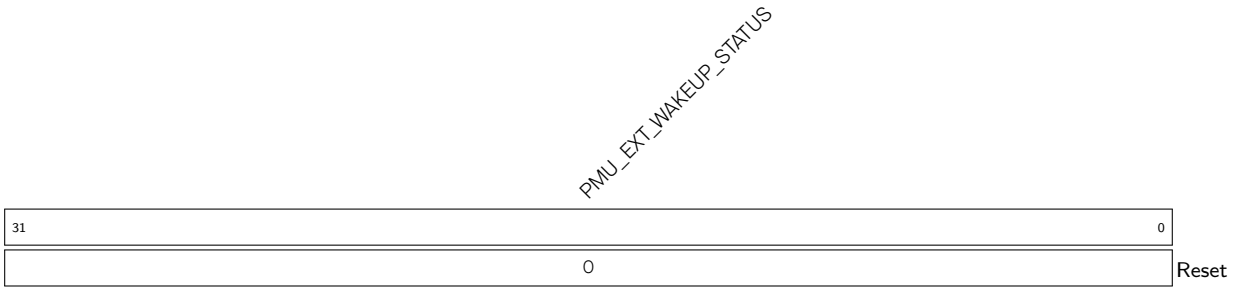
PMU_EXT_WAKEUP_SEL Configures whether to enable EXT wake-up.

0: Disable

1: Enable

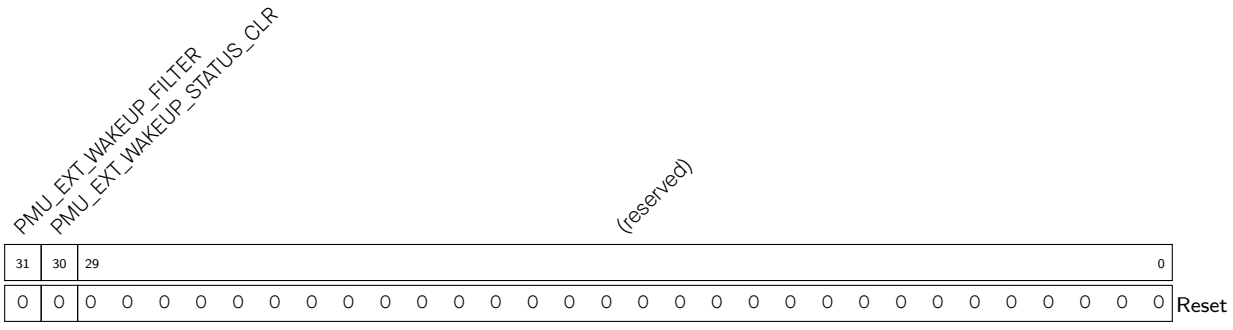
(R/W)

Register 15.68. PMU_EXT_WAKEUP_ST_REG (0x01F0)



PMU_EXT_WAKEUP_STATUS Gets the GPIO number for wake-up. 0 corresponds to GPIO0, 1 corresponds to GPIO1, and so on. (RO)

Register 15.69. PMU_EXT_WAKEUP_CNTL_REG (0x01F4)



PMU_EXT_WAKEUP_STATUS_CLR Clears the wake-up records. (R/W)

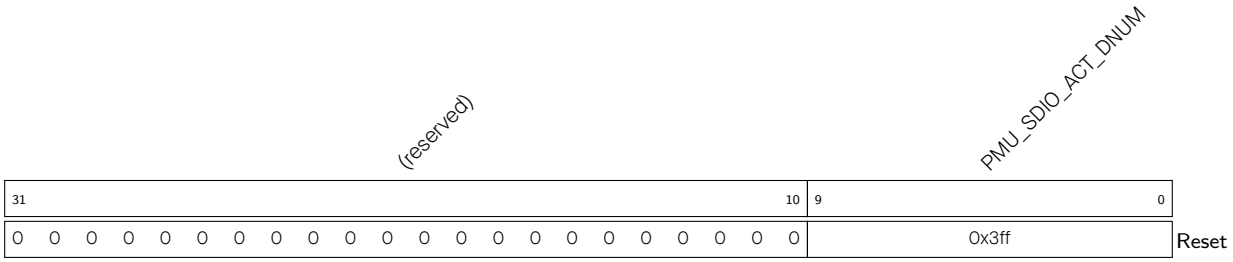
PMU_EXT_WAKEUP_FILTER Configures whether to enable the wake-up filter.

0: Disable

1: Enable

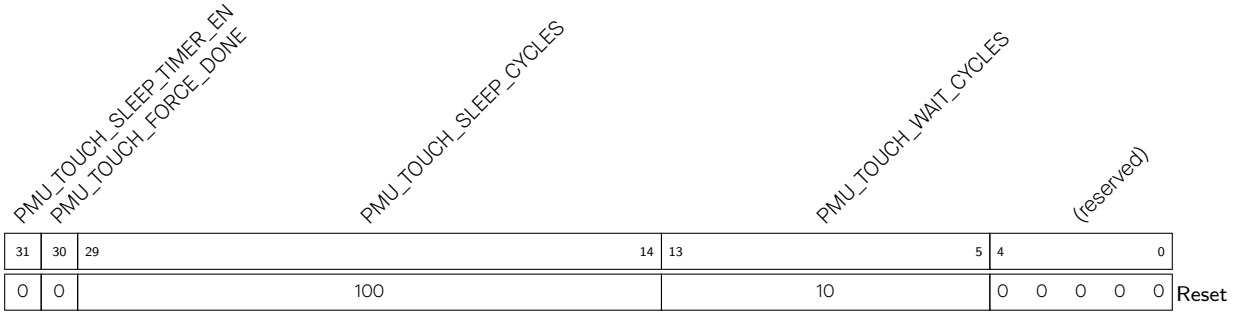
(R/W)

Register 15.70. PMU_SDIO_WAKEUP_CNTL_REG (0x01F8)



PMU_SDIO_ACT_DNUM Configures the duration for maintaining the working state of SDIO. The unit is LP_DYN_FAST_CLK. (R/W)

Register 15.71. PMU_TOUCH_PWR_CNTL_REG (0x0210)



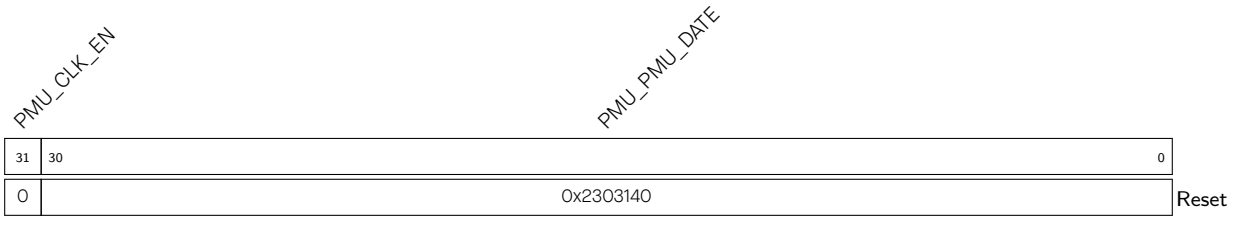
PMU_TOUCH_WAIT_CYCLES Configures the waiting time for the PMU to wake up from the TOUCH timer, with the unit of LP_DYN_SLOW_CLK. (R/W)

PMU_TOUCH_SLEEP_CYCLES Configure the sleep time for TOUCH, with the unit of LP_DYN_SLOW_CLK. (R/W)

PMU_TOUCH_FORCE_DONE Force return to the working state of the TOUCH timer. (R/W)

PMU_TOUCH_SLEEP_TIMER_EN Enables the TOUCH timer to start working. (R/W)

Register 15.72. PMU_DATE_REG (0x03FC)



PMU_PMU_DATE Version control register. (R/W)

PMU_CLK_EN Force enables the automatic clock gating of the register. (R/W)

Chapter 16

System Timer

16.1 Overview

ESP32-P4 provides a 52-bit system timer, which can be used to generate tick interrupts for the operating system, or be used as a general timer to generate periodic interrupts or one-time interrupts.

16.2 Features

The system timer has the following features:

- Two 52-bit counters and three 52-bit comparators
- Software accessing registers clocked by APB_CLK
- CNT_CLK used for counting, with an average frequency of 16 MHz in two counting cycles
- 40 MHz XTAL_CLK as the clock source of CNT_CLK
- 52-bit alarm values (t) and 26-bit alarm periods (δt)
- Two modes to generate alarms:
  - Target mode: only a one-time alarm is generated based on the alarm value (t)
  - Period mode: periodic alarms are generated based on the alarm period (δt)
- Three comparators generating three independent interrupts based on configured alarm value (t) or alarm period (δt)
- Software configuring the reference count value. For example, the system timer is able to load back the sleep time recorded by RTC timer via software after Light-sleep
- Able to stall or continue running when CPU stalls or enters the on-chip-debugging mode
- Alarm for Event Task Matrix (ETM) event

16.3 System Timer Structure

The timer consists of two counters: UNIT0 and UNIT1. The count values can be monitored by three comparators, COMP0, COMP1, and COMP2. See the timer block diagram in Figure 16.3-1.

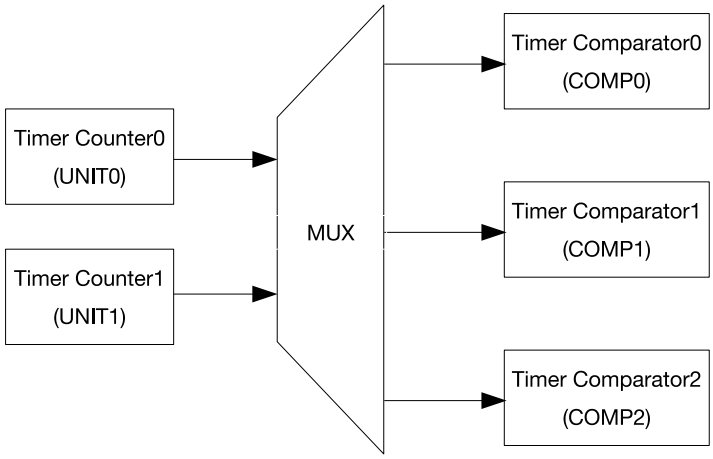


Figure 16.3-1. System Timer Structure

16.4 Clock Source Selection

The counters and comparators use XTAL_CLK or RC_FAST_CLK as the clock sources. The clock source can be selected by configuring field HP_SYS_CLKRST_SYSTIMER_CLK_SRC_SEL in register HP_SYS_CLKRST_PERI_CLK_CTRL21_REG. After XTAL_CLK is scaled by 2.5, a counter clock signal CNT_CLK clock is generated with a frequency of $f_{XTAL_CLK}/2.5$. The average clock frequency of CNT_CLK is 16 MHz, as shown in Figure 16.5-1. The timer counter is incremented by 1/16 μs on each CNT_CLK cycle.

Software operation such as configuring registers is clocked by APB_CLK. For more information about APB_CLK, see Chapter 11 *Reset and Clock*.

The following two bits of system registers are also used to control the system timer:

- Set HP_SYS_CLKRST_SYSTIMER_APB_CLK_EN in register HP_SYS_CLKRST_SOC_CLK_CTRL2_REG to enable APB_CLK signal to the system timer.
- Set HP_SYS_CLKRST_RST_EN_SYSTIMER in register HP_SYS_CLKRST_HP_RST_EN1_REG to reset the system timer.

Note that if the timer is reset, its registers will be restored to their default values. For more information, please refer to Chapter 11 *Reset and Clock*.

16.5 Functional Description

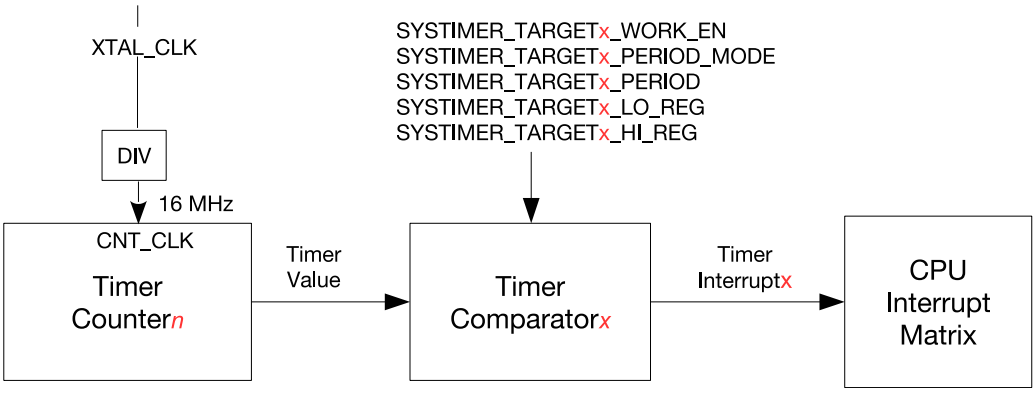


Figure 16.5-1. System Timer Alarm Generation

Figure 16.5-1 shows the procedure to generate alarm in system timer. In this process, one timer counter and one timer comparator are used. An alarm interrupt will be generated accordingly based on the comparison result in the comparator.

16.5.1 Counter

The system timer has two 52-bit timer counters, shown as UNIT*n* (*n* = 0 or 1). Their counting clock source is a 16 MHz clock, i.e., CNT_CLK. Whether UNIT*n* works or not is controlled by three bits in register SYSTIMER_CONF_REG:

- SYSTIMER_TIMER_UNIT*n*_WORK_EN: set this bit to enable the counter UNIT*n* in the system timer.
- SYSTIMER_TIMER_UNIT*n*_CORE0_STALL_EN: if this bit is set, the counter UNIT*n* stops when CPU0 is stalled. The counter continues its counting after the CPU resumes.
- SYSTIMER_TIMER_UNIT*n*_CORE1_STALL_EN: if this bit is set, the counter UNIT*n* stops when CPU1 is stalled. The counter continues its counting after the CPU resumes.

The configuration of the bits to control the counter UNIT*n* is shown below, assuming that CPU is stalled.

Table 16.5-1. UNIT*n* Configuration Bits

SYSTIMER_TIMER_ UNIT*n*_WORK_EN	SYSTIMER_TIMER_ UNIT*n*_CORE0_STALL_EN	SYSTIMER_TIMER_ UNIT*n*_CORE1_STALL_EN	Counter UNIT*n*
0	x *	x *	Not at work
1	1	x	Stop counting, but will continue its counting after the CPU0 resumes
1	x	1	Stop counting, but will continue its counting after the CPU1 resumes
1	0	0	Keep counting

* x: Don't-care.

When the counter UNIT*n* is at work, the count value is incremented on each counting cycle. When the counter UNIT*n* is stopped, the count value stops increasing and keeps unchanged.

The lower 32 and higher 20 bits of the initial count value are loaded from the registers SYSTIMER_TIMER_UNIT*n*_LOAD_LO and SYSTIMER_TIMER_UNIT*n*_LOAD_HI. Writing 1 to the bit SYSTIMER_TIMER_UNIT*n*_LOAD will trigger a reload event, and the current count value will be changed immediately. If UNIT*n* is at work, the counter will continue to count up from the new reloaded value.

Writing 1 to SYSTIMER_TIMER_UNIT*n*_UPDATE will trigger an update event. The lower 32 and higher 20 bits of the current count value will be locked into the registers SYSTIMER_TIMER_UNIT*n*_VALUE_LO and SYSTIMER_TIMER_UNIT*n*_VALUE_HI, and then SYSTIMER_TIMER_UNIT*n*_VALUE_VALID is asserted. Before the next update event, the values of SYSTIMER_TIMER_UNIT*n*_VALUE_LO and SYSTIMER_TIMER_UNIT*n*_VALUE_HI remain unchanged.

16.5.2 Comparator and Alarm

The system timer has three 52-bit comparators, shown as COMP*x* (*x* = 0, 1, or 2). The comparators can generate independent interrupts based on different alarm values (t) or alarm periods (δt).

Configure SYSTIMER_TARGET*x*_PERIOD_MODE to choose from the two alarm modes for each COMP*x*:

- 1: period mode
- 0: target mode

In period mode, the alarm period (δt) is provided by the register SYSTIMER_TARGET*x*_PERIOD. Assuming that current count value is t1, when it reaches (t1 + δt), an alarm interrupt will be generated. When the count value reaches (t1 + 2*δt), another alarm interrupt also will be generated. By such way, periodic alarms are generated.

In target mode, the lower 32 bits and higher 20 bits of the alarm value (t) are provided by SYSTIMER_TIMER_TARGET*x*_LO and SYSTIMER_TIMER_TARGET*x*_HI. Assuming that current count value is t2 (t2 <= t), an alarm interrupt will be generated when the count value reaches the alarm value (t). Unlike in period mode, only one alarm interrupt is generated in target mode.

SYSTIMER_TARGET*x*_TIMER_UNIT_SEL is used to choose the count value from which timer counter to be compared to generate alarms:

- 1: Use the count value from UNIT*1*
- 0: Use the count value from UNIT*0*

Finally, set SYSTIMER_TARGET*x*_WORK_EN and COMP*x* starts to compare the count value:

- In target mode, COMP*x* compares it with the alarm value (t).
- In period mode, COMP*x* compares it with the alarm period (t1 + n*δt).

An alarm is generated when the count value equals to the alarm value (t) in target mode or to the start value + n*alarm period δt (n = 1, 2, 3...) in period mode. But if the alarm value (t) set in registers is less than the current count value, i.e., the target has already passed, when the current count value is larger than the alarm value (t) within a range (0 ~ 2^{51} - 1), an alarm interrupt will also be generated immediately. No matter in target mode or period mode, the low 32 bits and high 20 bits of the real alarm value can always be read from SYSTIMER_TARGET*x*_LO_RO and SYSTIMER_TARGET*x*_HI_RO. The alarm trigger point and the relationship between current count value t_c and the alarm value t_t are shown below.

Table 16.5-2. Trigger Point

Relationship Between t_c and t_t	Trigger Point
t_c- t_t <= 0	t_c = t_t, an alarm is triggered.
0 <= t_c - t_t < 2^{51} - 1 (t_c < 2^{51} and t_t < 2^{51}, or t_c >= 2^{51} and t_t >= 2^{51})	An alarm is triggered immediately.
t_c - t_t >= 2^{51} - 1	t_c overflows after counting to its maximum value 52'hfffffffffffff, and then starts counting up from 0. When its value reaches t_t, an alarm is triggered.

16.5.3 Event Task Matrix

The system timer on ESP32-P4 supports the Event Task Matrix (ETM) function, which allows the system timer's ETM tasks to be triggered by any peripherals' ETM events, or system timer's ETM events to trigger any peripherals' ETM tasks. This section introduces the ETM tasks and events related to the system timer. For more information, please refer to Chapter 14 Event Task Matrix (ETM).

The system timer can generate the following ETM event:

- SYSTIMER_EVT_CNT_CMP*x*: Indicates the alarm events generated by COMP*x*.

When SYSTIMER_ETM_EN is set to 1, the alarm events can trigger the ETM event.

16.5.4 Synchronization Operation

The frequency of the clock used for software configuration is different from that of the clock driving the timer counters and comparators. Therefore, synchronization is required for configuring some registers. The steps are as follows:

1. Software writes specific values to configuration fields. See the first column in Table 16.5-3.
2. Software writes 1 to corresponding bits to start synchronization. See the second column in Table 16.5-3.

Table 16.5-3. Synchronization Operation for Configuration Registers

Configuration Fields	Synchronization Enable Bit
SYSTIMER_TIMER_UNIT*n*_LOAD_LO SYSTIMER_TIMER_UNIT*n*_LOAD_HI	SYSTIMER_TIMER_UNIT*n*_LOAD
SYSTIMER_TARGET*x*_PERIOD SYSTIMER_TIMER_TARGET*x*_HI SYSTIMER_TIMER_TARGET*x*_LO	SYSTIMER_TIMER_COMP*x*_LOAD

The frequency of the clock used for software reading is different from that of the clock driving the timer counters and comparators. Therefore, synchronization is also needed for reading some registers. The steps are as follows:

1. Software writes specific values to the updating register SYSTIMER_TIMER_UNIT*n*_UPDATE.

2. Software reads the corresponding bit SYSTIMER_TIMER_UNIT*n*_VALUE_VALID to be valid to confirm synchronization is done.
3. Software reads the corresponding status registers SYSTIMER_TIMER_UNIT*n*_VALUE_HI and SYSTIMER_TIMER_UNIT*n*_VALUE_LO.

16.6 Interrupts

ESP32-P4's system timer can generate the following interrupt signals that will be sent to the Interrupt Matrix.

- SYSTIMER_TARGET0_INT
- SYSTIMER_TARGET1_INT
- SYSTIMER_TARGET2_INT

There are several internal interrupt sources from the system timer that can generate the above interrupt signals. The interrupt sources from the system timer are listed with their trigger conditions and the resulted interrupt signals in Table 16.6-1.

Table 16.6-1. System Timer's Internal Interrupt Sources

Internal Interrupt Source	Trigger Condition	Interrupt Signal
SYSTIMER_INT0	When COMP0 alarms	SYSTIMER_TARGET0_INT
SYSTIMER_INT1	When COMP1 alarms	SYSTIMER_TARGET1_INT
SYSTIMER_INT2	When COMP2 alarms	SYSTIMER_TARGET2_INT

The above interrupts are level-type alarm interrupts. The interrupt signal is asserted high when the comparator starts to alarm. Until the software clears the interrupt, it remains high. To enable interrupts, set the bit SYSTIMER_TARGET*x*_INT_ENA.

Note:
For definitions of *interrupt*, *interrupt signal*, *interrupt source*, and their correlations, please refer to Chapter 13 *Interrupt Matrix* > Section 13.2 *Interrupt Terminology in ESP32-P4*.

Each interrupt source can be configured by a common set of registers that are described in Section *Interrupt Configuration Registers*. The specific registers can be found in Section 16.8 *Register Summary*.

16.7 Programming Procedure

When configuring COMP*x* and UNIT*n*, please ensure the corresponding COMP and UNIT are at work.

16.7.1 Read Current Count Value

1. Set SYSTIMER_TIMER_UNIT*n*_UPDATE to fill the current count value of COMP*x* into SYSTIMER_TIMER_UNIT*n*_VALUE_HI and SYSTIMER_TIMER_UNIT*n*_VALUE_LO.

2. Poll the reading of SYSTIMER_TIMER_UNIT*n*_VALUE_VALID till it is 1. Then, user can read the count value from SYSTIMER_TIMER_UNIT*n*_VALUE_HI and SYSTIMER_TIMER_UNIT*n*_VALUE_LO.
3. Read the lower 32 bits and higher 20 bits from SYSTIMER_TIMER_UNIT*n*_VALUE_LO and SYSTIMER_TIMER_UNIT*n*_VALUE_HI respectively.

16.7.2 Configure a One-Time Alarm in Target Mode

1. Set SYSTIMER_TARGET*x*_TIMER_UNIT_SEL to select the counter (UNIT*0* or UNIT*1*) used for comparison with COMP*x*.
2. Read the current count value with reference to Section 16.7.1. This value will be used to calculate the alarm value (t) in Step 4.
3. Clear SYSTIMER_TARGET*x*_PERIOD_MODE to enable target mode.
4. Set an alarm value (t), and fill its lower 32 bits into SYSTIMER_TIMER_TARGET*x*_LO, and the higher 20 bits into SYSTIMER_TIMER_TARGET*x*_HI.
5. Set SYSTIMER_TIMER_COMP*x*_LOAD to synchronize the alarm value (t) to COMP*x*, i.e., load the alarm value (t) to the COMP*x*.
6. Set SYSTIMER_TARGET*x*_WORK_EN to enable the selected COMP*x*. COMP*x* starts comparing the count value with the alarm value (t).
7. Set SYSTIMER_TARGET*x*_INT_ENA to enable the timer interrupt. When UNIT*n* reaches the alarm value (t), a SYSTIMER_TARGET*x*_INT interrupt is triggered.

16.7.3 Configure Periodic Alarms in Period Mode

1. Set SYSTIMER_TARGET*x*_TIMER_UNIT_SEL to select the counter (UNIT*0* or UNIT*1*) used for comparison with COMP*x*.
2. Set an alarm period (δt), and fill it into SYSTIMER_TARGET*x*_PERIOD.
3. Set SYSTIMER_TIMER_COMP*x*_LOAD to synchronize the alarm period (δt) to COMP*x*, i.e., load the alarm period (δt) to COMP*x*.
4. Clear and then set SYSTIMER_TARGET*x*_PERIOD_MODE to configure COMP*x* into period mode.
5. Set SYSTIMER_TARGET*x*_WORK_EN to enable the selected COMP*x*. COMP*x* starts comparing the count value with the sum of (start value + n*δt) (n = 1, 2, 3...).
6. Set SYSTIMER_TARGET*x*_INT_ENA to enable the timer interrupt. A SYSTIMER_TARGET*x*_INT interrupt is triggered when UNIT*n* reaches start value + n*δt (n = 1, 2, 3...) set in Step 2.

16.7.4 Update After Light-sleep

1. Configure RTC timer before the chip goes to Light-sleep mode, to record the exact sleep time. For more information, see Chapter 15 *Low-Power Management*.
2. Read the sleep time from the RTC timer when the chip wakes up from Light-sleep mode.
3. Read the current count value of system timer, see Section 16.7.1.

4. Convert the time value recorded by RTC timer from the clock cycles based on RTC_SLOW_CLK to that based on 16 MHz CNT_CLK. For example, if the frequency of RTC_SLOW_CLK is 32 kHz, the recorded RTC timer value should be converted by multiplying by 500.
5. Add the converted RTC value to the current count value of system timer:
   - Fill the new value into SYSTIMER_TIMER_UNIT*n*_LOAD_LO (low 32 bits) and SYSTIMER_TIMER_UNIT*n*_LOAD_HI (high 20 bits).
   - Set SYSTIMER_TIMER_UNIT*n*_LOAD to load the new timer value into the system timer. By such way, the system timer is updated.

16.8 Register Summary

The addresses in this section are relative to system timer base address provided in Table 9.3-2 in Chapter 9 *System and Memory*.

The abbreviations given in Column **Access** are explained in Section *Access Types for Registers*.

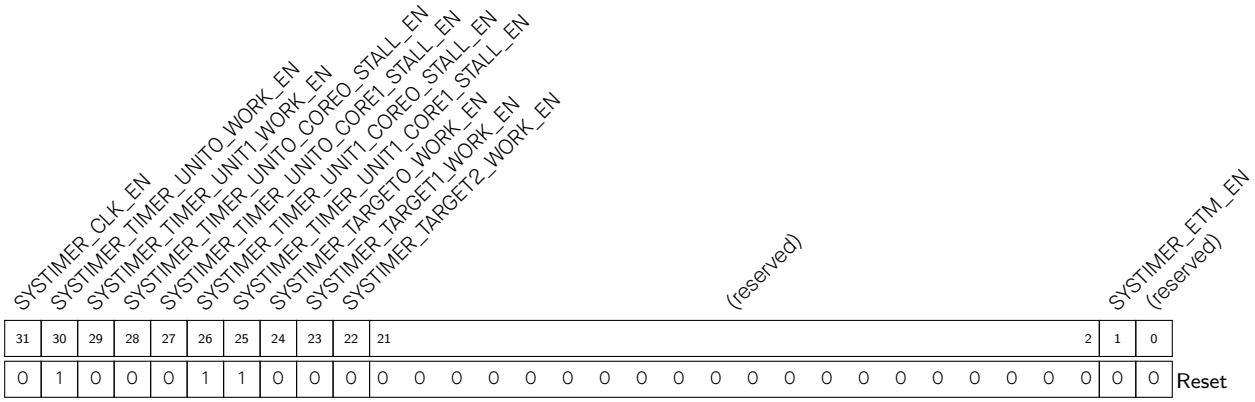
Name	Description	Address	Access
Clock Control Register			
SYSTIMER_CONF_REG	Configure system timer clock	0x0000	R/W
UNIT*0* Control and Configuration Registers			
SYSTIMER_UNIT0_OP_REG	Read UNIT0 value to registers	0x0004	varies
SYSTIMER_UNIT0_LOAD_HI_REG	High 20 bits to be loaded to UNIT0	0x000C	R/W
SYSTIMER_UNIT0_LOAD_LO_REG	Low 32 bits to be loaded to UNIT0	0x0010	R/W
SYSTIMER_UNIT0_VALUE_HI_REG	UNIT0 value, high 20 bits	0x0040	RO
SYSTIMER_UNIT0_VALUE_LO_REG	UNIT0 value, low 32 bits	0x0044	RO
SYSTIMER_UNIT0_LOAD_REG	UNIT0 synchronization register	0x005C	WT
UNIT*1* Control and Configuration Registers			
SYSTIMER_UNIT1_OP_REG	Read UNIT1 value to registers	0x0008	varies
SYSTIMER_UNIT1_LOAD_HI_REG	High 20 bits to be loaded to UNIT1	0x0014	R/W
SYSTIMER_UNIT1_LOAD_LO_REG	Low 32 bits to be loaded to UNIT1	0x0018	R/W
SYSTIMER_UNIT1_VALUE_HI_REG	UNIT1 value, high 20 bits	0x0048	RO
SYSTIMER_UNIT1_VALUE_LO_REG	UNIT1 value, low 32 bits	0x004C	RO
SYSTIMER_UNIT1_LOAD_REG	UNIT1 synchronization register	0x0060	WT
Comparator*0* Control and Configuration Registers			
SYSTIMER_TARGET0_HI_REG	Alarm value to be loaded to COMP0, high 20 bits	0x001C	R/W
SYSTIMER_TARGET0_LO_REG	Alarm value to be loaded to COMP0, low 32 bits	0x0020	R/W
SYSTIMER_TARGET0_CONF_REG	Configure COMP0 alarm mode	0x0034	R/W
SYSTIMER_COMP0_LOAD_REG	COMP0 synchronization register	0x0050	WT
Comparator*1* Control and Configuration Registers			
SYSTIMER_TARGET1_HI_REG	Alarm value to be loaded to COMP1, high 20 bits	0x0024	R/W
SYSTIMER_TARGET1_LO_REG	Alarm value to be loaded to COMP1, low 32 bits	0x0028	R/W
SYSTIMER_TARGET1_CONF_REG	Configure COMP1 alarm mode	0x0038	R/W
SYSTIMER_COMP1_LOAD_REG	COMP1 synchronization register	0x0054	WT
Comparator*2* Control and Configuration Registers			
SYSTIMER_TARGET2_HI_REG	Alarm value to be loaded to COMP2, high 20 bits	0x002C	R/W
SYSTIMER_TARGET2_LO_REG	Alarm value to be loaded to COMP2, low 32 bits	0x0030	R/W
SYSTIMER_TARGET2_CONF_REG	Configure COMP2 alarm mode	0x003C	R/W
SYSTIMER_COMP2_LOAD_REG	COMP2 synchronization register	0x0058	WT
Interrupt Registers			
SYSTIMER_INT_ENA_REG	Interrupt enable register of system timer	0x0064	R/W
SYSTIMER_INT_RAW_REG	Interrupt raw register of system timer	0x0068	R/WTC/SS
SYSTIMER_INT_CLR_REG	Interrupt clear register of system timer	0x006C	WT

Name	Description	Address	Access
SYSTIMER_INT_ST_REG	Interrupt status register of system timer	0x0070	RO
COMP*0* Status Registers			
SYSTIMER_REAL_TARGET0_LO_REG	Actual target value of COMP0, low 32 bits	0x0074	RO
SYSTIMER_REAL_TARGET0_HI_REG	Actual target value of COMP0, high 20 bits	0x0078	RO
COMP*1* Status Registers			
SYSTIMER_REAL_TARGET1_LO_REG	Actual target value of COMP1, low 32 bits	0x007C	RO
SYSTIMER_REAL_TARGET1_HI_REG	Actual target value of COMP1, high 20 bits	0x0080	RO
COMP*2* Status Registers			
SYSTIMER_REAL_TARGET2_LO_REG	Actual target value of COMP2, low 32 bits	0x0084	RO
SYSTIMER_REAL_TARGET2_HI_REG	Actual target value of COMP2, high 20 bits	0x0088	RO
Version Register			
SYSTIMER_DATE_REG	Version control register	0x00FC	R/W

16.9 Registers

The addresses in this section are relative to system timer base address provided in Table 9.3-2 in Chapter 9 *System and Memory*.

Register 16.1. SYSTIMER_CONF_REG (0x0000)


SYSTIMER_CLK_EN
SYSTIMER_TIMER_UNIT0_WORK_EN
SYSTIMER_TIMER_UNIT1_WORK_EN
SYSTIMER_TIMER_UNIT0_CORE0_STALL_EN
SYSTIMER_TIMER_UNIT0_CORE1_STALL_EN
SYSTIMER_TIMER_UNIT1_CORE0_STALL_EN
SYSTIMER_TIMER_UNIT1_CORE1_STALL_EN
SYSTIMER_TARGET0_WORK_EN
SYSTIMER_TARGET1_WORK_EN
SYSTIMER_TARGET2_WORK_EN
(reserved)
SYSTIMER_ETM_EN
(reserved)
31 30 29 28 27 26 25 24 23 22 21 2 1 0
0 1 0 0 0 1 1 0 0 0

Register 16.1. SYSTIMER_CONF_REG (0x0000)

Continued from the previous page...

SYSTIMER_TIMER_UNIT1_WORK_EN Configures whether to enable UNIT1.

0: Disable

1: Enable

(R/W)

SYSTIMER_TIMER_UNIT0_WORK_EN Configures whether to enable UNIT0.

0: Disable

1: Enable

(R/W)

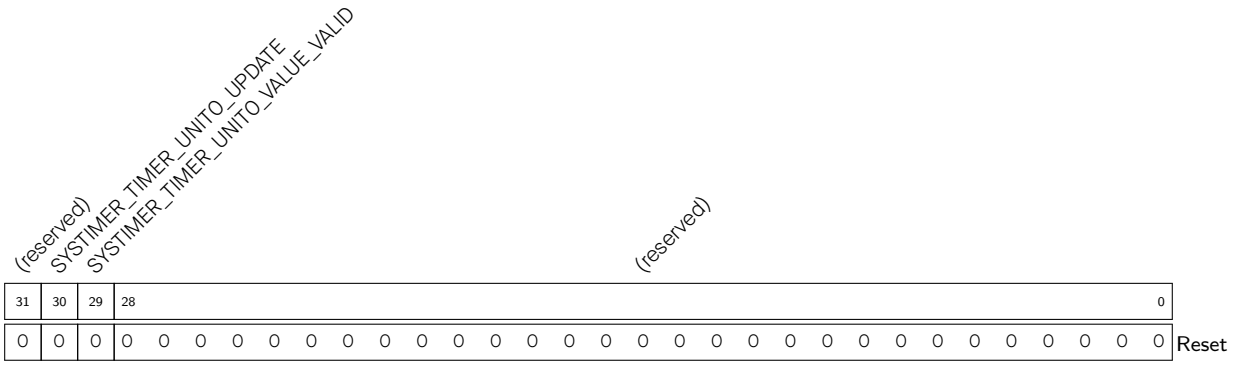
SYSTIMER_CLK_EN Configures register clock gating.

0: Only enable needed clock for register read or write operations.

1: Register clock is always enabled for read and write operations.

(R/W)

Register 16.2. SYSTIMER_UNIT0_OP_REG (0x0004)



SYSTIMER_TIMER_UNIT0_VALUE_VALID Represents whether UNIT0 value is synchronized and valid.

0: UNIT0 value is neither synchronized nor valid

1: UNIT0 value is synchronized and valid

(R/SS/WTC)

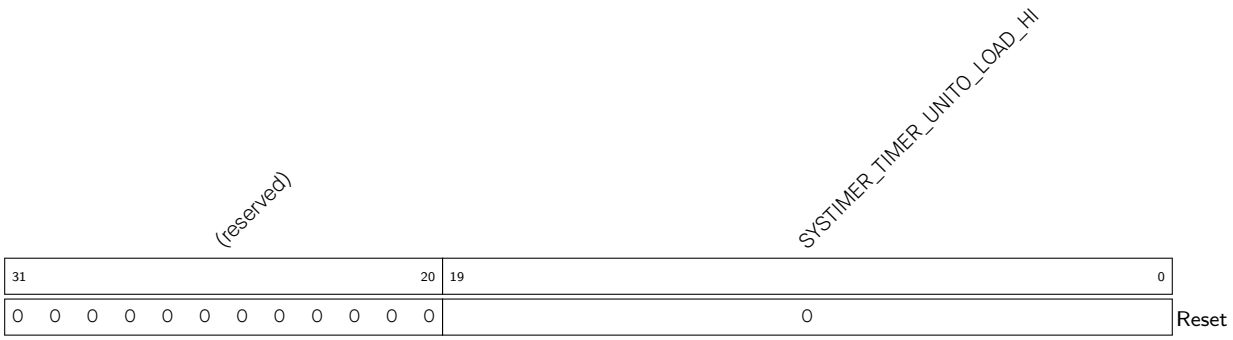
SYSTIMER_TIMER_UNIT0_UPDATE Configures whether to update timer UNIT0, i.e., reads the UNIT0 count value to SYSTIMER_TIMER_UNIT0_VALUE_HI and SYSTIMER_TIMER_UNIT0_VALUE_LO.

0: No effect

1: Update timer UNIT0

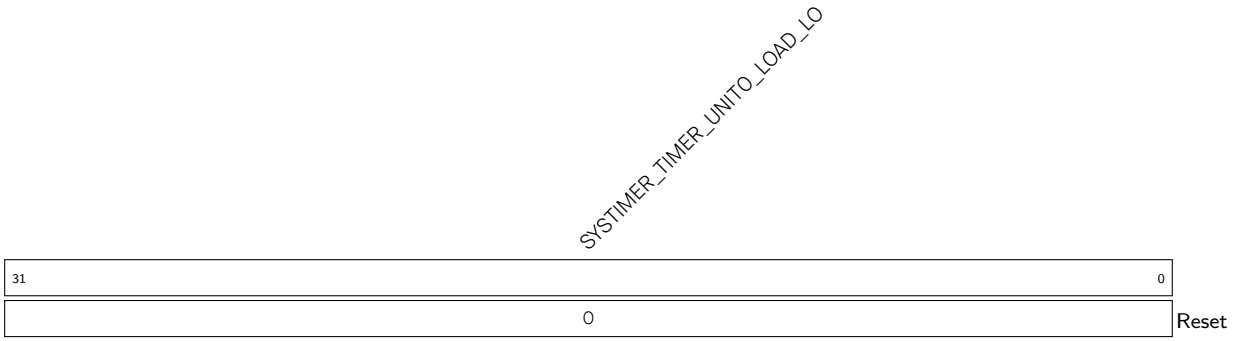
(WT)

Register 16.3. SYSTIMER_UNIT0_LOAD_HI_REG (0x000C)



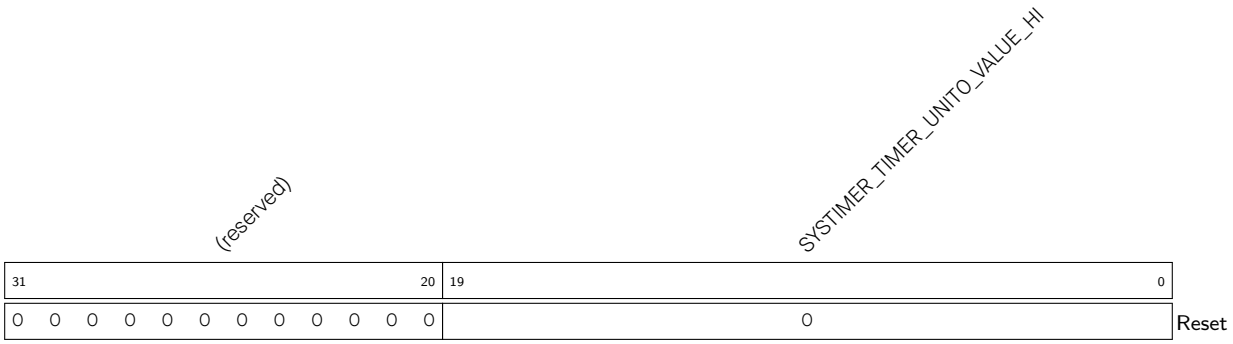
SYSTIMER_TIMER_UNIT0_LOAD_HI Configures the value to be loaded to UNIT0, high 20 bits. (R/W)

Register 16.4. SYSTIMER_UNIT0_LOAD_LO_REG (0x0010)



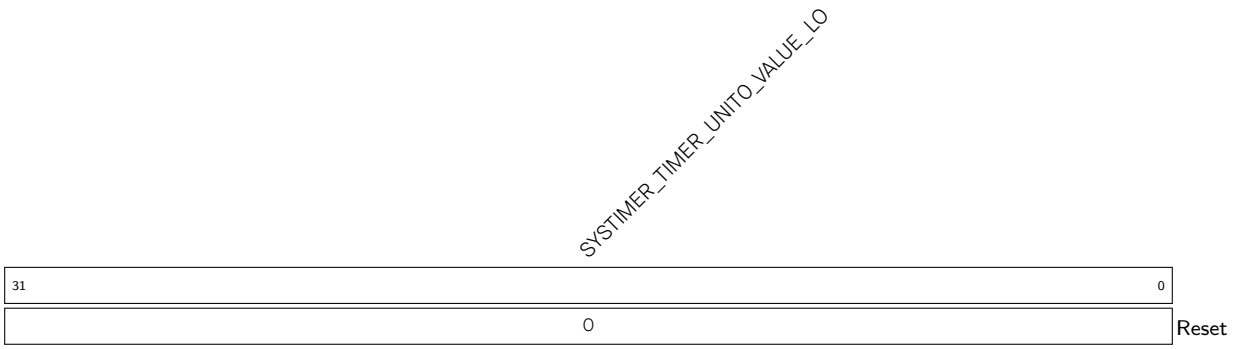
SYSTIMER_TIMER_UNIT0_LOAD_LO Configures the value to be loaded to UNIT0, low 32 bits. (R/W)

Register 16.5. SYSTIMER_UNIT0_VALUE_HI_REG (0x0040)



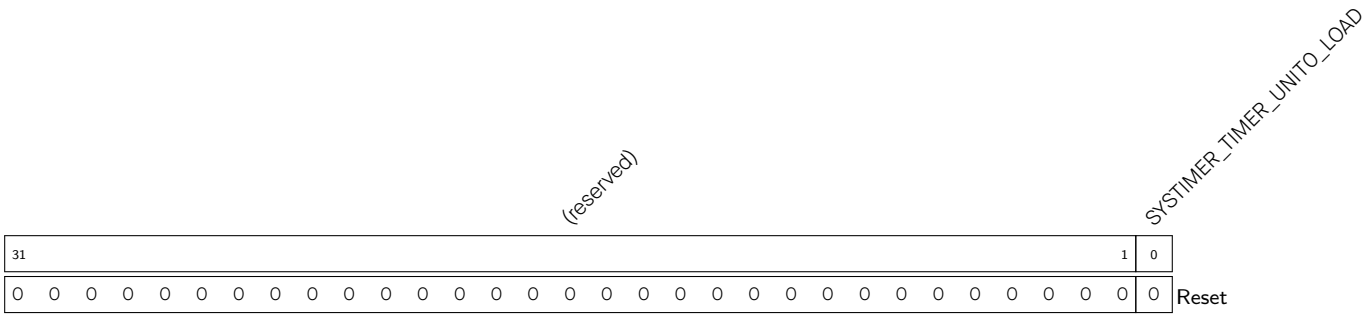
SYSTIMER_TIMER_UNIT0_VALUE_HI Represents UNIT0 read value, high 20 bits. (RO)

Register 16.6. SYSTIMER_UNIT0_VALUE_LO_REG (0x0044)



SYSTIMER_TIMER_UNIT0_VALUE_LO Represents UNIT0 read value, low 32 bits. (RO)

Register 16.7. SYSTIMER_UNIT0_LOAD_REG (0x005C)



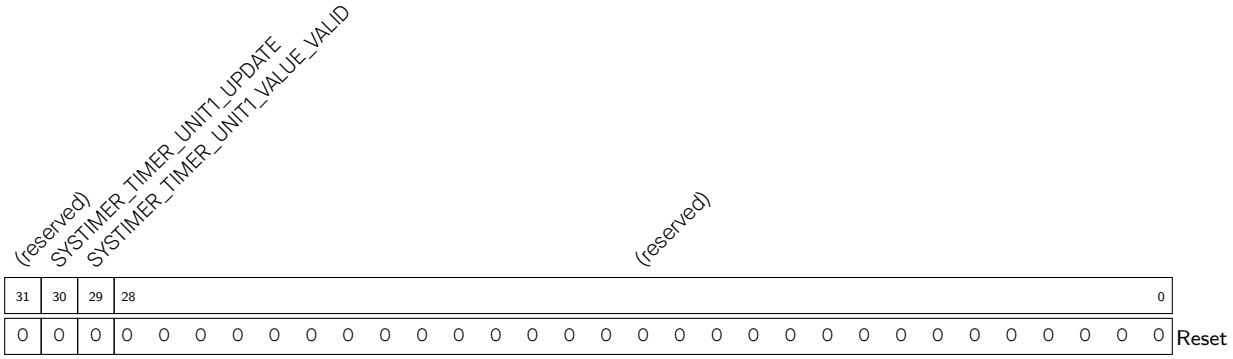
SYSTIMER_TIMER_UNIT0_LOAD Configures whether to reload the value of UNIT0, i.e., reloads the values of SYSTIMER_TIMER_UNIT0_VALUE_HI and SYSTIMER_TIMER_UNIT0_VALUE_LO to UNIT0.

0: No effect

1: Reload the value of UNIT0

(WT)

Register 16.8. SYSTIMER_UNIT1_OP_REG (0x0008)



SYSTIMER_TIMER_UNIT1_VALUE_VALID Represents UNIT1 value is synchronized and valid. (R/SS/WTC)

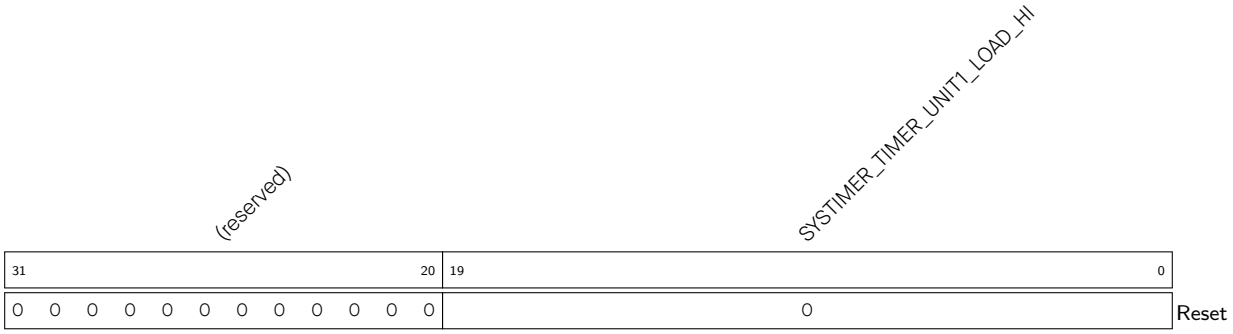
SYSTIMER_TIMER_UNIT1_UPDATE Configures whether to update timer UNIT1, i.e., reads the UNIT1 count value to SYSTIMER_TIMER_UNIT1_VALUE_HI and SYSTIMER_TIMER_UNIT1_VALUE_LO.

0: No effect

1: Update timer UNIT1

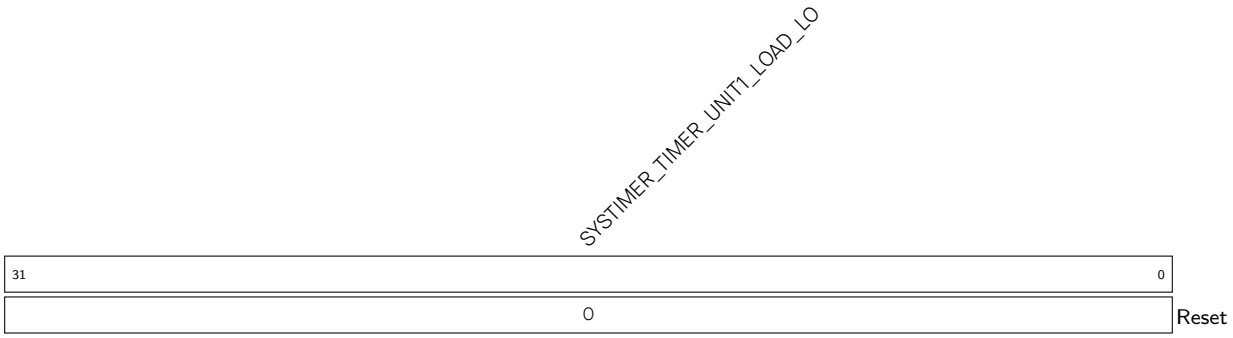
(WT)

Register 16.9. SYSTIMER_UNIT1_LOAD_HI_REG (0x0014)



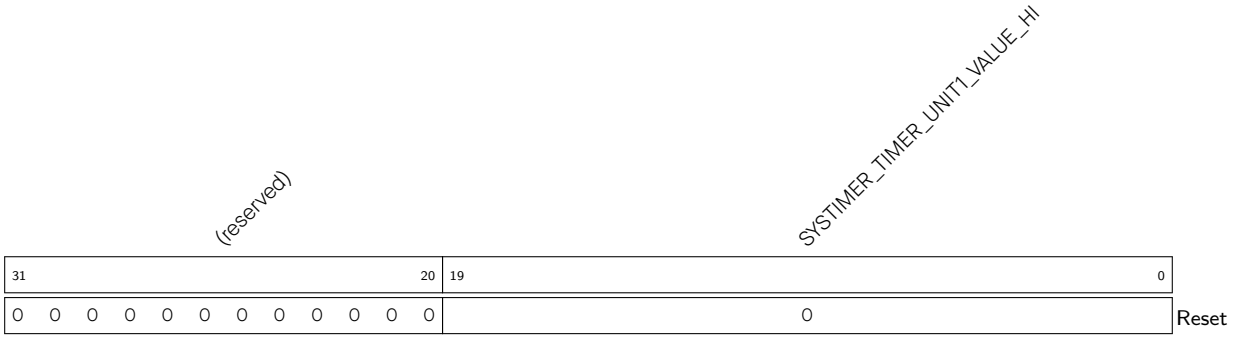
SYSTIMER_TIMER_UNIT1_LOAD_HI Configures the value to be loaded to UNIT1, high 20 bits. (R/W)

Register 16.10. SYSTIMER_UNIT1_LOAD_LO_REG (0x0018)



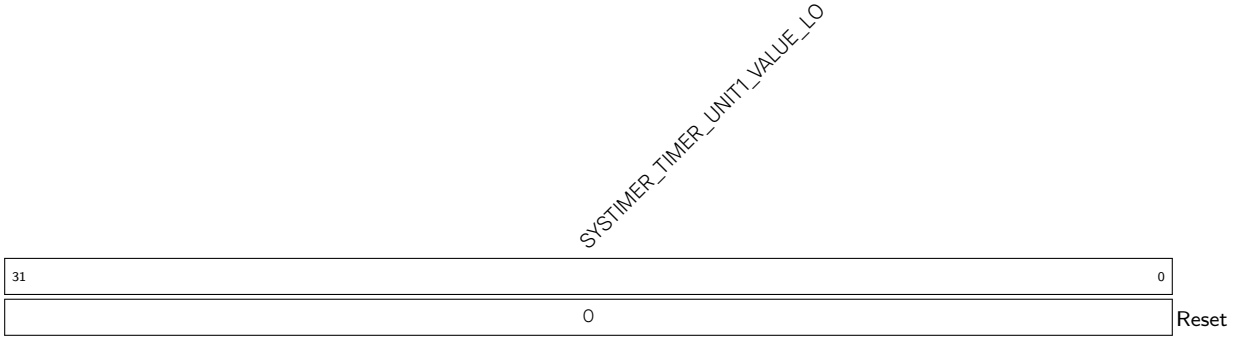
SYSTIMER_TIMER_UNIT1_LOAD_LO Configures the value to be loaded to UNIT1, low 32 bits. (R/W)

Register 16.11. SYSTIMER_UNIT1_VALUE_HI_REG (0x0048)



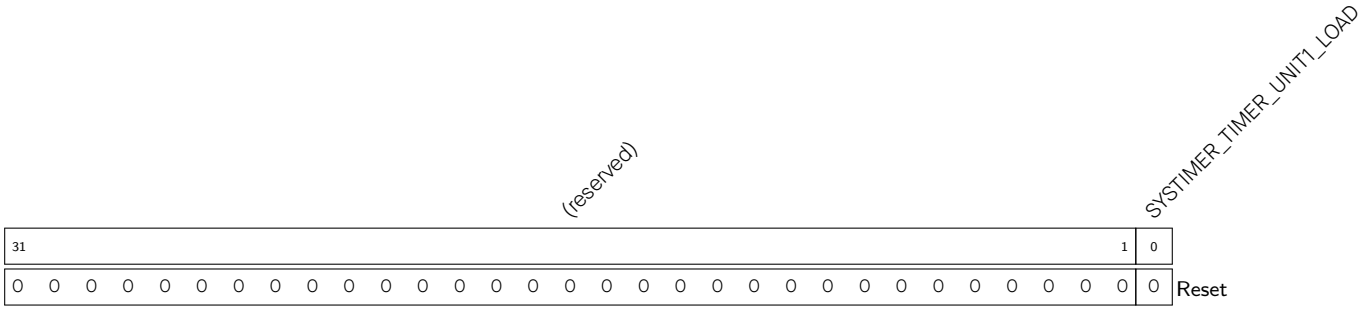
SYSTIMER_TIMER_UNIT1_VALUE_HI Represents UNIT1 read value, high 20 bits. (RO)

Register 16.12. SYSTIMER_UNIT1_VALUE_LO_REG (0x004C)



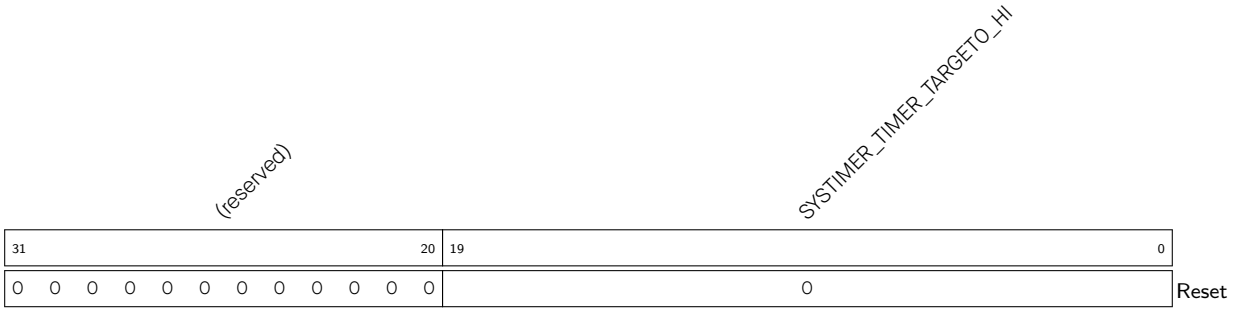
SYSTIMER_TIMER_UNIT1_VALUE_LO Represents UNIT1 read value, low 32 bits. (RO)

Register 16.13. SYSTIMER_UNIT1_LOAD_REG (0x0060)



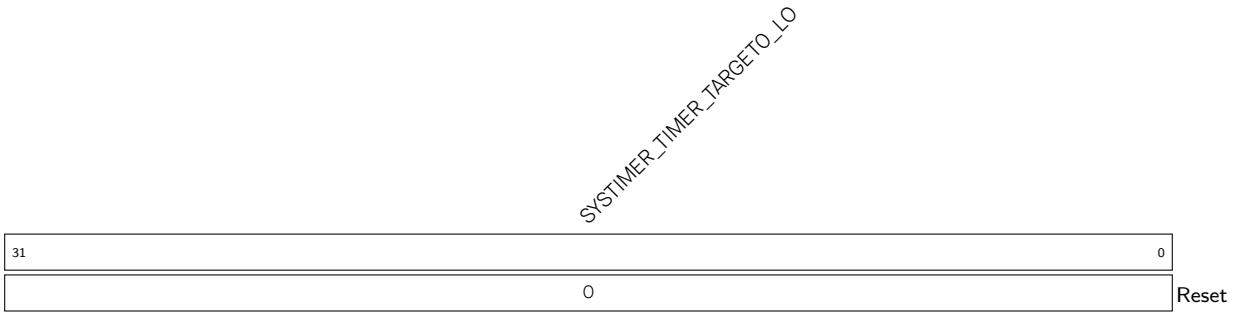
SYSTIMER_TIMER_UNIT1_LOAD Configures whether to reload the value of UNIT1, i.e., reload the values of SYSTIMER_TIMER_UNIT1_VALUE_HI and SYSTIMER_TIMER_UNIT1_VALUE_LO to UNIT1.
0: No effect
1: Reload the value of UNIT1
(WT)

Register 16.14. SYSTIMER_TARGET0_HI_REG (0x001C)



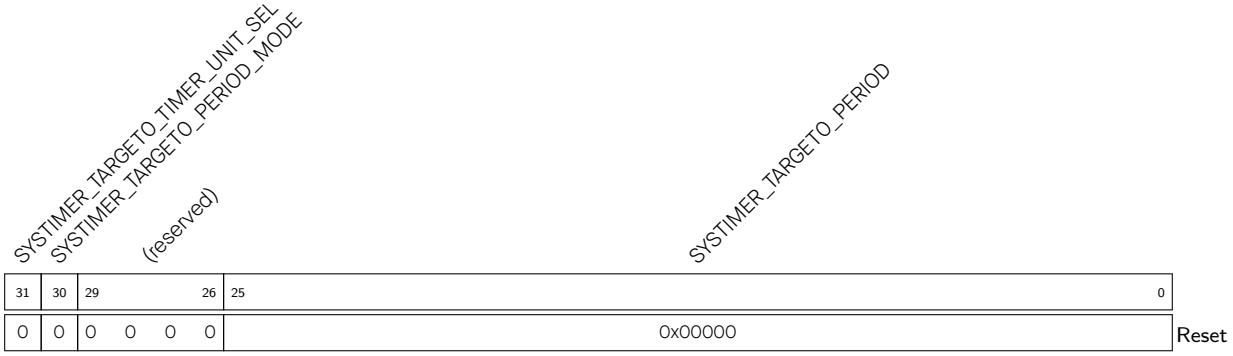
SYSTIMER_TIMER_TARGET0_HI Configures the alarm value to be loaded to COMP0, high 20 bits. (R/W)

Register 16.15. SYSTIMER_TARGET0_LO_REG (0x0020)



SYSTIMER_TIMER_TARGET0_LO Configures the alarm value to be loaded to COMP0, low 32 bits. (R/W)

Register 16.16. SYSTIMER_TARGET0_CONF_REG (0x0034)



SYSTIMER_TARGET0_PERIOD Configures COMP0 alarm period. (R/W)

SYSTIMER_TARGET0_PERIOD_MODE Selects the two alarm modes for COMP0.

0: Target mode

1: Period mode

(R/W)

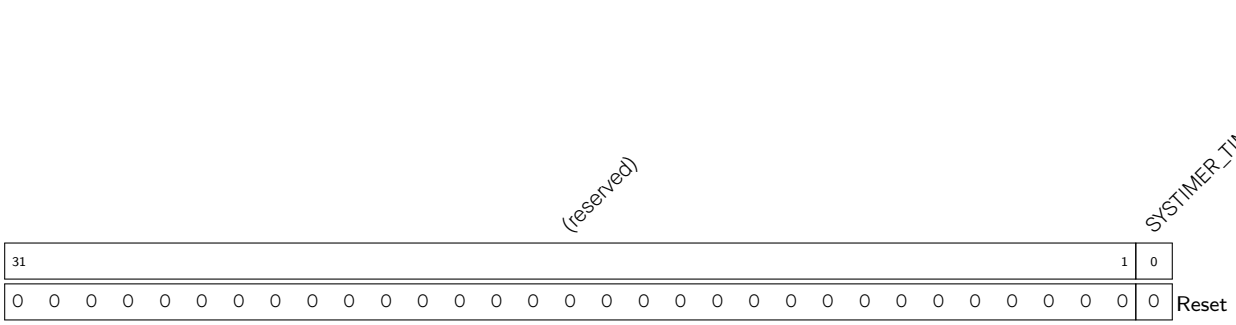
SYSTIMER_TARGET0_TIMER_UNIT_SEL Chooses the count value for comparison with COMP0.

0: Use the count value from UNIT*0*

1: Use the count value from UNIT*1*

(R/W)

Register 16.17. SYSTIMER_COMP0_LOAD_REG (0x0050)



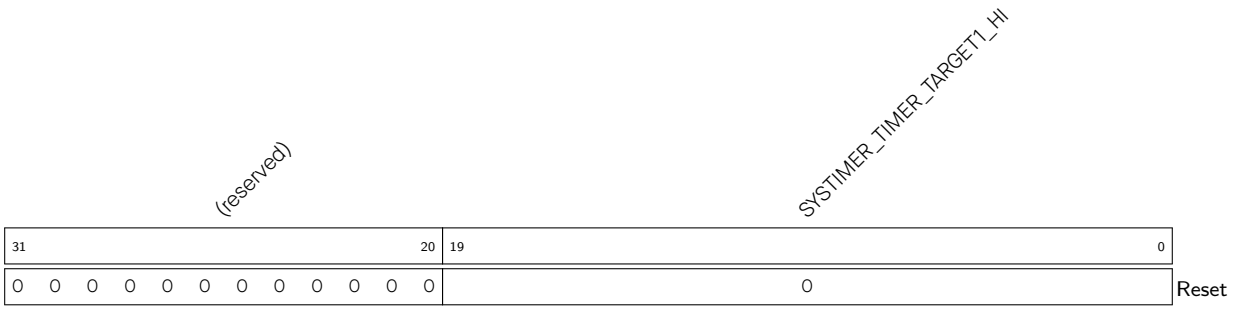
SYSTIMER_TIMER_COMP0_LOAD Configures whether to enable COMP0 synchronization, i.e., reload the alarm value/period to COMP0.

0: No effect

1: Enable COMP0 synchronization

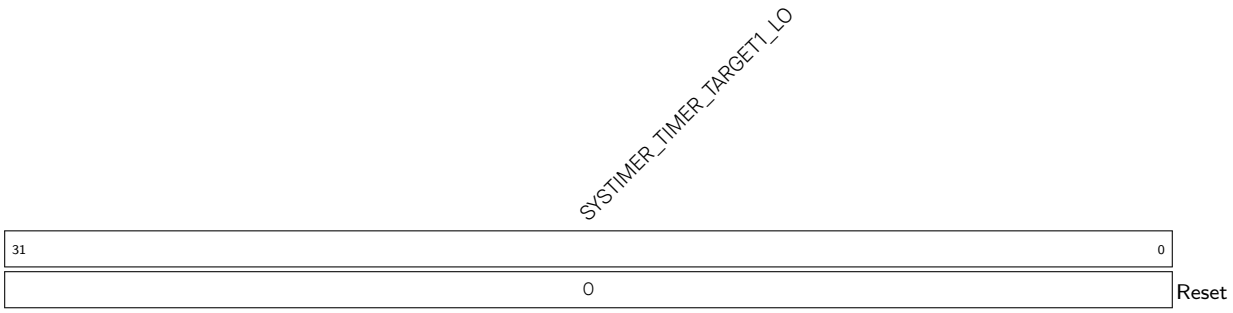
(WT)

Register 16.18. SYSTIMER_TARGET1_HI_REG (0x0024)



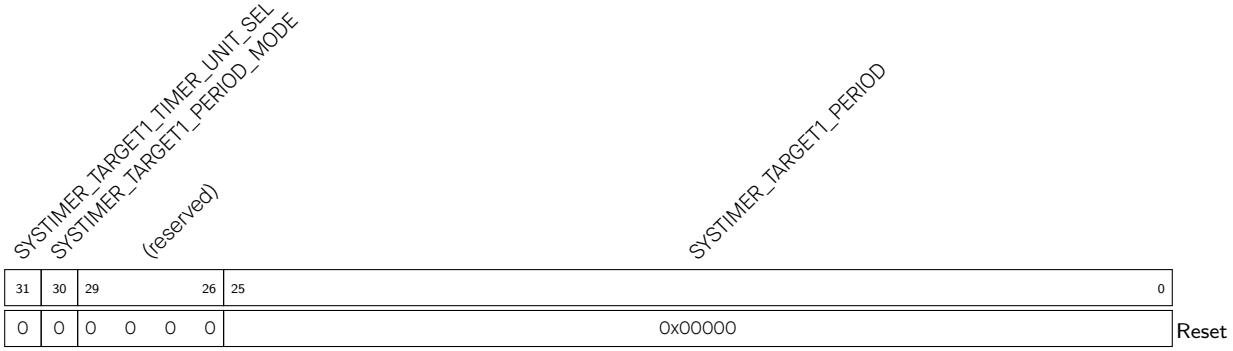
SYSTIMER_TIMER_TARGET1_HI Configures the alarm value to be loaded to COMP1, high 20 bits. (R/W)

Register 16.19. SYSTIMER_TARGET1_LO_REG (0x0028)



SYSTIMER_TIMER_TARGET1_LO Configures the alarm value to be loaded to COMP1, low 32 bits. (R/W)

Register 16.20. SYSTIMER_TARGET1_CONF_REG (0x0038)

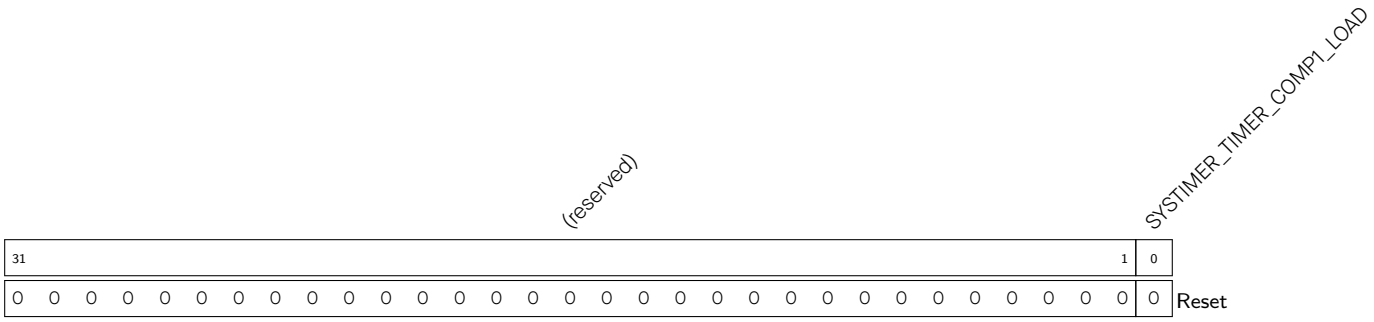


SYSTIMER_TARGET1_PERIOD Configures COMP1 alarm period. (R/W)

SYSTIMER_TARGET1_PERIOD_MODE Selects the two alarm modes for COMP1. See details in SYSTIMER_TARGET0_PERIOD_MODE. (R/W)

SYSTIMER_TARGET1_TIMER_UNIT_SEL Chooses the count value for comparison with COMP1. See details in SYSTIMER_TARGET0_TIMER_UNIT_SEL. (R/W)

Register 16.21. SYSTIMER_COMP1_LOAD_REG (0x0054)



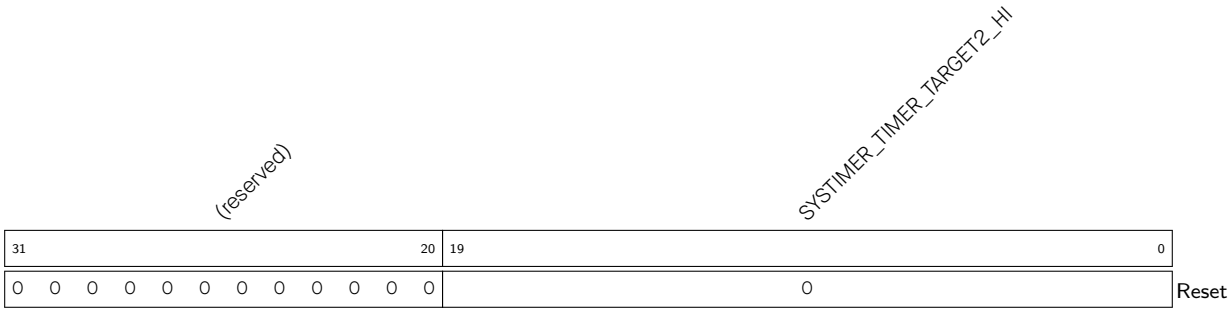
SYSTIMER_TIMER_COMP1_LOAD Configures whether to enable COMP1 synchronization, i.e., reload the alarm value/period to COMP1.

0: No effect

1: Enable COMP1 synchronization

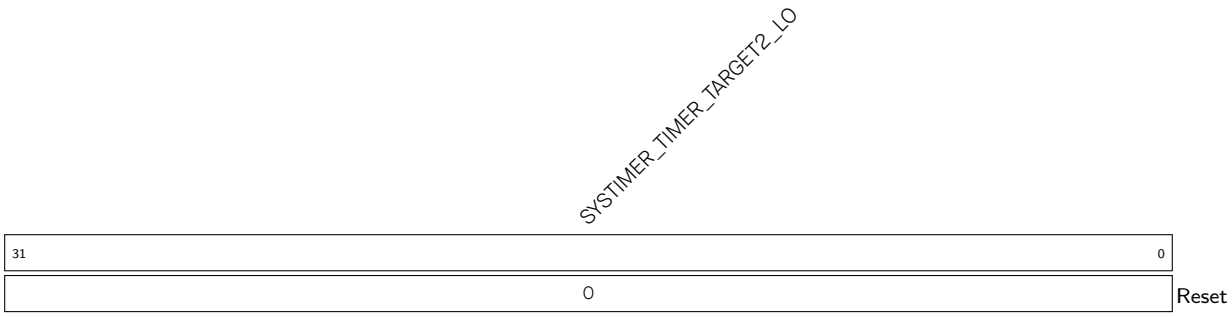
(WT)

Register 16.22. SYSTIMER_TARGET2_HI_REG (0x002C)



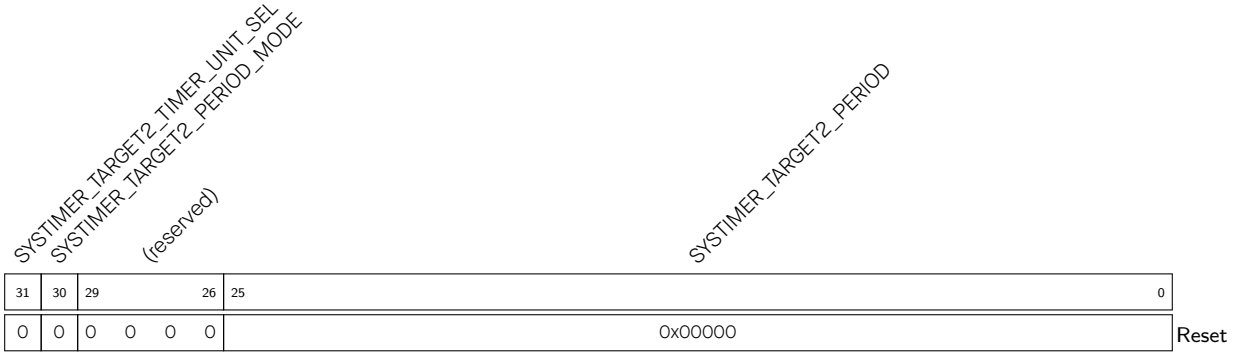
SYSTIMER_TIMER_TARGET2_HI Configures the alarm value to be loaded to COMP2, high 20 bits. (R/W)

Register 16.23. SYSTIMER_TARGET2_LO_REG (0x0030)



SYSTIMER_TIMER_TARGET2_LO Configures the alarm value to be loaded to COMP2, low 32 bits. (R/W)

Register 16.24. SYSTIMER_TARGET2_CONF_REG (0x003C)

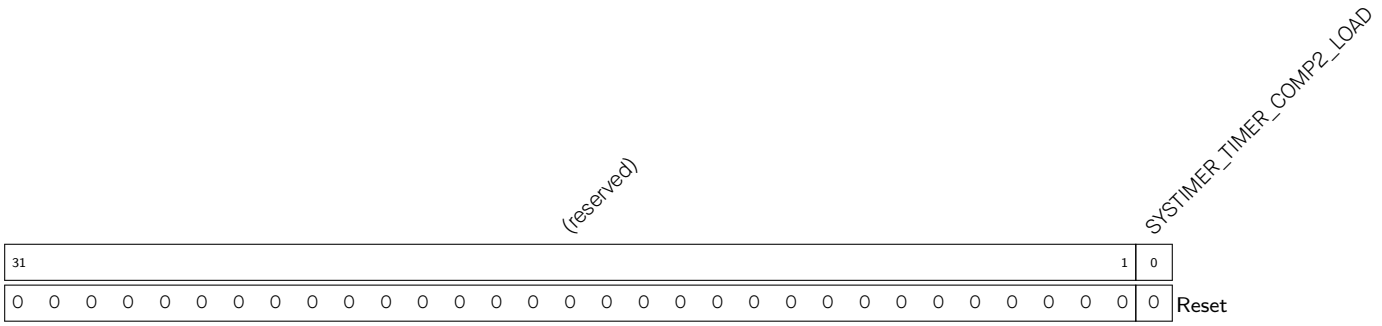


SYSTIMER_TARGET2_PERIOD Configures COMP2 alarm period. (R/W)

SYSTIMER_TARGET2_PERIOD_MODE Configures Configures the two alarm modes for COMP2. See details in SYSTIMER_TARGET0_PERIOD_MODE. (R/W)

SYSTIMER_TARGET2_TIMER_UNIT_SEL Chooses the count value for comparison with COMP2. See details in SYSTIMER_TARGET0_TIMER_UNIT_SEL. (R/W)

Register 16.25. SYSTIMER_COMP2_LOAD_REG (0x0058)



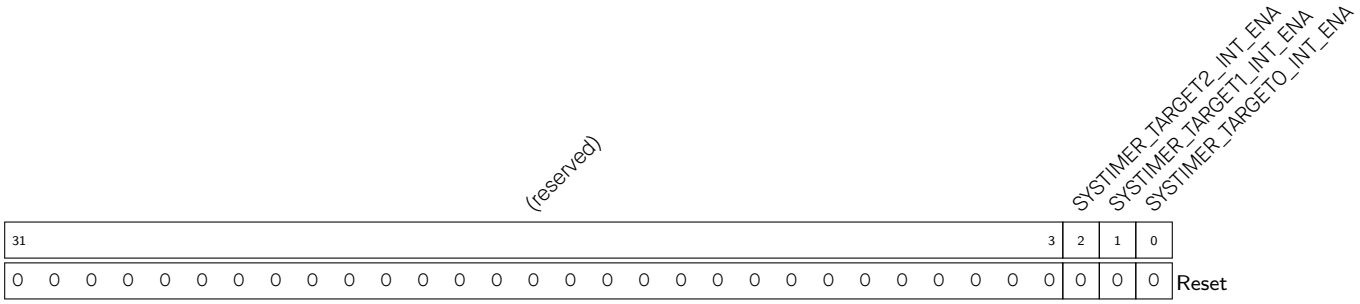
SYSTIMER_TIMER_COMP2_LOAD Configures whether to enable COMP2 synchronization, i.e., reload the alarm value/period to COMP2.

0: No effect

1: Enable COMP2 synchronization

(WT)

Register 16.26. SYSTIMER_INT_ENA_REG (0x0064)

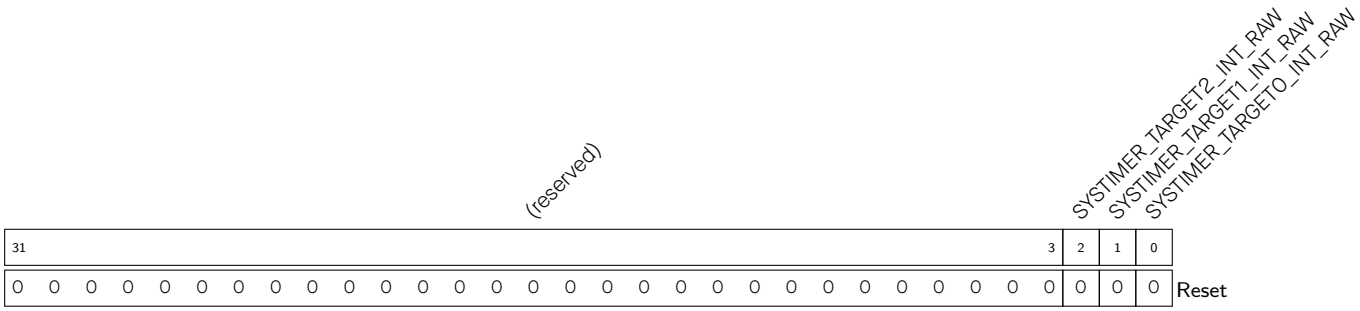


SYSTIMER_TARGET0_INT_ENA Write 1 to enable SYSTIMER_TARGET0_INT. (R/W)

SYSTIMER_TARGET1_INT_ENA Write 1 to enable SYSTIMER_TARGET1_INT. (R/W)

SYSTIMER_TARGET2_INT_ENA Write 1 to enable SYSTIMER_TARGET2_INT. (R/W)

Register 16.27. SYSTIMER_INT_RAW_REG (0x0068)

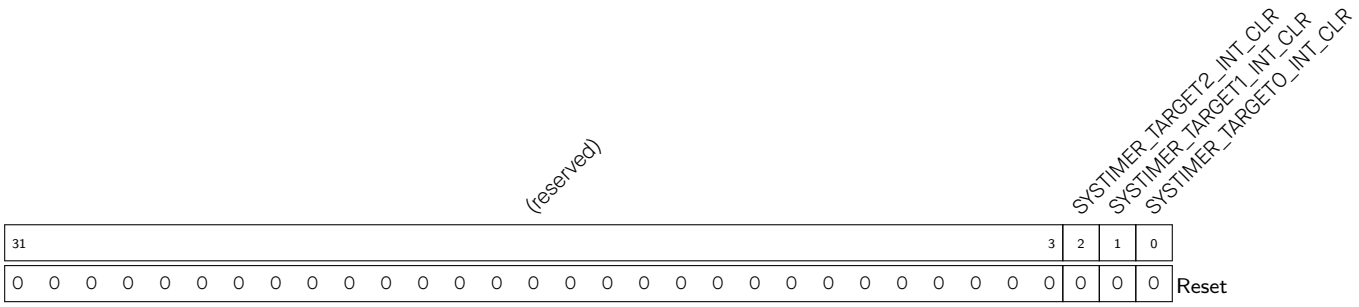


SYSTIMER_TARGET0_INT_RAW The raw interrupt status of SYSTIMER_TARGET0_INT. (R/WTC/SS)

SYSTIMER_TARGET1_INT_RAW The raw interrupt status of SYSTIMER_TARGET1_INT. (R/WTC/SS)

SYSTIMER_TARGET2_INT_RAW The raw interrupt status of SYSTIMER_TARGET2_INT. (R/WTC/SS)

Register 16.28. SYSTIMER_INT_CLR_REG (0x006C)

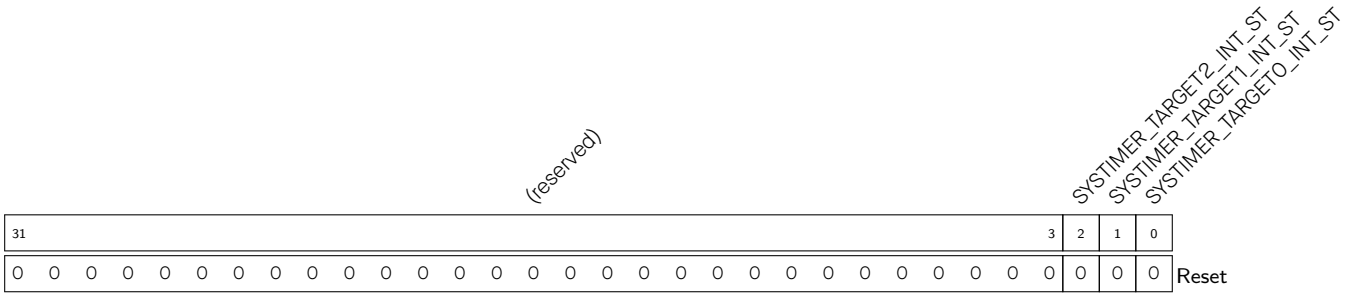


SYSTIMER_TARGET0_INT_CLR Write 1 to clear SYSTIMER_TARGET0_INT. (WT)

SYSTIMER_TARGET1_INT_CLR Write 1 to clear SYSTIMER_TARGET1_INT. (WT)

SYSTIMER_TARGET2_INT_CLR Write 1 to clear SYSTIMER_TARGET2_INT. (WT)

Register 16.29. SYSTIMER_INT_ST_REG (0x0070)

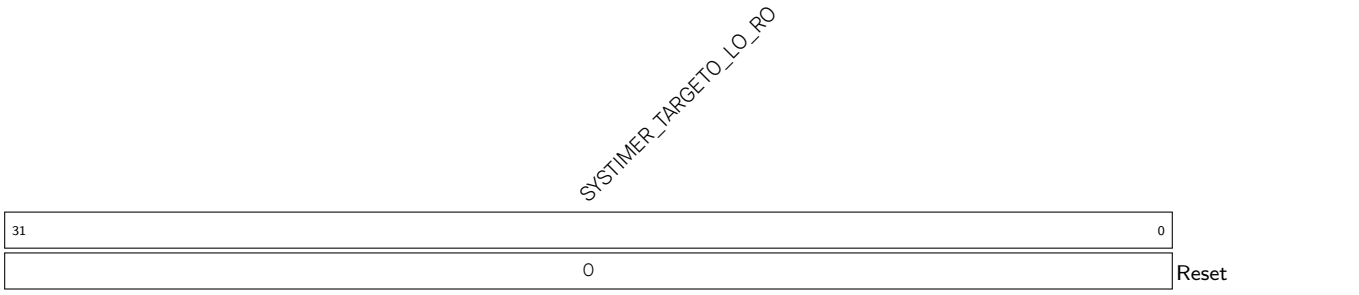


SYSTIMER_TARGET0_INT_ST The masked interrupt status of SYSTIMER_TARGET0_INT. (RO)

SYSTIMER_TARGET1_INT_ST The masked interrupt status of SYSTIMER_TARGET1_INT. (RO)

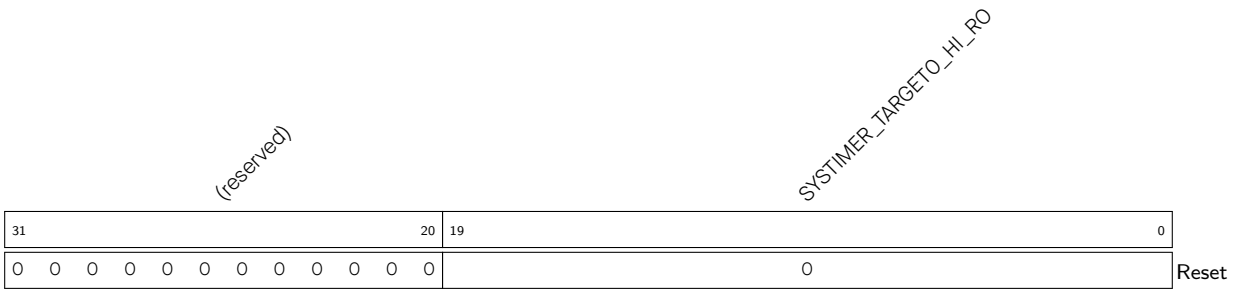
SYSTIMER_TARGET2_INT_ST The masked interrupt status of SYSTIMER_TARGET2_INT. (RO)

Register 16.30. SYSTIMER_REAL_TARGET0_LO_REG (0x0074)



SYSTIMER_TARGET0_LO_RO Represents the actual target value of COMP0, low 32 bits. (RO)

Register 16.31. SYSTIMER_REAL_TARGET0_HI_REG (0x0078)



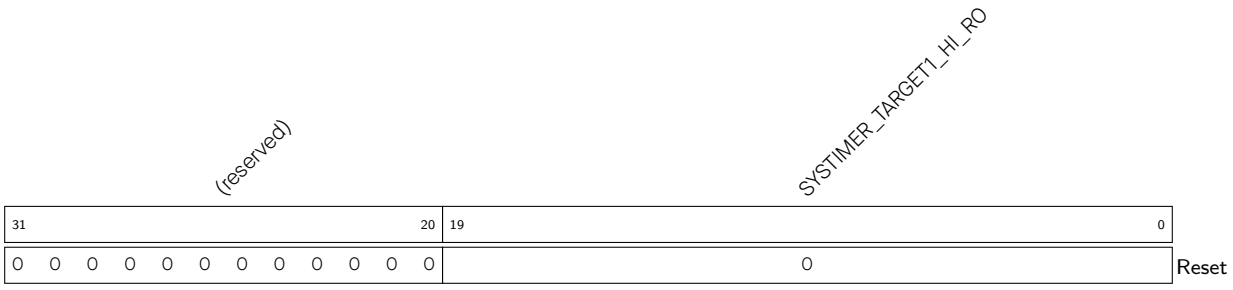
SYSTIMER_TARGET0_HI_RO Represents the actual target value of COMP0, high 20 bits. (RO)

Register 16.32. SYSTIMER_REAL_TARGET1_LO_REG (0x007C)



SYSTIMER_TARGET1_LO_RO Represents the actual target value of COMP1, low 32 bits. (RO)

Register 16.33. SYSTIMER_REAL_TARGET1_HI_REG (0x0080)



SYSTIMER_TARGET1_HI_RO Represents the actual target value of COMP1, high 20 bits. (RO)

Register 16.34. SYSTIMER_REAL_TARGET2_LO_REG (0x0084)



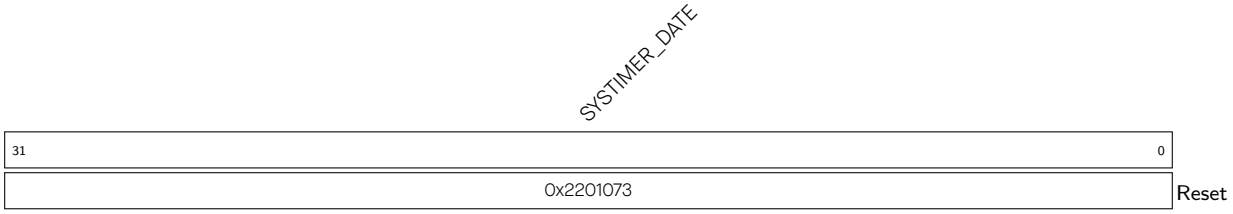
SYSTIMER_TARGET2_LO_RO Represents the actual target value of COMP2, low 32 bits. (RO)

Register 16.35. SYSTIMER_REAL_TARGET2_HI_REG (0x0088)



SYSTIMER_TARGET2_HI_RO Represents the actual target value of COMP2, high 20 bits. (RO)

Register 16.36. SYSTIMER_DATE_REG (0x00FC)



SYSTIMER_DATE Version control register. (R/W)

Chapter 17

Timer Group (TIMG)

17.1 Overview

General-purpose timers can be used to precisely time an interval, trigger an interrupt after a particular interval (periodically and aperiodically), or act as a hardware clock. As shown in Figure 17.1-1, the ESP32-P4 chip contains two timer groups, namely timer group 0 and timer group 1 (hereinafter referred to as TIMG*n*, where *n* can be 0 or 1). Each timer group consists of two general-purpose timers (hereinafter referred to as T*x*, where *x* can be 0 or 1) and one Main System Watchdog Timer (MWDT). The general-purpose timer is based on a 16-bit prescaler and a 54-bit auto-reload-capable up-down counter.

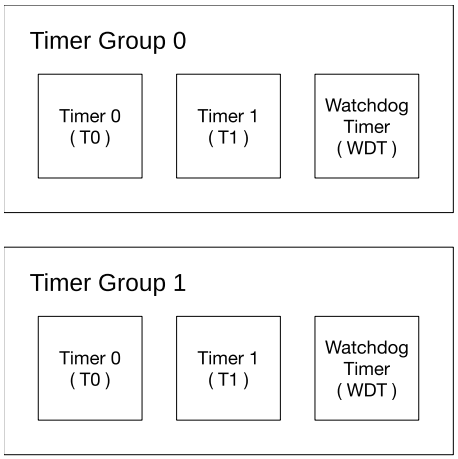


Figure 17.1-1. Timer Group Overview

Note that while the Main System Watchdog Timer registers are described in this chapter, their functional description is included in Chapter 18 *Watchdog Timers (WDT)*. Therefore, the term "timer" within this chapter refers to the general-purpose timer.

17.2 Features

The timer's features are summarized as follows:

- A 54-bit time-base counter programmable to incrementing or decrementing
- Three clock sources: PLL_F80M_CLK or XTAL_CLK or RC_FAST_CLK
- A 16-bit clock prescaler, from 2 to 65536
- Able to read real-time value of the time-base counter
- Able to halt and resume the time-base counter

- Programmable alarm generation
- Timer value reload — Auto-reload at alarm or software-controlled instant reload
- Calculate clock frequency — Calculate the measured frequency of the clock, which can be called TIMG0_CALI_CLK, based on the crystal clock
- Level interrupt generation
- Support several ETM tasks and events

17.3 Functional Description

Figure 17.3-1 shows Timer T*x* in timer group TIMG*n*. T*x* contains a 16-bit integer divider as a prescaler, a timer-based counter, and a comparator for alarm generation.

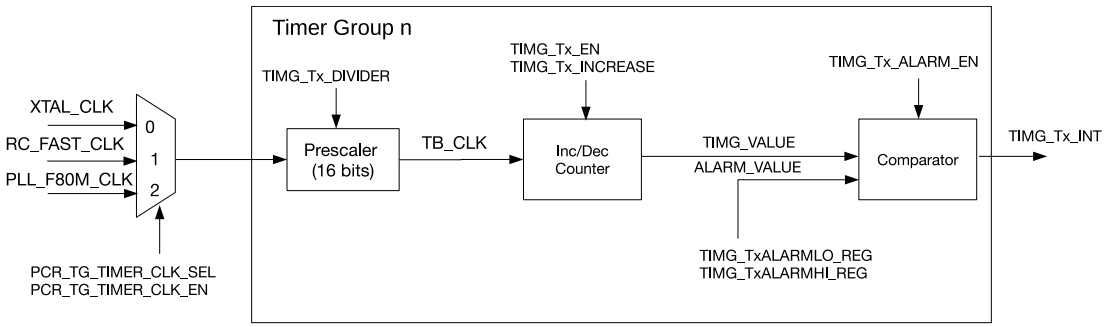


Figure 17.3-1. Timer Group Architecture

17.3.1 16-bit Prescaler and Clock Selection

Take the Timer T*x* in timer group TIMG*n* as an example:

- The timer can select its clock source by setting the HP_SYS_CLKRST_TIMERGRP*n*_T*x*_SRC_SEL field of the HP_SYS_CLKRST_PERI_CLK_CTRL20_REG register or the HP_SYS_CLKRST_PERI_CLK_CTRL21_REG register. When the field is 0, XTAL_CLK is selected; when the field is 1, RC_FAST_CLK is selected and when the field is 2, PLL_F80M_CLK is selected.
- The selected clock can be switched on by setting HP_SYS_CLKRST_TIMERGRP*n*_T*x*_CLK_EN field of the HP_SYS_CLKRST_PERI_CLK_CTRL20_REG register or the HP_SYS_CLKRST_PERI_CLK_CTRL21_REG register to 1 and switched off by setting it to 0. The clock is then divided by a 16-bit prescaler to generate the time-base counter clock (TB_CLK) used by the time-base counter. The divisor of the prescaler can be configured through the TIMG_T*x*_DIVIDER field.

TIMG_T*x*_DIVIDER field can be configured as 0 ~ 65535 for a divisor range of 2 ~ 65536. To be more specific, when TIMG_T*x*_DIVIDER is configured as:

- 0: the divisor is 65536
- 1: the divisor is 2
- 2: the divisor is also 2
- 3 ~ 65535: the divisor is 3 ~ 65535

To modify the 16-bit prescaler, please first configure the TIMG_T*x*_DIVIDER field, and then set TIMG_T*x*_DIVCNT_RST to 1. Meanwhile, the timer must be disabled (i.e., TIMG_T*x*_EN should be cleared). Otherwise, the result can be unpredictable.

17.3.2 54-bit Time-base Counter

The 54-bit time-base counter is based on TB_CLK and can be configured to increment or decrement via the TIMG_T*x*_INCREASE field. The time-base counter can be enabled or disabled by setting or clearing the TIMG_T*x*_EN field, respectively. When enabled, the time-base counter increments or decrements on each cycle of TB_CLK. When disabled, the time-base counter is essentially frozen. Note that the TIMG_T*x*_INCREASE field can be changed no matter whether TIMG_T*x*_EN is set or not, and this will cause the time-base counter to change direction instantly.

To read the 54-bit value of the time-base counter, the timer value must be latched to two registers before being read by the CPU (due to the CPU being 32-bit). By writing any value to the TIMG_T*x*UPDATE_REG, the current value of the 54-bit timer starts to be latched into the TIMG_T*x*LO_REG and TIMG_T*x*HI_REG registers containing the lower 32-bits and higher 22-bits, respectively. When TIMG_T*x*UPDATE_REG is cleared by hardware, it indicates the latch operation has been completed and current timer value can be read from the TIMG_T*x*LO_REG and TIMG_T*x*HI_REG registers. TIMG_T*x*LO_REG and TIMG_T*x*HI_REG registers will remain unchanged for the CPU to read in its own time until TIMG_T*x*UPDATE_REG is written to again.

17.3.3 Alarm Generation

A timer can be configured to trigger an alarm when the timer's current value matches the alarm value. An alarm will cause an interrupt to occur and (optionally) an automatic reload of the timer's current value (see Section 17.3.4).

The 54-bit alarm value is configured using TIMG_T*x*ALARMLO_REG and TIMG_T*x*ALARMHI_REG, which represent the lower 32-bits and higher 22-bits of the alarm value, respectively. However, the configured alarm value is ineffective until the alarm is enabled by setting the TIMG_T*x*_ALARM_EN field. To avoid alarm being enabled "too late" (i.e., the timer value has already passed the alarm value when the alarm is enabled), the hardware will trigger the alarm immediately if the current timer value is:

- higher than the alarm value (within a defined range) when the up-down counter increments
- lower than the alarm value (within a defined range) when the up-down counter decrements

Table 17.3-1 and Table 17.3-2 show the relationship between the current value of the timer, the alarm value, and when an alarm is triggered. The current time value and the alarm value are defined as follows:

- TIMG_VALUE = {TIMG_T*x*HI_REG, TIMG_T*x*LO_REG}
- ALARM_VALUE = {TIMG_T*x*ALARMHI_REG, TIMG_T*x*ALARMLO_REG}

Table 17.3-1. Alarm Generation When Up-Down Counter Increments

Scenario	Range	Alarm
1	ALARM_VALUE − TIMG_VALUE $> 2^{53}$	Triggered
2	0 < ALARM_VALUE − TIMG_VALUE $\leq 2^{53}$	Triggered when the up-down counter counts TIMG_VALUE up to ALARM_VALUE
3	0 $\leq$ TIMG_VALUE − ALARM_VALUE $< 2^{53}$	Triggered
4	TIMG_VALUE − ALARM_VALUE $\geq 2^{53}$	Triggered when the up-down counter restarts counting up from 0 after reaching the timer's maximum value and counts TIMG_VALUE up to ALARM_VALUE

Table 17.3-2. Alarm Generation When Up-Down Counter Decrements

Scenario	Range	Alarm
5	TIMG_VALUE − ALARM_VALUE $> 2^{53}$	Triggered
6	0 < TIMG_VALUE − ALARM_VALUE $\leq 2^{53}$	Triggered when the up-down counter counts TIMG_VALUE down to ALARM_VALUE
7	0 $\leq$ ALARM_VALUE − TIMG_VALUE $< 2^{53}$	Triggered
8	ALARM_VALUE − TIMG_VALUE $\geq 2^{53}$	Triggered when the up-down counter restarts counting down from the timer's maximum value after reaching the minimum value and counts TIMG_VALUE down to ALARM_VALUE

When an alarm occurs, the TIMG_T*x*_ALARM_EN field is automatically cleared and no alarm will occur again until the TIMG_T*x*_ALARM_EN is set next time.

17.3.4 Timer Reload

A timer is reloaded when a timer's current value is overwritten with a reload value stored in the TIMG_T*x*_LOAD_LO and TIMG_T*x*_LOAD_HI fields that correspond to the lower 32-bits and higher 22-bits of the timer's new value, respectively. However, writing a reload value to TIMG_T*x*_LOAD_LO and TIMG_T*x*_LOAD_HI will not cause the timer's current value to change. Instead, the reload value is ignored by the timer until a reload event occurs. A reload event can be triggered either by a software instant reload or an auto-reload at alarm.

A software instant reload is triggered by the CPU writing any value to TIMG_T*x*LOAD_REG, which causes the timer's current value to be instantly reloaded. If TIMG_T*x*_EN is set, the timer will continue incrementing or decrementing from the new value. In this case if TIMG_T*x*_ALARM_EN is set, the timer will still trigger alarms in scenarios listed in Table 17.3-1 and 17.3-2. If TIMG_T*x*_EN is cleared, the timer will remain frozen at the new value until counting is re-enabled.

An auto-reload at alarm will cause a timer reload when an alarm occurs, thus allowing the timer to continue incrementing or decrementing from the reload value. This is generally useful for resetting the timer's value when using periodic alarms. To enable auto-reload at alarm, the TIMG_T*x*_AUTORELOAD field should be set. If not enabled, the timer's value will continue to increment or decrement past the alarm value after an alarm.

17.3.5 Event Task Matrix Feature

The TIMG*n* on ESP32-P4 supports the Event Task Matrix (ETM) function, which allows TIMG*n*'s ETM tasks to be triggered by any peripherals' ETM events, or TIMG*n*'s ETM events to trigger any peripherals' ETM tasks. This section introduces the ETM tasks and events related to TIMG*n*. For more information, please refer to Chapter 14 Event Task Matrix (ETM).

TIMG*n* can receive the following ETM tasks:

- TG0_TASK_CNT_START_TIMER0: When triggered, it will enable the time-base counter.
- TG1_TASK_CNT_START_TIMER0: When triggered, it will enable the time-base counter.
- TG0_TASK_CNT_STOP_TIMER0: When triggered, it will disable the time-base counter.
- TG1_TASK_CNT_STOP_TIMER0: When triggered, it will disable the time-base counter.

> **Note:**
> The above two ETM tasks have the same function as the APB configuration TIMG_T0_EN. When these operations occur at the same time, the priority of each operation from high to low is as follows:
>
> 1. TG0_TASK_CNT_START_TIMER0 and TG1_TASK_CNT_START_TIMER0: When triggered, it will enable the time-base counter;
> 2. TG0_TASK_CNT_STOP_TIMER0 and TG1_TASK_CNT_STOP_TIMER0: When triggered, it will disable the time-base counter;
> 3. APB configuration TIMG_T0_EN: Enable or disable the time-base counter.

- TG0_TASK_ALARM_START_TIMER0: When triggered, it will enable the alarm generation.
- TG1_TASK_ALARM_START_TIMER0: When triggered, it will enable the alarm generation.

> **Note:**
> Alarm generation can also be enabled through APB method configuring TIMG_T0_ALARM_EN and hardware events. When these operations occur at the same time, the priority of each operation from high to low is as follows:
>
> 1. TG0_TASK_ALARM_START_TIMER0 and TG1_TASK_ALARM_START_TIMER0: When triggered, it will enable the alarm generation;
> 2. Alarm events: When triggered, it will disable the alarm generation;
> 3. APB configuration TIMG_T0_ALARM_EN: Enable or disable the alarm generation.

- TG0_TASK_CNT_CAP_TIMER0: When triggered, it will update the current counter value to the TIMG_T0LO_REG and TIMG_T0HI_REG registers.
- TG1_TASK_CNT_CAP_TIMER0: When triggered, it will update the current counter value to the TIMG_T0LO_REG and TIMG_T0HI_REG registers.
- TG0_TASK_CNT_RELOAD_TIMER0: When triggered, it will overwrite the current counter value with the reload value stored in TIMG_T0_LOAD_LO and TIMG_T0_LOAD_HI.
- TG1_TASK_CNT_RELOAD_TIMER0: When triggered, it will overwrite the current counter value with the

reload value stored in TIMG_T0_LOAD_LO and TIMG_T0_LOAD_HI.

TIMG*n* can generate the following ETM events:

- TG0_EVT_CNT_CMP_TIMER0: Indicates the interrupt event of T0 in TIMG0.
- TG1_EVT_CNT_CMP_TIMER0: Indicates the interrupt event of T0 in TIMG1.

> **Note:**
> All the ETM tasks and events of TIMG*n* will not take effect until the TIMG*n*_ETM_EN is set to 1.

In practical applications, timer groups' ETM events can trigger their own ETM tasks.

For example, TG0_TASK_ALARM_START_TIMER0 and TG1_TASK_ALARM_START_TIMER0 can be triggered respectively by TG0_EVT_CNT_CMP_TIMER0 and TG1_EVT_CNT_CMP_TIMER0 to realize periodic alarm. For configuration steps, please refer to 17.4.4 Timer as Periodic Alarm by ETM.

17.3.6 RTC_SLOW_CLK Frequency Calculation

Using XTAL_CLK as a reference, it is possible to calculate the frequency of clock sources for RTC_SLOW_CLK (i.e., RC_SLOW_CLK, RC_FAST_DIV_CLK, and XTAL32K_CLK) as follows:

1. Start periodic or one-shot frequency calculation (see Section 17.4.5 for details);
2. Once receiving the signal to start calculation, the counter of XTAL_CLK and the counter of RTC_SLOW_CLK begin to work at the same time. When the counter of RTC_SLOW_CLK counts to C0, the two counters stop counting simultaneously;
3. Assume the value of XTAL_CLK's counter is C1, and the frequency of RTC_SLOW_CLK would be calculated as: $f_rtc = \frac{C0 \times f_XTAL_CLK}{C1}$

17.3.7 Interrupts

ESP32-P4's TIMG*n* can generate the following interrupt signal(s) that will be sent to the Interrupt Matrix.

- TG*n*_T*x*_INT
- TG*n*_WDT_INT

There are several internal interrupt sources from TIMG*n* that can generate the above interrupt signal(s). The interrupt sources from TIMG*n* are listed with their trigger conditions and the resulted interrupt signal(s) in Table 17.3-3.

Table 17.3-3. TIMG*n*'s Internal Interrupt Sources

Internal Interrupt Source	Trigger Condition	Interrupt Signal
TIMG_T*x*_INT	Alarm generated by T*x*	TG*n*_T*x*_INT
TIMG_WDT_INT	Alarm generated by WDT	TG*n*_WDT_INT

> **Note:**
> For definitions of *interrupt*, *interrupt signal*, *interrupt source*, and their correlations, please refer to Chapter 13 *Interrupt Matrix* > Section 13.2 *Interrupt Terminology in ESP32-P4*.

Each interrupt source can be configured by a common set of registers that are described in Section *Interrupt Configuration Registers*. The specific registers can be found in Section 17.5 *Register Summary*.

17.4 Configuration and Usage

17.4.1 Timer as a Simple Clock

1. Configure the time-base counter
   - Select clock source by configuring the HP_SYS_CLKRST_TIMERGRP0_TGRT_CLK_SRC_SEL field of the HP_SYS_CLKRST_PERI_CLK_CTRL21_REG register.
   - Configure the 16-bit prescaler by setting TIMG_T*x*_DIVIDER.
   - Configure the timer direction by setting or clearing TIMG_T*x*_INCREASE.
   - Set the timer's starting value by writing the starting value to TIMG_T*x*_LOAD_LO and TIMG_T*x*_LOAD_HI, then reloading it into the timer by writing any value to TIMG_T*x*LOAD_REG.
2. Start the timer by setting TIMG_T*x*_EN.
3. Get the timer's current value.
   - Write any value to TIMG_T*x*UPDATE_REG to latch the timer's current value.
   - Wait until TIMG_T*x*UPDATE_REG is cleared by hardware.
   - Read the latched timer value from TIMG_T*x*LO_REG and TIMG_T*x*HI_REG.

17.4.2 Timer as One-shot Alarm

1. Configure the time-base counter following step 1 of Section 17.4.1.
2. Configure the alarm.
   - Configure the alarm value by setting TIMG_T*x*ALARMLO_REG and TIMG_T*x*ALARMHI_REG.
   - Enable interrupt by setting TIMG_T*x*_INT_ENA.
3. Disable auto reload by clearing TIMG_T*x*_AUTORELOAD.
4. Start the alarm by setting TIMG_T*x*_ALARM_EN.
5. Handle the alarm interrupt.
   - Clear the interrupt by setting the timer's corresponding bit in TIMG_T*x*_INT_CLR.
   - Disable the timer by clearing TIMG_T*x*_EN.

17.4.3 Timer as Periodic Alarm by APB

1. Configure the time-base counter following step 1 in Section 17.4.1.
2. Configure the alarm following step 2 in Section 17.4.2.
3. Enable auto reload by setting TIMG_T*x*_AUTORELOAD and configure the reload value via TIMG_T*x*_LOAD_LO and TIMG_T*x*_LOAD_HI.

4. Start the alarm by setting TIMG_Tx_ALARM_EN.
5. Handle the alarm interrupt (repeat on each alarm iteration).
   - Clear the interrupt by setting the timer's corresponding bit in TIMG_Tx_INT_CLR.
   - If the next alarm requires a new alarm value and reload value (i.e., different alarm interval per iteration), then TIMG_TxALARMLO_REG, TIMG_TxALARMHI_REG, TIMG_Tx_LOAD_LO, and TIMG_Tx_LOAD_HI should be reconfigured as needed. Otherwise, the aforementioned registers should remain unchanged.
   - Re-enable the alarm by setting TIMG_Tx_ALARM_EN.
6. Stop the timer (on final alarm iteration).
   - Clear the interrupt by setting the timer's corresponding bit in TIMG_Tx_INT_CLR.
   - Disable the timer by clearing TIMG_Tx_EN.

17.4.4 Timer as Periodic Alarm by ETM

1. Enable the ETM module's clock
2. Map ETM event to ETM task (which means using the event to trigger the task)
   - If TIMG_T0_AUTORELOAD is set to 1, map TG0_EVT_CNT_CMP_TIMER0 and TG1_EVT_CNT_CMP_TIMER0 to the TG0_TASK_ALARM_START_TIMER0 and TG1_TASK_ALARM_START_TIMER0 respectively by one ETM channel.
   - If TIMG_T0_AUTORELOAD is set to 0, in addition to mapping TG0_EVT_CNT_CMP_TIMER0 and TG1_EVT_CNT_CMP_TIMER0 to the TG0_TASK_ALARM_START_TIMER0 and TG1_TASK_ALARM_START_TIMER0, the TG0_EVT_CNT_CMP_TIMER0 and TG1_EVT_CNT_CMP_TIMER0 should also be mapped to TG0_TASK_CNT_RELOAD_TIMER0 and TG1_TASK_CNT_RELOAD_TIMER0 by another ETM channel.
3. Choose to enable the one or two ETM channels.
4. Set TIMER_ETM_EN to 1 to enable timer group's ETM events and tasks.
5. Configure the time-base counter following step 1 in Section 17.4.1.
6. Configure the alarm following step 2 in Section 17.4.2.
7. Configure the reload value via TIMG_T0_LOAD_LO and TIMG_T0_LOAD_HI.
8. Handle the TG0_EVT_CNT_CMP_TIMER0 and TG1_EVT_CNT_CMP_TIMER0.
   - When alarm generates, the TG0_EVT_CNT_CMP_TIMER0 and TG1_EVT_CNT_CMP_TIMER0 also generate, and the alarm generation will be disabled by the alarm.
   - If TIMG_T0_AUTORELOAD is 1, the current counter value is overwritten by the reloaded value. The alarm generation will be reopened by TG0_TASK_ALARM_START_TIMER0 and TG1_TASK_ALARM_START_TIMER0.
   - If TIMG_T0_AUTORELOAD is 0, the current counter value is overwritten by the reloaded value because of the TG0_TASK_CNT_RELOAD_TIMER0 and TG1_TASK_CNT_RELOAD_TIMER0. The alarm generation will be reopened by TG0_TASK_ALARM_START_TIMER0 and TG1_TASK_ALARM_START_TIMER0.

9. Stop the timer (on final alarm iteration).
    - Disable the ETM channels used to map timer group's event and task
    - Set TIMER_ETM_EN to 0.
    - Clear the interrupt by setting the timer's corresponding bit in TIMG_TO_INT_CLR.
    - Disable the timer by clearing TIMG_TO_EN.

17.4.5 RTC_SLOW_CLK Frequency Calculation

1. One-shot frequency calculation
    - Configure HP_SYS_CLKRST_TIMERGRP0_TGRT_CLK_EN to enable clock gate of TIMG0_CALI_CLK.
    - Configure HP_SYS_CLKRST_TIMERGRP0_TGRT_CLK_SRC_SEL to select one clock as the clock source of TIMG0_CALI_CLK.
    - Configure HP_SYS_CLKRST_TIMERGRP0_TGRT_CLK_DIV_NUM to output TIMG0_CALI_CLK after integer prescaler.
    - Configure the time of calculation via TIMG_RTC_CALI_MAX.
    - Select one-shot frequency calculation by clearing TIMG_RTC_CALI_START_CYCLING, and enable the two counters via TIMG_RTC_CALI_START.
    - Once TIMG_RTC_CALI_RDY becomes 1, read TIMG_RTC_CALI_VALUE to get the value of XTAL_CLK's counter, and calculate the frequency of RTC_SLOW_CLK according to the formula in Section 17.3.6.
2. Periodic frequency calculation
    - Select the clock whose frequency is to be calculated (clock source of RTC_SLOW_CLK) via HP_SYS_CLKRST_TIMERGRP0_TGRT_CLK_SRC_SEL, and configure the time of calculation via TIMG_RTC_CALI_MAX.
    - Select periodic frequency calculation by enabling TIMG_RTC_CALI_START_CYCLING.
    - When TIMG_RTC_CALI_CYCLING_DATA_VLD is 1, TIMG_RTC_CALI_VALUE is valid.
3. Timeout
   If the counter of RTC_SLOW_CLK cannot finish counting in TIMG_RTC_CALI_TIMEOUT_RST_CNT cycles, TIMG_RTC_CALI_TIMEOUT will be set to indicate a timeout.

17.5 Register Summary

The addresses in this section are relative to Timer Group base address provided in Table 9.3-2 in Chapter 9 *System and Memory*.

The abbreviations given in Column **Access** are explained in Section *Access Types for Registers*.

Name	Description	Address	Access
T0 Control and configuration registers			
TIMG_TOCONFIG_REG	Timer 0 configuration register	0x0000	varies
TIMG_TOLO_REG	Timer 0 current value, low 32 bits	0x0004	RO
TIMG_TOHI_REG	Timer 0 current value, high 22 bits	0x0008	RO
TIMG_TOUPDATE_REG	Write to copy current timer value to TIMG_TOLO_REG or TIMG_TOHI_REG	0x000C	R/W/SC
TIMG_TOALARMLO_REG	Timer 0 alarm value, low 32 bits	0x0010	R/W
TIMG_TOALARMHI_REG	Timer 0 alarm value, high 22 bits	0x0014	R/W
TIMG_TOLOADLO_REG	Timer 0 reload value, low 32 bits	0x0018	R/W
TIMG_TOLOADHI_REG	Timer 0 reload value, high 22 bits	0x001C	R/W
TIMG_TOLOAD_REG	Write to reload timer from TIMG_TOLOADLO_REG or TIMG_TOLOADHI_REG	0x0020	WT
T1 Control and configuration registers			
TIMG_T1CONFIG_REG	Timer 1 configuration register	0x0024	varies
TIMG_T1LO_REG	Timer 1 current value, low 32 bits	0x0028	RO
TIMG_T1HI_REG	Timer 1 current value, high 22 bits	0x002C	RO
TIMG_T1UPDATE_REG	Write to copy current timer value to TIMG_T1LO_REG or TIMG_T1HI_REG	0x0030	R/W/SC
TIMG_T1ALARMLO_REG	Timer 1 alarm value, low 32 bits	0x0034	R/W
TIMG_T1ALARMHI_REG	Timer 1 alarm value, high 22 bits	0x0038	R/W
TIMG_T1LOADLO_REG	Timer 1 reload value, low 32 bits	0x003C	R/W
TIMG_T1LOADHI_REG	Timer 1 reload value, high 22 bits	0x0040	R/W
TIMG_T1LOAD_REG	Write to reload timer from TIMG_T1LOADLO_REG or TIMG_T1LOADHI_REG	0x0044	WT
WDT Control and configuration registers			
TIMG_WDTCONFIG0_REG	Watchdog timer configuration register	0x0048	varies
TIMG_WDTCONFIG1_REG	Watchdog timer prescaler register	0x004C	varies
TIMG_WDTCONFIG2_REG	Watchdog timer stage 0 timeout value	0x0050	R/W
TIMG_WDTCONFIG3_REG	Watchdog timer stage 1 timeout value	0x0054	R/W
TIMG_WDTCONFIG4_REG	Watchdog timer stage 2 timeout value	0x0058	R/W
TIMG_WDTCONFIG5_REG	Watchdog timer stage 3 timeout value	0x005C	R/W
TIMG_WDTFEED_REG	Write to feed the watchdog timer	0x0060	WT
TIMG_WDTWPROTECT_REG	Watchdog write protect register	0x0064	R/W
RTC CALI Control and configuration registers			
TIMG_RTCCALICFG_REG	RTC calibration configure register	0x0068	varies
TIMG_RTCCALICFG1_REG	RTC calibration configure register 1	0x006C	RO
TIMG_RTCCALICFG2_REG	RTC calibration configure register 2	0x0080	varies

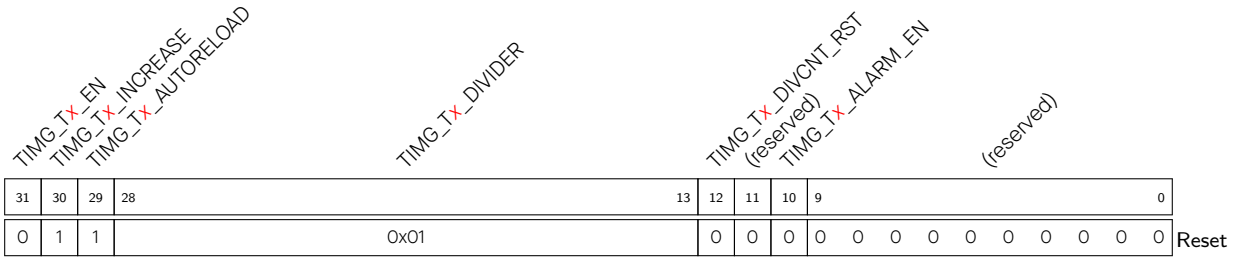
Name	Description	Address	Access
Interrupt registers			
TIMG_INT_ENA_TIMERS_REG	Interrupt enable bits	0x0070	R/W
TIMG_INT_RAW_TIMERS_REG	Raw interrupt status	0x0074	R/SS/WTC
TIMG_INT_ST_TIMERS_REG	Masked interrupt status	0x0078	RO
TIMG_INT_CLR_TIMERS_REG	Interrupt clear bits	0x007C	WT
Version register			
TIMG_NTIMERS_DATE_REG	Timer version control register	0x00F8	R/W
Clock configuration registers			
TIMG_REGCLK_REG	Timer group clock gate register	0x00FC	R/W

17.6 Registers

The addresses in this section are relative to Timer Group base address provided in Table 9.3-2 in Chapter 9 *System and Memory*.

For how to program reserved fields, please refer to Section Programming Reserved Register Field.

Register 17.1. TIMG_TxCONFIG_REG (x: 0-1) (0x0000+0x24*x)



TIMG_Tx_ALARM_EN Configures whether to enable Timer Tx alarm function. This bit will be automatically cleared once an alarm occurs.

0: Disable

1: Enable

(R/W/SC)

TIMG_Tx_DIVCNT_RST Configures to reset Timer Tx's clock divider counter.

0: No effect

1: Reset

(WT)

TIMG_Tx_DIVIDER Represents Timer *x* clock (Tx_clk) prescaler value. (R/W)

TIMG_Tx_AUTORELOAD Configures to enable Timer Tx auto-reload function at the time of alarm.

0: No effect

1: Enable

(R/W)

TIMG_Tx_INCREASE Configures the counting direction of Timer Tx time-base counter.

0: Decrement

1: Increment

(R/W)

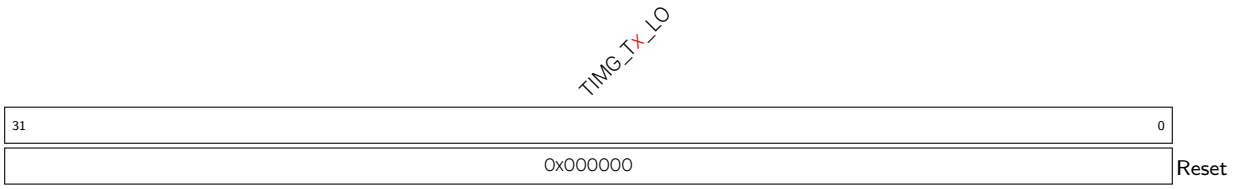
TIMG_Tx_EN Configures whether to enable Timer Tx time-base counter.

0: Disable

1: Enable

(R/W/SS/SC)

Register 17.2. TIMG_T*x*LO_REG (*x*: 0-1) (0x0004+0x24x*)**

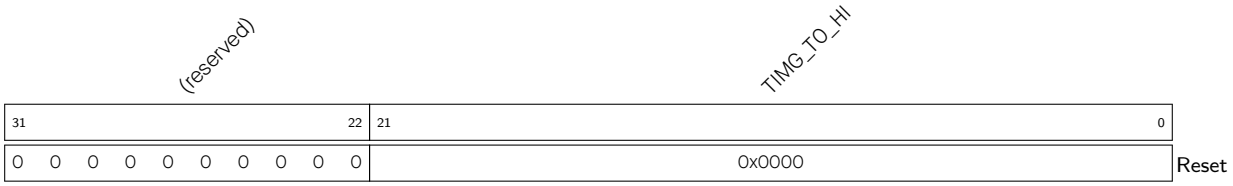


TIMG_T*x*_LO Represents the low 32 bits of the time-base counter of Timer T*x*. Valid only after writing to TIMG_T*x*UPDATE_REG.

Measurement unit: T*x*_clk.

(RO)

Register 17.3. TIMG_T*x*HI_REG (*x*: 0-1) (0x0008+0x24x*)**

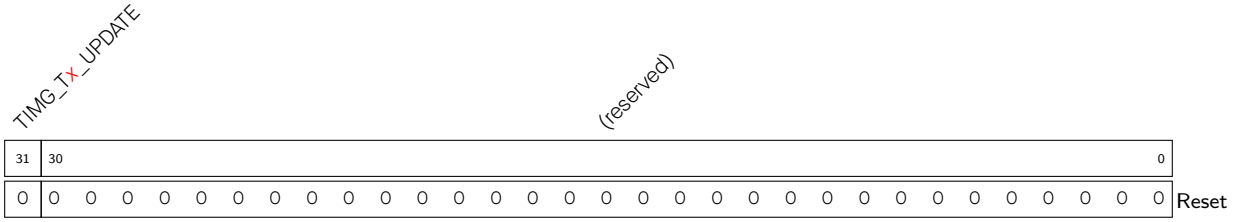


TIMG_T*x*_HI Represents the high 22 bits of the time-base counter of Timer T*x*. Valid only after writing to TIMG_T*x*UPDATE_REG.

Measurement unit: T*x*_clk.

(RO)

Register 17.4. TIMG_T*x*UPDATE_REG (*x*: 0-1) (0x000C+0x24x*)**



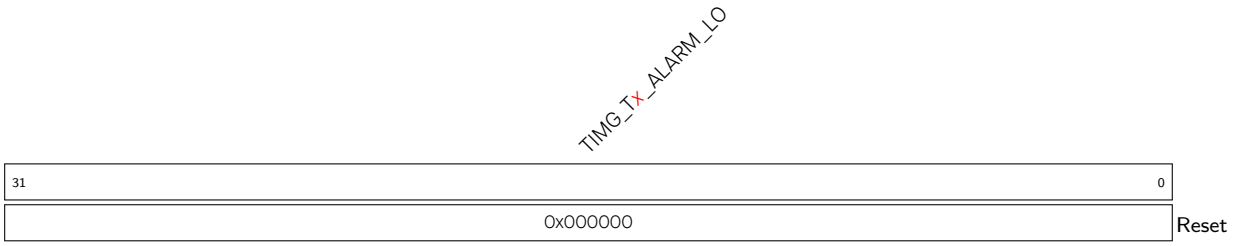
TIMG_T*x*_UPDATE Configures to latch the counter value.

0: Latch

1: Latch

(R/W/SC)

Register 17.5. TIMG_TxALARMLO_REG (x: 0-1) (0x0010+0x24*x)

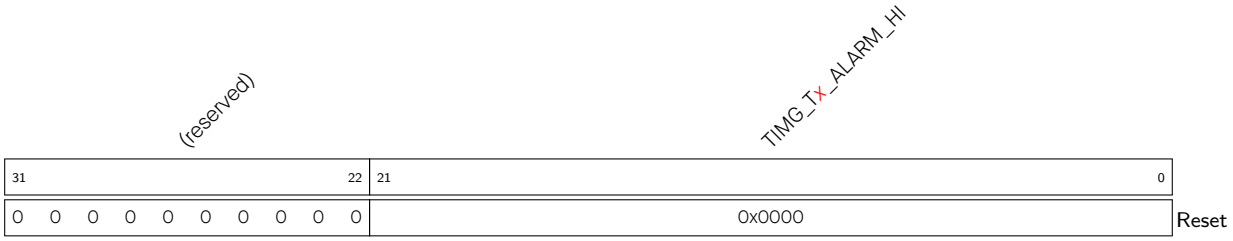


TIMG_Tx_ALARM_LO Configures the low 32 bits of Timer Tx alarm trigger time-base counter value. Valid only when TIMG_Tx_ALARM_EN is 1.

Measurement unit: Tx_clk.

(R/W)

Register 17.6. TIMG_TxALARMHI_REG (x: 0-1) (0x0014+0x24*x)

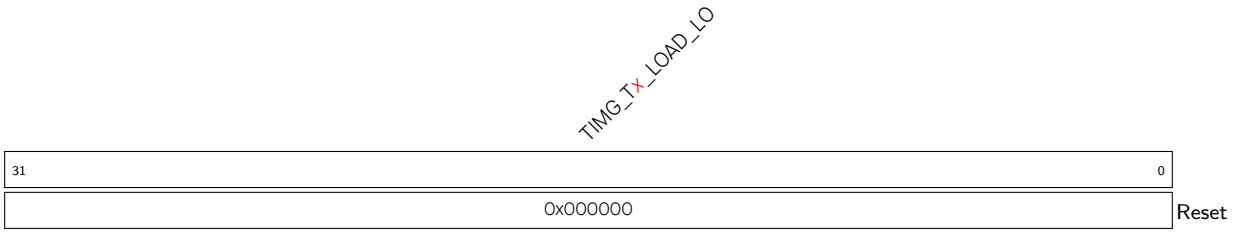


TIMG_Tx_ALARM_HI Configures the high 22 bits of Timer Tx alarm trigger time-base counter value. Valid only when TIMG_Tx_ALARM_EN is 1.

Measurement unit: Tx_clk.

(R/W)

Register 17.7. TIMG_TxLOADLO_REG (x: 0-1) (0x0018+0x24*x)

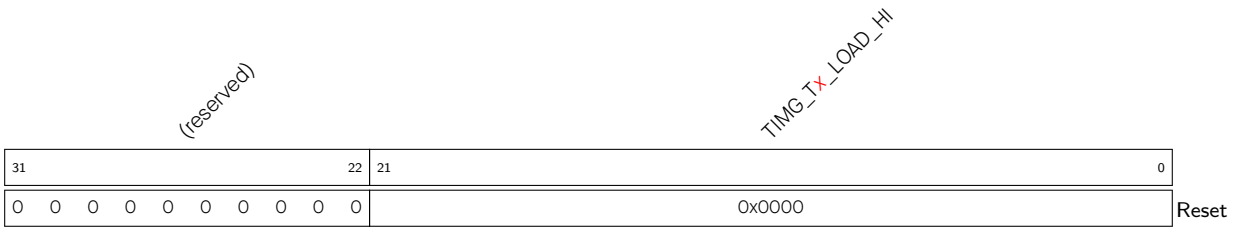


TIMG_Tx_LOAD_LO Configures low 32 bits of the value that a reload will load onto Timer Tx time-base counter.

Measurement unit: Tx_clk.

(R/W)

Register 17.8. TIMG_TxLOADHI_REG (x: 0-1) (0x001C+0x24*x)

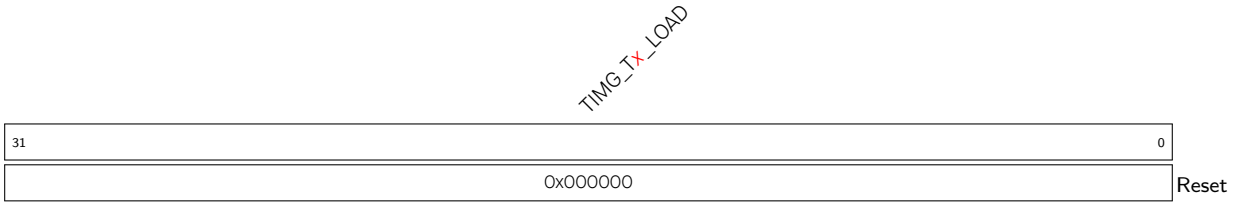


TIMG_Tx_LOAD_HI Configures high 22 bits of the value that a reload will load onto Timer Tx time-base counter.

Measurement unit: Tx_clk.

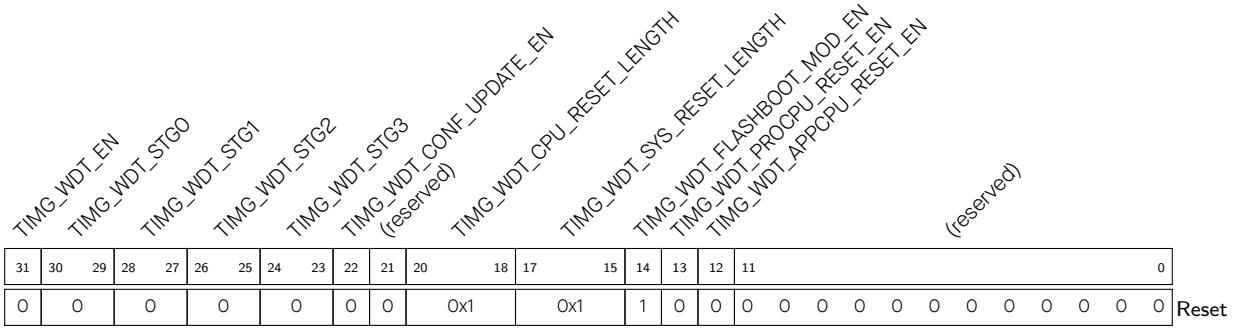
(R/W)

Register 17.9. TIMG_TxLOAD_REG (x: 0-1) (0x0020+0x24*x)



TIMG_Tx_LOAD Write any value to trigger Timer Tx time-base counter reload. (WT)

Register 17.10. TIMG_WDTCONFIG0_REG (0x0048)



TIMG_WDT_APPCPU_RESET_EN Configures whether to mask the APP CPU reset generated by MWDT. Valid only when write protection is disabled.

0: Mask

1: Unmask

(R/W)

TIMG_WDT_PROCPU_RESET_EN Configures whether to mask the PRO CPU reset generated by MWDT. Valid only when write protection is disabled.

0: Mask

1: Unmask

(R/W)

TIMG_WDT_FLASHBOOT_MOD_EN Configures whether to enable flash boot protection.

0: Disable

1: Enable

(R/W)

TIMG_WDT_SYS_RESET_LENGTH Configures the system reset signal length. Valid only when write protection is disabled.

Measurement unit: mwdt_clk.

0: 100 ns

1: 200 ns

2: 300 ns

3: 400 ns

4: 500 ns

5: 800 ns

6: 1.6 μs

7: 3.2 μs

(R/W)

Continued on the next page...

Register 17.10. TIMG_WDTCONFIG0_REG (0x0048)

Continued from the previous page...

TIMG_WDT_CPU_RESET_LENGTH Configures the CPU reset signal length. Valid only when write protection is disabled.
Measurement unit: mwdt_clk.

0: 100 ns
1: 200 ns
2: 300 ns
3: 400 ns
4: 500 ns
5: 800 ns
6: 1.6 μs
7: 3.2 μs

(R/W)

TIMG_WDT_CONF_UPDATE_EN Configures to update the WDT configuration registers.
0: No effect
1: Update
(WT)

TIMG_WDT_STG3 Configures the timeout action of stage 3. Valid only when write protection is disabled.
0: No effect
1: Interrupt
2: Reset CPU
3: Reset system
(R/W)

TIMG_WDT_STG2 Configures the timeout action of stage 2. Valid only when write protection is disabled.
0: No effect
1: Interrupt
2: Reset CPU
3: Reset system
(R/W)

TIMG_WDT_STG1 Configures the timeout action of stage 1. Valid only when write protection is disabled.
0: No effect
1: Interrupt
2: Reset CPU
3: Reset system
(R/W)

Continued on the next page...

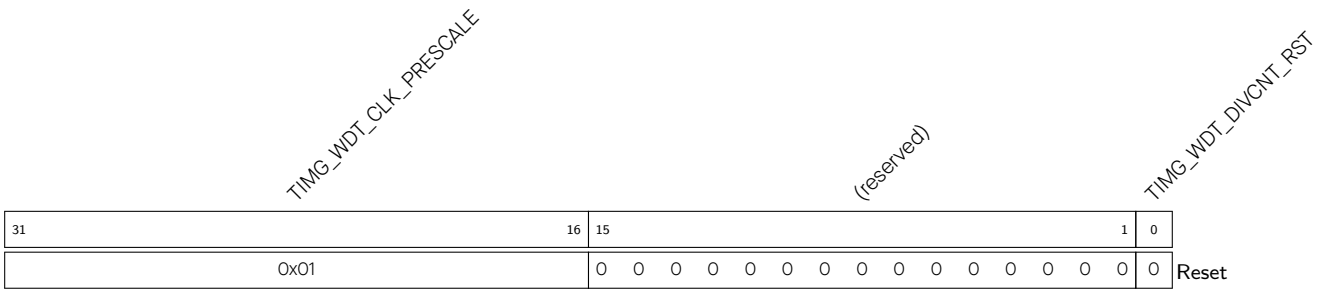
Register 17.10. TIMG_WDTCONFIG0_REG (0x0048)

Continued from the previous page...

TIMG_WDT_STG0 Configures the timeout action of stage 0. Valid only when write protection is disabled.
0: No effect
1: Interrupt
2: Reset CPU
3: Reset system
(R/W)

TIMG_WDT_EN Configures whether or not to enable the MWDT. Valid only when write protection is disabled.
0: Disable
1: Enable
(R/W)

Register 17.11. TIMG_WDTCONFIG1_REG (0x004C)



TIMG_WDT_DIVCNT_RST Configures whether to reset WDT's clock divider counter.

0: No effect

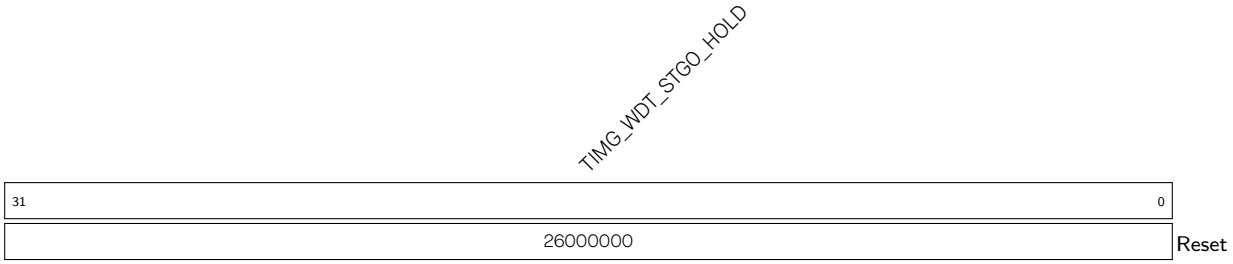
1: Reset

(WT)

TIMG_WDT_CLK_PRESCALE Configures MWDT clock prescaler value. Valid only when write protection is disabled.

MWDT clock period = 12.5 ns *TIMG_WDT_CLK_PRESCALE. (R/W)

Register 17.12. TIMG_WDTCONFIG2_REG (0x0050)

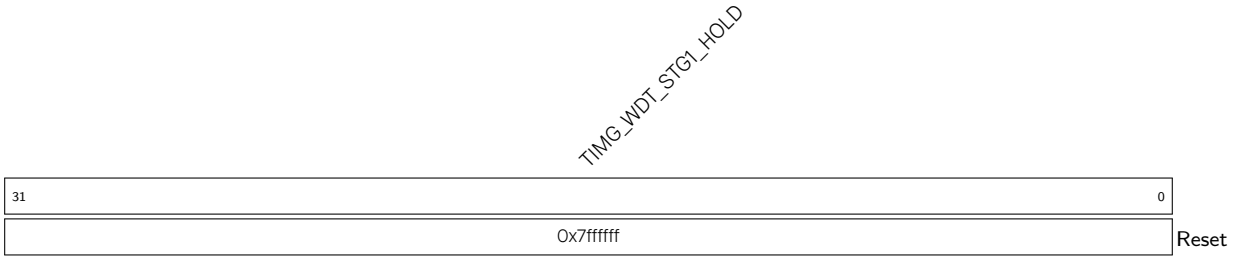


TIMG_WDT_STG0_HOLD Configures the stage 0 timeout value. Valid only when write protection is disabled.

Measurement unit: mwdt_clk.

(R/W)

Register 17.13. TIMG_WDTCONFIG3_REG (0x0054)

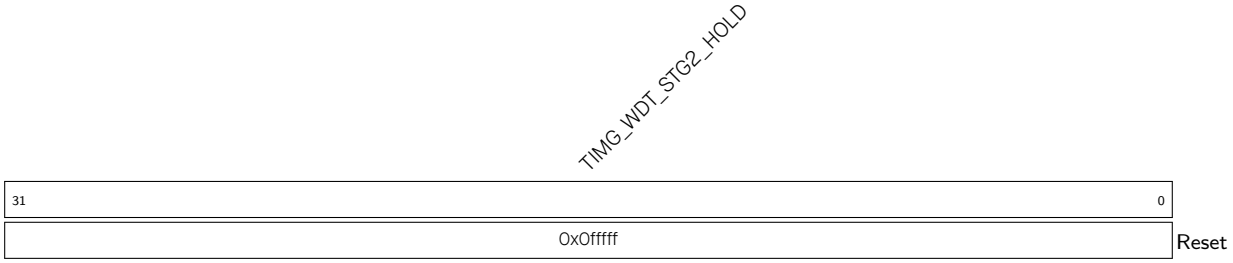


TIMG_WDT_STG1_HOLD Configures the stage 1 timeout value. Valid only when write protection is disabled.

Measurement unit: mwdt_clk.

(R/W)

Register 17.14. TIMG_WDTCONFIG4_REG (0x0058)

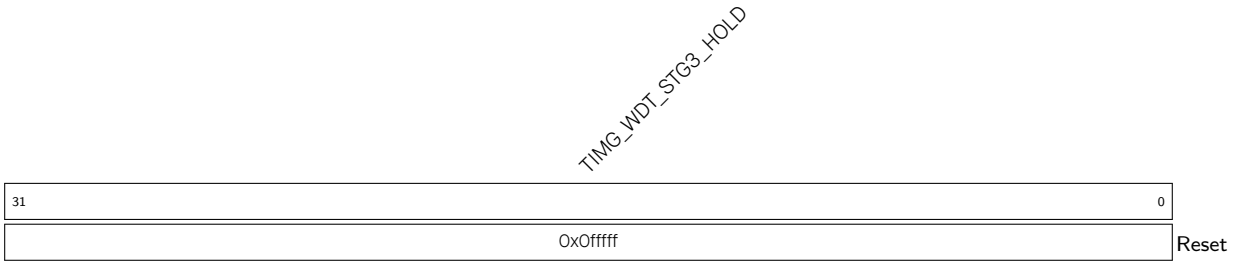


TIMG_WDT_STG2_HOLD Configures the stage 2 timeout value. Valid only when write protection is disabled.

Measurement unit: mwdt_clk.

(R/W)

Register 17.15. TIMG_WDTCONFIG5_REG (0x005C)

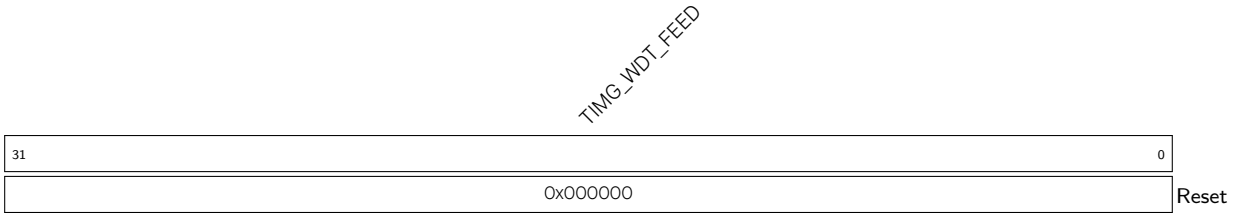


TIMG_WDT_STG3_HOLD Configures the stage 3 timeout value. Valid only when write protection is disabled.

Measurement unit: mwdt_clk.

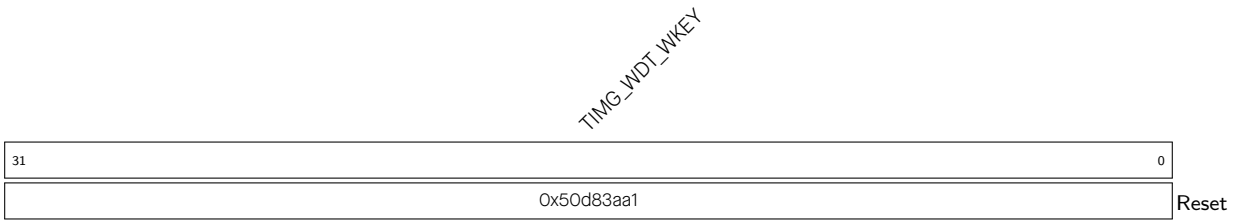
(R/W)

Register 17.16. TIMG_WDTFEED_REG (0x0060)



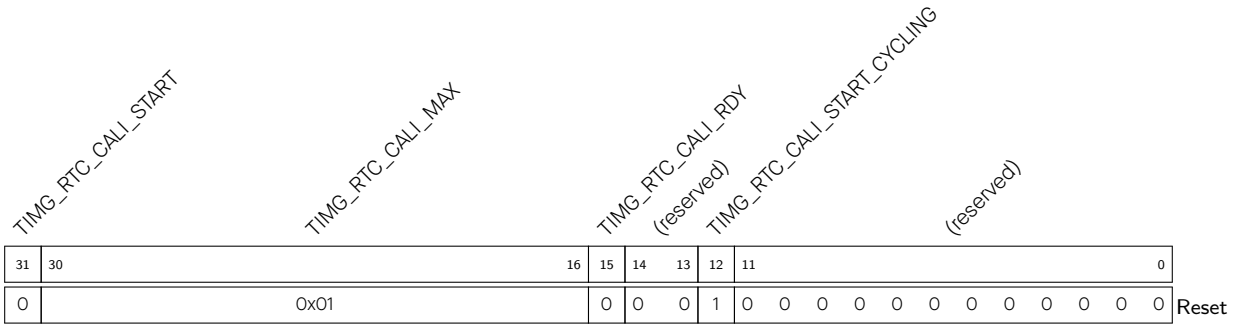
TIMG_WDT_FEED Write any value to feed the MWDT. Valid only when write protection is disabled. (WT)

Register 17.17. TIMG_WDTWPROTECT_REG (0x0064)



TIMG_WDT_WKEY Configures a different value than its reset value to enable write protection. (R/W)

Register 17.18. TIMG_RTCCALICFG_REG (0x0068)



TIMG_RTC_CALI_START_CYCLING Configures the frequency calculation mode.

0: One-shot frequency calculation

1: Periodic frequency calculation.

(R/W)

TIMG_RTC_CALI_RDY Represents whether one-shot frequency calculation is done.

0: Not done

1: Done

(RO)

TIMG_RTC_CALI_MAX Configures the time to calculate RTC slow clock's frequency.

Measurement unit: XTAL_CLK.

(R/W)

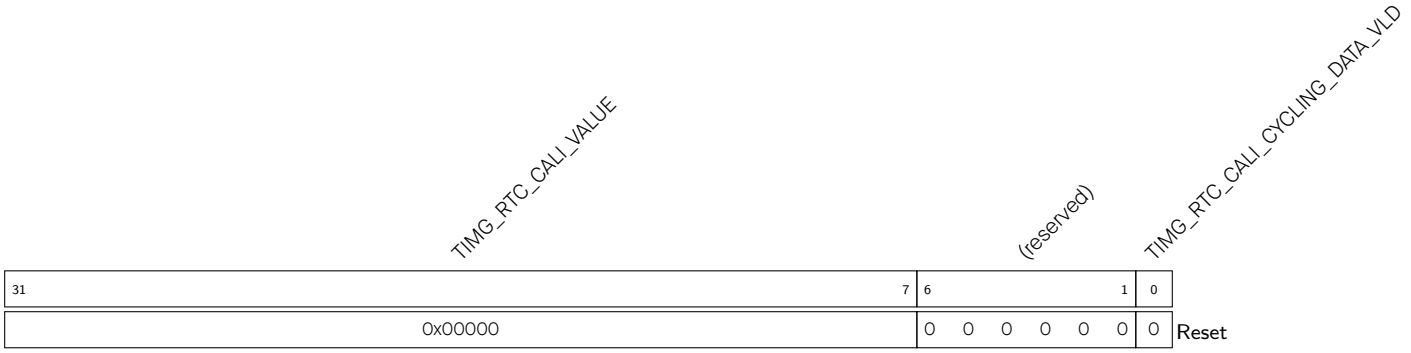
TIMG_RTC_CALI_START Configures whether to enable one-shot frequency calculation.

0: Disable

1: Enable

(R/W)

Register 17.19. TIMG_RTCCALICFG1_REG (0x006C)



TIMG_RTC_CALI_CYCLING_DATA_VLD Represents whether periodic frequency calculation is done.

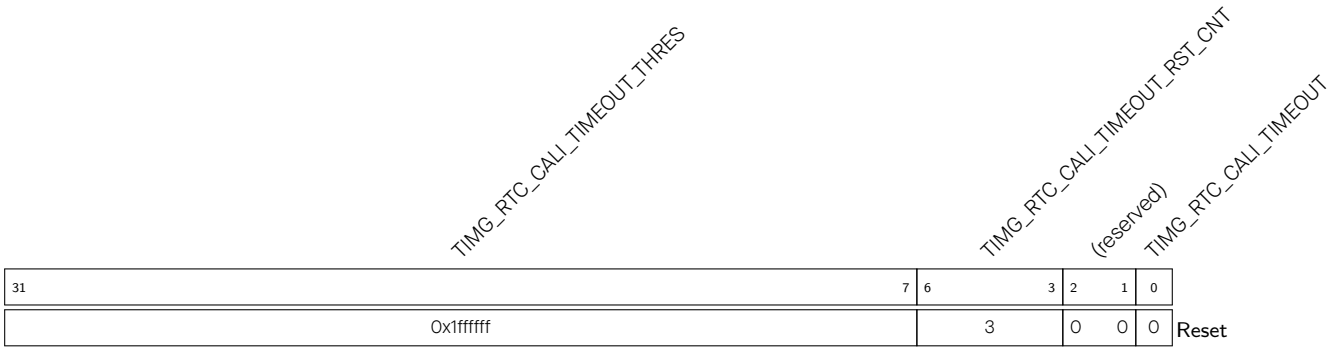
0: Not done

1: Done

(RO)

TIMG_RTC_CALI_VALUE Represents the value countered by XTAL_CLK when one-shot or periodic frequency calculation is done. It is used to calculate RTC slow clock's frequency. (RO)

Register 17.20. TIMG_RTCCALICFG2_REG (0x0080)



TIMG_RTC_CALI_TIMEOUT Represents whether RTC frequency calculation is timeout.

0: No timeout

1: Timeout

(RO)

TIMG_RTC_CALI_TIMEOUT_RST_CNT Configures the cycles that reset frequency calculation timeout.

Measurement unit: XTAL_CLK.

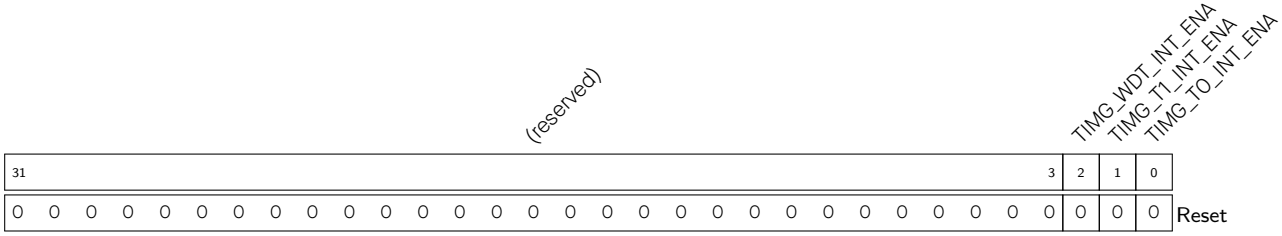
(R/W)

TIMG_RTC_CALI_TIMEOUT_THRES Configures the threshold value for the RTC frequency calculation timer. If the timer's value exceeds this threshold, a timeout is triggered.

Measurement unit: XTAL_CLK.

(R/W)

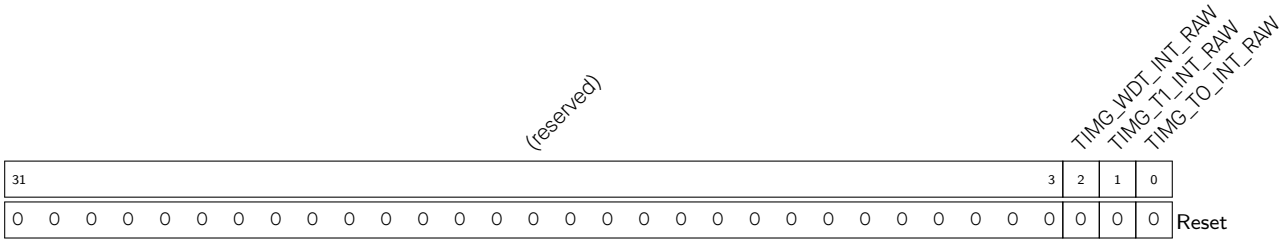
Register 17.21. TIMG_INT_ENA_TIMERS_REG (0x0070)



TIMG_T*x*_INT_ENA (*x*: 0-1) Write 1 to enable the TIMG_T*x*_INT interrupt. (R/W)

TIMG_WDT_INT_ENA Write 1 to enable the TIMG_WDT_INT interrupt. (R/W)

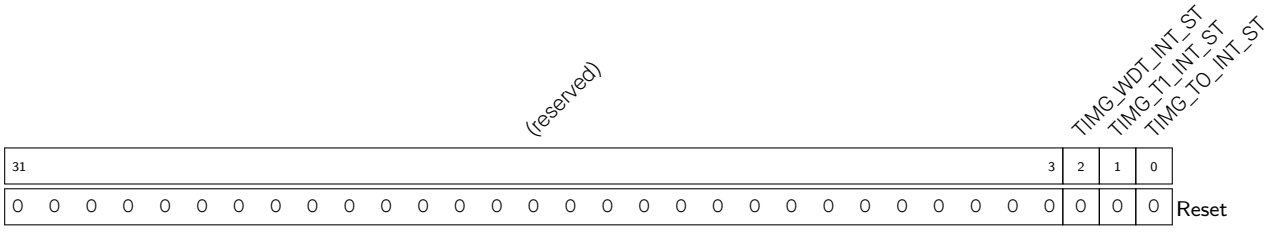
Register 17.22. TIMG_INT_RAW_TIMERS_REG (0x0074)



TIMG_T*x*_INT_RAW (*x*: 0-1) The raw interrupt status bit of the TIMG_T*x*_INT interrupt. (R/SS/WTC)

TIMG_WDT_INT_RAW The raw interrupt status bit of the TIMG_WDT_INT interrupt. (R/SS/WTC)

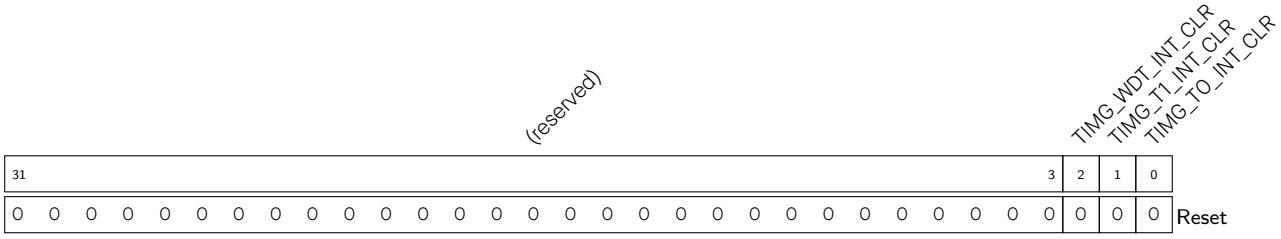
Register 17.23. TIMG_INT_ST_TIMERS_REG (0x0078)



TIMG_T*x*_INT_ST (*x*: 0-1) The masked interrupt status bit of the TIMG_T*x*_INT interrupt. (RO)

TIMG_WDT_INT_ST The masked interrupt status bit of the TIMG_WDT_INT interrupt. (RO)

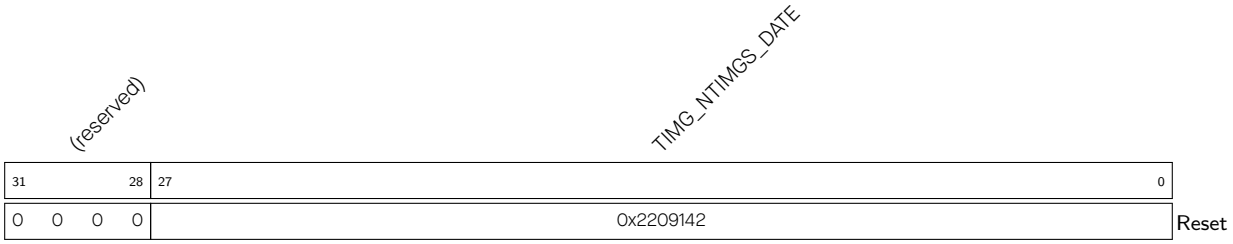
Register 17.24. TIMG_INT_CLR_TIMERS_REG (0x007C)



TIMG_T*x*_INT_CLR (*x*: 0-1) Write 1 to clear the TIMG_T*x*_INT interrupt. (WT)

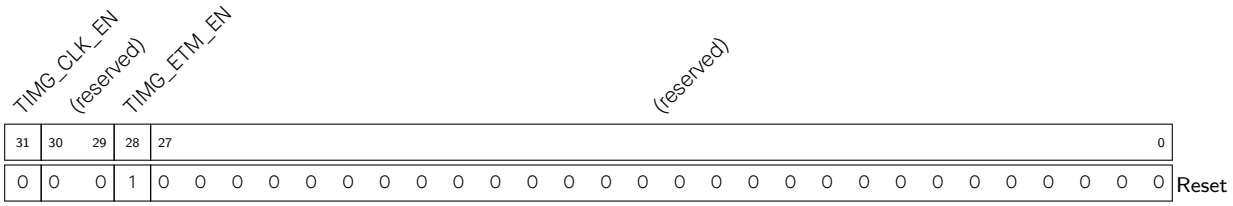
TIMG_WDT_INT_CLR Write 1 to clear the TIMG_WDT_INT interrupt. (WT)

Register 17.25. TIMG_NTIMERS_DATE_REG (0x00F8)



TIMG_NTIMGS_DATE Version control register (R/W)

Register 17.26. TIMG_REGCLK_REG (0x00FC)



TIMG_ETM_EN Configures whether to enable timer's ETM task and event.

0: Disable

1: Enable

(R/W)

TIMG_CLK_EN Configures whether to enable gate clock signal for registers.

0: Force clock on for registers

1: Support clock only when registers are read or written to by software.

(R/W)

Chapter 18

Watchdog Timers (WDT)

18.1 Overview

Watchdog timers are hardware timers used to detect and recover from malfunctions. They must be periodically fed (reset) to prevent a timeout. A system/software that is behaving unexpectedly (e.g., is stuck in a software loop or in overdue events) will fail to feed the watchdog, thus the watchdog timer will time out and trigger an interrupt or reset, directing the system to a known state for exception handling or a restart. Therefore, watchdog timers are useful for detecting and handling erroneous system/software behavior.

As shown in Figure 18.1-1, ESP32-P4 contains three digital watchdog timers: one in each of the two timer groups in Chapter 17 *Timer Group (TIMG)* (called Main System Watchdog Timers, or MWDT) and one in the LP system (called the RTC Watchdog Timer, or RWDT). Each digital watchdog timer allows for four separately configurable stages, and each stage can be programmed to take one action upon timeout, unless the watchdog is fed or disabled. MWDT supports three timeout actions: interrupt, HP CPU reset, and HP core reset, while RWDT supports four timeout actions: interrupt, HP CPU reset, HP core reset, and system reset (see details in Section 18.2.2.2 *Stages and Timeout Actions*). A timeout value can be set for each stage individually.

In SPI Boot mode, RWDT and the MWDT in timer group 0 are enabled automatically in order to detect errors that may occur during the flash boot process and facilitate recovery.

ESP32-P4 also has one analog watchdog timer: Super watchdog (SWD). It is an ultra-low-power circuit in analog domain that helps to prevent the system from operating in a sub-optimal state and resets the system if required.

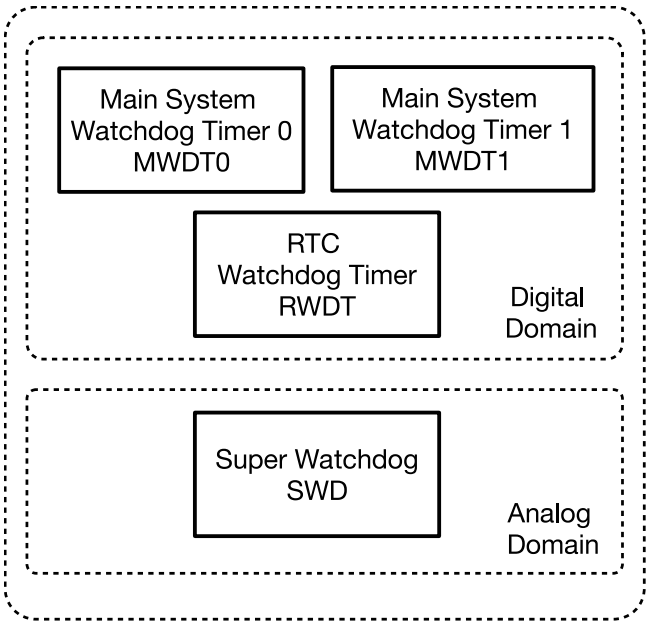


Figure 18.1-1. Watchdog Timers Overview

Note that while this chapter provides the functional descriptions of the watchdog timer's, MWDT register descriptions are detailed in Chapter 17 *Timer Group (TIMG)*, and the RWDT and SWD register descriptions are detailed in Section 18.5 *Register Summary*.

18.2 Digital Watchdog Timers

18.2.1 Features

Watchdog timers have the following features:

- Four stages, each with a separately programmable timeout value and timeout action
- Timeout actions:
  - MWDT: interrupt, HP CPU reset, HP core reset
  - RWDT: interrupt, HP CPU reset, HP core reset, system reset
- Flash boot protection under SPI Boot mode at stage 0:
  - MWDT0: HP core reset upon timeout
  - RWDT: system reset upon timeout
- Write protection that makes WDT register read only unless unlocked
- 32-bit timeout counter
- Clock source:
  - MWDT: PLL_F80M_CLK, RC_FAST_CLK or XTAL_CLK
  - RWDT: LP_DYN_SLOW_CLK

18.2.2 Functional Description

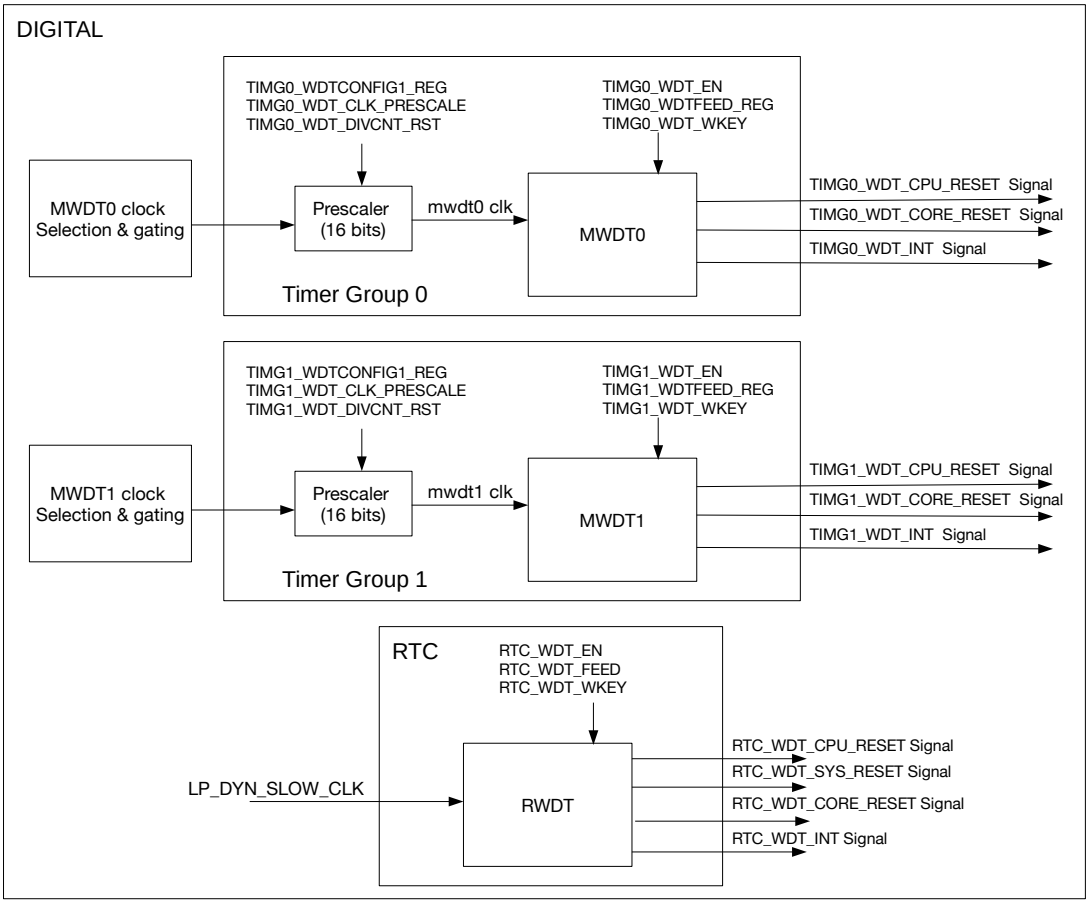


Figure 18.2-1. Digital Watchdog Timers in ESP32-P4

Figure 18.2-1 shows the three watchdog timers in ESP32-P4 digital systems.

18.2.2.1 Clock Source and 32-Bit Counter

At the core of each watchdog timer is a 32-bit counter.

Take MWDT0 as an example:

- MWDT0 can select between the PLL_F80M_CLK, RC_FAST_CLK or XTAL_CLK (external) clock as its clock source by setting the HP_SYS_TIMERGRP0_WDT_SRC_SEL field of the HP_SYS_CLKRST_PERI_CLK_CTRL20_REG register.
- The selected clock is switched on by setting HP_SYS_TIMERGRP0_WDT_CLK_EN field of the HP_SYS_CLKRST_PERI_CLK_CTRL20_REG register to 1 and switched off by setting it to 0. Then the selected clock is divided by a 16-bit configurable prescaler. See more details in Table 11.2-1 of Chapter 11.
- The 16-bit prescaler for MWDT is configured via the TIMG_WDT_CLK_PRESCALE field of TIMG_WDTCONFIG1_REG. When TIMG_WDT_DIVCNT_RST field is set, the prescaler is reset and it can be re-configured at once.

In contrast, the clock source of RWDT is derived directly from LP_DYN_SLOW_CLK (see details in Chapter 11

Reset and Clock).

MWDT and RWDT are enabled by setting the TIMG_WDT_EN and RTC_WDT_EN fields respectively. When enabled, the 32-bit counters of the watchdog will increment on each source clock cycle until the timeout value of the current stage is reached (i.e., timeout of the current stage). When this occurs, the current counter value is reset to zero and the next stage will become active. If a watchdog timer is fed by software, the timer will return to stage 0 and reset its counter value to zero. Software can feed a watchdog timer by writing any value to TIMG_WDTFEED_REG for MDWT and by writing 1 to RTC_WDT_FEED for RWDT.

18.2.2.2 Stages and Timeout Actions

Timer stages allow for a timer to have a series of different timeout values and corresponding timeout action. When one stage times out, the timeout action is triggered, the counter value is reset to zero, and the next stage becomes active.

MWDT/RWDT offers four stages (referred to as stages 0 to 3). The watchdog timers will progress through each stage in a loop (i.e., from stage 0 to 3, then back to stage 0).

Timeout values of each stage for MWDT are configured in TIMG_WDTCONFIG*i*_REG (where *i* ranges from 2 to 5), whilst timeout values for RWDT are configured using RTC_WDT_STG*j*_HOLD field (where *j* ranges from 0 to 3).

Please note that the timeout value of stage 0 for RWDT ($Thold_0$) is determined by the combination of the
EFUSE_WDT_DELAY_SEL field of eFuse register EFUSE_RD_REPEAT_DATA0_REG and RTC_WDT_STG0_HOLD field. The relationship is as follows:

$$T_{hold0} = RTC_WDT_STG0_HOLD << (EFUSE_WDT_DELAY_SEL + 1)$$

where $<<$ is a left-shift operator. For example, if RTC_WDT_STG0_HOLD is configured as 100 and EFUSE_WDT_DELAY_SEL is 1, the $Thold_0$ will be 400 cycles.

Upon the timeout of each stage, one of the following timeout actions will be executed:

Table 18.2-1. Timeout Actions

Timeout Action	Description
Interrupt	Trigger an interrupt
HP CPU reset	Reset the HP CPU0 and HP CPU1
HP core reset	HP core reset resets HP CPU0, HP CPI1, HP peripherals, HP GPIO, etc.
System reset	Reset the whole digital system, including the LP system.
Disabled	No effect on the system

For MWDT, the timeout action of all stages is configured in TIMG_WDTCONFIG0_REG. Likewise for RWDT, the timeout action is configured in RTC_WDT_CONFIG0_REG.

18.2.2.3 Write Protection

Watchdog timers are critical to detecting and handling erroneous system/software behavior, thus should not be disabled easily (e.g., due to a misplaced register write). Therefore, MWDT and RWDT incorporate a write

protection mechanism that prevent the watchdogs from being disabled or tampered with due to an accidental write.

The write protection mechanism is implemented using a write-key field for each timer (TIMG_WDT_WKEY for MWDT, RTC_WDT_WKEY for RWDT). The value 0x50D83AA1 must be written to the watchdog timer's write-key field before any other register of the same watchdog timer can be changed. Any attempts to write to a watchdog timer's registers (other than the write-key field itself) whilst the write-key field's value is not 0x50D83AA1 will be ignored. The recommended procedure for accessing a watchdog timer is as follows:

1. Disable the write protection by writing the value 0x50D83AA1 to the timer's write-key field.
2. Make the required modification of the watchdog such as feeding or changing its configuration.
3. Re-enable write protection by writing any value other than 0x50D83AA1 to the timer's write-key field.

18.2.2.4 Flash Boot Protection

Under the SPI boot mode, MWDT0 as well as RWDT, are automatically enabled. Stage 0 for the enabled MWDT0 is automatically configured as HP core reset action upon timeout, known as HP core reset. Likewise, stage 0 for RWDT is configured to system reset, which resets the main system and RTC when it times out. After SPI booting, TIMG_WDT_FLASHBOOT_MOD_EN and RTC_WDT_FLASHBOOT_MOD_EN should be cleared to stop the flash boot protection procedure for both MWDT0 and RWDT respectively. After this, MWDT0 and RWDT can be configured by software.

18.3 Super Watchdog

Super watchdog (SWD) is an ultra-low-power circuit in analog domain that helps to prevent the system from operating in a sub-optimal state and resets the system (system reset) if required. SWD contains a watchdog circuit that needs to be fed for at least once during its timeout period, which is slightly less than one second. About 100 ms before watchdog timeout, it will also send out a WD_INTR signal as a request to remind the system to feed the watchdog.

If the system doesn't respond to SWD feed request and watchdog finally times out, SWD will generate a system level signal SWD_RSTB to reset whole digital circuits on the chip (system reset) .

The source of the clock for SWD is constant and can not be selected.

18.3.1 Features

SWD has the following features:

- Ultra-low power
- Interrupt to indicate that the SWD is about to time out
- Various dedicated methods for software to feed SWD, which enables SWD to monitor the working state of the whole operating system

18.3.2 Super Watchdog Controller

18.3.2.1 Structure

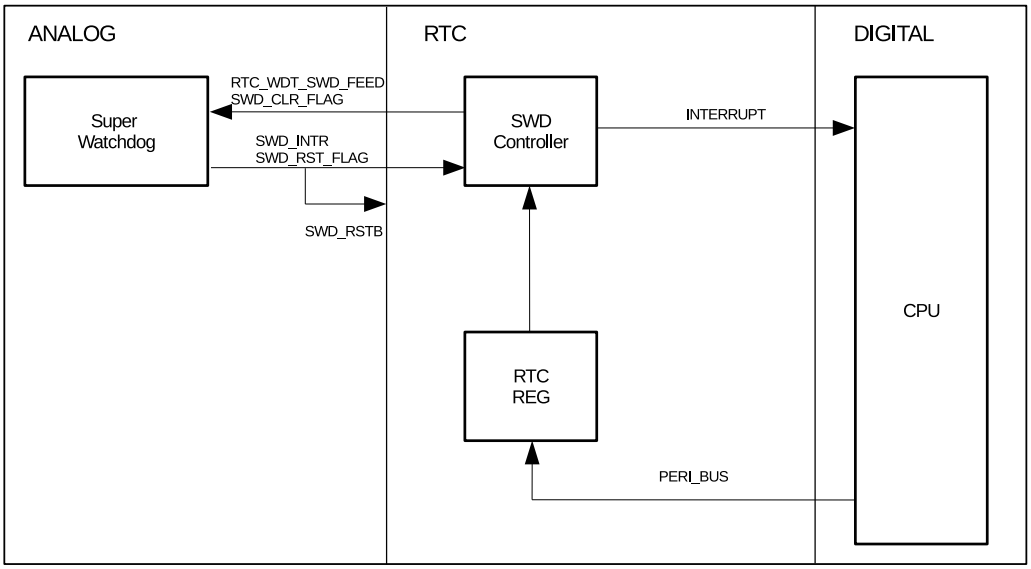


Figure 18.3-1. Super Watchdog Controller Structure

18.3.2.2 Workflow

In normal state:

- SWD controller receives feed request from SWD.
- SWD controller can send an interrupt to main CPU.
- Main CPU can feed SWD directly by setting RTC_WDT_SWD_FEED.
- When trying to feed SWD, CPU needs to disable SWD controller's write protection by writing 0x50D83AA1 to RTC_WDT_SWD_WKEY. This prevents SWD from being fed by mistake when the system is operating in sub-optimal state.
- If setting RTC_WDT_SWD_AUTO_FEED_EN to 1, SWD controller can also feed SWD itself without any interaction with CPU.

After reset:

- Check LP_AONCLKRST_LPCORE_RESET_CAUSE[4:0] for the cause of HP CPU reset.
  If LP_AONCLKRST_LPCORE_RESET_CAUSE[4:0] == 0x12, it indicates that the cause is SWD reset.
- Set RTC_WDT_SWD_RST_FLAG_CLR to clear the SWD reset flag.

18.4 Interrupts

For watchdog timer interrupts, please refer to Section 17.3.7 *Interrupts* in Chapter 17 *Timer Group (TIMG)*.

18.5 Register Summary

The addresses in this section are relative to RTC_WDT base address provided in Table 9.3-2 in Chapter 9 *System and Memory*.

The abbreviations given in Column **Access** are explained in Section *Access Types for Registers*.

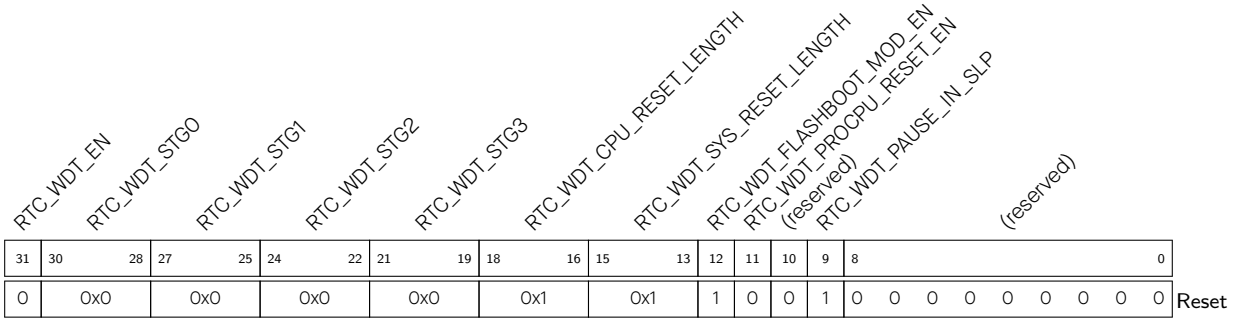
Name	Description	Address	Access
configuration register			
RTC_WDT_CONFIG0_REG	Configure the RWDT operation	0x0000	R/W
RTC_WDT_CONFIG1_REG	Configure the RWDT timeout time of stage0	0x0004	R/W
RTC_WDT_CONFIG2_REG	Configure the RWDT timeout time of stage1	0x0008	R/W
RTC_WDT_CONFIG3_REG	Configure the RWDT timeout time of stage2	0x000C	R/W
RTC_WDT_CONFIG4_REG	Configure the RWDT timeout time of stage3	0x0010	R/W
RTC_WDT_FEED_REG	Configure the feed function of RWDT	0x0014	WT
RTC_WDT_WPROTECT_REG	Configure the lock function of RWDT	0x0018	R/W
RTC_WDT_SWD_CONFIG_REG	Configure the SWD operation	0x001C	varies
RTC_WDT_SWD_WPROTECT_REG	Configure the lock function of SWD	0x0020	R/W
RTC_WDT_INT_RAW_REG	Configure whether to generate timeout interrupt	0x0024	R/WTC/SS
RTC_WDT_INT_ST_REG	The interrupt status register of WDT	0x0028	RO
RTC_WDT_INT_ENA_REG	The interrupt enable register of WDT	0x002C	R/W
RTC_WDT_INT_CLR_REG	The interrupt clear register of WDT	0x0030	WT
RTC_WDT_DATE_REG	Version control register	0x03FC	R/W

18.6 Registers

MWDT registers are part of the timer submodule and are described in Section 17.5 *Register Summary* in Chapter 17 *Timer Group (TIMG)*.

The addresses of RWDT and SWD registers in this section are relative to RTC_WDT base address provided in Table 9.3-2 in Chapter 9 *System and Memory*.

Register 18.1. RTC_WDT_CONFIG0_REG (0x0000)



RTC_WDT_PAUSE_IN_SLP Configure whether or not pause RWDT when chip is in sleep mode.

0: Enable

1: Disable

(R/W)

RTC_WDT_PROCPU_RESET_EN Configure whether or not to enable RWDT to reset CPU.

0: Disable

1: Enable

(R/W)

RTC_WDT_FLASHBOOT_MOD_EN Configure whether or not to enable RWDT when chip is in SPI boot mode.

0: Disable

1: Enable

(R/W)

RTC_WDT_SYS_RESET_LENGTH Configure the HP core reset time.

Measurement unit: LP_DYN_FAST_CLK

(R/W)

RTC_WDT_CPU_RESET_LENGTH Configure the HP CPU reset time.

Measurement unit: LP_DYN_FAST_CLK

(R/W)

RTC_WDT_STG3 Configure the timeout action of stage3.

0: No operation

1: Generate interrupt

2: Generate HP CPU reset

3: Generate HP core reset

4: Generate system reset

(R/W)

Continued on the next page...

Register 18.1. RTC_WDT_CONFIG0_REG (0x0000)

Continued from the previous page...

RTC_WDT_STG2 Configure the timeout action of stage2.

0: No operation

1: Generate interrupt

2: Generate HP CPU reset

3: Generate HP core reset

4: Generate system reset

(R/W)

RTC_WDT_STG1 Configure the timeout action of stage1.

0: No operation

1: Generate interrupt

2: Generate HP CPU reset

3: Generate HP core reset

4: Generate system reset

(R/W)

RTC_WDT_STG0 Configure the timeout action of stage0.

0: No operation

1: Generate interrupt

2: Generate HP CPU reset

3: Generate HP core reset

4: Generate system reset

(R/W)

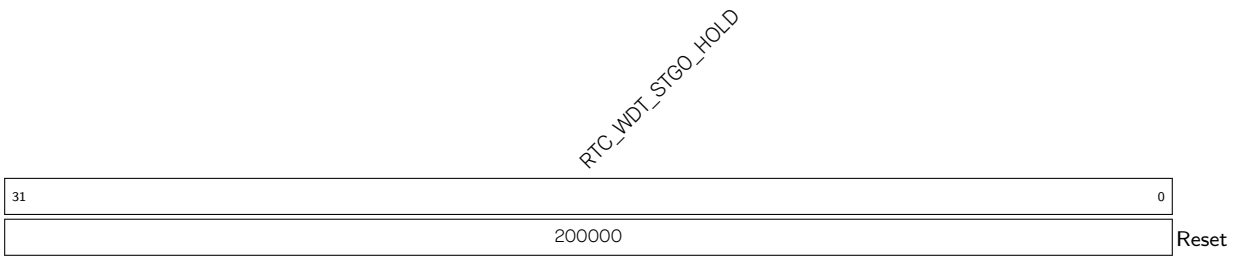
RTC_WDT_EN Configure whether or not enable RWDT.

0: Disable RWDT

1: Enable RWDT

(R/W)

Register 18.2. RTC_WDT_CONFIG1_REG (0x0004)

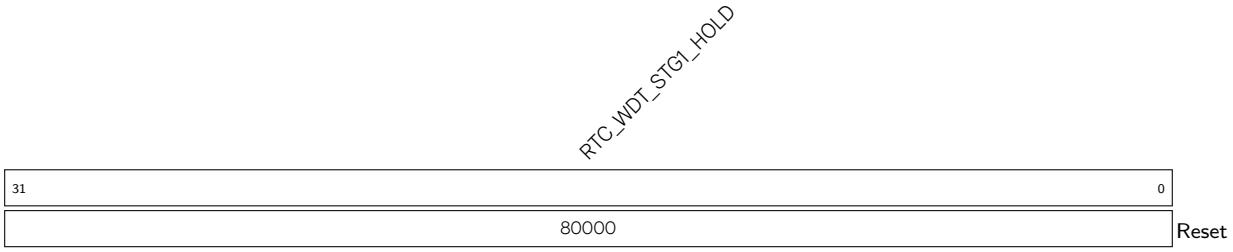


RTC_WDT_STG0_HOLD Configure the timeout time for stage0.

Measurement unit: LP_DYN_SLOW_CLK

(R/W)

Register 18.3. RTC_WDT_CONFIG2_REG (0x0008)

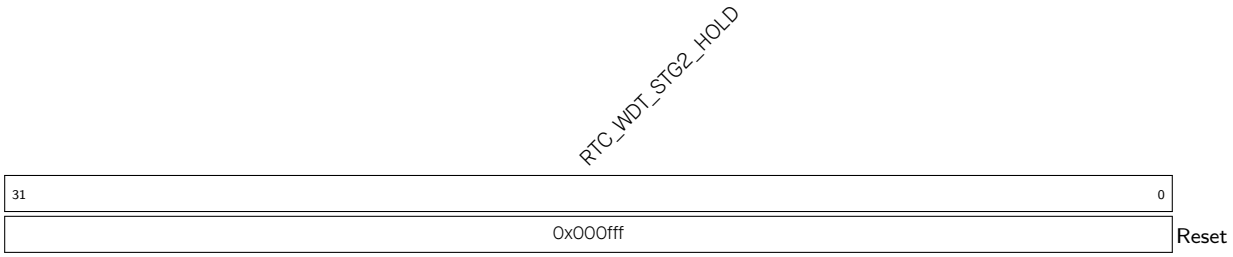


RTC_WDT_STG1_HOLD Configure the timeout time for stage1.

Measurement unit: LP_DYN_SLOW_CLK

(R/W)

Register 18.4. RTC_WDT_CONFIG3_REG (0x000C)

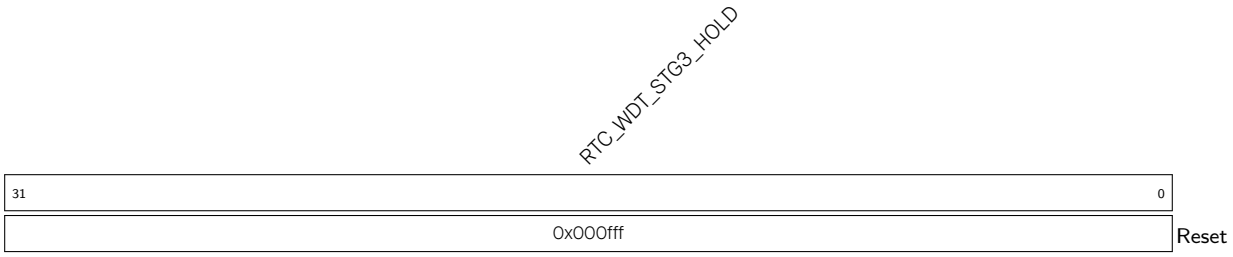


RTC_WDT_STG2_HOLD Configure the timeout time for stage2.

Measurement unit: LP_DYN_SLOW_CLK

(R/W)

Register 18.5. RTC_WDT_CONFIG4_REG (0x0010)

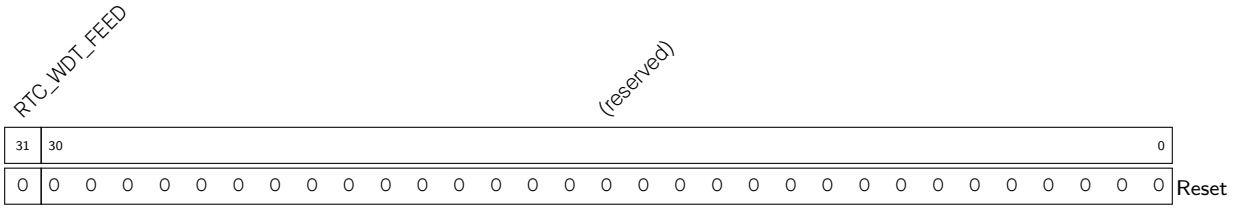


RTC_WDT_STG3_HOLD Configure the timeout time for stage3.

Measurement unit: LP_DYN_SLOW_CLK

(R/W)

Register 18.6. RTC_WDT_FEED_REG (0x0014)



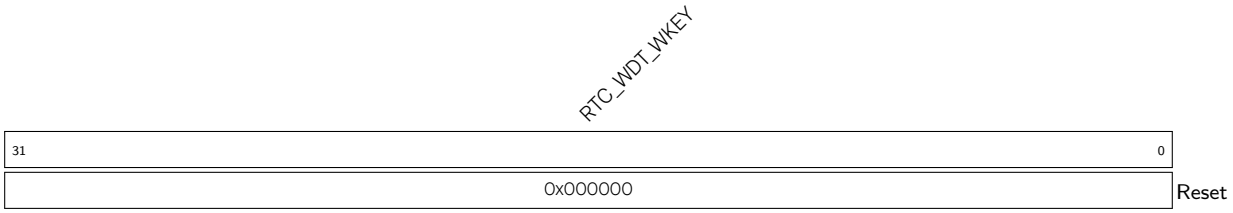
RTC_WDT_FEED Configure this bit to feed the RWDT.

0: Invalid

1: Feed RWDT

(WT)

Register 18.7. RTC_WDT_WPROTECT_REG (0x0018)



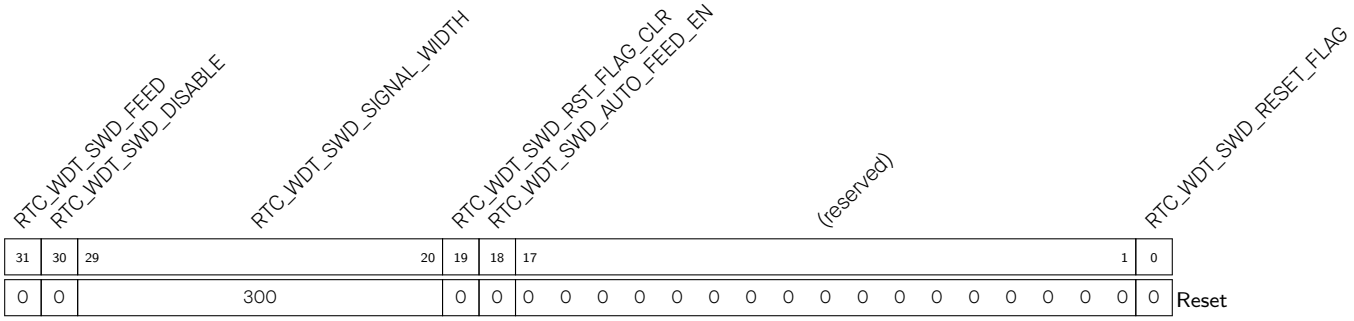
RTC_WDT_WKEY Configure this field to lock or unlock RWDT's configuration registers.

0x50D83AA1: unlock the RWDT configuration register

Others value: lock the RWDT configuration register which can't be modified by software.

(R/W)

Register 18.8. RTC_WDT_SWD_CONFIG_REG (0x001C)



RTC_WDT_SWD_RESET_FLAG Represents the SWD whether has generated the reset signal

0: No

1: Yes

(RO)

RTC_WDT_SWD_AUTO_FEED_EN Configure this bit to enable to feed SWD automatically by hardware.

0: Disable

1: Enable

(R/W)

RTC_WDT_SWD_RST_FLAG_CLR Configure this bit to clear SWD reset flag

0: Invalid

1: Clear the reset flag

(WT)

RTC_WDT_SWD_SIGNAL_WIDTH Configure the SWD signal length that output to analog circuit. Measurement unit: LP_DYN_FAST_CLK (R/W)

RTC_WDT_SWD_DISABLE Configure this bit to disable the SWD.

0: Enable the SWD

1: Disable the SWD

(R/W)

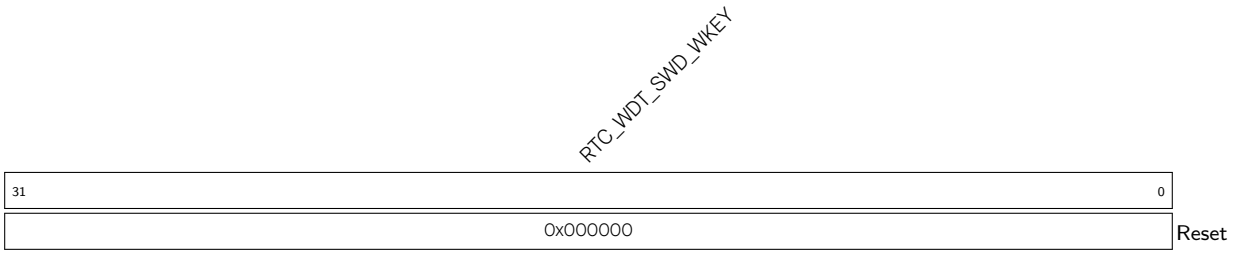
RTC_WDT_SWD_FEED Configure this bit to feed the SWD.

0: Invalid

1: Feed SWD

(WT)

Register 18.9. RTC_WDT_SWD_WPROTECT_REG (0x0020)



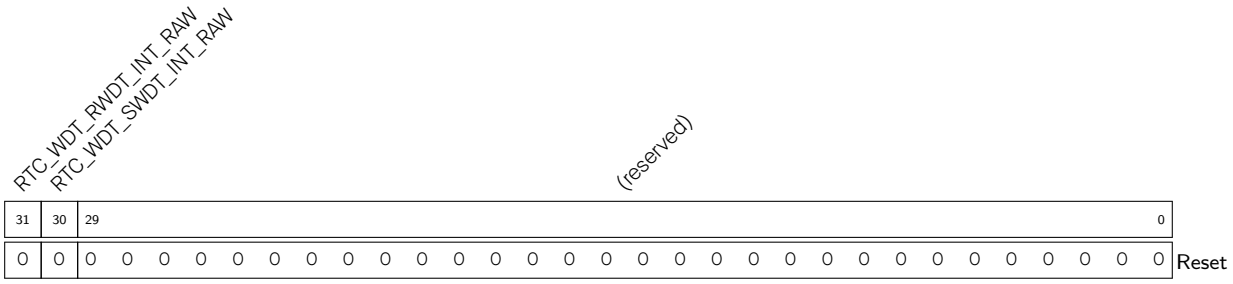
RTC_WDT_SWD_WKEY Configure this field to lock or unlock SWD's configuration registers.

0x50D83AA1: unlock the SWD configuration register.

Others value: lock the SWD configuration register which can't be modified by the software.

(R/W)

Register 18.10. RTC_WDT_INT_RAW_REG (0x0024)



RTC_WDT_SWDT_INT_RAW Represents the SWD whether or not has generated timeout interrupt.

0: No

1: Yes

(R/WTC/SS)

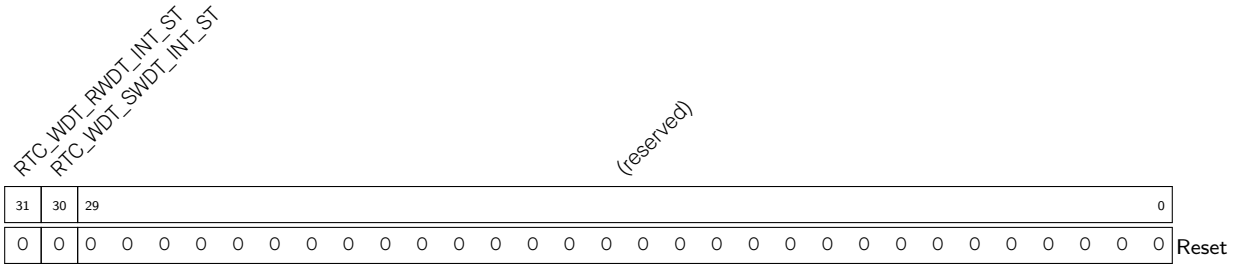
RTC_WDT_RWDT_INT_RAW Represents the RWDT whether or not generates timeout interrupt.

0: No

1: Yes

(R/WTC/SS)

Register 18.11. RTC_WDT_INT_ST_REG (0x0028)



RTC_WDT_SWDT_INT_ST Represents the SWD whether or not has generated and sent timeout interrupt to CPU.

0: No

1: Yes

(RO)

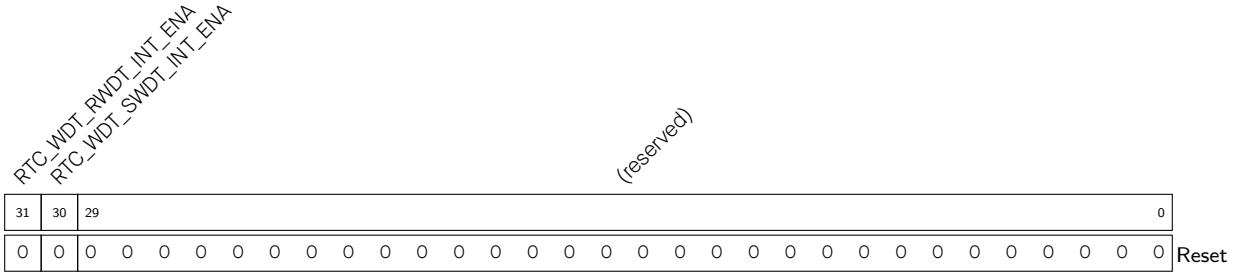
RTC_WDT_RWDT_INT_ST Represents the RWDT whether or not generates and sends timeout interrupt to CPU.

0: No

1: Yes

(RO)

Register 18.12. RTC_WDT_INT_ENA_REG (0x002C)



RTC_WDT_SWDT_INT_ENA Configure whether or not to enable the SWD to send timeout interrupt.

0: Disable

1: Enable

(R/W)

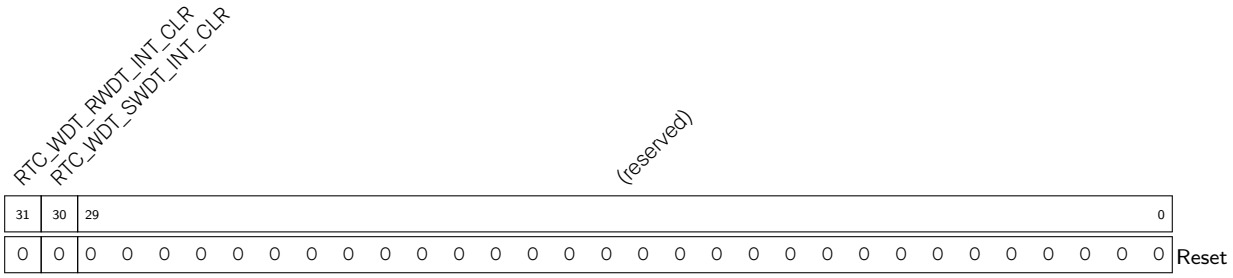
RTC_WDT_RWDT_INT_ENA Configure whether or not to enable the RWDT to send timeout interrupt.

0: Disable

1: Enable

(R/W)

Register 18.13. RTC_WDT_INT_CLR_REG (0x0030)



RTC_WDT_SWDT_INT_CLR Configure whether to clear the timeout interrupt signal sent by SWD to CPU.

0: No

1: Yes

(WT)

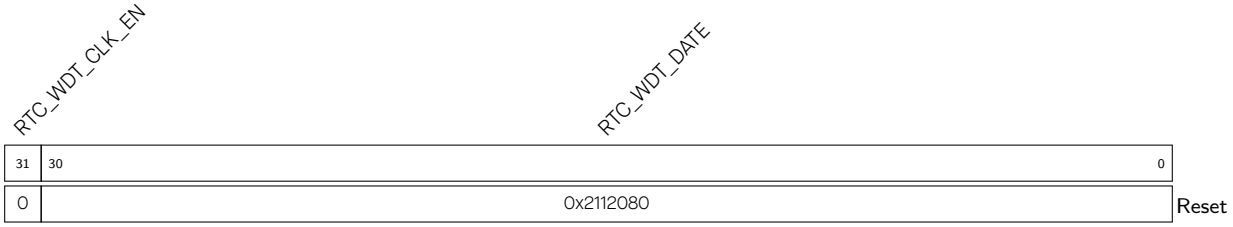
RTC_WDT_RWDT_INT_CLR Configure whether to clear the timeout interrupt signal sent by RWDT to CPU.

0: No

1: Yes

(WT)

Register 18.14. RTC_WDT_DATE_REG (0x03FC)



RTC_WDT_DATE Version control register. (R/W)

RTC_WDT_CLK_EN Reserved. (R/W)

Chapter 19

RTC Timer

19.1 Introduction

RTC Timer is an important module for implementing low power management of ESP32-P4. Based on a 48-bit readable counter, RTC Timer is mainly used as a system timer in low power mode when the timer peripheral in the HP system is unavailable. It also allows for configuring timer interrupts and logging the time when specific events happen in the system. The supported events are described in the 19.3 Functional Description.

19.2 Feature List

RTC Timer has the following features:

- 48-bit counter
- Time logging when one of the following events happens:
  - HP system reset
  - CPU enters stall state
  - CPU exits stall state
  - Crystal powers up
  - Crystal powers down
- Time logging through register configuration
- Occurrence time cached of the most recent two specific events
- Generation of interrupts at target times, which are configurable. It is also possible to configure two target times simultaneously.
- Uninterrupted operation during any reset or sleep mode, except for power-on reset of LP system.

19.3 Functional Description

- 48-bit counter
  - The implementation of the RTC Timer is based on a 48-bit counter, driven by the RC_SLOW_CLK in the AlwaysOn power domain. It uses continuous loop counting (except during LP system reset), and an overflow interrupt is generated when the 48-bit counter overflows.
- Log the occurrence time of specific events
  - RTC Timer supports logging the occurrence time of three types of events:

    * HP system reset
    * CPU enters or exits stall state
    * Crystal powers up or down
  - The above three types of events have their own independent configuration registers. When the configuration registers are enabled, if the corresponding events occur, the hardware pulse generated will trigger the RTC Timer to log the current time. The corresponding configuration registers are shown in the table below.

Table 19.3-1. Configure RTC Timer to Log the Occurrence Time of Specific Events

Configuration Register	Description
RTC_TIMER_MAIN_TIMER_SYS_RST	Enable the RTC Timer to log the time of system reset.
RTC_TIMER_MAIN_TIMER_SYS_STALL	Enable the RTC Timer to log the time when the CPU enters or exits the stall state.
RTC_TIMER_MAIN_TIMER_XTAL_OFF	Enable the RTC Timer to log the time when the crystal powers up or down.

- Log the current time through software configuration.
  - In addition to the above three specific events that can trigger the RTC Timer to log the time, the RTC Timer can also record the current time through the configuration register RTC_TIMER_MAIN_TIMER_UPDATE.
  - A pulse signal will be generated by configuring the RTC_TIMER_MAIN_TIMER_UPDATE, which will trigger the RTC Timer to record the current time.
- Generate counting interrupts at target times
  - RTC Timer supports configuring two target times simultaneously, referred to as target time 0 and target time 1. When the timer reaches target time 0, it will trigger the RTC_TIMER_CMP0_INT interrupt. Similarly, when the timer reaches Target Moment 1, it will trigger the RTC_TIMER_CMP1_INT interrupt.
  - Configure the target times through the registers below:

Table 19.3-2. Target times Configuration

Configuration Register	Description
RTC_TIMER_MAIN_TIMER_TAR_EN*n*	*n* = 0 or 1, enable target time configuration.
RTC_TIMER_MAIN_TIMER_TAR_LOW*n*	*n* = 0 or 1, configure the low 32 bits of the target time.
RTC_TIMER_MAIN_TIMER_TAR_HIGH*n*	*n* = 0 or 1, configure the high 16 bits of the target time.

- Read the time cached in the RTC Timer
  - There are two sets of registers used to cache the occurrence time of specific events, namely register group 0 and register group 1.
  - Register group 0 includes register RTC_TIMER_MAIN_BUF0_LOW_REG and RTC_TIMER_MAIN_BUF1_HIGH_REG which are used to cache the count value of the RTC Timer under the current trigger.

- Register group 1 includes register RTC_TIMER_MAIN_BUF1_LOW_REG and RTC_TIMER_MAIN_BUF1_HIGH_REG, which are used to cache the count value of the RTC Timer from the last trigger.
- On a new trigger, the record from the previous trigger will be moved from register group 0 to register group 1 (and the previous record in register group 1 will be overwritten), and the record of this trigger will be stored in register group 0. Therefore, only the last two triggers can be logged at any time.

19.4 Event Task Matrix Feature

The RTC Timer on ESP32-P4 supports the Event Task Matrix (ETM) function, which allows RTC Timer's ETM tasks to be triggered by any peripherals' ETM events, or RTC Timer's ETM events to trigger any peripherals' ETM tasks. This section introduces the ETM tasks and events related to RTC Timer. For more information, please refer to Chapter 14 Event Task Matrix (ETM).

RTC Timer doesn't receive any ETM tasks:

RTC Timer can generate the following ETM events:

- RTC_EVT_CMP: Indicate that the RTC Timer reaches the target time 0 configured by RTC_TIMER_MAIN_TIMER_TAR_LOW0 and RTC_TIMER_MAIN_TIMER_TAR_HIGH0.
- RTC_EVT_OVF: Indicate the 48-bit counter overflow.
- RTC_EVT_TICK: Indicates the event that occurs when the RTC timer increments by 1.

19.5 Interrupts

ESP32-P4's RTC Timer can generate the following interrupt signal(s) that will be sent to the Interrupt Matrix.

- LP_TIMER_REG_0_INTR
- LP_TIMER_REG_1_INTR

There are several internal interrupt sources from RTC Timer that can generate the above interrupt signal(s). The interrupt sources from RTC Timer are listed with their trigger conditions and the resulted interrupt signal(s) in Table 19.5-1.

Table 19.5-1. RTC Timer's Internal Interrupt Sources

Internal Interrupt Source	Trigger Condition	Interrupt Signal
RTC_TIMER_CMP0_INT	RTC Timer reaches the target time 0	LP_TIMER_REG_0_INTR
RTC_TIMER_CMP1_INT	RTC Timer reaches the target time 1	LP_TIMER_REG_1_INTR
RTC_TIMER_OVERFLOW_INT	48-bit counter overflow	LP_TIMER_REG_0_INTR LP_TIMER_REG_1_INTR

Note:
For definitions of *interrupt*, *interrupt signal*, *interrupt source*, and their correlations, please refer to Chapter 13 *Interrupt Matrix* > Section 13.2 *Interrupt Terminology in ESP32-P4*.

Each interrupt source can be configured by a common set of registers that are described in Section *Interrupt Configuration Registers*. The specific registers can be found in Section 19.6 *Register Summary*.

19.6 Register Summary

The addresses in this section are relative to RTC Timer base address provided in Table 9.3-2 in Chapter 9 *System and Memory*.

The abbreviations given in Column **Access** are explained in Section *Access Types for Registers*.

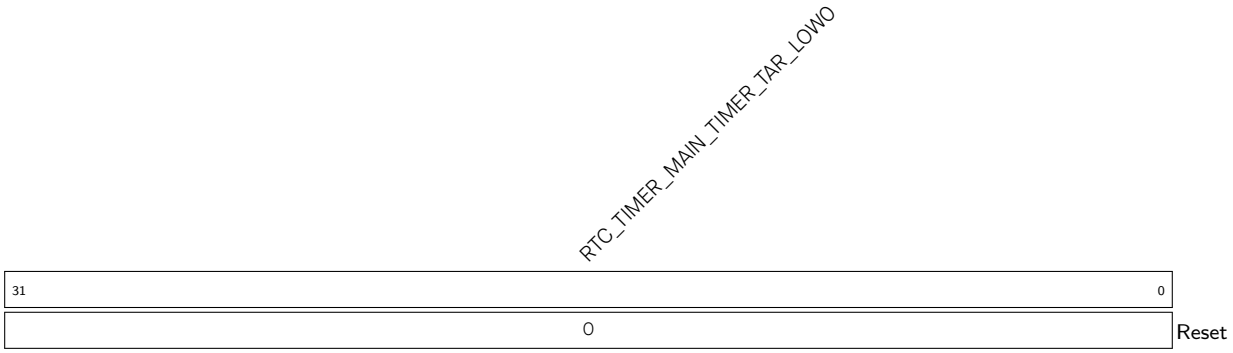
Name	Description	Address	Access
Configuration Registers			
RTC_TIMER_TAR0_LOW_REG	Configure the low 32 bits of the target time 0	0x0000	R/W
RTC_TIMER_TAR0_HIGH_REG	Configure the high 16 bits of the target time 0	0x0004	varies
RTC_TIMER_TAR1_LOW_REG	Configure the low 32 bits of the target time 1	0x0008	R/W
RTC_TIMER_TAR1_HIGH_REG	Configure the high 16 bits of the target time 1	0x000C	varies
RTC_TIMER_UPDATE_REG	Configure to enable the RTC Timer to latch current value	0x0010	varies
RTC_TIMER_MAIN_BUF0_LOW_REG	The low 32 bits of the cached value 0 in RTC Timer	0x0014	RO
RTC_TIMER_MAIN_BUF0_HIGH_REG	The high 16 bits of the cached value 0 in RTC Timer	0x0018	RO
RTC_TIMER_MAIN_BUF1_LOW_REG	The low 32 bits of the cached value 1 in RTC Timer	0x001C	RO
RTC_TIMER_MAIN_BUF1_HIGH_REG	The high 16 bits of the cached value 1 in RTC Timer	0x0020	RO
Interrupt Registers			
RTC_TIMER_INT_RAW_REG	The interrupt raw status register for the RTC Timer reaching the target time 0	0x0028	RO
RTC_TIMER_INT_ST_REG	The interrupt status register for the RTC Timer reaching the target time 0	0x002C	RO
RTC_TIMER_INT_ENA_REG	The interrupt enable register for the RTC Timer reaching the target time 0	0x0030	R/W
RTC_TIMER_INT_CLR_REG	The interrupt clear register for the RTC Timer reaching the target time 0	0x0034	WO
RTC_TIMER_LP_INT_RAW_REG	The interrupt raw status register for RTC Timer reaching the target time 1	0x0038	RO
RTC_TIMER_LP_INT_ST_REG	The interrupt status register for the RTC Timer reaching the target time 1	0x003C	RO
RTC_TIMER_LP_INT_ENA_REG	The interrupt enable register for the RTC Timer reaching the target time 1	0x0040	R/W
RTC_TIMER_LP_INT_CLR_REG	The interrupt clear register for the RTC Timer reaching the target time 1	0x0044	WO
Version Register			
RTC_TIMER_DATE_REG	Version control register	0x03FC	R/W

19.7 Registers

The addresses in this section are relative to RTC Timer base address provided in Table 9.3-2 in Chapter 9 *System and Memory*.

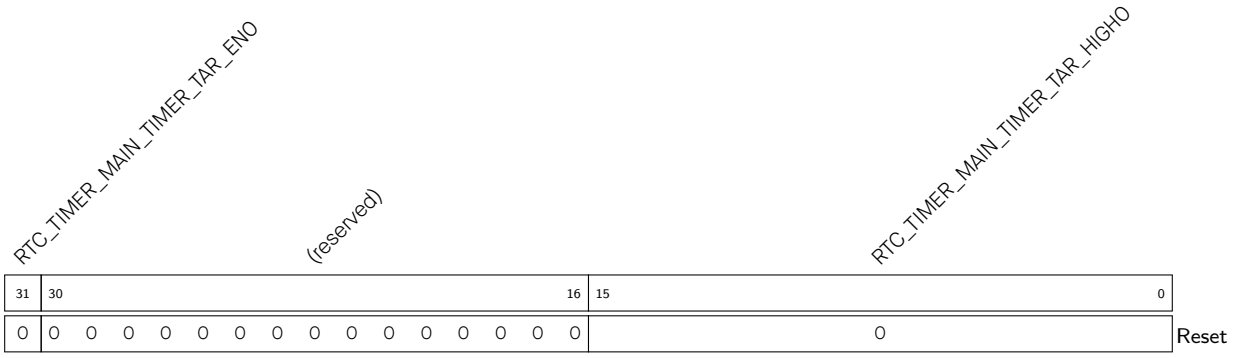
For how to program reserved fields, please refer to Section Programming Reserved Register Field.

Register 19.1. RTC_TIMER_TAR0_LOW_REG (0x0000)



RTC_TIMER_MAIN_TIMER_TAR_LOW0 Configure the low 32 bits of the target time 0 of the RTC Timer. (R/W)

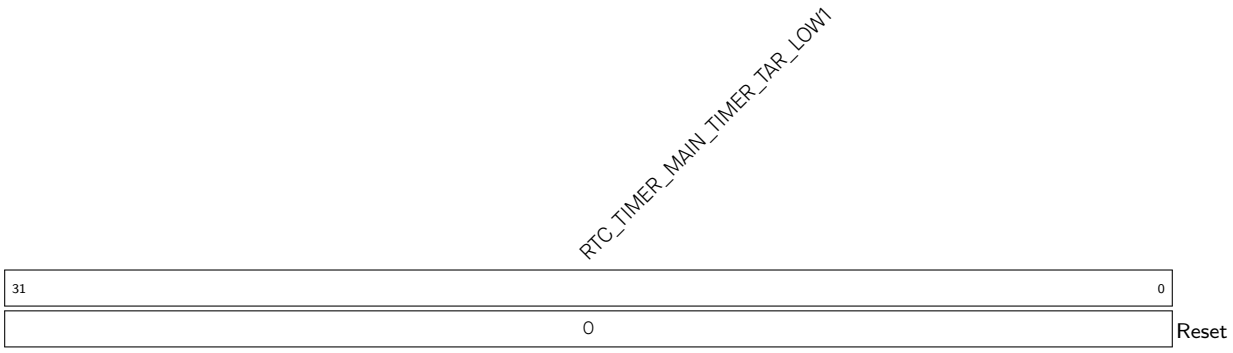
Register 19.2. RTC_TIMER_TAR0_HIGH_REG (0x0004)



RTC_TIMER_MAIN_TIMER_TAR_HIGH0 Configure the high 16 bits of the target time 0. (R/W)

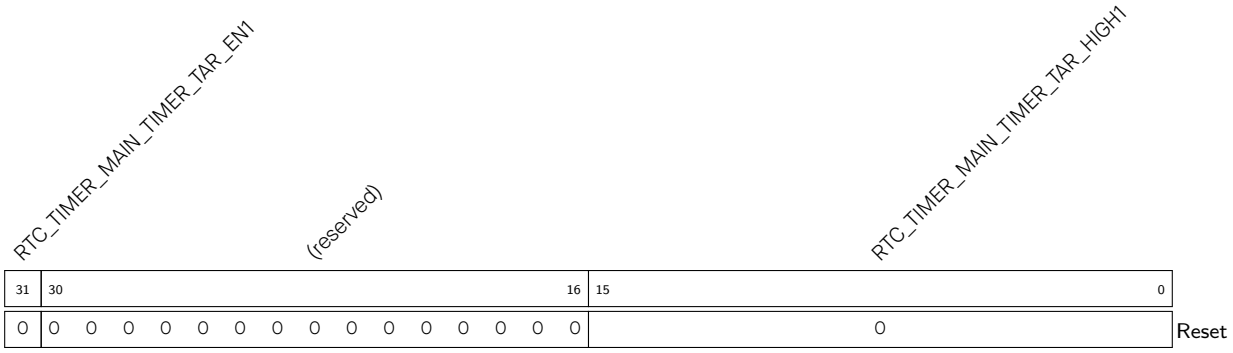
RTC_TIMER_MAIN_TIMER_TAR_EN0 Configure to enable the target time 0 of the RTC Timer. (WO)

Register 19.3. RTC_TIMER_TAR1_LOW_REG (0x0008)



RTC_TIMER_MAIN_TIMER_TAR_LOW1 Configure the low 32 bits of the target time 1. (R/W)

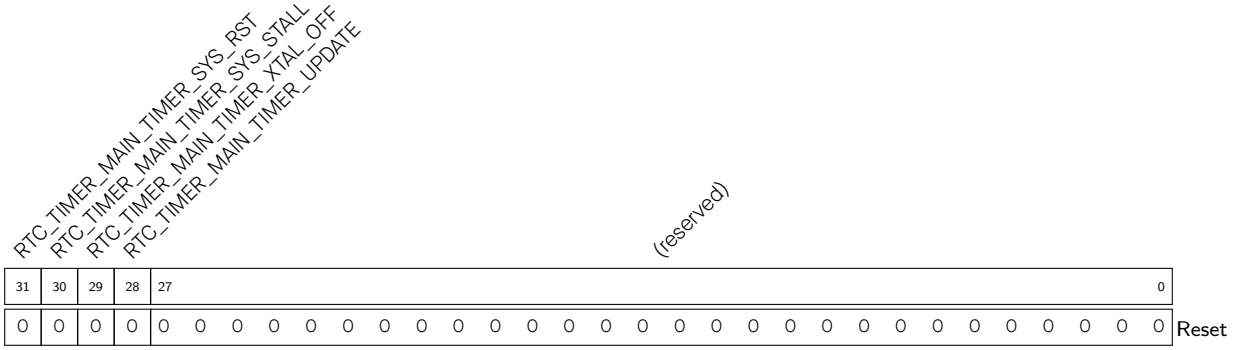
Register 19.4. RTC_TIMER_TAR1_HIGH_REG (0x000C)



RTC_TIMER_MAIN_TIMER_TAR_HIGH1 Configure the high 16 bits of the target time 1. (R/W)

RTC_TIMER_MAIN_TIMER_TAR_EN1 Configure to enable the target time 1 of the RTC Timer. (WO)

Register 19.5. RTC_TIMER_UPDATE_REG (0x0010)



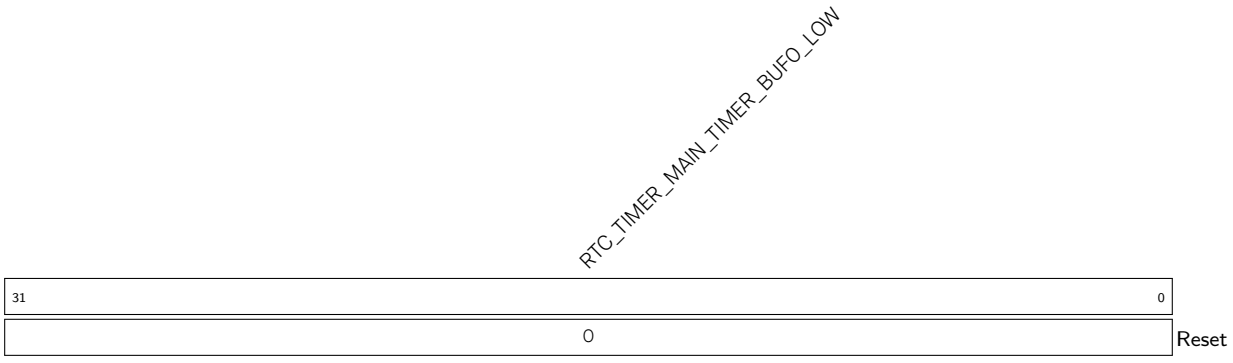
RTC_TIMER_MAIN_TIMER_UPDATE Configure to log the current time through software configuration. (WO)

RTC_TIMER_MAIN_TIMER_XTAL_OFF Configure to enable RTC Timer to log the time when the crystal powers up or down. (R/W)

RTC_TIMER_MAIN_TIMER_SYS_STALL Configure to enable the RTC Timer to log the time when the CPU enters or exits the stall state. (R/W)

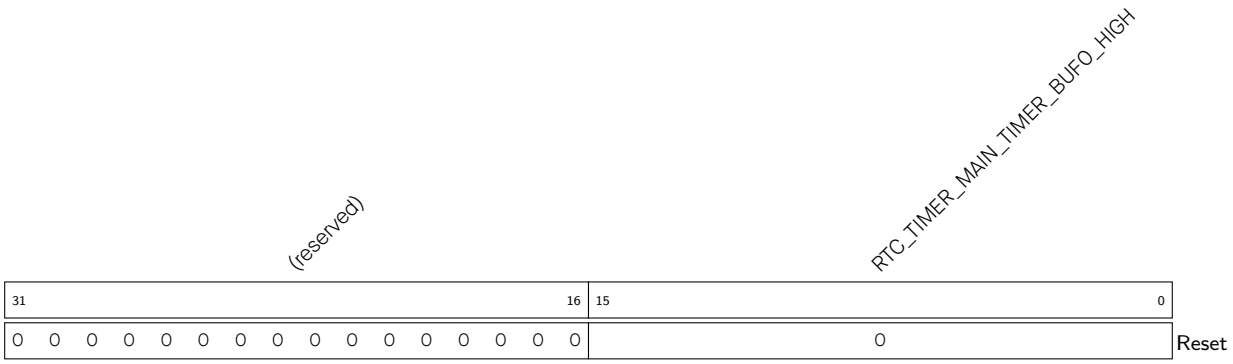
RTC_TIMER_MAIN_TIMER_SYS_RST Configure to enable RTC Timer to log the time of system reset. (R/W)

Register 19.6. RTC_TIMER_MAIN_BUF0_LOW_REG (0x0014)



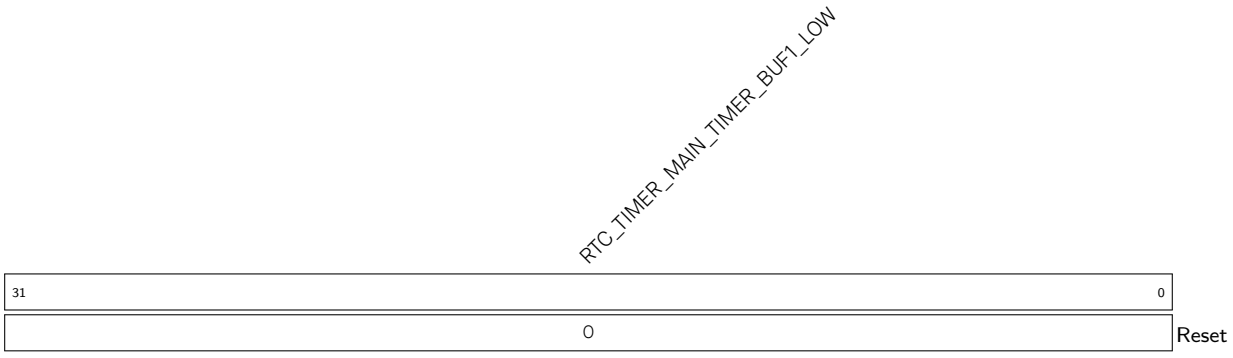
RTC_TIMER_MAIN_TIMER_BUF0_LOW Represent the low 32 bits of the cached value 0 in RTC Timer. (RO)

Register 19.7. RTC_TIMER_MAIN_BUF0_HIGH_REG (0x0018)



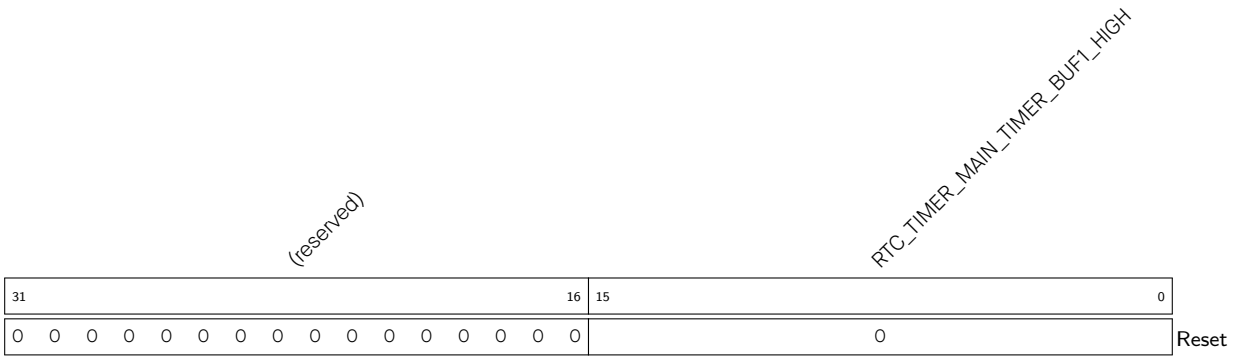
RTC_TIMER_MAIN_TIMER_BUF0_HIGH Represent the high 16 bits of the cached value 0 in RTC Timer. (RO)

Register 19.8. RTC_TIMER_MAIN_BUF1_LOW_REG (0x001C)



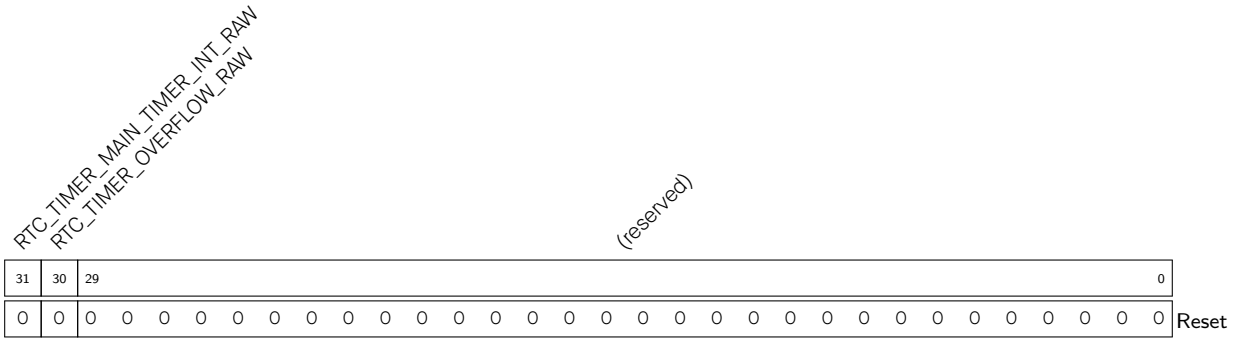
RTC_TIMER_MAIN_TIMER_BUF1_LOW Represent the low 32 bits of the cached value 1 in RTC Timer. (RO)

Register 19.9. RTC_TIMER_MAIN_BUF1_HIGH_REG (0x0020)



RTC_TIMER_MAIN_TIMER_BUF1_HIGH Represent the high 16 bits of the cached value 1 in RTC Timer. (RO)

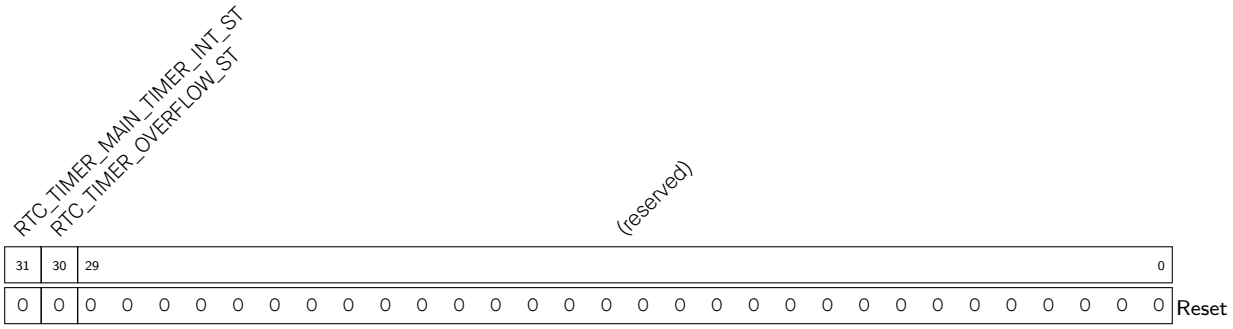
Register 19.10. RTC_TIMER_INT_RAW_REG (0x0028)



RTC_TIMER_OVERFLOW_RAW The interrupt raw status for the 48-bit counter overflow. (RO)

RTC_TIMER_MAIN_TIMER_INT_RAW The interrupt raw status for the RTC Timer reaching the target time 0. (RO)

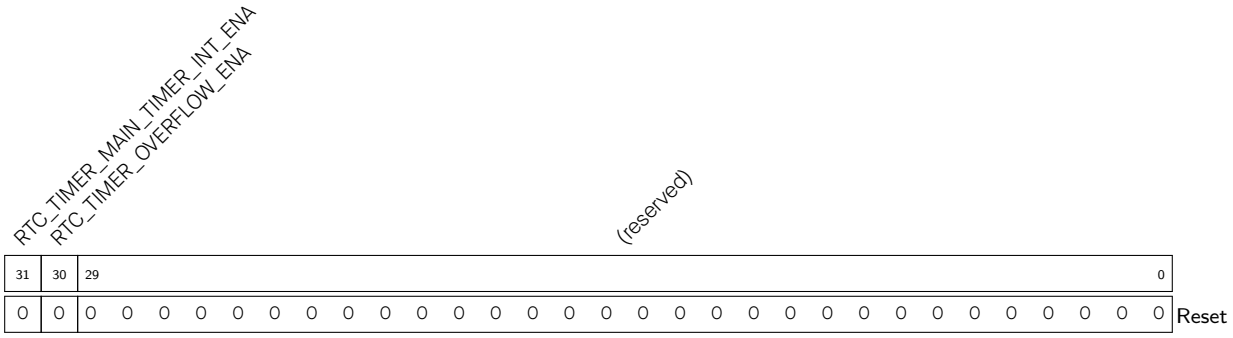
Register 19.11. RTC_TIMER_INT_ST_REG (0x002C)



RTC_TIMER_OVERFLOW_ST The interrupt status for the 48-bit counter overflow. (RO)

RTC_TIMER_MAIN_TIMER_INT_ST The interrupt status for the RTC Timer reaching the target time 0. (RO)

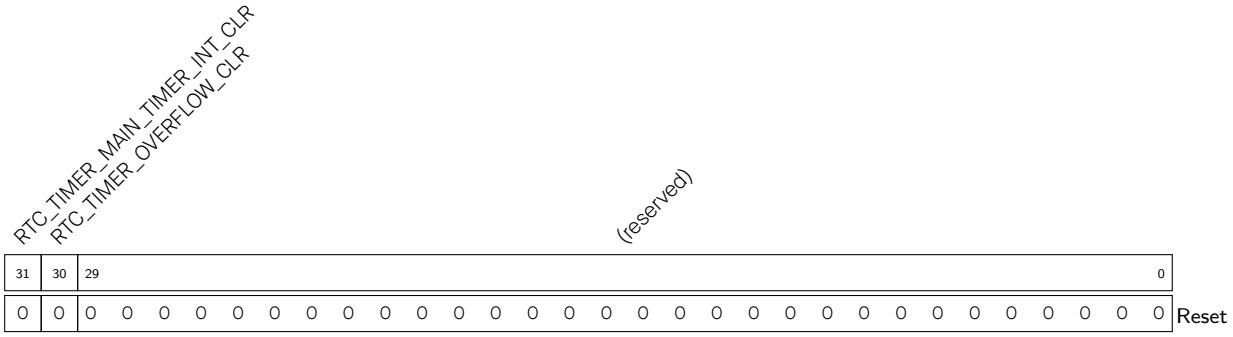
Register 19.12. RTC_TIMER_INT_ENA_REG (0x0030)



RTC_TIMER_OVERFLOW_ENA Configure to enable the interrupt for 48-bit counter overflow. (R/W)

RTC_TIMER_MAIN_TIMER_INT_ENA Configure to enable the interrupt for the RTC Timer reaching the target time 0. (R/W)

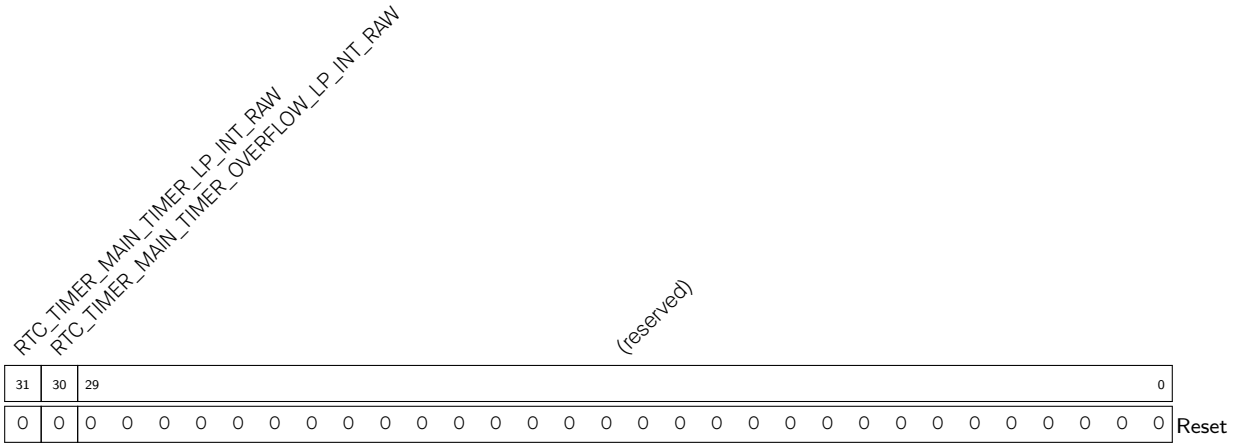
Register 19.13. RTC_TIMER_INT_CLR_REG (0x0034)



RTC_TIMER_OVERFLOW_CLR Configure to clear the interrupt for 48-bit counter overflow. (WO)

RTC_TIMER_MAIN_TIMER_INT_CLR Configure to clear the interrupt for the RTC Timer reaching the target time 0. (WO)

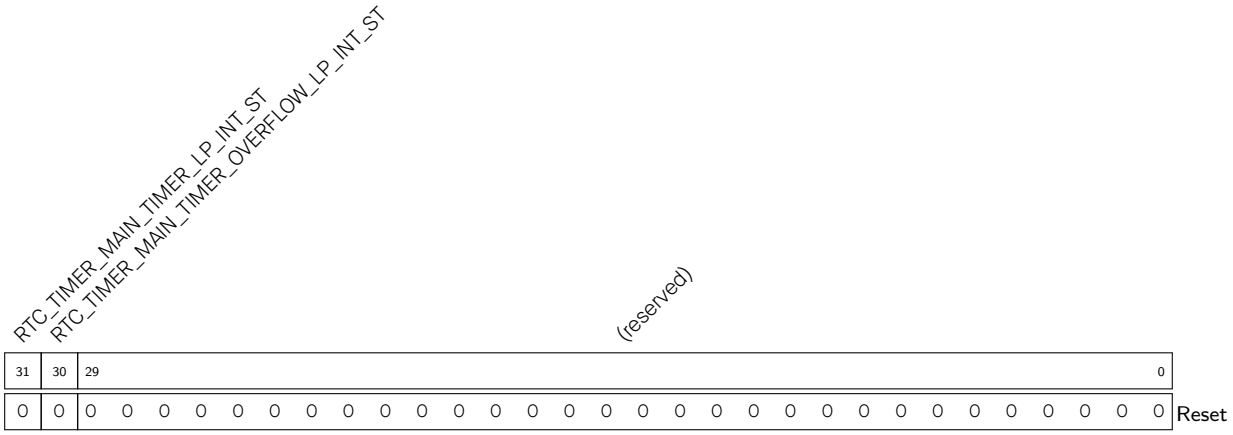
Register 19.14. RTC_TIMER_LP_INT_RAW_REG (0x0038)



RTC_TIMER_MAIN_TIMER_OVERFLOW_LP_INT_RAW The interrupt raw status for the 48-bit counter overflow. (RO)

RTC_TIMER_MAIN_TIMER_LP_INT_RAW The interrupt raw status for the RTC Timer reaching the target time 1.(RO)

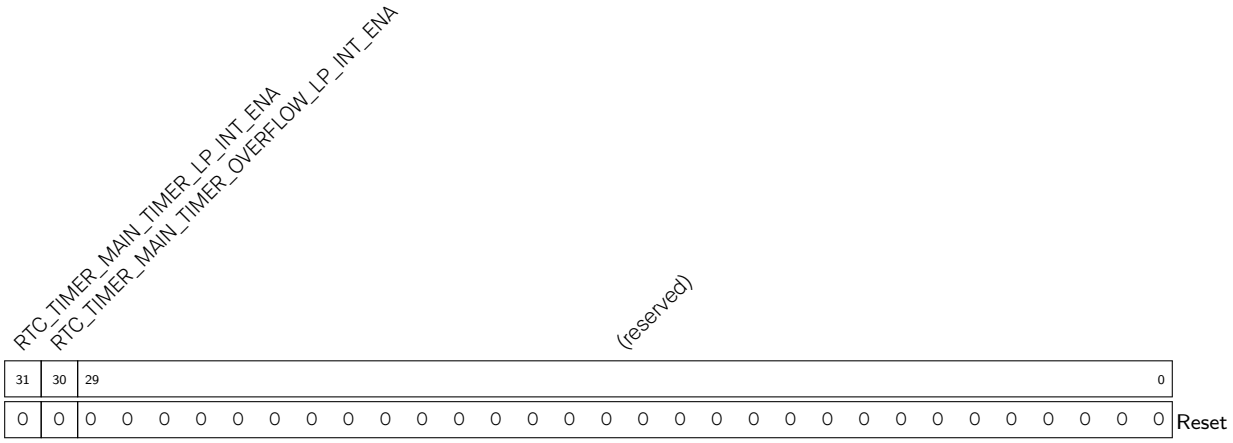
Register 19.15. RTC_TIMER_LP_INT_ST_REG (0x003C)



RTC_TIMER_MAIN_TIMER_OVERFLOW_LP_INT_ST The interrupt status for the 48-bit counter overflow. (RO)

RTC_TIMER_MAIN_TIMER_LP_INT_ST The interrupt status for the RTC Timer reaching the target time 1.(RO)

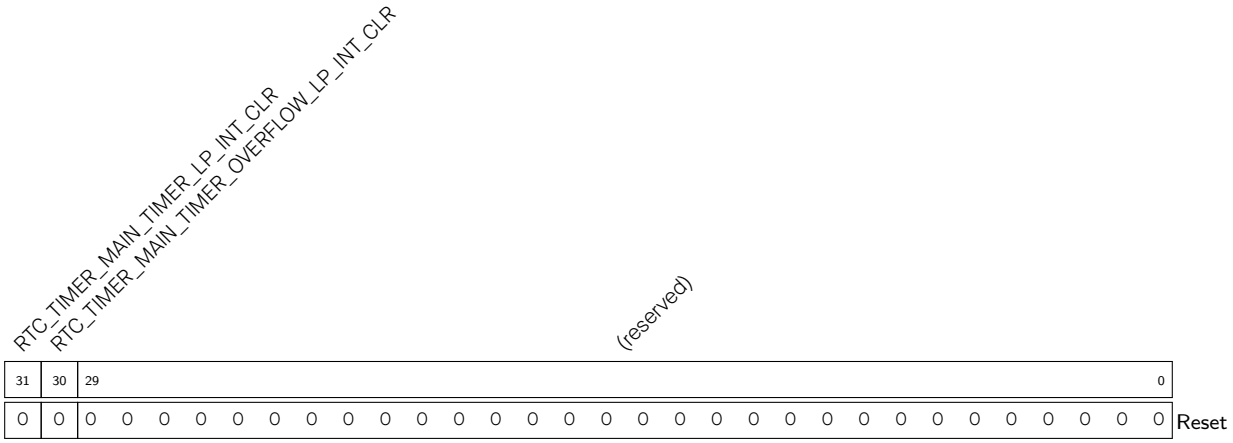
Register 19.16. RTC_TIMER_LP_INT_ENA_REG (0x0040)



RTC_TIMER_MAIN_TIMER_OVERFLOW_LP_INT_ENA Configure to enable the interrupt for the 48-bit counter overflow. (R/W)

RTC_TIMER_MAIN_TIMER_LP_INT_ENA Configure to enable the interrupt for the RTC Timer reaching the target time 1. (R/W)

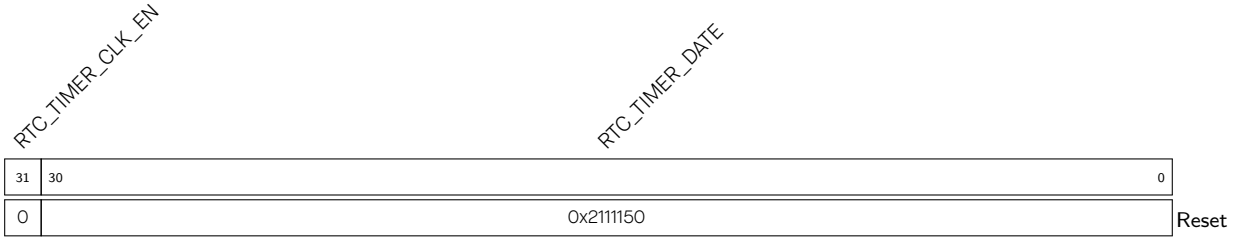
Register 19.17. RTC_TIMER_LP_INT_CLR_REG (0x0044)



RTC_TIMER_MAIN_TIMER_OVERFLOW_LP_INT_CLR Configure to clear the interrupt for 48-bit counter overflow. (WO)

RTC_TIMER_MAIN_TIMER_LP_INT_CLR Configure to clear the interrupt for the RTC Timer reaching the target time 1. (WO)

Register 19.18. RTC_TIMER_DATE_REG (0x03FC)



RTC_TIMER_DATE Version control register. (R/W)

RTC_TIMER_CLK_EN Configure to enable the register read/write clock. (R/W)

Chapter 20

Permission Control (PMS)

20.1 Overview

The permission control of ESP32-P4 consists of two parts: PMP (Physical Memory Protection) and APM (Access Permission Management).

PMP is located inside the HP CPUs and can control the HP CPUs' access to all address spaces. APM is an access permission management module located at the bus port, which can manage the permissions for the CPU (HP CPU0/1, LP CPU) to access the registers of all on-chip peripherals (see HP CPU PERI, HP PERI, and LP PERI in Table 20.1-1) and part of internal and external memory. The APM can also manage DMA masters like VDMA, USB OTG 2.0, to access parts of internal and external memory.

The APM module compares the configurable address range or the fixed address range, the access permissions of each address range, and the information carried on the bus like master ID, security mode, access address, access type, etc, to determine whether the access is allowed. With APM, users can precisely control the access of all masters to the memory and peripheral registers.

The distribution of management areas by PMP and APM is shown in Table 20.1-1. For HP CPU0/1, the control relationship between PMP and APM is shown in Figure 20.1-1.

Table 20.1-1. Management Areas by PMP and APM

Master / Slave	HP CPU0	HP CPU1	LP CPU	DMA Masters[1]
HP ROM[2]	PMP	PMP	HP APM	DMA APM
HP SPM	PMP	PMP	N/A	N/A
HP L2MEM[2]	PMP	PMP	HP APM	DMA APM
EXT MEM[2]	PMP	PMP	HP APM	DMA APM
Direct Access HP ROM[3]	PMP + HP APM	PMP + HP APM	N/A	N/A
Direct Access HP L2MEM[3]	PMP + HP APM	PMP + HP APM	N/A	N/A
Direct Access EXT MEM[3]	PMP + HP APM	PMP + HP APM	N/A	N/A
HP CPU PERI[4]	PMP + HP APM	PMP + HP APM	HP APM	N/A
HP PERI[5]	PMP + HP APM	PMP + HP APM	HP APM	N/A
LP PERI[6]	PMP + LP APM	PMP + LP APM	LP APM	N/A
LP ROM	N/A	N/A	N/A	N/A
LP SRAM	PMP + LP APM	PMP + LP APM	N/A	N/A

[1] DMA masters that support DMA transfers, such as VDMA, USB 2.0 OTG, MEM_MONITOR, etc.

[2]
- HP ROM (0x4FC0_0000 - 0x4FC1_FFFF)
- HP L2MEM (0x4FF0_0000 - 0x4FFB_FFFF)
- EXT MEM, including external flash (0x4000_0000 - 0x43FF_FFFF) and external RAM (0x4800_0000 - 0x4BFF_FFFF)

[3] "Direct access" means HP CPU0/1 accesses internal and external memory mapped into the following address spaces without going through the cache:
- HP ROM (0x8FC0_0000 - 0x8FC1_FFFF)
- HP L2MEM (0x8FF0_0000 - 0x8FFB_FFFF)
- EXT MEM, including external flash (0x8000_0000 - 0x83FF_FFFF) and external RAM (0x8800_0000 - 0x8BFF_FFFF)

[4] HP CPU peripheral registers (0x3FF0_0000 - 0x3FF1_FFFF)

[5] HP system peripheral registers (0x5000_0000 - 0x500F_FFFF)

[6] LP system peripheral registers (0x5011_0000 - 0x5012_FFFF)

[7] As can be seen from the table, APM includes HP APM, LP APM, and DMA APM, which will be described in the following sections.

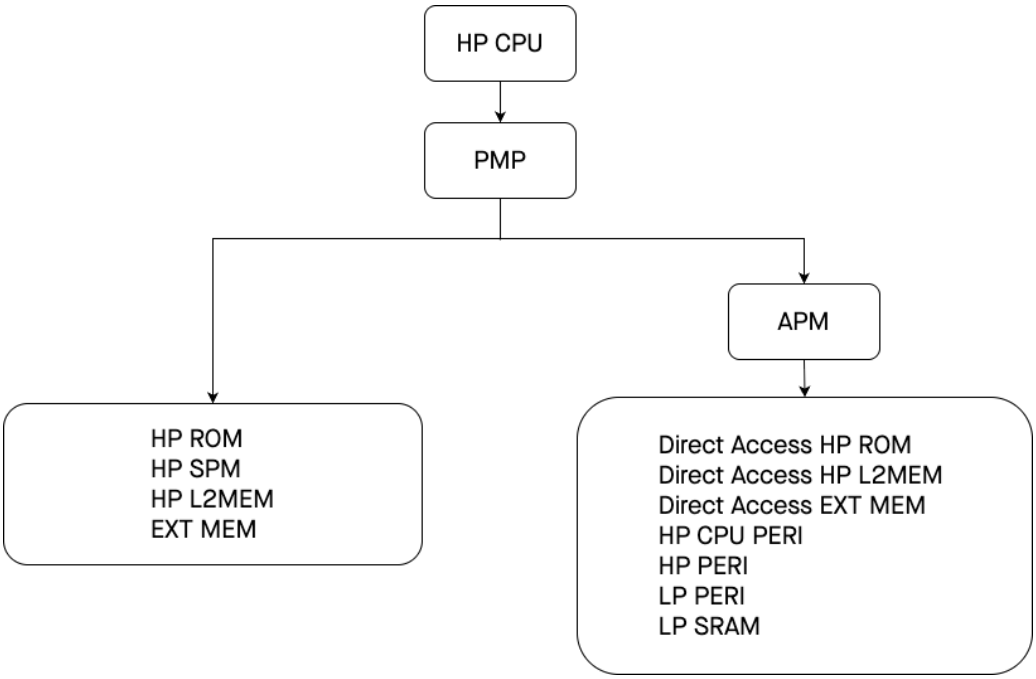


Figure 20.1-1. PMP-APM Management Relation

The diagram illustrates that when the HP CPUs access HP ROM, HP SPM, HP L2MEM, and EXT MEM, the access paths are solely managed by PMP. On the other hand, when the HP CPUs access the peripheral registers, HP ROM, HP L2MEM, and EXT MEM without going through the cache, the access paths are controlled by both PMP and APM. If the PMP check fails, the APM permission control will not be triggered.

PMP-related registers are located inside the HP CPUs and can be read or configured with special instructions. For how to configure PMP, please refer to Chapter 2 *High-Performance CPU* > 2.10.1 *Standard Physical Memory Protection*.

The following sections will provide a detailed introduction to the features, functions, and configurations of the APM module. When referring to HP CPU0/1, accessing HP ROM/HP L2MEM/EXT MEM always means direct access to HP ROM/HP L2MEM/EXT MEM.

20.2 Features

The APM module has the following features:

- Up to 32 configurable address ranges for each DMA master
- Access permission management for each CPU core to access internal memory, external memory, and peripheral registers
- Support for interrupts
- Support for exception information record

20.3 Functional Description

20.3.1 Architecture

The APM module contains three parts and five register modules as follows:

- **DMA APM**, containing register group HP_DMA_PMS_REG, responsible for:
  - configuring up to 32 address regions for DMA masters
  - managing the access permission to each address region for each DMA master
- **HP APM**, containing register groups HP_PERI_PMS_REG and LP2HP_PERI_PMS_REG, responsible for managing the access permissions to peripheral registers (HP CPU PERI, HP PERI), internal memory (HP ROM, HP L2MEM), and external memory (EXT MEM) in the HP system, specifically,
  - managing the access permissions for HP CPU0/1 in user mode to access all the above-mentioned slaves
  - managing the access permissions for HP CPU0/1 in machine mode to access all the above-mentioned slaves
  - managing the access permissions for LP CPU in machine mode to access all the above-mentioned slaves
- **LP APM**, containing register groups LP_PERI_PMS_REG and HP2LP_PERI_PMS_REG, responsible for managing the access permissions to peripheral registers (LP PERI) and internal memory (LP SRAM) in the LP system, specifically,
  - managing the access permissions for HP CPU0/1 in user mode to access LP PERI and LP SRAM
  - managing the access permissions for HP CPU0/1 in machine mode to access LP PERI and LP SRAM
  - managing the access permissions for LP CPU in machine mode to access LP PERI

Figure 20.3-1 shows the access paths managed by APM and the corresponding register groups.

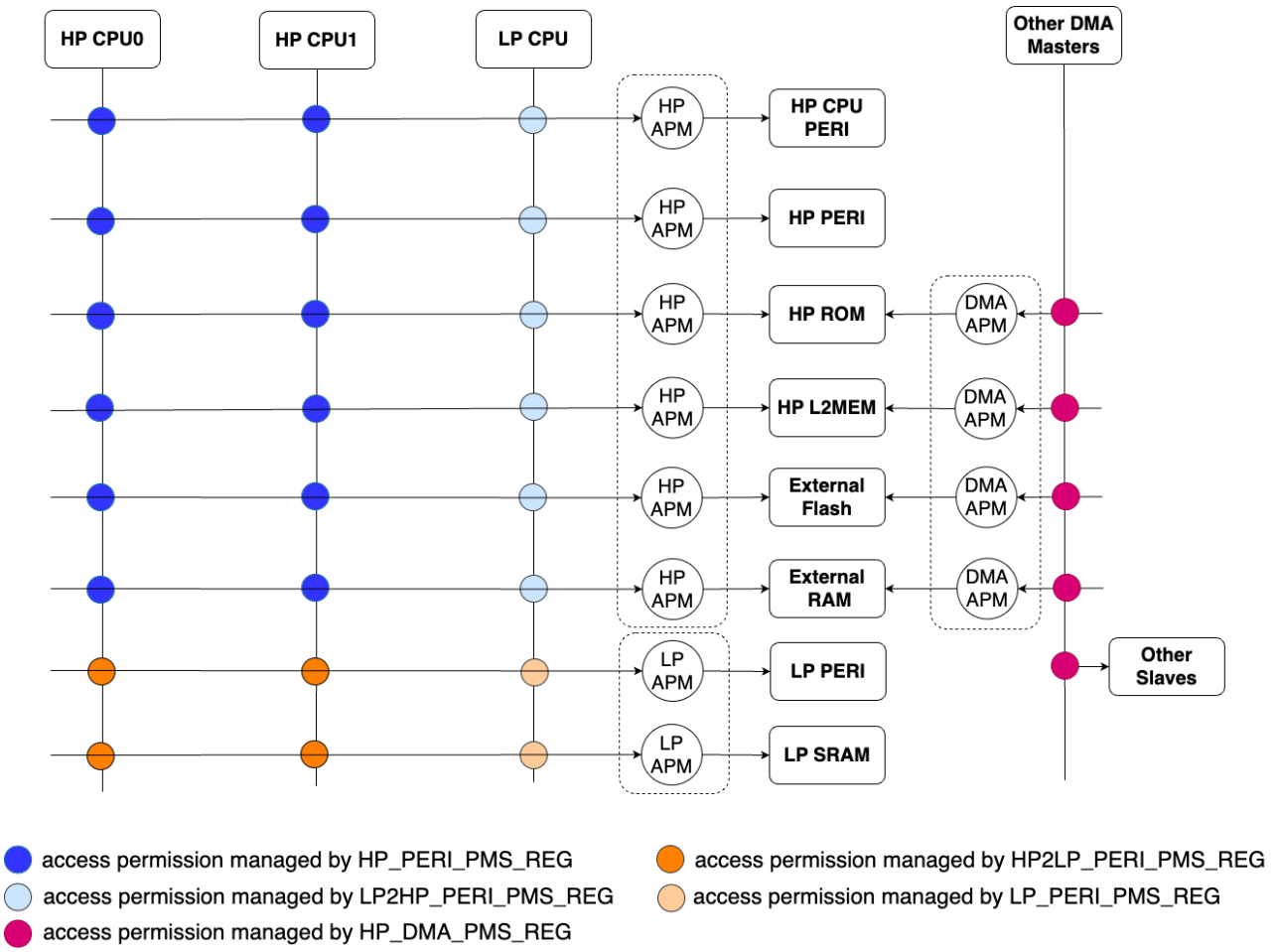


Figure 20.3-1. APM Architecture

Note:

- For register group names "HP2LP" and "LP2HP", the "HP" refers to HP CPU0/1, and "LP" refers to LP CPU.
- From Table 20.1-1, it can be seen that HP CPU and LP CPU access different address spaces of HP ROM, HP L2MEM, external flash, and external RAM. For HP CPU0/1, HP APM manages their direct access to internal and external memory without going through the cache.
- LP_PERI_PMS_REG includes eight additional configurable address ranges for HP APM and LP APM to manage the access permissions to peripheral registers, including HP CPU PERI, HP PERI, and LP PERI.

20.3.2 Address Ranges and Permissions Management

APM manages address ranges, which can be fixed or configurable. DMA APM can manage up to 32 configurable address ranges. In addition to the fixed address ranges, HP APM and LP APM can also manage in total eight configurable address ranges.

20.3.2.1 Address Ranges Managed by HP APM and LP APM

In the access paths controlled by HP APM and LP APM, the internal memory and external memory use fixed address ranges for access control, while peripheral registers not only support fixed address ranges permission control but also support eight configurable address ranges.

- Fixed address ranges
  - The fixed address ranges are the valid address spaces of peripheral registers, internal memory, and external memory. See the table notes under Table 20.1-1 for the address ranges of internal and external memory. Refer to Chapter 9 *System and Memory* > Table 9.3-2 for the address ranges of each peripheral.
  - To manage access to the fixed address ranges, please refer to Figure 20.3-1 for the access paths and their corresponding register groups.
- Configurable Address Range
  - APM can configure eight address ranges with PMS_PERI_REGION*n*/*m*_LOW_REG and PMS_PERI_REGION*n*/*m*_HIGH_REG (*n*/*m* = 0 ~ 7) in LP_PERI_PMS_REG. These configurable address ranges can be used by HP CPUs and LP CPU to access all on-chip peripheral registers (HP CPU PERI, HP PERI, LP PERI).
  - The access permissions are managed by PMS_PERI_REGION_PMS_REG in LP_PERI_PMS_REG.

When managing access permissions to peripheral registers, the permission management for the configurable address range should precede that of the fixed address range. Specifically:

- If the access address is within the configurable address range, then APM will check if this access is permitted to access the configurable address range.
  - If permitted, APM will continue to check if the access is permitted to the fixed address range.
  - If not permitted, the access will be denied and skip the permission check to the fixed address range.
- If the access address is outside the configurable address range, then it will skip the configurable address range check, but will continue to the fixed address range check.
- The management of configurable address ranges is a supplement to the management of fixed address ranges. For example, users can configure access permissions for a fixed address range, and deny access to two configurable ranges within the fixed range, resulting in a fixed address range containing two access-disabled address ranges.

20.3.2.2 Address Ranges Managed by DMA APM

The register group HP_DMA_PMS_REG provides 32 sets of registers (PMS_DMA_REGION*n*_LOW_REG ~ PMS_DMA_REGION*n*_HIGH_REG, *n* = 0 ~ 31) to configure address ranges for DMA masters to access. Note that since the address must be aligned to 4K bytes, the lowest 12 bits are set to 0.

HP_DMA_PMS_REG provides separate read and write permissions for each DMA master to access each address range.

20.4 Programming Procedure

As HP CPUs can work in either user mode or machine mode, HP APM and LP APM grant specific access permissions for each work mode. Therefore, users need to configure the work mode for HP CPUs first in order for the HP CPUs to access internal memory, external memory, and peripheral registers.

Since LP CPU only supports machine mode, HP APM and LP APM grant access permissions for the machine mode of LP CPU.

DMA masters can access slaves regardless of the CPUs' work mode. DMA APM provides separate read and write permissions for each DMA master.

20.4.1 Configuring Access Permissions for HP CPU0/1

The configuration process for HP CPU0/1 accessing internal memory, external memory, or peripheral registers is as follows:

1. Configure user mode or machine mode for HP CPU0/1. For how to configure the work mode, please refer to RISC-V Instruction Set Manual, Volume II: Privileged Architecture, Version 1.10.
2. If accessing peripheral registers, eight additional address ranges can be configured, please refer to Section 20.3.2 for details.
3. Configure access permissions:
   - Configure the access permissions to internal memory (HP L2MEM and HP ROM), external memory (External RAM and flash), and peripheral registers (HP PERI and HP CPU PERI) in the HP system through HP_PERI_PMS_REG:
     - Configure the permission of HP CPU*n* accessing the *MODULE_NAME* module in *MODE_NAME* mode using PMS_CORE*n*_*MODE_NAME*_*MODULE_NAME*_ALLOW, for example PMS_CORE*n*_MM_PSRAM_ALLOW.
   - Configure the access permissions to LP SRAM and LP PERI through HP2LP_PERI_PMS_REG:
     - Configure the permission of HP CPU*n* accessing the *MODULE_NAME* module in *MODE_NAME* mode using PMS_HP_CORE*n*_*MODE_NAME*_*MODULE_NAME*_ALLOW, for example PMS_HP_CORE*n*_MM_LP_SYSREG_ALLOW.

20.4.2 Configuring Access Permissions for LP CPU

LP CPU only supports machine mode. The configuration process for LP CPU accessing internal memory, external memory, or peripheral registers is as follows:

1. If accessing peripheral registers, eight additional address ranges can be configured, please refer to Section 20.3.2 for details.
2. Configure access permissions:
   - Configure the access permissions to internal memory (HP L2MEM and HP ROM), external memory (External RAM and flash), and peripheral registers (HP PERI and HP CPU PERI) in the HP system through LP2HP_PERI_PMS_REG:
     - Configure the permission of LP CPU accessing the *MODULE_NAME* module in machine mode using PMS_LP_MM_*MODULE_NAME*_ALLOW, for example PMS_LP_MM_PSRAM_ALLOW.
   - Configure the access permissions to peripheral registers (LP PERI) in the LP system through LP_PERI_PMS_REG:
     - Configure the permission of LP CPU accessing the *MODULE_NAME* module in machine mode using PMS_LP_MM_*MODULE_NAME*_ALLOW, for example PMS_LP_MM_LP_SYSREG_ALLOW.

20.4.3 Configuring Access Permission for DMA Masters

The configuration process for DMA masters accessing internal memory, external memory, and other address spaces is as follows:

1. Configure the address ranges accessible to all DMA masters:
   - Configure the high 20 bits of the starting address of region*n* using PMS_DMA_REGION*n*_LOW_REG.
   - Configure the high 20 bits of the ending address of region*n* using PMS_DMA_REGION*n*_HIGH_REG.
2. Configure the read and write permissions for each DMA master (*DMA_MASTER*) to access each address range:
   - Configure the read permissions using PMS_*DMA_MASTER*_PMS_R_REG, such as PMS_DMA_GMAC_PMS_R_REG.
   - Configure the write permissions using PMS_*DMA_MASTER*_PMS_W_REG, such as PMS_DMA_GMAC_PMS_W_REG.

20.5 Illegal Access and Interrupts

When the APM module manages access paths, it parses the information carried on the bus (e.g., master ID, access direction (read or write), access address, and working mode) and then determines the access permission. When the APM module confirms that the access lacks permission, ESP32-P4 will consider it as an unauthorized access and will take the following actions:

- Reject the access request and provide an error response.
- HP APM will trigger ICM_SYS_REG_INT interrupt signal.
- DMA APM will trigger ICM_SYS_REG_INT interrupt signal.
- LP APM will trigger LP_SYSREG_INT interrupt signal.

These interrupt signals are generated by the internal interrupt sources of the APM module. Table 20.5-1 lists the internal interrupt sources, interrupt trigger conditions, and the generated interrupt signals.

Table 20.5-1. APM Internal Interrupt Sources

Internal Interrupt Source	Trigger Condition	Interrupt Signal
ICM_CPU_ADDRHOLE_INTR	Triggered when HP APM detects unauthorized access; or when DMA APM detects unauthorized access from USB 2.0 High-Speed OTG, USB 2.0 Full-Speed OTG, EMAC, SDMMC, Trace0/1, GDMA-AHB, L2MEM Monitor, or SPM Monitor.	ICM_SYS_REG_INT
ICM_SYS_ADDRHOLE_INTR	Triggered when DMA APM detects unauthorized access from VDMA, GDMA-AXI, 2D-DMA, or H264 DMA.	ICM_SYS_REG_INT
LP_ADDRHOLE_INTR	Triggered when LP APM detects unauthorized access.	LP_SYSREG_INTR

APM automatically records relevant information about illegal accesses, including master ID, access direction (read or write), access address, illegal access flags, etc. All this information can be retrieved from the relevant registers described in Chapter *21 System Registers (SYSREG)*.

20.6 Register Summary

20.6.1 HP_DMA_PMS_REG

The addresses in this section are relative to the HP_DMA_PMS base address provided in Table 9.3-2 in Chapter 9 *System and Memory*.

The abbreviations given in Column **Access** are explained in Section *Access Types for Registers*.

Name	Description	Address	Access
Version Control Registers			
PMS_DMA_DATE_REG	Version control register	0x0000	R/W
Clock Gating Registers			
PMS_DMA_CLK_EN_REG	Clock gating register	0x0004	R/W
Region Configuration Registers			
PMS_DMA_REGION0_LOW_REG	Region0 start address configuration register	0x0008	R/W
PMS_DMA_REGION0_HIGH_REG	Region0 end address configuration register	0x000C	R/W
PMS_DMA_REGION1_LOW_REG	Region1 start address configuration register	0x0010	R/W
PMS_DMA_REGION1_HIGH_REG	Region1 end address configuration register	0x0014	R/W
PMS_DMA_REGION2_LOW_REG	Region2 start address configuration register	0x0018	R/W
PMS_DMA_REGION2_HIGH_REG	Region2 end address configuration register	0x001C	R/W
PMS_DMA_REGION3_LOW_REG	Region3 start address configuration register	0x0020	R/W
PMS_DMA_REGION3_HIGH_REG	Region3 end address configuration register	0x0024	R/W
PMS_DMA_REGION4_LOW_REG	Region4 start address configuration register	0x0028	R/W
PMS_DMA_REGION4_HIGH_REG	Region4 end address configuration register	0x002C	R/W
PMS_DMA_REGION5_LOW_REG	Region5 start address configuration register	0x0030	R/W
PMS_DMA_REGION5_HIGH_REG	Region5 end address configuration register	0x0034	R/W
PMS_DMA_REGION6_LOW_REG	Region6 start address configuration register	0x0038	R/W
PMS_DMA_REGION6_HIGH_REG	Region6 end address configuration register	0x003C	R/W
PMS_DMA_REGION7_LOW_REG	Region7 start address configuration register	0x0040	R/W

Name	Description	Address	Access
PMS_DMA_REGION7_HIGH_REG	Region7 end address configuration register	0x0044	R/W
PMS_DMA_REGION8_LOW_REG	Region8 start address configuration register	0x0048	R/W
PMS_DMA_REGION8_HIGH_REG	Region8 end address configuration register	0x004C	R/W
PMS_DMA_REGION9_LOW_REG	Region9 start address configuration register	0x0050	R/W
PMS_DMA_REGION9_HIGH_REG	Region9 end address configuration register	0x0054	R/W
PMS_DMA_REGION10_LOW_REG	Region10 start address configuration register	0x0058	R/W
PMS_DMA_REGION10_HIGH_REG	Region10 end address configuration register	0x005C	R/W
PMS_DMA_REGION11_LOW_REG	Region11 start address configuration register	0x0060	R/W
PMS_DMA_REGION11_HIGH_REG	Region11 end address configuration register	0x0064	R/W
PMS_DMA_REGION12_LOW_REG	Region12 start address configuration register	0x0068	R/W
PMS_DMA_REGION12_HIGH_REG	Region12 end address configuration register	0x006C	R/W
PMS_DMA_REGION13_LOW_REG	Region13 start address configuration register	0x0070	R/W
PMS_DMA_REGION13_HIGH_REG	Region13 end address configuration register	0x0074	R/W
PMS_DMA_REGION14_LOW_REG	Region14 start address configuration register	0x0078	R/W
PMS_DMA_REGION14_HIGH_REG	Region14 end address configuration register	0x007C	R/W
PMS_DMA_REGION15_LOW_REG	Region15 start address configuration register	0x0080	R/W
PMS_DMA_REGION15_HIGH_REG	Region15 end address configuration register	0x0084	R/W
PMS_DMA_REGION16_LOW_REG	Region16 start address configuration register	0x0088	R/W
PMS_DMA_REGION16_HIGH_REG	Region16 end address configuration register	0x008C	R/W
PMS_DMA_REGION17_LOW_REG	Region17 start address configuration register	0x0090	R/W
PMS_DMA_REGION17_HIGH_REG	Region17 end address configuration register	0x0094	R/W

Name	Description	Address	Access
PMS_DMA_REGION18_LOW_REG	Region18 start address configuration register	0x0098	R/W
PMS_DMA_REGION18_HIGH_REG	Region18 end address configuration register	0x009C	R/W
PMS_DMA_REGION19_LOW_REG	Region19 start address configuration register	0x00A0	R/W
PMS_DMA_REGION19_HIGH_REG	Region19 end address configuration register	0x00A4	R/W
PMS_DMA_REGION20_LOW_REG	Region20 start address configuration register	0x00A8	R/W
PMS_DMA_REGION20_HIGH_REG	Region20 end address configuration register	0x00AC	R/W
PMS_DMA_REGION21_LOW_REG	Region21 start address configuration register	0x00B0	R/W
PMS_DMA_REGION21_HIGH_REG	Region21 end address configuration register	0x00B4	R/W
PMS_DMA_REGION22_LOW_REG	Region22 start address configuration register	0x00B8	R/W
PMS_DMA_REGION22_HIGH_REG	Region22 end address configuration register	0x00BC	R/W
PMS_DMA_REGION23_LOW_REG	Region23 start address configuration register	0x00C0	R/W
PMS_DMA_REGION23_HIGH_REG	Region23 end address configuration register	0x00C4	R/W
PMS_DMA_REGION24_LOW_REG	Region24 start address configuration register	0x00C8	R/W
PMS_DMA_REGION24_HIGH_REG	Region24 end address configuration register	0x00CC	R/W
PMS_DMA_REGION25_LOW_REG	Region25 start address configuration register	0x00D0	R/W
PMS_DMA_REGION25_HIGH_REG	Region25 end address configuration register	0x00D4	R/W
PMS_DMA_REGION26_LOW_REG	Region26 start address configuration register	0x00D8	R/W
PMS_DMA_REGION26_HIGH_REG	Region26 end address configuration register	0x00DC	R/W
PMS_DMA_REGION27_LOW_REG	Region27 start address configuration register	0x00E0	R/W
PMS_DMA_REGION27_HIGH_REG	Region27 end address configuration register	0x00E4	R/W
PMS_DMA_REGION28_LOW_REG	Region28 start address configuration register	0x00E8	R/W

Name	Description	Address	Access
PMS_DMA_REGION28_HIGH_REG	Region28 end address configuration register	0x00EC	R/W
PMS_DMA_REGION29_LOW_REG	Region29 start address configuration register	0x00F0	R/W
PMS_DMA_REGION29_HIGH_REG	Region29 end address configuration register	0x00F4	R/W
PMS_DMA_REGION30_LOW_REG	Region30 start address configuration register	0x00F8	R/W
PMS_DMA_REGION30_HIGH_REG	Region30 end address configuration register	0x00FC	R/W
PMS_DMA_REGION31_LOW_REG	Region31 start address configuration register	0x0100	R/W
PMS_DMA_REGION31_HIGH_REG	Region31 end address configuration register	0x0104	R/W
DMA Masters Read and Write Permission Control Registers			
PMS_DMA_GDMA_CH0_R_PMS_REG	GDMA ch0 read permission control register	0x0108	R/W
PMS_DMA_GDMA_CH0_W_PMS_REG	GDMA ch0 write permission control register	0x010C	R/W
PMS_DMA_GDMA_CH1_R_PMS_REG	GDMA ch1 read permission control register	0x0110	R/W
PMS_DMA_GDMA_CH1_W_PMS_REG	GDMA ch1 write permission control register	0x0114	R/W
PMS_DMA_GDMA_CH2_R_PMS_REG	GDMA ch2 read permission control register	0x0118	R/W
PMS_DMA_GDMA_CH2_W_PMS_REG	GDMA ch2 write permission control register	0x011C	R/W
PMS_DMA_GDMA_CH3_R_PMS_REG	GDMA ch3 read permission control register	0x0120	R/W
PMS_DMA_GDMA_CH3_W_PMS_REG	GDMA ch3 write permission control register	0x0124	R/W
PMS_DMA_AHB_PDMA_ADC_R_PMS_REG	GDMA-AHB ADC read permission control register	0x0128	R/W
PMS_DMA_AHB_PDMA_ADC_W_PMS_REG	GDMA-AHB ADC write permission control register	0x012C	R/W
PMS_DMA_AHB_PDMA_I2S0_R_PMS_REG	GDMA-AHB I2S0 read permission control register	0x0130	R/W
PMS_DMA_AHB_PDMA_I2S0_W_PMS_REG	GDMA-AHB I2S0 write permission control register	0x0134	R/W
PMS_DMA_AHB_PDMA_I2S1_R_PMS_REG	GDMA-AHB I2S1 read permission control register	0x0138	R/W
PMS_DMA_AHB_PDMA_I2S1_W_PMS_REG	GDMA-AHB I2S1 write permission control register	0x013C	R/W

Name	Description	Address	Access
PMS_DMA_AHB_PDMA_I2S2_R_PMS_REG	GDMA-AHB I2S2 read permission control register	0x0140	R/W
PMS_DMA_AHB_PDMA_I2S2_W_PMS_REG	GDMA-AHB I2S2 write permission control register	0x0144	R/W
PMS_DMA_AHB_PDMA_I3C_MST_R_PMS_REG	GDMA-AHB I3C MST read permission control register	0x0148	R/W
PMS_DMA_AHB_PDMA_I3C_MST_W_PMS_REG	GDMA-AHB I3C MST write permission control register	0x014C	R/W
PMS_DMA_AHB_PDMA_UHCI0_R_PMS_REG	GDMA-AHB UHCI read permission control register	0x0150	R/W
PMS_DMA_AHB_PDMA_UHCI0_W_PMS_REG	GDMA-AHB UHCI write permission control register	0x0154	R/W
PMS_DMA_AHB_PDMA_RMT_R_PMS_REG	GDMA-AHB RMT read permission control register	0x0158	R/W
PMS_DMA_AHB_PDMA_RMT_W_PMS_REG	GDMA-AHB RMT write permission control register	0x0170	R/W
PMS_DMA_AXI_PDMA_LCDCAM_R_PMS_REG	GDMA-AXI LCD_CAM read permission control register	0x0174	R/W
PMS_DMA_AXI_PDMA_LCDCAM_W_PMS_REG	GDMA-AXI LCD_CAM write permission control register	0x0178	R/W
PMS_DMA_AXI_PDMA_GPSPI2_R_PMS_REG	GDMA-AXI GPSPI2 read permission control register	0x017C	R/W
PMS_DMA_AXI_PDMA_GPSPI2_W_PMS_REG	GDMA-AXI GPSPI2 write permission control register	0x0180	R/W
PMS_DMA_AXI_PDMA_GPSPI3_R_PMS_REG	GDMA-AXI GPSPI3 read permission control register	0x0184	R/W
PMS_DMA_AXI_PDMA_GPSPI3_W_PMS_REG	GDMA-AXI GPSPI3 write permission control register	0x0188	R/W
PMS_DMA_AXI_PDMA_PARLIO_R_PMS_REG	GDMA-AXI PARLIO read permission control register	0x018C	R/W
PMS_DMA_AXI_PDMA_PARLIO_W_PMS_REG	GDMA-AXI PARLIO write permission control register	0x0190	R/W
PMS_DMA_AXI_PDMA_AES_R_PMS_REG	GDMA-AXI AES read permission control register	0x0194	R/W
PMS_DMA_AXI_PDMA_AES_W_PMS_REG	GDMA-AXI AES write permission control register	0x0198	R/W
PMS_DMA_AXI_PDMA_SHA_R_PMS_REG	GDMA-AXI SHA read permission control register	0x019C	R/W
PMS_DMA_AXI_PDMA_SHA_W_PMS_REG	GDMA-AXI SHA write permission control register	0x01A0	R/W
PMS_DMA_DMA2D_JPEG_PMS_R_REG	2D-DMA JPEG read permission control register	0x01A4	R/W

Name	Description	Address	Access
PMS_DMA_DMA2D_JPEG_PMS_W_REG	2D-DMA JPEG write permission control register	0x01A8	R/W
PMS_DMA_USB_PMS_R_REG	USB 2.0 High-Speed OTG read permission control register	0x01AC	R/W
PMS_DMA_USB_PMS_W_REG	USB 2.0 High-Speed OTG write permission control register	0x01B0	R/W
PMS_DMA_GMAC_PMS_R_REG	EMAC read permission control register	0x01B4	R/W
PMS_DMA_GMAC_PMS_W_REG	EMAC write permission control register	0x01B8	R/W
PMS_DMA_SDMMC_PMS_R_REG	SDMMC read permission control register	0x01BC	R/W
PMS_DMA_SDMMC_PMS_W_REG	SDMMC write permission control register	0x01C0	R/W
PMS_DMA_USBOTG11_PMS_R_REG	USB 2.0 Full-Speed OTG read permission control register	0x01C4	R/W
PMS_DMA_USBOTG11_PMS_W_REG	USB 2.0 Full-Speed OTG write permission control register	0x01C8	R/W
PMS_DMA_TRACE0_PMS_R_REG	TRACE0 read permission control register	0x01CC	R/W
PMS_DMA_TRACE0_PMS_W_REG	TRACE0 write permission control register	0x01D0	R/W
PMS_DMA_TRACE1_PMS_R_REG	TRACE1 read permission control register	0x01D4	R/W
PMS_DMA_TRACE1_PMS_W_REG	TRACE1 write permission control register	0x01D8	R/W
PMS_DMA_L2MEM_MON_PMS_W_REG	L2MEM Monitor write permission control register	0x01E0	R/W
PMS_DMA_SPM_MON_PMS_W_REG	SPM Monitor write permission control register	0x01E8	R/W
PMS_DMA_H264_PMS_R_REG	H264 DMA read permission control register	0x01FC	R/W
PMS_DMA_H264_PMS_W_REG	H264 DMA write permission control register	0x0200	R/W
PMS_DMA_DMA2D_PPA_PMS_R_REG	2D-DMA PPA read permission control register	0x0204	R/W
PMS_DMA_DMA2D_PPA_PMS_W_REG	2D-DMA PPA write permission control register	0x0208	R/W
PMS_DMA_DMA2D_DUMMY_PMS_R_REG	2D-DMA Dummy read permission control register	0x020C	R/W
PMS_DMA_DMA2D_DUMMY_PMS_W_REG	2D-DMA Dummy write permission control register	0x0210	R/W
PMS_DMA_AHB_PDMA_DUMMY_R_PMS_REG	GDMA-AHB Dummy read permission control register	0x0214	R/W

Name	Description	Address	Access
PMS_DMA_AHB_PDMA_DUMMY_W_PMS_REG	GDMA-AHB Dummy write permission control register	0x0218	R/W
PMS_DMA_AXI_PDMA_DUMMY_R_PMS_REG	GDMA-AXI Dummy read permission control register	0x021C	R/W
PMS_DMA_AXI_PDMA_DUMMY_W_PMS_REG	GDMA-AXI Dummy write permission control register	0x0220	R/W

20.6.2 HP_PERI_PMS_REG

The addresses in this section are relative to the HP_PERI_PMS base address provided in Table 9.3-2 in Chapter 9 *System and Memory*.

The abbreviations given in Column **Access** are explained in Section *Access Types for Registers*.

Name	Description	Address	Access
Version Control Registers			
PMS_HP_PERI_PMS_DATE_REG	Version control register	0x0000	R/W
Clock Gating Registers			
PMS_HP_PERI_PMS_CLK_EN_REG	Clock gating register	0x0004	R/W
HP CPU Permission Control Registers			
PMS_COREO_MM_HP_PERI_PMS_REG0_REG	Permission control register0 for HP CPU0 in machine mode	0x0008	R/W
PMS_COREO_MM_HP_PERI_PMS_REG1_REG	Permission control register1 for HP CPU0 in machine mode	0x000C	R/W
PMS_COREO_MM_HP_PERI_PMS_REG2_REG	Permission control register2 for HP CPU0 in machine mode	0x0010	R/W
PMS_COREO_MM_HP_PERI_PMS_REG3_REG	Permission control register3 for HP CPU0 in machine mode	0x0014	R/W
PMS_COREO_UM_HP_PERI_PMS_REG0_REG	Permission control register0 for HP CPU0 in user mode	0x0018	R/W
PMS_COREO_UM_HP_PERI_PMS_REG1_REG	Permission control register1 for HP CPU0 in user mode	0x001C	R/W
PMS_COREO_UM_HP_PERI_PMS_REG2_REG	Permission control register2 for HP CPU0 in user mode	0x0020	R/W
PMS_COREO_UM_HP_PERI_PMS_REG3_REG	Permission control register3 for HP CPU0 in user mode	0x0024	R/W
PMS_CORE1_MM_HP_PERI_PMS_REG0_REG	Permission control register0 for HP CPU1 in machine mode	0x0028	R/W
PMS_CORE1_MM_HP_PERI_PMS_REG1_REG	Permission control register1 for HP CPU1 in machine mode	0x002C	R/W
PMS_CORE1_MM_HP_PERI_PMS_REG2_REG	Permission control register2 for HP CPU1 in machine mode	0x0030	R/W

Name	Description	Address	Access
PMS_CORE1_MM_HP_PERI_PMS_REG3_REG	Permission control register3 for HP CPU1 in machine mode	0x0034	R/W
PMS_CORE1_UM_HP_PERI_PMS_REG0_REG	Permission control register0 for HP CPU1 in user mode	0x0038	R/W
PMS_CORE1_UM_HP_PERI_PMS_REG1_REG	Permission control register1 for HP CPU1 in user mode	0x003C	R/W
PMS_CORE1_UM_HP_PERI_PMS_REG2_REG	Permission control register2 for HP CPU1 in user mode	0x0040	R/W
PMS_CORE1_UM_HP_PERI_PMS_REG3_REG	Permission control register3 for HP CPU1 in user mode	0x0044	R/W

20.6.3 LP2HP_PERI_PMS_REG

The addresses in this section are relative to the LP2HP_PERI_PMS base address provided in Table 9.3-2 in Chapter 9 *System and Memory*.

The abbreviations given in Column **Access** are explained in Section *Access Types for Registers*.

Name	Description	Address	Access
Version Control Registers			
PMS_LP2HP_PERI_PMS_DATE_REG	Version control register	0x0000	R/W
Clock Gating Registers			
PMS_LP2HP_PERI_PMS_CLK_EN_REG	Clock gating register	0x0004	R/W
LP CPU Permission Control Registers			
PMS_LP_MM_PMS_REG0_REG	Permission control register0 for the LP CPU in machine mode	0x0008	R/W
PMS_LP_MM_PMS_REG1_REG	Permission control register1 for the LP CPU in machine mode	0x0030	R/W
PMS_LP_MM_PMS_REG2_REG	Permission control register2 for the LP CPU in machine mode	0x00A4	R/W
PMS_LP_MM_PMS_REG3_REG	Permission control register3 for the LP CPU in machine mode	0x011C	R/W

20.6.4 LP_PERI_PMS_REG

The addresses in this section are relative to the LP_PERI_PMS base address provided in Table 9.3-2 in Chapter 9 *System and Memory*.

The abbreviations given in Column **Access** are explained in Section *Access Types for Registers*.

Name	Description	Address	Access
Version Control Registers			

Name	Description	Address	Access
PMS_LP_PERI_PMS_DATE_REG	Version control register	0x0000	R/W
Clock Gating Registers			
PMS_LP_PERI_PMS_CLK_EN_REG	Clock gating register	0x0004	R/W
LP CPU Permission Control Registers			
PMS_LP_MM_LP_PERI_PMS_REG0_REG	Permission control register0 for LP CPU in machine mode	0x0008	R/W
Configurable Address Range Configuration Registers			
PMS_PERI_REGION0_LOW_REG	Region0 start address configuration register	0x000C	R/W
PMS_PERI_REGION0_HIGH_REG	Region0 end address configuration register	0x0010	R/W
PMS_PERI_REGION1_LOW_REG	Region1 start address configuration register	0x0014	R/W
PMS_PERI_REGION_PMS_REG	Permission control of access to address region 0-1	0x001C	R/W
PMS_PERI_REGION_2_TO_7_PMS_REG	Permission control of access to address region 2-7	0x0020	R/W
PMS_PERI_REGION2_LOW_REG	Address region 2 start address configuration register	0x0024	R/W
PMS_PERI_REGION2_HIGH_REG	Address region 2 end address configuration register	0x0028	R/W
PMS_PERI_REGION3_LOW_REG	Address region 3 start address configuration register	0x002C	R/W
PMS_PERI_REGION3_HIGH_REG	Address region 3 end address configuration register	0x0030	R/W
PMS_PERI_REGION4_LOW_REG	Address region 4 start address configuration register	0x0034	R/W
PMS_PERI_REGION4_HIGH_REG	Address region 4 end address configuration register	0x0038	R/W
PMS_PERI_REGION5_LOW_REG	Address region 5 start address configuration register	0x003C	R/W
PMS_PERI_REGION5_HIGH_REG	Address region 5 end address configuration register	0x0040	R/W
PMS_PERI_REGION6_LOW_REG	Address region 6 start address configuration register	0x0044	R/W
PMS_PERI_REGION6_HIGH_REG	Address region 6 end address configuration register	0x0048	R/W
PMS_PERI_REGION7_LOW_REG	Address region 7 start address configuration register	0x004C	R/W
PMS_PERI_REGION7_HIGH_REG	Address region 7 end address configuration register	0x0050	R/W
PMS_PERI_REGION1_HIGH_REG	Region1 end address configuration register	0x0018	R/W

20.6.5 HP2LP_PERI_PMS_REG

The addresses in this section are relative to the HP2LP_PERI_PMS base address provided in Table 9.3-2 in Chapter 9 *System and Memory*.

The abbreviations given in Column **Access** are explained in Section *Access Types for Registers*.

Name	Description	Address	Access
Version Control Registers			
PMS_HP2LP_PERI_PMS_DATE_REG	Version control register	0x0000	R/W
Clock Gating Registers			
PMS_HP2LP_PERI_PMS_CLK_EN_REG	Clock gating register	0x0004	R/W
HP CPU Permission Control Registers			
PMS_HP_CORE0_MM_PMS_REG0_REG	Permission control register0 for HP CPU0 in machine mode	0x0008	R/W
PMS_HP_CORE0_UM_PMS_REG0_REG	Permission control register0 for HP CPU0 in user mode	0x000C	R/W
PMS_HP_CORE1_MM_PMS_REG0_REG	Permission control register0 for HP CPU1 in machine mode	0x0010	R/W
PMS_HP_CORE1_UM_PMS_REG0_REG	Permission control register0 for HP CPU1 in user mode	0x0014	R/W

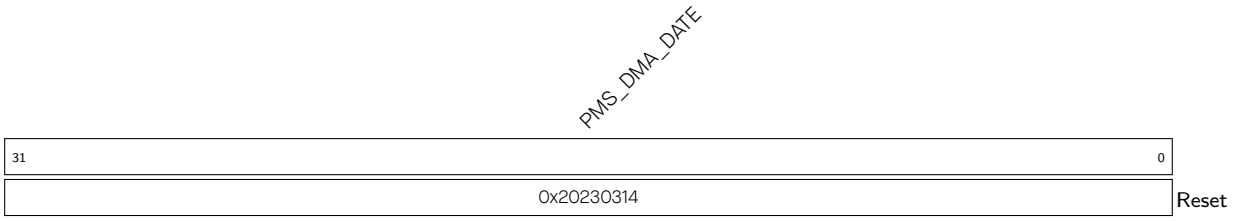
20.7 Registers

20.7.1 HP_DMA_PMS_REG

The addresses in this section are relative to the HP_DMA_PMS base address provided in Table 9.3-2 in Chapter 9 *System and Memory*.

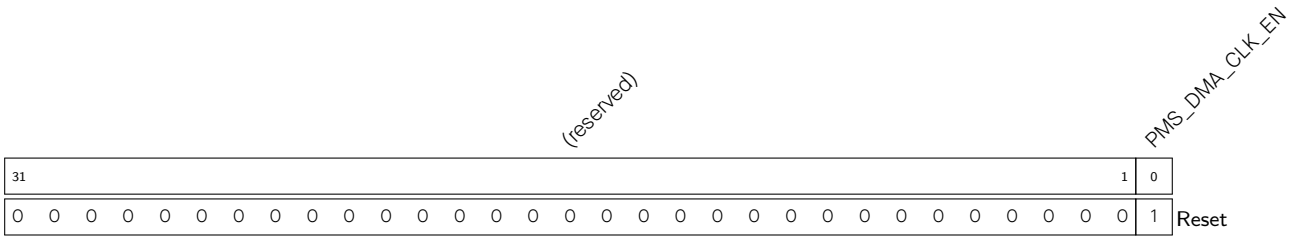
For how to program reserved fields, please refer to Section Programming Reserved Register Field.

Register 20.1. PMS_DMA_DATE_REG (0x0000)



PMS_DMA_DATE Version control register. (R/W)

Register 20.2. PMS_DMA_CLK_EN_REG (0x0004)



PMS_DMA_CLK_EN Configures whether to keep the clock always on.

0: Enable automatic clock gating.

1: Keep the clock always on.

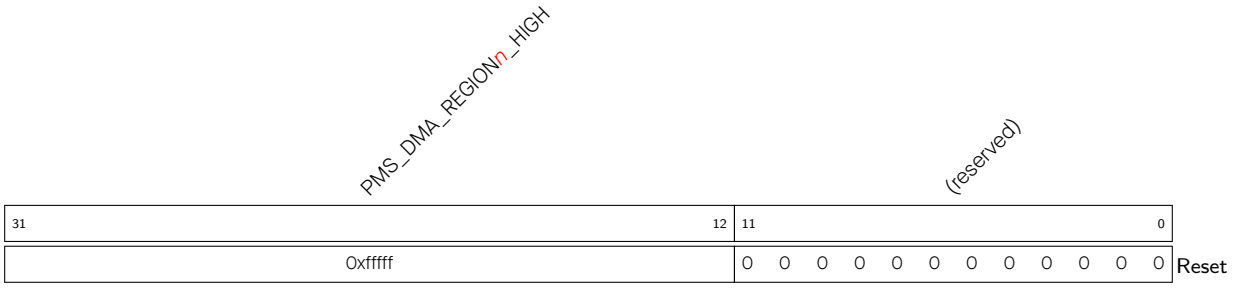
(R/W)

Register 20.3. PMS_DMA_REGION*n*_LOW_REG (*n*: 0-31) (0x0008+0x8n*)**



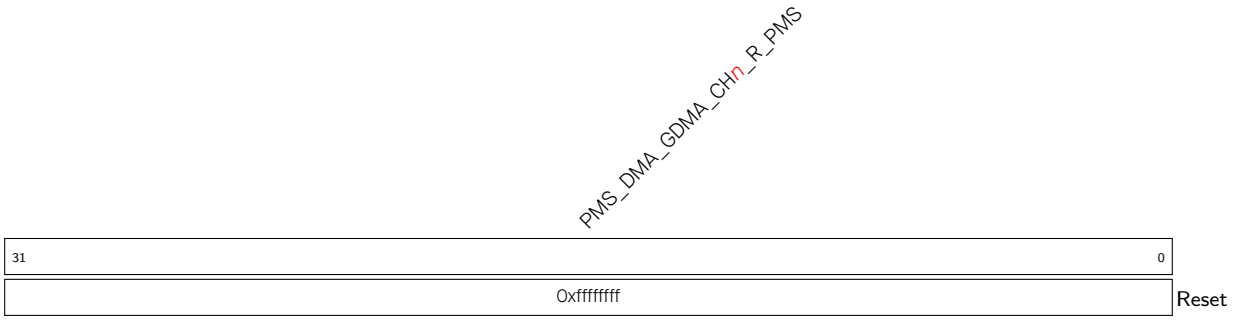
PMS_DMA_REGION*n*_LOW Configures the high 20 bits of the start address for region*n*. (R/W)

Register 20.4. PMS_DMA_REGIONn_HIGH_REG (n: 0-31) (0x000C+0x8*n)



PMS_DMA_REGIONn_HIGH Configures the high 20 bits of the end address for regionn. (R/W)

Register 20.5. PMS_DMA_GDMA_CHn_R_PMS_REG (n: 0-3) (0x0108+0x8*n)



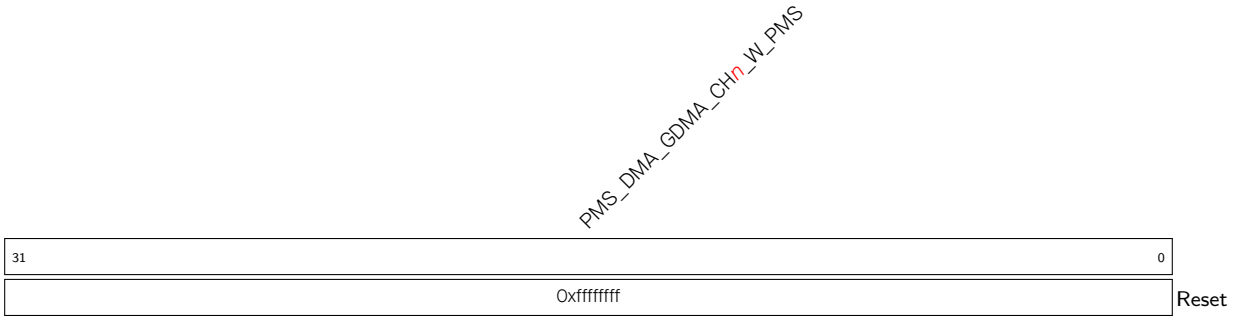
PMS_DMA_GDMA_CHn_R_PMS Configures the permission for GDMA chn to read 32 address regions. Bit 0 corresponds to region0, and so on.

0: Disable read permission.

1: Enable read permission.

(R/W)

Register 20.6. PMS_DMA_GDMA_CHn_W_PMS_REG (n: 0-3) (0x010C+0x8*n)



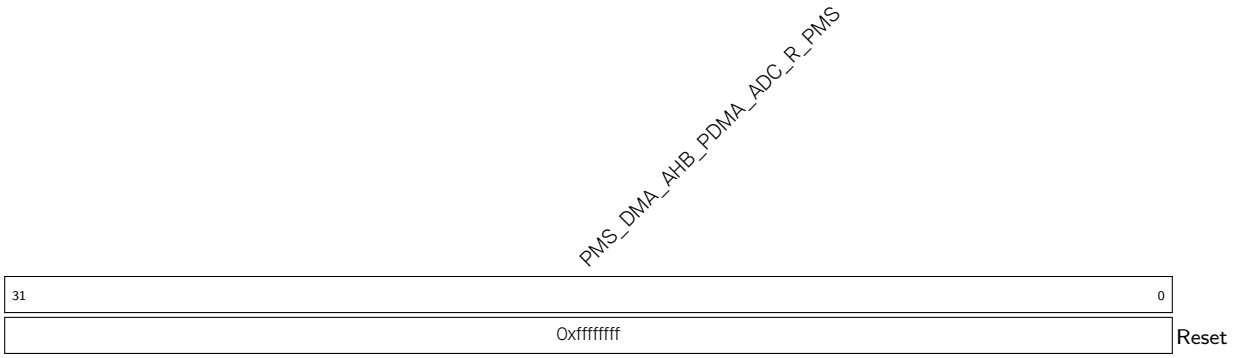
PMS_DMA_GDMA_CHn_W_PMS Configures the permission for GDMA chn to write 32 address regions. Bit 0 corresponds to region0, and so on.

0: Disable write permission.

1: Enable write permission.

(R/W)

Register 20.7. PMS_DMA_AHB_PDMA_ADC_R_PMS_REG (0x0128)



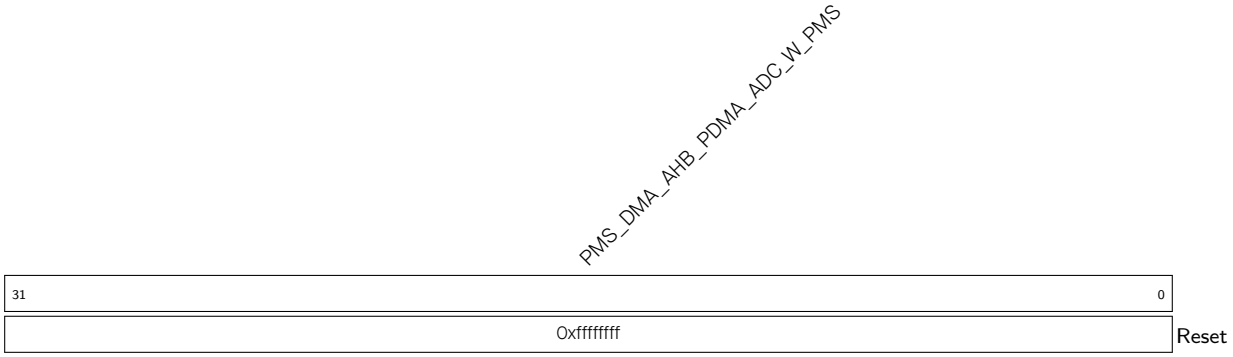
PMS_DMA_AHB_PDMA_ADC_R_PMS Configures GDMA-AHB permission to read 32 address ranges requested by ADC. Bit 0 corresponds to region0, and so on.

0: Disable read permission.

1: Enable read permission.

(R/W)

Register 20.8. PMS_DMA_AHB_PDMA_ADC_W_PMS_REG (0x012C)



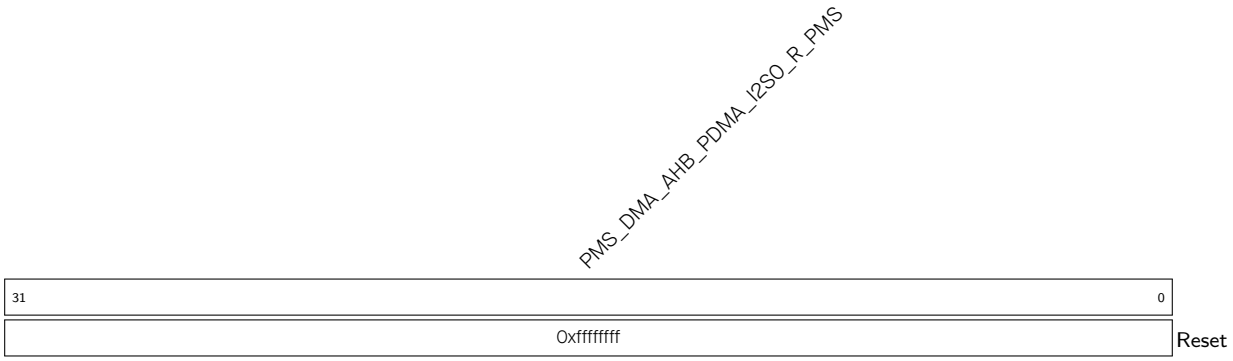
PMS_DMA_AHB_PDMA_ADC_W_PMS Configures GDMA-AHB permission to write 32 address ranges requested by ADC. Bit 0 corresponds to region0, and so on.

0: Disable write permission.

1: Enable write permission.

(R/W)

Register 20.9. PMS_DMA_AHB_PDMA_I2S0_R_PMS_REG (0x0130)



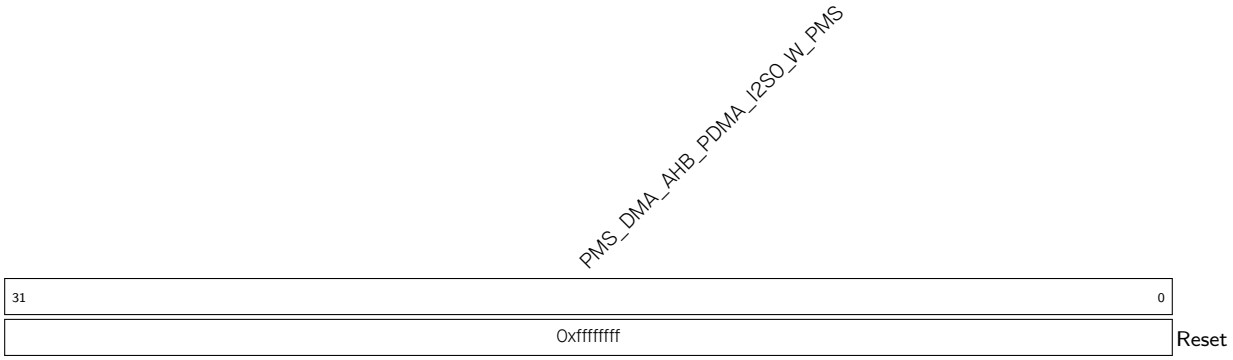
PMS_DMA_AHB_PDMA_I2S0_R_PMS Configures GDMA-AHB permission to read 32 address ranges requested by I2S0. Bit 0 corresponds to region0, and so on.

0: Disable read permission.

1: Enable read permission.

(R/W)

Register 20.10. PMS_DMA_AHB_PDMA_I2S0_W_PMS_REG (0x0134)



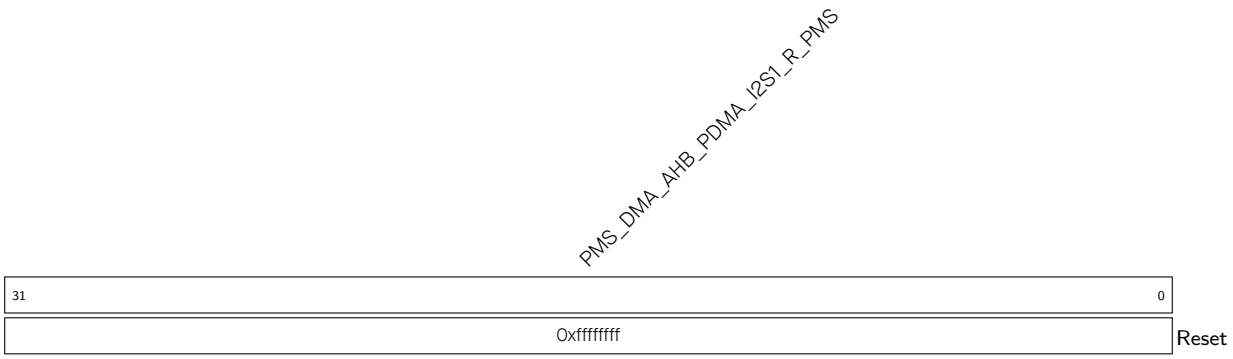
PMS_DMA_AHB_PDMA_I2S0_W_PMS Configures GDMA-AHB permission to write 32 address ranges requested by I2S0. Bit 0 corresponds to region0, and so on.

0: Disable write permission.

1: Enable write permission.

(R/W)

Register 20.11. PMS_DMA_AHB_PDMA_I2S1_R_PMS_REG (0x0138)



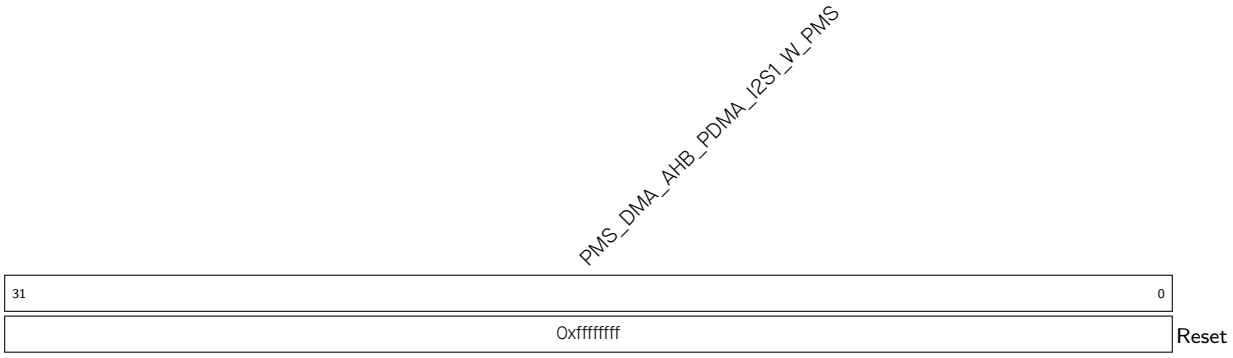
PMS_DMA_AHB_PDMA_I2S1_R_PMS Configures GDMA-AHB permission to read 32 address ranges requested by I2S1. Bit 0 corresponds to region0, and so on.

0: Disable read permission.

1: Enable read permission.

(R/W)

Register 20.12. PMS_DMA_AHB_PDMA_I2S1_W_PMS_REG (0x013C)



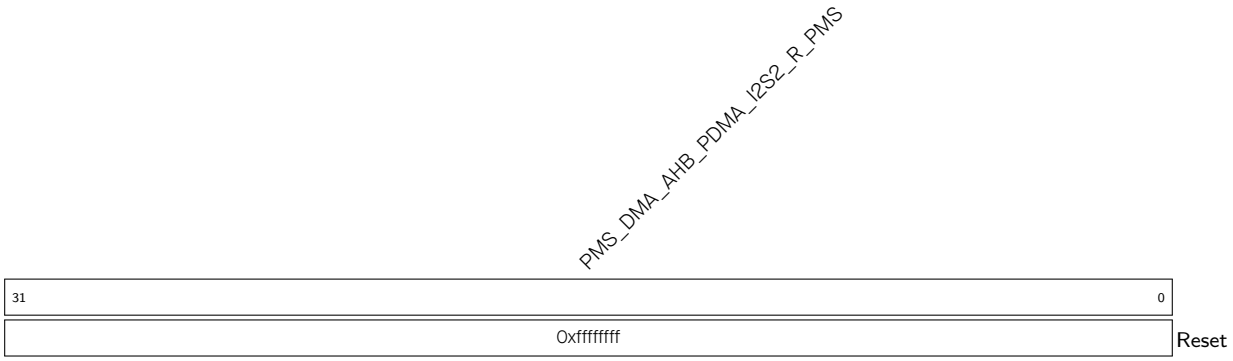
PMS_DMA_AHB_PDMA_I2S1_W_PMS Configures GDMA-AHB permission to write 32 address ranges requested by I2S1. Bit 0 corresponds to region0, and so on.

0: Disable write permission.

1: Enable write permission.

(R/W)

Register 20.13. PMS_DMA_AHB_PDMA_I2S2_R_PMS_REG (0x0140)



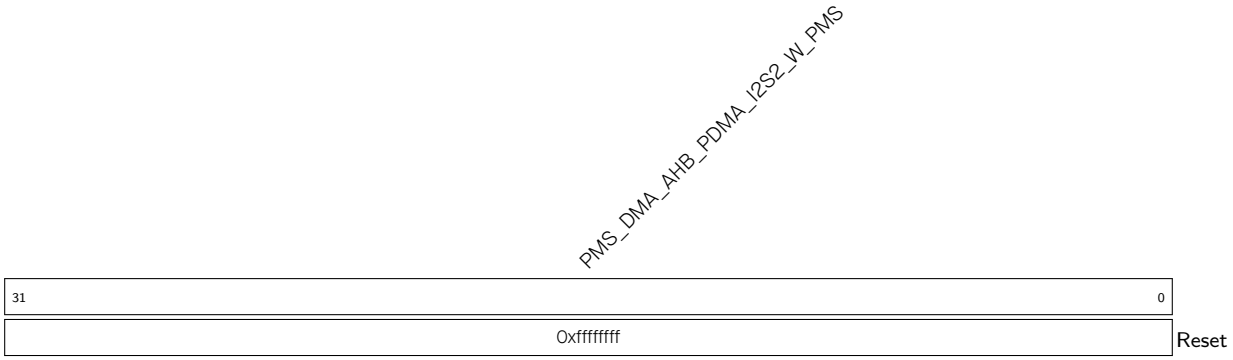
PMS_DMA_AHB_PDMA_I2S2_R_PMS Configures GDMA-AHB permission to read 32 address ranges requested by I2S2. Bit 0 corresponds to region0, and so on.

0: Disable read permission.

1: Enable read permission.

(R/W)

Register 20.14. PMS_DMA_AHB_PDMA_I2S2_W_PMS_REG (0x0144)



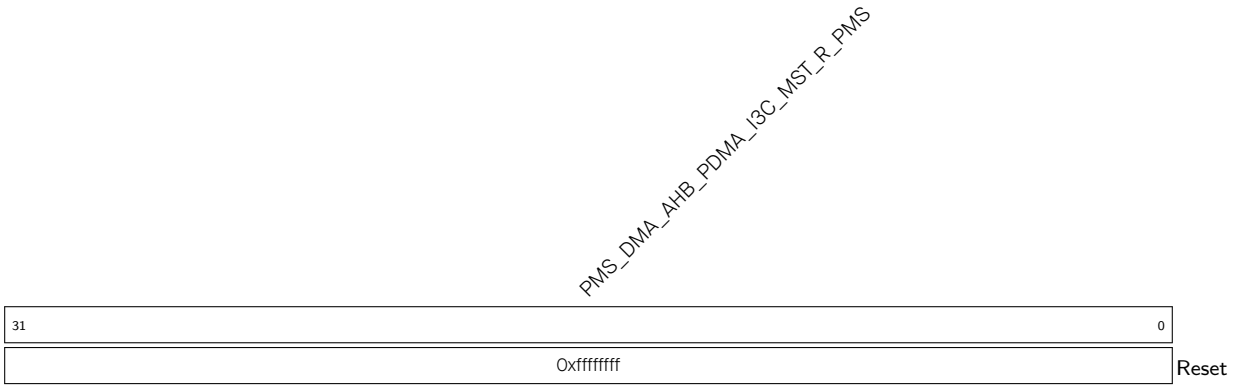
PMS_DMA_AHB_PDMA_I2S2_W_PMS Configures GDMA-AHB permission to write 32 address ranges requested by I2S2. Bit 0 corresponds to region0, and so on.

0: Disable write permission.

1: Enable write permission.

(R/W)

Register 20.15. PMS_DMA_AHB_PDMA_I3C_MST_R_PMS_REG (0x0148)



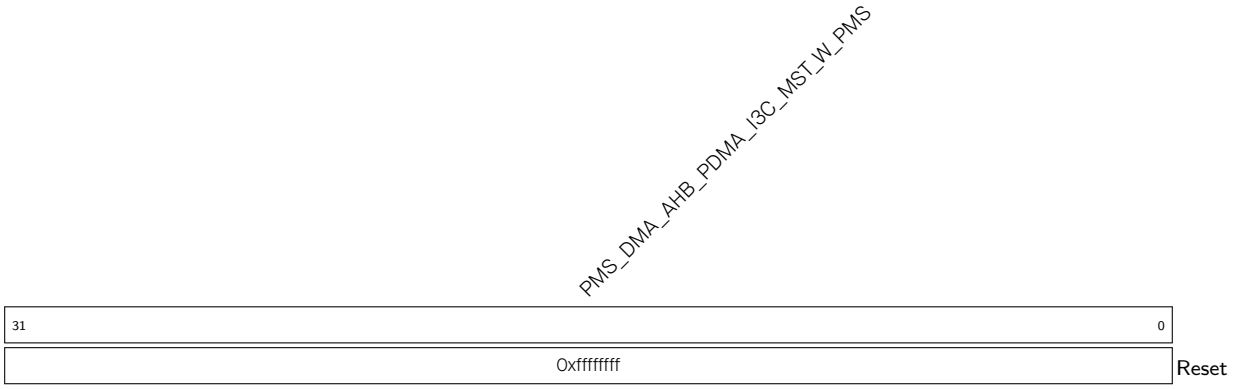
PMS_DMA_AHB_PDMA_I3C_MST_R_PMS Configures GDMA-AHB permission to read 32 address ranges requested by I3C master. Bit 0 corresponds to region0, and so on.

0: Disable read permission.

1: Enable read permission.

(R/W)

Register 20.16. PMS_DMA_AHB_PDMA_I3C_MST_W_PMS_REG (0x014C)



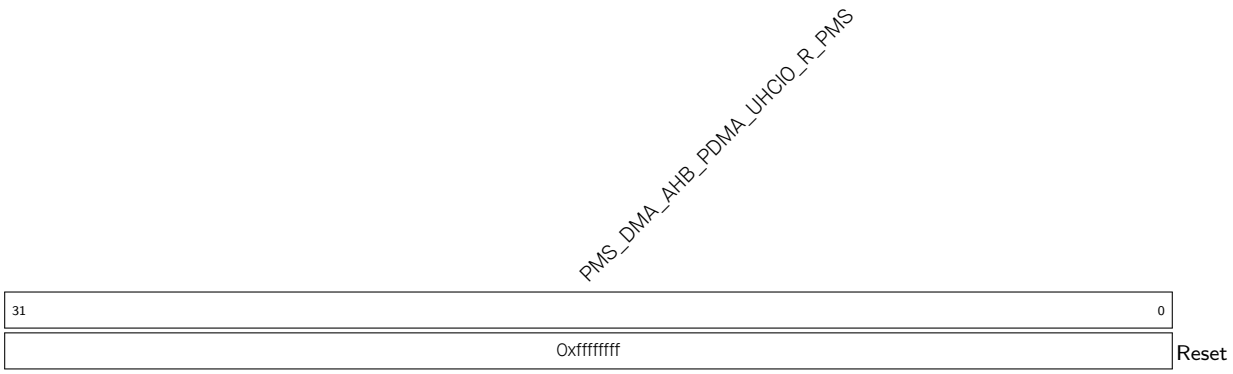
PMS_DMA_AHB_PDMA_I3C_MST_W_PMS Configures GDMA-AHB permission to write 32 address ranges requested by I3C master. Bit 0 corresponds to region0, and so on.

0: Disable write permission.

1: Enable write permission.

(R/W)

Register 20.17. PMS_DMA_AHB_PDMA_UHCI0_R_PMS_REG (0x0150)



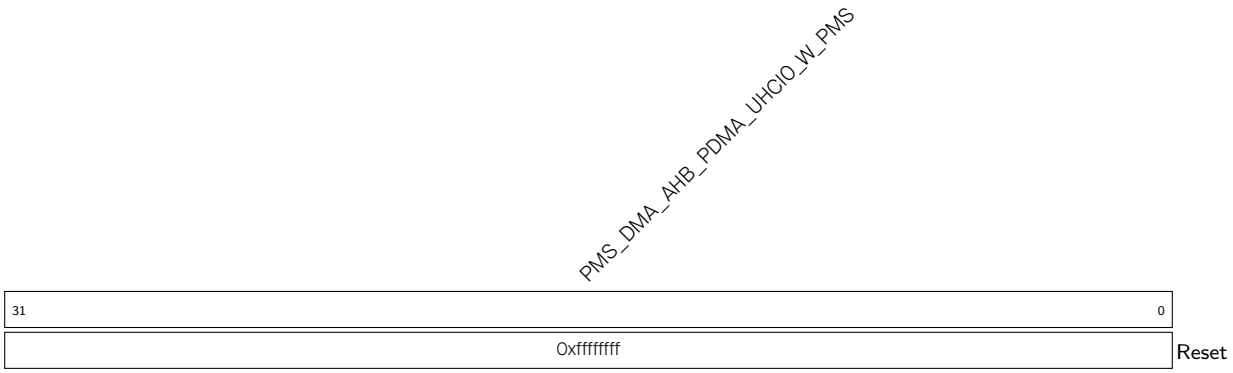
PMS_DMA_AHB_PDMA_UHCI0_R_PMS Configures GDMA-AHB permission to read 32 address ranges requested by UHCI. Bit 0 corresponds to region0, and so on.

0: Disable read permission.

1: Enable read permission.

(R/W)

Register 20.18. PMS_DMA_AHB_PDMA_UHCI0_W_PMS_REG (0x0154)



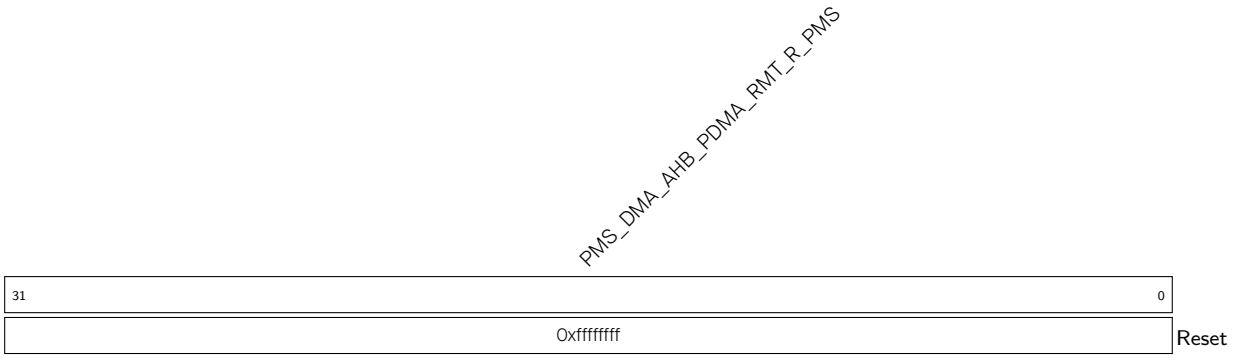
PMS_DMA_AHB_PDMA_UHCI0_W_PMS Configures GDMA-AHB permission to write 32 address ranges requested by UHCI. Bit 0 corresponds to region0, and so on.

0: Disable write permission.

1: Enable write permission.

(R/W)

Register 20.19. PMS_DMA_AHB_PDMA_RMT_R_PMS_REG (0x0158)



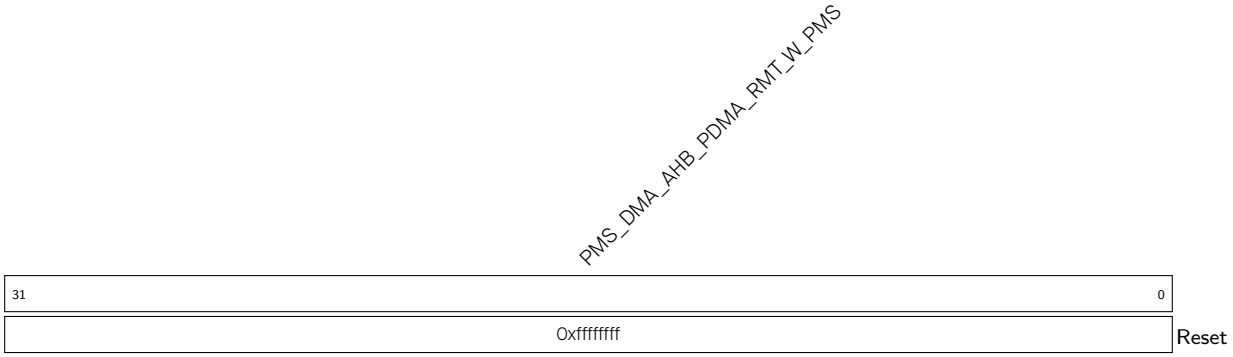
PMS_DMA_AHB_PDMA_RMT_R_PMS Configures GDMA-AHB permission to read 32 address ranges requested by RMT. Bit 0 corresponds to region0, and so on.

0: Disable read permission.

1: Enable read permission.

(R/W)

Register 20.20. PMS_DMA_AHB_PDMA_RMT_W_PMS_REG (0x0170)



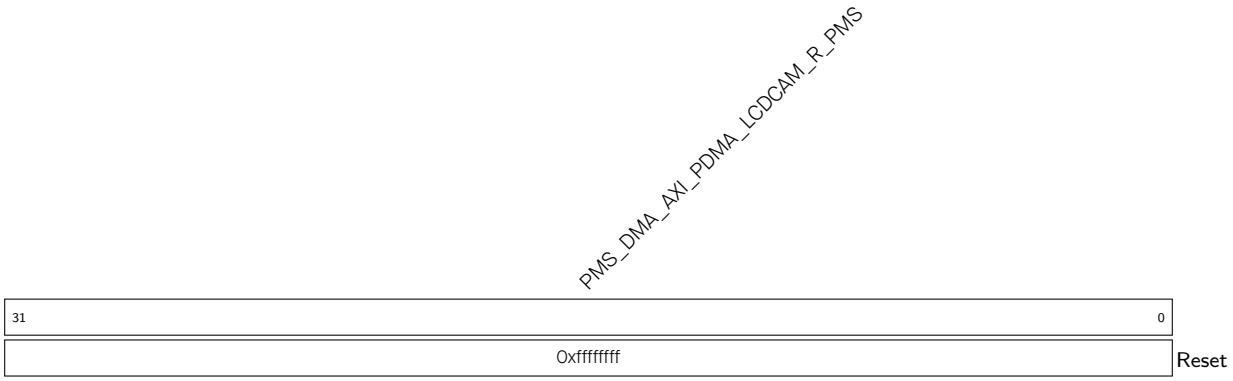
PMS_DMA_AHB_PDMA_RMT_W_PMS Configures GDMA-AHB permission to write 32 address ranges requested by RMT. Bit 0 corresponds to region0, and so on.

0: Disable write permission.

1: Enable write permission.

(R/W)

Register 20.21. PMS_DMA_AXI_PDMA_LCDCAM_R_PMS_REG (0x0174)



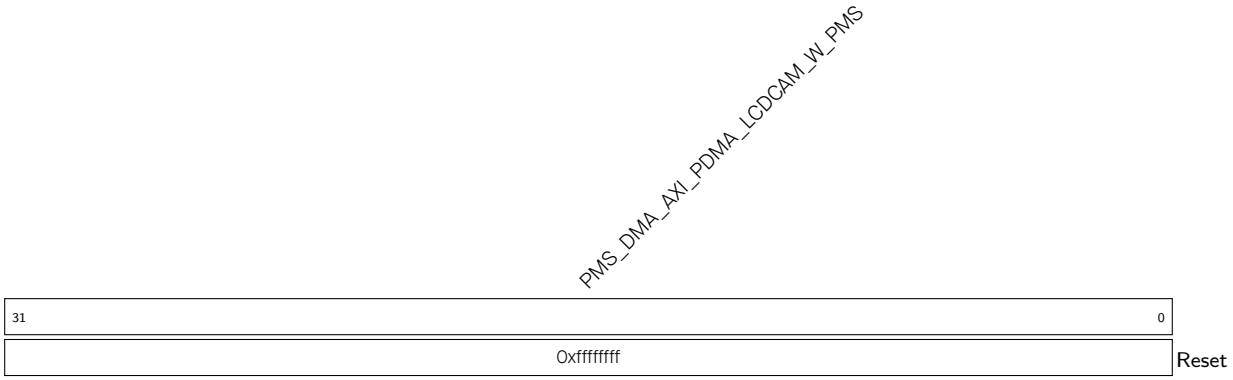
PMS_DMA_AXI_PDMA_LCDCAM_R_PMS Configures GDMA-AXI permission to read 32 address ranges requested by LCD_CAM. Bit 0 corresponds to region0, and so on.

0: Disable read permission.

1: Enable read permission.

(R/W)

Register 20.22. PMS_DMA_AXI_PDMA_LCDCAM_W_PMS_REG (0x0178)



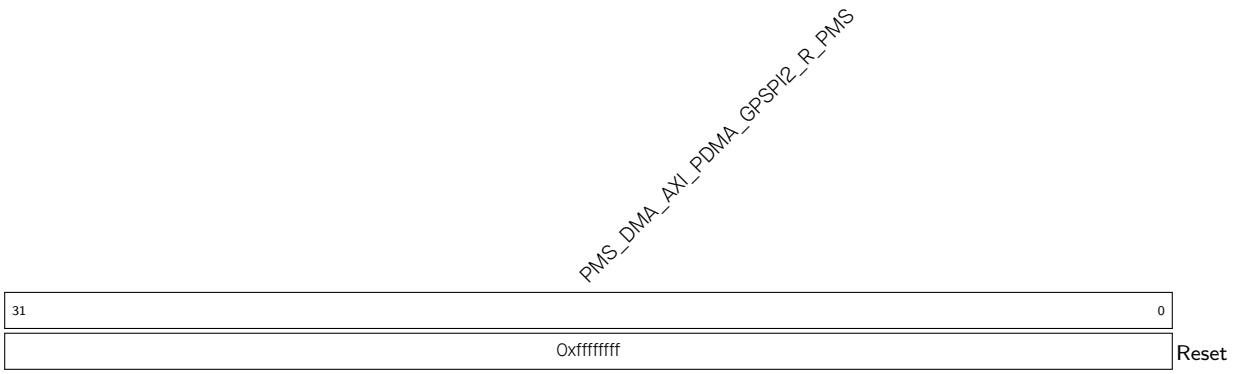
PMS_DMA_AXI_PDMA_LCDCAM_W_PMS Configures GDMA-AXI permission to write 32 address ranges requested by LCD_CAM. Bit 0 corresponds to region0, and so on.

0: Disable write permission.

1: Enable write permission.

(R/W)

Register 20.23. PMS_DMA_AXI_PDMA_GPSPI2_R_PMS_REG (0x017C)



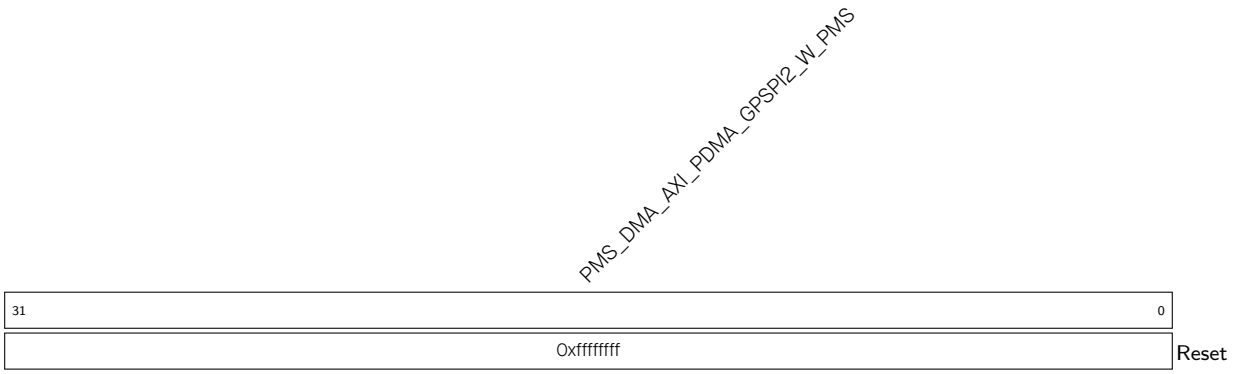
PMS_DMA_AXI_PDMA_GPSPI2_R_PMS Configures GDMA-AXI permission to read 32 address ranges requested by GP-SPI2. Bit 0 corresponds to region0, and so on.

0: Disable read permission.

1: Enable read permission.

(R/W)

Register 20.24. PMS_DMA_AXI_PDMA_GPSPI2_W_PMS_REG (0x0180)



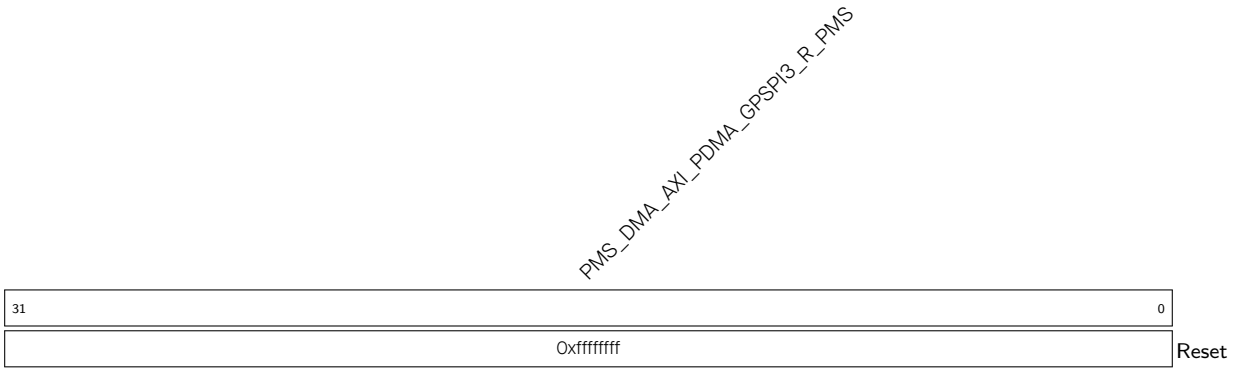
PMS_DMA_AXI_PDMA_GPSPI2_W_PMS Configures GDMA-AXI permission to write 32 address ranges requested by GP-SPI2. Bit 0 corresponds to region0, and so on.

0: Disable write permission.

1: Enable write permission.

(R/W)

Register 20.25. PMS_DMA_AXI_PDMA_GPSPI3_R_PMS_REG (0x0184)



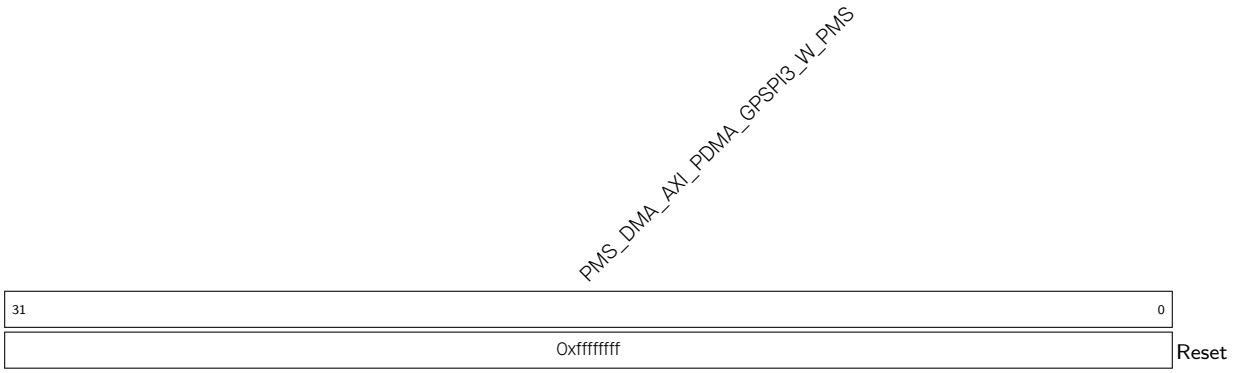
PMS_DMA_AXI_PDMA_GPSPI3_R_PMS Configures GDMA-AXI permission to read 32 address ranges requested by GP-SPI3. Bit 0 corresponds to region0, and so on.

0: Disable read permission.

1: Enable read permission.

(R/W)

Register 20.26. PMS_DMA_AXI_PDMA_GPSPI3_W_PMS_REG (0x0188)



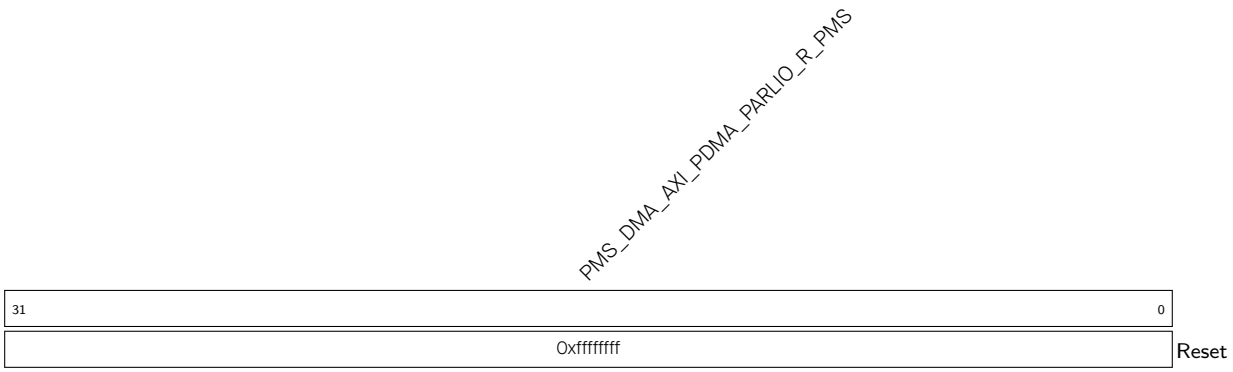
PMS_DMA_AXI_PDMA_GPSPI3_W_PMS Configures GDMA-AXI permission to write 32 address ranges requested by GP-SPI3. Bit 0 corresponds to region0, and so on.

0: Disable write permission.

1: Enable write permission.

(R/W)

Register 20.27. PMS_DMA_AXI_PDMA_PARLIO_R_PMS_REG (0x018C)



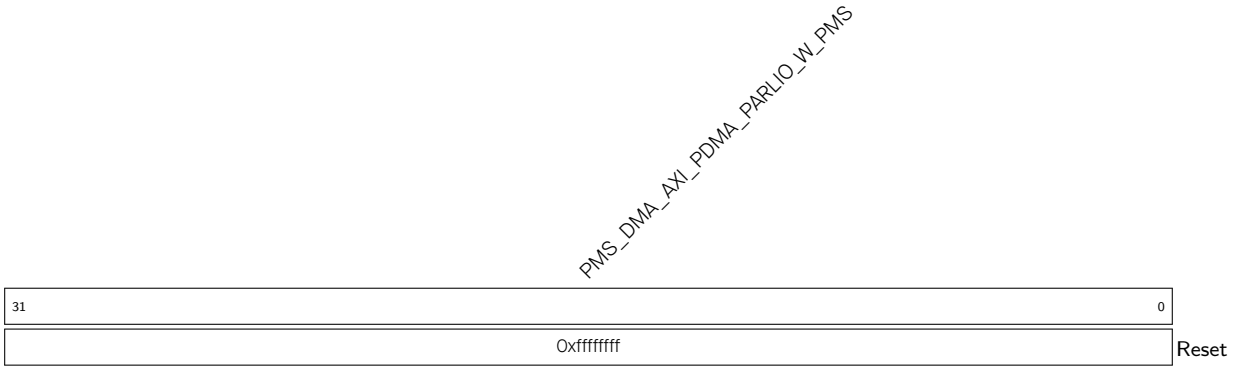
PMS_DMA_AXI_PDMA_PARLIO_R_PMS Configures GDMA-AXI permission to read 32 address ranges requested by PARLIO (Parallel IO Controller). Bit 0 corresponds to region0, and so on.

0: Disable read permission.

1: Enable read permission.

(R/W)

Register 20.28. PMS_DMA_AXI_PDMA_PARLIO_W_PMS_REG (0x0190)



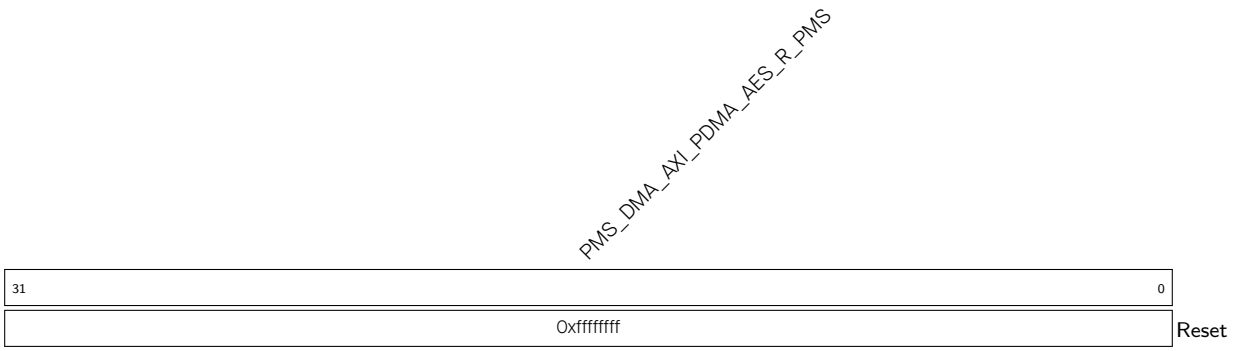
PMS_DMA_AXI_PDMA_PARLIO_W_PMS Configures GDMA-AXI permission to write 32 address ranges requested by PARLIO. Bit 0 corresponds to region0, and so on.

0: Disable write permission.

1: Enable write permission.

(R/W)

Register 20.29. PMS_DMA_AXI_PDMA_AES_R_PMS_REG (0x0194)



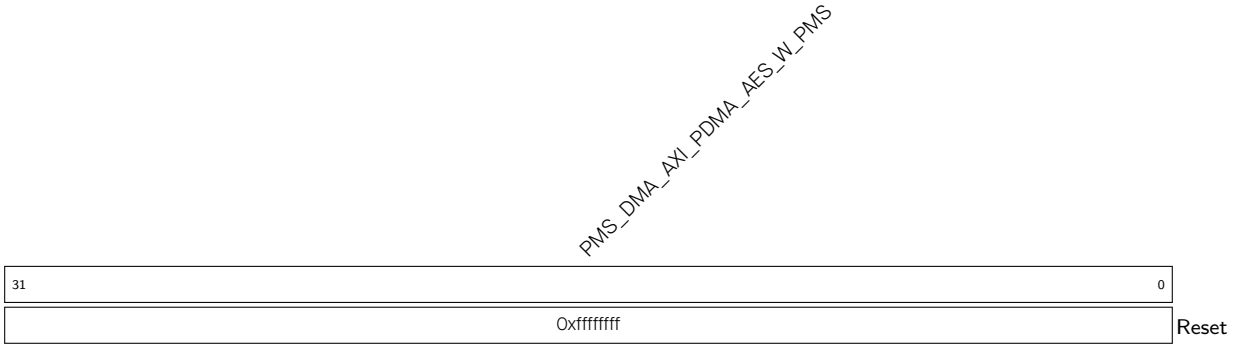
PMS_DMA_AXI_PDMA_AES_R_PMS Configures GDMA-AXI permission to read 32 address ranges requested by AES. Bit 0 corresponds to region0, and so on.

0: Disable read permission.

1: Enable read permission.

(R/W)

Register 20.30. PMS_DMA_AXI_PDMA_AES_W_PMS_REG (0x0198)



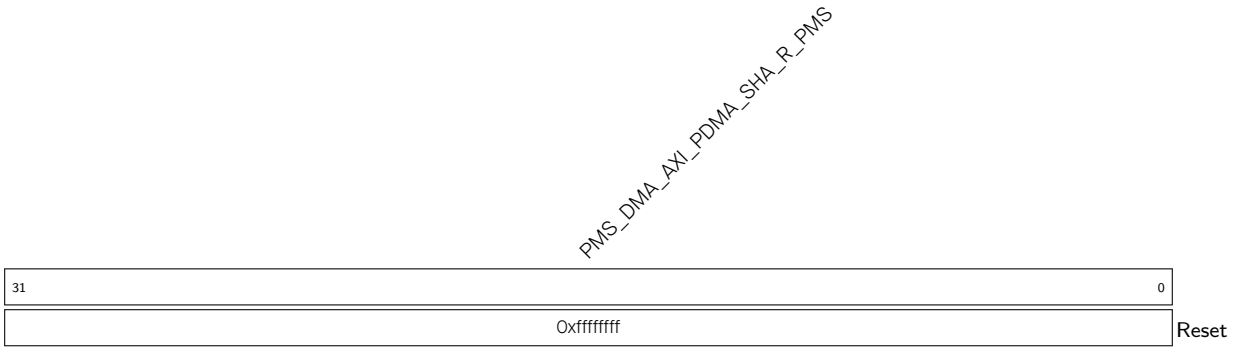
PMS_DMA_AXI_PDMA_AES_W_PMS Configures GDMA-AXI permission to write 32 address ranges requested by AES. Bit 0 corresponds to region0, and so on.

0: Disable write permission.

1: Enable write permission.

(R/W)

Register 20.31. PMS_DMA_AXI_PDMA_SHA_R_PMS_REG (0x019C)



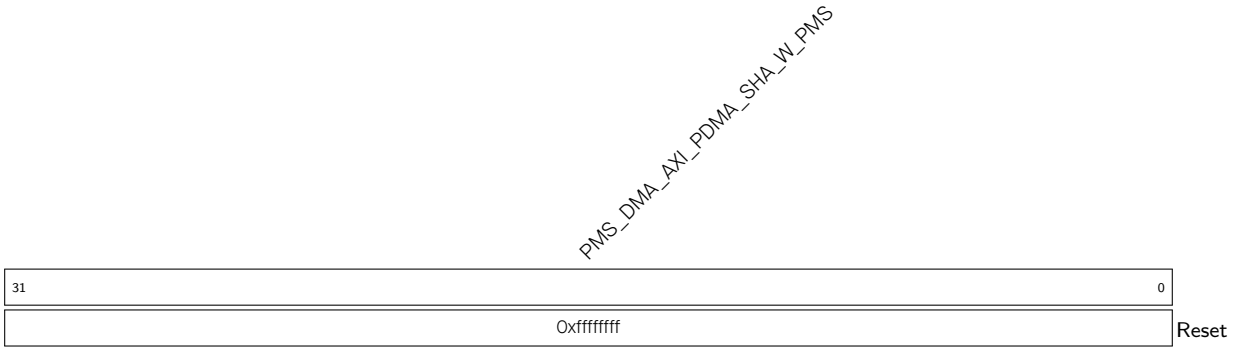
PMS_DMA_AXI_PDMA_SHA_R_PMS Configures GDMA-AXI permission to read 32 address ranges requested by SHA. Bit 0 corresponds to region0, and so on.

0: Disable read permission.

1: Enable read permission.

(R/W)

Register 20.32. PMS_DMA_AXI_PDMA_SHA_W_PMS_REG (0x01A0)



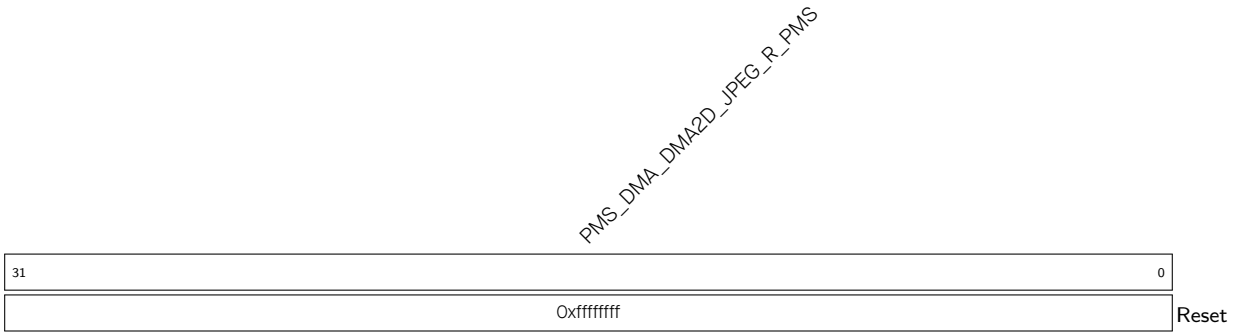
PMS_DMA_AXI_PDMA_SHA_W_PMS Configures GDMA-AXI permission to write 32 address ranges requested by SHA. Bit 0 corresponds to region0, and so on.

0: Disable write permission.

1: Enable write permission.

(R/W)

Register 20.33. PMS_DMA_DMA2D_JPEG_PMS_R_REG (0x01A4)



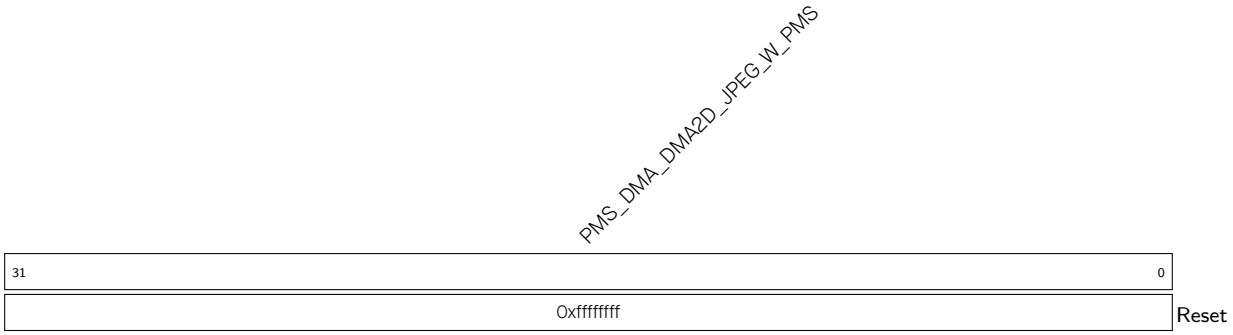
PMS_DMA_DMA2D_JPEG_R_PMS Configures 2D-DMA permission to read 32 address ranges requested by JPEG. Bit 0 corresponds to region0, and so on.

0: Disable read permission.

1: Enable read permission.

(R/W)

Register 20.34. PMS_DMA_DMA2D_JPEG_PMS_W_REG (0x01A8)



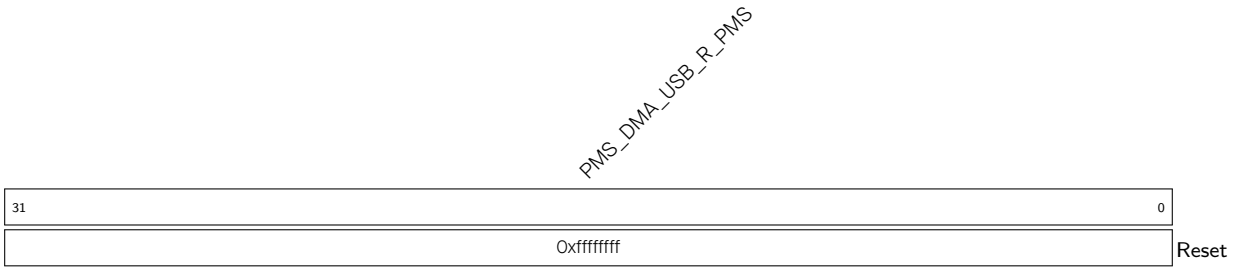
PMS_DMA_DMA2D_JPEG_W_PMS Configures 2D-DMA permission to write 32 address ranges requested by JPEG. Bit 0 corresponds to region0, and so on.

0: Disable write permission.

1: Enable write permission.

(R/W)

Register 20.35. PMS_DMA_USB_PMS_R_REG (0x01AC)



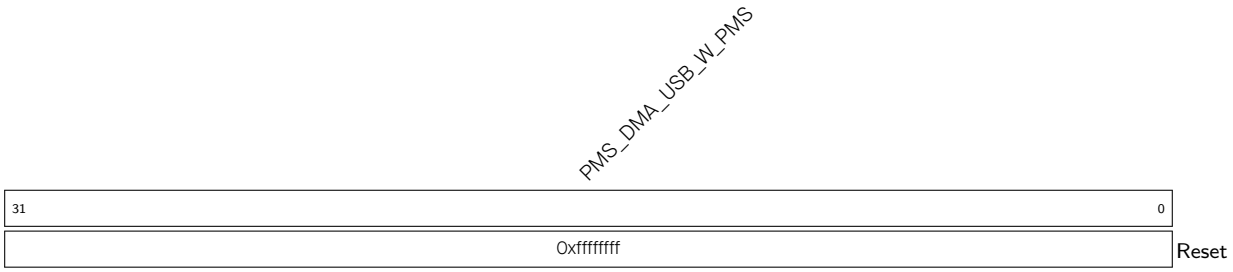
PMS_DMA_USB_R_PMS Configures read permission for USB 2.0 High-Speed OTG to access 32 address ranges. Bit 0 corresponds to region0, and so on.

0: Disable read permission.

1: Enable read permission.

(R/W)

Register 20.36. PMS_DMA_USB_PMS_W_REG (0x01B0)



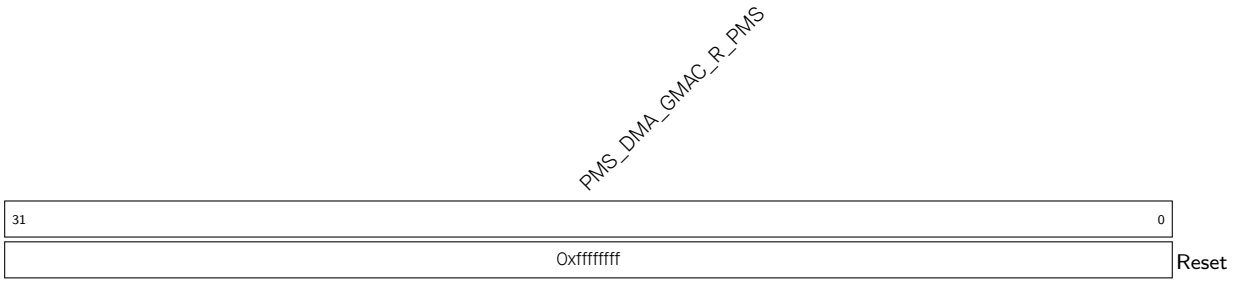
PMS_DMA_USB_W_PMS Configures write permission for USB 2.0 High-Speed OTG to access 32 address ranges. Bit 0 corresponds to region0, and so on.

0: Disable write permission.

1: Enable write permission.

(R/W)

Register 20.37. PMS_DMA_GMAC_PMS_R_REG (0x01B4)



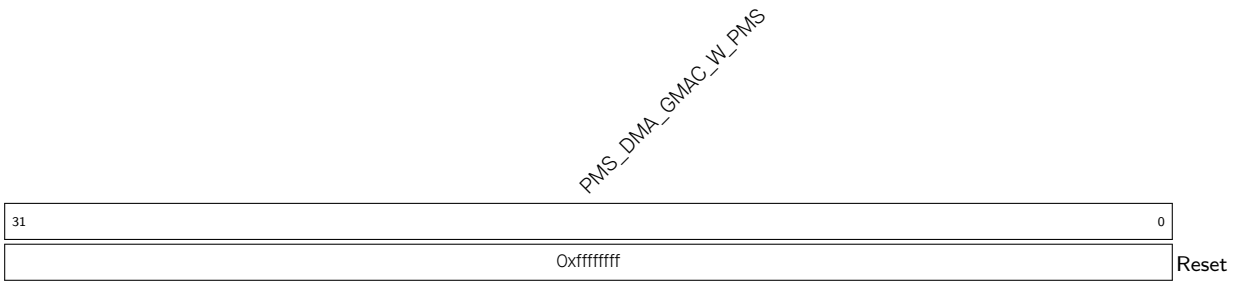
PMS_DMA_GMAC_R_PMS Configures read permission for EMAC to access 32 address ranges. Bit 0 corresponds to region0, and so on.

0: Disable read permission.

1: Enable read permission.

(R/W)

Register 20.38. PMS_DMA_GMAC_PMS_W_REG (0x01B8)



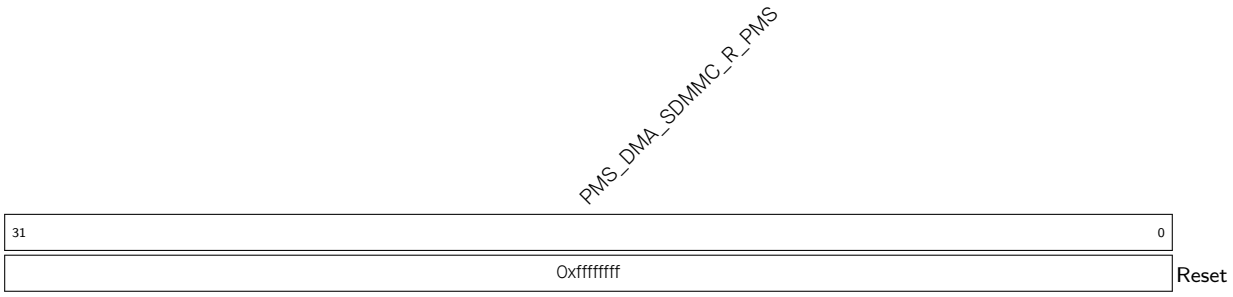
PMS_DMA_GMAC_W_PMS Configures write permission for EMAC to access 32 address ranges. Bit 0 corresponds to region0, and so on.

0: Disable write permission.

1: Enable write permission.

(R/W)

Register 20.39. PMS_DMA_SDMMC_PMS_R_REG (0x01BC)



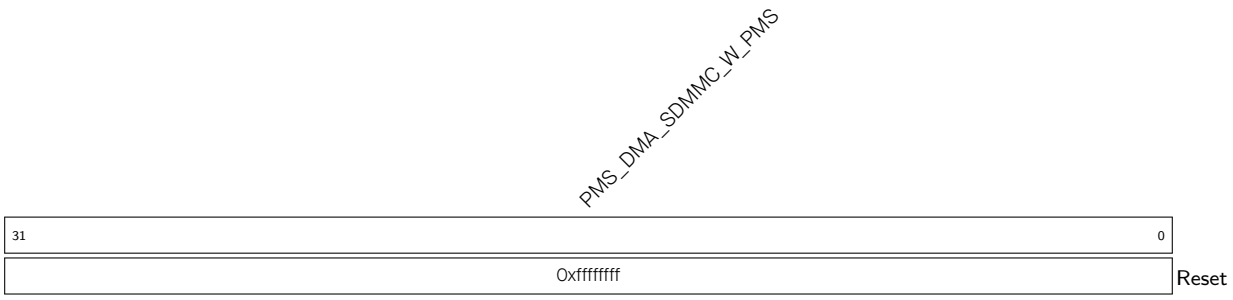
PMS_DMA_SDMMC_R_PMS Configures read permission for SDMMC to access 32 address ranges. Bit 0 corresponds to region0, and so on.

0: Disable read permission.

1: Enable read permission.

(R/W)

Register 20.40. PMS_DMA_SDMMC_PMS_W_REG (0x01C0)



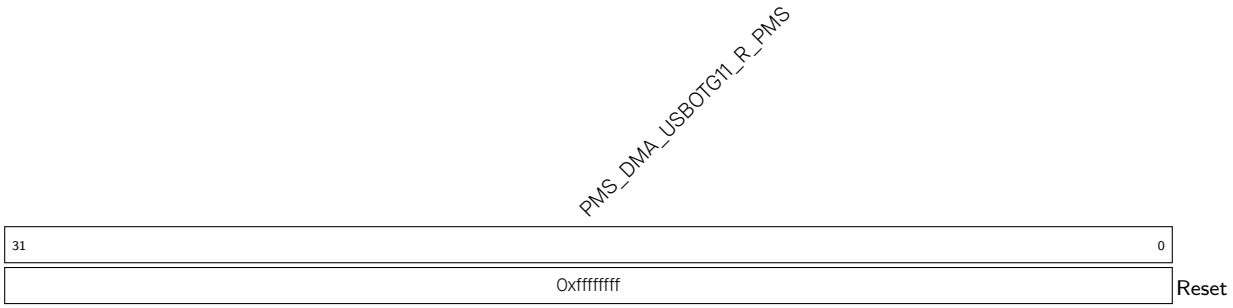
PMS_DMA_SDMMC_W_PMS Configures write permission for SDMMC to access 32 address ranges. Bit 0 corresponds to region0, and so on.

0: Disable write permission.

1: Enable write permission.

(R/W)

Register 20.41. PMS_DMA_USBOTG11_PMS_R_REG (0x01C4)



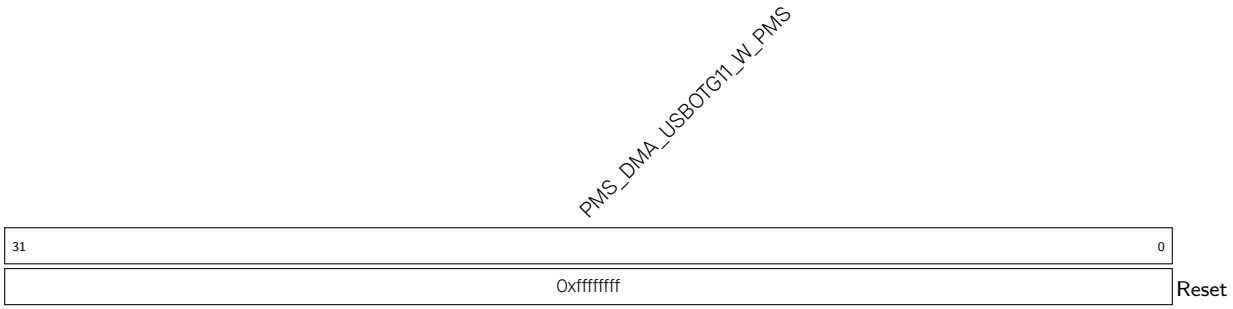
PMS_DMA_USBOTG11_R_PMS Configures read permission for USB 2.0 Full-Speed OTG to access 32 address ranges. Bit 0 corresponds to region0, and so on.

0: Disable read permission.

1: Enable read permission.

(R/W)

Register 20.42. PMS_DMA_USBOTG11_PMS_W_REG (0x01C8)



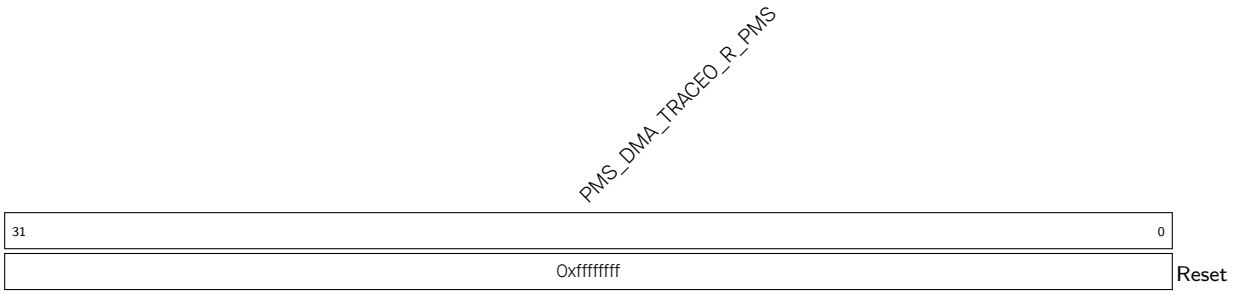
PMS_DMA_USBOTG11_W_PMS Configures write permission for USB 2.0 Full-Speed OTG to access 32 address ranges. Bit 0 corresponds to region0, and so on.

0: Disable write permission.

1: Enable write permission.

(R/W)

Register 20.43. PMS_DMA_TRACE0_PMS_R_REG (0x01CC)



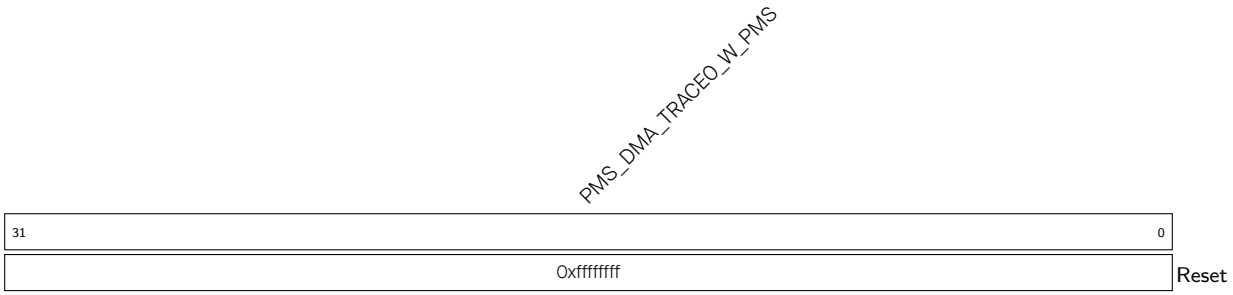
PMS_DMA_TRACE0_R_PMS Configures read permission for TRACE0 to access 32 address ranges. Bit 0 corresponds to region0, and so on.

0: Disable read permission.

1: Enable read permission.

(R/W)

Register 20.44. PMS_DMA_TRACE0_PMS_W_REG (0x01D0)



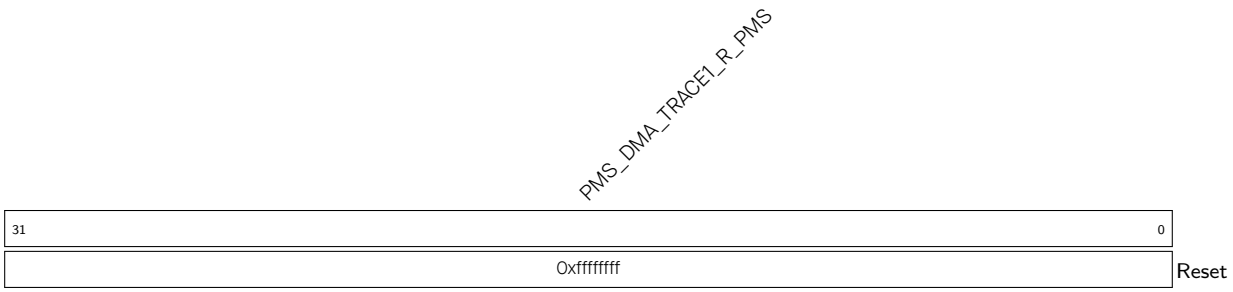
PMS_DMA_TRACE0_W_PMS Configures write permission for TRACE0 to access 32 address ranges. Bit 0 corresponds to region0, and so on.

0: Disable write permission.

1: Enable write permission.

(R/W)

Register 20.45. PMS_DMA_TRACE1_PMS_R_REG (0x01D4)



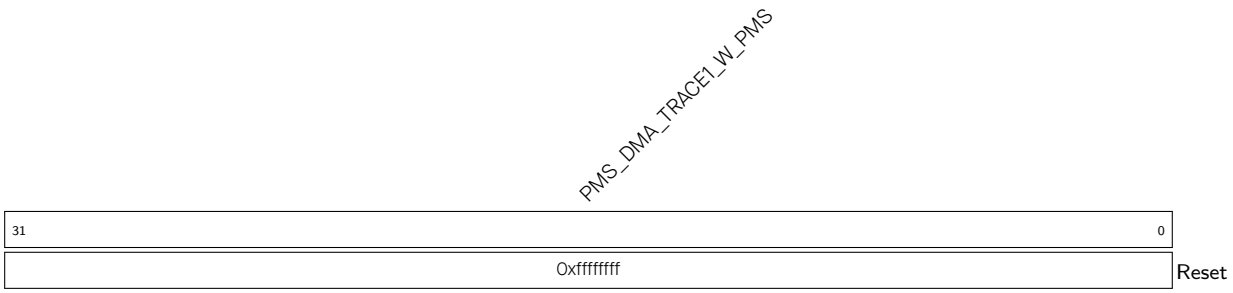
PMS_DMA_TRACE1_R_PMS Configures read permission for TRACE1 to access 32 address ranges. Bit 0 corresponds to region0, and so on.

0: Disable read permission.

1: Enable read permission.

(R/W)

Register 20.46. PMS_DMA_TRACE1_PMS_W_REG (0x01D8)



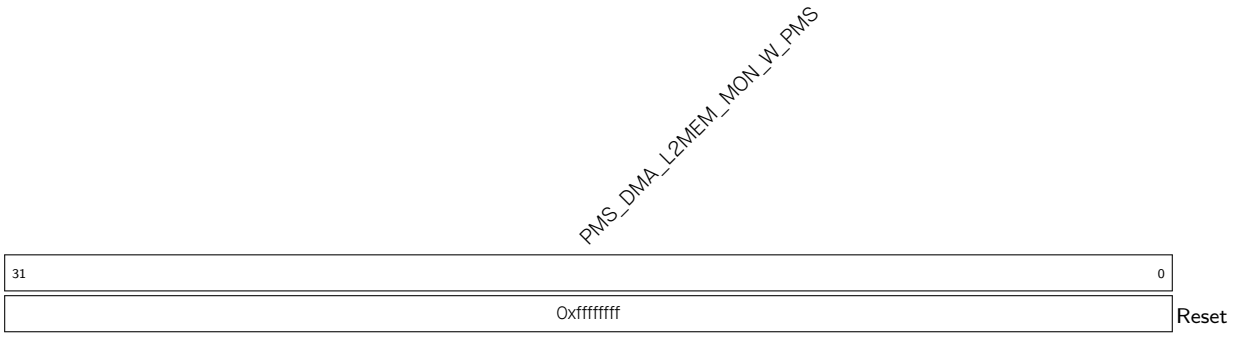
PMS_DMA_TRACE1_W_PMS Configures write permission for TRACE1 to access 32 address ranges. Bit 0 corresponds to region0, and so on.

0: Disable write permission.

1: Enable write permission.

(R/W)

Register 20.47. PMS_DMA_L2MEM_MON_PMS_W_REG (0x01E0)



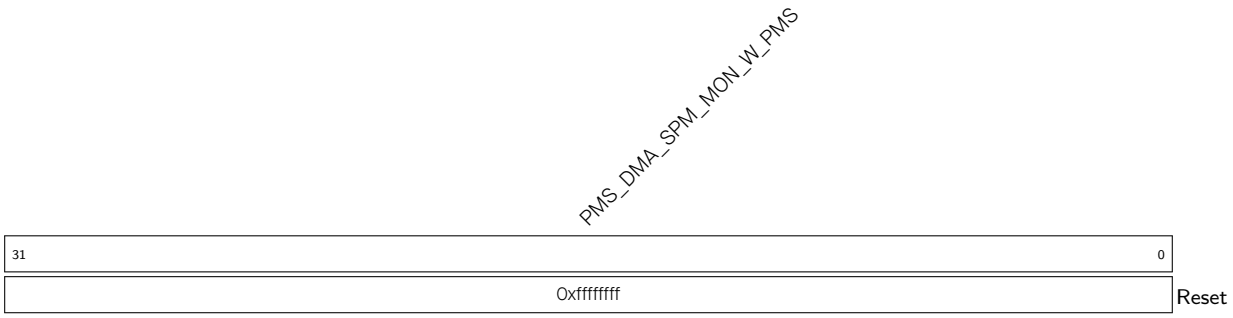
PMS_DMA_L2MEM_MON_W_PMS Configures write permission for L2MEM monitor to access 32 address ranges. Bit 0 corresponds to region0, and so on.

0: Disable write permission.

1: Enable write permission.

(R/W)

Register 20.48. PMS_DMA_SPM_MON_PMS_W_REG (0x01E8)



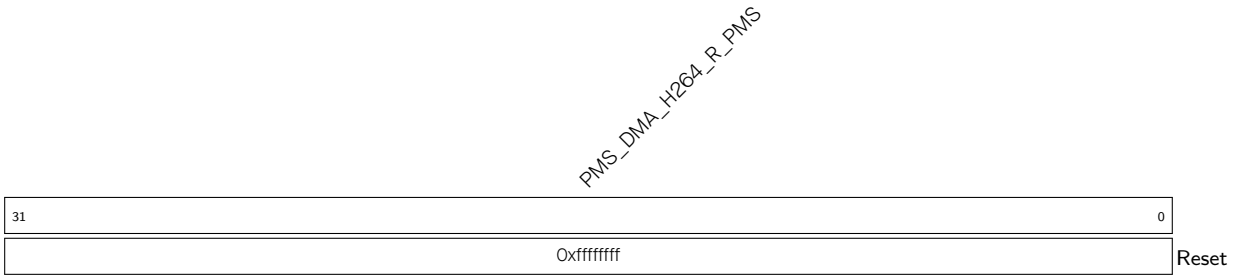
PMS_DMA_SPM_MON_W_PMS Configures write permission for SPM monitor to access 32 address ranges. Bit 0 corresponds to region0, and so on.

0: Disable write permission.

1: Enable write permission.

(R/W)

Register 20.49. PMS_DMA_H264_PMS_R_REG (0x01FC)



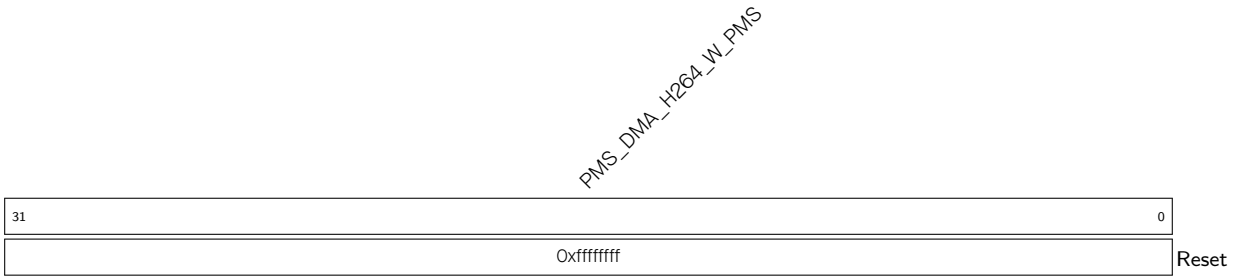
PMS_DMA_H264_R_PMS Configures read permission for H264 DMA to access 32 address ranges. Bit 0 corresponds to region0, and so on.

0: Disable read permission.

1: Enable read permission.

(R/W)

Register 20.50. PMS_DMA_H264_PMS_W_REG (0x0200)



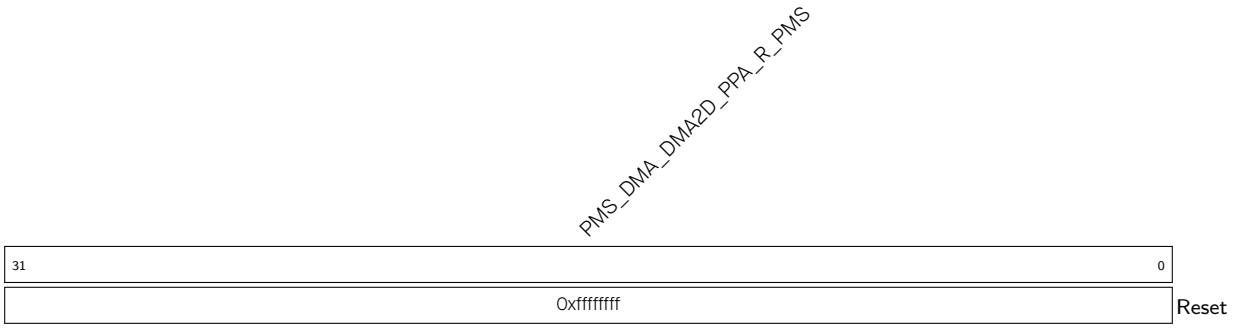
PMS_DMA_H264_W_PMS Configures write permission for H264 DMA to access 32 address ranges. Bit 0 corresponds to region0, and so on.

0: Disable write permission.

1: Enable write permission.

(R/W)

Register 20.51. PMS_DMA_DMA2D_PPA_PMS_R_REG (0x0204)



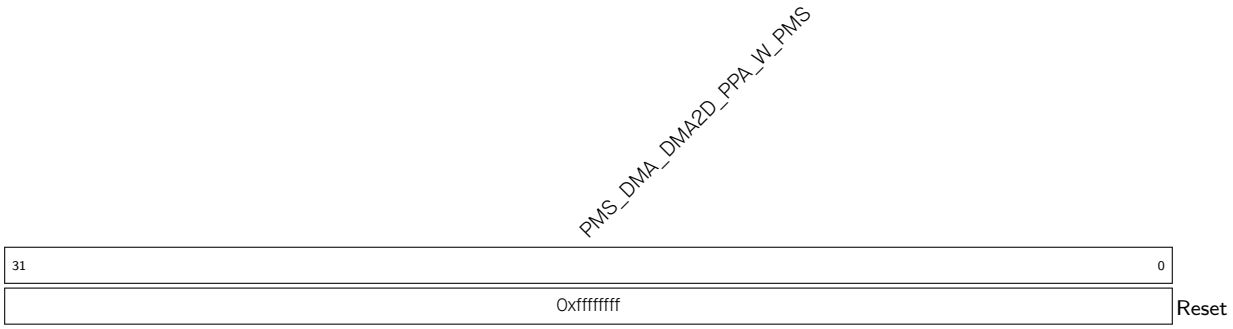
PMS_DMA_DMA2D_PPA_R_PMS Configures 2D-DMA permission to read 32 address ranges requested by PPA (Pixel-Processing Accelerator). Bit 0 corresponds to region0, and so on.

0: Disable read permission.

1: Enable read permission.

(R/W)

Register 20.52. PMS_DMA_DMA2D_PPA_PMS_W_REG (0x0208)



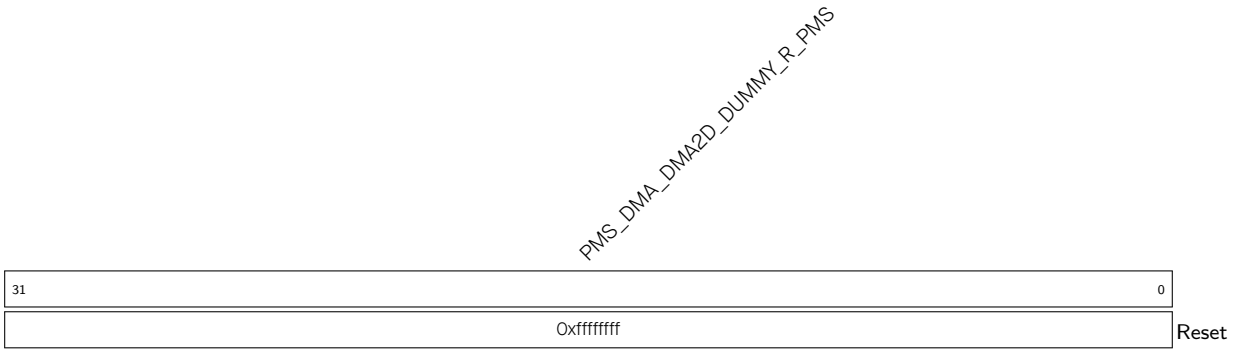
PMS_DMA_DMA2D_PPA_W_PMS Configures 2D-DMA permission to write 32 address ranges requested by PPA. Bit 0 corresponds to region0, and so on.

0: Disable write permission.

1: Enable write permission.

(R/W)

Register 20.53. PMS_DMA_DMA2D_DUMMY_PMS_R_REG (0x020C)



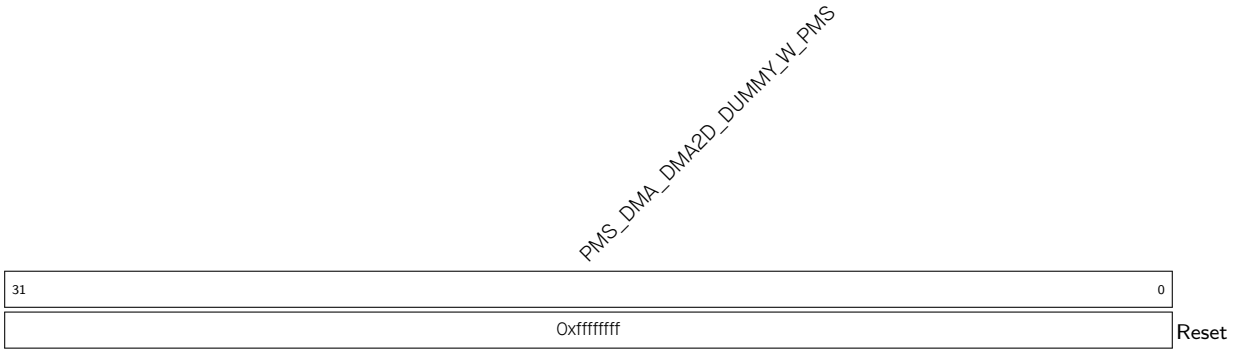
PMS_DMA_DMA2D_DUMMY_R_PMS Configures 2D-DMA permission to read 32 address ranges requested by Dummy. Bit 0 corresponds to region0, and so on.

0: Disable read permission.

1: Enable read permission.

(R/W)

Register 20.54. PMS_DMA_DMA2D_DUMMY_PMS_W_REG (0x0210)



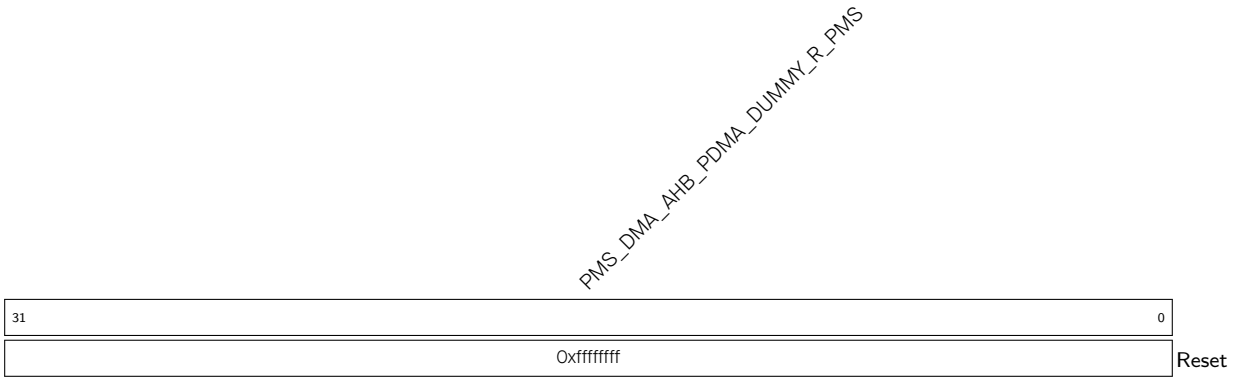
PMS_DMA_DMA2D_DUMMY_W_PMS Configures 2D-DMA permission to write 32 address ranges requested by Dummy. Bit 0 corresponds to region0, and so on.

0: Disable write permission.

1: Enable write permission.

(R/W)

Register 20.55. PMS_DMA_AHB_PDMA_DUMMY_R_PMS_REG (0x0214)



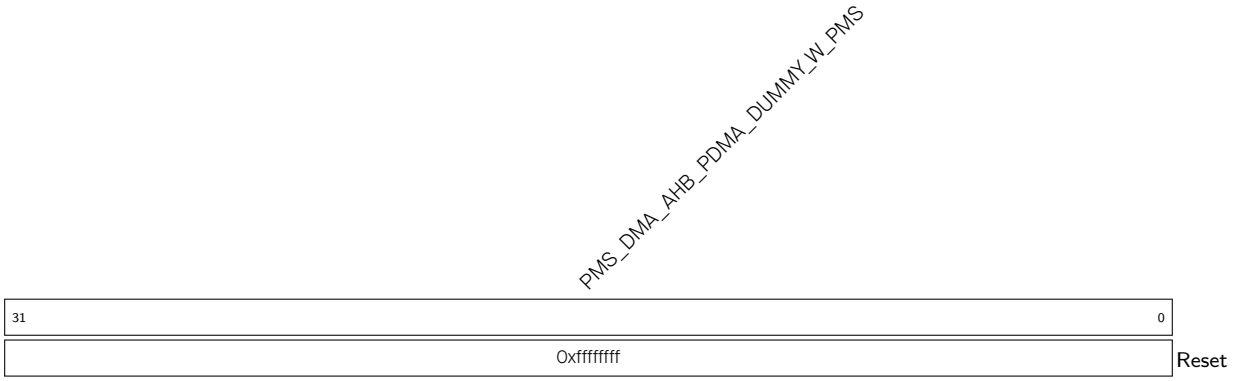
PMS_DMA_AHB_PDMA_DUMMY_R_PMS Configures GDMA-AHB permission to read 32 address ranges requested by Dummy. Bit 0 corresponds to region0, and so on.

0: Disable read permission.

1: Enable read permission.

(R/W)

Register 20.56. PMS_DMA_AHB_PDMA_DUMMY_W_PMS_REG (0x0218)



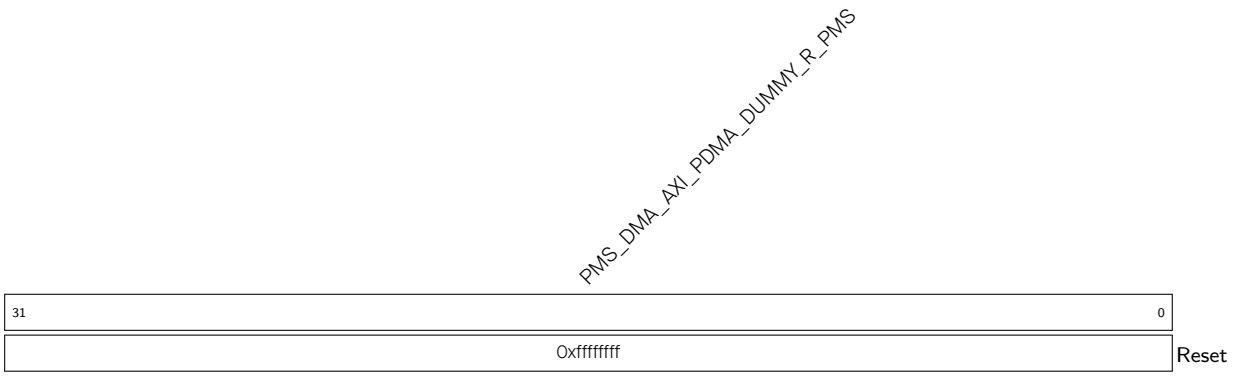
PMS_DMA_AHB_PDMA_DUMMY_W_PMS Configures GDMA-AHB permission to write 32 address ranges requested by Dummy. Bit 0 corresponds to region0, and so on.

0: Disable write permission.

1: Enable write permission.

(R/W)

Register 20.57. PMS_DMA_AXI_PDMA_DUMMY_R_PMS_REG (0x021C)



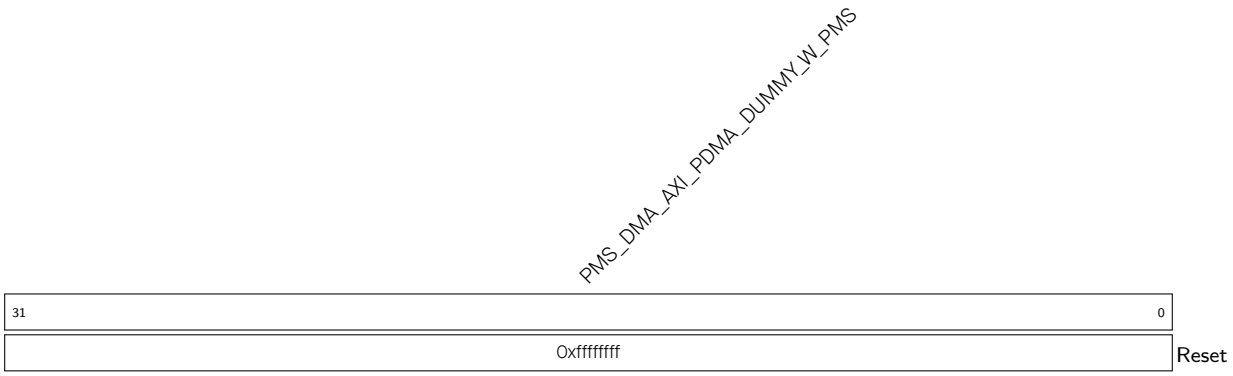
PMS_DMA_AXI_PDMA_DUMMY_R_PMS Configures GDMA-AXI permission to read 32 address ranges requested by Dummy. Bit 0 corresponds to region0, and so on.

0: Disable read permission.

1: Enable read permission.

(R/W)

Register 20.58. PMS_DMA_AXI_PDMA_DUMMY_W_PMS_REG (0x0220)



PMS_DMA_AXI_PDMA_DUMMY_W_PMS Configures GDMA-AXI permission to write 32 address ranges requested by Dummy. Bit 0 corresponds to region0, and so on.

0: Disable write permission.

1: Enable write permission.

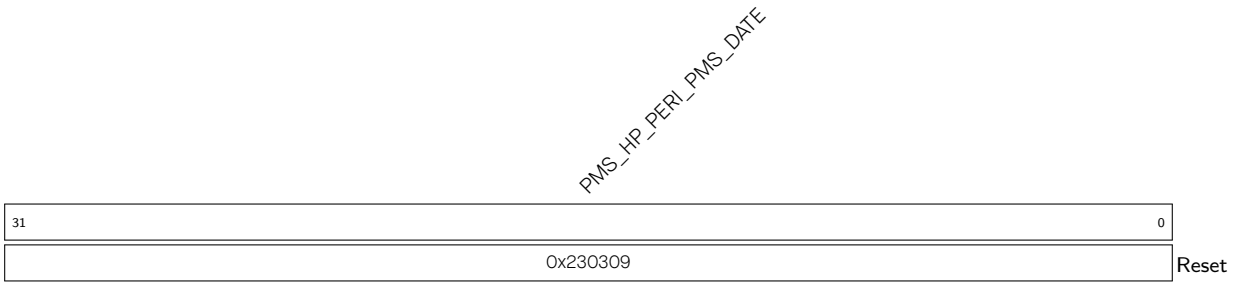
(R/W)

20.7.2 HP_PERI_PMS_REG

The addresses in this section are relative to the HP_PERI_PMS base address provided in Table 9.3-2 in Chapter 9 *System and Memory*.

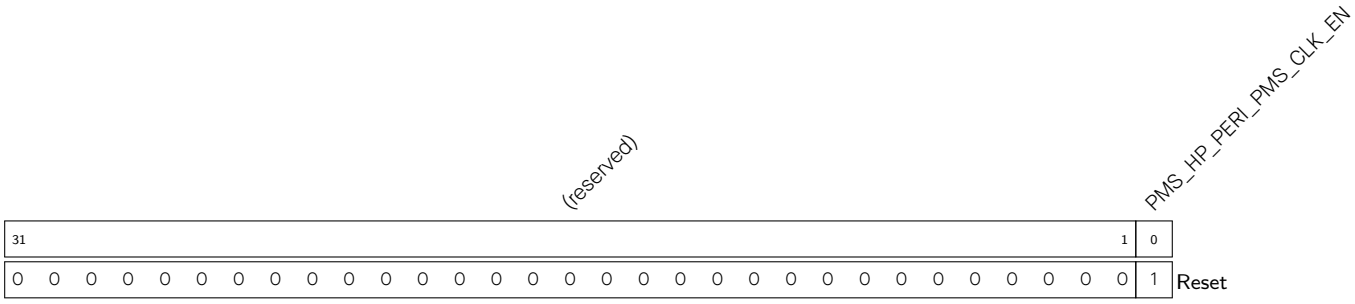
For how to program reserved fields, please refer to Section Programming Reserved Register Field.

Register 20.59. PMS_HP_PERI_PMS_DATE_REG (0x0000)



PMS_HP_PERI_PMS_DATE Version control register. (R/W)

Register 20.60. PMS_HP_PERI_PMS_CLK_EN_REG (0x0004)



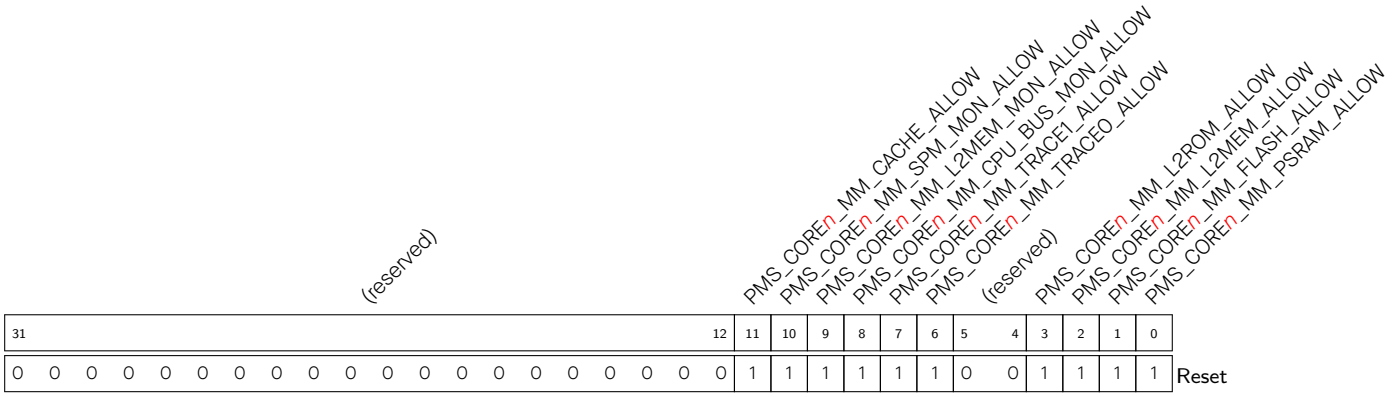
PMS_HP_PERI_PMS_CLK_EN Configures whether to keep the clock always on.

0: Enable automatic clock gating

1: Keep the clock always on

(R/W)

Register 20.61. PMS_CORE*n*_MM_HP_PERI_PMS_REG0_REG (*n*: 0-1) (0x0008+0x20*n)



PMS_CORE*n*_MM_PSRAM_ALLOW Configures whether HP CPU*n* in machine mode has permission to access external RAM without going through cache.

0: Not allowed

1: Allowed

(R/W)

PMS_CORE*n*_MM_FLASH_ALLOW Configures whether HP CPU*n* in machine mode has permission to access external flash without going through cache.

0: Not allowed

1: Allowed

(R/W)

PMS_CORE*n*_MM_L2MEM_ALLOW Configures whether HP CPU*n* in machine mode has permission to access HP L2MEM without going through cache.

0: Not allowed

1: Allowed

(R/W)

PMS_CORE*n*_MM_L2ROM_ALLOW Configures whether HP CPU*n* in machine mode has permission to access HP ROM without going through cache.

0: Not allowed

1: Allowed

(R/W)

PMS_CORE*n*_MM_TRACE0_ALLOW Configures whether HP CPU*n* in machine mode has permission to access TRACE0.

0: Not allowed

1: Allowed

(R/W)

PMS_CORE*n*_MM_TRACE1_ALLOW Configures whether HP CPU*n* in machine mode has permission to access TRACE1.

0: Not allowed

1: Allowed

(R/W)

Continued on the next page...

Register 20.61. PMS_CORE*n*_MM_HP_PERI_PMS_REG0_REG (*n*: 0-1) (0x0008+0x20**n*)

Continued from the previous page...

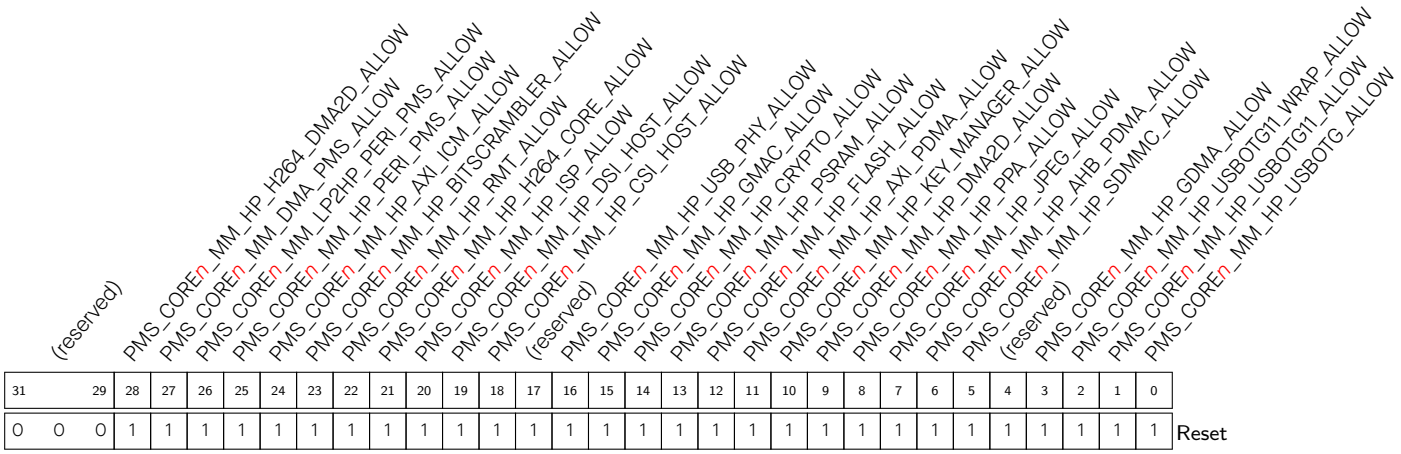
PMS_CORE*n*_MM_CPU_BUS_MON_ALLOW Configures whether HP CPU*n* in machine mode has permission to access CPU bus monitor.
0: Not allowed
1: Allowed
(R/W)

PMS_CORE*n*_MM_L2MEM_MON_ALLOW Configures whether HP CPU*n* in machine mode has permission to access L2MEM monitor.
0: Not allowed
1: Allowed
(R/W)

PMS_CORE*n*_MM_SPM_MON_ALLOW Configures whether HP CPU*n* in machine mode has permission to access SPM monitor.
0: Not allowed
1: Allowed
(R/W)

PMS_CORE*n*_MM_CACHE_ALLOW Configures whether HP CPU*n* in machine mode has permission to access cache.
0: Not allowed
1: Allowed
(R/W)

Register 20.62. PMS_CORE*n*_MM_HP_PERI_PMS_REG1_REG (*n*: 0-1) (0x000C+0x20n*)**



PMS_CORE*n*_MM_HP_USBOTG_ALLOW Configures whether HP CPU*n* in machine mode has permission to access USB 2.0 High-Speed OTG.

0: Not allowed

1: Allowed

(R/W)

PMS_CORE*n*_MM_HP_USBOTG11_ALLOW Configures whether HP CPU*n* in machine mode has permission to access USB 2.0 Full-Speed OTG.

0: Not allowed

1: Allowed

(R/W)

PMS_CORE*n*_MM_HP_USBOTG11_WRAP_ALLOW Configures whether HP CPU*n* in machine mode has permission to access USB 2.0 Full-Speed OTG's wrap.

0: Not allowed

1: Allowed

(R/W)

PMS_CORE*n*_MM_HP_GDMA_ALLOW Configures whether HP CPU*n* in machine mode has permission to access HP VDMA.

0: Not allowed

1: Allowed

(R/W)

PMS_CORE*n*_MM_HP_SDMMC_ALLOW Configures whether HP CPU*n* in machine mode has permission to access HP SDMMC.

0: Not allowed

1: Allowed

(R/W)

Continued on the next page...

Register 20.62. PMS_CORE*n*_MM_HP_PERI_PMS_REG1_REG (*n*: 0-1) (0x000C+0x20**n*)

Continued from the previous page...

PMS_CORE*n*_MM_HP_AHB_PDMA_ALLOW Configures whether HP CPU*n* in machine mode has permission to access GDMA-AHB.
0: Not allowed
1: Allowed
(R/W)

PMS_CORE*n*_MM_HP_JPEG_ALLOW Configures whether HP CPU*n* in machine mode has permission to access HP JPEG.
0: Not allowed
1: Allowed
(R/W)

PMS_CORE*n*_MM_HP_PPA_ALLOW Configures whether HP CPU*n* in machine mode has permission to access HP PPA.
0: Not allowed
1: Allowed
(R/W)

PMS_CORE*n*_MM_HP_DMA2D_ALLOW Configures whether HP CPU*n* in machine mode has permission to access HP 2D-DMA.
0: Not allowed
1: Allowed
(R/W)

PMS_CORE*n*_MM_HP_KEY_MANAGER_ALLOW Configures whether HP CPU*n* in machine mode has permission to access HP Key Manager.
0: Not allowed
1: Allowed
(R/W)

PMS_CORE*n*_MM_HP_AXI_PDMA_ALLOW Configures whether HP CPU*n* in machine mode has permission to access HP GDMA-AXI.
0: Not allowed
1: Allowed
(R/W)

PMS_CORE*n*_MM_HP_FLASH_ALLOW Configures whether HP CPU*n* in machine mode has permission to access HP flash MSPI controller.
0: Not allowed
1: Allowed
(R/W)

Continued on the next page...

Register 20.62. PMS_CORE*n*_MM_HP_PERI_PMS_REG1_REG (*n*: 0-1) (0x000C+0x20**n*)

Continued from the previous page...

PMS_CORE*n*_MM_HP_PSRAM_ALLOW Configures whether HP CPU*n* in machine mode has permission to access HP PSRAM MSPI controller.
0: Not allowed
1: Allowed
(R/W)

PMS_CORE*n*_MM_HP_CRYPTO_ALLOW Configures whether HP CPU*n* in machine mode has permission to access HP CRYPTO (including AES/SHA/RSA/HMAC Accelerators).
0: Not allowed
1: Allowed
(R/W)

PMS_CORE*n*_MM_HP_GMAC_ALLOW Configures whether HP CPU*n* in machine mode has permission to access HP EMAC.
0: Not allowed
1: Allowed
(R/W)

PMS_CORE*n*_MM_HP_USB_PHY_ALLOW Configures whether HP CPU*n* in machine mode has permission to access USB 2.0 High-Speed OTG PHY.
0: Not allowed
1: Allowed
(R/W)

PMS_CORE*n*_MM_HP_CSI_HOST_ALLOW Configures whether HP CPU*n* in machine mode has permission to access HP MIPI CSI host.
0: Not allowed
1: Allowed
(R/W)

PMS_CORE*n*_MM_HP_DSI_HOST_ALLOW Configures whether HP CPU*n* in machine mode has permission to access HP MIPI DSI host.
0: Not allowed
1: Allowed
(R/W)

PMS_CORE*n*_MM_HP_ISP_ALLOW Configures whether HP CPU*n* in machine mode has permission to access HP ISP (Image Signal Processor).
0: Not allowed
1: Allowed
(R/W)

Continued on the next page...

Register 20.62. PMS_CORE*n*_MM_HP_PERI_PMS_REG1_REG (*n*: 0-1) (0x000C+0x20**n*)

Continued from the previous page...

PMS_CORE*n*_MM_HP_H264_CORE_ALLOW Configures whether HP CPU*n* in machine mode has permission to access HP H264 Encoder.
0: Not allowed
1: Allowed
(R/W)

PMS_CORE*n*_MM_HP_RMT_ALLOW Configures whether HP CPU*n* in machine mode has permission to access HP RMT.
0: Not allowed
1: Allowed
(R/W)

PMS_CORE*n*_MM_HP_BITSCRAMBLER_ALLOW Configures whether HP CPU*n* in machine mode has permission to access HP bit scrambler.
0: Not allowed
1: Allowed
(R/W)

PMS_CORE*n*_MM_HP_AXI_ICM_ALLOW Configures whether HP CPU*n* in machine mode has permission to access HP AXI ICM.
0: Not allowed
1: Allowed
(R/W)

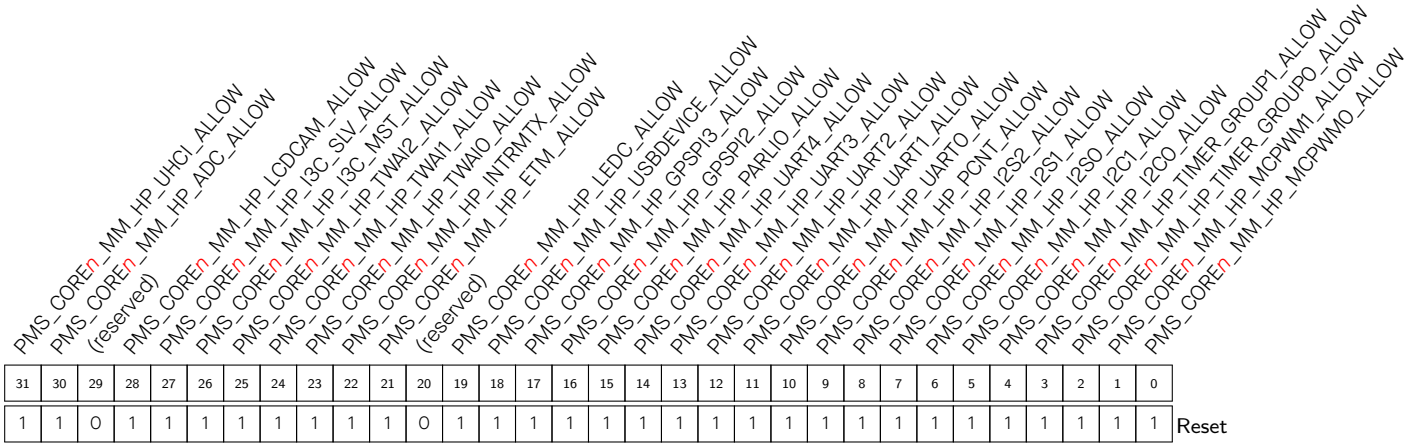
PMS_CORE*n*_MM_HP_PERI_PMS_ALLOW Configures whether HP CPU*n* in machine mode has permission to access HP_PERI_PMS_REG.
0: Not allowed
1: Allowed
(R/W)

PMS_CORE*n*_MM_LP2HP_PERI_PMS_ALLOW Configures whether HP CPU*n* in machine mode has permission to access LP2HP_PERI_PMS_REG.
0: Not allowed
1: Allowed
(R/W)

PMS_CORE*n*_MM_DMA_PMS_ALLOW Configures whether HP CPU*n* in machine mode has permission to access HP_DMA_PMS_REG.
0: Not allowed
1: Allowed
(R/W)

PMS_CORE*n*_MM_HP_H264_DMA2D_ALLOW Configures whether HP CPU*n* in machine mode has permission to access 2D-DMA.
0: Not allowed
1: Allowed
(R/W)

Register 20.63. PMS_CORE*n*_MM_HP_PERI_PMS_REG2_REG (*n*: 0-1) (0x0010+0x20**n*)



31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	1	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Reset

PMS_CORE*n*_MM_HP_MCPWM0_ALLOW Configures whether HP CPU*n* in machine mode has permission to access HP MCPWM0.

0: Not allowed

1: Allowed

(R/W)

PMS_CORE*n*_MM_HP_MCPWM1_ALLOW Configures whether HP CPU*n* in machine mode has permission to access HP MCPWM1.

0: Not allowed

1: Allowed

(R/W)

PMS_CORE*n*_MM_HP_TIMER_GROUP0_ALLOW Configures whether HP CPU*n* in machine mode has permission to access HP timer group0.

0: Not allowed

1: Allowed

(R/W)

PMS_CORE*n*_MM_HP_TIMER_GROUP1_ALLOW Configures whether HP CPU*n* in machine mode has permission to access HP timer group1.

0: Not allowed

1: Allowed

(R/W)

PMS_CORE*n*_MM_HP_I2C0_ALLOW Configures whether HP CPU*n* in machine mode has permission to access HP I2C0.

0: Not allowed

1: Allowed

(R/W)

PMS_CORE*n*_MM_HP_I2C1_ALLOW Configures whether HP CPU*n* in machine mode has permission to access HP I2C1.

0: Not allowed

1: Allowed

(R/W)

Continued on the next page...

Register 20.63. PMS_CORE*n*_MM_HP_PERI_PMS_REG2_REG (*n*: 0-1) (0x000C+0x20**n*)

Continued from the previous page...

PMS_CORE*n*_MM_HP_I2S0_ALLOW Configures whether HP CPU*n* in machine mode has permission to access HP I2S0.
0: Not allowed
1: Allowed
(R/W)

PMS_CORE*n*_MM_HP_I2S1_ALLOW Configures whether HP CPU*n* in machine mode has permission to access HP I2S1.
0: Not allowed
1: Allowed
(R/W)

PMS_CORE*n*_MM_HP_I2S2_ALLOW Configures whether HP CPU*n* in machine mode has permission to access HP I2S2.
0: Not allowed
1: Allowed
(R/W)

PMS_CORE*n*_MM_HP_PCNT_ALLOW Configures whether HP CPU*n* in machine mode has permission to access HP PCNT.
0: Not allowed
1: Allowed
(R/W)

PMS_CORE*n*_MM_HP_UART0_ALLOW Configures whether HP CPU*n* in machine mode has permission to access HP UART0.
0: Not allowed
1: Allowed
(R/W)

PMS_CORE*n*_MM_HP_UART1_ALLOW Configures whether HP CPU*n* in machine mode has permission to access HP UART1.
0: Not allowed
1: Allowed
(R/W)

PMS_CORE*n*_MM_HP_UART2_ALLOW Configures whether HP CPU*n* in machine mode has permission to access HP UART2.
0: Not allowed
1: Allowed
(R/W)

Continued on the next page...

Register 20.63. PMS_CORE*n*_MM_HP_PERI_PMS_REG2_REG (*n*: 0-1) (0x000C+0x20**n*)

Continued from the previous page...

PMS_CORE*n*_MM_HP_UART3_ALLOW Configures whether HP CPU*n* in machine mode has permission to access HP UART3.
0: Not allowed
1: Allowed
(R/W)

PMS_CORE*n*_MM_HP_UART4_ALLOW Configures whether HP CPU*n* in machine mode has permission to access HP UART4.
0: Not allowed
1: Allowed
(R/W)

PMS_CORE*n*_MM_HP_PARLIO_ALLOW Configures whether HP CPU*n* in machine mode has permission to access HP PARLIO.
0: Not allowed
1: Allowed
(R/W)

PMS_CORE*n*_MM_HP_GPSPI2_ALLOW Configures whether HP CPU*n* in machine mode has permission to access HP GP-SPI2.
0: Not allowed
1: Allowed
(R/W)

PMS_CORE*n*_MM_HP_GPSPI3_ALLOW Configures whether HP CPU*n* in machine mode has permission to access HP GP-SPI3.
0: Not allowed
1: Allowed
(R/W)

PMS_CORE*n*_MM_HP_USBDEVICE_ALLOW Configures whether HP CPU*n* in machine mode has permission to access HP USB Serial/JTAG Controller.
0: Not allowed
1: Allowed
(R/W)

PMS_CORE*n*_MM_HP_LEDC_ALLOW Configures whether HP CPU*n* in machine mode has permission to access HP LEDC.
0: Not allowed
1: Allowed
(R/W)

Continued on the next page...

Register 20.63. PMS_CORE*n*_MM_HP_PERI_PMS_REG2_REG (*n*: 0-1) (0x000C+0x20**n*)

Continued from the previous page...

PMS_CORE*n*_MM_HP_ETM_ALLOW Configures whether HP CPU*n* in machine mode has permission to access HP ETM (Event Task Matrix).
0: Not allowed
1: Allowed
(R/W)

PMS_CORE*n*_MM_HP_INTRMTX_ALLOW Configures whether HP CPU*n* in machine mode has permission to access HP interrupt matrix.
0: Not allowed
1: Allowed
(R/W)

PMS_CORE*n*_MM_HP_TWAI0_ALLOW Configures whether HP CPU*n* in machine mode has permission to access HP TWAI0.
0: Not allowed
1: Allowed
(R/W)

PMS_CORE*n*_MM_HP_TWAI1_ALLOW Configures whether HP CPU*n* in machine mode has permission to access HP TWAI1.
0: Not allowed
1: Allowed
(R/W)

PMS_CORE*n*_MM_HP_TWAI2_ALLOW Configures whether HP CPU*n* in machine mode has permission to access HP TWAI2.
0: Not allowed
1: Allowed
(R/W)

PMS_CORE*n*_MM_HP_I3C_MST_ALLOW Configures whether HP CPU*n* in machine mode has permission to access HP I3C master controller.
0: Not allowed
1: Allowed
(R/W)

PMS_CORE*n*_MM_HP_I3C_SLV_ALLOW Configures whether HP CPU*n* in machine mode has permission to access HP I3C slave controller.
0: Not allowed
1: Allowed
(R/W)

Continued on the next page...

Register 20.63. PMS_CORE*n*_MM_HP_PERI_PMS_REG2_REG (*n*: 0-1) (0x000C+0x20n*)**

Continued from the previous page...

PMS_CORE*n*_MM_HP_LCDCAM_ALLOW Configures whether HP CPU*n* in machine mode has permission to access HP LCD_CAM.

0: Not allowed

1: Allowed

(R/W)

PMS_CORE*n*_MM_HP_ADC_ALLOW Configures whether HP CPU*n* in machine mode has permission to access HP ADC.

0: Not allowed

1: Allowed

(R/W)

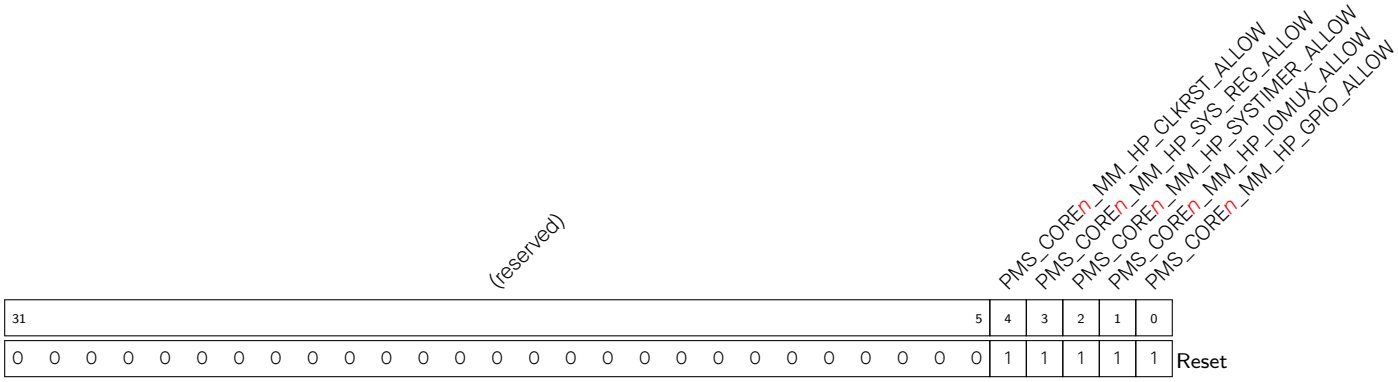
PMS_CORE*n*_MM_HP_UHCI_ALLOW Configures whether HP CPU*n* in machine mode has permission to access HP UHCI.

0: Not allowed

1: Allowed

(R/W)

Register 20.64. PMS_CORE*n*_MM_HP_PERI_PMS_REG3_REG (*n*: 0-1) (0x0014+0x20*n)



PMS_CORE*n*_MM_HP_GPIO_ALLOW Configures whether HP CPU*n* in machine mode has permission to access HP GPIO Matrix.

0: Not allowed

1: Allowed

(R/W)

PMS_CORE*n*_MM_HP_IOMUX_ALLOW Configures whether HP CPU*n* in machine mode has permission to access HP IO MUX.

0: Not allowed

1: Allowed

(R/W)

PMS_CORE*n*_MM_HP_SYSTIMER_ALLOW Configures whether HP CPU*n* in machine mode has permission to access HP system timer.

0: Not allowed

1: Allowed

(R/W)

PMS_CORE*n*_MM_HP_SYS_REG_ALLOW Configures whether HP CPU*n* in machine mode has permission to access HP system register.

0: Not allowed

1: Allowed

(R/W)

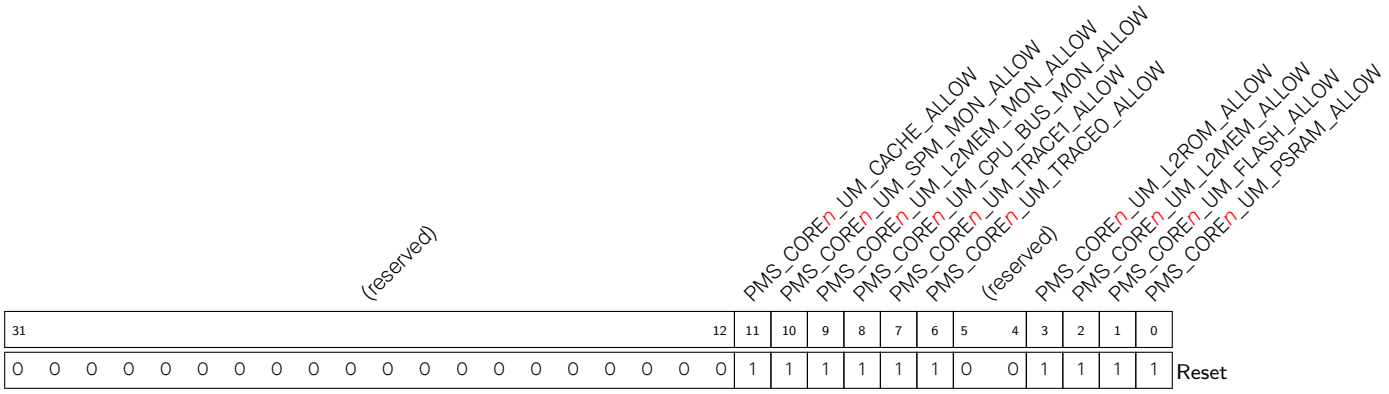
PMS_CORE*n*_MM_HP_CLKRST_ALLOW Configures whether HP CPU*n* in machine mode has permission to access HP_SYS_CLKRST.

0: Not allowed

1: Allowed

(R/W)

Register 20.65. PMS_CORE*n*_UM_HP_PERI_PMS_REG0_REG (*n*: 0-1) (0x0018+0x20**n*)



PMS_CORE*n*_UM_PSRAM_ALLOW Configures whether HP CPU*n* in user mode has permission to access external RAM without going through cache.

0: Not allowed

1: Allowed

(R/W)

PMS_CORE*n*_UM_FLASH_ALLOW Configures whether HP CPU*n* in user mode has permission to access external flash without going through cache.

0: Not allowed

1: Allowed

(R/W)

PMS_CORE*n*_UM_L2MEM_ALLOW Configures whether HP CPU*n* in user mode has permission to access HP L2MEM without going through cache.

0: Not allowed

1: Allowed

(R/W)

PMS_CORE*n*_UM_L2ROM_ALLOW Configures whether HP CPU*n* in user mode has permission to access HP ROM without going through cache.

0: Not allowed

1: Allowed

(R/W)

PMS_CORE*n*_UM_TRACE0_ALLOW Configures whether HP CPU*n* in user mode has permission to access TRACE0.

0: Not allowed

1: Allowed

(R/W)

PMS_CORE*n*_UM_TRACE1_ALLOW Configures whether HP CPU*n* in user mode has permission to access TRACE1.

0: Not allowed

1: Allowed

(R/W)

Continued on the next page...

Register 20.65. PMS_CORE*n*_UM_HP_PERI_PMS_REG0_REG (*n*: 0-1) (0x000C+0x20**n*)

Continued from the previous page...

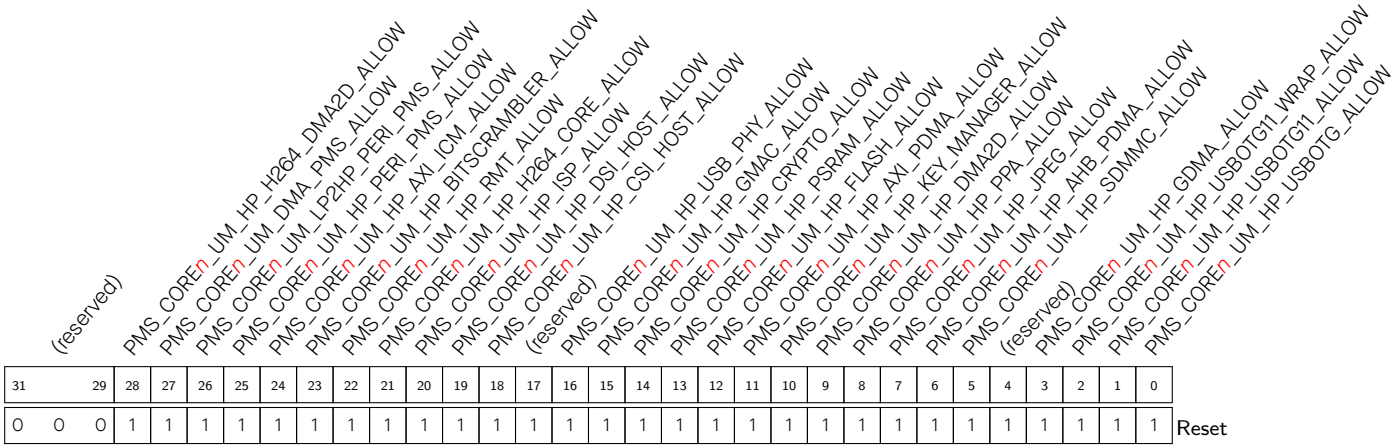
PMS_CORE*n*_UM_CPU_BUS_MON_ALLOW Configures whether HP CPU*n* in user mode has permission to access CPU bus monitor.
0: Not allowed
1: Allowed
(R/W)

PMS_CORE*n*_UM_L2MEM_MON_ALLOW Configures whether HP CPU*n* in user mode has permission to access L2MEM monitor.
0: Not allowed
1: Allowed
(R/W)

PMS_CORE*n*_UM_SPM_MON_ALLOW Configures whether HP CPU*n* in user mode has permission to access SPM monitor.
0: Not allowed
1: Allowed
(R/W)

PMS_CORE*n*_UM_CACHE_ALLOW Configures whether HP CPU*n* in user mode has permission to access cache.
0: Not allowed
1: Allowed
(R/W)

Register 20.66. PMS_CORE*n*_UM_HP_PERI_PMS_REG1_REG (*n*: 0-1) (0x001C+0x20**n*)



PMS_CORE*n*_UM_HP_USBOTG_ALLOW Configures whether HP CPU*n* in user mode has permission to access USB 2.0 High-Speed OTG.

0: Not allowed

1: Allowed

(R/W)

PMS_CORE*n*_UM_HP_USBOTG11_ALLOW Configures whether HP CPU*n* in user mode has permission to access USB 2.0 Full-Speed OTG.

0: Not allowed

1: Allowed

(R/W)

PMS_CORE*n*_UM_HP_USBOTG11_WRAP_ALLOW Configures whether HP CPU*n* in user mode has permission to access USB 2.0 Full-Speed OTG's wrap.

0: Not allowed

1: Allowed

(R/W)

PMS_CORE*n*_UM_HP_GDMA_ALLOW Configures whether HP CPU*n* in user mode has permission to access HP VDMA.

0: Not allowed

1: Allowed

(R/W)

PMS_CORE*n*_UM_HP_SDMMC_ALLOW Configures whether HP CPU*n* in user mode has permission to access HP SDMMC.

0: Not allowed

1: Allowed

(R/W)

Continued on the next page...

Register 20.66. PMS_CORE*n*_UM_HP_PERI_PMS_REG1_REG (*n*: 0-1) (0x000C+0x20**n*)

Continued from the previous page...

PMS_CORE*n*_UM_HP_AHB_PDMA_ALLOW Configures whether HP CPU*n* in user mode has permission to access GDMA-AHB.
0: Not allowed
1: Allowed
(R/W)

PMS_CORE*n*_UM_HP_JPEG_ALLOW Configures whether HP CPU*n* in user mode has permission to access HP JPEG.
0: Not allowed
1: Allowed
(R/W)

PMS_CORE*n*_UM_HP_PPA_ALLOW Configures whether HP CPU*n* in user mode has permission to access HP PPA.
0: Not allowed
1: Allowed
(R/W)

PMS_CORE*n*_UM_HP_DMA2D_ALLOW Configures whether HP CPU*n* in user mode has permission to access HP 2D-DMA.
0: Not allowed
1: Allowed
(R/W)

PMS_CORE*n*_UM_HP_KEY_MANAGER_ALLOW Configures whether HP CPU*n* in user mode has permission to access HP Key Manager.
0: Not allowed
1: Allowed
(R/W)

PMS_CORE*n*_UM_HP_AXI_PDMA_ALLOW Configures whether HP CPU*n* in user mode has permission to access HP GDMA-AXI.
0: Not allowed
1: Allowed
(R/W)

PMS_CORE*n*_UM_HP_FLASH_ALLOW Configures whether HP CPU*n* in user mode has permission to access HP flash MSPI controller.
0: Not allowed
1: Allowed
(R/W)

PMS_CORE*n*_UM_HP_PSRAM_ALLOW Configures whether HP CPU*n* in user mode has permission to access HP PSRAM MSPI controller.
0: Not allowed
1: Allowed
(R/W)

Continued on the next page...

Register 20.66. PMS_CORE*n*_UM_HP_PERI_PMS_REG1_REG (*n*: 0-1) (0x000C+0x20**n*)

Continued from the previous page...

PMS_CORE*n*_UM_HP_CRYPTO_ALLOW Configures whether HP CPU*n* in user mode has permission to access HP CRYPTO.
0: Not allowed
1: Allowed
(R/W)

PMS_CORE*n*_UM_HP_GMAC_ALLOW Configures whether HP CPU*n* in user mode has permission to access HP EMAC.
0: Not allowed
1: Allowed
(R/W)

PMS_CORE*n*_UM_HP_USB_PHY_ALLOW Configures whether HP CPU*n* in user mode has permission to access USB 2.0 High-Speed OTG PHY.
0: Not allowed
1: Allowed
(R/W)

PMS_CORE*n*_UM_HP_CSI_HOST_ALLOW Configures whether HP CPU*n* in user mode has permission to access HP MIPI CSI host.
0: Not allowed
1: Allowed
(R/W)

PMS_CORE*n*_UM_HP_DSI_HOST_ALLOW Configures whether HP CPU*n* in user mode has permission to access HP MIPI DSI host.
0: Not allowed
1: Allowed
(R/W)

PMS_CORE*n*_UM_HP_ISP_ALLOW Configures whether HP CPU*n* in user mode has permission to access HP ISP.
0: Not allowed
1: Allowed
(R/W)

PMS_CORE*n*_UM_HP_H264_CORE_ALLOW Configures whether HP CPU*n* in user mode has permission to access HP H264 Encoder.
0: Not allowed
1: Allowed
(R/W)

Continued on the next page...

Register 20.66. PMS_CORE*n*_UM_HP_PERI_PMS_REG1_REG (*n*: 0-1) (0x000C+0x20**n*)

Continued from the previous page...

PMS_CORE*n*_UM_HP_RMT_ALLOW Configures whether HP CPU*n* in user mode has permission to access HP RMT.
0: Not allowed
1: Allowed
(R/W)

PMS_CORE*n*_UM_HP_BITSCRAMBLER_ALLOW Configures whether HP CPU*n* in user mode has permission to access HP bit scrambler.
0: Not allowed
1: Allowed
(R/W)

PMS_CORE*n*_UM_HP_AXI_ICM_ALLOW Configures whether HP CPU*n* in user mode has permission to access HP AXI ICM.
0: Not allowed
1: Allowed
(R/W)

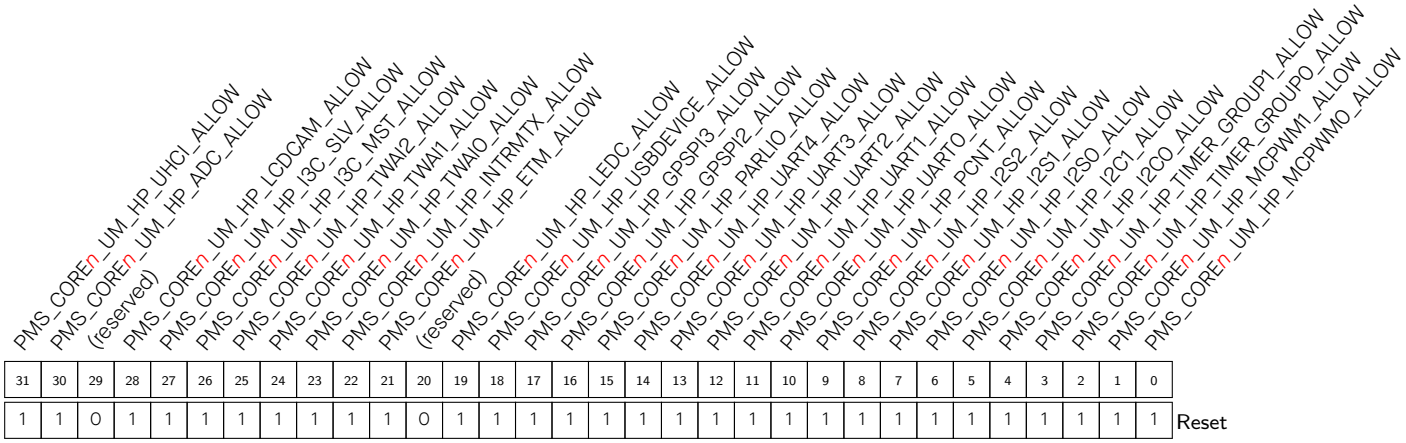
PMS_CORE*n*_UM_HP_PERI_PMS_ALLOW Configures whether HP CPU*n* in user mode has permission to access HP_PERI_PMS_REG.
0: Not allowed
1: Allowed
(R/W)

PMS_CORE*n*_UM_LP2HP_PERI_PMS_ALLOW Configures whether HP CPU*n* in user mode has permission to access LP2HP_PERI_PMS_REG.
0: Not allowed
1: Allowed
(R/W)

PMS_CORE*n*_UM_DMA_PMS_ALLOW Configures whether HP CPU*n* in user mode has permission to access HP_DMA_PMS_REG.
0: Not allowed
1: Allowed
(R/W)

PMS_CORE*n*_UM_HP_H264_DMA2D_ALLOW Configures whether HP CPU*n* in user mode has permission to access 2D-DMA.
0: Not allowed
1: Allowed
(R/W)

Register 20.67. PMS_CORE*n*_UM_HP_PERI_PMS_REG2_REG (*n*: 0-1) (0x0020+0x20**n*)



31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
1	1	0	1	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	Reset

PMS_CORE*n*_UM_HP_MCPWM0_ALLOW Configures whether HP CPU*n* in user mode has permission to access HP MCPWM0.

0: Not allowed

1: Allowed

(R/W)

PMS_CORE*n*_UM_HP_MCPWM1_ALLOW Configures whether HP CPU*n* in user mode has permission to access HP MCPWM1.

0: Not allowed

1: Allowed

(R/W)

PMS_CORE*n*_UM_HP_TIMER_GROUP0_ALLOW Configures whether HP CPU*n* in user mode has permission to access HP timer group0.

0: Not allowed

1: Allowed

(R/W)

PMS_CORE*n*_UM_HP_TIMER_GROUP1_ALLOW Configures whether HP CPU*n* in user mode has permission to access HP timer group1.

0: Not allowed

1: Allowed

(R/W)

PMS_CORE*n*_UM_HP_I2C0_ALLOW Configures whether HP CPU*n* in user mode has permission to access HP I2C0.

0: Not allowed

1: Allowed

(R/W)

PMS_CORE*n*_UM_HP_I2C1_ALLOW Configures whether HP CPU*n* in user mode has permission to access HP I2C1.

0: Not allowed

1: Allowed

(R/W)

Continued on the next page...

Register 20.67. PMS_CORE*n*_UM_HP_PERI_PMS_REG2_REG (*n*: 0-1) (0x0020+0x20**n*)

Continued from the previous page...

PMS_CORE*n*_UM_HP_I2S0_ALLOW Configures whether HP CPU*n* in user mode has permission to access HP I2S0.
0: Not allowed
1: Allowed
(R/W)

PMS_CORE*n*_UM_HP_I2S1_ALLOW Configures whether HP CPU*n* in user mode has permission to access HP I2S1.
0: Not allowed
1: Allowed
(R/W)

PMS_CORE*n*_UM_HP_I2S2_ALLOW Configures whether HP CPU*n* in user mode has permission to access HP I2S2.
0: Not allowed
1: Allowed
(R/W)

PMS_CORE*n*_UM_HP_PCNT_ALLOW Configures whether HP CPU*n* in user mode has permission to access HP PCNT.
0: Not allowed
1: Allowed
(R/W)

PMS_CORE*n*_UM_HP_UART0_ALLOW Configures whether HP CPU*n* in user mode has permission to access HP UART0.
0: Not allowed
1: Allowed
(R/W)

PMS_CORE*n*_UM_HP_UART1_ALLOW Configures whether HP CPU*n* in user mode has permission to access HP UART1.
0: Not allowed
1: Allowed
(R/W)

PMS_CORE*n*_UM_HP_UART2_ALLOW Configures whether HP CPU*n* in user mode has permission to access HP UART2.
0: Not allowed
1: Allowed
(R/W)

Continued on the next page...

Register 20.67. PMS_CORE*n*_UM_HP_PERI_PMS_REG2_REG (*n*: 0-1) (0x0020+0x20**n*)

Continued from the previous page...

PMS_CORE*n*_UM_HP_UART3_ALLOW Configures whether HP CPU*n* in user mode has permission to access HP UART3.
0: Not allowed
1: Allowed
(R/W)

PMS_CORE*n*_UM_HP_UART4_ALLOW Configures whether HP CPU*n* in user mode has permission to access HP UART4.
0: Not allowed
1: Allowed
(R/W)

PMS_CORE*n*_UM_HP_PARLIO_ALLOW Configures whether HP CPU*n* in user mode has permission to access HP PARLIO.
0: Not allowed
1: Allowed
(R/W)

PMS_CORE*n*_UM_HP_GPSPI2_ALLOW Configures whether HP CPU*n* in user mode has permission to access HP GP-SPI2.
0: Not allowed
1: Allowed
(R/W)

PMS_CORE*n*_UM_HP_GPSPI3_ALLOW Configures whether HP CPU*n* in user mode has permission to access HP GP-SPI3.
0: Not allowed
1: Allowed
(R/W)

PMS_CORE*n*_UM_HP_USBDEVICE_ALLOW Configures whether HP CPU*n* in user mode has permission to access HP USB/Serial JTAG Controller.
0: Not allowed
1: Allowed
(R/W)

PMS_CORE*n*_UM_HP_LEDC_ALLOW Configures whether HP CPU*n* in user mode has permission to access HP LEDC.
0: Not allowed
1: Allowed
(R/W)

Continued on the next page...

Register 20.67. PMS_CORE*n*_UM_HP_PERI_PMS_REG2_REG (*n*: 0-1) (0x0020+0x20**n*)

Continued from the previous page...

PMS_CORE*n*_UM_HP_ETM_ALLOW Configures whether HP CPU*n* in user mode has permission to access HP ETM.
0: Not allowed
1: Allowed
(R/W)

PMS_CORE*n*_UM_HP_INTRMTX_ALLOW Configures whether HP CPU*n* in user mode has permission to access HP interrupt matrix.
0: Not allowed
1: Allowed
(R/W)

PMS_CORE*n*_UM_HP_TWAI0_ALLOW Configures whether HP CPU*n* in user mode has permission to access HP TWAI0.
0: Not allowed
1: Allowed
(R/W)

PMS_CORE*n*_UM_HP_TWAI1_ALLOW Configures whether HP CPU*n* in user mode has permission to access HP TWAI1.
0: Not allowed
1: Allowed
(R/W)

PMS_CORE*n*_UM_HP_TWAI2_ALLOW Configures whether HP CPU*n* in user mode has permission to access HP TWAI2.
0: Not allowed
1: Allowed
(R/W)

PMS_CORE*n*_UM_HP_I3C_MST_ALLOW Configures whether HP CPU*n* in user mode has permission to access HP I3C master controller.
0: Not allowed
1: Allowed
(R/W)

PMS_CORE*n*_UM_HP_I3C_SLV_ALLOW Configures whether HP CPU*n* in user mode has permission to access HP I3C slave controller.
0: Not allowed
1: Allowed
(R/W)

Continued on the next page...

Register 20.67. PMS_CORE*n*_UM_HP_PERI_PMS_REG2_REG (*n*: 0-1) (0x0020+0x20**n*)

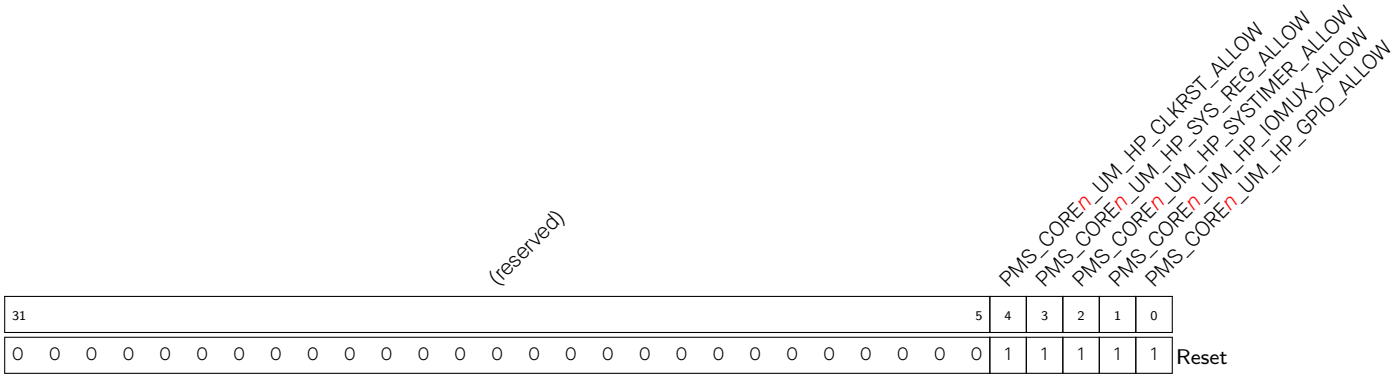
Continued from the previous page...

PMS_CORE*n*_UM_HP_LCDCAM_ALLOW Configures whether HP CPU*n* in user mode has permission to access HP LCD_CAM.
0: Not allowed
1: Allowed
(R/W)

PMS_CORE*n*_UM_HP_ADC_ALLOW Configures whether HP CPU*n* in user mode has permission to access HP ADC.
0: Not allowed
1: Allowed
(R/W)

PMS_CORE*n*_UM_HP_UHCI_ALLOW Configures whether HP CPU*n* in user mode has permission to access HP UHCI.
0: Not allowed
1: Allowed
(R/W)

Register 20.68. PMS_CORE*n*_UM_HP_PERI_PMS_REG3_REG (*n*: 0-1) (0x0024+0x20*n)



PMS_CORE*n*_UM_HP_GPIO_ALLOW Configures whether HP CPU*n* in user mode has permission to access HP GPIO Matrix.
0: Not allowed
1: Allowed
(R/W)

PMS_CORE*n*_UM_HP_IOMUX_ALLOW Configures whether HP CPU*n* in user mode has permission to access HP IO MUX.
0: Not allowed
1: Allowed
(R/W)

PMS_CORE*n*_UM_HP_SYSTIMER_ALLOW Configures whether HP CPU*n* in user mode has permission to access HP system timer.
0: Not allowed
1: Allowed
(R/W)

PMS_CORE*n*_UM_HP_SYS_REG_ALLOW Configures whether HP CPU*n* in user mode has permission to access HP system register.
0: Not allowed
1: Allowed
(R/W)

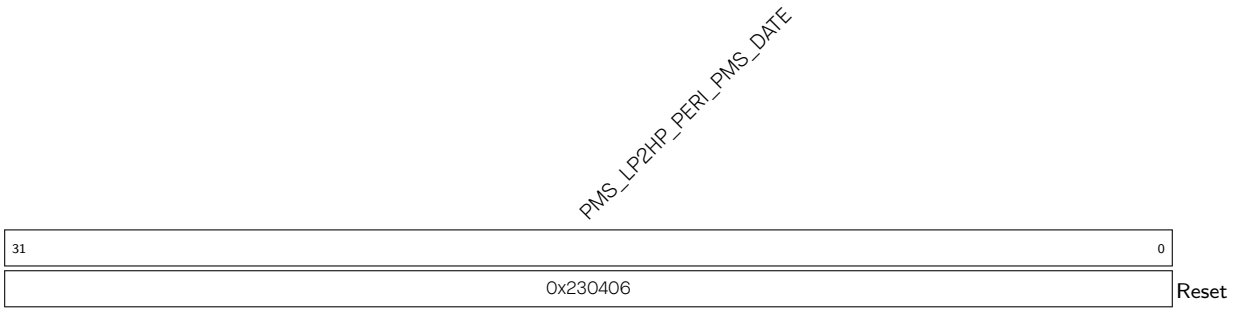
PMS_CORE*n*_UM_HP_CLKRST_ALLOW Configures whether HP CPU*n* in user mode has permission to access HP_SYS_CLKRST.
0: Not allowed
1: Allowed
(R/W)

20.7.3 LP2HP_PERI_PMS_REG

The addresses in this section are relative to the LP2HP_PERI_PMS base address provided in Table 9.3-2 in Chapter 9 *System and Memory*.

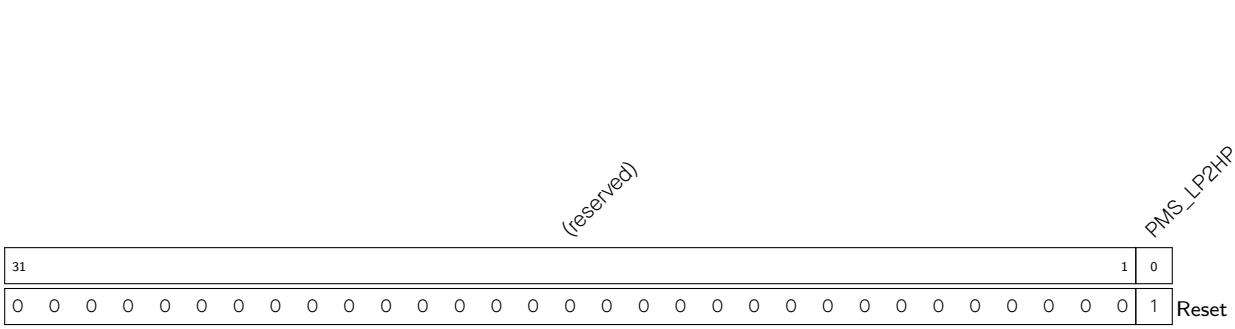
For how to program reserved fields, please refer to Section Programming Reserved Register Field.

Register 20.69. PMS_LP2HP_PERI_PMS_DATE_REG (0x0000)



PMS_LP2HP_PERI_PMS_DATE Version control register. (R/W)

Register 20.70. PMS_LP2HP_PERI_PMS_CLK_EN_REG (0x0004)



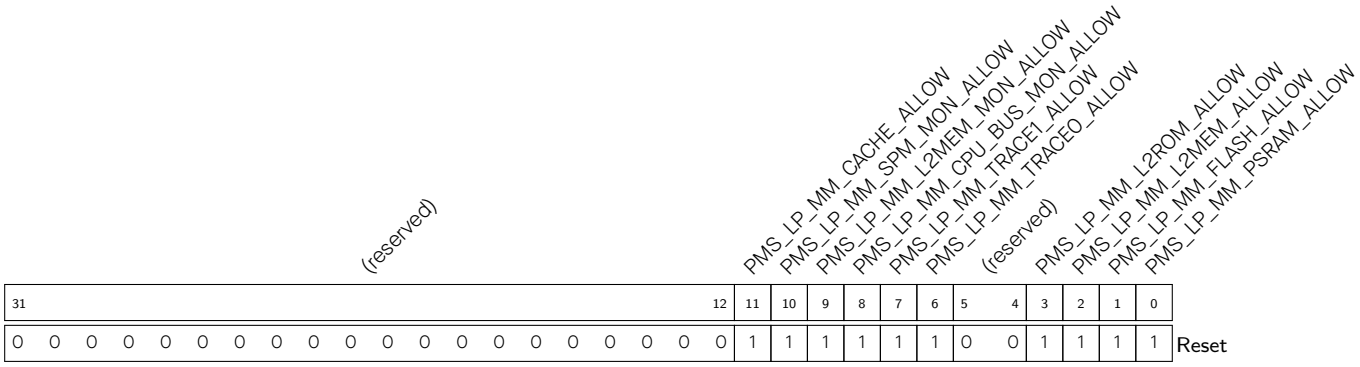
PMS_LP2HP_PERI_PMS_CLK_EN Configures whether to keep the clock always on.

0: Enable automatic clock gating.

1: Keep the clock always on.

(R/W)

Register 20.71. PMS_LP_MM_PMS_REG0_REG (0x0008)



PMS_LP_MM_PSRAM_ALLOW Configures whether the LP CPU in machine mode has permission to access external RAM without going through cache.

0: Not allowed

1: Allowed

(R/W)

PMS_LP_MM_FLASH_ALLOW Configures whether the LP CPU in machine mode has permission to access external flash without going through cache.

0: Not allowed

1: Allowed

(R/W)

PMS_LP_MM_L2MEM_ALLOW Configures whether the LP CPU in machine mode has permission to access HP L2M2M without going through cache.

0: Not allowed

1: Allowed

(R/W)

PMS_LP_MM_L2ROM_ALLOW Configures whether the LP CPU in machine mode has permission to access HP ROM without going through cache.

0: Not allowed

1: Allowed

(R/W)

PMS_LP_MM_TRACE0_ALLOW Configures whether the LP CPU in machine mode has permission to access TRACE0.

0: Not allowed

1: Allowed

(R/W)

PMS_LP_MM_TRACE1_ALLOW Configures whether the LP CPU in machine mode has permission to access TRACE1.

0: Not allowed

1: Allowed

(R/W)

Continued on the next page...

Register 20.71. PMS_LP_MM_PMS_REG0_REG (0x0008)

Continued from the previous page...

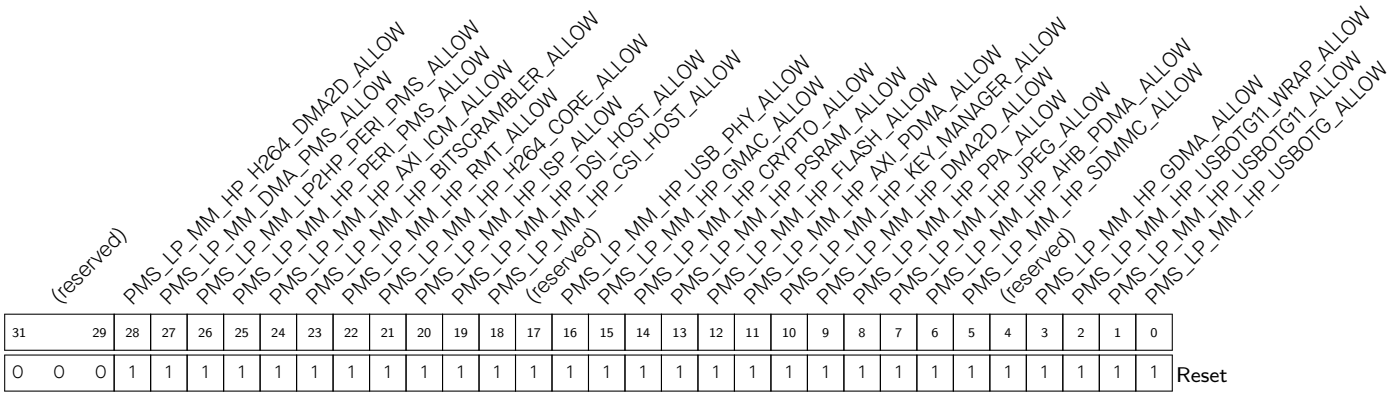
PMS_LP_MM_CPU_BUS_MON_ALLOW Configures whether the LP CPU in machine mode has permission to access CPU bus monitor.
0: Not allowed
1: Allowed
(R/W)

PMS_LP_MM_L2MEM_MON_ALLOW Configures whether the LP CPU in machine mode has permission to access L2MEM monitor.
0: Not allowed
1: Allowed
(R/W)

PMS_LP_MM_SPM_MON_ALLOW Configures whether the LP CPU in machine mode has permission to access SPM monitor.
0: Not allowed
1: Allowed
(R/W)

PMS_LP_MM_CACHE_ALLOW Configures whether the LP CPU in machine mode has permission to access cache.
0: Not allowed
1: Allowed
(R/W)

Register 20.72. PMS_LP_MM_PMS_REG1_REG (0x0030)



PMS_LP_MM_HP_USBOTG_ALLOW Configures whether the LP CPU in machine mode has permission to access USB 2.0 High-Speed OTG.

0: Not allowed

1: Allowed

(R/W)

PMS_LP_MM_HP_USBOTG11_ALLOW Configures whether the LP CPU in machine mode has permission to access USB 2.0 Full-Speed OTG.

0: Not allowed

1: Allowed

(R/W)

PMS_LP_MM_HP_USBOTG11_WRAP_ALLOW Configures whether the LP CPU in machine mode has permission to access USB 2.0 Full-Speed OTG's wrap.

0: Not allowed

1: Allowed

(R/W)

PMS_LP_MM_HP_GDMA_ALLOW Configures whether the LP CPU in machine mode has permission to access HP VDMA.

0: Not allowed

1: Allowed

(R/W)

PMS_LP_MM_HP_SDMMC_ALLOW Configures whether the LP CPU in machine mode has permission to access HP SDMMC.

0: Not allowed

1: Allowed

(R/W)

PMS_LP_MM_HP_AHB_PDMA_ALLOW Configures whether the LP CPU in machine mode has permission to access GDMA-AHB.

0: Not allowed

1: Allowed

(R/W)

Continued on the next page...

Register 20.72. PMS_LP_MM_PMS_REG1_REG (0x0030)

Continued from the previous page...

PMS_LP_MM_HP_JPEG_ALLOW Configures whether the LP CPU in machine mode has permission to access HP JPEG.
0: Not allowed
1: Allowed
(R/W)

PMS_LP_MM_HP_PPA_ALLOW Configures whether the LP CPU in machine mode has permission to access HP PPA.
0: Not allowed
1: Allowed
(R/W)

PMS_LP_MM_HP_DMA2D_ALLOW Configures whether the LP CPU in machine mode has permission to access HP 2D-DMA.
0: Not allowed 1: Allowed
(R/W)

PMS_LP_MM_HP_KEY_MANAGER_ALLOW Configures whether the LP CPU in machine mode has permission to access HP Key Manager.
0: Not allowed 1: Allowed
(R/W)

PMS_LP_MM_HP_AXI_PDMA_ALLOW Configures whether the LP CPU in machine mode has permission to access HP GDMA-AXI.
0: Not allowed
1: Allowed
(R/W)

PMS_LP_MM_HP_FLASH_ALLOW Configures whether the LP CPU in machine mode has permission to access HP flash MSPI controller.
0: Not allowed
1: Allowed
(R/W)

PMS_LP_MM_HP_PSRAM_ALLOW Configures whether the LP CPU in machine mode has permission to access HP PSRAM MSPI controller.
0: Not allowed
1: Allowed
(R/W)

PMS_LP_MM_HP_CRYPTO_ALLOW Configures whether the LP CPU in machine mode has permission to access HP CRYPTO.
0: Not allowed
1: Allowed
(R/W)

Continued on the next page...

Register 20.72. PMS_LP_MM_PMS_REG1_REG (0x0030)

Continued from the previous page...

PMS_LP_MM_HP_GMAC_ALLOW Configures whether the LP CPU in machine mode has permission to access HP EMAC.
0: Not allowed
1: Allowed
(R/W)

PMS_LP_MM_HP_USB_PHY_ALLOW Configures whether the LP CPU in machine mode has permission to access USB 2.0 High-Speed OTG PHY.
0: Not allowed
1: Allowed
(R/W)

PMS_LP_MM_HP_CSI_HOST_ALLOW Configures whether the LP CPU in machine mode has permission to access HP MIPI CSI host.
0: Not allowed
1: Allowed
(R/W)

PMS_LP_MM_HP_DSI_HOST_ALLOW Configures whether the LP CPU in machine mode has permission to access HP MIPI DSI host.
0: Not allowed
1: Allowed
(R/W)

PMS_LP_MM_HP_ISP_ALLOW Configures whether the LP CPU in machine mode has permission to access HP ISP.
0: Not allowed
1: Allowed
(R/W)

PMS_LP_MM_HP_H264_CORE_ALLOW Configures whether the LP CPU in machine mode has permission to access HP H264 Encoder.
0: Not allowed
1: Allowed
(R/W)

PMS_LP_MM_HP_RMT_ALLOW Configures whether the LP CPU in machine mode has permission to access HP RMT.
0: Not allowed
1: Allowed
(R/W)

Continued on the next page...

Register 20.72. PMS_LP_MM_PMS_REG1_REG (0x0030)

Continued from the previous page...

PMS_LP_MM_HP_BITSCRAMBLER_ALLOW Configures whether the LP CPU in machine mode has permission to access HP bit scrambler.
0: Not allowed
1: Allowed
(R/W)

PMS_LP_MM_HP_AXI_ICM_ALLOW Configures whether the LP CPU in machine mode has permission to access HP AXI ICM.
0: Not allowed
1: Allowed
(R/W)

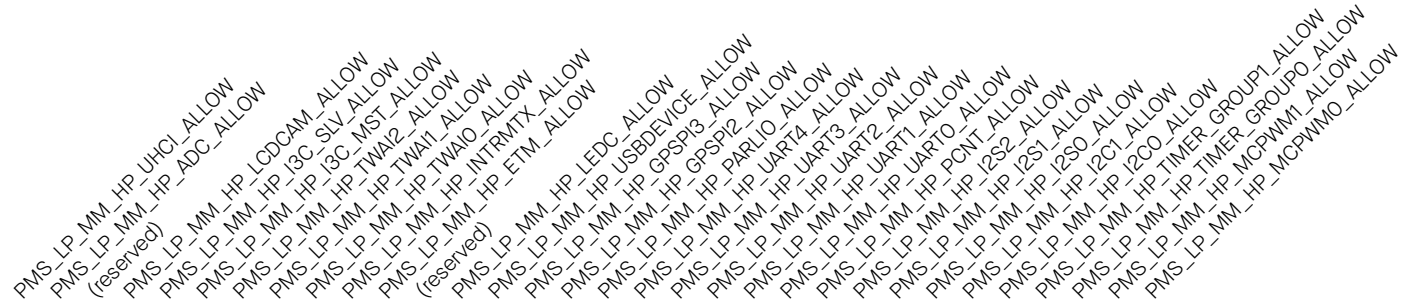
PMS_LP_MM_HP_PERI_PMS_ALLOW Configures whether the LP CPU in machine mode has permission to access HP_PERI_PMS_REG.
0: Not allowed
1: Allowed
(R/W)

PMS_LP_MM_LP2HP_PERI_PMS_ALLOW Configures whether the LP CPU in machine mode has permission to access LP2HP_PERI_PMS_REG.
0: Not allowed
1: Allowed
(R/W)

PMS_LP_MM_DMA_PMS_ALLOW Configures whether the LP CPU in machine mode has permission to access HP_DMA_PMS_REG.
0: Not allowed
1: Allowed
(R/W)

PMS_LP_MM_HP_H264_DMA2D_ALLOW Configures whether the LP CPU in machine mode has permission to access 2D-DMA.
0: Not allowed
1: Allowed
(R/W)

Register 20.73. PMS_LP_MM_PMS_REG2_REG (0x00A4)



31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
1	1	0	1	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	Reset

PMS_LP_MM_HP_MCPWM0_ALLOW Configures whether the LP CPU in machine mode has permission to access HP MCPWM0.

0: Not allowed

1: Allowed

(R/W)

PMS_LP_MM_HP_MCPWM1_ALLOW Configures whether the LP CPU in machine mode has permission to access HP MCPWM1.

0: Not allowed

1: Allowed

(R/W)

PMS_LP_MM_HP_TIMER_GROUP0_ALLOW Configures whether the LP CPU in machine mode has permission to access HP timer group0.

0: Not allowed

1: Allowed

(R/W)

PMS_LP_MM_HP_TIMER_GROUP1_ALLOW Configures whether the LP CPU in machine mode has permission to access HP timer group1.

0: Not allowed

1: Allowed

(R/W)

PMS_LP_MM_HP_I2C0_ALLOW Configures whether the LP CPU in machine mode has permission to access HP I2C0.

0: Not allowed

1: Allowed

(R/W)

PMS_LP_MM_HP_I2C1_ALLOW Configures whether the LP CPU in machine mode has permission to access HP I2C1.

0: Not allowed

1: Allowed

(R/W)

Continued on the next page...

Register 20.73. PMS_LP_MM_PMS_REG2_REG (0x00A4)

Continued from the previous page...

PMS_LP_MM_HP_I2S0_ALLOW Configures whether the LP CPU in machine mode has permission to access HP I2S0.
0: Not allowed
1: Allowed
(R/W)

PMS_LP_MM_HP_I2S1_ALLOW Configures whether the LP CPU in machine mode has permission to access HP I2S1.
0: Not allowed
1: Allowed
(R/W)

PMS_LP_MM_HP_I2S2_ALLOW Configures whether the LP CPU in machine mode has permission to access HP I2S2.
0: Not allowed
1: Allowed
(R/W)

PMS_LP_MM_HP_PCNT_ALLOW Configures whether the LP CPU in machine mode has permission to access HP PCNT.
0: Not allowed
1: Allowed
(R/W)

PMS_LP_MM_HP_UART0_ALLOW Configures whether the LP CPU in machine mode has permission to access HP UART0.
0: Not allowed
1: Allowed
(R/W)

PMS_LP_MM_HP_UART1_ALLOW Configures whether the LP CPU in machine mode has permission to access HP UART1.
0: Not allowed
1: Allowed
(R/W)

PMS_LP_MM_HP_UART2_ALLOW Configures whether the LP CPU in machine mode has permission to access HP UART2.
0: Not allowed
1: Allowed
(R/W)

Continued on the next page...

Register 20.73. PMS_LP_MM_PMS_REG2_REG (0x00A4)

Continued from the previous page...

PMS_LP_MM_HP_UART3_ALLOW Configures whether the LP CPU in machine mode has permission to access HP UART3.
0: Not allowed
1: Allowed
(R/W)

PMS_LP_MM_HP_UART4_ALLOW Configures whether the LP CPU in machine mode has permission to access HP UART4.
0: Not allowed
1: Allowed
(R/W)

PMS_LP_MM_HP_PARLIO_ALLOW Configures whether the LP CPU in machine mode has permission to access HP PARLIO.
0: Not allowed
1: Allowed
(R/W)

PMS_LP_MM_HP_GPSPI2_ALLOW Configures whether the LP CPU in machine mode has permission to access HP GP-SPI2.
0: Not allowed
1: Allowed
(R/W)

PMS_LP_MM_HP_GPSPI3_ALLOW Configures whether the LP CPU in machine mode has permission to access HP GP-SPI3.
0: Not allowed
1: Allowed
(R/W)

PMS_LP_MM_HP_USBDEVICE_ALLOW Configures whether the LP CPU in machine mode has permission to access HP USB/Serial JTAG Controller.
0: Not allowed
1: Allowed
(R/W)

PMS_LP_MM_HP_LEDC_ALLOW Configures whether the LP CPU in machine mode has permission to access HP LEDC.
0: Not allowed
1: Allowed
(R/W)

Continued on the next page...

Register 20.73. PMS_LP_MM_PMS_REG2_REG (0x00A4)

Continued from the previous page...

PMS_LP_MM_HP_ETM_ALLOW Configures whether the LP CPU in machine mode has permission to access HP ETM.
0: Not allowed
1: Allowed
(R/W)

PMS_LP_MM_HP_INTRMTX_ALLOW Configures whether the LP CPU in machine mode has permission to access HP interrupt matrix.
0: Not allowed
1: Allowed
(R/W)

PMS_LP_MM_HP_TWAI0_ALLOW Configures whether the LP CPU in machine mode has permission to access HP TWAI0.
0: Not allowed
1: Allowed
(R/W)

PMS_LP_MM_HP_TWAI1_ALLOW Configures whether the LP CPU in machine mode has permission to access HP TWAI1.
0: Not allowed
1: Allowed
(R/W)

PMS_LP_MM_HP_TWAI2_ALLOW Configures whether the LP CPU in machine mode has permission to access HP TWAI2.
0: Not allowed
1: Allowed
(R/W)

PMS_LP_MM_HP_I3C_MST_ALLOW Configures whether the LP CPU in machine mode has permission to access HP I3C master controller.
0: Not allowed
1: Allowed
(R/W)

PMS_LP_MM_HP_I3C_SLV_ALLOW Configures whether the LP CPU in machine mode has permission to access HP I3C slave controller.
0: Not allowed
1: Allowed
(R/W)

Continued on the next page...

Register 20.73. PMS_LP_MM_PMS_REG2_REG (0x00A4)

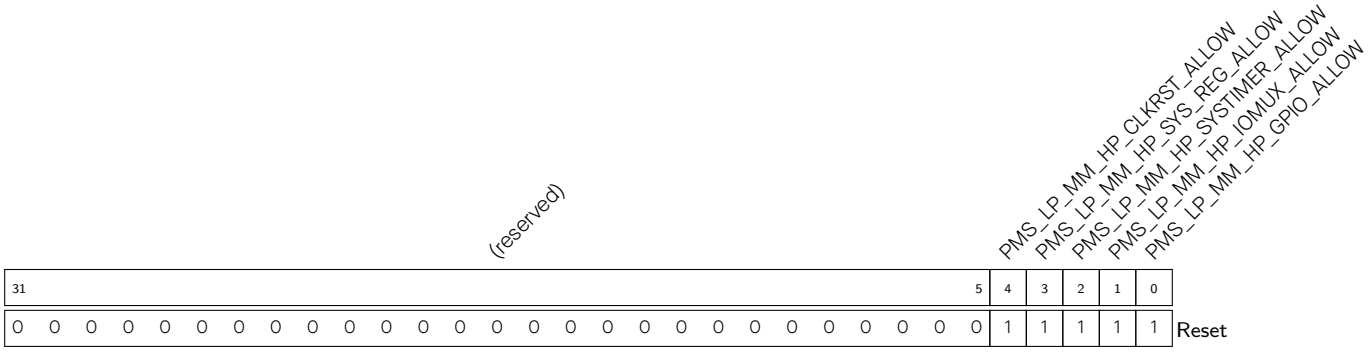
Continued from the previous page...

PMS_LP_MM_HP_LCDCAM_ALLOW Configures whether the LP CPU in machine mode has permission to access HP LCD_CAM.
0: Not allowed
1: Allowed
(R/W)

PMS_LP_MM_HP_ADC_ALLOW Configures whether the LP CPU in machine mode has permission to access HP ADC.
0: Not allowed
1: Allowed
(R/W)

PMS_LP_MM_HP_UHCI_ALLOW Configures whether the LP CPU in machine mode has permission to access HP UHCI.
0: Not allowed
1: Allowed
(R/W)

Register 20.74. PMS_LP_MM_PMS_REG3_REG (0x011C)



PMS_LP_MM_HP_GPIO_ALLOW Configures whether the LP CPU in machine mode has permission to access HP GPIO Matrix.

0: Not allowed

1: Allowed

(R/W)

PMS_LP_MM_HP_IOMUX_ALLOW Configures whether the LP CPU in machine mode has permission to access HP IO MUX.

0: Not allowed

1: Allowed

(R/W)

PMS_LP_MM_HP_SYSTIMER_ALLOW Configures whether the LP CPU in machine mode has permission to access HP system timer.

0: Not allowed

1: Allowed

(R/W)

PMS_LP_MM_HP_SYS_REG_ALLOW Configures whether the LP CPU in machine mode has permission to access HP system register.

0: Not allowed

1: Allowed

(R/W)

PMS_LP_MM_HP_CLKRST_ALLOW Configures whether the LP CPU in machine mode has permission to access HP_SYS_CLKRST.

0: Not allowed

1: Allowed

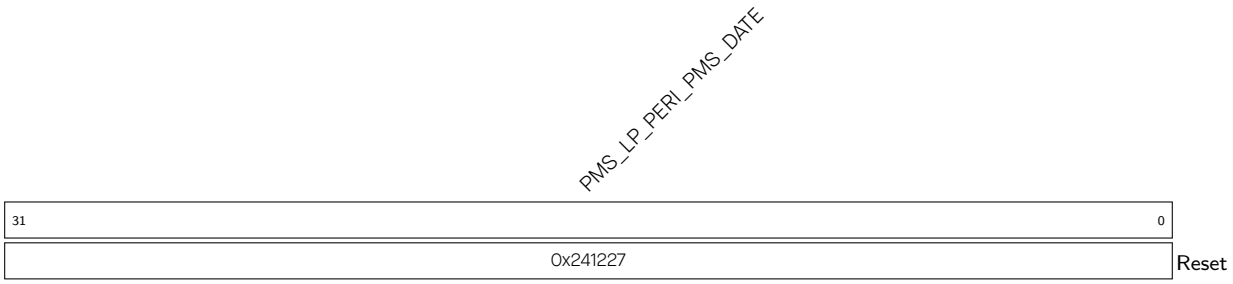
(R/W)

20.7.4 LP_PERI_PMS_REG

The addresses in this section are relative to the LP_PERI_PMS base address provided in Table 9.3-2 in Chapter 9 *System and Memory*.

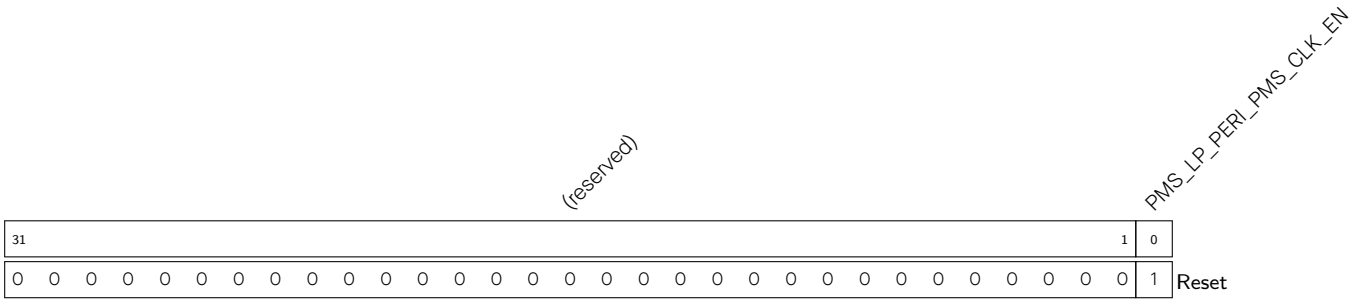
For how to program reserved fields, please refer to Section Programming Reserved Register Field.

Register 20.75. PMS_LP_PERI_PMS_DATE_REG (0x0000)



PMS_LP_PERI_PMS_DATE Version control register (R/W)

Register 20.76. PMS_LP_PERI_PMS_CLK_EN_REG (0x0004)



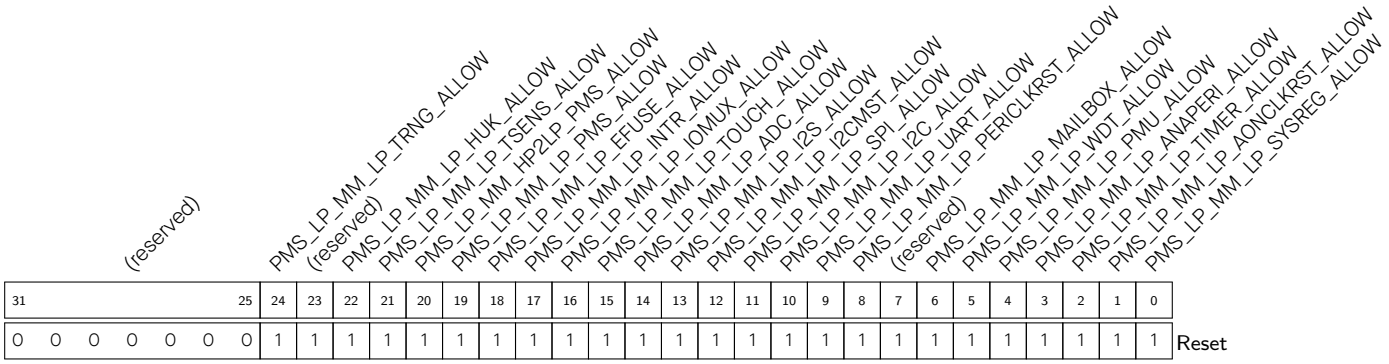
PMS_LP_PERI_PMS_CLK_EN Configures whether to keep the clock always on.

0: Enable automatic clock gating

1: Keep the clock always on

(R/W)

Register 20.77. PMS_LP_MM_LP_PERI_PMS_REG0_REG (0x0008)



PMS_LP_MM_LP_SYSREG_ALLOW Configures whether LP CPU in machine mode has permission to access LP system registers.

0: Not allowed

1: Allowed

(R/W)

PMS_LP_MM_LP_AONCLKRST_ALLOW Configures whether LP CPU in machine mode has permission to access LP_AONCLKRST (LP always-on clock and reset).

0: Not allowed

1: Allowed

(R/W)

PMS_LP_MM_LP_TIMER_ALLOW Configures whether LP CPU in machine mode has permission to access LP timer.

0: Not allowed

1: Allowed

(R/W)

PMS_LP_MM_LP_ANAPERI_ALLOW Configures whether LP CPU in machine mode has permission to access LP ANAPERI (analog peripherals).

0: Not allowed

1: Allowed

(R/W)

PMS_LP_MM_LP_PMU_ALLOW Configures whether LP CPU in machine mode has permission to access LP PMU (Power Management Unit).

0: Not allowed

1: Allowed

(R/W)

PMS_LP_MM_LP_WDT_ALLOW Configures whether LP CPU in machine mode has permission to access LP WDT (watchdog timer).

0: Not allowed

1: Allowed

(R/W)

Continued on the next page...

Register 20.77. PMS_LP_MM_LP_PERI_PMS_REG0_REG (0x0008)

Continued from the previous page...

PMS_LP_MM_LP_MAILBOX_ALLOW Configures whether LP CPU in machine mode has permission to access LP Mailbox Controller.
0: Not allowed
1: Allowed
(R/W)

PMS_LP_MM_LP_PERICLKRST_ALLOW Configures whether LP CPU in machine mode has permission to access LP PREICLKRST (peripheral clock and reset).
0: Not allowed
1: Allowed
(R/W)

PMS_LP_MM_LP_UART_ALLOW Configures whether LP CPU in machine mode has permission to access LP UART.
0: Not allowed
1: Allowed
(R/W)

PMS_LP_MM_LP_I2C_ALLOW Configures whether LP CPU in machine mode has permission to access LP I2S.
0: Not allowed
1: Allowed
(R/W)

PMS_LP_MM_LP_SPI_ALLOW Configures whether LP CPU in machine mode has permission to access LP SPI.
0: Not allowed
1: Allowed
(R/W)

PMS_LP_MM_LP_I2CMST_ALLOW Configures whether LP CPU in machine mode has permission to access LP I2C master.
0: Not allowed
1: Allowed
(R/W)

PMS_LP_MM_LP_I2S_ALLOW Configures whether LP CPU in machine mode has permission to access LP I2S.
0: Not allowed
1: Allowed
(R/W)

Continued on the next page...

Register 20.77. PMS_LP_MM_LP_PERI_PMS_REG0_REG (0x0008)

Continued from the previous page...

PMS_LP_MM_LP_ADC_ALLOW Configures whether LP CPU in machine mode has permission to access LP ADC.
0: Not allowed
1: Allowed
(R/W)

PMS_LP_MM_LP_TOUCH_ALLOW Configures whether LP CPU in machine mode has permission to access LP touch sensor.
0: Not allowed
1: Allowed
(R/W)

PMS_LP_MM_LP_IOMUX_ALLOW Configures whether LP CPU in machine mode has permission to access LP IO MUX.
0: Not allowed
1: Allowed
(R/W)

PMS_LP_MM_LP_INTR_ALLOW Configures whether LP CPU in machine mode has permission to access LP INTR (interrupt).
0: Not allowed
1: Allowed
(R/W)

PMS_LP_MM_LP_EFUSE_ALLOW Configures whether LP CPU in machine mode has permission to access LP eFuse.
0: Not allowed
1: Allowed
(R/W)

PMS_LP_MM_LP_PMS_ALLOW Configures whether LP CPU in machine mode has permission to access LP_PERI_PMS_REG.
0: Not allowed
1: Allowed
(R/W)

PMS_LP_MM_HP2LP_PMS_ALLOW Configures whether LP CPU in machine mode has permission to access HP2LP_PERI_PMS_REG.
0: Not allowed
1: Allowed
(R/W)

Continued on the next page...

Register 20.77. PMS_LP_MM_LP_PERI_PMS_REG0_REG (0x0008)

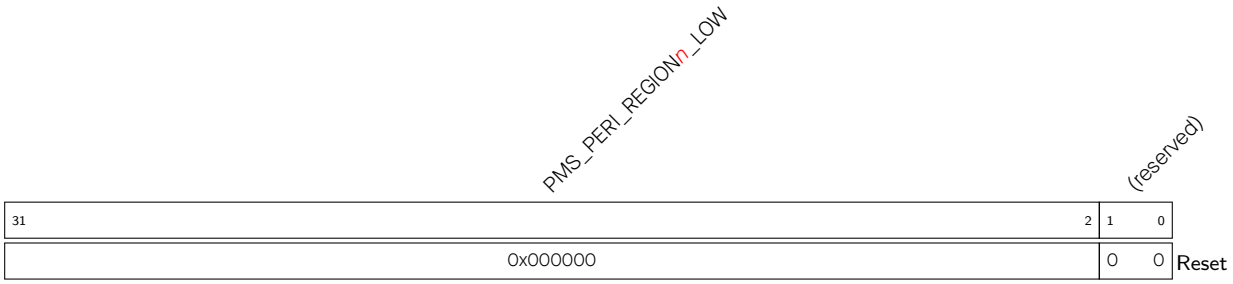
Continued from the previous page...

PMS_LP_MM_LP_TSENS_ALLOW Configures whether LP CPU in machine mode has permission to access LP temperature sensor.
0: Not allowed
1: Allowed
(R/W)

PMS_LP_MM_LP_HUK_ALLOW Configures whether LP CPU in machine mode has permission to access LP HUK (Hardware Unique Key).
0: Not allowed
1: Allowed
(R/W)

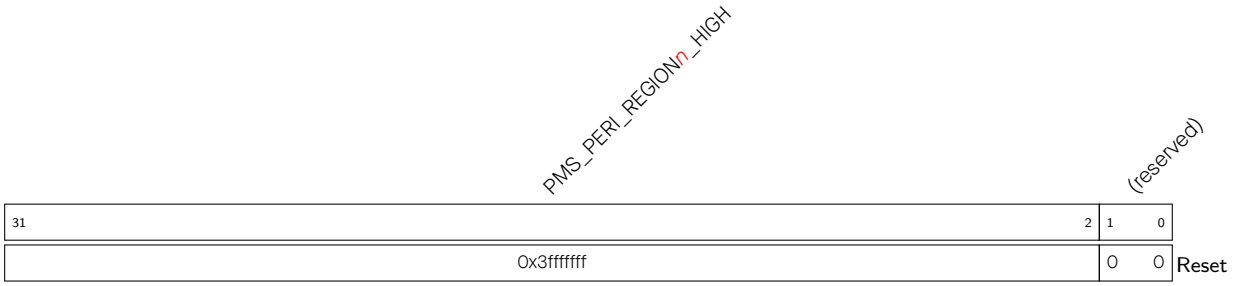
PMS_LP_MM_LP_TRNG_ALLOW Configures whether LP CPU in machine mode has permission to access LP TRNG.
0: Not allowed
1: Allowed
(R/W)

Register 20.78. PMS_PERI_REGION*n*_LOW_REG (*n*: 0-1) (0x000C+0x8**n*)



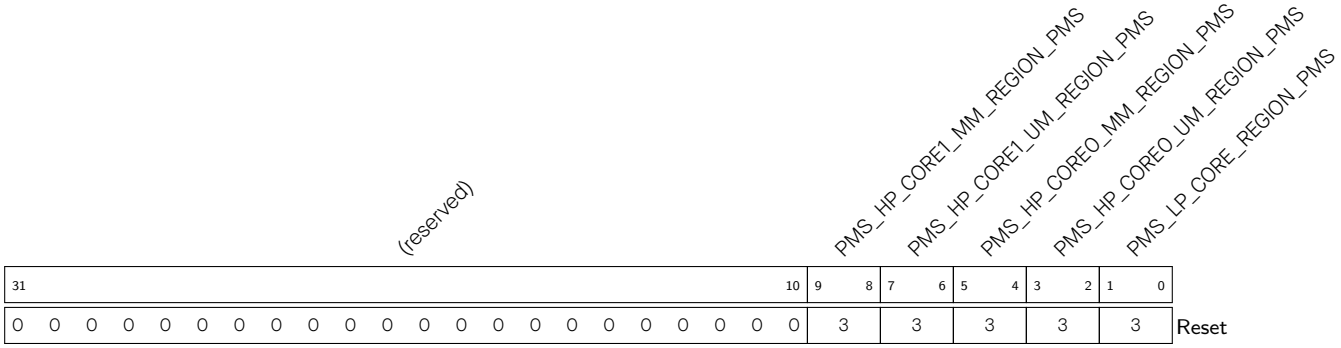
PMS_PERI_REGION*n*_LOW Configures the high 30 bits of the start address of peripheral register's region*n*. (R/W)

Register 20.79. PMS_PERI_REGION*n*_HIGH_REG (*n*: 0-1) (0x0010+0x8n*)**



PMS_PERI_REGION*n*_HIGH Configures the high 30 bits of the end address of peripheral register's region*n*. (R/W)

Register 20.80. PMS_PERI_REGION_PMS_REG (0x001C)



PMS_LP_CORE_REGION_PMS Configures whether LP core in machine mode has permission to access address region0 and address region1. Bit0 corresponds to region0 and bit1 corresponds to region1.

0: Not allowed

1: Allowed

(R/W)

PMS_HP_CORE0_UM_REGION_PMS Configures whether HP CPU0 in user mode has permission to access address region0 and address region1. Bit2 corresponds to region0 and bit3 corresponds to region1.

0: Not allowed

1: Allowed

(R/W)

PMS_HP_CORE0_MM_REGION_PMS Configures whether HP CPU0 in machine mode has permission to access address region0 and address region1. Bit4 corresponds to region0 and bit5 corresponds to region1.

0: Not allowed

1: Allowed

(R/W)

PMS_HP_CORE1_UM_REGION_PMS Configures whether HP CPU1 in user mode has permission to access address region0 and address region1. Bit6 corresponds to region0 and bit7 corresponds to region1.

0: Not allowed

1: Allowed

(R/W)

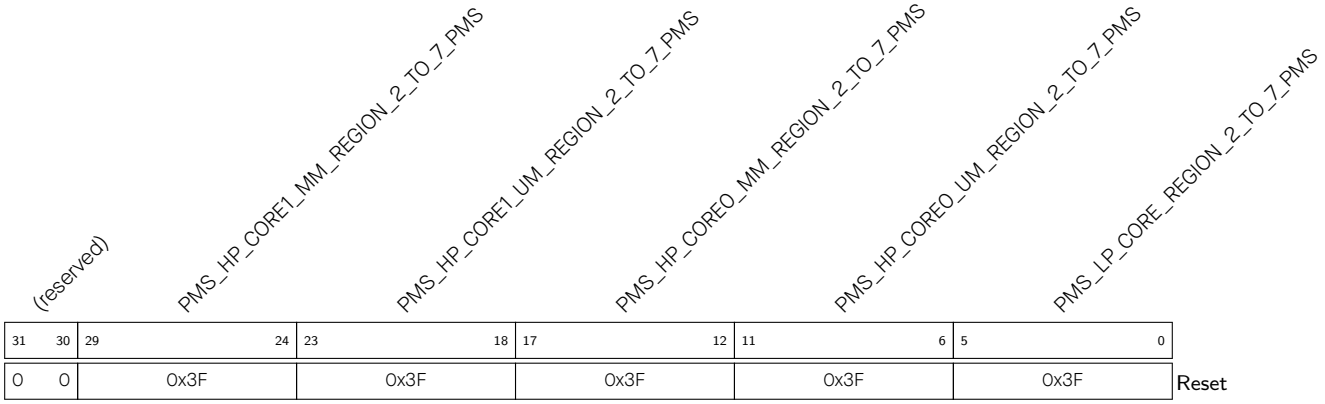
PMS_HP_CORE1_MM_REGION_PMS Configures whether HP CPU1 in machine mode has permission to access address region0 and address region1. Bit8 corresponds to region0 and bit9 corresponds to region1.

0: Not allowed

1: Allowed

(R/W)

Register 20.81. PMS_PERI_REGION_2_TO_7_PMS_REG (0x0020)



PMS_LP_CORE_REGION_2_TO_7_PMS Configures whether LP CPU in machine mode has permission to access address region2 to address region7. Bit0 corresponds to region2 and bit5 corresponds to region7.

0: Not allowed

1: Allowed

(R/W)

PMS_HP_CORE0_UM_REGION_2_TO_7_PMS Configures whether HP CPU0 in user mode has permission to access address region2 to address region7. Bit0 corresponds to region2 and bit5 corresponds to region7.

0: Not allowed

1: Allowed

(R/W)

PMS_HP_CORE0_MM_REGION_2_TO_7_PMS Configures whether HP CPU0 in machine mode has permission to access address region2 to address region7. Bit0 corresponds to region2 and bit5 corresponds to region7.

0: Not allowed

1: Allowed

(R/W)

PMS_HP_CORE1_UM_REGION_2_TO_7_PMS Configures whether HP CPU1 in user mode has permission to access address region2 to address region7. Bit0 corresponds to region2 and bit5 corresponds to region7.

0: Not allowed

1: Allowed

(R/W)

PMS_HP_CORE1_MM_REGION_2_TO_7_PMS Configures whether HP CPU0 in machine mode has permission to access address region2 to address region7. Bit0 corresponds to region2 and bit5 corresponds to region7.

0: Not allowed

1: Allowed

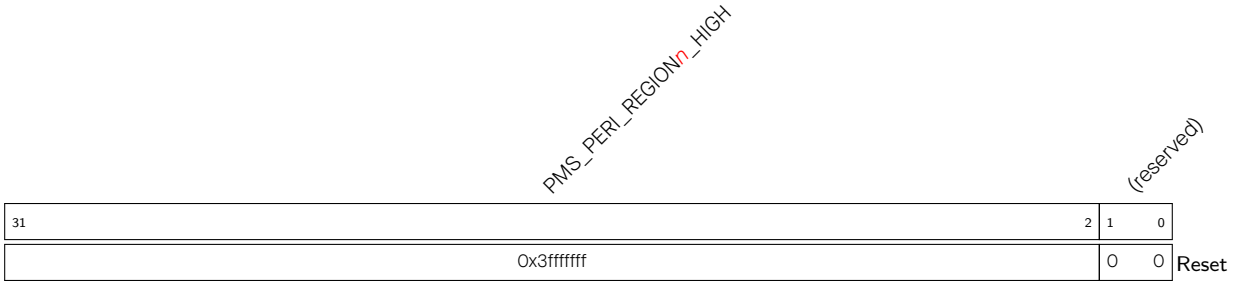
(R/W)

Register 20.82. PMS_PERI_REGION*m*_LOW_REG (*m*: 2-7) (0x0024+0x8*(*m*-2))



PMS_PERI_REGION*m*_LOW Configures the high 30 bits of the start address of peripheral register's region*m*. (R/W)

Register 20.83. PMS_PERI_REGION*m*_HIGH_REG (*m*: 2-7) (0x0028+0x8*(*m*-2))



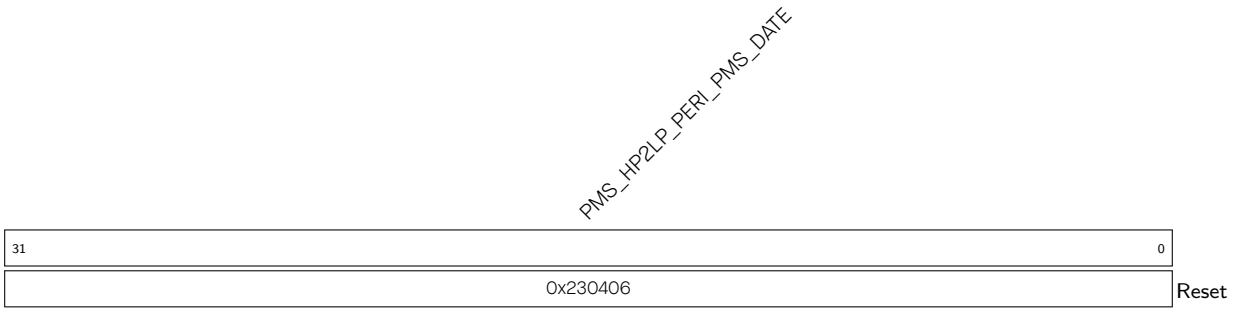
PMS_PERI_REGION*m*_HIGH Configures the high 30 bits of the end address of peripheral register's region*m*. (R/W)

20.7.5 HP2LP_PERI_PMS_REG

The addresses in this section are relative to the H2LP_PERI_PMS base address provided in Table 9.3-2 in Chapter 9 *System and Memory*.

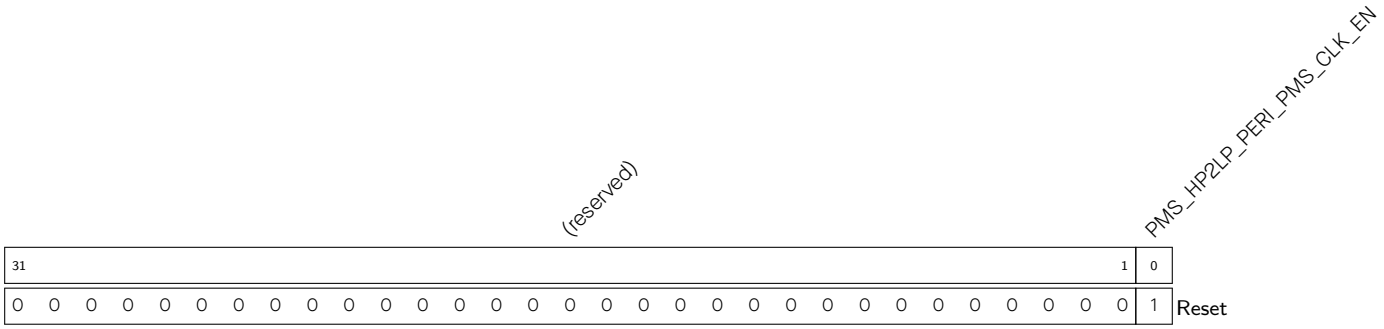
For how to program reserved fields, please refer to Section Programming Reserved Register Field.

Register 20.84. PMS_HP2LP_PERI_PMS_DATE_REG (0x0000)



PMS_HP2LP_PERI_PMS_DATE Version control register (R/W)

Register 20.85. PMS_HP2LP_PERI_PMS_CLK_EN_REG (0x0004)



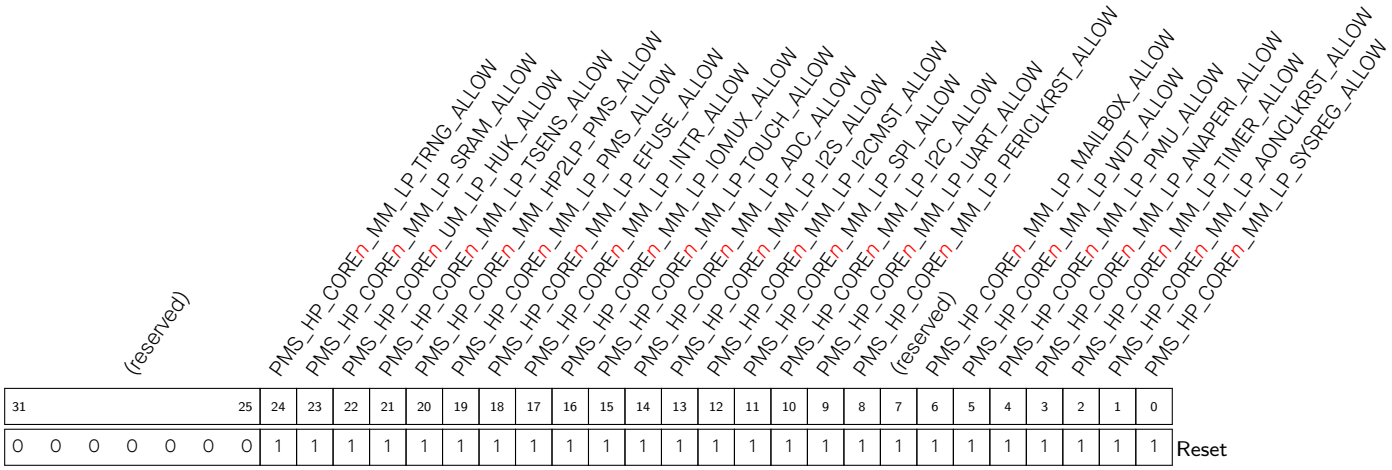
PMS_HP2LP_PERI_PMS_CLK_EN Configures whether to keep the clock always on.

0: Enable automatic clock gating

1: Keep the clock always on

(R/W)

Register 20.86. PMS_HP_CORE*n*_MM_PMS_REG0_REG (*n*: 0-1) (0x0008+0x8**n*)



PMS_HP_CORE*n*_MM_LP_SYSREG_ALLOW Configures whether HP CPU*n* in machine mode has permission to access LP System Registers.

0: Not allowed

1: Allowed

(R/W)

PMS_HP_CORE*n*_MM_LP_AONCLKRST_ALLOW Configures whether HP CPU*n* in machine mode has permission to access LP_AONCLKRST.

0: Not allowed

1: Allowed

(R/W)

PMS_HP_CORE*n*_MM_LP_TIMER_ALLOW Configures whether HP CPU*n* in machine mode has permission to access LP timer.

0: Not allowed

1: Allowed

(R/W)

PMS_HP_CORE*n*_MM_LP_ANAPERI_ALLOW Configures whether HP CPU*n* in machine mode has permission to access LP ANAPERI.

0: Not allowed

1: Allowed

(R/W)

PMS_HP_CORE*n*_MM_LP_PMU_ALLOW Configures whether HP CPU*n* in machine mode has permission to access LP PMU.

0: Not allowed

1: Allowed

(R/W)

Continued on the next page...

Register 20.86. PMS_HP_CORE*n*_MM_PMS_REG0_REG (*n*: 0-1) (0x0008+0x8**n*)

Continued from the previous page...

PMS_HP_CORE*n*_MM_LP_WDT_ALLOW Configures whether HP CPU*n* in machine mode has permission to access LP WDT.
0: Not allowed
1: Allowed
(R/W)

PMS_HP_CORE*n*_MM_LP_MAILBOX_ALLOW Configures whether HP CPU*n* in machine mode has permission to access LP Mailbox Controller.
0: Not allowed
1: Allowed
(R/W)

PMS_HP_CORE*n*_MM_LP_PERICLKRST_ALLOW Configures whether HP CPU*n* in machine mode has permission to access LP PERICLKRST.
0: Not allowed
1: Allowed
(R/W)

PMS_HP_CORE*n*_MM_LP_UART_ALLOW Configures whether HP CPU*n* in machine mode has permission to access LP UART.
0: Not allowed
1: Allowed
(R/W)

PMS_HP_CORE*n*_MM_LP_I2C_ALLOW Configures whether HP CPU*n* in machine mode has permission to access LP I2C.
0: Not allowed
1: Allowed
(R/W)

PMS_HP_CORE*n*_MM_LP_SPI_ALLOW Configures whether HP CPU*n* in machine mode has permission to access LP SPI.
0: Not allowed
1: Allowed
(R/W)

PMS_HP_CORE*n*_MM_LP_I2CMST_ALLOW Configures whether HP CPU*n* in machine mode has permission to access LP I2C master.
0: Not allowed
1: Allowed
(R/W)

Continued on the next page...

Register 20.86. PMS_HP_CORE*n*_MM_PMS_REG0_REG (*n*: 0-1) (0x0008+0x8**n*)

Continued from the previous page...

PMS_HP_CORE*n*_MM_LP_I2S_ALLOW Configures whether HP CPU*n* in machine mode has permission to access LP I2S.
0: Not allowed
1: Allowed
(R/W)

PMS_HP_CORE*n*_MM_LP_ADC_ALLOW Configures whether HP CPU*n* in machine mode has permission to access LP ADC.
0: Not allowed
1: Allowed
(R/W)

PMS_HP_CORE*n*_MM_LP_TOUCH_ALLOW Configures whether HP CPU*n* in machine mode has permission to access LP touch sensor.
0: Not allowed
1: Allowed
(R/W)

PMS_HP_CORE*n*_MM_LP_IOMUX_ALLOW Configures whether HP CPU*n* in machine mode has permission to access LP IO MUX.
0: Not allowed
1: Allowed
(R/W)

PMS_HP_CORE*n*_MM_LP_INTR_ALLOW Configures whether HP CPU*n* in machine mode has permission to access LP INTR.
0: Not allowed
1: Allowed
(R/W)

PMS_HP_CORE*n*_MM_LP_EFUSE_ALLOW Configures whether HP CPU*n* in machine mode has permission to access LP eFuse.
0: Not allowed
1: Allowed
(R/W)

PMS_HP_CORE*n*_MM_LP_PMS_ALLOW Configures whether HP CPU*n* in machine mode has permission to access LP_PERI_PMS_REG.
0: Not allowed
1: Allowed
(R/W)

Continued on the next page...

Register 20.86. PMS_HP_CORE*n*_MM_PMS_REG0_REG (*n*: 0-1) (0x0008+0x8***n*)

Continued from the previous page...

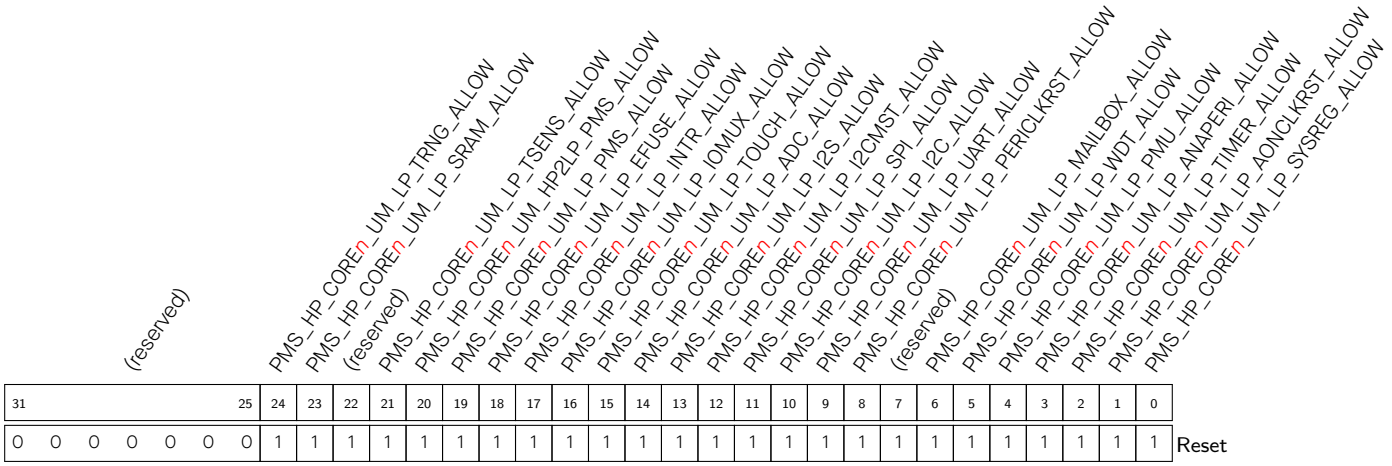
PMS_HP_CORE*n*_MM_HP2LP_PMS_ALLOW Configures whether HP CPU*n* in machine mode has permission to access HP2LP_PERI_PMS_REG.
0: Not allowed
1: Allowed
(R/W)

PMS_HP_CORE*n*_MM_LP_TSENS_ALLOW Configures whether HP CPU*n* in machine mode has permission to access LP temperature sensor.
0: Not allowed
1: Allowed
(R/W)

PMS_HP_CORE*n*_MM_LP_SRAM_ALLOW Configures whether HP CPU*n* in machine mode has permission to access LP SRAM.
0: Not allowed
1: Allowed
(R/W)

PMS_HP_CORE*n*_MM_LP_TRNG_ALLOW Configures whether HP CPU*n* in machine mode has permission to access LP TRNG.
0: Not allowed
1: Allowed
(R/W)

Register 20.87. PMS_HP_CORE*n*_UM_PMS_REG0_REG (*n*: 0-1) (0x000C+0x8**n*)



PMS_HP_CORE*n*_UM_LP_SYSREG_ALLOW Configures whether HP CPU*n* in user mode has permission to access LP System Registers.

0: Not allowed

1: Allowed

(R/W)

PMS_HP_CORE*n*_UM_LP_AONCLKRST_ALLOW Configures whether HP CPU*n* in user mode has permission to access LP_AONCLKRST.

0: Not allowed

1: Allowed

(R/W)

PMS_HP_CORE*n*_UM_LP_TIMER_ALLOW Configures whether HP CPU*n* in user mode has permission to access LP timer.

0: Not allowed

1: Allowed

(R/W)

PMS_HP_CORE*n*_UM_LP_ANAPERI_ALLOW Configures whether HP CPU*n* in user mode has permission to access LP ANAPERI.

0: Not allowed

1: Allowed

(R/W)

PMS_HP_CORE*n*_UM_LP_PMU_ALLOW Configures whether HP CPU*n* in user mode has permission to access LP PMU.

0: Not allowed

1: Allowed

(R/W)

Continued on the next page...

Register 20.87. PMS_HP_CORE*n*_UM_PMS_REG0_REG (*n*: 0-1) (0x000C+0x8**n*)

Continued from the previous page...

PMS_HP_CORE*n*_UM_LP_WDT_ALLOW Configures whether HP CPU*n* in user mode has permission to access LP WDT.
0: Not allowed
1: Allowed
(R/W)

PMS_HP_CORE*n*_UM_LP_MAILBOX_ALLOW Configures whether HP CPU*n* in user mode has permission to access LP Mailbox Controller.
0: Not allowed
1: Allowed
(R/W)

PMS_HP_CORE*n*_UM_LP_PERICLKRST_ALLOW Configures whether HP CPU*n* in user mode has permission to access LP PERICLKRST.
0: Not allowed
1: Allowed
(R/W)

PMS_HP_CORE*n*_UM_LP_UART_ALLOW Configures whether HP CPU*n* in user mode has permission to access LP UART.
0: Not allowed
1: Allowed
(R/W)

PMS_HP_CORE*n*_UM_LP_I2C_ALLOW Configures whether HP CPU*n* in user mode has permission to access LP I2C.
0: Not allowed
1: Allowed
(R/W)

PMS_HP_CORE*n*_UM_LP_SPI_ALLOW Configures whether HP CPU*n* in user mode has permission to access LP SPI.
0: Not allowed
1: Allowed
(R/W)

PMS_HP_CORE*n*_UM_LP_I2CMST_ALLOW Configures whether HP CPU*n* in user mode has permission to access LP I2C master.
0: Not allowed
1: Allowed
(R/W)

Continued on the next page...

Register 20.87. PMS_HP_CORE*n*_UM_PMS_REG0_REG (*n*: 0-1) (0x000C+0x8**n*)

Continued from the previous page...

PMS_HP_CORE*n*_UM_LP_I2S_ALLOW Configures whether HP CPU*n* in user mode has permission to access LP I2S.
0: Not allowed
1: Allowed
(R/W)

PMS_HP_CORE*n*_UM_LP_ADC_ALLOW Configures whether HP CPU*n* in user mode has permission to access LP ADC.
0: Not allowed
1: Allowed
(R/W)

PMS_HP_CORE*n*_UM_LP_TOUCH_ALLOW Configures whether HP CPU*n* in user mode has permission to access LP touch sensor.
0: Not allowed
1: Allowed
(R/W)

PMS_HP_CORE*n*_UM_LP_IOMUX_ALLOW Configures whether HP CPU*n* in user mode has permission to access LP IO MUX.
0: Not allowed
1: Allowed
(R/W)

PMS_HP_CORE*n*_UM_LP_INTR_ALLOW Configures whether HP CPU*n* in user mode has permission to access LP INTR.
0: Not allowed
1: Allowed
(R/W)

PMS_HP_CORE*n*_UM_LP_EFUSE_ALLOW Configures whether HP CPU*n* in user mode has permission to access LP eFuse.
0: Not allowed
1: Allowed
(R/W)

PMS_HP_CORE*n*_UM_LP_PMS_ALLOW Configures whether HP CPU*n* in user mode has permission to access LP_PERI_PMS_REG.
0: Not allowed
1: Allowed
(R/W)

Continued on the next page...

Register 20.87. PMS_HP_CORE*n*_UM_PMS_REG0_REG (*n*: 0-1) (0x000C+0x8**n*)

Continued from the previous page...

PMS_HP_CORE*n*_UM_HP2LP_PMS_ALLOW Configures whether HP CPU*n* in user mode has permission to access HP2LP_PERI_PMS_REG.
0: Not allowed
1: Allowed
(R/W)

PMS_HP_CORE*n*_UM_LP_TSENS_ALLOW Configures whether HP CPU*n* in user mode has permission to access LP temperature sensor.
0: Not allowed
1: Allowed
(R/W)

PMS_HP_CORE*n*_UM_LP_HUK_ALLOW Configures whether HP CPU*n* in user mode has permission to access LP HUK.
0: Not allowed
1: Allowed
(R/W)

PMS_HP_CORE*n*_UM_LP_SRAM_ALLOW Configures whether HP CPU*n* in user mode has permission to access LP SRAM.
0: Not allowed
1: Allowed
(R/W)

PMS_HP_CORE*n*_UM_LP_TRNG_ALLOW Configures whether HP CPU*n* in user mode has permission to access LP TRNG.
0: Not allowed
1: Allowed
(R/W)

Chapter 21

System Registers (SYSREG)

21.1 Overview

ESP32-P4 includes various features that can be configured through dedicated registers. This chapter mainly enumerates such registers. For detailed introduction about a specific feature, please go to respective chapters.

21.2 Function Description

21.2.1 HP System Registers

21.2.1.1 External Memory Encryption/Decryption Configuration

Register HP_SYSTEM_CRYPTO_CTRL_REG configures encryption and decryption options of the external memory (external Flash and RAM). For details, please refer to Chapter 34 *External Memory Encryption and Decryption (XTS_AES)*.

21.2.1.2 HP Peripherals Clock Configuration and Power Control

The following registers are used to control the power signals of memory in corresponding peripherals.

- RSA: HP_SYSTEM_HP_RSA_PD_CTRL_REG
- ECC: HP_SYSTEM_HP_ECC_PD_CTRL_REG
- RSA: HP_SYSTEM_HP_UART_PD_CTRL_REG

The following fields in register HP_SYSTEM_HP_PERI_MEM_CLK_FORCE_ON_REG control the clock gating signals of memory in corresponding peripherals.

- GDMA: HP_SYSTEM_GDMA_MEM_CLK_FORCE_ON
- BitScrambler
  - TX memory: HP_SYSTEM_BITSCRAMBLER_TX_MEM_CLK_FORCE_ON
  - RX memory: HP_SYSTEM_BITSCRAMBLER_RX_MEM_CLK_FORCE_ON
- RMT: HP_SYSTEM_RMT_MEM_CLK_FORCE_ON

21.2.1.3 HP Cache Clock and Reset Configuration

ESP32-P4 allows software to directly control the clock and reset signals of HP cache.

The following fields in register HP_SYSTEM_HP_CACHE_CLK_CONFIG_REG control the clock signals of different HP cache. Writing 1 to respective fields enables the clock for respective cache.

- L1 ichache 0: HP_SYSTEM_L1_I0_CACHE_CLK_ON
- L1 ichache 1: HP_SYSTEM_L1_I1_CACHE_CLK_ON
- L1 dcache: HP_SYSTEM_L1_D_CACHE_CLK_ON
- L2 cache: HP_SYSTEM_L2_CACHE_CLK_ON

The following fields in register HP_SYSTEM_CACHE_RESET_CONFIG_REG control the reset signals of different HP cache. Writing 1 and then 0 to respective fields resets respective cache.

- L1 ichache 0: HP_SYSTEM_L1_I0_CACHE_RESET
- L1 ichache 1: HP_SYSTEM_L1_I1_CACHE_RESET
- L1 dcache: HP_SYSTEM_L1_D_CACHE_RESET

21.2.1.4 HP SPM and HP L2MEM Clock Configuration

The following register fields control the clock signals of HP SPM (Scratchpad Memory) and HP L2MEM. Writing 1 to respective register fields forcibly enable the clock of respective memory.

- HP SPM: HP_SYSTEM_HP_SPM_CLK_FORCE_ON
- L2MEM: HP_SYSTEM_L2_MEM_CLK_FORCE_ON
- L2ROM: HP_SYSTEM_L2_ROM_CLK_FORCE_ON

21.2.1.5 HP SPM Parity Check Configuration

ESP32-P4's has implemented Parity Check in HP SPM to detect one-bit memory flip error.

This feature can be configured and controlled via the following registers:

- HP_SYSTEM_HP_SPM_PARITY_CHECK_CTRL_REG: write 1 to enable the parity check in HP SPM.
- HP_SYSTEM_HP_SPM_INIT_REG: configures the initialization of HP SPM before enabling the parity check.
- Interrupt (HP_SPM_PARITY_ERR_INT) related registers:
  - HP_SYSTEM_HP_SPM_INT_RAW_REG: the raw interrupt status of HP_SPM_PARITY_ERR_INT
  - HP_SYSTEM_HP_SPM_INT_ST_REG: the masked interrupt status of HP_SPM_PARITY_ERR_INT
  - HP_SYSTEM_HP_SPM_INT_ENA_REG: write 1 to enable HP_SPM_PARITY_ERR_INT
  - HP_SYSTEM_HP_SPM_INT_CLR_REG: write 1 to clear HP_SPM_PARITY_ERR_INT
- HP_SYSTEM_HP_SPM_PARITY_INT_RECORD_REG: records the address when HP_SPM_PARITY_ERR_INT occurs.
- HP_SYSTEM_HP_SPM_ERR_RESP_CTRL_REG: write 1 so HP SPM reports parity error to the HP CPU and triggers exception.

For detailed configuration steps, please refer to Chapter 9 *System and Memory*.

21.2.1.6 HP L2MEM ECC Check Configuration

ESP32-P4's has implemented the Error Correction Code (ECC) check in HP L2MEM (including L2RAM and cache) to correct one-bit memory flip error.

This feature can be configured and controlled via the following registers:

- HP_SYSTEM_L2_MEM_L2_RAM_ECC_REG and HP_SYSTEM_L2_MEM_L2_CACHE_ECC_REG: write 1 to enable the ECC check for L2MEM and cache respectively.
- HP_SYSTEM_L2_MEM_REFRESH_REG: configures the initialization of HP L2MEM before enabling the ECC check.
- Interrupt (L2_MEM_ECC_ERR_INT) related registers:
  - HP_SYSTEM_L2_MEM_INT_RAW_REG: the raw interrupt status of L2_MEM_ECC_ERR_INT
  - HP_SYSTEM_L2_MEM_INT_ST_REG: the masked interrupt status of L2_MEM_ECC_ERR_INT
  - HP_SYSTEM_L2_MEM_INT_ENA_REG: write 1 to enable L2_MEM_ECC_ERR_INT
  - HP_SYSTEM_L2_MEM_INT_CLR_REG: write 1 to clear L2_MEM_ECC_ERR_INT
- HP_SYSTEM_L2_MEM_INT_RECORD1_REG: records information when L2_MEM_ECC_ERR_INT occurs.
- HP_SYSTEM_L2_MEM_ERR_RESP_CTRL_REG: write 1 so HP L2MEM reports ECC error to HP CPU and triggers exception.

For detailed configuration steps, please refer to Chapter 9 *System and Memory*.

21.2.1.7 BitScrambler Configuration

ESP32-P4 has some peripherals that support DMA. These peripherals sometimes need BitScramblers to change the format of data before communicating with DMA.

Configure HP_SYSTEM_BITSCRAMBLER_PERI_SEL_REG to attach such a peripheral to the RX and TX channels of bitscamblers respectively.

For detailed configuration steps, please refer to Chapter 62 *BitScrambler*.

For a list of DMA-capable peripherals, please refer to Chapter 11 *Reset and Clock*.

21.2.1.8 Ethernet MAC Control

ESP32-P4 can send and receive data via Ethernet MAC (Media Access Controller) according to the IEEE 802.3 standard.

The control and configuration registers are:

- HP_SYSTEM_GMAC_CTRL0_REG
- HP_SYSTEM_GMAC_CTRL1_REG
- HP_SYSTEM_GMAC_CTRL2_REG

For details, please refer to Chapter 55 *Ethernet Media Access Controller (EMAC)*.

21.2.1.9 USB 2.0 High-Speed OTG Control

ESP32-P4 features a USB 2.0 High-Speed OTG peripheral (OTG_HS). To control it, use HP_SYSTEM_USBOTG20_CTRL_REG.

Register HP_SYSTEM_USB_OTGHS_PHY_ST_REG represents the USB OTGHS PHY status.

For details, please refer to Chapter *52 USB 2.0 High-Speed OTG*.

21.2.1.10 CPU Control and Record

ESP32-P4 configures CPU debug features and stores its status information via the following system registers.

- HP_SYSTEM_CPU_CORESTALLED_ST_REG: records the core stalled status of CPU0 and CPU1.
- HP_SYSTEM_CORE_DEBUG_RUNSTALL_CONF_REG: write 1 to enable debug runstall feature between HP CPU and LP CPU.
- HP_SYSTEM_HP_CORE_DMACTIVE_LPCORE_REG: represents the dmactive value of LP CPU's debug module.
- HP_SYSTEM_HP_CPU_WAKEUP_EVENT_REG: write 1 to wake up HP CPU0 or HP CPU1.

21.2.1.11 HP to LP Interrupt and Wakeup Control

HP system registers provide grouped control for routing peripheral interrupts and wakeup events from the HP domain to the LP CPU and PMU.

- Interrupt enable: HP_SYSTEM_HP2LP_INTR_GROUP0_EN_REG ~ HP_SYSTEM_HP2LP_INTR_GROUP3_EN_REG
- Interrupt status: HP_SYSTEM_HP2LP_INTR_GROUP0_ST_REG ~ HP_SYSTEM_HP2LP_INTR_GROUP3_ST_REG
- Wakeup enable: HP_SYSTEM_HP2LP_WAKEUP_GROUP0_EN_REG ~ HP_SYSTEM_HP2LP_WAKEUP_GROUP3_EN_REG

For details, please refer to Chapter *2 High-Performance CPU*.

21.2.1.12 HP GPIO Hold Control

Each GPIO pin of ESP32-P4 has an individual hold function that is controlled by the following registers:

- HP_SYSTEM_GPIO_O_HOLD_CTRL0_REG: controls the hold signal of HP GPIO16 ~ HP GPIO47
- HP_SYSTEM_GPIO_O_HOLD_CTRL1_REG: controls the hold signal of GPIO48 ~ HP GPIO56
- HP_SYSTEM_GPIO_DED_HOLD_CTRL_REG: the lower 6 bits control the hold signal of flash pads, and the higher 20 bits control the hold signal of 16 line PSRAM pad

For details, please refer to Chapter *10 GPIO Matrix and IO MUX*.

21.2.1.13 HP GPIO Output Control

Each GPIO pin of ESP32-P4 can be configured to directly output the desired value via the following registers (hysteresis):

- HP_SYSTEM_GPIO_O_HYS_CTRL0_REG: configures whether to enable the hysteresis feature of HP GPIO16 ~ HP GPIO47.
- HP_SYSTEM_GPIO_O_HYS_CTRL1_REG: configures whether to enable the hysteresis feature of HP GPIO48 ~ HP GPIO56.

For details, please refer to Section 10.10 in Chapter 10 *GPIO Matrix and IO MUX*.

21.2.1.14 Illegal Access and Unauthorized Access

ESP32-P4's buses return error response and record transfer address, transfer direction, and master id when illegal access and unauthorized access occur.

- Illegal access: access to unmapped address space
- Unauthorized access: access to mapped address space without permission

Such information will be logged in the following registers:

- HP CPU and other AHB masters (ICM_CPU_ADDRHOLE_INT)
  - HP_SYSTEM_ICM_CPU_ADDRHOLE_ADDR_REG
  - HP_SYSTEM_ICM_CPU_ADDRHOLE_INFO_REG
- AXI masters (ICM_SYS_ADDRHOLE_INT)
  - HP_SYSTEM_ICM_SYS_ADDRHOLE_ADDR_REG
  - HP_SYSTEM_ICM_SYS_ADDRHOLE_INFO_REG

For detailed information on address space mapping, please refer to Chapter 9 *System and Memory*.

For detailed information on access permission management, please refer to Chapter 20 *Permission Control (PMS)*.

21.2.1.15 HP CPU Bus Timeout Protection

ESP32-P4 supports Timeout Protection on HP CPU's IBUS, DBUS and AHB bus.

When a transfer is initiated, the counter inside the Timeout Protection module increments by one on each clock cycle.

- If the accumulated value remains below the timeout threshold and the slave device responds to the transfer, the internal counter is cleared.
- If the accumulated value exceeds the timeout threshold and the slave device has not responded, the Timeout Protection module takes control of the bus to return an error response to the HP CPU. At the same time, an interrupt is generated.

Related control and configuration registers are described below.

HP CPU IBUS Timeout Protection

Configure HP_SYSTEM_HP_CORE_IBUS_TIMEOUT_REG to enable timeout protection and set the timeout threshold for HP CPU accessing HP SPM, L2MEM and EXT MEM by IBUS.

When a timeout occurs, the HP_CORE*x*_IBUS_TIMEOUT_INT interrupt will be asserted.

HP CPU DBUS Timeout Protection

Configure HP_SYSTEM_HP_CORE_DBUS_TIMEOUT_REG to enable timeout protection and set the timeout threshold for HP CPU accessing HP SPM, L2MEM and EXT MEM by DBUS.

When a timeout occurs, the HP_CORE*x*_DBUS_TIMOUT_INT interrupt will be asserted.

HP CPU AHB BUS Timeout Protection

Configure HP_SYSTEM_HP_CORE_AHB_TIMEOUT_REG to enable timeout protection and set the timeout threshold for HP CPU accessing LP peripherals, HP peripherals registers by AHB bus.

When a timeout occurs, the HP_CORE*x*_AHB_TIMEOUT_INT interrupt will be asserted.

21.2.1.16 AXI Matrix

ESP32-P4's AXI Matrix connects multiple AXI-compliant master and slave devices, enabling efficient communication.

- Masters such as all AHB masters (master port1 through AHB2AXI bridge), HP CPU cache (master port2), DW GDMA_0 (master port6), DW GDMA_1 (master port7), AXI GDMA (master port9), 2DDMA (master port11), H264_0 (master port12), and H264_1 (master port13)
- Slaves such as L2MEM (slave port2), Flash (slave port3), PSRAM (slave port4), DSI Host (slave port6) and CSI Host(slave port7).

All AXI master ports except master port 2 in the AXI matrix support the QoS (Quality of Service) bandwidth regulator. The AXI matrix also includes a deadlock timeout protection mechanism, slave arbiter priority configuration for slave ports, and QoS arbiter priority configuration for master ports.

Deadlock Timeout Protection

ESP32-P4's Deadlock Timeout Protection is always enabled. Configure the timeout threshold for all AXI masters in the AXI matrix that access AXI Slaves such as Flash, PSRAM, DSI HOST, and CSI HOST via HP_SYSTEM_ICM_DLOCK_TIMEOUT_REG.

When a timeout occurs, AXI Matrix will record transfer direction, AXI ID, master ID and slave ID in HP_SYSTEM_ICM_DLOCK_STATUS_REG, and assert ICM_DLOCK_INTR interrupt at the same time.

Slave Arbiter Priority

Configure the arbitration priority for slaves in the AXI Matrix to arbitrate reading data and responses via HP_SYSTEM_ICM_SLV_ARB_PRIORITY_REG.

The higher the value, the higher the priority.

Master QoS Arbiter Priority

Configure the QoS (Quality of Service) values for masters in the AXI matrix to arbitrate the read address channel and write address channel via HP_SYSTEM_ICM_MST_ARQOS_REG0_REG and HP_SYSTEM_ICM_MST_AWQOS_REG0_REG, respectively.

A higher QoS value indicates a higher priority during arbitration.

21.2.1.17 Post Write

The following registers controls the Post Write mode of different components to speed up the write access.

- HP_SYSTEM_PERI1_APB_POSTW_EN_REG: enables the Post Write mode for APB2APB bridge between peripherals.
- HP_SYSTEM_APB_SYNC_POSTW_EN_REG: enables the Post Write mode for APB2APB bridge in DSI, CSI and ISP.
- HP_SYSTEM_CPU_ICM_H2X_POST_WR_EN: enables the Post Write mode for AHB2AXI bridge from AHB masters to AXI slaves.

21.2.2 LP System Registers

21.2.2.1 LP Timer Stall Control

The LP_SYSTEM_SYSTIMER_STALL_SEL field allows selection of the systimer_stall signal source for the LP timer, choosing between HP CPU0 and HP CPU1:

- 0: from CPU0
- 1: from CPU1

21.2.2.2 Focused Ion Beam (FIB) Control

The LP_SYSTEM_LP_FIB_SEL field allows selection of the control source for the brownout detector and the super watchdog, choosing between the analog system or software register.

- Bit-1: selects the control source for the brownout detector.
  - 0: controlled by software register
  - 1: controlled by the analog system.
- Bit-2: selects the control source for the super watchdog.
  - 0: controlled by software register
  - 1: controlled by the analog system.

Configure LP_SYSTEM_DIG_FIB to enable the brownout detector or the super watchdog via software register.

- Bit-1: write 1 to enable the brownout detector.
- Bit-2: write 1 to enable the super watchdog.

Read LP_SYSTEM_ANA_FIB to check the FIB state of brownout detector or the super watchdog via analog system (FIB state).

- Bit-1: brownout detector
- Bit-2: super watchdog

21.2.2.3 Boot Mode

Configure LP_SYSTEM_FORCE_DOWNLOAD_BOOT to forcibly switch the chip from SPI Boot mode to Joint Download Boot mode.

21.2.2.4 LP CPU Control

Configure the LP CPU via LP_SYSTEM_SYS_CTRL_REG. In particular:

- LP_SYSTEM_LP_CPU_BOOT_ADDR: configures the boot address of the LP CPU.
- LP_SYSTEM_LP_CPU_EXC_PC_REG: represents the exception PC address of the LP CPU.
- LP_SYSTEM_LP_CPU_DBG_PC_REG: represents the current PC address of the LP CPU.
- LP_SYSTEM_LP_CORE_ERR_RESP_DIS: configures whether or not to disable error response for respective LP CPU buses.

  - Bit-0: IBUS
  - Bit-1: DBUS
  - Bit-2: AHB bus

- LP_SYSTEM_LP_CORE_ETM_WAKEUP_FLAG: indicates the status of an ETM task triggered by a wakeup event of the LP CPU.
- LP_SYSTEM_LP_CORE_ETM_WAKEUP_FLAG_CLR: write 1 to clear LP_SYSTEM_LP_CORE_ETM_WAKEUP_FLAG.

21.2.2.5 Reset Control

ESP32-P4 supports for types of resets that occur at different levels. Such reset can be controlled by some system registers introduced in the following sections.

For a full list of all the reset types and reset sources, please refer to Chapter 11 *Reset and Clock*.

Analog and System Reset Source Bypass Control

LP_SYSTEM_ANA_RST_BYPASS: configures whether to bypass the following reset sources:

- Brown-out detector
- Super watchdog
- Power glitch detector

LP_SYSTEM_SYS_RST_BYPASS: configures whether to bypass the following reset sources:

- Software
- HP Watchdog timer
- LP Watchdog timer
- eFuse

Power-On Reset Bypass Control

LP_SYSTEM_HP_PO_RSTN_BYPASS_CTRL: configures whether or not to bypass the following reset sources for the HP power domain during the Power-On reset.

- Bit-0: eFuse
- Bit-1: LP watchdog timer
- Bit-2: USB (JTAG)
- Bit-3: USB (UART)
- Bit-4: Unused
- Bit-5: Unused
- Bit-6: HP watchdog timer
- Bit-7: Software

LP_SYSTEM_HP_PO_CNNT_RSTN_BYPASS_CTRL: configures whether or not to bypass the following reset sources for the HP CNNT power domain during the Power-On reset.

- Bit-0: eFuse
- Bit-1: LP watchdog timer
- Bit-2: USB (JTAG)
- Bit-3: USB (UART)
- Bit-4: Unused
- Bit-5: Unused
- Bit-6: HP watchdog timer
- Bit-7: Software

For details about ESP32-P4's power domains, please refer to Chapter 15 *Low-Power Management*.

Software System Reset

Configure LP_SYSTEM_SYS_SW_RST to trigger a software system reset.

IOMUX Reset

Configure LP_SYSTEM_IO_MUX_RESET_DISABLE to disable the HP IOMUX reset.

21.2.2.6 Wakeup Control

These registers control the behavior of some PMU wakeup sources.

LP_SYSTEM_USBOTG20_IN_SUSPEND: configures whether or not to enable wakeup signals from USB OTG2.0.

LP_SYSTEM_USBOTG20_WAKEUP_CLR: write 1 to clear wakeup signals from USB OTG2.0 to PMU.

LP_SYSTEM_USBOTG20_LS_MODE: configures whether USB OTG2.0 operates in low-speed mode.

Register LP_SYSTEM_HP_USB_OTGHS_PHY_CTRL_REG controls the USB OTGHS PHY pull-up and pull-down signals.

21.2.2.7 Pad Hold Control

The following registers control pad hold during sleep or reset:

- LP_SYSTEM_PAD_RTC_HOLD_CTRL0_REG: controls pad 0 ~ pad 31
- LP_SYSTEM_PAD_RTC_HOLD_CTRL1_REG: controls pad 32 ~ pad 56
- LP_SYSTEM_DED_PAD_RTC_HOLD_CTRL_REG: controls flash and PSRAM pads

21.2.2.8 Power Discharge Control

Register LP_SYSTEM_DISCHARGE_REG controls discharge of LDO flash and LP PUF memory.

21.2.2.9 LPROM and LP SPM Clock Control

LP_SYSTEM_LP_SPM_ROM_CLK_FORCE_ON: set to force on LP SPM_ROM clock if LP CPU clock is on.

LP_SYSTEM_LP_SPM_RAM_CLK_FORCE_ON: set to force on LP SPM_RAM clock.

21.2.2.10 Analog Voltage Comparator

Analog voltage comparator*x* (*x* = 0, 1) are configured by the following register fields:

- LP_SYSTEM_XPD_COMP*x* (*x* = 0, 1): configures whether or not to enable analog voltage comparator*x*.
- LP_SYSTEM_MODE_COMP*x* (*x* = 0, 1): configures the comparison mode:
  - 0: comparing the main voltage with the external reference voltage
  - 1: comparing the main voltage with the internal reference voltage
- LP_SYSTEM_DREF_COMP*x* (*x* = 0, 1): configures the internal reference voltage.
  - The adjustable voltage range is 0 to (0.7 * VDDPST_6) V.
  - The adjystable step is 0.1 * VDDPST_6 V.

For details, please refer to Chapter 66 *Analog Voltage Comparator*.

21.2.2.11 Illegal Access and Unauthorized Access

ESP32-P4's buses return error response and record transfer address, transfer direction, and master id when illegal access and unauthorized access occur.

- Illegal access: access to unmapped address space
- Unauthorized access: access to mapped address space without permission

Such information will be logged in the following registers:

- For LP CPU's access by IBUS and DBUS (IDBUS_ADDRHOLE_INTR)
  - LP_SYSTEM_IDBUS_ADDRHOLE_ADDR_REG
  - LP_SYSTEM_IDBUS_ADDRHOLE_INFO_REG
- For LP CPU's access by AHB or an access from HP CPU (LP_ADDRHOLE_INTR):
  - LP_SYSTEM_LP_ADDRHOLE_ADDR_REG
  - LP_SYSTEM_LP_ADDRHOLE_INFO_REG

For detailed information on address space mapping, please refer to Chapter 9 *System and Memory*.

For detailed information on access permission management, please refer to Chapter 20 *Permission Control (PMS)*.

21.2.2.12 LP Bus Timeout Protection

ESP32-P4 supports Timeout Protection on LP CPU's IBUS, DBUS and AHB bus.

When a transfer is initiated, the counter inside the Timeout Protection module increments by one on each clock cycle.

- If the accumulated value remains below the timeout threshold and the slave device responds to the transfer, the internal counter is cleared.
- If the accumulated value exceeds the timeout threshold and the slave device has not responded, the Timeout Protection module takes control of the bus to return an error response to the LP CPU. At the same time, an interrupt is generated.

Related control and configuration registers are described below.

LP CPU IBUS Timeout Protection

Configure LP_SYSTEM_LP_CORE_IBUS_TIMEOUT_REG to enable timeout protection and set the timeout threshold for LP CPU accessing LPROM and LP SPM by IBUS. When a timeout occurs, the LP_CORE_IBUS_TIMEOUT_INTR interrupt will be asserted.

LP CPU DBUS Timeout Protection

Configure LP_SYSTEM_LP_CORE_DBUS_TIMEOUT_REG to enable timeout protection and set the timeout threshold for LP CPU accessing LPROM and LP SPM by DBUS.

When a timeout occurs, the LP_CORE_DBUS_TIMEOUT_INTR interrupt will be asserted.

LP CPU AHB Bus Timeout Protection

Configure LP_SYSTEM_LP_CORE_AHB_TIMEOUT_REG to enable timeout protection and set the timeout threshold for LP CPU accessing LP peripherals, HP peripherals and CPU peripherals by AHB bus.

When a timeout occurs, the LP_CORE_AHB_TIMEOUT_INTR interrupt will be asserted.

21.2.2.13 RNG Control

Registers LP_SYSTEM_RNG_DATA_REG and LP_SYSTEM_RNG_CFG_REG are used to configure and control the random number generator.

21.3 Interrupt

ESP32-P4's System Registers can generate the following interrupt signal(s) that will be sent to the Interrupt Matrix.

- HP_SYSREG_INTR
- SYS_ICM_INTR
- LP_SYSREG_INTR

There are several internal interrupt sources from System Registers that can generate the above interrupt signal(s). The interrupt sources from System Registers are listed with their trigger conditions and the resulted interrupt signal(s) in Table 21.3-1.

Table 21.3-1. System Registers Interrupt Sources

Internal Interrupt Source	Trigger Condition	Interrupt Signal

CPU_INT_FROM_CPU0	Software interrupt. Triggered by writing 1 to HP_SYSTEM_HP_CPU_INT_FROM_CPU_0_REG	HP_SYSREG_INTR
CPU_INT_FROM_CPU1	Software interrupt. Triggered by writing 1 to HP_SYSTEM_HP_CPU_INT_FROM_CPU_1_REG	
CPU_INT_FROM_CPU2	Software interrupt. Triggered by writing 1 to HP_SYSTEM_HP_CPU_INT_FROM_CPU_2_REG	
CPU_INT_FROM_CPU3	Software interrupt. Triggered by writing 1 to HP_SYSTEM_HP_CPU_INT_FROM_CPU_3_REG	
L2_MEM_ERR_RESP_INT	Triggered when an error response occurs in L2MEM	
L2_MEM_EXCEED_ADDR_INT	Triggered when L2MEM exceeds its address range	
L2_MEM_ECC_ERR_INT	Triggered when a one-bit flip error is detected and corrected by ECC in L2MEM	
HP_SPM_PARITY_ERR_INT	Triggered when a parity error occurs in HP SPM	
CPU_ICM_H2X_BRESP_ERR_INT	Triggered when a BRESP error occurs in Post Write mode in AHB2AXI	
HP_CORE1_DBUS_TIMEOUT_INT	Triggered upon HP CPU1 DBUS timeout	
HP_CORE0_DBUS_TIMEOUT_INT	Triggered upon HP CPU0 DBUS timeout	
HP_CORE1_IBUS_TIMEOUT_INT	Triggered upon HP CPU1 IBUS timeout	
HP_CORE0_IBUS_TIMEOUT_INT	Triggered upon HP CPU0 IBUS timeout	
HP_CORE1_AHB_TIMEOUT_INT	Triggered upon HP CPU1 AHB bus timeout	
HP_CORE0_AHB_TIMEOUT_INT	Triggered upon HP CPU0 AHB bus timeout	
ICM_CPU_ADDRHOLE_INT	Triggered when illegal/unauthorized access occurs in HP CPU and AHB matrix	SYS_ICM_INTR
ICM_SYS_ADDRHOLE_INT	Triggered when illegal/unauthorized access occurs in HP AXI matrix	
ICM_DLOCK_INT	Triggered when any AXI transfer in the AXI matrix times out	LP_SYSREG_INTR
SLOW_CLK_TICK_INT	ETM tick event interrupt. The interrupt source is periodically triggered at the frequency of LP_SLOW_CLK when the LP CPU is not sleeping. This event source can be disabled via the LP_CLKRST_ETM_EVENT_TICK_EN.	
ETM_TASK_ULP_INT	From ULP_TASK_INT_CPU. Ref to 14 *Event Task Matrix (ETM)* for details	
LP_CORE_DBUS_TIMEOUT_INT	Triggered upon LP CPU DBUS timeout	
LP_CORE_IBUS_TIMEOUT_INT	Triggered upon LP CPU IBUS timeout	
LP_CORE_AHB_TIMEOUT_INT	Triggered upon LP CPU AHB timeout	
IDBUS_ADDRHOLE_INT	Triggered when illegal/unauthorized access occurs on LP CPU IBUS or DBUS	
LP_ADDRHOLE_INT	Triggered when illegal/unauthorized access occurs in LP AHB matrix	

Note:

For definitions of *interrupt*, *interrupt signal*, *interrupt source*, and their correlations, please refer to Chapter 13 *Interrupt Matrix* > Section 13.2 *Interrupt Terminology in ESP32-P4*.

21.4 Register Summary

21.4.1 HP Register Summary

The addresses in this section are relative to HP System Registers base address provided in Table 9.3-2 in Chapter 9 *System and Memory*.

The abbreviations given in Column **Access** are explained in Section Access Types for Registers.

Name	Description	Address	Access
Version Register			
HP_SYSTEM_VER_DATE_REG	Version control	0x0000	R/W
HP_SYSTEM_CLK_EN_REG	Clock gating control	0x0004	R/W
Interrupt Register			
HP_SYSTEM_HP_CPU_INT_FROM_CPU_0_REG	Software interrupt 0	0x0010	R/W
HP_SYSTEM_HP_CPU_INT_FROM_CPU_1_REG	Software interrupt 1	0x0014	R/W
HP_SYSTEM_HP_CPU_INT_FROM_CPU_2_REG	Software interrupt 2	0x0018	R/W
HP_SYSTEM_HP_CPU_INT_FROM_CPU_3_REG	Software interrupt 3	0x001C	R/W
HP_SYSTEM_HP_SPM_INT_RAW_REG	HP SPM interrupt raw	0x00FC	R/WTC /SS
HP_SYSTEM_HP_SPM_INT_ST_REG	Masked HP SPM interrupt status	0x0100	RO
HP_SYSTEM_HP_SPM_INT_ENA_REG	HP SPM interrupt enable	0x0104	R/W
HP_SYSTEM_HP_SPM_INT_CLR_REG	HP SPM interrupt clear	0x0108	WT
HP_SYSTEM_HP_CORE_TIMEOUT_INT_RAW_REG	HP CPU timeout interrupt raw	0x01A8	R/WTC /SS
HP_SYSTEM_HP_CORE_TIMEOUT_INT_ST_REG	HP CPU timeout masked interrupt	0x01AC	RO
HP_SYSTEM_HP_CORE_TIMEOUT_INT_ENA_REG	HP CPU timeout interrupt enable	0x01B0	R/W
HP_SYSTEM_HP_CORE_TIMEOUT_INT_CLR_REG	HP CPU timeout interrupt clear	0x01B4	WT
HP_SYSTEM_HP_CACHE_CLK_CONFIG_REG	Cache clock configuration	0x0020	R/W
HP_SYSTEM_CACHE_RESET_CONFIG_REG	Cache reset configuration	0x0024	R/W
HP_SYSTEM_HP_SPM_RAM_PWR_CTRL0_REG	HP SPM clock force on	0x003C	R/W
HP_SYSTEM_L2_ROM_PWR_CTRL0_REG	L2ROM clock force on	0x0040	R/W
HP_SYSTEM_HP_L2_MEM_RAM_PWR_CTRL0_REG	L2MEM clock force on	0x0060	R/W
HP_SYSTEM_CPU_CORESTALLED_ST_REG	HP CPU stall status	0x0064	RO
HP_SYSTEM_CRYPTO_CTRL_REG	Encryption control	0x0070	R/W
HP_SYSTEM_GPIO_0_HOLD_CTRL0_REG	HP GPIO hold control 0	0x0074	R/W
HP_SYSTEM_GPIO_0_HOLD_CTRL1_REG	HP GPIO hold control 1	0x0078	R/W
HP_SYSTEM_L2_MEM_INT_RAW_REG	HP L2MEM interrupt raw	0x009C	R/WTC /SS
HP_SYSTEM_L2_MEM_INT_ST_REG	Masked HP L2MEM interrupt status	0x00A0	RO
HP_SYSTEM_L2_MEM_INT_ENA_REG	HP L2MEM interrupt enable	0x00A4	R/W
HP_SYSTEM_L2_MEM_INT_CLR_REG	HP L2MEM interrupt clear	0x00A8	WT
HP_SYSTEM_AHB2AXI_BRESP_ERR_INT_RAW_REG	AHB to AXI BRESP error interrupt raw	0x0188	R/WTC /SS
HP_SYSTEM_AHB2AXI_BRESP_ERR_INT_ST_REG	AHB to AXI BRESP error interrupt status	0x018C	RO

Name	Description	Address	Access
HP_SYSTEM_AHB2AXI_BRESP_ERR_INT_ENA_REG	AHB to AXI BRESP error interrupt enable	0x0190	R/W
HP_SYSTEM_AHB2AXI_BRESP_ERR_INT_CLR_REG	AHB to AXI BRESP error interrupt clear	0x0194	WT
HP_SYSTEM_L2_MEM_L2_RAM_ECC_REG	HP L2MEM ECC enable	0x00AC	R/W
HP_SYSTEM_L2_MEM_INT_RECORD0_REG	HP L2MEM interrupt record 0	0x00B0	RO
HP_SYSTEM_L2_MEM_INT_RECORD1_REG	HP L2MEM interrupt record 1	0x00B4	RO
HP_SYSTEM_L2_MEM_L2_CACHE_ECC_REG	L2 cache ECC enable	0x00C4	R/W
HP_SYSTEM_HP_L1CACHE_BUS0_ID_REG	L1 cache bus 0 ID	0x00C8	R/W
HP_SYSTEM_HP_L1CACHE_BUS1_ID_REG	L1 cache bus 1 ID	0x00CC	R/W
HP_SYSTEM_GPIO_DED_HOLD_CTRL_REG	HP GPIO hold control	0x00F0	R/W
HP_SYSTEM_HP_USB20OTG_MEM_CTRL_REG	MEM in USB OTG FS clock force on	0x00F8	R/W
HP_SYSTEM_HP_SPM_PARITY_INT_RECORD_REG	HP SPM parity interrupt status	0x010C	RO
HP_SYSTEM_HP_L1_CACHE_PWR_CTRL_REG	HP L1 cache memory force on	0x0110	R/W
HP_SYSTEM_HP_L2_CACHE_PWR_CTRL_REG	HP L2 cache memory force on	0x0114	R/W
HP_SYSTEM_HP_CPU_WAITI_CONF_REG	CPU_WAITI configuration	0x0118	R/W
HP_SYSTEM_CORE_DEBUG_RUNSTALL_CONF_REG	CPU debug runstall configuration	0x011C	R/W
HP_SYSTEM_HP_CORE_AHB_TIMEOUT_REG	HP CPU AHB bus timeout	0x0120	R/W
HP_SYSTEM_HP_CORE_IBUS_TIMEOUT_REG	HP CPU IBUS timeout	0x0124	R/W
HP_SYSTEM_HP_CORE_DBUS_TIMEOUT_REG	HP CPU DBUS timeout	0x0128	R/W
HP_SYSTEM_HP_ICM_CPU_H2X_CFG_REG	AHB to AXI bridge configuration	0x0138	varies
HP_SYSTEM_PERI1_APB_POSTW_EN_REG	HP PERI1 APB post write enable	0x013C	R/W
HP_SYSTEM_BITSCRAMBLER_PERI_SEL_REG	BitScrambler peripherals select	0x0140	R/W
HP_SYSTEM_APB_SYNC_POSTW_EN_REG	HP PERI0 APB post write enable	0x0144	R/W
HP_SYSTEM_HP_RSA_PD_CTRL_REG	RSA memory PD control	0x01D0	R/W
HP_SYSTEM_HP_ECC_PD_CTRL_REG	ECC memory PD control	0x01D4	R/W
HP_SYSTEM_HP_RNG_CFG_REG	RNG configuration	0x01D8	R/W
HP_SYSTEM_HP_UART_PD_CTRL_REG	Memory PD control	0x01DC	R/W
HP_SYSTEM_HP_PERI_MEM_CLK_FORCE_ON_REG	HP PERI memory clock force on	0x01E0	R/W
HP_SYSTEM_HP_CPU_WAKEUP_EVENT_REG	CPU wakeup event control	0x01EC	R/W
HP_SYSTEM_HP2LP_INTR_GROUP0_EN_REG	HP to LP interrupt enable group 0	0x01F0	R/W
HP_SYSTEM_HP2LP_INTR_GROUP1_EN_REG	HP to LP interrupt enable group 1	0x01F4	R/W
HP_SYSTEM_HP2LP_INTR_GROUP2_EN_REG	HP to LP interrupt enable group 2	0x01F8	R/W
HP_SYSTEM_HP2LP_INTR_GROUP3_EN_REG	HP to LP interrupt enable group 3	0x01FC	R/W
HP_SYSTEM_HP2LP_WAKEUP_GROUP0_EN_REG	HP to LP wakeup enable group 0	0x0210	R/W
HP_SYSTEM_HP2LP_WAKEUP_GROUP1_EN_REG	HP to LP wakeup enable group 1	0x0214	R/W
HP_SYSTEM_HP2LP_WAKEUP_GROUP2_EN_REG	HP to LP wakeup enable group 2	0x0218	R/W
HP_SYSTEM_HP2LP_WAKEUP_GROUP3_EN_REG	HP to LP wakeup enable group 3	0x021C	R/W
HP_SYSTEM_GDMA_CTRL_REG	DW-GDMA control	0x0148	R/W
HP_SYSTEM_GMAC_CTRL0_REG	GMAC control 0	0x014C	varies
HP_SYSTEM_GMAC_CTRL1_REG	GMAC control 1	0x0150	RO
HP_SYSTEM_GMAC_CTRL2_REG	GMAC control 2	0x0154	RO
HP_SYSTEM_USBOTG20_CTRL_REG	USB OTGHS control	0x015C	varies
HP_SYSTEM_HP_SPM_ERR_RESP_CTRL_REG	HP SPM error response enable	0x0160	R/W

Name	Description	Address	Access
HP_SYSTEM_L2_MEM_REFRESH_REG	HP SPM refresh	0x0164	varies
HP_SYSTEM_HP_SPM_INIT_REG	HP SPM initialization	0x0168	varies
HP_SYSTEM_HP_SPM_PARITY_CHECK_CTRL_REG	HP SPM parity check enable	0x016C	R/W
HP_SYSTEM_L2_MEM_ERR_RESP_CTRL_REG	HP L2MEM error response	0x0198	R/W
HP_SYSTEM_L2_MEM_AHB_BUFFER_CTRL_REG	HP L2MEM AHB buffer enable	0x019C	R/W
HP_SYSTEM_HP_CORE_DMACTIVE_LPCORE_REG	HP CPU debug module active	0x01A0	RO
Control Registers			
HP_SYSTEM_HP_CORE_ERR_RESP_DIS_REG	HP CPU error response masked	0x01A4	R/W
HP_SYSTEM_GPIO_O_HYS_CTRL0_REG	HP GPIO hysteresis control 0	0x01C0	R/W
HP_SYSTEM_GPIO_O_HYS_CTRL1_REG	HP GPIO hysteresis control 1	0x01C4	R/W
Status Registers			
HP_SYSTEM_USB_OTGHS_PHY_ST_REG	USB OTGHS PHY status	0x01E8	RO
HP_SYSTEM_HP2LP_INTR_GROUP0_ST_REG	HP to LP interrupt status group 0	0x0200	RO
HP_SYSTEM_HP2LP_INTR_GROUP1_ST_REG	HP to LP interrupt status group 1	0x0204	RO
HP_SYSTEM_HP2LP_INTR_GROUP2_ST_REG	HP to LP interrupt status group 2	0x0208	RO
HP_SYSTEM_HP2LP_INTR_GROUP3_ST_REG	HP to LP interrupt status group 3	0x020C	RO

21.4.2 ICM Register Summary

The addresses in this section are relative to relative to AXI ICM base address provided in Table 9.3-2 in Chapter 9 *System and Memory*.

The abbreviations given in Column **Access** are explained in Section Access Types for Registers.

Name	Description	Address	Access
Version Register			
HP_SYSTEM_ICM_VER_DATE_REG	ICM version control	0x0000	R/W
Clock Gating Register			
HP_SYSTEM_ICM_CLK_EN_REG	Clock gating control	0x0004	R/W
Status Register			
HP_SYSTEM_ICM_DLOCK_STATUS_REG	ICM deadlock status	0x0008	RO
HP_SYSTEM_ICM_SYS_ADDRHOLE_ADDR_REG	System address hole address	0x0038	RO
HP_SYSTEM_ICM_SYS_ADDRHOLE_INFO_REG	System address hole information	0x003C	RO
HP_SYSTEM_ICM_CPU_ADDRHOLE_ADDR_REG	CPU address hole address	0x0040	RO
HP_SYSTEM_ICM_CPU_ADDRHOLE_INFO_REG	CPU address hole information	0x0044	RO
Interrupt Register			
HP_SYSTEM_ICM_INT_RAW_REG	ICM interrupt raw status	0x000C	R/WTC/SS
HP_SYSTEM_ICM_INT_ST_REG	ICM masked interrupt status	0x0010	RO
HP_SYSTEM_ICM_INT_ENA_REG	ICM interrupt enable	0x0014	R/W
HP_SYSTEM_ICM_INT_CLR_REG	ICM interrupt clear	0x0018	WT
Configuration Register			
HP_SYSTEM_ICM_SLV_ARB_PRIORITY_REG	Slave arbitration priority configuration	0x0024	R/W
HP_SYSTEM_ICM_MST_ARQOS_REG0_REG	ARQoS configuration	0x0028	R/W

Name	Description	Address	Access
HP_SYSTEM_ICM_MST_AWQOS_REG0_REG	AWQoS configuration	0x0030	R/W
HP_SYSTEM_ICM_DLOCK_TIMEOUT_REG	ICM deadlock timeout	0x0048	R/W

21.4.3 LP Register Summary

The addresses in this section are relative to LP System Registers base address provided in Table 9.3-2 in Chapter 9 *System and Memory*.

The abbreviations given in Column **Access** are explained in Section Access Types for Registers.

Name	Description	Address	Access
Configuration & Control Registers			
LP_SYSTEM_LP_SYS_VER_DATE_REG	Version control	0x0000	R/W
LP_SYSTEM_SYS_CTRL_REG	System control	0x0008	varies
LP_SYSTEM_LP_CLK_CTRL_REG	Clock control	0x000C	R/W
LP_SYSTEM_LP_RST_CTRL_REG	Reset control	0x0010	R/W
LP_SYSTEM_LP_CORE_BOOT_ADDR_REG	LP CPU boot address	0x0018	R/W
LP_SYSTEM_LP_SPM_PWR_CTRL_REG	LPROM and LP SPM clock control	0x0024	R/W
LP_SYSTEM_USB_CTRL_REG	USB control	0x0100	varies
LP_SYSTEM_PAD_COMP0_REG	Pad compensation 0	0x0148	R/W
LP_SYSTEM_PAD_COMP1_REG	Pad compensation 1	0x014C	R/W
LP_SYSTEM_HP_MEM_AUX_CTRL_REG	High-performance memory auxiliary control	0x0180	R/W
LP_SYSTEM_LP_MEM_AUX_CTRL_REG	Low-power memory auxiliary control	0x0184	R/W
LP_SYSTEM_HP_ROM_AUX_CTRL_REG	High-performance ROM auxiliary control	0x0188	R/W
LP_SYSTEM_LP_ROM_AUX_CTRL_REG	Low-power ROM auxiliary control	0x018C	R/W
LP_SYSTEM_HP_POR_RST_BYPASS_CTRL_REG	Power-on reset control	0x01A0	R/W
LP_SYSTEM_LP_CORE_AHB_TIMEOUT_REG	CPU AHB bus timeout	0x01B0	R/W
LP_SYSTEM_LP_CORE_IBUS_TIMEOUT_REG	CPU IBUS timeout	0x01B4	R/W
LP_SYSTEM_LP_CORE_DBUS_TIMEOUT_REG	CPU DBUS timeout	0x01B8	R/W
LP_SYSTEM_LP_CORE_ERR_RESP_DIS_REG	CPU error response control	0x01BC	R/W
LP_SYSTEM_RNG_CFG_REG	RNG configuration	0x01C0	varies
LP_SYSTEM_PAD_RTC_HOLD_CTRL0_REG	Pad hold control 0	0x01C4	R/W
LP_SYSTEM_PAD_RTC_HOLD_CTRL1_REG	Pad hold control 1	0x01C8	R/W
LP_SYSTEM_DED_PAD_RTC_HOLD_CTRL_REG	Dedicated pad hold control	0x01CC	R/W
LP_SYSTEM_DISCHARGE_REG	PUF memory and LDO flash discharge control	0x0200	R/W
LP_SYSTEM_HP_USB_OTGHS_PHY_CTRL_REG	USB OTGHS PHY control	0x0204	R/W
Status Registers			
LP_SYSTEM_LP_ADDRHOLE_ADDR_REG	AHB bus address hole address	0x0168	RO
LP_SYSTEM_LP_ADDRHOLE_INFO_REG	AHB bus address hole information	0x016C	RO
LP_SYSTEM_LP_CPU_DBG_PC_REG	LP CPU debug PC	0x0190	RO
LP_SYSTEM_LP_CPU_EXC_PC_REG	LP CPU exception PC	0x0194	RO

Name	Description	Address	Access
LP_SYSTEM_IDBUS_ADDRHOLE_ADDR_REG	CPU IBUS and DBUS address hole address	0x0198	RO
LP_SYSTEM_IDBUS_ADDRHOLE_INFO_REG	CPU IBUS and DBUS address hole information	0x019C	RO
LP_SYSTEM_RNG_DATA_REG	RNG data	0x01A4	RO
Interrupt Registers			
LP_SYSTEM_INT_RAW_REG	Raw interrupt status	0x0170	R/SS /WTC
LP_SYSTEM_INT_ST_REG	Masked interrupt status	0x0174	RO
LP_SYSTEM_INT_ENA_REG	Interrupt enable	0x0178	R/W
LP_SYSTEM_INT_CLR_REG	Interrupt clear	0x017C	WT

21.5 Registers

21.5.1 HP Registers

The addresses in this section are relative to HP System Registers base address provided in Table 9.3-2 in Chapter 9 *System and Memory*.

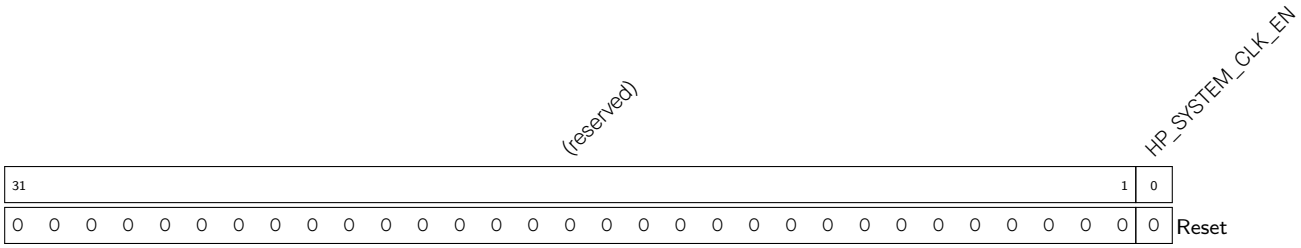
For how to program reserved fields, please refer to Section Programming Reserved Register Field.

Register 21.1. HP_SYSTEM_VER_DATE_REG (0x0000)



HP_SYSTEM_REG_VER_DATE Version control register. (R/W)

Register 21.2. HP_SYSTEM_CLK_EN_REG (0x0004)



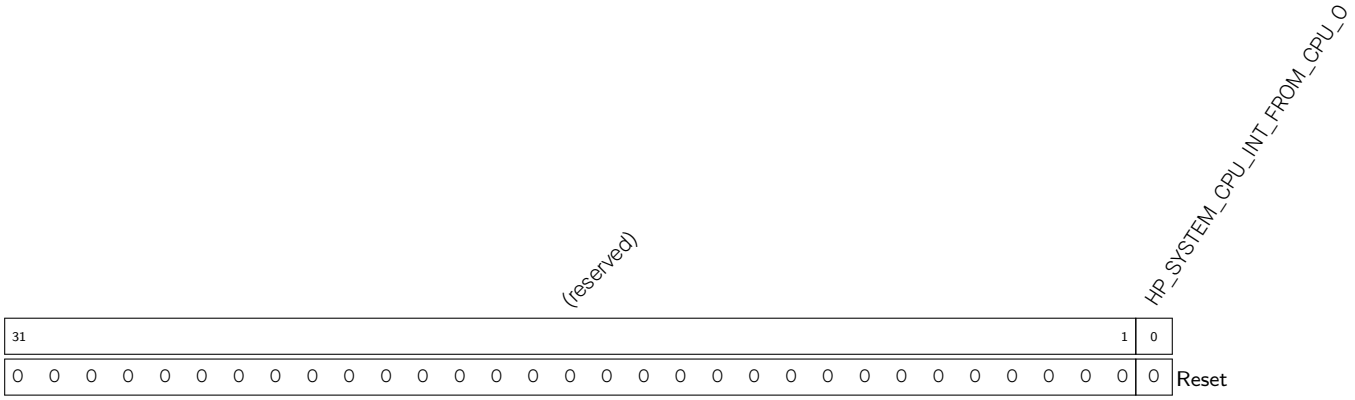
HP_SYSTEM_CLK_EN Configures whether to enable system clock.

0: Disable

1: Enable

(R/W)

Register 21.3. HP_SYSTEM_HP_CPU_INT_FROM_CPU_0_REG (0x0010)



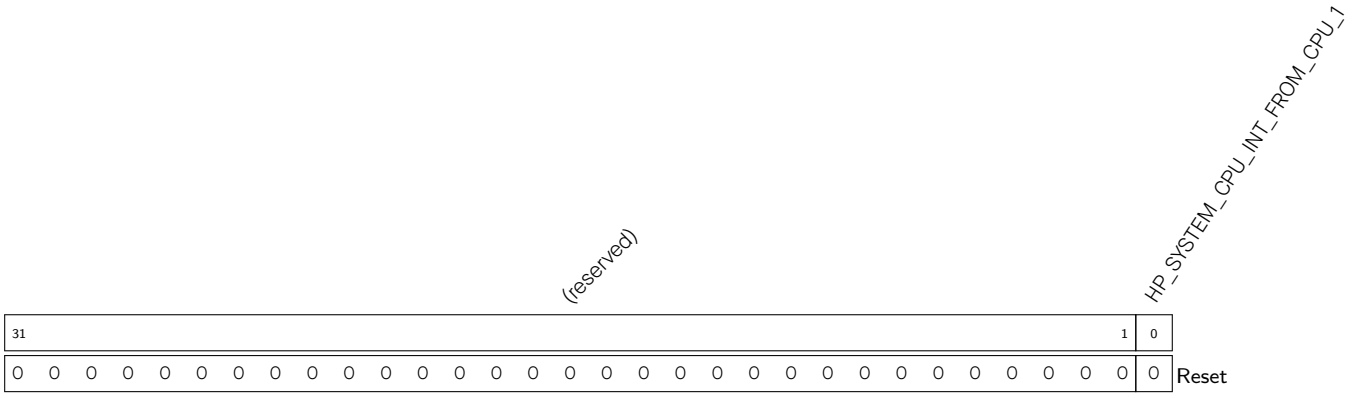
HP_SYSTEM_CPU_INT_FROM_CPU_0 Configures whether to generate CPU_INT_FROM_CPU0.

0: No effect

1: Generate

(R/W)

Register 21.4. HP_SYSTEM_HP_CPU_INT_FROM_CPU_1_REG (0x0014)



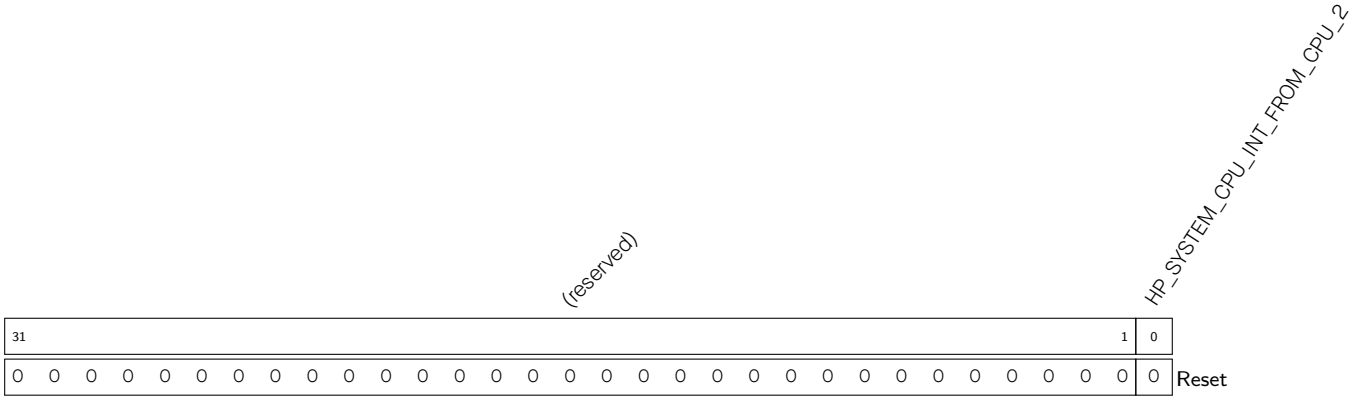
HP_SYSTEM_CPU_INT_FROM_CPU_1 Configures whether to generate CPU_INT_FROM_CPU1.

0: No effect

1: Generate

(R/W)

Register 21.5. HP_SYSTEM_HP_CPU_INT_FROM_CPU_2_REG (0x0018)



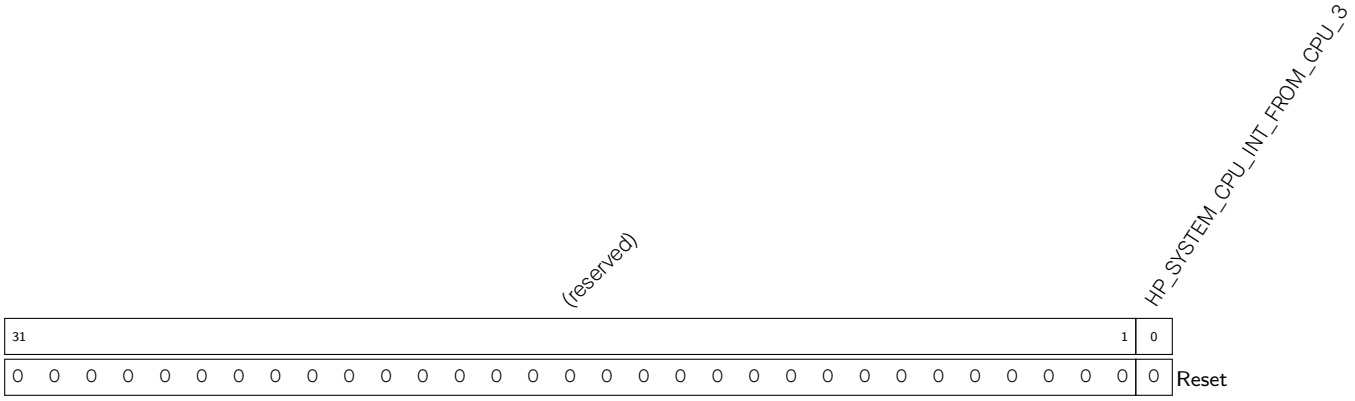
HP_SYSTEM_CPU_INT_FROM_CPU_2 Configures whether to generate CPU_INT_FROM_CPU2.

0: No effect

1: Generate

(R/W)

Register 21.6. HP_SYSTEM_HP_CPU_INT_FROM_CPU_3_REG (0x001C)



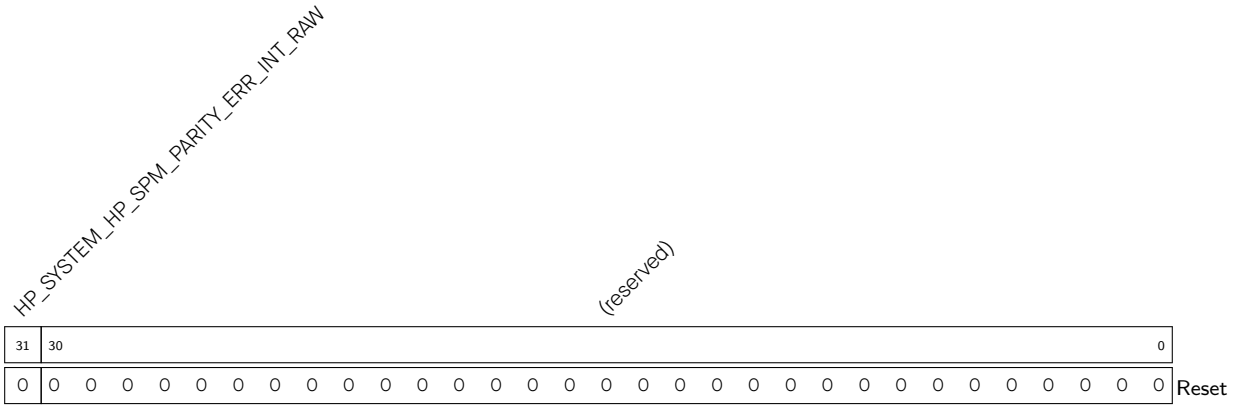
HP_SYSTEM_CPU_INT_FROM_CPU_3 Configures whether to generate CPU_INT_FROM_CPU3.

0: No effect

1: Generate

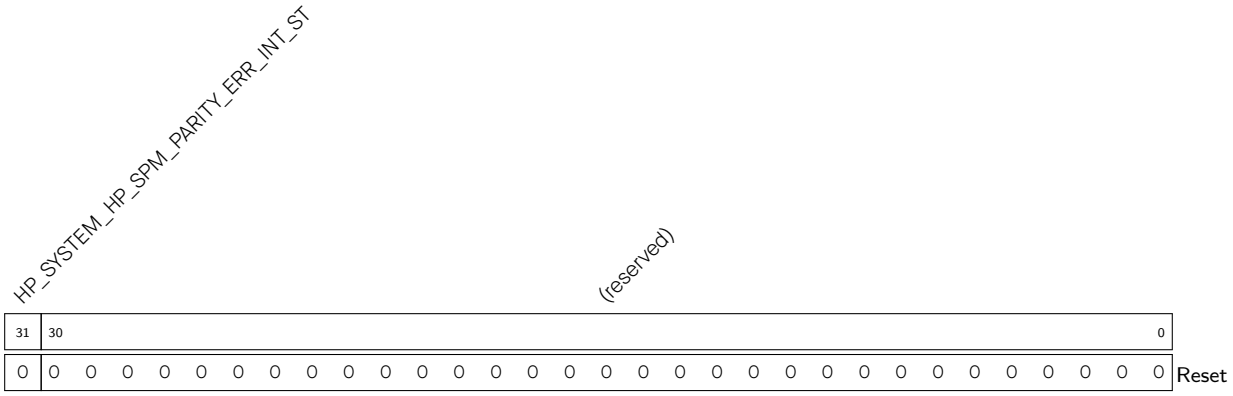
(R/W)

Register 21.7. HP_SYSTEM_HP_SPM_INT_RAW_REG (0x00FC)



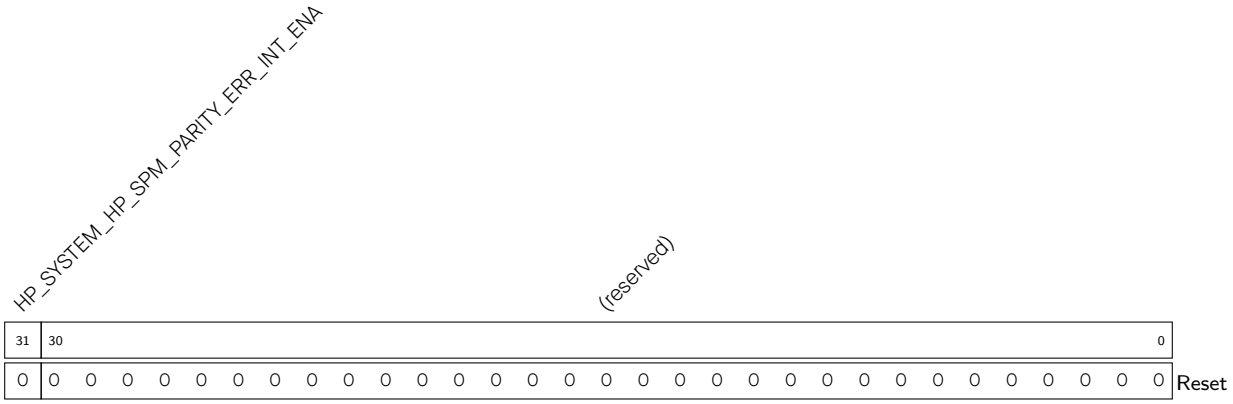
HP_SYSTEM_HP_SPM_PARITY_ERR_INT_RAW The raw interrupt status of HP_SPM_PARITY_ERR_INT, triggered when a parity error occurs. (R/WTC/SS)

Register 21.8. HP_SYSTEM_HP_SPM_INT_ST_REG (0x0100)



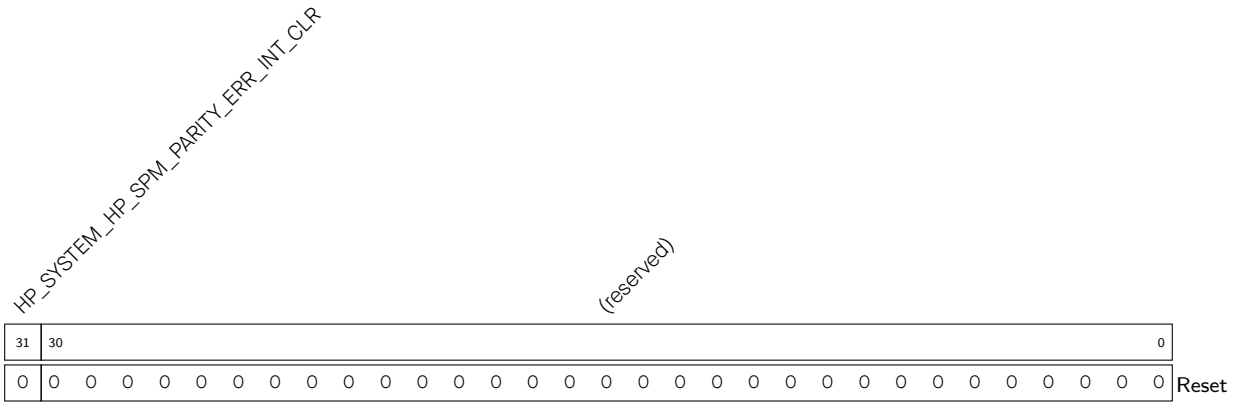
HP_SYSTEM_HP_SPM_PARITY_ERR_INT_ST The masked interrupt status of HP_SPM_PARITY_ERR_INT. (RO)

Register 21.9. HP_SYSTEM_HP_SPM_INT_ENA_REG (0x0104)



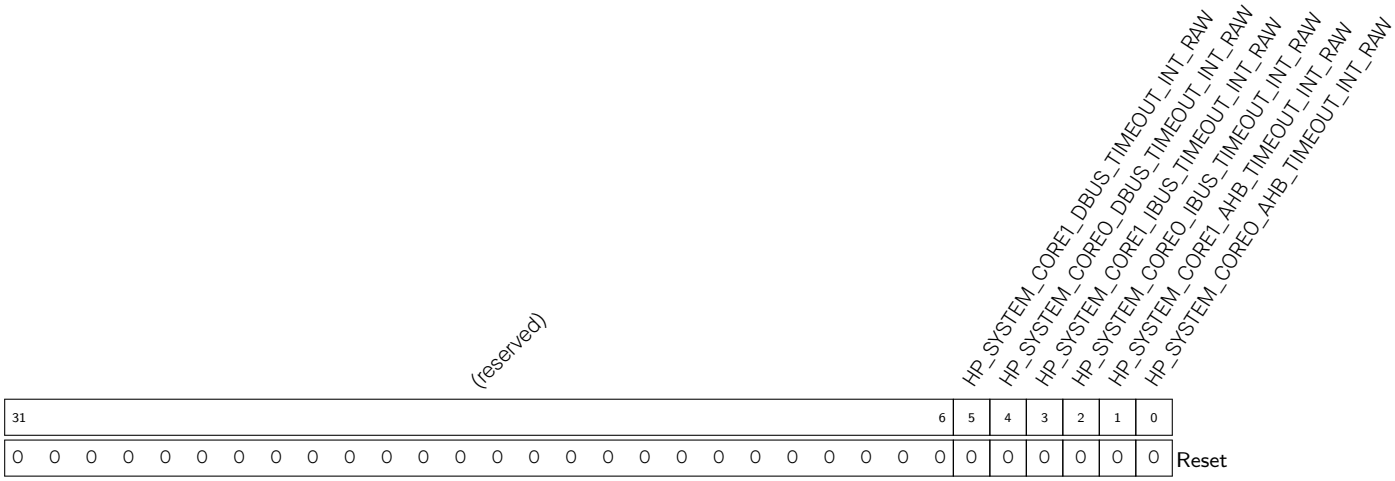
HP_SYSTEM_HP_SPM_PARITY_ERR_INT_ENA Write 1 to enable HP_SPM_PARITY_ERR_INT. (R/W)

Register 21.10. HP_SYSTEM_HP_SPM_INT_CLR_REG (0x0108)



HP_SYSTEM_HP_SPM_PARITY_ERR_INT_CLR Write 1 to clear HP_SPM_PARITY_ERR_INT. (WT)

Register 21.11. HP_SYSTEM_HP_CORE_TIMEOUT_INT_RAW_REG (0x01A8)



HP_SYSTEM_COREO_AHB_TIMEOUT_INT_RAW The raw interrupt status of HP_COREO_AHB_TIMEOUT_INT. (R/WTC/SS)

HP_SYSTEM_CORE1_AHB_TIMEOUT_INT_RAW The raw interrupt status of HP_CORE1_AHB_TIMEOUT_INT. (R/WTC/SS)

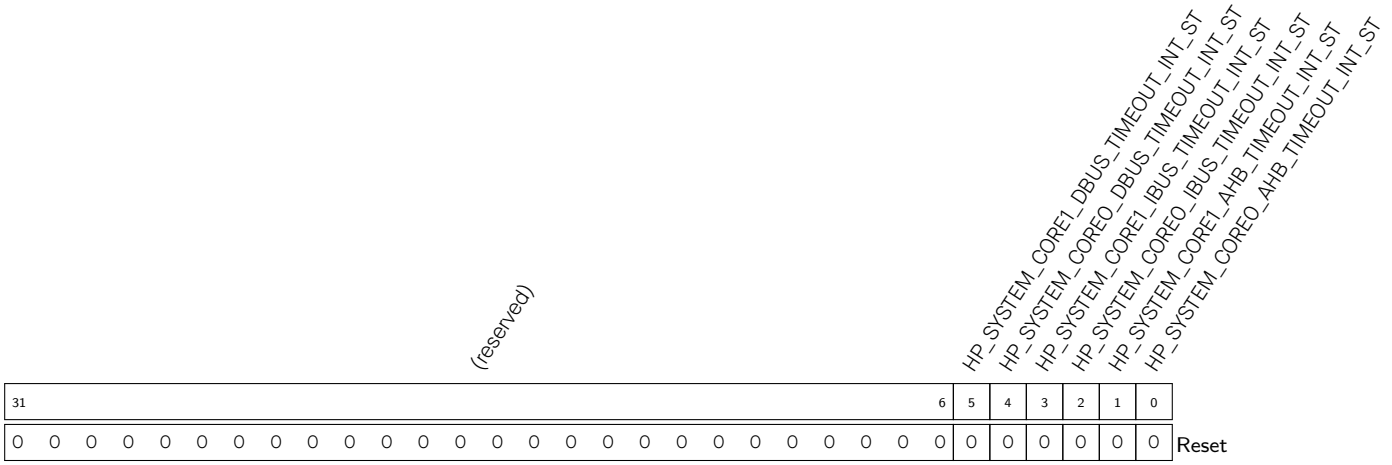
HP_SYSTEM_COREO_IBUS_TIMEOUT_INT_RAW The raw interrupt status of HP_COREO_IBUS_TIMEOUT_INT. (R/WTC/SS)

HP_SYSTEM_CORE1_IBUS_TIMEOUT_INT_RAW The raw interrupt status of HP_CORE1_IBUS_TIMEOUT_INT. (R/WTC/SS)

HP_SYSTEM_COREO_DBUS_TIMEOUT_INT_RAW The raw interrupt status of HP_COREO_DBUS_TIMEOUT_INT. (R/WTC/SS)

HP_SYSTEM_CORE1_DBUS_TIMEOUT_INT_RAW The raw interrupt status of HP_CORE1_DBUS_TIMEOUT_INT. (R/WTC/SS)

Register 21.12. HP_SYSTEM_HP_CORE_TIMEOUT_INT_ST_REG (0x01AC)



HP_SYSTEM_COREO_AHB_TIMEOUT_INT_ST The masked interrupt status of HP_COREO_AHB_TIMEOUT_INT. (RO)

HP_SYSTEM_CORE1_AHB_TIMEOUT_INT_ST The masked interrupt status of HP_CORE1_AHB_TIMEOUT_INT. (RO)

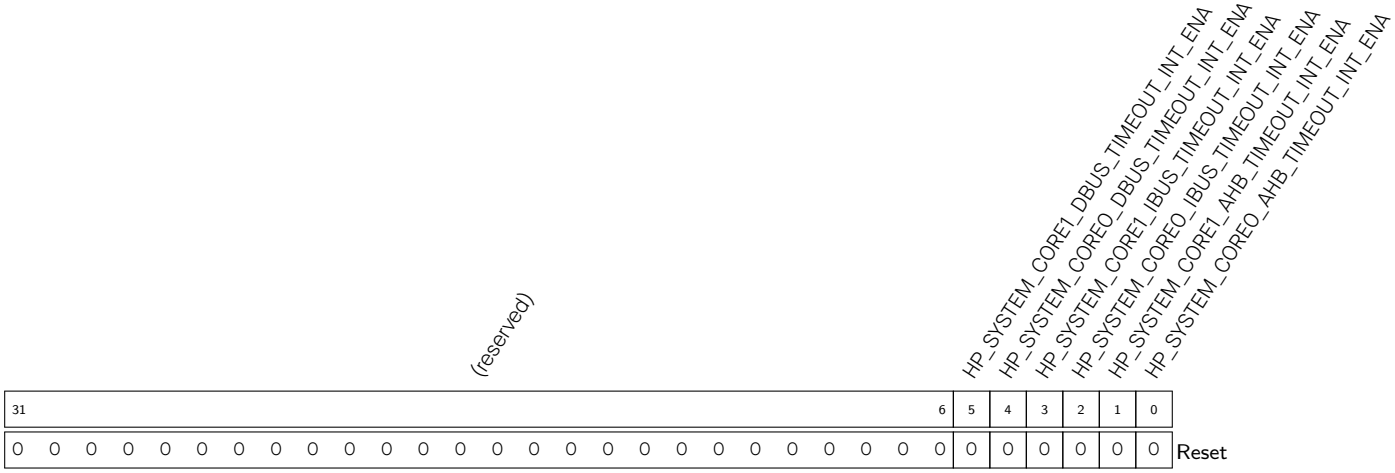
HP_SYSTEM_COREO_IBUS_TIMEOUT_INT_ST The masked interrupt status of HP_COREO_IBUS_TIMEOUT_INT. (RO)

HP_SYSTEM_CORE1_IBUS_TIMEOUT_INT_ST The masked interrupt status of HP_CORE1_IBUS_TIMEOUT_INT. (RO)

HP_SYSTEM_COREO_DBUS_TIMEOUT_INT_ST The masked interrupt status of HP_COREO_DBUS_TIMEOUT_INT. (RO)

HP_SYSTEM_CORE1_DBUS_TIMEOUT_INT_ST The masked interrupt status of HP_CORE1_DBUS_TIMEOUT_INT. (RO)

Register 21.13. HP_SYSTEM_HP_CORE_TIMEOUT_INT_ENA_REG (0x01B0)



HP_SYSTEM_CORE0_AHB_TIMEOUT_INT_ENA Write 1 to enable HP_CORE0_AHB_TIMEOUT_INT. (R/W)

HP_SYSTEM_CORE1_AHB_TIMEOUT_INT_ENA Write 1 to enable HP_CORE1_AHB_TIMEOUT_INT. (R/W)

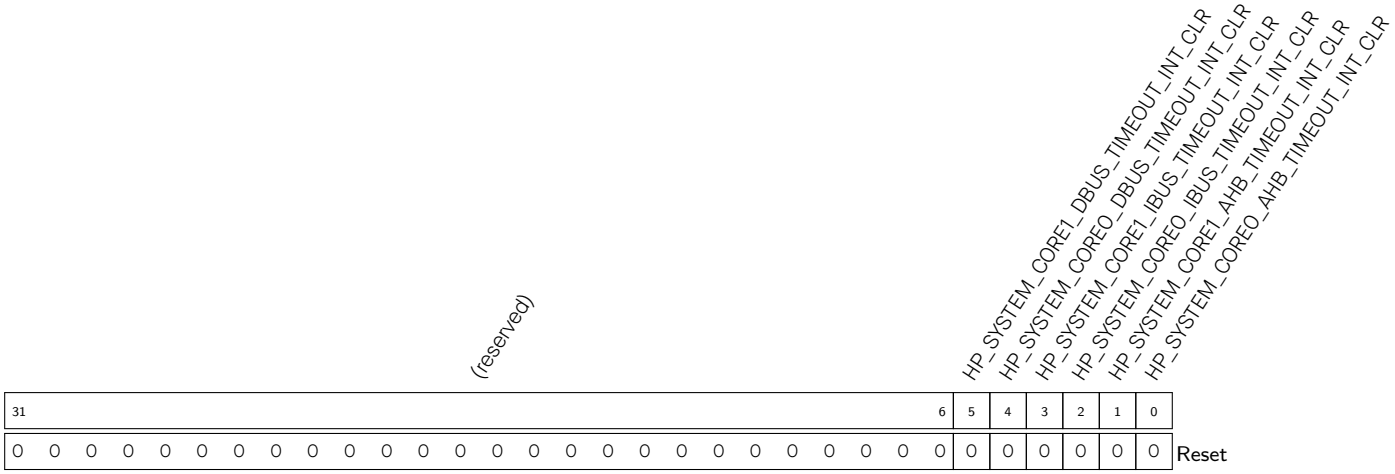
HP_SYSTEM_CORE0_IBUS_TIMEOUT_INT_ENA Write 1 to enable HP_CORE0_IBUS_TIMEOUT_INT. (R/W)

HP_SYSTEM_CORE1_IBUS_TIMEOUT_INT_ENA Write 1 to enable HP_CORE1_IBUS_TIMEOUT_INT. (R/W)

HP_SYSTEM_CORE0_DBUS_TIMEOUT_INT_ENA Write 1 to enable HP_CORE0_DBUS_TIMEOUT_INT. (R/W)

HP_SYSTEM_CORE1_DBUS_TIMEOUT_INT_ENA Write 1 to enable HP_CORE1_DBUS_TIMEOUT_INT. (R/W)

Register 21.14. HP_SYSTEM_HP_CORE_TIMEOUT_INT_CLR_REG (0x01B4)



HP_SYSTEM_CORE0_AHB_TIMEOUT_INT_CLR Write 1 to clear HP_CORE0_AHB_TIMEOUT_INT. (WT)

HP_SYSTEM_CORE1_AHB_TIMEOUT_INT_CLR Write 1 to clear HP_CORE1_AHB_TIMEOUT_INT. (WT)

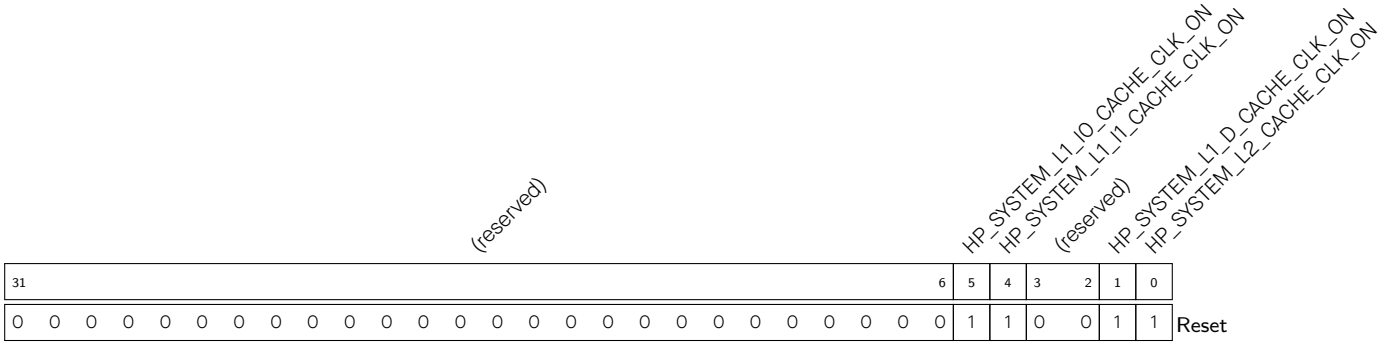
HP_SYSTEM_CORE0_IBUS_TIMEOUT_INT_CLR Write 1 to clear HP_CORE0_IBUS_TIMEOUT_INT. (WT)

HP_SYSTEM_CORE1_IBUS_TIMEOUT_INT_CLR Write 1 to clear HP_CORE1_IBUS_TIMEOUT_INT. (WT)

HP_SYSTEM_CORE0_DBUS_TIMEOUT_INT_CLR Write 1 to clear HP_CORE0_DBUS_TIMEOUT_INT. (WT)

HP_SYSTEM_CORE1_DBUS_TIMEOUT_INT_CLR Write 1 to clear HP_CORE1_DBUS_TIMEOUT_INT. (WT)

Register 21.15. HP_SYSTEM_HP_CACHE_CLK_CONFIG_REG (0x0020)



HP_SYSTEM_L2_CACHE_CLK_ON Configures whether to enable L2 cache clock.

0: Disable

1: Enable

(R/W)

HP_SYSTEM_L1_D_CACHE_CLK_ON Configures whether to enable L1 dcache clock.

0: Disable

1: Enable

(R/W)

HP_SYSTEM_L1_I1_CACHE_CLK_ON Configures whether to enable L1 ichache 1 clock.

0: Disable

1: Enable

(R/W)

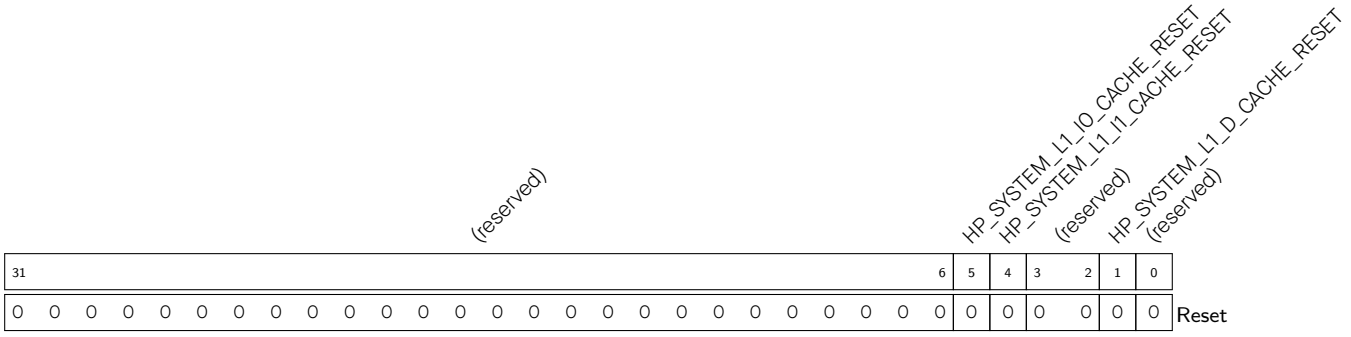
HP_SYSTEM_L1_I0_CACHE_CLK_ON Configures whether to enable L1 ichache 0 clock.

0: Disable

1: Enable

(R/W)

Register 21.16. HP_SYSTEM_CACHE_RESET_CONFIG_REG (0x0024)



HP_SYSTEM_L1_D_CACHE_RESET Configures whether to reset L1 dcache.

0: No effect

1: Reset

(R/W)

HP_SYSTEM_L1_I1_CACHE_RESET Configures whether to reset L1 ichache 1.

0: No effect

1: Reset

(R/W)

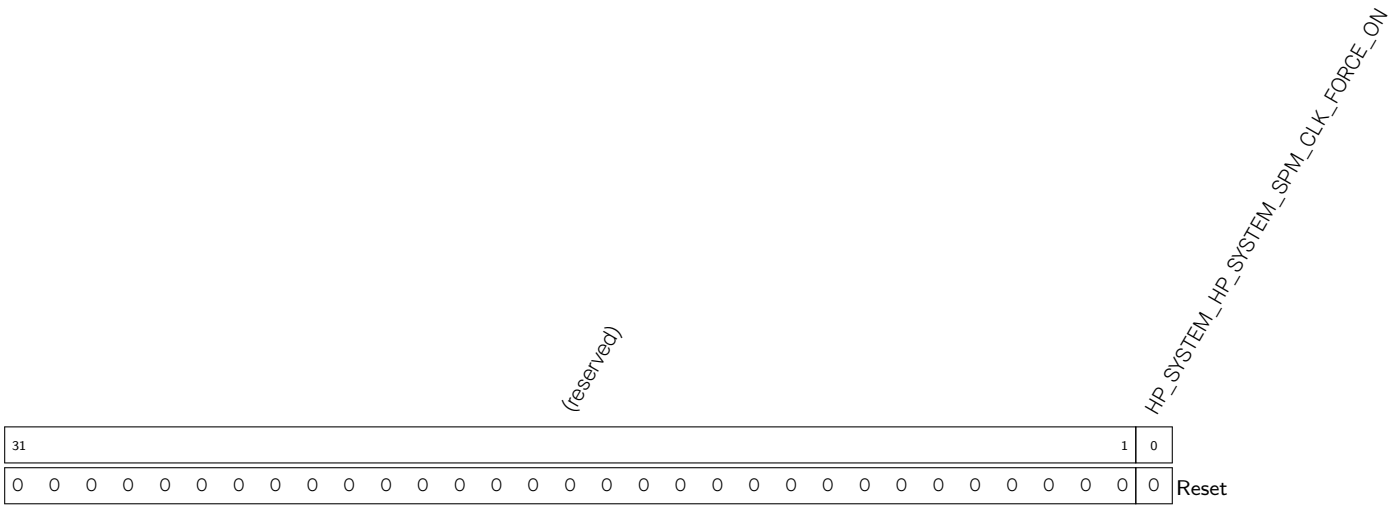
HP_SYSTEM_L1_IO_CACHE_RESET Configures whether to reset L1 ichache 0.

0: No effect

1: Reset

(R/W)

Register 21.17. HP_SYSTEM_HP_SPM_RAM_PWR_CTRL0_REG (0x003C)



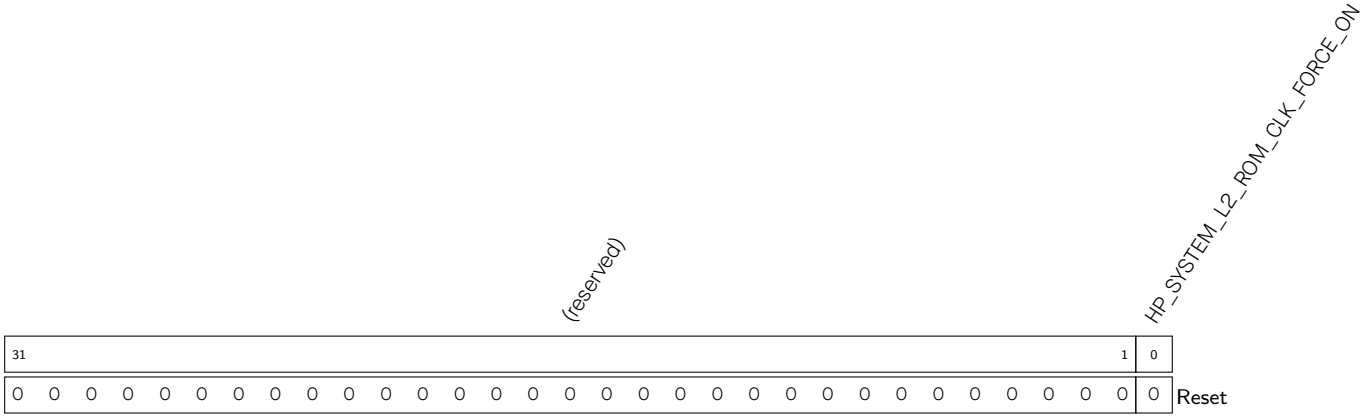
HP_SYSTEM_HP_SPM_CLK_FORCE_ON Configures whether to force on HP SPM clock.

0: No effect

1: Force on

(R/W)

Register 21.18. HP_SYSTEM_L2_ROM_PWR_CTRL0_REG (0x0040)



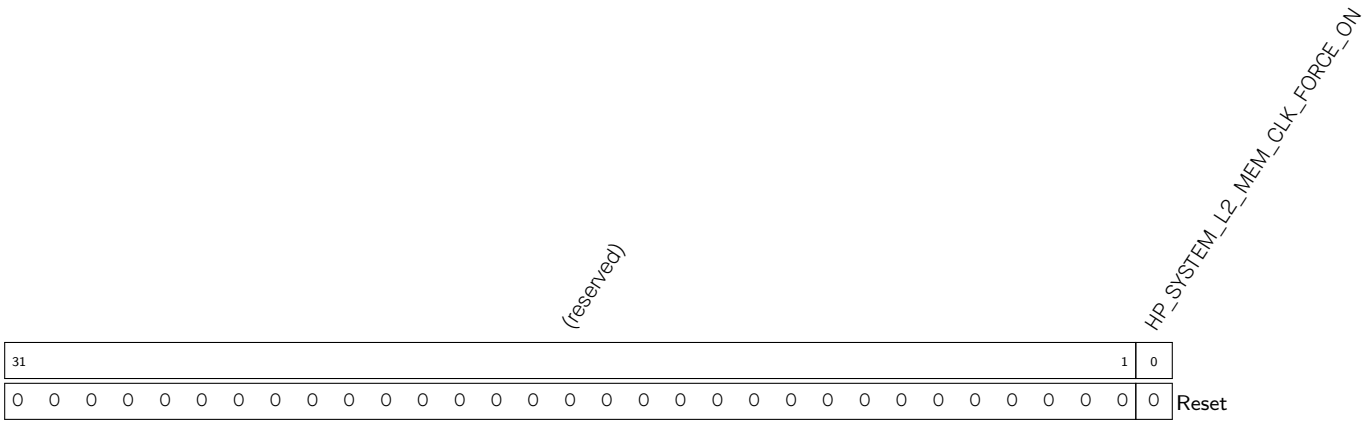
HP_SYSTEM_L2_ROM_CLK_FORCE_ON Configures whether to force on L2ROM clock.

0: No effect

1: Force on

(R/W)

Register 21.19. HP_SYSTEM_HP_SYSTEM_L2_MEM_RAM_PWR_CTRL0_REG (0x0060)



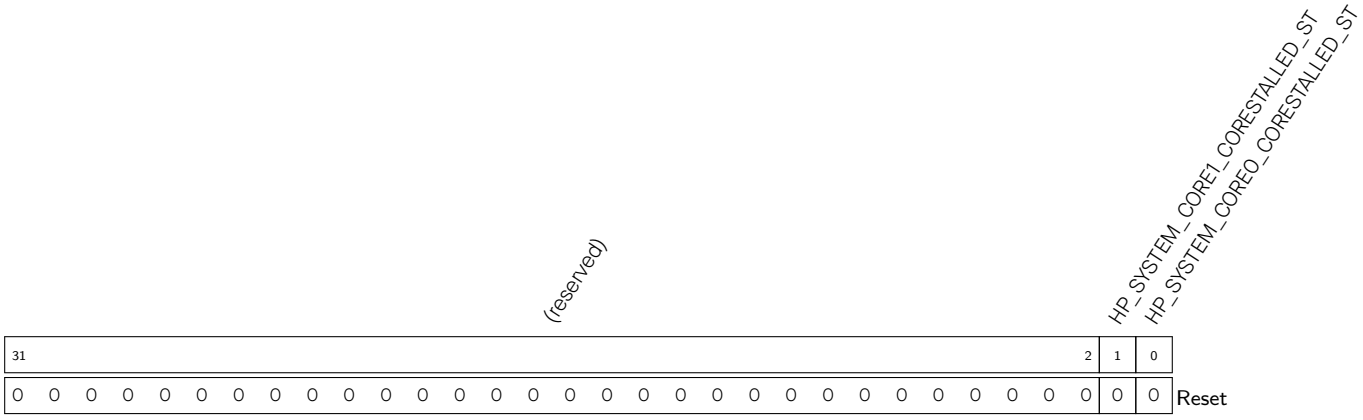
HP_SYSTEM_L2_MEM_CLK_FORCE_ON Configures whether to force on L2MEM clock.

0: No effect

1: Force on

(R/W)

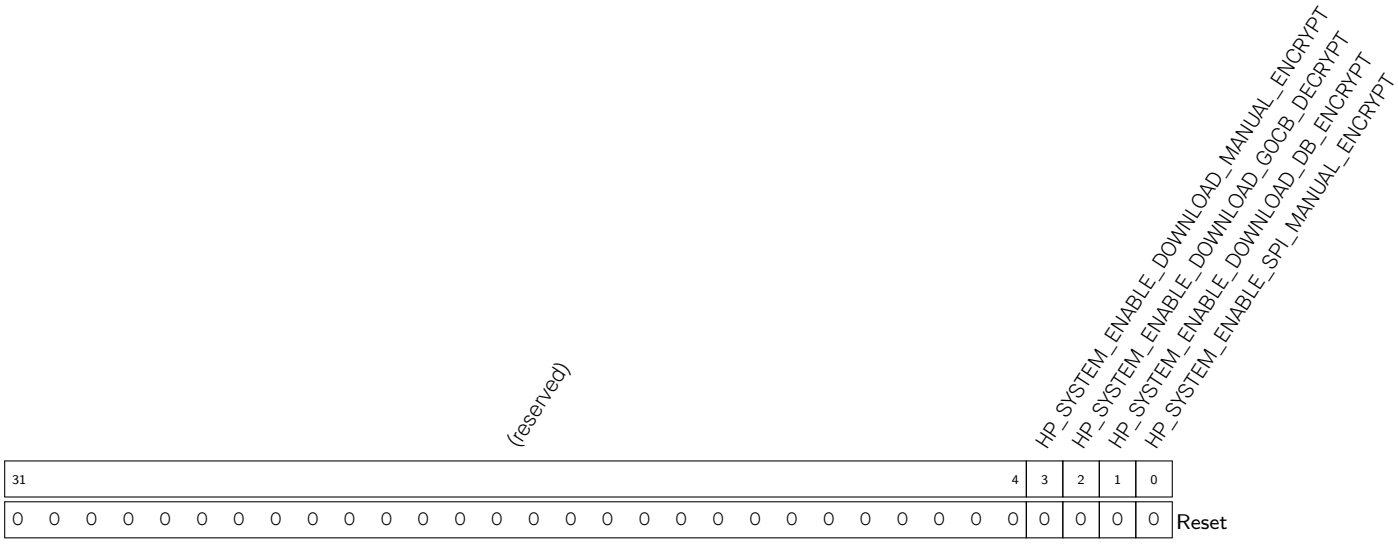
Register 21.20. HP_SYSTEM_CPU_CORESTALLED_ST_REG (0x0064)



HP_SYSTEM_CORE0_CORESTALLED_ST Represents the CPU stall status of HP CPU0. (RO)

HP_SYSTEM_CORE1_CORESTALLED_ST Represents the CPU stall status of HP CPU1. (RO)

Register 21.21. HP_SYSTEM_CRYPTO_CTRL_REG (0x0070)



HP_SYSTEM_ENABLE_SPI_MANUAL_ENCRYPT Configures whether to enable Manual Encryption in SPI Boot mode.

0: Disable

1: Enable

(R/W)

HP_SYSTEM_ENABLE_DOWNLOAD_DB_ENCRYPT Configures whether to enable Auto Encryption in Joint Download Boot mode.

0: Disable

1: Enable

(R/W)

HP_SYSTEM_ENABLE_DOWNLOAD_GOCB_DECRYPT Configures whether to enable Auto Decryption in Joint Download Boot mode.

0: Disable

1: Enable

(R/W)

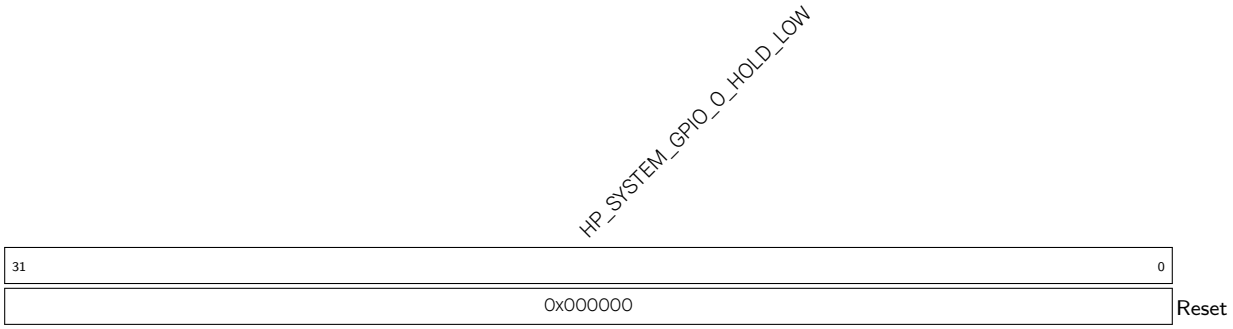
HP_SYSTEM_ENABLE_DOWNLOAD_MANUAL_ENCRYPT Configures whether to enable Manual Encryption in Joint Download Boot mode.

0: Disable

1: Enable

(R/W)

Register 21.22. HP_SYSTEM_GPIO_O_HOLD_CTRL0_REG (0x0074)



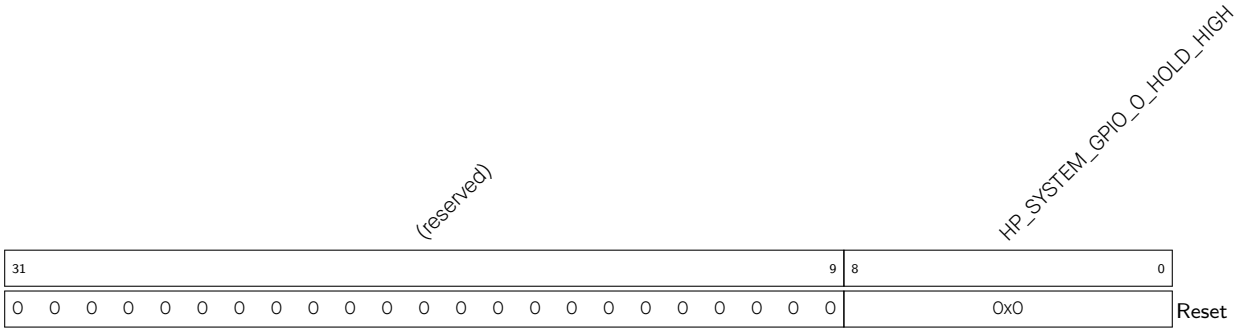
HP_SYSTEM_GPIO_O_HOLD_LOW Configures whether to hold HP GPIO16 ~ HP GPIO47.

0: No effect

1: Hold

(R/W)

Register 21.23. HP_SYSTEM_GPIO_O_HOLD_CTRL1_REG (0x0078)



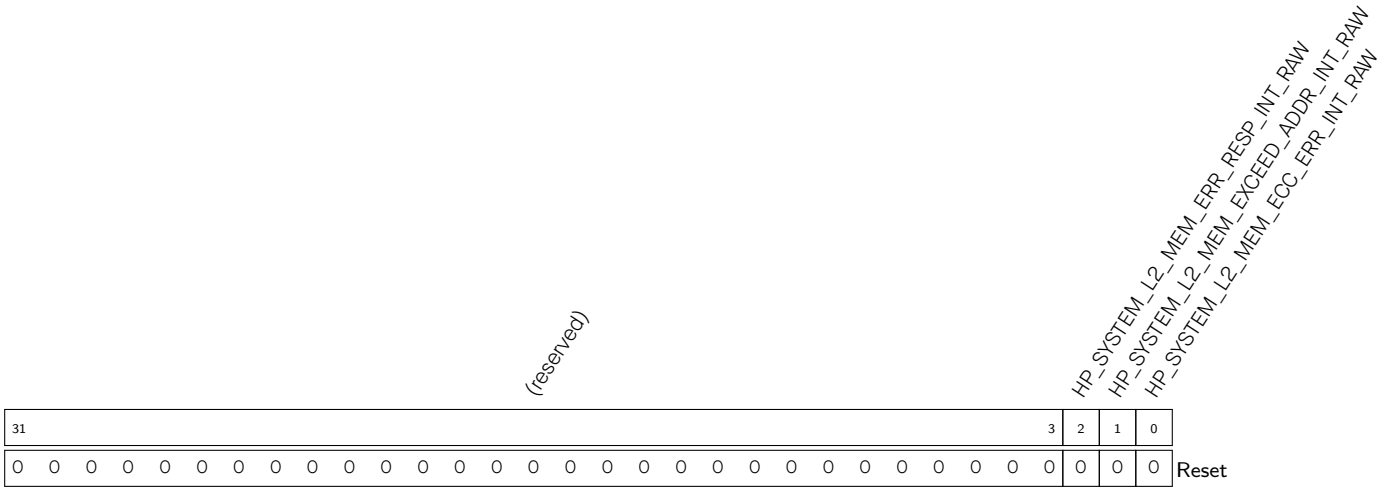
HP_SYSTEM_GPIO_O_HOLD_HIGH Configures whether to hold HP GPIO48 ~ HP GPIO56.

0: No effect

1: Hold

(R/W)

Register 21.24. HP_SYSTEM_L2_MEM_INT_RAW_REG (0x009C)

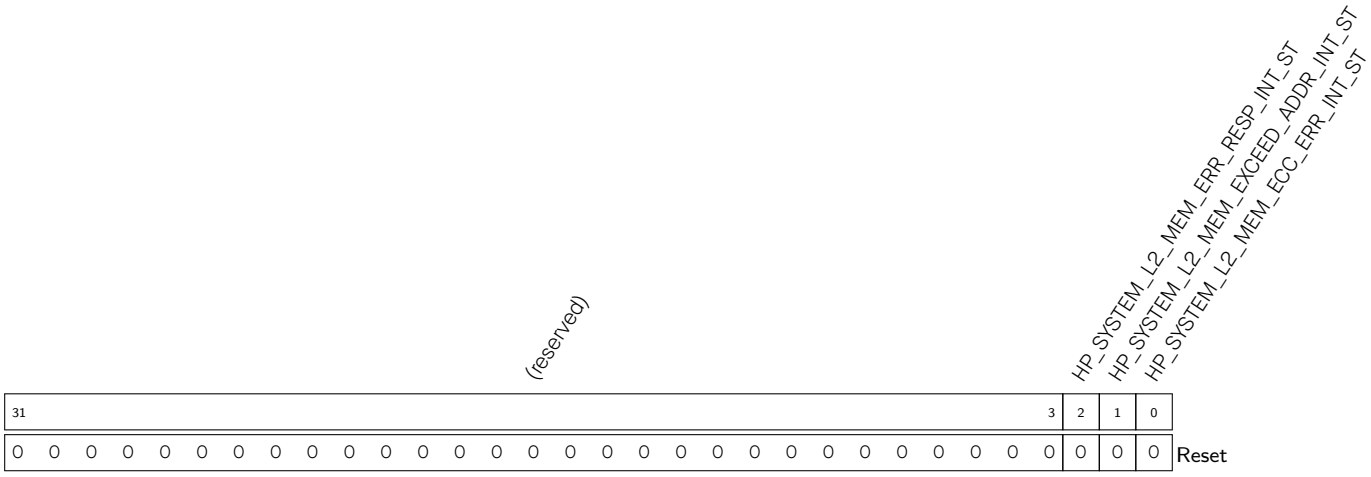


HP_SYSTEM_L2_MEM_ECC_ERR_INT_RAW The raw interrupt status of L2_MEM_ECC_ERR_INT, triggered when a one-bit flip error is detected and corrected by ECC. (R/WTC/SS)

HP_SYSTEM_L2_MEM_EXCEED_ADDR_INT_RAW The raw interrupt status of L2_MEM_EXCEED_ADDR_INT, triggered when access address exceeds limits in different modes.
Bypass mode: 0xff9ffff
L2 cache 128 KB mode: 0xff80000
L2 cache 256 KB mode: 0xff60000
(R/WTC/SS)

HP_SYSTEM_L2_MEM_ERR_RESP_INT_RAW The raw interrupt status of L2_MEM_ERR_RESP_INT, triggered when an error response occurs. (R/WTC/SS)

Register 21.25. HP_SYSTEM_L2_MEM_INT_ST_REG (0x00A0)

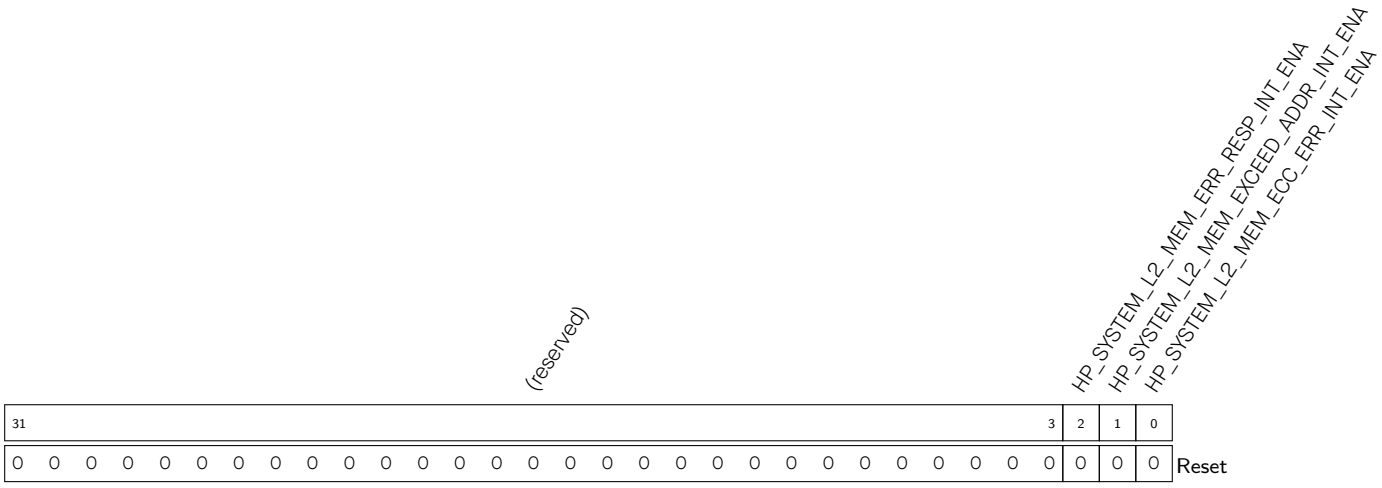


HP_SYSTEM_L2_MEM_ECC_ERR_INT_ST The masked interrupt status of L2_MEM_ECC_ERR_INT. (RO)

HP_SYSTEM_L2_MEM_EXCEED_ADDR_INT_ST The masked interrupt status of L2_MEM_EXCEED_ADDR_INT. (RO)

HP_SYSTEM_L2_MEM_ERR_RESP_INT_ST The masked interrupt status of L2_MEM_ERR_RESP_INT. (RO)

Register 21.26. HP_SYSTEM_L2_MEM_INT_ENA_REG (0x00A4)

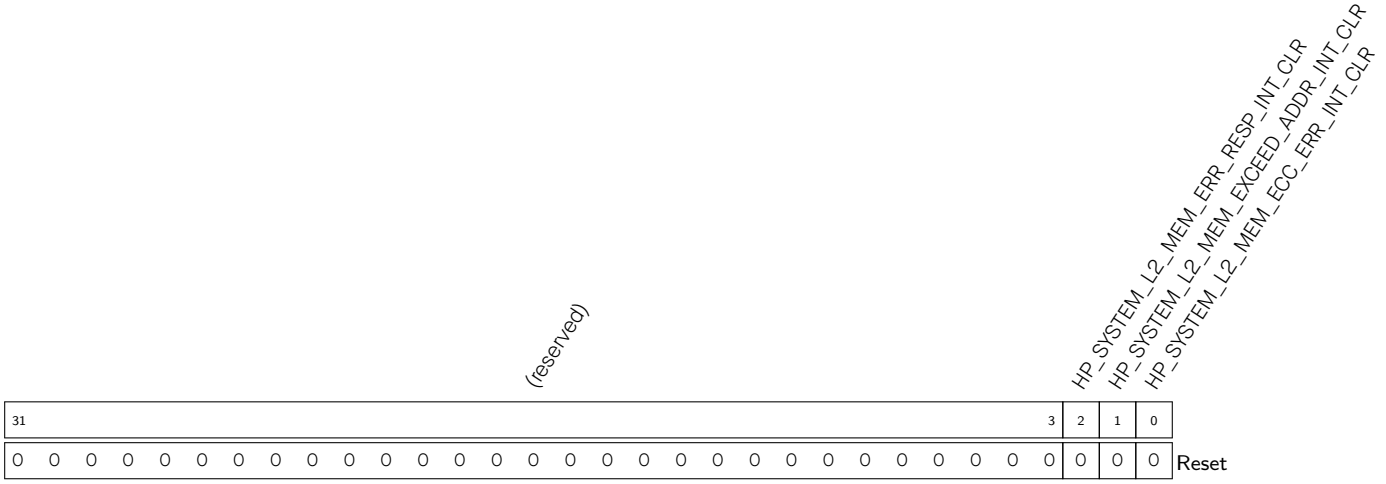


HP_SYSTEM_L2_MEM_ECC_ERR_INT_ENA Write 1 to enable L2_MEM_ECC_ERR_INT. (R/W)

HP_SYSTEM_L2_MEM_EXCEED_ADDR_INT_ENA Write 1 to enable L2_MEM_EXCEED_ADDR_INT. (R/W)

HP_SYSTEM_L2_MEM_ERR_RESP_INT_ENA Write 1 to enable L2_MEM_ERR_RESP_INT. (R/W)

Register 21.27. HP_SYSTEM_L2_MEM_INT_CLR_REG (0x00A8)

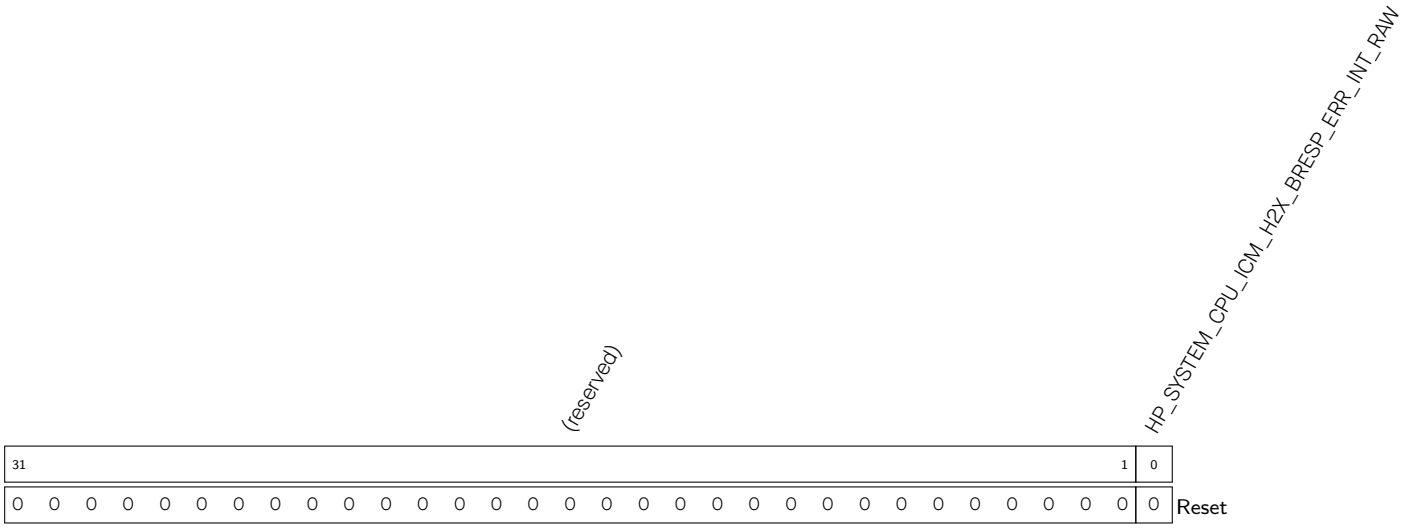


HP_SYSTEM_L2_MEM_ECC_ERR_INT_CLR Write 1 to clear L2_MEM_ECC_ERR_INT. (WT)

HP_SYSTEM_L2_MEM_EXCEED_ADDR_INT_CLR Write 1 to clear L2_MEM_EXCEED_ADDR_INT. (WT)

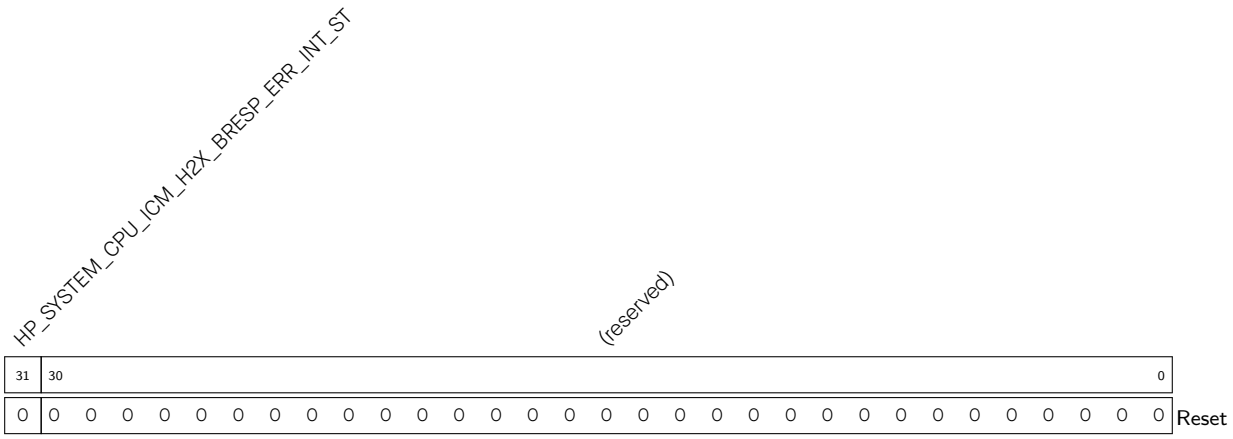
HP_SYSTEM_L2_MEM_ERR_RESP_INT_CLR Write 1 to clear L2_MEM_ERR_RESP_INT. (WT)

Register 21.28. HP_SYSTEM_AHB2AXI_BRESP_ERR_INT_RAW_REG (0x0188)



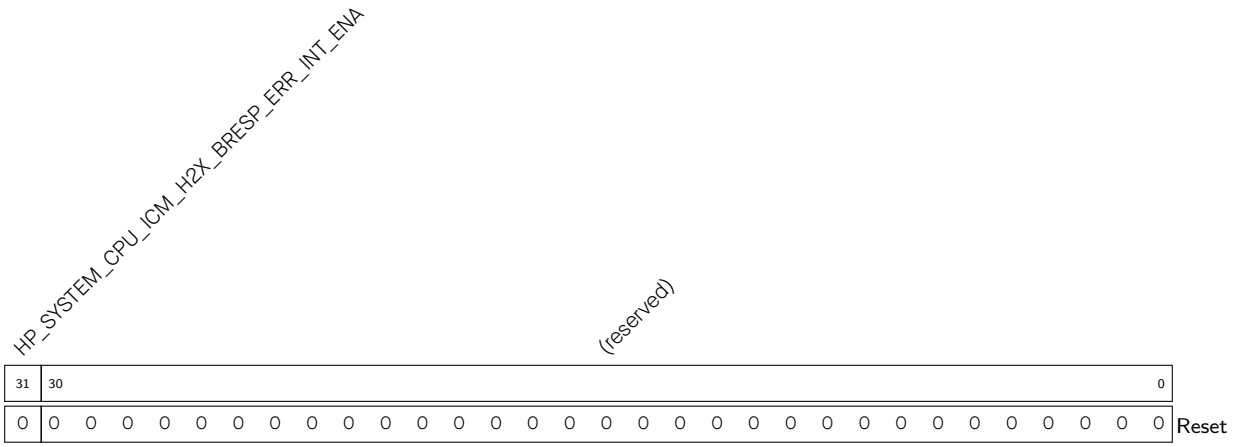
HP_SYSTEM_CPU_ICM_H2X_BRESP_ERR_INT_RAW The raw interrupt status of CPU_ICM_H2X_BRESP_ERR_INT. (R/WTC/SS)

Register 21.29. HP_SYSTEM_AHB2AXI_BRESP_ERR_INT_ST_REG (0x018C)



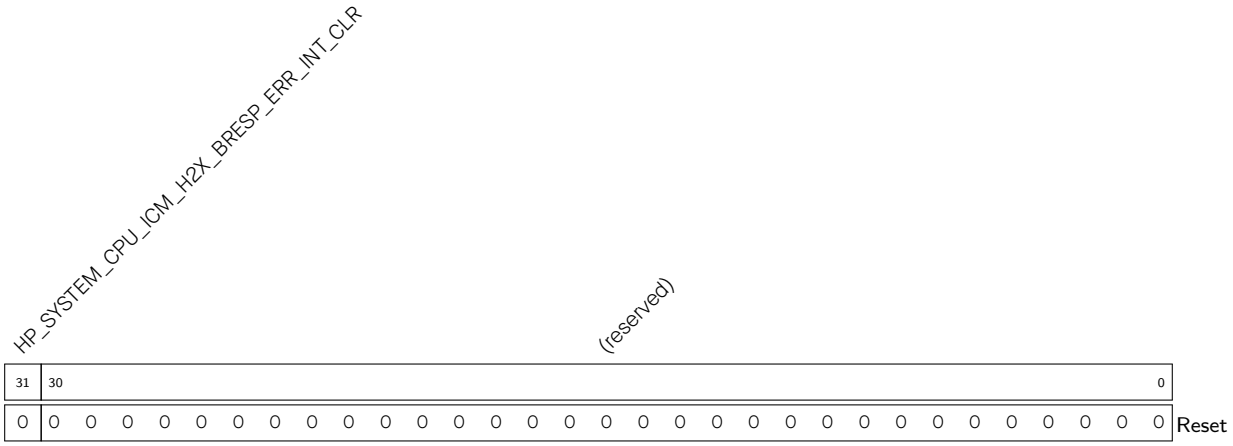
HP_SYSTEM_CPU_ICM_H2X_BRESP_ERR_INT_ST The masked interrupt status of CPU_ICM_H2X_BRESP_ERR_INT. (RO)

Register 21.30. HP_SYSTEM_AHB2AXI_BRESP_ERR_INT_ENA_REG (0x0190)



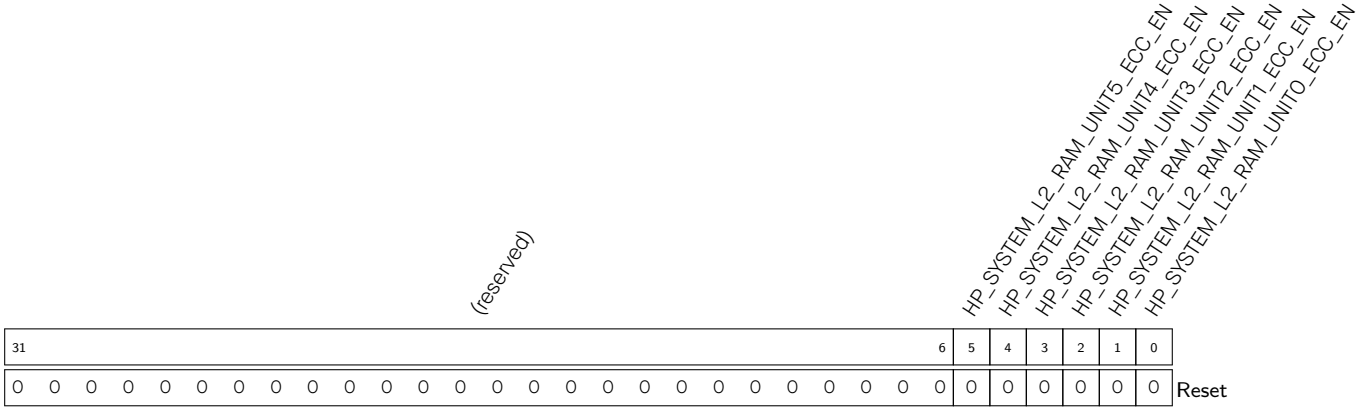
HP_SYSTEM_CPU_ICM_H2X_BRESP_ERR_INT_ENA Write 1 to enable CPU_ICM_H2X_BRESP_ERR_INT. (R/W)

Register 21.31. HP_SYSTEM_AHB2AXI_BRESP_ERR_INT_CLR_REG (0x0194)



HP_SYSTEM_CPU_ICM_H2X_BRESP_ERR_INT_CLR Write 1 to clear CPU_ICM_H2X_BRESP_ERR_INT. (WT)

Register 21.32. HP_SYSTEM_L2_MEM_L2_RAM_ECC_REG (0x00AC)



HP_SYSTEM_L2_RAM_UNIT0_ECC_EN Configures whether to enable the ECC check for L2MEM unit0.

0: Disable

1: Enable

(R/W)

HP_SYSTEM_L2_RAM_UNIT1_ECC_EN Configures whether to enable the ECC check for L2MEM unit1.

0: Disable

1: Enable

(R/W)

HP_SYSTEM_L2_RAM_UNIT2_ECC_EN Configures whether to enable the ECC check for L2MEM unit2.

0: Disable

1: Enable

(R/W)

HP_SYSTEM_L2_RAM_UNIT3_ECC_EN Configures whether to enable the ECC check for L2MEM unit3.

0: Disable

1: Enable

(R/W)

HP_SYSTEM_L2_RAM_UNIT4_ECC_EN Configures whether to enable the ECC check for L2MEM unit4.

0: Disable

1: Enable

(R/W)

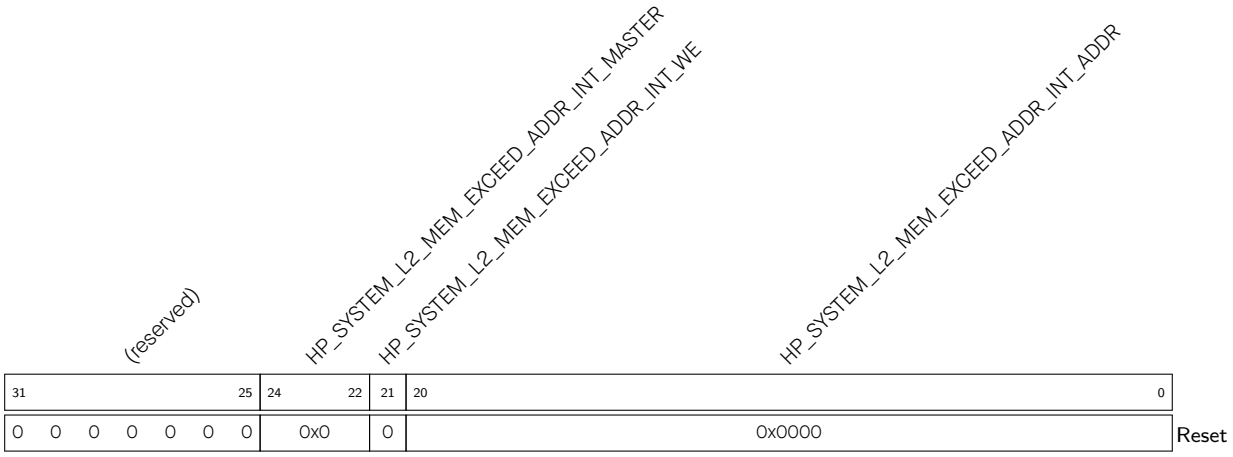
HP_SYSTEM_L2_RAM_UNIT5_ECC_EN Configures whether to enable the ECC check for L2MEM unit5.

0: Disable

1: Enable

(R/W)

Register 21.33. HP_SYSTEM_L2_MEM_INT_RECORD0_REG (0x00B0)



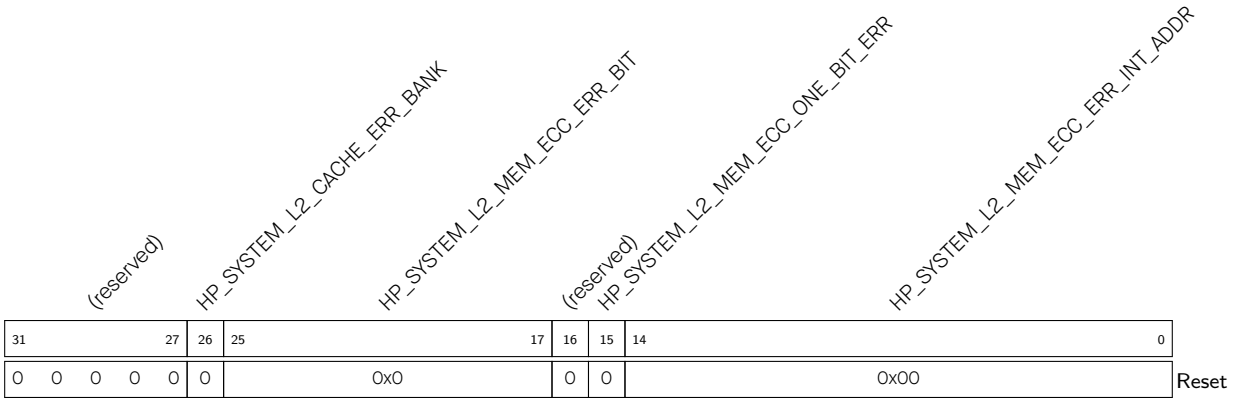
HP_SYSTEM_L2_MEM_EXCEED_ADDR_INT_ADDR Records the address when L2_MEM_EXCEED_ADDR_INT occurs. (RO)

HP_SYSTEM_L2_MEM_EXCEED_ADDR_INT_WE Records the operation (write/read) when L2_MEM_EXCEED_ADDR_INT occurs. (RO)

HP_SYSTEM_L2_MEM_EXCEED_ADDR_INT_MASTER Records source that triggered L2_MEM_EXCEED_ADDR_INT.

0: Invalid. No effect
1: cachebus0
2: cachebus1
3: dma_axi
4: Invalid. No effect
5: Invalid. No effect
6: ahb
7: Invalid. No effect
(RO)

Register 21.34. HP_SYSTEM_L2_MEM_INT_RECORD1_REG (0x00B4)



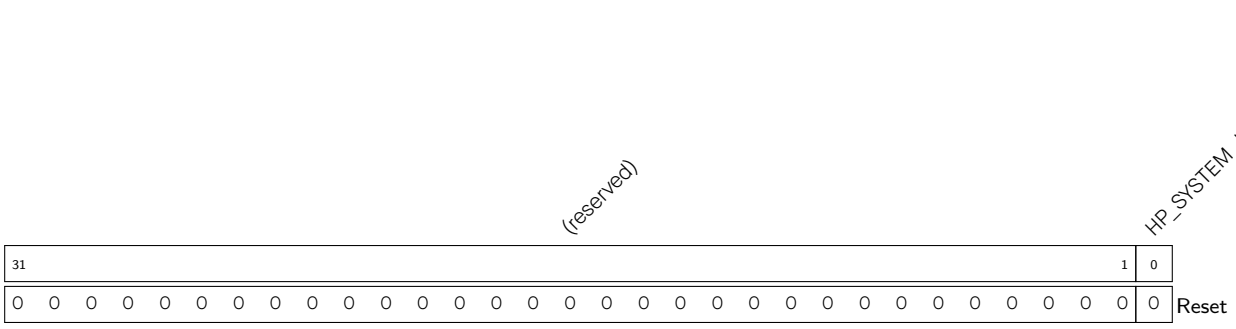
HP_SYSTEM_L2_MEM_ECC_ERR_INT_ADDR Records the address when L2_MEM_ECC_ERR_INT occurs. (RO)

HP_SYSTEM_L2_MEM_ECC_ONE_BIT_ERR Records the error when L2_MEM_ECC_ERR_INT occurs. (RO)

HP_SYSTEM_L2_MEM_ECC_ERR_BIT Records the flipped bit that triggered L2_MEM_ECC_ERR_INT. (RO)

HP_SYSTEM_L2_CACHE_ERR_BANK Records the bank number of L2 cache ECC error. (RO)

Register 21.35. HP_SYSTEM_L2_MEM_L2_CACHE_ECC_REG (0x00C4)



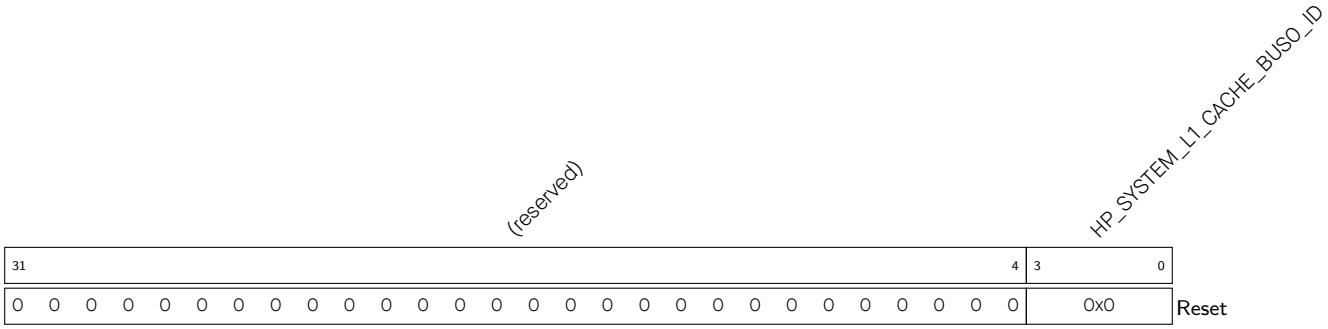
HP_SYSTEM_L2_CACHE_ECC_EN Configures whether to enable the ECC check for L2 cache.

0: Disable

1: Enable

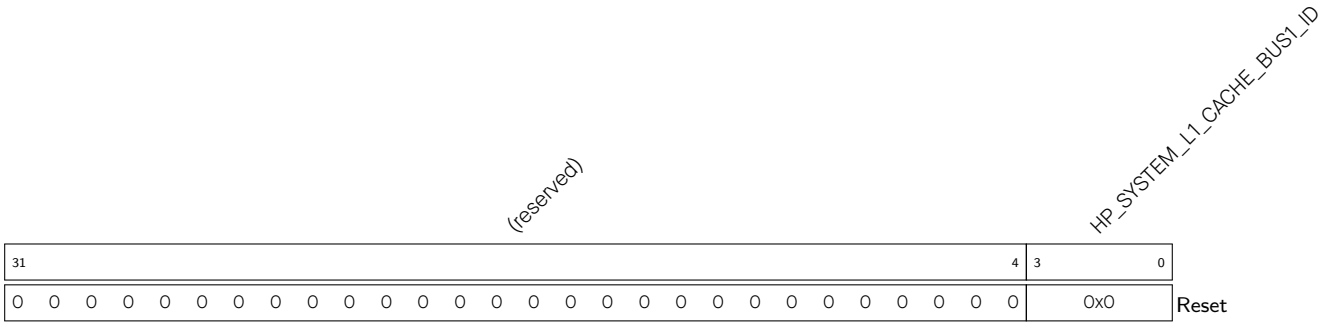
(R/W)

Register 21.36. HP_SYSTEM_HP_L1CACHE_BUS0_ID_REG (0x00C8)



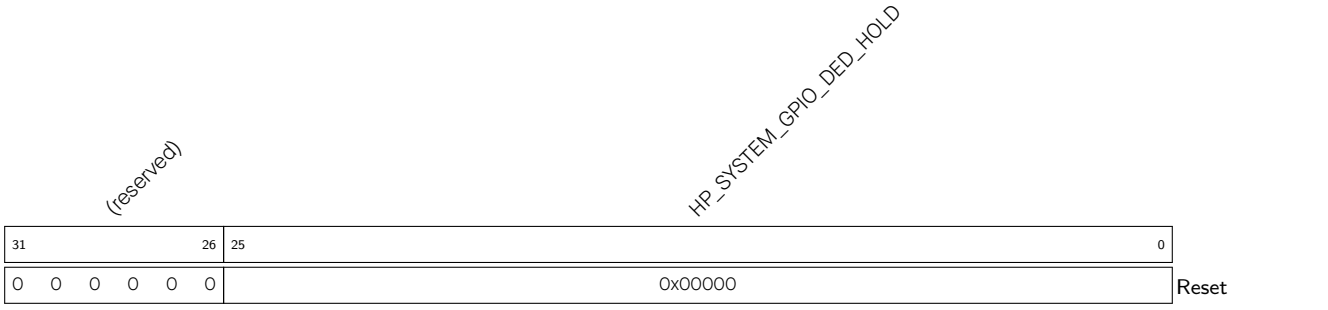
HP_SYSTEM_L1_CACHE_BUS0_ID Configures the L1 cache bus0 UID. (R/W)

Register 21.37. HP_SYSTEM_HP_L1CACHE_BUS1_ID_REG (0x00CC)



HP_SYSTEM_L1_CACHE_BUS1_ID Configures the L1 cache bus1 UID. (R/W)

Register 21.38. HP_SYSTEM_GPIO_DED_HOLD_CTRL_REG (0x00F0)



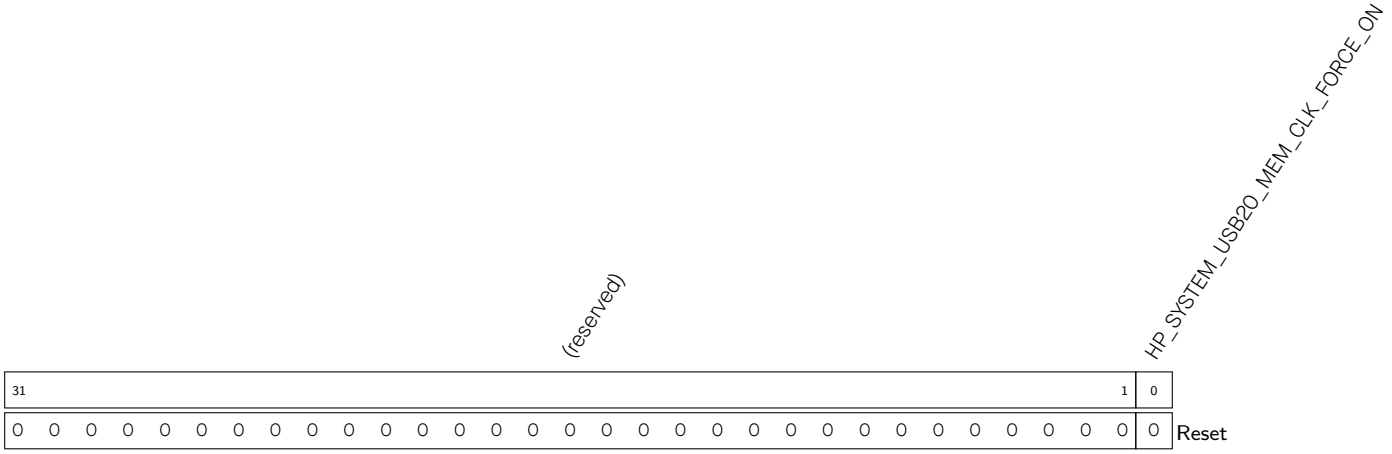
HP_SYSTEM_GPIO_DED_HOLD Configures whether to hold the pin status.

Bit[25:6]: 16 line PSRAM pins

Bit[5:0]: flash pins

(R/W)

Register 21.39. HP_SYSTEM_HP_USB20OTG_MEM_CTRL_REG (0x00F8)



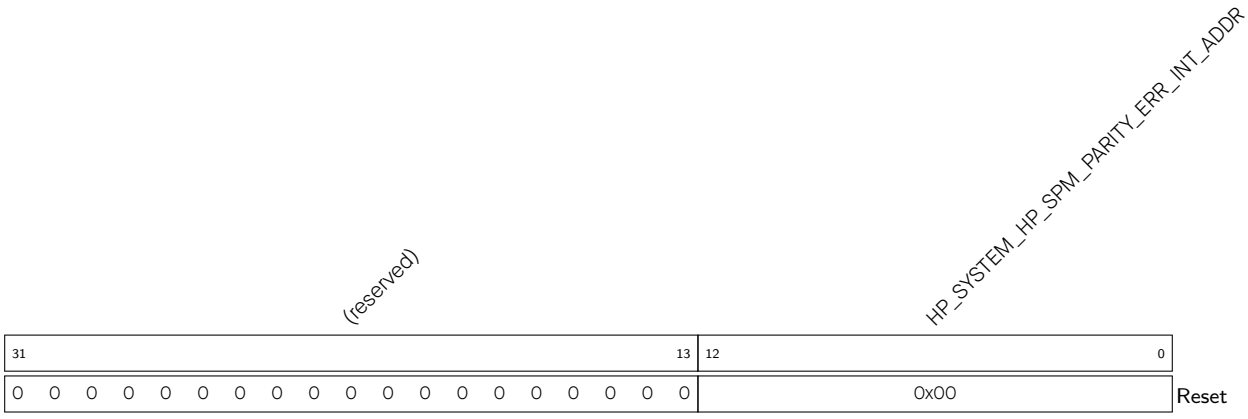
HP_SYSTEM_USB20_MEM_CLK_FORCE_ON Configures whether to force on the memory clock in USB OTGHS.

0: Disable

1: Enable

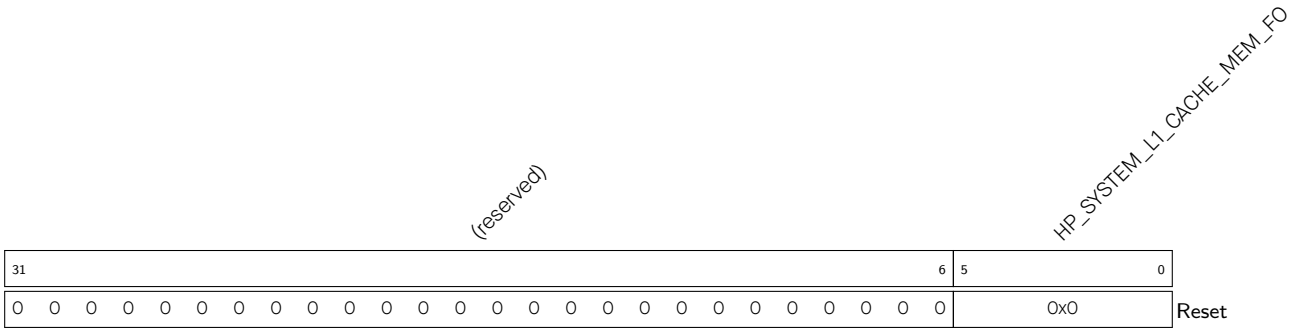
(R/W)

Register 21.40. HP_SYSTEM_HP_SPM_PARITY_INT_RECORD_REG (0x010C)



HP_SYSTEM_HP_SPM_PARITY_ERR_INT_ADDR Records the address when HP_SPM_PARITY_ERR_INT occurs. (RO)

Register 21.41. HP_SYSTEM_HP_L1_CACHE_PWR_CTRL_REG (0x0110)



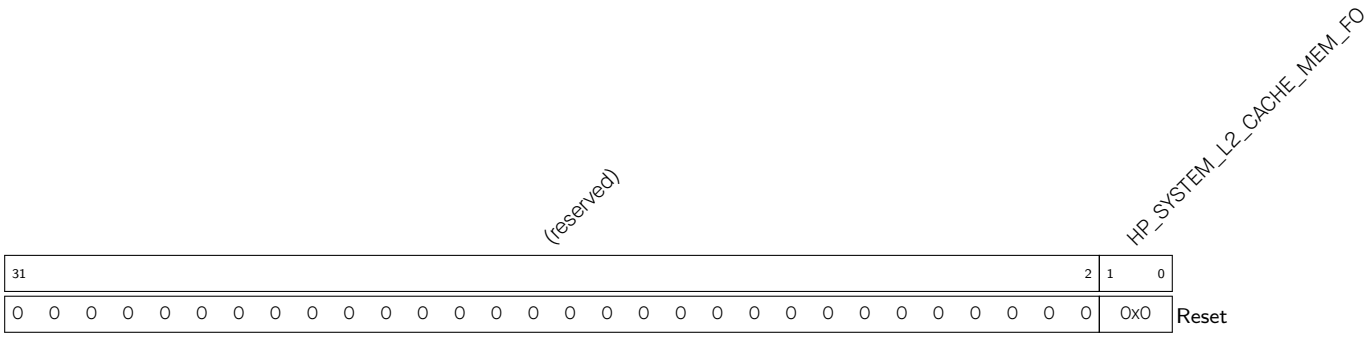
HP_SYSTEM_L1_CACHE_MEM_FO Configures whether to force L1 cache on.

0: Disable

1: Enable

(R/W)

Register 21.42. HP_SYSTEM_HP_L2_CACHE_PWR_CTRL_REG (0x0114)



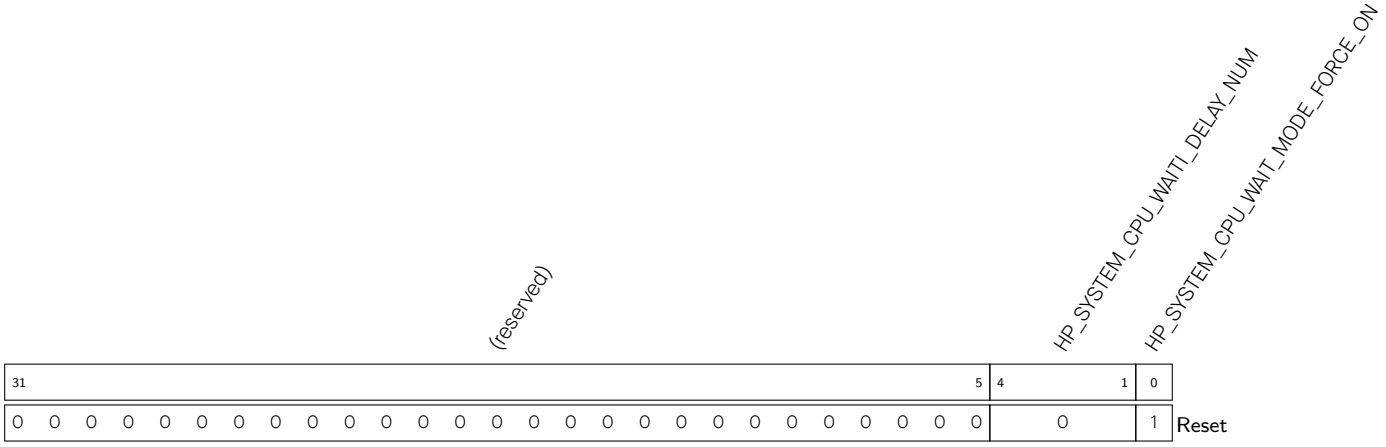
HP_SYSTEM_L2_CACHE_MEM_FO Configures whether to force L2 cache on.

0: Disable

1: Enable

(R/W)

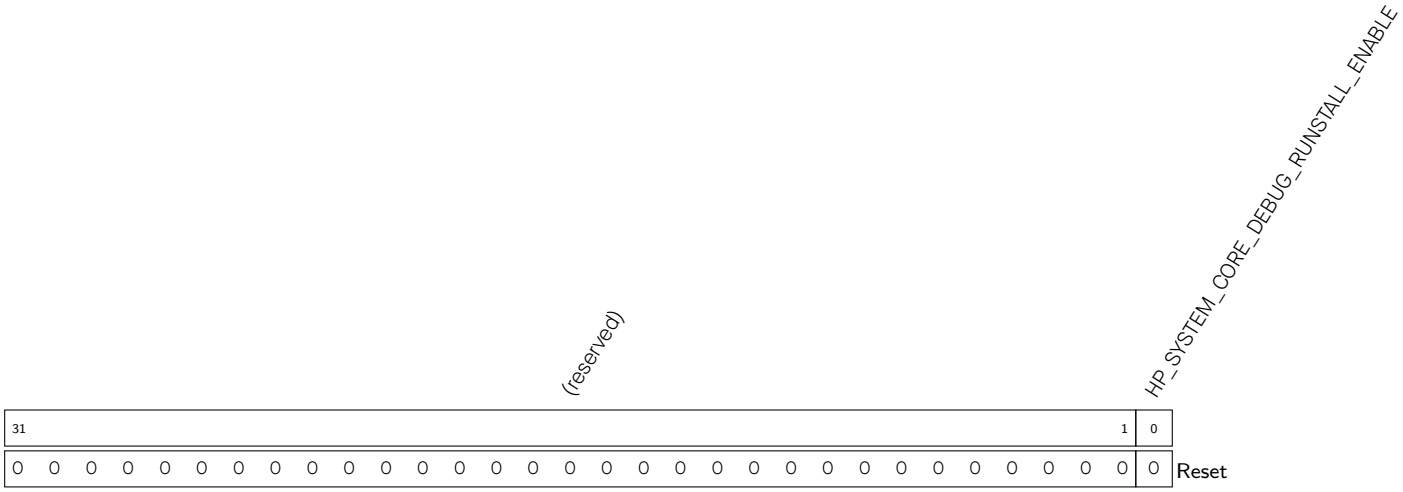
Register 21.43. HP_SYSTEM_HP_CPU_WAITI_CONF_REG (0x0118)



HP_SYSTEM_CPU_WAIT_MODE_FORCE_ON Configures whether to force on cpu_waiti_clk. (R/W)

HP_SYSTEM_CPU_WAITI_DELAY_NUM Configures the delay cycles when CPU enters wait mode. After the configured delay, cpu_waiti_clk turns off. (R/W)

Register 21.44. HP_SYSTEM_CORE_DEBUG_RUNSTALL_CONF_REG (0x011C)



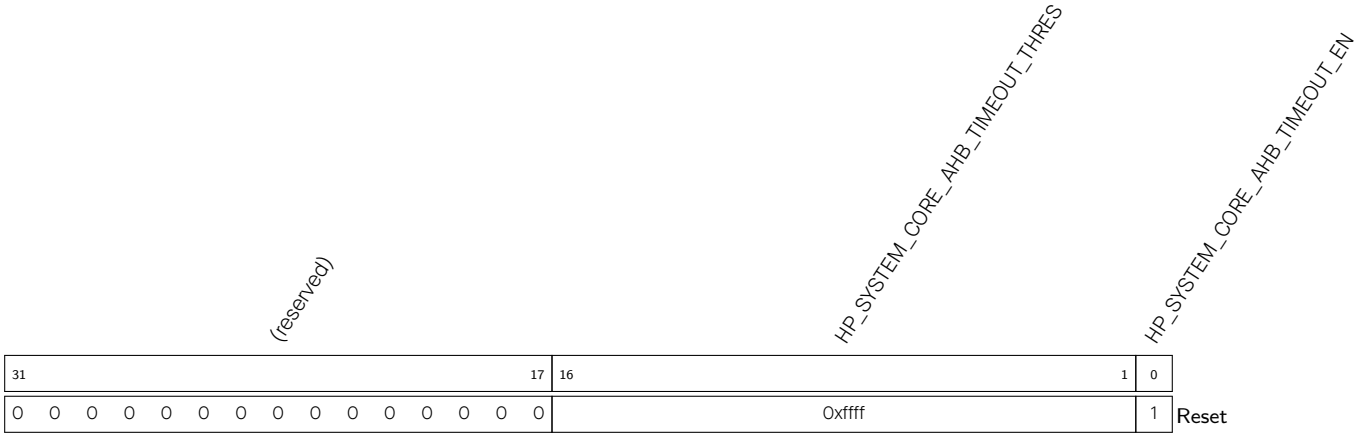
HP_SYSTEM_CORE_DEBUG_RUNSTALL_ENABLE Configures whether to enable debug runstall feature between HP CPU and LP CPU.

0: Disable

1: Enable

(R/W)

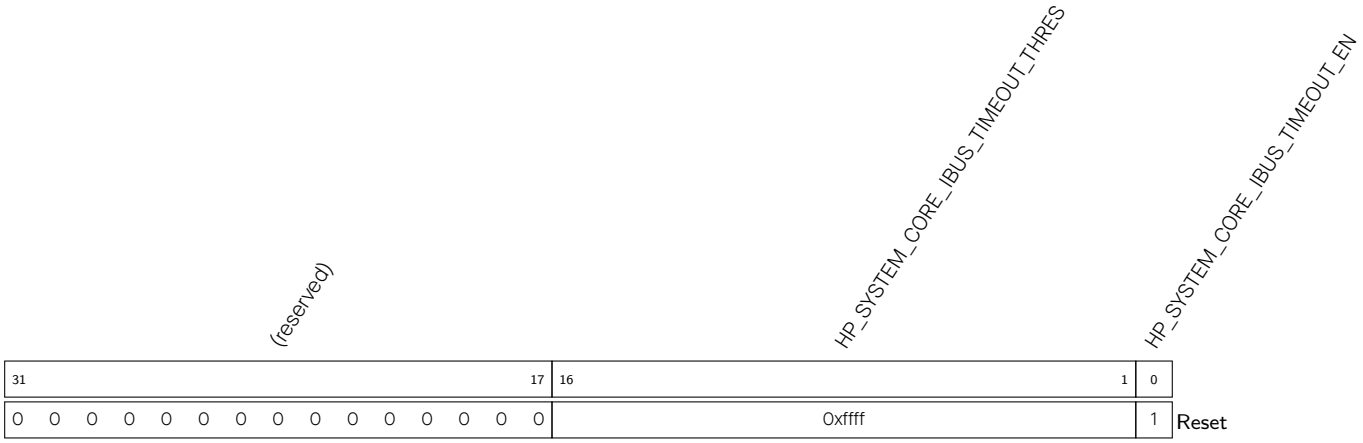
Register 21.45. HP_SYSTEM_HP_CORE_AHB_TIMEOUT_REG (0x0120)



HP_SYSTEM_CORE_AHB_TIMEOUT_EN Configures whether to enable timeout protection on HP CPU0/1 AHB bus. (R/W)

HP_SYSTEM_CORE_AHB_TIMEOUT_THRES Configures HP CPU0/1 AHB bus timeout threshold. (R/W)

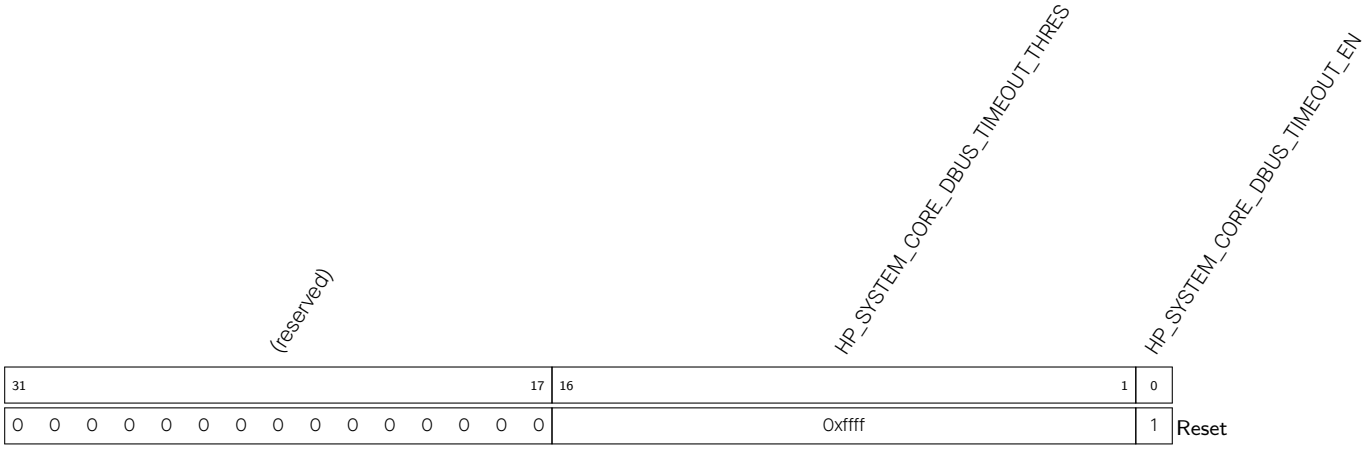
Register 21.46. HP_SYSTEM_HP_CORE_IBUS_TIMEOUT_REG (0x0124)



HP_SYSTEM_CORE_IBUS_TIMEOUT_EN Configures whether to enable timeout protection on HP CPU0/1 IBUS. (R/W)

HP_SYSTEM_CORE_IBUS_TIMEOUT_THRES Configures HP CPU0/1 IBUS timeout threshold. (R/W)

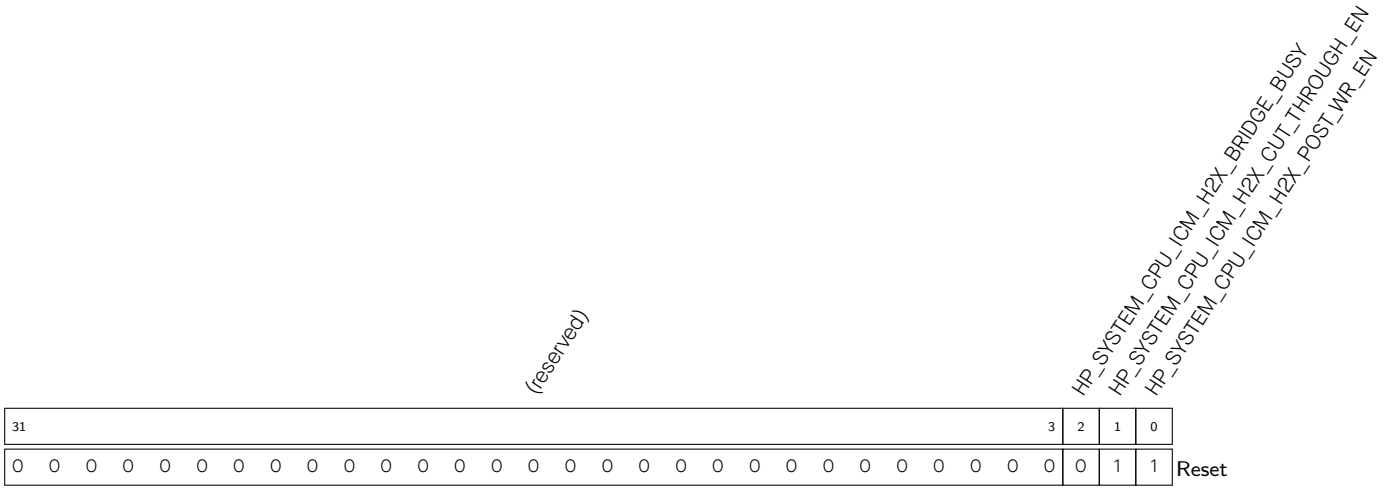
Register 21.47. HP_SYSTEM_HP_CORE_DBUS_TIMEOUT_REG (0x0128)



HP_SYSTEM_CORE_DBUS_TIMEOUT_EN Configures whether to enable timeout protection on HP CPU0/1 DBUS. (R/W)

HP_SYSTEM_CORE_DBUS_TIMEOUT_THRES Configures HP CPU0/1 DBUS timeout threshold. (R/W)

Register 21.48. HP_SYSTEM_HP_ICM_CPU_H2X_CFG_REG (0x0138)

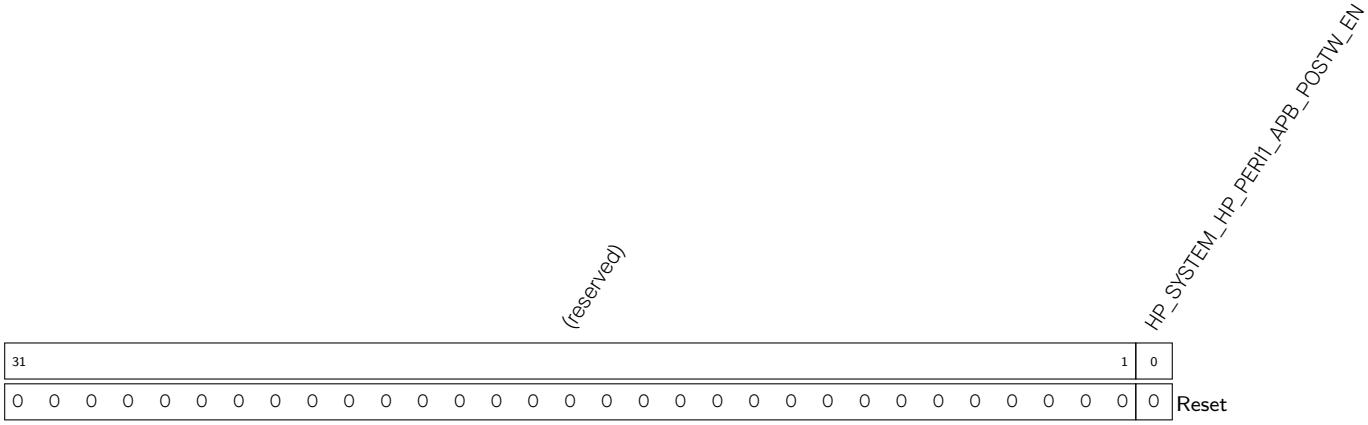


HP_SYSTEM_CPU_ICM_H2X_POST_WR_EN Configures whether to enable Post Write mode in AHB2AXI bridge for improved write performance. (R/W)

HP_SYSTEM_CPU_ICM_H2X_CUT_THROUGH_EN Configures whether to enable CUT Through mode in AHB2AXI bridge for improved write performance. (R/W)

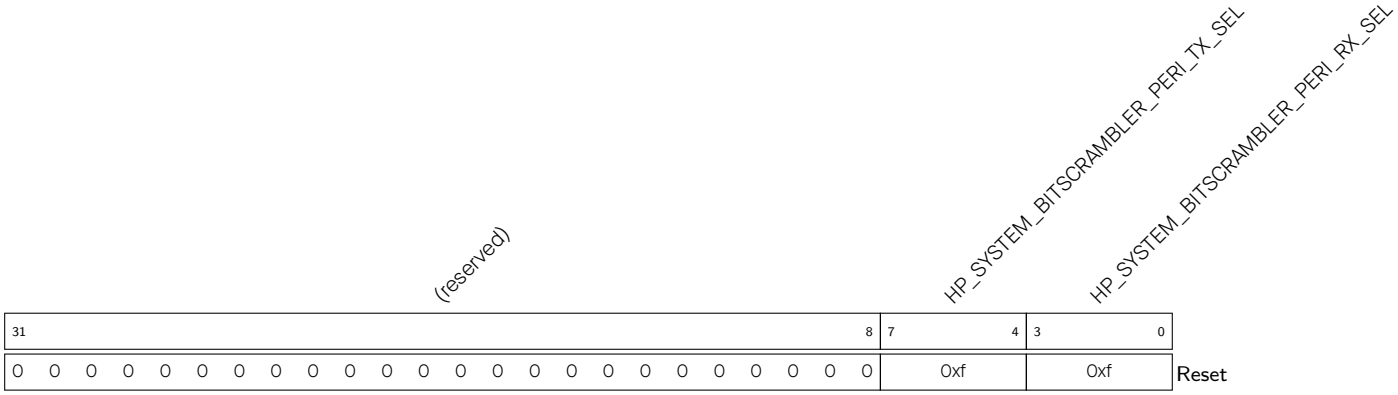
HP_SYSTEM_CPU_ICM_H2X_BRIDGE_BUSY Indicates when AHB2AXI bridge is busy. (RO)

Register 21.49. HP_SYSTEM_PERI1_APB_POSTW_EN_REG (0x013C)



HP_SYSTEM_HP_PERI1_APB_POSTW_EN Configures whether to enable Post Write mode for peripherals' APB2APB bridge, which accelerates write operations but requires time to update register values. (R/W)

Register 21.50. HP_SYSTEM_BITSCRAMBLER_PERI_SEL_REG (0x0140)



HP_SYSTEM_BITSCRAMBLER_PERI_RX_SEL Configures the DMA-capable peripheral for BitScrambler's Rx channel.

0: LCD_CAM

1: GPSPI2

2: GPSPI3

3: PARL_IO

4: AES

5: SHA

6: ADC

7: I2S0

8: I2S1

9: I2S2

10: I3C_MST

11: UHCI0

12: RMT

13: Invalid. No effect

14: Invalid. No effect

15: Invalid. No effect

(R/W)

Continued on the next page...

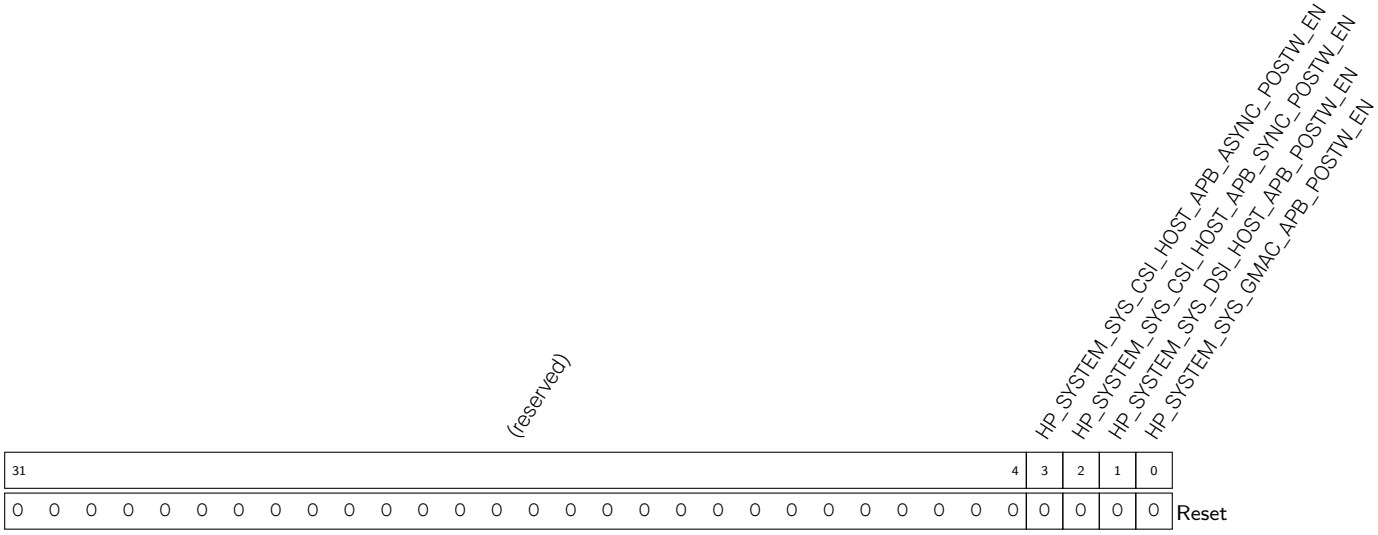
Register 21.50. HP_SYSTEM_BITSCRAMBLER_PERI_SEL_REG (0x0140)

Continued from the previous page...

HP_SYSTEM_BITSCRAMBLER_PERI_TX_SEL Configures the DMA-capable peripheral for BitScrambler's Tx channel.

0: LCD_CAM
1: GPSPI2
2: GPSPI3
3: PARL_IO
4: AES
5: SHA
6: ADC
7: I2S0
8: I2S1
9: I2S2
10: I3C_MST
11: UHCI0
12: RMT
13: Invalid. No effect
14: Invalid. No effect
15: Invalid. No effect
(R/W)

Register 21.51. HP_SYSTEM_APB_SYNC_POSTW_EN_REG (0x0144)



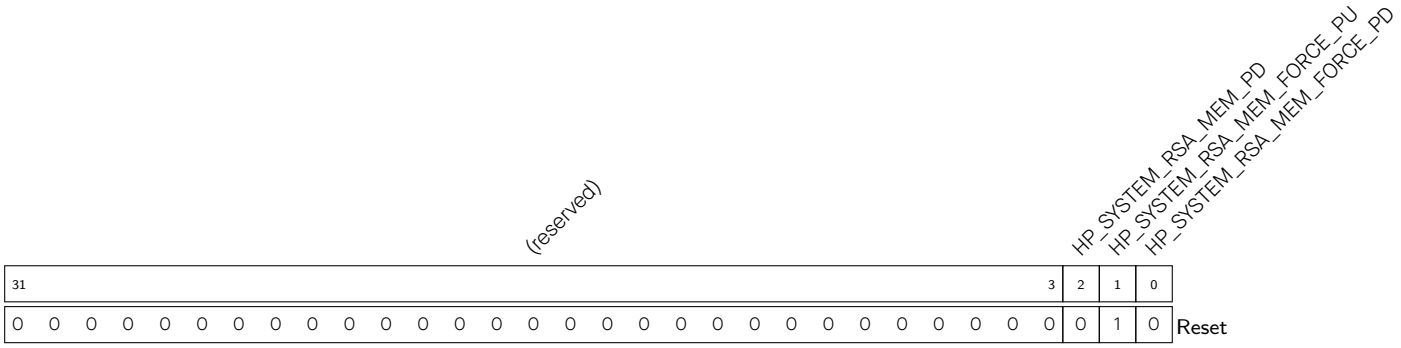
HP_SYSTEM_SYS_GMAC_APB_POSTW_EN Configures whether to enable Post Write mode for synchronous APB2APB bridge in GMAC, which accelerates write operations but requires time to update register values. (R/W)

HP_SYSTEM_SYS_DSI_HOST_APB_POSTW_EN Configures whether to enable Post Write mode for synchronous APB2APB bridge in DSI Host, which accelerates write operations but requires time to update register values. (R/W)

HP_SYSTEM_SYS_CSI_HOST_APB_SYNC_POSTW_EN Configures whether to enable Post Write mode for synchronous APB2APB bridge in CSI Host, which accelerates write operations but requires time to update register values. (R/W)

HP_SYSTEM_SYS_CSI_HOST_APB_ASYNC_POSTW_EN Configures whether to enable Post Write mode for asynchronous APB2APB bridge in CSI Host, which accelerates write operations but requires time to update register values. (R/W)

Register 21.52. HP_SYSTEM_HP_RSA_PD_CTRL_REG (0x01D0)



HP_SYSTEM_RSA_MEM_FORCE_PD Configures whether to force power down RSA internal memory. This setting has a lower priority than HP_SYSTEM_RSA_MEM_FORCE_PU.

0: No effect

1: Power down

(R/W)

HP_SYSTEM_RSA_MEM_FORCE_PU Configures whether to force power up RSA internal memory.

0: No effect

1: Power up

(R/W)

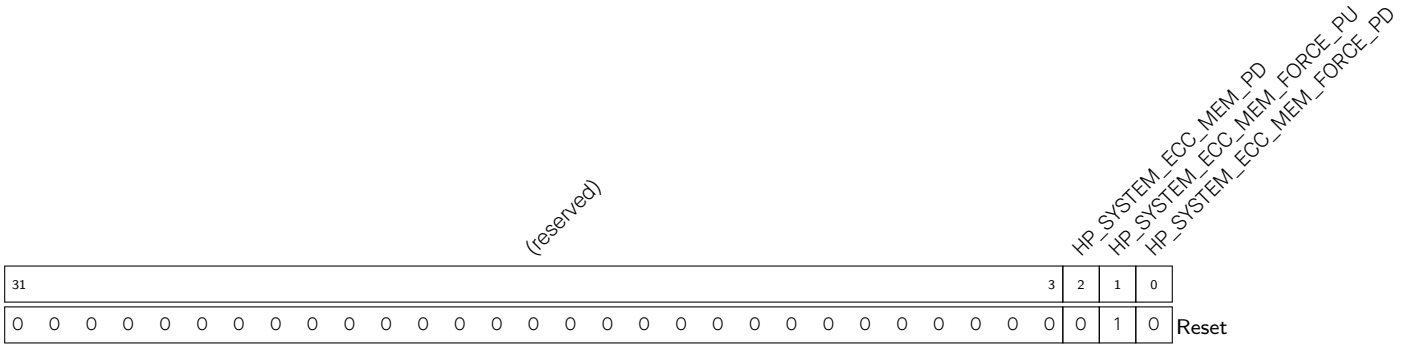
HP_SYSTEM_RSA_MEM_PD Configures whether to power down RSA internal memory.

0: No effect

1: Power down

(R/W)

Register 21.53. HP_SYSTEM_HP_ECC_PD_CTRL_REG (0x01D4)



HP_SYSTEM_ECC_MEM_FORCE_PD Configures whether to force power down ECC internal memory. This setting has a lower priority than HP_SYSTEM_ECC_MEM_FORCE_PU.

0: No effect

1: Power down

(R/W)

HP_SYSTEM_ECC_MEM_FORCE_PU Configures whether to force power up ECC internal memory.

0: No effect

1: Power up

(R/W)

HP_SYSTEM_ECC_MEM_PD Configures whether to power down ECC internal memory.

0: No effect

1: Power down

(R/W)

Register 21.54. HP_SYSTEM_HP_RNG_CFG_REG (0x01D8)



HP_SYSTEM_RNG_SAMPLE_ENABLE Configures whether to enable the RNG sample chain.

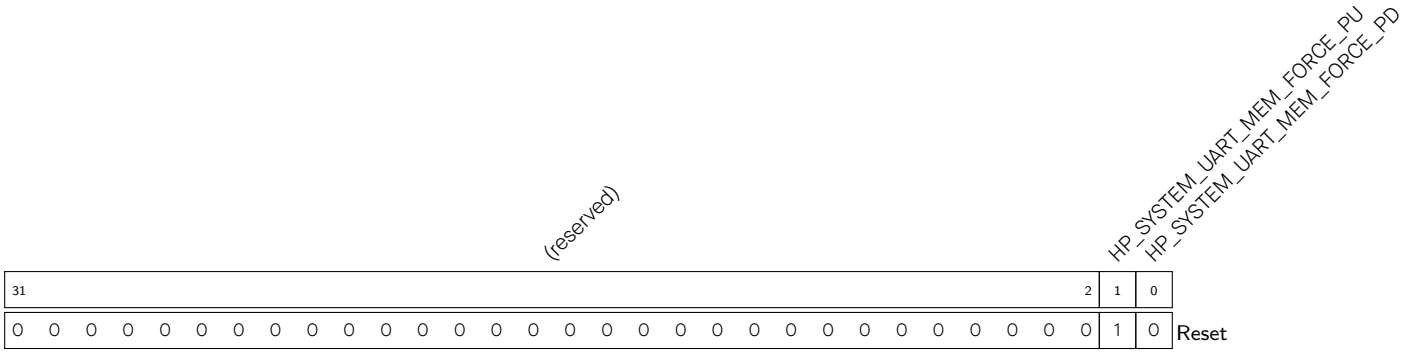
0: Disable

1: Enable

(R/W)

HP_SYSTEM_RNG_CHAIN_CLK_DIV_NUM Configures the chain clock division number for debugging. (R/W)

Register 21.55. HP_SYSTEM_HP_UART_PD_CTRL_REG (0x01DC)



HP_SYSTEM_UART_MEM_FORCE_PD Configures whether to force power down HP UART internal memory.

0: No effect

1: Power down

(R/W)

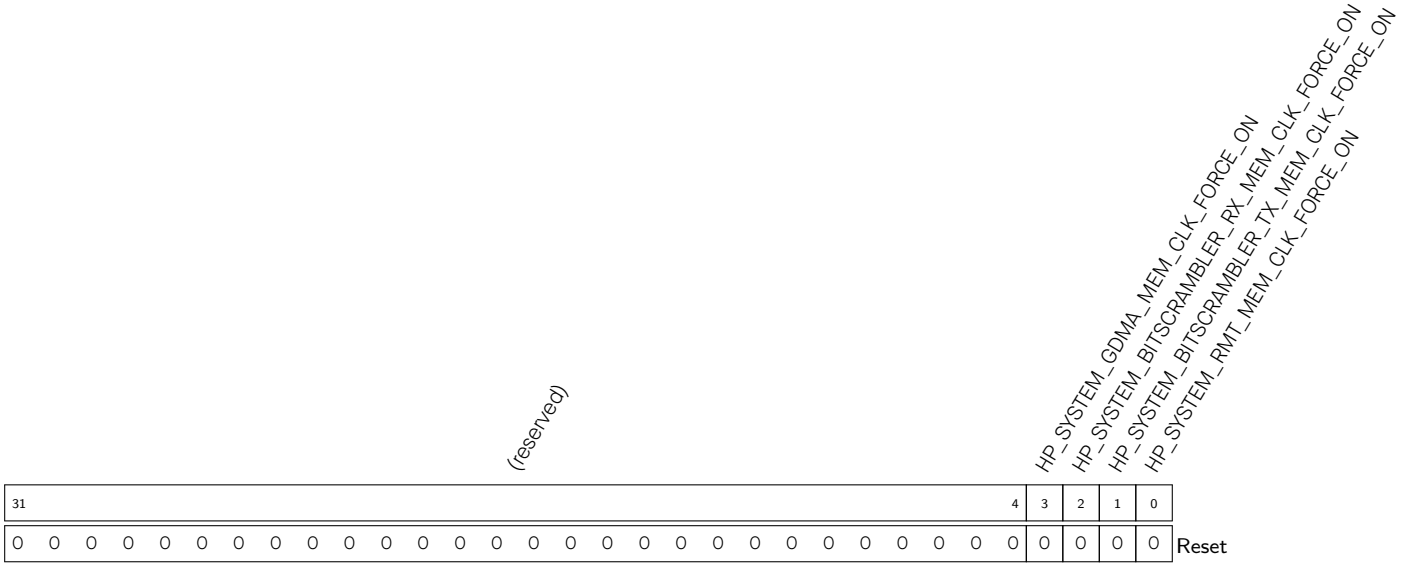
HP_SYSTEM_UART_MEM_FORCE_PU Configures whether to force power up HP UART internal memory.

0: No effect

1: Power up

(R/W)

Register 21.56. HP_SYSTEM_HP_PERI_MEM_CLK_FORCE_ON_REG (0x01E0)



HP_SYSTEM_RMT_MEM_CLK_FORCE_ON Configures whether to force enable the clock gating of the memory clock in RMT.

0: No effect

1: Force on

(R/W)

HP_SYSTEM_BITSCRAMBLER_TX_MEM_CLK_FORCE_ON Configures whether to force enable the clock gating of the TX memory clock in BitScrambler.

0: No effect

1: Force on

(R/W)

HP_SYSTEM_BITSCRAMBLER_RX_MEM_CLK_FORCE_ON Configures whether to force enable the clock gating of the RX memory clock in BitScrambler.

0: No effect

1: Force on

(R/W)

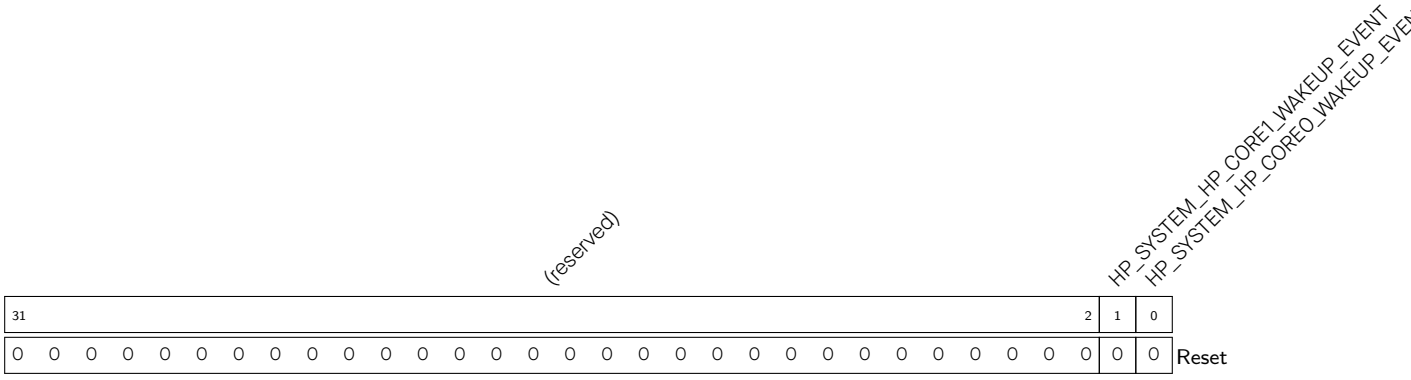
HP_SYSTEM_GDMA_MEM_CLK_FORCE_ON Configures whether to force enable the clock gating of the memory clock in GDMA.

0: No effect

1: Force on

(R/W)

Register 21.57. HP_SYSTEM_HP_CPU_WAKEUP_EVENT_REG (0x01EC)



HP_SYSTEM_HP_CORE0_WAKEUP_EVENT Configures whether to wake up HP CPU0.

0: No effect

1: Wake up

(R/W)

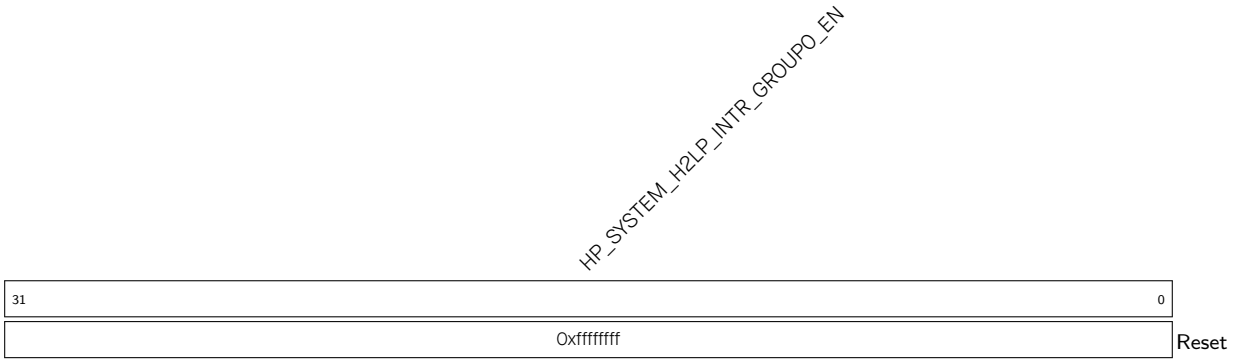
HP_SYSTEM_HP_CORE1_WAKEUP_EVENT Configures whether to wake up HP CPU1.

0: No effect

1: Wake up

(R/W)

Register 21.58. HP_SYSTEM_HP2LP_INTR_GROUP0_EN_REG (0x01F0)



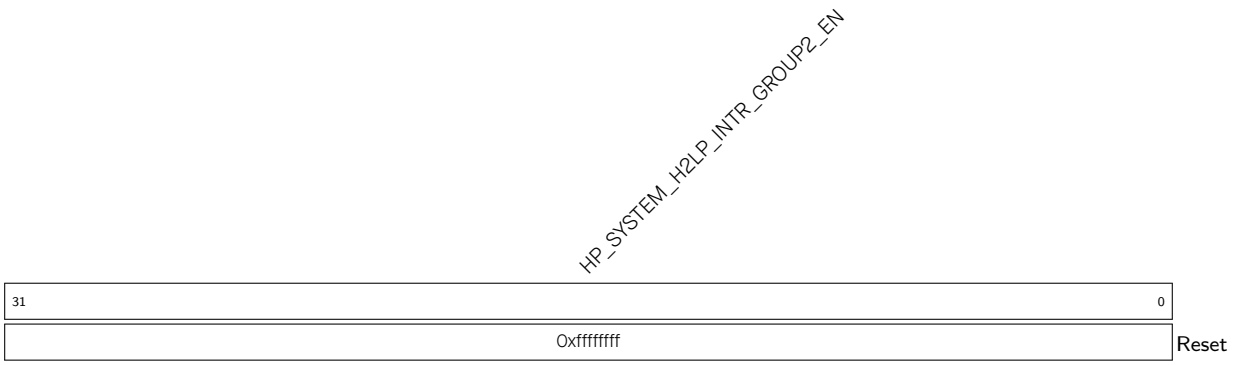
HP_SYSTEM_H2LP_INTR_GROUP0_EN Write 1 to enable the corresponding peripheral interrupt to LP CPU. (R/W)

Register 21.59. HP_SYSTEM_HP2LP_INTR_GROUP1_EN_REG (0x01F4)



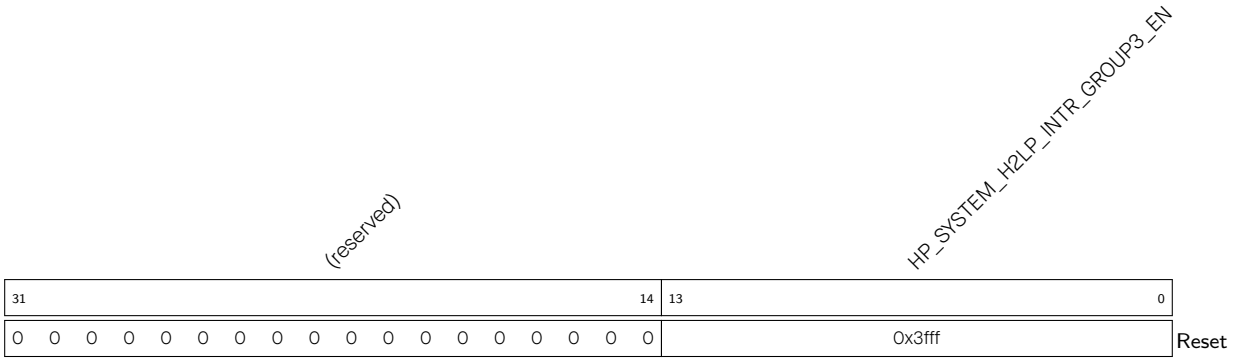
HP_SYSTEM_H2LP_INTR_GROUP1_EN Write 1 to enable the corresponding peripheral interrupt to LP CPU. (R/W)

Register 21.60. HP_SYSTEM_HP2LP_INTR_GROUP2_EN_REG (0x01F8)



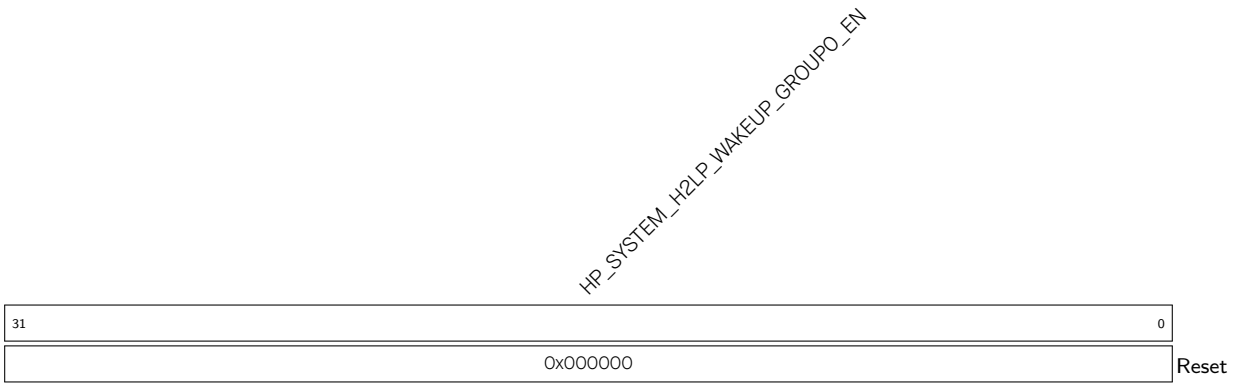
HP_SYSTEM_H2LP_INTR_GROUP2_EN Write 1 to enable the corresponding peripheral interrupt to LP CPU. (R/W)

Register 21.61. HP_SYSTEM_HP2LP_INTR_GROUP3_EN_REG (0x01FC)



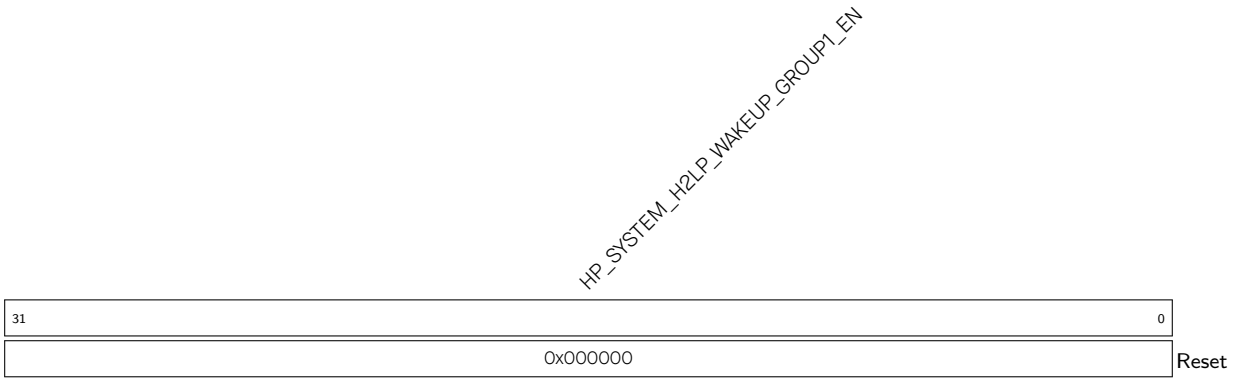
HP_SYSTEM_H2LP_INTR_GROUP3_EN Write 1 to enable the corresponding peripheral interrupt to LP CPU. (R/W)

Register 21.62. HP_SYSTEM_HP2LP_WAKEUP_GROUP0_EN_REG (0x0210)



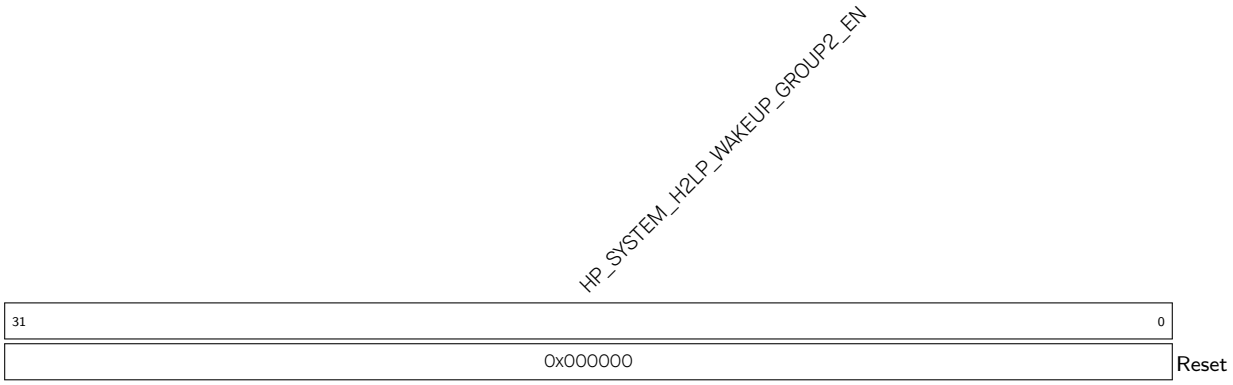
HP_SYSTEM_H2LP_WAKEUP_GROUP0_EN Write 1 to enable the corresponding peripheral wakeup to PMU. (R/W)

Register 21.63. HP_SYSTEM_HP2LP_WAKEUP_GROUP1_EN_REG (0x0214)



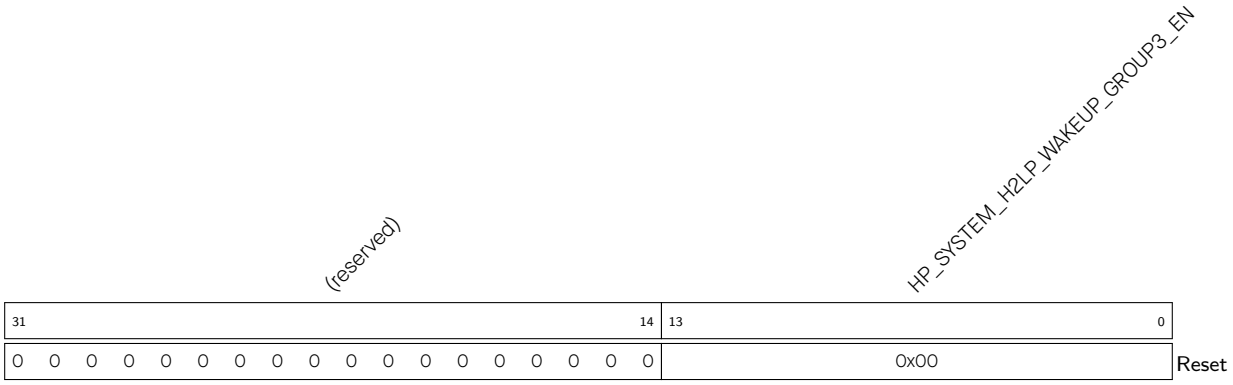
HP_SYSTEM_H2LP_WAKEUP_GROUP1_EN Write 1 to enable the corresponding peripheral wakeup to PMU. (R/W)

Register 21.64. HP_SYSTEM_HP2LP_WAKEUP_GROUP2_EN_REG (0x0218)



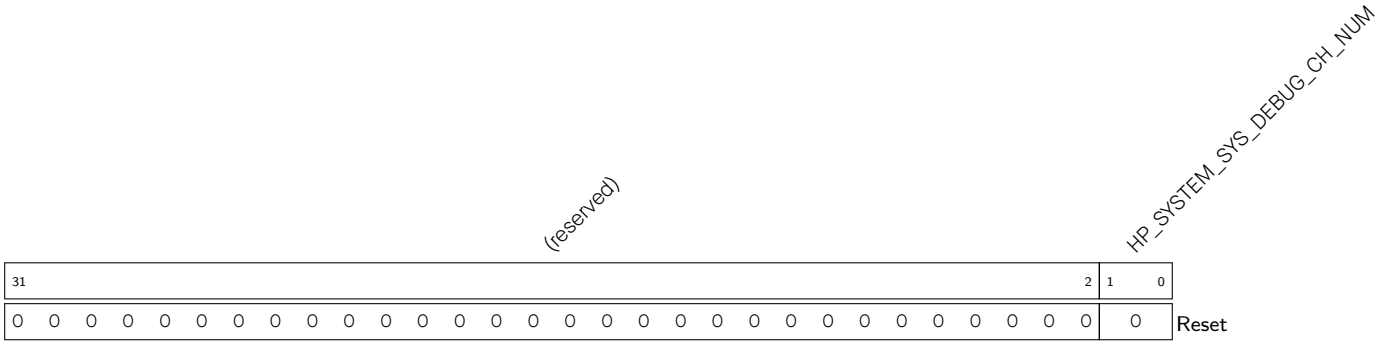
HP_SYSTEM_H2LP_WAKEUP_GROUP2_EN Write 1 to enable the corresponding peripheral wakeup to PMU. (R/W)

Register 21.65. HP_SYSTEM_HP2LP_WAKEUP_GROUP3_EN_REG (0x021C)



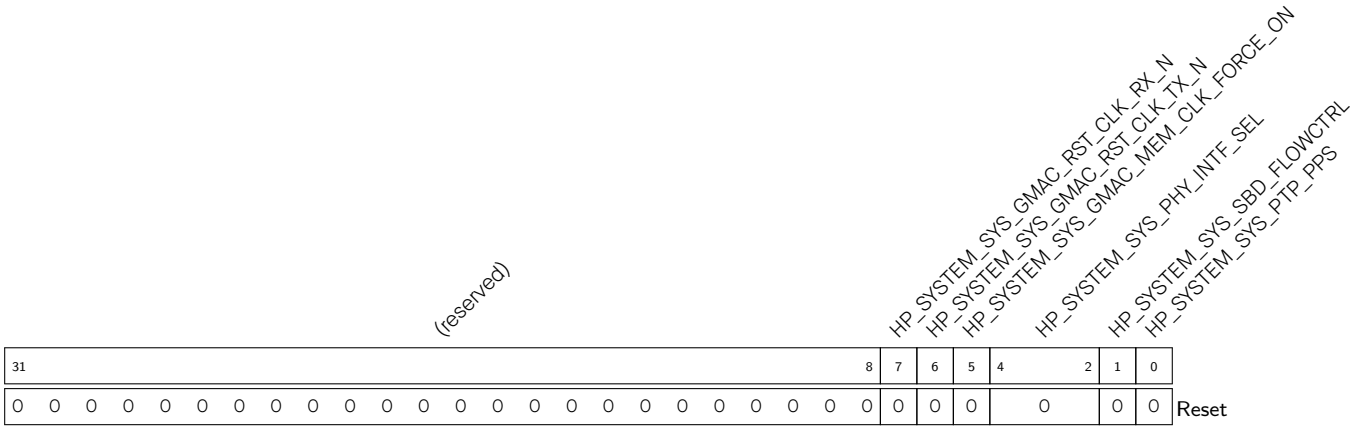
HP_SYSTEM_H2LP_WAKEUP_GROUP3_EN Write 1 to enable the corresponding peripheral wakeup to PMU. (R/W)

Register 21.66. HP_SYSTEM_GDMA_CTRL_REG (0x0148)



HP_SYSTEM_SYS_DEBUG_CH_NUM Configures the number of GDMA debug channels. (R/W)

Register 21.67. HP_SYSTEM_GMAC_CTRL0_REG (0x014C)



HP_SYSTEM_SYS_PTP_PPS Represents pulse per second output from Ethernet MAC. (RO)

HP_SYSTEM_SYS_SBD_FLOWCTRL Configures sideband flow control signal.

0: No effect

1: Instruct Ethernet MAC to transmit Pause frames in Full-duplex mode and enable back-pressure in half-duplex mode.

(R/W)

HP_SYSTEM_SYS_PHY_INTF_SEL Configures Ethernet PHY interface.

0: MII

1: Invalid. No effect

2: Invalid. No effect

3: Invalid. No effect

4: RMII

5: Invalid. No effect

6: Invalid. No effect

7: Invalid. No effect

(R/W)

HP_SYSTEM_SYS_GMAC_MEM_CLK_FORCE_ON Configures whether to force enable the memory clock of GMAC memory.

0: No effect

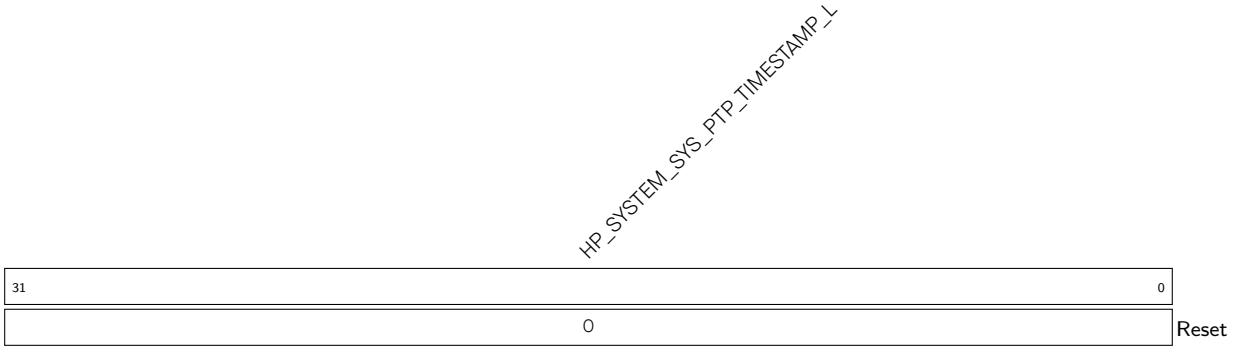
1: Force on

(R/W)

HP_SYSTEM_SYS_GMAC_RST_CLK_TX_N Represents an active-low reset signal synchronized to the Ethernet MAC Tx clock. (RO)

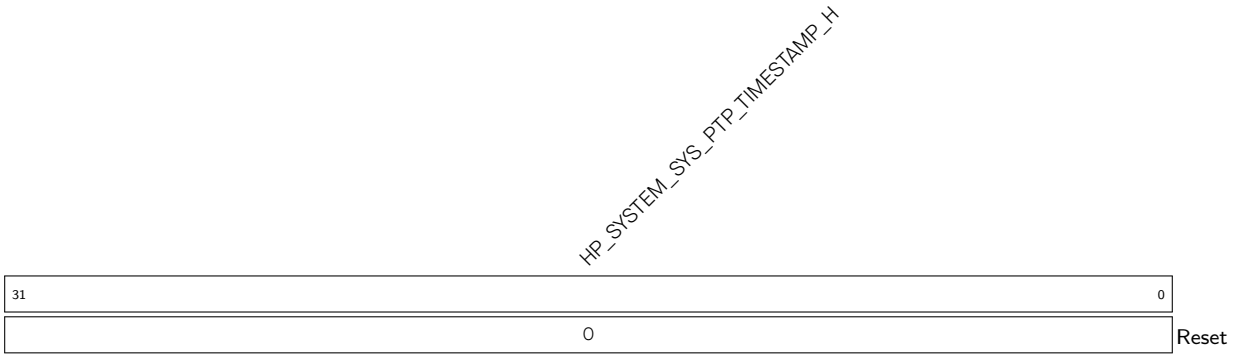
HP_SYSTEM_SYS_GMAC_RST_CLK_RX_N Represents an active-low reset signal synchronized to the Ethernet MAC Rx clock. (RO)

Register 21.68. HP_SYSTEM_GMAC_CTRL1_REG (0x0150)



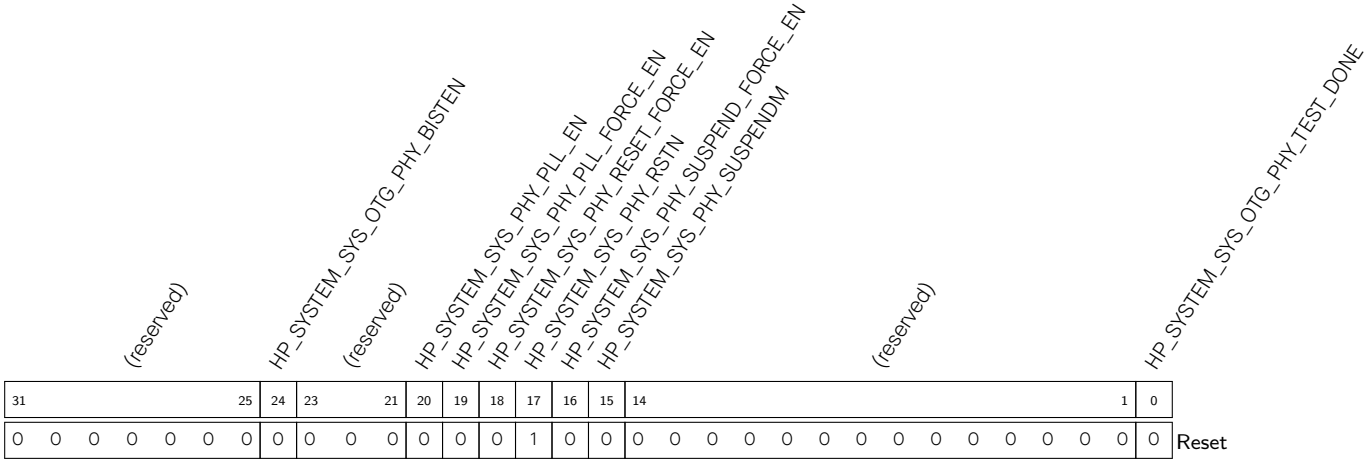
HP_SYSTEM_SYS_PTP_TIMESTAMP_L Represents the lower 32 bits of timestamp output from Ethernet MAC synchronized to clk_ptp_ref clock. (RO)

Register 21.69. HP_SYSTEM_GMAC_CTRL2_REG (0x0154)



HP_SYSTEM_SYS_PTP_TIMESTAMP_H Represents the upper 32 bits of timestamp output from Ethernet MAC synchronized to clk_ptp_ref clock. (RO)

Register 21.70. HP_SYSTEM_USBOTG20_CTRL_REG (0x015C)



HP_SYSTEM_SYS_OTG_PHY_TEST_DONE Indicates the status of USB OTG_HS PHY BIST test.

0: Fail

1: Pass

(RO)

HP_SYSTEM_SYS_PHY_SUSPENDM Configures whether to drive USB OTG_HS PHY into Suspend mode. This setting is only valid when HP_SYSTEM_PHY_SUSPEND_FORCE_EN is set to 1.

0: No effect

1: In Suspend mode

(R/W)

HP_SYSTEM_SYS_PHY_SUSPEND_FORCE_EN Configures whether to allow forcing USB OTG_HS PHY into Suspend mode by setting HP_SYSTEM_PHY_SUSPENDM.

0: No effect

1: Force in Suspend mode

(R/W)

HP_SYSTEM_SYS_PHY_RSTN Configures whether to reset USB OTG_HS PHY. This setting is only valid when HP_SYSTEM_PHY_RESET_FORCE_EN is set to 1.

0: Reset

1: No effect

(R/W)

HP_SYSTEM_SYS_PHY_RESET_FORCE_EN Configures whether to allow forcibly resetting USB OTG_HS PHY by clearing HP_SYSTEM_PHY_RSTN.

0: No effect

1: Force reset

(R/W)

Continued on the next page...

Register 21.70. HP_SYSTEM_USBOTG20_CTRL_REG (0x015C)

Continued from the previous page...

HP_SYSTEM_SYS_PHY_PLL_FORCE_EN Configures whether to allow forcibly enabling USB OTG_HS PHY PLL by setting HP_SYSTEM_PHY_PLL_EN.

0: No effect

1: Force enable

(R/W)

HP_SYSTEM_SYS_PHY_PLL_EN Configures whether to enable USB OTG_HS PHY PLL. This setting is only valid when HP_SYSTEM_PHY_PLL_FORCE_EN is set.

0: No effect

1: Enable

(R/W)

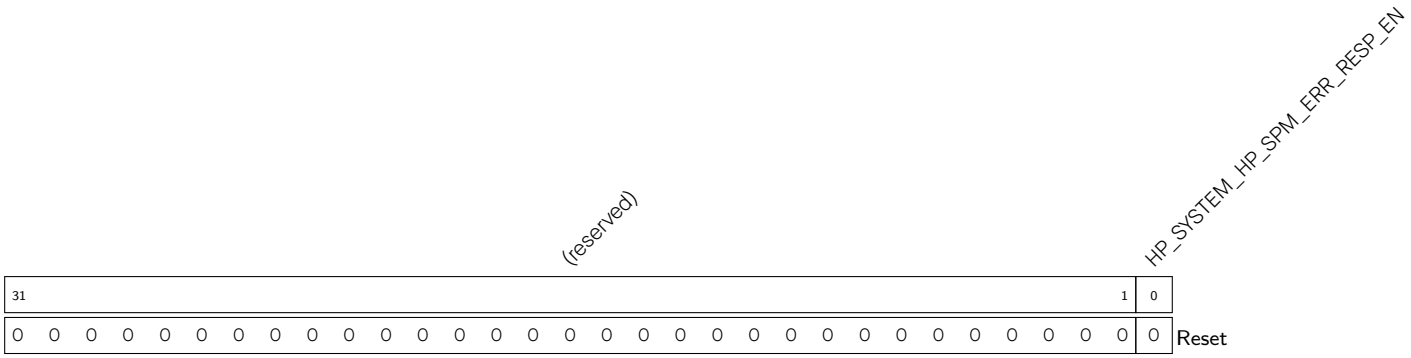
HP_SYSTEM_SYS_OTG_PHY_BISTEN Configures whether to drive USB OTG_HS PHY into BIST mode and perform self-test.

0: No effect

1: In BIST mode

(R/W)

Register 21.71. HP_SYSTEM_HP_SPM_ERR_RESP_CTRL_REG (0x0160)



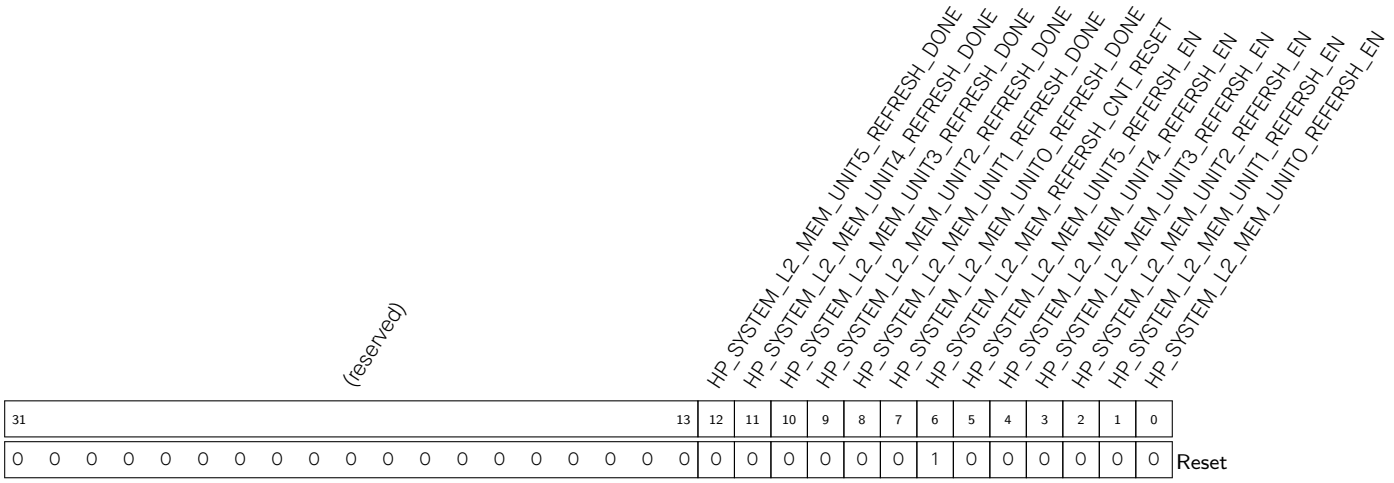
HP_SYSTEM_HP_SPM_ERR_RESP_EN Configures whether to report parity error to the HP CPU and trigger exception.

0: Disable

1: Enable

(R/W)

Register 21.72. HP_SYSTEM_L2_MEM_REFRESH_REG (0x0164)



HP_SYSTEM_L2_MEM_UNIT0_REFERSH_EN Configures whether to refresh the ECC bit of L2MEM unit0.

0: Disable

1: Enable

(R/W)

HP_SYSTEM_L2_MEM_UNIT1_REFERSH_EN Configures whether to refresh the ECC bit of L2MEM unit1.

0: Disable

1: Enable

(R/W)

HP_SYSTEM_L2_MEM_UNIT2_REFERSH_EN Configures whether to refresh the ECC bit of L2MEM unit2.

0: Disable

1: Enable

(R/W)

HP_SYSTEM_L2_MEM_UNIT3_REFERSH_EN Configures whether to refresh the ECC bit of L2MEM unit3.

0: Disable

1: Enable

(R/W)

HP_SYSTEM_L2_MEM_UNIT4_REFERSH_EN Configures whether to refresh the ECC bit of L2MEM unit4.

0: Disable

1: Enable

(R/W)

Continued on the next page...

Register 21.72. HP_SYSTEM_L2_MEM_REFRESH_REG (0x0164)

Continued from the previous page...

HP_SYSTEM_L2_MEM_UNIT5_REFERSH_EN Configures whether to refresh the ECC bit of L2MEM unit5.
0: Disable
1: Enable
(R/W)

HP_SYSTEM_L2_MEM_REFERSH_CNT_RESET Configures whether to reset L2MEM refresh count.
0: No effect
1: Reset
(R/W)

HP_SYSTEM_L2_MEM_UNIT0_REFRESH_DONE Represents the refresh status of L2MEM unit0 ECC bit. (RO)

HP_SYSTEM_L2_MEM_UNIT1_REFRESH_DONE Represents the refresh status of L2MEM unit1 ECC bit. (RO)

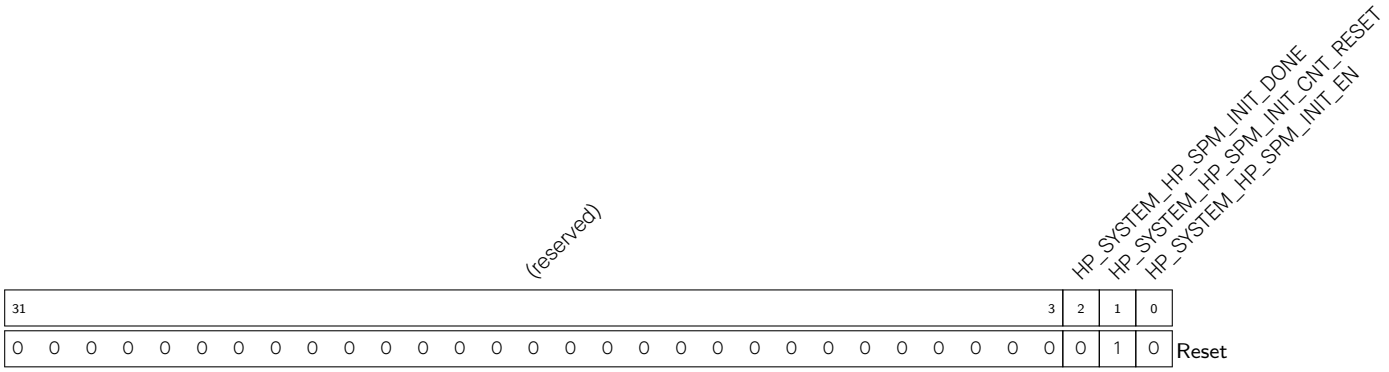
HP_SYSTEM_L2_MEM_UNIT2_REFRESH_DONE Represents the refresh status of L2MEM unit2 ECC bit. (RO)

HP_SYSTEM_L2_MEM_UNIT3_REFRESH_DONE Represents the refresh status of L2MEM unit3 ECC bit. (RO)

HP_SYSTEM_L2_MEM_UNIT4_REFRESH_DONE Represents the refresh status of L2MEM unit4 ECC bit. (RO)

HP_SYSTEM_L2_MEM_UNIT5_REFRESH_DONE Represents the refresh status of L2MEM unit5 ECC bit. (RO)

Register 21.73. HP_SYSTEM_HP_SPM_INIT_REG (0x0168)



HP_SYSTEM_HP_SPM_INIT_EN Configures whether to enable HP SPM parity bit initialization.

0: Disable

1: Enable

(R/W)

HP_SYSTEM_HP_SPM_INIT_CNT_RESET Configures whether to reset SPM initialization count.

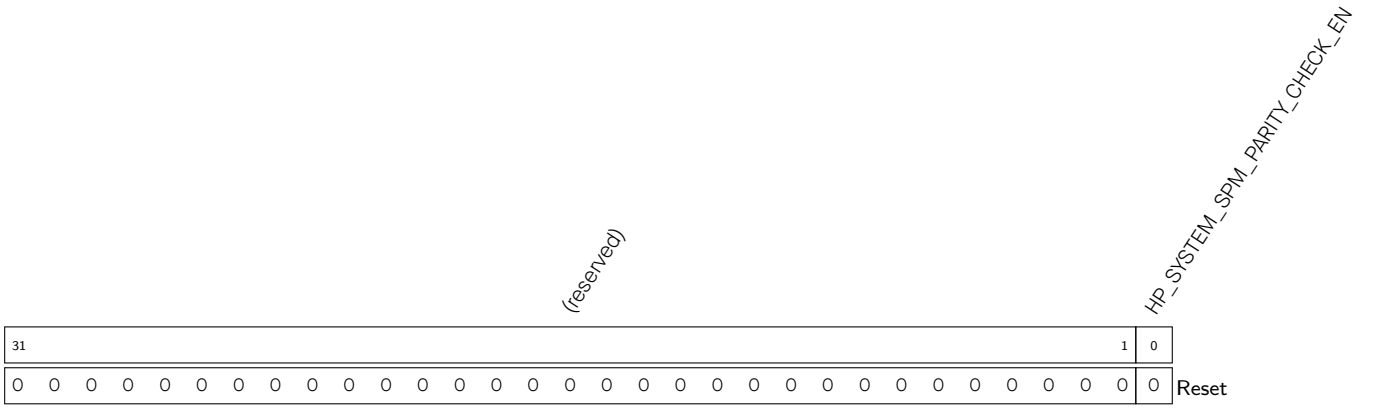
0: No effect

1: Reset

(R/W)

HP_SYSTEM_HP_SPM_INIT_DONE Represents the initialization status of HP SPM parity bits. (RO)

Register 21.74. HP_SYSTEM_HP_SPM_PARITY_CHECK_CTRL_REG (0x016C)



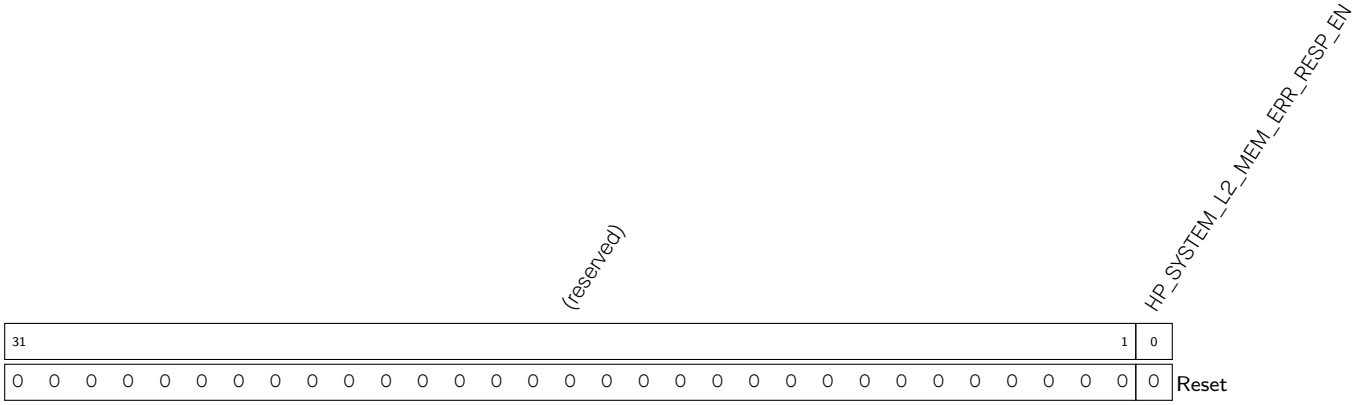
HP_SYSTEM_HP_SPM_PARITY_CHECK_EN Configures whether to enable parity error check in HP SPM.

0: Disable

1: Enable

(R/W)

Register 21.75. HP_SYSTEM_L2_MEM_ERR_RESP_CTRL_REG (0x0198)



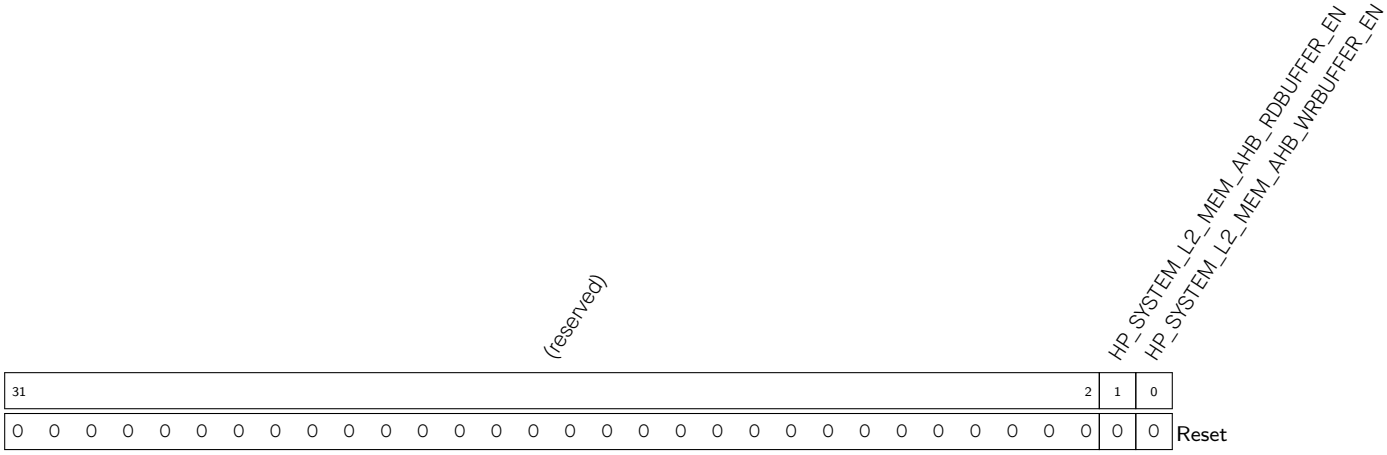
HP_SYSTEM_L2_MEM_ERR_RESP_EN Configures whether to report ECC error to the HP CPU and trigger exception.

0: Disable

1: Enable

(R/W)

Register 21.76. HP_SYSTEM_L2_MEM_AHB_BUFFER_CTRL_REG (0x019C)



HP_SYSTEM_L2_MEM_AHB_WRBUFFER_EN Configures whether to enable L2MEM AHB write buffer.

0: Disable

1: Enable

(R/W)

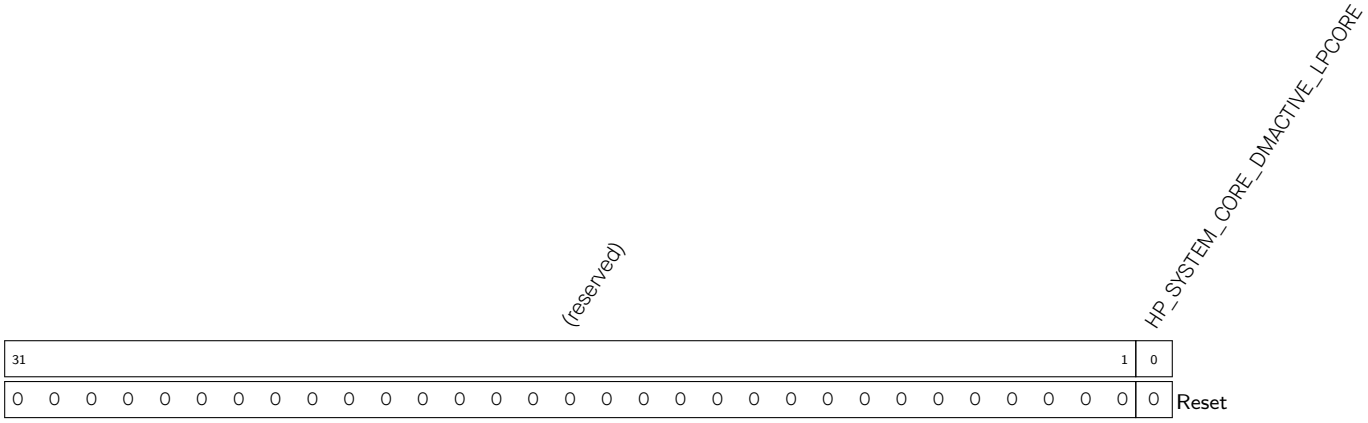
HP_SYSTEM_L2_MEM_AHB_RDBUFFER_EN Configures whether to enable L2MEM AHB read buffer.

0: Disable

1: Enable

(R/W)

Register 21.77. HP_SYSTEM_HP_CORE_DMACTIVE_LPCORE_REG (0x01A0)



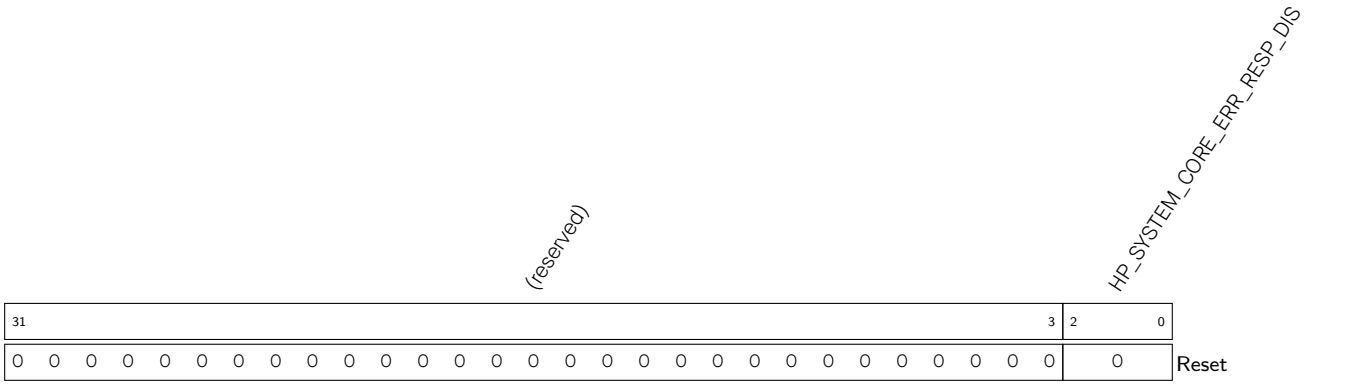
HP_SYSTEM_CORE_DMACTIVE_LPCORE Represents the dmactive value of LP CPU's debug module.

0: Indicates that the LP CPU is not connected to the JTAG interface, or is not in debug mode.

1: Indicates that the LP CPU is connected to the JTAG interface and is in debug mode.

(RO)

Register 21.78. HP_SYSTEM_HP_CORE_ERR_RESP_DIS_REG (0x01A4)



HP_SYSTEM_CORE_ERR_RESP_DIS Configures whether to disable the error response on different buses.

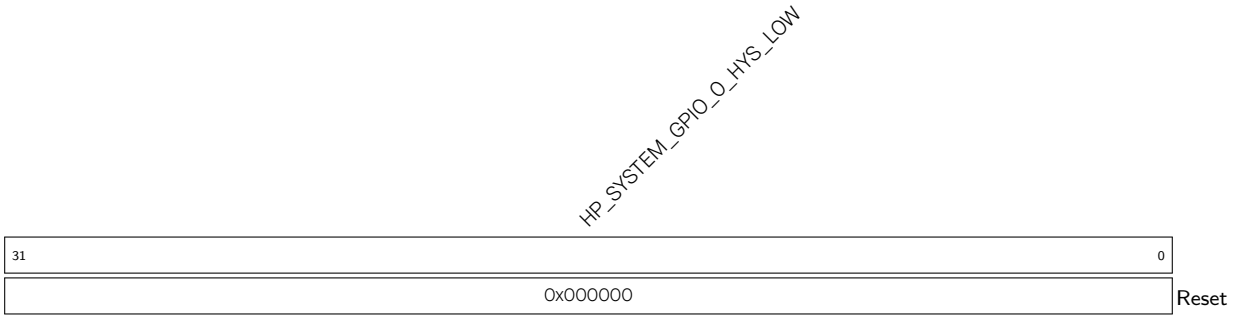
Bit[2]: AHB bus

Bit[1]: DBUS

Bit[0]: IBUS

(R/W)

Register 21.79. HP_SYSTEM_GPIO_O_HYS_CTRL0_REG (0x01C0)



HP_SYSTEM_GPIO_O_HYS_LOW Configures whether to enable the hysteresis feature of HP GPIO16 ~ HP GPIO47.

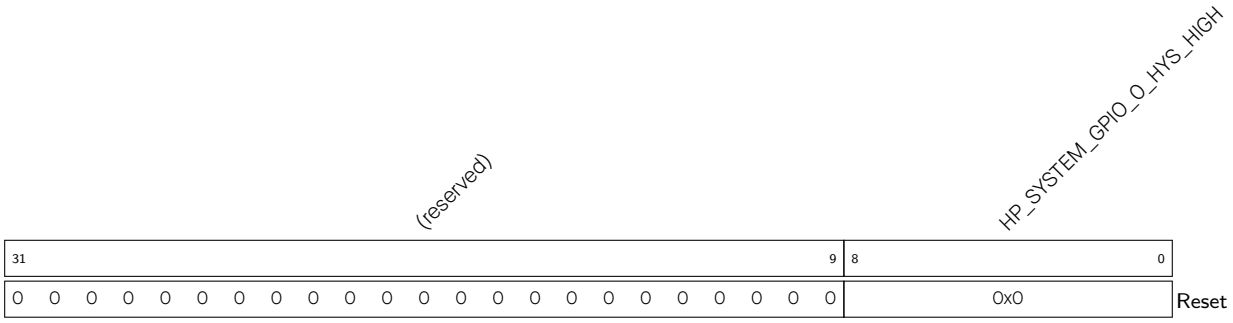
0: No effect

1: Enable

See detailed information in Section 10.10 of Chapter 10 *GPIO Matrix and IO MUX*.

(R/W)

Register 21.80. HP_SYSTEM_GPIO_O_HYS_CTRL1_REG (0x01C4)



HP_SYSTEM_GPIO_O_HYS_HIGH Configures whether to enable the hysteresis feature of HP GPIO48 ~ HP GPIO56.

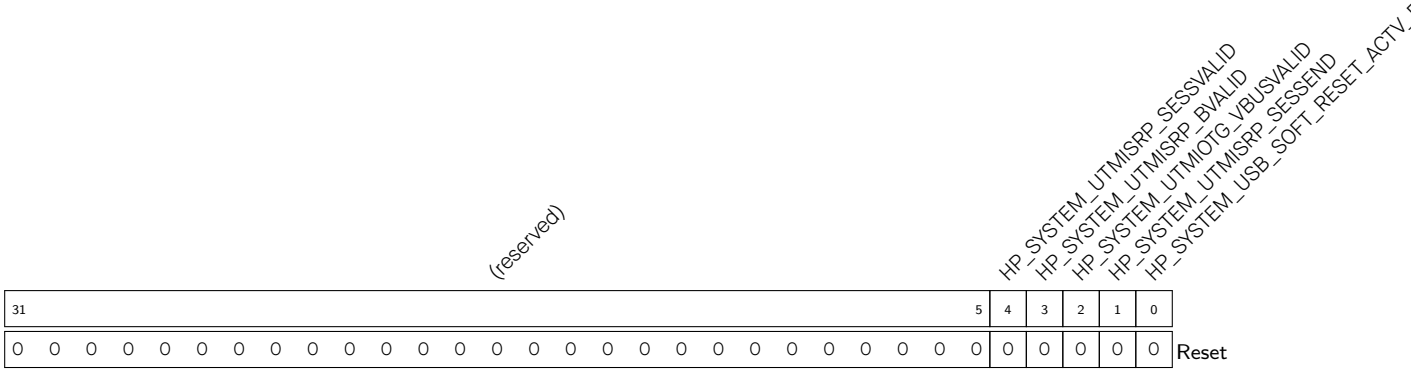
0: No effect

1: Enable

See detailed information in Section 10.10 of Chapter 10 *GPIO Matrix and IO MUX*.

(R/W)

Register 21.81. HP_SYSTEM_USB_OTGHS_PHY_ST_REG (0x01E8)



HP_SYSTEM_USB_SOFT_RESET_ACTV_PDOMAIN Represents the USB soft reset active status in the power domain. (RO)

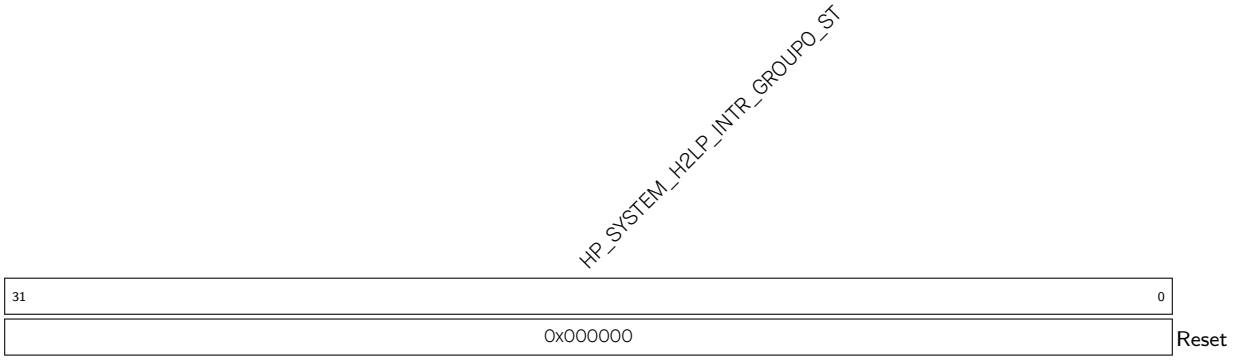
HP_SYSTEM_UTMISRP_SESSEND Represents the USB OTG2.0 session end status. (RO)

HP_SYSTEM_UTMIOTG_VBUSVALID Represents the USB OTG2.0 VBUS valid status. (RO)

HP_SYSTEM_UTMISRP_BVALID Represents the USB OTG2.0 B-valid status. (RO)

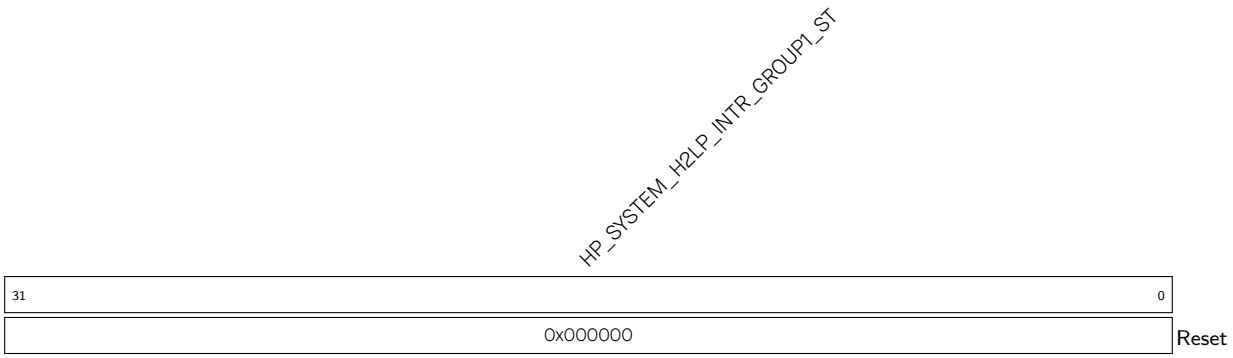
HP_SYSTEM_UTMISRP_SESSVALID Represents the USB OTG2.0 session valid status. (RO)

Register 21.82. HP_SYSTEM_HP2LP_INTR_GROUP0_ST_REG (0x0200)



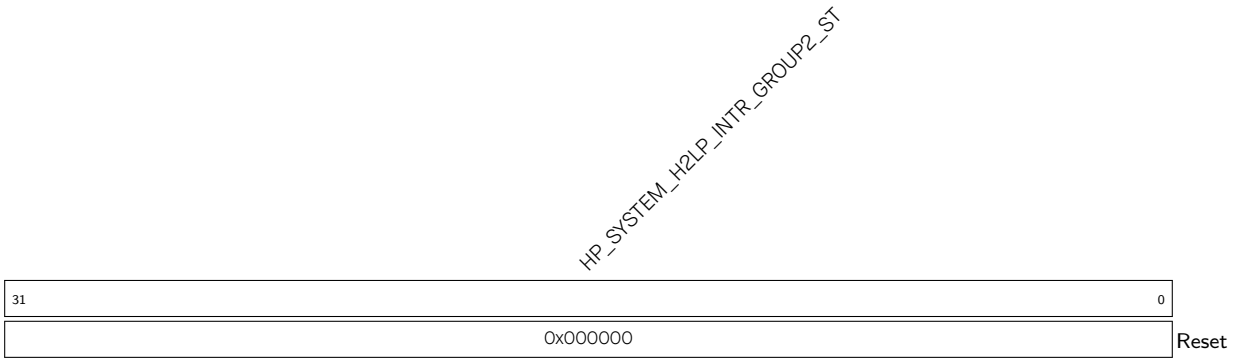
HP_SYSTEM_H2LP_INTR_GROUP0_ST Each bit represents the status of the corresponding peripheral interrupt to LP CPU. (RO)

Register 21.83. HP_SYSTEM_HP2LP_INTR_GROUP1_ST_REG (0x0204)



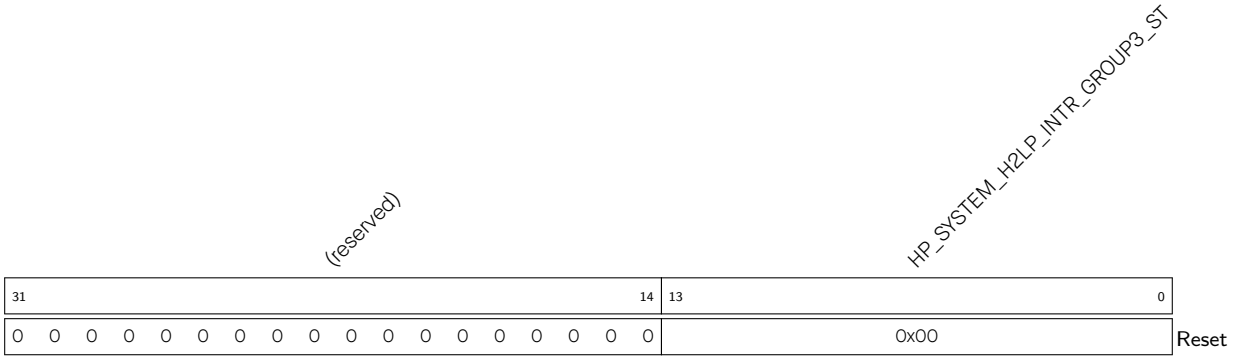
HP_SYSTEM_H2LP_INTR_GROUP1_ST Each bit represents the status of the corresponding peripheral interrupt to LP CPU. (RO)

Register 21.84. HP_SYSTEM_HP2LP_INTR_GROUP2_ST_REG (0x0208)



HP_SYSTEM_H2LP_INTR_GROUP2_ST Each bit represents the status of the corresponding peripheral interrupt to LP CPU. (RO)

Register 21.85. HP_SYSTEM_HP2LP_INTR_GROUP3_ST_REG (0x020C)



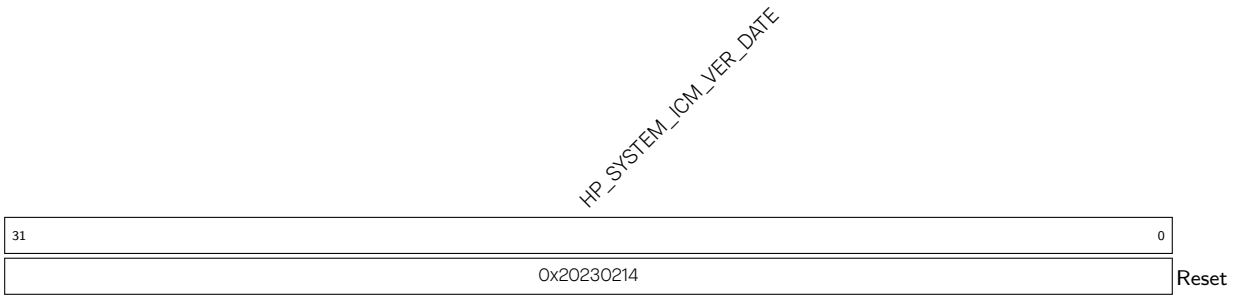
HP_SYSTEM_H2LP_INTR_GROUP3_ST Each bit represents the status of the corresponding peripheral interrupt to LP CPU. (RO)

21.5.2 ICM Register

The addresses in this section are relative to AXI ICM base address provided in Table 9.3-2 in Chapter 9 *System and Memory*.

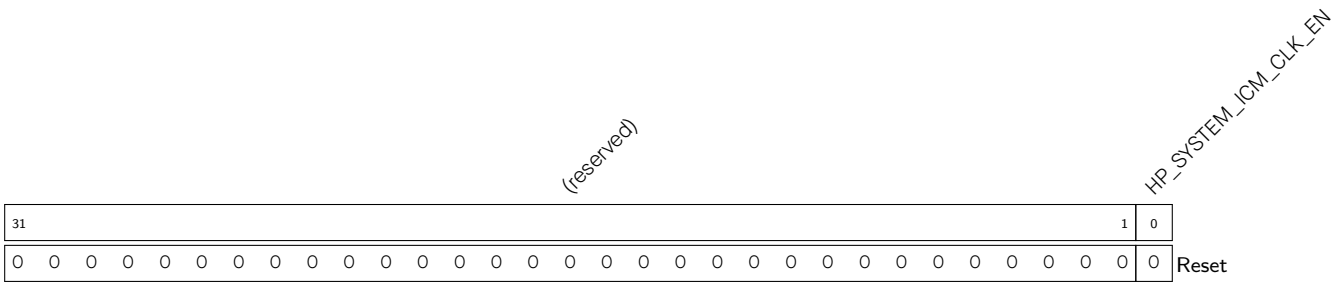
For how to program reserved fields, please refer to Section Programming Reserved Register Field.

Register 21.86. HP_SYSTEM_ICM_VER_DATE_REG (0x0000)



HP_SYSTEM_ICM_VER_DATE Version control register. (R/W)

Register 21.87. HP_SYSTEM_ICM_CLK_EN_REG (0x0004)



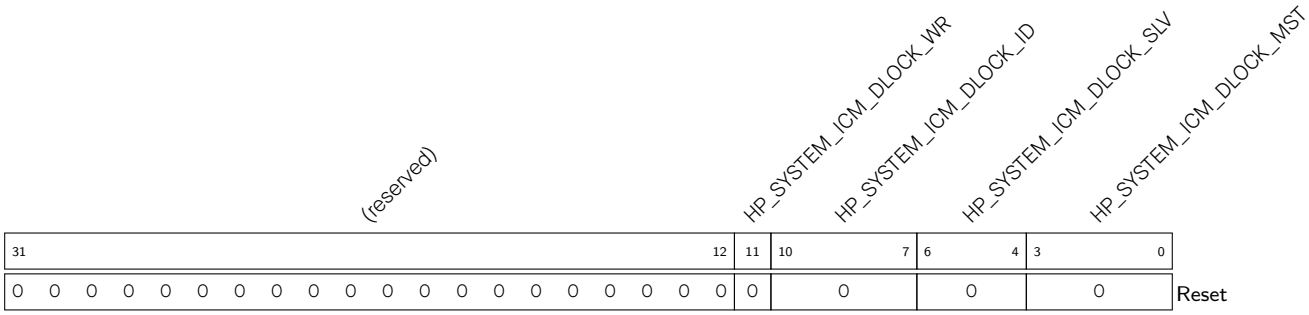
HP_SYSTEM_ICM_CLK_EN Configures whether to force the register clock on.

0: The clock is enabled only during register access

1: Force clock on for the register

(R/W)

Register 21.88. HP_SYSTEM_ICM_DLOCK_STATUS_REG (0x0008)



HP_SYSTEM_ICM_DLOCK_MST Represents the lowest numbered master involved in a deadlock. (RO)

HP_SYSTEM_ICM_DLOCK_SLV Represents the slave with which the deadlock has occurred. (RO)

HP_SYSTEM_ICM_DLOCK_ID Represents the AXI ID of the transaction that is deadlocked. (RO)

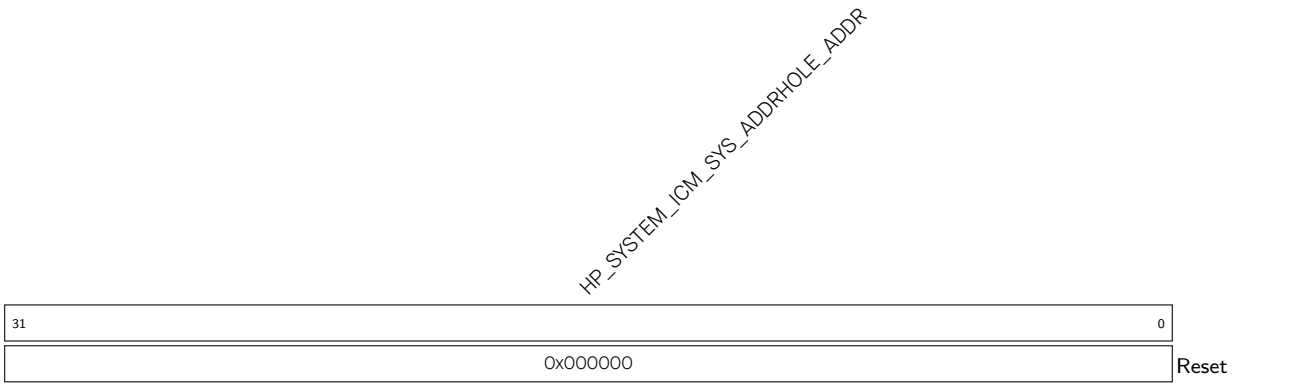
HP_SYSTEM_ICM_DLOCK_WR Represents whether the deadlocked transaction is a write operation.

0: Not a write operation

1: Write operation

(RO)

Register 21.89. HP_SYSTEM_ICM_SYS_ADDRHOLE_ADDR_REG (0x0038)



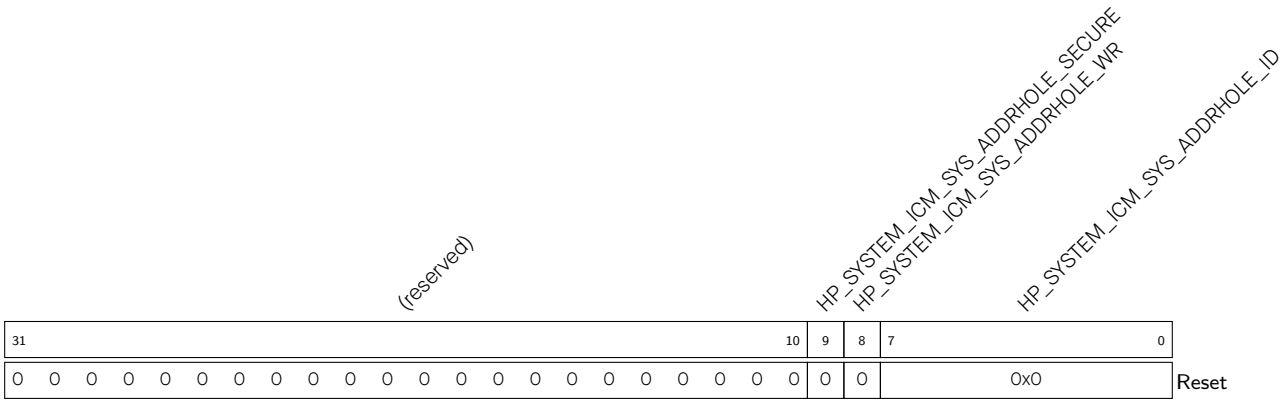
HP_SYSTEM_ICM_SYS_ADDRHOLE_ADDR Represents the address when ICM_SYS_ADDRHOLE_INT occurs.

If HP_SYSTEM_ICM_SYS_ADDRHOLE_SECURE is 0, the address is an unauthorized access address.

If HP_SYSTEM_ICM_SYS_ADDRHOLE_SECURE is 1, the address is an illegal access address.

(RO)

Register 21.90. HP_SYSTEM_ICM_SYS_ADDRHOLE_INFO_REG (0x003C)

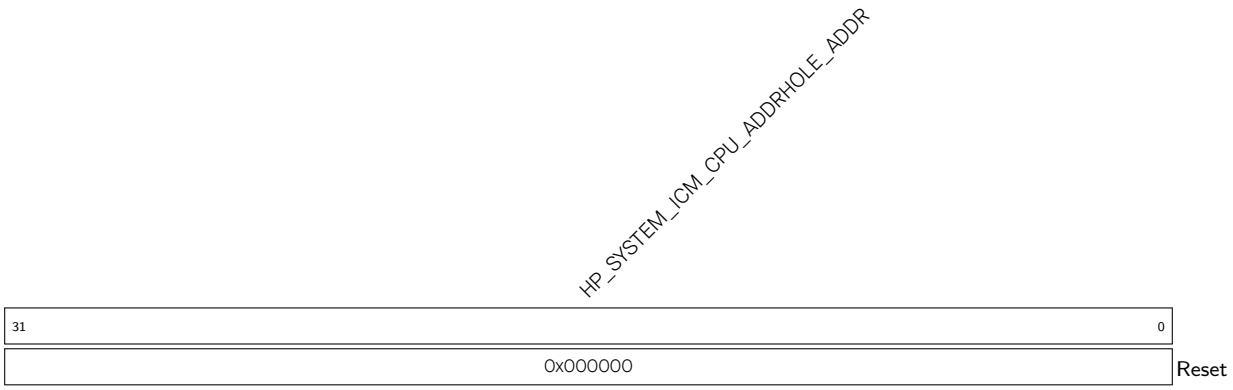


HP_SYSTEM_ICM_SYS_ADDRHOLE_ID Represents the master ID when ICM_SYS_ADDRHOLE_INT occurs. This ID consists of a 4-bit CID and a 4-bit UID (refer to the related IP for details). The CID is used for master verification in ICM. CID values correspond to the following masters:
1: CACHE
5: GDMA MST1
6: GDMA MST2
8: AXI GDMA
10: DMA2D
11: H264 MST1
12: H264 MST2
(RO)

HP_SYSTEM_ICM_SYS_ADDRHOLE_WR Represents the type of transfer when ICM_SYS_ADDRHOLE_INT occurs.
0: Read transfer
1: Write transfer
(RO)

HP_SYSTEM_ICM_SYS_ADDRHOLE_SECURE Represents the type of access error when ICM_SYS_ADDRHOLE_INT occurs.
0: Unauthorized access
1: Illegal access
(RO)

Register 21.91. HP_SYSTEM_ICM_CPU_ADDRHOLE_ADDR_REG (0x0040)

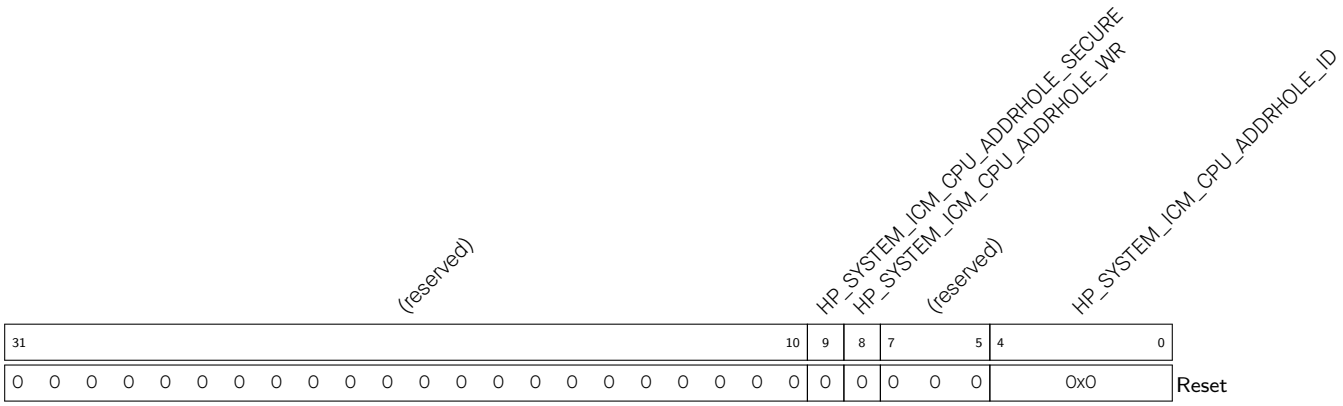


HP_SYSTEM_ICM_CPU_ADDRHOLE_ADDR Represents the address when ICM_CPU_ADDRHOLE_INT occurs.

If HP_SYSTEM_ICM_CPU_ADDRHOLE_SECURE is 0, the address is an unauthorized access address.

If HP_SYSTEM_ICM_CPU_ADDRHOLE_SECURE is 1, the address is an illegal access address. (RO)

Register 21.92. HP_SYSTEM_ICM_CPU_ADDRHOLE_INFO_REG (0x0044)



HP_SYSTEM_ICM_CPU_ADDRHOLE_ID Represents the master ID when ICM_CPU_ADDRHOLE_INT occurs.

0: HP CPU0

1: HP CPU1

2: LP CPU

3: USBOTGFS

5: GMAC

6: SDMMC

7: USBOTGHS

8: TRACE0

9: TRACE1

10: SPM Monitor

11: L2MEM Monitor

16-31: AHB GDMA

(RO)

HP_SYSTEM_ICM_CPU_ADDRHOLE_WR Represents the type of transfer when ICM_CPU_ADDRHOLE_INT occurs.

0: Read transfer

1: Write transfer

(RO)

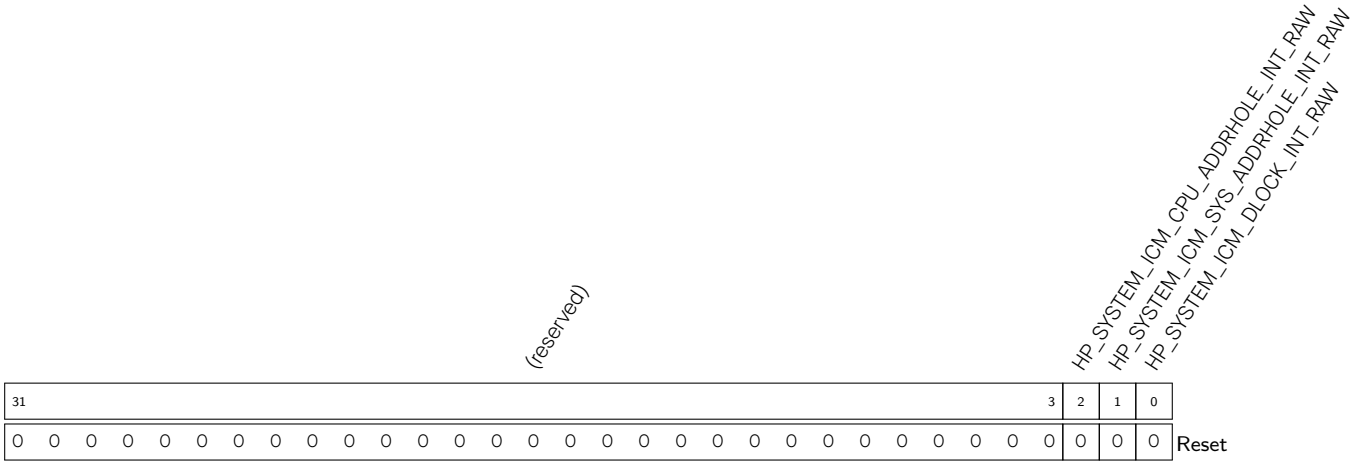
HP_SYSTEM_ICM_CPU_ADDRHOLE_SECURE Represents the type of access error when ICM_CPU_ADDRHOLE_INT occurs.

0: Unauthorized access

1: Illegal access

(RO)

Register 21.93. HP_SYSTEM_ICM_INT_RAW_REG (0x000C)

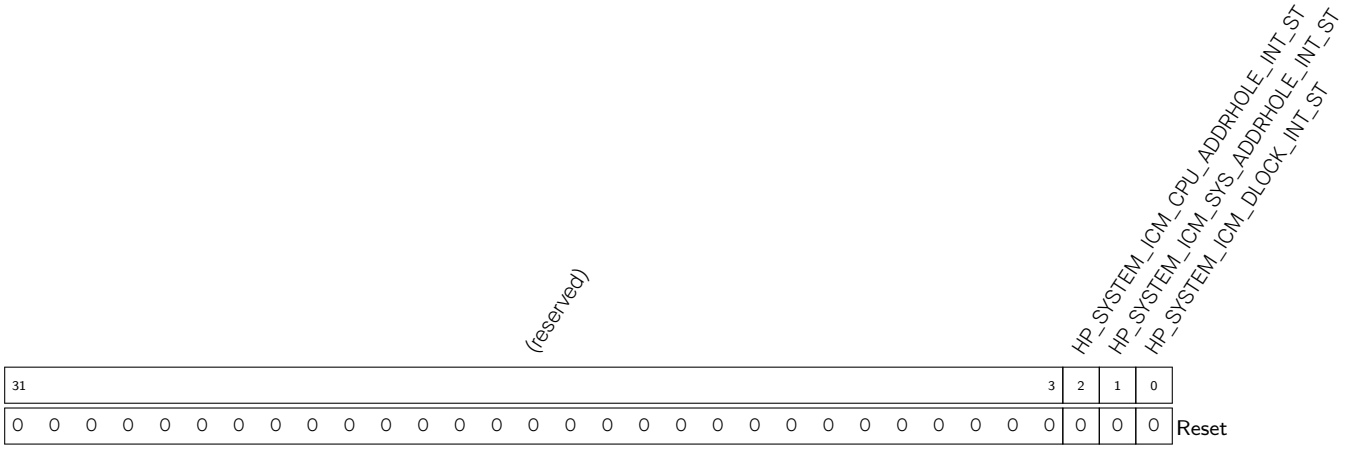


HP_SYSTEM_ICM_DLOCK_INT_RAW The raw interrupt status of ICM_DLOCK_INT. (R/WTC/SS)

HP_SYSTEM_ICM_SYS_ADDRHOLE_INT_RAW The raw interrupt status of ICM_SYS_ADDRHOLE_INT. (R/WTC/SS)

HP_SYSTEM_ICM_CPU_ADDRHOLE_INT_RAW The raw interrupt status of ICM_CPU_ADDRHOLE_INT. (R/WTC/SS)

Register 21.94. HP_SYSTEM_ICM_INT_ST_REG (0x0010)

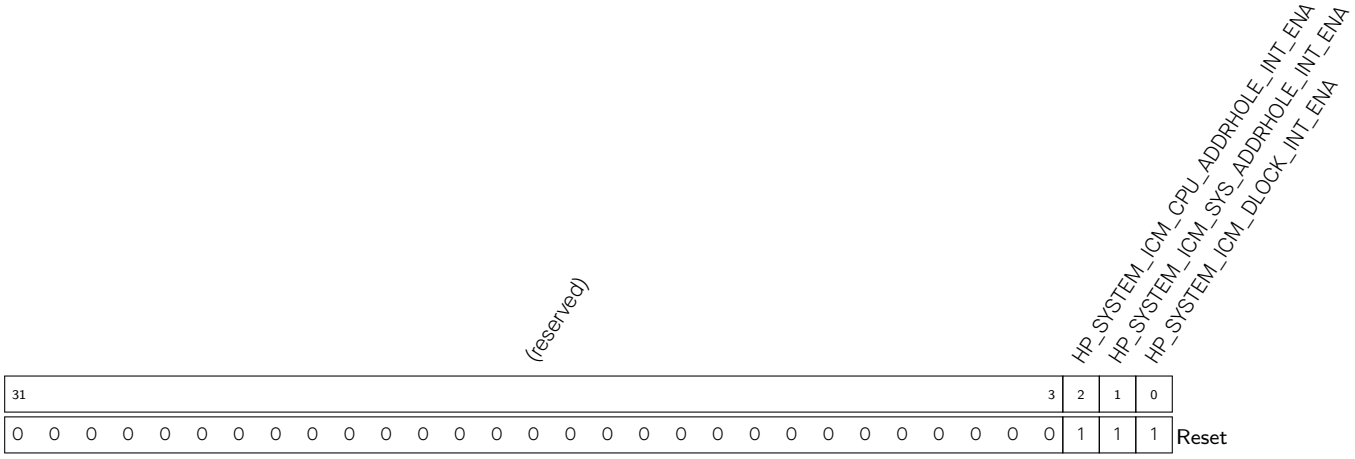


HP_SYSTEM_ICM_DLOCK_INT_ST The masked interrupt status of DLOCK_INT. (RO)

HP_SYSTEM_ICM_SYS_ADDRHOLE_INT_ST The masked interrupt status of ICM_SYS_ADDRHOLE_INT. (RO)

HP_SYSTEM_ICM_CPU_ADDRHOLE_INT_ST The masked interrupt status of ICM_CPU_ADDRHOLE_INT. (RO)

Register 21.95. HP_SYSTEM_ICM_INT_ENA_REG (0x0014)

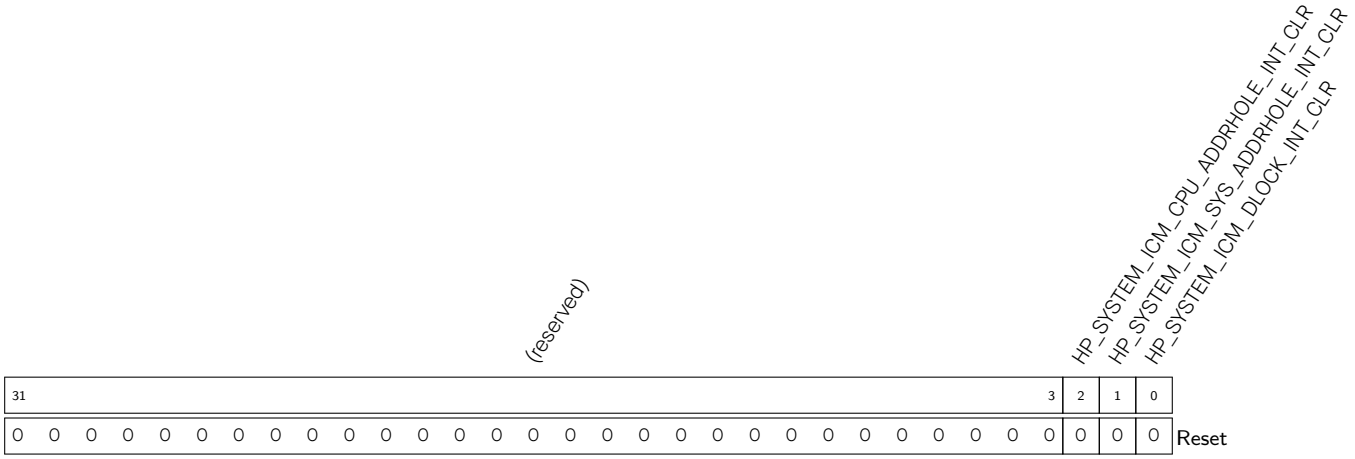


HP_SYSTEM_ICM_DLOCK_INT_ENA Write 1 to enable DLOCK_INT. (R/W)

HP_SYSTEM_ICM_SYS_ADDRHOLE_INT_ENA Write 1 to enable ICM_SYS_ADDRHOLE_INT. (R/W)

HP_SYSTEM_ICM_CPU_ADDRHOLE_INT_ENA Write 1 to enable ICM_CPU_ADDRHOLE_INT. (R/W)

Register 21.96. HP_SYSTEM_ICM_INT_CLR_REG (0x0018)

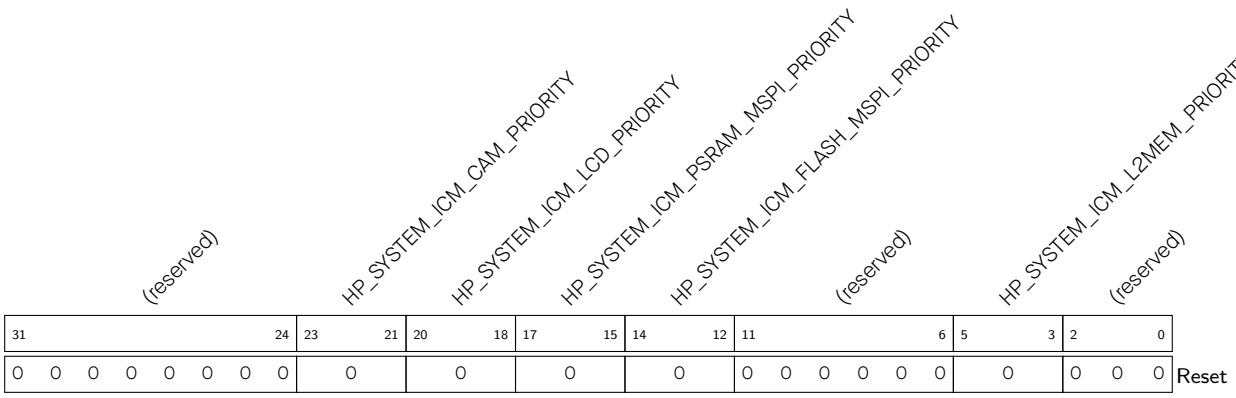


HP_SYSTEM_ICM_DLOCK_INT_CLR Write 1 to clear DLOCK_INT. (WT)

HP_SYSTEM_ICM_SYS_ADDRHOLE_INT_CLR Write 1 to clear ICM_SYS_ADDRHOLE_INT. (WT)

HP_SYSTEM_ICM_CPU_ADDRHOLE_INT_CLR Write 1 to clear ICM_CPU_ADDRHOLE_INT. (WT)

Register 21.97. HP_SYSTEM_ICM_SLV_ARB_PRIORITY_REG (0x0024)



HP_SYSTEM_ICM_L2MEM_PRIORITY Configures the arbitration priority of L2MEM for response channels among slaves connected to SYS ICM. (R/W)

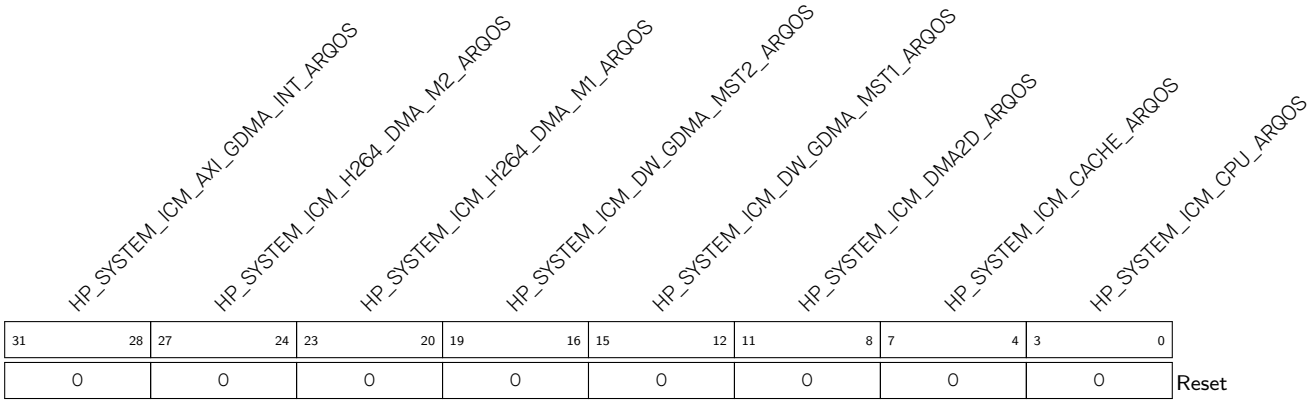
HP_SYSTEM_ICM_FLASH_MSPI_PRIORITY Configures the arbitration priority of FLASH MSPI for response channels among slaves connected to SYS ICM. (R/W)

HP_SYSTEM_ICM_PSRAM_MSPI_PRIORITY Configures the arbitration priority of PSRAM MSPI for response channels among slaves connected to SYS ICM. (R/W)

HP_SYSTEM_ICM_LCD_PRIORITY Configures the arbitration priority of MIPI_LCD registers for response channels among slaves connected to SYS ICM. (R/W)

HP_SYSTEM_ICM_CAM_PRIORITY Configures the arbitration priority of MIPI_CAM registers for response channels among slaves connected to SYS ICM. (R/W)

Register 21.98. HP_SYSTEM_ICM_MST_ARQOS_REG0_REG (0x0028)



HP_SYSTEM_ICM_CPU_ARQOS Configures the ARQoS value for CPU ICM. (R/W)

HP_SYSTEM_ICM_CACHE_ARQOS Configures the ARQoS value for CACHE. (R/W)

HP_SYSTEM_ICM_DMA2D_ARQOS Configures the ARQoS value for DMA2D. (R/W)

HP_SYSTEM_ICM_DW_GDMA_MST1_ARQOS Configures the ARQoS value for DW-GDMA MST1. (R/W)

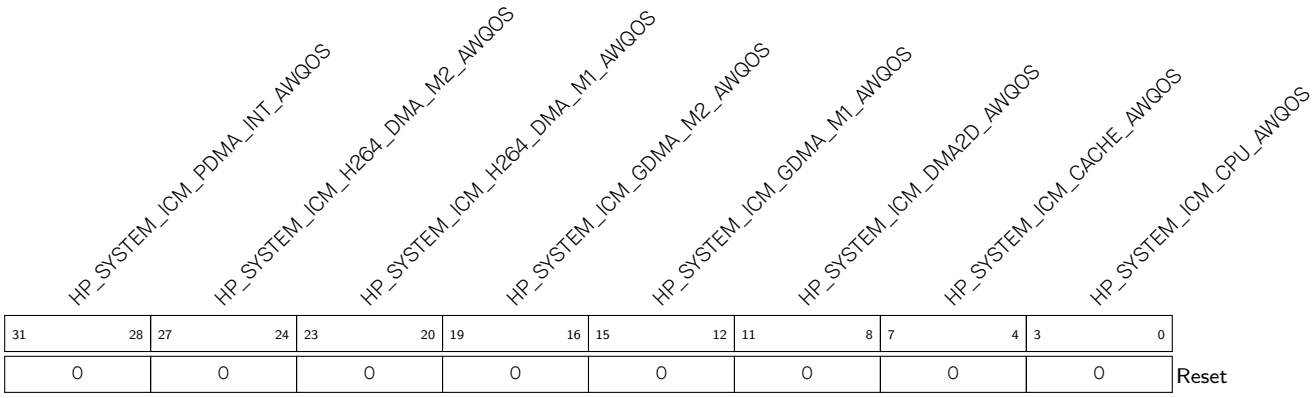
HP_SYSTEM_ICM_DW_GDMA_MST2_ARQOS Configures the ARQoS value for DW-GDMA MST2. (R/W)

HP_SYSTEM_ICM_H264_DMA_M1_ARQOS Configures the ARQoS value for H264 DMAMST1. (R/W)

HP_SYSTEM_ICM_H264_DMA_M2_ARQOS Configures the ARQoS value for H264 DMAMST2. (R/W)

HP_SYSTEM_ICM_AXI_GDMA_INT_ARQOS Configures the ARQoS value for AXI GDMA. (R/W)

Register 21.99. HP_SYSTEM_ICM_MST_AWQOS_REG0_REG (0x0030)



HP_SYSTEM_ICM_CPU_AWQOS Configures the AWQoS value for the CPU ICM. (R/W)

HP_SYSTEM_ICM_CACHE_AWQOS Configures the AWQoS value for the CACHE. (R/W)

HP_SYSTEM_ICM_DMA2D_AWQOS Configures the AWQoS value for the DMA2D. (R/W)

HP_SYSTEM_ICM_GDMA_M1_AWQOS Configures the AWQoS value for the DW-GDMA MST1. (R/W)

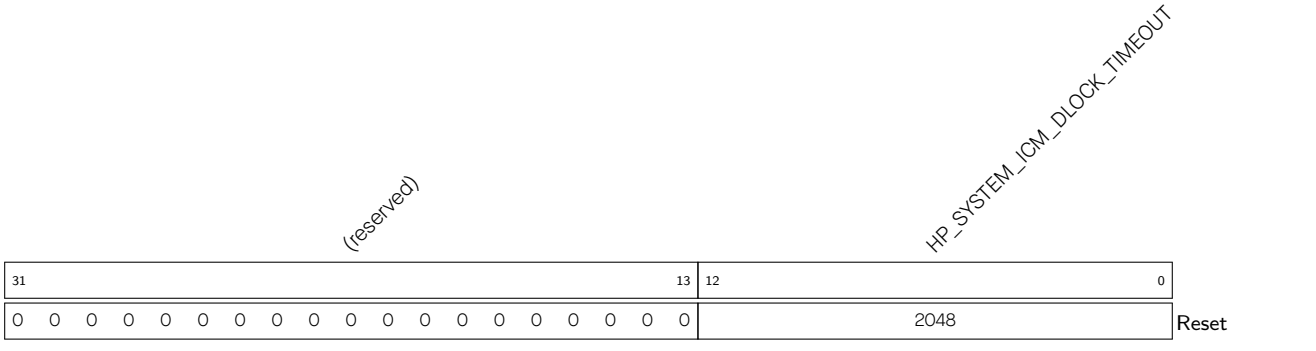
HP_SYSTEM_ICM_GDMA_M2_AWQOS Configures the AWQoS value for the DW-GDMA MST2. (R/W)

HP_SYSTEM_ICM_H264_DMA_M1_AWQOS Configures the AWQoS value for the H264 DMA MST1. (R/W)

HP_SYSTEM_ICM_H264_DMA_M2_AWQOS Configures the AWQoS value for the H264 DMA MST2. (R/W)

HP_SYSTEM_ICM_PDMA_INT_AWQOS Configures the AWQoS value for the AXI GDMA. (R/W)

Register 21.100. HP_SYSTEM_ICM_DLOCK_TIMEOUT_REG (0x0048)



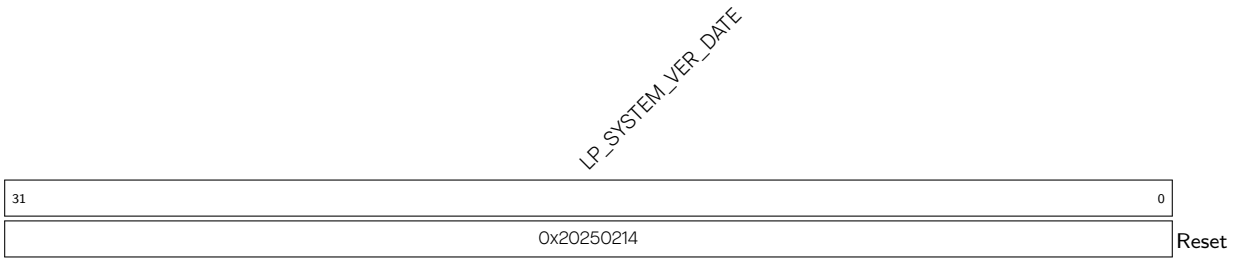
HP_SYSTEM_ICM_DLOCK_TIMEOUT Configures the deadlock timeout threshold value. Deadlock will occur if there is no response within the configured threshold.
(R/W)

21.5.3 LP Registers

The addresses in this section are relative to LP System Registers base address provided in Table 9.3-2 in Chapter 9 *System and Memory*.

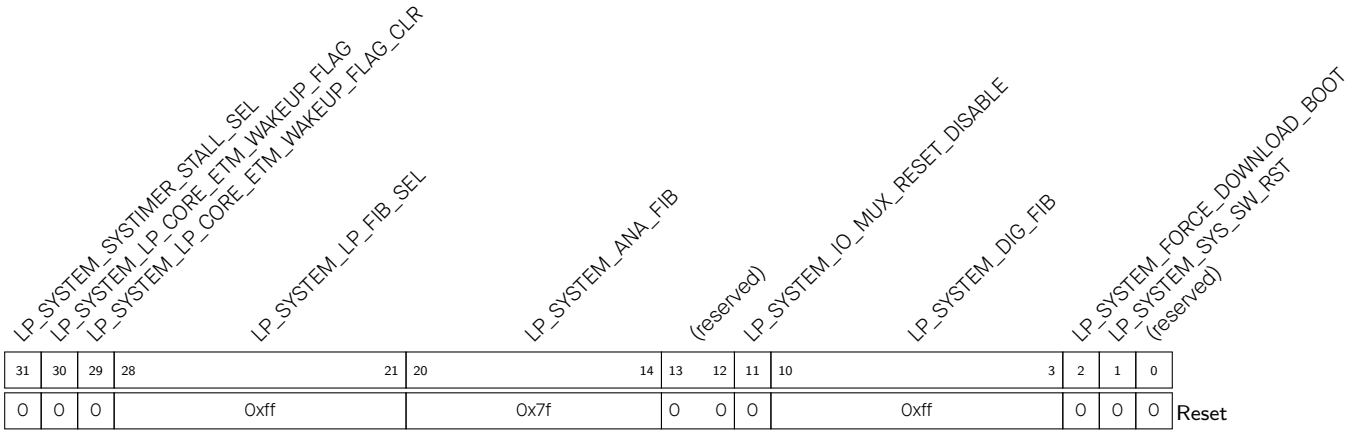
For how to program reserved fields, please refer to Section Programming Reserved Register Field.

Register 21.101. LP_SYSTEM_LP_SYS_VER_DATE_REG (0x0000)



LP_SYSTEM_VER_DATE Version control register. (R/W)

Register 21.102. LP_SYSTEM_SYS_CTRL_REG (0x0008)



LP_SYSTEM_SYS_SW_RST Configures whether to trigger a software system reset. (WT)

LP_SYSTEM_FORCE_DOWNLOAD_BOOT Configures whether to forcibly enter Joint Download Boot mode when the pad boot mode is set to SPI Boot mode.

0: Not forcibly enter in Joint Download Boot mode

1: Forcibly enter in Joint Download Boot mode

(R/W)

LP_SYSTEM_DIG_FIB Configures software FIB.

Bit[0]: Unused

Bit[1]: Software BOD enable

Bit[2]: Software SWD enable

Bit[3]: Unused

Bit[4]: Software DCDC switch

Bit[5]: Software DCDC VSET enable

Bit[6]: Unused

Bit[7]: Unused

(R/W)

LP_SYSTEM_IO_MUX_RESET_DISABLE Configures whether to disable the HP IOMUX reset.

0: Enable IOMUX reset

1: Disable IOMUX reset

(R/W)

Continued on the next page...

Register 21.102. LP_SYSTEM_SYS_CTRL_REG (0x0008)

Continued from the previous page...

LP_SYSTEM_ANA_FIB Represents analog FIB state.
Bit[0]: Unused
Bit[1]: Analog BOD enable
Bit[2]: Analog SWD enable
Bit[3]: Unused
Bit[4]: Unused
Bit[5]: Unused
Bit[6]: Unused
Bit[7]: Unused
(RO)

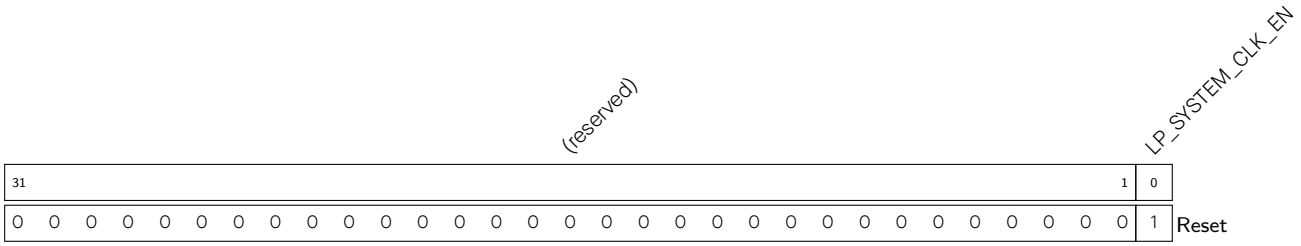
LP_SYSTEM_LP_FIB_SEL Configures FIB control source select.
Bit[0]: Unused
Bit[1]: Configures BOD enable control select
0: Controlled by software
1: Controlled by analog
Bit[2]: Configures SWD enable control select
0: Controlled by software
1: Controlled by analog
Bit[3]: Unused
Bit[4]: Unused
Bit[5]: Unused
Bit[6]: Unused
Bit[7]: Unused
(R/W)

LP_SYSTEM_LP_CORE_ETM_WAKEUP_FLAG_CLR Write 1 to clear LP_SYSTEM_LP_CORE_ETM_WAKEUP_FLAG. (WT)

LP_SYSTEM_LP_CORE_ETM_WAKEUP_FLAG Represents ETM task of ULP wakeup. (R/WTC/SS)

LP_SYSTEM_SYSTIMER_STALL_SEL Configures whether the LP systimer_stall signal is from HP CPU0 or HP CPU1.
0: From HP CPU0
1: From HP CPU1
(R/W)

Register 21.103. LP_SYSTEM_LP_CLK_CTRL_REG (0x000C)



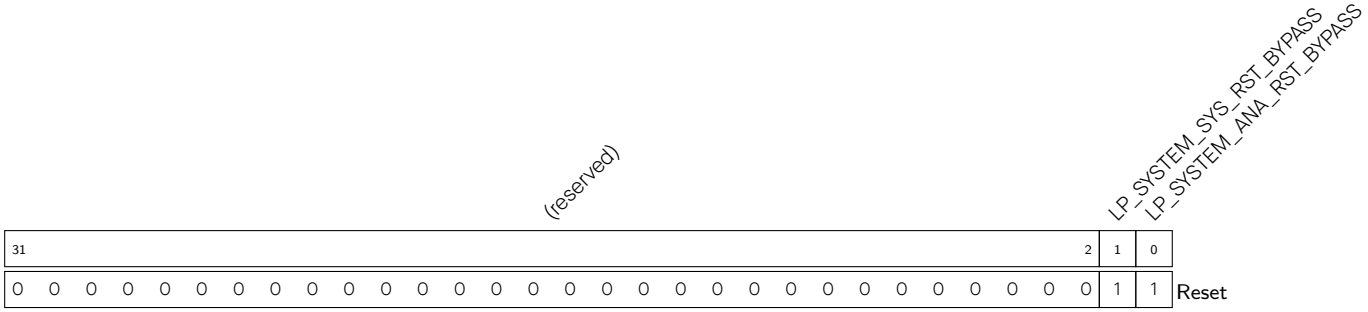
LP_SYSTEM_CLK_EN Configures whether the register clock is always on.

0: The clock is enabled only during register access

1: The clock is always on

(R/W)

Register 21.104. LP_SYSTEM_LP_RST_CTRL_REG (0x0010)



LP_SYSTEM_ANA_RST_BYPASS Configures whether to bypass these analog reset sources: brown-out detector, super WDT and power glitch detector.

0: Not bypass

1: Bypass

(R/W)

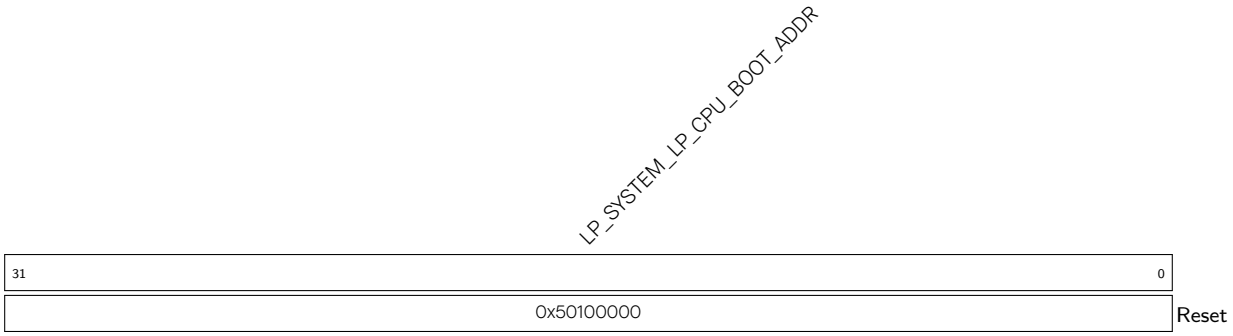
LP_SYSTEM_SYS_RST_BYPASS Configures whether to bypass these digital reset sources: software, HP watchdog timer, LP watchdog timer and eFuse.

0: Not bypass

1: Bypass

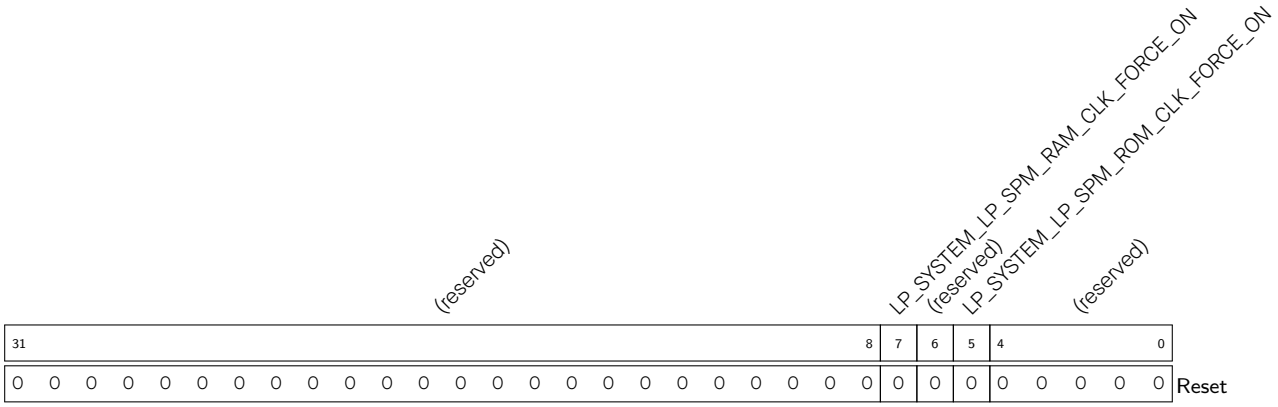
(R/W)

Register 21.105. LP_SYSTEM_LP_CORE_BOOT_ADDR_REG (0x0018)



LP_SYSTEM_LP_CPU_BOOT_ADDR Configures the LP CPU boot address. (R/W)

Register 21.106. LP_SYSTEM_LP_SPM_PWR_CTRL_REG (0x0024)



LP_SYSTEM_LP_SPM_ROM_CLK_FORCE_ON Configures whether to force on LP SPM_ROM clock when LP CPU clock is on.

0: Not force on

1: Force on

(R/W)

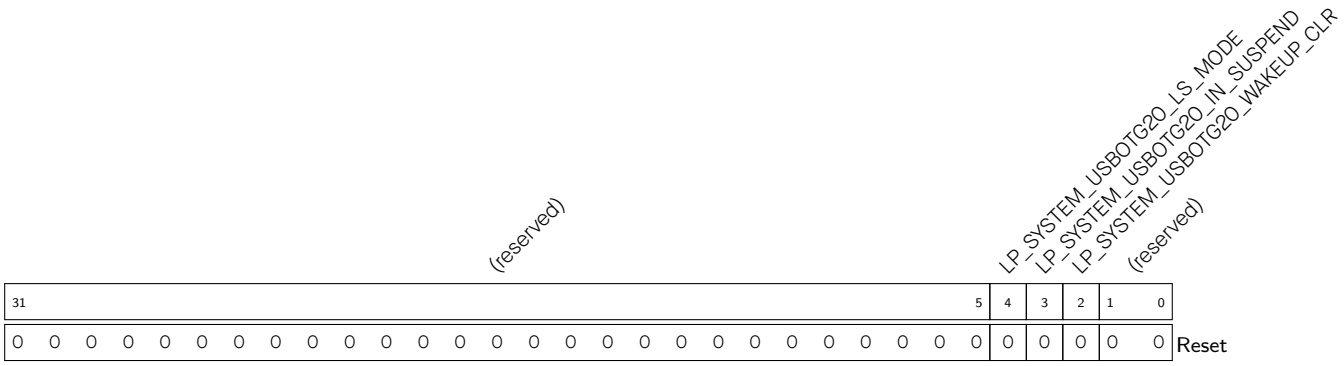
LP_SYSTEM_LP_SPM_RAM_CLK_FORCE_ON Configures whether to force on LP SPM_RAM clock.

0: Not force on

1: Force on

(R/W)

Register 21.107. LP_SYSTEM_USB_CTRL_REG (0x0100)



LP_SYSTEM_USBOTG20_WAKEUP_CLR Write 1 to clear wakeup signals from USB OTG2.0 to PMU. (WT)

LP_SYSTEM_USBOTG20_IN_SUSPEND Configures whether to enable wakeup signals from USB OTG2.0.

0: Disable

1: Enable

(R/W)

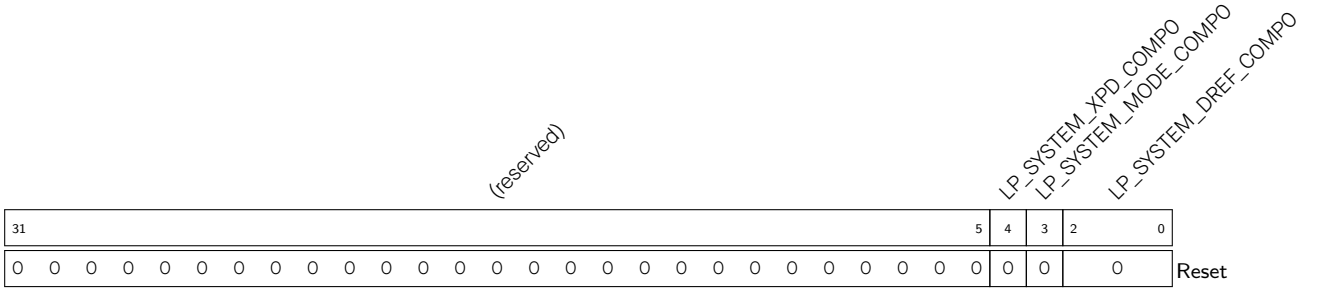
LP_SYSTEM_USBOTG20_LS_MODE Configures whether USB OTG2.0 operates in low-speed mode.

0: Not low-speed mode

1: Low-speed mode

(R/W)

Register 21.108. LP_SYSTEM_PAD_COMP0_REG (0x0148)



LP_SYSTEM_DREF_COMP0 Configures the internal reference voltage of pad comparator0. (R/W)

LP_SYSTEM_MODE_COMP0 Configures the comparison mode for pad comparator0.

0: Comparing the main voltage with the external reference voltage

1: Comparing the main voltage with the internal reference voltage

(R/W)

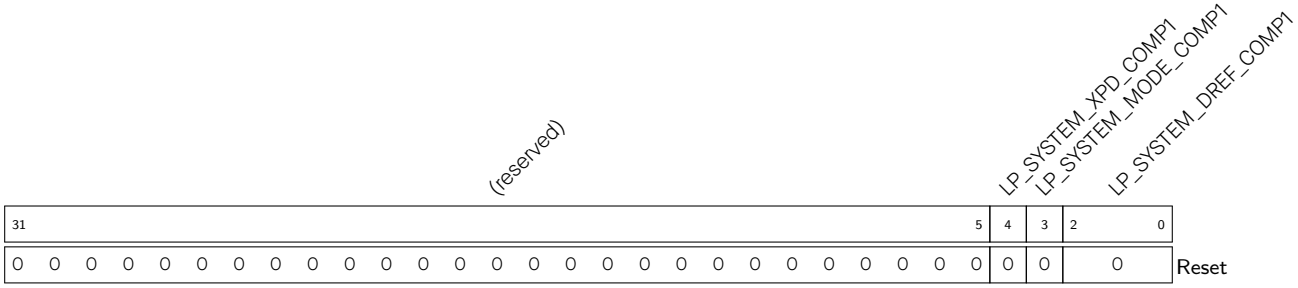
LP_SYSTEM_XPD_COMP0 Configures whether to enable pad comparator0.

0: Disable

1: Enable

(R/W)

Register 21.109. LP_SYSTEM_PAD_COMP1_REG (0x014C)



LP_SYSTEM_DREF_COMP1 Configures the internal reference voltage of pad comparator1. (R/W)

LP_SYSTEM_MODE_COMP1 Configures the comparison mode for pad comparator1.

0: Comparing the main voltage with the external reference voltage

1: Comparing the main voltage with the internal reference voltage

(R/W)

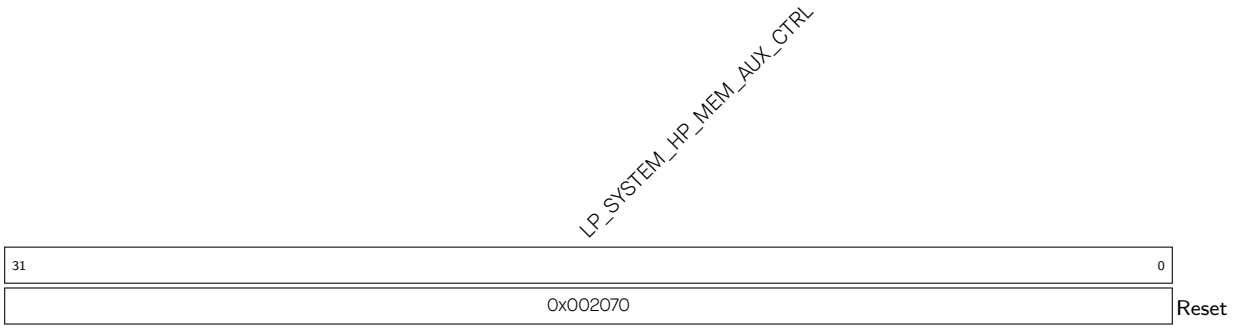
LP_SYSTEM_XPD_COMP1 Configures whether to enable pad comparator1.

0: Disable

1: Enable

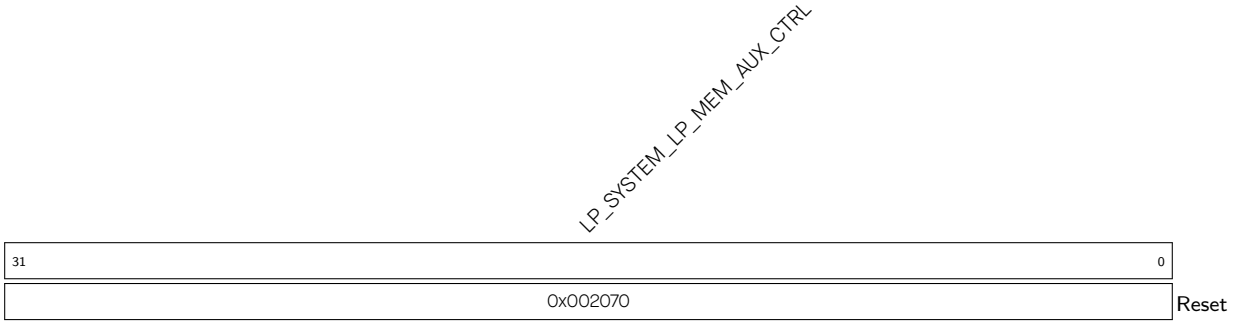
(R/W)

Register 21.110. LP_SYSTEM_HP_MEM_AUX_CTRL_REG (0x0180)



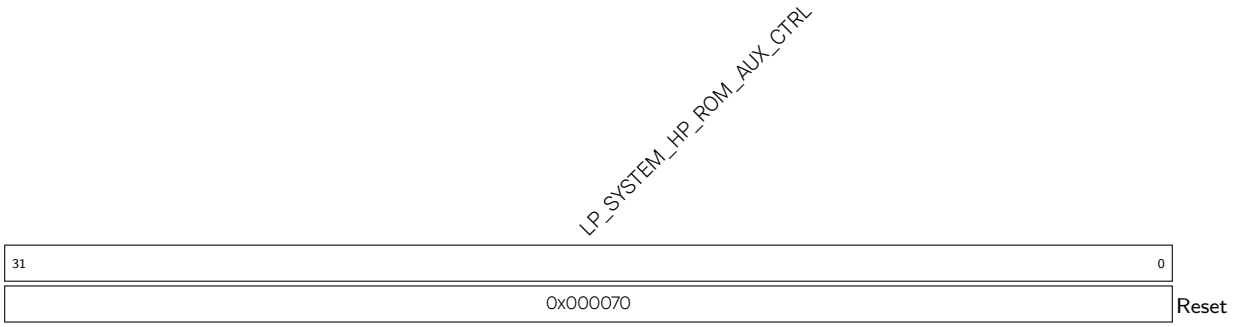
LP_SYSTEM_HP_MEM_AUX_CTRL Configures HP system memory aux control value. (R/W)

Register 21.111. LP_SYSTEM_LP_MEM_AUX_CTRL_REG (0x0184)



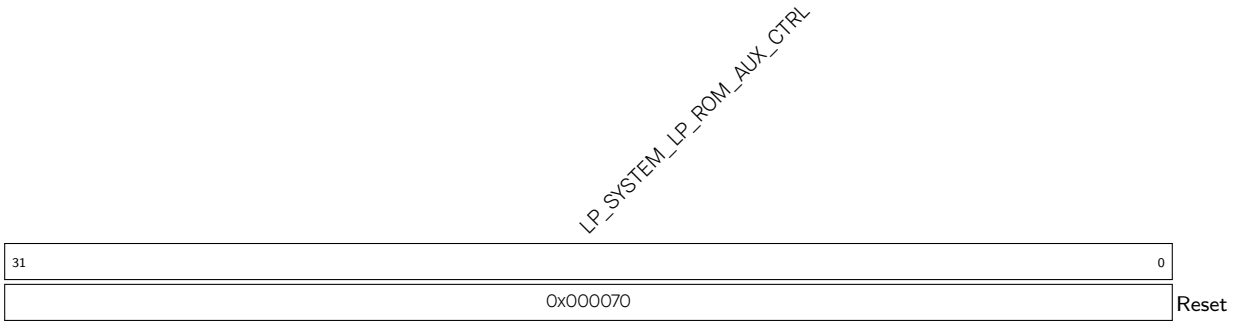
LP_SYSTEM_LP_MEM_AUX_CTRL Configures LP system memory aux control value. (R/W)

Register 21.112. LP_SYSTEM_HP_ROM_AUX_CTRL_REG (0x0188)



LP_SYSTEM_HP_ROM_AUX_CTRL Configures L2ROM aux control value. (R/W)

Register 21.113. LP_SYSTEM_LP_ROM_AUX_CTRL_REG (0x018C)



LP_SYSTEM_LP_ROM_AUX_CTRL Configures LPROM aux control value. (R/W)

Register 21.114. LP_SYSTEM_HP_POR_RST_BYPASS_CTRL_REG (0x01A0)



LP_SYSTEM_HP_PO_CNNT_RSTN_BYPASS_CTRL Configures whether to bypass reset sources for the HP_CNNT power domain during the Power-On reset.

Bit[0]: Configures whether to bypass eFuse reset

0: No effect

1: Bypass

Bit[1]: Configures whether to bypass LP watchdog timer reset

0: No effect

1: Bypass

Bit[2]: Configures whether to bypass USB (JTAG) reset

0: No effect

1: Bypass

Bit[3]: Configures whether to bypass USB (UART) reset

0: No effect

1: Bypass

Bit[4]: Unused

Bit[5]: Unused

Bit[6]: Configures whether to bypass HP watchdog timer reset

0: No effect

1: Bypass

Bit[7]: Configures whether to bypass software reset

0: No effect

1: Bypass

(R/W)

Continued on the next page...

Register 21.114. LP_SYSTEM_HP_POR_RST_BYPASS_CTRL_REG (0x01A0)

Continued from the previous page...

LP_SYSTEM_HP_PO_RSTN_BYPASS_CTRL Configures whether to bypass reset sources for the HP power domain during the Power-On reset.

Bit[0]: Configures whether to bypass eFuse reset

0: No effect

1: Bypass

Bit[1]: Configures whether to bypass LP watchdog timer reset

0: No effect

1: Bypass

Bit[2]: Configures whether to bypass USB (JTAG) reset

0: No effect

1: Bypass

Bit[3]: Configures whether to bypass USB (UART) reset

0: No effect

1: Bypass

Bit[4]: Unused

Bit[5]: Unused

Bit[6]: Configures whether to bypass HP watchdog timer reset

0: No effect

1: Bypass

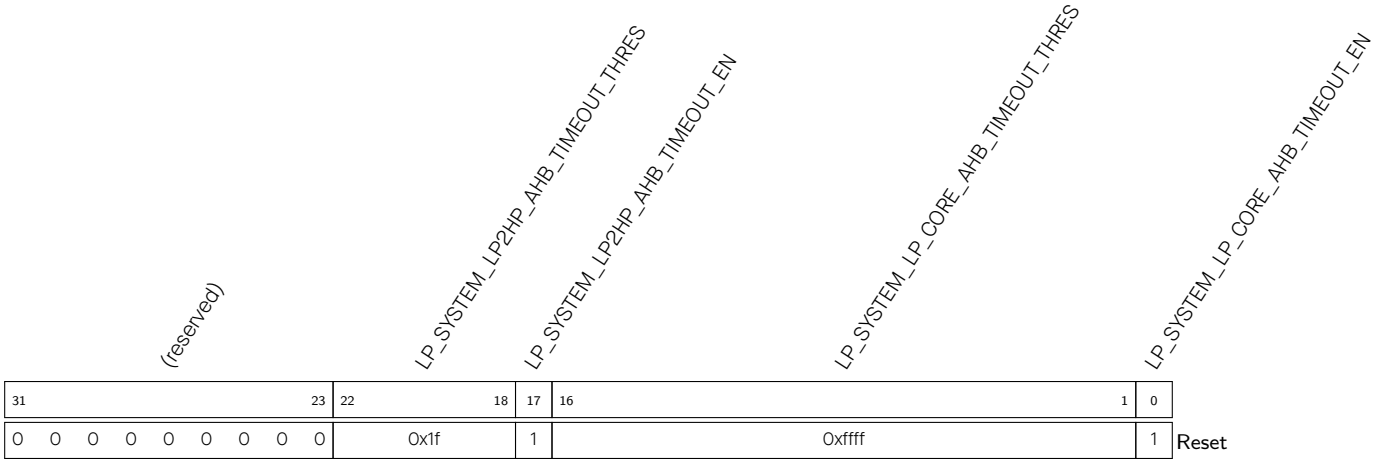
Bit[7]: Configures whether to bypass software reset

0: No effect

1: Bypass

(R/W)

Register 21.115. LP_SYSTEM_LP_CORE_AHB_TIMEOUT_REG (0x01B0)



LP_SYSTEM_LP_CORE_AHB_TIMEOUT_EN Configures whether to enable timeout protection on LP AHB bus.

0: Disable

1: Enable

(R/W)

LP_SYSTEM_LP_CORE_AHB_TIMEOUT_THRES Configures LP_CORE_AHB_TIMEOUT_INT threshold. (R/W)

LP_SYSTEM_LP2HP_AHB_TIMEOUT_EN Configures whether to enable timeout protection on LP2HP AHB bus.

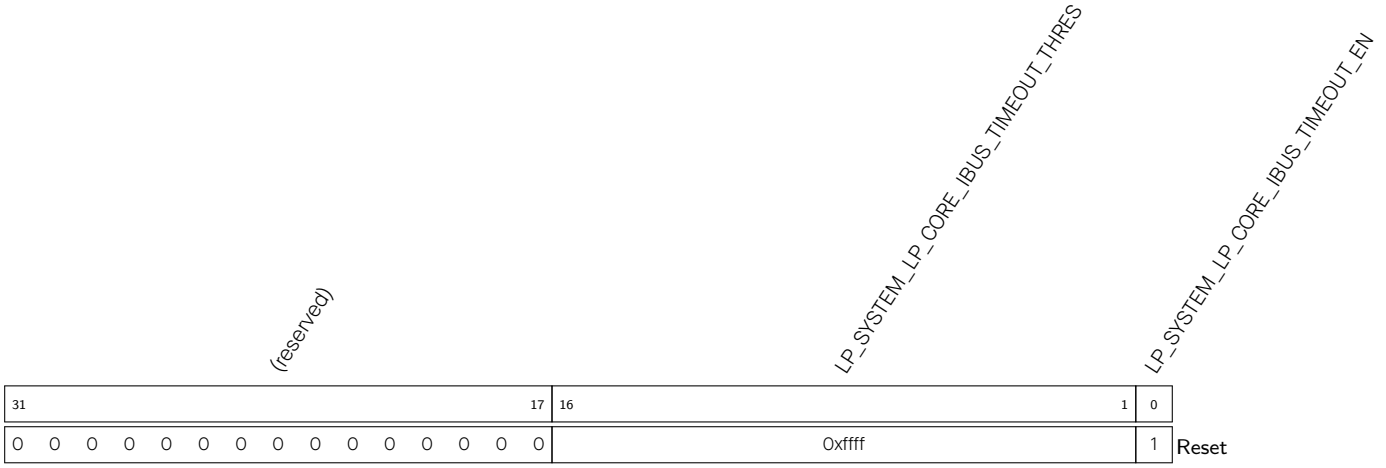
0: Disable

1: Enable

(R/W)

LP_SYSTEM_LP2HP_AHB_TIMEOUT_THRES Configures LP_CORE_AHB_TIMEOUT_INT threshold. (R/W)

Register 21.116. LP_SYSTEM_LP_CORE_IBUS_TIMEOUT_REG (0x01B4)



LP_SYSTEM_LP_CORE_IBUS_TIMEOUT_EN Configures whether to enable timeout protection on LP CPU IBUS.

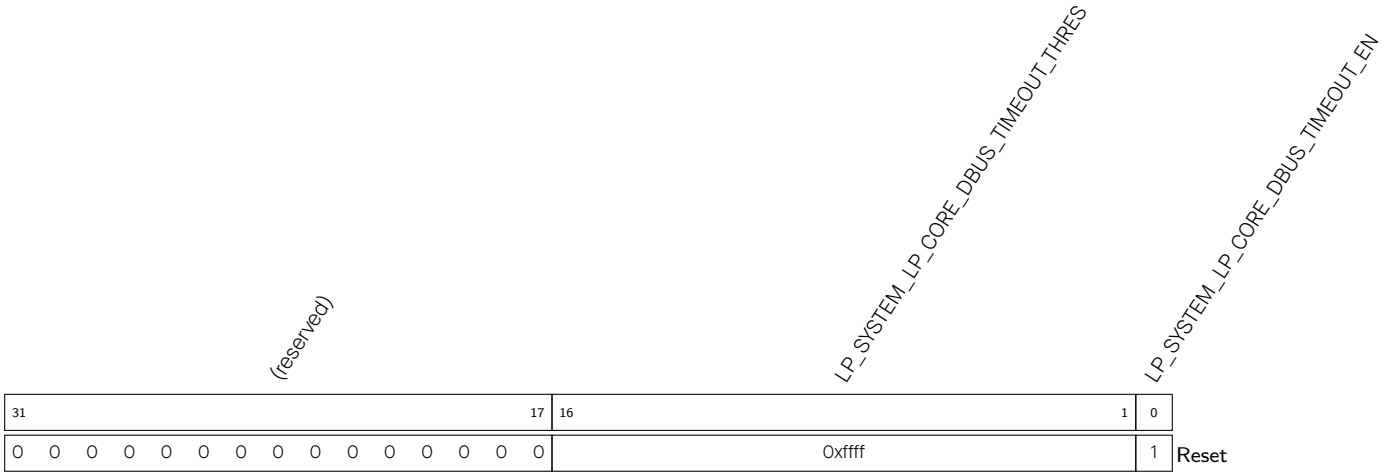
0: Disable

1: Enable

(R/W)

LP_SYSTEM_LP_CORE_IBUS_TIMEOUT_THRES Configures LP_CORE_IBUS_TIMEOUT_INT threshold. (R/W)

Register 21.117. LP_SYSTEM_LP_CORE_DBUS_TIMEOUT_REG (0x01B8)



LP_SYSTEM_LP_CORE_DBUS_TIMEOUT_EN Configures whether to enable timeout protection on LP CPU DBUS.

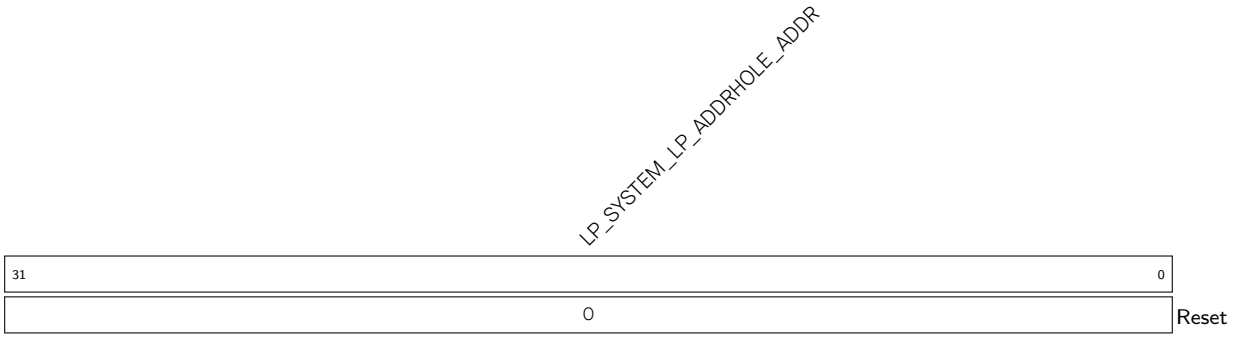
0: Disable

1: Enable

(R/W)

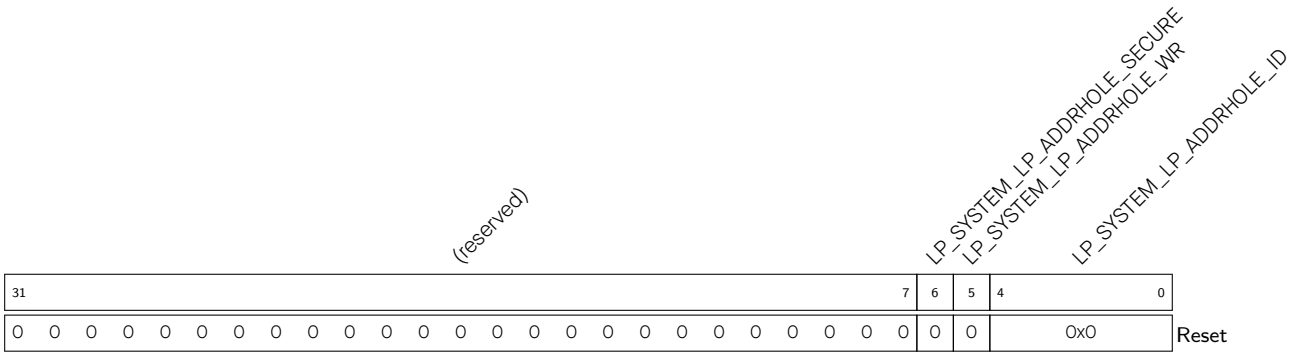
LP_SYSTEM_LP_CORE_DBUS_TIMEOUT_THRES Configures LP_CORE_DBUS_TIMEOUT_INT threshold. (R/W)

Register 21.118. LP_SYSTEM_LP_ADDRHOLE_ADDR_REG (0x0168)



LP_SYSTEM_LP_ADDRHOLE_ADDR Represents the address of transfer when LP_ADDRHOLE_INT occurs. (RO)

Register 21.119. LP_SYSTEM_LP_ADDRHOLE_INFO_REG (0x016C)



LP_SYSTEM_LP_ADDRHOLE_ID Represents the master ID when LP_ADDRHOLE_INT occurs.

0: LP CPU0
1: LP CPU1
2: LP CPU
3: USB OTG11
4: Reserved
5: GMAC
6: SDMMC
7: USBOTG20
8: TRACE0
9: TRACE1
10: SPM monitor
11: L2MEM monitor
16~31: AHB PDMA
(RO)

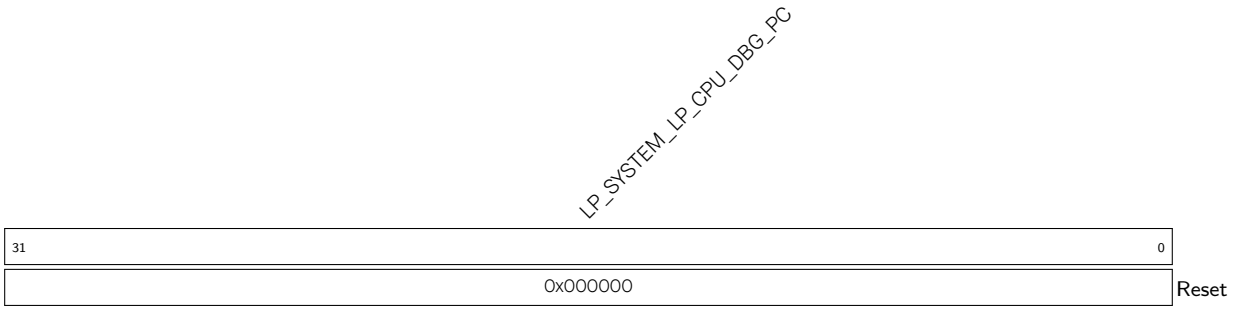
LP_SYSTEM_LP_ADDRHOLE_WR Represents the direction of transfer when LP_ADDRHOLE_INT occurs.

0: Read transfer
1: Write transfer
(RO)

LP_SYSTEM_LP_ADDRHOLE_SECURE Represents the address hole type when LP_ADDRHOLE_INT occurs.

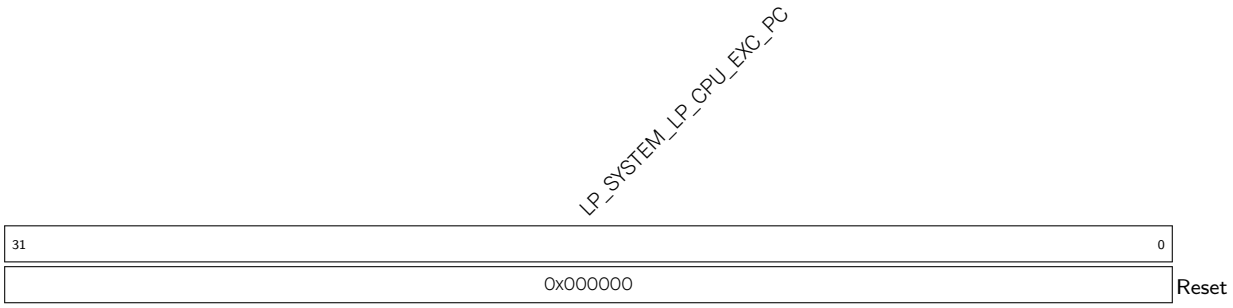
0: Access without permission
1: Illegal address access
(RO)

Register 21.120. LP_SYSTEM_LP_CPU_DBG_PC_REG (0x0190)



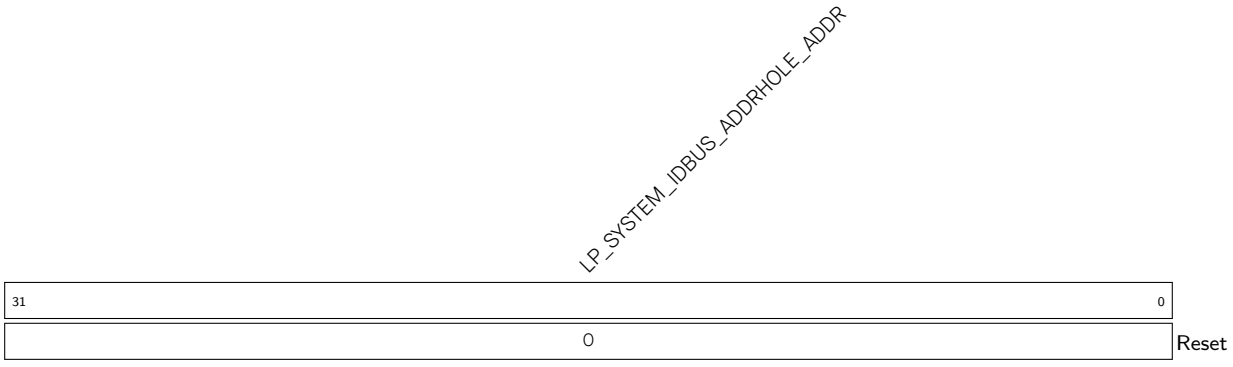
LP_SYSTEM_LP_CPU_DBG_PC Represents LP CPU current PC address. (RO)

Register 21.121. LP_SYSTEM_LP_CPU_EXC_PC_REG (0x0194)



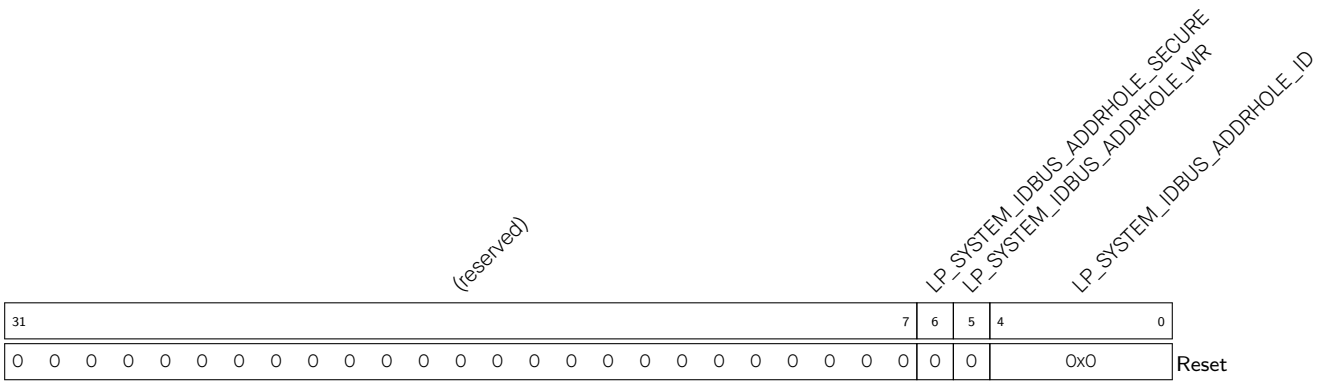
LP_SYSTEM_LP_CPU_EXC_PC Represents LP CPU exception PC address. (RO)

Register 21.122. LP_SYSTEM_IDBUS_ADDRHOLE_ADDR_REG (0x0198)



LP_SYSTEM_IDBUS_ADDRHOLE_ADDR Represents the address of transfer when IDBUS_ADDRHOLE_INT occurs. (RO)

Register 21.123. LP_SYSTEM_IDBUS_ADDRHOLE_INFO_REG (0x019C)



LP_SYSTEM_IDBUS_ADDRHOLE_ID Represents the LP CPU's ID value (00010) when IDBUS_ADDRHOLE_INT occurs. (RO)

LP_SYSTEM_IDBUS_ADDRHOLE_WR Represents the direction of transfer when IDBUS_ADDRHOLE_INT occurs.

0: Read transfer

1: Write transfer

(RO)

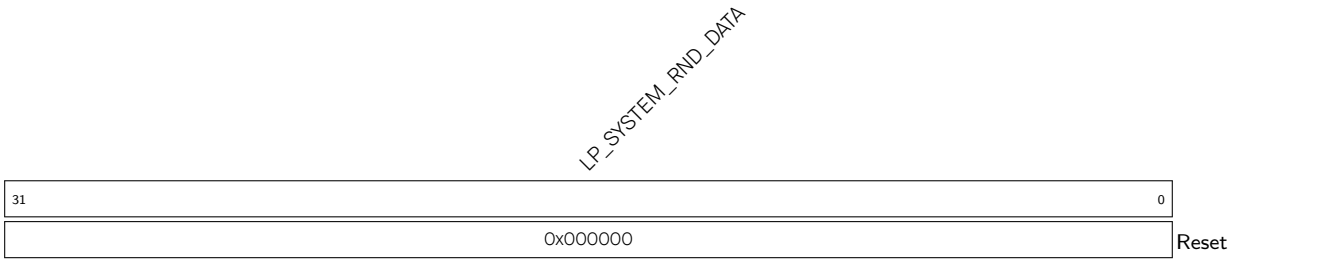
LP_SYSTEM_IDBUS_ADDRHOLE_SECURE Represents the address hole type when IDBUS_ADDRHOLE_INT occurs.

0: Access without permission

1: Illegal address access

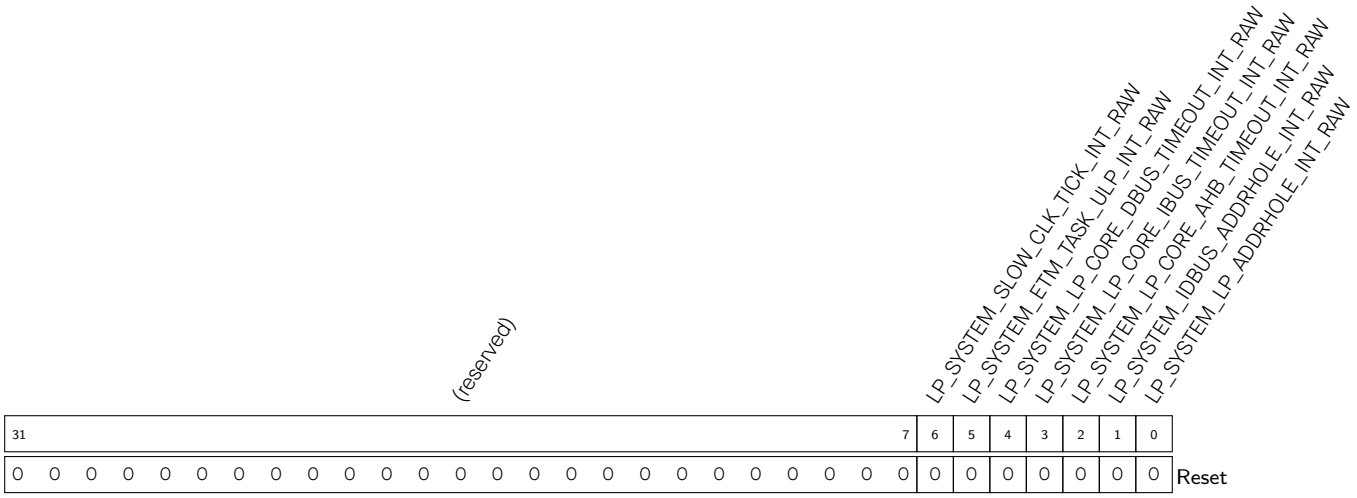
(RO)

Register 21.124. LP_SYSTEM_RNG_DATA_REG (0x01A4)



LP_SYSTEM_RND_DATA Represents the RNG-generated random numbers. (RO)

Register 21.125. LP_SYSTEM_INT_RAW_REG (0x0170)



LP_SYSTEM_LP_ADDRHOLE_INT_RAW The raw interrupt status of LP_ADDRHOLE_INT. (R/SS/WTC)

LP_SYSTEM_IDBUS_ADDRHOLE_INT_RAW The raw interrupt status of IDBUS_ADDRHOLE_INT. (R/SS/WTC)

LP_SYSTEM_LP_CORE_AHB_TIMEOUT_INT_RAW The raw interrupt status of LP_CORE_AHB_TIMEOUT_INT. (R/SS/WTC)

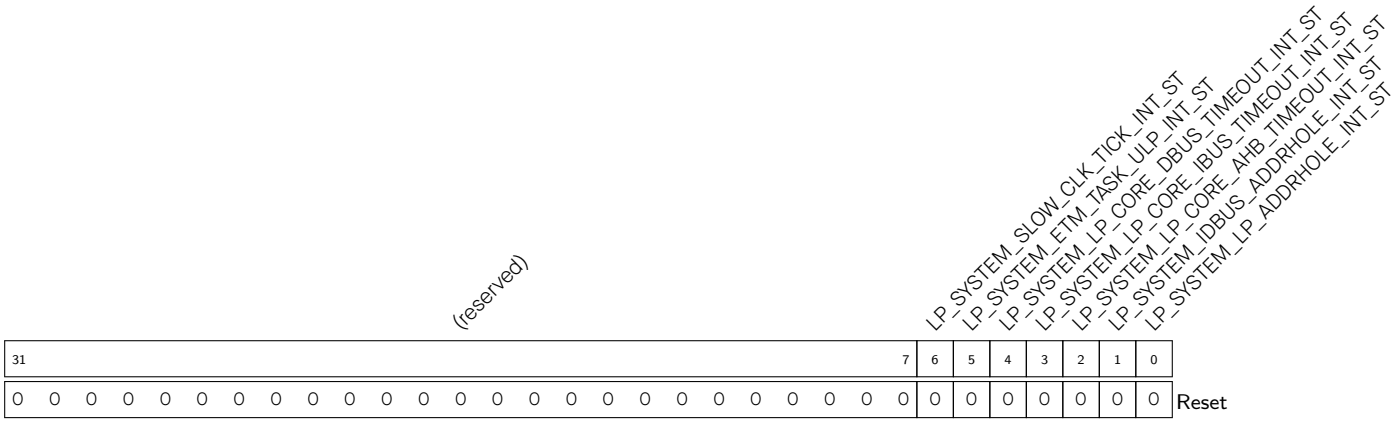
LP_SYSTEM_LP_CORE_IBUS_TIMEOUT_INT_RAW The raw interrupt status of LP_CORE_IBUS_TIMEOUT_INT. (R/SS/WTC)

LP_SYSTEM_LP_CORE_DBUS_TIMEOUT_INT_RAW The raw interrupt status of LP_CORE_DBUS_TIMEOUT_INT. (R/SS/WTC)

LP_SYSTEM_ETM_TASK_ULP_INT_RAW The raw interrupt status of ETM_TASK_ULP_INT. (R/SS/WTC)

LP_SYSTEM_SLOW_CLK_TICK_INT_RAW The raw interrupt status of SLOW_CLK_TICK_INT. (R/SS/WTC)

Register 21.126. LP_SYSTEM_INT_ST_REG (0x0174)



LP_SYSTEM_LP_ADDRHOLE_INT_ST The masked interrupt status of LP_ADDRHOLE_INT. (RO)

LP_SYSTEM_IDBUS_ADDRHOLE_INT_ST The masked interrupt status of IDBUS_ADDRHOLE_INT. (RO)

LP_SYSTEM_LP_CORE_AHB_TIMEOUT_INT_ST The masked interrupt status of LP_CORE_AHB_TIMEOUT_INT. (RO)

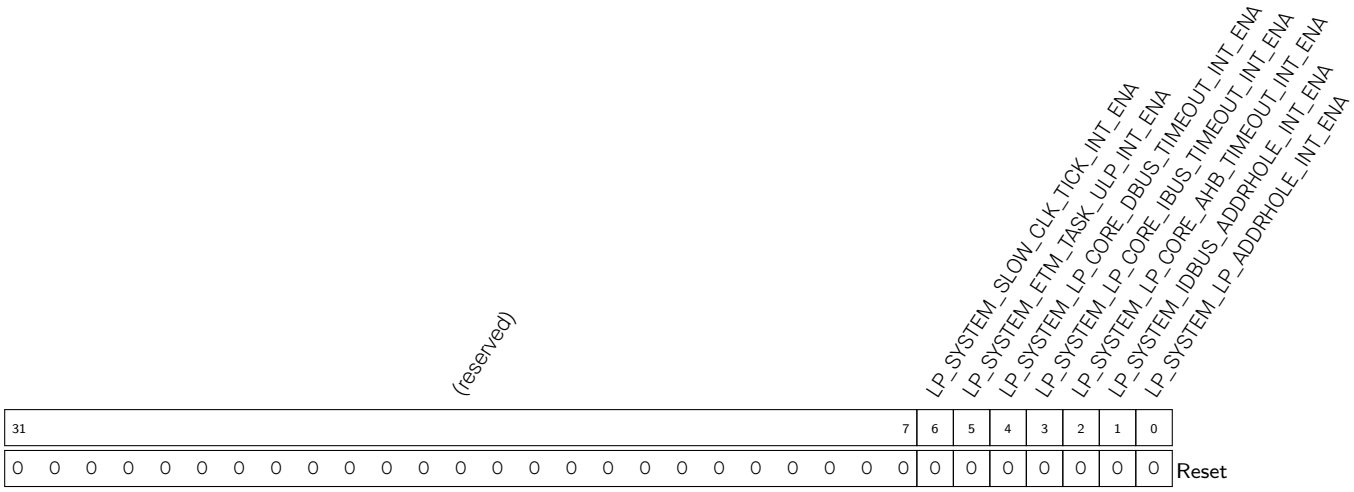
LP_SYSTEM_LP_CORE_IBUS_TIMEOUT_INT_ST The masked interrupt status of LP_CORE_IBUS_TIMEOUT_INT. (RO)

LP_SYSTEM_LP_CORE_DBUS_TIMEOUT_INT_ST The masked interrupt status of LP_CORE_DBUS_TIMEOUT_INT. (RO)

LP_SYSTEM_ETM_TASK_ULP_INT_ST The masked interrupt status of ETM_TASK_ULP_INT. (RO)

LP_SYSTEM_SLOW_CLK_TICK_INT_ST The masked interrupt status of SLOW_CLK_TICK_INT. (RO)

Register 21.127. LP_SYSTEM_INT_ENA_REG (0x0178)



LP_SYSTEM_LP_ADDRHOLE_INT_ENA Write 1 to enable LP_ADDRHOLE_INT. (R/W)

LP_SYSTEM_IDBUS_ADDRHOLE_INT_ENA Write 1 to enable IDBUS_ADDRHOLE_INT. (R/W)

LP_SYSTEM_LP_CORE_AHB_TIMEOUT_INT_ENA Write 1 to enable LP_CORE_AHB_TIMEOUT_INT. (R/W)

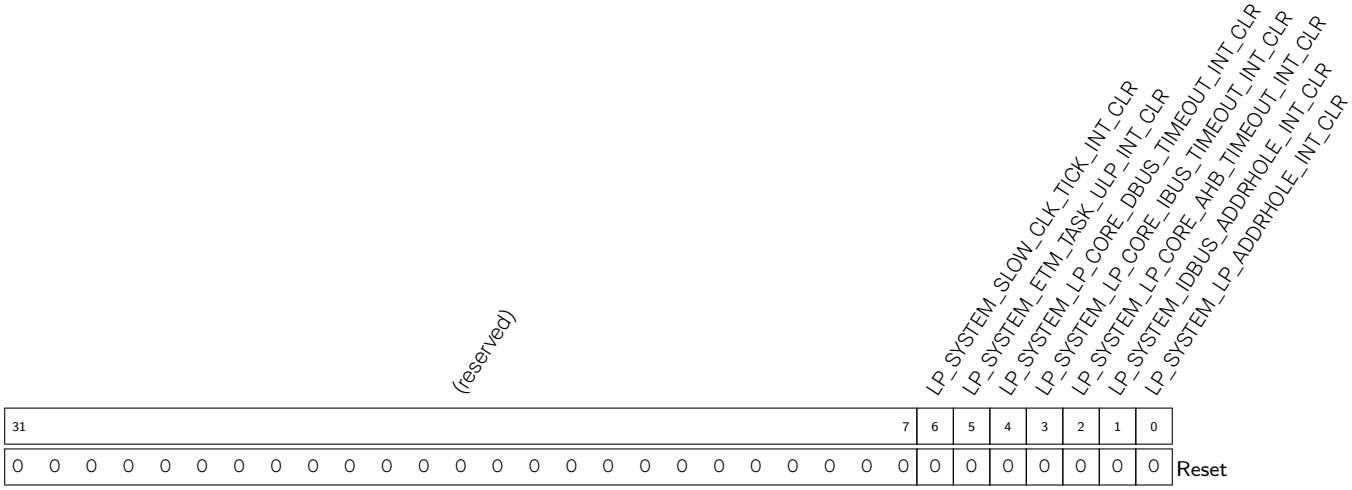
LP_SYSTEM_LP_CORE_IBUS_TIMEOUT_INT_ENA Write 1 to enable LP_CORE_IBUS_TIMEOUT_INT. (R/W)

LP_SYSTEM_LP_CORE_DBUS_TIMEOUT_INT_ENA Write 1 to enable LP_CORE_DBUS_TIMEOUT_INT. (R/W)

LP_SYSTEM_ETM_TASK_ULP_INT_ENA Write 1 to enable ETM_TASK_ULP_INT. (R/W)

LP_SYSTEM_SLOW_CLK_TICK_INT_ENA Write 1 to enable SLOW_CLK_TICK_INT. (R/W)

Register 21.128. LP_SYSTEM_INT_CLR_REG (0x017C)



LP_SYSTEM_LP_ADDRHOLE_INT_CLR Write 1 to clear LP_ADDRHOLE_INT. (WT)

LP_SYSTEM_IDBUS_ADDRHOLE_INT_CLR Write 1 to clear IDBUS_ADDRHOLE_INT. (WT)

LP_SYSTEM_LP_CORE_AHB_TIMEOUT_INT_CLR Write 1 to clear LP_CORE_AHB_TIMEOUT_INT. (WT)

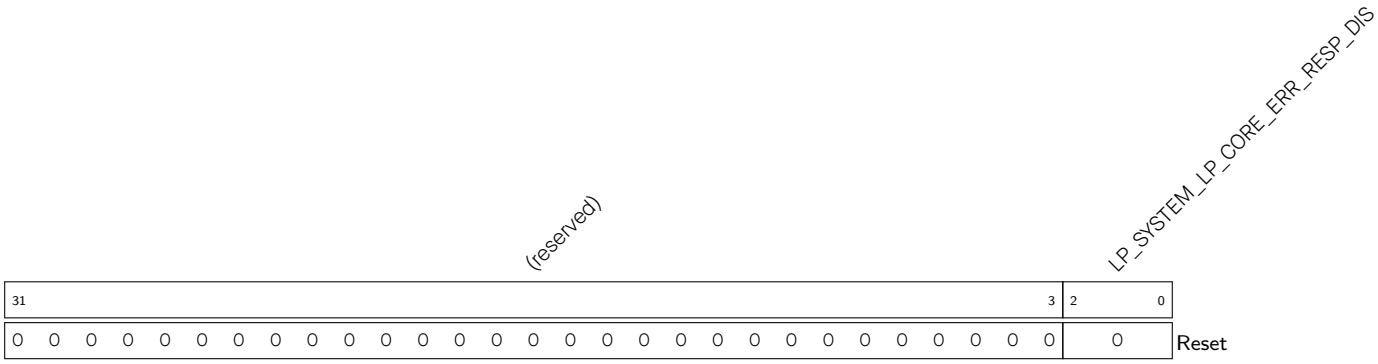
LP_SYSTEM_LP_CORE_IBUS_TIMEOUT_INT_CLR Write 1 to clear LP_CORE_IBUS_TIMEOUT_INT. (WT)

LP_SYSTEM_LP_CORE_DBUS_TIMEOUT_INT_CLR Write 1 to clear LP_CORE_DBUS_TIMEOUT_INT. (WT)

LP_SYSTEM_ETM_TASK_ULP_INT_CLR Write 1 to clear ETM_TASK_ULP_INT. (WT)

LP_SYSTEM_SLOW_CLK_TICK_INT_CLR Write 1 to clear SLOW_CLK_TICK_INT. (WT)

Register 21.129. LP_SYSTEM_LP_CORE_ERR_RESP_DIS_REG (0x01BC)



LP_SYSTEM_LP_CORE_ERR_RESP_DIS Configures whether to disable error response for LP CPU buses.

Bit[0]: Configures whether to disable error response for IBUS

0: Enable error response

1: Disable error response

Bit[1]: Configures whether to disable error response for DBUS

0: Enable error response

1: Disable error response

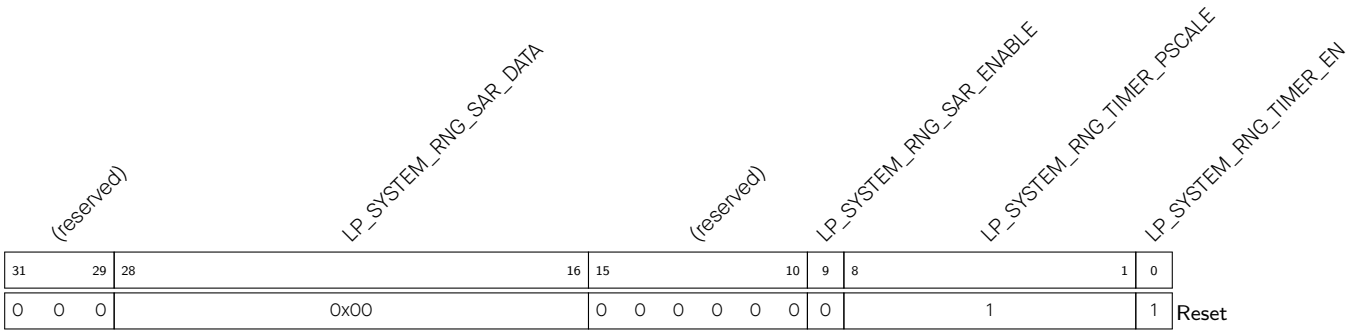
Bit[2]: Configures whether to disable error response for AHB bus

0: Enable error response

1: Disable error response

(R/W)

Register 21.130. LP_SYSTEM_RNG_CFG_REG (0x01C0)



LP_SYSTEM_RNG_TIMER_EN Configures whether to enable RNG timer.

0: Disable

1: Enable

(R/W)

LP_SYSTEM_RNG_TIMER_PSCALE Configures the clock division factor for the RNG timer. (R/W)

LP_SYSTEM_RNG_SAR_ENABLE Configures whether to enable RNG_SARADC.

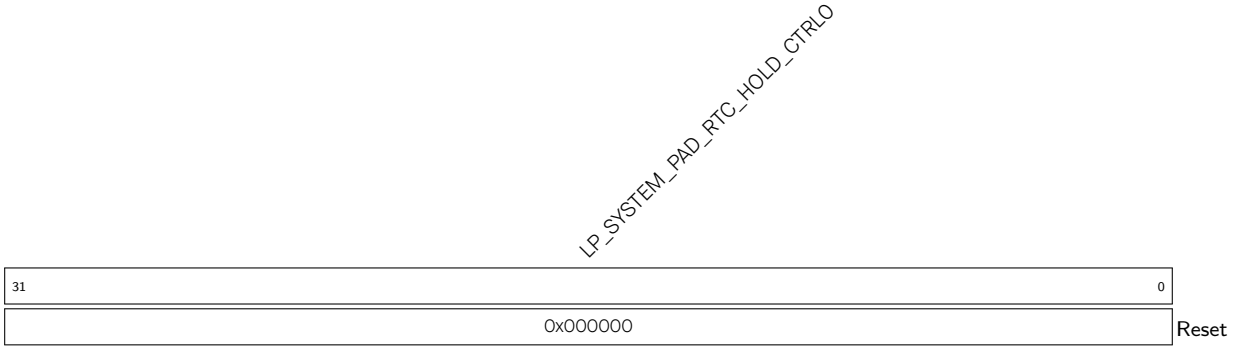
0: Disable

1: Enable

(R/W)

LP_SYSTEM_RNG_SAR_DATA Represents the analog SAR3_CAP value, which is used for debugging RNG_SARADC sample count. (RO)

Register 21.131. LP_SYSTEM_PAD_RTC_HOLD_CTRL0_REG (0x01C4)



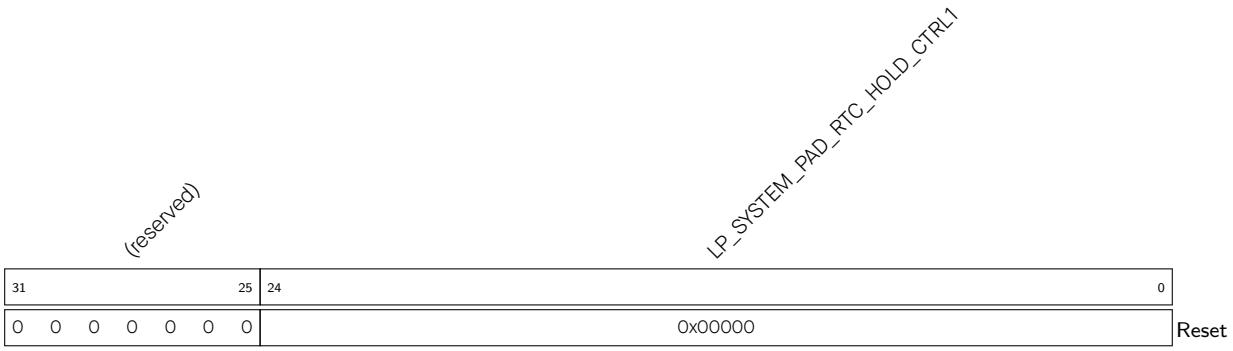
LP_SYSTEM_PAD_RTC_HOLD_CTRL0 Configures whether to hold the status of pad 0 ~ pad 31.

0: Not hold

1: Hold

(R/W)

Register 21.132. LP_SYSTEM_PAD_RTC_HOLD_CTRL1_REG (0x01C8)



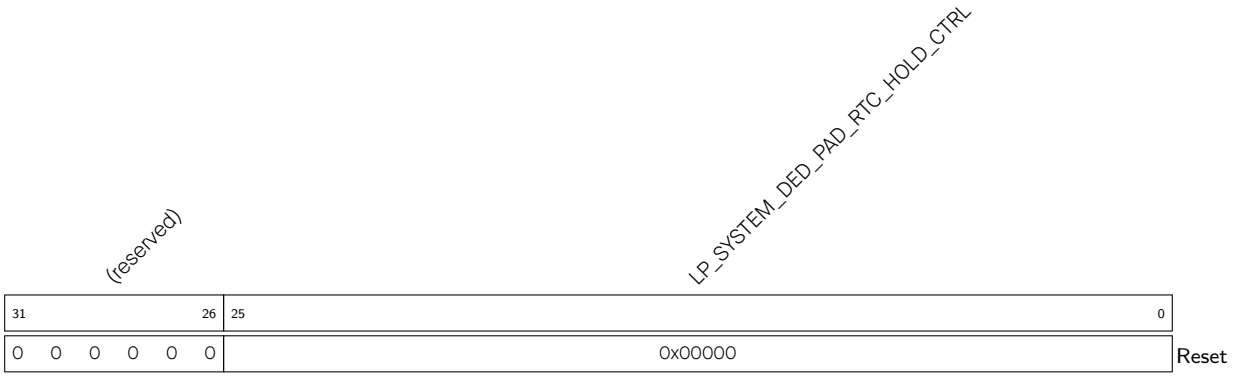
LP_SYSTEM_PAD_RTC_HOLD_CTRL1 Configures whether to hold the status of pad 32 ~ pad 56.

0: Not hold

1: Hold

(R/W)

Register 21.133. LP_SYSTEM_DED_PAD_RTC_HOLD_CTRL_REG (0x01CC)



LP_SYSTEM_DED_PAD_RTC_HOLD_CTRL Configures whether to hold pad status.

Bit[0:5]: Flash pads

0: Not hold

1: Hold

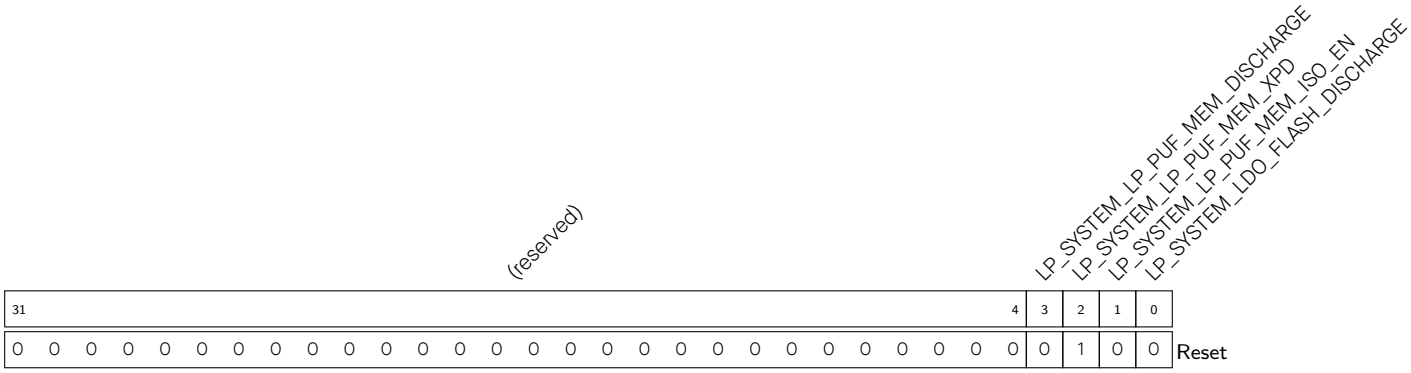
Bit[6:25]: PSRAM pads

0: Not hold

1: Hold

(R/W)

Register 21.134. LP_SYSTEM_DISCHARGE_REG (0x0200)



LP_SYSTEM_LDO_FLASH_DISCHARGE Configures whether to discharge LDO flash.

0: No effect

1: Discharge

(R/W)

LP_SYSTEM_LP_PUF_MEM_ISO_EN Configures whether to enable isolation for LP PUF memory.

0: Disable

1: Enable

(R/W)

LP_SYSTEM_LP_PUF_MEM_XPD Configures whether to power up LP PUF memory.

0: Power down

1: Power up

(R/W)

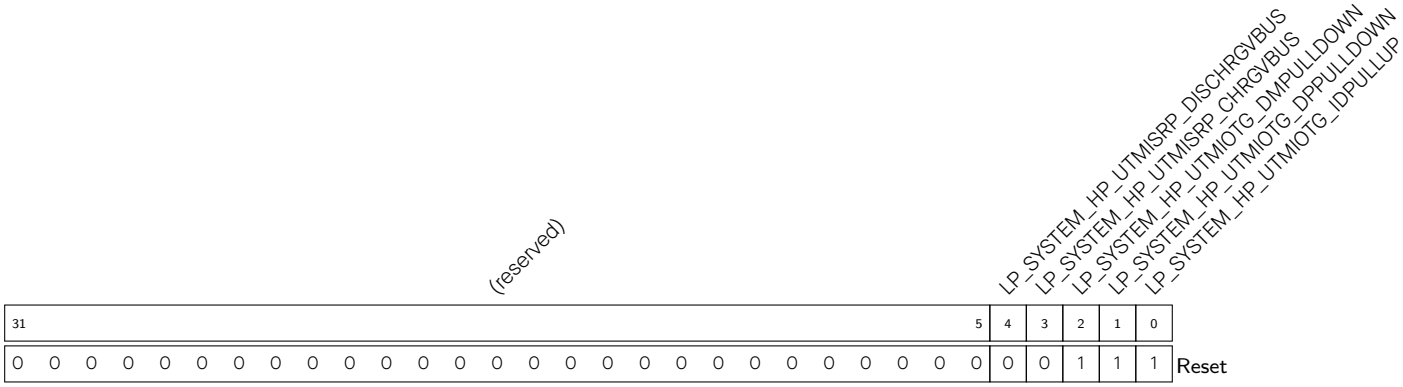
LP_SYSTEM_LP_PUF_MEM_DISCHARGE Configures whether to discharge LP PUF memory.

0: No effect

1: Discharge

(R/W)

Register 21.135. LP_SYSTEM_HP_USB_OTGHS_PHY_CTRL_REG (0x0204)



LP_SYSTEM_HP_UTMIOTG_IDPULLUP Configures whether to pull up USB OTG2.0 PHY ID.

0: Not pull up

1: Pull up

(R/W)

LP_SYSTEM_HP_UTMIOTG_DPPULLDOWN Configures whether to pull down USB OTG2.0 PHY DP.

0: Not pull down

1: Pull down

(R/W)

LP_SYSTEM_HP_UTMIOTG_DMPULLDOWN Configures whether to pull down USB OTG2.0 PHY DM.

0: Not pull down

1: Pull down

(R/W)

LP_SYSTEM_HP_UTMISRP_CHRGVBUS Configures whether to charge USB OTG2.0 PHY VBUS.

0: No effect

1: Charge

(R/W)

LP_SYSTEM_HP_UTMISRP_DISCHRGVBUS Configures whether to discharge USB OTG2.0 PHY VBUS.

0: No effect

1: Discharge

(R/W)

Chapter 22

Debug Assistant

22.1 Overview

Debug Assistant is an auxiliary module that features a set of functions to help locate bugs and issues during software debugging.

22.2 Features

- **Read/write monitoring**: Monitors whether the High-Performance dual-core CPU (HP CPU0 and HP CPU1) bus reads from or writes to a specified memory address space. A detected read or write in the monitored address space will trigger an interrupt.
- **Stack pointer (SP) monitoring**: Monitors whether the SP exceeds the specified address space. A bounds violation will trigger an interrupt.
- **Program counter (PC) logging**: Records the PC value. The developer can get the last PC value at the most recent reset of HP CPU0 or HP CPU1.
- **Bus access logging**: Records the information about bus access. When the HP CPU0, HP CPU1, or the Direct Memory Access controller (DMA) writes a specified value, the Debug Assistant module will record the data type, address of this write operation, and additionally the PC value when the write is performed by HP CPU0 or HP CPU1, and push such information to the HP L2MEM.

22.3 Functional Description

22.3.1 Region Read/Write Monitoring

The Debug Assistant module can monitor reads/writes performed by HP CPU0 bus in a certain address space, i.e., memory region. Whenever the bus reads or writes in the specified address space, an interrupt will be triggered. Please refer to Table 2.4-1 in Chapter 2 *High-Performance CPU*. In this chapter, when HP CPU0 bus accesses the data space, it is referred to as the Data bus, and when HP CPU0 bus accesses the AHB peripheral space, it is referred to as the Peripheral bus. The Data bus can monitor two memory regions (assuming they are HP CPU0 region 0 and HP CPU0 region 1, defined by developers' needs) at the same time, and so can Peripheral bus.

This is also the case for HP CPU1.

22.3.2 SP Monitoring

The Debug Assistant module can monitor the SP so as to prevent stack overflow or erroneous push/pop in HP CPU0. When the HP CPU0 SP goes below or beyond the lower or upper bound of the HP CPU0 SP monitored region, the module will record the PC pointer and generate an interrupt. The bound is configured by software.

This is also the case for HP CPU1.

22.3.3 PC Logging

In some cases, software developers want to know the PC at the last reset of HP CPU0. For instance, when the program is stuck and can only be reset, the developer may want to know where the program got stuck in order to debug. The Debug Assistant module can record the PC at the last reset of HP CPU0, which can be then read for software debugging.

This is also the case for HP CPU1.

22.3.4 CPU/DMA Bus Access Logging

The Debug Assistant module can record the information about the HP CPU0 Data bus's, HP CPU1 bus's, and DMA bus's write behaviors in real time. When a write operation occurs in or a specific value is written to a specified address space, the module will record the bus type, the address, PC (only when the write is performed by the HP CPU0 or HP CPU1, will PC be recorded), and other information, and then store the data in the HP L2MEM in a certain format. The specified address range must fall within the permitted memory regions of HP SPM or HP L2MEM.

22.4 Interrupts

The following interrupt sources from the Debug Assistant module can generate the interrupt signal ASSIST_DEBUG_INT.

- ASSIST_DEBUG_CORE_0_AREA_DRAM0_0_RD_INT: Triggered when the HP CPU0 Data bus reads in HP CPU0 region 0.
- ASSIST_DEBUG_CORE_0_AREA_DRAM0_0_WR_INT: Triggered when the HP CPU0 Data bus writes in HP CPU0 region 0.
- ASSIST_DEBUG_CORE_0_AREA_DRAM0_1_RD_INT: Triggered when the HP CPU0 Data bus reads in HP CPU0 region 1.
- ASSIST_DEBUG_CORE_0_AREA_DRAM0_1_WR_INT: Triggered when the HP CPU0 Data bus writes in HP CPU0 region 1.
- ASSIST_DEBUG_CORE_0_AREA_PIF_0_RD_INT: Triggered when the HP CPU0 Peripheral bus reads in HP CPU0 region 0.
- ASSIST_DEBUG_CORE_0_AREA_PIF_0_WR_INT: Triggered when the HP CPU0 Peripheral bus writes in HP CPU0 region 0.
- ASSIST_DEBUG_CORE_0_AREA_PIF_1_RD_INT: Triggered when the HP CPU0 Peripheral bus reads in HP CPU0 region 1.

- ASSIST_DEBUG_CORE_0_AREA_PIF_1_WR_INT: Triggered when the HP CPU0 Peripheral bus writes in HP CPU0 region 1.
- ASSIST_DEBUG_CORE_0_SP_SPILL_MIN_INT: Triggered when the HP CPU0 SP goes below the lower bound of the HP CPU0 SP monitored region.
- ASSIST_DEBUG_CORE_0_SP_SPILL_MAX_INT: Triggered when the HP CPU0 SP goes beyond the upper bound of the HP CPU0 SP monitored region.
- ASSIST_DEBUG_CORE_1_AREA_DRAM0_0_RD_INT: Triggered when the HP CPU1 Data bus reads in HP CPU1 region 0.
- ASSIST_DEBUG_CORE_1_AREA_DRAM0_0_WR_INT: Triggered when the HP CPU1 Data bus writes in HP CPU1 region 0.
- ASSIST_DEBUG_CORE_1_AREA_DRAM0_1_RD_INT: Triggered when the HP CPU1 Data bus reads in HP CPU1 region 1.
- ASSIST_DEBUG_CORE_1_AREA_DRAM0_1_WR_INT: Triggered when the HP CPU1 Data bus writes in HP CPU1 region 1.
- ASSIST_DEBUG_CORE_1_AREA_PIF_0_RD_INT: Triggered when the HP CPU1 Peripheral bus reads in HP CPU1 region 0.
- ASSIST_DEBUG_CORE_1_AREA_PIF_0_WR_INT: Triggered when the HP CPU1 Peripheral bus writes in HP CPU1 region 0.
- ASSIST_DEBUG_CORE_1_AREA_PIF_1_RD_INT: Triggered when the HP CPU1 Peripheral bus reads in HP CPU1 region 1.
- ASSIST_DEBUG_CORE_1_AREA_PIF_1_WR_INT: Triggered when the HP CPU1 Peripheral bus writes in HP CPU1 region 1.
- ASSIST_DEBUG_CORE_1_SP_SPILL_MIN_INT: Triggered when the HP CPU1 SP goes below the lower bound of the HP CPU1 SP monitored region.
- ASSIST_DEBUG_CORE_1_SP_SPILL_MAX_INT: Triggered when the HP CPU1 SP goes beyond the upper bound of the HP CPU1 SP monitored region.

Note:
For definitions of *interrupt*, *interrupt signal*, *interrupt source*, and their correlations, please refer to Chapter 13 *Interrupt Matrix* > Section 13.2 *Interrupt Terminology in ESP32-P4*.

Each interrupt source can be configured by a common set of registers that are described in Section *Interrupt Configuration Registers*. The specific registers can be found in Section 22.6 *Register Summary*.

22.5 Recommended Operation

22.5.1 Region Monitoring and SP Monitoring Configuration

The Debug Assistant module can monitor reads and writes performed by the Data bus and Peripheral bus of HP CPU0 and HP CPU1. Two memory regions on each bus can be monitored at the same time. All the

monitoring modes supported by the Debug Assistant module are listed below:

- Monitoring the read/write operations performed by Data bus
  - HP CPU0 Data bus reads in HP CPU0 region 0
  - HP CPU0 Data bus writes in HP CPU0 region 0
  - HP CPU0 Data bus reads in HP CPU0 region 1
  - HP CPU0 Data bus writes in HP CPU0 region 1
  - HP CPU1 Data bus reads in HP CPU1 region 0
  - HP CPU1 Data bus writes in HP CPU1 region 0
  - HP CPU1 Data bus reads in HP CPU1 region 1
  - HP CPU1 Data bus writes in HP CPU1 region 1
- Monitoring the read/write operations performed by Peripheral bus
  - HP CPU0 Peripheral bus reads in HP CPU0 region 0
  - HP CPU0 Peripheral bus writes in HP CPU0 region 0
  - HP CPU0 Peripheral bus reads in HP CPU0 region 1
  - HP CPU0 Peripheral bus writes in HP CPU0 region 1
  - HP CPU1 Peripheral bus reads in HP CPU1 region 0
  - HP CPU1 Peripheral bus writes in HP CPU1 region 0
  - HP CPU1 Peripheral bus reads in HP CPU1 region 1
  - HP CPU1 Peripheral bus writes in HP CPU1 region 1
- Monitoring SP bounds violations
  - HP CPU0 SP goes beyond the upper bound of the HP CPU0 SP monitored region
  - HP CPU0 SP goes below the lower bound of the HP CPU0 SP monitored region
  - HP CPU1 SP goes beyond the upper bound of the HP CPU1 SP monitored region
  - HP CPU1 SP goes below the lower bound of the HP CPU1 SP monitored region

The configuration process for region monitoring and SP monitoring is as follows:

1. Configure the monitored region and SP bounds.
   - Configure HP CPU0 Data bus region 0 with ASSIST_DEBUG_CORE_0_AREA_DRAM0_0_MIN_REG and ASSIST_DEBUG_CORE_0_AREA_DRAM0_0_MAX_REG.
   - Configure HP CPU0 Data bus region 1 with ASSIST_DEBUG_CORE_0_AREA_DRAM0_1_MIN_REG and ASSIST_DEBUG_CORE_0_AREA_DRAM0_1_MAX_REG.
   - Configure HP CPU0 Peripheral bus region 0 with ASSIST_DEBUG_CORE_0_AREA_PIF_0_MIN_REG and ASSIST_DEBUG_CORE_0_AREA_PIF_0_MAX_REG.
   - Configure HP CPU0 Peripheral bus region 1 with ASSIST_DEBUG_CORE_0_AREA_PIF_1_MIN_REG and ASSIST_DEBUG_CORE_0_AREA_PIF_1_MAX_REG.

- Configure HP CPU0 SP bounds with ASSIST_DEBUG_CORE_0_SP_MIN_REG and ASSIST_DEBUG_CORE_0_SP_MAX_REG.
- Configure HP CPU1 Data bus region 0 with ASSIST_DEBUG_CORE_1_AREA_DRAM0_0_MIN_REG and ASSIST_DEBUG_CORE_1_AREA_DRAM0_0_MAX_REG.
- Configure HP CPU1 Data bus region 1 with ASSIST_DEBUG_CORE_1_AREA_DRAM0_1_MIN_REG and ASSIST_DEBUG_CORE_1_AREA_DRAM0_1_MAX_REG.
- Configure HP CPU1 Peripheral bus region 0 with ASSIST_DEBUG_CORE_1_AREA_PIF_0_MIN_REG and ASSIST_DEBUG_CORE_1_AREA_PIF_0_MAX_REG.
- Configure HP CPU1 Peripheral bus region 1 with ASSIST_DEBUG_CORE_1_AREA_PIF_1_MIN_REG and ASSIST_DEBUG_CORE_1_AREA_PIF_1_MAX_REG.
- Configure HP CPU1 SP bounds with ASSIST_DEBUG_CORE_1_SP_MIN_REG and ASSIST_DEBUG_CORE_1_SP_MAX_REG.

2. Configure interrupts.
   - Configure ASSIST_DEBUG_CORE_0_INTR_ENA_REG or ASSIST_DEBUG_CORE_1_INTR_ENA_REG to enable the interrupt of a monitoring mode.
   - Read ASSIST_DEBUG_CORE_0_INTR_RAW_REG or ASSIST_DEBUG_CORE_1_INTR_RAW_REG to get the raw interrupt status of a monitoring mode.
   - Configure ASSIST_DEBUG_CORE_0_INTR_CLR_REG or ASSIST_DEBUG_CORE_1_INTR_CLR_REG to clear the interrupt of a monitoring mode.
3. Configure ASSIST_DEBUG_CORE_0_MONTR_ENA_REG or ASSIST_DEBUG_CORE_1_MONTR_ENA_REG to enable the monitoring mode(s). Various monitoring modes can be enabled at the same time.

Assuming that Debug Assistant module needs to monitor whether HP CPU0 Data bus has written to [A ~ B] address space, the user can enable monitoring in either HP CPU0 Data bus region 0 or region 1. The following configuration process is based on region 0:

1. Configure ASSIST_DEBUG_CORE_0_RCD_PDEBUGEN to 1 to enable HP CPU0 to update the PC signals to the Debug Assistant module.
2. Configure ASSIST_DEBUG_CORE_0_AREA_DRAM0_0_MIN_REG to Address A.
3. Configure ASSIST_DEBUG_CORE_0_AREA_DRAM0_0_MAX_REG to Address B.
4. Configure ASSIST_DEBUG_CORE_0_INTR_ENA_REG bit[1] to enable the interrupt for write operations by HP CPU0 Data bus in region 0.
5. Configure ASSIST_DEBUG_CORE_0_MONTR_ENA_REG bit[1] to enable monitoring write operations by HP CPU0 Data bus in region 0.
6. Configure interrupt matrix to map ASSIST_DEBUG_INTR into HP CPU0/1 interrupt (please refer to Chapter 13 *Interrupt Matrix*).
7. After the interrupt is triggered:
   - Read ASSIST_DEBUG_CORE_0_INTR_RAW_REG and ASSIST_DEBUG_CORE_1_INTR_RAW_REG to learn which operation triggered the interrupt.

- If the interrupt is triggered by HP CPU0 region monitoring, read ASSIST_DEBUG_CORE_0_AREA_PC_REG for the HP CPU0 PC value, and ASSIST_DEBUG_CORE_0_AREA_SP_REG for the HP CPU0 SP.
- If the interrupt is triggered by SP monitoring, read ASSIST_DEBUG_CORE_0_SP_PC_REG for the HP CPU0 PC value.
- Write 1 to the corresponding bits of ASSIST_DEBUG_CORE_0_INTR_RAW_REG and ASSIST_DEBUG_CORE_1_INTR_CLR_REG to clear the interrupts.

22.5.2 PC Logging Configuration

Configure ASSIST_DEBUG_CORE_0_RCD_PDEBUGEN to 1 to enable HP CPU0 to update the PC signals to the Debug Assistant module. If ASSIST_DEBUG_CORE_0_RCD_RECORDEN is also configured to 1, ASSIST_DEBUG_CORE_0_RCD_PDEBUGPC_REG will record the HP CPU0's PC signal and ASSIST_DEBUG_CORE_0_RCD_PDEBUGSP_REG will record the HP CPU0 SP value. Otherwise, the two registers keep the original values.

When the HP CPU0 resets, ASSIST_DEBUG_CORE_0_RCD_EN_REG will reset, while ASSIST_DEBUG_CORE_0_RCD_PDEBUGPC_REG and ASSIST_DEBUG_CORE_0_RCD_PDEBUGSP_REG will not. Therefore, the two registers will keep the PC value and SP value at the HP CPU0 reset.

This is also the case for HP CPU1.

22.5.3 CPU/DMA Bus Access Logging Configuration

22.5.3.1 HP CPU0/1 Bus Access Logging Configuration

The configuration process for HP CPU0/1 bus access logging (SPM Monitor) is described below.

1. Configure monitored address space.
   - Configure SPM_MEM_MONITOR_LOG_MIN_REG and SPM_MEM_MONITOR_LOG_MAX_REG to specify monitored address space. The monitored address space should range from 0x3010_0000 to 0x3010_1FFF.
2. Configure the monitoring mode with SPM_MEM_MONITOR_LOG_MODE:
   - Write monitoring (whether the bus has write operations)
   - Word monitoring (whether the bus writes a specific word)
   - Halfword monitoring (whether the bus writes a specific halfword)
   - Byte monitoring (whether the bus writes a specific byte)
3. Configure the specific values to be monitored.
   - In word monitoring mode, SPM_MEM_MONITOR_LOG_CHECK_DATA_REG specifies the monitored word.
   - In halfword monitoring mode, SPM_MEM_MONITOR_LOG_CHECK_DATA_REG[15:0] specifies the monitored halfword.
   - In byte monitoring mode, SPM_MEM_MONITOR_LOG_CHECK_DATA_REG[7:0] specifies the monitored byte.

- SPM_MEM_MONITOR_LOG_DATA_MASK_REG is used to mask the byte specified in SPM_MEM_MONITOR_LOG_CHECK_DATA_REG. A masked byte can be any value. For example, in word monitoring, if SPM_MEM_MONITOR_LOG_CHECK_DATA_REG is configured to 0x01020304 and SPM_MEM_MONITOR_LOG_DATA_MASK_REG is configured to 0x1, then any writes of the data matching the 0x010203XX pattern by the bus will be recorded.

4. Configure the storage space for recorded data.
   - SPM_MEM_MONITOR_LOG_MEM_START_REG and SPM_MEM_MONITOR_LOG_MEM_END_REG specify the storage space for recorded data. The storage space must be in the range of 0x4080_0000 ~ 0x4087_FFFF.
   - Set SPM_MEM_MONITOR_LOG_MEM_ADDR_UPDATE_REG to update the value in SPM_MEM_MONITOR_LOG_MEM_CURRENT_ADDR_REG to SPM_MEM_MONITOR_LOG_MEM_START_REG.
   - Configure the permission for the SPM Monitor function of Debug Assistant module to access the HP L2MEM. Only when the access permission is enabled can the Debug Assistant module access the HP L2MEM. For more information, please refer to TEE_DMA_SPM_MON_PMS_W_REG and other relevant registers in Chapter 20 *Permission Control (PMS)*.
5. Configure the writing mode for the recorded data: loop mode or non-loop mode.
   - In loop mode, writing to the specified address space is performed in loops. When writing reaches the end address, it will return to the starting address and continue, overwriting the previously recorded data. Set SPM_MEM_MONITOR_LOG_MEM_LOOP_ENABLE to enable loop mode.
     For example, there are 10 write operations (1 ~ 10) to address space 0 ~ 4 during bus access. After the 5th operation writes to address 4, the 6th operation will start writing from address 0. The 6th to 10th operations will overwrite the previous data written by the 1th to 5th operations.
   - In non-loop mode, when writing reaches the end address, it will stop at the end address and dump the remaining data, not overwriting the previously recorded data. Clear SPM_MEM_MONITOR_LOG_MEM_LOOP_ENABLE to use non-loop mode.
     For example, there are 10 write operations (1 ~ 10) to address space 0 ~ 4 during bus access. After the 5th operation writes to address 4, the 6th to 10th write operations will stop at address 4 and will not be performed any more. Therefore, the address 0 ~ 4 stores the values written by the 1 ~ 5 operations and the values of the 6 ~ 10 operations are dumped.
6. Configure bus enable registers.
   - Enable HP CPU0 or HP CPU1 bus access logging with SPM_MEM_MONITOR_LOG_CORE_ENA. They can be enabled at the same time.

The Debug Assistant module first writes the recorded data of HP CPU0 or HP CPU1 to the internal buffer A, and then fetches the data from the buffer and writes it to the configured memory space. When the monitored behaviors are triggered continuously, the generated recording packets may fully occupy the buffer, making it unable to take any incoming packets. At this time, the module dumps these incoming packets and buffers a HP CPU LOST packet instead before the buffer reaches its capacity. However, it is only possible to know the bus type of the dumped packets when the HP CPU LOST packet was generated, but not the number of the dumped packets before the generation.

When bus access logging is finished, the recorded data can be read from memory for decoding. The recorded data is in three packet formats, namely HP CPU0 packet (corresponding to HP CPU0 Data bus), HP

CPU1 packet (corresponding to HP CPU1 Data bus), and HP CPU LOST packet. The packet formats are shown in Table 22.5-1 and 22.5-2.

Table 22.5-1. HP CPU0/1 Packet Format

Bit[63:33]	Bit[32]	Bit[31:3]	Bit[2:1]	Bit[0]
pc_offset	anchored (1)	addr_offset	format	anchored (0)

Table 22.5-2. HP CPU LOST Packet Format

Bit[31:7]	Bit[6:3]	Bit[2:1]	Bit[0]
reserved	lost_source	format	anchored (0)

It can be seen from the data packet formats that the size of the HP CPU0/1 packet is 64 bits and that of the HP CPU LOST packet is 32 bits. These packets contain the following fields:

- **format** – the packet type. 0: HP CPU0 packet; 1: HP CPU1 packet; 2: HP CPU LOST packet; 3: Reserved.
- **pc_offset** - the offset of the HP CPU0/1 PC register at the time of access. Actual PC = pc_offset + 0x0000_0000.
- **addr_offset** - the address offset of a write operation. Actual address = addr_offset + SPM_MEM_MONITOR_LOG_MIN_REG[31:4], 4'h0.
- **lost_source** - the dumped packets when the HP CPU LOST packet was generated.
  - Bit[4:3]: 0 indicates HP CPU0 packets were not dumped. 3 indicates HP CPU0 packets were dumped. Other values are reserved.
  - Bit[6:5]: 0 indicates HP CPU1 packets were not dumped. 3 indicates HP CPU1 packets were dumped. Other values are reserved.
- **anchored** - the location of the 32 bits in the data packet. 0: Lower 32 bits. 1: Higher 32 bits.

The internal buffer of the module is 32 bits wide. When the HP CPU0 and HP CPU1 bus access loggings are both enabled and the record data is generated at the same time, the HP CPU0 data packets are first buffered, then the HP CPU1 packets. The Debug Assistant will automatically fetch the buffered data and store it in 32-bit data width into the specified memory space.

In loop mode, data looping several times in the storage memory may cause residual data, which can interfere with packet parsing. For example, the lower 32 bits of a HP CPU0/1 packet are overwritten, thus making its higher 32 bits residual data. Therefore, users need to filter out the possible residual data in order to determine the starting position of the first valid packet with SPM_MEM_MONITOR_LOG_MEM_CURRENT_ADDR_REG. Once the starting position of the packet is identified, check the anchored bit value of the packet. If it is 0, the data will be retained. If it is 1, it will be dumped.

The process of packet parsing is described below:

- Determine whether there is a data overflow with SPM_MEM_MONITOR_LOG_MEM_FULL_FLAG.
  - If no, the address space to read is SPM_MEM_MONITOR_LOG_MEM_START_REG ~ SPM_MEM_MONITOR_LOG_MEM_CURRENT_ADDR_REG - 4.

  - If yes and the loop mode is enabled, the address space is SPM_MEM_MONITOR_LOG_MEM_CURRENT_ADDR_REG ~ SPM_MEM_MONITOR_LOG_MEM_END_REG and SPM_MEM_MONITOR_LOG_MEM_START_REG ~ SPM_MEM_MONITOR_LOG_MEM_CURRENT_ADDR_REG - 4.
  - If yes and loop mode is not enabled, the address space is SPM_MEM_MONITOR_LOG_MEM_START_REG ~ SPM_MEM_MONITOR_LOG_MEM_END_REG.
- Read and parse data from the starting address. Read 32 bits each time.

After packet parsing is completed, clear the SPM_MEM_MONITOR_LOG_MEM_FULL_FLAG flag bit by setting SPM_MEM_MONITOR_CLR_LOG_MEM_FULL_FLAG.

22.5.3.2 DMA Bus Access Logging Configuration

The Debug Assistant module supports a total of four DMA bus channel groups, DMA_0, DMA_1, DMA_2, and DMA_3 channel groups. DMA_0 and DMA_1 channel groups are reserved and cannot be used. The DMA bus for the DMA_2 channel group is the bus on which the AXI matrix accesses the HP L2MEM. The DMA bus for the DMA_3 channel group is the bus on which the AHB matrix accesses the HP L2MEM. For more information, see Chapter 9 *System and Memory* > Section 9.3.2. The configuration process for DMA bus access logging (L2MEM Monitor) is described below.

1. Configure monitored address space.
   - Configure L2_MEM_MONITOR_LOG_MIN_REG and L2_MEM_MONITOR_LOG_MAX_REG to specify monitored address space. The monitored address space should range from 0x4FF0_0000 to 0x4FFB_FFFF or from 0x8FF0_0000 to 0x8FFB_FFFF.
2. Configure the monitoring mode with L2_MEM_MONITOR_LOG_MODE:
   - Write monitoring (whether the bus has write operations)
   - Word monitoring (whether the bus writes a specific word)
   - Halfword monitoring (whether the bus writes a specific halfword)
   - Byte monitoring (whether the bus writes a specific byte)
3. Configure the specific values to be monitored.
   - In word monitoring mode, L2_MEM_MONITOR_LOG_CHECK_DATA_REG specifies the monitored word.
   - In halfword monitoring mode, L2_MEM_MONITOR_LOG_CHECK_DATA_REG[15:0] specifies the monitored halfword.
   - In byte monitoring mode, L2_MEM_MONITOR_LOG_CHECK_DATA_REG[7:0] specifies the monitored byte.
   - L2_MEM_MONITOR_LOG_DATA_MASK_REG is used to mask the byte specified in L2_MEM_MONITOR_LOG_CHECK_DATA_REG. A masked byte can be any value. For example, in word monitoring, if L2_MEM_MONITOR_LOG_CHECK_DATA_REG is configured to 0x01020304 and L2_MEM_MONITOR_LOG_DATA_MASK_REG is configured to 0x1, then any writes of the data matching the 0x010203XX pattern by the bus will be recorded.
4. Configure the storage space for recorded data.

- L2_MEM_MONITOR_LOG_MEM_START_REG and L2_MEM_MONITOR_LOG_MEM_END_REG specify the storage space for recorded data. The storage space must be in the range of 0x4080_0000 ~ 0x4087_FFFF.
- Set L2_MEM_MONITOR_LOG_MEM_ADDR_UPDATE_REG to update the value in L2_MEM_MONITOR_LOG_MEM_CURRENT_ADDR_REG to L2_MEM_MONITOR_LOG_MEM_START_REG.
- Configure the permission for the L2MEM Monitor function of Debug Assistant module to access the HP L2MEM. Only when the access permission is enabled can the Debug Assistant module access the HP L2MEM. For more information, please refer to TEE_DMA_L2MEM_MON_PMS_W_REG and other relevant registers in Chapter 20 *Permission Control (PMS)*.

5. Configure the writing mode for the recorded data: loop mode or non-loop mode.
   - In loop mode, writing to the specified address space is performed in loops. When writing reaches the end address, it will return to the starting address and continue, overwriting the previously recorded data. Set L2_MEM_MONITOR_LOG_MEM_LOOP_ENABLE to enable loop mode.
     For example, there are 10 write operations (1 ~ 10) to address space 0 ~ 4 during bus access. After the 5th operation writes to address 4, the 6th operation will start writing from address 0. The 6th to 10th operations will overwrite the previous data written by the 1th to 5th operations.
   - In non-loop mode, when writing reaches the end address, it will stop at the end address and dump the remaining data, not overwriting the previously recorded data. Clear L2_MEM_MONITOR_LOG_MEM_LOOP_ENABLE to use non-loop mode.
     For example, there are 10 write operations (1 ~ 10) to address space 0 ~ 4 during bus access. After the 5th operation writes to address 4, the 6th to 10th write operations will stop at address 4 and will not be performed any more. Therefore, the address 0 ~ 4 stores the values written by the 1 ~ 5 operations and the values of the 6 ~ 10 operations are dumped.
   - See the example in 22.5.3.1 > step 5.
6. Configure bus enable registers.
   - Enable DMA_2 and DMA_3 channel groups bus access logging with L2_MEM_MONITOR_LOG_DMA_2_ENA and L2_MEM_MONITOR_LOG_DMA_3_ENA respectively. They can be enabled at the same time.

The Debug Assistant module first writes the DMA recorded data to the internal buffer B, and then fetches the data from the buffer and writes it to the configured memory space. When the monitored behaviors are triggered continuously, the generated recording packets may fully occupy the buffer, making it unable to take any incoming packets. At this time, the module dumps these incoming packets and buffers a DMA LOST packet instead before the buffer reaches its capacity. However, it is only possible to know the bus type of the dumped packets when the DMA LOST packet was generated, but not the number of the dumped packets before the generation.

When bus access logging is finished, the recorded data can be read from memory for decoding. The recorded data is in three packet formats, namely DMA_2 packet (corresponding to DMA_2 Data bus), DMA_3 packet (corresponding to DMA_3 Data bus), and DMA LOST packet. The packet formats are shown in Table 22.5-3, 22.5-4, and 22.5-5.

Table 22.5-3. DMA_2 Packet Format

Bit[31:9]	Bit[8:4]	Bit[3:1]	Bit[0]
addr_offset0	reserved	format	anchored (0)

Table 22.5-4. DMA_3 Packet Format

Bit[31:9]	Bit[8:4]	Bit[3:1]	Bit[0]
addr_offset1	reserved	format	anchored (0)

Table 22.5-5. DMA LOST Packet Format

Bit[31:10]	Bit[9:8]	Bit[7:4]	Bit[3:1]	Bit[0]
reserved	lost_source	reserved	format	anchored (0)

It can be seen from the data packet formats that the size of the DMA_2, DMA_3, and DMA LOST packet is 32 bits. These packets contain the following fields:

- **format** – the packet type. 4: DMA_2 packet; 5: DMA_3 packet; 6: DMA LOST packet; other values: Reserved.
- **addr_offset0** - the address offset of a write operation. Actual address = addr_offset0 + L2_MEM_MONITOR_LOG_MIN_REG[31:5],5'h0.
- **addr_offset1** - the address offset of a write operation. Actual address = addr_offset1 + L2_MEM_MONITOR_LOG_MIN_REG[31:2],2'h0.
- **lost_source** - the dumped packets when the DMA LOST packet was generated.
  - Bit[8]: 0 indicates DMA_2 packets were not dumped. 1 indicates DMA_2 packets were dumped.
  - Bit[9]: 0 indicates DMA_3 packets were not dumped. 1 indicates DMA_3 packets were dumped.
- **anchored** - the location of the 32 bits in the data packet. 0: Lower 32 bits. 1: Higher 32 bits.

The internal buffer of the module is 32 bits wide. When the bus access loggings of DMA_2 and DMA_3 channel groups are both enabled and the record data is generated at the same time, the DMA_2 data packets are first buffered, then the DMA_3 packets. The Debug Assistant will automatically fetch the buffered data and store it in 32-bit data width into the specified memory space.

The process of packet parsing is described below:

- Determine whether there is a data overflow with L2_MEM_MONITOR_LOG_MEM_FULL_FLAG.
  - If no, the address space to read is L2_MEM_MONITOR_LOG_MEM_START_REG ~ L2_MEM_MONITOR_LOG_MEM_CURRENT_ADDR_REG - 4.
  - If yes and the loop mode is enabled, the address space is L2_MEM_MONITOR_LOG_MEM_CURRENT_ADDR_REG ~ L2_MEM_MONITOR_LOG_MEM_END_REG and L2_MEM_MONITOR_LOG_MEM_START_REG ~ L2_MEM_MONITOR_LOG_MEM_CURRENT_ADDR_REG - 4.
  - If yes and loop mode is not enabled, the address space is L2_MEM_MONITOR_LOG_MEM_START_REG ~ L2_MEM_MONITOR_LOG_MEM_END_REG.

- Read and parse data from the starting address. Read 32 bits each time.

After packet parsing is completed, clear the L2_MEM_MONITOR_LOG_MEM_FULL_FLAG flag bit by setting L2_MEM_MONITOR_CLR_LOG_MEM_FULL_FLAG.

22.6 Register Summary

The addresses of **HP CPU bus logging configuration registers** (see 22.6.1) in this section are relative to the **SPM Monitor** base address. The addresses of **DMA bus logging configuration registers** (see 22.6.2) in this section are relative to the **L2MEM Monitor** base address. The addresses of other registers (see 22.6.3) are relative to the **Bus Monitor** base address. All base addresses are provided in Table 9.3-2 in Chapter 9 *System and Memory*.

The abbreviations given in Column **Access** are explained in Section *Access Types for Registers*.

22.6.1 HP CPU Bus Logging Configuration Register Summary

Name	Description	Address	Access
HP CPU bus logging configuration registers			
SPM_MEM_MONITOR_LOG_SETTING_REG	Configures bus access logging	0x0000	R/W
SPM_MEM_MONITOR_LOG_CHECK_DATA_REG	Configures monitored data in bus access logging	0x0008	R/W
SPM_MEM_MONITOR_LOG_DATA_MASK_REG	Configures masked data in bus access logging	0x000C	R/W
SPM_MEM_MONITOR_LOG_MIN_REG	Configures monitored address space in bus access logging	0x0010	R/W
SPM_MEM_MONITOR_LOG_MAX_REG	Configures monitored address space in bus access logging	0x0014	R/W
SPM_MEM_MONITOR_LOG_MEM_START_REG	Configures the starting address of the memory for recorded data	0x0018	R/W
SPM_MEM_MONITOR_LOG_MEM_END_REG	Configures the end address of the memory for recorded data	0x001C	R/W
SPM_MEM_MONITOR_LOG_MEM_CURRENT_ADDR_REG	Represents the address for the next write	0x0020	RO
SPM_MEM_MONITOR_LOG_MEM_ADDR_UPDATE_REG	Updates the address for the next write to the starting address for the recorded data	0x0024	WT
SPM_MEM_MONITOR_LOG_MEM_FULL_FLAG_REG	Represents the logging overflow status	0x0028	varies
Clock control register			
SPM_MEM_MONITOR_CLOCK_GATE_REG	Clock control register	0x002C	R/W
Version control register			
SPM_MEM_MONITOR_DATE_REG	Version control register	0x03FC	R/W

22.6.2 DMA Bus Logging Configuration Register Summary

Name	Description	Address	Access
DMA bus logging configuration registers			

Name	Description	Address	Access
L2_MEM_MONITOR_LOG_SETTING_REG	Configures bus access logging	0x0000	R/W
L2_MEM_MONITOR_LOG_SETTING1_REG	Configures bus access logging	0x0004	R/W
L2_MEM_MONITOR_LOG_CHECK_DATA_REG	Configures monitored data in bus access logging	0x0008	R/W
L2_MEM_MONITOR_LOG_DATA_MASK_REG	Configures masked data in bus access logging	0x000C	R/W
L2_MEM_MONITOR_LOG_MIN_REG	Configures monitored address space in bus access logging	0x0010	R/W
L2_MEM_MONITOR_LOG_MAX_REG	Configures monitored address space in bus access logging	0x0014	R/W
L2_MEM_MONITOR_LOG_MEM_START_REG	Configures the starting address of the memory for recorded data	0x0018	R/W
L2_MEM_MONITOR_LOG_MEM_END_REG	Configures the end address of the memory for recorded data	0x001C	R/W
L2_MEM_MONITOR_LOG_MEM_CURRENT_ADDR_REG	Represents the address for the next write	0x0020	RO
L2_MEM_MONITOR_LOG_MEM_ADDR_UPDATE_REG	Updates the address for the next write to the starting address for the recorded data	0x0024	WT
L2_MEM_MONITOR_LOG_MEM_FULL_FLAG_REG	Represents the logging overflow status	0x0028	varies
Clock control register			
L2_MEM_MONITOR_CLOCK_GATE_REG	Clock control register	0x002C	R/W
Version control register			
L2_MEM_MONITOR_DATE_REG	Version control register	0x03FC	R/W

22.6.3 Summary of Other Registers

Name	Description	Address	Access
Monitor configuration registers			
ASSIST_DEBUG_CORE_0_MONTR_ENA_REG	Configures whether to enable HP CPU0 monitoring	0x0000	R/W
ASSIST_DEBUG_CORE_0_AREA_DRAM0_0_MIN_REG	Configures the lower bound address of region 0 monitored on HP CPU0 Data bus	0x0010	R/W
ASSIST_DEBUG_CORE_0_AREA_DRAM0_0_MAX_REG	Configures the upper bound address of region 0 monitored on HP CPU0 Data bus	0x0014	R/W
ASSIST_DEBUG_CORE_0_AREA_DRAM0_1_MIN_REG	Configures the lower bound address of region 1 monitored on HP CPU0 Data bus	0x0018	R/W

Name	Description	Address	Access
ASSIST_DEBUG_CORE_0_AREA_DRAM0_1_MAX_REG	Configures the upper bound address of region 1 monitored on HP CPU0 Data bus	0x001C	R/W
ASSIST_DEBUG_CORE_0_AREA_PIF_0_MIN_REG	Configures the lower bound address of region 0 monitored on HP CPU0 Peripheral bus	0x0020	R/W
ASSIST_DEBUG_CORE_0_AREA_PIF_0_MAX_REG	Configures the upper bound address of region 0 monitored on HP CPU0 Peripheral bus	0x0024	R/W
ASSIST_DEBUG_CORE_0_AREA_PIF_1_MIN_REG	Configures the lower bound address of region 1 monitored on HP CPU0 Peripheral bus	0x0028	R/W
ASSIST_DEBUG_CORE_0_AREA_PIF_1_MAX_REG	Configures the upper bound address of region 1 monitored on HP CPU0 Peripheral bus	0x002C	R/W
ASSIST_DEBUG_CORE_0_AREA_PC_REG	Represents the PC status under HP CPU0 region monitoring	0x0030	RO
ASSIST_DEBUG_CORE_0_AREA_SP_REG	Represents the SP status under HP CPU0 region monitoring	0x0034	RO
ASSIST_DEBUG_CORE_0_SP_MIN_REG	Configures the lower bound address of the HP CPU0 SP monitored region	0x0038	R/W
ASSIST_DEBUG_CORE_0_SP_MAX_REG	Configures the upper bound address of the HP CPU0 SP monitored region	0x003C	R/W
ASSIST_DEBUG_CORE_0_SP_PC_REG	Represents the PC status under HP CPU0 SP monitoring	0x0040	RO
ASSIST_DEBUG_CORE_1_MONTR_ENA_REG	Configures whether to enable HP CPU1 monitoring	0x0080	R/W
ASSIST_DEBUG_CORE_1_AREA_DRAM0_0_MIN_REG	Configures the lower bound address of region 0 monitored on HP CPU1 Data bus	0x0090	R/W
ASSIST_DEBUG_CORE_1_AREA_DRAM0_0_MAX_REG	Configures the upper bound address of region 0 monitored on HP CPU1 Data bus	0x0094	R/W
ASSIST_DEBUG_CORE_1_AREA_DRAM0_1_MIN_REG	Configures the lower bound address of region 1 monitored on HP CPU1 Data bus	0x0098	R/W
ASSIST_DEBUG_CORE_1_AREA_DRAM0_1_MAX_REG	Configures the upper bound address of region 1 monitored on HP CPU1 Data bus	0x009C	R/W
ASSIST_DEBUG_CORE_1_AREA_PIF_0_MIN_REG	Configures the lower bound address of region 0 monitored on HP CPU1 Peripheral bus	0x00A0	R/W

Name	Description	Address	Access
ASSIST_DEBUG_CORE_1_AREA_PIF_0_MAX_REG	Configures the upper bound address of region 0 monitored on HP CPU1 Peripheral bus	0x00A4	R/W
ASSIST_DEBUG_CORE_1_AREA_PIF_1_MIN_REG	Configures the lower bound address of region 1 monitored on HP CPU1 Peripheral bus	0x00A8	R/W
ASSIST_DEBUG_CORE_1_AREA_PIF_1_MAX_REG	Configures the upper bound address of region 1 monitored on HP CPU1 Peripheral bus	0x00AC	R/W
ASSIST_DEBUG_CORE_1_AREA_PC_REG	Represents the PC status under HP CPU1 region monitoring	0x00B0	RO
ASSIST_DEBUG_CORE_1_AREA_SP_REG	Represents the SP status under HP CPU1 region monitoring	0x00B4	RO
ASSIST_DEBUG_CORE_1_SP_MIN_REG	Configures the lower bound address of the HP CPU1 SP monitored region	0x00B8	R/W
ASSIST_DEBUG_CORE_1_SP_MAX_REG	Configures the upper bound address of the HP CPU1 SP monitored region	0x00BC	R/W
ASSIST_DEBUG_CORE_1_SP_PC_REG	Represents the PC status under HP CPU1 SP monitoring	0x00C0	RO
Interrupt configuration registers			
ASSIST_DEBUG_CORE_0_INTR_RAW_REG	HP CPU0 raw interrupt status register	0x0004	RO
ASSIST_DEBUG_CORE_0_INTR_ENA_REG	HP CPU0 interrupt enable register	0x0008	R/W
ASSIST_DEBUG_CORE_0_INTR_CLR_REG	HP CPU0 interrupt clear register	0x000C	WT
ASSIST_DEBUG_CORE_1_INTR_RAW_REG	HP CPU1 raw interrupt status register	0x0084	RO
ASSIST_DEBUG_CORE_1_INTR_ENA_REG	HP CPU1 interrupt enable register	0x0088	R/W
ASSIST_DEBUG_CORE_1_INTR_CLR_REG	HP CPU1 interrupt clear register	0x008C	WT
PC logging configuration register			
ASSIST_DEBUG_CORE_0_RCD_EN_REG	HP CPU0 PC logging enable register	0x0044	R/W
ASSIST_DEBUG_CORE_1_RCD_EN_REG	HP CPU1 PC logging enable register	0x00C4	R/W
PC logging status registers			
ASSIST_DEBUG_CORE_0_RCD_PDEBUGPC_REG	HP CPU0 PC logging register	0x0048	RO
ASSIST_DEBUG_CORE_0_RCD_PDEBUGSP_REG	HP CPU0 SP logging register	0x004C	RO
ASSIST_DEBUG_CORE_1_RCD_PDEBUGPC_REG	HP CPU1 PC logging register	0x00C8	RO
ASSIST_DEBUG_CORE_1_RCD_PDEBUGSP_REG	HP CPU1 SP logging register	0x00CC	RO
CPU status registers			
ASSIST_DEBUG_CORE_0_LASTPC_BEFORE_EXCEPTION_REG	PC of the last command before HP CPU0 enters exception	0x0070	RO

Name	Description	Address	Access
ASSIST_DEBUG_CORE_0_DEBUG_MODE_REG	HP CPU0 debug mode status register	0x0074	RO
ASSIST_DEBUG_CORE_1_LASTPC_BEFORE_EXCEPTION_REG	PC of the last command before HP CPU1 enters exception	0x00F0	RO
ASSIST_DEBUG_CORE_1_DEBUG_MODE_REG	HP CPU1 debug mode status register	0x00F4	RO
Clock control register			
ASSIST_DEBUG_CLOCK_GATE_REG	Clock control register	0x0108	R/W
Version control register			
ASSIST_DEBUG_DATE_REG	Version control register	0x03FC	R/W

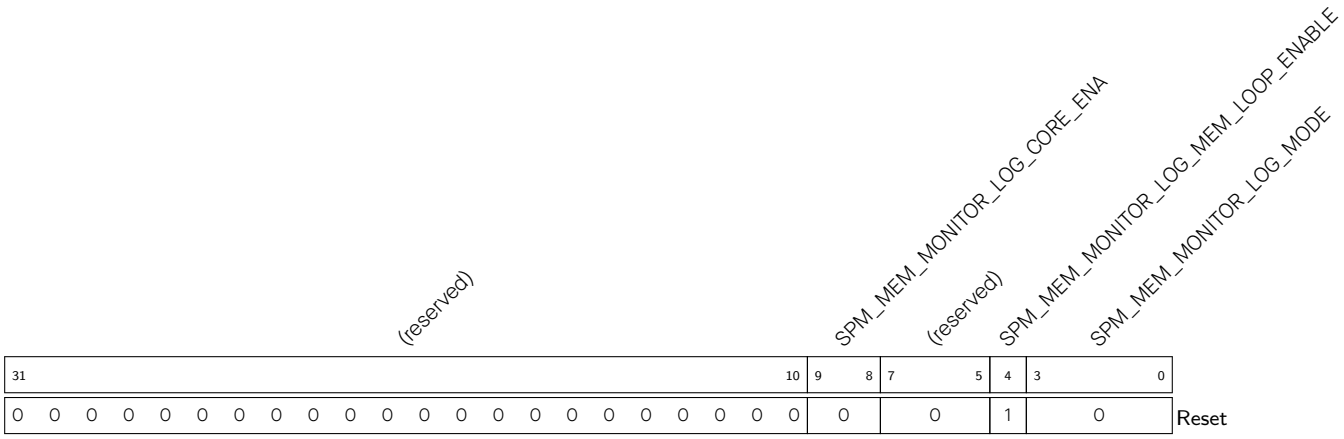
22.7 Registers

The addresses of **HP CPU bus logging configuration registers** (see 22.6.1) in this section are relative to the **SPM Monitor** base address. The addresses of **DMA bus logging configuration registers** (see 22.6.2) in this section are relative to the **L2MEM Monitor** base address. The addresses of other registers (see 22.7.3) are relative to the **Bus Monitor** base address. All base addresses are provided in Table 9.3-2 in Chapter 9 *System and Memory*.

For how to program reserved fields, please refer to Section *Programming Reserved Register Field*.

22.7.1 HP CPU Bus Logging Configuration Registers

Register 22.1. SPM_MEM_MONITOR_LOG_SETTING_REG (0x0000)



SPM_MEM_MONITOR_LOG_MODE Configures monitoring modes.

bit[0]: Configures write monitoring.
0: Disable
1: Enable
bit[1]: Configures word monitoring.
0: Disable
1: Enable
bit[2]: Configures halfword monitoring.
0: Disable
1: Enable
bit[3]: Configures byte monitoring.
0: Disable
1: Enable
(R/W)

SPM_MEM_MONITOR_LOG_MEM_LOOP_ENABLE Configures the writing mode for recorded data.

1: Loop mode
0: Non-loop mode
(R/W)

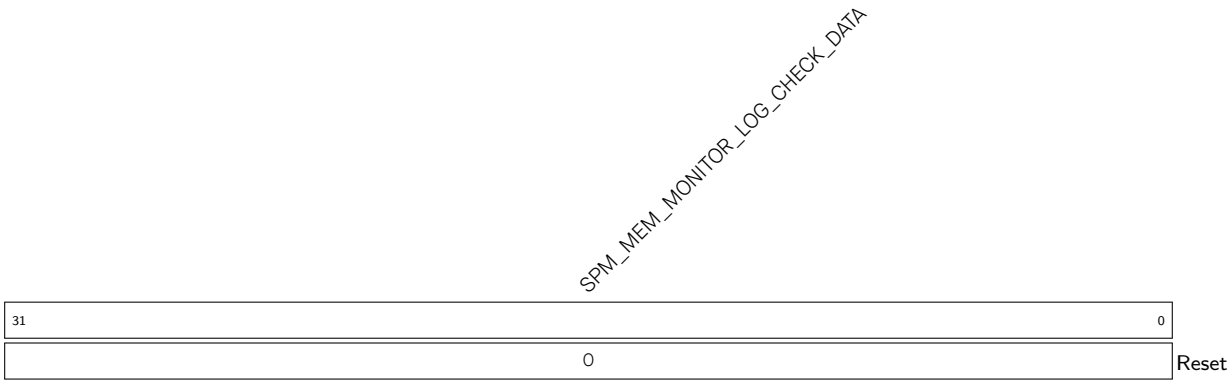
Continued on the next page...

Register 22.1. SPM_MEM_MONITOR_LOG_SETTING_REG (0x0000)

Continued from the previous page...

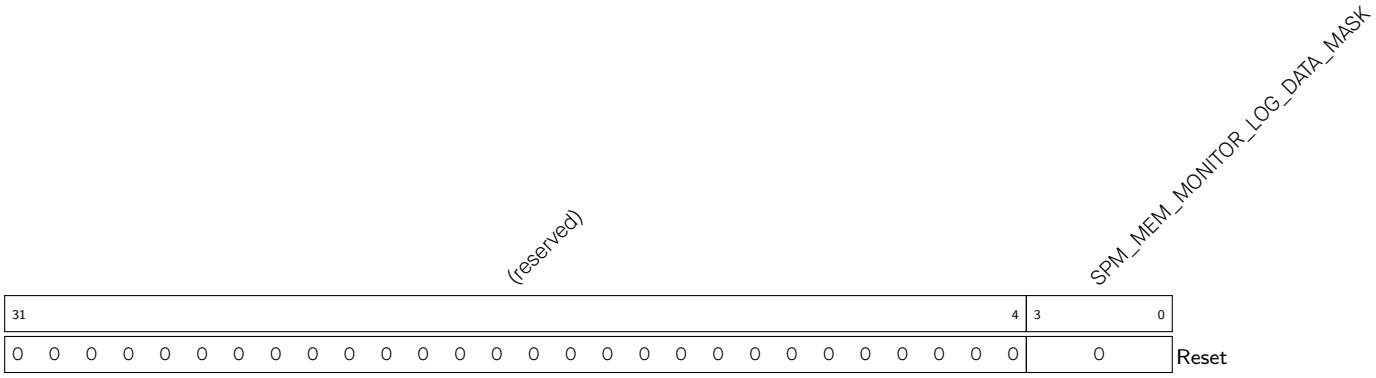
SPM_MEM_MONITOR_LOG_CORE_ENA Configures whether to enable HP CPU0 or HP CPU1 bus access logging.
bit[0]: Configures whether to enable HP CPU0 bus access logging.
0: Disable
1: Enable
bit[1]: Configures whether to enable HP CPU1 bus access logging.
0: Disable
1: Enable
(R/W)

Register 22.2. SPM_MEM_MONITOR_LOG_CHECK_DATA_REG (0x0008)



SPM_MEM_MONITOR_LOG_CHECK_DATA Configures the data to be monitored during bus accessing. (R/W)

Register 22.3. SPM_MEM_MONITOR_LOG_DATA_MASK_REG (0x000C)



SPM_MEM_MONITOR_LOG_DATA_MASK Configures which byte(s) in SPM_MEM_MONITOR_LOG_CHECK_DATA_REG to mask.

bit[0]: Configures whether to mask the least significant byte of SPM_MEM_MONITOR_LOG_CHECK_DATA_REG.

0: Not mask

1: Mask

bit[1]: Configures whether to mask the second least significant byte of SPM_MEM_MONITOR_LOG_CHECK_DATA_REG.

0: Not mask

1: Mask

bit[2]: Configures whether to mask the second most significant byte of SPM_MEM_MONITOR_LOG_CHECK_DATA_REG.

0: Not mask

1: Mask

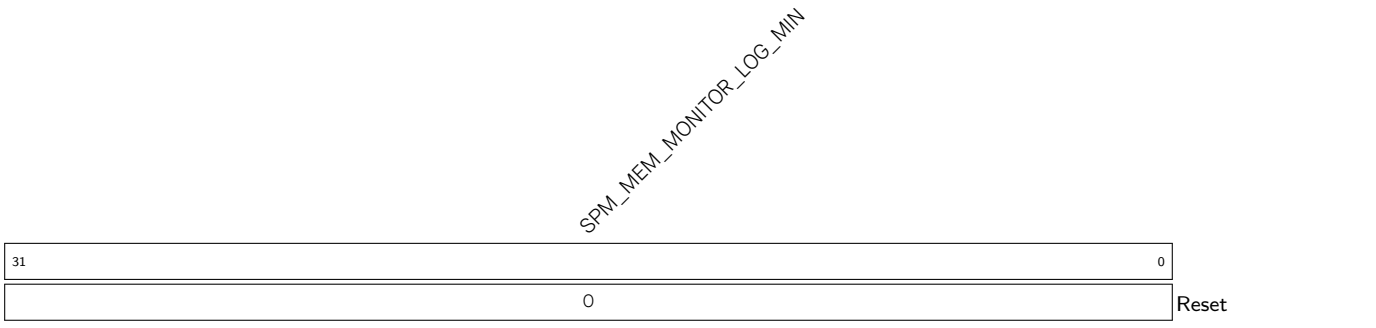
bit[3]: Configures whether to mask the most significant byte of SPM_MEM_MONITOR_LOG_CHECK_DATA_REG.

0: Not mask

1: Mask

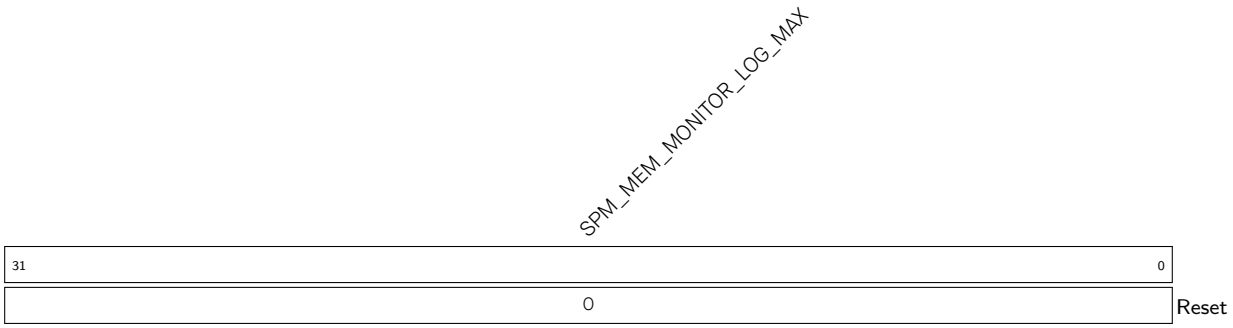
(R/W)

Register 22.4. SPM_MEM_MONITOR_LOG_MIN_REG (0x0010)



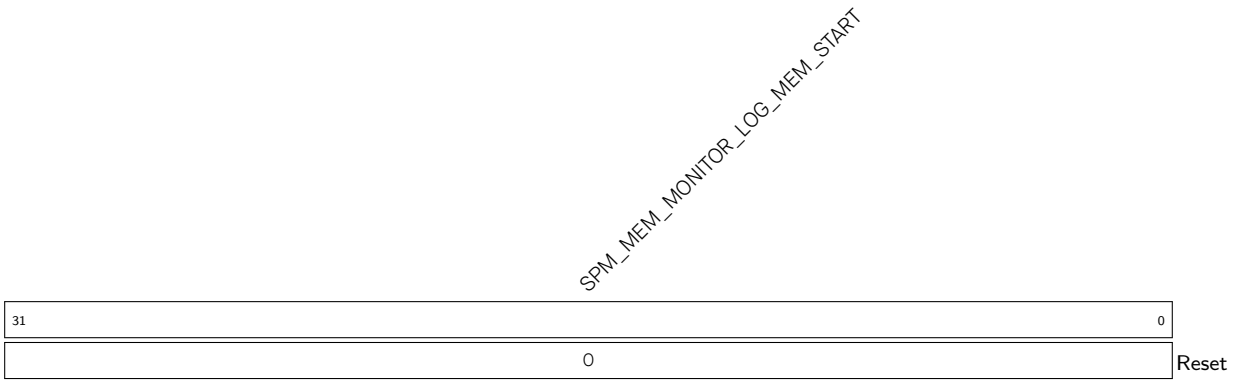
SPM_MEM_MONITOR_LOG_MIN Configures the lower bound address of the monitored address space. (R/W)

Register 22.5. SPM_MEM_MONITOR_LOG_MAX_REG (0x0014)



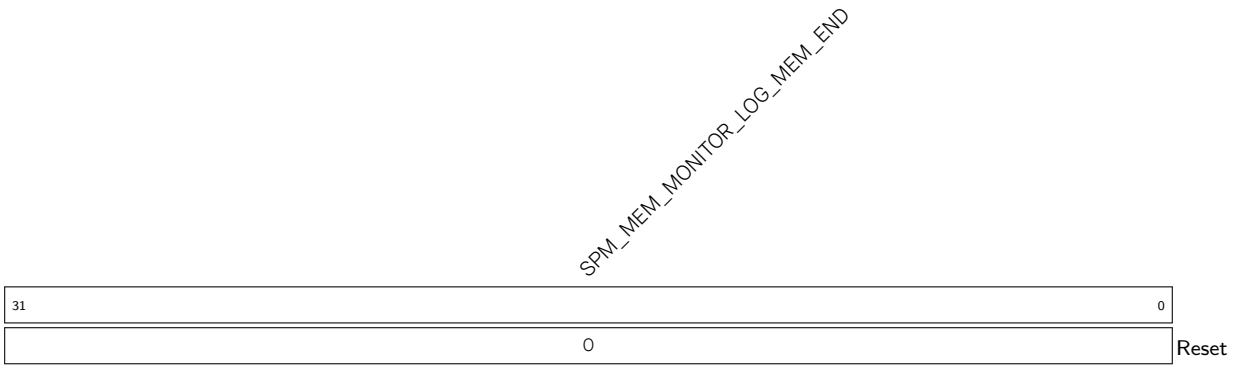
SPM_MEM_MONITOR_LOG_MAX Configures the upper bound address of the monitored address space. (R/W)

Register 22.6. SPM_MEM_MONITOR_LOG_MEM_START_REG (0x0018)



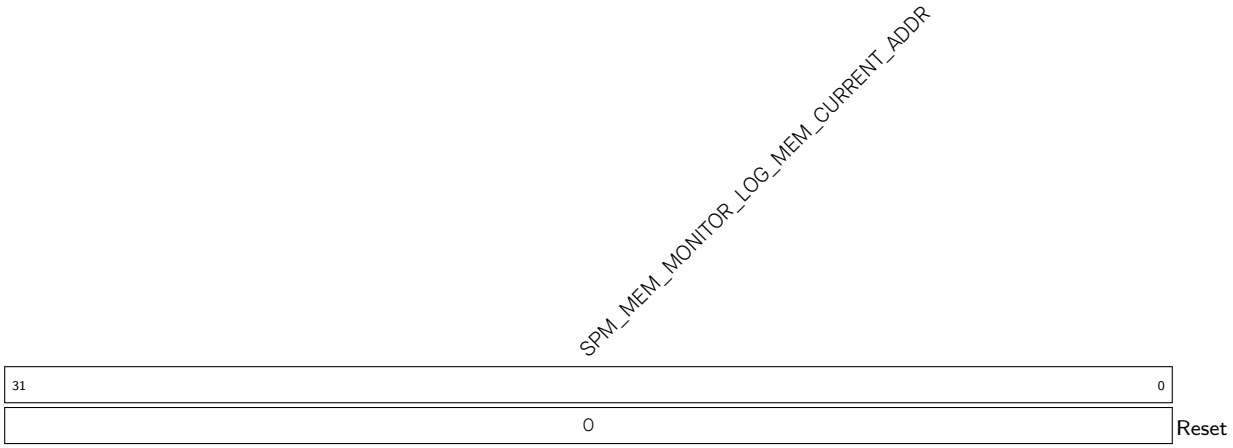
SPM_MEM_MONITOR_LOG_MEM_START Configures the starting address of the storage space for recorded data. (R/W)

Register 22.7. SPM_MEM_MONITOR_LOG_MEM_END_REG (0x001C)



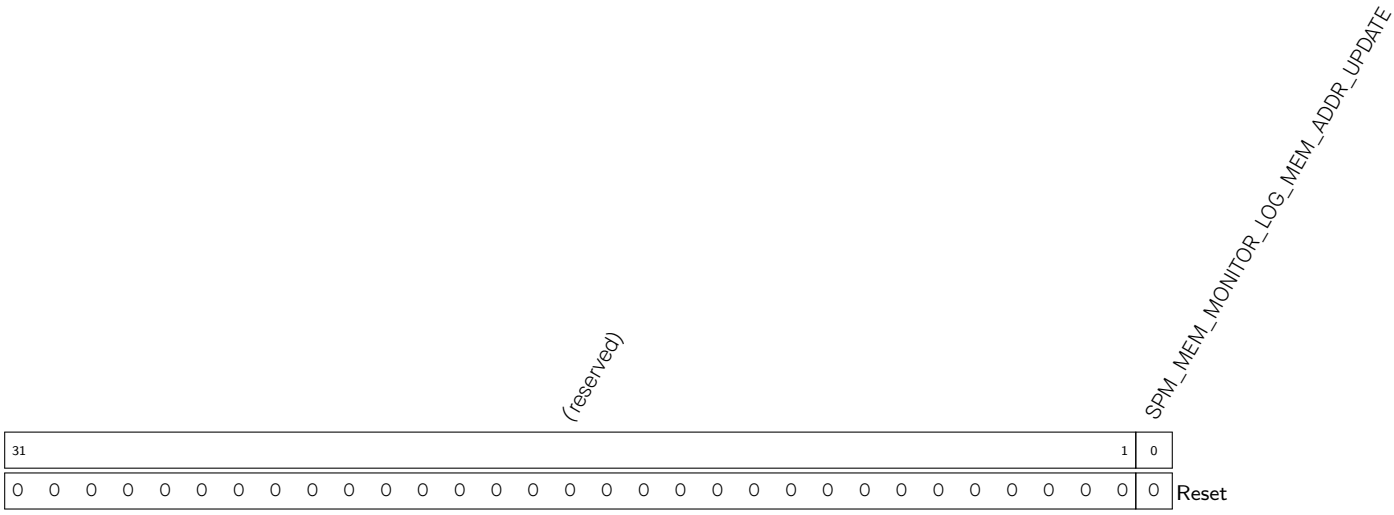
SPM_MEM_MONITOR_LOG_MEM_END Configures the ending address of the storage space for recorded data. (R/W)

Register 22.8. SPM_MEM_MONITOR_LOG_MEM_CURRENT_ADDR_REG (0x0020)



SPM_MEM_MONITOR_LOG_MEM_CURRENT_ADDR Represents the address of the next write. (RO)

Register 22.9. SPM_MEM_MONITOR_LOG_MEM_ADDR_UPDATE_REG (0x0024)



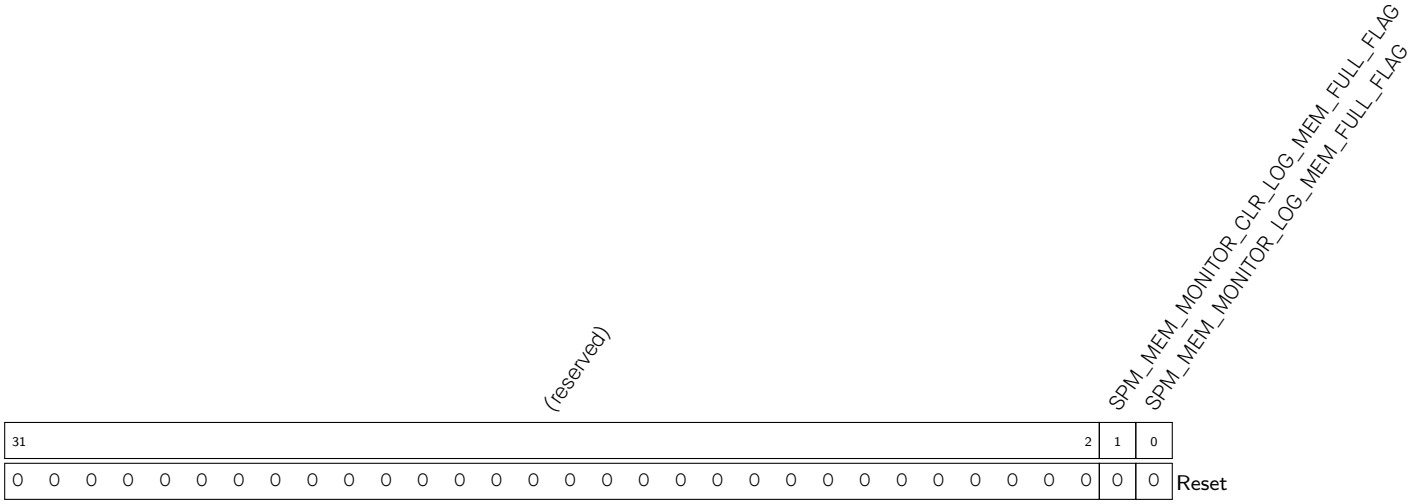
SPM_MEM_MONITOR_LOG_MEM_ADDR_UPDATE Configures whether to update the value in SPM_MEM_MONITOR_LOG_MEM_START_REG to SPM_MEM_MONITOR_LOG_MEM_CURRENT_ADDR_REG.

1: Update

0: Not update

(WT)

Register 22.10. SPM_MEM_MONITOR_LOG_MEM_FULL_FLAG_REG (0x0028)



SPM_MEM_MONITOR_LOG_MEM_FULL_FLAG Represents whether data overflows the storage space.

0: Not overflow

1: Overflow

(RO)

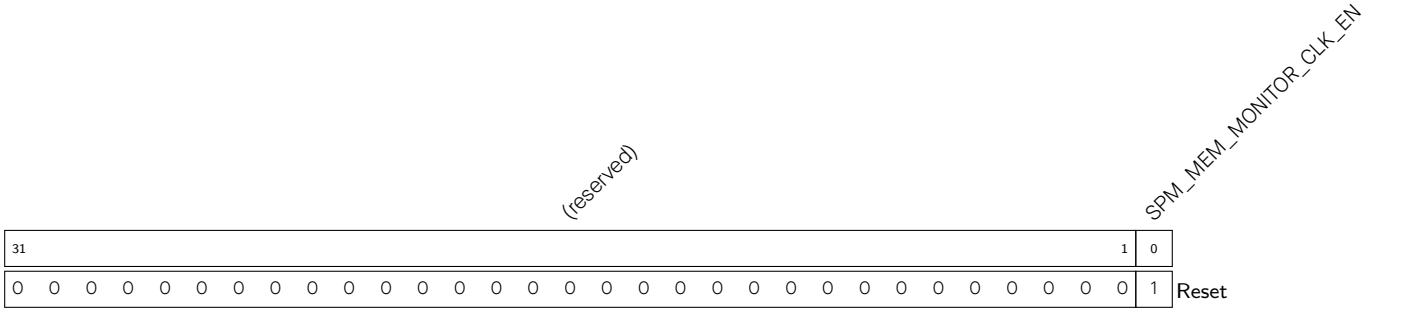
SPM_MEM_MONITOR_CLR_LOG_MEM_FULL_FLAG Configures whether to clear the SPM_MEM_MONITOR_LOG_MEM_FULL_FLAG flag bit.

0: Not clear (default)

1: Clear

(WT)

Register 22.11. SPMMEM_MONITOR_CLOCK_GATE_REG (0x002C)



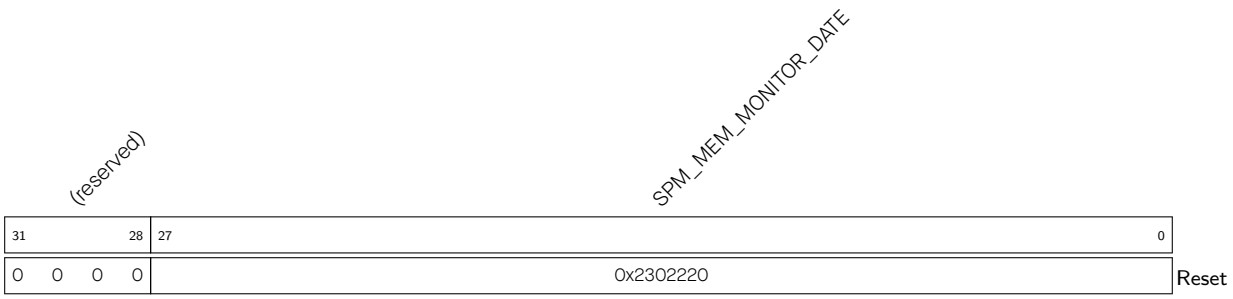
SPM_MEM_MONITOR_CLK_EN Configures whether to enable the register clock gating.

0: Disable

1: Enable

(R/W)

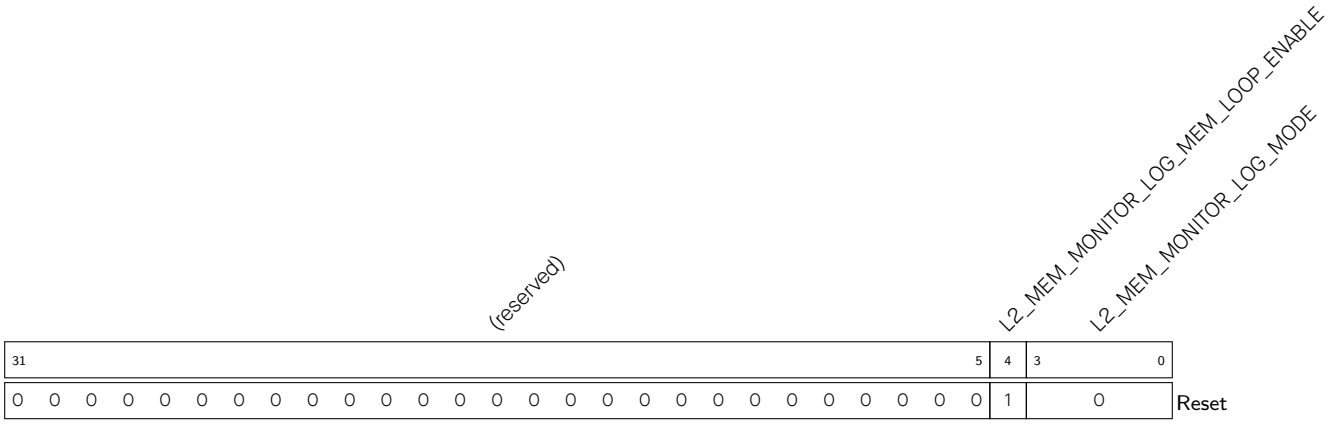
Register 22.12. SPM_MEM_MONITOR_DATE_REG (0x03FC)



SPM_MEM_MONITOR_DATE Version control register. (R/W)

22.7.2 DMA Bus Logging Configuration Registers

Register 22.13. L2_MEM_MONITOR_LOG_SETTING_REG (0x0000)



L2_MEM_MONITOR_LOG_MODE Configures monitoring modes.

bit[0]: Configures write monitoring.

0: Disable

1: Enable

bit[1]: Configures word monitoring.

0: Disable

1: Enable

bit[2]: Configures halfword monitoring.

0: Disable

1: Enable

bit[3]: Configures byte monitoring.

0: Disable

1: Enable

(R/W)

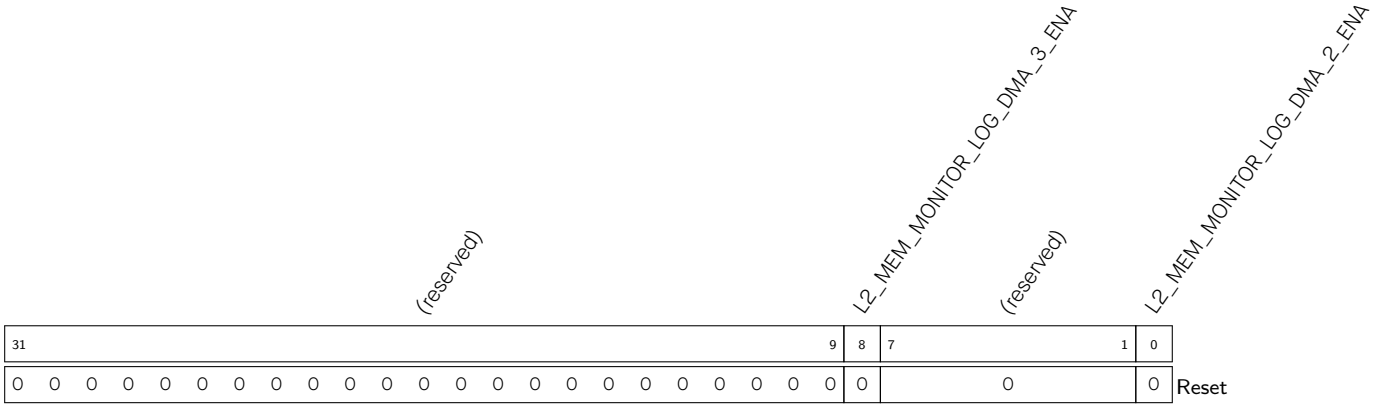
L2_MEM_MONITOR_LOG_MEM_LOOP_ENABLE Configures the writing mode for recorded data.

1: Loop mode

0: Non-loop mode

(R/W)

Register 22.14. L2_MEM_MONITOR_LOG_SETTING1_REG (0x0004)



L2_MEM_MONITOR_LOG_DMA_2_ENA Configures whether to enable DMA_2 bus access logging.

0: Disable

1: Enable

(R/W)

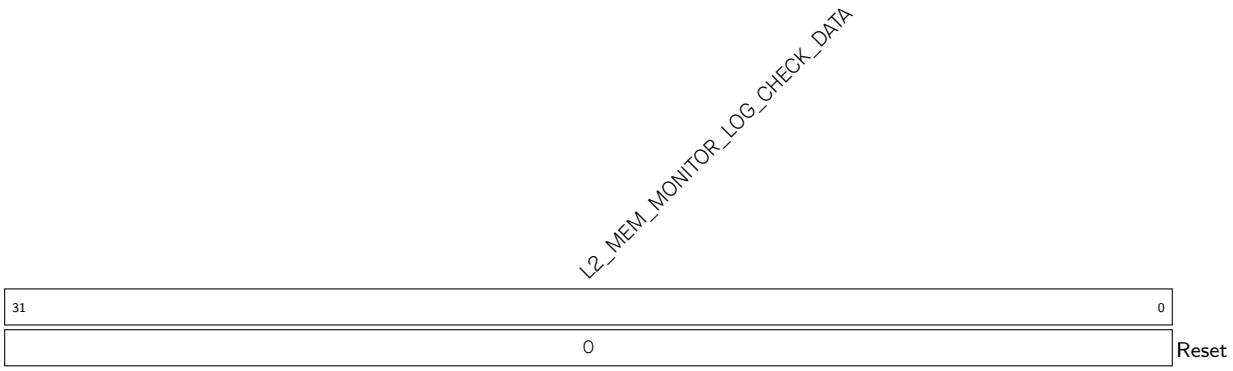
L2_MEM_MONITOR_LOG_DMA_3_ENA Configures whether to enable DMA_3 bus access logging.

0: Disable

1: Enable

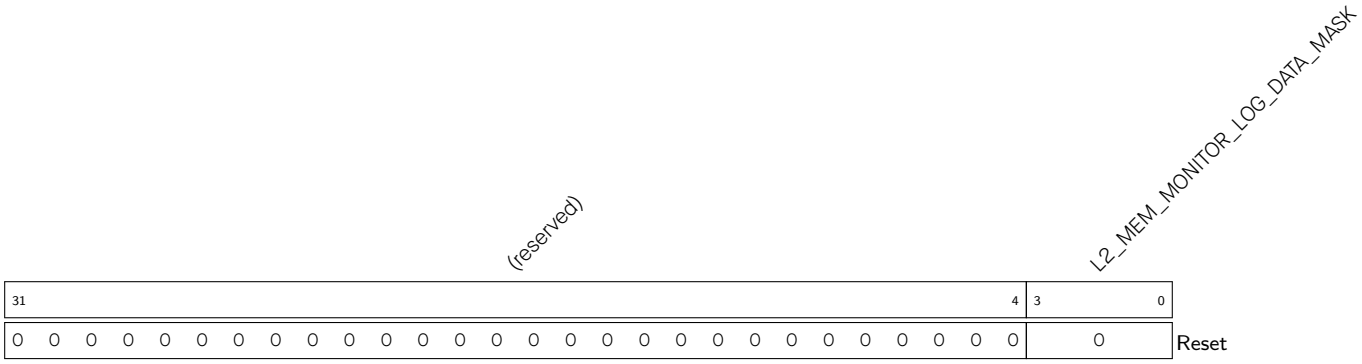
(R/W)

Register 22.15. L2_MEM_MONITOR_LOG_CHECK_DATA_REG (0x0008)



L2_MEM_MONITOR_LOG_CHECK_DATA Configures the data to be monitored during bus accessing. (R/W)

Register 22.16. L2_MEM_MONITOR_LOG_DATA_MASK_REG (0x000C)



L2_MEM_MONITOR_LOG_DATA_MASK Configures which byte(s) in L2_MEM_MONITOR_LOG_CHECK_DATA_REG to mask.

bit[0]: Configures whether to mask the least significant byte of L2_MEM_MONITOR_LOG_CHECK_DATA_REG.

0: Not mask

1: Mask

bit[1]: Configures whether to mask the second least significant byte of L2_MEM_MONITOR_LOG_CHECK_DATA_REG.

0: Not mask

1: Mask

bit[2]: Configures whether to mask the second most significant byte of L2_MEM_MONITOR_LOG_CHECK_DATA_REG.

0: Not mask

1: Mask

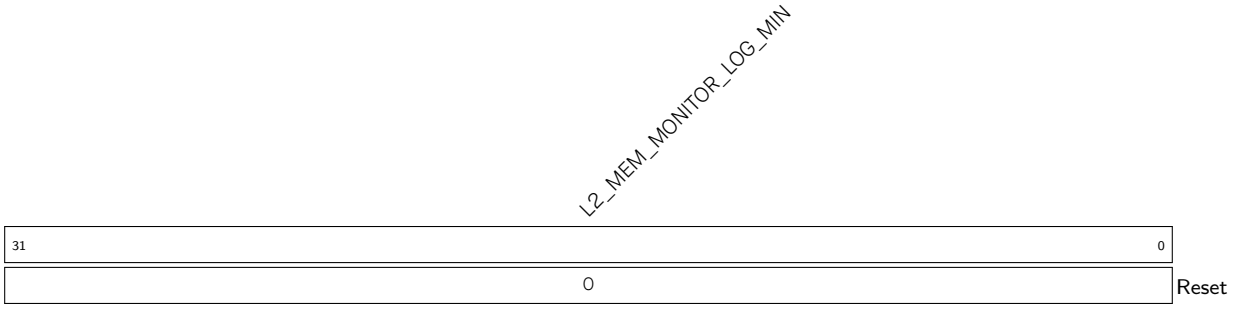
bit[3]: Configures whether to mask the most significant byte of L2_MEM_MONITOR_LOG_CHECK_DATA_REG.

0: Not mask

1: Mask

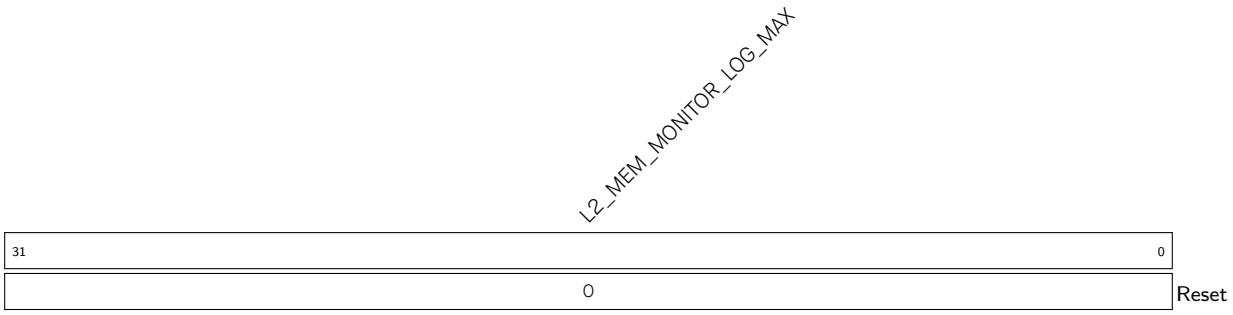
(R/W)

Register 22.17. L2_MEM_MONITOR_LOG_MIN_REG (0x0010)



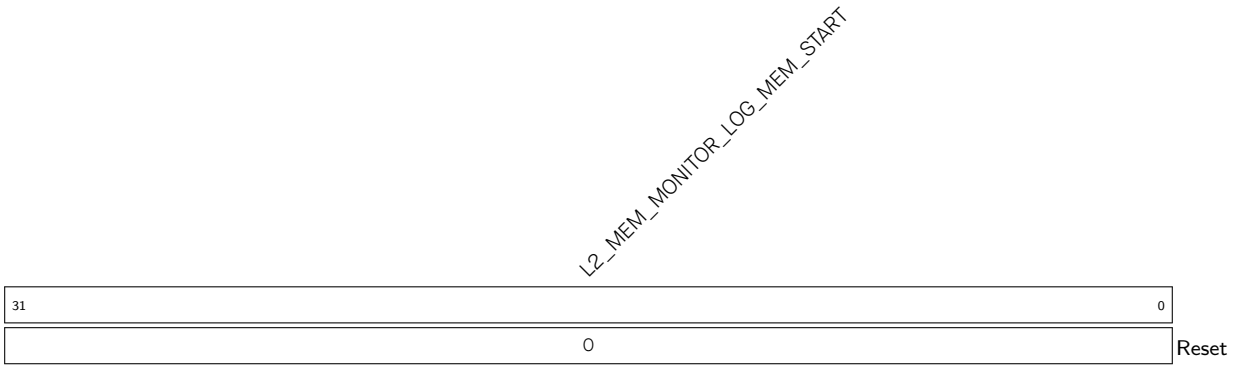
L2_MEM_MONITOR_LOG_MIN Configures the lower bound address of the monitored address space. (R/W)

Register 22.18. L2_MEM_MONITOR_LOG_MAX_REG (0x0014)



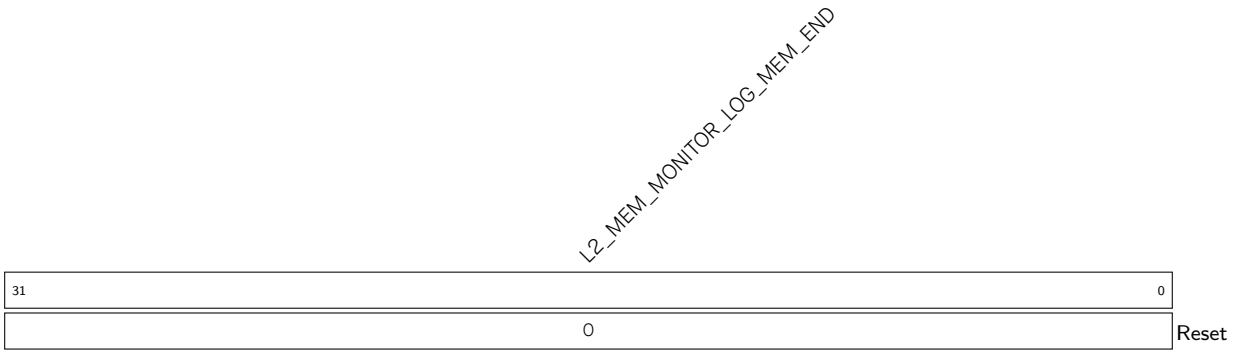
L2_MEM_MONITOR_LOG_MAX Configures the upper bound address of the monitored address space. (R/W)

Register 22.19. L2_MEM_MONITOR_LOG_MEM_START_REG (0x0018)



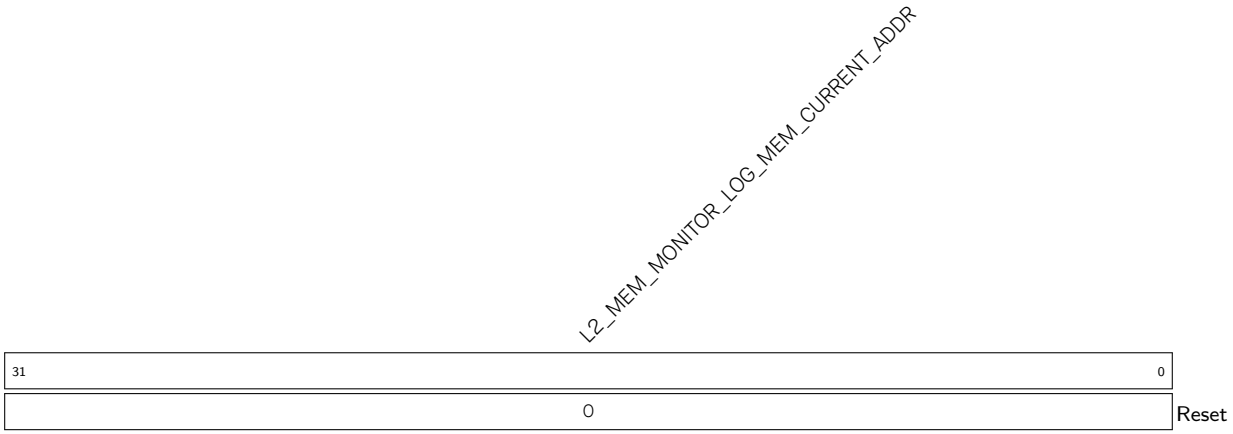
L2_MEM_MONITOR_LOG_MEM_START Configures the starting address of the storage space for recorded data. (R/W)

Register 22.20. L2_MEM_MONITOR_LOG_MEM_END_REG (0x001C)



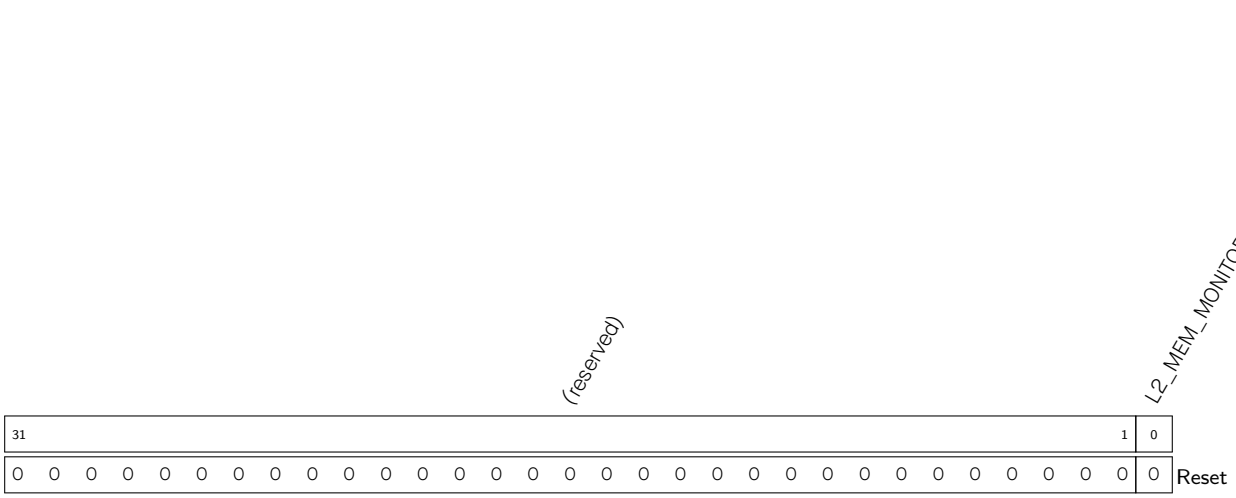
L2_MEM_MONITOR_LOG_MEM_END Configures the ending address of the storage space for recorded data. (R/W)

Register 22.21. L2_MEM_MONITOR_LOG_MEM_CURRENT_ADDR_REG (0x0020)



L2_MEM_MONITOR_LOG_MEM_CURRENT_ADDR Represents the address of the next write. (RO)

Register 22.22. L2_MEM_MONITOR_LOG_MEM_ADDR_UPDATE_REG (0x0024)



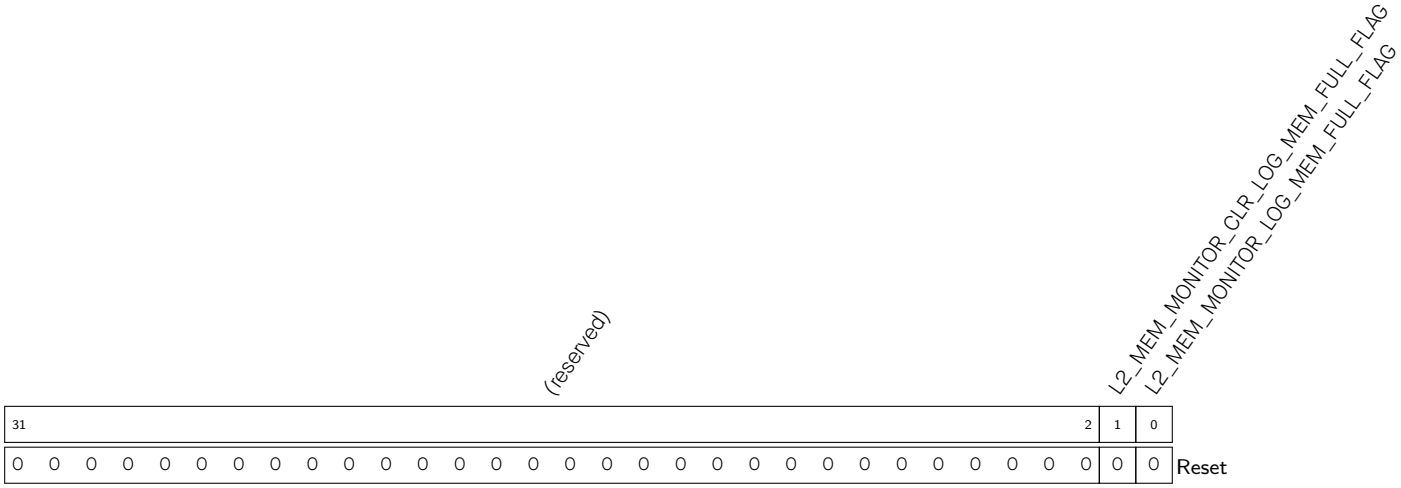
L2_MEM_MONITOR_LOG_MEM_ADDR_UPDATE Configures whether to update the value in L2_MEM_MONITOR_LOG_MEM_START_REG to L2_MEM_MONITOR_LOG_MEM_CURRENT_ADDR_REG.

1: Update

0: Not update (default)

(WT)

Register 22.23. L2_MEM_MONITOR_LOG_MEM_FULL_FLAG_REG (0x0028)



L2_MEM_MONITOR_LOG_MEM_FULL_FLAG Represents whether data overflows the storage space.

0: Not overflow

1: Overflow

(RO)

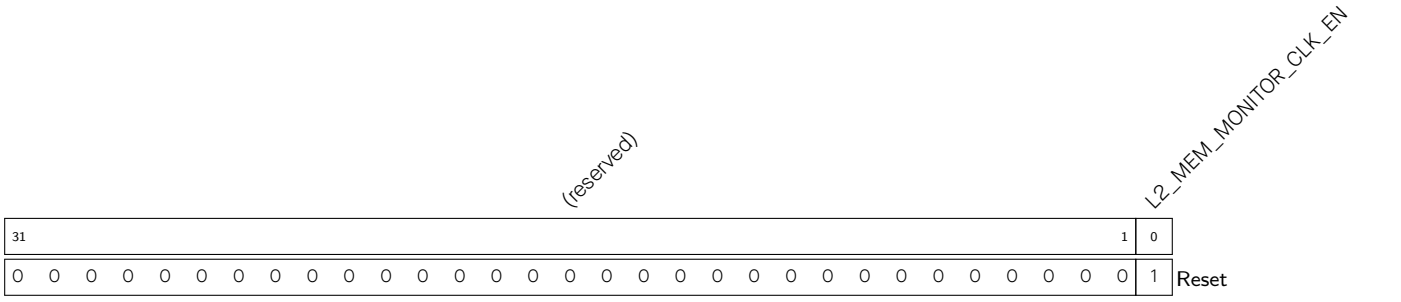
L2_MEM_MONITOR_CLR_LOG_MEM_FULL_FLAG Configures whether to clear the L2_MEM_MONITOR_LOG_MEM_FULL_FLAG flag bit.

0: Not clear (default)

1: Clear

(WT)

Register 22.24. L2_MEM_MONITOR_CLOCK_GATE_REG (0x002C)



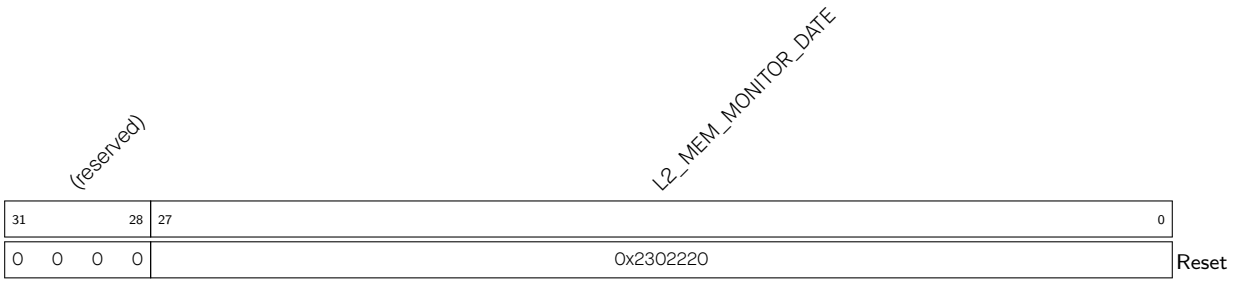
L2_MEM_MONITOR_CLK_EN Configures whether to enable the register clock gating.

0: Disable

1: Enable

(R/W)

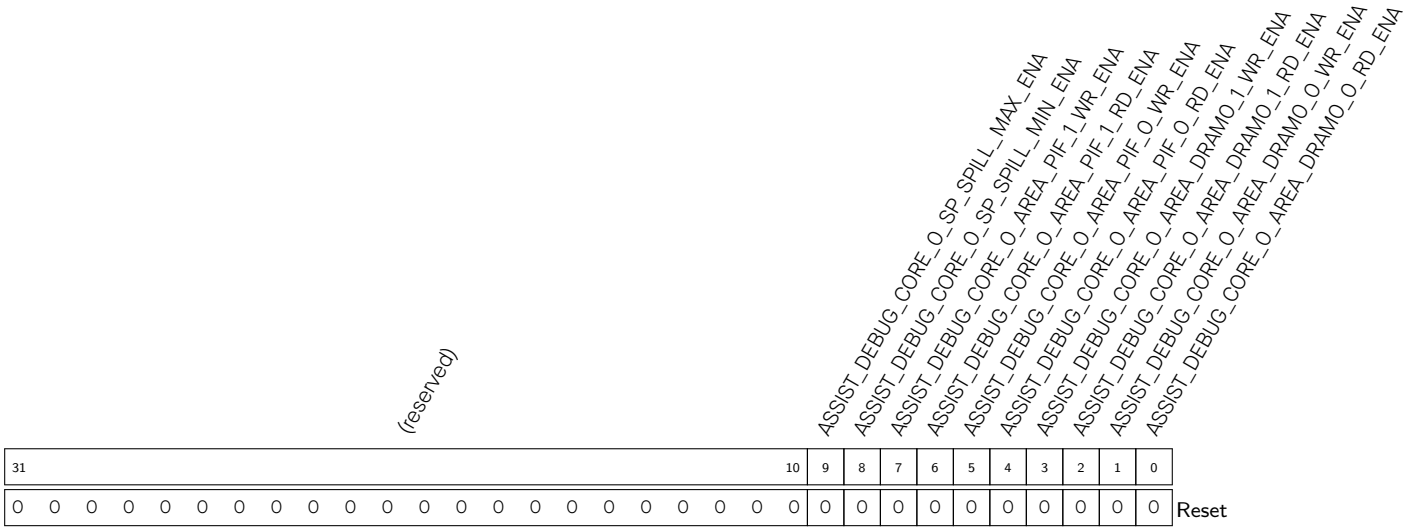
Register 22.25. L2_MEM_MONITOR_DATE_REG (0x03FC)



L2_MEM_MONITOR_DATE Version control register. (R/W)

22.7.3 Other Registers

Register 22.26. ASSIST_DEBUG_CORE_0_MONTR_ENA_REG (0x0000)



ASSIST_DEBUG_CORE_0_AREA_DRAM0_0_RD_ENA Configures whether to monitor read operations in HP CPU0 region 0 by the HP CPU0 Data bus.

0: Not monitor

1: Monitor

(R/W)

ASSIST_DEBUG_CORE_0_AREA_DRAM0_0_WR_ENA Configures whether to monitor write operations in HP CPU0 region 0 by the HP CPU0 Data bus.

0: Not monitor

1: Monitor

(R/W)

ASSIST_DEBUG_CORE_0_AREA_DRAM0_1_RD_ENA Configures whether to monitor read operations in HP CPU0 region 1 by the HP CPU0 Data bus.

0: Not monitor

1: Monitor

(R/W)

ASSIST_DEBUG_CORE_0_AREA_DRAM0_1_WR_ENA Configures whether to monitor write operations in HP CPU0 region 1 by the HP CPU0 Data bus.

0: Not monitor

1: Monitor

(R/W)

ASSIST_DEBUG_CORE_0_AREA_PIF_0_RD_ENA Configures whether to monitor read operations in region 0 by the Peripheral bus.

0: Not monitor

1: Monitor

(R/W)

Continued on the next page...

Register 22.26. ASSIST_DEBUG_CORE_0_MONTR_ENA_REG (0x0000)

Continued from the previous page...

ASSIST_DEBUG_CORE_0_AREA_PIF_0_WR_ENA Configures whether to monitor write operations in HP CPU0 region 0 by the HP CPU0 Peripheral bus.

0: Not monitor

1: Monitor

(R/W)

ASSIST_DEBUG_CORE_0_AREA_PIF_1_RD_ENA Configures whether to monitor read operations in HP CPU0 region 1 by the HP CPU0 Peripheral bus.

0: Not monitor

1: Monitor

(R/W)

ASSIST_DEBUG_CORE_0_AREA_PIF_1_WR_ENA Configures whether to monitor write operations in HP CPU0 region 1 by the HP CPU0 Peripheral bus.

0: Not monitor

1: Monitor

(R/W)

ASSIST_DEBUG_CORE_0_SP_SPILL_MIN_ENA Configures whether to monitor HP CPU0 SP going below the lower bound address of HP CPU0 SP monitored region.

0: Not monitor

1: Monitor

(R/W)

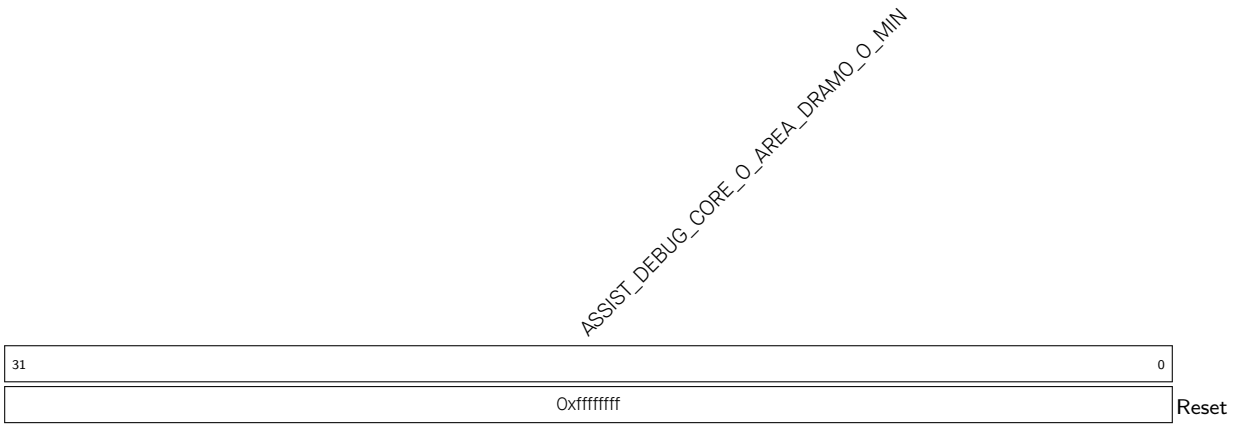
ASSIST_DEBUG_CORE_0_SP_SPILL_MAX_ENA Configures whether to monitor HP CPU0 SP going beyond the upper bound address of HP CPU0 SP monitored region.

0: Not monitor

1: Monitor

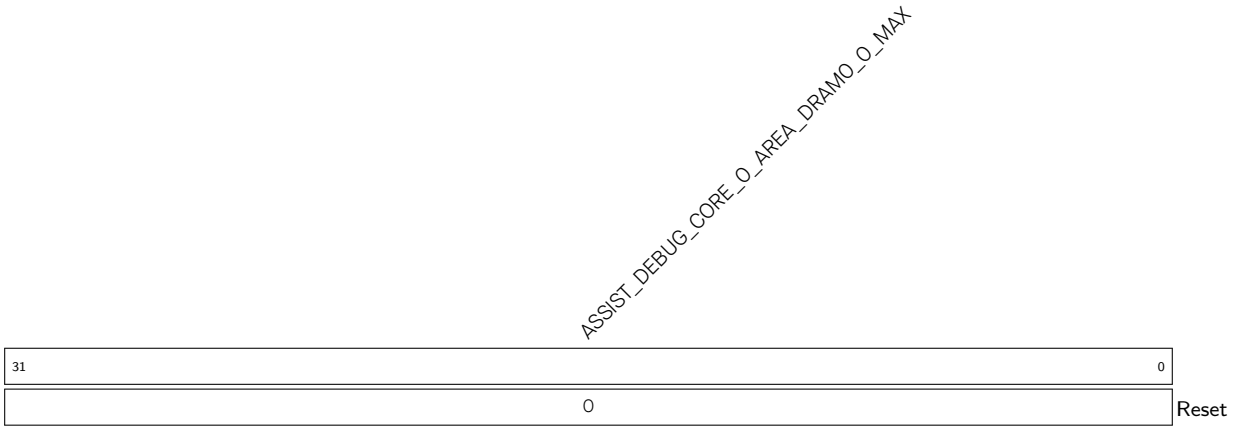
(R/W)

Register 22.27. ASSIST_DEBUG_CORE_0_AREA_DRAM0_0_MIN_REG (0x0010)



ASSIST_DEBUG_CORE_0_AREA_DRAM0_0_MIN Configures the lower bound address of HP CPU0 Data bus region 0. (R/W)

Register 22.28. ASSIST_DEBUG_CORE_0_AREA_DRAM0_0_MAX_REG (0x0014)



ASSIST_DEBUG_CORE_0_AREA_DRAM0_0_MAX Configures the upper bound address of HP CPU0 Data bus region 0. (R/W)

Register 22.29. ASSIST_DEBUG_CORE_0_AREA_DRAM0_1_MIN_REG (0x0018)



ASSIST_DEBUG_CORE_0_AREA_DRAM0_1_MIN Configures the lower bound address of HP CPU0 Data bus region 1. (R/W)

Register 22.30. ASSIST_DEBUG_CORE_0_AREA_DRAM0_1_MAX_REG (0x001C)



ASSIST_DEBUG_CORE_0_AREA_DRAM0_1_MAX Configures the upper bound address of HP CPU0 Data bus region 1. (R/W)

Register 22.31. ASSIST_DEBUG_CORE_0_AREA_PIF_0_MIN_REG (0x0020)



ASSIST_DEBUG_CORE_0_AREA_PIF_0_MIN Configures the lower bound address of HP CPU0 Peripheral bus region 0. (R/W)

Register 22.32. ASSIST_DEBUG_CORE_0_AREA_PIF_0_MAX_REG (0x0024)



ASSIST_DEBUG_CORE_0_AREA_PIF_0_MAX Configures the upper bound address of HP CPU0 Peripheral bus region 0. (R/W)

Register 22.33. ASSIST_DEBUG_CORE_0_AREA_PIF_1_MIN_REG (0x0028)



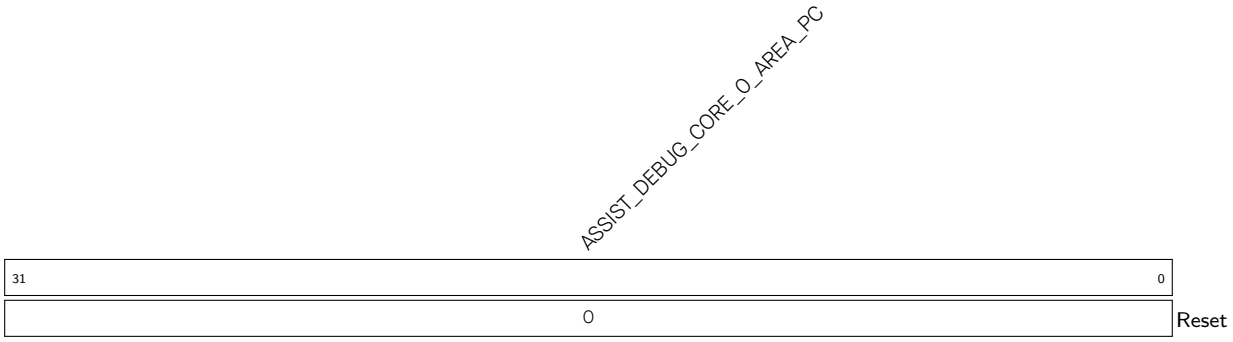
ASSIST_DEBUG_CORE_0_AREA_PIF_1_MIN Configures the lower bound address of HP CPU0 Peripheral bus region 1. (R/W)

Register 22.34. ASSIST_DEBUG_CORE_0_AREA_PIF_1_MAX_REG (0x002C)



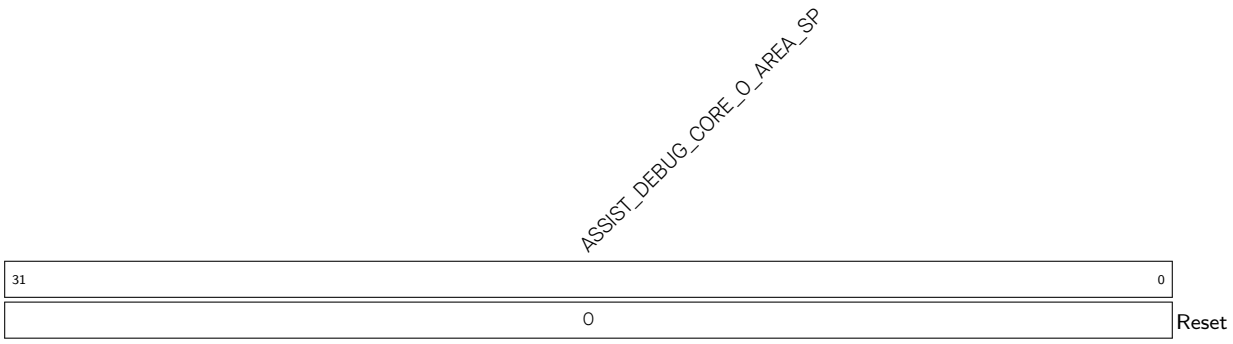
ASSIST_DEBUG_CORE_0_AREA_PIF_1_MAX Configures the upper bound address of HP CPU0 Peripheral bus region 1. (R/W)

Register 22.35. ASSIST_DEBUG_CORE_0_AREA_PC_REG (0x0030)



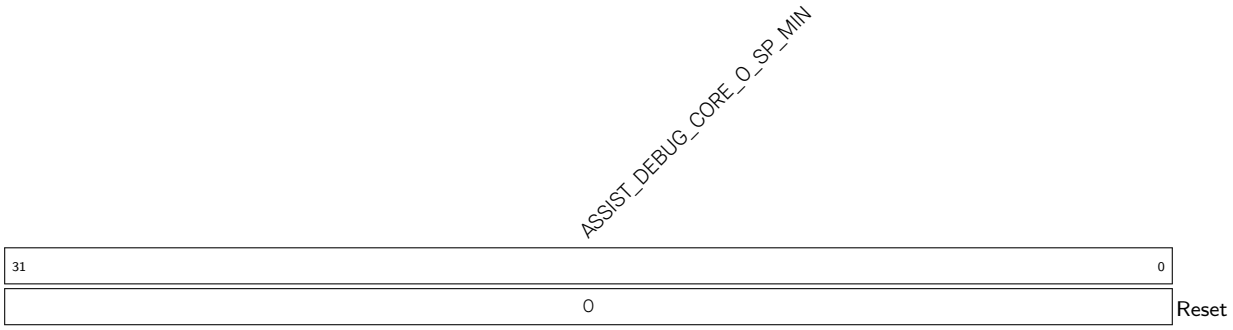
ASSIST_DEBUG_CORE_0_AREA_PC Represents the HP CPU0 PC value when an interrupt is triggered during region monitoring. (RO)

Register 22.36. ASSIST_DEBUG_CORE_0_AREA_SP_REG (0x0034)



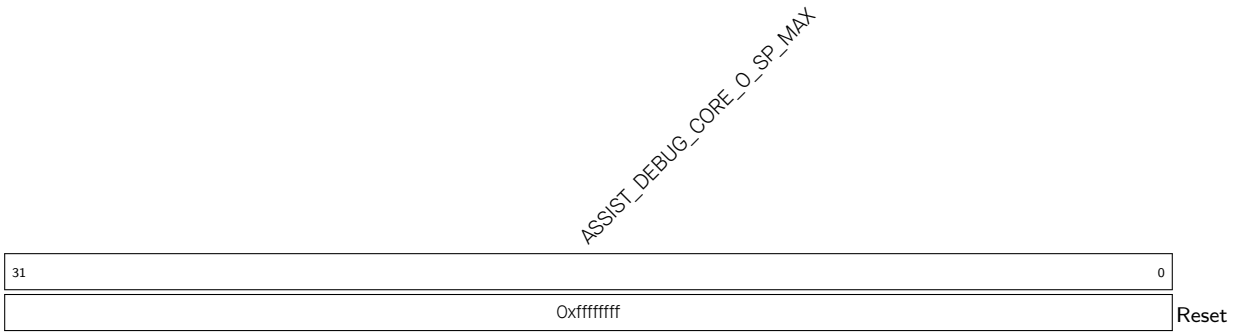
ASSIST_DEBUG_CORE_0_AREA_SP Represents the HP CPU0 SP value when an interrupt is triggered during region monitoring. (RO)

Register 22.37. ASSIST_DEBUG_CORE_0_SP_MIN_REG (0x0038)



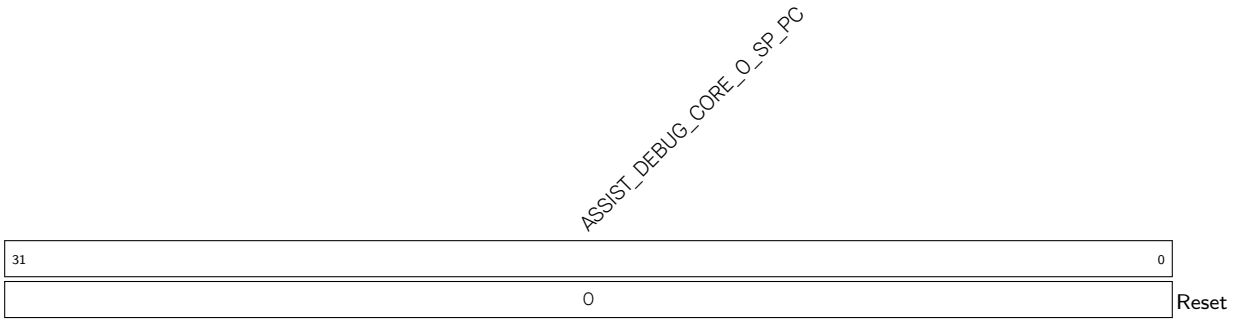
ASSIST_DEBUG_CORE_0_SP_MIN Configures the lower bound address of the HP CPU0 SP monitored region. (R/W)

Register 22.38. ASSIST_DEBUG_CORE_0_SP_MAX_REG (0x003C)



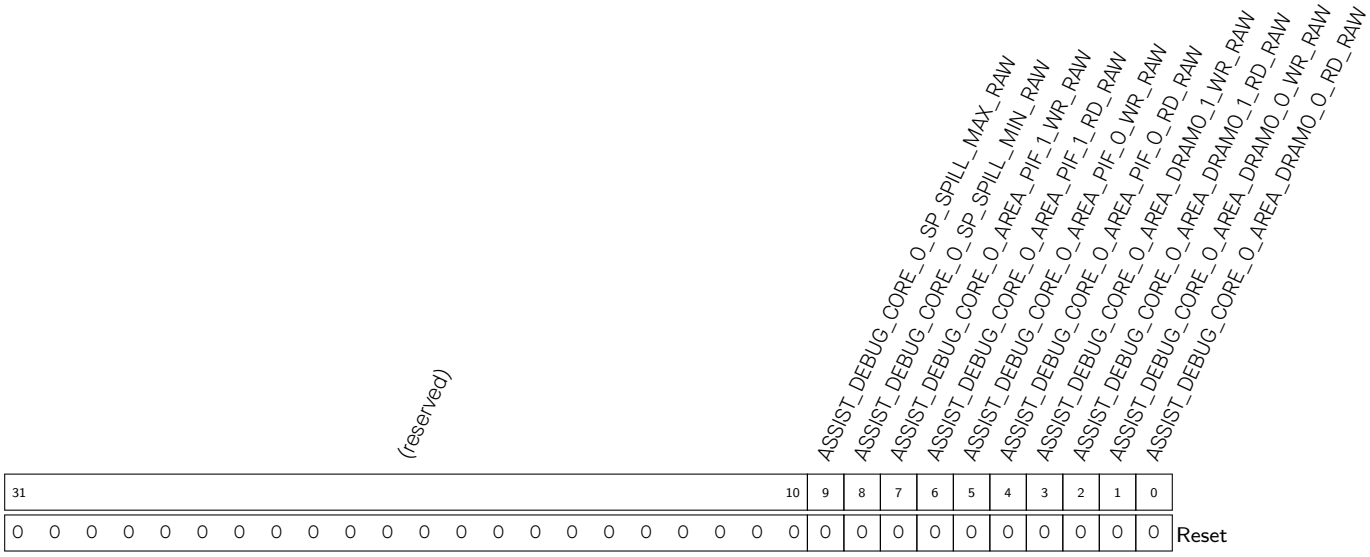
ASSIST_DEBUG_CORE_0_SP_MAX Configures the upper bound address of the HP CPU0 SP monitored region. (R/W)

Register 22.39. ASSIST_DEBUG_CORE_0_SP_PC_REG (0x0040)



ASSIST_DEBUG_CORE_0_SP_PC Represents the HP CPU0 PC value during SP monitoring. (RO)

Register 22.40. ASSIST_DEBUG_CORE_0_INTR_RAW_REG (0x0004)



ASSIST_DEBUG_CORE_0_AREA_DRAM0_0_RD_RAW The raw interrupt status of ASSIST_DEBUG_CORE_0_AREA_DRAM0_0_RD_INT. (RO)

ASSIST_DEBUG_CORE_0_AREA_DRAM0_0_WR_RAW The raw interrupt status of ASSIST_DEBUG_CORE_0_AREA_DRAM0_0_WR_INT. (RO)

ASSIST_DEBUG_CORE_0_AREA_DRAM0_1_RD_RAW The raw interrupt status of ASSIST_DEBUG_CORE_0_AREA_DRAM0_1_RD_INT. (RO)

ASSIST_DEBUG_CORE_0_AREA_DRAM0_1_WR_RAW The raw interrupt status of ASSIST_DEBUG_CORE_0_AREA_DRAM0_1_WR_INT. (RO)

ASSIST_DEBUG_CORE_0_AREA_PIF_0_RD_RAW The raw interrupt status of ASSIST_DEBUG_CORE_0_AREA_PIF_0_RD_INT. (RO)

ASSIST_DEBUG_CORE_0_AREA_PIF_0_WR_RAW The raw interrupt status of ASSIST_DEBUG_CORE_0_AREA_PIF_0_WR_INT. (RO)

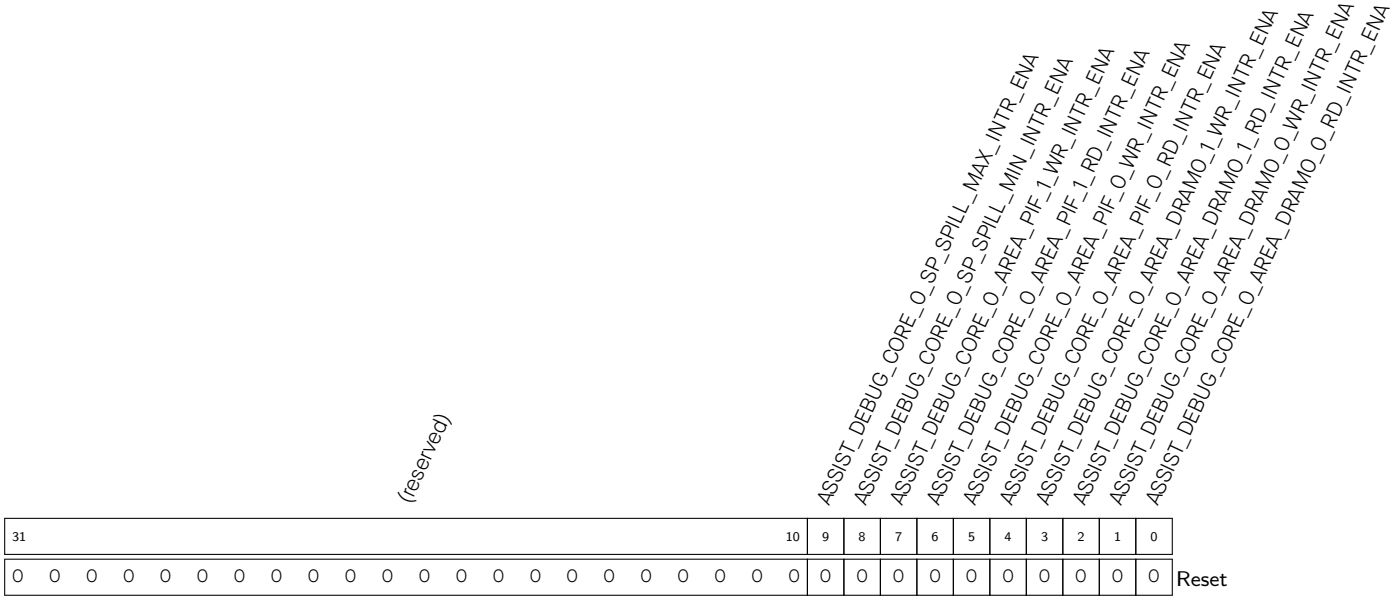
ASSIST_DEBUG_CORE_0_AREA_PIF_1_RD_RAW The raw interrupt status of ASSIST_DEBUG_CORE_0_AREA_PIF_1_RD_INT. (RO)

ASSIST_DEBUG_CORE_0_AREA_PIF_1_WR_RAW The raw interrupt status of ASSIST_DEBUG_CORE_0_AREA_PIF_1_WR_INT. (RO)

ASSIST_DEBUG_CORE_0_SP_SPILL_MIN_RAW The raw interrupt status of ASSIST_DEBUG_CORE_0_SP_SPILL_MIN_INT. (RO)

ASSIST_DEBUG_CORE_0_SP_SPILL_MAX_RAW The raw interrupt status of ASSIST_DEBUG_CORE_0_SP_SPILL_MAX_INT. (RO)

Register 22.41. ASSIST_DEBUG_CORE_0_INTR_ENA_REG (0x0008)



ASSIST_DEBUG_CORE_0_AREA_DRAM0_0_RD_INTR_ENA Write 1 to enable ASSIST_DEBUG_CORE_0_AREA_DRAM0_0_RD_INT. (R/W)

ASSIST_DEBUG_CORE_0_AREA_DRAM0_0_WR_INTR_ENA Write 1 to enable ASSIST_DEBUG_CORE_0_AREA_DRAM0_0_WR_INT. (R/W)

ASSIST_DEBUG_CORE_0_AREA_DRAM0_1_RD_INTR_ENA Write 1 to enable ASSIST_DEBUG_CORE_0_AREA_DRAM0_1_RD_INT. (R/W)

ASSIST_DEBUG_CORE_0_AREA_DRAM0_1_WR_INTR_ENA Write 1 to enable ASSIST_DEBUG_CORE_0_AREA_DRAM0_1_WR_INT. (R/W)

ASSIST_DEBUG_CORE_0_AREA_PIF_0_RD_INTR_ENA Write 1 to enable ASSIST_DEBUG_CORE_0_AREA_PIF_0_RD_INT. (R/W)

ASSIST_DEBUG_CORE_0_AREA_PIF_0_WR_INTR_ENA Write 1 to enable ASSIST_DEBUG_CORE_0_AREA_PIF_0_WR_INT. (R/W)

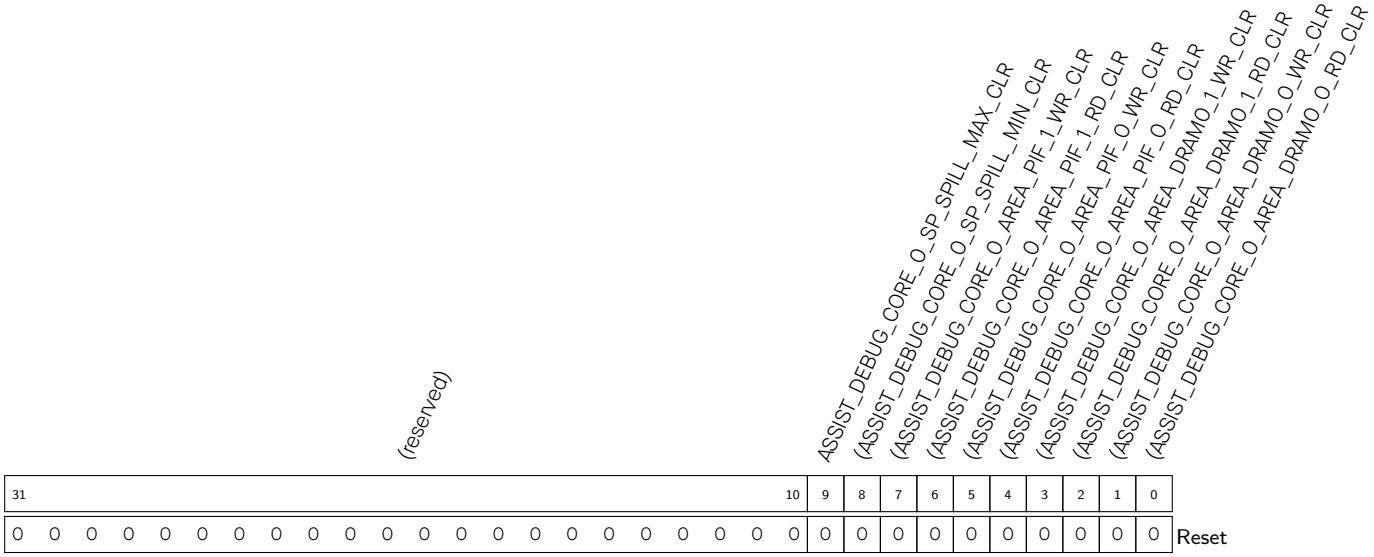
ASSIST_DEBUG_CORE_0_AREA_PIF_1_RD_INTR_ENA Write 1 to enable ASSIST_DEBUG_CORE_0_AREA_PIF_1_RD_INT. (R/W)

ASSIST_DEBUG_CORE_0_AREA_PIF_1_WR_INTR_ENA Write 1 to enable ASSIST_DEBUG_CORE_0_AREA_PIF_1_WR_INT. (R/W)

ASSIST_DEBUG_CORE_0_SP_SPILL_MIN_INTR_ENA Write 1 to enable ASSIST_DEBUG_CORE_0_SP_SPILL_MIN_INT. (R/W)

ASSIST_DEBUG_CORE_0_SP_SPILL_MAX_INTR_ENA Write 1 to enable ASSIST_DEBUG_CORE_0_SP_SPILL_MAX_INT. (R/W)

Register 22.42. ASSIST_DEBUG_CORE_0_INTR_CLR_REG (0x000C)



ASSIST_DEBUG_CORE_0_AREA_DRAM0_0_RD_CLR Write 1 to clear AS-SIST_DEBUG_CORE_0_AREA_DRAM0_0_RD_INT. (WT)

ASSIST_DEBUG_CORE_0_AREA_DRAM0_0_WR_CLR Write 1 to clear AS-SIST_DEBUG_CORE_0_AREA_DRAM0_0_WR_INT. (WT)

ASSIST_DEBUG_CORE_0_AREA_DRAM0_1_RD_CLR Write 1 to clear AS-SIST_DEBUG_CORE_0_AREA_DRAM0_1_RD_INT. (WT)

ASSIST_DEBUG_CORE_0_AREA_DRAM0_1_WR_CLR Write 1 to clear AS-SIST_DEBUG_CORE_0_AREA_DRAM0_1_WR_INT. (WT)

ASSIST_DEBUG_CORE_0_AREA_PIF_0_RD_CLR Write 1 to clear AS-SIST_DEBUG_CORE_0_AREA_PIF_0_RD_INT. (WT)

ASSIST_DEBUG_CORE_0_AREA_PIF_0_WR_CLR Write 1 to clear AS-SIST_DEBUG_CORE_0_AREA_PIF_0_WR_INT. (WT)

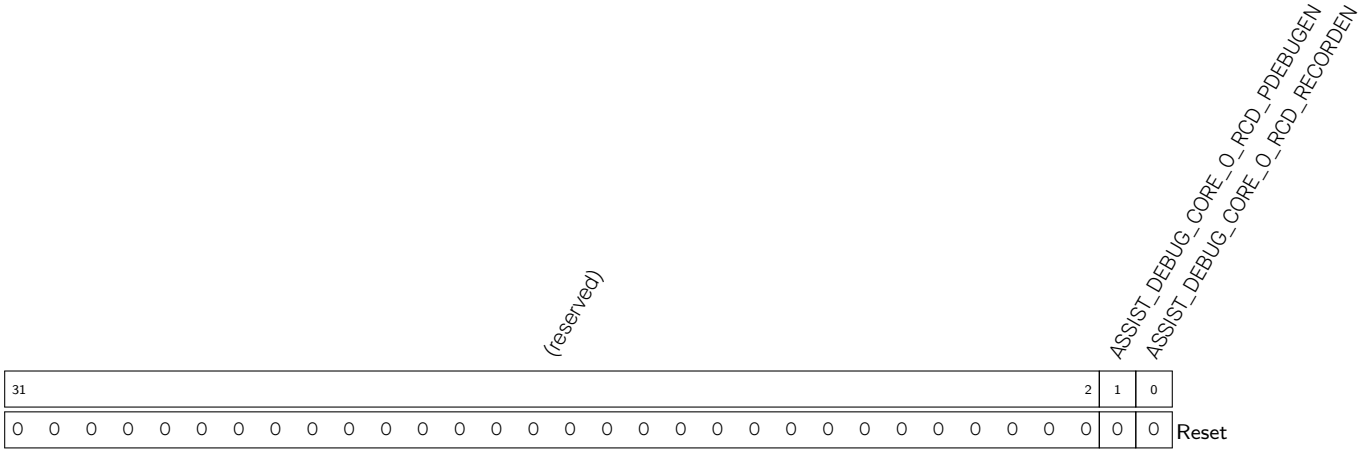
ASSIST_DEBUG_CORE_0_AREA_PIF_1_RD_CLR Write 1 to clear AS-SIST_DEBUG_CORE_0_AREA_PIF_1_RD_INT. (WT)

ASSIST_DEBUG_CORE_0_AREA_PIF_1_WR_CLR Write 1 to clear AS-SIST_DEBUG_CORE_0_AREA_PIF_1_WR_INT. (WT)

ASSIST_DEBUG_CORE_0_SP_SPILL_MIN_CLR Write 1 to clear AS-SIST_DEBUG_CORE_0_SP_SPILL_MIN_INT. (WT)

ASSIST_DEBUG_CORE_0_SP_SPILL_MAX_CLR Write 1 to clear AS-SIST_DEBUG_CORE_0_SP_SPILL_MAX_INT. (WT)

Register 22.43. ASSIST_DEBUG_CORE_0_RCD_EN_REG (0x0044)



ASSIST_DEBUG_CORE_0_RCD_RECORDEN Configures whether to enable HP CPU0 PC logging.

0: Disable

1: ASSIST_DEBUG_CORE_0_RCD_PDEBUGPC_REG starts to record HP CPU0 PC in real time

(R/W)

ASSIST_DEBUG_CORE_0_RCD_PDEBUGEN Configures whether to enable HP CPU0 debugging.

0: Disable

1: HP CPU0 outputs HP CPU0 PC

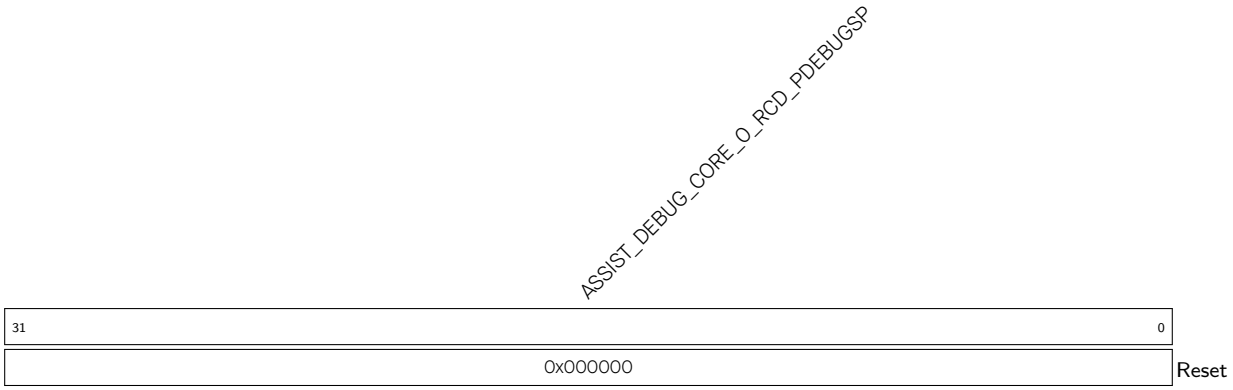
(R/W)

Register 22.44. ASSIST_DEBUG_CORE_0_RCD_PDEBUGPC_REG (0x0048)



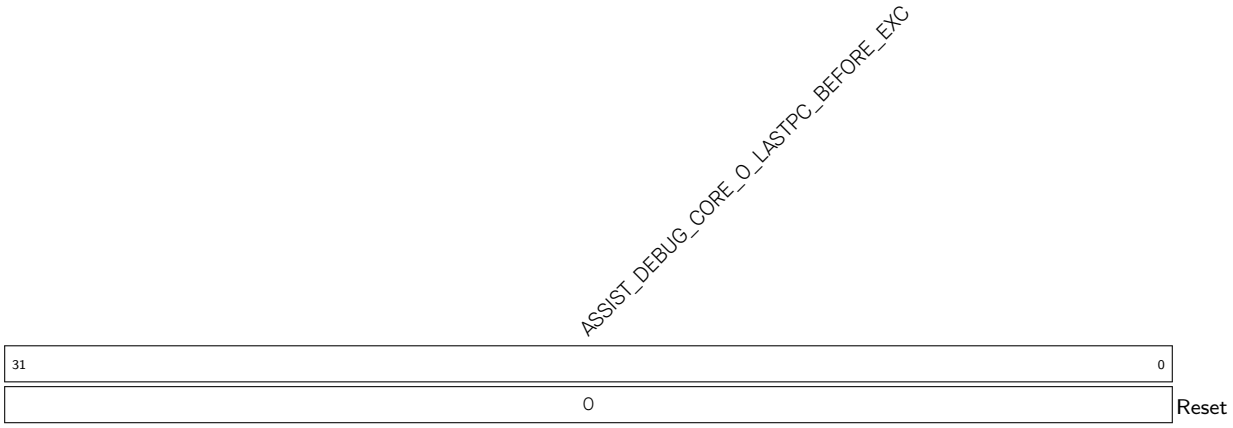
ASSIST_DEBUG_CORE_0_RCD_PDEBUGPC Represents the PC value at HP CPU0 reset. (RO)

Register 22.45. ASSIST_DEBUG_CORE_0_RCD_PDEBUGSP_REG (0x004C)



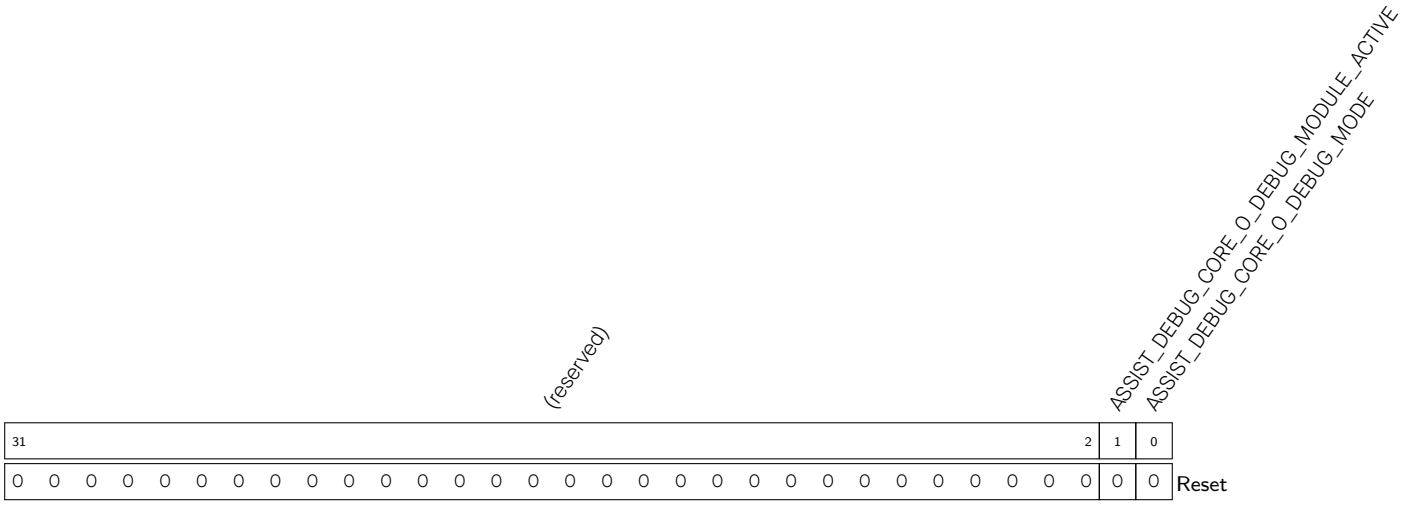
ASSIST_DEBUG_CORE_0_RCD_PDEBUGSP Represents the HP CPU0 SP. (RO)

Register 22.46. ASSIST_DEBUG_CORE_0_LASTPC_BEFORE_EXCEPTION_REG (0x0070)



ASSIST_DEBUG_CORE_0_LASTPC_BEFORE_EXC Represents the HP CPU0 PC of the last command before the HP CPU0 enters exception. (RO)

Register 22.47. ASSIST_DEBUG_CORE_0_DEBUG_MODE_REG (0x0074)



ASSIST_DEBUG_CORE_0_DEBUG_MODE Represents whether HP CPU0 is in debugging mode.

1: In debugging mode

0: Not in debugging mode

(RO)

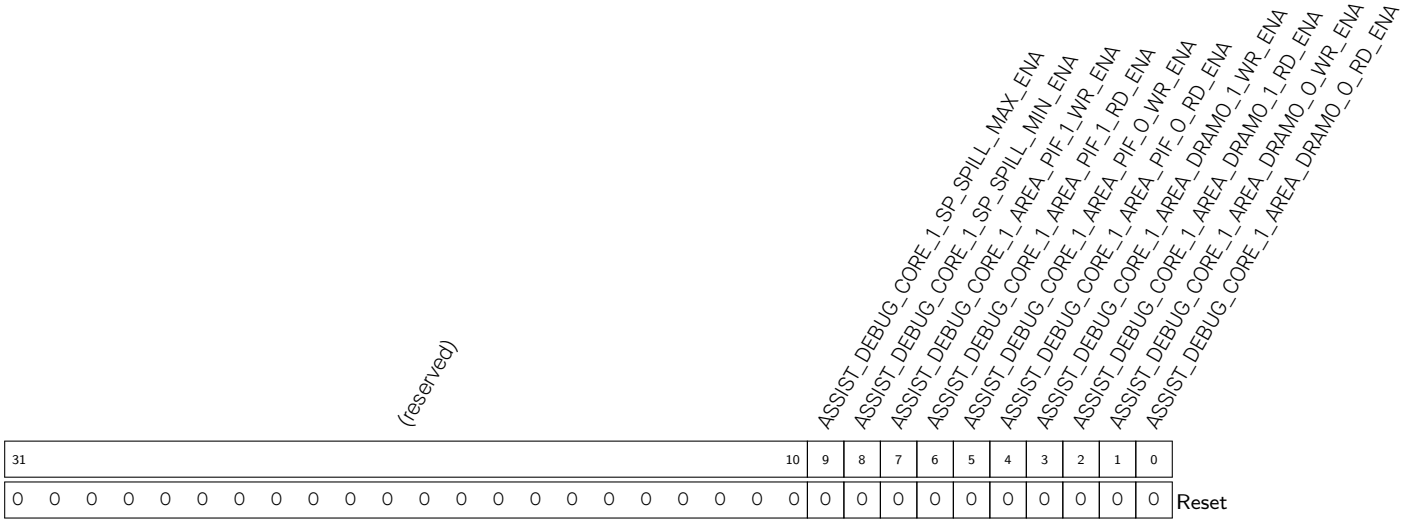
ASSIST_DEBUG_CORE_0_DEBUG_MODULE_ACTIVE Represents the status of the HP CPU0 debug module.

1: Active status

Other: Inactive status

(RO)

Register 22.48. ASSIST_DEBUG_CORE_1_MONTR_ENA_REG (0x0080)



ASSIST_DEBUG_CORE_1_AREA_DRAM0_0_RD_ENA Configures whether to monitor read operations in HP CPU1 region 0 by the HP CPU1 Data bus.

0: Not monitor

1: Monitor

(R/W)

ASSIST_DEBUG_CORE_1_AREA_DRAM0_0_WR_ENA Configures whether to monitor write operations in HP CPU1 region 0 by the HP CPU1 Data bus.

0: Not monitor

1: Monitor

(R/W)

ASSIST_DEBUG_CORE_1_AREA_DRAM0_1_RD_ENA Configures whether to monitor read operations in HP CPU1 region 1 by the HP CPU1 Data bus.

0: Not monitor

1: Monitor

(R/W)

ASSIST_DEBUG_CORE_1_AREA_DRAM0_1_WR_ENA Configures whether to monitor write operations in HP CPU1 region 1 by the HP CPU1 Data bus.

0: Not monitor

1: Monitor

(R/W)

ASSIST_DEBUG_CORE_1_AREA_PIF_0_RD_ENA Configures whether to monitor read operations in HP CPU1 region 0 by the HP CPU1 Peripheral bus.

0: Not monitor

1: Monitor

(R/W)

Continued on the next page...

Register 22.48. ASSIST_DEBUG_CORE_1_MONTR_ENA_REG (0x0000)

Continued from the previous page...

ASSIST_DEBUG_CORE_1_AREA_PIF_0_WR_ENA Configures whether to monitor write operations in HP CPU1 region 0 by the HP CPU1 Peripheral bus.
0: Not monitor
1: Monitor
(R/W)

ASSIST_DEBUG_CORE_1_AREA_PIF_1_RD_ENA Configures whether to monitor read operations in HP CPU1 region 1 by the HP CPU1 Peripheral bus.
0: Not monitor
1: Monitor
(R/W)

ASSIST_DEBUG_CORE_1_AREA_PIF_1_WR_ENA Configures whether to monitor write operations in HP CPU1 region 1 by the HP CPU1 Peripheral bus.
0: Not monitor
1: Monitor
(R/W)

ASSIST_DEBUG_CORE_1_SP_SPILL_MIN_ENA Configures whether to monitor HP CPU1 SP going below the lower bound address of HP CPU1 SP monitored region.
0: Not monitor
1: Monitor
(R/W)

ASSIST_DEBUG_CORE_1_SP_SPILL_MAX_ENA Configures whether to monitor HP CPU1 SP going beyond the upper bound address of HP CPU1 SP monitored region.
0: Not monitor
1: Monitor
(R/W)

Register 22.49. ASSIST_DEBUG_CORE_1_AREA_DRAM0_0_MIN_REG (0x0090)



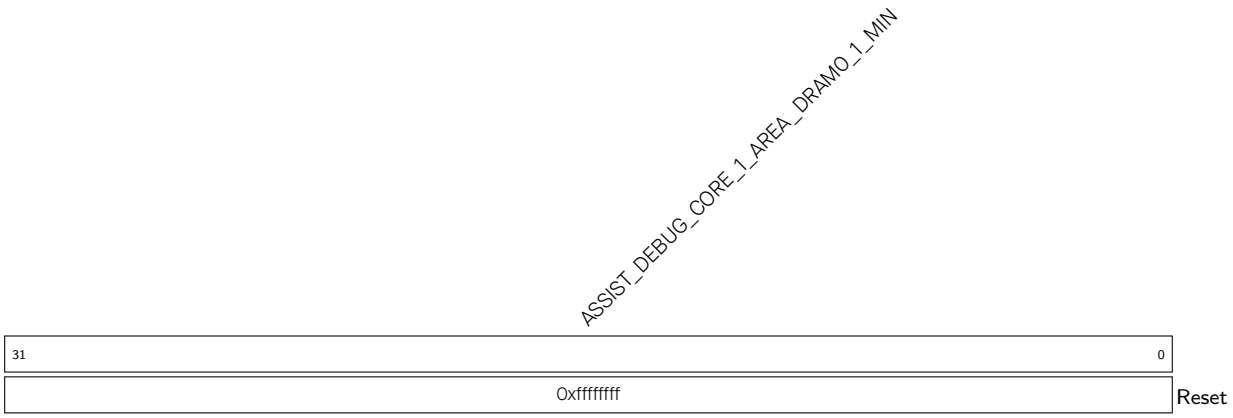
ASSIST_DEBUG_CORE_1_AREA_DRAM0_0_MIN Configures the lower bound address of HP CPU1 Data bus region 0. (R/W)

Register 22.50. ASSIST_DEBUG_CORE_1_AREA_DRAM0_0_MAX_REG (0x0094)



ASSIST_DEBUG_CORE_1_AREA_DRAM0_0_MAX Configures the upper bound address of HP CPU1 Data bus region 0. (R/W)

Register 22.51. ASSIST_DEBUG_CORE_1_AREA_DRAM0_1_MIN_REG (0x0098)



ASSIST_DEBUG_CORE_1_AREA_DRAM0_1_MIN Configures the lower bound address of HP CPU1 Data bus region 1. (R/W)

Register 22.52. ASSIST_DEBUG_CORE_1_AREA_DRAM0_1_MAX_REG (0x009C)



ASSIST_DEBUG_CORE_1_AREA_DRAM0_1_MAX Configures the upper bound address of HP CPU1 Data bus region 1. (R/W)

Register 22.53. ASSIST_DEBUG_CORE_1_AREA_PIF_0_MIN_REG (0x00A0)



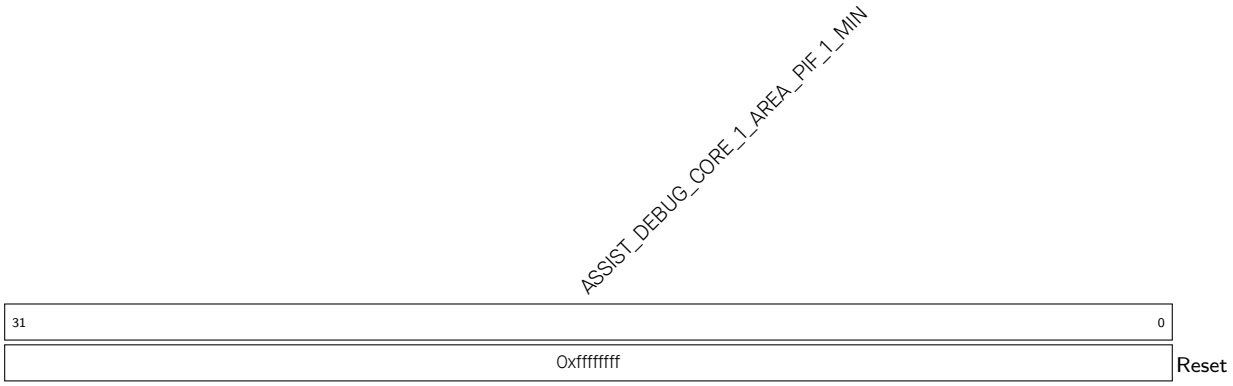
ASSIST_DEBUG_CORE_1_AREA_PIF_0_MIN Configures the lower bound address of HP CPU1 Peripheral bus region 0. (R/W)

Register 22.54. ASSIST_DEBUG_CORE_1_AREA_PIF_0_MAX_REG (0x00A4)



ASSIST_DEBUG_CORE_1_AREA_PIF_0_MAX Configures the upper bound address of HP CPU1 Peripheral bus region 0. (R/W)

Register 22.55. ASSIST_DEBUG_CORE_1_AREA_PIF_1_MIN_REG (0x00A8)



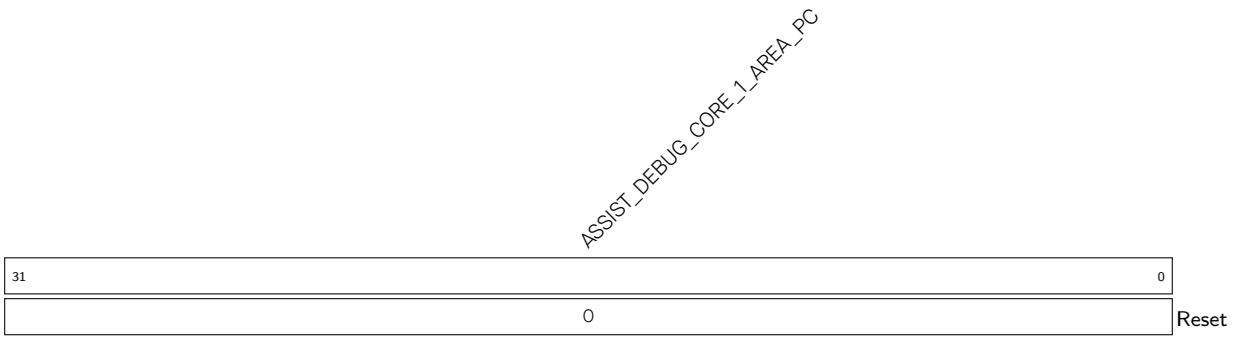
ASSIST_DEBUG_CORE_1_AREA_PIF_1_MIN Configures the lower bound address of HP CPU1 Peripheral bus region 1. (R/W)

Register 22.56. ASSIST_DEBUG_CORE_1_AREA_PIF_1_MAX_REG (0x00AC)



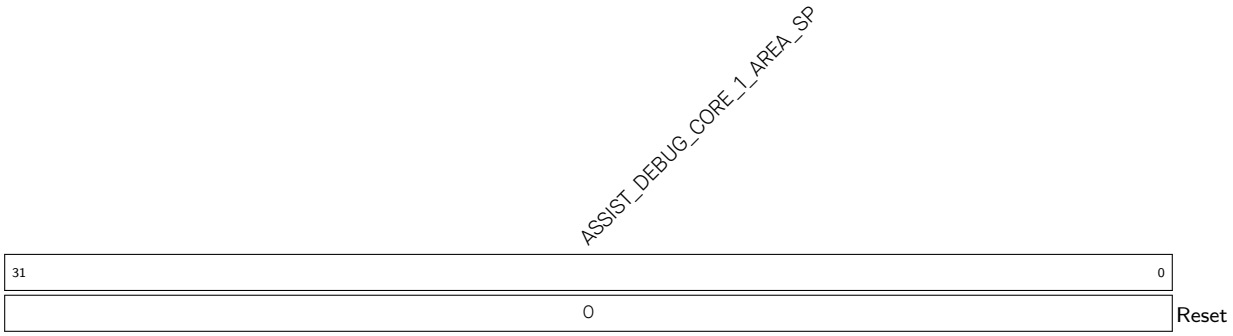
ASSIST_DEBUG_CORE_1_AREA_PIF_1_MAX Configures the upper bound address of HP CPU1 Peripheral bus region 1. (R/W)

Register 22.57. ASSIST_DEBUG_CORE_1_AREA_PC_REG (0x00B0)



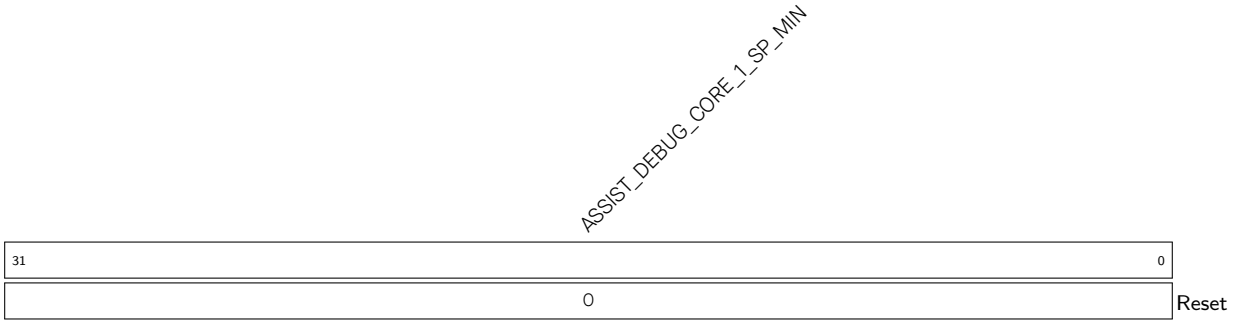
ASSIST_DEBUG_CORE_1_AREA_PC Represents the HP CPU1 PC value when an interrupt is triggered during region monitoring. (RO)

Register 22.58. ASSIST_DEBUG_CORE_1_AREA_SP_REG (0x00B4)



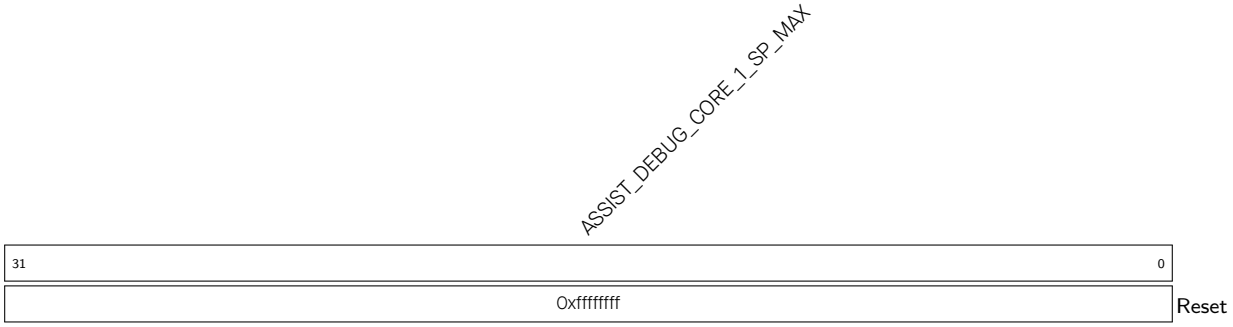
ASSIST_DEBUG_CORE_1_AREA_SP Represents the HP CPU1 SP value when an interrupt is triggered during region monitoring. (RO)

Register 22.59. ASSIST_DEBUG_CORE_1_SP_MIN_REG (0x00B8)



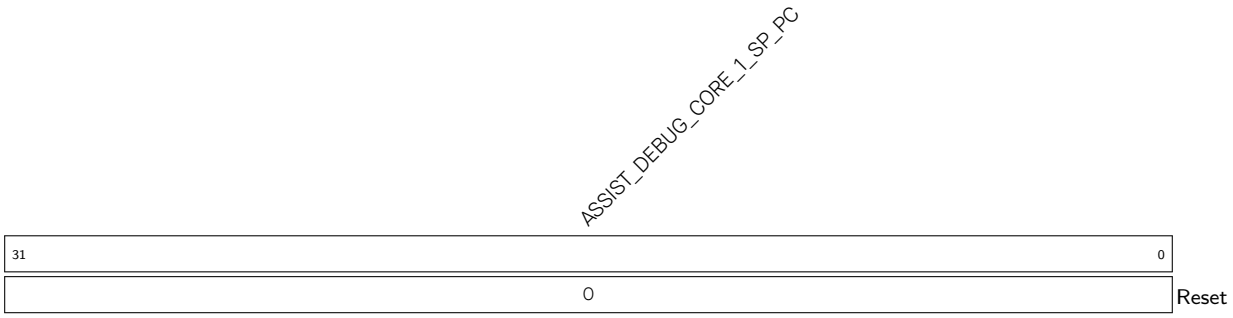
ASSIST_DEBUG_CORE_1_SP_MIN Configures the lower bound address of the HP CPU1 SP monitored region. (R/W)

Register 22.60. ASSIST_DEBUG_CORE_1_SP_MAX_REG (0x00BC)



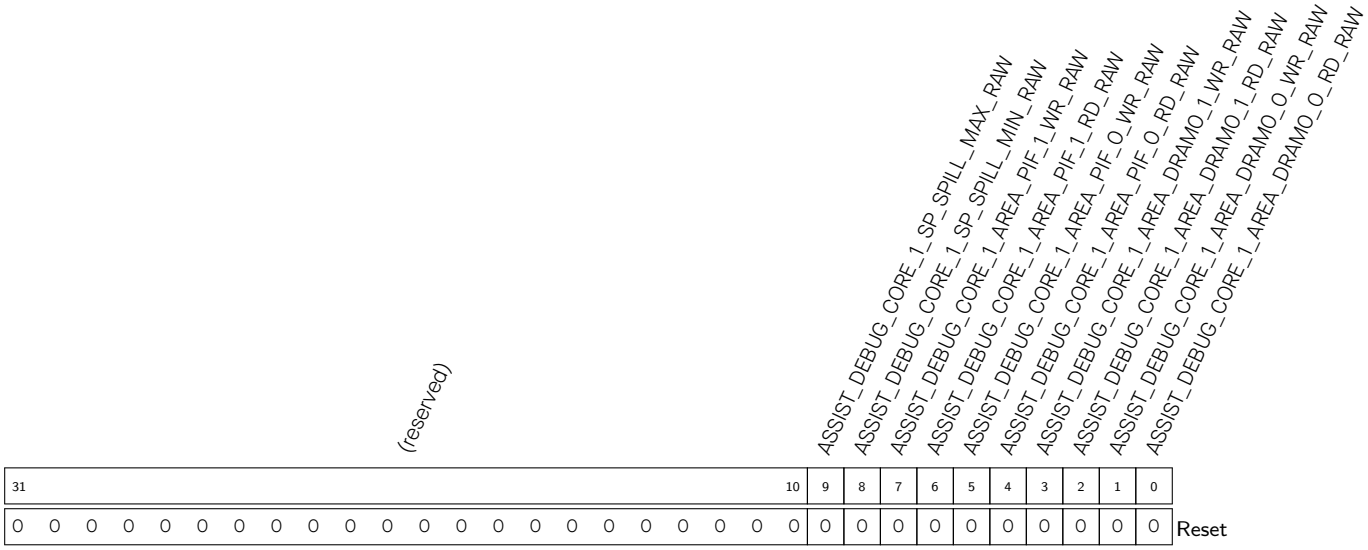
ASSIST_DEBUG_CORE_1_SP_MAX Configures the upper bound address of the HP CPU1 SP monitored region. (R/W)

Register 22.61. ASSIST_DEBUG_CORE_1_SP_PC_REG (0x00C0)



ASSIST_DEBUG_CORE_1_SP_PC Represents the HP CPU1 PC value during SP monitoring. (RO)

Register 22.62. ASSIST_DEBUG_CORE_1_INTR_RAW_REG (0x0084)



ASSIST_DEBUG_CORE_1_AREA_DRAM0_0_RD_RAW The raw interrupt status of ASSIST_DEBUG_CORE_1_AREA_DRAM0_0_RD_INT. (RO)

ASSIST_DEBUG_CORE_1_AREA_DRAM0_0_WR_RAW The raw interrupt status of ASSIST_DEBUG_CORE_1_AREA_DRAM0_0_WR_INT. (RO)

ASSIST_DEBUG_CORE_1_AREA_DRAM0_1_RD_RAW The raw interrupt status of ASSIST_DEBUG_CORE_1_AREA_DRAM0_1_RD_INT. (RO)

ASSIST_DEBUG_CORE_1_AREA_DRAM0_1_WR_RAW The raw interrupt status of ASSIST_DEBUG_CORE_1_AREA_DRAM0_1_WR_INT. (RO)

ASSIST_DEBUG_CORE_1_AREA_PIF_0_RD_RAW The raw interrupt status of ASSIST_DEBUG_CORE_1_AREA_PIF_0_RD_INT. (RO)

ASSIST_DEBUG_CORE_1_AREA_PIF_0_WR_RAW The raw interrupt status of ASSIST_DEBUG_CORE_1_AREA_PIF_0_WR_INT. (RO)

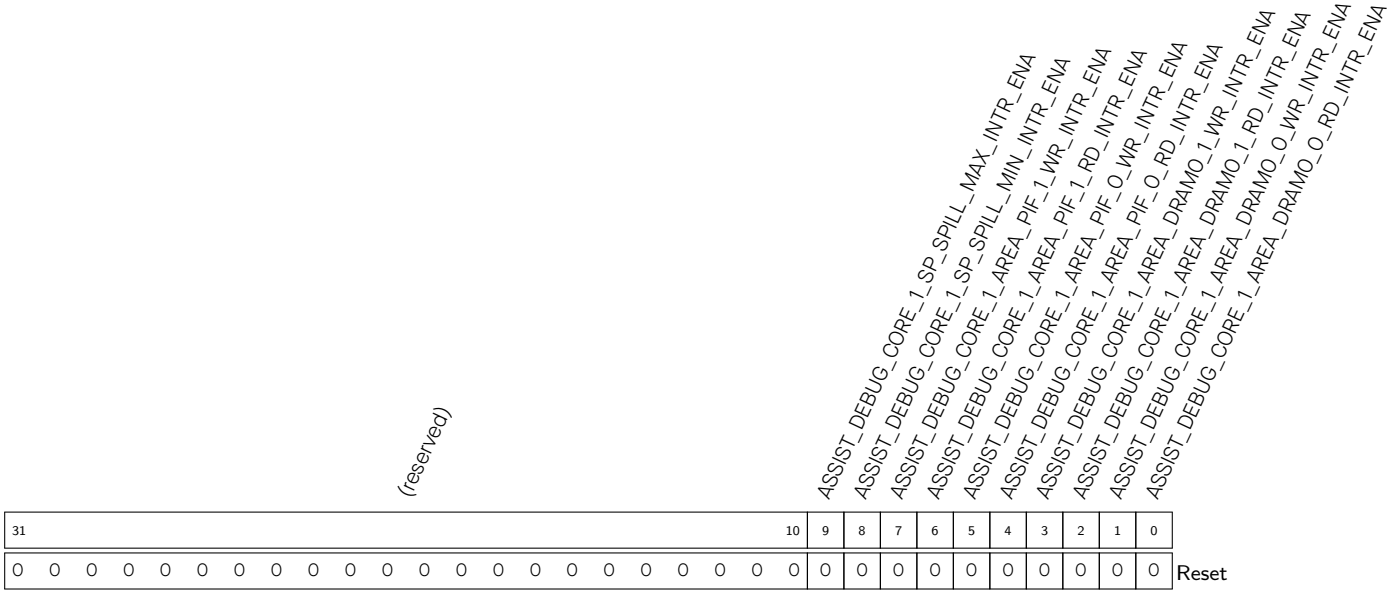
ASSIST_DEBUG_CORE_1_AREA_PIF_1_RD_RAW The raw interrupt status of ASSIST_DEBUG_CORE_1_AREA_PIF_1_RD_INT. (RO)

ASSIST_DEBUG_CORE_1_AREA_PIF_1_WR_RAW The raw interrupt status of ASSIST_DEBUG_CORE_1_AREA_PIF_1_WR_INT. (RO)

ASSIST_DEBUG_CORE_1_SP_SPILL_MIN_RAW The raw interrupt status of ASSIST_DEBUG_CORE_1_SP_SPILL_MIN_INT. (RO)

ASSIST_DEBUG_CORE_1_SP_SPILL_MAX_RAW The raw interrupt status of ASSIST_DEBUG_CORE_1_SP_SPILL_MAX_INT. (RO)

Register 22.63. ASSIST_DEBUG_CORE_1_INTR_ENA_REG (0x0088)



ASSIST_DEBUG_CORE_1_AREA_DRAMO_0_RD_INTR_ENA Write 1 to enable ASSIST_DEBUG_CORE_1_AREA_DRAMO_0_RD_INT. (R/W)

ASSIST_DEBUG_CORE_1_AREA_DRAMO_0_WR_INTR_ENA Write 1 to enable ASSIST_DEBUG_CORE_1_AREA_DRAMO_0_WR_INT. (R/W)

ASSIST_DEBUG_CORE_1_AREA_DRAMO_1_RD_INTR_ENA Write 1 to enable ASSIST_DEBUG_CORE_1_AREA_DRAMO_1_RD_INT. (R/W)

ASSIST_DEBUG_CORE_1_AREA_DRAMO_1_WR_INTR_ENA Write 1 to enable ASSIST_DEBUG_CORE_1_AREA_DRAMO_1_WR_INT. (R/W)

ASSIST_DEBUG_CORE_1_AREA_PIF_0_RD_INTR_ENA Write 1 to enable ASSIST_DEBUG_CORE_1_AREA_PIF_0_RD_INT. (R/W)

ASSIST_DEBUG_CORE_1_AREA_PIF_0_WR_INTR_ENA Write 1 to enable ASSIST_DEBUG_CORE_1_AREA_PIF_0_WR_INT. (R/W)

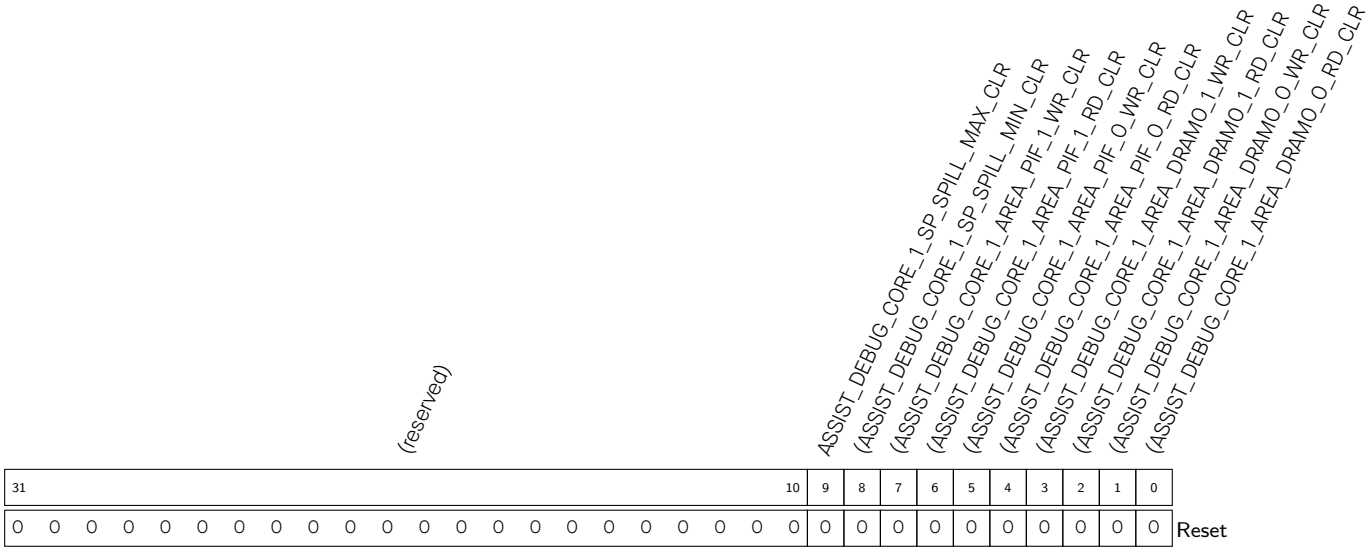
ASSIST_DEBUG_CORE_1_AREA_PIF_1_RD_INTR_ENA Write 1 to enable ASSIST_DEBUG_CORE_1_AREA_PIF_1_RD_INT. (R/W)

ASSIST_DEBUG_CORE_1_AREA_PIF_1_WR_INTR_ENA Write 1 to enable ASSIST_DEBUG_CORE_1_AREA_PIF_1_WR_INT. (R/W)

ASSIST_DEBUG_CORE_1_SP_SPILL_MIN_INTR_ENA Write 1 to enable ASSIST_DEBUG_CORE_1_SP_SPILL_MIN_INT. (R/W)

ASSIST_DEBUG_CORE_1_SP_SPILL_MAX_INTR_ENA Write 1 to enable ASSIST_DEBUG_CORE_1_SP_SPILL_MAX_INT. (R/W)

Register 22.64. ASSIST_DEBUG_CORE_1_INTR_CLR_REG (0x008C)



ASSIST_DEBUG_CORE_1_AREA_DRAM0_0_RD_CLR Write 1 to clear AS-SIST_DEBUG_CORE_1_AREA_DRAM0_0_RD_INT. (WT)

ASSIST_DEBUG_CORE_1_AREA_DRAM0_0_WR_CLR Write 1 to clear AS-SIST_DEBUG_CORE_1_AREA_DRAM0_0_WR_INT. (WT)

ASSIST_DEBUG_CORE_1_AREA_DRAM0_1_RD_CLR Write 1 to clear AS-SIST_DEBUG_CORE_1_AREA_DRAM0_1_RD_INT. (WT)

ASSIST_DEBUG_CORE_1_AREA_DRAM0_1_WR_CLR Write 1 to clear AS-SIST_DEBUG_CORE_1_AREA_DRAM0_1_WR_INT. (WT)

ASSIST_DEBUG_CORE_1_AREA_PIF_0_RD_CLR Write 1 to clear AS-SIST_DEBUG_CORE_1_AREA_PIF_0_RD_INT. (WT)

ASSIST_DEBUG_CORE_1_AREA_PIF_0_WR_CLR Write 1 to clear AS-SIST_DEBUG_CORE_1_AREA_PIF_0_WR_INT. (WT)

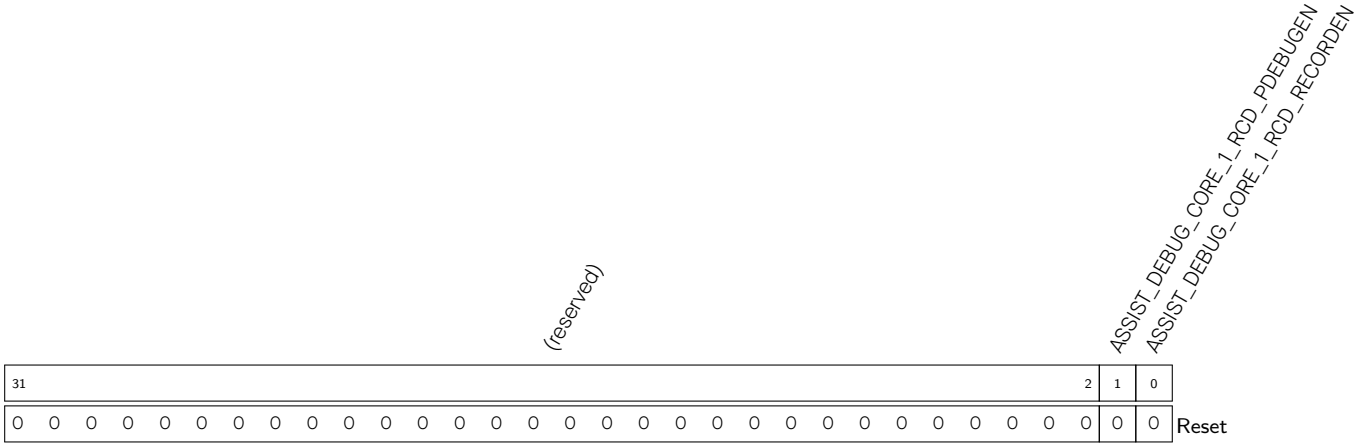
ASSIST_DEBUG_CORE_1_AREA_PIF_1_RD_CLR Write 1 to clear AS-SIST_DEBUG_CORE_1_AREA_PIF_1_RD_INT. (WT)

ASSIST_DEBUG_CORE_1_AREA_PIF_1_WR_CLR Write 1 to clear AS-SIST_DEBUG_CORE_1_AREA_PIF_1_WR_INT. (WT)

ASSIST_DEBUG_CORE_1_SP_SPILL_MIN_CLR Write 1 to clear AS-SIST_DEBUG_CORE_1_SP_SPILL_MIN_INT. (WT)

ASSIST_DEBUG_CORE_1_SP_SPILL_MAX_CLR Write 1 to clear AS-SIST_DEBUG_CORE_1_SP_SPILL_MAX_INT. (WT)

Register 22.65. ASSIST_DEBUG_CORE_1_RCD_EN_REG (0x00C4)



ASSIST_DEBUG_CORE_1_RCD_RECORDEN Configures whether to enable HP CPU1 PC logging.

0: Disable

1: ASSIST_DEBUG_CORE_1_RCD_PDEBUGPC_REG starts to record HP CPU1 PC in real time

(R/W)

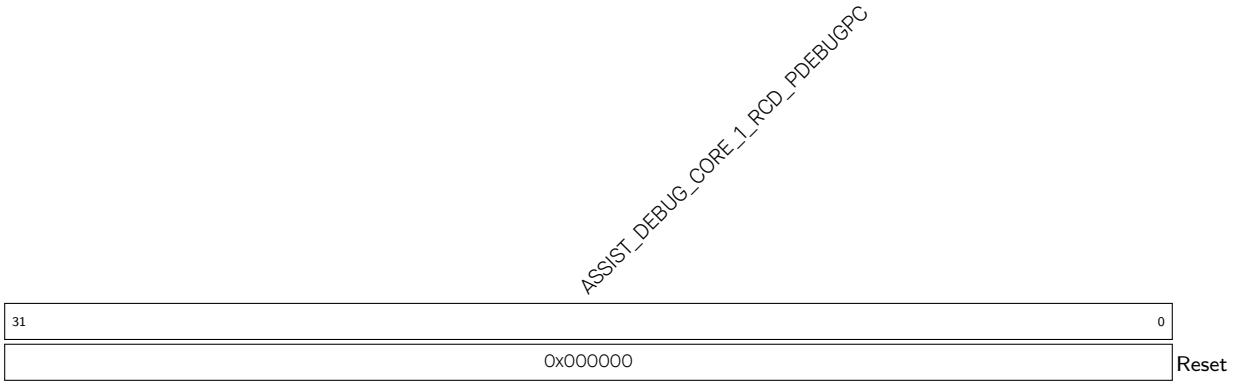
ASSIST_DEBUG_CORE_1_RCD_PDEBUGEN Configures whether to enable HP CPU1 debugging.

0: Disable

1: HP CPU1 outputs HP CPU1 PC

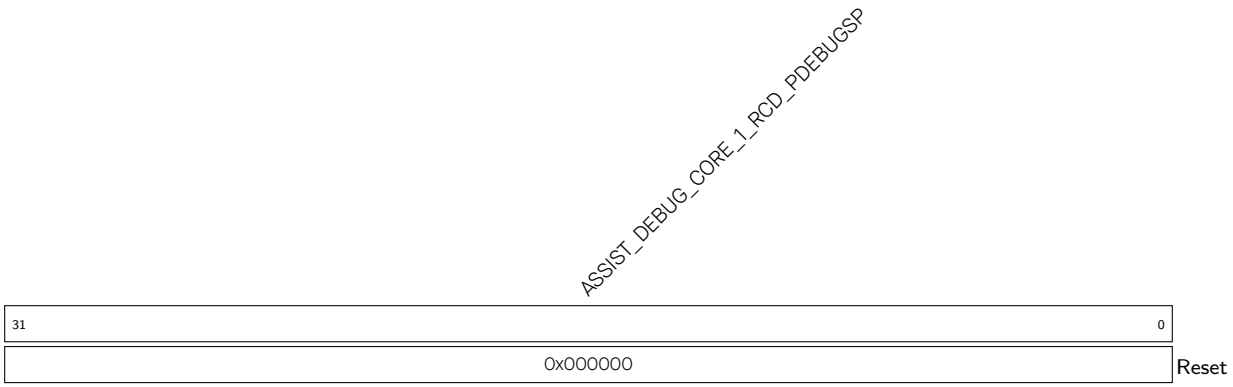
(R/W)

Register 22.66. ASSIST_DEBUG_CORE_1_RCD_PDEBUGPC_REG (0x00C8)



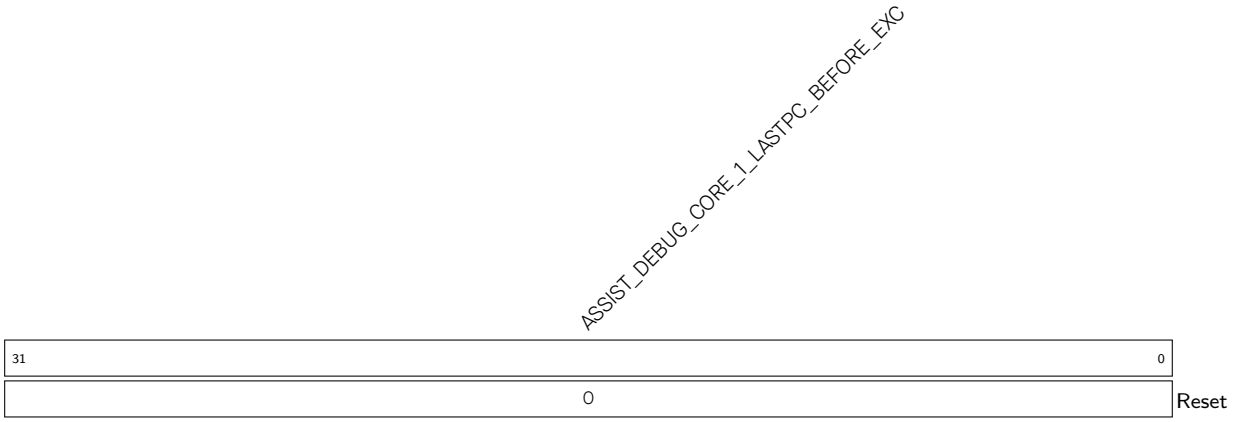
ASSIST_DEBUG_CORE_1_RCD_PDEBUGPC Represents the PC value at HP CPU1 reset. (RO)

Register 22.67. ASSIST_DEBUG_CORE_1_RCD_PDEBUGSP_REG (0x00CC)



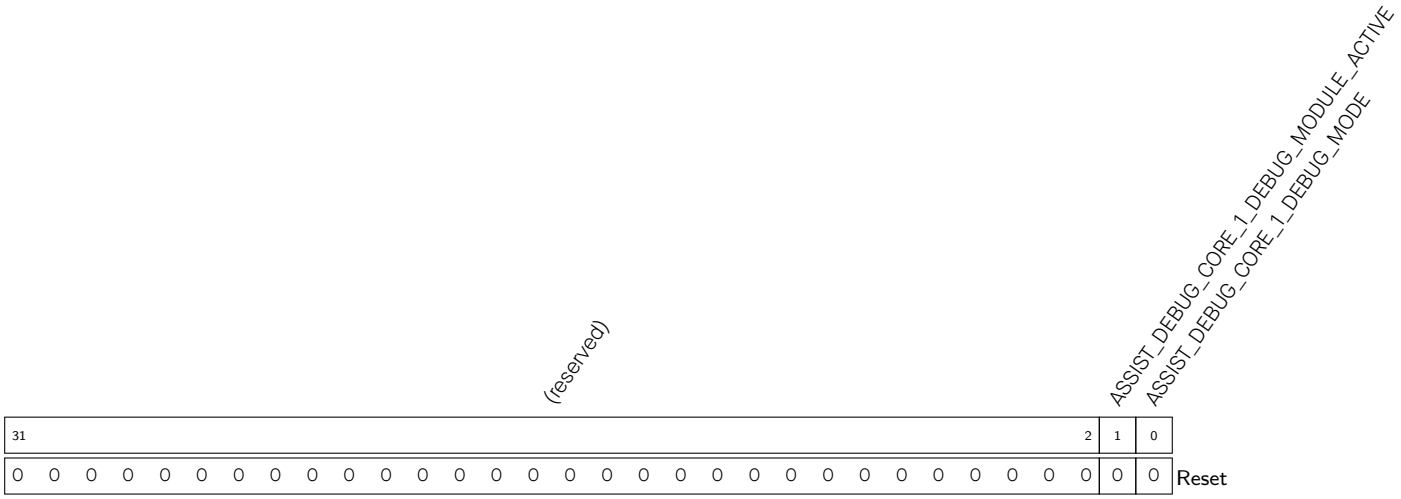
ASSIST_DEBUG_CORE_1_RCD_PDEBUGSP Represents the HP CPU1 SP. (RO)

Register 22.68. ASSIST_DEBUG_CORE_1_LASTPC_BEFORE_EXCEPTION_REG (0x00F0)



ASSIST_DEBUG_CORE_1_LASTPC_BEFORE_EXC Represents the HP CPU1 PC of the last command before the HP CPU1 enters exception. (RO)

Register 22.69. ASSIST_DEBUG_CORE_1_DEBUG_MODE_REG (0x00F4)



ASSIST_DEBUG_CORE_1_DEBUG_MODE Represents whether HP CPU1 is in debugging mode.

1: In debugging mode

0: Not in debugging mode

(RO)

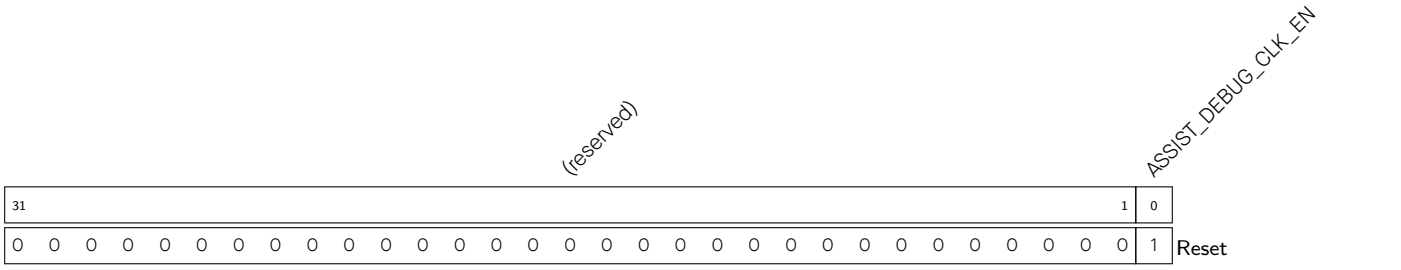
ASSIST_DEBUG_CORE_1_DEBUG_MODULE_ACTIVE Represents the status of the HP CPU1 debug module.

1: Active status

Other: Inactive status

(RO)

Register 22.70. ASSIST_DEBUG_CLOCK_GATE_REG (0x0108)



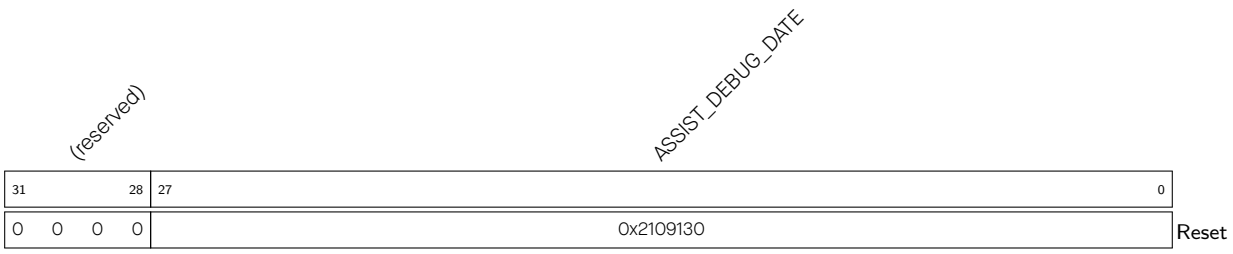
ASSIST_DEBUG_CLK_EN Configures whether to enable the register clock gating.

0: Disable

1: Enable

(R/W)

Register 22.71. ASSIST_DEBUG_DATE_REG (0x03FC)



ASSIST_DEBUG_DATE Version control register. (R/W)

Chapter 23

AXI Performance Monitor

23.1 Overview

The Advanced eXtensible Interface Performance Monitor (AXI Perf Mon) is used to monitor AXI bus transfer activities in real time and provide monitoring results based on measured metrics. Software can use these results to evaluate the current status of bus transfers, including performance-related data such as bandwidth and latency, for software testing or debugging.

AXI Perf Mon has nine monitoring ports, each corresponding to a device on the AXI matrix, including cache, vdma_mst1, vdma_mst2, 2d-dma, h264-dma_mst1, h264-dma_mst2, psram, camera, and lcd. AXI Perf Mon contains two monitoring units internally. Each monitoring unit is identical and completely independent, and is shared by all monitoring ports. Before starting monitoring, users can assign each unit to any port to monitor the bus behavior of the corresponding port.

23.2 Terminology

To better illustrate the functionality of AXI Perf Mon, the following terms are used in this chapter.

AXI Advanced eXtensible Interface, a part of the AMBA 3 (Advanced Microcontroller Bus Architecture) protocol that provides high-bandwidth, low-latency data transfer between memory and peripherals, or between memory blocks.

agent A device on the AXI bus, including both AXI Master interfaces and Slave interfaces.

outstanding In the AXI protocol, when the address of an AXI transaction has been accepted by the Slave, but the final response (BRESP/RRESP) has not yet been returned to the Master, the transaction is in the outstanding (pending/in-flight) state.

burst In the AXI protocol, burst transfer refers to a mechanism that completes multiple data transfers through consecutive data beats after a single address handshake. Burst is introduced to reduce the repeated overhead of the address channel and improve bus bandwidth utilization and transfer efficiency. A burst transfer consists of one address phase and multiple data phases, where the address phase occurs only once at the beginning of the burst, and subsequent data beats are consecutively transferred according to established rules until the burst ends.

AWVALID/AWREADY Handshake signals for address channel transfer in the AXI protocol. AWVALID asserted high indicates that the upstream initiates a write command transfer request, AWREADY asserted high indicates that the downstream responds to the upstream write command transfer request, and a successful handshake completes one write command transfer. ARVALID/ARREADY work similarly, representing read transfer command handshake.

WVALID/WREADY Handshake signals for data channel transfer in the AXI protocol. WVALID asserted high indicates that the upstream initiates a write data transfer request; WREADY asserted high indicates that the downstream responds to the upstream write data transfer request, and a successful handshake completes one beat of write data transfer. RVALID/RREADY work similarly, but read data originates from the downstream and the upstream receives the data, and a successful handshake completes one beat of read data transfer.

APB Advanced Peripheral Bus (APB) is a low-bandwidth, low-power bus protocol mainly used to connect low-speed peripherals (such as GPIO, timers, UART, ADC, etc.) with the system's main processor or other high-performance modules. APB is part of the ARM AMBA (Advanced Microcontroller Bus Architecture) bus family, forming the bus architecture of AMBA systems together with AHB (Advanced High-performance Bus) and AXI (Advanced eXtensible Interface).

port A logical interface at the boundary of a module for signal communication with external logic.

unit A hardware block that implements a specific function within a larger design. In this article, this refers to the unit responsible for performance monitoring.

n/*m* In the section on registers, the notation *n*/*m* is frequently used for indexing. *n* represents monitoring unit 0 or 1; *m* represents that each monitoring unit has multiple such registers. Refer to the register section for specific register quantities.

region In this design, a "region" is related to the concept of an address range, but with some distinctions. When monitoring instantaneous bandwidth, regions can be used to filter out irrelevant bandwidth statistics. A region must be defined based on complete slave addresses. It may include one or more entire slave address ranges, but it cannot filter only a portion of a single slave's address range.

23.3 Feature List

AXI Perf Mon has the following features:

- AXI3 bus interface compliance
- Two identical and independent monitoring units
- Nine monitoring ports
- Software-configurable mapping between monitoring units and monitoring ports
- Transaction ID filtering to exclude specific transactions from statistics
- Multiple measurement metrics available for user selection
  - Current outstanding transaction count for read/write address channels
  - Read/write address channel latency
  - Read/write data channel latency
  - Total idle cycles on write data channel within a single transaction
  - Read transaction latency
  - Write transaction latency

  - Average bandwidth measurement
  - Instantaneous bandwidth measurement
  - Instantaneous bandwidth range statistics and abnormal bandwidth recording
- Up to 6 measurement metric counters per monitoring unit
- Each counter can be configured independently, and each metric can be assigned to at most one counter within a monitoring unit
- Up to 6 range statistics metric counters per monitoring unit, i.e., supporting up to 6 range divisions for statistical analysis of instantaneous bandwidth monitoring results

23.4 Architectural Overview

Figure 23.4-1 shows the basic architecture of AXI Perf Mon.

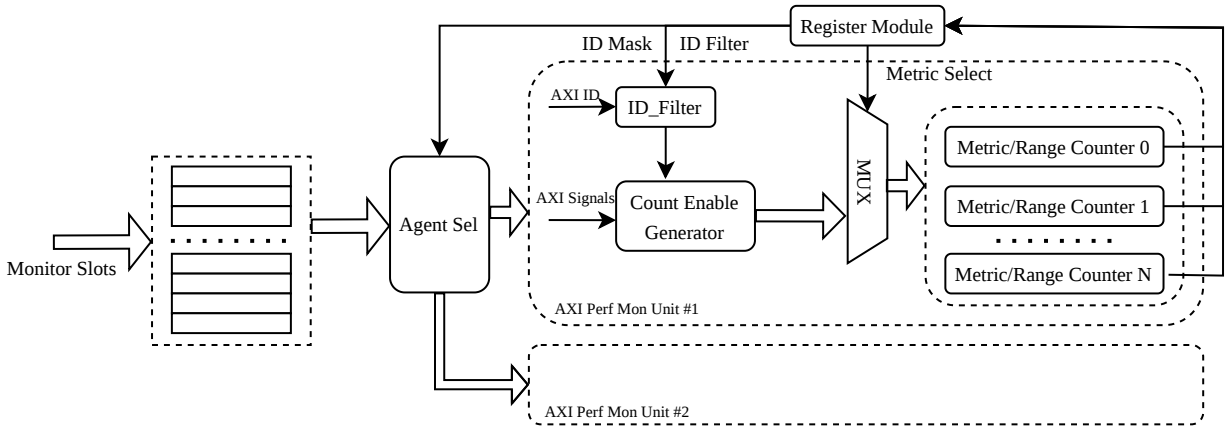


Figure 23.4-1. Architecture of AXI Perf Mon

AXI Perf Mon consists of the following functional modules:

- **Port Selection**: Selects which port to connect to the monitoring unit from the ports connected to AXI Perf Mon
- **ID Filtering**: Classifies monitoring events
- **Count Enable Generator**: Converts the metric information configured by users through registers into counter enable/disable controls
- **Counters**: Performs statistical counting

23.5 Functional Description

23.5.1 Clock

The AXI Perf Mon operates using two independent clock domains, each serving a specific interface:

- APB_CLK: APB bus clock used to configure AXI Perf Mon registers

- AXI_CLK: SoC system AXI bus clock

AXI Perf Mon is a system auxiliary debug component, so the clock is disabled by default during normal operation. To use AXI Perf Mon, the clock must be enabled first in the clock and reset module. The operating clock of the AXI Perf Mon monitoring unit is fully synchronous (same frequency and phase) with the clock of the monitored bus. All clock cycles mentioned in the following sections are based on the AXI_CLK of AXI Perf Mon.

For more information on clocks, see Chapter 11 *Reset and Clock*.

For detailed clock programming procedures, see Section 23.7 *Programming Procedures*.

23.5.2 Reset

AXI Perf Mon supports software reset, which can reset the entire module, including all monitoring units and registers.

Reset is performed by writing 1 and then 0 to the HP_SYS_CLKRST_REG_AXI_PERF_MON_SYS_RST_EN field.

23.5.3 Transaction ID Filtering

- In AXI transfer transactions, each transaction has a corresponding ID used to mark that transaction. Some Masters assign different transaction IDs to different types of transactions. The transaction ID filter is used to mask some transactions that the user is not concerned about, avoiding interference with statistical results.
- ID filtering can be configured through AXI_PERF_MON_SEL_AG*n*_RD_ID_MASK or AXI_PERF_MON_SEL_AG*n*_WR_ID_MASK (ID mask) and AXI_PERF_MON_SEL_AG*n*_RD_ID_FILTER or AXI_PERF_MON_SEL_AG*n*_WR_ID_FILTER (ID filter value).
- In the ID mask, 0 indicates an ignorable bit, and 1 indicates a bit used for filtering. If the transaction ID does not match the filter value on the bits where mask is 1, the transaction is masked and not included in statistics.
- ID filtering can be applied to both Master side and Slave side. Master-side AXI transaction IDs are 4 bits, while Slave-side transaction IDs are 8 bits.

23.5.4 Monitoring Metrics

AXI Perf Mon's port monitoring functionality is implemented by configuring the Metric Select Register (AXI_PERF_MON_SEL_AG*n*_METRIC_SELECT*m*_REG), where:

- *n* represents which AXI Perf Mon unit does it belong to, ranging from 0 to 1.
- *m* represents the serial number of this register in each unit, ranging from 1 to 2.

Each register is divided into four 8-bit segments, indexed by *i* (ranging from 0 to 3). Within each segment, bits [8**i*+4:8**i*] are used to select a measurement metric, and bits [8**i*+7:8**i*+5] are used to select a range statistics metric.

The following tables show the correspondence between register configuration values and the two types of metrics: measurement metrics and range statistics metrics.

Table 23.5-1. Setting of Measurement Metrics in Metric Select Registers

Metric Select Register [8*i+4:8*i] Value	Measurement Metric
0	Invalid value
1	Write transaction outstanding count
2	Read transaction outstanding count
3	Write address channel latency
4	Read address channel latency
5	Write data channel latency
6	Read data channel latency
7	Write data channel WVALID idle cycles
8	Read transaction transfer latency
9	Write transaction transfer latency
10	Valid data transfer count for read transaction in average bandwidth measurement
11	Time duration for read transfers in average bandwidth measurement
12	Valid data transfer count for write transaction in average bandwidth measurement
13	Time duration for write transfers in average bandwidth measurement
14	Valid data transfer count for read transaction in instantaneous bandwidth measurement
15	Time duration for read transfers in instantaneous bandwidth measurement
16	Valid data transfer count for write transaction in instantaneous bandwidth measurement
17	Time duration for write transfers in instantaneous bandwidth measurement
18–31	Invalid values

Table 23.5-2. Correspondence between Metric Select Registers and Range Statistics Metrics

Metric Select Register [8*i+7:8*i+5] Value	Range Statistics Metric
0	Invalid value
1	Instantaneous bandwidth range distribution for read transactions
2	Instantaneous bandwidth range distribution for write transactions
3–7	Invalid values

By configuring the Agent Select Register (AXI_PERF_MON_AGENT_SELECT_REG), the monitoring port for each monitoring unit can be selected.

This register is divided into two segments, indexed by *i* (ranging from 0 to 1), each corresponding to the port selection configuration of monitoring unit *i*:

- Bits [4:0] select the agent monitored by monitoring unit 0.
- Bits [9:5] select the agent monitored by monitoring unit 1.

The correspondence between the register configuration values and actual agents is shown in the following table 23.5-3.

Table 23.5-3. Correspondence between Agent Select Register Configuration Values and Actual Agents

Agent Select Register [5*i+4:5*i] Value	Agent
0	Invalid value
1	cache
2	vdma_mst1
3	vdma_mst2
4	2d-dma
5	h264-dma_mst1
6	h264-dma_mst2
7	psram
8	camera
9	lcd
10–16	Invalid values

23.5.4.1 Maximum Outstanding Count for Read/Write Address Channel Transactions

Outstanding capability reflects the Master's concurrent capability for issuing AXI transfer transactions. The stronger the outstanding capability, the lower the transfer latency. In actual transfer processes, outstanding capability depends not only on the Master configuration, but also on the outstanding capability of bus components and upstream/downstream devices. AXI Perf Mon supports measuring the outstanding count of transactions on the current monitored port.

- For read transfers, after the address channel information is validly issued and before the data channel completes the last beat of data transfer corresponding to this transaction, one outstanding transfer is accumulated. If the data transfer completes, the outstanding count decreases by one.
- For write transfers, after the address channel information is validly issued and before the write response channel completes the response for this transaction, one outstanding transfer is accumulated. If the corresponding write response is received, the outstanding count decreases by one.
- During transfer, the outstanding value generally changes continuously. AXI Perf Mon writes the maximum outstanding value into the result register for software analysis.

23.5.4.2 Read/Write Address Channel Latency

Address channel latency represents how long it takes for a transfer command to be received after it is issued. AXI Perf Mon supports measuring the read/write address channel latency of transactions on the current monitored port.

- For read transfers, when the address channel ARVALID is asserted high, it indicates that the Master is ready to issue a read transaction command. The Slave asserts ARREADY high according to its current idle status. After ARREADY is asserted high, the address channel completes a transfer command

handshake. The read address channel latency is the number of cycles from ARVALID asserted high to ARREADY asserted high on the monitored port.

- The measurement method for write address channel latency is the same as for read address channel latency.
- During transfer, for the same monitored port, the latency value generally changes continuously. AXI Perf Mon writes the maximum/minimum read/write address channel latency values into the result register for software analysis.

23.5.4.3 Read/Write Data Channel Latency

For AXI transfers, after a burst transfer begins, it cannot be interrupted. However, in actual transfer processes, the data channel does not always perform back-to-back transfers, and idle cycles may exist during transfer. AXI Perf Mon supports measuring the read/write data channel latency of transactions on the current monitored port.

- For read transfers, when the data channel RVALID is asserted high, it indicates that the Slave is ready to send one beat of read transaction data. The Master asserts RREADY high according to its current idle status. After RREADY is asserted high, the read data channel completes one beat of data transfer handshake. The read data channel latency is the number of cycles from RVALID asserted high to RREADY asserted high on the monitored port.
- The measurement method for write data channel latency is the same as for read data channel latency.
- During transfer, for the same monitored port, the latency value generally changes continuously. AXI Perf Mon writes the maximum/minimum read/write data channel latency values into the result register for software analysis.

23.5.4.4 Write Data Channel WVALID Idle Cycles

When the Master begins sending a burst data transfer, idle cycles between data transfers may come from the Slave's wait time to receive data, idle cycles in the Master's WVALID, and significant idle cycles generated when passing through a conversion bridge. AXI Perf Mon supports measuring WVALID idle cycles within a transaction on the current monitored port.

- Within a write transaction, if the transfer is not yet complete and WVALID is deasserted, the total number of cycles that WVALID remains low is counted.
- During transfer, for the same monitored port, the statistics of WVALID idle cycles on the write data channel generally change continuously. AXI Perf Mon writes the maximum/minimum number of WVALID idle cycles on the write data channel into the result register for software analysis.

23.5.4.5 Read/Write Transaction Transfer Latency

During transfer processes, the latencies of the address channel and data channel are individually important. Additionally, the latency from address command issue to data channel ready is also important within a transaction. The transaction latency reflects the time interval from issuing a transfer command to starting data transfer, and can assist in determining the size of the Master-side AXI transfer buffer. AXI Perf Mon supports measuring the read/write transaction transfer latency for a transaction on the current monitored port.

- For read transfers, when ARVALID of a transaction is asserted high, the metric measurement counter starts counting latency until the RVALID signal of the first beat of data for this transaction is asserted high, at which point the counter stops counting, obtaining the read transaction transfer latency.
- For write transfers, when AWVALID of a transaction is asserted high, the metric measurement counter starts counting latency until the first beat of data for this transaction completes transfer (the counting end point differs from read transfers), at which point the counter stops counting, obtaining the write transaction transfer latency.
- A monitored port may have multiple outstanding transfers, but in read/write transaction transfer latency metrics, there is only one metric counter, so only one transaction's latency can be counted at a time. After starting to count a certain transaction, other concurrent transactions will be ignored until that count completes. To make statistics representative, transactions with longer transfer times are preferentially selected.
- During transfer, for the same monitored port, the transfer latency statistics generally change continuously. AXI Perf Mon writes the maximum/minimum transfer latency values into the result register for software analysis.

23.5.4.6 Average Bandwidth Measurement

Bandwidth is one of the most intuitive metrics for evaluating bus performance. AXI Perf Mon supports measuring average bandwidth.

- AXI Perf Mon does not have internal calculation units and does not directly calculate bandwidth values. Instead, it indirectly measures bandwidth by counting time (number of bus cycles) and data transfer amount (number of data units).
- Since both time and data amount need to be observed, average bandwidth measurement requires using two metric counters simultaneously. Bandwidth statistics ignore the impact of data strobes and count all data, including invalid data.
- When average bandwidth measurement is enabled, the time counter increments with the bus cycle, while the data amount increments according to the current data channel handshake status. After each burst transfer completes, the current time and data amount statistics are updated to the result register. If average bandwidth measurement is not turned off after data transfer completes and the data channel remains idle, the statistical results will remain at the time and data amount statistics from the end of the last burst transfer, without adding idle time. Software needs to calculate the actual average bandwidth based on the bus frequency and the number of data bytes per beat.

23.5.4.7 Instantaneous Bandwidth Measurement

Average bandwidth reflects device bandwidth demand over a longer period, but when evaluating bus performance, device bandwidth changes within shorter time windows (instantaneous bandwidth) should also be considered. AXI Perf Mon also supports measuring instantaneous bandwidth.

- Instantaneous bandwidth is measured through sliding time windows. After configuring the time window length (in basic units of bus cycles) and enabling instantaneous bandwidth measurement, measurement proceeds in a manner invisible to software. The start and stop behavior of measurement is completely controlled by hardware. When one time window measurement ends, the next time window measurement begins, performing continuous measurement.

- Instantaneous bandwidth measurement is basically the same as average bandwidth measurement and also requires two metric counters to measure time duration and data amount respectively. When the time counter reaches the register-configured time window value, it indicates that one instantaneous bandwidth measurement is complete. The counted data amount is fed back to the next-level extreme value comparator, and both the data amount counter and time counter are cleared to begin the next round of counting.
- Range statistics. If instantaneous bandwidth only records extreme values, it is insufficient to reflect bus transfer conditions. Therefore, range statistics are introduced. Range statistics are based on range statistics counters with software-configured range upper and lower limits. The number of these counters is the same as metric counters. After a metric counter completes its measurement, its result is compared against the ranges defined by the range counters for classification. This provides a better reflection of system behavior.
- Event recording. During instantaneous bandwidth measurement, the most recent four instantaneous bandwidth measurement values (the current uncompleted instantaneous bandwidth value is not counted) and the most recent four AXI interface transfer information (ID/Burst length) are written to registers. Address channel information is recorded during address channel handshake, while instantaneous bandwidth measurement values are recorded at the end of measurement windows, creating a time difference in recording. This information is updated synchronously according to actual transfer conditions, facilitating software understanding of current transfer basics.
- Abnormal feedback. During instantaneous bandwidth measurement, abnormal bandwidth changes are also reported. A minimum instantaneous bandwidth threshold (based on data amount) is set for the monitored port, representing the minimum instantaneous bandwidth value considered non-abnormal for this port during transfer. Once the metric counter result is less than this threshold, an interrupt flag signal is triggered and broadcast to all AXI Perf Mon Units, stopping instantaneous bandwidth measurement for all ports. Event recording stops and the scene is preserved. Based on event recording results, it assists in determining which device seized bandwidth. Software can also configure AXI_PERF_MON_SEL_AG0_SW_RECORD_STOP_EN or AXI_PERF_MON_SEL_AG1_SW_RECORD_STOP_EN to pause event recording for the corresponding monitoring unit.
- Address filtering. Generally, accessing internal storage does not require additional bandwidth attention. During instantaneous bandwidth measurement, if abnormal feedback is needed, it is generally when multiple Masters simultaneously access the same external storage such as PSRAM. When analyzing abnormal results, other Masters' access to other Slaves needs to be excluded, as this bandwidth is invalid for the analyzed scenario. Therefore, address filtering functionality is added to filter based on access addresses, obtaining the true instantaneous bandwidth of competing multiple Masters. The basic unit of address filtering is Slave, meaning it supports address filtering for different Slaves, but does not support filtering partial address spaces of the same Slave.
- Monitoring recovery. After abnormal bandwidth behavior is discovered and broadcast, each AXI Perf Mon will pause instantaneous bandwidth measurement. After software completes analysis of the current scene results, if continued instantaneous bandwidth monitoring is desired without system reset, it can be configured through AXI_PERF_MON_SEL_AG0_SW_RECORD_STOP_CLR or AXI_PERF_MON_SEL_AG1_SW_RECORD_STOP_CLR to clear the stagnation state.

23.6 Interrupts

ESP32-P4's AXI Perf Mon can generate the following interrupt signal that will be sent to the Interrupt Matrix.

- AXI_PERF_MON_INTR

This interrupt signal is generated by AXI Perf Mon's internal interrupt sources. Table 23.6-1 lists AXI Perf Mon's internal interrupt sources, interrupt trigger conditions, and the generated interrupt signal.

Table 23.6-1. AXI PERF MON Internal Interrupt Sources

Internal Interrupt Source	Trigger Condition	Interrupt Signal
AXI_PERF_MON_SEL_AG*n*_RECORD_-CNT_UNDER_LIMIT_INT (*n* = 0, 1)	Instantaneous bandwidth measurement value is less than the set minimum instantaneous bandwidth threshold	AXI_PERF_MON_INTR

Note:
For definitions of *interrupt*, *interrupt signal*, *interrupt source*, and their correlations, please refer to Chapter 13 *Interrupt Matrix* > Section 13.2 *Interrupt Terminology in ESP32-P4*.

Each interrupt source can be configured by a common set of registers that are described in Section *Interrupt Configuration Registers*. The specific registers can be found in Section 23.8 *Register Summary*.

23.7 Programming Procedures

23.7.1 Using AXI Perf Mon to Measure Outstanding Transactions

To use AXI Perf Mon to monitor the outstanding capability of a port:

1. Configure register AXI_PERF_MON_AGENT_SELECT_REG, where every 5 bits represent an agent selected by the corresponding monitoring unit. Bits [4:0] indicate the port to be monitored selected by AXI Perf Mon Unit 0, and bits [9:5] represent the port to be monitored selected by AXI Perf Mon Unit 1.
2. Configure register AXI_PERF_MON_SEL_AG*n*_METRIC_SELECT*m*_REG (*n* represents monitoring unit 0 or 1; *m* represents that each monitoring unit has multiple such registers, refer to the register section for specific register quantities, here *m* = 1, 2) to select the monitoring metric. Every 8 bits represent the metric corresponding to a metric counter. The value of AXI_PERF_MON_SEL_AG*n*_METRIC_SELECT1 [4:0] represents the monitoring metric corresponding to measurement metric counter 0, and its [7:5] value represents the monitoring metric corresponding to range statistics metric counter 0. The value of AXI_PERF_MON_SEL_AG*n*_METRIC_SELECT2 [4:0] represents the monitoring metric corresponding to measurement metric counter 4, and its [7:5] value represents the monitoring metric corresponding to range statistics metric counter 4, and so on. After configuration is complete, monitoring begins.
3. Obtain status information by reading AXI_PERF_MON_SEL_AG*n*_COUNTER*m*_RESULT. Bits [15:0] are the counted minimum value, and bits [31:16] are the counted maximum value. The result counter number corresponds to the selected metric counter.

4. Configure the 8 bits of the corresponding metric counter in register AXI_PERF_MON_SEL_AG*n*_METRIC_SELECT*m*_REG to 0 to stop that metric counter from monitoring this metric on this port and reset the counter.

Outstanding count, address channel latency, data channel latency, and read/write transaction transfer latency can all be monitored through this programming procedure—simply set register AXI_PERF_MON_SEL_AG*n*_METRIC_SELECT*m*_REG to the desired monitoring metric.

23.7.2 Using AXI Perf Mon to Measure Read Average Bandwidth

To use AXI Perf Mon to monitor the average bandwidth of a port over a period of time:

1. Configure register AXI_PERF_MON_AGENT_SELECT_REG to select the port to be monitored.
2. Configure register AXI_PERF_MON_SEL_AG*n*_METRIC_SELECT*m*_REG to select the following metrics for read average bandwidth measurement: time duration and data amount.
3. Configure AXI_PERF_MON_SEL_AG*n*_RD_BANDW_TIME_CNT_VALID_STROBE_NUM and AXI_PERF_MON_SEL_AG*n*_RD_BANDW_DATA_CNT_VALID_STROBE_NUM to select the counting interval for time and data. If the counting interval is configured as 3, then every 4 valid cycles increment the counter by 1, and every 4 valid data beats increment the counter by 1.
4. Configure AXI_PERF_MON_SEL_AG0_RD_BANDW_TEST_EN or AXI_PERF_MON_SEL_AG1_RD_BANDW_TEST_EN to enable average bandwidth measurement. Configure AXI_PERF_MON_SEL_AG0_RD_BANDW_TEST_STOP or AXI_PERF_MON_SEL_AG1_RD_BANDW_TEST_STOP to stop average bandwidth measurement.
5. Measurement results can be read from register AXI_PERF_MON_SEL_AG*n*_COUNTER*m*_RESULT_REG. The result counter number corresponds to the selected metric counter. The high 16 bits in the result register represent the average bandwidth data/time statistics. A value of 16'hFFFF indicates that the counter has overflowed and the result is invalid. Configure AXI_PERF_MON_SEL_AG*n*_RD_BANDW_TIME_CNT_VALID_STROBE_NUM and AXI_PERF_MON_SEL_AG*n*_RD_BANDW_DATA_CNT_VALID_STROBE_NUM to increase the counting interval, or reduce the counting time and data amount.

23.7.3 Using AXI Perf Mon to Measure Read Instantaneous Bandwidth

To use AXI Perf Mon to monitor the instantaneous bandwidth of a port:

1. Configure register AXI_PERF_MON_AGENT_SELECT_REG to select the port to be monitored.
2. Configure register AXI_PERF_MON_SEL_AG*n*_METRIC_SELECT*m*_REG to select the following metrics for read instantaneous bandwidth measurement: time duration and data amount.
3. When measuring instantaneous bandwidth, software specifies the time duration for each measurement segment. After reaching the measurement duration, software can read the data statistics result register to obtain the data amount transferred during that time duration. At the same time, counting for the next measurement segment begins. Configure register AXI_PERF_MON_SEL_AG*n*_RD_INS_BANDW_TIME_THR_REG to determine the measurement time duration. Configure AXI_PERF_MON_SEL_AG0_RD_INS_BANDW_TEST_EN or AXI_PERF_MON_SEL_AG1_RD_INS_BANDW_TEST_EN to enable instantaneous bandwidth measurement.

4. When measuring instantaneous bandwidth, if you wish to report abnormal conditions where port bandwidth drops sharply in dense data transfer scenarios, configure AXI_PERF_MON_SEL_AG*n*_SW_RECORD_STOP_RD_DATA_LIMIT to set the data amount lower threshold for bandwidth. Once the counted instantaneous bandwidth data amount is less than this lower limit, AXI Perf Mon will generate an interrupt and stop measuring instantaneous bandwidth in the AXI Perf Mon Unit.
5. Recorded event information for instantaneous bandwidth can be queried from registers AXI_PERF_MON_SEL_AG*n*_INS_BANDW_DATA*m*_REG and AXI_PERF_MON_SEL_AG*n*_RD_AXI_INFO_RECORD*m*_REG. These register values are updated according to real-time counting conditions. When an interrupt occurs, these registers will pause updating to preserve the scene. AXI_PERF_MON_SEL_AG0_SW_RECORD_STOP_EN or AXI_PERF_MON_SEL_AG1_SW_RECORD_STOP_EN can be used to stop register updates. AXI_PERF_MON_SEL_AG0_SW_RECORD_STOP_CLR or AXI_PERF_MON_SEL_AG1_SW_RECORD_STOP_CLR can be used to clear the stagnation state and resume recording updates.
6. When measuring instantaneous bandwidth, address filtering for data is supported. Address filtering is performed on a per-region basis. Three regions are filtered using AXI_PERF_MON_SEL_AG*n*_RD_ADDR_MASK*m* and AXI_PERF_MON_SEL_AG*n*_RD_ADDR_FILTER*m*. The valid destination addresses among these regions are then selected via AXI_PERF_MON_SEL_AG0_RD_ADDR_REGION_SEL or AXI_PERF_MON_SEL_AG1_RD_ADDR_REGION_SEL.
7. Instantaneous bandwidth statistics support presenting bandwidth distribution systematically through range statistics. Configure the latter 3 bits of every 8 bits in register AXI_PERF_MON_SEL_AG*n*_METRIC_SELECT*m*_REG to specify the statistics metric for range counters. AXI_PERF_MON__SEL_AG*n*_METRIC_RANGE_CNT_LOW and AXI_PERF_MON__SEL_AG*n*_METRIC_RANGE_CNT_HIGH are used to specify the upper and lower limits for each range. Range statistics for instantaneous bandwidth can be queried through register AXI_PERF_MON_SEL_AG*n*_RANGE*m*_REG.

23.8 Register Summary

The addresses in this section are relative to AXI Perf Mon base address provided in Table 9.3-2 in Chapter 9 *System and Memory*.

The abbreviations given in Column **Access** are explained in Section *Access Types for Registers*.

Name	Description	Address	Access
Configuration Registers			
AXI_PERF_MON_CLK_EN_REG	Reserved	0x0000	R/W
AXI_PERF_MON_SEL_AG0_METRIC_RANGE0_REG	Configuration register 0 for upper and lower limits of range statistics in monitoring unit 0	0x00C8	R/W
AXI_PERF_MON_SEL_AG1_METRIC_RANGE0_REG	Configuration register 0 for upper and lower limits of range statistics in monitoring unit 1	0x00CC	R/W
AXI_PERF_MON_SEL_AG0_METRIC_RANGE1_REG	Configuration register 1 for upper and lower limits of range statistics in monitoring unit 0	0x00D0	R/W
AXI_PERF_MON_SEL_AG1_METRIC_RANGE1_REG	Configuration register 1 for upper and lower limits of range statistics in monitoring unit 1	0x00D4	R/W
AXI_PERF_MON_SEL_AG0_METRIC_RANGE2_REG	Configuration register 2 for upper and lower limits of range statistics in monitoring unit 0	0x00D8	R/W
AXI_PERF_MON_SEL_AG1_METRIC_RANGE2_REG	Configuration register 2 for upper and lower limits of range statistics in monitoring unit 1	0x00DC	R/W
AXI_PERF_MON_SEL_AG0_METRIC_RANGE3_REG	Configuration register 3 for upper and lower limits of range statistics in monitoring unit 0	0x00E0	R/W
AXI_PERF_MON_SEL_AG1_METRIC_RANGE3_REG	Configuration register 3 for upper and lower limits of range statistics in monitoring unit 1	0x00E4	R/W
AXI_PERF_MON_SEL_AG0_METRIC_RANGE4_REG	Configuration register 4 for upper and lower limits of range statistics in monitoring unit 0	0x00E8	R/W
AXI_PERF_MON_SEL_AG1_METRIC_RANGE4_REG	Configuration register 4 for upper and lower limits of range statistics in monitoring unit 1	0x00EC	R/W
AXI_PERF_MON_SEL_AG0_METRIC_RANGE5_REG	Configuration register 5 for upper and lower limits of range statistics in monitoring unit 0	0x00F0	R/W
AXI_PERF_MON_SEL_AG1_METRIC_RANGE5_REG	Configuration register 5 for upper and lower limits of range statistics in monitoring unit 1	0x00F4	R/W

Name	Description	Address	Access
AXI_PERF_MON_SEL_AG0_RD_ADDR_MASK0_REG	Read address filter mask register for monitoring unit 0 (region 0)	0x00F8	R/W
AXI_PERF_MON_SEL_AG1_RD_ADDR_MASK0_REG	Read address filter mask register for monitoring unit 1 (region 0)	0x00FC	R/W
AXI_PERF_MON_SEL_AG0_RD_ADDR_MASK1_REG	Read address filter mask register for monitoring unit 0 (region 1)	0x0100	R/W
AXI_PERF_MON_SEL_AG1_RD_ADDR_MASK1_REG	Read address filter mask register for monitoring unit 1 (region 1)	0x0104	R/W
AXI_PERF_MON_SEL_AG0_RD_ADDR_MASK2_REG	Read address filter mask register for monitoring unit 0 (region 2)	0x0108	R/W
AXI_PERF_MON_SEL_AG1_RD_ADDR_MASK2_REG	Read address filter mask register for monitoring unit 1 (region 2)	0x010C	R/W
AXI_PERF_MON_SEL_AG0_WR_ADDR_MASK0_REG	Write address filter mask register for monitoring unit 0 (region 0)	0x0110	R/W
AXI_PERF_MON_SEL_AG1_WR_ADDR_MASK0_REG	Write address filter mask register for monitoring unit 1 (region 0)	0x0114	R/W
AXI_PERF_MON_SEL_AG0_WR_ADDR_MASK1_REG	Write address filter mask register for monitoring unit 0 (region 1)	0x0118	R/W
AXI_PERF_MON_SEL_AG1_WR_ADDR_MASK1_REG	Write address filter mask register for monitoring unit 1 (region 1)	0x011C	R/W
AXI_PERF_MON_SEL_AG0_WR_ADDR_MASK2_REG	Write address filter mask register for monitoring unit 0 (region 2)	0x0120	R/W
AXI_PERF_MON_SEL_AG1_WR_ADDR_MASK2_REG	Write address filter mask register for monitoring unit 1 (region 2)	0x0124	R/W
AXI_PERF_MON_SEL_AG0_RD_ADDR_FILTER0_REG	Read address filter value register for monitoring unit 0 (region 0)	0x0128	R/W
AXI_PERF_MON_SEL_AG1_RD_ADDR_FILTER0_REG	Read address filter value register for monitoring unit 1 (region 0)	0x012C	R/W
AXI_PERF_MON_SEL_AG0_RD_ADDR_FILTER1_REG	Read address filter value register for monitoring unit 0 (region 1)	0x0130	R/W
AXI_PERF_MON_SEL_AG1_RD_ADDR_FILTER1_REG	Read address filter value register for monitoring unit 1 (region 1)	0x0134	R/W
AXI_PERF_MON_SEL_AG0_RD_ADDR_FILTER2_REG	Read address filter value register for monitoring unit 0 (region 2)	0x0138	R/W
AXI_PERF_MON_SEL_AG1_RD_ADDR_FILTER2_REG	Read address filter value register for monitoring unit 1 (region 2)	0x013C	R/W
AXI_PERF_MON_SEL_AG0_WR_ADDR_FILTER0_REG	Write address filter value register for monitoring unit 0 (region 0)	0x0140	R/W
AXI_PERF_MON_SEL_AG1_WR_ADDR_FILTER0_REG	Write address filter value register for monitoring unit 1 (region 0)	0x0144	R/W
AXI_PERF_MON_SEL_AG0_WR_ADDR_FILTER1_REG	Write address filter value register for monitoring unit 0 (region 1)	0x0148	R/W
AXI_PERF_MON_SEL_AG1_WR_ADDR_FILTER1_REG	Write address filter value register for monitoring unit 1 (region 1)	0x014C	R/W

Name	Description	Address	Access
AXI_PERF_MON_SEL_AG0_WR_ADDR_FILTER2_REG	Write address filter value register for monitoring unit 0 (region 2)	0x0150	R/W
AXI_PERF_MON_SEL_AG1_WR_ADDR_FILTER2_REG	Write address filter value register for monitoring unit 1 (region 2)	0x0154	R/W
AXI_PERF_MON_SEL_AG0_METRIC_SELECT1_REG	Metric select register 1 for monitoring unit 0	0x0158	R/W
AXI_PERF_MON_SEL_AG1_METRIC_SELECT1_REG	Metric select register 1 for monitoring unit 1	0x015C	R/W
AXI_PERF_MON_SEL_AG0_METRIC_SELECT2_REG	Metric select register 2 for monitoring unit 0	0x0160	R/W
AXI_PERF_MON_SEL_AG1_METRIC_SELECT2_REG	Metric select register 2 for monitoring unit 1	0x0164	R/W
AXI_PERF_MON_SEL_AG_RD_ADDR_REGION_SEL_REG	Selected read region for address filtering	0x0168	R/W
AXI_PERF_MON_SEL_AG_WR_ADDR_REGION_SEL_REG	Selected write region for address filtering	0x016C	R/W
AXI_PERF_MON_SEL_AG_ADDR_FILTER_EN_REG	Address filter enable	0x0170	R/W
AXI_PERF_MON_SEL_AG_SW_RECORD_STOP_EN_REG	Enable pause for instantaneous bandwidth event recording	0x0174	WT
AXI_PERF_MON_SEL_AG_SW_RECORD_STOP_CLR_REG	Resume event recording for instantaneous bandwidth monitoring	0x0178	WT
AXI_PERF_MON_SEL_AG_INS_BANDW_TEST_EN_REG	Instantaneous bandwidth test enable	0x017C	R/W
AXI_PERF_MON_SEL_AG0_SW_RECORD_STOP_DATA_LIMIT_REG	Minimum data amount threshold for instantaneous bandwidth of monitoring unit 0	0x0180	R/W
AXI_PERF_MON_SEL_AG1_SW_RECORD_STOP_DATA_LIMIT_REG	Minimum data amount threshold for instantaneous bandwidth of monitoring unit 1	0x0184	R/W
AXI_PERF_MON_SEL_AG0_ID_MASK_REG	ID filter mask register for monitoring unit 0	0x0188	R/W
AXI_PERF_MON_SEL_AG1_ID_MASK_REG	ID filter mask register for monitoring unit 1	0x018C	R/W
AXI_PERF_MON_SEL_AG0_ID_FILTER_REG	ID filter value register for monitoring unit 0	0x0190	R/W
AXI_PERF_MON_SEL_AG1_ID_FILTER_REG	ID filter value register for monitoring unit 1	0x0194	R/W
AXI_PERF_MON_SEL_AG_BANDW_TEST_EN_REG	Average bandwidth measurement enable	0x0198	R/W
AXI_PERF_MON_SEL_AG_BANDW_TEST_STOP_REG	Average bandwidth measurement stop	0x019C	R/W

Name	Description	Address	Access
AXI_PERF_MON_SEL_AG0_BANDW_TRIGGER_IN_SEL_REG	Trigger source select register for average bandwidth measurement of monitoring unit 0	0x01A0	R/W
AXI_PERF_MON_SEL_AG1_BANDW_TRIGGER_IN_SEL_REG	Trigger source select register for average bandwidth measurement of monitoring unit 1	0x01A4	R/W
AXI_PERF_MON_SEL_AG0_INS_BANDW_TIME_THR_REG	Time duration for instantaneous bandwidth measurement of monitoring unit 0	0x01B8	R/W
AXI_PERF_MON_SEL_AG1_INS_BANDW_TIME_THR_REG	Time duration for instantaneous bandwidth measurement of monitoring unit 1	0x01BC	R/W
Agent Select Register			
AXI_PERF_MON_AGENT_SELECT_REG	Monitored agent select register	0x0004	R/W
Status Registers			
AXI_PERF_MON_SEL_AG0_COUNTER0_REG	Metric count result register 0 for monitoring unit 0	0x0008	RO
AXI_PERF_MON_SEL_AG1_COUNTER0_REG	Metric count result register 0 for monitoring unit 1	0x000C	RO
AXI_PERF_MON_SEL_AG0_COUNTER1_REG	Metric count result register 1 for monitoring unit 0	0x0010	RO
AXI_PERF_MON_SEL_AG1_COUNTER1_REG	Metric count result register 1 for monitoring unit 1	0x0014	RO
AXI_PERF_MON_SEL_AG0_COUNTER2_REG	Metric count result register 2 for monitoring unit 0	0x0018	RO
AXI_PERF_MON_SEL_AG1_COUNTER2_REG	Metric count result register 2 for monitoring unit 1	0x001C	RO
AXI_PERF_MON_SEL_AG0_COUNTER3_REG	Metric count result register 3 for monitoring unit 0	0x0020	RO
AXI_PERF_MON_SEL_AG1_COUNTER3_REG	Metric count result register 3 for monitoring unit 1	0x0024	RO
AXI_PERF_MON_SEL_AG0_COUNTER4_REG	Metric count result register 4 for monitoring unit 0	0x0028	RO
AXI_PERF_MON_SEL_AG1_COUNTER4_REG	Metric count result register 4 for monitoring unit 1	0x002C	RO
AXI_PERF_MON_SEL_AG0_COUNTER5_REG	Metric count result register 5 for monitoring unit 0	0x0030	RO
AXI_PERF_MON_SEL_AG1_COUNTER5_REG	Metric count result register 5 for monitoring unit 1	0x0034	RO
AXI_PERF_MON_SEL_AG0_RANGE0_REG	Range count result register 0 for monitoring unit 0	0x0038	RO
AXI_PERF_MON_SEL_AG1_RANGE0_REG	Range count result register 0 for monitoring unit 1	0x003C	RO

Name	Description	Address	Access
AXI_PERF_MON_SEL_AG0_RANGE1_REG	Range count result register 1 for monitoring unit 0	0x0040	RO
AXI_PERF_MON_SEL_AG1_RANGE1_REG	Range count result register 1 for monitoring unit 1	0x0044	RO
AXI_PERF_MON_SEL_AG0_RANGE2_REG	Range count result register 2 for monitoring unit 0	0x0048	RO
AXI_PERF_MON_SEL_AG1_RANGE2_REG	Range count result register 2 for monitoring unit 1	0x004C	RO
AXI_PERF_MON_SEL_AG0_RANGE3_REG	Range count result register 3 for monitoring unit 0	0x0050	RO
AXI_PERF_MON_SEL_AG1_RANGE3_REG	Range count result register 3 for monitoring unit 1	0x0054	RO
AXI_PERF_MON_SEL_AG0_RANGE4_REG	Range count result register 4 for monitoring unit 0	0x0058	RO
AXI_PERF_MON_SEL_AG1_RANGE4_REG	Range count result register 4 for monitoring unit 1	0x005C	RO
AXI_PERF_MON_SEL_AG0_RANGE5_REG	Range count result register 5 for monitoring unit 0	0x0060	RO
AXI_PERF_MON_SEL_AG1_RANGE5_REG	Range count result register 5 for monitoring unit 1	0x0064	RO
AXI_PERF_MON_SEL_AG0_RD_AXI_INFO_RECORD0_REG	AXI read transaction information record register 0 for monitoring unit 0	0x0068	RO
AXI_PERF_MON_SEL_AG1_RD_AXI_INFO_RECORD0_REG	AXI read transaction information record register 0 for monitoring unit 1	0x006C	RO
AXI_PERF_MON_SEL_AG0_RD_AXI_INFO_RECORD1_REG	AXI read transaction information record register 1 for monitoring unit 0	0x0070	RO
AXI_PERF_MON_SEL_AG1_RD_AXI_INFO_RECORD1_REG	AXI read transaction information record register 1 for monitoring unit 1	0x0074	RO
AXI_PERF_MON_SEL_AG0_RD_AXI_INFO_RECORD2_REG	AXI read transaction information record register 2 for monitoring unit 0	0x0078	RO
AXI_PERF_MON_SEL_AG1_RD_AXI_INFO_RECORD2_REG	AXI read transaction information record register 2 for monitoring unit 1	0x007C	RO
AXI_PERF_MON_SEL_AG0_RD_AXI_INFO_RECORD3_REG	AXI read transaction information record register 3 for monitoring unit 0	0x0080	RO
AXI_PERF_MON_SEL_AG1_RD_AXI_INFO_RECORD3_REG	AXI read transaction information record register 3 for monitoring unit 1	0x0084	RO

Name	Description	Address	Access
AXI_PERF_MON_SEL_AG0_WR_AXI_INFO_RECORD0_REG	AXI write transaction information record register 0 for monitoring unit 0	0x0088	RO
AXI_PERF_MON_SEL_AG1_WR_AXI_INFO_RECORD0_REG	AXI write transaction information record register 0 for monitoring unit 1	0x008C	RO
AXI_PERF_MON_SEL_AG0_WR_AXI_INFO_RECORD1_REG	AXI write transaction information record register 1 for monitoring unit 0	0x0090	RO
AXI_PERF_MON_SEL_AG1_WR_AXI_INFO_RECORD1_REG	AXI write transaction information record register 1 for monitoring unit 1	0x0094	RO
AXI_PERF_MON_SEL_AG0_WR_AXI_INFO_RECORD2_REG	AXI write transaction information record register 2 for monitoring unit 0	0x0098	RO
AXI_PERF_MON_SEL_AG1_WR_AXI_INFO_RECORD2_REG	AXI write transaction information record register 2 for monitoring unit 1	0x009C	RO
AXI_PERF_MON_SEL_AG0_WR_AXI_INFO_RECORD3_REG	AXI write transaction information record register 3 for monitoring unit 0	0x00A0	RO
AXI_PERF_MON_SEL_AG1_WR_AXI_INFO_RECORD3_REG	AXI write transaction information record register 3 for monitoring unit 1	0x00A4	RO
AXI_PERF_MON_SEL_AG0_INS_BANDW_DATA0_REG	Instantaneous bandwidth record register 0 for monitoring unit 0	0x00A8	RO
AXI_PERF_MON_SEL_AG1_INS_BANDW_DATA0_REG	Instantaneous bandwidth record register 0 for monitoring unit 1	0x00AC	RO
AXI_PERF_MON_SEL_AG0_INS_BANDW_DATA1_REG	Instantaneous bandwidth record register 1 for monitoring unit 0	0x00B0	RO
AXI_PERF_MON_SEL_AG1_INS_BANDW_DATA1_REG	Instantaneous bandwidth record register 1 for monitoring unit 1	0x00B4	RO
AXI_PERF_MON_SEL_AG0_INS_BANDW_DATA2_REG	Instantaneous bandwidth record register 2 for monitoring unit 0	0x00B8	RO
AXI_PERF_MON_SEL_AG1_INS_BANDW_DATA2_REG	Instantaneous bandwidth record register 2 for monitoring unit 1	0x00BC	RO
AXI_PERF_MON_SEL_AG0_INS_BANDW_DATA3_REG	Instantaneous bandwidth record register 3 for monitoring unit 0	0x00C0	RO
AXI_PERF_MON_SEL_AG1_INS_BANDW_DATA3_REG	Instantaneous bandwidth record register 3 for monitoring unit 1	0x00C4	RO
AXI_PERF_MON_SEL_AG0_WR_BANDW_CNT_VALID_-STROBE_NUM_REG	Counting interval register for write average bandwidth measurement of monitoring unit 0	0x01A8	R/W

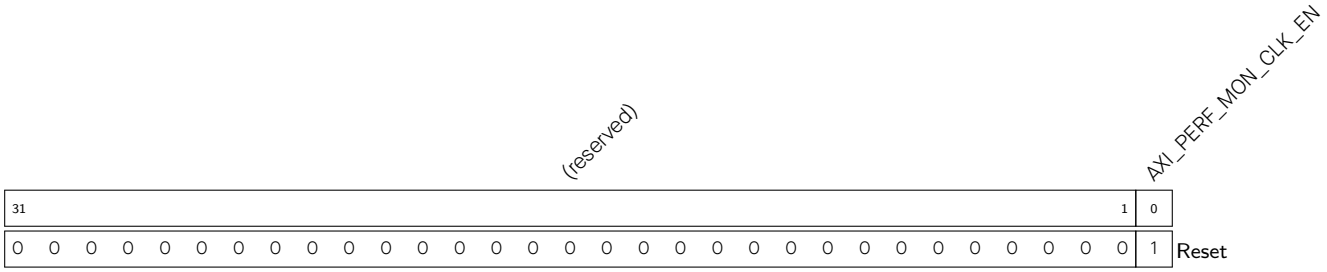
Name	Description	Address	Access
AXI_PERF_MON_SEL_AG1_WR_BANDW_CNT_VALID_STROBE_NUM_REG	Counting interval register for write average bandwidth measurement of monitoring unit 1	0x01AC	R/W
AXI_PERF_MON_SEL_AG0_RD_BANDW_CNT_VALID_STROBE_NUM_REG	Counting interval register for read average bandwidth measurement of monitoring unit 0	0x01B0	R/W
AXI_PERF_MON_SEL_AG1_RD_BANDW_CNT_VALID_STROBE_NUM_REG	Counting interval register for read average bandwidth measurement of monitoring unit 1	0x01B4	R/W
Interrupt Registers			
AXI_PERF_MON_SEL_AG_INT_RAW_REG	Raw interrupt register	0x01C0	R/WTC/SS
AXI_PERF_MON_SEL_AG_INT_ST_REG	Interrupt status register	0x01C4	RO
AXI_PERF_MON_SEL_AG_INT_ENA_REG	Interrupt enable register	0x01C8	R/W
AXI_PERF_MON_SEL_AG_INT_CLR_REG	Interrupt clear register	0x01CC	WT

23.9 Registers

The addresses in this section are relative to AXI Perf Mon base address provided in Table 9.3-2 in Chapter 9 *System and Memory*.

For how to program reserved fields, please refer to Section *Programming Reserved Register Field*.

Register 23.1. AXI_PERF_MON_CLK_EN_REG (0x0000)



AXI_PERF_MON_CLK_EN Configures whether to force enable the AXI performance monitor register module clock.

0: AXI performance monitor register module clock is automatically enabled or disabled

1: Force enable the AXI performance monitor register module clock

(R/W)

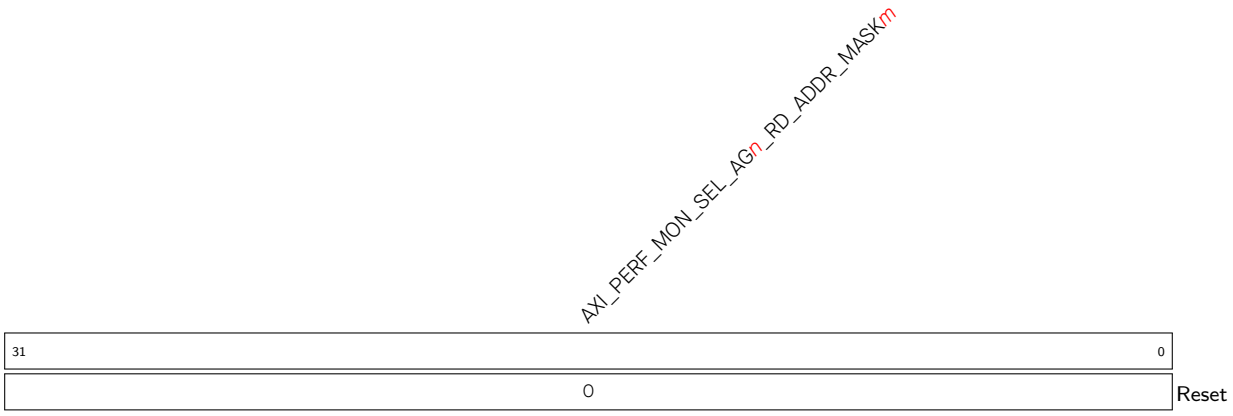
Register 23.2. AXI_PERF_MON_SEL_AG*n*_METRIC_RANGE*m*_REG (*n*: 0-1, *m*: 0-5) (0x00C8+0x4*n*+0x8*m*)



AXI_PERF_MON_SEL_AG*n*_METRIC_RANGE*m*_CNT_HIGH (*n*: 0-1, *m*: 0-5) Configures the upper limit of the range statistics for the selected metric in the selected monitoring unit. (R/W)

AXI_PERF_MON_SEL_AG*n*_METRIC_RANGE*m*_CNT_LOW (*n*: 0-1, *m*: 0-5) Configures the lower limit of the range statistics for the selected metric in the selected monitoring unit. (R/W)

Register 23.3. AXI_PERF_MON_SEL_AG*n*_RD_ADDR_MASK*m*_REG (*n*: 0-1, *m*: 0-2) (0x00F8+0x4n*+0x8**m*)**



AXI_PERF_MON_SEL_AG*n*_RD_ADDR_MASK*m* (*n*: 0-1, *m*: 0-2) Configures the read address mask for the address filter function of the selected monitoring unit. Masked bits are not compared. Valid only when AXI_PERF_MON_SEL_AG*n*_RD_ADDR_FILTER_EN is 1.
(R/W)

Register 23.4. AXI_PERF_MON_SEL_AG*n*_WR_ADDR_MASK*m*_REG (*n*: 0-1, *m*: 0-2) (0x0110+0x4n*+0x8**m*)**



AXI_PERF_MON_SEL_AG*n*_WR_ADDR_MASK*m* (*n*: 0-1, *m*: 0-2) Configures the write address mask for the address filter function of the selected monitoring unit. Masked bits are not compared.
Valid only when AXI_PERF_MON_SEL_AG*n*_WR_ADDR_FILTER_EN is 1.
(R/W)

Register 23.5. AXI_PERF_MON_SEL_AG*n*_RD_ADDR_FILTER*m*_REG (*n*: 0-1, *m*: 0-2) (0x0128+0x4n*+0x8**m*)**



AXI_PERF_MON_SEL_AG*n*_RD_ADDR_FILTER*m* (*n*: 0-1, *m*: 0-2) Configures the read address filter value for the selected monitoring unit. Bits that are not masked are compared with this value. Valid only when AXI_PERF_MON_SEL_AG*n*_RD_ADDR_FILTER_EN is 1.

(R/W)

Register 23.6. AXI_PERF_MON_SEL_AG*n*_WR_ADDR_FILTER*m*_REG (*n*: 0-1, *m*: 0-2) (0x0140+0x4n*+0x8**m*)**



AXI_PERF_MON_SEL_AG*n*_WR_ADDR_FILTER*m* (*n*: 0-1, *m*: 0-2) Configures the write address filter value for the selected monitoring unit. Bits that are not masked are compared with this value. Valid only when AXI_PERF_MON_SEL_AG*n*_WR_ADDR_FILTER_EN is 1.

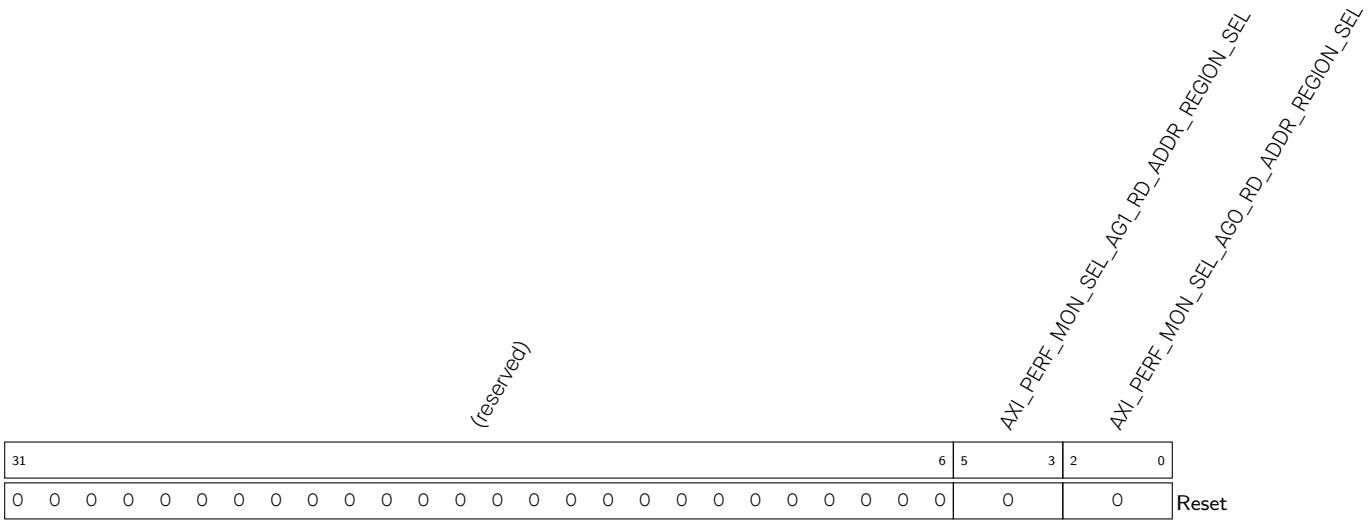
(R/W)

Register 23.7. AXI_PERF_MON_SEL_AG*n*_METRIC_SELECT*m*_REG (*n*: 0-1, *m*: 1-2) (0x0158+0x4*n*+0x8*(*m*-1))



AXI_PERF_MON_SEL_AG*n*_METRIC_SELECT*m* (*n*: 0-1, *m*: 1-2) Configures metric selection for the selected monitoring unit. Each counter uses 8 bits. In each 8-bit group, bits [4:0] select the measurement metric and bits [7:5] select the range statistics metric. Metric measurement and range statistics can be performed at the same time. (R/W)

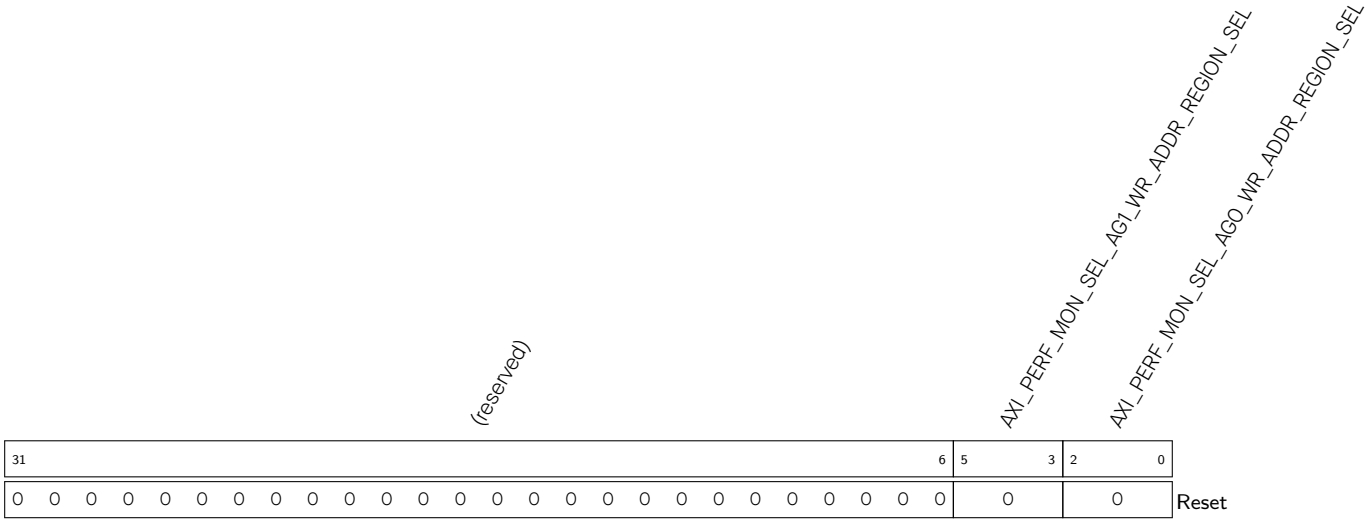
Register 23.8. AXI_PERF_MON_SEL_AG_RD_ADDR_REGION_SEL_REG (0x0168)



AXI_PERF_MON_SEL_AG0_RD_ADDR_REGION_SEL Configures the read region selection for monitoring unit 0. It operates in conjunction with the address mask and filter values. Only transactions with addresses falling within the selected region are counted. (R/W)

AXI_PERF_MON_SEL_AG1_RD_ADDR_REGION_SEL Configures the read region selection for monitoring unit 1. It operates in conjunction with the address mask and filter values. Only transactions with addresses falling within the selected region are counted. (R/W)

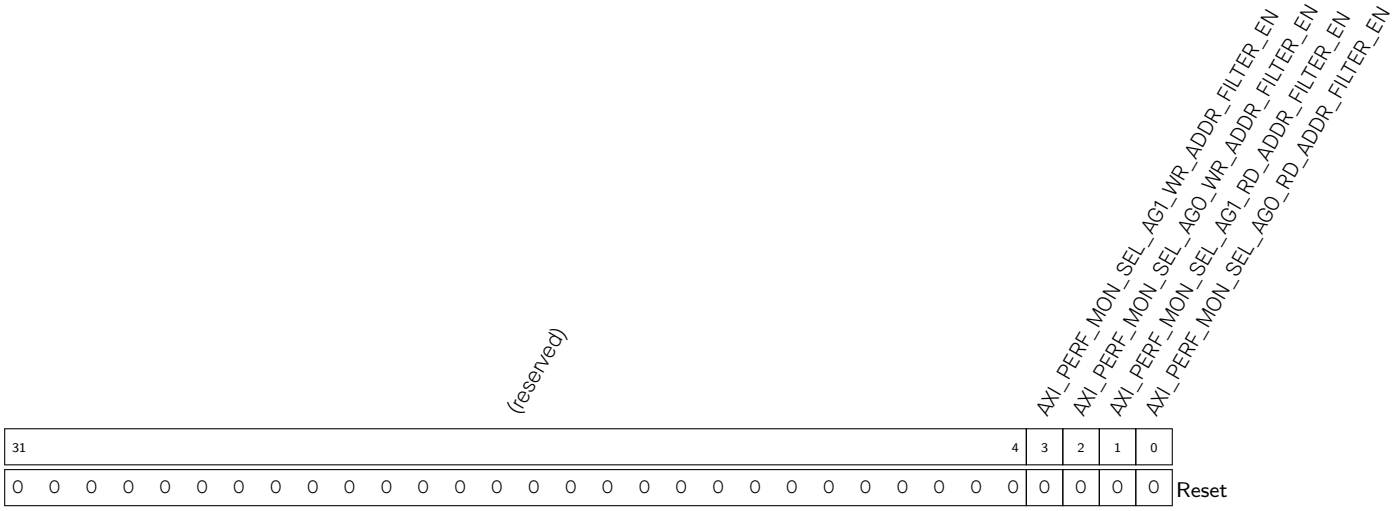
Register 23.9. AXI_PERF_MON_SEL_AG_WR_ADDR_REGION_SEL_REG (0x016C)



AXI_PERF_MON_SEL_AG0_WR_ADDR_REGION_SEL Configures the write region selection for monitoring unit 0. It operates in conjunction with the address mask and filter values. Only transactions with addresses falling within the selected region are counted. (R/W)

AXI_PERF_MON_SEL_AG1_WR_ADDR_REGION_SEL Configures the write region selection for monitoring unit 1. It operates in conjunction with the address mask and filter values. Only transactions with addresses falling within the selected region are counted. (R/W)

Register 23.10. AXI_PERF_MON_SEL_AG_ADDR_FILTER_EN_REG (0x0170)



AXI_PERF_MON_SEL_AG*n*_RD_ADDR_FILTER_EN (*n*: 0-1) Configures whether or not to enable the read address filter function.

0: Disable. All read address accesses are recorded

1: Enable

(R/W)

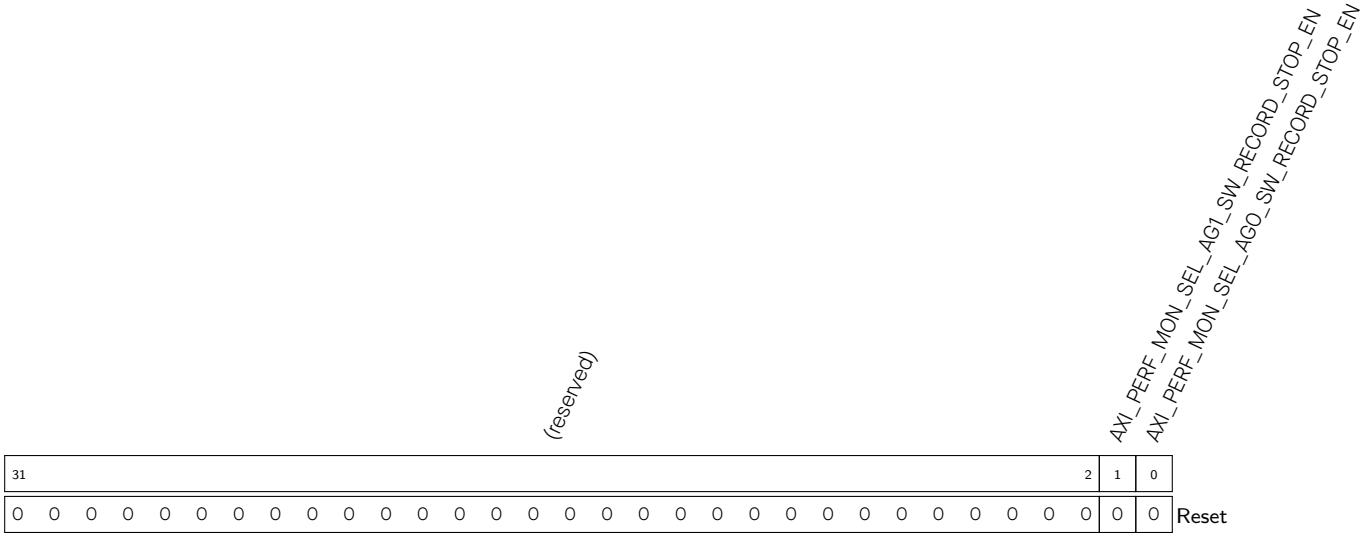
AXI_PERF_MON_SEL_AG*n*_WR_ADDR_FILTER_EN (*n*: 0-1) Configures whether or not to enable the write address filter function.

0: Disable. All write address accesses are recorded

1: Enable

(R/W)

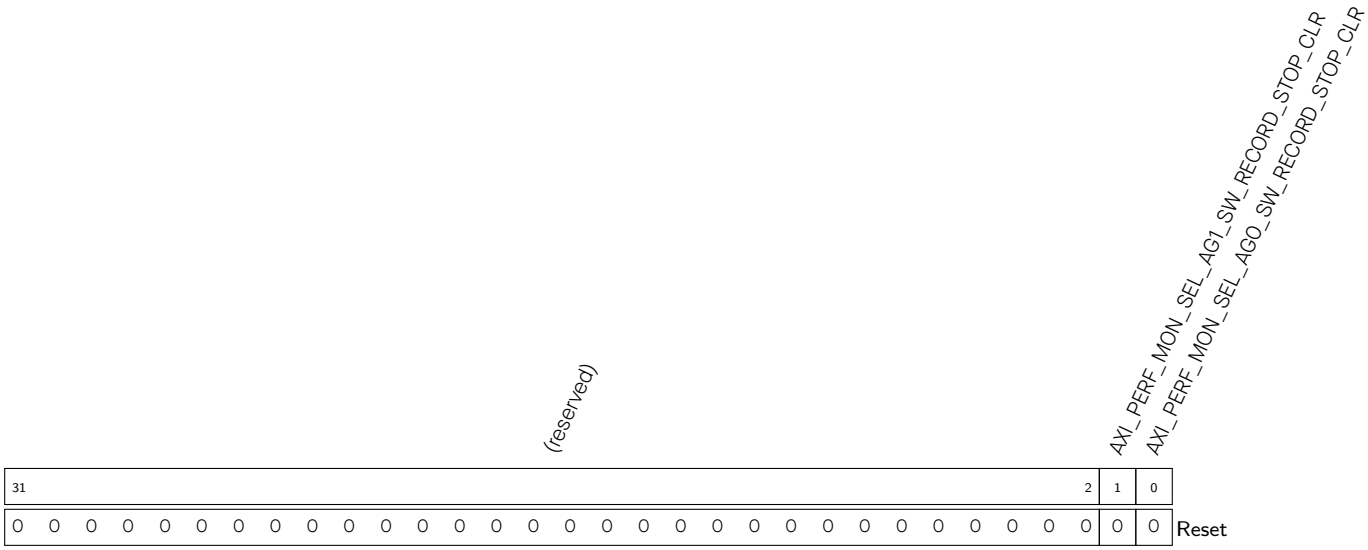
Register 23.11. AXI_PERF_MON_SEL_AG_SW_RECORD_STOP_EN_REG (0x0174)



AXI_PERF_MON_SEL_AG0_SW_RECORD_STOP_EN Write 1 to stop the event log function. The recorded information is kept. (WT)

AXI_PERF_MON_SEL_AG1_SW_RECORD_STOP_EN Write 1 to stop the event log function. The recorded information is kept. (WT)

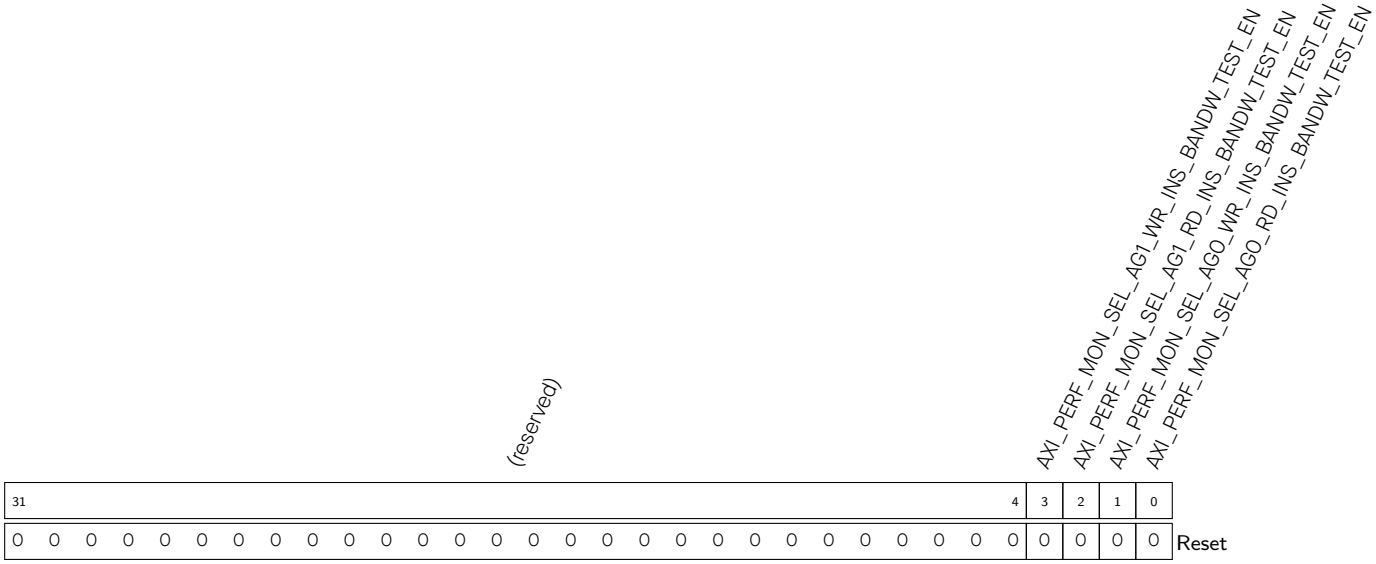
Register 23.12. AXI_PERF_MON_SEL_AG_SW_RECORD_STOP_CLR_REG (0x0178)



AXI_PERF_MON_SEL_AG0_SW_RECORD_STOP_CLR Write 1 to clear the event log stop status. Recording new transactions starts from now. (WT)

AXI_PERF_MON_SEL_AG1_SW_RECORD_STOP_CLR Write 1 to clear the event log stop status. Recording new transactions starts from now. (WT)

Register 23.13. AXI_PERF_MON_SEL_AG_INS_BANDW_TEST_EN_REG (0x017C)



AXI_PERF_MON_SEL_AG0_RD_INS_BANDW_TEST_EN Configures whether or not to enable the read instantaneous bandwidth test for monitoring unit 0.

0: Disable

1: Enable

(R/W)

AXI_PERF_MON_SEL_AG0_WR_INS_BANDW_TEST_EN Configures whether or not to enable the write instantaneous bandwidth test for monitoring unit 0.

0: Disable

1: Enable

(R/W)

AXI_PERF_MON_SEL_AG1_RD_INS_BANDW_TEST_EN Configures whether or not to enable the read instantaneous bandwidth test for monitoring unit 1.

0: Disable

1: Enable

(R/W)

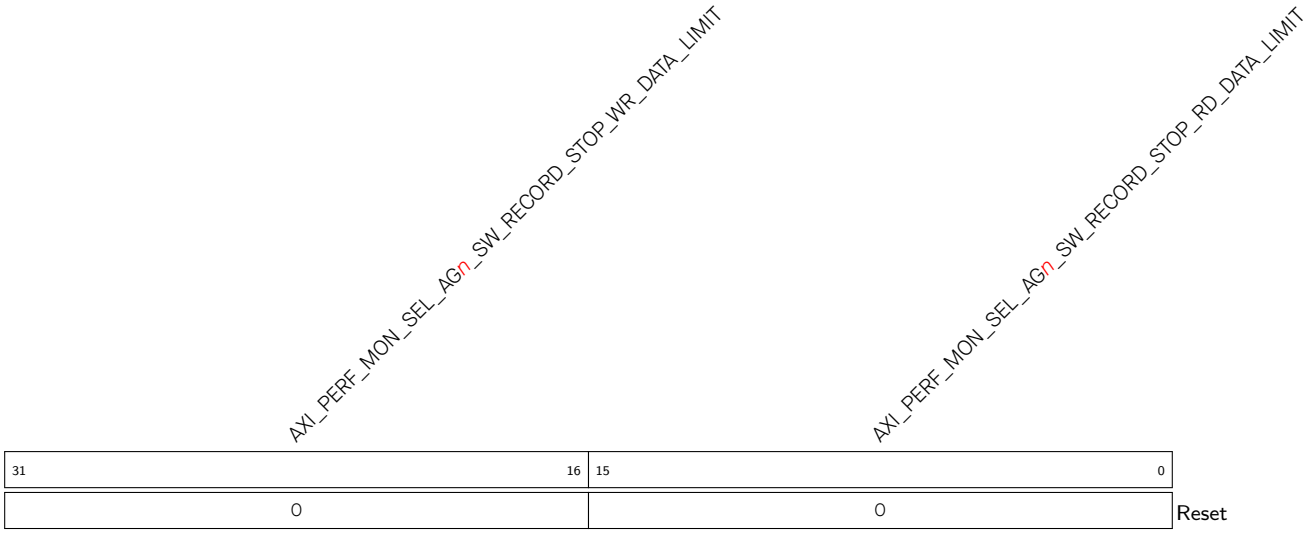
AXI_PERF_MON_SEL_AG1_WR_INS_BANDW_TEST_EN Configures whether or not to enable the write instantaneous bandwidth test for monitoring unit 1.

0: Disable

1: Enable

(R/W)

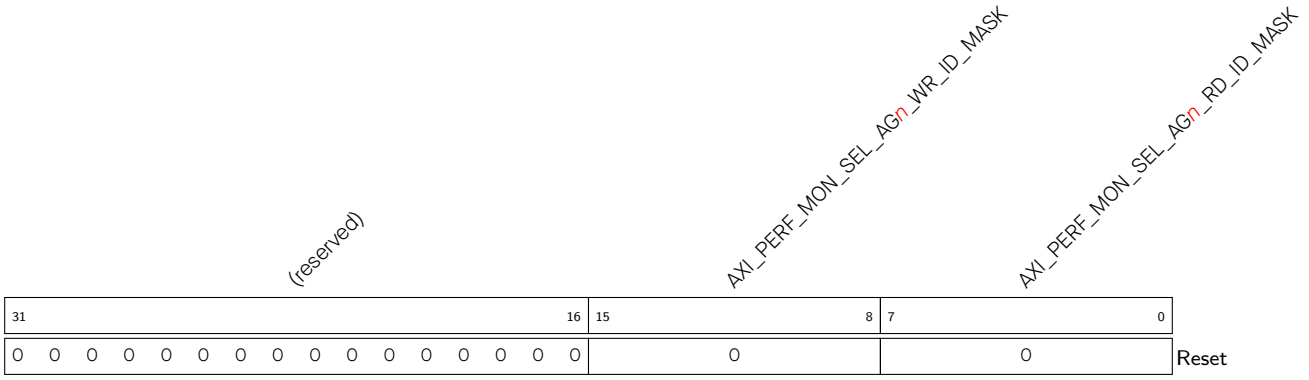
Register 23.14. AXI_PERF_MON_SEL_AG*n*_SW_RECORD_STOP_DATA_LIMIT_REG (*n*: 0-1) (0x0180+0x4n*)**



AXI_PERF_MON_SEL_AG*n*_SW_RECORD_STOP_RD_DATA_LIMIT (*n*: 0-1) Configures the lower threshold for read instantaneous bandwidth measurement. When the measured data reaches this lower limit, the interrupt is enabled. (R/W)

AXI_PERF_MON_SEL_AG*n*_SW_RECORD_STOP_WR_DATA_LIMIT (*n*: 0-1) Configures the lower threshold for write instantaneous bandwidth measurement. When the measured data reaches this lower limit, the interrupt is enabled. (R/W)

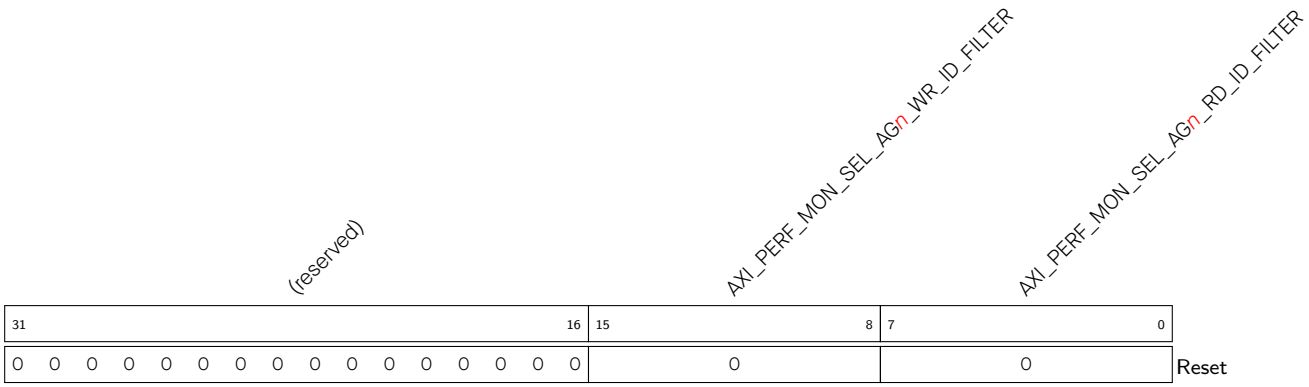
Register 23.15. AXI_PERF_MON_SEL_AG*n*_ID_MASK_REG (*n*: 0-1) (0x0188+0x4n*)**



AXI_PERF_MON_SEL_AG*n*_RD_ID_MASK (*n*: 0-1) Configures the read ID mask. Masked ID bits are not compared. (R/W)

AXI_PERF_MON_SEL_AG*n*_WR_ID_MASK (*n*: 0-1) Configures the write ID mask. Masked ID bits are not compared. (R/W)

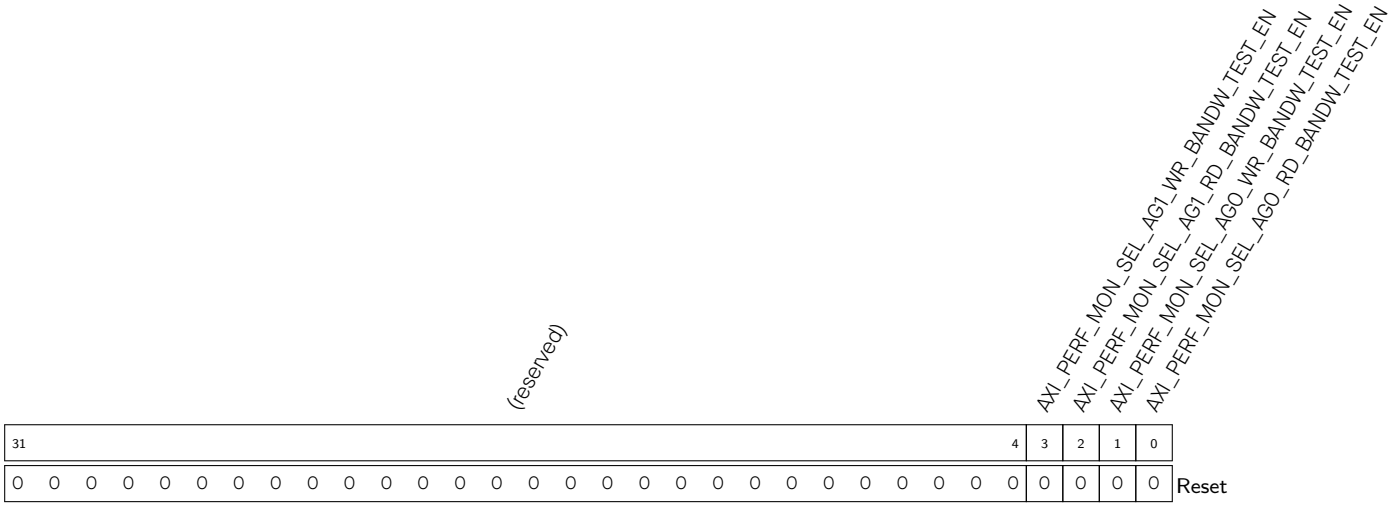
Register 23.16. AXI_PERF_MON_SEL_AG*n*_ID_FILTER_REG (*n*: 0-1) (0x0190+0x4n*)**



AXI_PERF_MON_SEL_AG*n*_RD_ID_FILTER (*n*: 0-1) Configures the read ID filter value. Bits that are not masked must match this value to pass the filter. (R/W)

AXI_PERF_MON_SEL_AG*n*_WR_ID_FILTER (*n*: 0-1) Configures the write ID filter value. Bits that are not masked must match this value to pass the filter. (R/W)

Register 23.17. AXI_PERF_MON_SEL_AG_BANDW_TEST_EN_REG (0x0198)



AXI_PERF_MON_SEL_AG0_RD_BANDW_TEST_EN Configures whether or not to enable the read average bandwidth test for monitoring unit 0.

0: Disable

1: Enable

(R/W)

AXI_PERF_MON_SEL_AG0_WR_BANDW_TEST_EN Configures whether or not to enable the write average bandwidth test for monitoring unit 0.

0: Disable

1: Enable

(R/W)

AXI_PERF_MON_SEL_AG1_RD_BANDW_TEST_EN Configures whether or not to enable the read average bandwidth test for monitoring unit 1.

0: Disable

1: Enable

(R/W)

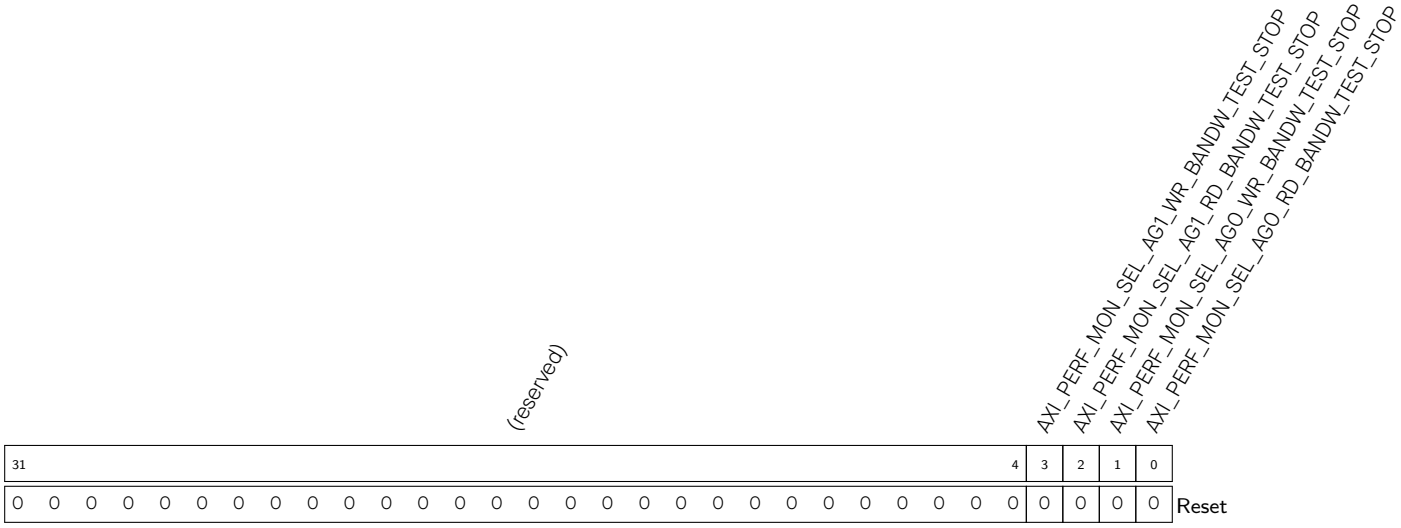
AXI_PERF_MON_SEL_AG1_WR_BANDW_TEST_EN Configures whether or not to enable the write average bandwidth test for monitoring unit 1.

0: Disable

1: Enable

(R/W)

Register 23.18. AXI_PERF_MON_SEL_AG_BANDW_TEST_STOP_REG (0x019C)



AXI_PERF_MON_SEL_AG0_RD_BANDW_TEST_STOP Configures whether or not to stop the read average bandwidth test for monitoring unit 0.

0: Continue

1: Stop

(R/W)

AXI_PERF_MON_SEL_AG0_WR_BANDW_TEST_STOP Configures whether or not to stop the write average bandwidth test for monitoring unit 0.

0: Continue

1: Stop

(R/W)

AXI_PERF_MON_SEL_AG1_RD_BANDW_TEST_STOP Configures whether or not to stop the read average bandwidth test for monitoring unit 1.

0: Continue

1: Stop

(R/W)

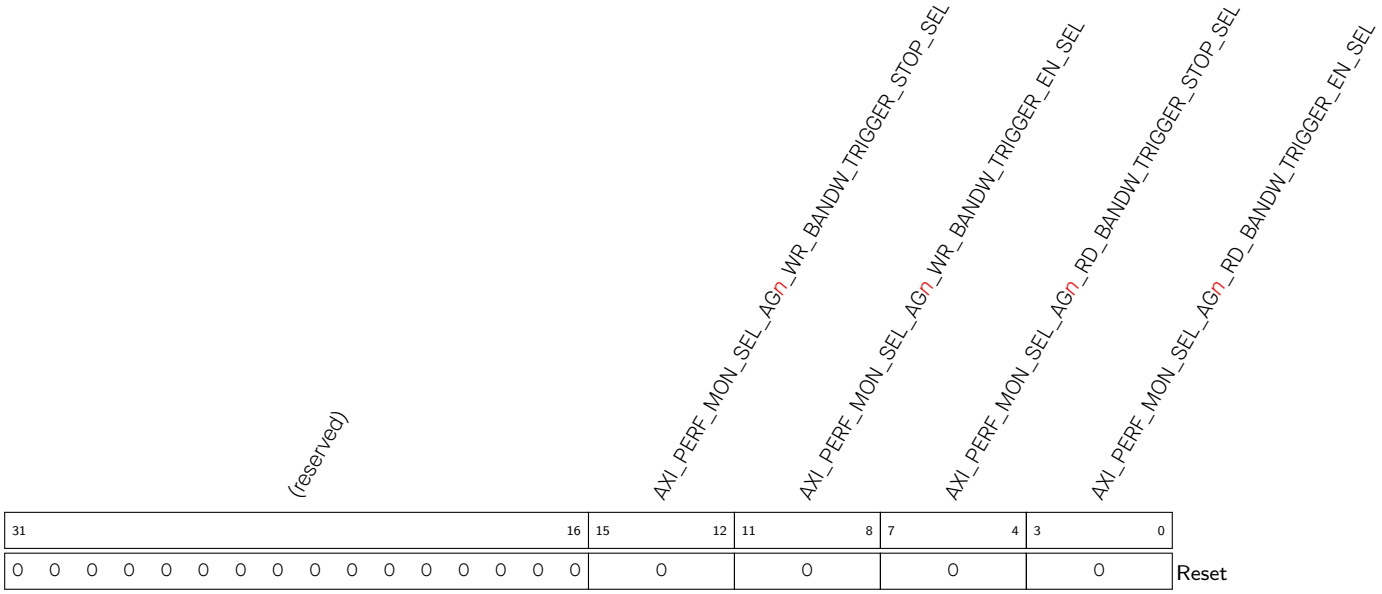
AXI_PERF_MON_SEL_AG1_WR_BANDW_TEST_STOP Configures whether or not to stop the write average bandwidth test for monitoring unit 1.

0: Continue

1: Stop

(R/W)

Register 23.19. AXI_PERF_MON_SEL_AG*n*_BANDW_TRIGGER_IN_SEL_REG (*n*: 0-1) (0x01A0+0x4n*)**



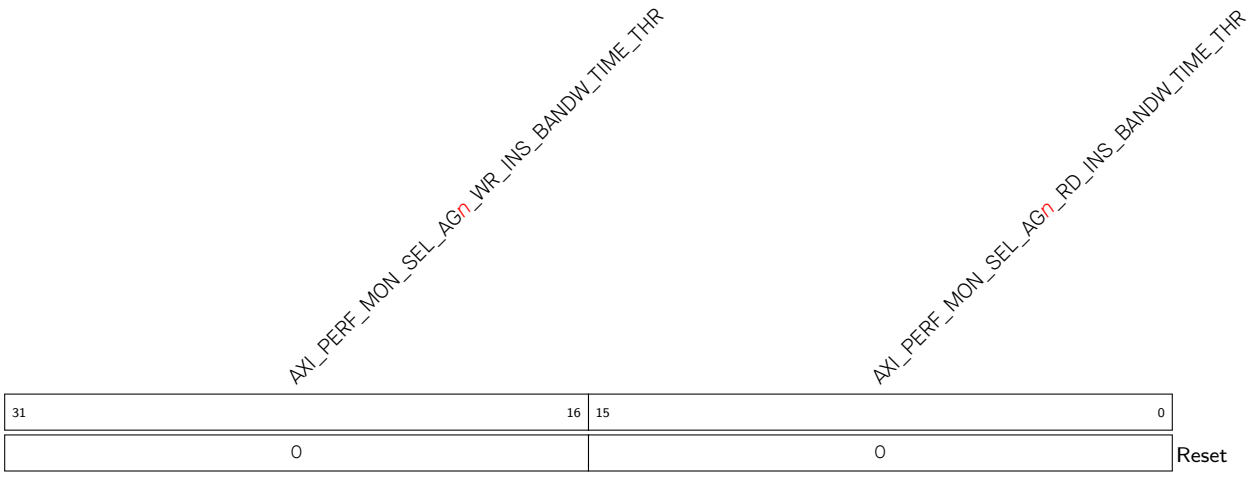
AXI_PERF_MON_SEL_AG*n*_RD_BANDW_TRIGGER_EN_SEL (*n*: 0-1) Configures the trigger enable source for the read average bandwidth test. The trigger enable signal can be sourced from the SoC hardware or issued via software configuration. (R/W)

AXI_PERF_MON_SEL_AG*n*_RD_BANDW_TRIGGER_STOP_SEL (*n*: 0-1) Configures the trigger stop source for the read average bandwidth test. The stop trigger can be sourced from the SoC hardware or via software configuration. (R/W)

AXI_PERF_MON_SEL_AG*n*_WR_BANDW_TRIGGER_EN_SEL (*n*: 0-1) Configures the trigger enable source for the write average bandwidth test. The trigger enable signal can be sourced from the SoC hardware or issued via software configuration. (R/W)

AXI_PERF_MON_SEL_AG*n*_WR_BANDW_TRIGGER_STOP_SEL (*n*: 0-1) Configures the trigger stop source for the write average bandwidth test. The stop trigger can be sourced from the SoC hardware or via software configuration. (R/W)

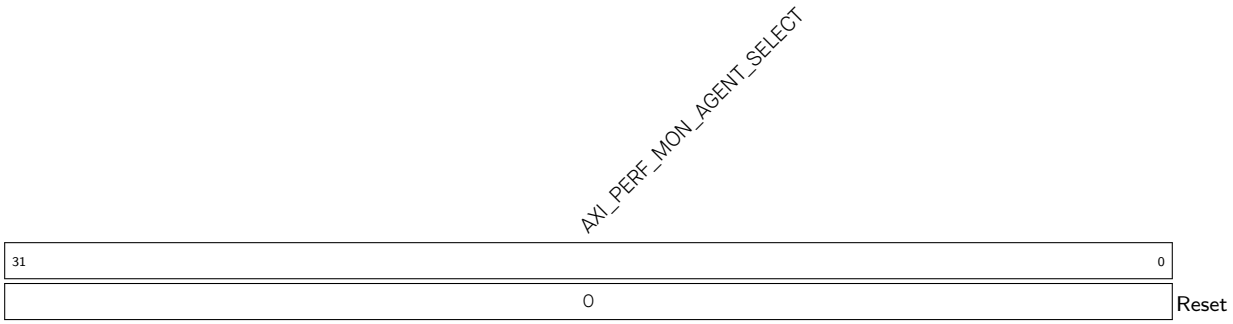
Register 23.20. AXI_PERF_MON_SEL_AG*n*_INS_BANDW_TIME_THR_REG (*n*: 0-1) (0x01B8+0x4**n*)



AXI_PERF_MON_SEL_AG*n*_RD_INS_BANDW_TIME_THR (*n*: 0-1) Configures the time threshold for the read instantaneous bandwidth test. The counter records the data amount within this time window and then refreshes to count again. (R/W)

AXI_PERF_MON_SEL_AG*n*_WR_INS_BANDW_TIME_THR (*n*: 0-1) Configures the time threshold for the write instantaneous bandwidth test. The counter records the data amount within this time window and then refreshes to count again. (R/W)

Register 23.21. AXI_PERF_MON_AGENT_SELECT_REG (0x0004)

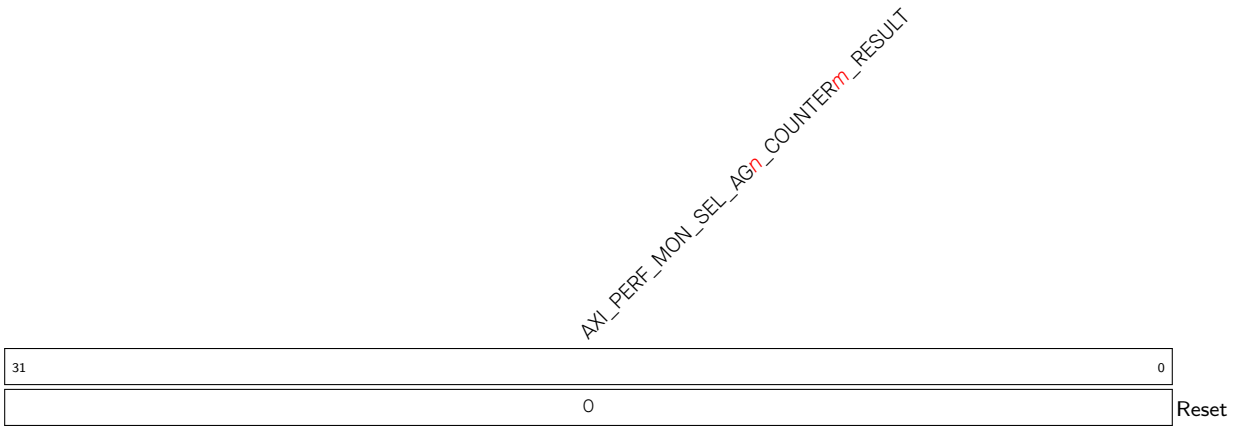


AXI_PERF_MON_AGENT_SELECT Configures the agent selection for each monitoring unit. Each agent selection uses 5 bits.

- Bits [4:0]: Selected agent for monitoring unit 0
- Bits [9:5]: Selected agent for monitoring unit 1
- Bits [31:10]: Reserved

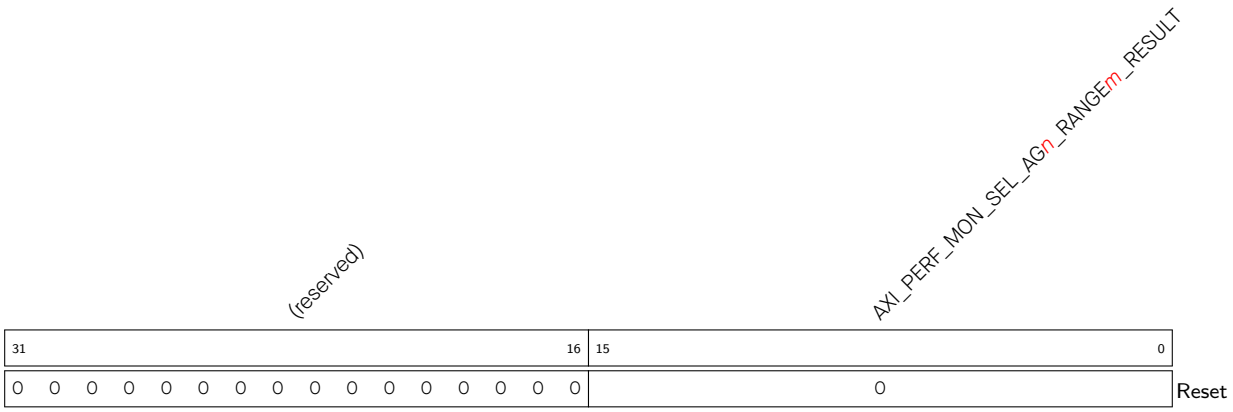
(R/W)

Register 23.22. AXI_PERF_MON_SEL_AG*n*_COUNTER*m*_REG (*n*: 0-1, *m*: 0-5) (0x0008+0x4*n*+0x8*m*)



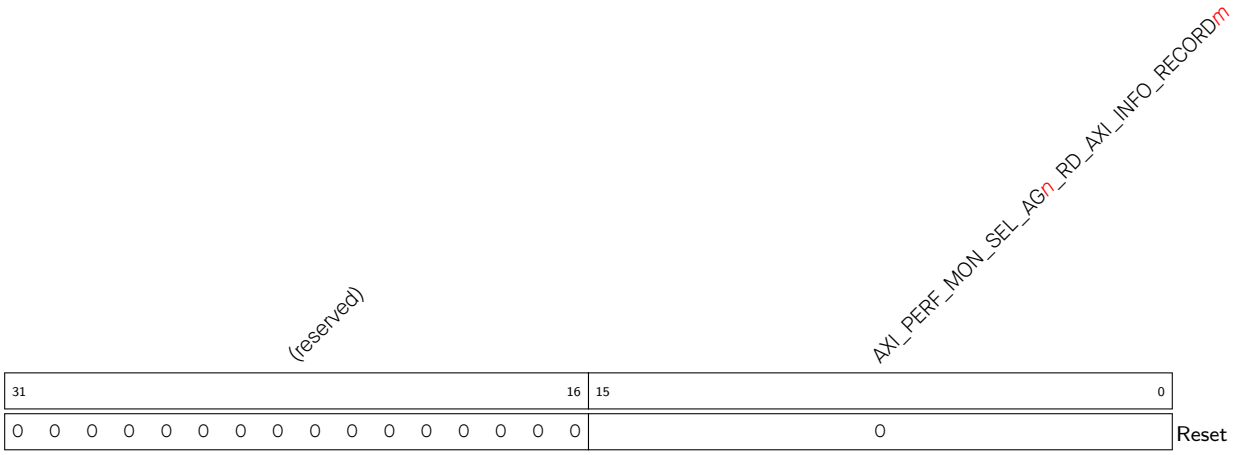
AXI_PERF_MON_SEL_AG*n*_COUNTER*m*_RESULT (*n*: 0-1, *m*: 0-5) Represents the metric measurement result for the selected monitoring unit in this counter. [15:0] is the minimum result and [31:16] is the maximum result. (RO)

Register 23.23. AXI_PERF_MON_SEL_AG*n*_RANGE*m*_REG (*n*: 0-1, *m*: 0-5) (0x0038+0x4*n*+0x8*m*)



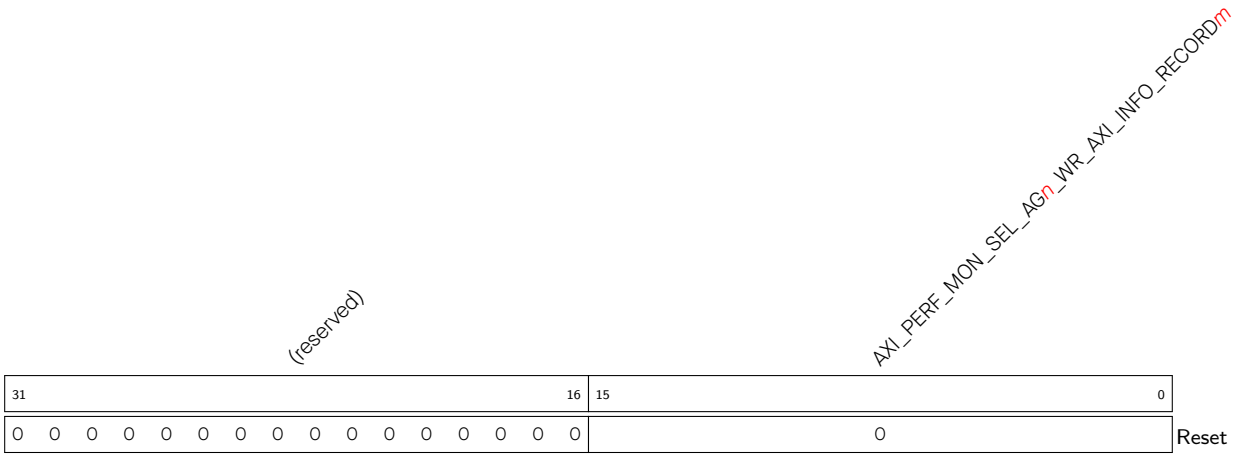
AXI_PERF_MON_SEL_AG*n*_RANGE*m*_RESULT (*n*: 0-1, *m*: 0-5) Represents the range statistics in this counter for the selected monitoring unit. (RO)

Register 23.24. AXI_PERF_MON_SEL_AG*n*_RD_AXI_INFO_RECORD*m*_REG (*n*: 0-1, *m*: 0-3) (0x0068+0x4n*+0x8**m*)**



AXI_PERF_MON_SEL_AG*n*_RD_AXI_INFO_RECORD*m* (*n*: 0-1, *m*: 0-3) Represents the record of the latest AXI read transaction information for the selected monitoring unit. [7:0] is ARLEN and [15:8] is ARID. (RO)

Register 23.25. AXI_PERF_MON_SEL_AG*n*_WR_AXI_INFO_RECORD*m*_REG (*n*: 0-1, *m*: 0-3) (0x0088+0x4n*+0x8**m*)**



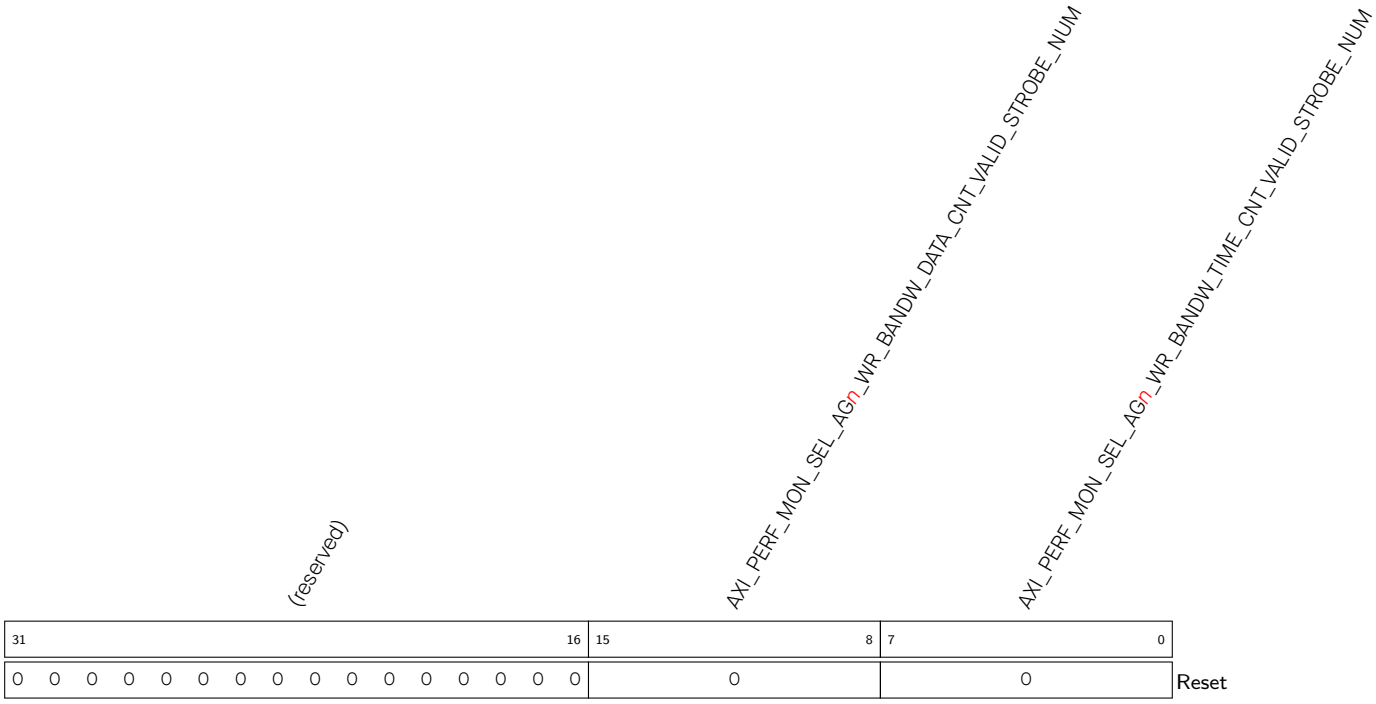
AXI_PERF_MON_SEL_AG*n*_WR_AXI_INFO_RECORD*m* (*n*: 0-1, *m*: 0-3) Represents the record of the latest AXI write transaction information for the selected monitoring unit. [7:0] is AWLEN and [15:8] is AWID. (RO)

Register 23.26. AXI_PERF_MON_SEL_AG*n*_INS_BANDW_DATA*m*_REG (*n*: 0-1, *m*: 0-3) (0x00A8+0x4n*+0x8**m*)**



AXI_PERF_MON_SEL_AG*n*_INS_BANDW_DATA*m* (*n*: 0-1, *m*: 0-3) Represents the latest instantaneous bandwidth measurement within the configured time window for the selected monitoring unit. [15:0] is read and [31:16] is write. (RO)

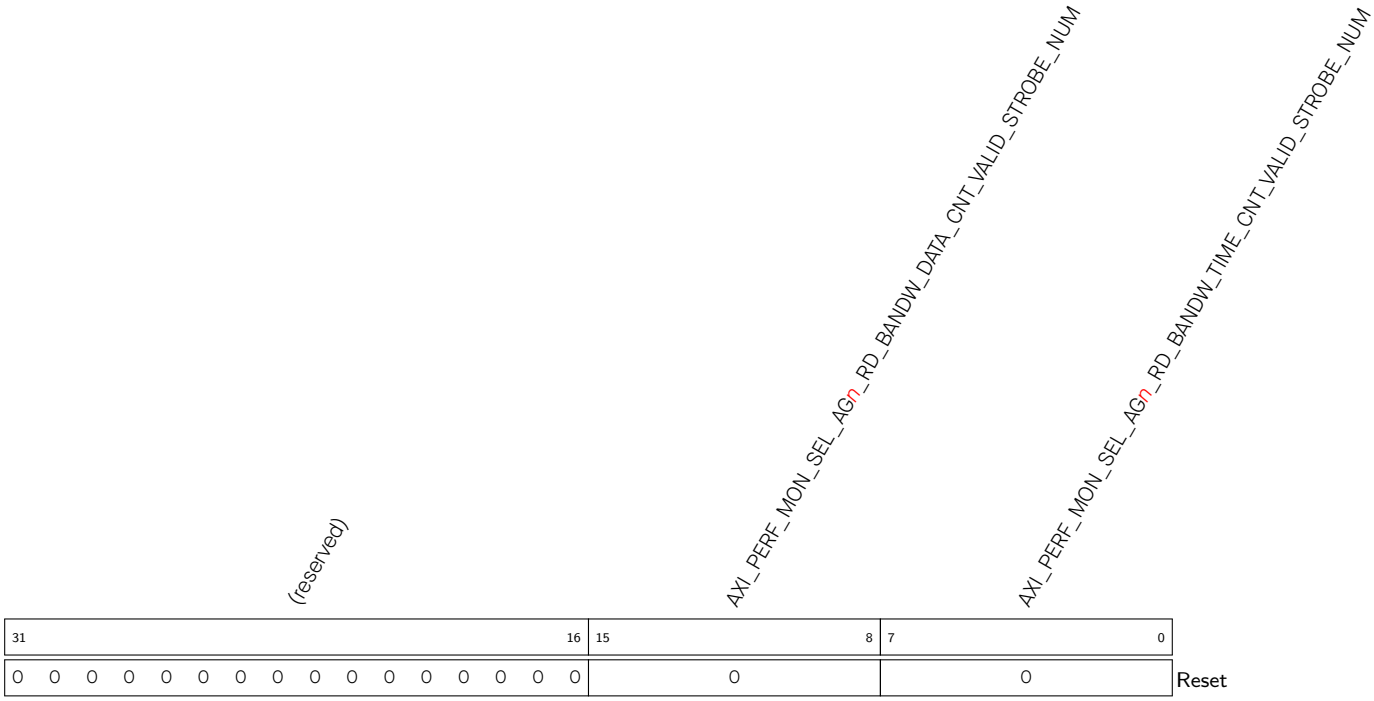
Register 23.27. AXI_PERF_MON_SEL_AG*n*_WR_BANDW_CNT_VALID_STROBE_NUM_REG (*n*: 0-1) (0x01A8+0x4n*)**



AXI_PERF_MON_SEL_AG*n*_WR_BANDW_TIME_CNT_VALID_STROBE_NUM (*n*: 0-1) Represents the scaling factor for the time counter in write bandwidth tests. (R/W)

AXI_PERF_MON_SEL_AG*n*_WR_BANDW_DATA_CNT_VALID_STROBE_NUM (*n*: 0-1) Represents the scaling factor for the data counter in write bandwidth tests. (R/W)

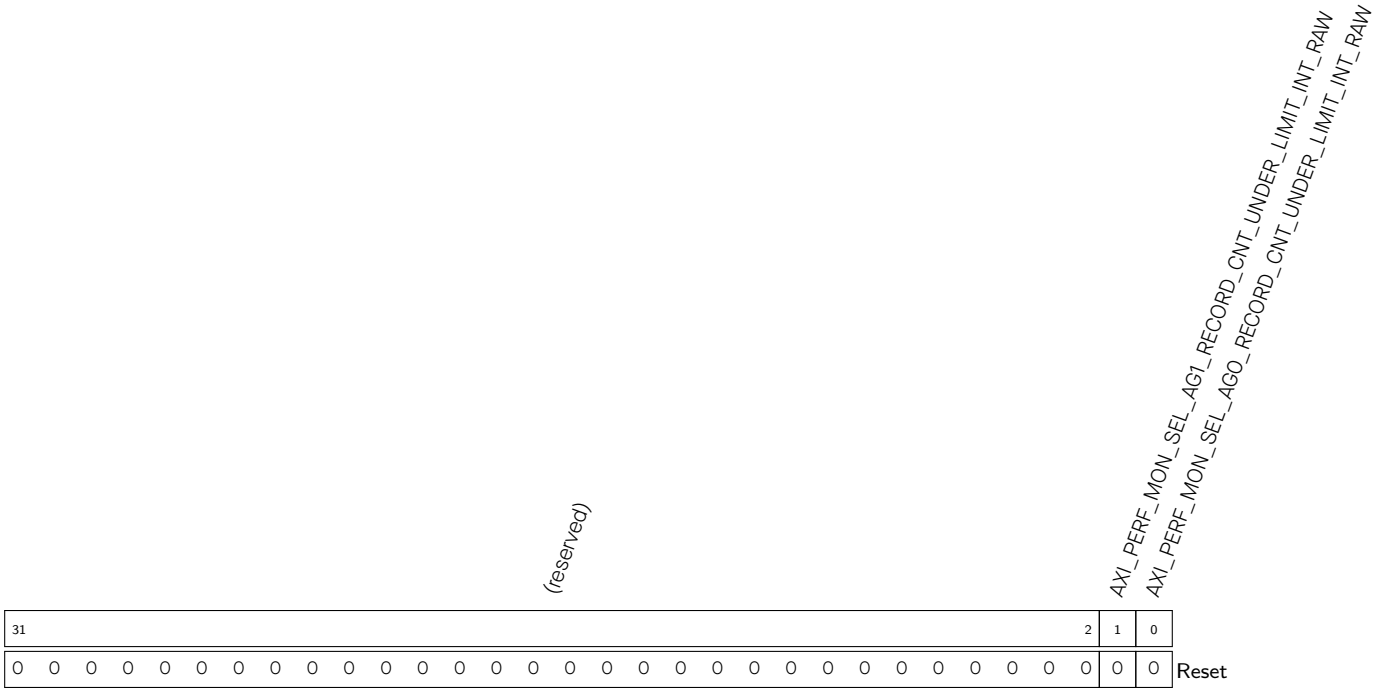
Register 23.28. AXI_PERF_MON_SEL_AG*n*_RD_BANDW_CNT_VALID_STROBE_NUM_REG (*n*: 0-1) (0x01B0+0x4n*)**



AXI_PERF_MON_SEL_AG*n*_RD_BANDW_TIME_CNT_VALID_STROBE_NUM (*n*: 0-1) Represents the scaling factor for the time counter in read bandwidth tests. (R/W)

AXI_PERF_MON_SEL_AG*n*_RD_BANDW_DATA_CNT_VALID_STROBE_NUM (*n*: 0-1) Represents the scaling factor for the data counter in read bandwidth tests. (R/W)

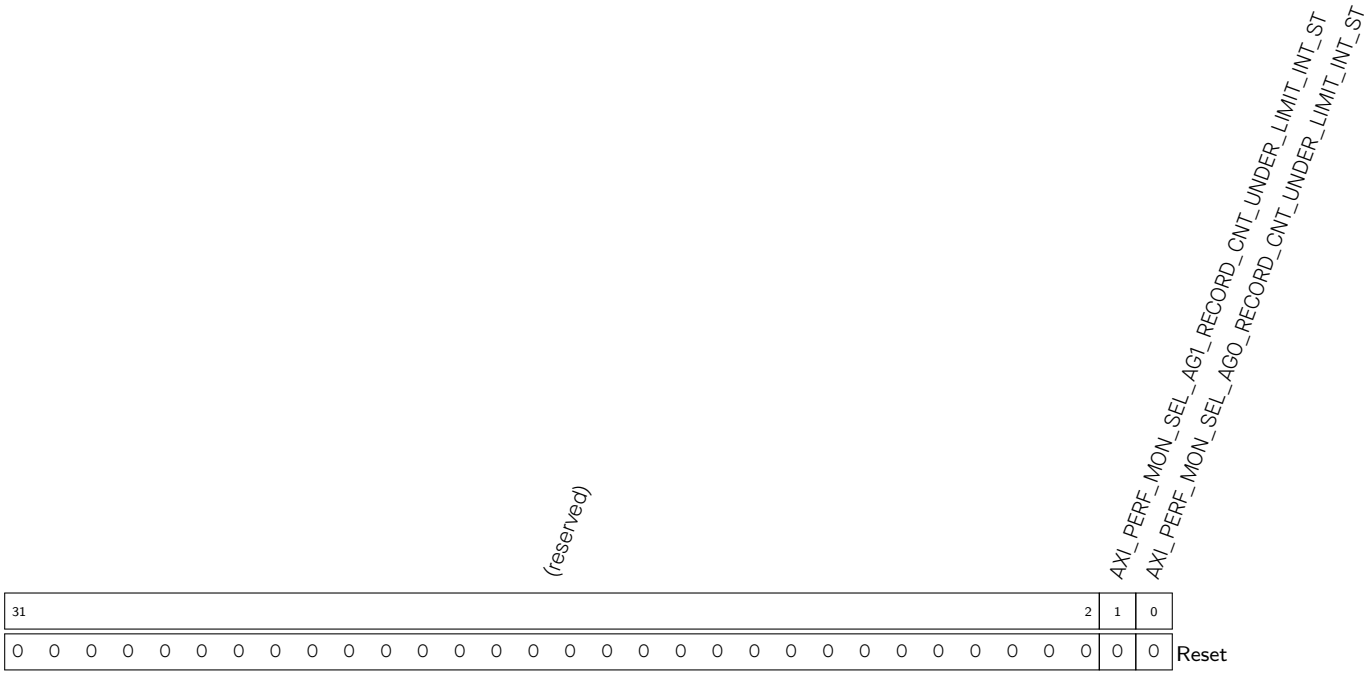
Register 23.29. AXI_PERF_MON_SEL_AG_INT_RAW_REG (0x01C0)



AXI_PERF_MON_SEL_AG0_RECORD_CNT_UNDER_LIMIT_INT_RAW The raw interrupt status of AXI_PERF_MON_SEL_AG0_RECORD_CNT_UNDER_LIMIT_INT. (R/WTC/SS)

AXI_PERF_MON_SEL_AG1_RECORD_CNT_UNDER_LIMIT_INT_RAW The raw interrupt status of AXI_PERF_MON_SEL_AG1_RECORD_CNT_UNDER_LIMIT_INT. (R/WTC/SS)

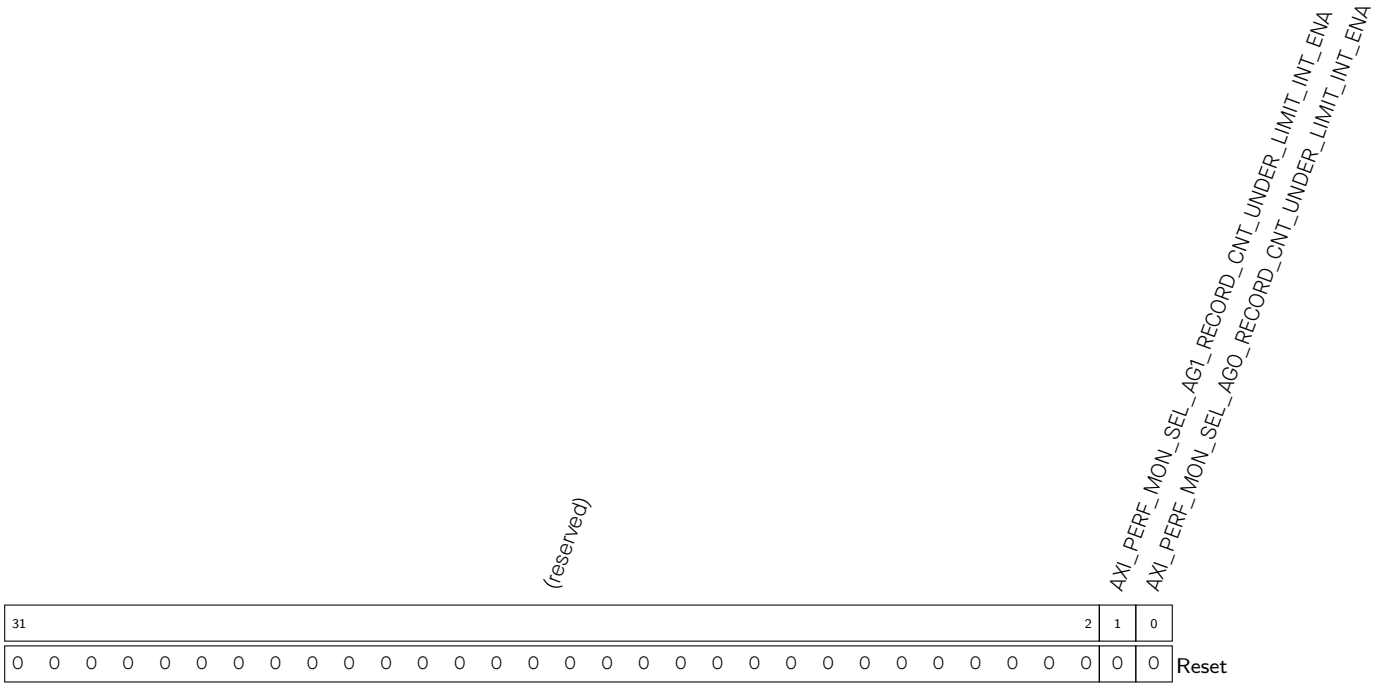
Register 23.30. AXI_PERF_MON_SEL_AG_INT_ST_REG (0x01C4)



AXI_PERF_MON_SEL_AG0_RECORD_CNT_UNDER_LIMIT_INT_ST The masked interrupt status of AXI_PERF_MON_SEL_AG0_RECORD_CNT_UNDER_LIMIT_INT. (RO)

AXI_PERF_MON_SEL_AG1_RECORD_CNT_UNDER_LIMIT_INT_ST The masked interrupt status of AXI_PERF_MON_SEL_AG1_RECORD_CNT_UNDER_LIMIT_INT. (RO)

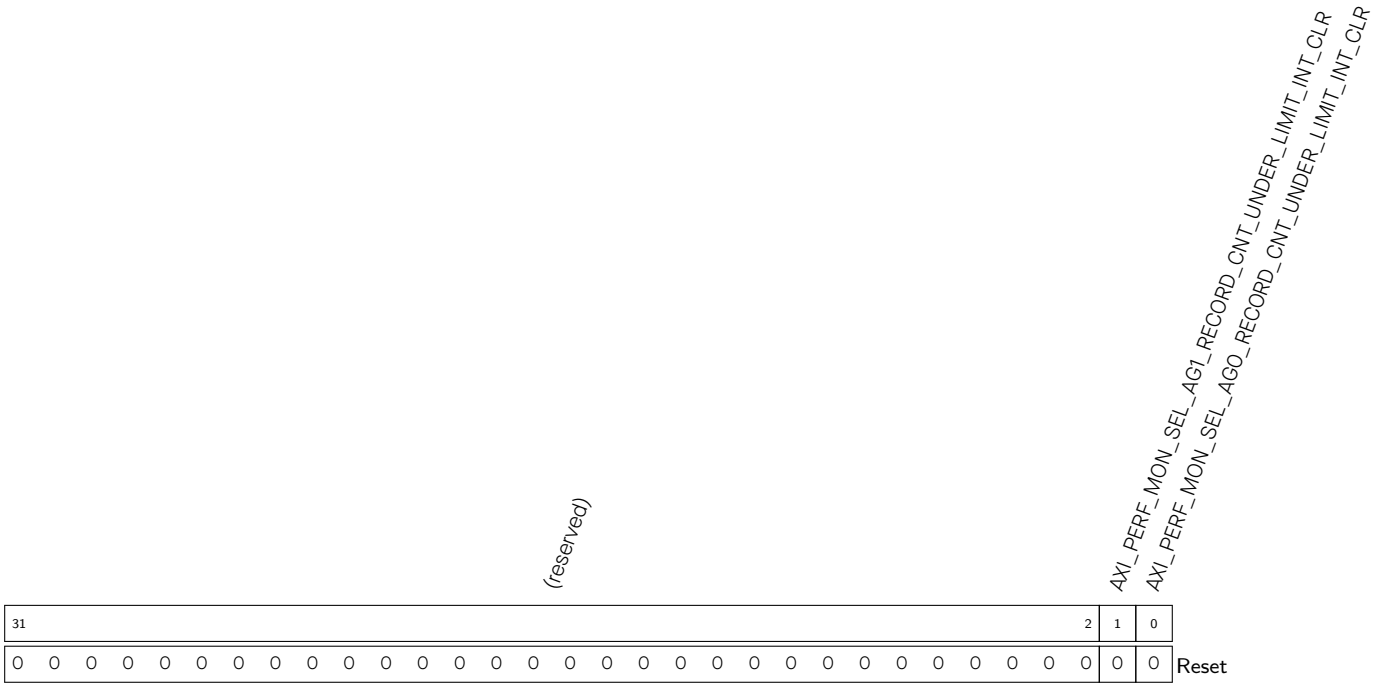
Register 23.31. AXI_PERF_MON_SEL_AG_INT_ENA_REG (0x01C8)



AXI_PERF_MON_SEL_AG0_RECORD_CNT_UNDER_LIMIT_INT_ENA Write 1 to enable AXI_PERF_MON_SEL_AG0_RECORD_CNT_UNDER_LIMIT_INT. (R/W)

AXI_PERF_MON_SEL_AG1_RECORD_CNT_UNDER_LIMIT_INT_ENA Write 1 to enable AXI_PERF_MON_SEL_AG1_RECORD_CNT_UNDER_LIMIT_INT. (R/W)

Register 23.32. AXI_PERF_MON_SEL_AG_INT_CLR_REG (0x01CC)



AXI_PERF_MON_SEL_AG0_RECORD_CNT_UNDER_LIMIT_INT_CLR Write 1 to clear
AXI_PERF_MON_SEL_AG0_RECORD_CNT_UNDER_LIMIT_INT. (WT)

AXI_PERF_MON_SEL_AG1_RECORD_CNT_UNDER_LIMIT_INT_CLR Write 1 to clear
AXI_PERF_MON_SEL_AG1_RECORD_CNT_UNDER_LIMIT_INT. (WT)

Chapter 24

LP Mailbox

24.1 Overview

ESP32-P4 integrates an LP Mailbox module which provides an efficient inter-core communication mechanism between the LP CPU and HP CPU0/1. The LP Mailbox module comprises of sixteen 32-bit message registers that the LP CPU and HP CPU0/1 can use to store and exchange message. Inter-core communication between LP CPU and HP CPU0/1 is achieved through an interrupt mechanism implemented within the LP Mailbox module.

24.2 Features

ESP32-P4 LP Mailbox has the following features:

- Sixteen 32-bit message registers for inter-core communication
- LP CPU external interrupt signal: MB_LP_INTR
- HP CPU*x* (*x*: 0 ~ 1) external interrupt signal: MB_HP_INTR

24.3 Functional Description

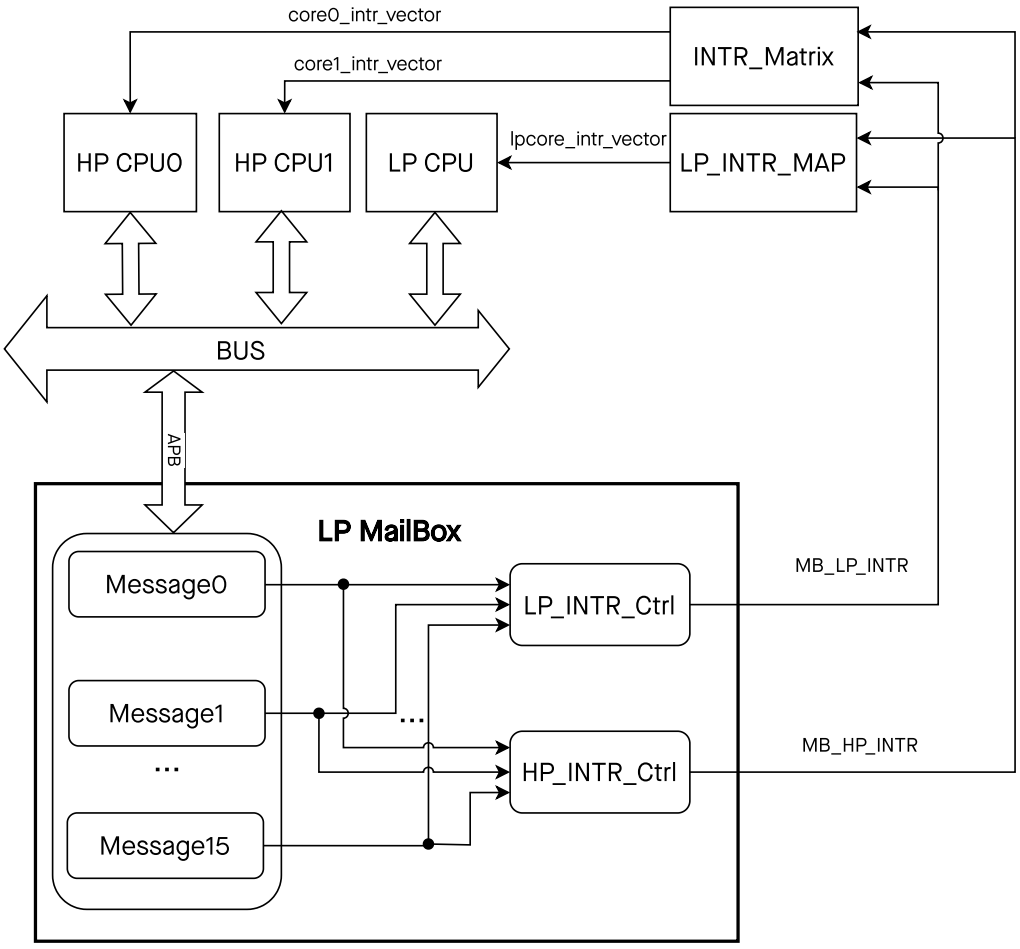


Figure 24.3-1. LP Mailbox Block Diagram

24.3.1 Message Registers

The LP Mailbox provides sixteen 32-bit message registers in total, denoted as MB_MESSAGE_0_REG ~ MB_MESSAGE_15_REG, for caching inter-core communication messages. Both LP CPU and HP CPU0/1 can read and write these message registers to pass or retrieve message.

Read and write message registers is done through the APB bus, using APB_CLK, which is the only one clock in LP Mailbox. For more information about the clock, please refer to Chapter 11 *Reset and Clock*.

24.3.2 Interrupts

ESP32-P4 LP Mailbox can generate the following external interrupt signals:

- MB_LP_INTR
- MB_HP_INTR

MB_LP_INTR is directly mapped to LP CPU (see *Low-Power CPU*), and MB_HP_INTR can be mapped to HP CPU0/1 through the Interrupt Matrix. These interrupt signals are generated by internal interrupt sources of the

LP Mailbox module. Table 24.3-1 lists the internal interrupt sources, interrupt trigger conditions, and generated interrupt signals of the LP Mailbox.

Table 24.3-1. LP Mailbox's Internal Interrupt Sources

Internal Interrupt Source	Trigger Condition	Interrupt Signal
MB_HP_*m*_INT	Write to MB_MESSAGE_*m*_REG	MB_HP_INTR
MB_LP_*n*_INT	Write to MB_MESSAGE_*n*_REG	MB_LP_INTR

Note:

- *m* and *n* can be 0 ~ 15.
- MB_HP_INTR can also serve as an external interrupt source for LP CPU and be mapped to LP CPU by LP_INTR_MAP as illustrated in Figure 24.3-1. For more information about LP_INTR_MAP, please refer to Chapter 5 *Low-Power CPU*.
- MB_LP_INTR can also serve as an external interrupt source for HP CPU*x* and can be mapped to HP CPU*x* through the Interrupt Matrix.
- When the LP Mailbox is required to assign different interrupt sources for HP CPU0 and HP CPU1, it can map MB_HP_INTR and MB_LP_INTR independently to HP CPU0 and HP CPU1 using the Interrupt Matrix.

24.3.3 Inter-Core Communication

When communication is required between LP CPU and HP CPU*x*, subset of or all message registers in the LP Mailbox can be utilized to share message as needed. The synchronization of message can be achieved through the corresponding interrupts. During communication, software must ensure that only the sender is allowed to write to the message registers to prevent data corruption.

- Example of initiating communication from HP CPU0 to LP CPU using MB_MESSAGE_*n*_REG:
  - Set MB_LP_*n*_INT_ENA to enable the MB_LP_*n*_INT interrupt.
  - HP CPU0 writes to MB_MASSEGE_*n*_REG, caching shared message.
  - LP CPU checks MB_LP_INT_ST_REG interrupt status register, finding the MB_LP_*n*_ST status bit set.
  - LP CPU reads MB_MASSEGE_*n*_REG to retrieve shared message.
- Example of initiating communication from LP CPU to HP CPU0:
  - Set MB_HP_*m*_INT_ENA to enable the MB_HP_*m*_INT interrupt.
  - Configure the Interrupt Matrix to map MB_HP_INTR to HP CPU0. See Chapter 13 *Interrupt Matrix*.
  - LP CPU writes to MB_MASSEGE_*m*_REG, caching shared messages.
  - HP CPU0 checks the MB_HP_INT_ST_REG interrupt status register, finding the MB_HP_*m*_ST status bit set.
  - HP CPU0 reads MB_MASSEGE_*m*_REG to retrieve shared messages.

24.4 Register Summary

The addresses in this section are relative to LP Mailbox base address provided in Table 9.3-2 in Chapter 9 *System and Memory*.

The abbreviations given in Column **Access** are explained in Section *Access Types for Registers*.

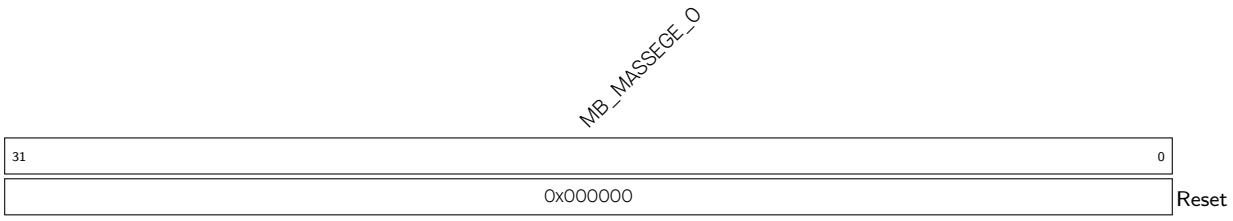
Name	Description	Address	Access
Configuration Registers			
MB_MASSEGE_0_REG	Message register 0	0x0000	R/W
MB_MASSEGE_1_REG	Message register 1	0x0004	R/W
MB_MASSEGE_2_REG	Message register 2	0x0008	R/W
MB_MASSEGE_3_REG	Message register 3	0x000C	R/W
MB_MASSEGE_4_REG	Message register 4	0x0010	R/W
MB_MASSEGE_5_REG	Message register 5	0x0014	R/W
MB_MASSEGE_6_REG	Message register 6	0x0018	R/W
MB_MASSEGE_7_REG	Message register 7	0x001C	R/W
MB_MASSEGE_8_REG	Message register 8	0x0020	R/W
MB_MASSEGE_9_REG	Message register 9	0x0024	R/W
MB_MASSEGE_10_REG	Message register 10	0x0028	R/W
MB_MASSEGE_11_REG	Message register 11	0x002C	R/W
MB_MASSEGE_12_REG	Message register 12	0x0030	R/W
MB_MASSEGE_13_REG	Message register 13	0x0034	R/W
MB_MASSEGE_14_REG	Message register 14	0x0038	R/W
MB_MASSEGE_15_REG	Message register 15	0x003C	R/W
MB_REG_CLK_EN_REG	Clock gate register	0x0060	R/W
Interrupt Registers			
MB_LP_INT_RAW_REG	MB_LP_INTR raw bits	0x0040	RO
MB_LP_INT_ST_REG	MB_LP_INTR status bits	0x0044	RO
MB_LP_INT_ENA_REG	MB_LP_INTR enable bits	0x0048	R/W
MB_LP_INT_CLR_REG	MB_LP_INTR clear bits	0x004C	WO
MB_HP_INT_RAW_REG	MB_HP_INTR raw bits	0x0050	RO
MB_HP_INT_ST_REG	MB_HP_INTR status bits	0x0054	RO
MB_HP_INT_ENA_REG	MB_HP_INTR enable bits	0x0058	R/W
MB_HP_INT_CLR_REG	MB_HP_INTR clear bits	0x005C	WO

24.5 Registers

The addresses in this section are relative to LP Mailbox base address provided in Table 9.3-2 in Chapter 9 *System and Memory*.

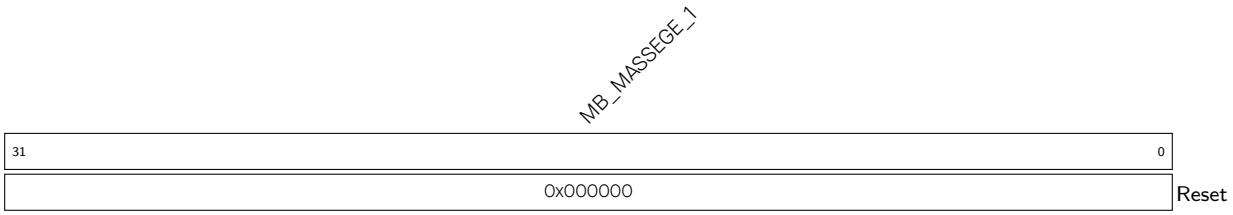
For how to program reserved fields, please refer to Section *Programming Reserved Register Field*.

Register 24.1. MB_MASSEGE_0_REG (0x0000)



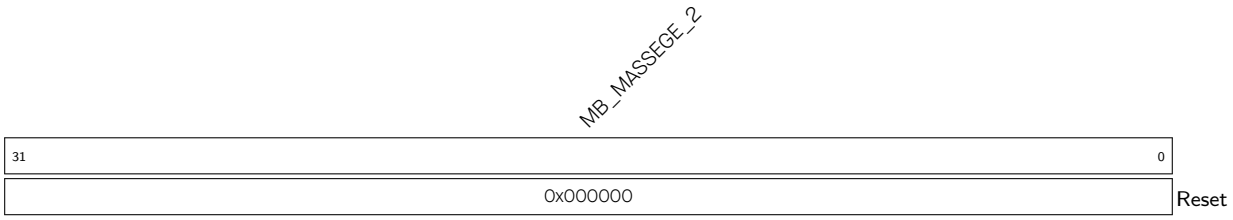
MB_MASSEGE_0 Message register 0. (R/W)

Register 24.2. MB_MASSEGE_1_REG (0x0004)



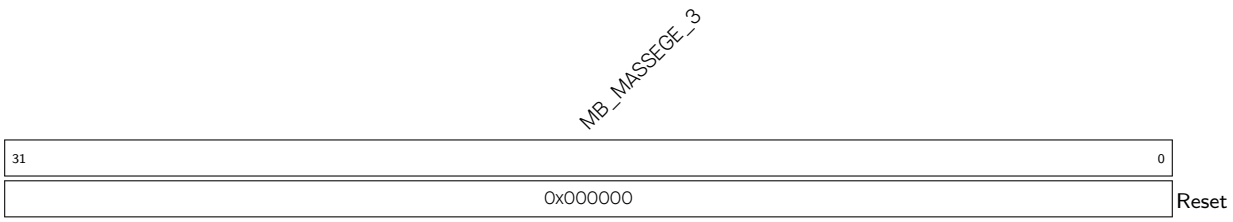
MB_MASSEGE_1 Message register 1. (R/W)

Register 24.3. MB_MASSEGE_2_REG (0x0008)



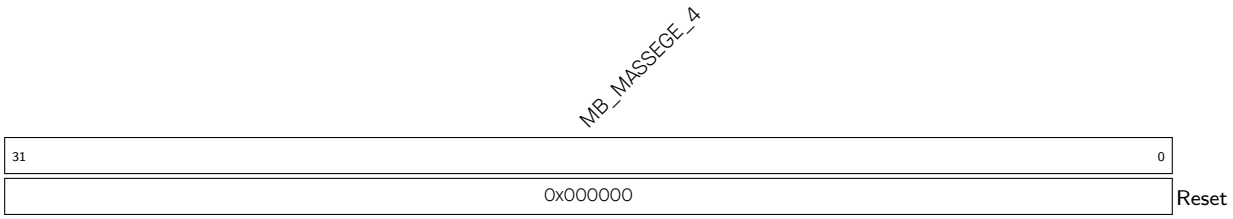
MB_MASSEGE_2 Message register 2. (R/W)

Register 24.4. MB_MASSEGE_3_REG (0x000C)



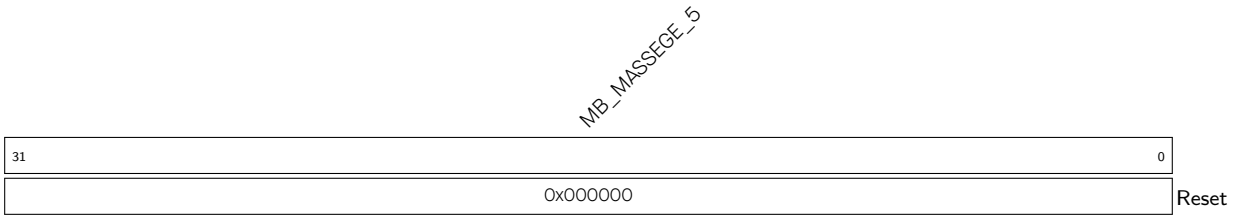
MB_MASSEGE_3 Message register 3. (R/W)

Register 24.5. MB_MASSEGE_4_REG (0x0010)



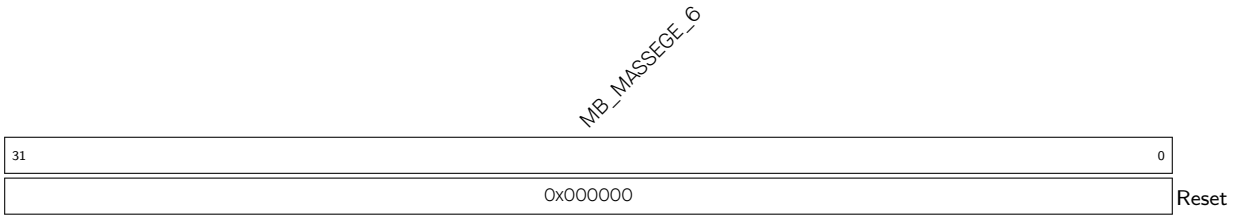
MB_MASSEGE_4 Message register 4. (R/W)

Register 24.6. MB_MASSEGE_5_REG (0x0014)



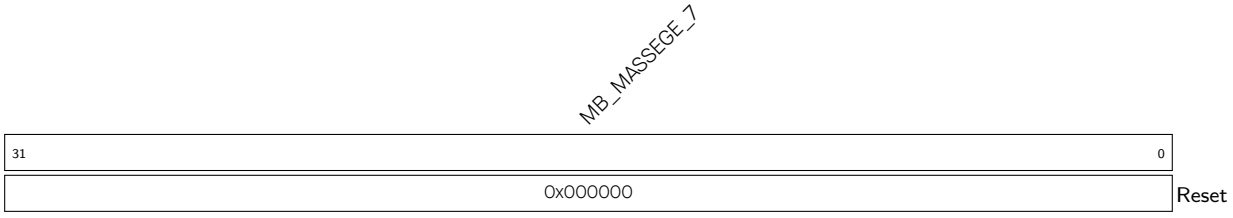
MB_MASSEGE_5 Message register 5. (R/W)

Register 24.7. MB_MASSEGE_6_REG (0x0018)



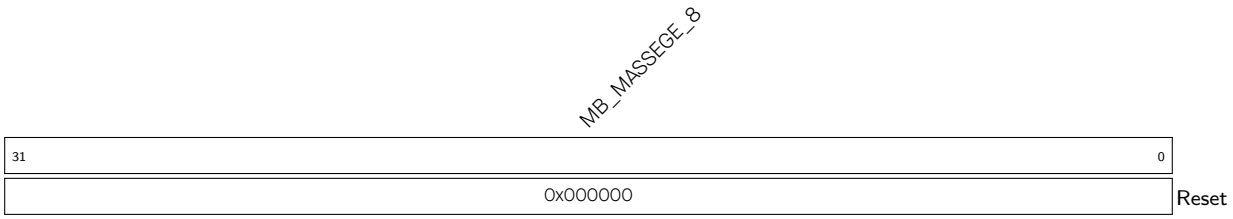
MB_MASSEGE_6 Message register 6. (R/W)

Register 24.8. MB_MASSEGE_7_REG (0x001C)



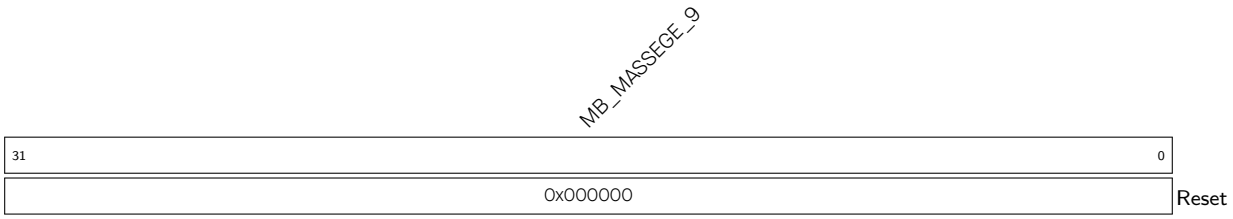
MB_MASSEGE_7 Message register 7. (R/W)

Register 24.9. MB_MASSEGE_8_REG (0x0020)



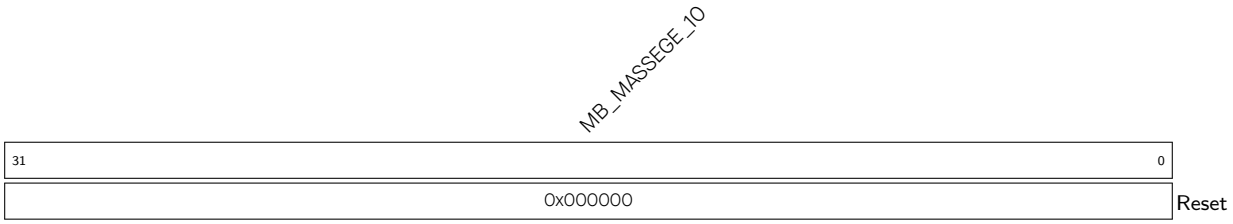
MB_MASSEGE_8 Message register 8. (R/W)

Register 24.10. MB_MASSEGE_9_REG (0x0024)



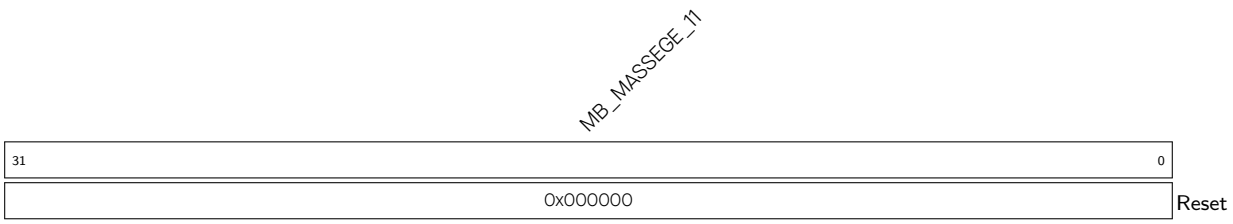
MB_MASSEGE_9 Message register 9. (R/W)

Register 24.11. MB_MASSEGE_10_REG (0x0028)



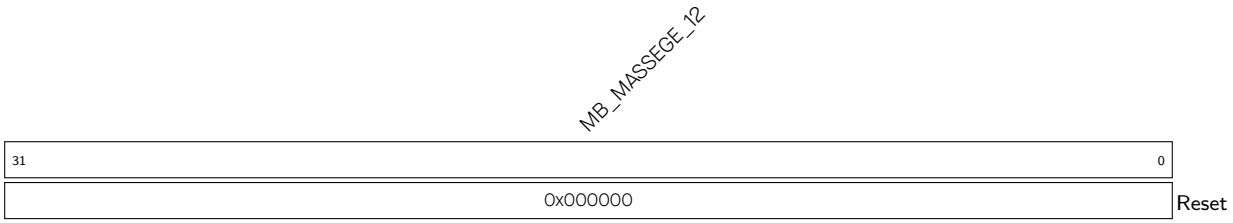
MB_MASSEGE_10 Message register 10. (R/W)

Register 24.12. MB_MASSEGE_11_REG (0x002C)



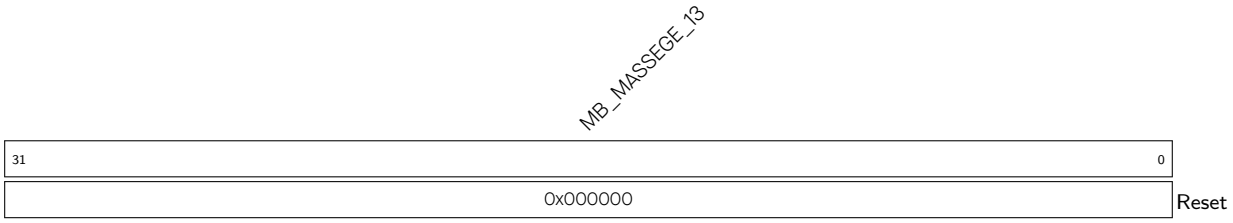
MB_MASSEGE_11 Message register 11. (R/W)

Register 24.13. MB_MASSEGE_12_REG (0x0030)



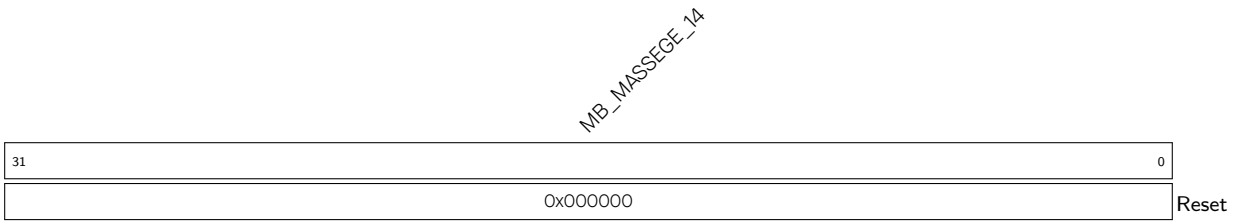
MB_MASSEGE_12 Message register 12. (R/W)

Register 24.14. MB_MASSEGE_13_REG (0x0034)



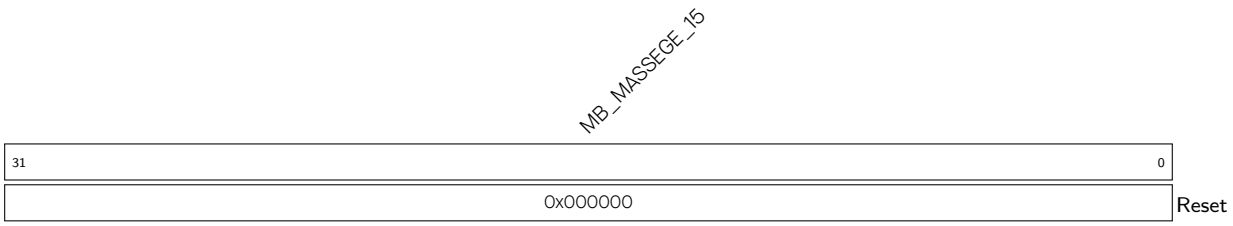
MB_MASSEGE_13 Message register 13. (R/W)

Register 24.15. MB_MASSEGE_14_REG (0x0038)



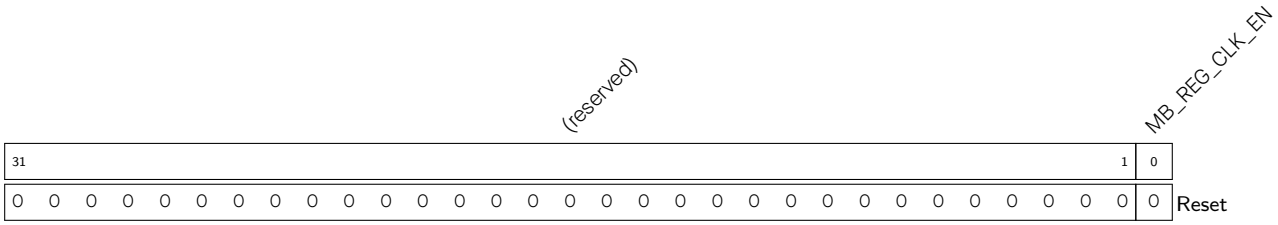
MB_MASSEGE_14 Message register 14. (R/W)

Register 24.16. MB_MASSEGE_15_REG (0x003C)



MB_MASSEGE_15 Message register 15. (R/W)

Register 24.17. MB_REG_CLK_EN_REG (0x0060)



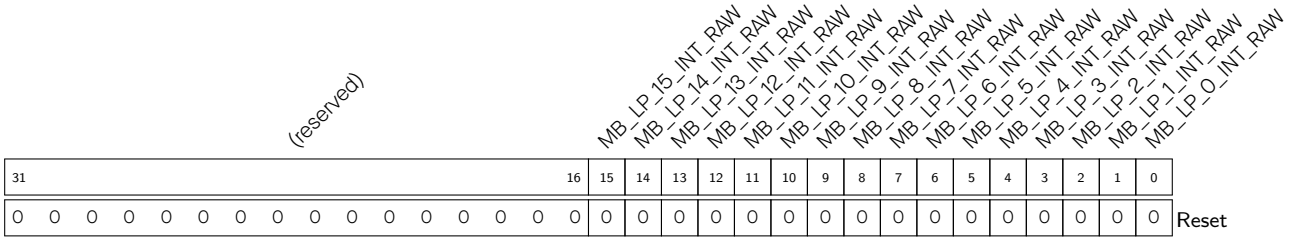
MB_REG_CLK_EN Configures register clock gating.

1: Register clock is always enabled for read and write operations.

0: Only enable clock for register read or write operations.

(R/W)

Register 24.18. MB_LP_INT_RAW_REG (0x0040)



MB_LP_0_INT_RAW The raw interrupt bit of the MB_LP_0_INT interrupt. (RO)

MB_LP_1_INT_RAW The raw interrupt bit of the MB_LP_1_INT interrupt. (RO)

MB_LP_2_INT_RAW The raw interrupt bit of the MB_LP_2_INT interrupt. (RO)

MB_LP_3_INT_RAW The raw interrupt bit of the MB_LP_3_INT interrupt. (RO)

MB_LP_4_INT_RAW The raw interrupt bit of the MB_LP_4_INT interrupt. (RO)

MB_LP_5_INT_RAW The raw interrupt bit of the MB_LP_5_INT interrupt. (RO)

MB_LP_6_INT_RAW The raw interrupt bit of the MB_LP_6_INT interrupt. (RO)

MB_LP_7_INT_RAW The raw interrupt bit of the MB_LP_7_INT interrupt. (RO)

MB_LP_8_INT_RAW The raw interrupt bit of the MB_LP_8_INT interrupt. (RO)

MB_LP_9_INT_RAW The raw interrupt bit of the MB_LP_9_INT interrupt. (RO)

MB_LP_10_INT_RAW The raw interrupt bit of the MB_LP_10_INT interrupt. (RO)

MB_LP_11_INT_RAW The raw interrupt bit of the MB_LP_11_INT interrupt. (RO)

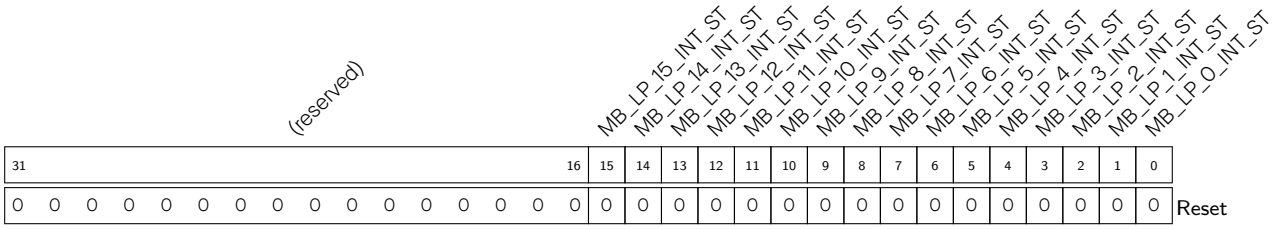
MB_LP_12_INT_RAW The raw interrupt bit of the MB_LP_12_INT interrupt. (RO)

MB_LP_13_INT_RAW The raw interrupt bit of the MB_LP_13_INT interrupt. (RO)

MB_LP_14_INT_RAW The raw interrupt bit of the MB_LP_14_INT interrupt. (RO)

MB_LP_15_INT_RAW The raw interrupt bit of the MB_LP_15_INT interrupt. (RO)

Register 24.19. MB_LP_INT_ST_REG (0x0044)



MB_LP_0_INT_ST The masked interrupt status bit of the MB_LP_0_INT interrupt. (RO)

MB_LP_1_INT_ST The masked interrupt status bit of the MB_LP_1_INT interrupt. (RO)

MB_LP_2_INT_ST The masked interrupt status bit of the MB_LP_2_INT interrupt. (RO)

MB_LP_3_INT_ST The masked interrupt status bit of the MB_LP_3_INT interrupt. (RO)

MB_LP_4_INT_ST The masked interrupt status bit of the MB_LP_4_INT interrupt. (RO)

MB_LP_5_INT_ST The masked interrupt status bit of the MB_LP_5_INT interrupt. (RO)

MB_LP_6_INT_ST The masked interrupt status bit of the MB_LP_6_INT interrupt. (RO)

MB_LP_7_INT_ST The masked interrupt status bit of the MB_LP_7_INT interrupt. (RO)

MB_LP_8_INT_ST The masked interrupt status bit of the MB_LP_8_INT interrupt. (RO)

MB_LP_9_INT_ST The masked interrupt status bit of the MB_LP_9_INT interrupt. (RO)

MB_LP_10_INT_ST The masked interrupt status bit of the MB_LP_10_INT interrupt. (RO)

MB_LP_11_INT_ST The masked interrupt status bit of the MB_LP_11_INT interrupt. (RO)

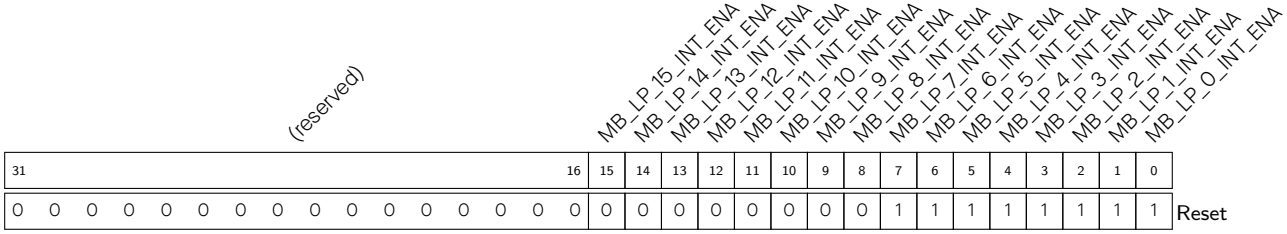
MB_LP_12_INT_ST The masked interrupt status bit of the MB_LP_12_INT interrupt. (RO)

MB_LP_13_INT_ST The masked interrupt status bit of the MB_LP_13_INT interrupt. (RO)

MB_LP_14_INT_ST The masked interrupt status bit of the MB_LP_14_INT interrupt. (RO)

MB_LP_15_INT_ST The masked interrupt status bit of the MB_LP_15_INT interrupt. (RO)

Register 24.20. MB_LP_INT_ENA_REG (0x0048)



MB_LP_0_INT_ENA Write 1 to enable the MB_LP_0_INT interrupt. (R/W)

MB_LP_1_INT_ENA Write 1 to enable the MB_LP_1_INT interrupt. (R/W)

MB_LP_2_INT_ENA Write 1 to enable the MB_LP_2_INT interrupt. (R/W)

MB_LP_3_INT_ENA Write 1 to enable the MB_LP_3_INT interrupt. (R/W)

MB_LP_4_INT_ENA Write 1 to enable the MB_LP_4_INT interrupt. (R/W)

MB_LP_5_INT_ENA Write 1 to enable the MB_LP_5_INT interrupt. (R/W)

MB_LP_6_INT_ENA Write 1 to enable the MB_LP_6_INT interrupt. (R/W)

MB_LP_7_INT_ENA Write 1 to enable the MB_LP_7_INT interrupt. (R/W)

MB_LP_8_INT_ENA Write 1 to enable the MB_LP_8_INT interrupt. (R/W)

MB_LP_9_INT_ENA Write 1 to enable the MB_LP_9_INT interrupt. (R/W)

MB_LP_10_INT_ENA Write 1 to enable the MB_LP_10_INT interrupt. (R/W)

MB_LP_11_INT_ENA Write 1 to enable the MB_LP_11_INT interrupt. (R/W)

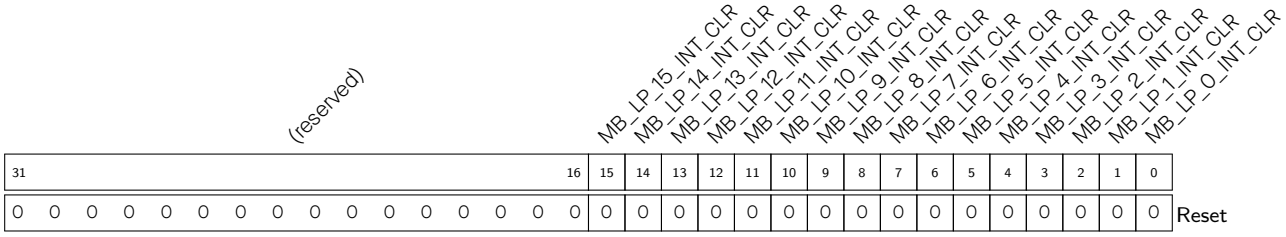
MB_LP_12_INT_ENA Write 1 to enable the MB_LP_12_INT interrupt. (R/W)

MB_LP_13_INT_ENA Write 1 to enable the MB_LP_13_INT interrupt. (R/W)

MB_LP_14_INT_ENA Write 1 to enable the MB_LP_14_INT interrupt. (R/W)

MB_LP_15_INT_ENA Write 1 to enable the MB_LP_15_INT interrupt. (R/W)

Register 24.21. MB_LP_INT_CLR_REG (0x004C)



MB_LP_0_INT_CLR Write 1 to clear the MB_LP_0_INT interrupt. (WO)

MB_LP_1_INT_CLR Write 1 to clear the MB_LP_1_INT interrupt. (WO)

MB_LP_2_INT_CLR Write 1 to clear the MB_LP_2_INT interrupt. (WO)

MB_LP_3_INT_CLR Write 1 to clear the MB_LP_3_INT interrupt. (WO)

MB_LP_4_INT_CLR Write 1 to clear the MB_LP_4_INT interrupt. (WO)

MB_LP_5_INT_CLR Write 1 to clear the MB_LP_5_INT interrupt. (WO)

MB_LP_6_INT_CLR Write 1 to clear the MB_LP_6_INT interrupt. (WO)

MB_LP_7_INT_CLR Write 1 to clear the MB_LP_7_INT interrupt. (WO)

MB_LP_8_INT_CLR Write 1 to clear the MB_LP_8_INT interrupt. (WO)

MB_LP_9_INT_CLR Write 1 to clear the MB_LP_9_INT interrupt. (WO)

MB_LP_10_INT_CLR Write 1 to clear the MB_LP_10_INT interrupt. (WO)

MB_LP_11_INT_CLR Write 1 to clear the MB_LP_11_INT interrupt. (WO)

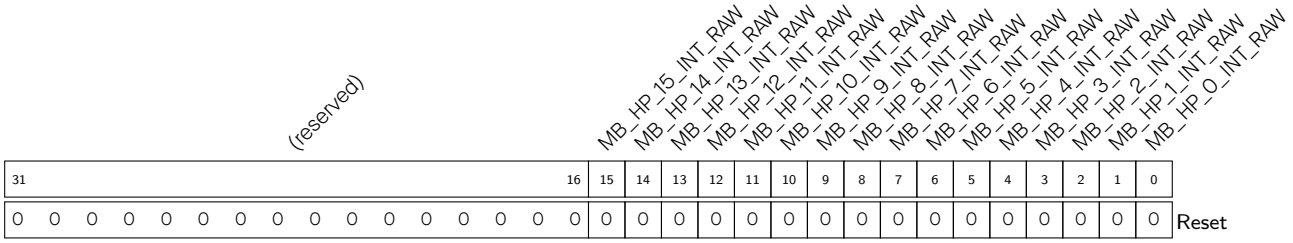
MB_LP_12_INT_CLR Write 1 to clear the MB_LP_12_INT interrupt. (WO)

MB_LP_13_INT_CLR Write 1 to clear the MB_LP_13_INT interrupt. (WO)

MB_LP_14_INT_CLR Write 1 to clear the MB_LP_14_INT interrupt. (WO)

MB_LP_15_INT_CLR Write 1 to clear the MB_LP_15_INT interrupt. (WO)

Register 24.22. MB_HP_INT_RAW_REG (0x0050)



MB_HP_0_INT_RAW The raw interrupt bit of the MB_HP_0_INT interrupt. (RO)

MB_HP_1_INT_RAW The raw interrupt bit of the MB_HP_1_INT interrupt. (RO)

MB_HP_2_INT_RAW The raw interrupt bit of the MB_HP_2_INT interrupt. (RO)

MB_HP_3_INT_RAW The raw interrupt bit of the MB_HP_3_INT interrupt. (RO)

MB_HP_4_INT_RAW The raw interrupt bit of the MB_HP_4_INT interrupt. (RO)

MB_HP_5_INT_RAW The raw interrupt bit of the MB_HP_5_INT interrupt. (RO)

MB_HP_6_INT_RAW The raw interrupt bit of the MB_HP_6_INT interrupt. (RO)

MB_HP_7_INT_RAW The raw interrupt bit of the MB_HP_7_INT interrupt. (RO)

MB_HP_8_INT_RAW The raw interrupt bit of the MB_HP_8_INT interrupt. (RO)

MB_HP_9_INT_RAW The raw interrupt bit of the MB_HP_9_INT interrupt. (RO)

MB_HP_10_INT_RAW The raw interrupt bit of the MB_HP_10_INT interrupt. (RO)

MB_HP_11_INT_RAW The raw interrupt bit of the MB_HP_11_INT interrupt. (RO)

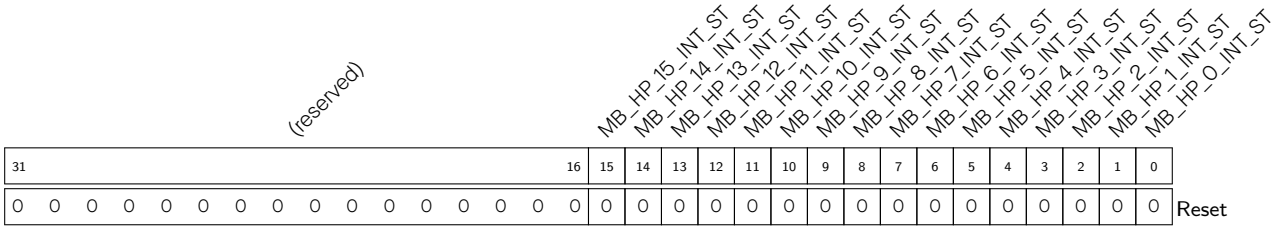
MB_HP_12_INT_RAW The raw interrupt bit of the MB_HP_12_INT interrupt. (RO)

MB_HP_13_INT_RAW The raw interrupt bit of the MB_HP_13_INT interrupt. (RO)

MB_HP_14_INT_RAW The raw interrupt bit of the MB_HP_14_INT interrupt. (RO)

MB_HP_15_INT_RAW The raw interrupt bit of the MB_HP_15_INT interrupt. (RO)

Register 24.23. MB_HP_INT_ST_REG (0x0054)



MB_HP_0_INT_ST The masked interrupt status bit of the MB_HP_0_INT interrupt. (RO)

MB_HP_1_INT_ST The masked interrupt status bit of the MB_HP_1_INT interrupt. (RO)

MB_HP_2_INT_ST The masked interrupt status bit of the MB_HP_2_INT interrupt. (RO)

MB_HP_3_INT_ST The masked interrupt status bit of the MB_HP_3_INT interrupt. (RO)

MB_HP_4_INT_ST The masked interrupt status bit of the MB_HP_4_INT interrupt. (RO)

MB_HP_5_INT_ST The masked interrupt status bit of the MB_HP_5_INT interrupt. (RO)

MB_HP_6_INT_ST The masked interrupt status bit of the MB_HP_6_INT interrupt. (RO)

MB_HP_7_INT_ST The masked interrupt status bit of the MB_HP_7_INT interrupt. (RO)

MB_HP_8_INT_ST The masked interrupt status bit of the MB_HP_8_INT interrupt. (RO)

MB_HP_9_INT_ST The masked interrupt status bit of the MB_HP_9_INT interrupt. (RO)

MB_HP_10_INT_ST The masked interrupt status bit of the MB_HP_10_INT interrupt. (RO)

MB_HP_11_INT_ST The masked interrupt status bit of the MB_HP_11_INT interrupt. (RO)

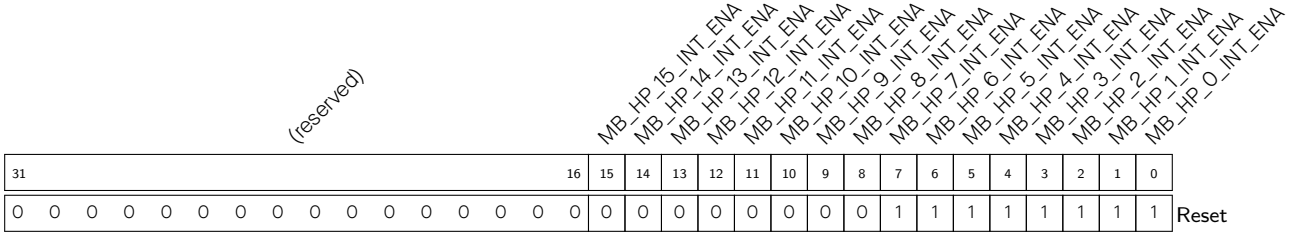
MB_HP_12_INT_ST The masked interrupt status bit of the MB_HP_12_INT interrupt. (RO)

MB_HP_13_INT_ST The masked interrupt status bit of the MB_HP_13_INT interrupt. (RO)

MB_HP_14_INT_ST The masked interrupt status bit of the MB_HP_14_INT interrupt. (RO)

MB_HP_15_INT_ST The masked interrupt status bit of the MB_HP_15_INT interrupt. (RO)

Register 24.24. MB_HP_INT_ENA_REG (0x0058)



MB_HP_0_INT_ENA Write 1 to enable the MB_HP_0_INT interrupt. (R/W)

MB_HP_1_INT_ENA Write 1 to enable the MB_HP_1_INT interrupt. (R/W)

MB_HP_2_INT_ENA Write 1 to enable the MB_HP_2_INT interrupt. (R/W)

MB_HP_3_INT_ENA Write 1 to enable the MB_HP_3_INT interrupt. (R/W)

MB_HP_4_INT_ENA Write 1 to enable the MB_HP_4_INT interrupt. (R/W)

MB_HP_5_INT_ENA Write 1 to enable the MB_HP_5_INT interrupt. (R/W)

MB_HP_6_INT_ENA Write 1 to enable the MB_HP_6_INT interrupt. (R/W)

MB_HP_7_INT_ENA Write 1 to enable the MB_HP_7_INT interrupt. (R/W)

MB_HP_8_INT_ENA Write 1 to enable the MB_HP_8_INT interrupt. (R/W)

MB_HP_9_INT_ENA Write 1 to enable the MB_HP_9_INT interrupt. (R/W)

MB_HP_10_INT_ENA Write 1 to enable the MB_HP_10_INT interrupt. (R/W)

MB_HP_11_INT_ENA Write 1 to enable the MB_HP_11_INT interrupt. (R/W)

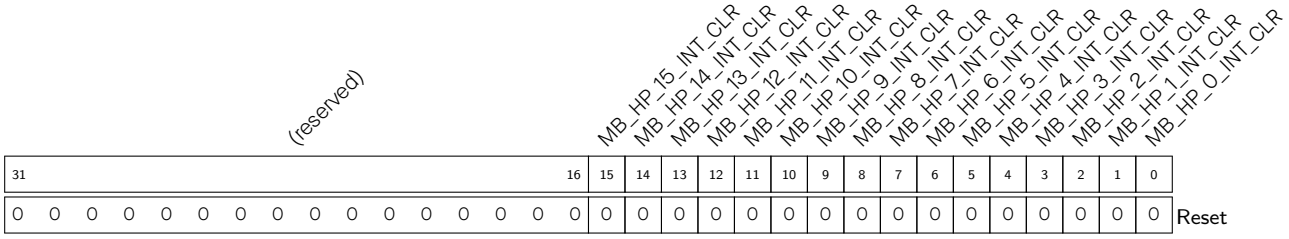
MB_HP_12_INT_ENA Write 1 to enable the MB_HP_12_INT interrupt. (R/W)

MB_HP_13_INT_ENA Write 1 to enable the MB_HP_13_INT interrupt. (R/W)

MB_HP_14_INT_ENA Write 1 to enable the MB_HP_14_INT interrupt. (R/W)

MB_HP_15_INT_ENA Write 1 to enable the MB_HP_15_INT interrupt. (R/W)

Register 24.25. MB_HP_INT_CLR_REG (0x005C)



MB_HP_0_INT_CLR Write 1 to clear the MB_HP_0_INT interrupr. (WO)

MB_HP_1_INT_CLR Write 1 to clear the MB_HP_1_INT interrupr. (WO)

MB_HP_2_INT_CLR Write 1 to clear the MB_HP_2_INT interrupr. (WO)

MB_HP_3_INT_CLR Write 1 to clear the MB_HP_3_INT interrupr. (WO)

MB_HP_4_INT_CLR Write 1 to clear the MB_HP_4_INT interrupr. (WO)

MB_HP_5_INT_CLR Write 1 to clear the MB_HP_5_INT interrupr. (WO)

MB_HP_6_INT_CLR Write 1 to clear the MB_HP_6_INT interrupr. (WO)

MB_HP_7_INT_CLR Write 1 to clear the MB_HP_7_INT interrupr. (WO)

MB_HP_8_INT_CLR Write 1 to clear the MB_HP_8_INT interrupr. (WO)

MB_HP_9_INT_CLR Write 1 to clear the MB_HP_9_INT interrupr. (WO)

MB_HP_10_INT_CLR Write 1 to clear the MB_HP_10_INT interrupr. (WO)

MB_HP_11_INT_CLR Write 1 to clear the MB_HP_11_INT interrupr. (WO)

MB_HP_12_INT_CLR Write 1 to clear the MB_HP_12_INT interrupr. (WO)

MB_HP_13_INT_CLR Write 1 to clear the MB_HP_13_INT interrupr. (WO)

MB_HP_14_INT_CLR Write 1 to clear the MB_HP_14_INT interrupr. (WO)

MB_HP_15_INT_CLR Write 1 to clear the MB_HP_15_INT interrupr. (WO)

Chapter 25

Brown-out Detector

25.1 Introduction

The brown-out detector of ESP32-P4 monitors the voltage levels of pins VDD_ANA and VDD_BAT. If the voltage on these pins drops below the predefined threshold (defaulting to 2.4 V), the detector triggers signals to shut down certain power-consuming blocks (e.g., flash), ensuring that the digital module has sufficient time to save and transfer important data.

25.2 Feature List

- Two monitored sources
  - Monitors the voltage level of pin VDD_ANA, which supplies power to the analog circuit.
  - Monitors the voltage level of pin VDD_BAT, which connects the external battery to the chip.
- Two configurable monitoring modes
  - Mode 0: The brown-out detector triggers interrupts when the brown-out counter reaches the predefined threshold and selects the reset mode according to the configuration.
  - Mode 1: The brown-out detector triggers a system reset when the voltage falls below the threshold.
- Configurable voltage-monitoring thresholds and noise tolerance
- Configurable handling modes for under-voltage events

25.3 Architectural Overview

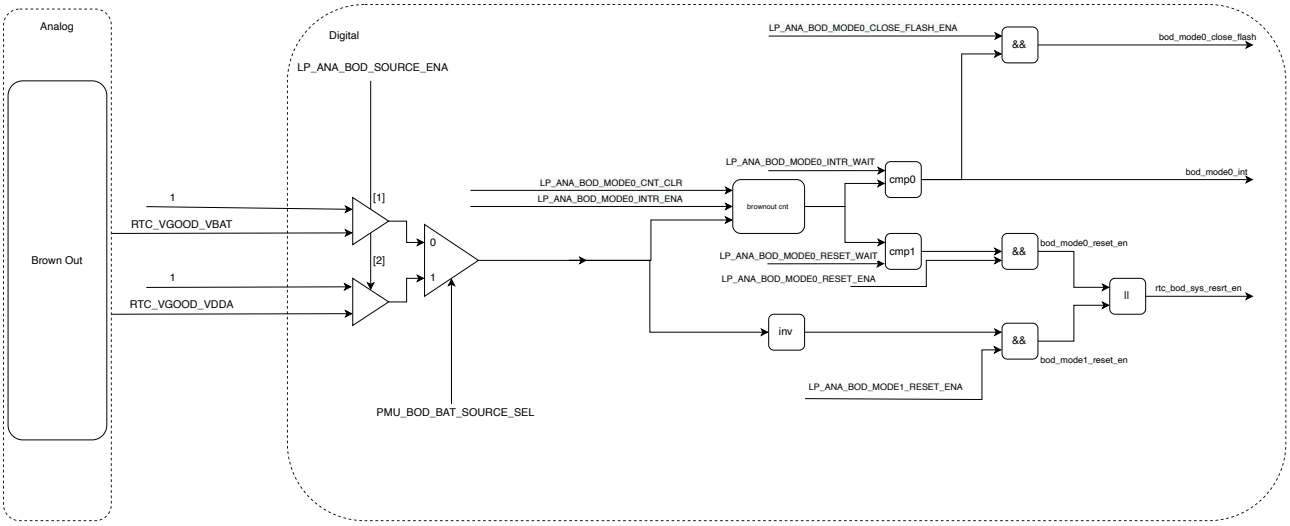


Figure 25.3-1. Brown-out Detector Overview

- Source selection for detection: The brown-out detector can select which power source to monitor by configuring LP_ANA_BOD_SOURCE_ENA. RTC_VGOOD_VBAT indicates the detection result of VDD_BAT, and RTC_VGOOD_VDDA indicates the detection result of VDD_ANA.
- Brown-out detector (Brown Out): An analog circuit designed to monitor the voltage of power pins and generate brown-out signals. It compares the voltage with the predefined threshold. When the voltage falls below this threshold, the detector outputs brown-out signals.
- Brown-out counter (brownout cnt): A counter used for filtering voltage noise on the detected power pins. When the brown-out signal is received, the counter starts counting based on the clock source LP_DYN_FAST_CLK. If the brown-out signal ceases during the counting process, the counter resets to 0. This counter can be cleared by configuring LP_ANA_BOD_MODE0_CNT_CLR.
- Interrupt comparator (cmp0): The comparator generates interrupts by comparing the counter value with the threshold. When the counter value exceeds the threshold, the comparator generates the interrupt signal bod_mode0_int. If LP_ANA_BOD_MODE0_CLOSE_FLASH_ENA is enabled, bod_mode0_close_flash will be triggered, causing flash to enter the SUSPEND state.
- Reset comparator (cmp1): The comparator triggers reset signals by comparing the counter value with the threshold. When the counter value exceeds the threshold, the comparator triggers reset signals.

25.4 Functional Description

- Monitored sources
  - VDD_ANA: Power pin for the analog circuit, supporting both Mode 0 and Mode 1.
  - VDD_BAT: Power pin connected to the battery, supporting both Mode 0 and Mode 1. In addition, it supports two configurable voltage thresholds:
    * VBAT_BOD: Used for brown-out detection of VDD_BAT

    * VBAT_CHARGER: Indicates whether the battery connected to VDD_BAT needs charging.

  These thresholds trigger different interrupts.

- Monitoring modes
  There are two monitoring modes based on the way the brown-out detector handles the brown-out signal:
  - Mode 0: When Mode 0 is enabled, the brown-out counter starts counting after detecting a brown-out signal. Interrupt and reset signals are generated when the counter reaches the thresholds defined by LP_ANA_BOD_MODE0_RESET_WAIT and LP_ANA_BOD_MODE0_INTR_WAIT, respectively. This helps filter out noise on the monitored pins. Configure LP_ANA_BOD_MODE0_RESET_SEL to select the reset options:
    * 0: Reset the chip.
    * 1: Reset the system.
  - Mode 1: Mode 1 is the default mode, which resets the system immediately upon detecting under-voltage.
- Voltage-monitoring thresholds
  The voltage-monitoring threshold is configured in the analog circuit by the analog I2C master. The threshold configuration registers are as follows. BIAS is the slave of the analog I2C master. The address of BIAS is 0x6a. REGXX indicates the register address of BIAS, and [x:x] indicates the bits of this register.
  - BIAS_OR_DREFL_VBAT (REG08[7:5]): threshold for releasing the under-voltage state of VDD_BAT
  - BIAS_OR_DREFH_VBAT (REG08[4:2]): under-voltage threshold of VDD_BAT
  - BIAS_OR_DREFL_VDDA (REG09[7:5]): threshold for releasing the under-voltage state of VDD_ANA
  - BIAS_OR_DREFH_VDDA (REG09[4:2]): under-voltage threshold of VDD_ANA
  - BIAS_OR_DREFL_VBAT_CHARGER (REG10[7:5]): threshold for releasing the under-voltage state of battery connected to VDD_BAT
  - BIAS_OR_DREFH_VBAT_CHARGER (REG10[4:2]): under-voltage threshold of battery connected to VDD_BAT

  The relationship between the threshold configuration of VBAT/VDDA and voltage is:
  - 0: 2.0 V
  - 1: 2.1 V
  - 2: 2.2 V
  - 3: 2.3 V
  - 4: 2.4 V
  - 5: 2.5 V
  - 6: 2.6 V
  - 7: 2.7 V

  The relationship between the threshold configuration of VBAT_CHARGER and voltage is:
  - 0: 2.2 V

- 1: 2.3 V
- 2: 2.4 V
- 3: 2.5 V
- 4: 2.6 V
- 5: 2.7 V
- 6: 2.8 V
- 7: 2.9 V

- Voltage-monitoring filter
  A dedicated glitch filter is used to process the detection results of VDD_BAT, as shown in the figure below.

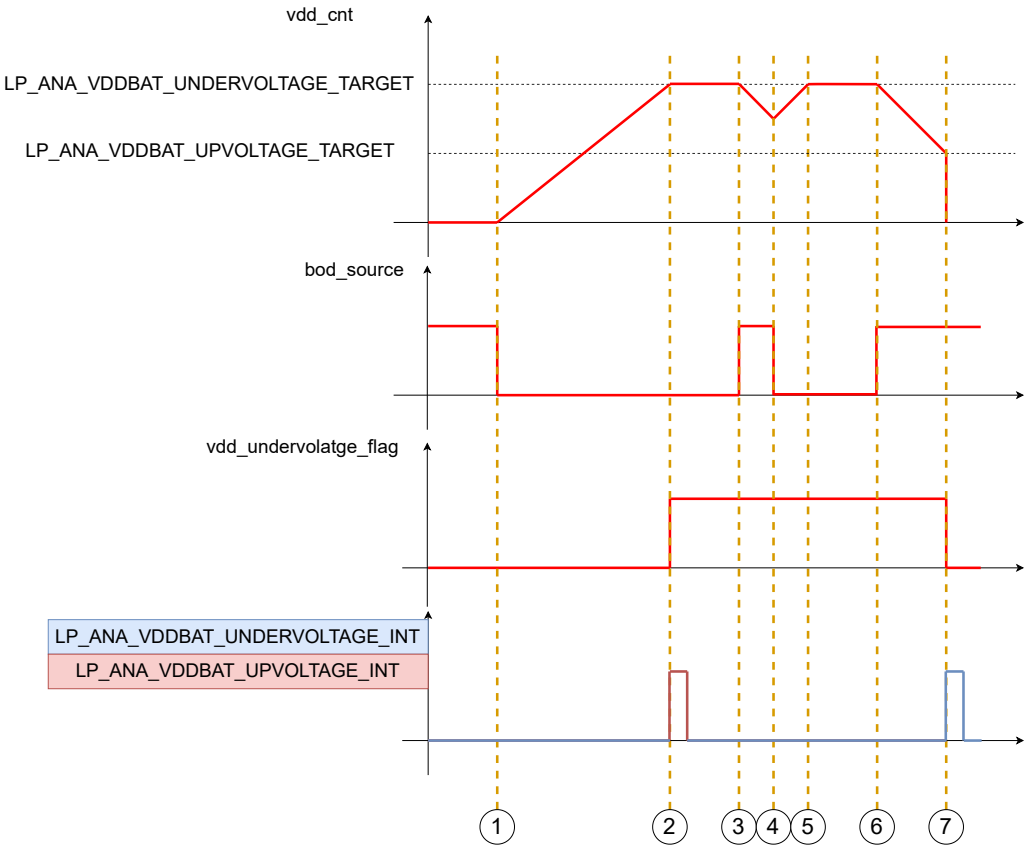


Figure 25.4-1. Structure of Brown-out Detector

- vdd_cnt: the value of the brown-out counter.
- bod_source: the monitored source, which is VDD_BAT in the figure.
- vdd_undervoltage_flag: the under-voltage flag.
- Stage 0 ~ 1: When bod_source is 1, it indicates the voltage is normal, and the brown-out counter (vdd_cnt) remains 0.
- Stage 1 ~ 2: When bod_source is 0, it indicates under-voltage happens, and the brown-out counter will start to count. When the counter value reaches the corresponding threshold, vdd_undervoltage_flag will be set to 1, and the interrupt is generated.

- Stage 2 ~ 3: When vdd_underboltage_flag is 1, the brown-out counter will stop counting, and stabilize at LP_ANA_VDDBAT_UNDERVOLTAGE_TARGET.
- Stage 3 ~ 4 ~ 5: When bod_source changes back to 0 for a short period, the value in the brown-out counter will decrease. However, as the value does not reach to LP_ANA_VDDBAT_UPVOLTAGE_TARGET, vdd_underboltage_flag will not be toggled.
- Stage 6 ~ 7: After bod_source remains 0 for a certain period, i.e., the counter value decreases to LP_ANA_VDDBAT_UPVOLTAGE_TARGET, vdd_underboltage_flag is reset to 0. Then the counter is cleared to 0.

25.5 Interrupts

ESP32-P4's brown-out detector can generate the following interrupt and send it to Interrupt Matrix.

- LP_ANAPERI_INT

The interrupt is generated by the internal interrupt sources of the brown-out detector.

- LP_ANA_VDDBAT_CHARGE_UPVOLTAGE_INT: Triggered when the voltage of VDD_BAT increases to the predefined charging threshold.
- LP_ANA_VDDBAT_CHARGE_UNDERVOLTAGE_INT: Triggered when the voltage of VDD_BAT decreases to the predefined charging threshold.
- LP_ANA_VDDBAT_UPVOLTAGE_INT: Triggered when the voltage of VDD_BAT increases to the predefined brown-out threshold.
- LP_ANA_VDDBAT_UNDERVOLTAGE_INT: Triggered when the voltage of VDD_BAT reduces to the predefined brown-out threshold.
- LP_ANA_BOD_MODE0_INT: Triggered when the value of the brown-out counter reaches the predefined threshold in Mode 0.

Each interrupt source can be configured by a common set of registers that are described in Chapter *Interrupt Configuration Registers*. The specific registers can be found in Section 25.7 *Register Summary*.

25.6 Programming Procedures

- Mode 0
  - Configure LP_ANA_BOD_MODE0_RESET_WAIT to set the counter threshold for triggering the reset signal.
  - Configure LP_ANA_BOD_MODE0_INTR_WAIT to set the counter threshold for triggering interrupts.
  - Set LP_ANA_BOD_MODE0_RESET_SEL to select the reset way in Mode 0.
  - Configure LP_ANA_BOD_SOURCE_ENA to enable the power pin to be detected.
  - Set LP_ANA_BOD_MODE0_RESET_ENA to enable the brown-out reset functionality.
  - Set LP_ANA_BOD_MODE0_INTR_ENA to enable the noise-filtering counter and interrupts.
- Mode 1

- Configure LP_ANA_BOD_SOURCE_ENA to enable the power pin to be detected.

25.7 Register Summary

The addresses in this section are relative to Brown-out Detector base address provided in Table 9.3-2 in Chapter 9 *System and Memory*.

The abbreviations given in Column **Access** are explained in Section *Access Types for Registers*.

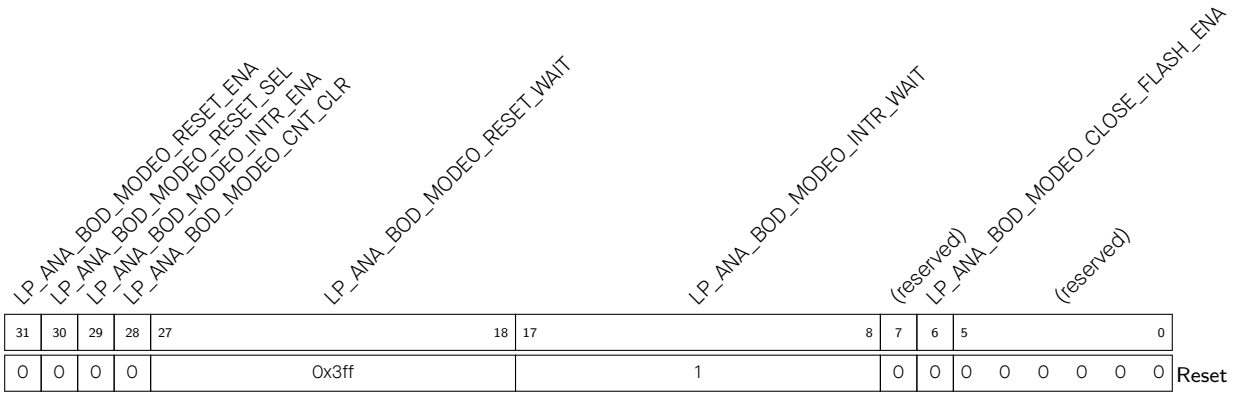
Name	Description	Address	Access
LP_ANA_BOD_MODE0_CNTL_REG	Control register of Mode 0	0x0000	R/W
LP_ANA_VDD_SOURCE_CNTL_REG	Control register of monitoring sources	0x0008	varies
LP_ANA_VDDBAT_BOD_CNTL_REG	Control register of brown-out detecting for VDD_BAT	0x000C	varies
LP_ANA_VDDBAT_CHARGE_CNTL_REG	Pre-charge control register for under-voltage of VDD_BAT	0x0010	varies
LP_ANA_INT_RAW_REG	Raw interrupt status	0x0020	RO
LP_ANA_INT_ST_REG	Interrupt state register	0x0024	RO
LP_ANA_INT_ENA_REG	Interrupt enable register	0x0028	R/W
LP_ANA_INT_CLR_REG	Interrupt clear register	0x002C	WT

25.8 Registers

The addresses in this section are relative to Brown-out Detector base address provided in Table 9.3-2 in Chapter 9 *System and Memory*.

For how to program reserved fields, please refer to Section Programming Reserved Register Field.

Register 25.1. LP_ANA_BOD_MODE0_CNTL_REG (0x0000)



LP_ANA_BOD_MODE0_CLOSE_FLASH_ENA Configure to close the flash functionality when under-voltage happens in Mode 0.

0: Disable

1: Enable

(R/W)

LP_ANA_BOD_MODE0_INTR_WAIT Configure the counter threshold that triggers interrupts in Mode 0. (R/W)

LP_ANA_BOD_MODE0_RESET_WAIT Configure the counter threshold that triggers reset in Mode 0. (R/W)

LP_ANA_BOD_MODE0_CNT_CLR Clear counter value in Mode 0. (R/W)

LP_ANA_BOD_MODE0_INTR_ENA Configure to enable the counter and interrupts in Mode 0.

0: Disable

1: Enable

(R/W)

LP_ANA_BOD_MODE0_RESET_SEL Configure the reset way when under-voltage happens in Mode 0.

0: Reset the chip

1: Reset the system

(R/W)

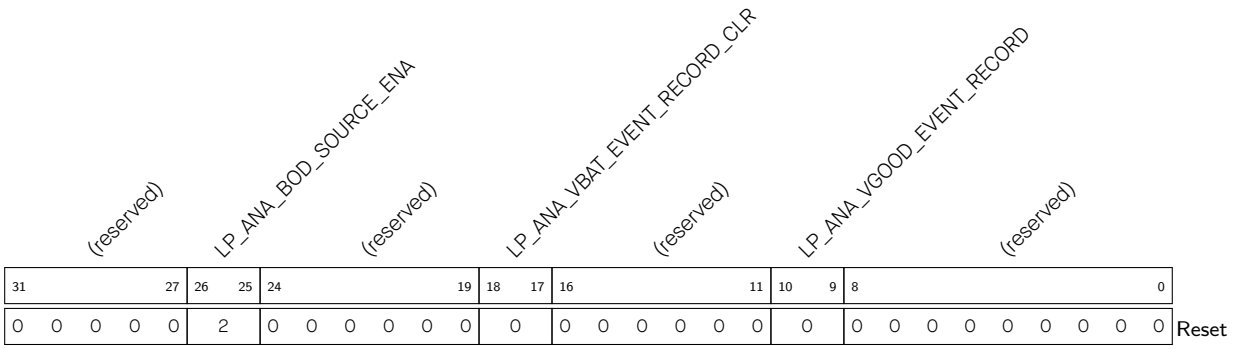
LP_ANA_BOD_MODE0_RESET_ENA Configure to enable reset in Mode 0.

0: Disable

1: Enable

(R/W)

Register 25.2. LP_ANA_VDD_SOURCE_CNTL_REG (0x0008)



LP_ANA_VGOOD_EVENT_RECORD Flags for indicating pins whose voltage is lower than the threshold.

0: Under-voltage does not happen on either VDD_BAT or VDD_ANA;

1: Under-voltage happens on VDD_BAT;

2: Under-voltage happens on VDD_ANA;

3: Under-voltage happens on both VDD_BAT and VDD_ANA.

(RO)

LP_ANA_VBAT_EVENT_RECORD_CLR Clear the flags.

0: Not clear

1: Clear the brown-out flag of VDD_BAT

2: Clear the brown-out flag of VDD_ANA

3: No effect

(WT)

LP_ANA_BOD_SOURCE_ENA Configure to detect monitoring sources.

0: Not detecting any pins;

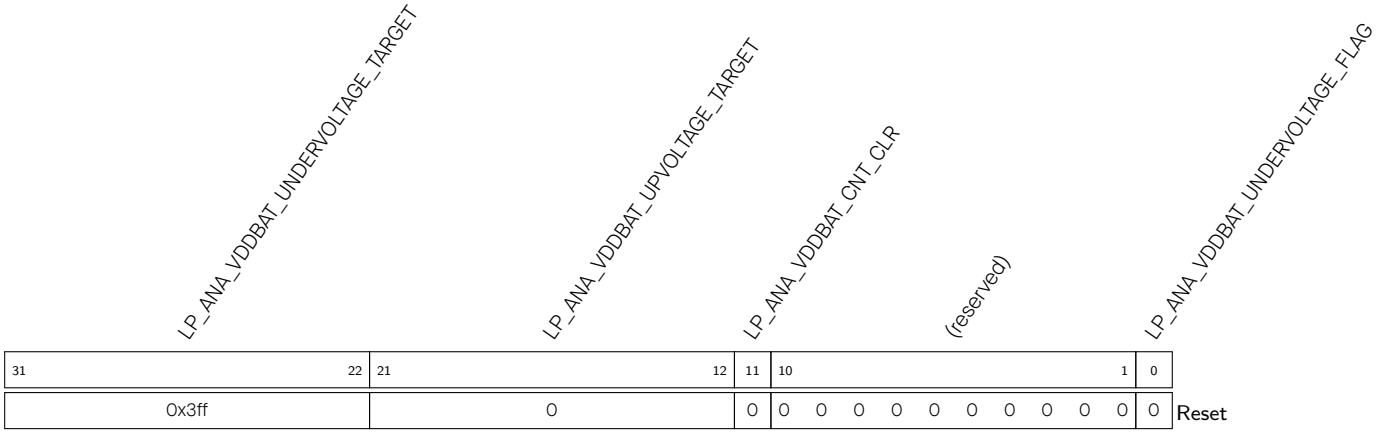
1: Only detect VDD_BAT;

2: Only detect VDD_ANA;

3: Detect both VDD_BAT and VDD_ANA.

(R/W)

Register 25.3. LP_ANA_VDDBAT_BOD_CNTL_REG (0x000C)



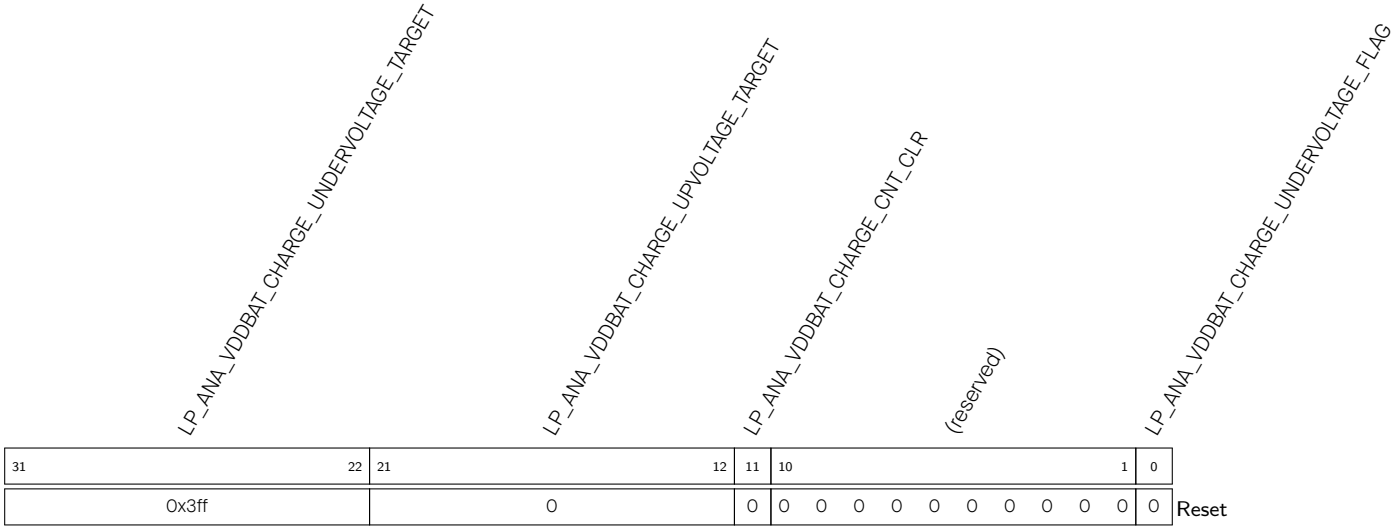
LP_ANA_VDDBAT_UNDERVOLTAGE_FLAG Flag indicating that under-voltage occurs on VDD_BAT. (RO)

LP_ANA_VDDBAT_CNT_CLR Clear the brown-out counter of VDD_BAT. (WT)

LP_ANA_VDDBAT_UPVOLTAGE_TARGET Configure the voltage-recovery threshold for the brown-out counter of VDD_BAT. When the counter decreases to this threshold, LP_ANA_VDDBAT_UNDERVOLTAGE_FLAG is cleared to 0. (R/W)

LP_ANA_VDDBAT_UNDERVOLTAGE_TARGET Configure the brown-out threshold for the brown-out counter of VDD_BAT. When the counter increases to this threshold, LP_ANA_VDDBAT_UNDERVOLTAGE_FLAG is set to 1. (R/W)

Register 25.4. LP_ANA_VDDBAT_CHARGE_CNTL_REG (0x0010)



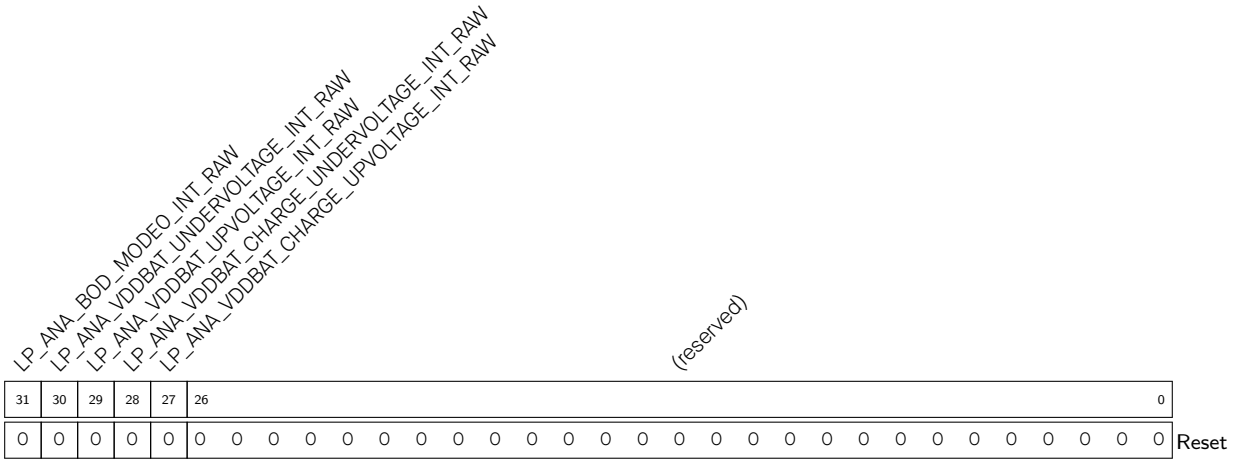
LP_ANA_VDDBAT_CHARGE_UNDERVOLTAGE_FLAG Flag indicating that VDD_BAT is charging. (RO)

LP_ANA_VDDBAT_CHARGE_CNT_CLR Clear the brown-out counter of VDD_BAT. (WT)

LP_ANA_VDDBAT_CHARGE_UPVOLTAGE_TARGET Configure the voltage-recovery threshold for the brown-out counter of VDD_BAT. When the counter decreases to this threshold, LP_ANA_VDDBAT_CHARGE_UNDERVOLTAGE_FLAG is cleared to 0. (R/W)

LP_ANA_VDDBAT_CHARGE_UNDERVOLTAGE_TARGET Configure the brown-out threshold for the brown-out counter of VDD_BAT. When the counter increases to this threshold, LP_ANA_VDDBAT_CHARGE_UNDERVOLTAGE_FLAG is set to 1. (R/W)

Register 25.5. LP_ANA_INT_RAW_REG (0x0020)



LP_ANA_VDDBAT_CHARGE_UPVOLTAGE_INT_RAW The interrupt is triggered when the voltage of VDD_BAT increases to the charging threshold. (RO)

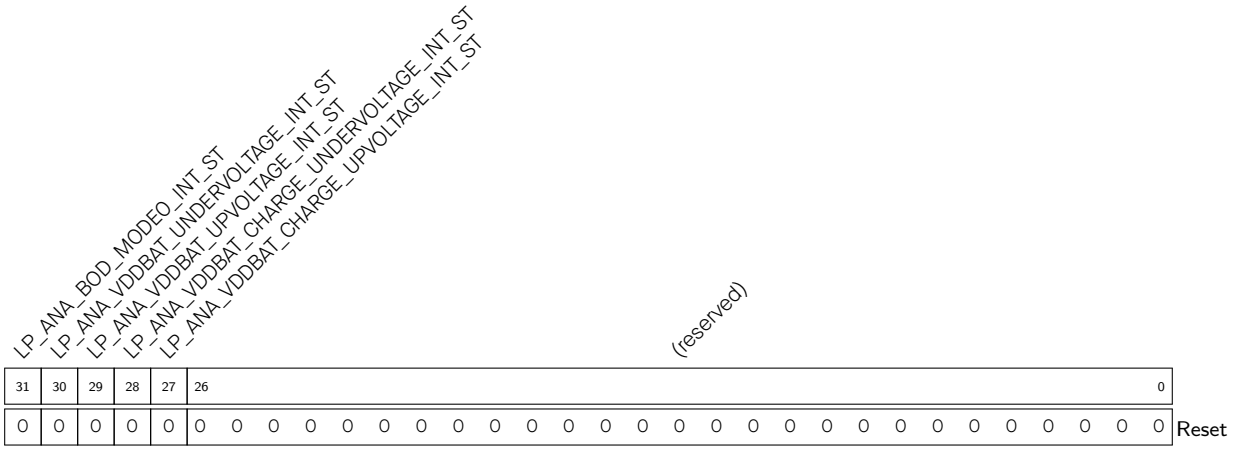
LP_ANA_VDDBAT_CHARGE_UNDERVOLTAGE_INT_RAW The interrupt is triggered when the voltage of VDD_BAT decreases to the charging threshold. (RO)

LP_ANA_VDDBAT_UPVOLTAGE_INT_RAW The interrupt is triggered when the voltage of VDD_BAT increases to the brown-out threshold. (RO)

LP_ANA_VDDBAT_UNDERVOLTAGE_INT_RAW The interrupt is triggered when the voltage of VDD_BAT decreases to the brown-out threshold. (RO)

LP_ANA_BOD_MODE0_INT_RAW Brown-out interrupt in Mode 0. (RO)

Register 25.6. LP_ANA_INT_ST_REG (0x0024)



LP_ANA_VDDBAT_CHARGE_UPVOLTAGE_INT_ST The interrupt is triggered and sent to CPU when the voltage of VDD_BAT increases to the charging threshold. (RO)

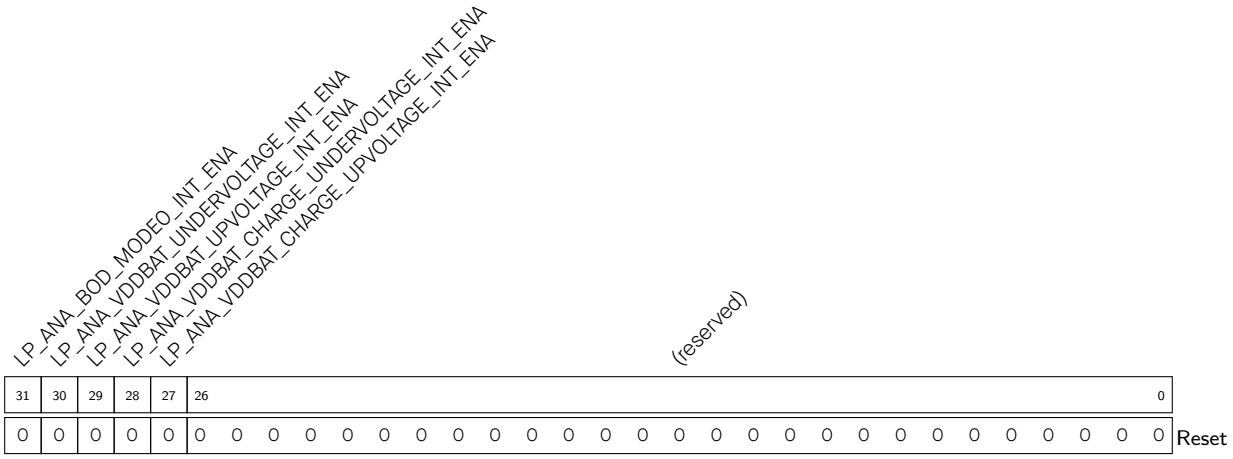
LP_ANA_VDDBAT_CHARGE_UNDERVOLTAGE_INT_ST The interrupt is triggered and sent to CPU when the voltage of VDD_BAT decreases to the charging threshold. (RO)

LP_ANA_VDDBAT_UPVOLTAGE_INT_ST The interrupt is triggered and sent to CPU when the voltage of VDD_BAT increases to the brown-out threshold. (RO)

LP_ANA_VDDBAT_UNDERVOLTAGE_INT_ST The interrupt is triggered and sent to CPU when the voltage of VDD_BAT decreases to the brown-out threshold. (RO)

LP_ANA_BOD_MODE0_INT_ST The brown-out interrupt in Mode 0 is triggered and sent to CPU. (RO)

Register 25.7. LP_ANA_INT_ENA_REG (0x0028)



LP_ANA_VDDBAT_CHARGE_UPVOLTAGE_INT_ENA Configure whether to enable the interrupt which is triggered when the voltage of VDD_BAT increases to the charging threshold.

0: No

1: Yes

(R/W)

LP_ANA_VDDBAT_CHARGE_UNDERVOLTAGE_INT_ENA Configures whether to enable the interrupt which is triggered when the voltage of VDD_BAT reduces to the charging threshold.

0: Yes

1: No

(R/W)

LP_ANA_VDDBAT_UPVOLTAGE_INT_ENA Configures whether to enable the interrupt which is triggered when the voltage of VDD_BAT increases to the brown-out threshold.

0: Yes

1: No

(R/W)

LP_ANA_VDDBAT_UNDERVOLTAGE_INT_ENA Configures whether to enable the interrupt which is triggered when the voltage of VDD_BAT reduces to the brown-out threshold.

0: No

1: Yes

(R/W)

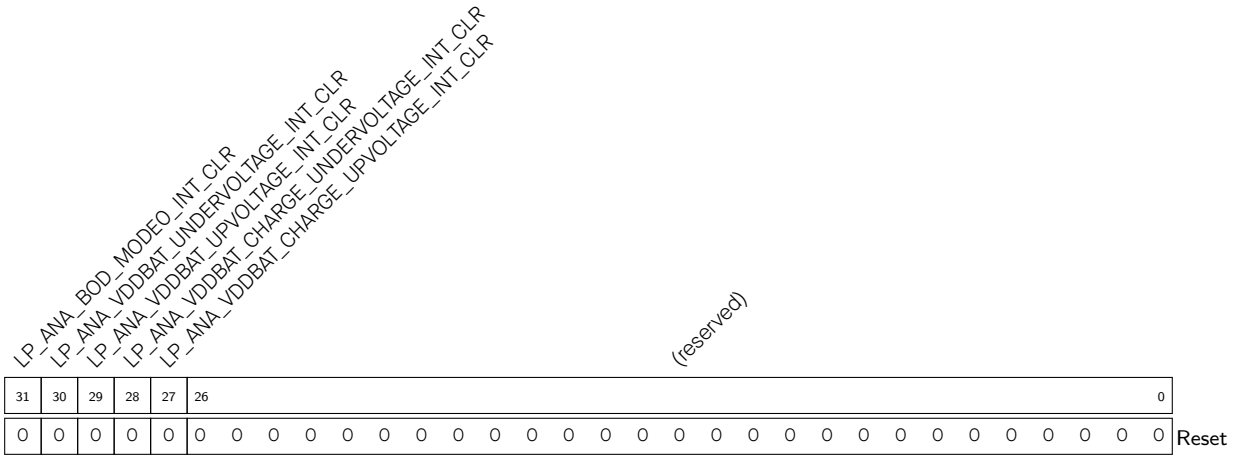
LP_ANA_BOD_MODE0_INT_ENA Configures whether to enable the brown-out interrupt in Mode 0.

0: No

1: Yes

(R/W)

Register 25.8. LP_ANA_INT_CLR_REG (0x002C)



LP_ANA_VDDBAT_CHARGE_UPVOLTAGE_INT_CLR Configures whether to clear the interrupt triggered by the VDD_BAT voltage increasing to the charging threshold. (WT)

LP_ANA_VDDBAT_CHARGE_UNDERVOLTAGE_INT_CLR Configures whether to clear the interrupt triggered by the VDD_BAT voltage reducing to the charging threshold. (WT)

LP_ANA_VDDBAT_UPVOLTAGE_INT_CLR Configures whether to clear the interrupt triggered by the VDD_BAT voltage increasing to the brown-out threshold. (WT)

LP_ANA_VDDBAT_UNDERVOLTAGE_INT_CLR Configures whether to clear the interrupt triggered by the VDD_BAT voltage reducing to the brown-out threshold. (WT)

LP_ANA_BOD_MODE0_INT_ST Configures whether to clear the brown-out interrupt in Mode 0. (WT)

Chapter 26

Power Supply Detector (PSDET)

26.1 Overview

The ESP32-P4 chip integrates a power supply detector, which is used to monitor the voltage of the on-chip HP system regulator. Specifically, this function is implemented by a voltage glitch detector, which prevents abnormal fluctuations and glitch attacks by real-time monitoring of the supply voltage. The power supply detector can operate continuously during the HP system operation.

26.2 Features

- Maintains monitoring during the chip's normal operation without requiring software intervention
- Automatically triggers a system reset when a voltage glitch of approximately 50 ns or longer is detected

26.3 Functional Description

The structure of the power supply detector based on voltage glitch detection is shown below:



Figure 26.3-1. Structure of the ESP32-P4 Power Supply Detector

The power supply detector of ESP32-P4 can continuously monitor the voltage of the power supply network while the chip is working. When an instantaneous voltage glitch—either with a fluctuation depth greater than 100 mV and a duration longer than 50 ns is detected, it automatically triggers a system reset. The detection does not require register configuration. System resets triggered by this function will be recorded by the HP CPU0 reset cause register LP_CLKRST_HPCORE0_RESET_CAUSE, corresponding to the reset value 0x1B.

This function is disabled by default. To enable it, please flash the EFUSE_PVT_GLITCH_EN and EFUSE_PVT_GLITCH_MODE fields. For more information about eFuses, please refer to Chapter 9.4 *eFuse Controller (EFUSE)*.

For information about reset, please refer to Chapter 11 *Reset and Clock*.

Part V

Cryptography/Security Component

Dedicated to security features, this part explores cryptographic accelerators like AES and ECC. It also covers digital signatures, random number generation, and encryption/decryption algorithms, showcasing the SoC's capabilities in cryptography and secure data processing.

Chapter 27

AES Accelerator (AES)

27.1 Introduction

ESP32-P4 integrates an Advanced Encryption Standard (AES) accelerator, which is a hardware device that speeds up computation using AES algorithm significantly, compared to AES algorithms implemented solely in software. The AES accelerator integrated in ESP32-P4 has two working modes, which are Typical AES and DMA-AES.

27.2 Features

The following functionality is supported:

- Typical AES working mode
  - AES-128/AES-256 encryption and decryption
- DMA-AES working mode
  - AES-128/AES-256 encryption and decryption
  - Block cipher mode
    - ECB (Electronic Codebook)
    - CBC (Cipher Block Chaining)
    - OFB (Output Feedback)
    - CTR (Counter)
    - CFB8 (8-bit Cipher Feedback)
    - CFB128 (128-bit Cipher Feedback)
  - GCM (Galois/Counter Mode)
  - Interrupt on completion of computation

27.3 Clock and Reset

The AES accelerator is activated by setting the HP_SYS_CLKRST_CRYPTO_AES_CLK_EN bit in the HP_SYS_CLKRST_PERI_CLK_CTRL25_REG register and clearing the HP_SYS_CLKRST_RST_EN_AES bit in the HP_SYS_CLKRST_HP_RST_EN2_REG register. Besides, due to resource reuse between cryptography accelerator modules, users also need to additionally clear the HP_SYS_CLKRST_RST_EN_DS bit and the HP_SYS_CLKRST_RST_EN_KM bit in the HP_SYS_CLKRST_HP_RST_EN2_REG register.

27.4 AES Working Modes

The AES accelerator integrated in ESP32-P4 has two working modes, which are Typical AES and DMA-AES.

- Typical AES Working Mode:
  - Supports encryption and decryption using cryptographic keys of 128 and 256 bits, specified in NIST FIPS 197.

  In this working mode, the plaintext and ciphertext is written and read via CPU directly.
- DMA-AES Working Mode:
  - Supports encryption and decryption using cryptographic keys of 128 and 256 bits, specified in NIST FIPS 197;
  - Supports block cipher modes ECB, CBC, OFB, CTR, CFB8, and CFB128 under NIST SP 800-38A.
  - Supports GCM under NIST SP 800-38D.

  In this working mode, the plaintext and ciphertext are written and read via DMA. An interrupt will be generated when operation completes.

Users can choose the working mode for AES accelerator by configuring the AES_DMA_ENABLE_REG register according to Table 27.4-1 below.

Table 27.4-1. AES Accelerator Working Mode

AES_DMA_ENABLE_REG	Working Mode
0	Typical AES
1	DMA-AES

Users can choose the length of cryptographic keys and encryption/decryption by configuring the AES_MODE_REG register according to Table 27.4-2 below.

Table 27.4-2. Key Length and Encryption/Decryption

AES_MODE_REG[2:0]	Key Length and Encryption/Decryption
0	AES-128 encryption
1	reserved
2	AES-256 encryption
3	reserved
4	AES-128 decryption
5	reserved
6	AES-256 decryption
7	reserved

For a detailed introduction to these two working modes, please refer to Section 27.5 and Section 27.6 below.

Notice:

ESP32-P4's RSA Digital Signature Peripheral (RSA_DS) module will call the AES accelerator. Therefore, users cannot access the AES accelerator when RSA Digital Signature Peripheral (RSA_DS) module is working.

27.5 Typical AES Working Mode

In the Typical AES working mode, users can check the working status of the AES accelerator by inquiring the AES_STATE_REG register and comparing the return value against the Table 27.5-1 below.

Table 27.5-1. Working Status under Typical AES Working Mode

AES_STATE_REG	Status	Description
0	IDLE	The AES accelerator is idle or completed operation.
1	WORK	The AES accelerator is in the middle of an operation.

27.5.1 Key, Plaintext, and Ciphertext

The encryption or decryption key is stored in AES_KEY_*n*_REG, which is a set of eight 32-bit registers.

- For AES-128 encryption or decryption, the 128-bit key is stored in AES_KEY_0_REG ~ AES_KEY_3_REG.
- For AES-256 encryption or decryption, the 256-bit key is stored in AES_KEY_0_REG ~ AES_KEY_7_REG.

The plaintext and ciphertext are stored in AES_TEXT_IN_*m*_REG and AES_TEXT_OUT_*m*_REG, which are two sets of four 32-bit registers.

- For AES-128 or AES-256 encryption, the AES_TEXT_IN_*m*_REG registers are initialized with plaintext. Then, the AES accelerator stores the ciphertext into AES_TEXT_OUT_*m*_REG after operation.
- For AES-128 or AES-256 decryption, the AES_TEXT_IN_*m*_REG registers are initialized with ciphertext. Then, the AES accelerator stores the plaintext into AES_TEXT_OUT_*m*_REG after operation.

27.5.2 Endianness

Text Endianness

In Typical AES working mode, the AES accelerator uses cryptographic keys to encrypt and decrypt data in blocks of 128 bits. When filling data into AES_TEXT_IN_*m*_REG register or reading result from AES_TEXT_OUT_*m*_REG registers, users should follow the text endianness type specified in Table 27.5-2.

Table 27.5-2. Text Endianness Type for Typical AES

Plaintext/Ciphertext					
State[1]		c[2]			
		0	1	2	3
r	0	AES_TEXT_*x*_0_REG[7:0]	AES_TEXT_*x*_1_REG[7:0]	AES_TEXT_*x*_2_REG[7:0]	AES_TEXT_*x*_3_REG[7:0]
	1	AES_TEXT_*x*_0_REG[15:8]	AES_TEXT_*x*_1_REG[15:8]	AES_TEXT_*x*_2_REG[15:8]	AES_TEXT_*x*_3_REG[15:8]
	2	AES_TEXT_*x*_0_REG[23:16]	AES_TEXT_*x*_1_REG[23:16]	AES_TEXT_*x*_2_REG[23:16]	AES_TEXT_*x*_3_REG[23:16]

Table 27.5-2. Text Endianness Type for Typical AES

Plaintext/Ciphertext					
	3	AES_TEXT_*x*_0_REG[31:24]	AES_TEXT_*x*_1_REG[31:24]	AES_TEXT_*x*_2_REG[31:24]	AES_TEXT_*x*_3_REG[31:24]

[1] The definition of "State (including c and r)" is described in Section 3.4 *The State* in NIST FIPS 197.
[2] Where *x* = IN or OUT.

Key Endianness

In Typical AES working mode, when filling keys into AES_KEY_*m*_REG registers, users should follow the key endianness type specified in Table 27.5-3 and Table 27.5-4.

Table 27.5-3. Key Endianness Type for AES-128 Encryption and Decryption

Bit[1]	w[0]	w[1]	w[2]	w[3][2]
[31:24]	AES_KEY_0_REG[7:0]	AES_KEY_1_REG[7:0]	AES_KEY_2_REG[7:0]	AES_KEY_3_REG[7:0]
[23:16]	AES_KEY_0_REG[15:8]	AES_KEY_1_REG[15:8]	AES_KEY_2_REG[15:8]	AES_KEY_3_REG[15:8]
[15:8]	AES_KEY_0_REG[23:16]	AES_KEY_1_REG[23:16]	AES_KEY_2_REG[23:16]	AES_KEY_3_REG[23:16]
[7:0]	AES_KEY_0_REG[31:24]	AES_KEY_1_REG[31:24]	AES_KEY_2_REG[31:24]	AES_KEY_3_REG[31:24]

[1] Column "Bit" specifies the bytes of each word stored in w[0] ~ w[3].

[2] w[0] ~ w[3] are "the first Nk words of the expanded key" as specified in Section 5.2 *Key Expansion* in NIST FIPS 197.

Table 27.5-4. Key Endianness Type for AES-256 Encryption and Decryption

Bit [1]	w[0]	w[1]	w[2]	w[3]	w[4]	w[5]	w[6]	w[7] [2]
[31:24]	AES_KEY_0_REG[7:0]	AES_KEY_1_REG[7:0]	AES_KEY_2_REG[7:0]	AES_KEY_3_REG[7:0]	AES_KEY_4_REG[7:0]	AES_KEY_5_REG[7:0]	AES_KEY_6_REG[7:0]	AES_KEY_7_REG[7:0]
[23:16]	AES_KEY_0_REG[15:8]	AES_KEY_1_REG[15:8]	AES_KEY_2_REG[15:8]	AES_KEY_3_REG[15:8]	AES_KEY_4_REG[15:8]	AES_KEY_5_REG[15:8]	AES_KEY_6_REG[15:8]	AES_KEY_7_REG[15:8]
[15:8]	AES_KEY_0_REG[23:16]	AES_KEY_1_REG[23:16]	AES_KEY_2_REG[23:16]	AES_KEY_3_REG[23:16]	AES_KEY_4_REG[23:16]	AES_KEY_5_REG[23:16]	AES_KEY_6_REG[23:16]	AES_KEY_7_REG[23:16]
[7:0]	AES_KEY_0_REG[31:24]	AES_KEY_1_REG[31:24]	AES_KEY_2_REG[31:24]	AES_KEY_3_REG[31:24]	AES_KEY_4_REG[31:24]	AES_KEY_5_REG[31:24]	AES_KEY_6_REG[31:24]	AES_KEY_7_REG[31:24]

[1] Column "Bit" specifies the bytes of each word stored in w[0] ~ w[7].

[2] w[0] ~ w[7] are "the first Nk words of the expanded key" as specified in Chapter 5.2 *Key Expansion* in NIST FIPS 197.

27.5.3 Operation Process

Single Operation

1. Write 0 to the AES_DMA_ENABLE_REG register.
2. Initialize registers AES_MODE_REG, AES_KEY_*n*_REG, and AES_TEXT_IN_*m*_REG.
3. Start operation by writing 1 to the AES_TRIGGER_REG register.
4. Wait till the content of the AES_STATE_REG register becomes 0, which indicates the operation is completed.
5. Read results from the AES_TEXT_OUT_*m*_REG register.

Consecutive Operations

In consecutive operations, primarily the input AES_TEXT_IN_*m*_REG and output AES_TEXT_OUT_*m*_REG registers (*m*: 0-3) are being written and read, while the content of AES_DMA_ENABLE_REG, AES_MODE_REG, and AES_KEY_*n*_REG is kept unchanged. Therefore, the initialization can be simplified during the consecutive operation.

1. Write 0 to the AES_DMA_ENABLE_REG register before starting the first operation.
2. Initialize registers AES_MODE_REG and AES_KEY_*n*_REG before starting the first operation.
3. Update the content of AES_TEXT_IN_*m*_REG.
4. Start operation by writing 1 to the AES_TRIGGER_REG register.
5. Wait till the content of the AES_STATE_REG register becomes 0, which indicates the operation completes.
6. Read results from the AES_TEXT_OUT_*m*_REG register, and return to Step 3 to continue the next operation.

27.6 DMA-AES Working Mode

In the DMA-AES working mode, the AES accelerator supports six block cipher modes: ECB, CBC, OFB, CTR, CFB8, and CFB128. Users can choose the block cipher mode by configuring the AES_BLOCK_MODE_REG register according to Table 27.6-1 below.

Table 27.6-1. Block Cipher Mode

AES_BLOCK_MODE_REG[2:0]	Block Cipher Mode
0	ECB (Electronic Codebook)
1	CBC (Cipher Block Chaining)
2	OFB (Output Feedback)
3	CTR (Counter)
4	CFB8 (8-bit Cipher Feedback)
5	CFB128 (128-bit Cipher Feedback)
6	GCM (Galois/Counter Mode)
7	reserved

Users can check the working status of the AES accelerator by inquiring the AES_STATE_REG register and comparing the return value against the Table 27.6-2 below.

Table 27.6-2. Working Status under DMA-AES Working mode

AES_STATE_REG[1:0]	Status	Description
0	IDLE	The AES accelerator is idle.
1	WORK	The AES accelerator is in the middle of an operation.
2	DONE	The AES accelerator completed operations.

When working in the DMA-AES working mode, the AES accelerator supports interrupt on the completion of computation. To enable this function, write 1 to the AES_INT_ENA_REG register. By default, the interrupt function is disabled. Also, note that the interrupt should be cleared by software after use.

27.6.1 Key, Plaintext, and Ciphertext

Block Operation

During the block operations, the AES accelerator reads source data from DMA, and writes result data to DMA after the computation.

- For encryption, DMA reads plaintext from memory, then passes it to AES as source data. After computation, AES passes ciphertext as result data back to DMA to write into memory.
- For decryption, DMA reads ciphertext from memory, then passes it to AES as source data. After computation, AES passes plaintext as result data back to DMA to write into memory.

During block operations, the lengths of the source data and result data are the same. The total computation time is reduced because the DMA data operation and AES computation can happen concurrently.

The length of source data for AES accelerator under DMA-AES working mode must be 128 bits or the integral multiples of 128 bits. Otherwise, trailing zeros will be added to the original source data, so the length of source data equals to the nearest integral multiples of 128 bits. Please see details in Table 27.6-3 below.

Table 27.6-3. TEXT-PADDING

Function : TEXT-PADDING( )
Input : X, bit string.
Output : Y = **TEXT-PADDING**(X), whose length is the nearest integral multiples of 128 bits.

Steps
Let us assume that X is a data-stream that can be split into *n* parts as following:
$X = X_1 \| \| X_2 \| \| \cdots \| \| X_{n-1} \| \| X_n$
Here, the lengths of $X_1, X_2, \cdots, X_{n-1}$ all equal to 128 bits, and the length of X_n is *t* (0<=*t*<=127).
If $t = 0$, then
TEXT-PADDING(X) = X;
If $0 < t <= 127$, define a 128-bit block, X_n^*, and let $X_n^* = X_n \| \| 0^{128-t}$, then
TEXT-PADDING(X) = $X_1 \| \| X_2 \| \| \cdots \| \| X_{n-1} \| \| X_n^*$ = $X \| \| 0^{128-t}$

27.6.2 Endianness

Under the DMA-AES working mode, the transmission of source data and result data for AES accelerator is solely controlled by DMA. Therefore, the AES accelerator cannot control the Endianness of the source data and result data, but does have requirement on how these data should be stored in memory and on the length of the data.

For example, let us assume DMA needs to write the following data into memory at address 0x0280.

- Data represented in hexadecimal:
  - 000102030405060708090A0B0C0D0E0F101112131415161718191A1B1C1D1E1F20
- Data Length:
  - Equals to 2 blocks.

Then, this data will be stored in memory as shown in Table 27.6-4 below.

Table 27.6-4. Text Endianness for DMA-AES

Address	Byte	Address	Byte	Address	Byte	Address	Byte
0x0280	0x01	0x0281	0x02	0x0282	0x03	0x0283	0x04
0x0284	0x05	0x0285	0x06	0x0286	0x07	0x0287	0x08
0x0288	0x09	0x0289	0x0A	0x028A	0x0B	0x028B	0x0C
0x028C	0x0D	0x028D	0x0E	0x028E	0x0F	0x028F	0x10
0x0290	0x11	0x0291	0x12	0x0292	0x13	0x0293	0x14
0x0294	0x15	0x0295	0x16	0x0296	0x17	0x0297	0x18
0x0298	0x19	0x0299	0x1A	0x029A	0x1B	0x029B	0x1C
0x029C	0x1D	0x029D	0x1E	0x029E	0x1F	0x029F	0x20

27.6.3 Standard Incrementing Function

AES accelerator provides two Standard Incrementing Functions for the CTR block operation, which are INC_{32} and INC_{128} Standard Incrementing Functions. By setting the AES_INC_SEL_REG register to 0 or 1, users can choose the INC_{32} or INC_{128} functions respectively. For details on the Standard Incrementing Function, please see Chapter B.1 *The Standard Incrementing Function* in NIST SP 800-38A.

27.6.4 Block Number

Register AES_BLOCK_NUM_REG stores the Block Number of plaintext P or ciphertext C. The length of this register equals to length(**TEXT-PADDING**(P))/128 or length(**TEXT-PADDING**(C))/128. The AES accelerator only uses this register when working in the DMA-AES mode.

27.6.5 Initialization Vector

AES_IV_MEM is a 16-byte memory, which is only available for AES accelerator working in block operations. For CBC/OFB/CFB8/CFB128 operations, the AES_IV_MEM memory stores the Initialization Vector (IV). For the CTR operation, the AES_IV_MEM memory stores the Initial Counter Block (ICB).

Both IV and ICB are 128-bit strings, which can be divided into Byte0, Byte1, Byte2 $\cdots$ Byte15 (from left to right). AES_IV_MEM stores data following the Endianness pattern presented in Table 27.6-4, i.e., the most significant (i.e., left-most) byte Byte0 is stored at the lowest address while the least significant (i.e., right-most) byte Byte15 at the highest address.

For more details on IV and ICB, please refer to NIST SP 800-38A.

27.6.6 Block Operation Process

1. Select one of DMA channels to connect with AES, configure the DMA linked list, and then start DMA. For details, please refer to Chapter 6 *GDMA Controller (GDMA-AHB, GDMA-AXI)*.
2. Initialize the AES accelerator-related registers:
   - Write 1 to the AES_DMA_ENABLE_REG register.
   - Configure the AES_INT_ENA_REG register to enable or disable the interrupt function.
   - Initialize registers AES_MODE_REG and AES_KEY_*n*_REG.
   - Select block cipher mode by configuring the AES_BLOCK_MODE_REG register. For details, see Table 27.6-1.
   - Initialize the AES_BLOCK_NUM_REG register. For details, see Section 27.6.4.
   - Initialize the AES_INC_SEL_REG register (only needed when AES accelerator is working under CTR block operation).
   - Initialize the AES_IV_MEM memory (This is always needed except for ECB block operation).
3. Start operation by writing 1 to the AES_TRIGGER_REG register.
4. Wait for the completion of computation, which happens when the content of AES_STATE_REG becomes 2 or the AES interrupt occurs.

5. Check if DMA completes data transmission from AES to memory. At this time, DMA had already written the result data in memory, which can be accessed directly. For details on DMA, please refer to Chapter 6 *GDMA Controller (GDMA-AHB, GDMA-AXI)*.
6. Clear interrupt by writing 1 to the AES_INT_CLR_REG register, if any AES interrupt occurred during the computation.
7. Release the AES accelerator by writing 1 to the AES_DMA_EXIT_REG register. After this, the content of the AES_STATE_REG register becomes 0. Note that, you can release DMA earlier, but only after Step 4 is completed.

27.6.7 GCM Operation Process

1. Configure DMA chain and start DMA. For details on DMA, please refer to Chapter 6 *GDMA Controller (GDMA-AHB, GDMA-AXI)*.
2. Initialize the AES accelerator-related registers:
   - Write 1 to the AES_DMA_ENABLE_REG register.
   - Configure the AES_INT_ENA_REG register to enable or disable the interrupt function.
   - Initialize registers AES_MODE_REG and AES_KEY_*n*_REG.
   - Write 6 to the AES_BLOCK_MODE_REG register.
   - Initialize the AES_BLOCK_NUM_REG register. Details about this register are described in Section 27.6.4 *Block Number*.
   - Initialize the AES_AAD_BLOCK_NUM_REG register. Details about this register are described in Section 27.7.4 *AAD Block Number*.
   - Initialize the AES_REMAINDER_BIT_NUM_REG register. Details about this register is described in Section 27.7.5 *Number of Effective Bits of Incomplete Blocks*.
3. Start operation by writing 1 to the AES_TRIGGER_REG register.
4. Wait for the completion of computation, which happens when the content of AES_STATE_REG becomes 2. For details on the working status of AES Accelerator, please refer to Table 27.6-2 *Working Status under DMA-AES Working mode*. At this step, no interrupt occurs.
5. Obtain the H value from the AES_H_MEM memory.
6. Generate J_0 and write it to the AES_J0_MEM memory.
7. Continue operating by writing 1 to the AES_CONTINUE_OP_REG register.
8. Wait for the completion of computation, which happens when the content of AES_STATE_REG becomes 2 or the AES interrupt occurs. For details on the working status of AES Accelerator, please refer to Table 27.6-2 *Working Status under DMA-AES Working mode*.
9. Obtain T_0 by reading AES_T0_MEM.
10. Check if DMA completes data transmission from AES to memory. At this time, DMA had already written the result data in memory, which can be accessed directly. For details on DMA, please refer to Chapter 6 *GDMA Controller (GDMA-AHB, GDMA-AXI)*.

11. Clear interrupt by writing 1 to the AES_INT_CLR_REG register, if any AES interrupt occurred during the computation.
12. Exit DMA by writing 1 to the AES_DMA_EXIT_REG register. After this, the content of the AES_STATE_REG becomes 0. Note that, you can exit DMA earlier, but only after Step 8 is completed.

27.7 GCM Algorithm

ESP32-P4's AES accelerator fully supports GCM Algorithm. In reality, the AAD, C and P that are longer than 2^{32}-1 bits are seldom used. Therefore, we specify that the length of AAD, C and P should be no longer than 2^{32}-1 here. Figure 27.7-1 below demonstrates how GCM encryption is implemented in the AES Accelerator of ESP32-P4.

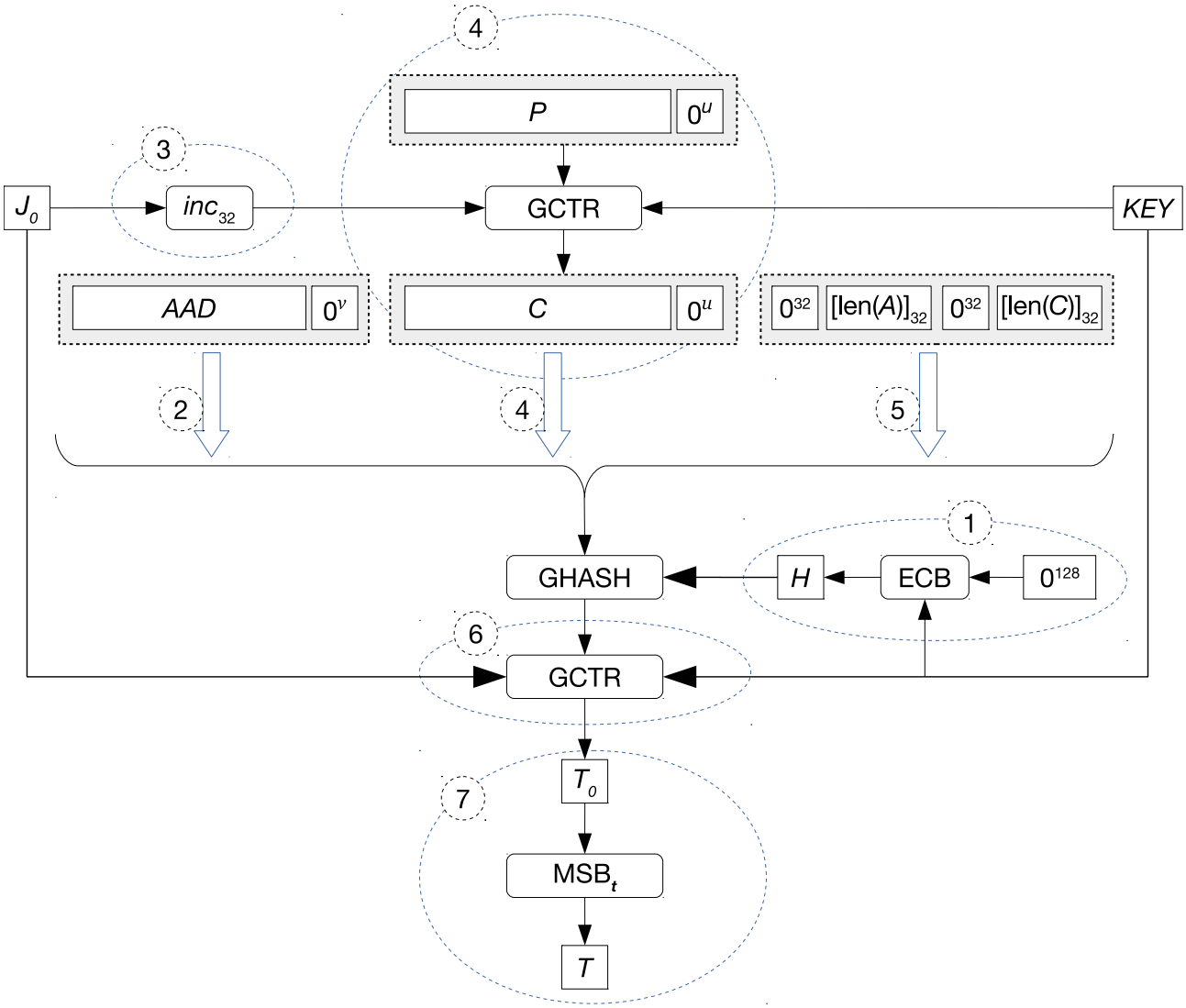


Figure 27.7-1. GCM Encryption Process

GCM encryption is implemented as follows:

1. Hardware executes the ECB Algorithm to obtain the Hash subkey H, which is needed in the Hash computation.
2. Hardware executes the GHASH Algorithm to perform Hash computation with the padded AAD.

3. Hardware gets ready for CTR encryption by obtaining the result of applying Standard Incrementing Function INC_{32} to J_0.
4. Hardware executes the GCTR Algorithm to encrypt the padded plaintext P, then executes the GHASH Algorithm to perform Hash computation on the padded ciphertext C.
5. Hardware executes the GHASH Algorithm to perform Hash computation on AAD Blocks, obtaining a 128-bit Hash result.
6. Hardware executes the GCTR Algorithm to encrypt J_0, obtaining T_0.
7. Software obtains the result T_0 from hardware, and executes MSB_t Algorithm to obtain the final authentication tag T.

The only difference between GCM decryption and GCM encryption lies in Step 4 in Figure 27.7-1. To be more specific, instead of executing GCTR Algorithm to encrypt the padded plaintext, the AES Accelerator executes the same algorithm to decrypt the padded ciphertext in GCM decryption. For details, please see NIST SP 800-38D.

27.7.1 Hash Subkey

During GCM operation, the Hash subkey H is a 128-bit value computed by hardware, which is demonstrated in Step 1 in Figure 27.7-1. Also you can find more information about Hash subkey at "Step 1. Let H = $\text{CIPH}_K(0^{128})$" in Chapter 7 GCM Specification of NIST SP 800-38D.

The Hash subkey H is stored in the AES_H_MEM memory. Just like other endianness, its most significant (i.e., left-most) byte Byte0 is stored at the lowest address in the memory while least significant (i.e., right-most) byte Byte15 at the highest address. For details, see Table 27.5-2.

27.7.2 J_0

J_0 is a 128-bit value computed by hardware, which is required during Step 3 and Step 6 in Figure 27.7-1. For details on the generation of J_0, please see Chapter 7 GCM Specification in NIST SP 800-38D.

J_0 is stored in the AES_J0_MEM memory. Just like other endianness, its most significant (i.e., left-most) byte Byte0 is stored at the lowest address in the memory while least significant (i.e., right-most) byte Byte15 at the highest address. For details, see Table 27.5-2.

27.7.3 Authentication Tag

Authentication Tag (Tag for short) is one of the key results of GCM computation, which is demonstrated in Step 7 of Figure 27.7-1. The value of the Tag is determined by its length t ($1 <= t <= 128$):

- When $t = 128$, the value of Tag equals to T_0, a 128-bit string that is stored in the AES_T0_MEM. Just like other endianness, its most significant (i.e., left-most) byte Byte0 is stored at the lowest address in the memory while least significant (i.e., right-most) byte Byte15 at the highest address. For details, see Table 27.5-2.
- When $1 <= t < 128$, the value of Tag equals to the t most significant (i.e., left-most) bits of T_0. In this case, Tag is represented as $\text{MSB}_t(T_0)$, which returns the t most significant bits of T_0. For example, MSB_4(111011010) = 1110 and MSB_5(11010011010) = 11010. For details on the $\text{MSB}_t(\,)$ function, please refer to Chapter 6 Mathematical Components of GCM in NIST SP 800-38D.

27.7.4 AAD Block Number

Register AES_AAD_BLOCK_NUM_REG stores the Block Number of Additional Authenticated Data (AAD). The length of this register equals to (**TEXT-PADDING**(AAD) length)/128. AES Accelerator only uses this register when working in the DMA-AES mode.

27.7.5 Number of Effective Bits of Incomplete Blocks

Register AES_REMAINDER_BIT_NUM_REG stores the Remainder Bit Number, which indicates the number of effective bits of incomplete blocks in plaintext/ciphertext. The value stored in this register equals to length(P)/128 or length(C)/128. AES Accelerator only uses this register when working in the DMA-AES mode.

Register AES_REMAINDER_BIT_NUM_REG does not affect the results of plaintext or ciphertext, but does impact the value of T_0, therefore the Tag value too.

The GCM Algorithm can be viewed as the combination of GCTR operation and GHASH operation, among which, the GCTR performs the encryption and decryption, while the GHASH sloves the Tag.

Note that the AES_REMAINDER_BIT_NUM_REG register is only effective for GCM encryption. To be more specific:

- For GCM encryption, the Hardware firstly computes C, then passes it in the form of **TEXT-PADDING**(C) as the input of GHASH operation. In this case, hardware determines how many trailing "0" should be added based on the content of AES_REMAINDER_BIT_NUM_REG.
- For GCM decrpytion, the padding is completed with the **TEXT-PADDING**(C) function. In this case, the AES_REMAINDER_BIT_NUM_REG register is not effective.

27.8 Pseudo-Round Anti-DPA Function

The pseudo-round function of ESP32-P4's AES can achieve higher anti-attack performance. When the pseudo-round function is enabled, AES will randomly add pseudo-rounds before and after the original operation rounds. In a pseudo-round, AES will use the pseudo-key, which is automatically generated by the hardware, to perform a round of fake operations. The operations in the pseudo-round do not affect the original AES operation results.

The configuration related to pseudo-round is as follows:

Mode of Pseudo-Round Function

Users can choose to enable pseudo-round function by configuring the AES_PSEUDO_EN register:

- 0: Not enable pseudo-round function;
- 1: Enable pseudo-round function.

Total number of Pseudo-Round

The total number of pseudo-rounds that will be randomly inserted into each AES operation round is controlled by:

- pseudo_base: Equal to the value of AES_PSEUDO_BASE, which configures the basic number of pseudo-rounds.

- pseudo_inc: Equal to the value of AES_PSEUDO_INC, which configures the random incremental number of pseudo-rounds.

The total number of pseudo-rounds will be randomly in the range

$$[pseudo_base, pseudo_base + (2^{pseudo_inc} - 1)]$$

Random Number Update Frequency

AES_PSEUDO_RNG_CNT configures the frequency of random key updates in the pseudo-round function. The higher the configured value, the higher the update frequency. This value is usually recommended to be set to maximum (7).

27.9 Interrupts

ESP32-P4's AES can generate the following interrupt signal(s) that will be sent to the Interrupt Matrix.

- AES_INTR

There is one internal interrupt source from AES that can generate the above interrupt signal(s). The interrupt source from AES are listed with its trigger condition and the resulted interrupt signal(s) in Table 27.9-1.

Table 27.9-1. AES's Internal Interrupt Sources

Internal Interrupt Source	Trigger Condition	Interrupt Signal
AES_DMA_DONE_INT	Completion of an AES DMA operation	AES_INTR

Note:
For definitions of *interrupt*, *interrupt signal*, *interrupt source*, and their correlations, please refer to Chapter 13 *Interrupt Matrix* > Section 13.2 *Interrupt Terminology in ESP32-P4*.

27.10 Memory Summary

The addresses in this section are relative to the AES accelerator base address provided in Table 9.3-2 in Chapter 9 *System and Memory*.

The abbreviations given in Column **Access** are explained in Section *Access Types for Registers*.

Name	Description	Size (byte)	Starting Address	Ending Address	Access
AES_IV_MEM	Memory IV	16 bytes	0x0050	0x005F	R/W
AES_H_MEM	Memory H	16 bytes	0x0060	0x006F	RO
AES_J0_MEM	Memory J0	16 bytes	0x0070	0x007F	R/W
AES_T0_MEM	Memory T0	16 bytes	0x0080	0x008F	RO

27.11 Register Summary

The addresses in this section are relative to the AES accelerator base address provided in Table 9.3-2 in Chapter 9 *System and Memory*.

The abbreviations given in Column **Access** are explained in Section *Access Types for Registers*.

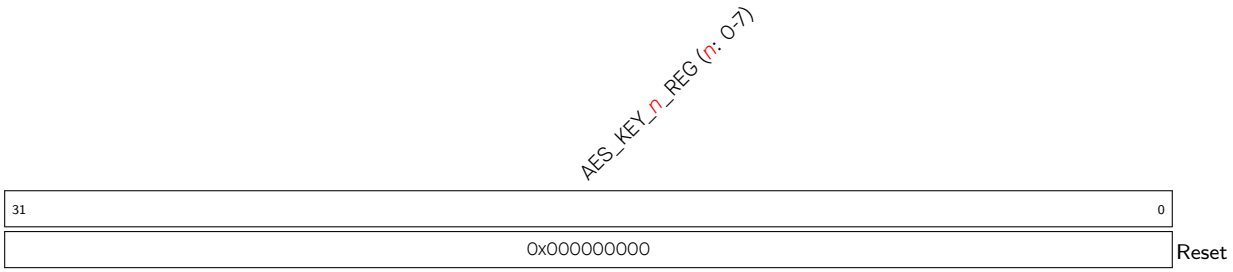
Name	Description	Address	Access
Key Registers			
AES_KEY_0_REG	AES key data register 0	0x0000	R/W
AES_KEY_1_REG	AES key data register 1	0x0004	R/W
AES_KEY_2_REG	AES key data register 2	0x0008	R/W
AES_KEY_3_REG	AES key data register 3	0x000C	R/W
AES_KEY_4_REG	AES key data register 4	0x0010	R/W
AES_KEY_5_REG	AES key data register 5	0x0014	R/W
AES_KEY_6_REG	AES key data register 6	0x0018	R/W
AES_KEY_7_REG	AES key data register 7	0x001C	R/W
TEXT_IN Registers			
AES_TEXT_IN_0_REG	Source text data register 0	0x0020	R/W
AES_TEXT_IN_1_REG	Source text data register 1	0x0024	R/W
AES_TEXT_IN_2_REG	Source text data register 2	0x0028	R/W
AES_TEXT_IN_3_REG	Source text data register 3	0x002C	R/W
TEXT_OUT Registers			
AES_TEXT_OUT_0_REG	Result text data register 0	0x0030	RO
AES_TEXT_OUT_1_REG	Result text data register 1	0x0034	RO
AES_TEXT_OUT_2_REG	Result text data register 2	0x0038	RO
AES_TEXT_OUT_3_REG	Result text data register 3	0x003C	RO
Control or Configuration Registers			
AES_MODE_REG	Defines key length and encryption/decryption	0x0040	R/W
AES_DMA_ENABLE_REG	Selects the working mode of the AES accelerator	0x0090	R/W
AES_BLOCK_MODE_REG	Defines the block cipher mode	0x0094	R/W
AES_BLOCK_NUM_REG	Block number configuration register	0x0098	R/W
AES_INC_SEL_REG	Standard incrementing function register	0x009C	R/W
AES_AAD_BLOCK_NUM_REG	AAD block number configuration register	0x00A0	R/W
AES_REMAINDER_BIT_NUM_REG	Remainder bit number of plaintext/ciphertext	0x00A4	R/W
AES_TRIGGER_REG	Operation start controlling register	0x0048	WT
AES_CONTINUE_OP_REG	Operation continue controlling register	0x00A8	WO
AES_DMA_EXIT_REG	Operation exit controlling register	0x00B8	WO
AES_PSEUDO_REG	Pseudo-round function configuration register	0x00D0	R/W
Status Register			
AES_STATE_REG	Operation status register	0x004C	RO
Interrupt Registers			
AES_INT_CLR_REG	DMA-AES interrupt clear register	0x00AC	WT
AES_INT_ENA_REG	DMA-AES interrupt enable register	0x00B0	R/W

27.12 Registers

The addresses in this section are relative to the AES accelerator base address provided in Table 9.3-2 in Chapter 9 *System and Memory*.

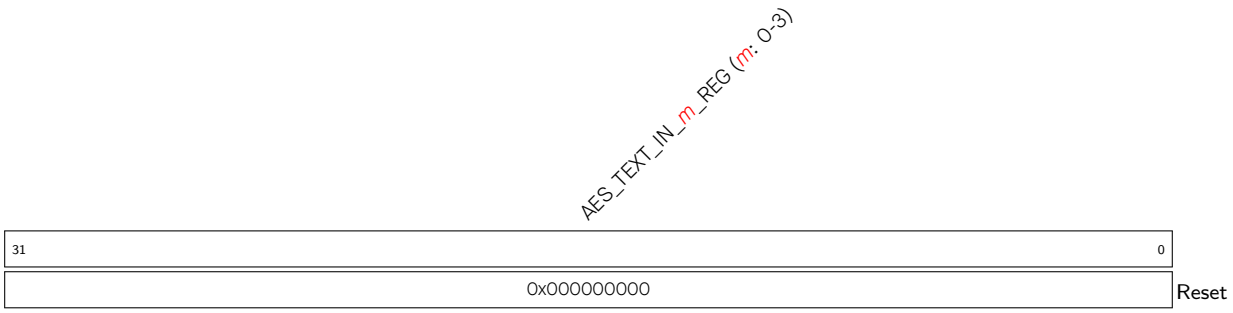
For how to program reserved fields, please refer to Section IX .

Register 27.1. AES_KEY_*n*_REG (*n*: 0-7) (0x0000+4n*)**



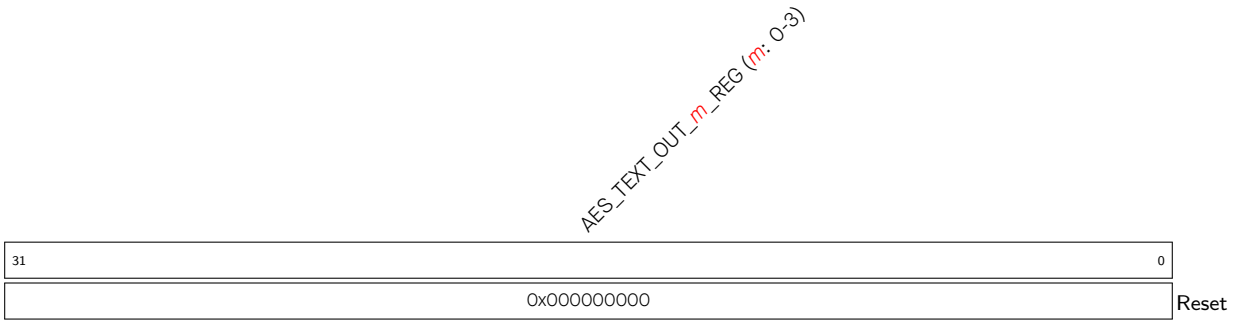
AES_KEY_*n*_REG (*n*: 0-7) Represents AES key data. (R/W)

Register 27.2. AES_TEXT_IN_*m*_REG (*m*: 0-3) (0x0020+4m*)**



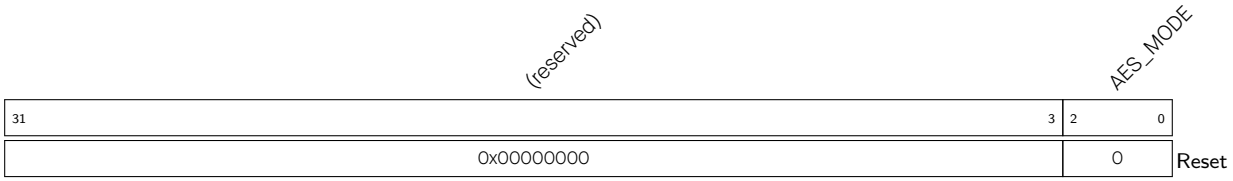
AES_TEXT_IN_*m*_REG (*m*: 0-3) Represents the source text data when the AES accelerator operates in the Typical AES working mode. (R/W)

Register 27.3. AES_TEXT_OUT_*m*_REG (*m*: 0-3) (0x0030+4m*)**



AES_TEXT_OUT_*m*_REG (*m*: 0-3) Represents the result text data when the AES accelerator operates in the Typical AES working mode. (RO)

Register 27.4. AES_MODE_REG (0x0040)



AES_MODE Configures the key length and encryption/decryption of the AES accelerator.

0: AES-128 encryption

1: Reserved

2: AES-256 encryption

3: Reserved

4: AES-128 decryption

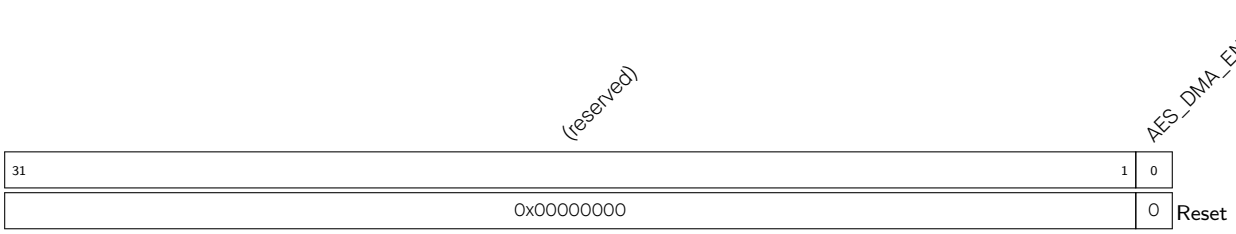
5: Reserved

6: AES-256 decryption

7: Reserved

(R/W)

Register 27.5. AES_DMA_ENABLE_REG (0x0090)



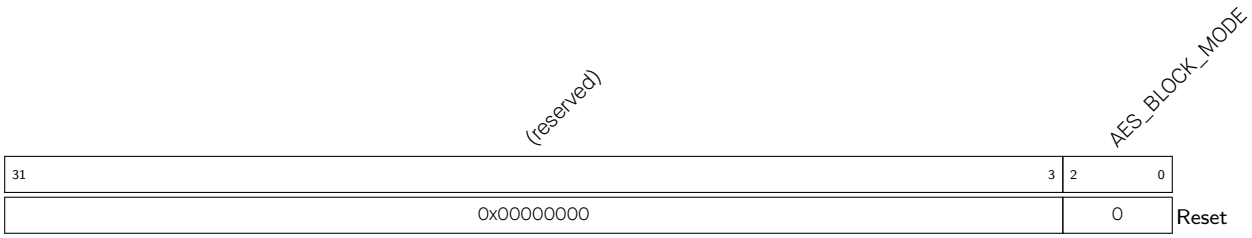
AES_DMA_ENABLE Configures the working mode of the AES accelerator.

0: Typical AES

1: DMA-AES

(R/W)

Register 27.6. AES_BLOCK_MODE_REG (0x0094)



AES_BLOCK_MODE Configures the block cipher mode of the AES accelerator operating under the DMA-AES working mode.

0: ECB (Electronic Code Block)

1: CBC (Cipher Block Chaining)

2: OFB (Output FeedBack)

3: CTR (Counter)

4: CFB8 (8-bit Cipher FeedBack)

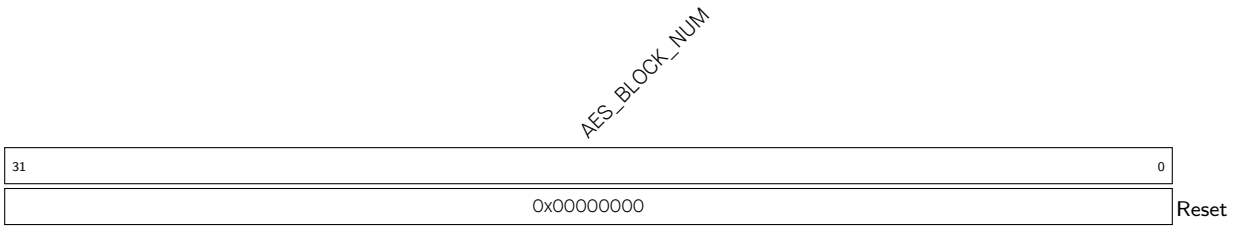
5: CFB128 (128-bit Cipher FeedBack)

6: GCM (Galois/Counter Mode)

7: Reserved

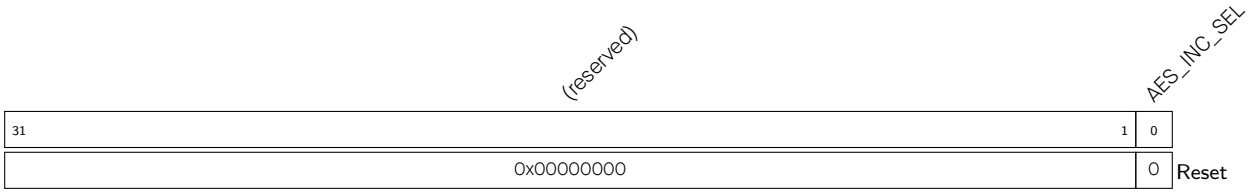
(R/W)

Register 27.7. AES_BLOCK_NUM_REG (0x0098)



AES_BLOCK_NUM Represents the Block Number of plaintext or ciphertext when the AES accelerator operates under the DMA-AES working mode. For details, see Section 27.6.4. (R/W)

Register 27.8. AES_INC_SEL_REG (0x009C)



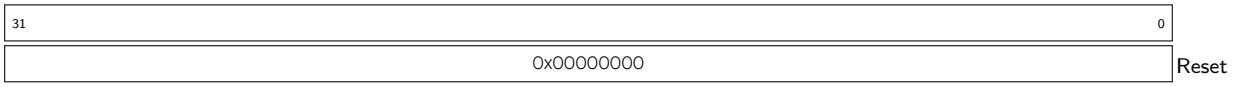
AES_INC_SEL Configures the Standard Incrementing Function for CTR block operation.

0: INC_{32}

1: INC_{128}

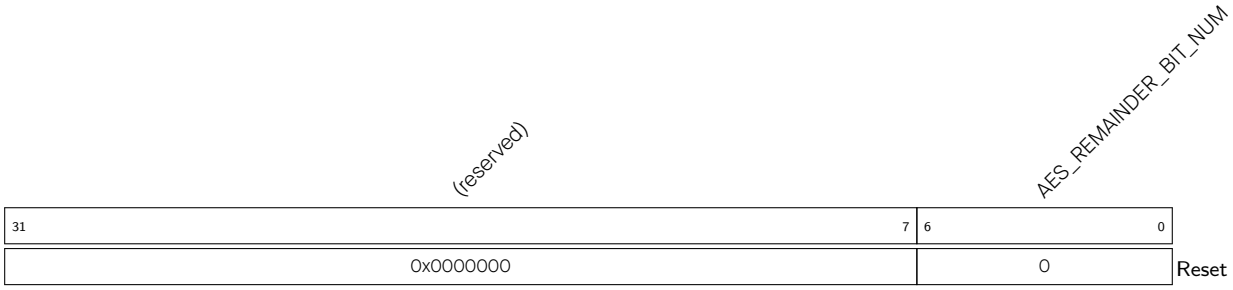
(R/W)

Register 27.9. AES_AAD_BLOCK_NUM_REG (0x00A0)



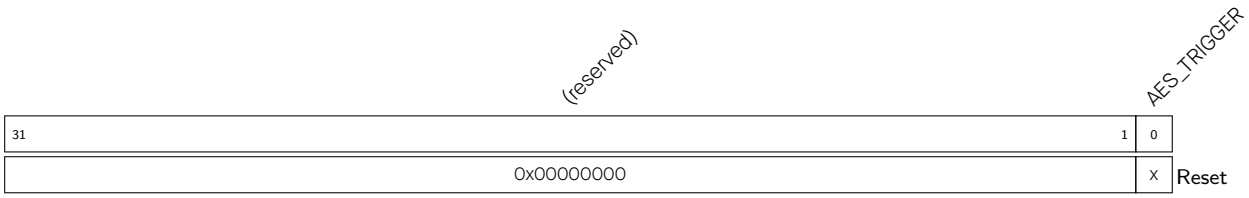
AES_AAD_BLOCK_NUM Stores the ADD block number for the GCM operation. For details, see Section 27.7.4. (R/W)

Register 27.10. AES_REMAINDER_BIT_NUM_REG (0x00A4)



AES_REMAINDER_BIT_NUM Stores the Remainder Bit Number for the GCM operation. For details, see Section 27.7.5. (R/W)

Register 27.11. AES_TRIGGER_REG (0x0048)



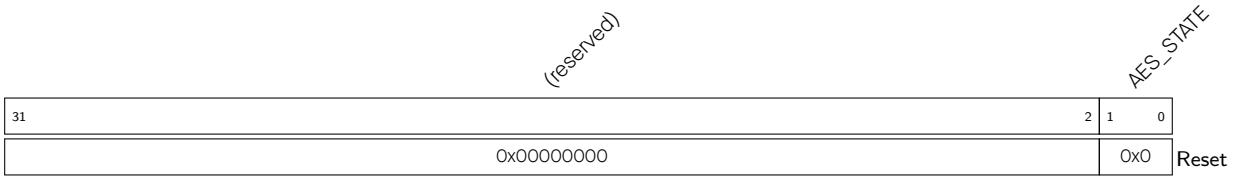
AES_TRIGGER Configures whether to start AES operation.

0: No effect

1: Start

(WT)

Register 27.12. AES_STATE_REG (0x004C)



AES_STATE Represents the working status of the AES accelerator.

In Typical AES working mode:

0: IDLE

1: WORK

2: No effect

3: No effect

In DMA-AES working mode:

0: IDLE

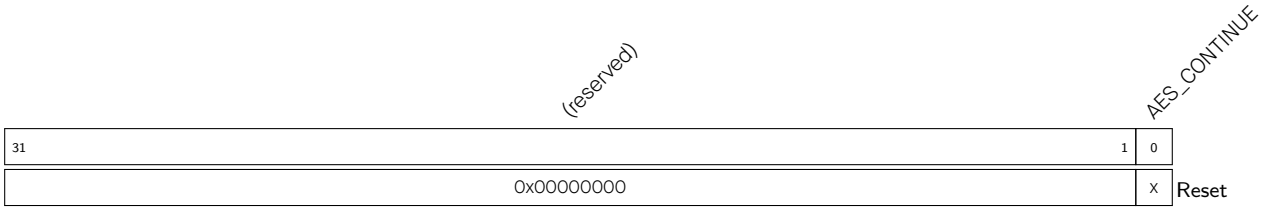
1: WORK

2: DONE

3: No effect

(RO)

Register 27.13. AES_CONTINUE_OP_REG (0x00A8)



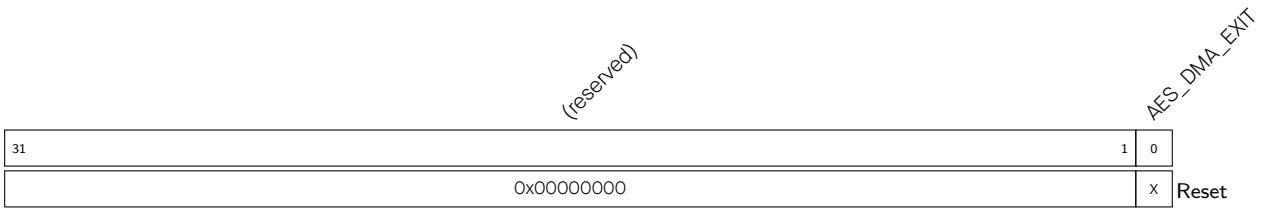
AES_CONTINUE_OP Configures whether to continue AES operation.

0: No effect

1: Continue

(WO)

Register 27.14. AES_DMA_EXIT_REG (0x00B8)



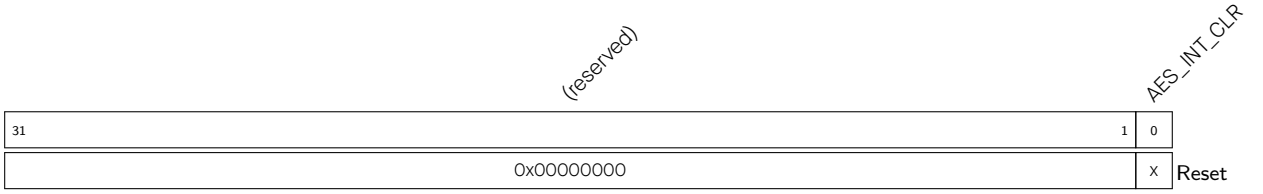
AES_DMA_EXIT Configures whether to exit AES operation.

0: No effect

1: Exit

Only valid for DMA-AES operation. (WO)

Register 27.15. AES_INT_CLR_REG (0x00AC)



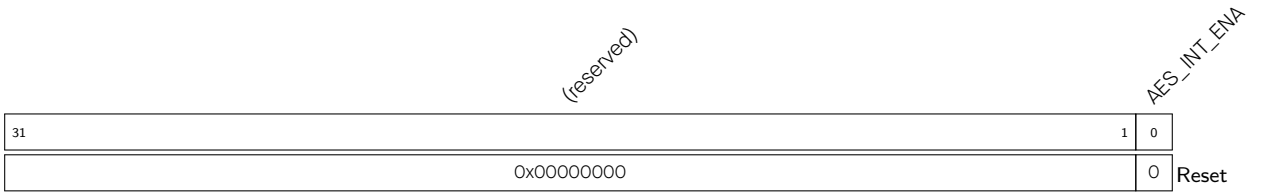
AES_INT_CLR Configures whether to clear AES interrupt.

0: No effect

1: Clear

(WT)

Register 27.16. AES_INT_ENA_REG (0x00B0)



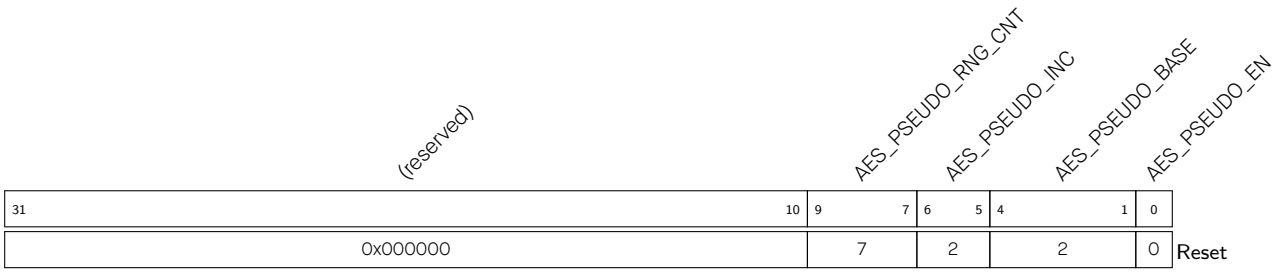
AES_INT_ENA Configures whether to enable AES interrupt.

0: Disable

1: Enable

(R/W)

Register 27.17. AES_PSEUDO_REG (0x00D0)



AES_PSEUDO_EN Configures whether to enable the pseudo-round function of AES.

0: Disable

1: Enable

(R/W)

AES_PSEUDO_BASE Configures the basic number of pseudo-rounds. (R/W)

AES_PSEUDO_INC Configures the random incremental number of pseudo-rounds. (R/W)

AES_PSEUDO_RNG_CNT Configures the frequency of pseudo-key updates in the pseudo-round function. (R/W)

Chapter 28

ECC Accelerator (ECC)

28.1 Overview

Elliptic Curve Cryptography (ECC) is an approach to public-key cryptography based on the algebraic structure of elliptic curves. ECC uses smaller keys compared to RSA cryptography while providing equivalent security.

ESP32-P4's ECC accelerator can complete various calculations based on different elliptic curves, thus accelerating the ECC algorithm and ECC-derived algorithms such as ECDSA.

28.2 Feature List

ESP32-P4's ECC accelerator has the following features:

- Three different elliptic curves, namely, P-192, P-256, and P-384 defined in FIPS 186-5
- 11 working modes
- Enhanced anti-attack performance
- Interrupt upon completion of calculation

28.3 ECC Basics

To better illustrate the functionality of the ECC accelerator, the basic knowledge and terminology used in this chapter are introduced in this section.

28.3.1 Elliptic Curve and Points on the Curves

The ECC algorithm is based on elliptic curves over prime fields, which can be represented as:

$$y^2 = x^3 + ax + b \bmod p$$

where,

- p is a prime number,
- a and b are two non-negative integers smaller than p,
- and (x, y) is a point on the curve satisfying the representation.

28.3.2 Affine Coordinates and Jacobian Coordinates

An elliptic curve can be represented as below:

- In affine coordinates:

$$y^2 = x^3 + ax + b \bmod p$$

- In Jacobian coordinates:

$$Y^2 = X^3 + aXZ^4 + bZ^6 \bmod p$$

To convert affine coordinates (x, y) to/from Jacobian coordinates (X, Y, Z):

- From Jacobian to Affine coordinates:

$$x = X/Z^2 \bmod p$$

$$y = Y/Z^3 \bmod p$$

- From Affine to Jacobian coordinates:

$$X = x$$

$$Y = y$$

$$Z = 1$$

28.3.3 Memory Blocks

ECC's memory blocks store the input and output data of the ECC operation.

Table 28.3-1. ECC Accelerator Memory Blocks

Memory Block	Size (byte)	Starting Address*	Ending Address*	Access
ECC_MULT_Mem_k	32	0x100	0x11F	R/W
ECC_MULT_Mem_Px	32	0x120	0x13F	R/W
ECC_MULT_Mem_Py	32	0x140	0x15F	R/W
ECC_MULT_Mem_Qx	32	0x160	0x17F	R/W
ECC_MULT_Mem_Qy	32	0x180	0x19F	R/W
ECC_MULT_Mem_Qz	32	0x1A0	0x1BF	R/W

* Address offset related to the ECC accelerator base address is provided in Table 9.3-2 in Chapter 9 *System and Memory*.

28.3.4 Data and Data Block

ESP32-P4's ECC can operate on 192-bit, 256-bit, or 384-bit data, depending on the elliptic curves.

For example, the 256-bit data $D[255:0]$ can be divided into eight 32-bit data blocks $D[n][31:0](n = 0, 1, \cdots, 7)$. Data blocks with smaller indexes correspond to lower binary bits. To be specific:

$$D[255:0] = D[7][31:0], D[6][31:0], D[5][31:0], D[4][31:0], D[3][31:0], D[2][31:0], D[1][31:0], D[0][31:0]$$

28.3.5 Writing Data

Writing data means writing data to an ECC memory block and using this data as the input to the ECC algorithm. To be specific, writing data to an ECC memory block means writing $D[n][31:0](n = 0, 1, \cdots, 7)$ to the "starting address of this ECC memory block + $4 \times n$" sequentially:

- write $D[0]$ to "starting address"
- write $D[1]$ to "starting address + 4"
- $\cdots$
- write $D[7]$ to "starting address + 28"

Note:
When storing data, write only the data block of the required length. Do not append a 0 to the most significant bit. For example, for 192-bit data, store only the 192 bits without appending 0.

28.3.6 Reading Data

Read data means reading data from the starting address of an ECC memory block and using this data as the output from the ECC algorithm. To be specific, read data from an ECC memory block means reading $D[n][31:0](n = 0, 1, \cdots, 7)$ from the "starting address of this ECC memory block + $4 \times n$" successively:

- read $D[0]$ from "starting address"
- read $D[1]$ from "starting address + 4"
- $\cdots$
- read $D[7]$ from "starting address + 28"

Note:
When reading data, only the data block of the required length needs to be read. For example, to read 192-bit data, simply read the lower 192 bits (i.e., 6 data blocks).

28.3.7 Standard Calculation and Jacobian Calculation

ESP32-P4's ECC performs Affine Point Calculation (including Affine Point Verification, Affine Point Add, and Affine Point Multiplication) using the affine coordinates and Jacobian Calculation (including Jacobian Point Verification, Jacobian Point Add, and Jacobian Point Multiplication) using the Jacobian coordinates.

28.4 Function Description

28.4.1 Curve Mode

ESP32-P4's ECC supports acceleration based on three curve modes, each corresponding to an elliptic curve. By configuring the ECC_MULT_CURVE_MODE field, users can select the desired key size. For details, see Table 28.4-1 below.

Table 28.4-1. ECC Accelerator Curve Mode Selection

ECC_MULT_CURVE_MODE	Elliptic Curves*
0	FIPS P-192
1	FIPS P-256
2	FIPS P-384

* See definition of FIPS P-192, P-256, and P-384 defined in FIPS 186-5.

28.4.2 Working Modes

ESP32-P4's ECC accelerator supports 11 working modes based on two elliptic curves described in the above section. By configuring the ECC_MULT_WORK_MODE field, users can select the desired working mode. For details, see Table 28.4-2.

Table 28.4-2. Working Modes of ECC Accelerator

ECC_MULT_WORK_MODE	Working Modes
0	Affine Point Multi
1	**Reserved**
2	Affine Point Verif
3	Affine Point Verif + Multi
4	Jacobian Point Multi
5	Point Add
6	Jacobian Point Verif
7	Affine Point Verif + Jacobian Point Multi
8	Mod Add
9	Mod Sub
10	Mod Multi
11	Mod Div

Note:
Note that the calculation of Jacobian Point Multi mode is about 10% faster than that of the Affine Point Multi mode.

Detailed descriptions about different working modes are provided in the following sections.

28.4.2.1 Affine Point Multiplication (Affine Point Multi)

Affine Point Multiplication can be represented as:

$$Q = (Q_x, Q_y) = (J_x, J_y, J_z) = k \cdot (P_x, P_y)$$

where,

- (Q_x, Q_y) is the affine expression of point Q.
- (J_x, J_y, J_z) is the Jacobian expression of point Q.

- Input: P_x, P_y, and k are stored in ECC_MULT_Mem_Px, ECC_MULT_Mem_Py, and ECC_MULT_Mem_k respectively.
- Output: Q_x and Q_y are stored in ECC_MULT_Mem_Px and ECC_MULT_Mem_Py respectively.

28.4.2.2 Affine Point Verification (Affine Point Verif)

Affine Point Verification can be used to verify if a point (P_x, P_y) is on a selected elliptic curve.

- Input: P_x and P_y are stored in ECC_MULT_Mem_Px and ECC_MULT_Mem_Py respectively.
- Output: The verification result is stored in the ECC_MULT_VERIFICATION_RESULT bit.

28.4.2.3 Affine Point Verification + Affine Point Multiplication (Affine Point Verif + Multi)

In this mode, ECC first verifies if point (P_x, P_y) is on the selected elliptic curve. If so, the following multiplication is performed:

$$Q = (Q_x, Q_y) = (J_x, J_y, J_z) = k \cdot (P_x, P_y)$$

where,

- (Q_x, Q_y) is the affine expression of point Q.
- (J_x, J_y, J_z) is the Jacobian expression of point Q.
- Input: P_x, P_y, and k are stored in ECC_MULT_Mem_Px, ECC_MULT_Mem_Py, and ECC_MULT_Mem_k respectively.
- Output:
  - The verification result is stored in the ECC_MULT_VERIFICATION_RESULT bit.
  - Q_x and Q_y are stored in ECC_MULT_Mem_Px and ECC_MULT_Mem_Py respectively.
  - J_x, J_y, and J_z are stored in ECC_MULT_Mem_Qx, ECC_MULT_Mem_Qy, and ECC_MULT_Mem_Qz.

28.4.2.4 Jacobian Point Multiplication (Jacobian Point Multi)

Jacobian Point Multiplication can be represented as:

$$Q = (Q_x, Q_y, Q_z) = k \cdot (P_x, P_y, 1)$$

where,

- (Q_x, Q_y, Q_z) is the Jacobian expression of point Q.
- 1 in the point's Jacobian coordinates is automatically completed by hardware.
- Input: P_x, P_y, and k are stored in ECC_MULT_Mem_Px, ECC_MULT_Mem_Py, and ECC_MULT_Mem_k respectively.
- Output: Q_x, Q_y, and Q_z are stored in ECC_MULT_Mem_Qx, ECC_MULT_Mem_Qy, and ECC_MULT_Mem_Qz respectively.

28.4.2.5 Point Addition (Point Add)

Point Addition can be represented as:

$$R = (R_x, R_y) = (J_x, J_y, J_z) = (P_x, P_y, 1) + (Q_x, Q_y, Q_z)$$

where,

- (R_x, R_y) is the affine expression of point R.
- (J_x, J_y, J_z) is the Jacobian expression of point R.
- 1 in the point's Jacobian coordinates is automatically completed by hardware.
- Input:
  - P_x and P_y are stored in ECC_MULT_Mem_Px and ECC_MULT_Mem_Py.
  - Q_x, Q_y, and Q_z are stored in ECC_MULT_Mem_Qx, ECC_MULT_Mem_Qy, and ECC_MULT_Mem_Qz.
- Output:
  - R_x and R_y are stored in ECC_MULT_Mem_Px and ECC_MULT_Mem_Py.
  - J_x, J_y and J_z are stored in ECC_MULT_Mem_Qx, ECC_MULT_Mem_Qy, and ECC_MULT_Mem_Qz.

28.4.2.6 Jacobian Point Verification (Jacobian Point Verif)

Jacobian Point Verification can be used to verify if point (Q_x, Q_y, Q_z) is on a selected elliptic curve.

- (Q_x, Q_y, Q_z) is the Jacobian expression of point Q.
- Input: Q_x, Q_y, and Q_z are stored in ECC_MULT_Mem_Qx, ECC_MULT_Mem_Qy, and ECC_MULT_Mem_Qz respectively.
- Output: The verification result is stored in the ECC_MULT_VERIFICATION_RESULT bit.

28.4.2.7 Affine Point Verification + Jacobian Point Multiplication (Affine Point Verif + Jacobian Point Multi)

In this mode, ECC first verifies if point (P_x, P_y) is on the selected elliptic curve. If so, the following multiplication is performed:

$$Q = (Q_x, Q_y, Q_z) = k \cdot (P_x, P_y, 1)$$

where,

- (Q_x, Q_y, Q_z) is the Jacobian expression of point Q.
- 1 in the point's Jacobian coordinates is automatically completed by hardware.
- Input: P_x, P_y, and k are stored in ECC_MULT_Mem_Px, ECC_MULT_Mem_Py, and ECC_MULT_Mem_k.
- Output:
  - The verification result is stored in the ECC_MULT_VERIFICATION_RESULT bit.
  - Q_x, Q_y, and Q_z are stored in ECC_MULT_Mem_Qx, ECC_MULT_Mem_Qy, and ECC_MULT_Mem_Qz.

28.4.2.8 Mod Addition (Mod Add)

Mod Addition can be represented as:

$$R = A + B \bmod N$$

where,

- Input:
  - A and B are stored in ECC_MULT_Mem_Px and ECC_MULT_Mem_Py.
  - The value of N is related to the register fields below:
    * ECC_MULT_CURVE_MODE to select the related curve.
    * ECC_MULT_MOD_BASE to choose using mod base or order of the base point.
- Output: R is stored in ECC_MULT_Mem_Px.

28.4.2.9 Mod Subtraction (Mod Sub)

Mod Subtraction can be represented as:

$$R = A - B \bmod N$$

where,

- Input:
  - A and B are stored in ECC_MULT_Mem_Px and ECC_MULT_Mem_Py.
  - The value of N is related to the register fields below:
    * ECC_MULT_CURVE_MODE to select the related curve.
    * ECC_MULT_MOD_BASE to choose using mod base or order of the base point.
- Output: R is stored in ECC_MULT_Mem_Px.

28.4.2.10 Mod Multiplication (Mod Multi)

Mod Multiplication can be represented as:

$$R = A \cdot B \bmod N$$

where,

- Input:
  - A and B are stored in ECC_MULT_Mem_Px and ECC_MULT_Mem_Py.
  - The value of N is related to the register fields below:
    * ECC_MULT_CURVE_MODE to select the related curve.
    * ECC_MULT_MOD_BASE to choose using mod base or order of the base point.
- Output: R is stored in ECC_MULT_Mem_Py.

28.4.2.11 Mod Division (Mod Div)

Mod Division can be represented as:

$$R = A \cdot B^{-1} \bmod N$$

where,

- Input:
  - A and B are stored in ECC_MULT_Mem_Px and ECC_MULT_Mem_Py.
  - The value of N is related to the register fields below:
    * ECC_MULT_CURVE_MODE to select the related curve.
    * ECC_MULT_MOD_BASE to choose using mod base or order of the base point.
- Output: R is stored in ECC_MULT_Mem_Py.

28.4.3 Enhancing Anti-Attack Performance

When performing different types of point multiplication calculations, i.e., mode 0, 3, 4, and 7, ESP32-P4's ECC accelerator offers an additional option to further enhance its anti-attack performance by ensuring that all point multiplication calculations take the same maximum runtime.

To enable the enhanced anti-attack performance, set ECC_MULT_SECURITY_MODE to 1. If this setting is enabled, each time the ECC accelerator perform a point multiplication calculation:

- The execution latency is constant for a given operation.
- Power consumption variations are minimized.

28.5 Clock and Reset

ESP32-P4's ECC only has one clock module (CRYPTO_ECC_CLK) and one reset module (CRYPTO_ECC_RST).

ECC's clock and reset is handled by the the Power/Clock/Reset (PCR) module (see Chapter 11 *Reset and Clock* for more information). Users should enable the ECC clock by setting HP_SYS_CLKRST_CRYPTO_ECC_CLK_EN and release the ECC reset by clearing HP_SYS_CLKRST_RST_EN_ECC before starting the ECC accelerator. Besides, due to resource reuse between cryptography accelerator modules, users also need to additionally clear the HP_SYS_CLKRST_RST_EN_ECDSA bit.

28.6 Interrupts

ESP32-P4's ECC accelerator can generate the ECC_INTR interrupt signal that will be sent to the Interrupt Matrix.

ECC_INTR has only one interrupt source to generate the ECC_INTR interrupt signal, i.e., ECC_MULT_CALC_DONE_INT, which is triggered on the completion of an ECC calculation.

Each interrupt source can be configured by a common set of registers that are described in Section *Interrupt Configuration Registers*. The ECC_MULT_CALC_DONE_INT interrupt source is configured by the following registers (refer to Section 28.8 *Register Summary* for more information):

- ECC_MULT_CALC_DONE_INT_RAW: Stores the raw interrupt status of ECC_MULT_CALC_DONE_INT.
- ECC_MULT_CALC_DONE_INT_ST: Indicates the status of the ECC_MULT_CALC_DONE_INT interrupt. This bit is generated by enabling or disabling the ECC_MULT_CALC_DONE_INT_RAW bit via ECC_MULT_CALC_DONE_INT_ENA.
- ECC_MULT_CALC_DONE_INT_ENA: Enables or disables the ECC_MULT_CALC_DONE_INT interrupt.
- ECC_MULT_CALC_DONE_INT_CLR: Set this bit to clear the ECC_MULT_CALC_DONE_INT interrupt status. By setting this bit to 1, the ECC_MULT_CALC_DONE_INT_RAW and ECC_MULT_CALC_DONE_INT_ST bits will be cleared.

Note:
For definitions of *interrupt*, *interrupt signal*, *interrupt source*, and their correlations, please refer to Chapter 13 *Interrupt Matrix* > Section 13.2 *Interrupt Terminology in ESP32-P4*.

28.7 Programming Procedures

The programming procedure for configuring ECC is described below:

1. Configure the ECC clock and reset. Refer to Section 28.5 for detailed information.
2. Select the key size and working mode as described in Section 28.4.
3. Enable the ECC_MULT_CALC_DONE_INT interrupt as described in Section 28.6.
4. Set the ECC_MULT_START field to start ECC calculation.
5. Wait for the ECC_MULT_CALC_DONE_INT interrupt, which indicates the completion of the ECC calculation.
6. Check the result as described in Section 28.4.

28.8 Register Summary

The addresses in this section are relative to ECC Accelerator base address provided in Table 9.3-2 in Chapter 9 *System and Memory*.

The abbreviations given in Column **Access** are explained in Section *Access Types for Registers*.

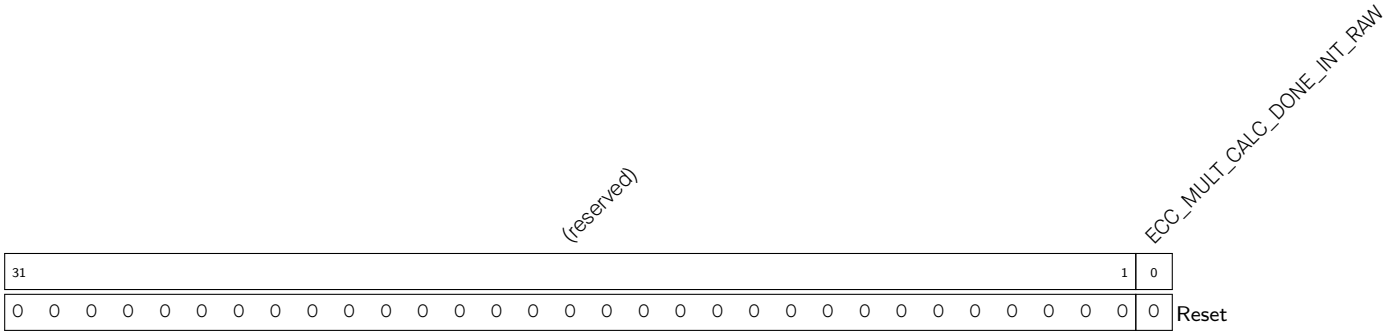
Name	Description	Address	Access
Interrupt Registers			
ECC_MULT_INT_RAW_REG	ECC raw interrupt status register	0x000C	R/SS/WTC
ECC_MULT_INT_ST_REG	ECC masked interrupt status register	0x0010	RO
ECC_MULT_INT_ENA_REG	ECC interrupt enable register	0x0014	R/W
ECC_MULT_INT_CLR_REG	ECC interrupt clear register	0x0018	WT
Configuration Register			
ECC_MULT_CONF_REG	ECC configuration register	0x001C	varies
Version Register			
ECC_MULT_DATE_REG	Version control register	0x00FC	R/W

28.9 Registers

The addresses in this section are relative to ECC Accelerator base address provided in Table 9.3-2 in Chapter 9 *System and Memory*.

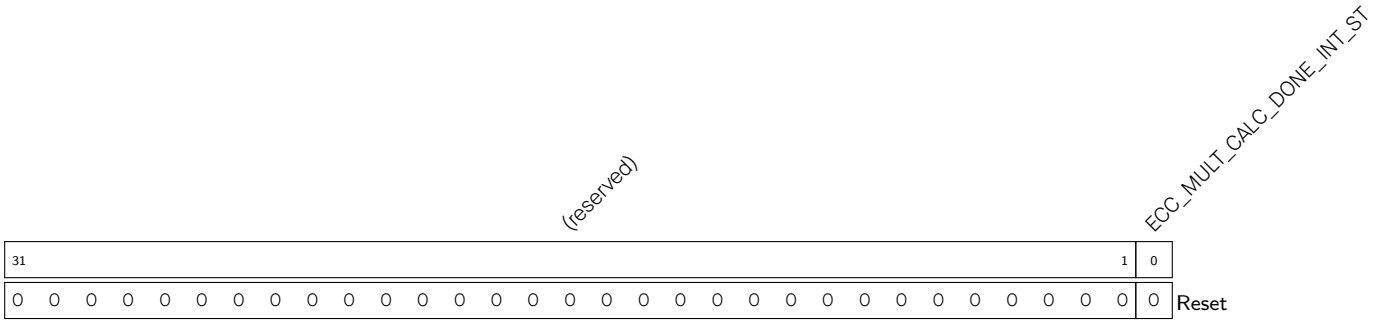
For how to program reserved fields, please refer to Section *Programming Reserved Register Field*.

Register 28.1. ECC_MULT_INT_RAW_REG (0x000C)



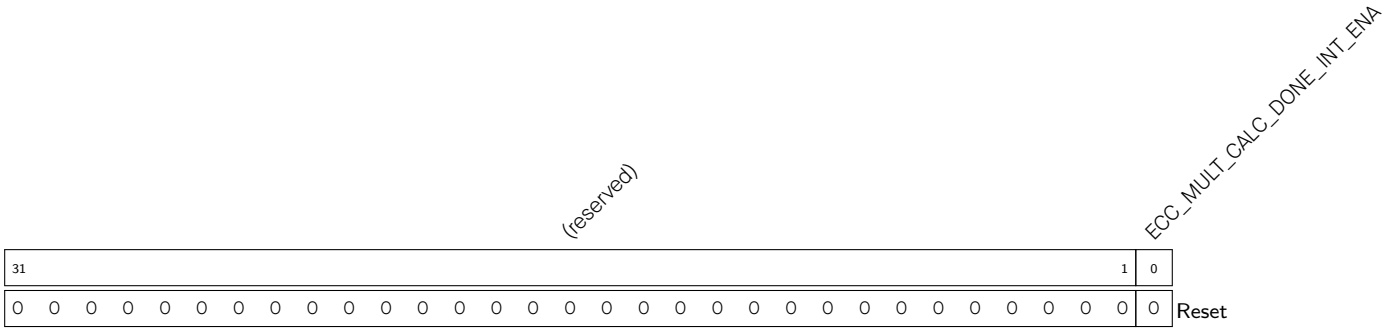
ECC_MULT_CALC_DONE_INT_RAW The raw interrupt status of the ECC_MULT_CALC_DONE_INT interrupt. (R/SS/WTC)

Register 28.2. ECC_MULT_INT_ST_REG (0x0010)



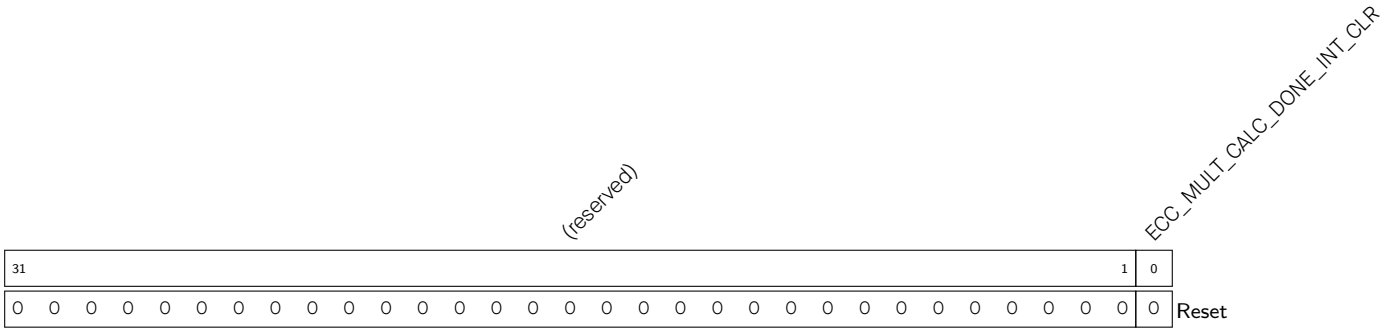
ECC_MULT_CALC_DONE_INT_ST The masked interrupt status of the ECC_MULT_CALC_DONE_INT interrupt. (RO)

Register 28.3. ECC_MULT_INT_ENA_REG (0x0014)



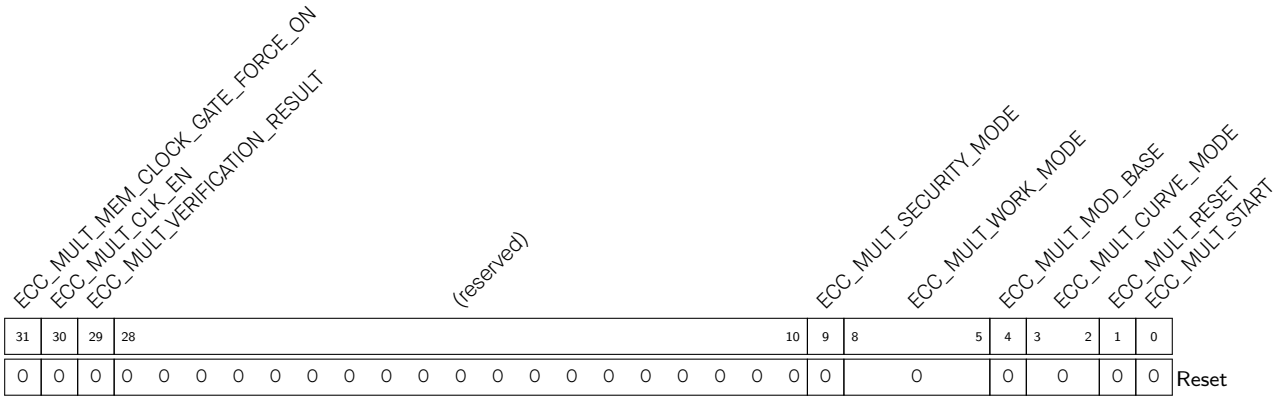
ECC_MULT_CALC_DONE_INT_ENA Write 1 to enable the ECC_MULT_CALC_DONE_INT interrupt. (R/W)

Register 28.4. ECC_MULT_INT_CLR_REG (0x0018)



ECC_MULT_CALC_DONE_INT_CLR Write 1 to clear the ECC_MULT_CALC_DONE_INT interrupt. (WT)

Register 28.5. ECC_MULT_CONF_REG (0x001C)



ECC_MULT_START Configures whether to start the calculation of the ECC accelerator. This bit will be self-cleared when the calculation is done.

0: No effect

1: Start the calculation of the ECC accelerator

(R/W/SC)

ECC_MULT_RESET Configures whether to reset the ECC accelerator.

0: No effect

1: Reset

(WT)

ECC_MULT_CURVE_MODE Configures the curve mode of the ECC accelerator.

0: P-192

1: P-256

2: P-384

3: Invalid

(R/W)

ECC_MULT_MOD_BASE Configures whether to choose using the mod base or order of base point in the mod operation. Only valid in working modes 8-11.

0: n (order of base point)

1: p (mod base of curve)

(R/W)

ECC_MULT_WORK_MODE Configures the working mode of the ECC accelerator.

0: Affine Point Multi mode

1: Reserved

2: Affine Point Verif mode

3: Affine Point Verif + Multi mode

4: Jacobian Point Multi mode

5: Point Add mode

6: Jacobian Point Verif mode

7: Affine Point Verif + Jacobian Point Multi mode

8: Mod Add mode

9. Mod Sub mode

10: Mod Multi mode

11: Mod Div mode

(R/W)

Continued on the next page...

Register 28.5. ECC_MULT_CONF_REG (0x001C)

Continued from the previous page...

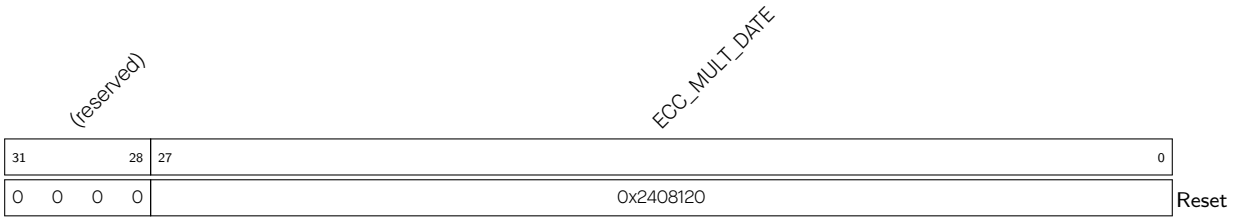
ECC_MULT_SECURITY_MODE Configures whether to enhance the anti-attack performance when calculating point multiplication. Only valid in working modes 0, 3, 4, and 7.
0: Disable
1: Enable
(R/W)

ECC_MULT_VERIFICATION_RESULT Represents the verification result of the ECC accelerator, valid only when the calculation is done.
0: Verification failed
1: Verification passed
(R/SS)

ECC_MULT_CLK_EN Configures whether to force on register clock gate.
0: No effect
1: Force on
(R/W)

ECC_MULT_MEM_CLOCK_GATE_FORCE_ON Configures whether to force on ECC memory clock gate.
0: No effect
1: Force on
(R/W)

Register 28.6. ECC_MULT_DATE_REG (0x00FC)



ECC_MULT_DATE Version control register. (R/W)

Chapter 29

HMAC Accelerator (HMAC)

The Hash-based Message Authentication Code (HMAC) module computes Message Authentication Codes (MACs) using hash algorithm SHA-256 and keys as described in RFC 2104. The 256-bit HMAC key is stored in an eFuse key block and can be set as read-protected, i.e., the key is not accessible from outside the HMAC accelerator.

29.1 Main Features

- Standard HMAC-SHA-256 algorithm
- HMAC-SHA-256 calculation based on key in eFuse,
  - whose result cannot be accessed by software in downstream mode for high security
  - whose result can be accessed by software in upstream mode
- Generates required keys for the RSA Digital Signature Peripheral (RSA_DS) in downstream mode
- Re-enables soft-disabled JTAG in downstream mode

29.2 Functional Description

The HMAC module operates in two modes: upstream mode and downstream mode. In upstream mode, users provide the HMAC message and read back the calculation result. In downstream mode, the HMAC module provides input to two possible other internal hardware modules: On the one hand, an HMAC can be used to enable JTAG after JTAG has been temporarily disabled before. On the other hand, an HMAC can be used as decryption key for Digital Signature parameters stored in flash memory for the RSA_DS peripheral. Furthermore, the calculations happen internally and automatically in downstream mode, so that confidentiality of any key and derived key material is ensured, given correct configuration.

Note:
After the reset signal being released, the HMAC module will check whether the RSA_DS_KEY exists in the eFuse. If the key exists, the HMAC module will enter downstream RSA_DS mode and finish the RSA_DS_KEY calculation automatically. This process is automatically completed by hardware and does not require software participation. When the downstream operation after reset is completed, the HMAC will automatically return to the idle state.

29.2.1 Upstream Mode

To calculate the HMAC value in upstream mode, users should perform the following steps:

1. Initialize the HMAC module and enter upstream mode.

2. Write the correctly padded message to the HMAC, one block at a time.
3. Read back the result from HMAC.

For details of this process, please see Section 29.2.5.

Note:
Common use cases for the upstream mode are challenge-response protocols supporting HMAC-SHA-256. Assume the two entities in the challenge-response protocol are A and B respectively, and the data message they expect to exchange is M. The general authentication process of this protocol is as follows:

- A calculates a unique random number M.
- A sends M to B.
- B calculates the HMAC (through M and KEY) and sends the result to A.
- A calculates the HMAC (through M and KEY) internally.
- A compares the two results. If the results are the same, then the identity of B is authenticated.

29.2.2 Downstream Mode - JTAG Enable Feature

JTAG debugging can be disabled by eFuse in a way which allows later re-enabling using the HMAC module. (For more details, please see Chapter 9.4 *eFuse Controller (EFUSE)*.) The HMAC module will expect the user to supply the HMAC result for one of the eFuse keys. The HMAC module will check whether the supplied HMAC matches the one calculated from the chosen key. If both HMACs are the same, JTAG will be enabled until the user calls the HMAC module to clear the results and consequently disable JTAG again.

To re-enable JTAG, users should perform the following steps:

1. Enable the HMAC module by initializing clock and reset signals of HMAC, and enter downstream JTAG enable mode by configuring HMAC_SET_PARA_PURPOSE_REG. Then, wait for the calculation to complete. Please see Section 29.2.5 for more details.
2. Write 1 to the HMAC_SOFT_JTAG_CTRL_REG register to enter JTAG re-enable mode.
3. Write the 256-bit HMAC value to register HMAC_WR_JTAG_REG. This value is obtained by performing a local HMAC calculation from the 32-byte 0x00 using SHA-256 and the key that has been written to the eFuse. It needs to be written 8 times and 32-bit each time in big-endian word order.
4. If the HMAC result calculated from the key in the eFuse matches the value that users wrote in step 3, then JTAG is re-enabled. Otherwise, JTAG remains disabled.
5. After writing 1 to HMAC_SET_INVALIDATE_JTAG_REG or resetting the chip, JTAG will be disabled. If users want to re-enable JTAG again, they need to repeat the above steps again.

29.2.3 Downstream Mode - RSA_DS Key Derivation Feature

The RSA Digital Signature Peripheral (RSA_DS) encrypts its parameters using the AES-CBC algorithm. The HMAC module is used as a Key Derivation Function (KDF) to derive the AES key to decrypt these parameters (parameter decryption key).

Before starting the RSA_DS peripheral, users need to obtain the parameter decryption key for the RSA_DS peripheral through HMAC calculation. For more information, please see Chapter 32 *RSA Digital Signature*

Peripheral (RSA_DS). After the chip is powered on, the HMAC module will check whether the key required to calculate the parameter decryption key has been burned in the eFuse block. If the key has been burned, HMAC module will automatically enter the downstream RSA_DS algorithm mode and complete the HMAC calculation based on the chosen key.

29.2.4 HMAC eFuse Configuration

Each HMAC key burned into an eFuse block has a key purpose, specifying for which functionality the key can be used. The HMAC module will not accept a key with a non-matching purpose for any functionality. The HMAC module provides three different functionalities: re-enabling JTAG, RSA_DS KDF in downstream mode, and pure HMAC calculation in upstream mode. For each functionality, there exists a corresponding key purpose, listed in Table 29.2-1. Additionally, another purpose specifies a key which may be used for re-enabling JTAG as well as for serving as RSA_DS KDF.

Before enabling HMAC to do calculations, user should make sure the key to be used has been burned in eFuse by reading the registers EFUSE_KEY_PURPOSE_x (We have a total of 6 keys in eFuse, so the value of x is 0 ~ 5. Among which, EFUSE_KEY_PURPOSE_0 ~ EFUSE_KEY_PURPOSE_1 belong to the register EFUSE_RD_REPEAT_DATA1_REG, and EFUSE_KEY_PURPOSE_2 ~ EFUSE_KEY_PURPOSE_5 belong to the register EFUSE_RD_REPEAT_DATA2_REG) from 9.4 *eFuse Controller (EFUSE)*. Take upstream mode as an example, if there is no EFUSE_KEY_PURPOSE_HMAC_UP in EFUSE_KEY_PURPOSE_0 ~ 5, it means there is no key in eFuse that can be used for the HMAC upstream mode. Users can burn a key to eFuse as follows:

1. Prepare a secret 256-bit HMAC key and burn the key to an empty eFuse block y. As there are 6 blocks for storing a key in eFuse and the numbers of those blocks range from 4 to 9, the value of y is 4 ~ 9. Hence, when talking about key0, it means eFuse block4. Then, program the purpose to EFUSE_KEY_PURPOSE_$(y-4)$. Take upstream mode as an example: after programming the key, the user should program EFUSE_KEY_PURPOSE_HMAC_UP (corresponding value is 8) to EFUSE_KEY_PURPOSE_$(y-4)$. Please see Chapter 9.4 *eFuse Controller (EFUSE)* on how to program eFuse keys.
2. If needed, configure this eFuse key block to be read protected, so that users cannot read its value. A copy of this key should be kept by any party who needs to verify this device.

Please note that the key whose purpose is EFUSE_KEY_PURPOSE_HMAC_DOWN_ALL can be used for both re-enabling JTAG or RSA_DS.

Table 29.2-1. HMAC Purposes and Configuration Value

Purpose	Mode	Value	Description
JTAG Re-enable	Downstream	6	EFUSE_KEY_PURPOSE_HMAC_DOWN_JTAG
RSA_DS KDF	Downstream	7	EFUSE_KEY_PURPOSE_HMAC_DOWN_DIGITAL_SIGNATURE
HMAC Calculation	Upstream	8	EFUSE_KEY_PURPOSE_HMAC_UP
Both JTAG Re-enable and RSA_DS KDF	Downstream	5	EFUSE_KEY_PURPOSE_HMAC_DOWN_ALL

Select eFuse Key Blocks and HMAC Purposes

The eFuse controller provides six key blocks, i.e., KEY0 ~ 5. To select a particular KEYn for an HMAC calculation, write the key number n to register HMAC_SET_PARA_KEY_REG.

Write a correct purpose to register HMAC_SET_PARA_PURPOSE_REG (see Section 29.2.5). Note that the purpose of the key has also been programmed to eFuse memory. Only when the configured HMAC purpose matches the defined purpose of KEYn, the HMAC module will execute the configured calculation. Otherwise, it will return a matching error and stop the current calculation.

For example, suppose a user selects KEY3 for HMAC calculation, and the value programmed to EFUSE_KEY_PURPOSE_3 is 6 (EFUSE_KEY_PURPOSE_HMAC_DOWN_JTAG). Based on Table 29.2-1, KEY3 can be used to re-enable JTAG. If the value written to register HMAC_SET_PARA_PURPOSE_REG is also 6, then the HMAC module will start the process to re-enable JTAG.

Select Key from Key Manager

Users can deploy the HMAC key in the Key Manager. For deployment procedures, please refer to Section 36 *Key Manager*. To select this key from the Key Manager, write the key index 7 to the register HMAC_SET_PARA_KEY_REG. When a key is selected from the Key Manager, the intended operation target in the register is automatically set to "HMAC Calculation", and all other configurations will be ignored.

29.2.5 HMAC Process (Detailed)

The process for users to call HMAC in ESP32-P4 is as follows:

29.2.5.1 Enable HMAC Module

1. Set the peripheral clock bits for HMAC and SHA peripherals in register HP_SYS_CLKRST_CRYPTO_HMAC_CLK_EN, and clear the corresponding peripheral reset bits in register HP_SYS_CLKRST_RST_EN_HMAC. For information on those registers, please see Chapter 11 *Reset and Clock*.
2. Write 1 to register HMAC_SET_START_REG.

29.2.5.2 Configure HMAC Keys and Key Purposes

1. Write the key purpose m to register HMAC_SET_PARA_PURPOSE_REG. The possible key purpose values are shown in Table 29.2-1. For more information, please refer to Section 29.2.4.
2. Select KEYn in eFuse memory as the key by writing n (ranges from 0 to 5, and 7) to register HMAC_SET_PARA_KEY_REG. For more information, please refer to Section 29.2.4.
3. Write 1 to register HMAC_SET_PARA_FINISH_REG to complete the configuration.
4. Read register HMAC_QUERY_ERROR_REG. If its value is 1, it means the purpose of the selected block does not match the configured key purpose and the calculation will not proceed. If its value is 0, it means the purpose of the selected block matches the configured key purpose, and then the calculation can proceed.
5. When the value of HMAC_SET_PARA_PURPOSE_REG is not 8, it means the HMAC module is in downstream mode, proceed with 29.2.5.3. When the value is 8, it means the HMAC module is in upstream mode, proceed with 29.2.5.4.

29.2.5.3 Downstream Mode Process

1. Poll Status register HMAC_QUERY_BUSY_REG until it reads 0.

2. To clear the result and make further usage of the dependent hardware (JTAG or RSA_DS) impossible, write 1 to either register HMAC_SET_INVALIDATE_JTAG_REG to clear the result generated by the JTAG key; or to register HMAC_SET_INVALIDATE_DS_REG to clear the result generated by RSA_DS_KEY. Afterwards, the HMAC process needs to be restarted to re-enable any of the dependent peripherals.

29.2.5.4 Upstream Mode Process

1. Write message in upstream mode:
   (a) Poll Status register HMAC_QUERY_BUSY_REG until it reads 0.
   (b) Apply SHA padding to message as described in Section 29.3.1, and get message block: Block_n ($n >= 0$)
   (c) Perform one of the following operations according to the block number n:
   - If there is only one message block in total which has included all padding bits:
     i. Write the 512-bit Block_n to register HMAC_WR_MESSAGE_*n*_REG (*n*: 0-15). Write 1 to register HMAC_SET_MESSAGE_ONE_REG, to trigger the processing of this message block.
     ii. Poll Status register HMAC_QUERY_BUSY_REG until it reads 0.
     iii. Write 1 to register HMAC_ONE_BLOCK_REG, and finish message transmission.
   - If Block_n is the last padded block:
     i. Write the 512-bit Block_0 to register HMAC_WR_MESSAGE_*n*_REG (*n*: 0-15). Write 1 to register HMAC_SET_MESSAGE_ONE_REG, to trigger the processing of this message block.
     ii. Poll Status register HMAC_QUERY_BUSY_REG until it reads 0.
     iii. Message transmission finished automatically.
   - If Block_n is the second last padded block:
     i. Write the 512-bit Block_n to register HMAC_WR_MESSAGE_*n*_REG (*n*: 0-15). Write 1 to register HMAC_SET_MESSAGE_ONE_REG, to trigger the processing of this message block.
     ii. Poll Status register HMAC_QUERY_BUSY_REG until it reads 0.
     iii. Write 1 to register HMAC_SET_MESSAGE_PAD_REG, and continue message transmission (jump back to step (c)).
   - If Block_n is neither the last nor the second last message block:
     i. Write the 512-bit Block_n to register HMAC_WR_MESSAGE_*n*_REG (*n*: 0-15). Write 1 to register HMAC_SET_MESSAGE_ONE_REG, to trigger the processing of this message block.
     ii. Poll Status register HMAC_QUERY_BUSY_REG until it reads 0.
     iii. Write 1 to register HMAC_SET_MESSAGE_ING_REG, and continue message transmission (jump back to step (c)).
2. Read hash result in upstream mode:
   (a) Poll Status register HMAC_QUERY_BUSY_REG until it reads 0.
   (b) Read hash result from register HMAC_RD_RESULT_*n*_REG (*n*: 0-7).

(c) Write 1 to register HMAC_SET_RESULT_FINISH_REG to finish calculation. The result will be cleared at the same time.

(d) Upstream mode operation is completed.

Note:
The SHA accelerator can be called directly, or used internally by the RSA_DS peripheral and the HMAC module. However, they can not share the hardware resources simultaneously. Therefore, the SHA module must not be called neither by the CPU nor by the RSA_DS peripheral when the HMAC module is in use.

29.3 HMAC Algorithm Details

29.3.1 Padding Bits

The HMAC module uses SHA-256 as hash algorithm. If the input message is not a multiple of 512 bits, the user must apply a SHA-256 padding algorithm in software. The SHA-256 padding algorithm is the same as described in Section *Padding the Message* of FIPS PUB 180-4. In downstream mode, users do not need to input any message or apply padding. The HMAC module uses a default 32-byte pattern of 0x00 for re-enabling JTAG and a 32-byte pattern of 0xff for deriving the AES key for the RSA_DS peripheral.

For the convenience of reading, here we will briefly describe the process of message padding. As shown in Figure 29.3-1, suppose the length of the unpadded message is m bits. Padding steps are as follows:

1. Append one bit of value "1" to the end of the unpadded message.
2. Append k bits of value "0", where k is the smallest non-negative number which satisfies $m + 1 + k \equiv 448 (mod 512)$.
3. Append a 64-bit integer value as a binary block. This block consists of the length of the unpadded message as a big-endian binary integer value m.

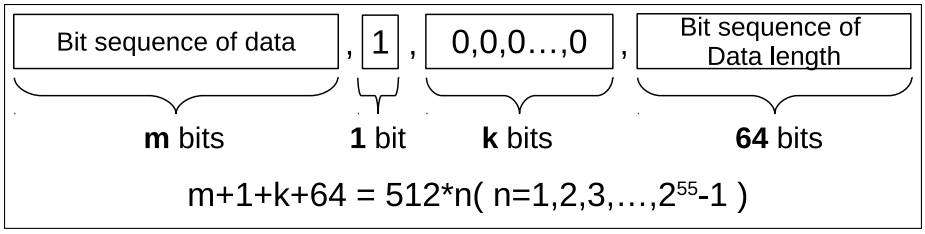


Figure 29.3-1. HMAC SHA-256 Padding Diagram

In upstream mode, if the length of the unpadded message is a multiple of 512 bits, users can configure hardware to apply SHA padding by writing 1 to HMAC_SET_MESSAGE_END_REG or do padding work themselves by writing 1 to HMAC_SET_MESSAGE_PAD_REG. If the length is not a multiple of 512 bits, SHA padding must be manually applied by the user. After the user prepared the padding data, they should complete the subsequent configuration according to the Section 29.2.5.

29.3.2 HMAC Algorithm Structure

The structure of the implemented algorithm in the HMAC module is shown in Figure 29.3-2. This is the standard HMAC algorithm as described in RFC 2104.

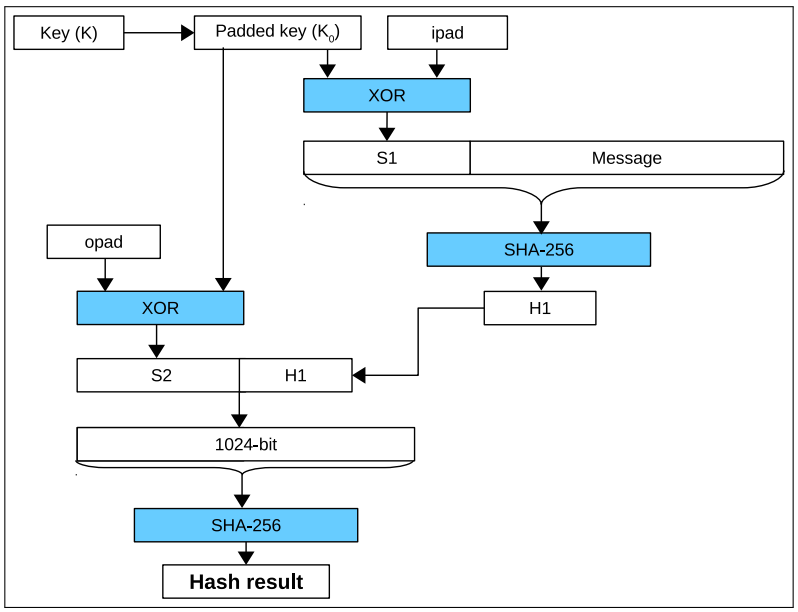


Figure 29.3-2. HMAC Structure Schematic Diagram

In Figure 29.3-2:

1. ipad is a 512-bit message block composed of 64 bytes of 0x36.
2. opad is a 512-bit message block composed of 64 bytes of 0x5c.

The HMAC module appends a 256-bit 0 sequence after the bit sequence of the 256-bit key K in order to get a 512-bit K_0. Then, the HMAC module XORs K_0 with ipad to get the 512-bit S1. Afterwards, the HMAC module appends the input message (multiple of 512 bits) after the 512-bit S1, and exercises the SHA-256 algorithm to get the 256-bit H1.

The HMAC module appends the 256-bit SHA-256 hash result H1 to the 512-bit S2 value, which is calculated using the XOR operation of K_0 and opad. A 768-bit sequence will be generated. Then, the HMAC module uses the SHA padding algorithm described in Section 29.3.1 to pad the 768-bit sequence to a 1024-bit sequence, and applies the SHA-256 algorithm to get the final hash result (256-bit).

29.4 Register Summary

The addresses in this section are relative to HMAC Accelerator base address provided in Table 9.3-2 in Chapter 9 *System and Memory*.

The abbreviations given in Column **Access** are explained in Section *Access Types for Registers*.

Name	Description	Address	Access
Control/Status Registers			
HMAC_SET_START_REG	HMAC start control register	0x0040	WO
HMAC_SET_PARA_FINISH_REG	HMAC configuration completion register	0x004C	WO
HMAC_SET_MESSAGE_ONE_REG	HMAC message control register	0x0050	WO
HMAC_SET_MESSAGE_ING_REG	HMAC message continue register	0x0054	WO
HMAC_SET_MESSAGE_END_REG	HMAC message end register	0x0058	WO
HMAC_SET_RESULT_FINISH_REG	HMAC result reading finish register	0x005C	WO
HMAC_SET_INVALIDATE_JTAG_REG	Invalidate JTAG result register	0x0060	WO
HMAC_SET_INVALIDATE_DS_REG	Invalidate RSA_DS result register	0x0064	WO
HMAC_QUERY_ERROR_REG	Stores matching results between keys generated by users and corresponding purposes	0x0068	RO
HMAC_QUERY_BUSY_REG	Busy state of HMAC module	0x006C	RO
HMAC_SET_MESSAGE_PAD_REG	Software padding register	0x00F0	WO
HMAC_ONE_BLOCK_REG	One block message register	0x00F4	WO
Configuration Registers			
HMAC_SET_PARA_PURPOSE_REG	HMAC parameter configuration register	0x0044	WO
HMAC_SET_PARA_KEY_REG	HMAC parameters configuration register	0x0048	WO
HMAC_SOFT_JTAG_CTRL_REG	Re-enable JTAG register 0	0x00F8	WO
HMAC_WR_JTAG_REG	Re-enable JTAG register 1	0x00FC	WO
HMAC Message Block			
HMAC_WR_MESSAGE_0_REG	Message register 0	0x0080	WO
HMAC_WR_MESSAGE_1_REG	Message register 1	0x0084	WO
HMAC_WR_MESSAGE_2_REG	Message register 2	0x0088	WO
HMAC_WR_MESSAGE_3_REG	Message register 3	0x008C	WO
HMAC_WR_MESSAGE_4_REG	Message register 4	0x0090	WO
HMAC_WR_MESSAGE_5_REG	Message register 5	0x0094	WO
HMAC_WR_MESSAGE_6_REG	Message register 6	0x0098	WO
HMAC_WR_MESSAGE_7_REG	Message register 7	0x009C	WO
HMAC_WR_MESSAGE_8_REG	Message register 8	0x00A0	WO
HMAC_WR_MESSAGE_9_REG	Message register 9	0x00A4	WO
HMAC_WR_MESSAGE_10_REG	Message register 10	0x00A8	WO
HMAC_WR_MESSAGE_11_REG	Message register 11	0x00AC	WO
HMAC_WR_MESSAGE_12_REG	Message register 12	0x00B0	WO
HMAC_WR_MESSAGE_13_REG	Message register 13	0x00B4	WO
HMAC_WR_MESSAGE_14_REG	Message register 14	0x00B8	WO
HMAC_WR_MESSAGE_15_REG	Message register 15	0x00BC	WO
HMAC Upstream Result			

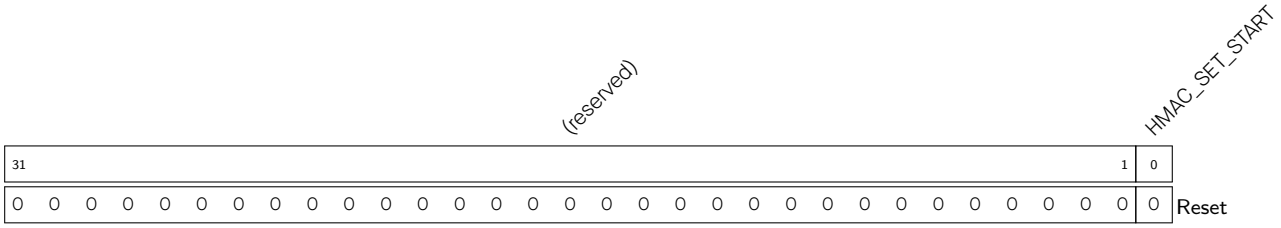
Name	Description	Address	Access
HMAC_RD_RESULT_0_REG	Hash result register 0	0x00C0	RO
HMAC_RD_RESULT_1_REG	Hash result register 1	0x00C4	RO
HMAC_RD_RESULT_2_REG	Hash result register 2	0x00C8	RO
HMAC_RD_RESULT_3_REG	Hash result register 3	0x00CC	RO
HMAC_RD_RESULT_4_REG	Hash result register 4	0x00D0	RO
HMAC_RD_RESULT_5_REG	Hash result register 5	0x00D4	RO
HMAC_RD_RESULT_6_REG	Hash result register 6	0x00D8	RO
HMAC_RD_RESULT_7_REG	Hash result register 7	0x00DC	RO
Version Register			
HMAC_DATE_REG	Version control register	0x01FC	R/W

29.5 Registers

The addresses in this section are relative to HMAC Accelerator base address provided in Table 9.3-2 in Chapter 9 *System and Memory*.

For how to program reserved fields, please refer to Section Programming Reserved Register Field.

Register 29.1. HMAC_SET_START_REG (0x0040)



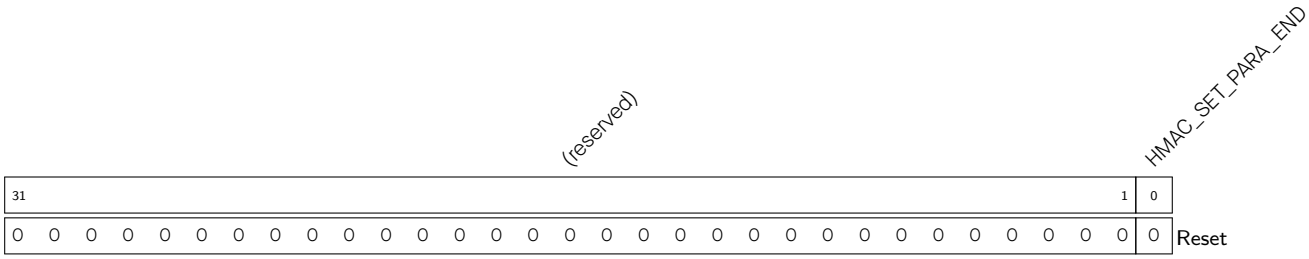
HMAC_SET_START Configures whether or not to enable HMAC.

0: Disable HMAC

1: Enable HMAC

(WO)

Register 29.2. HMAC_SET_PARA_FINISH_REG (0x004C)



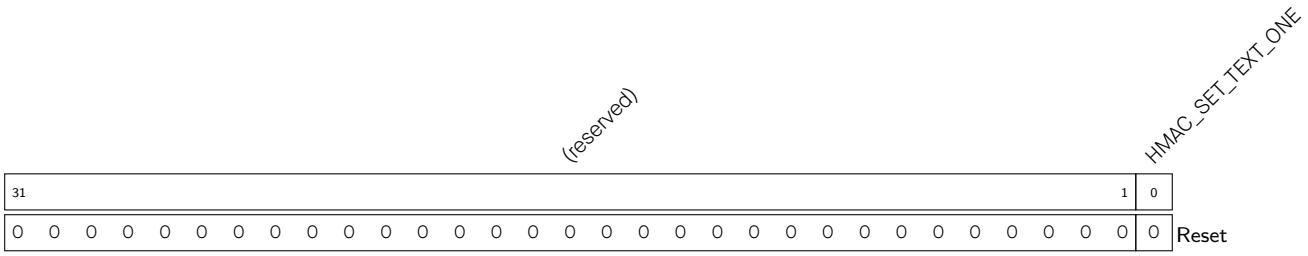
HMAC_SET_PARA_END Configures whether to finish HMAC configuration.

0: No effect

1: Finish configuration

(WO)

Register 29.3. HMAC_SET_MESSAGE_ONE_REG (0x0050)



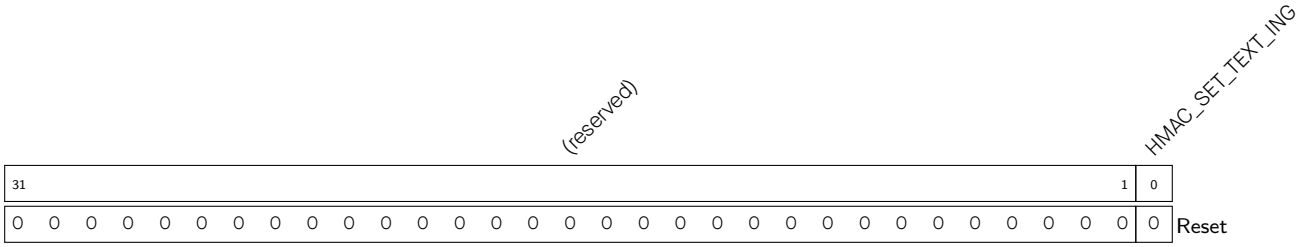
HMAC_SET_TEXT_ONE Calls SHA to calculate one message block.

0: No effect

1: Calls SHA to calculate one message block.

(WO)

Register 29.4. HMAC_SET_MESSAGE_ING_REG (0x0054)



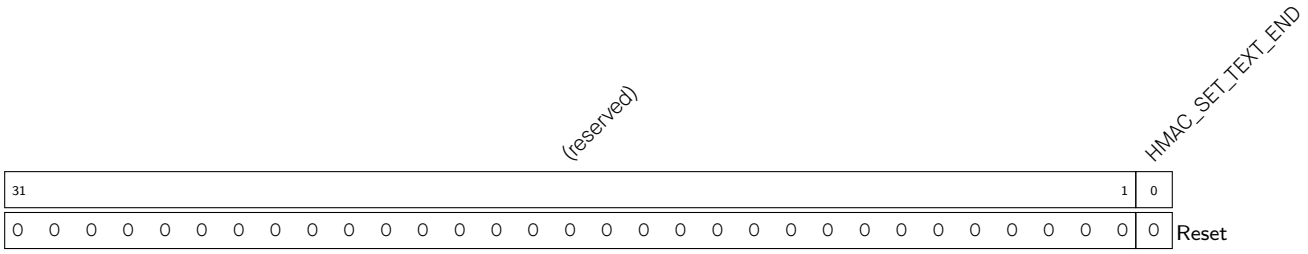
HMAC_SET_TEXT_ING Configures whether or not there are unprocessed message blocks.

0: No unprocessed message block

1: There are still some message blocks to be processed.

(WO)

Register 29.5. HMAC_SET_MESSAGE_END_REG (0x0058)



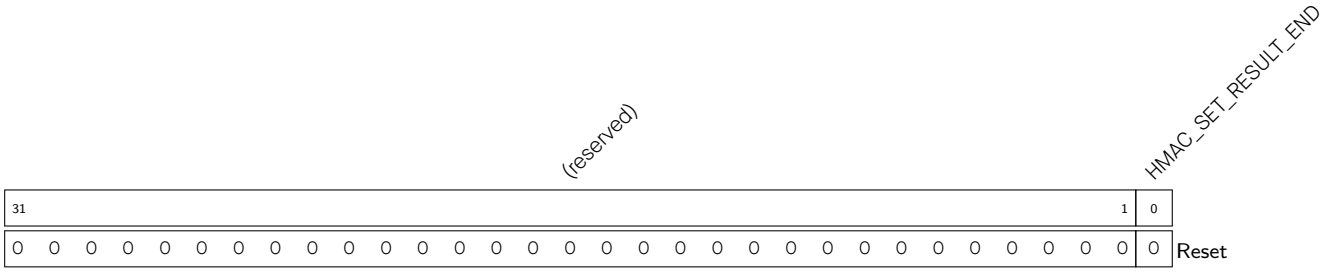
HMAC_SET_TEXT_END Configures whether to start hardware padding.

0: No effect

1: Start hardware padding

(WO)

Register 29.6. HMAC_SET_RESULT_FINISH_REG (0x005C)



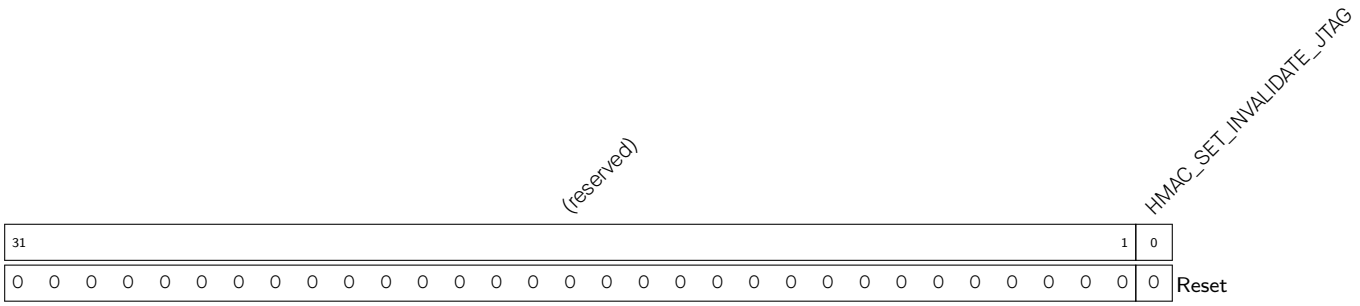
HMAC_SET_RESULT_END Configures whether to exit upstream mode and clear calculation results.

0: Not exit

1: Exit upstream mode and clear calculation results.

(WO)

Register 29.7. HMAC_SET_INVALIDATE_JTAG_REG (0x0060)



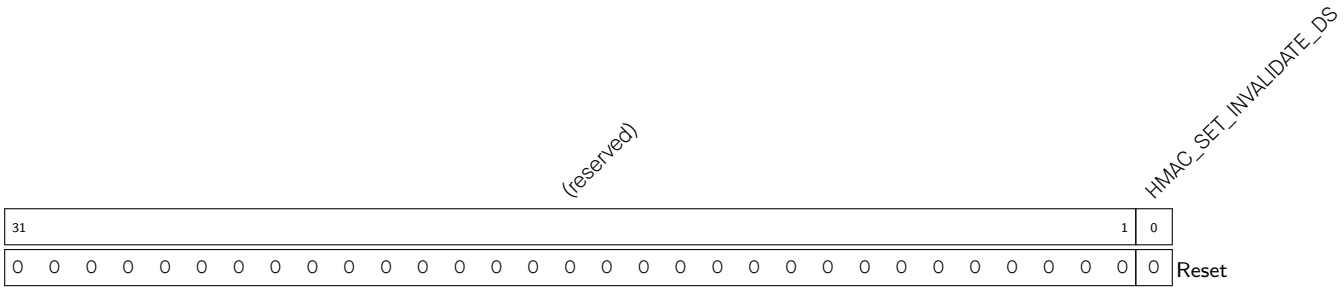
HMAC_SET_INVALIDATE_JTAG Configures whether or not to clear calculation results when re-enabling JTAG in downstream mode.

0: Not clear

1: Clear calculation results

(WO)

Register 29.8. HMAC_SET_INVALIDATE_DS_REG (0x0064)



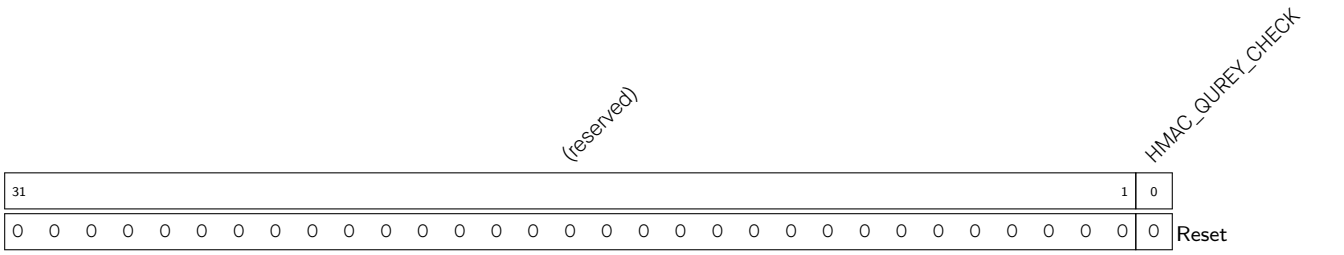
HMAC_SET_INVALIDATE_DS Configures whether or not to clear calculation results of the RSA_DS peripheral in downstream mode.

0: Not clear

1: Clear calculation results

(WO)

Register 29.9. HMAC_QUERY_ERROR_REG (0x0068)



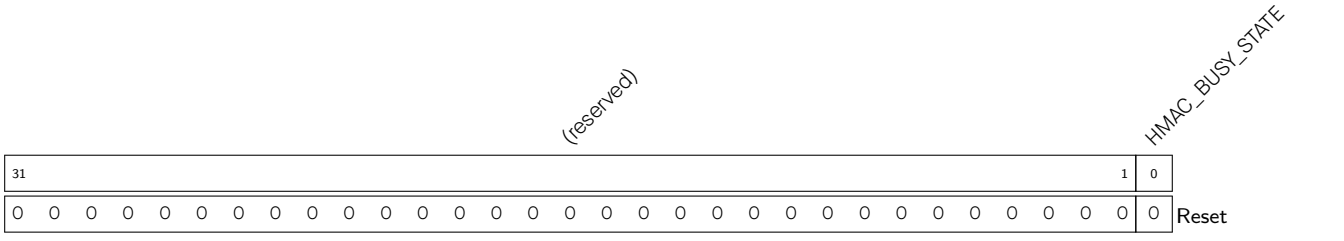
HMAC_QUREY_CHECK Represents whether or not an HMAC key matches the purpose.

0: Match

1: Error

(RO)

Register 29.10. HMAC_QUERY_BUSY_REG (0x006C)



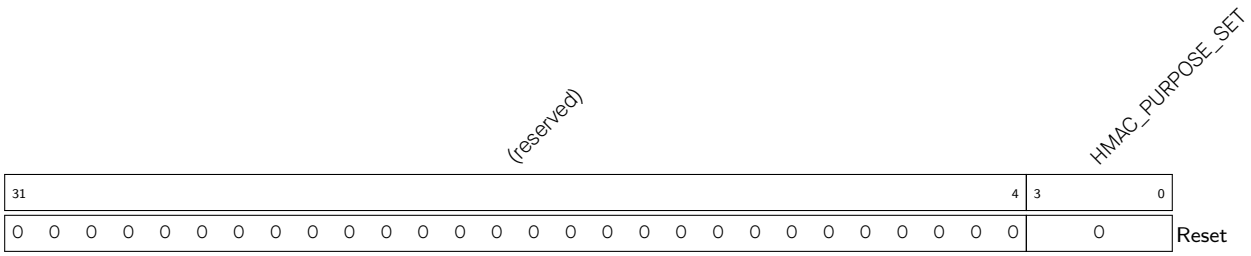
HMAC_BUSY_STATE Represents whether or not HMAC is in a busy state. Before configuring HMAC, please make sure HMAC is in an IDLE state.

0: Idle

1: HMAC is still working on the calculation

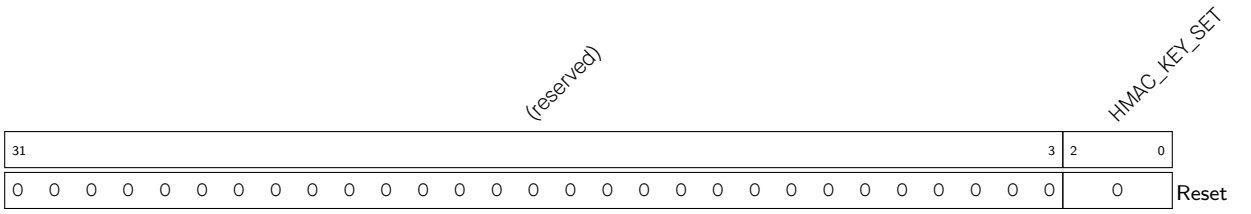
(RO)

Register 29.11. HMAC_SET_PARA_PURPOSE_REG (0x0044)



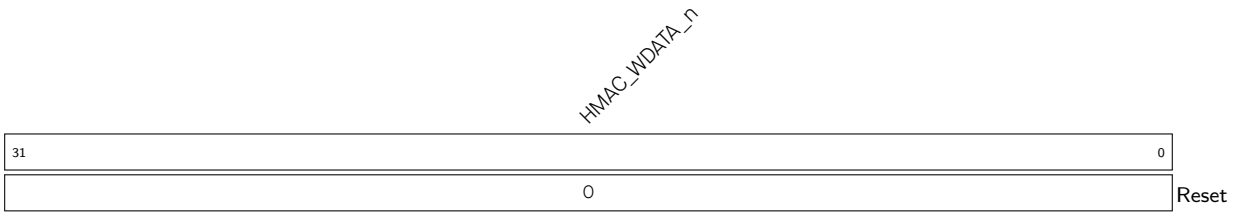
HMAC_PURPOSE_SET Configures the HMAC purpose, refer to the Table 29.2-1. (WO)

Register 29.12. HMAC_SET_PARA_KEY_REG (0x0048)



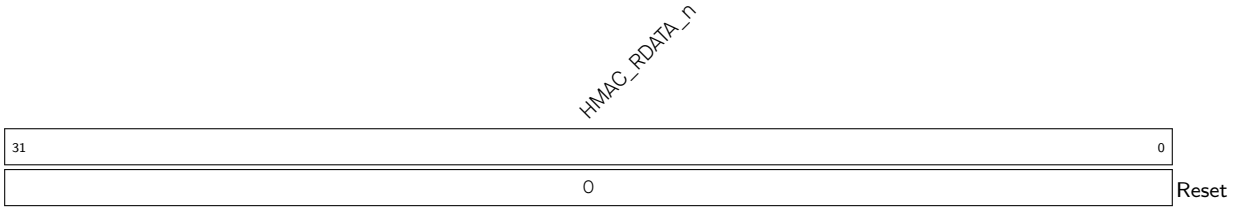
HMAC_KEY_SET Configures HMAC key. There are seven keys with index 0 ~ 5, and 7. Write the index of the selected key to this field. (WO)

Register 29.13. HMAC_WR_MESSAGE_*n*_REG (*n*: 0-15) (0x0080+4**n*)



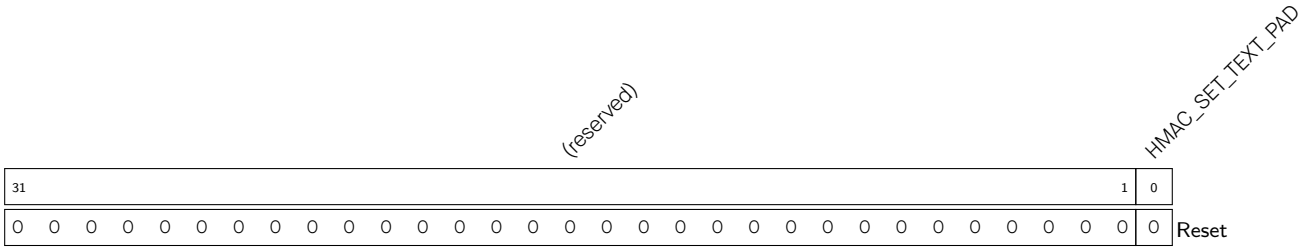
HMAC_WDATA_*n* Represents the *n*th 32-bit of message. (WO)

Register 29.14. HMAC_RD_RESULT_*n*_REG (*n*: 0-7) (0x00C0+4**n*)



HMAC_RDATA_*n* Represents the *n*th 32-bit of hash result. (RO)

Register 29.15. HMAC_SET_MESSAGE_PAD_REG (0x00F0)



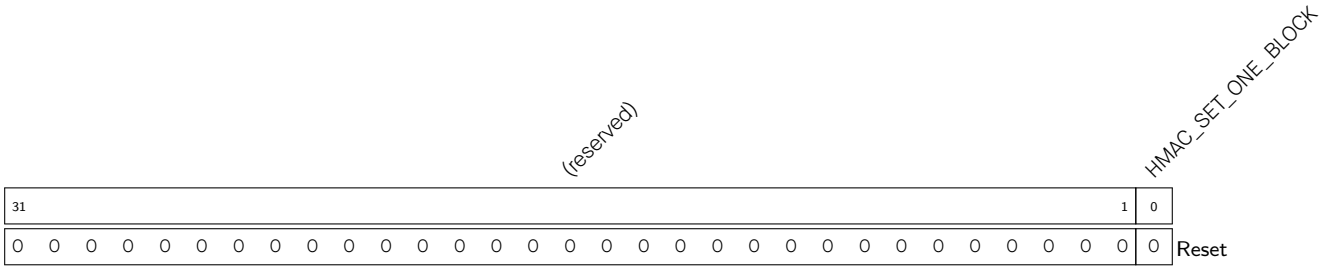
HMAC_SET_TEXT_PAD Configures whether or not the padding is applied by software.

0: Not applied by software

1: Applied by software

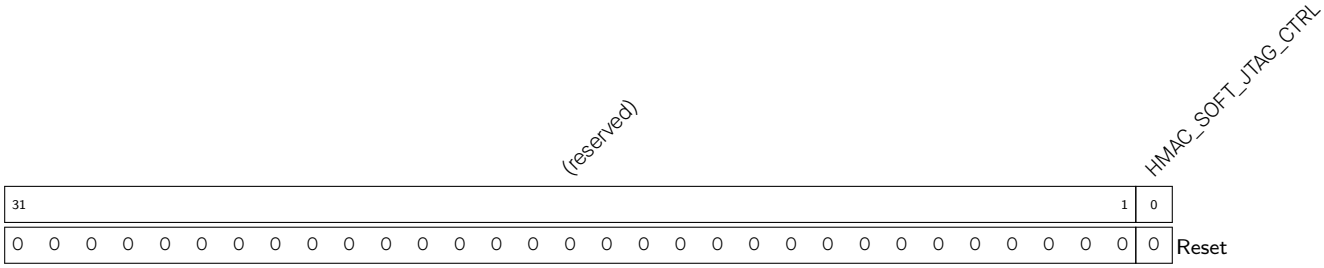
(WO)

Register 29.16. HMAC_ONE_BLOCK_REG (0x00F4)



HMAC_SET_ONE_BLOCK Write 1 to indicate there is only one block which already contains padding bits and there is no need for padding. (WO)

Register 29.17. HMAC_SOFT_JTAG_CTRL_REG (0x00F8)



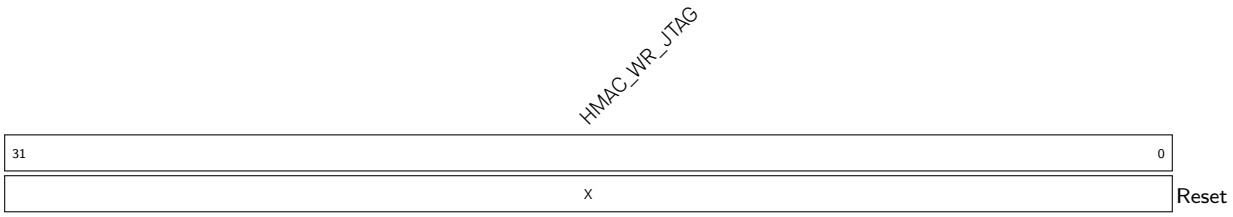
HMAC_SOFT_JTAG_CTRL Configures whether or not to enable JTAG authentication mode.

0: Disable

1: Enable

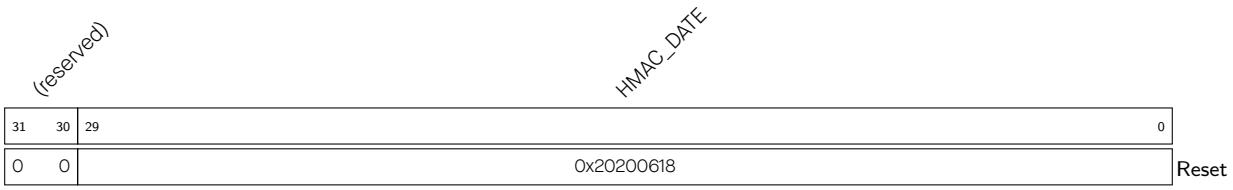
(WO)

Register 29.18. HMAC_WR_JTAG_REG (0x00FC)



HMAC_WR_JTAG Writes the comparing input used for re-enabling JTAG. (WO)

Register 29.19. HMAC_DATE_REG (0x01FC)



HMAC_DATE Version control register. (R/W)

Chapter 30

RSA Accelerator (RSA)

30.1 Introduction

The RSA accelerator provides hardware support for high-precision computation used in various RSA asymmetric cipher algorithms, significantly improving their run time and reducing their software complexity compared with RSA algorithms implemented solely in software. The RSA accelerator also supports operands of different lengths, which provides more flexibility during the computation.

30.2 Features

The following functionality is supported:

- Large-number modular exponentiation with two additional acceleration options to further increase the calculation speed
- Large-number modular multiplication
- Large-number multiplication
- Operands of different lengths
- Interrupt on completion of computation

30.3 Functional Description

The RSA accelerator is activated by setting the HP_SYS_CLKRST_CRYPTO_RSA_CLK_EN bit in the HP_SYS_CLKRST_PERI_CLK_CTRL25_REG register and clearing the HP_SYS_CLKRST_RST_EN_RSA bit in the HP_SYS_CLKRST_HP_RST_EN2_REG register. Additionally, users also need to clear HP_SYS_CLKRST_RST_EN_DS and HP_SYS_CLKRST_RST_EN_ECDSA bits to reset the RSA Digital Signature Peripheral (RSA_DS) and the ECDSA Digital Signature Peripheral (ECDSA_DS) accelerators.

The RSA accelerator is only available after the RSA-related memories are initialized. The content of the RSA_QUERY_CLEAN_REG register is 0 during initialization and will become 1 after the initialization is done. Therefore, wait until RSA_QUERY_CLEAN_REG becomes 1 before using the RSA accelerator.

The RSA_INT_ENA_REG register is used to control the interrupt triggered on completion of computation. Write 1 or 0 to this field to enable or disable the interrupt. By default, the interrupt function of the RSA accelerator is enabled.

Notice:

ESP32-P4's RSA Digital Signature Peripheral (RSA_DS) module also calls the RSA accelerator when working. Therefore,

users cannot access the RSA accelerator when the RSA Digital Signature Peripheral (RSA_DS) module is working.

30.3.1 Definitions and Representations

ESP32-P4's RSA accelerator supports operands of different lengths $N = 32 \times n$.

To represent the numbers used as operands, let us define a base-b positional notation, as follows:

$$b = 2^{32}$$

Using this notation, each number is represented by a sequence of base-b digits:

$$n = \frac{N}{32}$$
$$Z = (Z_{n-1}Z_{n-2}\cdots Z_0)_b$$
$$X = (X_{n-1}X_{n-2}\cdots X_0)_b$$
$$Y = (Y_{n-1}Y_{n-2}\cdots Y_0)_b$$
$$M = (M_{n-1}M_{n-2}\cdots M_0)_b$$
$$\overline{r} = (\overline{r}_{n-1}\overline{r}_{n-2}\cdots \overline{r}_0)_b$$

Each of the values in $Z_{n-1}\cdots Z_0$, $X_{n-1}\cdots X_0$, $Y_{n-1}\cdots Y_0$, $M_{n-1}\cdots M_0$, $\overline{r}_{n-1}\cdots \overline{r}_0$ represents one base-b digit (a 32-bit word).

Z_{n-1}, X_{n-1}, Y_{n-1}, M_{n-1} and $\overline{r}_{n-1}$ are the most significant bits of Z, X, Y, M, while Z_0, X_0, Y_0, M_0 and $\overline{r}_0$ are the least significant bits.

If we define $R = b^n$, the additional argument $\overline{r}$ can be calculated as $\overline{r} = R^2 \bmod M$.

Also, argument M' can be calculated using the formula below:

$$M' = -M^{-1} \bmod b$$

where, M^{-1} is the modular multiplicative inverse of M, and it can be calculated with the extended binary GCD algorithm.

30.3.2 Large-Number Modular Exponentiation

Large-number modular exponentiation performs $Z = X^Y \bmod M$. The computation is based on Montgomery multiplication. Therefore, aside from the X, Y, and M arguments, two additional ones are needed — $\overline{r}$ and M', which need to be calculated in advance by software.

The RSA accelerator supports operands of length $N = 32 \times n$, where $n \in \{1, 2, 3, \ldots, 128\}$. The bit lengths of arguments Z, X, Y, M, and $\overline{r}$ can be arbitrary N, but all numbers in a calculation must be of the same length. The bit length of M' must be 32.

Large-number modular exponentiation on the ESP32-P4 can be implemented as follows:

1. Write 1 or 0 to the RSA_INT_ENA field to enable or disable the interrupt function.
2. Configure relevant registers:
   (a) Write $(\frac{N}{32} - 1)$ to the RSA_MODE_REG register.

(b) Write M' to the RSA_M_PRIME_REG register.

(c) Configure registers related to the acceleration options, which are described later in Section 30.3.5.

3. Write X_i, Y_i, M_i and $\overline{r}_i$ for $i \in \{0, 1, \ldots, n-1\}$ to memory blocks RSA_X_MEM, RSA_Y_MEM, RSA_M_MEM and RSA_Z_MEM. The capacity of each memory block is 128 words. Each word of each memory block can store one base-b digit. The memory blocks use the little endian format for storage, i.e., the least significant digit of each number is in the lowest address.

   Users need to write data to each memory block only according to the length of the number; data beyond this length is ignored.

4. Write 1 to the RSA_SET_START_MODEXP field of the RSA_SET_START_MODEXP_REG register to start computation.
5. Wait for the completion of computation, which happens when the content of RSA_QUERY_IDLE becomes 1 or the RSA interrupt occurs, if enabled.
6. Read the result Z_i for $i \in \{0, 1, \ldots, n-1\}$ from RSA_Z_MEM.
7. If you have the interrupt enabled, write 1 to RSA_CLEAR_INTERRUPT to clear the interrupt.

After the computation, the RSA_MODE_REG register, memory blocks RSA_Y_MEM and RSA_M_MEM, as well as the RSA_M_PRIME_REG remain unchanged. However, X_i in RSA_X_MEM and $\overline{r}_i$ in RSA_Z_MEM computation are overwritten, and only these overwritten memory blocks need to be re-initialized before starting another computation.

30.3.3 Large-Number Modular Multiplication

Large-number modular multiplication performs $Z = X \times Y \bmod M$. This computation is based on Montgomery multiplication. Therefore, similar to the large-number modular exponentiation, two additional arguments are needed – $\overline{r}$ and M', which need to be calculated in advance by software.

The RSA accelerator supports large-number modular multiplication with operands of length $N = 32 \times n$, where $n \in \{1, 2, 3, \ldots, 128\}$.

The computation can be executed as follows:

1. Write 1 or 0 to the RSA_INT_ENA_REG register to enable or disable the interrupt function.
2. Configure relevant configuration registers:

   (a) Write $(\frac{N}{32} - 1)$ to the RSA_MODE_REG register.

   (b) Write M' to the RSA_M_PRIME_REG register.

3. Write X_i, Y_i, M_i, and $\overline{r}_i$ for $i \in \{0, 1, \ldots, n-1\}$ to memory blocks RSA_X_MEM, RSA_Y_MEM, RSA_M_MEM, and RSA_Z_MEM, respectively. The capacity of each memory block is 128 words. Each word of each memory block can store one base-b digit. The memory blocks use the little endian format for storage, i.e., the least significant digit of each number is in the lowest address.

   Users need to write data to each memory block only according to the length of the number; data beyond this length are ignored.

4. Write 1 to the RSA_SET_START_MODMULT field.

5. Wait for the completion of computation, which happens when the content of RSA_QUERY_IDLE becomes 1 or the RSA interrupt occurs, if enabled.
6. Read the result Z_i for $i \in \{0, 1, \ldots, n-1\}$ from RSA_Z_MEM.
7. If you have the interrupt enabled, write 1 to RSA_CLEAR_INTERRUPT to clear the interrupt, .

After the computation, the length of operands in RSA_MODE_REG, the X_i in memory RSA_X_MEM, the Y_i in memory RSA_Y_MEM, the M_i in memory RSA_M_MEM, and the M' in memory RSA_M_PRIME_REG remain unchanged. However, the $\overline{r}_i$ in memory RSA_Z_MEM has already been overwritten, and only this overwritten memory block needs to be re-initialized before starting another computation.

30.3.4 Large-Number Multiplication

Large-number multiplication performs $Z = X \times Y$. The length of result Z is twice that of operand X and operand Y. Therefore, the RSA accelerator only supports large-number multiplication with operand length $N = 32 \times n$, where $n \in \{1, 2, 3, \ldots, 64\}$. The length $\hat{N}$ of result Z is $2 \times N$.

The computation can be executed as follows:

1. Write 1 or 0 to the RSA_INT_ENA_REG register to enable or disable the interrupt function.
2. Write $(\frac{\hat{N}}{32} - 1)$, i.e., $(\frac{N}{16} - 1)$ to the RSA_MODE_REG register.
3. Write X_i and Y_i for $\in \{0, 1, \ldots, n-1\}$ to memory blocks RSA_X_MEM and RSA_Z_MEM. Each word of each memory block can store one base-b digit. The memory blocks use the little endian format for storage, i.e., the least significant digit of each number is in the lowest address. n is $\frac{N}{32}$.

   Write X_i for $i \in \{0, 1, \ldots, n-1\}$ to the address of the i words of the RSA_X_MEM memory block. Note that Y_i for $i \in \{0, 1, \ldots, n-1\}$ will not be written to the address of the i words of the RSA_Z_MEM register, but the address of the $n + i$ words, i.e., the base address of the RSA_Z_MEM memory plus the address offset $4 \times (n + i)$.

   Users need to write data to each memory block only according to the length of the number; data beyond this length is ignored.
4. Write 1 to the RSA_SET_START_MULT register.
5. Wait for the completion of computation, which happens when the content of RSA_QUERY_IDLE becomes 1 or the RSA interrupt occurs, if enabled.
6. Read the result Z_i for $i \in \{0, 1, \ldots, \hat{n}-1\}$ from the RSA_Z_MEM register. $\hat{n}$ is $2 \times n$.
7. If you have the interrupt enabled, write 1 to RSA_CLEAR_INTERRUPT to clear the interrupt.

After the computation, the length of operands in RSA_MODE_REG and the X_i in memory RSA_X_MEM remain unchanged. However, the Y_i in memory RSA_Z_MEM has already been overwritten, and only this overwritten memory block needs to be re-initialized before starting another computation.

30.3.5 Options for Additional Acceleration

The ESP32-P4 RSA accelerator also provides SEARCH and CONSTANT_TIME options that can be configured to further accelerate the large-number modular exponentiation. By default, both options are configured as no additional acceleration.

Users can choose to use one or two of these options to further accelerate the computation. Note that, even when none of these two options is configured, using the hardware RSA accelerator is still much faster than implementing the RSA algorithm in software.

To be more specific, when neither of these two options are configured for additional acceleration, the time required to calculate $Z = X^Y \bmod M$ is solely determined by the lengths of operands. When either or both of these two options are configured for additional acceleration, the time required is also correlated with the 0 and 1 bit distribution in Y.

To better illustrate how these two options work, first assume Y is represented in binaries as

$$Y = (\widetilde{Y}_{N-1}\widetilde{Y}_{N-2}\cdots\widetilde{Y}_{t+1}\widetilde{Y}_t\widetilde{Y}_{t-1}\cdots\widetilde{Y}_0)_2$$

where,

- N is the length of Y,
- $\widetilde{Y}_t$ is 1,
- $\widetilde{Y}_{N-1}$, $\widetilde{Y}_{N-2}$, ..., $\widetilde{Y}_{t+1}$ are all equal to 0,
- and $\widetilde{Y}_{t-1}$, $\widetilde{Y}_{t-2}$, ..., $\widetilde{Y}_0$ are either 0 or 1 but exactly m bits should be equal to 0 and t-m bits 1, i.e., the Hamming weight of $\widetilde{Y}_{t-1}\widetilde{Y}_{t-2}, \cdots, \widetilde{Y}_0$ is $t - m$.

When either of these two options is configured for additional acceleration:

- SEARCH Option (Configuring RSA_SEARCH_ENABLE to 1 for additional acceleration)
  - The accelerator ignores the bit positions of $\widetilde{Y}_i$, where $i > \alpha$. Search position α is set by configuring the RSA_SEARCH_POS_REG register. Set α to a number smaller than N-1, which otherwise leads to the same result as if this option is not used for additional acceleration. The best acceleration performance can be achieved by setting α to t, in which case all the $\widetilde{Y}_{N-1}$, $\widetilde{Y}_{N-2}$, ..., $\widetilde{Y}_{t+1}$ of 0 s are ignored during the calculation. Note that if you set α to be less than t, then the result of the modular exponentiation $Z = X^Y \bmod M$ will be incorrect.
  - Note that this option compromises the security because it ignores some bits, which essentially shortens the key length, thus should not be enabled for applications with high security requirement.
- CONSTANT_TIME Option (Configuring RSA_CONSTANT_TIME_REG to 0 for additional acceleration)
  - The accelerator speeds up the calculation by simplifying the calculation concerning the 0 bits of Y. Therefore, the higher the proportion of bits 0 against bits 1, the better is the acceleration performance.
  - Note that this option also compromises the security because its time cost correlates with the 0/1 distribution of the key, which can be used in a Side Channel Attack (SCA), thus should not be enabled for applications with high security requirement.

Below is an example to demonstrate the performance of the RSA accelerator under different combinations of SEARCH and CONSTANT_TIME configuration. In this example:

- We perform $Z = X^Y \bmod M$
- N = 3072

- Y = 65537
- X and M are taken at random
- When the SEARCH option is enabled, α, i.e., the position register RSA_SEARCH_POS_REG, is set to 16.

Table 30.3-1 below demonstrates the time cost in clock cycles under different combinations of SEARCH and CONSTANT_TIME configuration when performing $Z = X^Y \bmod M$ as described above.

Table 30.3-1. Acceleration Performance

SEARCH Option	CONSTANT_TIME Option	Time Cost (clock cycle)
No acceleration	No acceleration	174.7×10^6
Accelerated	No acceleration	1.023×10^6
No acceleration	Acceleration	0.546×10^6
Acceleration	Acceleration	0.540×10^6

As shown in Table 30.3-1:

- The time cost is biggest when none of these two options is configured for additional acceleration.
- The time cost is smallest when both of these two options are configured for additional acceleration.
- The time cost can be dramatically reduced when either or both option(s) are configured for additional acceleration.

30.4 Interrupts

ESP32-P4's RSA accelerator can generate the following interrupt signal that will be sent to the Interrupt Matrix.

- RSA_INTR

There is one internal interrupt source from the RSA accelerator that can generate the above interrupt signal. The interrupt source from the RSA accelerator is listed with its trigger condition and the resulting interrupt signal in Table 30.4-1.

Table 30.4-1. RSA's Internal Interrupt Source

Internal Interrupt Source	Trigger Condition	Interrupt Signal
RSA_CALC_DONE_INT	Completion of an RSA calculation	RSA_INTR

Note:
For definitions of *interrupt*, *interrupt signal*, *interrupt source*, and their correlations, please refer to Chapter 13 *Interrupt Matrix* > Section 13.2 *Interrupt Terminology in ESP32-P4*.

30.5 Memory Summary

The addresses in this section are relative to the RSA accelerator base address provided in Table 9.3-2 in Chapter 9 *System and Memory*.

The abbreviations given in Column **Access** are explained in Section *Access Types for Registers*.

Table 30.5-1. RSA Accelerator Memory Blocks

Name	Description	Size (byte)	Starting Address	Ending Address	Access
RSA_M_MEM	Memory M	512	0x0000	0x01FF	R/W
RSA_Z_MEM	Memory Z	512	0x0200	0x03FF	R/W
RSA_Y_MEM	Memory Y	512	0x0400	0x05FF	R/W
RSA_X_MEM	Memory X	512	0x0600	0x07FF	R/W

30.6 Register Summary

The addresses in this section are relative to the RSA accelerator base address provided in Table 9.3-2 in Chapter 9 *System and Memory*.

The abbreviations given in Column **Access** are explained in Section *Access Types for Registers*.

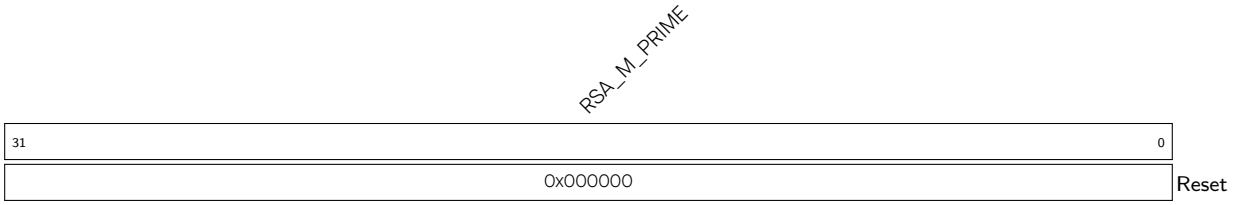
Name	Description	Address	Access
Control or Configuration Registers			
RSA_M_PRIME_REG	Represents M'	0x0800	R/W
RSA_MODE_REG	Configures RSA length	0x0804	R/W
RSA_SET_START_MODEXP_REG	Starts modular exponentiation	0x080C	WT
RSA_SET_START_MODMULT_REG	Starts modular multiplication	0x0810	WT
RSA_SET_START_MULT_REG	Starts multiplication	0x0814	WT
RSA_QUERY_IDLE_REG	Represents the RSA status	0x0818	RO
RSA_CONSTANT_TIME_REG	Configures the constant_time option	0x0820	R/W
RSA_SEARCH_ENABLE_REG	Configures the search option	0x0824	R/W
RSA_SEARCH_POS_REG	Configures the search position	0x0828	R/W
Status Register			
RSA_QUERY_CLEAN_REG	RSA initialization status	0x0808	RO
Interrupt Registers			
RSA_INT_CLR_REG	Clears RSA interrupt	0x081C	WT
RSA_INT_ENA_REG	Enables the RSA interrupt	0x082C	R/W
Version Control Register			
RSA_DATE_REG	Version control register	0x0830	R/W

30.7 Registers

The addresses in this section are relative to the RSA accelerator base address provided in Table 9.3-2 in Chapter 9 *System and Memory*.

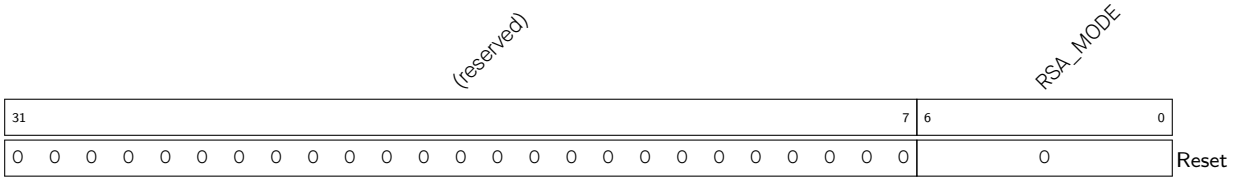
For how to program reserved fields, please refer to Section IX .

Register 30.1. RSA_M_PRIME_REG (0x0800)



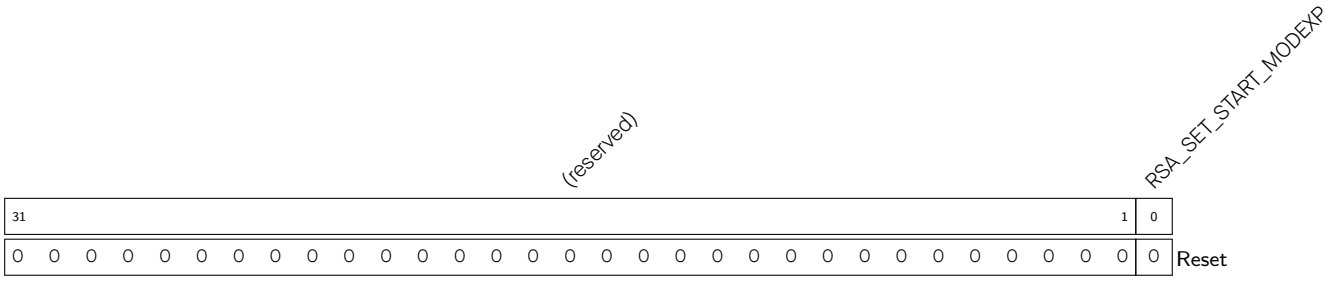
RSA_M_PRIME Represents M'. (R/W)

Register 30.2. RSA_MODE_REG (0x0804)



RSA_MODE Configures the RSA length. (R/W)

Register 30.3. RSA_SET_START_MODEXP_REG (0x080C)



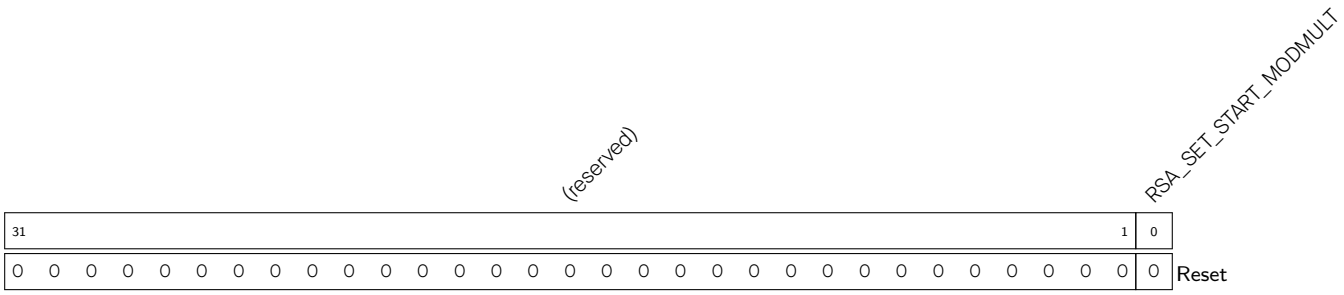
RSA_SET_START_MODEXP Configures whether or not to starts the modular exponentiation.

0: No effect

1: Start

(WT)

Register 30.4. RSA_SET_START_MODMULT_REG (0x0810)



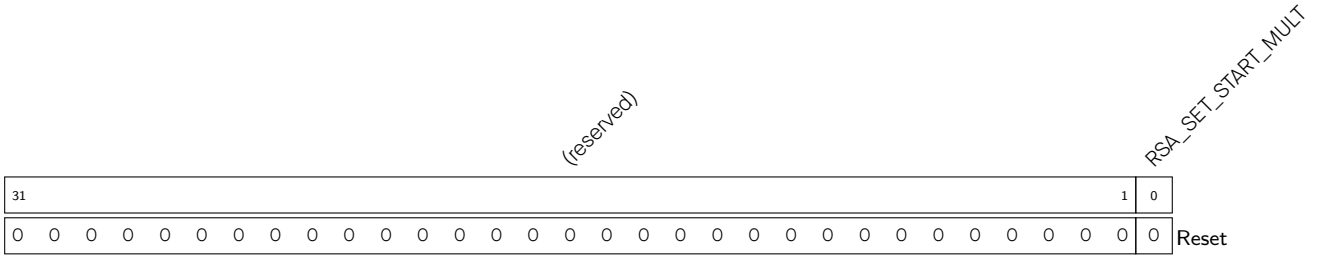
RSA_SET_START_MODMULT Configures whether or not to start the modular multiplication.

0: No effect

1: Start

(WT)

Register 30.5. RSA_SET_START_MULT_REG (0x0814)



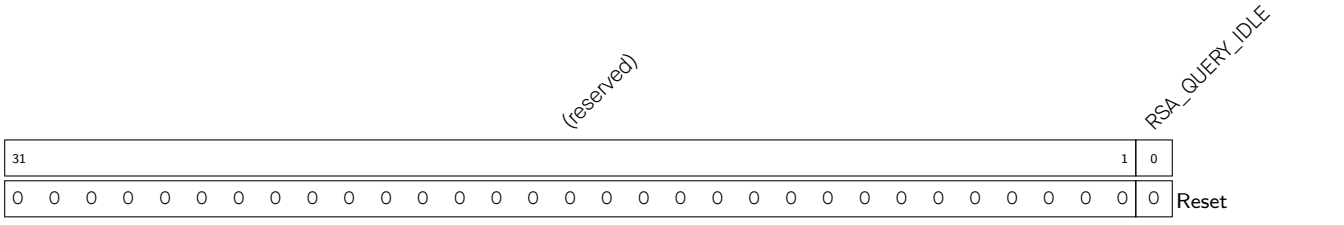
RSA_SET_START_MULT Configures whether or not to start the multiplication.

0: No effect

1: Start

(WT)

Register 30.6. RSA_QUERY_IDLE_REG (0x0818)



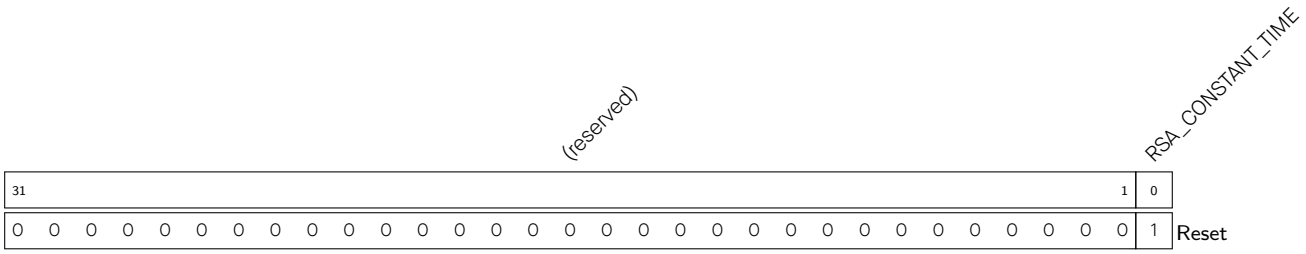
RSA_QUERY_IDLE Represents the RSA status.

0: Busy

1: Idle

(RO)

Register 30.7. RSA_CONSTANT_TIME_REG (0x0820)



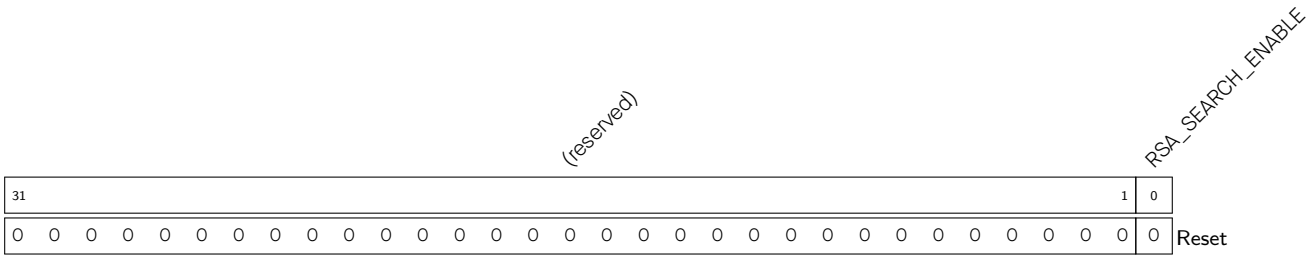
RSA_CONSTANT_TIME Configures the constant_time option.

0: Acceleration

1: No acceleration (default)

(R/W)

Register 30.8. RSA_SEARCH_ENABLE_REG (0x0824)



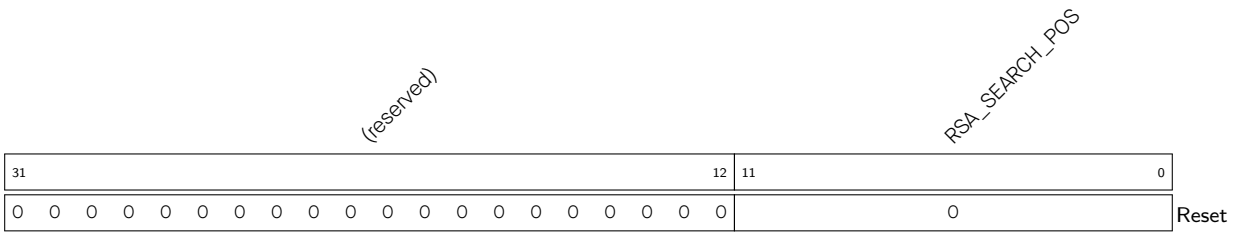
RSA_SEARCH_ENABLE Configures the search option.

0: No acceleration (default)

1: Acceleration

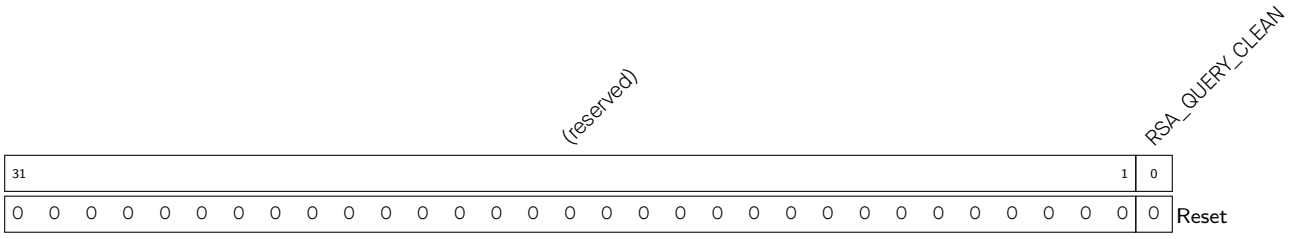
This option should be used together with RSA_SEARCH_POS_REG. (R/W)

Register 30.9. RSA_SEARCH_POS_REG (0x0828)



RSA_SEARCH_POS Configures the starting address to start search. This field should be used together with RSA_SEARCH_ENABLE_REG. The field is only valid when RSA_SEARCH_ENABLE is high. (R/W)

Register 30.10. RSA_QUERY_CLEAN_REG (0x0808)



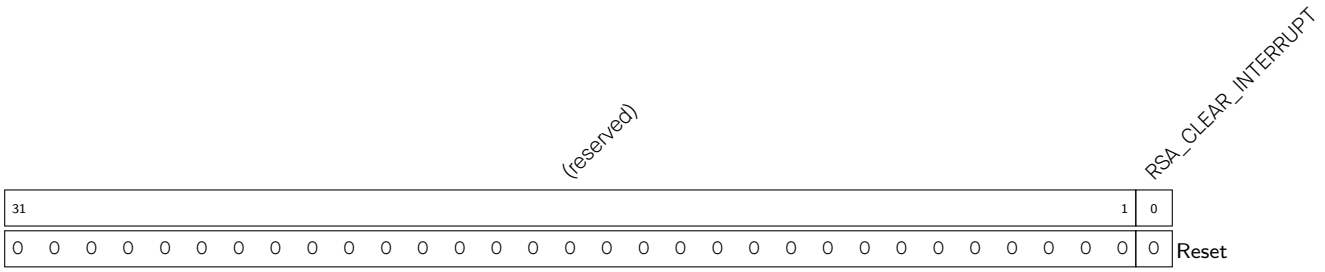
RSA_QUERY_CLEAN Represents whether or not the RSA memory completes initialization.

0: Not complete

1: Completed

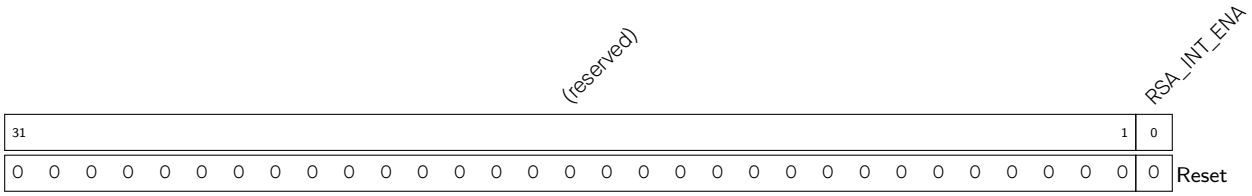
(RO)

Register 30.11. RSA_INT_CLR_REG (0x081C)



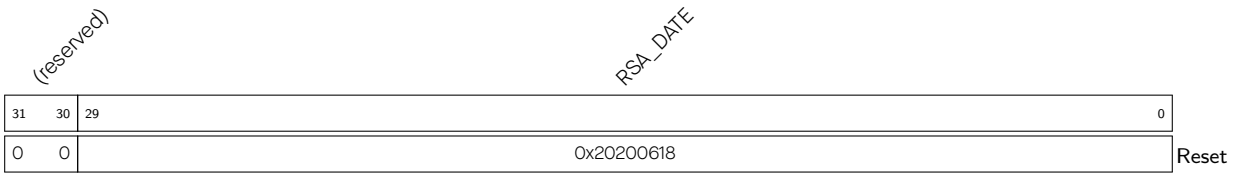
RSA_CLEAR_INTERRUPT Write 1 to clear the RSA interrupt. (WT)

Register 30.12. RSA_INT_ENA_REG (0x082C)



RSA_INT_ENA Write 1 to enable the RSA interrupt. (R/W)

Register 30.13. RSA_DATE_REG (0x0830)



RSA_DATE Version control register. (R/W)

Chapter 31

SHA Accelerator (SHA)

31.1 Introduction

ESP32-P4 integrates an SHA accelerator, which is a hardware device that speeds up the SHA algorithm significantly, compared to an SHA algorithm implemented solely in software. The SHA accelerator integrated in ESP32-P4 has two working modes, which are Typical SHA and DMA-SHA.

31.2 Features

The following functionality is supported:

- The following hash algorithms introduced in FIPS PUB 180-4 Spec.
  - SHA-1
  - SHA-224
  - SHA-256
  - SHA-384
  - SHA-512
  - SHA-512/224
  - SHA-512/256
  - SHA-512/*t*
- Two working modes
  - Typical SHA
  - DMA-SHA
- Interleaved function when working in Typical SHA working mode
- Interrupt function when working in DMA-SHA working mode

31.3 Working Modes

The SHA accelerator integrated in ESP32-P4 has two working modes.

- Typical SHA Working Mode: all the data is written and read via CPU directly.
- DMA-SHA Working Mode: all the data is read via DMA. That is, users can configure the GDMA-AXI controller to read all the data needed for hash operation, thus releasing CPU for completing other tasks.

The SHA accelerator is activated by setting the HP_SYS_CLKRST_CRYPTO_SHA_CLK_EN register field and clearing the HP_SYS_CLKRST_RST_EN_SHA register field. Additionally, due to the hardware reuse relationship between cryptography modules, set the related reset register field of RSA Digital Signature Peripheral (RSA_DS), HMAC Accelerator (HMAC), and ECDSA Digital Signature Peripheral (ECDSA_DS) to 1 can reset the SHA module. Therefore, users also need to clear HP_SYS_CLKRST_RST_EN_DS, HP_SYS_CLKRST_RST_EN_HMAC, and HP_SYS_CLKRST_RST_EN_ECDSA fields.

Users can start the SHA accelerator with different working modes by configuring registers SHA_START_REG and SHA_DMA_START_REG. For details, please see Table 31.3-1.

Table 31.3-1. SHA Accelerator Working Mode

Working Mode	Configuration Method
Typical SHA	Set SHA_START_REG to 1
DMA-SHA	Set SHA_DMA_START_REG to 1

Users can choose hash algorithms by configuring the SHA_MODE_REG register. For details, please see Table 31.3-2.

Table 31.3-2. SHA Hash Algorithm Selection

Hash Algorithm	SHA_MODE_REG Configuration
SHA-1	0
SHA-224	1
SHA-256	2
SHA-384	3
SHA-512	4
SHA-512/224	5
SHA-512/256	6
SHA-512/*t*	7

Notice:

ESP32-P4's RSA Digital Signature Peripheral (RSA_DS) and HMAC Accelerator (HMAC) modules also call the SHA accelerator when working. Therefore, users cannot access the SHA accelerator when these modules are working.

31.4 Function Description

The SHA accelerator generates the message digest via two steps: Preprocessing and Hash operation.

31.4.1 Preprocessing

Preprocessing consists of three steps: padding the message, parsing the message into message blocks and setting the initial hash value.

31.4.1.1 Padding the Message

The SHA accelerator can only process message blocks of 512 or 1024 bits, depending on the algorithm. Thus, all the messages should be padded to a multiple of 512 bits or 1024 bits before the hash operation.

Suppose that the length of the message M is m bits. Then M shall be padded as introduced below:

- **SHA-1**, **SHA-224** and **SHA-256**
  1. First, append the bit "1" to the end of the message;
  2. Second, append k bits of zeros, where k is the smallest, non-negative solution to the equation $m + 1 + k \equiv 448 \ mod \ 512$;
  3. Last, append the 64-bit block of value equal to the number m expressed using a binary representation.
- **SHA-384**, **SHA-512**, **SHA-512/224**, **SHA-512/256** and **SHA-512/t**
  1. First, append the bit "1" to the end of the message;
  2. Second, append k zero bits, where k is the smallest, non-negative solution to the equation $m + 1 + k \equiv 896 \ mod \ 1024$;
  3. Last, append the 128-bit block of value equal to the number m expressed using a binary representation.

For more details, please refer to FIPS PUB 180-4 Spec > Section "Padding the Message".

31.4.1.2 Parsing the Message

The message and its padding must be parsed into N 512-bit or 1024-bit blocks. During the task, all the message blocks should be written into the SHA_M_*n*_REG.

- For **SHA-1**, **SHA-224** and **SHA-256**:

  The message and its padding are parsed into N 512-bit blocks, $M^{(1)}$, $M^{(2)}$, ..., $M^{(N)}$. Since the 512 bits of the input block may be expressed as sixteen 32-bit words, the first 32 bits of message block i are denoted $M_0^{(i)}$, the next 32 bits are $M_1^{(i)}$, and so on up to $M_{15}^{(i)}$.

  During the task, $M_0^{(i)}$ should be stored in SHA_M_0_REG, $M_1^{(i)}$ stored in SHA_M_1_REG, ..., and $M_{15}^{(i)}$ stored in SHA_M_15_REG.

- For **SHA-384**, **SHA-512**, **SHA-512/224**, **SHA-512/256** and **SHA-512/t**:
  The message and its padding are parsed into N 1024-bit blocks. Since the 1024 bits of the input block may be expressed as sixteen 64-bit words, the first 64 bits of message block i are denoted $M_0^{(i)}$, the next 64 bits are $M_1^{(i)}$, and so on up to $M_{15}^{(i)}$.

  During the task, the most significant 32 bits and the least significant 32 bits of $M_0^{(i)}$ should be stored in SHA_M_0_REG and SHA_M_1_REG, respectively, ..., the most significant 32 bits and the least significant 32 bits of $M_{15}^{(i)}$ should be stored in SHA_M_30_REG and SHA_M_31_REG, respectively.

Note:

For more information about "message block", please refer to FIPS PUB 180-4 Spec > Section "Glossary of Terms and Acronyms".

31.4.1.3 Setting the Initial Hash Value

Before hash task begins for each of the secure hash algorithms, the initial Hash value $H^{(0)}$ must be set based on different algorithms, among which the SHA-1, SHA-224, SHA-256, SHA-384, SHA-512, SHA-512/224, and SHA-512/256 algorithms use the initial Hash values (constant C) stored in the hardware.

However, SHA-512/t requires a distinct initial hash value for each operation for a given value of t. Simply put, SHA-512/t is the generic name for a t-bit hash function based on SHA-512 whose output is truncated to t bits. t is any positive integer without a leading zero such that t<512, and t is not 384. The initial hash value for SHA-512/t for a given value of t can be calculated by performing SHA-512 from hexadecimal representation of the string "SHA-512/t". It's not hard to observe that when determining the initial hash values for SHA-512/t algorithms with different t, the only difference lies in the value of t.

Therefore, we have specially developed the following simplified method to calculate the initial hash value for SHA-512/t:

1. **Generate t_string and t_length**: t_string is a 32-bit data that stores the input message of t. t_length is a 7-bit data that stores the length of the input message. The t_string and t_length are generated in methods described below, depending on the value of t:

   - If $1 <= t <= 9$, then $t_length = 7'h48$ and t_string is padded in the following format:

$8'h30 + 8'ht_0$	$1'b1$	$23'b0$

     where $t_0 = t$.

     For example, if t = 8, then t_0 = 8 and t_string = $32'h38800000$.

   - If $10 <= t <= 99$, then $t_length = 7'h50$ and t_string is padded in the following format:

$8'h30 + 8'ht_1$	$8'h30 + 8'ht_0$	$1'b1$	$15'b0$

     where, $t_0 = t\%10$ and $t_1 = t/10$.

     For example, if t = 56, then t_0 = 6, t_1 = 5, and t_string = $32'h35368000$.

   - If $100 <= t < 512$, then $t_length = 7'h58$ and t_string is padded in the following format:

$8'h30 + 8'ht_2$	$8'h30 + 8'ht_1$	$8'h30 + 8'ht_0$	$1'b1$	$7'b0$

     where, $t_0 = t\%10$, $t_1 = (t/10)\%10$, and $t_2 = t/100$.

     For example, if t = 231, then t_0 = 1, t_1 = 3, t_2 = 2, and t_string = $32'h32333180$.

2. **Initialize relevant registers**: Initialize SHA_T_STRING_REG and SHA_T_LENGTH_REG with the generated t_string and t_length in the previous step.

3. **Obtain initial hash value**: Set the SHA_MODE_REG register to 7. Set the SHA_START_REG register to 1 to start the SHA accelerator. Then poll register SHA_BUSY_REG until the content of this register becomes 0, indicating the calculation of initial hash value is completed.

Please note that the initial value for SHA-512/t can be also calculated according to the Section "5.3.6 SHA-512/t" in FIPS PUB 180-4 Spec, which is performing SHA-512 operation (with its initial hash value set to the result of 8-bitwise XOR operation of C and 0xa5) from the hexadecimal representation of the string "SHA-512/t".

31.4.2 Hash Operation

After the preprocessing, the ESP32-P4 SHA accelerator starts to hash a message M and generates message digest of different lengths, depending on different hash algorithms. As described above, the ESP32-P4 SHA accelerator supports two working modes, which are Typical SHA and DMA-SHA. The operation process for the SHA accelerator under two working modes is described in the following subsections.

31.4.2.1 Typical SHA Mode Process

Usually, the SHA accelerator will process all blocks of a message and produce a message digest before starting the computation of the next message digest.

However, ESP32-P4 SHA also supports optional "interleaved" message digest calculation in Typical SHA mode, which means before SHA completes all blocks of the current message, users are given a chance to insert new computation of another message digest upon the completion of each individual block of the current message.

Specifically, users can read out the message digest from registers SHA_H_*n*_REG after completing part of a message digest calculation, and use the SHA accelerator for a different calculation. After the different calculation completes, users can restore the previous message digest to registers SHA_H_*n*_REG, and resume the accelerator with the previously paused calculation.

Typical SHA Process (except for SHA-512/*t*)

1. Select a hash algorithm.
   - Configure the SHA_MODE_REG register based on Table 31.3-2.
2. Process the current message block.
   - Write the message block in registers SHA_M_*n*_REG.
3. Start the SHA accelerator[1].
   - If this is the first time to execute this step, set the SHA_START_REG register to 1 to start the SHA accelerator. In this case, the accelerator uses the initial hash value stored in hardware for a given algorithm configured in Step 1 to start the calculation;
   - If this is not the first time to execute this step[2], set the SHA_CONTINUE_REG register to 1 to start the SHA accelerator. In this case, the accelerator uses the hash value stored in the SHA_H_*n*_REG register to start calculation.
4. Check the progress of the current message block.
   - Poll register SHA_BUSY_REG until the content of this register becomes 0, indicating the accelerator has completed the calculation for the current message block and now is in the "idle" status [3].
5. Decide if you have more message blocks to process:
   - If yes, please go back to Step 2.
   - Otherwise, please continue.
6. Obtain the message digest.
   - Read the message digest from registers SHA_H_*n*_REG.

Typical SHA Process (SHA-512/t)

1. Select a hash algorithm.
   - Configure the SHA_MODE_REG register to 7 for SHA-512/t.
2. Calculate the initial hash value.
   (a) Calculate t_stiring and t_length and initialize SHA_T_STRING_REG and SHA_T_LENGTH_REG with the generated t_string and t_length. For details, please refer to Section 31.4.1.3.
   (b) Set the SHA_START_REG register to 1 to start the SHA accelerator.
   (c) Poll register SHA_BUSY_REG until the content of this register becomes 0, indicating the calculation of initial hash value is completed.
3. Process the current message block[1].
   - Write the message block in registers SHA_M_n_REG.
4. Start the SHA accelerator
   - Set the SHA_CONTINUE_REG register to 1. In this case, the accelerator uses the hash value stored in the SHA_H_n_REG register to start calculation.
5. Check the progress of the calculation.
   - Poll register SHA_BUSY_REG until the content of this register becomes 0, indicating the accelerator has completed the calculation for the current message block and now is in the "idle" status[3].
6. Decide if you have more message blocks to process:
   - If yes, please go back to Step 3.
   - Otherwise, please continue.
7. Obtain the message digest.
   - Read the message digest from registers SHA_H_n_REG.

Note:

1. In this step, the software can also write the next message block (to be processed) in registers SHA_M_n_REG, if any, while the hardware starts SHA calculation, to save time.
2. You are resuming the SHA accelerator with the previously paused calculation.
3. Here you can decide if you want to insert other calculations. If yes, please go to the process for interleaved calculations for details.

As mentioned above, ESP32-P4 SHA accelerator supports **"interleaving" calculation under the Typical SHA working mode**.

The process to implement interleaved calculation is described below.

1. Prepare to hand the SHA accelerator over for an interleaved calculation by storing the following data of the previous calculation.
   - The selected hash algorithm configured in the SHA_MODE_REG register.
   - The message digest stored in registers SHA_H_n_REG.

2. Perform the interleaved calculation. For the detailed process of the interleaved calculation, please refer to Typical SHA process or DMA-SHA process, depending on the working mode of your interleaved calculation.
3. Prepare to hand the SHA accelerator back to the previously paused calculation by restoring the following data of the previous calculation.
    - Write the previously stored hash algorithm back to register SHA_MODE_REG.
    - Write the previously stored message digest back to registers SHA_H_*n*_REG.
4. Write the next message block from the previous paused calculation in registers SHA_M_*n*_REG, and set the SHA_CONTINUE_REG register to 1 to restart the SHA accelerator with the previously paused calculation.

31.4.2.2 DMA-SHA Mode Process

ESP32-P4 SHA accelerator does not support "interleaving" message digest calculation at the level of individual message blocks when using DMA, which means you cannot insert new calculation before a complete DMA-SHA process (of one or more message blocks) completes. In this case, users who need interleaved operation are recommended to divide the message blocks and perform several DMA-SHA calculations, instead of trying to compute all the messages in one go.

Single DMA-SHA calculation supports up to 63 message blocks.

In contrast to the Typical SHA working mode, when the SHA accelerator is working under the DMA-SHA mode, all data read are completed via GDMA-AXI. Therefore, users are required to configure the GDMA-AXI controller following the description in Chapter 6 *GDMA Controller (GDMA-AHB, GDMA-AXI)*.

DMA-SHA process (except SHA-512/t)

1. Select a hash algorithm.
    - Select a hash algorithm by configuring the SHA_MODE_REG register. For details, please refer to Table 31.3-2.
2. Configure the SHA_INT_ENA_REG register to enable or disable interrupt (Set 1 to enable).
3. Configure the number of message blocks.
    - Write the number of message blocks M to the SHA_DMA_BLOCK_NUM_REG register.
4. Start the DMA-SHA calculation.
    - If the current DMA-SHA calculation follows a previous calculation, firstly write the message digest from the previous calculation to registers SHA_H_*n*_REG, then write 1 to register SHA_DMA_CONTINUE_REG to start SHA accelerator;
    - Otherwise, write 1 to register SHA_DMA_START_REG to start the accelerator.
5. Wait till the completion of the DMA-SHA calculation, which happens when:
    - The content of SHA_BUSY_REG register becomes 0, or
    - An SHA interrupt occurs. In this case, please clear interrupt by writing 1 to the SHA_INT_CLEAR_REG register.

6. Obtain the message digest:
    - Read the message digest from registers SHA_H_*n*_REG.

DMA-SHA process for SHA-512/*t*

1. Select a hash algorithm.
    - Select SHA-512/*t* algorithm by configuring the SHA_MODE_REG register to 7.
2. Configure the SHA_INT_ENA_REG register to enable or disable interrupt (Set 1 to enable).
3. Calculate the initial hash value.
    (a) Calculate t_string and t_length and initialize SHA_T_STRING_REG and SHA_T_LENGTH_REG with the generated t_string and t_length. For details, please refer to Section 31.4.1.3.
    (b) Set the SHA_START_REG register to 1 to start the SHA accelerator.
    (c) Poll register SHA_BUSY_REG until the content of this register becomes 0, indicating the calculation of initial hash value is completed.
4. Configure the number of message blocks.
    - Write the number of message blocks M to the SHA_DMA_BLOCK_NUM_REG register.
5. Start the DMA-SHA calculation.
    - Write 1 to register SHA_DMA_CONTINUE_REG to start the accelerator.
6. Wait till the completion of the DMA-SHA calculation, which happens when:
    - The content of SHA_BUSY_REG register becomes 0, or
    - An SHA interrupt occurs. In this case, please clear interrupt by writing 1 to the SHA_INT_CLEAR_REG register.
7. Obtain the message digest:
    - Read the message digest from registers SHA_H_*n*_REG.

31.4.3 Message Digest

After the hash task completes, the SHA accelerator writes the message digest from the task to registers SHA_H_*n*_REG(*n*: 0 ~ 15). The lengths of the generated message digest are different depending on different hash algorithms. For details, see Table 31.4-4 below:

Table 31.4-4. The Storage and Length of Message Digest from Different Algorithms

Hash Algorithm	Length of Message Digest (in bits)	Storage[1]
SHA-1	160	SHA_H_0_REG ~ SHA_H_4_REG
SHA-224	224	SHA_H_0_REG ~ SHA_H_6_REG
SHA-256	256	SHA_H_0_REG ~ SHA_H_7_REG
SHA-384	384	SHA_H_0_REG ~ SHA_H_11_REG
SHA-512	512	SHA_H_0_REG ~ SHA_H_15_REG
SHA-512/224	224	SHA_H_0_REG ~ SHA_H_6_REG
SHA-512/256	256	SHA_H_0_REG ~ SHA_H_7_REG
SHA-512/*t*[2]	*t*	SHA_H_0_REG ~ SHA_H_*x*_REG

[1] The message digest is stored in registers from most significant bits to the least significant bits, with the first word stored in register SHA_H_0_REG and the second word stored in register SHA_H_1_REG... For details, please see subsection 31.4.1.2.

[2] The registers used for SHA-512/t algorithm depend on the value of t. x+1 indicates the number of 32-bit registers used to store t bits of message digest, so that $x = roundup(t/32)$-1. For example:

- When t = 8, then x = 0, indicating that the 8-bit long message digest is stored in the most significant 8 bits of register SHA_H_0_REG;
- When t = 32, then x = 0, indicating that the 32-bit long message digest is stored in register SHA_H_0_REG;
- When t = 132, then x = 4, indicating that the 132-bit long message digest is stored in registers SHA_H_0_REG, SHA_H_1_REG, SHA_H_2_REG, SHA_H_3_REG, and SHA_H_4_REG.

31.5 Interrupt

ESP32-P4's SHA accelerator can generate the following interrupt signal(s) that will be sent to the Interrupt Matrix.

- SHA_INTR

There are several internal interrupt sources from SHA that can generate the above interrupt signal(s). The interrupt sources from SHA are listed with their trigger conditions and the resulted interrupt signal(s) in Table 31.5-1.

Table 31.5-1. SHA's Internal Interrupt Sources

Internal Interrupt Source	Trigger Condition	Interrupt Signal
SHA_CALC_DONE_INT	Completion of an message digest calculation in DMA-SHA mode	SHA_INTR

Note:

- For definitions of *interrupt*, *interrupt signal*, *interrupt source*, and their correlations, please refer to Chapter 13

Interrupt Matrix > Section 13.2 *Interrupt Terminology in ESP32-P4*.

- Different from the standard interrupt register group, the interrupt register group of SHA only contains the INT_ENA (SHA_INT_ENA_REG) and INT_CLR (SHA_INT_CLEAR_REG) fields, and does not support users to read the INT_RAW and INT_ST fields.

31.6 Register Summary

The addresses in this section are relative to SHA accelerator base address provided in Table 9.3-2 in Chapter 9 *System and Memory*.

The abbreviations given in Column **Access** are explained in Section *Access Types for Registers*.

Name	Description	Address	Access
Control/Configuration Registers			
SHA_MODE_REG	Configures SHA algorithm	0x0000	R/W
SHA_T_STRING_REG	Initial Hash Value calculation factor t_string (for SHA-512/t)	0x0004	R/W
SHA_T_LENGTH_REG	Initial Hash Value calculation factor t_length (for SHA-512/t)	0x0008	R/W
SHA_CONTINUE_REG	Continues SHA operation (only effective in Typical SHA mode)	0x0014	WO
SHA_DMA_START_REG	Starts the SHA accelerator for DMA-SHA operation	0x001C	WO
SHA_START_REG	Starts the SHA accelerator for Typical SHA operation	0x0010	WO
SHA_DMA_CONTINUE_REG	Continues SHA operation (only effective in DMA-SHA mode)	0x0020	WO
SHA_DMA_BLOCK_NUM_REG	Block number register (only effective in DMA-SHA mode)	0x000C	R/W
Status Registers			
SHA_BUSY_REG	Represents if SHA Accelerator is busy or not	0x0018	RO
Interrupt Registers			
SHA_INT_CLEAR_REG	DMA-SHA interrupt clear register	0x0024	WO
SHA_INT_ENA_REG	DMA-SHA interrupt enable register	0x0028	R/W
Data Registers			
SHA_H_0_REG	Hash value	0x0040	R/W
SHA_H_1_REG	Hash value	0x0044	R/W
SHA_H_2_REG	Hash value	0x0048	R/W
SHA_H_3_REG	Hash value	0x004C	R/W
SHA_H_4_REG	Hash value	0x0050	R/W
SHA_H_5_REG	Hash value	0x0054	R/W
SHA_H_6_REG	Hash value	0x0058	R/W
SHA_H_7_REG	Hash value	0x005C	R/W
SHA_H_8_REG	Hash value	0x0060	R/W
SHA_H_9_REG	Hash value	0x0064	R/W
SHA_H_10_REG	Hash value	0x0068	R/W
SHA_H_11_REG	Hash value	0x006C	R/W
SHA_H_12_REG	Hash value	0x0070	R/W
SHA_H_13_REG	Hash value	0x0074	R/W
SHA_H_14_REG	Hash value	0x0078	R/W
SHA_H_15_REG	Hash value	0x007C	R/W
SHA_M_0_REG	Message	0x0080	R/W
SHA_M_1_REG	Message	0x0084	R/W
SHA_M_2_REG	Message	0x0088	R/W
SHA_M_3_REG	Message	0x008C	R/W
SHA_M_4_REG	Message	0x0090	R/W
SHA_M_5_REG	Message	0x0094	R/W

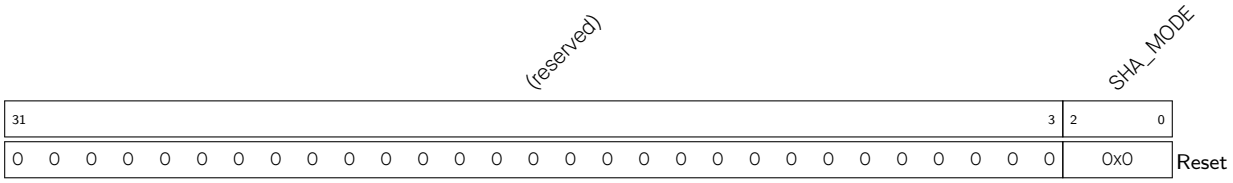
Name	Description	Address	Access
SHA_M_6_REG	Message	0x0098	R/W
SHA_M_7_REG	Message	0x009C	R/W
SHA_M_8_REG	Message	0x00A0	R/W
SHA_M_9_REG	Message	0x00A4	R/W
SHA_M_10_REG	Message	0x00A8	R/W
SHA_M_11_REG	Message	0x00AC	R/W
SHA_M_12_REG	Message	0x00B0	R/W
SHA_M_13_REG	Message	0x00B4	R/W
SHA_M_14_REG	Message	0x00B8	R/W
SHA_M_15_REG	Message	0x00BC	R/W
SHA_M_16_REG	Message	0x00C0	R/W
SHA_M_17_REG	Message	0x00C4	R/W
SHA_M_18_REG	Message	0x00C8	R/W
SHA_M_19_REG	Message	0x00CC	R/W
SHA_M_20_REG	Message	0x00D0	R/W
SHA_M_21_REG	Message	0x00D4	R/W
SHA_M_22_REG	Message	0x00D8	R/W
SHA_M_23_REG	Message	0x00DC	R/W
SHA_M_24_REG	Message	0x00E0	R/W
SHA_M_25_REG	Message	0x00E4	R/W
SHA_M_26_REG	Message	0x00E8	R/W
SHA_M_27_REG	Message	0x00EC	R/W
SHA_M_28_REG	Message	0x00F0	R/W
SHA_M_29_REG	Message	0x00F4	R/W
SHA_M_30_REG	Message	0x00F8	R/W
SHA_M_31_REG	Message	0x00FC	R/W
Version Register			
SHA_DATE_REG	Version control register	0x002C	R/W

31.7 Registers

The addresses in this section are relative to SHA accelerator base address provided in Table 9.3-2 in Chapter 9 *System and Memory*.

For how to program reserved fields, please refer to Section *Programming Reserved Register Field*.

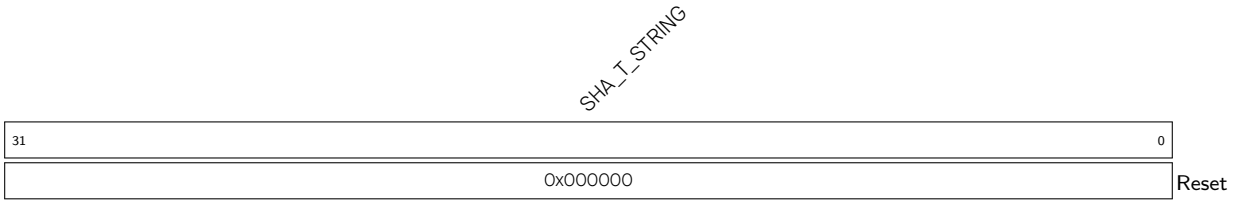
Register 31.1. SHA_MODE_REG (0x0000)



SHA_MODE Configures the SHA algorithm.

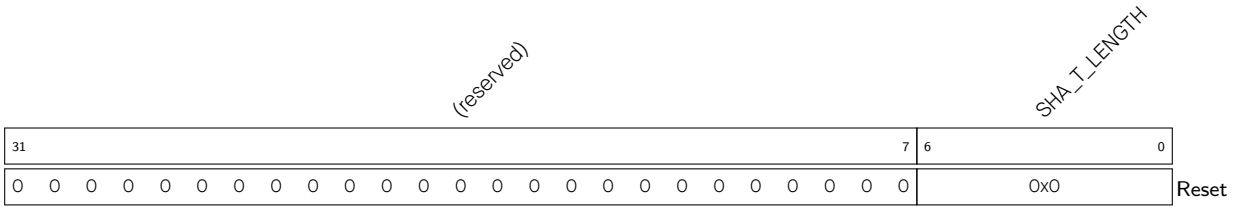
0: SHA-1
1: SHA-224
2: SHA-256
3: SHA-384
4: SHA-512
5: SHA-512/224
6: SHA-512/256
7: SHA-512/t
(R/W)

Register 31.2. SHA_T_STRING_REG (0x0004)



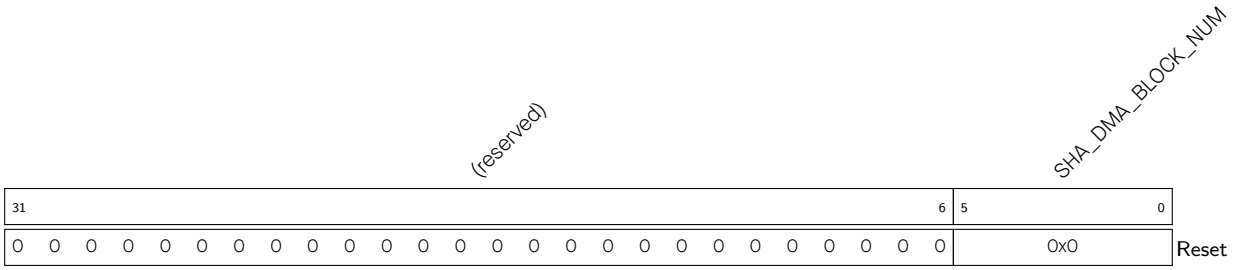
SHA_T_STRING Configures t_string for calculating the initial Hash value for SHA-512/t. (R/W)

Register 31.3. SHA_T_LENGTH_REG (0x0008)



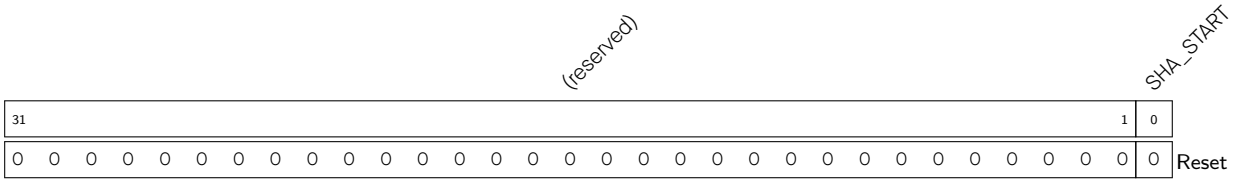
SHA_T_LENGTH Configures t_length for calculating the initial Hash value for SHA-512/t. (R/W)

Register 31.4. SHA_DMA_BLOCK_NUM_REG (0x000C)



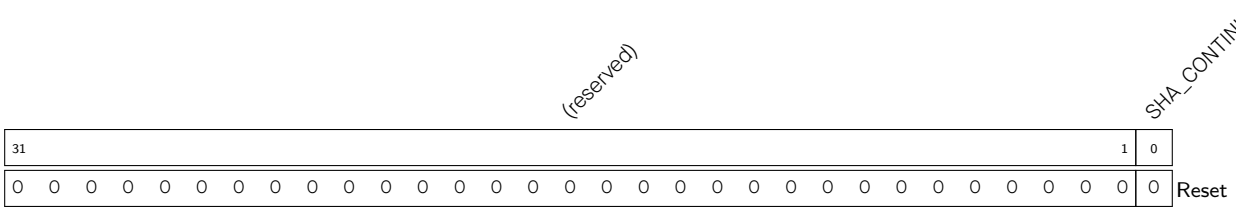
SHA_DMA_BLOCK_NUM Configures the DMA-SHA block number. (R/W)

Register 31.5. SHA_START_REG (0x0010)



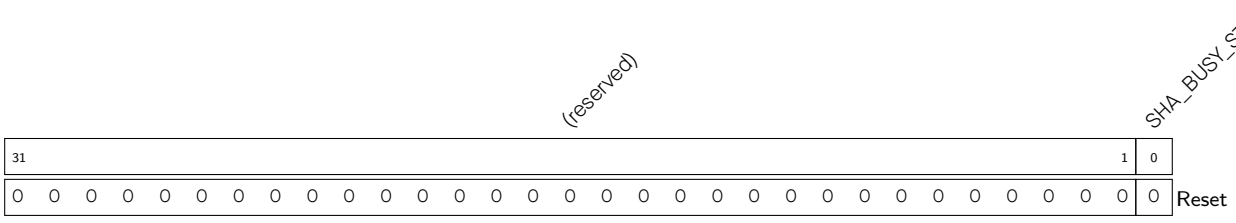
SHA_START Write 1 to start Typical SHA calculation. (WO)

Register 31.6. SHA_CONTINUE_REG (0x0014)



SHA_CONTINUE Write 1 to continue Typical SHA calculation. (WO)

Register 31.7. SHA_BUSY_REG (0x0018)



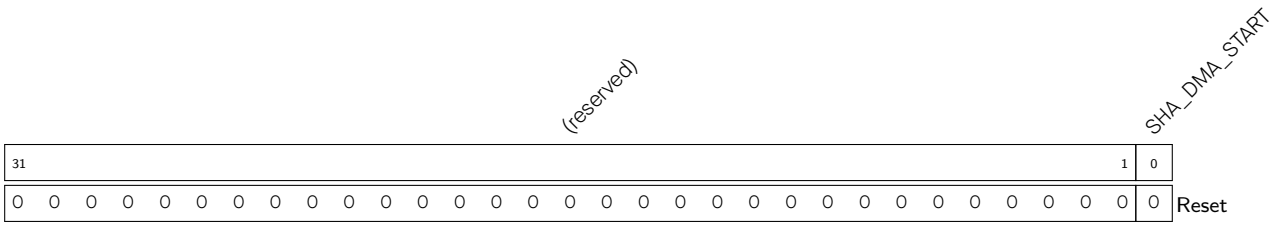
SHA_BUSY_STATE Represents the states of SHA accelerator.

0: idle

1: busy

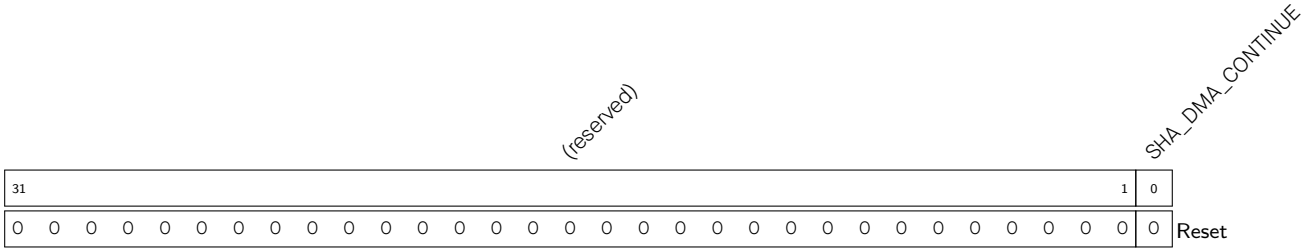
(RO)

Register 31.8. SHA_DMA_START_REG (0x001C)



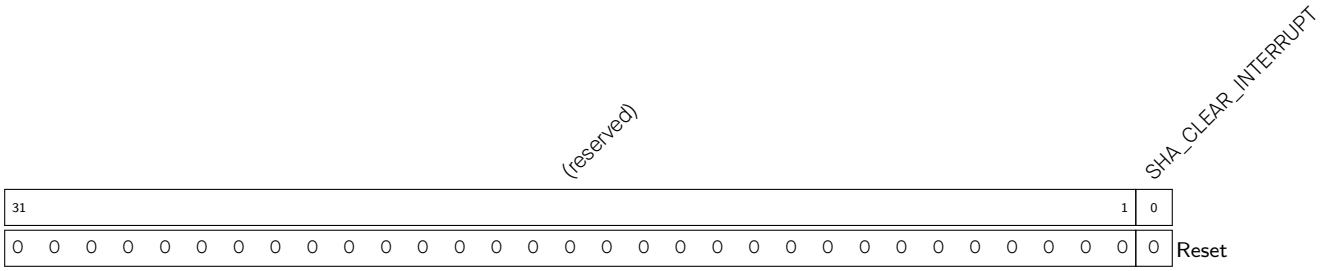
SHA_DMA_START Write 1 to start DMA-SHA calculation. (WO)

Register 31.9. SHA_DMA_CONTINUE_REG (0x0020)



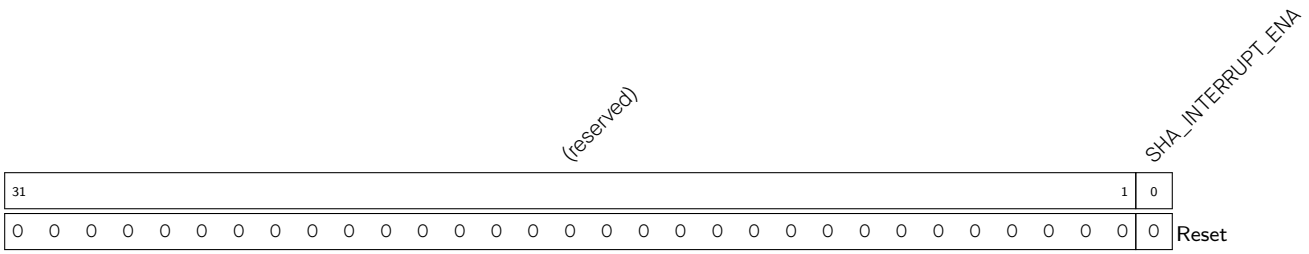
SHA_DMA_CONTINUE Write 1 to continue DMA-SHA calculation. (WO)

Register 31.10. SHA_INT_CLEAR_REG (0x0024)



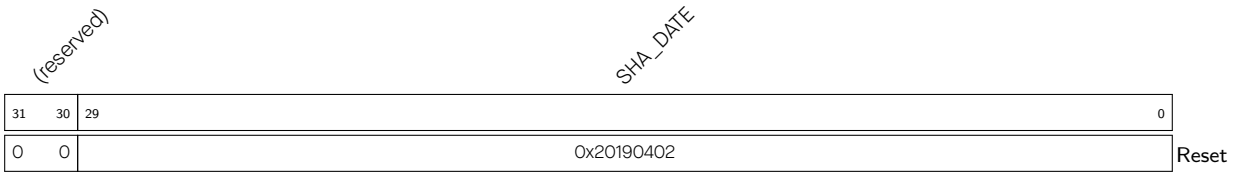
SHA_CLEAR_INTERRUPT Write 1 to clear DMA-SHA interrupt. (WO)

Register 31.11. SHA_INT_ENA_REG (0x0028)



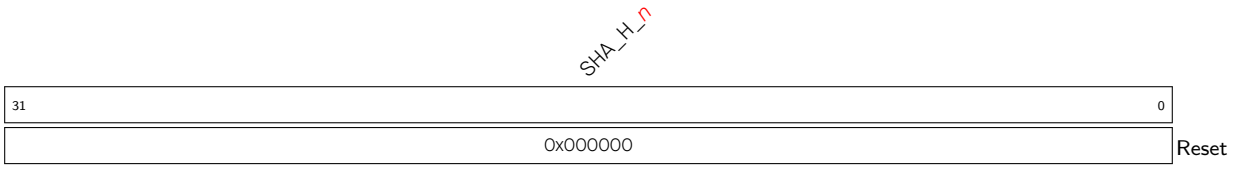
SHA_INTERRUPT_ENA Write 1 to enable DMA-SHA interrupt. (R/W)

Register 31.12. SHA_DATE_REG (0x002C)



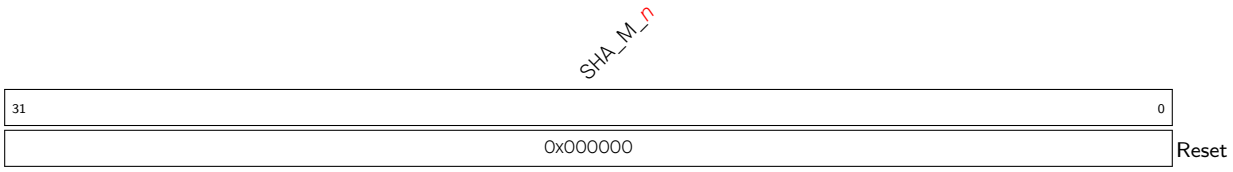
SHA_DATE Version control register. (R/W)

Register 31.13. SHA_H_*n*_REG (*n*: 0-15) (0x0040+4**n*)



SHA_H_*n* Represents the *n*th 32-bit piece of the Hash value. (R/W)

Register 31.14. SHA_M_*n*_REG (*n*: 0-31) (0x0080+4**n*)



SHA_M_*n* Represents the *n*th 32-bit piece of the message. (R/W)

Chapter 32

RSA Digital Signature Peripheral (RSA_DS)

32.1 Overview

The RSA Digital Signature Peripheral (RSA_DS) is used to verify the authenticity and integrity of a message using a cryptographic algorithm. This can be used to validate a device's identity to a server or to check the integrity of a message.

ESP32-P4 includes an RSA Digital Signature Peripheral (RSA_DS) providing hardware acceleration of messages' signatures based on RSA. HMAC is used as the key derivation function (KDF) to output the RSA_DS_KEY using a key stored in eFuse or deployed by the Key Manager as the input key. Subsequently, the RSA_DS peripheral uses RSA_DS_KEY to decrypt the pre-encrypted parameters and calculate the signature. The whole process happens in hardware so that all the keys involved during the calculating process cannot be seen by users, guaranteeing the security of the operation.

32.2 Features

- RSA digital signatures with key length up to 4096 bits
- Encrypted private key data, only decryptable by the RSA_DS peripheral
- SHA-256 digest to protect private key data against tampering by an attacker

32.3 Functional Description

32.3.1 Overview

The RSA_DS peripheral calculates RSA signatures as $Z = X^Y \bmod M$, where Z is the signature, X is the input message, and Y and M are the RSA private key parameters.

Private key parameters are stored in flash as ciphertext. They are decrypted using RSA_DS_KEY which can:

- Be calculated by the RSA_DS peripheral via the HMAC peripheral. The required inputs ($HMAC_KEY$) to generate the key are only stored in eFuse and can only be accessed by the HMAC peripheral. That is to say, the RSA_DS peripheral hardware can decrypt the private key, and the private key in plaintext is never accessed by the software. For more detailed information about eFuse and HMAC peripherals, please refer to Chapter 9.4 *eFuse Controller (EFUSE)* and Chapter 29 *HMAC Accelerator (HMAC)*.
- Deploy from Key Manager module. RSA_DS_KEY must be deployed by the Key Manager in a secured way.

The input message X will be sent directly to the RSA_DS peripheral by the software each time a signature is needed. After the RSA signature operation, the signature Z is read back by the software.

For better understanding, we define some symbols and functions here, which are only applicable to this chapter:

- **1**^{s}: A bit string consisting of s bits with the value of "1".
- **[**x**]**$_s$: A bit string of s bits, in which s is an integer multiple of 8 bits. If x is a number ($x < 2^s$), it is represented in little-endian byte order in the bit string. x may be a variable such as $[Y]_{4096}$ or a hexadecimal constant such as $[0x0C]_8$. If necessary, the value $[x]_t$ can be right-padded with $(s-t)$ number of zeros to reach s bits in length, and finally get $[x]_s$. For example, $[0x05]_8 = 00000101$, $[0x05]_{16} = 0000010100000000$, $[0x0005]_{16} = 0000000000000101$, $[0x13]_8 = 00010011$, $[0x13]_{16} = 0001001100000000$, $[0x0013]_{16} = 0000000000010011$.
- ||: A bit string concatenation operator for joining multiple-bit strings into a longer bit string.

32.3.2 Private Key Operands

Private key operands Y (private key exponent) and M (key modulus) are generated by the user. They have a particular RSA key length (up to 4096 bits). Two additional private key operands are needed: $\overline{r}$ and M'. These two operands are derived from Y and M.

Operands Y, M, $\overline{r}$, and M' are encrypted by the user along with an authentication digest and stored as a single ciphertext C. C is input to the RSA_DS peripheral in this encrypted format, decrypted by the hardware, and then used for RSA signature calculation. A detailed description of how to generate C is provided in Section 32.3.3.

The RSA_DS peripheral supports RSA signature calculation $Z = X^Y \bmod M$, in which the length of operands should be $N = 32 \times x$ where $x \in \{1, 2, 3, \ldots, 128\}$. The bit lengths of arguments Z, X, Y, M, and $\overline{r}$ should be an arbitrary value in N, and all of them in a calculation must be of the same length, while the bit length of M' should always be 32. For more detailed information about RSA calculation, please refer to Section 30.3.2 *Large-Number Modular Exponentiation* in Chapter 30 *RSA Accelerator (RSA)*.

32.3.3 Software Prerequisites

If you want to use the RSA_DS peripheral, the software needs a series of preparations, as shown in Figure 32.3-1. The left side lists preparations required by the software before the hardware starts the RSA signature calculation, while the right side lists the hardware workflow during the entire calculation procedure.

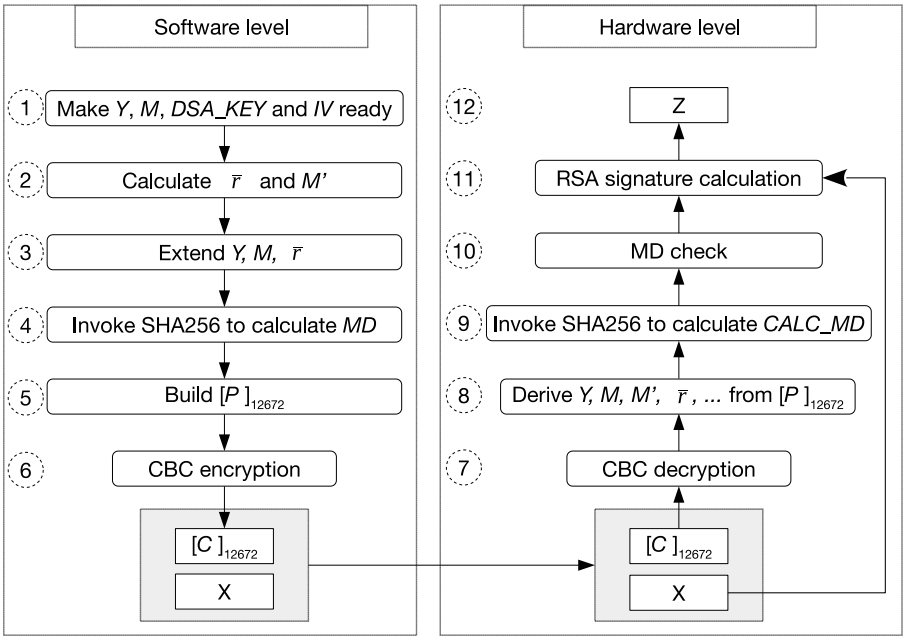


Figure 32.3-1. Software Preparations and Hardware Working Process

Note:

1. The software preparation (left side in Figure 32.3-1) is a one-time operation before any signature is calculated, while the hardware calculation (right side in Figure 32.3-1) repeats for every signature calculation.
2. Software preparation requires configuring the clock reset. For more information, please refer to Chapter 11 *Reset and Clock*.

Users need to follow the steps shown in the left part of Figure 32.3-1 to calculate C. Detailed instructions are as follows:

- **Step 1**: Prepare operands Y and M whose lengths should meet the requirements in Section 32.3.2. Define $[L]_{32} = \frac{N}{32} - 1$ (i.e., for RSA 4096, $[L]_{32}$ == $[0x80\text{-}1]_{32}$). Prepare $[RSA_DS_KEY]_{256}$:
  - when calculated by the RSA_DS peripheral via the HMAC peripheral: RSA_DS_KEY = HMAC-SHA256 ($[HMAC_KEY]_{256}$, 1^{256}).
  - when from Key Manager module: just use the deployed RSA_DS_KEY

  Generate a random $[IV]_{128}$ which should meet the requirements of the AES-CBC block encryption algorithm. For more information on AES, please refer to Chapter 27 *AES Accelerator (AES)*.
- **Step 2**: Calculate $\overline{r}$ and M' based on M.
- **Step 3**: Extend Y, M, and $\overline{r}$ in order to get $[Y]_{4096}$, $[M]_{4096}$, and $[\overline{r}]_{4096}$, respectively. This step is only required for Y, M, and $\overline{r}$ whose length are less than 4096 bits, since their largest length are 4096 bits.
- **Step 4**: Calculate MD authentication code using the SHA-256: $[MD]_{256}$ = SHA256 ($[Y]_{4096}||[M]_{4096}||[\overline{r}]_{4096}||[M']_{32}||[L]_{32}||[IV]_{128}$)
- **Step 5**: Build $[P]_{12672}$ = ($[Y]_{4096}||[M]_{4096}||[\overline{r}]_{4096}||[Box]_{384}$), where $[Box]_{384}$ = ($[MD]_{256}||[M']_{32}||[L]_{32}||[\beta]_{64}$) and $[\beta]_{64}$ is a PKCS#7 padding value, i.e., a $[0x0808080808080808]_{64}$ string composed of 8 bytes (0x80). The purpose of $[\beta]_{64}$ is to make the bit length of P a multiple of 128.

- **Step 6**: Calculate C = $[C]_{12672}$ = AES-CBC-ENC ($[P]_{12672}$, $[RSA_DS_KEY]_{256}$, $[IV]_{128}$), where C is the ciphertext with a length of 1200 bytes. C can also be calculated as C = $[C]_{12672}$ = ($[\widehat{Y}]_{4096}||[\widehat{M}]_{4096}||[\widehat{\overline{r}}]_{4096}||[\widehat{Box}]_{384}$), where $[\widehat{Y}]_{4096}$, $[\widehat{M}]_{4096}$, $[\widehat{\overline{r}}]_{4096}$, $[\widehat{Box}]_{384}$ are the four sub-parameters of C, and correspond to the ciphertext of $[Y]_{4096}$, $[M]_{4096}$, $[\overline{r}]_{4096}$, $[Box]_{384}$ respectively.

32.3.4 RSA_DS Operation at the Hardware Level

The hardware operation is triggered each time a digital signature needs to be calculated. The inputs are the pre-generated private key ciphertext C, a unique message X, and IV.

The RSA_DS operation at the hardware level can be divided into the following three stages:

1. **Decryption: Step 7 and 8 in Figure 32.3-1**

   The decryption process is the inverse of Step 6 in Figure 32.3-1. The RSA_DS peripheral will call the AES accelerator to decrypt C in CBC block mode and get the resulting plaintext. The decryption process can be represented by P = AES-CBC-DEC (C, RSA_DS_KEY, IV), where IV (i.e., $[IV]_{128}$) is defined by the user. $[RSA_DS_KEY]_{256}$ is

   - when calculated by the RSA_DS peripheral via the HMAC peripheral: provided by the HMAC module, derived from $HMAC_KEY$ stored in eFuse, which is not readable by users.
   - when from Key Manager module: RSA_DS_KEY deployed by Key Manager, which is not readable by users.

   With P, the RSA_DS peripheral can derive $[Y]_{4096}$, $[M]_{4096}$, $[\overline{r}]_{4096}$, $[M']_{32}$, $[L]_{32}$, MD authentication code, and the padding value $[\beta]_{64}$. This process is the inverse of Step 5.

2. **Check: Step 9 and 10 in Figure 32.3-1**

   The RSA_DS peripheral will perform two checks: MD check and padding check. The padding check is not shown in Figure 32.3-1, as it happens at the same time as the MD check.

   - MD check: The RSA_DS peripheral calls SHA-256 to calculate the hash value $[CALC_MD]_{256}$ ($[CALC_MD]_{256}$ is calculated the same way and with same parameters as $[MD]_{256}$, see step 4). Then, $[CALC_MD]_{256}$ is compared against the MD authentication code $[MD]_{256}$ from step 4. Only when the two match does the MD check pass.
   - Padding check: The RSA_DS peripheral checks if $[\beta]_{64}$ complies with the aforementioned PKCS#7 format. Only when $[\beta]_{64}$ complies with the format does the padding check pass.

   The RSA_DS peripheral will only perform subsequent operations if MD check passes. If the padding check fails, a warning is generated, but it does not affect the subsequent operations.

3. **Calculation: Step 11 and 12 in Figure 32.3-1**

   The RSA_DS peripheral treats X (input by the user) and Y, M, $\overline{r}$ (decrypted in step 8) as big numbers. With M', all operands to perform X^Y mod M are in place. The operand length is defined by L only. The RSA_DS peripheral will calculate the signed result Z by calling RSA to perform $Z = X^Y$ mod M.

32.3.5 RSA_DS Operation at the Software Level

The software steps below should be followed each time a digital signature needs to be calculated. The inputs are the pre-generated private key ciphertext C, a unique message X, and IV. These software steps trigger

the hardware steps described in Section 32.3.4.

We assume that the software has called the HMAC peripheral and the HMAC peripheral has calculated RSA_DS_KEY based on $HMAC_KEY$.

1. **Prerequisites**: Prepare operands C, X, IV according to Section 32.3.3.
2. **Activate the RSA_DS peripheral**: Write 1 to DSA_SET_START_REG.
3. **Check if RSA_DS_KEY is ready** depending on from where the RSA_DS_KEY comes from:
   - Write 1 to DSA_KEY_SOURCE_REG to select RSA_DS_KEY deployed by the Key Manager. Verify that RSA_DS_KEY has been successfully deployed by checking the register field KEYMNG_KEY_DS_VLD. When its value is 1, it indicates that RSA_DS_KEY has been successfully deployed and is ready.
   - Write 0 to DSA_KEY_SOURCE_REG to select RSA_DS_KEY from HMAC. Poll DSA_QUERY_BUSY_REG until the software reads 0.

   If the software does not read 0 in DSA_QUERY_BUSY_REG after approximately 1 ms, it indicates a problem with HMAC initialization. In such a case, the software can read register DSA_QUERY_KEY_WRONG_REG to get more information:
   - If the software reads 0 in DSA_QUERY_KEY_WRONG_REG, it indicates that the HMAC peripheral has not been called.
   - If the software reads any value from 1 to 15 in DSA_QUERY_KEY_WRONG_REG, it indicates that HMAC was called, but the RSA_DS peripheral did not successfully get RSA_DS_KEY from the HMAC peripheral. This may indicate that the HMAC operation has been interrupted due to a software concurrency problem.
4. **Configure register**: Write the content in the IV block to register DSA_IV_*m*_REG (*m*: 0 ~ 3). For more information on the IV block, please refer to Chapter *27 AES Accelerator (AES)*.
5. **Write X to memory block DSA_X_MEM**: Write X_i ($i \in \{0, 1, \ldots, n-1\}$), where $n = \frac{N}{32}$, to memory block DSA_X_MEM whose capacity is 96 words. Each word can store one base-b digit. The memory block uses the little endian format for storage, i.e., the least significant digit of the operand is in the lowest address. Words in DSA_X_MEM block after the configured length of X (N bits, as described in Section 32.3.2), are ignored.
6. **Write C to corresponding memory blocks**: Write the four sub-parameters of C to corresponding memory blocks:
   - Write $\widehat{Y_i}$ ($i \in \{0, 1, \ldots, 127\}$) to DSA_Y_MEM.
   - Write $\widehat{M_i}$ ($i \in \{0, 1, \ldots, 127\}$) to DSA_M_MEM.
   - Write $\widehat{r}_i$ ($i \in \{0, 1, \ldots, 127\}$) to DSA_RB_MEM.
   - write $\widehat{Box_i}$ ($i \in \{0, 1, \ldots, 11\}$) to DSA_BOX_MEM.

     The capacity of DSA_Y_MEM, DSA_M_MEM, and DSA_RB_MEM is 128 words, whereas the capacity of DSA_BOX_MEM is only 12 words. Each word can store one base-b digit. The memory blocks use the little endian format for storage, i.e., the least significant digit of the operand is in the lowest address.
7. **Start RSA_DS operation**: Write 1 to register DSA_SET_ME_REG.

8. **Wait for the operation to be completed**: Poll register DSA_QUERY_BUSY_REG until the software reads 0.
9. **Query check result**: Read register DSA_QUERY_CHECK_REG and conduct subsequent operations as illustrated below based on the return value:
    - If the value is 0, it indicates that both the padding check and MD check pass. Users can continue to get the signed result Z.
    - If the value is 1, it indicates that the padding check passes but MD check fails. The signed result Z is invalid. The operation will resume directly from Step 11.
    - If the value is 2, it indicates that the padding check fails but the MD check passes. Users can continue to get the signed result Z. But please note that the data does not comply with the aforementioned PKCS#7 padding format, which may not be what you want.
    - If the value is 3, it indicates that both the padding check and MD check fail. In this case, some fatal errors have occurred and the signed result Z is invalid. The operation will resume directly from Step 11.
10. **Read the signed result**: Read the signed result Z_i ($i \in \{0, 1, \ldots, n-1\}$), where $n = \frac{N}{32}$, from memory block DSA_Z_MEM. The memory block stores Z in little-endian byte order.
11. **Exit the operation**: Write 1 to DSA_SET_FINISH_REG, and then poll DSA_QUERY_BUSY_REG until the software reads 0.

After the operation, all the input/output registers and memory blocks are cleared.

32.4 Memory Summary

The addresses in this section are relative to the RSA Digital Signature Peripheral (RSA_DS) base address provided in Table 9.3-2 in Chapter 9 *System and Memory*.

Name	Description	Size (byte)	Starting Address	Ending Address	Access
DSA_Y_MEM	Memory block Y	512	0x0000	0x01FF	WO
DSA_M_MEM	Memory block M	512	0x0200	0x03FF	WO
DSA_RB_MEM	Memory block $\overline{r}$	512	0x0400	0x05FF	WO
DSA_BOX_MEM	Memory block Box	48	0x0600	0x062F	WO
DSA_X_MEM	Memory block X	512	0x0800	0x09FF	WO
DSA_Z_MEM	Memory block Z	512	0x0A00	0x0BFF	RO

32.5 Register Summary

The addresses in this section are relative to the RSA Digital Signature Peripheral (RSA_DS) base address provided in Table 9.3-2 in Chapter 9 *System and Memory*.

The abbreviations given in Column **Access** are explained in Section *Access Types for Registers*.

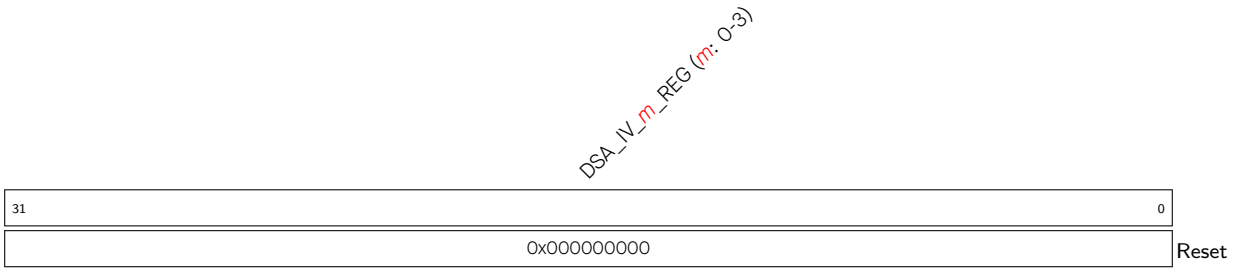
Name	Description	Address	Access
Configuration Registers			
DSA_IV_0_REG	IV block data	0x0630	WO
DSA_IV_1_REG	IV block data	0x0634	WO
DSA_IV_2_REG	IV block data	0x0638	WO
DSA_IV_3_REG	IV block data	0x063C	WO
DSA_KEY_SOURCE_REG	Configures RSA_DS_KEY source	0x0E18	R/W
Status/Control Registers			
DSA_SET_START_REG	Activates the RSA_DS peripheral	0x0E00	WO
DSA_SET_ME_REG	Starts RSA_DS operation	0x0E04	WO
DSA_SET_FINISH_REG	Ends RSA_DS operation	0x0E08	WO
DSA_QUERY_BUSY_REG	Status of the RSA_DS peripheral	0x0E0C	RO
DSA_QUERY_KEY_WRONG_REG	Checks the reason why RSA_DS_KEY is not ready	0x0E10	RO
DSA_QUERY_CHECK_REG	Queries DSA check result	0x0E14	RO
Version control register			
DSA_DATE_REG	Version control register	0x0E20	W/R

32.6 Registers

The addresses in this section are relative to the RSA Digital Signature Peripheral (RSA_DS) base address provided in Table 9.3-2 in Chapter 9 *System and Memory*.

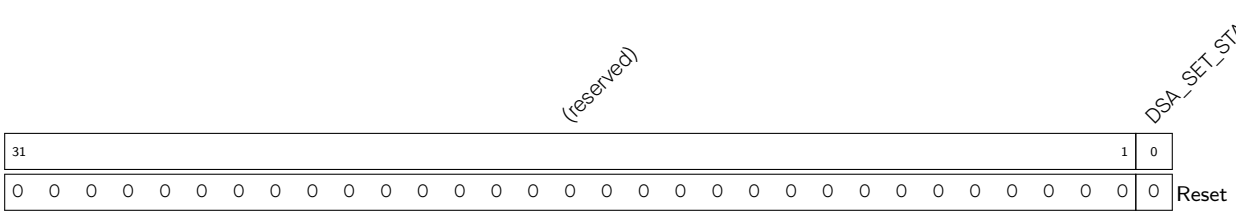
For how to program reserved fields, please refer to Section Programming Reserved Register Field.

Register 32.1. DSA_IV_*m*_REG (*m*: 0-3) (0x0630+4m*)**



DSA_IV_*m*_REG (*m*: 0-3) Writes IV block data. (WO)

Register 32.2. DSA_SET_START_REG (0x0E00)



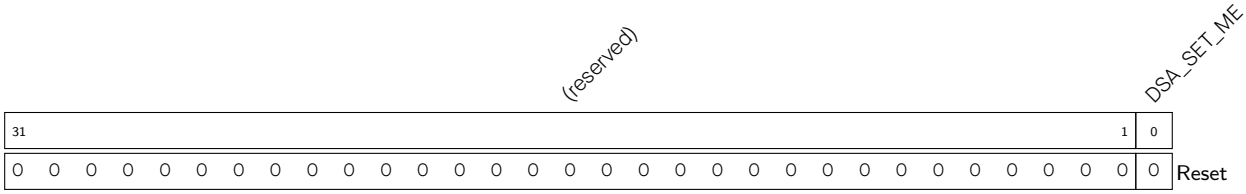
DSA_SET_START Configures whether to activate the RSA_DS peripheral.

0: No effect

1: Activate the RSA_DS peripheral

(WO)

Register 32.3. DSA_SET_ME_REG (0x0E04)



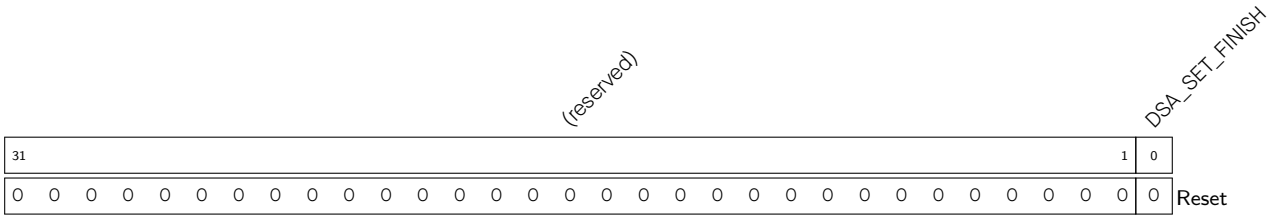
DSA_SET_ME Configures whether to start the RSA_DS operation.

0: No effect

1: Start the RSA_DS operation

(WO)

Register 32.4. DSA_SET_FINISH_REG (0x0E08)



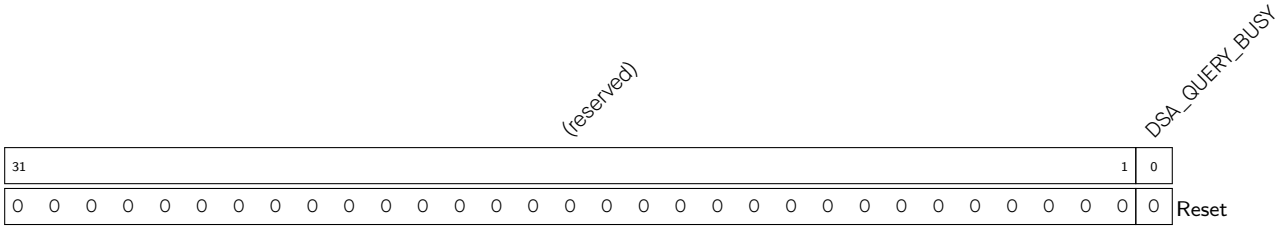
DSA_SET_FINISH Configures whether to end the RSA_DS operation.

0: No effect

1: End the RSA_DS operation

(WO)

Register 32.5. DSA_QUERY_BUSY_REG (0x0E0C)



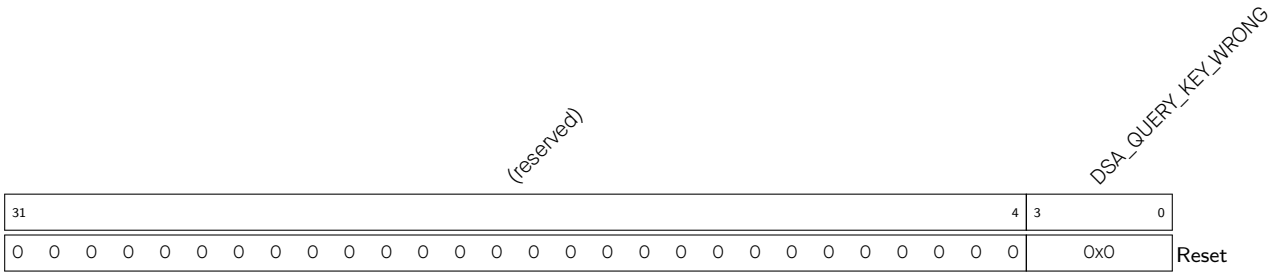
DSA_QUERY_BUSY Represents whether the RSA_DS peripheral is idle.

0: The RSA_DS peripheral is idle.

1: The RSA_DS peripheral is busy.

(RO)

Register 32.6. DSA_QUERY_KEY_WRONG_REG (0x0E10)



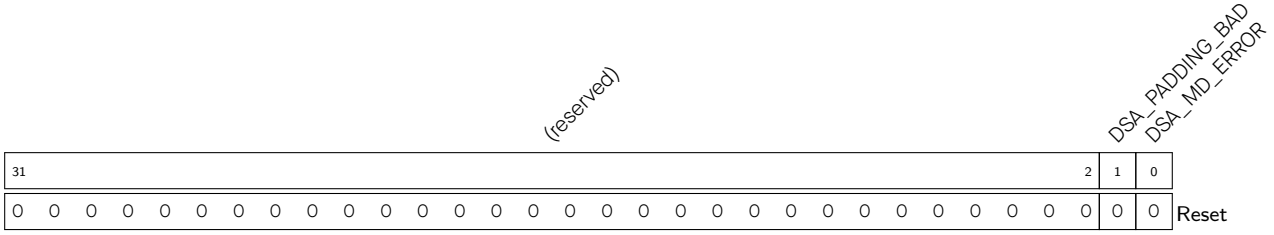
DSA_QUERY_KEY_WRONG Represents the specific problem with HMAC initialization.

0: HMAC is not called.

1-15: HMAC was activated, but the RSA_DS peripheral did not successfully receive RSA_DS_KEY from the HMAC peripheral.

(RO)

Register 32.7. DSA_QUERY_CHECK_REG (0x0E14)



DSA_PADDING_BAD Represents whether the padding check passes.

0: The padding check passes.

1: The padding check fails.

(RO)

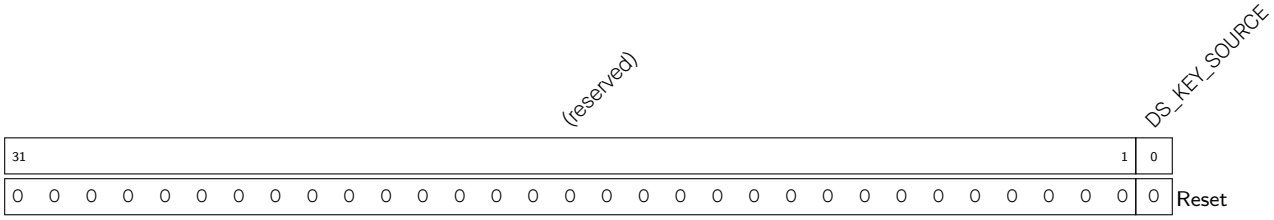
DSA_MD_ERROR Represents whether the MD check passes.

0: The MD check passes.

1: The MD check fails.

(RO)

Register 32.8. DSA_KEY_SOURCE_REG (0x0E18)



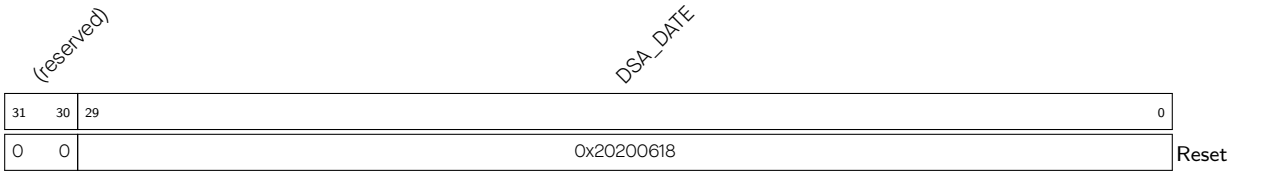
DSA_KEY_SOURCE Represents the source of RSA_DS_KEY.

0: HMAC

1: Key Manager

(R/W)

Register 32.9. DSA_DATE_REG (0x0E20)



DSA_DATE Version control register. (R/W)

Chapter 33

ECDSA Digital Signature Peripheral (ECDSA_DS)

33.1 Introduction

In cryptography, the Elliptic Curve Digital Signature Algorithm (ECDSA) offers a variant of the Digital Signature Algorithm (DSA) which uses elliptic-curve cryptography.

ESP32-P4's ECDSA Digital Signature Peripheral (ECDSA_DS) provides a secure and efficient environment for computing ECDSA signatures. It offers fast computations while ensuring the confidentiality of the signing process to prevent information leakage. This makes it a valuable tool for applications that require high-speed cryptographic operations with strong security guarantees. By using ECDSA_DS, users can be confident that their data is protected without sacrificing performance.

33.2 Features

ESP32-P4's ECDSA_DS module supports:

- Digital signature verification
- Three different elliptic curves, namely, P-192, P-256, and P-384 defined in FIPS 186-5
- Multiple hash algorithms for message hash in the ECDSA_DS operation, including SHA-224, SHA-256, SHA-384, SHA-512, SHA-512-224, and SHA-512-256, defined in FIPS PUB 180-4 Spec
- State-dependent register access control in different operation states to ensure information security

33.3 ECDSA Basics

33.3.1 Domain Parameters

ECDSA uses parameters that define the elliptic curve over a finite field, as well as the generator point and the order of the base point. These parameters are usually referred to as domain parameters, and they are required for key generation, signature generation, and signature verification.

The domain parameters used in ECDSA consist of the following:

- The elliptic curve domain parameters, which include:
  - The prime modulus p, which specifies the size of the finite field over which the elliptic curve is defined.
  - The coefficients a and b, which specify the elliptic curve equation.
  - The base point G on the curve, which can be used to generate public keys.

  - The order n of the base point, which is the number of points on the curve that can be generated by repeatedly adding G to itself.
- The parameters for the hash function, which include:
  - The hash algorithm used to generate a fixed-length hash value from the message being signed.
  - The length of the hash value, which determines the size of the signature.

Together, these parameters define the ECDSA domain.

33.3.2 Key Generation

The process of key generation in ECDSA is described below:

1. Select an elliptic curve domain by defining:
   - the prime modulus p
   - the coefficients a and b
   - the base point G and its order n
2. Generate a private key:
   - A private key d is a randomly chosen integer between 1 and n-1. It is essential to use a secure *Random Number Generator (RNG)* to ensure that the private key is truly random and cannot be predicted or reproduced.
   - The private key is used for signature generation. This key must be kept secret and secure.
3. Compute the public key:
   - The public key Q is calculated as $Q = dG$.
4. Export the public key:
   - The public key Q is typically represented as a pair of coordinates (Qx, Qy) that can be shared with others.

33.3.3 Signature Generation

The ECDSA signature generation process is described below:

1. Select a message m to be signed.
2. Calculate the hash of the message e: e is equal to HASH(m), where HASH is a cryptographic hash function, such as SHA-256.
3. Compute the digest of the message hash z: Let z be the L_n leftmost bits of e, where L_n is the bit length of the base point order n.
4. Select a random number k, which is chosen between 1 and n-1, where n is the order of the base point on the elliptic curve.
5. Compute the signature: The signature is calculated as follows:
   (a) Compute the point (x, y) = kG

(b) Calculate r = x mod n. If r is equal to zero, return to the previous step and select a new random value of k.

(c) Calculate s = k^{-1} * $(z + d * r)$ mod n. If s is equal to zero, return to step 4 and select a new random value of k.

(d) The signature is the pair (r, s).

6. Send the message m and signature (r, s) to the recipient.

33.3.4 Signature Verification

The recipient can then use the public key associated with the private key used for signature generation to verify the signature. The verification process involves checking that the signature was generated using the correct private key and that the signature is valid for the given message.

The ECDSA signature verification process is described below:

1. Obtain public key Q and receive the message m and signature (r, s).
2. Calculate the hash of the message e: e is equal to HASH(m), where HASH is a cryptographic hash function, such as SHA-256.
3. Compute the digest of the message z: Let z be the L_n leftmost bits of e, where L_n is the bit length of the base point order n.
4. Verify the signature: The signature is verified as follows:
   (a) Verify that r and s are integers between 1 and n-1, where n is the order of the base point on the elliptic curve. If either r or s is outside of this range, the signature is invalid.
   (b) Calculate $u_1 = z * s^{-1}$ mod n and $u_2 = r * s^{-1}$ mod n.
   (c) Calculate the point $(x_1, y_1) = u_1 * G + u_2 * Q$, where G is the base point on the elliptic curve, and Q is the public key associated with the private key used for signature generation.
   (d) Verify that $r = x_1$ mod n. If r is not equal to x_1 mod n, the signature is invalid.
5. Accept or reject the signature: If the signature is valid, the recipient can be confident that the message was not tampered with and that it came from the expected sender, thus can accept the message as authentic. Otherwise, the recipient rejects the message as invalid.

33.4 Functional Description

This section describes the details of ESP32-P4's ECDSA_DS module.

33.4.1 ECDSA_DS Working Modes

The ECDSA_DS module integrated in the ESP32-P4 supports Signature Verification mode.

Users can select the elliptic curves used by the ECDSA_DS module by configuring ECDSA_ECC_CURVE according to Table 33.4-1.

Table 33.4-1. ECDSA_DS Elliptic Curve Selection

ECDSA_ECC_CURVE	Elliptic Curve
0	P-192
1	P-256
2	P-384

Users can select the SHA algorithms for message hash by configuring ECDSA_SHA_MODE according to Table 33.4-2.

Table 33.4-2. ECDSA_DS SHA Algorithm

ECDSA_SHA_MODE	SHA Algorithm
1	SHA-224
2	SHA-256
3	SHA-384
4	SHA-512
5	SHA-512-224
6	SHA-512-256
Others	Invalid

Additionally, users can check the working states of the ECDSA_DS module by inquiring the ECDSA_STATE_REG register and comparing the return value against the Table 33.4-3 below.

Table 33.4-3. ECDSA_DS Working State

ECDSA_STATE_REG	State	Description
0	IDLE	Idle or completed operation. Corresponding to IDLE Stage.
1	LOAD	Waiting for users to load information into ECDSA_DS. Corresponding to LOAD Stage.
2	Reserved	–
3	BUSY	In the middle of a hardware operation. Corresponding to PREP, PROC & POST Stage.

33.4.2 Data and Data Block

ESP32-P4's ECDSA_DS module can operate on data of 192, 256, 384, or 512 bits. For example, the data ($D[255:0]$) can be divided into 32-bit data blocks.

Take 256-bit long data as an example, $D[n][31:0](n = 0, 1, \cdots, 7)$. Data blocks with the smaller serial number correspond to the lower binary bits. To be specific:

$$D[255:0] = D[7][31:0], D[6][31:0], D[5][31:0], D[4][31:0], D[3][31:0], D[2][31:0], D[1][31:0], D[0][31:0]$$

33.4.2.1 Writing Data

Writing data means writing data to an ECDSA_DS memory block and using this data as the input to ECDSA_DS. To be specific, writing data to an ECDSA_DS memory block means writing $D[n][31:0]$ to the "starting address of this ECDSA_DS memory block + $4 \times n$". For a 256-bit long data:

- write $D[0]$ to "starting address"
- write $D[1]$ to "starting address + 4"
- ...
- write $D[7]$ to "starting address + 28"

Note:
When storing data, write only the data block of the required length. Do not append a 0 to the most significant bit. For example, for 192-bit data, store only the 192 bits without appending 0.

33.4.2.2 Reading Data

Reading data means reading data from the starting address of an ECDSA_DS memory block and using this data as the output from ECDSA_DS. To be specific, reading data from an ECDSA_DS memory block means reading $D[n][31:0]$ from the "starting address of this ECDSA_DS memory block + $4 \times n$". For a 256-bit long data:

- read $D[0]$ from "starting address"
- read $D[1]$ from "starting address + 4"
- ...
- read $D[7]$ from "starting address + 28"

Note:
When reading data, only the data block of the required length needs to be read. For example, to read 192-bit data, simply read the lower 192 bits (i.e., 6 data blocks).

33.4.2.3 Padding the Message

The SHA accelerator can only process message blocks of 512 bits. Thus, all the messages should be padded to a multiple of 512 bits before the hash operation.

Suppose that the length of the message M is L_M bits. Then M shall be padded as introduced below:

1. First, append the bit "1" to the end of the message;
2. Second, append L_A bits of zeros, where L_A is the smallest, non-negative solution to the equation $L_M + 1 + L_A \equiv 448 \ mod \ 512$;
3. Last, append the 64-bit block of value equal to the number L_M expressed using a binary representation.

For more details, please refer to FIPS PUB 180-4 Spec > Section "Padding the Message".

33.4.2.4 Parsing the Message

The message and its padding must be parsed into N 512-bit message blocks: $M^{(1)}$, $M^{(2)}$, ..., $M^{(N)}$.

Note:

1. For more details about "parsing the message", please refer to FIPS PUB 180-4 Spec > Section "Parsing the Message".
2. For more information on "message block", please refer to FIPS PUB 180-4 Spec > Section "Glossary of Terms and Acronyms".

33.4.3 Security Features

To ensure the security of the operation process, the ECDSA_DS module implements a variety of security functions.

33.4.3.1 High Anti-Attack Performance

ESP32-P4's ECDSA_DS module leverages ECC's enhanced anti-attack performance (refer to Section 28.4.3 *Enhancing Anti-Attack Performance*) every time it performs an operation. This means that each time a signature is generated and verified by the ECDSA_DS:

- The execution latency is constant for a given operation.
- Power consumption variations are minimized.

This provides ESP32-P4's ECDSA_DS module strong anti-attack performance.

33.4.3.2 State-Dependent Register Access Control

ESP32-P4's ECDSA_DS module has implemented a state-dependent register access control mechanism to prevent any possibility of key theft by tampering with the configuration or accessing the data during the operation.

By implementing the state-dependent register access control, the accesses for ECDSA_DS registers are designed to vary in different states. For example, ECDSA_CONF_REG is only available for reading and writing when ECDSA_DS is in the IDLE state. In this way, the configuration information is protected from reading or writing when ECDSA_DS is in other states, such as LOAD and BUSY. For details about all ECDSA_DS working states, please refer to Table 33.4-3.

For detailed information on the state-dependent access control of each ECDSA_DS register, please refer to Section Register Summary.

33.4.3.3 Hardware Occupation

During the operation, the following hardware modules will be occupied by ESP32-P4's ECDSA_DS module:

- SHA Accelerator
- ECC Accelerator

Among them, the SHA accelerator will be released when ECDSA_SHA_RELEASE_INT is triggered, while the ECC accelerator will be occupied during the whole ECDSA_DS operation.

Note:

Hardware occupation is a mechanism to protect multiplexed modules and storage space. When a module is hardware occupied, the user will fail to:

- read or write data to the module's registers or memories.
- disable the module clock.
- reset the module.

After hardware occupation is finished, the occupied module will be automatically reset. In addition, when the user performs a software reset to the master module, all the occupied modules will be reset at the same time.

33.5 Programming Procedures

33.5.1 ECDSA_DS Process

The overall ECDSA_DS process consists of the following five stages.

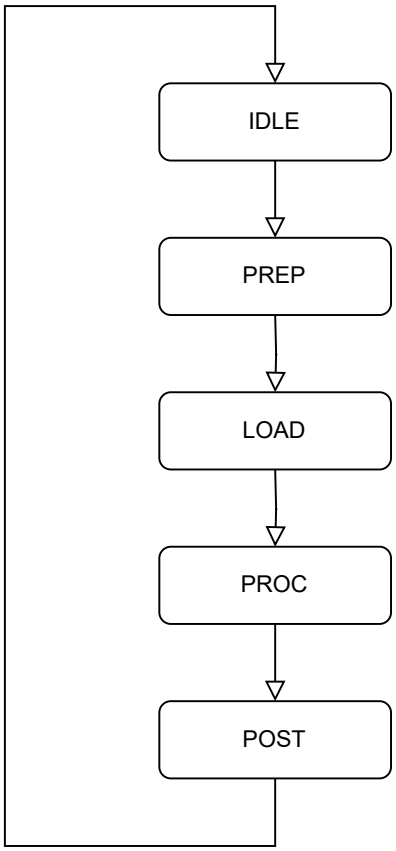


Figure 33.5-1. ECDSA_DS Process

The detailed programming procedures of each stage are described in the following sections.

33.5.1.1 IDLE Stage

In the IDLE stage:

1. Configure the static parameters including eFuse bits ECDSA_KEY, i.e., the value of private key d in ECDSA. To correctly configure the key value in eFuse, user need to write the key value in KEY*n* (n = 0 ~ 5), and set the corresponding EFUSE_KEY_PURPOSE_*n* as ECDSA_KEY. Please refer to Chapter 9.4 *eFuse Controller (EFUSE)* for more detailed configuration steps.
2. Configure the configuration register, including the following fields:
   (a) ECDSA_ECC_CURVE: Choose the elliptic curve of the ECDSA_DS module.
   (b) ECDSA_SOFTWARE_SET_Z: Choose to use direct input z.
3. Configure the register field ECDSA_START to enter the PREP stage.

33.5.1.2 PREP Stage

In the PREP stage, ECDSA_STATE_REG is BUSY, and the ECDSA_DS performs preparation.

Wait till the PREP stage to end by polling ECDSA_BUSY until it is not BUSY. Then the ECDSA_DS will automatically enter the LOAD stage.

33.5.1.3 LOAD Stage

In the LOAD stage:

1. Provide input z into the ECDSA_DS using one of the following options:
   - Direct input z: write z to ECDSA_Z_MEM.
   - ECDSA_DS SHA interface: generate z from the message. For details, please refer to Section 33.5.1.6.
2. Provide signature and public key required by signature verification:
   (a) write signature (r, s) to ECDSA_R_MEM and ECDSA_S_MEM.
   (b) write public key (Qx, Qy) to ECDSA_QX_MEM and ECDSA_QY_MEM.
3. Write 1 to ECDSA_LOAD_DONE, indicating that the configuration is done. Then ECDSA_DS will automatically enter the PROC stage.

33.5.1.4 PROC Stage

In the PROC stage, ECDSA_STATE_REG is BUSY, and the ECDSA_DS performs ECDSA signature verification.

Wait till the PROC stage to end by polling ECDSA_BUSY until it is not BUSY. Then the ECDSA_DS will automatically enter the POST stage.

33.5.1.5 POST Stage

In the POST stage, ECDSA_STATE_REG is BUSY, and the ECDSA_DS performs wrap-up work of the operation.

Wait till the POST stage to end by polling ECDSA_BUSY until it is not BUSY. Then the ECDSA_DS will automatically return to the IDLE stage.

33.5.1.6 ECDSA_DS SHA Interface

ESP32-P4's ECDSA_DS module can automatically execute hash operation and generates z based on a direct input message.

For message hash, the ECDSA_DS module supports SHA algorithms SHA-224 (only valid when P-192 is selected as the elliptic curve) and SHA-256.

To use the ECDSA_DS SHA interface, complete the following steps:

1. Pad the message by following the steps described in Section 33.4.2.3.
2. Parse the message and its padding into message blocks. See details in Section 33.4.2.4.
3. Process the current message block.
   - Write the current message block into ECDSA_MEM_M.
4. Start the ECDSA_DS SHA interface[1].
   - If this is the first time to execute this step, write 1 to ECDSA_SHA_START to start the ECDSA_DS SHA interface;
   - If this is not the first time to execute this step[2], write 1 to ECDSA_SHA_CONTINUE to continue the operation.
5. Check the progress of the current message block processing by polling.
   - Poll register ECDSA_SHA_BUSY until it's IDLE, indicating that the interface has completed the operation for the current message block.
6. Process the next message block:
   - If there is message block to process, go back to Step 3.
   - If there is no more message blocks, exit.

Note:

1. In this step, the software can also write the next message block to be processed, if any, in register ECDSA_MEM_M, while the interface starts SHA operation, to save time.
2. You are resuming the ECDSA_DS SHA interface with the previously paused operation.

33.5.2 Clocks and Resets

ESP32-P4's ECDSA_DS module has one clock module and one reset module. It is activated by setting the HP_SYS_CLKRST_CRYPTO_ECDSA_CLK_EN bit in the HP_SYS_CLKRST_PERI_CLK_CTRL25_REG register and clearing the HP_SYS_CLKRST_RST_EN_ECDSA bit in the HP_SYS_CLKRST_HP_RST_EN2_REG register. For details on how to configure the ECDSA_DS clock and reset, please refer to Chapter 11 *Reset and Clock*.

33.5.3 Interrupts

ESP32-P4's ECDSA_DS module can generate the interrupt signal ECDSA_INTR and send it to Interrupt Matrix.

The ECDSA_DS module has four interrupt sources that can generate the ECDSA_INTR interrupt signal as follows. Each interrupt source can be configured by a common set of registers that are described in Section *Interrupt Configuration Registers*.

- ECDSA_PREP_DONE_INT: triggered on the completion of the PREP stage. This interrupt source is configured by the following registers:
  - ECDSA_PREP_DONE_INT_RAW: stores the raw interrupt status of ECDSA_PREP_DONE_INT.
  - ECDSA_PREP_DONE_INT_ST: indicates the status of the ECDSA_PREP_DONE_INT interrupt. This field is generated by enabling/disabling the ECDSA_PREP_DONE_INT_RAW field.
  - ECDSA_PREP_DONE_INT_ENA: enables/disables the ECDSA_PREP_DONE_INT interrupt.
  - ECDSA_PREP_DONE_INT_CLR: set this bit to clear the ECDSA_PREP_DONE_INT interrupt status. By setting this bit to 1, fields ECDSA_PREP_DONE_INT_RAW and ECDSA_PREP_DONE_INT_ST will be cleared.
- ECDSA_PROC_DONE_INT: triggered on the completion of the PROC stage. This interrupt source is configured by the following registers:
  - ECDSA_PROC_DONE_INT_RAW: stores the raw interrupt status of ECDSA_PROC_DONE_INT.
  - ECDSA_PROC_DONE_INT_ST: indicates the status of the ECDSA_PROC_DONE_INT interrupt. This field is generated by enabling/disabling the ECDSA_PROC_DONE_INT_RAW field.
  - ECDSA_PROC_DONE_INT_ENA: enables/disables the ECDSA_PROC_DONE_INT interrupt.
  - ECDSA_PROC_DONE_INT_CLR: set this bit to clear the ECDSA_PROC_DONE_INT interrupt status. By setting this bit to 1, fields ECDSA_PROC_DONE_INT_RAW and ECDSA_PROC_DONE_INT_ST will be cleared.
- ECDSA_POST_DONE_INT: triggered on the completion of the POST stage. This interrupt source is configured by the following registers:
  - ECDSA_POST_DONE_INT_RAW: stores the raw interrupt status of ECDSA_POST_DONE_INT.
  - ECDSA_POST_DONE_INT_ST: indicates the status of the ECDSA_POST_DONE_INT interrupt. This field is generated by enabling/disabling the ECDSA_POST_DONE_INT_RAW field.
  - ECDSA_POST_DONE_INT_ENA: enables/disables the ECDSA_POST_DONE_INT interrupt.
  - ECDSA_POST_DONE_INT_CLR: set this bit to clear the ECDSA_POST_DONE_INT interrupt status. By setting this bit to 1, fields ECDSA_POST_DONE_INT_RAW and ECDSA_POST_DONE_INT_ST will be cleared.
- ECDSA_SHA_RELEASE_INT: triggered when SHA is released. This interrupt source is configured by the following registers:
  - ECDSA_SHA_RELEASE_INT_RAW: stores the raw interrupt status of ECDSA_SHA_RELEASE_INT.

- ECDSA_SHA_RELEASE_INT_ST: indicates the status of the ECDSA_SHA_RELEASE_INT interrupt. This field is generated by enabling/disabling the ECDSA_SHA_RELEASE_INT_RAW field via ECDSA_SHA_RELEASE_INT_ENA.
- ECDSA_SHA_RELEASE_INT_ENA: enables/disables the ECDSA_SHA_RELEASE_INT interrupt.
- ECDSA_SHA_RELEASE_INT_CLR: set this bit to clear the ECDSA_SHA_RELEASE_INT interrupt status. By setting this bit to 1, fields ECDSA_SHA_RELEASE_INT_RAW and ECDSA_SHA_RELEASE_INT_ST will be cleared.

Note:
For definitions of *interrupt*, *interrupt signal*, *interrupt source*, and their correlations, please refer to Chapter 13 *Interrupt Matrix* > Section 13.2 *Interrupt Terminology in ESP32-P4*.

33.6 Memory Blocks

ECDSA_DS's memory blocks store input data and output data of the operation.

Table 33.6-1. ECDSA_DS Memory Blocks

Memory	Size (byte)	Starting Address*	Ending Address*	Access
ECDSA_MEM_M	64	0x280	0x2BF	R/W
ECDSA_MEM_R	32	0xA00	0xA1F	R/W
ECDSA_MEM_S	32	0xA20	0xA3F	R/W
ECDSA_MEM_Z	32	0xA40	0xA5F	R/W
ECDSA_MEM_Qx	32	0xA60	0xA7F	R/W
ECDSA_MEM_Qy	32	0xA80	0xA9F	R/W

* Address offset related to the ECDSA_DS base address is provided in Table 9.3-2 in Chapter 9 *System and Memory*.

33.7 Register Summary

The addresses in this section are relative to the ECDSA_DS base address provided in Table 9.3-2 in Chapter 9 *System and Memory*.

To enhance security, ECDSA_DS registers have different read and write permissions in different operating states. The following is the abbreviation and corresponding relationship of each state:

- PI: IDLE state, corresponding to IDLE Stage.
- PL: LOAD state, corresponding to LOAD Stage.
- PB: BUSY state, corresponding to PREP, PROC and POST Stage.

Other abbreviations given in Column **Access** are explained in Section Access Types for Registers.

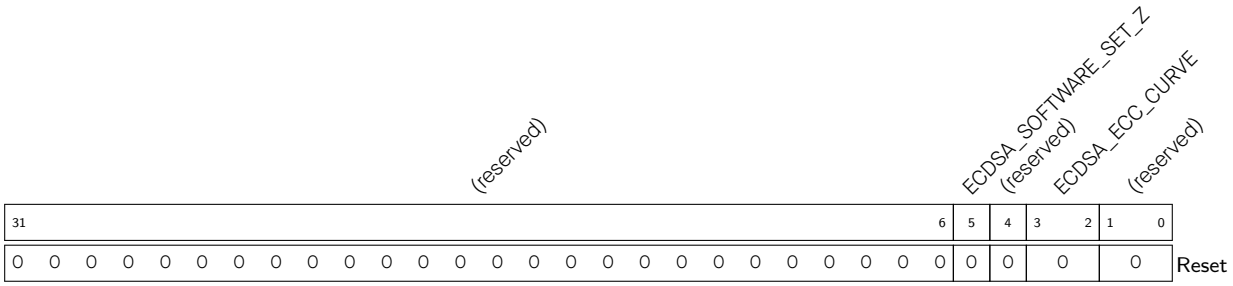
Name	Description	Address	Access			
			PI	PL	PG	P
Data Memory	See Table 33.6-1.					
Configuration Registers						
ECDSA_CONF_REG	ECDSA_DS configuration register	0x0004	R/W	N/A		
ECDSA_START_REG	ECDSA_DS start register	0x001C	WT			N
Clock and Reset Register						
ECDSA_CLK_REG	ECDSA_DS clock gate register	0x0008	R/W	N/A		
Interrupt Registers						
ECDSA_INT_RAW_REG	ECDSA_DS interrupt raw register	0x000C	RO/WTC/SS			
ECDSA_INT_ST_REG	ECDSA_DS interrupt status register	0x0010	RO			
ECDSA_INT_ENA_REG	ECDSA_DS interrupt enable register	0x0014	R/W			
ECDSA_INT_CLR_REG	ECDSA_DS interrupt clear register	0x0018	WT			
Status Registers						
ECDSA_STATE_REG	ECDSA_DS state register	0x0020	RO			
Result Register						
ECDSA_RESULT_REG	ECDSA_DS result register	0x0024	RO/SS			N
SHA Registers						
ECDSA_SHA_MODE_REG	ECDSA_DS controlling SHA register (Hash algorithm)	0x0200	N/A	R/W	N/A	
ECDSA_SHA_START_REG	ECDSA_DS controlling SHA register (operation)	0x0210	N/A	WT	N/A	
ECDSA_SHA_CONTINUE_REG	ECDSA_DS controlling SHA register (operation)	0x0214	N/A	WT	N/A	
ECDSA_SHA_BUSY_REG	ECDSA_DS controlling SHA status register	0x0218	N/A	RO	N/A	
Version Register						
ECDSA_DATE_REG	Version control register	0x00FC	R/W	N/A		

33.8 Registers

The addresses in this section are relative to the ECDSA_DS base address provided in Table 9.3-2 in Chapter 9 *System and Memory*.

For how to program reserved fields, please refer to Section *Programming Reserved Register Field*.

Register 33.1. ECDSA_CONF_REG (0x0004)



ECDSA_ECC_CURVE Configures the elliptic curve used.

0: P-192

1: P-256

2: P-384

3: Invalid

(R/W)

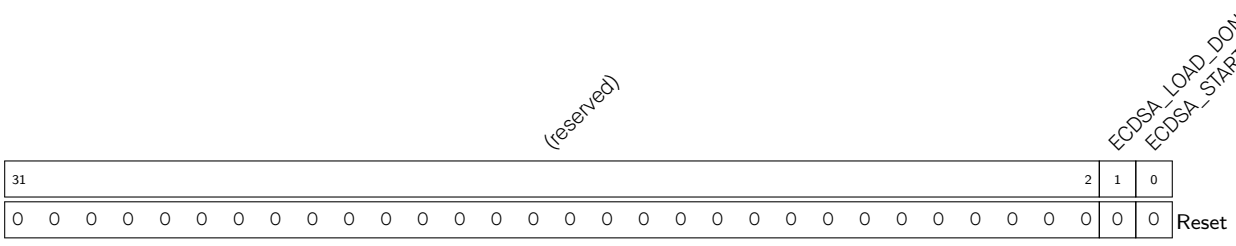
ECDSA_SOFTWARE_SET_Z Configures how the parameter z is set.

0: Generated from SHA result

1: Written by software

(R/W)

Register 33.2. ECDSA_START_REG (0x001C)



ECDSA_START Configures whether to start the ECDSA_DS operation. This bit will be self-cleared after configuration.

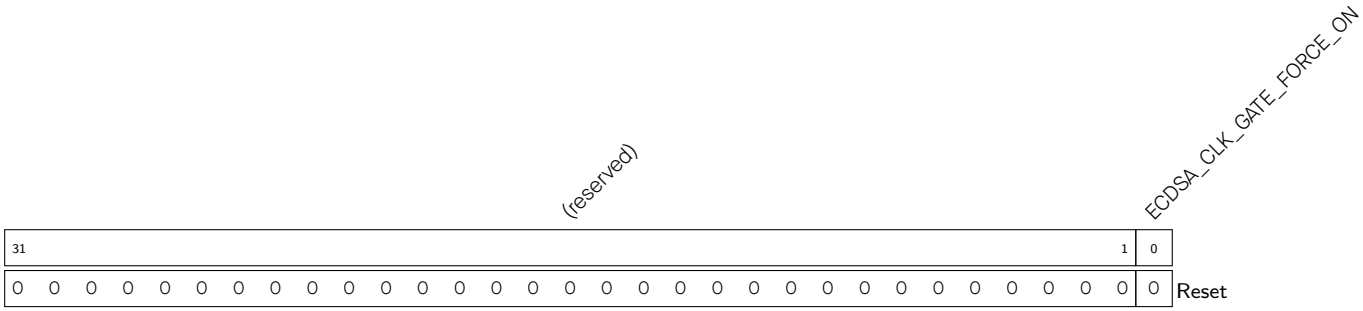
0: No effect

1: Start the ECDSA_DS operation

(WT)

ECDSA_LOAD_DONE Write 1 to generate a signal indicating the ECDSA_DS's LOAD operation is done. This bit will be self-cleared after configuration. (WT)

Register 33.3. ECDSA_CLK_REG (0x0008)



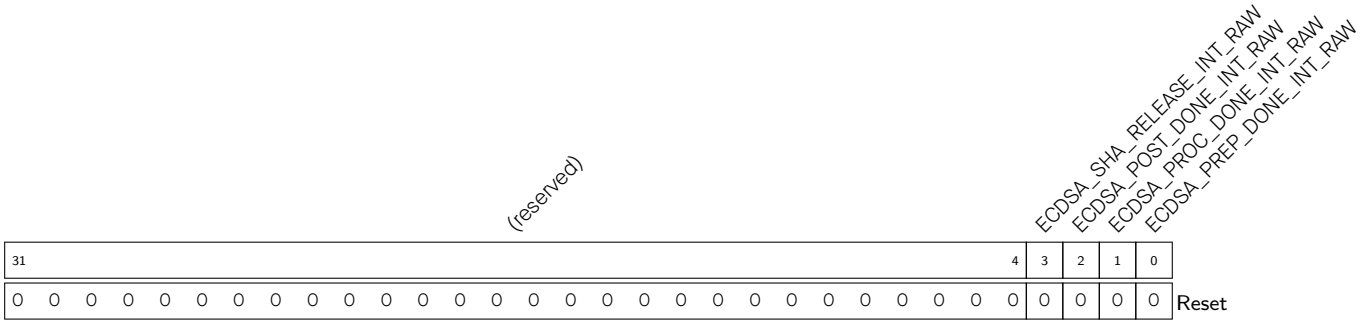
ECDSA_CLK_GATE_FORCE_ON Configures whether to force on ECC memory clock gate.

0: No effect

1: Force on

(R/W)

Register 33.4. ECDSA_INT_RAW_REG (0x000C)



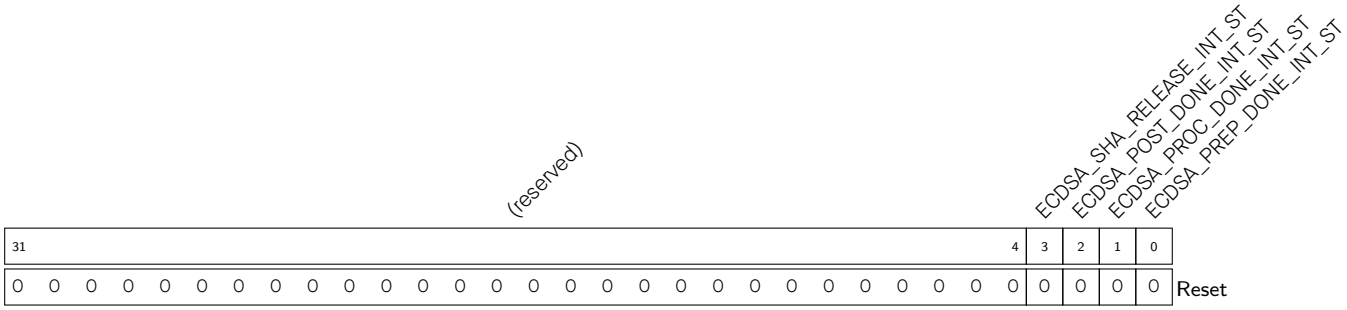
ECDSA_PREP_DONE_INT_RAW The raw interrupt status of the ECDSA_PREP_DONE_INT interrupt. (R/SS/WTC)

ECDSA_PROC_DONE_INT_RAW The raw interrupt status of the ECDSA_PROC_DONE_INT interrupt. (R/SS/WTC)

ECDSA_POST_DONE_INT_RAW The raw interrupt status of the ECDSA_POST_DONE_INT interrupt. (R/SS/WTC)

ECDSA_SHA_RELEASE_INT_RAW The raw interrupt status of the ECDSA_SHA_RELEASE_INT interrupt. (R/SS/WTC)

Register 33.5. ECDSA_INT_ST_REG (0x0010)



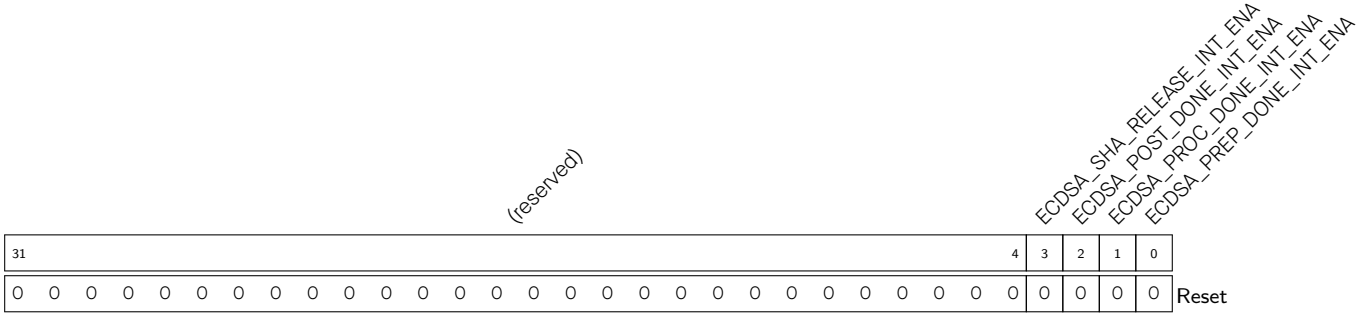
ECDSA_PREP_DONE_INT_ST The masked interrupt status of the ECDSA_PREP_DONE_INT interrupt. (RO)

ECDSA_PROC_DONE_INT_ST The masked interrupt status of the ECDSA_PROC_DONE_INT interrupt. (RO)

ECDSA_POST_DONE_INT_ST The masked interrupt status of the ECDSA_POST_DONE_INT interrupt. (RO)

ECDSA_SHA_RELEASE_INT_ST The masked interrupt status of the ECDSA_SHA_RELEASE_INT interrupt. (RO)

Register 33.6. ECDSA_INT_ENA_REG (0x0014)



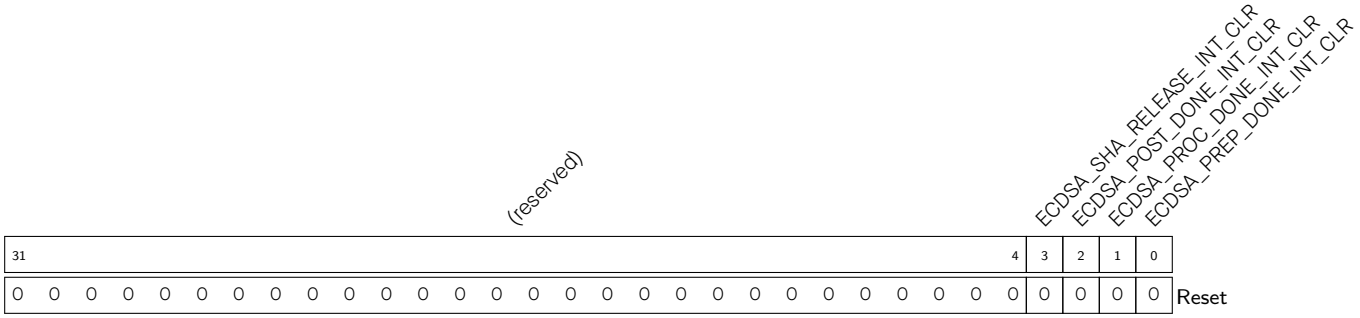
ECDSA_PREP_DONE_INT_ENA Write 1 to enable the ECDSA_PREP_DONE_INT interrupt. (R/W)

ECDSA_PROC_DONE_INT_ENA Write 1 to enable the ECDSA_PROC_DONE_INT interrupt. (R/W)

ECDSA_POST_DONE_INT_ENA Write 1 to enable the ECDSA_POST_DONE_INT interrupt. (R/W)

ECDSA_SHA_RELEASE_INT_ENA Write 1 to enable the ECDSA_SHA_RELEASE_INT interrupt. (R/W)

Register 33.7. ECDSA_INT_CLR_REG (0x0018)



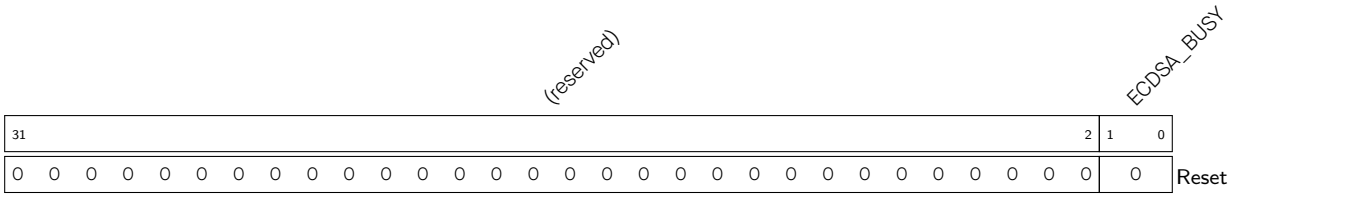
ECDSA_PREP_DONE_INT_CLR Write 1 to clear the ECDSA_PREP_DONE_INT interrupt. (WT)

ECDSA_PROC_DONE_INT_CLR Write 1 to clear the ECDSA_PROC_DONE_INT interrupt. (WT)

ECDSA_POST_DONE_INT_CLR Write 1 to clear the ECDSA_POST_DONE_INT interrupt. (WT)

ECDSA_SHA_RELEASE_INT_CLR Write 1 to clear the ECDSA_SHA_RELEASE_INT interrupt. (WT)

Register 33.8. ECDSA_STATE_REG (0x0020)



ECDSA_BUSY Represents the working state of the ECDSA_DS module.

0: IDLE

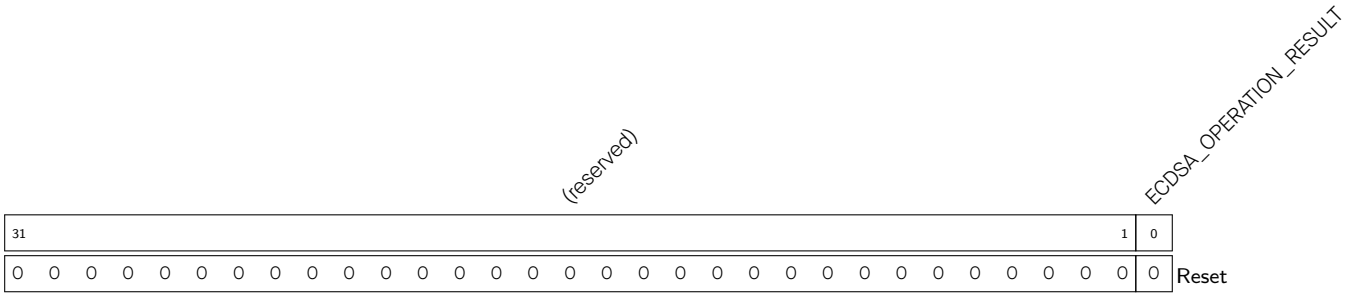
1: LOAD

2: Reserved

3: BUSY

(RO)

Register 33.9. ECDSA_RESULT_REG (0x0024)



ECDSA_OPERATION_RESULT Indicates if the ECDSA_DS operation is successful.

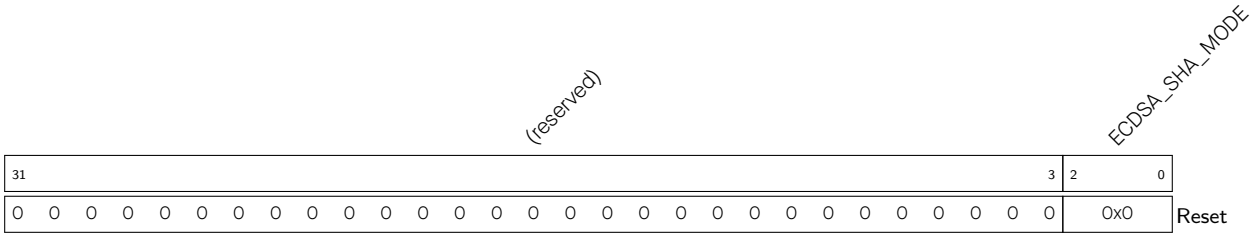
0: Not successful

1: Successful

Only valid when the ECDSA_DS operation is done.

(RO/SS)

Register 33.10. ECDSA_SHA_MODE_REG (0x0200)



ECDSA_SHA_MODE Configures SHA algorithms for message hash.

1: SHA-224

2: SHA-256

3: SHA-384

4: SHA-512

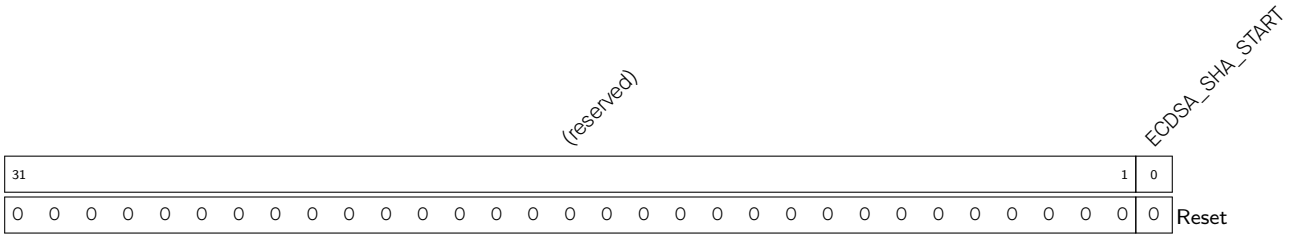
5: SHA-512-224

6: SHA-512-256

Others: invalid

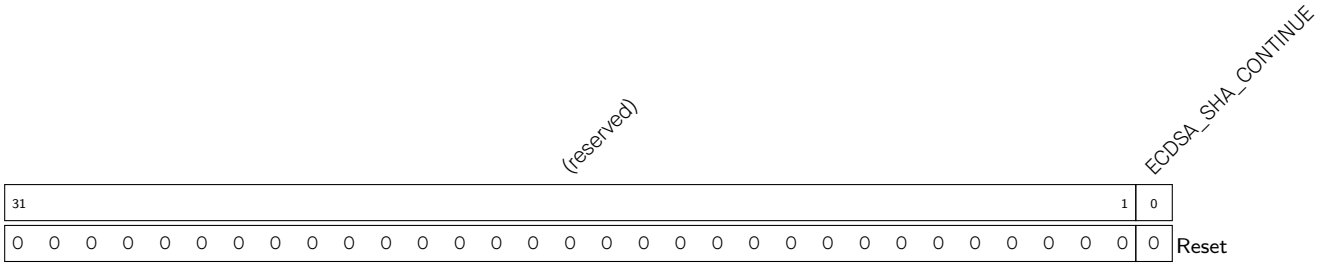
(R/W)

Register 33.11. ECDSA_SHA_START_REG (0x0210)



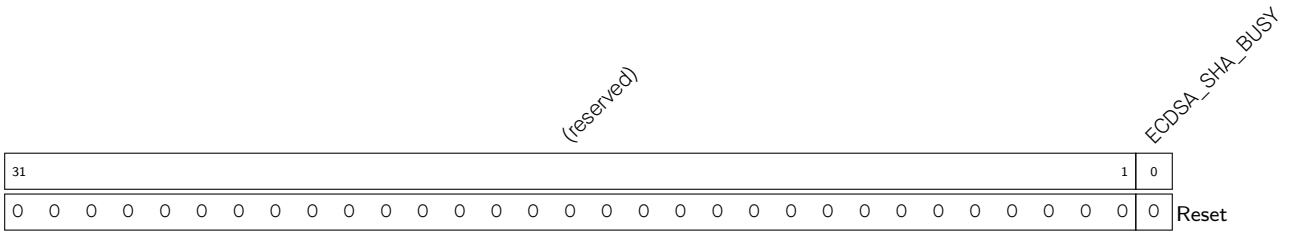
ECDSA_SHA_START Write 1 to start the first SHA operation in the ECDSA_DS process. This bit will be self-cleared after configuration. (WT)

Register 33.12. ECDSA_SHA_CONTINUE_REG (0x0214)



ECDSA_SHA_CONTINUE Write 1 to start the latter SHA operation in the ECDSA_DS process. This bit will be self-cleared after configuration. (WT)

Register 33.13. ECDSA_SHA_BUSY_REG (0x0218)



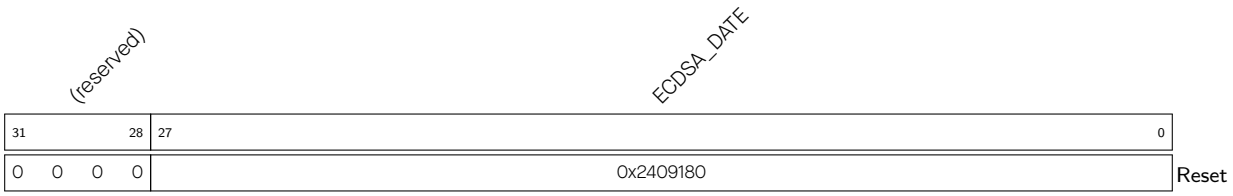
ECDSA_SHA_BUSY Represents the working state of the SHA accelerator in the ECDSA_DS process.

0: IDLE

1: BUSY

(RO)

Register 33.14. ECDSA_DATE_REG (0x00FC)



ECDSA_DATE The ECDSA_DS version control register. (R/W)

Chapter 34

External Memory Encryption and Decryption (XTS_AES)

34.1 Overview

The ESP32-P4 integrates an External Memory Encryption and Decryption module that complies with the XTS-AES standard algorithm specified in IEEE Std 1619-2007, providing security for users' application code and data stored in the external memory (flash and RAM). Users can store proprietary firmware and sensitive data (e.g., credentials for gaining access to a private network) in the external flash, or store general data in the external RAM.

34.2 Features

- General XTS-AES algorithm, compliant with IEEE Std 1619-2007
- Software-based manual encryption
- High-speed auto encryption and decryption without software's participation
- Encryption and decryption functions jointly enabled/disabled by registers configuration, eFuse parameters, and boot mode
- Configurable Anti-DPA

34.3 Module Structure

The External Memory Encryption and Decryption module consists of three blocks, namely the Manual Encryption block, Auto Encryption block, and Auto Decryption block. The module architecture is shown in Figure 34.3-1.

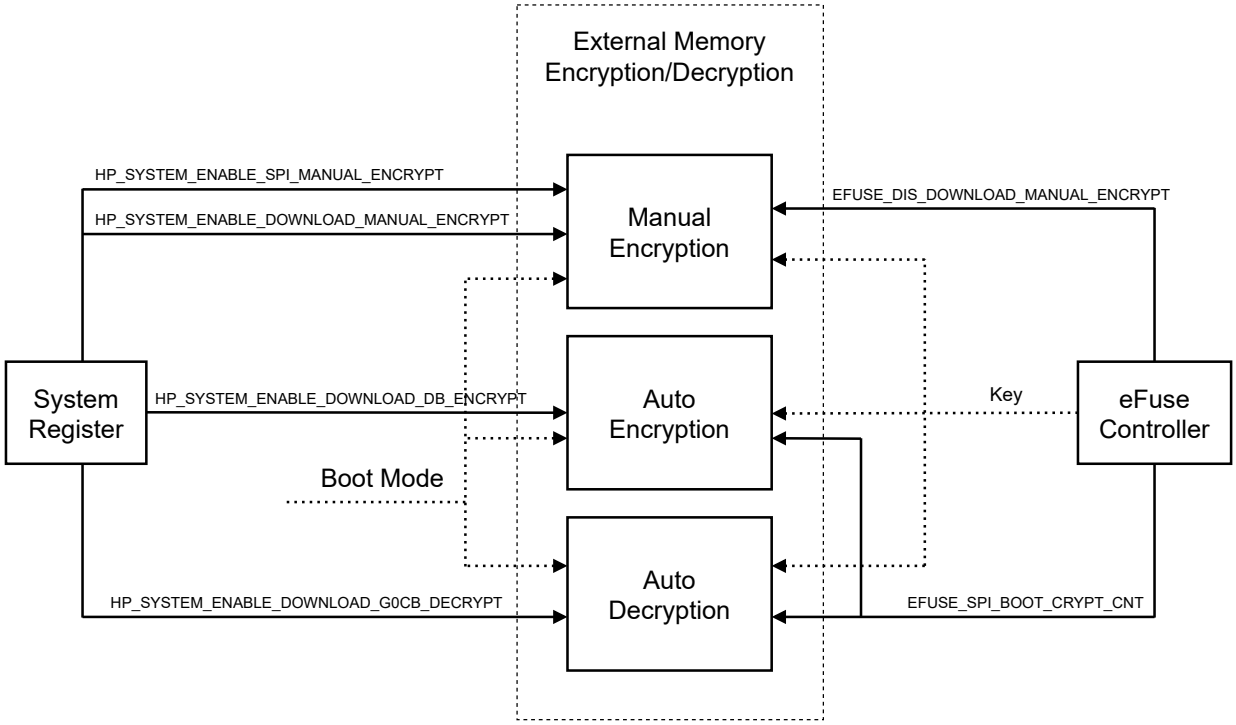


Figure 34.3-1. Architecture of the External Memory Encryption and Decryption

The Manual Encryption block can encrypt instructions and data, which will then be written to the external flash as ciphertext via SPI1.

When the CPU writes data to the external RAM through cache, the Auto Encryption block will automatically encrypt the data first, then the data will be written to the external RAM as ciphertext.

When the CPU reads from the external flash or external RAM through cache, the Auto Decryption block will automatically decrypt the ciphertext to retrieve instructions and data.

In the System Registers peripheral (see *21 System Registers (SYSREG)*), the following four bits in register HP_SYSTEM_CRYPTO_CTRL_REG are relevant to the external memory encryption and decryption:

- HP_SYSTEM_ENABLE_DOWNLOAD_MANUAL_ENCRYPT
- HP_SYSTEM_ENABLE_DOWNLOAD_G0CB_DECRYPT
- HP_SYSTEM_ENABLE_DOWNLOAD_DB_ENCRYPT
- HP_SYSTEM_ENABLE_SPI_MANUAL_ENCRYPT

The XTS_AES module also fetches two parameters from the peripheral eFuse Controller, which are: EFUSE_DIS_DOWNLOAD_MANUAL_ENCRYPT and EFUSE_SPI_BOOT_CRYPT_CNT. For detailed information, please see Chapter 9.4 *eFuse Controller (EFUSE)*.

34.4 Functional Description

34.4.1 XTS Algorithm

The manual encryption and auto encryption/decryption all use the same algorithm. During implementation, the XTS algorithm is characterized by a "data unit" of 1024 bits, defined in the Section *XTS-AES encryption procedure* of
XTS-AES Tweakable Block Cipher Standard. For more information about the XTS-AES algorithm, please refer to IEEE Std 1619-2007.

34.4.2 Key

The Manual Encryption block, Auto Encryption block, and Auto Decryption block share the same Key when implementing the XTS algorithm. The Key is provided by the eFuse hardware or the Key Manager and cannot be accessed by users.

The Key can be either 256-bit or 512-bit long. The value and length of the Key are determined by the content in the eFuse blocks from Block4 ~ Block9 and corresponding eFuse parameters. For easier description, we define:

- Block$_A$: the block whose key purpose is EFUSE_KEY_PURPOSE_XTS_AES_256_KEY_1 (please refer to Table 9.4-2 *Secure Key Purpose Values*). If Block$_A$ exists, a 256-bit Key_A is stored in it.
- Block$_B$: the block whose key purpose is EFUSE_KEY_PURPOSE_XTS_AES_256_KEY_2 (please refer to Table 9.4-2 *Secure Key Purpose Values*). If Block$_B$ exists, a 256-bit Key_B is stored in it.
- Block$_C$: the block whose key purpose is EFUSE_KEY_PURPOSE_XTS_AES_128_KEY (please refer to Table 9.4-2 *Secure Key Purpose Values*). If Block$_C$ exists, a 256-bit Key_C is stored in it.

There are five possibilities of how the Key is generated depending on whether Block$_A$, Block$_B$, and Block$_C$ exist or not, as shown in Table 34.4-1. In each case, the Key can be uniquely determined by Block$_A$, Block$_B$ or Block$_C$.

Table 34.4-1. Key generated based on Key_A, Key_B, and Key_C

Block$_A$	Block$_B$	Block$_C$	Key	Key Length (bit)
Yes	Yes	Don't care	$Key_A \| \| Key_B$	512
Yes	No	Don't care	$Key_A \| \| 0^{256}$	512
No	Yes	Don't care	$0^{256} \| \| Key_B$	512
No	No	Yes	Key_C	256
No	No	No	0^{256}	256

Notes:

- "YES" indicates that the block exists
- "NO" indicates that the block does not exist
- "0^{256}" indicates a bit string that consists of 256-bit zeros
- "||" is a bonding operator for joining one-bit string to another

For more information on key purposes, please refer to Table 9.4-2 *Secure Key Purpose Values* in Chapter 9.4 *eFuse Controller (EFUSE)*.

34.4.3 Target Memory Space

The target memory space refers to a continuous address space in the external memory where the first encrypted ciphertext is stored. The target memory space can be uniquely determined by three relevant parameters: type, size, and base address, whose definitions are listed below.

- Type: the $type$ of the target memory space, either external flash or external RAM. Value 0 indicates external flash, while 1 indicates external RAM.
- Size: the $size$ of the target memory space, indicating the number of bytes encrypted in one encryption operation, which supports 16, 32, or 64 bytes.
- Base address: the $base_addr$ of the target memory space. It is a 30-bit physical address, with a range of 0x0000_0000 ~ 0x3FFF_FFFF. It should be aligned to $size$, i.e., $base_addr\%size == 0$.

For example, if there are 16 bytes of instruction data that need to be encrypted and written to address 0x130 ~ 0x13F in the external flash, then the target space is 0x130 ~ 0x13F, type is 0 (external flash), size is 16 (bytes), and the base address is 0x130.

The encryption of any length (must be multiples of 16 bytes) of plaintext instruction/data can be completed separately in multiple operations, and each operation has individual target memory space and relevant parameters.

For Auto Encryption/Decryption blocks, these parameters are automatically defined by hardware. For the Manual Encryption block, these parameters should be configured manually by users.

Note:

The "tweak" defined in Chapter *5.1 Data units and tweaks* of IEEE Std 1619-2007 is a 128-bit non-negative integer ($tweak$), which can be generated according to $tweak$ = $type * 2^{30}$ + ($base_addr$ & 0x3FFFFF80). The lowest 7 bits and the highest 97 bits in $tweak$ are always zero.

34.4.4 Data Writing

For Auto Encryption/Decryption blocks, data writing is automatically applied in hardware. For Manual Encryption blocks, data writing should be applied by users. The Manual Encryption block has a register block which consists of 16 registers, i.e., XTS_AES_PLAIN_*n*_REG (*n*: 0 ~ 15), that are dedicated to data writing and can store up to 512 bits of plaintext at a time.

Actually, the Manual Encryption block does not care where the plaintext comes from, but only where the ciphertext will be stored. Because of the strict correspondence between plaintext and ciphertext, in order to better describe how the plaintext is stored in the register block, we assume that the plaintext is stored in the target memory space in the first place and replaced by ciphertext after encryption. Therefore, the following description in this section no longer has the concept of "plaintext", but uses "target memory space" instead.

How mapping between target memory space and registers works:

Assume a word in the target memory space is stored in $address$, define $offset$ = $address\%64$, n = $offset/4$, then the word will be stored in register XTS_AES_PLAIN_n_REG.

For example, when the $size$ is 64, all registers in the register block will be used. The mapping between $offset$ and registers now is shown in Table 34.4-2.

Table 34.4-2. Mapping Between Offsets and Registers

offset	Register	*offset*	Register
0x00	XTS_AES_PLAIN_0_REG	0x20	XTS_AES_PLAIN_8_REG
0x04	XTS_AES_PLAIN_1_REG	0x24	XTS_AES_PLAIN_9_REG
0x08	XTS_AES_PLAIN_2_REG	0x28	XTS_AES_PLAIN_10_REG
0x0C	XTS_AES_PLAIN_3_REG	0x2C	XTS_AES_PLAIN_11_REG
0x10	XTS_AES_PLAIN_4_REG	0x30	XTS_AES_PLAIN_12_REG
0x14	XTS_AES_PLAIN_5_REG	0x34	XTS_AES_PLAIN_13_REG
0x18	XTS_AES_PLAIN_6_REG	0x38	XTS_AES_PLAIN_14_REG
0x1C	XTS_AES_PLAIN_7_REG	0x3C	XTS_AES_PLAIN_15_REG

34.4.5 Manual Encryption Block

The Manual Encryption block is a peripheral module. It is equipped with registers and can be accessed by the CPU directly. Registers embedded in this block, the System Registers peripheral, eFuse parameters, and boot mode jointly configure and use this module. Please note that the Manual Encryption block can only encrypt for storage in the external flash.

The Manual Encryption block is operational only under certain conditions:

- In SPI Boot mode:

  If bit HP_SYSTEM_ENABLE_SPI_MANUAL_ENCRYPT in register HP_SYSTEM_CRYPTO_CTRL_REG is 1, the Manual Encryption block can be enabled. Otherwise, it is not operational.

- In Joint Download Boot mode:

  If bit HP_SYSTEM_ENABLE_DOWNLOAD_MANUAL_ENCRYPT in register HP_SYSTEM_CRYPTO_CTRL_REG is 1 and the eFuse parameter EFUSE_DIS_DOWNLOAD_MANUAL_ENCRYPT is 0, the Manual Encryption block can be enabled. Otherwise, it is not operational.

Note:

Even though the CPU can skip cache and get the encrypted instruction/data directly by reading the external memory, users can by no means access Key.

34.4.6 Auto Encryption Block

The Auto Encryption block is not a conventional peripheral, so it does not have any registers and cannot be accessed by the CPU directly. The System Registers peripheral, eFuse parameters, and boot mode jointly configure and use this block.

The Auto Encryption block is operational only under certain conditions:

- In SPI Boot mode:

  when EFUSE_SPI_BOOT_CRYPT_CNT (3 bits) is set to 1, 2, 4 or 7 (i.e., there is an odd number of 1s in its binary representation), then the Auto Encryption block can be enabled. Otherwise, it is not operational.

- In Joint Download Boot mode:

  when bit HP_SYSTEM_ENABLE_DOWNLOAD_DB_ENCRYPT in register HP_SYSTEM_CRYPTO_CTRL_REG is 1, the Auto Encryption block can be enabled. Otherwise, it is not operational.

Note:

- When the Auto Encryption block is enabled, it will automatically encrypt data if the CPU writes data to the external RAM, and then the encrypted ciphertext will be written to the external RAM. The entire encryption process does not need software participation and is transparent to the cache. Users can by no means obtain the encryption Key during the process.
- When the Auto Encryption block is disabled, it will ignore the CPU's access request to the cache and not process the data. Therefore, the data will be written to the external RAM as plaintext directly.

34.4.7 Auto Decryption Block

The Auto Decryption block is not a conventional peripheral, so it does not have any registers and cannot be accessed by the CPU directly. The System Registers peripheral, eFuse parameters, and boot mode jointly configure and use this block.

The Auto Decryption block is operational only under certain conditions:

- In SPI Boot mode:

  when EFUSE_SPI_BOOT_CRYPT_CNT (3 bits in total) is set to 1, 2,4 or 7 (i.e., there is an odd number of 1s in its binary representation), then the Auto Decryption block can be enabled. Otherwise, it is not operational.

- In Joint Download Boot mode:

  when bit HP_SYSTEM_ENABLE_DOWNLOAD_GOCB_DECRYPT in register HP_SYSTEM_CRYPTO_CTRL_REG is 1, the Auto Decryption block can be enabled. Otherwise, it is not operational.

Note:

- When the Auto Decryption block is enabled, it will automatically decrypt the ciphertext if the CPU reads instructions/data from the external memory via cache to retrieve the instructions/data. The entire decryption process does not need software participation and is transparent to the cache. The software can by no means obtain the decryption Key during the process.
- When the Auto Decryption block is disabled, it does not have any effect on the contents stored in the external memory, no matter if they are encrypted or not. Therefore, what the CPU reads via cache is the original information stored in the external memory.

34.5 Software Process

When the Manual Encryption block operates, software needs to be involved in the process. The steps are as follows:

1. Configure XTS_AES:
   - Set register XTS_AES_DESTINATION_REG to $type$ = 0.
   - Set register XTS_AES_PHYSICAL_ADDRESS_REG to $base_addr$.
   - Set register XTS_AES_LINESIZE_REG to $\frac{size}{32}$.

   For definitions of $base_addr$ and $size$, please refer to Section 34.4.3.
2. Write plaintext instructions/data to the registers block XTS_AES_PLAIN_*n*_REG (*n*: 0-15). For detailed information, please refer to Section 34.4.4.
   Please write data to registers according to your actual needs, and the unused ones could be set to arbitrary values.
3. Wait for Manual Encryption block to be idle. Poll register XTS_AES_STATE_REG until it reads 0 which indicates the Manual Encryption block is idle.
4. Trigger manual encryption by writing 1 to register XTS_AES_TRIGGER_REG.
5. Wait for the encryption process completion. Poll register XTS_AES_STATE_REG until it reads 2.
   Step 1 to 5 are the steps of encrypting plaintext instructions/data with the Manual Encryption block using the Key.
6. Write 1 to register XTS_AES_RELEASE_REG to grant SPI1 the access to the encrypted ciphertext. After this, the value of register XTS_AES_STATE_REG will become 3.
7. Call SPI1 to write the ciphertext in the external flash (see Section API Reference - Flash Encrypt in ESP-IDF Programming Guide).
8. Write 1 to register XTS_AES_DESTROY_REG to destroy the ciphertext. After this, the value of register XTS_AES_STATE_REG will become 0.

Repeat the above steps according to the amount of plaintext instructions/data that need to be encrypted.

34.6 Anti-DPA

DPA (Differential Power Analysis) is a side-channel attack method in cryptography, through which an attacker can statistically analyze data collected from multiple encryption operations to calculate intermediate values in the encryption computation. ESP32-P4 XTS_AES supports Anti-DPA to defend against external DPA attacks.
The XTS-AES algorithm can be divided into two steps, according to IEEE Std 1619-2007:

- Step 1: Calculating T value. In this section, we define this step as "calculating T".
- Step 2: Calculating Cipher/Plain text. In this section, we define this step as "calculating D".

Different security levels can be configured through registers:

- First we define the below parameters for a better description:
  - $select_reg$ = XTS_AES_CRYPT_DPA_SELECT_REGISTER
  - $reg_d_dpa_en$ = XTS_AES_CRYPT_CALC_D_DPA_EN
  - $efuse_dpa_en$ = EFUSE_CRYPT_DPA_ENABLE

  - $reg_anti_dpa_level =$ XTS_AES_CRYPT_SECURITY_LEVEL
  - $efuse_anti_dpa_level = 3$

- Configure the security level of Anti-DPA for the XTS_AES module:

$$Anti_DPA_level = select_reg\ ?\ (reg_anti_dpa_level) : (efuse_dpa_en * efuse_anti_dpa_level)$$

  When $Anti_DPA_level$ equals 0, Anti-DPA is disabled. The higher the value of $Anti_DPA_level$ is, the stronger the Anti-DPA ability is.

- Configure whether to enable Anti-DPA when the XTS-AES algorithm is calculating D:

$$Anti_DPA_enabled_in_calc_D = select_reg\ ?\ reg_d_dpa_en : efuse_dpa_en$$

  If $Anti_DPA_level$ is not 0, when $Anti_DPA_enabled_in_calc_D$ equals to 1, Anti-DPA is enabled when XTS-AES algorithm is calculating D.
  If $Anti_DPA_level$ is not 0, Anti-DPA is always enabled when the XTS-AES algorithm is calculating T.

Note:

- Even if efuse_dpa_en is set to 1, you can still disable anti-DPA by configuring $select_reg$ = 1 and $reg_anti_dpa_level$ = 0.
- Configuring whether or not to enable Anti-DPA will have an impact on the external storage access bandwidth:
  - When Anti-DPA is enabled during the calculation of D, the read and write bandwidth will be reduced by more than 50% when the Anti-DPA level >= 4.
  - When Anti-DPA is disabled during the calculation of D, the read and write bandwidth will be reduced by more than 50% when the Anti-DPA level >= 6.

34.7 Register Summary

The addresses in this section are relative to the External Memory Encryption and Decryption base address provided in Table 9.3-2 in Chapter 9 *System and Memory*.

The abbreviations given in Column **Access** are explained in Section *Access Types for Registers*.

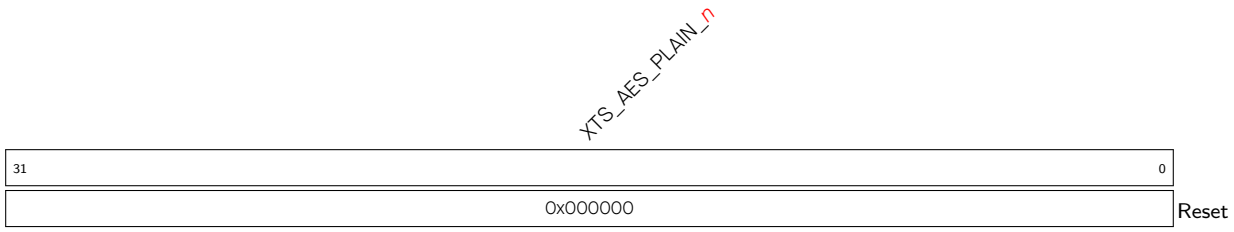
Name	Description	Address	Access
Plaintext Register Heap			
XTS_AES_PLAIN_0_REG	Plaintext register 0	0x0300	R/W
XTS_AES_PLAIN_1_REG	Plaintext register 1	0x0304	R/W
XTS_AES_PLAIN_2_REG	Plaintext register 2	0x0308	R/W
XTS_AES_PLAIN_3_REG	Plaintext register 3	0x030C	R/W
XTS_AES_PLAIN_4_REG	Plaintext register 4	0x0310	R/W
XTS_AES_PLAIN_5_REG	Plaintext register 5	0x0314	R/W
XTS_AES_PLAIN_6_REG	Plaintext register 6	0x0318	R/W
XTS_AES_PLAIN_7_REG	Plaintext register 7	0x031C	R/W
XTS_AES_PLAIN_8_REG	Plaintext register 8	0x0320	R/W
XTS_AES_PLAIN_9_REG	Plaintext register 9	0x0324	R/W
XTS_AES_PLAIN_10_REG	Plaintext register 10	0x0328	R/W
XTS_AES_PLAIN_11_REG	Plaintext register 11	0x032C	R/W
XTS_AES_PLAIN_12_REG	Plaintext register 12	0x0330	R/W
XTS_AES_PLAIN_13_REG	Plaintext register 13	0x0334	R/W
XTS_AES_PLAIN_14_REG	Plaintext register 14	0x0338	R/W
XTS_AES_PLAIN_15_REG	Plaintext register 15	0x033C	R/W
Configuration Registers			
XTS_AES_LINESIZE_REG	Configures the size of target memory space	0x0340	R/W
XTS_AES_DESTINATION_REG	Configures the type of the external memory	0x0344	R/W
XTS_AES_PHYSICAL_ADDRESS_REG	Physical address	0x0348	R/W
XTS_AES_DPA_CTRL_REG	Configures the Anti-DPA function	0x0388	R/W
Control/Status Registers			
XTS_AES_TRIGGER_REG	Activates AES algorithm	0x034C	WO
XTS_AES_RELEASE_REG	Release control	0x0350	WO
XTS_AES_DESTROY_REG	Destroy control	0x0354	WO
XTS_AES_STATE_REG	Status register	0x0358	RO
Version Register			
XTS_AES_DATE_REG	Version control register	0x035C	R/W

34.8 Registers

The addresses in this section are relative to the External Memory Encryption and Decryption base address provided in Table 9.3-2 in Chapter 9 *System and Memory*.

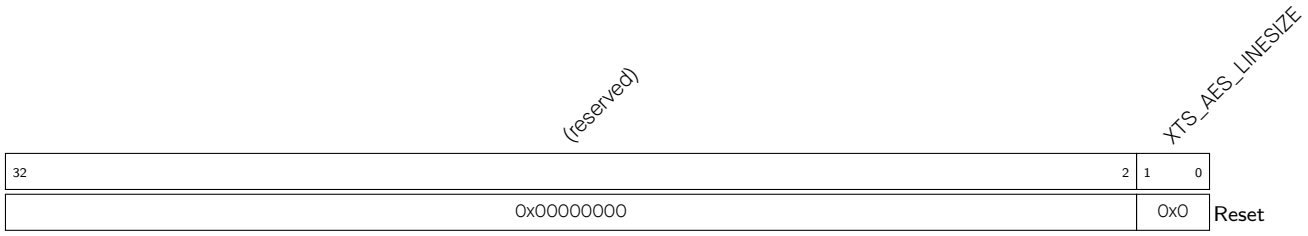
For how to program reserved fields, please refer to Section Programming Reserved Register Field.

Register 34.1. XTS_AES_PLAIN_*n*_REG (*n*: 0-15) (0x0300+4n*)**



XTS_AES_PLAIN_*n* Configures the *n*th 32-bit piece of plain text. (R/W)

Register 34.2. XTS_AES_LINESIZE_REG (0x0340)



XTS_AES_LINESIZE Configures the data size of one encryption operation.

0: 16 bytes

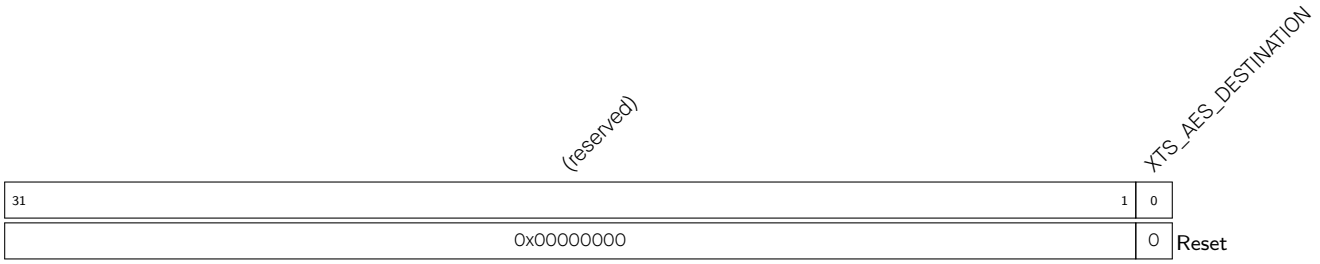
1: 32 bytes

2: 64 bytes

3: Invalid

(R/W)

Register 34.3. XTS_AES_DESTINATION_REG (0x0344)



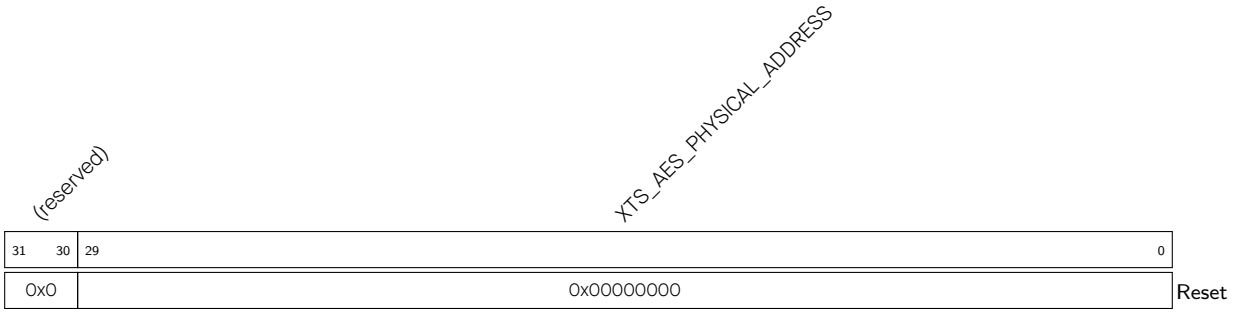
XTS_AES_DESTINATION Configures the type of external memory for Manual Encryption. Currently, it must be set to 0, as the Manual Encryption block only supports flash encryption. Set this bit to 1 may cause an error.

0: flash

1: external RAM

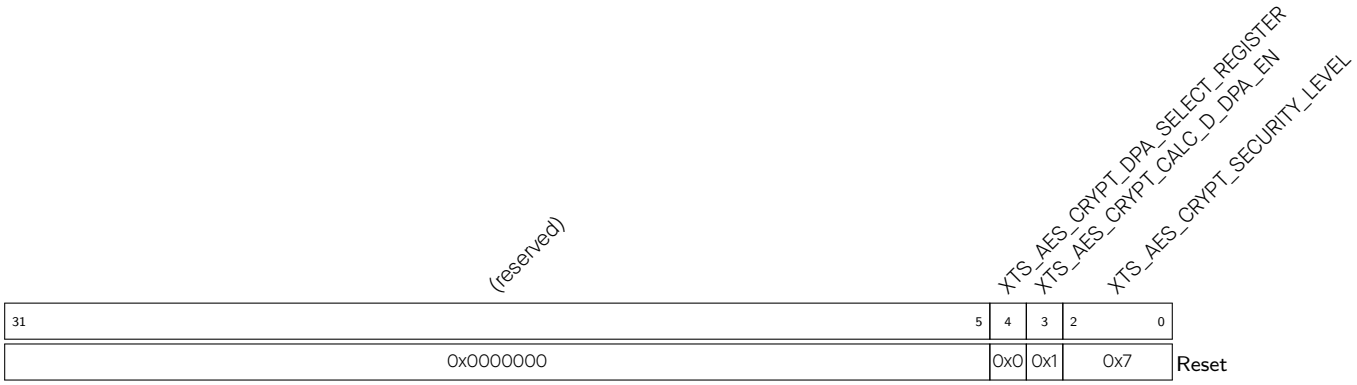
(R/W)

Register 34.4. XTS_AES_PHYSICAL_ADDRESS_REG (0x0348)



XTS_AES_PHYSICAL_ADDRESS Configures physical address. Note that its value should be within the range between 0x0000_0000 and 0x00FF_FFFF. (R/W)

Register 34.5. XTS_AES_DPA_CTRL_REG (0x0388)



XTS_AES_CRYPT_DPA_SELECT_REGISTER Configures whether the Anti-DPA function is controlled by eFuse or register.

0: The Anti-DPA function is configured by register.

1: The Anti-DPA function is configured by eFuse.

(R/W)

XTS_AES_CRYPT_CALC_D_DPA_EN Configures whether to enable Anti-DPA in the XTS_AES algorithm.

0: Enable Anti-DPA only when calculating T

1: Enable Anti-DPA both when calculating T and D

Note that this field is only effective when XTS_AES_CRYPT_SECURITY_LEVEL is not 0.

(R/W)

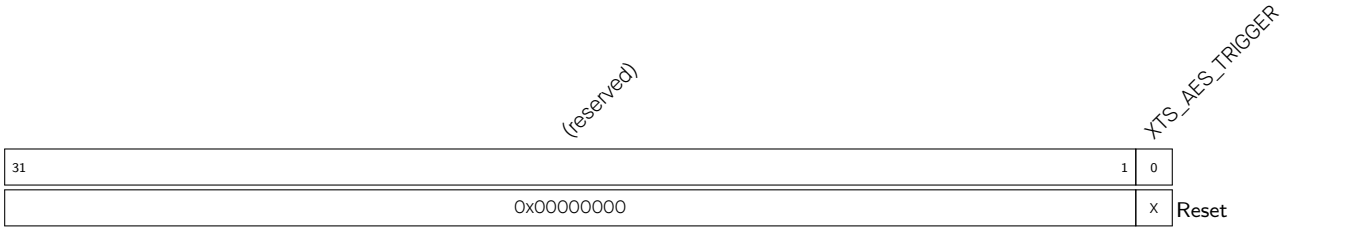
XTS_AES_CRYPT_SECURITY_LEVEL Configures the security level of external memory encryption and decryption.

0: Disable the Anti-DPA function

1-7: The bigger the number is, the more secure the encryption and decryption are

(R/W)

Register 34.6. XTS_AES_TRIGGER_REG (0x034C)



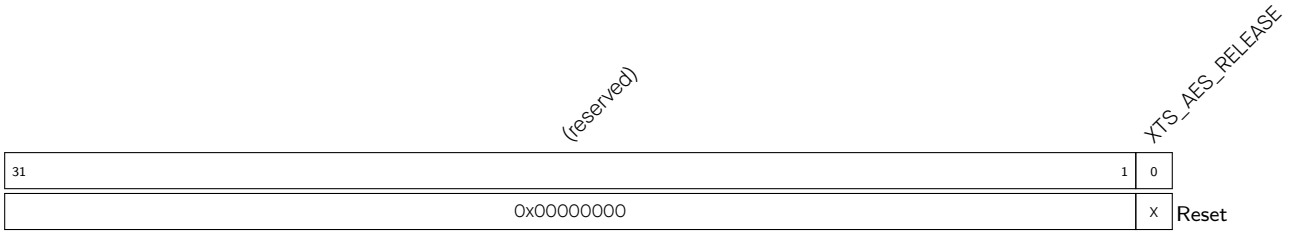
XTS_AES_TRIGGER Configures whether or not to enable manual encryption.

0: Disable manual encryption

1: Enable manual encryption

(WO)

Register 34.7. XTS_AES_RELEASE_REG (0x0350)



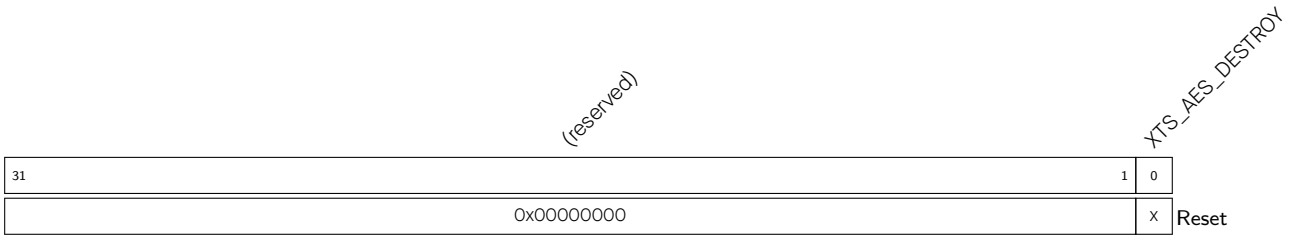
XTS_AES_RELEASE Configures whether to grant SPI1 access to the encrypted result.

0: No effect

1: Grant SPI1 access

(WO)

Register 34.8. XTS_AES_DESTROY_REG (0x0354)



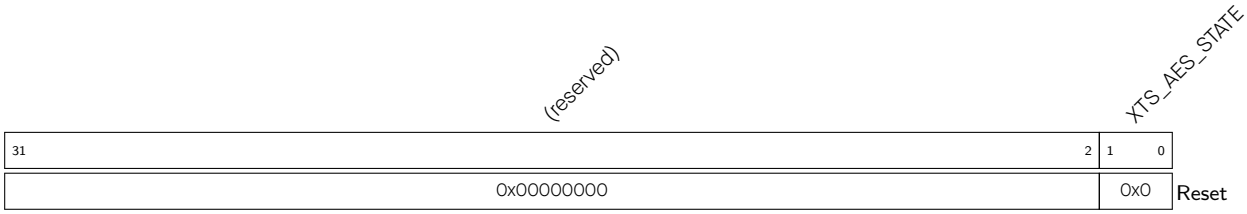
XTS_AES_DESTROY Configures whether to destroy the encrypted result.

0: No effect

1: Destroy encrypted result

(WO)

Register 34.9. XTS_AES_STATE_REG (0x0358)



XTS_AES_STATE Represents the status of the Manual Encryption block. 0 (XTS_AES_IDLE): Idle

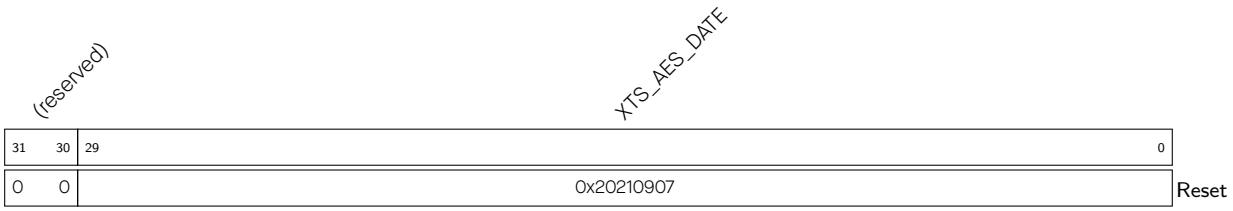
1 (XTS_AES_BUSY): Busy with encryption

2 (XTS_AES_DONE): Encryption completed, but the encrypted result is not accessible to SPI

3 (XTS_AES_RELEASE): Encrypted result is accessible to SPI

(RO)

Register 34.10. XTS_AES_DATE_REG (0x035C)



XTS_AES_DATE Version control register. (R/W)

Chapter 35

Random Number Generator (RNG)

35.1 Introduction

The ESP32-P4 contains a true random number generator, which generates 32-bit random numbers that can be used for cryptographical operations, among other things.

35.2 Feature List

The random number generator in ESP32-P4 generates true random numbers, which means random numbers generated from a physical process, rather than by means of an algorithm. No number generated within the specified range is more or less likely to appear than any other number.

35.3 Functional Description

Every 32-bit value that the system reads from the LPSYSREG_RNG_DATA_REG register of the random number generator is a true random number. These true random numbers are generated based on the system's **thermal noise**, the **asynchronous clock mismatch** and the **BUF_CHAIN**.

- **Thermal noise** comes from the SAR ADC. Whenever the SAR ADC is enabled, bit streams will be generated and fed into the random number generator through an XOR logic gate as random seeds.
- RC_FAST_CLK is an **asynchronous clock** source and it increases the RNG entropy by introducing circuit metastability. Please refer to the Note about the RC_FAST_CLK.
- The count value generated by the **BUF_CHAIN**, implemented using BUF_CELL, is used as a random seed for the random number generator.

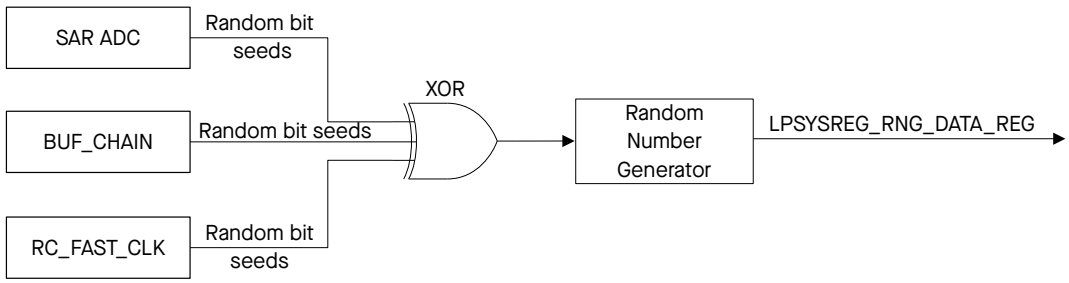


Figure 35.3-1. Noise Source

When there is noise coming from the SAR ADC, the random number generator is fed with a 1-bit entropy in one clock cycle of RC_FAST_CLK, which is generated from an internal RC oscillator (see Chapter 11 *Reset and Clock* for details). Thus, it is advisable to read the LPSYSREG_RNG_DATA_REG register at a maximum rate of 1 MHz to obtain the maximum entropy.

35.4 Programming Procedure

When using the random number generator, make sure at least either the SAR ADC or RC_FAST_CLK is enabled. Otherwise, pseudo-random numbers will be returned.

- For SAR ADC, please refer to Chapter 65 *ADC Controller (ADC)*.
- RC_FAST_CLK is enabled by setting the PMU_HP_SLEEP_XPD_FOSC_CLK bit in the PMU_HP_SLEEP_LP_CK_POWER_REG register. Please refer to the Note about the RC_FAST_CLK.
- RC_FAST_CLK is gated by setting the LPPERI_CK_EN_RNG bit in the LPPERI_CLK_EN_REG register.
- The BUF_CHAIN can be enabled by setting the RNG_CFG_REG register's RNG_SAMPLE_ENABLE bit.

Note:
Enabling RC_FAST_CLK increases the RNG entropy. However, to ensure maximum entropy, it's recommended to always enable an ADC source as well.

When using the random number generator, read the LPSYSREG_RNG_DATA_REG register multiple times until sufficient random numbers have been generated. Ensure the rate at which the register is read does not exceed the frequencies described in section 35.3 above.

35.5 Register Summary

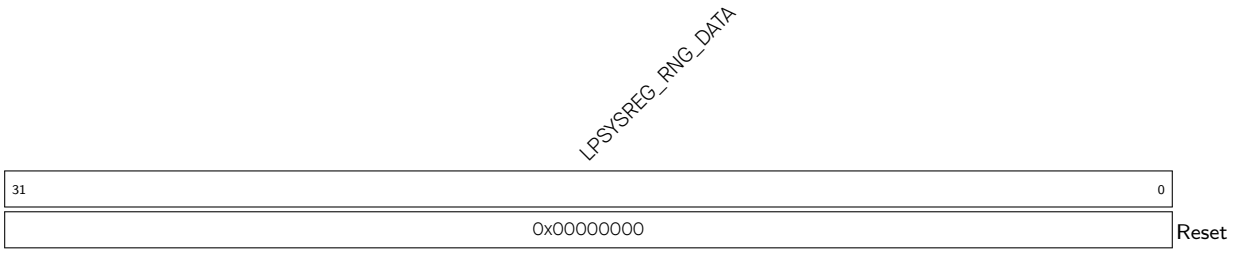
The abbreviations given in Column **Access** are explained in Section *Access Types for Registers*.

Name	Description	Address	Access
LPSYSREG_RNG_DATA_REG	Random source data	0x5011_01A4	RO
LPPERI_CLK_EN_REG	Integrated clock gating enable signal of RC_FAST_CLK	0x5012_0000	R/W
RNG_CFG_REG	Configuration register of RNG	0x5012_6000	R/W

35.6 Registers

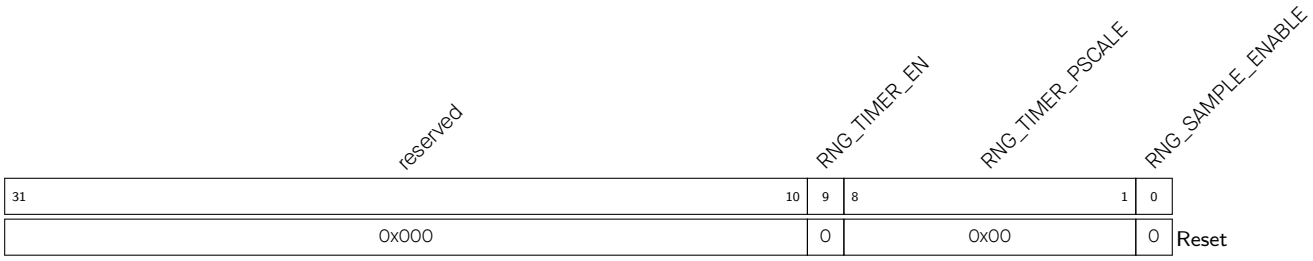
For how to program reserved fields, please refer to Section Programming Reserved Register Field.

Register 35.1. LPSYSREG_RNG_DATA_REG (0x5011_01A4)



LPSYSREG_RNG_DATA Random number source. (RO)

Register 35.2. RNG_CFG_REG (0x5012_6000)



RNG_SAMPLE_ENABLE Configures whether to enable the RNG asynchronous clock noise source.

1: Enable

0: Disable

(R/W)

RNG_TIMER_PSCALE Represents the prescale factor of the RNG internal timer.(R/W)

RNG_RTC_TIMER_EN Configures whether to enable the RNG internal timer.

1: Enable and use the counter value of this timer when generating random numbers.

0: Disable the timer.

(R/W)

Chapter 36

Key Manager

36.1 Overview

This Key Management module is the security core of the system. It has a Key Manager and a Hardware Unique Key (HUK) Generator. They can store and deploy keys securely. By utilizing the HUK as the root of trust for each chip, the system ensures that no plaintext keys are ever stored on the chip, enhancing its overall security.

Compared with using eFuse to store keys, the Key Manager and HUK Generator can improve the system in the following ways:

- Environmental adaptability: It supports secure key deployment and can complete key deployment safely, even if the external environment is untrusted.
- Anti-attack ability: Each chip's SRAM has a unique default value that software cannot access. The chip stores no plaintext key. It resists slice and copy attacks. For example, observing the keys in eFuse directly by slicing will fail.
- Key management flexibility: The Key Manager enables users to store key information (non-plaintext) in external memory. Unlike eFuse, which has limitations on the number of keys it can store, Key Manager allows users to create an unlimited number of keys, perform dynamic key changes, key replacements, and etc.

36.2 Background Knowledge

An SRAM PUF (Physical Unclonable Function) uses the natural power-up state of SRAM cells as a kind of silicon "fingerprint." When a chip powers on, each uninitialized SRAM bit tends to fall into 0 or 1 based on tiny manufacturing differences in its transistors, and because those differences are unique for every chip, the resulting bit pattern is device-unique.

We use this behavior to create a Hardware Unique Key (HUK) without ever storing that key anywhere on chip: the chip can regenerate its secret from physics on each boot instead of reading it from flash or fuses. The HUK then acts as the root of trust for secure boot, device identity and attestation, encryption keys, and wrapping other keys.

The raw SRAM PUF bits are not perfectly stable, though: most bits come up the same way on every power cycle, but some are noisy and occasionally flip due to temperature, voltage, or noise. In order to generate a stable HUK, the chip generates auxiliary data (huk_info) with error correction capabilities during the HUK generation stage to recover the HUK. Users need to store this auxiliary data in flash memory.

On every later boot, hardware reads the new (noisy) PUF response, uses the helper data to correct any flipped bits back to the reference pattern, and reconstructs exactly the same HUK as during enrollment. That HUK is

loaded into hidden key slots inside crypto hardware (never into normal RAM), giving the device a repeatable, per-chip unique secret key derived from SRAM PUF, even though the underlying bits are not 100% identical at every power-up.

36.3 Glossary

To better illustrate the functions of the HUK Generator and Key Manager, the following terms and parameters are used in this chapter.

Terms

HUK	Hardware Unique Key (HUK), a hardware-generated key derived from the default power-on value using SRAM PUF.
SRAM PUF	Physical Unclonable Function (PUF), based on SRAM, a security technology that leverages the unique, unpredictable power-up behavior of SRAM to generate a unique fingerprint or cryptographic key for a device.
HUK Generation Mode	In this mode, the HUK Generator creates a new HUK.
HUK Recovery Mode	In this mode, the HUK Generator recovers a previously deployed HUK.
Random Deploy Mode	In this mode, the Key Manager deploys a random private key generated by the Random Number Generator.
AES Deploy Mode	In this mode, the Key Manager deploys a specified private key ($k1$) onto the chip.
ECDH0 Deploy Mode	In this mode, the Key Manager deploys a negotiated private key ($k1 * k2 * G$) onto the chip. $k1 * k2 * G$ is generated using $k1$ and $k2 * G$.
ECDH1 Deploy Mode	In this mode, the Key Manager deploys a negotiated private key ($k1 * k2 * G$) onto the chip. $k1 * k2 * G$ is generated using $k1$, $k2 * G$, and $k2_info$.
Private Key Recovery Mode	In this mode, the Key Manager recovers a previously deployed private key.
key_info **Export Mode**	In this mode, the Key Manager regenerates the key_info. Users can export this key_info from the internal memory of the Key Manager.

Parameters

huk_info	Contains fuzzy extraction information and error-correcting code information after RS coding of HUK, but does not contain any information of HUK itself. This huk_info later can be used to recover the deployed HUK.
key_info	In any working mode of the Key Manager, this key_info is generated by calculating AES{AES{key,rand_num}||rand_num,HUK}.
$AES\{a, b\}$	Encrypt plaintext a with b as the key, using AES_ECB algorithm, which is the typical mode of the AES module.
$rand_num$	A random number
$a\|\|b$	Concatenates two pieces of data into one.
$k1$	A specified private key

$k2$	Auxiliary key used to assist with key deployment, applicable in both AES Deploy Mode and ECDH1 Deploy Mode. In AES Deployment Mode, its value does not affect the deployed key. In ECDH1 deployment mode, its value affects the negotiated deployment key.
$k1_encrypted$	A key generated by encrypting $k1$ with $k2$. In AES Deploy Mode or ECDH1 Deploy Mode: $k1_encrypted$ = AES{k1,k2}
$k2_info$	The recovery information of $k2$. In AES Deploy Mode or ECDH1 Deploy Mode: $k2_info$ = AES{AES{k2,rand_num}||rand_num,init_key}.
$init_key$	Used in the generation of $k2_info$ as well as in the recovery process from $k2_info$ to $k2$. Its value can be stored in the eFuse block whose purpose is EFUSE_KM_INIT_KEY or provided through software configuration.
sw_init_key	Software configured $init_key$
$k1 * k2 * G$	A negotiated key generated by calculating $k2 * G$ through ECC.
$k1 * G$	A key generated by calculating $k1$ through ECC.
$k2 * G$	An auxiliary public key generated by calculating $k2 * G$ through ECC. This key is used later to generate $k1 * k2 * G$.
$key_purpose$	The intended usage of the key.

36.4 Features

The HUK Generator has the following features:

- HUK Generation Mode:
  - Generates a new HUK and its recovery information
- HUK Recovery Mode:
  - Recovers a deployed HUK with its recovery information
- Prompt for HUK recovery error
- Prompt for HUK risk level

The Key Manager has the following features:

- Unlimited number of keys
- Specified private key deployment (AES Deploy Mode):
  - Users specify the value of the key
- Negotiated private key deployment (ECDH0 Deploy Mode):
  - Highest security mode: there is no need to worry about the leaks of data in external channels
  - Requires chip initiation to obtain the private key
  - Negotiates the key value between each chip and the user
- Negotiated private key deployment (ECDH1 Deploy Mode):

  - Deploy the negotiated private key using the auxiliary key
  - No need to boot the chip to obtain the private key
- Random key deployment (Random Deploy Mode):
  - Deploys a hardware-generated random key with nobody knowing the exact value
- Private key recovery deployment (Private Key Recovery Mode):
  - Recovers exactly the same key by entering the key information generated during deployment
- Key information export (key_info Export Mode):
  - Generates unique key information for the same key each time

36.5 Architectural Overview

The top-level architecture of the Key Manager and HUK Generator is shown in the figure below.

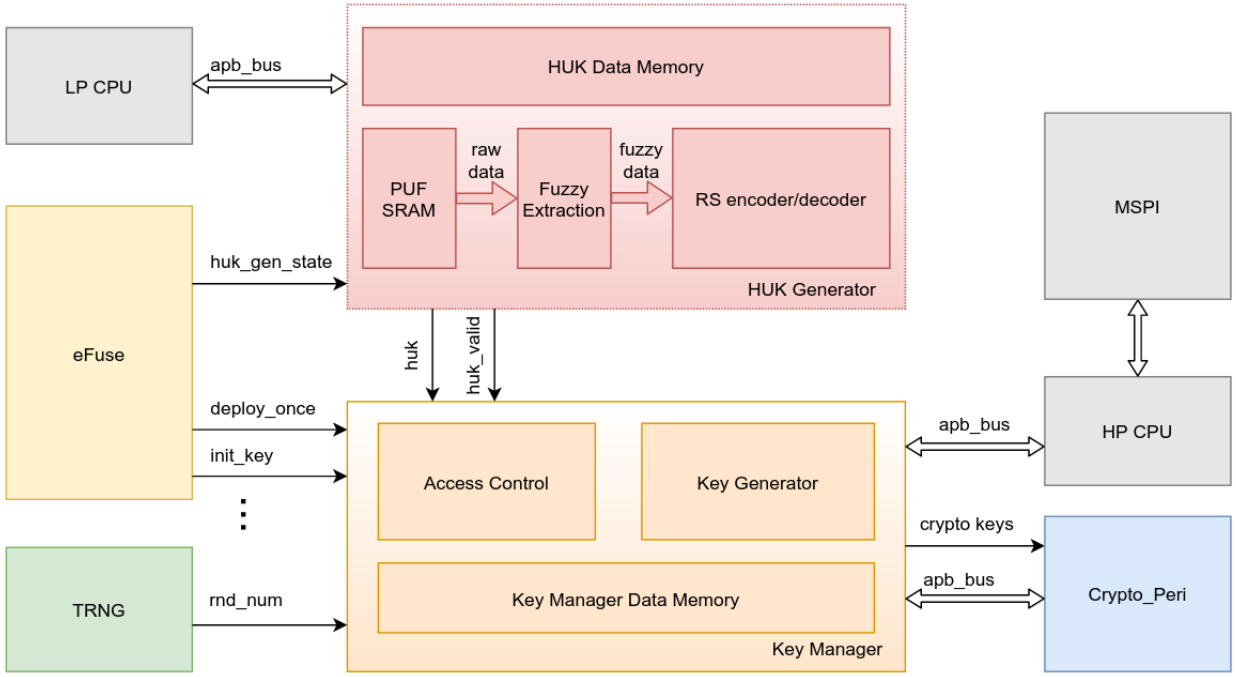


Figure 36.5-1. Architecture of Key Manager and HUK Generator

The Key Manager has the following interaction paths with external modules:

- **eFuse path:** Controls the configuration of Key Manager with the help of eFuse bits.
- **HP APB path:** Configures registers and transmits memory data via HP APB bus.
- **HUK path:** Receives HUK from HUK Generator.
- **Crypto path:** Controls cryptographic accelerators such as ECC/AES to complete cryptographic operations via Crypto APB bus.
- **TRNG path:** Receives true random data from the TRNG.

The HUK Generator has the following interaction paths with external modules.

- **eFuse path:** Controls the configuration of HUK Generator with the help of eFuse bits.
- **LP APB path:** Configures registers and transmits memory data via LP APB bus.
- **Key Manager path:** Sends HUK to Key Manager.

36.6 Functional Description

36.6.1 HUK Generator

HUK Generator generates unique HUKs for each chip based on SRAM Physical Unclonable Function (PUF). This HUK is completely inaccessible by software, ensuring robust security for the Key Manager, which relies on this HUK for secure operations later on.

36.6.1.1 HUK Generation Process

The HUK Generator has the following modes:

- **HUK Generation Mode:** Creates a new HUK and generates corresponding HUK recovery information (huk_info).
- **HUK Recovery Mode:** Recovers HUK according to huk_info.

The modes above are jointly controlled by the eFuse field EFUSE_KM_HUK_GEN_STATE, and the register HUK_MODE, as shown in the following table.

Table 36.6-1. HUK Generator Working Mode

HUK_MODE	eFuse *	Mode
1	Total number of bits set to 1 is even	HUK Generation Mode
	Total number of bits set to 1 is odd	HUK Recovery Mode
0	—	HUK Recovery Mode

* Total number of bits set to 1 in EFUSE_KM_HUK_GEN_STATE.

Note:

- Once the process is completed in HUK Generation Mode, it is recommended to write a 1 to EFUSE_KM_HUK_GEN_STATE field, ensuring that the total number of bits set to 1 in EFUSE_KM_HUK_GEN_STATE is an odd number. With this, when the chip starts next time, the HUK Generator can only enter HUK Recovery Mode. The HUK restored in HUK Recovery Mode will be the same as the one generated this time in HUK Generation Mode.
- If you do not update the eFuse bits EFUSE_KM_HUK_GEN_STATE field after completing the HUK Generation Mode and before powering off the chip, the HUK Generator can still be configured to enter HUK Generation Mode again when the chip is powered on next time. The newly generated HUK this time will have a different value from the previous one that is used in private keys. By this way, you can generate multiple HUKs and restore the one you need with its huk_info.
- huk_info can only be used with the same chip; that is, an huk_info generated on chip A can not be used to recover the HUK on chip B.

36.6.1.2 HUK Status Check

Users can read the register HUK_STATUS_REG to check the current status of the HUK and decide whether to regenerate the HUK. HUK_STATUS_REG has the following fields:

- HUK_STATUS: Indicates the current status of the HUK.
  - 0: The HUK has not been generated
  - 1: The HUK has been successfully generated and is valid
  - 2: The HUK has been generated but is invalid
  - 3: Reserved
- HUK_RISK_LEVEL: Indicates the current risk level of the HUK. The value ranges from 0 to 7, with higher values indicating greater risk. When the value is 7, the HUK is no longer available. At this point, the HUK_STATUS will be set to 2.

Note:

- HUK Generator uses the default value of SRAM PUF to generate HUK. The HUK risk level refers to the difference between the current default value of SRAM PUF and the default value of SRAM PUF at the time when the HUK was created.
- In certain special applications, such as when users want to discard a key and invalidate the corresponding key_info, they can select to update the HUK. Once the HUK is changed, the previously deployed key, as well as its key_info, both become invalid, i.e., the new huk_info together with the previous key_info can not be used to recover the key.
- It is strongly recommended that software monitor the risk level and regenerate the HUK at the value 4-6.

36.6.2 Key Manager

Note:
Each time before using the Key Manager for key deployment, please ensure that the HUK is valid by checking the register HUK_STATUS_REG.

Key Manager provides the following modes:

- **Key Deployment Modes:**
  - Random Deploy Mode: Deploys a random private key.
  - AES Deploy Mode: Deploys a specified private key.
  - ECDH0 Deploy Mode: Deploys a negotiated private key.
  - ECDH1 Deploy Mode: Deploys a negotiated private key.

  These four deployment modes allow users to select the most convenient deployment process in environments with different security levels according to their own needs.
- **Key Recovery Mode:**
  - Private Key Recovery Mode: Recovers the deployed private key.

This recovery mode allows users to freely select the key they want to recover from any number of the deployed keys.

- **Export Mode:**
  - key_info Export Mode: Generates and outputs key_info of the deployed private key.

  This export mode allows users to regenerate key_info for already deployed keys.

36.6.2.1 Random Deploy Mode

Users can use this mode to deploy a random key generated with the help of the Random Number Generator.

Note:
The Random Deploy Mode is highly convenient and well-suited for application scenarios where the exact value of the private key does not need to be known.

36.6.2.2 AES Deploy Mode

Users can use this mode to deploy a specified private key.

This mode requires the following keys and parameters:

- HUK: generated by the HUK Generator and passed to the Key Manager.
- $k1$: the specified private key to be deployed.
- $k2$: Auxiliary key used to encrypt $k1$, ensuring that $k1$ is not transmitted in plaintext over the communication link, thereby enhancing data security.
- $init_key$: Initial auxiliary key used for generating and decrypting the $k2_info$ key.
- $k2_info$: Generated from $k2$ and $init_key$, and used to recover $k2$. The calculation formula is: $k2_info = AES\{AES\{k2, rand_num\}||rand_num, init_key\}$.

The deployment flow chart is as follows.

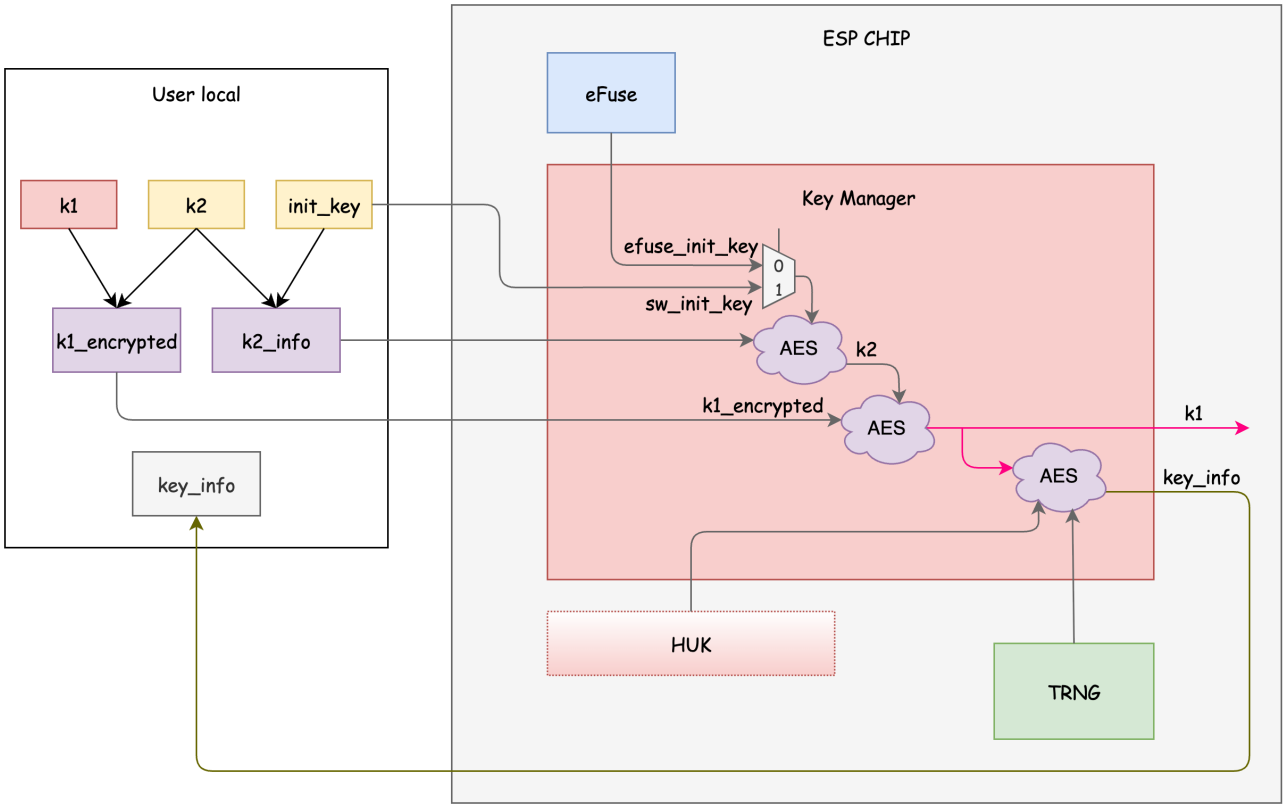


Figure 36.6-1. AES Deploy Mode

Detailed Process

1. Generate locally $k1$, $k2$, and $init_key$.
2. Calculate locally $k1_encrypted$ and $k2_info$.
3. Then complete the key deployment in Key Manager:
   (a) Write $init_key$ into the eFuse block whose purpose is EFUSE_KM_INIT_KEY, or to sw_init_key, depending on your need.
   (b) Write $k2_info$ and $k1_encrypted$ into internal memory of the Key Manager.
   (c) Wait for the Key Manager to complete the deployment.
   (d) Read key_info from the internal memory of the Key Manager by specifying the address.

Notes:

- Due to the necessity of re-deploying the key each time when the chip powers up, if key_info is to be used for key recovery upon the next power-up, ensure that key_info must persist after the chip powers off. A common method is to store it in the flash.
- AES Deploy Mode is suitable for factory production: the manufacturer provides the factory with $init_key$ and the firmware encrypted with $k1$, and store $k2_info$ and $k1_encrypted$ in the firmware.
  - In this case, it is recommended to include a private key first-time deployment process in the firmware, to store key_info. In the subsequent chip power-ups, use the recovery deployment mode, instead of deploying via AES mode each time. This helps effectively defend against eFuse slice attacks.

  - Note that this deployment method requires a trusted factory, as the factory will have access to the plaintext $init_key$ (used to be burned into eFuse key block). If the factory is untrusted and knows how to use $init_key$ to decrypt $k2_info$ into $k2$, it may result in a potential leakage of $k1$.

- When using AES Deploy Mode to deploy the key each time, it is essential to ensure that the values of $init_key$, $k2_info$, and $k1_encrypted$ are not leaked simultaneously.
  - If using $init_key$ stored in the EFUSE_KM_INIT_KEY eFuse block, $k2_info$ and $k1_encrypted$ may be transmitted in an untrusted environment. However, this requires to locally retain the $init_key$ that was written to EFUSE_KM_INIT_KEY eFuse block at the first time.
  - If using sw_init_key, the transmission of sw_init_key, $k2_info$, and $k1_encrypted$ must occur in a trusted environment.

36.6.2.3 ECDH0 Deploy Mode

Users can use this mode to deploy a negotiated private key.

This mode requires the following keys and parameters:

- HUK, generated by the HUK Generator and passed to the Key Manager.
- $k1$: root key, a private key from users.
- $k1 * G$: obtained by calculating $k1 * G$ using ECC.
- $k2 * G$: a public auxiliary key, generated by calculating $k2 * G$ using ECC.
- $k1 * k2 * G$: the negotiated private key to be deployed, generated by Key Manager using $k1 * G$ and $k2$ or by the user locally using $k1$ and $k2 * G$.
- key_info: the recovery information of $k1 * k2 * G$.

Any information disclosure during this deployment, occurring outside the users' local environment or the chip, will not affect the generation of $k1 * k2 * G$ or pose a risk of exposure. The deployment flow chart is as follows.

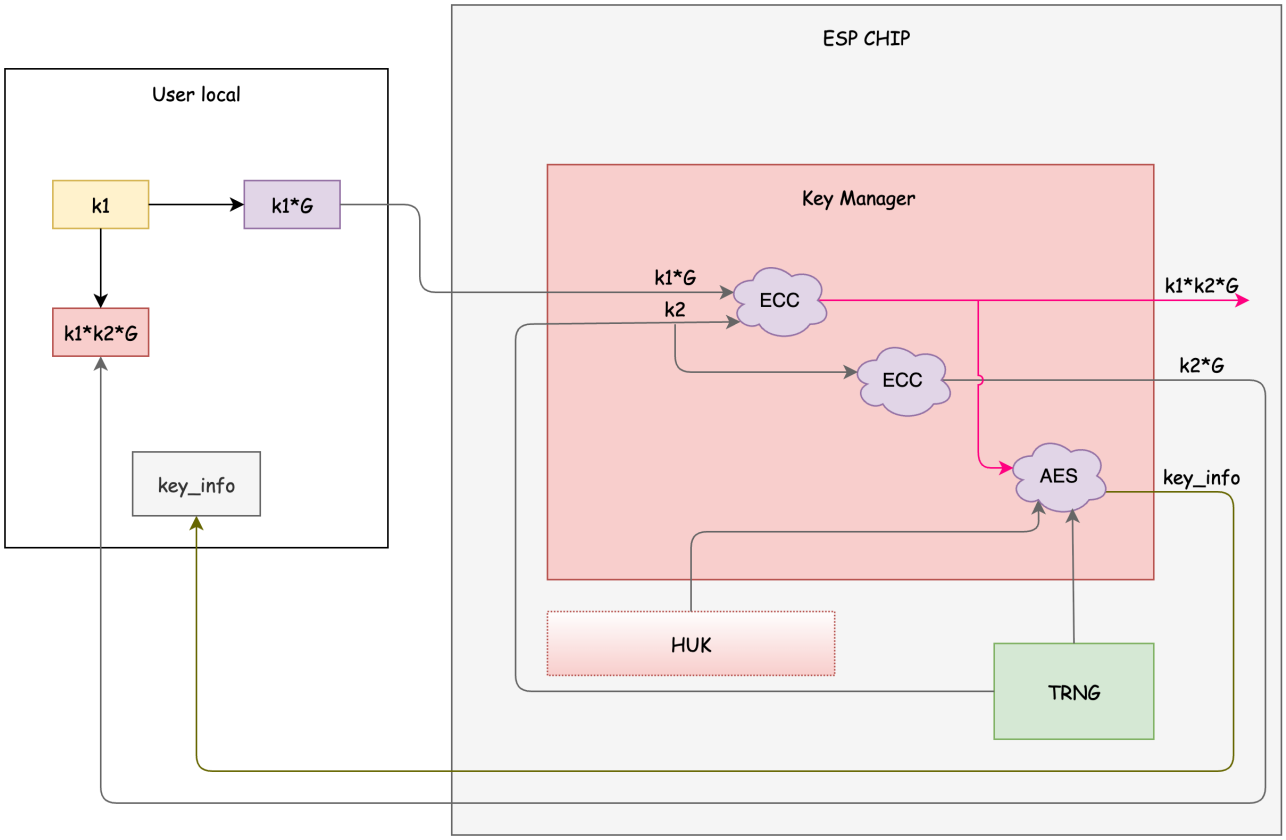


Figure 36.6-2. ECDH0 Deploy Mode

Detailed Process

1. Generate locally $k1$ and calculate $k1 * G$.
2. Complete the key deployment in Key Manager:
   (a) Write $k1 * G$ to the internal memory of the Key Manager.
   (b) Wait for the Key Manager to complete the deployment.
   (c) Read $k2 * G$ and key_info (corresponding to the negotiated private key $k1 * k2 * G$) from the internal memory of the Key Manager by specifying the address.
3. Generate the negotiated key $k1 * k2 * G$ by using $k2 * G$ together with the local private key $k1$.

Note:

- NIST-P256 curve is used in the ECDH0 mode, all the ECC point multiplication is calculated on NIST-P256 curve.
- The ECDH0 Deploy Mode ensures that the negotiated private key is generated only locally by the user and internally by the Key Manager. Any data leakage during the deployment process will not result in the leakage of the negotiated private key. Therefore, data transmission in this mode can occur in an untrusted environment.
- In ECDH0 Deploy Mode, the negotiated private key can only be obtained after running the Key Manager. This mode is not suitable for scenarios where the exact value of the private key needs to be known in advance.

36.6.2.4 ECDH1 Deploy Mode

Users can use this mode to deploy a negotiated private key.

This mode requires the following keys and parameters:

- HUK, generated by the HUK Generator and passed to the Key Manager.
- $k1$: root key, a private key from users.
- $k1 * G$: obtained by calculating $k1$ using ECC.
- $k2$: an auxiliary private key, in ECDH1 Deploy Mode only existing locally or inside Key Manager.
- $k2 * G$: an auxiliary public key, generated by using the chip supplier's software or script.
- $k2_info$: generated by using $k2$ and $init_key$, and used to recover $k2$. The calculation formula is: $k2_info = AES\{AES\{k2, rand_num\}||rand_num, init_key\}$.
- $k1 * k2 * G$: the negotiated private key to be deployed. The chip generates the negotiated private key using $k1 * G$ and $k2_info$, while the user generates the negotiated private key using $k1$ and $k2 * G$.
- key_info: the recovery information of $k1 * k2 * G$.

The deployment flow chart is as follows.

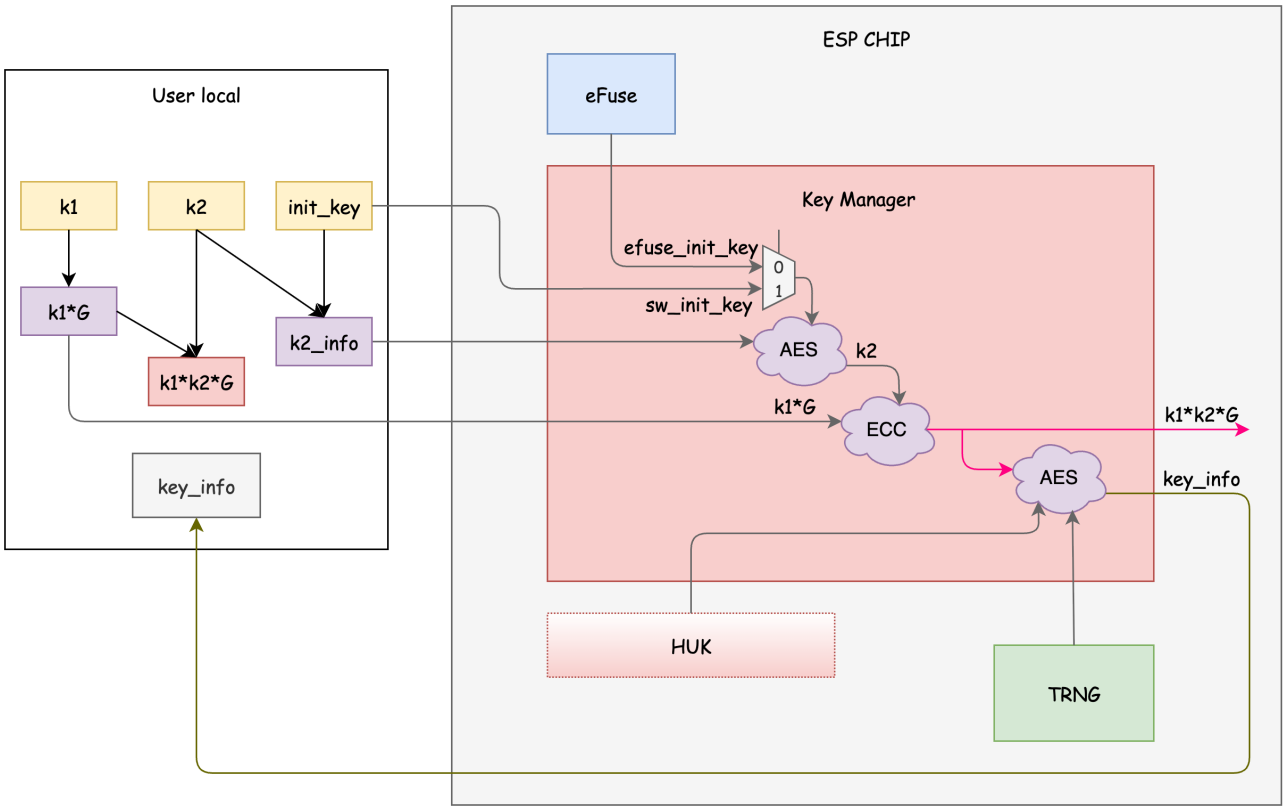


Figure 36.6-3. ECDH1 Deploy Mode

Detailed Process

1. Generate locally $k1$, $k2$ and $init_key$.
2. Calculate locally $k1 * G$, $k2 * G$, $k2_info$, and $k1 * k2 * G$.
3. Complete the key deployment in Key Manager:
   (a) Write $init_key$ into EFUSE_KM_INIT_KEY eFuse block or into sw_init_key, depending on your needs. See the notes below.

(b) Write $k1 * G$ and $k2_info$ to the internal memory of the Key Manager.

(c) Wait for the Key Manager to complete the deployment.

(d) Read $k2 * G$ and key_info (corresponding to the negotiated private key $k1 * k2 * G$) from the internal memory of the Key Manager by specifying the address.

Notes:

- NIST-P256 curve is used in the ECDH1 mode, all the ECC point multiplication is calculated on NIST-P256 curve.
- ECDH1 Deploy Mode is suitable for factory production: the manufacturer provides the factory with $init_key$ and the firmware encrypted with $k1 * k2 * G$, and store $k2_info$ and $k1 * G$ in the firmware.
  - In this case, it is recommended to include a private key first-time deployment process in the firmware, to store key_info. In the subsequent chip power-ups, use the recovery deployment mode, instead of deploying via ECDH1 mode each time. This helps effectively defend against eFuse slice attacks.
  - Note that this deployment method requires a trusted factory, as the factory will have access to the plaintext $init_key$ (used to be burned into eFuse). If the factory is untrusted and knows how to use $init_key$ to decrypt $k2_info$ into $k2$, it may result in a potential leakage of $k1 * k2 * G$.
- Similar to AES Deploy Mode, when using ECDH1 Deploy Mode to deploy the key each time, it is essential to ensure that the values of $init_key$, $k2_info$, and $k1 * G$ are not leaked simultaneously.
  - If using $init_key$ stored in EFUSE_KM_INIT_KEY eFuse block, $k2_info$ and $k1 * G$ may be transmitted in an untrusted environment. However, this requires to locally retain the $init_key$ that was written to EFUSE_KM_INIT_KEY eFuse block at the first time.
  - If using sw_init_key, the transmission of sw_init_key, $k2_info$, and $k1 * G$ must occur in a trusted environment.
- When updating keys using the ECDH1 Deploy Mode, it provides higher security performance compared to the AES Deploy Mode. Since the value of $k1 * G$ is independent of $k2$ and $init_key$, you can update the negotiated key in ECDH1 mode by updating only $k2_info$ without changing k1*G. Users can:
  - Write $k1 * G$ into the firmware in a trusted environment.
  - When updating the key using the ECDH1 Deploy Mode, transmit only $init_key$ and $k2_info$. In this case, users can deploy the key in ECDH1 mode using sw_init_key even in an untrusted environment.

36.6.2.5 Private Key Recovery Mode

After deploying the private key, users can restore it using the key_info generated during deployment. The specific process is as follows.

1. Read the key_info from the external memory and store it to the fixed address in the internal memory of Key Manager.
2. Wait for the Key Manager to complete the deployment.
3. Repeat Step 1 $\sim$ Step 2 above until all required private keys are restored.

Note that if two or more private keys with the same $key_purpose$ are restored, only the last one will take effect. For more information about $key_purpose$, see Section 4.

36.6.2.6 key_info Export Mode

After successfully deploying the private key, users can regenerate key_info and obtain it with the help of this mode.
The specific process is as follows.

1. Select the successfully deployed $key_purpose$ to enter the key_info export mode.
2. Wait for the Key Manager process to complete.
3. Read key_info from the internal memory of the Key Manager.

Note:

- This mode can be used to output different key_info for the same key.
- This mode can also only update HUK without updating the private key. The detailed operation is:
  - Update the HUK using the HUK Generation Mode.
  - Update the key_info through the key_info Export Mode.

  By such way, the chip in the next startup can use the new key_info to restore the same private key under the new HUK.

36.7 Programming Procedures

36.7.1 HUK Generator

The HUK generation process is divided into six phases:

1. **IDLE**: In this phase, users configure the mode and static parameters.
2. **PREP**: In this phase, HUK_STATE is BUSY. The HUK Generator prepares for HUK generation. Users need to wait for this phase to complete. The HUK Generator will automatically proceed to the LOAD phase.
3. **LOAD**: In this phase, users store the HUK information into the internal memory of the HUK Generator based on the selected mode.
4. **PROC**: In this phase, HUK_STATE is BUSY. The HUK Generator performs the HUK generation process. Users need to wait for the phase to complete. The HUK Generator will automatically proceed to the GAIN phase.
5. **GAIN**: In this phase, users read the required HUK information from the internal memory of the HUK Generator based on the selected mode.
6. **POST**: In this phase, HUK_STATE is BUSY. The HUK Generator completes the follow-up tasks for HUK generation. Users need to wait for this phase to complete. The HUK Generator will automatically return to the IDLE phase.

36.7.1.1 IDLE Phase

Users check HUK_STATE to confirm that the HUK Generator is in the IDLE phase. In the IDLE phase, users need to perform the following configurations.

1. **Set the Working Mode:** Configure the working mode of the HUK Generator as specified in Table 36.6-1.
2. **Transition to PREP Phase:** Set HUK_START to complete the IDLE phase and enter the PREP phase.

36.7.1.2 PREP Phase

Users need to wait for the PREP phase to complete. This can be accomplished using one of the following methods:

- **Monitor HUK_STATE:** Continuously check HUK_STATE until it is no longer BUSY.
- **Use Interrupts:** Enable the interrupt HUK_PREP_DONE_INT and handle the completion in the interrupt service routine.

36.7.1.3 LOAD Phase

Users check HUK_STATE to confirm that the HUK Generator is in the LOAD phase. In the LOAD phase, users need to perform the following configurations.

1. **Configure Based on Mode**: Depending on the mode of the HUK Generator, configure as follows:
   - **HUK Generation Mode:** No additional configuration is required.
   - **HUK Recovery Mode:** Write the HUK recovery information (huk_info, 165 words) into HUK_INFO_MEM.

   Note:
   If multiple huk_info have been generated and stored in external memory previously, using any huk_info in the HUK Recovery Mode will restore its corresponding HUK.

2. **Transition to PROC Phase:** Set the bit HUK_CONTINUE to 1 to complete the LOAD phase and enter the PROC phase.

36.7.1.4 PROC Phase

Users need to wait for the PROC phase to complete. This can be done using one of the following methods.

- **Monitor HUK_STATE:** Continuously check HUK_STATE until it is no longer BUSY.
- **Use Interrupts:** Enable the interrupt HUK_PROC_DONE_INT and handle the completion in the interrupt service routine.

36.7.1.5 GAIN Phase

Users check HUK_STATE to confirm that the HUK Generator is in the GAIN phase. In the GAIN phase, users extract the required key information based on the selected mode of the HUK Generator.

1. **Extract Key Information:**

- **HUK Generation Mode:** Read huk_info (165 words) from HUK_INFO_MEM.
- **HUK Recovery Mode:** Update huk_info (165 words), depending on HUK_UPDATE_REQ:
  - 0: Do not need to update huk_info, or read any information.
  - 1: Must update huk_info by reading a new value (165 words) from HUK_INFO_MEM.

2. **Transition to POST Phase:** Set the bit HUK_CONTINUE to complete the GAIN phase and enter the POST phase.

36.7.1.6 POST Phase

Users need to wait for the POST phase to complete. This can be done using one of the following methods.

- **Monitor HUK_STATE:** Continuously check HUK_STATE until it is no longer BUSY.
- **Use Interrupts:** Enable the interrupt HUK_POST_DONE_INT and handle the completion in the interrupt service routine.

36.7.2 Key Manager

The key deployment process of the Key Manager is divided into six phases:

1. **IDLE**: In this phase, users configure the mode, key usage, and static parameters.
2. **PREP**: In this phase, KEYMNG_STATE is BUSY. The Key Manager performs the necessary preparations for key deployment. Users need to wait for this phase to complete. The Key Manager will automatically proceed to the LOAD phase.
3. **LOAD**: In this phase, users store the key information into the internal memory of the Key Manager based on the selected mode.
4. **PROC**: In this phase, KEYMNG_STATE is BUSY. The Key Manager performs the key deployment process. Users need to wait for this phase to complete. The Key Manager will automatically proceed to the GAIN phase.
5. **GAIN**: In this phase, users read the required key information from the internal memory of the Key Manager based on the selected mode.
6. **POST**: In this phase, KEYMNG_STATE is BUSY. The Key Manager completes the follow-up tasks for key deployment. Users need to wait for this phase to complete. The Key Manager will automatically return to the IDLE phase.

36.7.2.1 IDLE Phase

Users check KEYMNG_STATE to confirm that the Key Manager is in the IDLE phase. In the IDLE phase, users need to perform the following configurations.

1. **Complete HUK Generation Process:**
   (a) Select one of the following two processes, depending on whether the HUK has been generated or not:
   - **If HUK has not been generated:** Follow the HUK generation process described in Section 36.7.1.

- **If HUK has already been generated:** Skip this HUK generation process.

(b) **Verify HUK Validity:** Check KEYMNG_HUK_VALID:

- 0: the HUK is invalid. Return to Step 1a until its value is 1.
- 1: the HUK is valid.

2. **Initialize TRNG correctly:** The user needs to make sure the TRNG is working correctly.
3. **Configure Static Parameters:** Set the following parameters, including eFuse bits, registers and their corresponding locks:
   - **EFUSE_KM_INIT_KEY eFuse block:** 256-bit initial key used by the Key Manager in AES Deploy Mode and ECDH1 Deploy Mode.
   - **EFUSE_KM_DEPLOY_ONLY_ONCE:** Controls whether the corresponding private key can only be deployed once after a power-on.
     - **Bit[0]:** Configures whether ECDSA Digital Signature Peripheral (ECDSA_DS) key can be deployed only once.
       * 0: ECDSA_DS key can be deployed more than once.
       * 1: ECDSA_DS key can be deployed only once.
     - **Bit[1]**: Configures whether XTS key can be deployed only once.
       * 0: XTS key can be deployed more than once.
       * 1: XTS key can be deployed only once.
   - **EFUSE_FORCE_USE_KEY_MANAGER_KEY:** Controls whether the corresponding private key must come from the Key Manager.
     - **Bit[0]:** Configures whether ECDSA_DS key must come from the Key Manager.
       * 0: ECDSA_DS key does not have to come from the Key Manager.
       * 1: ECDSA_DS key must come from the Key Manager.
     - **Bit[1]:** Configures whether flash encryption key must come from the Key Manager.
       * 0: Flash key does not have to come from the Key Manager.
       * 1: Flash key must come from the Key Manager.
     - **Bit[2]**: Configures whether HMAC key must come from the Key Manager.
       * 0: HMAC key does not have to come from the Key Manager.
       * 1: HMAC key must come from the Key Manager.
     - **Bit[3]**: Configures whether RSA Digital Signature Peripheral (RSA_DS) key must come from the Key Manager.
       * 0: RSA_DS key does not have to come from the Key Manager.
       * 1: RSA_DS key must come from the Key Manager.
     - **Bit[4]**: Configures whether PSRAM key must come from the Key Manager.
       * 0: PSRAM key does not have to come from the Key Manager.

    * 1: PSRAM key must come from the Key Manager.

- **KEYMNG_USE_EFUSE_KEY:** Configures whether the key bit comes from eFuse block, valid only when the corresponding bit in EFUSE_FORCE_USE_KEY_MANAGER_KEY is 0.
  - **Bit[0]:** Configures whether ECDSA_DS key comes from eFuse block.
    * 0: No effect.
    * 1: ECDSA_DS key comes from eFuse block, valid only when bit[0] in EFUSE_FORCE_USE_KEY_MANAGER_KEY is 0.
  - **Bit[1]:** Configures whether flash encryption key comes from eFuse block.
    * 0: No effect.
    * 1: Flash key comes from eFuse block, valid only when bit[1] in EFUSE_FORCE_USE_KEY_MANAGER_KEY is 0.
  - **Bit[2]:** Configures whether HMAC key comes from eFuse block.
    * 0: No effect.
    * 1: Flash key comes from eFuse block, valid only when bit[2] in EFUSE_FORCE_USE_KEY_MANAGER_KEY is 0.
  - **Bit[3]:** Configures whether RSA Digital Signature Peripheral (RSA_DS) key comes from eFuse block.
    * 0: No effect.
    * 1: RSA_DS key comes from eFuse block, valid only when bit[3] in EFUSE_FORCE_USE_KEY_MANAGER_KEY is 0.
  - **Bit[4]:** Configures whether PSRAM key comes from eFuse block.
    * 0: No effect.
    * 1: PSRAM key comes from eFuse block, valid only when bit[4] in EFUSE_FORCE_USE_KEY_MANAGER_KEY is 0.
- **EFUSE_FORCE_DISABLE_SW_INIT_KEY:** Controls whether the use of software-source $init_key$ (sw_init_key) is permanently disabled.
  - 0: Not disabled.
  - 1: Permanently disabled.
- **KEYMNG_USE_SW_INIT_KEY:** Configures whether or not to use the software configured sw_init_key instead of the eFuse block whose purpose is EFUSE_KM_INIT_KEY, valid only when EFUSE_FORCE_DISABLE_SW_INIT_KEY is 0.
  - 0: Use the eFuse block whose purpose is EFUSE_KM_INIT_KEY.
  - 1: Use the software configured sw_init_key.
- **EFUSE_KM_RND_SWITCH_CYCLE:** Configures the interval for the Key Manager to switch random numbers. It is recommended to set this switch cycle to the number of Key Manager clock cycles corresponding to the TRNG module clock cycles.
  - 0: Switch cycle is controlled by KEYMNG_RND_SWITCH_CYCLE.

  - 1: 8 cycles of Key Manager clock.
  - 2: 16 cycles of Key Manager clock.
  - 3: 32 cycles of Key Manager clock.

- **KEYMNG_RND_SWITCH_CYCLE:** Configures the interval for the Key Manager to switch random numbers, valid only when EFUSE_KM_RND_SWITCH_CYCLE is 0. It is recommended to set this switch cycle to the number of Key Manager clock cycles corresponding to the TRNG module clock cycles.

4. **Configure Working Mode:** Configure KEYMNG_KGEN_MODE to complete the working mode configuration of the Key Manager as described in Section 36.6.2.
   - 0: Random Deploy Mode
   - 1: AES Deploy Mode
   - 2: ECDH0 Deploy Mode
   - 3: ECDH1 Deploy Mode
   - 4: Private Key Recovery mode
   - 5: key_info Export Mode
   - Others: Not valid
5. **Configure Target Key:** Configure KEYMNG_KEY_PURPOSE to the target $key_purpose$:
   - 1: ecdsa_key_192
   - 2: ecdsa_key_256
   - 3: flash_256_1_key
   - 4: flash_256_2_key
   - 5: flash_128_key
   - 6: hmac_key
   - 7: ds_key
   - 8: psram_256_1_key
   - 9: psram_256_2_key
   - 10: psram_128_key
   - 11: ecdsa_key_384_l
   - 12: ecdsa_key_384_h
   - Other values: Not valid
6. **Transition to PREP Phase:** Configure KEYMNG_START to complete the IDLE phase and enter the PREP phase.

36.7.2.2 PREP Phase

Users need to wait for the PREP phase to complete. This can be done using one of the following methods:

- **Monitor KEYMNG_STATE:** Continuously check KEYMNG_STATE until it is no longer BUSY.
- **Use Interrupt:** Enable the interrupt KEYMNG_PREP_DONE_INT and handle the completion in the interrupt service routine.

36.7.2.3 LOAD Phase

Users check KEYMNG_STATE to confirm that the Key Manager is in the LOAD phase. In the LOAD phase, users need to perform the following configurations.

1. **Configure Based on Mode**: Depending on the mode of the Key Manager, configure as follows:
   - **Random Deploy Mode:** No additional configuration is required.
   - **AES Deploy Mode:**
     (a) If using software to configure $init_key$: write sw_init_key (256 bits) into KEYMNG_SW_INIT_KEY_MEM.
     (b) Write $k2_info$ (512 bits) into KEYMNG_ASSIST_INFO_MEM.
     (c) Write $k1_encrypted$ (256 bits) into KEYMNG_PUBLIC_INFO_MEM.
   - **ECDH0 Deploy Mode:**
     (a) Write $k1 * G$ (512 bits) into KEYMNG_PUBLIC_INFO_MEM.
   - **ECDH1 Deploy Mode:**
     (a) If the software configured init_key is used: write sw_init_key (256 bits) into KEYMNG_SW_INIT_KEY_MEM.
     (b) Write $k2_info$ (512 bits) into KEYMNG_ASSIST_INFO_MEM.
     (c) Write $k1 * G$ (512 bits) into KEYMNG_PUBLIC_INFO_MEM.
   - **Private Key Recovery Mode:**
     (a) Write key_info (512 bits) into KEYMNG_ASSIST_INFO_MEM.
   - key_info **Export Mode:** No additional configuration is required.
2. **Transition to PROC Phase:** Configure KEYMNG_CONTINUE to complete the LOAD phase and enter the PROC phase.

36.7.2.4 PROC Phase

Users need to wait for the PROC phase to complete. This can be done using one of the following methods.

- **Monitor KEYMNG_STATE:** Continuously check KEYMNG_STATE until it is no longer BUSY.
- **Use Interrupt:** Enable the interrupt KEYMNG_PROC_DONE_INT and handle the completion in the interrupt service routine.

36.7.2.5 GAIN Phase

Users check KEYMNG_STATE to confirm that the Key Manager is in the GAIN phase. In the GAIN phase, users extract the required key information based on the selected mode of the Key Manager.

1. **Extract Key Information:**
   - **Random Deploy Mode:**
     (a) Read key_info (512 bits) from KEYMNG_PUBLIC_INFO_MEM.
   - **AES Deploy Mode:**
     (a) Read key_info (512 bits) from KEYMNG_PUBLIC_INFO_MEM.
   - **ECDH0 Deploy Mode:**
     (a) Read key_info (512 bits) from KEYMNG_PUBLIC_INFO_MEM.
     (b) Read $k2 * G$ (512 bits) from KEYMNG_ASSIST_INFO_MEM.
   - **ECDH1 Deploy Mode:**
     (a) Read key_info (512 bits) from KEYMNG_PUBLIC_INFO_MEM.
   - **Private Key Recovery Mode:** no need to read any information.
   - key_info **Export Mode:**
     (a) Read key_info (512 bits) from KEYMNG_PUBLIC_INFO_MEM.
2. **Transition to POST Phase:** Configure KEYMNG_CONTINUE to complete the GAIN phase and enter the POST phase.

36.7.2.6 POST Phase

Users need to wait for the POST phase to complete. This can be done using one of the following methods.

- **Monitor KEYMNG_STATE:** Continuously check KEYMNG_STATE until it is no longer BUSY.
- **Use Interrupt:** Enable the interrupt KEYMNG_POST_DONE_INT and handle the completion in the interrupt service routine.

36.8 Programming Examples

Based on the previous sections, we can outline a comprehensive chip deployment process. This section explains the operation procedures for the HUK Generator and Key Manager during two typical chip startup scenarios.

Note:
In this section, all configuration processes are described in a simplified manner. For detailed instructions, please refer to Section 36.7.

36.8.1 Software Process in Chip Private Key Deploy Phase

The following steps outline the process for booting up the chip for the first time and deploying the private key for encrypting and decrypting external memory.

1. **Enter Download Mode:** The chip enters the download mode.
2. **Configure HUK Generator:** Since it is the first time to start the chip, configure the HUK Generator to enter the HUK Generation Mode.
   (a) Configure the IDLE phase and set the working mode as HUK Generation Mode as specified in Table 36.6-1, then configure HUK_START.
   (b) After entering the LOAD phase, configure HUK_CONTINUE.
   (c) After entering GAIN phase, read out huk_info, then configure HUK_CONTINUE.
   (d) Wait for the HUK Generator to return to the IDLE phase.
3. **Set HUK Generation State:** Write EFUSE_KM_HUK_GEN_STATE to make the total number of bits set to 1 is odd, so that the HUK Generator will work only in HUK Recovery Mode during subsequent startups.
4. **Configure Key Deployment:** Take the key deployment in ECDHO Deploy Mode as an example.
   (a) Configure the IDLE phase, set KEYMNG_KEY_PURPOSE as XTS, and configure KEYMNG_START.
   (b) After entering the LOAD phase, write the locally generated $k1 * G$ to the Key Manager, then configure KEYMNG_CONTINUE.
   (c) After entering the GAIN phase, read out key_info and $k2 * G$, then configure KEYMNG_CONTINUE.
   (d) Wait for the Key Manager to return to the IDLE phase.
5. **Upload Key:** Upload $k2 * G$ to the user through the communication interface.
6. **Generate and Encrypt Data:**
   - Generate the negotiated private key $k1 * k2 * G$ based on the obtained $k2 * G$ and the local $k1$.
   - Encrypt data locally and write the ciphertext to flash.

36.8.2 Software Process in Chip Normal Startup Phase

Once the chip has been deployed, the complete process for a normal startup is as follows.

1. **Start in SPI Boot Mode:**

   The chip begins in SPI boot mode and first enters the unencrypted program segment.
2. **Configure HUK Recovery Mode:**

   Since the HUK has been generated, configure the HUK Recovery Mode as follows:
   (a) Configure the IDLE phase and set the working mode as HUK Recovery Mode as specified in Table 36.6-1, then configure HUK_START.
   (b) After entering the LOAD phase, write huk_info into the HUK Generator, then configure HUK_CONTINUE.
   (c) After entering GAIN phase, configure HUK_CONTINUE.

(d) Wait for the HUK Generator to return to the IDLE phase.

3. **Configure Key Manager for Private Key Recovery Mode:**

   (a) Configure the IDLE phase and set KEYMNG_KEY_PURPOSE as flash_128_key, then configure KEYMNG_START.

   (b) After entering the LOAD phase, write key_info into the Key Manager, then configure KEYMNG_CONTINUE.

   (c) After entering the GAIN phase, configure KEYMNG_CONTINUE.

   (d) Wait for the Key Manager to return to the IDLE phase.

4. **Enter Encrypted Program Segment:**

   At this point, the encryption/decryption private key in the external memory has taken effect, and the system enters the encrypted program segment.

36.9 Interrupts

ESP32-P4's Key Manager and HUK Generator can generate the following interrupt signals that will be sent to the Interrupt Matrix.

- KEYMNG_INTR
- HUK_INTR

There are several internal interrupt sources from Key Manager and HUK Generator that can generate the above interrupt signals. The interrupt sources from Key Manager and HUK Generator are listed with their trigger conditions and the resulted interrupt signals in Table 36.9-1.

Table 36.9-1. Key Manager and HUK Generator's Internal Interrupt Sources

Internal Interrupt Source	Trigger Condition	Interrupt Signal
KEYMNG_PREP_DONE_INT	Completion of Key Manager's PREP phase	KEYMNG_INTR
KEYMNG_PROC_DONE_INT	Completion of Key Manager's PROC phase	KEYMNG_INTR
KEYMNG_POST_DONE_INT	Completion of Key Manager's POST phase	KEYMNG_INTR
HUK_PREP_DONE_INT	Completion of HUK Generator's PREP phase	HUK_INTR
HUK_PROC_DONE_INT	Completion of HUK Generator's PROC phase	HUK_INTR
HUK_POST_DONE_INT	Completion of HUK Generator's POST phase	HUK_INTR

Note:
For definitions of *interrupt*, *interrupt signal*, *interrupt source*, and their correlations, please refer to Chapter 13 *Interrupt Matrix* > Section 13.2 *Interrupt Terminology in ESP32-P4*.

36.10 Memory Blocks

The addresses in this section are relative to Key Manager or HUK Generator base address provided in Table 9.3-2 in Chapter 9 *System and Memory*.

Table 36.10-1. Key Manager Memory Blocks

Name	Description	Size (byte)	Starting Address	Ending Address	Access
KEYMNG_ASSIST_INFO_MEM	Stores assistant information	64	0x0100	0x013F	Varies
KEYMNG_PUBLIC_INFO_MEM	Stores public information	64	0x0140	0x017F	Varies
KEYMNG_SW_INIT_KEY_MEM	Stores sw_init_key	32	0x0180	0x019F	Varies

Table 36.10-2. HUK Generator Memory Blocks

Name	Description	Size (byte)	Starting Address	Ending Address	Access
HUK_INFO_MEM	Stores huk_info	660	0x0100	0x0393	Varies

36.11 Register Summary

36.11.1 HUK Register Summary

The addresses in this section are relative to HUK Generator base address provided in Table 9.3-2 in Chapter 9 *System and Memory*.

The abbreviations given in Column **Access** are explained in Section *Access Types for Registers*.

Name	Description	Address	Access
Clock Gate Register			
HUK_CLK_REG	Clock gate control register	0x0004	R/W
Interrupt Registers			
HUK_INT_RAW_REG	Interrupt raw register (valid in level)	0x0008	RO/WTC/SS
HUK_INT_ST_REG	Interrupt status register	0x000C	RO
HUK_INT_ENA_REG	Interrupt enable register	0x0010	R/W
HUK_INT_CLR_REG	Interrupt clear register	0x0014	WT
Configuration Register			
HUK_CONF_REG	Configuration register	0x0020	R/W
Control Register			
HUK_START_REG	Control register	0x0024	WT
State Register			
HUK_STATE_REG	HUK Generator state register	0x0028	RO
Result Register			
HUK_STATUS_REG	HUK status register	0x0034	RO
Version Register			
HUK_DATE_REG	Version control register	0x00FC	R/W

36.11.2 Key Manager Register Summary

The addresses in this section are relative to Key Manager base address provided in Table 9.3-2 in Chapter 9 *System and Memory*.

The abbreviations given in Column **Access** are explained in Section *Access Types for Registers*.

Name	Description	Address	Access
Clock Gate Register			
KEYMNG_CLK_REG	Clock gate control register	0x0004	R/W
Interrupt Registers			
KEYMNG_INT_RAW_REG	Interrupt raw register (valid in level)	0x0008	RO/WTC/SS
KEYMNG_INT_ST_REG	Interrupt status register	0x000C	RO
KEYMNG_INT_ENA_REG	Interrupt enable register	0x0010	R/W
KEYMNG_INT_CLR_REG	Interrupt clear register	0x0014	WT
Static Configuration Registers			
KEYMNG_STATIC_REG	Static configuration register	0x0018	R/W
KEYMNG_LOCK_REG	Static configuration lock register	0x001C	R/W1

Name	Description	Address	Access
Configuration register			
KEYMNG_CONF_REG	Configuration register	0x0020	R/W
Control Register			
KEYMNG_START_REG	Control register	0x0024	WT
State Register			
KEYMNG_STATE_REG	State register	0x0028	RO
Result Registers			
KEYMNG_RESULT_REG	Operation result register	0x002C	RO/SS
KEYMNG_KEY_VLD_REG	Key status register	0x0030	RO
KEYMNG_HUK_VLD_REG	HUK status register	0x0034	RO
Version Register			
KEYMNG_DATE_REG	Version control register	0x00FC	R/W

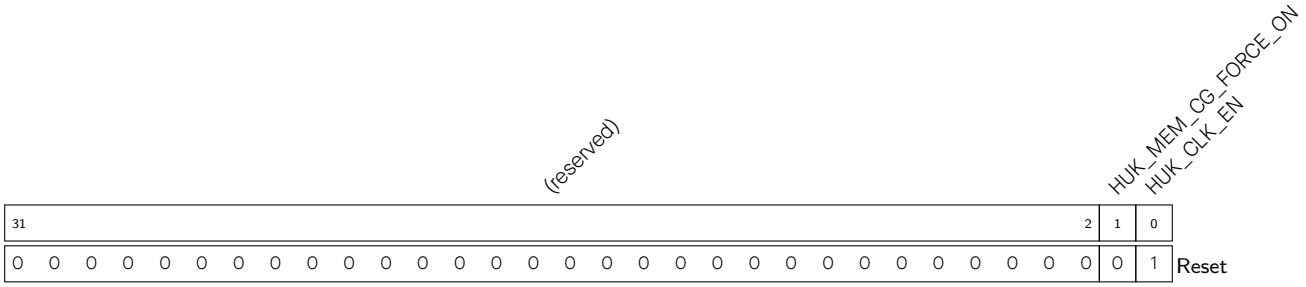
36.12 Registers

36.12.1 HUK Registers

The addresses in this section are relative to HUK base address provided in Table 9.3-2 in Chapter 9 *System and Memory*.

For how to program reserved fields, please refer to Section IX .

Register 36.1. HUK_CLK_REG (0x0004)



HUK_CLK_EN Configures whether or not to force on register clock gate.

0: Not force on

1: Force on

(R/W)

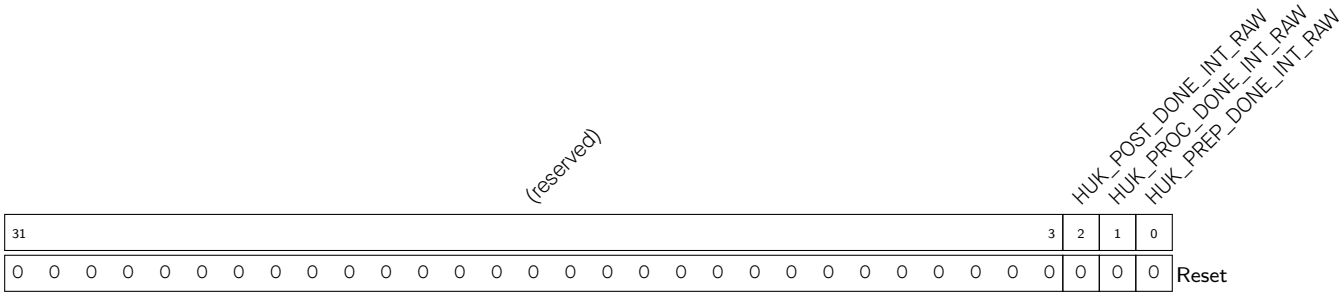
HUK_MEM_CG_FORCE_ON Configures whether or not to force on memory clock gate.

0: Not force on

1: Force on

(R/W)

Register 36.2. HUK_INT_RAW_REG (0x0008)



HUK_PREP_DONE_INT_RAW The raw interrupt status of HUK_PREP_DONE_INT interrupt.

(RO/WTC/SS)

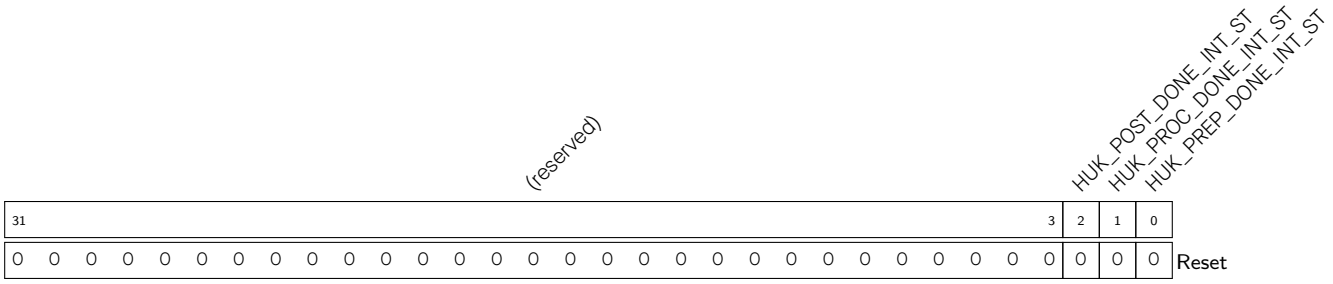
HUK_PROC_DONE_INT_RAW The raw interrupt status of HUK_PROC_DONE_INT interrupt.

(RO/WTC/SS)

HUK_POST_DONE_INT_RAW The raw interrupt status of HUK_POST_DONE_INT interrupt.

(RO/WTC/SS)

Register 36.3. HUK_INT_ST_REG (0x000C)

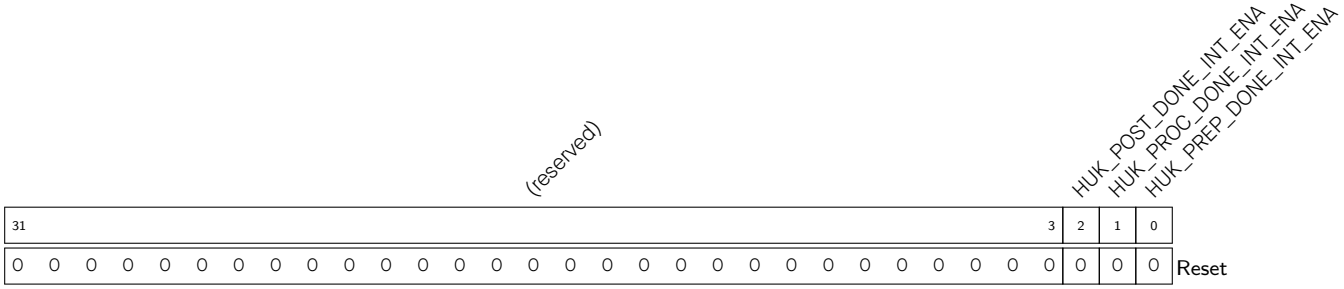


HUK_PREP_DONE_INT_ST The masked interrupt status of HUK_PREP_DONE_INT interrupt. (RO)

HUK_PROC_DONE_INT_ST The masked interrupt status of HUK_PROC_DONE_INT interrupt. (RO)

HUK_POST_DONE_INT_ST The masked interrupt status of HUK_POST_DONE_INT interrupt. (RO)

Register 36.4. HUK_INT_ENA_REG (0x0010)

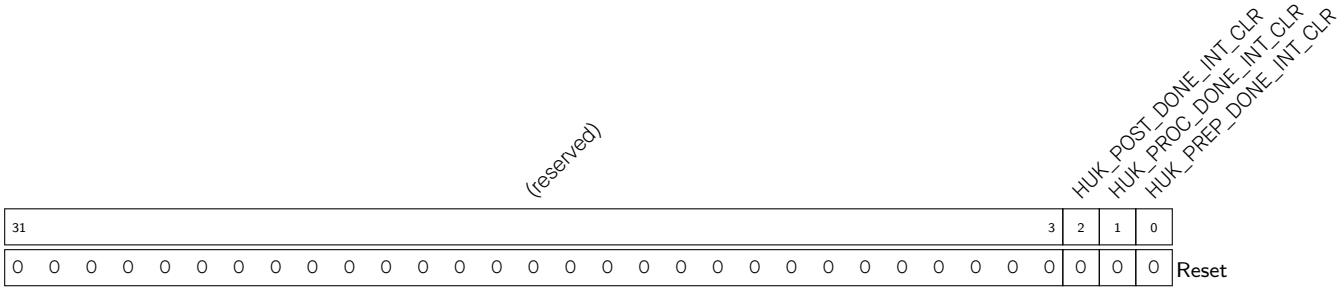


HUK_PREP_DONE_INT_ENA Write 1 to enable the HUK_PREP_DONE_INT interrupt. (R/W)

HUK_PROC_DONE_INT_ENA Write 1 to enable the HUK_PROC_DONE_INT interrupt. (R/W)

HUK_POST_DONE_INT_ENA Write 1 to enable the HUK_POST_DONE_INT interrupt. (R/W)

Register 36.5. HUK_INT_CLR_REG (0x0014)

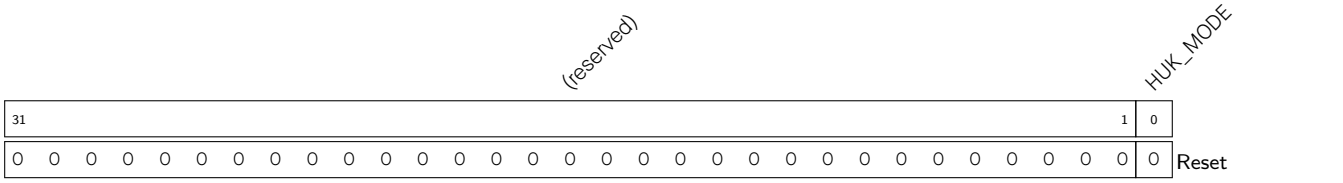


HUK_PREP_DONE_INT_CLR Write 1 to clear the HUK_PREP_DONE_INT interrupt. (WT)

HUK_PROC_DONE_INT_CLR Write 1 to clear the HUK_PROC_DONE_INT interrupt. (WT)

HUK_POST_DONE_INT_CLR Write 1 to clear the HUK_POST_DONE_INT interrupt. (WT)

Register 36.6. HUK_CONF_REG (0x0020)



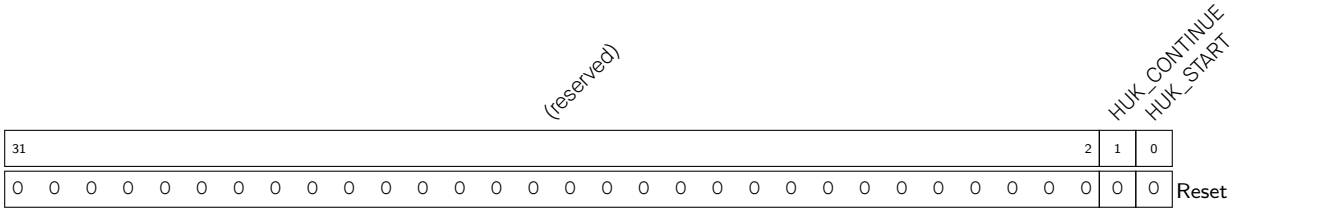
HUK_MODE Configures HUK mode.

0: HUK Recovery Mode

1: HUK Generation Mode

(R/W)

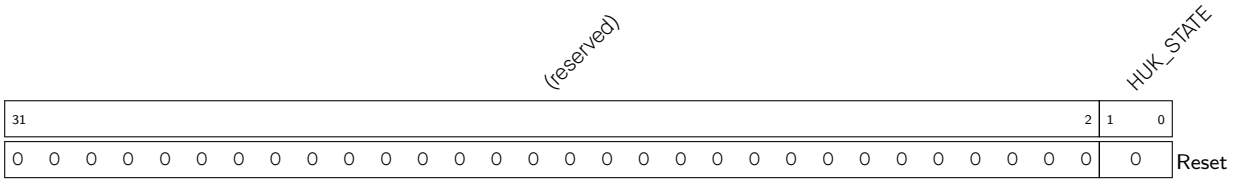
Register 36.7. HUK_START_REG (0x0024)



HUK_START Write 1 to start HUK Generator at IDLE phase. (WT)

HUK_CONTINUE Write 1 to continue HUK Generator operation at LOAD/GAIN phase. (WT)

Register 36.8. HUK_STATE_REG (0x0028)



HUK_STATE Represents the state of HUK Generator.

0: IDLE

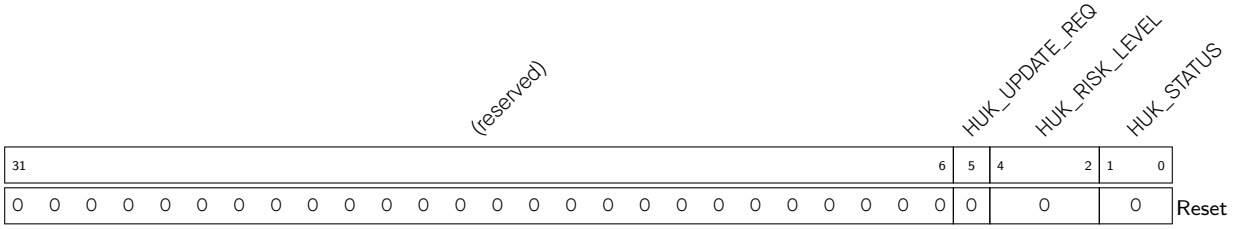
1: LOAD

2: GAIN

3: BUSY

(RO)

Register 36.9. HUK_STATUS_REG (0x0034)



HUK_STATUS Represents the HUK generation status.

0: HUK is not generated.

1: HUK is generated and valid.

2: HUK is generated but invalid.

3: Reserved.

(RO)

HUK_RISK_LEVEL Represents the risk level of HUK.

0∼6: The higher the risk level is, the more error bits there are in the PUF SRAM.

7: Error Level, HUK is invalid.

(RO)

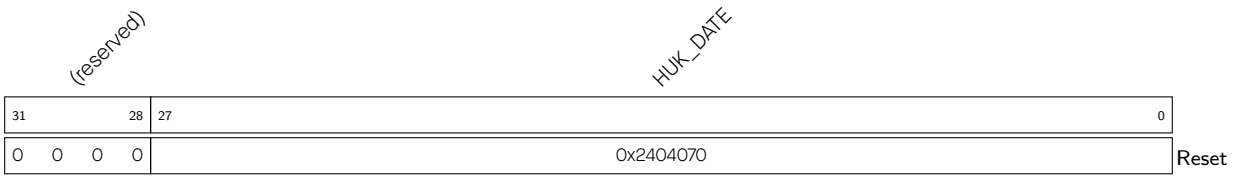
HUK_UPDATE_REQ Represents the update request of huk_info.

0: User can update huk_info according to the risk level.

1: The huk_info is expired, and user need to update it.

(RO)

Register 36.10. HUK_DATE_REG (0x00FC)



HUK_DATE Version control register.

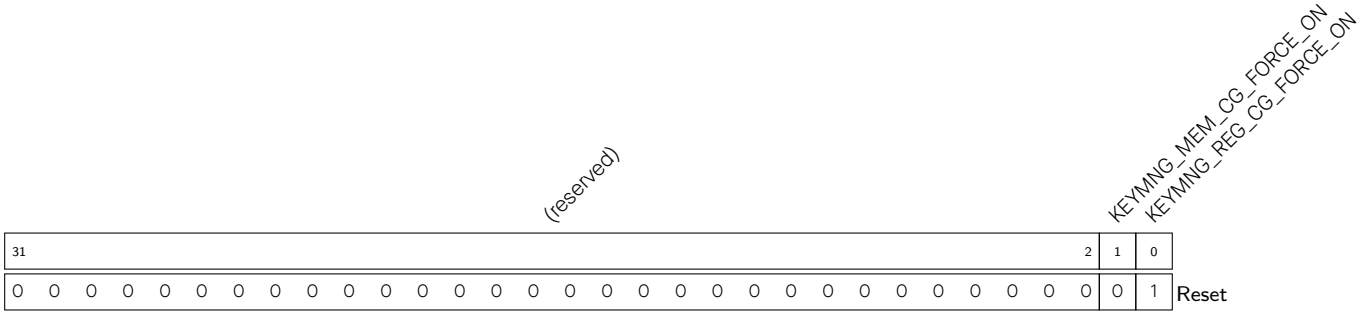
(R/W)

36.12.2 Key Manager Registers

The addresses in this section are relative to Key Manager base address provided in Table 9.3-2 in Chapter 9 *System and Memory*.

For how to program reserved fields, please refer to Section IX .

Register 36.11. KEYMNG_CLK_REG (0x0004)



KEYMNG_REG_CG_FORCE_ON Configures whether or not to force on register clock gate.

0: Not force on

1: Force on

(R/W)

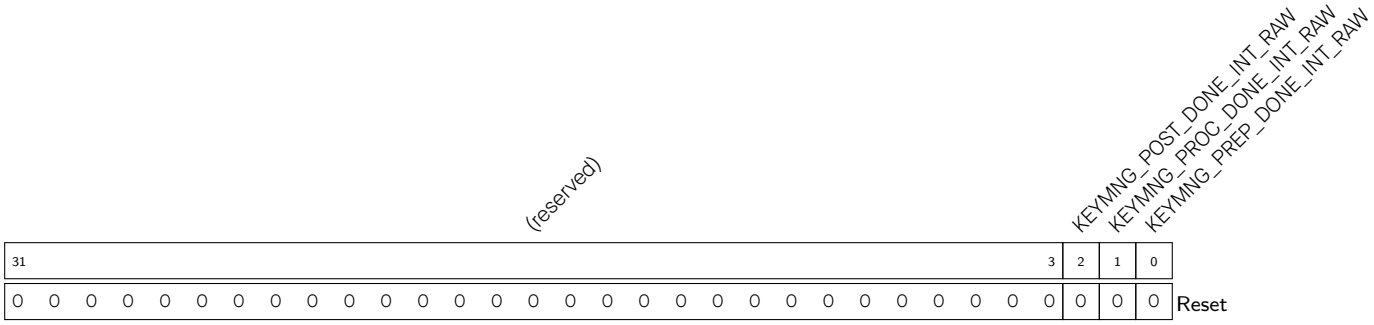
KEYMNG_MEM_CG_FORCE_ON Configures whether or not to force on memory clock gate.

0: Not force on

1: Force on

(R/W)

Register 36.12. KEYMNG_INT_RAW_REG (0x0008)

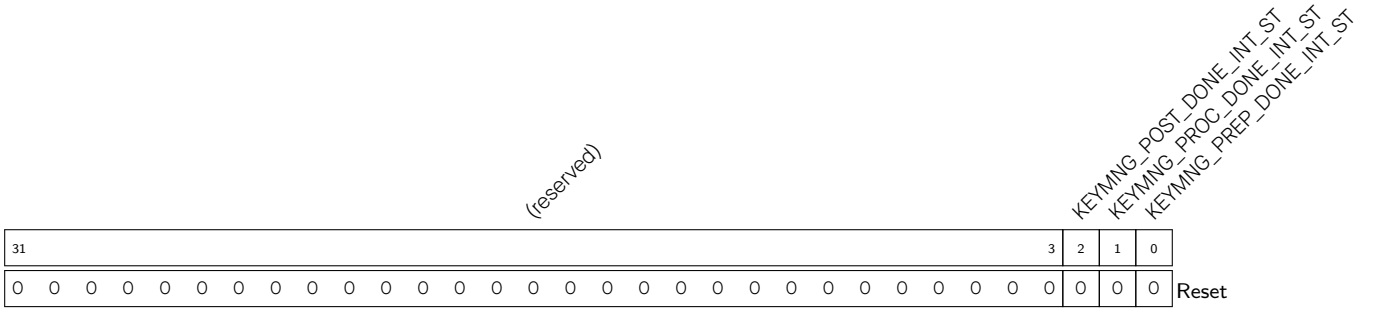


KEYMNG_PREP_DONE_INT_RAW The raw interrupt status of KEYMNG_PREP_DONE_INT interrupt. (RO/WTC/SS)

KEYMNG_PROC_DONE_INT_RAW The raw interrupt status of KEYMNG_PROC_DONE_INT interrupt. (RO/WTC/SS)

KEYMNG_POST_DONE_INT_RAW The raw interrupt status of KEYMNG_POST_DONE_INT interrupt. (RO/WTC/SS)

Register 36.13. KEYMNG_INT_ST_REG (0x000C)

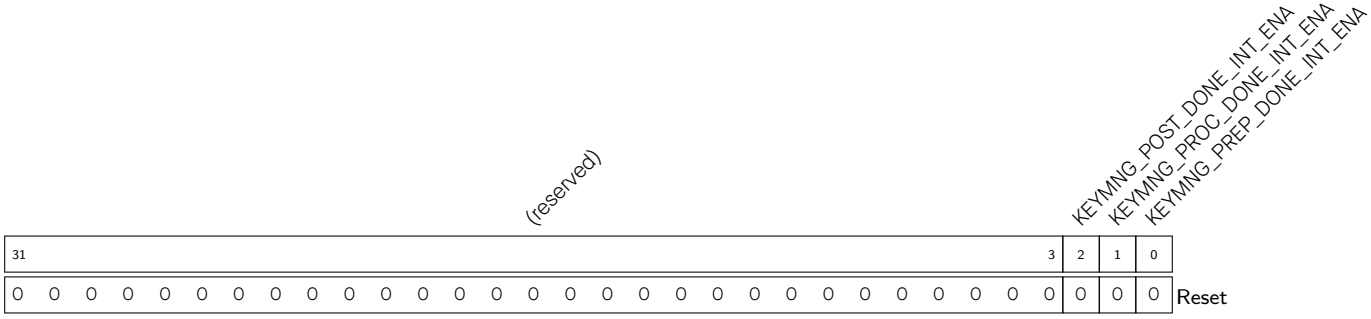


KEYMNG_PREP_DONE_INT_ST The masked interrupt status of KEYMNG_PREP_DONE_INT interrupt.
(RO)

KEYMNG_PROC_DONE_INT_ST The masked interrupt status of KEYMNG_PROC_DONE_INT interrupt.
(RO)

KEYMNG_POST_DONE_INT_ST The masked interrupt status of KEYMNG_POST_DONE_INT interrupt.
(RO)

Register 36.14. KEYMNG_INT_ENA_REG (0x0010)

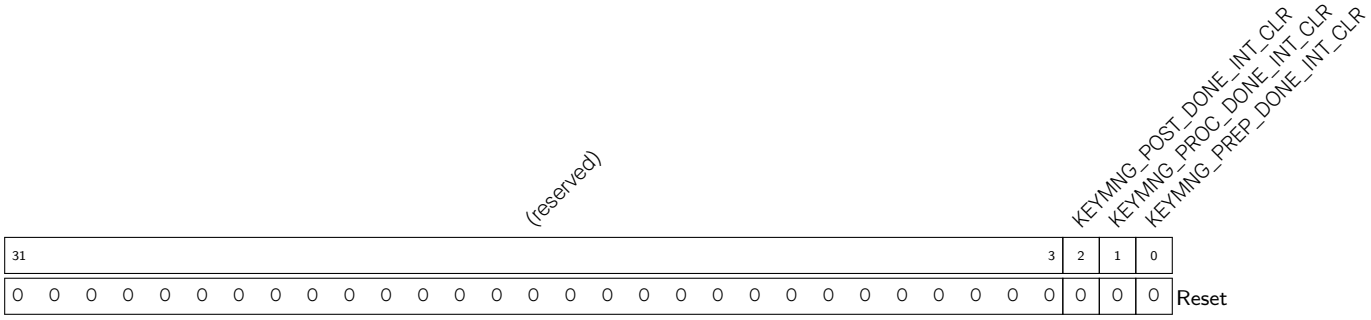


KEYMNG_PREP_DONE_INT_ENA Write 1 to enable the KEYMNG_PREP_DONE_INT interrupt. (R/W)

KEYMNG_PROC_DONE_INT_ENA Write 1 to enable the KEYMNG_PROC_DONE_INT interrupt. (R/W)

KEYMNG_POST_DONE_INT_ENA Write 1 to enable the KEYMNG_POST_DONE_INT interrupt. (R/W)

Register 36.15. KEYMNG_INT_CLR_REG (0x0014)

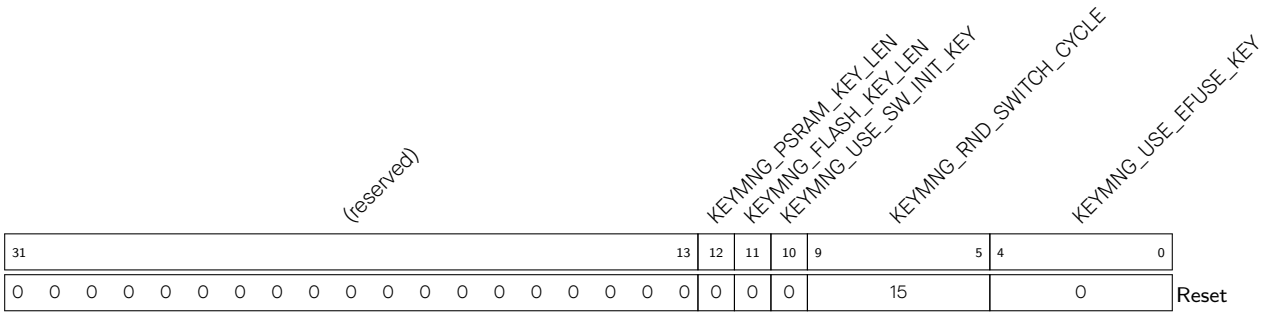


KEYMNG_PREP_DONE_INT_CLR Write 1 to clear the KEYMNG_PREP_DONE_INT interrupt. (WT)

KEYMNG_PROC_DONE_INT_CLR Write 1 to clear the KEYMNG_PROC_DONE_INT interrupt. (WT)

KEYMNG_POST_DONE_INT_CLR Write 1 to clear the KEYMNG_POST_DONE_INT interrupt. (WT)

Register 36.16. KEYMNG_STATIC_REG (0x0018)



KEYMNG_USE_EFUSE_KEY Configures whether or not to use eFuse key instead of Key Manager deployed key.

- Bit[0]: configures whether to use eFuse key for ECDSA_DS key.
  0: No effect
  1: Use eFuse Key, valid only when bit[0] in EFUSE_FORCE_USE_KEY_MANAGER_KEY is 0.
- Bit[1] configures whether to use eFuse key for flash key.
  0: No effect
  1: Use eFuse Key, valid only when bit[1] in EFUSE_FORCE_USE_KEY_MANAGER_KEY is 0.
- Bit[2] configures whether to use eFuse key for HMAC key.
  0: No effect
  1: Use eFuse Key, valid only when bit[2] in EFUSE_FORCE_USE_KEY_MANAGER_KEY is 0.
- Bit[3] configures whether to use eFuse key for RSA_DS_KEY.
  0: No effect
  1: Use eFuse Key, valid only when bit[3] in EFUSE_FORCE_USE_KEY_MANAGER_KEY is 0.
- Bit[4] configures whether to use eFuse key for PSRAM key.
  0: No effect
  1: Use eFuse Key, valid only when bit[4] in EFUSE_FORCE_USE_KEY_MANAGER_KEY is 0.

(R/W)

KEYMNG_RND_SWITCH_CYCLE Configures cycles of the Key Manager to switch random numbers.
Measurement: Key Manager clock cycle.
This field is valid only when EFUSE_KM_RND_SWITCH_CYCLE is set to 0.
It is recommended to set this switch cycle to the number of Key Manager clock cycles corresponding to the TRNG module's clock cycle.
(R/W)

KEYMNG_USE_SW_INIT_KEY Configures whether or not to use sw_init_key, instead of EFUSE_KM_INIT_KEY, valid only when EFUSE_FORCE_DISABLE_SW_INIT_KEY is 0.
0: Use EFUSE_KM_INIT_KEY
1: Use software written sw_init_key
(R/W)

Continued on the next page...

Register 36.16. KEYMNG_STATIC_REG (0x0018)

Continued from the previous page...

KEYMNG_FLASH_KEY_LEN Configures the key length for flash crypt.

0: Use xts-aes-128

1: Use xts-aes-256

(R/W)

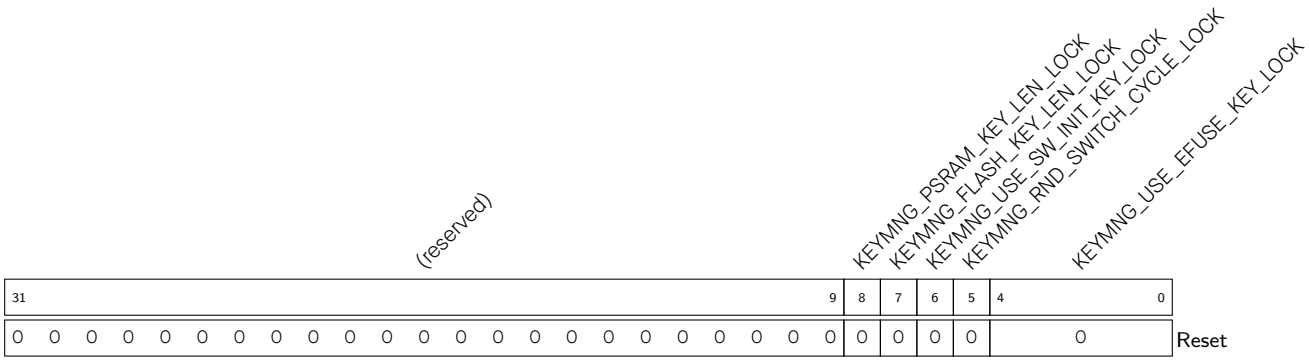
KEYMNG_PSRAM_KEY_LEN Configures the key length for PSRAM crypt.

0: Use xts-aes-128

1: Use xts-aes-256

(R/W)

Register 36.17. KEYMNG_LOCK_REG (0x001C)



KEYMNG_USE_EFUSE_KEY_LOCK Write 1 to lock KEYMNG_USE_EFUSE_KEY.

Each bit locks the corresponding bit of KEYMNG_USE_EFUSE_KEY.

(R/W1)

KEYMNG_RND_SWITCH_CYCLE_LOCK Write 1 to lock KEYMNG_RND_SWITCH_CYCLE.

(R/W1)

KEYMNG_USE_SW_INIT_KEY_LOCK Write 1 to lock KEYMNG_USE_SW_INIT_KEY.

(R/W1)

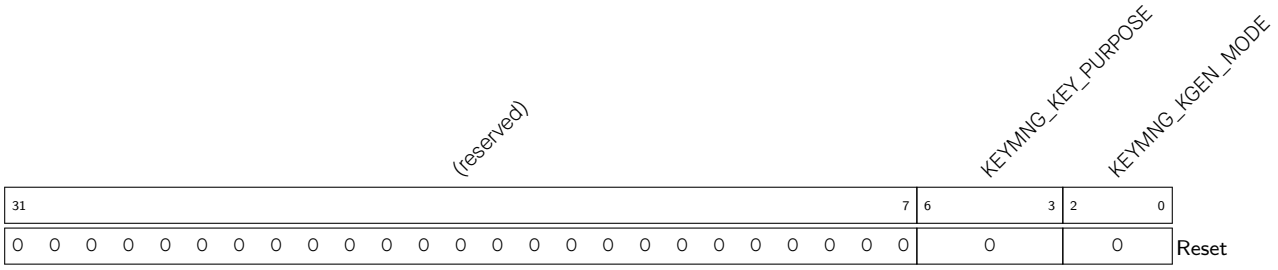
KEYMNG_FLASH_KEY_LEN_LOCK Write 1 to lock KEYMNG_FLASH_KEY_LEN.

(R/W1)

KEYMNG_PSRAM_KEY_LEN_LOCK Write 1 to lock KEYMNG_PSRAM_KEY_LEN.

(R/W1)

Register 36.18. KEYMNG_CONF_REG (0x0020)



KEYMNG_KGEN_MODE Configures deployment mode.

0: Random Deploy Mode

1: AES Deploy Mode

2: ECDH0 Deploy Mode

3: ECDH1 Deploy Mode

4: Private Key Recovery Mode

5: key_info Export Mode

6∼7: Reserved

(R/W)

KEYMNG_KEY_PURPOSE Configures key purpose.

1: ecdsa_key_192

2: ecdsa_key_256

3: flash_256_1_key

4: flash_256_2_key

5: flash_128_key

6: hmac_key

7: ds_key

8: psram_256_1_key

9: psram_256_2_key

10: psram_128_key

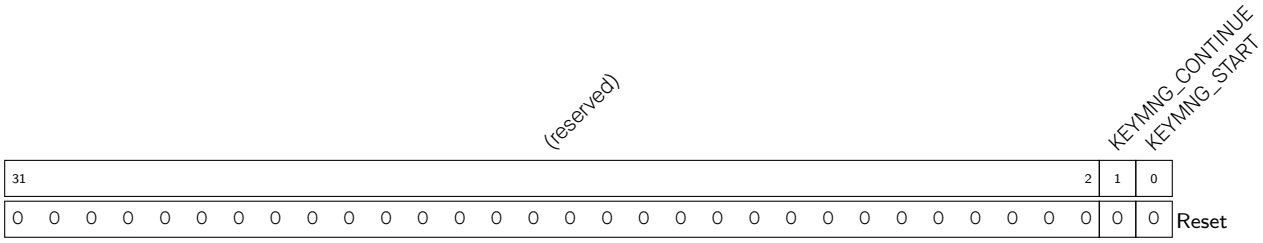
11: ecdsa_key_384_l

12: ecdsa_key_384_h

Others: Reserved

(R/W)

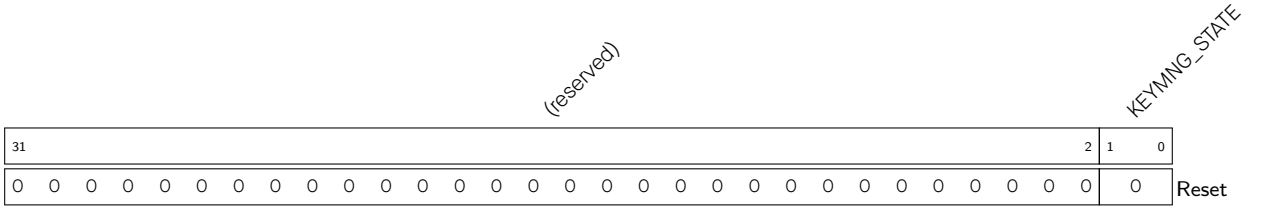
Register 36.19. KEYMNG_START_REG (0x0024)



KEYMNG_START Write 1 to start Key Manager operation at IDLE phase.
(WT)

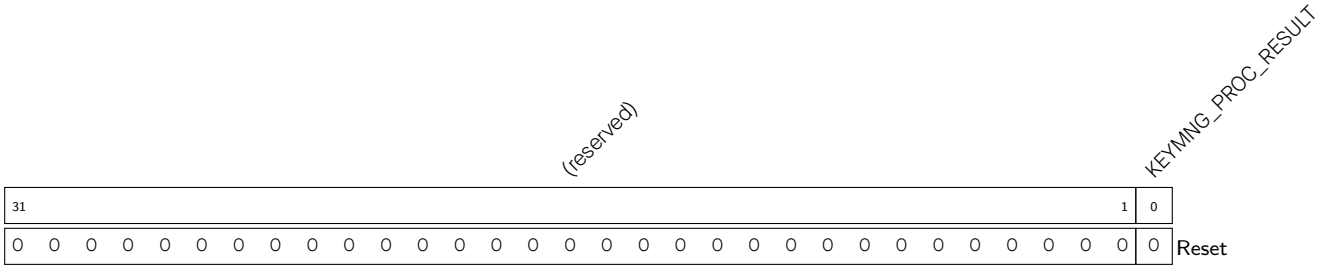
KEYMNG_CONTINUE Write 1 to continue the operation of Key Manager at LOAD/GAIN phase.
(WT)

Register 36.20. KEYMNG_STATE_REG (0x0028)



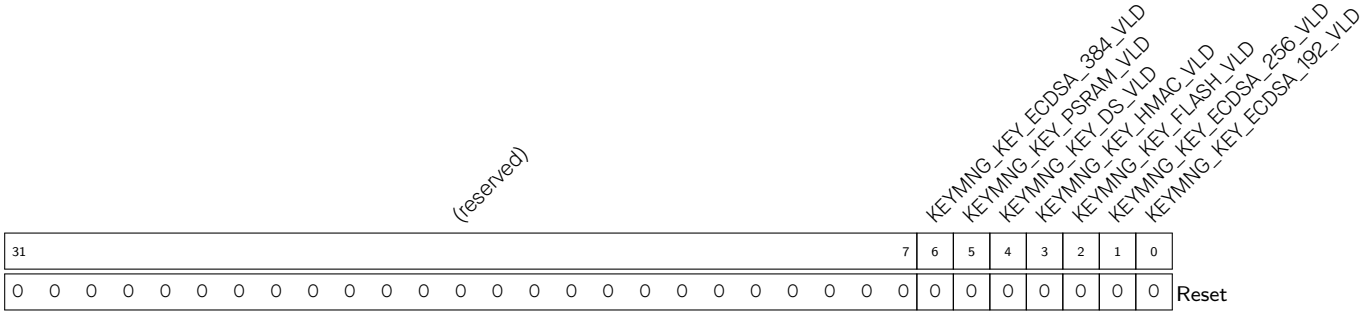
KEYMNG_STATE Represents the state of Key Manager.
0: IDLE
1: LOAD
2: GAIN
3: BUSY
(RO)

Register 36.21. KEYMNG_RESULT_REG (0x002C)



KEYMNG_PROC_RESULT Represents the procedure result of Key Manager, valid only when Key Manager procedure is done.
0: Key Manager procedure failed.
1: Key Manager procedure succeeded.
(RO/SS)

Register 36.22. KEYMNG_KEY_VLD_REG (0x0030)



KEYMNG_KEY_ECDSA_192_VLD Represents the status of key_ecdsa_192.

0: The key has not been deployed yet.

1: The key has been deployed correctly.

(RO)

KEYMNG_KEY_ECDSA_256_VLD Represents the status of key_ecdsa_256.

0: The key has not been deployed yet.

1: The key has been deployed correctly.

(RO)

KEYMNG_KEY_FLASH_VLD Represents the status of key_flash.

0: The key has not been deployed yet.

1: The key has been deployed correctly.

(RO)

KEYMNG_KEY_HMAC_VLD Represents the status of key_hmac.

0: The key has not been deployed yet.

1: The key has been deployed correctly.

(RO)

KEYMNG_KEY_DS_VLD Represents the status of key_ds.

0: The key has not been deployed yet.

1: The key has been deployed correctly.

(RO)

KEYMNG_KEY_PSRAM_VLD Represents the status of key_psram.

0: The key has not been deployed yet.

1: The key has been deployed correctly.

(RO)

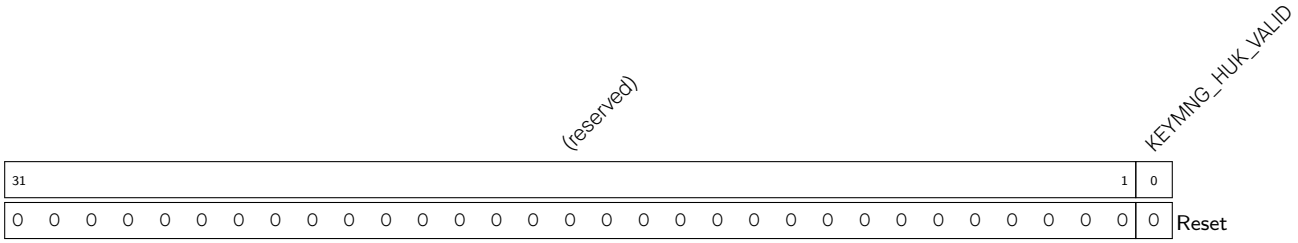
KEYMNG_KEY_ECDSA_384_VLD Represents the status of key_ecdsa_384.

0: The key has not been deployed yet.

1: The key has been deployed correctly.

(RO)

Register 36.23. KEYMNG_HUK_VLD_REG (0x0034)



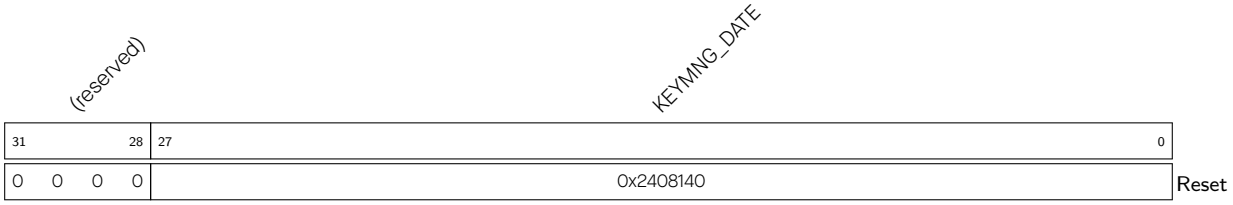
KEYMNG_HUK_VALID Represents the HUK status.

0: HUK is not valid.

1: HUK is valid.

(RO)

Register 36.24. KEYMNG_DATE_REG (0x00FC)



KEYMNG_DATE Version control register.

(R/W)

Part VI

Image and Voice Processing

Focusing on multimedia processing, this part classifies components related to image and voice processing, including camera controllers, video encoders, image signal processors, and voice activity detection.

Chapter 37

JPEG Codec

ESP32-P4's JPEG codec is an image codec, which is based on the JPEG baseline standard, for compressing (encoding) and decompressing (decoding) images to reduce the bandwidth required to transmit images or the space required to store images, making it possible to process large-resolution images. The image encoding process mainly includes discrete cosine transform (DCT), zigzag scanning, quantization, and Huffman coding. The image decoding process, which is the inverse operation of the encoding process, mainly includes Huffman decoding, inverse quantization, inverse zigzag scanning, and inverse discrete cosine transform (IDCT).

37.1 Terminology

MCU	Minimum coded unit. The basic unit of codec processing
baseline	A particular sequential DCT-based encoding and decoding process specified in the standard, which is required for all DCT-based decoding processes
image	A set of two-dimensional arrays of integer data
component	One of the two-dimensional arrays that comprise an image
data unit	An 8 × 8 block of pixels of one component
marker	A two-byte code in which the first byte is hexadecimal FF (0xFF) and the second byte is a value between 1 and hexadecimal FE (0xFE). It is served to identify the various structural parts of the compressed data formats
restart marker	Used to isolate coded data segments and for application-specific uses, such as decoder re-synchronization and error recovery
restart interval	The integer number of MCUs processed between two restart markers
scan	A single pass through the data for one or more of the components in an image

Figure 37.1-1 shows the schematic diagram of JPEG bitstream format. In this diagram, the dashed lines mark the structural composition of the corresponding parts of the bitstream, and the orange blocks indicate the marker codes.

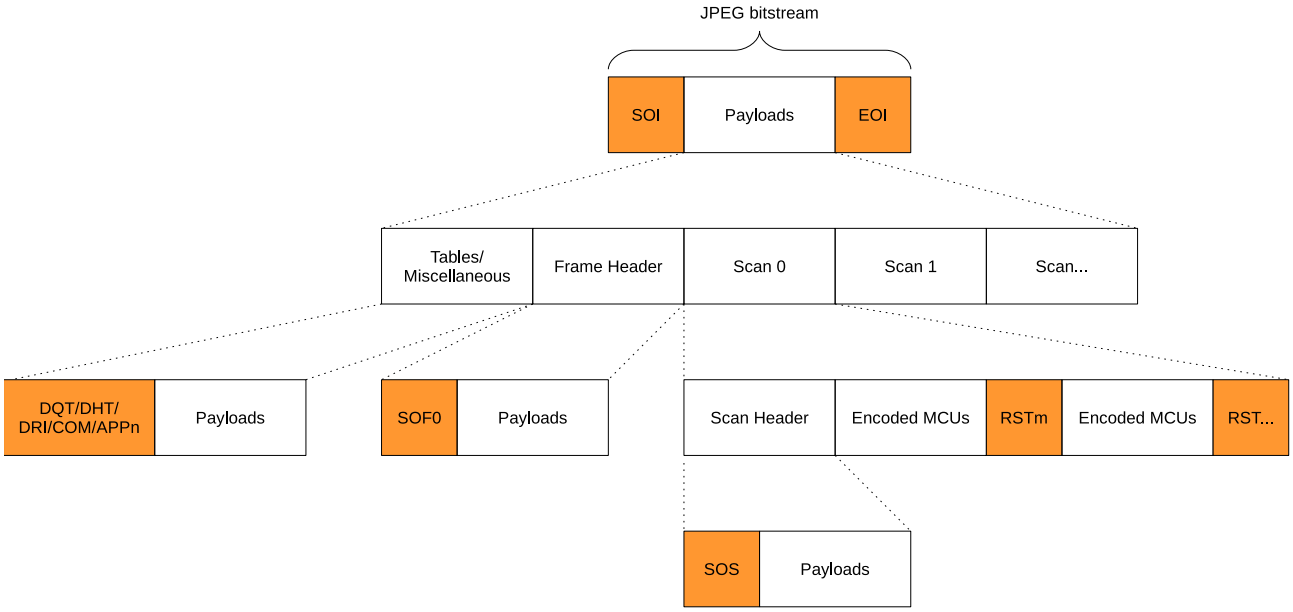


Figure 37.1-1. JPEG Bitstream Format

The marker code assignments are shown in Table 37.1-2.

Table 37.1-2. Marker Code Assignments

Code Assignment	Marker	Description
0xFFC0	SOF0	Start of frame (Baseline DCT)
0xFFC4	DHT	Define Huffman table(s)
0xFFDm	RSTm	Restart with modulo 8 count “m”
0xFFD8	SOI	Start of image
0xFFD9	EOI	End of image
0xFFDA	SOS	Start of scan
0xFFDB	DQT	Define quantization table(s)
0xFFDD	DRI	Define restart interval
0xFFEn	APPn	Reserved for application segments, “n” is 0 to F
0xFFFE	COM	Comment

37.2 Introduction

The JPEG codec is based on the JPEG baseline standard, which specifies that an encoding process is completed through one or more scans. An scan contains one or more MCUs. MCU is the sequence of data units defined by the sampling factors of the image components in the scan, which means MCU of different image formats contains different sequence of data units, as shown in Table 37.2-1.

Table 37.2-1. The Number of Data Units in the MCU of Different Image Format

Image Format	MCU[1]		
	Luminance Component Y	Chrominance Component U	Chrominance Component V
YUV444	1 x 1	1 x 1	1 x 1

YUV422	2 x 1	1 x 1	1 x 1
YUV420	2 x 2	1 x 1	1 x 1
GRAY	1 x 1	none	none

[1] The MCU of each image format is represented by the number of data units in the horizontal direction x the number of data units in the vertical direction of each component.

At a time, the JPEG codec can only be configured as an encoder or a decoder, that is, it cannot work as an encoder and a decoder at the same time.

- When used as an encoder, it obtains the image to be compressed through 2D DMA and sends the encoded bitstream to the 2D DMA.
- When used as a decoder, it obtains the coded bitstream through 2D DMA and sends the decoded image to 2D DMA.

The workflow of JPEG codec is shown in Figure 37.2-1.

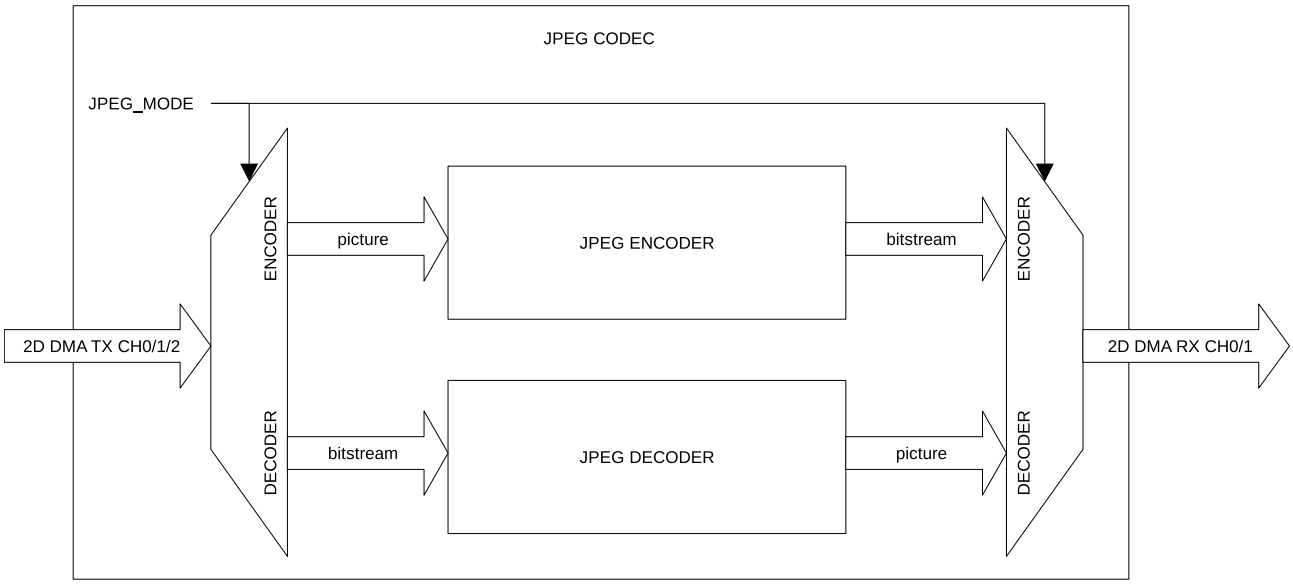


Figure 37.2-1. JPEG codec system connection block diagram

The function clock of the JPEG codec is SYS_CLK.

37.3 Features

When used as an encoder, the JPEG codec has the following features:

- integrated discrete cosine transform algorithm
- integrated canonical Huffman coding
- RGB888, RGB565, YUV422 and GRAY as original input image formats
- conversion of RGB888 and RGB565 into YUV444, YUV422 or YUV420 (the only formats supported by impression) for image compression

- 4 x configurable quantization coefficient tables with 8-bit or 16-bit precision
- performance:
  - still image compression: up to 4K resolution
  - dynamic image compression: up to 1080P@40fps,720P@70fps (excluding header encoding time)
- automatically added stuffed zero byte
- automatically added EOI marker

When used as a decoder, the JPEG codec has the following features:

- integrated inverse discrete cosine transform algorithm
- integrated Huffman decoding
- supported image formats for compressed bitstream decoding: YUV444, YUV422, YUV420, and GRAY.
- 4 x configurable quantization coefficient tables with 8-bit or 16-bit precision
- 2 x DC and 2 x AC Huffman tables
- supports image decoding of any resolution. However, the resolution of output decoded image differs from the format of the input image:
  - YUV444, GRAY: both the horizontal and vertical resolutions of the output decoded image are multiples of 8, i.e., 150 × 150 images with an output resolution of 152 × 152
  - YUV422: the horizontal resolution of the output decoded image is the multiples of 16 and the vertical resolution is multiples of 8, i.e., 150 × 150 images with an output resolution of 160 × 152
  - YUV420: both the horizontal and vertical resolutions of the output decoded image are multiples of 16, i.e., 150 × 150 images with an output resolution of 160 × 160
- performance:
  - still image decoding: up to 4K resolution
  - dynamic image decoding: up to 1080P@40fps,720P@70fps (excluding header parsing time)

37.4 Architectural Overview

When the JPEG codec is configured as an encoder, its architecture is shown in Figure 37.4-1.

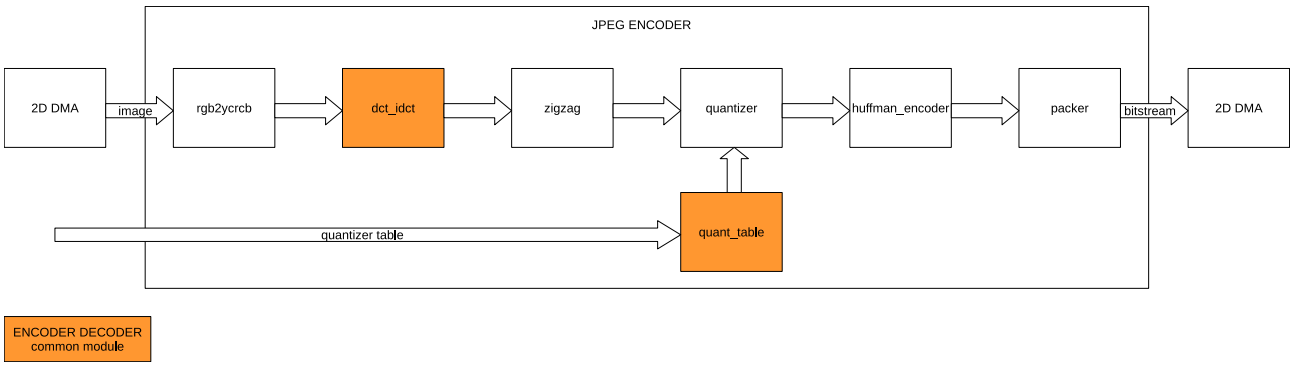


Figure 37.4-1. JPEG encoder architecture

- **rgb2ycrcb** implements the color space conversion, and converts the original image into an image format supported by the codec for compressing.
- **dct_idct** implements two-dimensional DCT transformation of data units.
- **zigzag** performs zigzag scanning and sorting on the DCT transformed data units.
- **quant_table** stores the quantization coefficient table delivered by the software.
- **quantizer** quantifies the zigzag sorted data units based on the quantization coefficient table stored in **quant_table**.
- **huffman_encoder** performs Huffman encoding on the quantized result of the data units.
- **packer** packs the Huffman-encoded data into groups of 32 bits, and outputs the final encoded bitstream.

When the JPEG codec is configured as a decoder, its architecture is shown in figure 37.4-2.

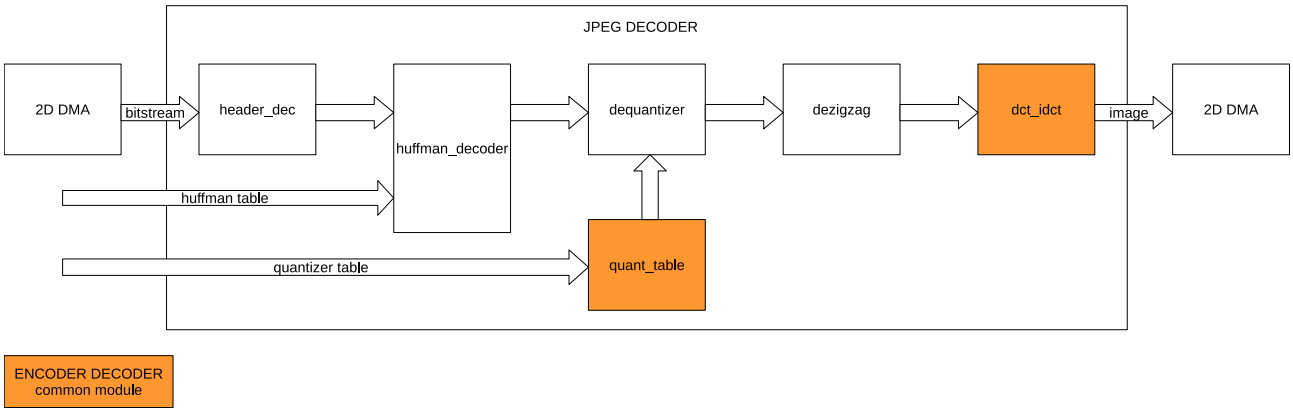


Figure 37.4-2. JPEG decoder architecture

- **header_dec** implements bitstream extraction and marker SOS and EOI parsing, and controls the decoding process.
- **huffman_decoder** stores the Huffman table delivered by the software, and implements Huffman decoding of the bitstream.
- **quant_table** stores the quantization coefficient table delivered by the software.
- **dequantizer** dequantizes the Huffman decoded data based on the quantization coefficient table stored in **quant_table**.
- **dezigzag** implements reverse zigzag scanning and sorting on the dequantized data units, and combines them into dequantized data units
- **dct_idct** implements the two-dimensional inverse DCT transformation of the dequantized data units, and outputs the final decoded data units.

Since the JPEG codec cannot be used as an encoder and decoder at the same time, the encoder and decoder share the **quant_table** and **dct_idct** modules to save hardware resources.

37.5 Functional Description

The JPEG Codec can be configured to work as an encoder or a decoder.

37.5.1 JPEG Encoder

37.5.1.1 Pause

The JPEG encoder can be paused during the operation for other higher priority tasks. This feature is controlled by register field JPEG_PAUSE_EN.

- Set JPEG_PAUSE_EN to 1, the JPEG encoder will no longer output the bitstream to the 2D DMA RX channel.
- Set JPEG_PAUSE_EN back to 0, the JPEG encoder will be resumed, and output the bitstream normally.

37.5.1.2 Color Space Conversion

According to the standard, the image formats supported by the JPEG encoder are YUV444, YUV422, YUV420, and GRAY. However, ESP32-P4's JPEG encoder has integrated a color space conversion function to convert RGB888 and RGB565 formats into the above-mentioned image formats supported by the encoder before encoding. In this way, the range of supported image formats is expanded.

The color space conversion formulas for converting RGB to YUV are:

$$
\begin{aligned}
Y &= (0.299 * R) + (0.587 * G) + (0.114 * B) \\
U &= (-0.1687 * R) + (-0.3313 * G) + (0.5 * B) + 128 \\
V &= (0.5 * R) + (-0.4187 * G) + (-0.0813 * B) + 128
\end{aligned}
$$

YUV formats only support downsampling. For example, YUV444 can remain unchanged or be downsampled to YUV422 or YUV420, and YUV422 can remain unchanged or be downsampled to YUV420.

The image format before conversion is represented by its format as well as pixel order. Table 37.5-1 demonstrates the representations of the supported image formats for color space conversion:

Table 37.5-1. Supported Image Formats for Color Space Conversion

JPEG_EXTD_ COLOR_SPACE_EN	JPEG_ COLOR_SPACE	JPEG_EXTD_ COLOR_SPACE	Format[1]	JPEG_ PIXEL_REV	Pixel Order
0	0	N/A	RGB888	0	RGB
				1	BGR
	1		YUV422[2]	0	YUYV
				1	UYVY
	2		RGB565	0	RGB
				1	BGR
	3		GRAY[2]	N/A	YYY
1	N/A	0	YUV444	0	YUV
				1	VUY
		1	YUV420[2]	N/A	YYU/YYV[3]

[1] The image formats can be converted into other formats supported by the encoder via configuring register field JPEG_SAMPLE_SEL:

- 0: YUV444
- 1: YUV422
- 2: YUV420

[2] Image format conversion is limited:

- YUV422: can only remain unchanged or be downsampled to YUV420. Setting JPEG_SAMPLE_SEL to 0 has no effect.
- YUV420 or GRAY: will not be converted, regardless of the value of JPEG_SAMPLE_SEL.

[3] The pixel arrangement alternates between rows: odd-numbered rows use the YYU pattern, and even-numbered rows use the YYV pattern.

37.5.1.3 Configurable Quantization Coefficient Table

ESP32-P4 JEPG encoder has implemented four 8 x 8 quantization coefficient tables in its hardware, which allows 64 software-configurable coefficient values per table.

The precision of each quantization table is configurable via JPEG_QNR_PRECISION:

- 0: low 8-bit effective
- 1: low 16-bit effective

The coefficient values for each table can be configured in FIFO mode or non-FIFO mode depending on the value of JPEG_QNR_FIFO_EN:

- 0: non-FIFO mode, in which each coefficient is written to a specific address
- 1: FIFO mode, in which all coefficients are written to the same address

Table 37.5-2. Position Number of Coefficients in a Quantization Coefficient Table (Example)

0	1	2	3	4	5	6	7
8	9	10	11	12	13	14	15
...							

56	57	58	59	60	61	62	63

In different modes, the configuration of the quantization coefficient table is different. Note, *n* (*n* = 0, 1, 2, 3) indicates the number of each quantization coefficient table:

- non-FIFO mode: each quantization coefficient in a specific position of a quantization coefficient table *n* must be written to a specific address, which is (JPEG codec base address + *n* * 0x100 + *quantization coefficient position number* * 0x4), in which the *quantization coefficient position number* can be 0, 1, 2, ..., 63 (see Table 37.5-2). In this mode, since each coefficient is written to a specific address, configuring the 64 quantization coefficients in sequence is not required.
- FIFO mode: all 64 quantization coefficients of the quantization table *n* are written to the same address, which is JPEG_T*n*QNR_REG. In this mode, since all coefficients are written to the same address, configuring the 64 quantization coefficients one by one in sequence is required. Afterwards, the hardware will automatically store these 64 quantization coefficients in sequence as shown in Table 37.5-2.

If there are less than 4 quantization tables, the corresponding quantization table will not be written.

To use a specific quantization coefficient table *n* for the luminance component or the chrominance component, just set the corresponding register fields to *n*:

- luminance component: JPEG_LQNR_TBL_SEL
- chrominance component: JPEG_CQNR_TBL_SEL

The JPEG baseline standard recommended 8-bit precision quantization coefficient tables for the luminance component and the chrominance component are shown in Table 37.5-3 and Table 37.5-4.

Table 37.5-3. Recommended 8-bit Precision Quantization Coefficient Table for the Luminance Component

16	11	10	16	24	40	51	61
12	12	14	19	26	58	60	55
14	13	16	24	40	57	69	56
14	17	22	29	51	87	80	62
18	22	37	56	68	109	103	77
24	35	55	64	81	104	113	92
49	64	78	87	103	121	120	101
72	92	95	98	112	100	103	99

Table 37.5-4. Recommended 8-bit Precision Quantization Coefficient Table for the Chrominance Component

17	18	24	47	99	99	99	99
18	21	26	66	99	99	99	99
24	26	56	99	99	99	99	99
47	66	99	99	99	99	99	99
99	99	99	99	99	99	99	99
99	99	99	99	99	99	99	99
99	99	99	99	99	99	99	99
99	99	99	99	99	99	99	99

More information can be found in ITU T.81 Recommendation. Adjusting the quantization table can change the compression ratio (the ratio between the uncompressed image size and compressed image size) and the quality of encoded images (the larger the quantization coefficient, the greater the compression ratio and the worse the image quality).

37.5.1.4 Stuffed Zero Byte

As required in the JPEG baseline standard to avoid any marker within a Huffman-coded segment, any 0xFF byte generated by the Huffman coding should be followed by a "stuffed" zero byte.

JPEG encoder has implemented a mechanism to automatically add this "stuffed" 0x00 byte after each 0xFF byte in the Huffman-coded segment of bitstream.

Users can configure whether to add 0x00 byte after each 0xFF byte via JPEG_FF_CHECK_EN:

- 0: not add
- 1: add (by default)

37.5.1.5 EOI Marker

The end of Image (EOI) marker is used to mark the end of the bitstream of a compressed image. The EOI marker code is 0xFFD9.

JPEG encoder has implemented a mechanism to automatically add the EOI marker at the end of the bitstream.

Users can configure whether to add EOF marker via JPEG_TAILER_EN:

- 0: not add
- 1: add

37.5.2 JPEG Decoder

The information in the bitstream must be parsed by the software, then sent to the hardware through the APB configuration bus. Note that the hardware does not support parsing any marker segments, except the SOS, RST and EOI marker segments.

37.5.2.1 Multiple Chrominance Components

The frame header contains information about the number of chrominance components of the image, the ID of each component, and the sampling factors in the horizontal and vertical directions of each component.

JPEG decoder's software must parse such information and pass it to respective registers JPEG_COMPONENT_NUM, JPEG_C*n*_ID, JPEG_C*n*_X_FACTOR, and JPEG_C*n*_Y_FACTOR. Here, *n* can be 0, 1, 2, indicating that up to 3 chrominance components are supported by hardware. Note that if less than 3 components are involved, just do not configure the rest of the registers.

According to the value of register JPEG_C*n*_ID of each chrominance component, the hardware will check whether it is consistent with the ID in the scan header information. The sampling factor indicates the number of data units of this component that an MCU contains in both the horizontal and vertical directions. So

according to the value of register JPEG_COMPONENT_NUM, JPEG_C*n*_X_FACTOR and JPEG_C*n*_Y_FACTOR, the hardware can obtain the proportional relationship of chrominance components, that is, the format of the image to be decoded. See Table 37.5-5.

Table 37.5-5. Sampling Factor and Image Format

JPEG_COMPONENT_NUM	JPEG_C*n*_X/Y_FACTOR	Format of the Image to be Decoded
0	-	invalid
1	C0_X_FACTOR is 1 C0_Y_FACTOR is 1	GRAY
2	-	invalid
3	C0_X_FACTOR is 1 C0_Y_FACTOR is 1 C1_X_FACTOR is 1 C1_Y_FACTOR is 1 C2_X_FACTOR is 1 C2_Y_FACTOR is 1	YUV444
	C0_X_FACTOR is 2 C0_Y_FACTOR is 1 C1_X_FACTOR is 1 C1_Y_FACTOR is 1 C2_X_FACTOR is 1 C2_Y_FACTOR is 1	YUV422
	C0_X_FACTOR is 2 C0_Y_FACTOR is 2 C1_X_FACTOR is 1 C1_Y_FACTOR is 1 C2_X_FACTOR is 1 C2_Y_FACTOR is 1	YUV420

Regardless of the format of the image to be decoded, the decoder only supports one scan, which contains all components. After decoding, the image will be sent to 2D DMA, so please refer to Chapter 8 *2D-DMA Controller (2D-DMA)* for the layout of the decoded image in each format.

37.5.2.2 Parsing RST Marker

The bitstream may have a restart (RST) marker in the scan segment. The RST marker code is 0xFFD*x* (*x* = 0, 1, ..., 7).

JPEG decoder's hardware can parse RST markers and check whether the actual number of MCUs between two RST markers is the same as the restart interval defined in the DRI segment of the Tables/Miscellaneous segment.

Firstly, the software parses the DRI segment and pass the defined restart interval to register JPEG_RESTART_INTERVAL. Then, the hardware will check whether the subsequent bitstream has an RST marker every time JPEG_RESTART_INTERVAL MCUs are decoded.

Configuring JPEG_RESTART_INTERVAL to 0 indicates the bitstream does not contain any RST marker. In this case, the hardware will not check RST markers.

37.5.2.3 Configurable Quantization Coefficient Table

ESP32-P4 JPEG decoder also supports 4 configurable quantization coefficient tables. Since the JPEG codec cannot be used as an encoder and decoder at the same time, decoder can share the same four configurable quantization coefficient tables with the encoder to save hardware resources.

Therefore, the configuration of decoder's quantization coefficient tables is similar to that of the encoder (see Section 37.5.1.3) with some exceptions described below:

- The software needs to parse the DQT segment in the Tables/Miscellaneous segment to obtain the quantization coefficient tables of the decoder. The recommended quantization coefficient tables are the same as the encoder shown in Table 37.5-3 and Table 37.5-4.
- The precision of the quantization table*n* is configured via higher 4 bits of register JPEG_T*n*_DQT_INFO:
  - 0: low 8-bit effective
  - 1: low 16-bit effective

  The precision is also obtained from the DQT segment in the Tables/Miscellaneous segment.
- For each chrominance component, a different quantization coefficient table can be selected. The lower 4 bits of register JPEG_T*n*_DQT_INFO specify the ID of each quantization coefficient table, and each chrominance component selects the quantization coefficient table whose ID is the value of register JPEG_C*n*_DQT_TBL_SEL. The ID of each quantization coefficient table is also obtained from the DQT segment in the Tables/Miscellaneous segment.

37.5.2.4 Configurable Huffman Table

ESP32-P4 supports up to 2 DC Huffman tables and 2 AC Huffman tables. Users can configure as many Huffman tables as needed within the limits.

Each Huffman table can be established with three types of information, which can be obtained from the DHT segment in the Tables/Miscellaneous segment:

- the number of codeword with a codeword length of 1 to 16 bits,
- the symbols corresponding to the decoded codeword
  - DC table: up to 16 symbols
  - AC table: up to 256 symbols
- the minimum codeword with a codeword length of 1 to 16 bits.

Both of the DC and AC Huffman tables can be configured in a FIFO fashion or a non-FIFO fashion depending on the value of JPEG_DHT_FIFO_EN:

- 0: non-FIFO mode
- 1: FIFO mode

In different modes, the configuration of the Huffman table is different. Here is the configuration of DC table*n* (*n* = 0, 1):

- non-FIFO mode: each piece in different type of information used to establish the DC*n* Huffman table must be written to a specific address.

  - the number of codeword with a codeword length of 1 to 16 bits: JPEG codec base address + 0x500 + *n* * 0x80 + (*codeword length* - 1) * 0x4, in which *codeword length* = 1, 2, ..., 16.
  - the symbols corresponding to the decoded codeword: JPEG codec base address + 0x600 + *n* * 0x40 + *symbol number* * 0x4, in which *symbol number* = 0, 1, ..., 15.
  - the minimum codeword with a codeword length of 1 to 16 bits: JPEG codec base address + 0xF00 + *n* * 0x80 + (*codeword length* - 1) * 0x4, in which *codeword length* = 1, 2, ..., 16.

  In this mode, since each piece of different type of information is written to a specific address, configuring each piece of information in sequence is not required.

- FIFO mode: all pieces of the same type of information used to establish the DC*n* Huffman table must be written to the same address, respectively.
  - the number of codeword with a codeword length of 1 to 16 bits: JPEG_DHT_TOTLEN_DC*n*_REG.
  - all symbols corresponding to the decoded codeword (DC table has up to 16 symbols): JPEG_DHT_VAL_DC*n*_REG.
  - the minimum codeword with a codeword length of 1 to 16 bits: JPEG_DHT_CODEMIN_DC*n*_REG.

  In this mode, since all pieces of the same type of information are written to the same address, configuring each piece of same type of information one by one in sequence is required. Afterwards, the hardware will automatically store such information in sequence.

The configuration of AC*n* (*n* = 0, 1) tables is very similar to that of the DC*n* (*n* = 0, 1) tables. Here below is the configuration of AC*n* (*n* = 0, 1) tables:

- non-FIFO mode: each piece in different type of information used to establish the AC*n* Huffman table must be written to a specific address.
  - the number of codeword with a codeword length of 1 to 16 bits: JPEG codec base address + 0x540 + *n* * 0x80 + (*codeword length* - 1) * 0x4, in which *codeword length* = 1, 2, ..., 16.
  - the symbols corresponding to the decoded codeword: JPEG codec base address + 0x680 + *n* * 0x400 + *symbol number* * 0x4, in which *symbol number* = 0, 1, ..., 256.
  - the minimum codeword with a codeword length of 1 to 16 bits: JPEG codec base address + 0xF40 + *n* * 0x80 + (*codeword length* - 1) * 0x4, in which *codeword length* = 1, 2, ..., 16.

  In this mode, since each piece of different type of information is written to a specific address, configuring each piece of information in sequence is not required.

- FIFO mode: all pieces of the same type of information used to establish the AC*n* Huffman table must be written to the same address, respectively.
  - the number of codeword with a codeword length of 1 to 16 bits: JPEG_DHT_TOTLEN_AC*n*_REG.
  - all symbols corresponding to the decoded codeword (AC table has up to 256 symbols): JPEG_DHT_VAL_AC*n*_REG.
  - the minimum codeword with a codeword length of 1 to 16 bits: JPEG_DHT_CODEMIN_AC*n*_REG.

  In this mode, since all pieces of the same type of information are written to the same address, configuring each piece of same type of information one by one in sequence is required. Afterwards, the hardware will automatically store such information in sequence.

For the minimum codeword with a codeword length of 1 to 16 bits, it should be moved left to the high bit of the 16-bit configuration data. If a codeword length does not exist, the configuration value is 0xFFFF. For example, if a minimum codeword is 0x7C, the 16-bit configuration data should be 0x7C00.

The JPEG baseline standard recommended DC and AC Huffman tables can be found in ITU T.81 Recommendation.

The software can obtain the IDs of DC*n* and AC*n* Huffman tables by parsing the DHT segment in the Tables/Miscellaneous segment, then pass these ID to the corresponding register JPEG_DC*n*_DHT_ID and JPEG_AC*n*_DHT_ID.

The hardware will automatically use the DC*n* and AC*n* Huffman tables with the same ID from JPEG_DC*n*_DHT_ID and JPEG_AC*n*_DHT_ID.

37.5.2.5 Timeout Detection

Additionally, timeout detection is supported when the JPEG codec is working as a decoder. The hardware triggers the timeout interrupt JPEG_DECODE_TIMEOUT_INT:

- when the decoder stops reading the bitstream for more than $(2^{JPEG_DECODE_TIMEOUT_THRES} - 1)$ clock cycles,
- or when the number of clock cycles consumed for decoding an image exceeds the value of $(2^{JPEG_DECODE_TIMEOUT_THRES} - 1)$

In this case, the decoder will be reset depending on the configuration of JPEG_DECODE_TIMEOUT_TASK_SEL:

- 0: the hardware will not perform any operation, and the software is required to configure the JPEG_SOFT_RST or JPEG_FSM_RST to perform a reset operation.
- 1: the hardware will automatically reset the decoder to exit the timeout state and enter the initial state.

37.6 Interrupts

ESP32-P4's JPEG Codec can generate the JPEG_INTR interrupt signal that will be sent to the Interrupt Matrix.

Interrupt signal JPEG_INTR can be generated by the following internal interrupt sources:

- JPEG_DONE_INT: Triggered when a frame of image encoding or decoding ends.
- JPEG_RLE_PARALLEL_ERR_INT: Triggered when a run-length encoding error occurs.
- JPEG_CID_ERR_INT: Triggered when the decoded component ID is different from the component ID configured by the software.
- JPEG_C_DHT_DC_ID_ERR_INT: Triggered when the decoded DC Huffman table ID of each component is not the software configured DC0 Huffman table ID or DC1 Huffman table ID.
- JPEG_C_DHT_AC_ID_ERR_INT: Triggered when the decoded AC Huffman table ID of each component is not the software configured AC0 Huffman table ID or AC1 Huffman table ID.
- JPEG_C_DQT_ID_ERR_INT: Triggered when the decoded quantization table ID of each component is different from the software configured quantization table ID.

- JPEG_RST_UXP_ERR_INT: Triggered when the JPEG_RESTART_INTERVAL configured by the software is 0 but the RST marker is parsed by the decoder.
- JPEG_RST_CHECK_NONE_ERR_INT: Triggered when the JPEG_RESTART_INTERVAL configured by the software is non-0 but the RST marker cannot be parsed by the decoder.
- JPEG_RST_CHECK_POS_ERR_INT: Triggered when the MCU number between two parsed RST markers is not equal to the JPEG_RESTART_INTERVAL configured by the software.
- JPEG_OUT_EOF_INT: Triggered when the EOF marker is read from the TX channel of 2D DMA.
- JPEG_SR_COLOR_MODE_ERR_INT: This interrupt is invalid.
- JPEG_DCT_DONE_INT: Triggered when the DCT or IDCT calculation of one data unit is completed.
- JPEG_BS_LAST_BLOCK_EOF_INT: Triggered when the encoding of the last data unit is completed.
- JPEG_SCAN_CHECK_NONE_ERR_INT: Triggered when an image frame has multiple scans to be decoded and the SOS marker is not parsed within (JPEG_SOS_CHECK_BYTE_NUM + 1) bytes in any scan header information.
- JPEG_SCAN_CHECK_POS_ERR_INT: Triggered when the position of the scan header information parsed by the decoder is wrong.
- JPEG_UXP_DET_INT: Triggered when the marker parsed by the decoder is not supported by the hardware.
- JPEG_EN_FRAME_EOF_ERR_INT: Triggered when the number of data units to be encoded read from 2D DMA is less than the number of data units calculated based on the image resolution configured by the software.
- JPEG_EN_FRAME_EOF_LACK_INT: Triggered when a frame of image to be encoded is completely read from 2D DMA but the EOF marker is not read.
- JPEG_DE_FRAME_EOF_ERR_INT: Triggered when the number of data units obtained after decoding a frame of image is different from the number of data units calculated based on the image resolution configured by the software.
- JPEG_DE_FRAME_EOF_LACK_INT: Triggered when the bitstream of a image is completely read from 2D DMA but the EOF marker or EOI marker is not read.
- JPEG_SOS_UNMATCH_ERR_INT: Triggered when the number of components in the scan header information parsed by the decoder is 0 or the header length in the scan header information parsed by the decoder does not match the actual header length.
- JPEG_MARKER_ERR_FST_SCAN_INT: Triggered when there is an error in the first scan header information parsed by the decoder. The errors include errors triggering JPEG_CID_ERR_INT, errors triggering JPEG_C_DHT_DC_ID_ERR_INT, errors triggering JPEG_C_DHT_AC_ID_ERR_INT, errors triggering JPEG_C_DQT_ID_ERR_INT and errors triggering JPEG_SOS_UNMATCH_ERR_INT.
- JPEG_MARKER_ERR_OTHER_SCAN_INT: Triggered when there is an error in the non-first scan header information parsed by the decoder. The error type is the same as the errors that trigger JPEG_MARKER_ERR_FST_SCAN_INT.
- JPEG_UNDET_INT: Triggered when the bitstream of an image is completely read from 2D DMA but the SOS marker is not read.

- JPEG_DECODE_TIMEOUT_INT: Triggered when the decoder is timeout. Section 37.7.2 describes how this interrupt will occur.

The above interrupt sources can only be triggered when the interrupt enable register *interrupt_source_name*_ENA is set to 1. Write 1 to *interrupt_source_name*_CLR will clear the interrupt.

Note:
For definitions of *interrupt*, *interrupt signal*, *interrupt source*, and their correlations, please refer to Chapter 13 *Interrupt Matrix* > Section 13.2 *Interrupt Terminology in ESP32-P4*.

Each interrupt source can be configured by a common set of registers that are described in Section *Interrupt Configuration Registers*. The specific registers can be found in Section 37.8 *Register Summary*.

37.7 Programming Procedures

37.7.1 JPEG Encoder

When the JPEG codec is working as an encoder, the configuration process is:

1. Activate the JPEG codec:
   - Set HP_SYS_CLKRST_JPEG_SYS_CLK_EN to 1 to enable the JPEG codec clock;
   - Set HP_SYS_CLKRST_RST_EN_JPEG to 0 to release the JPEG codec system level reset.
2. Configure the JPEG codec:
   - Set JPEG_SOFT_RST to 1 to reset the JPEG codec, and then set JPEG_SOFT_RST to 0 to release the JPEG codec reset.
   - Set JPEG_MODE to 0 to configure the JPEG codec to be used as an encoder.
   - Select the original image format by configuring the following registers:
     - Format:
       * JPEG_COLOR_SPACE
       * JPEG_EXTD_COLOR_SPACE_EN
       * JPEG_EXTD_COLOR_SPACE
     - Pixel order: JPEG_PIXEL_REV

     See details in Table 37.5-1.
   - Select whether to add the EOI marker (0xFFD9) at the end of the bitstream by configuring JPEG_TAILER_EN:
     - 0: disable
     - 1: enable
   - Select the image format to be compressed by configuring JPEG_SAMPLE_SEL:
     - 0: YUV444

  - 1: YUV422
  - 2: YUV420

  Note that this register is valid only when the original image format is RGB, YUV444, or YUV422. See details in Table 37.5-1.

- Configure JPEG_HA as the original image width and JPEG_VA as the original image height.
- Configure the quantization coefficient table by following the instructions in Section 37.5.1.3.

3. Configure 2D DMA (For detailed instructions or introduction to related concepts, please refer to Chapter *8 2D-DMA Controller (2D-DMA)*.)

   (a) Configure the linked list of 2D DMA TX channel.

   - Set 2DEN to 1 to enable 2D function.
   - Configure the Buffer address pointer as the starting address where the original image is stored.
   - Configure pbyte to select the number of bytes of one pixel of the original image.
   - Configure HA as the width of the original image and VA as the height of the original image.
   - Configure hb as the number of pixels in the horizontal direction of an image block and vb as the number of pixels in the vertical direction of an image block.
   - Configure mod as 1, which means continuous reading of image blocks of hb × vb size.
   - Configure (X, Y) as the coordinate of the first pixel in the starting image block. Among them, according to the original image format and the format of the image to be compressed, there are restrictions on hb and vb, as shown in the table 37.7-1:

Table 37.7-1. JPEG Encoder hb and vb Configuration

Format of Original Image	Format of the Image to be Compressed	hb	vb
RGB/YUV444	YUV444	Multiples of 8	Multiples of 8
	YUV422	Multiples of 16	Multiples of 8
	YUV420	Multiples of 16	Multiples of 16
YUV422	YUV422	Multiples of 16	Multiples of 8
	YUV420	Multiples of 16	16
YUV420	YUV420	Multiples of 16	16
GRAY	GRAY	Multiples of 8	Multiples of 8

   (b) Configure the linked list of the 2D DMA RX channel:

   - Set 2DEN to 0 to enable 1D function.
   - Configure the Buffer address pointer as the starting address where the encoded bitstream will be stored.
   - Configure size as the number of bytes in the space to store the encoded bitstream.
   - configure mod as 0, which means one-time reading of image blocks of hb × vb size.

   (c) Connect 2D DMA channels to JPEG codec:

- Connect the TX channel*n* to the JPEG codec by configuring DMA2D_OUT_PERI_SEL_CH*n* to 0;
- Connect the RX channel*n* to the JPEG codec by configuring DMA2D_IN_PERI_SEL_CH*n* to 0.

(d) Reset 2D DMA channels:

- 2D DMA TX channel*n*: Set DMA2D_OUT_RST_CH*n* to 1 to reset 2D DMA TX channel*n*, then set DMA2D_OUT_RST_CH*n* to 0 to release the reset of 2D DMA TX channel*n*.
- 2D DMA RX channel*n*: Set DMA2D_IN_RST_CH*n* to 1 to reset 2D DMA RX channel*n*, then set DMA2D_IN_RST_CH*n* to 0 to release the reset of 2D DMA RX channel*n*.

(e) Select the burst length of 2D DMA channels:

- Configure DMA2D_OUT_MEM_BURST_LENGTH_CH*n* to select the burst length of 2D DMA TX channel*n*.
- Configure DMA2D_IN_MEM_BURST_LENGTH_CH*n* to select the burst length of 2D DMA RX channel*n*.

(f) Select the 2D DMA output image block size by configuring DMA2D_OUT_MACRO_BLOCK_SIZE_CH*n* according to the format of original image and the format of the image to be compressed. See details in Table 37.7-2:

- 0: 8x8
- 1: 16x8
- 2: 16x16

Table 37.7-2. DMA2D_OUT_MACRO_BLOCK_SIZE_CH*n* Configuration

Format of Original Image	Format of the Image to be Compressed	DMA2D_OUT_MACRO_BLOCK_SIZE_CH*n*
RGB/YUV444	YUV444	0
	YUV422	1
	YUV420	2
YUV422	YUV422	1
	YUV420	2
YUV420	YUV420	2
GRAY	GRAY	0

(g) Enable the reorder function to improve bandwidth utilization by setting DMA2D_OUT_REORDER_EN_CH0 to 1. Note that, this function is only available for 2D DMA TX channel0, not for any other TX channels.

(h) Configure 2D DMA channel linked list address:

- 2D DMA TX channel: DMA2D_OUTLINK_ADDR_CH*n*
- 2D DMA RX channel: DMA2D_INLINK_ADDR_CH*n*

4. Enable interrupt sources:

- DMA2D_OUT_INT_ENA_CH*n*_REG
- DMA2D_IN_INT_ENA_CH*n*_REG

- JPEG_INT_ENA_REG

5. Start the transmission of 2D DMA channels:
   - 2D DMA TX channel: set DMA2D_OUTLINK_START_CH*n* to 1 and then to 0
   - 2D DMA RX channel: set DMA2D_INLINK_START_CH*n* to 1 and then to 0
6. Start the JPEG codec encoding by setting the JPEG_JPEG_START field.
7. Wait till DMA2D_IN_SUC_EOF_CH*n*_INT becomes 1, indicating that the encoded bitstream has been completely written to the memory.

37.7.2 JPEG Decoder

When the JPEG codec is working as a decoder, the configuration process is:

1. Activate the JPEG codec:
   - Set HP_SYS_CLKRST_JPEG_SYS_CLK_EN to 1 to enable the JPEG codec clock
   - Set HP_SYS_CLKRST_RST_EN_JPEG to 0 to release the JPEG codec system level reset.
2. Configure the JPEG codec:
   - Set JPEG_SOFT_RST to 1 to reset the JPEG codec, and then set JPEG_SOFT_RST to 0 to release the JPEG codec reset.
   - Set JPEG_MODE to 1 to configure the JPEG codec to be used as a decoder.
   - Enable the RST marker check following the instructions in Section 37.5.2.2, if required.
   - Configure the quantization coefficient tables by following the instructions in Section 37.5.2.3.
   - Configure the Huffman tables by following the instructions in Section 37.5.2.4.
   - Configure the width and height of the decoded image:
     - Width: write width of the decoded image to JPEG_HA.
     - Height: write the height of the decoded image to JPEG_VA.
     - According to different decoded image formats, JPEG_HA and JPEG_VA must meet the requirements of Table 37.7-3.

Table 37.7-3. JPEG Decoder JPEG_HA and JPEG_VA Configuration

Format of Decoded Image	JPEG_HA	JPEG_VA
YUV444	Multiples of 8	8
YUV422	Multiples of 16	8
YUV420	Multiples of 16	16
GRAY	Multiples of 8	8

3. Configure 2D DMA (For detailed instructions or introduction to related concepts, please refer to Chapter 8 *2D-DMA Controller (2D-DMA)*.)
   (a) Configure the linked list of the 2D DMA TX channel.

- Set 2DEN to 0 to enable the 1D function.
- Configure the Buffer address pointer as the starting address where the bitstream to be decoded is stored.
- Configure the length and size as the number of bytes in the space to store the bitstream to be decoded.
- configure mod as 0, which means one-time reading of image blocks of hb x vb size.

(b) Configure the linked list of 2D DMA RX channel.

- Set 2DEN to 1 to enable 2D function.
- Configure the Buffer address pointer as the starting address where the decoded image will be stored.
- Configure pbyte to select the number of bytes of one pixel of the image which is decoded and has undergone 2D DMA color space conversion.
- Configure HA as the width of the output decoded image, configure VA as the height of the output decoded image, configure hb as the number of pixels in the horizontal direction of an image block, configure vb as the number of pixels in the vertical direction of an image block.
- Configure mod as 1, which means continuous reading of image blocks of hb × vb size.
- Configure (X, Y) as the coordinates of the first pixel in the starting image block. Among them, according to the decoded image format, there are restrictions on hb and vb, as shown in the table 37.7-4:

Table 37.7-4. JPEG Decoder hb and vb Configuration

Decoded image format	hb	vb
YUV444	Multiples of 8	Multiples of 8
YUV422	Multiples of 16	Multiples of 8
YUV420	Multiples of 16	Multiples of 16
GRAY	Multiples of 8	Multiples of 8

(c) Connect 2D DMA channels to JPEG codec:

- Connect the TX channel*n* to the JPEG codec by configuring DMA2D_OUT_PERI_SEL_CH*n* to 0;
- Connect the RX channel*n* to the JPEG codec by configuring DMA2D_IN_PERI_SEL_CH*n* to 0.

(d) Reset 2D DMA channels:

- 2D DMA TX channel*n*: set DMA2D_OUT_RST_CH*n* to 1 to reset 2D DMA TX channel*n*, then set DMA2D_OUT_RST_CH*n* to 0 to release the reset of 2D DMA TX channel*n*.
- 2D DMA RX channel*n*: set DMA2D_IN_RST_CH*n* to 1 to reset 2D DMA RX channel*n*, then set DMA2D_IN_RST_CH*n* to 0 to release the reset of 2D DMA RX channel*n*.

(e) Select the burst length of 2D DMA channels:

- Configure DMA2D_OUT_MEM_BURST_LENGTH_CH*n* to select the burst length of 2D DMA TX channel*n*

- Configure DMA2D_IN_MEM_BURST_LENGTH_CH*n* to select the burst length of 2D DMA RX channel*n*

(f) Select 2D DMA receive image block size by configuring DMA2D_IN_MACRO_BLOCK_SIZE_CH*n* according to the format of the decoded image. See details in Table 37.7-5:

- 0: 8x8
- 1: 16x8
- 2: 16x16

Table 37.7-5. DMA2D_IN_MACRO_BLOCK_SIZE_CH*n* Configuration

Format of Decoded Image	DMA2D_IN_MACRO_BLOCK_SIZE_CH*n*
YUV444	0
YUV422	1
YUV420	2
GRAY	0

(g) Enable the reorder function to improve bandwidth utilization by setting set DMA2D_IN_REORDER_EN_CH0 to 1. Note that, this function is only available for 2D DMA RX channel0, not for any other RX channels.

(h) Select whether to scramble the pixel order of the decoded image before 2D DMA color space conversion by configuring DMA2D_IN_SCRAMBLE_SEL_PRE_CH0:

- 0: BYTE2-1-0
- 1: BYTE2-0-1
- 2: BYTE1-0-2
- 3: BYTE1-2-0
- 4: BYTE0-2-1
- 5: BYTE0-1-2

Configuring DMA2D_IN_SCRAMBLE_SEL_PRE_CH0 to 5 means swapping the pixel order, for example, swapping YUV444 output by the JPEG decoder to VUY444. Note that, this function is only available for 2D DMA RX channel0, not for any other RX channels.

(i) Select the color conversion methods by configuring DMA2D_IN_COLOR_INPUT_SEL_CH0, DMA2D_IN_COLOR_3B_PROC_EN_CH0, and DMA2D_IN_COLOR_OUTPUT_SEL_CH0. See details in Table 37.7-6:

Table 37.7-6. 2D DMA Color Space Conversion Configuration

Color Space Conversion Method	INPUT_SEL_CH0	3B_PROC_EN_CH0	OUTPUT_SEL_CH0
YUV420 → RGB	0	1	1
YUV422 → RGB	0	1	1
YUV444 → RGB	2	1	1
YUV444 (only reverse the pixel order)	2	0	1
No color space conversion	7	N/A	N/A

Note that, this function is only available for 2D DMA RX channel0, not for any other RX channels.

(j) Configure parameters required for the color space conversion formula, including DMA2D_IN_COLOR_PARAM_H0_CH0, DMA2D_IN_COLOR_PARAM_H1_CH0, DMA2D_IN_COLOR_PARAM_M0_CH0, DMA2D_IN_COLOR_PARAM_M1_CH0, DMA2D_IN_COLOR_PARAM_L0_CH0, and DMA2D_IN_COLOR_PARAM_L1_CH0. Note that, this function is only available for 2D DMA RX channel0, not for any other RX channels.

(k) Select whether to reverse the pixel order of the image after 2D DMA color space conversion by configuring DMA2D_IN_SCRAMBLE_SEL_POST_CH0:

- 0: BYTE2-1-0
- 1: BYTE2-0-1
- 2: BYTE1-0-2
- 3: BYTE1-2-0
- 4: BYTE0-2-1
- 5: BYTE0-1-2

Configuration of 5 means reversing the pixel order, for example, reversing the RGB after 2D DMA color space conversion to BGR. Note that, this function is only available for 2D DMA RX channel0, not for any other RX channels.

(l) Configure DMA2D_OUTLINK_ADDR_CH*n* as the 2D DMA TX channel linked list address, configure DMA2D_INLINK_ADDR_CH*n* as the 2D DMA RX channel linked list address.

4. Enable interrupt sources:
   - DMA2D_OUT_INT_ENA_CH*n*_REG
   - DMA2D_IN_INT_ENA_CH*n*_REG
   - JPEG_INT_ENA_REG
5. Start the transmission of 2D DMA channels:
   - 2D DMA TX channel*n*: set DMA2D_OUTLINK_START_CH*n* to 1 and then to 0
   - 2D DMA RX channel*n*: set DMA2D_INLINK_START_CH*n* to 1 and then to 0
6. Start JPEG codec decoding by setting the JPEG_JPEG_START field.
7. Wait till DMA2D_IN_SUC_EOF_CH*n*_INT becomes 1, indicating that the decoded image has been completely written to the memory.

37.7.3 Reset

Users can **reset** JPEG codec entirely or partially, depending on the actual scenarios, at any time during the encoding or decoding process:

- The entire JPEG codec, including all the RAMs and FIFOs as well as all the state machines: set JPEG_SOFT_RST to 1. In this case, the register configuration value will not be reset.

- Only all the RAMs and FIFOs inside the JPEG codec, for example, changing the quantization coefficient tables or Huffman tables: set JPEG_FIFO_RST to 1.
- Only all the state machines inside the JPEG codec, for example, only stop the current encoding or decoding process: set JPEG_FSM_RST to 1.

37.8 Register Summary

The addresses in this section are relative to JEPG Codec base address provided in Table 9.3-2 in Chapter 9 *System and Memory*.

The abbreviations given in Column **Access** are explained in Section *Access Types for Registers*.

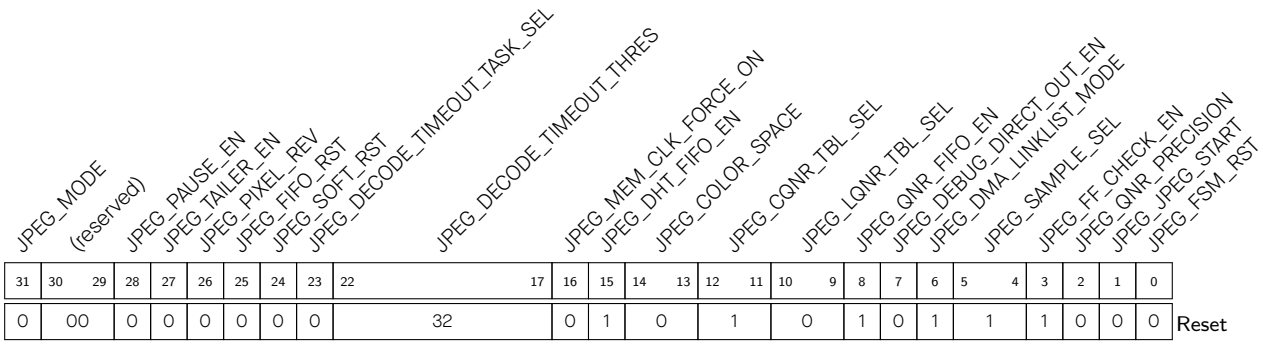
Name	Description	Address	Access
Control and Configuration registers			
JPEG_CONFIG_REG	System level control and configuration register	0x0000	varies
JPEG_DQT_INFO_REG	Quantization coefficient table information register	0x0004	R/W
JPEG_PIC_SIZE_REG	Image size register	0x0008	R/W
JPEG_EXTD_CONFIG_REG	Extended color space conversion configuration register	0x000C	R/W
JPEG_T0QNR_REG	Quantization coefficient table0 register	0x0010	HRO
JPEG_T1QNR_REG	Quantization coefficient table1 register	0x0014	HRO
JPEG_T2QNR_REG	Quantization coefficient table2 register	0x0018	HRO
JPEG_T3QNR_REG	Quantization coefficient table3 register	0x001C	HRO
JPEG_DECODE_CONF_REG	Decoder configuration register	0x0020	varies
JPEG_C0_REG	Color component0 register	0x0024	R/W
JPEG_C1_REG	Color component1 register	0x0028	R/W
JPEG_C2_REG	Color component2 register	0x002C	R/W
JPEG_C3_REG	Color component3 register	0x0030	R/W
JPEG_DHT_INFO_REG	Huffman table information register	0x0034	R/W
JPEG_DHT_TOTLEN_DC0_REG	DC0 Huffman table codeword length register	0x0058	HRO
JPEG_DHT_VAL_DC0_REG	DC0 Huffman table symbol register	0x005C	HRO
JPEG_DHT_TOTLEN_AC0_REG	AC0 Huffman table codeword length register	0x0060	HRO
JPEG_DHT_VAL_AC0_REG	AC0 Huffman table symbol register	0x0064	HRO
JPEG_DHT_TOTLEN_DC1_REG	DC1 Huffman table codeword length register	0x0068	HRO
JPEG_DHT_VAL_DC1_REG	DC1 Huffman table symbol register	0x006C	HRO
JPEG_DHT_TOTLEN_AC1_REG	AC1 Huffman table codeword length register	0x0070	HRO
JPEG_DHT_VAL_AC1_REG	AC1 Huffman table symbol register	0x0074	HRO
JPEG_DHT_CODEMIN_DC0_REG	DC0 Huffman table minimum codeword register	0x0078	HRO
JPEG_DHT_CODEMIN_AC0_REG	AC0 Huffman table minimum codeword register	0x007C	HRO
JPEG_DHT_CODEMIN_DC1_REG	DC1 Huffman table minimum codeword register	0x0080	HRO
JPEG_DHT_CODEMIN_AC1_REG	AC1 Huffman table minimum codeword register	0x0084	HRO
JPEG_SYS_REG	System configuration register	0x00F8	R/W
Interrupt registers			
JPEG_INT_RAW_REG	Interrupt raw status register	0x0038	R/WTC/SS
JPEG_INT_ENA_REG	Interrupt enable register	0x003C	R/W
JPEG_INT_ST_REG	Interrupt masked status register	0x0040	RO
JPEG_INT_CLR_REG	Interrupt clear register	0x0044	WT
Version Register			
JPEG_VERSION_REG	Version control register	0x00FC	R/W

37.9 Registers

The addresses in this section are relative to the JEPG Codec base address provided in Table 9.3-2 in Chapter 9 *System and Memory*.

For how to program reserved fields, please refer to Section Programming Reserved Register Field.

Register 37.1. JPEG_CONFIG_REG (0x0000)



JPEG_FSM_RST Configures whether or not to reset the state machine of JPEG Codec.

0: Invalid. No effect

1: Reset the state machine

(WT)

JPEG_JPEG_START Configures whether or not to start compressing a new image.

0: Invalid. No effect

1: Start compressing a new image

(WT)

JPEG_QNR_PRECISION Configures the quantization coefficient table precision for the encoder.

0: 8-bit precision

1: 16-bit precision

(R/W)

JPEG_FF_CHECK_EN Configures whether or not to add “0x00” after “0xFF”.

0: Not add

1: Add

(R/W)

JPEG_SAMPLE_SEL Configures the format of the image to be compressed.

0: YUV444

1: YUV422

2: YUV420

3: Invalid. No effect

(R/W)

Continued on the next page...

Register 37.1. JPEG_CONFIG_REG (0x0000)

Continued from the previous page...

JPEG_DMA_LINKLIST_MODE Represents 2D DMA linked list mode.
0: Invalid. No effect
1: 2D DMA uses linked lists to configure
(RO)

JPEG_DEBUG_DIRECT_OUT_EN Configures whether out not to enable debug mode for encoder.
0: Disable debug mode, the encoder will work in normal mode
1: Enable debug mode, the input image will be directly output by the encoder
(R/W)

JPEG_QNR_FIFO_EN Configures whether or not to enable FIFO mode when configuring quantization coefficient tables.
0: Disable. Use non-FIFO mode
1: Enable
(R/W)

JPEG_LQNR_TBL_SEL Configures the luminance quantization table ID for the encoder. (R/W)

JPEG_CQNR_TBL_SEL Configures the chrominance quantization table ID for the encoder. (R/W)

JPEG_COLOR_SPACE Configures the original image's color space.
0: RGB888
1: YUV422
2: RGB565
3: GRAY
(R/W)

JPEG_DHT_FIFO_EN Configures whether or not to enable FIFO mode when configuring Huffman tables.
0: Disable. Use non-FIFO mode
1: Enable
Note that to read Huffman tables only non-FIFO mode is supported.
(R/W)

JPEG_MEM_CLK_FORCE_ON Configures whether or not to force on memory's clock gate.
0: Not force on
1: Force on
(R/W)

JPEG_DECODE_TIMEOUT_THRES Configures decode timeout period to trigger JPEG_DECODE_TIMEOUT_INT_RAW. The clock cycles of timeout period = $2^{JPEG_DECODE_TIMEOUT_THRES} - 1$. (R/W)

Continued on the next page...

Register 37.1. JPEG_CONFIG_REG (0x0000)

Continued from the previous page...

JPEG_DECODE_TIMEOUT_TASK_SEL Configures the reset mode upon the decoder timeout.

0: Software uses reset to abort the decoding process

1: Hardware automatically aborts decoding process

(R/W)

JPEG_SOFT_RST Configures whether or not to apply soft reset to the JPEG Codec.

0: Release the soft reset

1: Apply soft reset to the JPEG Codec (the register configuration value will not be reset)

(R/W)

JPEG_FIFO_RST Configures whether or not to apply FIFO reset to the JPEG Codec.

0: Release the FIFO reset

1: Apply FIFO reset to the JPEG Codec

(R/W)

JPEG_PIXEL_REV Configures whether or not to reverse the original image's pixel order.

0: Not reverse

1: Reverse

(R/W)

JPEG_TAILER_EN Configures whether or not to add EOI marker "0xFFD9" at the end of bitstream.

0: Not add

1: Add

(R/W)

JPEG_PAUSE_EN Configures whether or not to pause the JPEG Codec.

0: Not pause

1: Pause

(R/W)

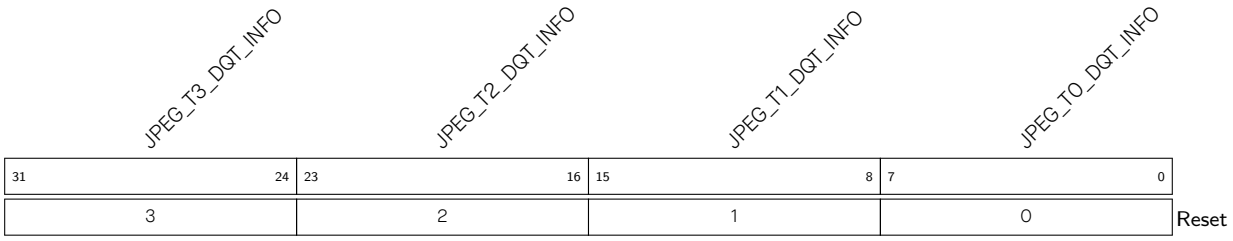
JPEG_MODE Configures if the JPEG Codec is working as an encoder or a decoder.

0: Encoder

1: Decoder

(R/W)

Register 37.2. JPEG_DQT_INFO_REG (0x0004)



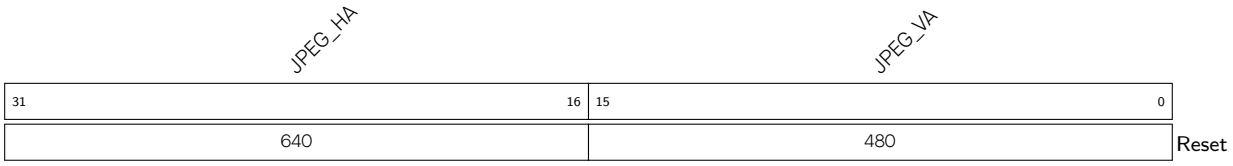
JPEG_T0_DQT_INFO Configures quantization coefficient table0's precision and ID for decoder.
Bit[7:4]: Configures precision
Bit[3:0]: Configures ID
(R/W)

JPEG_T1_DQT_INFO Configures quantization coefficient table1's precision and ID for decoder. See details in JPEG_T0_DQT_INFO. (R/W)

JPEG_T2_DQT_INFO Configures quantization coefficient table2's precision and ID for decoder. See details in JPEG_T0_DQT_INFO. (R/W)

JPEG_T3_DQT_INFO Configures quantization coefficient table3's precision and ID for decoder. See details in JPEG_T0_DQT_INFO. (R/W)

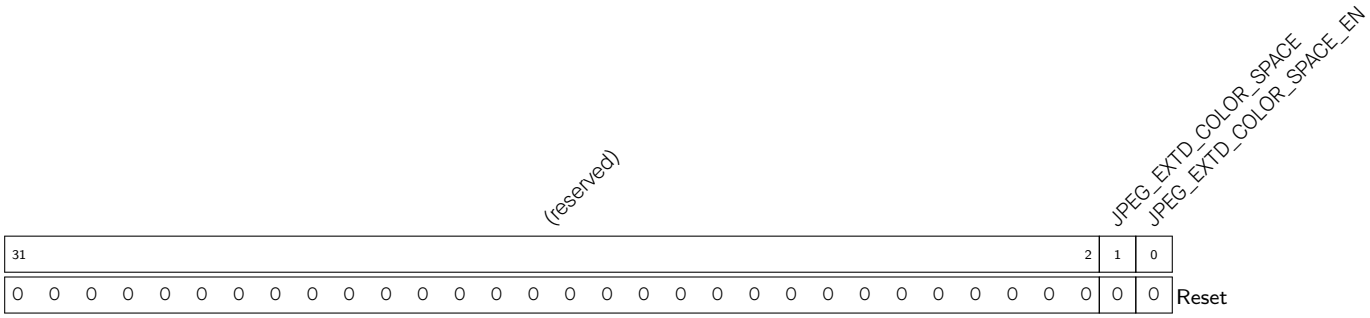
Register 37.3. JPEG_PIC_SIZE_REG (0x0008)



JPEG_VA Configures the image's height.
When JPEG Codec works as an encoder, the maximum configurable bits is 14.
When JPEG Codec works as a decoder, the maximum configurable bits is 16.
(R/W)

JPEG_HA Configures the image's width.
When JPEG Codec works as an encoder, the maximum configurable bits is 14.
When JPEG Codec works as a decoder, the maximum configurable bits is 16.
(R/W)

Register 37.4. JPEG_EXTD_CONFIG_REG (0x000C)



JPEG_EXTD_COLOR_SPACE_EN Configure whether to enable extended color space conversion options.

0: Disable

1: Enable

(R/W)

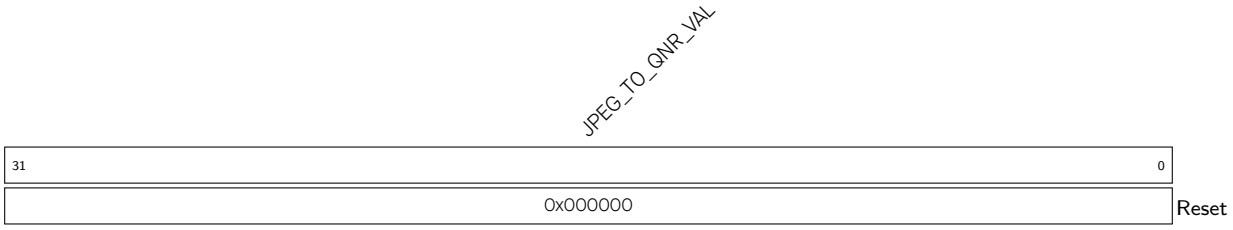
JPEG_EXTD_COLOR_SPACE Configure the extended color space format to be converted. Valid when JPEG_EXTD_COLOR_SPACE_EN configured to 1.

0: YUV444

1: YUV420

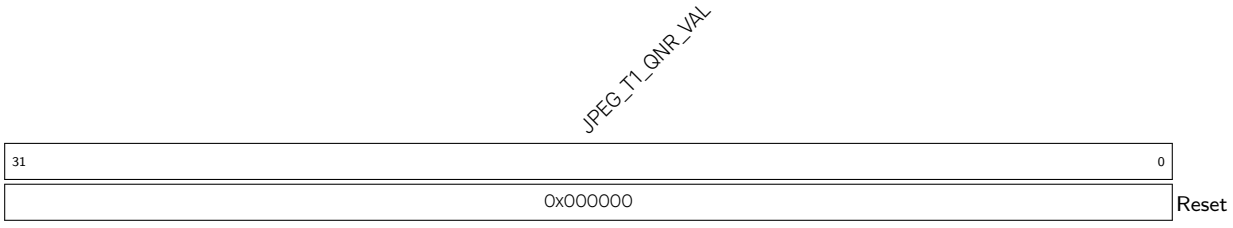
(R/W)

Register 37.5. JPEG_T0QNR_REG (0x0010)



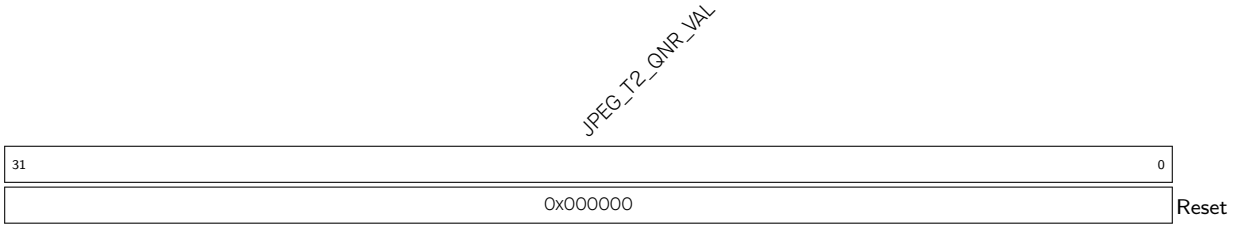
JPEG_T0_QNR_VAL Configures the quantization coefficients of the quantization table 0 in FIFO mode. (HRO)

Register 37.6. JPEG_T1QNR_REG (0x0014)



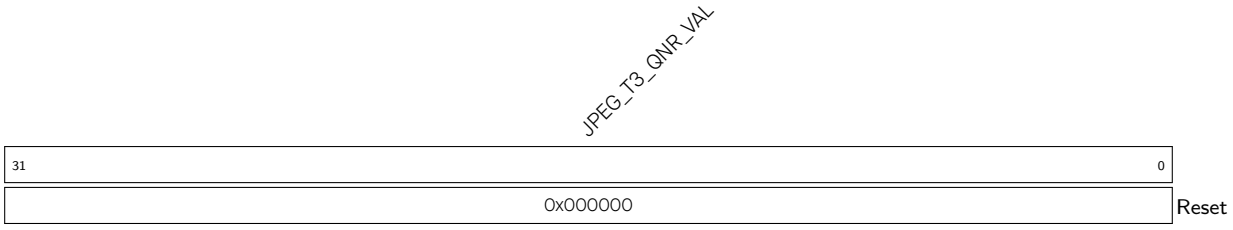
JPEG_T1_QNR_VAL Configures the quantization coefficients of the quantization table 1 in FIFO mode. (HRO)

Register 37.7. JPEG_T2QNR_REG (0x0018)



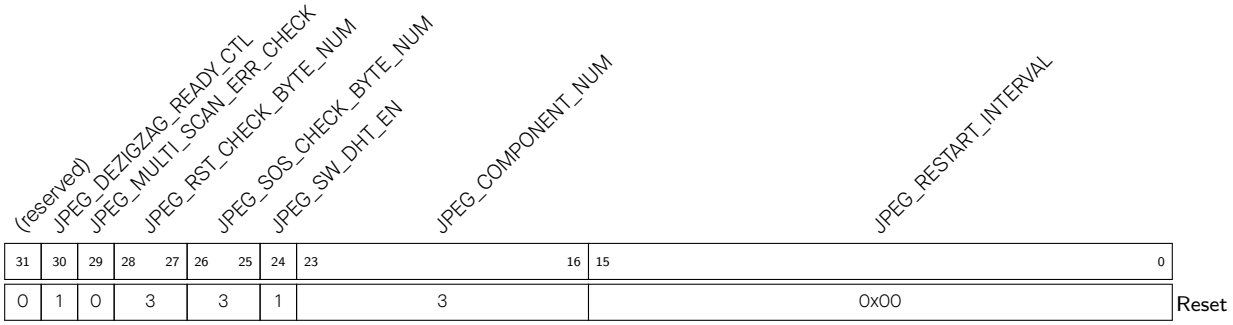
JPEG_T2_QNR_VAL Configures the quantization coefficients of the quantization table 2 in FIFO mode. (HRO)

Register 37.8. JPEG_T3QNR_REG (0x001C)



JPEG_T3_QNR_VAL Configures the quantization coefficients of the quantization table 3 in FIFO mode. (HRO)

Register 37.9. JPEG_DECODE_CONF_REG (0x0020)



JPEG_RESTART_INTERVAL Configures the RST interval in DRI segment when decoding. (R/W)

JPEG_COMPONENT_NUM Configures the number of chrominance components in the image when decoding. (R/W)

JPEG_SW_DHT_EN Represents the Huffman table configured by the software for the decoder. The value is always 1. (RO)

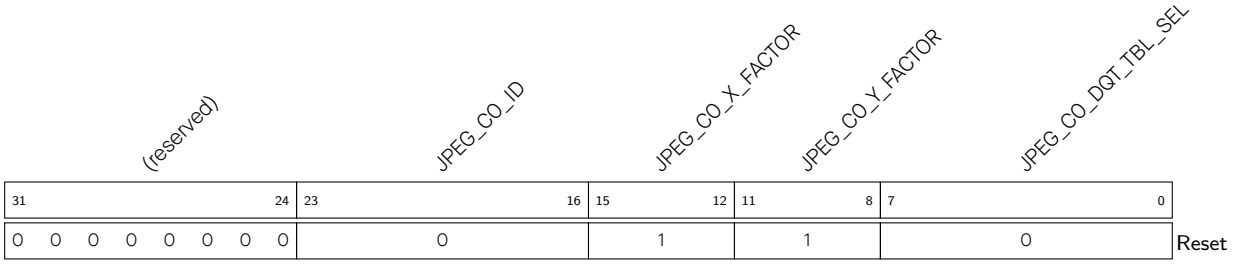
JPEG_SOS_CHECK_BYTE_NUM Configures the byte number to check the next SOS marker in the multi-scan image after one scan is decoded down. The actual check number is JPEG_SOS_CHECK_BYTE_NUM + 1 (R/W)

JPEG_RST_CHECK_BYTE_NUM Configures the byte number to check the next RST marker after one RST interval is decoded down. The actual check number is JPEG_RST_CHECK_BYTE_NUM + 1. (R/W)

JPEG_MULTI_SCAN_ERR_CHECK Reserved for decoder and should not be configured. (R/W)

JPEG_DEZIGZAG_READY_CTL Reserved for decoder and should not be configured. (R/W)

Register 37.10. JPEG_C0_REG (0x0024)



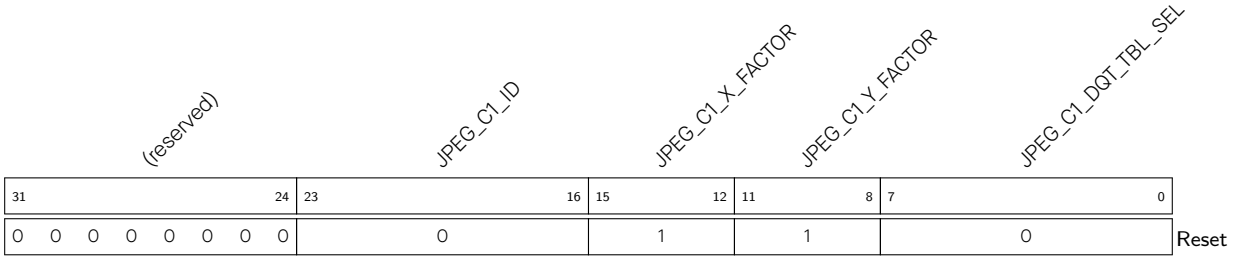
JPEG_C0_DQT_TBL_SEL Configures the selected quantization coefficient table ID for component0 in decoder mode. (R/W)

JPEG_C0_Y_FACTOR Configures the vertical sampling factor of component0 in decoder mode. (R/W)

JPEG_C0_X_FACTOR Configures the horizontal sampling factor of component0 in decoder mode. (R/W)

JPEG_C0_ID Configures the ID of component0 in decoder mode. (R/W)

Register 37.11. JPEG_C1_REG (0x0028)



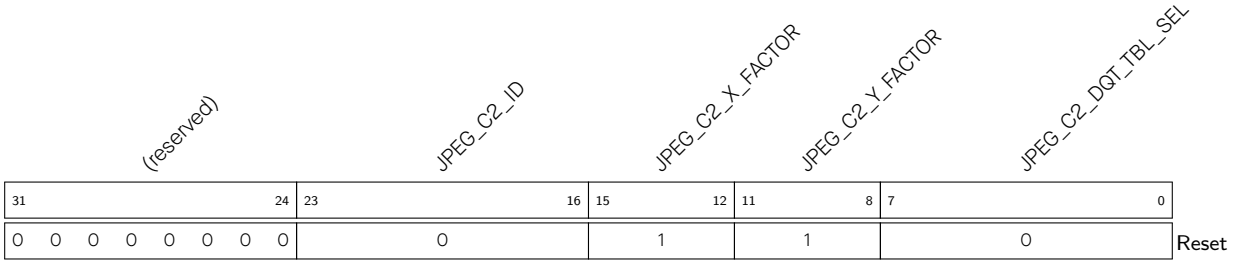
JPEG_C1_DQT_TBL_SEL Configures the selected quantization coefficient table ID for component1 in decoder mode. (R/W)

JPEG_C1_Y_FACTOR Configures the vertical sampling factor of component1 in decoder mode. (R/W)

JPEG_C1_X_FACTOR Configures the horizontal sampling factor of component1 in decoder mode. (R/W)

JPEG_C1_ID Configures the ID of component1 in decoder mode. (R/W)

Register 37.12. JPEG_C2_REG (0x002C)



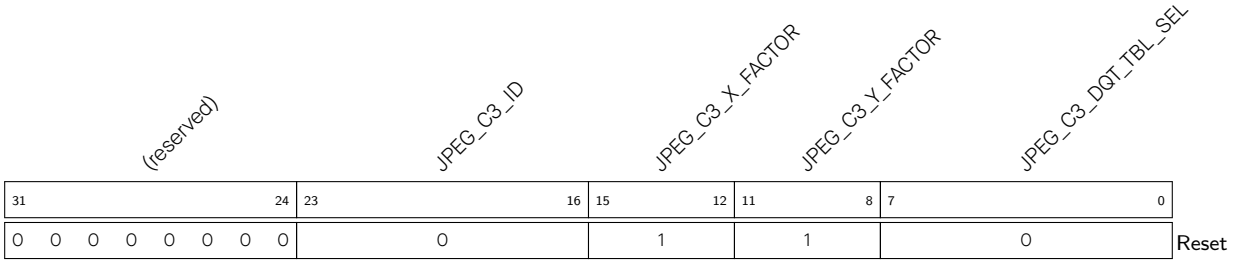
JPEG_C2_DQT_TBL_SEL Configures the selected quantization coefficient table ID for component2 in decoder mode. (R/W)

JPEG_C2_Y_FACTOR Configures the vertical sampling factor of component2 in decoder mode. (R/W)

JPEG_C2_X_FACTOR Configures the horizontal sampling factor of component2 in decoder mode. (R/W)

JPEG_C2_ID Configures the ID of component2 in decoder mode. (R/W)

Register 37.13. JPEG_C3_REG (0x0030)



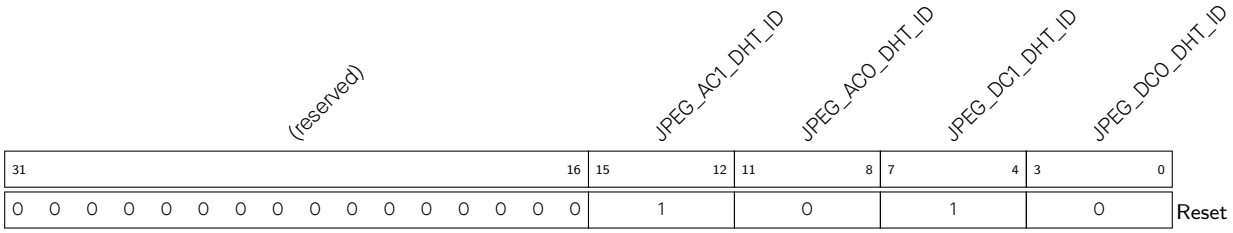
JPEG_C3_DQT_TBL_SEL Configures the selected quantization coefficient table ID for component3 in decoder mode. (R/W)

JPEG_C3_Y_FACTOR Configures the vertical sampling factor of component3 in decoder mode. (R/W)

JPEG_C3_X_FACTOR Configures the horizontal sampling factor of component3 in decoder mode. (R/W)

JPEG_C3_ID Configures the ID of component3 in decoder mode. (R/W)

Register 37.14. JPEG_DHT_INFO_REG (0x0034)



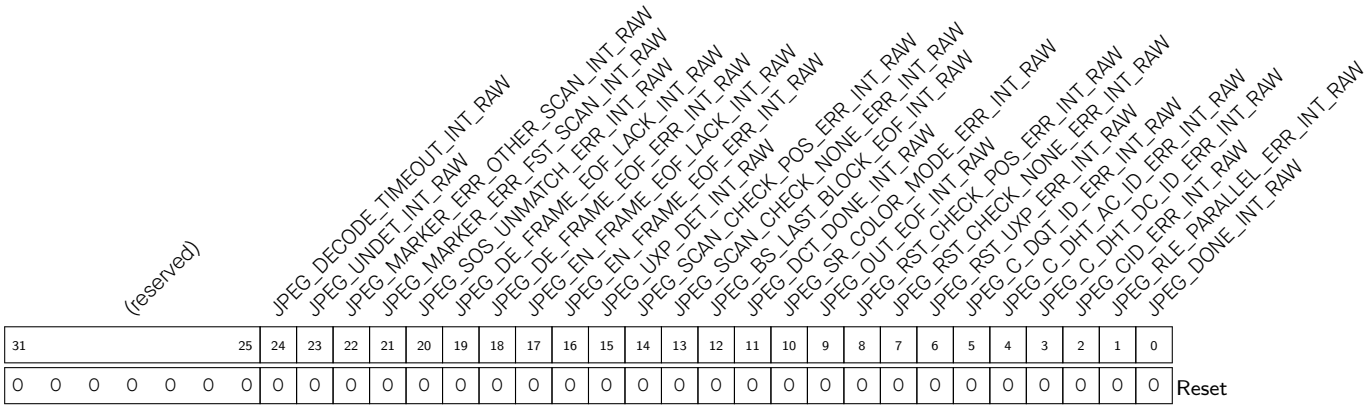
JPEG_DC0_DHT_ID Configures the ID of DC0 Huffman table in decoder mode. (R/W)

JPEG_DC1_DHT_ID Configures the ID of DC1 Huffman table in decoder mode. (R/W)

JPEG_AC0_DHT_ID Configures the ID of AC0 Huffman table in decoder mode. (R/W)

JPEG_AC1_DHT_ID Configures the ID of AC1 Huffman table in decoder mode. (R/W)

Register 37.15. JPEG_INT_RAW_REG (0x0038)



JPEG_DONE_INT_RAW The raw status of JPEG_DONE_INT. (R/WTC/SS)

JPEG_RLE_PARALLEL_ERR_INT_RAW The raw status of JPEG_RLE_PARALLEL_ERR_INT. (R/WTC/SS)

JPEG_CID_ERR_INT_RAW The raw status of JPEG_CID_ERR_INT. (R/WTC/SS)

JPEG_C_DHT_DC_ID_ERR_INT_RAW The raw status of JPEG_C_DHT_DC_ID_ERR_INT. (R/WTC/SS)

JPEG_C_DHT_AC_ID_ERR_INT_RAW The raw status of JPEG_C_DHT_AC_ID_ERR_INT. (R/WTC/SS)

JPEG_C_DQT_ID_ERR_INT_RAW The raw status of JPEG_C_DQT_ID_ERR_INT. (R/WTC/SS)

JPEG_RST_UXP_ERR_INT_RAW The raw status of JPEG_RST_UXP_ERR_INT. (R/WTC/SS)

JPEG_RST_CHECK_NONE_ERR_INT_RAW The raw status of JPEG_RST_CHECK_NONE_ERR_INT. (R/WTC/SS)

Continued on the next page...

Register 37.15. JPEG_INT_RAW_REG (0x0038)

Continued from the previous page...

JPEG_RST_CHECK_POS_ERR_INT_RAW The raw status of JPEG_RST_CHECK_POS_ERR_INT. (R/WTC/SS)

JPEG_OUT_EOF_INT_RAW The raw status of JPEG_OUT_EOF_INT. (R/WTC/SS)

JPEG_SR_COLOR_MODE_ERR_INT_RAW The raw status of JPEG_SR_COLOR_MODE_ERR_INT. This interrupt is invalid. (R/WTC/SS)

JPEG_DCT_DONE_INT_RAW The raw status of JPEG_DCT_DONE_INT. (R/WTC/SS)

JPEG_BS_LAST_BLOCK_EOF_INT_RAW The raw status of JPEG_BS_LAST_BLOCK_EOF_INT. (R/WTC/SS)

JPEG_SCAN_CHECK_NONE_ERR_INT_RAW The raw status of JPEG_SCAN_CHECK_NONE_ERR_INT. (R/WTC/SS)

JPEG_SCAN_CHECK_POS_ERR_INT_RAW The raw status of JPEG_SCAN_CHECK_POS_ERR_INT. (R/WTC/SS)

JPEG_UXP_DET_INT_RAW The raw status of JPEG_UXP_DET_INT. (R/WTC/SS)

JPEG_EN_FRAME_EOF_ERR_INT_RAW The raw status of JPEG_EN_FRAME_EOF_ERR_INT. (R/WTC/SS)

JPEG_EN_FRAME_EOF_LACK_INT_RAW The raw status of JPEG_EN_FRAME_EOF_LACK_INT. (R/WTC/SS)

JPEG_DE_FRAME_EOF_ERR_INT_RAW The raw status of JPEG_DE_FRAME_EOF_ERR_INT. (R/WTC/SS)

JPEG_DE_FRAME_EOF_LACK_INT_RAW The raw status of JPEG_DE_FRAME_EOF_LACK_INT. (R/WTC/SS)

JPEG_SOS_UNMATCH_ERR_INT_RAW The raw status of JPEG_SOS_UNMATCH_ERR_INT. (R/WTC/SS)

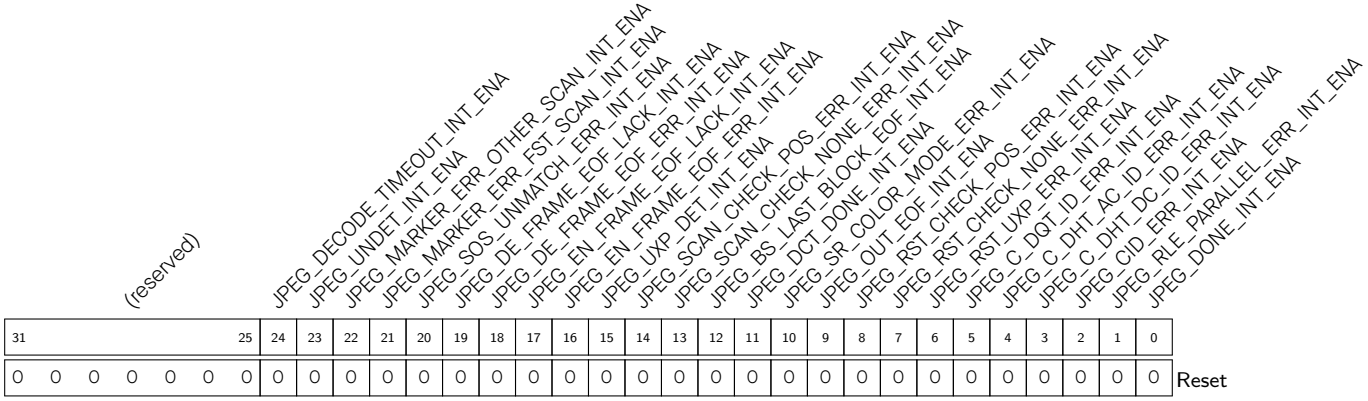
JPEG_MARKER_ERR_FST_SCAN_INT_RAW The raw status of JPEG_MARKER_ERR_FST_SCAN_INT. (R/WTC/SS)

JPEG_MARKER_ERR_OTHER_SCAN_INT_RAW The raw status of JPEG_MARKER_ERR_OTHER_SCAN_INT. (R/WTC/SS)

JPEG_UNDET_INT_RAW The raw status of JPEG_UNDET_INT. (R/WTC/SS)

JPEG_DECODE_TIMEOUT_INT_RAW The raw status of JPEG_DECODE_TIMEOUT_INT. (R/WTC/SS)

Register 37.16. JPEG_INT_ENA_REG (0x003C)



JPEG_DONE_INT_ENA Write 1 to enable JPEG_DONE_INT_ENA. (R/W)

JPEG_RLE_PARALLEL_ERR_INT_ENA Write 1 to enable JPEG_RLE_PARALLEL_ERR_INT. (R/W)

JPEG_CID_ERR_INT_ENA Write 1 to enable JPEG_CID_ERR_INT. (R/W)

JPEG_C_DHT_DC_ID_ERR_INT_ENA Write 1 to enable JPEG_C_DHT_DC_ID_ERR_INT. (R/W)

JPEG_C_DHT_AC_ID_ERR_INT_ENA Write 1 to enable JPEG_C_DHT_AC_ID_ERR_INT. (R/W)

JPEG_C_DQT_ID_ERR_INT_ENA Write 1 to enable JPEG_C_DQT_ID_ERR_INT. (R/W)

JPEG_RST_UXP_ERR_INT_ENA Write 1 to enable JPEG_RST_UXP_ERR_INT. (R/W)

JPEG_RST_CHECK_NONE_ERR_INT_ENA Write 1 to enable JPEG_RST_CHECK_NONE_ERR_INT. (R/W)

JPEG_RST_CHECK_POS_ERR_INT_ENA Write 1 to enable JPEG_RST_CHECK_POS_ERR_INT. (R/W)

JPEG_OUT_EOF_INT_ENA Write 1 to enable JPEG_OUT_EOF_INT. (R/W)

JPEG_SR_COLOR_MODE_ERR_INT_ENA Write 1 to enable JPEG_SR_COLOR_MODE_ERR_INT. (R/W)

JPEG_DCT_DONE_INT_ENA Write 1 to enable JPEG_DCT_DONE_INT. (R/W)

JPEG_BS_LAST_BLOCK_EOF_INT_ENA Write 1 to enable JPEG_BS_LAST_BLOCK_EOF_INT. (R/W)

JPEG_SCAN_CHECK_NONE_ERR_INT_ENA Write 1 to enable JPEG_SCAN_CHECK_NONE_ERR_INT. (R/W)

JPEG_SCAN_CHECK_POS_ERR_INT_ENA Write 1 to enable JPEG_SCAN_CHECK_POS_ERR_INT. (R/W)

JPEG_UXP_DET_INT_ENA Write 1 to enable JPEG_UXP_DET_INT. (R/W)

Continued on the next page...

Register 37.16. JPEG_INT_ENA_REG (0x003C)

Continued from the previous page...

JPEG_EN_FRAME_EOF_ERR_INT_ENA Write 1 to enable JPEG_EN_FRAME_EOF_ERR_INT. (R/W)

JPEG_EN_FRAME_EOF_LACK_INT_ENA Write 1 to enable JPEG_EN_FRAME_EOF_LACK_INT. (R/W)

JPEG_DE_FRAME_EOF_ERR_INT_ENA Write 1 to enable JPEG_DE_FRAME_EOF_ERR_INT. (R/W)

JPEG_DE_FRAME_EOF_LACK_INT_ENA Write 1 to enable JPEG_DE_FRAME_EOF_LACK_INT. (R/W)

JPEG_SOS_UNMATCH_ERR_INT_ENA Write 1 to enable JPEG_SOS_UNMATCH_ERR_INT. (R/W)

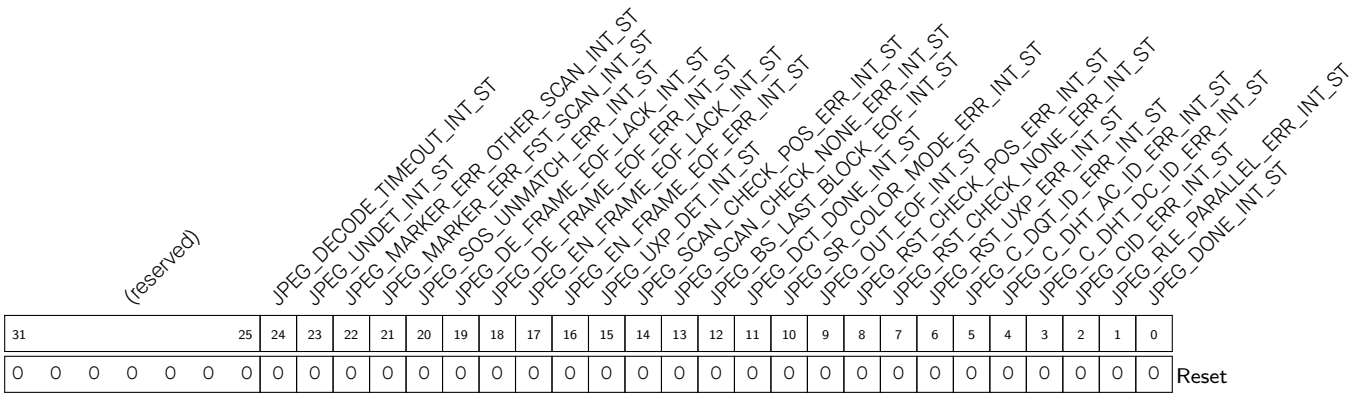
JPEG_MARKER_ERR_FST_SCAN_INT_ENA Write 1 to enable JPEG_MARKER_ERR_FST_SCAN_INT. (R/W)

JPEG_MARKER_ERR_OTHER_SCAN_INT_ENA Write 1 to enable JPEG_MARKER_ERR_OTHER_SCAN_INT. (R/W)

JPEG_UNDET_INT_ENA Write 1 to enable JPEG_UNDET_INT. (R/W)

JPEG_DECODE_TIMEOUT_INT_ENA Write 1 to enable JPEG_DECODE_TIMEOUT_INT. (R/W)

Register 37.17. JPEG_INT_ST_REG (0x0040)



JPEG_DONE_INT_ST The masked interrupt status of JPEG_DONE_INT. (RO)

JPEG_RLE_PARALLEL_ERR_INT_ST The masked interrupt status of JPEG_RLE_PARALLEL_ERR_INT. (RO)

JPEG_CID_ERR_INT_ST The masked interrupt status of JPEG_CID_ERR_INT. (RO)

JPEG_C_DHT_DC_ID_ERR_INT_ST The masked interrupt status of JPEG_C_DHT_DC_ID_ERR_INT. (RO)

JPEG_C_DHT_AC_ID_ERR_INT_ST The masked interrupt status of JPEG_C_DHT_AC_ID_ERR_INT. (RO)

JPEG_C_DQT_ID_ERR_INT_ST The masked interrupt status of JPEG_C_DQT_ID_ERR_INT. (RO)

JPEG_RST_UXP_ERR_INT_ST The masked interrupt status of JPEG_RST_UXP_ERR_INT. (RO)

JPEG_RST_CHECK_NONE_ERR_INT_ST The masked interrupt status of JPEG_RST_CHECK_NONE_ERR_INT. (RO)

JPEG_RST_CHECK_POS_ERR_INT_ST The masked interrupt status of JPEG_RST_CHECK_POS_ERR_INT. (RO)

JPEG_OUT_EOF_INT_ST The masked interrupt status of JPEG_OUT_EOF_INT. (RO)

JPEG_SR_COLOR_MODE_ERR_INT_ST The masked interrupt status of JPEG_SR_COLOR_MODE_ERR_INT. (RO)

JPEG_DCT_DONE_INT_ST The masked interrupt status of JPEG_DCT_DONE_INT. (RO)

JPEG_BS_LAST_BLOCK_EOF_INT_ST The masked interrupt status of JPEG_BS_LAST_BLOCK_EOF_INT. (RO)

JPEG_SCAN_CHECK_NONE_ERR_INT_ST The masked interrupt status of JPEG_SCAN_CHECK_NONE_ERR_INT. (RO)

JPEG_SCAN_CHECK_POS_ERR_INT_ST The masked interrupt status of JPEG_SCAN_CHECK_POS_ERR_INT. (RO)

JPEG_UXP_DET_INT_ST The masked interrupt status of JPEG_UXP_DET_INT. (RO)

Continued on the next page...

Register 37.17. JPEG_INT_ST_REG (0x0040)

Continued from the previous page...

JPEG_EN_FRAME_EOF_ERR_INT_ST The masked interrupt status of JPEG_EN_FRAME_EOF_ERR_INT. (RO)

JPEG_EN_FRAME_EOF_LACK_INT_ST The masked interrupt status of JPEG_EN_FRAME_EOF_LACK_INT. (RO)

JPEG_DE_FRAME_EOF_ERR_INT_ST The masked interrupt status of JPEG_DE_FRAME_EOF_ERR_INT. (RO)

JPEG_DE_FRAME_EOF_LACK_INT_ST The masked interrupt status of JPEG_DE_FRAME_EOF_LACK_INT. (RO)

JPEG_SOS_UNMATCH_ERR_INT_ST The masked interrupt status of JPEG_SOS_UNMATCH_ERR_INT. (RO)

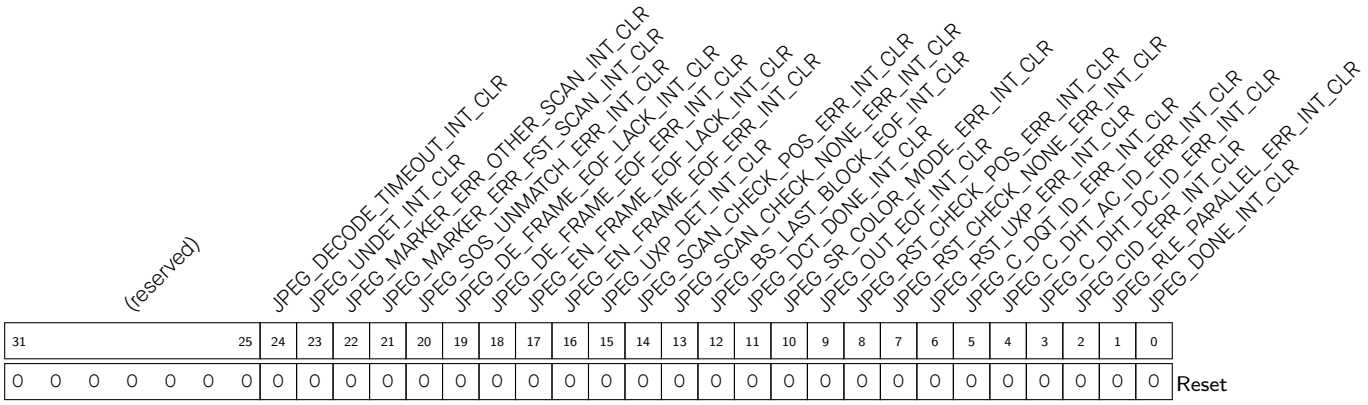
JPEG_MARKER_ERR_FST_SCAN_INT_ST The masked interrupt status of JPEG_MARKER_ERR_FST_SCAN_INT. (RO)

JPEG_MARKER_ERR_OTHER_SCAN_INT_ST The masked interrupt status of JPEG_MARKER_ERR_OTHER_SCAN_INT. (RO)

JPEG_UNDET_INT_ST The masked interrupt status of JPEG_UNDET_INT. (RO)

JPEG_DECODE_TIMEOUT_INT_ST The masked interrupt status of JPEG_DECODE_TIMEOUT_INT. (RO)

Register 37.18. JPEG_INT_CLR_REG (0x0044)



JPEG_DONE_INT_CLR Write 1 to clear JPEG_DONE_INT. (WT)

JPEG_RLE_PARALLEL_ERR_INT_CLR Write 1 to clear JPEG_RLE_PARALLEL_ERR_INT. (WT)

JPEG_CID_ERR_INT_CLR Write 1 to clear JPEG_CID_ERR_INT. (WT)

JPEG_C_DHT_DC_ID_ERR_INT_CLR Write 1 to clear JPEG_C_DHT_DC_ID_ERR_INT. (WT)

JPEG_C_DHT_AC_ID_ERR_INT_CLR Write 1 to clear JPEG_C_DHT_AC_ID_ERR_INT. (WT)

JPEG_C_DQT_ID_ERR_INT_CLR Write 1 to clear JPEG_C_DQT_ID_ERR_INT. (WT)

JPEG_RST_UXP_ERR_INT_CLR Write 1 to clear JPEG_RST_UXP_ERR_INT. (WT)

JPEG_RST_CHECK_NONE_ERR_INT_CLR Write 1 to clear JPEG_RST_CHECK_NONE_ERR_INT. (WT)

JPEG_RST_CHECK_POS_ERR_INT_CLR Write 1 to clear JPEG_RST_CHECK_POS_ERR_INT. (WT)

JPEG_OUT_EOF_INT_CLR Write 1 to clear JPEG_OUT_EOF_INT. (WT)

JPEG_SR_COLOR_MODE_ERR_INT_CLR Write 1 to clear JPEG_SR_COLOR_MODE_ERR_INT. (WT)

JPEG_DCT_DONE_INT_CLR Write 1 to clear JPEG_DCT_DONE_INT. (WT)

JPEG_BS_LAST_BLOCK_EOF_INT_CLR Write 1 to clear JPEG_BS_LAST_BLOCK_EOF_INT. (WT)

JPEG_SCAN_CHECK_NONE_ERR_INT_CLR Write 1 to clear JPEG_SCAN_CHECK_NONE_ERR_INT. (WT)

JPEG_SCAN_CHECK_POS_ERR_INT_CLR Write 1 to clear JPEG_SCAN_CHECK_POS_ERR_INT. (WT)

Continued on the next page...

Register 37.18. JPEG_INT_CLR_REG (0x0044)

Continued from the previous page...

JPEG_UXP_DET_INT_CLR Write 1 to clear JPEG_UXP_DET_INT. (WT)

JPEG_EN_FRAME_EOF_ERR_INT_CLR Write 1 to clear JPEG_EN_FRAME_EOF_ERR_INT. (WT)

JPEG_EN_FRAME_EOF_LACK_INT_CLR Write 1 to clear JPEG_EN_FRAME_EOF_LACK_INT. (WT)

JPEG_DE_FRAME_EOF_ERR_INT_CLR Write 1 to clear JPEG_DE_FRAME_EOF_ERR_INT. (WT)

JPEG_DE_FRAME_EOF_LACK_INT_CLR Write 1 to clear JPEG_DE_FRAME_EOF_LACK_INT. (WT)

JPEG_SOS_UNMATCH_ERR_INT_CLR Write 1 to clear JPEG_SOS_UNMATCH_ERR_INT. (WT)

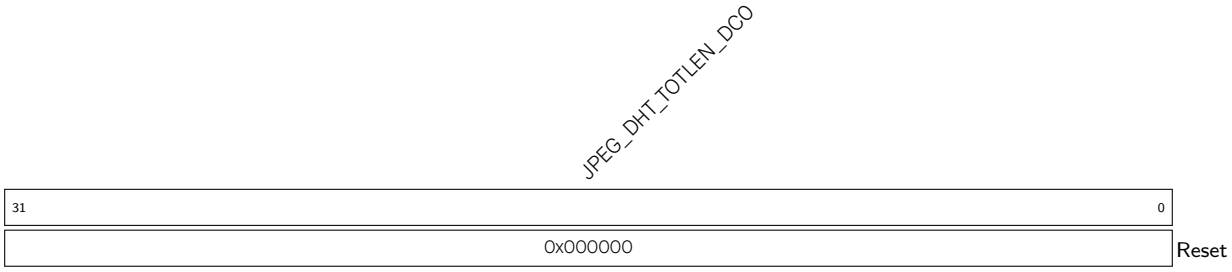
JPEG_MARKER_ERR_FST_SCAN_INT_CLR Write 1 to clear JPEG_MARKER_ERR_FST_SCAN_INT. (WT)

JPEG_MARKER_ERR_OTHER_SCAN_INT_CLR Write 1 to clear JPEG_MARKER_ERR_OTHER_SCAN_INT. (WT)

JPEG_UNDET_INT_CLR Write 1 to clear JPEG_UNDET_INT. (WT)

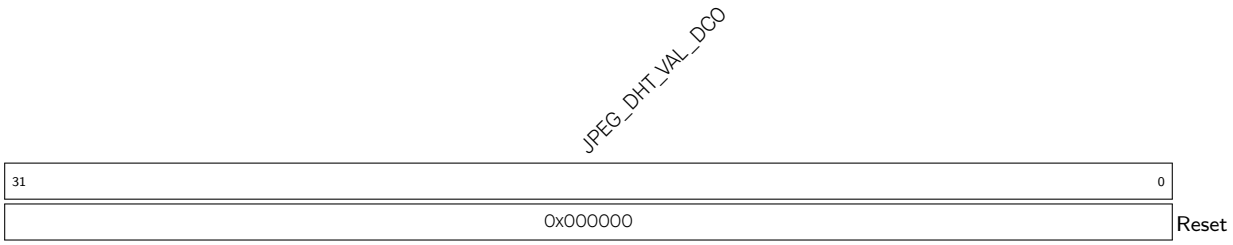
JPEG_DECODE_TIMEOUT_INT_CLR Write 1 to clear JPEG_DECODE_TIMEOUT_INT. (WT)

Register 37.19. JPEG_DHT_TOTLEN_DC0_REG (0x0058)



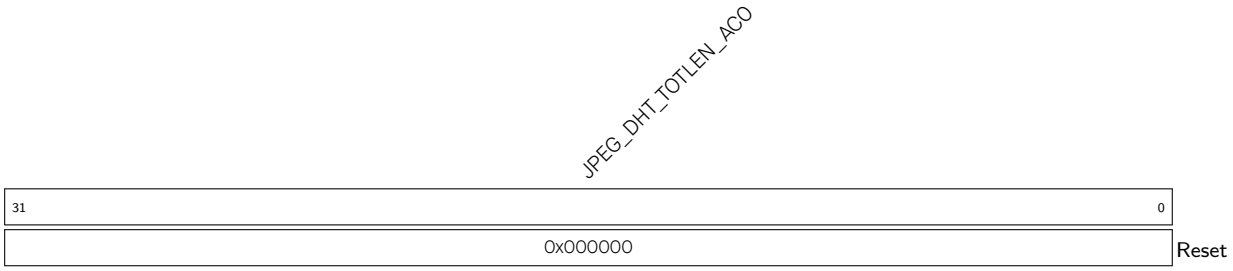
JPEG_DHT_TOTLEN_DC0 Configures the number of codeword with a codeword length of 1 to 16 bits for DC0 Huffman table in FIFO mode. (HRO)

Register 37.20. JPEG_DHT_VAl_DC0_REG (0x005C)



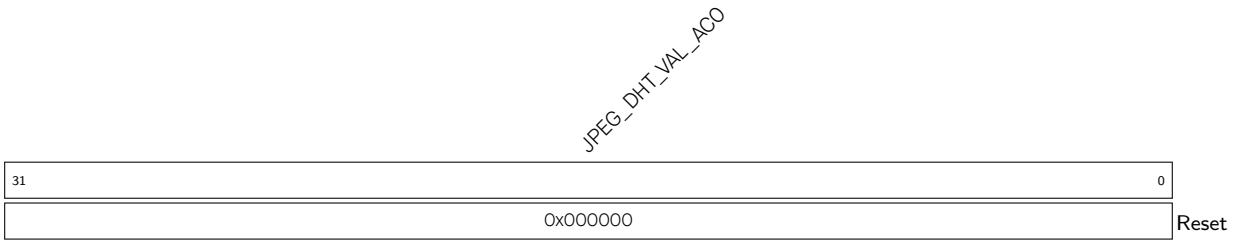
JPEG_DHT_VAL_DC0 Configures the symbols corresponding to the decoded codeword for DC0 Huffman table in FIFO mode. (HRO)

Register 37.21. JPEG_DHT_TOTLEN_AC0_REG (0x0060)



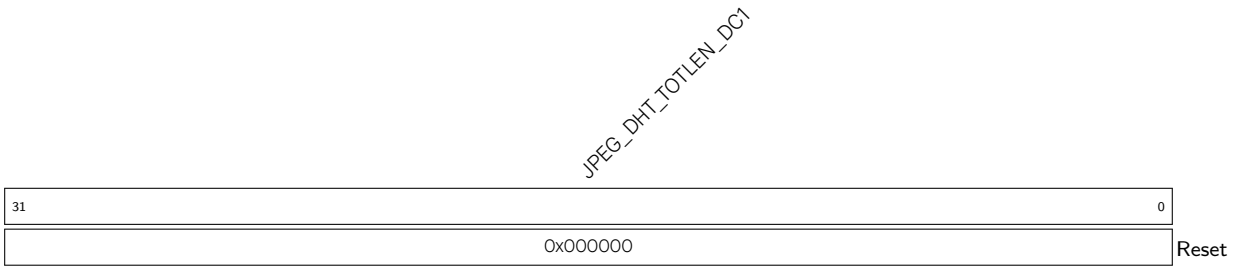
JPEG_DHT_TOTLEN_AC0 Configures the number of codeword with a codeword length of 1 to 16 bits for AC0 Huffman table in FIFO mode. (HRO)

Register 37.22. JPEG_DHT_VAl_AC0_REG (0x0064)



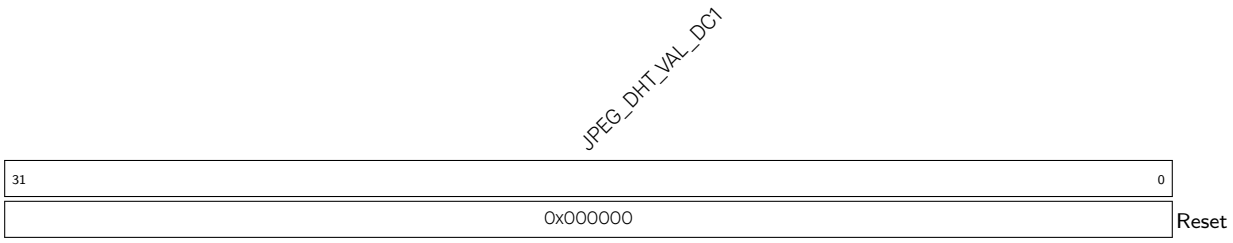
JPEG_DHT_VAL_AC0 Configures the symbols corresponding to the decoded codeword for AC0 Huffman table in FIFO mode. (HRO)

Register 37.23. JPEG_DHT_TOTLEN_DC1_REG (0x0068)



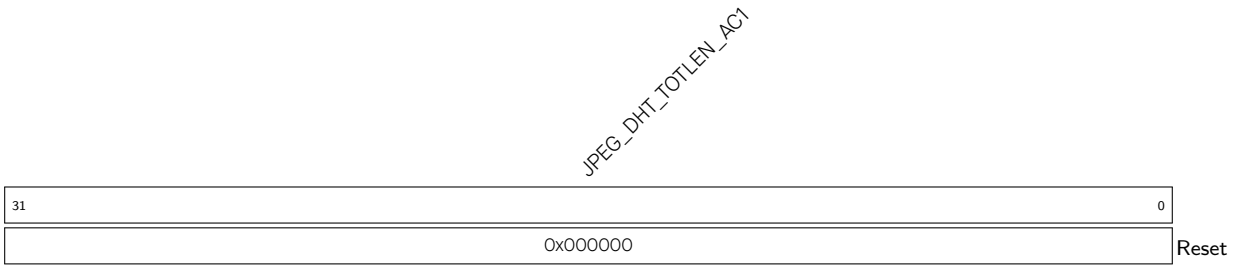
JPEG_DHT_TOTLEN_DC1 Configures the number of codeword with a codeword length of 1 to 16 bits for DC1 Huffman table in FIFO mode. (HRO)

Register 37.24. JPEG_DHT_VAl_DC1_REG (0x006C)



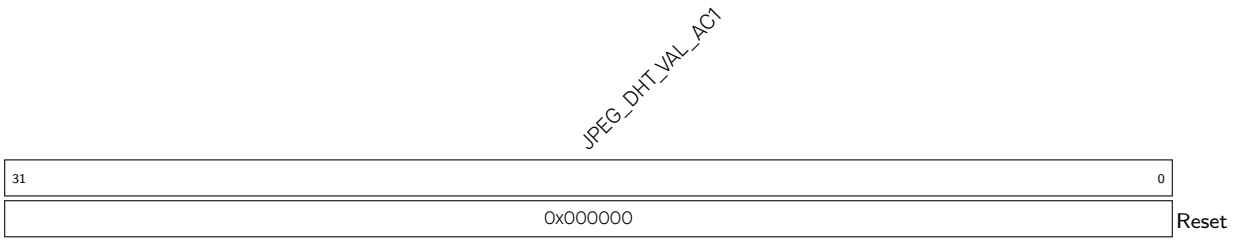
JPEG_DHT_VAL_DC1 Configures the symbols corresponding to the decoded codeword for DC1 Huffman table in FIFO mode. (HRO)

Register 37.25. JPEG_DHT_TOTLEN_AC1_REG (0x0070)



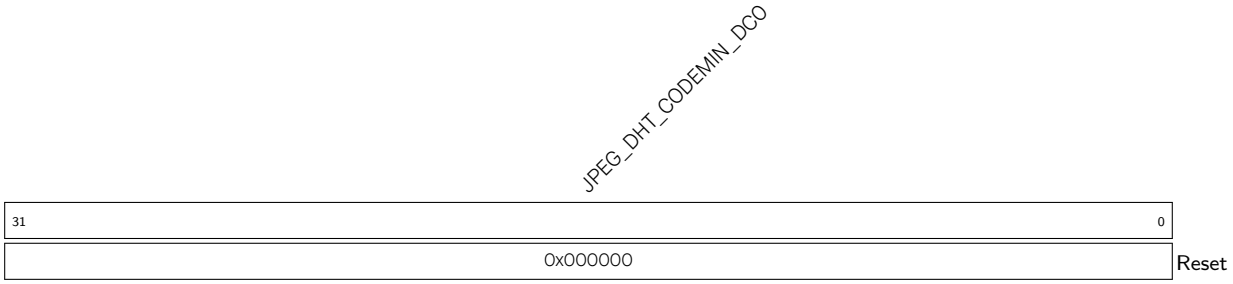
JPEG_DHT_TOTLEN_AC1 Configures the number of codeword with a codeword length of 1 to 16 bits for AC1 Huffman table in FIFO mode. (HRO)

Register 37.26. JPEG_DHT_VAl_AC1_REG (0x0074)



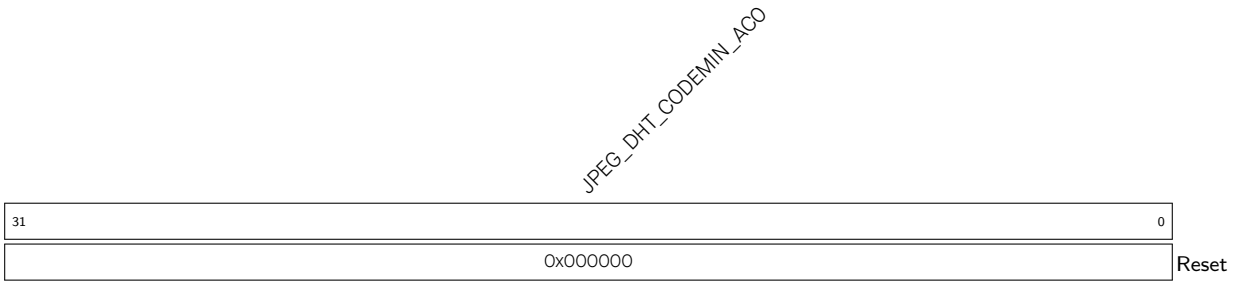
JPEG_DHT_VAL_AC1 Configures the symbols corresponding to the decoded codeword for AC1 Huffman table in FIFO mode. (HRO)

Register 37.27. JPEG_DHT_CODEMIN_DC0_REG (0x0078)



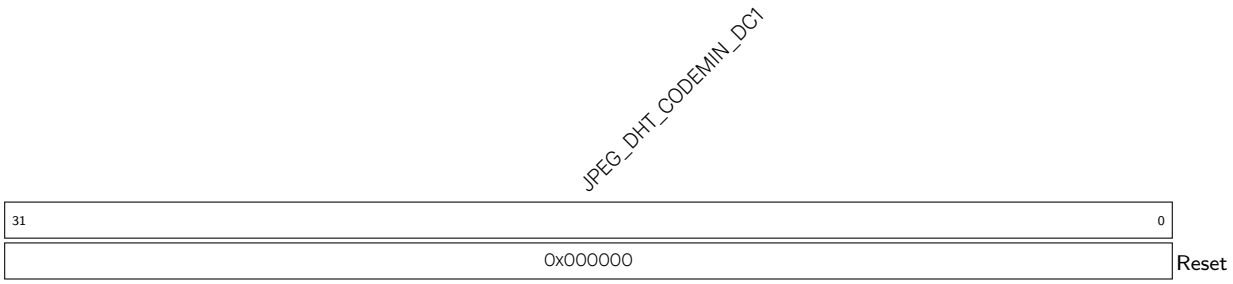
JPEG_DHT_CODEMIN_DC0 Configures the minimum codeword with a codeword length of 1 to 16 bits for DC0 Huffman table in FIFO mode. The codeword should be left shifted to the MSB position of a 16-bit word. (HRO)

Register 37.28. JPEG_DHT_CODEMIN_AC0_REG (0x007C)



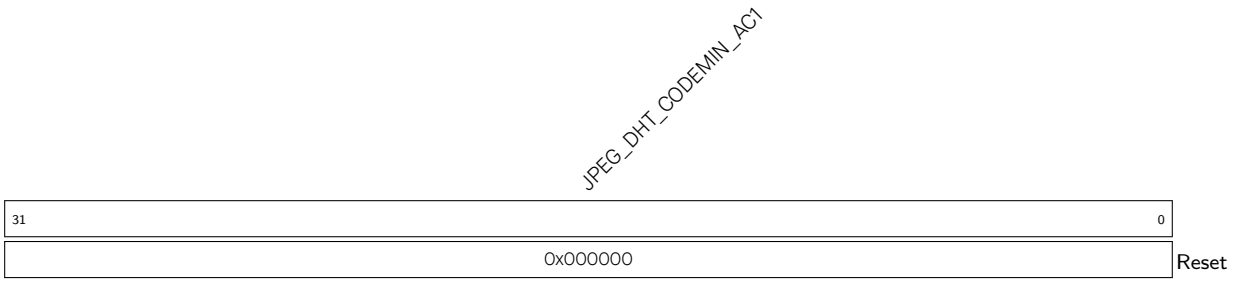
JPEG_DHT_CODEMIN_AC0 Configures the minimum codeword with a codeword length of 1 to 16 bits for AC0 Huffman table in FIFO mode. The codeword should be left shifted to the MSB position of a 16-bit word. (HRO)

Register 37.29. JPEG_DHT_CODEMIN_DC1_REG (0x0080)



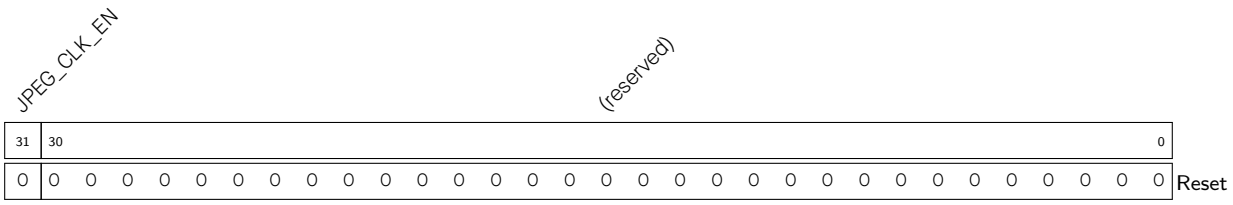
JPEG_DHT_CODEMIN_DC1 Configures the minimum codeword with a codeword length of 1 to 16 bits for DC1 Huffman table in FIFO mode. The codeword should be left shifted to the MSB position of a 16-bit word. (HRO)

Register 37.30. JPEG_DHT_CODEMIN_AC1_REG (0x0084)



JPEG_DHT_CODEMIN_AC1 Configures the minimum codeword with a codeword length of 1 to 16 bits for AC1 Huffman table in FIFO mode. The codeword should be left shifted to the MSB position of a 16-bit word. (HRO)

Register 37.31. JPEG_SYS_REG (0x00F8)



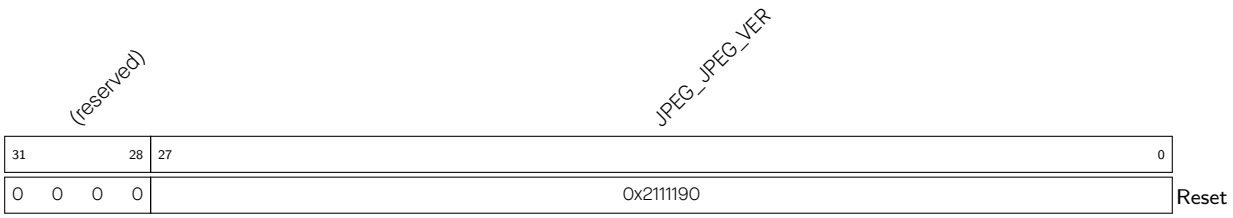
JPEG_CLK_EN Configures whether or not to force open register clock gate.

0: Open the clock gate only when the application writes registers

1: Force open the clock gate for register

(R/W)

Register 37.32. JPEG_VERSION_REG (0x00FC)



JPEG_JPEG_VER Version control. (R/W)

Chapter 38

Image Signal Processor (ISP)

38.1 Introduction

ESP32-P4 includes an image signal processor (ISP), which is a pipeline composed of various image processing algorithms.

ISP reads image data from the DVP or MIPI-CSI camera interface, or from the system memory directly, and writes the processed image data to the system memory through VDMA.

The SoC architecture of ISP is shown in Figure 38.1-1. ISP must work with other modules to read and write data and can not work alone.

- When outputting images, ISP uses CSI_Bridge to convert data to AXI interface and stores the images to system memory through the VDMA controller. For more information about VDMA, see Chapter 7 *VDMA Controller (VDMA)*
- When the input image comes from the system memory, ISP must acquire the image data through VDMA
- When the input image comes from the MIPI-CSI camera interface,
  - ISP must acquire the image data through CSI HOST. For details, see Chapter 42 *MIPI CSI*
  - CSI_CM can preprocess it to perform 8-bit or 16-bit bit-swapping, color-channel reordering for RGB888, RGB565, or YUV422, downsample RGB888 to RGB565, or downsample YUV422 to YUV420
- When the input image comes from the DVP camera interface, ISP acquire the image data through the DVP IO. For details, see Chapter 10 *GPIO Matrix and IO MUX*

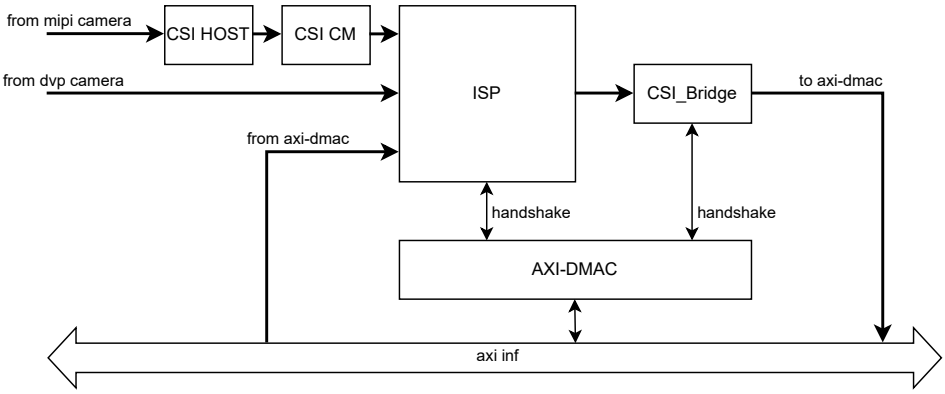


Figure 38.1-1. ESP32-P4 ISP SoC Architecture

38.2 Terminology

MIPI-CSI	Camera serial interface, a high-speed serial interface for cameras complaint with MIPI specifications
Image Interface	Image data interface, the output interface of CSI HOST Classified into Image Interface 32 and Image Interface 64 based on bit width
DVP	Digital video parallel interface, generally composed of vsync, hsync, de, and data signals
RAW	Unprocessed data directly output from an image sensor, typically divided into R, Gr, Gb, and B channels Classified into RAW8, RAW10, RAW12, etc., based on bit width
RGB	Colored image format composed of red, green, and blue colors Classified into RGB888, RGB565, etc., based on the bit width of each color
YUV	Colored image format composed of luminance and chrominance Classified into YUV444, YUV422, YUV420, etc., based on the data arrangement

38.3 Feature List

ISP supports the following features:

- maximum resolution: 1920 x 1080
- three input channels: MIPI-CSI, DVP, and VDMA
- input formats: RAW8, RAW10, and RAW12
- output formats: RAW8, RGB888, RGB565, YUV422, and YUV420
- pipeline features:
  - Black level correction (BLC)
  - Defect pixel correction (DPC)
  - Bayer filter (BF)
  - Lens shading correction (LSC)
  - Demosaic
  - White balance gain (WBG)
  - Color correction matrix (CCM)
  - Gamma correction
  - RGB2YUV
  - Sharpen
  - Contrast/hue/saturation/luminance adjustment (COLOR)
  - YUV_limit
  - YUV2RGB
  - Crop
  - Automatic exposure statistics (AE)
  - Automatic focus statistics (AF)

- Automatic white balance statistics (AWB)
- Histogram statistics (HIST)

38.4 Architectural Overview

ISP architecture is shown in Figure 38.4-1, consisting of ISP_Header, ISP_Pipeline, ISP_Tail, and CSI_Bridge.

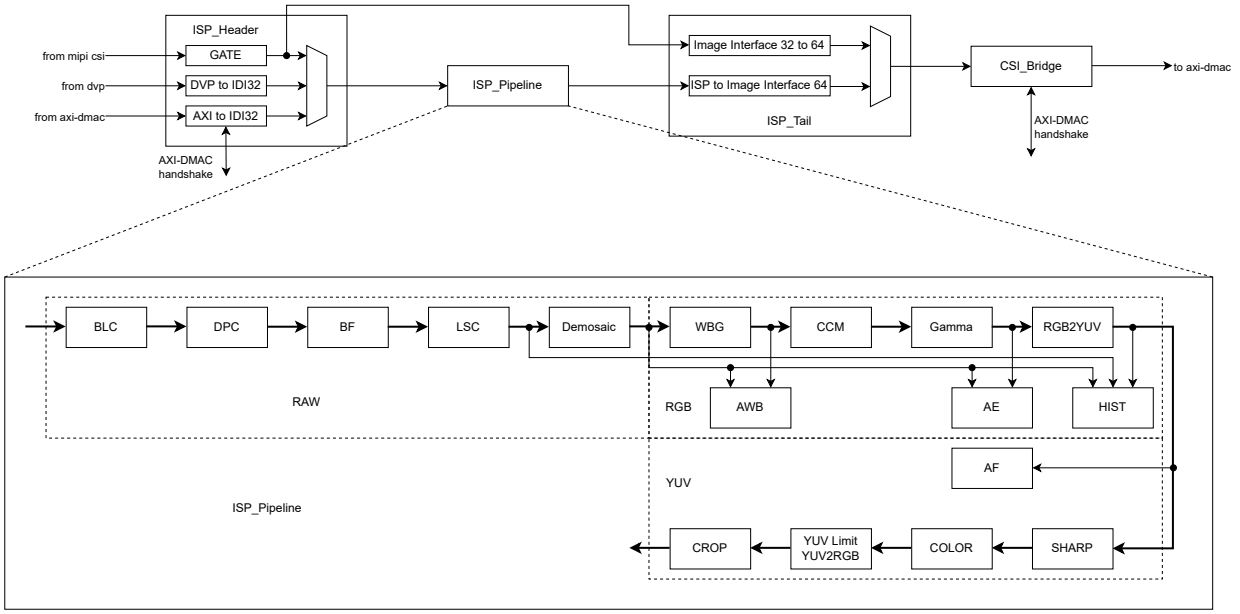


Figure 38.4-1. ISP IP Architecture

- ISP_Header selects data from various input sources based on the configuration and buffers it. Then it transmits the buffered data to ISP_Pipeline in a standardized format.
- ISP_Pipeline serves as the core module of the ISP, containing all the algorithms for image processing.
- ISP_Tail has two input sources, MIPI-CSI and ISP_Pipeline. When ISP is disabled, Image Interface 32 data from MIPI-CSI is converted into Image Interface 64 format by ISP_Tail and transmitted to CSI_Bridge. When ISP is enabled, the output of ISP_Pipeline is cropped and converted into Image Interface 64 format by ISP_Tail, and then transmitted to CSI_Bridge.
- As the output of the ISP, CSI_Bridge transmits the images from ISP to the system memory through VDMA.

38.5 Functional Description

38.5.1 ISP_Header

ISP_Header serves as the input of ISP. The main features are:

- Select image data as input source based on the configuration of ISP_IN_SRC
- Buffer image data, and convert images to the format required by ISP_Pipeline based on the configured type and size of image data

- When MIPI-CSI is the input, set the input type via ISP_DATA_TYPE
- When DVP is the input, set ISP_DATA_TYPE to 0, then set the data type via ISP_CAM_DATA_TYPE
- When VDMA is the input, set the input type via ISP_DATA_TYPE, and configure ISP_DMA_DATA_TYPE accordingly

38.5.2 ISP_Pipeline

ISP_Pipeline is the core of ISP, housing all the image processing algorithms. Each algorithm module can be enabled or disabled via the corresponding bit of ISP_CNTL_REG. Except for modules related to color space conversion (demosaic, RGB2YUV, and YUV2RGB), other modules can be enabled or disabled dynamically without the need to shut down the ISP for reconfiguration. Moreover, once enabled, each algorithm module generates an end-of-frame interrupt, capable of triggering an interrupt after processing each image frame.

38.5.2.1 Black Level Correction (BLC)

This module is used to subtract the black level offset of the sensor. The black level offset to be subtracted for each Bayer channel is configured via ISP_BLC_Rx_VALUE. With the black level offset subtracted, the pixel range will be reduced, which can be stretched back to 0-255 via ISP_BLC_Rx_STRETCH.

There are four Bayer channels in total. The mapping between channel and index is as follows:

- R0 refers to the top-left channel
- R1 refers to the top-right channel
- R2 refers to the bottom-left channel
- R3 refers to to the bottom-right channel

38.5.2.2 Defective Pixel Correction (DPC)

This module removes defective pixels from the input image. DPC supports both dynamic correction and static correction, which are controlled by ISP_STA_EN and ISP_DYN_EN, respectively. The two correction modes can be enabled simultaneously.

Dynamic correction automatically detects defective pixels based on the input pixel values and replaces them with the median value. Two defective pixel detection algorithms are supported:

- **Algorithm 0:** Compute the minimum value *min8* and the maximum value *max8* among the eight neighboring pixels of the current pixel. If the current pixel value is greater than max8 + ISP_DPC_THRESHOLD_H, or less than min8 - ISP_DPC_THRESHOLD_L, the pixel is classified as defective.
- **Algorithm 1:** Let the current pixel value be *pix*. Compute the mean value *est* of the eight neighboring pixels, and the absolute difference *dif* = $|est - pix|$. Define *avg* = (sum of the eight neighbors + $8 \times pix$) / 16. Detection is performed in two steps:
  - **Step 1:** Determine the maximum value *max8* among the eight neighboring pixels. If $pix < max8 \times$ ISP_DPC_THRESHOLD_H and $pix > max8 \times$ ISP_DPC_THRESHOLD_L, the pixel is considered non-defective; otherwise, proceed to Step 2.
  - **Step 2:** The pixel is classified as defective if either of the following conditions is met:

    * $dif > avg \times$ ISP_DPC_FACTOR_DARK and $est \geq pix$
    * $dif > (255 - avg) \times$ ISP_DPC_FACTOR_BRIG and $est < pix$

Static correction is implemented using a defective pixel coordinate lookup table (LUT). Pixels whose coordinates are stored in the LUT are replaced with the median value. The defective pixel coordinates must be obtained through static calibration. During calibration, neither static nor dynamic correction can be enabled.

The calibration procedure is as follows:

- Black image defective pixel calibration
  - Set ISP_DPC_BLACK_EN = 1
  - Set ISP_DPC_CHECK_EN = 1
  - Set ISP_DPC_EN = 1
  - Input a black image and wait for the calibration to complete
  - Set ISP_DPC_CHECK_EN = 0
  - Software reads all defective pixel coordinates for the black image from the LUT registers
- White image defective pixel calibration
  - Set ISP_DPC_BLACK_EN = 0
  - Set ISP_DPC_CHECK_EN = 1
  - Set ISP_DPC_EN = 1
  - Input a white image and wait for the calibration to complete
  - Set ISP_DPC_CHECK_EN = 0
  - Software reads all defective pixel coordinates for the white image from the LUT registers
- Software merges the defective pixel coordinates obtained from the black and white images, removes duplicates, and writes the final list back to the LUT

The LUT read and write procedure is described in Section 38.5.2.4. By setting ISP_LUT_NUM = 1, the defective pixel coordinate LUT for DPC is selected. This LUT has a depth of 512, allowing up to 512 defective pixel coordinates to be stored.

38.5.2.3 Bayer Filter (BF)

This module is designed for denoising input images in the Bayer domain. The denoising strength can be adjusted using ISP_SIGMA, and the denoising template can be fine-tuned using ISP_GAU_TEMPLATExx.

38.5.2.4 Lens Shading Correction (LSC)

This module is utilized for lens shading correction in the Bayer domain. It corrects the R, Gr, Gb, and B channels independently by dividing each channel into a 32 x 32 grid (equivalent to a 64 x 64 division for the entire image) and stores the correction coefficients for each channel in a lookup table (LUT).

Before using this module, calibration is necessary to obtain the LUT correction coefficients. Specifically, first we capture a white image in a scene with uniform ambient lighting. Divide the image into a 32 x 32 grid for

each channel, and use the luminance at the center of the image as a reference. Calculate the luminance gain at each grid vertex based on this reference luminance. For grids extending beyond the image boundaries on the right or bottom, calculate the gain at the image boundary points. Then, use a linear relationship based on the gain of the previous point to calculate the gain at the next grid boundary. Finally, store these gains in the LUT in a left-to-right, top-to-bottom order to complete the initialization.

In the context of a 1920 x 1080 resolution image, each channel operates at 960 x 540 resolution and is divided into a 32 x 32 grid, consisting of 30 grids horizontally and 17 grids vertically. Assuming the image's luminance center aligns with its geometric center, we derive the reference luminance ref_lum from the average luminance of pixels around coordinates (480, 270). Moving methodically from left to right and top to bottom, derive the actual pixel luminance act_lum for each grid vertex. Using the formula $gain = ref_lum/act_lum$, we compute the correction coefficient, which is then stored sequentially in the LUT.

In cases where grids extend beyond the image boundaries, such as the vertical coordinate of the last grid reaching 543 (beyond the maximum vertical coordinate of 539), we leverage the correction coefficients gain_0 at coordinate 511 and gain_1 at coordinate 539. Then the gain for coordinate 543 can be calculated based on $gain = gain_0 + (gain_1 - gain_0)/(539 - 511) * (543 - 511)$.

LSC employs two LUTs for storing gains related to specific color channels: Gb and B are stored in one LUT, while R and Gr are stored in the other. Each LUT has a depth of 560. The number of gain coefficients per channel, determined by the image resolution, must not exceed this value. The gain values are represented as 10-bit fixed-point numbers, with the high 2 bits representing the integer part and the low 8 bits the fractional part. The storage order within the LUTs is detailed in Table 38.5-1. The LUTs can be accessed through specific registers:

- A write operation writes the data from ISP_LUT_WDATA_REG into the LUT.
- A read operation stores the data read from the LUT into ISP_LUT_RDATA_REG.
- Trigger LUT operations via ISP_LUT_CMD_REG. Each write to this register initiates a read or write operation on the LUT. It is required to write to all the fields within this register at the same time.
  - ISP_LUT_CMD: Set to 0 for a read operation and 1 for a write operation
  - ISP_LUT_NUM: Set to 0 to select the LSC LUT
  - ISP_LUT_ADDR: Specifies the LUT address to access. If bits [11:10] are 0, LUT_Gb_B is selected; if bits [11:10] are 1, LUT_R_Gr is selected

Table 38.5-1. Gain Storage Sequence in LSC LUTs

LUT sel	bit [19:10]	bit [9:0]
LUT_Gb_B	Gb	B
LUT_R_Gr	R	Gr

Before enabling LSC, ISP_LSC_XTABLESIZE should be configured to set the number of horizontal correction coefficients. The number is determined by $ISP_LSC_XTABLESIZE = fix((line_width - 1)/2/32) + 2$, where $fix()$ denotes rounding down. For example, if the input RAW image has a horizontal width of 1920 pixels, then $ISP_LSC_XTABLESIZE = fix((1920 - 1)/2/32) + 2 = 29 + 2 = 31$.

38.5.2.5 Demosaic

The demosaicing process converts Bayer images to RGB888 images. The conversion effect can be fine-tuned using ISP_DEMOSAIC_GRAD_RATIO. This register field comprises 2 integer bits and 4 fractional bits, with a default value of 1.0.

38.5.2.6 White Balance Gain (WBG)

This module performs white balance adjustment on the input image. The adjustment is achieved by multiplying the pixel value of each color channel by its corresponding ISP_WBG_R/B/G gain coefficient. Each gain coefficient consists of a 4-bit integer part and an 8-bit fractional part.

38.5.2.7 Color Correction Matrix (CCM)

CCM is a correction matrix for image pixels that can adjust RGB888 pixels, including white balance adjustments. The relationship between the output R', G', B' and the input R, G, B is described by the following matrix. The parameters RR, RG, RB, GR, GG, GB, BR, BG, and BB can be configured with ISP_CCM_xx. Each parameter is a 13-bit fixed-point number, where bit [12] is the sign bit, bits [11:0] represent the absolute value of the number, bits [11:8] the integer part, and bits [7:0] the fractional part.

$$\begin{bmatrix} R' \\ G' \\ B' \end{bmatrix} = \begin{bmatrix} RR & RG & RB \\ GR & GG & GB \\ BR & BG & BB \end{bmatrix} \times \begin{bmatrix} R \\ G \\ B \end{bmatrix} \tag{38.1}$$

38.5.2.8 Gamma Correction

This module is used to adjust the gamma curve of an image. The gamma curve should be configured before gamma correction is enabled. Gamma curves can be configured using registers, with each R, G, B channel having an independent curve consisting of 16 sampling points. For the X component (input value), the values are indirectly obtained through ISP_GAMMA_R/G/B_Xn (where n ranges from 00 to 0F). The Y component (output value) can be directly set using ISP_GAMMA_R/G/B_Yn (where n ranges from 00 to 0F). Once configuration is complete, writing 1 to ISP_GAMMA_UPDATE applies the gamma curve settings. The gamma curve is illustrated in Figure 38.5-1.

For the X component, taking the R channel as an example, the following conditions must be satisfied:

$$X(n) - X(n-1) = 2^{ISP_GAMMA_R_X(n)}$$

$$\sum(2^{ISP_GAMMA_R_X(n)}) = 256, n = 00\sim0F$$

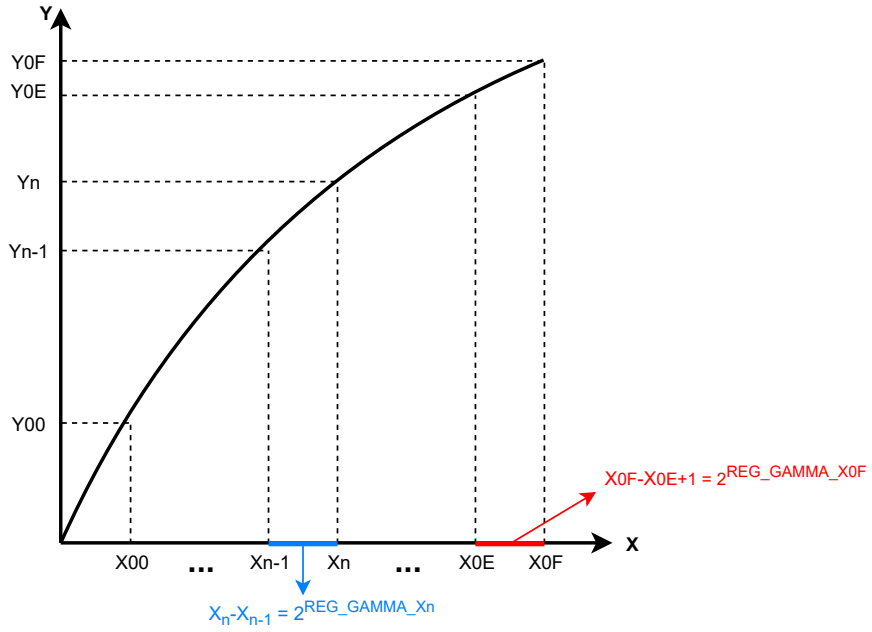


Figure 38.5-1. Gamma Curve

38.5.2.9 RGB2YUV

The RGB2YUV module converts RGB888 pixels into full-range YUV422 format. The conversion protocol can be configured with ISP_YUV_MODE.

38.5.2.10 Sharpen

The sharpen module is used to sharpen images. This module separates the low-frequency and high-frequency components of the image using a low-pass filter template (such as a mean template or Gaussian template) via ISP_SHARP_FILTER_COExx. For low-frequency components, no processing will be done. For high-frequency components, each pixel undergoes the following processing to achieve sharpening:

- Pixels with frequencies below ISP_SHARP_THRESHOLD_LOW are set to 0.
- Pixels with frequencies above ISP_SHARP_THRESHOLD_LOW but below ISP_SHARP_THRESHOLD_HIGH are multiplied by the coefficient ISP_SHARP_AMOUNT_LOW.
- Pixels with frequencies above ISP_SHARP_THRESHOLD_HIGH are multiplied by the coefficient ISP_SHARP_AMOUNT_HIGH.

At the end of each frame, the maximum value of high-frequency component pixels in that frame can be obtained by reading ISP_SHARP_GRADIENT_MAX. This value can serve as a reference for configuring settings for the next frame.

38.5.2.11 Contrast/Hue/Saturation/Luminance Adjustment (COLOR)

This module is designed to modify the contrast, saturation, hue, and luminance of an image. Contrast, saturation, and luminance are configured through ISP_COLOR_CONTRAST, ISP_COLOR_SATURATION, and ISP_COLOR_BRIGHTNESS, respectively. Hue adjustment is achieved by combining ISP_COLOR_HUE_H and ISP_COLOR_HUE to form a 9-bit control value.

38.5.2.12 YUV_Limit and YUV2RGB

The YUV_Limit module converts full-range YUV422 pixels to limited-range YUV422 pixels. This module is activated only when RGB2YUV is enabled, YUV2RGB is disabled, and ISP_YUV_RANGE is set to 1. With this configuration, the ISP output type is either YUV422 or YUV420.

YUV2RGB converts full-range YUV422 pixels to RGB888 format. The conversion protocol can be configured with ISP_YUV_MODE.

Note:
For full-range YUV, Y, U, and V channels range from 0 to 255.
For limited-range YUV, Y channel ranges from 16 to 235, and U and V channels range from 16 to 240.

38.5.2.13 Cropping (CROP)

The CROP module performs image cropping. The start and end coordinates in the X and Y directions are configured using ISP_CROP_X_START, ISP_CROP_X_END, ISP_CROP_Y_START, and ISP_CROP_Y_END, respectively. The start coordinates must be even, and the end coordinates must be odd.

38.5.2.14 Automatic Exposure Statistics (AE)

Auto Exposure (AE) uses 25 statistical sub-windows for image luminance statistics, forming a 5 x 5 grid of statistical window. The size of these windows is set using the register ISP_AE_BX/BY_REG. Each sub-window is numbered as shown in Figure 38.5-2. The average luminance of each sub-window, represented as an 8-bit value, can be read with ISP_AE_Bxx_MEAN. These statistics can be utilized to implement automatic exposure algorithms for cameras.

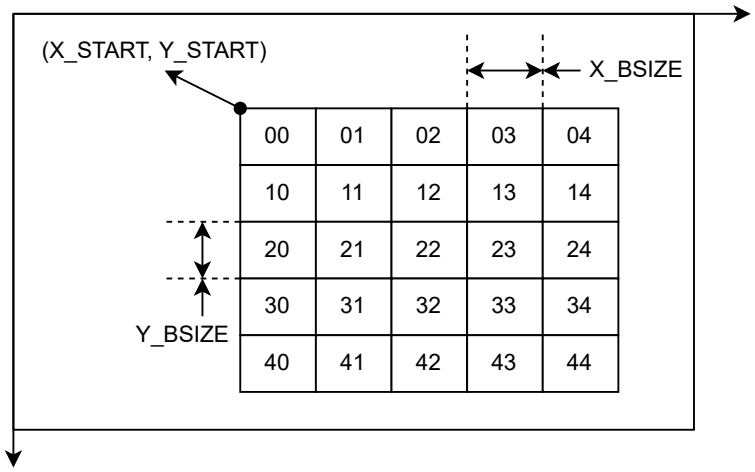


Figure 38.5-2. Sub-Window Numbering

AE utilizes two data sampling points: the output of the Demosaic module and the output of the Gamma module. These sampling points can be configured using ISP_AE_SELECT.

AE also includes a simple luminance monitoring module, implemented either by hardware or software. This module calculates the average luminance of all sub-windows every ISP_AE_MONITO_RPERIOD frames. If the average luminance falls outside the thresholds set by ISP_AE_MONITOR_TH/TL, interrupt for AE luminance

monitoring will be triggered. Note that the luminance monitoring module does not support setting different weights for individual sub-windows. Therefore, if you wish to implement monitoring with different weights (for instance, prioritize highlights), you need to perform software-triggered manual statistics and process the statistical data accordingly.

38.5.2.15 Automatic Focus Statistics (AF)

AF gathers luminance and sharpness information within specified statistical windows based on the output of the sharpen module. There are 3 independent statistical windows, and the size and position of each window can be configured using ISP_AF_H/VSCALE_A/B/C_REG. The module supports automatic and manual trigger modes for statistics collection. The results of luminance and sharpness statistics for each window can be accessed using ISP_AF_LUMA/B/C and ISP_AF_SUMA/B/C. These statistics can be used to implement automatic focus algorithms for cameras.

AF also includes a focus scene monitoring module, which users can choose to implement hardware-based monitoring with this module or implement via software. This module evaluates cumulative changes in luminance and sharpness across all statistical windows as criteria. It performs statistics every ISP_AF_ENV_PERIOD frames. If consecutive statistics show a difference from baseline values for luminance and sharpness that exceeds the specified threshold, an interrupt is triggered indicating a change in the focus scene that requires refocusing. The baseline values are automatically updated after each automatic or manually triggered statistics collection. There are three methods for setting the threshold:

- Directly set the luminance and sharpness thresholds using ISP_AF_ENV_USER_THRESHOLD_LUM and ISP_AF_ENV_USER_THRESHOLD_SUM
- If ISP_AF_ENV_USER_THRESHOLD_SUM is 0, use the corresponding baseline sharpness value multiplied by ISP_AF_ENV_THRESHOLD (4 fractional bits) as the sharpness threshold
- If ISP_AF_ENV_USER_THRESHOLD_LUM is 0, use the corresponding baseline luminance value multiplied by ISP_AF_ENV_THRESHOLD (4 fractional bits) as the luminance threshold

38.5.2.16 Automatic White Balance Statistics (AWB)

AWB gathers white balance information for specified windows containing white patches. The coordinates of the window can be set using ISP_AWB_H/VSCALE_REG. White patches are filtered based on luminance limits ISP_AWB_MAX/MIN_LUM, R/G limits ISP_AWB_MAX/MIN_RG, and B/G limits ISP_AWB_MAX/MIN_BG.

The statistics include the number of white patches and the accumulated values of the R, G, B components for all white patches. These statistics can be obtained using ISP_AWBO_WHITE_CNT and ISP_AWBO_ACC_R/G/B. They can be used to implement automatic white balance algorithms in cameras.

In addition, AWB supports dividing the window into 5 × 5 sub-windows, which are configured via ISP_AWB_X_START, ISP_AWB_X_BSIZE, ISP_AWB_Y_START, and ISP_AWB_Y_BSIZE. The sub-window range must not exceed the range of the main window, as shown in figure 38.5-3. The statistics results are written to the AWB LUT and can be accessed by setting ISP_LUT_NUM = 2. The detailed access procedure can be found in Section 38.5.2.4. The statistics are stored in the order of sub-windows from left to right and from top to bottom. Each sub-window occupies four consecutive addresses, which store, from low to high addresses, the number of white patches, and the accumulated values of the R, G, B components for white patches.

AWB relies on two data sampling points: the output of Demosaic and the output of CCM, which can be configured with ISP_AWB_SAMPLE.

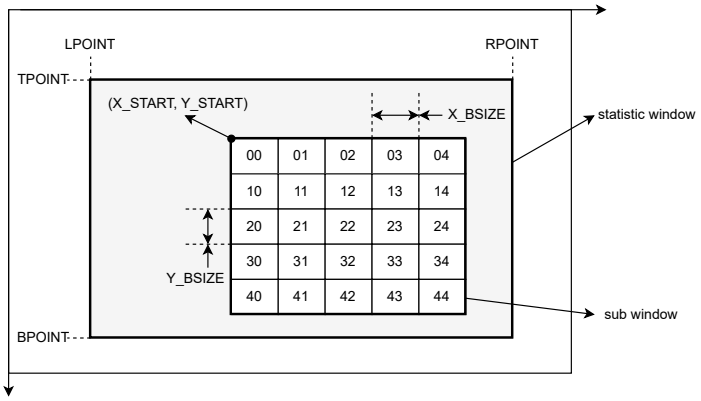


Figure 38.5-3. AWB window

38.5.2.17 Histogram Statistics (HIST)

HIST calculates brightness information for the image through statistical analysis. This module can define a statistical window, which can be divided into 5 x 5 sub-windows similar to AE. The configuration is done using registers ISP_HIST_OFFS_REG and ISP_HIST_SIZE_REG. The weight of each sub-window can be independently adjusted with ISP_HIST_WEIGHT_xx.

The statistics are presented as a histogram with 16 intervals. The x-axis of the histogram is defined using ISP_HIST_SEG_x_x, and the final statistics can be retrieved using ISP_HIST_BIN_x.

HIST involves three data sampling points: BF, demosaic, and RGB2YUV, corresponding to RAW, RGB, and YUV data sampling points respectively. These can be configured using ISP_HIST_MODE. When the sampling point is set to RGB, RGB weighting coefficients can be adjusted via ISP_HIST_COEFF_R/G/B.

38.5.3 ISP_Tail

ISP_Tail is the post-processing section that handles data from two input sources, ultimately outputting in Image Interface 64 format to CSI_Bridge.

When the ISP is turned off, ISP_Tail receives Image Interface 32 data from the MIPI CSI HOST, converts it to Image Interface 64 format, and outputs it to CSI_Bridge. The byte order of the input Image Interface 32 data can be adjusted using ISP_BYTE_ENDIAN_ORDER.

When the ISP is turned on, ISP_Tail receives data from ISP_Pipeline and processes them before outputting to CSI_Bridge. This processing includes:

- Converting RGB888 to RGB565 based on register configurations
- Converting YUV422 to YUV420 based on register configurations
- Converting data from ISP_Pipeline to Image Interface 64 format

38.5.4 Sequence Control

For modules involving matrix operations, it is necessary to buffer multiple lines of data before starting operations. In such cases, it is crucial for the module to generate an output sequence when the last few lines of image data are buffered. This output sequence can be controlled using ISP_BF/DEMOSAIC/SHARP_TAIL_PIXEN_PULSE_TH/TL, to prevent buffer overflow in the CSI_Bridge due to excessive speed. Please exercise caution when setting this register too large, as it may disrupt the normal data sequence. TH must be smaller than ISP_HADR_NUM-1, and it is recommended to set ISP_BF/DEMOSAIC/SHARP_TAIL_PIXEN_PULSE_TH/TL to a value less than or equal to 8.

38.5.5 Shadow Registers

Some ISP modules support shadow registers. The supported modules include:

- BLC
- DPC
- BF
- WBG
- CCM
- GAMMA
- SHARP
- COLOR

For the GAMMA module, the shadow register update method is fixed and must strictly follow the procedure described in Section 38.5.2.8. For all other modules, the update method can be configured via ISP_SHADOW_UPDATE_SEL.

38.5.6 CSI_CM

For MIPI CSI input, the CSI_CM module can perform simple data conversions by configuring CSI_BRIG_HOST_CM_CTRL_REG, including:

- Swapping high and low 8-bit data
- Swapping high and low 16-bit data
- Adjusting RGB888 color channel order (pixels per line must be a multiple of 4)
- Down-sampling from RGB888 to RGB565 (pixels per line must be a multiple of 4)
- Adjusting RGB565 color channel order (pixels per line must be a multiple of 2)
- Adjusting YUV422 color channel order (pixels per line must be a multiple of 2)
- Down-sampling from YUV422 to YUV420 (pixels per line must be a multiple of 16)

The module structure is shown in Figure 38.5-4.

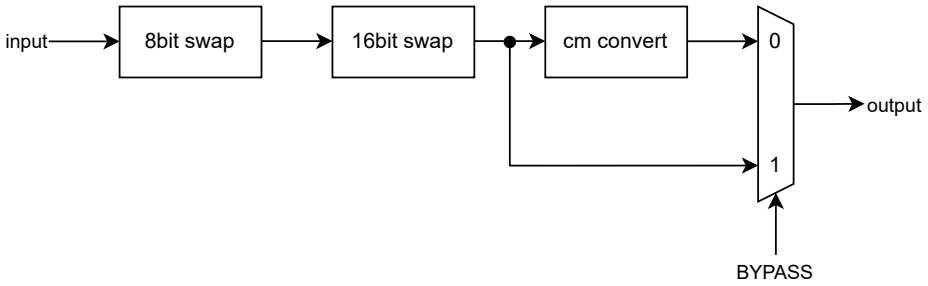


Figure 38.5-4. ESP32-P4 CSI CM Architecture

38.5.7 CSI_Bridge

CSI_Bridge serves as the interface between the ISP and the system memory. This module utilizes VDMA to transmit data from the ISP into the system memory. The image can be transferred either as a single block or divided into multiple blocks, which can be configured via CSI_BRIG_DMA_BURST_LEN and CSI_BRIG_DMABLK_SIZE.

- CSI_BRIG_DMA_BURST_LEN configures the amount of 64-bit data in a VDMA burst transfer.
- CSI_BRIG_DMABLK_SIZE configures the number of VDMA bursts in a VDMA block transfer. If an image frame is transferred with only one block, and this block contains n VDMA bursts, then configure this field to a value greater than n. It is recommended to set it to the maximum value, 0x1FFF.

38.5.8 Color Mode and Byte Order

38.5.8.1 Output Pixel Layout Format

When the ISP is disabled, ISP_Tail directly receives Image Interface 32 data from the MIPI CSI HOST. The pixel layout can be referenced from Chapter 42 *MIPI CSI*. In this case, the Image Interface 32 data output from CSI HOST can go through two levels of byte ordering in ISP_Tail and CSI_Bridge, refer to Byte Order for details.

After the ISP is enabled and the data has been processed by the ISP_Pipeline, the layout format of the output pixel is as shown in Table 38.5-2.

Table 38.5-2. ISP Output Pixel Layout Format

Color Mode	Addr + 7	Addr + 6	Addr + 5	Addr + 4	Addr + 3	Addr + 2	Addr + 1	Addr + 0
RAW8	RAW7[7:0]	RAW6[7:0]	RAW5[7:0]	RAW4[7:0]	RAW3[7:0]	RAW2[7:0]	RAW1[7:0]	RAW0[7:0]
RGB888	G2[7:0]	B2[7:0]	R1[7:0]	G1[7:0]	B1[7:0]	R0[7:0]	G0[7:0]	B0[7:0]
	B5[7:0]	R4[7:0]	G4[7:0]	B4[7:0]	R3[7:0]	G3[7:0]	B3[7:0]	R2[7:0]
	R7[7:0]	G7[7:0]	B7[7:0]	R6[7:0]	G6[7:0]	B6[7:0]	R5[7:0]	G5[7:0]
RGB565	R3[4:0]G3[5:3]	G3[2:0]B3[4:0]	R2[4:0]G2[5:3]	G2[2:0]B2[4:0]	R1[4:0]G1[5:3]	G1[2:0]B1[4:0]	R0[4:0]G0[5:3]	G0[2:0]B0[4:0]
YUV422	Y3[7:0]	V2[7:0]	Y2[7:0]	U2[7:0]	Y1[7:0]	V0[7:0]	Y0[7:0]	U0[7:0]
YUV420 (Odd-numbered rows)	Y4[7:0]	U4[7:0]	Y3[7:0]	Y2[7:0]	U2[7:0]	Y1[7:0]	Y0[7:0]	U0[7:0]
	U10[7:0]	Y9[7:0]	Y8[7:0]	U8[7:0]	Y7[7:0]	Y6[7:0]	U6[7:0]	Y5[7:0]
	Y15[7:0]	Y14[7:0]	U14[7:0]	Y13[7:0]	Y12[7:0]	U12[7:0]	Y11[7:0]	Y10[7:0]
YUV420 (Even-numbered rows)	Y4[7:0]	V4[7:0]	Y3[7:0]	Y2[7:0]	V2[7:0]	Y1[7:0]	Y0[7:0]	V0[7:0]
	V10[7:0]	Y9[7:0]	Y8[7:0]	V8[7:0]	Y7[7:0]	Y6[7:0]	V6[7:0]	Y5[7:0]
	Y15[7:0]	Y14[7:0]	V14[7:0]	Y13[7:0]	Y12[7:0]	V12[7:0]	Y11[7:0]	Y10[7:0]

38.5.8.2 Byte Order

CSI_Bridge can reorder the bytes of the input 64-bit data via CSI_BRIG_BYTE_ENDIAN_ORDER. ISP_Tail can also reorder the bytes via ISP_BYTE_ENDIAN_ORDER. The byte ordering functionality within ISP_Tail differs from that of CSI_Bridge, as illustrated in Figure 38.5-5.

- ISP_Tail byte ordering: This operation applies to its Image Interface 32 input. It reverses the high and low bytes of the 32-bit input data and then concatenates them to output 64-bit data. This step only functions when the ISP is disabled and is mainly used for processing Image Interface 32 YUV422 and YUV420 input types.
- CSI_Bridge byte ordering: This operation acts on the 64-bit input data (i.e., the output data from ISP_Tail) by reversing its high and low bytes.

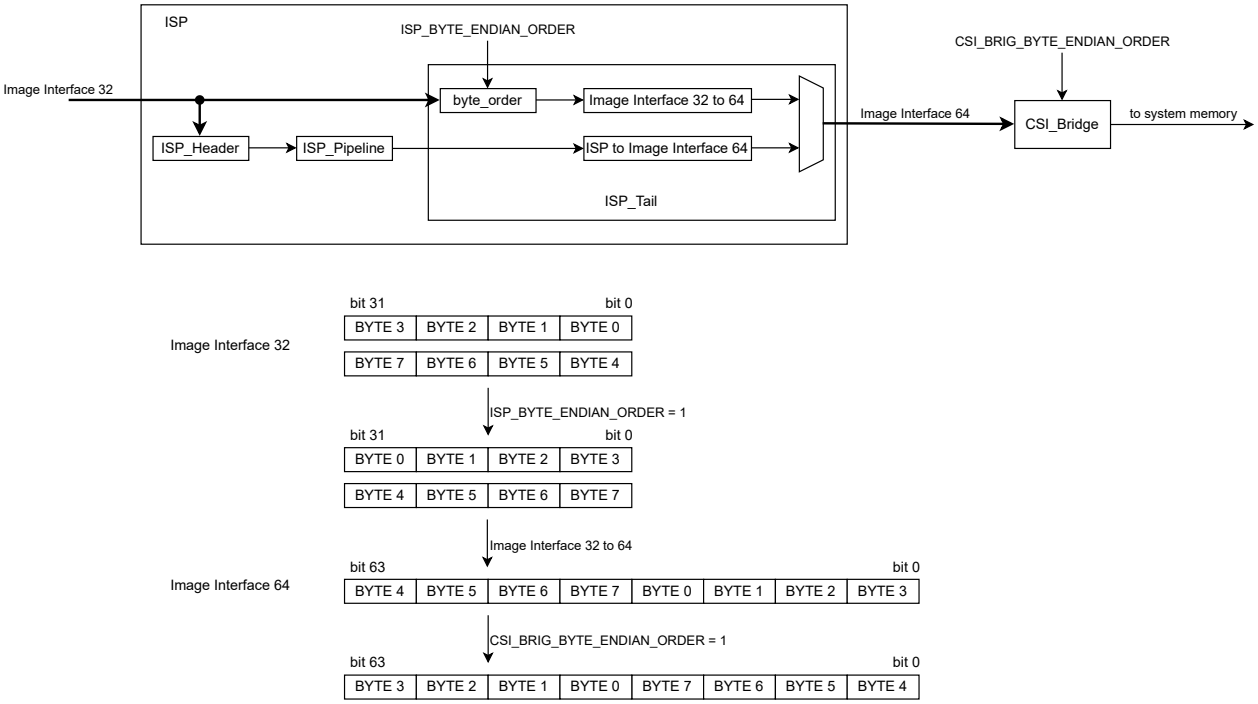


Figure 38.5-5. Byte Ordering

38.6 Interrupts

ESP32-P4's ISP can generate the following interrupt signals that will be sent to the Interrupt Matrix.

- isp_interrupt
- csi_bridge_interrupt

There are several internal interrupt sources from ISP that can generate the above interrupt signal. The interrupt sources from ISP are listed with their trigger conditions and the resulted interrupt signal in Table 38.6-1.

Table 38.6-1. ISP's Internal Interrupt Sources

Internal Interrupt Source	Trigger Condition	Interrupt Signal

ISP_CROP_ERR_INT	When the crop module window is configured incorrectly	isp_interrupt
ISP_WBG_FRAME_INT	After module WBG processes a frame	isp_interrupt
ISP_CROP_FRAME_INT	After module crop processes a frame	isp_interrupt
ISP_HEADER_IDI_FRAME_INT	After MIPI-CSI inputs a frame. This interrupt is not affected by whether ISP_MIPI_DATA_EN is enabled	isp_interrupt
ISP_TAIL_IDI_FRAME_INT	After ISP_Tail outputs a frame	isp_interrupt
ISP_YUV2RGB_FRAME_INT	After module YUV2RGB processes a frame	isp_interrupt
ISP_COLOR_FRAME_INT	After module contrast/hue/saturation/luminance adjustment processes a frame	isp_interrupt
ISP_SHARP_FRAME_INT	After module sharpen processes a frame	isp_interrupt
ISP_RGB2YUV_FRAME_INT	After module RGB2YUV processes a frame	isp_interrupt
ISP_GAMMA_FRAME_INT	After module gamma correction processes a frame	isp_interrupt
ISP_CCM_FRAME_INT	After module CCM processes a frame	isp_interrupt
ISP_DEMOSAIC_FRAME_INT	After module demosaic processes a frame	isp_interrupt
ISP_BF_FRAME_INT	After module BF processes a frame	isp_interrupt
ISP_DPC_FRAME_INT	After module DPC processes a frame	isp_interrupt
ISP_LSC_FRAME_INT	After module LSC processes a frame	isp_interrupt
ISP_BLC_FRAME_INT	After module BLC processes a frame	isp_interrupt
ISP_FRAME_INT	After ISP_Pipeline outputs a frame	isp_interrupt
ISP_HIST_FDONE_INT	After module HIST processes a frame	isp_interrupt
ISP_AWB_FDONE_INT	After module AWB processes a frame	isp_interrupt
ISP_AF_ENV_INT	When module AF detects a change in the focus scene	isp_interrupt
ISP_AF_FDONE_INT	After module AF processes a frame	isp_interrupt
ISP_AE_FRAME_DONE_INT	After module AE processes a frame	isp_interrupt
ISP_AE_MONITOR_INT	When module AE detects a change in the luminance	isp_interrupt
ISP_GAMMA_XCOORD_ERR_INT	When the GAMMA parameter is set incorrectly, the X length in ISP_GAMMA_R/G/B_X00-0F is not 256	isp_interrupt
ISP_DPC_CHECK_DONE_INT	After DPC static calibration processes a frame	isp_interrupt
ISP_MIPI_HNUM_UNMATCH_INT	When the width HNUM settings do not match the actual input from MIPI-CSI	isp_interrupt
ISP_DATA_TYPE_SETTING_ERR_INT	When DATA_TYPE settings do not match the actual input data	isp_interrupt

ISP_HVNUM_SETTING_ERR_INT	When the width HNUM and the height VNUM settings are incorrect: Image dimensions must be even; with the RAW10 input, HNUM must be a multiple of 4	isp_interrupt
ISP_BUF_FULL_INT	When ISP_Header buffer is full	isp_interrupt
ISP_ASYNC_FIFO_OVF_INT	When asynchronous FIFO of ISP_Header is overflowed	isp_interrupt
ISP_DATA_TYPE_ERR_INT	When illegal DATA_TYPE is used for input data (not RAW8/RAW10/RAW12)	isp_interrupt
CSI_BRIG_DMA_CFG_HAS_UPDATED_INT	When VDMA configuration of CSI_Bridge is updated	csi_bridge_interrupt
CSI_BRIG_CSI_ASYNC_FIFO_OVF_INT	When asynchronous FIFO of CSI_Bridge is overflowed	csi_bridge_interrupt
CSI_BRIG_CSI_BUF_OVERRUN_INT	When CSI_Bridge buffer is overflowed	csi_bridge_interrupt
CSI_BRIG_DISCARD_INT	When any frame is discarded in CSI_Bridge	csi_bridge_interrupt
CSI_BRIG_VADR_NUM_LT_INT	When CSI_BRIG_VADR_NUM is less than the actual value	csi_bridge_interrupt
CSI_BRIG_VADR_NUM_GT_INT	When CSI_BRIG_VADR_NUM is greater than the actual value	csi_bridge_interrupt

Note:
For definitions of *interrupt*, *interrupt signal*, *interrupt source*, and their correlations, please refer to Chapter 13 *Interrupt Matrix* > Section 13.2 *Interrupt Terminology in ESP32-P4*.

Each interrupt source can be configured by a common set of registers that are described in Section *Interrupt Configuration Registers*. The specific registers can be found in Section 38.8 *Register Summary*.

38.7 Programming Procedures

38.7.1 ISP Clock Reset Configuration

1. Enable the bus clock for the ISP and CSI_Bridge modules;
   - Set HP_SYS_CLKRST_CSI_BRG_SYS_CLK_EN to 1 to activate the clock for ISP and CSI_Bridge
2. Configure the working clock for ISP;
   - Choose the clock source via HP_SYS_CLKRST_ISP_CLK_SRC_SEL
   - Configure the clock divisor via HP_SYS_CLKRST_ISP_CLK_DIV_NUM
   - Set HP_SYS_CLKRST_ISP_CLK_EN to 1 to activate the ISP working clock
3. Reset ISP and CSI_Bridge modules.

- Set HP_SYS_CLKRST_RST_EN_CSI_BRG to 1 and then 0 to reset ISP and CSI_Bridge

38.7.2 CSI_Bridge Configuration

1. Set the width and height of the image via CSI_BRIG_HADR_NUM and CSI_BRIG_VADR_NUM. The width is measured in "64-bit" units, and the height is measured in rows;
2. Set the valid range for DATA_TYPE via CSI_BRIG_DATA_TYPE_MIN and CSI_BRIG_DATA_TYPE_MAX;
3. Specify the burst length via CSI_BRIG_DMA_BURST_LEN. Ensure that this value matches the SRC_MSIZE (length per burst) and SRC_TR_WIDTH (data width) of the corresponding VDMA channel. The data width for CSI_Bridge is measured in "64-bit" units;
4. If the image is transferred in multiple blocks, specify the maximum number of VDMA bursts per block via CSI_BRIG_DMABLK_SIZE;
5. Enable CSI_Bridge by writing 1 to CSI_BRIG_CSI_BRIG_EN.

38.7.3 VDMA Configuration

1. Configure the register corresponding to the channel;
   - Set SRC/DST_MULTBLK_TYPE to 3 (configure with linked list)
   - Configure TT_FC
     - For CSI_Bridge, set TT_FC to 4 (peripheral to memory with peripheral flow control)
     - For memory to ISP, set TT_FC to 6 (memory to peripheral with peripheral flow control)
   - Configure HS_SEL
     - For CSI_Bridge, set HS_SEL_SRC to 0 (peripheral uses hardware handshake), and set HS_SEL_DST to 1 (memory uses software handshake)
     - For memory to ISP, set HS_SEL_SRC to 1 (memory uses software handshake), and set HS_SEL_DST to 0 (peripheral uses hardware handshake)
   - Assign hardware handshake channels
     - For CSI_Bridge, set SRC_PER to 1
     - For memory to ISP, set DST_PER to 2
   - Set SRC/DST_TR_WIDTH to 3 (data width of 64 bits)
2. Establish the linked list based on the configured registers.

Note:
The VDMA configuration outlined in this section covers essential steps. For detailed procedures, refer to Chapter 7 *VDMA Controller (VDMA)*.
Ensure that image dimensions and start addresses are aligned to 8 bytes.

38.7.4 ISP General Configuration

1. Configure the input image size via ISP_HADR_NUM and ISP_VADR_NUM
2. Set image arrangement mode in the Bayer domain via ISP_BAYER_MODE;
3. Specify the ISP output data format via ISP_OUT_TYPE;
4. Configure each module in ISP_Pipeline. Refer to the corresponding section in 38.5.2 for details;
5. Enable the necessary Pipeline modules by setting the fields in register ISP_CNTL_REG. Ensure that the format conversion-related modules (Demosaic, RGB2YUV, YUV2RGB) are enabled before enabling the ISP, and that they match the final output format.

38.7.5 Enabling ISP for Image Capture from MIPI-CSI

1. Configure ISP and CSI_Bridge clocks and reset as described in Section 38.7.1;
2. Configure CSI_Bridge as described in Section 38.7.2;
3. Configure VDMA as described in Section 38.7.3;
4. Configure MIPI CSI HOST according to Chapter 42 *MIPI CSI*;
5. Configure ISP general settings as described in Section 38.7.4;
6. Select the input source as MIPI-CSI by setting ISP_IN_SRC to 0;
7. Configure the input data type via ISP_DATA_TYPE;
8. Enable the ISP by setting ISP_EN to 1;
9. Enable MIPI-CSI data input by setting ISP_MIPI_DATA_EN to 1.

38.7.6 Enabling ISP for Image Capture from DVP

1. Configure ISP and CSI_Bridge clocks and reset as described in Section 38.7.1;
2. Configure CSI_Bridge as described in Section 38.7.2;
3. Configure VDMA as described in Section 38.7.3;
4. Configure ISP general settings as described in Section 38.7.4;
5. Select the input source as DVP by setting ISP_IN_SRC to 1;
6. Configure the input data type via ISP_CAM_DATA_TYPE, and write 0 to ISP_DATA_TYPE;
7. Enable the ISP by setting ISP_EN to 1;
8. Configure the data mode for DVP interface via ISP_CAM_CONF_REG;
9. Reset the DVP interface by setting ISP_CAM_RESET to 1, waiting for a period, then setting it to 0;
10. Write 1 to ISP_CAM_UPDATE_REG;
11. Enable the DVP interface by setting ISP_CAM_EN to 1;
12. Wait for ISP_CAM_UPDATE_REG to automatically clear for initialization.

38.7.7 Enabling ISP for Image Capture from VDMA

1. Configure ISP and CSI_Bridge clocks and reset as described in Section 38.7.1;
2. Configure CSI_Bridge as described in Section 38.7.2;
3. Configure VDMA as described in Section 38.7.3;
4. Configure ISP general settings as described in Section 38.7.4;
5. Select the input source as VDMA by setting ISP_IN_SRC to 2;
6. Configure the input data type via ISP_DMA_DATA_TYPE and ISP_DATA_TYPE;
7. Configure the total amount of raw image data for one frame in "64-bit" units using ISP_DMA_RAW_NUM_TOTAL, which value should equal image width x image height x bits per pixel/64. Then, write 1 to ISP_DMA_RAW_NUM_TOTAL_SET to activate the configuration;
8. Configure the burst length via ISP_DMA_BURST_LEN, which needs to align with the VDMA configuration;
9. Write 1 to ISP_DMA_UPDATE_REG to update the VDMA configuration, and wait for the update to complete;
10. Send a frame of data by setting ISP_DMA_EN to 1.

38.7.8 LSC LUT Configuration

LSC can be configured after the ISP is enabled. Before enabling LSC, initialize the LSC LUT as follows:

1. Write to LUT;
   - Write the data to be stored in the LUT by writing to ISP_LUT_WDATA_REG
   - Store the data at the specified address by writing to ISP_LUT_CMD_REG
2. Read from LUT.
   - Retrieve the data stored at the specified address by writing to ISP_LUT_CMD_REG
   - Obtain the data read from the LUT by reading from ISP_LUT_RDATA_REG

38.7.9 AE Configuration

AE can be configured after the ISP is enabled as follows:

1. Select sampling points via ISP_AE_SELECT;
2. Set up statistical windows;
   - Set the horizontal starting point with ISP_AE_X_START
   - Configure the horizontal dimension of each sub-window with ISP_AE_X_BSIZE
   - Set the vertical starting point with ISP_AE_Y_START
   - Configure the vertical dimension of each sub-window with ISP_AE_Y_BSIZE
   - Specify the number of pixels in each sub-window with ISP_AE_SUBWIN_PIXNUM
   - Set the reciprocal of the number of pixels in each sub-window with ISP_AE_SUBWIN_RECIP (20 fractional bits)

3. Enable luminance monitoring;
    - If monitoring is required, set ISP_AE_MONITOR_TH and ISP_AE_MONITOR_TL to non-zero values
    - Specify the monitoring statistical period in frames via ISP_AE_MONITOR_PERIOD
4. Enable AE by writing 1 to ISP_AE_EN;
5. Obtain statistics;
    - Clear the completion status by writing 1 to ISP_AE_FRAME_DONE_INT_CLR
    - Initiate statistics by writing 1 to ISP_AE_UPDATE. Each write to this register resets the frame count for monitoring. It is recommended to disable monitoring before manual statistics and re-enable it after completion to prevent automatic results from overwriting manual results prematurely
    - Wait for the statistics to complete
        - Wait for the ISP interrupt triggered by AE_FDONE_INT
        - Or query the status using ISP_AE_FRAME_DONE_INT_RAW
    - Obtain statistics for each sub-window by reading ISP_AE_Bxx_MEAN
6. Perform luminance monitoring.
    - Wait for the ISP interrupt triggered by AE_MONITOR_INT
    - Or query the status using ISP_AE_MONITOR_INT_RAW
    - After detecting a change in luminance, disable luminance monitoring, re-initiate camera exposure, and then re-enable monitoring

38.7.10 AF Configuration

AF can be configured after the ISP is enabled as follows:

1. Set up three statistical windows A, B, and C, ensuring each window does not exceed 1024 x 1024;
    - Set the left boundary using ISP_AF_LPOINT_A/B/C
    - Set the right boundary using ISP_AF_RPOINT_A/B/C
    - Set the top boundary using ISP_AF_TPOINT_A/B/C
    - Set the bottom boundary using ISP_AF_BPOINT_A/B/C
2. Set the sharpness threshold via ISP_AF_THRESHOLD;
3. Configure focus scene monitoring;
    - If focus scene monitoring is required, set the monitoring period ISP_AF_ENV_PERIOD to a non-zero value
    - There are two ways to set the threshold
        - If ISP_AF_ENV_USER_THRESHOLD_SUM or ISP_AF_ENV_USER_THRESHOLD_LUM is non-zero, use the specified values as thresholds for sharpness and brightness changes

  - If ISP_AF_ENV_USER_THRESHOLD_SUM or ISP_AF_ENV_USER_THRESHOLD_LUM is zero, use the sharpness and brightness of the most recent statistics multiplied by ISP_AF_ENV_THRESHOLD (4 fractional bits) as the change threshold

4. Configure the statistical trigger mode. After each trigger completes, update the monitoring baseline;
   - If ISP_AF_AUTO_UPDATE is 1, it is in automatic triggering mode, which performs statistics for every frame with AF enabled. Since each statistical calculation clears the frame count for monitoring, the monitoring will be disabled
   - If ISP_AF_AUTO_UPDATE is 0, it is in manual triggering mode. The monitoring is working only under this condition
5. Enable AF by writing 1 to ISP_AF_EN;
6. Obtain statistics;
   - Clear the completion status by writing 1 to ISP_AF_FDONE_INT_CLR
   - Trigger statistics
     - With automatic triggering, statistics are performed for every frame
     - With manual triggering, write 1 to ISP_AF_MANUAL_UPDATE to trigger statistics manually
   - Wait for the statistics to complete
     - Wait for the ISP interrupt triggered by AF_FDONE_INT.
     - Or query the status using ISP_AF_FDONE_INT_RAW
   - Obtain sharpness statistics by reading ISP_AF_SUMA/B/C and obtain brightness statistics by reading ISP_AF_LUMA/B/C
7. Performs focus scene monitoring.
   - Wait for the ISP interrupt triggered by AF_ENV_INT
   - Or query the status using ISP_AF_ENV_INT_RAW
   - After detecting a change in the focus scene, disable monitoring, refocus the camera, and then re-enable monitoring

38.7.11 AWB Configuration

AWB can be configured after the ISP is enabled.

1. Select sampling points via ISP_AWB_SAMPLE and write 1 to ISP_AWB_MODE;
2. Set up statistical windows;
   - Set the left boundary using ISP_AWB_LPOINT
   - Set the right boundary using ISP_AWB_RPOINT
   - Set the top boundary using ISP_AWB_TPOINT
   - Set the bottom boundary using ISP_AWB_BPOINT
3. Configure white patch constraints;

- Set maximum luminance with ISP_AWB_MAX_LUM
- Set minimum luminance with ISP_AWB_MIN_LUM
- Set maximum R/G ratio with ISP_AWB_MAX_RG (2 integer bits, 8 fractional bits)
- Set minimum R/G ratio with ISP_AWB_MIN_RG (2 integer bits, 8 fractional bits)
- Set maximum B/G ratio with ISP_AWB_MAX_BG (2 integer bits, 8 fractional bits)
- Set minimum B/G ratio with ISP_AWB_MIN_BG (2 integer bits, 8 fractional bits)

4. Obtain statistics;
   - Clear the completion status by writing 1 to ISP_AWB_FDONE_INT_CLR
   - Trigger statistics by writing 0 and then 1 to ISP_AWB_EN
   - Wait for the statistics to complete
     - Wait for the ISP interrupt triggered by AWB_FDONE_INT
     - Or query the status using ISP_AWB_FDONE_INT_RAW
   - Get the number of white patches within the statistical window by reading ISP_AWBO_WHITE_CNT
   - Get the accumulated values of the R, G, B components for the white patches within the statistical window by reading ISP_AWBO_ACC_R/G/B

38.7.12 HIST Configuration

HIST can be configured after the ISP is enabled.

1. Select sampling points via ISP_HIST_MODE;
   - If sampling points are set to RGB, use ISP_HIST_COEFF_R/G/B to set RGB weights with 8 fractional bits
2. Set up statistical windows;
   - Set the horizontal start point with ISP_HIST_X_OFFS
   - Configure the horizontal dimension of each sub-window with ISP_HIST_X_SIZE
   - Set the vertical start point with ISP_HIST_Y_OFFS
   - Configure the vertical dimension of each sub-window with ISP_HIST_Y_SIZE
3. Configure histograms;
   - Set histogram interval divisions using ISP_HIST_SEG_x_x
   - Set window weights using ISP_HIST_WEIGHT_xx with 8 fractional bits
4. Enable HIST by writing 1 to ISP_HIST_EN;
5. Obtain statistics.
   - Clear the completion status by writing 1 to ISP_HIST_FDONE_INT_CLR
   - With HIST enabled, statistics are performed at the end of every frame. Wait for the statistics to complete

  - Wait for the ISP interrupt triggered by HIST_FDONE_INT
  - Or query the status using ISP_HIST_FDONE_INT_RAW.
- Obtain the statistics for each interval by reading ISP_HIST_BIN_x.

38.7.13 ISP_Pipeline Module Update Configuration

1. Disable the module via ISP_CNTL_REG;
2. Wait for the interrupt from the corresponding module;
3. Update module configurations;
4. Re-enable the module.

38.8 Register Summary

38.8.1 ISP Register Summary

The addresses in this section are relative to **ISP** base address provided in Table 9.3-2 in Chapter 9 *System and Memory*.

The abbreviations given in Column **Access** are explained in Section *Access Types for Registers*.

Name	Description	Address	Access
Configuration Registers			
ISP_CLK_EN_REG	ISP clock control register	0x0004	R/W
ISP_CNTL_REG	ISP enable register	0x0008	R/W
ISP_HSYNC_CNT_REG	Hsync interval control register	0x000C	R/W
ISP_FRAME_CFG_REG	Image control register	0x0010	R/W
ISP_CCM_COEF0_REG	CCM parameter register 0	0x0014	R/W
ISP_CCM_COEF1_REG	CCM parameter register 1	0x0018	R/W
ISP_CCM_COEF3_REG	CCM parameter register 3	0x001C	R/W
ISP_CCM_COEF4_REG	CCM parameter register 4	0x0020	R/W
ISP_CCM_COEF5_REG	CCM parameter register 5	0x0024	R/W
ISP_BF_MATRIX_CTRL_REG	BF pixel-to-matrix control register	0x0028	R/W
ISP_BF_SIGMA_REG	BF noise reduction strength control register	0x002C	R/W
ISP_BF_GAU0_REG	BF noise reduction template register 0	0x0030	R/W
ISP_BF_GAU1_REG	BF noise reduction template register 1	0x0034	R/W
ISP_DPC_CTRL_REG	DPC control register	0x0038	R/W
ISP_DPC_CONF_REG	DPC parameter register	0x003C	R/W
ISP_DPC_MATRIX_CTRL_REG	DPC pixel-to-matrix control register	0x0040	R/W
ISP_LUT_CMD_REG	LUT command register	0x0048	WT
ISP_LUT_WDATA_REG	LUT write register	0x004C	R/W
ISP_LSC_TABLESIZE_REG	LSC parameter control register	0x0054	R/W
ISP_DEMOSAIC_MATRIX_CTRL_REG	Demosaic pixel-to-matrix control register	0x0058	R/W
ISP_DEMOSAIC_GRAD_RATIO_REG	Demosaic parameter control register	0x005C	R/W
ISP_GAMMA_CTRL_REG	Gamma correction control register	0x0074	R/W
ISP_GAMMA_RY1_REG	Gamma curve R channel Y axis configuration register 1	0x0078	R/W
ISP_GAMMA_RY2_REG	Gamma curve R channel Y axis configuration register 2	0x007C	R/W
ISP_GAMMA_RY3_REG	Gamma curve R channel Y axis configuration register 3	0x0080	R/W
ISP_GAMMA_RY4_REG	Gamma curve R channel Y axis configuration register 4	0x0084	R/W
ISP_GAMMA_GY1_REG	Gamma curve G channel Y axis configuration register 1	0x0088	R/W
ISP_GAMMA_GY2_REG	Gamma curve G channel Y axis configuration register 2	0x008C	R/W

Name	Description	Address	Access
ISP_GAMMA_GY3_REG	Gamma curve G channel Y axis configuration register 3	0x0090	R/W
ISP_GAMMA_GY4_REG	Gamma curve G channel Y axis configuration register 4	0x0094	R/W
ISP_GAMMA_BY1_REG	Gamma curve B channel Y axis configuration register 1	0x0098	R/W
ISP_GAMMA_BY2_REG	Gamma curve B channel Y axis configuration register 2	0x009C	R/W
ISP_GAMMA_BY3_REG	Gamma curve B channel Y axis configuration register 3	0x00A0	R/W
ISP_GAMMA_BY4_REG	Gamma curve B channel Y axis configuration register 4	0x00A4	R/W
ISP_GAMMA_RX1_REG	Gamma curve R channel X axis configuration register 1	0x00A8	R/W
ISP_GAMMA_RX2_REG	Gamma curve R channel X axis configuration register 2	0x00AC	R/W
ISP_GAMMA_GX1_REG	Gamma curve G channel X axis configuration register 1	0x00B0	R/W
ISP_GAMMA_GX2_REG	Gamma curve G channel X axis configuration register 2	0x00B4	R/W
ISP_GAMMA_BX1_REG	Gamma curve B channel X axis configuration register 1	0x00B8	R/W
ISP_GAMMA_BX2_REG	Gamma curve B channel X axis configuration register 2	0x00BC	R/W
ISP_AE_CTRL_REG	AE control register	0x00C0	varies
ISP_AE_MONITOR_REG	AE monitoring control register	0x00C4	R/W
ISP_AE_BX_REG	AE statistical window X direction configuration register	0x00C8	R/W
ISP_AE_BY_REG	AE statistical window Y direction configuration register	0x00CC	R/W
ISP_AE_WINPIXNUM_REG	AE sub-window pixel number configuration register	0x00D0	R/W
ISP_AE_WIN_RECIPROCAL_REG	AE sub-window pixel number reciprocal configuration register	0x00D4	R/W
ISP_SHARP_CTRL0_REG	Sharpen control register 0	0x00F4	R/W
ISP_SHARP_FILTER0_REG	Sharpen low-frequency template configuration register 0	0x00F8	R/W
ISP_SHARP_FILTER1_REG	Sharpen low-frequency template configuration register 1	0x00FC	R/W
ISP_SHARP_FILTER2_REG	Sharpen low-frequency template configuration register 2	0x0100	R/W
ISP_SHARP_MATRIX_CTRL_REG	Sharpen pixel-to-matrix control register	0x0104	R/W
ISP_SHARP_CTRL1_REG	Sharpen control register 1	0x0108	RO
ISP_DMA_CNTL_REG	ISP VDMA input control register	0x010C	varies

Name	Description	Address	Access
ISP_DMA_RAW_DATA_REG	ISP VDMA input data amount configuration register	0x0110	varies
ISP_CAM_CNTL_REG	ISP DVP input control register	0x0114	R/W
ISP_CAM_CONF_REG	ISP DVP input configuration register	0x0118	R/W
ISP_AF_CTRL0_REG	AF control register 0	0x011C	varies
ISP_AF_ENV_USER_TH_SUM_REG	AF scene monitoring sharpness threshold configuration register	0x0128	R/W
ISP_AF_ENV_USER_TH_LUM_REG	AF scene monitoring luminance threshold configuration register	0x012C	R/W
ISP_AF_THRESHOLD_REG	AF statistics threshold configuration register	0x0130	varies
ISP_AF_HSCALE_A_REG	AF statistical window A horizontal direction configuration register	0x0134	R/W
ISP_AF_VSCALE_A_REG	AF statistical window A vertical direction configuration register	0x0138	R/W
ISP_AF_HSCALE_B_REG	AF statistical window B horizontal direction configuration register	0x013C	R/W
ISP_AF_VSCALE_B_REG	AF statistical window B vertical direction configuration register	0x0140	R/W
ISP_AF_HSCALE_C_REG	AF statistical window C horizontal direction configuration register	0x0144	R/W
ISP_AF_VSCALE_C_REG	AF statistical window C vertical direction configuration register	0x0148	R/W
ISP_AWB_MODE_REG	AWB control register	0x0164	R/W
ISP_AWB_HSCALE_REG	AWB statistical window horizontal direction configuration register	0x0168	R/W
ISP_AWB_VSCALE_REG	AWB statistical window vertical direction configuration register	0x016C	R/W
ISP_AWB_TH_LUM_REG	AWB white patch luminance threshold configuration register	0x0170	R/W
ISP_AWB_TH_RG_REG	AWB white patch R/G threshold configuration register	0x0174	R/W
ISP_AWB_TH_BG_REG	AWB white patch B/G threshold configuration register	0x0178	R/W
ISP_COLOR_CTRL_REG	Contrast/saturation/luminance control register	0x018C	R/W
ISP_BLC_VALUE_REG	BLC control register	0x0190	R/W
ISP_BLC_CTRL0_REG	BLC pixel stretch register	0x0194	R/W
ISP_HIST_MODE_REG	HIST control register	0x01A4	R/W
ISP_HIST_COEFF_REG	HIST RGB parameter configuration register	0x01A8	R/W
ISP_HIST_OFFS_REG	HIST statistical window starting point configuration register	0x01AC	R/W
ISP_HIST_SIZE_REG	HIST statistical sub-window size configuration register	0x01B0	R/W
ISP_HIST_SEG0_REG	HIST X coordinate interval configuration register 0	0x01B4	R/W

Name	Description	Address	Access
ISP_HIST_SEG1_REG	HIST X coordinate interval configuration register 1	0x01B8	R/W
ISP_HIST_SEG2_REG	HIST X coordinate interval configuration register 2	0x01BC	R/W
ISP_HIST_SEG3_REG	HIST X coordinate interval configuration register 3	0x01C0	R/W
ISP_HIST_WEIGHT0_REG	HIST sub-window weight configuration register 0	0x01C4	R/W
ISP_HIST_WEIGHT1_REG	HIST sub-window weight configuration register 1	0x01C8	R/W
ISP_HIST_WEIGHT2_REG	HIST sub-window weight configuration register 2	0x01CC	R/W
ISP_HIST_WEIGHT3_REG	HIST sub-window weight configuration register 3	0x01D0	R/W
ISP_HIST_WEIGHT4_REG	HIST sub-window weight configuration register 4	0x01D4	R/W
ISP_HIST_WEIGHT5_REG	HIST sub-window weight configuration register 5	0x01D8	R/W
ISP_HIST_WEIGHT6_REG	HIST sub-window weight configuration register 6	0x01DC	R/W
ISP_YUV_FORMAT_REG	YUV format control register	0x0234	R/W
ISP_CROP_CTRL_REG	CROP control register	0x0244	WT
ISP_CROP_Y_CAPTURE_REG	CROP Y direction control register	0x0248	R/W
ISP_CROP_X_CAPTURE_REG	CROP X direction control register	0x024C	R/W
ISP_CROP_ERR_ST_REG	CROP error status register	0x0250	RO
ISP_WBG_COEF_R_REG	WBG R channel gain register	0x0254	R/W
ISP_WBG_COEF_G_REG	WBG G channel gain register	0x0258	R/W
ISP_WBG_COEF_B_REG	WBG B channel gain register	0x025C	R/W
ISP_COLOR_HUE_CTRL_REG	Hue control register	0x0260	R/W
ISP_AWB_BX_REG	AWB sub-window X direction control register	0x0264	R/W
ISP_AWB_BY_REG	AWB sub-window Y direction control register	0x0268	R/W
ISP_STATE_REG	ISP state register	0x026C	RO
ISP_SHADOW_REG_CTRL_REG	Shadow register control register	0x0270	R/W
Status Registers			
ISP_DPC_DEADPIX_CNT_REG	DPC static calibration defective pixel number register	0x0044	RO
ISP_LUT_RDATA_REG	LUT read register	0x0050	RO
ISP_AE_BLOCK_MEAN_0_REG	AE statistics register 0	0x00D8	RO
ISP_AE_BLOCK_MEAN_1_REG	AE statistics register 1	0x00DC	RO
ISP_AE_BLOCK_MEAN_2_REG	AE statistics register 2	0x00E0	RO
ISP_AE_BLOCK_MEAN_3_REG	AE statistics register 3	0x00E4	RO
ISP_AE_BLOCK_MEAN_4_REG	AE statistics register 4	0x00E8	RO
ISP_AE_BLOCK_MEAN_5_REG	AE statistics register 5	0x00EC	RO
ISP_AE_BLOCK_MEAN_6_REG	AE statistics register 6	0x00F0	RO

Name	Description	Address	Access
ISP_AF_SUM_A_REG	AF window A sharpness statistics register	0x014C	RO
ISP_AF_SUM_B_REG	AF window B sharpness statistics register	0x0150	RO
ISP_AF_SUM_C_REG	AF window C sharpness statistics register	0x0154	RO
ISP_AF_LUM_A_REG	AF window A luminance statistics register	0x0158	RO
ISP_AF_LUM_B_REG	AF window B luminance statistics register	0x015C	RO
ISP_AF_LUM_C_REG	AF window C luminance statistics register	0x0160	RO
ISP_AWB0_WHITE_CNT_REG	AWB white patch count statistics register	0x017C	RO
ISP_AWB0_ACC_R_REG	AWB R channel statistics register	0x0180	RO
ISP_AWB0_ACC_G_REG	AWB G channel statistics register	0x0184	RO
ISP_AWB0_ACC_B_REG	AWB B channel statistics register	0x0188	RO
ISP_HIST_BIN0_REG	HIST interval 0 statistics register	0x01E0	RO
ISP_HIST_BIN1_REG	HIST interval 1 statistics register	0x01E4	RO
ISP_HIST_BIN2_REG	HIST interval 2 statistics register	0x01E8	RO
ISP_HIST_BIN3_REG	HIST interval 3 statistics register	0x01EC	RO
ISP_HIST_BIN4_REG	HIST interval 4 statistics register	0x01F0	RO
ISP_HIST_BIN5_REG	HIST interval 5 statistics register	0x01F4	RO
ISP_HIST_BIN6_REG	HIST interval 6 statistics register	0x01F8	RO
ISP_HIST_BIN7_REG	HIST interval 7 statistics register	0x01FC	RO
ISP_HIST_BIN8_REG	HIST interval 8 statistics register	0x0200	RO
ISP_HIST_BIN9_REG	HIST interval 9 statistics register	0x0204	RO
ISP_HIST_BIN10_REG	HIST interval 10 statistics register	0x0208	RO
ISP_HIST_BIN11_REG	HIST interval 11 statistics register	0x020C	RO
ISP_HIST_BIN12_REG	HIST interval 12 statistics register	0x0210	RO
ISP_HIST_BIN13_REG	HIST interval 13 statistics register	0x0214	RO
ISP_HIST_BIN14_REG	HIST interval 14 statistics register	0x0218	RO
ISP_HIST_BIN15_REG	HIST interval 15 statistics register	0x021C	RO
Interrupt Registers			
ISP_INT_RAW_REG	Raw interrupt status register	0x0064	R/SS/WTC
ISP_INT_ST_REG	Masked interrupt status register	0x0068	RO
ISP_INT_ENA_REG	Interrupt enable register	0x006C	R/W
ISP_INT_CLR_REG	Interrupt clear register	0x0070	WT
Version Register			
ISP_VER_DATE_REG	Version control register	0x0000	R/W

38.8.2 MIPI CSI_Bridge Register Summary

The addresses in this section are relative to **MIPI CSI_Bridge** base address provided in Table 9.3-2 in Chapter 9 *System and Memory*.

The abbreviations given in Column **Access** are explained in Section *Access Types for Registers*.

Name	Description	Address	Access
Clock Gating Control Register			

Name	Description	Address	Access
CSI_BRIG_CLK_EN_REG	CSI_Bridge clock gate register	0x0000	R/W
CSI_Bridge Control Registers			
CSI_BRIG_CSI_EN_REG	CSI_Bridge enable register	0x0004	R/W
CSI_BRIG_BUF_FLOW_CTL_REG	CSI_Bridge buffer control register	0x000C	varies
DMA Control Registers			
CSI_BRIG_DMA_REQ_CFG_REG	DMA request control register	0x0008	R/W
CSI_BRIG_DMA_REQ_INTERVAL_REG	DMA request interval register	0x002C	R/W
CSI_BRIG_DMABLK_SIZE_REG	DMA data block size control register	0x0030	R/W
Frame Format Configuration Registers			
CSI_BRIG_DATA_TYPE_CFG_REG	DATA_TYPE control register	0x0010	R/W
CSI_BRIG_FRAME_CFG_REG	Image control register	0x0014	R/W
CSI_BRIG_ENDIAN_MODE_REG	Byte order control register	0x0018	R/W
Interrupt Registers			
CSI_BRIG_INT_RAW_REG	CSI_Bridge raw interrupt status register	0x001C	R/ WTC/ SS
CSI_BRIG_INT_CLR_REG	CSI_Bridge interrupt clear register	0x0020	WT
CSI_BRIG_INT_ST_REG	CSI_Bridge masked interrupt status register	0x0024	RO
CSI_BRIG_INT_ENA_REG	CSI_Bridge interrupt enable register	0x0028	R/W
CSI HOST Control Register			
CSI_BRIG_HOST_CTRL_REG	CSI HOST control register	0x0040	R/W

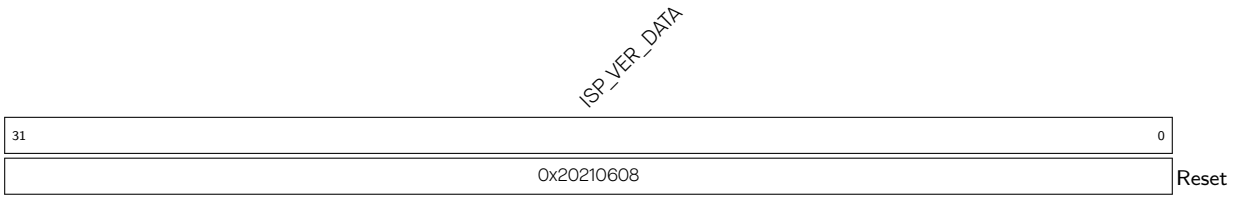
38.9 Registers

38.9.1 ISP Registers

The addresses in this section are relative to **ISP** base address provided in Table 9.3-2 in Chapter 9 *System and Memory*.

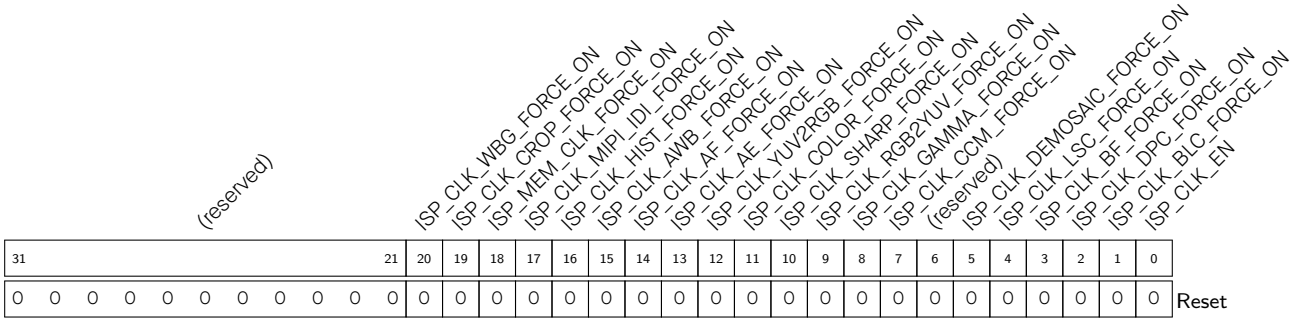
For how to program reserved fields, please refer to Section Programming Reserved Register Field.

Register 38.1. ISP_VER_DATE_REG (0x0000)



ISP_VER_DATA ISP version control register. (R/W)

Register 38.2. ISP_CLK_EN_REG (0x0004)



ISP_CLK_EN Configures whether the ISP clock is always on.

0: The clock is enabled only during register access

1: The clock is always on

(R/W)

ISP_CLK_BLC_FORCE_ON Configures whether to force enable the BLC clock.

0: Not force enable

1: Force enable

(R/W)

ISP_CLK_DPC_FORCE_ON Configures whether to force enable the DPC clock.

0: Not force enable

1: Force enable

(R/W)

ISP_CLK_BF_FORCE_ON Configures whether to force enable the BF clock.

0: Not force enable

1: Force enable

(R/W)

ISP_CLK_LSC_FORCE_ON Configures whether to force enable the LSC clock.

0: Not force enable

1: Force enable

(R/W)

ISP_CLK_DEMOSAIC_FORCE_ON Configures whether to force enable the demosaic clock.

0: Not force enable

1: Force enable

(R/W)

ISP_CLK_CCM_FORCE_ON Configures whether to force enable the CCM clock.

0: Not force enable

1: Force enable

(R/W)

Continued on the next page...

Register 38.2. ISP_CLK_EN_REG (0x0004)

Continued from the previous page...

ISP_CLK_GAMMA_FORCE_ON Configures whether to force enable the gamma correction clock.
0: Not force enable
1: Force enable
(R/W)

ISP_CLK_RGB2YUV_FORCE_ON Configures whether to force enable the RGB2YUV clock.
0: Not force enable
1: Force enable
(R/W)

ISP_CLK_SHARP_FORCE_ON Configures whether to force enable the sharpen clock.
0: Not force enable
1: Force enable
(R/W)

ISP_CLK_COLOR_FORCE_ON Configures whether to force enable the contrast/hue/saturation/luminance clock.
0: Not force enable
1: Force enable
(R/W)

ISP_CLK_YUV2RGB_FORCE_ON Configures whether to force enable the YUV2RGB clock.
0: Not force enable
1: Force enable
(R/W)

ISP_CLK_AE_FORCE_ON Configures whether to force enable the AE clock.
0: Not force enable
1: Force enable
(R/W)

ISP_CLK_AF_FORCE_ON Configures whether to force enable the AF clock.
0: Not force enable
1: Force enable
(R/W)

ISP_CLK_AWB_FORCE_ON Configures whether to force enable the AWB clock.
0: Not force enable
1: Force enable
(R/W)

ISP_CLK_HIST_FORCE_ON Configures whether to force enable the HIST clock.
0: Not force enable
1: Force enable
(R/W)

Continued on the next page...

Register 38.2. ISP_CLK_EN_REG (0x0004)

Continued from the previous page...

ISP_CLK_MIPI_IDI_FORCE_ON Configures whether to force enable the MIPI Image Interface 32 input clock.

0: Not force enable

1: Force enable

(R/W)

ISP_MEM_CLK_FORCE_ON Configures whether to force enable all the ISP memory clocks.

0: Not force enable

1: Force enable

(R/W)

ISP_CLK_CROP_FORCE_ON Configures whether to force enable the CROP clock.

0: Not force enable

1: Force enable

(R/W)

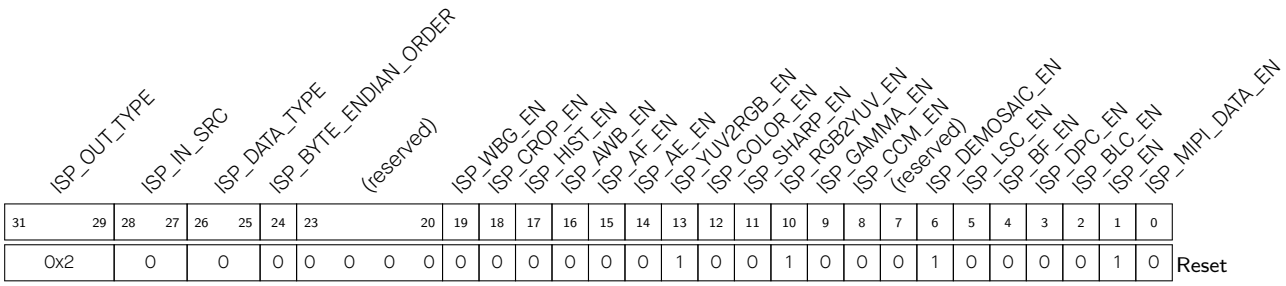
ISP_CLK_WBG_FORCE_ON Configures whether to force enable the WBG clock.

0: Not force enable

1: Force enable

(R/W)

Register 38.3. ISP_CNTL_REG (0x0008)



ISP_MIPI_DATA_EN Configures whether to enable the MIPI Image Interface 32 input.

0: Disable

1: Enable

(R/W)

ISP_EN Configures whether to enable the ISP.

0: Disable

1: Enable

(R/W)

ISP_BLC_EN Configures whether to enable the BLC module.

0: Disable

1: Enable

(R/W)

ISP_DPC_EN Configures whether to enable the DPC module.

0: Disable

1: Enable

(R/W)

ISP_BF_EN Configures whether to enable the BF module.

0: Disable

1: Enable

(R/W)

ISP_LSC_EN Configures whether to enable the LSC module.

0: Disable

1: Enable

(R/W)

ISP_DEMOSAIC_EN Configures whether to enable the demosaic module.

0: Disable

1: Enable

(R/W)

ISP_CCM_EN Configures whether to enable the CCM module.

0: Disable

1: Enable

(R/W)

Continued on the next page...

Register 38.3. ISP_CNTL_REG (0x0008)

Continued from the previous page...

ISP_GAMMA_EN Configures whether to enable the gamma correction module.

0: Disable

1: Enable

(R/W)

ISP_RGB2YUV_EN Configures whether to enable the RGB2YUV module.

0: Disable

1: Enable

(R/W)

ISP_SHARP_EN Configures whether to enable the sharpen module.

0: Disable

1: Enable

(R/W)

ISP_COLOR_EN Configures whether to enable the contrast/hue/saturation/luminance module.

0: Disable

1: Enable

(R/W)

ISP_YUV2RGB_EN Configures whether to enable the YUV2RGB module.

0: Disable

1: Enable

(R/W)

ISP_AE_EN Configures whether to enable the AE module.

0: Disable

1: Enable

(R/W)

ISP_AF_EN Configures whether to enable the AF module.

0: Disable

1: Enable

(R/W)

ISP_AWB_EN Configures whether to enable the AWB module.

0: Disable

1: Enable

(R/W)

ISP_HIST_EN Configures whether to enable the HIST module.

0: Disable

1: Enable

(R/W)

Continued on the next page...

Register 38.3. ISP_CNTL_REG (0x0008)

Continued from the previous page...

ISP_CROP_EN Configures whether to enable the CROP module.

0: Disable

1: Enable

(R/W)

ISP_WBG_EN Configures whether to enable the WBG module.

0: Disable

1: Enable

(R/W)

ISP_BYTE_ENDIAN_ORDER Configures the byte order for MIPI Image Interface 32 input when ISP is disabled.

0: csi_data[31:0]

1: [7:0], [15:8], [23:16], [31:24]

(R/W)

ISP_DATA_TYPE Configures the input data type.

0: RAW8

1: RAW10

2: RAW12

3: invalid

(R/W)

ISP_IN_SRC Configures the input data source.

0: MIPI Image Interface 32 data from CSI HOST

1: Data from DVP

2: Data from VDMA

3: invalid

(R/W)

ISP_OUT_TYPE Configures the output data type.

0: RAW8

1: YUV422

2: RGB888

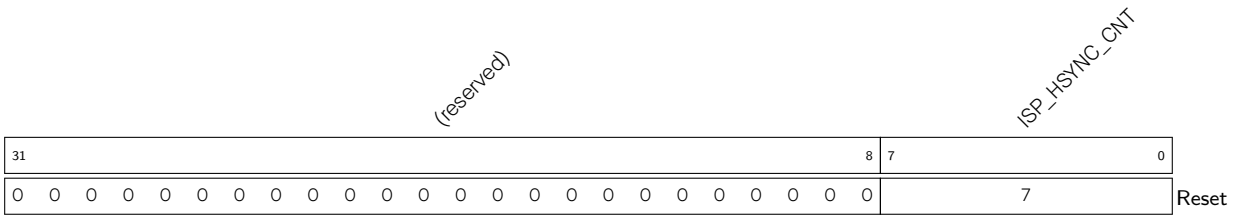
3: YUV420

4: RGB565

Others: Invalid

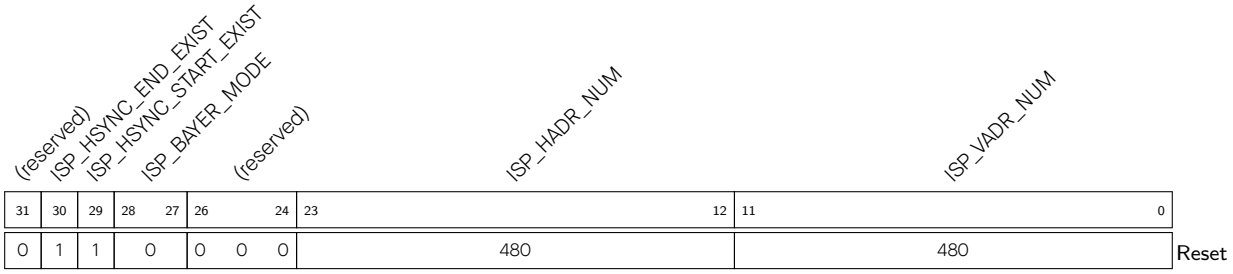
(R/W)

Register 38.4. ISP_HSYNC_CNT_REG (0x000C)



ISP_HSYNC_CNT Configures the interval between hsync and the previous vsync or line_end when converting Image Interface 32 data to ISP data. Generally, the default value is used. Measurement unit: Number of ISP_CLK clock cycles. (R/W)

Register 38.5. ISP_FRAME_CFG_REG (0x0010)



ISP_VADR_NUM Configures the height of the input image, which should be set as the actual number of lines - 1. (R/W)

ISP_HADR_NUM Configures the width of the input image, which should be set as the actual width of lines - 1. (R/W)

ISP_BAYER_MODE Configures the Bayer mode of the input image.

0: BG/GR

1: GB/RG

2: GR/BG

3: RG/GB

(R/W)

ISP_HSYNC_START_EXIST Configures whether the hsync_start packet exists.

0: Not exist

1: Exist

(R/W)

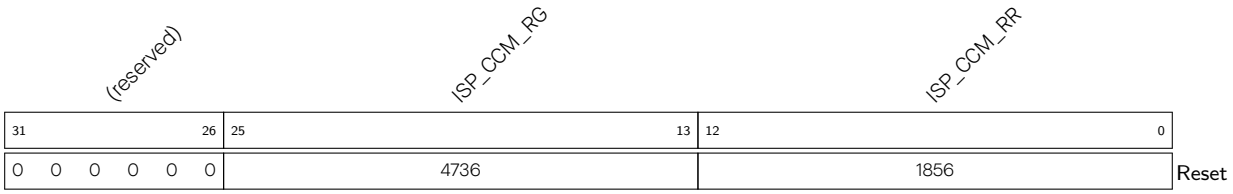
ISP_HSYNC_END_EXIST Configures whether the hsync_end packet exists. This must have the same value as the ISP_HSYNC_START_EXIST field.

0: Not exist

1: Exist

(R/W)

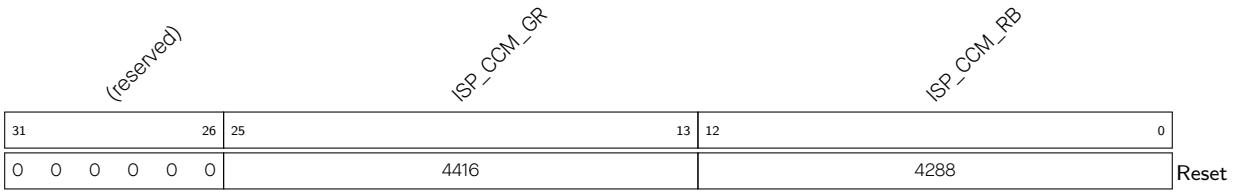
Register 38.6. ISP_CCM_COEF0_REG (0x0014)



ISP_CCM_RR Configures the CCM RR parameter. The 12th bit is the sign bit, and bits [11:0] represent the absolute value. Bits [11:10] are the integer part, and bits [9:0] are the fractional part. (R/W)

ISP_CCM_RG Configures the CCM RG parameter, following the same bit structure and interpretation as ISP_CCM_RR. (R/W)

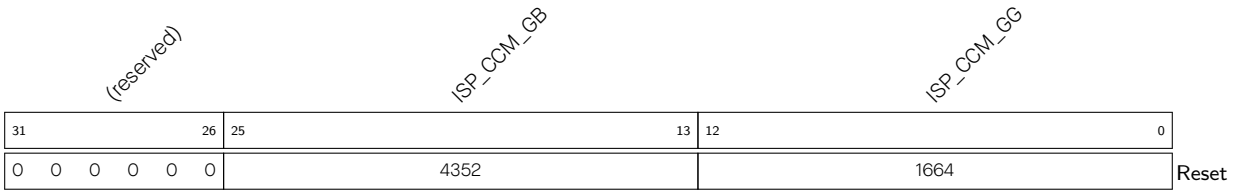
Register 38.7. ISP_CCM_COEF1_REG (0x0018)



ISP_CCM_RB Configures the CCM RB parameter, following the same bit structure and interpretation as ISP_CCM_RR. (R/W)

ISP_CCM_GR Configures the CCM GR parameter, following the same bit structure and interpretation as ISP_CCM_RR. (R/W)

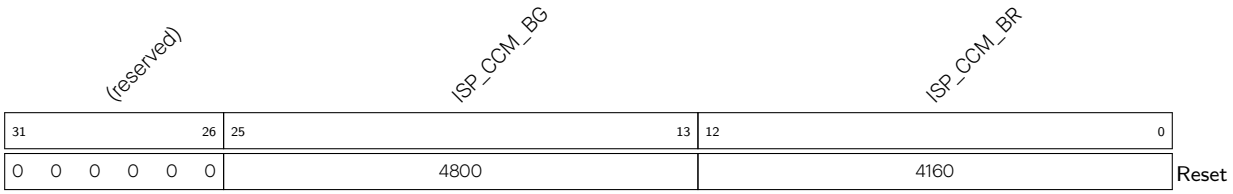
Register 38.8. ISP_CCM_COEF3_REG (0x001C)



ISP_CCM_GG Configures the CCM GG parameter, following the same bit structure and interpretation as ISP_CCM_RR. (R/W)

ISP_CCM_GB Configures the CCM GB parameter, following the same bit structure and interpretation as ISP_CCM_RR. (R/W)

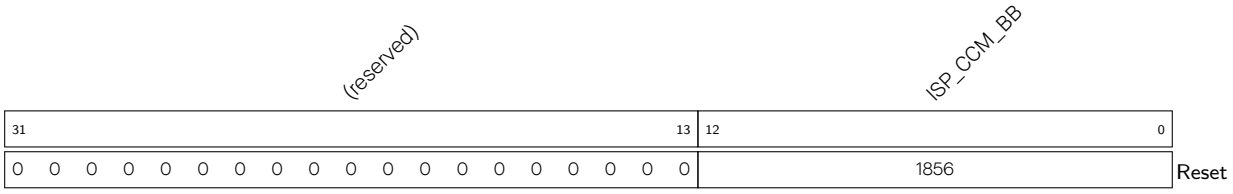
Register 38.9. ISP_CCM_COEF4_REG (0x0020)



ISP_CCM_BR Configures the CCM BR parameter, following the same bit structure and interpretation as ISP_CCM_RR. (R/W)

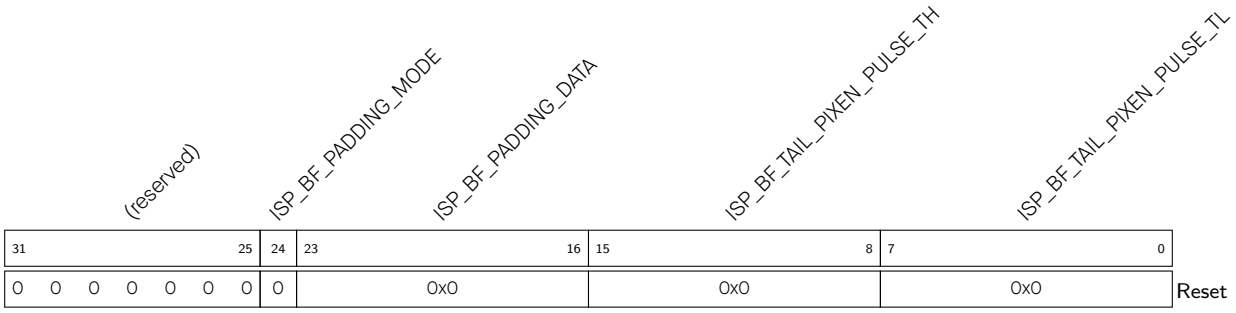
ISP_CCM_BG Configures the CCM BG parameter, following the same bit structure and interpretation as ISP_CCM_RR. (R/W)

Register 38.10. ISP_CCM_COEF5_REG (0x0024)



ISP_CCM_BB Configures the CCM BB parameter, following the same bit structure and interpretation as ISP_CCM_RR. (R/W)

Register 38.11. ISP_BF_MATRIX_CTRL_REG (0x0028)



ISP_BF_TAIL_PIXEN_PULSE_TL Configures the data control cycle for the tail-row data rate during pixel-to-matrix conversion in the BF module.

Within ISP_BF_TAIL_PIXEN_PULSE_TL clock cycles, only the first ISP_BF_TAIL_PIXEN_PULSE_TH cycles are valid. Be cautious to configure this field, to prevent data processing from being deferred to the next frame.

This feature is enabled only when both ISP_BF_TAIL_PIXEN_PULSE_TL and ISP_BF_TAIL_PIXEN_PULSE_TH are non-zero, and ISP_BF_TAIL_PIXEN_PULSE_TH < ISP_BF_TAIL_PIXEN_PULSE_TL. (R/W)

ISP_BF_TAIL_PIXEN_PULSE_TH Configures the valid data cycle of the tail-row data rate during pixel-to-matrix conversion in the BF module, which must be less than ISP_HADR_NUM - 1. (R/W)

ISP_BF_PADDING_DATA Configures the padding data used during image edge expansion in pixel-to-matrix conversion of the BF module. (R/W)

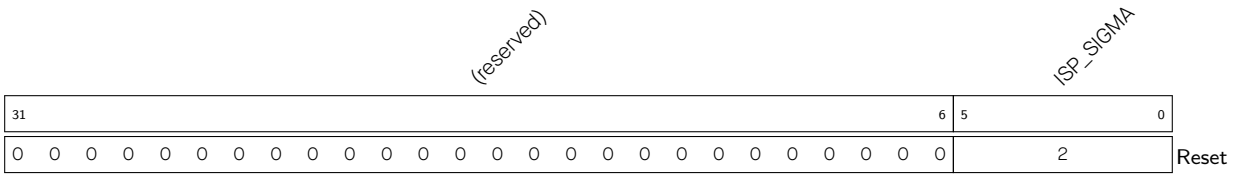
ISP_BF_PADDING_MODE Configures the method of image edge expansion during pixel-to-matrix conversion in the BF module.

0: Automatically padded with pixel values from the image edge

1: Padded with the data in ISP_BF_PADDING_DATA

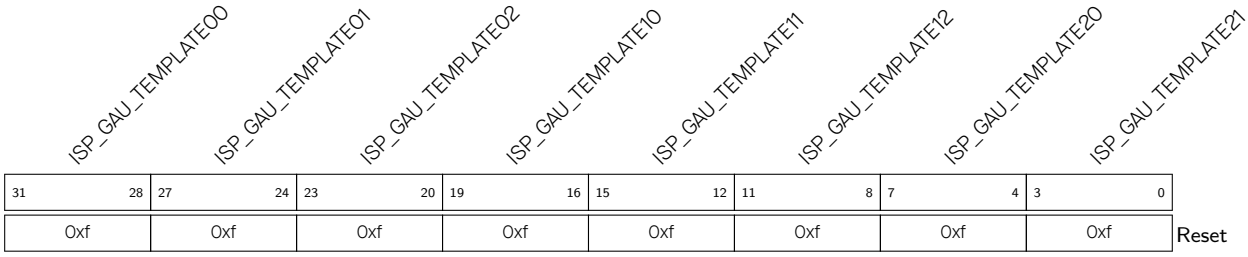
(R/W)

Register 38.12. ISP_BF_SIGMA_REG (0x002C)



ISP_SIGMA Configures the strength of noise reduction in the BF module. Values from 2 to 20 represent increasing strength, while other values indicate maximum strength. (R/W)

Register 38.13. ISP_BF_GAU0_REG (0x0030)



ISP_GAU_TEMPLATE21 Configures the value of coordinate 21 in the noise reduction template of the BF module. (R/W)

ISP_GAU_TEMPLATE20 Configures the value of coordinate 20 in the noise reduction template of the BF module. (R/W)

ISP_GAU_TEMPLATE12 Configures the value of coordinate 12 in the noise reduction template of the BF module. (R/W)

ISP_GAU_TEMPLATE11 Configures the value of coordinate 11 in the noise reduction template of the BF module. (R/W)

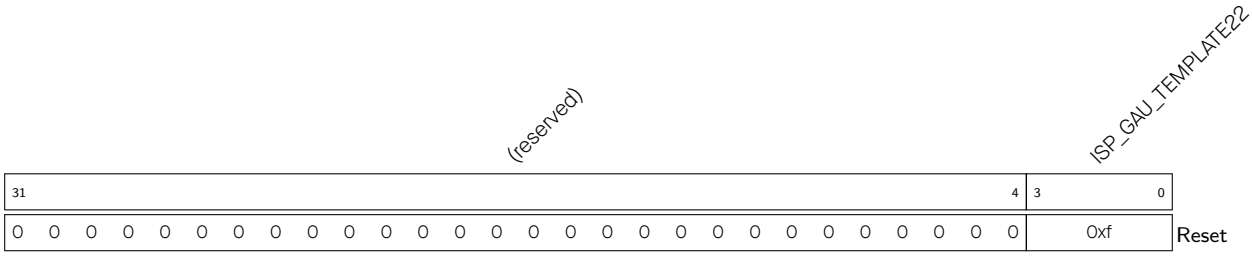
ISP_GAU_TEMPLATE10 Configures the value of coordinate 10 in the noise reduction template of the BF module. (R/W)

ISP_GAU_TEMPLATE02 Configures the value of coordinate 02 in the noise reduction template of the BF module. (R/W)

ISP_GAU_TEMPLATE01 Configures the value of coordinate 01 in the noise reduction template of the BF module. (R/W)

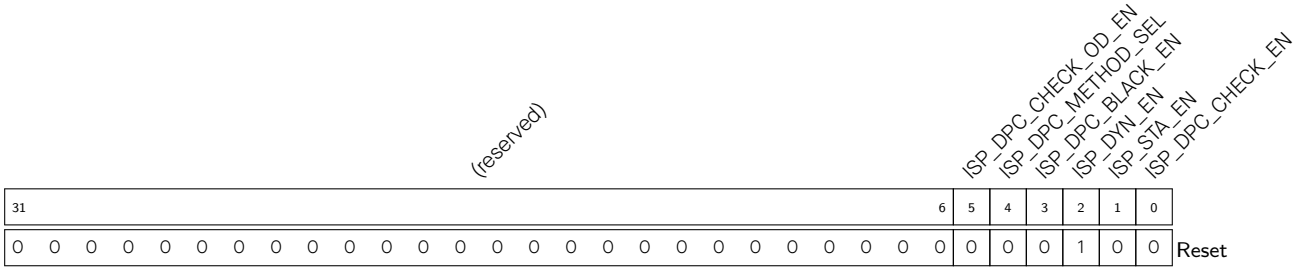
ISP_GAU_TEMPLATE00 Configures the value of coordinate 00 in the noise reduction template of the BF module. (R/W)

Register 38.14. ISP_BF_GAU1_REG (0x0034)



ISP_GAU_TEMPLATE22 Configures the value of coordinate 22 in the noise reduction template of the BF module. (R/W)

Register 38.15. ISP_DPC_CTRL_REG (0x0038)



ISP_DPC_CHECK_EN Configures whether to enable the static calibration function. Calibration and correction cannot be enabled at the same time.

0: Disable

1: Enable

(R/W)

ISP_STA_EN Configures whether to enable the static correction function.

0: Disable

1: Enable

(R/W)

ISP_DYN_EN Configures whether to enable the dynamic correction function.

0: Disable

1: Enable

(R/W)

ISP_DPC_BLACK_EN Configures the image type used for static calibration.

0: White image

1: Black image

(R/W)

ISP_DPC_METHOD_SEL Configures the algorithm selection for dynamic correction.

0: Select Algorithm 0

1: Select Algorithm 1

(R/W)

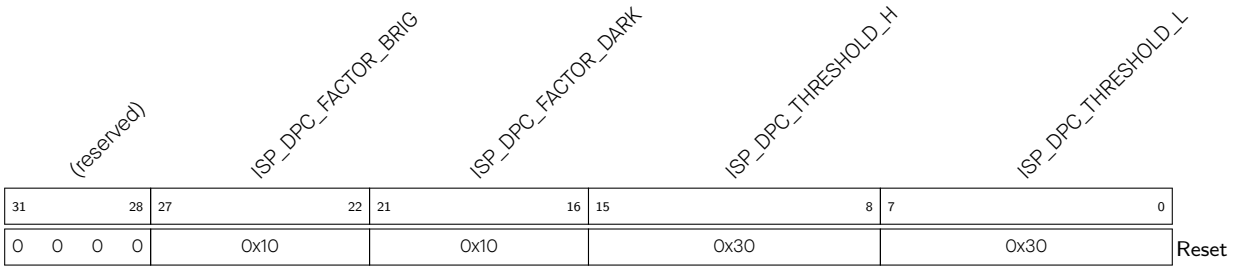
ISP_DPC_CHECK_OD_EN Configures whether to output image data during the static calibration process.

0: Do not output image

1: Output image

(R/W)

Register 38.16. ISP_DPC_CONF_REG (0x003C)



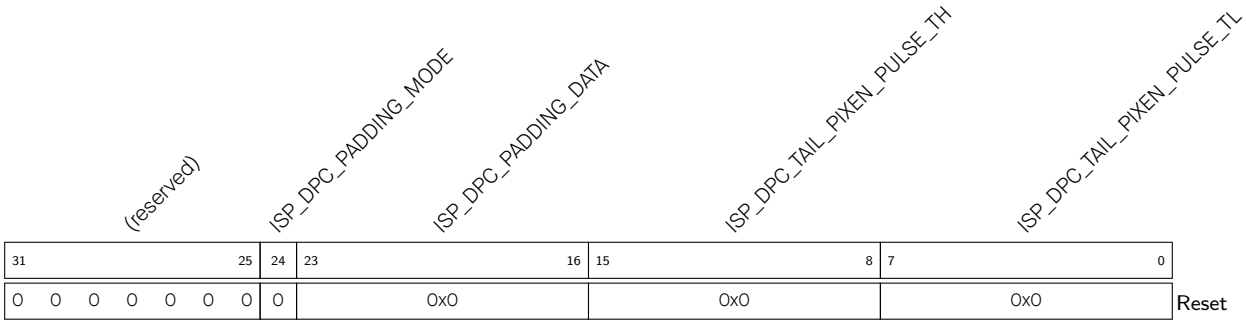
ISP_DPC_THRESHOLD_L Configures the defective pixel threshold for black image static calibration, or the low threshold in dynamic correction Algorithm 0 (8-bit valid), or the low threshold in dynamic correction Algorithm 1 (5-bit valid, maximum value is 16). For details, refer to Section 38.5.2.2. (R/W)

ISP_DPC_THRESHOLD_H Configures the defective pixel threshold for white image static calibration, or the high threshold in dynamic correction Algorithm 0 (8-bit valid), or the high threshold in dynamic correction Algorithm 1 (5-bit valid, maximum value is 16). For details, refer to Section 38.5.2.2. (R/W)

ISP_DPC_FACTOR_DARK Configures the dark pixel factor used in dynamic correction Algorithm 1. (R/W)

ISP_DPC_FACTOR_BRIG Configures the bright pixel factor used in dynamic correction Algorithm 1. (R/W)

Register 38.17. ISP_DPC_MATRIX_CTRL_REG (0x0040)



ISP_DPC_TAIL_PIXEN_PULSE_TL Configures the data control cycle for the tail-row data rate during pixel-to-matrix conversion in the DPC module. Within the ISP_DPC_TAIL_PIXEN_PULSE_TL clock cycles, only the first ISP_DPC_TAIL_PIXEN_PULSE_TH cycles are valid. Be cautious to configure this field, to prevent data processing from being deferred to the next frame.

This function is enabled only when both ISP_DPC_TAIL_PIXEN_PULSE_TL and ISP_DPC_TAIL_PIXEN_PULSE_TH are non-zero, and ISP_DPC_TAIL_PIXEN_PULSE_TH < ISP_DPC_TAIL_PIXEN_PULSE_TL. (R/W)

ISP_DPC_TAIL_PIXEN_PULSE_TH Configures the valid data cycle of the tail-row data rate during pixel-to-matrix conversion in the DPC module, which must be less than ISP_HADR_NUM − 1. (R/W)

ISP_DPC_PADDING_DATA Configures the padding data used for image edge expansion during pixel-to-matrix conversion in the DPC module. (R/W)

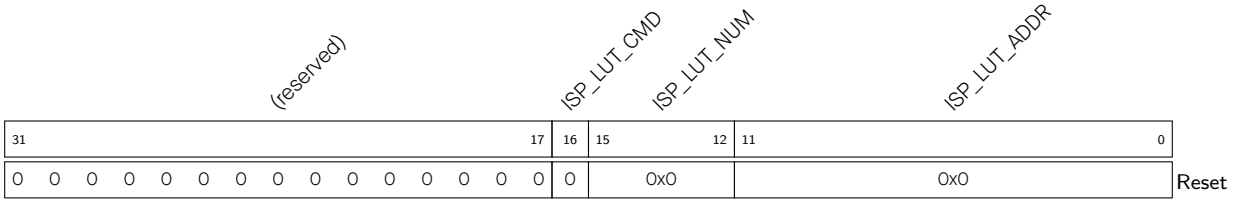
ISP_DPC_PADDING_MODE Configures the method of image edge expansion during pixel-to-matrix conversion in the DPC module.

0: Automatically padded with pixel values from the image edge

1: Padded with the data in ISP_DPC_PADDING_DATA

(R/W)

Register 38.18. ISP_LUT_CMD_REG (0x0048)

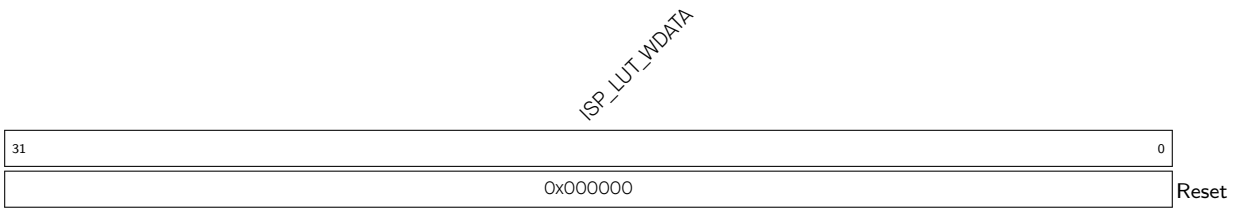


ISP_LUT_ADDR Configures the LUT address. When ISP_LUT_NUM selects LSC LUT, [11:10] being 0 indicates selecting Gb_B LUT, and being 1 indicates selecting R_Gr LUT. Bits [9:0] represent the actual LUT address.
(WT)

ISP_LUT_NUM Configures the LUT selection.
0: Select LSC LUT
Others: Invalid
(WT)

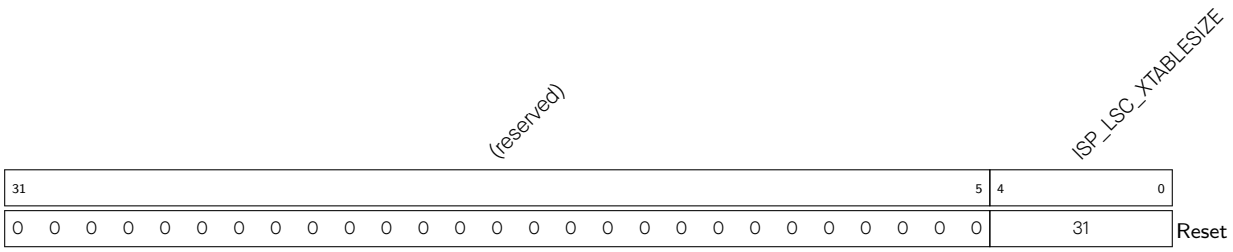
ISP_LUT_CMD Configures whether to read from or write to the LUT.
0: Read from the LUT
1: Write to the LUT
(WT)

Register 38.19. ISP_LUT_WDATA_REG (0x004C)



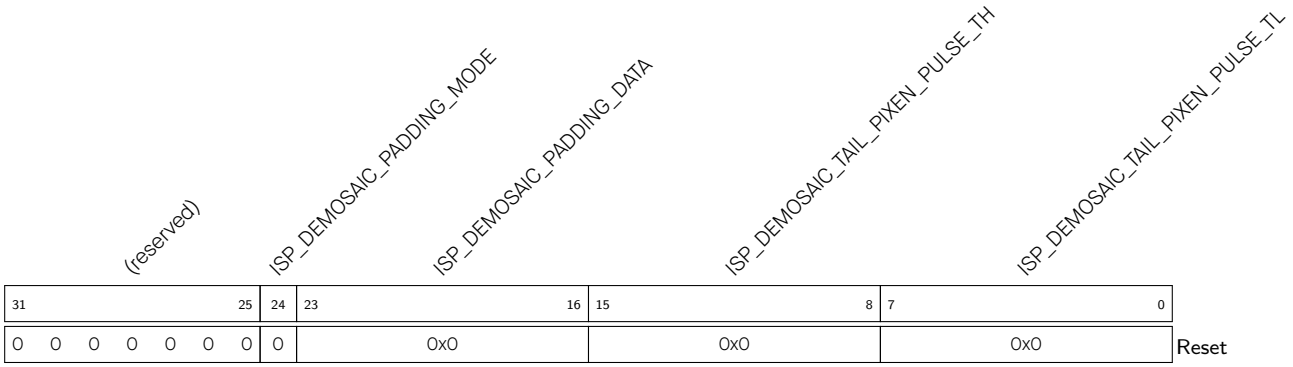
ISP_LUT_WDATA Data to be written to the LUT, which must be configured before writing to ISP_LUT_CMD_REG. (R/W)

Register 38.20. ISP_LSC_TABLESIZE_REG (0x0054)



ISP_LSC_XTABLESIZE Configures the number of horizontal grids in the LSC module. This value is obtained by $fix((line_number - 1)/2/32) + 2$, where $fix()$ denotes the rounding. (R/W)

Register 38.21. ISP_DEMOSAIC_MATRIX_CTRL_REG (0x0058)



ISP_DEMOSAIC_TAIL_PIXEN_PULSE_TL Configures the data control cycle for the tail-row data rate during pixel-to-matrix conversion in the demosaic module.

Within ISP_DEMOSAIC_TAIL_PIXEN_PULSE_TL clock cycles, only the first ISP_DEMOSAIC_TAIL_PIXEN_PULSE_TH cycles are valid. Be cautious to configure this field, to prevent data processing from being deferred to the next frame.

This feature is enabled only when both ISP_DEMOSAIC_TAIL_PIXEN_PULSE_TL and ISP_DEMOSAIC_TAIL_PIXEN_PULSE_TH are non-zero, and ISP_DEMOSAIC_TAIL_PIXEN_PULSE_TH < ISP_DEMOSAIC_TAIL_PIXEN_PULSE_TL. (R/W)

ISP_DEMOSAIC_TAIL_PIXEN_PULSE_TH Configures the valid data cycle of the tail-row data rate during pixel-to-matrix conversion in the demosaic module, which must be less than ISP_HADR_NUM - 1. (R/W)

ISP_DEMOSAIC_PADDING_DATA Configures the padding data used during image edge expansion in pixel-to-matrix conversion of the demosaic module. (R/W)

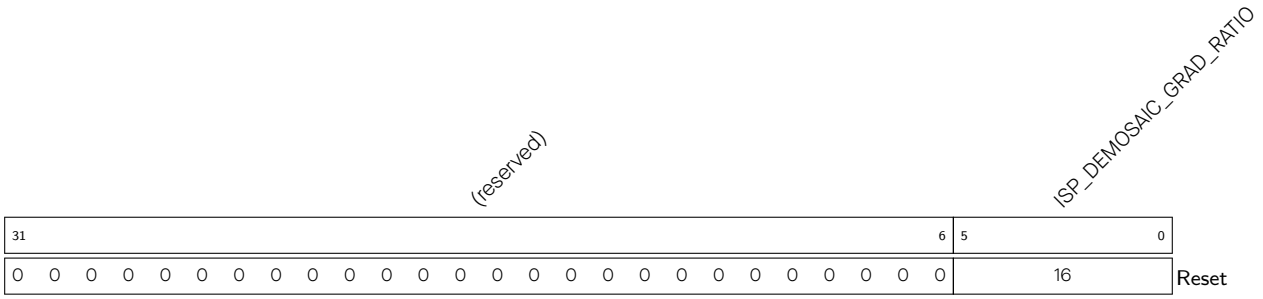
ISP_DEMOSAIC_PADDING_MODE Configures the method of image edge expansion during pixel-to-matrix conversion in the demosaic module.

0: Automatically padded with pixel values from the image edge

1: Padded with the data in ISP_DEMOSAIC_PADDING_DATA

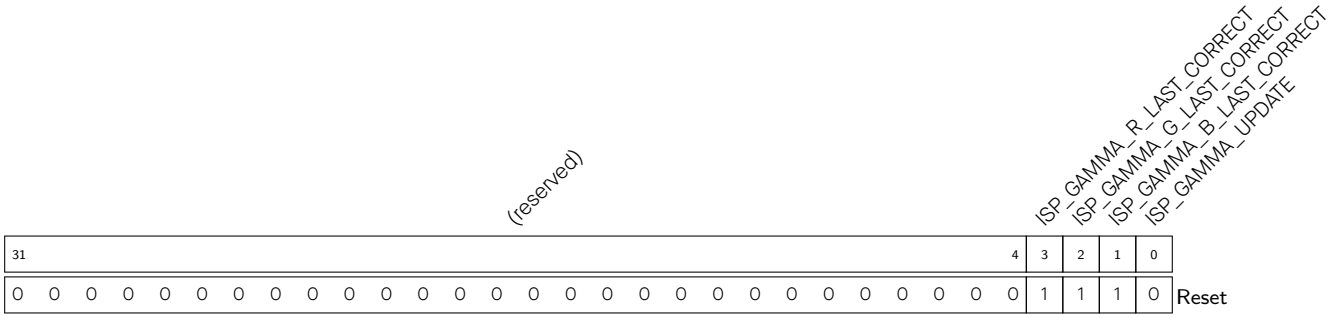
(R/W)

Register 38.22. ISP_DEMOSAIC_GRAD_RATIO_REG (0x005C)



ISP_DEMOSAIC_GRAD_RATIO Configures the demosaic effect, with bits [5:4] as the integer part and bits [3:0] as the fractional part. (R/W)

Register 38.23. ISP_GAMMA_CTRL_REG (0x0074)



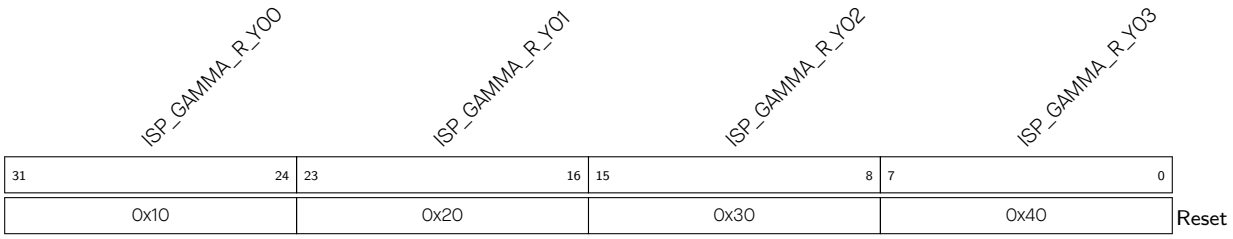
ISP_GAMMA_UPDATE Configures whether to update the gamma curve parameters. Writing 1 to this field updates the gamma curve parameters. It will be automatically cleared to 0 after the update is complete. (R/W)

ISP_GAMMA_B_LAST_CORRECT Configures whether to enable correction for the parameters of the last interval on the X axis of B channel gamma curve.
0: Not enable
1: Enable
(R/W)

ISP_GAMMA_G_LAST_CORRECT Configures whether to enable correction for the parameters of the last interval on the X axis of G channel gamma curve.
0: Not enable
1: Enable
(R/W)

ISP_GAMMA_R_LAST_CORRECT Configures whether to enable correction for the parameters of the last interval on the X axis of X channel gamma curve.
0: Not enable
1: Enable
(R/W)

Register 38.24. ISP_GAMMA_RY1_REG (0x0078)



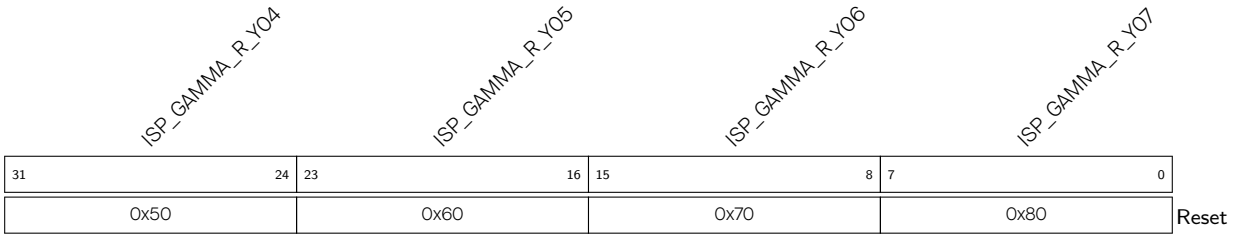
ISP_GAMMA_R_Y03 Configures the value of the 3rd point on the Y axis of R channel gamma curve. (R/W)

ISP_GAMMA_R_Y02 Configures the value of the 2nd point on the Y axis of R channel gamma curve. (R/W)

ISP_GAMMA_R_Y01 Configures the value of the 1st point on the Y axis of R channel gamma curve. (R/W)

ISP_GAMMA_R_Y00 Configures the value of the 0th point on the Y axis of R channel gamma curve. (R/W)

Register 38.25. ISP_GAMMA_RY2_REG (0x007C)



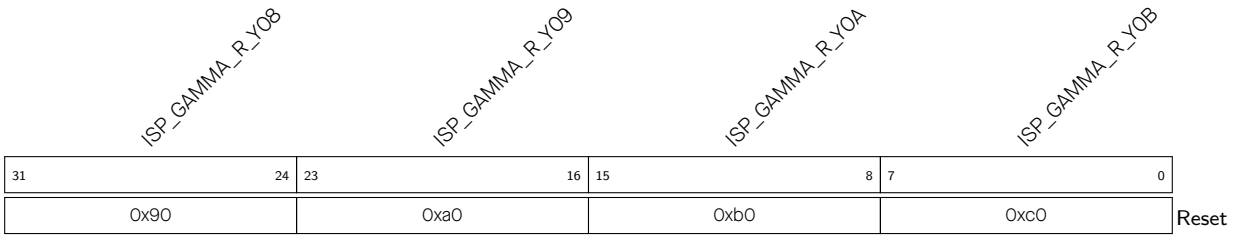
ISP_GAMMA_R_Y07 Configures the value of the 7th point on the Y axis of R channel gamma curve. (R/W)

ISP_GAMMA_R_Y06 Configures the value of the 6th point on the Y axis of R channel gamma curve. (R/W)

ISP_GAMMA_R_Y05 Configures the value of the 5th point on the Y axis of R channel gamma curve. (R/W)

ISP_GAMMA_R_Y04 Configures the value of the 4th point on the Y axis of R channel gamma curve. (R/W)

Register 38.26. ISP_GAMMA_RY3_REG (0x0080)



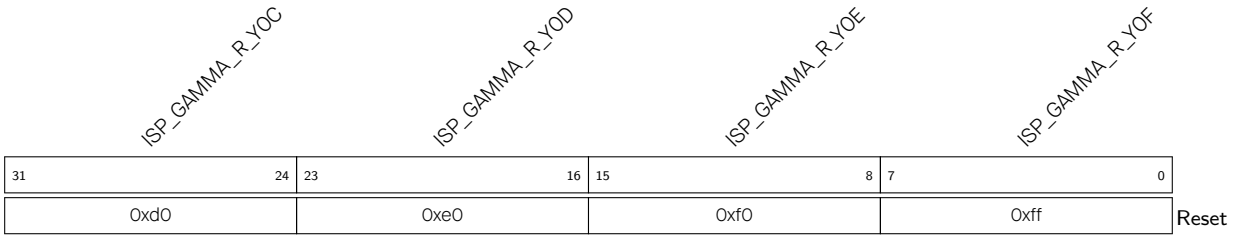
ISP_GAMMA_R_Y0B Configures the value of the 11th point on the Y axis of R channel gamma curve. (R/W)

ISP_GAMMA_R_Y0A Configures the value of the 10th point on the Y axis of R channel gamma curve. (R/W)

ISP_GAMMA_R_Y09 Configures the value of the 9th point on the Y axis of R channel gamma curve. (R/W)

ISP_GAMMA_R_Y08 Configures the value of the 8th point on the Y axis of R channel gamma curve. (R/W)

Register 38.27. ISP_GAMMA_RY4_REG (0x0084)



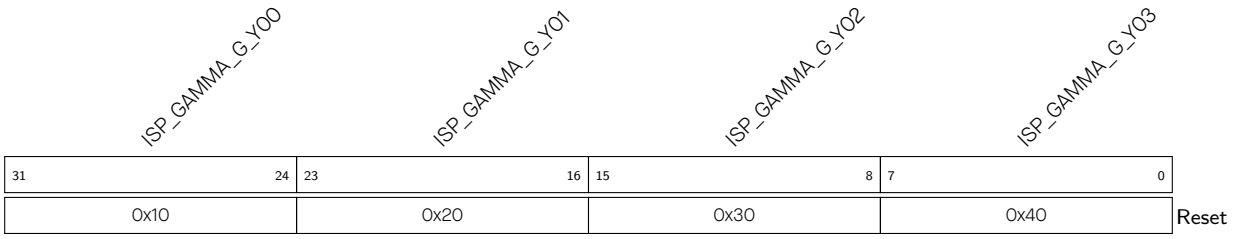
ISP_GAMMA_R_Y0F Configures the value of the 15th point on the Y axis of R channel gamma curve. (R/W)

ISP_GAMMA_R_Y0E Configures the value of the 14th point on the Y axis of R channel gamma curve. (R/W)

ISP_GAMMA_R_Y0D Configures the value of the 13th point on the Y axis of R channel gamma curve. (R/W)

ISP_GAMMA_R_Y0C Configures the value of the 12th point on the Y axis of R channel gamma curve. (R/W)

Register 38.28. ISP_GAMMA_GY1_REG (0x0088)



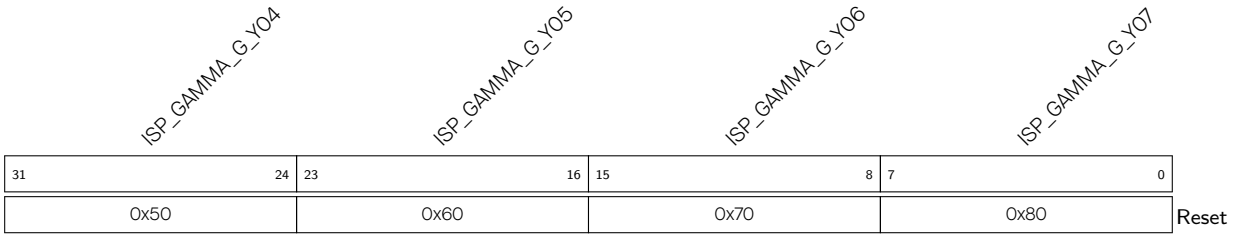
ISP_GAMMA_G_Y03 Configures the value of the 3rd point on the Y axis of G channel gamma curve. (R/W)

ISP_GAMMA_G_Y02 Configures the value of the 2nd point on the Y axis of G channel gamma curve. (R/W)

ISP_GAMMA_G_Y01 Configures the value of the 1st point on the Y axis of G channel gamma curve. (R/W)

ISP_GAMMA_G_Y00 Configures the value of the 0th point on the Y axis of G channel gamma curve. (R/W)

Register 38.29. ISP_GAMMA_GY2_REG (0x008C)



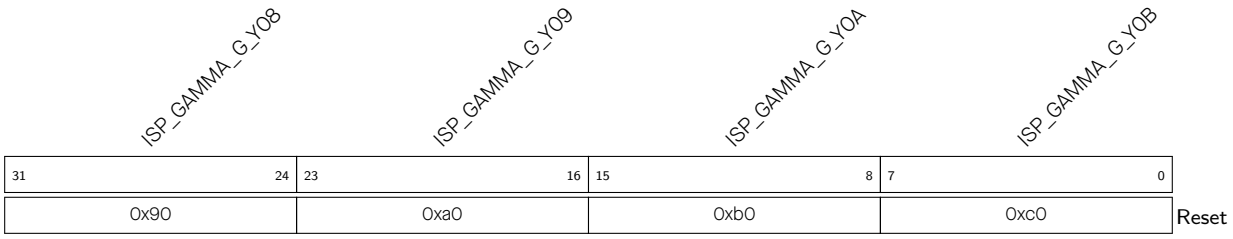
ISP_GAMMA_G_Y07 Configures the value of the 7th point on the Y axis of G channel gamma curve. (R/W)

ISP_GAMMA_G_Y06 Configures the value of the 6th point on the Y axis of G channel gamma curve. (R/W)

ISP_GAMMA_G_Y05 Configures the value of the 5th point on the Y axis of G channel gamma curve. (R/W)

ISP_GAMMA_G_Y04 Configures the value of the 4th point on the Y axis of G channel gamma curve. (R/W)

Register 38.30. ISP_GAMMA_GY3_REG (0x0090)



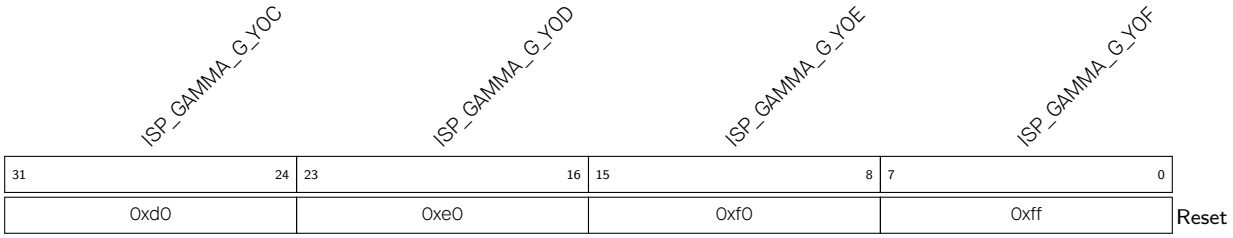
ISP_GAMMA_G_Y0B Configures the value of the 11th point on the Y axis of G channel gamma curve. (R/W)

ISP_GAMMA_G_Y0A Configures the value of the 10th point on the Y axis of G channel gamma curve. (R/W)

ISP_GAMMA_G_Y09 Configures the value of the 9th point on the Y axis of G channel gamma curve. (R/W)

ISP_GAMMA_G_Y08 Configures the value of the 8th point on the Y axis of G channel gamma curve. (R/W)

Register 38.31. ISP_GAMMA_GY4_REG (0x0094)



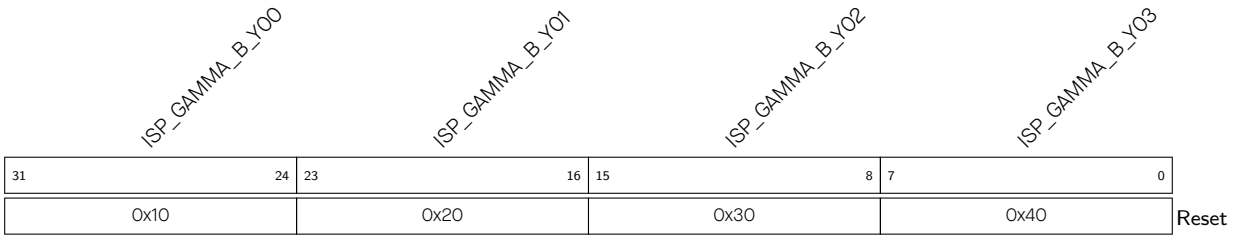
ISP_GAMMA_G_Y0F Configures the value of the 15th point on the Y axis of G channel gamma curve. (R/W)

ISP_GAMMA_G_Y0E Configures the value of the 14th point on the Y axis of G channel gamma curve. (R/W)

ISP_GAMMA_G_Y0D Configures the value of the 13th point on the Y axis of G channel gamma curve. (R/W)

ISP_GAMMA_G_Y0C Configures the value of the 12th point on the Y axis of G channel gamma curve. (R/W)

Register 38.32. ISP_GAMMA_BY1_REG (0x0098)



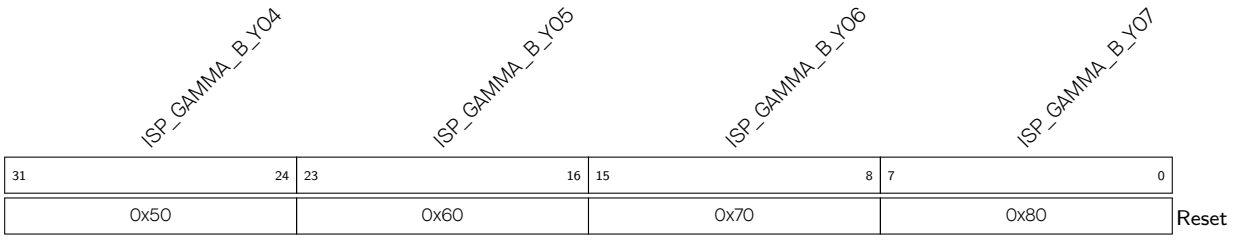
ISP_GAMMA_B_Y03 Configures the value of the 3rd point on the Y axis of B channel gamma curve. (R/W)

ISP_GAMMA_B_Y02 Configures the value of the 2nd point on the Y axis of B channel gamma curve. (R/W)

ISP_GAMMA_B_Y01 Configures the value of the 1st point on the Y axis of B channel gamma curve. (R/W)

ISP_GAMMA_B_Y00 Configures the value of the 0th point on the Y axis of B channel gamma curve. (R/W)

Register 38.33. ISP_GAMMA_BY2_REG (0x009C)



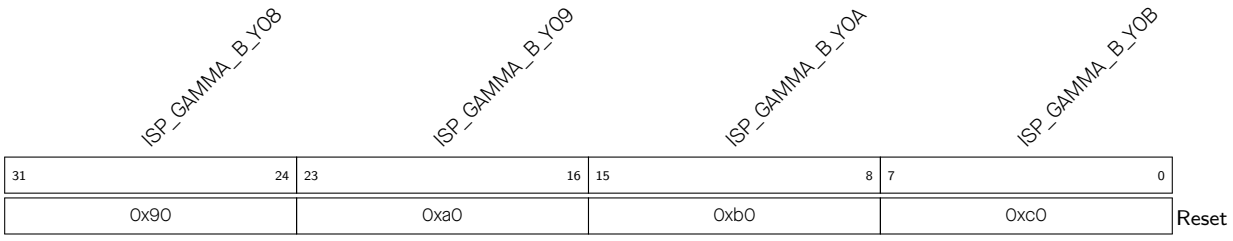
ISP_GAMMA_B_Y07 Configures the value of the 7th point on the Y axis of B channel gamma curve. (R/W)

ISP_GAMMA_B_Y06 Configures the value of the 6th point on the Y axis of B channel gamma curve. (R/W)

ISP_GAMMA_B_Y05 Configures the value of the 5th point on the Y axis of B channel gamma curve. (R/W)

ISP_GAMMA_B_Y04 Configures the value of the 4th point on the Y axis of B channel gamma curve. (R/W)

Register 38.34. ISP_GAMMA_BY3_REG (0x00A0)



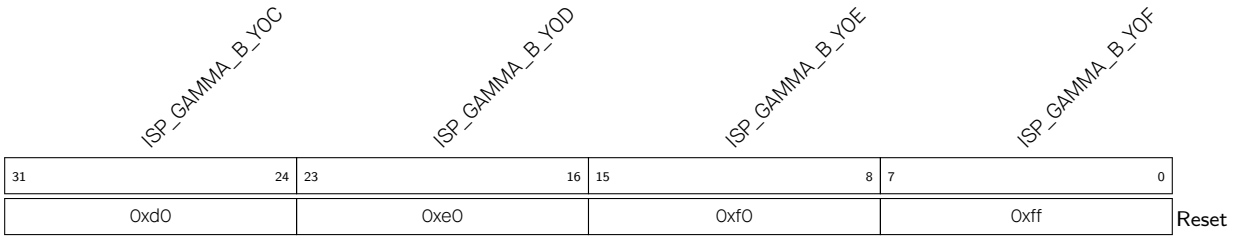
ISP_GAMMA_B_Y0B Configures the value of the 11th point on the Y axis of B channel gamma curve. (R/W)

ISP_GAMMA_B_Y0A Configures the value of the 10th point on the Y axis of B channel gamma curve. (R/W)

ISP_GAMMA_B_Y09 Configures the value of the 9th point on the Y axis of B channel gamma curve. (R/W)

ISP_GAMMA_B_Y08 Configures the value of the 8th point on the Y axis of B channel gamma curve. (R/W)

Register 38.35. ISP_GAMMA_BY4_REG (0x00A4)



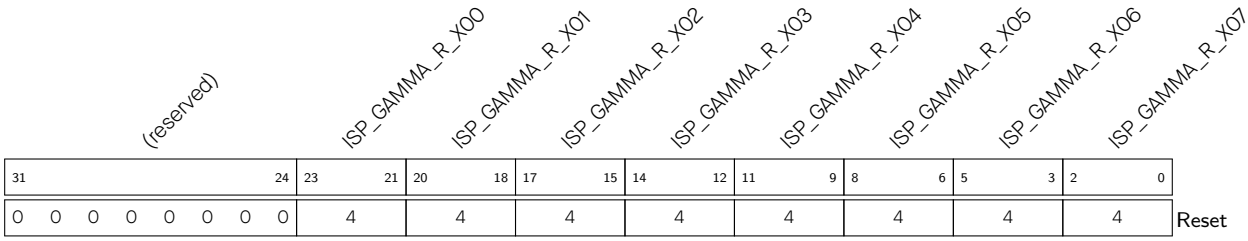
ISP_GAMMA_B_Y0F Configures the value of the 15th point on the Y axis of B channel gamma curve. (R/W)

ISP_GAMMA_B_Y0E Configures the value of the 14th point on the Y axis of B channel gamma curve. (R/W)

ISP_GAMMA_B_Y0D Configures the value of the 13th point on the Y axis of B channel gamma curve. (R/W)

ISP_GAMMA_B_Y0C Configures the value of the 12th point on the Y axis of B channel gamma curve. (R/W)

Register 38.36. ISP_GAMMA_RX1_REG (0x00A8)



ISP_GAMMA_R_X07 Configures the width of the 7th interval on the X axis of R channel gamma curve. Refer to section 38.5.2.8 for configuration. (R/W)

ISP_GAMMA_R_X06 Configures the width of the 6th interval on the X axis of R channel gamma curve. (R/W)

ISP_GAMMA_R_X05 Configures the width of the 5th interval on the X axis of R channel gamma curve. (R/W)

ISP_GAMMA_R_X04 Configures the width of the 4th interval on the X axis of R channel gamma curve. (R/W)

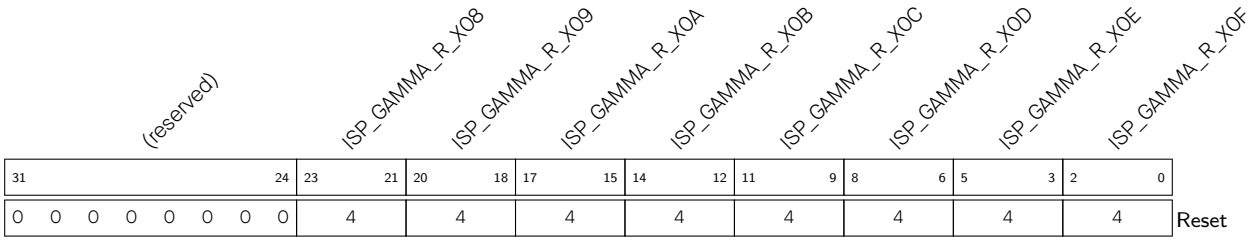
ISP_GAMMA_R_X03 Configures the width of the 3rd interval on the X axis of R channel gamma curve. (R/W)

ISP_GAMMA_R_X02 Configures the width of the 2nd interval on the X axis of R channel gamma curve. (R/W)

ISP_GAMMA_R_X01 Configures the width of the 1st interval on the X axis of R channel gamma curve. (R/W)

ISP_GAMMA_R_X00 Configures the width of the 0th interval on the X axis of R channel gamma curve. (R/W)

Register 38.37. ISP_GAMMA_RX2_REG (0x00AC)



ISP_GAMMA_R_X0F Configures the width of the 15th interval on the X axis of R channel gamma curve. Refer to section 38.5.2.8 for configuration. (R/W)

ISP_GAMMA_R_X0E Configures the width of the 14th interval on the X axis of R channel gamma curve. (R/W)

ISP_GAMMA_R_X0D Configures the width of the 13th interval on the X axis of R channel gamma curve. (R/W)

ISP_GAMMA_R_X0C Configures the width of the 12th interval on the X axis of R channel gamma curve. (R/W)

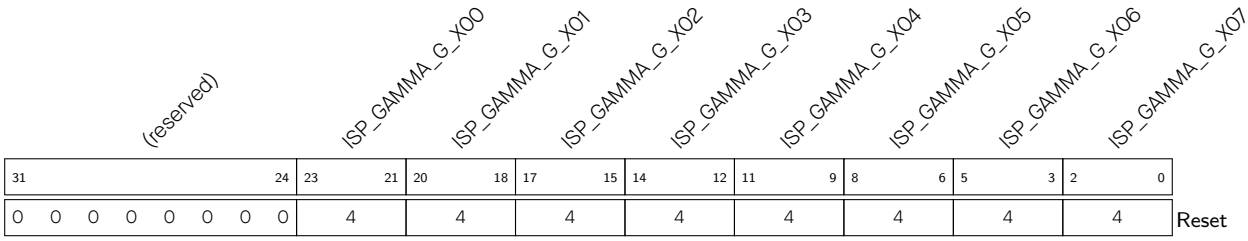
ISP_GAMMA_R_X0B Configures the width of the 11th interval on the X axis of R channel gamma curve. (R/W)

ISP_GAMMA_R_X0A Configures the width of the 10th interval on the X axis of R channel gamma curve. (R/W)

ISP_GAMMA_R_X09 Configures the width of the 9th interval on the X axis of R channel gamma curve. (R/W)

ISP_GAMMA_R_X08 Configures the width of the 8th interval on the X axis of R channel gamma curve. (R/W)

Register 38.38. ISP_GAMMA_GX1_REG (0x00B0)



ISP_GAMMA_G_X07 Configures the width of the 7th interval on the X axis of G channel gamma curve. Refer to section 38.5.2.8 for configuration. (R/W)

ISP_GAMMA_G_X06 Configures the width of the 6th interval on the X axis of G channel gamma curve. (R/W)

ISP_GAMMA_G_X05 Configures the width of the 5th interval on the X axis of G channel gamma curve. (R/W)

ISP_GAMMA_G_X04 Configures the width of the 4th interval on the X axis of G channel gamma curve. (R/W)

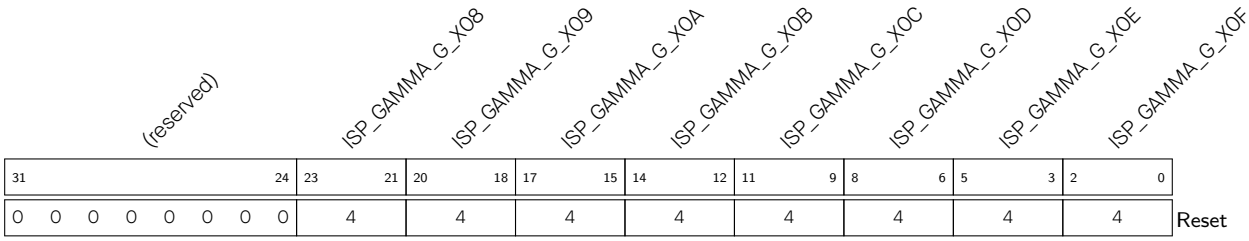
ISP_GAMMA_G_X03 Configures the width of the 3rd interval on the X axis of G channel gamma curve. (R/W)

ISP_GAMMA_G_X02 Configures the width of the 2nd interval on the X axis of G channel gamma curve. (R/W)

ISP_GAMMA_G_X01 Configures the width of the 1st interval on the X axis of G channel gamma curve. (R/W)

ISP_GAMMA_G_X00 Configures the width of the 0th interval on the X axis of G channel gamma curve. (R/W)

Register 38.39. ISP_GAMMA_GX2_REG (0x00B4)



ISP_GAMMA_G_X0F Configures the width of the 15th interval on the X axis of G channel gamma curve. Refer to section 38.5.2.8 for configuration. (R/W)

ISP_GAMMA_G_X0E Configures the width of the 14th interval on the X axis of G channel gamma curve. (R/W)

ISP_GAMMA_G_X0D Configures the width of the 13th interval on the X axis of G channel gamma curve. (R/W)

ISP_GAMMA_G_X0C Configures the width of the 12th interval on the X axis of G channel gamma curve. (R/W)

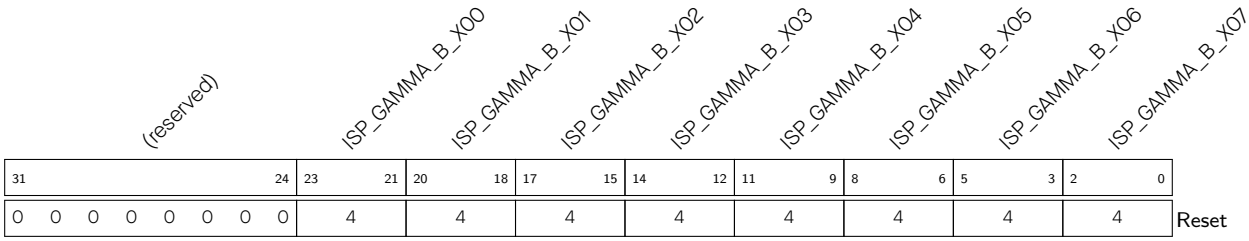
ISP_GAMMA_G_X0B Configures the width of the 11th interval on the X axis of G channel gamma curve. (R/W)

ISP_GAMMA_G_X0A Configures the width of the 10th interval on the X axis of G channel gamma curve. (R/W)

ISP_GAMMA_G_X09 Configures the width of the 9th interval on the X axis of G channel gamma curve. (R/W)

ISP_GAMMA_G_X08 Configures the width of the 8th interval on the X axis of G channel gamma curve. (R/W)

Register 38.40. ISP_GAMMA_BX1_REG (0x00B8)



ISP_GAMMA_B_X07 Configures the width of the 7th interval on the X axis of B channel gamma curve. Refer to section 38.5.2.8 for configuration. (R/W)

ISP_GAMMA_B_X06 Configures the width of the 6th interval on the X axis of B channel gamma curve. (R/W)

ISP_GAMMA_B_X05 Configures the width of the 5th interval on the X axis of B channel gamma curve. (R/W)

ISP_GAMMA_B_X04 Configures the width of the 4th interval on the X axis of B channel gamma curve. (R/W)

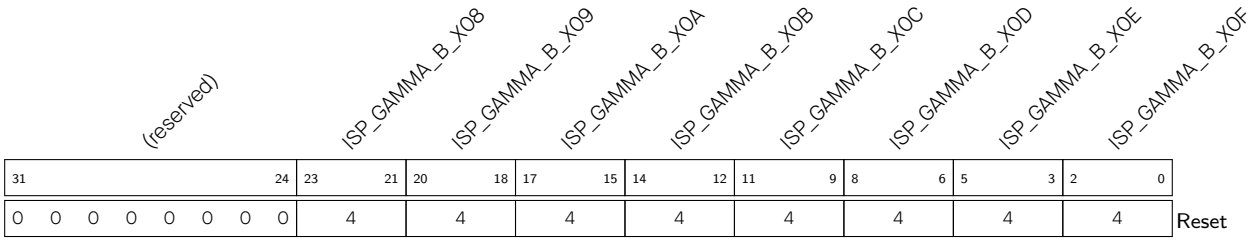
ISP_GAMMA_B_X03 Configures the width of the 3rd interval on the X axis of B channel gamma curve. (R/W)

ISP_GAMMA_B_X02 Configures the width of the 2nd interval on the X axis of B channel gamma curve. (R/W)

ISP_GAMMA_B_X01 Configures the width of the 1st interval on the X axis of B channel gamma curve. (R/W)

ISP_GAMMA_B_X00 Configures the width of the 0th interval on the X axis of B channel gamma curve. (R/W)

Register 38.41. ISP_GAMMA_BX2_REG (0x00BC)



ISP_GAMMA_B_X0F Configures the width of the 15th interval on the X axis of B channel gamma curve. Refer to section 38.5.2.8 for configuration. (R/W)

ISP_GAMMA_B_X0E Configures the width of the 14th interval on the X axis of B channel gamma curve. (R/W)

ISP_GAMMA_B_X0D Configures the width of the 13th interval on the X axis of B channel gamma curve. (R/W)

ISP_GAMMA_B_X0C Configures the width of the 12th interval on the X axis of B channel gamma curve. (R/W)

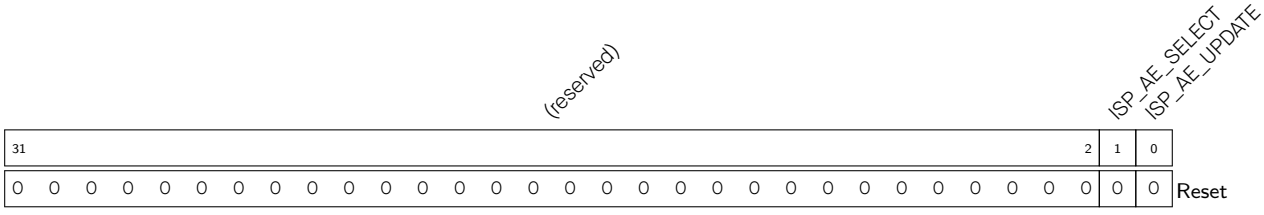
ISP_GAMMA_B_X0B Configures the width of the 11th interval on the X axis of B channel gamma curve. (R/W)

ISP_GAMMA_B_X0A Configures the width of the 10th interval on the X axis of B channel gamma curve. (R/W)

ISP_GAMMA_B_X09 Configures the width of the 9th interval on the X axis of B channel gamma curve. (R/W)

ISP_GAMMA_B_X08 Configures the width of the 8th interval on the X axis of B channel gamma curve. (R/W)

Register 38.42. ISP_AE_CTRL_REG (0x00C0)



ISP_AE_UPDATE Configures AE statistics sampling. With AE enabled, writing 1 triggers a statistics collection. (WT)

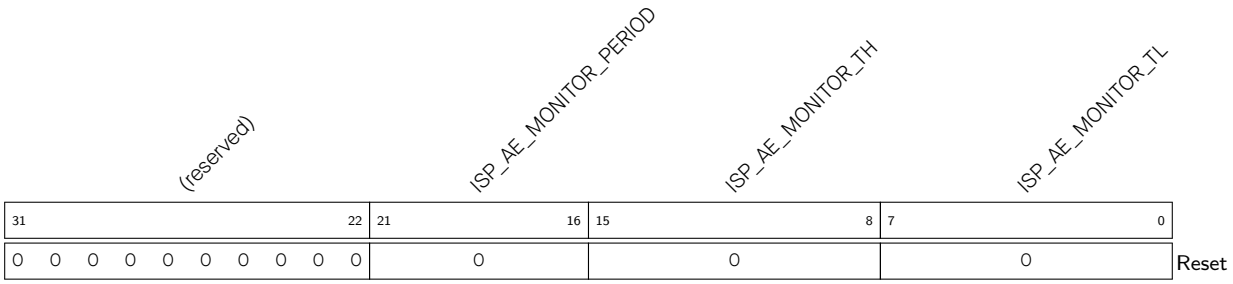
ISP_AE_SELECT Configures the AE sampling point.

0: Use data output from the demosaic module

1: Use data output from the gamma correction module

(R/W)

Register 38.43. ISP_AE_MONITOR_REG (0x00C4)

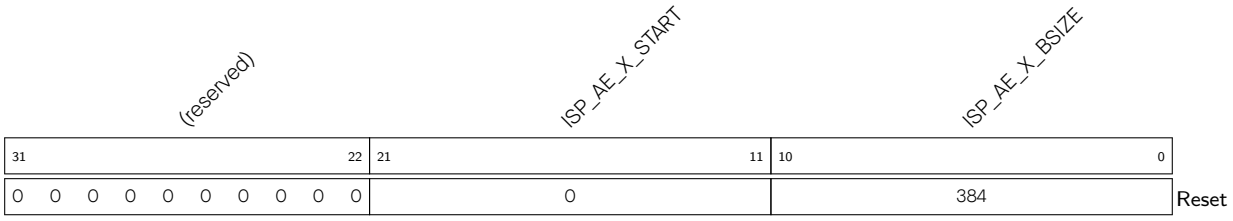


ISP_AE_MONITOR_TL Configures the lower threshold for AE luminance monitoring. (R/W)

ISP_AE_MONITOR_TH Configures the upper threshold for AE luminance monitoring. (R/W)

ISP_AE_MONITOR_PERIOD Configures the frame interval for AE monitoring. (R/W)

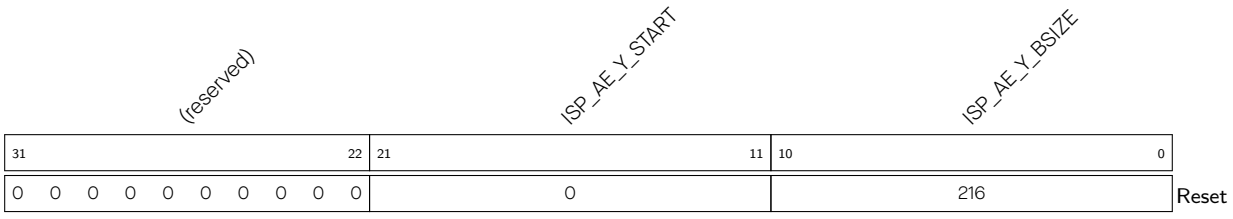
Register 38.44. ISP_AE_BX_REG (0x00C8)



ISP_AE_X_BSIZE Configures the horizontal size of each sub-window. (R/W)

ISP_AE_X_START Configures the start coordinate of the AE statistical window in the horizontal direction. (R/W)

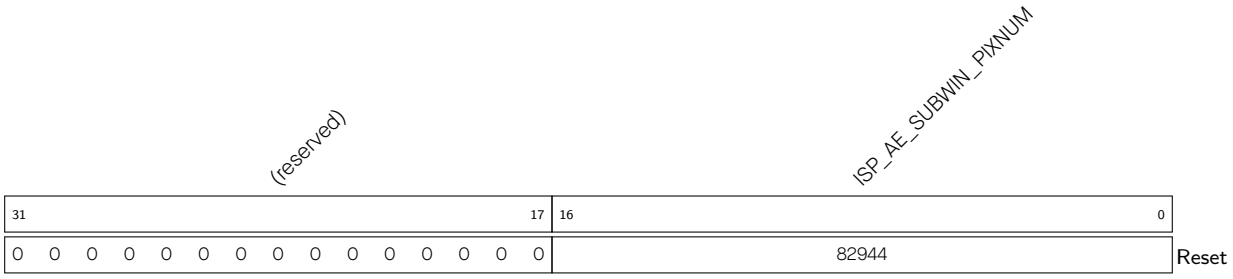
Register 38.45. ISP_AE_BY_REG (0x00CC)



ISP_AE_Y_BSIZE Configures the vertical dimension of each sub-window. (R/W)

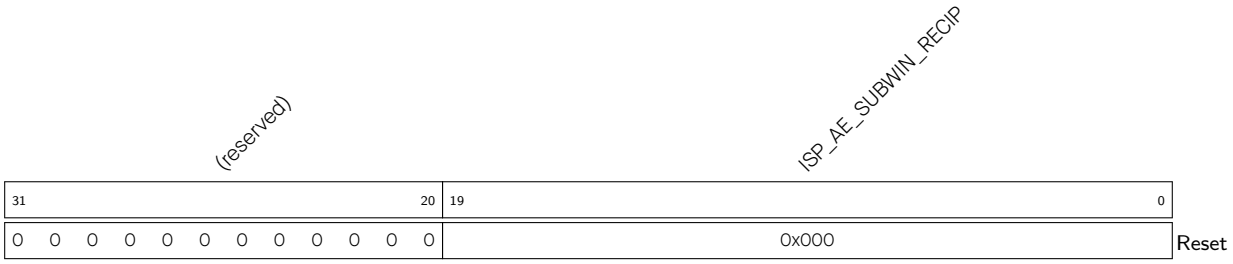
ISP_AE_Y_START Configures the start coordinate of the AE statistical window in the vertical direction. (R/W)

Register 38.46. ISP_AE_WINPIXNUM_REG (0x00D0)



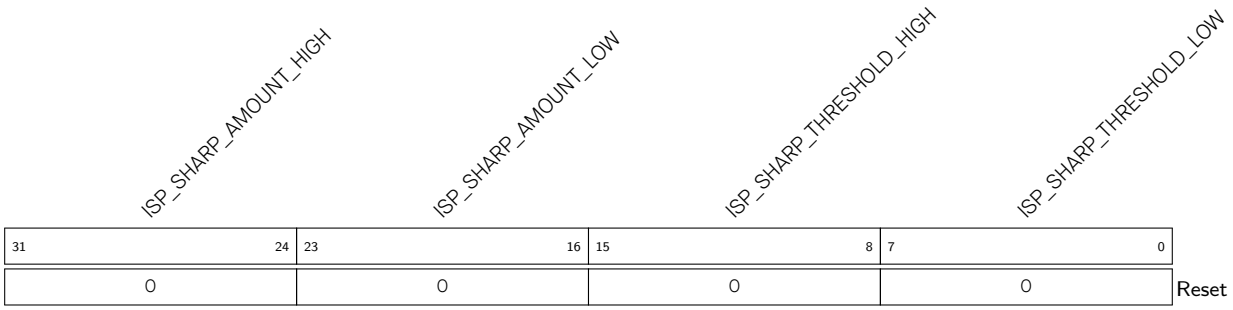
ISP_AE_SUBWIN_PIXNUM Configures the number of pixels in each sub-window. (R/W)

Register 38.47. ISP_AE_WIN_RECIPROCAL_REG (0x00D4)



ISP_AE_SUBWIN_RECIP Configures the reciprocal of the number of pixels in each sub-window. It is a 20-bit fractional number, with [19:0] representing the fractional part. (R/W)

Register 38.48. ISP_SHARP_CTRL0_REG (0x00F4)



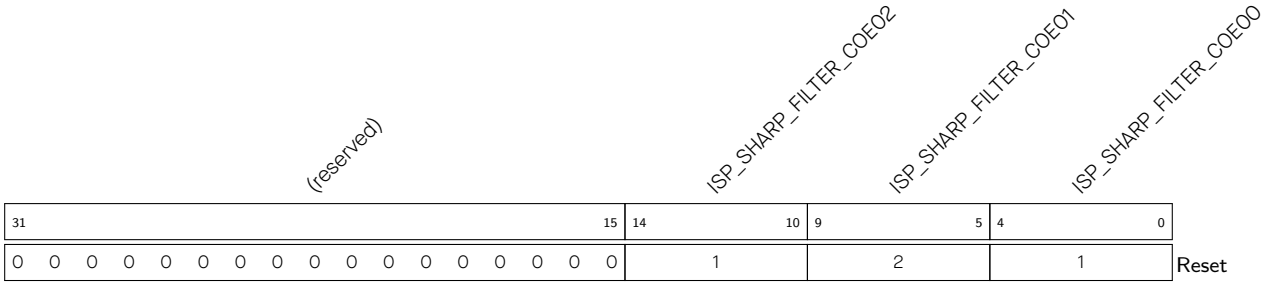
ISP_SHARP_THRESHOLD_LOW Configures the threshold for sharpening high-frequency details. Refer to section 38.5.2.10 for configuration. (R/W)

ISP_SHARP_THRESHOLD_HIGH Configures the threshold for sharpening high-frequency edges. Refer to section 38.5.2.10 for configuration. (R/W)

ISP_SHARP_AMOUNT_LOW Configures the gain for sharpening high-frequency details. Refer to section 38.5.2.10 for configuration. (R/W)

ISP_SHARP_AMOUNT_HIGH Configures the gain for sharpening high-frequency edges. Refer to section 38.5.2.10 for configuration. (R/W)

Register 38.49. ISP_SHARP_FILTER0_REG (0x00F8)

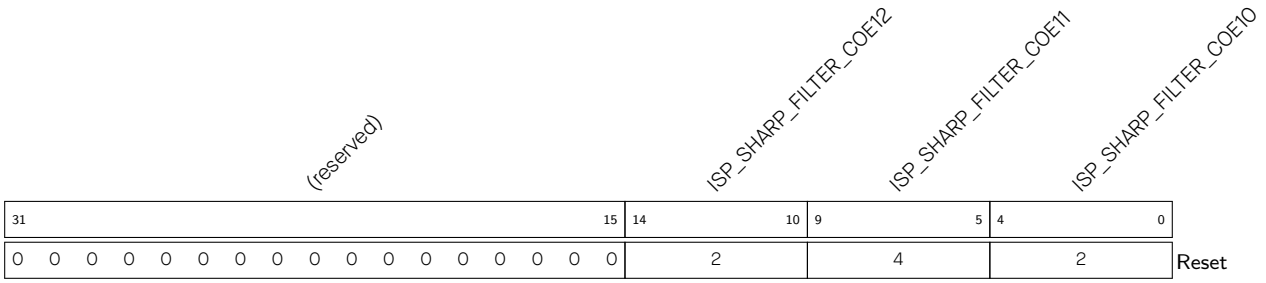


ISP_SHARP_FILTER_COE00 Configures the value of coordinate 00 in the low-pass filter template of the sharpen module. (R/W)

ISP_SHARP_FILTER_COE01 Configures the value of coordinate 01 in the low-pass filter template of the sharpen module. (R/W)

ISP_SHARP_FILTER_COE02 Configures the value of coordinate 02 in the low-pass filter template of the sharpen module. (R/W)

Register 38.50. ISP_SHARP_FILTER1_REG (0x00FC)

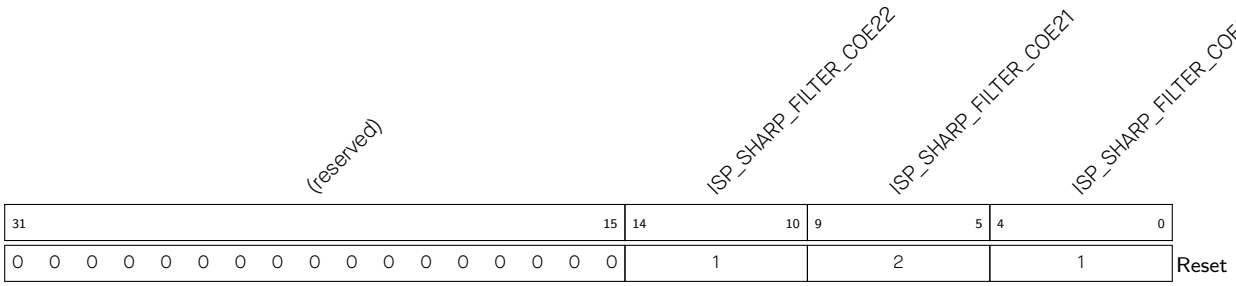


ISP_SHARP_FILTER_COE10 Configures the value of coordinate 10 in the low-pass filter template of the sharpen module. (R/W)

ISP_SHARP_FILTER_COE11 Configures the value of coordinate 11 in the low-pass filter template of the sharpen module. (R/W)

ISP_SHARP_FILTER_COE12 Configures the value of coordinate 12 in the low-pass filter template of the sharpen module. (R/W)

Register 38.51. ISP_SHARP_FILTER2_REG (0x0100)

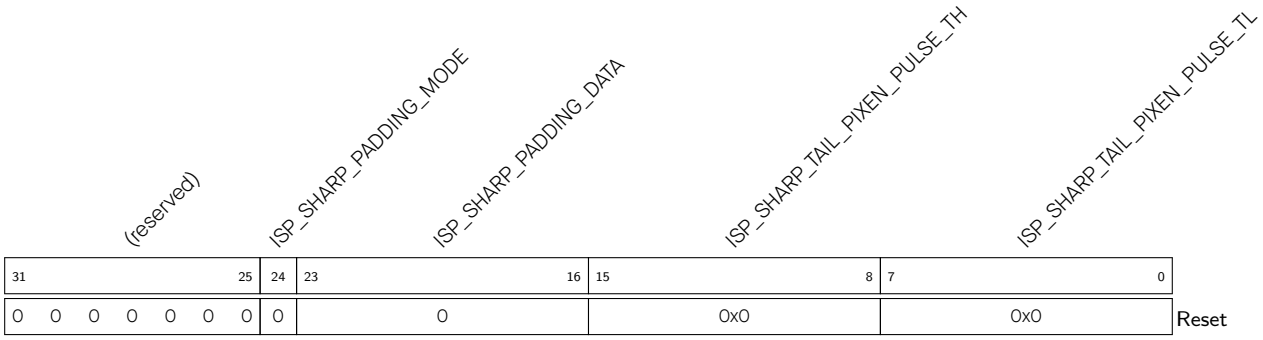


ISP_SHARP_FILTER_COE20 Configures the value of coordinate 20 in the low-pass filter template of the sharpen module. (R/W)

ISP_SHARP_FILTER_COE21 Configures the value of coordinate 21 in the low-pass filter template of the sharpen module. (R/W)

ISP_SHARP_FILTER_COE22 Configures the value of coordinate 22 in the low-pass filter template of the sharpen module. (R/W)

Register 38.52. ISP_SHARP_MATRIX_CTRL_REG (0x0104)



ISP_SHARP_TAIL_PIXEN_PULSE_TL Configures the data control cycle for the tail-row data rate during pixel-to-matrix conversion in the sharpen module.

Within ISP_SHARP_TAIL_PIXEN_PULSE_TL clock cycles, only the first ISP_SHARP_TAIL_PIXEN_PULSE_TH cycles are valid. Be cautious to configure this field, to prevent data processing from being deferred to the next frame.

This feature is enabled only when both ISP_SHARP_TAIL_PIXEN_PULSE_TL and ISP_SHARP_TAIL_PIXEN_PULSE_TH are non-zero, and ISP_SHARP_TAIL_PIXEN_PULSE_TH < ISP_SHARP_TAIL_PIXEN_PULSE_TL. (R/W)

ISP_SHARP_TAIL_PIXEN_PULSE_TH Configures the valid data cycle of the tail-row data rate during pixel-to-matrix conversion in the sharpen module, which must be less than ISP_HADR_NUM - 1. (R/W)

ISP_SHARP_PADDING_DATA Configures the padding data used during image edge expansion in pixel-to-matrix conversion of the sharpen module. (R/W)

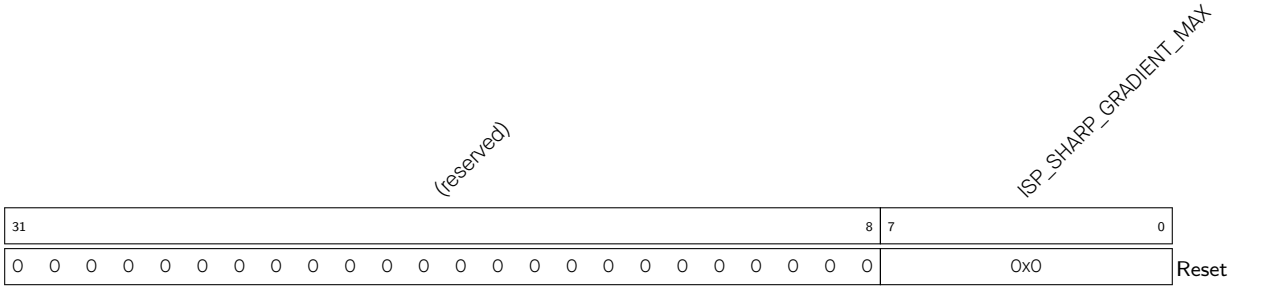
ISP_SHARP_PADDING_MODE Configures the method of image edge expansion during pixel-to-matrix conversion in the sharpen module.

0: Automatically padded with pixel values from the image edge

1: Padded with the data in ISP_SHARP_PADDING_DATA

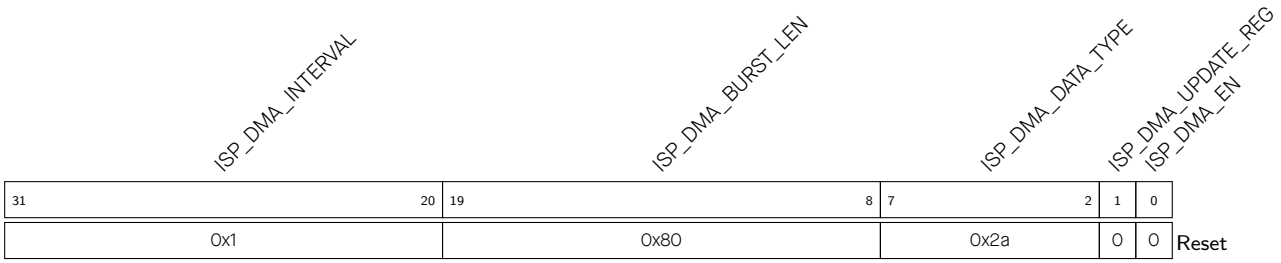
(R/W)

Register 38.53. ISP_SHARP_CTRL1_REG (0x0108)



ISP_SHARP_GRADIENT_MAX Represents the maximum pixel value of high-frequency components in the sharpen module, automatically refreshed every frame. (RO)

Register 38.54. ISP_DMA_CNTL_REG (0x010C)



ISP_DMA_EN Configures whether to trigger VDMA transfer of one frame of data.

0: Not trigger

1: Trigger

(WT)

ISP_DMA_UPDATE_REG Configures whether to update the configuration for ISP_DMA_DATA_TYPE, ISP_DMA_BURST_LEN, and ISP_DMA_INTERVAL.

0: Not update

1: Update, will be automatically cleared to 0 after the update is complete

(R/W)

ISP_DMA_DATA_TYPE Configures the Image Interface data_type for the transferred data.

0x2A: RAW8

0x2B: RAW10

0x2C: RAW12

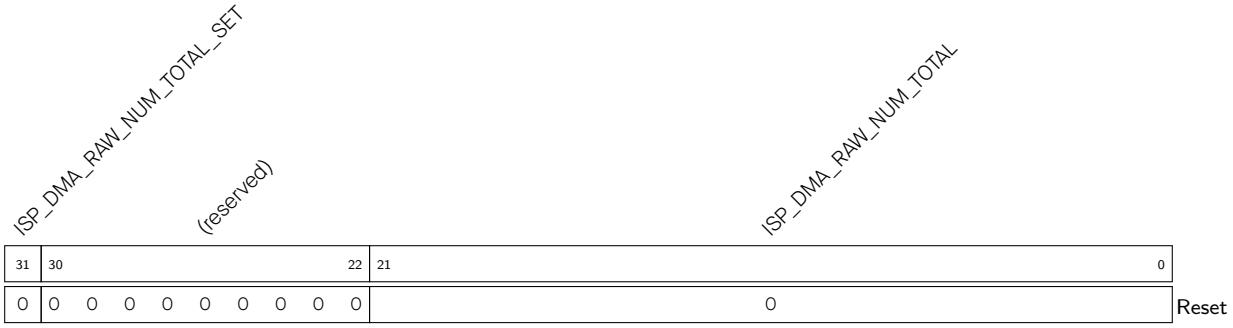
Others: Invalid

(R/W)

ISP_DMA_BURST_LEN Configures the burst length for one VDMA transfer, which needs to align with the VDMA configuration. (R/W)

ISP_DMA_INTERVAL Configures the interval of VDMA requests. 12'b1: 1 clock cycle, 12'b11: 2 clock cycles, etc. (R/W)

Register 38.55. ISP_DMA_RAW_DATA_REG (0x0110)



ISP_DMA_RAW_NUM_TOTAL Configures the amount of 64-bit data in a frame of image. (R/W)

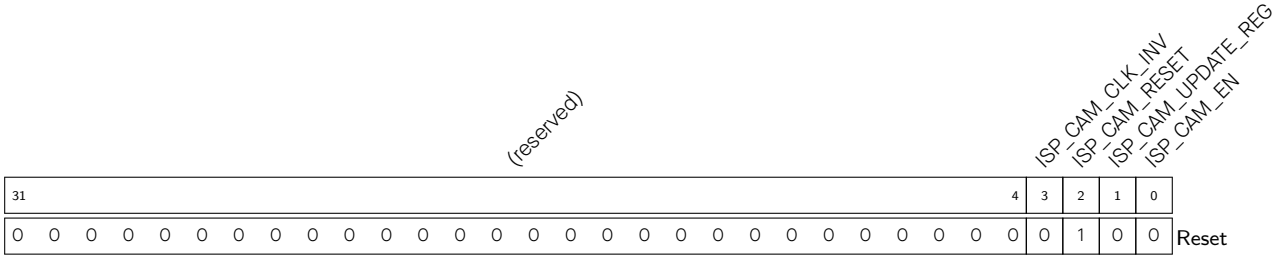
ISP_DMA_RAW_NUM_TOTAL_SET Configures whether to update the configuration for ISP_DMA_RAW_NUM_TOTAL.

0: Not update

1: Update

(WT)

Register 38.56. ISP_CAM_CNTL_REG (0x0114)



ISP_CAM_EN Configures whether to enable the input via the DVP interface.

0: Disable

1: Enable

(R/W)

ISP_CAM_UPDATE_REG Configures whether to update the configuration for ISP_CAM_CONF_REG.

0: Not update

1: Update

(R/W)

ISP_CAM_RESET Configures whether to reset the DVP input configuration.

0: Not reset

1: Reset

(R/W)

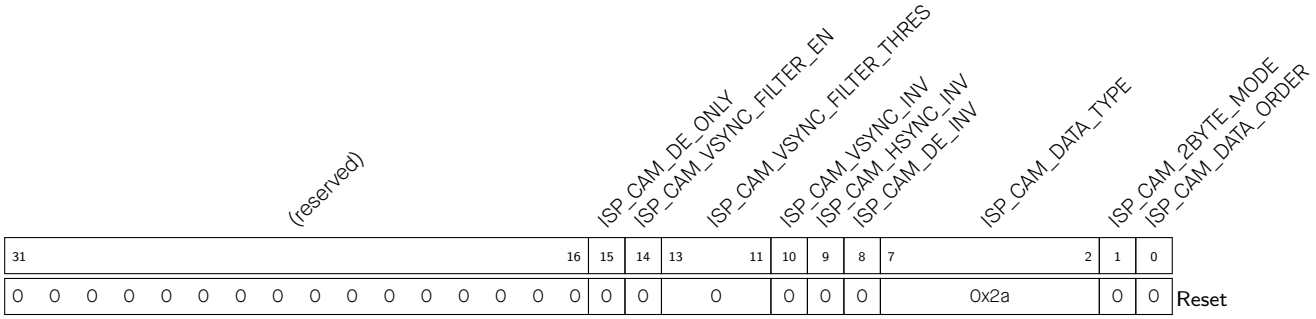
ISP_CAM_CLK_INV Configures whether to invert the DVP clock.

0: Not invert

1: Invert

(R/W)

Register 38.57. ISP_CAM_CONF_REG (0x0118)



ISP_CAM_DATA_ORDER Configures the order of DVP input data.

0: cam_data_in

1: cam_data_in[7:0], cam_data_in[15:8]

(R/W)

ISP_CAM_2BYTE_MODE Configures whether to enable the 2-byte mode.

0: Disable

1: Enable

(R/W)

ISP_CAM_DATA_TYPE Configures the format of DVP input data.

0x2a: RAW8

0x2b: RAW10

0x2c: RAW12

(R/W)

ISP_CAM_DE_INV Configures whether to invert the DVP de signal.

0: Not invert

1: Invert

(R/W)

ISP_CAM_HSYNC_INV Configures whether to invert the DVP hsync signal.

0: Not invert

1: Invert

(R/W)

ISP_CAM_VSYNC_INV Configures whether to invert the DVP vsync signal.

0: Not invert

1: Invert

(R/W)

ISP_CAM_VSYNC_FILTER_THRES Configures vsync filtering. Vsyncs shorter than this length will be filtered out. (R/W)

Continued on the next page...

Register 38.57. ISP_CAM_CONF_REG (0x0118)

Continued from the previous page...

ISP_CAM_VSYNC_FILTER_EN Configures whether to enable the vsync filtering.

0: Disable

1: Enable

(R/W)

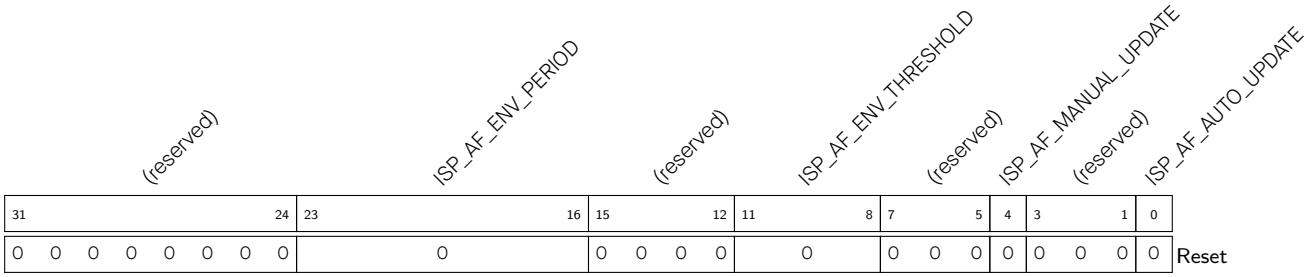
ISP_CAM_DE_ONLY Configures whether the DVP interface uses only DE data without HSYNC data.

0: HSYNC is present

1: HSYNC is not present

(R/W)

Register 38.58. ISP_AF_CTRL0_REG (0x011C)



ISP_AF_AUTO_UPDATE Configures whether to enable AF automatic statistics.

0: Disable automatic statistics, and manual trigger is required

1: Enable automatic statistics, and AF will perform statistics for every frame with the AF module enabled

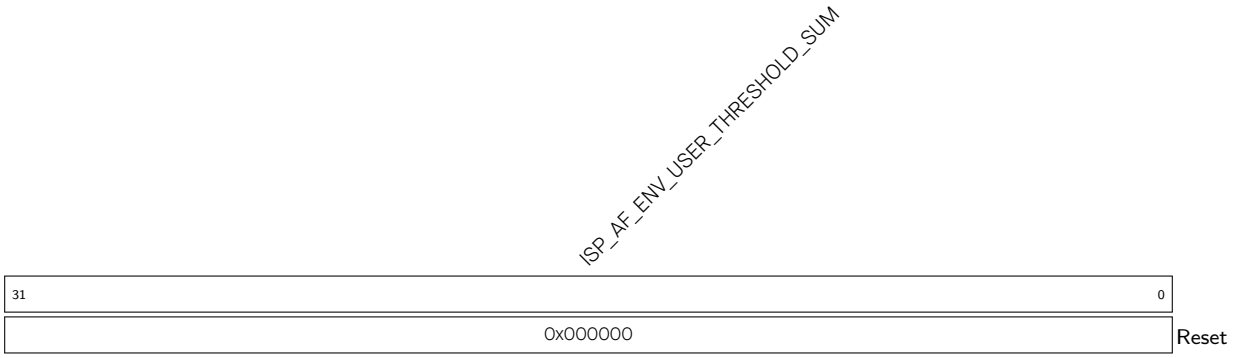
(R/W)

ISP_AF_MANUAL_UPDATE Configures AF manual statistics. With AF automatic statistics disabled, writing 1 triggers a statistics collection. (WT)

ISP_AF_ENV_THRESHOLD Configures the threshold for AF scene monitoring. If ISP_AF_ENV_USER_THRESHOLD_SUM or ISP_AF_ENV_USER_THRESHOLD_LUM is 0, an interrupt for scene monitoring will be triggered when the changes in sharpness or luminance exceed the product of the baseline value and this field for two consecutive times. 4 fractional bits. (R/W)

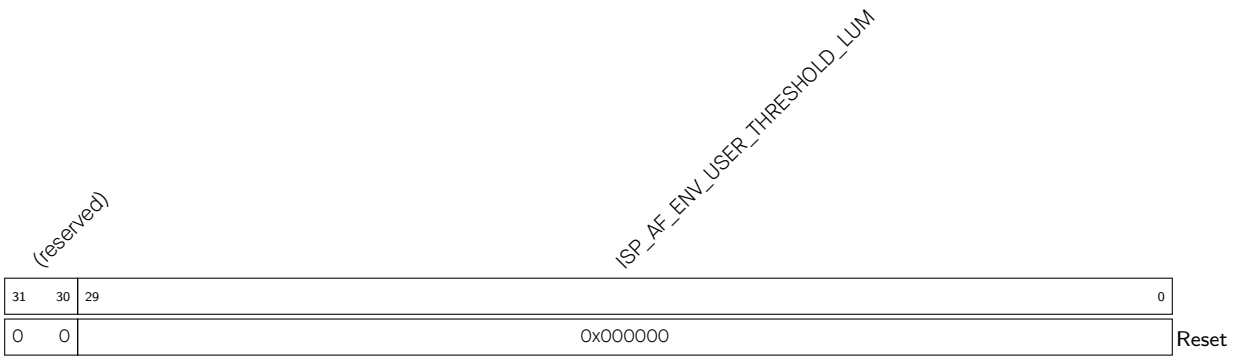
ISP_AF_ENV_PERIOD Configures the statistical interval frames for AF scene monitoring. Setting this filed to 0 disables the scene monitoring. (R/W)

Register 38.59. ISP_AF_ENV_USER_TH_SUM_REG (0x0128)



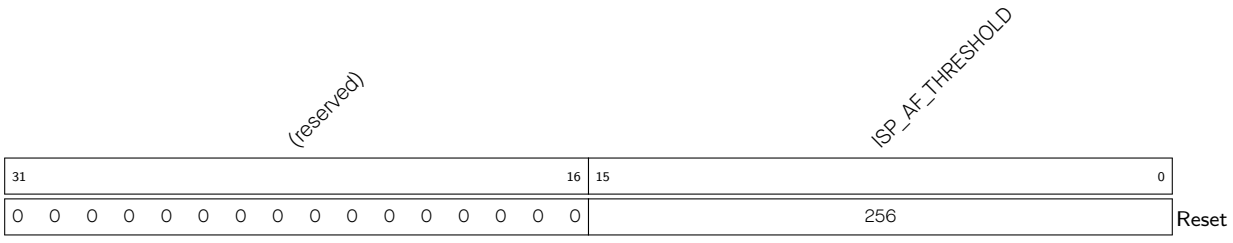
ISP_AF_ENV_USER_THRESHOLD_SUM Configures the threshold for sharpness change in AF scene monitoring. (R/W)

Register 38.60. ISP_AF_ENV_USER_TH_LUM_REG (0x012C)



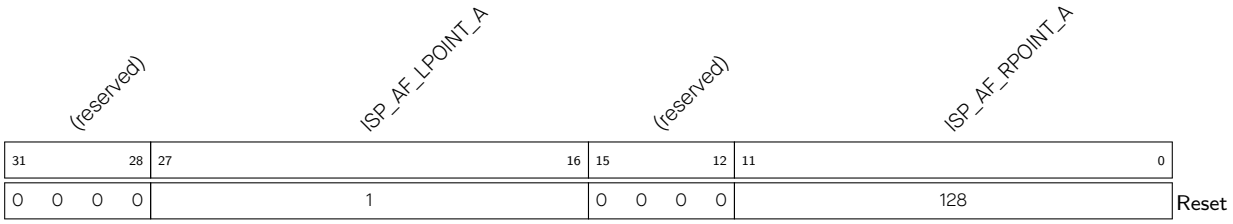
ISP_AF_ENV_USER_THRESHOLD_LUM Configures the threshold for luminance change in AF scene monitoring. (R/W)

Register 38.61. ISP_AF_THRESHOLD_REG (0x0130)



ISP_AF_THRESHOLD Configures the sharpness threshold for AF statistics. (R/W)

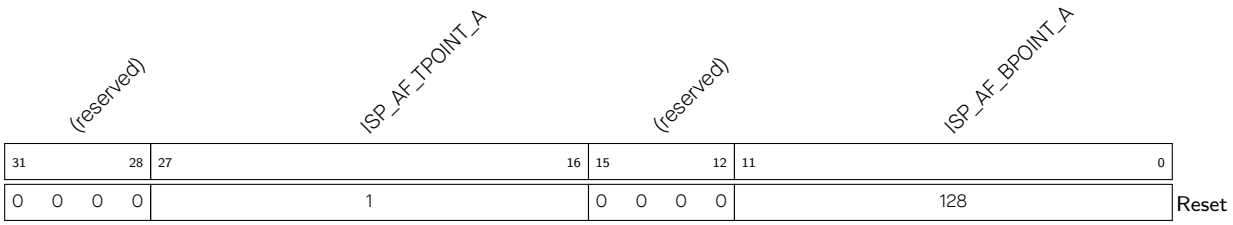
Register 38.62. ISP_AF_HSCALE_A_REG (0x0134)



ISP_AF_RPOINT_A Configures the right boundary coordinate of AF statistical window A, recommended to be less than HNUM-1. (R/W)

ISP_AF_LPOINT_A Configures the left boundary coordinate of AF statistical window A, recommended to be greater than or equal to 2. (R/W)

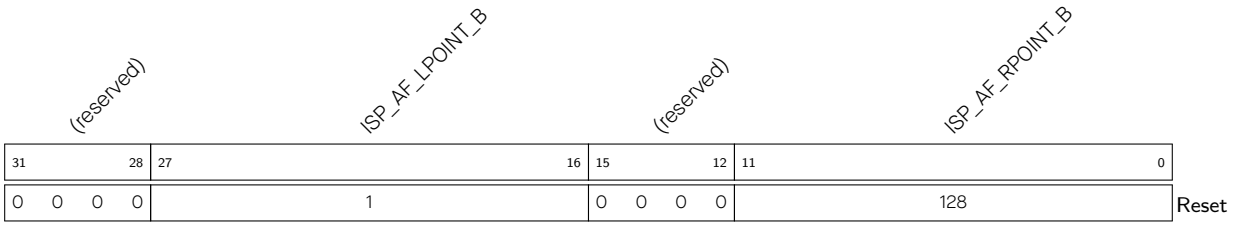
Register 38.63. ISP_AF_VSCALE_A_REG (0x0138)



ISP_AF_BPOINT_A Configures the bottom boundary coordinate of AF statistical window A, recommended to be less than HNUM-1. (R/W)

ISP_AF_TPOINT_A Configures the top boundary coordinate of AF statistical window A, recommended to be greater than or equal to 2. (R/W)

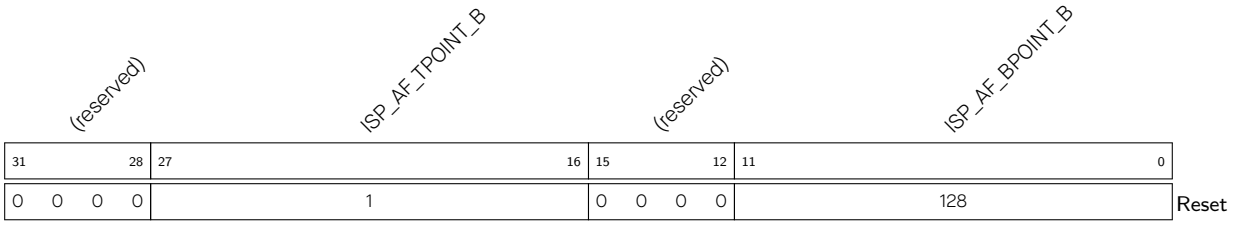
Register 38.64. ISP_AF_HSCALE_B_REG (0x013C)



ISP_AF_RPOINT_B Configures the right boundary coordinate of AF statistical window B, recommended to be less than HNUM-1. (R/W)

ISP_AF_LPOINT_B Configures the left boundary coordinate of AF statistical window B, recommended to be greater than or equal to 2. (R/W)

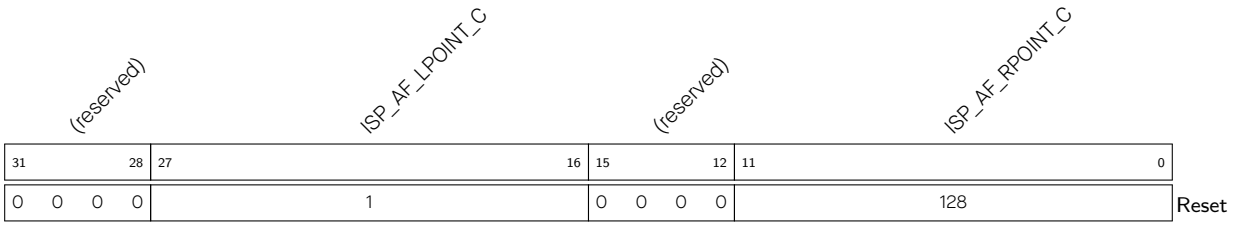
Register 38.65. ISP_AF_VSCALE_B_REG (0x0140)



ISP_AF_BPOINT_B Configures the bottom boundary coordinate of AF statistical window B, recommended to be less than HNUM-1. (R/W)

ISP_AF_TPOINT_B Configures the top boundary coordinate of AF statistical window B, recommended to be greater than or equal to 2. (R/W)

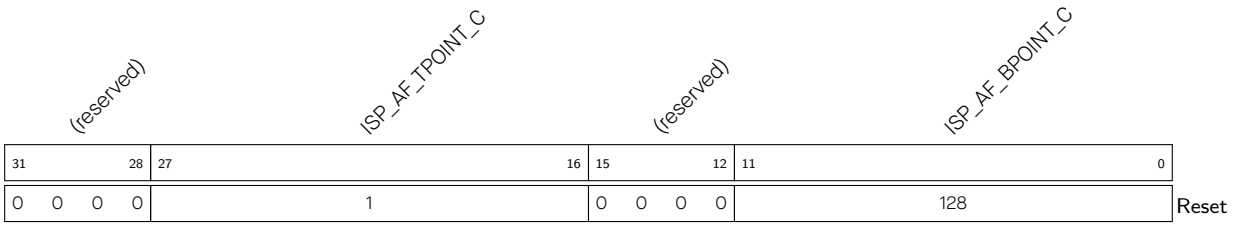
Register 38.66. ISP_AF_HSCALE_C_REG (0x0144)



ISP_AF_RPOINT_C Configures the right boundary coordinate of AF statistical window C, recommended to be less than HNUM-1. (R/W)

ISP_AF_LPOINT_C Configures the left boundary coordinate of AF statistical window C, recommended to be greater than or equal to 2. (R/W)

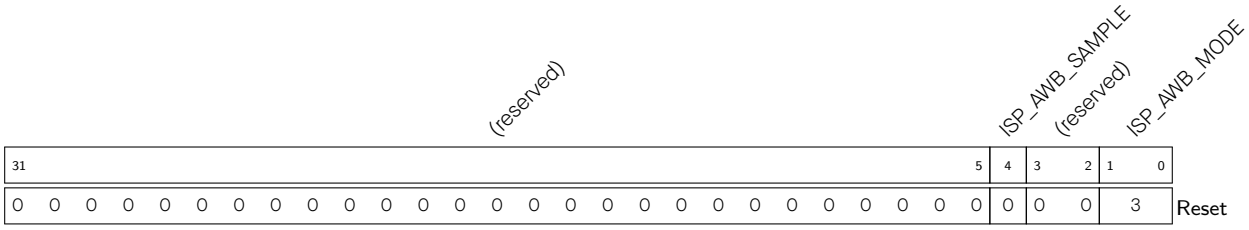
Register 38.67. ISP_AF_VSCALE_C_REG (0x0148)



ISP_AF_BPOINT_C Configures the bottom boundary coordinate of AF statistical window C, recommended to be less than HNUM-1. (R/W)

ISP_AF_TPOINT_C Configures the top boundary coordinate of AF statistical window C, recommended to be greater than or equal to 2. (R/W)

Register 38.68. ISP_AWB_MODE_REG (0x0164)



ISP_AWB_MODE Configures whether to enable the AWB algorithm.

1: Enable

Others: Invalid

(R/W)

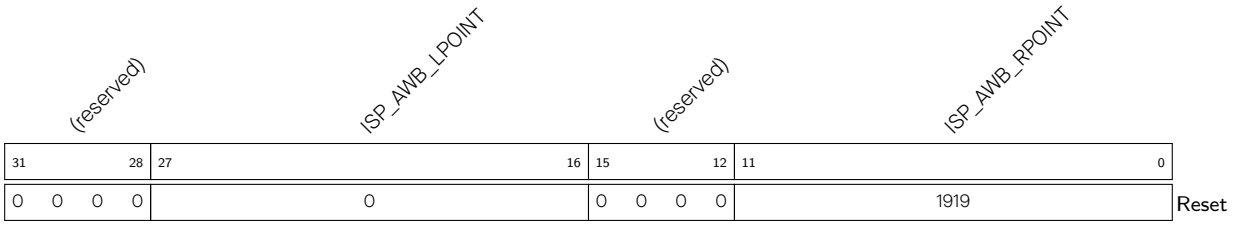
ISP_AWB_SAMPLE Configures the AWB sampling point.

0: Use data output before processed by CCM

1: Use data output after processed by CCM

(R/W)

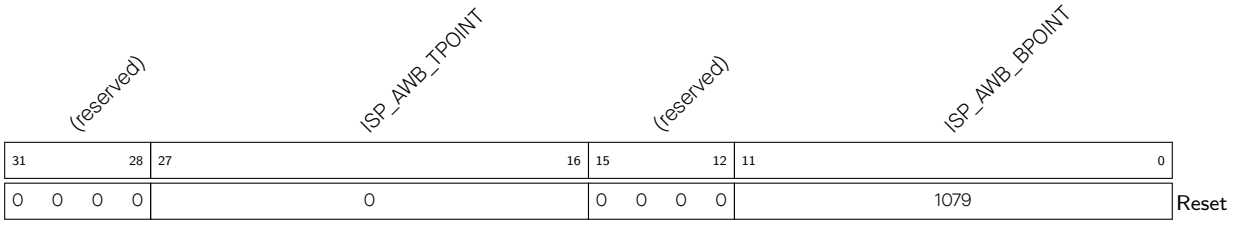
Register 38.69. ISP_AWB_HSCALE_REG (0x0168)



ISP_AWB_RPOINT Configures the right boundary coordinate of the AWB statistical window. (R/W)

ISP_AWB_LPOINT Configures the left boundary coordinate of the AWB statistical window. (R/W)

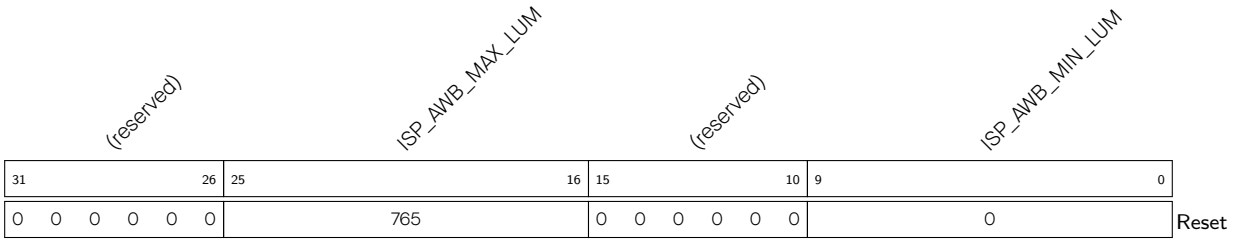
Register 38.70. ISP_AWB_VSCALE_REG (0x016C)



ISP_AWB_BPOINT Configures the bottom boundary coordinate of the AWB statistical window. (R/W)

ISP_AWB_TPOINT Configures the top boundary coordinate of the AWB statistical window. (R/W)

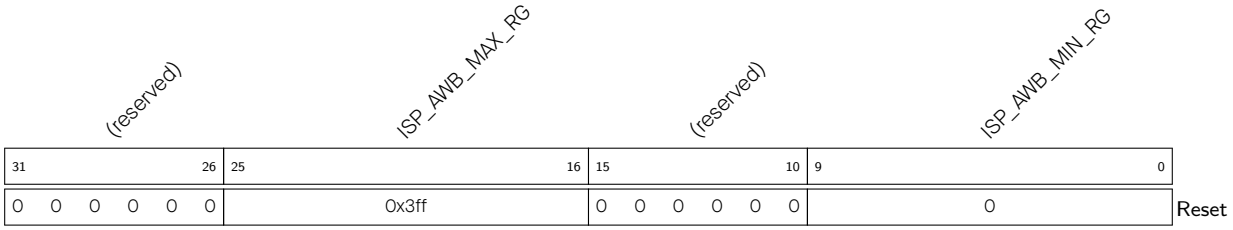
Register 38.71. ISP_AWB_TH_LUM_REG (0x0170)



ISP_AWB_MIN_LUM Configures the lower limit of R+G+B luminance for AWB white patch filtering. (R/W)

ISP_AWB_MAX_LUM Configures the upper limit of R+G+B luminance for AWB white patch filtering. (R/W)

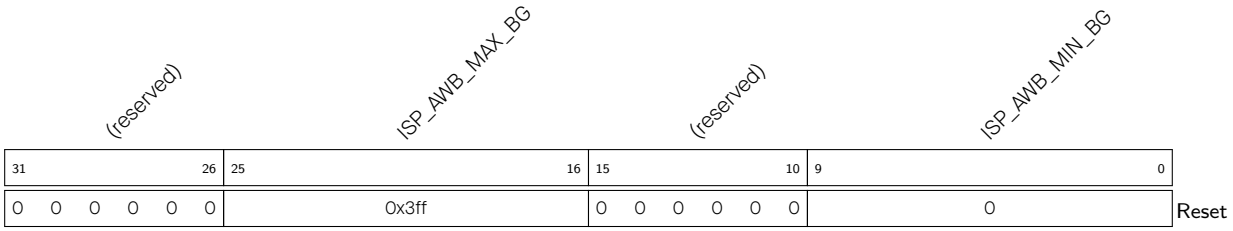
Register 38.72. ISP_AWB_TH_RG_REG (0x0174)



ISP_AWB_MIN_RG Configures the lower limit of R/G ratio for AWB white point filtering. Bits [9:8] are the integer part, and bits [7:0] are the fractional part. (R/W)

ISP_AWB_MAX_RG Configures the upper limit of R/G ratio for AWB white point filtering. Bits [9:8] are the integer part, and bits [7:0] are the fractional part. (R/W)

Register 38.73. ISP_AWB_TH_BG_REG (0x0178)



ISP_AWB_MIN_BG Configures the lower limit of B/G ratio for AWB white point filtering. Bits [9:8] are the integer part, and bits [7:0] are the fractional part. (R/W)

ISP_AWB_MAX_BG Configures the upper limit of B/G ratio for AWB white point filtering. Bits [9:8] are the integer part, and bits [7:0] are the fractional part. (R/W)

Register 38.74. ISP_COLOR_CTRL_REG (0x018C)



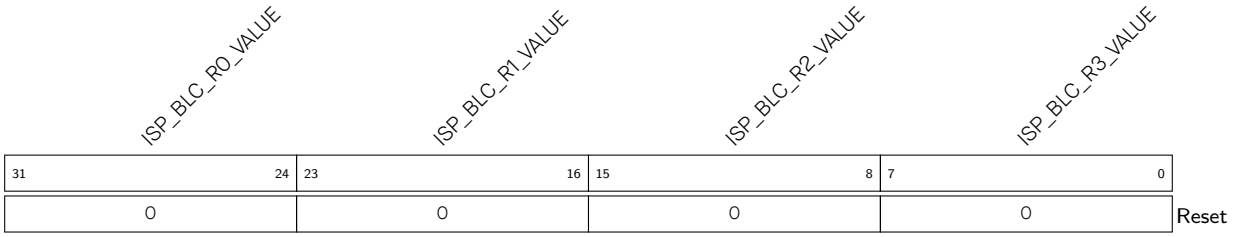
ISP_COLOR_SATURATION Configures the saturation. Bit [7] is the integer part, and bits [6:0] are the fractional part. (R/W)

ISP_COLOR_HUE Configures the lower 8 bits of the hue value. The hue value is 9 bits in total, with the MSB provided by ISP_COLOR_HUE_H. Valid range: 0–359. (R/W)

ISP_COLOR_CONTRAST Configures the saturation. Bit [7] is the integer part, and bits [6:0] are the fractional part. (R/W)

ISP_COLOR_BRIGHTNESS Configures the brightness in two's complement. (R/W)

Register 38.75. ISP_BLC_VALUE_REG (0x0190)



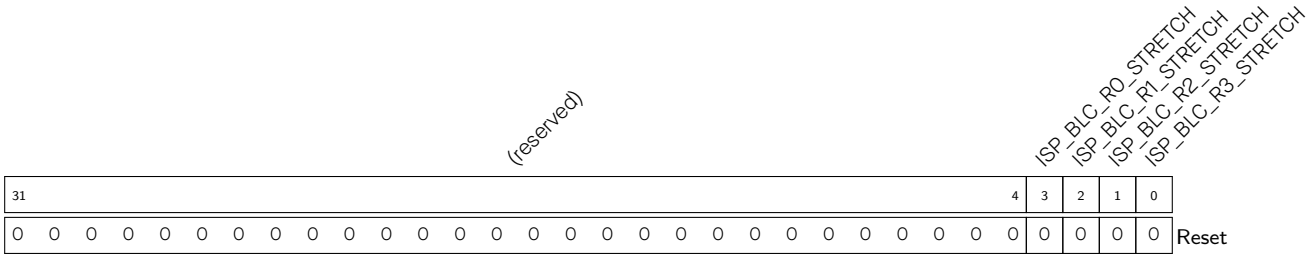
ISP_BLC_R3_VALUE Configures the black level offset of channel 3 in the Bayer image. (R/W)

ISP_BLC_R2_VALUE Configures the black level offset of channel 2 in the Bayer image. (R/W)

ISP_BLC_R1_VALUE Configures the black level offset of channel 1 in the Bayer image. (R/W)

ISP_BLC_R0_VALUE Configures the black level offset of channel 0 in the Bayer image. (R/W)

Register 38.76. ISP_BLC_CTRL0_REG (0x0194)



ISP_BLC_R3_STRETCH Configures whether to stretch the pixel range of channel 3.

0: Stretch

1: Not stretch

(R/W)

ISP_BLC_R2_STRETCH Configures whether to stretch the pixel range of channel 2.

0: Stretch

1: Not stretch

(R/W)

ISP_BLC_R1_STRETCH Configures whether to stretch the pixel range of channel 1.

0: Stretch

1: Not stretch

(R/W)

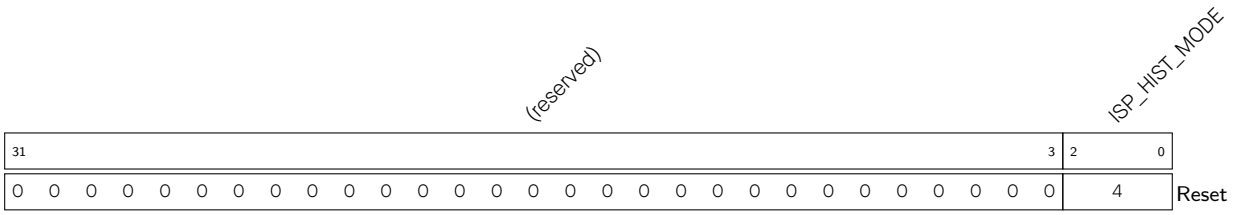
ISP_BLC_R0_STRETCH Configures whether to stretch the pixel range of channel 0.

0: Stretch

1: Not stretch

(R/W)

Register 38.77. ISP_HIST_MODE_REG (0x01A4)



ISP_HIST_MODE Configures the HIST sampling mode.

0: RAW_B

1: RAW_GB

2: RAW_GR

3: RAW_R

4: RGB

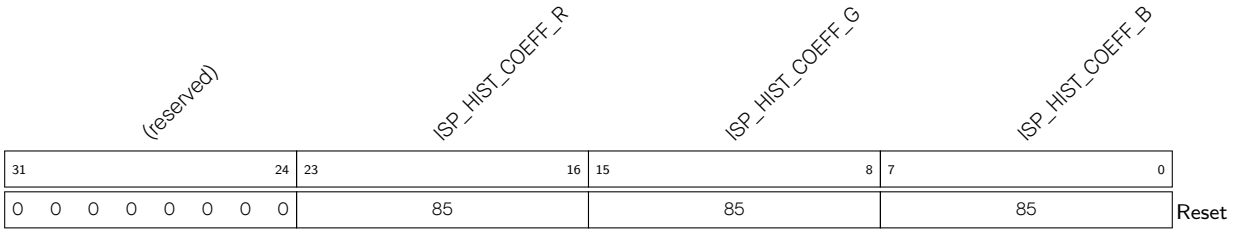
5: YUV_Y

6: YUV_U

7: YUV_V

(R/W)

Register 38.78. ISP_HIST_COEFF_REG (0x01A8)

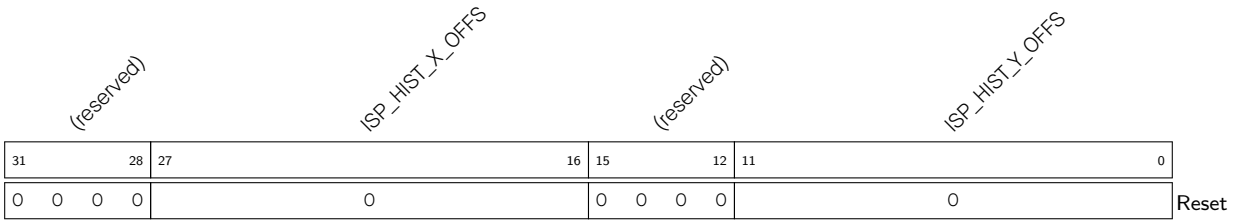


ISP_HIST_COEFF_B Configures the B weight for RGB to brightness conversion when HIST statistics mode set to RGB. Bits [7:0] are the fractional part. (R/W)

ISP_HIST_COEFF_G Configures the G weight for RGB to brightness conversion when HIST statistics mode set to RGB. Bits [7:0] are the fractional part. (R/W)

ISP_HIST_COEFF_R Configures the R weight for RGB to brightness conversion when HIST statistics mode set to RGB. Bits [7:0] are the fractional part. (R/W)

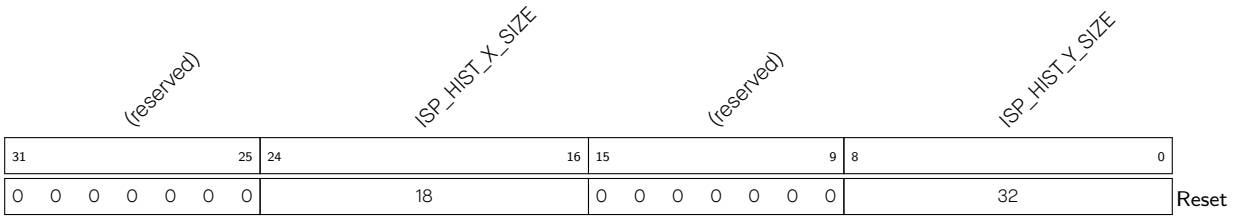
Register 38.79. ISP_HIST_OFFS_REG (0x01AC)



ISP_HIST_Y_OFFS Configures the start coordinate of the HIST statistics window in the vertical direction. (R/W)

ISP_HIST_X_OFFS Configures the start coordinate of the HIST statistics window in the horizontal direction. (R/W)

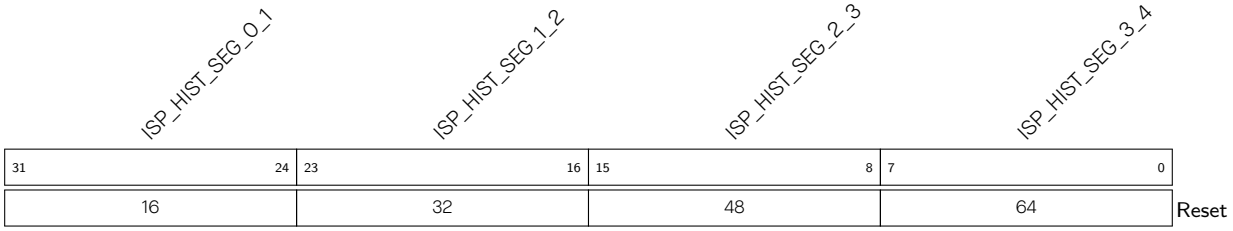
Register 38.80. ISP_HIST_SIZE_REG (0x01B0)



ISP_HIST_Y_SIZE Configures the vertical dimension of the HIST sub-window. (R/W)

ISP_HIST_X_SIZE Configures the horizontal dimension of the HIST sub-window. (R/W)

Register 38.81. ISP_HIST_SEG0_REG (0x01B4)



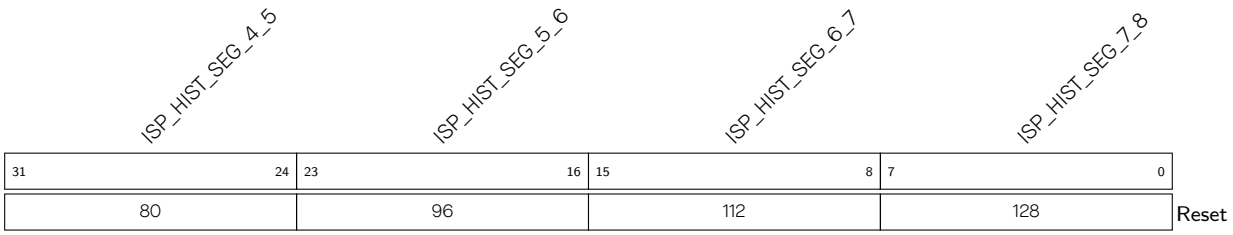
ISP_HIST_SEG_3_4 Configures the threshold between the HIST interval 3 and 4. (R/W)

ISP_HIST_SEG_2_3 Configures the threshold between the HIST interval 2 and 3. (R/W)

ISP_HIST_SEG_1_2 Configures the threshold between the HIST interval 1 and 2. (R/W)

ISP_HIST_SEG_0_1 Configures the threshold between the HIST interval 0 and 1. (R/W)

Register 38.82. ISP_HIST_SEG1_REG (0x01B8)



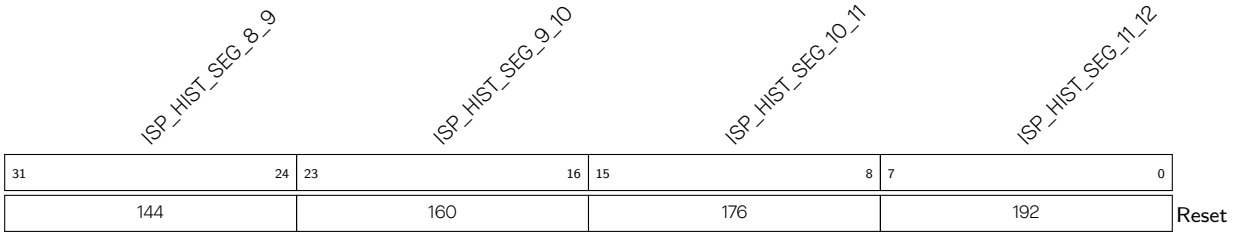
ISP_HIST_SEG_7_8 Configures the threshold between the HIST interval 7 and 8. (R/W)

ISP_HIST_SEG_6_7 Configures the threshold between the HIST interval 6 and 7. (R/W)

ISP_HIST_SEG_5_6 Configures the threshold between the HIST interval 5 and 6. (R/W)

ISP_HIST_SEG_4_5 Configures the threshold between the HIST interval 4 and 5. (R/W)

Register 38.83. ISP_HIST_SEG2_REG (0x01BC)



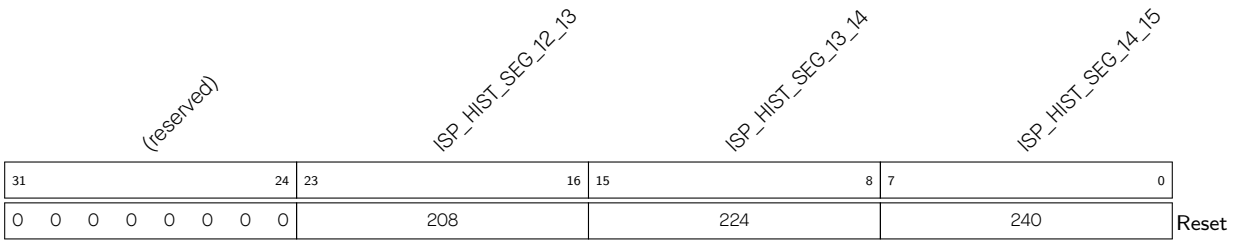
ISP_HIST_SEG_11_12 Configures the threshold between the HIST interval 11 and 12. (R/W)

ISP_HIST_SEG_10_11 Configures the threshold between the HIST interval 10 and 11. (R/W)

ISP_HIST_SEG_9_10 Configures the threshold between the HIST interval 9 and 10. (R/W)

ISP_HIST_SEG_8_9 Configures the threshold between the HIST interval 8 and 9. (R/W)

Register 38.84. ISP_HIST_SEG3_REG (0x01C0)

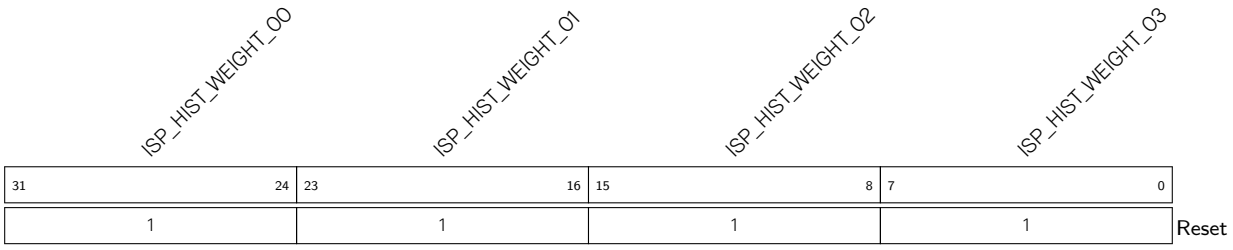


ISP_HIST_SEG_14_15 Configures the threshold between the HIST interval 14 and 15. (R/W)

ISP_HIST_SEG_13_14 Configures the threshold between the HIST interval 13 and 14. (R/W)

ISP_HIST_SEG_12_13 Configures the threshold between the HIST interval 12 and 13. (R/W)

Register 38.85. ISP_HIST_WEIGHT0_REG (0x01C4)



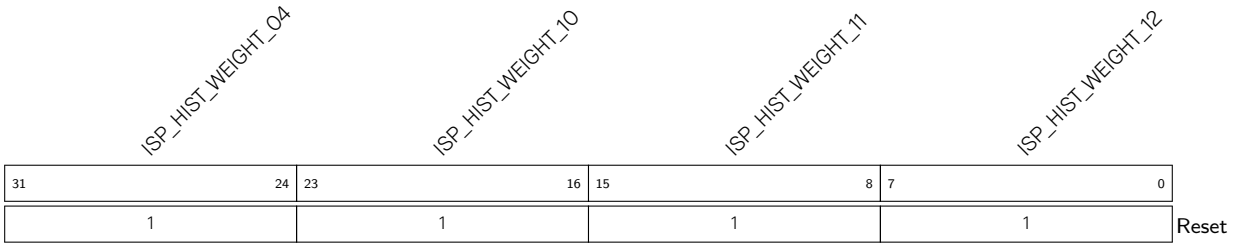
ISP_HIST_WEIGHT_03 Configures the weight of HIST sub-window 03. Bits [7:0] are the fractional part. (R/W)

ISP_HIST_WEIGHT_02 Configures the weight of HIST sub-window 02. Bits [7:0] are the fractional part. (R/W)

ISP_HIST_WEIGHT_01 Configures the weight of HIST sub-window 01. Bits [7:0] are the fractional part. (R/W)

ISP_HIST_WEIGHT_00 Configures the weight of HIST sub-window 00. Bits [7:0] are the fractional part. (R/W)

Register 38.86. ISP_HIST_WEIGHT1_REG (0x01C8)



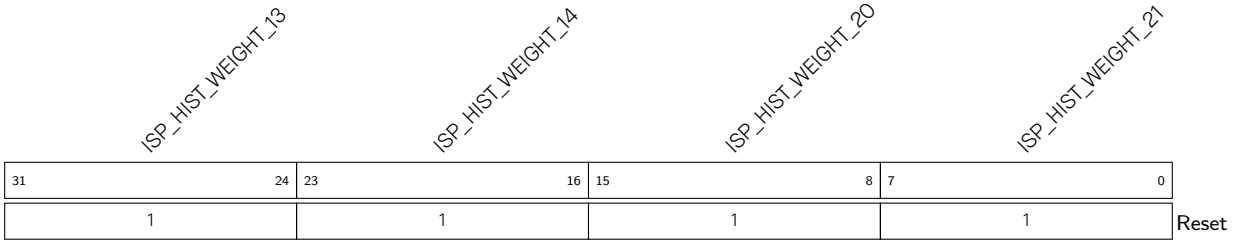
ISP_HIST_WEIGHT_12 Configures the weight of HIST sub-window 12. Bits [7:0] are the fractional part. (R/W)

ISP_HIST_WEIGHT_11 Configures the weight of HIST sub-window 11. Bits [7:0] are the fractional part. (R/W)

ISP_HIST_WEIGHT_10 Configures the weight of HIST sub-window 10. Bits [7:0] are the fractional part. (R/W)

ISP_HIST_WEIGHT_04 Configures the weight of HIST sub-window 04. Bits [7:0] are the fractional part. (R/W)

Register 38.87. ISP_HIST_WEIGHT2_REG (0x01CC)



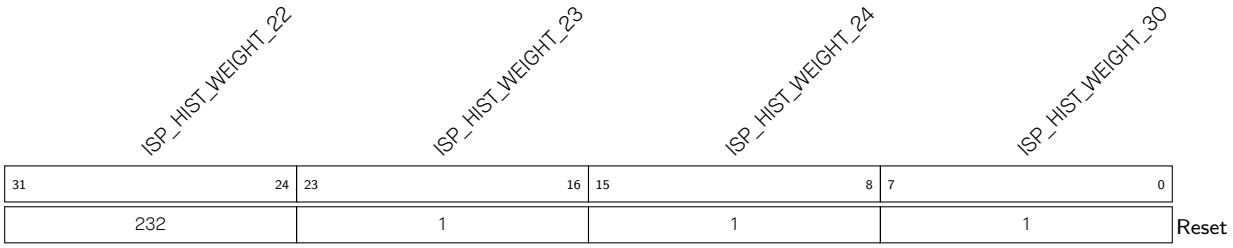
ISP_HIST_WEIGHT_21 Configures the weight of HIST sub-window 21. Bits [7:0] are the fractional part. (R/W)

ISP_HIST_WEIGHT_20 Configures the weight of HIST sub-window 20. Bits [7:0] are the fractional part. (R/W)

ISP_HIST_WEIGHT_14 Configures the weight of HIST sub-window 14. Bits [7:0] are the fractional part. (R/W)

ISP_HIST_WEIGHT_13 Configures the weight of HIST sub-window 13. Bits [7:0] are the fractional part. (R/W)

Register 38.88. ISP_HIST_WEIGHT3_REG (0x01D0)



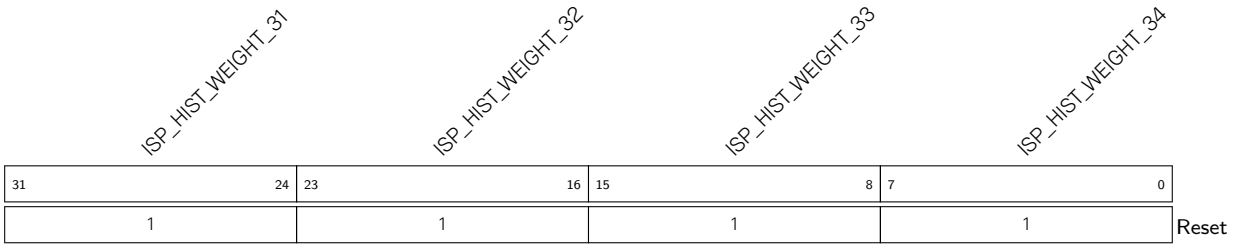
ISP_HIST_WEIGHT_30 Configures the weight of HIST sub-window 30. Bits [7:0] are the fractional part. (R/W)

ISP_HIST_WEIGHT_24 Configures the weight of HIST sub-window 24. Bits [7:0] are the fractional part. (R/W)

ISP_HIST_WEIGHT_23 Configures the weight of HIST sub-window 23. Bits [7:0] are the fractional part. (R/W)

ISP_HIST_WEIGHT_22 Configures the weight of HIST sub-window 22. Bits [7:0] are the fractional part. (R/W)

Register 38.89. ISP_HIST_WEIGHT4_REG (0x01D4)



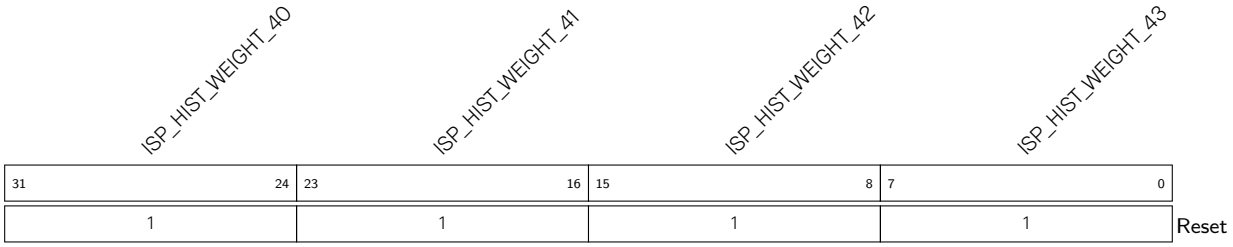
ISP_HIST_WEIGHT_34 Configures the weight of HIST sub-window 34. Bits [7:0] are the fractional part. (R/W)

ISP_HIST_WEIGHT_33 Configures the weight of HIST sub-window 33. Bits [7:0] are the fractional part. (R/W)

ISP_HIST_WEIGHT_32 Configures the weight of HIST sub-window 32. Bits [7:0] are the fractional part. (R/W)

ISP_HIST_WEIGHT_31 Configures the weight of HIST sub-window 31. Bits [7:0] are the fractional part. (R/W)

Register 38.90. ISP_HIST_WEIGHT5_REG (0x01D8)



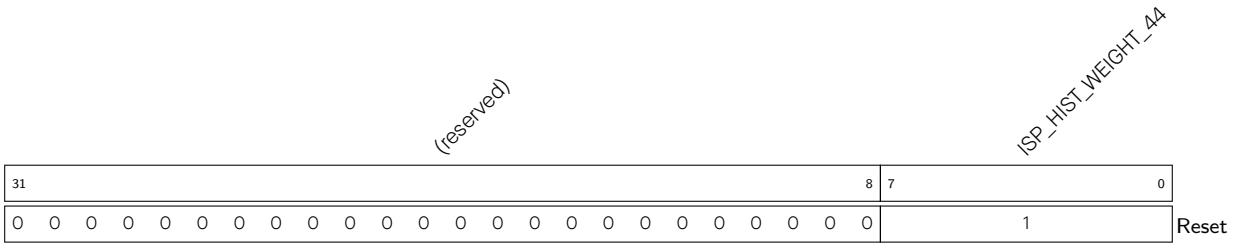
ISP_HIST_WEIGHT_43 Configures the weight of HIST sub-window 43. Bits [7:0] are the fractional part. (R/W)

ISP_HIST_WEIGHT_42 Configures the weight of HIST sub-window 42. Bits [7:0] are the fractional part. (R/W)

ISP_HIST_WEIGHT_41 Configures the weight of HIST sub-window 41. Bits [7:0] are the fractional part. (R/W)

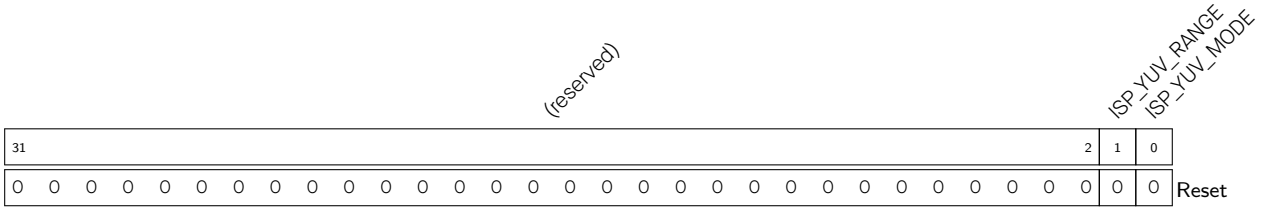
ISP_HIST_WEIGHT_40 Configures the weight of HIST sub-window 40. Bits [7:0] are the fractional part. (R/W)

Register 38.91. ISP_HIST_WEIGHT6_REG (0x01DC)



ISP_HIST_WEIGHT_44 Configures the weight of HIST sub-window 44. Bits [7:0] are the fractional part. (R/W)

Register 38.92. ISP_YUV_FORMAT_REG (0x0234)



ISP_YUV_MODE Configures the YUV output mode.

0: ITU-R BT.601

1: ITU-R BT.709

(R/W)

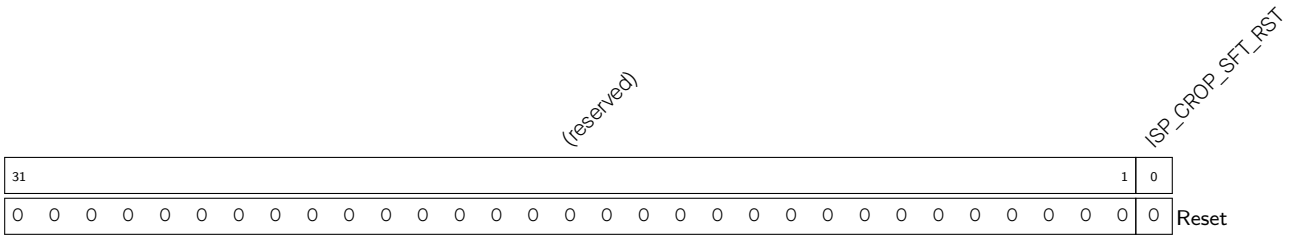
ISP_YUV_RANGE Configures the YUV output range.

0: Full range (YUV2RGB)

1: Limit range (YUV_Limit)

(R/W)

Register 38.93. ISP_CROP_CTRL_REG (0x0244)



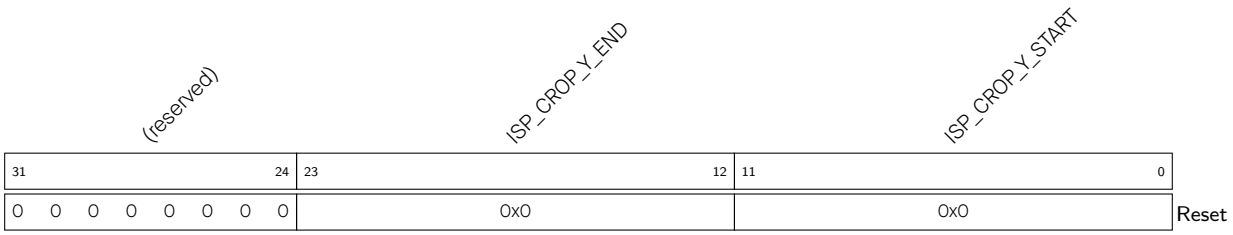
ISP_CROP_SFT_RST Configures whether to reset the CROP error status.

0: Not reset

1: Reset

(WT)

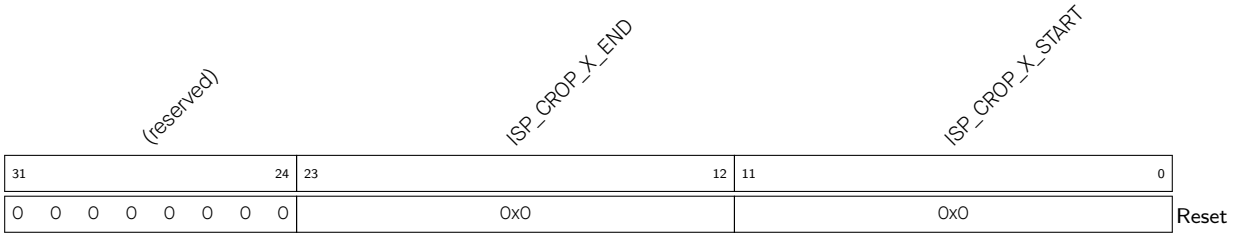
Register 38.94. ISP_CROP_Y_CAPTURE_REG (0x0248)



ISP_CROP_Y_START Configures the start coordinate in the Y direction. (R/W)

ISP_CROP_Y_END Configures the end coordinate in the Y direction. (R/W)

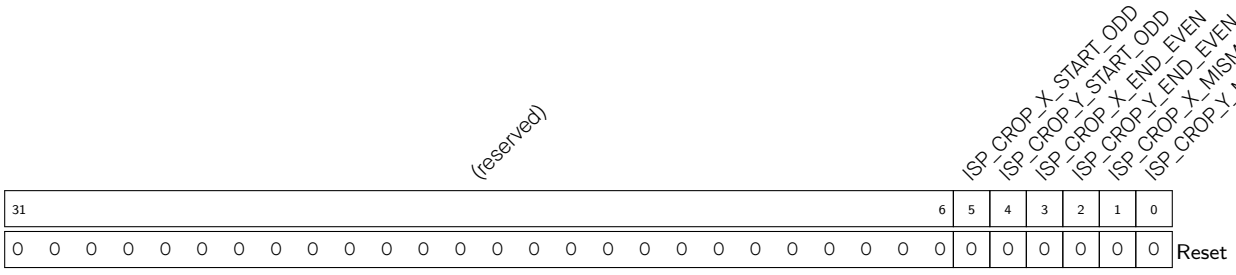
Register 38.95. ISP_CROP_X_CAPTURE_REG (0x024C)



ISP_CROP_X_START Configures the start coordinate in the X direction. (R/W)

ISP_CROP_X_END Configures the end coordinate in the X direction. (R/W)

Register 38.96. ISP_CROP_ERR_ST_REG (0x0250)



ISP_CROP_Y_MISMATCH Represents an error where the configured CROP Y coordinate exceeds the image boundary. (RO)

ISP_CROP_X_MISMATCH Represents an error where the configured CROP X coordinate exceeds the image boundary. (RO)

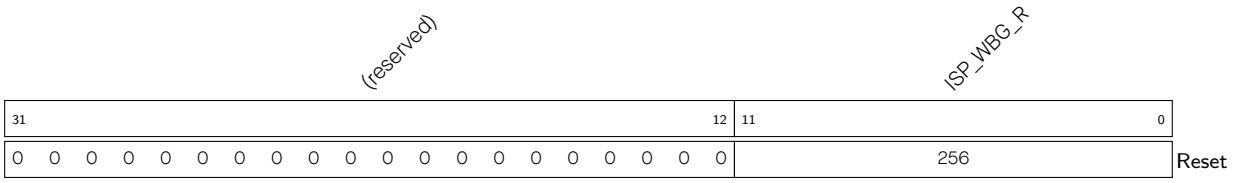
ISP_CROP_Y_END_EVEN Represents an error where the Y-direction end coordinate is even. (RO)

ISP_CROP_X_END_EVEN Represents an error where the X-direction end coordinate is even. (RO)

ISP_CROP_Y_START_ODD Represents an error where the Y-direction start coordinate is odd. (RO)

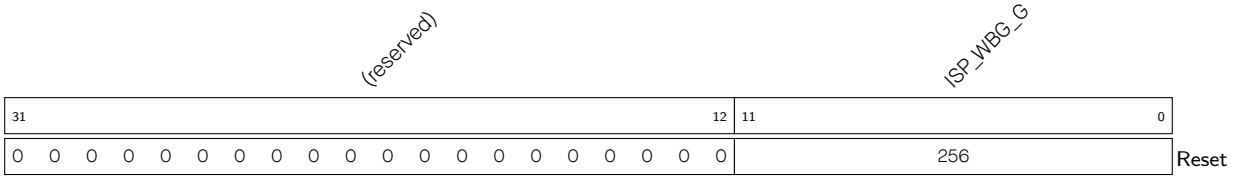
ISP_CROP_X_START_ODD Represents an error where the X-direction start coordinate is odd. (RO)

Register 38.97. ISP_WBG_COEF_R_REG (0x0254)



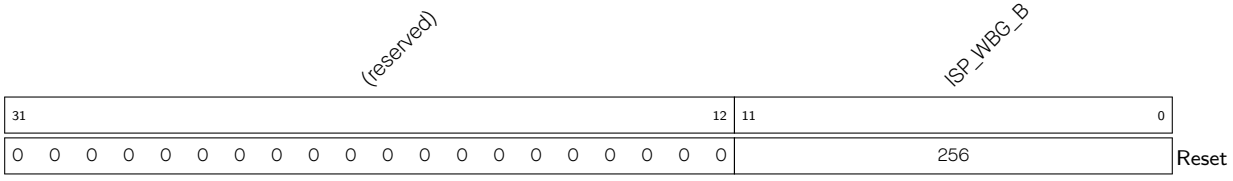
ISP_WBG_R Configures the WBG R-channel gain: 4-bit integer and 8-bit fractional. (R/W)

Register 38.98. ISP_WBG_COEF_G_REG (0x0258)



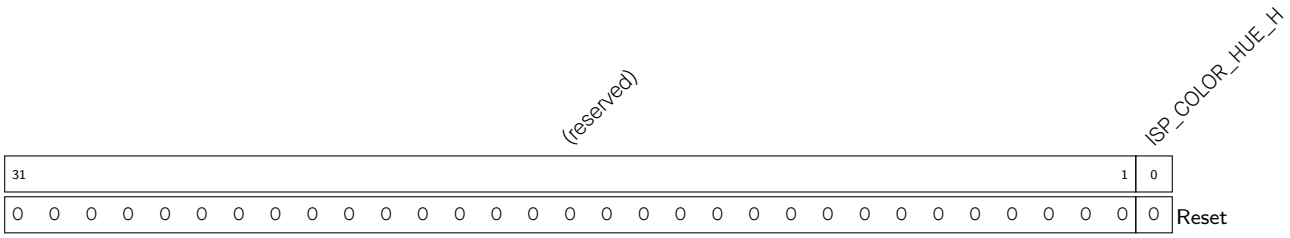
ISP_WBG_G Configures the WBG G-channel gain: 4-bit integer and 8-bit fractional. (R/W)

Register 38.99. ISP_WBG_COEF_B_REG (0x025C)



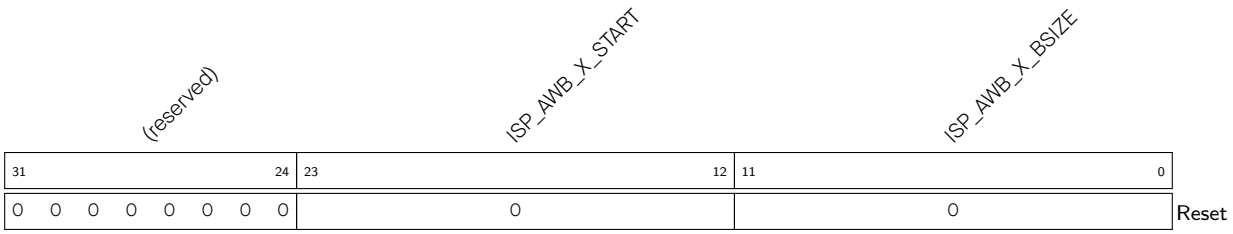
ISP_WBG_B Configures the WBG B-channel gain: 4-bit integer and 8-bit fractional. (R/W)

Register 38.100. ISP_COLOR_HUE_CTRL_REG (0x0260)



ISP_COLOR_HUE_H Configures the most significant bit of HUE. (R/W)

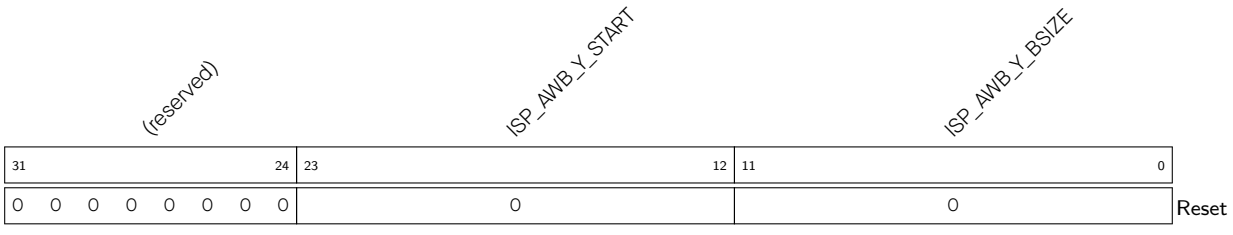
Register 38.101. ISP_AWB_BX_REG (0x0264)



ISP_AWB_X_BSIZE Configures the AWB sub-window size in the X direction; minimum is 4. (R/W)

ISP_AWB_X_START Configures the AWB sub-window start coordinate in the X direction. (R/W)

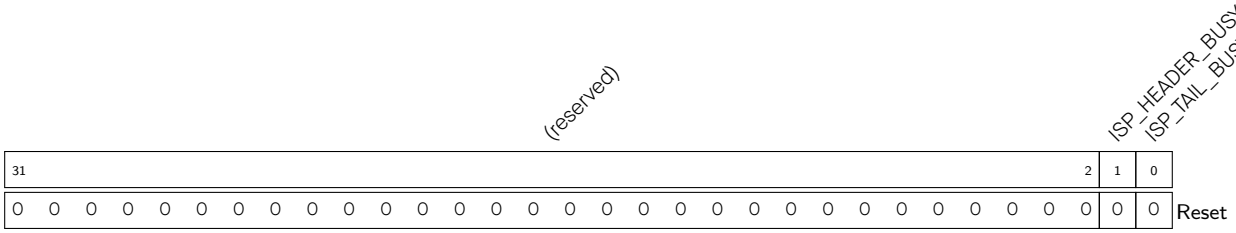
Register 38.102. ISP_AWB_BY_REG (0x0268)



ISP_AWB_Y_BSIZE Configures the AWB sub-window size in the Y direction. (R/W)

ISP_AWB_Y_START Configures the AWB sub-window start coordinate in the Y direction. (R/W)

Register 38.103. ISP_STATE_REG (0x026C)



ISP_TAIL_BUSY Represents whether isp_tail is busy.

0: Idle

1: Busy

(RO)

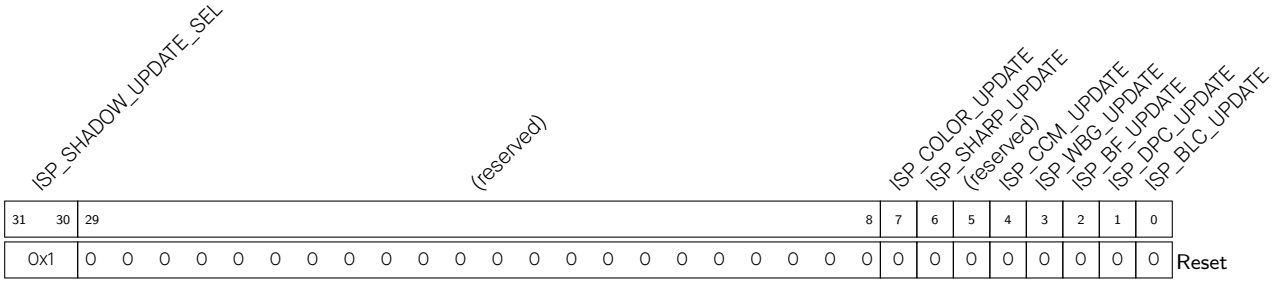
ISP_HEADER_BUSY Represents whether isp_header is busy.

0: Idle

1: Busy

(RO)

Register 38.104. ISP_SHADOW_REG_CTRL_REG (0x0270)



ISP_BLC_UPDATE Write 1 to update the BLC registers. (R/W)

ISP_DPC_UPDATE Write 1 to update the DPC registers. (R/W)

ISP_BF_UPDATE Write 1 to update the BF registers. (R/W)

ISP_WBG_UPDATE Write 1 to update the WBG registers. (R/W)

ISP_CCM_UPDATE Write 1 to update the CCM registers. (R/W)

ISP_SHARP_UPDATE Write 1 to update the SHARP registers. (R/W)

ISP_COLOR_UPDATE Write 1 to update the COLOR registers. (R/W)

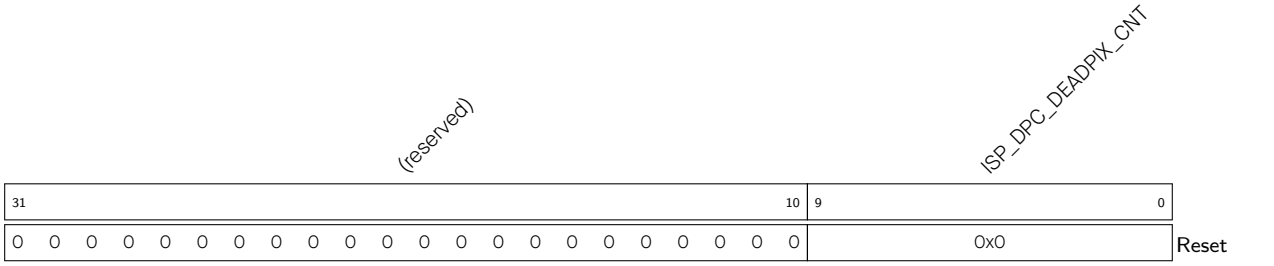
ISP_SHADOW_UPDATE_SEL Configures the shadow register update mode.

0: Disable shadow registers

1: Automatically update at every vsync

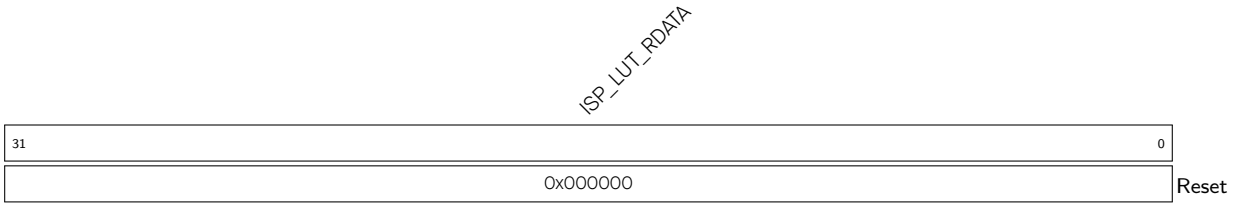
2: After writing 1 to the corresponding update register, automatically update at the next vsync (R/W)

Register 38.105. ISP_DPC_DEADPIX_CNT_REG (0x0044)



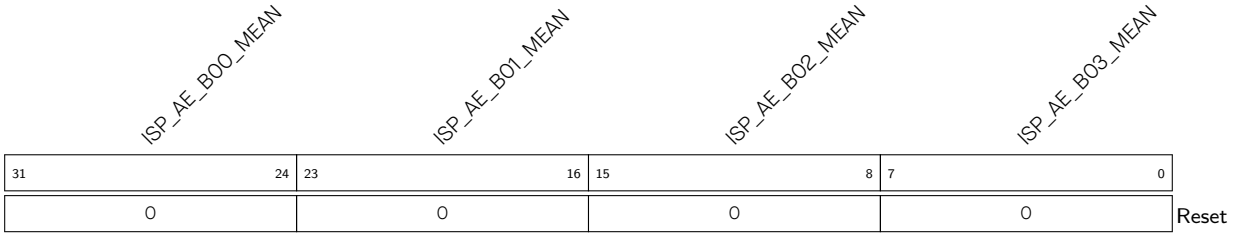
ISP_DPC_DEADPIX_CNT Represents the number of dead pixels counted during static calibration. (RO)

Register 38.106. ISP_LUT_RDATA_REG (0x0050)



ISP_LUT_RDATA Data read from the LUT. Read from this field after configuring ISP_LUT_CMD. (RO)

Register 38.107. ISP_AE_BLOCK_MEAN_0_REG (0x00D8)



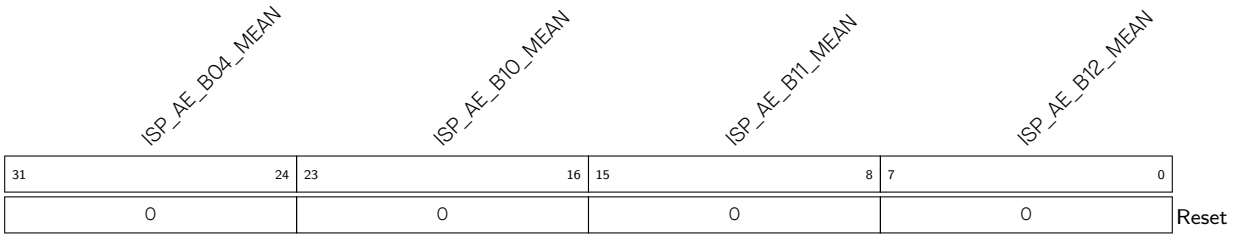
ISP_AE_B03_MEAN Represents the statistics from AE sub-window 03. (RO)

ISP_AE_B02_MEAN Represents the statistics from AE sub-window 02. (RO)

ISP_AE_B01_MEAN Represents the statistics from AE sub-window 01. (RO)

ISP_AE_B00_MEAN Represents the statistics from AE sub-window 00. (RO)

Register 38.108. ISP_AE_BLOCK_MEAN_1_REG (0x00DC)



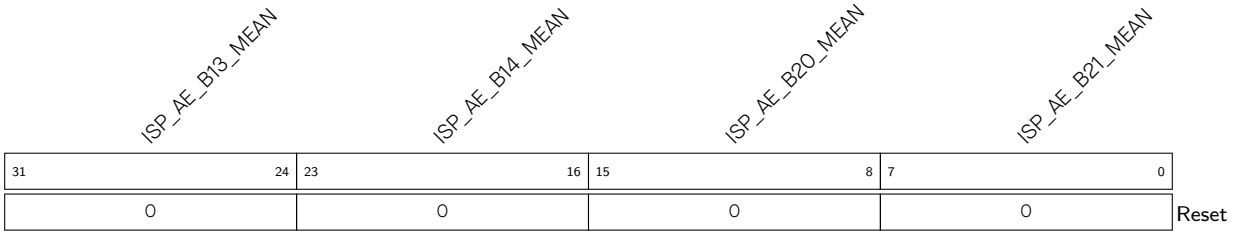
ISP_AE_B12_MEAN Represents the statistics from AE sub-window 12. (RO)

ISP_AE_B11_MEAN Represents the statistics from AE sub-window 11. (RO)

ISP_AE_B10_MEAN Represents the statistics from AE sub-window 10. (RO)

ISP_AE_B04_MEAN Represents the statistics from AE sub-window 04. (RO)

Register 38.109. ISP_AE_BLOCK_MEAN_2_REG (0x00E0)



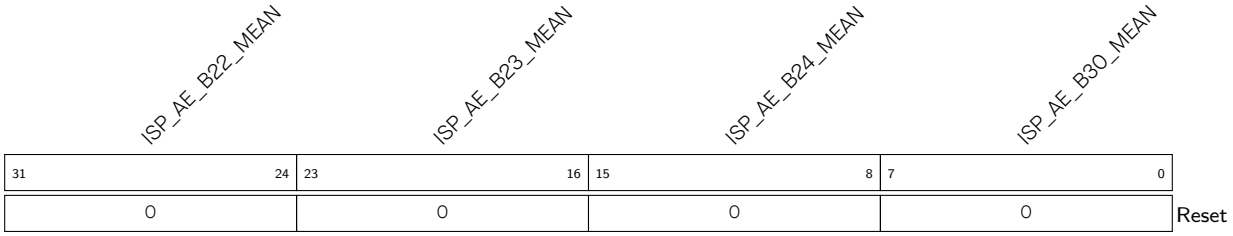
ISP_AE_B21_MEAN Represents the statistics from AE sub-window 21. (RO)

ISP_AE_B20_MEAN Represents the statistics from AE sub-window 20. (RO)

ISP_AE_B14_MEAN Represents the statistics from AE sub-window 14. (RO)

ISP_AE_B13_MEAN Represents the statistics from AE sub-window 13. (RO)

Register 38.110. ISP_AE_BLOCK_MEAN_3_REG (0x00E4)



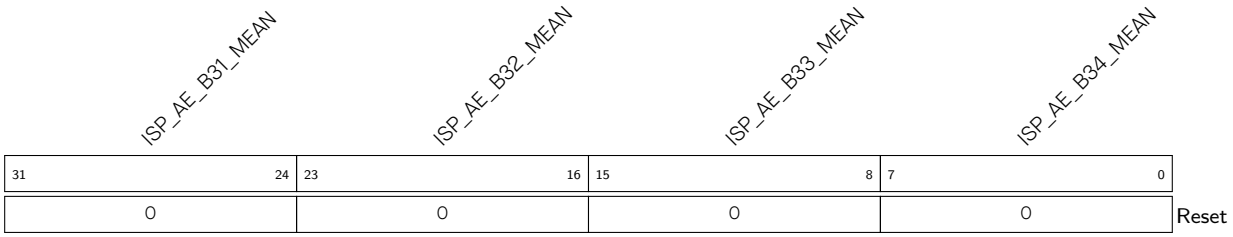
ISP_AE_B30_MEAN Represents the statistics from AE sub-window 30. (RO)

ISP_AE_B24_MEAN Represents the statistics from AE sub-window 24. (RO)

ISP_AE_B23_MEAN Represents the statistics from AE sub-window 23. (RO)

ISP_AE_B22_MEAN Represents the statistics from AE sub-window 22. (RO)

Register 38.111. ISP_AE_BLOCK_MEAN_4_REG (0x00E8)



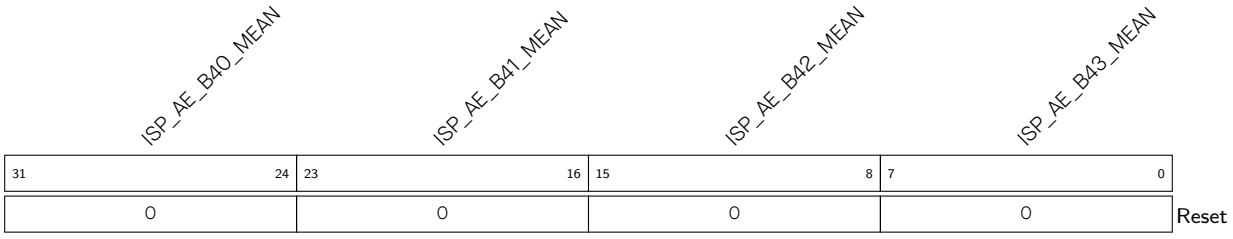
ISP_AE_B34_MEAN Represents the statistics from AE sub-window 34. (RO)

ISP_AE_B33_MEAN Represents the statistics from AE sub-window 33. (RO)

ISP_AE_B32_MEAN Represents the statistics from AE sub-window 32. (RO)

ISP_AE_B31_MEAN Represents the statistics from AE sub-window 31. (RO)

Register 38.112. ISP_AE_BLOCK_MEAN_5_REG (0x00EC)



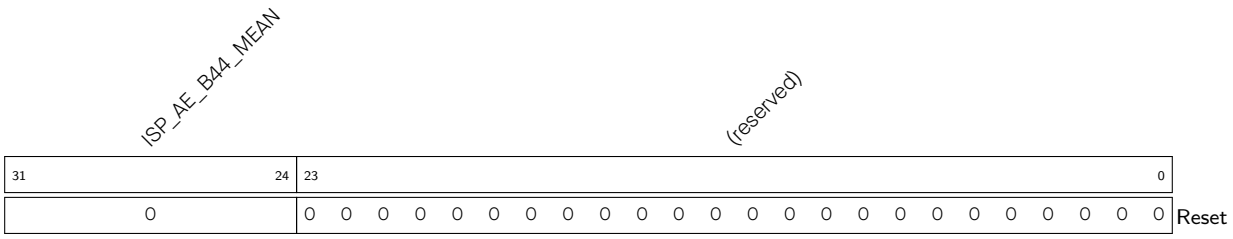
ISP_AE_B43_MEAN Represents the statistics from AE sub-window 43. (RO)

ISP_AE_B42_MEAN Represents the statistics from AE sub-window 42. (RO)

ISP_AE_B41_MEAN Represents the statistics from AE sub-window 41. (RO)

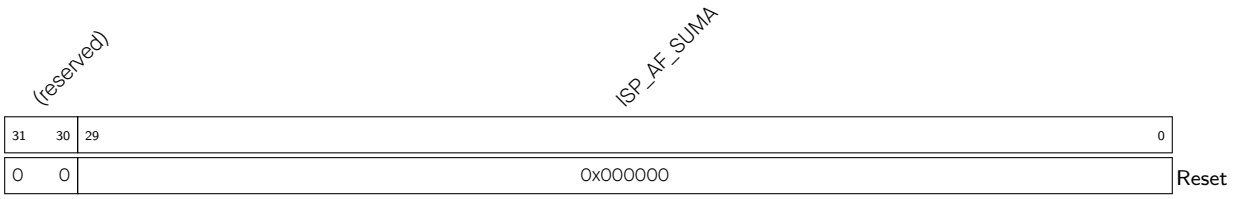
ISP_AE_B40_MEAN Represents the statistics from AE sub-window 40. (RO)

Register 38.113. ISP_AE_BLOCK_MEAN_6_REG (0x00F0)



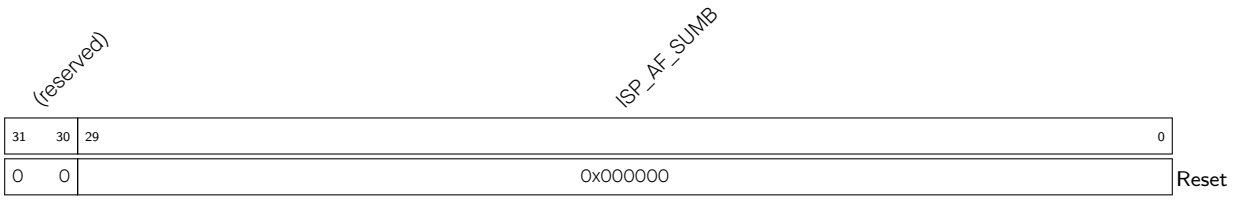
ISP_AE_B44_MEAN Represents the statistics from AE sub-window 44. (RO)

Register 38.114. ISP_AF_SUM_A_REG (0x014C)



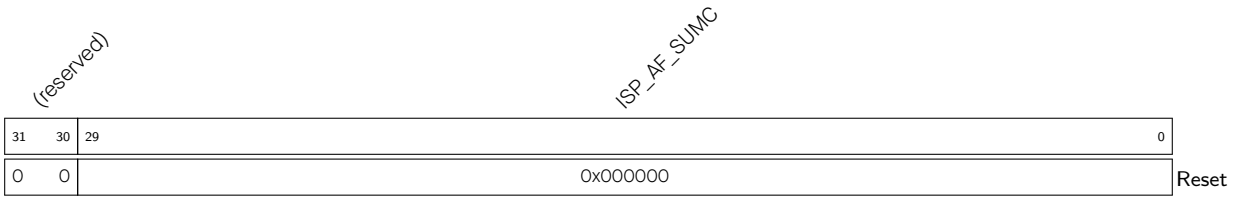
ISP_AF_SUMA Represents the sharpness statistics from AF window A. (RO)

Register 38.115. ISP_AF_SUM_B_REG (0x0150)



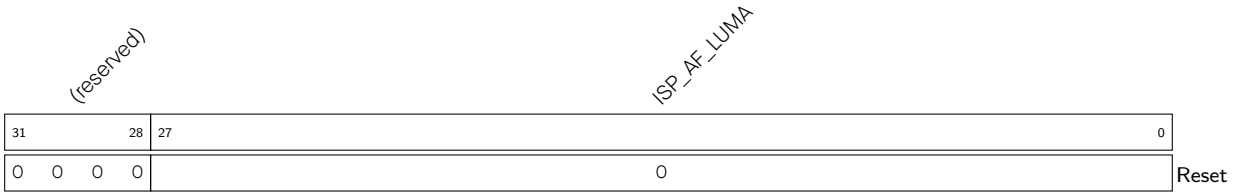
ISP_AF_SUMB Represents the sharpness statistics from AF window B. (RO)

Register 38.116. ISP_AF_SUM_C_REG (0x0154)



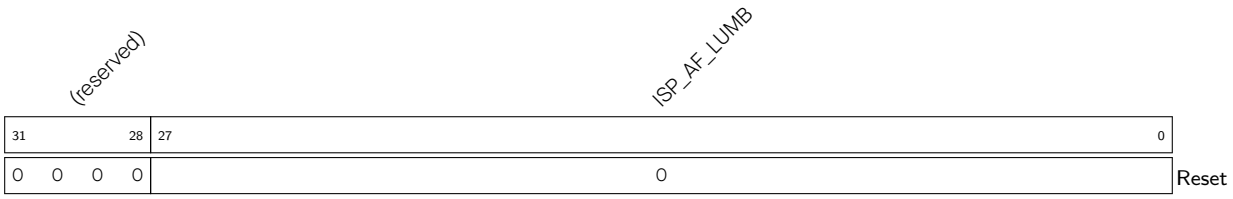
ISP_AF_SUMC Represents the sharpness statistics from AF window C. (RO)

Register 38.117. ISP_AF_LUM_A_REG (0x0158)



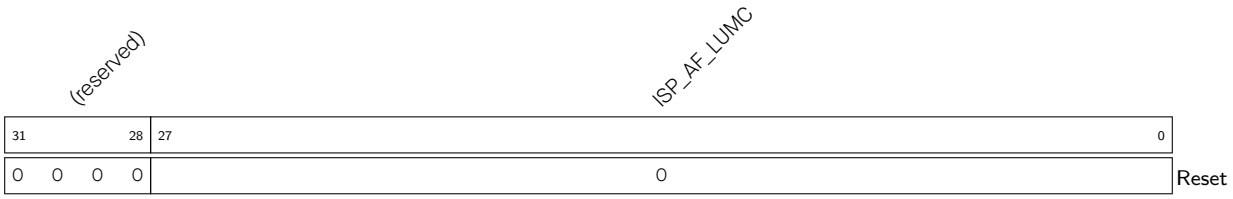
ISP_AF_LUMA Represents the luminance statistics from AF window A. (RO)

Register 38.118. ISP_AF_LUM_B_REG (0x015C)



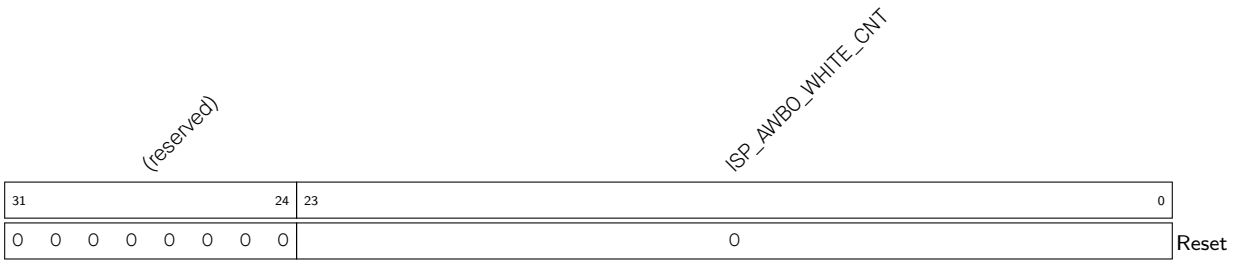
ISP_AF_LUMB Represents the luminance statistics from AF window B. (RO)

Register 38.119. ISP_AF_LUM_C_REG (0x0160)



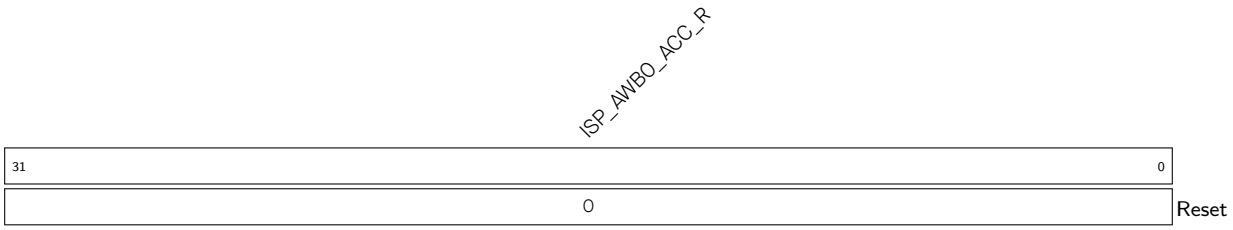
ISP_AF_LUMC Represents the luminance statistics from AF window C. (RO)

Register 38.120. ISP_AWB0_WHITE_CNT_REG (0x017C)



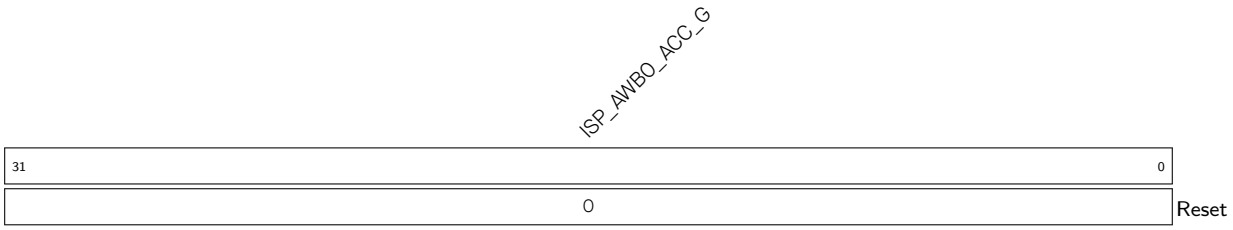
ISP_AWB0_WHITE_CNT Represents the number of white patches within the AWB statistical window. (RO)

Register 38.121. ISP_AWB0_ACC_R_REG (0x0180)



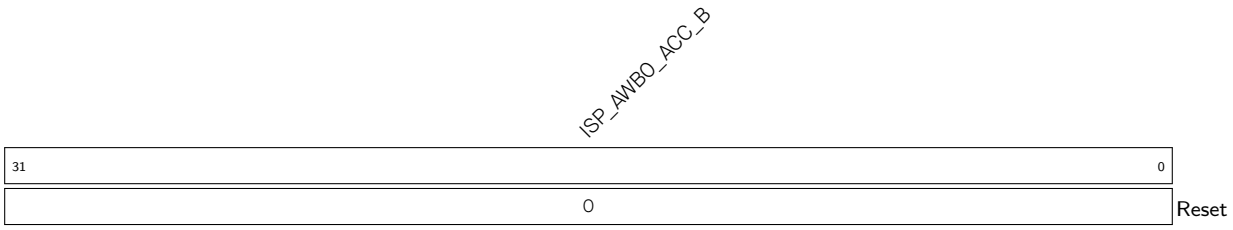
ISP_AWB0_ACC_R Represents the accumulated value of the R component for white patches within the AWB statistical window. (RO)

Register 38.122. ISP_AWB0_ACC_G_REG (0x0184)



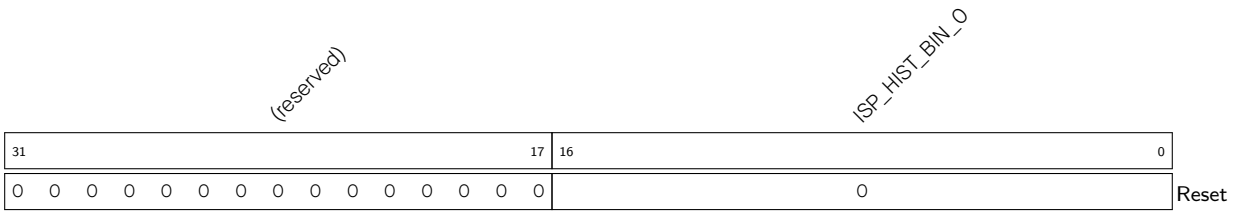
ISP_AWB0_ACC_G Represents the accumulated value of the G component for white patches within the AWB statistical window. (RO)

Register 38.123. ISP_AWB0_ACC_B_REG (0x0188)



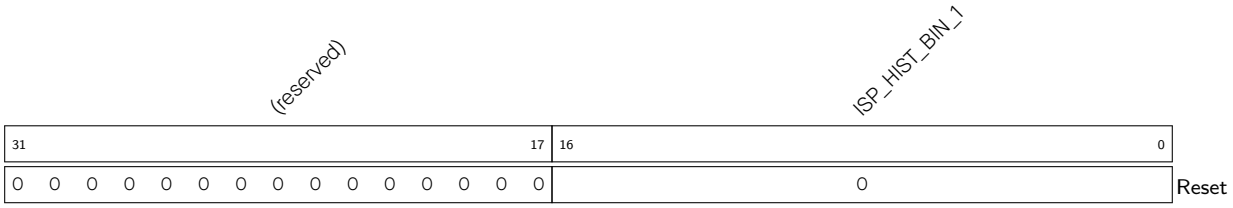
ISP_AWB0_ACC_B Represents the accumulated value of the B component for white patches within the AWB statistical window. (RO)

Register 38.124. ISP_HIST_BIN0_REG (0x01E0)



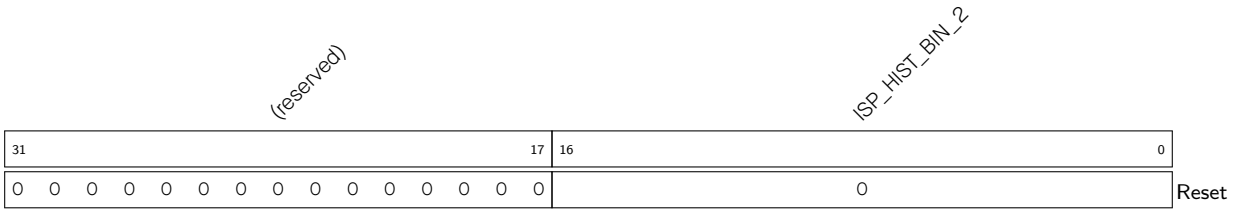
ISP_HIST_BIN_0 Represents the number of pixels where the luminance falls in the range of interval 0 in the HIST statistical window. (RO)

Register 38.125. ISP_HIST_BIN1_REG (0x01E4)



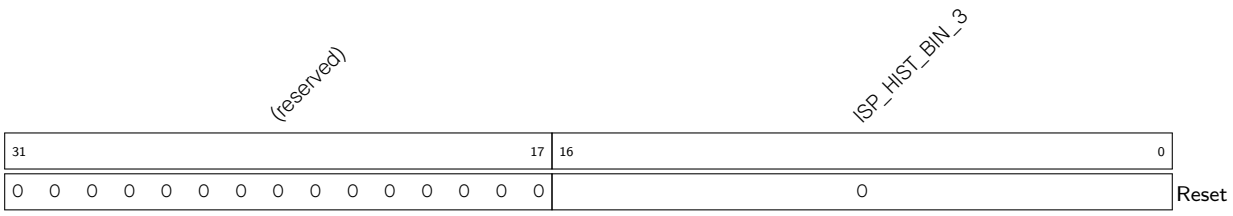
ISP_HIST_BIN_1 Represents the number of pixels where the luminance falls in the range of interval 1 in the HIST statistical window. (RO)

Register 38.126. ISP_HIST_BIN2_REG (0x01E8)



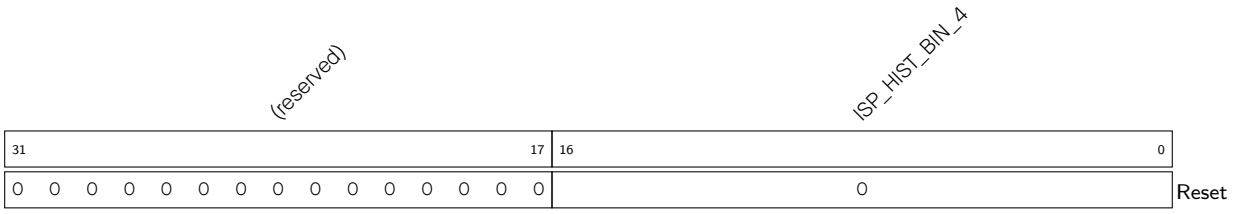
ISP_HIST_BIN_2 Represents the number of pixels where the luminance falls in the range of interval 2 in the HIST statistical window. (RO)

Register 38.127. ISP_HIST_BIN3_REG (0x01EC)



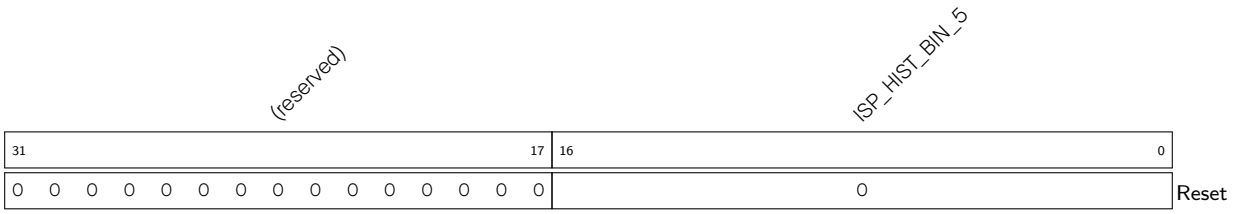
ISP_HIST_BIN_3 Represents the number of pixels where the luminance falls in the range of interval 3 in the HIST statistical window. (RO)

Register 38.128. ISP_HIST_BIN4_REG (0x01F0)



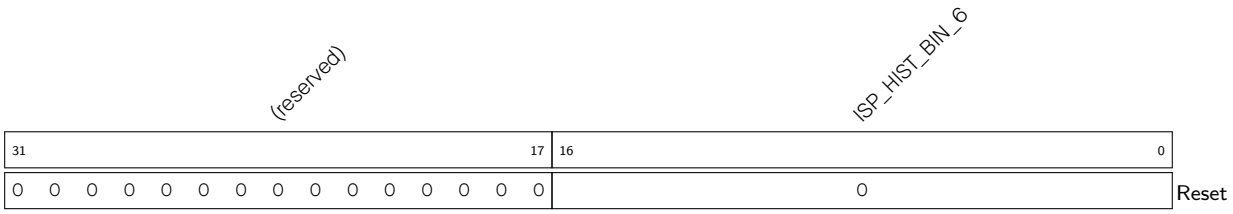
ISP_HIST_BIN_4 Represents the number of pixels where the luminance falls in the range of interval 4 in the HIST statistical window. (RO)

Register 38.129. ISP_HIST_BIN5_REG (0x01F4)



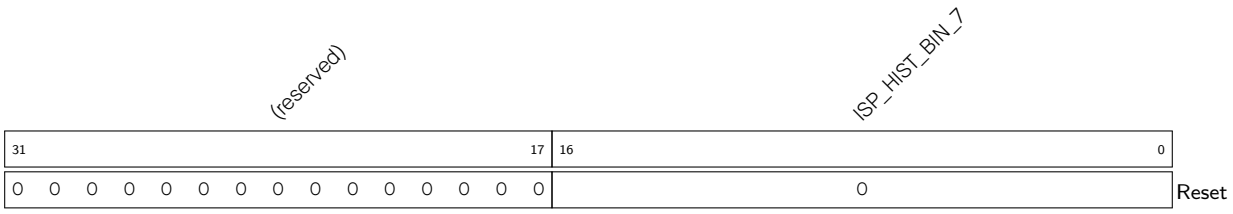
ISP_HIST_BIN_5 Represents the number of pixels where the luminance falls in the range of interval 5 in the HIST statistical window. (RO)

Register 38.130. ISP_HIST_BIN6_REG (0x01F8)



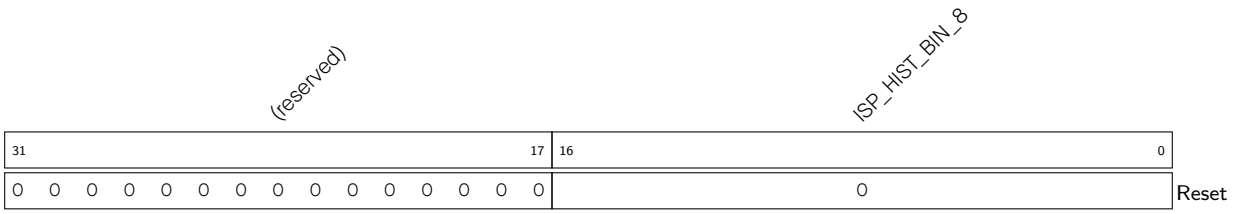
ISP_HIST_BIN_6 Represents the number of pixels where the luminance falls in the range of interval 6 in the HIST statistical window. (RO)

Register 38.131. ISP_HIST_BIN7_REG (0x01FC)



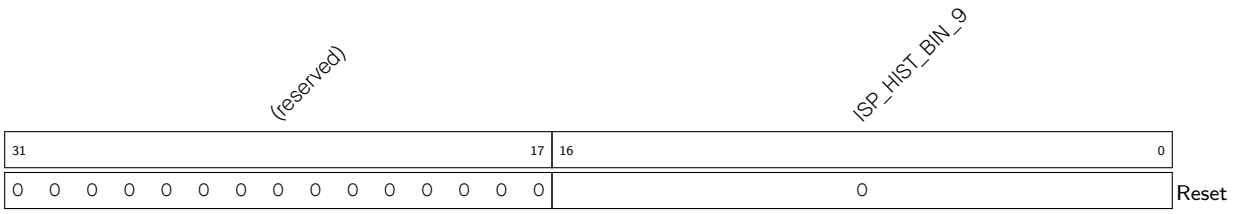
ISP_HIST_BIN_7 Represents the number of pixels where the luminance falls in the range of interval 7 in the HIST statistical window. (RO)

Register 38.132. ISP_HIST_BIN8_REG (0x0200)



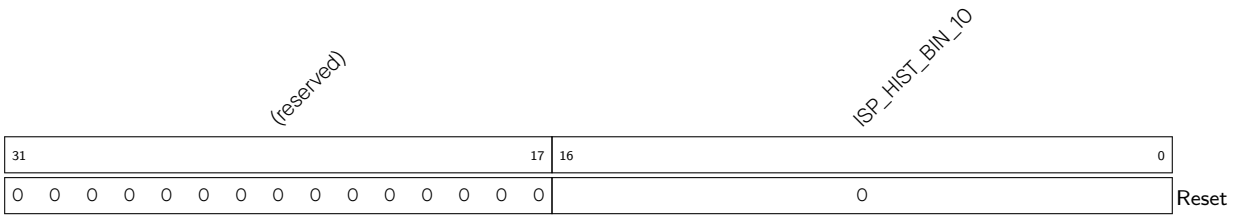
ISP_HIST_BIN_8 Represents the number of pixels where the luminance falls in the range of interval 8 in the HIST statistical window. (RO)

Register 38.133. ISP_HIST_BIN9_REG (0x0204)



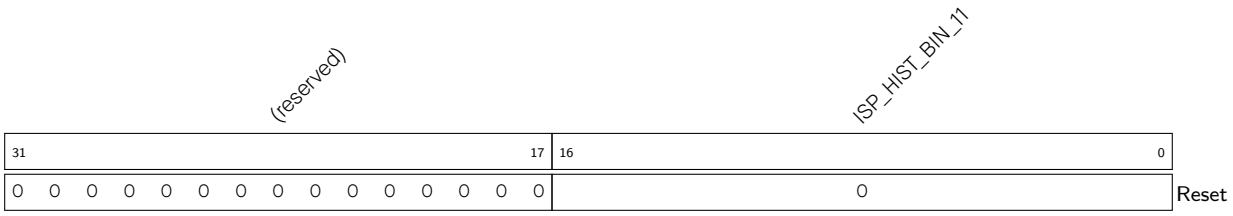
ISP_HIST_BIN_9 Represents the number of pixels where the luminance falls in the range of interval 9 in the HIST statistical window. (RO)

Register 38.134. ISP_HIST_BIN10_REG (0x0208)



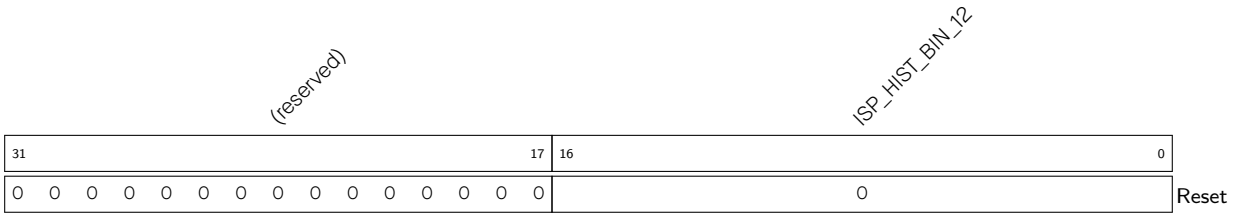
ISP_HIST_BIN_10 Represents the number of pixels where the luminance falls in the range of interval 10 in the HIST statistical window. (RO)

Register 38.135. ISP_HIST_BIN11_REG (0x020C)



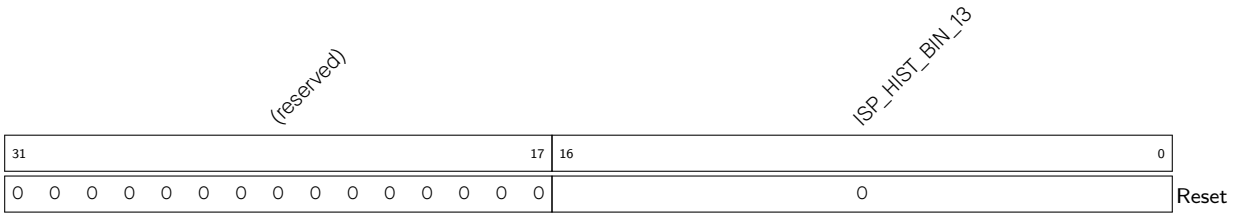
ISP_HIST_BIN_11 Represents the number of pixels where the luminance falls in the range of interval 11 in the HIST statistical window. (RO)

Register 38.136. ISP_HIST_BIN12_REG (0x0210)



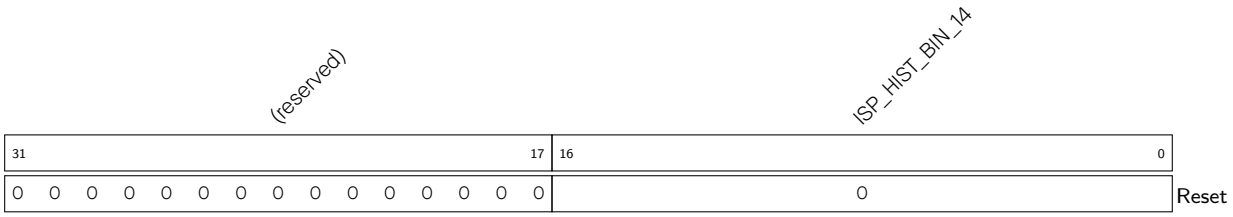
ISP_HIST_BIN_12 Represents the number of pixels where the luminance falls in the range of interval 12 in the HIST statistical window. (RO)

Register 38.137. ISP_HIST_BIN13_REG (0x0214)



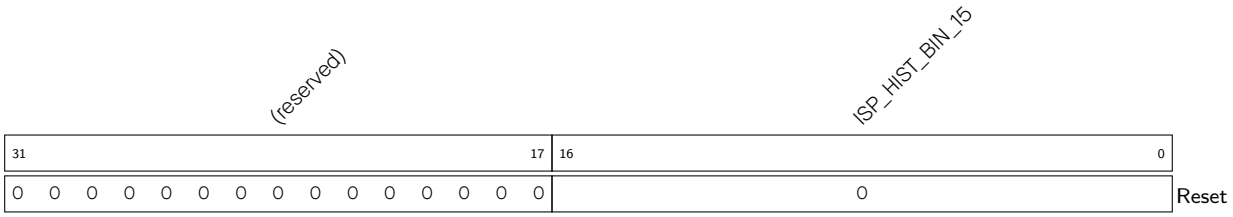
ISP_HIST_BIN_13 Represents the number of pixels where the luminance falls in the range of interval 13 in the HIST statistical window. (RO)

Register 38.138. ISP_HIST_BIN14_REG (0x0218)



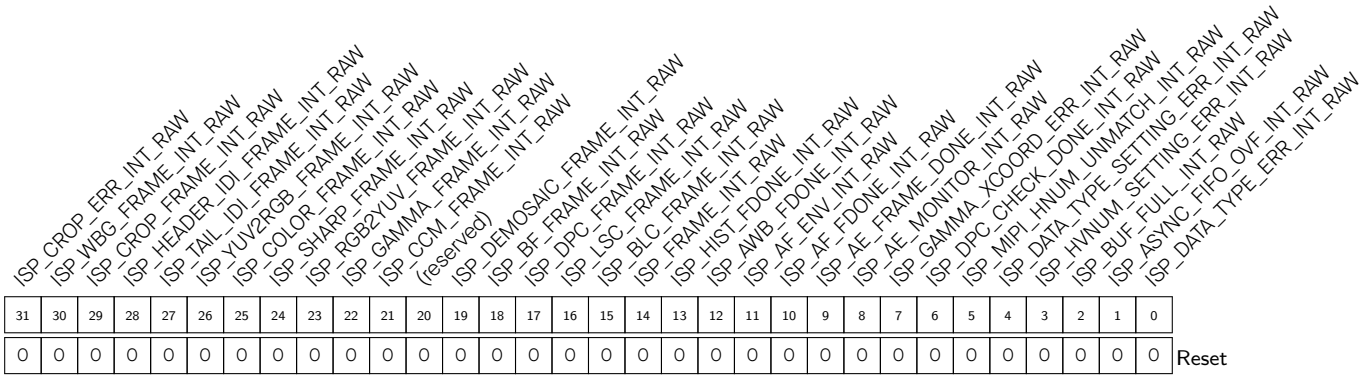
ISP_HIST_BIN_14 Represents the number of pixels where the luminance falls in the range of interval 14 in the HIST statistical window. (RO)

Register 38.139. ISP_HIST_BIN15_REG (0x021C)



ISP_HIST_BIN_15 Represents the number of pixels where the luminance falls in the range of interval 15 in the HIST statistical window. (RO)

Register 38.140. ISP_INT_RAW_REG (0x0064)



ISP_DATA_TYPE_ERR_INT_RAW The raw interrupt status of ISP_DATA_TYPE_ERR_INT. (R/SS/WTC)

ISP_ASYNC_FIFO_OVF_INT_RAW The raw interrupt status of ISP_ASYNC_FIFO_OVF_INT. (R/SS/WTC)

ISP_BUF_FULL_INT_RAW The raw interrupt status of ISP_BUF_FULL_INT. (R/SS/WTC)

ISP_HVNUM_SETTING_ERR_INT_RAW The raw interrupt status of ISP_HVNUM_SETTING_ERR_INT. (R/SS/WTC)

ISP_DATA_TYPE_SETTING_ERR_INT_RAW The raw interrupt status of ISP_DATA_TYPE_SETTING_ERR_INT. (R/SS/WTC)

ISP_MIPI_HNUM_UNMATCH_INT_RAW The raw interrupt status of ISP_MIPI_HNUM_UNMATCH_INT. (R/SS/WTC)

ISP_DPC_CHECK_DONE_INT_RAW The raw interrupt status of ISP_DPC_CHECK_DONE_INT. (R/SS/WTC)

ISP_GAMMA_XCOORD_ERR_INT_RAW The raw interrupt status of ISP_GAMMA_XCOORD_ERR_INT. (R/SS/WTC)

ISP_AE_MONITOR_INT_RAW The raw interrupt status of ISP_AE_MONITOR_INT. (R/SS/WTC)

ISP_AE_FRAME_DONE_INT_RAW The raw interrupt status of ISP_AE_FRAME_DONE_INT. (R/SS/WTC)

ISP_AF_FDONE_INT_RAW The raw interrupt status of ISP_AF_FDONE_INT. (R/SS/WTC)

ISP_AF_ENV_INT_RAW The raw interrupt status of ISP_AF_ENV_INT. (R/SS/WTC)

ISP_AWB_FDONE_INT_RAW The raw interrupt status of ISP_AWB_FDONE_INT. (R/SS/WTC)

ISP_HIST_FDONE_INT_RAW The raw interrupt status of ISP_HIST_FDONE_INT. (R/SS/WTC)

ISP_FRAME_INT_RAW The raw interrupt status of ISP_FRAME_INT. (R/SS/WTC)

ISP_BLC_FRAME_INT_RAW The raw interrupt status of ISP_BLC_FRAME_INT. (R/SS/WTC)

ISP_LSC_FRAME_INT_RAW The raw interrupt status of ISP_LSC_FRAME_INT. (R/SS/WTC)

Continued on the next page...

Register 38.140. ISP_INT_RAW_REG (0x0064)

Continued from the previous page...

ISP_DPC_FRAME_INT_RAW The raw interrupt status of ISP_DPC_FRAME_INT. (R/SS/WTC)

ISP_BF_FRAME_INT_RAW The raw interrupt status of ISP_BF_FRAME_INT. (R/SS/WTC)

ISP_DEMOSAIC_FRAME_INT_RAW The raw interrupt status of ISP_DEMOSAIC_FRAME_INT. (R/SS/WTC)

ISP_CCM_FRAME_INT_RAW The raw interrupt status of ISP_CCM_FRAME_INT. (R/SS/WTC)

ISP_GAMMA_FRAME_INT_RAW The raw interrupt status of ISP_GAMMA_FRAME_INT. (R/SS/WTC)

ISP_RGB2YUV_FRAME_INT_RAW The raw interrupt status of ISP_RGB2YUV_FRAME_INT. (R/SS/WTC)

ISP_SHARP_FRAME_INT_RAW The raw interrupt status of ISP_SHARP_FRAME_INT. (R/SS/WTC)

ISP_COLOR_FRAME_INT_RAW The raw interrupt status of ISP_COLOR_FRAME_INT. (R/SS/WTC)

ISP_YUV2RGB_FRAME_INT_RAW The raw interrupt status of ISP_YUV2RGB_FRAME_INT. (R/SS/WTC)

ISP_TAIL_IDI_FRAME_INT_RAW The raw interrupt status of ISP_TAIL_IDI_FRAME_INT. (R/SS/WTC)

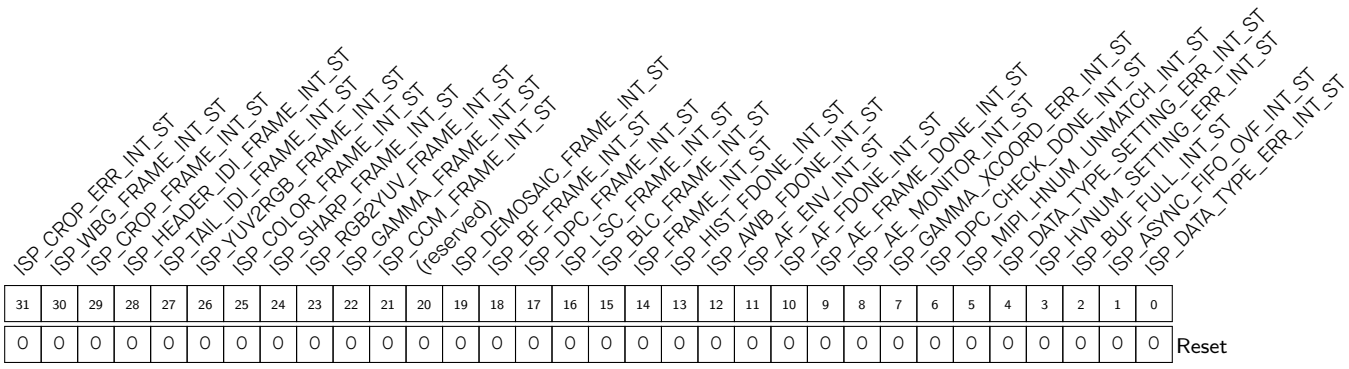
ISP_HEADER_IDI_FRAME_INT_RAW The raw interrupt status of ISP_HEADER_IDI_FRAME_INT. (R/SS/WTC)

ISP_CROP_FRAME_INT_RAW The raw interrupt status of ISP_CROP_FRAME_INT. (R/SS/WTC)

ISP_WBG_FRAME_INT_RAW The raw interrupt status of ISP_WBG_FRAME_INT. (R/SS/WTC)

ISP_CROP_ERR_INT_RAW The raw interrupt status of ISP_CROP_ERR_INT. (R/SS/WTC)

Register 38.141. ISP_INT_ST_REG (0x0068)



ISP_DATA_TYPE_ERR_INT_ST The masked interrupt status of ISP_DATA_TYPE_ERR_INT. (RO)

ISP_ASYNC_FIFO_OVF_INT_ST The masked interrupt status of ISP_ASYNC_FIFO_OVF_INT. (RO)

ISP_BUF_FULL_INT_ST The masked interrupt status of ISP_BUF_FULL_INT. (RO)

ISP_HVNUM_SETTING_ERR_INT_ST The masked interrupt status of ISP_HVNUM_SETTING_ERR_INT. (RO)

ISP_DATA_TYPE_SETTING_ERR_INT_ST The masked interrupt status of ISP_DATA_TYPE_SETTING_ERR_INT. (RO)

ISP_MIPI_HNUM_UNMATCH_INT_ST The masked interrupt status of ISP_MIPI_HNUM_UNMATCH_INT. (RO)

ISP_DPC_CHECK_DONE_INT_ST The masked interrupt status of ISP_DPC_CHECK_DONE_INT. (RO)

ISP_GAMMA_XCOORD_ERR_INT_ST The masked interrupt status of ISP_GAMMA_XCOORD_ERR_INT. (RO)

ISP_AE_MONITOR_INT_ST The masked interrupt status of ISP_AE_MONITOR_INT. (RO)

ISP_AE_FRAME_DONE_INT_ST The masked interrupt status of ISP_AE_FRAME_DONE_INT. (RO)

ISP_AF_FDONE_INT_ST The masked interrupt status of ISP_AF_FDONE_INT. (RO)

ISP_AF_ENV_INT_ST The masked interrupt status of ISP_AF_ENV_INT. (RO)

ISP_AWB_FDONE_INT_ST The masked interrupt status of ISP_AWB_FDONE_INT. (RO)

ISP_HIST_FDONE_INT_ST The masked interrupt status of ISP_HIST_FDONE_INT. (RO)

ISP_FRAME_INT_ST The masked interrupt status of ISP_FRAME_INT. (RO)

ISP_BLC_FRAME_INT_ST The masked interrupt status of ISP_BLC_FRAME_INT. (RO)

ISP_LSC_FRAME_INT_ST The masked interrupt status of ISP_LSC_FRAME_INT. (RO)

Continued on the next page...

Register 38.141. ISP_INT_ST_REG (0x0068)

Continued from the previous page...

ISP_DPC_FRAME_INT_ST The masked interrupt status of ISP_DPC_FRAME_INT. (RO)

ISP_BF_FRAME_INT_ST The masked interrupt status of ISP_BF_FRAME_INT. (RO)

ISP_DEMOSAIC_FRAME_INT_ST The masked interrupt status of ISP_DEMOSAIC_FRAME_INT. (RO)

ISP_CCM_FRAME_INT_ST The masked interrupt status of ISP_CCM_FRAME_INT. (RO)

ISP_GAMMA_FRAME_INT_ST The masked interrupt status of ISP_GAMMA_FRAME_INT. (RO)

ISP_RGB2YUV_FRAME_INT_ST The masked interrupt status of ISP_RGB2YUV_FRAME_INT. (RO)

ISP_SHARP_FRAME_INT_ST The masked interrupt status of ISP_SHARP_FRAME_INT. (RO)

ISP_COLOR_FRAME_INT_ST The masked interrupt status of ISP_COLOR_FRAME_INT. (RO)

ISP_YUV2RGB_FRAME_INT_ST The masked interrupt status of ISP_YUV2RGB_FRAME_INT. (RO)

ISP_TAIL_IDI_FRAME_INT_ST The masked interrupt status of ISP_TAIL_IDI_FRAME_INT. (RO)

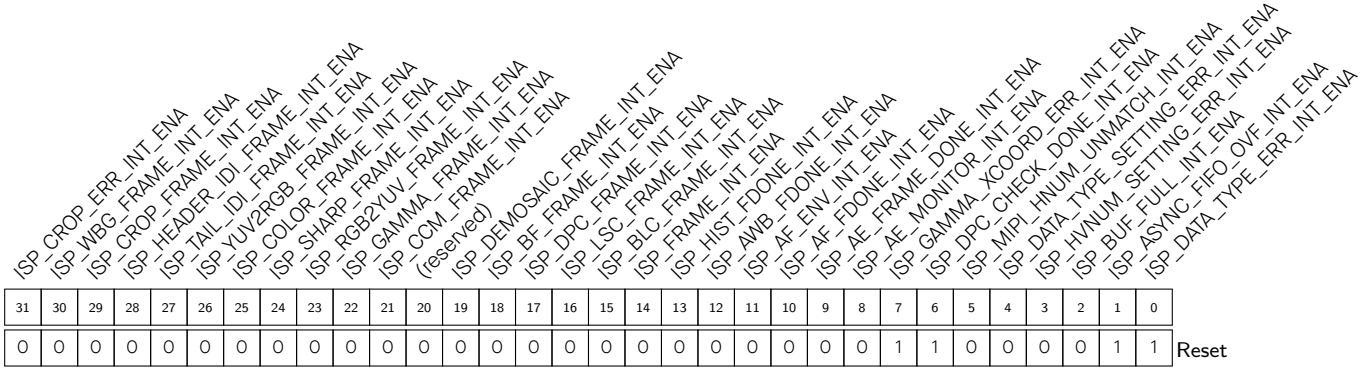
ISP_HEADER_IDI_FRAME_INT_ST The masked interrupt status of ISP_HEADER_IDI_FRAME_INT. (RO)

ISP_CROP_FRAME_INT_ST The masked interrupt status of ISP_CROP_FRAME. (RO)

ISP_WBG_FRAME_INT_ST The masked interrupt status of ISP_WBG_FRAME_INT. (RO)

ISP_CROP_ERR_INT_ST The masked interrupt status of ISP_CROP_ERR_INT. (RO)

Register 38.142. ISP_INT_ENA_REG (0x006C)



ISP_DATA_TYPE_ERR_INT_ENA Write 1 to enable ISP_DATA_TYPE_ERR_INT. (R/W)

ISP_ASYNC_FIFO_OVF_INT_ENA Write 1 to enable ISP_ASYNC_FIFO_OVF_INT. (R/W)

ISP_BUF_FULL_INT_ENA Write 1 to enable ISP_BUF_FULL_INT. (R/W)

ISP_HVNUM_SETTING_ERR_INT_ENA Write 1 to enable ISP_HVNUM_SETTING_ERR_INT. (R/W)

ISP_DATA_TYPE_SETTING_ERR_INT_ENA Write 1 to enable ISP_DATA_TYPE_SETTING_ERR_INT. (R/W)

ISP_MIPI_HNUM_UNMATCH_INT_ENA Write 1 to enable ISP_MIPI_HNUM_UNMATCH_INT. (R/W)

ISP_DPC_CHECK_DONE_INT_ENA Write 1 to enable ISP_DPC_CHECK_DONE_INT. (R/W)

ISP_GAMMA_XCOORD_ERR_INT_ENA Write 1 to enable ISP_GAMMA_XCOORD_ERR_INT. (R/W)

ISP_AE_MONITOR_INT_ENA Write 1 to enable ISP_AE_MONITOR_INT. (R/W)

ISP_AE_FRAME_DONE_INT_ENA Write 1 to enable ISP_AE_FRAME_DONE_INT. (R/W)

ISP_AF_FDONE_INT_ENA Write 1 to enable ISP_AF_FDONE_INT. (R/W)

ISP_AF_ENV_INT_ENA Write 1 to enable ISP_AF_ENV_INT. (R/W)

ISP_AWB_FDONE_INT_ENA Write 1 to enable ISP_AWB_FDONE_INT. (R/W)

ISP_HIST_FDONE_INT_ENA Write 1 to enable ISP_HIST_FDONE_INT. (R/W)

ISP_FRAME_INT_ENA Write 1 to enable ISP_FRAME_INT. (R/W)

ISP_BLC_FRAME_INT_ENA Write 1 to enable ISP_BLC_FRAME_INT. (R/W)

ISP_LSC_FRAME_INT_ENA Write 1 to enable ISP_LSC_FRAME_INT. (R/W)

ISP_DPC_FRAME_INT_ENA Write 1 to enable ISP_DPC_FRAME_INT. (R/W)

ISP_BF_FRAME_INT_ENA Write 1 to enable ISP_BF_FRAME_INT. (R/W)

ISP_DEMOSAIC_FRAME_INT_ENA Write 1 to enable ISP_DEMOSAIC_FRAME_INT. (R/W)

Continued on the next page...

Register 38.142. ISP_INT_ENA_REG (0x006C)

Continued from the previous page...

ISP_CCM_FRAME_INT_ENA Write 1 to enable ISP_CCM_FRAME_INT. (R/W)

ISP_GAMMA_FRAME_INT_ENA Write 1 to enable ISP_GAMMA_FRAME_INT. (R/W)

ISP_RGB2YUV_FRAME_INT_ENA Write 1 to enable ISP_RGB2YUV_FRAME_INT. (R/W)

ISP_SHARP_FRAME_INT_ENA Write 1 to enable ISP_SHARP_FRAME_INT. (R/W)

ISP_COLOR_FRAME_INT_ENA Write 1 to enable ISP_COLOR_FRAME_INT. (R/W)

ISP_YUV2RGB_FRAME_INT_ENA Write 1 to enable ISP_YUV2RGB_FRAME_INT. (R/W)

ISP_TAIL_IDI_FRAME_INT_ENA Write 1 to enable ISP_TAIL_IDI_FRAME_INT. (R/W)

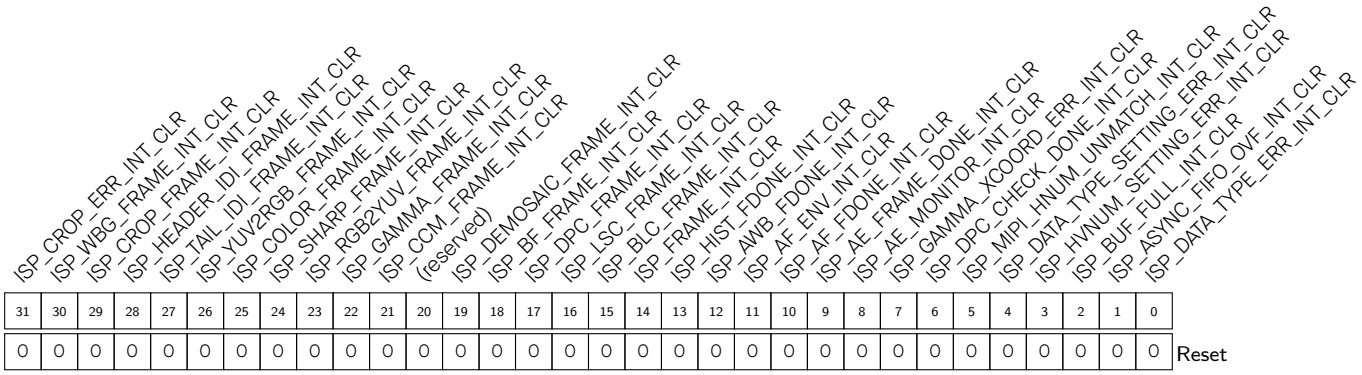
ISP_HEADER_IDI_FRAME_INT_ENA Write 1 to enable ISP_HEADER_IDI_FRAME_INT. (R/W)

ISP_CROP_FRAME_INT_ENA Write 1 to enable ISP_CROP_FRAME_INT. (R/W)

ISP_WBG_FRAME_INT_ENA Write 1 to enable ISP_WBG_FRAME_INT. (R/W)

ISP_CROP_ERR_INT_ENA Write 1 to enable ISP_CROP_ERR_INT. (R/W)

Register 38.143. ISP_INT_CLR_REG (0x0070)



ISP_DATA_TYPE_ERR_INT_CLR Write 1 to clear ISP_DATA_TYPE_ERR_INT. (WT)

ISP_ASYNC_FIFO_OVF_INT_CLR Write 1 to clear ISP_ASYNC_FIFO_OVF_INT. (WT)

ISP_BUF_FULL_INT_CLR Write 1 to clear ISP_BUF_FULL_INT. (WT)

ISP_HVNUM_SETTING_ERR_INT_CLR Write 1 to clear ISP_HVNUM_SETTING_ERR_INT. (WT)

ISP_DATA_TYPE_SETTING_ERR_INT_CLR Write 1 to clear ISP_DATA_TYPE_SETTING_ERR_INT. (WT)

ISP_MIPI_HNUM_UNMATCH_INT_CLR Write 1 to clear ISP_MIPI_HNUM_UNMATCH_INT. (WT)

ISP_DPC_CHECK_DONE_INT_CLR Write 1 to clear ISP_DPC_CHECK_DONE_INT. (WT)

ISP_GAMMA_XCOORD_ERR_INT_CLR Write 1 to clear ISP_GAMMA_XCOORD_ERR_INT. (WT)

ISP_AE_MONITOR_INT_CLR Write 1 to clear ISP_AE_MONITOR_INT. (WT)

ISP_AE_FRAME_DONE_INT_CLR Write 1 to clear ISP_AE_FRAME_DONE_INT. (WT)

ISP_AF_FDONE_INT_CLR Write 1 to clear ISP_AF_FDONE_INT. (WT)

ISP_AF_ENV_INT_CLR Write 1 to clear ISP_AF_ENV_INT. (WT)

ISP_AWB_FDONE_INT_CLR Write 1 to clear ISP_AWB_FDONE_INT. (WT)

ISP_HIST_FDONE_INT_CLR Write 1 to clear ISP_HIST_FDONE_INT. (WT)

ISP_FRAME_INT_CLR Write 1 to clear ISP_FRAME_INT. (WT)

ISP_BLC_FRAME_INT_CLR Write 1 to clear ISP_BLC_FRAME_INT. (WT)

ISP_LSC_FRAME_INT_CLR Write 1 to clear ISP_LSC_FRAME_INT. (WT)

ISP_DPC_FRAME_INT_CLR Write 1 to clear ISP_DPC_FRAME_INT. (WT)

ISP_BF_FRAME_INT_CLR Write 1 to clear ISP_BF_FRAME_INT. (WT)

ISP_DEMOSAIC_FRAME_INT_CLR Write 1 to clear ISP_DEMOSAIC_FRAME_INT. (WT)

ISP_CCM_FRAME_INT_CLR Write 1 to clear ISP_CCM_FRAME_INT. (WT)

ISP_GAMMA_FRAME_INT_CLR Write 1 to clear ISP_GAMMA_FRAME_INT. (WT)

Continued on the next page...

Register 38.143. ISP_INT_CLR_REG (0x0070)

Continued from the previous page...

ISP_RGB2YUV_FRAME_INT_CLR Write 1 to clear ISP_RGB2YUV_FRAME_INT. (WT)

ISP_SHARP_FRAME_INT_CLR Write 1 to clear ISP_SHARP_FRAME_INT. (WT)

ISP_COLOR_FRAME_INT_CLR Write 1 to clear ISP_COLOR_FRAME_INT. (WT)

ISP_YUV2RGB_FRAME_INT_CLR Write 1 to clear ISP_YUV2RGB_FRAME_INT. (WT)

ISP_TAIL_IDI_FRAME_INT_CLR Write 1 to clear ISP_TAIL_IDI_FRAME_INT. (WT)

ISP_HEADER_IDI_FRAME_INT_CLR Write 1 to clear ISP_HEADER_IDI_FRAME_INT. (WT)

ISP_CROP_FRAME_INT_CLR Write 1 to clear ISP_CROP_FRAME_INT. (WT)

ISP_WBG_FRAME_INT_CLR Write 1 to clear ISP_WBG_FRAME_INT. (WT)

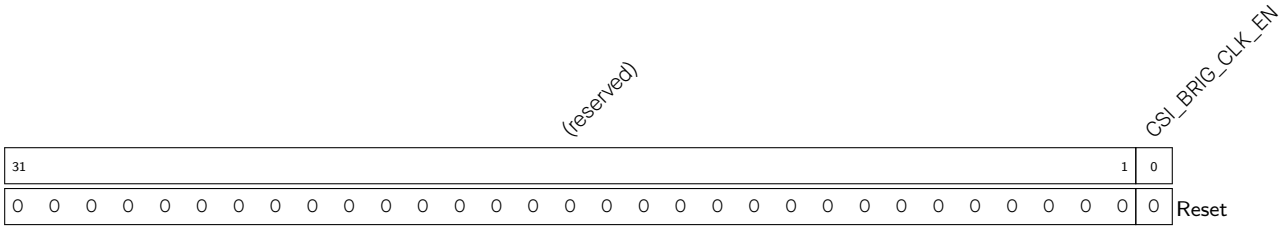
ISP_CROP_ERR_INT_CLR Write 1 to clear ISP_CROP_ERR_INT. (WT)

38.9.2 MIPI CSI_Bridge Registers

The addresses in this section are relative to **MIPI CSI_Bridge** base address provided in Table 9.3-2 in Chapter 9 *System and Memory*.

For how to program reserved fields, please refer to Section Programming Reserved Register Field.

Register 38.144. CSI_BRIG_CLK_EN_REG (0x0000)



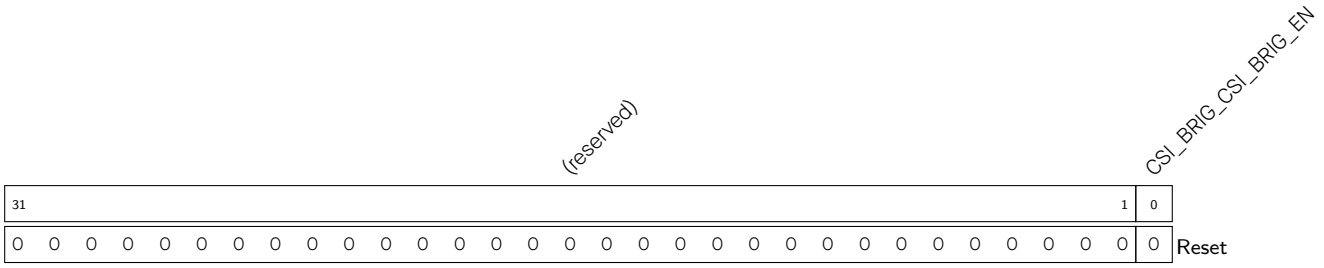
CSI_BRIG_CLK_EN Configures whether the CSI_Bridge clock is always on.

0: The clock is on only during register access

1: The clock is always on

(R/W)

Register 38.145. CSI_BRIG_CSI_EN_REG (0x0004)



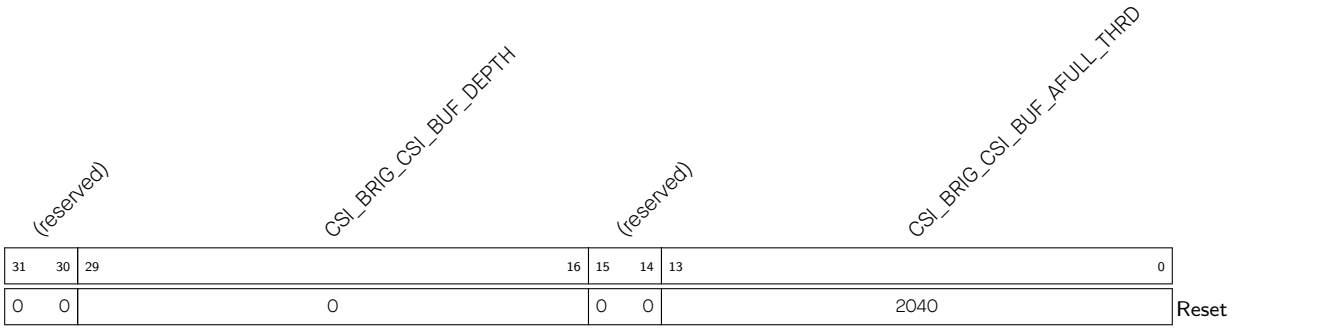
CSI_BRIG_CSI_BRIG_EN Configures whether to enable CSI_Bridge.

0: Disable

1: Enable

(R/W)

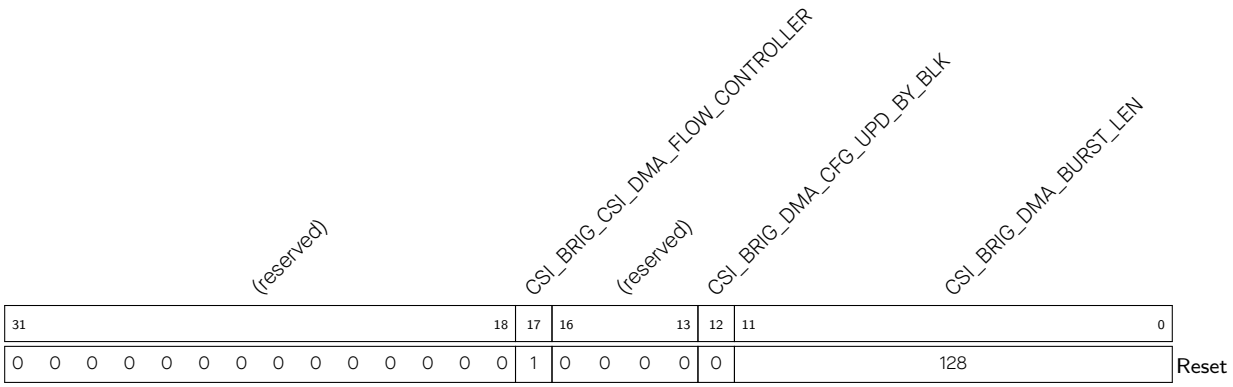
Register 38.146. CSI_BRIG_BUF_FLOW_CTL_REG (0x000C)



CSI_BRIG_CSI_BUF_AFULL_THRD Configures the size threshold at which the CSI_Bridge buffer is about to be full. (R/W)

CSI_BRIG_CSI_BUF_DEPTH Represents the size of the CSI_Bridge buffer in use. (RO)

Register 38.147. CSI_BRIG_DMA_REQ_CFG_REG (0x0008)



CSI_BRIG_DMA_BURST_LEN Configures the amount of 64-bit data in a VDMA burst transfer. (R/W)

CSI_BRIG_DMA_CFG_UPD_BY_BLK Configures when CSI_BRIG_DMA_BURST_LEN and CSI_BRIG_DMABLK_SIZE are updated.

0: Update upon completion of each VDMA block transfer

1: Update at the end of each frame transfer

(R/W)

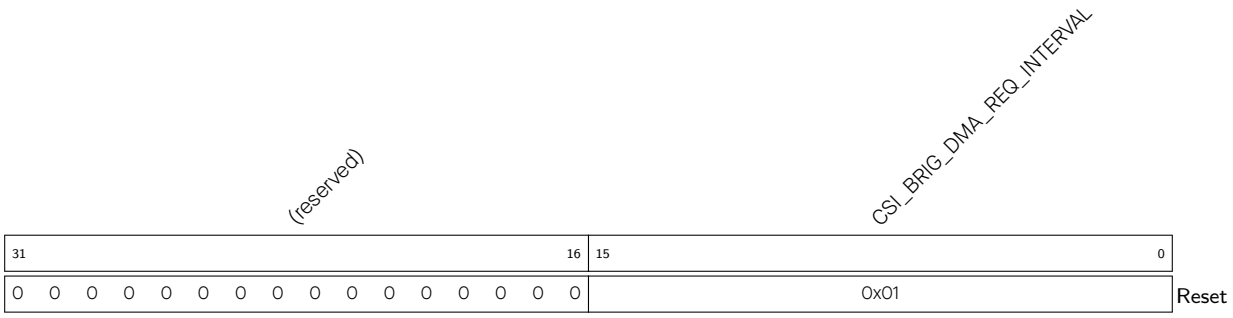
CSI_BRIG_CSI_DMA_FLOW_CONTROLLER Configures the DMA flow controller.

0: DMA acts as the flow controller

1: CSI_bridge acts as the flow controller

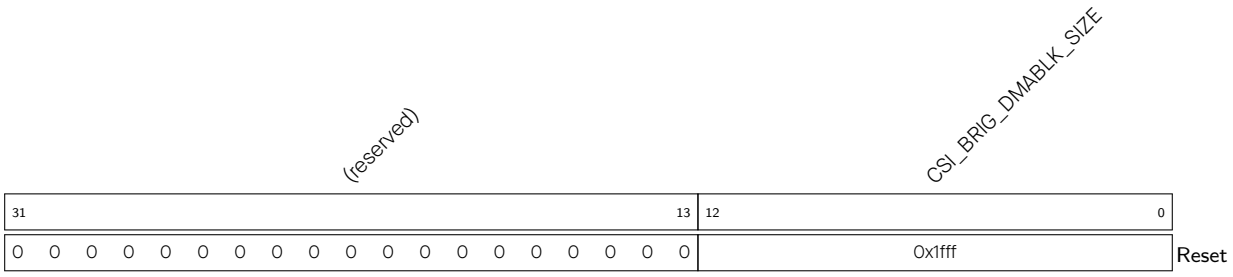
(R/W)

Register 38.148. CSI_BRIG_DMA_REQ_INTERVAL_REG (0x002C)



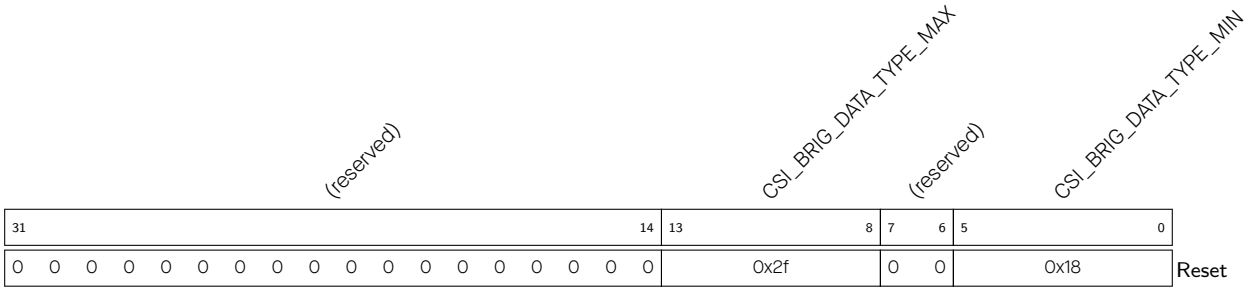
CSI_BRIG_DMA_REQ_INTERVAL Configures the interval of VDMA requests. 16'b1: 1 clock cycle, 16'b11: 2 clock cycles, 16'hFFFF: 16 clock cycles, etc. (R/W)

Register 38.149. CSI_BRIG_DMABLK_SIZE_REG (0x0030)



CSI_BRIG_DMABLK_SIZE Configures the number of VDMA bursts in a VDMA block transfer. (R/W)

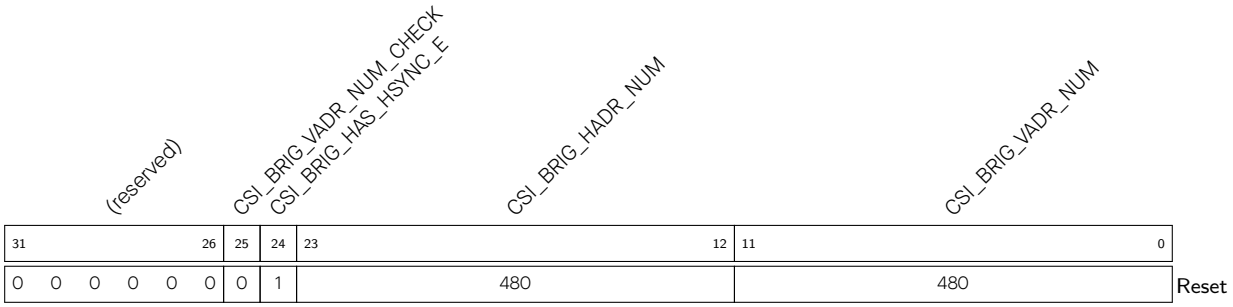
Register 38.150. CSI_BRIG_DATA_TYPE_CFG_REG (0x0010)



CSI_BRIG_DATA_TYPE_MIN Configures the minimum value of DATA_TYPE for valid pixels. (R/W)

CSI_BRIG_DATA_TYPE_MAX Configures the maximum value of DATA_TYPE for valid pixels. (R/W)

Register 38.151. CSI_BRIG_FRAME_CFG_REG (0x0014)



CSI_BRIG_VADR_NUM Configures the height of an image frame. Measurement unit: line. (R/W)

CSI_BRIG_HADR_NUM Configures the amount of 64-bit data in the width of an image frame. (R/W)

CSI_BRIG_HAS_HSYNC_E Configures whether the Image Interface 64 input includes HSYNC_START and HSYNC_END packets.

0: Not included

1: Included

(R/W)

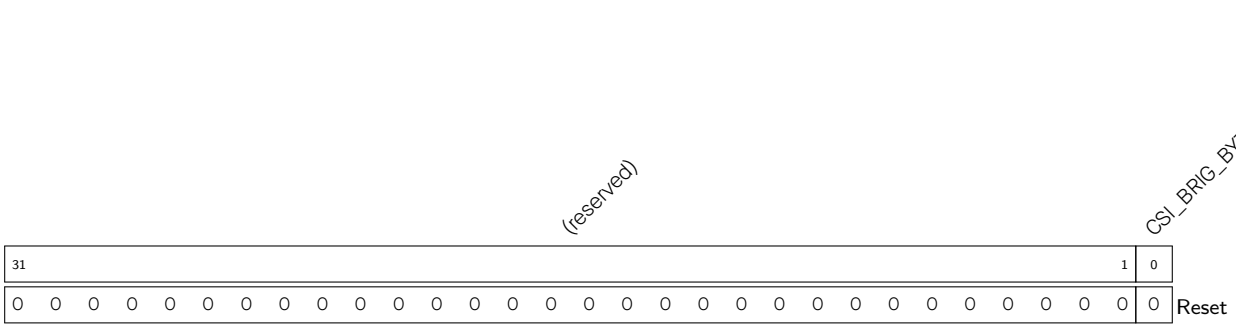
CSI_BRIG_VADR_NUM_CHECK Configures whether to enable line number checking.

0: Disable

1: Enable

(R/W)

Register 38.152. CSI_BRIG_ENDIAN_MODE_REG (0x0018)



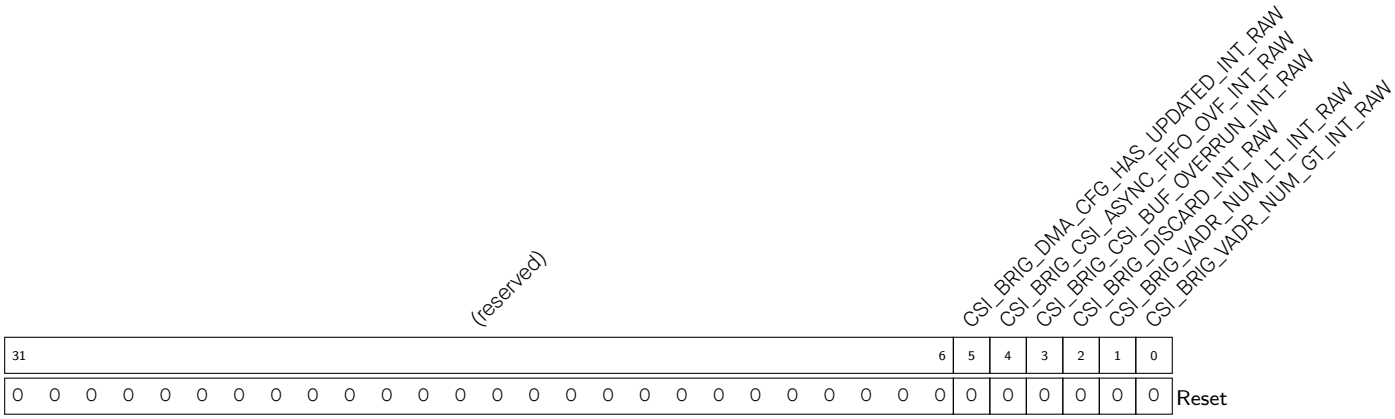
CSI_BRIG_BYTE_ENDIAN_ORDER Configures the byte order of 64-bit data.

0: Byte order remains unchanged

1: Reverse the high and low bytes

(R/W)

Register 38.153. CSI_BRIG_INT_RAW_REG (0x001C)



CSI_BRIG_VADR_NUM_GT_INT_RAW The raw interrupt status of CSI_BRIG_VADR_NUM_GT_INT. (R/WTC/SS)

CSI_BRIG_VADR_NUM_LT_INT_RAW The raw interrupt status of CSI_BRIG_VADR_NUM_LT_INT. (R/WTC/SS)

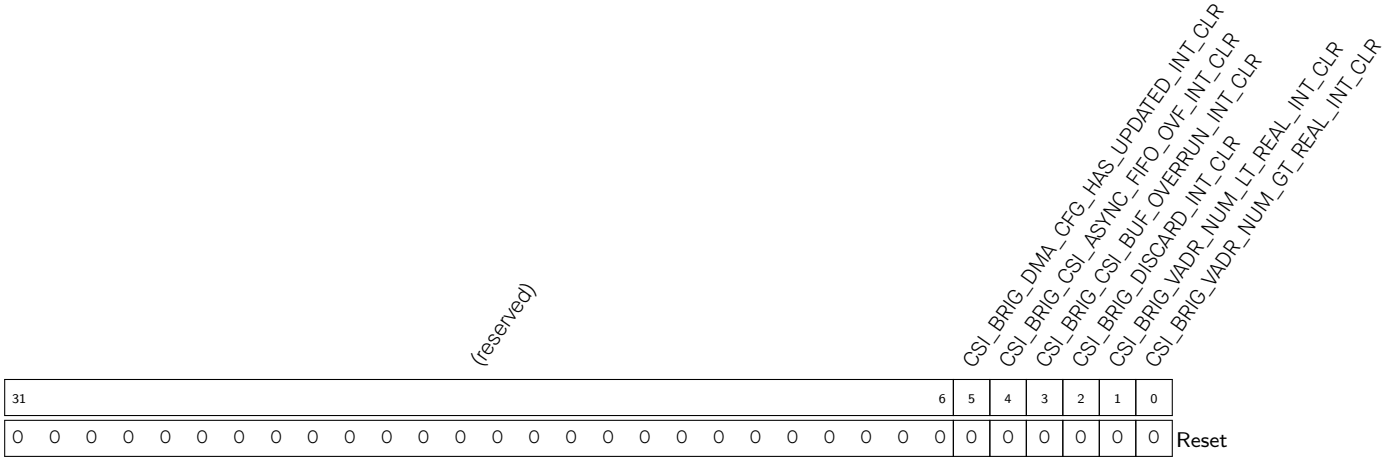
CSI_BRIG_DISCARD_INT_RAW The raw interrupt status of CSI_BRIG_DISCARD_INT. (R/WTC/SS)

CSI_BRIG_CSI_BUF_OVERRUN_INT_RAW The raw interrupt status of CSI_BRIG_CSI_BUF_OVERRUN_INT. (R/WTC/SS)

CSI_BRIG_CSI_ASYNC_FIFO_OVF_INT_RAW The raw interrupt status of CSI_BRIG_CSI_ASYNC_FIFO_OVF_INT. (R/WTC/SS)

CSI_BRIG_DMA_CFG_HAS_UPDATED_INT_RAW The raw interrupt status of CSI_BRIG_DMA_CFG_HAS_UPDATED_INT. (R/WTC/SS)

Register 38.154. CSI_BRIG_INT_CLR_REG (0x0020)



CSI_BRIG_VADR_NUM_GT_REAL_INT_CLR Write 1 to clear CSI_BRIG_VADR_NUM_GT_REAL_INT. (WT)

CSI_BRIG_VADR_NUM_LT_REAL_INT_CLR Write 1 to clear CSI_BRIG_VADR_NUM_LT_REAL_INT. (WT)

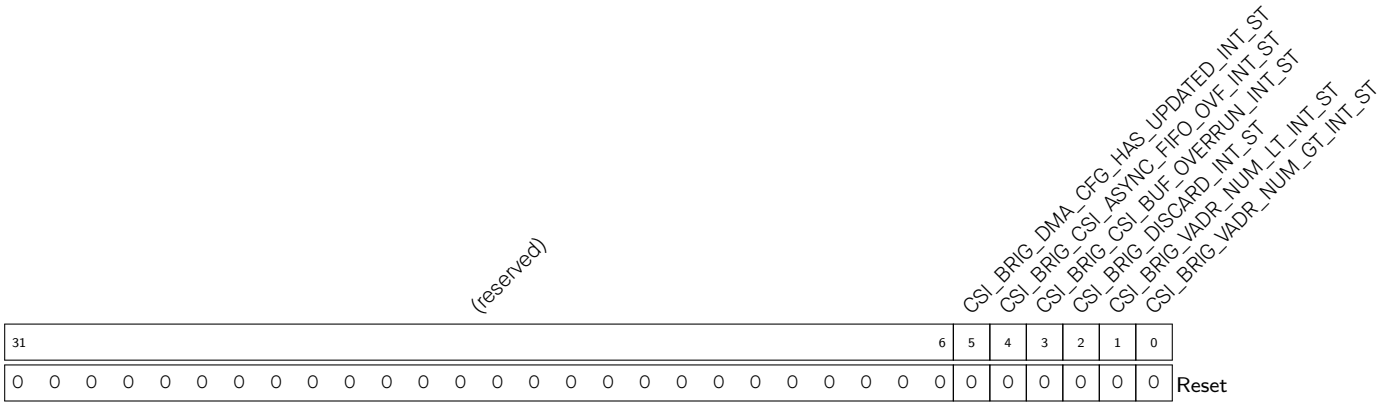
CSI_BRIG_DISCARD_INT_CLR Write 1 to clear CSI_BRIG_DISCARD_INT. (WT)

CSI_BRIG_CSI_BUF_OVERRUN_INT_CLR Write 1 to clear CSI_BRIG_CSI_BUF_OVERRUN_INT. (WT)

CSI_BRIG_CSI_ASYNC_FIFO_OVF_INT_CLR Write 1 to clear CSI_BRIG_CSI_ASYNC_FIFO_OVF_INT. (WT)

CSI_BRIG_DMA_CFG_HAS_UPDATED_INT_CLR Write 1 to clear CSI_BRIG_DMA_CFG_HAS_UPDATED_INT. (WT)

Register 38.155. CSI_BRIG_INT_ST_REG (0x0024)



CSI_BRIG_VADR_NUM_GT_INT_ST The masked interrupt status of CSI_BRIG_VADR_NUM_GT_INT. (RO)

CSI_BRIG_VADR_NUM_LT_INT_ST The masked interrupt status of CSI_BRIG_VADR_NUM_LT_INT. (RO)

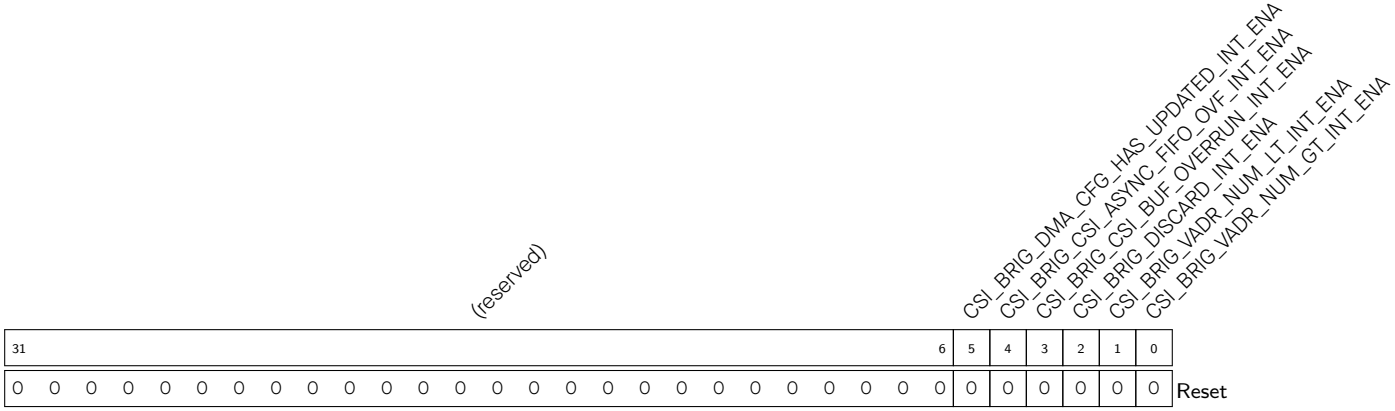
CSI_BRIG_DISCARD_INT_ST The masked interrupt status of CSI_BRIG_DISCARD_INT. (RO)

CSI_BRIG_CSI_BUF_OVERRUN_INT_ST The masked interrupt status of CSI_BRIG_CSI_BUF_OVERRUN_INT. (RO)

CSI_BRIG_CSI_ASYNC_FIFO_OVF_INT_ST The masked interrupt status of CSI_BRIG_CSI_ASYNC_FIFO_OVF_INT. (RO)

CSI_BRIG_DMA_CFG_HAS_UPDATED_INT_ST The masked interrupt status of CSI_BRIG_DMA_CFG_HAS_UPDATED_INT. (RO)

Register 38.156. CSI_BRIG_INT_ENA_REG (0x0028)



CSI_BRIG_VADR_NUM_GT_INT_ENA Write 1 to enable CSI_BRIG_VADR_NUM_GT_INT. (R/W)

CSI_BRIG_VADR_NUM_LT_INT_ENA Write 1 to enable CSI_BRIG_VADR_NUM_LT_INT. (R/W)

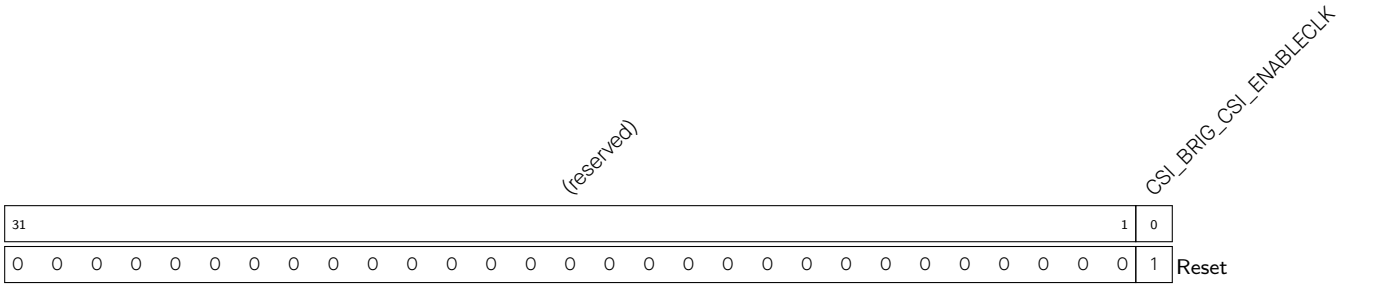
CSI_BRIG_DISCARD_INT_ENA Write 1 to enable CSI_BRIG_DISCARD_INT. (R/W)

CSI_BRIG_CSI_BUF_OVERRUN_INT_ENA Write 1 to enable CSI_BRIG_CSI_BUF_OVERRUN_INT. (R/W)

CSI_BRIG_CSI_ASYNC_FIFO_OVF_INT_ENA Write 1 to enable CSI_BRIG_CSI_ASYNC_FIFO_OVF_INT. (R/W)

CSI_BRIG_DMA_CFG_HAS_UPDATED_INT_ENA Write 1 to enable CSI_BRIG_DMA_CFG_HAS_UPDATED_INT. (R/W)

Register 38.157. CSI_BRIG_HOST_CTRL_REG (0x0040)



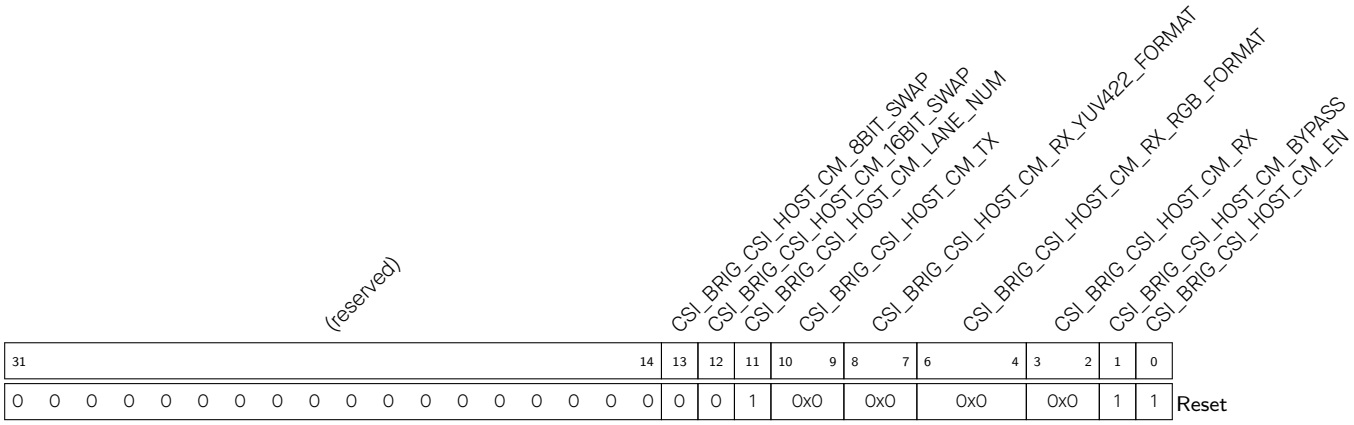
CSI_BRIG_CSI_ENABLECLK Configures whether to enable the clock lane of CSI PHY.

0: Disable

1: Enable

(R/W)

Register 38.158. CSI_BRIG_HOST_CM_CTRL_REG (0x0048)



CSI_BRIG_CSI_HOST_CM_EN Configures whether to enable CSI CM.

0: Disable. In this case, no data is output from CSI CM, and the downstream ISP cannot receive data.

1: Enable

(R/W)

CSI_BRIG_CSI_HOST_CM_BYPASS Configures whether to bypass CSI CM.

0: Do not bypass

1: Bypass

(R/W)

CSI_BRIG_CSI_HOST_CM_RX Configures the input image format of CSI CM.

0: RGB888

1: RGB565

2: YUV422

3: YUV420

(R/W)

CSI_BRIG_CSI_HOST_CM_RX_RGB_FORMAT Configures the channel order of input RGB data.

0: RGB

1: BGR

2: RGB

3: BRG

4: GRB

5: GBR

Others: Invalid

(R/W)

Continued on the next page...

Register 38.158. CSI_BRIG_HOST_CM_CTRL_REG (0x0048)

Continued from the previous page...

CSI_BRIG_CSI_HOST_CM_RX_YUV422_FORMAT Configures the channel order of input YUV422 data.

0: YVYU

1: YUYV

2: VYUY

3: UYVY

(R/W)

CSI_BRIG_CSI_HOST_CM_TX Configures the output image format of CSI CM.

0: RGB888

1: RGB565

2: YUV422

3: YUV420

(R/W)

CSI_BRIG_CSI_HOST_CM_LANE_NUM Configures the number of CSI channels; valid only when both input and output formats are RGB888.

0: 1 channel

1: 2 channels

(R/W)

CSI_BRIG_CSI_HOST_CM_16BIT_SWAP Configures whether to enable 16-bit high/low swapping.

0: Disable

1: Enable

(R/W)

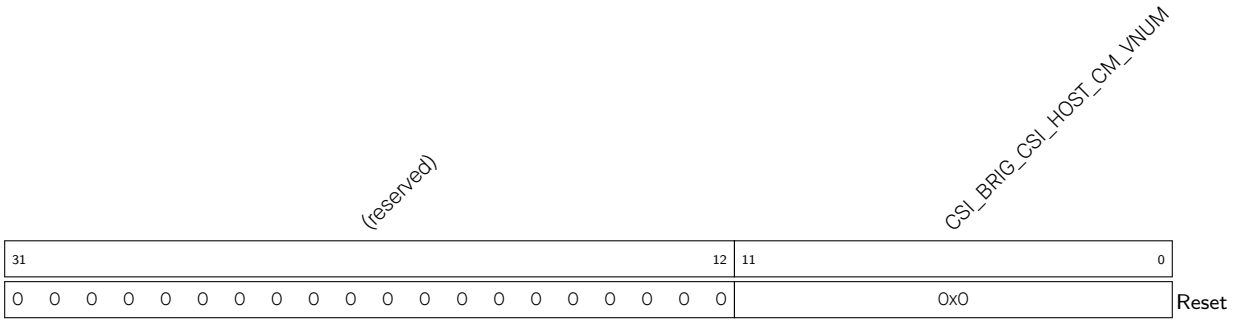
CSI_BRIG_CSI_HOST_CM_8BIT_SWAP Configures whether to enable 8-bit high/low swapping.

0: Disable

1: Enable

(R/W)

Register 38.159. CSI_BRIG_HOST_SIZE_CTRL_REG (0x004C)



CSI_BRIG_CSI_HOST_CM_HNUM Configures the horizontal size of the MIPI CSI input image interface (in units of 32-bit data). The value is line_pix_num * bits_per_pix / 32 - 1, and is valid only when converting from YUV422 to YUV420. (R/W)

Chapter 39

Pixel-Processing Accelerator (PPA)

39.1 Overview

ESP32-P4 includes a pixel-processing accelerator (PPA) designed for 2D image display. This module realizes hardware-level acceleration of image algorithms and implements functions such as image rotation, scaling, mirroring, and blending.

PPA consists of two functional modules: scaling-rotation-mirroring (SRM) and image blending (BLEND). The SRM module implements image rotation, scaling, and XY-axis mirroring functions, while the BLEND module achieves the blending of two layers of the same size based on the Alpha channel, i.e., transparency.

PPA processes images in units of pixel block. These pixel blocks are acquired from memory through 2D-DMA. The connection between PPA and 2D-DMA is shown in Figure 39.1-1.

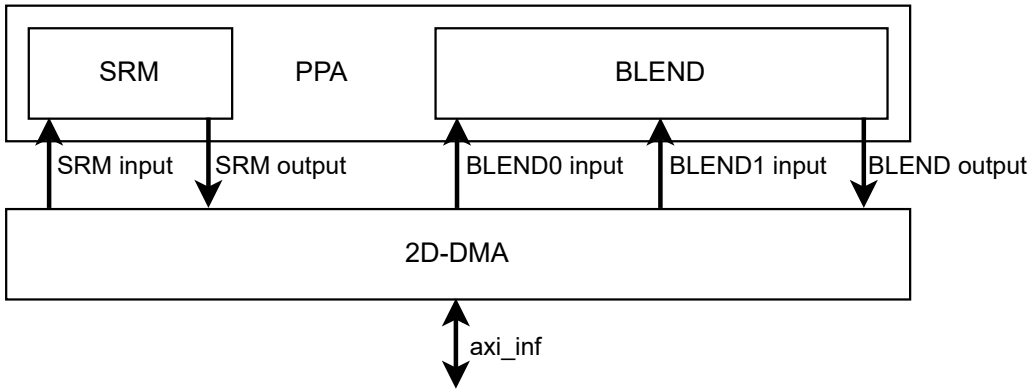


Figure 39.1-1. PPA 2D-DMA Connection

39.2 Terminology

To better illustrate the functions of the PPA module, the following terms are used in this chapter:

image	A complete image stored in the system memory
image block	A portion cropped from an image at a certain size, with the maximum size equivalent to the entire image. i.e., hb, vb size under 2D-DMA 2D-MODO.
pixel block	A portion cropped from an image block. SRM reads and writes data in pixel blocks.

39.3 Features

- Image rotation, scaling, and mirroring by SRM:

  - Input formats: ARGB8888, RGB888, RGB565, YUV422, YUV420, GRAY
  - Output formats: ARGB8888, RGB888, RGB565, YUV422, YUV420, GRAY
  - Counterclockwise rotation angles: 0°, 90°, 180°, 270°
  - Horizontal and vertical scaling with scaling factors of 8-bit integer part and 4-bit fractional part
  - Horizontal and vertical mirroring
- Blending two layers of the same size and filling images with specific pixels by BLEND:
  - Foreground input formats: ARGB8888, RGB888, RGB565, L4, L8, A4, A8
  - Background input formats: ARGB8888, RGB888, RGB565, YUV422, YUV420, GRAY, L4, L8
  - Output formats: ARGB8888, RGB888, RGB565, YUV422, YUV420, GRAY
  - Layer blending based on the Alpha channel. If layers lack an Alpha channel, it can be provided through register configuration.
  - Special color filtering by setting color-key ranges of foreground and background layers

39.4 Architectural Overview

As shown in figure 39.4-1, there are two independent functional modules in PPA: SRM and BLEND.

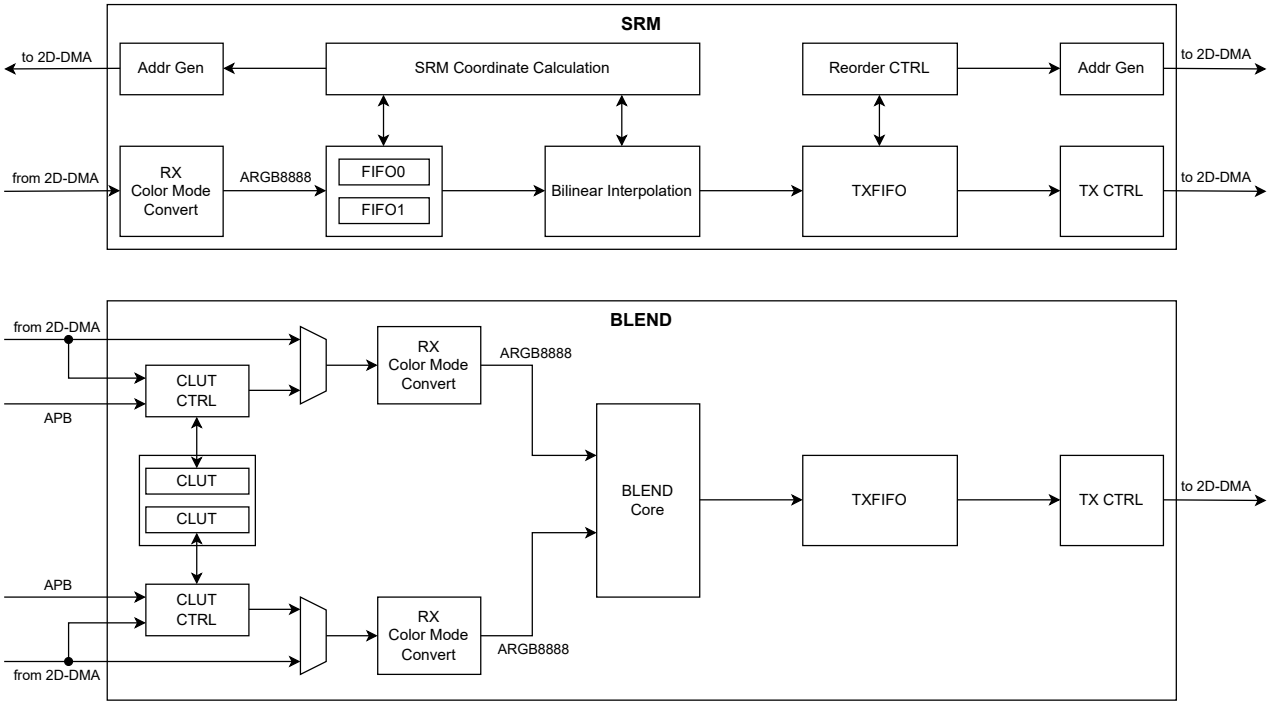


Figure 39.4-1. PPA Architecture

For SRM, SRM Coordinate Calculation scans coordinates of the output pixel blocks and calculates coordinates of the corresponding input pixel blocks. Addr Gen calculates the address according to the input pixel block coordinates and sends them to 2D-DMA. After that, the PPA receives the pixel block data from 2D-DMA, converts it into ARGB8888 format according to the register configuration, and stores it into the RX FIFO (a ping-pong FIFO composed of two SRAMs, each of which can store the whole pixel block). When a complete

pixel block is cached, Bilinear Interpolation reads the input pixels from RX FIFO, interpolates them to get the output pixels, and outputs them to TX FIFO. Reorder CTRL then calculates the output pixel block coordinates and address according to the register configuration and sends them to 2D-DMA. At the same time, Reorder CTRL takes out the corresponding pixels from TX FIFO according to the output pixel block coordinates and sends them to 2D-DMA, which saves them to the corresponding address.

For BLEND, PPA also receives data from 2D-DMA and converts it to ARGB8888 format based on the input data type. Yet the converted pixels will be processed directly by the BLEND Core algorithm module, enter the TX FIFO, and ultimately output to the system memory through 2D-DMA. Unlike SRM, BLEND does not need to calculate the address itself as it is completely controlled by 2D-DMA through a linked list.

39.5 Functional Description

39.5.1 PPA Color Space

PPA supports both true color and pseudo color. True color refers to a color representation in which each pixel in a color image consists of three primary color components: R (red), G (green), and B (blue). Each of these components directly determines the intensity of the corresponding primary color on the display device. The true color formats supported by PPA include ARGB8888, RGB888, and RGB565.

Pseudo color refers to a color representation in which the color of each pixel in a color image is not directly determined by the numerical values of the primary color components. Instead, the pixel value serves as an entry into a Color Look-Up Table (CLUT) to retrieve the corresponding A, R, G, and B values. PPA supports CLUT entry addresses with 8-bit or 4-bit width.

In addition, SRM supports the YUV422, YUV420, and GRAY data formats. The storage order of supported color formats in memory is shown in Table 39.5-1. The input YUV422 storage order can be set using the SRM or BLEND YUV422_BYTE_ORDER register; the output, however, is always stored with YUV422_BYTE_ORDER = 1.

Table 39.5-1. Pixel Storage Order

Color Format	Start Addr + 3	Start Addr + 2	Start Addr + 1	Start Addr + 0
ARGB8888	A0[7:0]	R0[7:0]	G0[7:0]	B0[7:0]
RGB888	B1[7:0]	R0[7:0]	G0[7:0]	B0[7:0]
	G2[7:0]	B2[7:0]	R1[7:0]	G1[7:0]
	R3[7:0]	G3[7:0]	B3[7:0]	R2[7:0]
RGB565	R1[4:0]G1[5:3]	G1[2:0]B1[4:0]	R0[4:0]G0[5:3]	G0[2:0]B0[4:0]
L8	L3[7:0]	L2[7:0]	L1[7:0]	L0[7:0]
L4	L7[3:0]L6[3:0]	L5[3:0]L4[3:0]	L3[3:0]L2[3:0]	L1[3:0]L0[3:0]
A8	A3[7:0]	A2[7:0]	A1[7:0]	A0[7:0]
A4	A7[3:0]A6[3:0]	A5[3:0]A4[3:0]	A3[3:0]A2[3:0]	A1[3:0]A0[3:0]
YUV422 YUV422_BYTE_ORDER=0	Y1[7:0]	V1[7:0]	Y0[7:0]	U0[7:0]

Continued on the next page...

Table 39.5-1 – ...Continued from the previous page

Color Format	Start Addr + 3	Start Addr + 2	Start Addr + 1	Start Addr + 0
YUV422 YUV422_BYTE_ORDER=1	Y1[7:0]	U1[7:0]	Y0[7:0]	V0[7:0]
YUV422 YUV422_BYTE_ORDER=2	v1[7:0]	y1[7:0]	U0[7:0]	Y0[7:0]
YUV422 YUV422_BYTE_ORDER=3	U1[7:0]	Y1[7:0]	V0[7:0]	Y0[7:0]
YUV420 Odd rows such as 1, 3, 5...	U2[7:0]	Y1[7:0]	Y0[7:0]	U0[7:0]
	Y4[7:0]	U4[7:0]	Y3[7:0]	Y2[7:0]
	Y7[7:0]	Y6[7:0]	U6[7:0]	Y5[7:0]
YUV420 Even rows such as 2, 4, 6...	V2[7:0]	Y1[7:0]	Y0[7:0]	V0[7:0]
	Y4[7:0]	V4[7:0]	Y3[7:0]	Y2[7:0]
	Y7[7:0]	Y6[7:0]	V6[7:0]	Y5[7:0]
GRAY	P3[7:0]	P2[7:0]	P1[7:0]	P0[7:0]

39.5.1.1 SRM Color Space

SRM supports ARGB8888, RGB888, RGB565, and YUV422, YUV420, and GRAY formats for both input and output. The input format is configured by PPA_SRM_RX_CM, while the output format is configured by PPA_SRM_TX_CM. The SRM input pixel processing steps are shown in Figure 39.5-1.

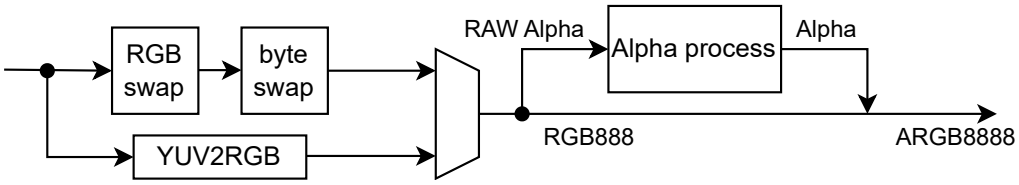


Figure 39.5-1. PPA SRM Input Pixel Processing

SRM supports RGB and byte swapping for ARGB8888, RGB888, and RGB565 input formats. The specific configurations are as follows:

- Set PPA_SRM_RX_RGB_SWAP_EN to 1 to reverse RGB to BGR and ARGB to BGRA.
- Set PPA_SRM_RX_BYTE_SWAP_EN to 1 to reverse bytes 0, 1 and 2, 3 respectively, i.e., {0, 1, 2, 3} will be reverted to {1, 0, 3, 2}. RGB888 would not be affected by this configuration as it only has three bytes per pixel.

SRM uses ARGB8888 when processing images internally. When the input lacks an Alpha channel, the default Alpha channel value for the image is set to 255. The Alpha channel can be further configured using the PPA_SRM_FIX_ALPHA_REG register, as shown in Figure 39.5-2. For specific configuration effects, please refer to Table 39.5-2.

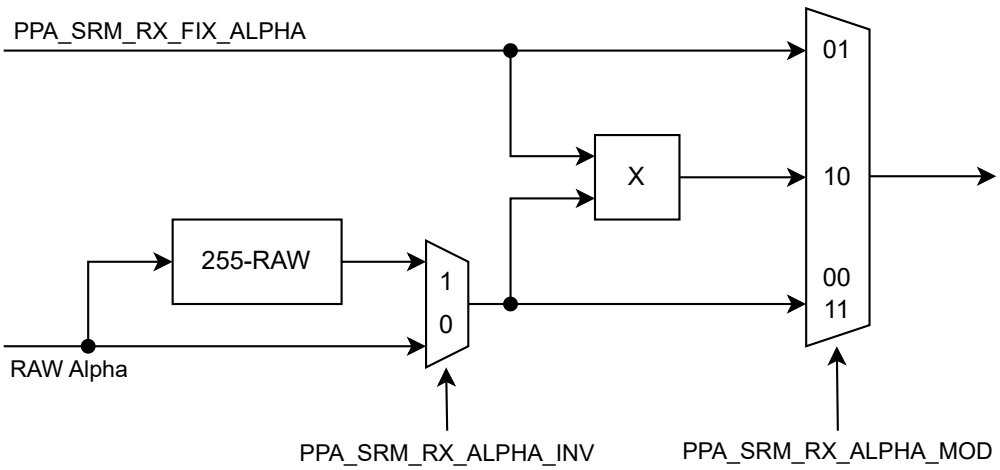


Figure 39.5-2. PPA SRM Alpha Configuration Logic

Table 39.5-2. PPA SRM Alpha Configuration

PPA_SRM_RX_ALPHA_INV	PPA_SRM_RX_ALPHA_MOD	Output
0	00/11	Original Alpha value
	01	The configuration value of PPA_SRM_RX_FIX_ALPHA
	10	The high 8 bits of the value obtained by multiplying the original Alpha value with PPA_SRM_RX_FIX_ALPHA
1	00/11	The value obtained by subtracting the original Alpha value from 255
	01	The configuration value of PPA_SRM_RX_FIX_ALPHA
	10	The high 8 bits of the value obtained by multiplying the result of subtracting the original Alpha value from 255 by PPA_SRM_RX_FIX_ALPHA

Note that when the input or output format is YUV, further configuration of YUV to RGB color conversion is required.

When the input format is YUV420, the input YUV data range needs to be configured as full-range or limit-range via the PPA_YUV_RX_RANGE field, and the protocol used for YUV to RGB conversion needs to be configured as BT601 or BT709 via the PPA_YUV2RGB_PROTOCAL field. The conversion formulas are as follows:

- YUV_{full} to YUV_{limit}
  - $Y_{limit} = \frac{220}{256} Y_{full} + 16$
  - $U_{limit} = \frac{225}{256} U_{full} + 16$
  - $V_{limit} = \frac{225}{256} V_{full} + 16$
- YUV_{limit} to RGB in BT601
  - $R = \frac{298}{256} Y_{limit} + \frac{409}{256} V_{limit} - \frac{56906}{256}$
  - $G = \frac{298}{256} Y_{limit} - \frac{100}{256} U_{limit} - \frac{208}{256} V_{limit} + \frac{34707}{256}$
  - $B = \frac{298}{256} Y_{limit} + \frac{516}{256} U_{limit} - \frac{70836}{256}$

- YUV_{limit} to RGB in BT709
  - $R = \frac{298}{256}Y_{limit} + \frac{459}{256}V_{limit} - \frac{63367}{256}$
  - $G = \frac{298}{256}Y_{limit} - \frac{55}{256}U_{limit} - \frac{136}{256}V_{limit} + \frac{19681}{256}$
  - $B = \frac{298}{256}Y_{limit} + \frac{541}{256}U_{limit} - \frac{73918}{256}$

When the output format is YUV420, the output YUV data range needs to be configured as full-range or limit-range via the PPA_YUV_TX_RANGE field, and the protocol used for YUV to RGB conversion needs to be configured as BT601 or BT709 via the PPA_RGB2YUV_PROTOCAL field. The conversion formulas are as follows:

- YUV_{limit} to YUV_{full}
  - $Y_{full} = \frac{298}{256}Y_{limit} + \frac{4768}{256}$
  - $U_{full} = \frac{291}{256}U_{limit} + \frac{4550}{256}$
  - $V_{full} = \frac{291}{256}V_{limit} + \frac{4550}{256}$
- RGB to YUV_{limit} in BT601
  - $Y_{limit} = \frac{4096}{256} + \frac{66}{256}R + \frac{129}{256}G + \frac{25}{256}B$
  - $U_{limit} = \frac{32768}{256} - \frac{38}{256}R - \frac{74}{256}G + \frac{112}{256}B$
  - $V_{limit} = \frac{32768}{256} + \frac{112}{256}R - \frac{94}{256}G - \frac{18}{256}B$
- RGB to YUV_{limit} in BT709
  - $Y_{limit} = \frac{4096}{256} + \frac{47}{256}R + \frac{157}{256}G + \frac{16}{256}B$
  - $U_{limit} = \frac{32768}{256} - \frac{26}{256}R - \frac{86}{256}G + \frac{112}{256}B$
  - $V_{limit} = \frac{32768}{256} + \frac{112}{256}R - \frac{102}{256}G - \frac{10}{256}B$

39.5.1.2 BLEND Color Space

BLEND supports ARGB8888, RGB888, RGB565, L4, L8, A4, and A8 foreground input formats, and ARGB8888, RGB888, RGB565, YUV422, YUV420, GRAY, L4, L8 background input formats. The input color format of the foreground and background layers can be configured respectively via the PPA_BLEND0/1_RX_CM field. Among the above input formats, A4 and A8 only support the foreground layer, and when the input format is L4 or A4, the size of the image block, hb, and the offset in the image, x, must be an even number. For the output format, BLEND supports ARGB8888, RGB888, and RGB565, which can be configured via the PPA_BLEND_TX_CM field.

BLEND's RGB conversion, byte conversion, and alpha channel configuration are consistent with SRM input. Additionally, when the foreground layer data format is A4 or A8, the input data will be Alpha channel values. In this case, if PPA_BLEND_BYPASS is set as 1, the foreground layer's input Alpha value replaces the background layer's Alpha channel and output. If set as 0, it will be a normal BLEND operation, and the foreground layer's RGB values will be specified by PPA_BLEND_RGB_REG.

When the input data format is L4 or L8, it requires to first initialize the CLUT for BLEND. BLEND has two CLUTs with a depth of 256 and a width of 32. There are two initialization modes: FIFO and MEM.

- Write 0 to PPA_APB_FIFO_MASK to enter the FIFO mode.

In the FIFO mode, access to CLUT is achieved by reading/writing PPA_RDWR_WORD_BLENDx_CLUT, where 0 corresponds to the background channel CLUT and 1 corresponds to the foreground channel CLUT.
Each time this register is read/written, the corresponding read/write address is automatically incremented by 1. The read/write address can be reset by writing 1 and then 0 to PPA_BLENDx_CLUT_MEM_RST. The read address can be reset by writing 1 then 0 to PPA_BLENDx_CLUT_MEM_RDADDR_RST.

- Write 1 to PPA_APB_FIFO_MASK to enter the MEM mode.
  In the MEM mode, CLUT can be directly accessed through addresses. The CLUT addresses in the PPA's bus address range are as shown in Table 39.5-3.

Table 39.5-3. BLEND CLUT Address

addr[11:10]	addr[9:2]	addr[1:0]
01: Select BLEND0 CLUT 10: Select BLEND1 CLUT	CLUT actual address, 0 ~ 255	00

39.5.2 PPA 2D-DMA Linked List Configuration

PPA exchanges data with the system memory via 2D-DMA. For both SRM and BLEND, it is necessary to configure the 2D-DMA linked list information for the required image block operations in one go. The 2D-DMA linked list is shown in Figure 39.5-3. For detailed information, please refer to 8 *2D-DMA Controller (2D-DMA)*.

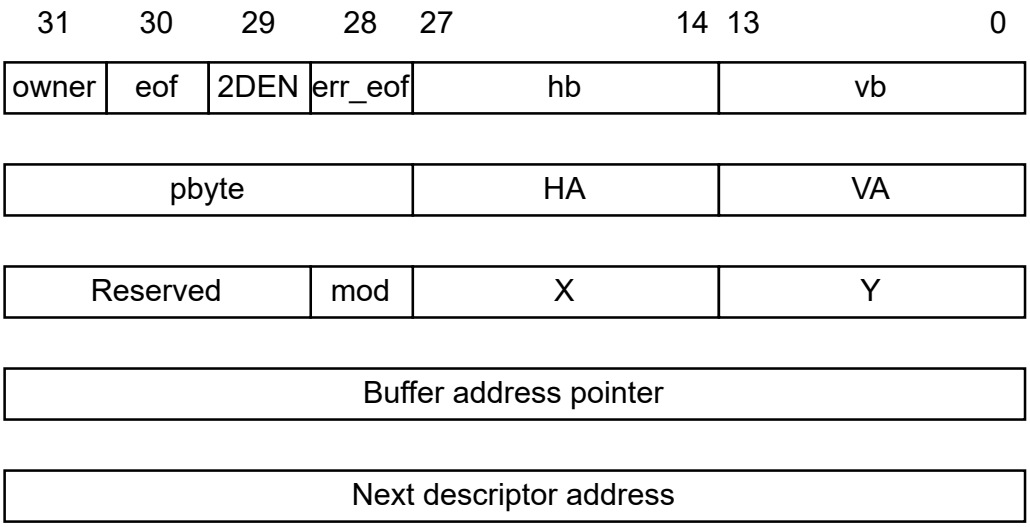


Figure 39.5-3. 2D-DMA Linked List

The corresponding linked list parameters of PPA are shown in Table 39.5-4.

Table 39.5-4. PPA Linked List Configuration

Field	Configuration
owner	1
eof	1
2DEN	1
err_eof	0
hb	Horizontal size of the image block. Must be even if the input or output data format is YUV420
vb	Vertical size of the image block. Must be even if the input or output data format is YUV420
pbyte	ARGB8888: 5 (4 byte/pixel) RGB888: 4 (3 byte/pixel) RGB565: 3 (2 byte/pixel) YUV420: 2 (1.5 byte/pixel) A8/L8: 1 (1 byte/pixel) A4/L4: 0 (0.5 byte/pixel)
HA	Horizontal size of the entire image. Must be even if the input or output data format is YUV420
VA	Vertical size of the entire image. Must be even if the input or output data format is YUV420
mod	0
X	Horizontal offset of the image block within the image (offset starts from 0). Must be even if the input or output data is YUV420
Y	Vertical offset of the image block within the image (offset starts from 0). Must be even if the input or output data is YUV420
Buffer address pointer	Starting address of the image
Next descriptor address	N/A

39.5.2.1 SRM Special Configuration

Once the 2D-DMA linked lists for input and output image blocks are configured, SRM needs to select the required pixel blocks from the input image block memory space and process them in units of pixel blocks. The processed pixel blocks are then stored in the corresponding memory space of the output image block. Throughout this process, the starting address of each pixel block needs to be controlled by SRM. To realize this functionality, write 1 to the DMA2D_IN/OUT_DSCR_PORT_EN_CHx field of the corresponding channel of the 2D-DMA to enable the descriptor interface (also called DSCR-PORT in 2D-DMA), allowing PPA to control

the starting address of the next pixel block in real-time.

For the data input, SRM must receive a complete pixel block each time. When the pixel block selection reaches the boundary of the image block, 2D-DMA needs to automatically pad the pixel block beyond the boundary. This can be achieved by configuring the DMA2D_OUT_DSCR_PORT_BLK_V/H_CHx field of the corresponding channel of 2D-DMA to the pixel block size. The required values depend on the image format, as shown in Table 39.5-5 below.

Table 39.5-5. Automatic Padding Block Size Configuration

PPA_SRM_BK_SIZE_SEL	Pixel Format	DMA2D_OUT_DSCR_PORT_BLK_H_CHx	DMA2D_OUT_DSCR_PORT_BLK_V_CHx
0	ARGB8888 RGB888 RGB565 GRAY	34	34
0	YUV422	36	34
0	YUV420	36	36
1	ARGB8888 RGB888 RGB565 GRAY	18	18
1	YUV422	20	18
1	YUV420	20	20

Note:
For optimal memory-access performance, use the default setting PPA_SRM_BK_SIZE_SEL = 0.

39.5.3 Scaling - Rotation - Mirroring (SRM)

39.5.3.1 Basic Functionality

SRM supports scaling, rotation, and mirroring. It first scales the image based on the origin point, then rotates it around the center point (X_{middle}, Y_{middle}), and finally mirrors it horizontally and vertically based on the center point (X_{target}, Y_{target}), as shown in Figure 39.5-4. Scaling, rotation, and mirroring parameters are all configured through registers:

- PPA_SRM_SCAL_X_INT: Integer part of the horizontal scaling factor
- PPA_SRM_SCAL_X_FRAG: Fractional part of the horizontal scaling factor
- PPA_SRM_SCAL_Y_INT: Integer part of the vertical scaling factor
- PPA_SRM_SCAL_Y_FRAG: Fractional part of the vertical scaling factor
- PPA_SRM_ROTATE_ANGLE: Counterclockwise rotation angle
- PPA_SRM_MIRROR_X: Enable horizontal mirroring
- PPA_SRM_MIRROR_Y: Enable vertical mirroring

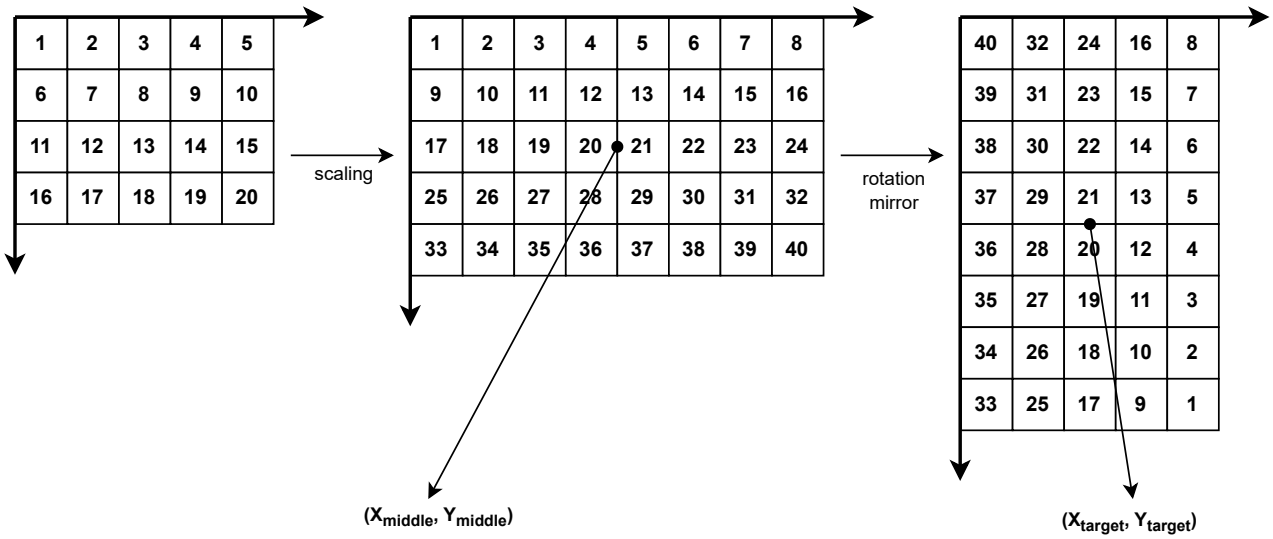


Figure 39.5-4. SRM Procedure

39.5.3.2 Pixel Block Rearrangement

When the output pixel block is small, initiating a 2D-DMA transfer for each pixel block would result in low bus bandwidth utilization. To improve bus bandwidth utilization, SRM can cache and concatenate output pixel blocks into a larger pixel block before outputting them all at once. This feature is enabled by default and can be disabled by setting PPA_SRM_MACRO_BK_RO_BYPASS to 1.

Once pixel block rearrangement is enabled, under certain conditions, the output pixel blocks will be automatically rearranged and concatenated, as specified in Table 39.5-6.

- For the output pixel block in ARGB8888/RGB888 format, if its vertical height is less than or equal to 32, and the horizontal width is less than or equal to 16, pixel block rearrangement will be automatically enabled.
- For the output pixel block in RGB565 format, if its vertical height is less than or equal to 32, and the horizontal width is less than or equal to 32, pixel block rearrangement will be automatically enabled.
- For the output pixel block in YUV420 format, if its vertical height is less than or equal to 32, and horizontal width is less than or equal to 32, pixel block rearrangement will be automatically enabled.

Table 39.5-6. Number of Pixel Block Rearrangements in SRM Output

ARGB8888/RGB888		RGB565		YUV420	
Horizontal Size	Rearrangemer Number	Horizontal Size	Rearrangemer Number	Horizontal Size (Even)	Rearrangement Number
1	32	1	64	2	32
2	16	2	32	4	16
3	10	3	20	6	10
4	8	4	16	8	8
5	6	5	12	10	6
6	5	6	10	12	5
7	4	7	9	14	4

Continued on the next page...

Table 39.5-6 – ...Continued from the previous page

ARGB8888/RGB888		RGB565		YUV420	
Horizontal Size	Rearrangemer Number	Horizontal Size	Rearrangemer Number	Horizontal Size (Even)	Rearrangement Number
8	4	8	8	16	4
9	3	9	7	18	3
10	3	10	6	20-32	2
11	2	11	5		
12	2	12	5		
13	2	13	4		
14	2	14	4		
15	2	15	4		
16	2	16	4		
		17	3		
		18	3		
		19-32	2		

39.5.4 Layer Blending (BLEND)

39.5.4.1 Basic Functionality

BLEND combines two image blocks of the same size using the Alpha channel and then outputs the result. The blending formula is as follows, where Ab is the Alpha channel of the background layer, Af is the Alpha channel of the foreground layer, Cb corresponds to the R, G, and B components of the background layer, and Cf corresponds to the R, G, B components of the foreground layer:

- $Aout = Ab + Af - Ab * Af$
- $Cout = (Cb * Ab * (1 - Af) + Cf * Af)/(Ab + Af - Ab * Af)$

Additionally, BLEND supports color-keying based on pixel color. By configuring color ranges via PPA_CK_FG/BG_HIGH/LOW_REG, BLEND allows for image segmentation into four distinct regions, each subjected to different operations:

- When the foreground pixel falls within the color-key range and the background pixel falls outside the color-key range, output the background pixel.
- When the background pixel falls within the color-key range and the foreground pixel falls outside the color-key range, the output depends on the value of PPA_COLORKEY_FG_BG_REVERSE. If 0, output the background pixel; if 1, output the foreground pixel.
- When both the background and foreground pixels fall within the color-key range, the output is set to the color defined by PPA_COLORKEY_DEFAULT_R/G/B.
- When neither the background nor foreground pixels fall within the color-key range, output follows the normal Alpha Blending process.

When PPA_BLEND_BYPASS is set to 1, BLEND directly outputs the background layer data. If the foreground input data type is A4/A8 at this time, the foreground input data will replace the background input data's Alpha channel and then be output.

39.5.4.2 Filled Image Output

BLEND also supports filled image output. When this feature is enabled, only the output side of BLEND works, continuously outputting the filled image. The size and data of the filled image can be configured via PPA_BLEND_TX_SIZE_REG and PPA_BLEND_TX_FIX_PIXEL.

39.6 Interrupts

ESP32-P4's PPA can generate the following interrupt signal that will be sent to the Interrupt Matrix.

- PPA_INTR

The following interrupt sources can generate PPA_INTR interrupt signal:

- PPA_SRM_PARAM_CFG_ERR_INT: Triggered when invalid parameters are configured in the SRM mode. You can acquire the specific error type through PPA_SRM_PARAM_ERR_ST_REG, as shown in Table 39.6-1.
- PPA_BLEND_PARAM_CFG_ERR_INT: Triggered when invalid parameters are configured in the BLEND mode. You can acquire the specific error type through PPA_BLEND_ST_REG, as shown in Table 39.6-2.
- PPA_SRM_EOF_INT: Triggered when SRM workflow reaches EOF.
- PPA_BLEND_EOF_INT: Triggered when BLEND workflow reaches EOF.

Table 39.6-1. SRM Parameter Error Types

Error Type	Details
PPA_TX_DSCR_VB_ERR_ST	The sum of the vertical size of the output image block from SRM and the vertical offset Y specified in the output descriptor exceeds the total vertical size of the image configured in the output descriptor (VA)
PPA_TX_DSCR_HB_ERR_ST	The sum of the horizontal size of the output image block from SRM and the vertical offset X specified in the output descriptor exceeds the total horizontal size of the image configured in the output descriptor (HA)
PPA_Y_RX_SCAL_EQUAL_0_ERR_ST	The vertical scaling factor for the input image block in SRM is set to 0
PPA_RX_DSCR_VB_ERR_ST	The sum of the vertical size VB and the vertical offset Y in the SRM input descriptor exceeds the overall vertical size VA in the output descriptor
PPA_YDST_LEN_TOO_SAMLL_ERR_ST	The vertical size of the image block after SRM scaling is 0. For example, if the vertical size of the image block is 14 and the vertical scaling factor is 1/16 without rotation, the scaled vertical size of the image block becomes 0
PPA_YDST_LEN_TOO_LARGE_ERR_ST	The vertical size of the image block after SRM scaling exceeds 8191
PPA_X_RX_SCAL_EQUAL_0_ERR_ST	The horizontal scaling factor for the input image block in SRM is 0

Continued on the next page...

Table 39.6-1 – ...Continued from the previous page

Error Type	Details
PPA_RX_DSCR_HB_ERR_ST	The sum of the horizontal size HB of the input image block configured in the SRM input descriptor and the horizontal offset X exceeds the entire image's horizontal size HA configured in the output descriptor
PPA_XDST_LEN_TOO_SAMLL_ERR_ST	The horizontal size of the image block after SRM scaling is 0
PPA_XDST_LEN_TOO_LARGE_ERR_ST	The horizontal size of the image block after SRM scaling exceeds 8191
PPA_X_YUV420_RX_SCALE_ERR_ST	The input descriptor parameters HA/hb/X happen to be odd numbers when the SRM input format is YUV420
PPA_Y_YUV420_RX_SCALE_ERR_ST	The input descriptor parameters VA/vb/Y happen to be odd numbers when the SRM input format is YUV420
PPA_X_YUV420_TX_SCALE_ERR_ST	The output descriptor parameters HA/hb/X happen to be odd numbers when the SRM output format is YUV420
PPA_Y_YUV420_TX_SCALE_ERR_ST	The output descriptor parameters VA/vb/Y happen to be odd numbers when the SRM output format is YUV420

Table 39.6-2. BLEND Parameter Error Types

Error Type	Details
PPA_BLEND_SIZE_DIFF_ST	The BLEND foreground and background sizes do not match
PPA_BLEND_YUV_X_SCALE_ERR_ST	The BLEND background image format is YUV422 or YUV420, the horizontal size is an odd number
PPA_BLEND_YUV_Y_SCALE_ERR_ST	The BLEND background image format is YUV420, the vertical size is an odd number

39.7 Programming Procedures

39.7.1 PPA Clock Reset Configuration

1. Enable the clock of the PPA and 2D-DMA modules:
   - Write 1 to HP_SYS_CLKRST_PPA_SYS_CLK_EN to enable the PPA clock
   - Write 1 to HP_SYS_CLKRST_DMA2D_SYS_CLK_EN to enable the 2D-DMA clock
2. Reset the PPA and 2D-DMA modules:
   - Write 1 and then 0 to HP_SYS_CLKRST_RST_EN_PPA reset PPA
   - Write 1 and then 0 to HP_SYS_CLKRST_RST_EN_DMA2D reset 2D-DMA

39.7.2 SRM Configuration

1. Refer to Section 39.7.1 to enable and reset PPA and 2D-DMA.
2. Configure the inlink and outlink of 2D-DMA. For detailed parameters, please refer to Table 39.5-4.

3. Select the input and output channels of 2D-DMA:
   - Write 0x1 to DMA2D_OUT_PERI_SEL_CHx to allocate one 2D-DMA output channel to the PPA SRM input
   - Write 0x1 to DMA2D_IN_PERI_SEL_CHx to allocate one 2D-DMA input channel to the PPA SRM output
4. Enable descriptor interface for 2D-DMA channels used by PPA:
   - Write 1 to DMA2D_OUT_DSCR_PORT_EN_CHx to enable descriptor interface
   - Write 1 to DMA2D_IN_DSCR_PORT_EN_CHx to enable descriptor interface
5. Configure the descriptor interface completion parameters of 2D-DMA:
   - Configure the horizontal completion parameters via DMA2D_OUT_DSCR_PORT_BLK_H_CHx. See Table 39.5-5 for details
   - Configure the vertical completion parameters via DMA2D_OUT_DSCR_PORT_BLK_V_CHx. See Table 39.5-5 for details
6. Enable the input and output channels of 2D-DMA:
   - Write 1 to DMA2D_OUTLINK_START_CHx to enable the output channel of 2D-DMA
   - Write 1 to DMA2D_INLINK_START_CHx to enable the input channel of 2D-DMA
7. Configure SRM parameters:
   - Configure the size of image blocks to be processed via PPA_SRM_BK_SIZE_SEL
   - Configure SRM input data format via PPA_SRM_RX_CM
   - Configure SRM output data format via PPA_SRM_TX_CM
   - Configure the integer part of the horizontal scaling factor via PPA_SRM_SCAL_X_INT
   - Configure the fractional part of the horizontal scaling factor via PPA_SRM_SCAL_X_FRAG
   - Configure the integer part of the vertical scaling factor via PPA_SRM_SCAL_Y_INT
   - Configure the fractional part of the vertical scaling factor via PPA_SRM_SCAL_Y_FRAG
   - Configure the counterclockwise rotation angle via PPA_SRM_ROTATE_ANGLE
   - Enable horizontal mirroring via PPA_SRM_MIRROR_X
   - Enable vertical mirroring via PPA_SRM_MIRROR_Y
8. Write 1 to PPA_SCAL_ROTATE_START to start the SRM workflow.

39.7.3 BLEND CLUT Configuration

1. In the FIFO mode:
   - Write 0 to PPA_APB_FIFO_MASK to enter the FIFO mode
   - Write data to PPA_RDWR_WORD_BLEND0_CLUT to initialize the background layer's CLUT
   - Write data to PPA_RDWR_WORD_BLEND1_CLUT to initialize the foreground layer's CLUT
2. In the MEM mode:
   - Write 1 to PPA_APB_FIFO_MASK to enter the MEM mode

- Initialize the CLUT by writing data directly to the corresponding address, referring to Table 39.5-3

39.7.4 BLEND Configuration

1. Refer to Section 39.7.1 to enable and reset PPA and 2D-DMA.

2. Configure the input and output link lists of 2D-DMA. For detailed parameters, please refer to Table 39.5-4.

3. Select the input and output channels of 2D-DMA:

   - Write 0x2 to DMA2D_OUT_PERI_SEL_CHx to allocate one 2D-DMA output channel to PPA BLEND for background layer input
   - Write 0x3 to DMA2D_OUT_PERI_SEL_CHx to allocate one 2D-DMA output channel to PPA BLEND for foreground layer input
   - Write 0x2 to DMA2D_IN_PERI_SEL_CHx to allocate one 2D-DMA input channel to PPA BLEND for output

4. Enable the input and output channels for 2D-DMA. The x in the registers refers to the two channels allocated to BLEND foreground and background respectively:

   - Write 1 to DMA2D_OUTLINK_START_CHx to enable the output channel of 2D-DMA
   - Write 1 to DMA2D_INLINK_START_CHx to enable the input channel of 2D-DMA

5. Configure BLEND parameter:

   - Configure the data type for the background layer input of BLEND via PPA_BLEND0_RX_CM. If the format is L4/L8, initialize the CLUT first as described in 39.7.3
   - Configure the data type for the background layer input of BLEND via PPA_BLEND1_RX_CM. If the format is L4/L8, initialize the CLUT first as described in 39.7.3
   - Configure the output data format of BLEND via PPA_BLEND_TX_CM
   - Configure the upper and lower limits of the color-key for the background layer via PPA_CK_BG_HIGH_REG and PPA_CK_BG_LOW_REG
   - Configure the upper and lower limits of the color-key for the foreground layer via PPA_CK_FG_HIGH_REG and PPA_CK_FG_LOW_REG
   - Configure the default value and behavior for the color-key via PPA_CK_DEFAULT_REG
   - Write 1 to PPA_BLEND_EN to enable BLEND

6. Write 1 to PPA_BLEND_TRANS_MODE_UPDATE to start BLEND workflow

39.7.5 BLEND Image Filling Configuration

1. Refer to Section 39.7.1 to enable and reset PPA and 2D-DMA.

2. Configure the input link lists of 2D-DMA (output link lists of PPA BLEND). For detailed parameters, please refer to Table 39.5-4.

3. Write 0x2 to DMA2D_IN_PERI_SEL_CHx to allocate one 2D-DMA input channel to PPA BLEND output.

4. Write 1 to DMA2D_INLINK_START_CHx to enable the output channel of 2D-DMA.

5. Configure BLEND parameters:
   - Configure the output data format of BLEND via PPA_BLEND_TX_CM
   - Write 1 to PPA_BLEND_FIX_PIXEL_FILL_EN to enable image filling
   - Configure the pixel value for image filling via PPA_BLEND_TX_FIX_PIXEL
   - Configure the filled image size via PPA_BLEND_HB and PPA_BLEND_VB
6. Write 1 to PPA_BLEND_TRANS_MODE_UPDATE to start BLEND workflow.

39.7.6 Error Handling

- Refer to Section 39.7.1 to reset the whole PPA, and configure again.
- Or write 1 and then 0 to reset SRM and BLEND via PPA_SCAL_ROTATE_RST, PPA_BLEND_RST, and configure again.

Note:

- The 2D-DMA configurations mentioned in this chapter involve only key steps. For detailed procedures, please refer to 8 *2D-DMA Controller (2D-DMA)*.
- In this chapter, DMA2D_OUT corresponds to the PPA input direction, and DMA2D_IN corresponds to the PPA output direction.

39.8 Register Summary

The addresses in this section are relative to the Pixel-Processing Accelerator base address provided in Table 9.3-2 in Chapter 9 *System and Memory*.

The abbreviations given in Column **Access** are explained in Section *Access Types for Registers*.

Name	Description	Address	Access
Configuration Registers			
PPA_BLEND0_CLUT_DATA_REG	CLUT SRAM data read/write register of BLEND background layer	0x0000	R/W
PPA_BLEND1_CLUT_DATA_REG	CLUT SRAM data read/write register of BLEND foreground layer	0x0004	R/W
PPA_CLUT_CONF_REG	Configures BLEND CLUT	0x000C	R/W
PPA_SRM_COLOR_MODE_REG	Configures SRM color space	0x0020	R/W
PPA_BLEND_COLOR_MODE_REG	Configures BLEND color space	0x0024	R/W
PPA_SRM_BYTE_ORDER_REG	Configures SRM byte order	0x0028	R/W
PPA_BLEND_BYTE_ORDER_REG	Configures BLEND byte order	0x002C	R/W
PPA_BLEND_TRANS_MODE_REG	Configures the BLEND mode	0x0034	varies
PPA_SRM_FIX_ALPHA_REG	Configures SRM Alpha channel	0x0038	R/W
PPA_BLEND_TX_SIZE_REG	Configures BLEND image filling size	0x003C	R/W
PPA_BLEND_FIX_ALPHA_REG	Configures BLEND Alpha override	0x0040	R/W
PPA_BLEND_RGB_REG	Configures RGB color	0x0048	R/W
PPA_BLEND_FIX_PIXEL_REG	Configures BLEND image filling pixel	0x004C	R/W
PPA_CK_FG_LOW_REG	Configures the foreground color-key lower threshold of BLEND	0x0050	R/W
PPA_CK_FG_HIGH_REG	Configures the foreground color-key higher threshold of BLEND	0x0054	R/W
PPA_CK_BG_LOW_REG	Configures the background color-key lower threshold of BLEND	0x0058	R/W
PPA_CK_BG_HIGH_REG	Configures the background color-key higher threshold of BLEND	0x005C	R/W
PPA_CK_DEFAULT_REG	Configures the default color-key value of BLEND	0x0060	R/W
PPA_SRM_SCAL_ROTATE_REG	Configures the SRM mode	0x0064	varies
PPA_SRM_MEM_PD_REG	Configures SRM memory	0x0068	R/W
PPA_REG_CONF_REG	Enables register clock	0x006C	R/W
PPA_RGB2GRAY_REG	Configures the coefficient in RGB to GRAY conversion	0x0098	R/W
Interrupt Registers			
PPA_INT_RAW_REG	Raw status interrupt	0x0010	R/WTC/SS
PPA_INT_ST_REG	Masked interrupt	0x0014	RO
PPA_INT_ENA_REG	Interrupt enable bits	0x0018	R/W

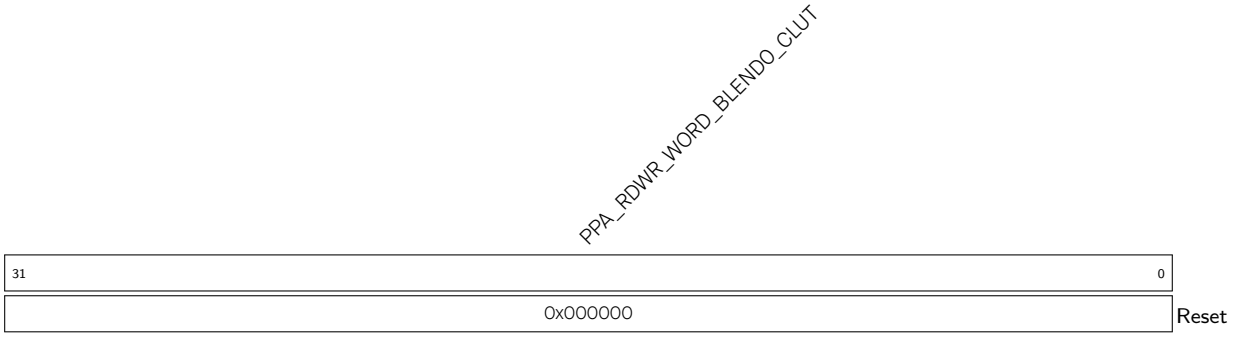
Name	Description	Address	Access
PPA_INT_CLR_REG	Interrupt clear bits	0x001C	WT
Status Registers			
PPA_CLUT_CNT_REG	BLEND CLUT write counter register	0x0070	RO
PPA_BLEND_ST_REG	BLEND status register	0x0074	RO
PPA_SRM_PARAM_ERR_ST_REG	SRM configuration error status register	0x0078	RO
PPA_SRM_STATUS_REG	SRM FSM register	0x007C	RO
Debug Register			
PPA_DEBUG_CTRLO_REG	Debug register	0x0090	R/W
PPA_DEBUG_CTRL1_REG	Debug register	0x0094	R/W
Version Register			
PPA_DATE_REG	PPA Version register	0x0100	R/W

39.9 Registers

The addresses in this section are relative to the Pixel-Processing Accelerator base address provided in Table 9.3-2 in Chapter 9 *System and Memory*.

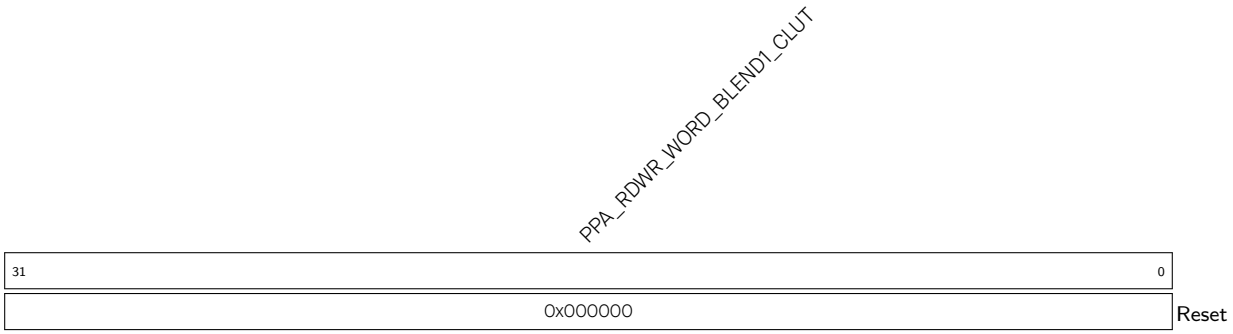
For how to program reserved fields, please refer to Section Programming Reserved Register Field.

Register 39.1. PPA_BLEND0_CLUT_DATA_REG (0x0000)



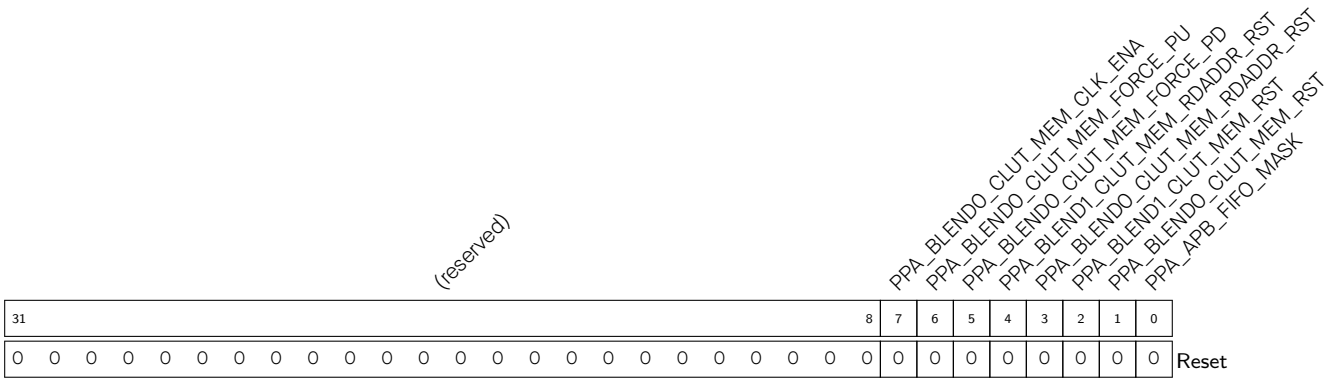
PPA_RDWR_WORD_BLEND0_CLUT Configures BLEND background layer CLUT access in FIFO mode. (R/W)

Register 39.2. PPA_BLEND1_CLUT_DATA_REG (0x0004)



PPA_RDWR_WORD_BLEND1_CLUT Configures BLEND foreground layer CLUT access in FIFO mode. (R/W)

Register 39.3. PPA_CLUT_CONF_REG (0x000C)



PPA_APB_FIFO_MASK Configures the access mode of BLEND CLUT

1'b0: Access CLUT in FIFO mode. For details, please refer to 39.5.1.2.

1'b1: Access CLUT in MEM mode. For details, please refer to 39.5.1.2.

(R/W)

PPA_BLEND0_CLUT_MEM_RST Configures whether to reset BLEND0 CLUT.

0: Release reset

1: Reset

(R/W)

PPA_BLEND1_CLUT_MEM_RST Configures whether to reset BLEND1 CLUT.

0: Release reset

1: Reset

(R/W)

PPA_BLEND0_CLUT_MEM_RDADDR_RST Configures whether to reset BLEND0 CLUT read address in FIFO mode.

0: Release reset

1: Reset

(R/W)

PPA_BLEND1_CLUT_MEM_RDADDR_RST Configures whether to reset BLEND1 CLUT read address in FIFO mode.

0: Release reset

1: Reset

(R/W)

PPA_BLEND0_CLUT_MEM_FORCE_PD Configures whether to enable the force power down of BLEND CLUT.

0: Disable

1: Enable

(R/W)

Continued on the next page...

Register 39.3. PPA_CLUT_CONF_REG (0x000C)

Continued from the previous page...

PPA_BLEND0_CLUT_MEM_FORCE_PU Configures whether to enable the force power up of BLEND CLUT.

0: Disable

1: Enable

(R/W)

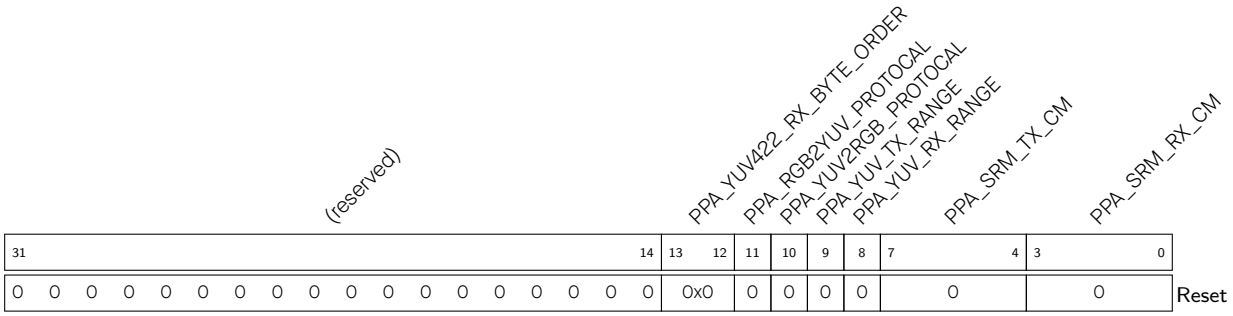
PPA_BLEND0_CLUT_MEM_CLK_ENA Configures whether to enable the force clock on of BLEND CLUT.

0: Disable

1: Enable

(R/W)

Register 39.4. PPA_SRM_COLOR_MODE_REG (0x0020)



PPA_SRM_RX_CM Configures the input image color format for SRM.

0: ARGB8888

1: RGB888

2: RGB565

8: YUV420

Others: Reserved

(R/W)

PPA_SRM_TX_CM Configures the output image color format for SRM.

0: ARGB8888

1: RGB888

2: RGB565

8: YUV420

Others: Reserved

(R/W)

PPA_YUV_RX_RANGE Configures the YUV range when the SRM input format is YUV.

0: limit range

1: full range

(R/W)

PPA_YUV_TX_RANGE Configures the YUV range when the SRM output format is YUV.

0: limit range

1: full range

(R/W)

PPA_YUV2RGB_PROTOCAL Configures the protocol used for the SRM input direction YUV to RGB.

0: BT601

1: BT709

(R/W)

PPA_RGB2YUV_PROTOCAL Configures the protocol used for the SRM output direction RGB to YUV.

0: BT601

1: BT709

(R/W)

Continued on the next page...

Register 39.4. PPA_SRM_COLOR_MODE_REG (0x0020)

Continued from the previous page...

PPA_YUV422_RX_BYTE_ORDER Configures the byte order when the SRM input format is YUV422.

0: YVYU

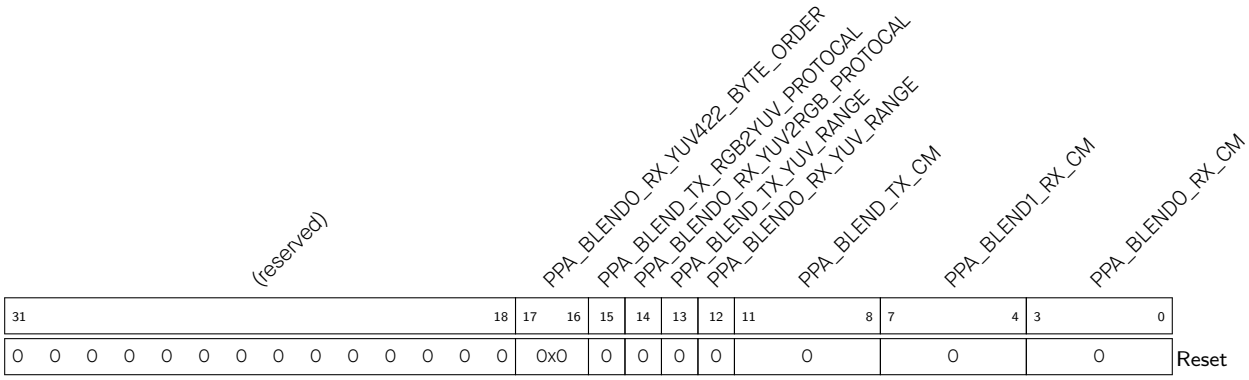
1: YUYV

2: VYUY

3: UYVY

(R/W)

Register 39.5. PPA_BLEND_COLOR_MODE_REG (0x0024)



PPA_BLEND0_RX_CM Configures the input image color format for BLEND background layer.

0: ARGB8888
1: RGB888
2: RGB565
4: L8
5: L4
8: YUV420
9: YUV422
12: GRAY
Others: Reserved
(R/W)

PPA_BLEND1_RX_CM Configures the input image color format for BLEND foreground layer.

0: ARGB8888
1: RGB888
2: RGB565
4: L8
5: L4
6: A8
7: A4
Others: Reserved
(R/W)

PPA_BLEND_TX_CM Configures the output image color format of BLEND.

0: ARGB8888
1: RGB888
2: RGB565
8: YUV420
9: YUV422
12: GRAY
Others: Reserved
(R/W)

Continued on the next page...

Register 39.5. PPA_BLEND_COLOR_MODE_REG (0x0024)

Continued from the previous page...

PPA_BLEND0_RX_YUV_RANGE Configures the YUV range when the BLEND background input is YUV.
0: Limited range
1: Full range
(R/W)

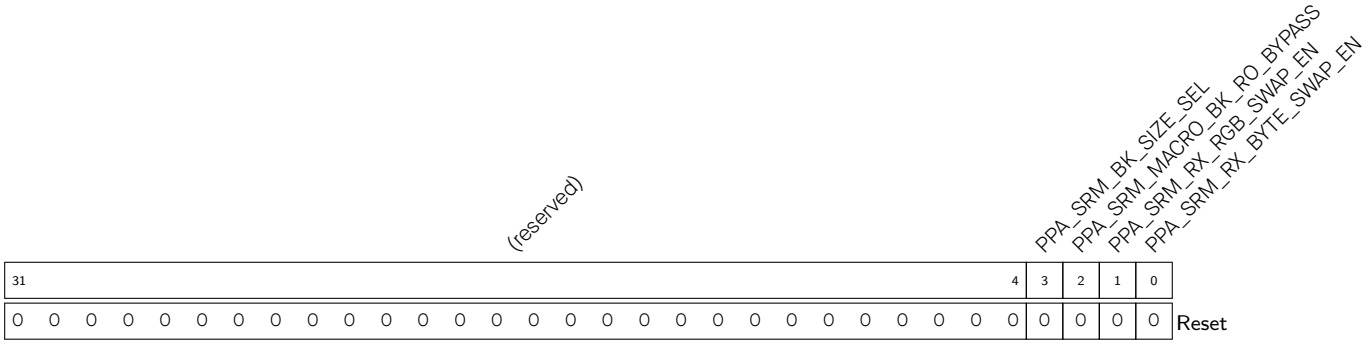
PPA_BLEND_TX_YUV_RANGE Configures the YUV range when the BLEND output is YUV.
0: Limited range
1: Full range
(R/W)

PPA_BLEND0_RX_YUV2RGB_PROTOCAL Configures the YUV to RGB conversion protocol for the BLEND background input.
0: BT601
1: BT709
(R/W)

PPA_BLEND_TX_RGB2YUV_PROTOCAL Configures the YUV to RGB conversion protocol for the BLEND output.
0: BT601
1: BT709
(R/W)

PPA_BLEND0_RX_YUV422_BYTE_ORDER Configures the byte order when the BLEND background input is YUV422.
0: YVYU
1: YUYV
2: VYUY
3: UYVY
(R/W)

Register 39.6. PPA_SRM_BYTE_ORDER_REG (0x0028)



PPA_SRM_RX_BYTE_SWAP_EN Configures whether to swap the SRM valid data input in byte.

0: Not swap

1: Swap

(R/W)

PPA_SRM_RX_RGB_SWAP_EN Configures whether to swap the order of each channel when the SRM input is RGB or ARGB.

0: Not swap

1: Swap RGB to BGR, and ARGB to BGRA

(R/W)

PPA_SRM_MACRO_BK_RO_BYPASS Configures whether to enable the pixel block order function

0: Enable and automatically calculate if the current block needs to be reordered

1: Disable

(R/W)

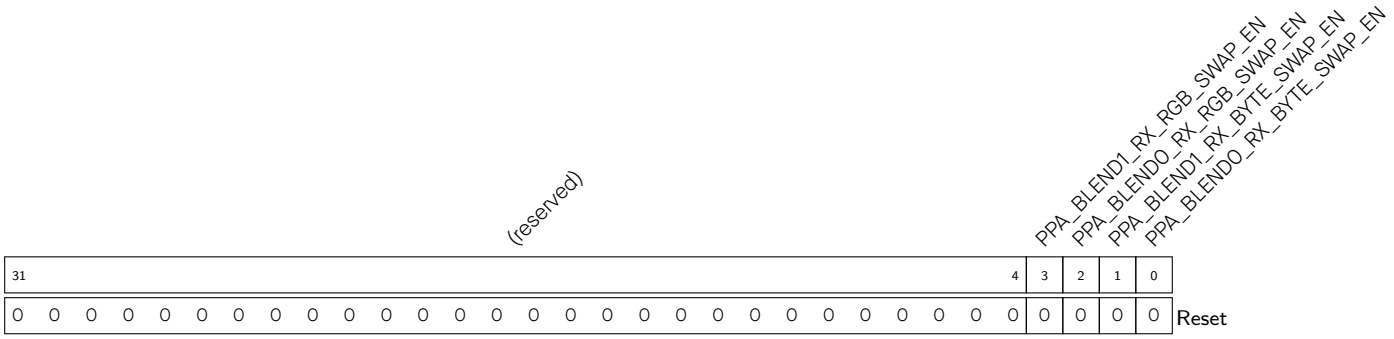
PPA_SRM_BK_SIZE_SEL Configures the size of SRM pixel blocks.

0: 32x32

1: 16x16

(R/W)

Register 39.7. PPA_BLEND_BYTE_ORDER_REG (0x002C)



PPA_BLEND0_RX_BYTE_SWAP_EN Configures whether to swap the BLEND background layer valid data input in byte.

0: Not swap

1: Swap

(R/W)

PPA_BLEND1_RX_BYTE_SWAP_EN Configures whether to swap the BLEND foreground layer valid data input in byte.

0: Not swap

1: Swap

(R/W)

PPA_BLEND0_RX_RGB_SWAP_EN Configures whether to swap the order of each channel when the BLEND background layer input is RGB or ARGB.

0: Not swap

1: Swap RGB to BGR, and ARGB to BGRA

(R/W)

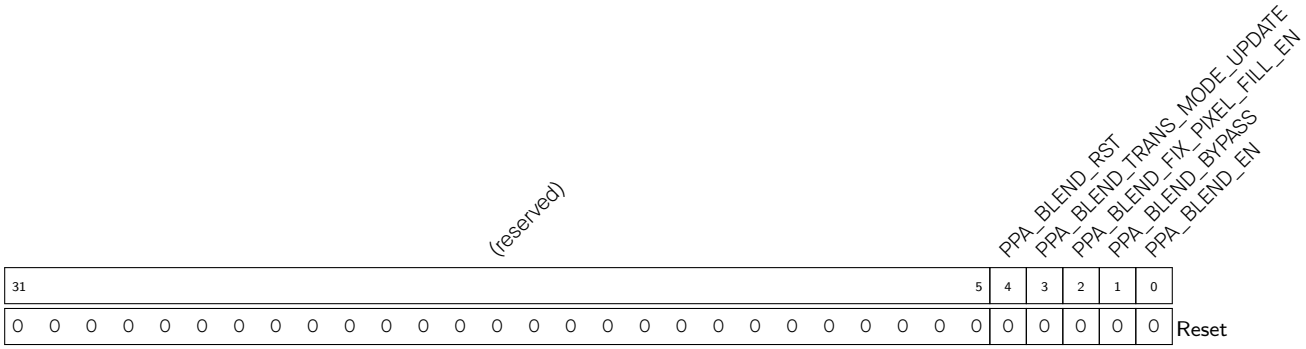
PPA_BLEND1_RX_RGB_SWAP_EN Configures whether to swap the order of each channel when the BLEND foreground layer input is RGB or ARGB.

0: Not swap

1: Swap RGB to BGR, and ARGB to BGRA

(R/W)

Register 39.8. PPA_BLEND_TRANS_MODE_REG (0x0034)



PPA_BLEND_EN Configures whether to enable BLEND.

0: Disable

1: Enable

(R/W)

PPA_BLEND_BYPASS Configures whether to bypass BLEND and directly output the background layer.

0: Bypass

1: Not bypass

(R/W)

PPA_BLEND_FIX_PIXEL_FILL_EN Configures whether to enable filled image output.

0: Disable

1: Enable, where only BLEND TX works and the output pixel and size are configured by PPA_BLEND_TX_SIZE_REG and PPA_BLEND_TX_FIX_PIXEL.

(R/W)

PPA_BLEND_TRANS_MODE_UPDATE Write 1 to update the transfer mode, active PPA_BLEND_FIX_PIXEL_FILL_EN, and start BLEND workflow. (WT)

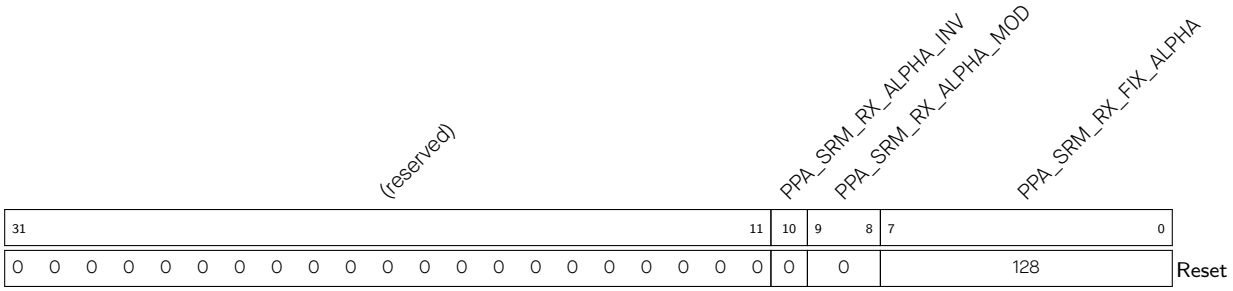
PPA_BLEND_RST Configures whether to reset BLEND.

0: Release reset

1: Reset

(R/W)

Register 39.9. PPA_SRM_FIX_ALPHA_REG (0x0038)



PPA_SRM_RX_FIX_ALPHA Configures the Alpha channal value of SRM. (R/W)

PPA_SRM_RX_ALPHA_MOD Configures the Alpha mode of SRM input data.

1: Use the value of PPA_SRM_RX_FIX_ALPHA

2: Use the value of the higher eight bits of the original Alpha value multiplied by PPA_SRM_RX_FIX_ALPHA

Others: Use the value of the original Alpha value

(R/W)

PPA_SRM_RX_ALPHA_INV Configures whether to invert the original Alpha value of SRM.

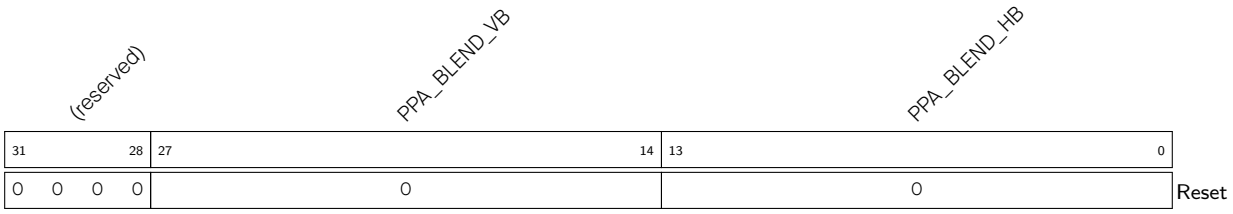
0: Not invert

1: Invert to 255 minus the original Alpha value

For data formats without Alpha channels, the original Alpha value is 255.

(R/W)

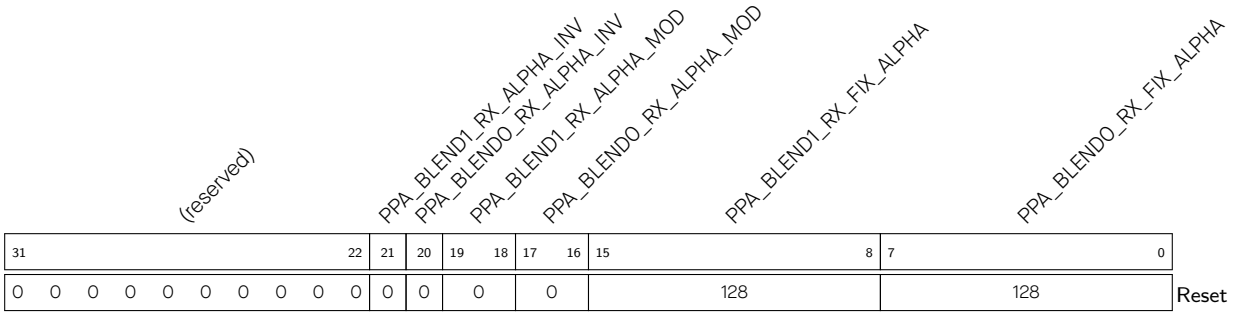
Register 39.10. PPA_BLEND_TX_SIZE_REG (0x003C)



PPA_BLEND_HB Configures the horizontal width of image block that would be filled in the image filling mode. Measurement unit: pixel. (R/W)

PPA_BLEND_VB Configures the vertical width of image block that would be filled in the image filling mode. Measurement unit: pixel. (R/W)

Register 39.11. PPA_BLEND_FIX_ALPHA_REG (0x0040)



PPA_BLEND0_RX_FIX_ALPHA Configures the Alpha channal value of the BLEND background layer. (R/W)

PPA_BLEND1_RX_FIX_ALPHA Configures the Alpha channal value of the BLEND foreground layer. (R/W)

PPA_BLEND0_RX_ALPHA_MOD Configures the Alpha mode of BLEND background layer data.

1: Use the value of PPA_BLEND0_RX_FIX_ALPHA

2: Use the value of the higher eight bits of the original Alpha value multiplied by PPA_BLEND0_RX_FIX_ALPHA

Others: Use the value of the original Alpha value

(R/W)

PPA_BLEND1_RX_ALPHA_MOD Configures the Alpha mode of BLEND foreground layer data.

1: Use the value of PPA_BLEND1_RX_FIX_ALPHA

2: Use the value of the higher eight bits of the original Alpha value multiplied by PPA_BLEND1_RX_FIX_ALPHA

Others: Use the value of the original Alpha value

(R/W)

PPA_BLEND0_RX_ALPHA_INV Configures whether to invert the original Alpha value of the BLEND background layer data.

0: Not invert

1: Invert to 255 minus the original Alpha value

For data formats without Alpha channels, the original Alpha value is 255.

(R/W)

PPA_BLEND1_RX_ALPHA_INV Configures whether to invert the original Alpha value of the BLEND foreground layer data.

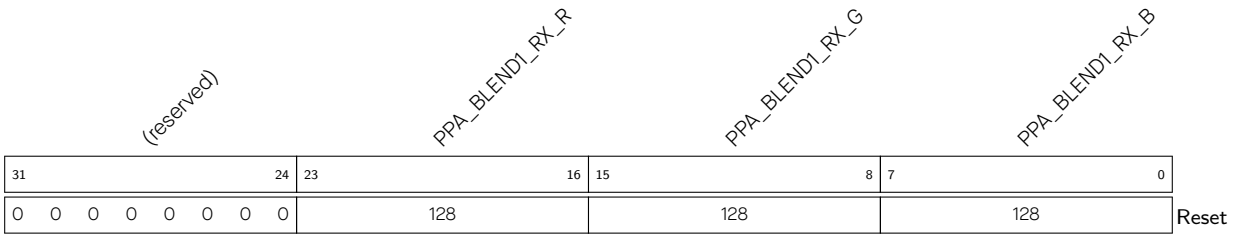
0: Not invert

1: Invert to 255 minus the original Alpha value

For data formats without Alpha channels, the original Alpha value is 255.

(R/W)

Register 39.12. PPA_BLEND_RGB_REG (0x0048)

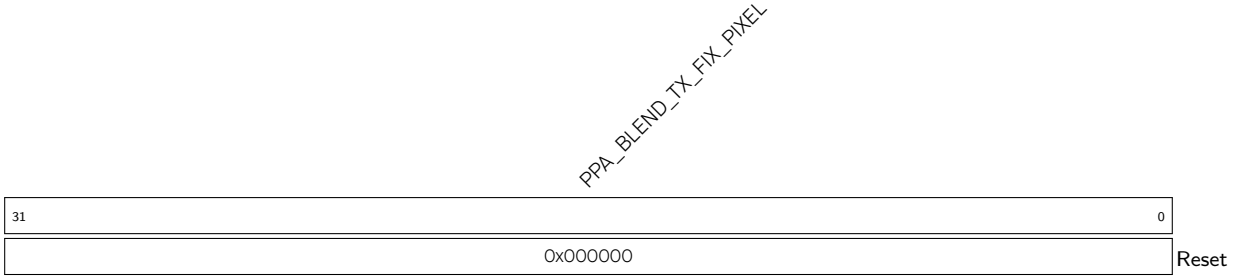


PPA_BLEND1_RX_B Configures the B channel value when the BLEND foreground layer is in A4/A8. (R/W)

PPA_BLEND1_RX_G Configures the G channel value when the BLEND foreground layer is in A4/A8. (R/W)

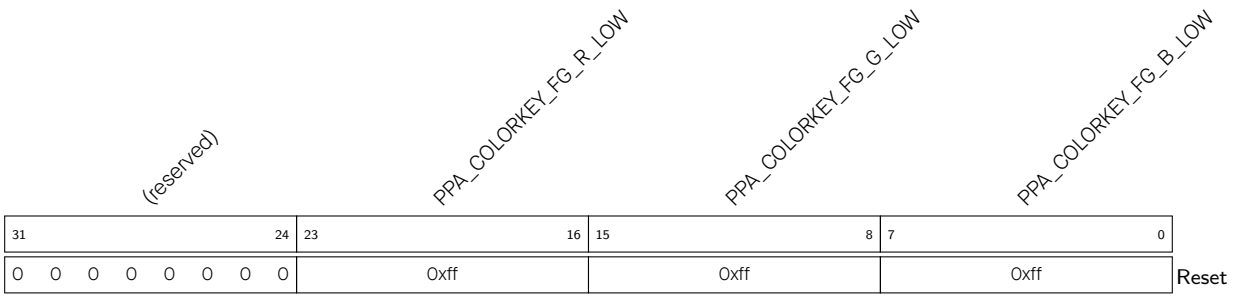
PPA_BLEND1_RX_R Configures the R channel value when the BLEND foreground layer is in A4/A8. (R/W)

Register 39.13. PPA_BLEND_FIX_PIXEL_REG (0x004C)



PPA_BLEND_TX_FIX_PIXEL Configures the pixel in the BLEND image filling mode.
[31:24]: A, [23:16]:R, [15:8]: G, [7:0]: B.
(R/W)

Register 39.14. PPA_CK_FG_LOW_REG (0x0050)

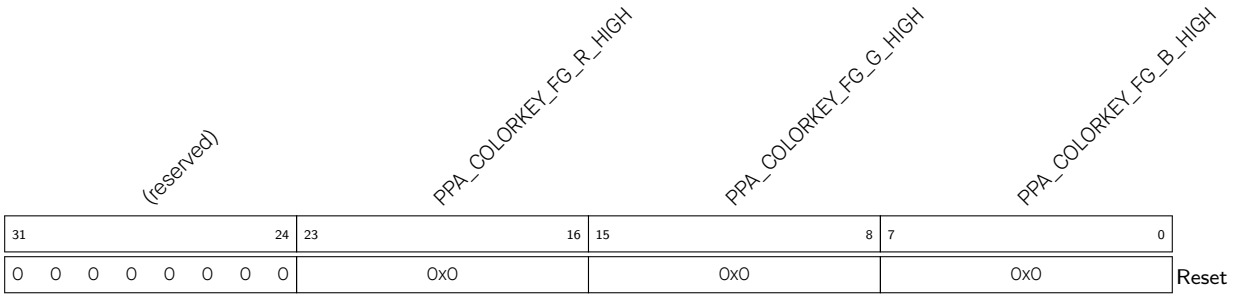


PPA_COLORKEY_FG_B_LOW Configures the color-key lower threshold of the B channel value of the BLEND foreground layer. (R/W)

PPA_COLORKEY_FG_G_LOW Configures the color-key lower threshold of the G channel value of the BLEND foreground layer. (R/W)

PPA_COLORKEY_FG_R_LOW Configures the color-key lower threshold of the R channel value of the BLEND foreground layer. (R/W)

Register 39.15. PPA_CK_FG_HIGH_REG (0x0054)

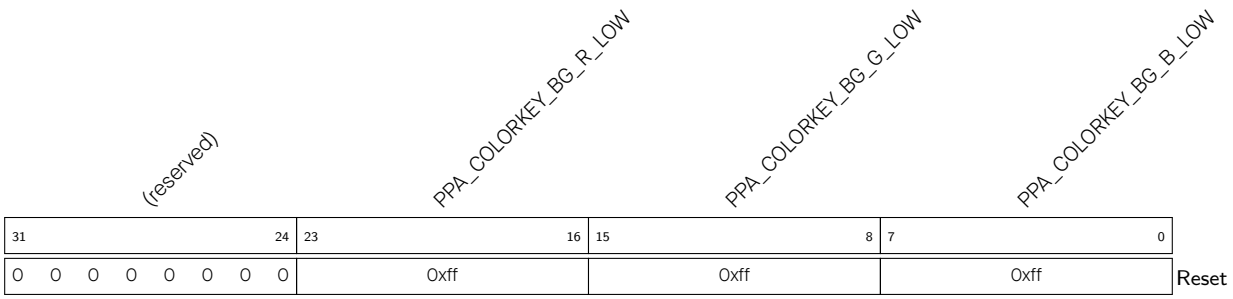


PPA_COLORKEY_FG_B_HIGH Configures the color-key higher threshold of the B channel value of the BLEND foreground layer. (R/W)

PPA_COLORKEY_FG_G_HIGH Configures the color-key higher threshold of the G channel value of the BLEND foreground layer. (R/W)

PPA_COLORKEY_FG_R_HIGH Configures the color-key higher threshold of the R channel value of the BLEND foreground layer. (R/W)

Register 39.16. PPA_CK_BG_LOW_REG (0x0058)

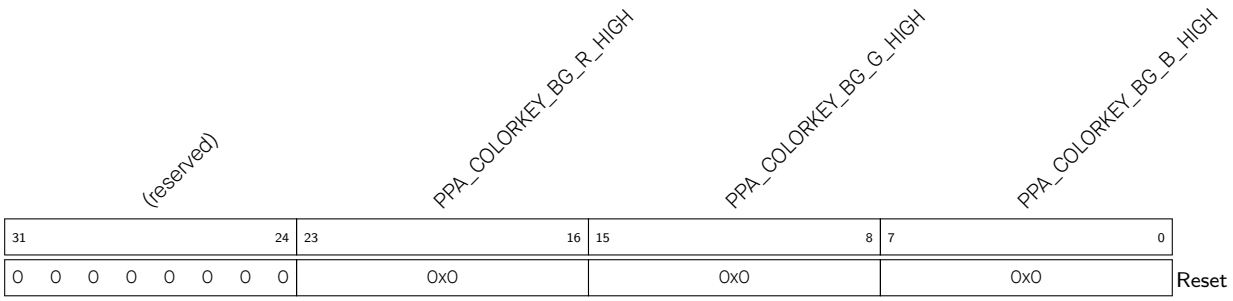


PPA_COLORKEY_BG_B_LOW Configures the color-key lower threshold of the B channel value of the BLEND background layer. (R/W)

PPA_COLORKEY_BG_G_LOW Configures the color-key lower threshold of the G channel value of the BLEND background layer. (R/W)

PPA_COLORKEY_BG_R_LOW Configures the color-key lower threshold of the R channel value of the BLEND background layer. (R/W)

Register 39.17. PPA_CK_BG_HIGH_REG (0x005C)



PPA_COLORKEY_BG_B_HIGH Configures the color-key higher threshold of the B channel value of the BLEND background layer. (R/W)

PPA_COLORKEY_BG_G_HIGH Configures the color-key higher threshold of the G channel value of the BLEND background layer. (R/W)

PPA_COLORKEY_BG_R_HIGH Configures the color-key higher threshold of the R channel value of the BLEND background layer. (R/W)

Register 39.18. PPA_CK_DEFAULT_REG (0x0060)



PPA_COLORKEY_DEFAULT_B Configures the default B channel value of BLEND color-key. (R/W)

PPA_COLORKEY_DEFAULT_G Configures the default G channel value of BLEND color-key. (R/W)

PPA_COLORKEY_DEFAULT_R Configures the default R channel value of BLEND color-key. (R/W)

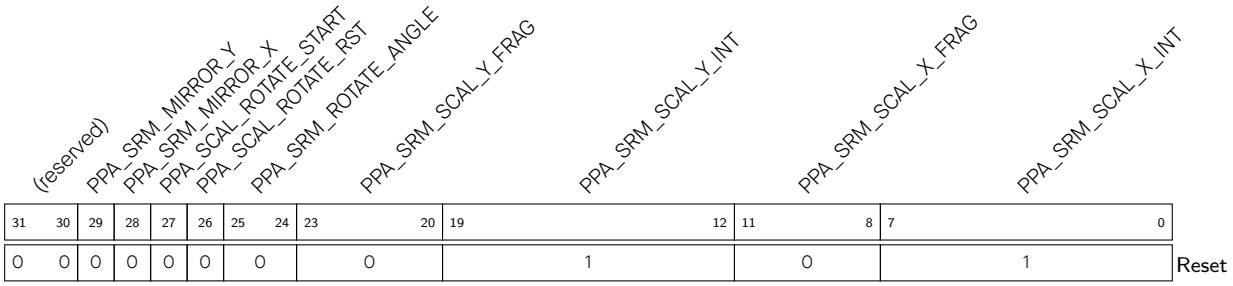
PPA_COLORKEY_FG_BG_REVERSE Configures the workflow when when a pixel is within the background color-key range but not within the foreground color-key range.

0: Output background pixel

1: Output foreground pixel

(R/W)

Register 39.19. PPA_SRM_SCAL_ROTATE_REG (0x0064)



PPA_SRM_SCAL_X_INT Configures the integer part of the horizontal scaling factor for SRM. (R/W)

PPA_SRM_SCAL_X_FRAG Configures the fragment part of the horizontal scaling factor for SRM. (R/W)

PPA_SRM_SCAL_Y_INT Configures the integer part of the vertical scaling factor for SRM. (R/W)

PPA_SRM_SCAL_Y_FRAG Configures the fragment part of the vertical scaling factor for SRM. (R/W)

PPA_SRM_ROTATE_ANGLE Configures the counterclockwise rotation angle for the SRM.

0: 0 degree
1: 90 degree
2: 180 degree
3: 270 degree
(R/W)

PPA_SCAL_ROTATE_RST Configures whether to reset SRM.

0: Release reset
1: Reset
(R/W)

PPA_SCAL_ROTATE_START Write 1 to enable SRM. (WT)

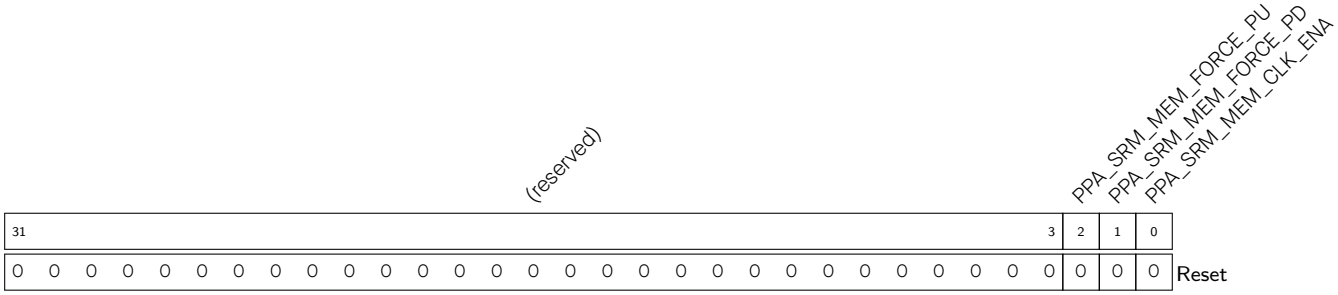
PPA_SRM_MIRROR_X Configures whether to enable the horizontal mirroring for SRM.

0: Disable
1: Enable
(R/W)

PPA_SRM_MIRROR_Y Configures whether to enable the vertical mirroring for SRM.

0: Disable
1: Enable
(R/W)

Register 39.20. PPA_SRM_MEM_PD_REG (0x0068)



PPA_SRM_MEM_CLK_ENA Configures whether to enable the force enable of the SRM MEM clock.

0: Disable

1: Enable

(R/W)

PPA_SRM_MEM_FORCE_PD Configures whether to enable the force power down of SRM MEM.

0: Disable

1: Enable

(R/W)

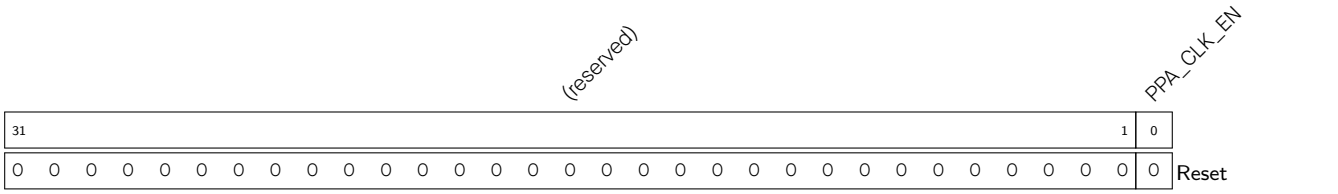
PPA_SRM_MEM_FORCE_PU Configures whether to enable the force power up of SRM MEM.

0: Disable

1: Enable

(R/W)

Register 39.21. PPA_REG_CONF_REG (0x006C)



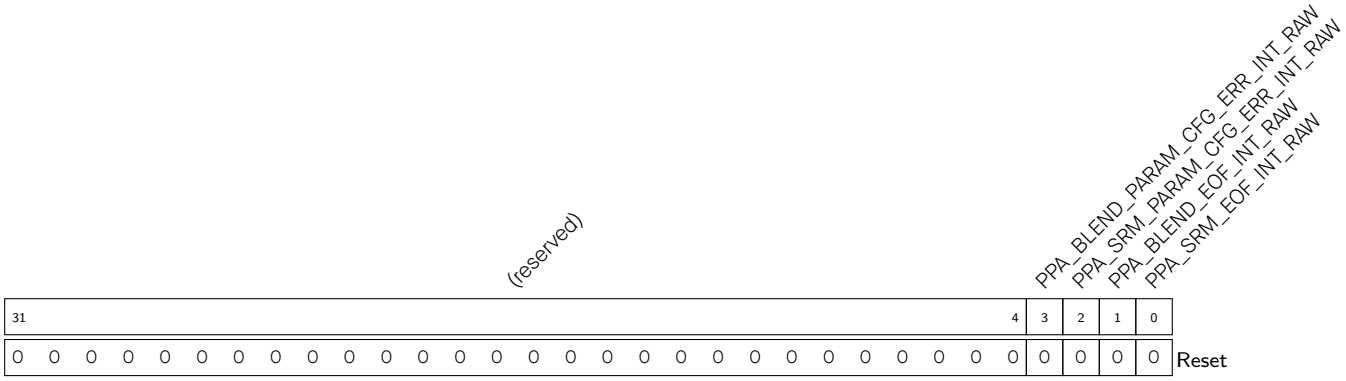
PPA_CLK_EN Configures whether to keep the PPA register clock always on.

0: Clock only turns on when there's a register access

1: Clock always on

(R/W)

Register 39.22. PPA_INT_RAW_REG (0x0010)



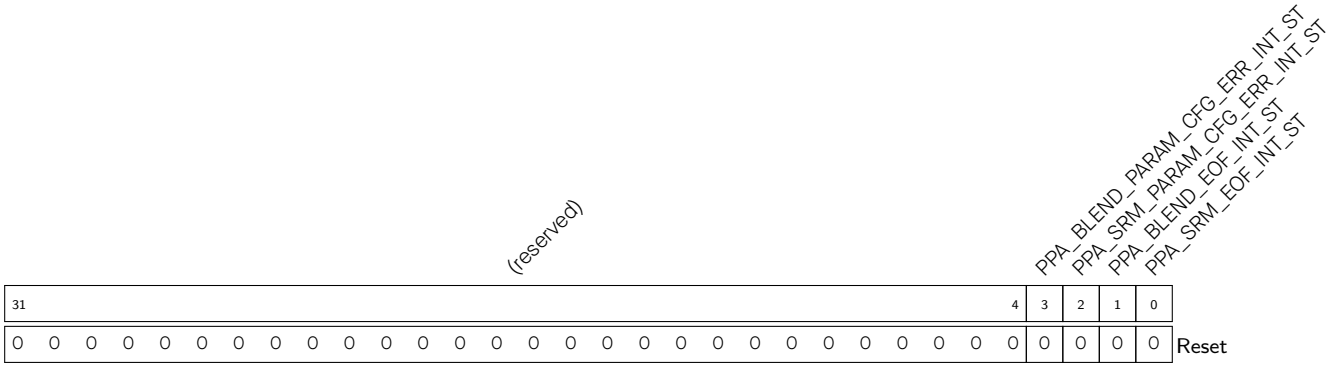
PPA_SRM_EOF_INT_RAW The raw interrupt status of PPA_SRM_EOF_INT.
(R/WTC/SS)

PPA_BLEND_EOF_INT_RAW The raw interrupt status of PPA_BLEND_EOF_INT.
(R/WTC/SS)

PPA_SRM_PARAM_CFG_ERR_INT_RAW The raw interrupt status of PPA_SRM_PARAM_CFG_ERR_INT. Check PPA_SRM_PARAM_ERR_ST_REG to get the specific error.
(R/WTC/SS)

PPA_BLEND_PARAM_CFG_ERR_INT_RAW The raw interrupt status of PPA_BLEND_PARAM_CFG_ERR_INT. Check PPA_BLEND_ST_REG to get the specific error.
(R/WTC/SS)

Register 39.23. PPA_INT_ST_REG (0x0014)



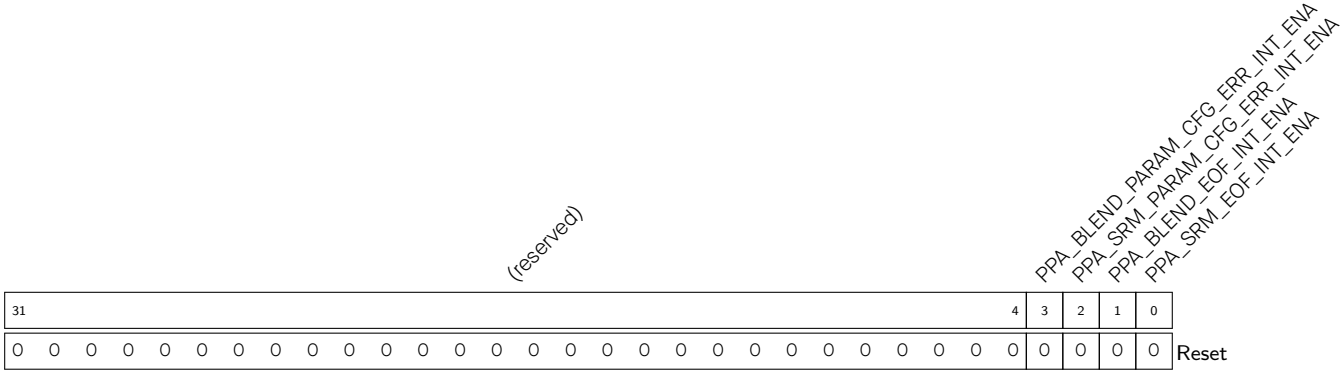
PPA_SRM_EOF_INT_ST The masked interrupt status of PPA_SRM_EOF_INT. (RO)

PPA_BLEND_EOF_INT_ST The masked interrupt status of PPA_BLEND_EOF_INT. (RO)

PPA_SRM_PARAM_CFG_ERR_INT_ST The masked interrupt status of PPA_SRM_PARAM_CFG_ERR_INT. (RO)

PPA_BLEND_PARAM_CFG_ERR_INT_ST The masked interrupt status of PPA_BLEND_PARAM_CFG_ERR_INT. (RO)

Register 39.24. PPA_INT_ENA_REG (0x0018)



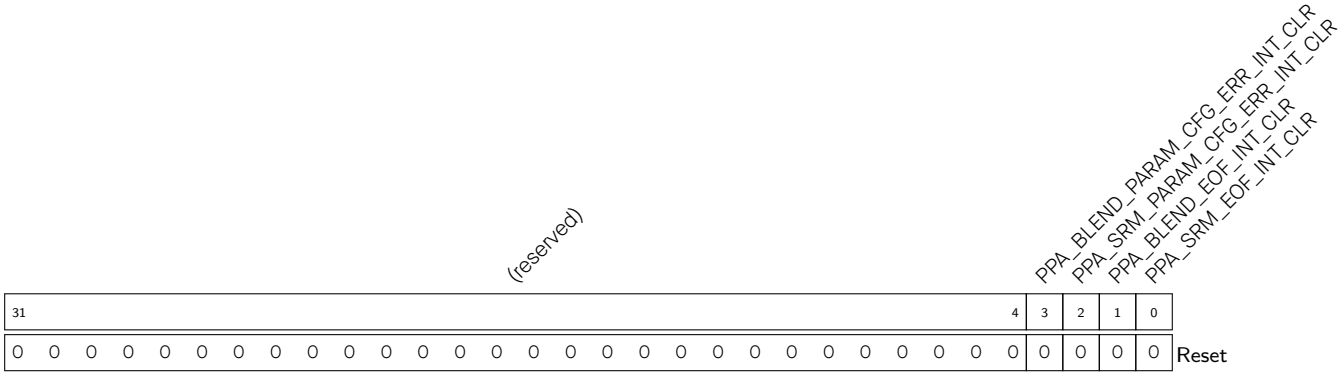
PPA_SRM_EOF_INT_ENA Write 1 to enable PPA_SRM_EOF_INT. (R/W)

PPA_BLEND_EOF_INT_ENA Write 1 to enable PPA_BLEND_EOF_INT. (R/W)

PPA_SRM_PARAM_CFG_ERR_INT_ENA Write 1 to enable PPA_SRM_PARAM_CFG_ERR_INT. (R/W)

PPA_BLEND_PARAM_CFG_ERR_INT_ENA Write 1 to enable PPA_BLEND_PARAM_CFG_ERR_INT. (R/W)

Register 39.25. PPA_INT_CLR_REG (0x001C)



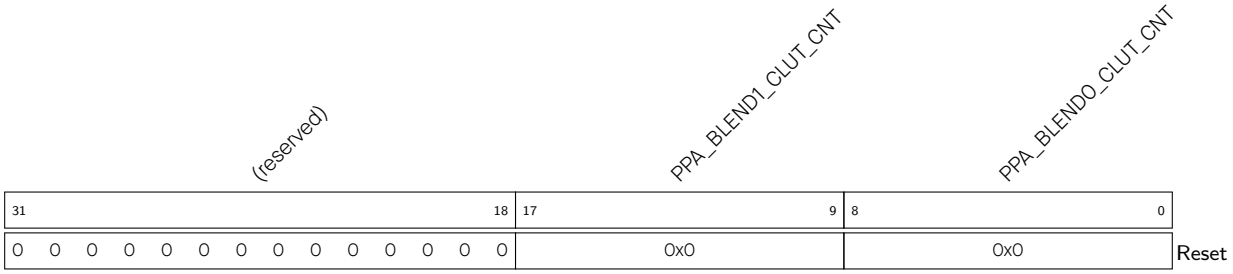
PPA_SRM_EOF_INT_CLR Write 1 to clear PPA_SRM_EOF_INT. (WT)

PPA_BLEND_EOF_INT_CLR Write 1 to clear PPA_BLEND_EOF_INT. (WT)

PPA_SRM_PARAM_CFG_ERR_INT_CLR Write 1 to clear PPA_SRM_PARAM_CFG_ERR_INT. (WT)

PPA_BLEND_PARAM_CFG_ERR_INT_CLR □ 1 □□ PPA_BLEND_PARAM_CFG_ERR_INT□ (WT)

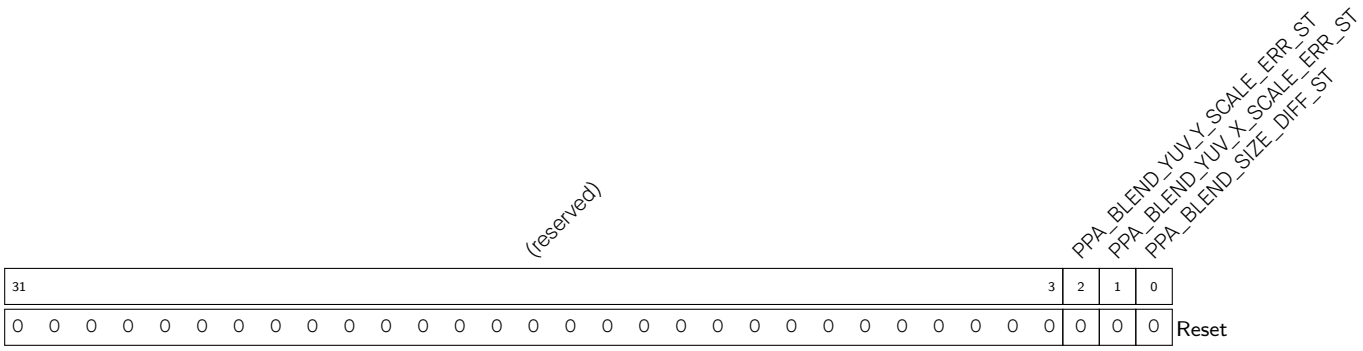
Register 39.26. PPA_CLUT_CNT_REG (0x0070)



PPA_BLEND0_CLUT_CNT Represents the current write address of the BLEND background layer CLUT in FIFO mode. (RO)

PPA_BLEND1_CLUT_CNT Represents the current write address of the BLEND foreground layer CLUT in FIFO mode. (RO)

Register 39.27. PPA_BLEND_ST_REG (0x0074)



PPA_BLEND_SIZE_DIFF_ST Represents whether the size of the BLEND background and foreground images are consistent.

0: Consistent

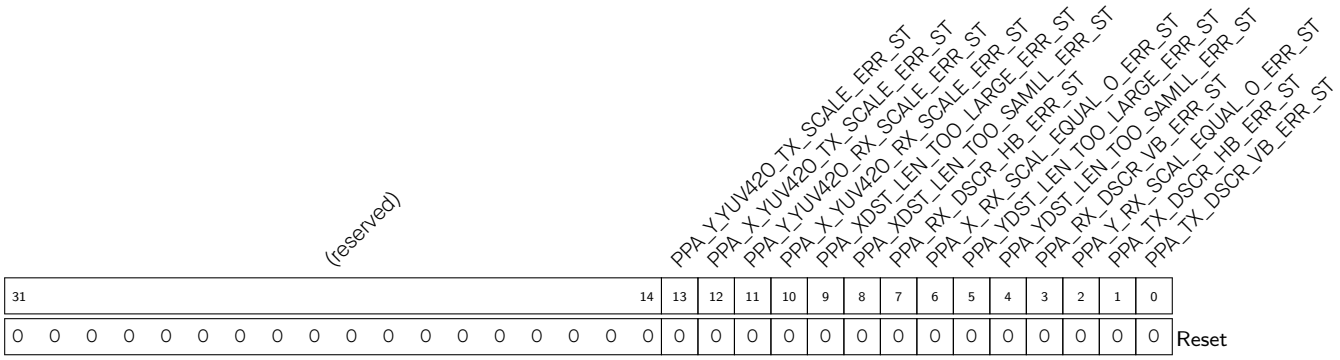
1: Different

(RO)

PPA_BLEND_YUV_X_SCALE_ERR_ST Represents the image width (X) is odd for YUV422 or YUV420 formats. (RO)

PPA_BLEND_YUV_Y_SCALE_ERR_ST Represents the image height (Y) is odd for the YUV420 format. (RO)

Register 39.28. PPA_SRM_PARAM_ERR_ST_REG (0x0078)



PPA_TX_DSCR_VB_ERR_ST Represents whether the vertical size of the SRM output image block plus the vertical offset Y configured in the output list exceeds the vertical size of the entire image configured in the output list, VA.

0: Not exceeded

1: Exceeded

(RO)

PPA_TX_DSCR_HB_ERR_ST Represents whether the horizontal size of the SRM output image block plus the vertical offset X configured in the output list exceeds the horizontal size of the entire image configured in the output list, HA.

0: Not exceeded

1: Exceeded

(RO)

PPA_Y_RX_SCAL_EQUAL_0_ERR_ST Represents whether the vertical direction scaling factor of the SRM input image block is 0.

0: Not 0

1: 0

(RO)

PPA_RX_DSCR_VB_ERR_ST Represents whether the sum of the vertical direction image block size VB configured in the SRM input list and the vertical offset Y exceeds the entire image's vertical size VA.

0: Not exceeded

1: Exceeded

(RO)

PPA_YDST_LEN_TOO_SAMLL_ERR_ST Represents whether the vertical size of the image block after scaling in the SRM is 0.

0: Not 0

1: 0

(RO)

Continued on the next page...

Register 39.28. PPA_SRM_PARAM_ERR_ST_REG (0x0078)

Continued from the previous page...

PPA_YDST_LEN_TOO_LARGE_ERR_ST Represents whether the vertical size of the image block after scaling in the SRM is larger than 8191.
0: Within 8191
1: Larger than 8191
(RO)

PPA_X_RX_SCAL_EQUAL_0_ERR_ST Represents whether the horizontal direction scaling factor of the SRM input image block is 0.
0: Not 0
1: 0
(RO)

PPA_RX_DSCR_HB_ERR_ST Represents whether the sum of the horizontal direction image block size HB configured in the SRM input list and the horizontal offset X exceeds the entire image's horizontal size HA.
0: Not exceeded
1: Exceeded
(RO)

PPA_XDST_LEN_TOO_SAMLL_ERR_ST Represents whether the horizontal size of the image block after scaling in the SRM is 0.
0: Not 0
1: 0
(RO)

PPA_XDST_LEN_TOO_LARGE_ERR_ST Represents whether the horizontal size of the image block after scaling in the SRM is larger than 8191.
0: Within 8191
1: Larger than 8191
(RO)

PPA_X_YUV420_RX_SCALE_ERR_ST Represents whether the parameters HA/HB/X of the SRM input list are odd when the input format is YUV420.
0: Even
1: Odd
(RO)

PPA_Y_YUV420_RX_SCALE_ERR_ST Represents whether the parameters VA/VB/Y of the SRM input list are odd when the input format is YUV420.
0: Even
1: Odd
(RO)

Continued on the next page...

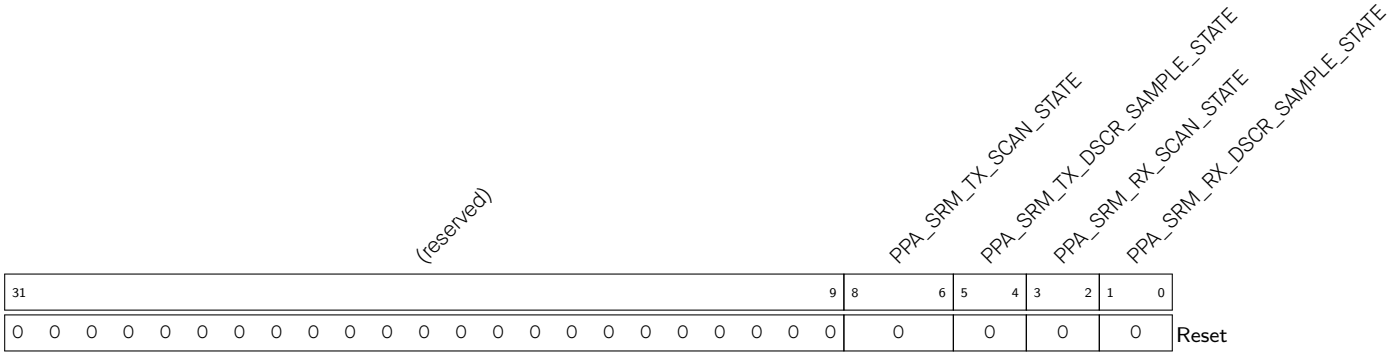
Register 39.28. PPA_SRM_PARAM_ERR_ST_REG (0x0078)

Continued from the previous page...

PPA_X_YUV420_TX_SCALE_ERR_ST Represents whether the parameters HA/HB/X of the SRM output list are odd when the output format is YUV420.
0: Even
1: Odd
(RO)

PPA_Y_YUV420_TX_SCALE_ERR_ST Represents whether the parameters VA/VB/Y of the SRM output list are odd when the output format is YUV420.
0: Even
1: Odd
(RO)

Register 39.29. PPA_SRM_STATUS_REG (0x007C)



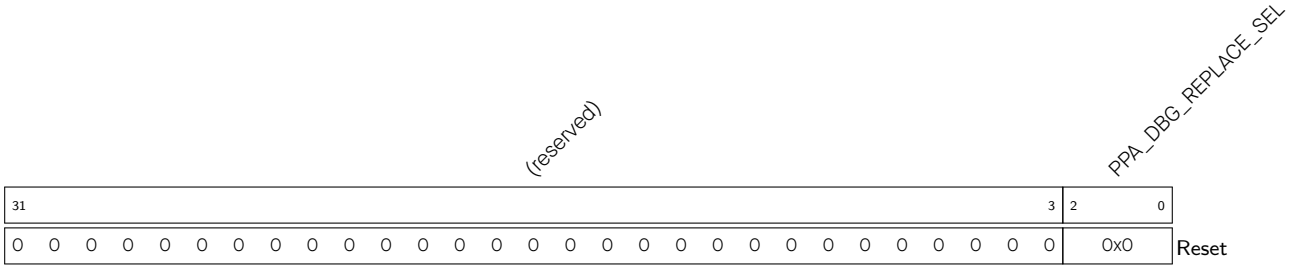
PPA_SRM_RX_DSCR_SAMPLE_STATE Represents the SRM RX descriptor state. (RO)

PPA_SRM_RX_SCAN_STATE Represents the SRM RX pixel scan state. (RO)

PPA_SRM_TX_DSCR_SAMPLE_STATE Represents the SRM TX descriptor state. (RO)

PPA_SRM_TX_SCAN_STATE Represents the SRM TX pixel scan state. (RO)

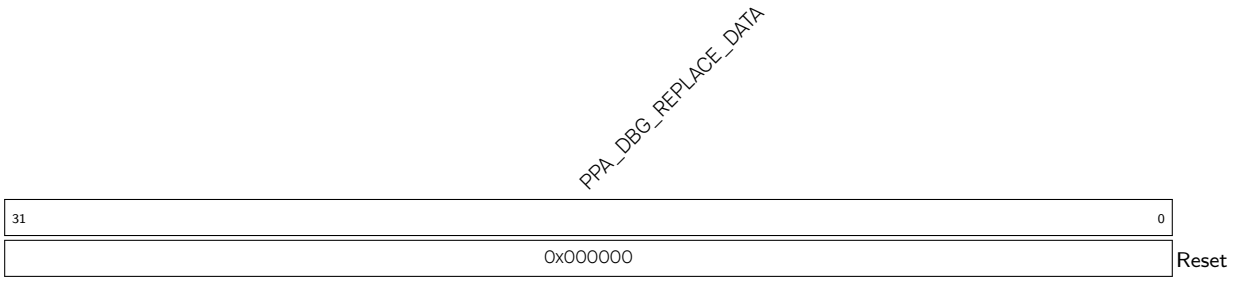
Register 39.30. PPA_DEBUG_CTRL0_REG (0x0090)



PPA_DBG_REPLACE_SEL Configures where the replacement pixel is inserted.

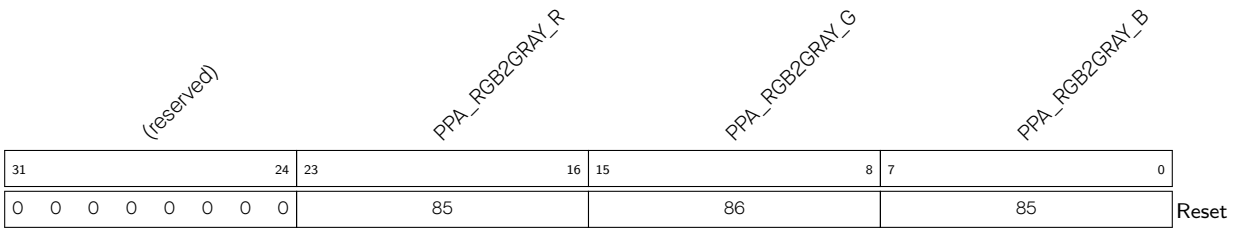
0: No replacement
1: SRM RX input
2: SRM interpolation output
3: SRM TX output
4: BLEND foreground input
5: BLEND background input
6: BLEND output
(R/W)

Register 39.31. PPA_DEBUG_CTRL1_REG (0x0094)



PPA_DBG_REPLACE_DATA Configures the replacement pixel value. (R/W)

Register 39.32. PPA_RGB2GRAY_REG (0x0098)

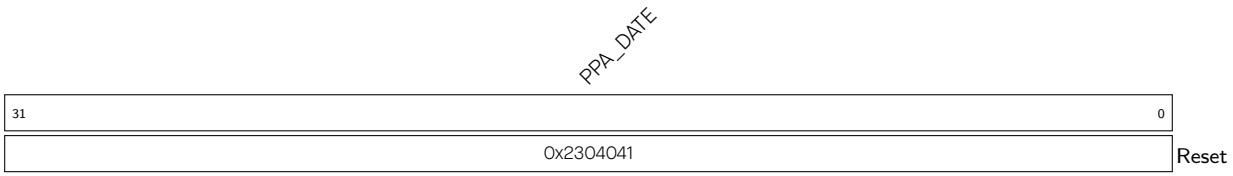


PPA_RGB2GRAY_B Configures the coefficient for B in RGB to GRAY conversion. (R/W)

PPA_RGB2GRAY_G Configures the coefficient for G in RGB to GRAY conversion. (R/W)

PPA_RGB2GRAY_R Configures the coefficient for R in RGB to GRAY conversion. (R/W)

Register 39.33. PPA_DATE_REG (0x0100)



PPA_DATE Version control register. (R/W)

Chapter 40

LCD and Camera Controller (LCD_CAM)

40.1 Overview

The LCD and Camera controller (LCD_CAM) on the ESP32-P4, consisting of an independent LCD control module and a camera control module, is a versatile component designed to facilitate interfacing with both LCDs and cameras.

The LCD module provides an LCD interface that can connect to RGB, Motorola 6800 (MOTO6800), and Intel 8080 (I8080) compatible LCD devices. The LCD module is responsible for sending video data to the display.

The Camera module provides one parallel camera interface that can connect to external digital video port (DVP) cameras in 8/16-bit modes. It manages communication with the camera to capture images or video frames.

The LCD_CAM controller offers flexibility for projects that involve both displaying information on screens and working with cameras for image or video processing.

40.2 Features

LCD_CAM has the following features:

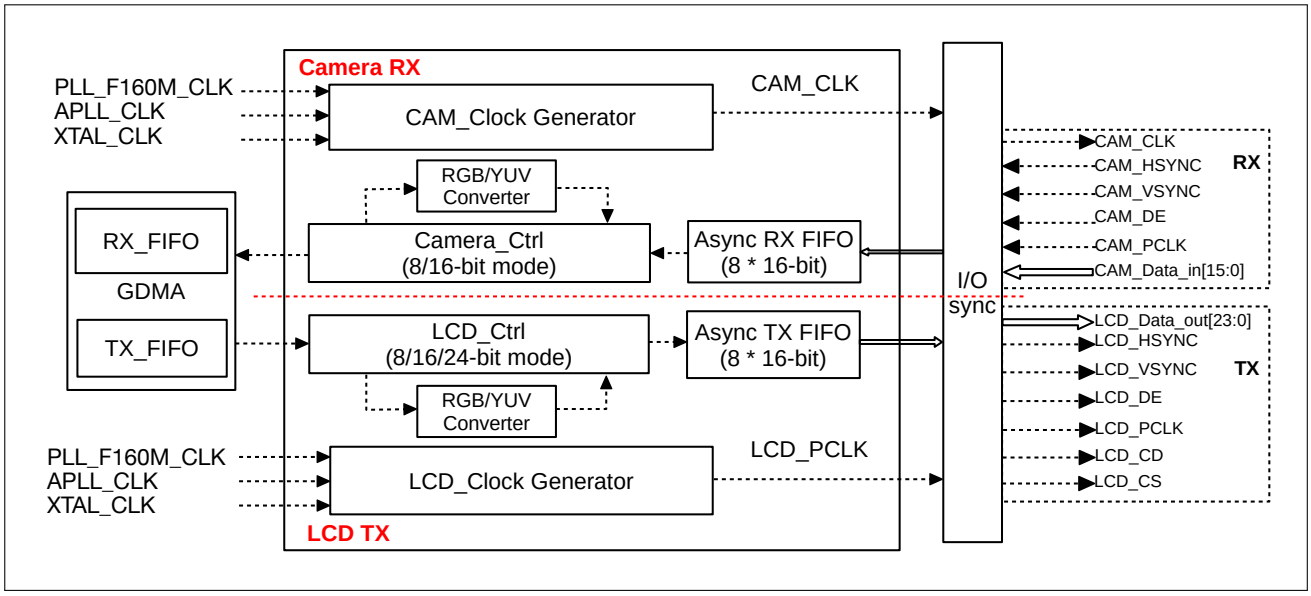
- Supports the following operation modes:
  - LCD master TX mode
  - Camera slave RX mode
  - Camera master RX mode
- Supports simultaneous connection to an external LCD and a camera
- When interfacing with an external LCD, the following is supported:
  - 8/16/24-bit parallel output modes
  - RGB, MOTO6800, and I8080 LCD formats
  - LCD data retrieved from internal memory or external memory via GDMA
- When interfacing with an external camera (i.e., DVP image sensor), the following is supported:
  - 8/16-bit parallel input modes
  - Camera data stored in internal or external memory via GDMA
- Supports interrupts

40.3 Functional Description

40.3.1 Block Diagram

Figure 40.3-1 shows the structure of LCD_CAM which includes:

- One TX control unit (LCD_Ctrl)
- One RX control unit (Camera_Ctrl)
- One asynchronous TX FIFO (Async TX FIFO) for communicating with external devices, e.g., LCDs
- One asynchronous RX FIFO (Async RX FIFO) for communicating with external devices, e.g., cameras
- Two clock generators (LCD_Clock Generator and CAM_Clock Generator) for generating clocks for each module
- Two format converters (RGB/YUV Converter) for converting video data into various formats



- →: control signal =>: data flow

Figure 40.3-1. LCD_CAM Block Diagram

40.3.2 Signal Description

Table 40.3-1 shows the input and output signals required for the operation of the LCD and Camera controller.

The definition of the three operation modes is as follows:

- **Camera Slave RX Mode:** In this mode, the Camera module is the receiver of control signals.
- **Camera Master RX Mode:** In this mode, the Camera module sends clock signals to the slave, which will transmit data to the master upon receiving the signals.
- **LCD Master TX Mode:** In this mode, the LCD module initiates communication with the slave and sends control signals to the slave.

Table 40.3-1. LCD_CAM Input and Output Signal Descriptions

Operation Mode	Signal[1]	Direction	Function
Camera Slave RX Mode	CAM_PCLK	Input	Camera pixel clock signal
	CAM_VSYNC	Input	Vertical synchronization signal (VSYNC)[3]
	CAM_HSYNC	Input	Horizontal synchronization signal (HSYNC)[3]
	CAM_DE	Input	Horizontal enable signal (DE)[3]
	CAM_Data_in[*N*:0][2]	Input	Camera parallel input data bus, 8/16-bit modes supported
Camera Master RX Mode	CAM_PCLK	Input	Camera pixel clock input signal
	CAM_CLK	Output	Camera master clock output signal
	CAM_VSYNC	Input	Vertical synchronization signal[3]
	CAM_HSYNC	Input	Horizontal synchronization signal[3]
	CAM_DE	Input	Horizontal enable signal[3]
	CAM_Data_in[*N*:0][2]	Input	Camera parallel input data bus, 8/16-bit modes supported
LCD Master TX Mode	LCD_PCLK	Output	LCD pixel clock signal
	LCD_HSYNC	Output	Horizontal synchronization signal in RGB format
	LCD_VSYNC	Output	Vertical synchronization signal in RGB format
	LCD_DE	Output	Horizontal enable signal in RGB format
	LCD_CD	Output	Command and data (CD) signal in I8080 format
	LCD_CS	Output	Chip select (CS) signal in I8080/MOTO6800 format
	LCD_Data_out[*M*:0][2]	Output	LCD parallel output data bus, 8/16/24-bit modes supported

[1] All the signals must be mapped to the chip's pins via GPIO matrix. For more information, see Chapter 10 *GPIO Matrix and IO MUX*.

[2] For the input signals with 8 or 16-bit width, *N* = 7 or 15 respectively. For the output signals with 8, 16, or 24-bit width, *M* = 7, 15, or 23 respectively.

[3] If LCD_CAM_CAM_VH_DE_MODE_EN is set, i.e., VSYNC + HSYNC mode is selected, then VSYNC, HSYNC, and DE signals control the data. In this case, users need to wire the three signal lines. If LCD_CAM_CAM_VH_DE_MODE_EN is cleared, i.e., DE mode is selected, then VSYNC and DE signals control the data. In this case, wiring HSYNC signal line is not a must. But in this case, the YUV-RGB conversion function of the Camera module is not available.

40.3.3 LCD_CAM Module Clocks

40.3.3.1 LCD Clock

The clocks used in the LCD module are generated from clock sources by the LCD_Clock Generator, see Figure 40.3-2. The clocks include:

- Master clock: LCD_CLK, divided from the clock sources
- Pixel clock: LCD_PCLK, divided from LCD_CLK

HP_SYS_CLKRST_LCD_CLK_EN is used to enable LCD_CLK_SRC (the clock source), and HP_SYS_CLKRST_LCD_CLK_SRC_SEL is used to select LCD_CLK_SRC from one of the following clock sources:

- 0: XTAL_CLK
- 1: PLL_F160M_CLK
- 2: APLL_CLK
- 3: Disable LCD clock source

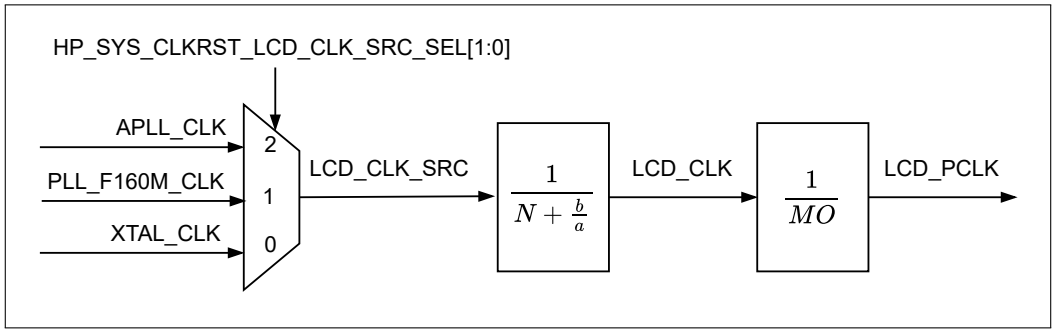


Figure 40.3-2. LCD Clock

The following formula shows the relation between the frequency of LCD_CLK ($f_{\text{LCD_CLK}}$) and LCD_CLK_SRC ($f_{\text{LCD_CLK_SRC}}$):

$$f_{\text{LCD_CLK}} = \frac{f_{\text{LCD_CLK_SRC}}}{N + \frac{b}{a}}$$

The values of *N*, *a*, and *b* are related to HP_SYS_CLKRST_LCD_CLK_DIV_NUM, HP_SYS_CLKRST_LCD_CLK_DIV_NUMERATOR, and HP_SYS_CLKRST_LCD_CLK_DIV_DENOMINATOR. Specifically,

- When HP_SYS_CLKRST_LCD_CLK_DIV_NUM = 0, *N* = 1. The values of *a* and *b* do not take effect. The divider is always 1.
- When HP_SYS_CLKRST_LCD_CLK_DIV_NUM >=1, *N* = HP_SYS_CLKRST_LCD_CLK_DIV_NUM + 1.
  - For integer divider, please clear HP_SYS_CLKRST_LCD_CLK_DIV_NUMERATOR and HP_SYS_CLKRST_LCD_CLK_DIV_DENOMINATOR.
  - For fractional divider, *b* = HP_SYS_CLKRST_LCD_CLK_DIV_NUMERATOR, *a* = HP_SYS_CLKRST_LCD_CLK_DIV_DENOMINATOR. The value of HP_SYS_CLKRST_LCD_CLK_DIV_NUMERATOR should be smaller than the value of HP_SYS_CLKRST_LCD_CLK_DIV_DENOMINATOR.

The following formula shows the relation between the frequencies of LCD_PCLK ($f_{\text{LCD_PCLK}}$) and LCD_CLK ($f_{\text{LCD_CLK}}$):

$$f_{\text{LCD_PCLK}} = \frac{f_{\text{LCD_CLK}}}{\text{MO}}$$

MO is determined by LCD_CAM_LCD_CLK_EQU_SYSCLK and LCD_CAM_LCD_CLKCNT_N, specifically:

- When LCD_CAM_LCD_CLK_EQU_SYSCLK = 1, MO = 1.

- When LCD_CAM_LCD_CLK_EQU_SYSCLK = 0, MO = LCD_CAM_LCD_CLKCNT_N + 1.

Notes:

- LCD_CAM_LCD_CLKCNT_N must not be configured as 0.
- Using fractional divider may introduce clock jitters. In case LCD_CLK and LCD_PCLK can not be generated from PLL_F160M_CLK by integer divider, APLL_CLK can be used as the clock source. For more information, please refer to Chapter 11 *Reset and Clock*.

40.3.3.2 Camera Clock

The clocks used in the Camera module are generated from clock sources by the CAM_Clock Generator, see Figure 40.3-3. The clocks include:

- Master clock: CAM_CLK, master clock output from the Camera module, divided from the clock sources.
- Pixel clock: CAM_PCLK, clock input from camera slave.

HP_SYS_CLKRST_CAM_CLK_EN is used to enable the CAM_CLK_SRC (the clock source), and HP_SYS_CLKRST_CAM_CLK_SRC_SEL is used to select CAM_CLK_SRC from one of the following clock sources:

- 0: XTAL_CLK
- 1: PLL_F160M_CLK
- 2: APLL_CLK
- 3: Disable camera clock source

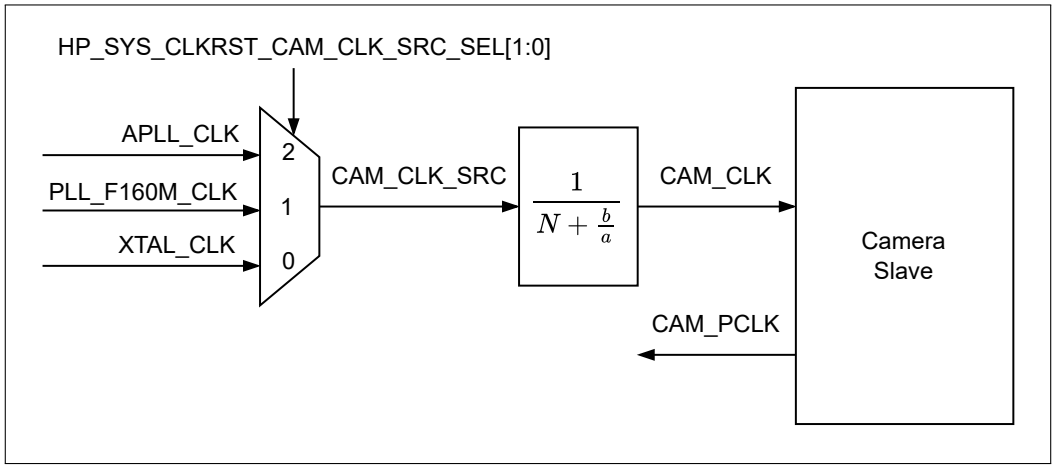


Figure 40.3-3. Camera Clock

The following formula shows the relation between the frequencies of CAM_CLK (f_{CAM_CLK}) and the divider's clock source ($f_{CAM_CLK_SRC}$):

$$f_{CAM_CLK} = \frac{f_{CAM_CLK_SRC}}{N + \frac{b}{a}}$$

N is an integer value between 2 and 256. The value of *N* corresponds to the value of HP_SYS_CLKRST_CAM_CLK_DIV_NUM as follows:

- When HP_SYS_CLKRST_CAM_CLK_DIV_NUM = 0, *N* = 256.

- When HP_SYS_CLKRST_CAM_CLK_DIV_NUM = 1, *N* = 2.
- When HP_SYS_CLKRST_CAM_CLK_DIV_NUM = other value, *N* = HP_SYS_CLKRST_CAM_CLK_DIV_NUM.

b corresponds to the value of HP_SYS_CLKRST_CAM_CLK_DIV_NUMERATOR, and *a* to the value of HP_SYS_CLKRST_CAM_CLK_DIV_DENOMINATOR.

- For integer divider, please clear HP_SYS_CLKRST_CAM_CLK_DIV_NUMERATOR and HP_SYS_CLKRST_CAM_CLK_DIV_DENOMINATOR.
- For fractional divider, the value of HP_SYS_CLKRST_CAM_CLK_DIV_NUMERATOR should be smaller than the value of HP_SYS_CLKRST_CAM_CLK_DIV_DENOMINATOR.

40.3.4 LCD_CAM Reset

LCD_CAM module has the following reset registers that can reset different parts of the module:

- HP_SYS_CLKRST_RST_EN_LCDCAM: Setting this register will reset the whole LCD_CAM module.
- LCD_CAM_LCD_RESET: Setting this register will reset the whole LCD module, including the LCD control unit (LCD_Ctrl), the RGB/YUV Converter, and Async TX FIFO. Configuring this register will not affect the Camera module.
- LCD_CAM_CAM_RESET: Setting this register will reset the whole Camera module, including the Camera control unit (Camera_Ctrl), the RGB/YUV Converter, and Async RX FIFO. Configuring this register will not affect the LCD module.
- LCD_CAM_LCD_AFIFO_RESET: Setting this register will reset the Async TX FIFO without affecting other parts in the module. For example, setting this reset register every time the LCD finishes sending a frame can prevent transmission errors of the previous frame from affecting the transmission of the next frame.
- LCD_CAM_CAM_AFIFO_RESET: Setting this register will reset the Async RX FIFO without affecting other parts in the module. For example, setting this reset register every time the Camera finishes receiving a frame can prevent transmission errors of the previous frame from affecting the receiving of the next frame.

Notes:

- The reset register HP_SYS_CLKRST_RST_EN_LCDCAM is not self-clearing, meaning writing 1 enables the reset, and writing 0 releases the reset.
- All the above reset registers, except for HP_SYS_CLKRST_RST_EN_LCDCAM, are self-clearing, meaning that after writing 1, the hardware will generate a reset pulse and automatically clear the register.
- The module clocks LCD_CLK and CAM_CLK must be configured first before the module and FIFO are reset.

40.3.5 LCD_CAM Data Format Control

40.3.5.1 LCD Data Format Control

When the data is transmitted by the LCD module, it undergoes three stages of processing:

1. Stage One: Data preprocessing
   The LCD module processes the input data from GDMA and sends it to the YUV-RGB converter.

2. Stage Two: Data color space conversion
   The YUV-RGB converter processes the data. If the YUV-RGB converter is disabled, the data remains unchanged at this stage.
3. Stage Three: Data post-processing
   The LCD module gets data from the YUV-RGB converter for post-processing and then transmits the data to GPIO.

Note:
The output data from the previous stage is the input data for the subsequent stage.

The detailed configuration of each stage is described below.

Stage One: Data Preprocessing

At this stage, configure the following registers to adjust the bit width, bit order, and byte order of the data from GDMA.

- LCD_CAM_LCD_BYTE_MODE
  - 0: The data width of the LCD input from GDMA is 8 bits.
  - 1: The data width of the LCD input from GDMA is 16 bits.
  - 2: The data width of the LCD input from GDMA is 24 bits.
- LCD_CAM_LCD_BIT_ORDER
  - 0: Do not invert.
  - 1: Invert the input data bit order.
- LCD_CAM_LCD_BYTE_ORDER
  - 0: Do not invert.
  - 1: Invert data byte order, only valid in 16/24-bit modes.

For the detailed configuration, see Table 40.3-2.

Table 40.3-2. LCD Data Format Control – Stage One

GDMA Data Order[1]	LCD_CAM_LCD_BYTE_MODE	LCD_CAM_LCD_BIT_ORDER	LCD_CAM_LCD_BYTE_ORDER[2]	Output Data Order[3,4]
B0,B1,B2,B3,B4,B5	0	0	0	{B0}{B1}{B2}{B3}{B4}{B5}
		1	0	{B0'}{B1'}{B2'}{B3'}{B4'}{B5'}
	1	0	0	{B1,B0}{B3,B2}{B5,B4}
			1	{B0,B1}{B2,B3}{B4,B5}
		1	0	{B1',B0'}{B3',B2'}{B5',B4'}
			1	{B0',B1'}{B2',B3'}{B4',B5'}
	2	0	0	{B2,B1,B0}{B5,B4,B3}
			1	{B0,B1,B2}{B3,B4,B5}
		1	0	{B2',B1',B0'}{B5',B4',B3'}
			1	{B0',B1',B2'}{B3',B4',B5'}

[1] B0 ~ B5 represent the bytes of the data from GDMA, from low address to high address.

[2] Only the configuration listed in the table is valid. Other configurations may cause unexpected data errors.

[3] In output data, the bits in {} are in big-endian, and are in parallel with each other, while the data of {} is in serial with the data of other {}. Data is sent out from left to right. Take {B0}{B1}{B2}{B3} as an example. The bits in {B0} are in parallel with each other, but are in serial with the bits in {B1}{B2}{B3}. {B0} is sent out first.

[4] In output data, Bn'[7:0] = Bn[0:7] (n = 0,1,2,3,4,5).

Stage Two: Data Color Space Conversion

At this stage, the data undergoes color space conversion. For detailed configurations, please refer to Section 40.3.6.

> **Note:**
> Users can skip Stage Two by configuring LCD_CAM_LCD_CONV_ENABLE as 0, in which case the output of Stage One will directly become the input of Stage Three.

Stage Three: Data Post-processing

At this stage, configure the following registers to adjust the bit and byte order of the output data.

- LCD_CAM_LCD_WIRE_MODE
  - 0: The bit width of the data transmitted to GPIO is 8 bits.
  - 1: The bit width of the data transmitted to GPIO 16 bits.
  - 2: The bit width of the data transmitted to GPIO is 24 bits.
- LCD_CAM_LCD_DOUT_BIT_ORDER
  - 0: Do not invert.
  - 1: Invert the output data bit order.
- LCD_CAM_LCD_DOUT_BYTE_SWIZZLE_ENABLE
  - 0: Disable LCD output data byte reordering.
  - 1: Enable LCD output data byte reordering by configuring LCD_CAM_LCD_DOUT_BYTE_SWIZZLE_MODE. For the reordering results see Table 40.3-3.

Table 40.3-3. LCD Output Data Reordering

Stage Three Input Data Byte Order[1]	LCD_CAM_LCD_DOUT_BYTE_SWIZZLE_MODE[2]	Stage Three Output Data Byte Order[3]
B0,B1,B2,B3,B4,B5	0	B1,B0,B3,B2,B5,B4
	1	B0,B2,B1,B3,B5,B4
	2	B1,B0,B2,B4,B3,B5
	3	B1,B2,B0,B4,B5,B3
	4	B2,B0,B1,B5,B3,B4
	5	B2,B1,B0,B5,B4,B3

[1] B0 ~ B5 represent the bytes of the input data in stage three, i.e., the input data is arranged in a little-endian byte sequence. For example, when the valid data width of the input data is 16 bits, the original input data sequence B1,B0B3,B2B5,B4 is converted to B0,B1,B2,B3,B4,B5.

[2] Only the values listed in the table are valid. Other values may cause unexpected data errors.

[3] The byte order of data that is ultimately transmitted to GPIO is determined by the "Stage Three Output Data Byte Order" and the bit width. For example, if the "Stage Three Output Data Byte Order" is B1,B0,B2,B4,B3,B5 and the transmitted data width is 16 bits, then the transmitted data byte order would be {B0,B1}{B4,B2}{B5,B3}. In this sequence, the data within each {} is in big-endian parallel format, and the data in each {} is in serial with the data in other {}. Data is transmitted from left to right. In this particular example, the data in {B0,B1} are in parallel with each other, while the data in {B0,B1} is in serial with the data in {B4,B2} and {B5,B3}. {B0,B1} is transmitted first.

40.3.5.2 Camera Data Format Control

When the Camera module receives data, configure the following registers to adjust the bit/byte order of the data sent to GDMA.

- LCD_CAM_CAM_2BYTE_EN
  - 0: The data width of the Camera input is 8 bits.
  - 1: The data width of the Camera input is 16 bits.
- LCD_CAM_CAM_BIT_ORDER
  - 0: Do not invert.
  - 1: Invert data bit order.
    - Invert CAM_DATA_in[7:0] to CAM_DATA_in[0:7] in 8-bit mode.
    - Invert CAM_DATA_in[15:0] to CAM_DATA_in[0:15] in 16-bit mode.
- LCD_CAM_CAM_BYTE_ORDER
  - 0: Do not invert.
  - 1: Invert data byte order, only valid in 16-bit mode.

For the detailed configuration, see Table 40.3-4.

Table 40.3-4. Camera Data Format Control

RX Data Order[1]	LCD_CAM_CAM_2BYTE_EN	LCD_CAM_CAM_BIT_ORDER[2]	LCD_CAM_CAM_BYTE_ORDER[2]	GDMA Data Order[3,4]
{B0}{B1}{B2}{B3}	0	0	0	B0,B1,B2,B3
		1	0	B0',B1',B2',B3'
{B1,B0}{B3,B2}	1	0	0	B0,B1,B2,B3
			1	B1,B0,B3,B2
		1	0	B0',B1',B2',B3'
			1	B1',B0',B3',B2'

[1] In RX data, the bits in {} are in big-endian, and are in parallel with each other, while the data in {} is in serial with the data in other {}. Data is received from left to right. Take {B0}{B1}{B2}{B3} as an example. The bits in {B0} are in parallel with each other, but are in serial with the bits in {B1}{B2}{B3}. {B0} is received first.

[2] Only the configuration listed in the table is valid. Other configurations may cause unexpected data errors.

[3] B0 ~ B3 represent the bytes of the data to GDMA, from low address to high address.

[4] In the data to GDMA, Bn'[7:0] = Bn[0:7] (n = 0,1,2,3).

Note:
If only one byte is received each time, CAM_Data_in[7:0] is valid data. For such case, users must connect CAM_Data_in[7:0] with the master.

40.3.6 YUV-RGB Data Format Conversion

LCD_CAM is capable of converting data formats between YUV and RGB. The LCD module and Camera module each have a data format converter. The converters support format conversion:

- under BT601 and BT709 standards
- from RGB565 to YUV422/420/411 (full/limited range)
- from YUV422/YUV411 (full/limited range) to RGB565
- between YUV422/411 (full/limited range) formats

The converter in the LCD module also supports conversion:

- from YUV422/411 (full/limited range) to RGB888 (full/limited range)
- from RGB565 to YUV444 (full/limited range)
- between RGB888 (full/limited range) and RGB565
- from YUV422/411 (full/limited range) to YUV444 (full/limited range)

40.3.6.1 YUV Formats

In LCD_CAM module, assume that there are 8 pixels to be transmitted, corresponding to YUV data [Y_i, U_i, V_i] (i = 1 ~ 8). Then:

- In YUV422 mode, the LCD sends (or the camera receives) the data as follows:

Y_1	U_1	Y_2	V_2	Y_3	U_3	Y_4	V_4	Y_5	U_5	Y_6	V_6	Y_7	U_7	Y_8	V_8

- In YUV420 mode, the LCD and the camera do not participate in format conversion.
- In YUV411 mode, the LCD sends (or the camera receives) the data as follows:

Y_1	U_1	Y_2	Y_3	V_3	Y_4	Y_5	U_5	Y_6	Y_7	V_7	Y_8

40.3.6.2 Format Conversion Configuration

The configuration process for the format conversion in the Camera module is almost identical to that in the LCD module. Therefore, we will illustrate the process using the example of format conversion in the LCD module.

1. Enable YUV-RGB format converter by setting LCD_CAM_LCD_CONV_ENABLE.
2. Configure the valid bit width of the input data by configuring LCD_CAM_LCD_CONV_MODE_8BITS_ON:
   - 0: The valid bit width is 16 bits.
   - 1: The valid bit width is 8 bits.
3. Configure the valid bit width of the output data:
   - For the format conversion in the Camera module, the valid bit width of the output data is equal to that of the input data.
   - For the format conversion in the LCD module, when LCD_CAM_LCD_WIRE_MODE is:
     - 2: The valid bit width of the format converter's output data is 24 bits.
     - Other value: The valid bit width of the format converter's output data is equal to that of the input data.
4. Select the standard by configuring LCD_CAM_LCD_CONV_PROTOCOL_MODE:
   - 0: BT601 standard
   - 1: BT709 standard
5. Configure the conversion mode:
   - In 24-bit output mode:

Table 40.3-5. Conversion Mode Control in 24-bit Mode

Conversion Mode	TRANS_MODE[1]	YUV_MODE[2]	YUV2YUV_MODE[3]	RGB2RGB_MODE[4]
RGB565 → YUV444	1	-	3	-
RGB565 → RGB888	0	-	-	1
YUV422 → RGB888	0	0	3	-
YUV411 → RGB888	0	2	3	-
YUV422 → YUV444	1	0	0/1/2	-
YUV411 → YUV444	1	2	0/1/2	-

[1] The value of LCD_CAM_LCD_CONV_TRANS_MODE
[2] The value of LCD_CAM_LCD_CONV_YUV_MODE
[3] The value of LCD_CAM_LCD_CONV_YUV2YUV_MODE
[4] The value of LCD_CAM_LCD_CONV_RGB2RGB_MODE

- In other output modes:

Table 40.3-6. Conversion Mode Control in Other Modes

Conversion Mode	TRANS_MODE[1]	YUV_MODE[2]	YUV2YUV_MODE[3]	RGB2RGB_MODE[4]
RGB565 → YUV422	1	0	3	-
RGB565 → YUV420	1	1	3	-
RGB565 → YUV411	1	2	3	-
RGB888 → RGB565	0	-	-	1
YUV422 → RGB565	0	0	3	-
YUV411 → RGB565	0	2	3	-
YUV422 → YUV420	1	0	1	-
YUV422 → YUV411	1	0	2	-
YUV411 → YUV422	1	2	0	-
YUV411 → YUV420	1	2	1	-

[1] The value of LCD_CAM_LCD_CONV_TRANS_MODE
[2] The value of LCD_CAM_LCD_CONV_YUV_MODE
[3] The value of LCD_CAM_LCD_CONV_YUV2YUV_MODE
[4] The value of LCD_CAM_LCD_CONV_RGB2RGB_MODE

6. Configure the color range for the input data by configuring LCD_CAM_LCD_CONV_DATA_IN_MODE:
   - 0: limited color range[1]
   - 1: full color range[2]
7. Configure the color range for the output data by configuring LCD_CAM_LCD_CONV_DATA_OUT_MODE:
   - 0: limited color range[1]
   - 1: full color range[2]

Note:

1. If the limited color range is selected,

- the color range of RGB is: 16 ~ 240.
- the color range of YUV is:
  - Y: 16 ~ 240.
  - U-V: 16 ~ 235.

2. If the full color range is selected, the color range of RGB or YUV is 0 ~ 255.

40.3.7 LCD_CAM Timing

40.3.7.1 LCD Timing (RGB Format)

Figure 40.3-4 shows the LCD frame structure.

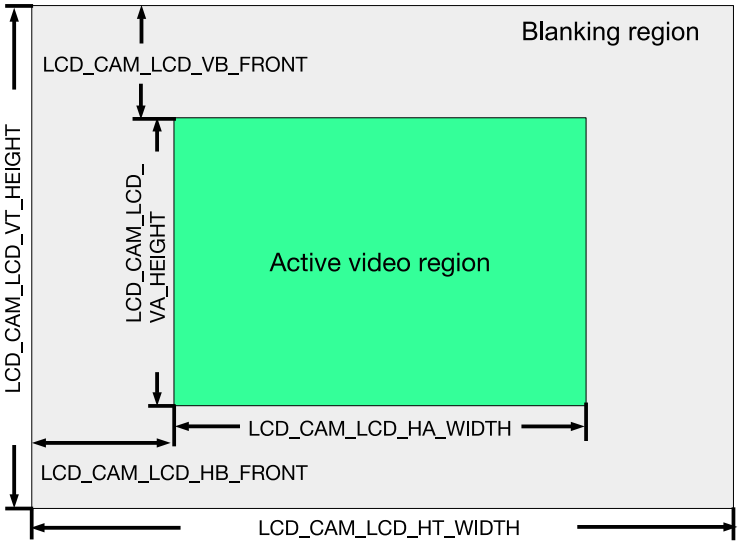


Figure 40.3-4. LCD Frame Structure

As depicted in the figure, the frame structure can be set up with the following registers:

- LCD_CAM_LCD_VT_HEIGHT
- LCD_CAM_LCD_VA_HEIGHT
- LCD_CAM_LCD_HB_FRONT
- LCD_CAM_LCD_HT_WIDTH
- LCD_CAM_LCD_HA_WIDTH
- LCD_CAM_VB_FRONT

Figure 40.3-5 shows the timing of an LCD full frame.

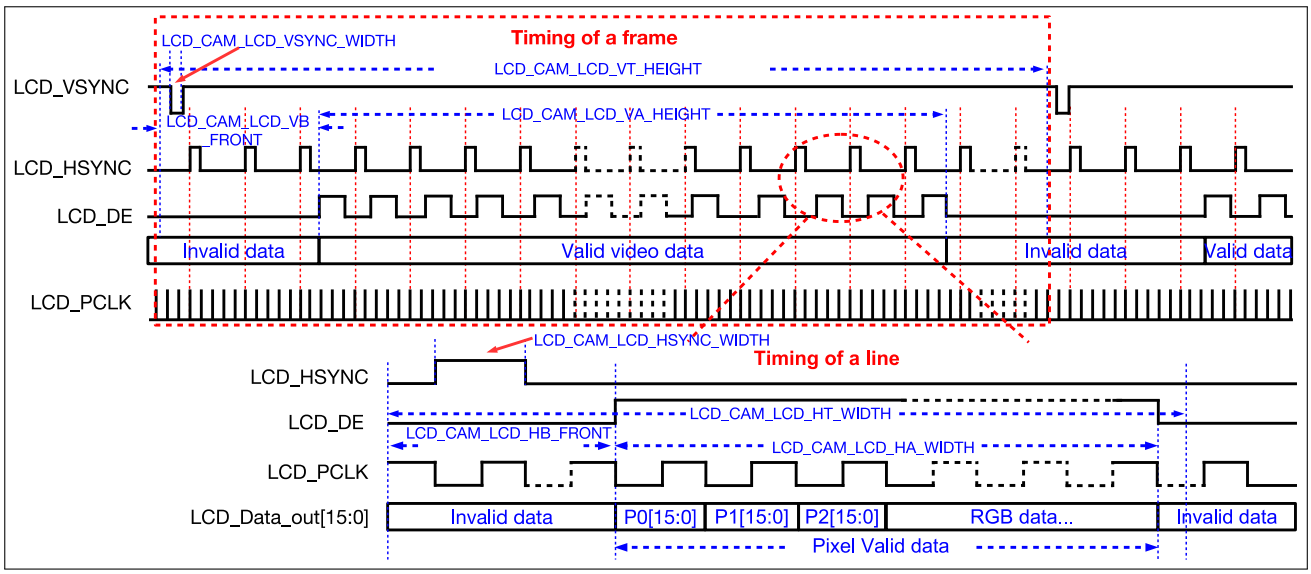


Figure 40.3-5. LCD Timing (RGB Format)

Note:

When configuring the parameters shown in the figures above, please note that the parameter is equal to the register value + 1. For example, if the expected VSYNC width is 1, then please configure LCD_CAM_LCD_VSYNC_WIDTH to 0. For more information, see the register description.

40.3.7.2 LCD Timing (I8080/MOTO6800 Format)

Figure 40.3-6 shows the LCD timing sequence in I8080/MOTO6800 format.

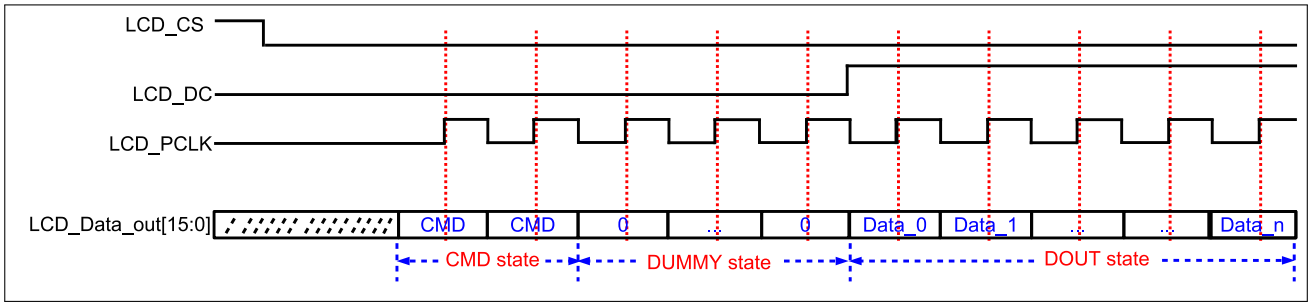


Figure 40.3-6. LCD Timing (I8080/MOTO6800 Format)

40.4 Interrupts

ESP32-P4's LCD_CAM can generate the following interrupt signal that will be sent to the Interrupt Matrix.

- LCD_CAM_INTR

The following internal interrupt sources from LCD_CAM can generate LCD_CAM_INTR:

- LCD_CAM_CAM_HS_INT: Triggered when the Camera module receives lines more than or equal to the value of LCD_CAM_CAM_LINE_INT_NUM + 1.
- LCD_CAM_CAM_VSYNC_INT: Triggered when the Camera module receives a whole frame.

- LCD_CAM_LCD_TRANS_DONE_INT: Triggered when the LCD module finishes the transmission.
- LCD_CAM_LCD_VSYNC_INT: Triggered when the LCD module transmits a whole frame.

Note:
For definitions of *interrupt*, *interrupt signal*, *interrupt source*, and their correlations, please refer to Chapter 13 *Interrupt Matrix* > Section 13.2 *Interrupt Terminology in ESP32-P4*.

40.5 Software Configuration Process

Note:
Updating the configurations of the LCD module or the Camera module requires setting the LCD_CAM_LCD_UPDATE and LCD_CAM_CAM_UPDATE respectively to synchronize registers from the APB clock domain to the LCD/camera clock domain. Check the configuration examples below for more details.

40.5.1 Configure LCD (RGB Format) as TX Mode

Follow the steps below to configure LCD (RGB format) as TX mode via software:

1. Configure clock according to Section 40.3.3.
2. Configure signal pins according to Table 40.3-1 and Chapter 10 *GPIO Matrix and IO MUX*.
3. Enable corresponding interrupts, see Section 40.4.
4. Enable RGB format by setting LCD_CAM_LCD_RGB_MODE_EN.
5. Configure frame structure according to Section 40.3.7.1.
6. Set LCD_CAM_LCD_UPDATE so that the configurations take effect.
7. Reset TX control unit (LCD_Ctrl) and Async TX FIFO as described in Section 40.3.4.
8. Configure GDMA outlink, see Chapter 7 *VDMA Controller (VDMA)*.
9. Start transmitting data:
   - Wait till LCD slave gets ready.
   - Set LCD_CAM_LCD_START to start transmitting data.
10. Wait for the interrupt signals set in Step 3.
11. In frame intervals during data transmission, check if LCD_CAM_LCD_NEXT_FRAME_EN is set:
    - If yes, set LCD_CAM_LCD_UPDATE (repeating the steps above) to continue transmitting next frame.
    - If not, LCD stops transmitting data.
12. Clear LCD_CAM_LCD_START if data transmission is done.

40.5.2 Configure LCD (I8080/MOTO6800 Format) as TX Mode

Follow the steps below to configure LCD (I8080/MOTO6800 format) as TX mode via software:

1. Configure clock according to Section 40.3.3.

2. Configure signal pins according to Table 40.3-1 and Chapter 10 *GPIO Matrix and IO MUX*.
3. Enable corresponding interrupts, see Section 40.4.
4. Clear LCD_CAM_LCD_RGB_MODE_EN to disable RGB format.
5. Configure CMD phase with LCD_CAM_LCD_CMD, LCD_CAM_LCD_CMD_2_CYCLE_EN, and LCD_CAM_LCD_LATTER_CMD_VALUE.
6. Configure DUMMY phase with LCD_CAM_LCD_DUMMY and LCD_CAM_LCD_DUMMY_CYCLELEN.
7. Configure DOUT phase with LCD_CAM_LCD_DOUT, depending on the output mode:
   - In a fixed-length output[1], configure data length in LCD_CAM_LCD_DOUT_CYCLELEN.
   - In a continuous output[2], set LCD_CAM_LCD_ALWAYS_OUT_EN. Users do not need to configure LCD_CAM_LCD_DOUT_CYCLELEN.

   Note:

   (a) In a fixed-length output, the LCD module stops sending data once the data length reaches the value set in LCD_CAM_LCD_DOUT_CYCLELEN.

   (b) In a continuous output, LCD module keeps sending data till:

   - LCD_CAM_LCD_START is cleared;
   - or LCD_CAM_LCD_RESET is set;
   - or all the data in GDMA is sent out.

8. Configure the CD signal mode, including the default value of the CD signal and the values at each phase, see the description of LCD_CAM_LCD_MISC_REG.
9. Set LCD_CAM_LCD_UPDATE so that the configurations take effect.
10. Reset TX control unit (LCD_Ctrl) and Async TX FIFO as described in Section 40.3.4.
11. Configure GDMA outlink, see Chapter 7 *VDMA Controller (VDMA)*.
12. Start transmitting data:
    - Wait till LCD slave gets ready.
    - Set LCD_CAM_LCD_START to start transmitting data.
13. Wait for the interrupt signals set in Step 2.
14. Clear LCD_CAM_LCD_START if data transmission is done.

Notes:

No matter in which format, RGB or I8080/MOTO6800, the rules below must be followed when accessing internal and external memory via GDMA:

- If LCD data bus is configured to 8-bit parallel output mode, then
  - The pixel clock frequency must be less than 80 MHz.
  - If YUV-RGB format conversion is being used at the same time, the pixel clock frequency must be less than 60 MHz.
- If LCD data bus is configured to 16-bit parallel output mode, then

- The pixel clock frequency must be less than 40 MHz.
- If YUV-RGB format conversion is being used at the same time, the pixel clock frequency must be less than 30 MHz.

40.5.3 Configure Camera as RX Mode

Follow the steps below to configure the Camera as RX mode via software:

1. Configure clock according to Section 40.3.3. Note that in slave mode, the module clock frequency should be twice as fast as the PCLK frequency of the image sensor.
2. Configure signal pins according to Table 40.3-1 and Chapter 10 *GPIO Matrix and IO MUX*.
3. Set or clear LCD_CAM_CAM_VH_DE_MODE_EN according to the control signal HSYNC.
4. Set needed RX data format according to Section 40.3.5.2.
5. Set LCD_CAM_CAM_UPDATE so that the configurations take effect.
6. Reset RX control unit (Camera_Ctrl) and Async RX FIFO as described in Section 40.3.4.
7. Enable corresponding interrupts, see Section 40.4.
8. Configure GDMA inlink, and set the length of RX data in LCD_CAM_CAM_REC_DATA_BYTELEN.
9. Start receiving data:
   - In master mode, when the slave is ready, set LCD_CAM_CAM_START to start receiving data.
   - In slave mode, set LCD_CAM_CAM_START. Receiving data starts after the master provides clock signal and control signal.
10. Receive data and store the data to GDMA. Then corresponding interrupts set in Step 7 will be generated.

Notes:

- No matter in which operation mode, camera master RX mode or camera slave RX mode, the rules below must be followed when accessing internal memory via GDMA:
  - If 8-bit parallel data input mode is selected, then
    * The pixel clock frequency must be less than 80 MHz.
    * If YUV-RGB format is being used at the same time, the pixel clock frequency must be less than 60 MHz.
  - If 16-bit parallel data input mode is selected, then
    * The pixel clock frequency must be less than 40 MHz.
    * If YUV-RGB format conversion is being used at the same time, the pixel clock frequency must be less than 30 MHz.

Note:
If both LCD and camera are connected externally at the same time, it is necessary to ensure that the maximum data throughput on the interface is less than the total data bandwidth of GDMA when accessing internal/external storage. The default frequency of APB_CLK is 80 MHz in this scenario. For more information about APB_CLK, see Chapter 11 *Reset and Clock*.

40.6 Register Summary

The addresses in this section are relative to LCD_CAM base address provided in Table 9.3-2 in Chapter 9 *System and Memory*.

The abbreviations given in Column **Access** are explained in Section *Access Types for Registers*.

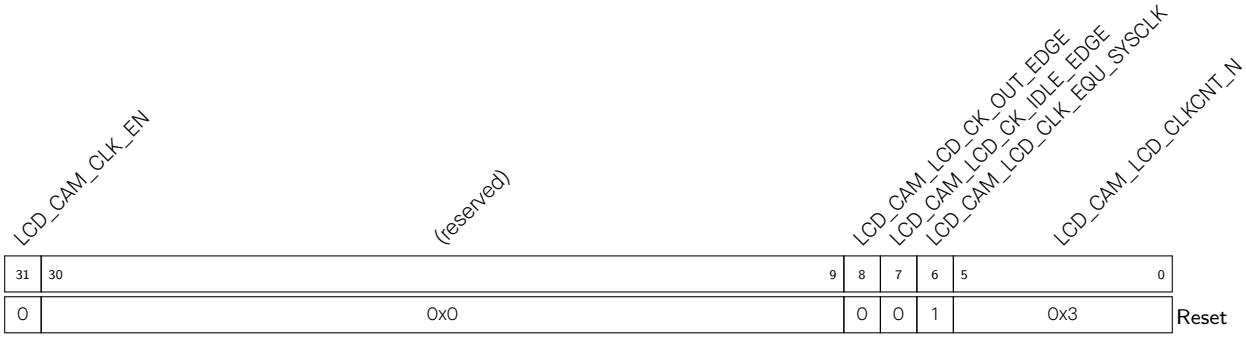
Name	Description	Address	Access
LCD Module Configuration Registers			
LCD_CAM_LCD_CLOCK_REG	LCD clock configuration register	0x0000	R/W
LCD_CAM_LCD_RGB_YUV_REG	LCD data format conversion register	0x0010	R/W
LCD_CAM_LCD_USER_REG	LCD user configuration register	0x0014	varies
LCD_CAM_LCD_MISC_REG	LCD MISC configuration register	0x0018	varies
LCD_CAM_LCD_CTRL_REG	LCD signal configuration register	0x001C	R/W
LCD_CAM_LCD_CTRL1_REG	LCD signal configuration register 1	0x0020	R/W
LCD_CAM_LCD_CTRL2_REG	LCD signal configuration register 2	0x0024	R/W
LCD_CAM_LCD_FIRST_CMD_VAL_REG	LCD command value configuration register	0x0028	R/W
LCD_CAM_LCD_LATTER_CMD_VAL_REG	LCD command value configuration register	0x002C	R/W
LCD_CAM_LCD_DLY_MODE_CFG1_REG	LCD data/signal delay configuration register	0x0030	R/W
LCD_CAM_LCD_DLY_MODE_CFG2_REG	LCD signal delay mode configuration register	0x0038	R/W
Camera Module Configuration Registers			
LCD_CAM_CAM_CTRL_REG	Camera clock configuration	0x0004	R/W
LCD_CAM_CAM_CTRL1_REG	Camera control register	0x0008	varies
LCD_CAM_CAM_RGB_YUV_REG	Camera data format conversion register	0x000C	R/W
Interrupt Registers			
LCD_CAM_LC_DMA_INT_ENA_REG	LCD_CAM GDMA interrupt enable register	0x0064	R/W
LCD_CAM_LC_DMA_INT_RAW_REG	LCD_CAM GDMA raw interrupt status register	0x0068	RO
LCD_CAM_LC_DMA_INT_ST_REG	LCD_CAM GDMA masked interrupt status register	0x006C	RO
LCD_CAM_LC_DMA_INT_CLR_REG	LCD_CAM GDMA interrupt clear register	0x0070	WO
Version Control Register			
LCD_CAM_LC_DATE_REG	Version control register	0x00FC	R/W

40.7 Registers

The addresses in this section are relative to LCD_CAM base address provided in Table 9.3-2 in Chapter 9 *System and Memory*.

For how to program reserved fields, please refer to Section Programming Reserved Register Field.

Register 40.1. LCD_CAM_LCD_CLOCK_REG (0x0000)



LCD_CAM_LCD_CLKCNT_N Determines the frequency of the LCD pixel clock LCD_PCLK together with LCD_CAM_LCD_CLK_EQU_SYSCLK.

When LCD_CAM_LCD_CLK_EQU_SYSCLK = 0,

$f_{LCD_PCLK} = f_{LCD_CLK}/$(LCD_CAM_LCD_CLKCNT_N + 1)

Note: this field must not be configured to 0.

(R/W)

LCD_CAM_LCD_CLK_EQU_SYSCLK Determines the frequency of the LCD pixel clock LCD_PCLK together with LCD_CAM_LCD_CLK_EQU_SYSCLK.

0: $f_{LCD_PCLK} = f_{LCD_CLK}/$(LCD_CAM_LCD_CLKCNT_N + 1)

1: $f_{LCD_PCLK} = f_{LCD_CLK}$

(R/W)

LCD_CAM_LCD_CK_IDLE_EDGE Indicates the level of LCD_PCLK in idle state.

0: LCD_PCLK is low.

1: LCD_PCLK is high.

(R/W)

LCD_CAM_LCD_CK_OUT_EDGE Indicates the level of LCD_PCLK in the first half clock cycle.

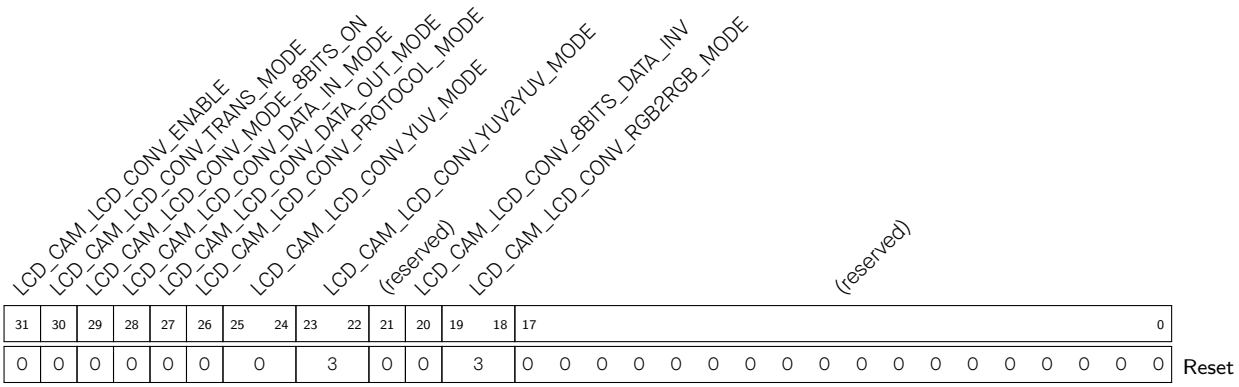
0: LCD_PCLK is low.

1: LCD_PCLK is high.

(R/W)

LCD_CAM_CLK_EN Set this bit to force enable the clock for all configuration registers. Clock gate is not used. (R/W)

Register 40.2. LCD_CAM_LCD_RGB_YUV_REG (0x0010)



LCD_CAM_LCD_CONV_RGB2RGB_MODE Configures data format conversion in RGB-to-RGB mode.

0: Data is converted to RGB565 format.

1: Data is converted to RGB888 format.

2/3: Data format conversion is disabled.

(R/W)

LCD_CAM_LCD_CONV_8BITS_DATA_INV Configures whether to swap every two 8-bit GDMA input data.

0: Do not swap.

1: Swap.

(R/W)

LCD_CAM_LCD_CONV_YUV2YUV_MODE Configures data format conversion in YUV-to-YUV mode.

0: Data is converted to YUV422 format.

1: Data is converted to YUV420 format.

2: Data is converted to YUV411 format.

3: Data format conversion is disabled.

Only valid when LCD_CAM_LCD_CONV_TRANS_MODE = 1.

(R/W)

LCD_CAM_LCD_CONV_YUV_MODE Configures the YUV format of input data in YUV-to-YUV mode and YUV-to-RGB mode.

0: Input data is in YUV422 format.

1: Input data is in YUV420 format.

2: Input data is in YUV411 format.

(R/W)

LCD_CAM_LCD_CONV_PROTOCOL_MODE Configures the data format conversion standard.

0: BT601.

1: BT709.

(R/W)

LCD_CAM_LCD_CONV_DATA_OUT_MODE Configures the color range for output data.

0: Limited color range.

1: Full color range.

(R/W)

Continued on the next page...

Register 40.2. LCD_CAM_LCD_RGB_YUV_REG (0x0010)

Continued from the previous page...

LCD_CAM_LCD_CONV_DATA_IN_MODE Configures the color range for input data.

0: Limited color range.

1: Full color range.

(R/W)

LCD_CAM_LCD_CONV_MODE_8BITS_ON Configures the valid bit width of the YUV-RGB converter's input data.

0: 16 bits.

1: 8 bits.

(R/W)

LCD_CAM_LCD_CONV_TRANS_MODE Configures the RGB or YUV format for data conversion.

0: Converted to RGB format.

1: Converted to YUV format.

(R/W)

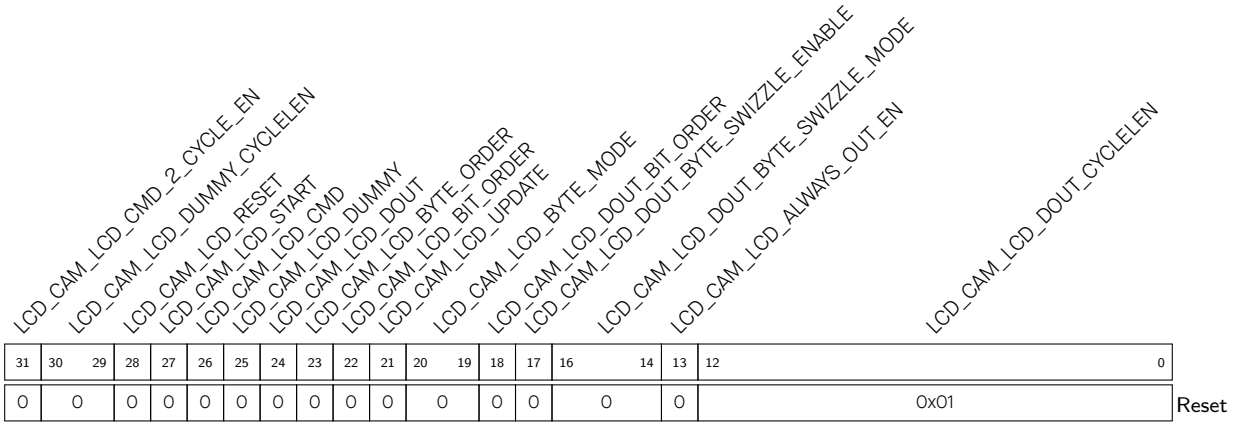
LCD_CAM_LCD_CONV_ENABLE Configures whether to enable YUV-RGB converter.

0: Bypass the converter.

1: Enable the converter.

(R/W)

Register 40.3. LCD_CAM_LCD_USER_REG (0x0014)



LCD_CAM_LCD_DOUT_CYCLELEN Configures the period of the LCD module's data output phase (DOUT). The actual period duration = the value of this field + 1. (R/W)

LCD_CAM_LCD_ALWAYS_OUT_EN Enables continuous output of the LCD module. In this mode, the LCD module continuously outputs data during the DOUT phase, until LCD_CAM_LCD_START is cleared or LCD_CAM_LCD_RESET is set. (R/W)

LCD_CAM_LCD_DOUT_BYTE_SWIZZLE_MODE Configures LCD output data byte reordering. For details please refer to Table 40.3-3. (R/W)

LCD_CAM_LCD_DOUT_BYTE_SWIZZLE_ENABLE Configures whether to enable LCD output data byte reordering.

0: Disable

1: Enable

(R/W)

LCD_CAM_LCD_DOUT_BIT_ORDER Configures whether to invert the LCD output data bit order.

0: Do not invert.

1: Invert the bit order. In n-bit mode, LCD_DATA_in[n-1:0] is inverted to LCD_DATA_in[0:n-1].

(R/W)

LCD_CAM_LCD_BYTE_MODE Configures the data bit width from GDMA.

0: 8-bit

1: 16-bit

2: 24-bit

(R/W)

LCD_CAM_LCD_UPDATE Configures whether to update the LCD register configurations.

0: Do not update.

1: Update configurations. This bit is cleared by hardware.

(R/W)

LCD_CAM_LCD_BIT_ORDER Configures whether to invert the bit order of the LCD input data.

0: Do not invert.

1: Invert the bit order. In n-bit mode, LCD_DATA_in[n-1:0] is invertd to LCD_DATA_in[0:n-1].

(R/W)

Continued on the next page...

Register 40.3. LCD_CAM_LCD_USER_REG (0x0014)

Continued from the previous page...

LCD_CAM_LCD_BYTE_ORDER Configures whether to invert the byte order of the LCD input data.

0: Do not invert.

1: Invert data byte order, only valid in 16/24-bit mode.

(R/W)

LCD_CAM_LCD_DOUT Configures the LCD DOUT phase.

0: Disable.

1: The LCD module sends data in RGB/I8080 format.

(R/W)

LCD_CAM_LCD_DUMMY Configures the LCD DUMMY phase.

0: Disable.

1: Enable the DUMMY phase when LCD initiates.

(R/W)

LCD_CAM_LCD_CMD Configures the LCD CMD phase.

0: Disable.

1: The LCD module sends commands.

(R/W)

LCD_CAM_LCD_START When set to 1, the LCD module starts transmitting data. (R/W)

LCD_CAM_LCD_RESET When set to 1, the LCD module is reset. (WO)

LCD_CAM_LCD_DUMMY_CYCLELEN Configures DUMMY cycles. DUMMY cycles = this value + 1. (R/W)

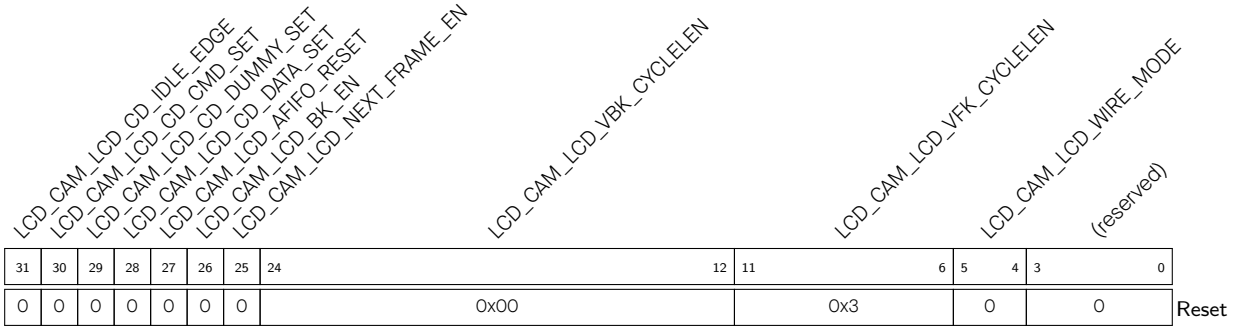
LCD_CAM_LCD_CMD_2_CYCLE_EN Configures the number of cycles of the Command phase.

0: One cycle.

1: Two cycles.

(R/W)

Register 40.4. LCD_CAM_LCD_MISC_REG (0x0018)



LCD_CAM_LCD_WIRE_MODE Configures the bit width of LCD output data to GPIO.

0: 8-bit

1: 16-bit

2: 24-bit

(R/W)

LCD_CAM_LCD_VFK_CYCLELEN Configures the clock cycles of the setup time in LCD non-RGB mode. Setup clock cycles = this value + 1. (R/W)

LCD_CAM_LCD_VBK_CYCLELEN Configures the clock cycles of the hold time in LCD non-RGB mode. Hold clock cycles = this value + 1. (R/W)

LCD_CAM_LCD_NEXT_FRAME_EN Configures whether to send the next frame.

0: The LCD module stops when the current frame is sent out.

1: The LCD module continues sending the next frame when the current frame is sent out.

(R/W)

LCD_CAM_LCD_BK_EN Configures whether to enable blanking region when LCD sends data.

0: No blanking region.

1: Enable blanking region.

(R/W)

LCD_CAM_LCD_AFIFO_RESET When set to 1, Async TX FIFO is reset. (WO)

LCD_CAM_LCD_CD_DATA_SET Configures LCD_CD value in DOUT phase for LCD I8080 mode.

0: LCD_CD = LCD_CAM_LCD_CD_IDLE_EDGE

1: LCD_CD = !LCD_CAM_LCD_CD_IDLE_EDGE

(R/W)

LCD_CAM_LCD_CD_DUMMY_SET Configures LCD_CD value in DUMMY phase for LCD I8080 mode.

0: LCD_CD = LCD_CAM_LCD_CD_IDLE_EDGE

1: LCD_CD = !LCD_CAM_LCD_CD_IDLE_EDGE

(R/W)

LCD_CAM_LCD_CD_CMD_SET Configures LCD_CD value in CMD phase for LCD I8080 mode.

0: LCD_CD = LCD_CAM_LCD_CD_IDLE_EDGE

1: LCD_CD = !LCD_CAM_LCD_CD_IDLE_EDGE

(R/W)

LCD_CAM_LCD_CD_IDLE_EDGE The default value of LCD_CD. (R/W)

Register 40.5. LCD_CAM_LCD_CTRL_REG (0x001C)



LCD_CAM_LCD_HB_FRONT Configures the value of (HSYNC_POSITION + HSYNC_WIDTH + horizontal back porch). (R/W)

LCD_CAM_LCD_VA_HEIGHT Configures the vertical active height of a frame. (R/W)

LCD_CAM_LCD_VT_HEIGHT Configures the vertical total height of a frame. (R/W)

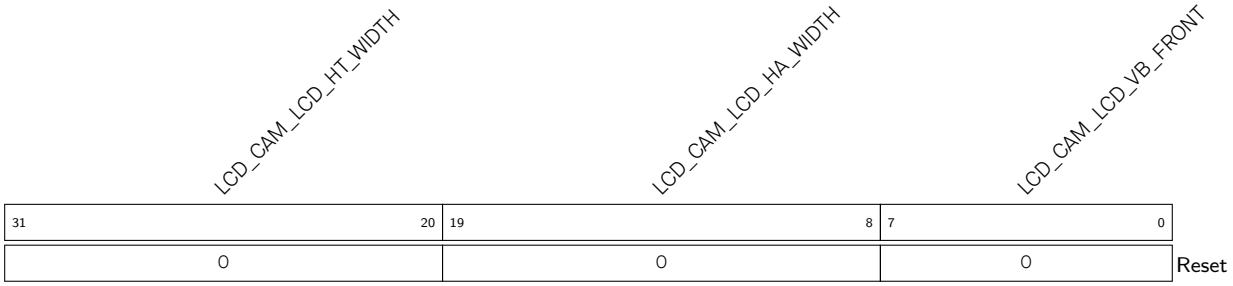
LCD_CAM_LCD_RGB_MODE_EN Configures whether to enable RGB mode.

0: Disable RGB mode.

1: Enable RGB mode and input VSYNC, HSYNC, and DE signals.

(R/W)

Register 40.6. LCD_CAM_LCD_CTRL1_REG (0x0020)

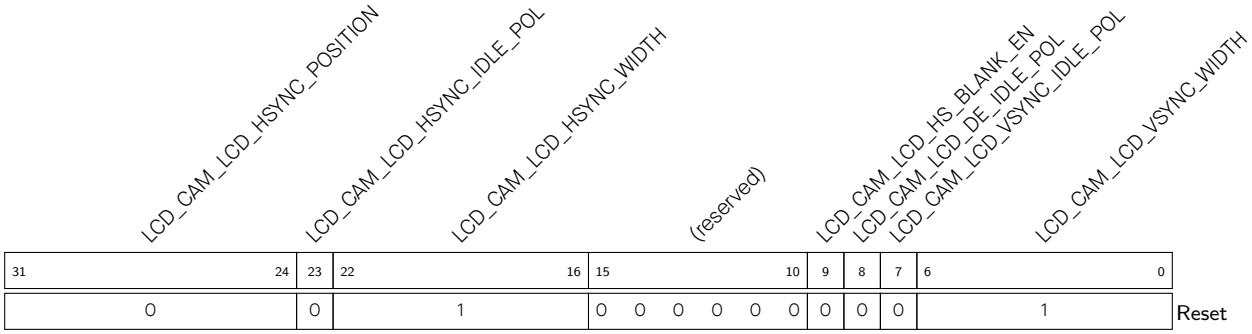


LCD_CAM_LCD_VB_FRONT Configures the value of (VSYNC_WIDTH + vertical back porch). (R/W)

LCD_CAM_LCD_HA_WIDTH Configures the horizontal active width of a frame. (R/W)

LCD_CAM_LCD_HT_WIDTH Configures the horizontal total width of a frame. (R/W)

Register 40.7. LCD_CAM_LCD_CTRL2_REG (0x0024)



LCD_CAM_LCD_VSYNC_WIDTH Configures the width of LCD_VSYNC active pulse in a line. Expected width = this value + 1. (R/W)

LCD_CAM_LCD_VSYNC_IDLE_POL Configures the idle value of LCD_VSYNC.

0: The idle value is low.

1: The idle value is high.

(R/W)

LCD_CAM_LCD_DE_IDLE_POL Configures the idle value of LCD_DE.

0: The idle value is low.

1: The idle value is high.

(R/W)

LCD_CAM_LCD_HS_BLANK_EN Configures the LCD_HSYNC output in RGB mode.

0: LCD_HSYNC is output only in active video lines.

1: LCD_HSYNC can be output in vertical blanking intervals.

(R/W)

LCD_CAM_LCD_HSYNC_WIDTH Configures the width of LCD_HSYNC active pulse. Expected width = this value +1. (R/W)

LCD_CAM_LCD_HSYNC_IDLE_POL Configures the idle value of LCD_HSYNC.

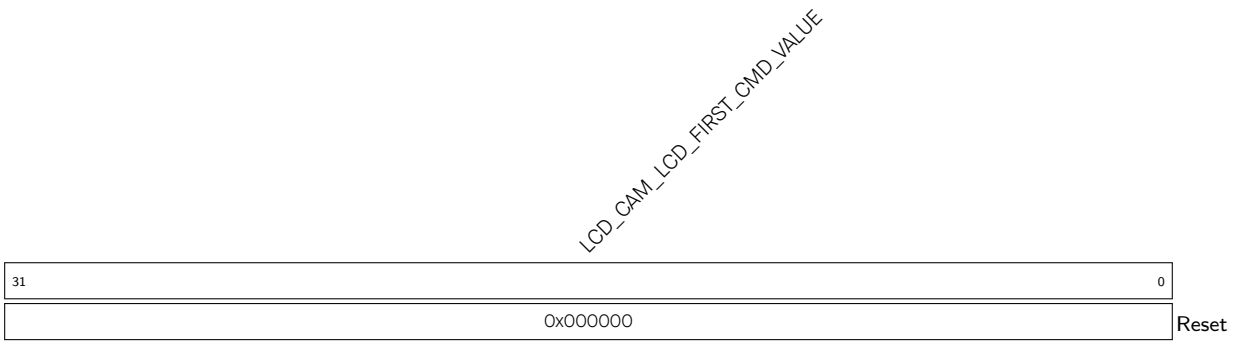
0: The idle value is low.

1: The idle value is high.

(R/W)

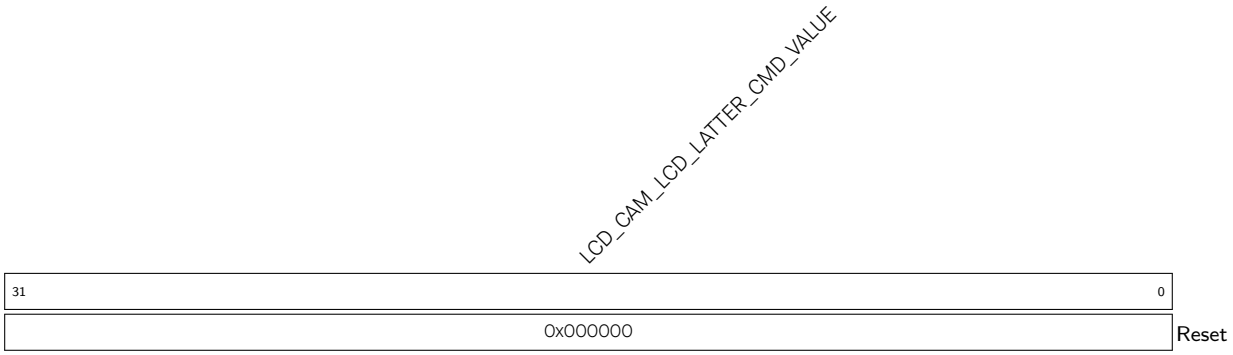
LCD_CAM_LCD_HSYNC_POSITION Configures the position of LCD_HSYNC active pulse. Expected position = this value + 1. Unit is a pixel. (R/W)

Register 40.8. LCD_CAM_LCD_FIRST_CMD_VAL_REG (0x0028)



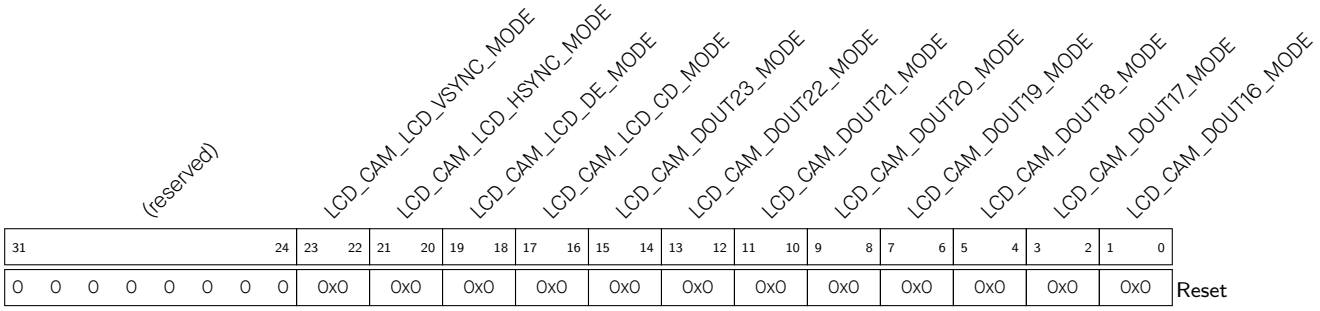
LCD_CAM_LCD_FIRST_CMD_VALUE Configures the value of the transmitted command when the LCD module is in the first cycle of the CMD phase. (R/W)

Register 40.9. LCD_CAM_LCD_LATTER_CMD_VAL_REG (0x002C)



LCD_CAM_LCD_LATTER_CMD_VALUE Configures the value of the transmitted command when the LCD module is in the second cycle of the CMD phase. (R/W)

Register 40.10. LCD_CAM_LCD_DLY_MODE_CFG1_REG (0x0030)



LCD_CAM_DOUT*n*_MODE (*n*: 16 - 23) Configures the delay of the output data bit (*n*).

0: No delay.

1: Delayed at the rising edge of LCD_CLK.

2: Delayed at the falling edge of LCD_CLK.

(R/W)

LCD_CAM_LCD_CD_MODE Configures the delay of LCD_CD output signal.

0: No delay.

1: Delayed at the rising edge of LCD_CLK.

2: Delayed at the falling edge of LCD_CLK.

(R/W)

LCD_CAM_LCD_DE_MODE Configures the delay of LCD_DE output signal.

0: No delay.

1: Delayed at the rising edge of LCD_CLK.

2: Delayed at the falling edge of LCD_CLK.

(R/W)

LCD_CAM_LCD_HSYNC_MODE Configures the delay of LCD_HSYNC output signal.

0: No delay.

1: Delayed at the rising edge of LCD_CLK.

2: Delayed at the falling edge of LCD_CLK.

(R/W)

LCD_CAM_LCD_VSYNC_MODE Configures the delay of LCD_VSYNC output signal.

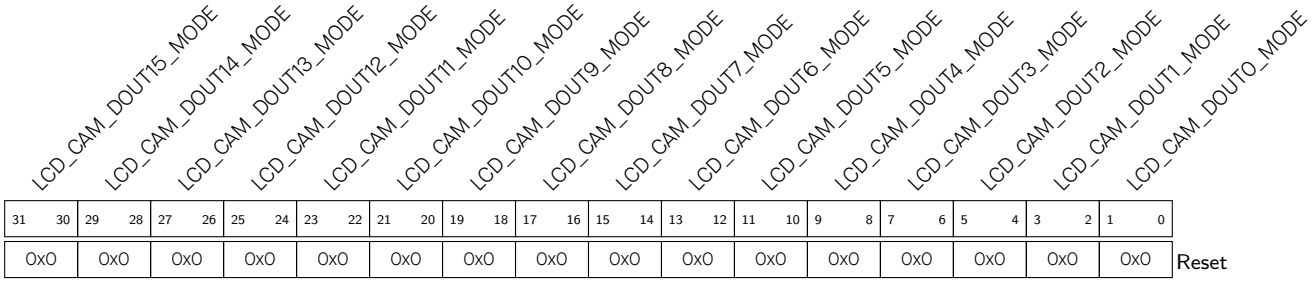
0: No delay.

1: Delayed at the rising edge of LCD_CLK.

2: Delayed at the falling edge of LCD_CLK.

(R/W)

Register 40.11. LCD_CAM_LCD_DLY_MODE_CFG2_REG (0x0038)



LCD_CAM_DOUT*n*_MODE (*n*: 0 - 15) Configures the delay of the output data bit *n*.

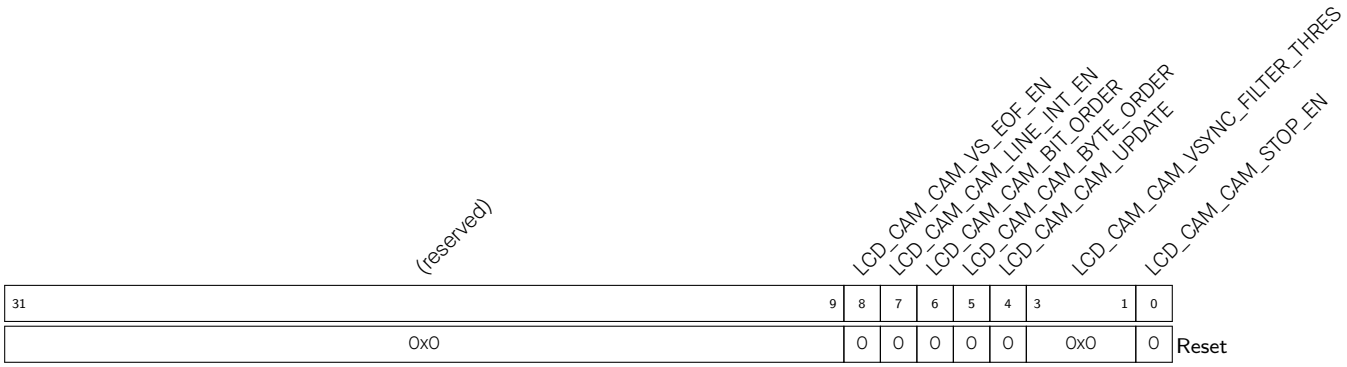
0: No delay.

1: Delayed at the rising edge of LCD_CLK.

2: Delayed at the falling edge of LCD_CLK.

(R/W)

Register 40.12. LCD_CAM_CAM_CTRL_REG (0x0004)



LCD_CAM_CAM_STOP_EN Configures whether to stop the Camera module.

0: Do not stop.

1: The Camera module stops when GDMA RX FIFO is full.

(R/W)

LCD_CAM_CAM_VSYNC_FILTER_THRES Configures the filter threshold value for CAM_VSYNC signal. (R/W)

LCD_CAM_CAM_UPDATE Configures whether to update the Camera register configurations.

0: No effect.

1: Update configurations. This bit is cleared by hardware.

(R/W)

LCD_CAM_CAM_BYTE_ORDER Configures whether to invert the byte order of the Camera's input data.

0: Do not invert.

1: Invert data byte order, only valid in 16-bit mode.

(R/W)

LCD_CAM_CAM_BIT_ORDER Configures whether to invert the bit order of the Camera's input data.

0: Do not invert.

1: Invert the bit order. In n-bit mode, CAM_DATA_in[n-1:0] is invertd to CAM_DATA_in[0:n-1]. n = 8 or 15.

(R/W)

LCD_CAM_CAM_LINE_INT_EN Configures whether the RAW bit of LCD_CAM_CAM_HS_INT can generate.

0: Disable the interrupt, i.e., LCD_CAM_CAM_HS_INT_RAW cannot generate.

1: Enable the interrupt, i.e., LCD_CAM_CAM_HS_INT_RAW can generate.

(R/W)

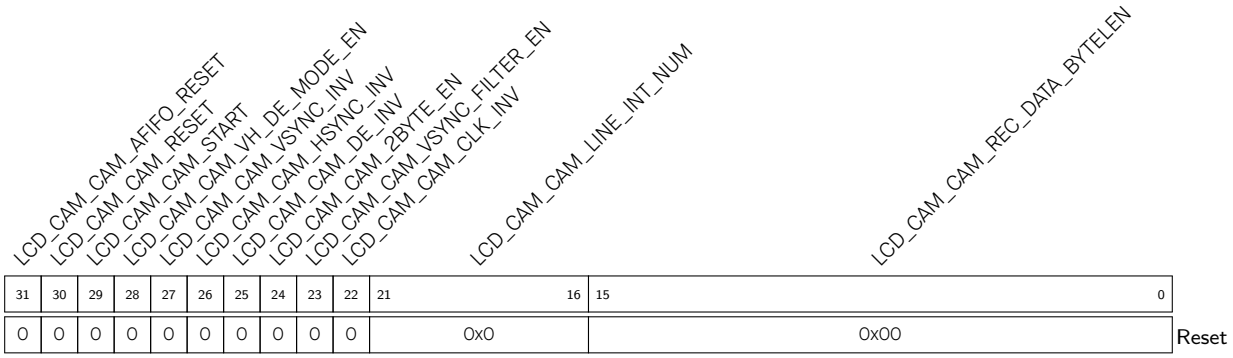
LCD_CAM_CAM_VS_EOF_EN Configures whether to enable CAM_VSYNC to generate in_suc_eof.

0: in_suc_eof is controlled by LCD_CAM_CAM_REC_DATA_BYTELEN.

1: in_suc_eof is controlled by CAM_VSYNC.

(R/W)

Register 40.13. LCD_CAM_CAM_CTRL1_REG (0x0008)



LCD_CAM_CAM_REC_DATA_BYTELEN Configures the data byte length received by the Camera module. When the length of received data reaches this value + 1, GDMA in_suc_eof_int is triggered. (R/W)

LCD_CAM_CAM_LINE_INT_NUM Configures the number of video lines. When the number of video lines reaches this value + 1, LCD_CAM_CAM_HS_INT is triggered. (R/W)

LCD_CAM_CAM_CLK_INV Configures whether to invert the input signal CAM_PCLK.

0: Do not invert.

1: Invert.

(R/W)

LCD_CAM_CAM_VSYNC_FILTER_EN Configures whether to enable CAM_VSYNC filter function.

0: Bypass.

1: Enable.

(R/W)

LCD_CAM_CAM_2BYTE_EN Configures the width of input data.

0: 8 bits.

1: 16 bits.

(R/W)

LCD_CAM_CAM_DE_INV Configures whether to invert the input signal CAM_DE.

0: Do not invert.

1: Invert.

(R/W)

LCD_CAM_CAM_HSYNC_INV Configures whether to invert the input signal CAM_HSYNC.

0: Do not invert.

1: Invert.

(R/W)

LCD_CAM_CAM_VSYNC_INV Configures whether to invert the input signal CAM_VSYNC.

0: Do not invert.

1: Invert.

(R/W)

Continued on the next page...

Register 40.13. LCD_CAM_CAM_CTRL1_REG (0x0008)

Continued from the previous page...

LCD_CAM_CAM_VH_DE_MODE_EN Configures the input control signals.

0: VSYNC and DE signals control the data. In this case, wiring HSYNC signal line is not a must. But in this case, the YUV-RGB conversion function of the camera module is not available.

1: VSYNC, HSYNC, and DE signals control the data. In this case, users need to wire the three signal lines.

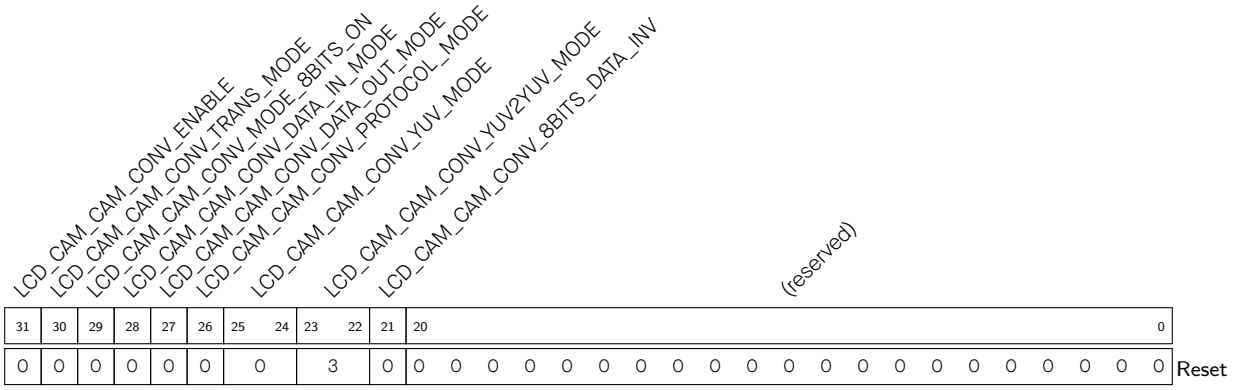
(R/W)

LCD_CAM_CAM_START Camera module start signal. (R/W)

LCD_CAM_CAM_RESET When set to 1, the Camera module is reset. (WO)

LCD_CAM_CAM_AFIFO_RESET When set to 1, the Camera Async RX FIFO is reset. (WO)

Register 40.14. LCD_CAM_CAM_RGB_YUV_REG (0x000C)


LCD_CAM_CAM_CONV_ENABLE
LCD_CAM_CAM_CONV_TRANS_MODE
LCD_CAM_CAM_CONV_MODE_8BITS_ON
LCD_CAM_CAM_CONV_DATA_IN_MODE
LCD_CAM_CAM_CONV_DATA_OUT_MODE
LCD_CAM_CAM_CONV_PROTOCOL_MODE
LCD_CAM_CAM_CONV_YUV_MODE
LCD_CAM_CAM_CONV_YUV2YUV_MODE
LCD_CAM_CAM_CONV_8BITS_DATA_INV
(reserved)
31 30 29 28 27 26 25 24 23 22 21 20 0
0 0 0 0 0 0 0 3 0 0 0 0 0 0

Register 40.14. LCD_CAM_CAM_RGB_YUV_REG (0x000C)

Continued from the previous page...

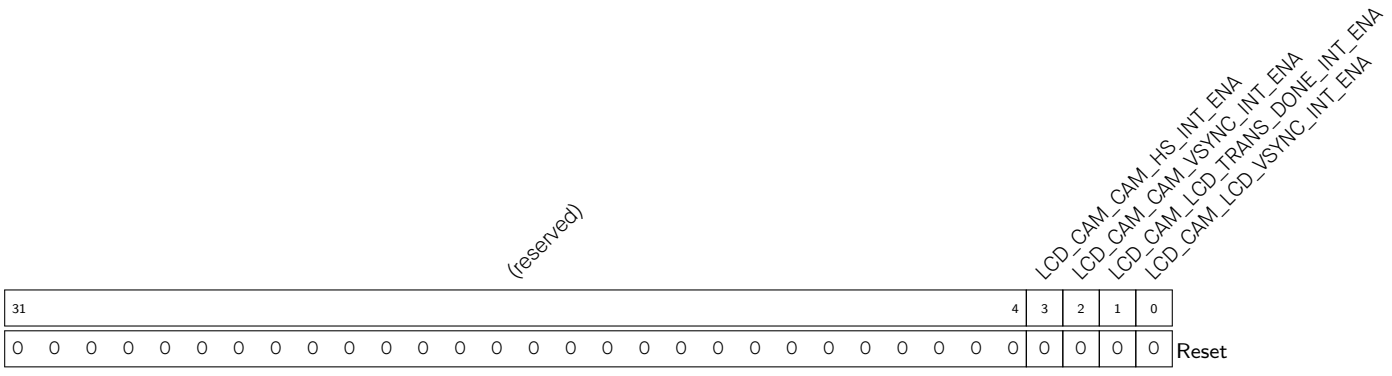
LCD_CAM_CAM_CONV_DATA_IN_MODE Configures the color range for the YUV-RGB converter's input data.
0: Limited color range.
1: Full color range.
(R/W)

LCD_CAM_CAM_CONV_MODE_8BITS_ON Configures the bit width of the YUV-RGB converter's input data.
0: 16-bit mode.
1: 8-bit mode.
(R/W)

LCD_CAM_CAM_CONV_TRANS_MODE Configures the data conversion format.
0: Data is converted to RGB format.
1: Data is converted to YUV format.
(R/W)

LCD_CAM_CAM_CONV_ENABLE Configures whether to enable the YUV-RGB converter.
0: Bypass.
1: Enable.
(R/W)

Register 40.15. LCD_CAM_LC_DMA_INT_ENA_REG (0x0064)



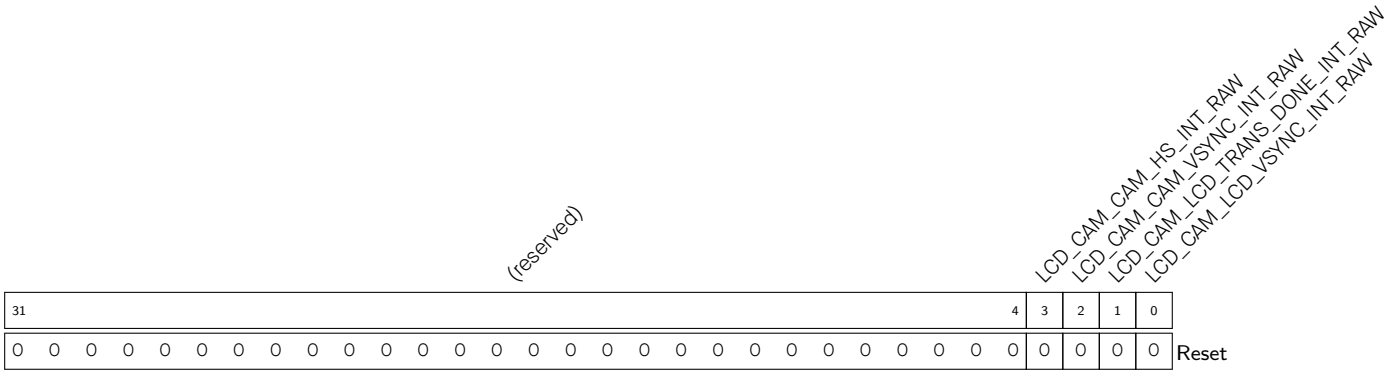
LCD_CAM_LCD_VSYNC_INT_ENA Write 1 to enable LCD_CAM_LCD_VSYNC_INT. (R/W)

LCD_CAM_LCD_TRANS_DONE_INT_ENA Write 1 to enable LCD_CAM_LCD_TRANS_DONE_INT. (R/W)

LCD_CAM_CAM_VSYNC_INT_ENA Write 1 to enable LCD_CAM_CAM_VSYNC_INT. (R/W)

LCD_CAM_CAM_HS_INT_ENA Write 1 to enable LCD_CAM_CAM_HS_INT. (R/W)

Register 40.16. LCD_CAM_LC_DMA_INT_RAW_REG (0x0068)



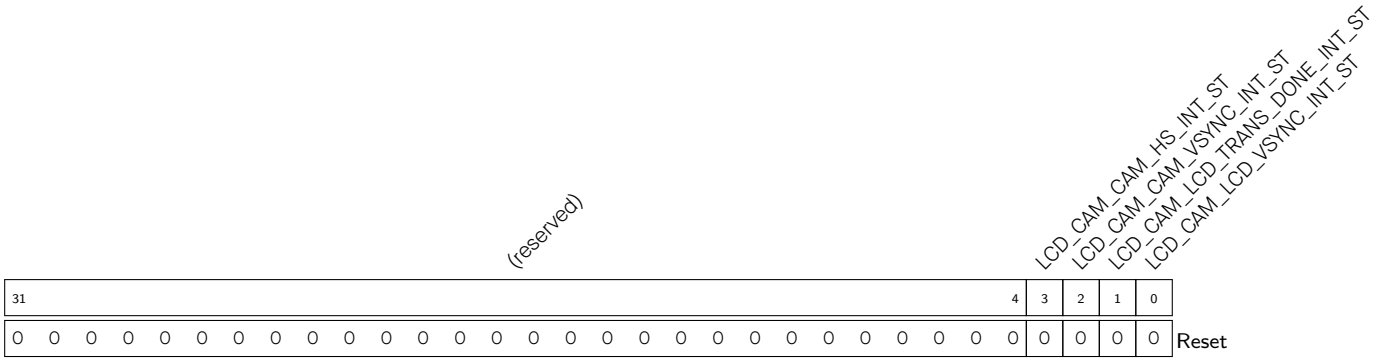
LCD_CAM_LCD_VSYNC_INT_RAW The raw interrupt status of LCD_CAM_LCD_VSYNC_INT. (RO)

LCD_CAM_LCD_TRANS_DONE_INT_RAW The raw interrupt status of LCD_CAM_LCD_TRANS_DONE_INT. (RO)

LCD_CAM_CAM_VSYNC_INT_RAW The raw interrupt status of LCD_CAM_CAM_VSYNC_INT. (RO)

LCD_CAM_CAM_HS_INT_RAW The raw interrupt status of LCD_CAM_CAM_HS_INT. (RO)

Register 40.17. LCD_CAM_LC_DMA_INT_ST_REG (0x006C)



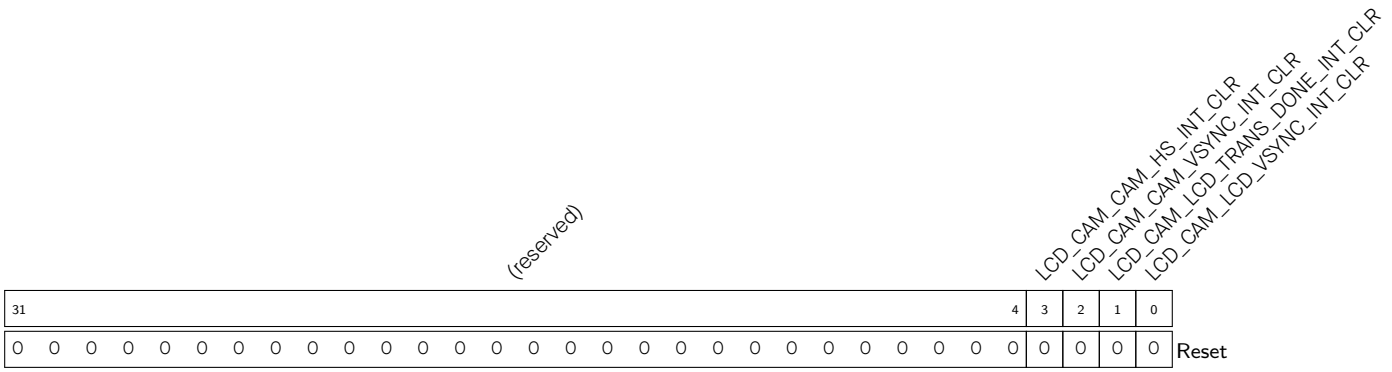
LCD_CAM_LCD_VSYNC_INT_ST The masked interrupt status of LCD_CAM_LCD_VSYNC_INT. (RO)

LCD_CAM_LCD_TRANS_DONE_INT_ST The masked interrupt status of LCD_CAM_LCD_TRANS_DONE_INT. (RO)

LCD_CAM_CAM_VSYNC_INT_ST The masked interrupt status of LCD_CAM_CAM_VSYNC_INT. (RO)

LCD_CAM_CAM_HS_INT_ST The masked interrupt status of LCD_CAM_CAM_HS_INT. (RO)

Register 40.18. LCD_CAM_LC_DMA_INT_CLR_REG (0x0070)



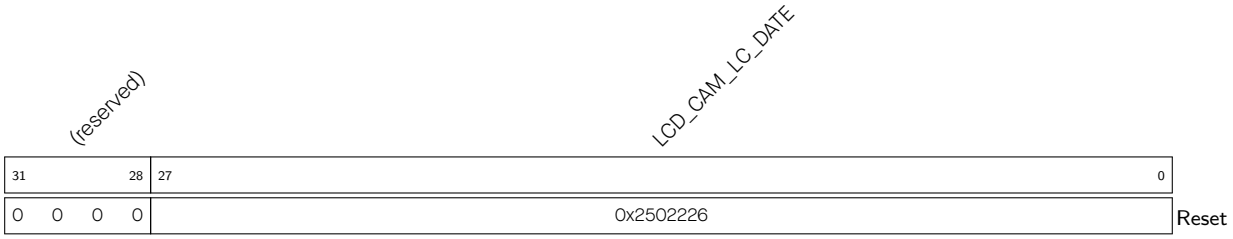
LCD_CAM_LCD_VSYNC_INT_CLR Write 1 to clear LCD_CAM_LCD_VSYNC_INT. (WO)

LCD_CAM_LCD_TRANS_DONE_INT_CLR Write 1 to clear LCD_CAM_LCD_TRANS_DONE_INT. (WO)

LCD_CAM_CAM_VSYNC_INT_CLR Write 1 to clear LCD_CAM_CAM_VSYNC_INT. (WO)

LCD_CAM_CAM_HS_INT_CLR Write 1 to clear LCD_CAM_CAM_HS_INT. (WO)

Register 40.19. LCD_CAM_LC_REG_DATE_REG (0x00FC)



LCD_CAM_LC_DATE Version control register. (R/W)

Chapter 41

H264 Encoder

41.1 Overview

The H264 encoder is based on the H264 standard. It is used for real-time compression of video sequences, which can significantly reduce the total amount of data with minimal loss of video quality.

41.2 Terminology

bitstream	A sequence of bits that forms the representation of one or more coded video sequences.
profile	A subset of the entire bitstream syntax that is specified by the H264 standard.
level	Specified within each profile. A specified set of constraints imposed on values of the syntax elements in the bitstream.
MB	The full name is macroblock. It is a 16 × 16 block of luma samples and two corresponding blocks of chroma samples of a picture that has three sample arrays.
slice	An integer number of MBs.
I frame	Intra frame.
P frame	Inter frame.
I slice	Slice in I frame.
P slice	Slice in P frame.
P-skip MBs	MBs in P frame that skip the encoding.
CAVLC	Context-based adaptive variable length coding.
MV	Motion vector.
GOP	Group of pictures. In a GOP, the first frame is an I frame, and subsequent frames are P frames.
QP	Quantization parameter.
ROI	Region of interest.
AXI	Advanced extensible interface.
TX	Transmit.
RX	Receive.
DB	Deblocking filter.
NUM_H_MB	Number of horizontal MBs in a picture.

41.3 Features

The H264 encoder supports the following features:

- Baseline profile and level 4.1.
- Encoded video color space: 8-bit YUV420.
- Original video color space: RGB888/RGB565/8-bit YUV444/8-bit YUV422/8-bit YUV420(hardware can convert it to 8-bit YUV420 for encoding).
- Progressive video with a maximum resolution of 1920 × 1088 @ 30 fps.
- I and P frames.
- Two working modes: GOP mode and dual stream mode. In dual stream mode, the total bandwidth of the two video sequences to be encoded should not exceed 1920 × 1088 @ 30 fps.
- 4 × 4 and 16 × 16 intra frame luma MB segmentation.
- All nine prediction modes for intra 4 × 4 luma MBs, and all four prediction modes for intra 16 × 16 luma MBs.
- All four prediction modes for intra chroma MBs.
- All partition size for inter luma MBs: 4 × 4, 4 × 8, 8 × 4, 8 × 8, 8 × 16, 16 × 8, 16 × 16.
- One-half and one-quarter pixel precision motion estimation.
- Horizontal motion search range [-29.75, +16.75] and vertical motion search range [-13.75, +13.75].
- Deblocking filter.
- CAVLC.
- P-skip MBs.
- I MB in P slices.
- Decimation operation of luma and chroma component quantization results.
- Fixed QP and MB level rate control.
- MV merge function. If the MV of MB is greater than 0, it can be output to memory.
- ROI function, with up to eight rectangular ROI configurable at any position (allowing overlap, fixed priority). Each ROI can be configured with a constant QP or QP offset, and non-ROI can be configured with a QP offset.
- Debug function: the original video pixel data, the MV merge result data, the bitstream data, the deblocking filter intermediate data and the deblocking filter result data can be replaced by configuration register value.

41.4 Architecture

The architecture of H264 is shown in Figure 41.4-1.

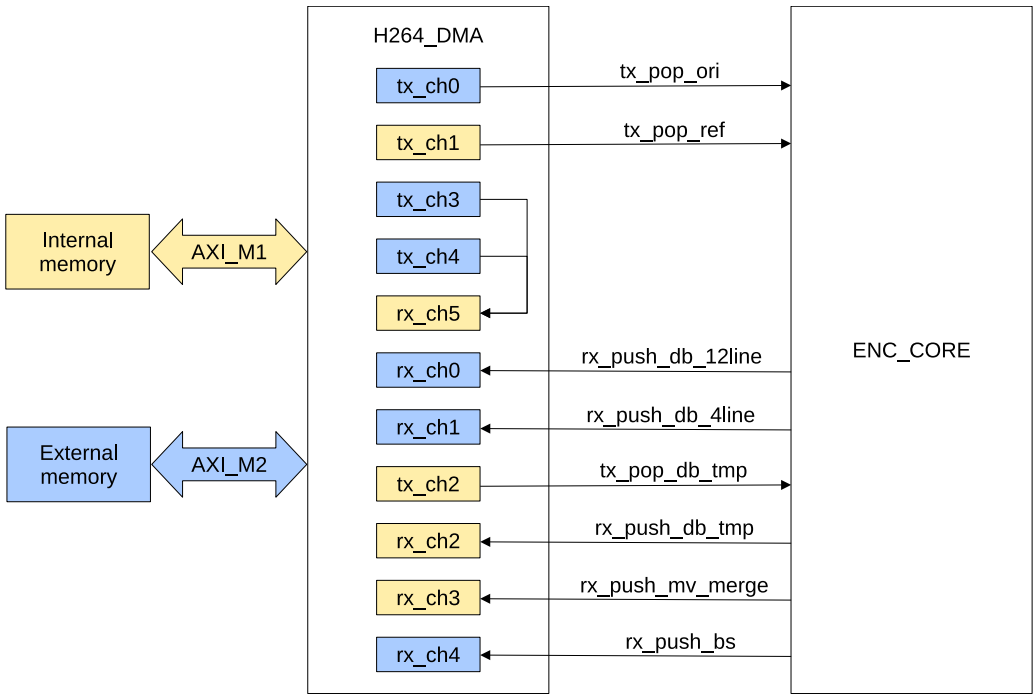


Figure 41.4-1. H264 Video Encoder Architecture

The H264 encoder can be divided into two parts, H264 dedicated DMA (H264_DMA) and encoder algorithm core (ENC_CORE). H264_DMA is used for data transfer, and is connected to the internal/external memory through two sets of AXI buses. Meanwhile, it is connected to the ENC_CORE through multiple DMA sending/receiving channels (H264_DMA has five sending channels (TX channel *n*) and six receiving channels (RX channel *n*)). TX channel 3, TX channel 4, and RX channel 5 are the internal channels of H264_DMA, which are not connected to the ENC_CORE. The ENC_CORE is used to compress video sequences.

The function of each H264_DMA sending and receiving channel is as follows:

- **TX channel 0** sends the original picture to ENC_CORE.
- **TX channel 1** sends the reference picture to ENC_CORE.
- **TX channel 2** sends the deblocking filter intermediate data to ENC_CORE.
- Sending channel **TX channel 3** and TX channel 4 send the reference picture (i.e., the previous deblocking filtered picture) to receiving channel **RX channel 5**.
- **RX channel 0** and **RX channel 1** send the result of the deblocking filter (i.e., reference picture) to external memory.
- **RX channel 2** sends the deblocking filter intermediate data to internal memory.
- **RX channel 3** sends the MV merge result to internal memory.
- **RX channel 4** sends the encoded bitstream to external memory.
- **RX channel 5** sends the reference picture (i.e., the previous deblocking filtered picture) to internal memory.

From the provided description, the H264_DMA channels can be divided into two types:

- Reading or writing internal memory (through AXI_M1 bus): TX channel 1/2, RX channel 2/3/5

- Reading or writing external memory (through AXI_M2 bus): TX channel 0/3/4, RX channel 0/1/4

41.5 Functional Description

41.5.1 Encoder Algorithm Core (ENC_CORE)

The function of the ENC_CORE is to compress and encode the video sequences according to the software configuration.

41.5.1.1 Architecture

The architecture of the encoder algorithm core is shown in Figure 41.5-1 *ENC_CORE Architecture*.

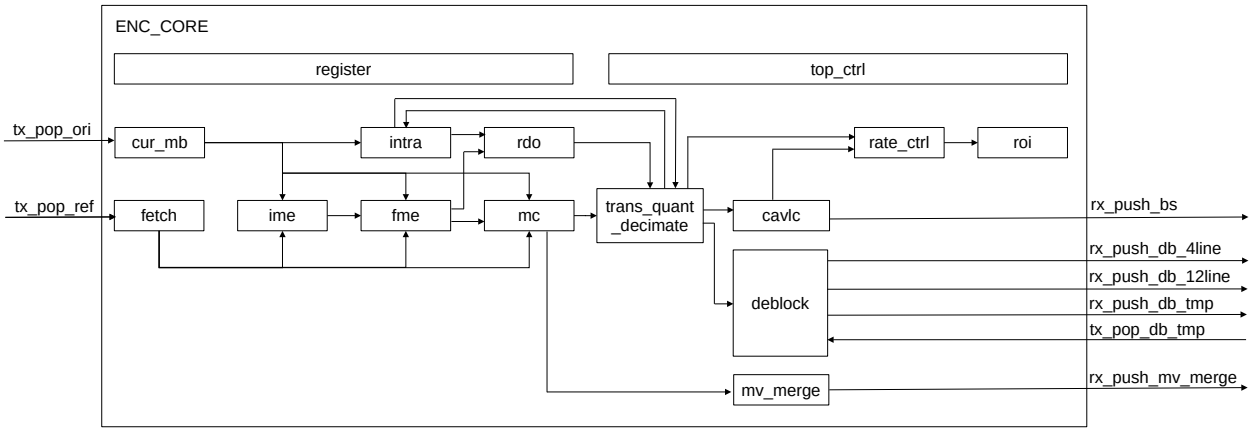


Figure 41.5-1. ENC_CORE Architecture

The functions of each module are as follows:

- register: Implements register configuration and selection.
- top_ctrl: Realizes the control and scheduling of each module of the encoder algorithm core.
- cur_mb: Reads the original picture MB from the TX channel 0 (tx_pop_ori) of H264_DMA, converts its color space to 8-bit YUV420, and distributes it to the intra, ime, fme, and mc modules.
- intra: Uses the reconstruction result calculated by the trans_quant_decimate module to perform intra-frame prediction of the MB, sends the intra-frame prediction and residual results to the trans_quant_decimate module, and sends the intra-frame prediction cost to the rdo module.
- fetch: Reads reference picture MBs within the search range from the TX channel 1 channel (tx_pop_ref) of H264_DMA, and distributes them to the ime, fme, and mc modules.
- ime: Receives the original picture MB from the cur_mb module, receives the reference picture MB from the fetch module, performs integer motion estimation within the search range, and sends the integer motion estimation result to the fme module.
- fme: Receives the original picture MB from the cur_mb module, receives the reference picture MB from the fetch module, receives the integer motion estimation result from the ime module, and performs

fractional motion estimation on the basis of the integer motion estimation. The fractional motion estimation result is sent to the mc module, and the inter prediction cost is sent to the rdo module.

- mc: Receives the original picture MB from the cur_mb module, receives the reference picture MB from the fetch module, receives the fractional motion estimation result from the fme module, performs motion compensation according to the fractional motion estimation result. The inter-frame prediction and the sum of the residual results are sent to the trans_quant_decimate module, and the motion vector is sent to the mv_merge module.
- rdo: Receives the intra-frame prediction cost from the intra module, receives the inter-frame prediction cost from the fme module, selects the final type of the current MB, implements the P slice to support the I MB function, and sends the final MB type to trans_quant_decimate module.
- trans_quant_decimate: Receives the final MB type from the rdo module, receives the intra prediction and residual results from the intra module, receives the inter prediction and residual results from the mc module. Then, it performs DCT, quantization, inverse quantization, and inverse DCT, and quantization result decimation, sends the residual information to the rate_ctrl module, sends the reduced quantization result to the cavlc module, and sends the reconstruction result to the deblock module.
- cavlc: Receives the quantization result from the trans_quant_decimate module and the prediction information of the current MB, performs context adaptive variable length coding (CAVLC), and writes the coded stream to the RX channel 4 of H264_DMA.
- deblock: Receives reconstruction result from the trans_quant_decimate module and the prediction information of the current MB, performs deblocking filtering, and writes the deblocking filter result (i.e., the reference picture) to the RX channel 0 (rx_push_db_12line) and RX channel 1 (rx_push_db_4line). This module reads and writes the deblocking filter intermediate data through the TX channel 2 channel (tx_pop_db_tmp) and RX channel 2 channel (rx_push_db_tmp) of H264_DMA.
- mv_merge: Receives the motion vector from the mc module, performs MV merging, and writes the merging result (MB motion information) to the RX channel 3 channel (rx_push_mv_merge) of H264_DMA.
- rate_ctrl: Receives the residual information from the trans_quant_decimate module, receives the encoding information from the cavlc module, receives the frame-level information configured by the register module, performs MB level rate control, and converts the calculated quantization parameter (QP) to the roi module.
- roi: Receives QPs from the rate_ctrl module, receives the ROI information configured by the register module, adjusts the QPs, and uses the adjusted QPs to process the current MB.

41.5.1.2 Original Video Color Space Convert

Because the original video color space can be RGB888/RGB565/8-bit YUV444/8-bit YUV422/8-bit YUV420 and the ENC_CORE can only encode 8-bit YUV420 video, the ENC_CORE should convert it to 8-bit YUV420.

The convert steps are as follows:

- convert to RGB888 by padding zeros at lower bits(only for RGB565 original video, for RGB888/8-bit YUV444/8-bit YUV422/8-bit YUV420 original video, skip this step):

RGB888_R={RGB565_R,3-bit 0}

RGB888_G={RGB565_G,2-bit 0}

RGB888_B={RGB565_B,3-bit 0}

- convert to 8-bit YUV444 by the equation below (only for RGB565/RGB888 original video, for 8-bit YUV444/8-bit YUV422/8-bit YUV420 original video, skip this step):

YUV444_Y=(RGB888_R × 4899 + RGB888_G × 9617 + RGB888_B × 1868)»14

YUV444_U=(128 × 16384 + RGB888_B × 8192 - RGB888_R × 2764 - RGB888_G × 5428)»14

YUV444_V=(128 × 16384 + RGB888_R × 8192 - RGB888_G × 6860 - RGB888_B × 1332)»14

- subsample 8-bit YUV444/8-bit YUV422 to 8-bit YUV420(only for RGB565/RGB888/8-bit YUV444/8-bit YUV422 original video, for 8-bit YUV420 original video, skip this step)

The original video color space is configured by register H264_*x*_ORI_COLOR_SPACE. *x* can be A or B, indicating the register corresponding to video sequence A or B.

41.5.1.3 Quantization Result Decimation

The ENC_CORE calculates the score of each MB based on the magnitude of its final quantization result, and then compares the score with the configurable threshold register. If it is below the threshold, the quantization result will be set to 0. Otherwise, the quantization result will remain unchanged, so as to further improve the compression ratio. For each MB, its luma and chroma component carry out the above process independently. Different types of MBs have different comparison methods, as follows:

- luma component I MB with 16 × 16 partition:

$$\text{I_score_mb_cmp} + \text{H264_}x\text{_I16×16_DECI_SCORE_OFFSET} < \text{H264_}x\text{_L_DECI_SCORE}$$

  Among them, I_score_mb_cmp is the score of the luma component of the current MB calculated by ENC_CORE. H264_*x*_I16×16_DECI_SCORE_OFFSET is the configurable offset register. H264_*x*_L_DECI_SCORE is the configurable threshold register. *x* can be A or B, indicating the register corresponding to video sequence A or B. When the above requirement is met, the quantization result of the current luma component MB will be decimated to 0.

- luma component I MB with 4 × 4 partition: Current luma component MB quantization result will not be decimated to 0.

- chroma component I MB:

$$\text{c_score_mb_cmp} + \text{H264_}x\text{_I_chroma_DECI_SCORE_OFFSET} < \text{H264_}x\text{_C_DECI_SCORE}$$

  Among them, c_score_mb_cmp is the score of current chroma component MB calculated by ENC_CORE. H264_*x*_I_chroma_DECI_SCORE_OFFSET is the configurable offset register. H264_*x*_C_DECI_SCORE is the configurable threshold register. *x* can be A or B, indicating the register corresponding to video sequence A or B. When the above requirement is met, the quantization result of the current chroma component MB will be decimated to 0.

- luma component P MB:

$$\text{I_score_mb_cmp} + \text{H264_}x\text{_P16×16_DECI_SCORE_OFFSET} < \text{H264_}x\text{_L_DECI_SCORE}$$

  Among them, I_score_mb_cmp is the score of current luma component MB calculated by ENC_CORE. H264_*x*_P16×16_DECI_SCORE_OFFSET is the configurable offset register. H264_*x*_L_DECI_SCORE is the configurable threshold register. *x* can be A or B, indicating the register corresponding to video

sequence A or B. When the above requirement is met, the quantization result of the current luma component MB will be decimated to 0.

- chroma component P MB:

  $$c_score_mb_cmp + H264_x_P_chroma_DECI_SCORE_OFFSET < H264_x_C_DECI_SCORE$$

  Among them, c_score_mb_cmp is the score of current chroma component MB calculated by ENC_CORE. H264_*x*_P_chroma_DECI_SCORE_OFFSET is the configurable offset register. H264_*x*_C_DECI_SCORE is the configurable threshold register. *x* can be A or B, indicating the register corresponding to video sequence A or B. When the above requirement is met, the quantization result of the current chroma component MB will be decimated to 0.

41.5.1.4 Motion Vector (MV) Merging

The ENC_CORE merges the MVs of the luma MB according to the configuration of the registers related to MV merging, and writes it out to the RX channel 3 channel (rx_push_mv_merge) of H264_DMA, and finally stores it in the internal memory for the upper layer application use, such as motion detection.

The registers related to MV merging are as follows:

- H264_*x*_MV_MERGE_EN: This register controls the enablement of MV merging. Setting this register to 1 means enabling the MV merging function of the corresponding video sequence. Setting this register to 0 means turning off the MV merging function of the corresponding video sequence. *x* can be A or B, indicating the register corresponding to video sequence A or B. Even if the MV merging function is enabled, the I MB will not output any data to the memory because the I MB has no MV.
- H264_MV_MERGE_TYPE: This register controls the type of MV merging. Configuring this register to 0 means only retaining the MV of the MB whose final type is P MB and the partition of luma component is 16 × 16. Configuring this register to 1 means only retaining the MV with the minimum modulus of the MB whose final type is P MB. Configuring this register to 2 means only retaining the MV with the maximum modulus of the MB whose final type is P MB. 3 is an invalid value. When the register H264_*x*_MV_MERGE_EN is set to 1, the configuration of this register will take effect.
- H264_INT_MV_OUT_EN: This register controls the type of merged MV ultimately stored in memory. Setting this register to 0 means that if the reserved merged MV is not 0, it will be stored in memory. Otherwise, it will be discarded. Setting this register to 1 means that if the integer part of the reserved merged MV is not 0, it will be stored in memory, otherwise discarded. When the register H264_*x*_MV_MERGE_EN is set to 1, the configuration of this register will take effect.
- H264_MB_VALID_NUM: This register represents the number of MV merging results stored in the memory in the current picture. One MB corresponds to one MV merging result. When the register H264_*x*_MV_MERGE_EN is set to 1, the value of this register is meaningful, otherwise the value of this register is meaningless and remains 0.

41.5.1.5 MB Level Rate Control

The ENC_CORE calculates the QP adjustment amount of each MB based on the configuration of the rate control related registers, the number of encoding bits of the current MB, and the mean absolute value of the residual (MAD). Then it uses the frame-level QP configured in the register and the QP adjustment amount to calculate the QP of each MB, thereby adjusting the number of coded bits.

The MB level rate control related registers are as follows:

- H264_*x*_MB_RATE_CTRL_EN: This register controls the enablement of MB level rate control. Setting this register to 1 means enabling the MB level rate control function of the corresponding video sequence. Setting this register to 0 means turning off the MB level rate control function of the corresponding video sequence. *x* can be A or B, indicating the register corresponding to video sequence A or B. Since the I frame uses a constant QP encoding method, the QP of each macro block of the I frame will not change even if the MB level rate control function is enabled.
- H264_*x*_RATE_CTRL_U: The value of this register is configured by the parameters calculated by the frame-level rate control in the software. ENC_CORE needs this value to calculate the QP adjustment amount of each macro block. *x* can be A or B, indicating the register corresponding to video sequence A or B. The calculation method of the configuration value of this register belongs to the category of the frame-level rate control algorithm in the software, which is beyond the scope of this chapter and will not be described.
- H264_*x*_MAD_FRAME_PRED: This register is the predicted value of frame-level MAD, which is configured by the frame-level rate control in the software. ENC_CORE needs this value to calculate the QP adjustment amount of each MB, *x* can be A or B, indicating the register corresponding to video sequence A or B. The calculation method of the configuration value of this register belongs to the category of the frame-level rate control algorithm in the software, which is beyond the scope of this chapter and will not be described.
- H264_*x*_QP: This register is the frame-level QP value, and the value range is 10 to 51. It is configured by the frame-level rate control in the software. ENC_CORE will adjust the QP value of each MB based on it, so as to realize the MB level rate control, *x* can be A or B, indicating the register corresponding to video sequence A or B. The calculation method of the configuration value of this register belongs to the category of the frame-level rate control algorithm in the software, which is beyond the scope of this chapter and will not be described.
- H264_*x*_QP_MAX: This register is used to limit the maximum value of the QP finally calculated by the MB level rate control, and the value range is from 10 to 51. If the QP value finally calculated by the MB level rate control is greater than the configuration value of this register, it will be limited to the configured value of this register, otherwise the original value is retained. *x* can be A or B, indicating the register corresponding to video sequence A or B.
- H264_*x*_QP_MIN: This register is used to limit the minimum value of the QP finally calculated by the MB level rate control, and the value range is from 10 to 51. If the QP value finally calculated by the MB level rate control is smaller than the configuration value of this register, it will be limited to the configured value of this register, otherwise it remains the original value, *x* can be A or B, indicating the register corresponding to video sequence A or B.
- H264_FRAME_MAD_SUM: This register represents the cumulative sum of the MADs of all MBs in the current frame. The software needs to read the value of this register after the encoding of a frame of video is completed for calculation of frame-level rate control. The frame-level rate control algorithm in the software is beyond the scope of this chapter and will not be described.
- H264_FRAME_ENC_BITS: This register represents the sum of the number of encoding bits of all MBs in the current frame. The software needs to read the value of this register after completely encoding a frame of video to perform frame-level rate control. The frame-level rate control algorithm in the software is beyond the scope of this chapter and will not be described.
- H264_FRAME_QP_SUM: This register represents the sum of QP of all MBs in the current frame. The

software needs to read the value of this register after completely encoding of a frame of video to perform frame-level rate control. The frame-level rate control algorithm in the software is beyond the scope of this chapter and will not be described.

41.5.1.6 Region of Interest (ROI)

ENC_CORE can have up to eight overlapping fixed-priority rectangular ROI in a frame of video according to the configuration of the relevant registers. The QP (calculated by MB level rate control) is adjusted, so that each ROI has different encoded picture quality.

The registers related to the ROI are as follows:

- H264_*x*_ROI_EN: This register is used to control the switch of the ROI function. Set it to 1 to enable the ROI function, and set it to 0 to disable the ROI function. *x* can be A or B, indicating the register corresponding to video sequence A or B.
- H264_*x*_ROI_MODE: This register is used to specify the working mode of ROI. If it is configured as 0, ROI will work in fixed QP mode. If it is configured as 1, ROI will work in QP offset mode. *x* can be A or B, indicating the register corresponding to video sequence A or B.
- H264_*x*_ROI_REGION*n*_REG: This register is used to configure the enable and position information of each rectangular ROI. *x* can be A or B, indicating the register corresponding to video sequence A or B. *n* can be 0 to 7, indicating 8 ROI. From ROI 0 to ROI 7, the priority decreases in order. When there are overlapping areas, the configuration of the ROI with the highest priority takes effect. This register contains the following fields:
  - H264_*x*_ROI_REGION*n*_EN: Set to 1 to enable ROI *n*, set it to 0 to close the ROI *n*. When the register H264_*x*_ROI_EN set to 1, this register can control the ROI *n* switch, otherwise, this register has no effect.
  - H264_*x*_ROI_REGION*n*_X: Configure the horizontal starting MB coordinates of the ROI *n* in a frame of video.
  - H264_*x*_ROI_REGION*n*_Y: Configure the vertical starting MB coordinates of the ROI *n* in a frame of video.
  - H264_*x*_ROI_REGION*n*_X_LEN: Configure the number of horizontal MBs in the ROI *n* in a frame of video.
  - H264_*x*_ROI_REGION*n*_Y_LEN: Configure the number of vertical MBs in the ROI *n* in a frame of video.
- H264_*x*_ROI_REGION*n*_QP: This register is used to configure the QP adjustment amount of the MBs in each rectangular ROI. *x* can be A or B, indicating the register corresponding to video sequence A or B. *n* can be 0 to 7, representing 8 ROI. The configuration method of this register is related to the configuration value of the H264_*x*_ROI_MODE register:
  - When H264_*x*_ROI_MODE is set to 0: The ROI mode is fixed QP mode. The value of the H264_*x*_ROI_REGION*n*_QP register directly specifies the QP value for MBs in the corresponding ROI and ranges from 10 to 51.
  - When H264_*x*_ROI_MODE is set to 1: It indicates that the ROI mode is QP offset mode. At this time, bit 0 to bit 5 of the H264_*x*_ROI_REGION*n*_QP register indicates the QP offset, and the value range

is 0 to 51. Bit 6 indicates the sign of the offset, where 0 indicates a positive offset, and 1 indicates a negative offset.

- H264_*x*_NO_ROI_REGION_QP: This register is used to configure the QP offset of the MB in the non-ROI. Bit 0 to bit 5 represent the QP offset, and the value range is 0 to 51. Bit 6 represents the sign of the offset, where 0 represents a positive offset, and 1 indicates a negative offset. *x* can be A or B, indicating the register corresponding to video sequence A or B. When the ROI function is disabled, that is, when the H264_*x*_ROI_EN register is set to 0, this register will not take effect, and the QP of the MB in the non-ROI will not be changed.

Please note that the adjusted QP value of any ROI should not be less than 10.

Regardless of whether the MB is in the ROI or non-ROI, its final adjusted QP value will be limited to 10 to 51 to ensure that it does not exceed the value range in the H264 standard.

41.5.1.7 Data Replace For Debug Function

For debug which part of data is wrong in ENC_CORE processing flow, the original video pixel data, the MV merge result data, the bitstream data, the deblocking filter intermediate data and the deblocking filter result data can be replaced by configuration register value(H264_*x*_DBG_REPLACE_ORI_DATA, H264_*x*_DBG_REPLACE_MV_MERGE_DATA, H264_*x*_DBG_REPLACE_WR_BS_DATA, H264_*x*_DBG_REPLACE_WR_DB_TEMP_DATA, H264_*x*_DBG_REPLACE_RD_DB_TEMP_DATA, H264_*x*_DBG_REPLACE_WR_DB_DATA, H264_*x*_DBG_REPLACE_REF_DATA).

- original video pixel data replace: Set register H264_*x*_DBG_REPLACE_ORI_DATA_EN to 1 to enable this function. The original video pixel data which is read from H264_DMA will be replaced by H264_*x*_DBG_REPLACE_ORI_DATA.
- MV merge result data replace: Set register H264_*x*_DBG_REPLACE_MV_MERGE_DATA_EN to 1 to enable this function. The MV merge data which is written to H264_DMA will be replaced by H264_*x*_DBG_REPLACE_MV_MERGE_DATA.
- bitstream data replace: Set register H264_*x*_DBG_REPLACE_WR_BS_DATA_EN to 1 to enable this function. The bitstream data which is written to H264_DMA will be replaced by H264_*x*_DBG_REPLACE_WR_BS_DATA.
- deblocking filter intermediate data replace: Set register H264_*x*_DBG_REPLACE_WR_DB_TEMP_DATA_EN to 1 to enable the function that the deblocking filter intermediate data which is written to H264_DMA will be replaced by H264_*x*_DBG_REPLACE_WR_DB_TEMP_DATA. Set register H264_*x*_DBG_REPLACE_RD_DB_TEMP_DATA_EN to 1 to enable the function that the deblocking filter intermediate data which is read from H264_DMA will be replaced by H264_*x*_DBG_REPLACE_RD_DB_TEMP_DATA.
- deblocking filter result data replace: Set register H264_*x*_DBG_REPLACE_WR_DB_DATA_EN to 1 to enable the function that the deblocking filter result data which is written to H264_DMA will be replaced by H264_*x*_DBG_REPLACE_WR_DB_DATA. Set register H264_*x*_DBG_REPLACE_REF_DATA_EN to 1 to enable the function that the reference picture pixel data(i.e. deblocking filter result data) which is read from H264_DMA will be replaced by H264_*x*_DBG_REPLACE_REF_DATA.

41.5.2 H264 Dedicated DMA

The H264 dedicated DMA controls the data transfers between encoder algorithm core and memory based on the software configuration.

41.5.2.1 Architecture

H264_DMA architecture is shown in the figure 41.5-2:

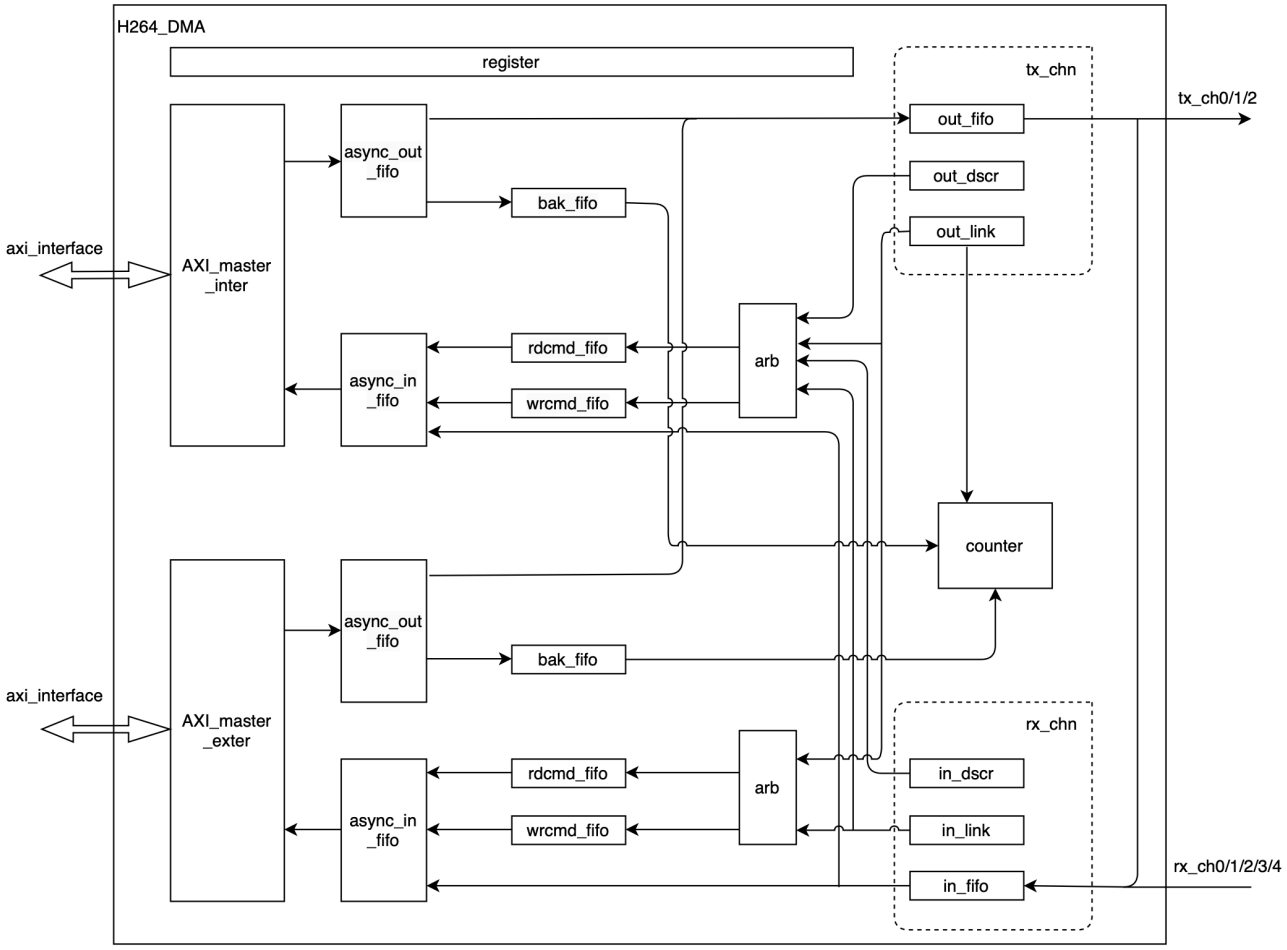


Figure 41.5-2. H264_DMA Architecture

The functions of each module are:

- register: This module implements register configuration and selection.
- in_dscr: This module initiates an AXI read request to internal memory, obtains the RX channel's linked list descriptor, and sends the descriptor to the in_link module of the corresponding RX channel.
- in_link: This module receives the descriptor of the corresponding channel in_dscr module and splits it into multiple AXI write data requests to guide the corresponding in_fifo module to write data to internal memory or external memory. RX channel 0/1/4 is designed for writing data to external memory, while RX channel 2/3/5 is for writing data to internal memory.
- out_dscr: This module initiates an AXI read internal memory request, obtains the TX channel linked list descriptor, and sends the descriptor to the out_link module of corresponding TX channel.

- out_link: This module receives the descriptor of the corresponding channel out_dscr module and divides it into multiple AXI read data requests to obtain data from internal memory or external memory. TX channel 0/3/4 reads data from external memory and TX channel 1/2 reads data from internal memory.
- out_fifo: This module receives the data returned by the AXI read request and waits for h264 to read, or RX channel 5 to read. TX channel 0/1/2 is for h264 to read, and TX channel 3/4 is for RX channel 5 to read.
- arb: This module receives read/write requests for external or internal memory, and arbitrates the read/write requests. The requests are sent to the rdcmd_fifo and wrcmd_fifo modules of external memory, and the rdcmd_fifo and wrcmd_fifo modules of internal memory respectively.
- wrcmd_fifo: This module caches the AXI write request arbitrated by the arb module, and cuts the write request and data into multiple 8-byte data chunks before sending them to the async_in_fifo module.
- rdcmd_fifo: This module caches the AXI read request arbitrated by the arb module and sends the read request to the async_in_fifo module.
- bak_fifo: This module caches the read/write request information arbitrated by the arb module, including channel id, AXI_id, read/write request data length, read/write request source and read request starting address. The module finds the information corresponding to the request based on the AXI_id returned by the async_out_fifo module, and sends the completion information of the request and the read data in the async_out_fifo module to the corresponding channel.
- async_in_fifo: This module is used to receive AXI requests from the wrcmd_fifo module and rdcmd_fifo module, and synchronize the request from H264_DMA core clock domain to the AXI clock domain, waiting for the AXI master to read.
- async_out_fifo: This module is used to receive the read data returned by the AXI_master_inter/AXI_master_exter module and the completion of the read/write request, and synchronize it from the AXI clock domain to the H264_DMA core clock domain.
- AXI_master_inter/AXI_master_exter: This module reads the read/write request in the async_in_fifo module and converts it into an AXI bus protocol format output. The module receives the read data and read/write completion returned by the AXI slave and converts it into an internal private interface. The resulting format is written to the async_out_fifo module. The AXI_master_inter module is used to access the internal memory space, and the AXI_master_exter module is used to access the external memory space.

41.5.2.2 Linked List Descriptor

H264_DMA linked list descriptor is shown in the figure 41.5-3:

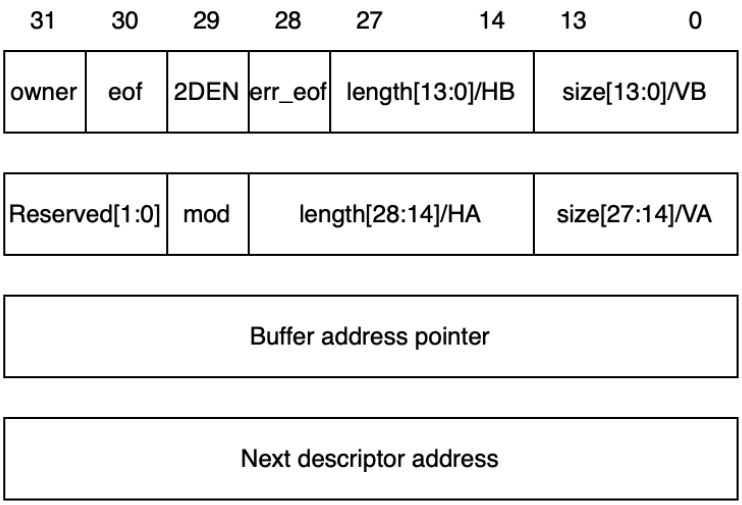


Figure 41.5-3. H264_DMA Linked List Descriptor

- owner: Indicates the operators allowed by the buffer corresponding to the current descriptor.
  - 0: Allow the operator to be CPU.
  - 1: The operator is allowed to be H264_DMA. H264_DMA will clear this bit after using the descriptor.
  - For transmit descriptors, since a descriptor will not be written back by default, the default owner will not be cleared. Configure H264_DMA_OUT_AUTO_WRBACK to enable writeback.
- suc_eof: end flag.
  - 0: The current descriptor is not the last descriptor.
  - 1: The current descriptor is the last descriptor.
- mod: Data block read mode.
  - 0: H264_DMA read/write a picture block with a horizontal width of hb and a vertical height of vb only once.
  - 1: H264_DMA read/write multiple times the picture block with a horizontal width of hb and a vertical height of vb until the picture data with a horizontal width of HA and a vertical height of VA is read/written. Note: The unit of hb/vb/HA/VA is pixel.
  - In 1D mode, this field must be 0.
- err_eof: Receive end error flag.
  - For the receive descriptor, the hardware will set this bit to 1 after receiving a packet and detecting a received data error.
  - For transmit descriptors, this bit is fixed to 0.
- 2DEN: 2D function enable signal.
  - 0: The function of H264_DMA is the same as the general DMA function. At this time, it can be used to transfer continuous data segments.
  - 1: The H264_DMA function is enabled and data is transferred according to the configured picture blocks.

- length[13:0]/HB:
  - When 2DEN = 0, this field represents the length of the data being transferred (least significant 14 bits). For data reception, this field is filled in after the hardware receives a frame of data. For data transmission, this field indicates the length of the data to be sent. The unit is byte.
  - When 2DEN = 1, this field represents the horizontal width of the picture block (HB). The unit is pixel.
- size[13:0]/VB:
  - When 2DEN = 0, this field represents the size of the memory space allocated for transferring data (least significant 14 bits). The unit is byte.
  - When 2DEN = 1, this field represents the vertical height of the picture block (VB). The unit is pixel.
- length[28:14]/HA:
  - WHen 2DEN = 0, this field represents the length of the transferred data (most significant 14 bits, only bit14-bit27 is used). For data reception, this field is filled after the hardware receives a frame of data. For data transmission, this field indicates the length of the data to be sent. The unit is byte.
  - When 2DEN = 1, this field represents the horizontal width of the picture corresponding to the descriptor (HA). The unit is pixel.
- size[27:14]/VA:
  - When 2DEN = 0, this field represents the size of the memory space for transferring data (most significant 14 bits). The unit is byte.
  - When 2DEN = 1, this field represents the vertical height of the picture corresponding to the descriptor (VA). The unit is pixel.
- Buffer address pointer:
  - When 2DEN = 0, it is the buffer address pointer.
  - When 2DEN = 1, it is the address pointer of the origin coordinates of the HA × VA picture block in the buffer.
- Next descriptor address: Next descriptor address pointer.

41.5.2.3 Transfer Initialization

H264_DMA will read the linked list from the internal memory to the local according to the configuration of the registers related to each channel linked list descriptor, so that it can be used by the corresponding channel to transfer data, thereby start the corresponding channel.

The registers associated with each linked list descriptor are as follows:

- H264_DMA_OUTLINK/INLINK_START_CH*x*: This register controls whether the corresponding channel of H264_DMA starts to obtain descriptors. Setting this register to 1 means that the corresponding channel starts to obtain the descriptor. After obtaining the descriptor, the hardware automatically sets this register to 0, and then automatically obtain subsequent descriptors. *x* can be 0, 1, 2, 3, and 4, indicating the registers corresponding to channels 0, 1, 2, 3, and 4.
- H264_DMA_OUTLINK/INLINK_ADDR_CH*x*: This register represents the memory address where the first descriptor is stored, and the address requires 8-byte alignment.

41.5.2.4 Transfer Reset

In order to ensure the safe reset of H264_DMA and prevent channel from being stuck, the reset of H264_DMA channel must be carried out according to the following procedure:

1. Write 1 to register H264_DMA_OUT/IN_CMD_DISABLE_CH*x* of the corresponding channel.
2. Wait for H264_DMA_OUT/IN_RESET_AVAIL_CH*x* of the corresponding channel to be set.
3. Write 1 to register H264_DMA_OUT/IN_RST_CH*x* of the corresponding channel.
4. Write 0 to register H264_DMA_OUT/IN_RST_CH*x* of the corresponding channel.

The registers related to channel reset are as follows:

- H264_DMA_OUT/IN_CMD_DISABLE_CH*x*: This register controls whether the corresponding channel of H264_DMA stops initiating read/write requests to AXI. Setting this register to 1 means that the corresponding channel stops initiating read/write requests to AXI. Setting this register to 0 means that the corresponding channel can initiate read/write requests to AXI. *x* can be 0, 1, and 2 in TX channel, indicating the registers corresponding to channels 0, 1, and 2. In RX channel, they can be 0, 1, 2, 3, 4, and 5, indicating the registers corresponding to channels 0, 1, 2, 3, 4, and 5.
- H264_DMA_OUT/IN_RESET_AVAIL_CH*x*: This register indicates whether the channel corresponding to H264_DMA can be safely reset. If this register is 1, it means that the corresponding channel can be safely reset. If this register is 0, it means that the corresponding channel cannot be safely reset.
- H264_DMA_OUT/IN_RST_CH*x*: This register controls whether the corresponding channel of H264_DMA performs soft reset. Setting this register to 1 means that the corresponding channel performs a soft reset. Setting this register to 0 means that the corresponding channel does not perform a soft reset.

Notes:

- Since TX channel 3, TX channel 4, and RX channel 5 are bound, these three channels only have one set of H264_DMA_OUT/IN_CMD_DISABLE_CH5, H264_DMA_OUT/IN_RESET_AVAIL_CH5, and H264_DMA_OUT/IN_RST_CH5 registers. When resetting these three channels, you only need to operate this group of registers to realize the safe reset of the three H264_DMA channels.
- In order to ensure the completion of data transmission, before resetting RX channel 0, RX channel 1, RX channel 3, and RX channel 4, you need to check whether the corresponding channel linked list descriptor is written back successfully. Only after the linked list descriptor of the channel is written back successfully, which means all data transmission of the channel is completed, can the channel be reset. If the MV merging function is not enabled or the number of video frames in a GOP is 1, which means the register H264_GOP_NUM is configured as 1, there is no need to check whether the linked list descriptor of RX channel 3 is written back successfully.

41.5.2.5 Data Replace For Debug Function

H264_DMA module supports replacing the original data of each channel with debug data.

- Register H264_DMA_CH_DBG_EN_REG controls whether to enable debug data replace for each channel
- After debug-data replacement is enabled, the original data of each channel is replaced with the data configured by H264_DMA_IN/OUT_CHx_DBG_DATA_H/L_REG. Except for TX channel 0, all channels are

replaced directly with the 64-bit debug data. According to the pbyte configuration, the replacement for TX channel 0 is as follows:

- H264_DMA_ORI_PBYTE=0, bit 3:0 of H264_OUT_CH0_DBG_DATA_L is valid
- H264_DMA_ORI_PBYTE=1, bit 7:0 of H264_OUT_CH0_DBG_DATA_L is valid
- H264_DMA_ORI_PBYTE=2, bit 23:0 of H264_OUT_CH0_DBG_DATA_L is valid
- H264_DMA_ORI_PBYTE=3, bit 15:0 of H264_OUT_CH0_DBG_DATA_L is valid
- H264_DMA_ORI_PBYTE=4, bit 23:0 of H264_OUT_CH0_DBG_DATA_L is valid

41.5.2.6 Channel Configuration

Before the software starts H264_DMA, it needs to configure H264_DMA first. The configuration consists of register configuration and linked list descriptor configuration.

Linked list descriptor configuration: The channels that need to be configured with descriptors are TX channel 0 ~ TX channel 4 and RX channel 0 ~ RX channel 4. The linked list descriptors are all circular linked lists.

1. Configure the channel of reading the original picture (TX channel 0):
   - mode: 2D, mode1.
   - length/size: HA is the number of horizontal pixels of the picture. VA is the number of vertical pixels of the picture, HB is 16 × 4, and VB is 16.
2. Configure the channel of reading reference picture (TX channel 1):
   - mode: 2D, mode1.
   - length/size: HA is NUM_H_MB × 16 × 16, VA is 3, HB is 16 × 16, and VB is 3.
3. Configure the channel of reading DB intermediate data (TX channel 2):
   - mode: 1D.
   - length/size: The read space size is NUM_H_MB × (16 × 4 + 8 × 4 × 2).
4. Configure the channel of reading first 12 lines of DB data (TX channel 3):
   - mode: 1D.
   - length/size: The read space size is (NUM_H_MB × (number of vertical macro block lines in the picture - 1) × (16 × 12 + 8 × 4 × 2)) + (NUM_H_MB × 16 × 16 × 1.5).
5. Configure the channel of reading last 4 lines DB data (TX channel 4):
   - mode: 1D.
   - length/size: The read space size is (NUM_H_MB × (number of vertical MB lines in the picture - 1) × (16 × 4 + 8 × 4 × 2)).
6. Configure the channel of writing first 12 lines of DB data (RX channel 0):
   - mode: 1D.
   - length/size: The size of the writing space is the same as the size of TX channel 3.
7. Configure the channel of writing last 4 lines of DB data (RX channel 1):

- mode: 1D.
- length/size: The size of the writing space is the same as the size of TX channel 4.

8. Configure the channel of writing DB intermediate data (RX channel 2):
   - mode: 1D.
   - length/size: The size of the writing space is the same as the size of TX channel 2.
9. Configure the channel of writing MV merge data (RX channel 3):
   - mode: 1D.
   - length/size: The writing space size is NUM_H_MB × the number of vertical MB lines × 4.
10. Configure the channel of writing bit stream data (RX channel 4):
    - mode: 1D.
    - length/size: The size of the writing space is the number of horizontal pixels of the picture × the number of vertical pixel lines × 1.5.

Register configuration:

- H264_DMA_ORI_PBYTE: This register configures the bytes number of each pixel in original picture.
- H264_DMA_OUT/IN_MEM_BURST_LENGTH_CH*x*: This register controls the AXI burst length of the corresponding channel.
  - 0: 8 bytes. 1: 16 bytes. 2: 32 bytes. 3: 64 bytes. 4: 128 bytes.
  - In TX channel, *x* can be 0 ~ 4. In RX channel, *x* can be 0 ~ 5.
  - The burst length of TX channel 2 and RX channel 2 must be configured the same.
- H264_DMA_OUT_REORDER_EN_CH0: This register controls the TX channel 0 reorder enable. When the linked list descriptor configuration HB is not 16 in TX channel 0, this register must be configured as 1.
- H264_DMA_BLOCK_START_ADDR_CH5: This register controls the data starting address of the RX channel 5. The register configuration must be consistent with the data buffer address in the linked list descriptor of the TX channel 1.
- H264_DMA_BLOCK_ROW_LENGTH_4LINE_CH5: This register specifies the number of bytes contained in a row block of 4 lines. The data size is (NUM_H_MB × (16 × 4 + 8 × 4 × 2)).
- H264_DMA_BLOCK_ROW_LENGTH_12LINE_CH5: This register specifies the number of bytes contained in a row block of 12 lines. The data size is (NUM_H_MB × (16 × 12 + 8 × 4 × 2)).
- H264_DMA_BLOCK_LENGTH_4LINE_CH5: This register specifies the number of bytes contained in a block of 4 lines. The data size is (16 × 4 + 8 × 4 × 2).
- H264_DMA_BLOCK_LENGTH_12LINE_CH5: This register specifies the number of bytes contained in a block of 12 lines. The data size is (16 × 12 + 8 × 4 × 2).

41.6 Interrupts

ESP32-P4's H264 Encoder can generate the following interrupt signals, which are then sent to the Interrupt Matrix.

- H264_REG_INT
- H264_DMA_IN_CH0_INT
- H264_DMA_IN_CH1_INT
- H264_DMA_IN_CH2_INT
- H264_DMA_IN_CH3_INT
- H264_DMA_IN_CH4_INT
- H264_DMA_IN_CH5_INT
- H264_DMA_OUT_CH0_INT
- H264_DMA_OUT_CH1_INT
- H264_DMA_OUT_CH2_INT
- H264_DMA_OUT_CH3_INT
- H264_DMA_OUT_CH4_INT

There are several internal interrupt sources from H264 Encoder that can generate the above interrupt signals.

Interrupt signal H264_REG_INT can be generated by the following internal interrupt sources:

- H264_BS_BUFFER_OVERFLOW_INT: When the number of encoded bytes of a 4 × 4 sub-block of ENC_CORE exceeds H264_BS_BUFFER_THRESHOLD, the interrupt is triggered.
- H264_DMA_MOVE_2MB_LINE_DONE_INT: When H264_DMA completes moving the reference pictures of two MB rows from external memory to internal memory, the interrupt is triggered.
- H264_FRAME_DONE_INT: When ENC_CORE finishes compressing and encoding a frame of video, the interrupt is triggered.
- H264_REC_READY_INT: When ENC_CORE writes the deblocking filter result (i.e., the reference picture) of H264_*x*_REC_READY_TRIGGER_MB_LINES lines of MBs to the external memory, the interrupt is triggered. *x* can be A or B, indicating the register corresponds to video sequence A or B.
- H264_DB_TMP_READY_INT: When ENC_CORE writes the deblocking filter intermediate data of H264_*x*_DB_TMP_READY_TRIGGER_MB_NUM MBs to the internal memory, the interrupt is triggered. *x* can be A or B indicates the register corresponds to video sequence A or B.

The above interrupt sources can only be triggered when the interrupt enable register *XXX*_ENA is set to 1. Write 1 to *XXX*_CLR will clear the interrupt (*XXX* is the interrupt source name).

The H264_DMA_IN_CH*n*_INT (*n* is 0 ~ 5) signals are generated by the following interrupt sources:

- H264_DMA_IN_DONE_CH*n*_INT: Triggered when all data corresponding to a receive descriptor has been received via RX channel *n* (*n* is 0 ~ 5).
- H264_DMA_IN_SUC_EOF_CH*n*_INT: Triggered when the suc_eof bit in a receive descriptor is 1 and the data corresponding to this receive descriptor has been received (i.e. when the data frame or packet corresponding to an inlink has been received) via RX channel *n* (*n* is 0 ~ 5).
- H264_DMA_IN_ERR_EOF_CH*n*_INT: Triggered when an error is detected in the data frame or packet received via RX channel *n* (*n* is 0 ~ 4).

- H264_DMA_IN_DSCR_EMPTY_CH*n*_INT: Triggered when the size of the buffer pointed by receive descriptors is smaller than the length of data to be received via RX channel *n* (*n* is 0 ~ 4).
- H264_DMA_IN_DSCR_ERR_CH*n*_INT: Triggered when a receive descriptor on RX channel *n* (*n* is 0 ~ 4) fails any of the two descriptor checks.
- H264_DMA_INFIFO_OVF_L1_CH*n*_INT: Triggered when the L1 RX FIFO of RX channel *n* (*n* is 0 ~ 5) overflows.
- H264_DMA_INFIFO_OVF_L2_CH*n*_INT: Triggered when the L2 RX FIFO of RX channel *n* (*n* is 0 ~ 4) overflows.
- H264_DMA_INFIFO_UDF_L1_CH*n*_INT: Triggered when the L1 RX FIFO of RX channel *n* (*n* is 0 ~ 5) underflows.
- H264_DMA_INFIFO_UDF_L2_CH*n*_INT: Triggered when the L2 RX FIFO of RX channel *n* (*n* is 0 ~ 4) underflows.
- H264_DMA_IN_DSCR_TASK_OVF_CH*n*_INT: Triggered when descriptor ready task FIFO of RX channel *n* (*n* is 0 ~ 4) overflows.
- H264_DMA_FETCH_MB_COL_CNT_OVF_CH*n*_INT: Triggered when fetch MB columns cnt of RX channel *n* (*n* is 5) overflows.

The H264_DMA_OUT_CH*n*_INT (*n* is 0 ~ 4) signals are generated by the following interrupt sources:

- H264_DMA_OUT_DONE_CH*n*_INT: Triggered when all data corresponding to a transmit descriptor has been sent via TX channel *n*.
- H264_DMA_OUT_EOF_CH*n*_INT: Triggered when the suc_eof bit in a transmit descriptor is 1 and data corresponding to this descriptor has been sent via TX channel *n*. If H264_DMA_OUT_EOF_MODE_CH*n* is 0, this interrupt will be triggered when the last byte of data corresponding to this descriptor enters H264_DMA's TX channel; if H264_DMA_OUT_EOF_MODE_CH*n* is 1, this interrupt is triggered when the last byte of data is taken from H264_DMA's TX channel.
- H264_DMA_OUT_DSCR_ERR_CH*n*_INT: Triggered when a transmit descriptor on TX channel *n* fails any of the two descriptor checks.
- H264_DMA_OUT_TOTAL_EOF_CH*n*_INT: Triggered when all data corresponding to a linked list (including multiple descriptors) has been sent via TX channel *n*.
- H264_DMA_OUTFIFO_OVF_L1_CH*n*_INT: Triggered when the L1 TX FIFO of H264_DMA overflows.
- H264_DMA_OUTFIFO_OVF_L2_CH*n*_INT: Triggered when the L2 TX FIFO of H264_DMA overflows.
- H264_DMA_OUTFIFO_UDF_L1_CH*n*_INT: Triggered when the L1 TX FIFO of H264_DMA underflows.
- H264_DMA_OUTFIFO_UDF_L2_CH*n*_INT: Triggered when the L2 TX FIFO of H264_DMA underflows.
- H264_DMA_OUT_DSCR_TASK_OVF_CH*n*_INT: Triggered when descriptor ready task FIFO of TX channel *n* overflows.

41.7 Programming Procedures

The H264 encoder can work in two modes. In GOP mode, it periodically inserts an I frame into a specified number (configurable) of P frame intervals. In dual stream mode, it takes turns to process two video

sequences, and the hardware automatically switches the ENC_CORE configurations.

41.7.1 GOP Mode

In GOP mode, a single video sequence A is encoded. Take a GOP containing three pictures as an example, the encoding sequence of pictures is shown in Figure 41.7-1:

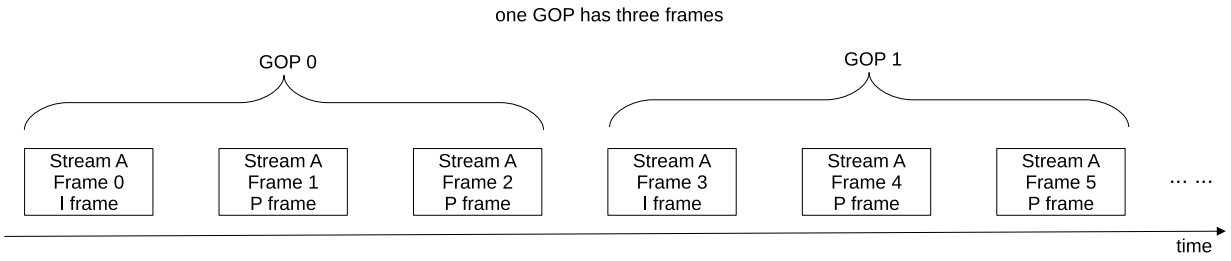


Figure 41.7-1. Coding Order of Pictures in GOP Mode

In GOP mode, the register with the prefix H264_A_ is valid, and the register with the prefix H264_B_ is invalid.

The GOP mode configuration process is as follows:

1. Refer to the description in chapter 41.5.2.6 to configure each H264_DMA channel:
   (a) Configure H264_DMA TX channel 0.
   (b) Configure H264_DMA TX channel 1.
   (c) Configure H264_DMA TX channel 2.
   (d) Configure H264_DMA TX channel 3 and TX channel 4.
   (e) Configure H264_DMA RX channel 0 and RX channel 1.
   (f) Configure H264_DMA RX channel 2.
   (g) Configure H264_DMA RX channel 3.
   (h) Configure H264_DMA RX channel 4.
   (i) Configure H264_DMA RX channel 5.
2. Configure ENC_CORE according to the required functions:
   (a) Configure GOP mode: Set register H264_FRAME_MODE to 0 and register H264_DUAL_STREAM_MODE to 0.
   (b) Configure the number of pictures in a GOP: Configure the register H264_GOP_NUM as n, indicating that a GOP contains n pictures. When the register is configured as 0, the first frame of a GOP is an I frame, and the subsequent frames of a GOP are always P frame until the encoder is reset.
   (c) Configure the MB resolution of a picture: configure the register H264_A_SYS_TOTAL_MB_X as the number of MBs in the horizontal direction of the picture, and configure the register H264_A_SYS_TOTAL_MB_Y as the number of MBs in the vertical direction of the picture. The number of MBs in the horizontal direction is equal to the width of the video divided by 16 and

rounded up. And the number of MBs in the vertical direction is equal to the height of the video divided by 16 and rounded up.

(d) Configure the initial QP value of a picture: configure the register H264_A_QP as the initial QP value, ranging from 10 to 51.

(e) Configure deblocking filter: Set register H264_A_BYPASS_DB_FILTER to 0 to enable the deblocking filter, set register H264_A_BYPASS_DB_FILTER to 1 to disable the deblocking filter.

(f) Configure slice header information: According to the information required to be included in the slice header in the H264 standard, configure the relevant registers H264_SLICE_REMAIN_BIT, H264_SLICE_REMAIN_BITLENGTH, H264_SLICE_BYTE_LENGTH, H264_SLICE_BYTE_LSB, and H264_SLICE_BYTE_MSB. For example, if the slice header is a variable with 41 bits and the most significant bit is the first bit of the slice header, these registers should be set in the following ways.

- H264_SLICE_BYTE_MSB: From bit 41 to bit 10 of the slice header
- H264_SLICE_BYTE_LSB: From bit 9 to bit 2 of the slice header
- H264_SLICE_REMAIN_BIT: Bit 1 of the slice header
- H264_SLICE_BYTE_LENGTH: 5
- H264_SLICE_REMAIN_BITLENGTH: 1

(g) Configure original video color space: Configure the register H264_A_ORI_COLOR_SPACE as the corresponding original video color space.

(h) To use the quantization result decimation function, configure its related registers referring to the description in section 41.5.1.3.

(i) To use the MB level rate control function, configure its related registers referring to the description in section 41.5.1.5.

(j) To use the ROI function, configure its related registers referring to the description in chapter 41.5.1.6.

(k) To use the MV merge function, configure its related registers referring to the description in section 41.5.1.4.

(l) To use the debug function, configure its related registers referring to the description in section 41.5.1.7.

3. Perform the corresponding operations based on the type of video frame to be encoded (In a GOP, the first frame is an I frame, and subsequent frames are P frames):

- The video frame to be encoded is an I frame:

(a) Start H264_DMA TX channel 0: Refer to chapter 41.5.2.3 for the configuration.

(b) Start H264_DMA RX channel 0 and RX channel 1: Refer to chapter 41.5.2.3 for the configuration.

(c) Start H264_DMA RX channel 2: Refer to chapter 41.5.2.3 for the configuration.

(d) To use the MV merge function, start H264_DMA RX channel 3. Refer to chapter 41.5.2.3 for the configuration.

(e) Start H264_DMA RX channel 4: Refer to chapter 41.5.2.3 for the configuration.

(f) Clear H264 interrupts and enable interrupts.

(g) Start the encoding of a picture: Write 1 to register H264_FRAME_START.

(h) Wait for the H264_DB_TMP_READY_INT interrupt. When the interrupt is triggered, start H264_DMA TX channel 2. Refer to chapter 41.5.2.3 for the configuration.

(i) Wait for the H264_REC_READY_INT interrupt. When the interrupt is triggered, start H264_DMA RX channel 5, H264_DMA TX channel 3, and TX channel 4. Refer to section 41.5.2.3 for the configuration.

(j) Start H264_DMA to move the reference picture from external memory to internal memory: Write 1 to register H264_DMA_MOVE_START.

(k) Wait for the H264_DMA_MOVE_2MB_LINE_DONE_INT interrupt. When the interrupt is triggered, start H264_DMA TX channel 1. Refer to chapter 41.5.2.3 for the configuration.

(l) Wait for the H264_FRAME_DONE_INT interrupt. When the interrupt is triggered, it indicates that the picture encoding is completed.

(m) Reset H264_DMA TX channel 2 and RX channel 2: Refer to chapter 41.5.2.4 for the configuration.

(n) Reset the read and write time counter of the deblocking filter intermediate data in H264_DMA: Set register H264_DMA_RX_CH2_INTER_COUNTER_RST to 1, then set to 0.

- The video frame to be encoded is P frame:

  (a) Clear H264 interrupts and enable interrupts.

  (b) Start H264_DMA RX channel 2: Refer to chapter 41.5.2.3 for the configuration.

  (c) Start encoding a picture: Write 1 to register H264_FRAME_START.

  (d) Wait for the H264_DB_TMP_READY_INT interrupt. When the interrupt is triggered, start H264_DMA TX channel 2. Refer to chapter 41.5.2.3 for the configuration.

  (e) Wait for the H264_FRAME_DONE_INT interrupt. When the interrupt is triggered, it indicates that the picture encoding is completed.

  (f) Reset H264_DMA TX channel 2 and RX channel 2: Refer to chapter 41.5.2.4 for the configuration.

  (g) Reset the read and write time counter of deblocking filter intermediate data in H264_DMA: Set register H264_DMA_RX_CH2_INTER_COUNTER_RST to 1, then set to 0.

- After any frame ends, the following registers can be reconfigured:
  - Registers related to the quantization result decimation function. Refer to chapter 41.5.1.3 for details.
  - Registers related to the MB level rate control function. Refer to chapter 41.5.1.5 for details.
  - Deblocking filter enable register: H264_A_BYPASS_DB_FILTER.
  - ROI function related registers. Refer to chapter 41.5.1.6 for details.
  - Registers related to slice header information: H264_SLICE_REMAIN_BIT, H264_SLICE_REMAIN_BITLENGTH, H264_SLICE_BYTE_LENGTH, H264_SLICE_BYTE_LSB, H264_SLICE_BYTE_MSB.

- Registers related to the MV merge function. Refer to chapter 41.5.1.4 for details.
- Registers related to the debug function. Refer to chapter 41.5.1.7 for details.

4. Repeat step 3 until all video frames in a GOP are encoded.
5. Make sure the data is written completely. Refer to chapter 41.5.2.4 for the method.
6. Reset each sending and receiving channel of H264_DMA referring to chapter 41.5.2.4.
7. Reset the read and write times counter of deblocking filter result data, and the read and write times counter of reference picture data in H264_DMA. To be specific, set register H264_DMA_RX_CH0_EXTER_COUNTER_RST to 1, then set it to 0; Set register H264_DMA_RX_CH1_EXTER_COUNTER_RST to 1, then set it to 0; Set register H264_DMA_RX_CH5_INTER_COUNTER_RST to 1, then set it to 0.
8. Reconfigure from step 2 and start encoding the next GOP.

41.7.2 Dual Stream Mode

In the dual stream mode, two video sequences A and B are encoded. The encoding sequence of pictures is shown in Figure 41.7-2 (a GOP contains three pictures).

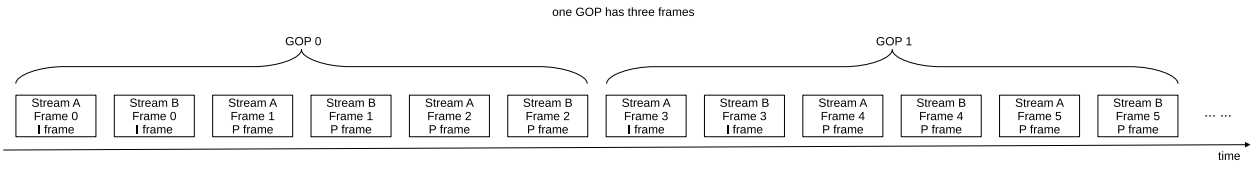


Figure 41.7-2. Coding Order of Pictures in Dual Stream Mode

In dual stream mode, the frame order of H264 processing is fixed as A1, B1, A2, B2, A3, B3... When processing frames in video sequence A, registers with the prefix H264_A_ will be automatically applied. When processing frames in video sequence B, registers with the prefix H264_B_ will be automatically applied. The registers with other prefixes are valid for both video sequence frames.

The configuration process of dual stream mode is as follows:

1. Configure ENC_CORE according to the required function:
   (a) Configure dual stream mode: Set register H264_FRAME_MODE to 1, and set register H264_DUAL_STREAM_MODE to 1.
   (b) Configure GOP (the GOP configuration of video sequence A and B must be the same): Configure the register H264_GOP_NUM as n, indicating that a GOP contains n pictures. When the register is configured as 0, the first frame of a GOP is an I frame, and the subsequent frames of a GOP are always P frames until the encoder is reset.
   (c) Configure the MB resolution of a picture: Configure the register H264_A_SYS_TOTAL_MB_X as the number of MBs in the horizontal direction of the video sequence A, and configure the register H264_A_SYS_TOTAL_MB_Y as the number of MBs in the vertical direction of the video sequence A. Configure the register H264_B_SYS_TOTAL_MB_X as the number of MBs in the horizontal direction of the video sequence B, and configure the register H264_B_SYS_TOTAL_MB_Y as the number of MBs in the vertical direction of the video sequence B. The number of MBs in the horizontal

direction is equal to the width of the video divided by 16 and rounded up. And the number of MBs in the vertical direction is equal to the height of the video divided by 16 and rounded up.

(d) Configure the initial QP of a picture: Configure the register H264_A_QP as the initial QP of video sequence A, ranging from 10 to 51. Configure the register H264_B_QP as the initial QP of video sequence B, ranging from 10 to 51.

(e) Configure the deblocking filter: Set the register H264_A_BYPASS_DB_FILTER to 0 to enable the deblocking filter of video sequence A. Set the register H264_A_BYPASS_DB_FILTER to 1 to disable the deblocking filter of video sequence A. Set the register H264_B_BYPASS_DB_FILTER to 0 to enable the deblocking filter of video sequence B. Set the register H264_B_BYPASS_DB_FILTER to 1 to disable the deblocking filter of video sequence B.

(f) Configure slice header information: According to the information required to be included in the slice header in the H264 standard, configure the relevant registers H264_SLICE_REMAIN_BIT, H264_SLICE_REMAIN_BITLENGTH, H264_SLICE_BYTE_LENGTH, H264_SLICE_BYTE_LSB, and H264_SLICE_BYTE_MSB. For example, if the slice header is a variable with 41 bits and the most significant bit is the first bit of the slice header, these registers should be set in the following ways.

- H264_SLICE_BYTE_MSB: From bit 41 to bit 10 of the slice header
- H264_SLICE_BYTE_LSB: From bit 9 to bit 2 of the slice header
- H264_SLICE_REMAIN_BIT: Bit 1 of the slice header
- H264_SLICE_BYTE_LENGTH: 5
- H264_SLICE_REMAIN_BITLENGTH: 1

(g) Configure original video color space: Configure the register H264_A_ORI_COLOR_SPACE as the corresponding original video color space of video sequence A. Configure the register H264_B_ORI_COLOR_SPACE as the corresponding original video color space of video sequence B.

(h) To use the quantization result decimation function, configure its related registers referring to section 41.5.1.3.

(i) To use the MB level rate control function, configure its related registers referring to chapter 41.5.1.5.

(j) To use the ROI function, configure its related registers referring to chapter 41.5.1.6.

(k) To use the MV merge function, configure its related registers referring to chapter 41.5.1.4.

(l) To use the debug function, configure its related registers referring to the description in section 41.5.1.7.

2. According to whether the frame to be encoded belongs to video sequence A or B, use the information corresponding to the video sequence and configure each H264_DMA channel referring to section 41.5.2.6:

(a) Configure H264_DMA TX channel 0.

(b) Configure H264_DMA TX channel 1.

(c) Configure H264_DMA TX channel 2.

(d) Configure H264_DMA TX channel 3 and TX channel 4.

(e) Configure H264_DMA RX channel 0 and RX channel 1.

(f) Configure H264_DMA RX channel 2.

(g) Configure H264_DMA RX channel 3.

(h) Configure H264_DMA RX channel 4.

(i) Configure H264_DMA RX channel 5.

3. Perform the corresponding operations based on the type of video frame to be encoded. Video sequence A and B are encoded alternately, i.e., first a picture in video sequence A is encoded, and then a picture in video sequence B, etc. In a GOP, the first frame is an I frame, and the subsequent frames are P frames for any video sequence.

   - The video frame to be encoded is an I frame:

     (a) Start H264_DMA TX channel 0: Refer to section 41.5.2.3 for the configuration.

     (b) Start H264_DMA RX channel 0 and RX channel 1: Refer to section 41.5.2.3 for the configuration.

     (c) Start H264_DMA RX channel 2: Refer to section 41.5.2.3 for the configuration.

     (d) Start H264_DMA RX channel 4: Refer to section 41.5.2.3 for the configuration.

     (e) Clear H264 interrupts and enable interrupts.

     (f) Start encoding a picture: Write 1 to register H264_FRAME_START.

     (g) Wait for the H264_DB_TMP_READY_INT interrupt. When the interrupt is triggered, start H264_DMA TX channel 2. Refer to section 41.5.2.3 for the configuration.

     (h) Wait for the H264_FRAME_DONE_INT interrupt. When the interrupt is triggered, it means that the picture encoding is completed.

     (i) Make sure the data is written completely. Refer to section 41.5.2.4 for the method.

     (j) Reset each sending and receiving channel of H264_DMA referring to section 41.5.2.4.

     (k) Reset the read and write counter of the deblocking filter intermediate data, and the read and write counter of the reference picture data in H264_DMA. To be specific, set register H264_DMA_RX_CH2_INTER_COUNTER_RST to 1, and then set it to 0. Then set H264_DMA_RX_CH5_INTER_COUNTER_RST to 1 and then to 0.

   - The video frame to be encoded is P frame:

     (a) Clear H264 interrupts and enable interrupts.

     (b) Start H264_DMA TX channel 0: Refer to section 41.5.2.3 for the configuration.

     (c) Start H264_DMA RX channel 0 and RX channel 1: Refer to section 41.5.2.3 for the configuration.

     (d) Start H264_DMA RX channel 2: Refer to section 41.5.2.3 for the configuration.

     (e) Start H264_DMA RX channel 4: Refer to section 41.5.2.3 for the configuration.

     (f) Start H264_DMA RX channel 5: Refer to section 41.5.2.3 for the configuration.

     (g) Start H264_DMA TX channel 3 and TX channel 4: Refer to section 41.5.2.3 for the configuration.

     (h) To use the MV merge function, start H264_DMA RX channel 3: Refer to section 41.5.2.3 for the configuration.

(i) Start H264_DMA to move the reference picture from external memory to internal memory: Write 1 to register H264_DMA_MOVE_START.

(j) Wait for H264_DMA_MOVE_2MB_LINE_DONE_INT interrupt. When the interrupt is triggered, start H264_DMA TX channel 1. Refer to section 41.5.2.3 for the configuration.

(k) Start encoding of a picture: Write 1 to register H264_FRAME_START.

(l) Wait for the H264_DB_TMP_READY_INT interrupt. When the interrupt is triggered, start H264_DMA TX channel 2. Refer to section 41.5.2.3 for the configuration.

(m) Wait for the H264_FRAME_DONE_INT interrupt. When the interrupt is triggered, it indicates that the picture encoding is completed.

(n) Make sure the data is written completely referring to section 41.5.2.4.

(o) Reset each sending and receiving channel of H264_DMA referring to section 41.5.2.4.

(p) Reset the read and write counter of the deblocking filter intermediate data and the read and write counter of the reference picture data in H264_DMA. To be specific, set the register H264_DMA_RX_CH2_INTER_COUNTER_RST to 1, and then set it to 0; Set the register H264_DMA_RX_CH5_INTER_COUNTER_RST to 1 and then set it to 0.

- After any frame ends, the following registers corresponding to the currently encoded video sequence can be reconfigured:
  - Registers related to the quantization result decimation function, refer to the description in chapter 41.5.1.3.
  - Registers related to the MB level rate control function, refer to the description in chapter 41.5.1.5.
  - Deblocking filter enable register: H264_A_BYPASS_DB_FILTER.
  - ROI function related registers, refer to the description in chapter 41.5.1.6.
  - Registers related to the slice header information: H264_SLICE_REMAIN_BIT, H264_SLICE_REMAIN_BITLENGTH, H264_SLICE_BYTE_LENGTH, H264_SLICE_BYTE_LSB, and H264_SLICE_BYTE_MSB.
  - Registers related to the MV merge function, refer to the description in chapter 41.5.1.4.
  - Registers related to the debug function. Refer to chapter 41.5.1.7 for details.

4. Repeat steps 2 and 3 until one GOP of each of the two video sequences is encoded.
5. Reset the read and write times counter of the deblocking filter result data in H264_DMA. To be specific, set the register H264_DMA_RX_CHO_EXTER_COUNTER_RST to 1, then set to 0. Set the register H264_DMA_RX_CH1_EXTER_COUNTER_RST to 1, then set to 0.
6. Reconfigure from step 1 to start the encoding of the next GOP.

41.7.3 Soft Reset

No matter what working mode H264 is in, reset it according to the following process can stop encoding at any time.

1. Reset ENC_CORE: Write 1 to register H264_SYS_RST_PULSE.

2. Reset each sending and receiving channel of H264_DMA referring to section 41.5.2.4.
3. Reset each read and write time counter in H264_DMA. To be specific, set register H264_DMA_RX_CH0_EXTER_COUNTER_RST to 1, then set it to 0; Set register H264_DMA_RX_CH1_EXTER_COUNTER_RST to 1, then set it to 0; Set the register H264_DMA_RX_CH2_INTER_COUNTER_RST to 1, and then set it to 0; Set the register H264_DMA_RX_CH5_INTER_COUNTER_RST to 1, and then set it to 0.
4. H264 returns to the initial state and can be reconfigured as needed.

41.8 Register Summary

The addresses in this section are relative to **H264 Encoder** base address and **H264 DMA** base address. These base addresses are provided in Table 9.3-2 *Module/Peripheral Address Mapping* in Chapter 9 *System and Memory*.

The abbreviations given in Column **Access** are explained in Section *Access Types for Registers*.

41.8.1 H264 Encoder Register Summary

Name	Description	Address	Access
Configuration Registers			
H264_SYS_CTRL_REG	H264 system level control register	0x0000	varies
H264_GOP_CONF_REG	GOP related configuration register	0x0004	R/W
H264_SLICE_HEADER_REMAIN_REG	Frame slice header remain bit register	0x00AC	R/W
H264_SLICE_HEADER_BYTE_LENGTH_REG	Frame slice header byte length register	0x00B0	R/W
H264_BS_THRESHOLD_REG	Bitstream buffer overflow threshold register	0x00B4	R/W
H264_SLICE_HEADER_BYTE0_REG	Frame slice header byte low 32 bit register	0x00B8	R/W
H264_SLICE_HEADER_BYTE1_REG	Frame slice header byte high 32 bit register	0x00BC	R/W
H264_CONF_REG	General configuration register	0x00D0	R/W
H264_MV_MERGE_CONFIG_REG	MV merge configuration register	0x00D4	varies
Video Sequence A Configuration Registers			
H264_A_SYS_MB_RES_REG	Video sequence A horizontal and vertical MB resolution register	0x0008	R/W
H264_A_SYS_CONF_REG	Video sequence A system level configuration register	0x000C	R/W
H264_A_DECI_SCORE_REG	Video sequence A luma and chroma MB decimate score register	0x0010	R/W
H264_A_DECI_SCORE_OFFSET_REG	Video sequence A luma and chroma MB decimate score offset register	0x0014	R/W
H264_A_RC_CONF0_REG	Video sequence A rate control configuration register0	0x0018	R/W
H264_A_RC_CONF1_REG	Video sequence A rate control configuration register1	0x001C	R/W
H264_A_DB_BYPASS_REG	Video sequence A Deblocking bypass register	0x0020	R/W
H264_A_ROI_REGION0_REG	Video sequence A H264 ROI0 range configure register	0x0024	R/W
H264_A_ROI_REGION1_REG	Video sequence A H264 ROI1 range configure register	0x0028	R/W
H264_A_ROI_REGION2_REG	Video sequence A H264 ROI2 range configure register	0x002C	R/W

Name	Description	Address	Access
H264_A_ROI_REGION3_REG	Video sequence A H264 ROI3 range configure register	0x0030	R/W
H264_A_ROI_REGION4_REG	Video sequence A H264 ROI4 range configure register	0x0034	R/W
H264_A_ROI_REGION5_REG	Video sequence A H264 ROI5 range configure register	0x0038	R/W
H264_A_ROI_REGION6_REG	Video sequence A H264 ROI6 range configure register	0x003C	R/W
H264_A_ROI_REGION7_REG	Video sequence A H264 ROI7 range configure register	0x0040	R/W
H264_A_ROI_REGION0_3_QP_REG	Video sequence A H264 ROI0-3 QP register	0x0044	R/W
H264_A_ROI_REGION4_7_QP_REG	Video sequence A H264 ROI4-7 QP register	0x0048	R/W
H264_A_NO_ROI_REGION_QP_OFFSET_REG	Video sequence A H264 no ROI QP register	0x004C	R/W
H264_A_ROI_CONFIG_REG	Video sequence A H264 ROI configure register	0x0050	R/W
Video Sequence B Configuration Registers			
H264_B_SYS_MB_RES_REG	Video sequence B horizontal and vertical MB resolution register	0x0054	R/W
H264_B_SYS_CONF_REG	Video sequence B system level configuration register	0x0058	R/W
H264_B_DECI_SCORE_REG	Video sequence B luma and chroma MB decimate score register	0x005C	R/W
H264_B_DECI_SCORE_OFFSET_REG	Video sequence B luma and chroma MB decimate score offset register	0x0060	R/W
H264_B_RC_CONF0_REG	Video sequence B rate control configuration register0.	0x0064	R/W
H264_B_RC_CONF1_REG	Video sequence B rate control configuration register1.	0x0068	R/W
H264_B_DB_BYPASS_REG	Video sequence B Deblocking bypass register	0x006C	R/W
H264_B_ROI_REGION0_REG	Video sequence B H264 ROI0 range configure register	0x0070	R/W
H264_B_ROI_REGION1_REG	Video sequence B H264 ROI1 range configure register	0x0074	R/W
H264_B_ROI_REGION2_REG	Video sequence B H264 ROI2 range configure register	0x0078	R/W
H264_B_ROI_REGION3_REG	Video sequence B H264 ROI3 range configure register	0x007C	R/W
H264_B_ROI_REGION4_REG	Video sequence B H264 ROI4 range configure register	0x0080	R/W

Name	Description	Address	Access
H264_B_ROI_REGION5_REG	Video sequence B H264 ROI5 range configure register	0x0084	R/W
H264_B_ROI_REGION6_REG	Video sequence B H264 ROI6 range configure register	0x0088	R/W
H264_B_ROI_REGION7_REG	Video sequence B H264 ROI7 range configure register	0x008C	R/W
H264_B_ROI_REGION0_3_QP_REG	Video sequence B H264 ROI0-3 QP register	0x0090	R/W
H264_B_ROI_REGION4_7_QP_REG	Video sequence B H264 ROI4-7 QP register	0x0094	R/W
H264_B_NO_ROI_REGION_QP_OFFSET_REG	Video sequence B H264 no ROI QP register	0x0098	R/W
H264_B_ROI_CONFIG_REG	Video sequence B H264 ROI configure register	0x009C	R/W
H264_A_ORI_CONF_REG	Original picture configuration register of video sequence A	0x00F4	R/W
H264_B_ORI_CONF_REG	Original picture configuration register of video sequence B	0x00F8	R/W
H264_ORI_DEBUG_CONF_REG	Original picture debug configuration register	0x00FC	R/W
H264_MV_MERGE_DEBUG_CONF_REG	MV merge debug configuration register	0x0100	R/W
H264_BS_DEBUG_CONG_REG	Encoded bitstream debug configuration register	0x0104	R/W
H264_DB_WR_TEMP_DEBUG_CONG_REG	Deblocking filter write temporary data debug configuration register	0x0108	R/W
H264_DB_RD_TEMP_DEBUG_CONG_REG	Deblocking filter read temporary data debug configuration register	0x010C	R/W
H264_DB_WR_DEBUG_CONG_REG	Deblocking filter final data debug configuration register	0x0110	R/W
H264_REF_DEBUG_CONG_REG	Reference frame debug configuration register	0x0114	R/W
Status Registers			
H264_RC_STATUS0_REG	Rate control status register0	0x00A0	RO
H264_RC_STATUS1_REG	Rate control status register1	0x00A4	RO
H264_RC_STATUS2_REG	Rate control status register2	0x00A8	RO
H264_SYS_STATUS_REG	System status register	0x00DC	RO
H264_FRAME_CODE_LENGTH_REG	Frame code byte length register	0x00E0	RO
Interrupt Registers			
H264_INT_RAW_REG	Interrupt raw status register	0x00C0	R/WTC/SS
H264_INT_ST_REG	Interrupt masked status register	0x00C4	RO
H264_INT_ENA_REG	Interrupt enable register	0x00C8	R/W
H264_INT_CLR_REG	Interrupt clear register	0x00CC	WT
Version Register			
H264_DATE_REG	Version control register	0x00F0	R/W

41.8.2 H264 DMA Register Summary

Name	Description	Address	Access
Configuration Registers			
TX Configuration Registers			
H264_DMA_OUT_CONF0_CH0_REG	TX CH0 config0 register	0x0000	R/W
H264_DMA_OUT_PUSH_CH0_REG	TX CH0 outfifo push register	0x0018	varies
H264_DMA_OUT_LINK_CONF_CH0_REG	TX CH0 out_link dscr ctrl register	0x001C	varies
H264_DMA_OUT_LINK_ADDR_CH0_REG	TX CH0 out_link dscr addr register	0x0020	R/W
H264_DMA_OUT_ARB_CH0_REG	TX CH0 arb register	0x003C	R/W
H264_DMA_OUT_RO_PD_CONF_CH0_REG	TX CH0 reorder power config register	0x0044	R/W
H264_DMA_OUT_MODE_ENABLE_CH0_REG	TX CH0 mode enable register	0x0050	R/W
H264_DMA_OUT_MODE_YUV_CH0_REG	TX CH0 test mode yuv value register	0x0054	R/W
H264_DMA_OUT_ETM_CONF_CH0_REG	TX CH0 ETM config register	0x0068	R/W
H264_DMA_OUT_CONF0_CH1_REG	TX CH1 config0 register	0x0100	R/W
H264_DMA_OUT_PUSH_CH1_REG	TX CH1 outfifo push register	0x0118	varies
H264_DMA_OUT_LINK_CONF_CH1_REG	TX CH1 out_link dscr ctrl register	0x011C	varies
H264_DMA_OUT_LINK_ADDR_CH1_REG	TX CH1 out_link dscr addr register	0x0120	R/W
H264_DMA_OUT_ARB_CH1_REG	TX CH1 arb register	0x013C	R/W
H264_DMA_OUT_ETM_CONF_CH1_REG	TX CH1 ETM config register	0x0168	R/W
H264_DMA_OUT_CONF0_CH2_REG	TX CH2 config0 register	0x0200	R/W
H264_DMA_OUT_PUSH_CH2_REG	TX CH2 outfifo push register	0x0218	varies
H264_DMA_OUT_LINK_CONF_CH2_REG	TX CH2 out_link dscr ctrl register	0x021C	varies
H264_DMA_OUT_LINK_ADDR_CH2_REG	TX CH2 out_link dscr addr register	0x0220	R/W
H264_DMA_OUT_ARB_CH2_REG	TX CH2 arb register	0x023C	R/W
H264_DMA_OUT_ETM_CONF_CH2_REG	TX CH2 ETM config register	0x0268	R/W
H264_DMA_OUT_CONF0_CH3_REG	TX CH3 config0 register	0x0300	R/W
H264_DMA_OUT_PUSH_CH3_REG	TX CH3 outfifo push register	0x0318	varies
H264_DMA_OUT_LINK_CONF_CH3_REG	TX CH3 out_link dscr ctrl register	0x031C	varies
H264_DMA_OUT_LINK_ADDR_CH3_REG	TX CH3 out_link dscr addr register	0x0320	R/W
H264_DMA_OUT_ARB_CH3_REG	TX CH3 arb register	0x033C	R/W
H264_DMA_OUT_ETM_CONF_CH3_REG	TX CH3 ETM config register	0x0368	R/W
H264_DMA_OUT_CONF0_CH4_REG	TX CH4 config0 register	0x0400	R/W
H264_DMA_OUT_PUSH_CH4_REG	TX CH4 outfifo push register	0x0418	varies
H264_DMA_OUT_LINK_CONF_CH4_REG	TX CH4 out_link dscr ctrl register	0x041C	varies
H264_DMA_OUT_LINK_ADDR_CH4_REG	TX CH4 out_link dscr addr register	0x0420	R/W
H264_DMA_OUT_ARB_CH4_REG	TX CH4 arb register	0x043C	R/W
H264_DMA_OUT_ETM_CONF_CH4_REG	TX CH4 ETM config register	0x0468	R/W
RX Configuration Registers			
H264_DMA_IN_CONF0_CH0_REG	RX CH0 config0 register	0x0500	R/W
H264_DMA_IN_POP_CH0_REG	RX CH0 INFIFO pop register	0x0518	varies
H264_DMA_IN_LINK_CONF_CH0_REG	RX CH0 in_link dscr ctrl register	0x051C	varies
H264_DMA_IN_LINK_ADDR_CH0_REG	RX CH0 in_link dscr addr register	0x0520	R/W
H264_DMA_IN_ARB_CH0_REG	RX CH0 arb register	0x0540	R/W

Name	Description	Address	Access
H264_DMA_IN_RO_PD_CONF_CH0_REG	RX CH0 reorder power config register	0x0548	R/W
H264_DMA_IN_ETM_CONF_CH0_REG	RX CH0 ETM config register	0x056C	R/W
H264_DMA_IN_CONF0_CH1_REG	RX CH1 config0 register	0x0600	R/W
H264_DMA_IN_POP_CH1_REG	RX CH1 INFIFO pop register	0x0618	varies
H264_DMA_IN_LINK_CONF_CH1_REG	RX CH1 in_link dscr ctrl register	0x061C	varies
H264_DMA_IN_LINK_ADDR_CH1_REG	RX CH1 in_link dscr addr register	0x0620	R/W
H264_DMA_IN_ARB_CH1_REG	RX CH1 arb register	0x0640	R/W
H264_DMA_IN_ETM_CONF_CH1_REG	RX CH1 ETM config register	0x0648	R/W
H264_DMA_IN_CONF0_CH2_REG	RX CH2 config0 register	0x0700	R/W
H264_DMA_IN_POP_CH2_REG	RX CH2 INFIFO pop register	0x0718	varies
H264_DMA_IN_LINK_CONF_CH2_REG	RX CH2 in_link dscr ctrl register	0x071C	varies
H264_DMA_IN_LINK_ADDR_CH2_REG	RX CH2 in_link dscr addr register	0x0720	R/W
H264_DMA_IN_ARB_CH2_REG	RX CH2 arb register	0x0740	R/W
H264_DMA_IN_ETM_CONF_CH2_REG	RX CH2 ETM config register	0x0748	R/W
H264_DMA_IN_CONF0_CH3_REG	RX CH3 config0 register	0x0800	R/W
H264_DMA_IN_POP_CH3_REG	RX CH3 INFIFO pop register	0x0818	varies
H264_DMA_IN_LINK_CONF_CH3_REG	RX CH3 in_link dscr ctrl register	0x081C	varies
H264_DMA_IN_LINK_ADDR_CH3_REG	RX CH3 in_link dscr addr register	0x0820	R/W
H264_DMA_IN_ARB_CH3_REG	RX CH3 arb register	0x0840	R/W
H264_DMA_IN_ETM_CONF_CH3_REG	RX CH3 ETM config register	0x0848	R/W
H264_DMA_IN_CONF0_CH4_REG	RX CH4 config0 register	0x0900	R/W
H264_DMA_IN_POP_CH4_REG	RX CH4 INFIFO pop register	0x0918	varies
H264_DMA_IN_LINK_CONF_CH4_REG	RX CH4 in_link dscr ctrl register	0x091C	varies
H264_DMA_IN_LINK_ADDR_CH4_REG	RX CH4 in_link dscr addr register	0x0920	R/W
H264_DMA_IN_ARB_CH4_REG	RX CH4 arb register	0x0940	R/W
H264_DMA_IN_ETM_CONF_CH4_REG	RX CH4 ETM config register	0x0948	R/W
H264_DMA_IN_CONF0_CH5_REG	RX CH5 config0 register	0x0A00	R/W
H264_DMA_IN_CONF1_CH5_REG	RX CH5 config1 register	0x0A04	R/W
H264_DMA_IN_CONF2_CH5_REG	RX CH5 config2 register	0x0A08	R/W
H264_DMA_IN_CONF3_CH5_REG	RX CH5 config3 register	0x0A0C	R/W
H264_DMA_IN_POP_CH5_REG	RX CH5 INFIFO pop register	0x0A24	varies
H264_DMA_IN_ARB_CH5_REG	RX CH5 arb register	0x0A40	R/W
Other Configuration Registers			
H264_DMA_RST_CONF_REG	AXI reset config register	0x0B08	R/W
H264_DMA_INTER_MEM_START_ADDR0_REG	Start address of internal memory range0 register	0x0B0C	R/W
H264_DMA_INTER_MEM_END_ADDR0_REG	end address of internal memory range0 register	0x0B10	R/W
H264_DMA_INTER_MEM_START_ADDR1_REG	Start address of internal memory range1 register	0x0B14	R/W
H264_DMA_INTER_MEM_END_ADDR1_REG	end address of internal memory range1 register	0x0B18	R/W

Name	Description	Address	Access
H264_DMA_EXTER_MEM_START_ADDR0_REG	Start address of external memory range0 register	0x0B20	R/W
H264_DMA_EXTER_MEM_END_ADDR0_REG	end address of external memory range0 register	0x0B24	R/W
H264_DMA_EXTER_MEM_START_ADDR1_REG	Start address of external memory range1 register	0x0B28	R/W
H264_DMA_EXTER_MEM_END_ADDR1_REG	end address of external memory range1 register	0x0B2C	R/W
H264_DMA_OUT_ARB_CONFIG_REG	reserved	0x0B30	R/W
H264_DMA_IN_ARB_CONFIG_REG	reserved	0x0B34	R/W
H264_DMA_COUNTER_RST_REG	counter reset register	0x0B50	R/W
H264_DMA_PBYTE_REG	image pbyte register	0x0B64	R/W
H264_DMA_CH_DBG_EN_REG	channel debug enable register	0x0B68	R/W
H264_DMA_OUT_CH0_DBG_DATA_L_REG	out channel 0 debug data register	0x0B6C	R/W
H264_DMA_OUT_CH0_DBG_DATA_H_REG	out channel 0 debug data register	0x0B70	R/W
H264_DMA_OUT_CH1_DBG_DATA_L_REG	out channel 1 debug data register	0x0B74	R/W
H264_DMA_OUT_CH1_DBG_DATA_H_REG	out channel 1 debug data register	0x0B78	R/W
H264_DMA_OUT_CH2_DBG_DATA_L_REG	out channel 2 debug data register	0x0B7C	R/W
H264_DMA_OUT_CH2_DBG_DATA_H_REG	out channel 2 debug data register	0x0B80	R/W
H264_DMA_OUT_CH3_DBG_DATA_L_REG	out channel 3 debug data register	0x0B84	R/W
H264_DMA_OUT_CH3_DBG_DATA_H_REG	out channel 3 debug data register	0x0B88	R/W
H264_DMA_OUT_CH4_DBG_DATA_L_REG	out channel 4 debug data register	0x0B8C	R/W
H264_DMA_OUT_CH4_DBG_DATA_H_REG	out channel 4 debug data register	0x0B90	R/W
H264_DMA_IN_CH0_DBG_DATA_L_REG	in channel 0 debug data register	0x0B94	R/W
H264_DMA_IN_CH0_DBG_DATA_H_REG	in channel 0 debug data register	0x0B98	R/W
H264_DMA_IN_CH1_DBG_DATA_L_REG	in channel 1 debug data register	0x0B9C	R/W
H264_DMA_IN_CH1_DBG_DATA_H_REG	in channel 1 debug data register	0x0BA0	R/W
H264_DMA_IN_CH2_DBG_DATA_L_REG	in channel 2 debug data register	0x0BA4	R/W
H264_DMA_IN_CH2_DBG_DATA_H_REG	in channel 2 debug data register	0x0BA8	R/W
H264_DMA_IN_CH3_DBG_DATA_L_REG	in channel 3 debug data register	0x0BAC	R/W
H264_DMA_IN_CH3_DBG_DATA_H_REG	in channel 3 debug data register	0x0BB0	R/W
H264_DMA_IN_CH4_DBG_DATA_L_REG	in channel 4 debug data register	0x0BB4	R/W
H264_DMA_IN_CH4_DBG_DATA_H_REG	in channel 4 debug data register	0x0BB8	R/W
H264_DMA_IN_CH5_DBG_DATA_L_REG	in channel 5 debug data register	0x0BBC	R/W
H264_DMA_IN_CH5_DBG_DATA_H_REG	in channel 5 debug data register	0x0BC0	R/W
Interrupt Registers			
TX Interrupt Registers			
H264_DMA_OUT_INT_RAW_CH0_REG	TX CH0 interrupt raw register	0x0004	R/WTC/SS
H264_DMA_OUT_INT_ENA_CH0_REG	TX CH0 interrupt ena register	0x0008	R/W
H264_DMA_OUT_INT_ST_CH0_REG	TX CH0 interrupt st register	0x000C	RO
H264_DMA_OUT_INT_CLR_CH0_REG	TX CH0 interrupt clr register	0x0010	WT
H264_DMA_OUT_INT_RAW_CH1_REG	TX CH1 interrupt raw register	0x0104	R/WTC/SS
H264_DMA_OUT_INT_ENA_CH1_REG	TX CH1 interrupt ena register	0x0108	R/W

Name	Description	Address	Access
H264_DMA_OUT_INT_ST_CH1_REG	TX CH1 interrupt st register	0x010C	RO
H264_DMA_OUT_INT_CLR_CH1_REG	TX CH1 interrupt clr register	0x0110	WT
H264_DMA_OUT_INT_RAW_CH2_REG	TX CH2 interrupt raw register	0x0204	R/WTC/SS
H264_DMA_OUT_INT_ENA_CH2_REG	TX CH2 interrupt ena register	0x0208	R/W
H264_DMA_OUT_INT_ST_CH2_REG	TX CH2 interrupt st register	0x020C	RO
H264_DMA_OUT_INT_CLR_CH2_REG	TX CH2 interrupt clr register	0x0210	WT
H264_DMA_OUT_INT_RAW_CH3_REG	TX CH3 interrupt raw register	0x0304	R/WTC/SS
H264_DMA_OUT_INT_ENA_CH3_REG	TX CH3 interrupt ena register	0x0308	R/W
H264_DMA_OUT_INT_ST_CH3_REG	TX CH3 interrupt st register	0x030C	RO
H264_DMA_OUT_INT_CLR_CH3_REG	TX CH3 interrupt clr register	0x0310	WT
H264_DMA_OUT_INT_RAW_CH4_REG	TX CH4 interrupt raw register	0x0404	R/WTC/SS
H264_DMA_OUT_INT_ENA_CH4_REG	TX CH4 interrupt ena register	0x0408	R/W
H264_DMA_OUT_INT_ST_CH4_REG	TX CH4 interrupt st register	0x040C	RO
H264_DMA_OUT_INT_CLR_CH4_REG	TX CH4 interrupt clr register	0x0410	WT
RX Interrupt Registers			
H264_DMA_IN_INT_RAW_CH0_REG	RX CH0 interrupt raw register	0x0504	R/WTC/SS
H264_DMA_IN_INT_ENA_CH0_REG	RX CH0 interrupt ena register	0x0508	R/W
H264_DMA_IN_INT_ST_CH0_REG	RX CH0 interrupt st register	0x050C	RO
H264_DMA_IN_INT_CLR_CH0_REG	RX CH0 interrupt clr register	0x0510	WT
H264_DMA_IN_INT_RAW_CH1_REG	RX CH1 interrupt raw register	0x0604	R/WTC/SS
H264_DMA_IN_INT_ENA_CH1_REG	RX CH1 interrupt ena register	0x0608	R/W
H264_DMA_IN_INT_ST_CH1_REG	RX CH1 interrupt st register	0x060C	RO
H264_DMA_IN_INT_CLR_CH1_REG	RX CH1 interrupt clr register	0x0610	WT
H264_DMA_IN_INT_RAW_CH2_REG	RX CH2 interrupt raw register	0x0704	R/WTC/SS
H264_DMA_IN_INT_ENA_CH2_REG	RX CH2 interrupt ena register	0x0708	R/W
H264_DMA_IN_INT_ST_CH2_REG	RX CH2 interrupt st register	0x070C	RO
H264_DMA_IN_INT_CLR_CH2_REG	RX CH2 interrupt clr register	0x0710	WT
H264_DMA_IN_INT_RAW_CH3_REG	RX CH3 interrupt raw register	0x0804	R/WTC/SS
H264_DMA_IN_INT_ENA_CH3_REG	RX CH3 interrupt ena register	0x0808	R/W
H264_DMA_IN_INT_ST_CH3_REG	RX CH3 interrupt st register	0x080C	RO
H264_DMA_IN_INT_CLR_CH3_REG	RX CH3 interrupt clr register	0x0810	WT
H264_DMA_IN_INT_RAW_CH4_REG	RX CH4 interrupt raw register	0x0904	R/WTC/SS
H264_DMA_IN_INT_ENA_CH4_REG	RX CH4 interrupt ena register	0x0908	R/W
H264_DMA_IN_INT_ST_CH4_REG	RX CH4 interrupt st register	0x090C	RO
H264_DMA_IN_INT_CLR_CH4_REG	RX CH4 interrupt clr register	0x0910	WT
H264_DMA_IN_INT_RAW_CH5_REG	RX CH5 interrupt raw register	0x0A10	R/WTC/SS
H264_DMA_IN_INT_ENA_CH5_REG	RX CH5 interrupt ena register	0x0A14	R/W
H264_DMA_IN_INT_ST_CH5_REG	RX CH5 interrupt st register	0x0A18	RO
H264_DMA_IN_INT_CLR_CH5_REG	RX CH5 interrupt clr register	0x0A1C	WT
Status Registers			
TX Status Registers			
H264_DMA_OUTFIFO_STATUS_CH0_REG	TX CH0 outfifo status register	0x0014	RO
H264_DMA_OUT_STATE_CH0_REG	TX CH0 state register	0x0024	RO

Name	Description	Address	Access
H264_DMA_OUT_EOF_DES_ADDR_CH0_REG	TX CH0 eof des addr register	0x0028	RO
H264_DMA_OUT_DSCR_CH0_REG	TX CH0 next dscr addr register	0x002C	RO
H264_DMA_OUT_DSCR_BF0_CH0_REG	TX CH0 last dscr addr register	0x0030	RO
H264_DMA_OUT_DSCR_BF1_CH0_REG	TX CH0 second-to-last dscr addr register	0x0034	RO
H264_DMA_OUT_RO_STATUS_CH0_REG	TX CH0 reorder status register	0x0040	RO
H264_DMA_OUT_BUF_LEN_CH0_REG	TX CH0 buf len register	0x0070	RO
H264_DMA_OUT_FIFO_BCNT_CH0_REG	TX CH0 fifo byte cnt register	0x0074	RO
H264_DMA_OUT_PUSH_BYTECNT_CH0_REG	TX CH0 push byte cnt register	0x0078	RO
H264_DMA_OUT_XADDR_CH0_REG	TX CH0 xaddr register	0x007C	RO
H264_DMA_OUTFIFO_STATUS_CH1_REG	TX CH1 outfifo status register	0x0114	RO
H264_DMA_OUT_STATE_CH1_REG	TX CH1 state register	0x0124	RO
H264_DMA_OUT_EOF_DES_ADDR_CH1_REG	TX CH1 eof des addr register	0x0128	RO
H264_DMA_OUT_DSCR_CH1_REG	TX CH1 next dscr addr register	0x012C	RO
H264_DMA_OUT_DSCR_BF0_CH1_REG	TX CH1 last dscr addr register	0x0130	RO
H264_DMA_OUT_DSCR_BF1_CH1_REG	TX CH1 second-to-last dscr addr register	0x0134	RO
H264_DMA_OUT_BUF_LEN_CH1_REG	TX CH1 buf len register	0x0170	RO
H264_DMA_OUT_FIFO_BCNT_CH1_REG	TX CH1 fifo byte cnt register	0x0174	RO
H264_DMA_OUT_PUSH_BYTECNT_CH1_REG	TX CH1 push byte cnt register	0x0178	RO
H264_DMA_OUT_XADDR_CH1_REG	TX CH1 xaddr register	0x017C	RO
H264_DMA_OUTFIFO_STATUS_CH2_REG	TX CH2 outfifo status register	0x0214	RO
H264_DMA_OUT_STATE_CH2_REG	TX CH2 state register	0x0224	RO
H264_DMA_OUT_EOF_DES_ADDR_CH2_REG	TX CH2 eof des addr register	0x0228	RO
H264_DMA_OUT_DSCR_CH2_REG	TX CH2 next dscr addr register	0x022C	RO
H264_DMA_OUT_DSCR_BF0_CH2_REG	TX CH2 last dscr addr register	0x0230	RO
H264_DMA_OUT_DSCR_BF1_CH2_REG	TX CH2 second-to-last dscr addr register	0x0234	RO
H264_DMA_OUT_BUF_LEN_CH2_REG	TX CH2 buf len register	0x0270	RO
H264_DMA_OUT_FIFO_BCNT_CH2_REG	TX CH2 fifo byte cnt register	0x0274	RO
H264_DMA_OUT_PUSH_BYTECNT_CH2_REG	TX CH2 push byte cnt register	0x0278	RO
H264_DMA_OUT_XADDR_CH2_REG	TX CH2 xaddr register	0x027C	RO
H264_DMA_OUTFIFO_STATUS_CH3_REG	TX CH3 outfifo status register	0x0314	RO
H264_DMA_OUT_STATE_CH3_REG	TX CH3 state register	0x0324	RO
H264_DMA_OUT_EOF_DES_ADDR_CH3_REG	TX CH3 eof des addr register	0x0328	RO
H264_DMA_OUT_DSCR_CH3_REG	TX CH3 next dscr addr register	0x032C	RO
H264_DMA_OUT_DSCR_BF0_CH3_REG	TX CH3 last dscr addr register	0x0330	RO
H264_DMA_OUT_DSCR_BF1_CH3_REG	TX CH3 second-to-last dscr addr register	0x0334	RO
H264_DMA_OUT_BUF_LEN_CH3_REG	TX CH3 buf len register	0x0370	RO
H264_DMA_OUT_FIFO_BCNT_CH3_REG	TX CH3 fifo byte cnt register	0x0374	RO
H264_DMA_OUT_PUSH_BYTECNT_CH3_REG	TX CH3 push byte cnt register	0x0378	RO
H264_DMA_OUT_XADDR_CH3_REG	TX CH3 xaddr register	0x037C	RO

Name	Description	Address	Access
H264_DMA_OUT_BLOCK_BUF_LEN_CH3_REG	TX CH3 block buf len register	0x0380	RO
H264_DMA_OUTFIFO_STATUS_CH4_REG	TX CH4 outfifo status register	0x0414	RO
H264_DMA_OUT_STATE_CH4_REG	TX CH4 state register	0x0424	RO
H264_DMA_OUT_EOF_DES_ADDR_CH4_REG	TX CH4 eof des addr register	0x0428	RO
H264_DMA_OUT_DSCR_CH4_REG	TX CH4 next dscr addr register	0x042C	RO
H264_DMA_OUT_DSCR_BF0_CH4_REG	TX CH4 last dscr addr register	0x0430	RO
H264_DMA_OUT_DSCR_BF1_CH4_REG	TX CH4 second-to-last dscr addr register	0x0434	RO
H264_DMA_OUT_BUF_LEN_CH4_REG	TX CH4 buf len register	0x0470	RO
H264_DMA_OUT_FIFO_BCNT_CH4_REG	TX CH4 fifo byte cnt register	0x0474	RO
H264_DMA_OUT_PUSH_BYTECNT_CH4_REG	TX CH4 push byte cnt register	0x0478	RO
H264_DMA_OUT_XADDR_CH4_REG	TX CH4 xaddr register	0x047C	RO
H264_DMA_OUT_BLOCK_BUF_LEN_CH4_REG	TX CH4 block buf len register	0x0480	RO
RX Status Registers			
H264_DMA_INFIFO_STATUS_CH0_REG	RX CH0 INFIFO status register	0x0514	RO
H264_DMA_IN_STATE_CH0_REG	RX CH0 state register	0x0524	RO
H264_DMA_IN_SUC_EOF_DES_ADDR_CH0_REG	RX CH0 eof des addr register	0x0528	RO
H264_DMA_IN_ERR_EOF_DES_ADDR_CH0_REG	RX CH0 err eof des addr register	0x052C	RO
H264_DMA_IN_DSCR_CH0_REG	RX CH0 next dscr addr register	0x0530	RO
H264_DMA_IN_DSCR_BF0_CH0_REG	RX CH0 last dscr addr register	0x0534	RO
H264_DMA_IN_DSCR_BF1_CH0_REG	RX CH0 second-to-last dscr addr register	0x0538	RO
H264_DMA_IN_FIFO_CNT_CH0_REG	RX CH0 fifo cnt register	0x0580	RO
H264_DMA_IN_POP_DATA_CNT_CH0_REG	RX CH0 pop data cnt register	0x0584	RO
H264_DMA_IN_XADDR_CH0_REG	RX CH0 xaddr register	0x0588	RO
H264_DMA_IN_BUF_HB_RCV_CH0_REG	RX CH0 buf len hb rcv register	0x058C	RO
H264_DMA_INFIFO_STATUS_CH1_REG	RX CH1 INFIFO status register	0x0614	RO
H264_DMA_IN_STATE_CH1_REG	RX CH1 state register	0x0624	RO
H264_DMA_IN_SUC_EOF_DES_ADDR_CH1_REG	RX CH1 eof des addr register	0x0628	RO
H264_DMA_IN_ERR_EOF_DES_ADDR_CH1_REG	RX CH1 err eof des addr register	0x062C	RO
H264_DMA_IN_DSCR_CH1_REG	RX CH1 next dscr addr register	0x0630	RO
H264_DMA_IN_DSCR_BF0_CH1_REG	RX CH1 last dscr addr register	0x0634	RO
H264_DMA_IN_DSCR_BF1_CH1_REG	RX CH1 second-to-last dscr addr register	0x0638	RO
H264_DMA_IN_FIFO_CNT_CH1_REG	RX CH1 fifo cnt register	0x0680	RO
H264_DMA_IN_POP_DATA_CNT_CH1_REG	RX CH1 pop data cnt register	0x0684	RO
H264_DMA_IN_XADDR_CH1_REG	RX CH1 xaddr register	0x0688	RO
H264_DMA_IN_BUF_HB_RCV_CH1_REG	RX CH1 buf len hb rcv register	0x068C	RO
H264_DMA_INFIFO_STATUS_CH2_REG	RX CH2 INFIFO status register	0x0714	RO
H264_DMA_IN_STATE_CH2_REG	RX CH2 state register	0x0724	RO
H264_DMA_IN_SUC_EOF_DES_ADDR_CH2_REG	RX CH2 eof des addr register	0x0728	RO
H264_DMA_IN_ERR_EOF_DES_ADDR_CH2_REG	RX CH2 err eof des addr register	0x072C	RO
H264_DMA_IN_DSCR_CH2_REG	RX CH2 next dscr addr register	0x0730	RO

Name	Description	Address	Access
H264_DMA_IN_DSCR_BF0_CH2_REG	RX CH2 last dscr addr register	0x0734	RO
H264_DMA_IN_DSCR_BF1_CH2_REG	RX CH2 second-to-last dscr addr register	0x0738	RO
H264_DMA_IN_FIFO_CNT_CH2_REG	RX CH2 fifo cnt register	0x0780	RO
H264_DMA_IN_POP_DATA_CNT_CH2_REG	RX CH2 pop data cnt register	0x0784	RO
H264_DMA_IN_XADDR_CH2_REG	RX CH2 xaddr register	0x0788	RO
H264_DMA_IN_BUF_HB_RCV_CH2_REG	RX CH2 buf len hb rcv register	0x078C	RO
H264_DMA_INFIFO_STATUS_CH3_REG	RX CH3 INFIFO status register	0x0814	RO
H264_DMA_IN_STATE_CH3_REG	RX CH3 state register	0x0824	RO
H264_DMA_IN_SUC_EOF_DES_ADDR_CH3_REG	RX CH3 eof des addr register	0x0828	RO
H264_DMA_IN_ERR_EOF_DES_ADDR_CH3_REG	RX CH3 err eof des addr register	0x082C	RO
H264_DMA_IN_DSCR_CH3_REG	RX CH3 next dscr addr register	0x0830	RO
H264_DMA_IN_DSCR_BF0_CH3_REG	RX CH3 last dscr addr register	0x0834	RO
H264_DMA_IN_DSCR_BF1_CH3_REG	RX CH3 second-to-last dscr addr register	0x0838	RO
H264_DMA_IN_FIFO_CNT_CH3_REG	RX CH3 fifo cnt register	0x0880	RO
H264_DMA_IN_POP_DATA_CNT_CH3_REG	RX CH3 pop data cnt register	0x0884	RO
H264_DMA_IN_XADDR_CH3_REG	RX CH3 xaddr register	0x0888	RO
H264_DMA_IN_BUF_HB_RCV_CH3_REG	RX CH3 buf len hb rcv register	0x088C	RO
H264_DMA_INFIFO_STATUS_CH4_REG	RX CH4 INFIFO status register	0x0914	RO
H264_DMA_IN_STATE_CH4_REG	RX CH4 state register	0x0924	RO
H264_DMA_IN_SUC_EOF_DES_ADDR_CH4_REG	RX CH4 eof des addr register	0x0928	RO
H264_DMA_IN_ERR_EOF_DES_ADDR_CH4_REG	RX CH4 err eof des addr register	0x092C	RO
H264_DMA_IN_DSCR_CH4_REG	RX CH4 next dscr addr register	0x0930	RO
H264_DMA_IN_DSCR_BF0_CH4_REG	RX CH4 last dscr addr register	0x0934	RO
H264_DMA_IN_DSCR_BF1_CH4_REG	RX CH4 second-to-last dscr addr register	0x0938	RO
H264_DMA_IN_FIFO_CNT_CH4_REG	RX CH4 fifo cnt register	0x0980	RO
H264_DMA_IN_POP_DATA_CNT_CH4_REG	RX CH4 pop data cnt register	0x0984	RO
H264_DMA_IN_XADDR_CH4_REG	RX CH4 xaddr register	0x0988	RO
H264_DMA_IN_BUF_HB_RCV_CH4_REG	RX CH4 buf len hb rcv register	0x098C	RO
H264_DMA_INFIFO_STATUS_CH5_REG	RX CH5 INFIFO status register	0x0A20	RO
H264_DMA_IN_STATE_CH5_REG	RX CH5 state register	0x0A28	RO
H264_DMA_IN_FIFO_CNT_CH5_REG	RX CH5 fifo cnt register	0x0A80	RO
H264_DMA_IN_POP_DATA_CNT_CH5_REG	RX CH5 pop data cnt register	0x0A84	RO
H264_DMA_IN_XADDR_CH5_REG	RX CH5 xaddr register	0x0A88	RO
H264_DMA_IN_BUF_HB_RCV_CH5_REG	RX CH5 buf len hb rcv register	0x0A8C	RO
H264_DMA_INTER_AXI_ERR_REG	internal memory AXI err register	0x0B00	RO
H264_DMA_EXTER_AXI_ERR_REG	external memory AXI err register	0x0B04	RO
H264_DMA_RX_CH0_COUNTER_REG	RX ch0 counter register	0x0B54	RO
H264_DMA_RX_CH1_COUNTER_REG	RX ch1 counter register	0x0B58	RO
H264_DMA_RX_CH2_COUNTER_REG	RX ch2 counter register	0x0B5C	RO
H264_DMA_RX_CH5_COUNTER_REG	RX ch5 counter register	0x0B60	RO

Name	Description	Address	Access
Version Register			
H264_DMA_DATE_REG	Version control register	0x0B3C	R/W

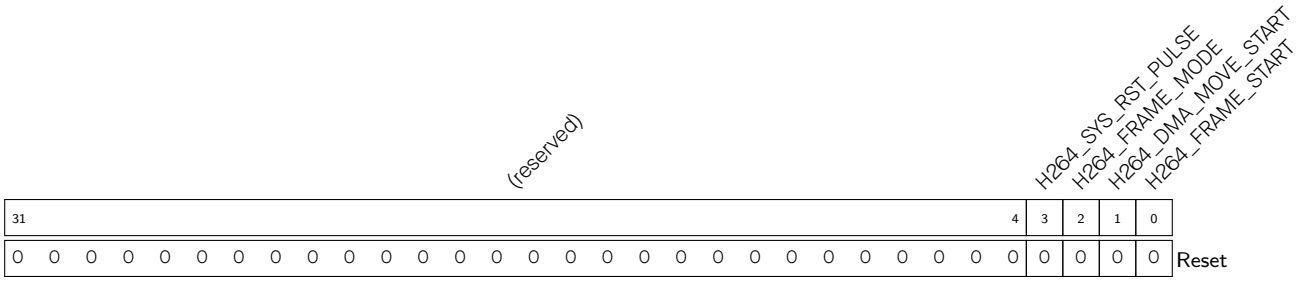
41.9 Registers

The addresses in this section are relative to **H264 Encoder** base address and **H264 DMA** base address. These base addresses are provided in Table 9.3-2 in Chapter 9 *System and Memory*.

For how to program reserved fields, please refer to Section *Programming Reserved Register Field*.

41.9.1 H264 Encoder Registers

Register 41.1. H264_SYS_CTRL_REG (0x0000)



H264_FRAME_START Configures whether to start encoding of a picture.

0: Invalid. No effect

1: Start encoding of a picture

(WT)

H264_DMA_MOVE_START Configures whether to start moving reference data from external mem.

0: Invalid. No effect

1: H264 start moving two MB lines of reference frame from external mem to internal mem

(WT)

H264_FRAME_MODE Configures H264 running mode. When field H264_DUAL_STREAM_MODE is set to 1, this field must be set to 1 too.

0: GOP mode. Before the start of every GOP, the H264_DMA needs to be reconfigured

1: Frame mode. Before the start of every frame, the H264_DMA needs to be reconfigured

(R/W)

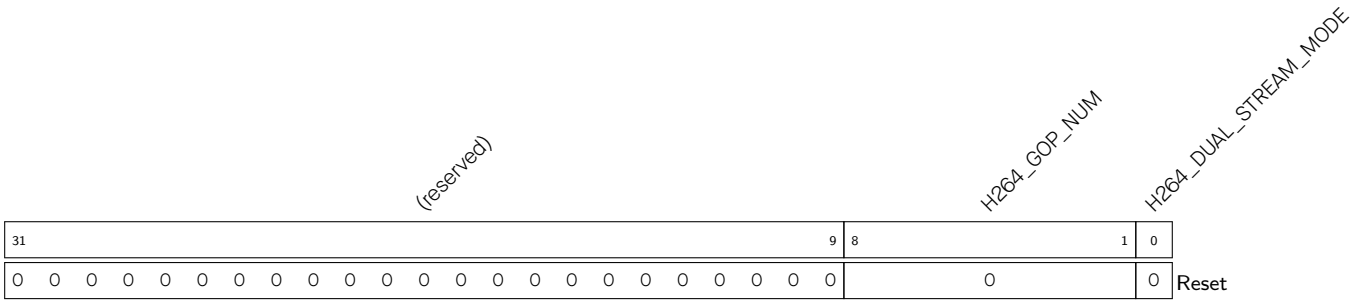
H264_SYS_RST_PULSE Configures whether to reset H264 encoder.

0: Invalid. No effect

1: Reset H264 encoder

(WT)

Register 41.2. H264_GOP_CONF_REG (0x0004)



H264_DUAL_STREAM_MODE Configures whether to enable dual stream mode. When this field is set to 1, H264_FRAME_MODE field must be set to 1 too.

0: GOP mode

1: Dual stream mode

(R/W)

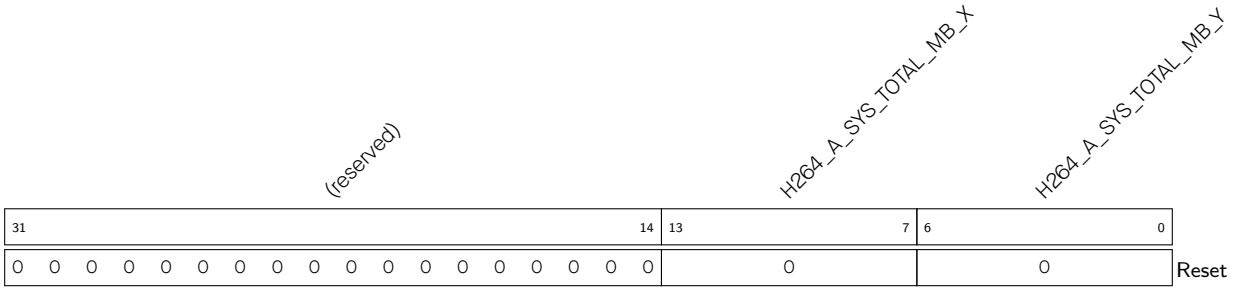
H264_GOP_NUM Configures the frame number of one GOP.

0: The frame number of one GOP is infinite

Others: Actual frame number of one GOP

(R/W)

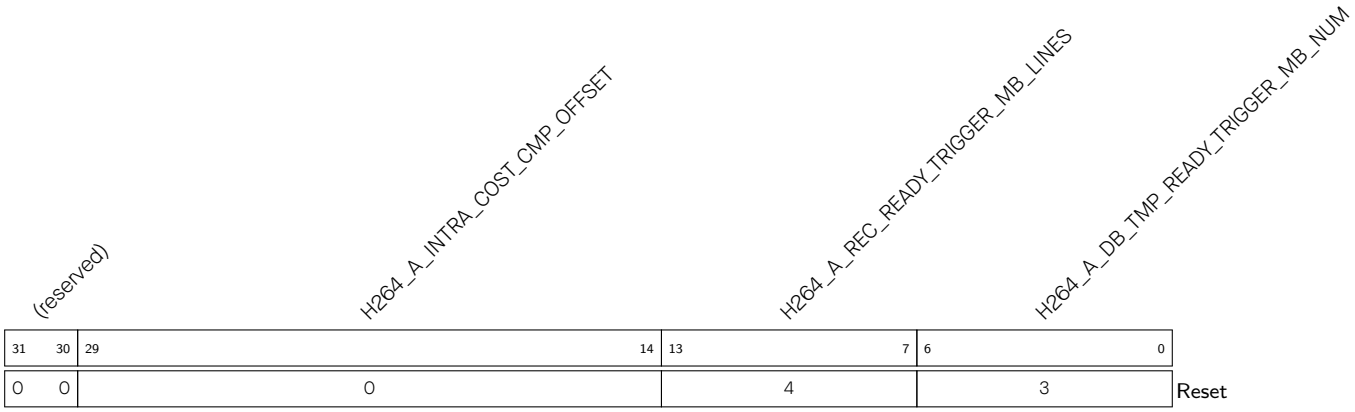
Register 41.3. H264_A_SYS_MB_RES_REG (0x0008)



H264_A_SYS_TOTAL_MB_Y Configures video sequence A vertical MB resolution. (R/W)

H264_A_SYS_TOTAL_MB_X Configures video sequence A horizontal MB resolution. (R/W)

Register 41.4. H264_A_SYS_CONF_REG (0x000C)

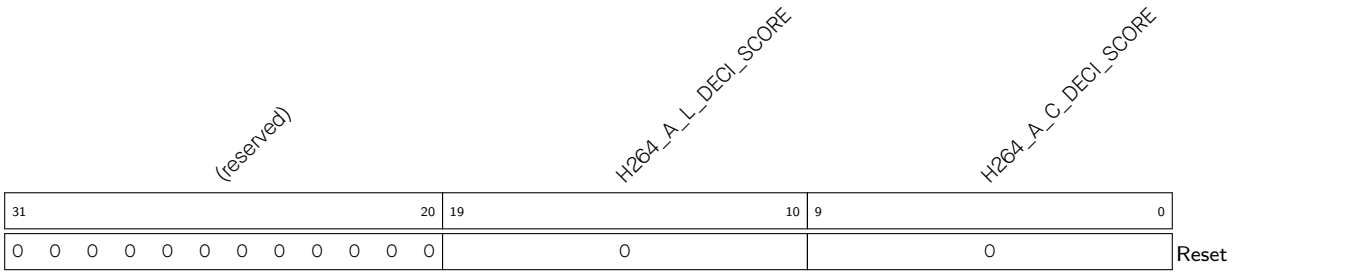


H264_A_DB_TMP_READY_TRIGGER_MB_NUM Configures when to trigger video sequence A H264_DB_TMP_READY_INT. When the (MB number of written db temp+1) is greater than this field in first MB line, trigger H264_DB_TMP_READY_INT. Min is 3. (R/W)

H264_A_REC_READY_TRIGGER_MB_LINES Configures when to trigger video sequence A H264_REC_READY_INT. When the MB line number of generated reconstruct pixel is greater than this field, trigger H264_REC_READY_INT. Min is 4. (R/W)

H264_A_INTRA_COST_CMP_OFFSET Configures video sequence A intra cost offset when I MB compared with P MB. (R/W)

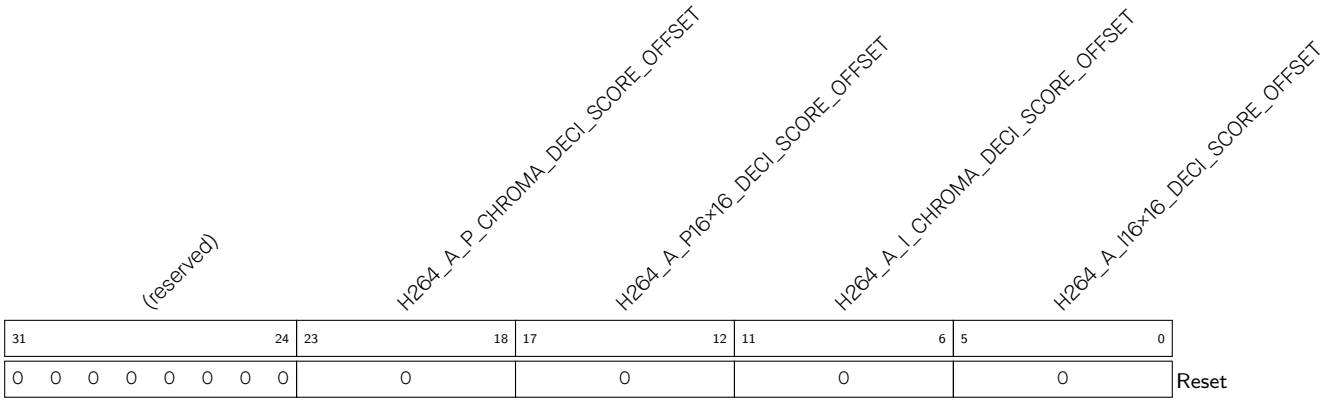
Register 41.5. H264_A_DECI_SCORE_REG (0x0010)



H264_A_C_DECI_SCORE Configures video sequence A chroma MB decimate score. When chroma score is smaller than it, chroma decimate will be enabled. (R/W)

H264_A_L_DECI_SCORE Configures video sequence A luma MB decimate score. When luma score is smaller than it, luma decimate will be enabled. (R/W)

Register 41.6. H264_A_DECI_SCORE_OFFSET_REG (0x0014)



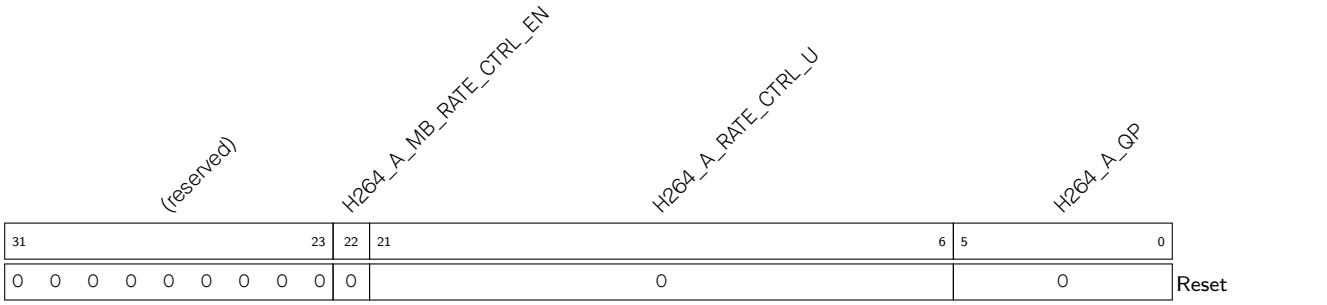
H264_A_I16×16_DECI_SCORE_OFFSET Configures video sequence A i16 × 16 MB decimate score offset. This offset will be added to i16 × 16 MB score. (R/W)

H264_A_I_CHROMA_DECI_SCORE_OFFSET Configures video sequence A I chroma MB decimate score offset. This offset will be added to I chroma MB score. (R/W)

H264_A_P16×16_DECI_SCORE_OFFSET Configures video sequence A p16 × 16 MB decimate score offset. This offset will be added to p16 × 16 MB score. (R/W)

H264_A_P_CHROMA_DECI_SCORE_OFFSET Configures video sequence A p chroma MB decimate score offset. This offset will be added to p chroma MB score. (R/W)

Register 41.7. H264_A_RC_CONF0_REG (0x0018)



H264_A_QP Configures video sequence A frame level initial luma QP value. (R/W)

H264_A_RATE_CTRL_U Configures video sequence A parameter U value. U = `(int)((float)u « 8)`. (R/W)

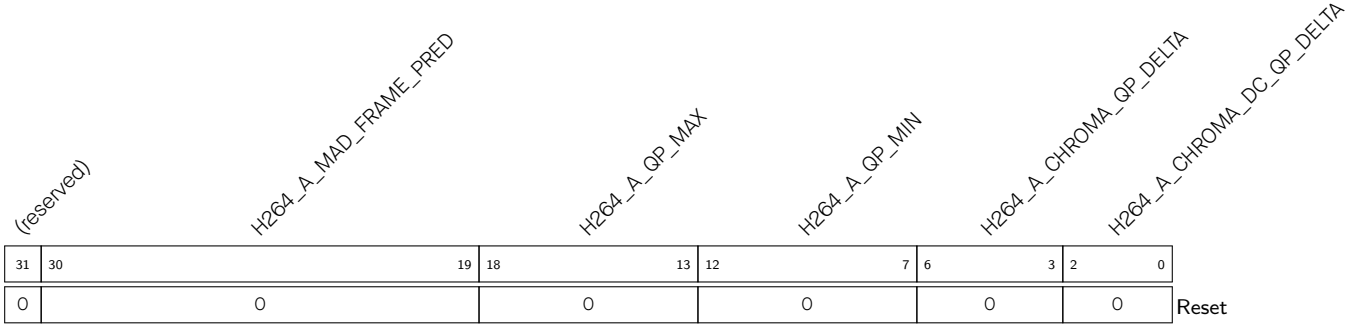
H264_A_MB_RATE_CTRL_EN Configures video sequence A whether to enable the MB level rate ctrl.

0: Disable the MB level rate ctrl

1: Enable the MB level rate ctrl

(R/W)

Register 41.8. H264_A_RC_CONF1_REG (0x001C)



H264_A_CHROMA_DC_QP_DELTA Configures video sequence A chroma DC QP offset based on Chroma QP. Chroma DC QP = Chroma QP(after map) + reg_chroma_dc_QP_delta. (R/W)

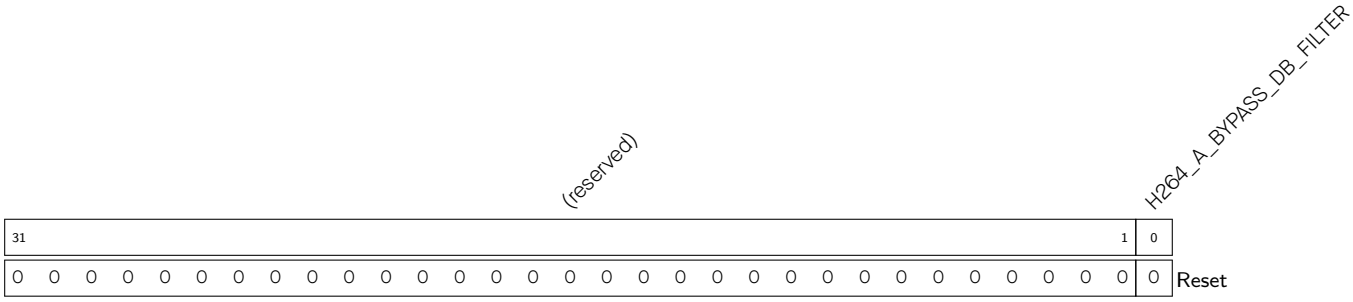
H264_A_CHROMA_QP_DELTA Configures video sequence A chroma QP offset based on luma QP. Chroma QP(before map) = Luma QP + reg_chroma_QP_delta. (R/W)

H264_A_QP_MIN Configures video sequence A allowed luma QP min value. (R/W)

H264_A_QP_MAX Configures video sequence A allowed luma QP max value. (R/W)

H264_A_MAD_FRAME_PRED Configures video sequence A frame level predicted MB MAD value. (R/W)

Register 41.9. H264_A_DB_BYPASS_REG (0x0020)



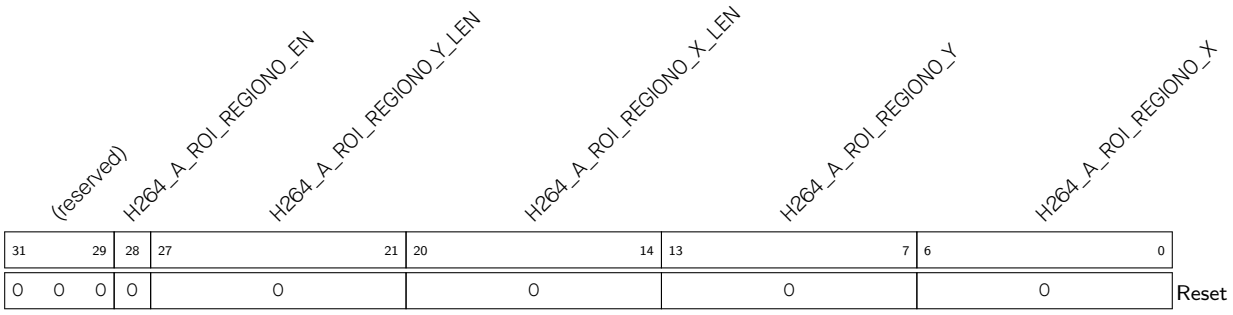
H264_A_BYPASS_DB_FILTER Configures whether to bypass video sequence A deblocking filter.

0: Do not bypass the deblocking filter

1: Bypass the deblocking filter

(R/W)

Register 41.10. H264_A_ROI_REGION0_REG (0x0024)



H264_A_ROI_REGION0_X Configures the horizontal start MBs of ROI 0 in video sequence A. (R/W)

H264_A_ROI_REGION0_Y Configures the vertical start MBs of ROI 0 in video sequence A. (R/W)

H264_A_ROI_REGION0_X_LEN Configures the number of MBs in horizontal direction of ROI 0 in video sequence A. (R/W)

H264_A_ROI_REGION0_Y_LEN Configures the number of MBs in vertical direction of ROI 0 in video sequence A. (R/W)

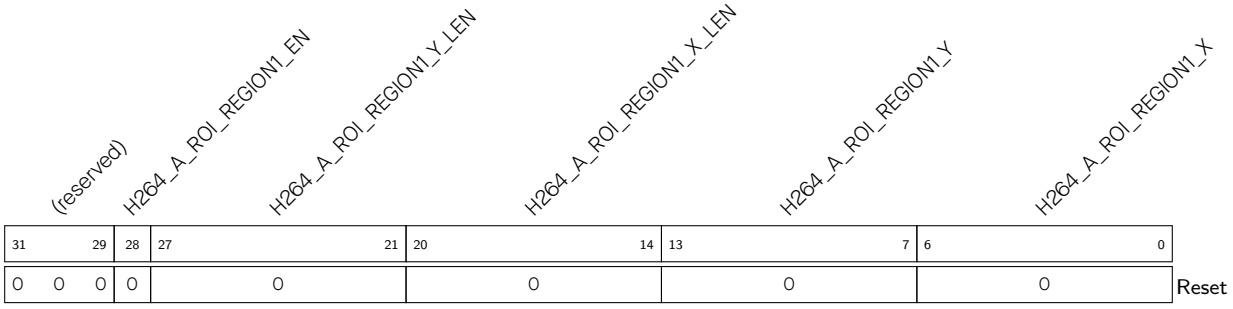
H264_A_ROI_REGION0_EN Configures whether to enable ROI 0 in video sequence A.

0: Disable ROI

1: Enable ROI

(R/W)

Register 41.11. H264_A_ROI_REGION1_REG (0x0028)



H264_A_ROI_REGION1_X Configures the horizontal start MBs of ROI 1 in video sequence A. (R/W)

H264_A_ROI_REGION1_Y Configures the vertical start MBs of ROI 1 in video sequence A. (R/W)

H264_A_ROI_REGION1_X_LEN Configures the number of MBs in horizontal direction of ROI 1 in video sequence A. (R/W)

H264_A_ROI_REGION1_Y_LEN Configures the number of MBs in vertical direction of ROI 1 in video sequence A. (R/W)

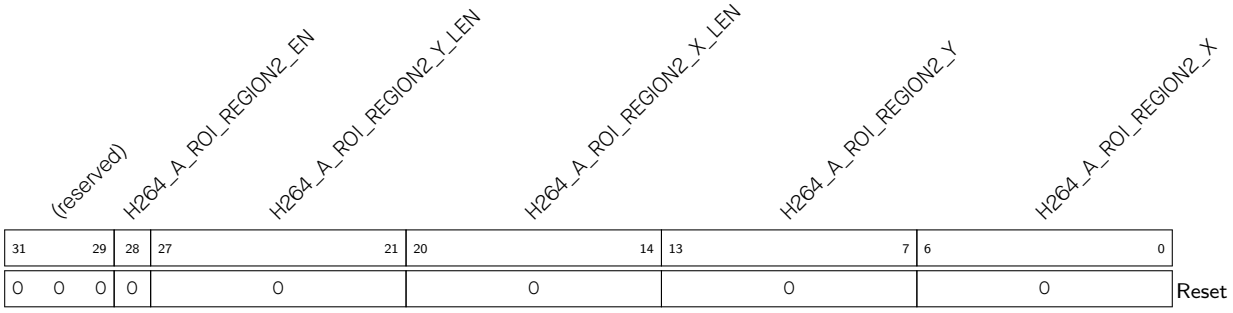
H264_A_ROI_REGION1_EN Configures whether to enable ROI 1 in video sequence A.

0: Disable ROI

1: Enable ROI

(R/W)

Register 41.12. H264_A_ROI_REGION2_REG (0x002C)



H264_A_ROI_REGION2_X Configures the horizontal start MBs of ROI 2 in video sequence A. (R/W)

H264_A_ROI_REGION2_Y Configures the vertical start MBs of ROI 2 in video sequence A. (R/W)

H264_A_ROI_REGION2_X_LEN Configures the number of MBs in horizontal direction of ROI 2 in video sequence A. (R/W)

H264_A_ROI_REGION2_Y_LEN Configures the number of MBs in vertical direction of ROI 2 in video sequence A. (R/W)

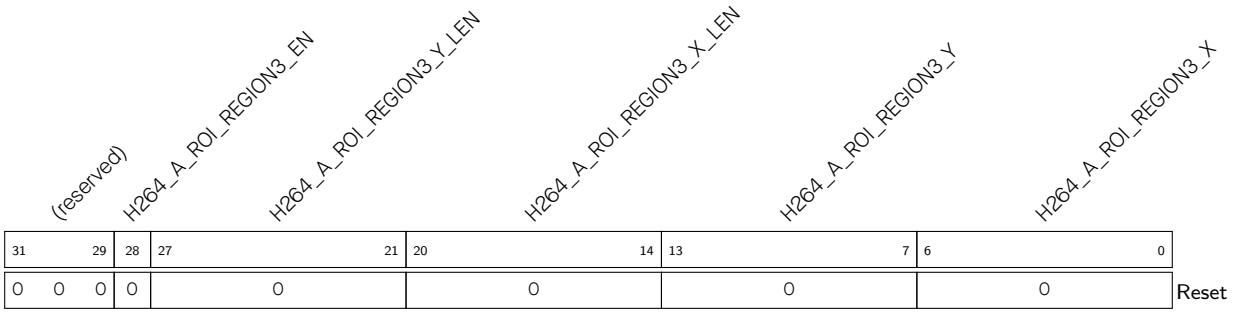
H264_A_ROI_REGION2_EN Configures whether to enable ROI 2 in video sequence A.

0: Disable ROI

1: Enable ROI

(R/W)

Register 41.13. H264_A_ROI_REGION3_REG (0x0030)



H264_A_ROI_REGION3_X Configures the horizontal start MBs of ROI 3 in video sequence A. (R/W)

H264_A_ROI_REGION3_Y Configures the vertical start MBs of ROI 3 in video sequence A. (R/W)

H264_A_ROI_REGION3_X_LEN Configures the number of MBs in horizontal direction of ROI 3 in video sequence A. (R/W)

H264_A_ROI_REGION3_Y_LEN Configures the number of MBs in vertical direction of ROI 3 in video sequence A. (R/W)

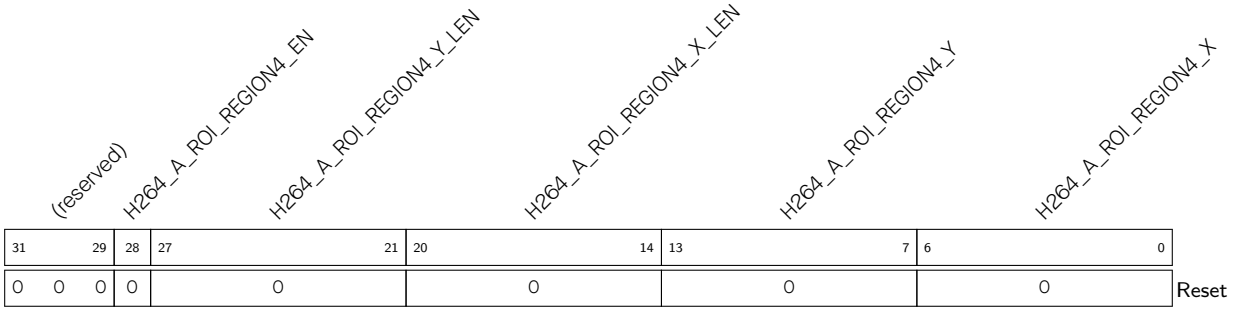
H264_A_ROI_REGION3_EN Configures whether to enable ROI 3 in video sequence A.

0: Disable ROI

1: Enable ROI

(R/W)

Register 41.14. H264_A_ROI_REGION4_REG (0x0034)



H264_A_ROI_REGION4_X Configures the horizontal start MBs of ROI 4 in video sequence A. (R/W)

H264_A_ROI_REGION4_Y Configures the vertical start MBs of ROI 4 in video sequence A. (R/W)

H264_A_ROI_REGION4_X_LEN Configures the number of MBs in horizontal direction of ROI 4 in video sequence A. (R/W)

H264_A_ROI_REGION4_Y_LEN Configures the number of MBs in vertical direction of ROI 4 in video sequence A. (R/W)

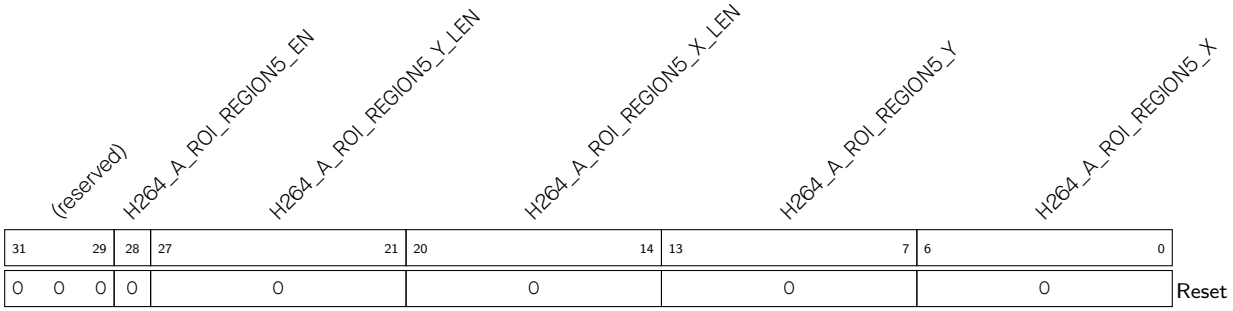
H264_A_ROI_REGION4_EN Configures whether to enable ROI 4 in video sequence A.

0: Disable ROI

1: Enable ROI

(R/W)

Register 41.15. H264_A_ROI_REGION5_REG (0x0038)



H264_A_ROI_REGION5_X Configures the horizontal start MBs of ROI 5 in video sequence A. (R/W)

H264_A_ROI_REGION5_Y Configures the vertical start MBs of ROI 5 in video sequence A. (R/W)

H264_A_ROI_REGION5_X_LEN Configures the number of MBs in horizontal direction of ROI 5 in video sequence A. (R/W)

H264_A_ROI_REGION5_Y_LEN Configures the number of MBs in vertical direction of ROI 5 in video sequence A. (R/W)

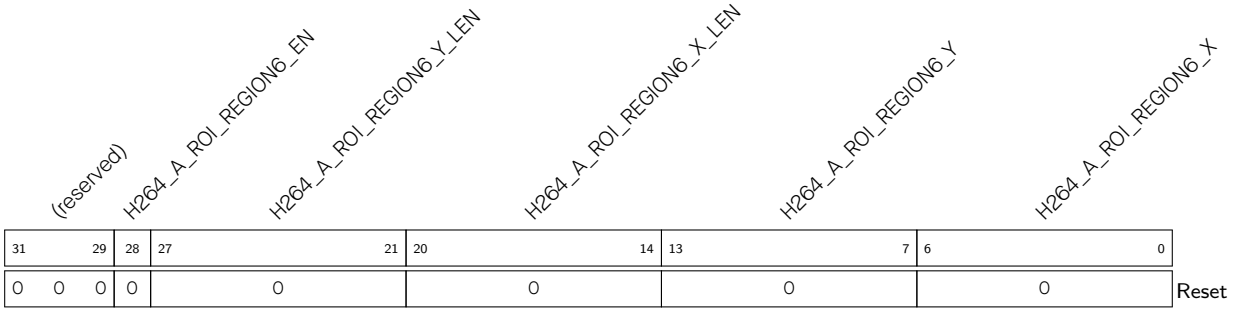
H264_A_ROI_REGION5_EN Configures whether to enable ROI 5 in video sequence A.

0: Disable ROI

1: Enable ROI

(R/W)

Register 41.16. H264_A_ROI_REGION6_REG (0x003C)



H264_A_ROI_REGION6_X Configures the horizontal start MBs of ROI 6 in video sequence A. (R/W)

H264_A_ROI_REGION6_Y Configures the vertical start MBs of ROI 6 in video sequence A. (R/W)

H264_A_ROI_REGION6_X_LEN Configures the number of MBs in horizontal direction of ROI 6 in video sequence A. (R/W)

H264_A_ROI_REGION6_Y_LEN Configures the number of MBs in vertical direction of ROI 6 in video sequence A. (R/W)

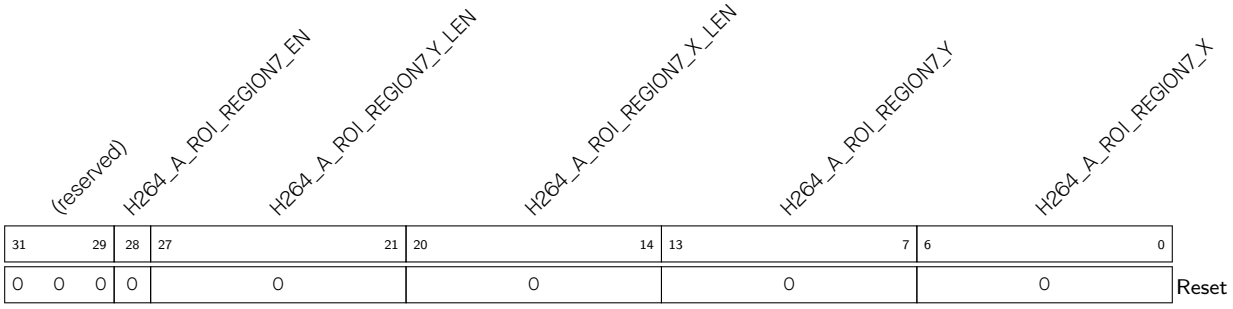
H264_A_ROI_REGION6_EN Configures whether to enable ROI 6 in video sequence A.

0: Disable ROI

1: Enable ROI

(R/W)

Register 41.17. H264_A_ROI_REGION7_REG (0x0040)



H264_A_ROI_REGION7_X Configures the horizontal start MBs of ROI 7 in video sequence A. (R/W)

H264_A_ROI_REGION7_Y Configures the vertical start MBs of ROI 7 in video sequence A. (R/W)

H264_A_ROI_REGION7_X_LEN Configures the number of MBs in horizontal direction of ROI 7 in video sequence A. (R/W)

H264_A_ROI_REGION7_Y_LEN Configures the number of MBs in vertical direction of ROI 7 in video sequence A. (R/W)

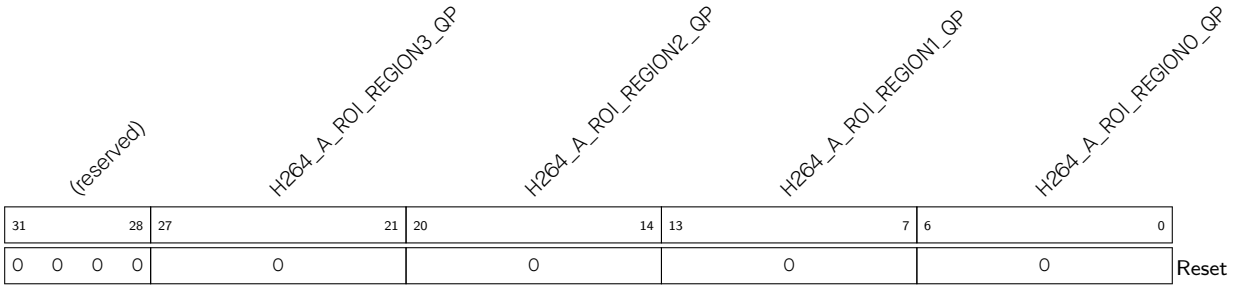
H264_A_ROI_REGION7_EN Configures whether to enable ROI 7 in video sequence A.

0: Disable ROI

1: Enable ROI

(R/W)

Register 41.18. H264_A_ROI_REGION0_3_QP_REG (0x0044)



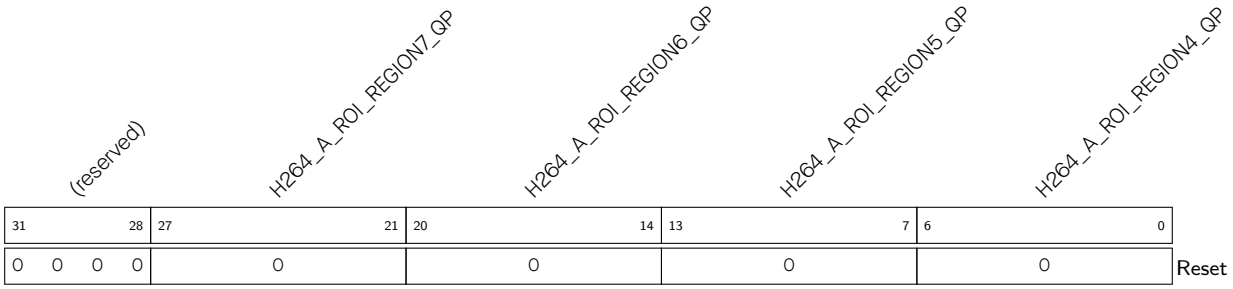
H264_A_ROI_REGION0_QP Configures H264 ROI 0 QP adjustment amount in video sequence A. (R/W)

H264_A_ROI_REGION1_QP Configures H264 ROI 1 QP adjustment amount in video sequence A. (R/W)

H264_A_ROI_REGION2_QP Configures H264 ROI 2 QP adjustment amount in video sequence A. (R/W)

H264_A_ROI_REGION3_QP Configures H264 ROI 3 QP adjustment amount in video sequence A. (R/W)

Register 41.19. H264_A_ROI_REGION4_7_QP_REG (0x0048)



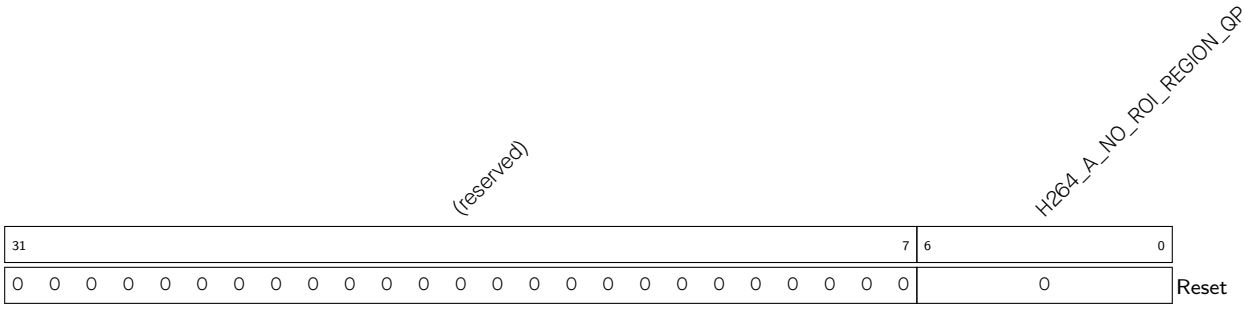
H264_A_ROI_REGION4_QP Configures H264 ROI 4 QP adjustment amount in video sequence A. (R/W)

H264_A_ROI_REGION5_QP Configures H264 ROI 5 QP adjustment amount in video sequence A. (R/W)

H264_A_ROI_REGION6_QP Configures H264 ROI 6 QP adjustment amount in video sequence A. (R/W)

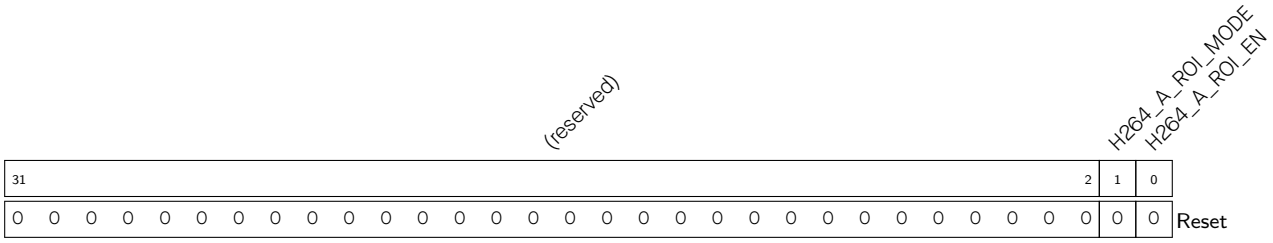
H264_A_ROI_REGION7_QP Configures H264 ROI 7 QP adjustment amount in video sequence A. (R/W)

Register 41.20. H264_A_NO_ROI_REGION_QP_OFFSET_REG (0x004C)



H264_A_NO_ROI_REGION_QP Configures H264 non-ROI QP in video sequence A, delta QP. (R/W)

Register 41.21. H264_A_ROI_CONFIG_REG (0x0050)



H264_A_ROI_EN Configures whether to enable ROI in video sequence A.

0: Disable ROI

1: Enable ROI

(R/W)

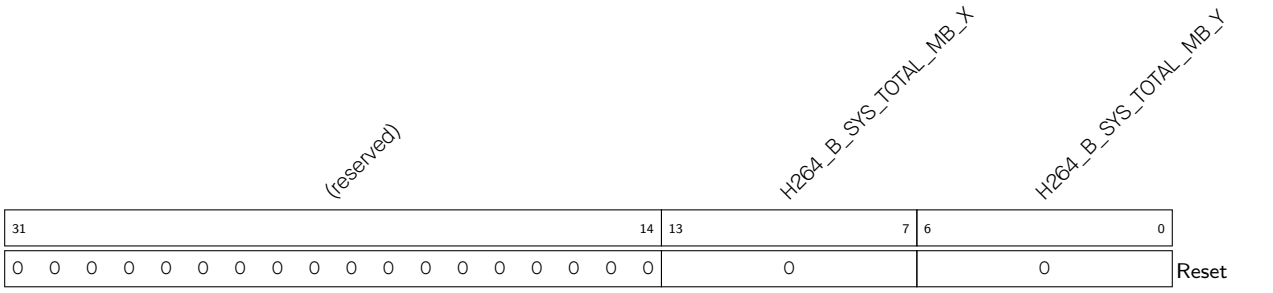
H264_A_ROI_MODE Configures the mode of ROI in video sequence A.

0: Fixed QP

1: Delta QP

(R/W)

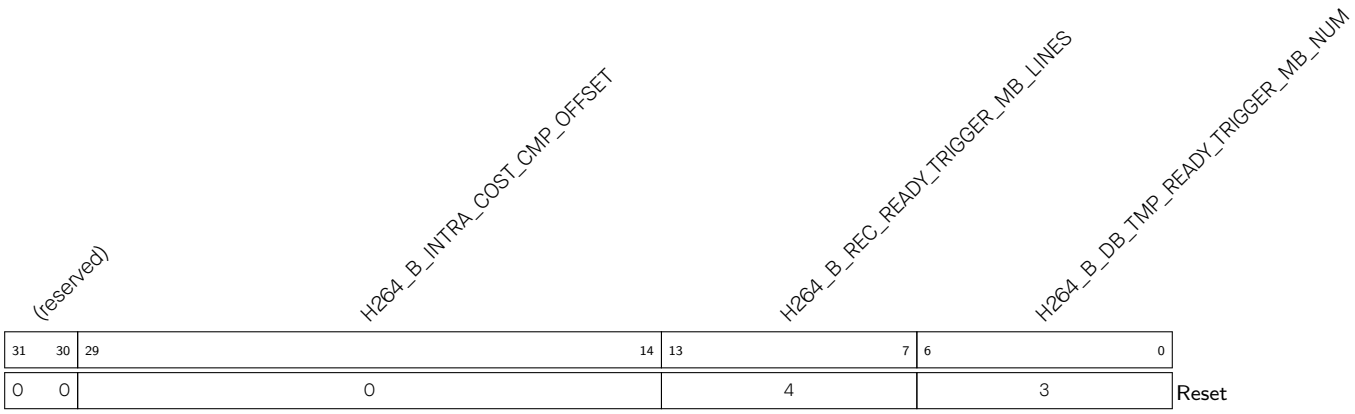
Register 41.22. H264_B_SYS_MB_RES_REG (0x0054)



H264_B_SYS_TOTAL_MB_Y Configures video sequence B vertical MB resolution. (R/W)

H264_B_SYS_TOTAL_MB_X Configures video sequence B horizontal MB resolution. (R/W)

Register 41.23. H264_B_SYS_CONF_REG (0x0058)

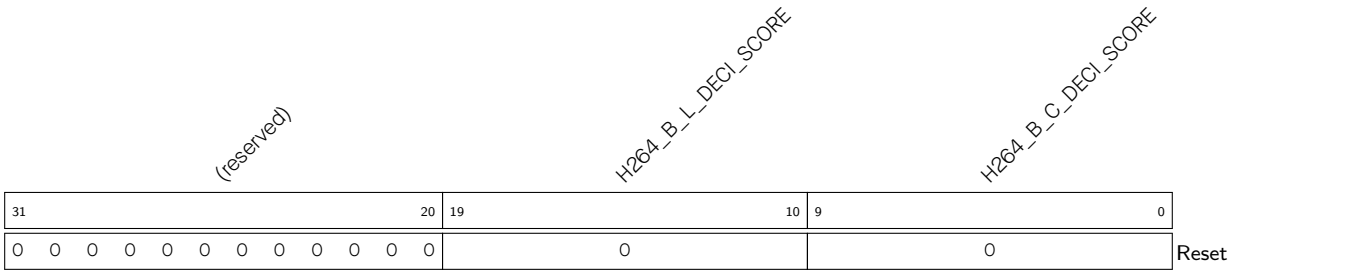


H264_B_DB_TMP_READY_TRIGGER_MB_NUM Configures when to trigger video sequence B H264_DB_TMP_READY_INT. When the (MB number of written db temp+1) is greater than this field in first MB line, trigger H264_DB_TMP_READY_INT. Min is 3. (R/W)

H264_B_REC_READY_TRIGGER_MB_LINES Configures when to trigger video sequence B H264_REC_READY_INT. When the MB line number of generated reconstruct pixel is greater than this field, trigger H264_REC_READY_INT. Min is 4. (R/W)

H264_B_INTRA_COST_CMP_OFFSET Configures video sequence B intra cost offset when I MB compared with P MB. (R/W)

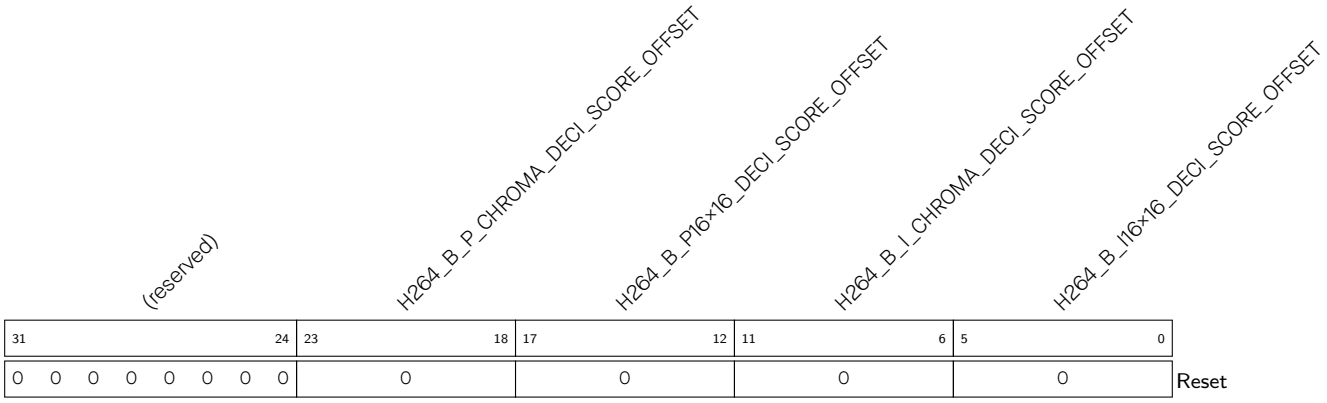
Register 41.24. H264_B_DECI_SCORE_REG (0x005C)



H264_B_C_DECI_SCORE Configures video sequence B chroma MB decimate score. When chroma score is smaller than it, chroma decimate will be enabled. (R/W)

H264_B_L_DECI_SCORE Configures video sequence B luma MB decimate score. When luma score is smaller than it, luma decimate will be enabled. (R/W)

Register 41.25. H264_B_DECI_SCORE_OFFSET_REG (0x0060)



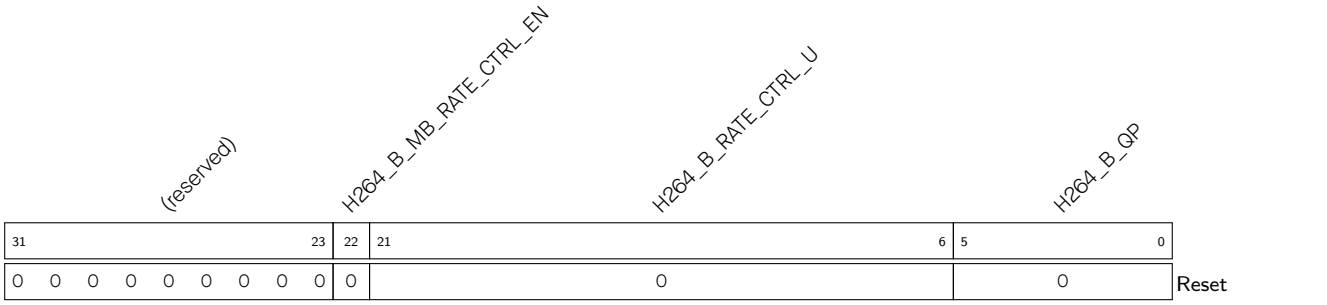
H264_B_I16×16_DECI_SCORE_OFFSET Configures video sequence B i16 × 16 MB decimate score offset. This offset will be added to i16 × 16 MB score. (R/W)

H264_B_I_CHROMA_DECI_SCORE_OFFSET Configures video sequence B I chroma MB decimate score offset. This offset will be added to I chroma MB score. (R/W)

H264_B_P16×16_DECI_SCORE_OFFSET Configures video sequence B p16 × 16 MB decimate score offset. This offset will be added to p16 × 16 MB score. (R/W)

H264_B_P_CHROMA_DECI_SCORE_OFFSET Configures video sequence B p chroma MB decimate score offset. This offset will be added to p chroma MB score. (R/W)

Register 41.26. H264_B_RC_CONF0_REG (0x0064)

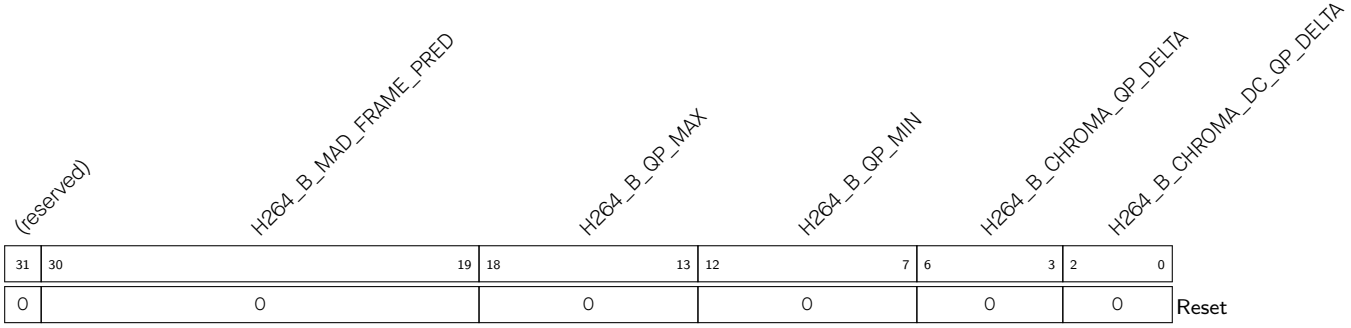


H264_B_QP Configures video sequence B frame level initial luma QP value. (R/W)

H264_B_RATE_CTRL_U Configures video sequence B parameter U value. U = int((float) u « 8). (R/W)

H264_B_MB_RATE_CTRL_EN Configures video sequence B whether to enable MB level rate ctrl.
0: Disable the MB level rate ctrl
1: Enable the MB level rate ctrl
(R/W)

Register 41.27. H264_B_RC_CONF1_REG (0x0068)



H264_B_CHROMA_DC_QP_DELTA Configures video sequence B chroma DC QP offset based on Chroma QP. Chroma DC QP = Chroma QP (after mapping) + reg_chroma_dc_QP_delta. (R/W)

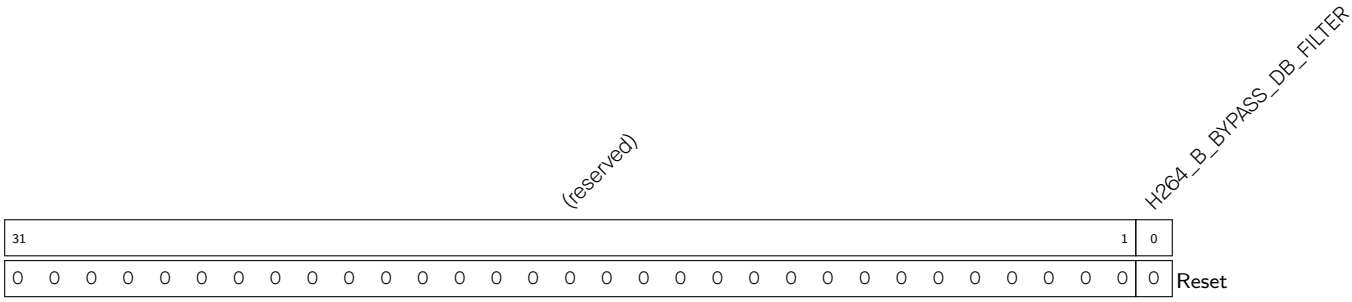
H264_B_CHROMA_QP_DELTA Configures video sequence B chroma QP offset based on luma QP. Chroma QP(before mapping) = Luma QP + reg_chroma_QP_delta. (R/W)

H264_B_QP_MIN Configures the minimum allowed luma QP value for video sequence B. (R/W)

H264_B_QP_MAX Configures the maximum allowed luma QP value for video sequence B. (R/W)

H264_B_MAD_FRAME_PRED Configures the MAD value for frame-level predicted MB in video sequence B. (R/W)

Register 41.28. H264_B_DB_BYPASS_REG (0x006C)



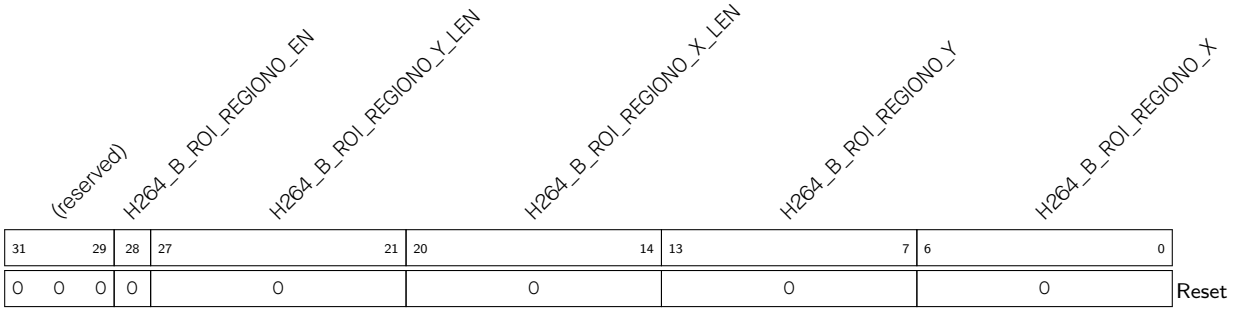
H264_B_BYPASS_DB_FILTER Configures whether to bypass video sequence B deblocking filter.

0: Do not bypass the deblocking filter

1: Bypass the deblocking filter

(R/W)

Register 41.29. H264_B_ROI_REGION0_REG (0x0070)



H264_B_ROI_REGION0_X Configures the horizontal start MBs of ROI 0 in video sequence B. (R/W)

H264_B_ROI_REGION0_Y Configures the vertical start MBs of ROI 0 in video sequence B. (R/W)

H264_B_ROI_REGION0_X_LEN Configures the number of MBs in horizontal direction of ROI 0 in video sequence B. (R/W)

H264_B_ROI_REGION0_Y_LEN Configures the number of MBs in vertical direction of ROI 0 in video sequence B. (R/W)

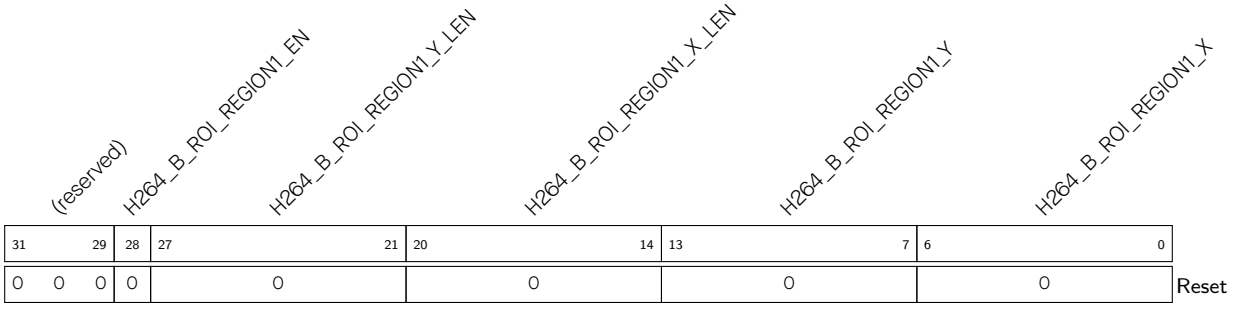
H264_B_ROI_REGION0_EN Configures whether to enable ROI 0 in video sequence B.

0: Disable ROI

1: Enable ROI

(R/W)

Register 41.30. H264_B_ROI_REGION1_REG (0x0074)



H264_B_ROI_REGION1_X Configures the horizontal start MBs of ROI 1 in video sequence B. (R/W)

H264_B_ROI_REGION1_Y Configures the vertical start MBs of ROI 1 in video sequence B. (R/W)

H264_B_ROI_REGION1_X_LEN Configures the number of MBs in horizontal direction of ROI 1 in video sequence B. (R/W)

H264_B_ROI_REGION1_Y_LEN Configures the number of MBs in vertical direction of ROI 1 in video sequence B. (R/W)

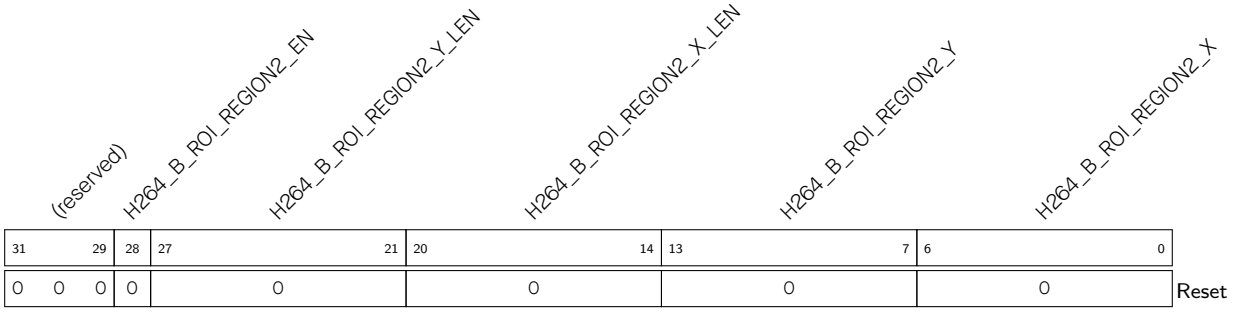
H264_B_ROI_REGION1_EN Configures whether to enable ROI 1 in video sequence B.

0: Disable ROI

1: Enable ROI

(R/W)

Register 41.31. H264_B_ROI_REGION2_REG (0x0078)



H264_B_ROI_REGION2_X Configures the horizontal start MBs of ROI 2 in video sequence B. (R/W)

H264_B_ROI_REGION2_Y Configures the vertical start MBs of ROI 2 in video sequence B. (R/W)

H264_B_ROI_REGION2_X_LEN Configures the number of MBs in horizontal direction of ROI 2 in video sequence B. (R/W)

H264_B_ROI_REGION2_Y_LEN Configures the number of MBs in vertical direction of ROI 2 in video sequence B. (R/W)

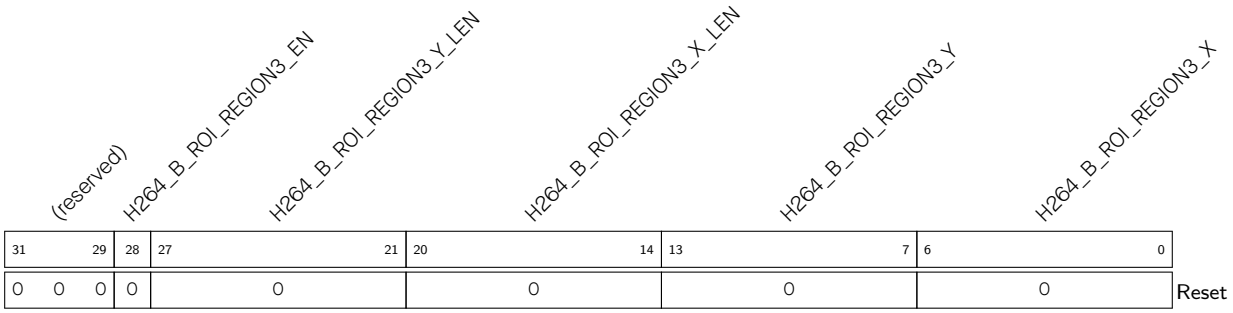
H264_B_ROI_REGION2_EN Configures whether to enable ROI 2 in video sequence B.

0: Disable ROI

1: Enable ROI

(R/W)

Register 41.32. H264_B_ROI_REGION3_REG (0x007C)



H264_B_ROI_REGION3_X Configures the horizontal start MBs of ROI 3 in video sequence B. (R/W)

H264_B_ROI_REGION3_Y Configures the vertical start MBs of ROI 3 in video sequence B. (R/W)

H264_B_ROI_REGION3_X_LEN Configures the number of MBs in horizontal direction of ROI 3 in video sequence B. (R/W)

H264_B_ROI_REGION3_Y_LEN Configures the number of MBs in vertical direction of ROI 3 in video sequence B. (R/W)

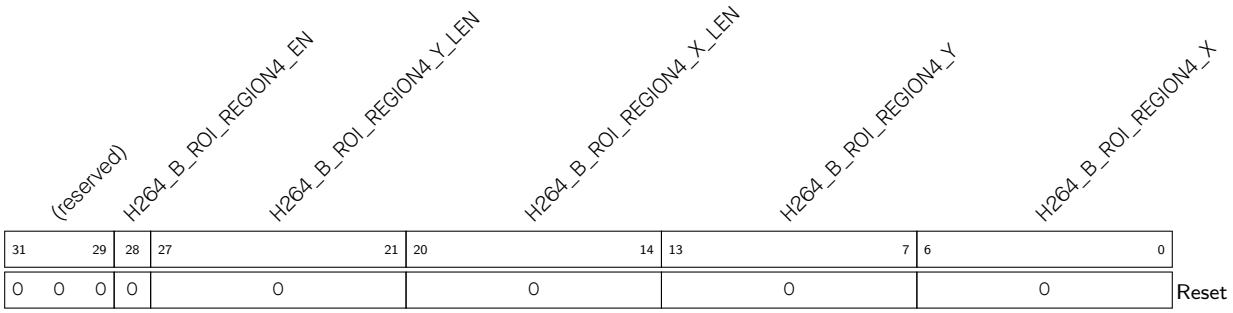
H264_B_ROI_REGION3_EN Configures whether to enable ROI 3 in video sequence B.

0: Disable ROI

1: Enable ROI

(R/W)

Register 41.33. H264_B_ROI_REGION4_REG (0x0080)



H264_B_ROI_REGION4_X Configures the horizontal start MBs of ROI 4 in video sequence B. (R/W)

H264_B_ROI_REGION4_Y Configures the vertical start MBs of ROI 4 in video sequence B. (R/W)

H264_B_ROI_REGION4_X_LEN Configures the number of MBs in horizontal direction of ROI 4 in video sequence B. (R/W)

H264_B_ROI_REGION4_Y_LEN Configures the number of MBs in vertical direction of ROI 4 in video sequence B. (R/W)

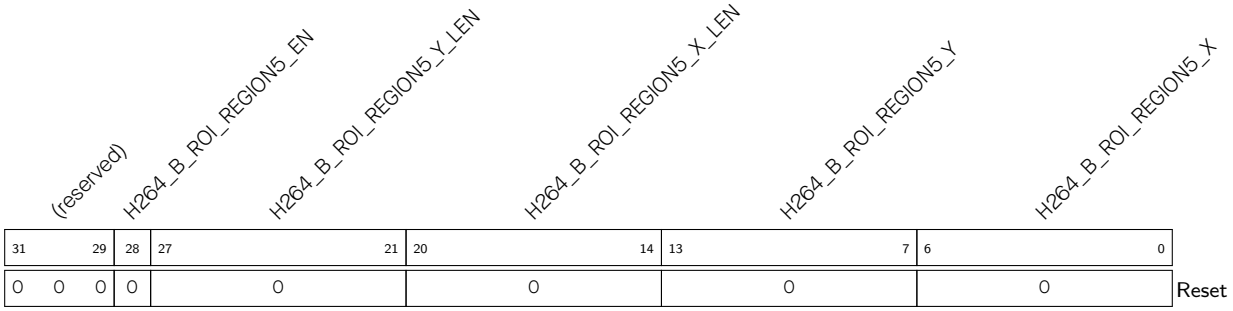
H264_B_ROI_REGION4_EN Configures whether to enable ROI 4 in video sequence B.

0: Disable ROI

1: Enable ROI

(R/W)

Register 41.34. H264_B_ROI_REGION5_REG (0x0084)



H264_B_ROI_REGION5_X Configures the horizontal start MBs of ROI 5 in video sequence B. (R/W)

H264_B_ROI_REGION5_Y Configures the vertical start MBs of ROI 5 in video sequence B. (R/W)

H264_B_ROI_REGION5_X_LEN Configures the number of MBs in horizontal direction of ROI 5 in video sequence B. (R/W)

H264_B_ROI_REGION5_Y_LEN Configures the number of MBs in vertical direction of ROI 5 in video sequence B. (R/W)

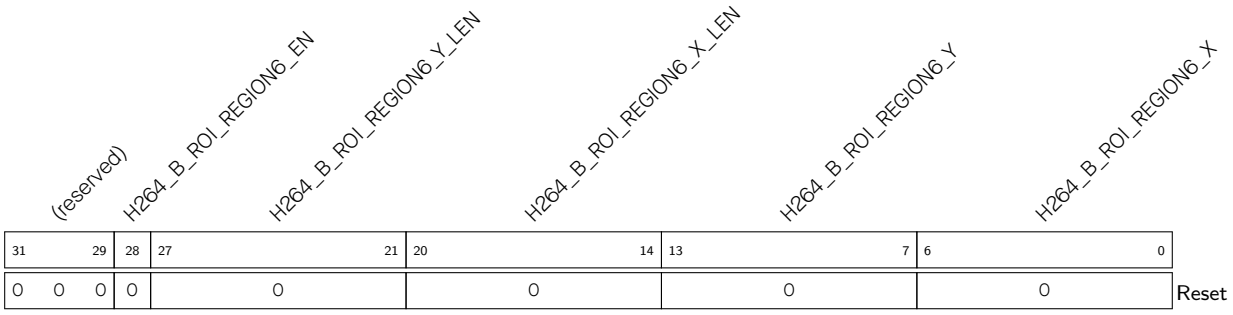
H264_B_ROI_REGION5_EN Configures whether to enable ROI 5 in video sequence B.

0: Disable ROI

1: Enable ROI

(R/W)

Register 41.35. H264_B_ROI_REGION6_REG (0x0088)



H264_B_ROI_REGION6_X Configures the horizontal start MBs of ROI 6 in video sequence B. (R/W)

H264_B_ROI_REGION6_Y Configures the vertical start MBs of ROI 6 in video sequence B. (R/W)

H264_B_ROI_REGION6_X_LEN Configures the number of MBs in horizontal direction of ROI 6 in video sequence B. (R/W)

H264_B_ROI_REGION6_Y_LEN Configures the number of MBs in vertical direction of ROI 6 in video sequence B. (R/W)

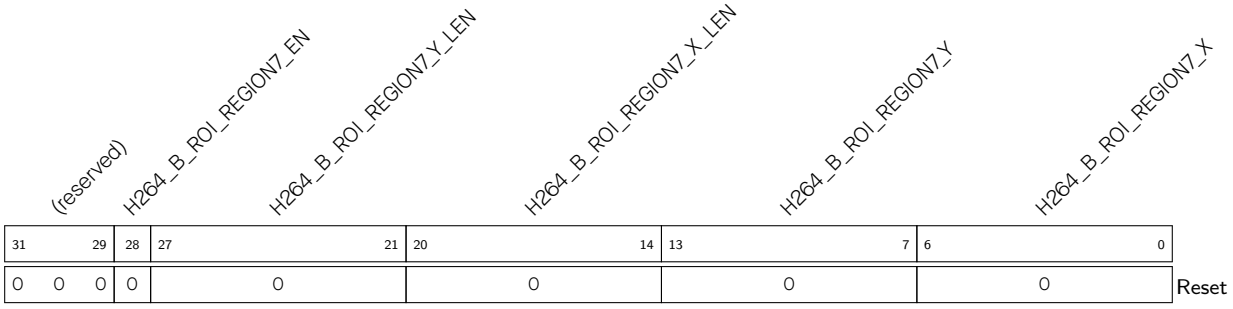
H264_B_ROI_REGION6_EN Configures whether to enable ROI 6 in video sequence B.

0: Disable ROI

1: Enable ROI

(R/W)

Register 41.36. H264_B_ROI_REGION7_REG (0x008C)



H264_B_ROI_REGION7_X Configures the horizontal start MBs of ROI 7 in video sequence B. (R/W)

H264_B_ROI_REGION7_Y Configures the vertical start MBs of ROI 7 in video sequence B. (R/W)

H264_B_ROI_REGION7_X_LEN Configures the number of MBs in horizontal direction of ROI 7 in video sequence B. (R/W)

H264_B_ROI_REGION7_Y_LEN Configures the number of MBs in vertical direction of ROI 7 in video sequence B. (R/W)

H264_B_ROI_REGION7_EN Configures whether to enable ROI 7 in video sequence B.

0: Disable ROI

1: Enable ROI

(R/W)

Register 41.37. H264_B_ROI_REGION0_3_QP_REG (0x0090)



H264_B_ROI_REGION0_QP Configures H264 ROI 0 QP adjustment amount in video sequence B. (R/W)

H264_B_ROI_REGION1_QP Configures H264 ROI 1 QP adjustment amount in video sequence B. (R/W)

H264_B_ROI_REGION2_QP Configures H264 ROI 2 QP adjustment amount in video sequence B. (R/W)

H264_B_ROI_REGION3_QP Configures H264 ROI 3 QP adjustment amount in video sequence B. (R/W)

Register 41.38. H264_B_ROI_REGION4_7_QP_REG (0x0094)



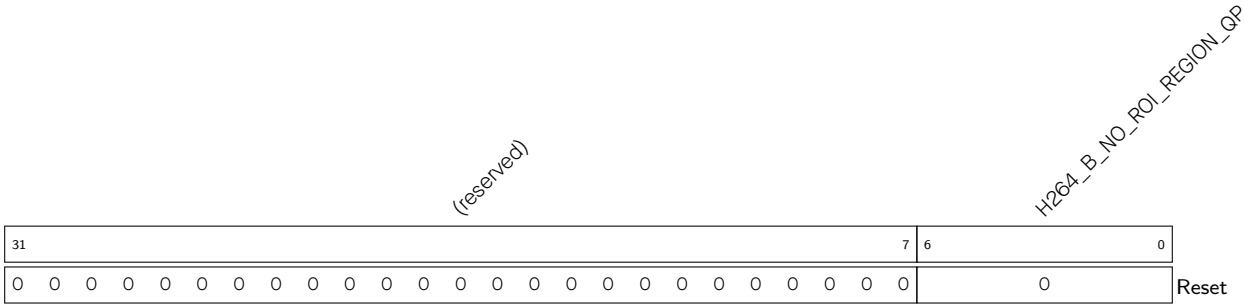
H264_B_ROI_REGION4_QP Configures H264 ROI 4 QP adjustment amount in video sequence B. (R/W)

H264_B_ROI_REGION5_QP Configures H264 ROI 5 QP adjustment amount in video sequence B. (R/W)

H264_B_ROI_REGION6_QP Configures H264 ROI 6 QP adjustment amount in video sequence B. (R/W)

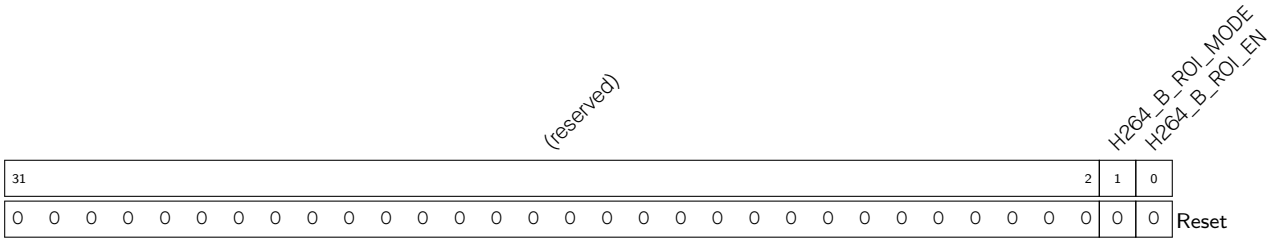
H264_B_ROI_REGION7_QP Configures H264 ROI 7 QP adjustment amount in video sequence B. (R/W)

Register 41.39. H264_B_NO_ROI_REGION_QP_OFFSET_REG (0x0098)



H264_B_NO_ROI_REGION_QP Configures H264 non-ROI QP in video sequence B, delta QP. (R/W)

Register 41.40. H264_B_ROI_CONFIG_REG (0x009C)



H264_B_ROI_EN Configures whether to enable ROI in video sequence B.

0: Disable ROI

1: Enable ROI

(R/W)

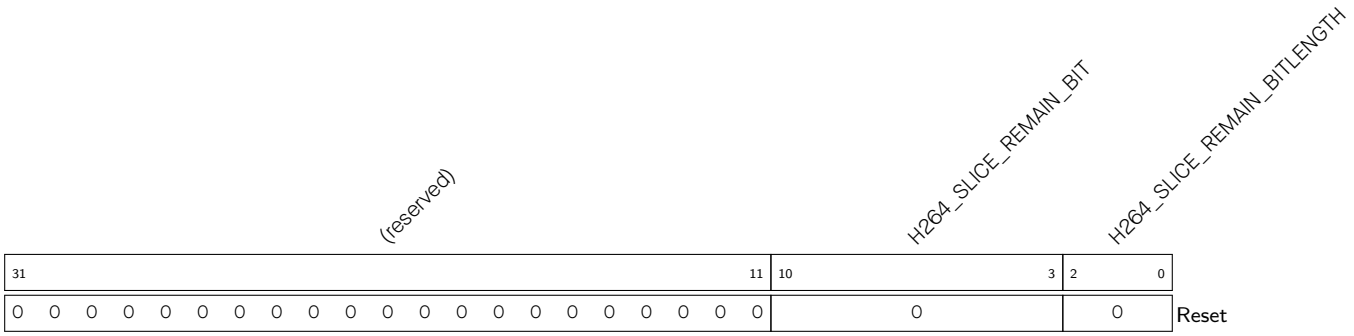
H264_B_ROI_MODE Configures the mode of ROI in video sequence B.

0: Fixed QP

1: Delta QP

(R/W)

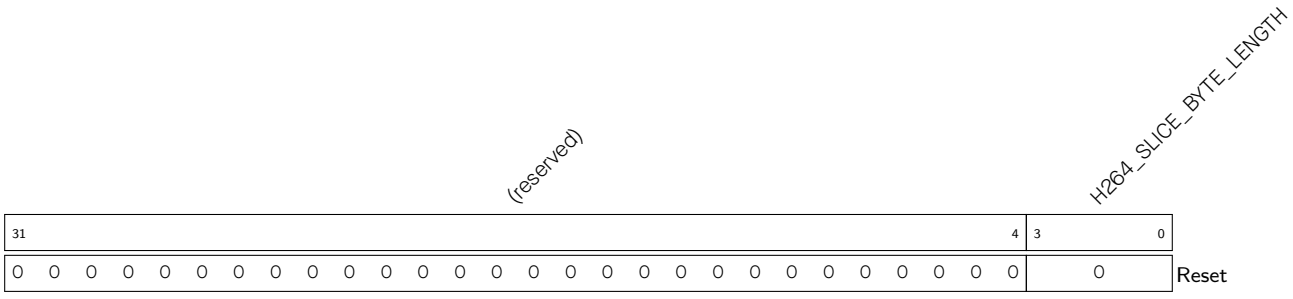
Register 41.41. H264_SLICE_HEADER_REMAIN_REG (0x00AC)



H264_SLICE_REMAIN_BITLENGTH Configures slice header remain bit number. (R/W)

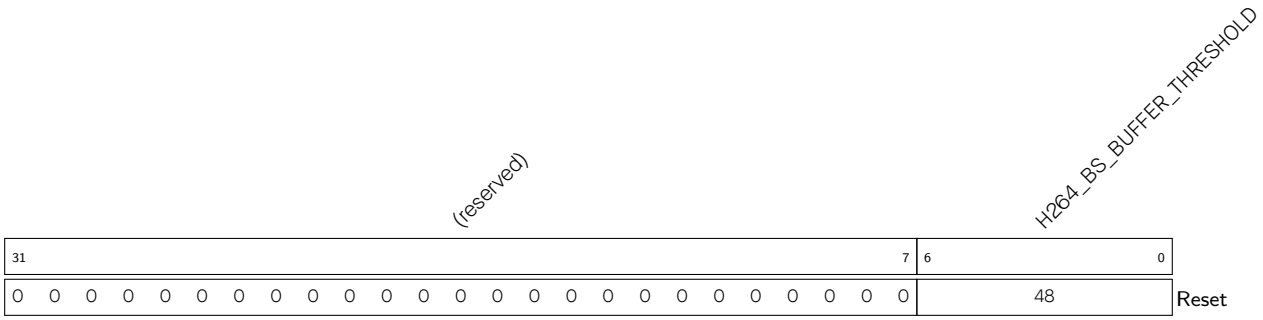
H264_SLICE_REMAIN_BIT Configures slice header remain bit. (R/W)

Register 41.42. H264_SLICE_HEADER_BYTE_LENGTH_REG (0x00B0)



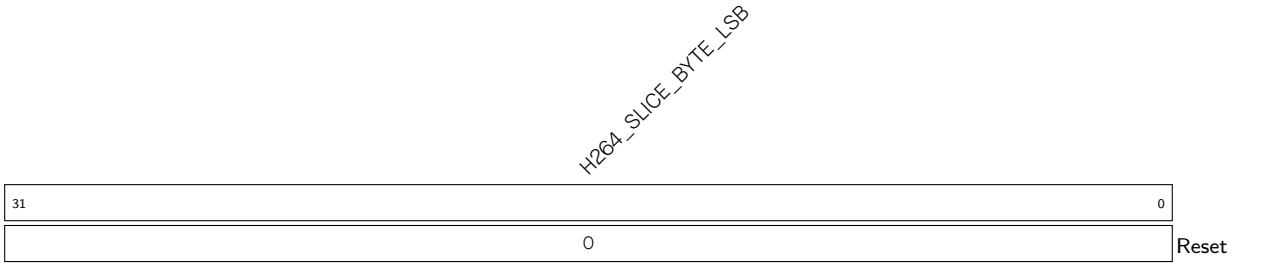
H264_SLICE_BYTE_LENGTH Configures slice header byte number. (R/W)

Register 41.43. H264_BS_THRESHOLD_REG (0x00B4)



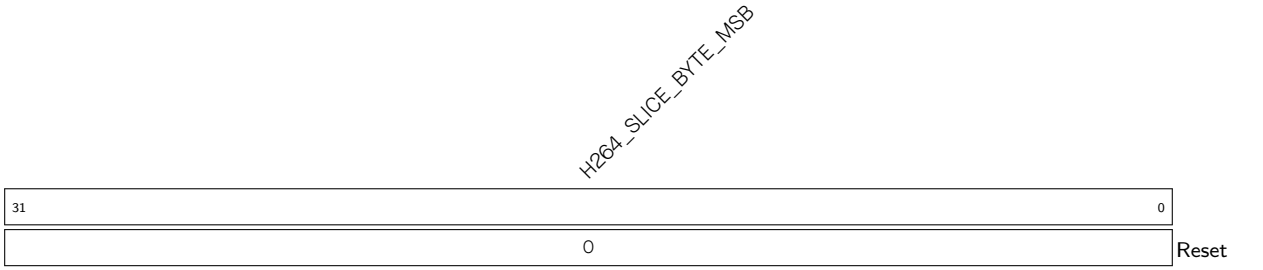
H264_BS_BUFFER_THRESHOLD Configures bitstream buffer overflow threshold. This value should be bigger than the encoded bytes of one 4 × 4 sub MB. (R/W)

Register 41.44. H264_SLICE_HEADER_BYTE0_REG (0x00B8)



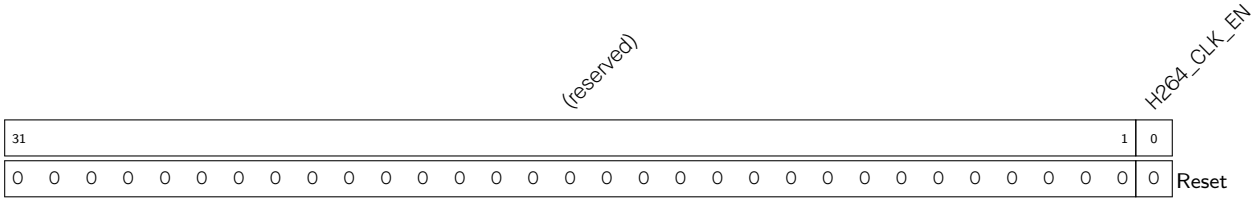
H264_SLICE_BYTE_LSB Configures slice header low 32 bit. (R/W)

Register 41.45. H264_SLICE_HEADER_BYTE1_REG (0x00BC)



H264_SLICE_BYTE_MSB Configures slice header high 32 bit. (R/W)

Register 41.46. H264_CONF_REG (0x00D0)



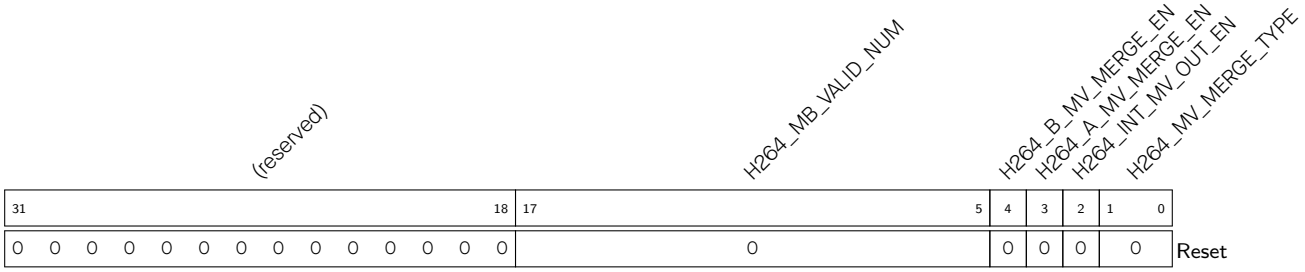
H264_CLK_EN Configures whether to enable the register clock gate.

0: Enable the clock gate only when application writes registers

1: Force enable the clock gate for register

(R/W)

Register 41.47. H264_MV_MERGE_CONFIG_REG (0x00D4)



H264_MV_MERGE_TYPE Configures MV merge type.

0: Merge p16 × 16 mv

1: Merge min mv

2: Merge max mv

3: Invalid. No effect

(R/W)

H264_INT_MV_OUT_EN Configures the output type of the merged MV.

0: Output the merged MV where any part (integer and fractional) is not zero

1: Output the merged MV where the integer part is not zero

(R/W)

H264_A_MV_MERGE_EN Configures whether to enable MV merge for video sequence A.

0: Disable

1: Enable

(R/W)

H264_B_MV_MERGE_EN Configures whether to enable MV merge for video sequence B.

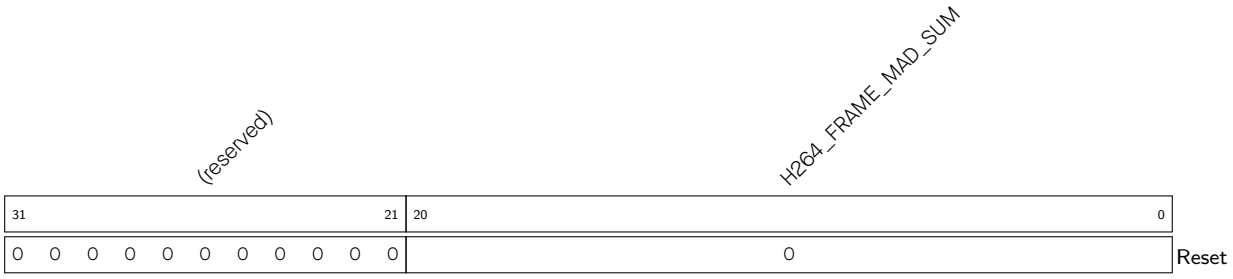
0: Disable

1: Enable

(R/W)

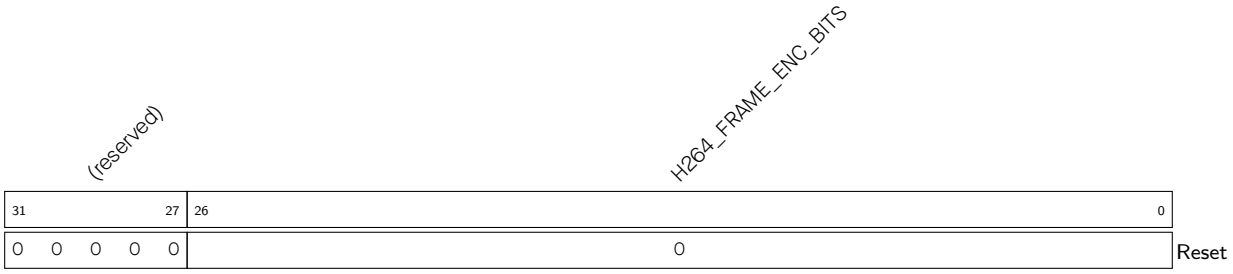
H264_MB_VALID_NUM Represents the number of MBs in the output that contains valid merged MVs. (RO)

Register 41.48. H264_RC_STATUS0_REG (0x00A0)



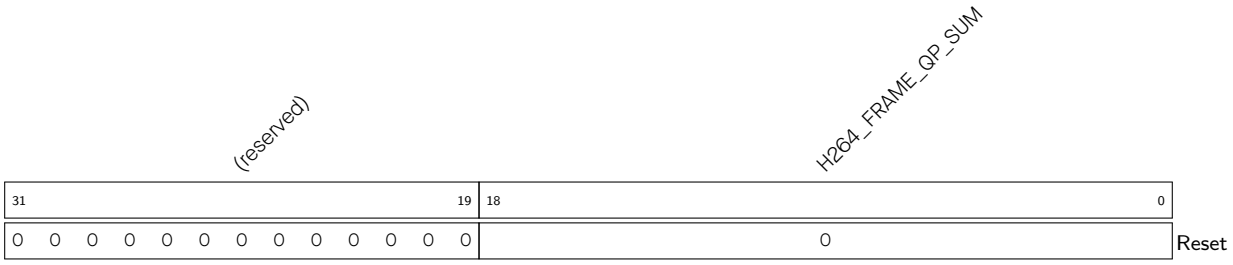
H264_FRAME_MAD_SUM Represents the sum of all MB actual MAD for one frame. (RO)

Register 41.49. H264_RC_STATUS1_REG (0x00A4)



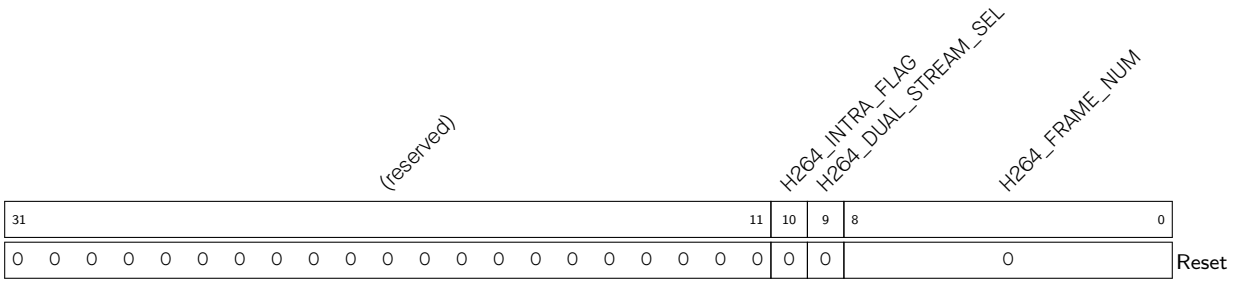
H264_FRAME_ENC_BITS Represents the sum of all MB actual encoding bits for one frame. (RO)

Register 41.50. H264_RC_STATUS2_REG (0x00A8)



H264_FRAME_QP_SUM Represents all MB actual luma QP sum value of one frame. (RO)

Register 41.51. H264_SYS_STATUS_REG (0x00DC)



H264_FRAME_NUM Represents the current frame number. (RO)

H264_DUAL_STREAM_SEL Represents which register group is used for the current frame.

0: Register group A is used

1: Register group B is used

(RO)

H264_INTRA_FLAG Represents the type of the current encoding frame.

0: P frame

1: I frame

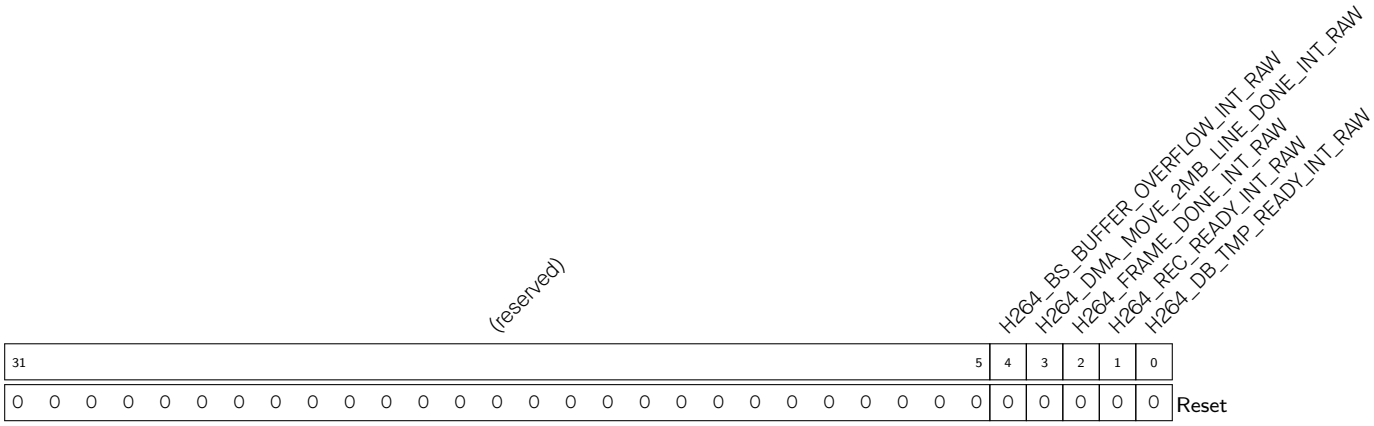
(RO)

Register 41.52. H264_FRAME_CODE_LENGTH_REG (0x00E0)



H264_FRAME_CODE_LENGTH Represents the byte length of the current frame code. (RO)

Register 41.53. H264_INT_RAW_REG (0x00C0)



H264_DB_TMP_READY_INT_RAW The raw interrupt status of H264_DB_TMP_READY_INT. (R/WTC/SS)

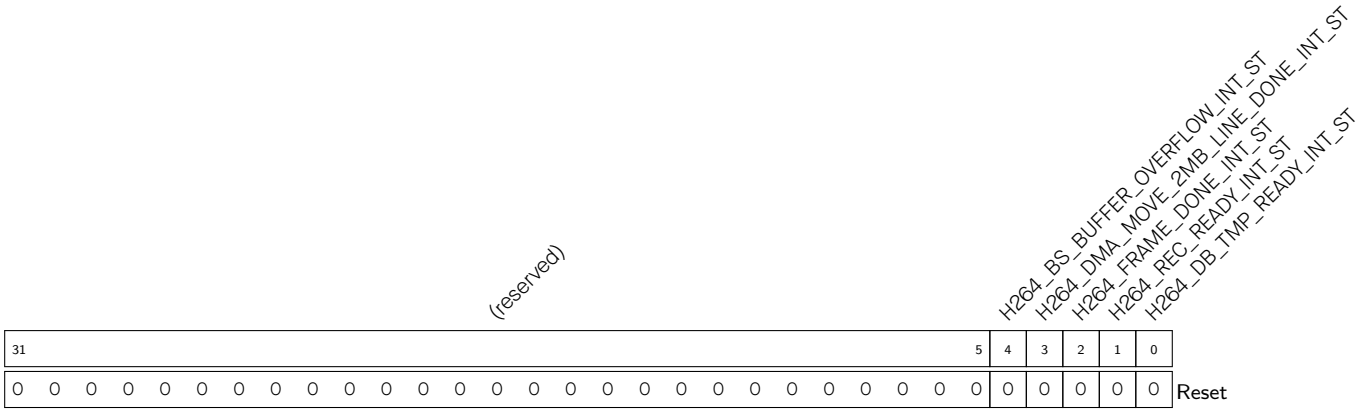
H264_REC_READY_INT_RAW The raw interrupt status of H264_REC_READY_INT. (R/WTC/SS)

H264_FRAME_DONE_INT_RAW The raw interrupt status of H264_FRAME_DONE_INT. (R/WTC/SS)

H264_DMA_MOVE_2MB_LINE_DONE_INT_RAW The raw interrupt status of H264_DMA_MOVE_2MB_LINE_DONE_INT. (R/WTC/SS)

H264_BS_BUFFER_OVERFLOW_INT_RAW The raw interrupt status of H264_BS_BUFFER_OVERFLOW_INT. (R/WTC/SS)

Register 41.54. H264_INT_ST_REG (0x00C4)



H264_DB_TMP_READY_INT_ST The masked interrupt status of H264_DB_TMP_READY_INT. (RO)

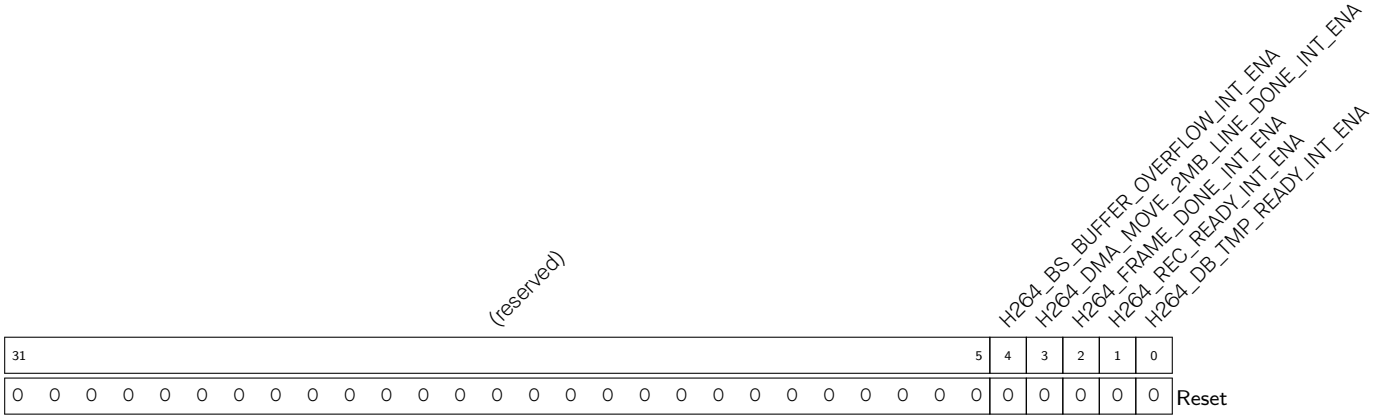
H264_REC_READY_INT_ST The masked interrupt status of H264_REC_READY_INT. (RO)

H264_FRAME_DONE_INT_ST The masked interrupt status of H264_FRAME_DONE_INT. (RO)

H264_DMA_MOVE_2MB_LINE_DONE_INT_ST The masked interrupt status of H264_DMA_MOVE_2MB_LINE_DONE_INT. (RO)

H264_BS_BUFFER_OVERFLOW_INT_ST The masked interrupt status of H264_BS_BUFFER_OVERFLOW_INT. (RO)

Register 41.55. H264_INT_ENA_REG (0x00C8)



H264_DB_TMP_READY_INT_ENA Write 1 to enable H264_DB_TMP_READY_INT. (R/W)

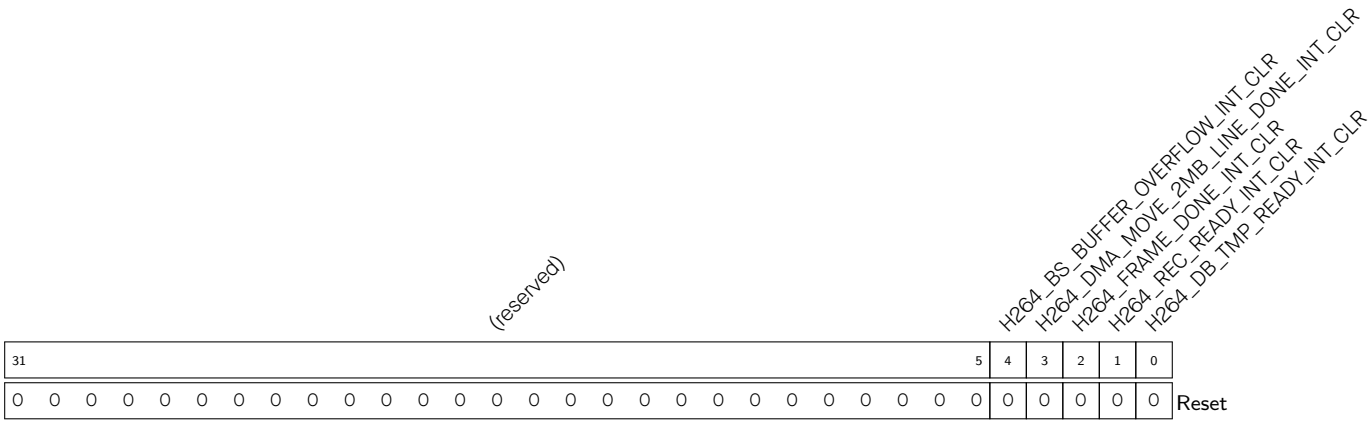
H264_REC_READY_INT_ENA Write 1 to enable H264_REC_READY_INT. (R/W)

H264_FRAME_DONE_INT_ENA Write 1 to enable H264_FRAME_DONE_INT. (R/W)

H264_DMA_MOVE_2MB_LINE_DONE_INT_ENA Write 1 to enable H264_DMA_MOVE_2MB_-LINE_DONE_INT. (R/W)

H264_BS_BUFFER_OVERFLOW_INT_ENA Write 1 to enable H264_BS_BUFFER_OVERFLOW_INT. (R/W)

Register 41.56. H264_INT_CLR_REG (0x00CC)



H264_DB_TMP_READY_INT_CLR Write 1 to clear H264_DB_TMP_READY_INT. (WT)

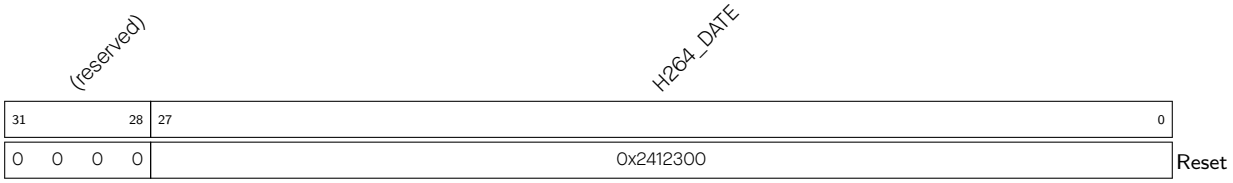
H264_REC_READY_INT_CLR Write 1 to clear H264_REC_READY_INT. (WT)

H264_FRAME_DONE_INT_CLR Write 1 to clear H264_FRAME_DONE_INT. (WT)

H264_DMA_MOVE_2MB_LINE_DONE_INT_CLR Write 1 to clear H264_DMA_MOVE_2MB_LINE_-DONE_INT. (WT)

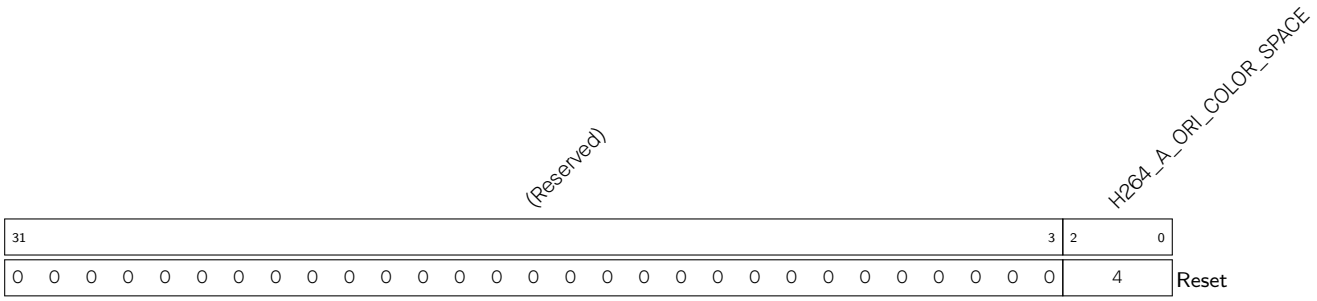
H264_BS_BUFFER_OVERFLOW_INT_CLR Write 1 to clear H264_BS_BUFFER_OVERFLOW_INT. (WT)

Register 41.57. H264_DATE_REG (0x00F0)



H264_DATE Version control register. (R/W)

Register 41.58. H264_A_ORI_CONF_REG (0x00F4)



H264_A_ORI_COLOR_SPACE Configures the original picture color space of video sequence A.

0: RGB888

1: RGB565

2: YUV444

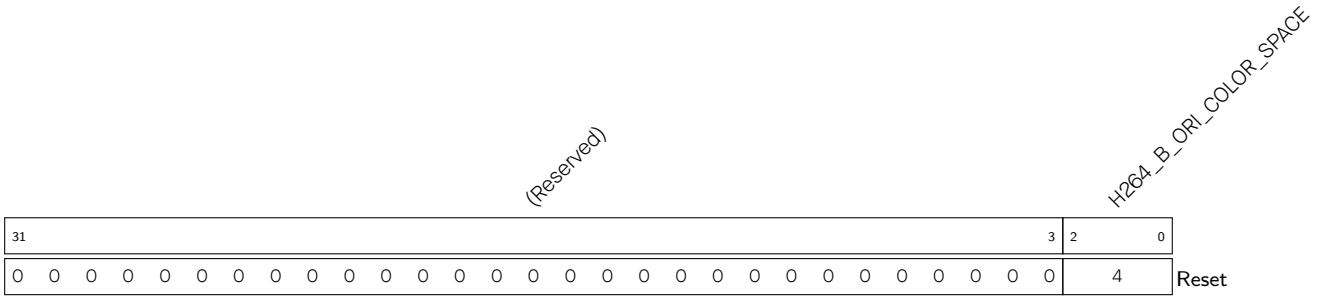
3: YUV422

4: YUV420

Others: Invalid. No effect

(R/W)

Register 41.59. H264_B_ORI_CONF_REG (0x00F8)



H264_B_ORI_COLOR_SPACE Configures the original picture color space of video sequence B.

0: RGB888

1: RGB565

2: YUV444

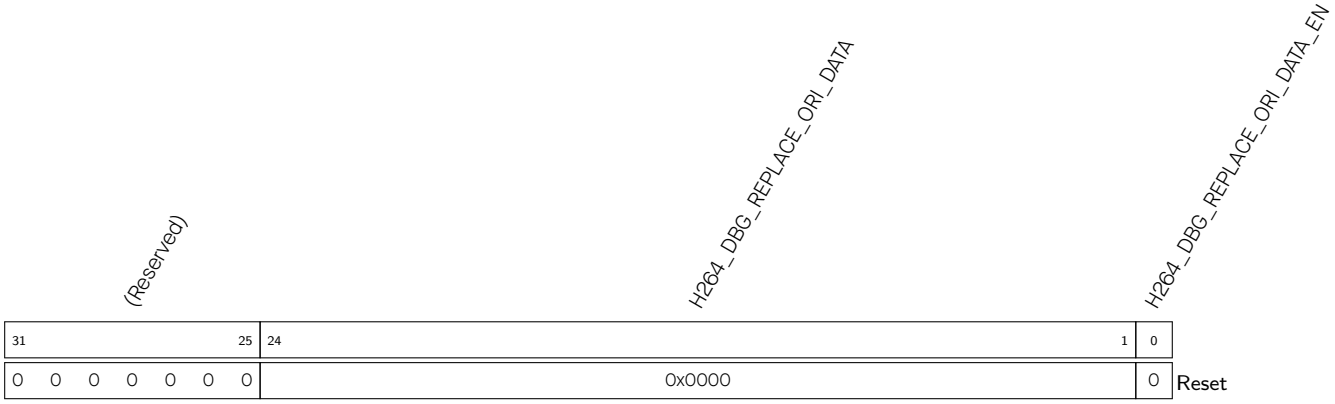
3: YUV422

4: YUV420

Others: Invalid. No effect

(R/W)

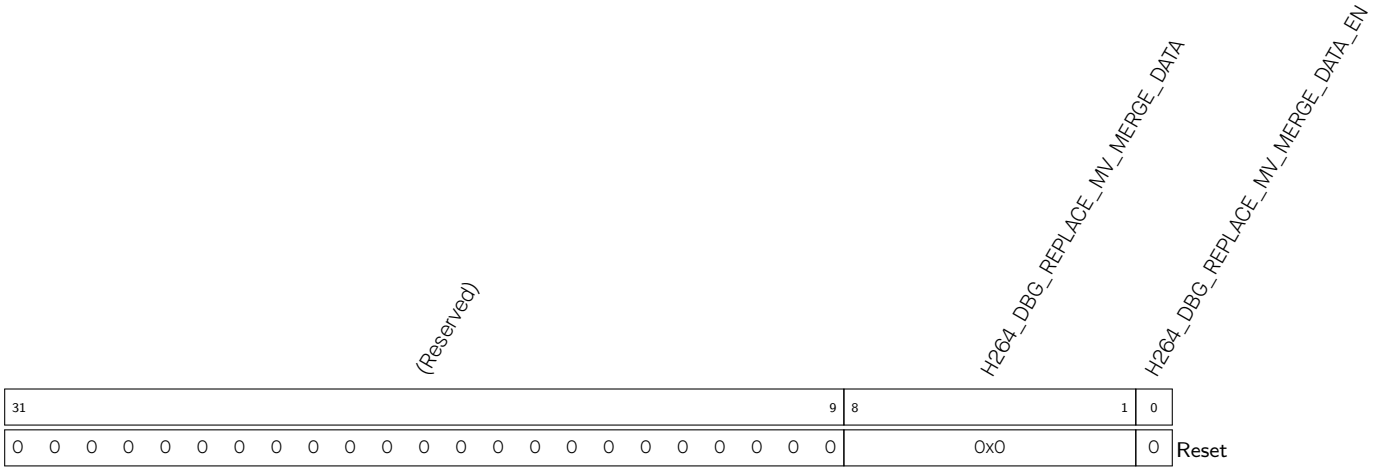
Register 41.60. H264_ORI_DEBUG_CONF_REG (0x00FC)



H264_DBG_REPLACE_ORI_DATA_EN Configures whether to replace the original video pixel data. 0: do not replace, 1: replace. (R/W)

H264_DBG_REPLACE_ORI_DATA Configures the original video pixel data to replace. When the original video color space is RGB, byte0∼2 is BGR. When the color space is YUV, byte0∼2 is VUY. (R/W)

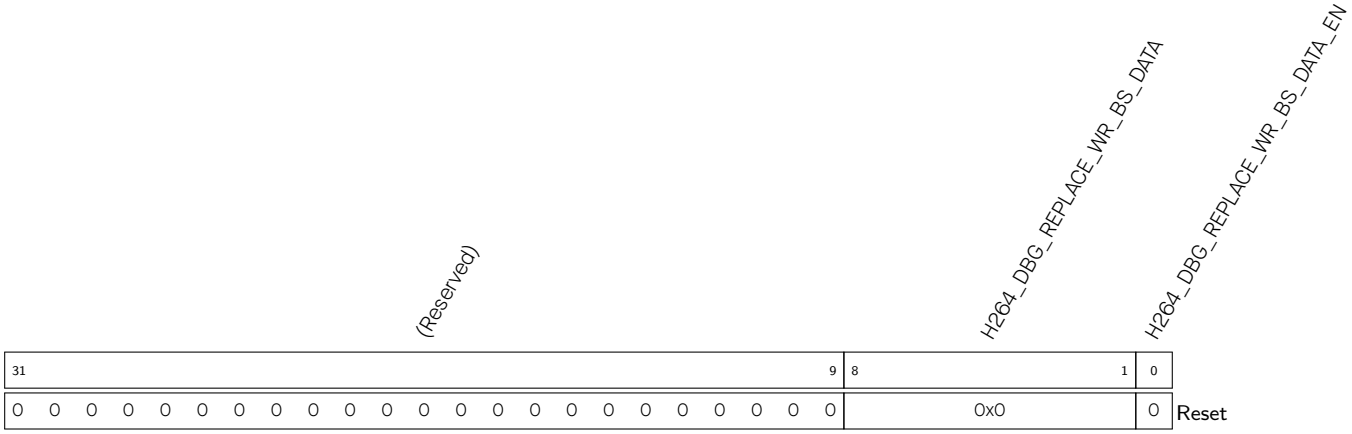
Register 41.61. H264_MV_MERGE_DEBUG_CONF_REG (0x0100)



H264_DBG_REPLACE_MV_MERGE_DATA_EN Configures whether to replace the MV merge data. 0: do not replace, 1: replace. (R/W)

H264_DBG_REPLACE_MV_MERGE_DATA Configures the MV merge data to replace. (R/W)

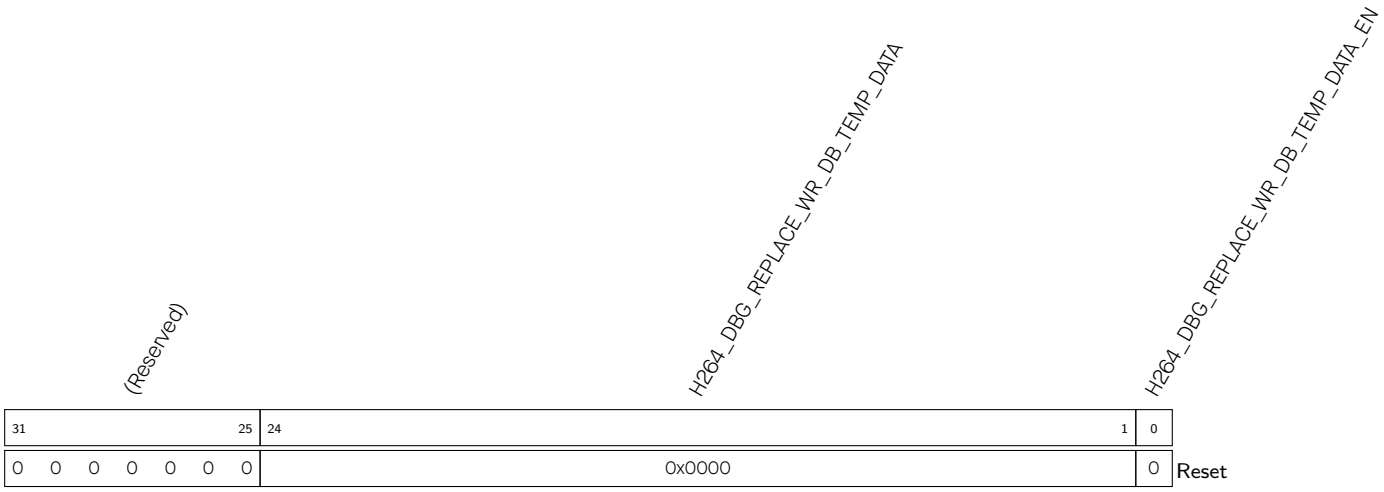
Register 41.62. H264_BS_DEBUG_CONG_REG (0x0104)



H264_DBG_REPLACE_WR_BS_DATA_EN Configures whether to replace the bitstream data. 0: do not replace, 1: replace. (R/W)

H264_DBG_REPLACE_WR_BS_DATA Configures the bitstream data to replace. (R/W)

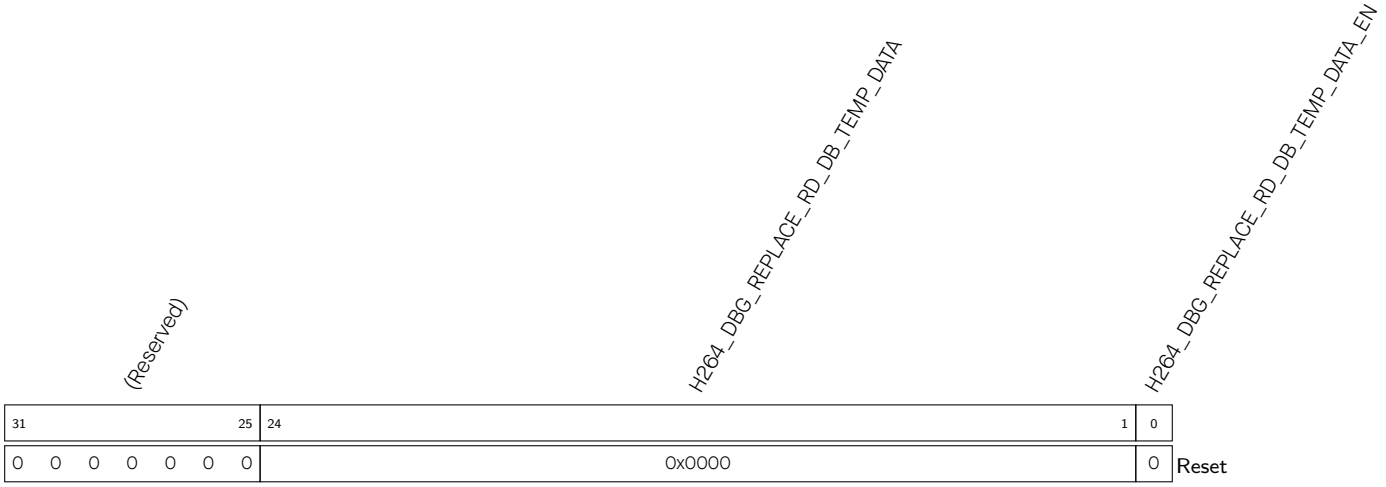
Register 41.63. H264_DB_WR_TEMP_DEBUG_CONG_REG (0x0108)



H264_DBG_REPLACE_WR_DB_TEMP_DATA_EN Configures whether to replace the deblocking filter intermediate data written to H264_DMA. 0: do not replace, 1: replace. (R/W)

H264_DBG_REPLACE_WR_DB_TEMP_DATA Configures the deblocking filter intermediate data to replace when writing. byte0~2 is VUY. (R/W)

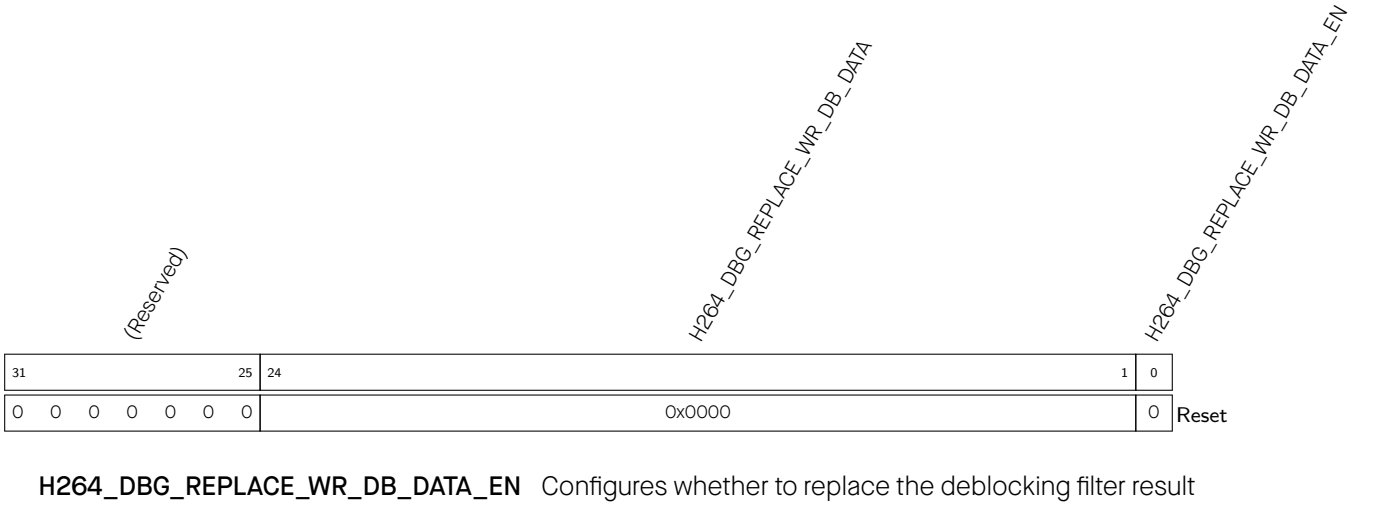
Register 41.64. H264_DB_RD_TEMP_DEBUG_CONG_REG (0x010C)



H264_DBG_REPLACE_RD_DB_TEMP_DATA_EN Configures whether to replace the deblocking filter intermediate data read from H264_DMA. 0: do not replace, 1: replace. (R/W)

H264_DBG_REPLACE_RD_DB_TEMP_DATA Configures the deblocking filter intermediate data to replace when reading. byte0~2 is VUY. (R/W)

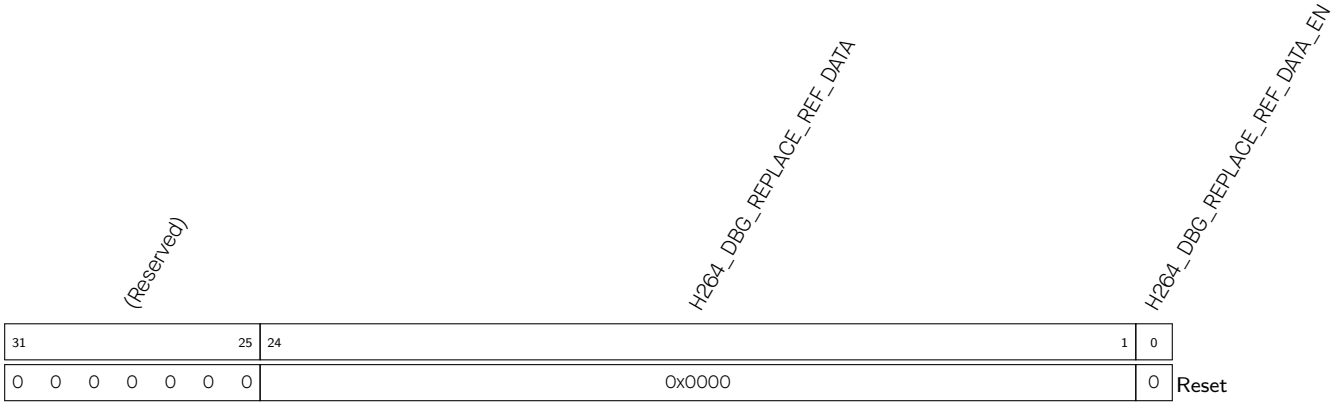
Register 41.65. H264_DB_WR_DEBUG_CONG_REG (0x0110)



H264_DBG_REPLACE_WR_DB_DATA_EN Configures whether to replace the deblocking filter result data. 0: do not replace, 1: replace. (R/W)

H264_DBG_REPLACE_WR_DB_DATA Configures the deblocking filter result data to replace. byte0~2 is VUY. (R/W)

Register 41.66. H264_REF_DEBUG_CONG_REG (0x0114)

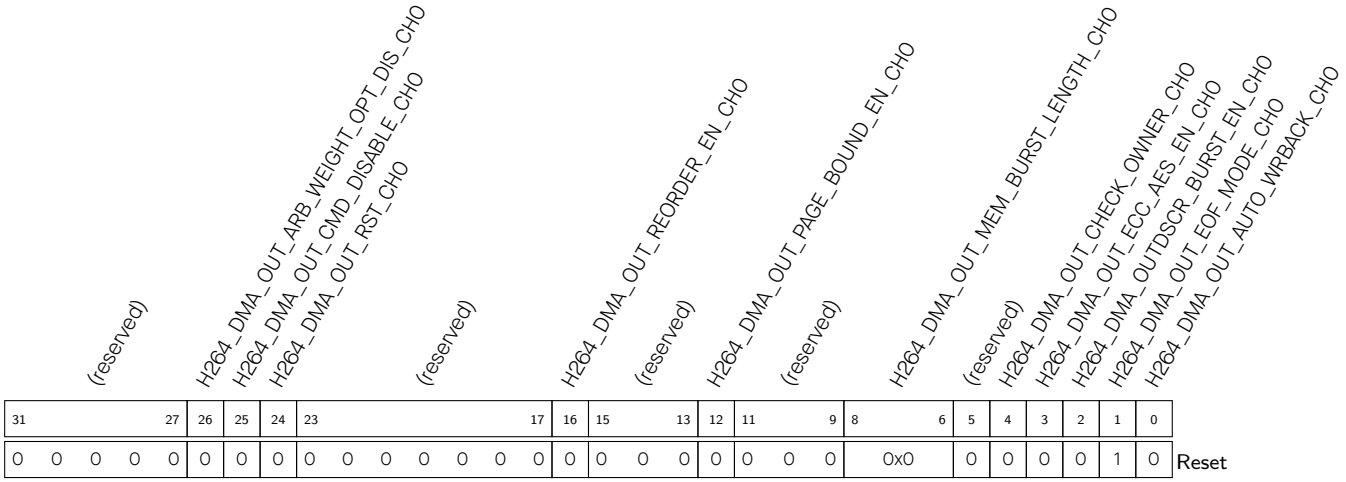


H264_DBG_REPLACE_REF_DATA_EN Configures whether to replace the reference picture pixel data. 0: do not replace, 1: replace. (R/W)

H264_DBG_REPLACE_REF_DATA Configures the reference picture pixel data to replace. byte0∼2 is VUY. (R/W)

41.9.2 H264 DMA Registers

Register 41.67. H264_DMA_OUT_CONF0_CH0_REG (0x0000)



H264_DMA_OUT_AUTO_WRBACK_CH0 Configures whether to enable automatic outlink-writeback when all the data pointed by outlink descriptor has been received.

0: Disable

1: Enable

(R/W)

H264_DMA_OUT_EOF_MODE_CH0 Configures EOF flag generation mode for TX channel 0.

0: EOF flag is generated when last data has been pushed into FIFO in DMA

1: EOF flag is generated when last data has been popped from FIFO in DMA

(R/W)

H264_DMA_OUTDSCR_BURST_EN_CH0 Configures whether to enable INCR burst transfer for TX channel 0 to read descriptors.

0: Disable

1: Enable

(R/W)

H264_DMA_OUT_ECC_AES_EN_CH0 Configures whether DMA can access external memory space for ECC and AES via TX channel 0.

0: Not access

1: Access ECC/AES area. In this case, the start address of the memory region should be 16-byte aligned. The product of the width and the bytes per pixel must also be 16-byte aligned.

(R/W)

H264_DMA_OUT_CHECK_OWNER_CH0 Configures whether to enable owner bit check for TX channel 0.

0: Disable

1: Enable

(R/W)

Continued on the next page...

Register 41.67. H264_DMA_OUT_CONF0_CH0_REG (0x0000)

Continued from the previous page...

H264_DMA_OUT_MEM_BURST_LENGTH_CH0 Configures the burst length for TX channel 0.
0: 8 bytes
1: 16 bytes
2: 32 bytes
3: 64 bytes
4: 128 bytes
5 ~ 7: Invalid. No effect
(R/W)

H264_DMA_OUT_PAGE_BOUND_EN_CH0 Configures whether to make sure AXI read data doesn't cross the address boundary which define by mem_burst_length.
0: AXI read data can cross the address boundary
1: AXI read data doesn't cross the address boundary
(R/W)

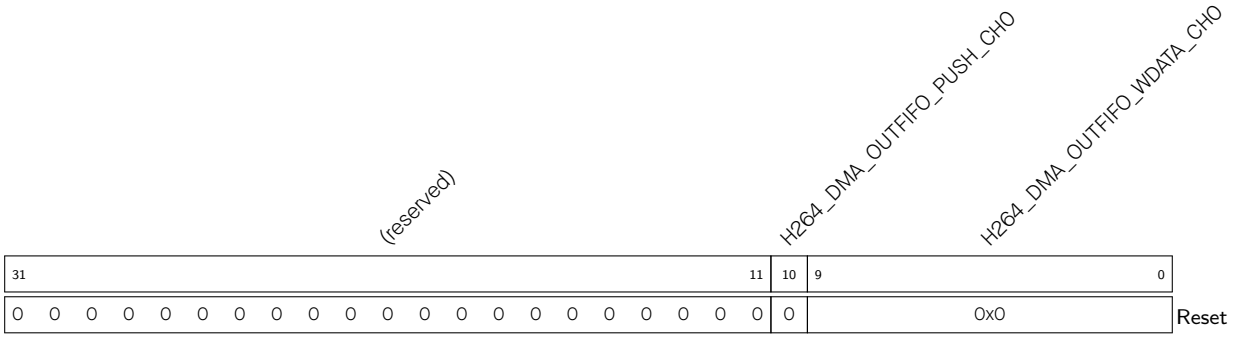
H264_DMA_OUT_REORDER_EN_CH0 Configures whether to enable TX channel 0 macro block reorder.
0: Disable
1: Enable
(R/W)

H264_DMA_OUT_RST_CH0 Write 1 then write 0 to reset TX channel 0. (R/W)

H264_DMA_OUT_CMD_DISABLE_CH0 Configures whether to disable command on TX channel 0.
0: Disable
1: Enable
(R/W)

H264_DMA_OUT_ARB_WEIGHT_OPT_DIS_CH0 Configures whether to disable weight arbitration for TX channel 0.
0: Enable
1: Disable
(R/W)

Register 41.68. H264_DMA_OUT_PUSH_CH0_REG (0x0018)



H264_DMA_OUTFIFO_WDATA_CH0 Represents the data that needs to be pushed into DMA TX FIFO. (R/W)

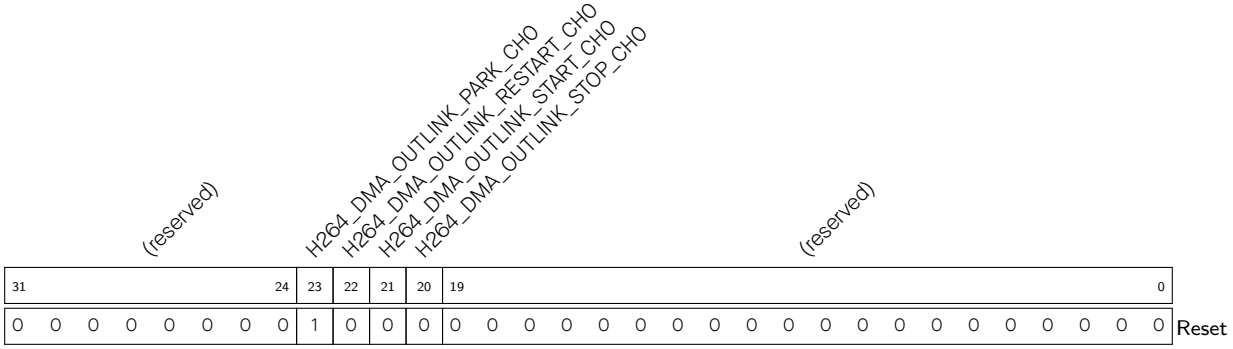
H264_DMA_OUTFIFO_PUSH_CH0 Configures whether to push data into DMA TX FIFO.

0: No effect

1: Push

(R/W/SC)

Register 41.69. H264_DMA_OUT_LINK_CONF_CH0_REG (0x001C)



H264_DMA_OUTLINK_STOP_CH0 Configures whether to stop processing the outlink descriptors.

0: No effect

1: Stop

(R/W/SC)

H264_DMA_OUTLINK_START_CH0 Configures whether to start processing the outlink descriptors.

0: No effect

1: Start

(R/W/SC)

H264_DMA_OUTLINK_RESTART_CH0 Configures whether to restart a new outlink from the last address.

0: No effect

1: Restart

(R/W/SC)

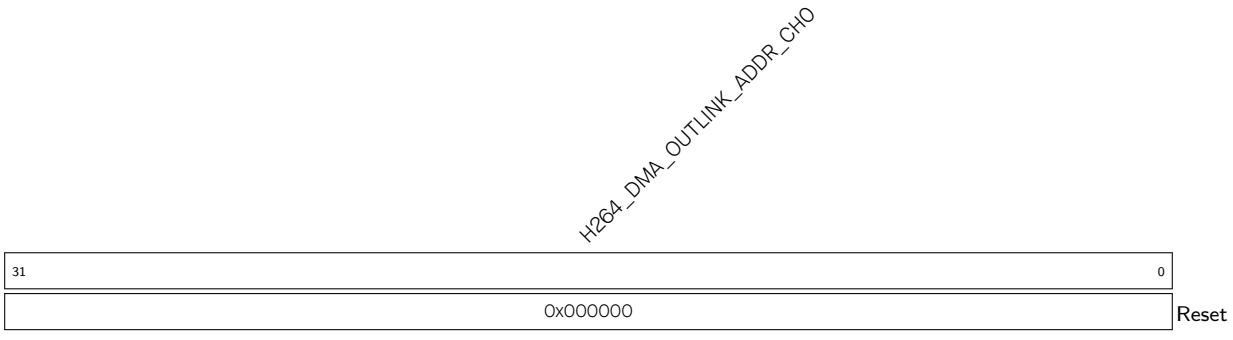
H264_DMA_OUTLINK_PARK_CH0 Represents the status of the transmit descriptor's FSM.

0: The outlink descriptor's FSM is working

1: The outlink descriptor's FSM is in idle state

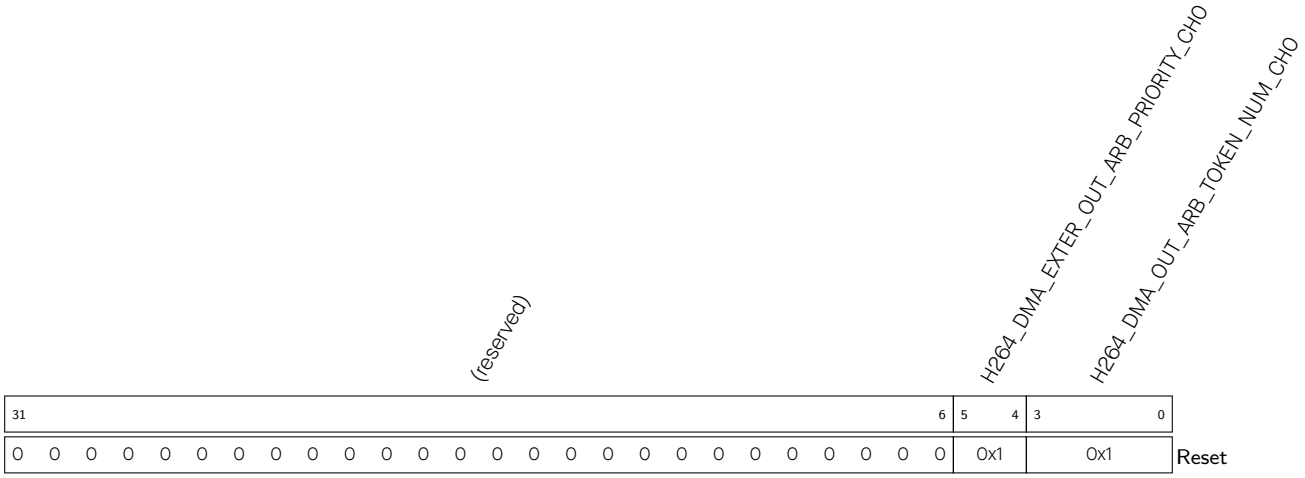
(RO)

Register 41.70. H264_DMA_OUT_LINK_ADDR_CH0_REG (0x0020)



H264_DMA_OUTLINK_ADDR_CH0 Configures the first outlink descriptor's address. (R/W)

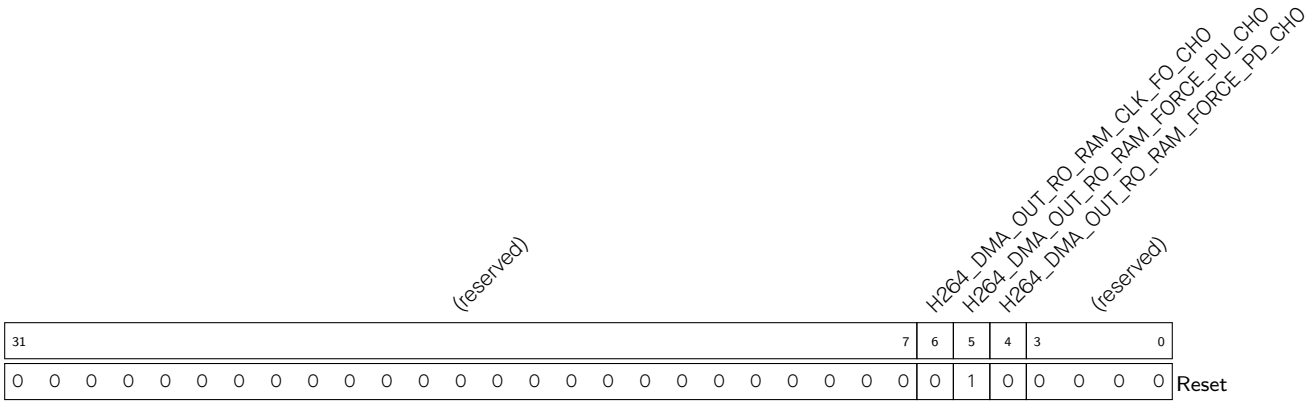
Register 41.71. H264_DMA_OUT_ARB_CH0_REG (0x003C)



H264_DMA_OUT_ARB_TOKEN_NUM_CH0 Configures the maximum token count for the arbiter configuration. (R/W)

H264_DMA_EXTER_OUT_ARB_PRIORITY_CH0 Configures the priority of external memory access for TX channel 0. (R/W)

Register 41.72. H264_DMA_OUT_RO_PD_CONF_CH0_REG (0x0044)



H264_DMA_OUT_RO_RAM_FORCE_PD_CH0 Configures whether to force power down for DMA reorder RAM.

0: Not force power down

1: Force power down

(R/W)

H264_DMA_OUT_RO_RAM_FORCE_PU_CH0 Configures whether to force power up for DMA reorder RAM.

0: Not force power up

1: Force power up

(R/W)

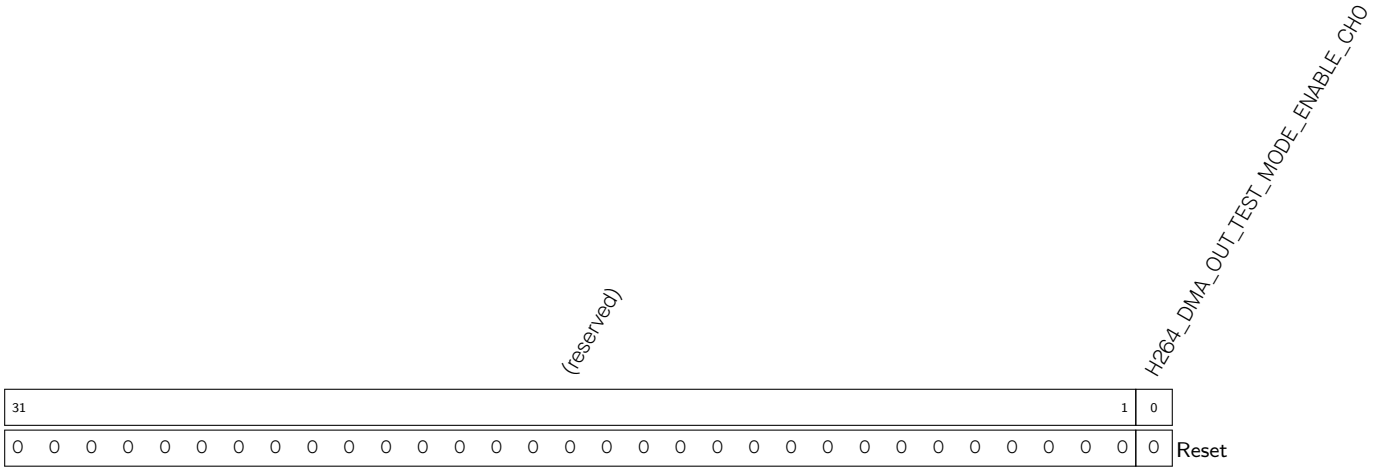
H264_DMA_OUT_RO_RAM_CLK_FO_CH0 Configures whether to force enable clock gating for DMA TX reorder RAM.

0: A clock gating will be used when accessing the RAM in DMA

1: Force to open the clock and bypass the clock gating when accessing the RAM in DMA

(R/W)

Register 41.73. H264_DMA_OUT_MODE_ENABLE_CH0_REG (0x0050)



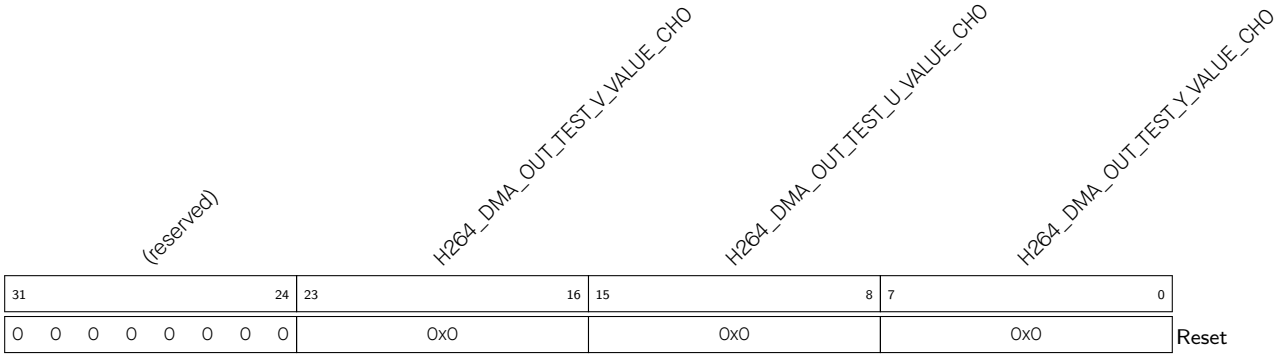
H264_DMA_OUT_TEST_MODE_ENABLE_CH0 Configures whether to enable test mode of TX channel 0.

0: TX channel 0 work in normal mode

1: TX channel 0 work in test mode

(R/W)

Register 41.74. H264_DMA_OUT_MODE_YUV_CH0_REG (0x0054)

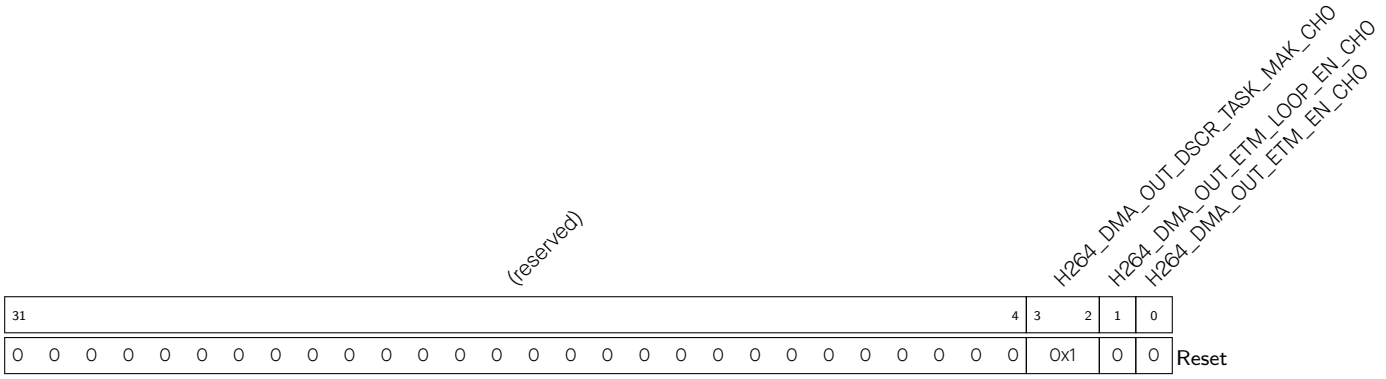


H264_DMA_OUT_TEST_Y_VALUE_CH0 Configures y value in test mode. (R/W)

H264_DMA_OUT_TEST_U_VALUE_CH0 Configures u value in test mode. (R/W)

H264_DMA_OUT_TEST_V_VALUE_CH0 Configures v value in test mode. (R/W)

Register 41.75. H264_DMA_OUT_ETM_CONF_CH0_REG (0x0068)

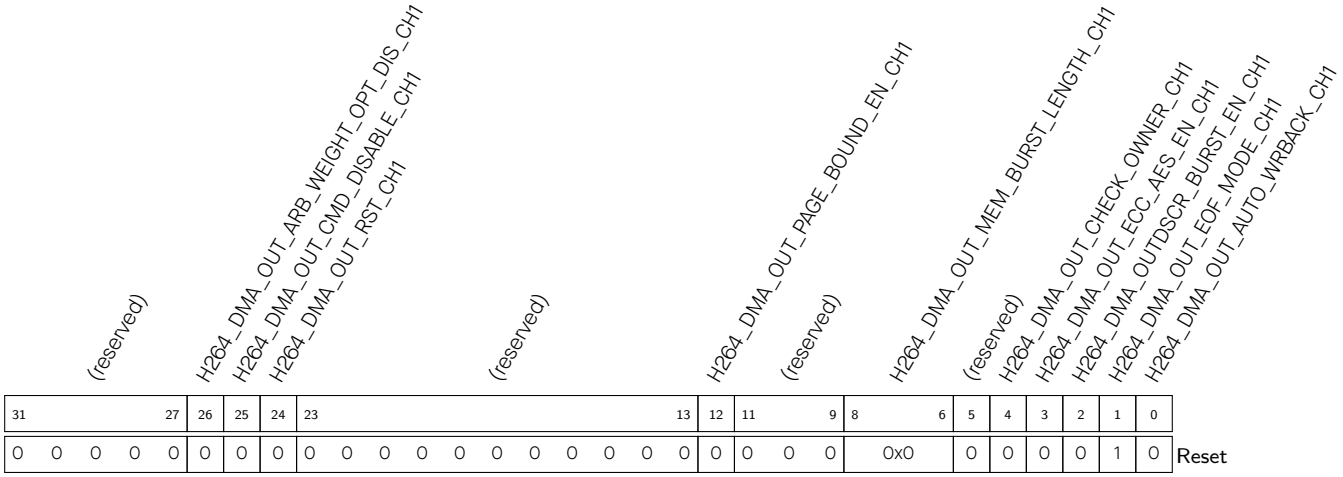


H264_DMA_OUT_ETM_EN_CH0 Set this bit to 1 to enable ETM task function (R/W)

H264_DMA_OUT_ETM_LOOP_EN_CH0 when this bit is 1, dscr can be processed after receiving a task (R/W)

H264_DMA_OUT_DSCR_TASK_MAK_CH0 ETM dscr_ready maximum cache numbers (R/W)

Register 41.76. H264_DMA_OUT_CONF0_CH1_REG (0x0100)


(reserved)
H264_DMA_OUT_ARB_WEIGHT_OPT_DIS_CH1
H264_DMA_OUT_CMD_DISABLE_CH1
H264_DMA_OUT_RST_CH1
(reserved)
H264_DMA_OUT_PAGE_BOUND_EN_CH1
(reserved)
H264_DMA_OUT_MEM_BURST_LENGTH_CH1
(reserved)
H264_DMA_OUT_CHECK_OWNER_CH1
H264_DMA_OUT_ECC_AES_EN_CH1
H264_DMA_OUTDSCR_BURST_EN_CH1
H264_DMA_OUT_EOF_MODE_CH1
H264_DMA_OUT_AUTO_WRBACK_CH1
31 27 26 25 24 23 13 12 11 9 8 6 5 4 3 2 1 0
0 0 0

Register 41.76. H264_DMA_OUT_CONF0_CH1_REG (0x0100)

Continued from the previous page...

H264_DMA_OUT_MEM_BURST_LENGTH_CH1 Configures the burst length for TX channel 1.

0: 8 bytes
1: 16 bytes
2: 32 bytes
3: 64 bytes
4: 128 bytes
5 ~ 7: Invalid. No effect
(R/W)

H264_DMA_OUT_PAGE_BOUND_EN_CH1 Configures whether to make sure AXI read data doesn't cross the address boundary which define by mem_burst_length.

0: AXI read data can cross the address boundary
1: AXI read data doesn't cross the address boundary
(R/W)

H264_DMA_OUT_RST_CH1 Write 1 then write 0 to reset TX channel 1. (R/W)

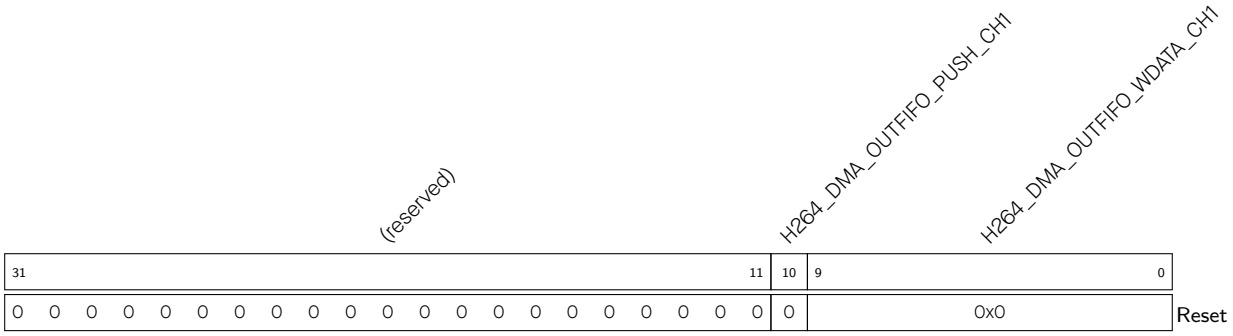
H264_DMA_OUT_CMD_DISABLE_CH1 Configures whether to disable command on TX channel 1.

0: Disable
1: Enable
(R/W)

H264_DMA_OUT_ARB_WEIGHT_OPT_DIS_CH1 Configures whether to disable weight arbitration for TX channel 1.

0: Enable
1: Disable
(R/W)

Register 41.77. H264_DMA_OUT_PUSH_CH1_REG (0x0118)



H264_DMA_OUTFIFO_WDATA_CH1 Represents the data that needs to be pushed into DMA TX FIFO. (R/W)

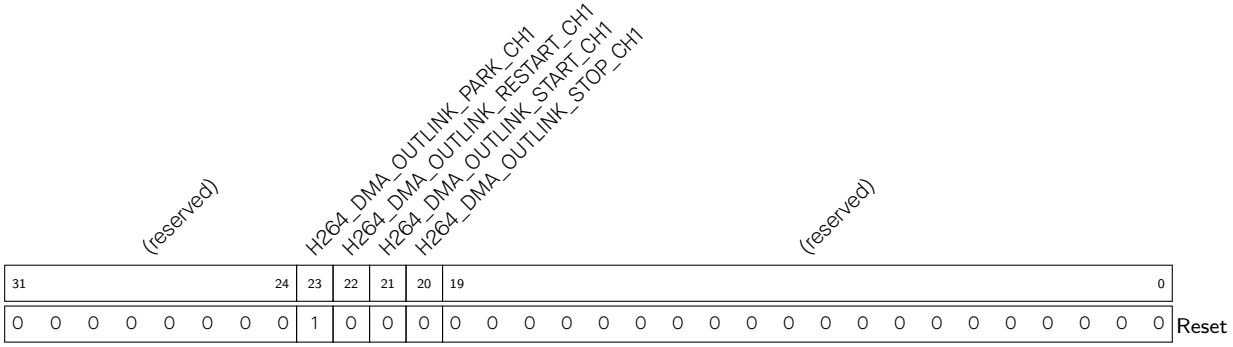
H264_DMA_OUTFIFO_PUSH_CH1 Configures whether to push data into DMA TX FIFO.

0: No effect

1: Push

(R/W/SC)

Register 41.78. H264_DMA_OUT_LINK_CONF_CH1_REG (0x011C)



H264_DMA_OUTLINK_STOP_CH1 Configures whether to stop processing the outlink descriptors.

0: No effect

1: Stop

(R/W/SC)

H264_DMA_OUTLINK_START_CH1 Configures whether to start processing the outlink descriptors.

0: No effect

1: Start

(R/W/SC)

H264_DMA_OUTLINK_RESTART_CH1 Configures whether to restart a new outlink from the last address.

0: No effect

1: Restart

(R/W/SC)

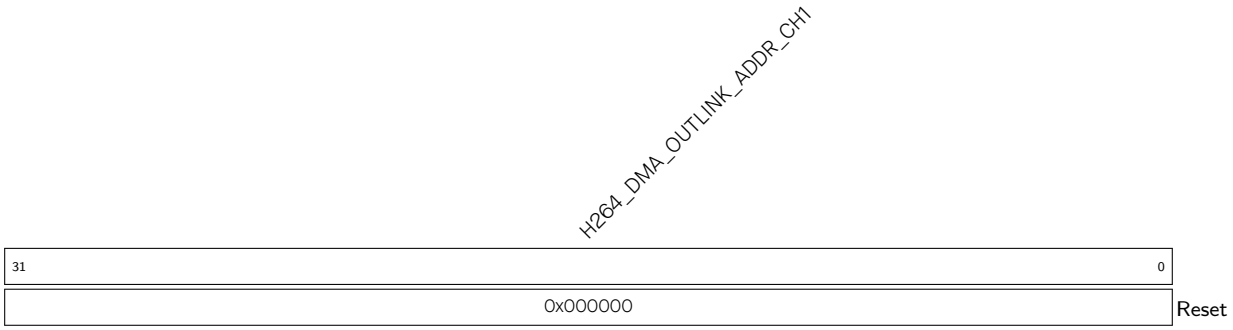
H264_DMA_OUTLINK_PARK_CH1 Represents the status of the transmit descriptor's FSM.

0: The outlink descriptor's FSM is working

1: The outlink descriptor's FSM is in idle state

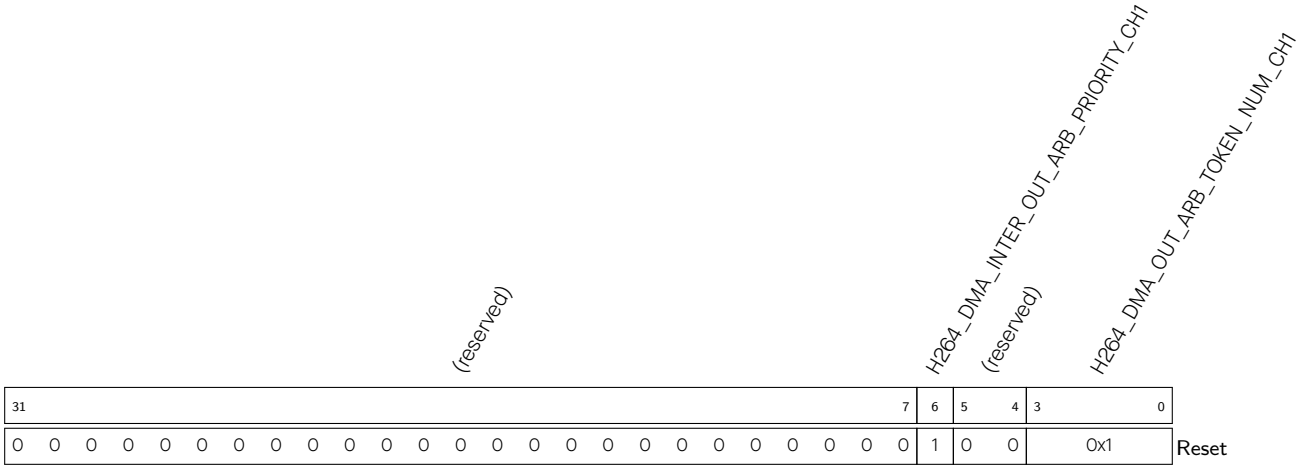
(RO)

Register 41.79. H264_DMA_OUT_LINK_ADDR_CH1_REG (0x0120)



H264_DMA_OUTLINK_ADDR_CH1 Configures the first outlink descriptor's address. (R/W)

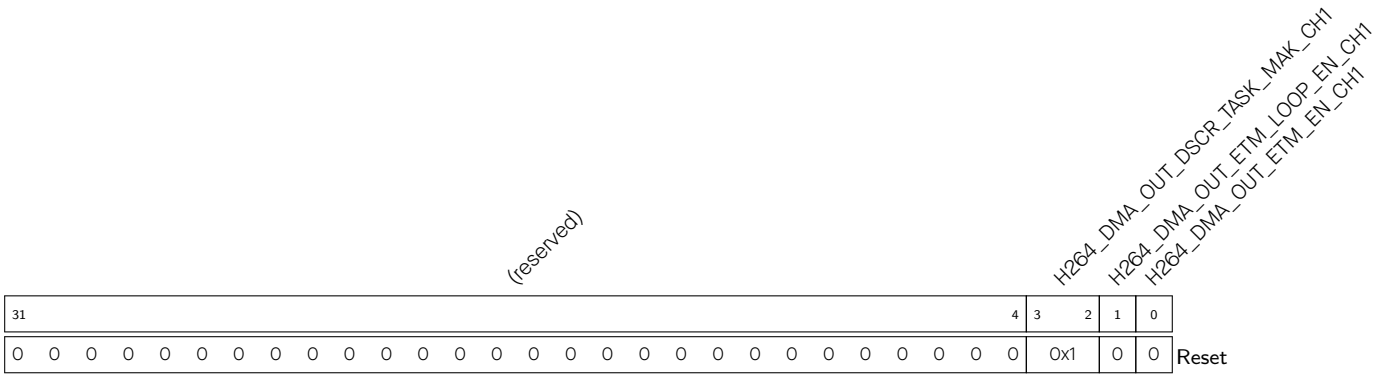
Register 41.80. H264_DMA_OUT_ARB_CH1_REG (0x013C)



H264_DMA_OUT_ARB_TOKEN_NUM_CH1 Configures the maximum token count for the arbiter configuration. (R/W)

H264_DMA_INTER_OUT_ARB_PRIORITY_CH1 Configures the priority of internal memory access for TX channel 1. (R/W)

Register 41.81. H264_DMA_OUT_ETM_CONF_CH1_REG (0x0168)

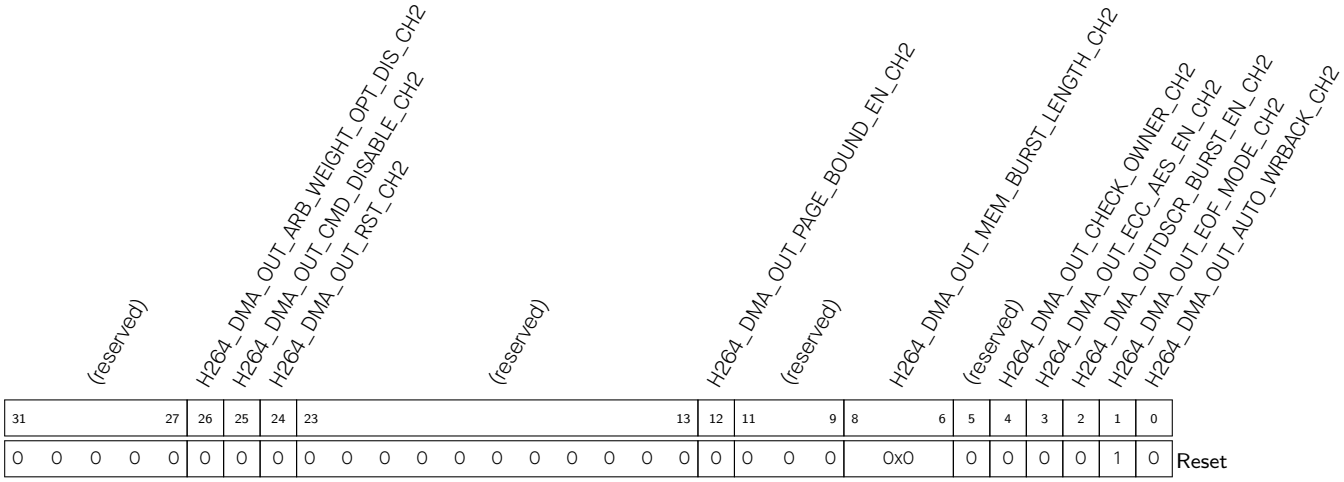


H264_DMA_OUT_ETM_EN_CH1 Set this bit to 1 to enable ETM task function (R/W)

H264_DMA_OUT_ETM_LOOP_EN_CH1 when this bit is 1, dscr can be processed after receiving a task (R/W)

H264_DMA_OUT_DSCR_TASK_MAK_CH1 ETM dscr_ready maximum cache numbers (R/W)

Register 41.82. H264_DMA_OUT_CONF0_CH2_REG (0x0200)


(reserved)
H264_DMA_OUT_ARB_WEIGHT_OPT_DIS_CH2
H264_DMA_OUT_CMD_DISABLE_CH2
H264_DMA_OUT_RST_CH2
(reserved)
H264_DMA_OUT_PAGE_BOUND_EN_CH2
(reserved)
H264_DMA_OUT_MEM_BURST_LENGTH_CH2
(reserved)
H264_DMA_OUT_CHECK_OWNER_CH2
H264_DMA_OUT_ECC_AES_EN_CH2
H264_DMA_OUTDSCR_BURST_EN_CH2
H264_DMA_OUT_EOF_MODE_CH2
H264_DMA_OUT_AUTO_WRBACK_CH2
31 27 26 25 24 23 13 12 11 9 8 6 5 4 3 2 1 0
0 0 0

Register 41.82. H264_DMA_OUT_CONF0_CH2_REG (0x0200)

Continued from the previous page...

H264_DMA_OUT_MEM_BURST_LENGTH_CH2 Configures the burst length for TX channel 2.

0: 8 bytes

1: 16 bytes

2: 32 bytes

3: 64 bytes

4: 128 bytes

5 ~ 7: Invalid. No effect

(R/W)

H264_DMA_OUT_PAGE_BOUND_EN_CH2 Configures whether to make sure AXI read data doesn't cross the address boundary which define by mem_burst_length.

0: AXI read data can cross the address boundary

1: AXI read data doesn't cross the address boundary

(R/W)

H264_DMA_OUT_RST_CH2 Write 1 then write 0 to reset TX channel 2. (R/W)

H264_DMA_OUT_CMD_DISABLE_CH2 Configures whether to disable command on TX channel 2.

0: Disable

1: Enable

(R/W)

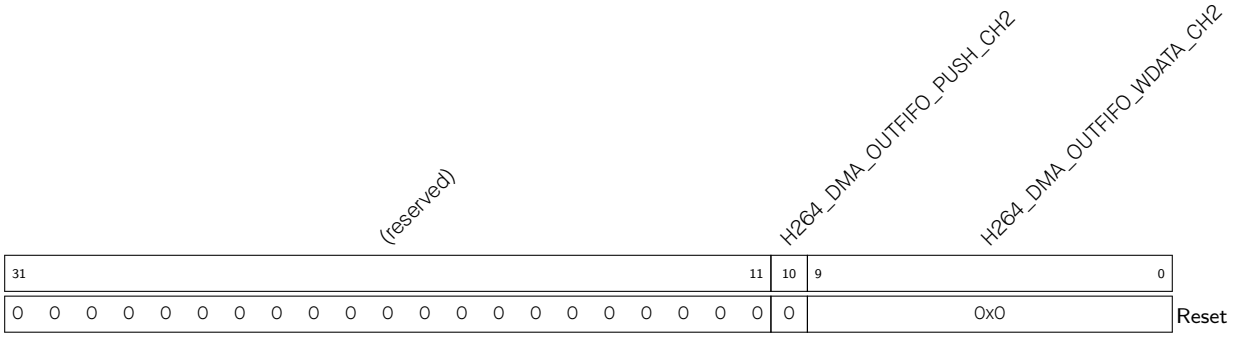
H264_DMA_OUT_ARB_WEIGHT_OPT_DIS_CH2 Configures whether to disable weight arbitration for TX channel 2.

0: Enable

1: Disable

(R/W)

Register 41.83. H264_DMA_OUT_PUSH_CH2_REG (0x0218)



H264_DMA_OUTFIFO_WDATA_CH2 Represents the data that needs to be pushed into DMA TX FIFO. (R/W)

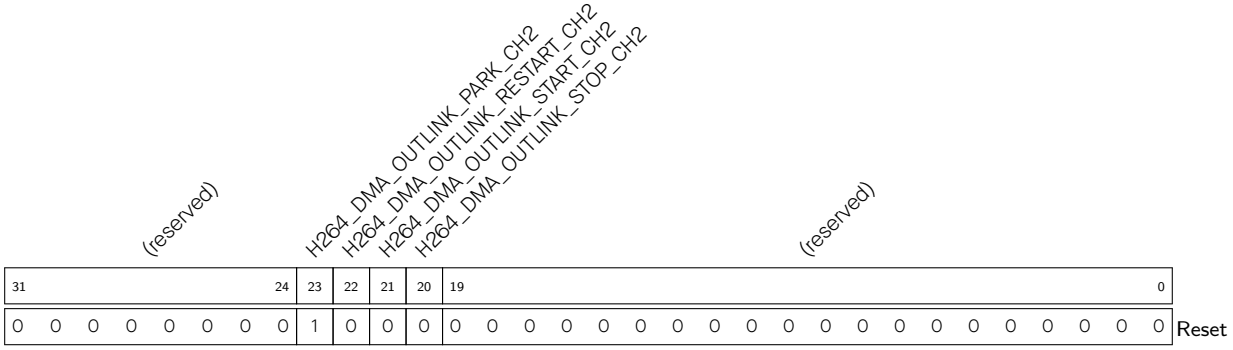
H264_DMA_OUTFIFO_PUSH_CH2 Configures whether to push data into DMA TX FIFO.

0: No effect

1: Push

(R/W/SC)

Register 41.84. H264_DMA_OUT_LINK_CONF_CH2_REG (0x021C)



H264_DMA_OUTLINK_STOP_CH2 Configures whether to stop processing the outlink descriptors.

0: No effect

1: Stop

(R/W/SC)

H264_DMA_OUTLINK_START_CH2 Configures whether to start processing the outlink descriptors.

0: No effect

1: Start

(R/W/SC)

H264_DMA_OUTLINK_RESTART_CH2 Configures whether to restart a new outlink from the last address.

0: No effect

1: Restart

(R/W/SC)

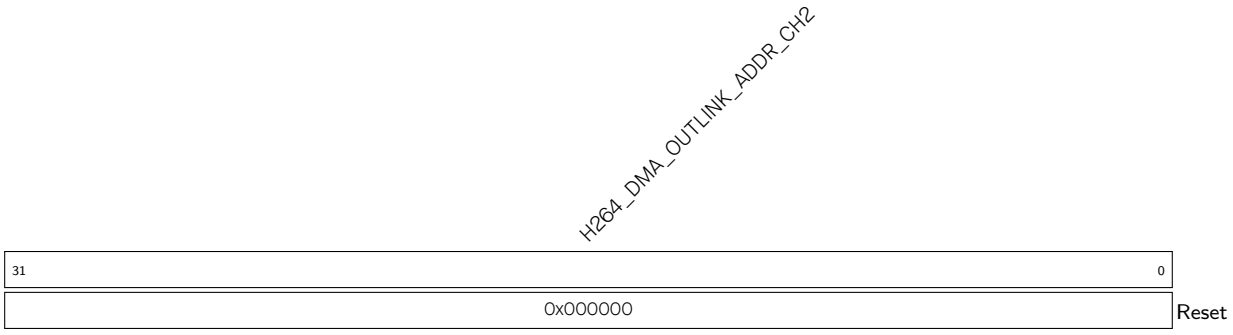
H264_DMA_OUTLINK_PARK_CH2 Represents the status of the transmit descriptor's FSM.

0: The outlink descriptor's FSM is working

1: The outlink descriptor's FSM is in idle state

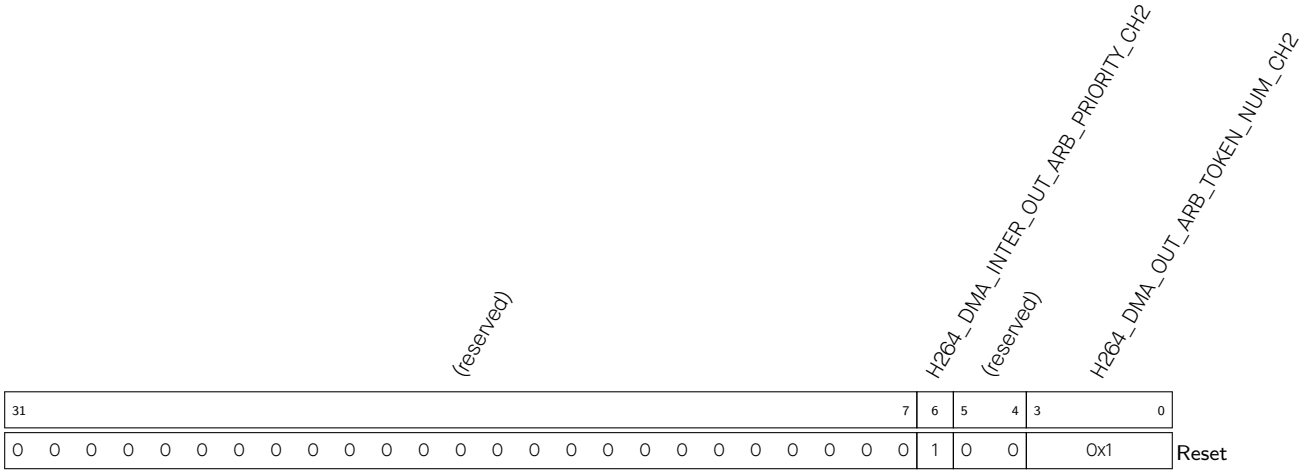
(RO)

Register 41.85. H264_DMA_OUT_LINK_ADDR_CH2_REG (0x0220)



H264_DMA_OUTLINK_ADDR_CH2 Configures the first outlink descriptor's address. (R/W)

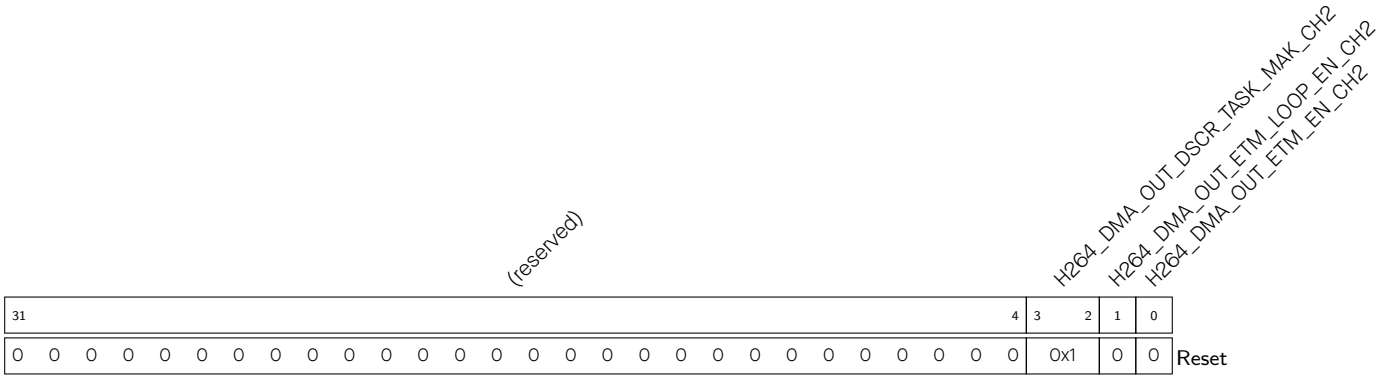
Register 41.86. H264_DMA_OUT_ARB_CH2_REG (0x023C)



H264_DMA_OUT_ARB_TOKEN_NUM_CH2 Configures the maximum token count for the arbiter configuration. (R/W)

H264_DMA_INTER_OUT_ARB_PRIORITY_CH2 Configures the priority of internal memory access for TX channel 2.

Register 41.87. H264_DMA_OUT_ETM_CONF_CH2_REG (0x0268)

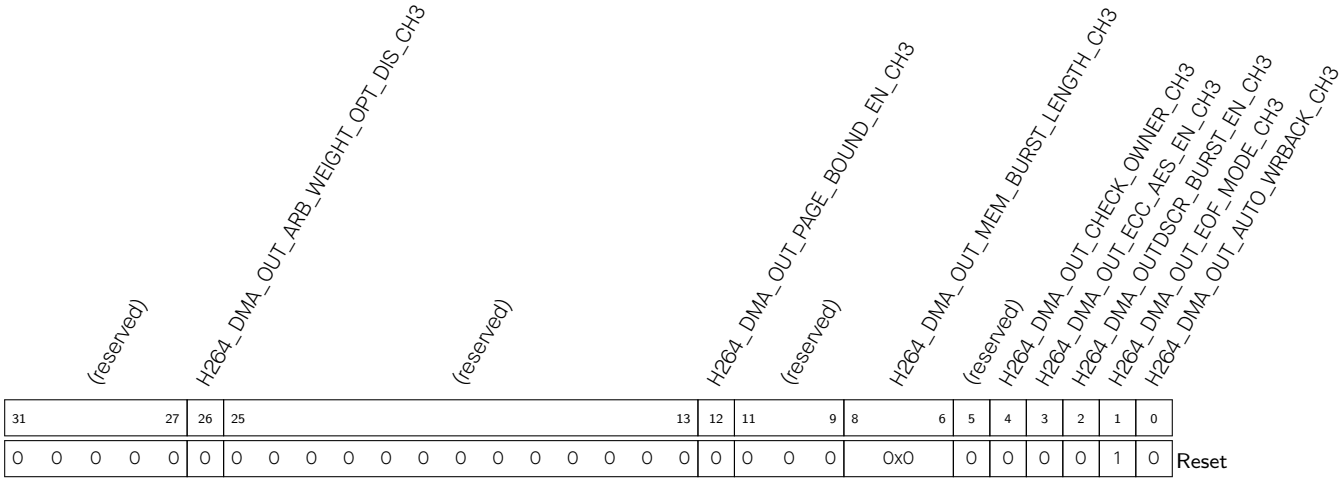


H264_DMA_OUT_ETM_EN_CH2 Set this bit to 1 to enable ETM task function (R/W)

H264_DMA_OUT_ETM_LOOP_EN_CH2 when this bit is 1, dscr can be processed after receiving a task (R/W)

H264_DMA_OUT_DSCR_TASK_MAK_CH2 ETM dscr_ready maximum cache numbers (R/W)

Register 41.88. H264_DMA_OUT_CONF0_CH3_REG (0x0300)


(reserved)
H264_DMA_OUT_ARB_WEIGHT_OPT_DIS_CH3
(reserved)
H264_DMA_OUT_PAGE_BOUND_EN_CH3
(reserved)
H264_DMA_OUT_MEM_BURST_LENGTH_CH3
(reserved)
H264_DMA_OUT_CHECK_OWNER_CH3
H264_DMA_OUT_ECC_AES_EN_CH3
H264_DMA_OUTDSCR_BURST_EN_CH3
H264_DMA_OUT_EOF_MODE_CH3
H264_DMA_OUT_AUTO_WRBACK_CH3
31 27 26 25 13 12 11 9 8 6 5 4 3 2 1 0
0 0 0

Register 41.88. H264_DMA_OUT_CONF0_CH3_REG (0x0300)

Continued from the previous page...

H264_DMA_OUT_MEM_BURST_LENGTH_CH3 Configures the burst length for TX channel 3.

0: 8 bytes

1: 16 bytes

2: 32 bytes

3: 64 bytes

4: 128 bytes

5 ~ 7: Invalid. No effect

(R/W)

H264_DMA_OUT_PAGE_BOUND_EN_CH3 Configures whether to make sure AXI read data doesn't cross the address boundary which define by mem_burst_length.

0: AXI read data can cross the address boundary

1: AXI read data doesn't cross the address boundary

(R/W)

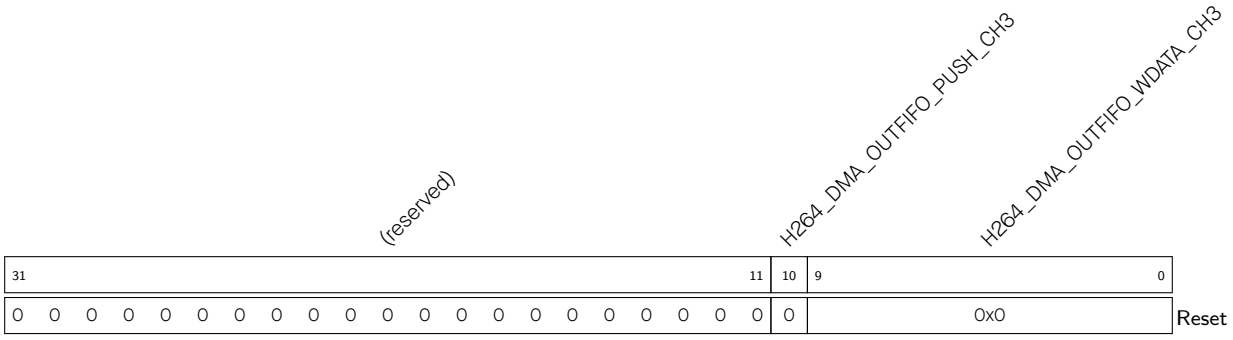
H264_DMA_OUT_ARB_WEIGHT_OPT_DIS_CH3 Configures whether to disable weight arbitration for TX channel 3.

0: Enable

1: Disable

(R/W)

Register 41.89. H264_DMA_OUT_PUSH_CH3_REG (0x0318)



H264_DMA_OUTFIFO_WDATA_CH3 Represents the data that needs to be pushed into DMA TX FIFO. (R/W)

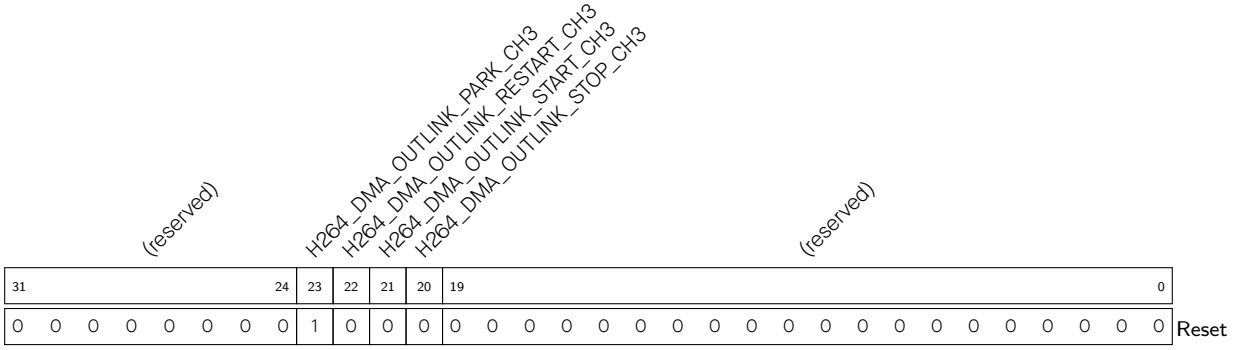
H264_DMA_OUTFIFO_PUSH_CH3 Configures whether to push data into DMA TX FIFO.

0: No effect

1: Push

(R/W/SC)

Register 41.90. H264_DMA_OUT_LINK_CONF_CH3_REG (0x031C)



H264_DMA_OUTLINK_STOP_CH3 Configures whether to stop processing the outlink descriptors.

0: No effect

1: Stop

(R/W/SC)

H264_DMA_OUTLINK_START_CH3 Configures whether to start processing the outlink descriptors.

0: No effect

1: Start

(R/W/SC)

H264_DMA_OUTLINK_RESTART_CH3 Configures whether to restart a new outlink from the last address.

0: No effect

1: Restart

(R/W/SC)

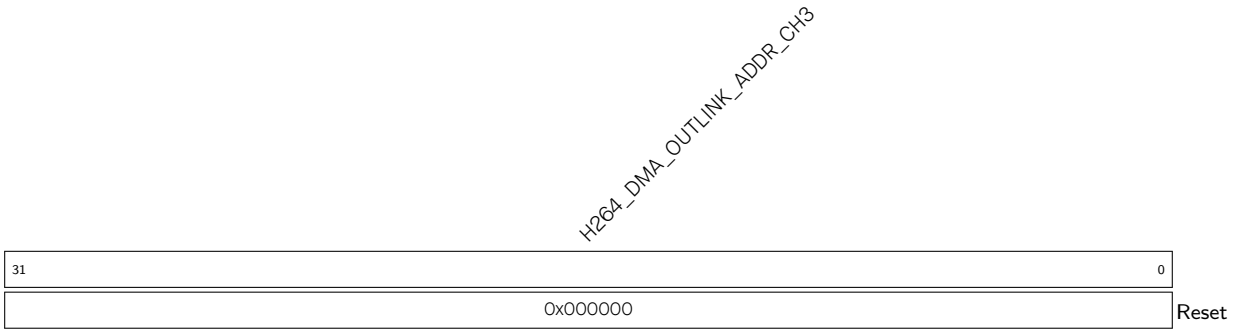
H264_DMA_OUTLINK_PARK_CH3 Represents the status of the transmit descriptor's FSM.

0: The outlink descriptor's FSM is working

1: The outlink descriptor's FSM is in idle state

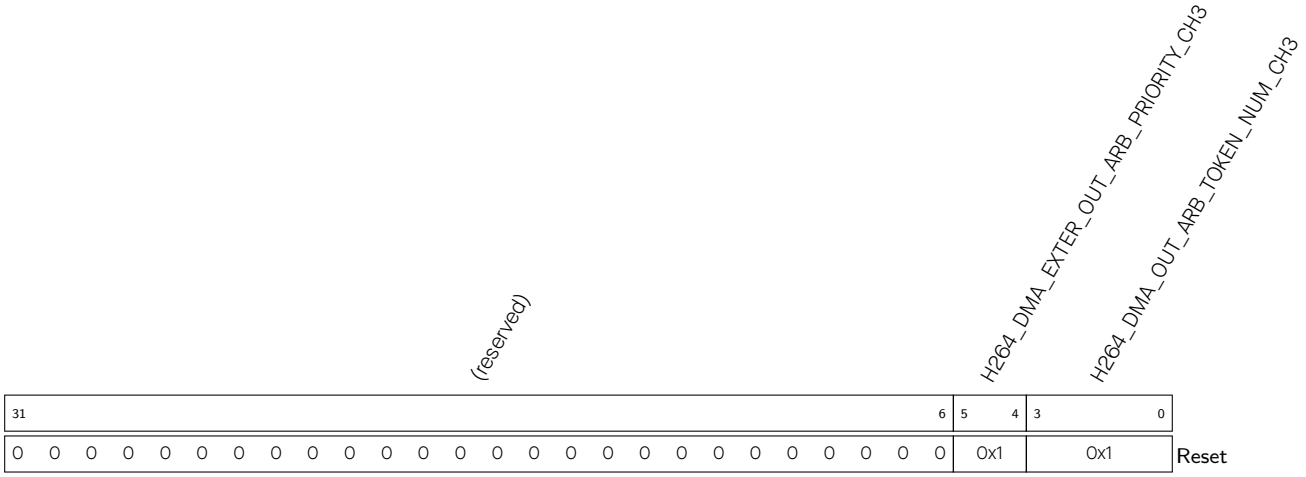
(RO)

Register 41.91. H264_DMA_OUT_LINK_ADDR_CH3_REG (0x0320)



H264_DMA_OUTLINK_ADDR_CH3 Configures the first outlink descriptor's address. (R/W)

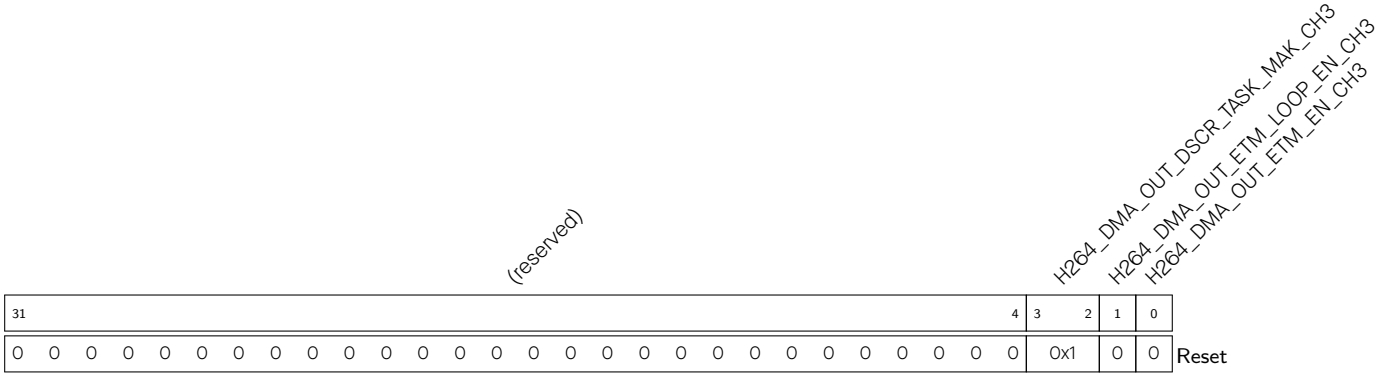
Register 41.92. H264_DMA_OUT_ARB_CH3_REG (0x033C)



H264_DMA_OUT_ARB_TOKEN_NUM_CH3 Configures the maximum token count for the arbiter configuration. (R/W)

H264_DMA_EXTER_OUT_ARB_PRIORITY_CH3 Configures the priority of external memory access for TX channel 3. (R/W)

Register 41.93. H264_DMA_OUT_ETM_CONF_CH3_REG (0x0368)

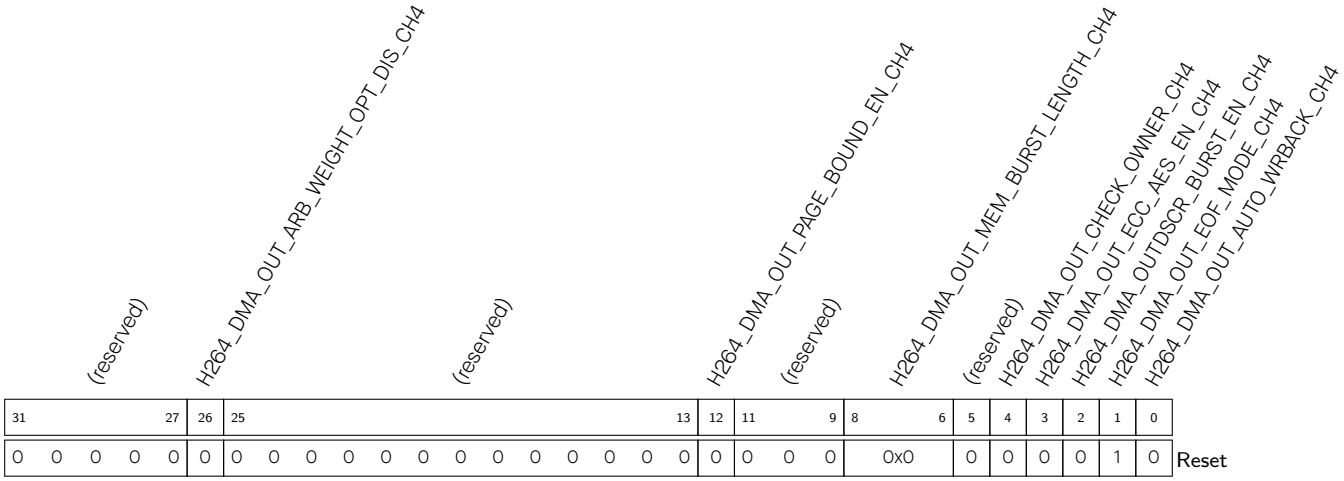


H264_DMA_OUT_ETM_EN_CH3 Set this bit to 1 to enable ETM task function (R/W)

H264_DMA_OUT_ETM_LOOP_EN_CH3 when this bit is 1, dscr can be processed after receiving a task (R/W)

H264_DMA_OUT_DSCR_TASK_MAK_CH3 ETM dscr_ready maximum cache numbers (R/W)

Register 41.94. H264_DMA_OUT_CONF0_CH4_REG (0x0400)



H264_DMA_OUT_AUTO_WRBACK_CH4 Configures whether to enable automatic outlink-writeback when all the data pointed by outlink descriptor has been received.

0: Disable

1: Enable

(R/W)

H264_DMA_OUT_EOF_MODE_CH4 Configures EOF flag generation mode for TX channel 4.

0: EOF flag is generated when last data has been pushed into FIFO in DMA

1: EOF flag is generated when last data has been popped from FIFO in DMA

(R/W)

H264_DMA_OUTDSCR_BURST_EN_CH4 Configures whether to enable INCR burst transfer for TX channel 4 to read descriptors.

0: Disable

1: Enable

(R/W)

H264_DMA_OUT_ECC_AES_EN_CH4 Configures whether DMA can access external memory space for ECC and AES via TX channel 4.

0: Not access

1: Access ECC/AES area. In this case, the start address of the memory region should be 16-byte aligned. The product of the width and the bytes per pixel must also be 16-byte aligned.

(R/W)

H264_DMA_OUT_CHECK_OWNER_CH4 Configures whether to enable owner bit check for TX channel 4.

0: Disable

1: Enable

(R/W)

Continued on the next page...

Register 41.94. H264_DMA_OUT_CONF0_CH4_REG (0x0400)

Continued from the previous page...

H264_DMA_OUT_MEM_BURST_LENGTH_CH4 Configures the burst length for TX channel 4.

0: 8 bytes

1: 16 bytes

2: 32 bytes

3: 64 bytes

4: 128 bytes

5 ~ 7: Invalid. No effect

(R/W)

H264_DMA_OUT_PAGE_BOUND_EN_CH4 Configures whether to make sure AXI read data doesn't cross the address boundary which define by mem_burst_length.

0: AXI read data can cross the address boundary

1: AXI read data doesn't cross the address boundary

(R/W)

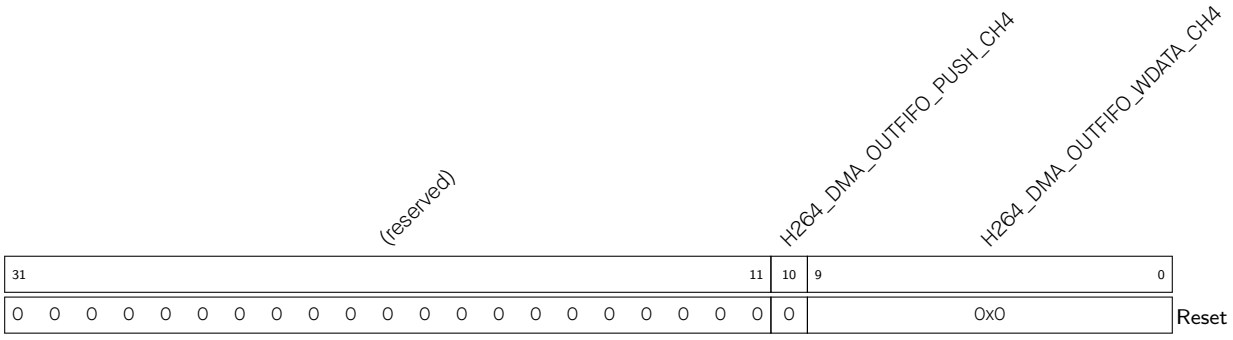
H264_DMA_OUT_ARB_WEIGHT_OPT_DIS_CH4 Configures whether to disable weight arbitration for TX channel 4.

0: Enable

1: Disable

(R/W)

Register 41.95. H264_DMA_OUT_PUSH_CH4_REG (0x0418)



H264_DMA_OUTFIFO_WDATA_CH4 Represents the data that needs to be pushed into DMA TX FIFO. (R/W)

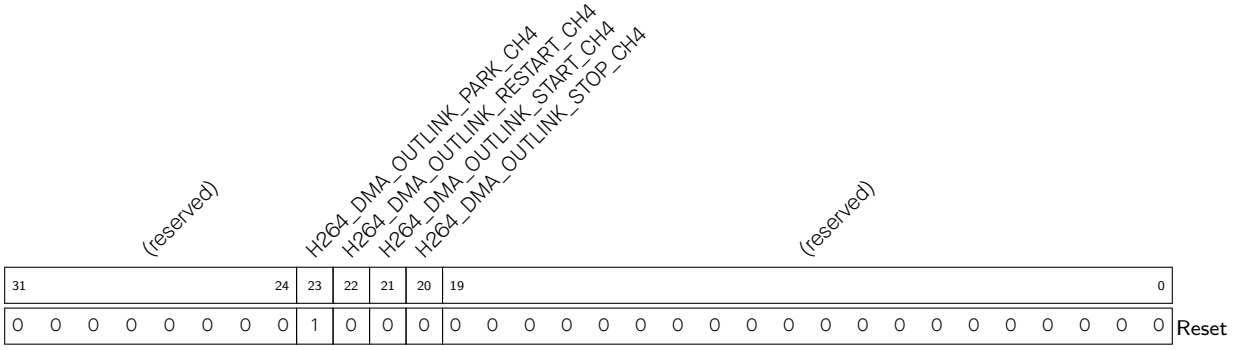
H264_DMA_OUTFIFO_PUSH_CH4 Configures whether to push data into DMA TX FIFO.

0: No effect

1: Push

(R/W/SC)

Register 41.96. H264_DMA_OUT_LINK_CONF_CH4_REG (0x041C)



H264_DMA_OUTLINK_STOP_CH4 Configures whether to stop processing the outlink descriptors.

0: No effect

1: Stop

(R/W/SC)

H264_DMA_OUTLINK_START_CH4 Configures whether to start processing the outlink descriptors.

0: No effect

1: Start

(R/W/SC)

H264_DMA_OUTLINK_RESTART_CH4 Configures whether to restart a new outlink from the last address.

0: No effect

1: Restart

(R/W/SC)

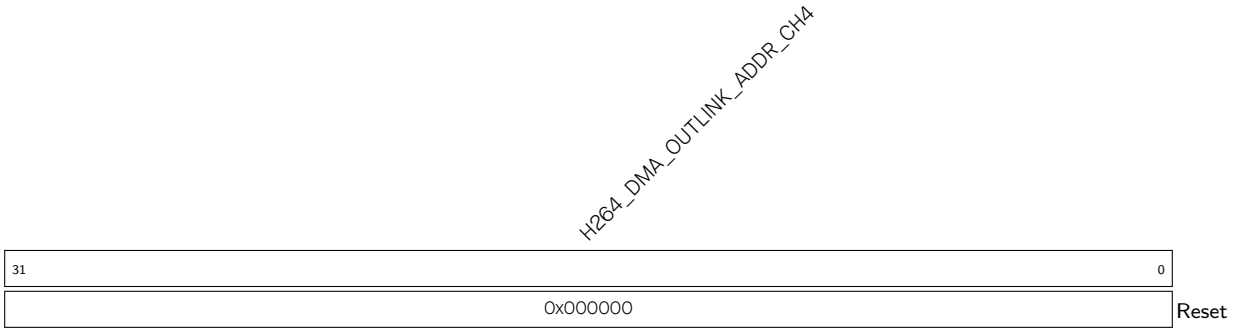
H264_DMA_OUTLINK_PARK_CH4 Represents the status of the transmit descriptor's FSM.

0: The outlink descriptor's FSM is working

1: The outlink descriptor's FSM is in idle state

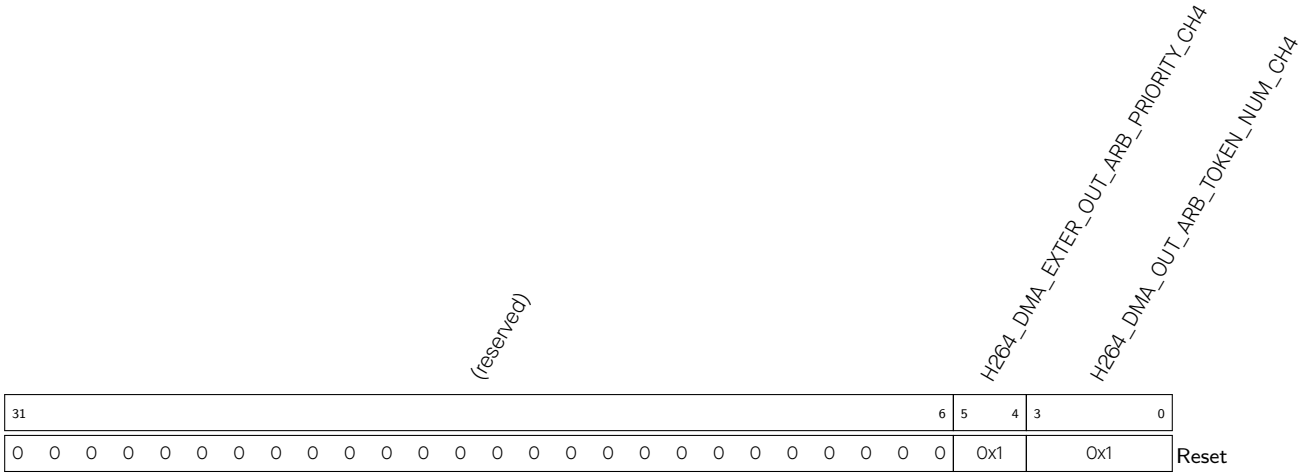
(RO)

Register 41.97. H264_DMA_OUT_LINK_ADDR_CH4_REG (0x0420)



H264_DMA_OUTLINK_ADDR_CH4 Configures the first outlink descriptor's address. (R/W)

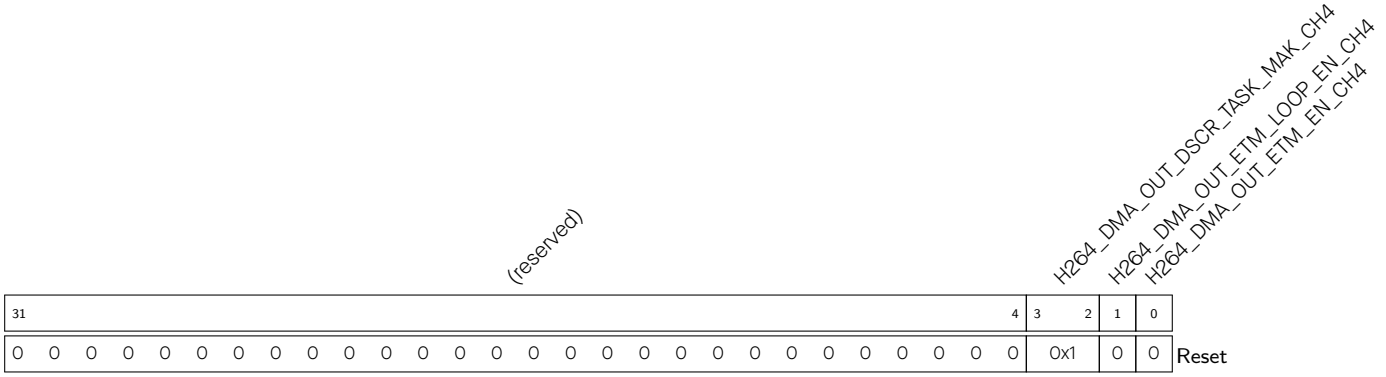
Register 41.98. H264_DMA_OUT_ARB_CH4_REG (0x043C)



H264_DMA_OUT_ARB_TOKEN_NUM_CH4 Configures the maximum token count for the arbiter configuration. (R/W)

H264_DMA_EXTER_OUT_ARB_PRIORITY_CH4 Configures the priority of external memory access for TX channel 4. (R/W)

Register 41.99. H264_DMA_OUT_ETM_CONF_CH4_REG (0x0468)

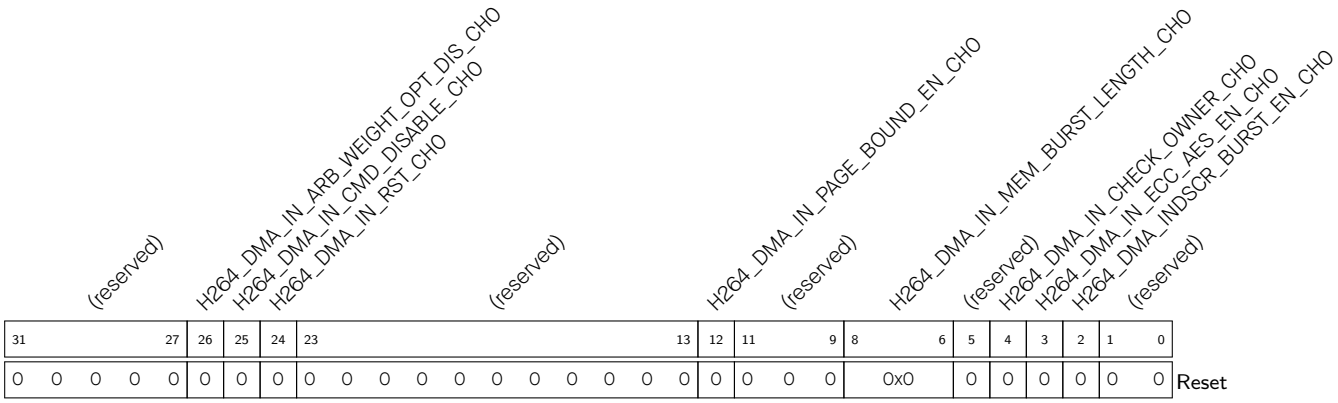


H264_DMA_OUT_ETM_EN_CH4 Set this bit to 1 to enable ETM task function (R/W)

H264_DMA_OUT_ETM_LOOP_EN_CH4 when this bit is 1, dscr can be processed after receiving a task (R/W)

H264_DMA_OUT_DSCR_TASK_MAK_CH4 ETM dscr_ready maximum cache numbers (R/W)

Register 41.100. H264_DMA_IN_CONF0_CH0_REG (0x0500)


(reserved)
H264_DMA_IN_ARB_WEIGHT_OPT_DIS_CH0
H264_DMA_IN_CMD_DISABLE_CH0
H264_DMA_IN_RST_CH0
(reserved)
H264_DMA_IN_PAGE_BOUND_EN_CH0
(reserved)
H264_DMA_IN_MEM_BURST_LENGTH_CH0
(reserved)
H264_DMA_IN_CHECK_OWNER_CH0
H264_DMA_IN_ECC_AES_EN_CH0
H264_DMA_INDSCR_BURST_EN_CH0
(reserved)
31 27 26 25 24 23 13 12 11 9 8 6 5 4 3 2 1 0
0 0 0

Register 41.100. H264_DMA_IN_CONF0_CH0_REG (0x0500)

Continued from the previous page...

H264_DMA_IN_CMD_DISABLE_CH0 Configures whether to disable command on RX channel 0.

0: Disable

1: Enable

(R/W)

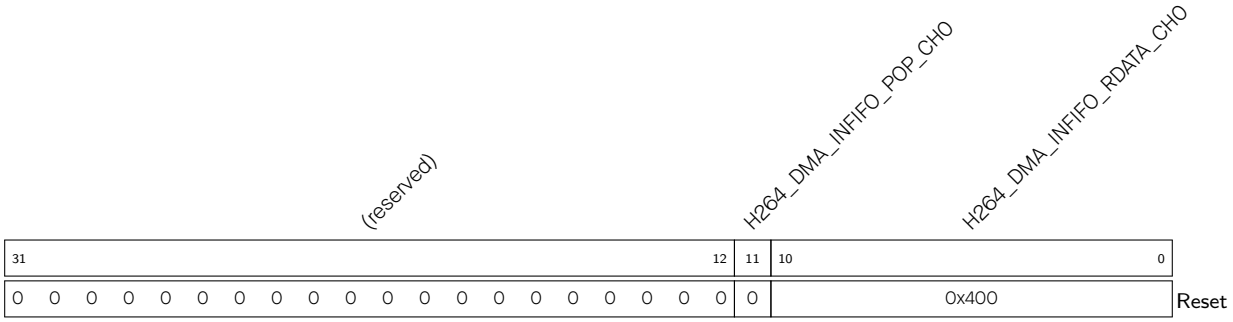
H264_DMA_IN_ARB_WEIGHT_OPT_DIS_CH0 Configures whether to disable weight arbitration for RX channel 0.

0: Enable

1: Disable

(R/W)

Register 41.101. H264_DMA_IN_POP_CH0_REG (0x0518)



H264_DMA_INFIFO_RDATA_CH0 Represents the data popped from DMA RX FIFO. (RO)

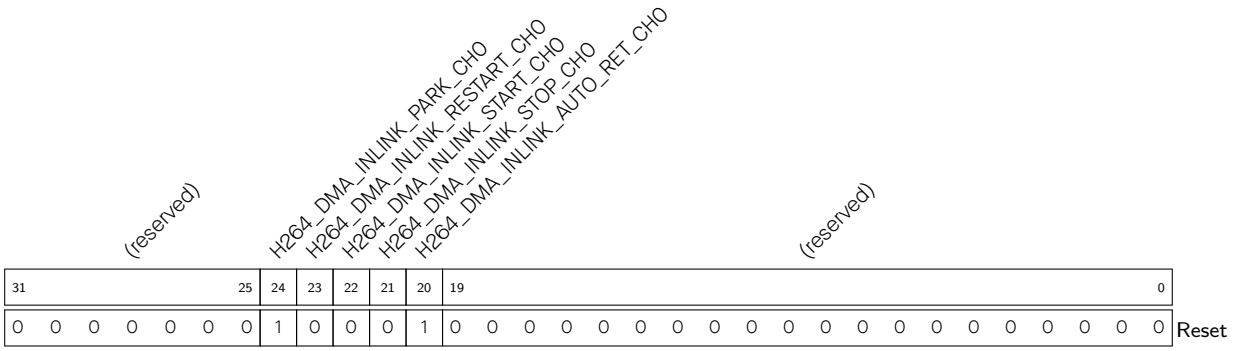
H264_DMA_INFIFO_POP_CH0 Configures whether to pop data from DMA RX FIFO.

0: No effect

1: Pop

(R/W/SC)

Register 41.102. H264_DMA_IN_LINK_CONF_CH0_REG (0x051C)



H264_DMA_INLINK_AUTO_RET_CH0 Configures whether to return to the address of the current receive descriptor when there are errors during data reception.

0: Not return

1: Return

(R/W)

H264_DMA_INLINK_STOP_CH0 Configures whether to stop processing the inlink descriptors.

0: No effect

1: Stop

(R/W/SC)

H264_DMA_INLINK_START_CH0 Configures whether to start processing the inlink descriptors.

0: No effect

1: Start

(R/W/SC)

H264_DMA_INLINK_RESTART_CH0 Configures whether to restart a new inlink from the last address.

0: No effect

1: Restart

(R/W/SC)

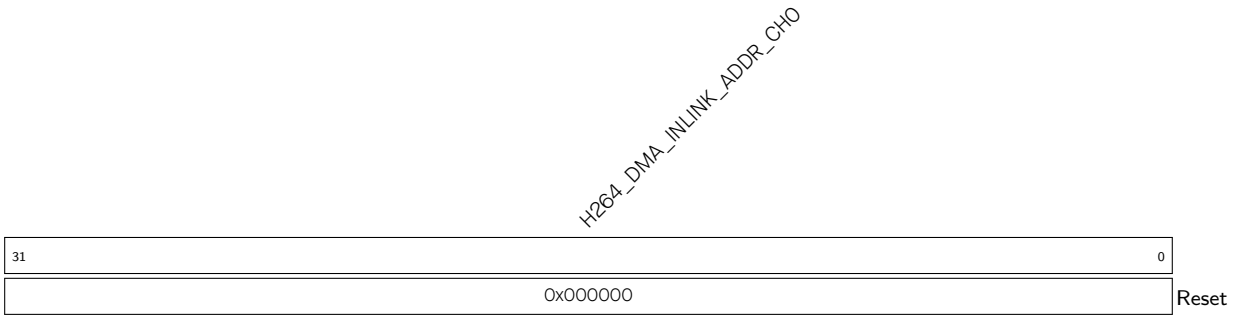
H264_DMA_INLINK_PARK_CH0 Represents the status of the inlink descriptor's FSM.

0: The inlink descriptor's FSM is working

1: The inlink descriptor's FSM is in idle state

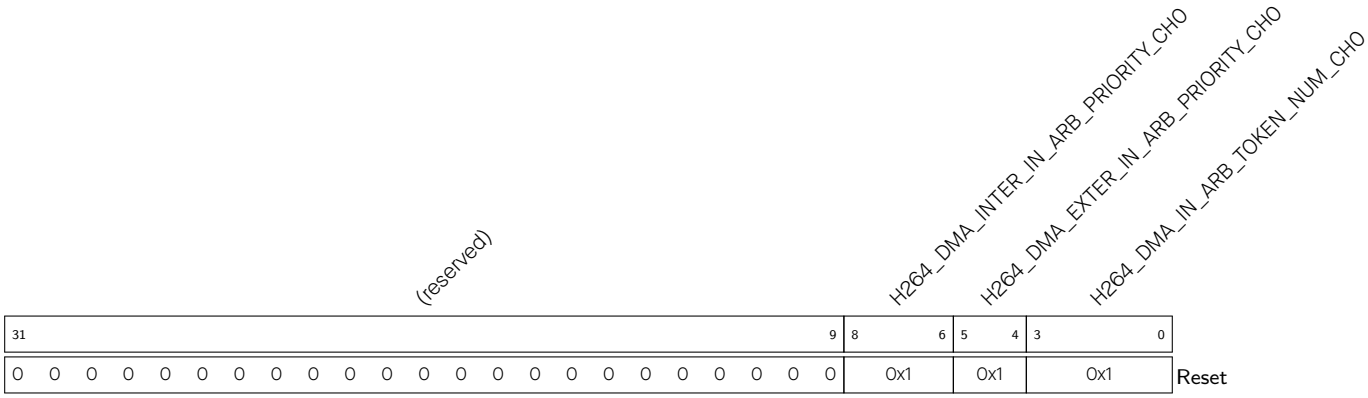
(RO)

Register 41.103. H264_DMA_IN_LINK_ADDR_CH0_REG (0x0520)



H264_DMA_INLINK_ADDR_CH0 Configures the first inlink descriptor's address. (R/W)

Register 41.104. H264_DMA_IN_ARB_CH0_REG (0x0540)

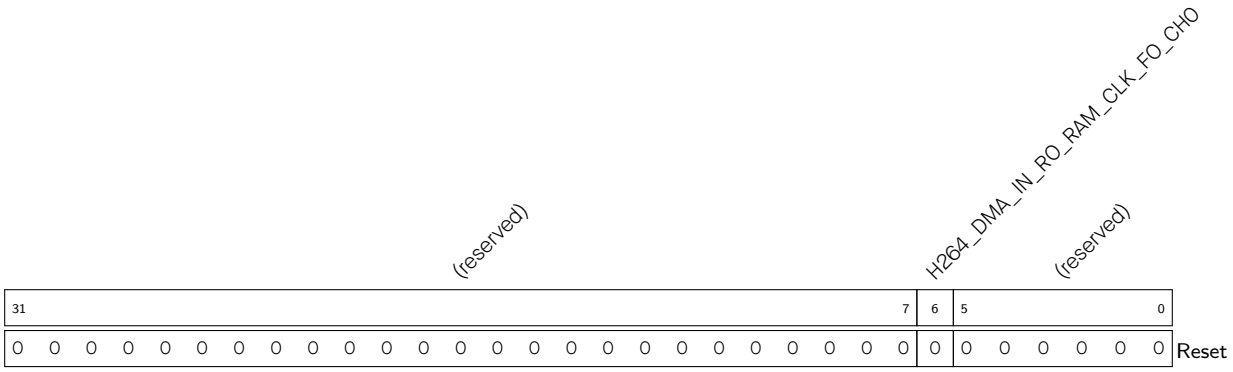


H264_DMA_IN_ARB_TOKEN_NUM_CH0 Configures the maximum token count for the arbiter configuration. (R/W)

H264_DMA_EXTER_IN_ARB_PRIORITY_CH0 Configures the priority of external memory access for RX channel 0. (R/W)

H264_DMA_INTER_IN_ARB_PRIORITY_CH0 Configures the priority of internal memory access for RX channel 0. (R/W)

Register 41.105. H264_DMA_IN_RO_PD_CONF_CH0_REG (0x0548)



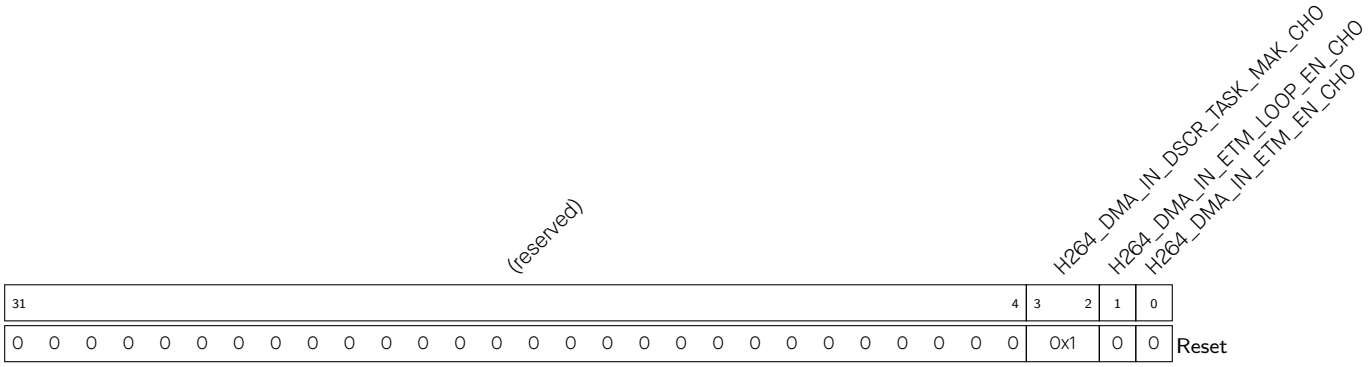
H264_DMA_IN_RO_RAM_CLK_FO_CH0 Configures whether to force enable clock gating for DMA RX reorder RAM.

0: A clock gating will be used when accessing the RAM in DMA

1: Force to open the clock and bypass the clock gating when accessing the RAM in DMA

(R/W)

Register 41.106. H264_DMA_IN_ETM_CONF_CH0_REG (0x056C)

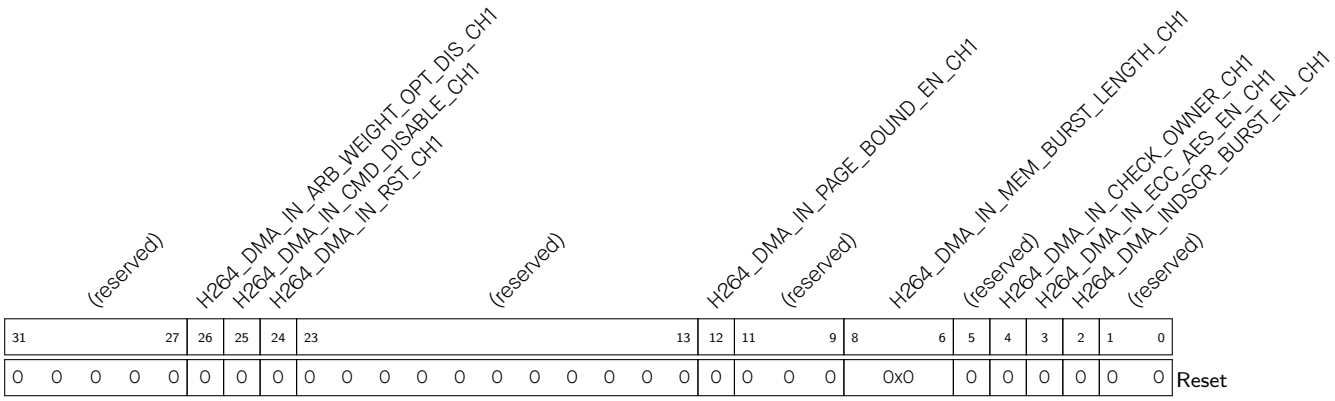


H264_DMA_IN_ETM_EN_CH0 Set this bit to 1 to enable ETM task function (R/W)

H264_DMA_IN_ETM_LOOP_EN_CH0 when this bit is 1, dscr can be processed after receiving a task (R/W)

H264_DMA_IN_DSCR_TASK_MAK_CH0 ETM dscr_ready maximum cache numbers (R/W)

Register 41.107. H264_DMA_IN_CONF0_CH1_REG (0x0600)


(reserved)
H264_DMA_IN_ARB_WEIGHT_OPT_DIS_CH1
H264_DMA_IN_CMD_DISABLE_CH1
H264_DMA_IN_RST_CH1
(reserved)
H264_DMA_IN_PAGE_BOUND_EN_CH1
(reserved)
H264_DMA_IN_MEM_BURST_LENGTH_CH1
(reserved)
H264_DMA_IN_CHECK_OWNER_CH1
H264_DMA_IN_ECC_AES_EN_CH1
H264_DMA_INDSCR_BURST_EN_CH1
(reserved)
31 27 26 25 24 23 13 12 11 9 8 6 5 4 3 2 1 0
0 0 0

Register 41.107. H264_DMA_IN_CONF0_CH1_REG (0x0600)

Continued from the previous page...

H264_DMA_IN_CMD_DISABLE_CH1 Configures whether to disable command on RX channel 1.

0: Disable

1: Enable

(R/W)

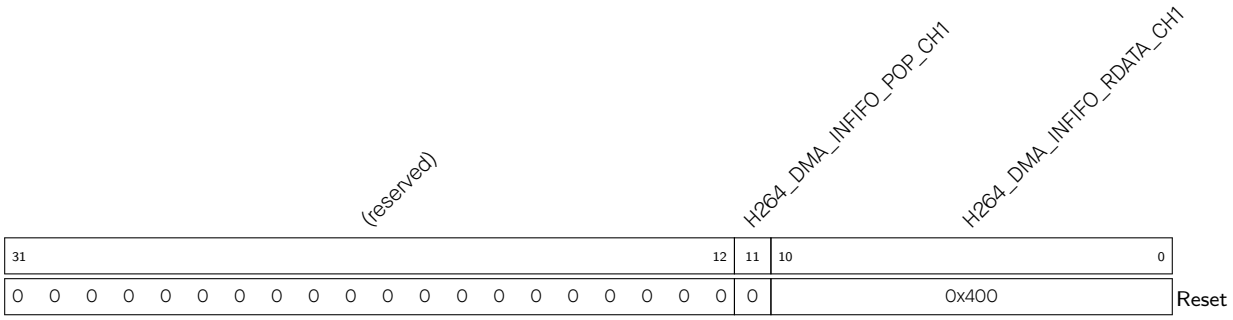
H264_DMA_IN_ARB_WEIGHT_OPT_DIS_CH1 Configures whether to disable weight arbitration for RX channel 1.

0: Enable

1: Disable

(R/W)

Register 41.108. H264_DMA_IN_POP_CH1_REG (0x0618)



H264_DMA_INFIFO_RDATA_CH1 Represents the data popped from DMA RX FIFO. (RO)

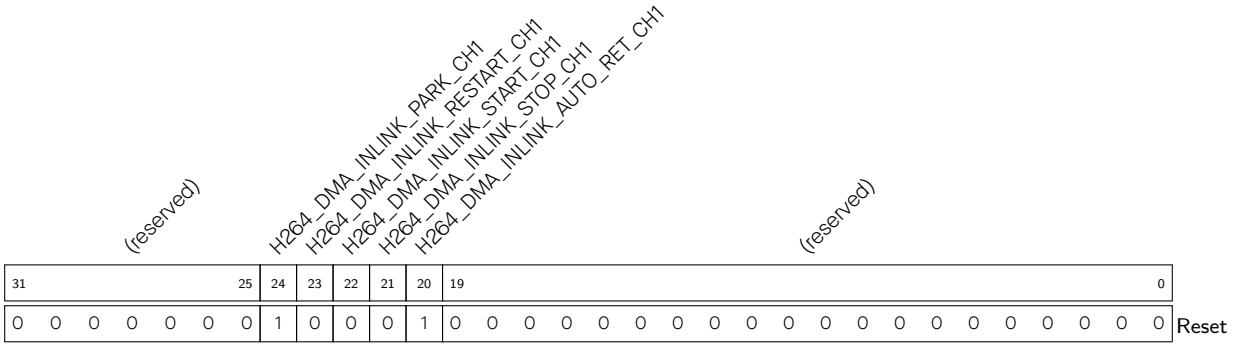
H264_DMA_INFIFO_POP_CH1 Configures whether to pop data from DMA RX FIFO.

0: No effect

1: Pop

(R/W/SC)

Register 41.109. H264_DMA_IN_LINK_CONF_CH1_REG (0x061C)



H264_DMA_INLINK_AUTO_RET_CH1 Configures whether to return to the address of the current receive descriptor when there are errors during data reception.

0: Not return

1: Return

(R/W)

H264_DMA_INLINK_STOP_CH1 Configures whether to stop processing the inlink descriptors.

0: No effect

1: Stop

(R/W/SC)

H264_DMA_INLINK_START_CH1 Configures whether to start processing the inlink descriptors.

0: No effect

1: Start

(R/W/SC)

H264_DMA_INLINK_RESTART_CH1 Configures whether to restart a new inlink from the last address.

0: No effect

1: Restart

(R/W/SC)

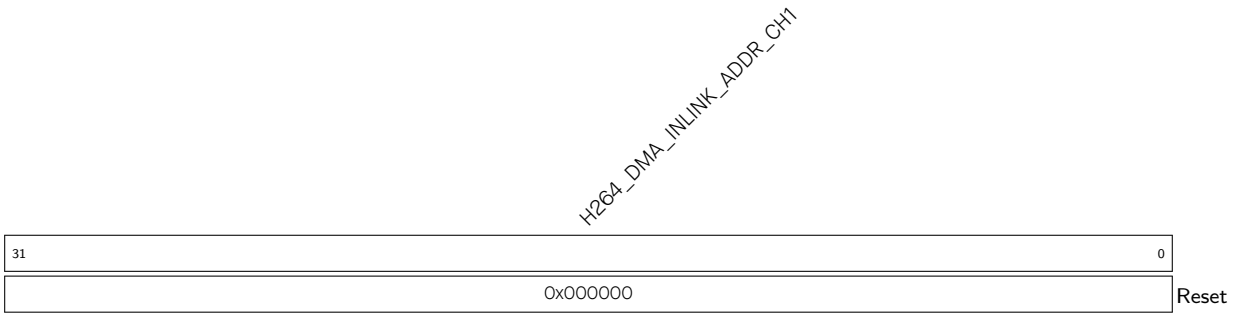
H264_DMA_INLINK_PARK_CH1 Represents the status of the inlink descriptor's FSM.

0: The inlink descriptor's FSM is working

1: The inlink descriptor's FSM is in idle state

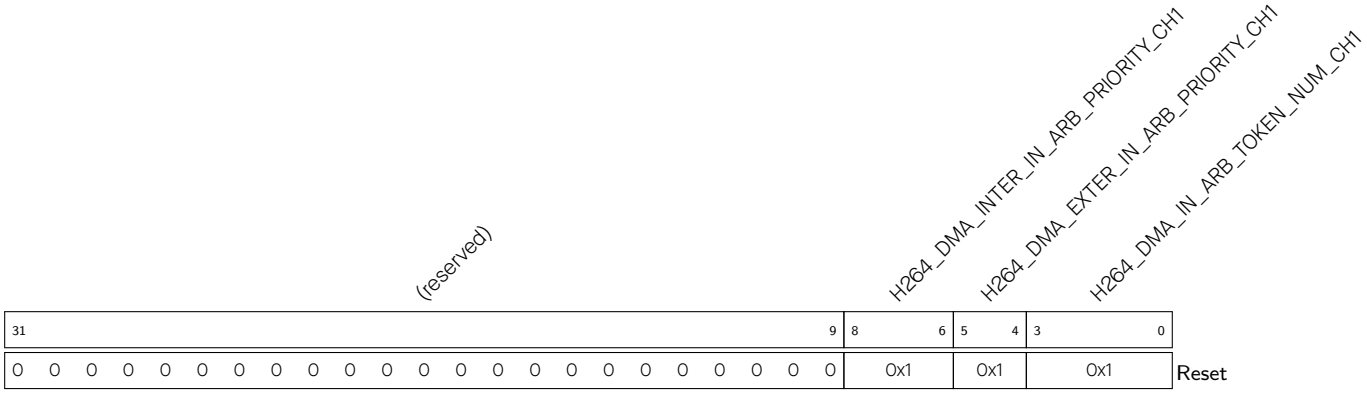
(RO)

Register 41.110. H264_DMA_IN_LINK_ADDR_CH1_REG (0x0620)



H264_DMA_INLINK_ADDR_CH1 Configures the first inlink descriptor's address. (R/W)

Register 41.111. H264_DMA_IN_ARB_CH1_REG (0x0640)

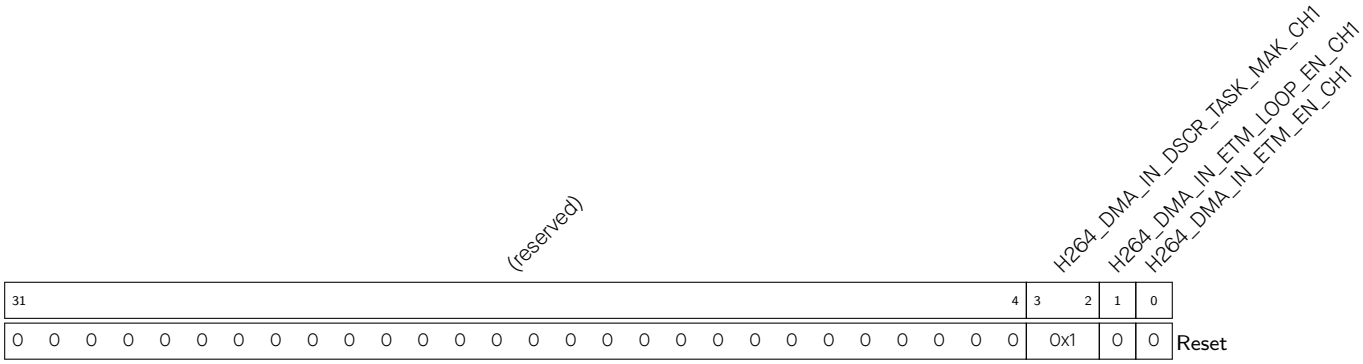


H264_DMA_IN_ARB_TOKEN_NUM_CH1 Configures the maximum token count for the arbiter configuration. (R/W)

H264_DMA_EXTER_IN_ARB_PRIORITY_CH1 Configures the priority of external memory access for RX channel 1. (R/W)

H264_DMA_INTER_IN_ARB_PRIORITY_CH1 Configures the priority of internal memory access for RX channel 1. (R/W)

Register 41.112. H264_DMA_IN_ETM_CONF_CH1_REG (0x0648)

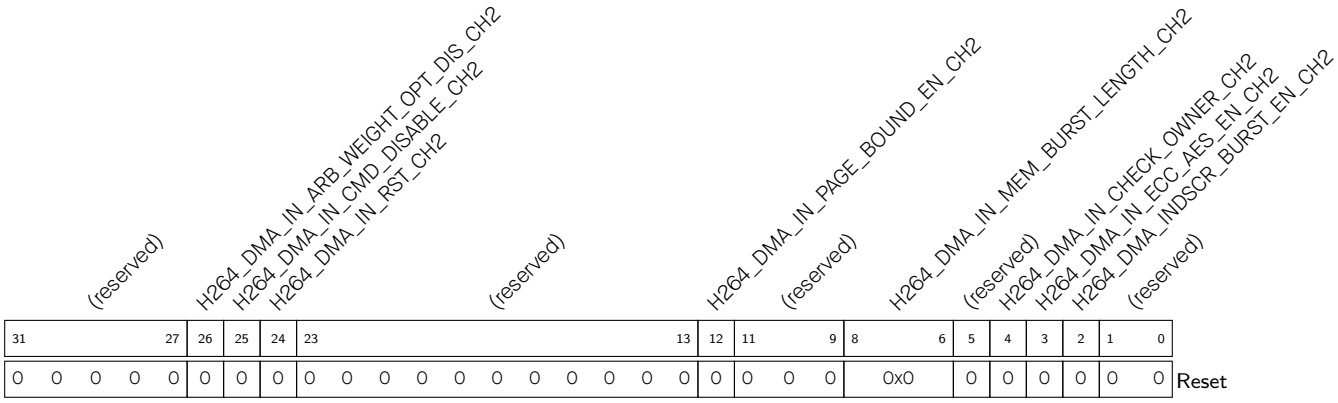


H264_DMA_IN_ETM_EN_CH1 Set this bit to 1 to enable ETM task function (R/W)

H264_DMA_IN_ETM_LOOP_EN_CH1 when this bit is 1, dscr can be processed after receiving a task (R/W)

H264_DMA_IN_DSCR_TASK_MAK_CH1 ETM dscr_ready maximum cache numbers (R/W)

Register 41.113. H264_DMA_IN_CONF0_CH2_REG (0x0700)


(reserved)
H264_DMA_IN_ARB_WEIGHT_OPT_DIS_CH2
H264_DMA_IN_CMD_DISABLE_CH2
H264_DMA_IN_RST_CH2
(reserved)
H264_DMA_IN_PAGE_BOUND_EN_CH2
(reserved)
H264_DMA_IN_MEM_BURST_LENGTH_CH2
(reserved)
H264_DMA_IN_CHECK_OWNER_CH2
H264_DMA_IN_ECC_AES_EN_CH2
H264_DMA_INDSCR_BURST_EN_CH2
(reserved)
31 27 26 25 24 23 13 12 11 9 8 6 5 4 3 2 1 0
0 0 0

Register 41.113. H264_DMA_IN_CONF0_CH2_REG (0x0700)

Continued from the previous page...

H264_DMA_IN_CMD_DISABLE_CH2 Configures whether to disable command on RX channel 2.

0: Disable

1: Enable

(R/W)

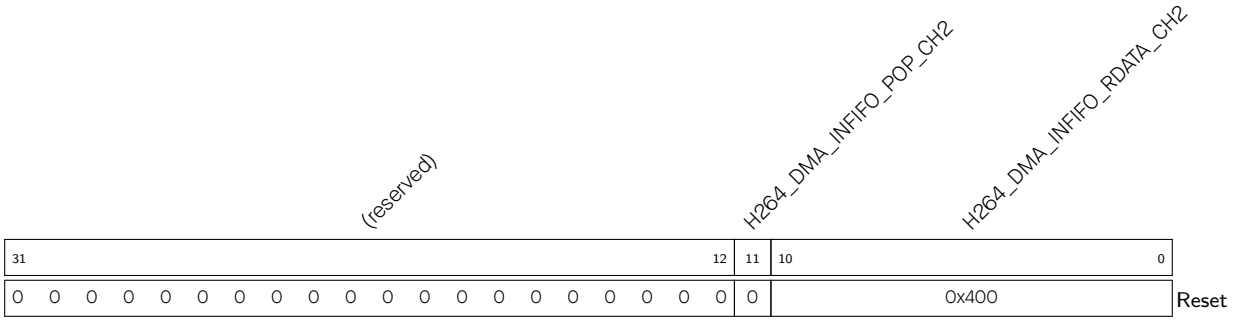
H264_DMA_IN_ARB_WEIGHT_OPT_DIS_CH2 Configures whether to disable weight arbitration for RX channel 2.

0: Enable

1: Disable

(R/W)

Register 41.114. H264_DMA_IN_POP_CH2_REG (0x0718)



H264_DMA_INFIFO_RDATA_CH2 Represents the data popped from DMA RX FIFO. (RO)

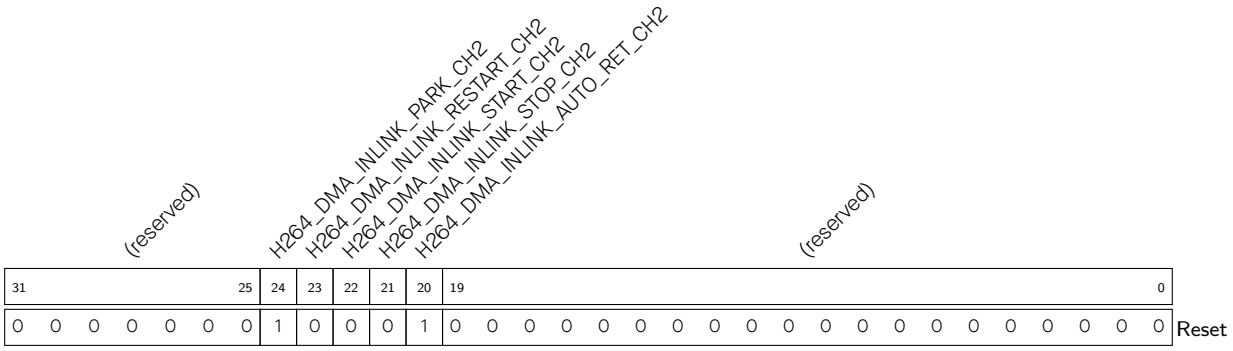
H264_DMA_INFIFO_POP_CH2 Configures whether to pop data from DMA RX FIFO.

0: No effect

1: Pop

(R/W/SC)

Register 41.115. H264_DMA_IN_LINK_CONF_CH2_REG (0x071C)



H264_DMA_INLINK_AUTO_RET_CH2 Configures whether to return to the address of the current receive descriptor when there are errors during data reception.

0: Not return

1: Return

(R/W)

H264_DMA_INLINK_STOP_CH2 Configures whether to stop processing the inlink descriptors.

0: No effect

1: Stop

(R/W/SC)

H264_DMA_INLINK_START_CH2 Configures whether to start processing the inlink descriptors.

0: No effect

1: Start

(R/W/SC)

H264_DMA_INLINK_RESTART_CH2 Configures whether to restart a new inlink from the last address.

0: No effect

1: Restart

(R/W/SC)

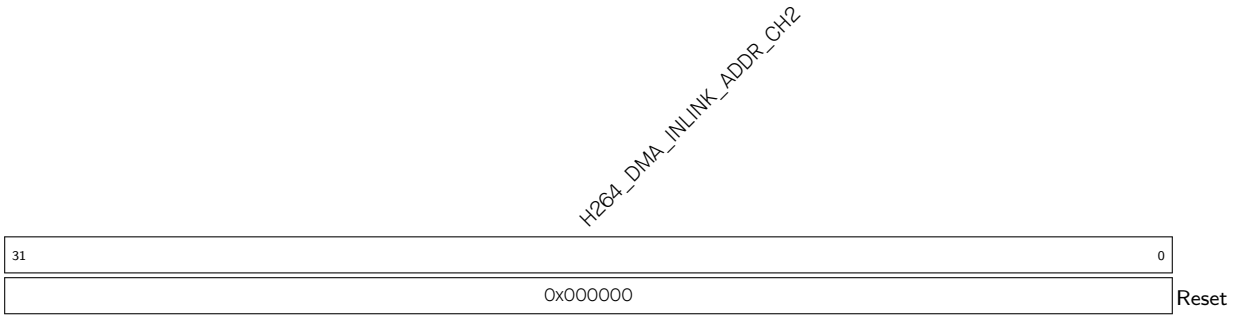
H264_DMA_INLINK_PARK_CH2 Represents the status of the inlink descriptor's FSM.

0: The inlink descriptor's FSM is working

1: The inlink descriptor's FSM is in idle state

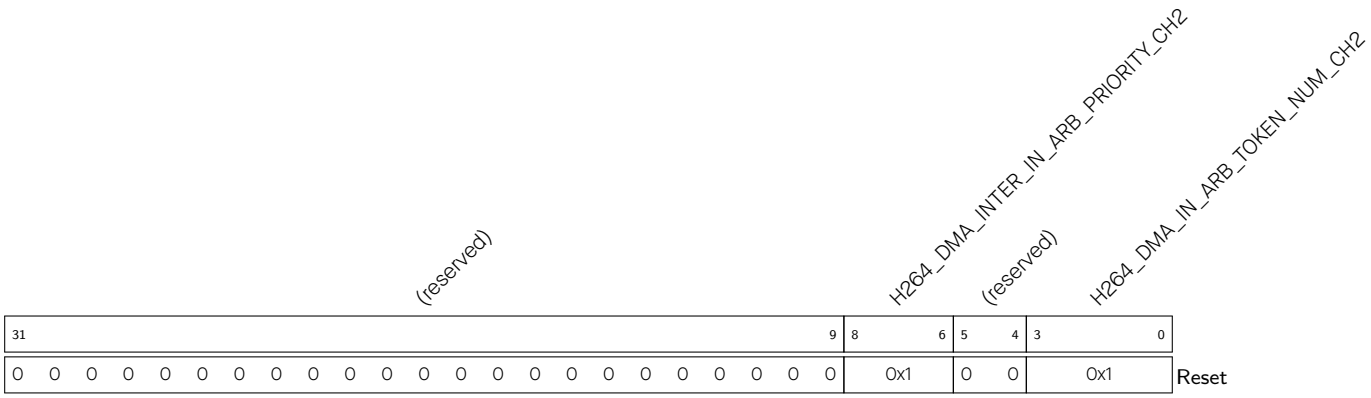
(RO)

Register 41.116. H264_DMA_IN_LINK_ADDR_CH2_REG (0x0720)



H264_DMA_INLINK_ADDR_CH2 Configures the first inlink descriptor's address. (R/W)

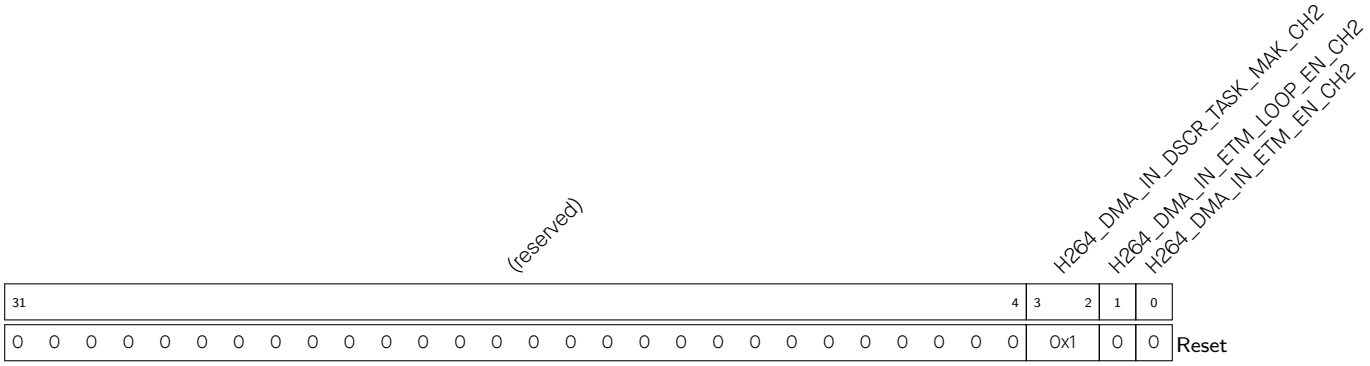
Register 41.117. H264_DMA_IN_ARB_CH2_REG (0x0740)



H264_DMA_IN_ARB_TOKEN_NUM_CH2 Configures the maximum token count for the arbiter configuration. (R/W)

H264_DMA_INTER_IN_ARB_PRIORITY_CH2 Configures the priority of internal memory access for RX channel 2. (R/W)

Register 41.118. H264_DMA_IN_ETM_CONF_CH2_REG (0x0748)

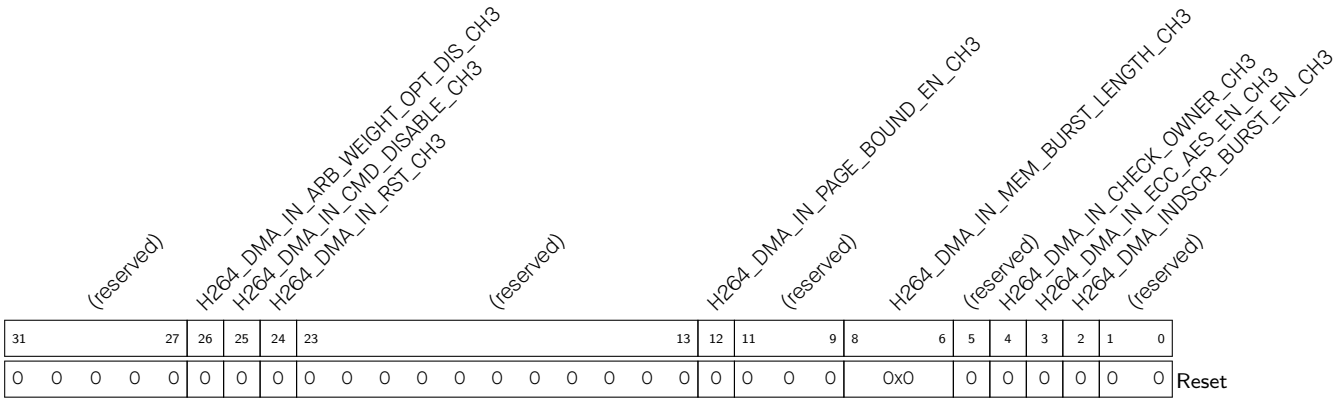


H264_DMA_IN_ETM_EN_CH2 Set this bit to 1 to enable ETM task function (R/W)

H264_DMA_IN_ETM_LOOP_EN_CH2 when this bit is 1, dscr can be processed after receiving a task (R/W)

H264_DMA_IN_DSCR_TASK_MAK_CH2 ETM dscr_ready maximum cache numbers (R/W)

Register 41.119. H264_DMA_IN_CONF0_CH3_REG (0x0800)


(reserved)
H264_DMA_IN_ARB_WEIGHT_OPT_DIS_CH3
H264_DMA_IN_CMD_DISABLE_CH3
H264_DMA_IN_RST_CH3
(reserved)
H264_DMA_IN_PAGE_BOUND_EN_CH3
(reserved)
H264_DMA_IN_MEM_BURST_LENGTH_CH3
(reserved)
H264_DMA_IN_CHECK_OWNER_CH3
H264_DMA_IN_ECC_AES_EN_CH3
H264_DMA_INDSCR_BURST_EN_CH3
(reserved)
31 27 26 25 24 23 13 12 11 9 8 6 5 4 3 2 1 0
0 0 0

Register 41.119. H264_DMA_IN_CONF0_CH3_REG (0x0800)

Continued from the previous page...

H264_DMA_IN_CMD_DISABLE_CH3 Configures whether to disable command on RX channel 3.

0: Disable

1: Enable

(R/W)

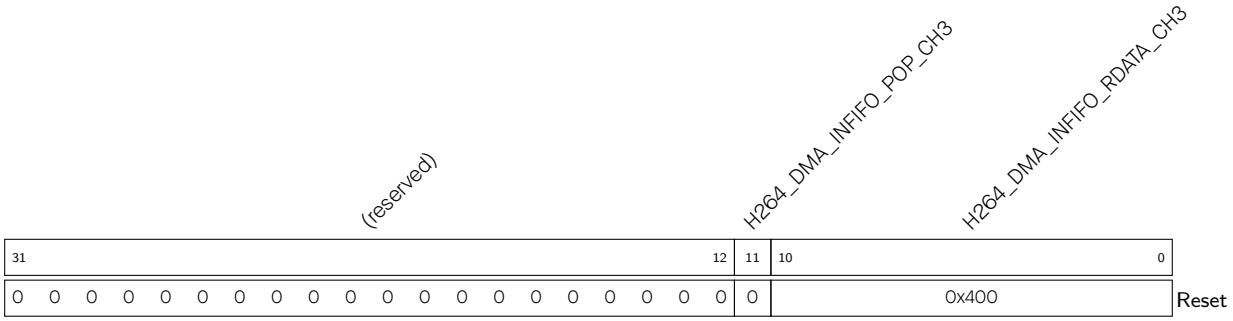
H264_DMA_IN_ARB_WEIGHT_OPT_DIS_CH3 Configures whether to disable weight arbitration for RX channel 3.

0: Enable

1: Disable

(R/W)

Register 41.120. H264_DMA_IN_POP_CH3_REG (0x0818)



H264_DMA_INFIFO_RDATA_CH3 Represents the data popped from DMA RX FIFO. (RO)

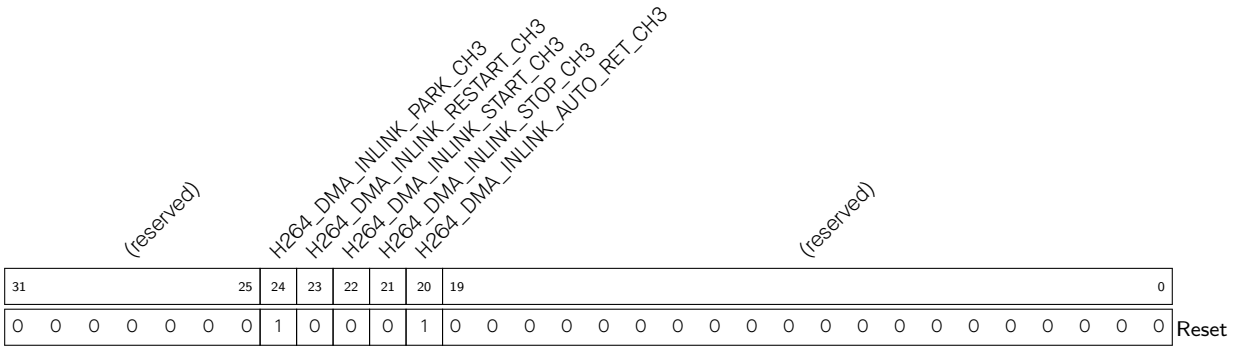
H264_DMA_INFIFO_POP_CH3 Configures whether to pop data from DMA RX FIFO.

0: No effect

1: Pop

(R/W/SC)

Register 41.121. H264_DMA_IN_LINK_CONF_CH3_REG (0x081C)



H264_DMA_INLINK_AUTO_RET_CH3 Configures whether to return to the address of the current receive descriptor when there are errors during data reception.

0: Not return

1: Return

(R/W)

H264_DMA_INLINK_STOP_CH3 Configures whether to stop processing the inlink descriptors.

0: No effect

1: Stop

(R/W/SC)

H264_DMA_INLINK_START_CH3 Configures whether to start processing the inlink descriptors.

0: No effect

1: Start

(R/W/SC)

H264_DMA_INLINK_RESTART_CH3 Configures whether to restart a new inlink from the last address.

0: No effect

1: Restart

(R/W/SC)

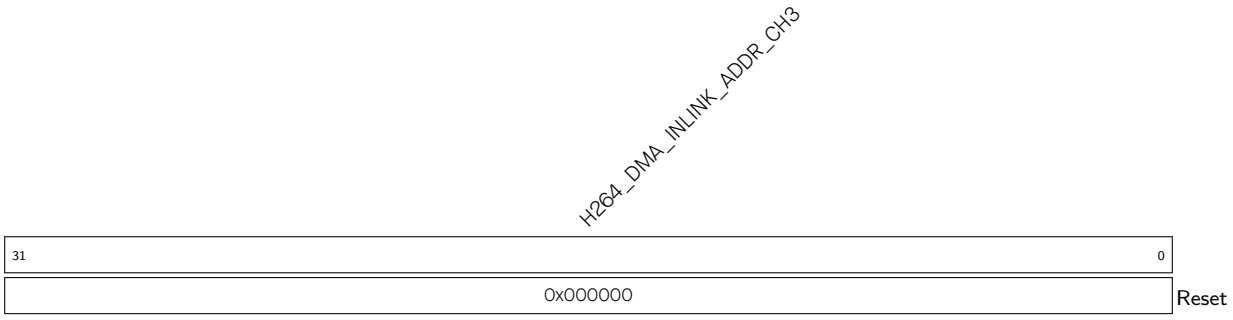
H264_DMA_INLINK_PARK_CH3 Represents the status of the inlink descriptor's FSM.

0: The inlink descriptor's FSM is working

1: The inlink descriptor's FSM is in idle state

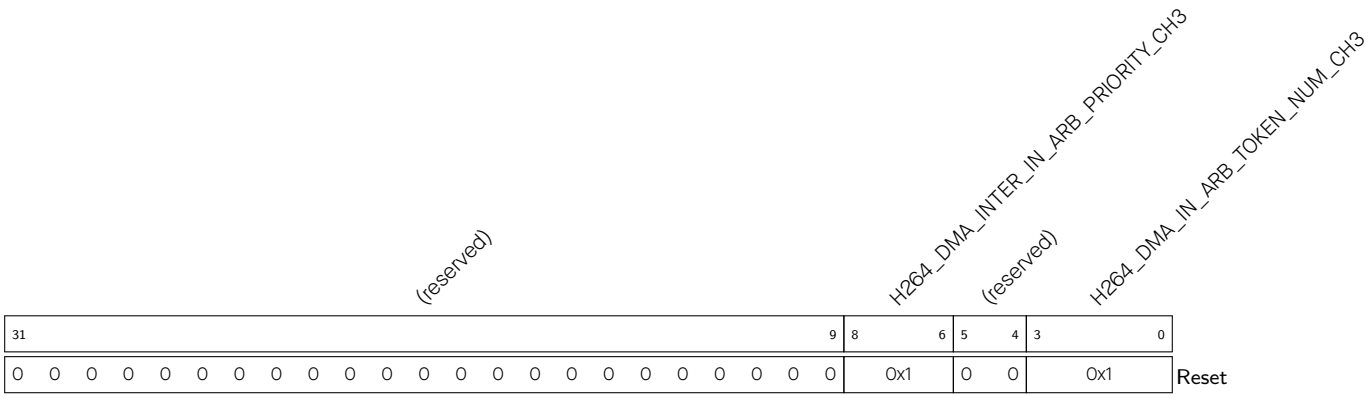
(RO)

Register 41.122. H264_DMA_IN_LINK_ADDR_CH3_REG (0x0820)



H264_DMA_INLINK_ADDR_CH3 Configures the first inlink descriptor's address. (R/W)

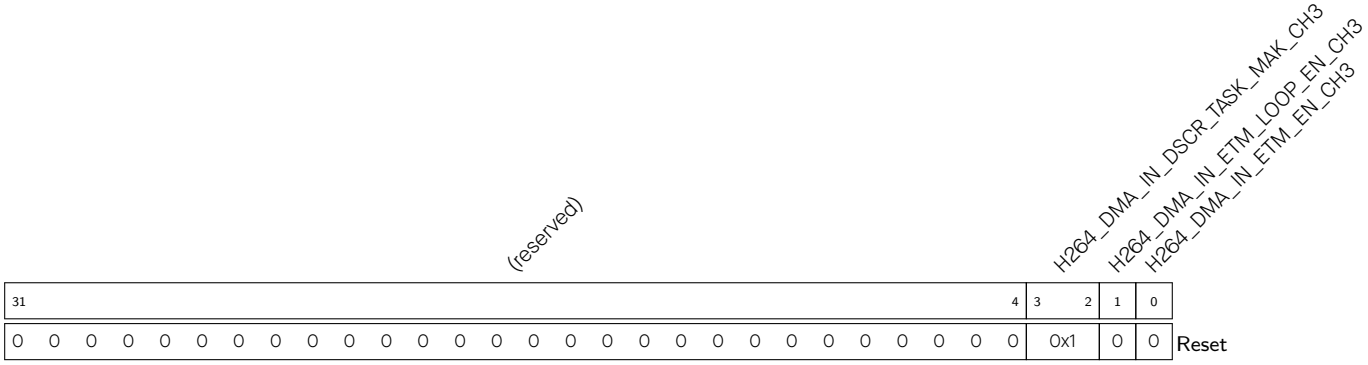
Register 41.123. H264_DMA_IN_ARB_CH3_REG (0x0840)



H264_DMA_IN_ARB_TOKEN_NUM_CH3 Configures the maximum token count for the arbiter configuration. (R/W)

H264_DMA_INTER_IN_ARB_PRIORITY_CH3 Configures the priority of internal memory access for RX channel 3. (R/W)

Register 41.124. H264_DMA_IN_ETM_CONF_CH3_REG (0x0848)

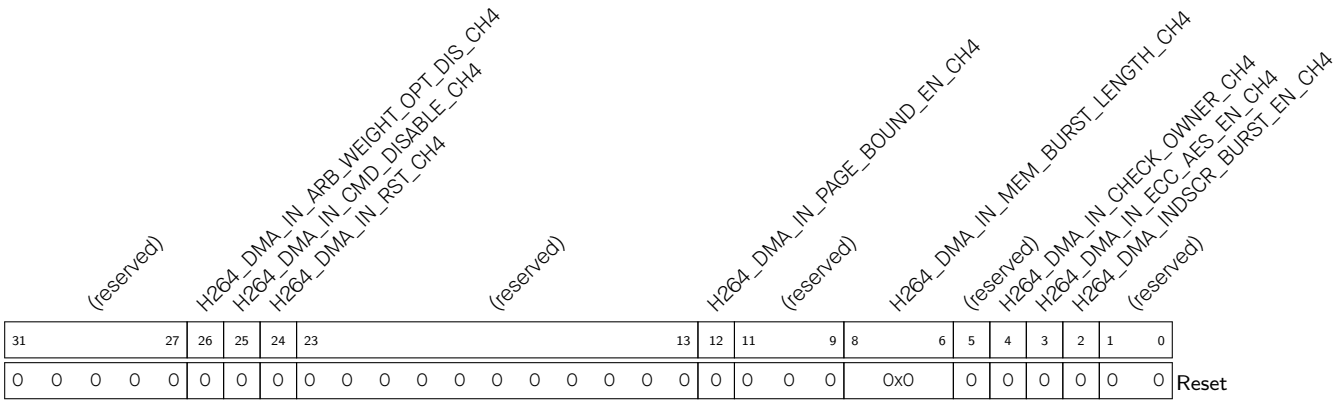


H264_DMA_IN_ETM_EN_CH3 Set this bit to 1 to enable ETM task function (R/W)

H264_DMA_IN_ETM_LOOP_EN_CH3 when this bit is 1, dscr can be processed after receiving a task (R/W)

H264_DMA_IN_DSCR_TASK_MAK_CH3 ETM dscr_ready maximum cache numbers (R/W)

Register 41.125. H264_DMA_IN_CONF0_CH4_REG (0x0900)


(reserved)
H264_DMA_IN_ARB_WEIGHT_OPT_DIS_CH4
H264_DMA_IN_CMD_DISABLE_CH4
H264_DMA_IN_RST_CH4
(reserved)
H264_DMA_IN_PAGE_BOUND_EN_CH4
(reserved)
H264_DMA_IN_MEM_BURST_LENGTH_CH4
(reserved)
H264_DMA_IN_CHECK_OWNER_CH4
H264_DMA_IN_ECC_AES_EN_CH4
H264_DMA_INDSCR_BURST_EN_CH4
(reserved)
31 27 26 25 24 23 13 12 11 9 8 6 5 4 3 2 1 0
0 0 0

Register 41.125. H264_DMA_IN_CONF0_CH4_REG (0x0900)

Continued from the previous page...

H264_DMA_IN_CMD_DISABLE_CH4 Configures whether to disable command on RX channel 4.

0: Disable

1: Enable

(R/W)

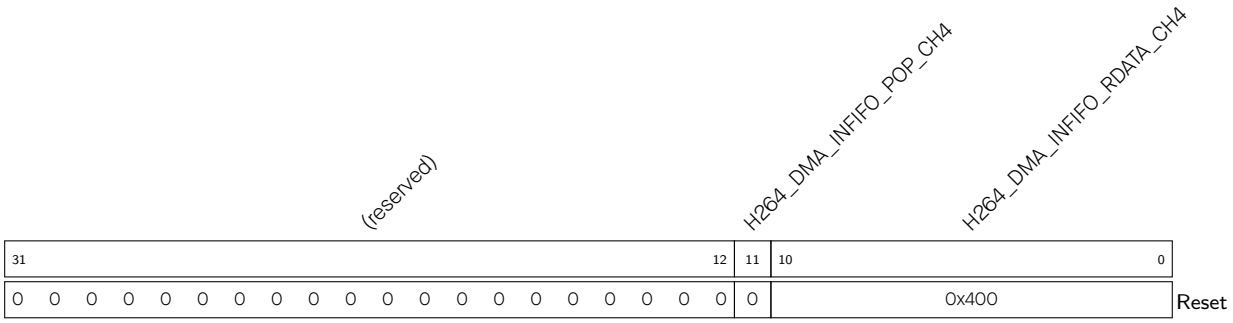
H264_DMA_IN_ARB_WEIGHT_OPT_DIS_CH4 Configures whether to disable weight arbitration for RX channel 4.

0: Enable

1: Disable

(R/W)

Register 41.126. H264_DMA_IN_POP_CH4_REG (0x0918)



H264_DMA_INFIFO_RDATA_CH4 Represents the data popped from DMA RX FIFO. (RO)

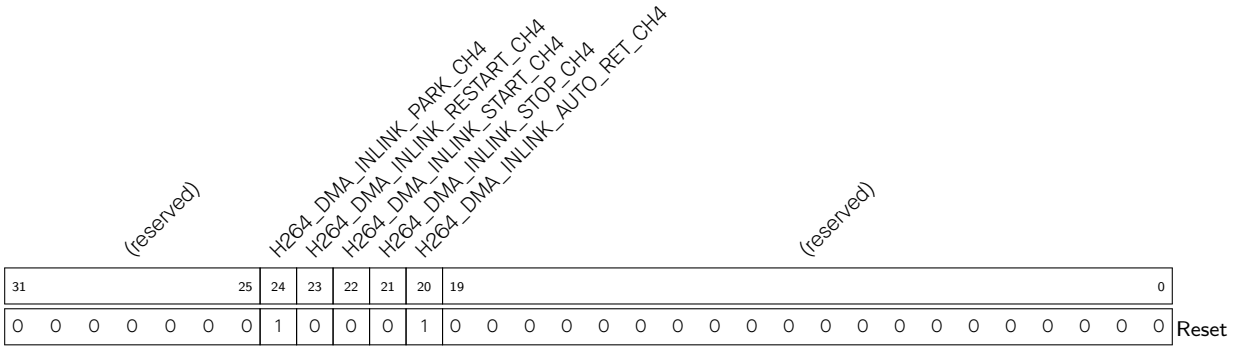
H264_DMA_INFIFO_POP_CH4 Configures whether to pop data from DMA RX FIFO.

0: No effect

1: Pop

(R/W/SC)

Register 41.127. H264_DMA_IN_LINK_CONF_CH4_REG (0x091C)



H264_DMA_INLINK_AUTO_RET_CH4 Configures whether to return to the address of the current receive descriptor when there are errors during data reception.

0: Not return

1: Return

(R/W)

H264_DMA_INLINK_STOP_CH4 Configures whether to stop processing the inlink descriptors.

0: No effect

1: Stop

(R/W/SC)

H264_DMA_INLINK_START_CH4 Configures whether to start processing the inlink descriptors.

0: No effect

1: Start

(R/W/SC)

H264_DMA_INLINK_RESTART_CH4 Configures whether to restart a new inlink from the last address.

0: No effect

1: Restart

(R/W/SC)

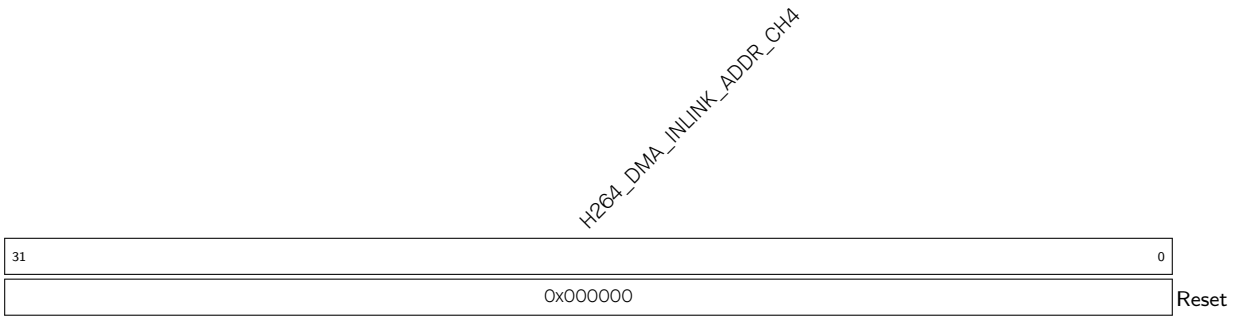
H264_DMA_INLINK_PARK_CH4 Represents the status of the inlink descriptor's FSM.

0: The inlink descriptor's FSM is working

1: The inlink descriptor's FSM is in idle state

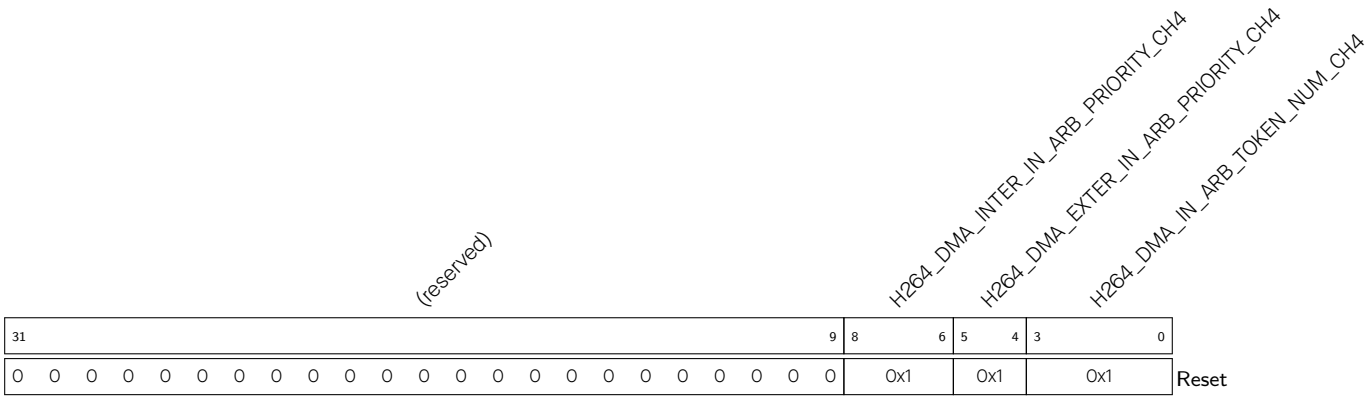
(RO)

Register 41.128. H264_DMA_IN_LINK_ADDR_CH4_REG (0x0920)



H264_DMA_INLINK_ADDR_CH4 Configures the first inlink descriptor's address. (R/W)

Register 41.129. H264_DMA_IN_ARB_CH4_REG (0x0940)

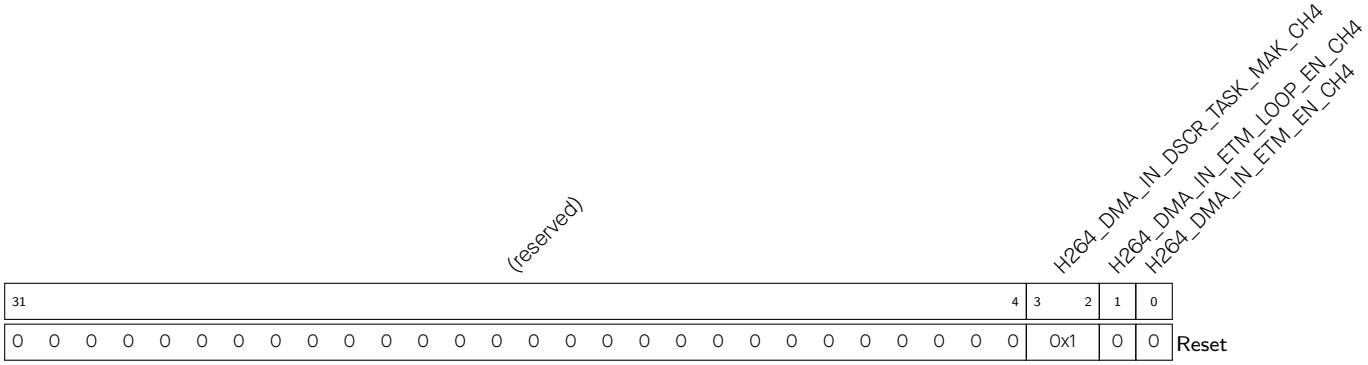


H264_DMA_IN_ARB_TOKEN_NUM_CH4 Configures the maximum token count for the arbiter configuration. (R/W)

H264_DMA_EXTER_IN_ARB_PRIORITY_CH4 Configures the priority of external memory access for RX channel 4. (R/W)

H264_DMA_INTER_IN_ARB_PRIORITY_CH4 Configures the priority of internal memory access for RX channel 4. (R/W)

Register 41.130. H264_DMA_IN_ETM_CONF_CH4_REG (0x0948)

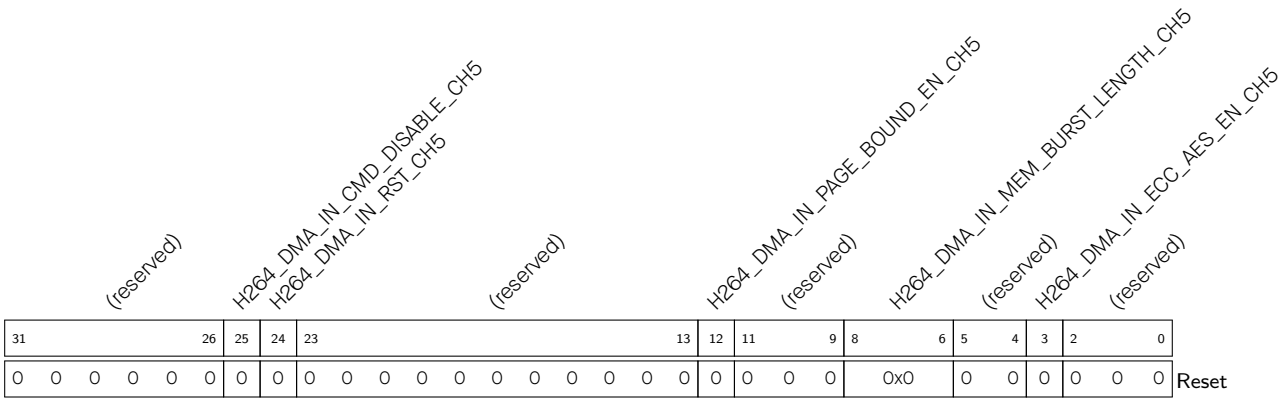


H264_DMA_IN_ETM_EN_CH4 Set this bit to 1 to enable ETM task function (R/W)

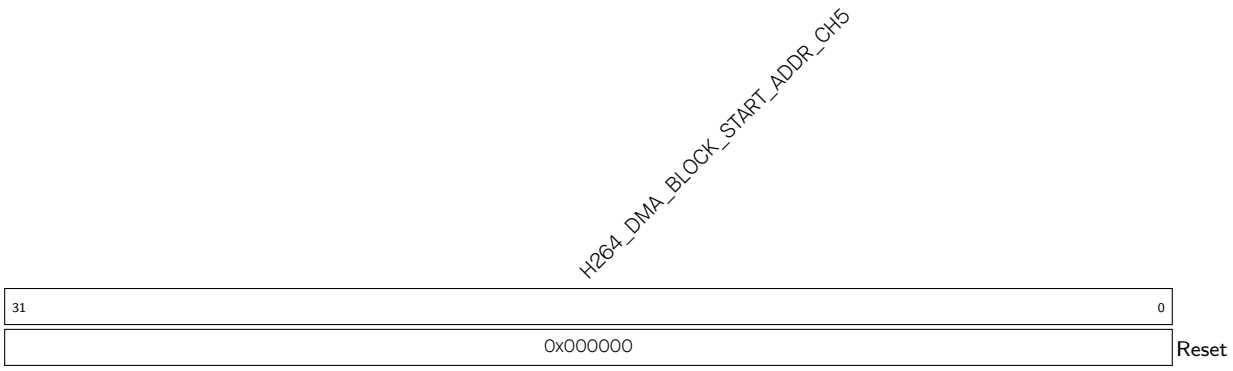
H264_DMA_IN_ETM_LOOP_EN_CH4 when this bit is 1, dscr can be processed after receiving a task (R/W)

H264_DMA_IN_DSCR_TASK_MAK_CH4 ETM dscr_ready maximum cache numbers (R/W)

Register 41.131. H264_DMA_IN_CONF0_CH5_REG (0x0A00)

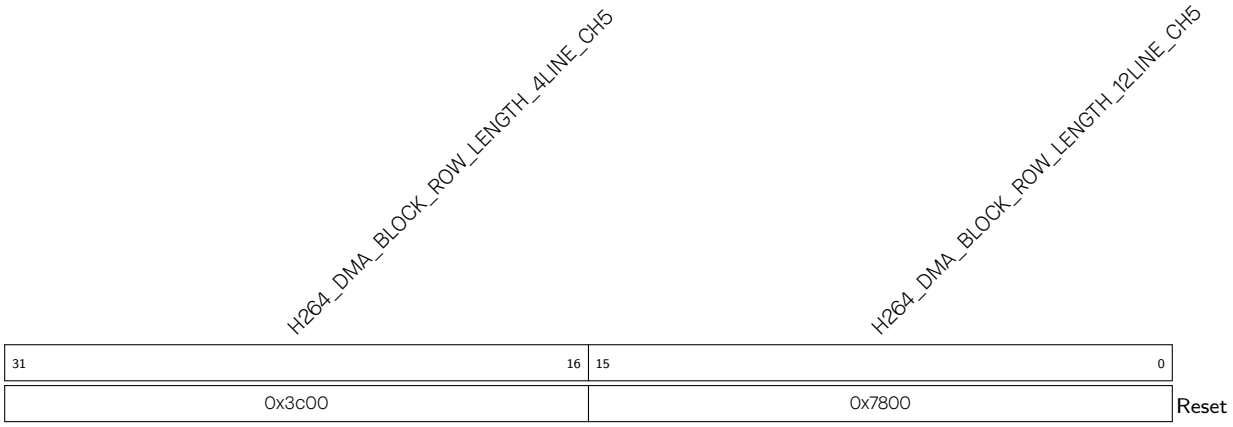

(reserved)
H264_DMA_IN_CMD_DISABLE_CH5
H264_DMA_IN_RST_CH5
(reserved)
H264_DMA_IN_PAGE_BOUND_EN_CH5
(reserved)
H264_DMA_IN_MEM_BURST_LENGTH_CH5
(reserved)
H264_DMA_IN_ECC_AES_EN_CH5
(reserved)
31 26 25 24 23 13 12 11 9 8 6 5 4 3 2 0
0 0 0

Register 41.132. H264_DMA_IN_CONF1_CH5_REG (0x0A04)



H264_DMA_BLOCK_START_ADDR_CH5 Configures the destination start address of RX channel 5. (R/W)

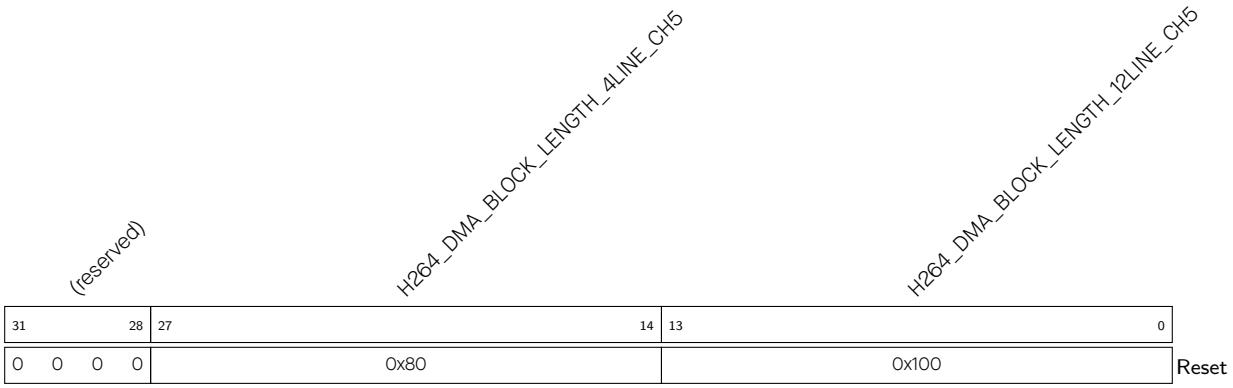
Register 41.133. H264_DMA_IN_CONF2_CH5_REG (0x0A08)



H264_DMA_BLOCK_ROW_LENGTH_12LINE_CH5 Configures the number of bytes contained in a row block 12line. (R/W)

H264_DMA_BLOCK_ROW_LENGTH_4LINE_CH5 Configures the number of bytes contained in a row block 4line. (R/W)

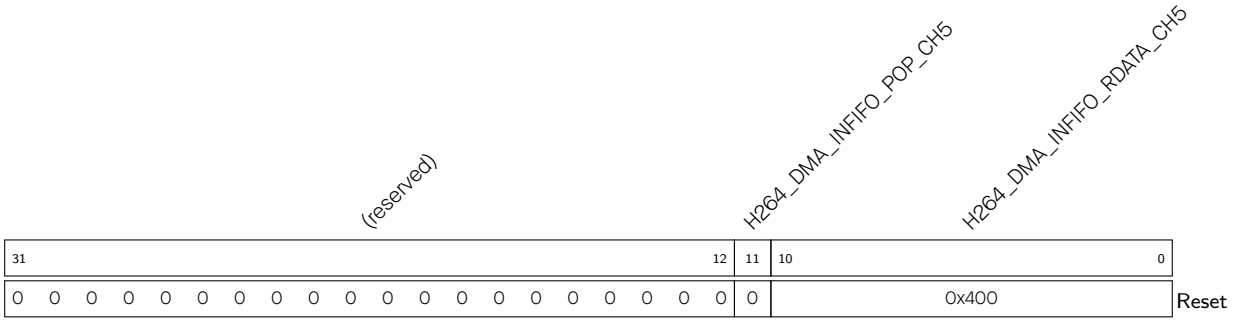
Register 41.134. H264_DMA_IN_CONF3_CH5_REG (0x0A0C)



H264_DMA_BLOCK_LENGTH_12LINE_CH5 Configures the number of bytes contained in a block 12line (R/W)

H264_DMA_BLOCK_LENGTH_4LINE_CH5 Configures the number of bytes contained in a block 4line (R/W)

Register 41.135. H264_DMA_IN_POP_CH5_REG (0x0A24)



H264_DMA_INFIFO_RDATA_CH5 Represents the data popped from DMA RX FIFO. (RO)

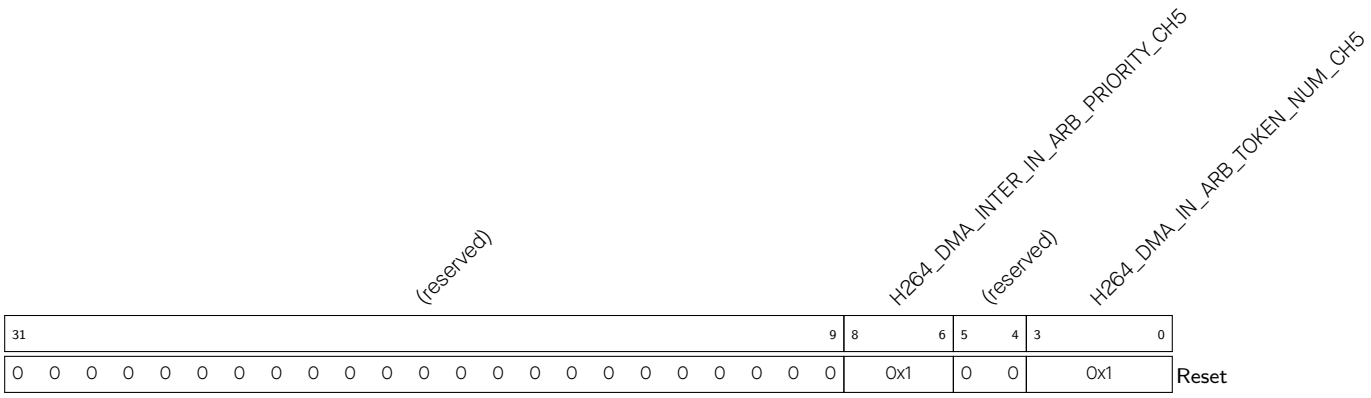
H264_DMA_INFIFO_POP_CH5 Configures whether to pop data from DMA RX FIFO.

0: No effect

1: Pop

(R/W/SC)

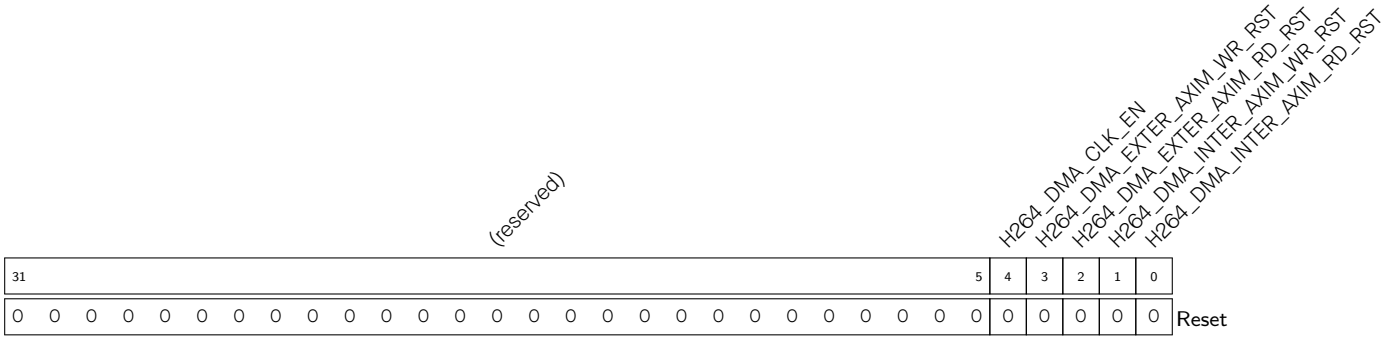
Register 41.136. H264_DMA_IN_ARB_CH5_REG (0x0A40)



H264_DMA_IN_ARB_TOKEN_NUM_CH5 Configures the maximum token count for the arbiter configuration. (R/W)

H264_DMA_INTER_IN_ARB_PRIORITY_CH5 Configures the priority of internal memory access for RX channel 5. (R/W)

Register 41.137. H264_DMA_RST_CONF_REG (0x0B08)



H264_DMA_INTER_AXIM_RD_RST Write 1 and then 0 to reset the inter AXI read FSM. (R/W)

H264_DMA_INTER_AXIM_WR_RST Write 1 and then 0 to reset the inter AXI write FSM. (R/W)

H264_DMA_EXTER_AXIM_RD_RST Write 1 and then 0 to reset the exter AXI read FSM. (R/W)

H264_DMA_EXTER_AXIM_WR_RST Write 1 and then 0 to reset the exter AXI write FSM. (R/W)

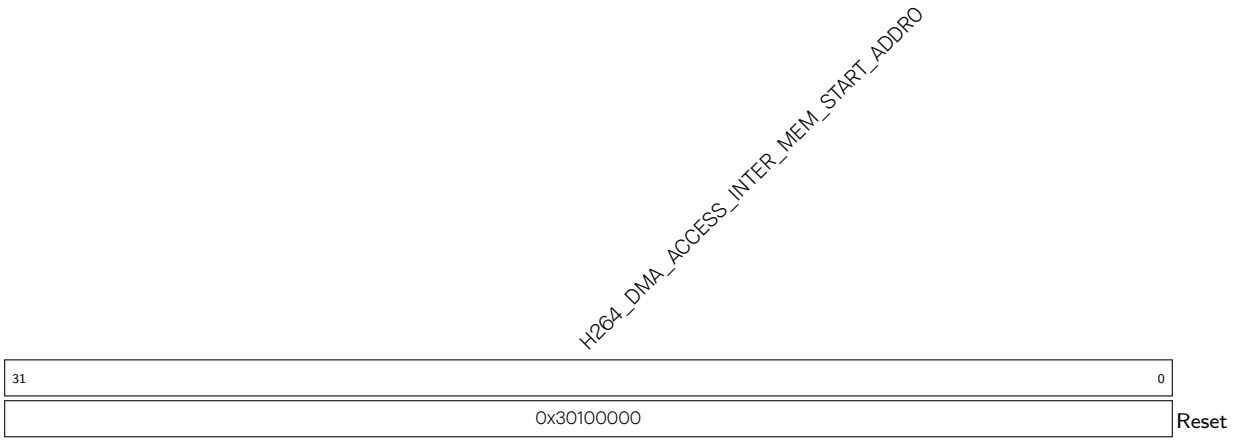
H264_DMA_CLK_EN Configures DMA clock gating.

0: Enable the clock only when the application writes registers

1: Always enable the clock for registers

(R/W)

Register 41.138. H264_DMA_INTER_MEM_START_ADDR0_REG (0x0B0C)



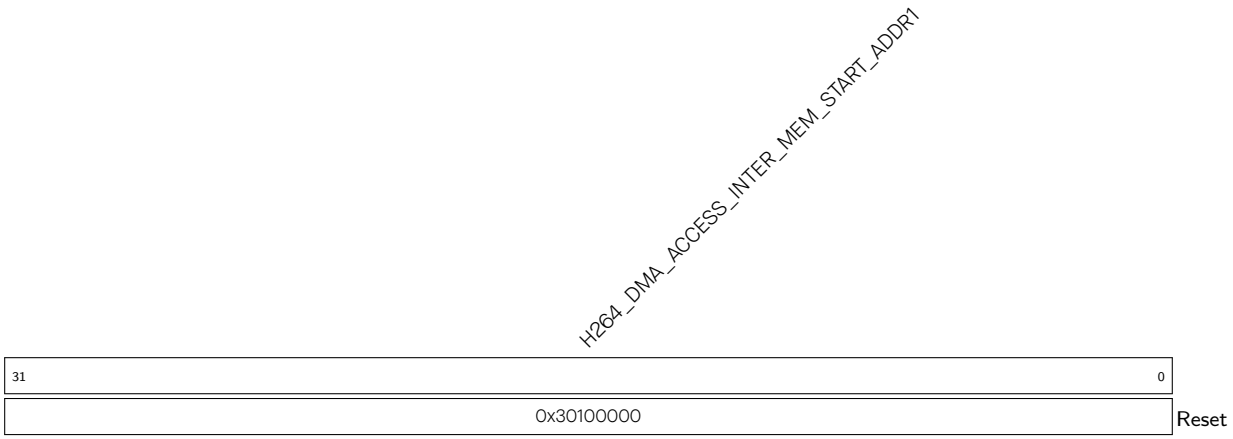
H264_DMA_ACCESS_INTER_MEM_START_ADDR0 Configures the start address of the accessible internal memory address0 space. (R/W)

Register 41.139. H264_DMA_INTER_MEM_END_ADDR0_REG (0x0B10)



H264_DMA_ACCESS_INTER_MEM_END_ADDR0 Configures the end address of the accessible internal memory address0 space. (R/W)

Register 41.140. H264_DMA_INTER_MEM_START_ADDR1_REG (0x0B14)



H264_DMA_ACCESS_INTER_MEM_START_ADDR1 Configures the start address of the accessible internal memory address1 space. (R/W)

Register 41.141. H264_DMA_INTER_MEM_END_ADDR1_REG (0x0B18)



H264_DMA_ACCESS_INTER_MEM_END_ADDR1 Configures the end address of the accessible internal memory address1 space. (R/W)

Register 41.142. H264_DMA_EXTER_MEM_START_ADDR0_REG (0x0B20)



H264_DMA_ACCESS_EXTER_MEM_START_ADDR0 Configures the start address of the accessible external memory address0 space. (R/W)

Register 41.143. H264_DMA_EXTER_MEM_END_ADDR0_REG (0x0B24)



H264_DMA_ACCESS_EXTER_MEM_END_ADDR0 Configures the end address of the accessible external memory address0 space. (R/W)

Register 41.144. H264_DMA_EXTER_MEM_START_ADDR1_REG (0x0B28)



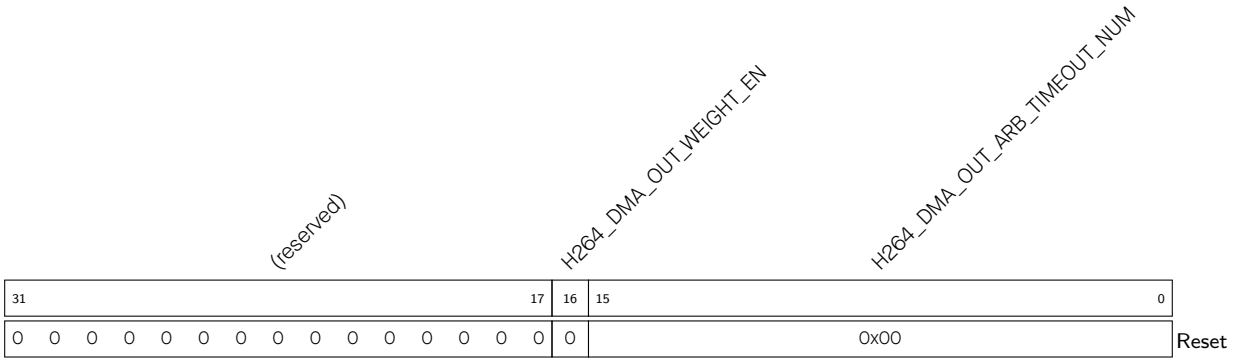
H264_DMA_ACCESS_EXTER_MEM_START_ADDR1 Configures the start address of the accessible external memory address1 space. (R/W)

Register 41.145. H264_DMA_EXTER_MEM_END_ADDR1_REG (0x0B2C)



H264_DMA_ACCESS_EXTER_MEM_END_ADDR1 Configures the end address of the accessible external memory address1 space. (R/W)

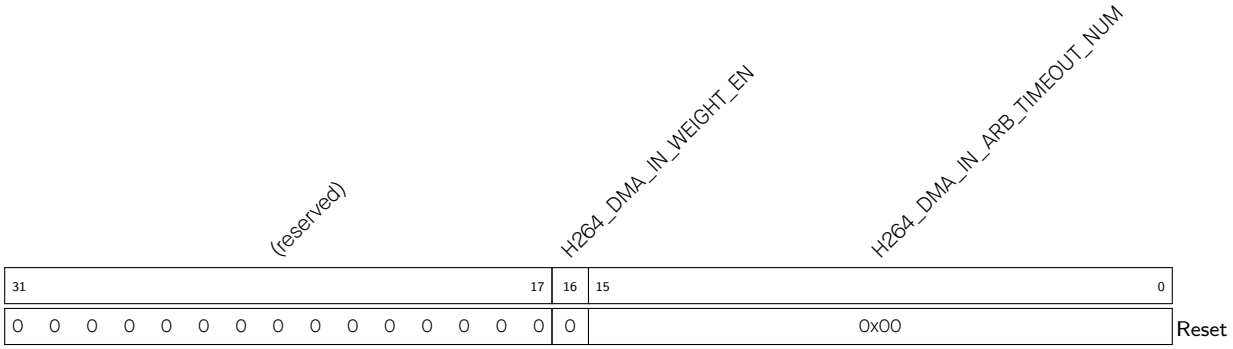
Register 41.146. H264_DMA_OUT_ARB_CONFIG_REG (0x0B30)



H264_DMA_OUT_ARB_TIMEOUT_NUM Configures the max number of timeout count of arbiter. (R/W)

H264_DMA_OUT_WEIGHT_EN Configures whether to enable weight arbitration for TX. (R/W)

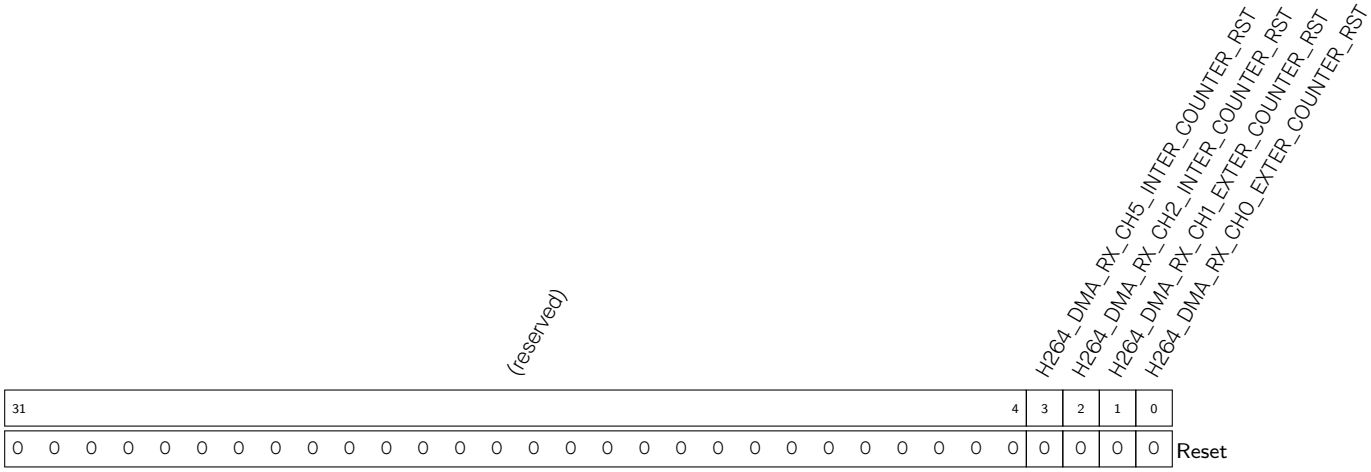
Register 41.147. H264_DMA_IN_ARB_CONFIG_REG (0x0B34)



H264_DMA_IN_ARB_TIMEOUT_NUM Configures the max number of timeout count of arbiter. (R/W)

H264_DMA_IN_WEIGHT_EN Configures whether to enable weight arbitration for RX. (R/W)

Register 41.148. H264_DMA_COUNTER_RST_REG (0x0B50)



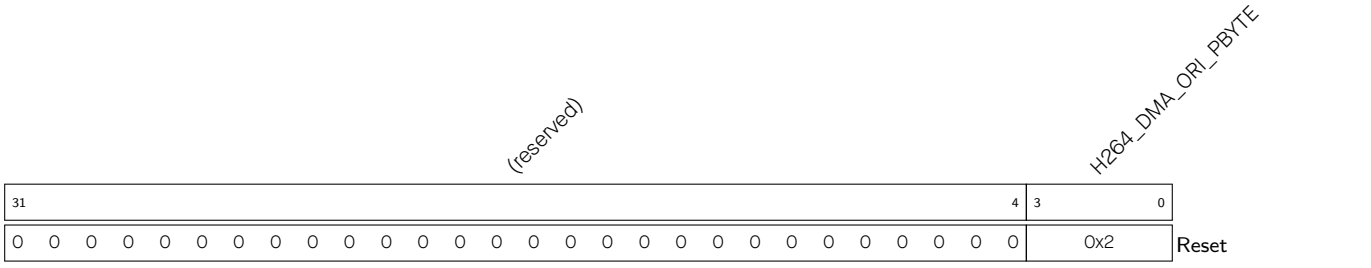
H264_DMA_RX_CH0_EXTER_COUNTER_RST Write 1 then write 0 to reset RX ch0 counter. (R/W)

H264_DMA_RX_CH1_EXTER_COUNTER_RST Write 1 then write 0 to reset RX ch1 counter. (R/W)

H264_DMA_RX_CH2_INTER_COUNTER_RST Write 1 then write 0 to reset RX ch2 counter. (R/W)

H264_DMA_RX_CH5_INTER_COUNTER_RST Write 1 then write 0 to reset RX ch5 counter. (R/W)

Register 41.149. H264_DMA_PBYTE_REG (0x0B64)



H264_DMA_ORI_PBYTE Configures bytes per pixel for the original picture.

0: 0.5 byte/pixel

1: 1 byte/pixel

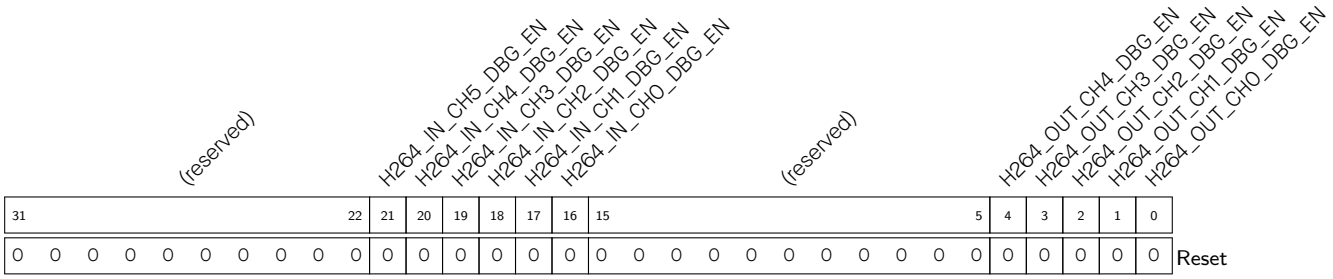
2: 1.5 bytes/pixel

3: 2 bytes/pixel

4: 3 bytes/pixel

(R/W)

Register 41.150. H264_DMA_CH_DBG_EN_REG (0x0B68)



H264_OUT_CH0_DBG_EN Configures whether to enable out channel 0 debug.

0: Disable

1: Enable

(R/W)

H264_OUT_CH1_DBG_EN Configures whether to enable out channel 1 debug.

0: Disable

1: Enable

(R/W)

H264_OUT_CH2_DBG_EN Configures whether to enable out channel 2 debug.

0: Disable

1: Enable

(R/W)

H264_OUT_CH3_DBG_EN Configures whether to enable out channel 3 debug.

0: Disable

1: Enable

(R/W)

H264_OUT_CH4_DBG_EN Configures whether to enable out channel 4 debug.

0: Disable

1: Enable

(R/W)

H264_IN_CH0_DBG_EN Configures whether to enable in channel 0 debug.

0: Disable

1: Enable

(R/W)

H264_IN_CH1_DBG_EN Configures whether to enable in channel 1 debug.

0: Disable

1: Enable

(R/W)

H264_IN_CH2_DBG_EN Configures whether to enable in channel 2 debug.

0: Disable

1: Enable

(R/W)

H264_IN_CH3_DBG_EN Configures whether to enable in channel 3 debug.

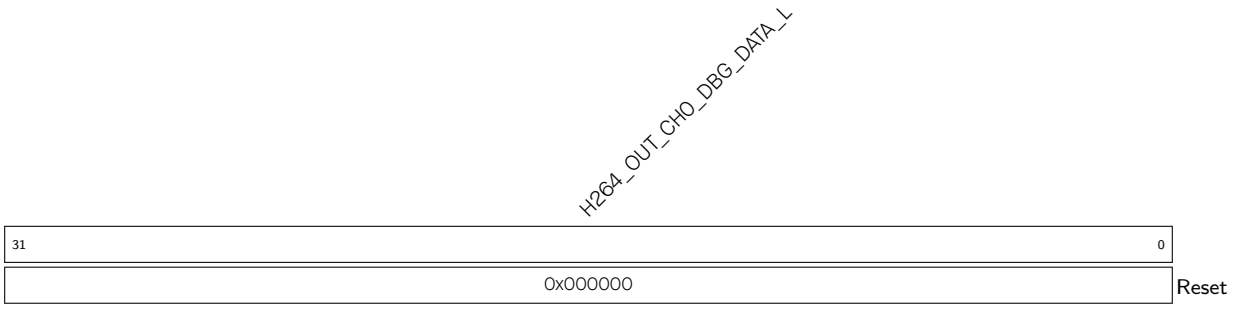
0: Disable

1: Enable

(R/W)

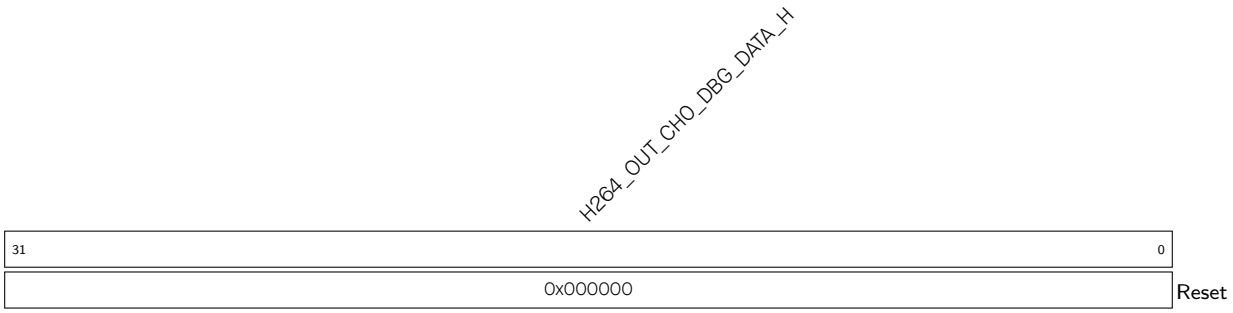
H264_IN_CH4_DBG_EN Configures whether to enable in channel 4 debug.

Register 41.151. H264_DMA_OUT_CH0_DBG_DATA_L_REG (0x0B6C)



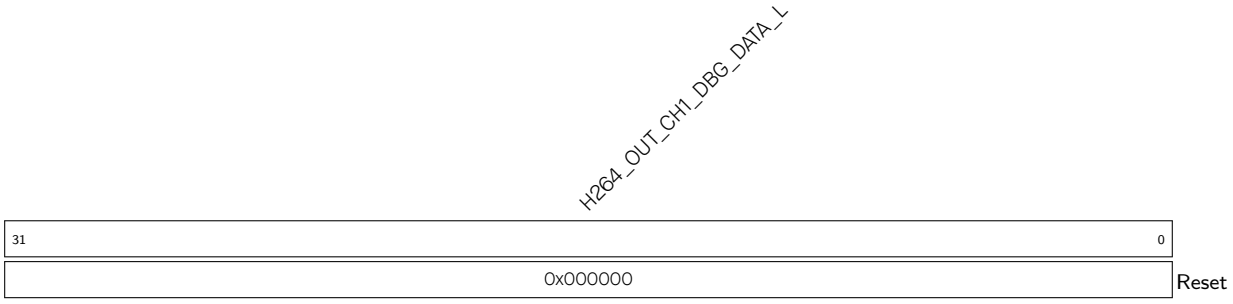
H264_OUT_CH0_DBG_DATA_L Configures out channel 0 debug data bit 31-0 (R/W)

Register 41.152. H264_DMA_OUT_CH0_DBG_DATA_H_REG (0x0B70)



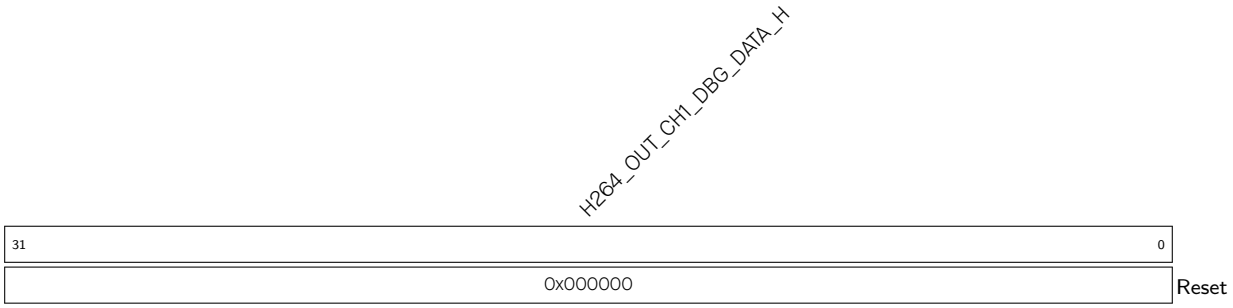
H264_OUT_CH0_DBG_DATA_H Configures out channel 0 debug data bit 63-32 (R/W)

Register 41.153. H264_DMA_OUT_CH1_DBG_DATA_L_REG (0x0B74)



H264_OUT_CH1_DBG_DATA_L Configures out channel 1 debug data bit 31-0 (R/W)

Register 41.154. H264_DMA_OUT_CH1_DBG_DATA_H_REG (0x0B78)



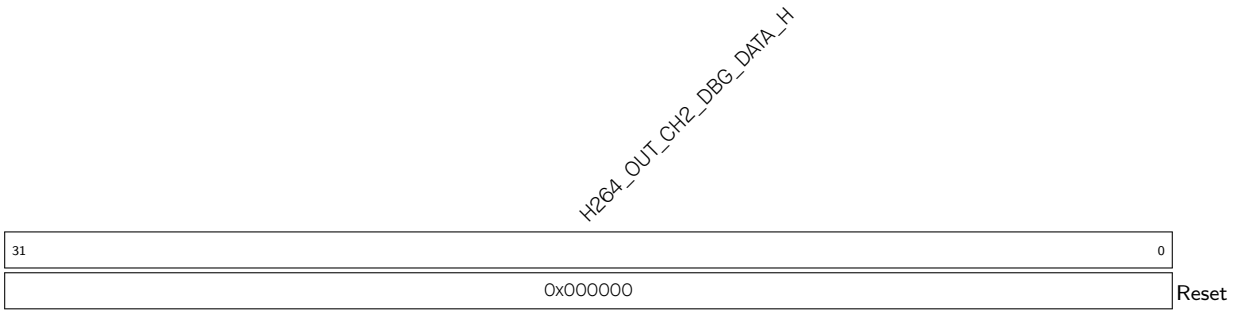
H264_OUT_CH1_DBG_DATA_H Configures out channel 1 debug data bit 63-32 (R/W)

Register 41.155. H264_DMA_OUT_CH2_DBG_DATA_L_REG (0x0B7C)



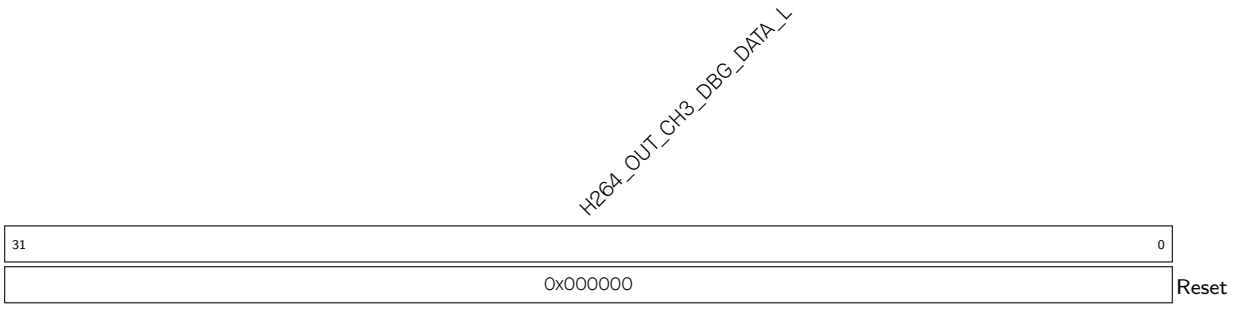
H264_OUT_CH2_DBG_DATA_L Configures out channel 2 debug data bit 31-0 (R/W)

Register 41.156. H264_DMA_OUT_CH2_DBG_DATA_H_REG (0x0B80)



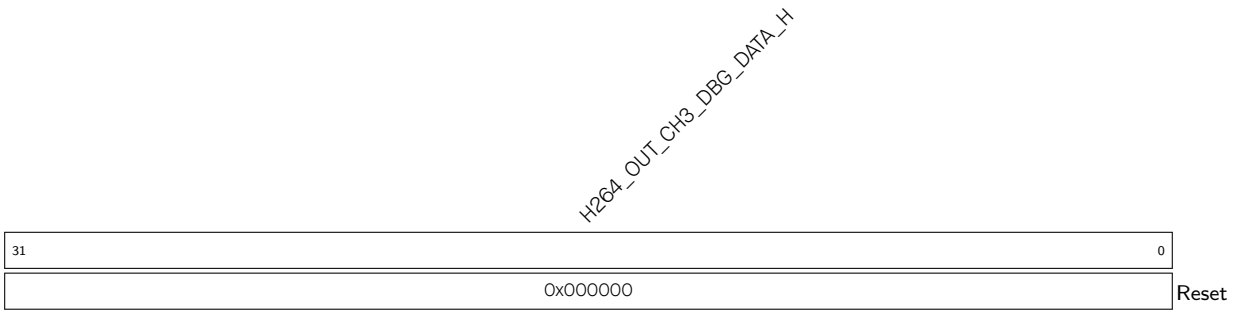
H264_OUT_CH2_DBG_DATA_H Configures out channel 2 debug data bit 63-32 (R/W)

Register 41.157. H264_DMA_OUT_CH3_DBG_DATA_L_REG (0x0B84)



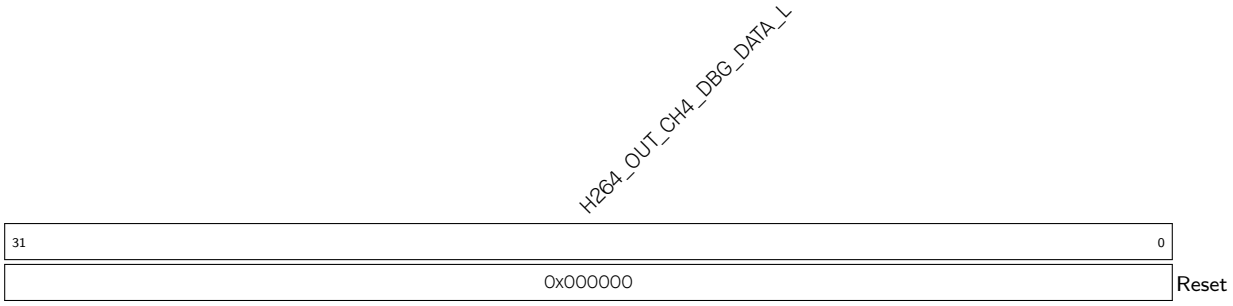
H264_OUT_CH3_DBG_DATA_L Configures out channel 3 debug data bit 31-0 (R/W)

Register 41.158. H264_DMA_OUT_CH3_DBG_DATA_H_REG (0x0B88)



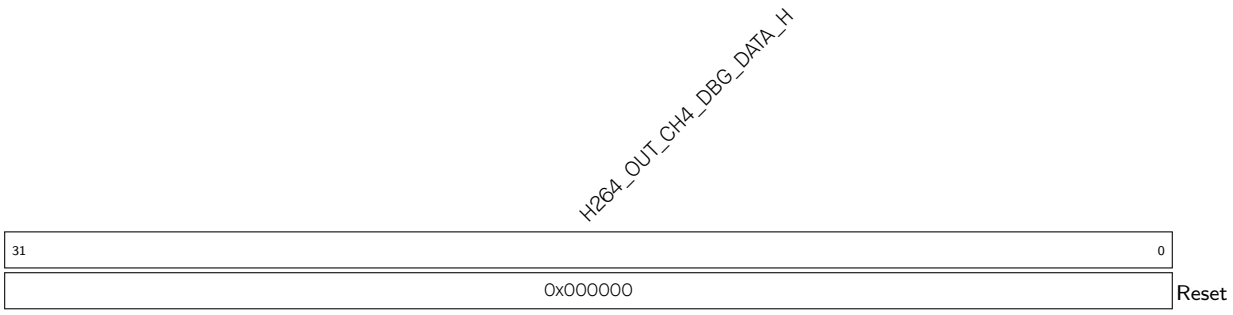
H264_OUT_CH3_DBG_DATA_H Configures out channel 3 debug data bit 63-32 (R/W)

Register 41.159. H264_DMA_OUT_CH4_DBG_DATA_L_REG (0x0B8C)



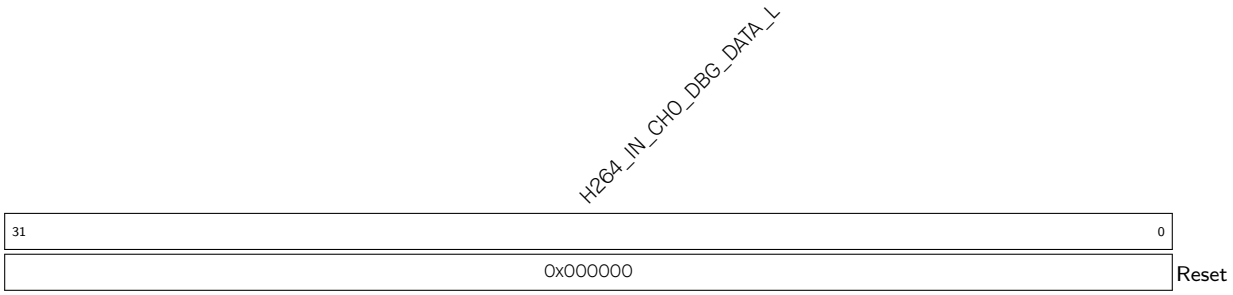
H264_OUT_CH4_DBG_DATA_L Configures out channel 4 debug data bit 31-0 (R/W)

Register 41.160. H264_DMA_OUT_CH4_DBG_DATA_H_REG (0x0B90)



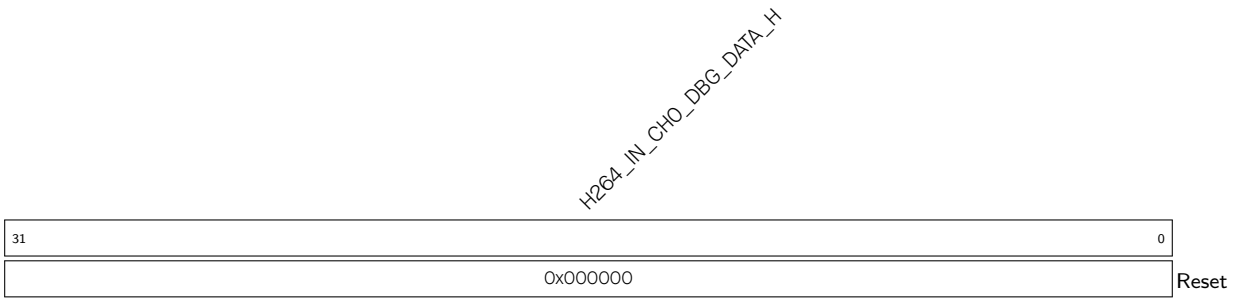
H264_OUT_CH4_DBG_DATA_H Configures out channel 4 debug data bit 63-32 (R/W)

Register 41.161. H264_DMA_IN_CH0_DBG_DATA_L_REG (0x0B94)



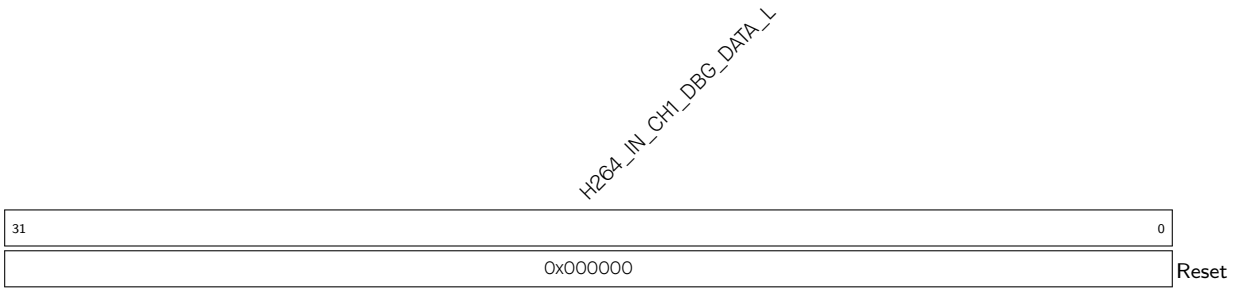
H264_IN_CH0_DBG_DATA_L Configures in channel 0 debug data bit 31-0 (R/W)

Register 41.162. H264_DMA_IN_CH0_DBG_DATA_H_REG (0x0B98)



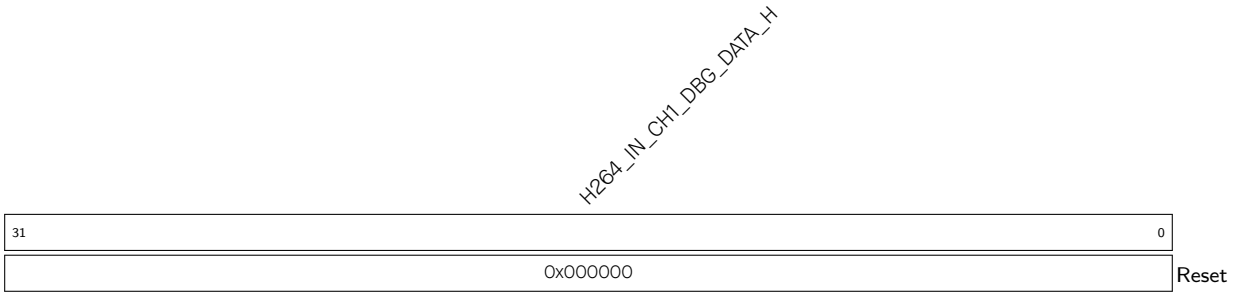
H264_IN_CH0_DBG_DATA_H Configures in channel 0 debug data bit 63-32 (R/W)

Register 41.163. H264_DMA_IN_CH1_DBG_DATA_L_REG (0x0B9C)



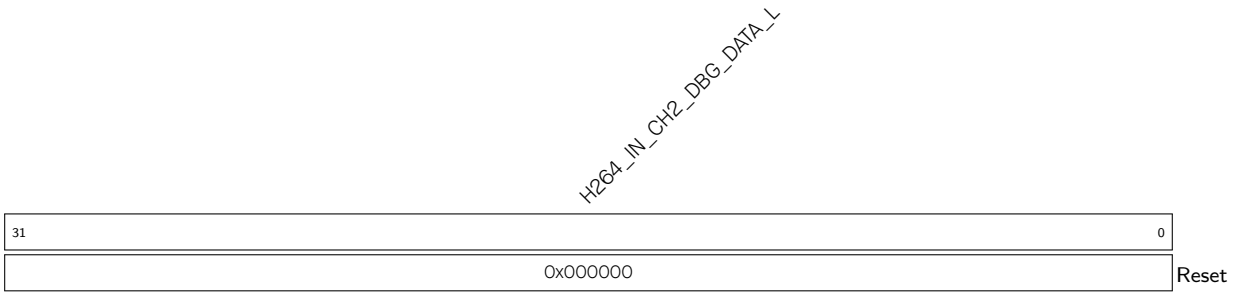
H264_IN_CH1_DBG_DATA_L Configures in channel 1 debug data bit 31-0 (R/W)

Register 41.164. H264_DMA_IN_CH1_DBG_DATA_H_REG (0x0BA0)



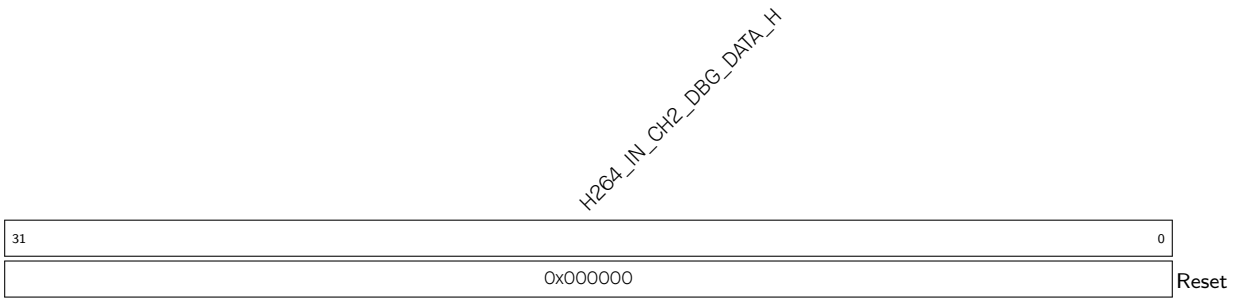
H264_IN_CH1_DBG_DATA_H Configures in channel 1 debug data bit 63-32 (R/W)

Register 41.165. H264_DMA_IN_CH2_DBG_DATA_L_REG (0x0BA4)



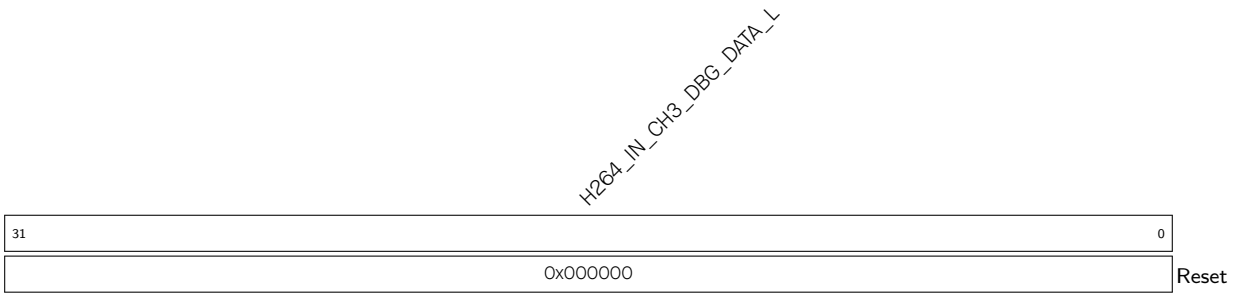
H264_IN_CH2_DBG_DATA_L Configures in channel 2 debug data bit 31-0 (R/W)

Register 41.166. H264_DMA_IN_CH2_DBG_DATA_H_REG (0x0BA8)



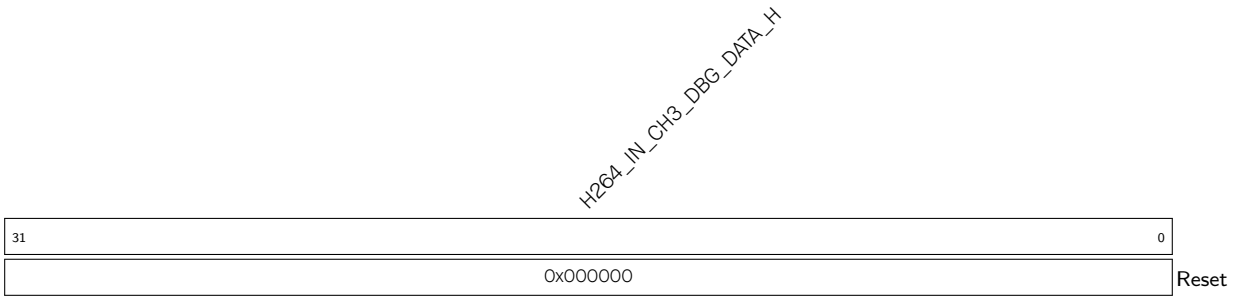
H264_IN_CH2_DBG_DATA_H Configures in channel 2 debug data bit 63-32 (R/W)

Register 41.167. H264_DMA_IN_CH3_DBG_DATA_L_REG (0x0BAC)



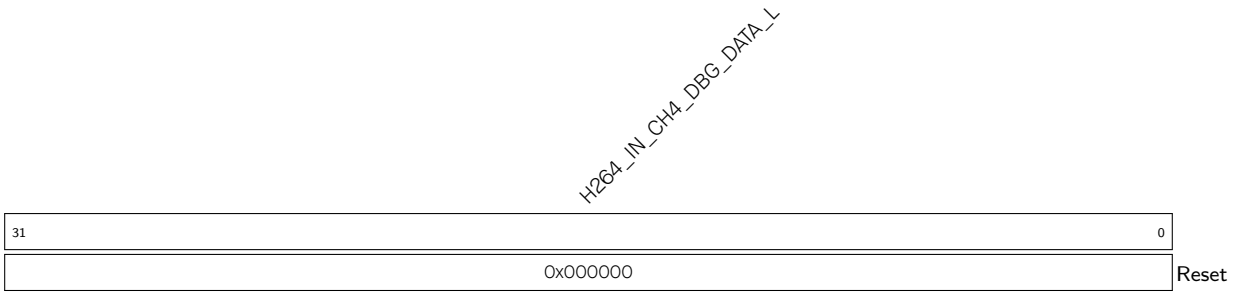
H264_IN_CH3_DBG_DATA_L Configures in channel 3 debug data bit 31-0 (R/W)

Register 41.168. H264_DMA_IN_CH3_DBG_DATA_H_REG (0x0BB0)



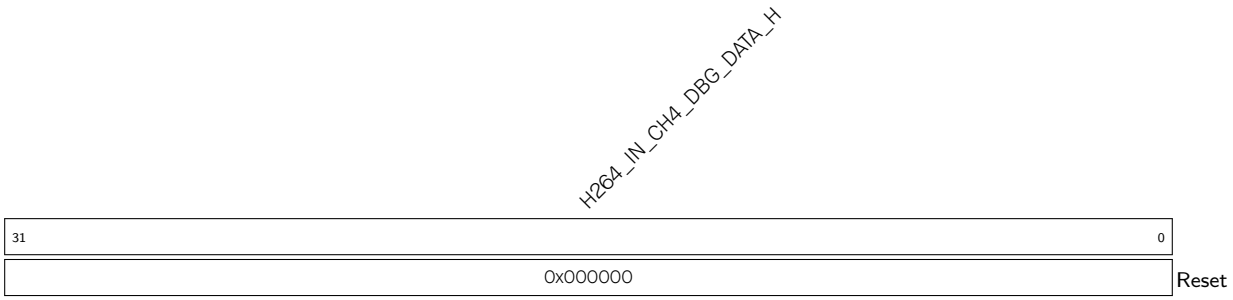
H264_IN_CH3_DBG_DATA_H Configures in channel 3 debug data bit 63-32 (R/W)

Register 41.169. H264_DMA_IN_CH4_DBG_DATA_L_REG (0x0BB4)



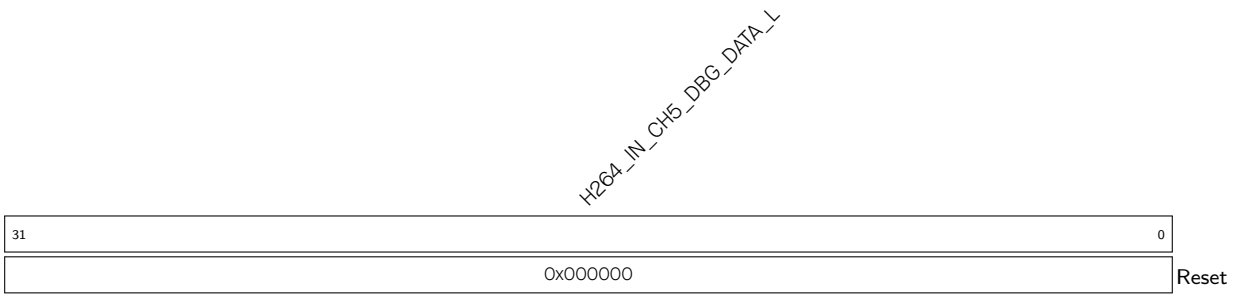
H264_IN_CH4_DBG_DATA_L Configures in channel 4 debug data bit 31-0 (R/W)

Register 41.170. H264_DMA_IN_CH4_DBG_DATA_H_REG (0x0BB8)



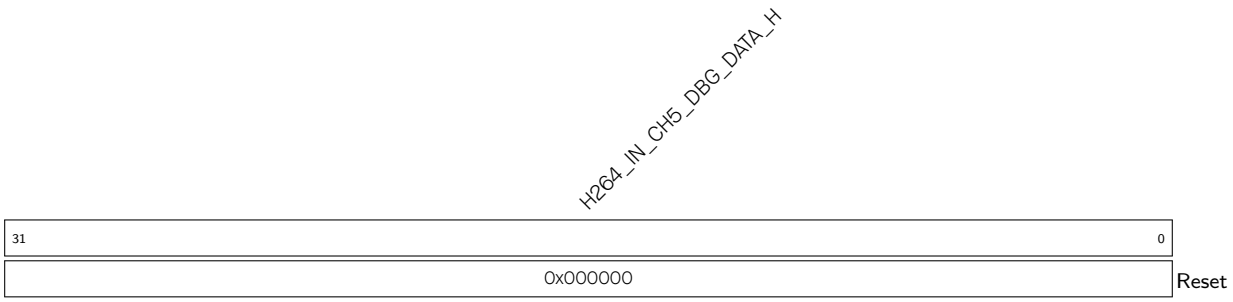
H264_IN_CH4_DBG_DATA_H Configures in channel 4 debug data bit 63-32 (R/W)

Register 41.171. H264_DMA_IN_CH5_DBG_DATA_L_REG (0x0BBC)



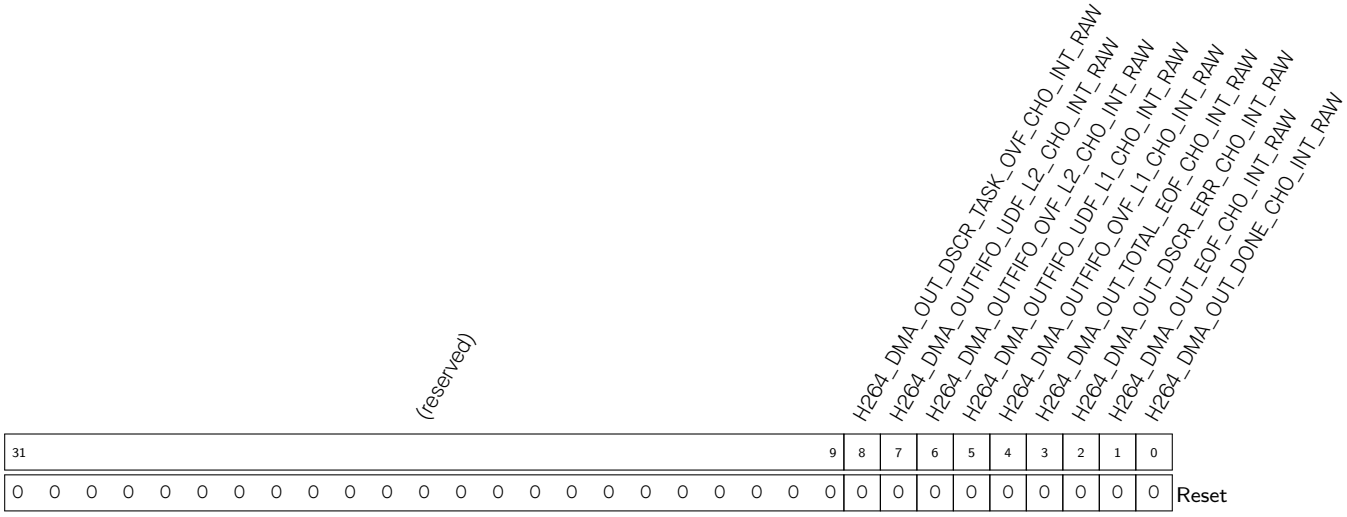
H264_IN_CH5_DBG_DATA_L Configures in channel 5 debug data bit 31-0 (R/W)

Register 41.172. H264_DMA_IN_CH5_DBG_DATA_H_REG (0x0BC0)



H264_IN_CH5_DBG_DATA_H Configures in channel 5 debug data bit 63-32 (R/W)

Register 41.173. H264_DMA_OUT_INT_RAW_CH0_REG (0x0004)



H264_DMA_OUT_DONE_CH0_INT_RAW The raw interrupt status of H264_DMA_OUT_DONE_CH0_INT. (R/WTC/SS)

H264_DMA_OUT_EOF_CH0_INT_RAW The raw interrupt status of H264_DMA_OUT_EOF_CH0_INT. (R/WTC/SS)

H264_DMA_OUT_DSCR_ERR_CH0_INT_RAW The raw interrupt status of H264_DMA_OUT_DSCR_ERR_CH0_INT. (R/WTC/SS)

H264_DMA_OUT_TOTAL_EOF_CH0_INT_RAW The raw interrupt status of H264_DMA_OUT_TOTAL_EOF_CH0_INT. (R/WTC/SS)

H264_DMA_OUTFIFO_OVF_L1_CH0_INT_RAW The raw interrupt status of H264_DMA_OUTFIFO_OVF_L1_CH0_INT. (R/WTC/SS)

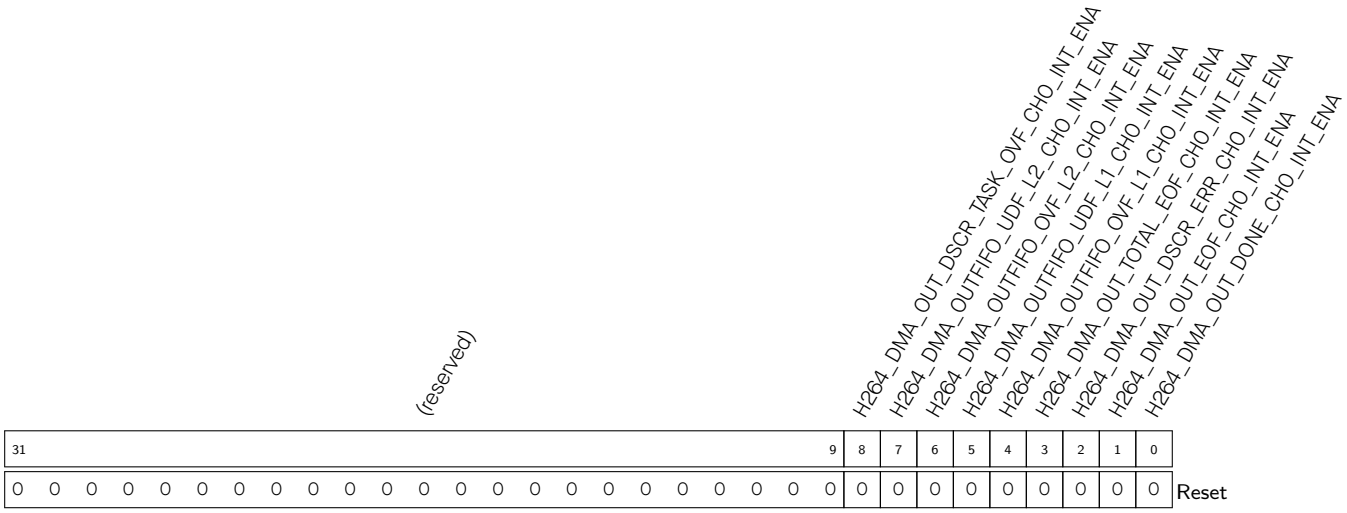
H264_DMA_OUTFIFO_UDF_L1_CH0_INT_RAW The raw interrupt status of H264_DMA_OUTFIFO_UDF_L1_CH0_INT. (R/WTC/SS)

H264_DMA_OUTFIFO_OVF_L2_CH0_INT_RAW The raw interrupt status of H264_DMA_OUTFIFO_OVF_L2_CH0_INT. (R/WTC/SS)

H264_DMA_OUTFIFO_UDF_L2_CH0_INT_RAW The raw interrupt status of H264_DMA_OUTFIFO_UDF_L2_CH0_INT. (R/WTC/SS)

H264_DMA_OUT_DSCR_TASK_OVF_CH0_INT_RAW The raw interrupt status of H264_DMA_OUT_DSCR_TASK_OVF_CH0_INT. (R/WTC/SS)

Register 41.174. H264_DMA_OUT_INT_ENA_CH0_REG (0x0008)



H264_DMA_OUT_DONE_CH0_INT_ENA Write 1 to enable H264_DMA_OUT_DONE_CH0_INT. (R/W)

H264_DMA_OUT_EOF_CH0_INT_ENA Write 1 to enable H264_DMA_OUT_EOF_CH0_INT. (R/W)

H264_DMA_OUT_DSCR_ERR_CH0_INT_ENA Write 1 to enable H264_DMA_OUT_DSCR_ERR_-CH0_INT. (R/W)

H264_DMA_OUT_TOTAL_EOF_CH0_INT_ENA Write 1 to enable H264_DMA_OUT_TOTAL_EOF_-CH0_INT. (R/W)

H264_DMA_OUTFIFO_OVF_L1_CH0_INT_ENA Write 1 to enable H264_DMA_OUTFIFO_OVF_L1_-CH0_INT. (R/W)

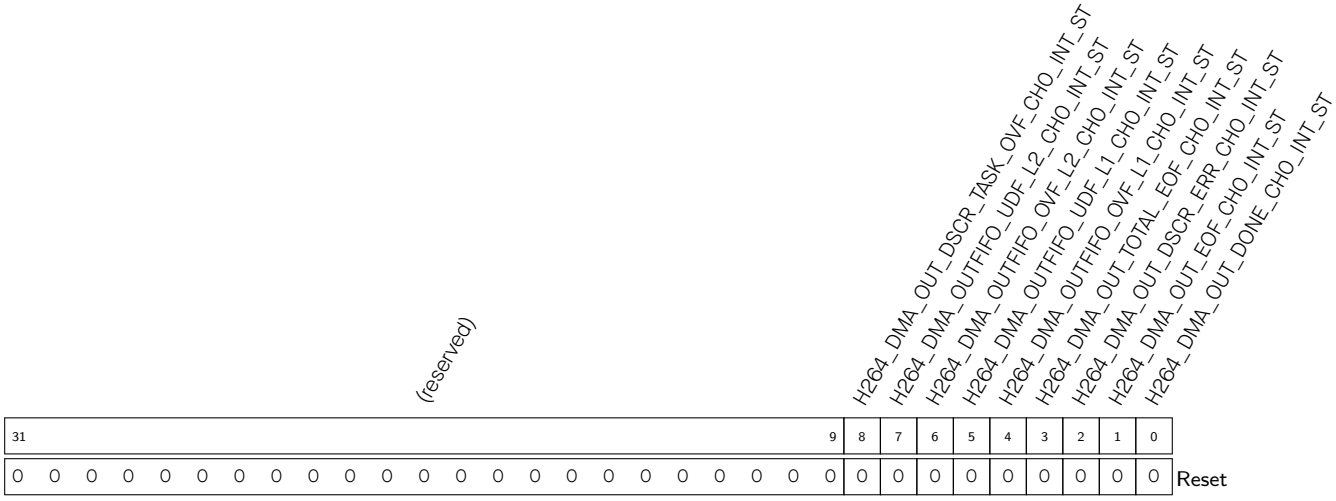
H264_DMA_OUTFIFO_UDF_L1_CH0_INT_ENA Write 1 to enable H264_DMA_OUTFIFO_UDF_L1_-CH0_INT. (R/W)

H264_DMA_OUTFIFO_OVF_L2_CH0_INT_ENA Write 1 to enable H264_DMA_OUTFIFO_OVF_L2_-CH0_INT. (R/W)

H264_DMA_OUTFIFO_UDF_L2_CH0_INT_ENA Write 1 to enable H264_DMA_OUTFIFO_UDF_L2_-CH0_INT. (R/W)

H264_DMA_OUT_DSCR_TASK_OVF_CH0_INT_ENA Write 1 to enable H264_DMA_OUT_DSCR_-TASK_OVF_CH0_INT. (R/W)

Register 41.175. H264_DMA_OUT_INT_ST_CH0_REG (0x000C)



H264_DMA_OUT_DONE_CH0_INT_ST The masked interrupt status of H264_DMA_OUT_DONE_CH0_INT. (RO)

H264_DMA_OUT_EOF_CH0_INT_ST The masked interrupt status of H264_DMA_OUT_EOF_CH0_INT. (RO)

H264_DMA_OUT_DSCR_ERR_CH0_INT_ST The masked interrupt status of H264_DMA_OUT_DSCR_ERR_CH0_INT. (RO)

H264_DMA_OUT_TOTAL_EOF_CH0_INT_ST The masked interrupt status of H264_DMA_OUT_TOTAL_EOF_CH0_INT. (RO)

H264_DMA_OUTFIFO_OVF_L1_CH0_INT_ST The masked interrupt status of H264_DMA_OUTFIFO_OVF_L1_CH0_INT. (RO)

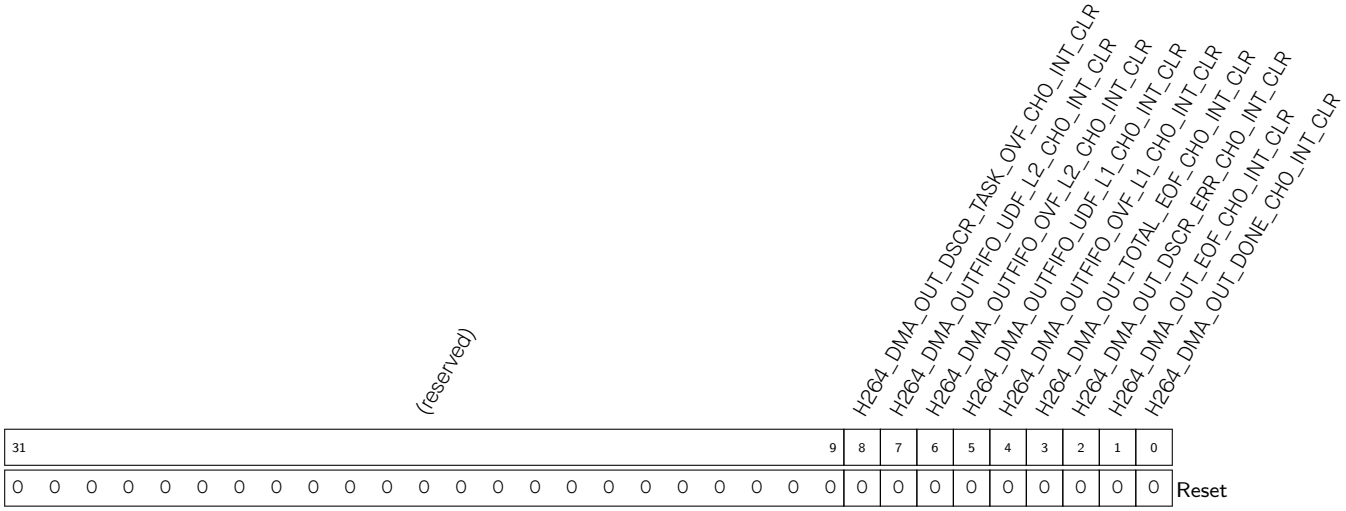
H264_DMA_OUTFIFO_UDF_L1_CH0_INT_ST The masked interrupt status of H264_DMA_OUTFIFO_UDF_L1_CH0_INT. (RO)

H264_DMA_OUTFIFO_OVF_L2_CH0_INT_ST The masked interrupt status of H264_DMA_OUTFIFO_OVF_L2_CH0_INT. (RO)

H264_DMA_OUTFIFO_UDF_L2_CH0_INT_ST The masked interrupt status of H264_DMA_OUTFIFO_UDF_L2_CH0_INT. (RO)

H264_DMA_OUT_DSCR_TASK_OVF_CH0_INT_ST The masked interrupt status of H264_DMA_OUT_DSCR_TASK_OVF_CH0_INT. (RO)

Register 41.176. H264_DMA_OUT_INT_CLR_CH0_REG (0x0010)



H264_DMA_OUT_DONE_CH0_INT_CLR Write 1 to clear H264_DMA_OUT_DONE_CH0_INT. (WT)

H264_DMA_OUT_EOF_CH0_INT_CLR Write 1 to clear H264_DMA_OUT_EOF_CH0_INT. (WT)

H264_DMA_OUT_DSCR_ERR_CH0_INT_CLR Write 1 to clear H264_DMA_OUT_DSCR_ERR_CH0_-INT. (WT)

H264_DMA_OUT_TOTAL_EOF_CH0_INT_CLR Write 1 to clear H264_DMA_OUT_TOTAL_EOF_-CH0_INT. (WT)

H264_DMA_OUTFIFO_OVF_L1_CH0_INT_CLR Write 1 to clear H264_DMA_OUTFIFO_OVF_L1_-CH0_INT. (WT)

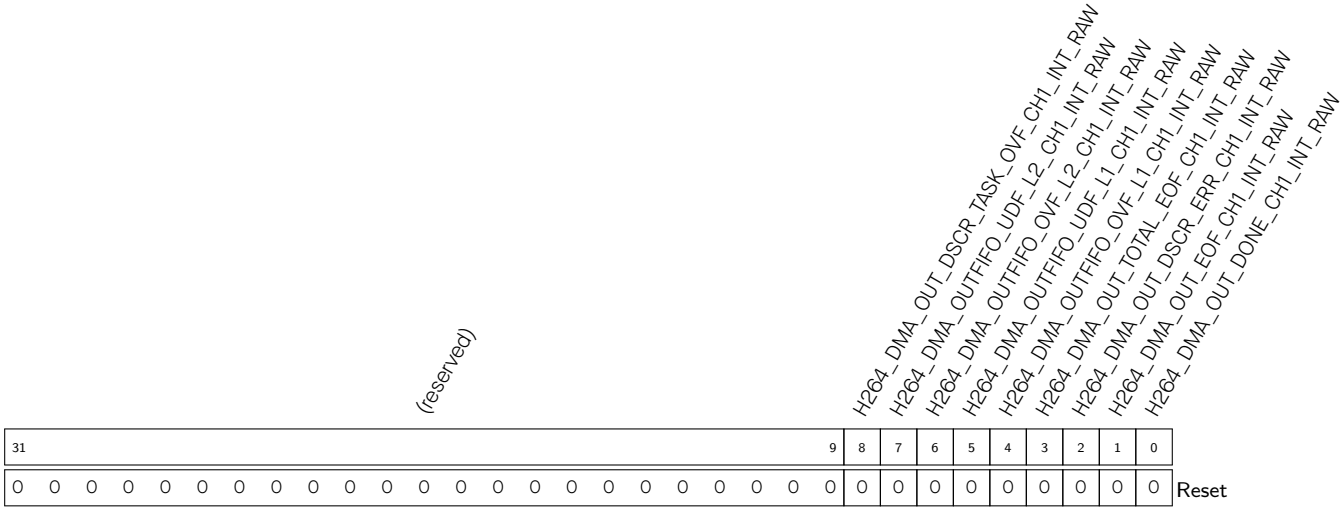
H264_DMA_OUTFIFO_UDF_L1_CH0_INT_CLR Write 1 to clear H264_DMA_OUTFIFO_UDF_L1_-CH0_INT. (WT)

H264_DMA_OUTFIFO_OVF_L2_CH0_INT_CLR Write 1 to clear H264_DMA_OUTFIFO_OVF_L2_-CH0_INT. (WT)

H264_DMA_OUTFIFO_UDF_L2_CH0_INT_CLR Write 1 to clear H264_DMA_OUTFIFO_UDF_L2_-CH0_INT. (WT)

H264_DMA_OUT_DSCR_TASK_OVF_CH0_INT_CLR Write 1 to clear H264_DMA_OUT_DSCR_-TASK_OVF_CH0_INT. (WT)

Register 41.177. H264_DMA_OUT_INT_RAW_CH1_REG (0x0104)



H264_DMA_OUT_DONE_CH1_INT_RAW The raw interrupt status of H264_DMA_OUT_DONE_CH1_INT. (R/WTC/SS)

H264_DMA_OUT_EOF_CH1_INT_RAW The raw interrupt status of H264_DMA_OUT_EOF_CH1_INT. (R/WTC/SS)

H264_DMA_OUT_DSCR_ERR_CH1_INT_RAW The raw interrupt status of H264_DMA_OUT_DSCR_ERR_CH1_INT. (R/WTC/SS)

H264_DMA_OUT_TOTAL_EOF_CH1_INT_RAW The raw interrupt status of H264_DMA_OUT_TOTAL_EOF_CH1_INT. (R/WTC/SS)

H264_DMA_OUTFIFO_OVF_L1_CH1_INT_RAW The raw interrupt status of H264_DMA_OUTFIFO_OVF_L1_CH1_INT. (R/WTC/SS)

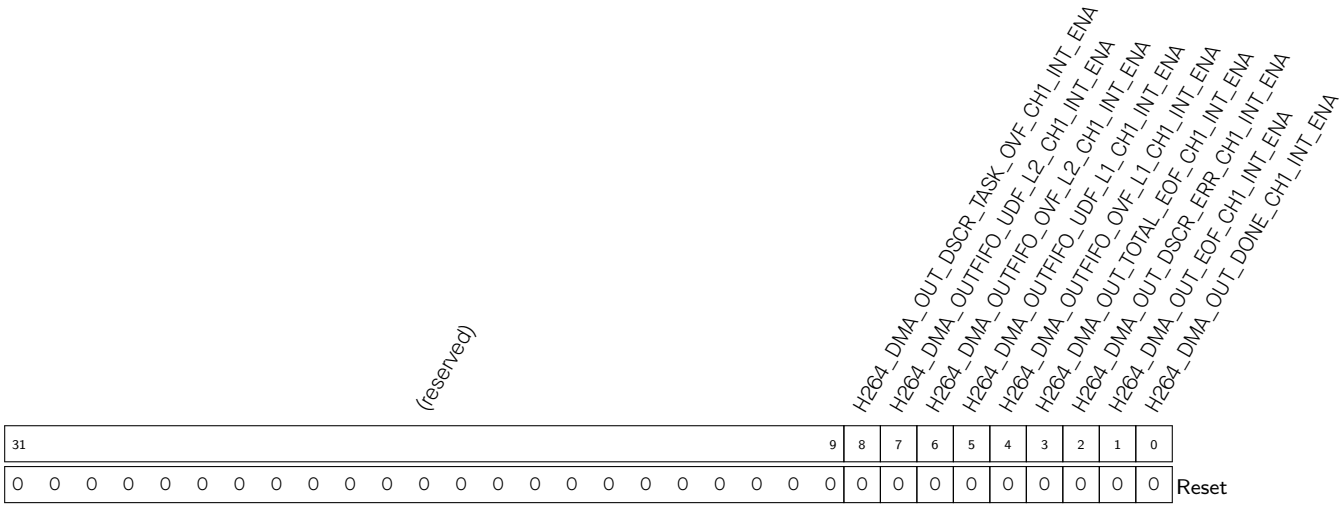
H264_DMA_OUTFIFO_UDF_L1_CH1_INT_RAW The raw interrupt status of H264_DMA_OUTFIFO_UDF_L1_CH1_INT. (R/WTC/SS)

H264_DMA_OUTFIFO_OVF_L2_CH1_INT_RAW The raw interrupt status of H264_DMA_OUTFIFO_OVF_L2_CH1_INT. (R/WTC/SS)

H264_DMA_OUTFIFO_UDF_L2_CH1_INT_RAW The raw interrupt status of H264_DMA_OUTFIFO_UDF_L2_CH1_INT. (R/WTC/SS)

H264_DMA_OUT_DSCR_TASK_OVF_CH1_INT_RAW The raw interrupt status of H264_DMA_OUT_DSCR_TASK_OVF_CH1_INT. (R/WTC/SS)

Register 41.178. H264_DMA_OUT_INT_ENA_CH1_REG (0x0108)



H264_DMA_OUT_DONE_CH1_INT_ENA Write 1 to enable H264_DMA_OUT_DONE_CH1_INT. (R/W)

H264_DMA_OUT_EOF_CH1_INT_ENA Write 1 to enable H264_DMA_OUT_EOF_CH1_INT. (R/W)

H264_DMA_OUT_DSCR_ERR_CH1_INT_ENA Write 1 to enable H264_DMA_OUT_DSCR_ERR_CH1_INT. (R/W)

H264_DMA_OUT_TOTAL_EOF_CH1_INT_ENA Write 1 to enable H264_DMA_OUT_TOTAL_EOF_CH1_INT. (R/W)

H264_DMA_OUTFIFO_OVF_L1_CH1_INT_ENA Write 1 to enable H264_DMA_OUTFIFO_OVF_L1_CH1_INT. (R/W)

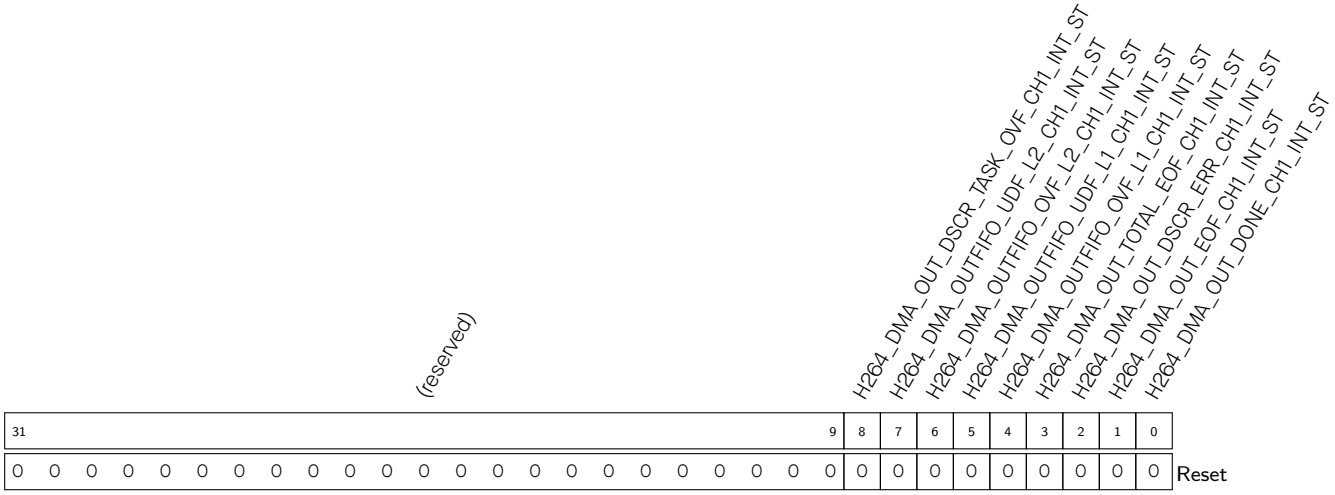
H264_DMA_OUTFIFO_UDF_L1_CH1_INT_ENA Write 1 to enable H264_DMA_OUTFIFO_UDF_L1_CH1_INT. (R/W)

H264_DMA_OUTFIFO_OVF_L2_CH1_INT_ENA Write 1 to enable H264_DMA_OUTFIFO_OVF_L2_CH1_INT. (R/W)

H264_DMA_OUTFIFO_UDF_L2_CH1_INT_ENA Write 1 to enable H264_DMA_OUTFIFO_UDF_L2_CH1_INT. (R/W)

H264_DMA_OUT_DSCR_TASK_OVF_CH1_INT_ENA Write 1 to enable H264_DMA_OUT_DSCR_TASK_OVF_CH1_INT. (R/W)

Register 41.179. H264_DMA_OUT_INT_ST_CH1_REG (0x010C)



H264_DMA_OUT_DONE_CH1_INT_ST The masked interrupt status of H264_DMA_OUT_DONE_CH1_INT. (RO)

H264_DMA_OUT_EOF_CH1_INT_ST The masked interrupt status of H264_DMA_OUT_EOF_CH1_INT. (RO)

H264_DMA_OUT_DSCR_ERR_CH1_INT_ST The masked interrupt status of H264_DMA_OUT_DSCR_ERR_CH1_INT. (RO)

H264_DMA_OUT_TOTAL_EOF_CH1_INT_ST The masked interrupt status of H264_DMA_OUT_TOTAL_EOF_CH1_INT. (RO)

H264_DMA_OUTFIFO_OVF_L1_CH1_INT_ST The masked interrupt status of H264_DMA_OUTFIFO_OVF_L1_CH1_INT. (RO)

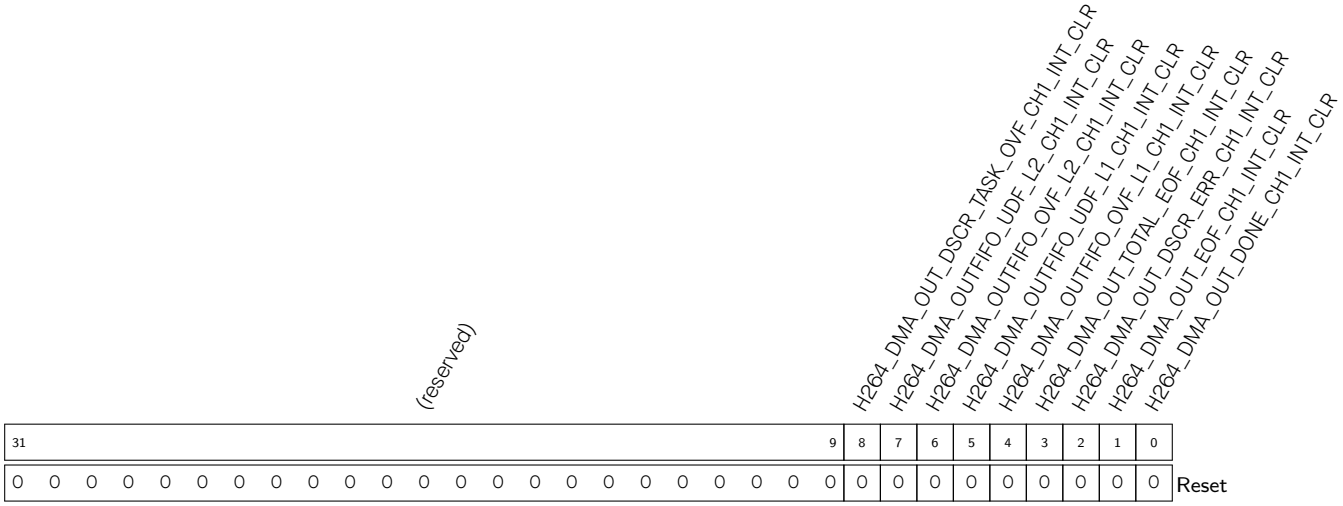
H264_DMA_OUTFIFO_UDF_L1_CH1_INT_ST The masked interrupt status of H264_DMA_OUTFIFO_UDF_L1_CH1_INT. (RO)

H264_DMA_OUTFIFO_OVF_L2_CH1_INT_ST The masked interrupt status of H264_DMA_OUTFIFO_OVF_L2_CH1_INT. (RO)

H264_DMA_OUTFIFO_UDF_L2_CH1_INT_ST The masked interrupt status of H264_DMA_OUTFIFO_UDF_L2_CH1_INT. (RO)

H264_DMA_OUT_DSCR_TASK_OVF_CH1_INT_ST The masked interrupt status of H264_DMA_OUT_DSCR_TASK_OVF_CH1_INT. (RO)

Register 41.180. H264_DMA_OUT_INT_CLR_CH1_REG (0x0110)



H264_DMA_OUT_DONE_CH1_INT_CLR Write 1 to clear H264_DMA_OUT_DONE_CH1_INT. (WT)

H264_DMA_OUT_EOF_CH1_INT_CLR Write 1 to clear H264_DMA_OUT_EOF_CH1_INT. (WT)

H264_DMA_OUT_DSCR_ERR_CH1_INT_CLR Write 1 to clear H264_DMA_OUT_DSCR_ERR_CH1_INT. (WT)

H264_DMA_OUT_TOTAL_EOF_CH1_INT_CLR Write 1 to clear H264_DMA_OUT_TOTAL_EOF_CH1_INT. (WT)

H264_DMA_OUTFIFO_OVF_L1_CH1_INT_CLR Write 1 to clear H264_DMA_OUTFIFO_OVF_L1_CH1_INT. (WT)

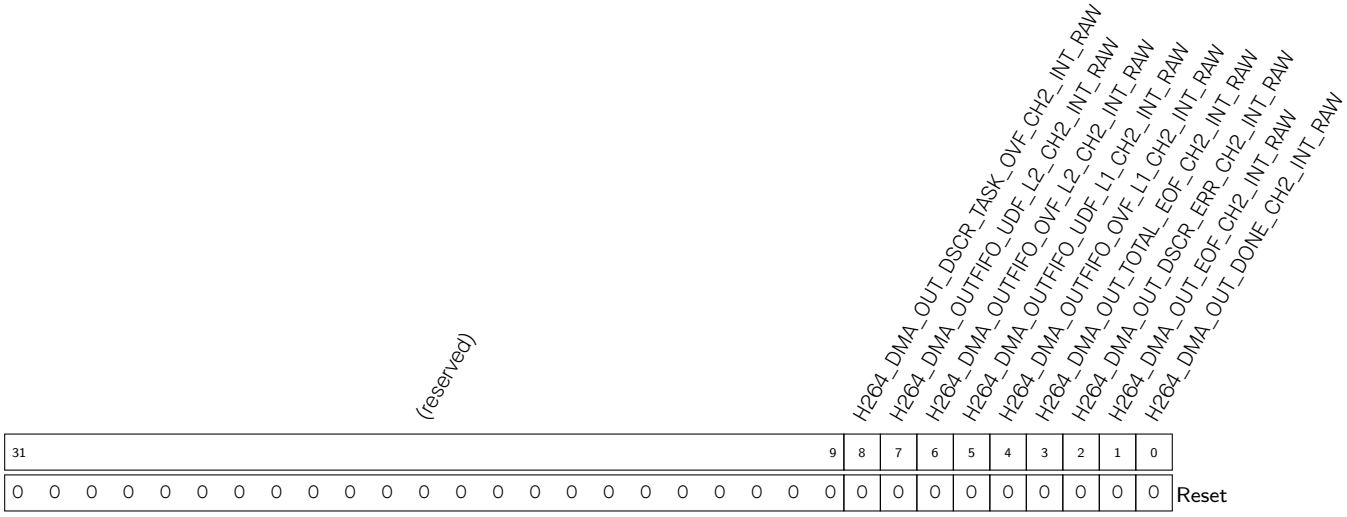
H264_DMA_OUTFIFO_UDF_L1_CH1_INT_CLR Write 1 to clear H264_DMA_OUTFIFO_UDF_L1_CH1_INT. (WT)

H264_DMA_OUTFIFO_OVF_L2_CH1_INT_CLR Write 1 to clear H264_DMA_OUTFIFO_OVF_L2_CH1_INT. (WT)

H264_DMA_OUTFIFO_UDF_L2_CH1_INT_CLR Write 1 to clear H264_DMA_OUTFIFO_UDF_L2_CH1_INT. (WT)

H264_DMA_OUT_DSCR_TASK_OVF_CH1_INT_CLR Write 1 to clear H264_DMA_OUT_DSCR_TASK_OVF_CH1_INT. (WT)

Register 41.181. H264_DMA_OUT_INT_RAW_CH2_REG (0x0204)



H264_DMA_OUT_DONE_CH2_INT_RAW The raw interrupt status of H264_DMA_OUT_DONE_CH2_INT. Triggered when all data corresponding to a transmit descriptor has been sent via TX channel 2. (R/WTC/SS)

H264_DMA_OUT_EOF_CH2_INT_RAW The raw interrupt status of H264_DMA_OUT_EOF_CH2_INT. (R/WTC/SS)

H264_DMA_OUT_DSCR_ERR_CH2_INT_RAW The raw interrupt status of H264_DMA_OUT_DSCR_ERR_CH2_INT. (R/WTC/SS)

H264_DMA_OUT_TOTAL_EOF_CH2_INT_RAW The raw interrupt status of H264_DMA_OUT_TOTAL_EOF_CH2_INT. (R/WTC/SS)

H264_DMA_OUTFIFO_OVF_L1_CH2_INT_RAW The raw interrupt status of H264_DMA_OUTFIFO_OVF_L1_CH2_INT. (R/WTC/SS)

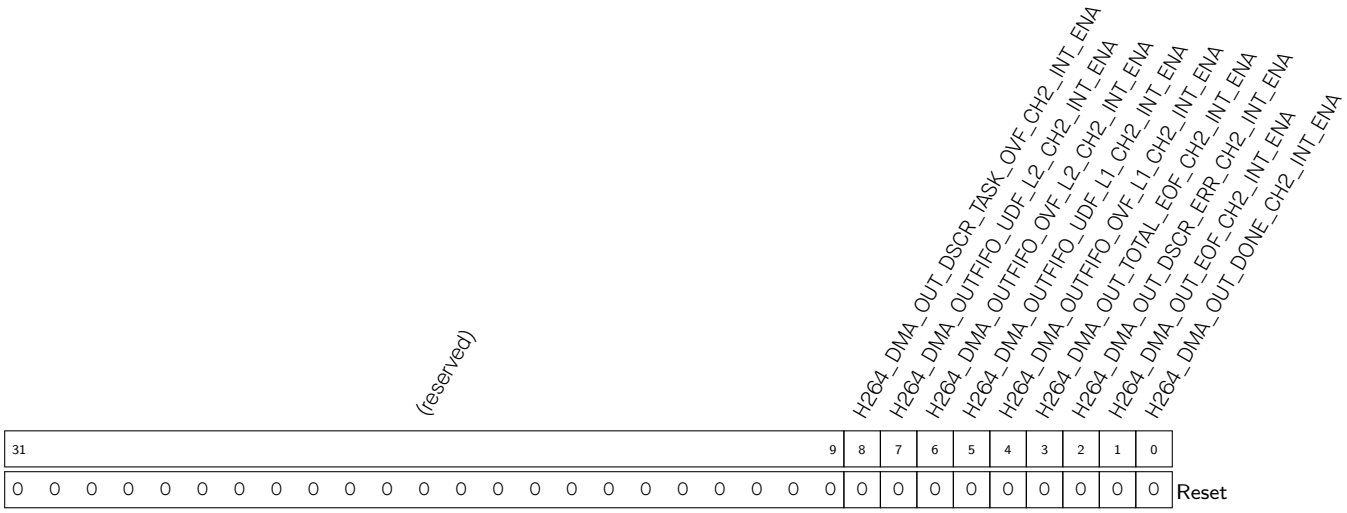
H264_DMA_OUTFIFO_UDF_L1_CH2_INT_RAW The raw interrupt status of H264_DMA_OUTFIFO_UDF_L1_CH2_INT. (R/WTC/SS)

H264_DMA_OUTFIFO_OVF_L2_CH2_INT_RAW The raw interrupt status of H264_DMA_OUTFIFO_OVF_L2_CH2_INT. (R/WTC/SS)

H264_DMA_OUTFIFO_UDF_L2_CH2_INT_RAW The raw interrupt status of H264_DMA_OUTFIFO_UDF_L2_CH2_INT. (R/WTC/SS)

H264_DMA_OUT_DSCR_TASK_OVF_CH2_INT_RAW The raw interrupt status of H264_DMA_OUT_DSCR_TASK_OVF_CH2_INT. (R/WTC/SS)

Register 41.182. H264_DMA_OUT_INT_ENA_CH2_REG (0x0208)



H264_DMA_OUT_DONE_CH2_INT_ENA Write 1 to enable H264_DMA_OUT_DONE_CH2_INT. (R/W)

H264_DMA_OUT_EOF_CH2_INT_ENA Write 1 to enable H264_DMA_OUT_EOF_CH2_INT. (R/W)

H264_DMA_OUT_DSCR_ERR_CH2_INT_ENA Write 1 to enable H264_DMA_OUT_DSCR_ERR_CH2_INT. (R/W)

H264_DMA_OUT_TOTAL_EOF_CH2_INT_ENA Write 1 to enable H264_DMA_OUT_TOTAL_EOF_CH2_INT. (R/W)

H264_DMA_OUTFIFO_OVF_L1_CH2_INT_ENA Write 1 to enable H264_DMA_OUTFIFO_OVF_L1_CH2_INT. (R/W)

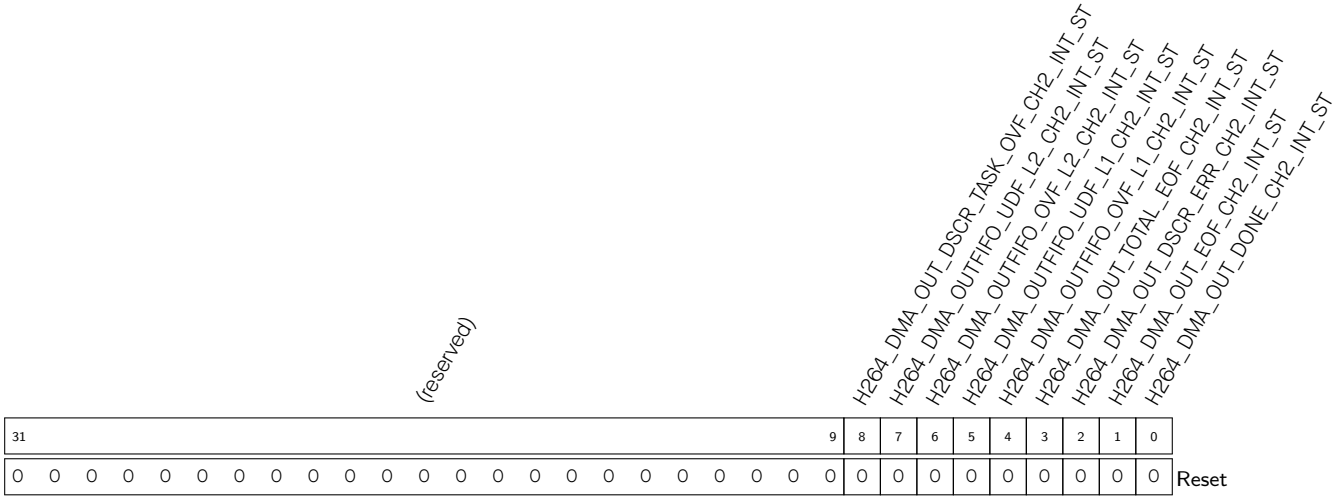
H264_DMA_OUTFIFO_UDF_L1_CH2_INT_ENA Write 1 to enable H264_DMA_OUTFIFO_UDF_L1_CH2_INT. (R/W)

H264_DMA_OUTFIFO_OVF_L2_CH2_INT_ENA Write 1 to enable H264_DMA_OUTFIFO_OVF_L2_CH2_INT. (R/W)

H264_DMA_OUTFIFO_UDF_L2_CH2_INT_ENA Write 1 to enable H264_DMA_OUTFIFO_UDF_L2_CH2_INT. (R/W)

H264_DMA_OUT_DSCR_TASK_OVF_CH2_INT_ENA Write 1 to enable H264_DMA_OUT_DSCR_TASK_OVF_CH2_INT. (R/W)

Register 41.183. H264_DMA_OUT_INT_ST_CH2_REG (0x020C)



H264_DMA_OUT_DONE_CH2_INT_ST The masked interrupt status of H264_DMA_OUT_DONE_CH2_INT. (RO)

H264_DMA_OUT_EOF_CH2_INT_ST The masked interrupt status of H264_DMA_OUT_EOF_CH2_INT. (RO)

H264_DMA_OUT_DSCR_ERR_CH2_INT_ST The masked interrupt status of H264_DMA_OUT_DSCR_ERR_CH2_INT. (RO)

H264_DMA_OUT_TOTAL_EOF_CH2_INT_ST The masked interrupt status of H264_DMA_OUT_TOTAL_EOF_CH2_INT. (RO)

H264_DMA_OUTFIFO_OVF_L1_CH2_INT_ST The masked interrupt status of H264_DMA_OUTFIFO_OVF_L1_CH2_INT. (RO)

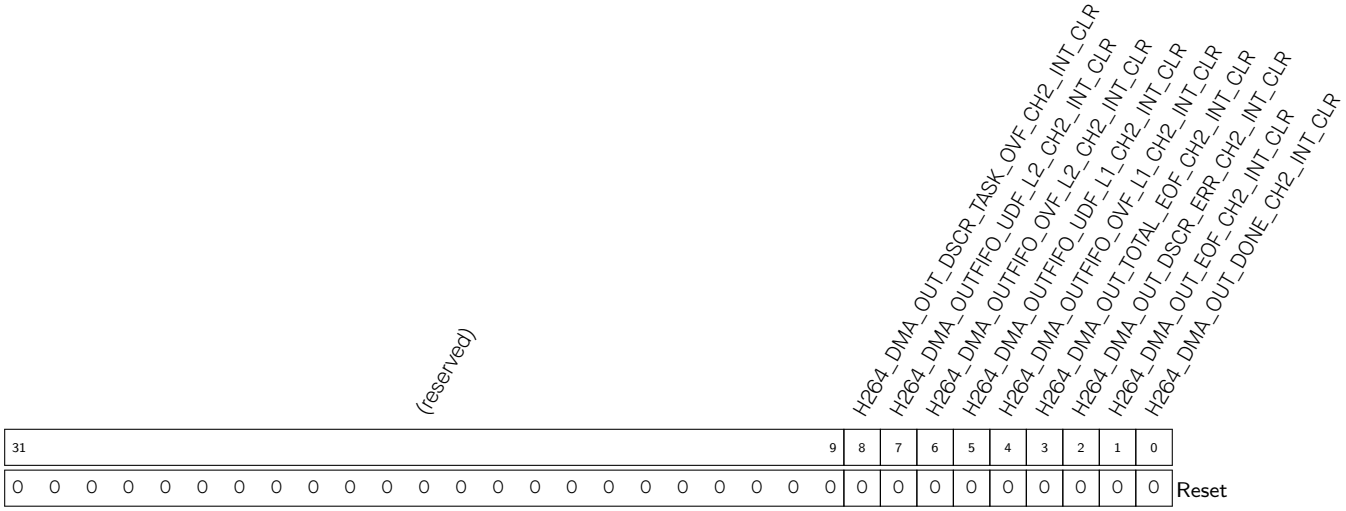
H264_DMA_OUTFIFO_UDF_L1_CH2_INT_ST The masked interrupt status of H264_DMA_OUTFIFO_UDF_L1_CH2_INT. (RO)

H264_DMA_OUTFIFO_OVF_L2_CH2_INT_ST The masked interrupt status of H264_DMA_OUTFIFO_OVF_L2_CH2_INT. (RO)

H264_DMA_OUTFIFO_UDF_L2_CH2_INT_ST The masked interrupt status of H264_DMA_OUTFIFO_UDF_L2_CH2_INT. (RO)

H264_DMA_OUT_DSCR_TASK_OVF_CH2_INT_ST The masked interrupt status of H264_DMA_OUT_DSCR_TASK_OVF_CH2_INT. (RO)

Register 41.184. H264_DMA_OUT_INT_CLR_CH2_REG (0x0210)



H264_DMA_OUT_DONE_CH2_INT_CLR Write 1 to clear H264_DMA_OUT_DONE_CH2_INT. (WT)

H264_DMA_OUT_EOF_CH2_INT_CLR Write 1 to clear H264_DMA_OUT_EOF_CH2_INT. (WT)

H264_DMA_OUT_DSCR_ERR_CH2_INT_CLR Write 1 to clear H264_DMA_OUT_DSCR_ERR_CH2_-INT. (WT)

H264_DMA_OUT_TOTAL_EOF_CH2_INT_CLR Write 1 to clear H264_DMA_OUT_TOTAL_EOF_-CH2_INT. (WT)

H264_DMA_OUTFIFO_OVF_L1_CH2_INT_CLR Write 1 to clear H264_DMA_OUTFIFO_OVF_L1_-CH2_INT. (WT)

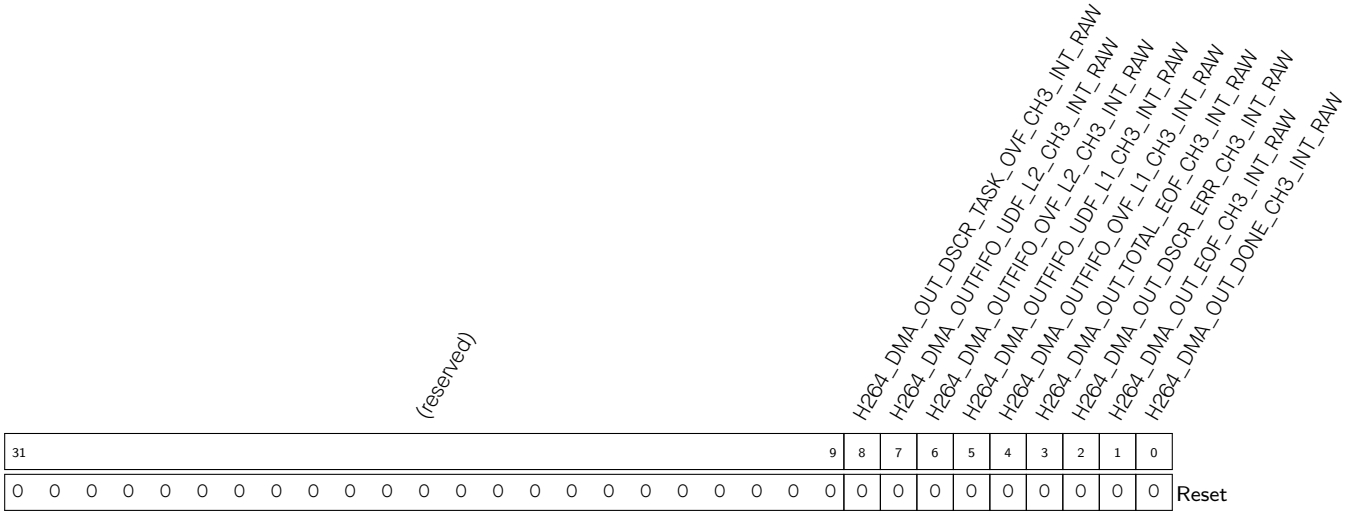
H264_DMA_OUTFIFO_UDF_L1_CH2_INT_CLR Write 1 to clear H264_DMA_OUTFIFO_UDF_L1_-CH2_INT. (WT)

H264_DMA_OUTFIFO_OVF_L2_CH2_INT_CLR Write 1 to clear H264_DMA_OUTFIFO_OVF_L2_-CH2_INT. (WT)

H264_DMA_OUTFIFO_UDF_L2_CH2_INT_CLR Write 1 to clear H264_DMA_OUTFIFO_UDF_L2_-CH2_INT. (WT)

H264_DMA_OUT_DSCR_TASK_OVF_CH2_INT_CLR Write 1 to clear H264_DMA_OUT_DSCR_-TASK_OVF_CH2_INT. (WT)

Register 41.185. H264_DMA_OUT_INT_RAW_CH3_REG (0x0304)



H264_DMA_OUT_DONE_CH3_INT_RAW The raw interrupt status of H264_DMA_OUT_DONE_CH3_INT. (R/WTC/SS)

H264_DMA_OUT_EOF_CH3_INT_RAW The raw interrupt status of H264_DMA_OUT_EOF_CH3_INT. (R/WTC/SS)

H264_DMA_OUT_DSCR_ERR_CH3_INT_RAW The raw interrupt status of H264_DMA_OUT_DSCR_ERR_CH3_INT. (R/WTC/SS)

H264_DMA_OUT_TOTAL_EOF_CH3_INT_RAW The raw interrupt status of H264_DMA_OUT_TOTAL_EOF_CH3_INT. (R/WTC/SS)

H264_DMA_OUTFIFO_OVF_L1_CH3_INT_RAW The raw interrupt status of H264_DMA_OUTFIFO_OVF_L1_CH3_INT. (R/WTC/SS)

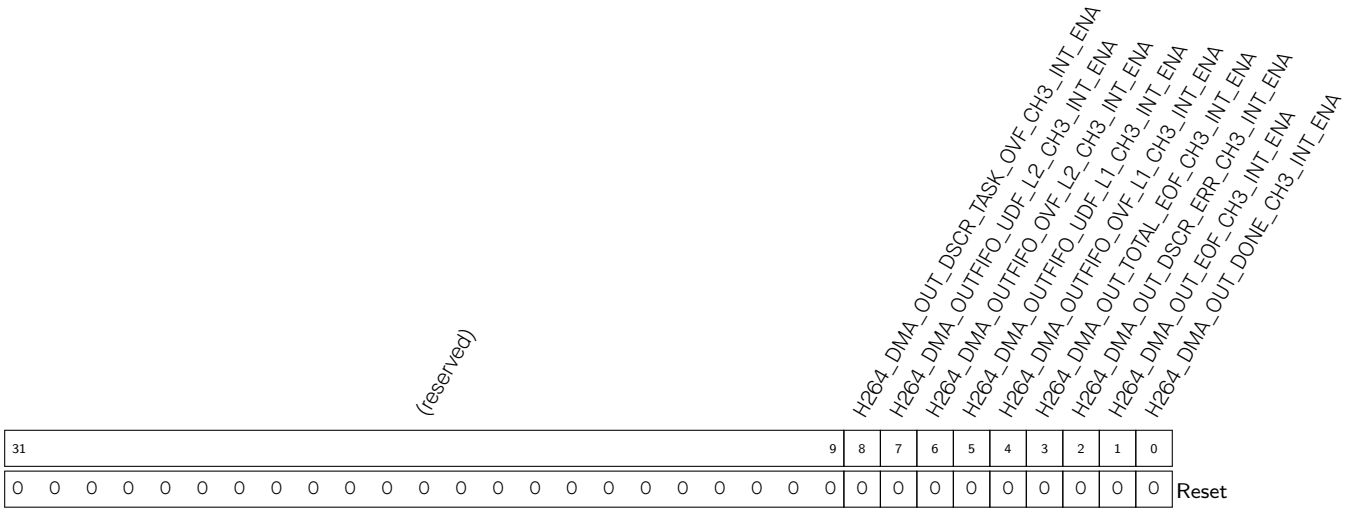
H264_DMA_OUTFIFO_UDF_L1_CH3_INT_RAW The raw interrupt status of H264_DMA_OUTFIFO_UDF_L1_CH3_INT. (R/WTC/SS)

H264_DMA_OUTFIFO_OVF_L2_CH3_INT_RAW The raw interrupt status of H264_DMA_OUTFIFO_OVF_L2_CH3_INT. (R/WTC/SS)

H264_DMA_OUTFIFO_UDF_L2_CH3_INT_RAW The raw interrupt status of H264_DMA_OUTFIFO_UDF_L2_CH3_INT. (R/WTC/SS)

H264_DMA_OUT_DSCR_TASK_OVF_CH3_INT_RAW The raw interrupt status of H264_DMA_OUT_DSCR_TASK_OVF_CH3_INT. (R/WTC/SS)

Register 41.186. H264_DMA_OUT_INT_ENA_CH3_REG (0x0308)



H264_DMA_OUT_DONE_CH3_INT_ENA Write 1 to enable H264_DMA_OUT_DONE_CH3_INT. (R/W)

H264_DMA_OUT_EOF_CH3_INT_ENA Write 1 to enable H264_DMA_OUT_EOF_CH3_INT. (R/W)

H264_DMA_OUT_DSCR_ERR_CH3_INT_ENA Write 1 to enable H264_DMA_OUT_DSCR_ERR_CH3_INT. (R/W)

H264_DMA_OUT_TOTAL_EOF_CH3_INT_ENA Write 1 to enable H264_DMA_OUT_TOTAL_EOF_CH3_INT. (R/W)

H264_DMA_OUTFIFO_OVF_L1_CH3_INT_ENA Write 1 to enable H264_DMA_OUTFIFO_OVF_L1_CH3_INT. (R/W)

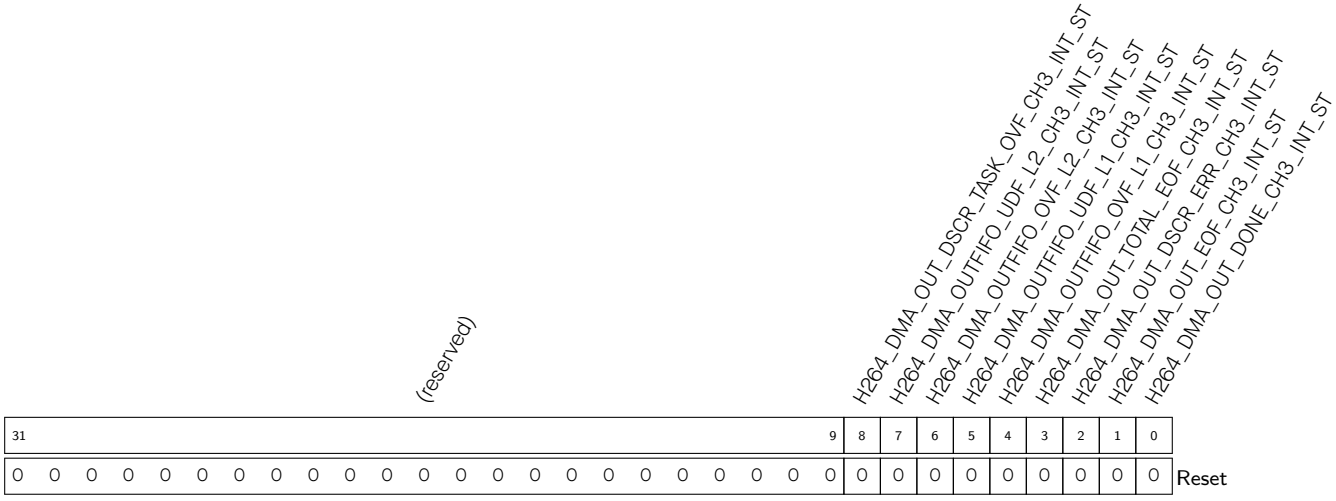
H264_DMA_OUTFIFO_UDF_L1_CH3_INT_ENA Write 1 to enable H264_DMA_OUTFIFO_UDF_L1_CH3_INT. (R/W)

H264_DMA_OUTFIFO_OVF_L2_CH3_INT_ENA Write 1 to enable H264_DMA_OUTFIFO_OVF_L2_CH3_INT. (R/W)

H264_DMA_OUTFIFO_UDF_L2_CH3_INT_ENA Write 1 to enable H264_DMA_OUTFIFO_UDF_L2_CH3_INT. (R/W)

H264_DMA_OUT_DSCR_TASK_OVF_CH3_INT_ENA Write 1 to enable H264_DMA_OUT_DSCR_TASK_OVF_CH3_INT. (R/W)

Register 41.187. H264_DMA_OUT_INT_ST_CH3_REG (0x030C)



H264_DMA_OUT_DONE_CH3_INT_ST The masked interrupt status of H264_DMA_OUT_DONE_CH3_INT. (RO)

H264_DMA_OUT_EOF_CH3_INT_ST The masked interrupt status of H264_DMA_OUT_EOF_CH3_INT. (RO)

H264_DMA_OUT_DSCR_ERR_CH3_INT_ST The masked interrupt status of H264_DMA_OUT_DSCR_ERR_CH3_INT. (RO)

H264_DMA_OUT_TOTAL_EOF_CH3_INT_ST The masked interrupt status of H264_DMA_OUT_TOTAL_EOF_CH3_INT. (RO)

H264_DMA_OUTFIFO_OVF_L1_CH3_INT_ST The masked interrupt status of H264_DMA_OUTFIFO_OVF_L1_CH3_INT. (RO)

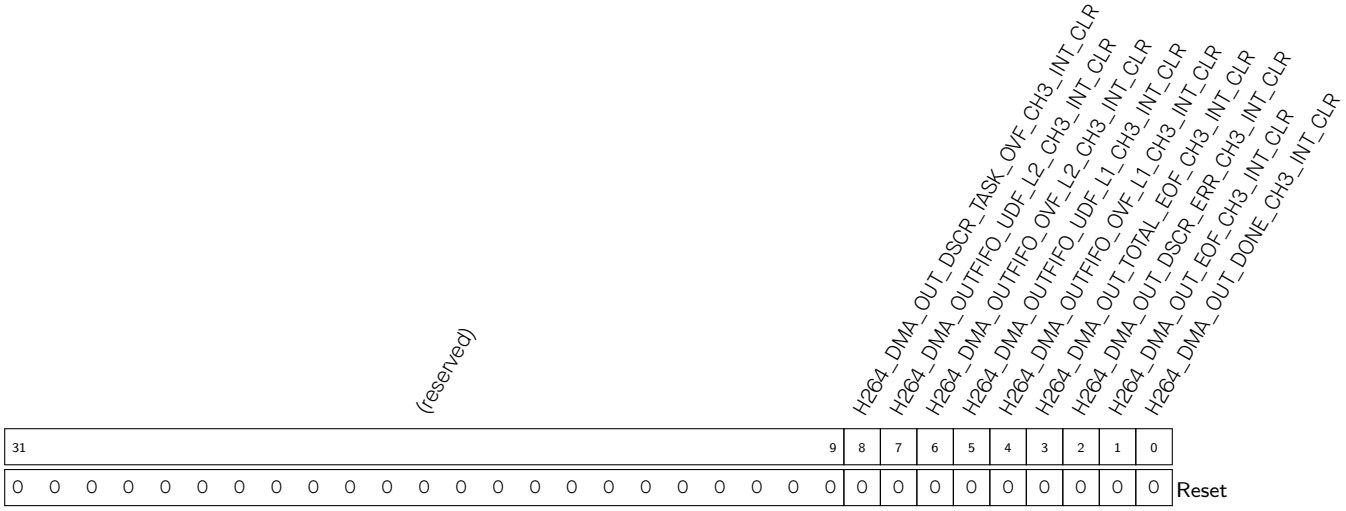
H264_DMA_OUTFIFO_UDF_L1_CH3_INT_ST The masked interrupt status of H264_DMA_OUTFIFO_UDF_L1_CH3_INT. (RO)

H264_DMA_OUTFIFO_OVF_L2_CH3_INT_ST The masked interrupt status of H264_DMA_OUTFIFO_OVF_L2_CH3_INT. (RO)

H264_DMA_OUTFIFO_UDF_L2_CH3_INT_ST The masked interrupt status of H264_DMA_OUTFIFO_UDF_L2_CH3_INT. (RO)

H264_DMA_OUT_DSCR_TASK_OVF_CH3_INT_ST The masked interrupt status of H264_DMA_OUT_DSCR_TASK_OVF_CH3_INT. (RO)

Register 41.188. H264_DMA_OUT_INT_CLR_CH3_REG (0x0310)



H264_DMA_OUT_DONE_CH3_INT_CLR Write 1 to clear H264_DMA_OUT_DONE_CH3_INT. (WT)

H264_DMA_OUT_EOF_CH3_INT_CLR Write 1 to clear H264_DMA_OUT_EOF_CH3_INT. (WT)

H264_DMA_OUT_DSCR_ERR_CH3_INT_CLR Write 1 to clear H264_DMA_OUT_DSCR_ERR_CH3_INT. (WT)

H264_DMA_OUT_TOTAL_EOF_CH3_INT_CLR Write 1 to clear H264_DMA_OUT_TOTAL_EOF_CH3_INT. (WT)

H264_DMA_OUTFIFO_OVF_L1_CH3_INT_CLR Write 1 to clear H264_DMA_OUTFIFO_OVF_L1_CH3_INT. (WT)

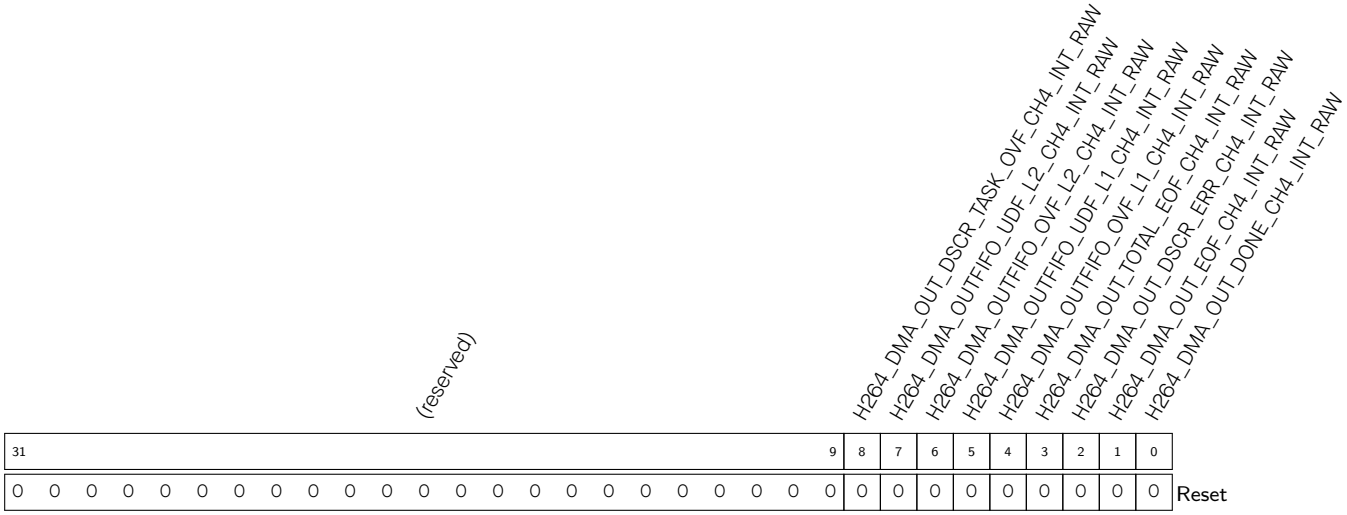
H264_DMA_OUTFIFO_UDF_L1_CH3_INT_CLR Write 1 to clear H264_DMA_OUTFIFO_UDF_L1_CH3_INT. (WT)

H264_DMA_OUTFIFO_OVF_L2_CH3_INT_CLR Write 1 to clear H264_DMA_OUTFIFO_OVF_L2_CH3_INT. (WT)

H264_DMA_OUTFIFO_UDF_L2_CH3_INT_CLR Write 1 to clear H264_DMA_OUTFIFO_UDF_L2_CH3_INT. (WT)

H264_DMA_OUT_DSCR_TASK_OVF_CH3_INT_CLR Write 1 to clear H264_DMA_OUT_DSCR_TASK_OVF_CH3_INT. (WT)

Register 41.189. H264_DMA_OUT_INT_RAW_CH4_REG (0x0404)



H264_DMA_OUT_DONE_CH4_INT_RAW The raw interrupt status of H264_DMA_OUT_DONE_CH4_INT. (R/WTC/SS)

H264_DMA_OUT_EOF_CH4_INT_RAW The raw interrupt status of H264_DMA_OUT_EOF_CH4_INT. (R/WTC/SS)

H264_DMA_OUT_DSCR_ERR_CH4_INT_RAW The raw interrupt status of H264_DMA_OUT_DSCR_ERR_CH4_INT. (R/WTC/SS)

H264_DMA_OUT_TOTAL_EOF_CH4_INT_RAW The raw interrupt status of H264_DMA_OUT_TOTAL_EOF_CH4_INT. (R/WTC/SS)

H264_DMA_OUTFIFO_OVF_L1_CH4_INT_RAW The raw interrupt status of H264_DMA_OUTFIFO_OVF_L1_CH4_INT. (R/WTC/SS)

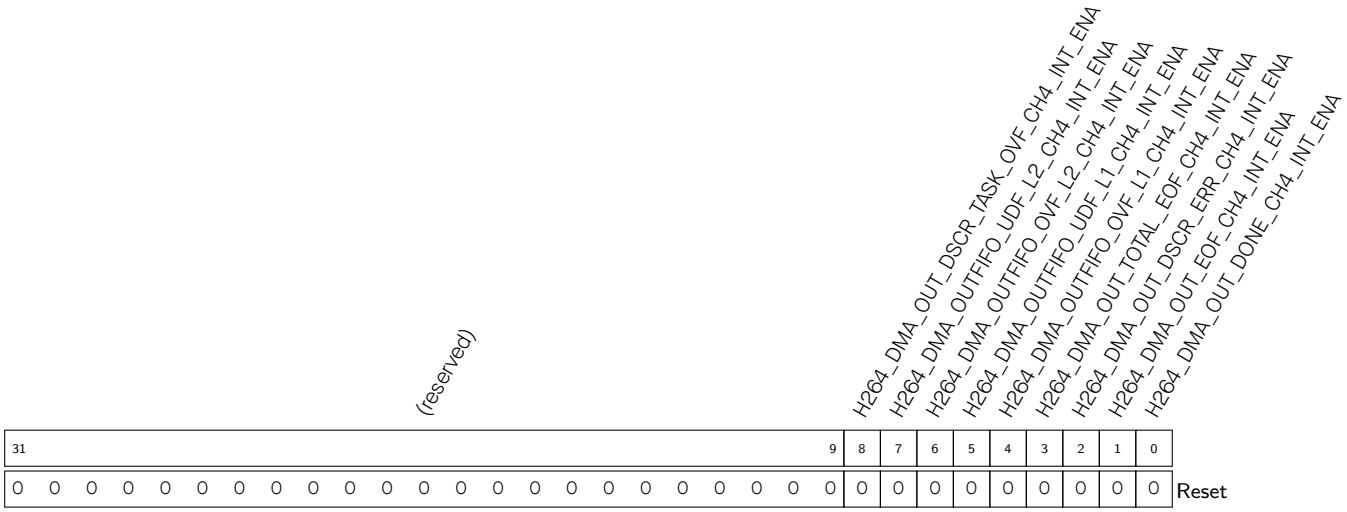
H264_DMA_OUTFIFO_UDF_L1_CH4_INT_RAW The raw interrupt status of H264_DMA_OUTFIFO_UDF_L1_CH4_INT. (R/WTC/SS)

H264_DMA_OUTFIFO_OVF_L2_CH4_INT_RAW The raw interrupt status of H264_DMA_OUTFIFO_OVF_L2_CH4_INT. (R/WTC/SS)

H264_DMA_OUTFIFO_UDF_L2_CH4_INT_RAW The raw interrupt status of H264_DMA_OUTFIFO_UDF_L2_CH4_INT. (R/WTC/SS)

H264_DMA_OUT_DSCR_TASK_OVF_CH4_INT_RAW The raw interrupt status of H264_DMA_OUT_DSCR_TASK_OVF_CH4_INT. (R/WTC/SS)

Register 41.190. H264_DMA_OUT_INT_ENA_CH4_REG (0x0408)



H264_DMA_OUT_DONE_CH4_INT_ENA Write 1 to enable H264_DMA_OUT_DONE_CH4_INT. (R/W)

H264_DMA_OUT_EOF_CH4_INT_ENA Write 1 to enable H264_DMA_OUT_EOF_CH4_INT. (R/W)

H264_DMA_OUT_DSCR_ERR_CH4_INT_ENA Write 1 to enable H264_DMA_OUT_DSCR_ERR_CH4_INT. (R/W)

H264_DMA_OUT_TOTAL_EOF_CH4_INT_ENA Write 1 to enable H264_DMA_OUT_TOTAL_EOF_CH4_INT. (R/W)

H264_DMA_OUTFIFO_OVF_L1_CH4_INT_ENA Write 1 to enable H264_DMA_OUTFIFO_OVF_L1_CH4_INT. (R/W)

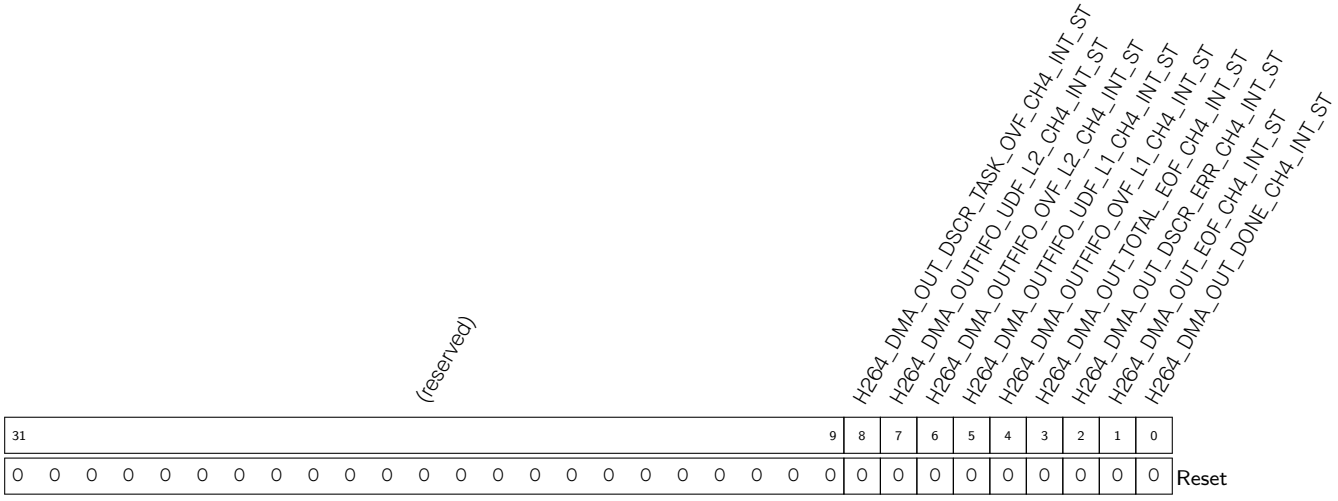
H264_DMA_OUTFIFO_UDF_L1_CH4_INT_ENA Write 1 to enable H264_DMA_OUTFIFO_UDF_L1_CH4_INT. (R/W)

H264_DMA_OUTFIFO_OVF_L2_CH4_INT_ENA Write 1 to enable H264_DMA_OUTFIFO_OVF_L2_CH4_INT. (R/W)

H264_DMA_OUTFIFO_UDF_L2_CH4_INT_ENA Write 1 to enable H264_DMA_OUTFIFO_UDF_L2_CH4_INT. (R/W)

H264_DMA_OUT_DSCR_TASK_OVF_CH4_INT_ENA Write 1 to enable H264_DMA_OUT_DSCR_TASK_OVF_CH4_INT. (R/W)

Register 41.191. H264_DMA_OUT_INT_ST_CH4_REG (0x040C)



H264_DMA_OUT_DONE_CH4_INT_ST The masked interrupt status of H264_DMA_OUT_DONE_CH4_INT. (RO)

H264_DMA_OUT_EOF_CH4_INT_ST The masked interrupt status of H264_DMA_OUT_EOF_CH4_INT. (RO)

H264_DMA_OUT_DSCR_ERR_CH4_INT_ST The masked interrupt status of H264_DMA_OUT_DSCR_ERR_CH4_INT. (RO)

H264_DMA_OUT_TOTAL_EOF_CH4_INT_ST The masked interrupt status of H264_DMA_OUT_TOTAL_EOF_CH4_INT. (RO)

H264_DMA_OUTFIFO_OVF_L1_CH4_INT_ST The masked interrupt status of H264_DMA_OUTFIFO_OVF_L1_CH4_INT. (RO)

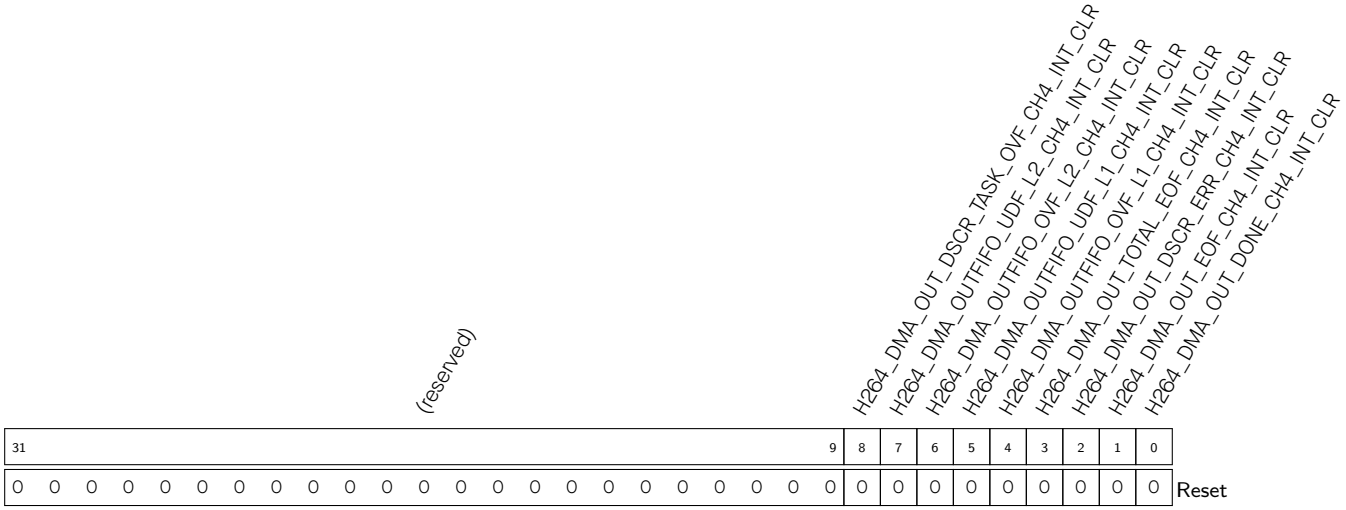
H264_DMA_OUTFIFO_UDF_L1_CH4_INT_ST The masked interrupt status of H264_DMA_OUTFIFO_UDF_L1_CH4_INT. (RO)

H264_DMA_OUTFIFO_OVF_L2_CH4_INT_ST The masked interrupt status of H264_DMA_OUTFIFO_OVF_L2_CH4_INT. (RO)

H264_DMA_OUTFIFO_UDF_L2_CH4_INT_ST The masked interrupt status of H264_DMA_OUTFIFO_UDF_L2_CH4_INT. (RO)

H264_DMA_OUT_DSCR_TASK_OVF_CH4_INT_ST The masked interrupt status of H264_DMA_OUT_DSCR_TASK_OVF_CH4_INT. (RO)

Register 41.192. H264_DMA_OUT_INT_CLR_CH4_REG (0x0410)



H264_DMA_OUT_DONE_CH4_INT_CLR Write 1 to clear H264_DMA_OUT_DONE_CH4_INT. (WT)

H264_DMA_OUT_EOF_CH4_INT_CLR Write 1 to clear H264_DMA_OUT_EOF_CH4_INT. (WT)

H264_DMA_OUT_DSCR_ERR_CH4_INT_CLR Write 1 to clear H264_DMA_OUT_DSCR_ERR_CH4_-INT. (WT)

H264_DMA_OUT_TOTAL_EOF_CH4_INT_CLR Write 1 to clear H264_DMA_OUT_TOTAL_EOF_-CH4_INT. (WT)

H264_DMA_OUTFIFO_OVF_L1_CH4_INT_CLR Write 1 to clear H264_DMA_OUTFIFO_OVF_L1_-CH4_INT. (WT)

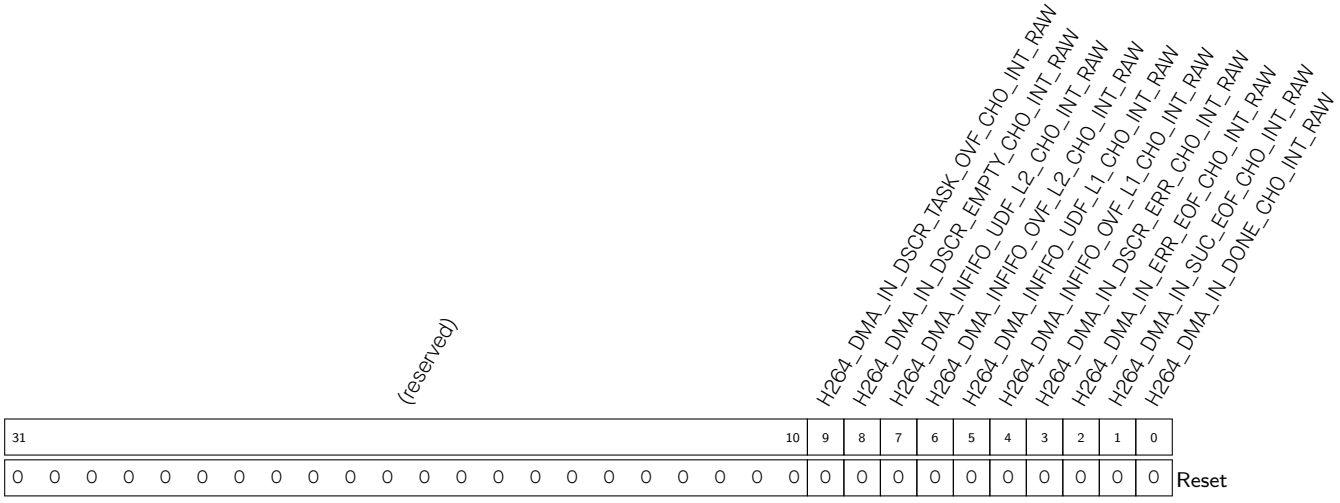
H264_DMA_OUTFIFO_UDF_L1_CH4_INT_CLR Write 1 to clear H264_DMA_OUTFIFO_UDF_L1_-CH4_INT. (WT)

H264_DMA_OUTFIFO_OVF_L2_CH4_INT_CLR Write 1 to clear H264_DMA_OUTFIFO_OVF_L2_-CH4_INT. (WT)

H264_DMA_OUTFIFO_UDF_L2_CH4_INT_CLR Write 1 to clear H264_DMA_OUTFIFO_UDF_L2_-CH4_INT. (WT)

H264_DMA_OUT_DSCR_TASK_OVF_CH4_INT_CLR Write 1 to clear H264_DMA_OUT_DSCR_-TASK_OVF_CH4_INT. (WT)

Register 41.193. H264_DMA_IN_INT_RAW_CH0_REG (0x0504)



H264_DMA_IN_DONE_CH0_INT_RAW The raw interrupt status of H264_DMA_IN_DONE_CH0_INT. (R/WTC/SS)

H264_DMA_IN_SUC_EOF_CH0_INT_RAW The raw interrupt status of H264_DMA_IN_SUC_EOF_CH0_INT. (R/WTC/SS)

H264_DMA_IN_ERR_EOF_CH0_INT_RAW The raw interrupt status of H264_DMA_IN_ERR_EOF_CH0_INT. (R/WTC/SS)

H264_DMA_IN_DSCR_ERR_CH0_INT_RAW The raw interrupt status of H264_DMA_IN_DSCR_ERR_CH0_INT. (R/WTC/SS)

H264_DMA_INFIFO_OVF_L1_CH0_INT_RAW The raw interrupt status of H264_DMA_INFIFO_OVF_L1_CH0_INT. (R/WTC/SS)

H264_DMA_INFIFO_UDF_L1_CH0_INT_RAW The raw interrupt status of H264_DMA_INFIFO_UDF_L1_CH0_INT. (R/WTC/SS)

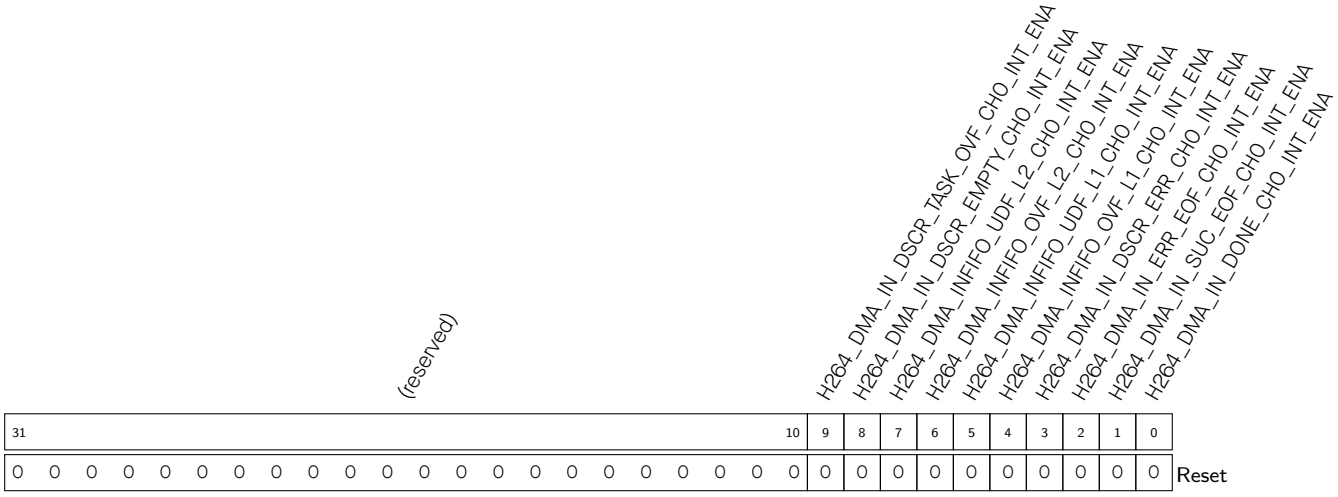
H264_DMA_INFIFO_OVF_L2_CH0_INT_RAW The raw interrupt status of H264_DMA_INFIFO_OVF_L2_CH0_INT. (R/WTC/SS)

H264_DMA_INFIFO_UDF_L2_CH0_INT_RAW The raw interrupt status of H264_DMA_INFIFO_UDF_L2_CH0_INT. (R/WTC/SS)

H264_DMA_IN_DSCR_EMPTY_CH0_INT_RAW The raw interrupt status of H264_DMA_IN_DSCR_EMPTY_CH0_INT. (R/WTC/SS)

H264_DMA_IN_DSCR_TASK_OVF_CH0_INT_RAW The raw interrupt status of H264_DMA_IN_DSCR_TASK_OVF_CH0_INT. (R/WTC/SS)

Register 41.194. H264_DMA_IN_INT_ENA_CH0_REG (0x0508)



H264_DMA_IN_DONE_CH0_INT_ENA Write 1 to enable H264_DMA_IN_DONE_CH0_INT. (R/W)

H264_DMA_IN_SUC_EOF_CH0_INT_ENA Write 1 to enable H264_DMA_IN_SUC_EOF_CH0_INT. (R/W)

H264_DMA_IN_ERR_EOF_CH0_INT_ENA Write 1 to enable H264_DMA_IN_ERR_EOF_CH0_INT. (R/W)

H264_DMA_IN_DSCR_ERR_CH0_INT_ENA Write 1 to enable H264_DMA_IN_DSCR_ERR_CH0_-INT. (R/W)

H264_DMA_INFIFO_OVF_L1_CH0_INT_ENA Write 1 to enable H264_DMA_INFIFO_OVF_L1_CH0_-INT. (R/W)

H264_DMA_INFIFO_UDF_L1_CH0_INT_ENA Write 1 to enable H264_DMA_INFIFO_UDF_L1_CH0_-INT. (R/W)

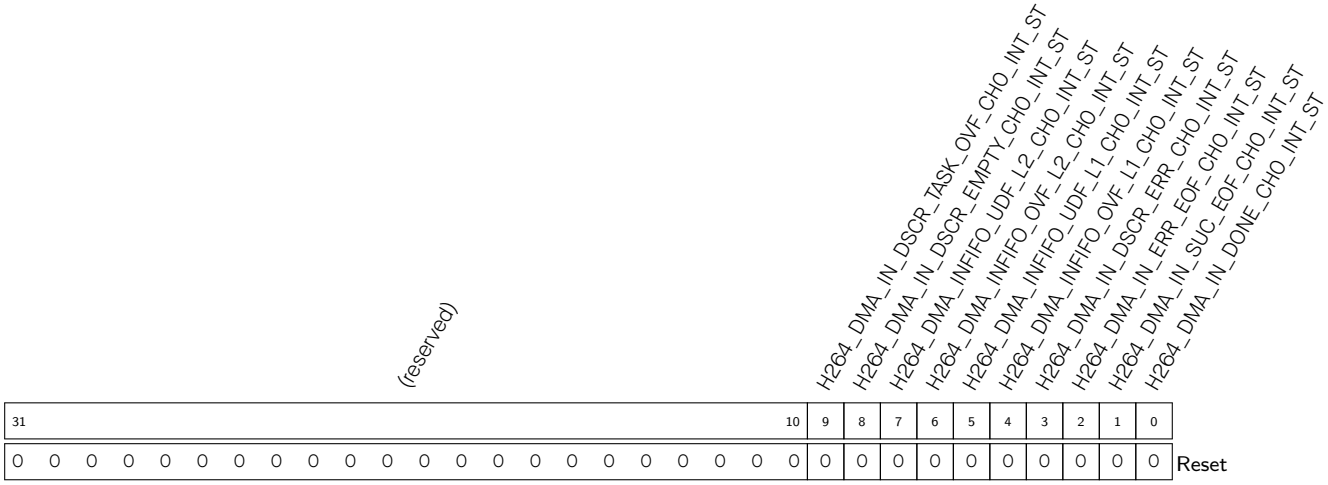
H264_DMA_INFIFO_OVF_L2_CH0_INT_ENA Write 1 to enable H264_DMA_INFIFO_OVF_L2_-CH0_INT. (R/W)

H264_DMA_INFIFO_UDF_L2_CH0_INT_ENA Write 1 to enable H264_DMA_INFIFO_UDF_L2_-CH0_INT. (R/W)

H264_DMA_IN_DSCR_EMPTY_CH0_INT_ENA Write 1 to enable H264_DMA_IN_DSCR_EMPTY_-CH0_INT. (R/W)

H264_DMA_IN_DSCR_TASK_OVF_CH0_INT_ENA Write 1 to enable H264_DMA_IN_DSCR_TASK_-OVF_CH0_INT. (R/W)

Register 41.195. H264_DMA_IN_INT_ST_CH0_REG (0x050C)



H264_DMA_IN_DONE_CH0_INT_ST The masked interrupt status of H264_DMA_IN_DONE_CH0_INT. (RO)

H264_DMA_IN_SUC_EOF_CH0_INT_ST The masked interrupt status of H264_DMA_IN_SUC_EOF_CH0_INT. (RO)

H264_DMA_IN_ERR_EOF_CH0_INT_ST The masked interrupt status of H264_DMA_IN_ERR_EOF_CH0_INT. (RO)

H264_DMA_IN_DSCR_ERR_CH0_INT_ST The masked interrupt status of H264_DMA_IN_DSCR_ERR_CH0_INT. (RO)

H264_DMA_INFIFO_OVF_L1_CH0_INT_ST The masked interrupt status of H264_DMA_INFIFO_OVF_L1_CH0_INT. (RO)

H264_DMA_INFIFO_UDF_L1_CH0_INT_ST The masked interrupt status of H264_DMA_INFIFO_UDF_L1_CH0_INT. (RO)

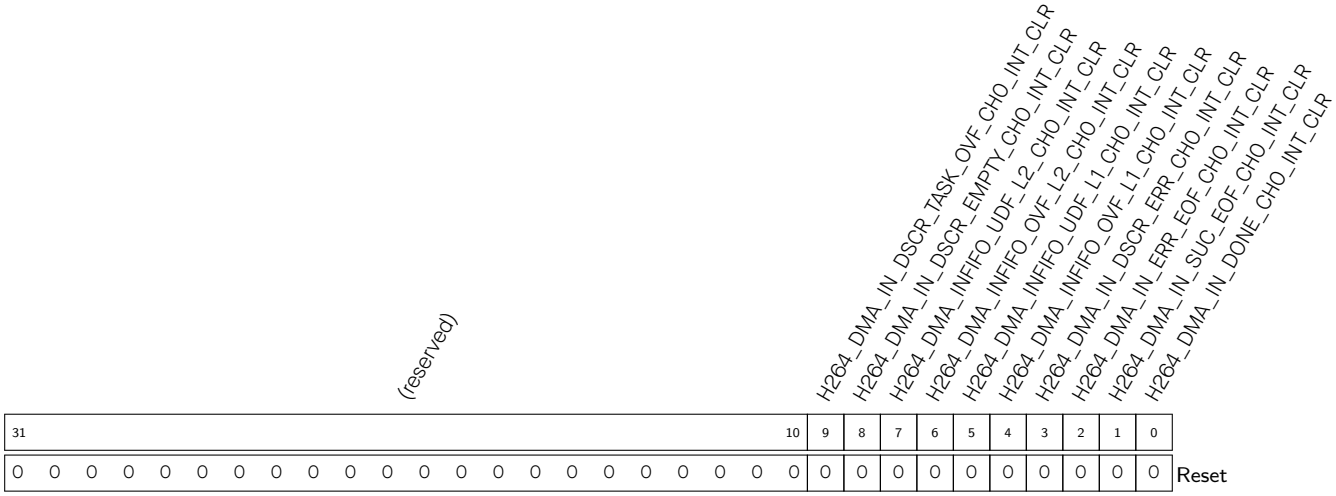
H264_DMA_INFIFO_OVF_L2_CH0_INT_ST The masked interrupt status of H264_DMA_INFIFO_OVF_L2_CH0_INT. (RO)

H264_DMA_INFIFO_UDF_L2_CH0_INT_ST The masked interrupt status of H264_DMA_INFIFO_UDF_L2_CH0_INT. (RO)

H264_DMA_IN_DSCR_EMPTY_CH0_INT_ST The masked interrupt status of H264_DMA_IN_DSCR_EMPTY_CH0_INT. (RO)

H264_DMA_IN_DSCR_TASK_OVF_CH0_INT_ST The masked interrupt status of H264_DMA_IN_DSCR_TASK_OVF_CH0_INT. (RO)

Register 41.196. H264_DMA_IN_INT_CLR_CH0_REG (0x0510)



H264_DMA_IN_DONE_CH0_INT_CLR Write 1 to clear H264_DMA_IN_DONE_CH0_INT. (WT)

H264_DMA_IN_SUC_EOF_CH0_INT_CLR Write 1 to clear H264_DMA_IN_SUC_EOF_CH0_INT. (WT)

H264_DMA_IN_ERR_EOF_CH0_INT_CLR Write 1 to clear H264_DMA_IN_ERR_EOF_CH0_INT. (WT)

H264_DMA_IN_DSCR_ERR_CH0_INT_CLR Write 1 to clear H264_DMA_IN_DSCR_ERR_CH0_INT. (WT)

H264_DMA_INFIFO_OVF_L1_CH0_INT_CLR Write 1 to clear H264_DMA_INFIFO_OVF_L1_CH0_INT. (WT)

H264_DMA_INFIFO_UDF_L1_CH0_INT_CLR Write 1 to clear H264_DMA_INFIFO_UDF_L1_CH0_INT. (WT)

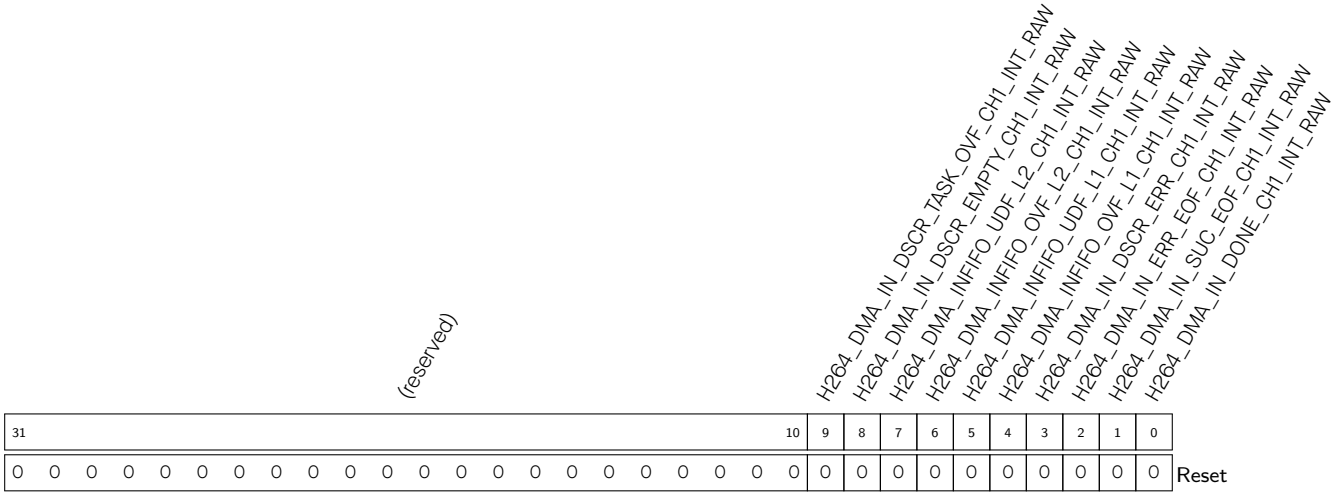
H264_DMA_INFIFO_OVF_L2_CH0_INT_CLR Write 1 to clear H264_DMA_INFIFO_OVF_L2_CH0_INT. (WT)

H264_DMA_INFIFO_UDF_L2_CH0_INT_CLR Write 1 to clear H264_DMA_INFIFO_UDF_L2_CH0_INT. (WT)

H264_DMA_IN_DSCR_EMPTY_CH0_INT_CLR Write 1 to clear H264_DMA_IN_DSCR_EMPTY_CH0_INT. (WT)

H264_DMA_IN_DSCR_TASK_OVF_CH0_INT_CLR Write 1 to clear H264_DMA_IN_DSCR_TASK_OVF_CH0_INT. (WT)

Register 41.197. H264_DMA_IN_INT_RAW_CH1_REG (0x0604)



H264_DMA_IN_DONE_CH1_INT_RAW The raw interrupt status of H264_DMA_IN_DONE_CH1_INT. (R/WTC/SS)

H264_DMA_IN_SUC_EOF_CH1_INT_RAW The raw interrupt status of H264_DMA_IN_SUC_EOF_CH1_INT. (R/WTC/SS)

H264_DMA_IN_ERR_EOF_CH1_INT_RAW The raw interrupt status of H264_DMA_IN_ERR_EOF_CH1_INT. (R/WTC/SS)

H264_DMA_IN_DSCR_ERR_CH1_INT_RAW The raw interrupt status of H264_DMA_IN_DSCR_ERR_CH1_INT. (R/WTC/SS)

H264_DMA_INFIFO_OVF_L1_CH1_INT_RAW The raw interrupt status of H264_DMA_INFIFO_OVF_L1_CH1_INT. (R/WTC/SS)

H264_DMA_INFIFO_UDF_L1_CH1_INT_RAW The raw interrupt status of H264_DMA_INFIFO_UDF_L1_CH1_INT. (R/WTC/SS)

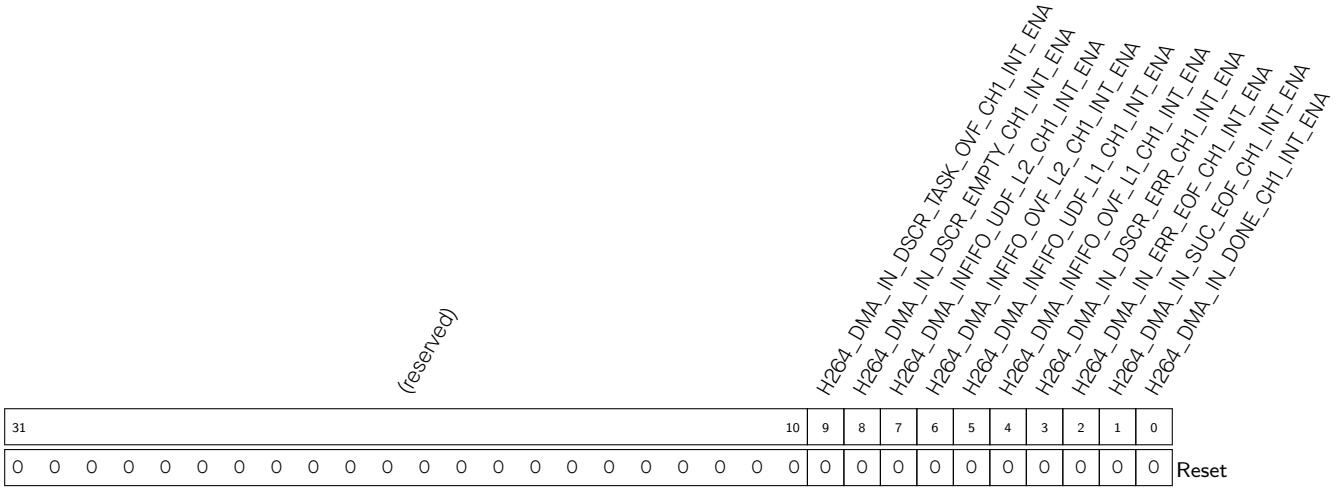
H264_DMA_INFIFO_OVF_L2_CH1_INT_RAW The raw interrupt status of H264_DMA_INFIFO_OVF_L2_CH1_INT. (R/WTC/SS)

H264_DMA_INFIFO_UDF_L2_CH1_INT_RAW The raw interrupt status of H264_DMA_INFIFO_UDF_L2_CH1_INT. (R/WTC/SS)

H264_DMA_IN_DSCR_EMPTY_CH1_INT_RAW The raw interrupt status of H264_DMA_IN_DSCR_EMPTY_CH1_INT. (R/WTC/SS)

H264_DMA_IN_DSCR_TASK_OVF_CH1_INT_RAW The raw interrupt status of H264_DMA_IN_DSCR_TASK_OVF_CH1_INT. (R/WTC/SS)

Register 41.198. H264_DMA_IN_INT_ENA_CH1_REG (0x0608)



H264_DMA_IN_DONE_CH1_INT_ENA Write 1 to enable H264_DMA_IN_DONE_CH1_INT. (R/W)

H264_DMA_IN_SUC_EOF_CH1_INT_ENA Write 1 to enable H264_DMA_IN_SUC_EOF_CH1_INT. (R/W)

H264_DMA_IN_ERR_EOF_CH1_INT_ENA Write 1 to enable H264_DMA_IN_ERR_EOF_CH1_INT. (R/W)

H264_DMA_IN_DSCR_ERR_CH1_INT_ENA Write 1 to enable H264_DMA_IN_DSCR_ERR_CH1_INT. (R/W)

H264_DMA_INFIFO_OVF_L1_CH1_INT_ENA Write 1 to enable H264_DMA_INFIFO_OVF_L1_CH1_INT. (R/W)

H264_DMA_INFIFO_UDF_L1_CH1_INT_ENA Write 1 to enable H264_DMA_INFIFO_UDF_L1_CH1_INT. (R/W)

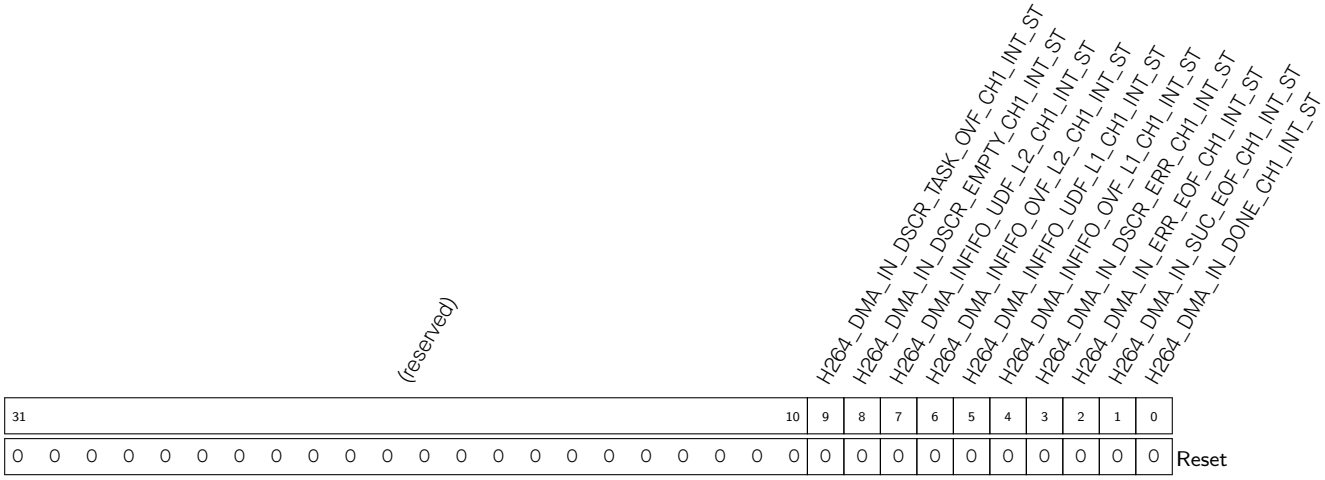
H264_DMA_INFIFO_OVF_L2_CH1_INT_ENA Write 1 to enable H264_DMA_INFIFO_OVF_L2_CH1_INT. (R/W)

H264_DMA_INFIFO_UDF_L2_CH1_INT_ENA Write 1 to enable H264_DMA_INFIFO_UDF_L2_CH1_INT. (R/W)

H264_DMA_IN_DSCR_EMPTY_CH1_INT_ENA Write 1 to enable H264_DMA_IN_DSCR_EMPTY_CH1_INT. (R/W)

H264_DMA_IN_DSCR_TASK_OVF_CH1_INT_ENA Write 1 to enable H264_DMA_IN_DSCR_TASK_OVF_CH1_INT. (R/W)

Register 41.199. H264_DMA_IN_INT_ST_CH1_REG (0x060C)



H264_DMA_IN_DONE_CH1_INT_ST The masked interrupt status of H264_DMA_IN_DONE_CH1_INT. (RO)

H264_DMA_IN_SUC_EOF_CH1_INT_ST The masked interrupt status of H264_DMA_IN_SUC_EOF_CH1_INT. (RO)

H264_DMA_IN_ERR_EOF_CH1_INT_ST The masked interrupt status of H264_DMA_IN_ERR_EOF_CH1_INT. (RO)

H264_DMA_IN_DSCR_ERR_CH1_INT_ST The masked interrupt status of H264_DMA_IN_DSCR_ERR_CH1_INT. (RO)

H264_DMA_INFIFO_OVF_L1_CH1_INT_ST The masked interrupt status of H264_DMA_INFIFO_OVF_L1_CH1_INT. (RO)

H264_DMA_INFIFO_UDF_L1_CH1_INT_ST The masked interrupt status of H264_DMA_INFIFO_UDF_L1_CH1_INT. (RO)

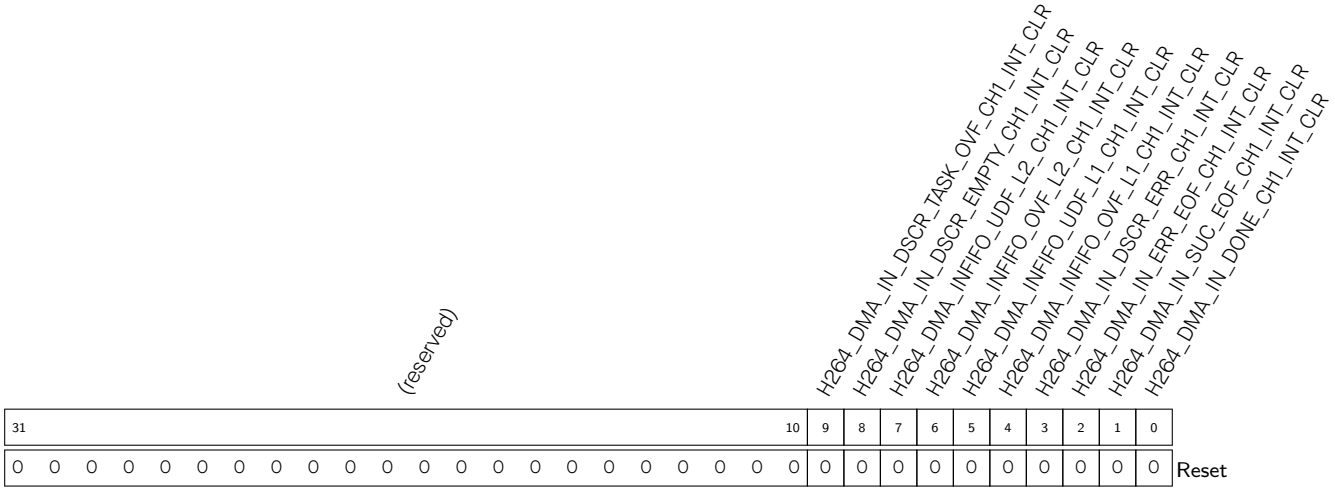
H264_DMA_INFIFO_OVF_L2_CH1_INT_ST The masked interrupt status of H264_DMA_INFIFO_OVF_L2_CH1_INT. (RO)

H264_DMA_INFIFO_UDF_L2_CH1_INT_ST The masked interrupt status of H264_DMA_INFIFO_UDF_L2_CH1_INT. (RO)

H264_DMA_IN_DSCR_EMPTY_CH1_INT_ST The masked interrupt status of H264_DMA_IN_DSCR_EMPTY_CH1_INT. (RO)

H264_DMA_IN_DSCR_TASK_OVF_CH1_INT_ST The masked interrupt status of H264_DMA_IN_DSCR_TASK_OVF_CH1_INT. (RO)

Register 41.200. H264_DMA_IN_INT_CLR_CH1_REG (0x0610)



H264_DMA_IN_DONE_CH1_INT_CLR Write 1 to clear H264_DMA_IN_DONE_CH1_INT. (WT)

H264_DMA_IN_SUC_EOF_CH1_INT_CLR Write 1 to clear H264_DMA_IN_SUC_EOF_CH1_INT. (WT)

H264_DMA_IN_ERR_EOF_CH1_INT_CLR Write 1 to clear H264_DMA_IN_ERR_EOF_CH1_INT. (WT)

H264_DMA_IN_DSCR_ERR_CH1_INT_CLR Write 1 to clear H264_DMA_IN_DSCR_ERR_CH1_INT. (WT)

H264_DMA_INFIFO_OVF_L1_CH1_INT_CLR Write 1 to clear H264_DMA_INFIFO_OVF_L1_CH1_INT. (WT)

H264_DMA_INFIFO_UDF_L1_CH1_INT_CLR Write 1 to clear H264_DMA_INFIFO_UDF_L1_CH1_INT. (WT)

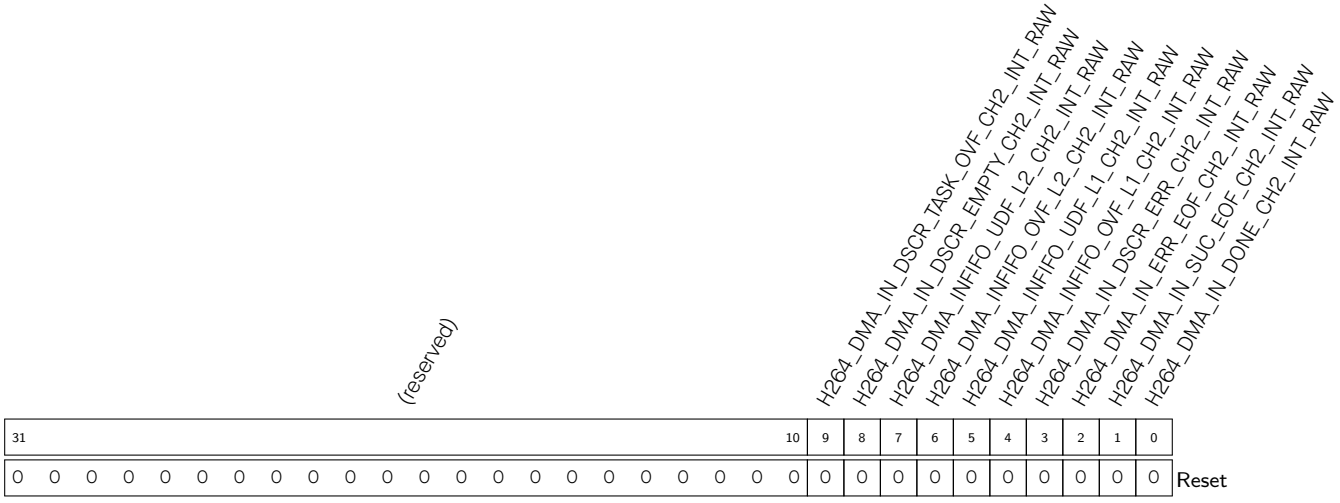
H264_DMA_INFIFO_OVF_L2_CH1_INT_CLR Write 1 to clear H264_DMA_INFIFO_OVF_L2_CH1_INT. (WT)

H264_DMA_INFIFO_UDF_L2_CH1_INT_CLR Write 1 to clear H264_DMA_INFIFO_UDF_L2_CH1_INT. (WT)

H264_DMA_IN_DSCR_EMPTY_CH1_INT_CLR Write 1 to clear H264_DMA_IN_DSCR_EMPTY_CH1_INT. (WT)

H264_DMA_IN_DSCR_TASK_OVF_CH1_INT_CLR Write 1 to clear H264_DMA_IN_DSCR_TASK_OVF_CH1_INT. (WT)

Register 41.201. H264_DMA_IN_INT_RAW_CH2_REG (0x0704)



H264_DMA_IN_DONE_CH2_INT_RAW The raw interrupt status of H264_DMA_IN_DONE_CH2_INT. (R/WTC/SS)

H264_DMA_IN_SUC_EOF_CH2_INT_RAW The raw interrupt status of H264_DMA_IN_SUC_EOF_CH2_INT. (R/WTC/SS)

H264_DMA_IN_ERR_EOF_CH2_INT_RAW The raw interrupt status of H264_DMA_IN_ERR_EOF_CH2_INT. (R/WTC/SS)

H264_DMA_IN_DSCR_ERR_CH2_INT_RAW The raw interrupt status of H264_DMA_IN_DSCR_ERR_CH2_INT. (R/WTC/SS)

H264_DMA_INFIFO_OVF_L1_CH2_INT_RAW The raw interrupt status of H264_DMA_INFIFO_OVF_L1_CH2_INT. (R/WTC/SS)

H264_DMA_INFIFO_UDF_L1_CH2_INT_RAW The raw interrupt status of H264_DMA_INFIFO_UDF_L1_CH2_INT. (R/WTC/SS)

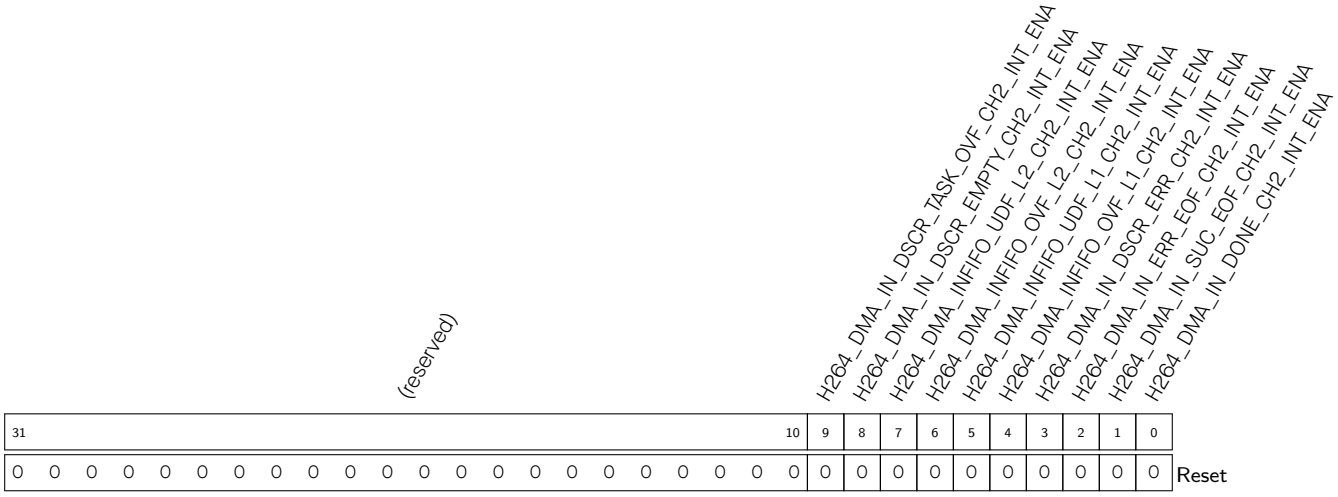
H264_DMA_INFIFO_OVF_L2_CH2_INT_RAW The raw interrupt status of H264_DMA_INFIFO_OVF_L2_CH2_INT. (R/WTC/SS)

H264_DMA_INFIFO_UDF_L2_CH2_INT_RAW The raw interrupt status of H264_DMA_INFIFO_UDF_L2_CH2_INT. (R/WTC/SS)

H264_DMA_IN_DSCR_EMPTY_CH2_INT_RAW The raw interrupt status of H264_DMA_IN_DSCR_EMPTY_CH2_INT. (R/WTC/SS)

H264_DMA_IN_DSCR_TASK_OVF_CH2_INT_RAW The raw interrupt status of H264_DMA_IN_DSCR_TASK_OVF_CH2_INT. (R/WTC/SS)

Register 41.202. H264_DMA_IN_INT_ENA_CH2_REG (0x0708)



H264_DMA_IN_DONE_CH2_INT_ENA Write 1 to enable H264_DMA_IN_DONE_CH2_INT. (R/W)

H264_DMA_IN_SUC_EOF_CH2_INT_ENA Write 1 to enable H264_DMA_IN_SUC_EOF_CH2_INT. (R/W)

H264_DMA_IN_ERR_EOF_CH2_INT_ENA Write 1 to enable H264_DMA_IN_ERR_EOF_CH2_INT. (R/W)

H264_DMA_IN_DSCR_ERR_CH2_INT_ENA Write 1 to enable H264_DMA_IN_DSCR_ERR_CH2_-INT. (R/W)

H264_DMA_INFIFO_OVF_L1_CH2_INT_ENA Write 1 to enable H264_DMA_INFIFO_OVF_L1_CH2_-INT. (R/W)

H264_DMA_INFIFO_UDF_L1_CH2_INT_ENA Write 1 to enable H264_DMA_INFIFO_UDF_L1_CH2_-INT. (R/W)

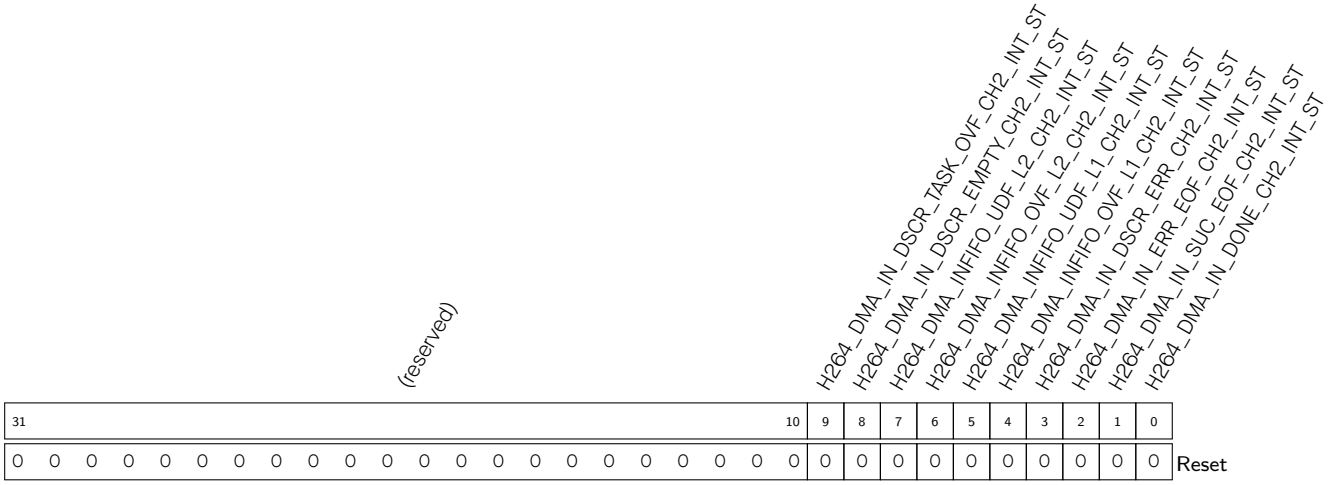
H264_DMA_INFIFO_OVF_L2_CH2_INT_ENA Write 1 to enable H264_DMA_INFIFO_OVF_L2_-CH2_INT. (R/W)

H264_DMA_INFIFO_UDF_L2_CH2_INT_ENA Write 1 to enable H264_DMA_INFIFO_UDF_L2_-CH2_INT. (R/W)

H264_DMA_IN_DSCR_EMPTY_CH2_INT_ENA Write 1 to enable H264_DMA_IN_DSCR_EMPTY_-CH2_INT. (R/W)

H264_DMA_IN_DSCR_TASK_OVF_CH2_INT_ENA Write 1 to enable H264_DMA_IN_DSCR_TASK_-OVF_CH2_INT. (R/W)

Register 41.203. H264_DMA_IN_INT_ST_CH2_REG (0x070C)



H264_DMA_IN_DONE_CH2_INT_ST The masked interrupt status of H264_DMA_IN_DONE_CH2_INT. (RO)

H264_DMA_IN_SUC_EOF_CH2_INT_ST The masked interrupt status of H264_DMA_IN_SUC_EOF_CH2_INT. (RO)

H264_DMA_IN_ERR_EOF_CH2_INT_ST The masked interrupt status of H264_DMA_IN_ERR_EOF_CH2_INT. (RO)

H264_DMA_IN_DSCR_ERR_CH2_INT_ST The masked interrupt status of H264_DMA_IN_DSCR_ERR_CH2_INT. (RO)

H264_DMA_INFIFO_OVF_L1_CH2_INT_ST The masked interrupt status of H264_DMA_INFIFO_OVF_L1_CH2_INT. (RO)

H264_DMA_INFIFO_UDF_L1_CH2_INT_ST The masked interrupt status of H264_DMA_INFIFO_UDF_L1_CH2_INT. (RO)

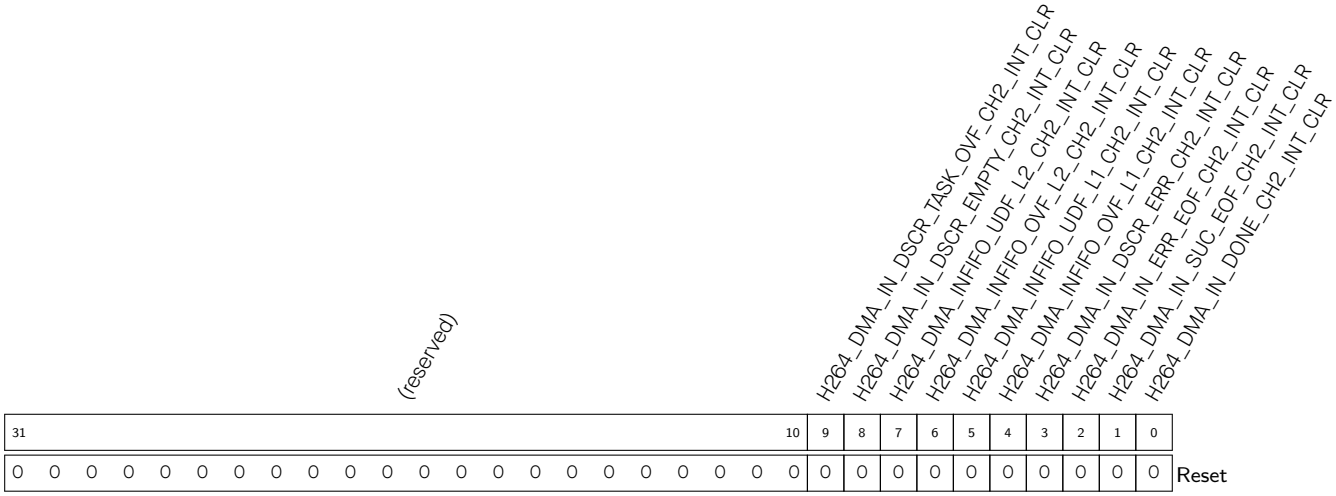
H264_DMA_INFIFO_OVF_L2_CH2_INT_ST The masked interrupt status of H264_DMA_INFIFO_OVF_L2_CH2_INT. (RO)

H264_DMA_INFIFO_UDF_L2_CH2_INT_ST The masked interrupt status of H264_DMA_INFIFO_UDF_L2_CH2_INT. (RO)

H264_DMA_IN_DSCR_EMPTY_CH2_INT_ST The masked interrupt status of H264_DMA_IN_DSCR_EMPTY_CH2_INT. (RO)

H264_DMA_IN_DSCR_TASK_OVF_CH2_INT_ST The masked interrupt status of H264_DMA_IN_DSCR_TASK_OVF_CH2_INT. (RO)

Register 41.204. H264_DMA_IN_INT_CLR_CH2_REG (0x0710)



H264_DMA_IN_DONE_CH2_INT_CLR Write 1 to clear H264_DMA_IN_DONE_CH2_INT. (WT)

H264_DMA_IN_SUC_EOF_CH2_INT_CLR Write 1 to clear H264_DMA_IN_SUC_EOF_CH2_INT. (WT)

H264_DMA_IN_ERR_EOF_CH2_INT_CLR Write 1 to clear H264_DMA_IN_ERR_EOF_CH2_INT. (WT)

H264_DMA_IN_DSCR_ERR_CH2_INT_CLR Write 1 to clear H264_DMA_IN_DSCR_ERR_CH2_INT. (WT)

H264_DMA_INFIFO_OVF_L1_CH2_INT_CLR Write 1 to clear H264_DMA_INFIFO_OVF_L1_CH2_INT. (WT)

H264_DMA_INFIFO_UDF_L1_CH2_INT_CLR Write 1 to clear H264_DMA_INFIFO_UDF_L1_CH2_INT. (WT)

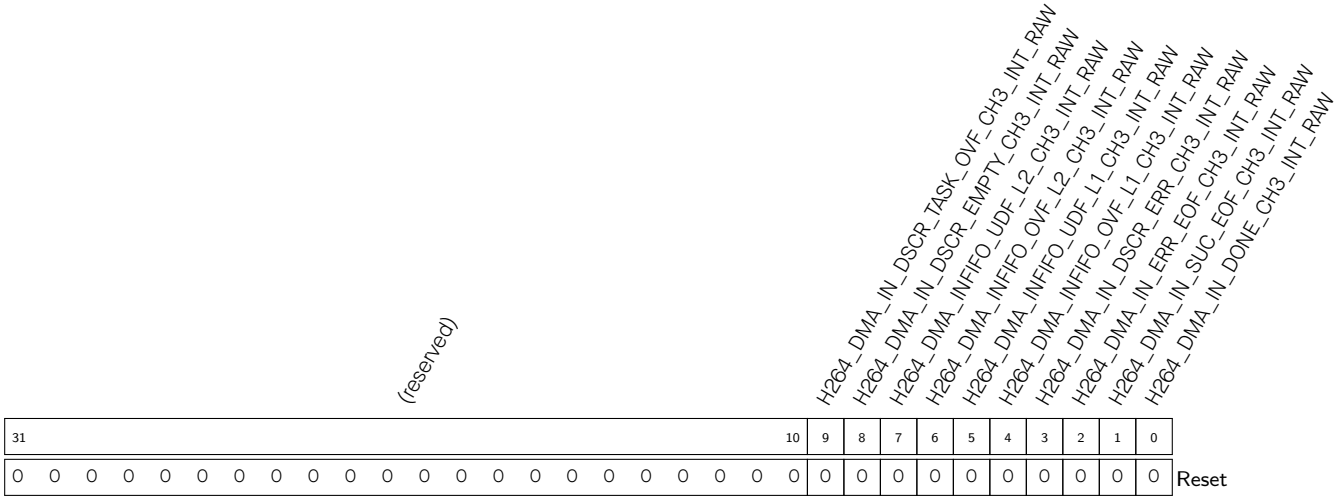
H264_DMA_INFIFO_OVF_L2_CH2_INT_CLR Write 1 to clear H264_DMA_INFIFO_OVF_L2_CH2_INT. (WT)

H264_DMA_INFIFO_UDF_L2_CH2_INT_CLR Write 1 to clear H264_DMA_INFIFO_UDF_L2_CH2_INT. (WT)

H264_DMA_IN_DSCR_EMPTY_CH2_INT_CLR Write 1 to clear H264_DMA_IN_DSCR_EMPTY_CH2_INT. (WT)

H264_DMA_IN_DSCR_TASK_OVF_CH2_INT_CLR Write 1 to clear H264_DMA_IN_DSCR_TASK_OVF_CH2_INT. (WT)

Register 41.205. H264_DMA_IN_INT_RAW_CH3_REG (0x0804)



H264_DMA_IN_DONE_CH3_INT_RAW The raw interrupt status of H264_DMA_IN_DONE_CH3_INT. (R/WTC/SS)

H264_DMA_IN_SUC_EOF_CH3_INT_RAW The raw interrupt status of H264_DMA_IN_SUC_EOF_CH3_INT. (R/WTC/SS)

H264_DMA_IN_ERR_EOF_CH3_INT_RAW The raw interrupt status of H264_DMA_IN_ERR_EOF_CH3_INT. (R/WTC/SS)

H264_DMA_IN_DSCR_ERR_CH3_INT_RAW The raw interrupt status of H264_DMA_IN_DSCR_ERR_CH3_INT. channel 3. (R/WTC/SS)

H264_DMA_INFIFO_OVF_L1_CH3_INT_RAW The raw interrupt status of H264_DMA_INFIFO_OVF_L1_CH3_INT. (R/WTC/SS)

H264_DMA_INFIFO_UDF_L1_CH3_INT_RAW The raw interrupt status of H264_DMA_INFIFO_UDF_L1_CH3_INT. (R/WTC/SS)

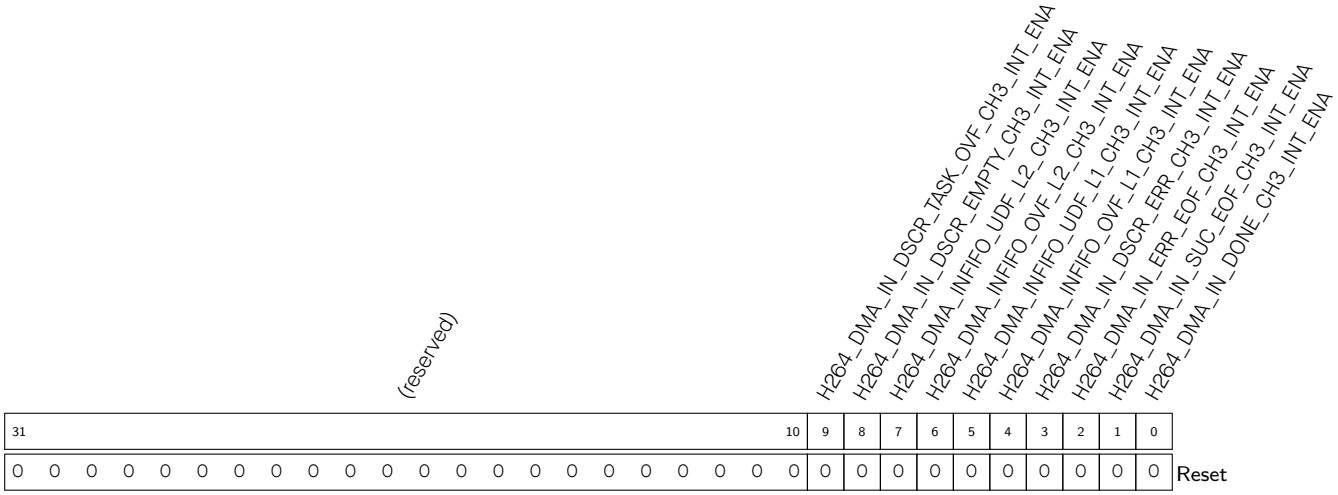
H264_DMA_INFIFO_OVF_L2_CH3_INT_RAW The raw interrupt status of H264_DMA_INFIFO_OVF_L2_CH3_INT. (R/WTC/SS)

H264_DMA_INFIFO_UDF_L2_CH3_INT_RAW The raw interrupt status of H264_DMA_INFIFO_UDF_L2_CH3_INT. (R/WTC/SS)

H264_DMA_IN_DSCR_EMPTY_CH3_INT_RAW The raw interrupt status of H264_DMA_IN_DSCR_EMPTY_CH3_INT. (R/WTC/SS)

H264_DMA_IN_DSCR_TASK_OVF_CH3_INT_RAW The raw interrupt status of H264_DMA_IN_DSCR_TASK_OVF_CH3_INT. (R/WTC/SS)

Register 41.206. H264_DMA_IN_INT_ENA_CH3_REG (0x0808)



H264_DMA_IN_DONE_CH3_INT_ENA Write 1 to enable H264_DMA_IN_DONE_CH3_INT. (R/W)

H264_DMA_IN_SUC_EOF_CH3_INT_ENA Write 1 to enable H264_DMA_IN_SUC_EOF_CH3_INT. (R/W)

H264_DMA_IN_ERR_EOF_CH3_INT_ENA Write 1 to enable H264_DMA_IN_ERR_EOF_CH3_INT. (R/W)

H264_DMA_IN_DSCR_ERR_CH3_INT_ENA Write 1 to enable H264_DMA_IN_DSCR_ERR_CH3_-INT. (R/W)

H264_DMA_INFIFO_OVF_L1_CH3_INT_ENA Write 1 to enable H264_DMA_INFIFO_OVF_L1_CH3_-INT. (R/W)

H264_DMA_INFIFO_UDF_L1_CH3_INT_ENA Write 1 to enable H264_DMA_INFIFO_UDF_L1_CH3_-INT. (R/W)

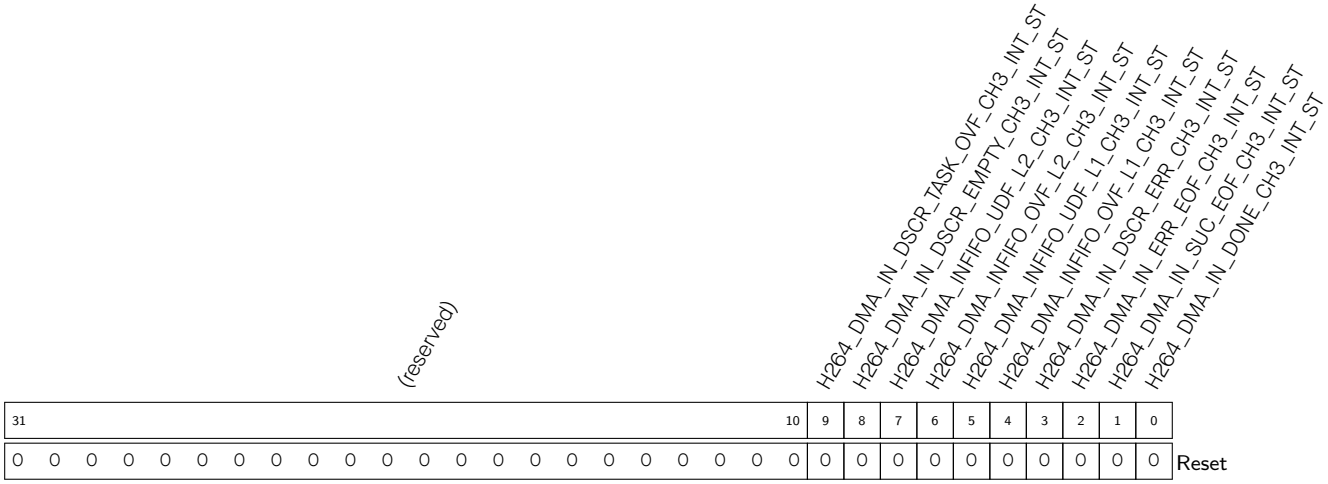
H264_DMA_INFIFO_OVF_L2_CH3_INT_ENA Write 1 to enable H264_DMA_INFIFO_OVF_L2_-CH3_INT. (R/W)

H264_DMA_INFIFO_UDF_L2_CH3_INT_ENA Write 1 to enable H264_DMA_INFIFO_UDF_L2_-CH3_INT. (R/W)

H264_DMA_IN_DSCR_EMPTY_CH3_INT_ENA Write 1 to enable H264_DMA_IN_DSCR_EMPTY_-CH3_INT. (R/W)

H264_DMA_IN_DSCR_TASK_OVF_CH3_INT_ENA Write 1 to enable H264_DMA_IN_DSCR_TASK_-OVF_CH3_INT. (R/W)

Register 41.207. H264_DMA_IN_INT_ST_CH3_REG (0x080C)



H264_DMA_IN_DONE_CH3_INT_ST The masked interrupt status of H264_DMA_IN_DONE_CH3_INT. (RO)

H264_DMA_IN_SUC_EOF_CH3_INT_ST The masked interrupt status of H264_DMA_IN_SUC_EOF_CH3_INT. (RO)

H264_DMA_IN_ERR_EOF_CH3_INT_ST The masked interrupt status of H264_DMA_IN_ERR_EOF_CH3_INT. (RO)

H264_DMA_IN_DSCR_ERR_CH3_INT_ST The masked interrupt status of H264_DMA_IN_DSCR_ERR_CH3_INT. (RO)

H264_DMA_INFIFO_OVF_L1_CH3_INT_ST The masked interrupt status of H264_DMA_INFIFO_OVF_L1_CH3_INT. (RO)

H264_DMA_INFIFO_UDF_L1_CH3_INT_ST The masked interrupt status of H264_DMA_INFIFO_UDF_L1_CH3_INT. (RO)

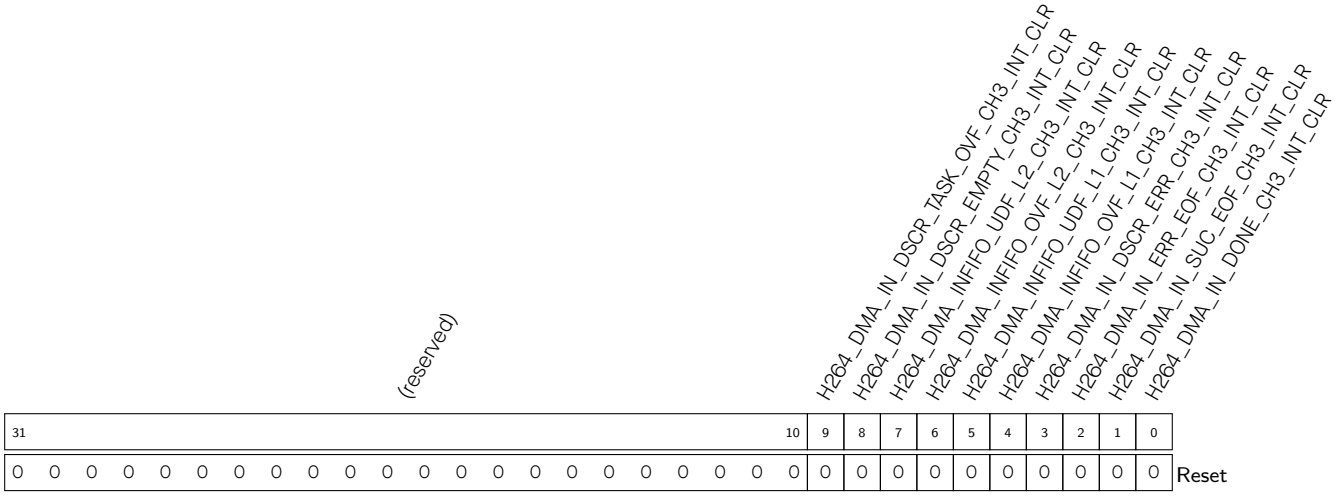
H264_DMA_INFIFO_OVF_L2_CH3_INT_ST The masked interrupt status of H264_DMA_INFIFO_OVF_L2_CH3_INT. (RO)

H264_DMA_INFIFO_UDF_L2_CH3_INT_ST The masked interrupt status of H264_DMA_INFIFO_UDF_L2_CH3_INT. (RO)

H264_DMA_IN_DSCR_EMPTY_CH3_INT_ST The masked interrupt status of H264_DMA_IN_DSCR_EMPTY_CH3_INT. (RO)

H264_DMA_IN_DSCR_TASK_OVF_CH3_INT_ST The masked interrupt status of H264_DMA_IN_DSCR_TASK_OVF_CH3_INT. (RO)

Register 41.208. H264_DMA_IN_INT_CLR_CH3_REG (0x0810)



H264_DMA_IN_DONE_CH3_INT_CLR Write 1 to clear H264_DMA_IN_DONE_CH3_INT. (WT)

H264_DMA_IN_SUC_EOF_CH3_INT_CLR Write 1 to clear H264_DMA_IN_SUC_EOF_CH3_INT. (WT)

H264_DMA_IN_ERR_EOF_CH3_INT_CLR Write 1 to clear H264_DMA_IN_ERR_EOF_CH3_INT. (WT)

H264_DMA_IN_DSCR_ERR_CH3_INT_CLR Write 1 to clear H264_DMA_IN_DSCR_ERR_CH3_INT. (WT)

H264_DMA_INFIFO_OVF_L1_CH3_INT_CLR Write 1 to clear H264_DMA_INFIFO_OVF_L1_CH3_INT. (WT)

H264_DMA_INFIFO_UDF_L1_CH3_INT_CLR Write 1 to clear H264_DMA_INFIFO_UDF_L1_CH3_INT. (WT)

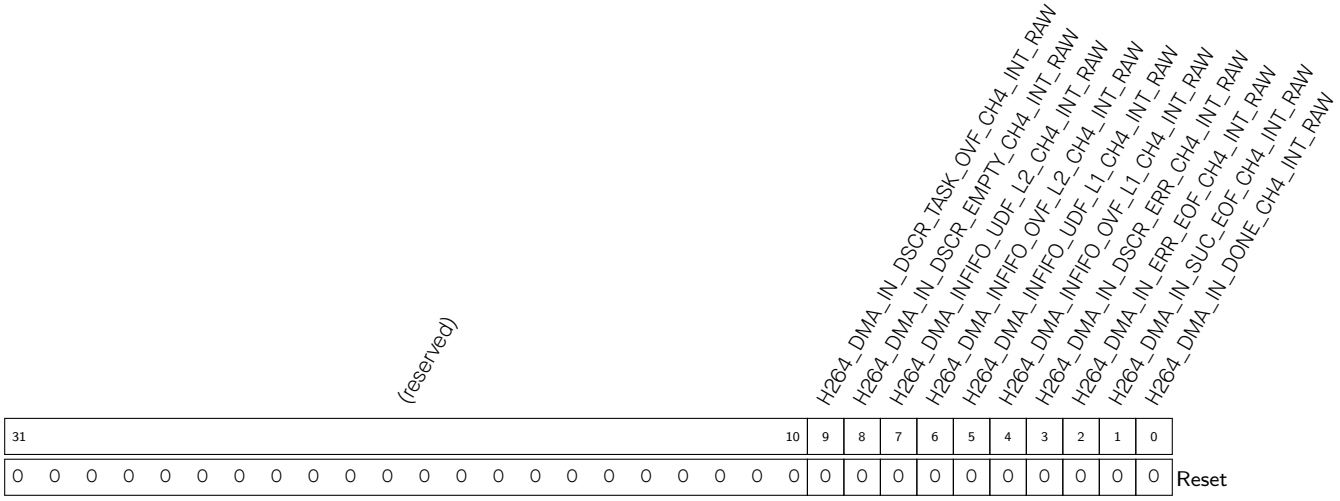
H264_DMA_INFIFO_OVF_L2_CH3_INT_CLR Write 1 to clear H264_DMA_INFIFO_OVF_L2_CH3_INT. (WT)

H264_DMA_INFIFO_UDF_L2_CH3_INT_CLR Write 1 to clear H264_DMA_INFIFO_UDF_L2_CH3_INT. (WT)

H264_DMA_IN_DSCR_EMPTY_CH3_INT_CLR Write 1 to clear H264_DMA_IN_DSCR_EMPTY_CH3_INT. (WT)

H264_DMA_IN_DSCR_TASK_OVF_CH3_INT_CLR Write 1 to clear H264_DMA_IN_DSCR_TASK_OVF_CH3_INT. (WT)

Register 41.209. H264_DMA_IN_INT_RAW_CH4_REG (0x0904)



H264_DMA_IN_DONE_CH4_INT_RAW The raw interrupt status of H264_DMA_IN_DONE_CH4_INT. (R/WTC/SS)

H264_DMA_IN_SUC_EOF_CH4_INT_RAW The raw interrupt status of H264_DMA_IN_SUC_EOF_CH4_INT. (R/WTC/SS)

H264_DMA_IN_ERR_EOF_CH4_INT_RAW The raw interrupt status of H264_DMA_IN_ERR_EOF_CH4_INT. (R/WTC/SS)

H264_DMA_IN_DSCR_ERR_CH4_INT_RAW The raw interrupt status of H264_DMA_IN_DSCR_ERR_CH4_INT. (R/WTC/SS)

H264_DMA_INFIFO_OVF_L1_CH4_INT_RAW The raw interrupt status of H264_DMA_INFIFO_OVF_L1_CH4_INT. (R/WTC/SS)

H264_DMA_INFIFO_UDF_L1_CH4_INT_RAW The raw interrupt status of H264_DMA_INFIFO_UDF_L1_CH4_INT. (R/WTC/SS)

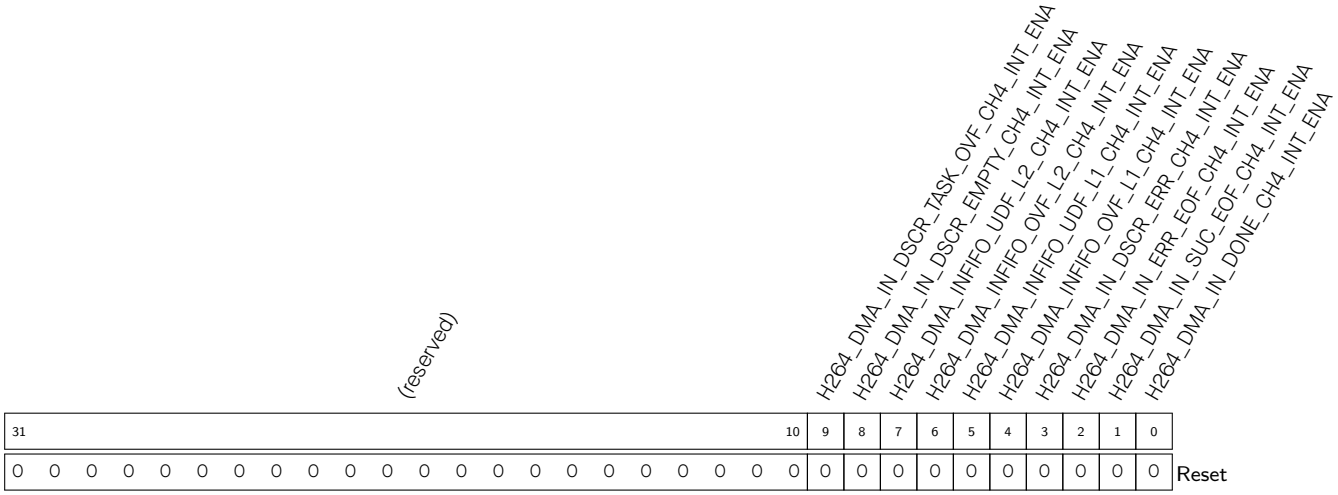
H264_DMA_INFIFO_OVF_L2_CH4_INT_RAW The raw interrupt status of H264_DMA_INFIFO_OVF_L2_CH4_INT. (R/WTC/SS)

H264_DMA_INFIFO_UDF_L2_CH4_INT_RAW The raw interrupt status of H264_DMA_INFIFO_UDF_L2_CH4_INT. (R/WTC/SS)

H264_DMA_IN_DSCR_EMPTY_CH4_INT_RAW The raw interrupt status of H264_DMA_IN_DSCR_EMPTY_CH4_INT. (R/WTC/SS)

H264_DMA_IN_DSCR_TASK_OVF_CH4_INT_RAW The raw interrupt status of H264_DMA_IN_DSCR_TASK_OVF_CH4_INT. (R/WTC/SS)

Register 41.210. H264_DMA_IN_INT_ENA_CH4_REG (0x0908)



H264_DMA_IN_DONE_CH4_INT_ENA Write 1 to enable H264_DMA_IN_DONE_CH4_INT. (R/W)

H264_DMA_IN_SUC_EOF_CH4_INT_ENA Write 1 to enable H264_DMA_IN_SUC_EOF_CH4_INT. (R/W)

H264_DMA_IN_ERR_EOF_CH4_INT_ENA Write 1 to enable H264_DMA_IN_ERR_EOF_CH4_INT. (R/W)

H264_DMA_IN_DSCR_ERR_CH4_INT_ENA Write 1 to enable H264_DMA_IN_DSCR_ERR_CH4_-INT. (R/W)

H264_DMA_INFIFO_OVF_L1_CH4_INT_ENA Write 1 to enable H264_DMA_INFIFO_OVF_L1_CH4_-INT. (R/W)

H264_DMA_INFIFO_UDF_L1_CH4_INT_ENA Write 1 to enable H264_DMA_INFIFO_UDF_L1_CH4_-INT. (R/W)

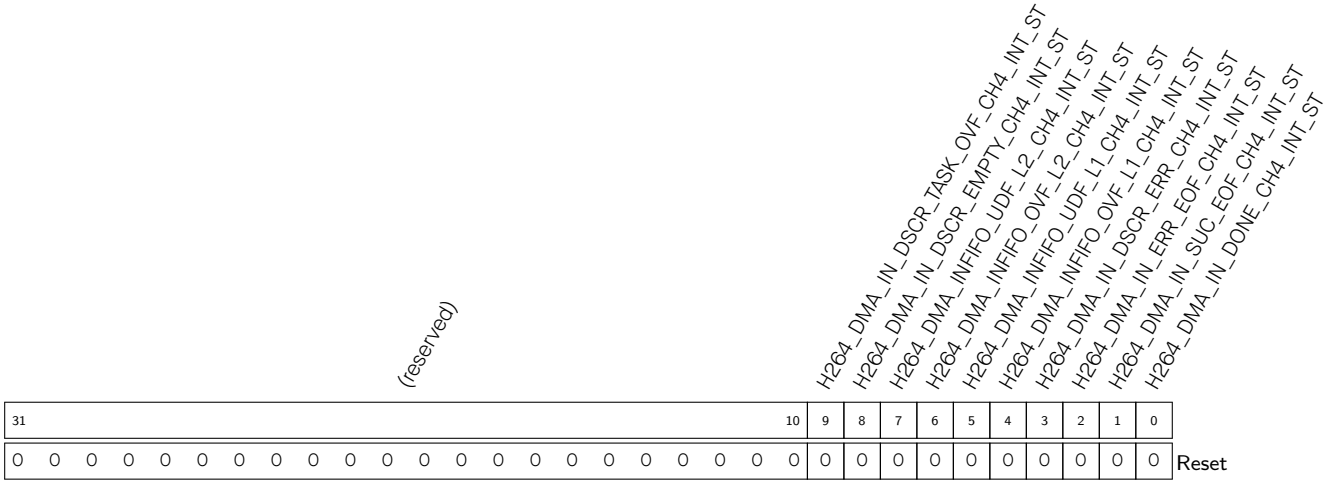
H264_DMA_INFIFO_OVF_L2_CH4_INT_ENA Write 1 to enable H264_DMA_INFIFO_OVF_L2_-CH4_INT. (R/W)

H264_DMA_INFIFO_UDF_L2_CH4_INT_ENA Write 1 to enable H264_DMA_INFIFO_UDF_L2_-CH4_INT. (R/W)

H264_DMA_IN_DSCR_EMPTY_CH4_INT_ENA Write 1 to enable H264_DMA_IN_DSCR_EMPTY_-CH4_INT. (R/W)

H264_DMA_IN_DSCR_TASK_OVF_CH4_INT_ENA Write 1 to enable H264_DMA_IN_DSCR_TASK_-OVF_CH4_INT. (R/W)

Register 41.211. H264_DMA_IN_INT_ST_CH4_REG (0x090C)



H264_DMA_IN_DONE_CH4_INT_ST The masked interrupt status of H264_DMA_IN_DONE_CH4_INT. (RO)

H264_DMA_IN_SUC_EOF_CH4_INT_ST The masked interrupt status of H264_DMA_IN_SUC_EOF_CH4_INT. (RO)

H264_DMA_IN_ERR_EOF_CH4_INT_ST The masked interrupt status of H264_DMA_IN_ERR_EOF_CH4_INT. (RO)

H264_DMA_IN_DSCR_ERR_CH4_INT_ST The masked interrupt status of H264_DMA_IN_DSCR_ERR_CH4_INT. (RO)

H264_DMA_INFIFO_OVF_L1_CH4_INT_ST The masked interrupt status of H264_DMA_INFIFO_OVF_L1_CH4_INT. (RO)

H264_DMA_INFIFO_UDF_L1_CH4_INT_ST The masked interrupt status of H264_DMA_INFIFO_UDF_L1_CH4_INT. (RO)

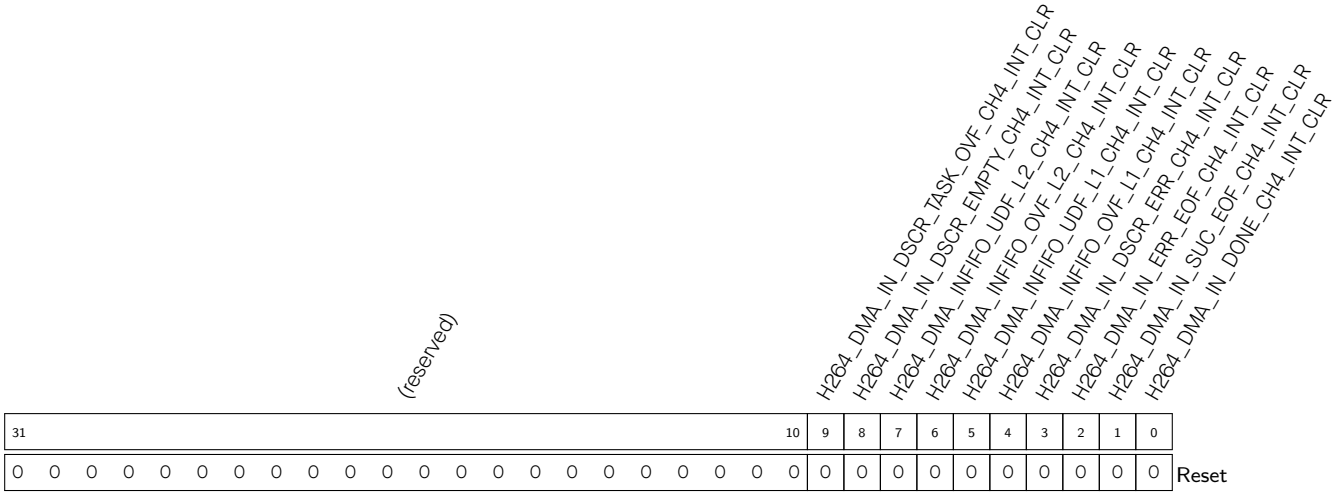
H264_DMA_INFIFO_OVF_L2_CH4_INT_ST The masked interrupt status of H264_DMA_INFIFO_OVF_L2_CH4_INT. (RO)

H264_DMA_INFIFO_UDF_L2_CH4_INT_ST The masked interrupt status of H264_DMA_INFIFO_UDF_L2_CH4_INT. (RO)

H264_DMA_IN_DSCR_EMPTY_CH4_INT_ST The masked interrupt status of H264_DMA_IN_DSCR_EMPTY_CH4_INT. (RO)

H264_DMA_IN_DSCR_TASK_OVF_CH4_INT_ST The masked interrupt status of H264_DMA_IN_DSCR_TASK_OVF_CH4_INT. (RO)

Register 41.212. H264_DMA_IN_INT_CLR_CH4_REG (0x0910)



H264_DMA_IN_DONE_CH4_INT_CLR Write 1 to clear H264_DMA_IN_DONE_CH4_INT. (WT)

H264_DMA_IN_SUC_EOF_CH4_INT_CLR Write 1 to clear H264_DMA_IN_SUC_EOF_CH4_INT. (WT)

H264_DMA_IN_ERR_EOF_CH4_INT_CLR Write 1 to clear H264_DMA_IN_ERR_EOF_CH4_INT. (WT)

H264_DMA_IN_DSCR_ERR_CH4_INT_CLR Write 1 to clear H264_DMA_IN_DSCR_ERR_CH4_INT. (WT)

H264_DMA_INFIFO_OVF_L1_CH4_INT_CLR Write 1 to clear H264_DMA_INFIFO_OVF_L1_CH4_INT. (WT)

H264_DMA_INFIFO_UDF_L1_CH4_INT_CLR Write 1 to clear H264_DMA_INFIFO_UDF_L1_CH4_INT. (WT)

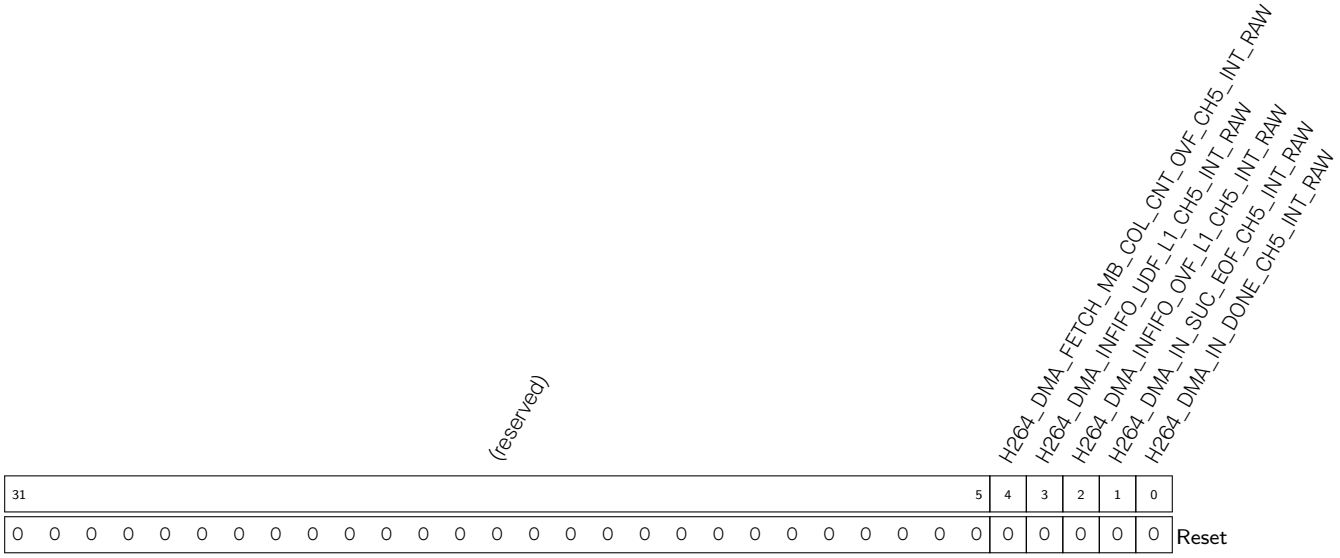
H264_DMA_INFIFO_OVF_L2_CH4_INT_CLR Write 1 to clear H264_DMA_INFIFO_OVF_L2_CH4_INT. (WT)

H264_DMA_INFIFO_UDF_L2_CH4_INT_CLR Write 1 to clear H264_DMA_INFIFO_UDF_L2_CH4_INT. (WT)

H264_DMA_IN_DSCR_EMPTY_CH4_INT_CLR Write 1 to clear H264_DMA_IN_DSCR_EMPTY_CH4_INT. (WT)

H264_DMA_IN_DSCR_TASK_OVF_CH4_INT_CLR Write 1 to clear H264_DMA_IN_DSCR_TASK_OVF_CH4_INT. (WT)

Register 41.213. H264_DMA_IN_INT_RAW_CH5_REG (0x0A10)



H264_DMA_IN_DONE_CH5_INT_RAW The raw interrupt status of H264_DMA_IN_DONE_CH5_INT. (R/WTC/SS)

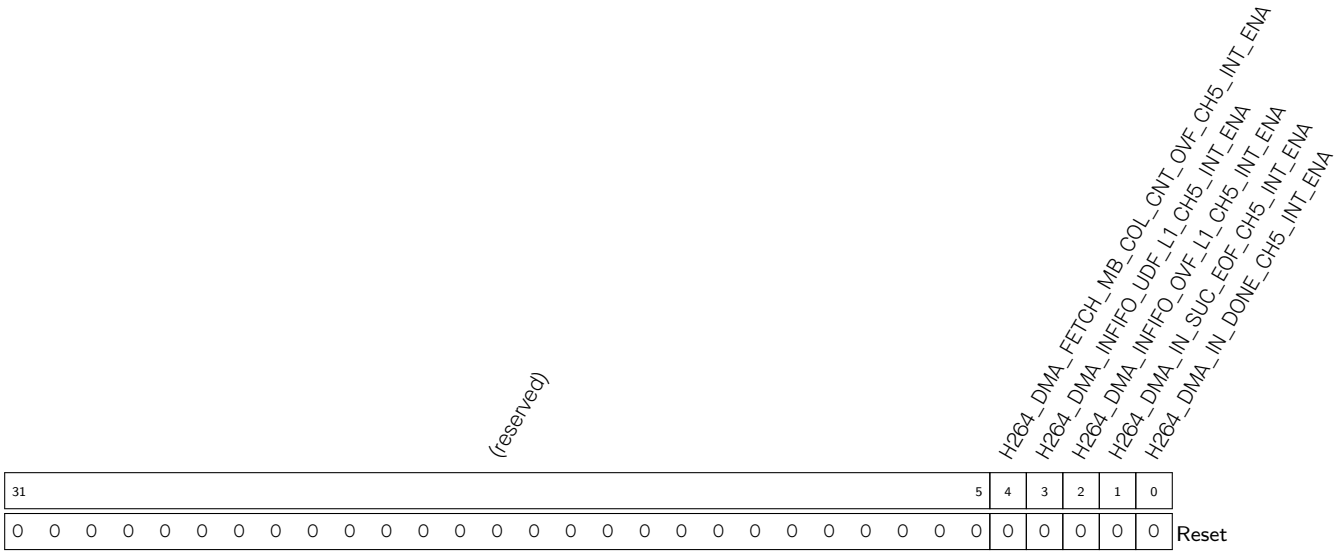
H264_DMA_IN_SUC_EOF_CH5_INT_RAW The raw interrupt status of H264_DMA_IN_SUC_EOF_CH5_INT. (R/WTC/SS)

H264_DMA_INFIFO_OVF_L1_CH5_INT_RAW The raw interrupt status of H264_DMA_INFIFO_OVF_L1_CH5_INT. (R/WTC/SS)

H264_DMA_INFIFO_UDF_L1_CH5_INT_RAW The raw interrupt status of H264_DMA_INFIFO_UDF_L1_CH5_INT. (R/WTC/SS)

H264_DMA_FETCH_MB_COL_CNT_OVF_CH5_INT_RAW The raw interrupt status of H264_DMA_FETCH_MB_COL_CNT_OVF_CH5_INT. (R/WTC/SS)

Register 41.214. H264_DMA_IN_INT_ENA_CH5_REG (0x0A14)



H264_DMA_IN_DONE_CH5_INT_ENA Write 1 to enable H264_DMA_IN_DONE_CH5_INT. (R/W)

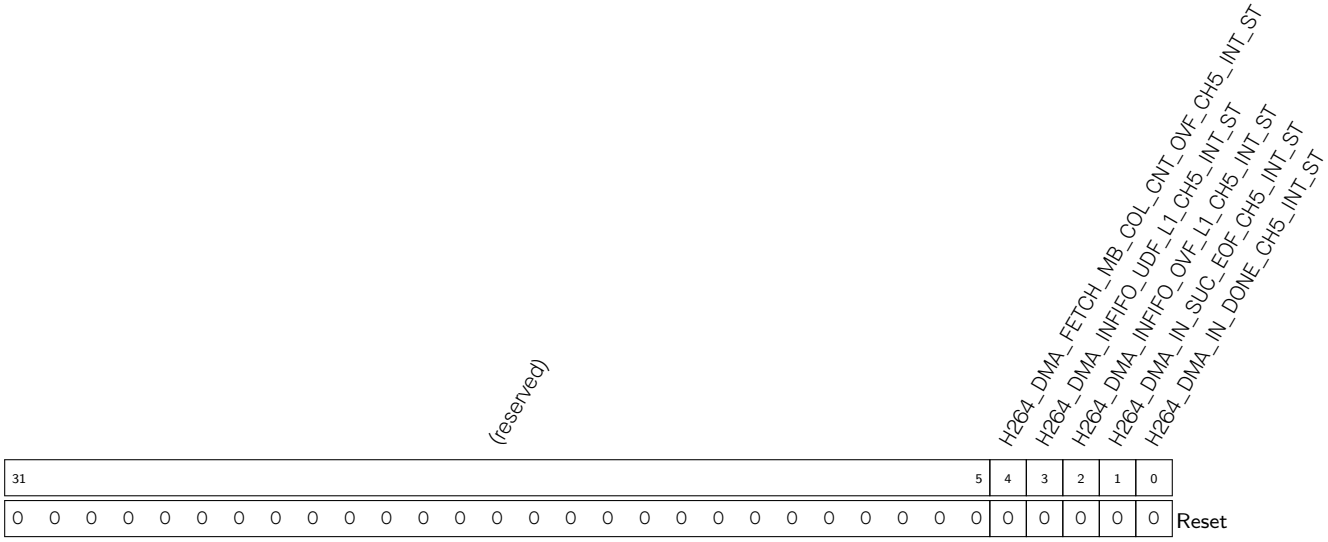
H264_DMA_IN_SUC_EOF_CH5_INT_ENA Write 1 to enable H264_DMA_IN_SUC_EOF_CH5_INT. (R/W)

H264_DMA_INFIFO_OVF_L1_CH5_INT_ENA Write 1 to enable H264_DMA_INFIFO_OVF_L1_CH5_-INT. (R/W)

H264_DMA_INFIFO_UDF_L1_CH5_INT_ENA Write 1 to enable H264_DMA_INFIFO_UDF_L1_CH5_-INT. (R/W)

H264_DMA_FETCH_MB_COL_CNT_OVF_CH5_INT_ENA Write 1 to enable H264_DMA_FETCH_-MB_COL_CNT_OVF_CH5_INT. (R/W)

Register 41.215. H264_DMA_IN_INT_ST_CH5_REG (0x0A18)



H264_DMA_IN_DONE_CH5_INT_ST The masked interrupt status of H264_DMA_IN_DONE_CH5_INT. (RO)

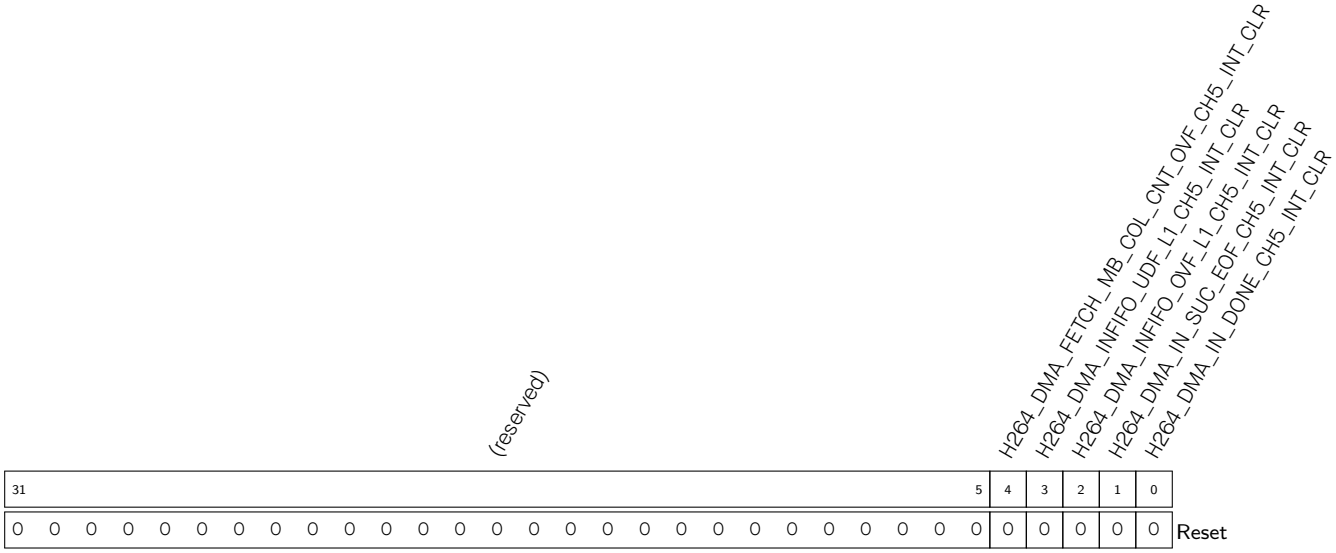
H264_DMA_IN_SUC_EOF_CH5_INT_ST The masked interrupt status of H264_DMA_IN_SUC_EOF_CH5_INT. (RO)

H264_DMA_INFIFO_OVF_L1_CH5_INT_ST The masked interrupt status of H264_DMA_INFIFO_OVF_L1_CH5_INT. (RO)

H264_DMA_INFIFO_UDF_L1_CH5_INT_ST The masked interrupt status of H264_DMA_INFIFO_UDF_L1_CH5_INT. (RO)

H264_DMA_FETCH_MB_COL_CNT_OVF_CH5_INT_ST The masked interrupt status of H264_DMA_FETCH_MB_COL_CNT_OVF_CH5_INT. (RO)

Register 41.216. H264_DMA_IN_INT_CLR_CH5_REG (0x0A1C)



H264_DMA_IN_DONE_CH5_INT_CLR Write 1 to clear H264_DMA_IN_DONE_CH5_INT. (WT)

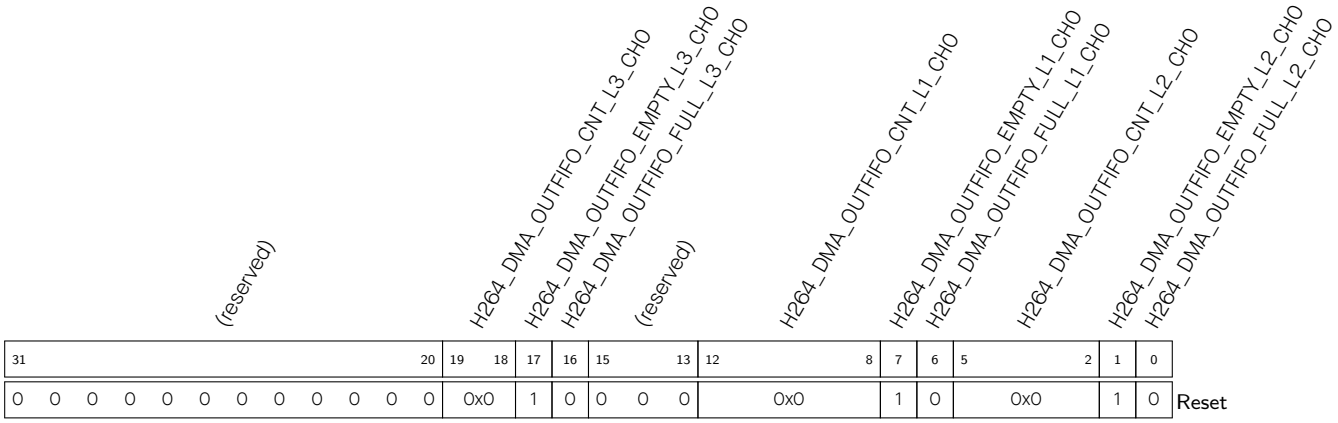
H264_DMA_IN_SUC_EOF_CH5_INT_CLR Write 1 to clear H264_DMA_IN_SUC_EOF_CH5_INT. (WT)

H264_DMA_INFIFO_OVF_L1_CH5_INT_CLR Write 1 to clear H264_DMA_INFIFO_OVF_L1_CH5_-INT. (WT)

H264_DMA_INFIFO_UDF_L1_CH5_INT_CLR Write 1 to clear H264_DMA_INFIFO_UDF_L1_CH5_-INT. (WT)

H264_DMA_FETCH_MB_COL_CNT_OVF_CH5_INT_CLR Write 1 to clear H264_DMA_FETCH_MB_COL_CNT_OVF_CH5. (WT)

Register 41.217. H264_DMA_OUTFIFO_STATUS_CH0_REG (0x0014)



H264_DMA_OUTFIFO_FULL_L2_CH0 Represents whether l2 FIFO of TX channel 0 is full.

0: Not Full

1: Full

(RO)

H264_DMA_OUTFIFO_EMPTY_L2_CH0 Represents whether l2 FIFO of TX channel 0 is empty.

0: Not empty

1: Empty

(RO)

H264_DMA_OUTFIFO_CNT_L2_CH0 Represents the data quantity in l2 FIFO for TX channel 0. Measurement unit: 16 bytes. (RO)

H264_DMA_OUTFIFO_FULL_L1_CH0 Represents whether l1 FIFO of TX channel 0 is full.

0: Not Full

1: Full

(RO)

H264_DMA_OUTFIFO_EMPTY_L1_CH0 Represents whether l1 FIFO of TX channel 0 is empty.

0: Not empty

1: Empty

(RO)

H264_DMA_OUTFIFO_CNT_L1_CH0 Represents the data quantity in l1 FIFO for TX channel 0. Measurement unit: byte. (RO)

H264_DMA_OUTFIFO_FULL_L3_CH0 Represents whether l3 FIFO of TX channel 0 is full.

0: Not Full

1: Full

(RO)

H264_DMA_OUTFIFO_EMPTY_L3_CH0 Represents whether l3 FIFO of TX channel 0 is empty.

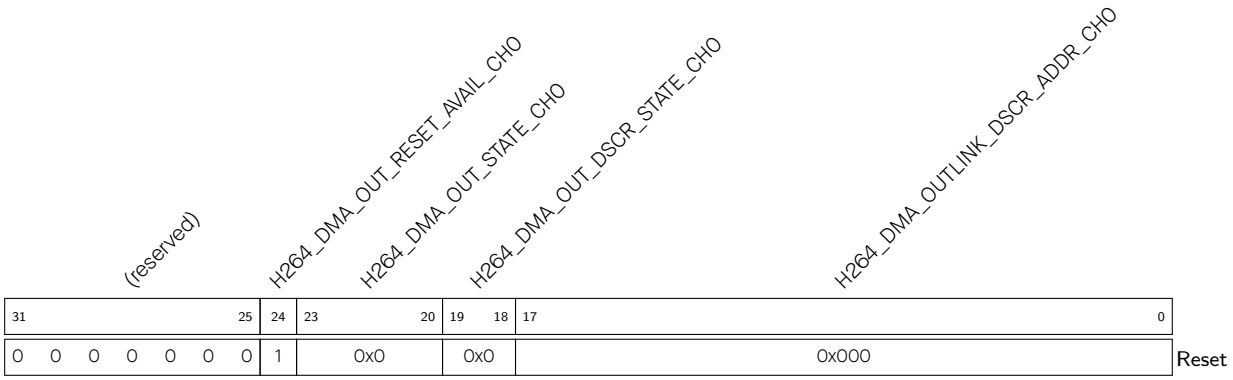
0: Not empty

1: Empty

(RO)

H264_DMA_OUTFIFO_CNT_L3_CH0 Represents the data quantity in l3 FIFO for TX channel 0. Measurement unit: 8 bytes. (RO)

Register 41.218. H264_DMA_OUT_STATE_CH0_REG (0x0024)



H264_DMA_OUTLINK_DSCR_ADDR_CH0 Represents the current outlink descriptor's address for TX channel 0. (RO)

H264_DMA_OUT_DSCR_STATE_CH0 Represents the current state of the descriptor state machine for TX channel 0. (RO)

H264_DMA_OUT_STATE_CH0 Represents the current control module state machine state for TX channel 0. (RO)

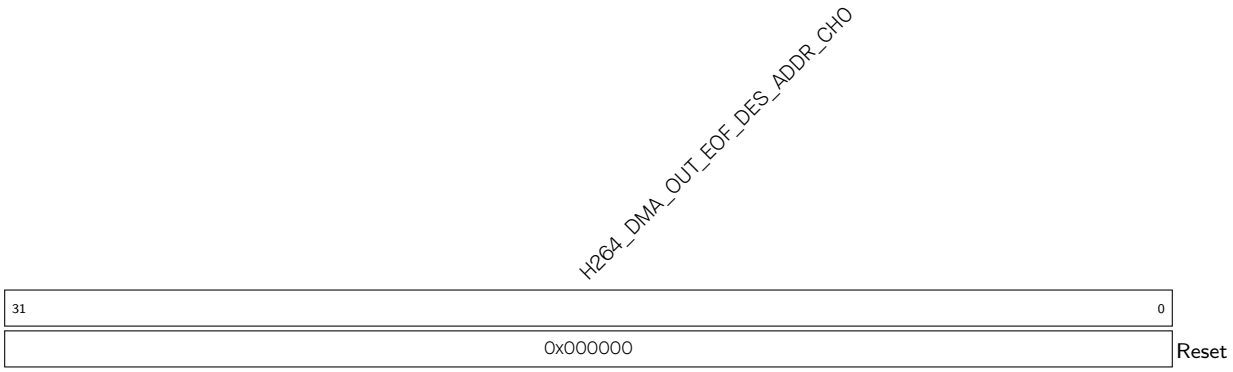
H264_DMA_OUT_RESET_AVAIL_CH0 Represents whether it is safe to reset the channel.

0: Unsafe

1: Safe

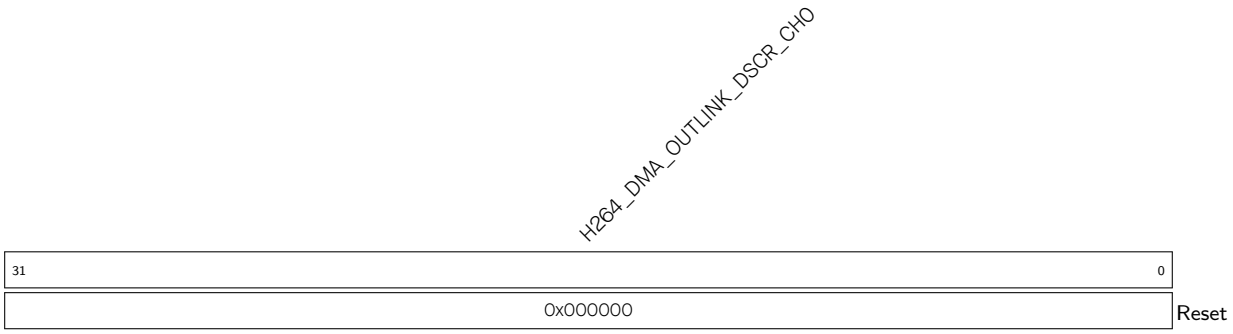
(RO)

Register 41.219. H264_DMA_OUT_EOF_DES_ADDR_CH0_REG (0x0028)



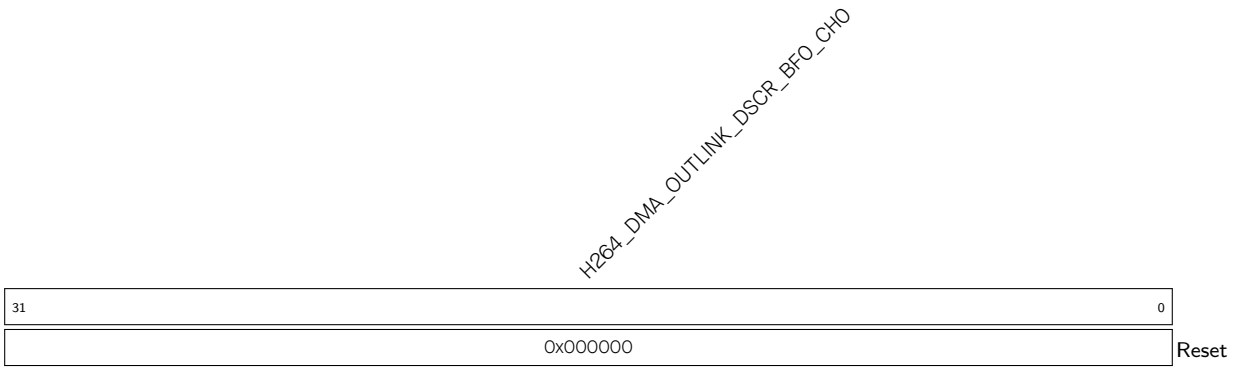
H264_DMA_OUT_EOF_DES_ADDR_CH0 Represents the address of the outlink descriptor when the EOF bit in this descriptor is 1. (RO)

Register 41.220. H264_DMA_OUT_DSCR_CH0_REG (0x002C)



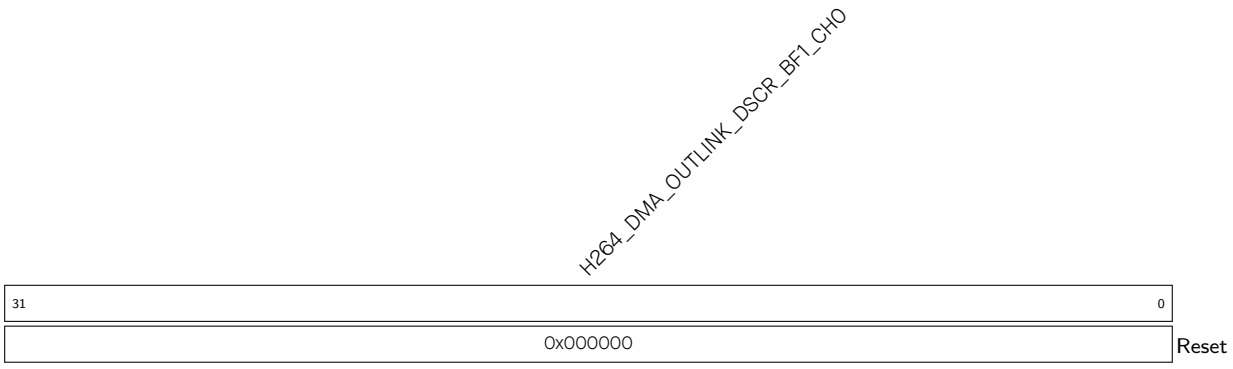
H264_DMA_OUTLINK_DSCR_CH0 Represents the address of the next descriptor y+1 pointed by the current descriptor that is pre-read. (RO)

Register 41.221. H264_DMA_OUT_DSCR_BF0_CH0_REG (0x0030)



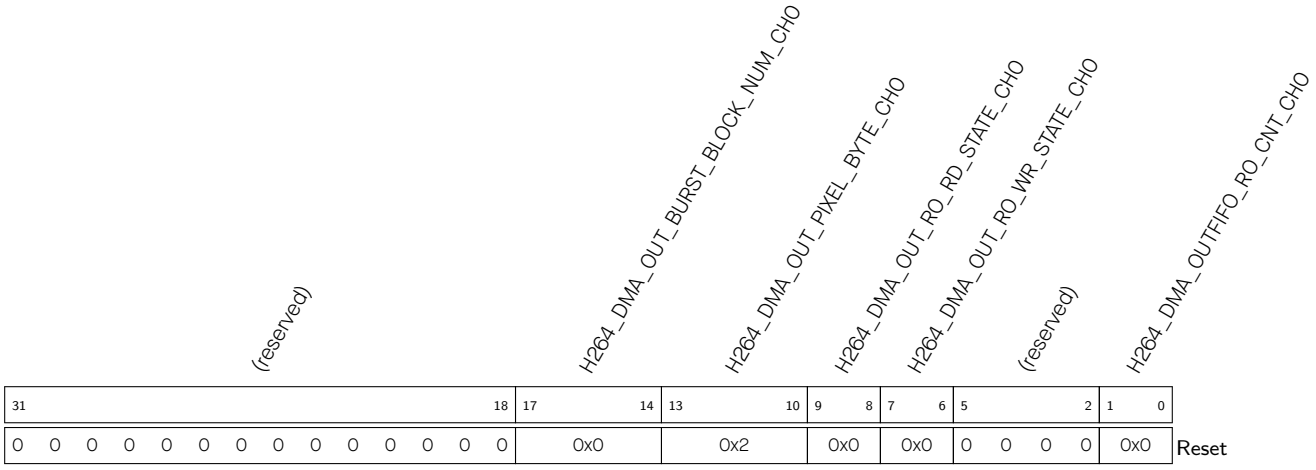
H264_DMA_OUTLINK_DSCR_BF0_CH0 Represents the address of the current descriptor y that is pre-read. (RO)

Register 41.222. H264_DMA_OUT_DSCR_BF1_CH0_REG (0x0034)



H264_DMA_OUTLINK_DSCR_BF1_CH0 Represents the address of the previous descriptor y-1 that is pre-read. (RO)

Register 41.223. H264_DMA_OUT_RO_STATUS_CH0_REG (0x0040)



H264_DMA_OUTFIFO_RO_CNT_CH0 Represents the 8-byte number of the data in the reordered TX FIFO for channel 0. (RO)

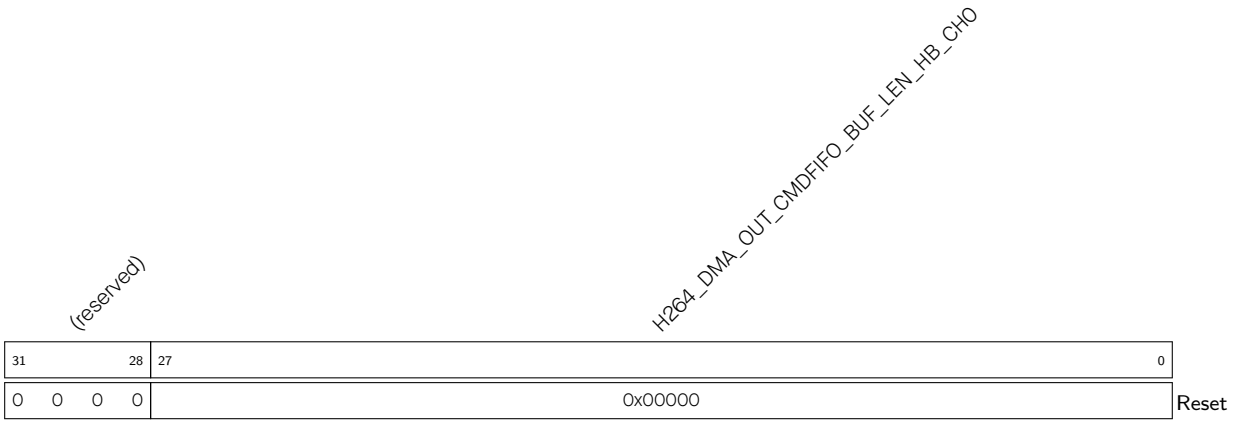
H264_DMA_OUT_RO_WR_STATE_CH0 Represents the state of writing RAM for reorder. (RO)

H264_DMA_OUT_RO_RD_STATE_CH0 Represents the state of reading RAM for reorder. (RO)

H264_DMA_OUT_PIXEL_BYTE_CH0 Represents the number of bytes contained in a pixel at TX channel 0.
0: 1 byte
1: 1.5 bytes
2: 2 bytes
3: 2.5 bytes
4: 3 bytes
5: 4 bytes
(RO)

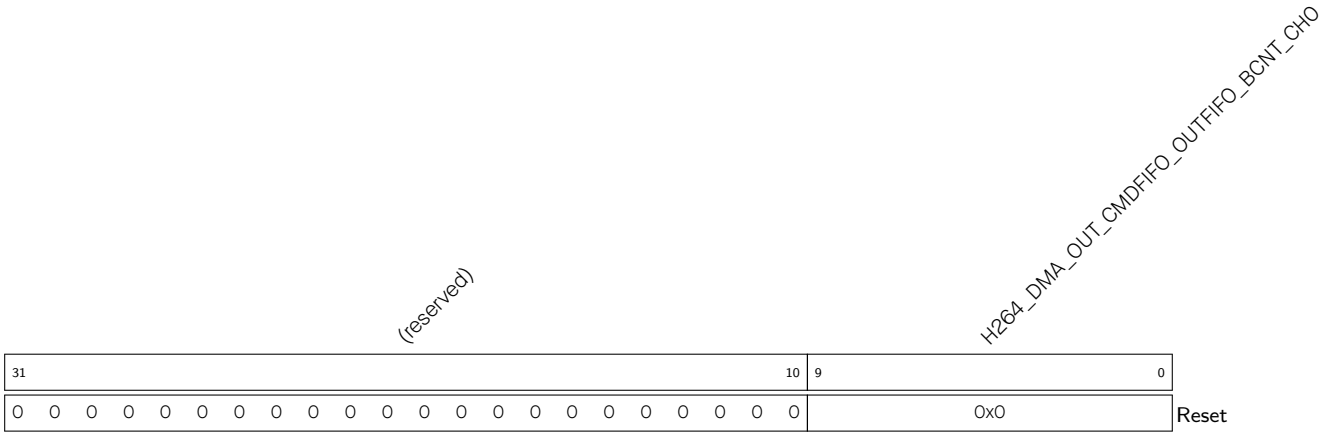
H264_DMA_OUT_BURST_BLOCK_NUM_CH0 Reserved. (RO)

Register 41.224. H264_DMA_OUT_BUF_LEN_CH0_REG (0x0070)



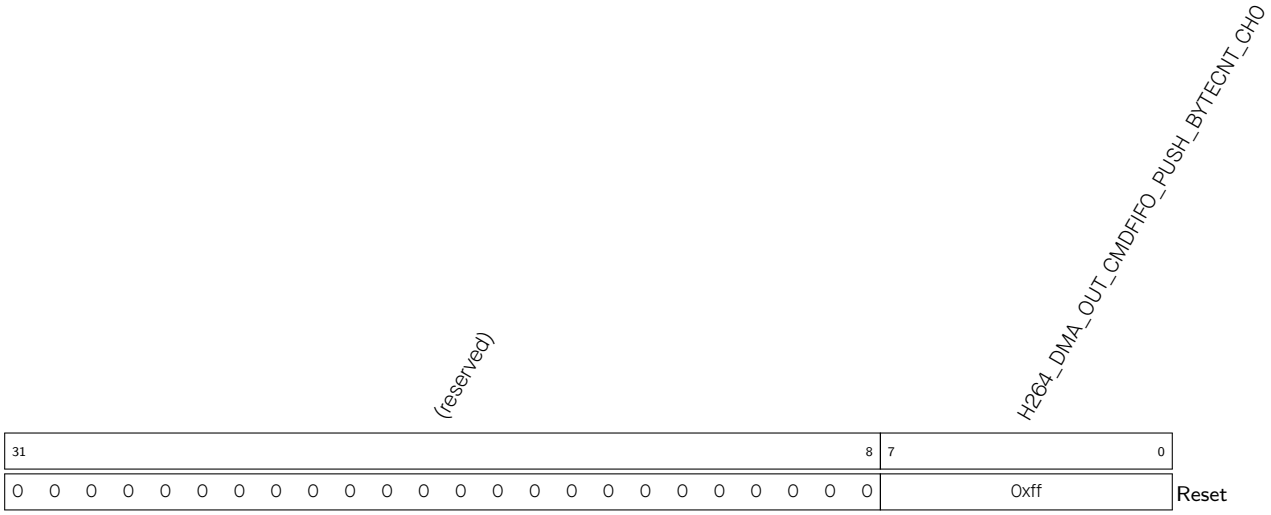
H264_DMA_OUT_CMDFIFO_BUF_LEN_HB_CH0 Represents the remaining request data. (RO)

Register 41.225. H264_DMA_OUT_FIFO_BCNT_CH0_REG (0x0074)



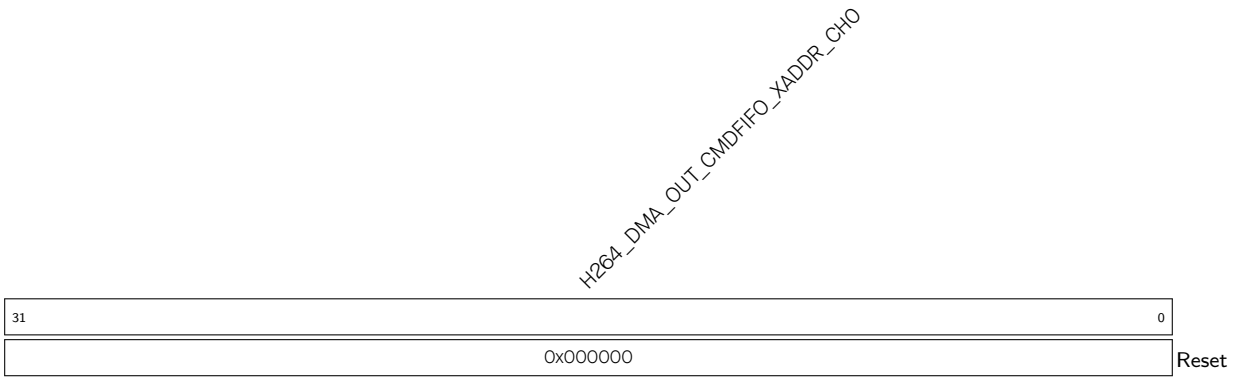
H264_DMA_OUT_CMDFIFO_OUTFIFO_BCNT_CH0 Represents that the data space has been used in FIFO. (RO)

Register 41.226. H264_DMA_OUT_PUSH_BYTECNT_CH0_REG (0x0078)



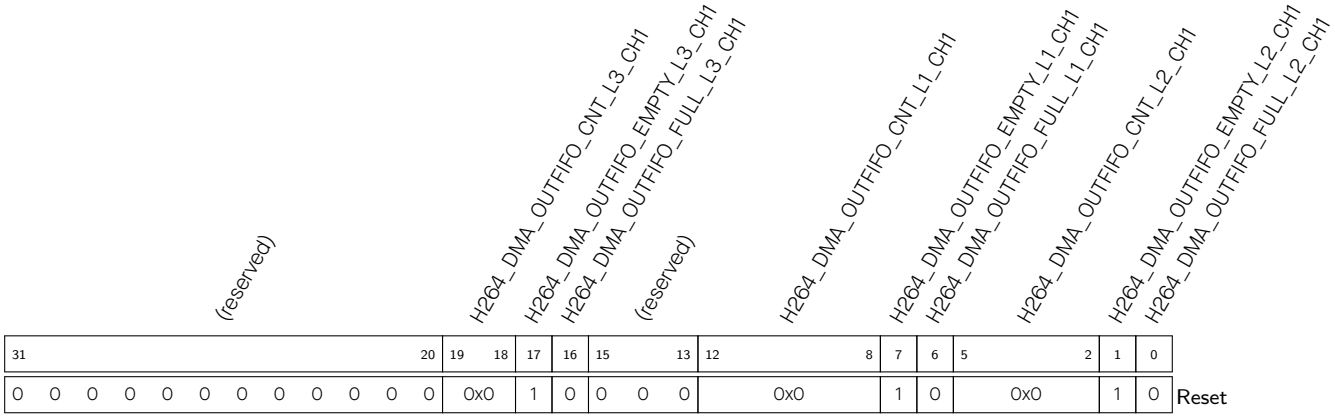
H264_DMA_OUT_CMDFIFO_PUSH_BYTECNT_CH0 Represents the data quantity in the current AXI request. (RO)

Register 41.227. H264_DMA_OUT_XADDR_CH0_REG (0x007C)



H264_DMA_OUT_CMDFIFO_XADDR_CH0 Represents the address of the current AXI request. (RO)

Register 41.228. H264_DMA_OUTFIFO_STATUS_CH1_REG (0x0114)



H264_DMA_OUTFIFO_FULL_L2_CH1 Represents whether l2 FIFO of TX channel 1 is full.

0: Not Full

1: Full

(RO)

H264_DMA_OUTFIFO_EMPTY_L2_CH1 Represents whether l2 FIFO of TX channel 1 is empty.

0: Not empty

1: Empty

(RO)

H264_DMA_OUTFIFO_CNT_L2_CH1 Represents the data quantity in l2 FIFO for TX channel 1. Measurement unit: 16 bytes. (RO)

H264_DMA_OUTFIFO_FULL_L1_CH1 Represents whether l1 FIFO of TX channel 1 is full.

0: Not Full

1: Full

(RO)

H264_DMA_OUTFIFO_EMPTY_L1_CH1 Represents whether l1 FIFO of TX channel 1 is empty.

0: Not empty

1: Empty

(RO)

H264_DMA_OUTFIFO_CNT_L1_CH1 Represents the data quantity in l1 FIFO for TX channel 1. Measurement unit: byte. (RO)

H264_DMA_OUTFIFO_FULL_L3_CH1 Represents whether l3 FIFO of TX channel 1 is full.

0: Not Full

1: Full

(RO)

H264_DMA_OUTFIFO_EMPTY_L3_CH1 Represents whether l3 FIFO of TX channel 1 is empty.

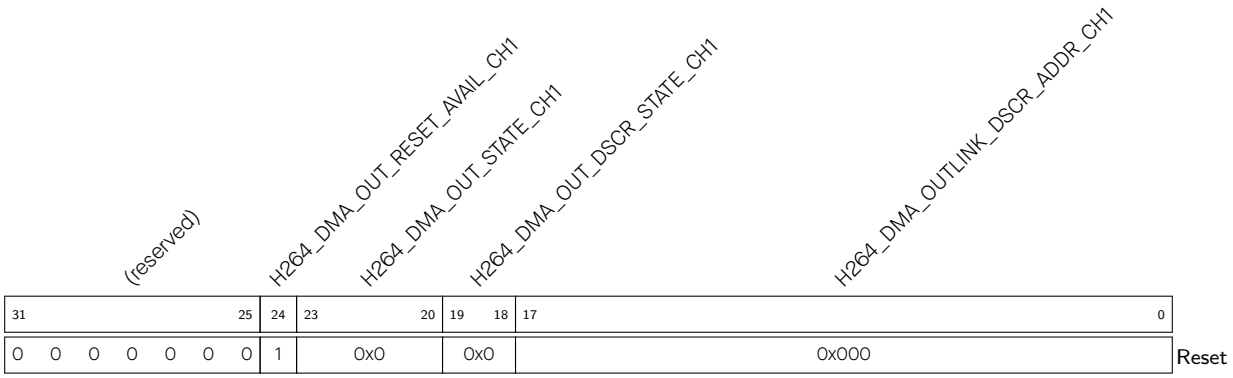
0: Not empty

1: Empty

(RO)

H264_DMA_OUTFIFO_CNT_L3_CH1 Represents the data quantity in l3 FIFO for TX channel 1. Measurement unit: 8 bytes. (RO)

Register 41.229. H264_DMA_OUT_STATE_CH1_REG (0x0124)



H264_DMA_OUTLINK_DSCR_ADDR_CH1 Represents the current outlink descriptor's address for TX channel 1. (RO)

H264_DMA_OUT_DSCR_STATE_CH1 Represents the current state of the descriptor state machine for TX channel 1. (RO)

H264_DMA_OUT_STATE_CH1 Represents the current control module state machine state for TX channel 1. (RO)

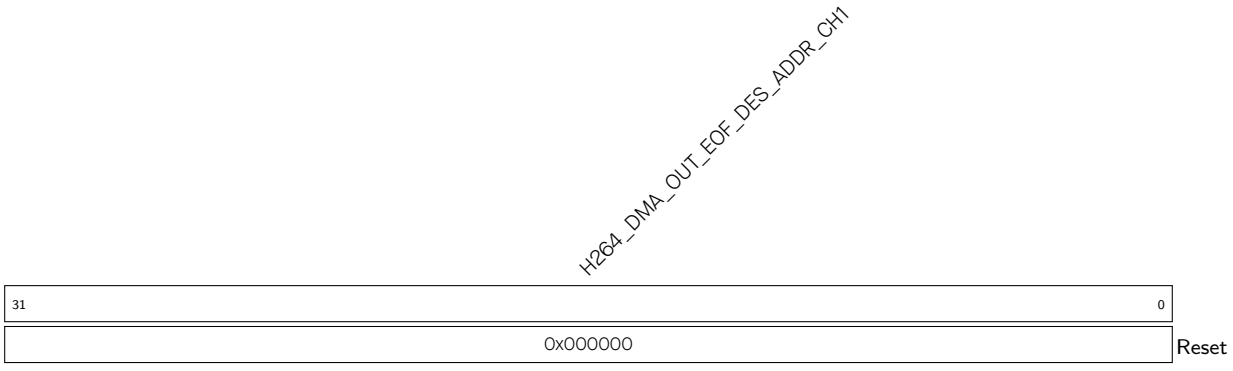
H264_DMA_OUT_RESET_AVAIL_CH1 Represents whether it is safe to reset the channel.

0: Unsafe

1: Safe

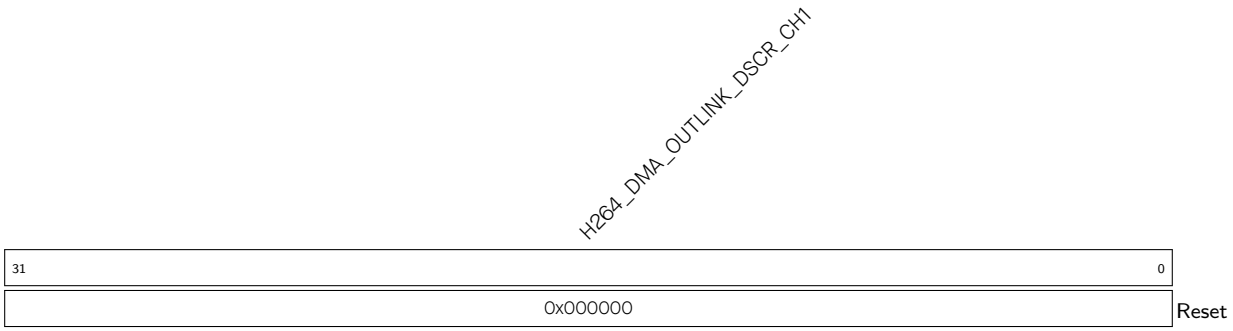
(RO)

Register 41.230. H264_DMA_OUT_EOF_DES_ADDR_CH1_REG (0x0128)



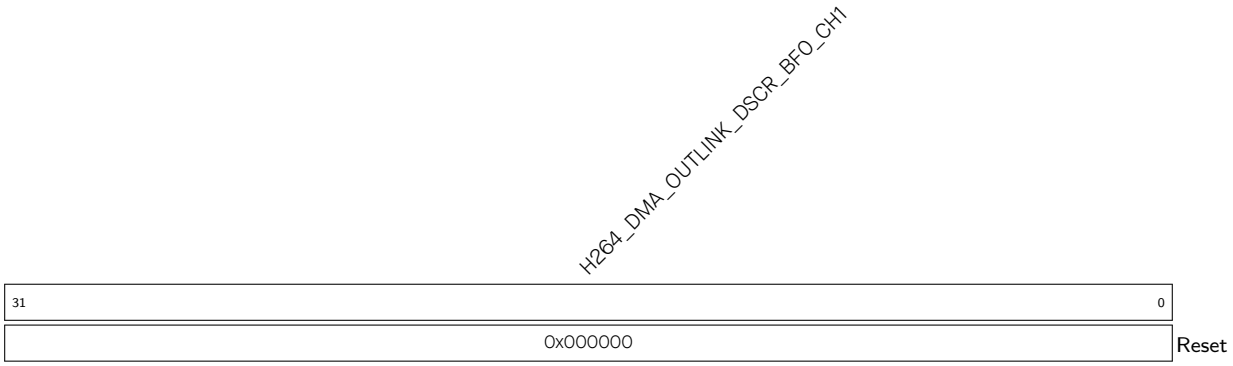
H264_DMA_OUT_EOF_DES_ADDR_CH1 Represents the address of the outlink descriptor when the EOF bit in this descriptor is 1. (RO)

Register 41.231. H264_DMA_OUT_DSCR_CH1_REG (0x012C)



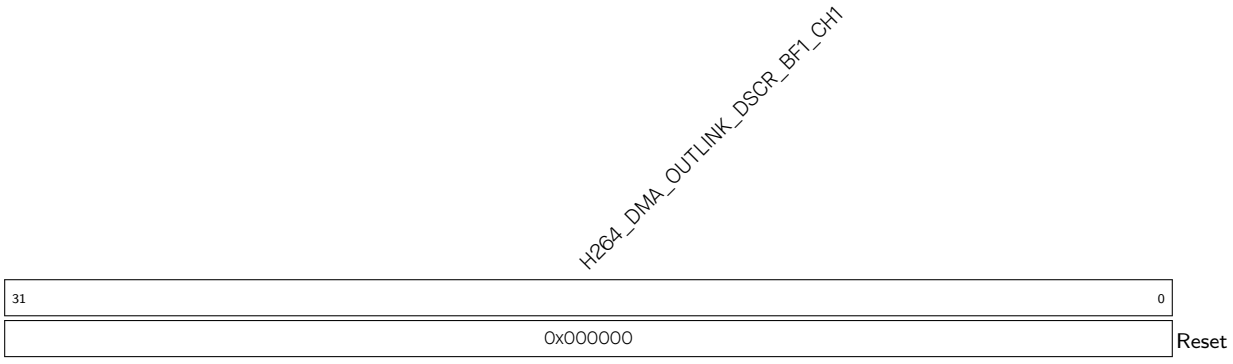
H264_DMA_OUTLINK_DSCR_CH1 Represents the address of the next descriptor y+1 pointed by the current descriptor that is pre-read. (RO)

Register 41.232. H264_DMA_OUT_DSCR_BF0_CH1_REG (0x0130)



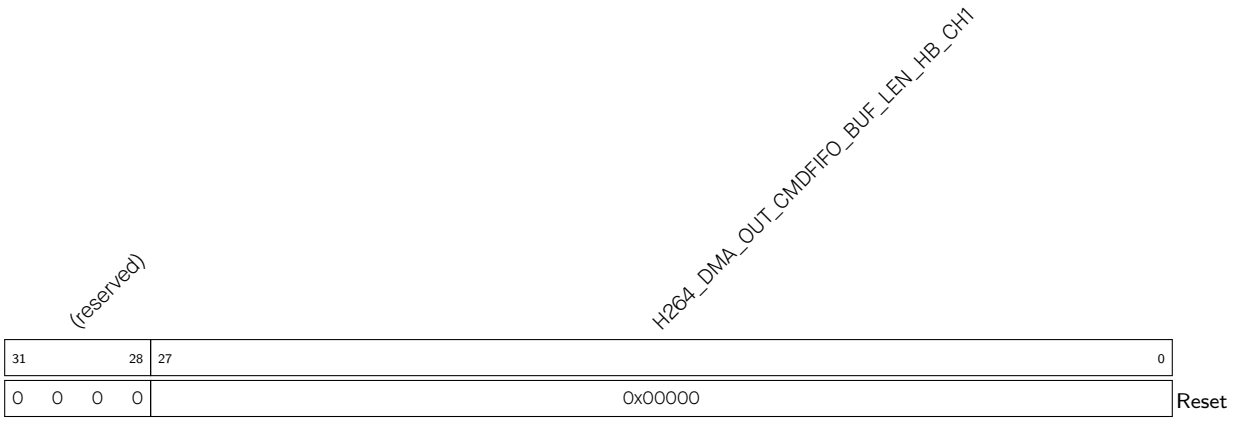
H264_DMA_OUTLINK_DSCR_BF0_CH1 Represents the address of the current descriptor y that is pre-read. (RO)

Register 41.233. H264_DMA_OUT_DSCR_BF1_CH1_REG (0x0134)



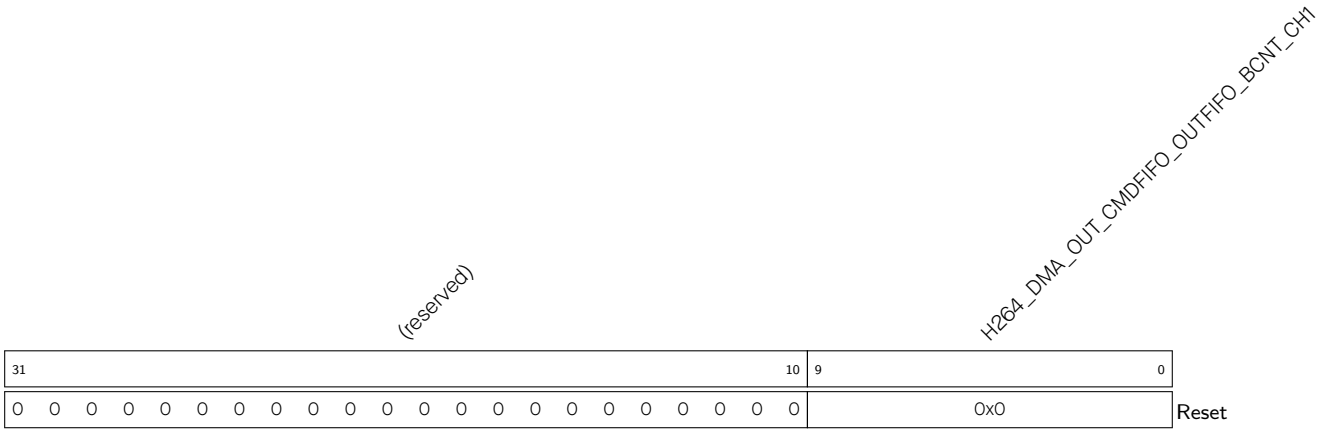
H264_DMA_OUTLINK_DSCR_BF1_CH1 Represents the address of the previous descriptor y-1 that is pre-read. (RO)

Register 41.234. H264_DMA_OUT_BUF_LEN_CH1_REG (0x0170)



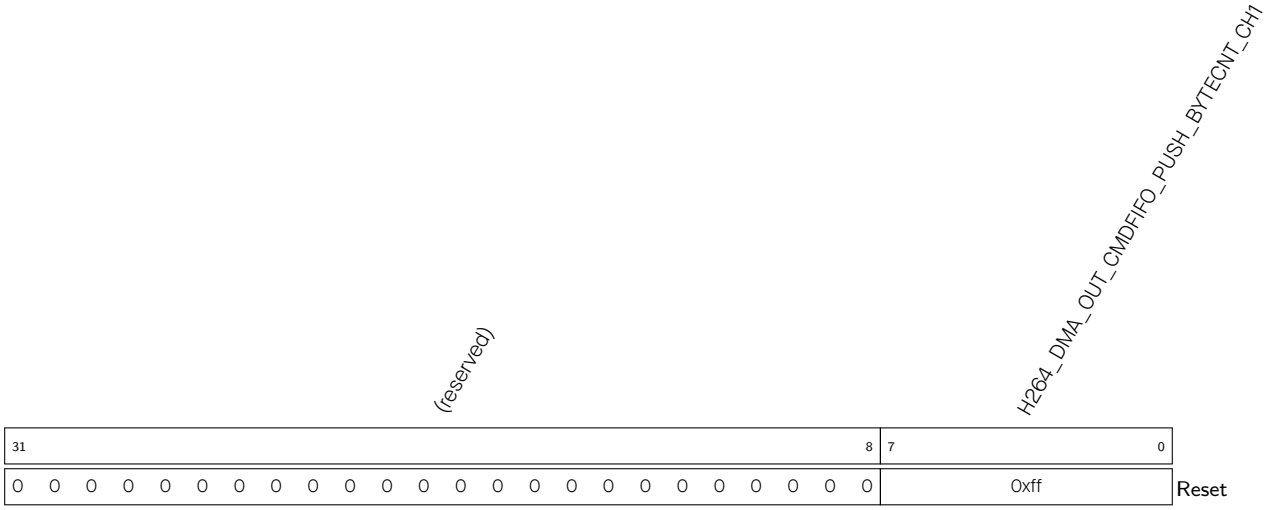
H264_DMA_OUT_CMDFIFO_BUF_LEN_HB_CH1 Represents the remaining request data. (RO)

Register 41.235. H264_DMA_OUT_FIFO_BCNT_CH1_REG (0x0174)



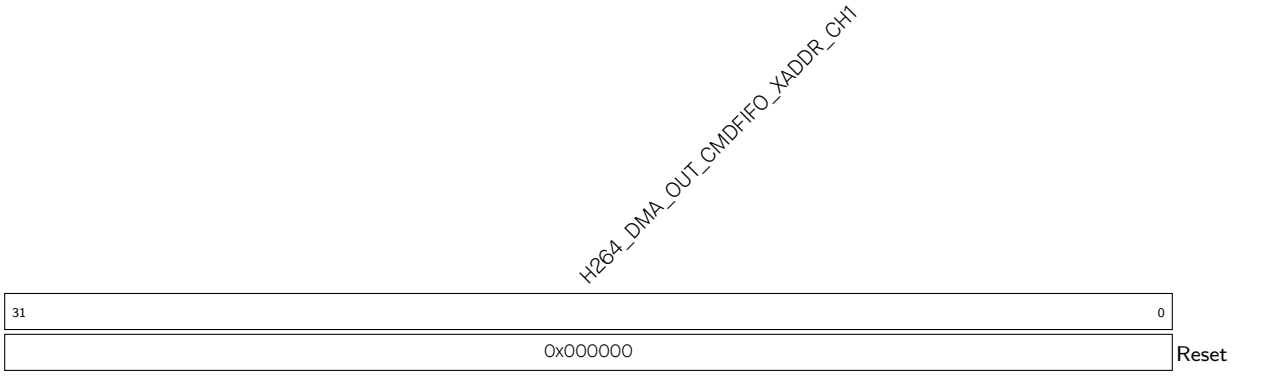
H264_DMA_OUT_CMDFIFO_OUTFIFO_BCNT_CH1 Represents that the data space has been used in FIFO. (RO)

Register 41.236. H264_DMA_OUT_PUSH_BYTECNT_CH1_REG (0x0178)



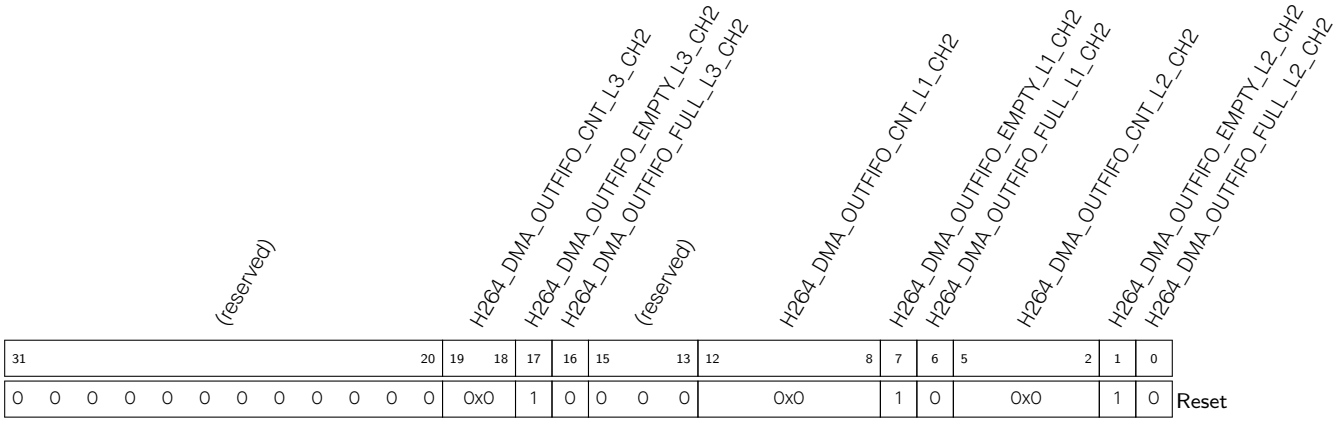
H264_DMA_OUT_CMDFIFO_PUSH_BYTECNT_CH1 Represents the data quantity in the current AXI request. (RO)

Register 41.237. H264_DMA_OUT_XADDR_CH1_REG (0x017C)



H264_DMA_OUT_CMDFIFO_XADDR_CH1 Represents the address of the current AXI request. (RO)

Register 41.238. H264_DMA_OUTFIFO_STATUS_CH2_REG (0x0214)



H264_DMA_OUTFIFO_FULL_L2_CH2 Represents whether l2 FIFO of TX channel 2 is full.

0: Not Full

1: Full

(RO)

H264_DMA_OUTFIFO_EMPTY_L2_CH2 Represents whether l2 FIFO of TX channel 2 is empty.

0: Not empty

1: Empty

(RO)

H264_DMA_OUTFIFO_CNT_L2_CH2 Represents the data quantity in l2 FIFO for TX channel 2. Measurement unit: 16 bytes. (RO)

H264_DMA_OUTFIFO_FULL_L1_CH2 Represents whether l1 FIFO of TX channel 2 is full.

0: Not Full

1: Full

(RO)

H264_DMA_OUTFIFO_EMPTY_L1_CH2 Represents whether l1 FIFO of TX channel 2 is empty.

0: Not empty

1: Empty

(RO)

H264_DMA_OUTFIFO_CNT_L1_CH2 Represents the data quantity in l1 FIFO for TX channel 2. Measurement unit: byte. (RO)

H264_DMA_OUTFIFO_FULL_L3_CH2 Represents whether l3 FIFO of TX channel 2 is full.

0: Not Full

1: Full

(RO)

H264_DMA_OUTFIFO_EMPTY_L3_CH2 Represents whether l3 FIFO of TX channel 2 is empty.

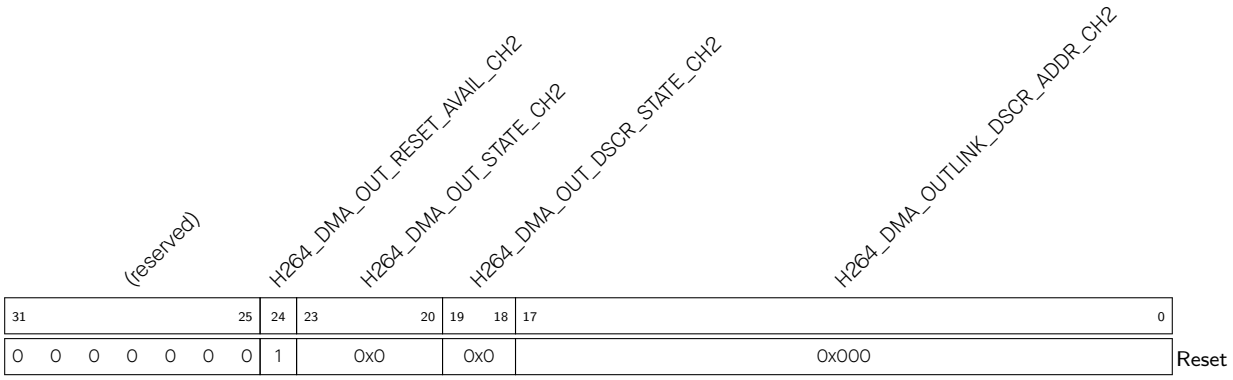
0: Not empty

1: Empty

(RO)

H264_DMA_OUTFIFO_CNT_L3_CH2 Represents the data quantity in l3 FIFO for TX channel 2. Measurement unit: 8 bytes. (RO)

Register 41.239. H264_DMA_OUT_STATE_CH2_REG (0x0224)



H264_DMA_OUTLINK_DSCR_ADDR_CH2 Represents the current outlink descriptor's address for TX channel 2. (RO)

H264_DMA_OUT_DSCR_STATE_CH2 Represents the current state of the descriptor state machine for TX channel 2. (RO)

H264_DMA_OUT_STATE_CH2 Represents the current control module state machine state for TX channel 2. (RO)

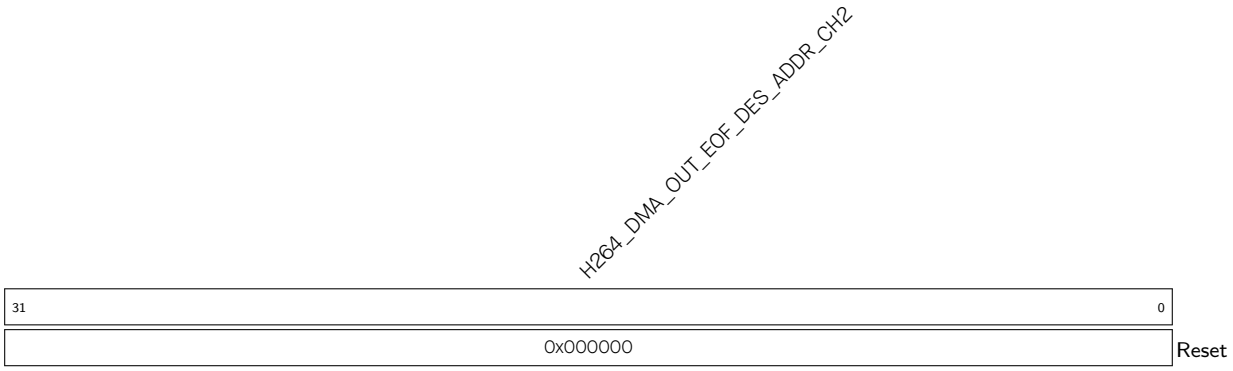
H264_DMA_OUT_RESET_AVAIL_CH2 Represents whether it is safe to reset the channel.

0: Unsafe

1: Safe

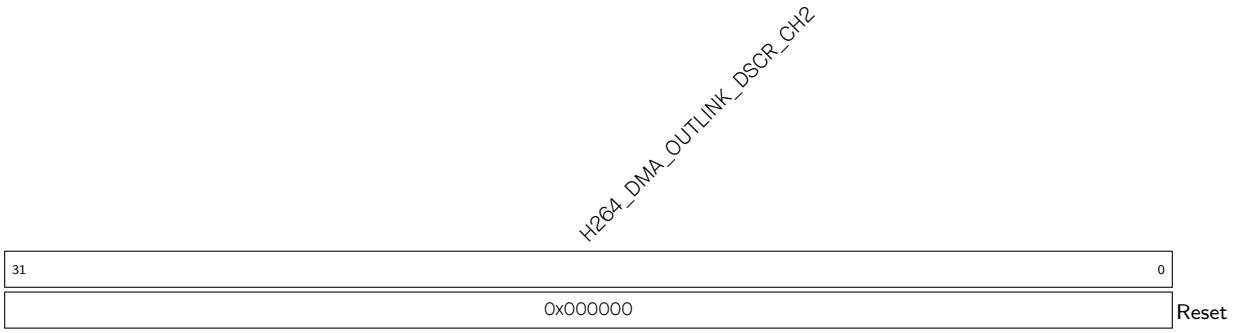
(RO)

Register 41.240. H264_DMA_OUT_EOF_DES_ADDR_CH2_REG (0x0228)



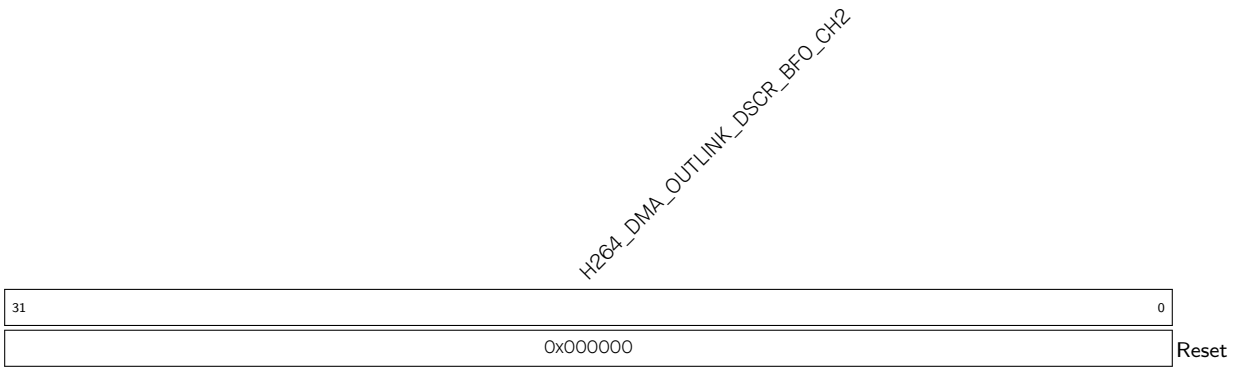
H264_DMA_OUT_EOF_DES_ADDR_CH2 Represents the address of the inlink descriptor when the EOF bit in this descriptor is 1. (RO)

Register 41.241. H264_DMA_OUT_DSCR_CH2_REG (0x022C)



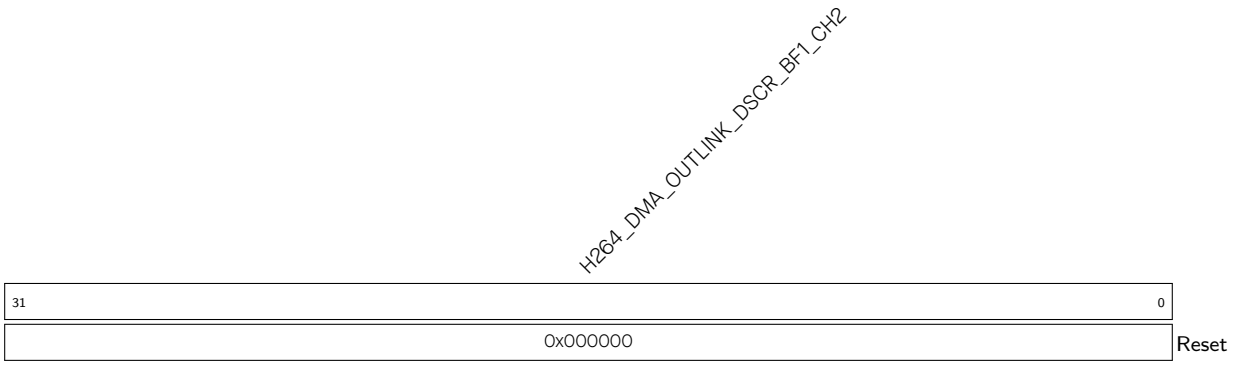
H264_DMA_OUTLINK_DSCR_CH2 Represents the address of the next descriptor y+1 pointed by the current descriptor that is pre-read. (RO)

Register 41.242. H264_DMA_OUT_DSCR_BF0_CH2_REG (0x0230)



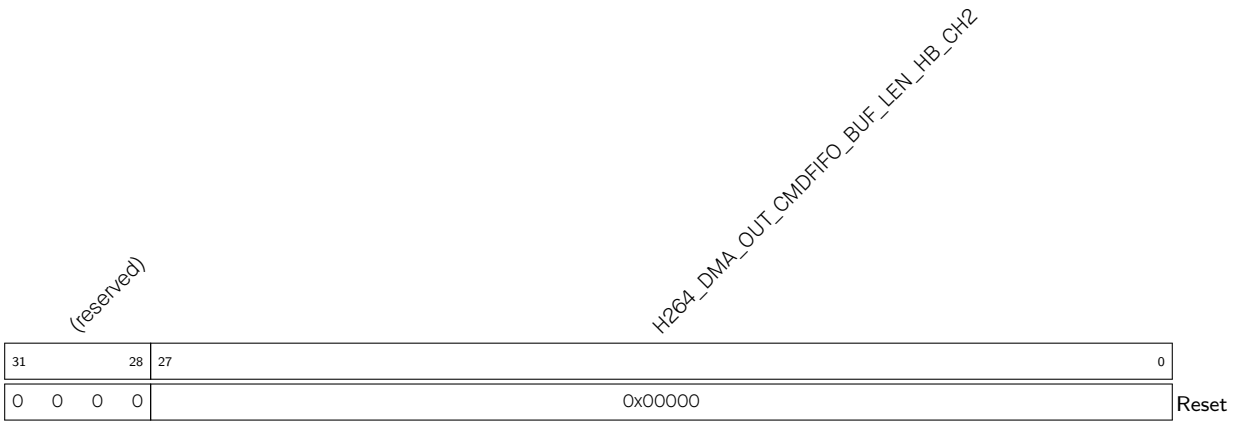
H264_DMA_OUTLINK_DSCR_BF0_CH2 Represents the address of the current descriptor y that is pre-read. (RO)

Register 41.243. H264_DMA_OUT_DSCR_BF1_CH2_REG (0x0234)



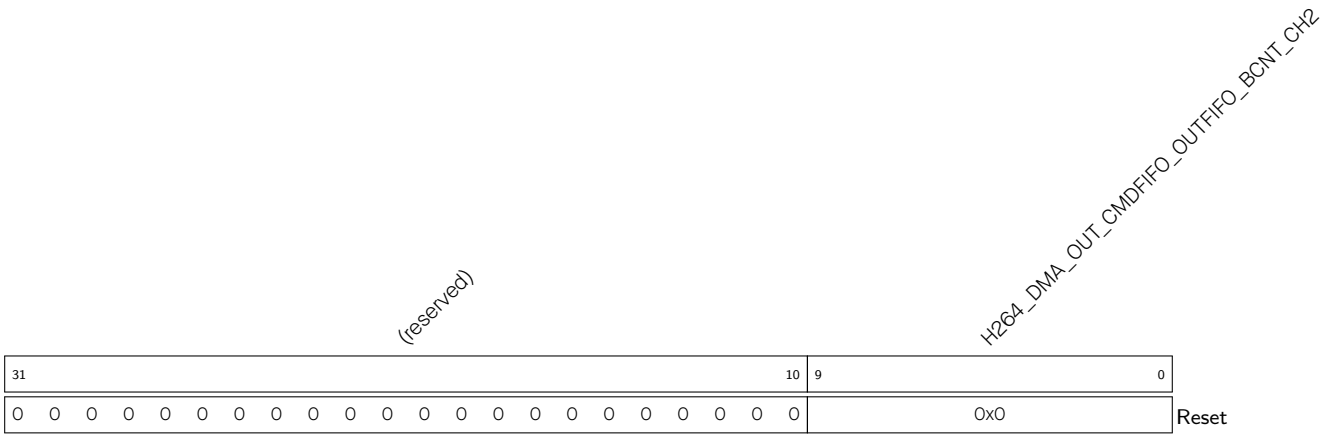
H264_DMA_OUTLINK_DSCR_BF1_CH2 Represents the address of the previous descriptor y-1 that is pre-read. (RO)

Register 41.244. H264_DMA_OUT_BUF_LEN_CH2_REG (0x0270)



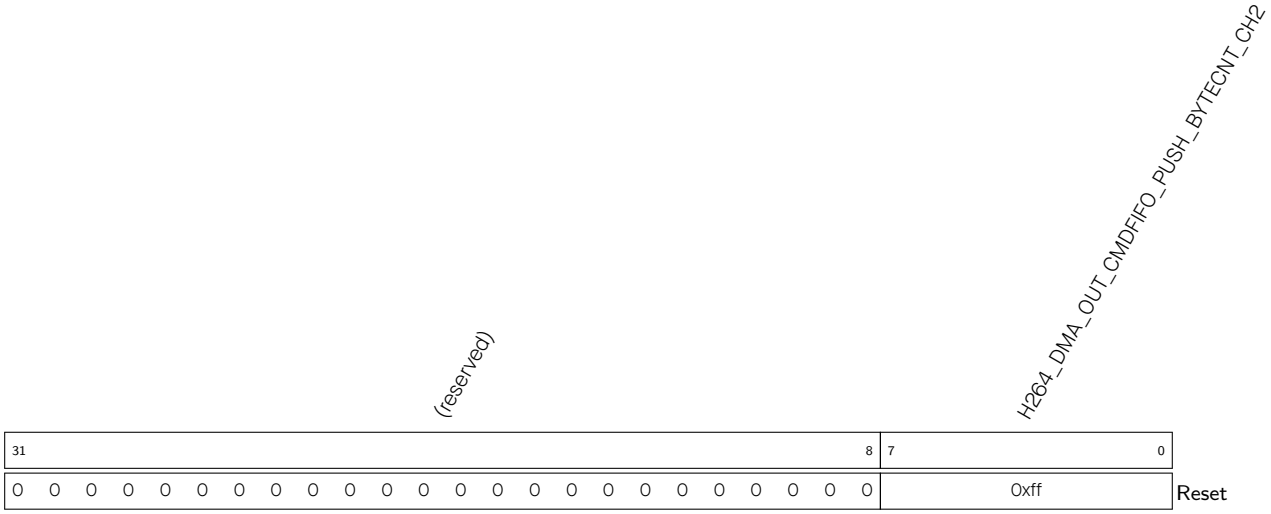
H264_DMA_OUT_CMDFIFO_BUF_LEN_HB_CH2 Represents the remaining request data. (RO)

Register 41.245. H264_DMA_OUT_FIFO_BCNT_CH2_REG (0x0274)



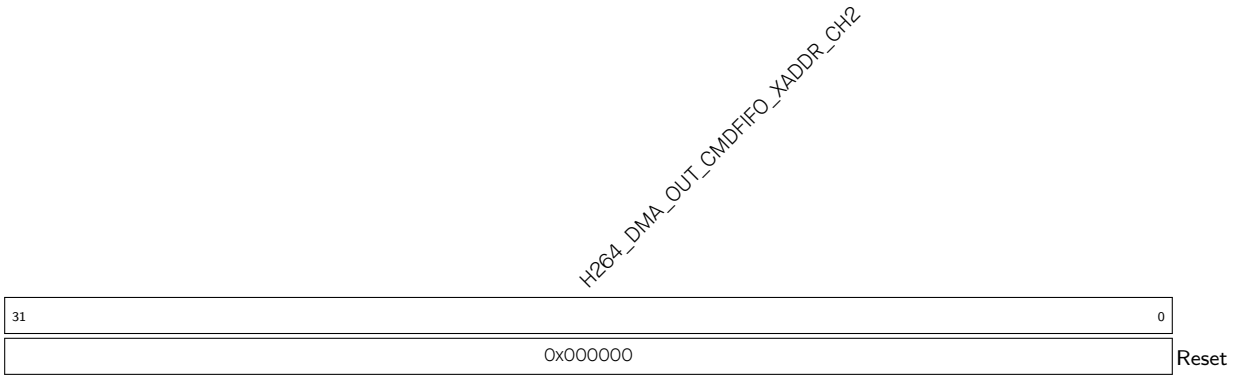
H264_DMA_OUT_CMDFIFO_OUTFIFO_BCNT_CH2 Represents that the data space has been used in FIFO. (RO)

Register 41.246. H264_DMA_OUT_PUSH_BYTECNT_CH2_REG (0x0278)



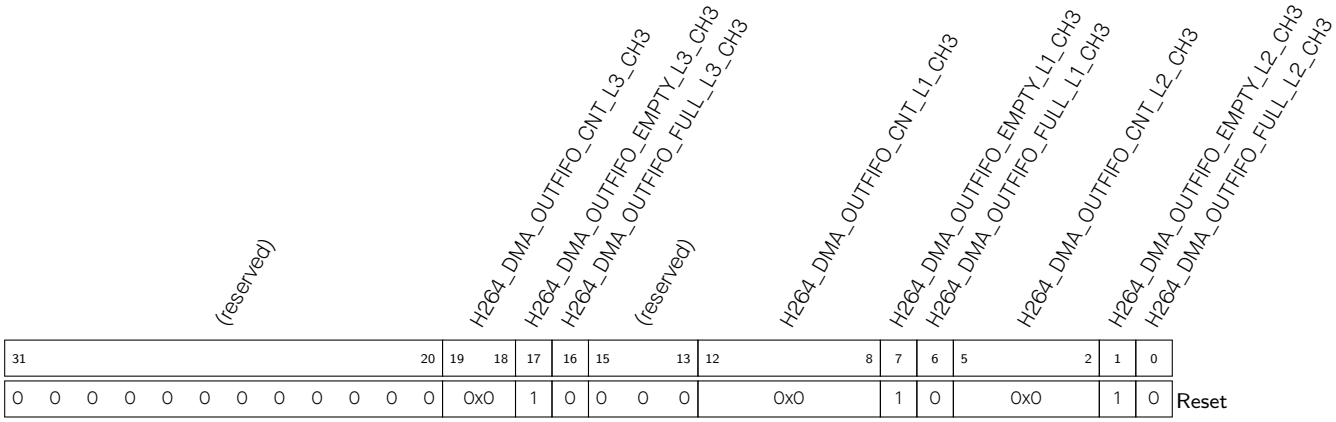
H264_DMA_OUT_CMDFIFO_PUSH_BYTECNT_CH2 Represents the data quantity in the current AXI request. (RO)

Register 41.247. H264_DMA_OUT_XADDR_CH2_REG (0x027C)



H264_DMA_OUT_CMDFIFO_XADDR_CH2 Represents the address of the current AXI request. (RO)

Register 41.248. H264_DMA_OUTFIFO_STATUS_CH3_REG (0x0314)



H264_DMA_OUTFIFO_FULL_L2_CH3 Represents whether l2 FIFO of TX channel 3 is full.

0: Not Full

1: Full

(RO)

H264_DMA_OUTFIFO_EMPTY_L2_CH3 Represents whether l2 FIFO of TX channel 3 is empty.

0: Not empty

1: Empty

(RO)

H264_DMA_OUTFIFO_CNT_L2_CH3 Represents the data quantity in l2 FIFO for TX channel 3. Measurement unit: 16 bytes. (RO)

H264_DMA_OUTFIFO_FULL_L1_CH3 Represents whether l1 FIFO of TX channel 3 is full.

0: Not Full

1: Full

(RO)

H264_DMA_OUTFIFO_EMPTY_L1_CH3 Represents whether l1 FIFO of TX channel 3 is empty.

0: Not empty

1: Empty

(RO)

H264_DMA_OUTFIFO_CNT_L1_CH3 Represents the data quantity in l1 FIFO for TX channel 3. Measurement unit: byte. (RO)

H264_DMA_OUTFIFO_FULL_L3_CH3 Represents whether l3 FIFO of TX channel 3 is full.

0: Not Full

1: Full

(RO)

H264_DMA_OUTFIFO_EMPTY_L3_CH3 Represents whether l3 FIFO of TX channel 3 is empty.

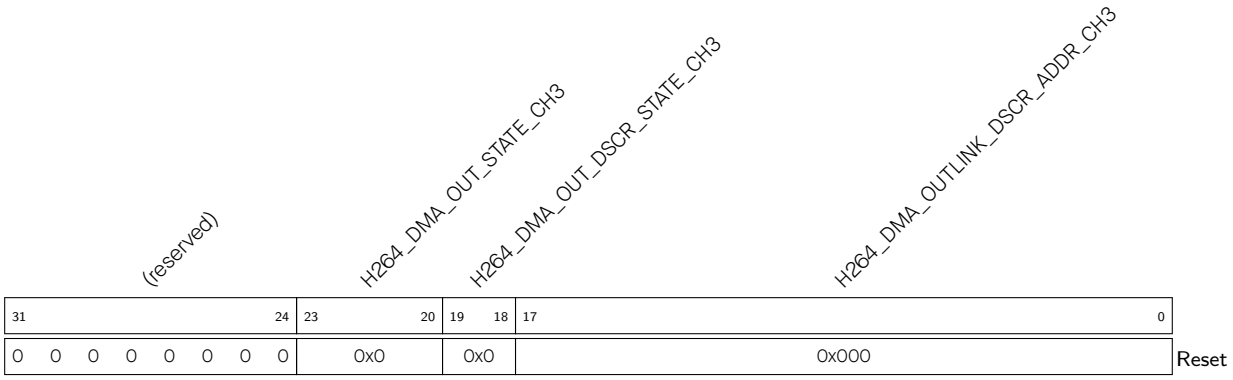
0: Not empty

1: Empty

(RO)

H264_DMA_OUTFIFO_CNT_L3_CH3 Represents the data quantity in l3 FIFO for TX channel 3. Measurement unit: 8 bytes. (RO)

Register 41.249. H264_DMA_OUT_STATE_CH3_REG (0x0324)

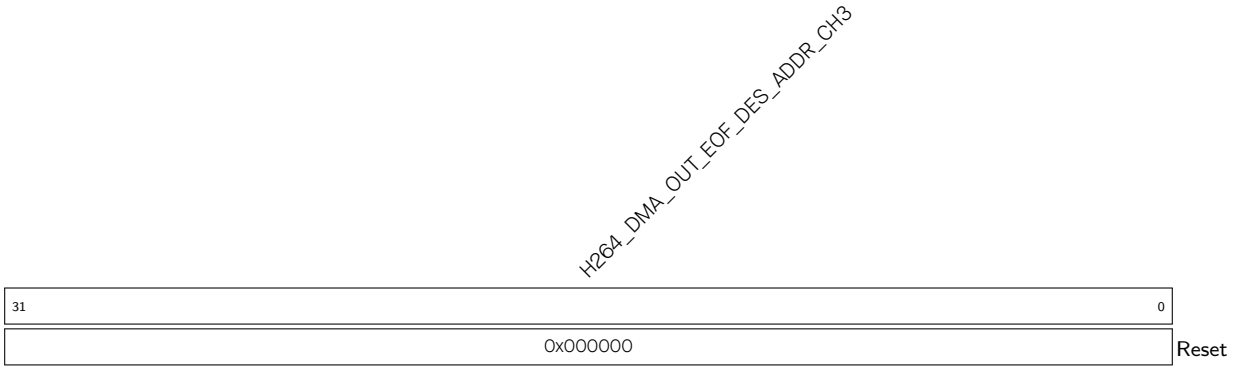


H264_DMA_OUTLINK_DSCR_ADDR_CH3 Represents the current outlink descriptor's address for TX channel 3. (RO)

H264_DMA_OUT_DSCR_STATE_CH3 Represents the current state of the descriptor state machine for TX channel 3. (RO)

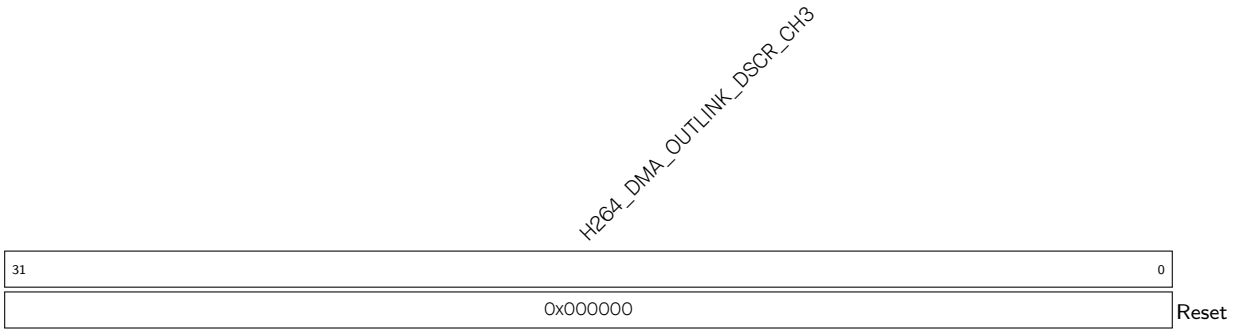
H264_DMA_OUT_STATE_CH3 Represents the current control module state machine state for TX channel 3. (RO)

Register 41.250. H264_DMA_OUT_EOF_DES_ADDR_CH3_REG (0x0328)



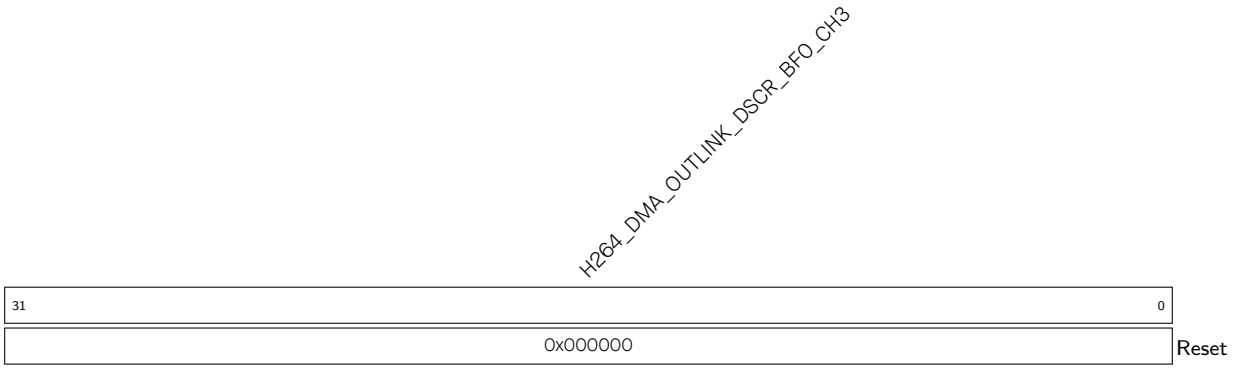
H264_DMA_OUT_EOF_DES_ADDR_CH3 Represents the address of the inlink descriptor when the EOF bit in this descriptor is 1. (RO)

Register 41.251. H264_DMA_OUT_DSCR_CH3_REG (0x032C)



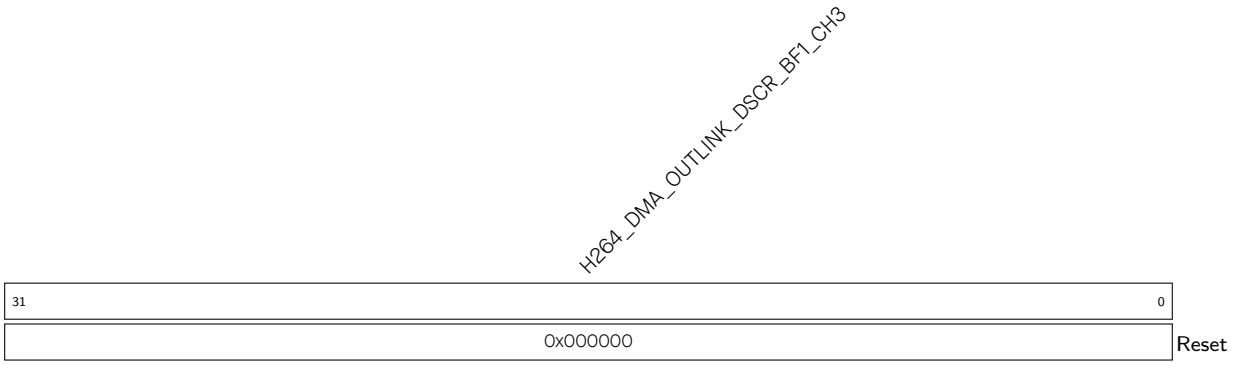
H264_DMA_OUTLINK_DSCR_CH3 Represents the address of the next descriptor y+1 pointed by the current descriptor that is pre-read. (RO)

Register 41.252. H264_DMA_OUT_DSCR_BF0_CH3_REG (0x0330)



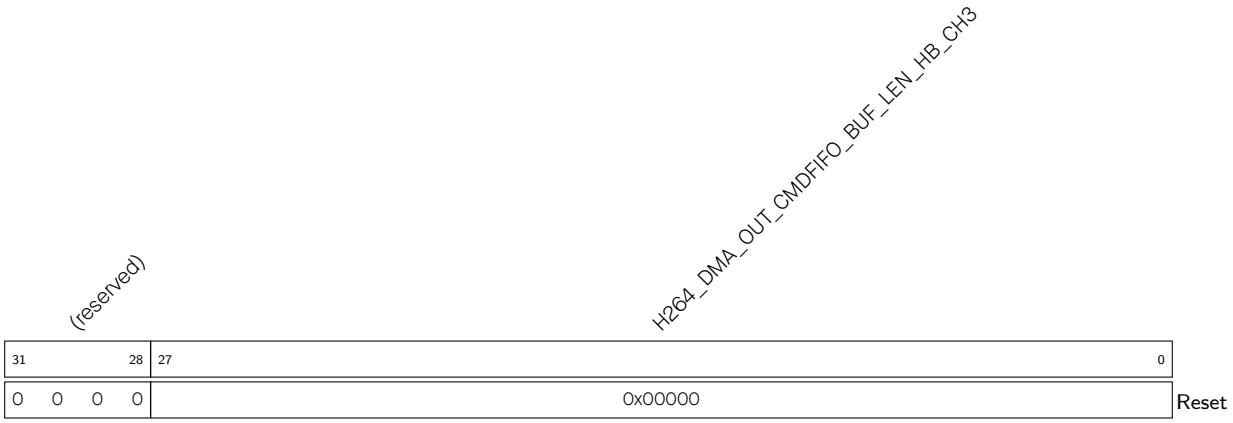
H264_DMA_OUTLINK_DSCR_BF0_CH3 Represents the address of the current descriptor y that is pre-read. (RO)

Register 41.253. H264_DMA_OUT_DSCR_BF1_CH3_REG (0x0334)



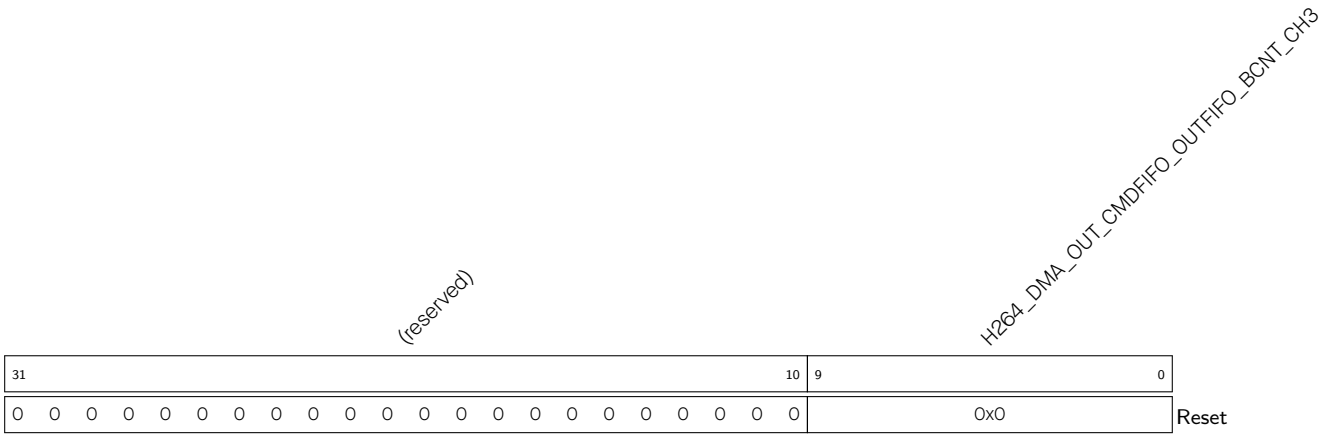
H264_DMA_OUTLINK_DSCR_BF1_CH3 Represents the address of the previous descriptor y-1 that is pre-read. (RO)

Register 41.254. H264_DMA_OUT_BUF_LEN_CH3_REG (0x0370)



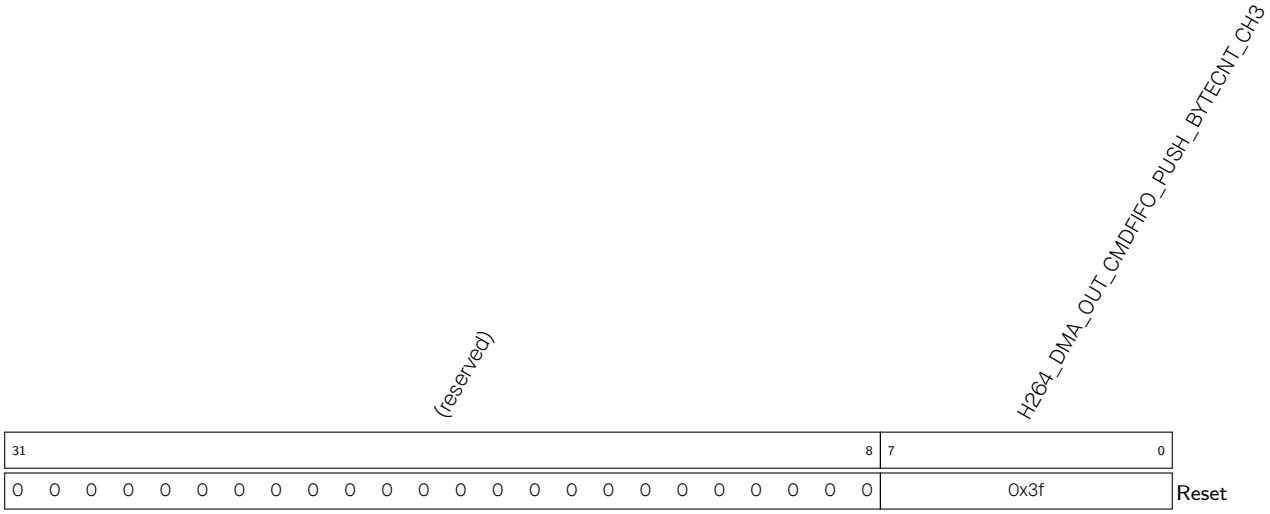
H264_DMA_OUT_CMDFIFO_BUF_LEN_HB_CH3 Represents the remaining request data. (RO)

Register 41.255. H264_DMA_OUT_FIFO_BCNT_CH3_REG (0x0374)



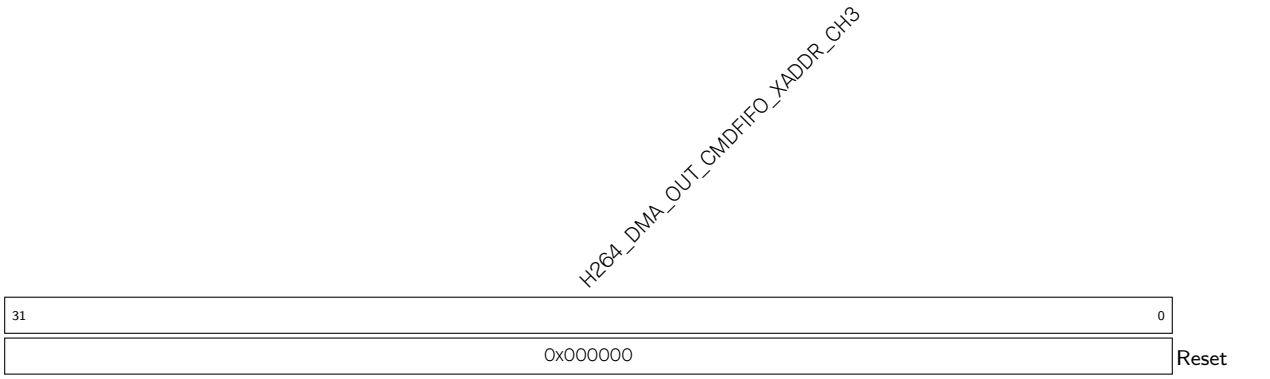
H264_DMA_OUT_CMDFIFO_OUTFIFO_BCNT_CH3 Represents that the data space has been used in FIFO. (RO)

Register 41.256. H264_DMA_OUT_PUSH_BYTECNT_CH3_REG (0x0378)



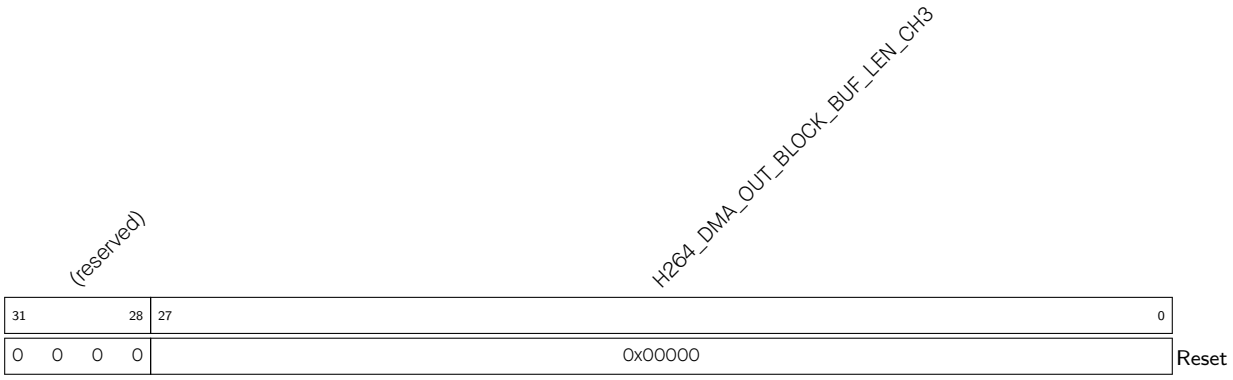
H264_DMA_OUT_CMDFIFO_PUSH_BYTECNT_CH3 Represents the data quantity in the current AXI request. (RO)

Register 41.257. H264_DMA_OUT_XADDR_CH3_REG (0x037C)



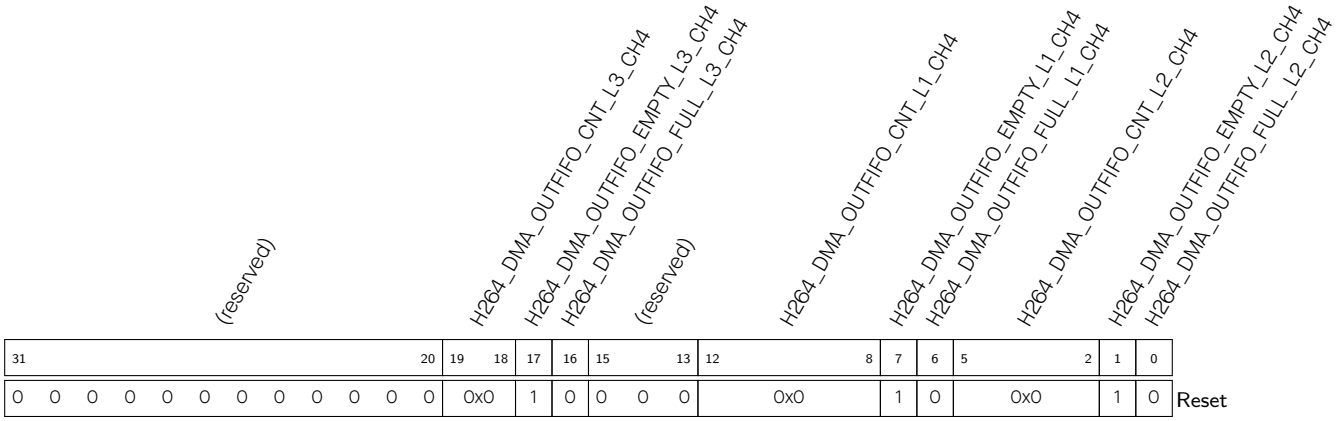
H264_DMA_OUT_CMDFIFO_XADDR_CH3 Represents the address of the current AXI request. (RO)

Register 41.258. H264_DMA_OUT_BLOCK_BUF_LEN_CH3_REG (0x0380)



H264_DMA_OUT_BLOCK_BUF_LEN_CH3 Represents the remaining block request data. (RO)

Register 41.259. H264_DMA_OUTFIFO_STATUS_CH4_REG (0x0414)



H264_DMA_OUTFIFO_FULL_L2_CH4 Represents whether l2 FIFO of TX channel 4 is full.

0: Not Full

1: Full

(RO)

H264_DMA_OUTFIFO_EMPTY_L2_CH4 Represents whether l2 FIFO of TX channel 4 is empty.

0: Not empty

1: Empty

(RO)

H264_DMA_OUTFIFO_CNT_L2_CH4 Represents the data quantity in l2 FIFO for TX channel 4. Measurement unit: 16 bytes. (RO)

H264_DMA_OUTFIFO_FULL_L1_CH4 Represents whether l1 FIFO of TX channel 4 is full.

0: Not Full

1: Full

(RO)

H264_DMA_OUTFIFO_EMPTY_L1_CH4 Represents whether l1 FIFO of TX channel 4 is empty.

0: Not empty

1: Empty

(RO)

H264_DMA_OUTFIFO_CNT_L1_CH4 Represents the data quantity in l1 FIFO for TX channel 4. Measurement unit: byte. (RO)

H264_DMA_OUTFIFO_FULL_L3_CH4 Represents whether l3 FIFO of TX channel 4 is full.

0: Not Full

1: Full

(RO)

H264_DMA_OUTFIFO_EMPTY_L3_CH4 Represents whether l3 FIFO of TX channel 4 is empty.

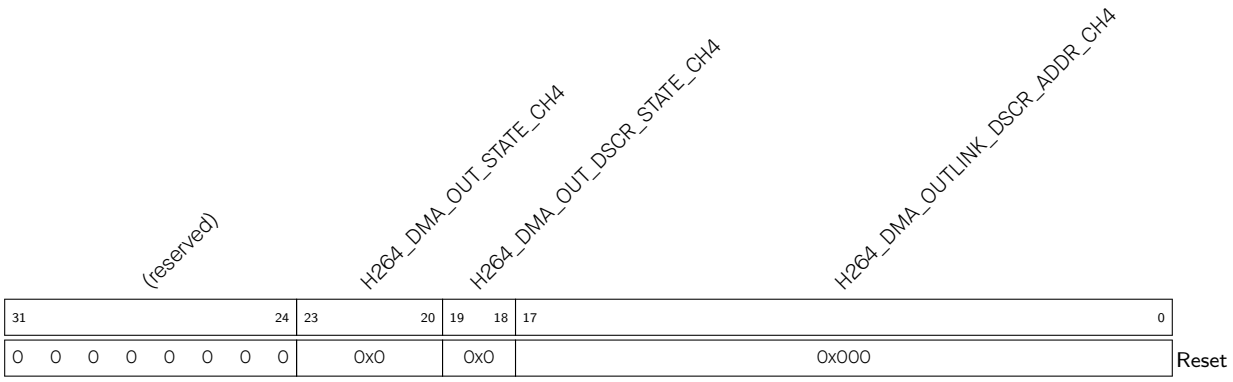
0: Not empty

1: Empty

(RO)

H264_DMA_OUTFIFO_CNT_L3_CH4 Represents the data quantity in l3 FIFO for TX channel 4. Measurement unit: 8 bytes. (RO)

Register 41.260. H264_DMA_OUT_STATE_CH4_REG (0x0424)

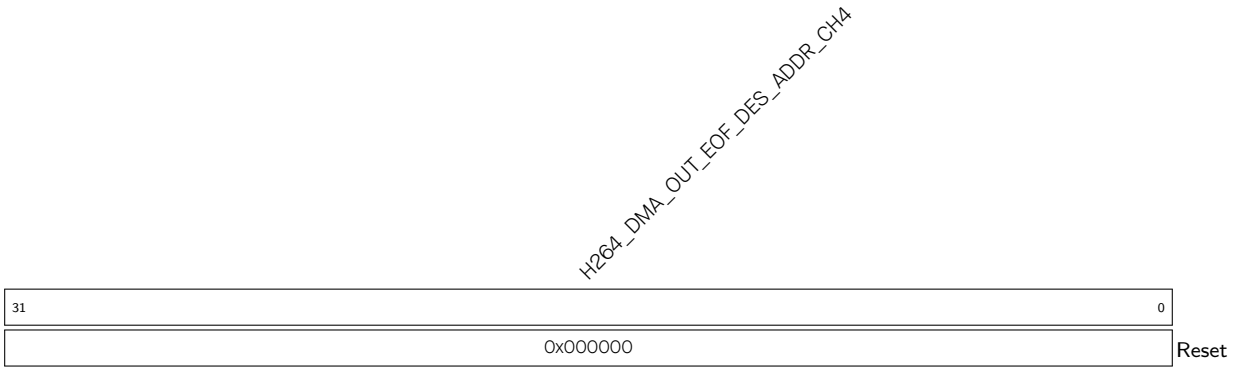


H264_DMA_OUTLINK_DSCR_ADDR_CH4 Represents the current outlink descriptor's address for TX channel 4. (RO)

H264_DMA_OUT_DSCR_STATE_CH4 Represents the current state of the descriptor state machine for TX channel 4. (RO)

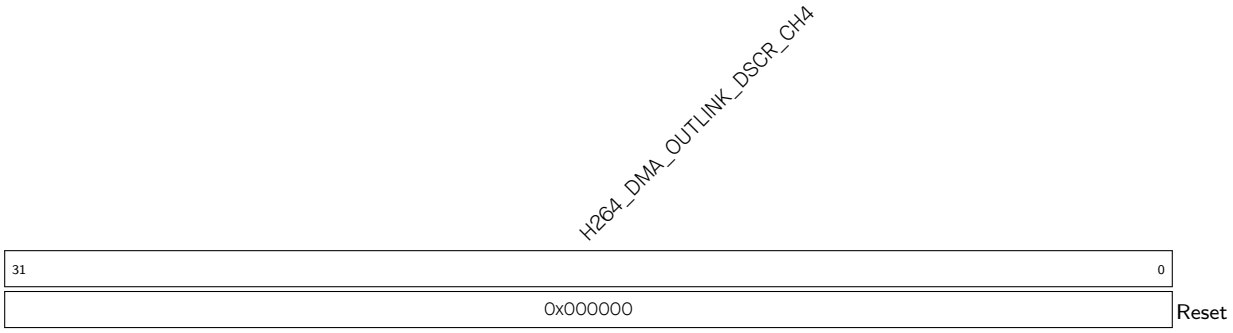
H264_DMA_OUT_STATE_CH4 Represents the current control module state machine state for TX channel 4. (RO)

Register 41.261. H264_DMA_OUT_EOF_DES_ADDR_CH4_REG (0x0428)



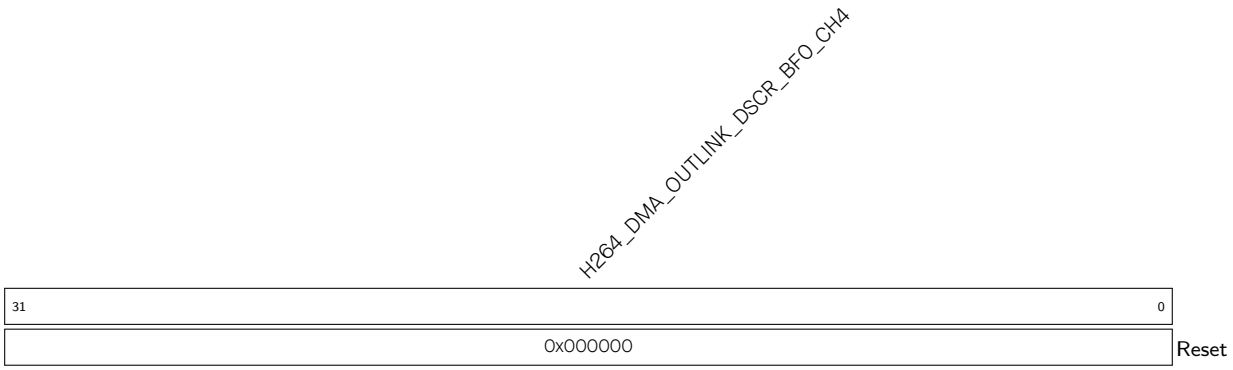
H264_DMA_OUT_EOF_DES_ADDR_CH4 Represents the address of the inlink descriptor when the EOF bit in this descriptor is 1. (RO)

Register 41.262. H264_DMA_OUT_DSCR_CH4_REG (0x042C)



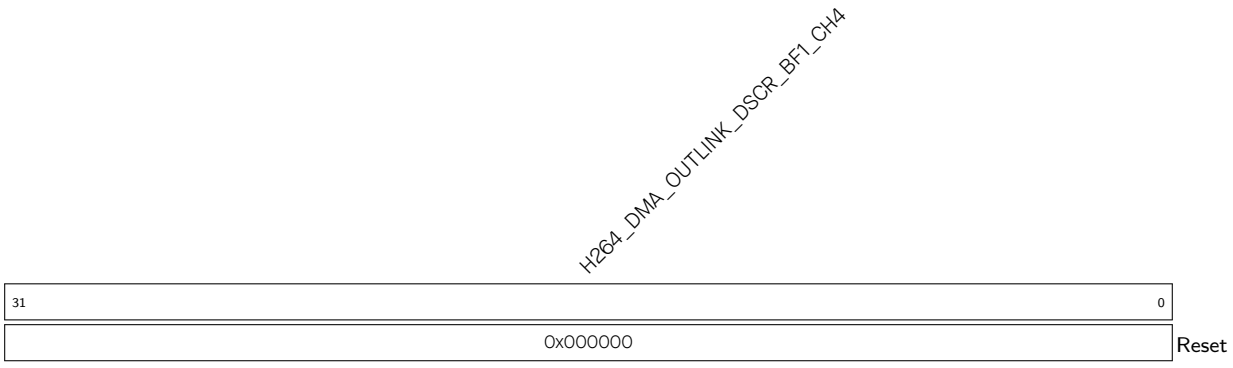
H264_DMA_OUTLINK_DSCR_CH4 Represents the address of the next descriptor y+1 pointed by the current descriptor that is pre-read. (RO)

Register 41.263. H264_DMA_OUT_DSCR_BF0_CH4_REG (0x0430)



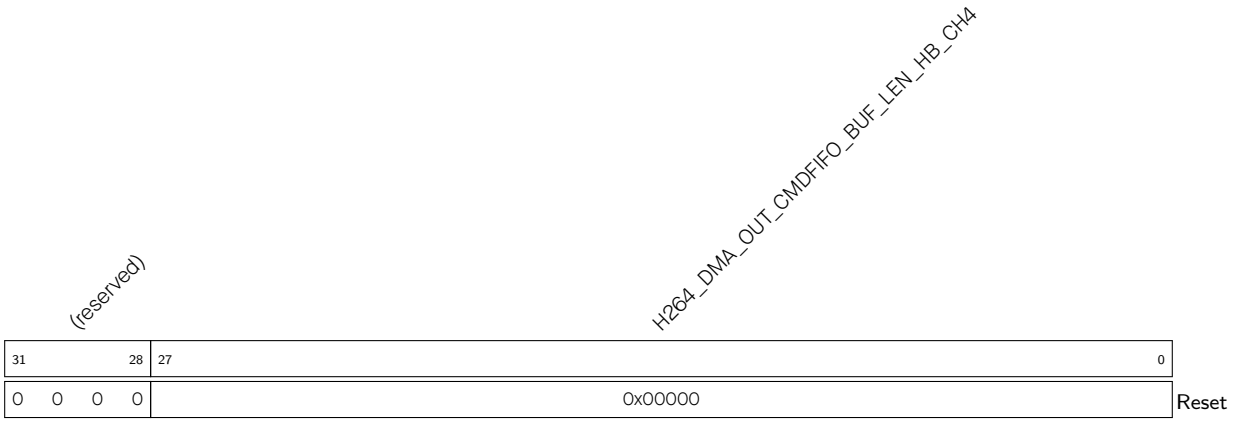
H264_DMA_OUTLINK_DSCR_BF0_CH4 Represents the address of the current descriptor y that is pre-read. (RO)

Register 41.264. H264_DMA_OUT_DSCR_BF1_CH4_REG (0x0434)



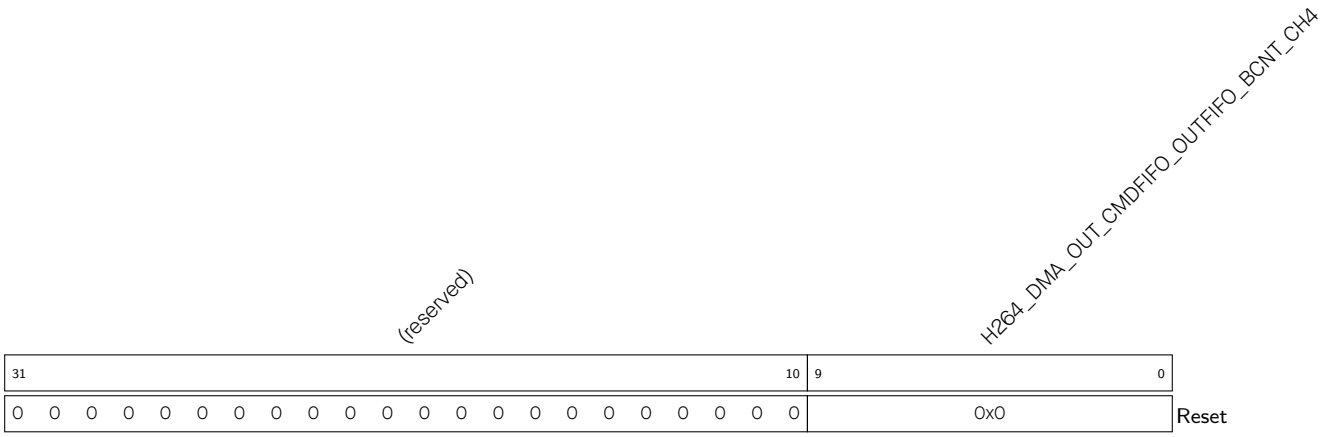
H264_DMA_OUTLINK_DSCR_BF1_CH4 Represents the address of the previous descriptor y-1 that is pre-read. (RO)

Register 41.265. H264_DMA_OUT_BUF_LEN_CH4_REG (0x0470)



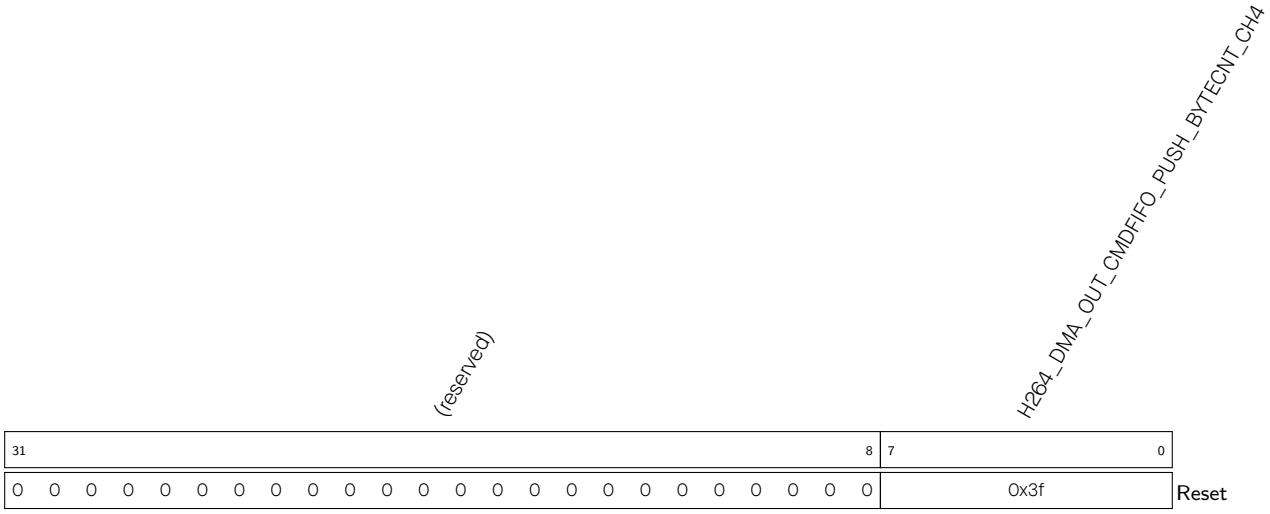
H264_DMA_OUT_CMDFIFO_BUF_LEN_HB_CH4 Represents the remaining request data. (RO)

Register 41.266. H264_DMA_OUT_FIFO_BCNT_CH4_REG (0x0474)



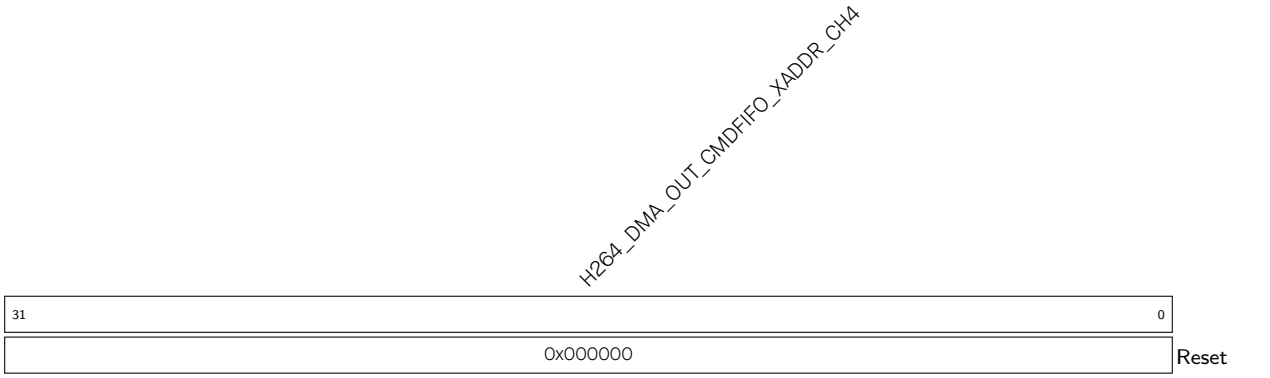
H264_DMA_OUT_CMDFIFO_OUTFIFO_BCNT_CH4 Represents that the data space has been used in FIFO. (RO)

Register 41.267. H264_DMA_OUT_PUSH_BYTECNT_CH4_REG (0x0478)



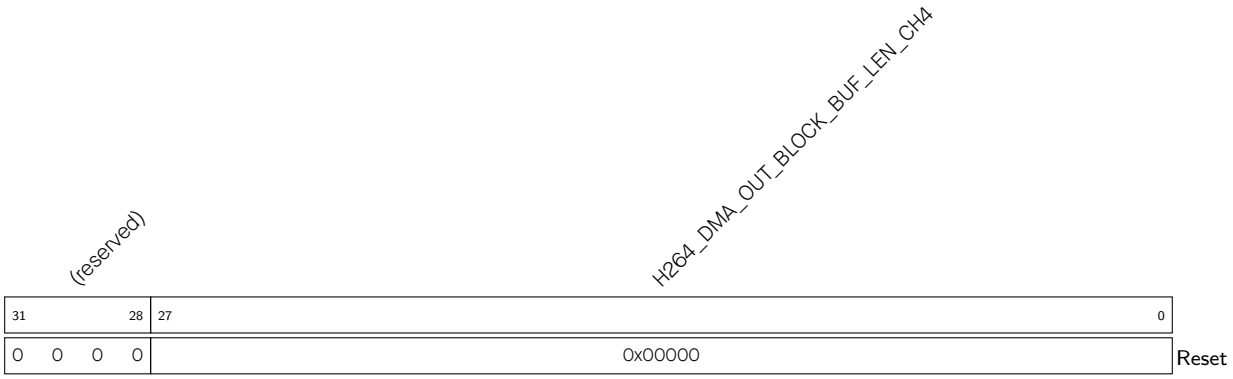
H264_DMA_OUT_CMDFIFO_PUSH_BYTECNT_CH4 Represents the data quantity in the current AXI request. (RO)

Register 41.268. H264_DMA_OUT_XADDR_CH4_REG (0x047C)



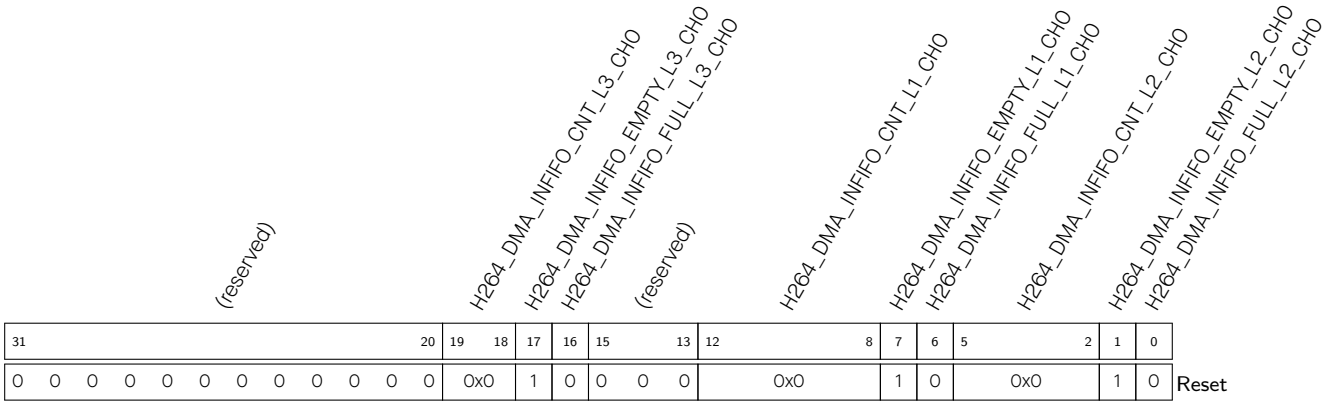
H264_DMA_OUT_CMDFIFO_XADDR_CH4 Represents the address of the current AXI request. (RO)

Register 41.269. H264_DMA_OUT_BLOCK_BUF_LEN_CH4_REG (0x0480)



H264_DMA_OUT_BLOCK_BUF_LEN_CH4 Represents the remaining block request data. (RO)

Register 41.270. H264_DMA_INFIFO_STATUS_CH0_REG (0x0514)



H264_DMA_INFIFO_FULL_L2_CH0 Represents whether l2 FIFO of RX channel 0 is full.

0: Not Full

1: Full

(RO)

H264_DMA_INFIFO_EMPTY_L2_CH0 Represents whether l2 FIFO of RX channel 0 is empty.

0: Not empty

1: Empty

(RO)

H264_DMA_INFIFO_CNT_L2_CH0 Represents the data quantity in l2 FIFO for RX channel 0. Measurement unit: 16 bytes. (RO)

H264_DMA_INFIFO_FULL_L1_CH0 Represents whether l1 FIFO of RX channel 0 is full.

0: Not Full

1: Full

(RO)

H264_DMA_INFIFO_EMPTY_L1_CH0 Represents whether l1 FIFO of RX channel 0 is empty.

0: Not empty

1: Empty

(RO)

H264_DMA_INFIFO_CNT_L1_CH0 Represents the data quantity in l1 FIFO for RX channel 0. Measurement unit: byte. (RO)

H264_DMA_INFIFO_FULL_L3_CH0 Represents whether l3 FIFO of RX channel 0 is full.

0: Not Full

1: Full

(RO)

H264_DMA_INFIFO_EMPTY_L3_CH0 Represents whether l3 FIFO of RX channel 0 is empty.

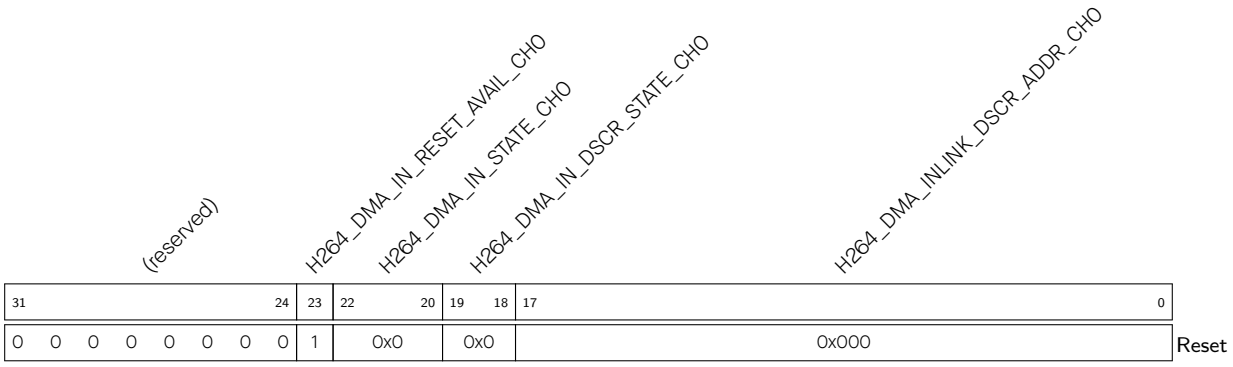
0: Not empty

1: Empty

(RO)

H264_DMA_INFIFO_CNT_L3_CH0 Represents the data quantity in l3 FIFO for RX channel 0. Measurement unit: 8 bytes. (RO)

Register 41.271. H264_DMA_IN_STATE_CH0_REG (0x0524)



H264_DMA_INLINK_DSCR_ADDR_CH0 Represents the current inlink descriptor's address for RX channel 0. (RO)

H264_DMA_IN_DSCR_STATE_CH0 Represents the current state of the descriptor state machine for RX channel 0. (RO)

H264_DMA_IN_STATE_CH0 Represents the current control module state machine state for RX channel 0. (RO)

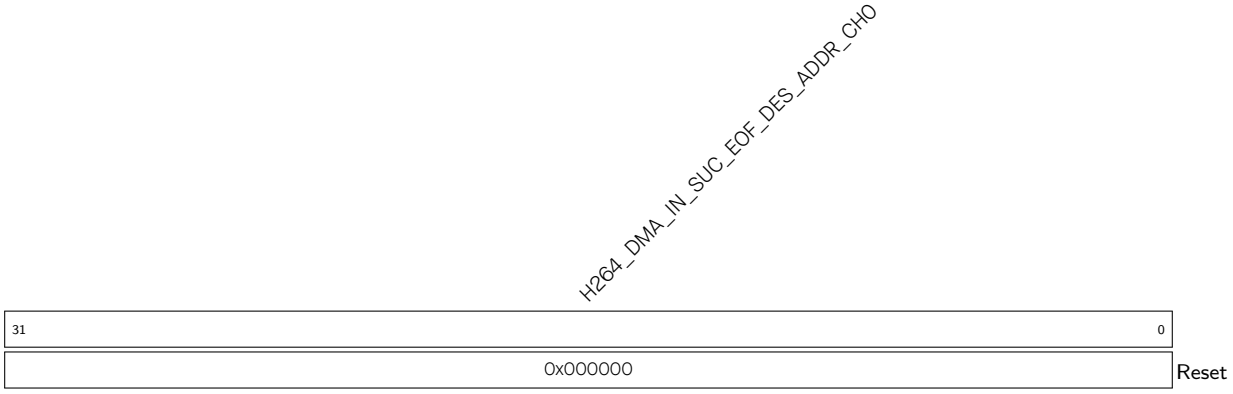
H264_DMA_IN_RESET_AVAIL_CH0 Represents whether it is safe to reset the channel.

0: Unsafe

1: Safe

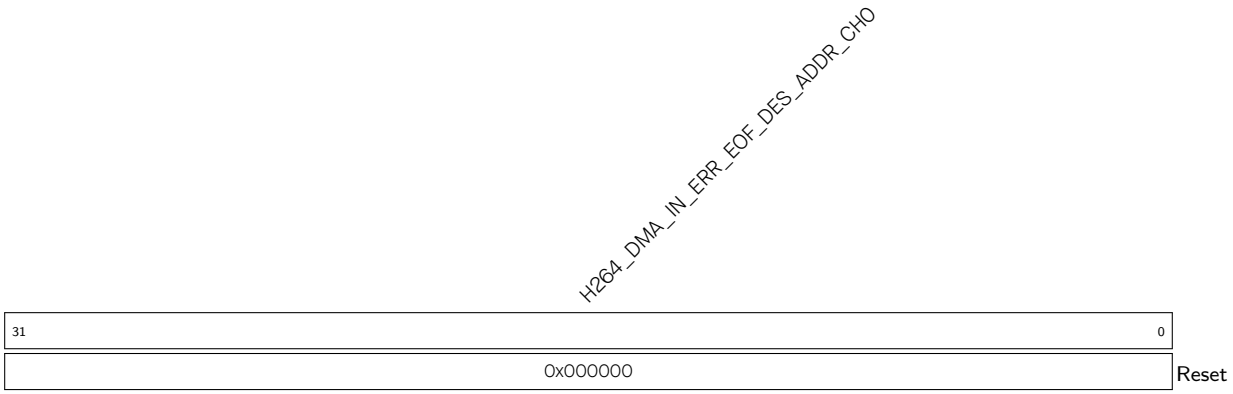
(RO)

Register 41.272. H264_DMA_IN_SUC_EOF_DES_ADDR_CH0_REG (0x0528)



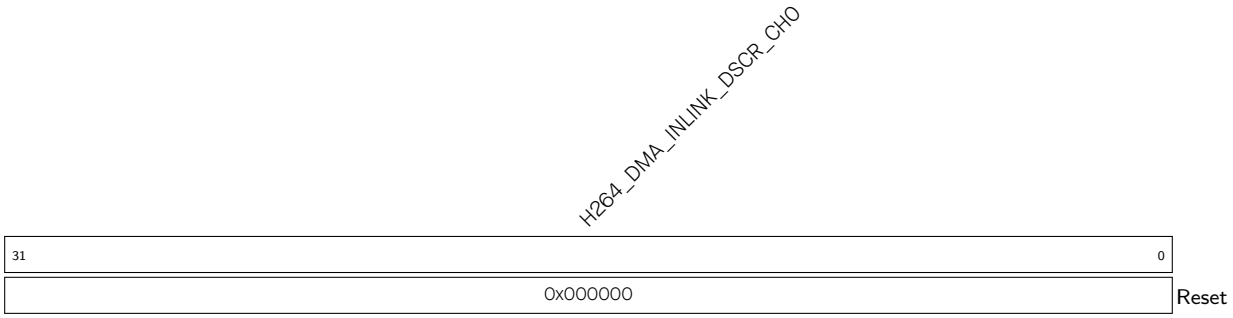
H264_DMA_IN_SUC_EOF_DES_ADDR_CH0 Represents the address of the inlink descriptor when the EOF bit in this descriptor is 1. (RO)

Register 41.273. H264_DMA_IN_ERR_EOF_DES_ADDR_CH0_REG (0x052C)



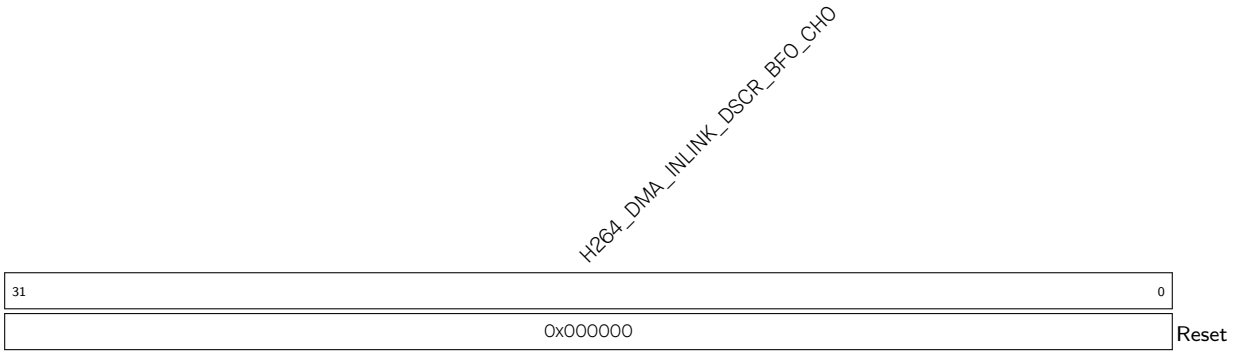
H264_DMA_IN_ERR_EOF_DES_ADDR_CH0 Represents the address of the inlink descriptor when there are errors during the current data reception. (RO)

Register 41.274. H264_DMA_IN_DSCR_CH0_REG (0x0530)



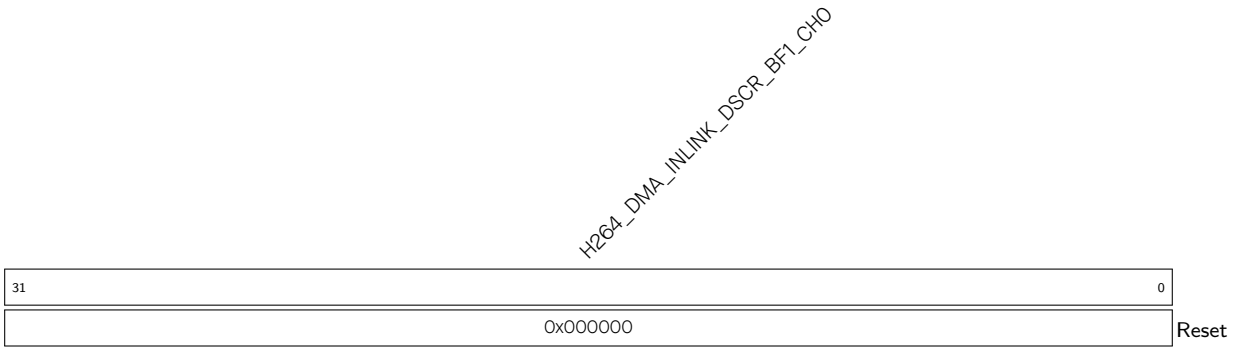
H264_DMA_INLINK_DSCR_CH0 Represents the address of the next descriptor x+1 pointed by the current descriptor that is pre-read. (RO)

Register 41.275. H264_DMA_IN_DSCR_BF0_CH0_REG (0x0534)



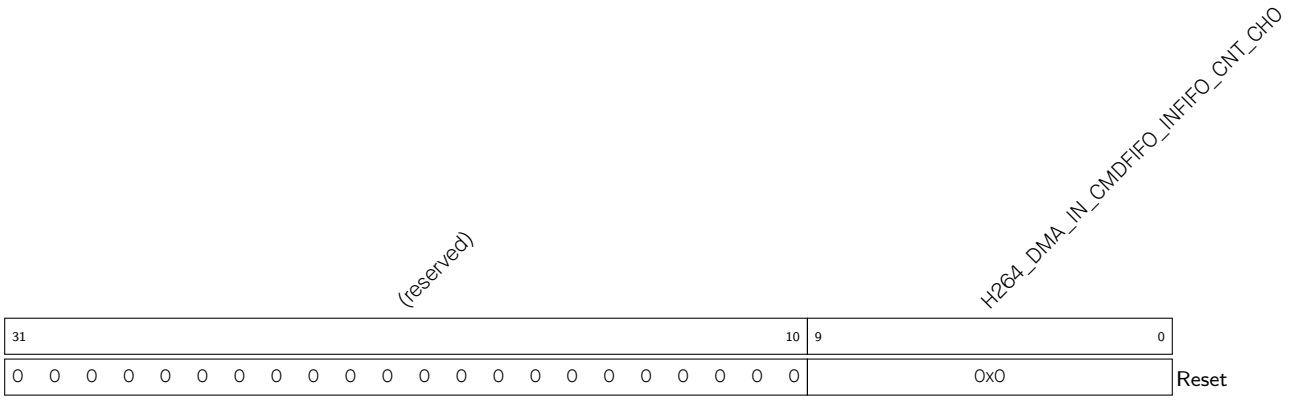
H264_DMA_INLINK_DSCR_BF0_CH0 Represents the address of the current descriptor x that is pre-read. (RO)

Register 41.276. H264_DMA_IN_DSCR_BF1_CH0_REG (0x0538)



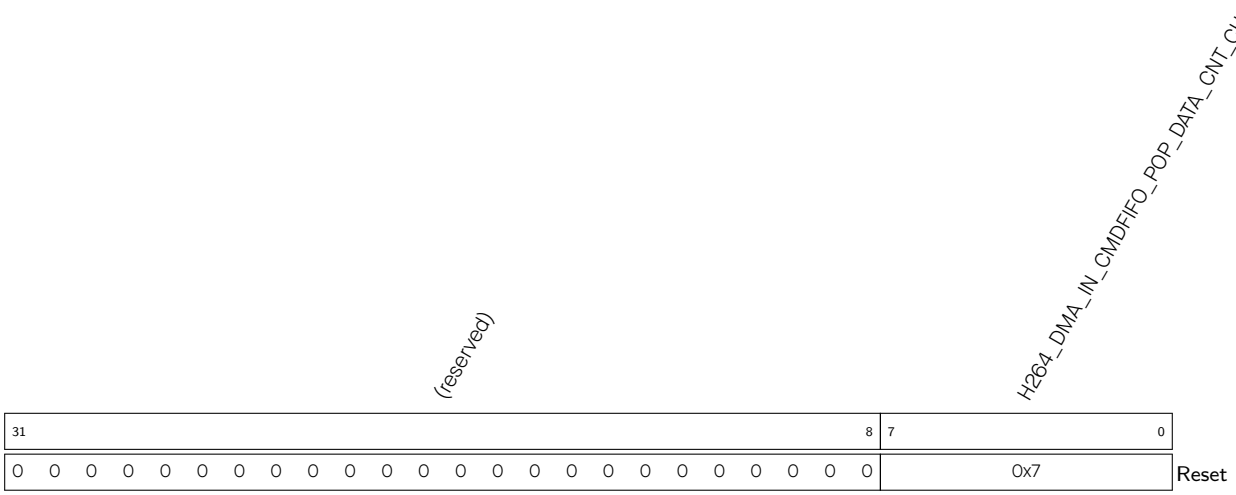
H264_DMA_INLINK_DSCR_BF1_CH0 Represents the address of the previous descriptor x-1 that is pre-read. (RO)

Register 41.277. H264_DMA_IN_FIFO_CNT_CH0_REG (0x0580)



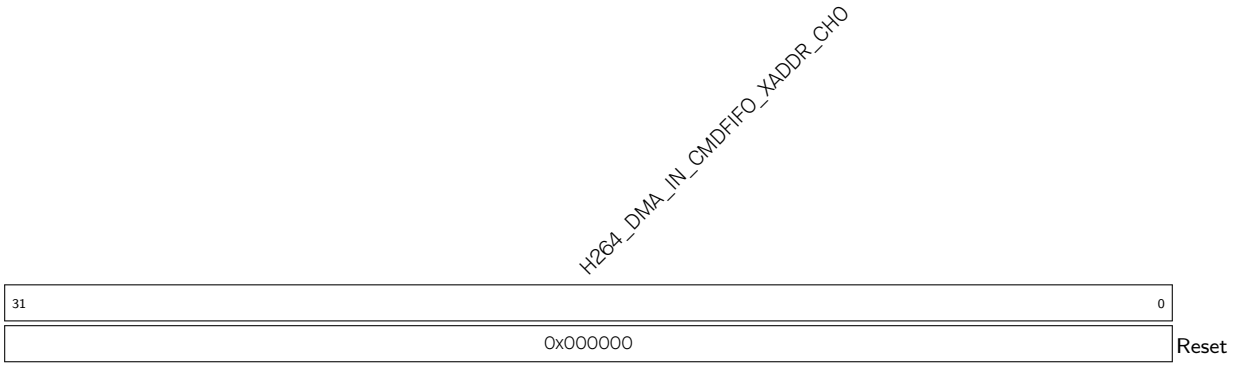
H264_DMA_IN_CMDFIFO_INFIFO_CNT_CH0 Represents the remaining request data in FIFO. (RO)

Register 41.278. H264_DMA_IN_POP_DATA_CNT_CH0_REG (0x0584)



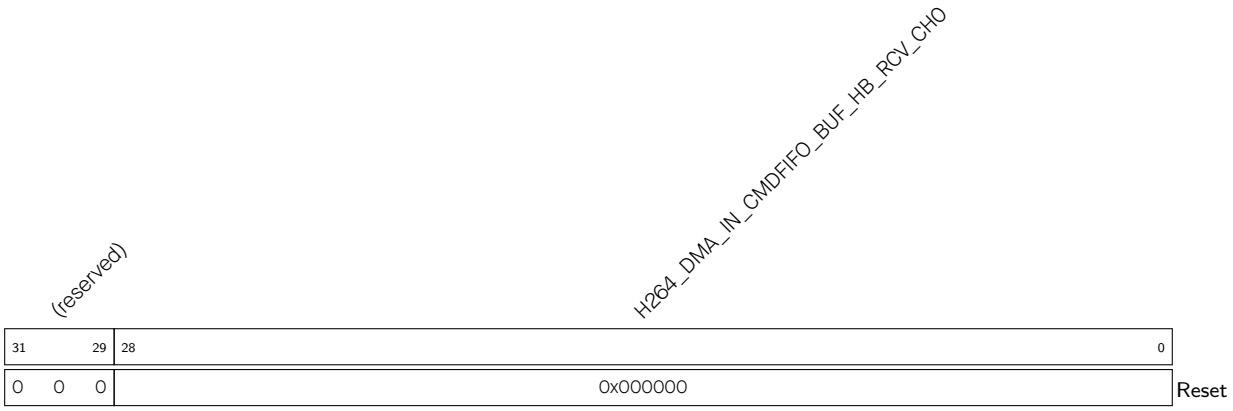
H264_DMA_IN_CMDFIFO_POP_DATA_CNT_CH0 Represents the data quantity in the current AXI request. (RO)

Register 41.279. H264_DMA_IN_XADDR_CH0_REG (0x0588)



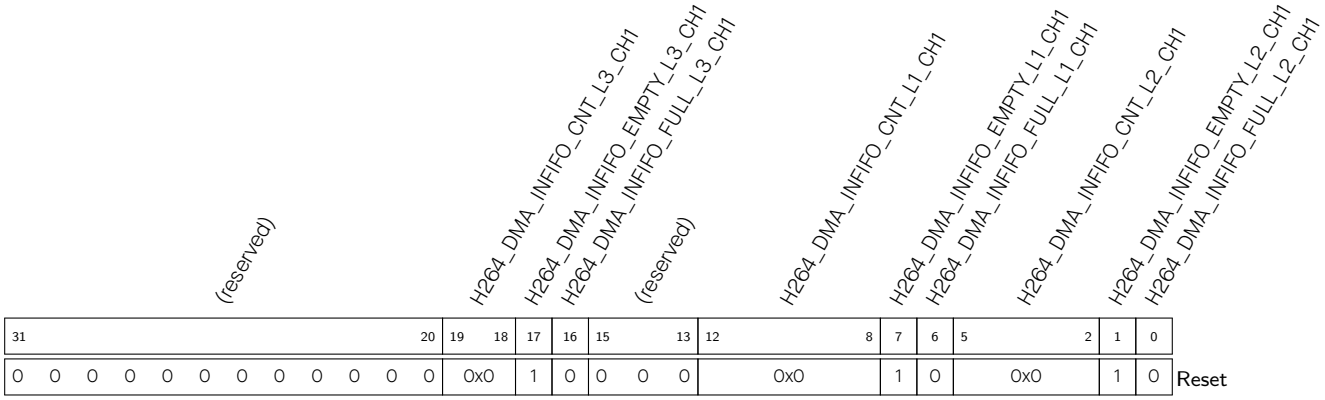
H264_DMA_IN_CMDFIFO_XADDR_CH0 Represents the address of the current AXI request. (RO)

Register 41.280. H264_DMA_IN_BUF_HB_RCV_CH0_REG (0x058C)



H264_DMA_IN_CMDFIFO_BUF_HB_RCV_CH0 Represents the data quantity that has been requested. (RO)

Register 41.281. H264_DMA_INFIFO_STATUS_CH1_REG (0x0614)



H264_DMA_INFIFO_FULL_L2_CH1 Represents whether l2 FIFO of RX channel 1 is full.

0: Not Full

1: Full

(RO)

H264_DMA_INFIFO_EMPTY_L2_CH1 Represents whether l2 FIFO of RX channel 1 is empty.

0: Not empty

1: Empty

(RO)

H264_DMA_INFIFO_CNT_L2_CH1 Represents the data quantity in l2 FIFO for RX channel 1. Measurement unit: 16 bytes. (RO)

H264_DMA_INFIFO_FULL_L1_CH1 Represents whether l1 FIFO of RX channel 1 is full.

0: Not Full

1: Full

(RO)

H264_DMA_INFIFO_EMPTY_L1_CH1 Represents whether l1 FIFO of RX channel 1 is empty.

0: Not empty

1: Empty

(RO)

H264_DMA_INFIFO_CNT_L1_CH1 Represents the data quantity in l1 FIFO for RX channel 1. Measurement unit: byte. (RO)

H264_DMA_INFIFO_FULL_L3_CH1 Represents whether l3 FIFO of RX channel 1 is full.

0: Not Full

1: Full

(RO)

H264_DMA_INFIFO_EMPTY_L3_CH1 Represents whether l3 FIFO of RX channel 1 is empty.

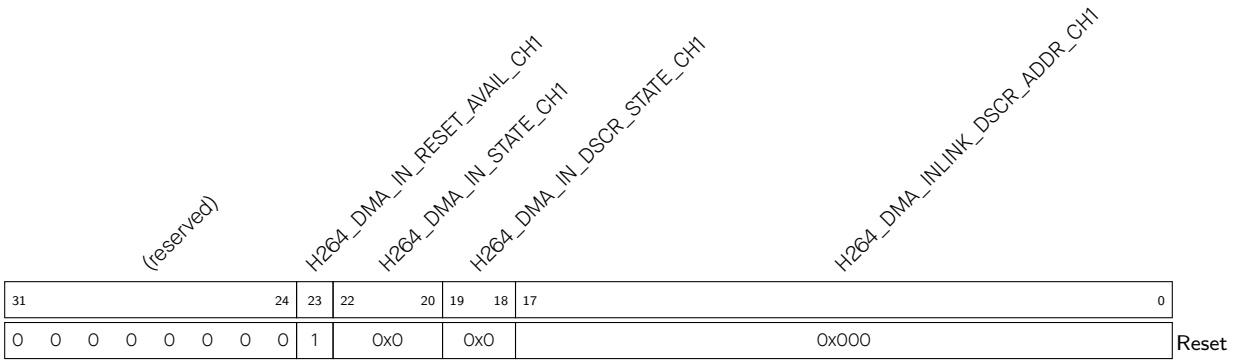
0: Not empty

1: Empty

(RO)

H264_DMA_INFIFO_CNT_L3_CH1 Represents the data quantity in l3 FIFO for RX channel 1. Measurement unit: 8 bytes. (RO)

Register 41.282. H264_DMA_IN_STATE_CH1_REG (0x0624)



H264_DMA_INLINK_DSCR_ADDR_CH1 Represents the current inlink descriptor's address for RX channel 1. (RO)

H264_DMA_IN_DSCR_STATE_CH1 Represents the current state of the descriptor state machine for RX channel 1. (RO)

H264_DMA_IN_STATE_CH1 Represents the current control module state machine state for RX channel 1. (RO)

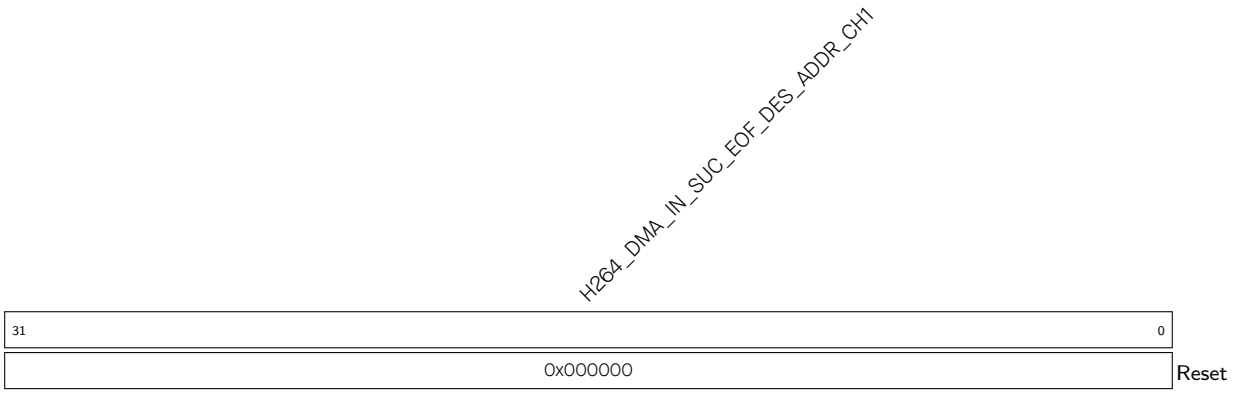
H264_DMA_IN_RESET_AVAIL_CH1 Represents whether it is safe to reset the channel.

0: Unsafe

1: Safe

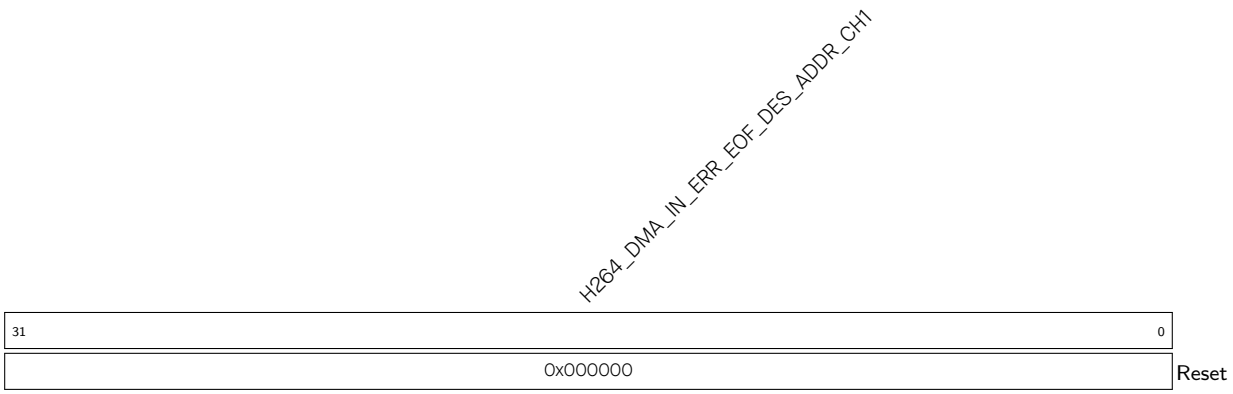
(RO)

Register 41.283. H264_DMA_IN_SUC_EOF_DES_ADDR_CH1_REG (0x0628)



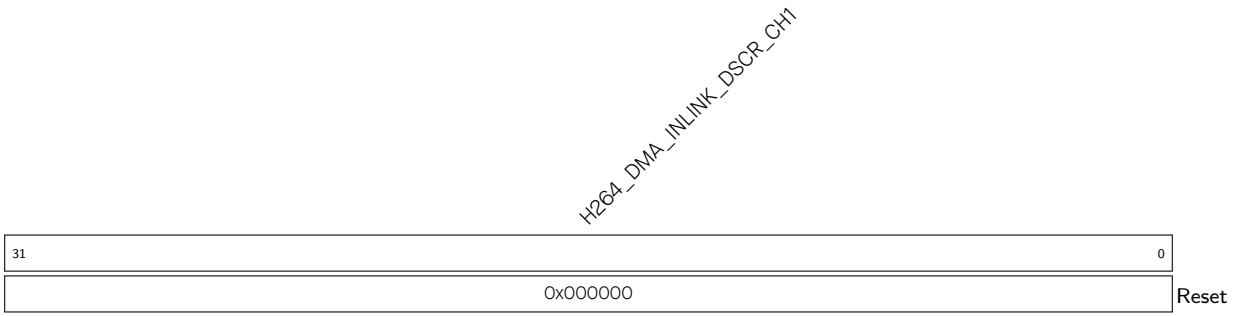
H264_DMA_IN_SUC_EOF_DES_ADDR_CH1 Represents the address of the inlink descriptor when the EOF bit in this descriptor is 1. (RO)

Register 41.284. H264_DMA_IN_ERR_EOF_DES_ADDR_CH1_REG (0x062C)



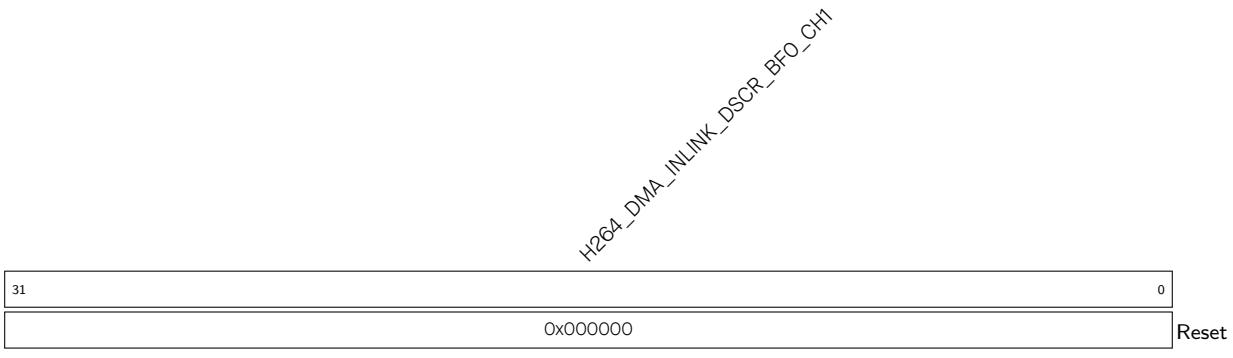
H264_DMA_IN_ERR_EOF_DES_ADDR_CH1 Represents the address of the inlink descriptor when there are errors during the current data reception. (RO)

Register 41.285. H264_DMA_IN_DSCR_CH1_REG (0x0630)



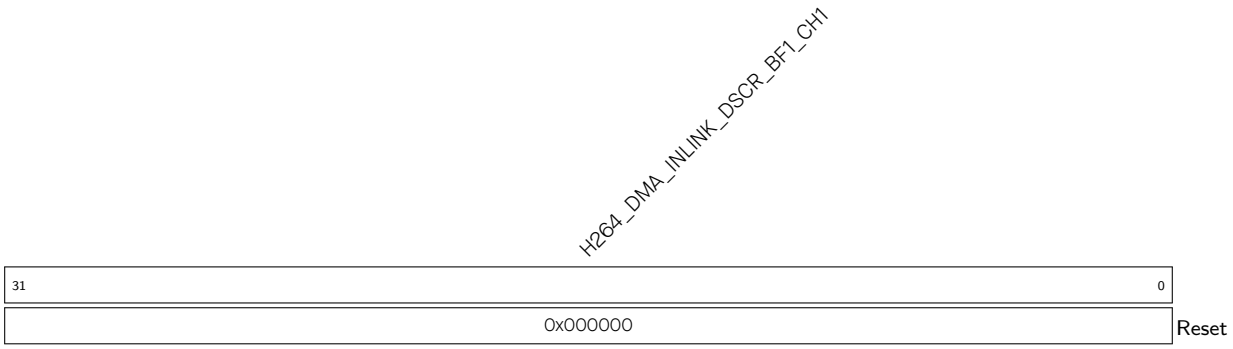
H264_DMA_INLINK_DSCR_CH1 Represents the address of the next descriptor x+1 pointed by the current descriptor that is pre-read. (RO)

Register 41.286. H264_DMA_IN_DSCR_BF0_CH1_REG (0x0634)



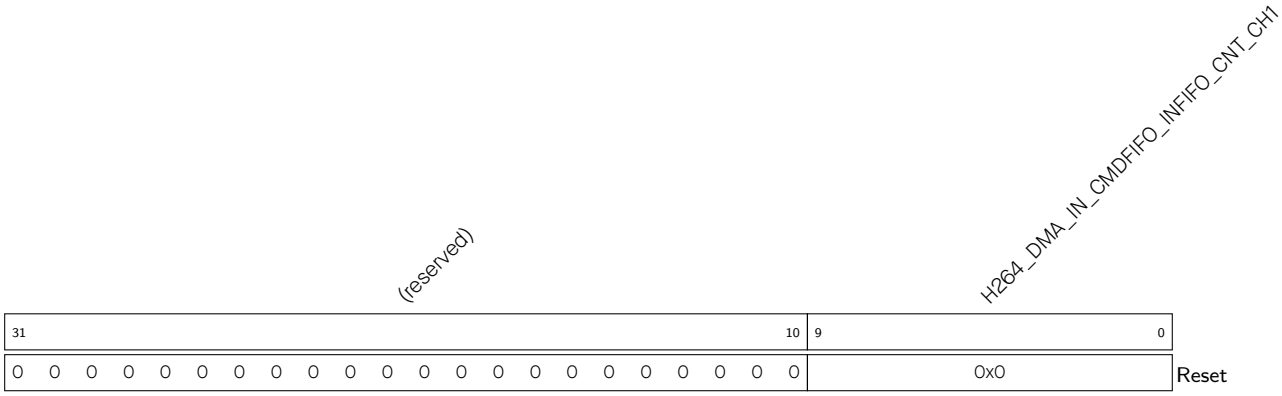
H264_DMA_INLINK_DSCR_BF0_CH1 Represents the address of the current descriptor x that is pre-read. (RO)

Register 41.287. H264_DMA_IN_DSCR_BF1_CH1_REG (0x0638)



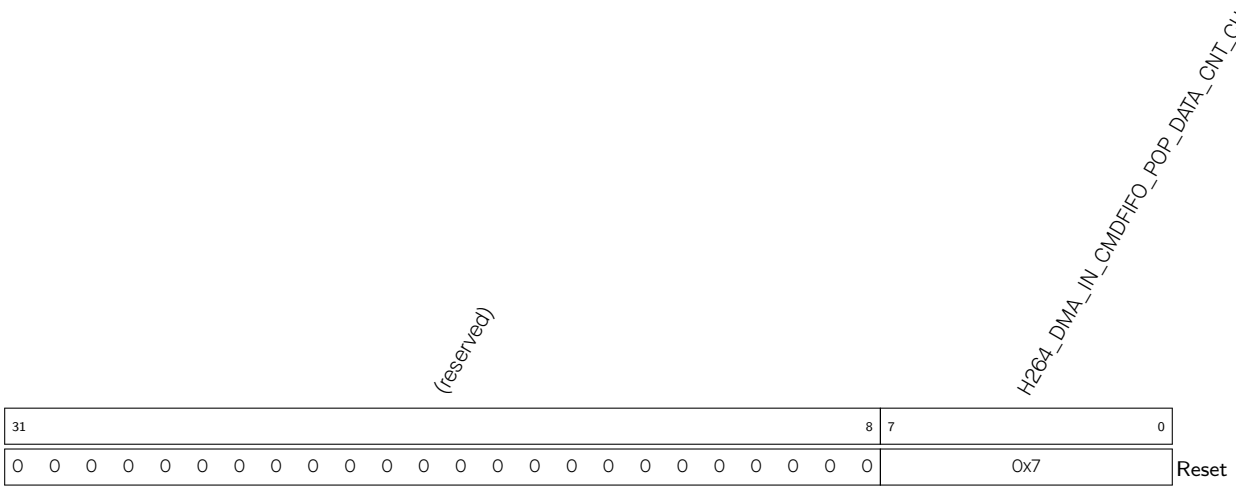
H264_DMA_INLINK_DSCR_BF1_CH1 Represents the address of the previous descriptor x-1 that is pre-read. (RO)

Register 41.288. H264_DMA_IN_FIFO_CNT_CH1_REG (0x0680)



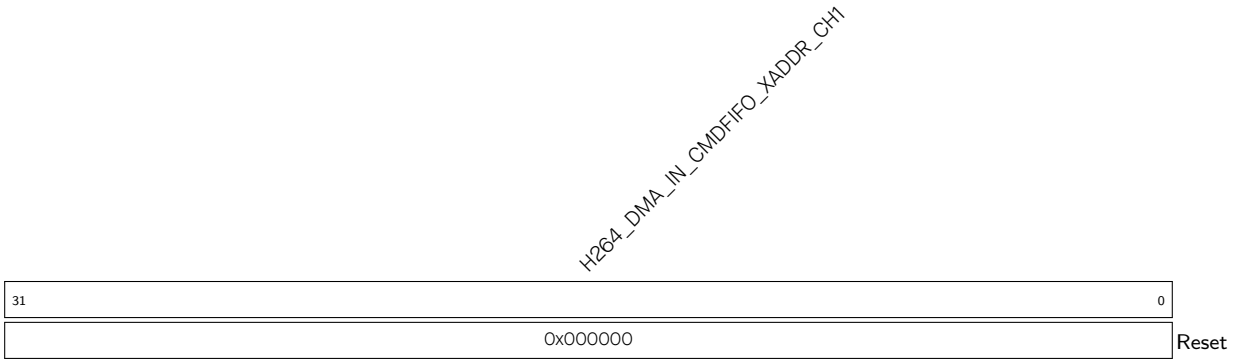
H264_DMA_IN_CMDFIFO_INFIFO_CNT_CH1 Represents the remaining request data in FIFO. (RO)

Register 41.289. H264_DMA_IN_POP_DATA_CNT_CH1_REG (0x0684)



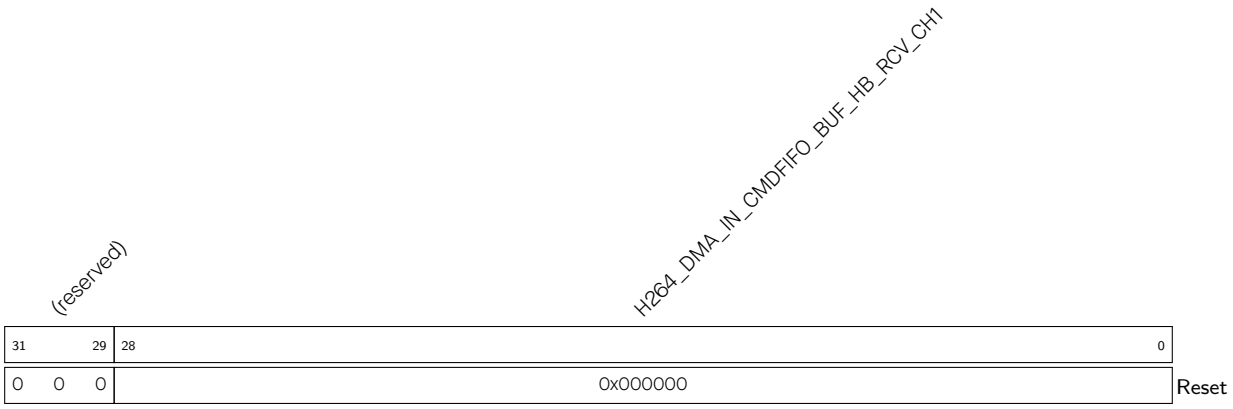
H264_DMA_IN_CMDFIFO_POP_DATA_CNT_CH1 Represents the data quantity in the current AXI request. (RO)

Register 41.290. H264_DMA_IN_XADDR_CH1_REG (0x0688)



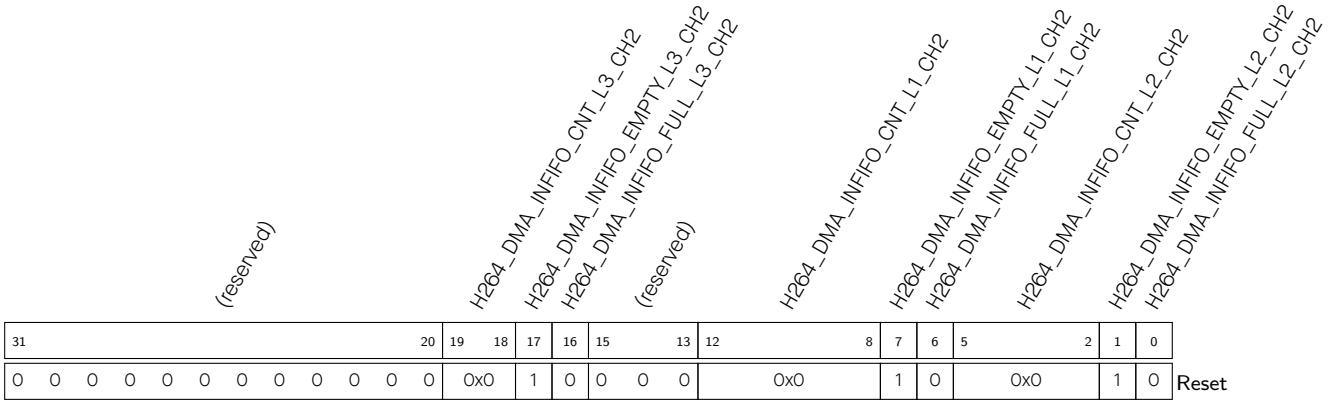
H264_DMA_IN_CMDFIFO_XADDR_CH1 Represents the address of the current AXI request. (RO)

Register 41.291. H264_DMA_IN_BUF_HB_RCV_CH1_REG (0x068C)



H264_DMA_IN_CMDFIFO_BUF_HB_RCV_CH1 Represents the data quantity that has been requested. (RO)

Register 41.292. H264_DMA_INFIFO_STATUS_CH2_REG (0x0714)



H264_DMA_INFIFO_FULL_L2_CH2 Represents whether l2 FIFO of RX channel 2 is full.

0: Not Full

1: Full

(RO)

H264_DMA_INFIFO_EMPTY_L2_CH2 Represents whether l2 FIFO of RX channel 2 is empty.

0: Not empty

1: Empty

(RO)

H264_DMA_INFIFO_CNT_L2_CH2 Represents the data quantity in l2 FIFO for RX channel 2. Measurement unit: 16 bytes. (RO)

H264_DMA_INFIFO_FULL_L1_CH2 Represents whether l1 FIFO of RX channel 2 is full.

0: Not Full

1: Full

(RO)

H264_DMA_INFIFO_EMPTY_L1_CH2 Represents whether l1 FIFO of RX channel 2 is empty.

0: Not empty

1: Empty

(RO)

H264_DMA_INFIFO_CNT_L1_CH2 Represents the data quantity in l1 FIFO for RX channel 2. Measurement unit: byte. (RO)

H264_DMA_INFIFO_FULL_L3_CH2 Represents whether l3 FIFO of RX channel 2 is full.

0: Not Full

1: Full

(RO)

H264_DMA_INFIFO_EMPTY_L3_CH2 Represents whether l3 FIFO of RX channel 2 is empty.

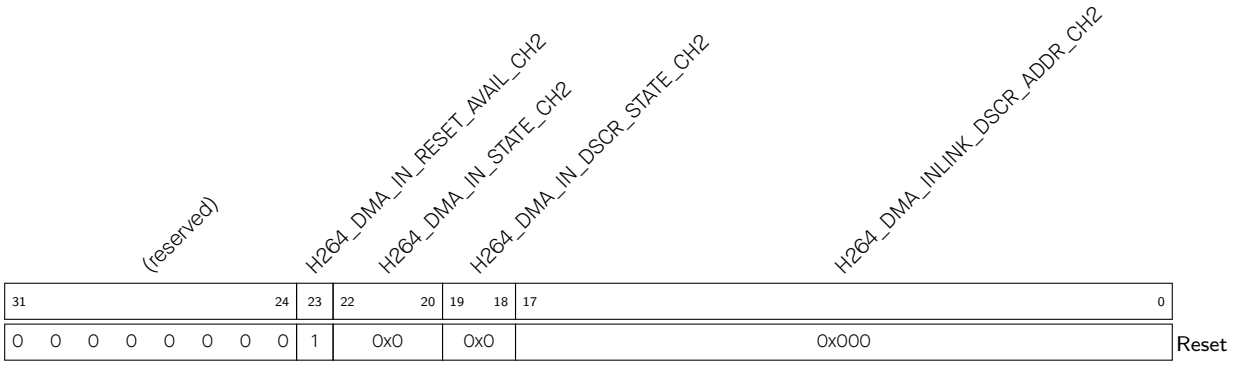
0: Not empty

1: Empty

(RO)

H264_DMA_INFIFO_CNT_L3_CH2 Represents the data quantity in l3 FIFO for RX channel 2. Measurement unit: 8 bytes. (RO)

Register 41.293. H264_DMA_IN_STATE_CH2_REG (0x0724)



H264_DMA_INLINK_DSCR_ADDR_CH2 Represents the current inlink descriptor's address for RX channel 2. (RO)

H264_DMA_IN_DSCR_STATE_CH2 Represents the current state of the descriptor state machine for RX channel 2. (RO)

H264_DMA_IN_STATE_CH2 Represents the current control module state machine state for RX channel 2. (RO)

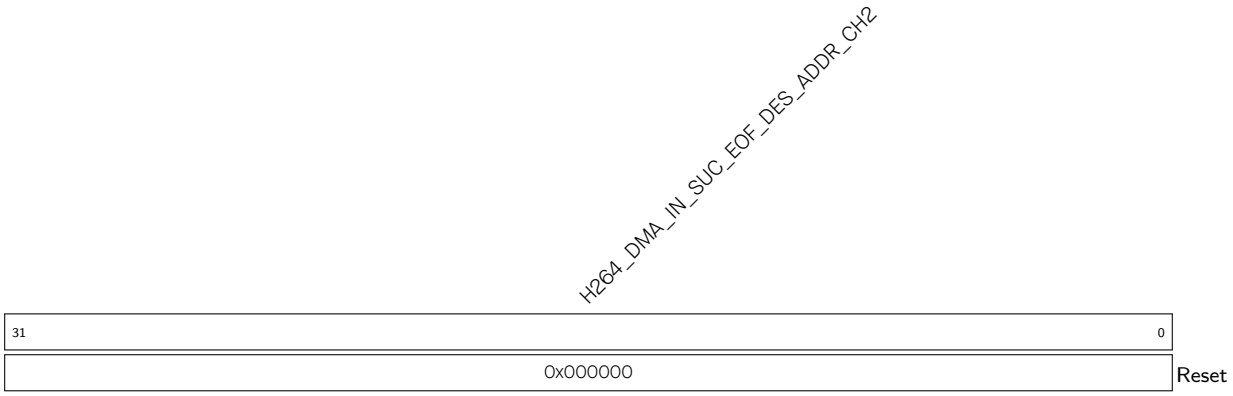
H264_DMA_IN_RESET_AVAIL_CH2 Represents whether it is safe to reset the channel.

0: Unsafe

1: Safe

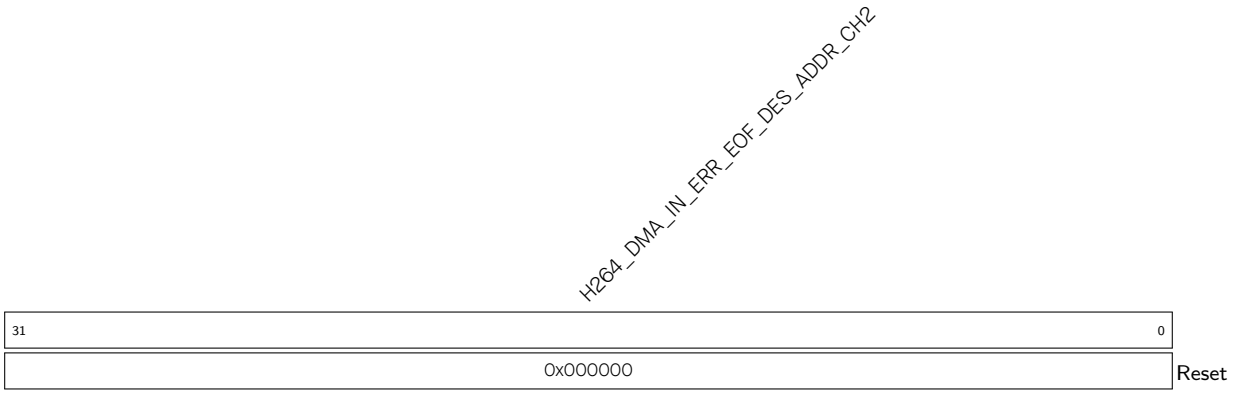
(RO)

Register 41.294. H264_DMA_IN_SUC_EOF_DES_ADDR_CH2_REG (0x0728)



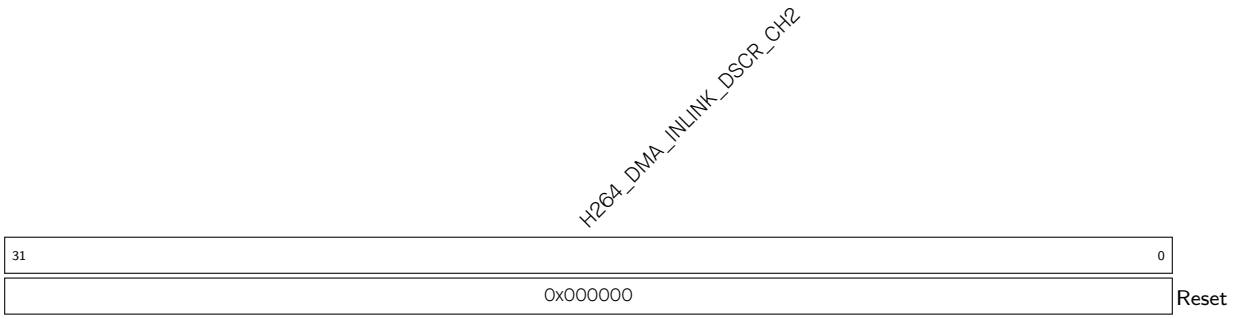
H264_DMA_IN_SUC_EOF_DES_ADDR_CH2 Represents the address of the inlink descriptor when the EOF bit in this descriptor is 1. (RO)

Register 41.295. H264_DMA_IN_ERR_EOF_DES_ADDR_CH2_REG (0x072C)



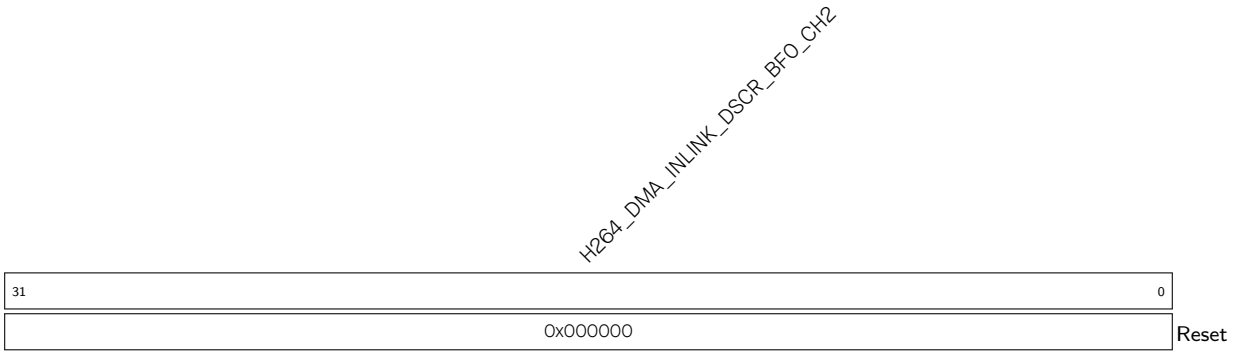
H264_DMA_IN_ERR_EOF_DES_ADDR_CH2 Represents the address of the inlink descriptor when there are errors during the current data reception. (RO)

Register 41.296. H264_DMA_IN_DSCR_CH2_REG (0x0730)



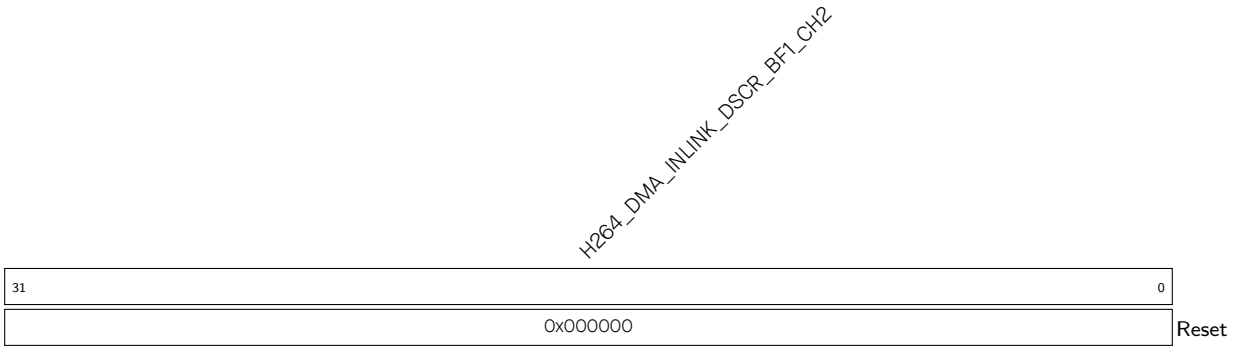
H264_DMA_INLINK_DSCR_CH2 Represents the address of the next descriptor x+1 pointed by the current descriptor that is pre-read. (RO)

Register 41.297. H264_DMA_IN_DSCR_BF0_CH2_REG (0x0734)



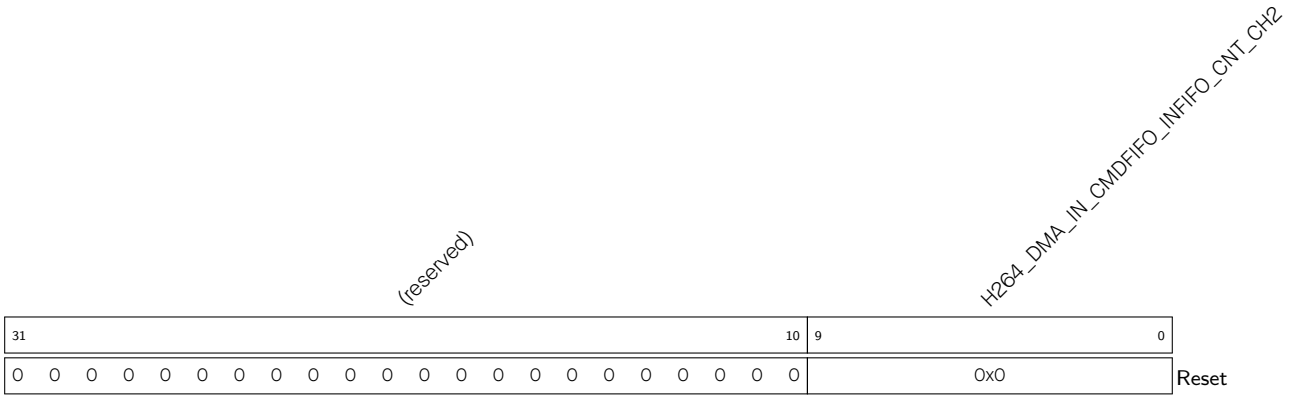
H264_DMA_INLINK_DSCR_BF0_CH2 Represents the address of the current descriptor x that is pre-read. (RO)

Register 41.298. H264_DMA_IN_DSCR_BF1_CH2_REG (0x0738)



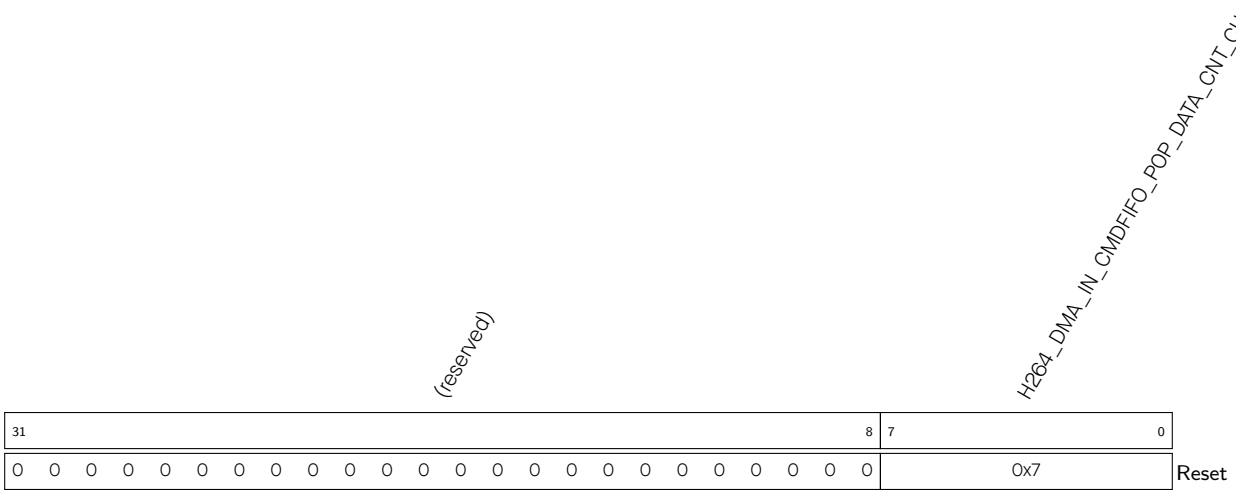
H264_DMA_INLINK_DSCR_BF1_CH2 Represents the address of the previous descriptor x-1 that is pre-read. (RO)

Register 41.299. H264_DMA_IN_FIFO_CNT_CH2_REG (0x0780)



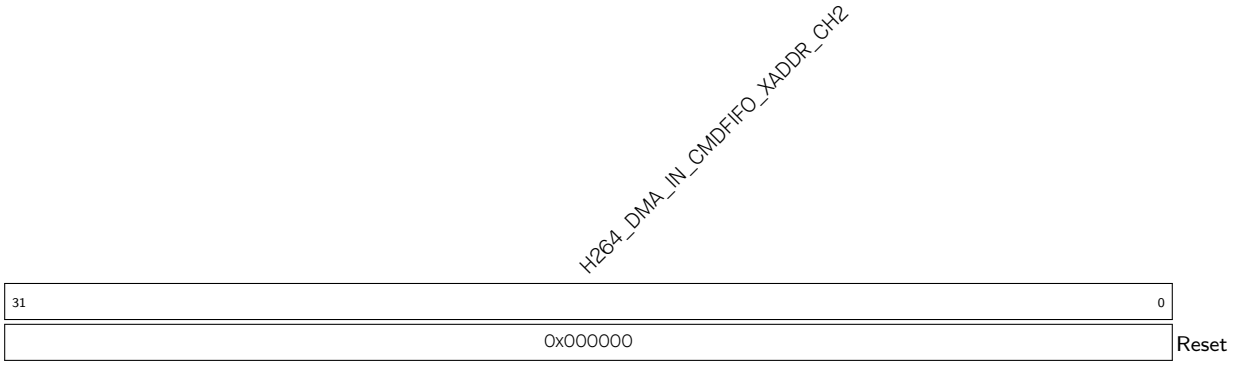
H264_DMA_IN_CMDFIFO_INFIFO_CNT_CH2 Represents the remaining request data in FIFO. (RO)

Register 41.300. H264_DMA_IN_POP_DATA_CNT_CH2_REG (0x0784)



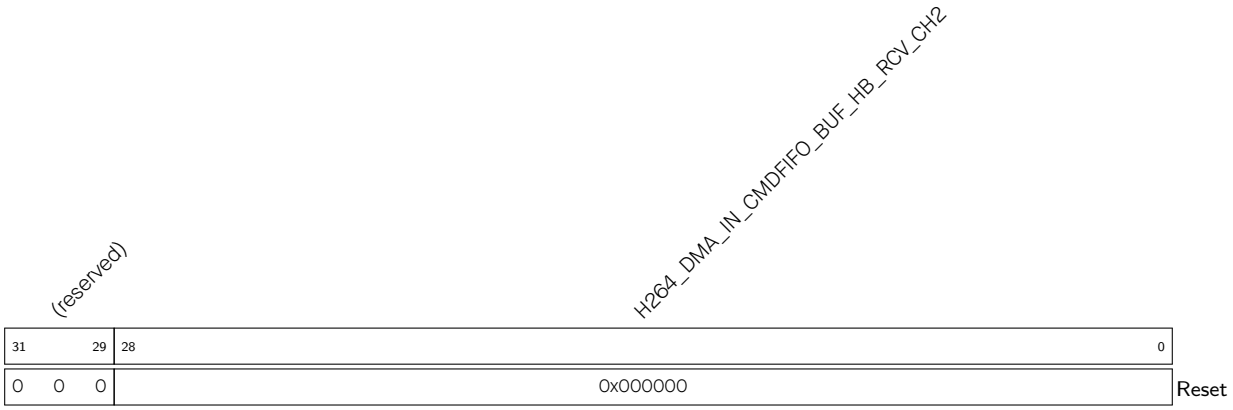
H264_DMA_IN_CMDFIFO_POP_DATA_CNT_CH2 Represents the data quantity in the current AXI request. (RO)

Register 41.301. H264_DMA_IN_XADDR_CH2_REG (0x0788)



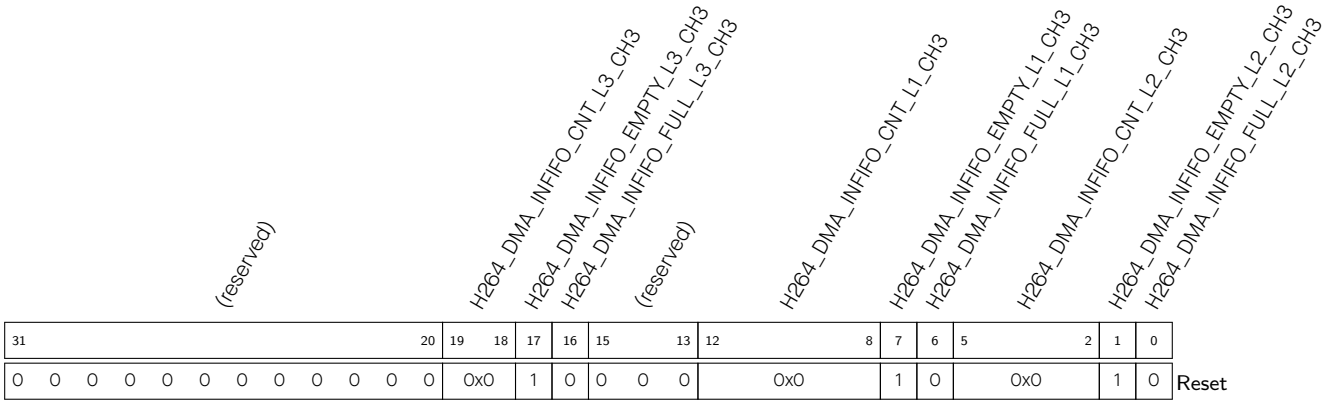
H264_DMA_IN_CMDFIFO_XADDR_CH2 Represents the address of the current AXI request. (RO)

Register 41.302. H264_DMA_IN_BUF_HB_RCV_CH2_REG (0x078C)



H264_DMA_IN_CMDFIFO_BUF_HB_RCV_CH2 Represents the data quantity that has been requested. (RO)

Register 41.303. H264_DMA_INFIFO_STATUS_CH3_REG (0x0814)



H264_DMA_INFIFO_FULL_L2_CH3 Represents whether l2 FIFO of RX channel 3 is full.

0: Not Full

1: Full

(RO)

H264_DMA_INFIFO_EMPTY_L2_CH3 Represents whether l2 FIFO of RX channel 3 is empty.

0: Not empty

1: Empty

(RO)

H264_DMA_INFIFO_CNT_L2_CH3 Represents the data quantity in l2 FIFO for RX channel 3. Measurement unit: 16 bytes. (RO)

H264_DMA_INFIFO_FULL_L1_CH3 Represents whether l1 FIFO of RX channel 3 is full.

0: Not Full

1: Full

(RO)

H264_DMA_INFIFO_EMPTY_L1_CH3 Represents whether l1 FIFO of RX channel 3 is empty.

0: Not empty

1: Empty

(RO)

H264_DMA_INFIFO_CNT_L1_CH3 Represents the data quantity in l1 FIFO for RX channel 3. Measurement unit: byte. (RO)

H264_DMA_INFIFO_FULL_L3_CH3 Represents whether l3 FIFO of RX channel 3 is full.

0: Not Full

1: Full

(RO)

H264_DMA_INFIFO_EMPTY_L3_CH3 Represents whether l3 FIFO of RX channel 3 is empty.

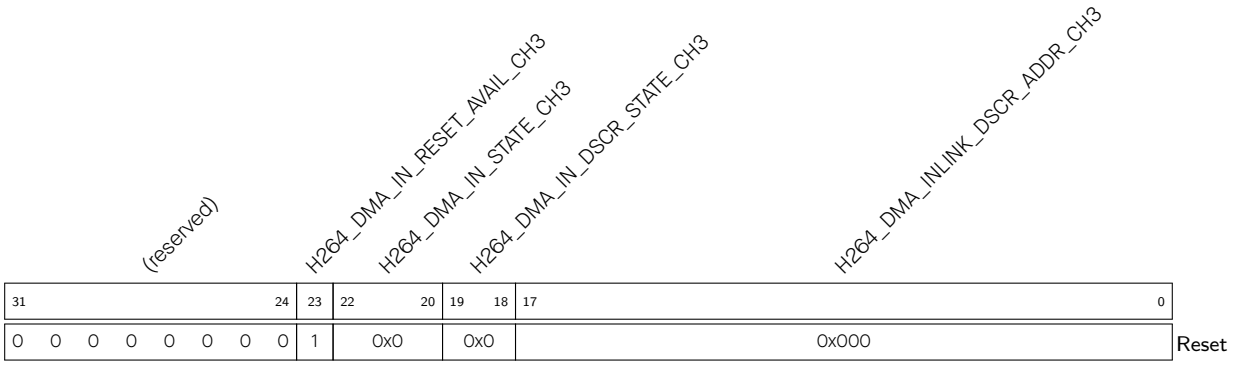
0: Not empty

1: Empty

(RO)

H264_DMA_INFIFO_CNT_L3_CH3 Represents the data quantity in l3 FIFO for RX channel 3. Measurement unit: 8 bytes. (RO)

Register 41.304. H264_DMA_IN_STATE_CH3_REG (0x0824)



H264_DMA_INLINK_DSCR_ADDR_CH3 Represents the current inlink descriptor's address for RX channel 3. (RO)

H264_DMA_IN_DSCR_STATE_CH3 Represents the current state of the descriptor state machine for RX channel 3. (RO)

H264_DMA_IN_STATE_CH3 Represents the current control module state machine state for RX channel 3. (RO)

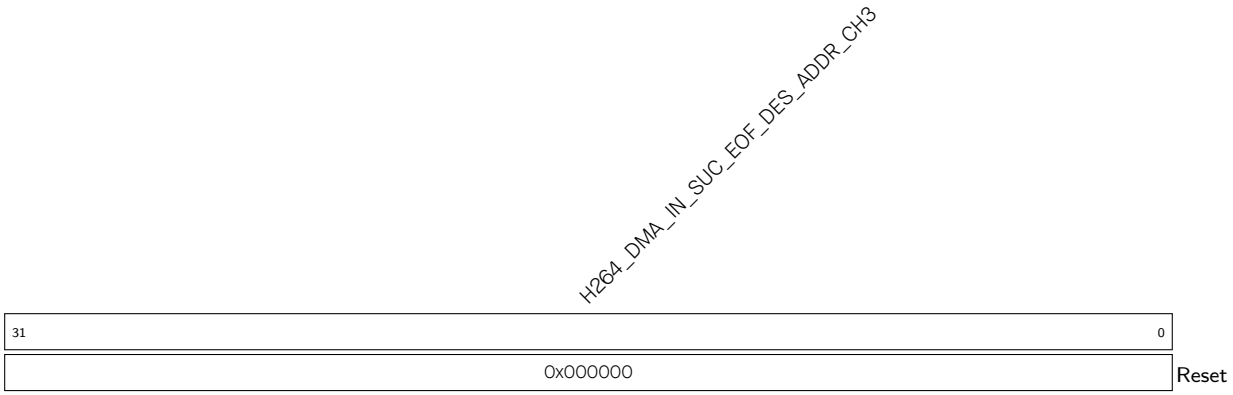
H264_DMA_IN_RESET_AVAIL_CH3 Represents whether it is safe to reset the channel.

0: Unsafe

1: Safe

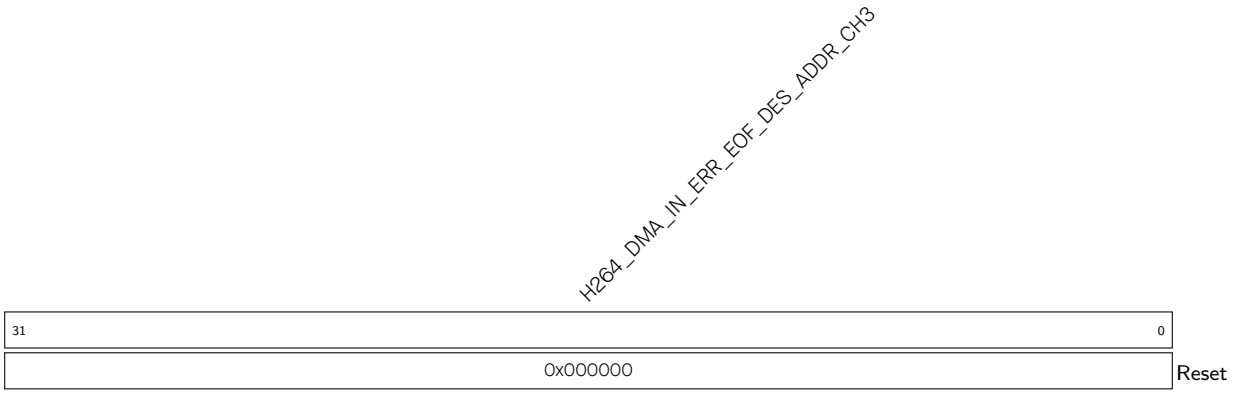
(RO)

Register 41.305. H264_DMA_IN_SUC_EOF_DES_ADDR_CH3_REG (0x0828)



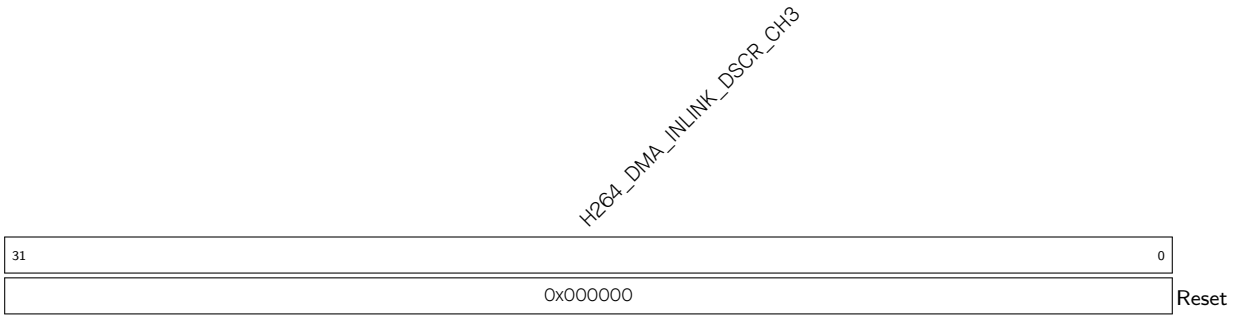
H264_DMA_IN_SUC_EOF_DES_ADDR_CH3 Represents the address of the inlink descriptor when the EOF bit in this descriptor is 1. (RO)

Register 41.306. H264_DMA_IN_ERR_EOF_DES_ADDR_CH3_REG (0x082C)



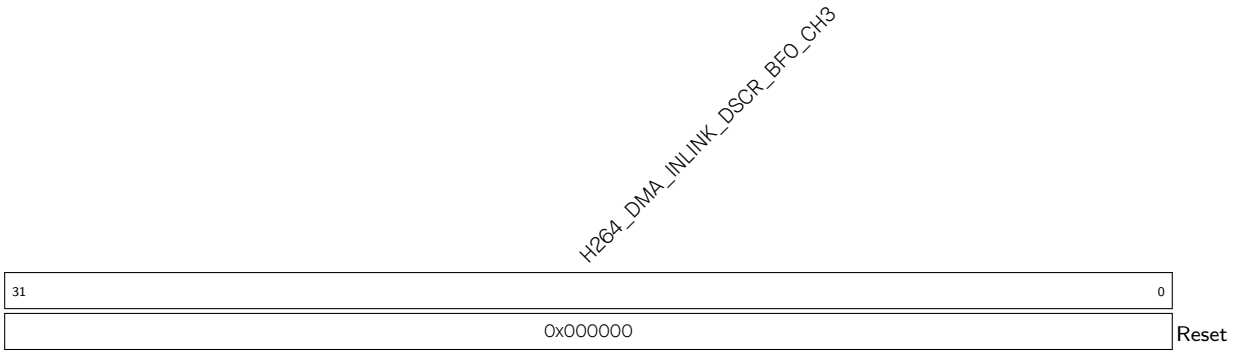
H264_DMA_IN_ERR_EOF_DES_ADDR_CH3 Represents the address of the inlink descriptor when there are errors during the current data reception. (RO)

Register 41.307. H264_DMA_IN_DSCR_CH3_REG (0x0830)



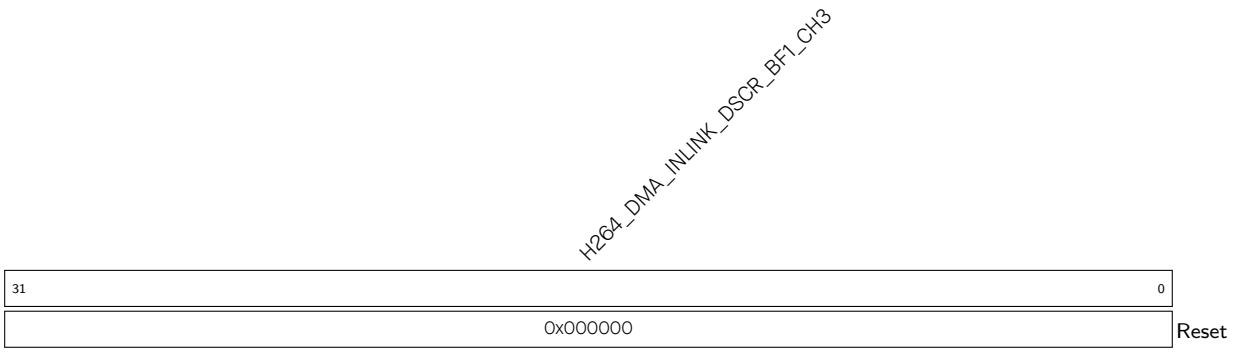
H264_DMA_INLINK_DSCR_CH3 Represents the address of the next descriptor x+1 pointed by the current descriptor that is pre-read. (RO)

Register 41.308. H264_DMA_IN_DSCR_BF0_CH3_REG (0x0834)



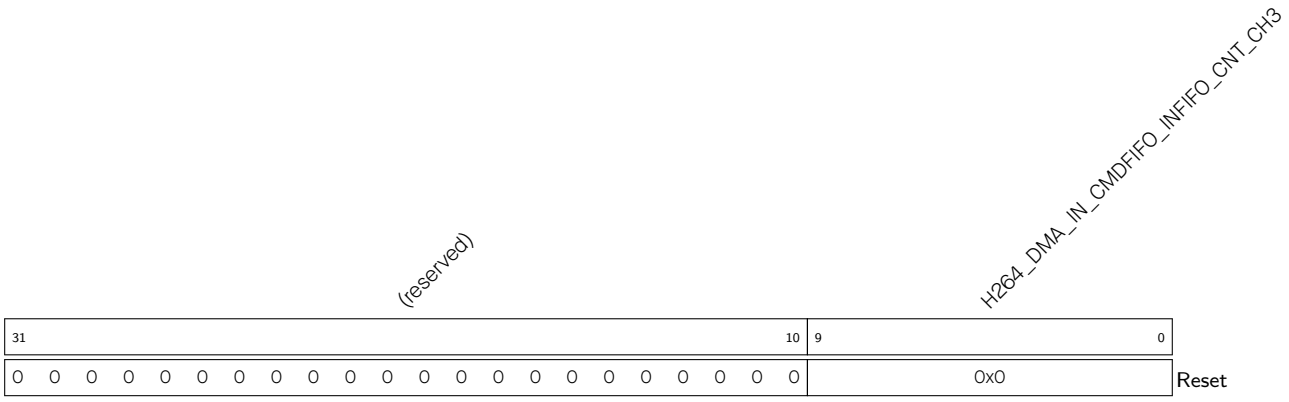
H264_DMA_INLINK_DSCR_BF0_CH3 Represents the address of the current descriptor x that is pre-read. (RO)

Register 41.309. H264_DMA_IN_DSCR_BF1_CH3_REG (0x0838)



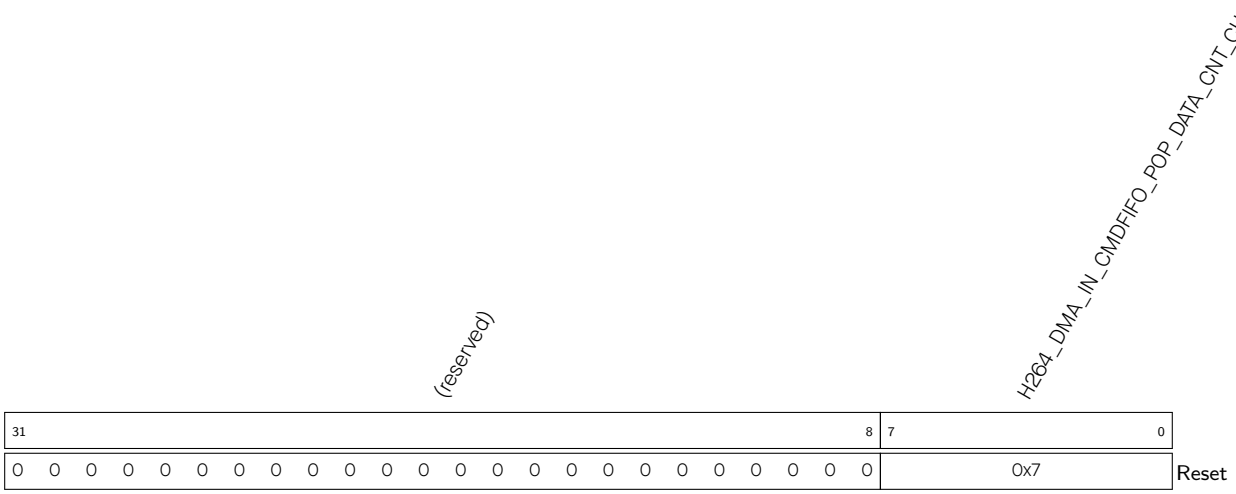
H264_DMA_INLINK_DSCR_BF1_CH3 Represents the address of the previous descriptor x-1 that is pre-read. (RO)

Register 41.310. H264_DMA_IN_FIFO_CNT_CH3_REG (0x0880)



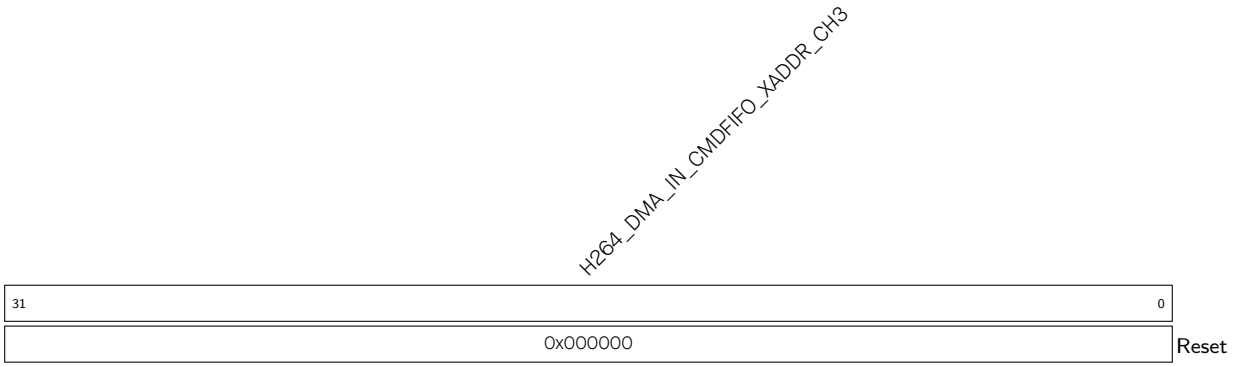
H264_DMA_IN_CMDFIFO_INFIFO_CNT_CH3 Represents the remaining request data in FIFO. (RO)

Register 41.311. H264_DMA_IN_POP_DATA_CNT_CH3_REG (0x0884)



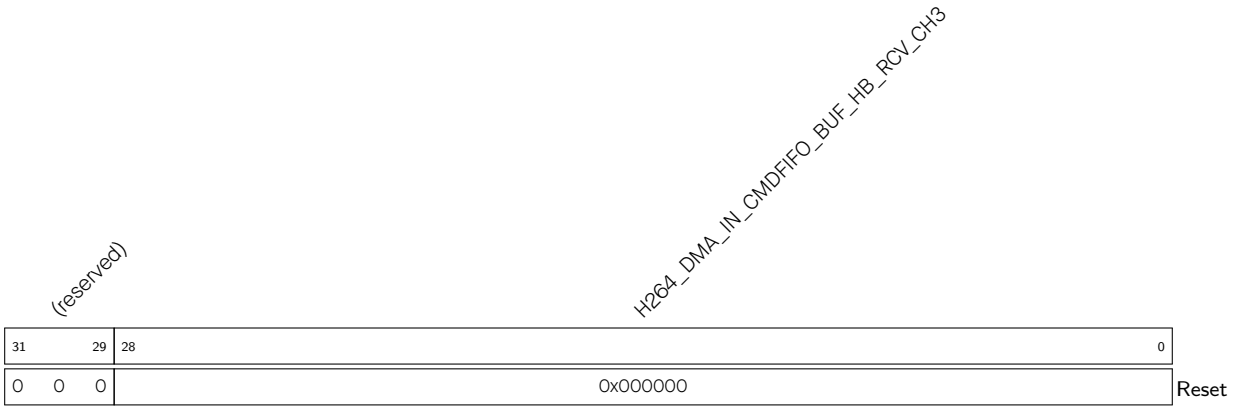
H264_DMA_IN_CMDFIFO_POP_DATA_CNT_CH3 Represents the data quantity in the current AXI request. (RO)

Register 41.312. H264_DMA_IN_XADDR_CH3_REG (0x0888)



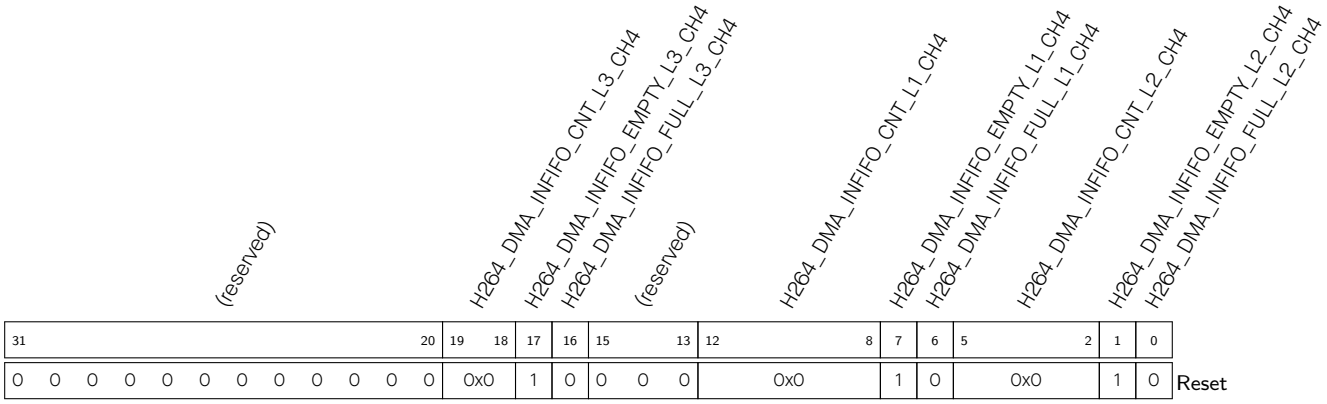
H264_DMA_IN_CMDFIFO_XADDR_CH3 Represents the address of the current AXI request. (RO)

Register 41.313. H264_DMA_IN_BUF_HB_RCV_CH3_REG (0x088C)



H264_DMA_IN_CMDFIFO_BUF_HB_RCV_CH3 Represents the data quantity that has been requested. (RO)

Register 41.314. H264_DMA_INFIFO_STATUS_CH4_REG (0x0914)



H264_DMA_INFIFO_FULL_L2_CH4 Represents whether l2 FIFO of RX channel 4 is full.

0: Not Full

1: Full

(RO)

H264_DMA_INFIFO_EMPTY_L2_CH4 Represents whether l2 FIFO of RX channel 4 is empty.

0: Not empty

1: Empty

(RO)

H264_DMA_INFIFO_CNT_L2_CH4 Represents the data quantity in l2 FIFO for RX channel 4. Measurement unit: 16 bytes. (RO)

H264_DMA_INFIFO_FULL_L1_CH4 Represents whether l1 FIFO of RX channel 4 is full.

0: Not Full

1: Full

(RO)

H264_DMA_INFIFO_EMPTY_L1_CH4 Represents whether l1 FIFO of RX channel 4 is empty.

0: Not empty

1: Empty

(RO)

H264_DMA_INFIFO_CNT_L1_CH4 Represents the data quantity in l1 FIFO for RX channel 4. Measurement unit: byte. (RO)

H264_DMA_INFIFO_FULL_L3_CH4 Represents whether l3 FIFO of RX channel 4 is full.

0: Not Full

1: Full

(RO)

H264_DMA_INFIFO_EMPTY_L3_CH4 Represents whether l3 FIFO of RX channel 4 is empty.

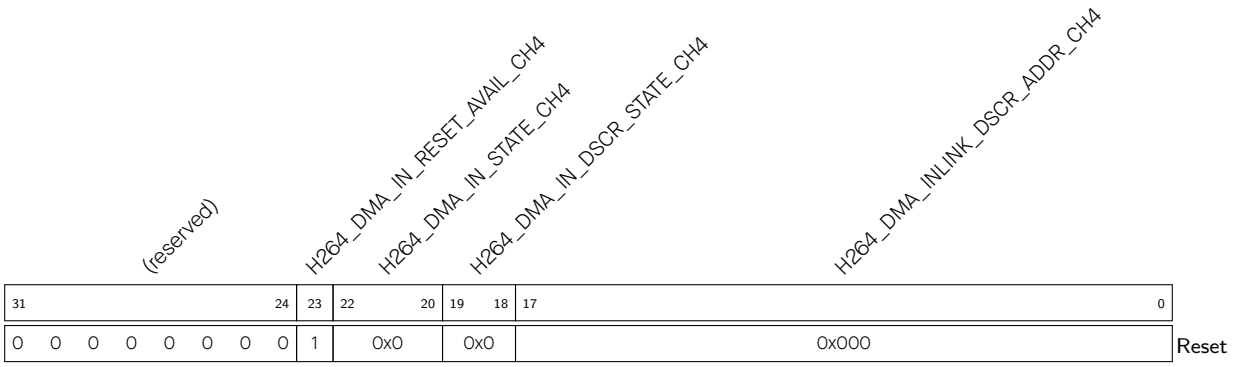
0: Not empty

1: Empty

(RO)

H264_DMA_INFIFO_CNT_L3_CH4 Represents the data quantity in l3 FIFO for RX channel 4. Measurement unit: 8 bytes. (RO)

Register 41.315. H264_DMA_IN_STATE_CH4_REG (0x0924)



H264_DMA_INLINK_DSCR_ADDR_CH4 Represents the current inlink descriptor's address for RX channel 4. (RO)

H264_DMA_IN_DSCR_STATE_CH4 Represents the current state of the descriptor state machine for RX channel 4. (RO)

H264_DMA_IN_STATE_CH4 Represents the current control module state machine state for RX channel 4. (RO)

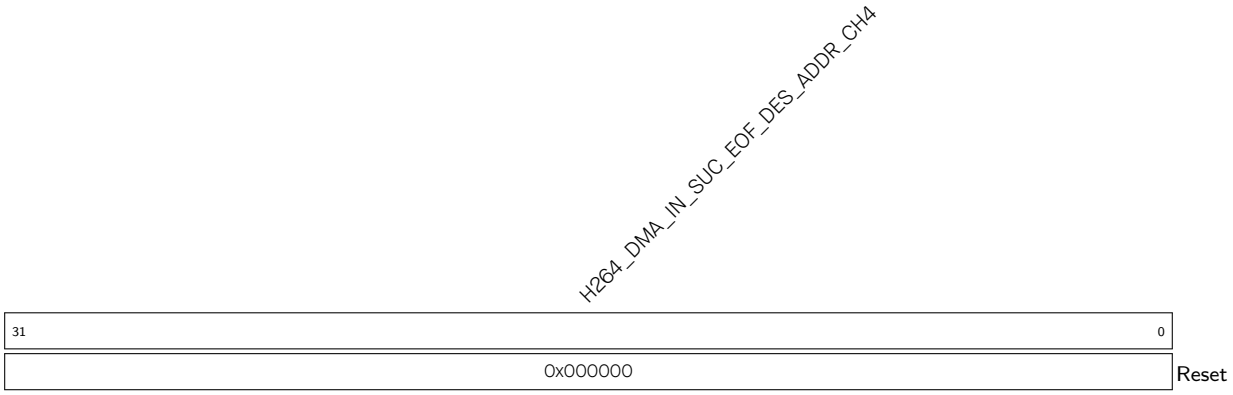
H264_DMA_IN_RESET_AVAIL_CH4 Represents whether it is safe to reset the channel.

0: Unsafe

1: Safe

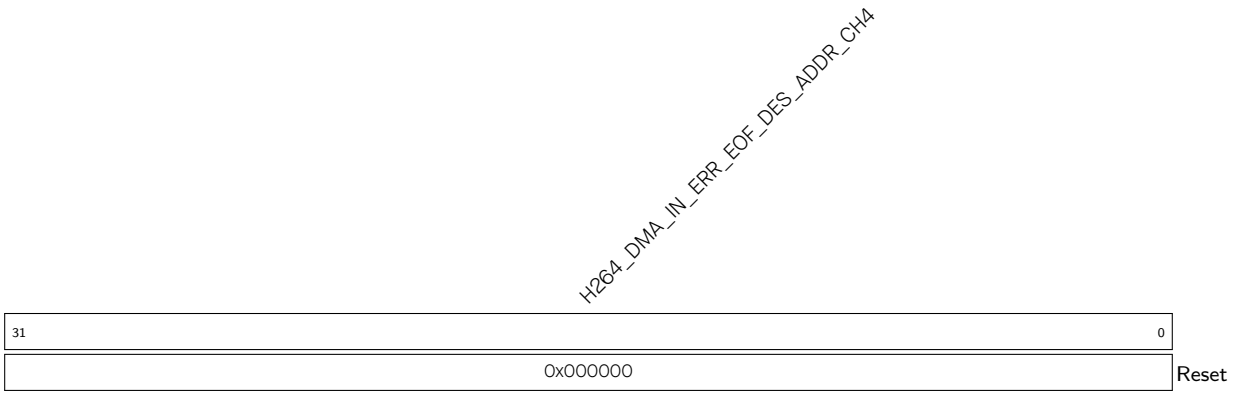
(RO)

Register 41.316. H264_DMA_IN_SUC_EOF_DES_ADDR_CH4_REG (0x0928)



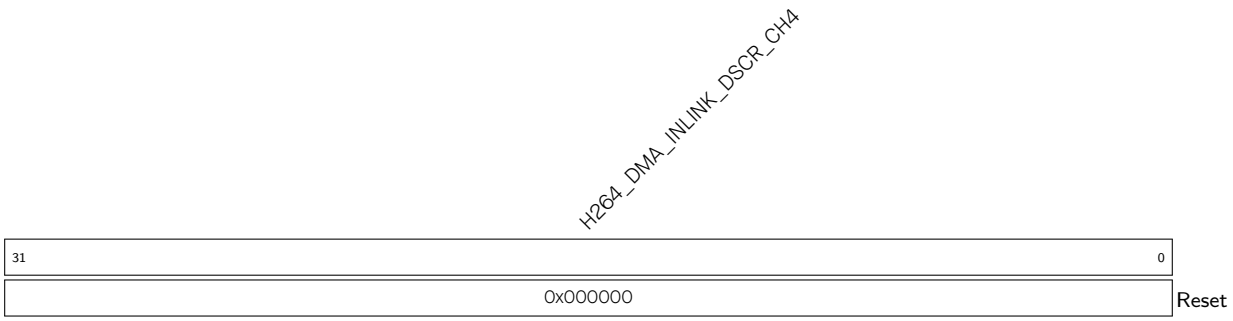
H264_DMA_IN_SUC_EOF_DES_ADDR_CH4 Represents the address of the inlink descriptor when the EOF bit in this descriptor is 1. (RO)

Register 41.317. H264_DMA_IN_ERR_EOF_DES_ADDR_CH4_REG (0x092C)



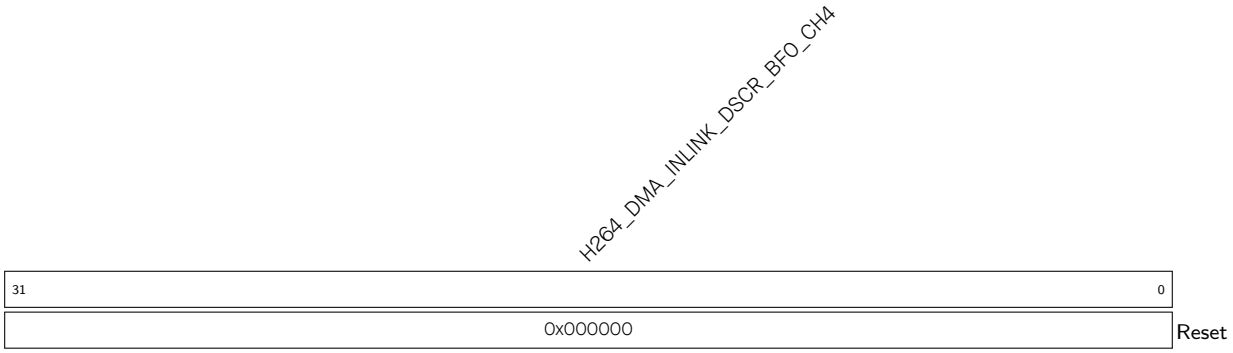
H264_DMA_IN_ERR_EOF_DES_ADDR_CH4 Represents the address of the inlink descriptor when there are errors during the current data reception. (RO)

Register 41.318. H264_DMA_IN_DSCR_CH4_REG (0x0930)



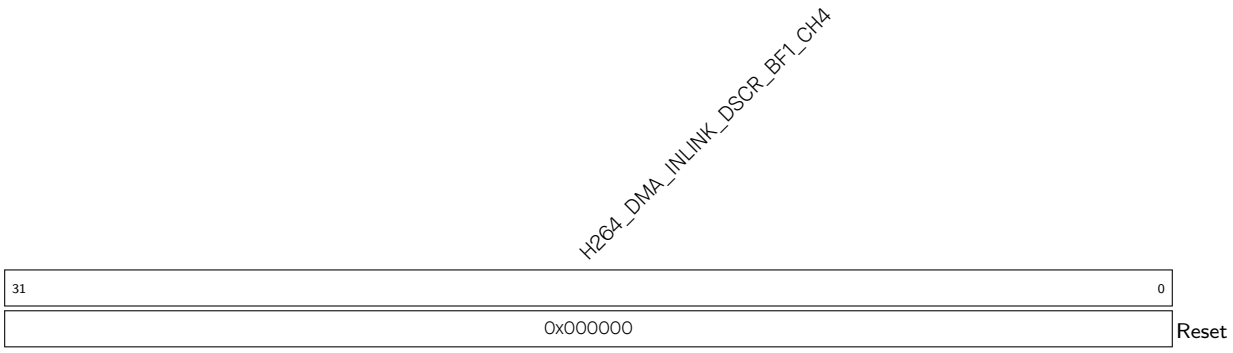
H264_DMA_INLINK_DSCR_CH4 Represents the address of the next descriptor x+1 pointed by the current descriptor that is pre-read. (RO)

Register 41.319. H264_DMA_IN_DSCR_BF0_CH4_REG (0x0934)



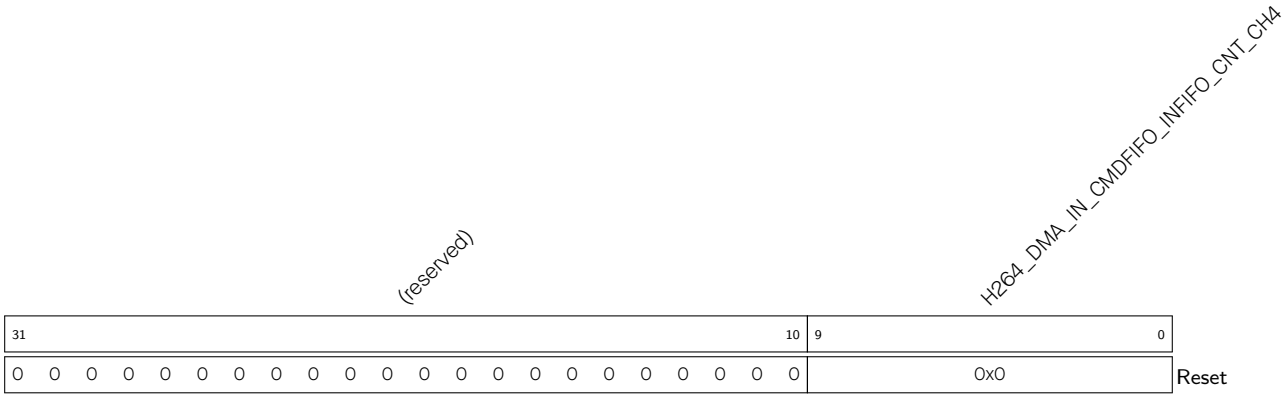
H264_DMA_INLINK_DSCR_BF0_CH4 Represents the address of the current descriptor x that is pre-read. (RO)

Register 41.320. H264_DMA_IN_DSCR_BF1_CH4_REG (0x0938)



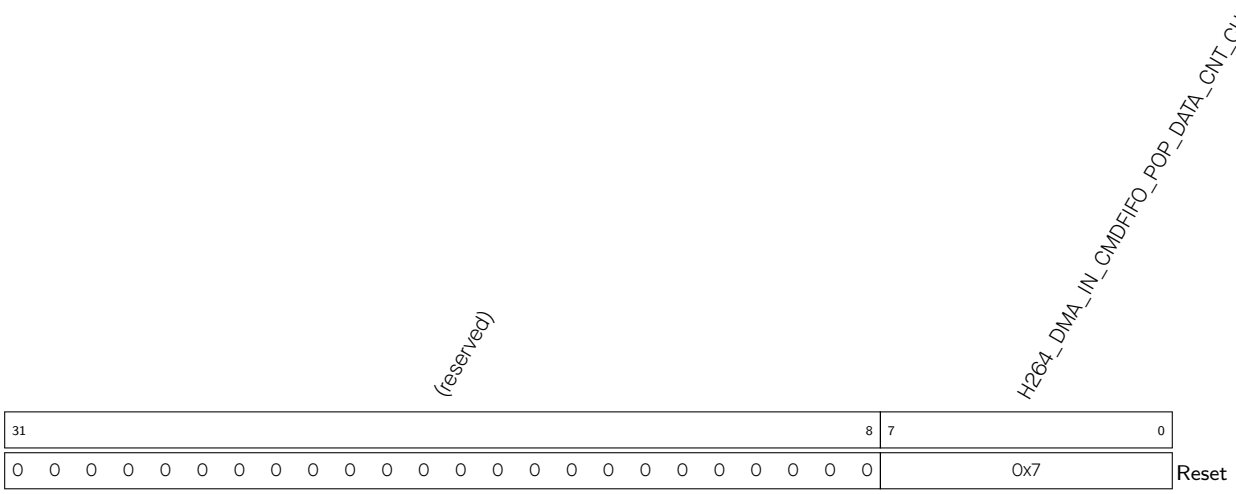
H264_DMA_INLINK_DSCR_BF1_CH4 Represents the address of the previous descriptor x-1 that is pre-read. (RO)

Register 41.321. H264_DMA_IN_FIFO_CNT_CH4_REG (0x0980)



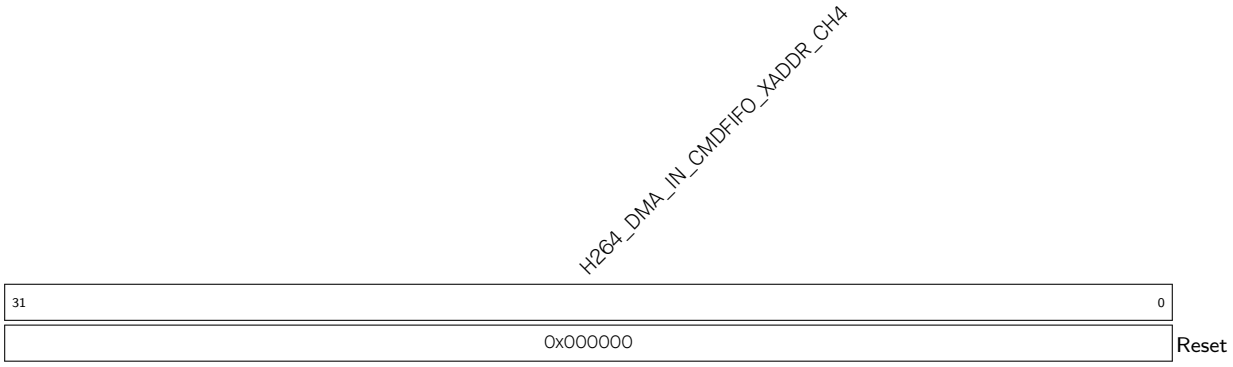
H264_DMA_IN_CMDFIFO_INFIFO_CNT_CH4 Represents the remaining request data in FIFO. (RO)

Register 41.322. H264_DMA_IN_POP_DATA_CNT_CH4_REG (0x0984)



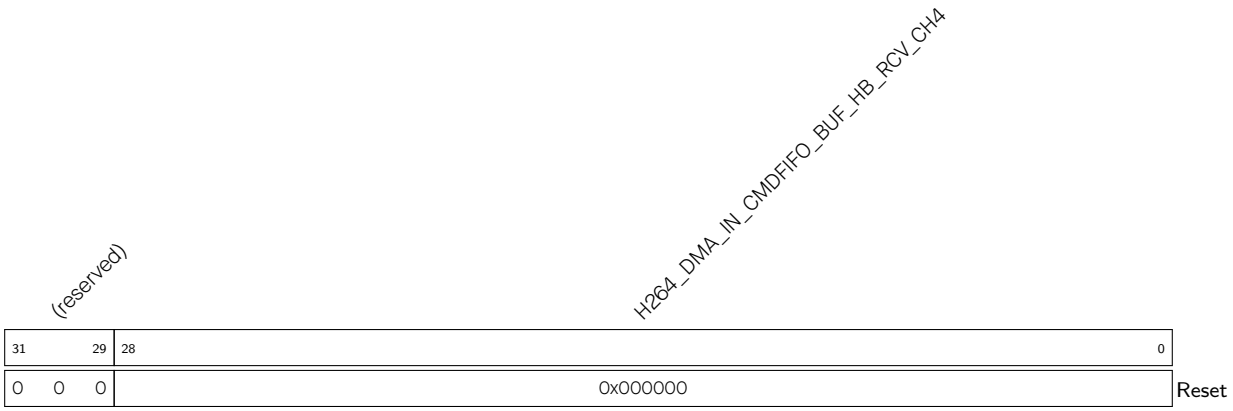
H264_DMA_IN_CMDFIFO_POP_DATA_CNT_CH4 Represents the data quantity in the current AXI request. (RO)

Register 41.323. H264_DMA_IN_XADDR_CH4_REG (0x0988)



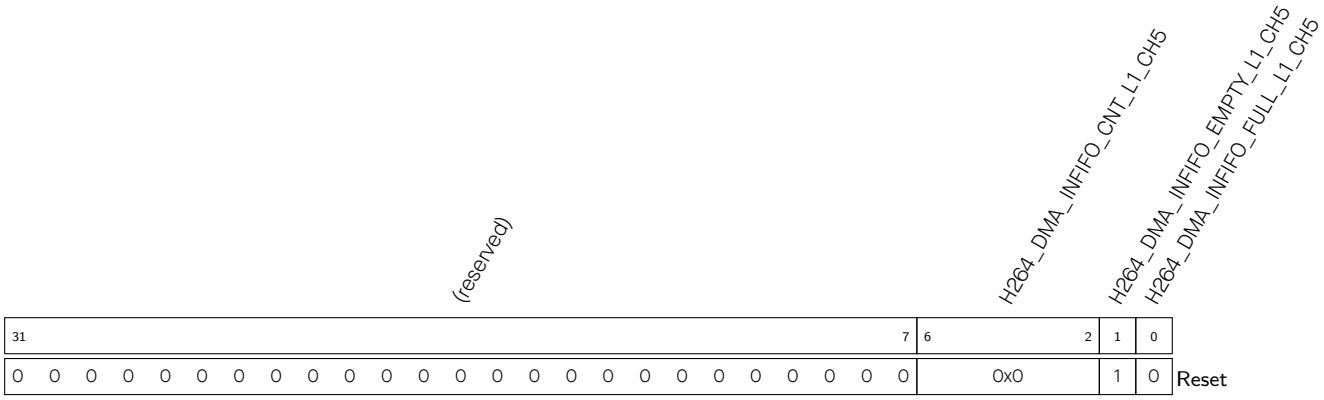
H264_DMA_IN_CMDFIFO_XADDR_CH4 Represents the address of the current AXI request. (RO)

Register 41.324. H264_DMA_IN_BUF_HB_RCV_CH4_REG (0x098C)



H264_DMA_IN_CMDFIFO_BUF_HB_RCV_CH4 Represents the data quantity that has been requested. (RO)

Register 41.325. H264_DMA_INFIFO_STATUS_CH5_REG (0x0A20)



H264_DMA_INFIFO_FULL_L1_CH5 Represents whether l1 FIFO of RX channel 5 is full.

0: Not Full

1: Full

(RO)

H264_DMA_INFIFO_EMPTY_L1_CH5 Represents whether l1 FIFO of RX channel 5 is empty.

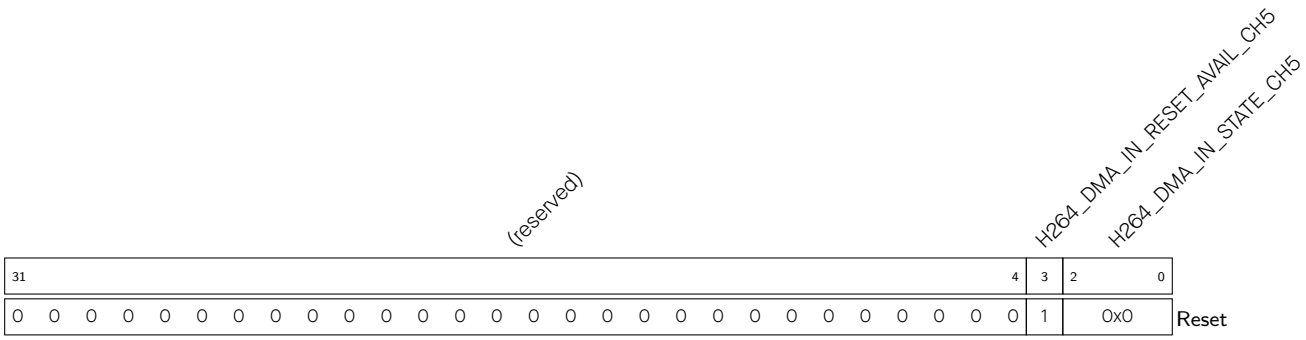
0: Not empty

1: Empty

(RO)

H264_DMA_INFIFO_CNT_L1_CH5 Represents the data quantity in l1 FIFO for RX channel 5. Measurement unit: byte. (RO)

Register 41.326. H264_DMA_IN_STATE_CH5_REG (0x0A28)



H264_DMA_IN_STATE_CH5 Represents the current control module state machine state for RX channel 5. (RO)

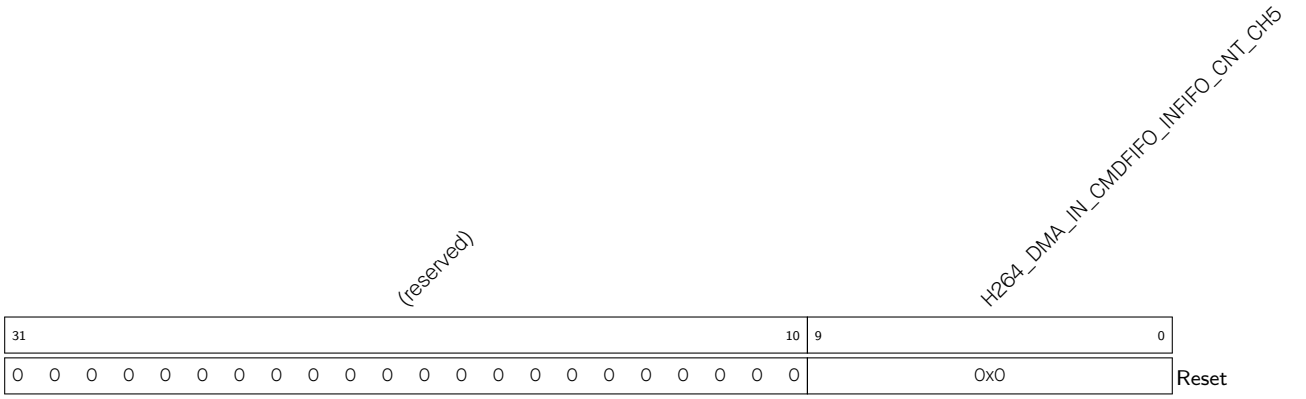
H264_DMA_IN_RESET_AVAIL_CH5 Represents whether it is safe to reset the channel.

0: Unsafe

1: Safe

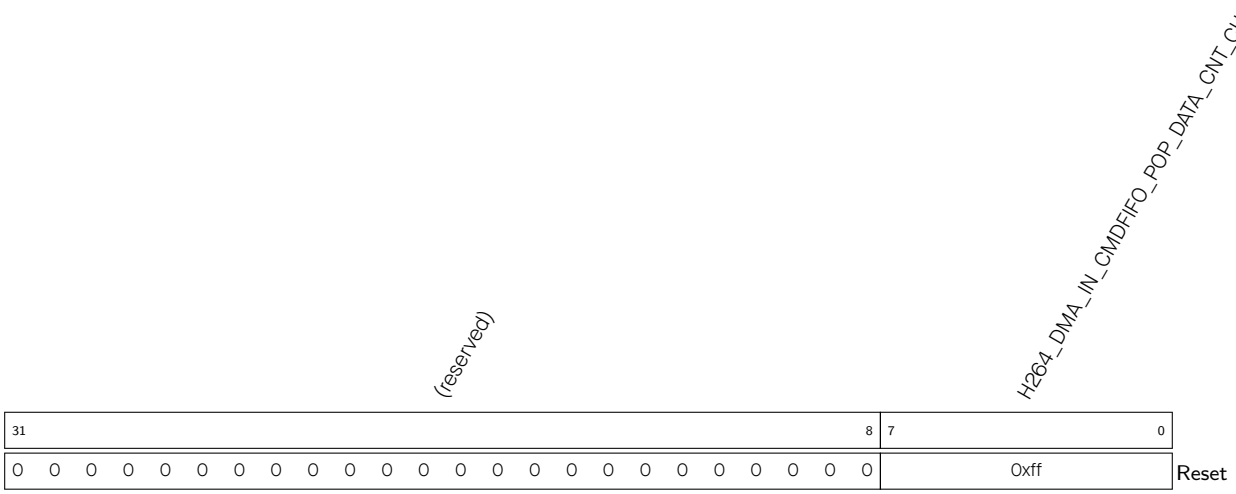
(RO)

Register 41.327. H264_DMA_IN_FIFO_CNT_CH5_REG (0x0A80)



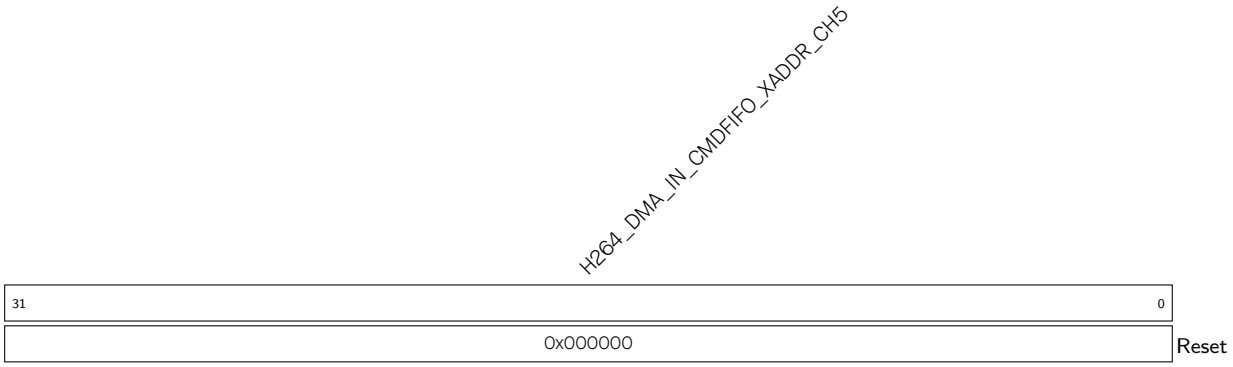
H264_DMA_IN_CMDFIFO_INFIFO_CNT_CH5 Represents the remaining request data in FIFO. (RO)

Register 41.328. H264_DMA_IN_POP_DATA_CNT_CH5_REG (0x0A84)



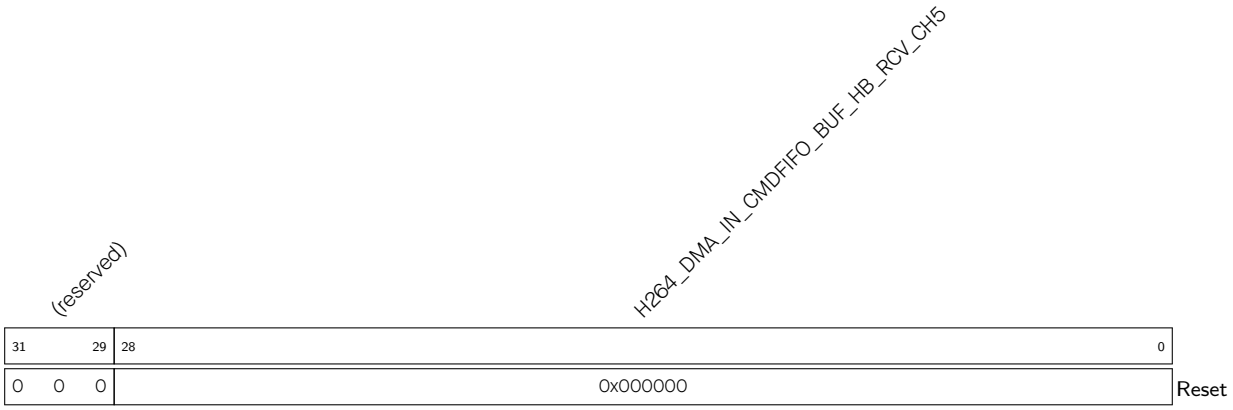
H264_DMA_IN_CMDFIFO_POP_DATA_CNT_CH5 Represents the data quantity in the current AXI request. (RO)

Register 41.329. H264_DMA_IN_XADDR_CH5_REG (0x0A88)



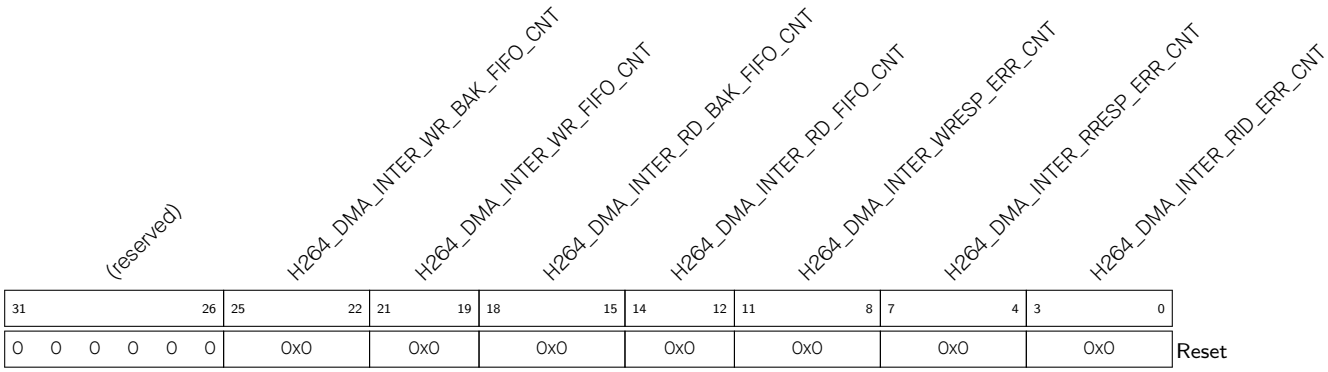
H264_DMA_IN_CMDFIFO_XADDR_CH5 Represents the address of the current AXI request. (RO)

Register 41.330. H264_DMA_IN_BUF_HB_RCV_CH5_REG (0x0A8C)



H264_DMA_IN_CMDFIFO_BUF_HB_RCV_CH5 Represents the data quantity that has been requested. (RO)

Register 41.331. H264_DMA_INTER_AXI_ERR_REG (0x0B00)



H264_DMA_INTER_RID_ERR_CNT Represents the count of errors in the internal memory ID for AXI read operations. (RO)

H264_DMA_INTER_RRESP_ERR_CNT Represents the count of errors in the response for internal memory during AXI read operations. (RO)

H264_DMA_INTER_WRESP_ERR_CNT Represents the count of AXI write internal memory resp err. (RO)

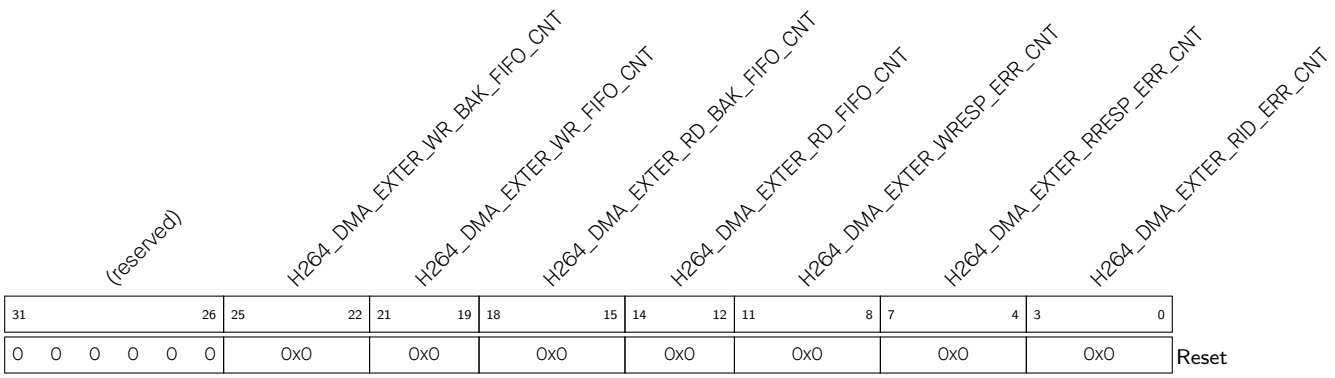
H264_DMA_INTER_RD_FIFO_CNT Represents the count of cmd FIFO remain cmd in AXI read internal memory module. (RO)

H264_DMA_INTER_RD_BAK_FIFO_CNT Represents the count of backup cmd FIFO remain cmd in AXI read internal memory module.(RO)

H264_DMA_INTER_WR_FIFO_CNT Represents the count of cmd FIFO remain cmd in AXI write internal memory module. (RO)

H264_DMA_INTER_WR_BAK_FIFO_CNT Represents the count of backup cmd FIFO remain cmd in AXI write internal memory module. (RO)

Register 41.332. H264_DMA_EXTER_AXI_ERR_REG (0x0B04)



H264_DMA_EXTER_RID_ERR_CNT Represents the count of errors in the external memory ID for AXI read. (RO)

H264_DMA_EXTER_RRESP_ERR_CNT Represents the count of errors in the response for external memory during AXI read operations. (RO)

H264_DMA_EXTER_WRESP_ERR_CNT Represents the count of AXI write external memory resp err. (RO)

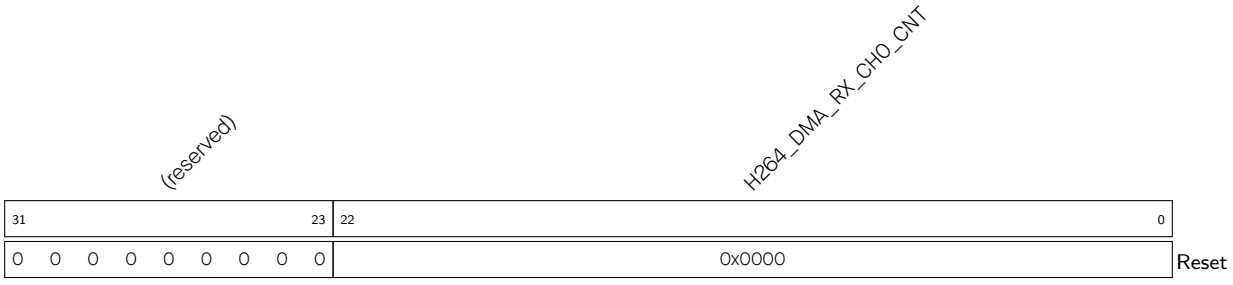
H264_DMA_EXTER_RD_FIFO_CNT Represents the count of cmd FIFO remain cmd in AXI read external memory module. (RO)

H264_DMA_EXTER_RD_BAK_FIFO_CNT Represents the count of backup cmd FIFO remain cmd in AXI read external memory module.(RO)

H264_DMA_EXTER_WR_FIFO_CNT Represents the count of cmd FIFO remain cmd in AXI write external memory module. (RO)

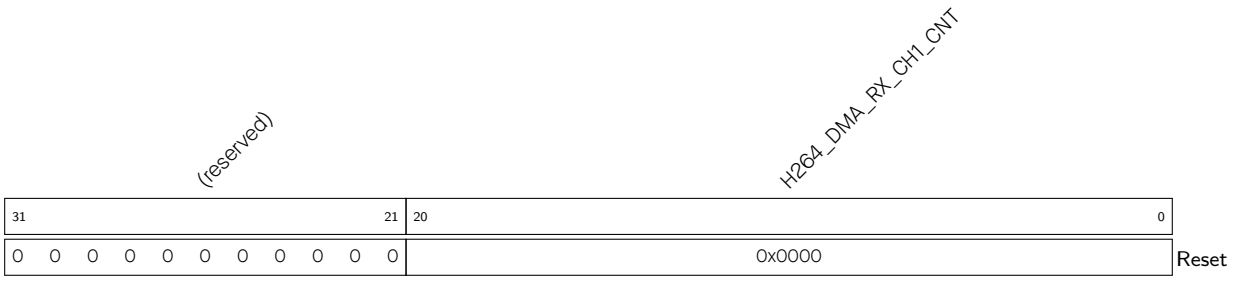
H264_DMA_EXTER_WR_BAK_FIFO_CNT Represents the count of backup cmd FIFO remain cmd in AXI write external memory module. (RO)

Register 41.333. H264_DMA_RX_CH0_COUNTER_REG (0x0B54)



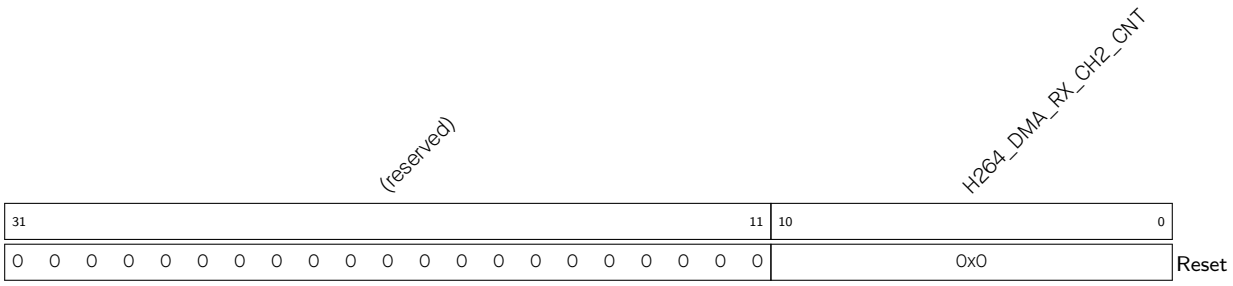
H264_DMA_RX_CH0_CNT Represents the RX channel 0 counter register. (RO)

Register 41.334. H264_DMA_RX_CH1_COUNTER_REG (0x0B58)



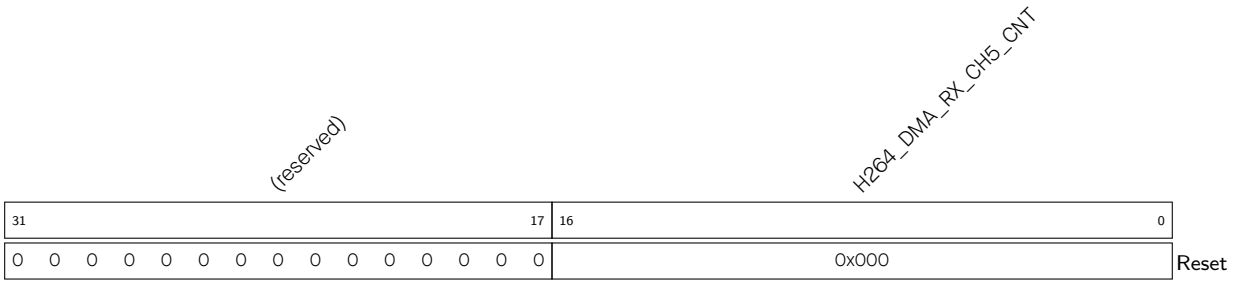
H264_DMA_RX_CH1_CNT Represents the RX channel 1 counter register. (RO)

Register 41.335. H264_DMA_RX_CH2_COUNTER_REG (0x0B5C)



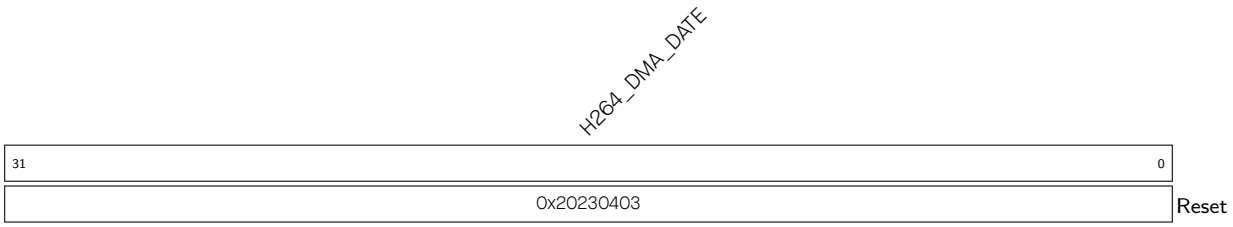
H264_DMA_RX_CH2_CNT Represents the RX channel 2 counter register. (RO)

Register 41.336. H264_DMA_RX_CH5_COUNTER_REG (0x0B60)



H264_DMA_RX_CH5_CNT Represents the RX channel 5 counter register (RO)

Register 41.337. H264_DMA_DATE_REG (0x0B3C)



H264_DMA_DATE Version control register. (R/W)

Chapter 42

MIPI CSI

42.1 Introduction

The MIPI CSI module of ESP32-P4 provides a standard interface with camera modules for mobile devices, such as smartphones, tablets, and other portable electronics. It connects cameras to application processors in mobile devices, enabling high-speed data transfer and efficient communication between the camera and the host system.

ESP32-P4 MIPI CSI implements a MIPI RX D-PHY and a MIPI CSI-2 host controller. It provides one clock lane and two data lanes, supporting a data transmission of up to 1.5 Gbps per lane to communicate with camera sensors compliant with the MIPI CSI-2 specification.

Note:
The MIPI CSI-2 specification defines the communication between image application processors and cameras. It is part of the communication protocols defined by MIPI Alliance standards, designed for chip-to-chip communication in mobile systems.

42.2 Terminology

This section covers terminology used to describe the functionality of MIPI CSI.

MIPI CSI-2 MIPI Alliance Specification for Camera Serial Interface 2. It receives camera data through the PHY Protocol Interface (PPI) from the MIPI RX D-PHY and outputs data through a 32-bit image interface to the Image Signal Processor (ISP).

MIPI D-PHY MIPI Alliance Specification for the source synchronous physical layer. It implements the physical layer of universal lanes for the MIPI D-PHY interface, which receives clock and data from MIPI camera sensors through one clock lane and two data lanes.

PPI PHY Protocol Interface defined in MIPI Alliance Specification. It connects the CSI-2 host controller and D-PHY.

ISP Image Signal Processor, used to process images received from the camera. For more details about ISP, please refer to Chapter 38 *Image Signal Processor (ISP)*.

Transmitter The device that transmits data, usually a MIPI camera.

Receiver The device that receives data, in this chapter, the ESP32-P4 chip.

Clock lane The clock lane of the MIPI D-PHY interface, used to transmit high-speed differential clock signals.

Data lane	The data lane of the MIPI D-PHY interface, used to transmit data and commands.
Line	The line of the data lane of the MIPI D-PHY interface; each data lane consists of two lines: P and N.
Sequence	A string of state codes received through the data lane, controlling the MIPI RX D-PHY to enter other modes from Control mode.
Control mode	One of the operating modes of the MIPI RX D-PHY, used to receive requests from the transmitter. All other modes return to this mode.
High-Speed Data Reception mode	One of the operating modes of the MIPI RX D-PHY, used to receive data in burst mode.
Escape mode	One of the operating modes of the MIPI RX D-PHY, used to support low-speed asynchronous communication on the data lane to maintain a low-power state.
Ultra Low Power State (ULPS)	One of the operating modes of the MIPI RX D-PHY, in which the clock lane and data lane are in idle state, maintaining the lowest power consumption except for the Shut-Down state.
Test code	Specific control or configuration codes used to manage and test the MIPI D-PHY, such as setting operational mode.

42.3 Feature List

MIPI CSI in ESP32-P4 supports the following features:

- MIPI RX D-PHY compliant with MIPI D-PHY interface specification revision 1.1
- MIPI CSI-2 host compliant with MIPI CSI-2 specification
- Two data lanes, each with a rate ranging from 80 Mbps to 1.5 Gbps
- Ultra Low Power State in Escape mode
- 32-bit image interface with ISP
- Various input image formats:
  - RGB888/RGB666/RGB565
  - YUV422/YUV420
  - RAW8/RAW10/RAW12
- Error detection and correction at PHY level, packet level, line level, and frame level
- Electromagnetic Interference (EMI) mitigation by data scrambling

Note:
ESP32-P4 does not support different virtual channels defined in the MIPI specification, so virtual channel numbers in the packet header are ignored.

42.4 Architectural Overview

Figure 42.4-1 shows the architecture of MIPI CSI and its connection to other system components. The MIPI CSI consists of the following components:

- **MIPI RX D-PHY**: Implements the physical link layer, including one clock lane (CP/CN) and two data lanes (D0P/D0N and D1P/D1N) to connect with cameras.
- **MIPI CSI-2 host**: Implements the CSI-2 protocol, receives camera data from D-PHY, and outputs through the Image Interface, which includes:
  - **Pipeline**: Registers PPI signals.
  - **Descrambler**: Converts PPI scrambled data to its original values when enabled.
  - **PHY adaptation layer**: Manages the PHY interface, including PHY error handling.
  - **Packet analyzer**: Processes received data from lanes, including data lane merging, header decoding, and various error detection and correction.
  - **Image interface**: Reorders pixels into 32-bit Image Interface data and generates timing-accurate video synchronization signals.
  - **Error management**: Monitors and notifies about error conditions on the CSI-2 link.
  - **Register bank**: Provides access to configuration and control registers.
- **PPI interface**: Connects the MIPI RX D-PHY and the MIPI CSI-2 host controller.
- **Test code interface**: Sends configuration details from the MIPI CSI-2 host controller to the MIPI RX D-PHY.

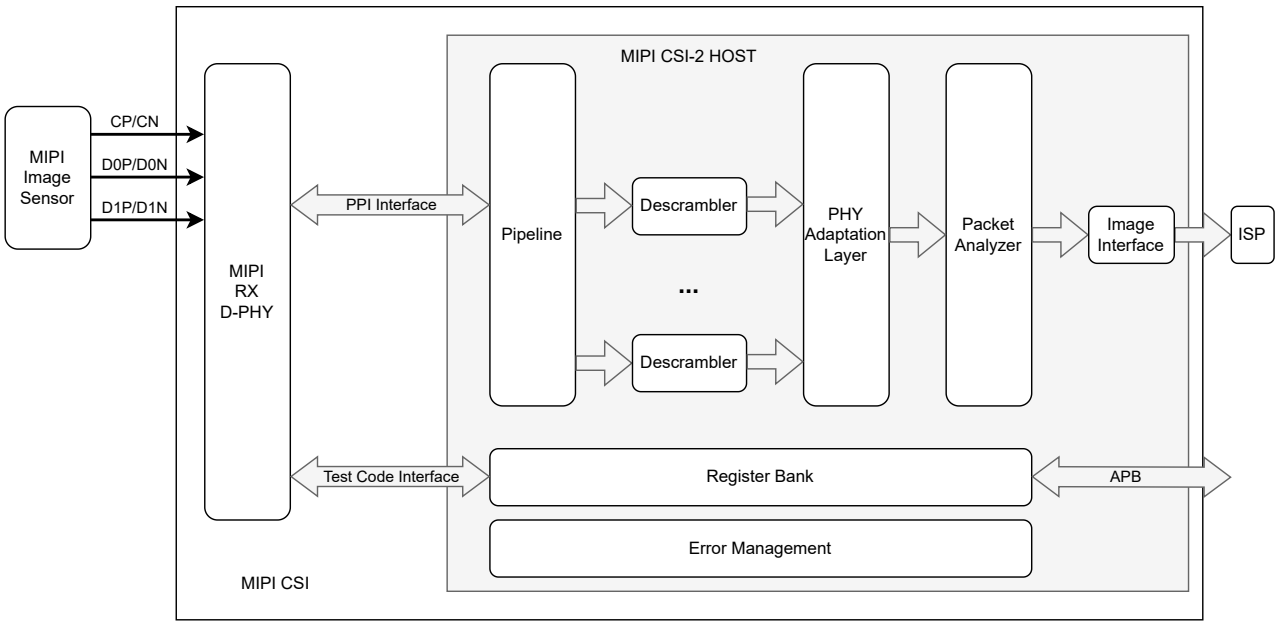


Figure 42.4-1. MIPI CSI Architecture

42.5 Functional Description

This section introduces the operating states and modes of the MIPI RX D-PHY and the major functions that facilitate the use of MIPI CSI. For detailed configurations, refer to Section 42.7 Programming Procedures.

42.5.1 Data Lane State

The functions of the transmitter determine the data lane state by driving certain line levels. During normal operation, either an HS-TX or an LP-TX is driving a lane.

- HS-TX: High-Speed transaction. An HS-TX always drives the lane differentially. It operates in two states: HS-0 and HS-1. When HS-TX drives the lane, the lane remains in High-Speed Data Reception mode.
  - HS-0: Transmits differential data 0 in High-Speed Data Reception mode.
  - HS-1: Transmits differential data 1 in High-Speed Data Reception mode.
- LP-TX: Low-Power transaction. The two LP-TXs drive the two lines of a lane independently and single-ended. It operates in four states: LP-00, LP-01, LP-10, and LP-11. When LP-TX drives the lane, the lane is in Control mode or Escape mode.
  - LP-00: A bridge state between other LP states (LP-01, LP-10, LP-11) in Control Mode or Escape Mode. When in Ultra Low Power State (ULPS), lanes are also in the LP-00 state.
  - LP-01: Transmits data 0 in Escape mode, or indicates entry into High-Speed Data Reception mode from Control mode.
    The complete sequence for entering High-Speed Data Reception mode is: LP-11 → LP-01 → LP-00
  - LP-10: Transmits data 1 in Escape mode, or indicates entry into Escape mode or Turnaround mode from Control mode.
    The complete sequence for entering Escape mode is: LP-11 → LP-10 → LP-00 → LP-01 → LP-00.
    The complete sequence for entering the Turnaround mode is: LP-11 → LP-10 → LP-00 → LP-10 → LP-00
  - LP-11: The stop state, i.e., the default idle state of a lane.

Note:
The Turnaround mode is used for switching the data transmission direction. This mode is not used in the ESP32-P4 MIPI CSI, and will not be further explained in this chapter.

Figure 42.5-1 shows the lane line levels for High-Speed (HS) and Low-Power (LP) transactions.

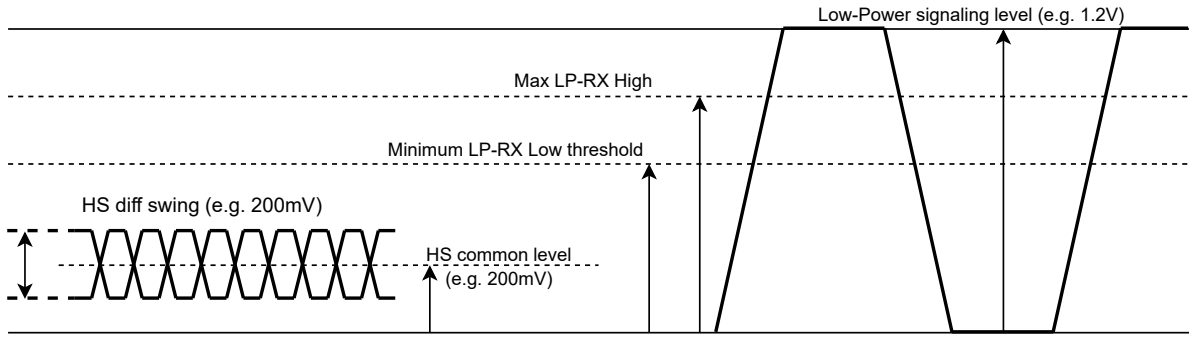


Figure 42.5-1. Level of MIPI Data Lane Lines

Table 42.5-1 shows the available line states.

Table 42.5-1. Lane State Descriptions

State Code	DP Line	DN Line	Corresponding State in Different Modes		
			High-Speed Data Reception Mode	Control Mode	Escape Mode
HS-0	HS Low	HS High	0	N/A	N/A
HS-1	HS High	HS Low	1	N/A	N/A
LP-00	LP Low	LP Low	N/A	Bridge	Space
LP-01	LP Low	LP High	N/A	HS-Rqst	0
LP-10	LP High	LP Low	N/A	LP-Rqst	1
LP-11	LP High	LP High	N/A	Stop	N/A

42.5.2 MIPI RX D-PHY Operation

Figure 42.5-2 illustrates the various states and modes of the MIPI RX D-PHY during initialization and active state. Detailed descriptions of each states and modes are presented in the following sections.

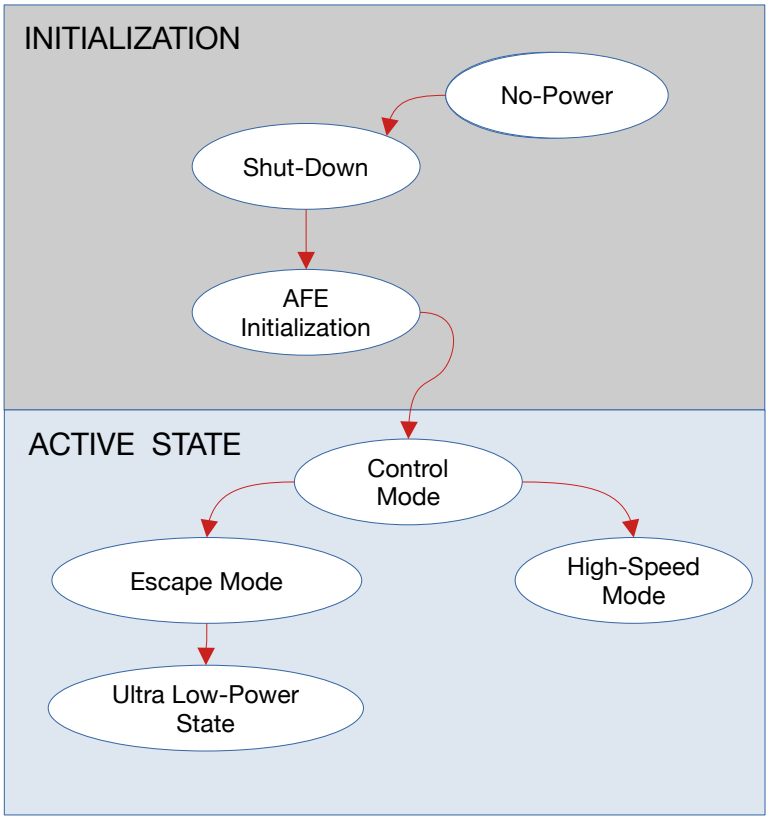


Figure 42.5-2. MIPI RX D-PHY Operating States and Modes

42.5.2.1 No-Power State

In the No-Power state, no supply voltage is applied to the PHY.

The MIPI RX D-PHY of MIPI CSI is in No-Power state when the power supply VDD_HP_*n* (n: 0-3) and VDD_MIPI_DPHY are off.

42.5.2.2 Shut-Down State

Shut-Down state refers to the idle state.

In Shut-Down state, the PHY disables all analog blocks, resets all digital logic, and sets the DOP/DON, D1P/D1N, and CP/CN lanes to high impedance (Hi-Z). Activating the Shut-Down state ensures the lowest power consumption. To enable the Shut-Down state of the PHY, set CSI_HOST_DPHY_RSTZ and CSI_HOST_PHY_SHUTDOWNZ to 0.

Before leaving the Shut-Down state, configure several parameters, including the lane number to be used and the lane operation frequency range.

The MIPI RX D-PHY consists of two data lanes. Configure CSI_HOST_N_LANES to select either one or both data lanes. These configurations should remain static or stable before exiting the Shut-Down state.

Depending on the specific requirements for the MIPI RX D-PHY, additional configuration steps might be necessary. By default, the PHY operates within a lower range of 80-110 Mbps. To accommodate higher bit rates, adjust the HS frequency ranges (the hsfreqrange field) in test code HS RX Control of lane 0 with the appropriate code. It is recommended to perform this adjustment in the Shut-Down state, as the control interface is independent of the rest of the MIPI RX D-PHY.

The different ranges supported are shown in Table 42.5-2. The hsfreqrange field indicates the HS operating frequency range and is accessible through test code 0x44. For detailed configuration, please refer to Section *42.5.3 MIPI RX D-PHY Configuration*.

Table 42.5-2. Frequency Ranges Supported by MIPI RX D-PHY

Frequency Range (Mbps)	hsfreqrange[5:0]
80-89	000000
90-99	010000
100-109	100000
110-129	000001
130-139	010001
140-149	100001
150-169	000010
170-179	010010
180-199	100010
200-219	000011
220-239	010011
240-249	100011
250-269	000100
270-299	010100
300-329	000101
330-359	010101
360-399	100101
400-449	000110
450-499	010110
500-549	000111
550-599	010111
600-649	001000
650-699	011000
700-749	001001
750-799	011001
800-849	101001
850-899	111001
900-949	001010
950-999	011010
1000-1049	101010
1050-1099	111010
1100-1149	001011
1150-1199	011011
1200-1249	101011
1250-1299	111011
1300-1349	001100
1350-1399	011100
1400-1449	101100

Frequency Range (Mbps)	hsfreqrange[5:0]
1450-1500	111100

When CSI_HOST_DPHY_RSTZ and CSI_HOST_PHY_SHUTDOWNZ are set to 1, the MIPI RX D-PHY leaves the Shut-Down state and starts the initialization procedure.

42.5.2.3 AFE Initialization

After releasing CSI_HOST_DPHY_RSTZ and CSI_HOST_PHY_SHUTDOWNZ, the PHY begins an initialization sequence for normal operation. It is recommended to release CSI_HOST_PHY_SHUTDOWNZ before CSI_HOST_DPHY_RSTZ. Additionally, ensure that CSI_DPHY_CFG_CLK (refer to Table 11.2-5 in Chapter 11 *Reset and Clock* for details) is available and stable at this point.

The initialization sequence includes several steps, such as enabling internal blocks and performing internal calibrations. Once completed, control transfers to the lanes, which manage power for LP/HS requests from the transmitter by enabling or disabling the corresponding receivers.

Set CSI_HOST_PHY_STOPSTATEDATA_*n* and CSI_HOST_PHY_STOPSTATECLK to 1 to perform all initialization steps. In this state, both clock and data lanes are in stop state (LP-11).

The initialization period (TINIT) is a protocol-dependent parameter with a minimum duration of 100 μs, as defined by the specification. After setting CSI_HOST_PHY_STOPSTATEDATA_*n* and CSI_HOST_PHY_STOPSTATECLK to 1, the software should wait more than 100 μs before starting camera data reception.

42.5.2.4 Control Mode

After completing AFE initialization, the MIPI RX D-PHY remains in Control mode by default. It receives the LP-11 state code (stop state) on the lines until a request is made. Upon receiving a request, the PHY enters either High-Speed Data Reception mode, Escape mode, or Ultra Low Power State. All requests must begin and end with the lanes in the stop state.

42.5.2.5 High-Speed Data Reception Mode

In High-Speed Data Reception mode, the PHY receives data in bursts. The lane operates in high-speed mode only during these bursts.

As shown in Figure 42.5-3, a burst consists of three parts: the low-power initialization sequence, the high-speed data payload, and the end of reception sequence.

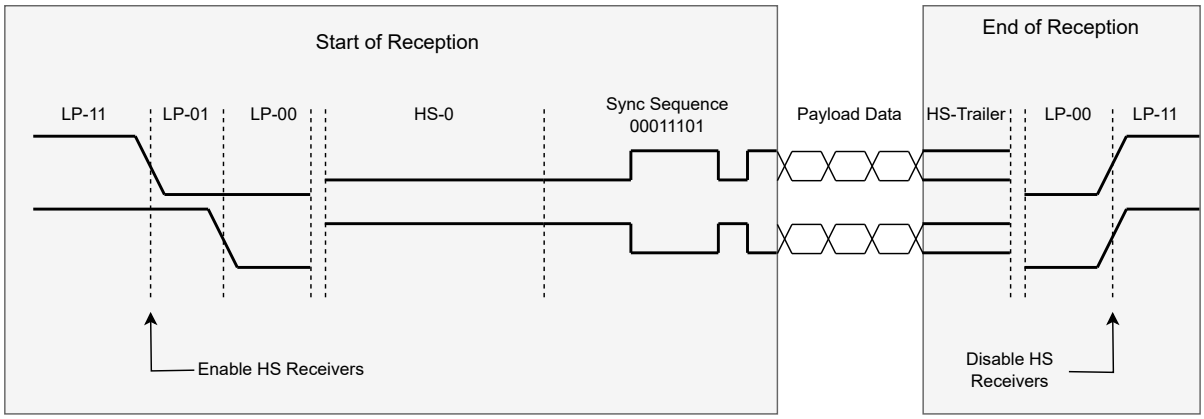


Figure 42.5-3. MIPI CSI HS Data Reception Sequence

The receiver enters high-speed mode following the low-power sequence of [LP-11 > LP-01 > LP-00]. Synchronization is achieved through a sync sequence of “00011101”. After synchronization, the PHY receives high-speed data until a transition from LP-00 to LP-11 is detected on the lane.

42.5.2.6 Escape Mode

Escape mode allows asynchronous communication using data lanes at low speed. During Escape mode, the lane remains in low-power state. A data lane enters this mode after Escape mode request sequence [LP-11 > LP-10 > LP-00 > LP-01 > LP-00]. If an LP-11 state code is detected before the lane reaches LP-00, the request will be aborted, and the lane will return to stop state. Upon entering Escape mode, the transmitter sends an 8-bit command to indicate a requested action.

MIPI CSI only supports Ultra Low Power State (ULPS) in escape mode. The 8-bit command to enter ULPS is “b00011110”. If the entry command is not valid, it will be ignored, and the receiver will wait until the transmitter returns to the stop state.

Ultra Low Power State

Ultra Low Power State involves the lowest power consumption, except for the Shut-Down state.

For data lanes, this mode is activated by sending an Ultra Low Power State entry command “00011110”. After that, the clock lane and data lanes enter the space state (LP-00).

Figure 42.5-4 shows the ULPS sequences for clock and data lanes.

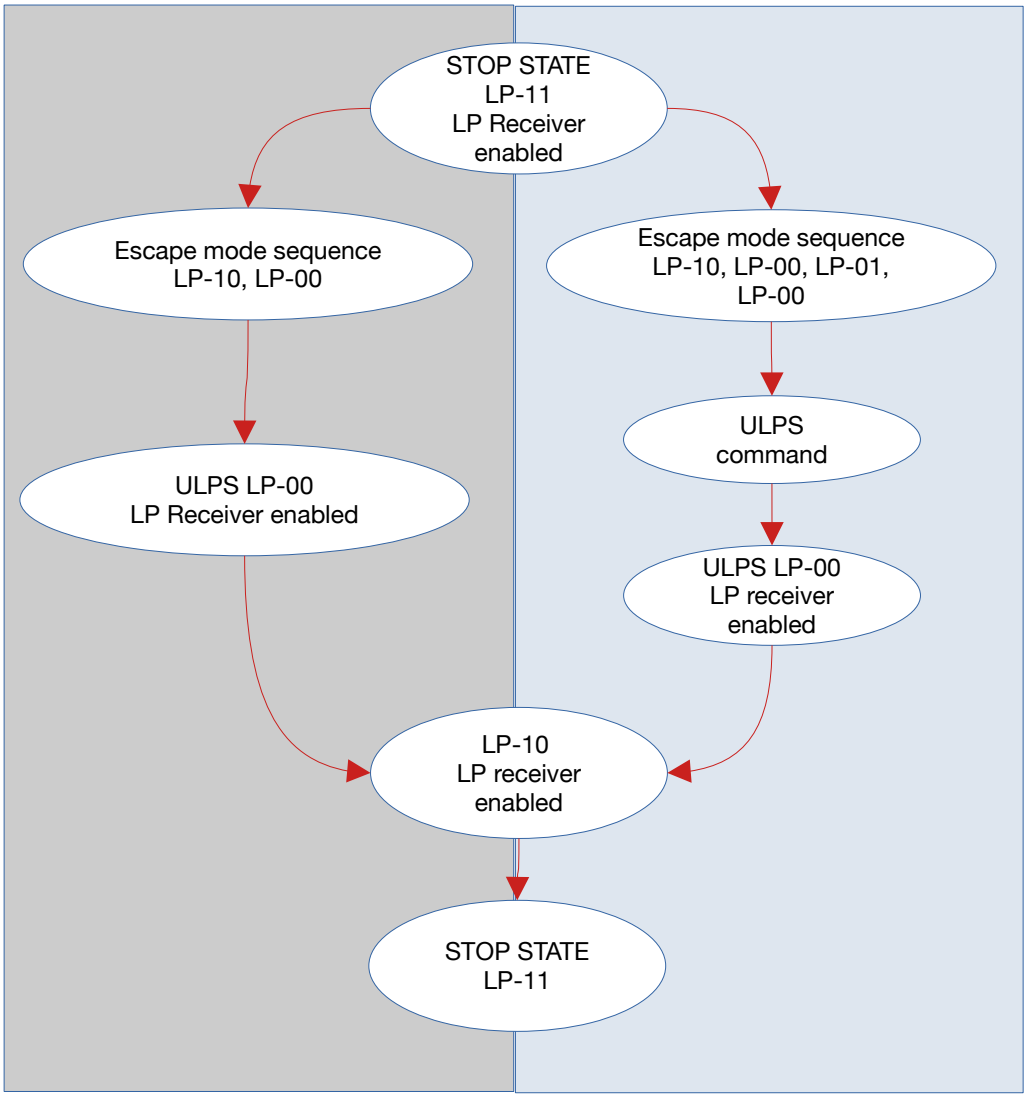


Figure 42.5-4. ULPS Sequences for Clock and Data Lanes

42.5.3 MIPI RX D-PHY Configuration

The MIPI RX D-PHY includes test codes for configuring MIPI RX D-PHY operation. Refer to Section 42.5.3.2 for all supported test codes.

42.5.3.1 MIPI RX D-PHY Programming Interface

The MIPI RX D-PHY test code programming involves two steps: first, program the test code into the registers; second, provide the test data to the tester's inputs.

Before configuring a test code, perform the following steps:

- Write 0 to CSI_HOST_PHY_SHUTDOWNZ and CSI_HOST_DPHY_RSTZ to shut down and reset the PHY.
- It is also recommended to set CSI_HOST_PHY_TESTCLR to 1 to apply a reset pulse before the first configuration of any test code.

Next, configure the test code by following these steps:

- Set the test code.

The test code loads when CSI_HOST_PHY_TESTEN is set to 1 and with the falling edge on CSI_HOST_PHY_TESTCLK.

- Ensure that CSI_HOST_PHY_TESTCLK is set to 1.
- Set CSI_HOST_PHY_TESTDIN to the 8-bit test code.
- Set CSI_HOST_PHY_TESTEN to 1.
- Set CSI_HOST_PHY_TESTCLK to 0. The falling edge on CSI_HOST_PHY_TESTCLK latches the value of CSI_HOST_PHY_TESTDIN[7:0] as the current test code.
- Set CSI_HOST_PHY_TESTEN to 0.

- Enter the test data.

  The test data programs to the latest loaded test code when CSI_HOST_PHY_TESTEN is 0 and with the rising edge on CSI_HOST_PHY_TESTCLK.

  - Set CSI_HOST_PHY_TESTCLK to 0, if not done already.
  - Set CSI_HOST_PHY_TESTDIN to the 8-bit test data.
  - Set CSI_HOST_PHY_TESTCLK to 1. The rising edge on CSI_HOST_PHY_TESTCLK programs the test data internally.
  - Repeat the above steps to add more test data for the same test code.

Set CSI_HOST_PHY_TESTCLR to 1 to reset a test code. This is necessary only before the first programming operation or if you wish to reset the PHY's configuration to its default value.

Figure 42.5-5 shows a timing diagram for the MIPI RX D-PHY control interface. After programming a test code, CSI_HOST_PHY_TESTDOUT asynchronously outputs the relevant data associated with the specific test code. This data may include read-back information or other meaningful signals.

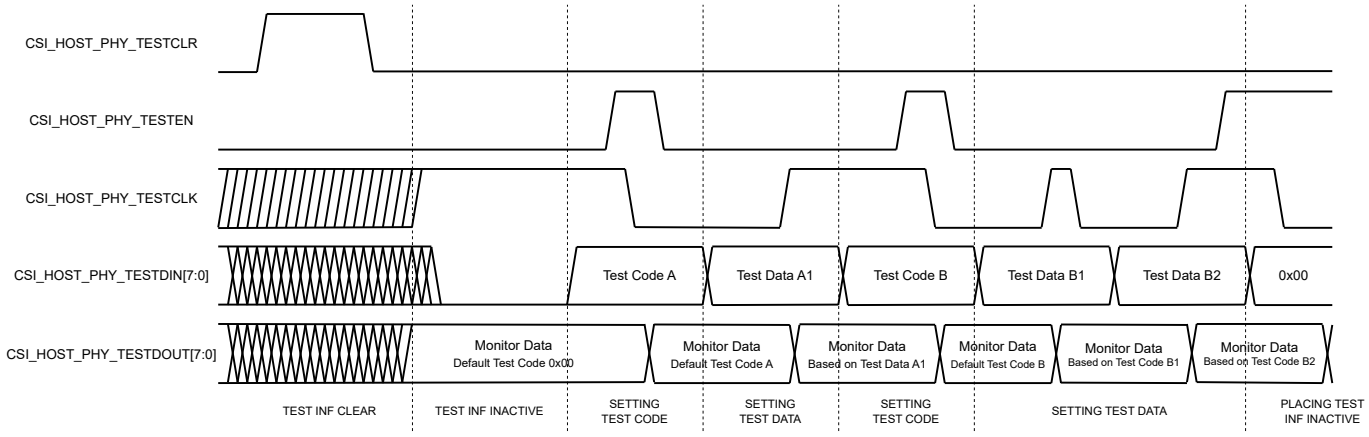


Figure 42.5-5. Control Interface Timing Diagram

Note:

Some test codes require two write data operations. In the first operation, the value of CSI_HOST_PHY_TESTCLK changes from 1 to 0. Ensure that the falling edge in the clock does not occur when CSI_HOST_PHY_TESTEN is set to 1. Otherwise, the current test data will be latched as an erroneous test code.

It is highly recommended to set the test interface to an inactive state by programming the test code 0x00 when not

used, as shown in the final sequence in Figure 42.5-5.

42.5.3.2 MIPI RX D-PHY Test Code

The MIPI RX D-PHY is configured through test codes via CSI_HOST_PHY_TEST_CTRL0_REG and CSI_HOST_PHY_TEST_CTRL1_REG. Refer to Section 42.5.3.1 *MIPI RX D-PHY Programming Interface* for more details.

Normal Operation

Use the following test code to enable normal operation of the PHY and the wake-up state of the test interface. In this mode, the test interface is inactive.

- **Test Code**: 0x00
- **Test Data**: None

HS RX Control of Lane 0

Use the following test code to select the HS operating frequency range (hsfreqrange).

- **Test Code**: 0x44
- **Test Data**:

Table 42.5-3. Test Data in HS RX Control of Lane 0

w-1'b0	w-6'b0	w-1'b0
Program Selector	HS operating frequency range selection (hsfreqrange)	Reserved

When Program Selector is 0, Test Data[6:1] programs hsfreqrange.

- **Test Dout:**

Table 42.5-4. Test Dout in HS RX Control of Lane 0

r-1'b0	r-6'b0	r-1'b0
1'b0	HS operating frequency range selection (hsfreqrange)	1'b0

When CSI_HOST_PHY_TESTDIN[7] is 0, Test Dout[6:1] reflects the value of the programmed hsfreqrange.

Others

Other test codes are reserved.

42.5.4 Descrambler

Data scrambling uses randomization techniques to mitigate electromagnetic interference (EMI) and radio frequency (RF) self-interference. This spreads the transmission energy of the link across a wider frequency band.

The transmitted data is scrambled with a Pseudo-Random-Bit-Stream (PRBS) to reduce the likelihood of repetitive patterns on a lane basis. The PRBS is generated using a Linear Feedback Shift Register (LFSR) that implements the generator polynomial:

$$G(x) = x^{16} + x^5 + x^4 + x^3 + 1 \tag{42.1}$$

Each data lane has an instance of the de-scrambling block. Set CSI_HOST_SCRAMBLE_ENABLE to 1 to enable this block. Each lane can be configured with a different 16-bit LFSR seed value via CSI_HOST_SCRAMBLE_SEED_LANE*n*.

The default seed values after resetting the controller are listed in 42.5-5.

Table 42.5-5. Scrambler PRBS Initial Seed Values for Lane 1 and Lane 2

Lane	Initial Seed Value
1	0x1008
2	0x1188

42.5.5 Error Management

The MIPI CSI-2 host analyzes received packets to identify protocol errors. Possible errors are listed in Table 42.5-6. These errors serve as interrupt sources and can generate the CSI_INTR signal. See Section 42.6 *Interrupts* for more details.

Table 42.5-6. Detectable Errors by MIPI CSI-2 Host

Error/Interrupt Source	Description	Corresponding Register Field
PHY_ERRSOTSYNCHS_*n* (*n*: 0-1)	Start-of-transmission error (no synchronization achieved) on data lane *n*	CSI_HOST_ST_PHY_ERRSOTSYNCHS_*n* (*n*: 0-1)
ERR_ECC_DOUBLE	Header ECC contains at least two errors (unrecoverable)	CSI_HOST_ST_ERR_ECC_DOUBLE
SHORTER_PAYLOAD	Reported word count exceeds received word count (unrecoverable)	CSI_HOST_ST_SHORTER_PAYLOAD
ERR_F_BNDRY_MATCH_VC*n* (*n*: 0-15)	Error matching frame start with frame end for virtual channel *n*	CSI_HOST_ST_ERR_F_BNDRY_MATCH_VC*n* (*n*: 0-15)
ERR_F_SEQ_VC*n* (*n*: 0-15)	Incorrect frame sequence detected on virtual channel *n*	CSI_HOST_ST_ERR_F_SEQ_VC*n* (*n*: 0-15)
ERR_FRAME_DATA_VC*n* (*n*: 0-15)	At least one CRC error in the last received frame on virtual channel *n*	CSI_HOST_ST_ERR_FRAME_DATA_VC*n* (*n*: 0-15)
ERR_CRC_VC*n* (*n*: 0-15)	Payload CRC error detected on virtual channel *n*	CSI_HOST_ST_ERR_CRC_VC*n* (*n*: 0-15)
ERR_ID_VC*n* (*n*: 0-15)	Unrecognized or unimplemented data type detected on virtual channel *n*	CSI_HOST_ST_ERR_ID_VC*n* (*n*: 0-15)

ERR_ECC_CORRECTED_VC*n* (*n*: 0-15)	Checksum error detected on virtual channel *n*	CSI_HOST_ST_ERR_ECC_CORRECTED_VC*n* (*n*: 0-15)
PHY_ERRSOTHS_*n* (*n*: 0-1)	Start-of-transmission error (synchronization can still be achieved) on data lane *n*	CSI_HOST_ST_PHY_ERRSOTHS_*n* (*n*: 0-1)
PHY_ERRESC_*n* (*n*: 0-1)	Escape entry error (ULPS) on data lane *n*	CSI_HOST_ST_PHY_ERRESC_*n* (*n*: 0-1)

Note:
MIPI CSI does not support virtual channel interleaving. Before starting transmission, verify which virtual channel is in use. Additionally, read error information from the specified virtual channel to detect any errors.

A reset is recommended for all errors in "*_FATAL_REG" registers.

42.6 Interrupts

ESP32-P4's MIPI CSI can generate the CSI_INTR interrupt signal that will be sent to the Interrupt Matrix.

The CSI_HOST_INT_ST_<group> registers report error conditions and trigger the CSI_INTR interrupt signal.

You can mask the interrupt pin by using the CSI_HOST_INT_MSK_<group> registers. All errors are masked by default. Set any bit of these registers to 1 to enable the specific error interrupt. The error asserts the respective bit in the CSI_HOST_INT_ST_<group> register. The CSI_HOST_INT_ST_<group> registers clear after a read operation.

Setting any bit of the Interrupt Force registers (CSI_HOST_INT_FORCE_<group>) to 1 triggers the corresponding interrupt without needing to activate the error conditions. These registers are for test purposes.

Figure42.6-1 shows the interrupt mechanism:

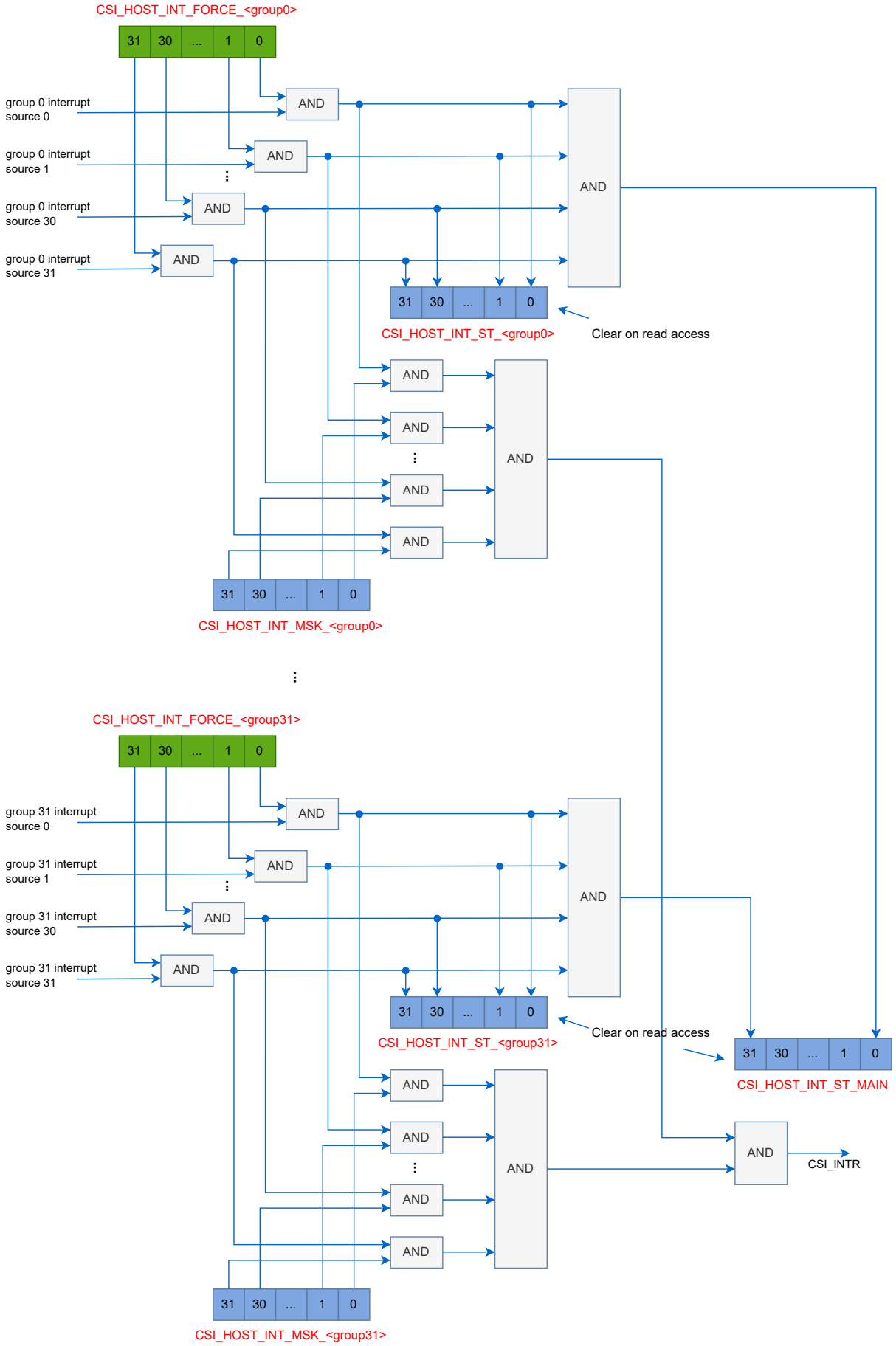


Figure 42.6-1. MIPI CSI Interrupt Mechanism

The interrupt signal CSI_INTR can be generated by the following internal interrupt sources (see Table 42.5-6 *Detectable Errors by MIPI CSI-2 Host* for details interrupt source description):

- PHY Fatal Interrupt:
  - PHY_ERRSOTSYNCHS_*n* (*n*: 0-1)
- Packet Fatal Interrupt:
  - ERR_ECC_DOUBLE
  - SHORTER_PAYLOAD
- Boundary Frame Fatal Interrupt:
  - ERR_F_BNDRY_MATCH_VC*n* (*n*: 0-15)
- Sequence Frame Fatal Interrupt:
  - ERR_F_SEQ_VC*n* (*n*: 0-15)
- CRC Frame Fatal Interrupt:
  - ERR_FRAME_DATA_VC*n* (*n*: 0-15)
- Payload CRC Fatal Interrupt:
  - ERR_CRC_VC*n* (*n*: 0-15)
- Data ID Interrupt:
  - ERR_ID_VC*n* (*n*: 0-15)
- ECC Corrected Interrupt:
  - ERR_ECC_CORRECTED_VC*n* (*n*: 0-15)
- PHY Interrupt:
  - PHY_ERRSOTHS_*n* (*n*: 0-1)
  - PHY_ERRESC_*n* (*n*: 0-1)

Note:
For definitions of *interrupt*, *interrupt signal*, *interrupt source*, and their correlations, please refer to Chapter 13 *Interrupt Matrix* > Section 13.2 *Interrupt Terminology in ESP32-P4*.

42.7 Programming Procedures

This section describes how to configure the MIPI RX D-PHY and the MIPI CSI-2 host for image data reception. MIPI CSI receives image data only in high-speed data reception mode.

42.7.1 Start High-Speed Data Reception

42.7.1.1 Clock and Reset

- Set HP_SYS_CLKRST_CSI_HOST_SYS_CLK_EN to 1 to enable the MIPI CSI clock.

- Set HP_SYS_CLKRST_RST_EN_CSI_HOST to 0 to release the MIPI CSI reset.
- Configure HP_SYS_CLKRST_MIPI_CSI_DPHY_CLK_SRC_SEL to select a proper clock source for the MIPI RX D-PHY configuration clock.
- Set HP_SYS_CLKRST_MIPI_CSI_DPHY_CFG_CLK_EN to 1 to enable the MIPI RX D-PHY configuration clock.
- Set HP_SYS_CLKRST_RST_EN_CSI_BRG to 1 to release the CSI_Bridge module of the ISP and reset the ISP.

42.7.1.2 MIPI RX D-PHY and MIPI Host Initialization

- Clear CSI_HOST_CSI2_RESETN to reset the CSI_HOST controller logic.
- Set CSI_HOST_PHY_TESTCLR to 1, then clear CSI_HOST_PHY_TESTCLR to reset the test code logic.
- Configure the MIPI RX D-PHY frequency range in test code 0x44 using the MIPI RX D-PHY programming interface.
- Configure CSI_HOST_N_LANES to define the number of active lanes in the system.
- Set CSI_HOST_PHY_SHUTDOWNZ and CSI_HOST_DPHY_RSTZ to 1 to release the MIPI RX D-PHY from the reset state.
- Set the corresponding bit in the CSI_HOST_INT_MSK_*_REG register to 1 to enable the error interrupt.
- Configure the descrambler function if needed.
- Set CSI_HOST_CSI2_RESETN to 1 to release the CSI_HOST reset.
- Wait until CSI_HOST_PHY_STOPSTATEDATA_*n* and CSI_HOST_PHY_STOPSTATECLK are 1.
- Start the camera image capture.

42.7.2 Stop High-Speed Data Reception

- Disable related channels in . Wait until is disabled. Refer to Chapter 7 *VDMA Controller (VDMA)*.
- Reset the ISP. Refer to Chapter 38 *Image Signal Processor (ISP)*.
- Set HP_SYS_CLKRST_RST_EN_CSI_HOST to 1 to reset the MIPI CSI.

42.8 Register Summary

The addresses in this section are relative to CSI Host base address provided in Table 9.3-2 in Chapter 9 *System and Memory*.

The abbreviations given in Column **Access** are explained in Section *Access Types for Registers*.

Name	Description	Address	Access
Version Register			
CSI_HOST_VERSION_REG	CSI_HOST version control register	0x0000	RO
Configuration Registers			
CSI_HOST_N_LANES_REG	Active lane number configuration register	0x0004	R/W
CSI_HOST_CSI2_RESETN_REG	CSI_HOST reset control register	0x0008	R/W
CSI_HOST_PHY_SHUTDOWNZ_REG	RX D-PHY Shut-Down state control register	0x0040	R/W
CSI_HOST_DPHY_RSTZ_REG	RX D-PHY reset control register	0x0044	R/W
CSI_HOST_PHY_RX_REG	RX D-PHY RX-related signal status register	0x0048	RO
CSI_HOST_PHY_TEST_CTRL0_REG	RX D-PHY test code interface control register 0	0x0050	R/W
CSI_HOST_PHY_TEST_CTRL1_REG	RX D-PHY test code interface control register 1	0x0054	varies
CSI_HOST_SCRAMBLING_REG	Descrambler enable register	0x0300	R/W
CSI_HOST_SCRAMBLING_SEED1_REG	Data lane 0 Descrambler seed value control register	0x0304	R/W
CSI_HOST_SCRAMBLING_SEED2_REG	Data lane 1 Descrambler seed value control register	0x0308	R/W
Interrupt Registers			
CSI_HOST_INT_ST_MAIN_REG	Main interrupt status register	0x000C	RC
CSI_HOST_INT_ST_PHY_FATAL_REG	RX D-PHY fatal interrupt status register	0x00E0	RC
CSI_HOST_INT_MSK_PHY_FATAL_REG	RX D-PHY fatal interrupt mask register	0x00E4	R/W
CSI_HOST_INT_FORCE_PHY_FATAL_REG	RX D-PHY fatal interrupt force register	0x00E8	R/W
CSI_HOST_INT_ST_PKT_FATAL_REG	Packet fatal interrupt status register	0x00F0	RC
CSI_HOST_INT_MSK_PKT_FATAL_REG	Packet fatal interrupt mask register	0x00F4	R/W
CSI_HOST_INT_FORCE_PKT_FATAL_REG	Packet fatal interrupt force register	0x00F8	R/W
CSI_HOST_INT_ST_PHY_REG	RX D-PHY interrupt status register	0x0110	RC
CSI_HOST_INT_MSK_PHY_REG	RX D-PHY interrupt mask register	0x0114	R/W
CSI_HOST_INT_FORCE_PHY_REG	RX D-PHY interrupt force register	0x0118	R/W

GoBack

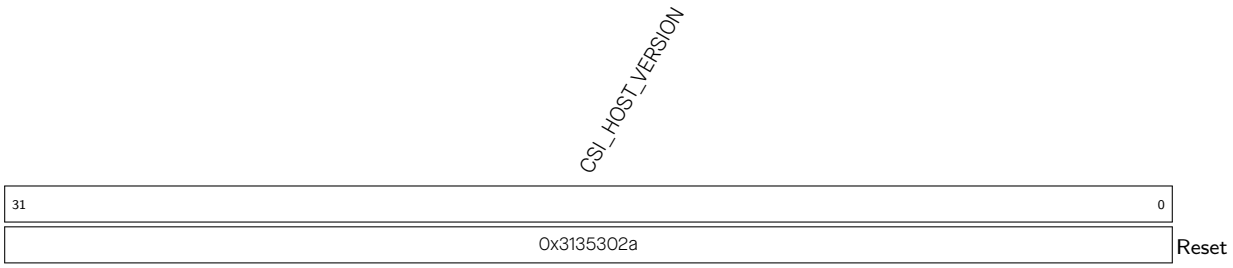
Name	Description	Address	Access
CSI_HOST_INT_ST_BNDRY_FRAME_FATAL_REG	Frame boundary fatal interrupt status register	0x0280	RC
CSI_HOST_INT_MSK_BNDRY_FRAME_FATAL_REG	Frame boundary fatal interrupt mask register	0x0284	R/W
CSI_HOST_INT_FORCE_BNDRY_FRAME_FATAL_REG	Frame boundary fatal interrupt force register	0x0288	R/W
CSI_HOST_INT_ST_SEQ_FRAME_FATAL_REG	Frame sequence fatal interrupt status register	0x0290	RC
CSI_HOST_INT_MSK_SEQ_FRAME_FATAL_REG	Frame sequence fatal interrupt mask register	0x0294	R/W
CSI_HOST_INT_FORCE_SEQ_FRAME_FATAL_REG	Frame sequence fatal interrupt force register	0x0298	R/W
CSI_HOST_INT_ST_CRC_FRAME_FATAL_REG	Frame CRC fatal interrupt status register	0x02A0	RC
CSI_HOST_INT_MSK_CRC_FRAME_FATAL_REG	Frame CRC fatal interrupt mask register	0x02A4	R/W
CSI_HOST_INT_FORCE_CRC_FRAME_FATAL_REG	Frame CRC fatal interrupt force register	0x02A8	R/W
CSI_HOST_INT_ST_PLD_CRC_FATAL_REG	Payload CRC fatal interrupt status register	0x02B0	RC
CSI_HOST_INT_MSK_PLD_CRC_FATAL_REG	Payload CRC fatal interrupt mask register	0x02B4	R/W
CSI_HOST_INT_FORCE_PLD_CRC_FATAL_REG	Payload CRC fatal interrupt force register	0x02B8	R/W
CSI_HOST_INT_ST_DATA_ID_REG	Data ID interrupt status register	0x02C0	RC
CSI_HOST_INT_MSK_DATA_ID_REG	Data ID interrupt mask register	0x02C4	R/W
CSI_HOST_INT_FORCE_DATA_ID_REG	Data ID interrupt force register	0x02C8	R/W
CSI_HOST_INT_ST_ECC_CORRECTED_REG	Header error detected and corrected interrupt status register	0x02D0	RC
CSI_HOST_INT_MSK_ECC_CORRECTED_REG	Header error detected and corrected interrupt mask register	0x02D4	R/W
CSI_HOST_INT_FORCE_ECC_CORRECTED_REG	Header error detected and corrected interrupt force register	0x02D8	R/W
Status Register			
CSI_HOST_PHY_STOPSTATE_REG	RX D-PHY stop state signal status register	0x004C	RO

42.9 Registers

The addresses in this section are relative to CSI Host base address provided in Table 9.3-2 in Chapter 9 *System and Memory*.

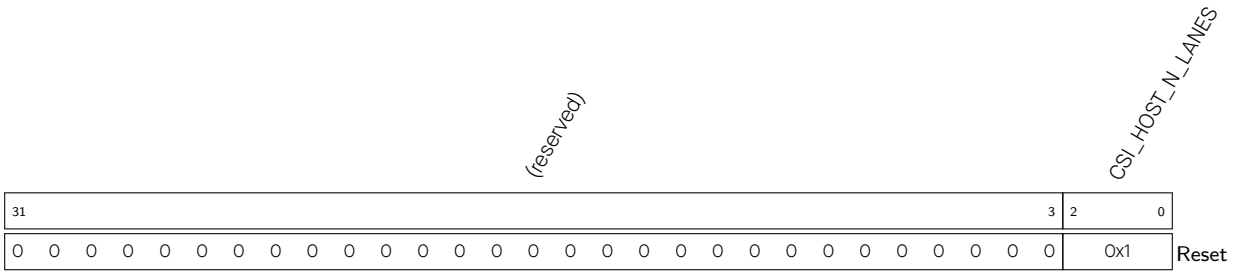
For how to program reserved fields, please refer to Section *Programming Reserved Register Field*.

Register 42.1. CSI_HOST_VERSION_REG (0x0000)



CSI_HOST_VERSION Version control register. (RO)

Register 42.2. CSI_HOST_N_LANES_REG (0x0004)



CSI_HOST_N_LANES Configures the number of active lanes that the MIPI CSI-2 host uses to receive the camera device data. This can only be updated when the RX D-PHY lane is in stop state.

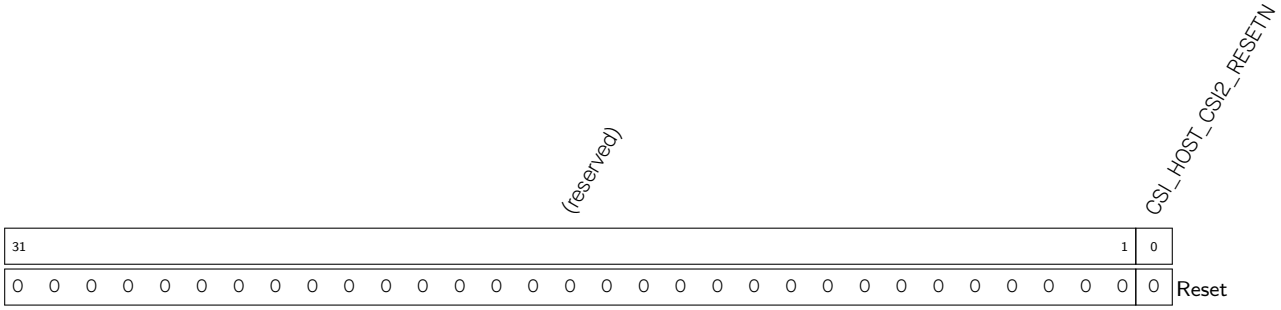
0: Data lane 0 is active

1: Both data lane 0 and data lane 1 are active

Other values: Invalid

(R/W)

Register 42.3. CSI_HOST_CSI2_RESETN_REG (0x0008)



CSI_HOST_CSI2_RESETN Configures whether to reset the internal logic of the MIPI CSI-2 host controller.

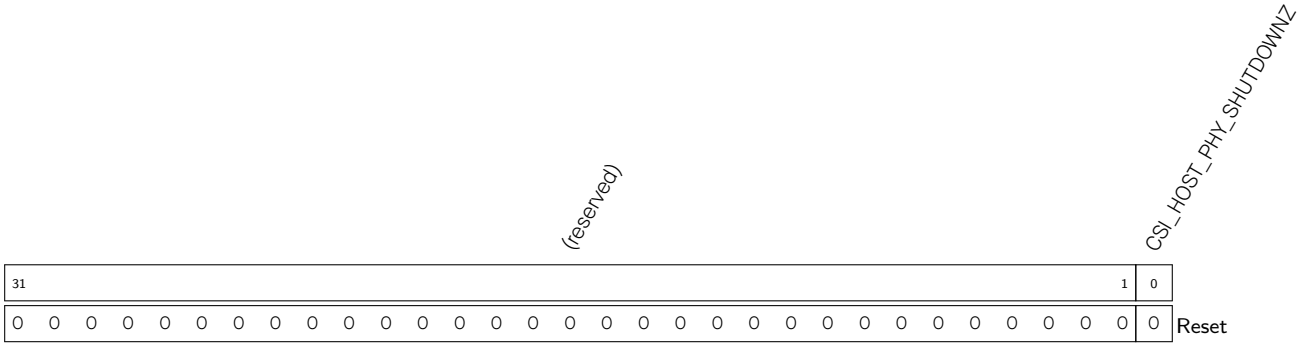
0: Reset

1: Release the reset

This register only affects the internal logic of the controller. It will not reset the configuration to default values.

(R/W)

Register 42.4. CSI_HOST_PHY_SHUTDOWNZ_REG (0x0040)



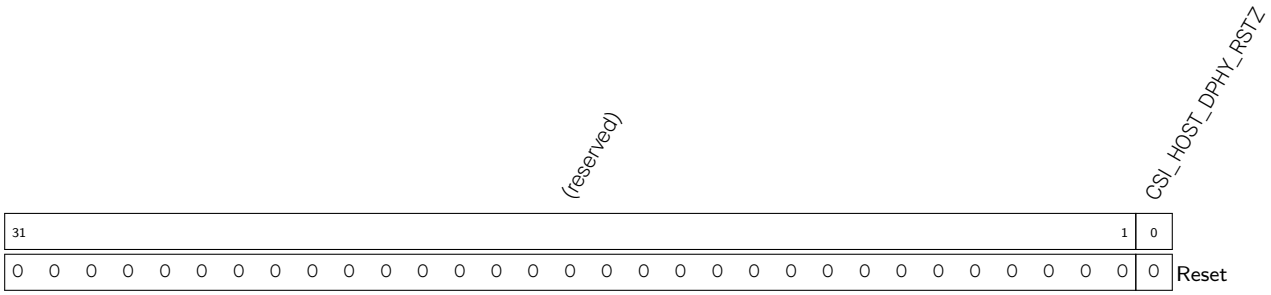
CSI_HOST_PHY_SHUTDOWNZ Configures whether to enable the Shut-Down state of the RX D-PHY.

0: Enable

1: Disable

(R/W)

Register 42.5. CSI_HOST_DPHY_RSTZ_REG (0x0044)



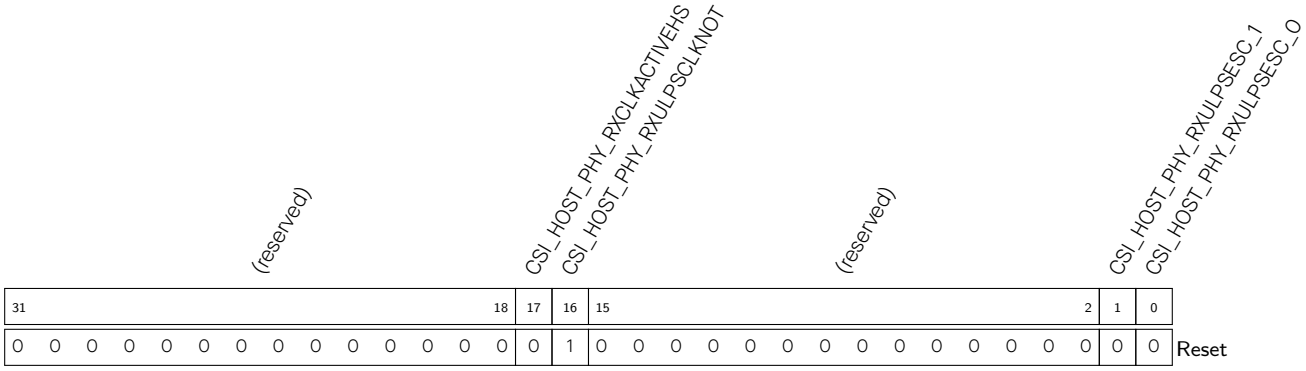
CSI_HOST_DPHY_RSTZ Configures whether to reset the RX D-PHY digital circuitry.

0: Reset

1: Release the reset

(R/W)

Register 42.6. CSI_HOST_PHY_RX_REG (0x0048)



CSI_HOST_PHY_RXULPSESC_*n* (*n*: 0-1) Represents whether data lane *n* has entered the Ultra Low Power State.

0: Entered

1: Not entered

(RO)

CSI_HOST_PHY_RXULPSCLKNOT Represents whether RX D-PHY clock lane has entered the Ultra Low Power State.

0: Entered

1: Not entered

(RO)

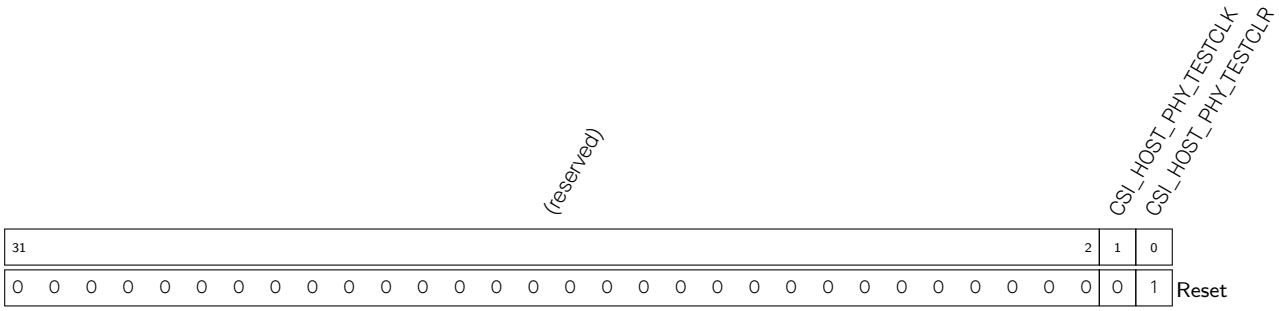
CSI_HOST_PHY_RXCLKACTIVEHS Represents whether RX D-PHY clock lane is actively receiving a DDR clock.

0: Not actively receiving

1: Actively receiving

(RO)

Register 42.7. CSI_HOST_PHY_TEST_CTRL0_REG (0x0050)



CSI_HOST_PHY_TESTCLR Configures whether to initialize the test code interface.

0: Not initialize

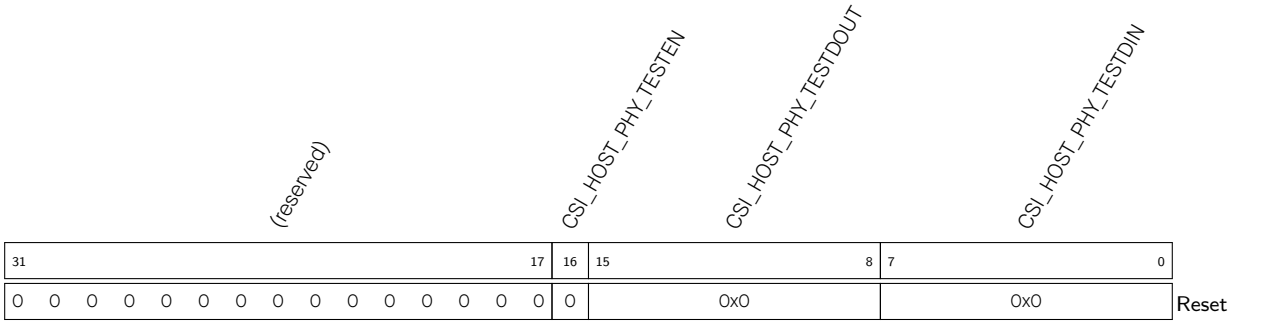
1: Initialize

To ensure the analog programmability default values are preset, this bit must be set to 1 then clear to 0 after power-up.

(R/W)

CSI_HOST_PHY_TESTCLK Configures clock to capture CSI_HOST_PHY_TESTDIN bus contents into D-PHY test code logic, with CSI_HOST_PHY_TESTEN controlling the operation selection. (R/W)

Register 42.8. CSI_HOST_PHY_TEST_CTRL1_REG (0x0054)



CSI_HOST_PHY_TESTDIN Configures the 8-bit data input of the test code interface for programming internal registers and accessing test functionalities. (R/W)

CSI_HOST_PHY_TESTDOUT Represents the 8-bit data output of the test code interface for reading data and other probing functionalities. (RO)

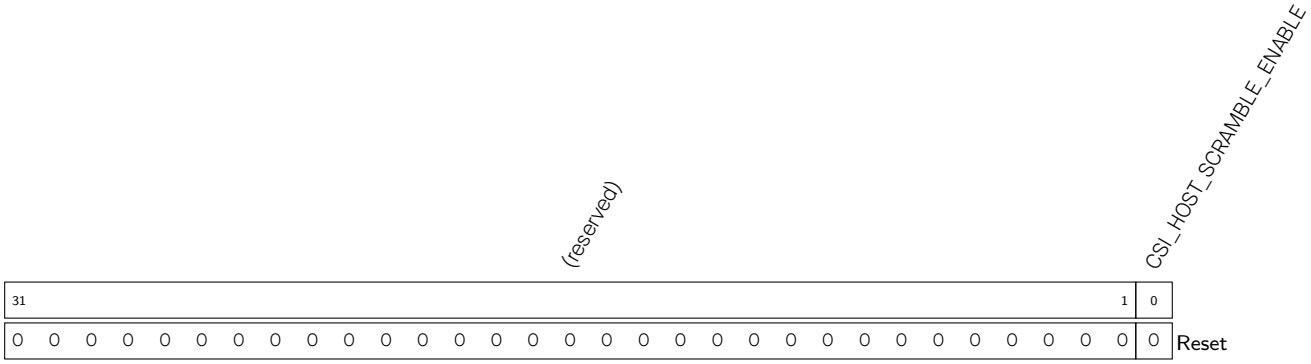
CSI_HOST_PHY_TESTEN Configures whether to performs the test data write operation of the test code interface on the rising edge or the falling edge of CSI_HOST_PHY_TESTCLK.

0: On the rising edge

1: On the falling edge

(R/W)

Register 42.9. CSI_HOST_SCRAMBLING_REG (0x0300)



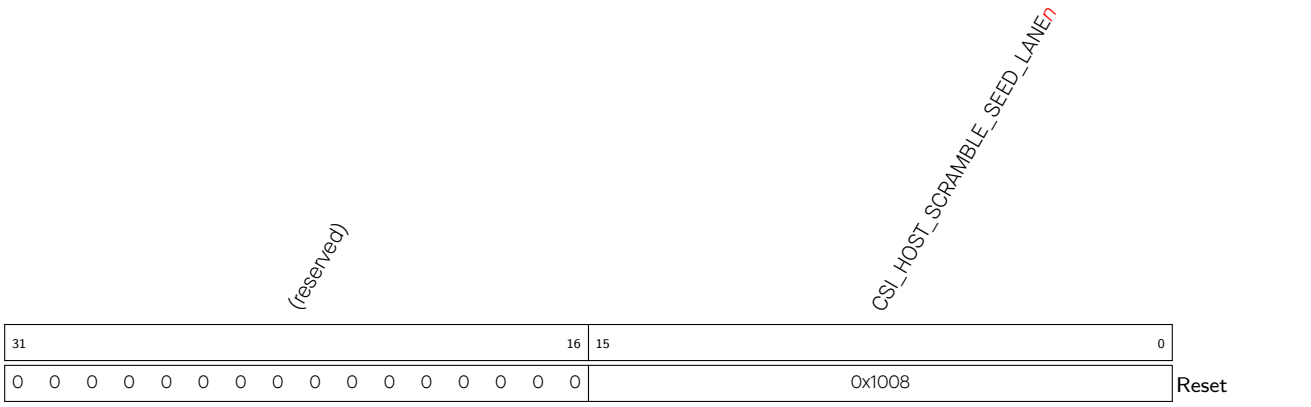
CSI_HOST_SCRAMBLE_ENABLE Configures whether to enable the descrambler.

0: Disable

1: Enable

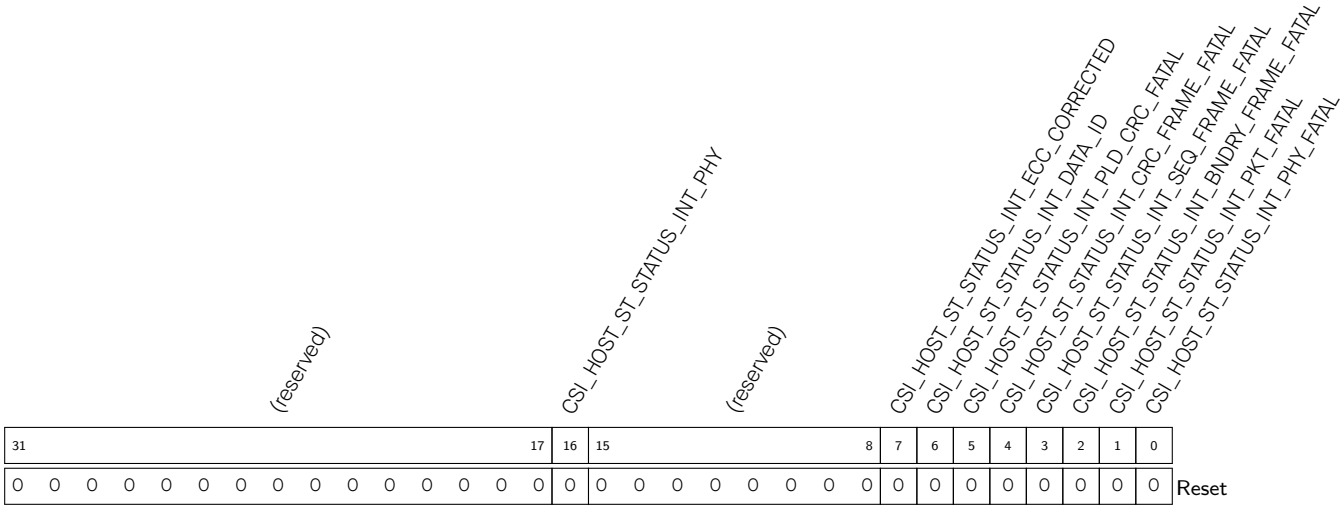
(R/W)

Register 42.10. CSI_HOST_SCRAMBLING_SEED*n*_REG (*n*: 1-2) (0x0300 + 0x04**n*)



CSI_HOST_SCRAMBLE_SEED_LANE*n-1* Configures the seed value used by the descrambler for lane *n-1*. (R/W)

Register 42.11. CSI_HOST_INT_ST_MAIN_REG (0x000C)



CSI_HOST_ST_STATUS_INT_PHY_FATAL Represents the status of CSI_HOST_INT_ST_PHY_FATAL_REG. (RC)

CSI_HOST_ST_STATUS_INT_PKT_FATAL Represents the status of CSI_HOST_INT_ST_PKT_FATAL_REG. (RC)

CSI_HOST_ST_STATUS_INT_BNDRY_FRAME_FATAL Represents the status of CSI_HOST_INT_ST_BNDRY_FRAME_FATAL_REG. (RC)

CSI_HOST_ST_STATUS_INT_SEQ_FRAME_FATAL Represents the status of CSI_HOST_INT_ST_SEQ_FRAME_FATAL_REG. (RC)

CSI_HOST_ST_STATUS_INT_CRC_FRAME_FATAL Represents the status of CSI_HOST_INT_ST_CRC_FRAME_FATAL_REG. (RC)

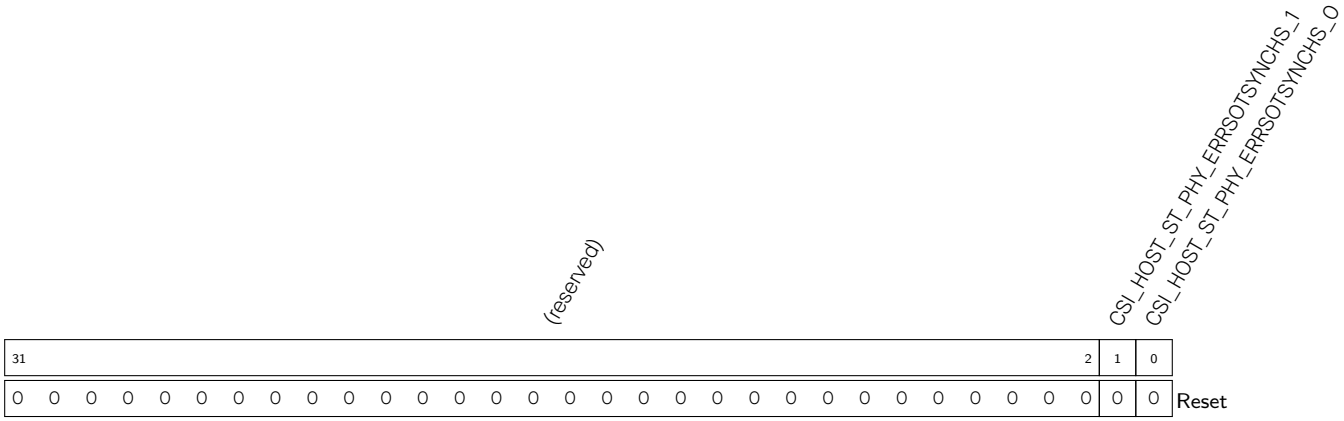
CSI_HOST_ST_STATUS_INT_PLD_CRC_FATAL Represents the status of CSI_HOST_INT_ST_PLD_CRC_FATAL_REG. (RC)

CSI_HOST_ST_STATUS_INT_DATA_ID Represents the status of CSI_HOST_INT_ST_DATA_ID_REG. (RC)

CSI_HOST_ST_STATUS_INT_ECC_CORRECTED Represents the status of CSI_HOST_INT_ST_ECC_CORRECTED_REG. (RC)

CSI_HOST_ST_STATUS_INT_PHY Represents the status of CSI_HOST_INT_PHY_REG. (RC)

Register 42.12. CSI_HOST_INT_ST_PHY_FATAL_REG (0x00E0)



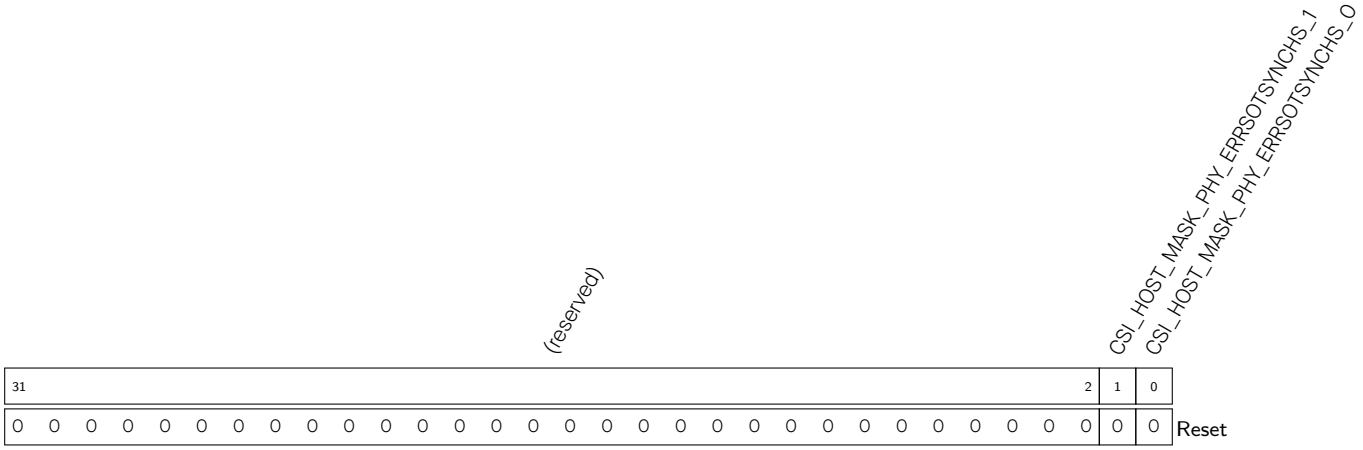
CSI_HOST_ST_PHY_ERRSOTSYNCHS_*n* (*n*: 0-1) Represents whether the PHY_ERRSOTSYNCHS_*n* error occurs.

0: Do not occur

1: Occur

(RC)

Register 42.13. CSI_HOST_INT_MSK_PHY_FATAL_REG (0x00E4)



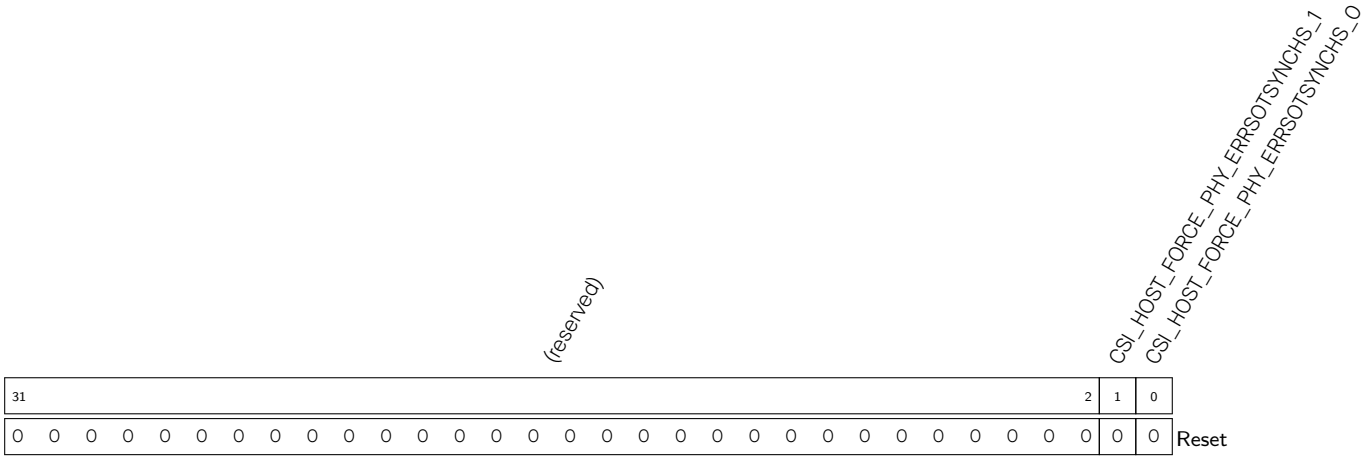
CSI_HOST_MASK_PHY_ERRSOTSYNCHS_*n* (*n*: 0-1) Configures whether to mask CSI_HOST_ST_PHY_ERRSOTSYNCHS_*n*.

0: Mask the error interrupt

1: Enable the error interrupt

(R/W)

Register 42.14. CSI_HOST_INT_FORCE_PHY_FATAL_REG (0x00E8)



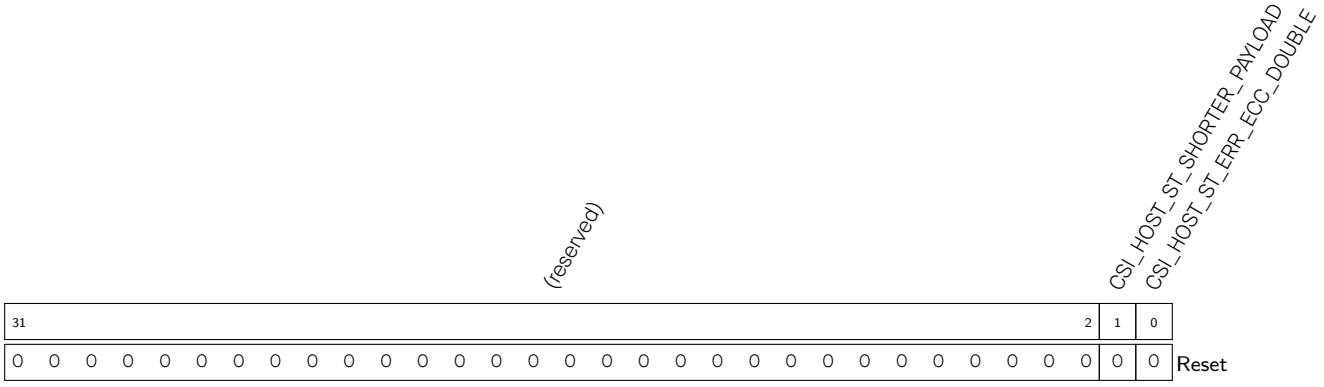
CSI_HOST_FORCE_PHY_ERRSOTSYNCHS_*n* (*n*: 0-1) Configures whether to force set CSI_HOST_ST_PHY_ERRSOTSYNCHS_*n* to 1.

0: Do not force set

1: Force set

(R/W)

Register 42.15. CSI_HOST_INT_ST_PKT_FATAL_REG (0x00F0)



CSI_HOST_ST_ERR_ECC_DOUBLE Represents whether the ERR_ECC_DOUBLE error occurs.

0: Do not occur

1: Occur

(RC)

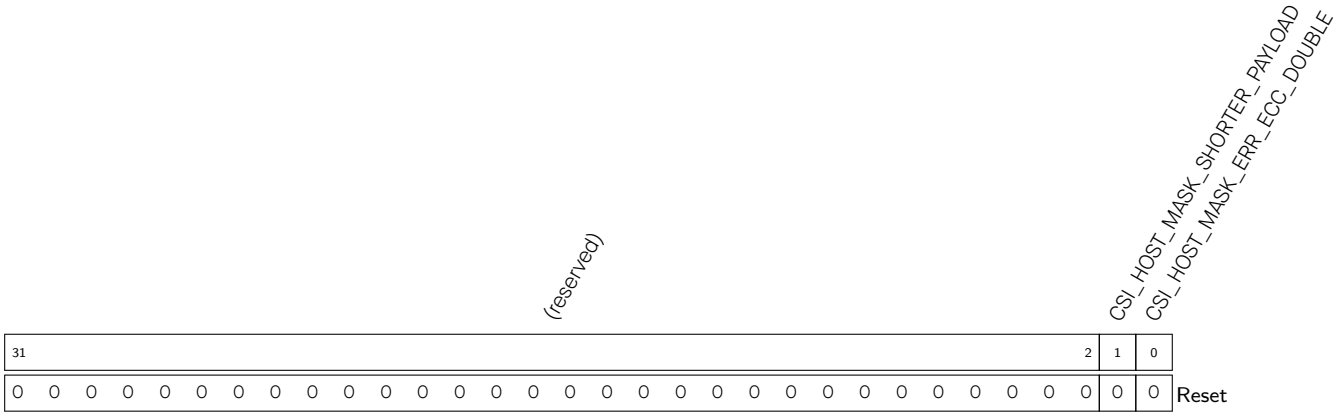
CSI_HOST_ST_SHORTER_PAYLOAD Represents whether the SHORTER_PAYLOAD error occurs.

0: Do not occur

1: Occur

(RC)

Register 42.16. CSI_HOST_INT_MSK_PKT_FATAL_REG (0x00F4)



CSI_HOST_MASK_ERR_ECC_DOUBLE Configures whether to mask CSI_HOST_ST_ERR_ECC_DOUBLE.

0: Mask the error interrupt

1: Enable the error interrupt

(R/W)

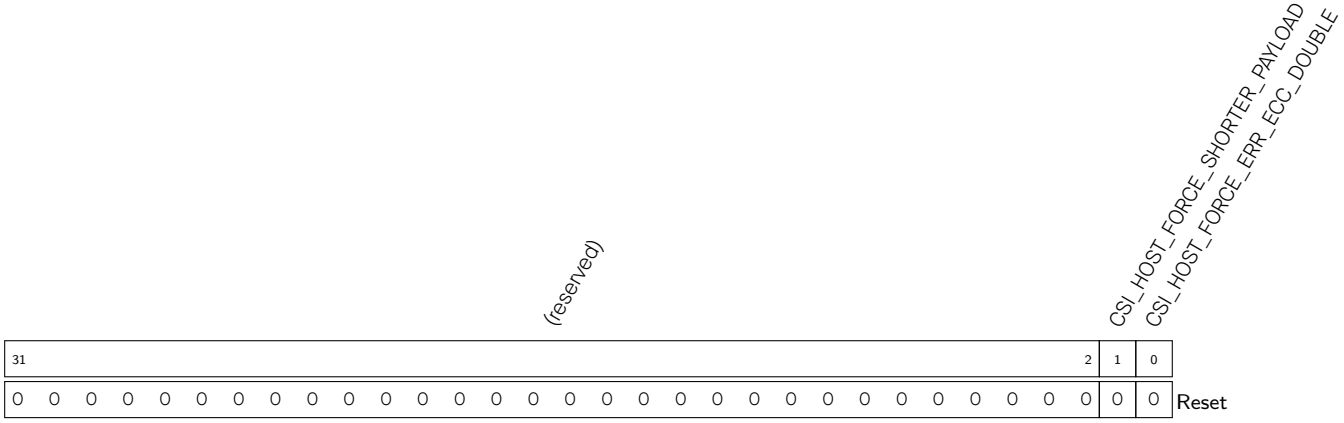
CSI_HOST_MASK_SHORTER_PAYLOAD Configures whether to mask CSI_HOST_ST_SHORTER_PAYLOAD.

0: Mask the error interrupt

1: Enable the error interrupt

(R/W)

Register 42.17. CSI_HOST_INT_FORCE_PKT_FATAL_REG (0x00F8)



CSI_HOST_FORCE_ERR_ECC_DOUBLE Configures whether to force set CSI_HOST_ST_ERR_ECC_DOUBLE to 1.

0: Do not force set

1: Force set

(R/W)

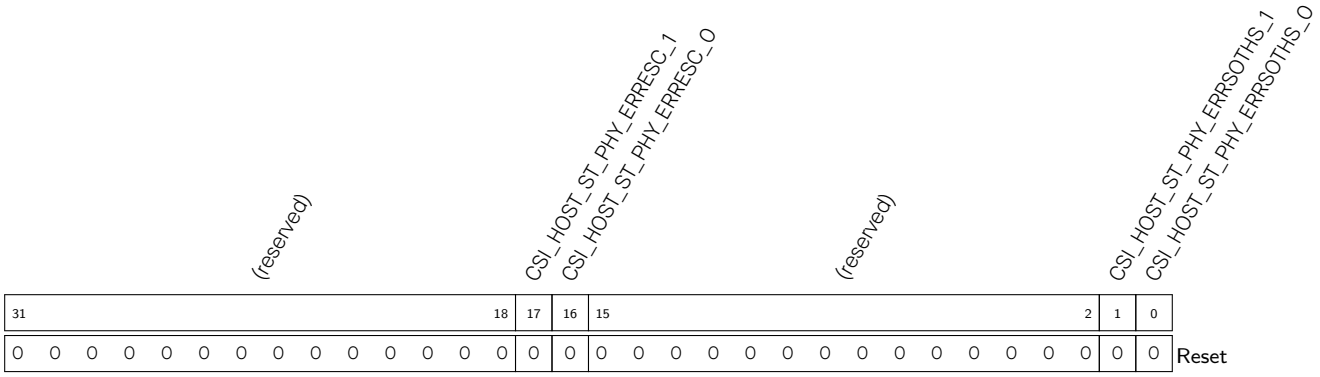
CSI_HOST_FORCE_SHORTER_PAYLOAD Configures whether to force set CSI_HOST_ST_SHORTER_PAYLOAD to 1.

0: Do not force set

1: Force set

(R/W)

Register 42.18. CSI_HOST_INT_ST_PHY_REG (0x0110)



CSI_HOST_ST_PHY_ERRSOTHS_*n* (*n*: 0-1) Represents whether the PHY_ERRSOTHS_*n* error occurs.

0: Do not occur

1: Occur

(RC)

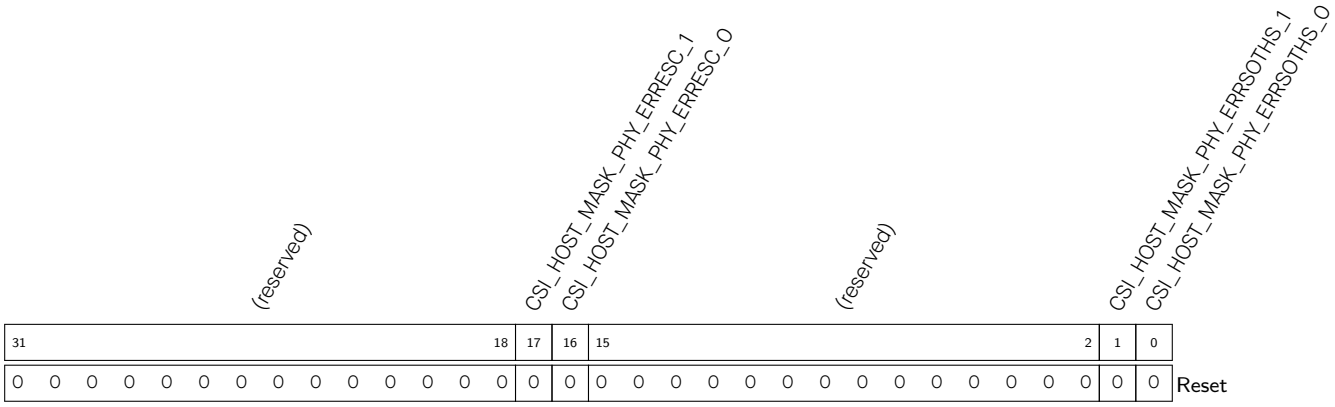
CSI_HOST_ST_PHY_ERRESC_*n* (*n*: 0-1) Represents whether the PHY_ERRESC_*n* error occurs.

0: Do not occur

1: Occur

(RC)

Register 42.19. CSI_HOST_INT_MSK_PHY_REG (0x0114)



CSI_HOST_MASK_PHY_ERRSOTHS_*n* (*n*: 0-1) Configures whether to mask CSI_HOST_ST_PHY_ERRSOTHS_*n*.

0: Mask the error interrupt

1: Enable the error interrupt

(R/W)

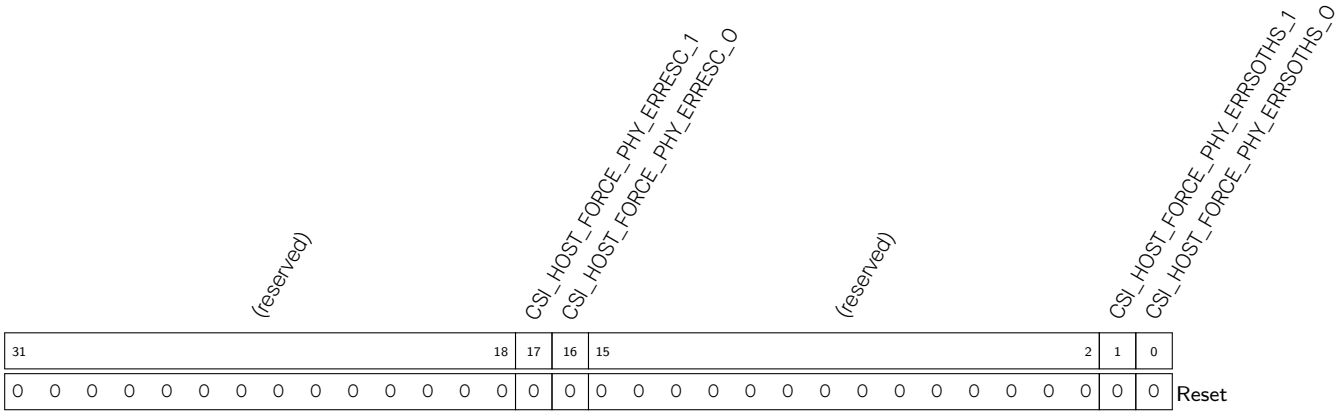
CSI_HOST_MASK_PHY_ERRESC_*n* (*n*: 0-1) Configures whether to mask CSI_HOST_ST_PHY_ERRESC_*n*.

0: Mask the error interrupt

1: Enable the error interrupt

(R/W)

Register 42.20. CSI_HOST_INT_FORCE_PHY_REG (0x0118)



CSI_HOST_FORCE_PHY_ERRSOTHS_*n* (*n*: 0-1) Configures whether to force set CSI_HOST_ST_PHY_ERRSOTHS_*n* to 1.

0: Do not force set

1: Force set

(R/W)

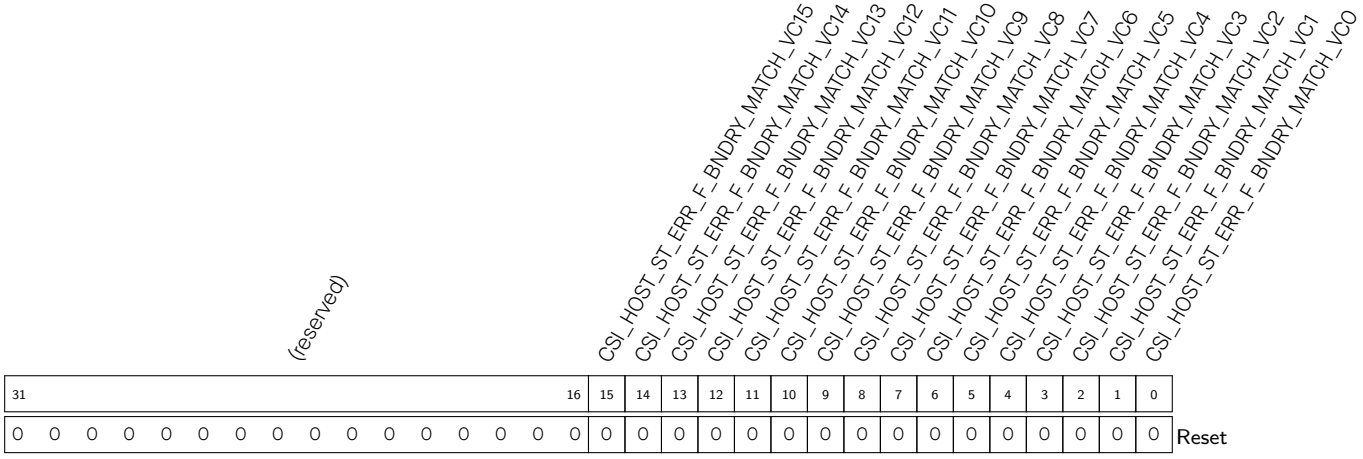
CSI_HOST_FORCE_PHY_ERRESC_*n* (*n*: 0-1) Configures whether to force set CSI_HOST_ST_PHY_ERRESC_*n* to 1.

0: Do not force set

1: Force set

(R/W)

Register 42.21. CSI_HOST_INT_ST_BNDRY_FRAME_FATAL_REG (0x0280)



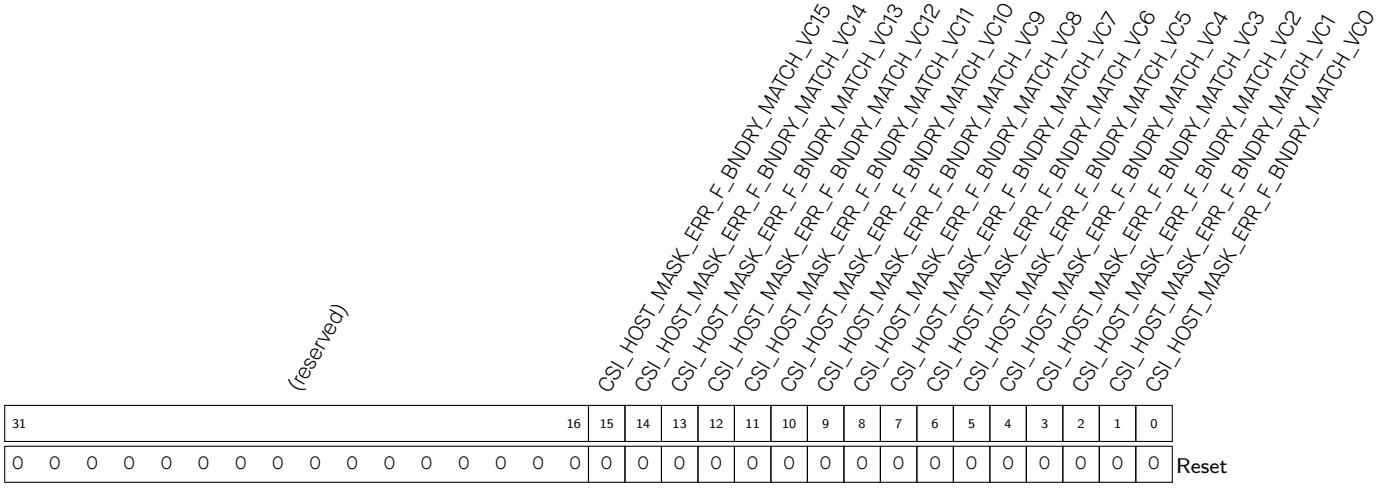
CSI_HOST_ST_ERR_F_BNDRY_MATCH_VC*n* (*n*: 0-15) Represents whether the ERR_F_BNDRY_MATCH_VC*n* error occurs.

0: Do not occur

1: Occur

(RC)

Register 42.22. CSI_HOST_INT_MSK_BNDRY_FRAME_FATAL_REG (0x0284)



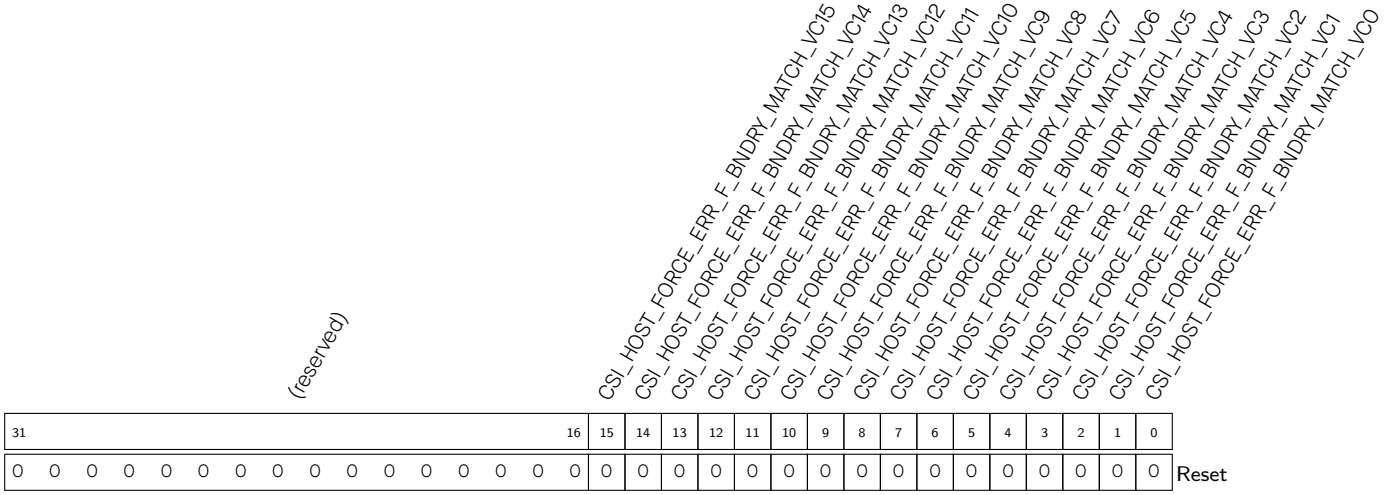
CSI_HOST_MASK_ERR_F_BNDRY_MATCH_VC*n* (*n*: 0-15) Configures whether to mask CSI_HOST_ST_ERR_F_BNDRY_MATCH_VC*n*.

0: Mask the error interrupt

1: Enable the error interrupt

(R/W)

Register 42.23. CSI_HOST_INT_FORCE_BNDRY_FRAME_FATAL_REG (0x0288)



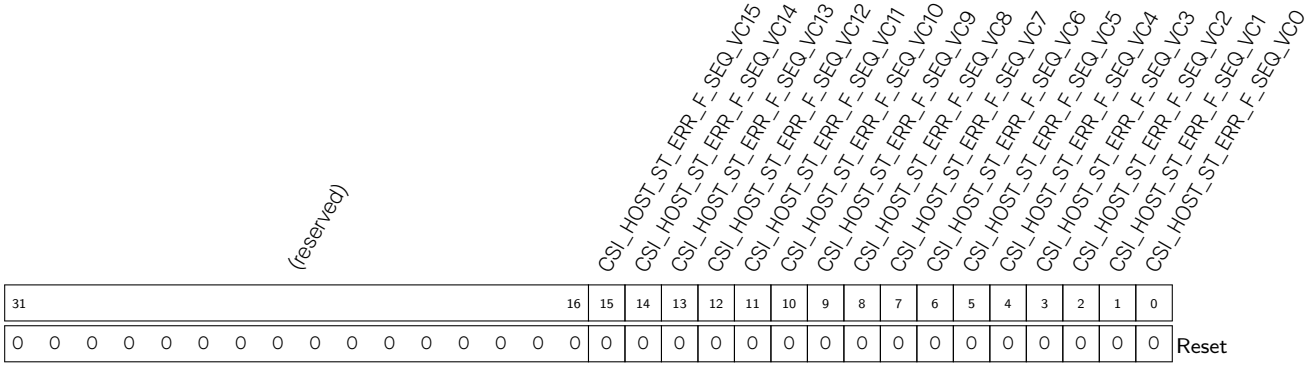
CSI_HOST_FORCE_ERR_F_BNDRY_MATCH_VC*n* (*n*: 0-15) Configures whether to force set CSI_HOST_ST_ERR_F_BNDRY_MATCH_VC*n* to 1.

0: Do not force set

1: Force set

(R/W)

Register 42.24. CSI_HOST_INT_ST_SEQ_FRAME_FATAL_REG (0x0290)



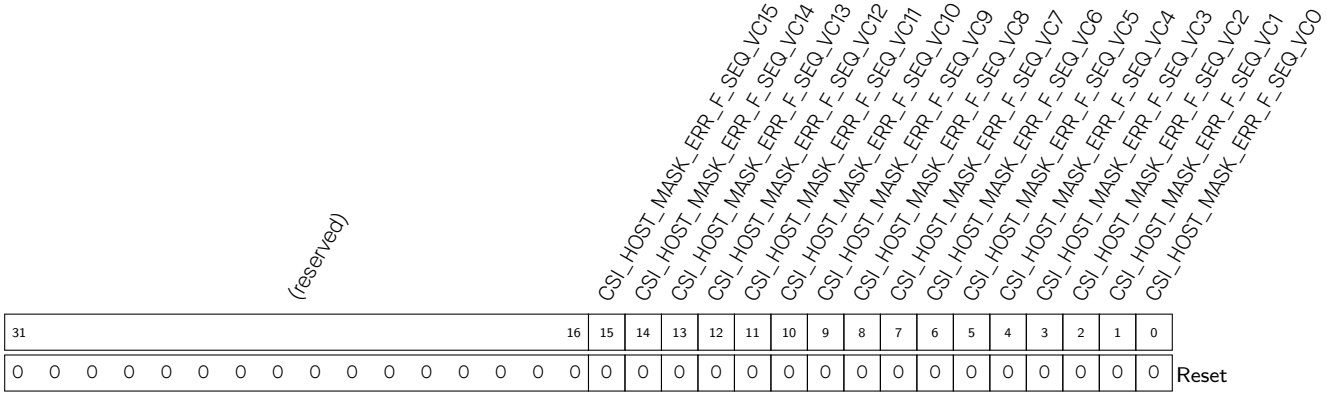
CSI_HOST_ST_ERR_F_SEQ_VC*n* (*n*: 0-15) Represents whether the ERR_F_SEQ_VC*n* error occurs.

0: Do not occur

1: Occur

(RC)

Register 42.25. CSI_HOST_INT_MSK_SEQ_FRAME_FATAL_REG (0x0294)



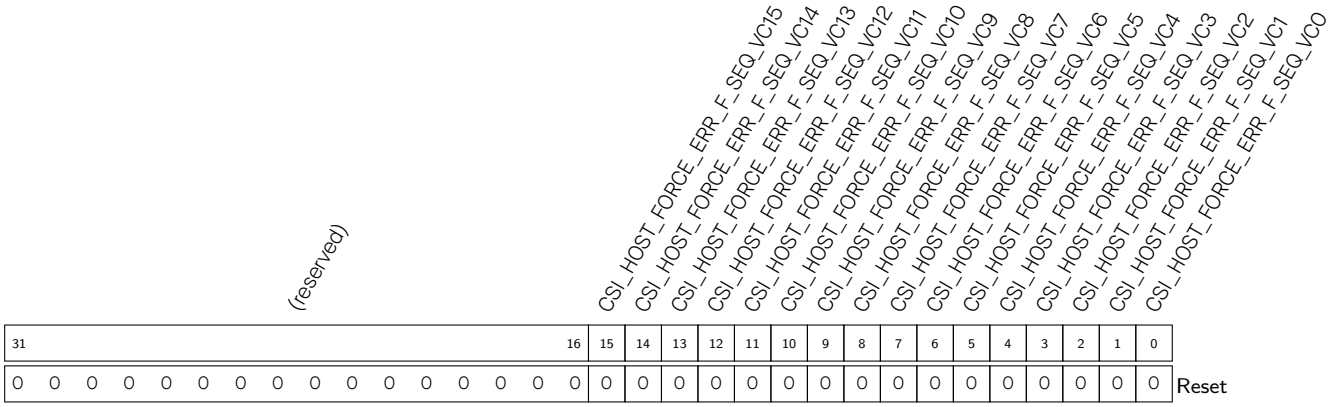
CSI_HOST_MASK_ERR_F_SEQ_VC*n* (*n*: 0-15) Configures whether to mask CSI_HOST_ST_ERR_F_SEQ_VC*n*.

0: Mask the error interrupt

1: Enable the error interrupt

(R/W)

Register 42.26. CSI_HOST_INT_FORCE_SEQ_FRAME_FATAL_REG (0x0298)



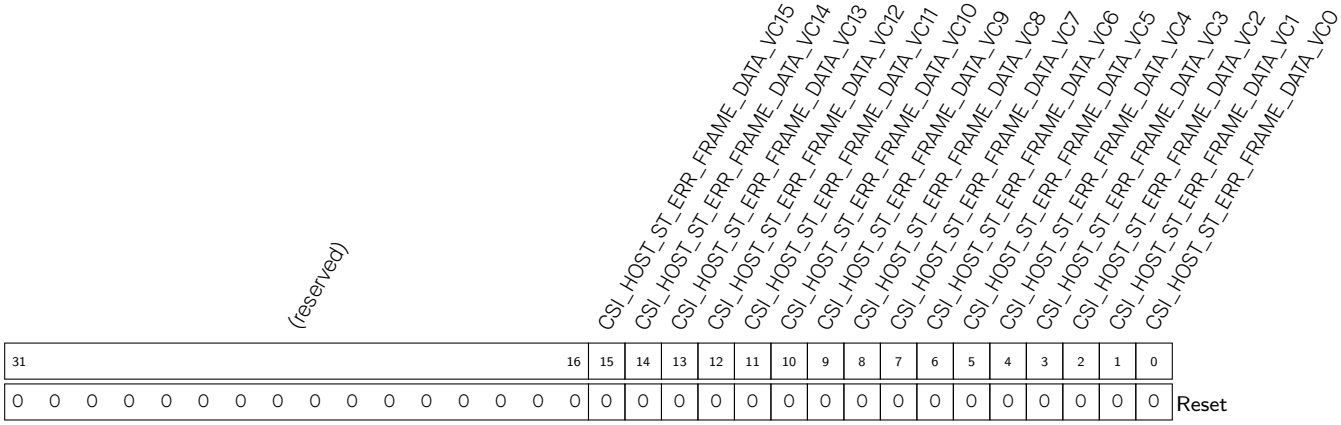
CSI_HOST_FORCE_ERR_F_SEQ_VC*n* (*n*: 0-15) Configures whether to force set CSI_HOST_ST_ERR_F_SEQ_VC*n* to 1.

0: Do not force set

1: Force set

(R/W)

Register 42.27. CSI_HOST_INT_ST_CRC_FRAME_FATAL_REG (0x02A0)



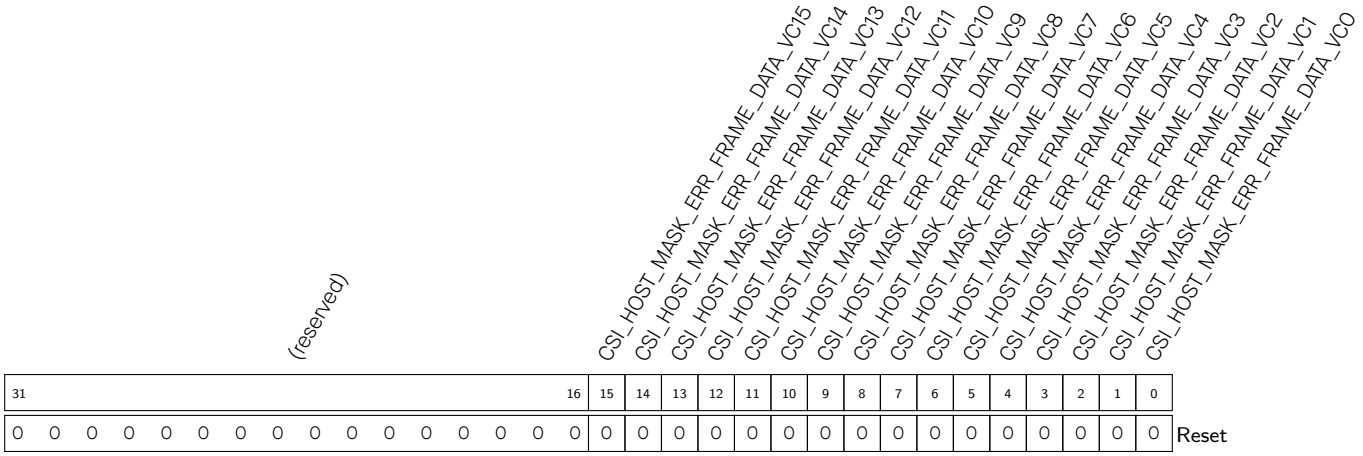
CSI_HOST_ST_ERR_FRAME_DATA_VC*n* (*n*: 0-15) Represents whether the ERR_FRAME_DATA_VC*n* error occurs.

0: Do not occur

1: Occur

(RC)

Register 42.28. CSI_HOST_INT_MSK_CRC_FRAME_FATAL_REG (0x02A4)



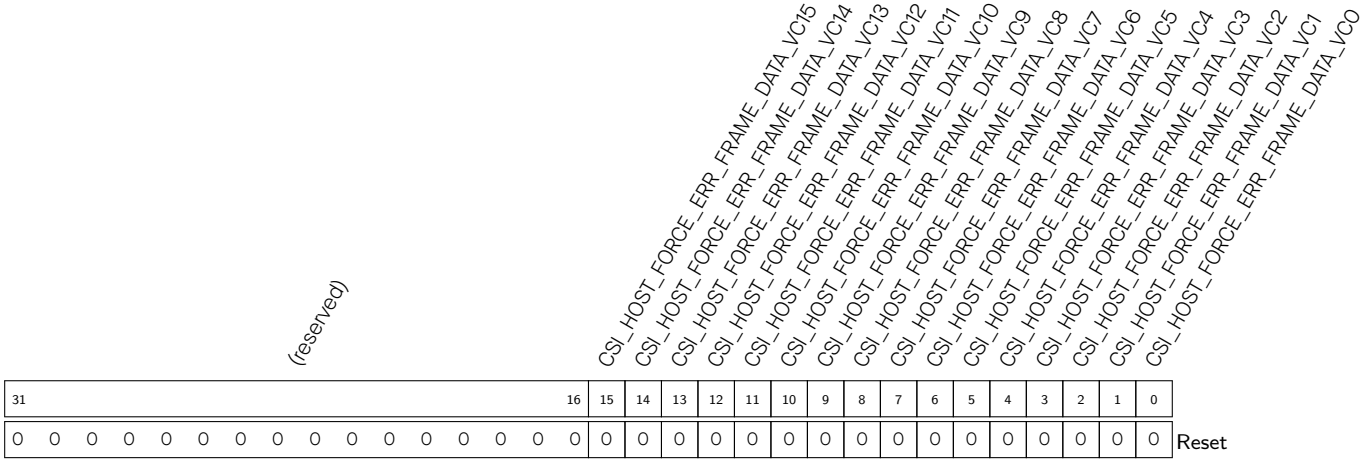
CSI_HOST_MASK_ERR_FRAME_DATA_VC*n* (*n*: 0-15) Configures whether to mask CSI_HOST_ST_ERR_FRAME_DATA_VC*n*.

0: Mask the error interrupt

1: Enable the error interrupt

(R/W)

Register 42.29. CSI_HOST_INT_FORCE_CRC_FRAME_FATAL_REG (0x02A8)



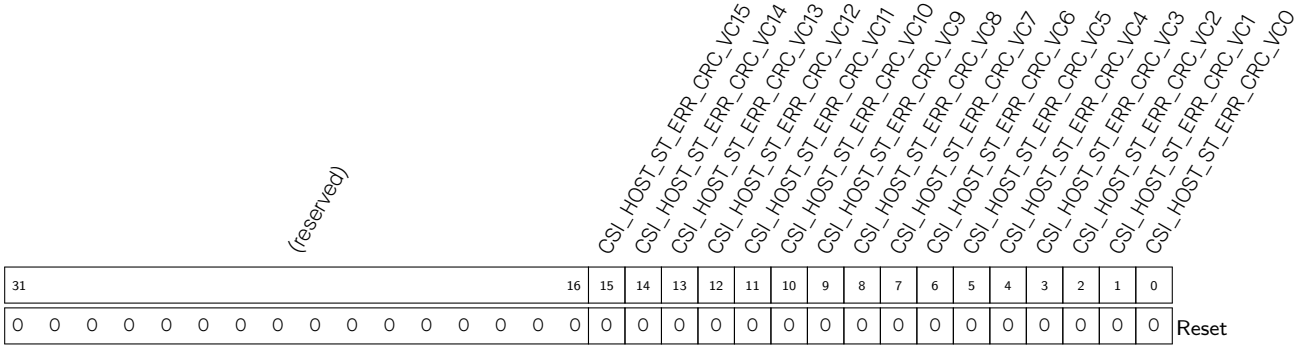
CSI_HOST_FORCE_ERR_FRAME_DATA_VC*n* (*n*: 0-15) Configures whether to force set CSI_HOST_ST_ERR_FRAME_DATA_VC*n* to 1.

0: Do not force set

1: Force set

(R/W)

Register 42.30. CSI_HOST_INT_ST_PLD_CRC_FATAL_REG (0x02B0)



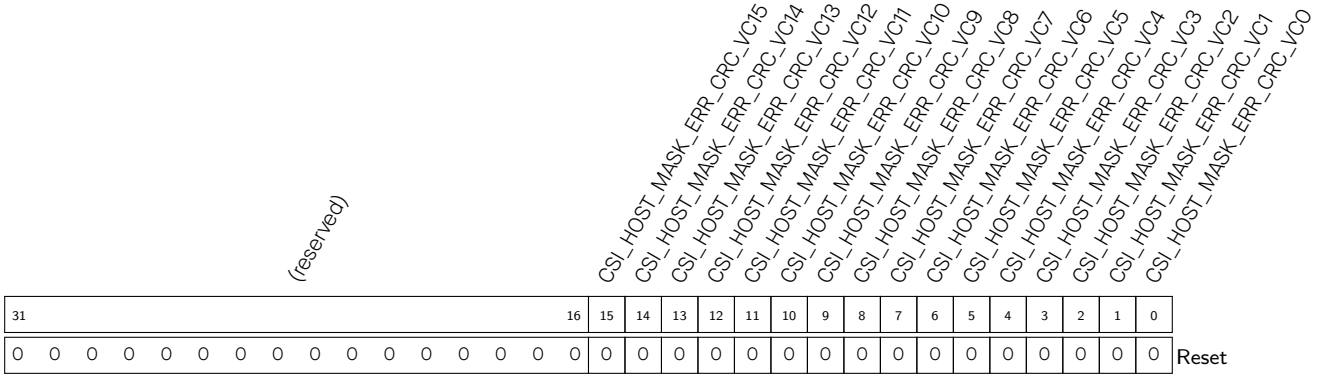
CSI_HOST_ST_ERR_CRC_VC*n* (*n*: 0-15) Represents whether the ERR_CRC_VC*n* error occurs.

0: Do not occur

1: Occur

(RC)

Register 42.31. CSI_HOST_INT_MSK_PLD_CRC_FATAL_REG (0x02B4)



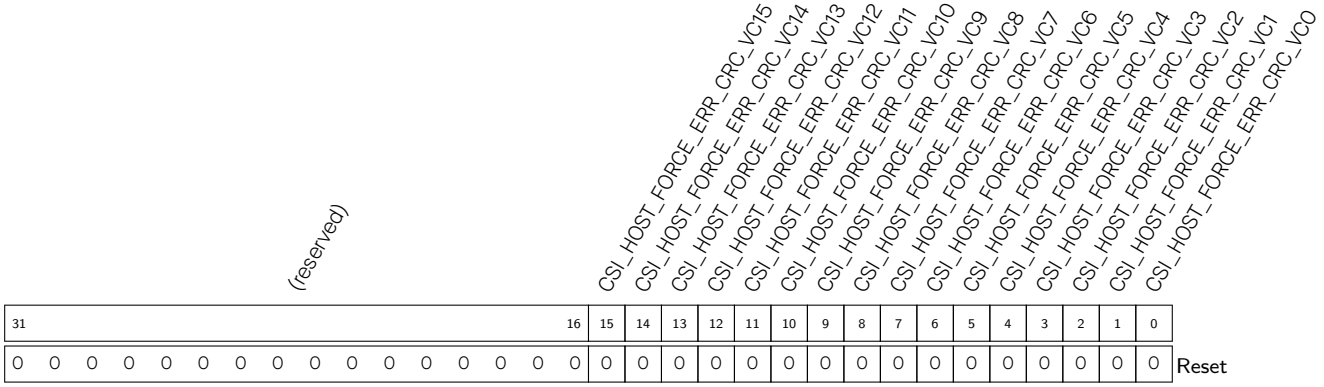
CSI_HOST_MASK_ERR_CRC_VC*n* (*n*: 0-15) Configures whether to mask CSI_HOST_ST_ERR_CRC_VC*n*.

0: Mask the error interrupt

1: Enable the error interrupt

(R/W)

Register 42.32. CSI_HOST_INT_FORCE_PLD_CRC_FATAL_REG (0x02B8)



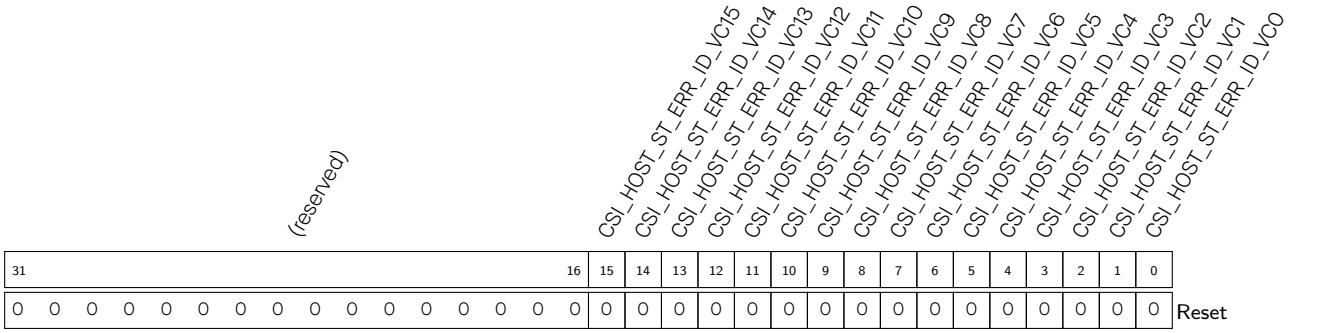
CSI_HOST_FORCE_ERR_CRC_VC*n* (*n*: 0-15) Configures whether to force set CSI_HOST_ST_ERR_CRC_VC*n* to 1.

0: Do not force set

1: Force set

(R/W)

Register 42.33. CSI_HOST_INT_ST_DATA_ID_REG (0x02C0)



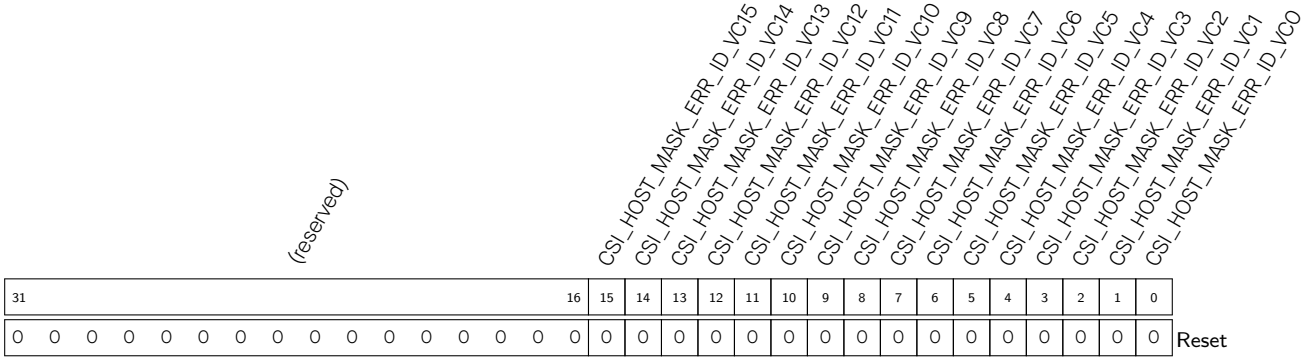
CSI_HOST_ST_ERR_ID_VC*n* (*n*: 0-15) Represents whether the ERR_ID_VC*n* error occurs.

0: Do not occur

1: Occur

(RC)

Register 42.34. CSI_HOST_INT_MSK_DATA_ID_REG (0x02C4)



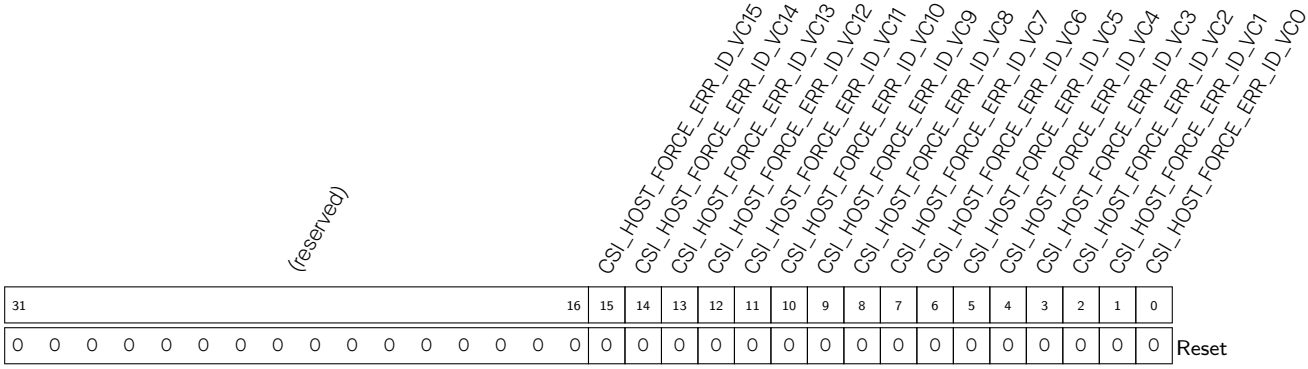
CSI_HOST_MASK_ERR_ID_VC*n* (*n*: 0-15) Configures whether to mask CSI_HOST_ST_ERR_ID_VC*n*.

0: Mask the error interrupt

1: Enable the error interrupt

(R/W)

Register 42.35. CSI_HOST_INT_FORCE_DATA_ID_REG (0x02C8)



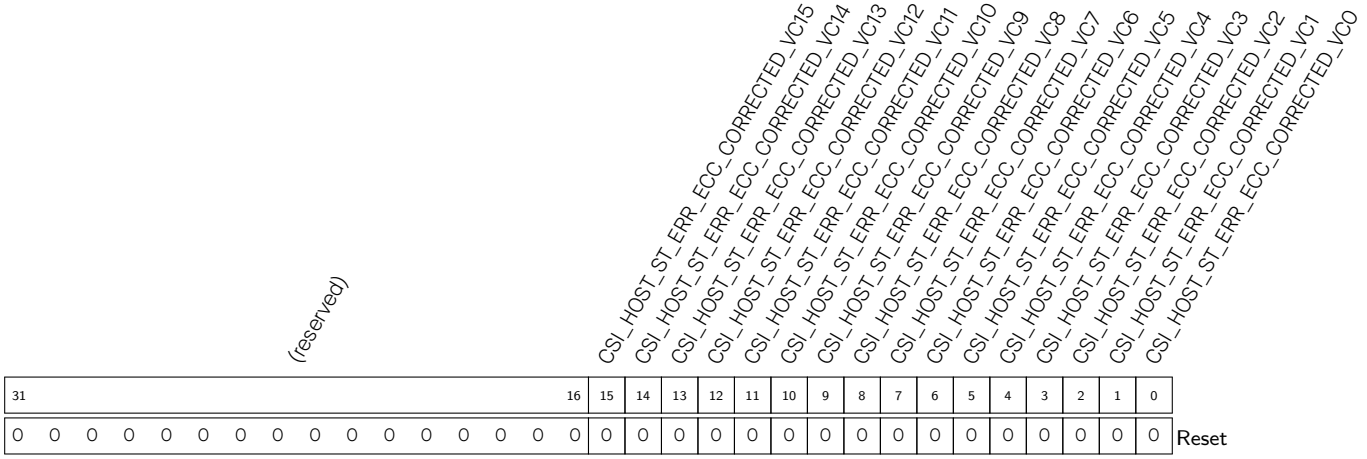
CSI_HOST_FORCE_ERR_ID_VC*n* (*n*: 0-15) Configures whether to force set CSI_HOST_ST_ERR_ID_VC*n* to 1.

0: Do not force set

1: Force set

(R/W)

Register 42.36. CSI_HOST_INT_ST_ECC_CORRECTED_REG (0x02D0)



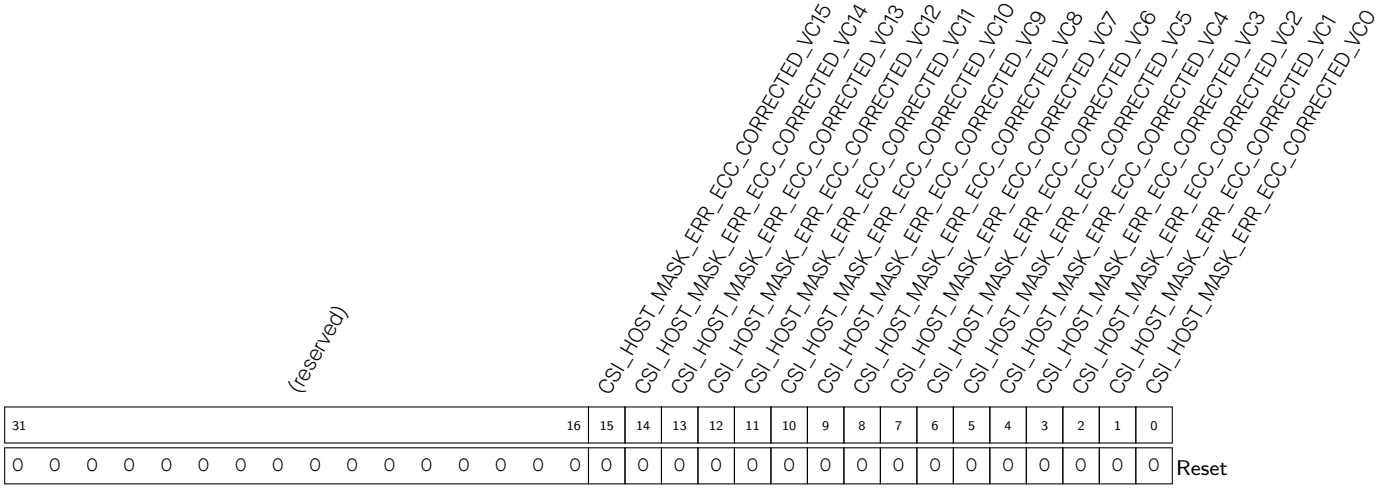
CSI_HOST_ST_ERR_ECC_CORRECTED_VC*n* (*n*: 0-15) Represents whether the ERR_ECC_CORRECTED_VC*n* error occurs.

0: Do not occur

1: Occur

(RC)

Register 42.37. CSI_HOST_INT_MSK_ECC_CORRECTED_REG (0x02D4)



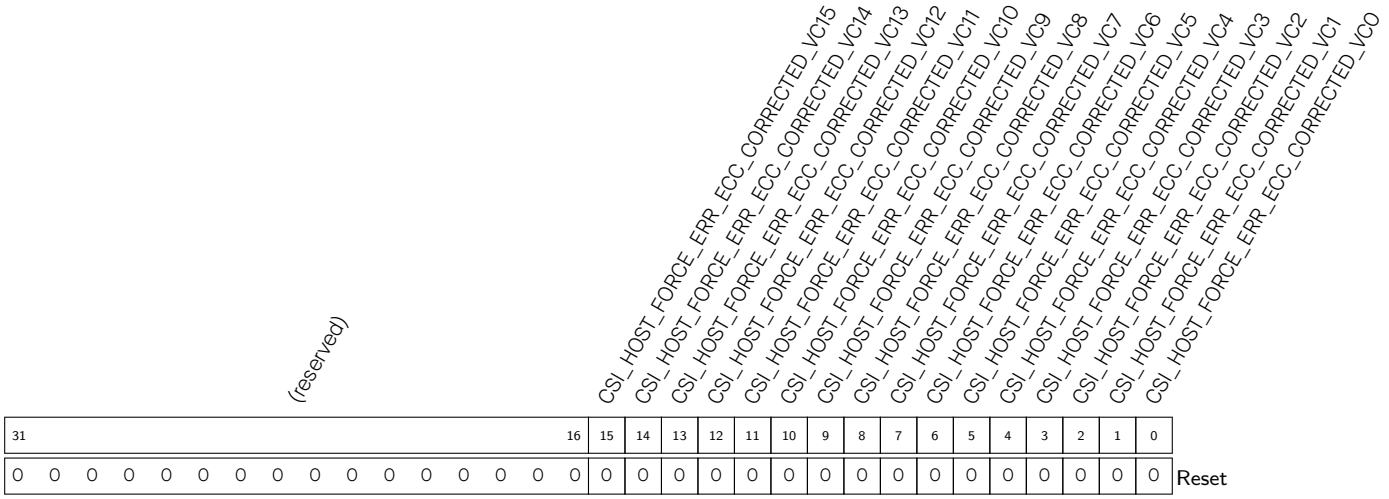
CSI_HOST_MASK_ERR_ECC_CORRECTED_VC*n* (*n*: 0-15) Configures whether to mask CSI_HOST_ST_ERR_ECC_CORRECTED_VC*n*.

0: Mask the error interrupt

1: Enable the error interrupt

(R/W)

Register 42.38. CSI_HOST_INT_FORCE_ECC_CORRECTED_REG (0x02D8)



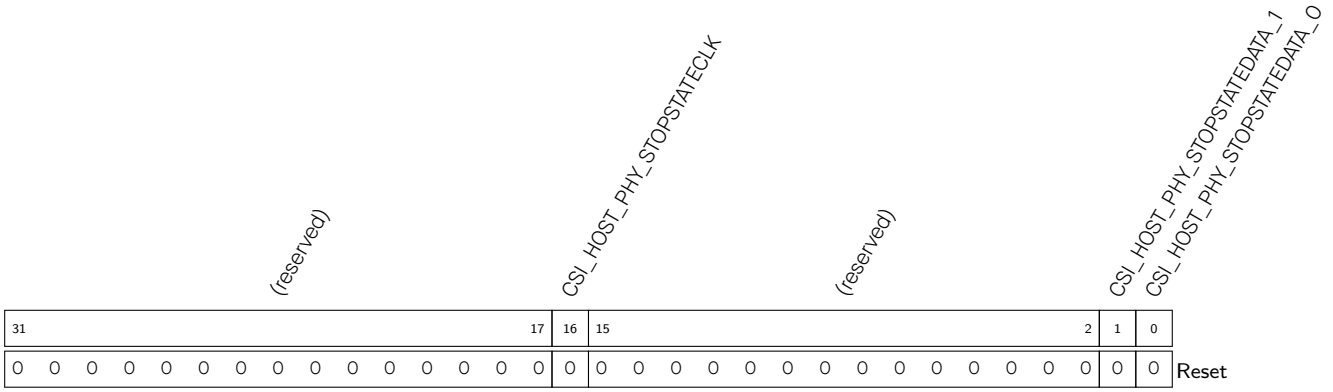
CSI_HOST_FORCE_ERR_ECC_CORRECTED_VC*n* (*n*: 0-15) Configures whether to force set CSI_HOST_ST_ERR_ECC_CORRECTED_VC*n* to 1.

0: Do not force set

1: Force set

(R/W)

Register 42.39. CSI_HOST_PHY_STOPSTATE_REG (0x004C)



CSI_HOST_PHY_STOPSTATEDATA_*n* (*n*: 0-1) Represents whether data lane *n* is in stop state.

0: Not in stop state

1: In stop state

(RO)

CSI_HOST_PHY_STOPSTATECLK Represents whether clock lane in stop state.

0: Not in stop state

1: In stop state

(RO)

Chapter 43

MIPI DSI

43.1 Introduction

MIPI DSI stands for Mobile Industry Processor Interface Display Serial Interface. It is a standard interface protocol for connecting displays to mobile devices, such as smartphones, tablets, and other embedded systems. MIPI DSI in ESP32-P4 consists of three parts: a DSI Bridge, an MIPI DSI Host Controller, and an MIPI Bidirectional D-PHY. It provides one clock lane and two data lanes, and supports a maximum bit rate of 1.5 Gbps per lane for communication with a DSI-compliant display. DSI (Display Serial Interface) is part of a group of communication protocols defined by the MIPI Alliance.

The DSI Bridge provides an interface between system memory and the MIPI DSI Host Controller. It receives image data through VDMA from system memory, and outputs image data through the Display Pixel Interface (DPI) to the MIPI DSI Host Controller.

The MIPI DSI Host Controller implements the protocol functions defined in the MIPI DSI Specification. It receives image data from the DSI Bridge and outputs image data through the PHY-Protocol Interface (PPI) to the MIPI Bidirectional D-PHY.

The MIPI Bidirectional D-PHY implements the physical layer of bidirectional universal lanes for the MIPI D-PHY interface, which provides one clock lane and two data lanes. The PHY is only used as master.

43.2 Feature List

MIPI DSI in ESP32-P4 supports the following features:

- Configuration of DSI bridge or VDMA as flow controller when reading image data from memory
- Configurable DPI video timing in DSI Bridge
- Various input image formats: RGB888, RGB666, RGB565, YUV422, YUV420, GRAY
- Various DPI image output formats: RGB888, RGB666, RGB565
- Maximum DPI frequency: 240 MHz
- MIPI DSI Host compliant with MIPI DSI specification
- PHY compliant with MIPI D-PHY interface specification, revision 1.1
- The PHY attached with a PLL
- continuous and non-continuous clock lanes
- Escape Mode through data lane 0
- Lane operation ranging from 80 Mbps to 1.5 Gbps in HP TX

- burst and non-burst Video Modes
- Command Mode through APB interface to access display registers
- Supports commands defined in MIPI Alliance specification for Display Command Set (DCS)
- Video pattern generator
- Supports Ultra Low-Power mode

43.3 Architectural Overview

Figure 43.3-1 shows the architecture of MIPI DSI.

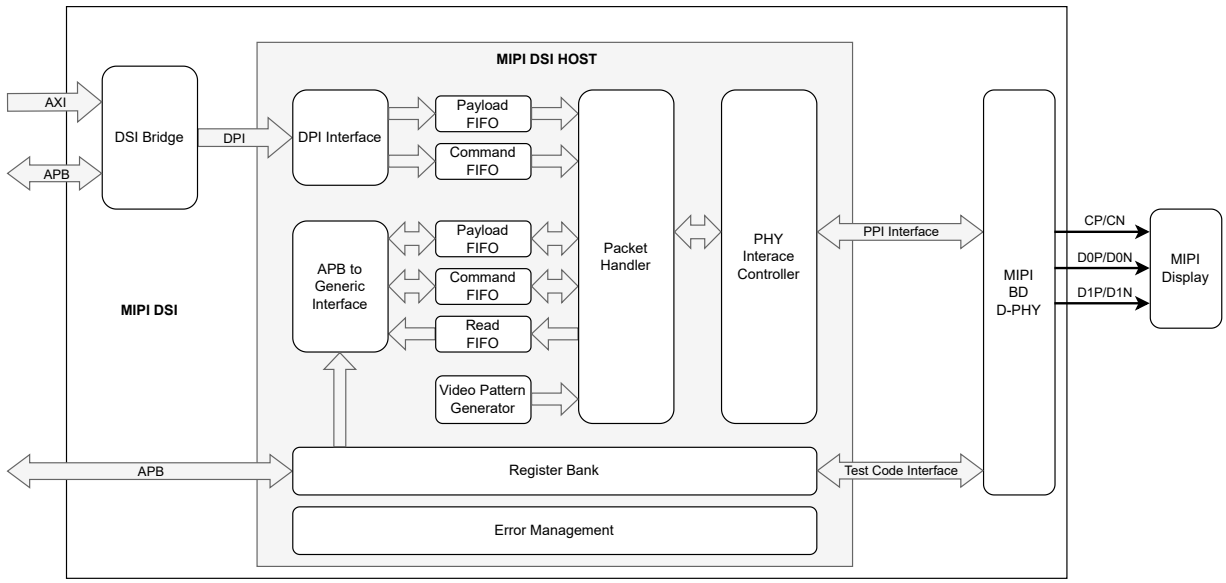


Figure 43.3-1. MIPI DSI architecture

- **DSI Bridge**: The DSI Bridge receives image data from system memory through VDMA, converts it to the DPI data format, and then outputs it to the MIPI DSI Host.
- **MIPI DSI Host**: The MIPI DSI Host receives image data from two sources: the DSI Bridge, or the video pattern generator. Then the received image data is packaged by the Packet Handler according to the MIPI DSI Specification and sent to the PHY Interface Controller. Finally, the image data is converted to the PPI data format and output to the MIPI DSI Bidirectional D-PHY, which includes:
  - **DPI Interface**: Receives pixel data from the DSI Bridge and buffers it to a FIFO.
  - **Register Bank**: Provides access to configuration and control registers.
  - **APB to Generic Interface**: Converts APB operations into Generic Commands.
  - **Video Pattern Generator**: Generates video test patterns.
  - **Packet Handler**: Schedules activities inside the link, including packet building, generating ECC CRC codes, packet reception, and so on.
  - **PHY Interface Controller**: Manages the PHY PPI interfaces, including enabling low-power(LP) transmission/reception, high-speed(HS) transmission, and so on.

  - **Error Management**: Monitors and notifies error conditions on the DSI link.

- **MIPI DSI D-PHY**: The MIPI DSI Bidirectional D-PHY implements the physical layer with one clock lane and two data lanes, connecting to the external display driver. MIPI DSI Bidirectional D-PHY also includes a PLL to generate the HS TX clock.

43.4 Functional Description

This section introduces different operating states and modes of MIPI Bi-Direction (BD) D-PHY and some major functions that facilitate the use of MIPI DSI.

43.4.1 DSI Bridge

DSI Bridge receives image data from system memory through VDMA. The data transfer flow controller can be configured to VDMA or DSI Bridge through DSI_BRG_DSI_DMA_FLOW_CONTROLLER. It is recommended to use VDMA as the flow controller for better performance.

When receiving image data with VDMA as the flow controller, the image can be split into multiple block transfers. To enable this functionality, set DSI_BRG_DMA_MULTIBLK_EN to 1 and configure DSI_BRG_DMA_FLOW_MULTIBLK_NUM with the desired block number. For a detailed explanation of block transfers, refer to *Transfer Hierarchy* in 7 *VDMA Controller (VDMA)*.

DSI Bridge supports the following input data types: RGB888, RGB666, RGB565, YUV422, YUV420, and GRAY. For YUV422, DSI Bridge supports 8 different pixel arrangements. To select whether the input data type is YUV, configure the DSI_BRG_DATA_IN_TYPE register, then use DSI_BRG_RAW_TYPE to specify the exact input format. For the output format sent to DSI Host, use DSI_BRG_DPI_TYPE to specify the RGB format.

All supported input pixel arrangements is list in table 43.4-1.

Table 43.4-1. input pixel arrangement

Color Mode	Addr + 7	Addr + 6	Addr + 5	Addr + 4	Addr + 3	Addr + 2	Addr + 1	Addr + 0
RGB888	G2[7:0]	B2[7:0]	R1[7:0]	G1[7:0]	B1[7:0]	R0[7:0]	G0[7:0]	B0[7:0]
	B5[7:0]	R4[7:0]	G4[7:0]	B4[7:0]	R3[7:0]	G3[7:0]	B3[7:0]	R2[7:0]
	R7[7:0]	G7[7:0]	B7[7:0]	R6[7:0]	G6[7:0]	B6[7:0]	R5[7:0]	G5[7:0]
RGB565	R3[4:0]G3[5:3]	G3[2:0]B3[4:0]	R2[4:0]G2[5:3]	G2[2:0]B2[4:0]	R1[4:0]G1[5:3]	G1[2:0]B1[4:0]	R0[4:0]G0[5:3]	G0[2:0]B0[4:0]
YUV422 DSI_BRG_YUV422 _FORMAT = 0 DSI_BRG_YUV _PIX_ENDIAN = 0	Y3[7:0]	U3[7:0]	Y2[7:0]	V2[7:0]	Y1[7:0]	U1[7:0]	Y0[7:0]	V0[7:0]
YUV422 DSI_BRG_YUV422 _FORMAT = 0 DSI_BRG_YUV _PIX_ENDIAN = 1	Y0[7:0]	U0[7:0]	Y1[7:0]	V1[7:0]	Y2[7:0]	U2[7:0]	Y3[7:0]	V3[7:0]
YUV422 DSI_BRG_YUV422 _FORMAT = 1 DSI_BRG_YUV _PIX_ENDIAN = 0	Y3[7:0]	V3[7:0]	Y2[7:0]	U2[7:0]	Y1[7:0]	V1[7:0]	Y0[7:0]	U0[7:0]
YUV422 DSI_BRG_YUV422 _FORMAT = 1 DSI_BRG_YUV _PIX_ENDIAN = 1	Y0[7:0]	V0[7:0]	Y1[7:0]	U1[7:0]	Y2[7:0]	V2[7:0]	Y3[7:0]	U3[7:0]

Cont'd on next page

Table 43.4-1 – cont'd from previous page

Color Mode	Addr + 7	Addr + 6	Addr + 5	Addr + 4	Addr + 3	Addr + 2	Addr + 1	Addr + 0
YUV422 DSI_BRG_YUV422_FORMAT = 2 DSI_BRG_YUV_PIX_ENDIAN = 0	U3[7:0]	Y3[7:0]	V2[7:0]	Y2[7:0]	U1[7:0]	Y1[7:0]	V0[7:0]	Y0[7:0]
YUV422 DSI_BRG_YUV422_FORMAT = 2 DSI_BRG_YUV_PIX_ENDIAN = 1	U0[7:0]	Y0[7:0]	V1[7:0]	Y1[7:0]	U2[7:0]	Y2[7:0]	V3[7:0]	Y3[7:0]
YUV422 DSI_BRG_YUV422_FORMAT = 3 DSI_BRG_YUV_PIX_ENDIAN = 0	V3[7:0]	Y3[7:0]	U2[7:0]	Y2[7:0]	V1[7:0]	Y1[7:0]	U0[7:0]	Y0[7:0]
YUV422 DSI_BRG_YUV422_FORMAT = 3 DSI_BRG_YUV_PIX_ENDIAN = 1	V0[7:0]	Y0[7:0]	U1[7:0]	Y1[7:0]	V2[7:0]	Y2[7:0]	U3[7:0]	Y3[7:0]
YUV420 Odd lines (1, 3, 5...)	Y4[7:0]	U4[7:0]	Y3[7:0]	Y2[7:0]	U2[7:0]	Y1[7:0]	Y0[7:0]	U0[7:0]
	U10[7:0]	Y9[7:0]	Y8[7:0]	U8[7:0]	Y7[7:0]	Y6[7:0]	U6[7:0]	Y5[7:0]
	Y15[7:0]	Y14[7:0]	U14[7:0]	Y13[7:0]	Y12[7:0]	U12[7:0]	Y11[7:0]	Y10[7:0]
YUV420 Even lines (2, 4, 6...)	Y4[7:0]	V4[7:0]	Y3[7:0]	Y2[7:0]	V2[7:0]	Y1[7:0]	Y0[7:0]	V0[7:0]
	V10[7:0]	Y9[7:0]	Y8[7:0]	V8[7:0]	Y7[7:0]	Y6[7:0]	V6[7:0]	Y5[7:0]
	Y15[7:0]	Y14[7:0]	V14[7:0]	Y13[7:0]	Y12[7:0]	V12[7:0]	Y11[7:0]	Y10[7:0]
GRAY	P7[7:0]	P6[7:0]	P5[7:0]	P4[7:0]	P3[7:0]	P2[7:0]	P1[7:0]	P0[7:0]

DSI Bridge outputs data through the DPI interface, and supports several RGB output bit arrangements specified by DSI_BRG_DPI_CONFIG. All supported bit arrangements are listed in table 43.4-2.

Table 43.4-2. DPI bit arrangement

	RGB565			RGB666		RGB888
DSI_BRG_DPI_CONFIG	0	1	2	0	1	0
bit-23						R7
bit-22						R6
bit-21			R4		R5	R5
bit-20		R4	R3		R4	R4
bit-19		R3	R2		R3	R3
bit-18		R2	R1		R2	R2
bit-17		R1	R0	R5	R1	R1
bit-16		R0		R4	R0	R0
bit-15	R4			R3		G7
bit-14	R3			R2		G6
bit-13	R2	G5	G5	R1	G5	G5
bit-12	R1	G4	G4	R0	G4	G4
bit-11	R0	G3	G3	G5	G3	G3
bit-10	G5	G2	G2	G4	G2	G2
bit-9	G4	G1	G1	G3	G1	G1
bit-8	G3	G0	G0	G2	G0	G0
bit-7	G2			G1		B7
bit-6	G1			G0		B6
bit-5	G0		B4	B5	B5	B5
bit-4	B4	B4	B3	B4	B4	B4
bit-3	B3	B3	B2	B3	B3	B3
bit-2	B2	B2	B1	B2	B2	B2
bit-1	B1	B1	B0	B1	B1	B1
bit-0	B0	B0		B0	B0	B0

43.4.2 DSI Host

43.4.2.1 Video Transmission

DSI Host uses DPI as input interface. It is used to transmit a real-time pixel stream from HOST to peripherals in Video Mode.

There are two kinds of video transmission modes:

- **Burst Mode**: In Burst Mode, one pixel line is buffered in FIFO and transmitted in one single packet without any interruptions. It allows the pixel line be sent in a short time and then return to lane LP state, which can reduce power consumption and enable more time for other data transmission.
- **Non-Burst Mode**: In Non-Burst Mode, one pixel line is divided into several packets. It is used to match pixel bandwidth with DSI link bandwidth, and reduce the memory requirement for both host and peripheral. There are two kinds of Non-Burst mode:
  - Non-Burst mode with sync pulse: allows the peripheral to accurately reconstruct original video timing, including sync pulse widths.
  - Non-Burst mode with sync event: accurate reconstruction of sync pulse is not required.

If the host and peripheral meet the following requirements, it is recommended to select Burst Mode. Otherwise, the pixel data may be lost, causing malfunction of the display peripheral.

- The display peripheral supports receiving a full pixel line in one single block and the reception buffer won't overflow.
- The DSI output bandwidth is much higher than the DPI input bandwidth, which allows the lane to go to LP state after each line.

If any of the requirements cannot be met, Non-Burst Mode should be selected. In Non-Burst Mode, the DSI host should be configured to match DSI output ratio and DPI input ratio. The following equation shows how the transmission parameters should be configured (The left side represents the DSI output ratio, and the right side represents the DPI input ratio):

- If null packets are enabled (DSI_HOST_VID_NULL_SIZE $\neq$ 0):

$$\begin{aligned} & lanebyteclk_period \times DSI_HOST_VID_NUM_CHUNKS \\ & \quad \times (DSI_HOST_VID_PKT_SIZE \times bytes_per_pixel + 12 \\ & \quad + DSI_HOST_VID_NULL_SIZE)/number_of_lanes \\ & = dpiclk_period \times pixels_per_line \end{aligned}$$

- If null packets are disabled (DSI_HOST_VID_NULL_SIZE $=$ 0):

$$\begin{aligned} & lanebyteclk_period \times DSI_HOST_VID_NUM_CHUNKS \\ & \quad \times (DSI_HOST_VID_PKT_SIZE \times bytes_per_pixel + 6) \\ & /number_of_lanes = dpiclk_period \times pixels_per_line \end{aligned}$$

Refer to 43.4.4 for the definitions of lanebyteclk and dpiclk.

43.4.2.2 Command Transmission

DSI Host allows the transmission of generic commands described in DSI specification as manufacturer-specific in Command Mode through APB Slave Interface.

The command packets are built using the APB registers:

- DSI_HOST_GEN_PLD_DATA_REG: Writing to this register sends the data as the payload of a command packet. Reading this register returns the payload of read-back packets.
- DSI_HOST_GEN_HDR_REG: Contains the command packet header type and header data. Writing to this register triggers a command transmission.

The valid packets are as follows:

- Generic Write Short Packet 0 Parameters
- Generic Write Short Packet 1 Parameters
- Generic Write Short Packet 2 Parameters
- Generic Read Short Packet 0 Parameters
- Generic Read Short Packet 1 Parameters
- Generic Read Short Packet 2 Parameters
- Maximum Read Packet Configuration
- Generic Long Write Packet

Command packets can be transmitted in lane HS or LP states, while in video mode Blanking or Low-Power periods (BLLP). BLLP corresponds to the shaded areas in figure 43.4-1.

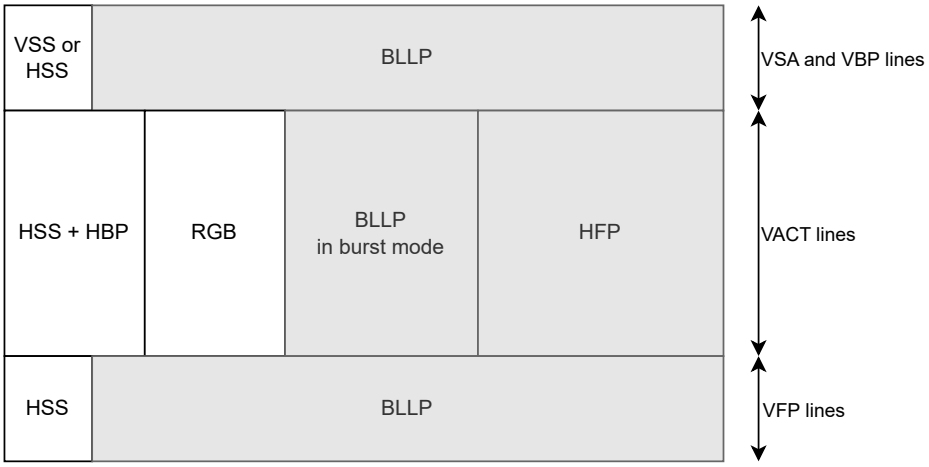


Figure 43.4-1. Command Transmission Periods

43.4.2.2.1 Command Transmission in LP mode

To enable sending commands in LP state during video transmission, set DSI_HOST_LP_CMD_EN to 1. In this case, even if bits 8 to 13 of DSI_HOST_VID_MODE_CFG_REG are set to 0, commands can still be sent during these periods.

- DSI_HOST_OUTVACT_LPCMD_TIME configures the time available (in bytes) to transmit commands in LP mode (based on escape clock) during VSA, VBP, and VFP. The outvact_lpcmd_time for Non-Burst with

Sync Pulse is shown in figure 43.4-2, and the outvact_lpcmd_time for Burst or Non-Burst with Sync Event is shown in figure 43.4-3.

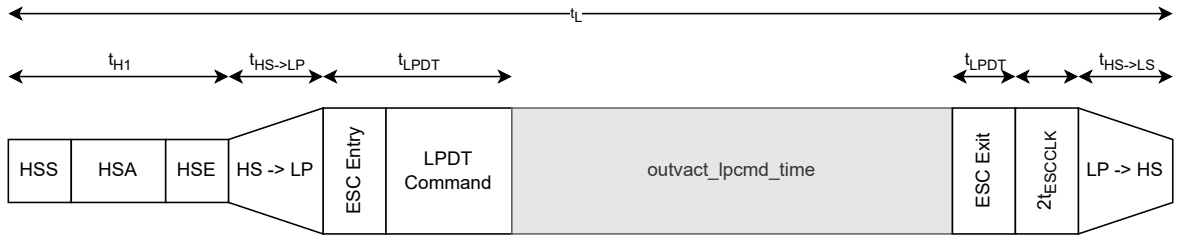


Figure 43.4-2. outvact_lpcmd_time for Non-Burst with Sync Pulse

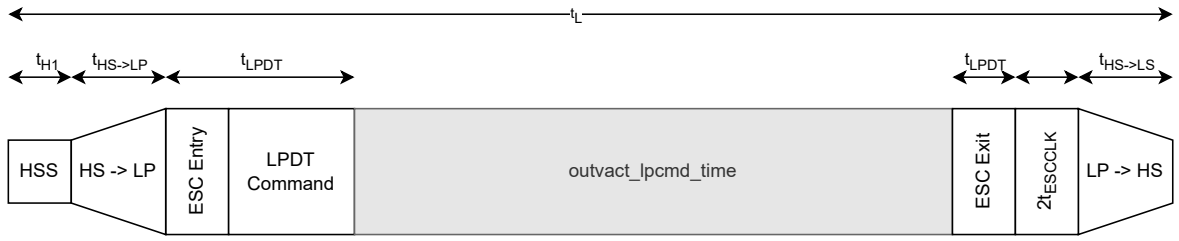


Figure 43.4-3. outvact_lpcmd_time for Burst or Non-Burst with Sync Event

The outvact_lpcmd_time is calculated by:

$$outvact_lpcmd_time = t_L - (t_{H1} + t_{HS \rightarrow LP} + t_{LP \rightarrow HS} + t_{LPDT} + 2 \times t_{ESCCLK}) / (2 \times 8 \times t_{ESCCLK})$$

- DSI_HOST_INVACT_LPCMD_TIME configures the time available (in bytes) to transmit commands in LP mode (based on escape clock) during HFP. The invact_lpcmd_time for Non-Burst with Sync Pulse is shown in figure 43.4-4; and the invact_lpcmd_time for Non-Burst with Sync Event is shown in figure 43.4-5.

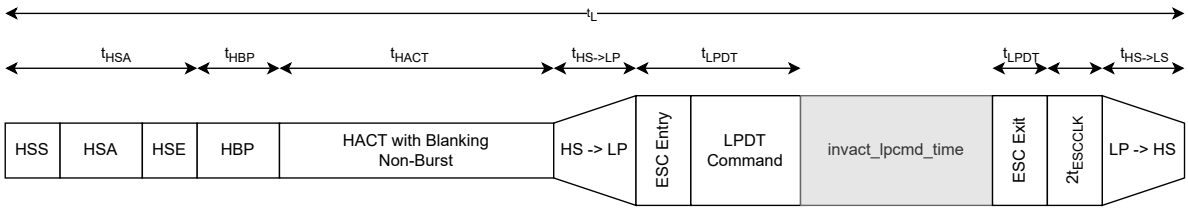


Figure 43.4-4. invact_lpcmd_time for Non-Burst with Sync Pulse

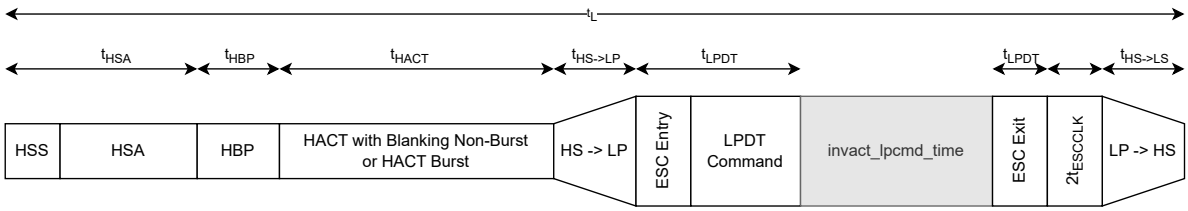


Figure 43.4-5. invact_lpcmd_time for Burst or Non-Burst with Sync Event

The invact_lpcmd_time is calculated by:

$$invact_lpcmd_time = t_L - (t_{HSA} + t_{HBP} + t_{HACT} + t_{HS \to LP} + t_{LP \to HS} + t_{LPDT} + 2 \times t_{ESCCLK}) / (2 \times 8 \times t_{ESCCLK})$$

43.4.2.2.2 Command Transmission in HS mode

If DSI_HOST_LP_CMD_EN is set to 0, the commands are sent in HS mode during video transmission. In this case, no programming is required.

43.4.2.3 Video Pattern Generator

The Video Pattern Generator can output horizontal or vertical color bar or BER testing pattern. For the pattern resolution, refer to Clocks and Display Resolution. The following register fields are used to configure the Video Pattern Generator:

- DSI_HOST_VPG_ORIENTATION: Indicates the orientation of the color bar
- DSI_HOST_VPG_MODE: Selects the pattern
- DSI_HOST_VPG_EN: Enables the video pattern generator

43.4.2.3.1 Color Bar Pattern

The color bar pattern has eight color bars: white, yellow, cyan, green, magenta, red, blue, and black. Each color width can be calculated by dividing the pattern size by eight. If the size is not divisible by eight, the last color black contains the remaining size. Figure 43.4-6 and 43.4-7 show the vertical and horizontal color bars.

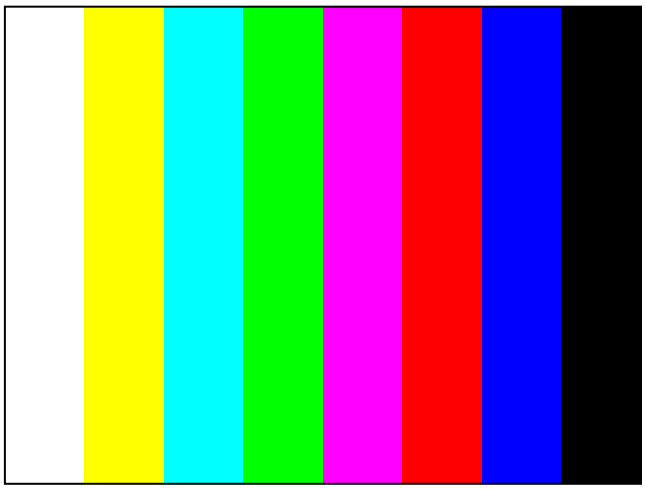


Figure 43.4-6. Vertical Color Bar

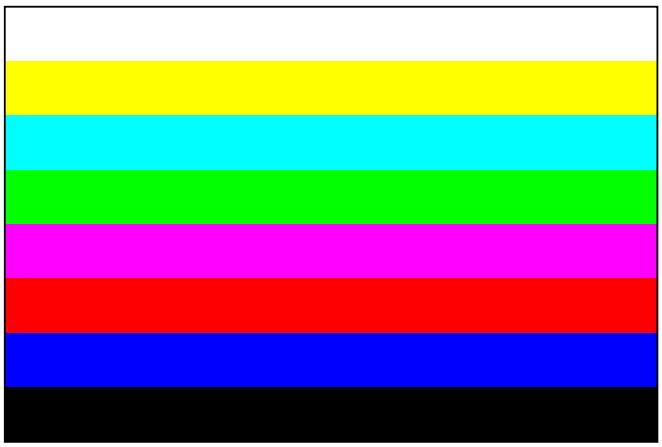


Figure 43.4-7. Horizontal Color Bar

Table 43.4-3 shows the color bar pattern RGB coding.

Table 43.4-3. Color Bar Coding

	White	Yellow	Cyan	Green	Magenta	Red	Blue	Black
R	High	High	Low	Low	High	High	Low	Low
G	High	High	High	High	Low	Low	Low	Low
B	High	Low	High	Low	High	Low	High	Low

43.4.2.3.2 BER Testing Pattern

The BER testing pattern tests whether the device RX PHY receives data correctly. The following data patterns are generated:

- X bytes of 0xAA (high-frequency pattern)
- X bytes of 0x33 (mid-frequency pattern)
- X bytes of 0xF0 (low-frequency pattern)
- X bytes of 0x7F (lone 0 pattern)
- X bytes of 0x55 (high-frequency pattern)
- X bytes of 0xCC (mid-frequency pattern)
- X bytes of 0x0F (low-frequency pattern)
- Y bytes of 0x80 (lone 1 pattern)

X and Y depend on pattern resolution, and Y equals X in most cases. With RGB888 color coding and horizontal resolution in multiples of eight, Figure 43.4-8 shows the pattern that appears on the DSI display:

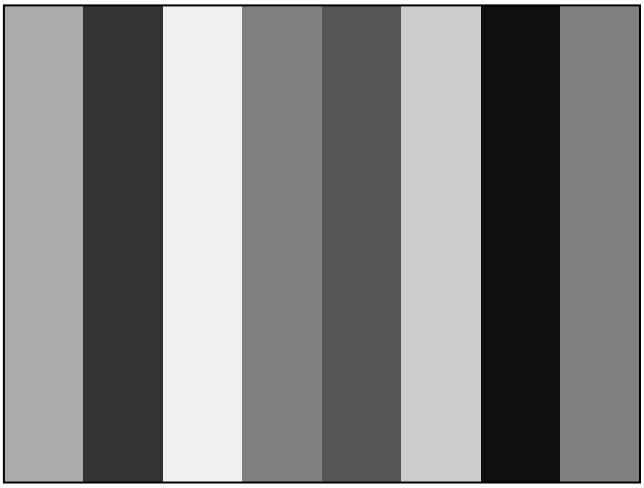


Figure 43.4-8. RGB888 BER Testing Pattern

43.4.2.4 Errors

The DSI_HOST_INT_ST0/1_REG registers indicate the DSI Host error reports. These registers can also trigger an interrupt to notify the system. To trigger the error interrupt, the corresponding bit in mask registers DSI_HOST_INT_MSK0/1_REG should be cleared. The DSI_HOST_INT_ST0/1_REG will be cleared after a read operation.

43.4.3 D-PHY

This section describes the D-PHY operating modes and PLL configurations.

43.4.3.1 D-PHY Operating Modes

Figure 43.4-9 shows the PHY initialization and active operating modes.

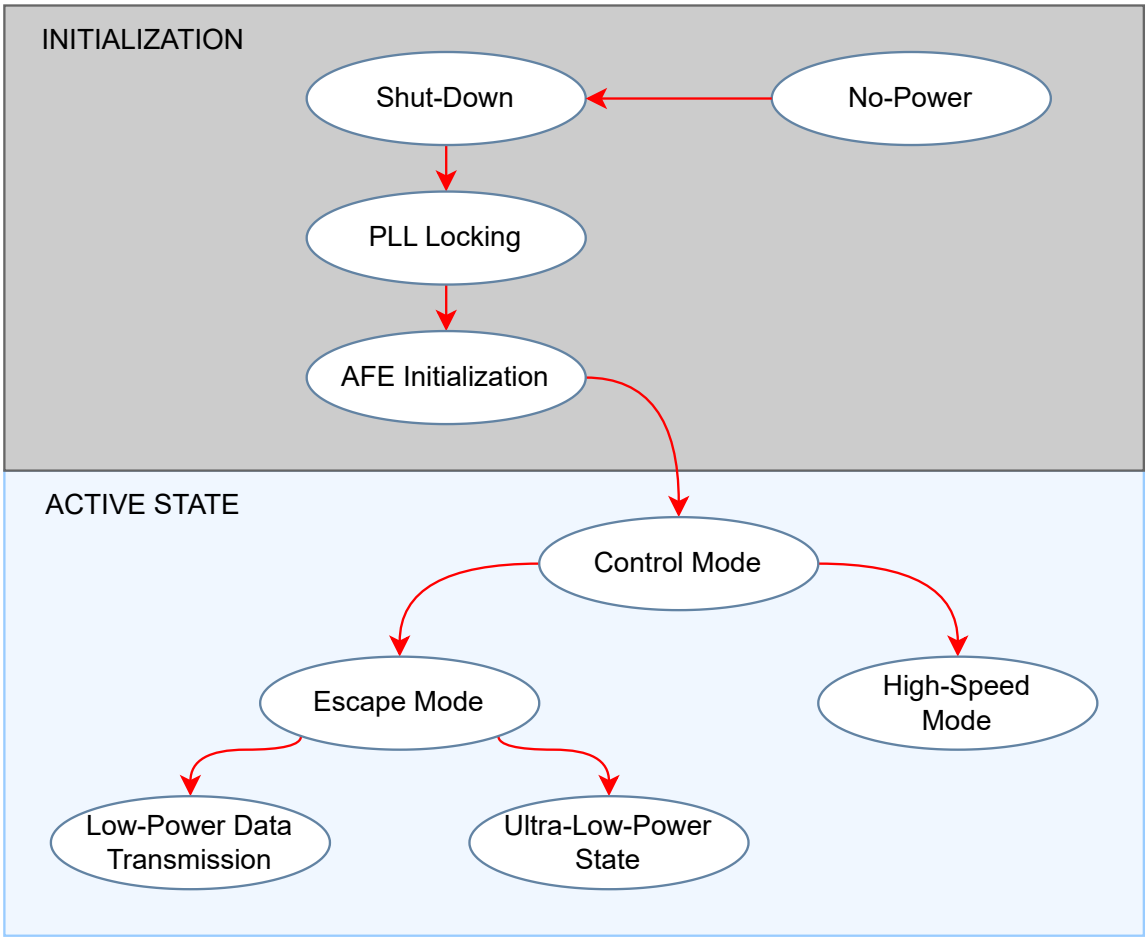


Figure 43.4-9. PHY operating modes

43.4.3.1.1 No Power State

No-Power state is characterized by the absence of any supply voltage applied to the PHY.

For ESP32-P4's MIPI DSI, MIPI BD D-PHY is in No Power state when the power supply VDD_HP_*n* (n: 0-3) and VDD_MIPI_DPHY are off.

43.4.3.1.2 Shutdown State

In Shutdown state, all analog blocks are disabled, all digital logic is reset, and DP/DN and CP/CN lane are high impedance (Hi-Z). To enable shutdown Mode of the PHY, clear DSI_HOST_PHY_RSTZ and DSI_HOST_PHY_SHUTDOWNZ to 0.

Before leaving the Shutdown state, some parameters should be configured, including the number of lanes to be used, the lane operation frequency range, the PLL configurations.

The MIPI BD D-PHY consists of two data lanes. Configure DSI_HOST_N_LANES to select either one or both data lanes for use.

By default, the PHY is set to operate within a lower range of 80-110 Mbps. To accommodate higher bit rates,

you can adjust the HS frequency ranges (the hsfreqrange field) in test code test code 0x44 with the proper code. It is recommended to perform this adjustment in Shutdown state if the PHY works always at the same bit rate, as the control interface is independent of the rest of the MIPI BD D-PHY. The supported ranges are shown in table 43.4-4:

Table 43.4-4. Frequency Ranges Supported by MIPI BD D-PHY

Frequency Range (Mbps)	hsfreqrange[5:0]*
80-89	000000
90-99	010000
100-109	100000
110-129	000001
130-139	010001
140-149	100001
150-169	000010
170-179	010010
180-199	100010
200-219	000011
220-239	010011
240-249	100011
250-269	000100
270-299	010100
300-329	000101
330-359	010101
360-399	100101
400-449	000110
450-499	010110
500-549	000111
550-599	010111
600-649	001000
650-699	011000
700-749	001001
750-799	011001
800-849	101001
850-899	111001
900-949	001010
950-999	011010
1000-1049	101010
1050-1099	111010
1100-1149	001011
1150-1199	011011
1200-1249	101011
1250-1299	111011
1300-1349	001100
1350-1399	011100

Continued on next page

Table 43.4-4 – continued from previous page

Frequency Range (Mbps)	hsfreqrange[5:0]*
1400-1449	101100
1450-1500	111100

It is also necessary to configure PLL input and output frequency through test code interface. Refer to PLL parameters for detailed description of PLL parameters.

When DSI_HOST_PHY_RSTZ and DSI_HOST_PHY_SHUTDOWNZ are set to 1, the MIPI BD D-PHY leaves the Shutdown state and starts an initialization procedure.

43.4.3.1.3 PLL-Locking and AFE Initialization

After DSI_HOST_PHY_RSTZ and DSI_HOST_PHY_SHUTDOWNZ are released, the PHY begins an initialization sequence to enable its normal operation. It is recommended to release DSI_HOST_PHY_SHUTDOWNZ before releasing DSI_HOST_PHY_RSTZ. Additionally, ensure that DSI_DPHY_CFG_CLK and DSI_DPHY_PLL_REFCLK (refer to Table 11.2-5 in Chapter 11 *Reset and Clock* for details) are available and stable at this point.

After that, the PLL becomes active and the PHY waits for the PLL lock which is indicated by DSI_HOST_PHY_LOCK.

Following the PLL lock, AFE Initialization is performed. All initialization steps are done when DSI_HOST_PHY_STOPSTATECLKLANE DSI_HOST_PHY_STOPSTATEnLANE are asserted and the lanes are in stop state (LP-11).

43.4.3.1.4 Control Mode

After the AFE initialization is completed, the MIPI BD D-PHY remains in Control mode by default while driving the stop state LP-11 on the lines until a request is made. Any request must start from and end in this state. After a request, a lane can enter High-Speed Data Transfer mode, Escape mode, Ultra Low Power state, or turnaround.

43.4.3.1.5 High-Speed Data Transfer

DSI HOST sends a request for High-Speed Data Transfer after receiving data from DSI Bridge. In High-Speed Data Reception mode, the PHY transmits image data in bursts. The lane operates in high-speed mode only during these bursts.

As shown in Figure 43.4-10, a burst on lane consists of three parts: the low-power initialization sequence, the high-speed data payload, and the end of reception sequence.

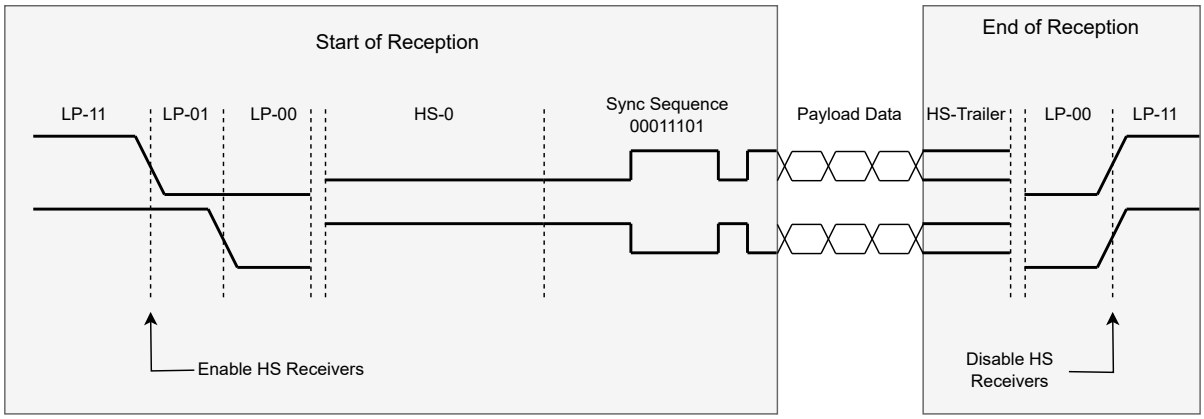


Figure 43.4-10. MIPI DSI HS Data Reception Sequence

43.4.3.1.6 Escape Mode

Escape mode allows asynchronous communication over data lanes at low speed. During Escape mode, the lane remains in low-power state. MIPI DSI in ESP32-P4 supports Low-Power Data Transmission (LPDT) and Ultra Low Power State (ULPS) in Escape mode.

Commands will be sent in LPDT if DSI_HOST_LP_CMD_EN is set to 1.

You can drive the PHY into ULPS through DSI_HOST_PHY_ULPS_CTRL_REG when DSI is idle to save power. The lane state can be observed through DSI_HOST_PHY_STATUS_REG.

43.4.3.1.7 Turnaround

The PHY can swap transmission direction through turnaround. This is used to receive command response from DSI devices.

43.4.3.2 PLL Parameters

The VCO output frequency of the PLL is a function of the frequency of input reference clock DSI_DPHY_PLL_REFCLK, as well as the multiplication ratio M, and the division ratio N:

$$f_{VCO} = \frac{M}{N} \times f_{REFCLK}$$

And the following limit must be met:

$$40\,\text{MHz} \geq \frac{f_{REFCLK}}{N} \geq 5\,\text{MHz}$$

Additionally, M must be an even number. The values of M and N can be programmed through test code 0x17 and test code 0x18. The programmed value is equal to the actual value minus 1, as shown in table 43.4-5. After that, bit-5 and bit-4 of test code 0x19 should be set to 1 to make the M and N configuration valid.

Table 43.4-5. M and N ratios

m[8:0] = M-1	M	n[6:0] = N-1	N
9'h1	2	7'h0	1
9'h3	4	7'h1	2
...	...	...	...

The VCO range of f_{VCO} can also be selected through test code 0x10. The valid VCO ranges are in table 43.4-6.

Table 43.4-6. VCO ranges

vcorange[2:0]	VCO Frequency (MHz)
000	80-200
001	200-300
010	300-500
011	500-700
100	700-900
101	900-1100
110	1100-1300
111	1300-1500

The loop bandwidth should be configured through test code interface depending on the output frequency to ensure proper operation of the PLL. The related parameters are listed below, and the valid value are listed in table 43.4-7.

- VCO capacitance (vcocap[1:0]), program through test code 0x10.
- CP current (icpctrl[3:0]), program through test code 0x11.
- LPF characteristic (lpfctrl[5:0]), program through test code 0x12.

Table 43.4-7. PLL CP and LPF Control Bits

VCO fmin (MHz)	VCO fmax (MHz)	vcorange[2:0]	icpctrl[3:0]	lpfctrl[5:0]	vcocap[1:0]
80	110	000	0100	010000	00
110	150		1000	001000	00
150	200		1100	001000	00
200	250	001	1100	001000	00
250	300		0010	001000	00
300	400	010	0101	001000	00
400	500		0101	000001	00
500	600	011	0110	010000	00
600	700		0110	000100	00
700	900	100	0110	000100	00
900	1100	101	0111	010000	00
1100	1300	110	0111	001000	00

Continued on next page

Table 43.4-7 – continued from previous page

VCO fmin (MHz)	VCO fmax (MHz)	vcorange[2:0]	icpctrl[3:0]	lpfctrl[5:0]	vcocap[1:0]
1300	1500	111	0111	001000	00

Note:

vcorange, icpctrl, lpfctrl are automatically set to the default values for the corresponding hsfreqrange, if the corresponding bit is 0.

- test code 0x10, bit7, clear to 0 to bypass vcorange
- test code 0x12, bit7, clear to 0 to bypass icpctrl
- test code 0x12, bit6, clear to 0 to bypass lpfctrl

43.4.3.3 Test Codes

The MIPI BD D-PHY provides a set of test codes for configuring MIPI BD D-PHY operation. The configuration can be done through Test Code Interface using DSI_HOST_PHY_TST_CTRL0_REG and DSI_HOST_PHY_TST_CTRL1_REG. For details of the Test Code Interface, refer to section 42.5.3.1 in Chapter 42 *MIPI CSI*.

All supported Test Codes are:

43.4.3.3.1 Normal Operation

This mode represents the normal operation of the PHY and also the wake up state of the test interface. In this mode, the test interface is inactive.

- **Test Code**: 0x00
- **Test Data**: None

43.4.3.3.2 PLL Bias Current Selector / Filter Capacitance Control / VCO Control

Controls the biasing of the PLL and the VCO transfer function.

- **Test Code**: 0x10
- **Test Data**:

Table 43.4-8. Test Data of 0x10

w-1'b0	w-1'b0	w-3'b000	w-2'b00	w-1'b1
Bypass VCO range	Reserved	VCO range control	VCO internal capacitance control	Reference bias current selector

  - Bit 7: Bypass VCO range
    * 0 - VCO range is programmed with the default value for the corresponding hsfreqrange
    * 1 - VCO range is programmed with bits 5-3

- Bit 6: Reserved
- Bits 5-3: VCO range control, refer to table 43.4-6 for details
- Bits 2-1: VCO internal capacitance control
  * 00 - Default capacitance
  * 01 - Low capacitance (four times lower than default value)
  * 10 - High capacitance (double of the default value)
  * 11 - Not allowed
- Bit 0: Reference bias current selector
  * 0 - Internal biasing
  * 1 - Clean external current source of 20uA

- **Test Dout**:

Table 43.4-9. Test Dout of 0x10

r-1'b0	r-1'b0	r-3'b000	r-2'b00	r-1'b1
Bypass VCO range loopback	1'b0	VCO range control loopback	VCO internal capacitance control loopback	Reference bias current selector loopback

43.4.3.3.3 PLL CP Control / PLL Lock Bypass for Initialization and for ULP

Controls the Charge-Pump (CP) current and the bypass of the PLL lock for initialization and ULP exit.

- **Test Code**: 0x11
- **Test Data**:

Table 43.4-10. Test Data of 0x11

w-4'b0000	w-4'b0010
PLL lock bypass	CP current

- Bits 7-4: PLL lock bypass
  * xxx1 - Bypass PLL lock during initialization
  * x11x - Bypass PLL lock after exiting ULP
- Bits 3-0: CP current (uA)
  * 0000 - 1.5
  * 0001 - 3
  * 0010 - 4.5
  * 0011 - 6
  * 0100 - 2.5

* 0101 - 5
* 0110 - 7.5
* 0111 - 10
* 1000 - 3
* 1001 - 6
* 1010 - 9
* 1011 - 12
* 1100 - 4
* 1101 - 8
* 1110 - 12
* 1111 - 16

- **Test Dout**:

Table 43.4-11. Test Dout of 0x11

r-1'b0	r-1'b0
PLL lock bypass loopback	CP current loopback

43.4.3.3.4 PLL LPF and CP control

Controls the Charge-Pump (CP) current and the Low-Pass-Filter (LPF) resistor.

- **Test Code**: 0x12
- **Test Data**:

Table 43.4-12. Test Data of 0x12

w-1'b0	w-1'b0	w-6'b000010
Bypass CP default values	Bypass LPF default values	LPF resistor

  - Bit 7: Bypass CP default values
    * 0 - CP is programmed with the default value for the corresponding hsfreqrange
    * 1 - CP is programmed with bits 3-0 using test code 0x11
  - Bit 6: Bypass LPF default values
    * 0 - LPF is programmed with the default value for the corresponding hsfreqrange
    * 1 - LPF is programmed with bits 5-0
  - Bits 5-0: LPF resistor ($k\Omega$)
    * 000000 - 18
    * 000001 - 15.6
    * 000010 - 15

* 000100 - 14.4
* 001000 - 12.8
* 010000 - 11.4
* 100000 - 10.5
* others - reserved

- **Test Dout**:

Table 43.4-13. Test Dout of 0x12

r-1'b0	r-1'b0	r-6'b000010
CP bypass loopbakc	LPF bypass loopback	LPF resistor loopback

43.4.3.3.5 PLL Input Divider Ratio

Controls the PLL's input divider N. $N = n + 1$, where N is the actual divider ratio, and n is the programmed value in the test code.

to enable the programmed value, ensure that bit-4 of test code 0x19 is set to 1.

- **Test Code**: 0x17
- **Test Data**:

Table 43.4-14. Test Data of 0x17

w-1'b0	w-7'b0000001
Reserved	Input divider

- **Test Dout**:

Table 43.4-15. Test Dout of 0x17

r-1'b0	r-7'b0000001
Reserved	Input divider loopback

43.4.3.3.6 PLL Loop Divider Ratio

Controls the PLL's feedback divider M. $M = m + 1$, where M is the actual feedback ratio, and m is the programmed value in the test code.

To enable the programmed value, ensure that bit-5 of test code 0x19 is set to 1.

- **Test Code**: 0x18
- **Test Data**:

Table 43.4-16. Test Data of 0x18

w-1'b0	w-2'b00	w-5'b00101
Set 0	Reserved	Feedback divider [4:0]

w-1'b1	w-3'b00	w-4'b0000
Set 1	Reserved	Feedback divider [8:5]

- Bit 7: Program selector
  - 0 - Set 0 is programmed
  - 1 - Set 1 is programmed
- Bits 6-5: Reserved
- Bits 4-0: Feedback divider

- **Test Dout**:
  TESTDIN[7] - Set 0

Table 43.4-17. Test Dout of 0x18

r-3'b000	r-5'b00101
3'b000	Feedback divider [4:0] loopback

TESTDIN[7] - Set 1

Table 43.4-18. Test Dout of 0x18

r-4'b0000	r-4'b0000
4'b0000	Feedback divider [8:5] loopback

43.4.3.3.7 PLL Input and Loop Divider Ratios Control

Controls the PLL input and loop divider ratios.

- **Test Code**: 0x19
- **Test Data**:

Table 43.4-19. Test Data of 0x19

w-2'b00	w-1'b0	w-1'b0	w-4'b0000
Reserved	Bypass PLL loop divider default values	Bypass PLL input divider default values	Reserved

- Bits 7-6: Reserved
- Bit 5: Bypass the PLL loop divider default values
  - 0 - PLL loop divider is programmed with the default values for the corresponding hsfreqrange
  - 1 - PLL loop divider is programmed using test code 0x18
- Bit 4: Bypass the PLL input divider default values
  - 0 - PLL input divider is programmed with the default values for the corresponding hsfreqrange
  - 1 - PLL input divider is programmed using test code 0x17

- Bits 3-0: Reserved

- **Test Dout**:

Table 43.4-20. Test Dout of 0x19

r-2'b00	r-1'b0	r-1'b0	r-4'b0000
Reserved	Bypass PLL loop divider default loopback	Bypass PLL input divider default loopback	Reserved

43.4.3.3.8 AFE/BIAS/Bandgap Analog Programmability

Controls the various parameters in the PHY.

- **Test Code**: 0x22
- **Test Data**:

Table 43.4-21. Test Data of 0x22

w-1'b0	w-7'b0000000	
Set 0	Analog programmability [6:0]	
w-1'b1	w-3'b00	w-4'b0000
Set 1	Reserved	Analog programmability [10:7]

  - Bit 7: Program selector
    - 0 - Set 0 is programmed (also selects bits 6:0 for observability)
    - 1 - Set 1 is programmed (also selects bits 10:7 for observability)
  - Bits 6-0: Analog programmability
    - Bit 10: 1.5Gbps support. Should be enabled for speed higher than 1Gbps and up to 1.5Gbps
      - 0 - 1.5Gbps analog circuitry support disabled
      - 1 - 1.5Gbps analog circuitry support enabled
    - Bits 9-7: Adjust bandgap reference voltage (typical corner)
      - 000 - 97.07%
      - 001 - 98.05%
      - 010 - 99.02%
      - 011 - 100.00%
      - 100 - 93.17%
      - 101 - 94.15%
      - 110 - 95.12%
      - 111 - 96.10%
    - Bit 6: Adjust LP RX bias current
      - 0 - 100%

        - 1 - 50%
    * Bits 5-3: Adjust LP TX bias current
        - 000 - 76.92%
        - 001 - 83.33%
        - 010 - 90.91%
        - 011 - 100.00%
        - 100 - 111.10%
        - 101 - 125.00%
        - 110 - 142.80%
        - 111 - 166.70%
    * Bits 2-0: Biasextr: adjust internal resistor
        - 000 - 87.1%
        - 001 - 91.5%
        - 010 - 95.9%
        - 011 - 100%
        - 100 - 105.94%
        - 101 - 111.88%
        - 110 - 118.8%
        - 111 - 127.7%

- **Test Dout**:
  TESTDIN[7] - Set 0

Table 43.4-22. Test Dout of 0x22

r-1'b0	r-7'b0000000
1'b0	Analog programmability [6:0] loopback

TESTDIN[7] - Set 1

Table 43.4-23. Test Dout of 0x22

r-4'b0000	r-4'b0000
4'b0000	Analog programmability [10:7] loopback

43.4.3.3.9 HS RX Control of Lane 0

Controls the hsfreqrange.

- **Test Code**: 0x44

- **Test Data**:

Table 43.4-24. Test Data of 0x44

w-1'b0	w-6'b000000	w-1'b0
Set 2	HS operating frequency range selection (hsfreqrange)	Bypass HS-RX settle filter

  - Bits 7: Program selector
    - 0 - Set 2 is programmed
    - 1 - Invalid
  - Bit 6-1: HS frequency range selection. Refer to table 43.4-4 for detailed values.
  - Bits 0: Bypass HS settle filter

- **Test Dout**:

Table 43.4-25. Test Dout of 0x44

r-1'b0	r-6'b000000	r-1'b0
1'b0	HS operating frequency range selection loopback (hsfreqrange)	Bypass HS-RX settle filter loopback

43.4.4 Clocks and Display Resolution

The clock architecture of DSI is shown in figure 43.4-11. All clocks are:

- **PCLK**: the apb bus clock
- **ACLK**: the axi bus clock
- **DSI_DPICLK**: dpi interface clock
- **DSI_DPHY_CFG_CLK**: DPHY configuration clock
- **DSI_DPHY_PLL_REFCLK**: DPHY PLL input reference clock
- **TX_HS_CLK**: DPHY PLL output clock
- **TX_BYTE_CLK**: High-Speed (HS) transfer byte clock, TX_HS_CLK divided by 8.
- **TX_ESC_CLK**: Low-Power (LP) escape mode transfer clock, divided from TX_BYTE_CLK. The divider ratio can be configured through DSI_HOST_TO_CLK_DIVISION, and the allowed max-frequency is 20 MHz.

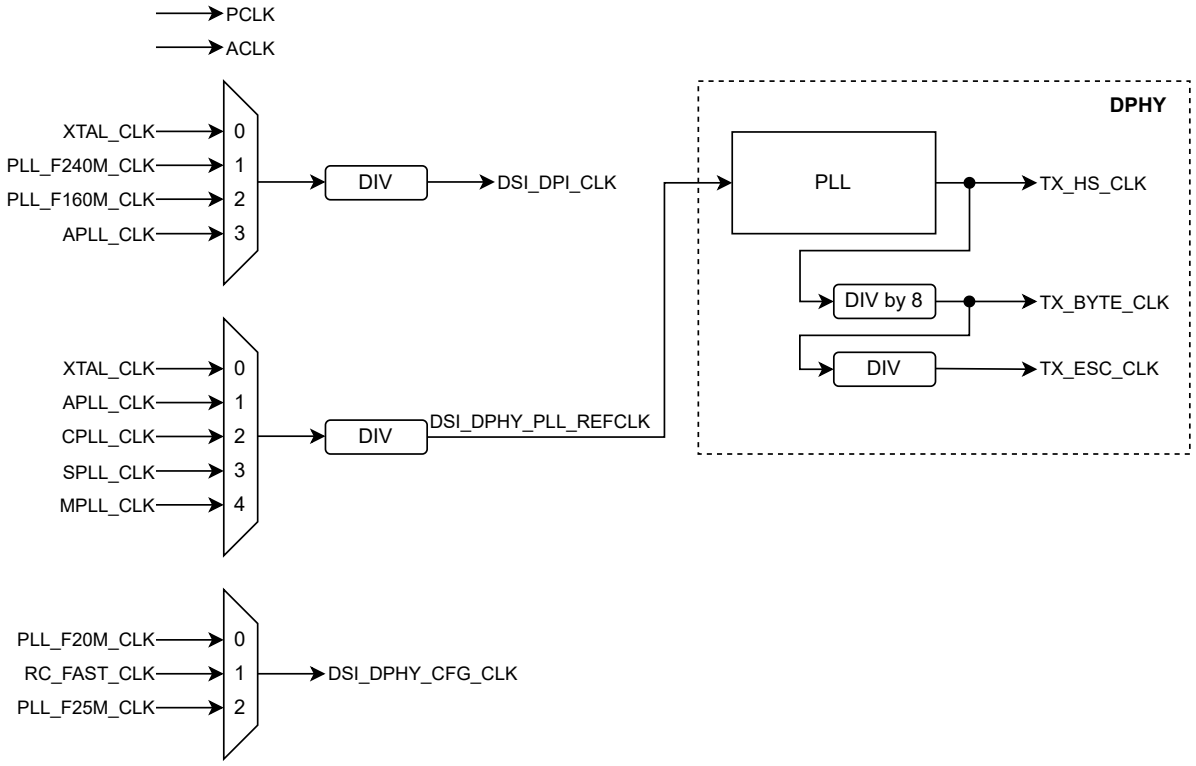


Figure 43.4-11. MIPI DSI clock architectural

DSI supports continuous and non-continuous clock lane

- For continuous clock, set DSI_HOST_AUTO_CLKLANE_CTRL to 1, set DSI_HOST_PHY_TXREQUESTCLKHS to 1
- For non-continuous clock, set DSI_HOST_AUTO_CLKLANE_CTRL to 0, set DSI_HOST_PHY_TXREQUESTCLKHS to 1

In video mode transmission, the video timing parameters are shown in figure 43.4-12.

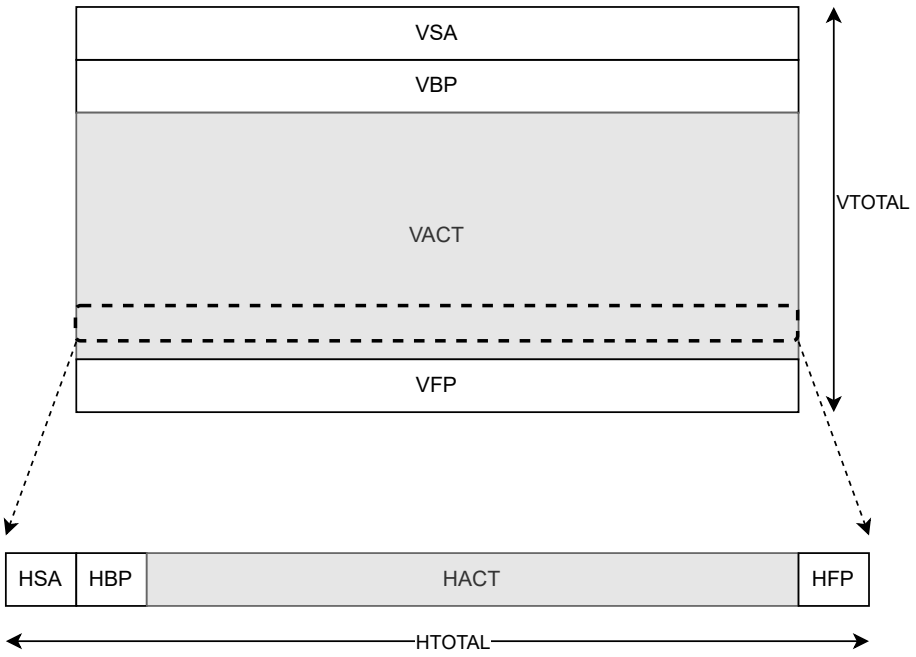


Figure 43.4-12. Video Timing

For DPI, the video timing and resolution can be configured through the following DSI Bridge register fields. After that, write 1 to DSI_BRG_DPI_CONFIG_UPDATE to make the configuration valid.

- DSI_BRG_HTOTAL: The HTOTAL length measured in DSI_DPICLK
- DSI_BRG_HDISP: The HACT length measured in DSI_DPICLK
- DSI_BRG_HBANK: The HBP length measured in DSI_DPICLK
- DSI_BRG_HSYNC: The HSA length measured in DSI_DPICLK
- DSI_BRG_VTOTAL: The VTOTAL length measured in lines
- DSI_BRG_VDISP: The VACT length measured in lines
- DSI_BRG_VBANK: The VBP length measured in lines
- DSI_BRG_VSYNC: The VSA length measured in lines

Similar configuration for video timing and resolution is required for DSI, and the related registers fields are listed below:

- DSI_HOST_VID_HSA_TIME: The HSA length measured in TX_BYTE_CLK
- DSI_HOST_VID_HBP_TIME: The HBP length measured in TX_BYTE_CLK
- DSI_HOST_VID_HLINE_TIME: The HTOTAL length measured in TX_BYTE_CLK
- DSI_HOST_VSA_LINES: The VSA length measured in lines
- DSI_HOST_VBP_LINES: The VBP length measured in lines
- DSI_HOST_VFP_LINES: The VFP length measured in lines
- DSI_HOST_VACTIVE_LINES: The VACT length measured in lines

The video timing configuration of DPI and DSI should match to avoid line buffer overflow. The actual timing of each line in DPI and DSI should be the same.

43.5 Interrupts

ESP32-P4's MIPI DSI can generate the following interrupt signals, which will be sent to the Interrupt Matrix.

- dsi_bridge_interrupt
- dsi_interrupt

The following interrupt sources can generate the dsi_bridge_interrupt interrupt signal:

Table 43.5-1. DSI Bridge interrupts

Interrupt Source	Detail
DSI_BRG_UNDERRUN_INT	Triggered when DSI Bridge buffer underruns.

The following interrupt sources can generate the dsi_interrupt interrupt signal:

Table 43.5-2. DSI Host Interrupts

Interrupt	Cause of Interrupt	Recommended Handing Method
DSI_HOST_ACK_WITH_ERR_0	SoT error from the Acknowledge error report	Transmit the packet again if the packet is critical
DSI_HOST_ACK_WITH_ERR_1	SoT Sync error from the Acknowledge error report	Transmit the packet again
DSI_HOST_ACK_WITH_ERR_2	EoT Sync error from the Acknowledge error report	Transmit the packet again
DSI_HOST_ACK_WITH_ERR_3	Escape Mode Entry Command error from the Acknowledge error report, the device does not recognize the Escape Mode Entry code	Check the device capability
DSI_HOST_ACK_WITH_ERR_4	Low-power Transmit Sync error from the Acknowledge error report	Transmit the packet again

Continued on the next page...

Table 43.5-2 – ...Continued from the previous page

Error Name	Cause of the Error	Recommended Method of Handing the Error
DSI_HOST_ACK_WITH_ERR_5	Peripheral Timeout error from the Acknowledge error report	Check whether the Host timeout counter configuration is consistent with the Device specifications; Host timeout counter configuration should be longer than the Deivice. This error is automatically recovered, but with no guarantee that the transmission or reception is complete
DSI_HOST_ACK_WITH_ERR_6	False Control error from the Acknowledge error report	Reset the DSI Host and DPHY
DSI_HOST_ACK_WITH_ERR_7	Reserved (specific to device) from the Acknowledge error report	Check the Device capabilities and connectivity and transmit again
DSI_HOST_ACK_WITH_ERR_8	ECC error, single-bit (detected and corrected) from the Acknowledge error report	No action is required. Analyze the signal integrity or the noise conditions of the link if this error is recurrent
DSI_HOST_ACK_WITH_ERR_9	ECC error, multi-bit (detected, not corrected) from the Acknowledge error report	Transmit the packets again if the packets are critical. Analyze the signal integrity or the nosie conditions of the link if this error is recurrent
DSI_HOST_ACK_WITH_ERR_10	Checksum error (long packet only) from the Acknowledge error report	Transmit the packets again if the packets are critical. Analyze the connectivity if this error is recurrent
DSI_HOST_ACK_WITH_ERR_11	Not recognized DSI data type from the Acknowledge error report. Some packets may not be supported	Check the Device capabilities
DSI_HOST_ACK_WITH_ERR_13	Invalid transmission length from the Acknowledge error report	Transmit the packets again. Analyze the connectivity if this error is recurrent
DSI_HOST_ACK_WITH_ERR_14	Reserved (specific to device) from the Acknowledge error report	Check the Device specification for possible reason and actions

Continued on the next page...

Table 43.5-2 – ...Continued from the previous page

Error Name	Cause of the Error	Recommended Method of Handing the Error
DSI_HOST_ACK_WITH_ERR_15	DSI protocol violation from the Acknowledge error report	Check the display specification. With this error, the Device should have another read-back command which reports additional error information
DSI_HOST_DPHY_ERRORS_0	ErrEsc escape entry error from Lane 0. The Escape Entry Code is not recognized	Repeat the reception
DSI_HOST_DPHY_ERRORS_1	ErrSyncEsc low-power data transmission synchronization error from Lane 0	Reset DSI Host and repeat the transmission
DSI_HOST_DPHY_ERRORS_2	Control error ErrControl from Lane 0	Reset DSI Host and DPHY
DSI_HOST_DPHY_ERRORS_3	LP0 contention error ErrContentionLP0 from Lane 0	Reset DSI Host and repeat the transmission
DSI_HOST_DPHY_ERRORS_4	LP1 contention error ErrContentionLP1 from Lane 0	Reset DSI Host and repeat the transmission
DSI_HOST_TO_HS_TX	High-speed transmission timeout error	Check the timeout configuration. Host timeout counter configuration should be longer than the Deivice
DSI_HOST_TO_LP_RX	Low-power reception timeout error	Check the timeout configuration. Host timeout counter configuration should be longer than the Deivice
DSI_HOST_ECC_SINGLE_ERR	ECC single error has been detected and corrected in a received packet	No action is required
DSI_HOST_ECC_MILTI_ERR	ECC multiple error has been detected in a received packet	Repeat the transmission
DSI_HOST_CRC_ERR	CRC error has been detected in the received packet payload	Reset DSI Host and repeat the transmission
DSI_HOST_PKT_SIZE_ERR	Packet size error has been detected during the packet reception	Reset DSI Host and repeat the transmission
DSI_HOST_EOPT_ERR	EoTp packet has not been received at the end of the incoming peripheral transmission	This error is not critical for data integrity of the received packet. Check if the Device supports EoTp packets

Continued on the next page...

Table 43.5-2 – ...Continued from the previous page

Error Name	Cause of the Error	Recommended Method of Handing the Error
DSI_HOST_DPI_PLD_WR_ERR	During a DPI pixel line storage, the payload FIFO becomes full and the data stored is corrupted	Check Video Mode configuration, and reset DSI Host and re-initiate the video transmission
DSI_HOST_GEN_CMD_WR_ERR	The system tried to write a command through the Generic interface and the FIFO is full. Therefore, the command is not written	Reset DSI Host and repeat the transmission
DSI_HOST_GEN_PLD_WR_ERR	The system tried to write a payload data through the Generic interface and the FIFO is full. Therefore, the payload is not written	Reset DSI Host and repeat the transmission
DSI_HOST_GEN_PLD_SEND_ERR	During a Generic interface packet build, the payload FIFO beomes empty and corrupt data is sent	Reset DSI Host and repeat the transmission
DSI_HOST_GEN_PLD_RD_ERR	During a DCS read data, the payload FIFO becomes empty and the data sent to the interface is corrupted	Reset DSI Host and repeat the transmission
DSI_HOST_GEN_PLD_RECEV_ERR	During a Generic interface packet read back, the payload FIFO becomes full and the received data is corrupted	Reset DSI Host and repeat the transmission
DSI_HOST_DPI_BUFF_PLD_UNDER	An underflow has occurred when reading payload to build DSI packet for video mode	Reset DSI Host and repeat the transmission

43.6 Programming Procedures

43.6.1 DSI Bridge Configuration

- Configure DSI Bridge clock and release reset.
- Configure input and output color mode. Refer to DSI Bridge for details.
- Configure video timing. Refer to Clocks and Display Resolution for details.
- Write 0 to DSI_BRG_DSI_DMA_FLOW_CONTROLLER to set VDMA as flow controller.
- Configure DSI_BRG_DMA_BURST_LEN to set the DMA burst length to match VDMA configuration.
- Configure DSI_BRG_DSI_RAW_BUF_ALMOST_EMPTY_THRD to

$1024 - DSI_BRG_DMA_BURST_LEN$ for better performance.

- Write 1 to DSI_BRG_DPI_EN to enable DPI.
- Write 1 to DSI_BRG_DSI_EN to enable DSI Bridge.
- Configure VDMA and set VDMA as flow controller. Refer to 7 for details.

43.6.2 D-PHY configuration

- Set DSI_HOST_PHY_SHUTDOWNZ and DSI_HOST_PHY_RSTZ to 0, set DSI_HOST_PHY_TESTCLR to 1.
- Apply an appropriate DSI_DPHY_CFG_CLK and DSI_DPHY_PLL_REFCLK.
- Configure the hsfreqrange through test code 0x44. Refer to table 43.4-4 for detailed value.
- Configure PLL parameters. Refer to PLL Parameters for details.
- Configure DSI_HOST_TO_CLK_DIVISION to apply an appropriate TX_ESC_CLK divider ratio.
- Set DSI_HOST_PHY_SHUTDOWNZ and DSI_HOST_PHY_RSTZ to 1.
- Waits for DSI_HOST_PHY_STOPSTATE0LANE and DSI_HOST_PHY_STOPSTATE1LANE asserted.

43.6.3 MIPI DSI common configuration

- Release global reset.
- Configure DSI_HOST_N_LANES to define the number of lanes to be used.
- Configure video timing settings. Refer to Clocks and Display Resolution for details.
- Initialize the DPHY through D-PHY configuration.
- Write DSI_HOST_SHUTDOWNZ to 1 to release DSI Host from shutdown.

43.6.4 System memory data output flow configuration

- Configure DSI Bridge through DSI Bridge configuration.
- Configure DSI Host through MIPI DSI common configuration.

43.6.5 Video pattern output flow configuration

- Configure pattern through register field DSI_HOST_VPG_ORIENTATION, DSI_HOST_VPG_MODE, and DSI_HOST_VPG_EN. Refer to Video Pattern Generator for details.
- Configure DSI Host through MIPI DSI common configuration.

43.6.6 Using DCS command to operate Display register

- Write to Display
  - Write packet payload to DSI_HOST_GEN_PLD_DATA_REG.
  - Write packet header to DSI_HOST_GEN_HDR_REG.
- Read from Display

- Write packet header to DSI_HOST_GEN_HDR_REG.
- Read response data from DSI_HOST_GEN_PLD_DATA_REG.

43.7 Register Summary

43.7.1 MIPI DSI Bridge Register Summary

The addresses in this section are relative to the MIPI DSI Bridge base address provided in Table 9.3-2 in Chapter 9 *System and Memory*.

The abbreviations given in Column **Access** are explained in Section *Access Types for Registers*.

Name	Description	Address	Access
Configuration Registers			
DSI_BRG_CLK_EN_REG	DSI Bridge clock enable register	0x0000	R/W
DSI_BRG_EN_REG	DSI Bridge enable register	0x0004	R/W
DSI_BRG_DMA_REQ_CFG_REG	DSI Bridge DMA burst length register	0x0008	R/W
DSI_BRG_RAW_NUM_CFG_REG	DSI Bridge raw pixel count control register	0x000C	varies
DSI_BRG_RAW_BUF_CREDIT_CTL_REG	DSI Bridge raw buffer credit register	0x0010	R/W
DSI_BRG_PIXEL_TYPE_REG	DSI Bridge DPI pixel type control register	0x0018	R/W
DSI_BRG_DMA_BLOCK_INTERVAL_REG	DSI Bridge DMA block interval control register	0x001C	R/W
DSI_BRG_DMA_REQ_INTERVAL_REG	DSI Bridge DMA request interval control register	0x0020	R/W
DSI_BRG_DPI_LCD_CTL_REG	DSI Bridge DPI signal control register	0x0024	R/W
DSI_BRG_DPI_RSV_DPI_DATA_REG	DSI Bridge DPI reserved data register	0x0028	R/W
DSI_BRG_DPI_V_CFG0_REG	DSI Bridge DPI vertical timing config register 0	0x0030	R/W
DSI_BRG_DPI_V_CFG1_REG	DSI Bridge DPI vertical timing config register 1	0x0034	R/W
DSI_BRG_DPI_H_CFG0_REG	DSI Bridge DPI horizontal timing config register 0	0x0038	R/W
DSI_BRG_DPI_H_CFG1_REG	DSI Bridge DPI horizontal timing config register 1	0x003C	R/W
DSI_BRG_DPI_MISC_CONFIG_REG	DSI Bridge DPI miscellaneous config register	0x0040	R/W
DSI_BRG_DPI_CONFIG_UPDATE_REG	DSI Bridge DPI config update register	0x0044	WT
DSI_BRG_HOST_TRIGGER_REV_REG	DSI Bridge host trigger reverse control register	0x0064	R/W
DSI_BRG_BLK_RAW_NUM_CFG_REG	DSI Bridge block raw pixel count control register	0x0068	varies
DSI_BRG_DMA_FRAME_INTERVAL_REG	DSI Bridge DMA frame interval control register	0x006C	R/W
DSI_BRG_MEM_AUX_CTRL_REG	DSI Bridge memory auxiliary control register	0x0070	R/W
DSI_BRG_RDN_ECO_LOW_REG	DSI Bridge RDN ECO low register	0x0078	R/W
DSI_BRG_RDN_ECO_HIGH_REG	DSI Bridge RDN ECO high register	0x007C	R/W
DSI_BRG_HOST_CTRL_REG	DSI Bridge host control register	0x0080	R/W
DSI_BRG_MEM_CLK_CTRL_REG	DSI Bridge memory clock force-on control register	0x0084	R/W
DSI_BRG_DMA_FLOW_CTRL_REG	DSI Bridge DMA flow controller register	0x0088	R/W
DSI_BRG_RAW_BUF_ALMOST_EMPTY_THRD_REG	DSI Bridge raw buffer almost-empty threshold register	0x008C	R/W
DSI_BRG_YUV_CFG_REG	DSI Bridge YUV format config register	0x0090	R/W
DSI_BRG_PHY_LP_LOOPBACK_CTRL_REG	DSI PHY LP loopback test control register	0x0094	varies
DSI_BRG_PHY_HS_LOOPBACK_CTRL_REG	DSI PHY HS loopback test control register	0x0098	varies
DSI_BRG_PHY_LOOPBACK_CNT_REG	DSI PHY loopback test counter register	0x009C	R/W

Name	Description	Address	Access
Status Registers			
DSI_BRG_FIFO_FLOW_STATUS_REG	DSI Bridge FIFO flow status register	0x0014	RO
DSI_BRG_HOST_BIST_CTL_REG	DSI Bridge host BIST control register	0x0060	varies
DSI_BRG_RDN_ECO_CS_REG	DSI Bridge RDN ECO chip select register	0x0074	varies
Interrupt Registers			
DSI_BRG_INT_ENA_REG	DSI Bridge interrupt enable register	0x0050	R/W
DSI_BRG_INT_CLR_REG	DSI Bridge interrupt clear register	0x0054	WT
DSI_BRG_INT_RAW_REG	DSI Bridge raw interrupt status register	0x0058	R/ WTC/ SS
DSI_BRG_INT_ST_REG	DSI Bridge masked interrupt status register	0x005C	RO
Version Register			
DSI_BRG_VER_DATE_REG	DSI Bridge version date register	0x0100	R/W

43.7.2 MIPI DSI Host Register Summary

The addresses in this section are relative to MIPI DSI Host base address provided in Table 9.3-2 in Chapter 9 *System and Memory*.

The abbreviations given in Column **Access** are explained in Section *Access Types for Registers*.

Name	Description	Address	Access
Version Register			
DSI_HOST_VERSION_REG	Version register	0x0000	RO
Configuration Registers			
DSI_HOST_PWR_UP_REG	DSI Host power up control register	0x0004	R/W
DSI_HOST_CLKMGR_CFG_REG	Internal divider control register	0x0008	R/W
DSI_HOST_DPI_VCID_REG	Virtual channel ID for DPI configuration register	0x000C	R/W
DSI_HOST_DPI_COLOR_CODING_REG	DPI color coding register	0x0010	R/W
DSI_HOST_DPI_CFG_POL_REG	DPI signal polarity register	0x0014	R/W
DSI_HOST_DPI_LP_CMD_TIM_REG	low-power command timing configuration register	0x0018	R/W
DSI_HOST_PCKHDL_CFG_REG	Packet header configuration register, configures how EoTp, BTA, CRC, ECC are to be used	0x002C	R/W
DSI_HOST_GEN_VCID_REG	Virtual channel ID of READ response for Generic Interface configuration register	0x0030	R/W
DSI_HOST_MODE_CFG_REG	Video or Command Mode configuration register	0x0034	R/W
DSI_HOST_VID_MODE_CFG_REG	Video Mode configuration register	0x0038	R/W
DSI_HOST_VID_PKT_SIZE_REG	Video packet size configuration register	0x003C	R/W
DSI_HOST_VID_NUM_CHUNKS_REG	Chunk number configuration register	0x0040	R/W
DSI_HOST_VID_NULL_SIZE_REG	Null packet size configuration register	0x0044	R/W
DSI_HOST_VID_HSA_TIME_REG	HSA configuration register	0x0048	R/W
DSI_HOST_VID_HBP_TIME_REG	HBP configuration register	0x004C	R/W
DSI_HOST_VID_HLINE_TIME_REG	HTOTAL configuration register	0x0050	R/W

Name	Description	Address	Access
DSI_HOST_VID_VSA_LINES_REG	VSA configuration register	0x0054	R/W
DSI_HOST_VID_VBP_LINES_REG	VBP configuration register	0x0058	R/W
DSI_HOST_VID_VFP_LINES_REG	VFP configuration register	0x005C	R/W
DSI_HOST_VID_VACTIVE_LINES_REG	VACT configuration register	0x0060	R/W
DSI_HOST_CMD_MODE_CFG_REG	Command Mode configuration register	0x0068	R/W
DSI_HOST_GEN_HDR_REG	Generic Interface packet header register	0x006C	R/W
DSI_HOST_GEN_PLD_DATA_REG	Generic Interface packet payload register	0x0070	R/W
DSI_HOST_TO_CNT_CFG_REG	Time out counter register	0x0078	R/W
DSI_HOST_HS_RD_TO_CNT_REG	HS read time out counter register	0x007C	R/W
DSI_HOST_LP_RD_TO_CNT_REG	LP read time out counter register	0x0080	R/W
DSI_HOST_HS_WR_TO_CNT_REG	HS write time out counter register	0x0084	R/W
DSI_HOST_LP_WR_TO_CNT_REG	LP write time out counter register	0x0088	R/W
DSI_HOST_BTA_TO_CNT_REG	Bus Turnaround time out counter register	0x008C	R/W
DSI_HOST_LPCLK_CTRL_REG	Non-continuous clock configuration register	0x0094	R/W
DSI_HOST_PHY_TMR_LPCLK_CFG_REG	Time of CLK lane switch between HS and LP configuration register	0x0098	R/W
DSI_HOST_PHY_TMR_CFG_REG	Time of DATA lane switch between HS and LP configuration register	0x009C	R/W
DSI_HOST_PHY_RSTZ_REG	DPHY reset control register	0x00A0	R/W
DSI_HOST_PHY_IF_CFG_REG	DPHY lane number and stop state configuration register	0x00A4	R/W
DSI_HOST_PHY_ULPS_CTRL_REG	DPHY ULPS control register	0x00A8	R/W
DSI_HOST_PHY_TX_TRIGGERS_REG	DPHY trigger register	0x00AC	R/W
DSI_HOST_PHY_TST_CTRL0_REG	DPHY test code clk and clear control register	0x00B4	R/W
DSI_HOST_PHY_TST_CTRL1_REG	DPHY test code data and enable control register	0x00B8	varies
DSI_HOST_PHY_CAL_REG	DPHY skew calibration control register	0x00CC	R/W
DSI_HOST_PHY_TMR_RD_CFG_REG	DPHY read timer configuration register	0x00F4	R/W
DSI_HOST_VID_SHADOW_CTRL_REG	DPI shadow control register	0x0100	R/W
Status Registers			
DSI_HOST_CMD_PKT_STATUS_REG	Generic Interface FIFO status register	0x0074	RO
DSI_HOST_PHY_STATUS_REG	DPHY status register	0x00B0	RO
DSI_HOST_DPI_VCID_ACT_REG	Actual VCID value register	0x010C	RO
DSI_HOST_DPI_COLOR_CODING_ACT_REG	Actual DPI_COLOR_CODING value register	0x0110	RO
DSI_HOST_DPI_LP_CMD_TIM_ACT_REG	Actual DPI_LP_CMD_TIM value register	0x0118	RO
DSI_HOST_VID_MODE_CFG_ACT_REG	Actual VID_MODE_CFG value register	0x0138	RO
DSI_HOST_VID_PKT_SIZE_ACT_REG	Actual VID_PKT_SIZE value register	0x013C	RO
DSI_HOST_VID_NUM_CHUNKS_ACT_REG	Actual VID_NUM_CHUNKS value register	0x0140	RO
DSI_HOST_VID_NULL_SIZE_ACT_REG	Actual VID_NULL_SIZE value register	0x0144	RO
DSI_HOST_VID_HSA_TIME_ACT_REG	Actual HSA value register	0x0148	RO
DSI_HOST_VID_HBP_TIME_ACT_REG	Actual HBP value register	0x014C	RO
DSI_HOST_VID_HLINE_TIME_ACT_REG	Actual HLINE value register	0x0150	RO
DSI_HOST_VID_VSA_LINES_ACT_REG	Actual VSA value register	0x0154	RO

Name	Description	Address	Access
DSI_HOST_VID_VBP_LINES_ACT_REG	Actual VBP value register	0x0158	RO
DSI_HOST_VID_VFP_LINES_ACT_REG	Actual VFP value register	0x015C	RO
DSI_HOST_VID_VACTIVE_LINES_ACT_REG	Actual VACT value register	0x0160	RO
DSI_HOST_VID_PKT_STATUS_REG	DPI FIFO status register	0x0168	RO
Interrupt Registers			
DSI_HOST_INT_ST0_REG	Interrupt status register	0x00BC	RO
DSI_HOST_INT_ST1_REG	Interrupt status register	0x00C0	RO
DSI_HOST_INT_MSK0_REG	Interrupt mask register	0x00C4	R/W
DSI_HOST_INT_MSK1_REG	Interrupt mask register	0x00C8	R/W

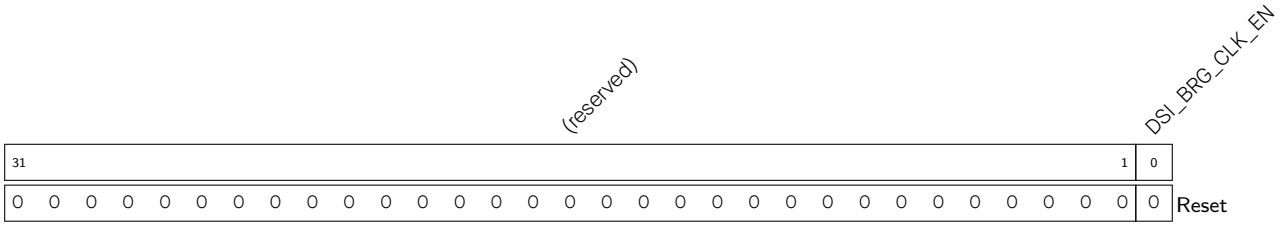
43.8 Registers

43.8.1 MIPI DSI Bridge Register

The addresses in this section are relative to MIPI DSI Bridge base address provided in Table 9.3-2 in Chapter 9 *System and Memory*.

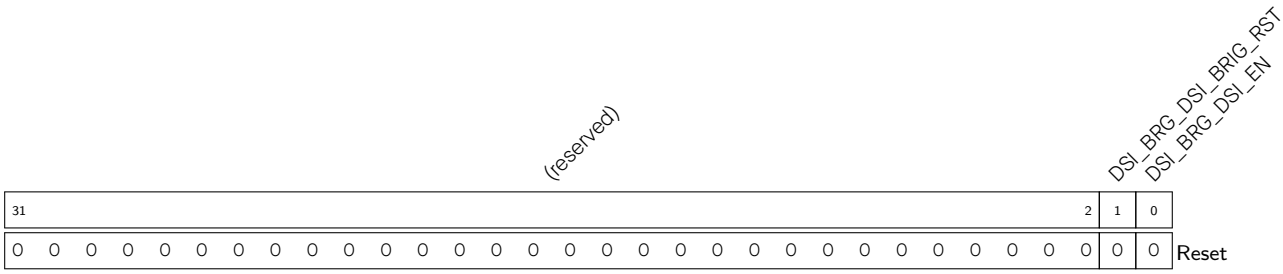
For how to program reserved fields, please refer to Section Programming Reserved Register Field.

Register 43.1. DSI_BRG_CLK_EN_REG (0x0000)



DSI_BRG_CLK_EN Configures force_on of dsi_bridge register clock gate. (R/W)

Register 43.2. DSI_BRG_EN_REG (0x0004)



DSI_BRG_DSI_EN Configures module enable of dsi_bridge.

0: disable

1: enable

(R/W)

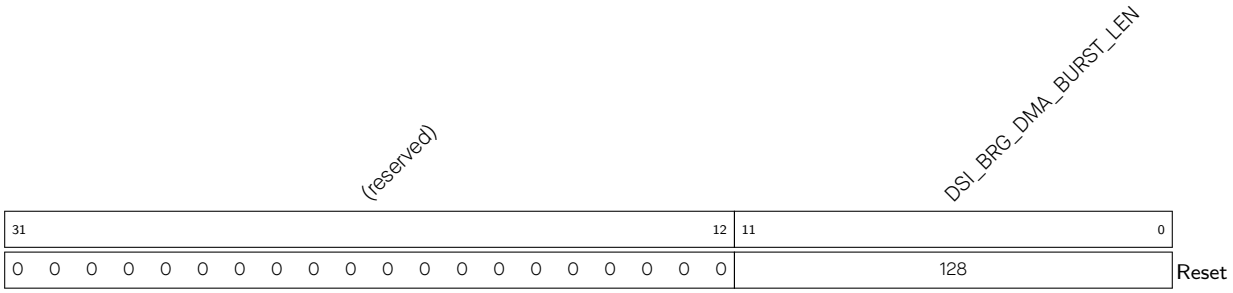
DSI_BRG_DSI_BRIG_RST Configures software reset of dsi_bridge.

0: release reset

1: reset

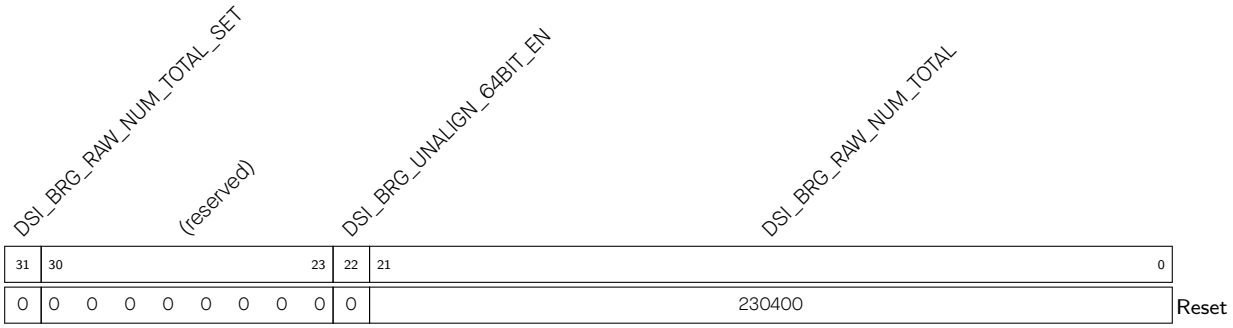
(R/W)

Register 43.3. DSI_BRG_DMA_REQ_CFG_REG (0x0008)



DSI_BRG_DMA_BURST_LEN Configures the number of 64-bit in one DMA burst transfer, valid only when dsi_bridge as flow controller. (R/W)

Register 43.4. DSI_BRG_RAW_NUM_CFG_REG (0x000C)



DSI_BRG_RAW_NUM_TOTAL Configures number of total pix bits/64. (R/W)

DSI_BRG_UNALIGN_64BIT_EN Configures whether the total pix bits is a multiple of 64bits.

0: align to 64-bit

1: unalign to 64-bit

(R/W)

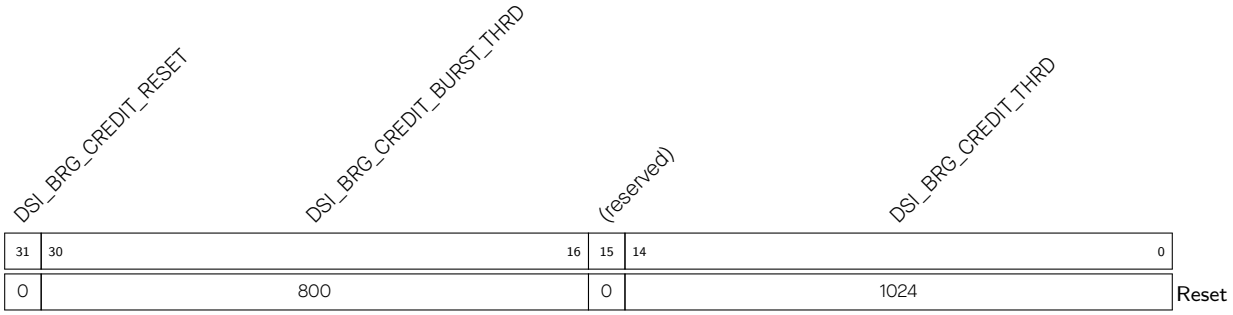
DSI_BRG_RAW_NUM_TOTAL_SET Configures enable of reloading reg_raw_num_total to the internal counter, valid only when dsi_bridge as flow controller.

0: disable

1: enable

(WT)

Register 43.5. DSI_BRG_RAW_BUF_CREDIT_CTL_REG (0x0010)



DSI_BRG_CREDIT_THRD Configures the threshold whether dsi_bridge FIFO can receive one more 64-bit, valid only when dsi_bridge as flow controller. (R/W)

DSI_BRG_CREDIT_BURST_THRD Configures the threshold whether dsi_bridge FIFO can receive one more DMA burst, valid only when dsi_bridge as flow controller. (R/W)

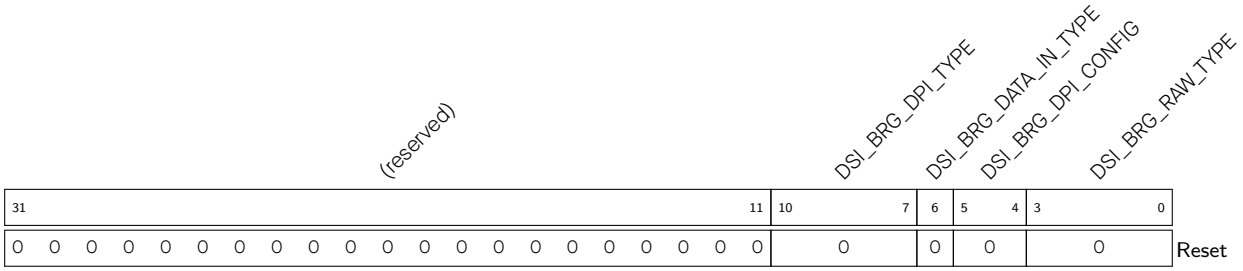
DSI_BRG_CREDIT_RESET Configures internal credit counter clear, valid only when dsi_bridge as flow controller.

0: no operation

1: reset

(R/W)

Register 43.6. DSI_BRG_PIXEL_TYPE_REG (0x0018)



DSI_BRG_RAW_TYPE Configures the input pixel type.

0: RGB888

1: RGB666

2: RGB565

8: YUV444

9: YUV422

10: YUV420

12: GRAY

(R/W)

DSI_BRG_DPI_CONFIG Configures the pixel arrange type of DPI interface. (R/W)

DSI_BRG_DATA_IN_TYPE Configures input data type.

0: RGB

1: YUV

(R/W)

DSI_BRG_DPI_TYPE Configures the DPI output pixel type.

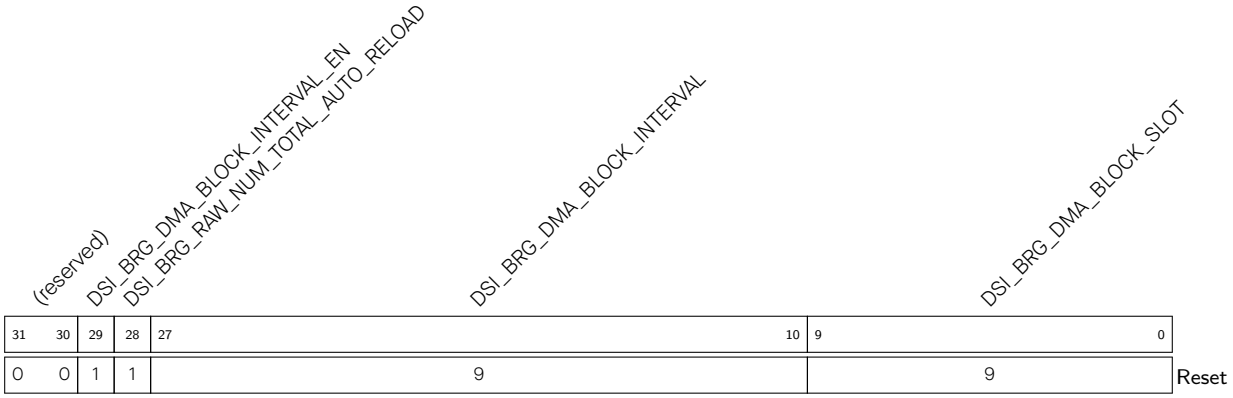
0: RGB888

1: RGB666

2: RGB565

(R/W)

Register 43.7. DSI_BRG_DMA_BLOCK_INTERVAL_REG (0x001C)



DSI_BRG_DMA_BLOCK_SLOT Configures the max block_slot_cnt. (R/W)

DSI_BRG_DMA_BLOCK_INTERVAL Configures the max block_interval_cnt, block_interval_cnt increased by 1 when block_slot_cnt if full. (R/W)

DSI_BRG_RAW_NUM_TOTAL_AUTO_RELOAD Configures enable of auto reload reg_raw_num_total.

0: disable

1: enable

(R/W)

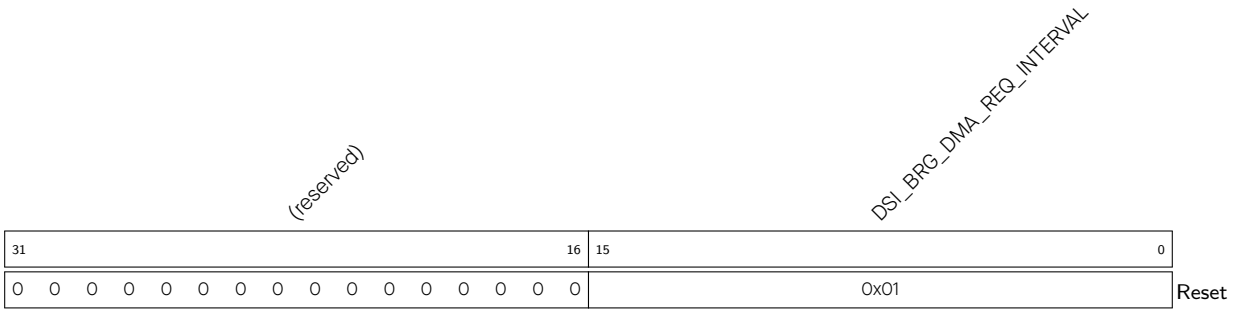
DSI_BRG_DMA_BLOCK_INTERVAL_EN Configures the DMA block transfer interval enable.

0: disable

1: enable

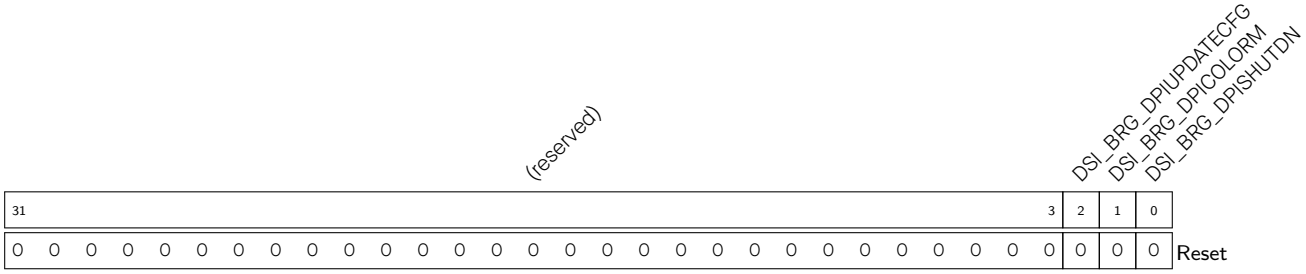
(R/W)

Register 43.8. DSI_BRG_DMA_REQ_INTERVAL_REG (0x0020)



DSI_BRG_DMA_REQ_INTERVAL Configures the interval between DMA request events. (R/W)

Register 43.9. DSI_BRG_DPI_LCD_CTL_REG (0x0024)

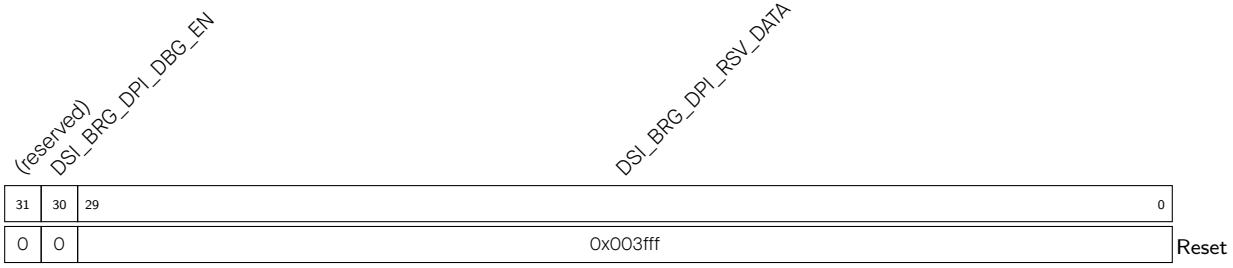


DSI_BRG_DPISHUTDN Configures dpishutdn signal in DPI interface. (R/W)

DSI_BRG_DPICOLORM Configures dpicolorm signal in DPI interface. (R/W)

DSI_BRG_DPIUPDATECFG Configures dpiupdatecfg signal in DPI interface. (R/W)

Register 43.10. DSI_BRG_DPI_RSV_DPI_DATA_REG (0x0028)



DSI_BRG_DPI_RSV_DATA Configures the pixel data sent to dsi_host when dsi_bridge FIFO underflow. (R/W)

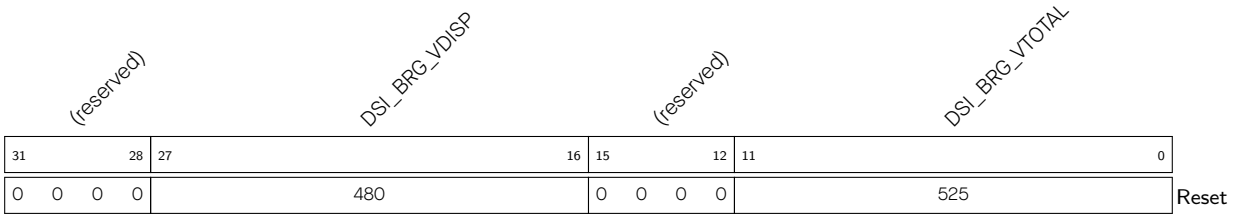
DSI_BRG_DPI_DBG_EN Configures data debug feature enable.

0: disable

1: enable

(R/W)

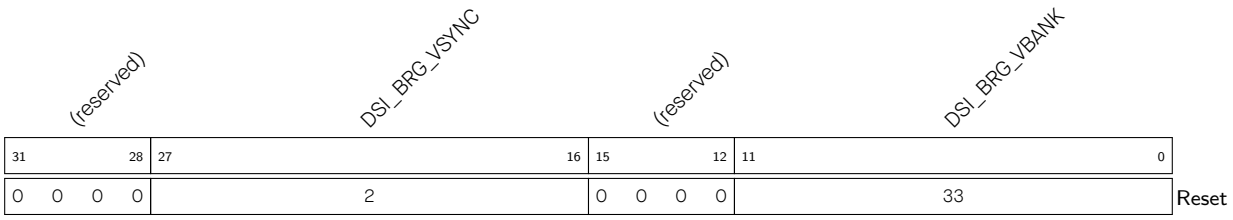
Register 43.11. DSI_BRG_DPI_V_CFG0_REG (0x0030)



DSI_BRG_VTOTAL Configures the total length of one frame (by line) for DPI output, must meet: reg_vtotal > reg_vdisp+reg_vsync+reg_vbank. (R/W)

DSI_BRG_VDISP Configures the length of valid line (by line) for DPI output. (R/W)

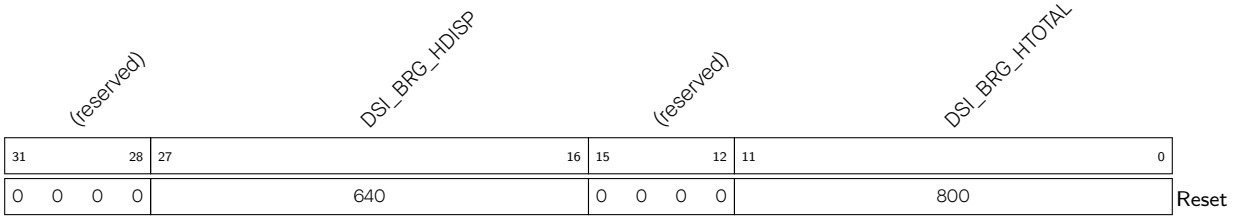
Register 43.12. DSI_BRG_DPI_V_CFG1_REG (0x0034)



DSI_BRG_VBANK Configures the length between vsync and valid line (by line) for DPI output. (R/W)

DSI_BRG_VSYNC Configures the length of vsync (by line) for DPI output. (R/W)

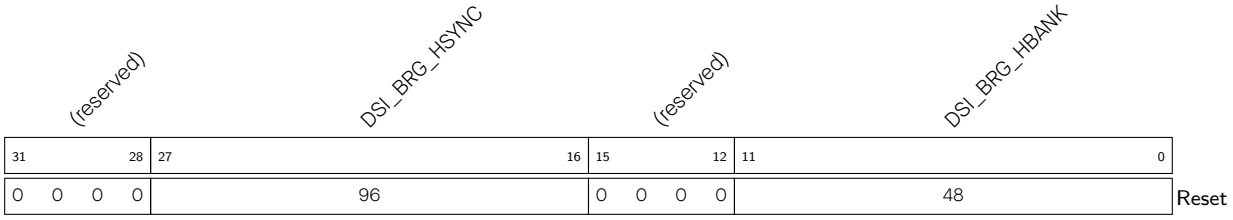
Register 43.13. DSI_BRG_DPI_H_CFG0_REG (0x0038)



DSI_BRG_HTOTAL Configures the total length of one line (by pixel count) for DPI output, must meet: reg_htotal > reg_hdisp+reg_hsync+reg_hbank. (R/W)

DSI_BRG_HDISP Configures the length of valid pixel data (by pixel count) for DPI output. (R/W)

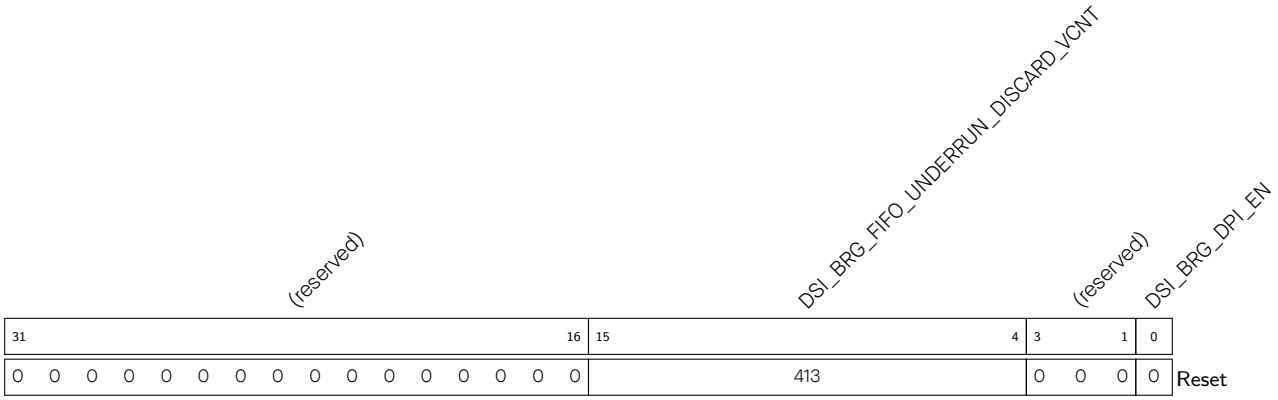
Register 43.14. DSI_BRG_DPI_H_CFG1_REG (0x003C)



DSI_BRG_HBANK Configures the length between hsync and pixel data valid (by pixel count) for DPI output. (R/W)

DSI_BRG_HSYNC Configures the length of hsync (by pixel count) for DPI output. (R/W)

Register 43.15. DSI_BRG_DPI_MISC_CONFIG_REG (0x0040)



DSI_BRG_DPI_EN Configures DPI output enable.

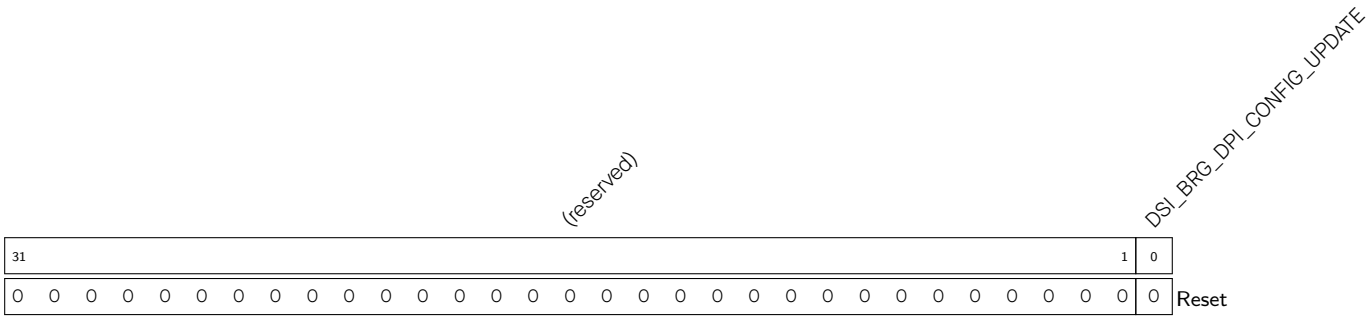
0: disable

1: enable

(R/W)

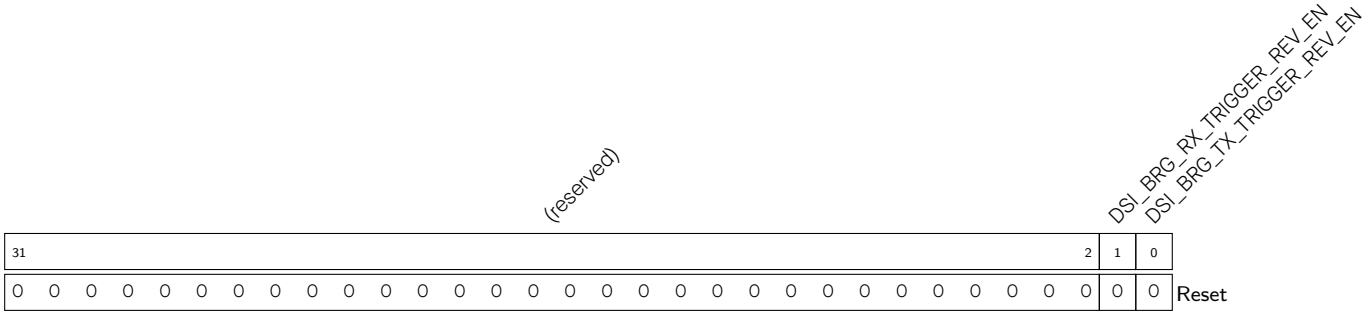
DSI_BRG_FIFO_UNDERRUN_DISCARD_VCNT Configures the underrun interrupt mask; when underrun occurs and line count is less than this field, the interrupt is suppressed. (R/W)

Register 43.16. DSI_BRG_DPI_CONFIG_UPDATE_REG (0x0044)



DSI_BRG_DPI_CONFIG_UPDATE Write 1 to update DPI config register MIPI_DSI_BRG_DPI_*. (WT)

Register 43.17. DSI_BRG_HOST_TRIGGER_REV_REG (0x0064)



DSI_BRG_TX_TRIGGER_REV_EN Configures DSI tx_trigger reverse.

0: disable

1: enable

(R/W)

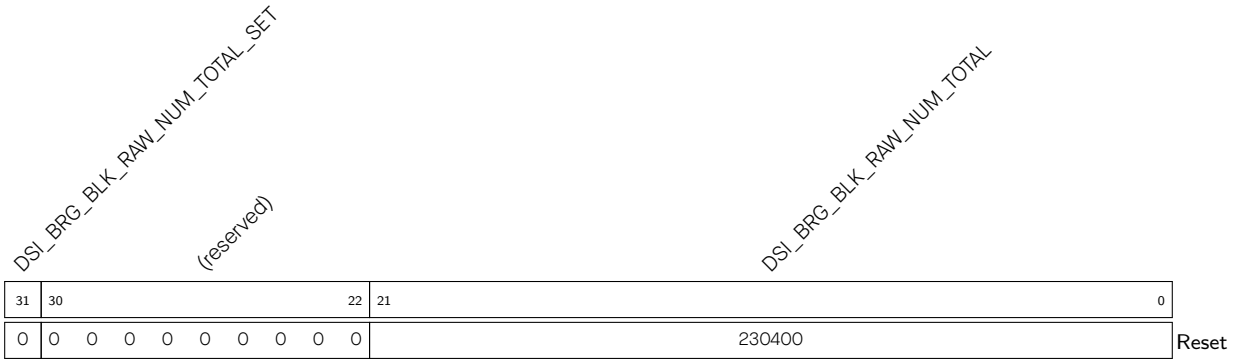
DSI_BRG_RX_TRIGGER_REV_EN Configures DSI rx_trigger reverse.

0: disable

1: enable

(R/W)

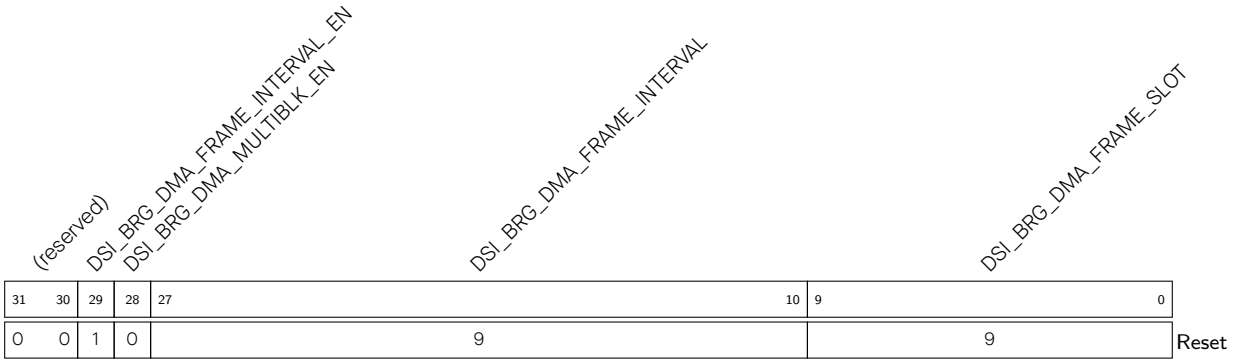
Register 43.18. DSI_BRG_BLK_RAW_NUM_CFG_REG (0x0068)



DSI_BRG_BLK_RAW_NUM_TOTAL Configures number of total block pix bits/64. (R/W)

DSI_BRG_BLK_RAW_NUM_TOTAL_SET Write 1 to reload reg_blk_raw_num_total to the internal counter. (WT)

Register 43.19. DSI_BRG_DMA_FRAME_INTERVAL_REG (0x006C)



DSI_BRG_DMA_FRAME_SLOT Configures the max frame_slot_cnt. (R/W)

DSI_BRG_DMA_FRAME_INTERVAL Configures the max frame_interval_cnt, frame_interval_cnt increased by 1 when frame_slot_cnt if full. (R/W)

DSI_BRG_DMA_MULTIBLK_EN Configures multi-block transfer enable.

0: disable

1: enable

(R/W)

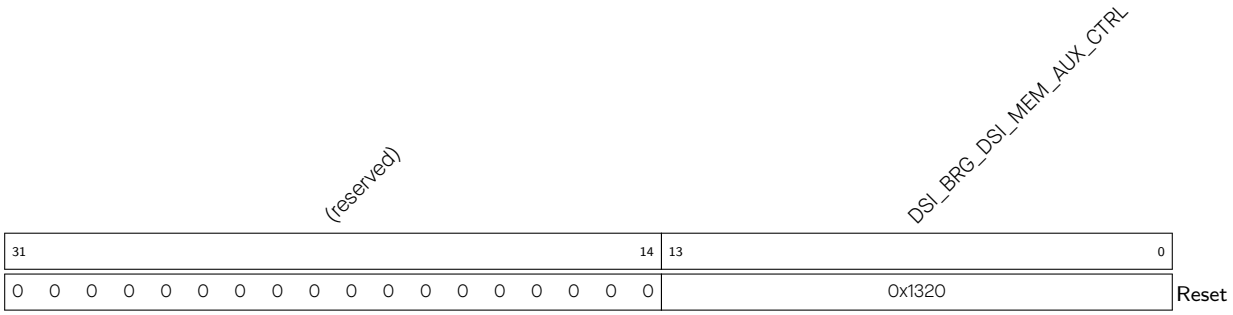
DSI_BRG_DMA_FRAME_INTERVAL_EN Configures enable interval between frame transfer.

0: disable

1: enable

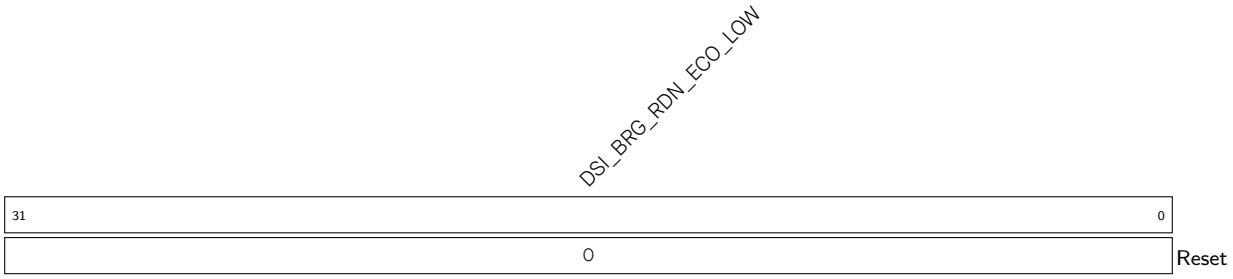
(R/W)

Register 43.20. DSI_BRG_MEM_AUX_CTRL_REG (0x0070)



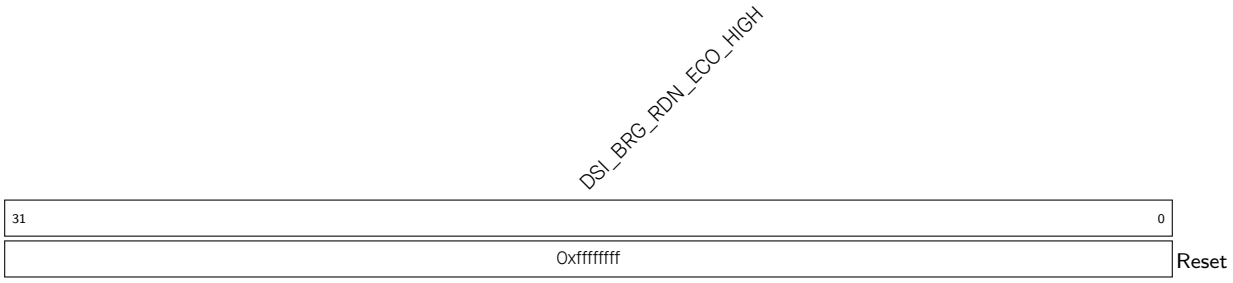
DSI_BRG_DSI_MEM_AUX_CTRL Configures dsi_bridge FIFO FIFO memory auxiliary control. (R/W)

Register 43.21. DSI_BRG_RDN_ECO_LOW_REG (0x0078)



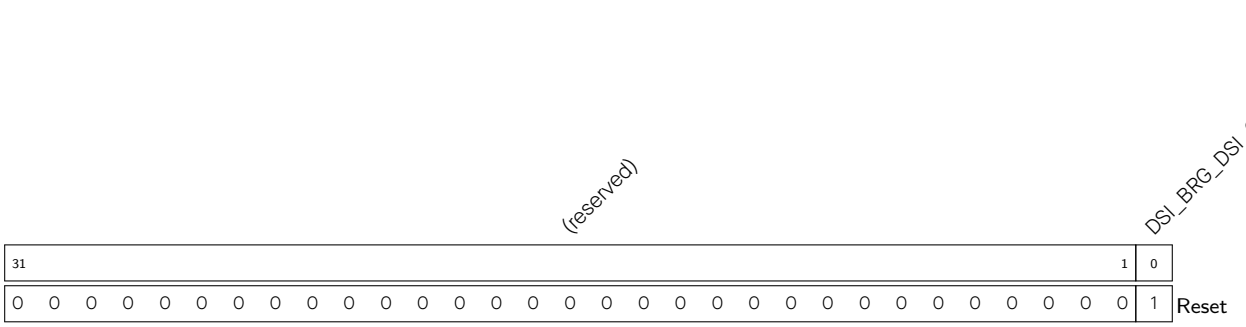
DSI_BRG_RDN_ECO_LOW RDN ECO low register value. (R/W)

Register 43.22. DSI_BRG_RDN_ECO_HIGH_REG (0x007C)



DSI_BRG_RDN_ECO_HIGH RDN ECO high register value. (R/W)

Register 43.23. DSI_BRG_HOST_CTRL_REG (0x0080)



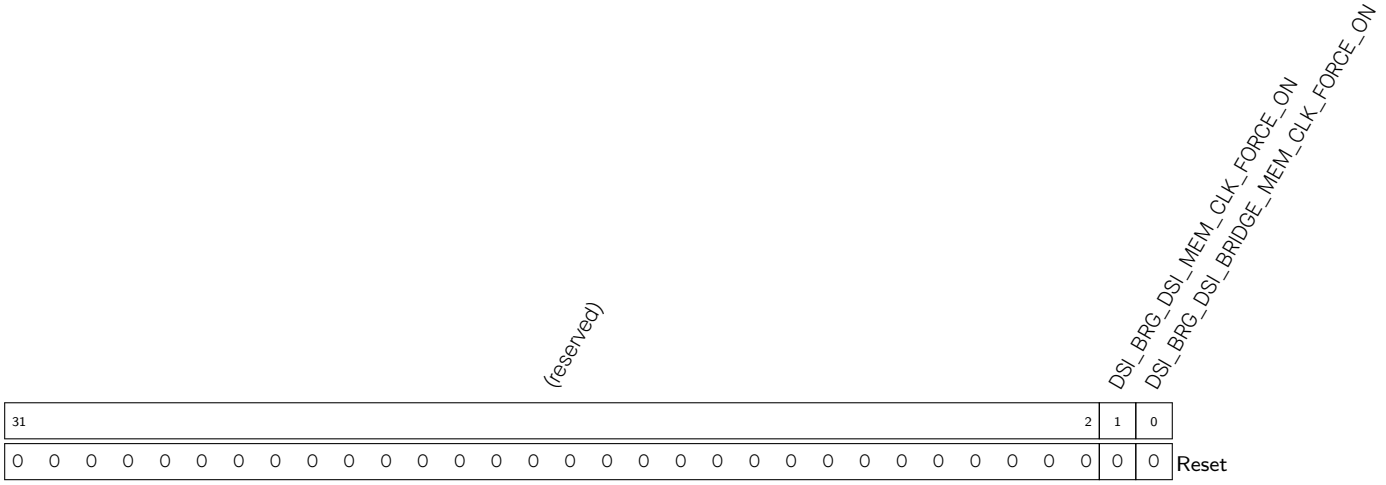
DSI_BRG_DSI_CFG_REF_CLK_EN Configures the clk enable refclk and cfg_clk of dsi_host.

0: disable

1: enable

(R/W)

Register 43.24. DSI_BRG_MEM_CLK_CTRL_REG (0x0084)



DSI_BRG_DSI_BRIDGE_MEM_CLK_FORCE_ON Configures the clock force on of dsi_bridge FIFO memory.

0: disable

1: force on

(R/W)

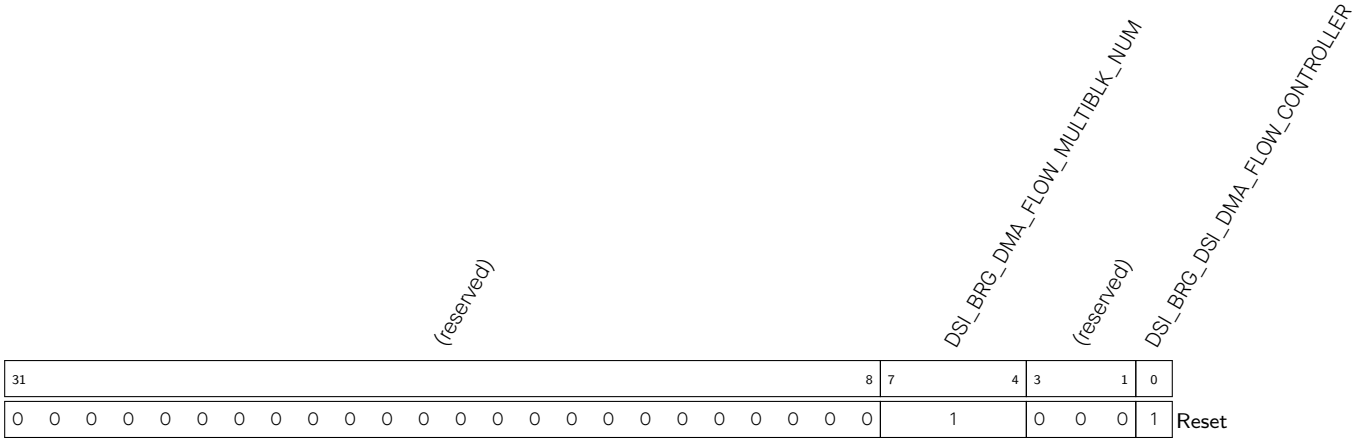
DSI_BRG_DSI_MEM_CLK_FORCE_ON Configures the clock force on of DPI FIFO memory.

0: disable

1: force on

(R/W)

Register 43.25. DSI_BRG_DMA_FLOW_CTRL_REG (0x0088)



DSI_BRG_DSI_DMA_FLOW_CONTROLLER Configures the flow controller.

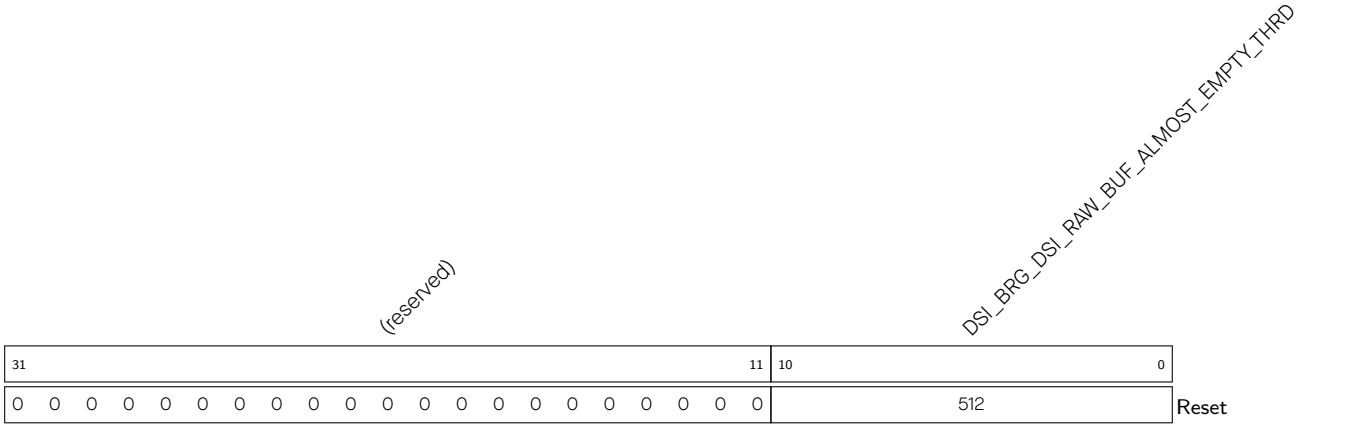
0: DMAC as flow controller

1: dsi_bridge as flow controller

(R/W)

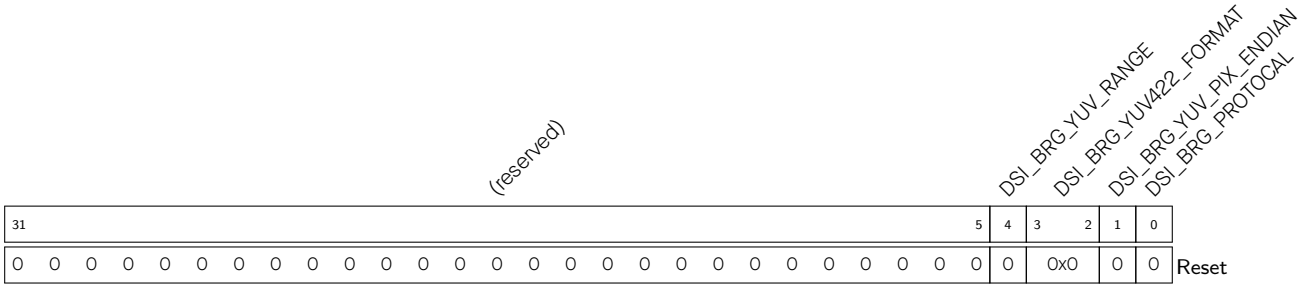
DSI_BRG_DMA_FLOW_MULTIBLK_NUM Configures the number of blocks when multi-block mode is enabled and DMAC is the flow controller. (R/W)

Register 43.26. DSI_BRG_RAW_BUF_ALMOST_EMPTY_THRD_REG (0x008C)



DSI_BRG_DSI_RAW_BUF_ALMOST_EMPTY_THRD Configures the FIFO almost empty threshold, valid only when DMAC as flow controller. (R/W)

Register 43.27. DSI_BRG_YUV_CFG_REG (0x0090)



DSI_BRG_PROTOCAL Configures YUV to RGB protocol.

0: BT.601

1: BT.709

(R/W)

DSI_BRG_YUV_PIX_ENDIAN Configures YUV pixel endian.

0: Y0U0Y1V1Y2U2Y3V3

1: Y3U3Y2V2Y1U1Y0V0

(R/W)

DSI_BRG_YUV422_FORMAT Configures YUV422 store format.

0: YUYV

1: YVYU

2: UYVY

3: VYUY

(R/W)

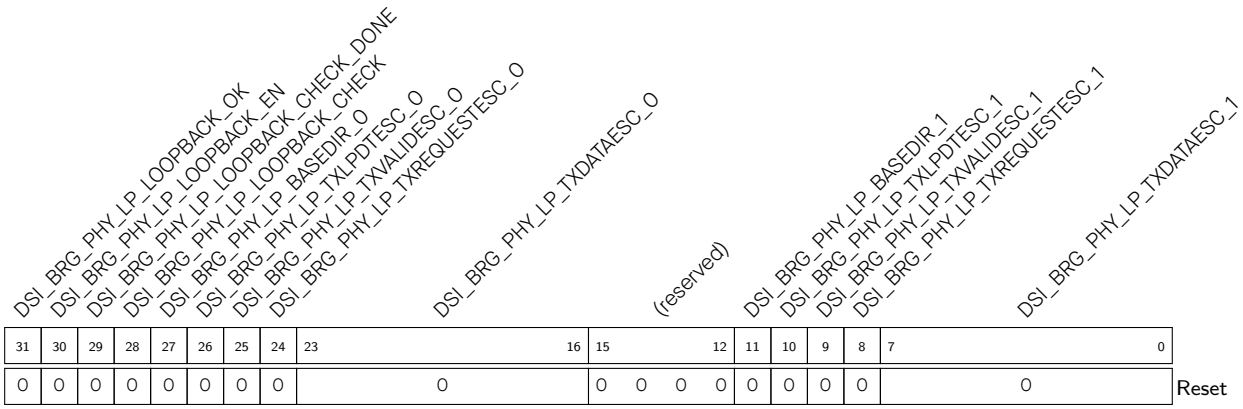
DSI_BRG_YUV_RANGE Configures YUV pixel range.

0: limit range

1: full range

(R/W)

Register 43.28. DSI_BRG_PHY_LP_LOOPBACK_CTRL_REG (0x0094)



DSI_BRG_PHY_LP_TXDATAESC_1 txdataesc_1 ctrl when enable DSI PHY lp_loopback_test. (R/W)

DSI_BRG_PHY_LP_TXREQUESTESC_1 txrequestesc_1 ctrl when enable DSI PHY lp_loopback_test. (R/W)

DSI_BRG_PHY_LP_TXVALIDESC_1 txvalidesc_1 ctrl when enable DSI PHY lp_loopback_test. (R/W)

DSI_BRG_PHY_LP_TXLPDTESC_1 txlpdtesc_1 ctrl when enable DSI PHY lp_loopback_test. (R/W)

DSI_BRG_PHY_LP_BASEDIR_1 basedir_1 ctrl when enable DSI PHY lp_loopback_test. (R/W)

DSI_BRG_PHY_LP_TXDATAESC_0 txdataesc_0 ctrl when enable DSI PHY lp_loopback_test. (R/W)

DSI_BRG_PHY_LP_TXREQUESTESC_0 txrequestesc_0 ctrl when enable DSI PHY lp_loopback_test. (R/W)

DSI_BRG_PHY_LP_TXVALIDESC_0 txvalidesc_0 ctrl when enable DSI PHY lp_loopback_test. (R/W)

DSI_BRG_PHY_LP_TXLPDTESC_0 txlpdtesc_0 ctrl when enable DSI PHY lp_loopback_test. (R/W)

DSI_BRG_PHY_LP_BASEDIR_0 basedir_0 ctrl when enable DSI PHY lp_loopback_test. (R/W)

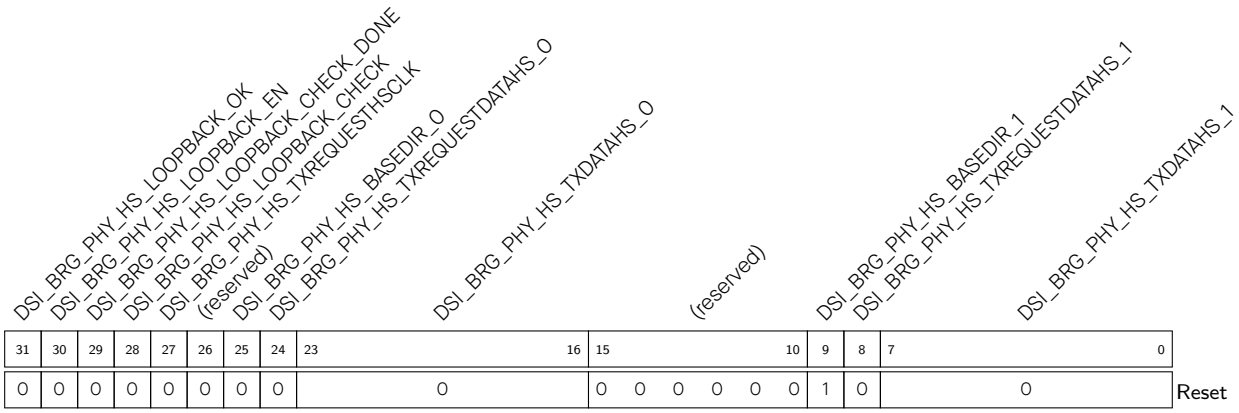
DSI_BRG_PHY_LP_LOOPBACK_CHECK DSI PHY lp_loopback test start check. (WT)

DSI_BRG_PHY_LP_LOOPBACK_CHECK_DONE DSI PHY lp_loopback test check done. (RO)

DSI_BRG_PHY_LP_LOOPBACK_EN DSI PHY LP loopback control enable. (R/W)

DSI_BRG_PHY_LP_LOOPBACK_OK result of DSI PHY lp_loopback test. (RO)

Register 43.29. DSI_BRG_PHY_HS_LOOPBACK_CTRL_REG (0x0098)



DSI_BRG_PHY_HS_TXDATAHS_1 txdatahs_1 ctrl when enable DSI PHY hs_loopback_test. (R/W)

DSI_BRG_PHY_HS_TXREQUESTDATAHS_1 txrequestdatahs_1 ctrl when enable DSI PHY hs_loopback_test. (R/W)

DSI_BRG_PHY_HS_BASEDIR_1 basedir_1 ctrl when enable DSI PHY hs_loopback_test. (R/W)

DSI_BRG_PHY_HS_TXDATAHS_0 txdatahs_0 ctrl when enable DSI PHY hs_loopback_test. (R/W)

DSI_BRG_PHY_HS_TXREQUESTDATAHS_0 txrequestdatahs_0 ctrl when enable DSI PHY hs_loopback_test. (R/W)

DSI_BRG_PHY_HS_BASEDIR_0 basedir_0 ctrl when enable DSI PHY hs_loopback_test. (R/W)

DSI_BRG_PHY_HS_TXREQUESTHSCLK txrequesthsclk when enable DSI PHY hs_loopback_test. (R/W)

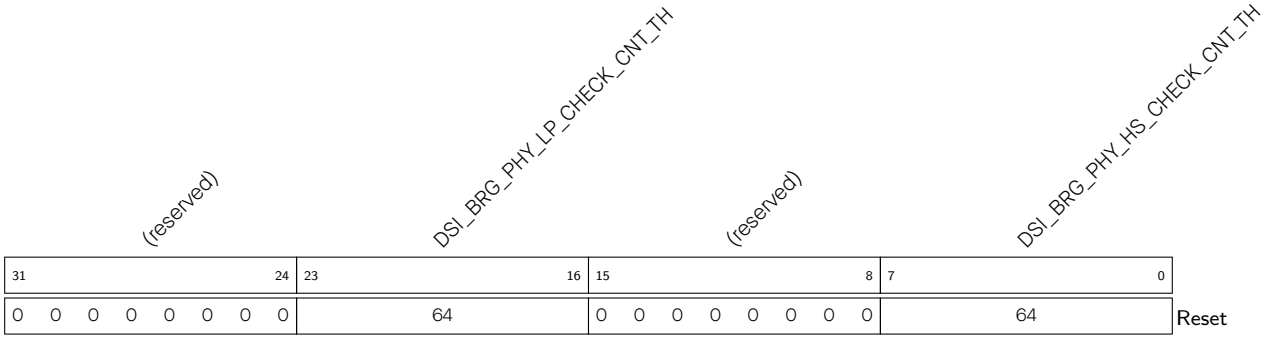
DSI_BRG_PHY_HS_LOOPBACK_CHECK DSI PHY hs_loopback test start check. (WT)

DSI_BRG_PHY_HS_LOOPBACK_CHECK_DONE DSI PHY hs_loopback test check done. (RO)

DSI_BRG_PHY_HS_LOOPBACK_EN DSI PHY HS loopback control enable. (R/W)

DSI_BRG_PHY_HS_LOOPBACK_OK result of DSI PHY hs_loopback test. (RO)

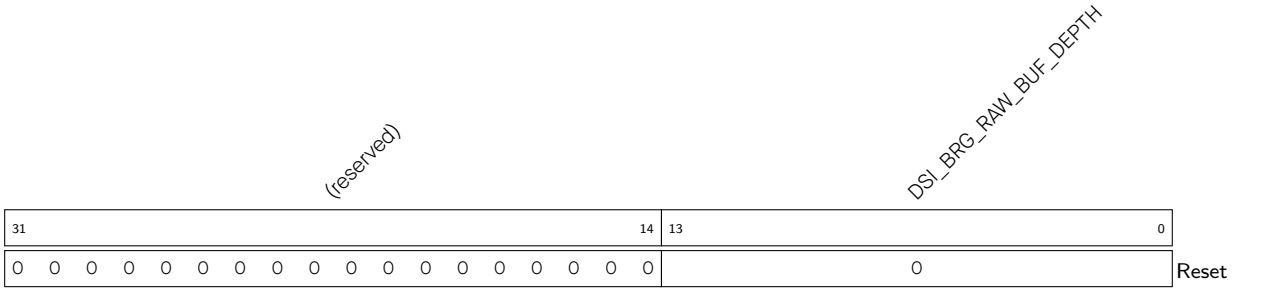
Register 43.30. DSI_BRG_PHY_LOOPBACK_CNT_REG (0x009C)



DSI_BRG_PHY_HS_CHECK_CNT_TH hs_loopback test check count. (R/W)

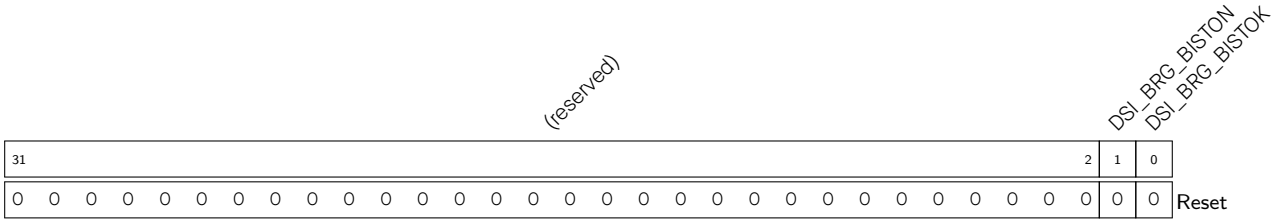
DSI_BRG_PHY_LP_CHECK_CNT_TH lp_loopback test check count. (R/W)

Register 43.31. DSI_BRG_FIFO_FLOW_STATUS_REG (0x0014)



DSI_BRG_RAW_BUF_DEPTH Represents the current depth of dsi_bridge FIFO. (RO)

Register 43.32. DSI_BRG_HOST_BIST_CTL_REG (0x0060)



DSI_BRG_BISTOK BIST result indicator. (RO)

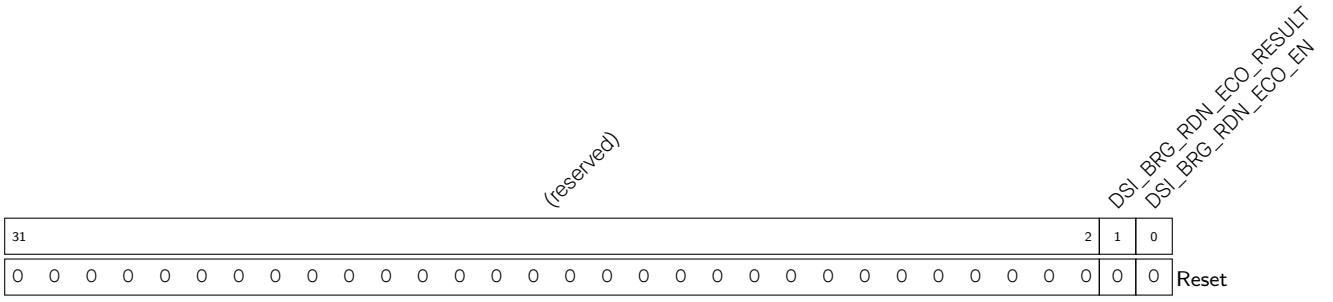
DSI_BRG_BISTON Configures BIST enable.

0: disable

1: enable

(R/W)

Register 43.33. DSI_BRG_RDN_ECO_CS_REG (0x0074)



DSI_BRG_RDN_ECO_EN Configures RDN ECO enable.

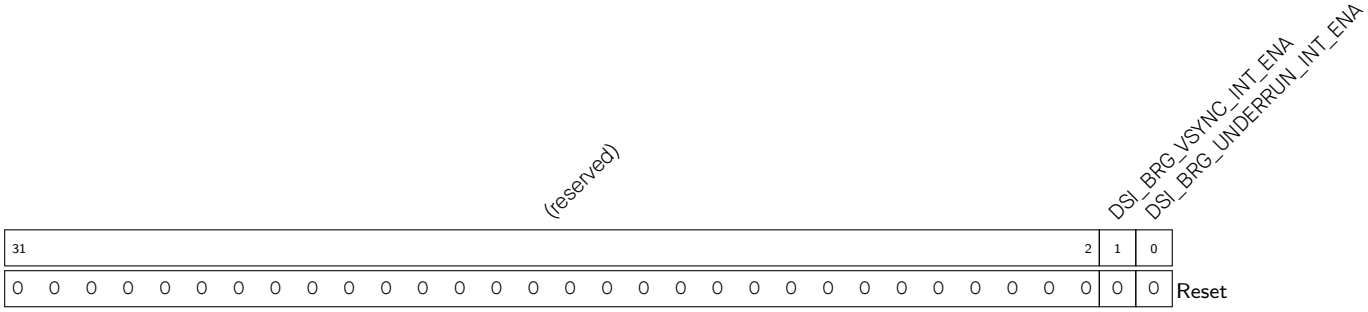
0: disable

1: enable

(R/W)

DSI_BRG_RDN_ECO_RESULT RDN ECO result. (RO)

Register 43.34. DSI_BRG_INT_ENA_REG (0x0050)



DSI_BRG_UNDERRUN_INT_ENA Configures DSI_BRG_UNDERRUN_INT enable.

0: disable

1: enable

(R/W)

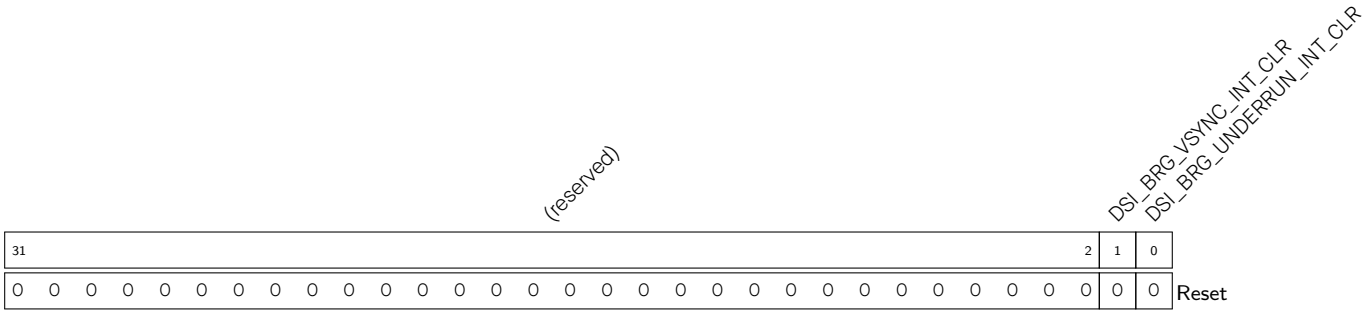
DSI_BRG_VSYNC_INT_ENA Configures DSI_BRG_VSYNC_INT enable.

0: disable

1: enable

(R/W)

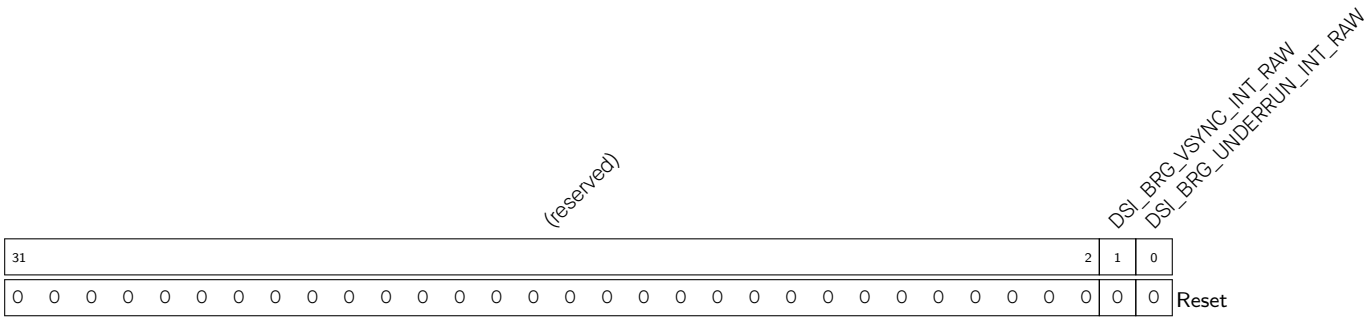
Register 43.35. DSI_BRG_INT_CLR_REG (0x0054)



DSI_BRG_UNDERRUN_INT_CLR write 1 to clear dpi_underrun_int_raw field of MIPI_DSI_BRG_INT_RAW_REG. (WT)

DSI_BRG_VSYNC_INT_CLR write 1 to clear dpi_vsync_int_raw field of MIPI_DSI_BRG_INT_RAW_REG. (WT)

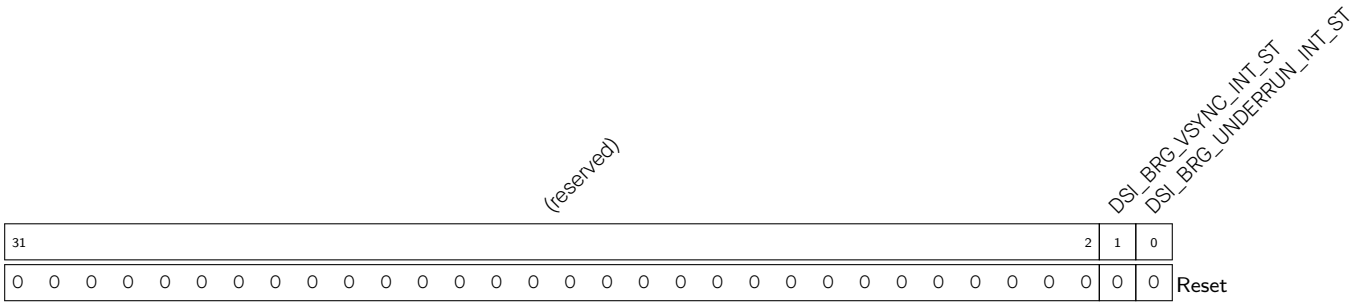
Register 43.36. DSI_BRG_INT_RAW_REG (0x0058)



DSI_BRG_UNDERRUN_INT_RAW Represents raw interrupt status of dpi_underrun. (R/WTC/SS)

DSI_BRG_VSYNC_INT_RAW Represents raw interrupt status of dpi_vsync. (R/WTC/SS)

Register 43.37. DSI_BRG_INT_ST_REG (0x005C)



DSI_BRG_UNDERRUN_INT_ST Represents the masked interrupt status of dpi_underrun. (RO)

DSI_BRG_VSYNC_INT_ST Represents the masked interrupt status of dpi_vsync. (RO)

Register 43.38. DSI_BRG_VER_DATE_REG (0x0100)



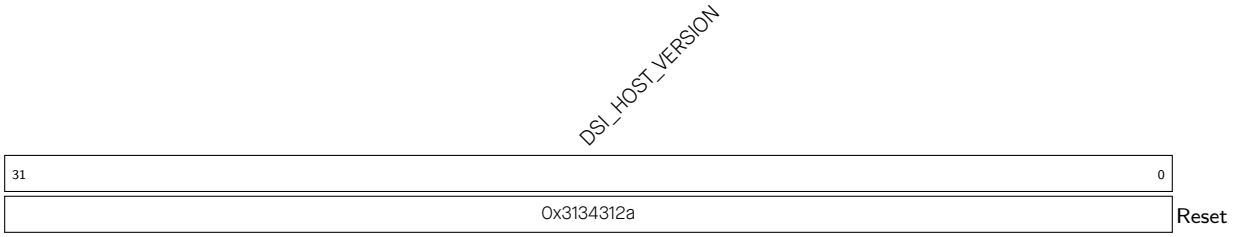
DSI_BRG_VER_DATA Represents the CSV version date of this register file. (R/W)

43.8.2 MIPI DSI Host Register

The addresses in this section are relative to MIPI DSI Host base address provided in Table 9.3-2 in Chapter 9 *System and Memory*.

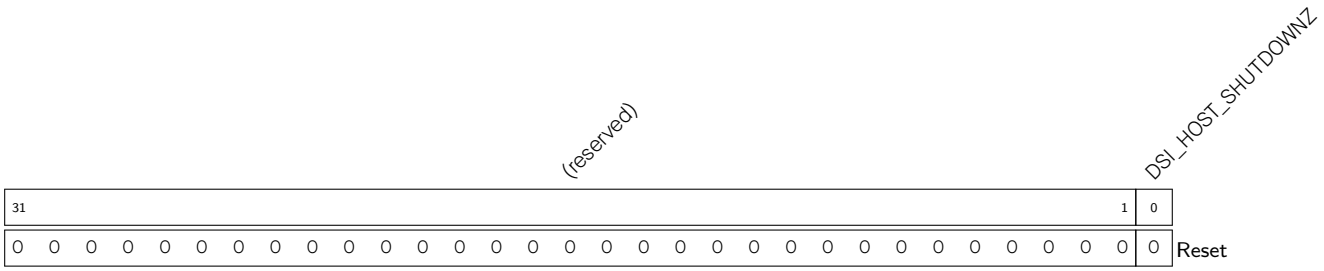
For how to program reserved fields, please refer to Section Programming Reserved Register Field.

Register 43.39. DSI_HOST_VERSION_REG (0x0000)



DSI_HOST_VERSION Represents DSI Host version. (RO)

Register 43.40. DSI_HOST_PWR_UP_REG (0x0004)



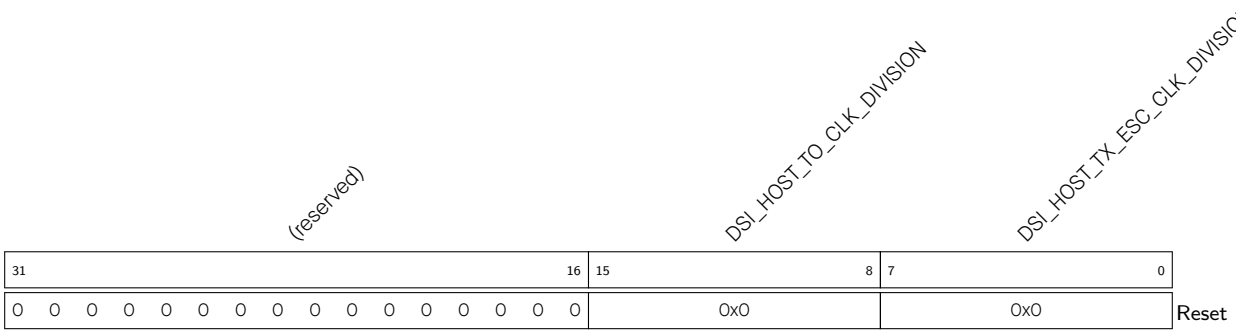
DSI_HOST_SHUTDOWNZ Configures DSI Host power.

0: reset

1: power up

(R/W)

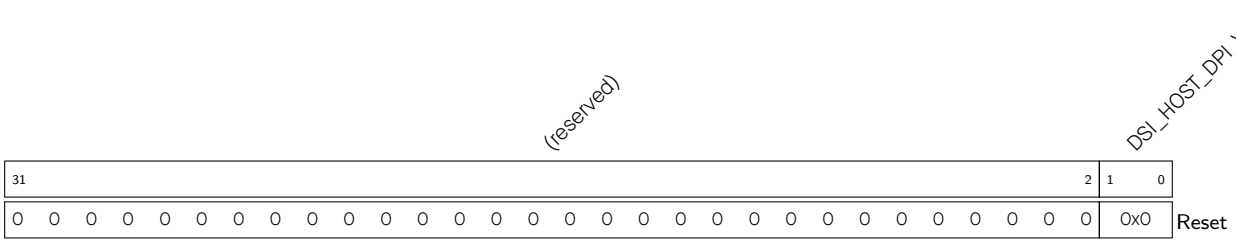
Register 43.41. DSI_HOST_CLKMGR_CFG_REG (0x0008)



DSI_HOST_TX_ESC_CLK_DIVISION Configures the division factor of TX Escape clock. The values 0 or 1 will stop the TX Eacape clock. (R/W)

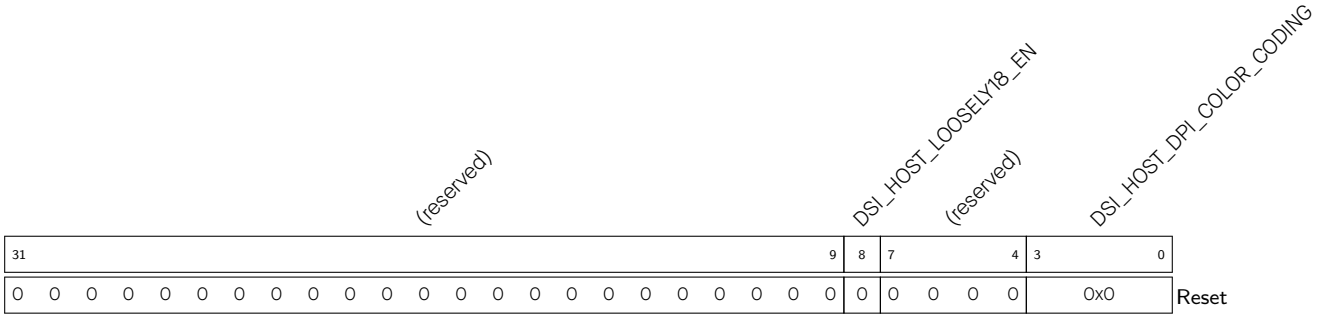
DSI_HOST_TO_CLK_DIVISION Configures the division factor of Time Out counter clock for HS to LP and LP to HS. (R/W)

Register 43.42. DSI_HOST_DPI_VCID_REG (0x000C)



DSI_HOST_DPI_VCID Configures the DPI virtual channel ID. (R/W)

Register 43.43. DSI_HOST_DPI_COLOR_CODING_REG (0x0010)



DSI_HOST_DPI_COLOR_CODING Configures the DPI color coding, refer to table 43.4-2 for details

0: RGB565 configuration 0

1: RGB565 configuration 1

2: RGB565 configuration 2

3: RGB666 configuration 0

4: RGB666 configuration 1

5: RGB888

others: reserved

(R/W)

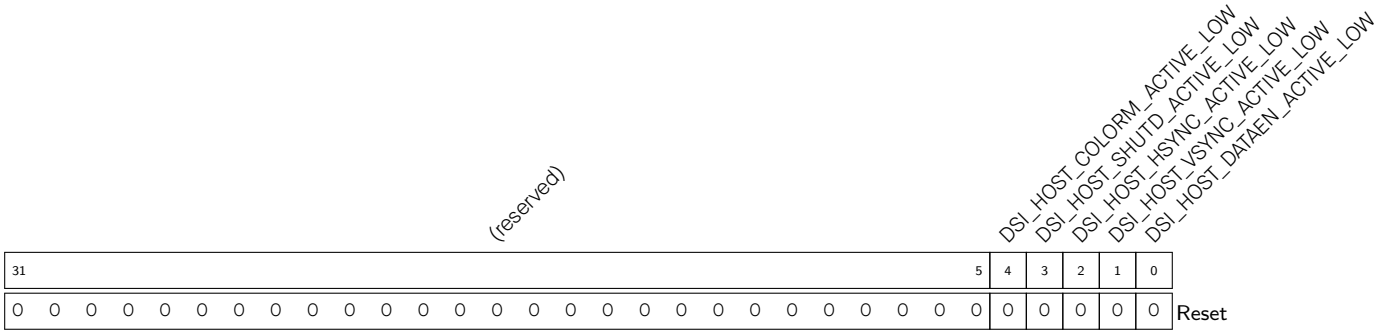
DSI_HOST_LOOSELY18_EN Configures loosely packet for 18-bit configurations.

0:disable

1:enable

(R/W)

Register 43.44. DSI_HOST_DPI_CFG_POL_REG (0x0014)



DSI_HOST_DATAEN_ACTIVE_LOW Configures DPI dpidataen signal polarity.

0: active high

1: active low

(R/W)

DSI_HOST_VSYNC_ACTIVE_LOW Configures DPI dpivsync signal polarity.

0: active high

1: active low

(R/W)

DSI_HOST_HSYNC_ACTIVE_LOW Configures DPI dpihsync signal polarity.

0: active high

1: active low

(R/W)

DSI_HOST_SHUTD_ACTIVE_LOW Configures DPI dpishutdn signal polarity.

0: active high

1: active low

(R/W)

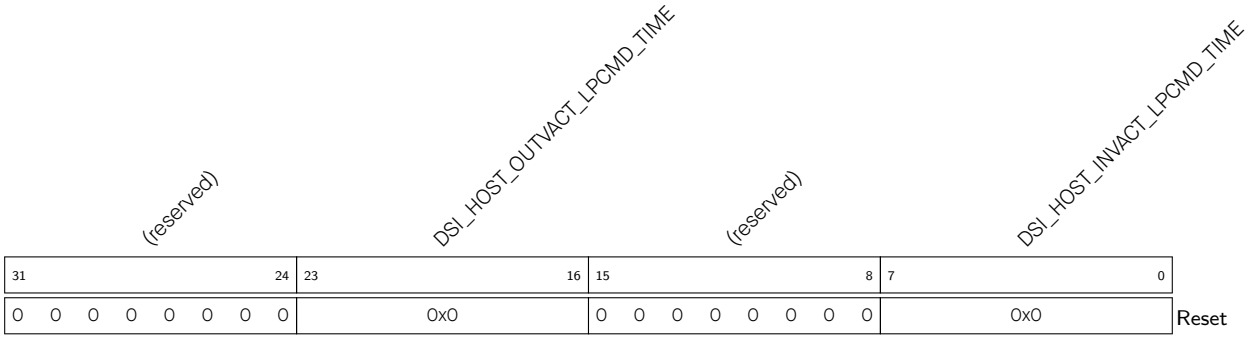
DSI_HOST_COLORM_ACTIVE_LOW Configures DPI dpicolorm signal polarity.

0: active high

1: active low

(R/W)

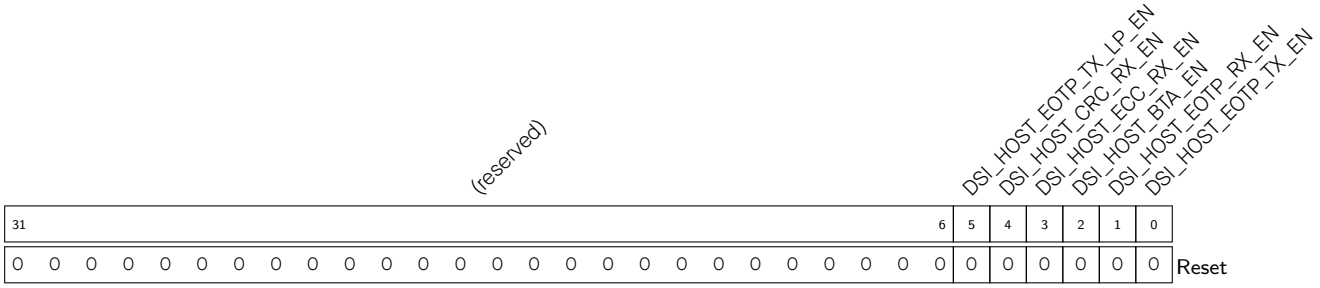
Register 43.45. DSI_HOST_DPI_LP_CMD_TIM_REG (0x0018)



DSI_HOST_INVACT_LPCMD_TIME Configures the size in bytes of the largest packet that can fit in a line during VACT region for LP command transmission. (R/W)

DSI_HOST_OUTVACT_LPCMD_TIME Configures the size in bytes of the largest packet that can fit in a line during VSA VBP VFP region for LP command transmission. (R/W)

Register 43.46. DSI_HOST_PCKHDL_CFG_REG (0x002C)



DSI_HOST_EOTP_TX_EN Configures the enable of the EoTp transmission in HS.

0: disable

1: enable

(R/W)

DSI_HOST_EOTP_RX_EN Configures the enable of the EoTp reception.

0: disable

1: enable

(R/W)

DSI_HOST_BTA_EN Configures the enable of the Bus Turn-Around (BTA) request.

0: disable

1: enable

(R/W)

DSI_HOST_ECC_RX_EN Configures the enable of ECC reception, error correction, and reporting.

0: disable

1: enable

(R/W)

DSI_HOST_CRC_RX_EN Configures the enable of CRC reception and error reporting.

0: disable

1: enable

(R/W)

DSI_HOST_EOTP_TX_LP_EN Configures the enable of the EoTp transmission in LP.

0: disable

1: enable

(R/W)

Register 43.47. DSI_HOST_GEN_VCID_REG (0x0030)

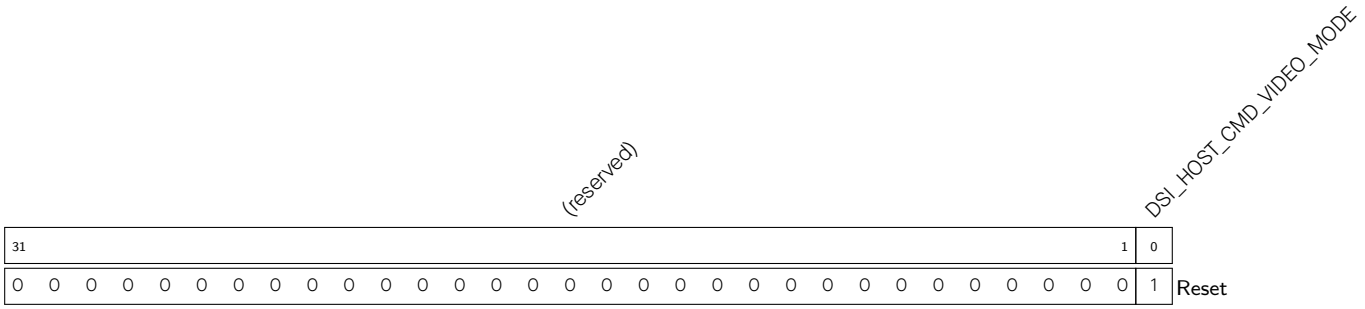


DSI_HOST_GEN_VCID_RX Configures Generic Interface read-back virtual channel ID. (R/W)

DSI_HOST_GEN_VCID_TEAR_AUTO Configures hardware TEA effect virtual channel ID. (R/W)

DSI_HOST_GEN_VCID_TX_AUTO Configures Generic Interface auto generated or transmitted packet virtual channel ID. (R/W)

Register 43.48. DSI_HOST_MODE_CFG_REG (0x0034)



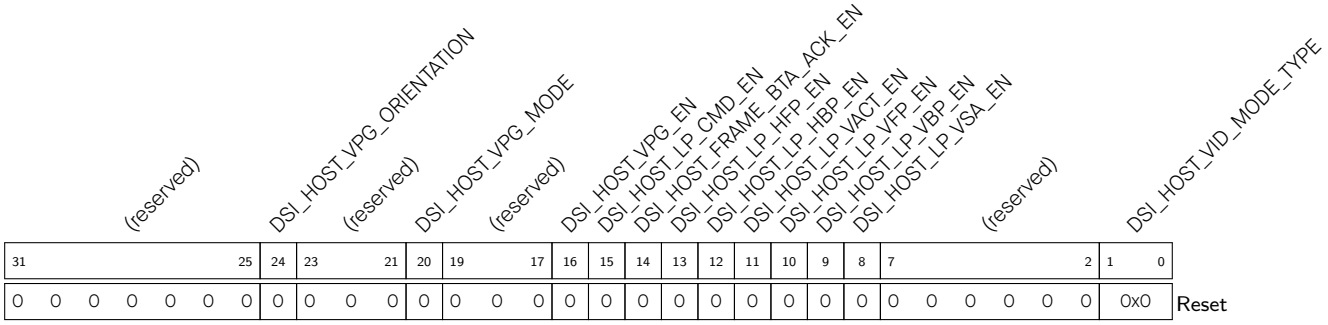
DSI_HOST_CMD_VIDEO_MODE Configures operation mode.

0: video mode

1: command mode

(R/W)

Register 43.49. DSI_HOST_VID_MODE_CFG_REG (0x0038)


(reserved)
DSI_HOST_VPG_ORIENTATION
(reserved)
DSI_HOST_VPG_MODE
(reserved)
DSI_HOST_VPG_EN
DSI_HOST_LP_CMD_EN
DSI_HOST_FRAME_BTA_ACK_EN
DSI_HOST_LP_HFP_EN
DSI_HOST_LP_HBP_EN
DSI_HOST_LP_VACT_EN
DSI_HOST_LP_VFP_EN
DSI_HOST_LP_VBP_EN
DSI_HOST_LP_VSA_EN
(reserved)
DSI_HOST_VID_MODE_TYPE
31 25 24 23 21 20 19 17 16 15 14 13 12 11 10 9 8 7 2 1 0
0 0 0

Register 43.49. DSI_HOST_VID_MODE_CFG_REG (0x0038)

Continued from the previous page...

DSI_HOST_FRAME_BTA_ACK_EN Configures whether to enable the request for an acknowledge response at the end of a frame.

0: disable

1: enable

(R/W)

DSI_HOST_LP_CMD_EN Configures whether to enable the command transmission only in LP mode.

0: disable

1: enable

(R/W)

DSI_HOST_VPG_EN Configures the video pattern generator enable.

0: disable

1: enable

(R/W)

DSI_HOST_VPG_MODE Configures the pattern type.

0: color bar

1: ber pattern

(R/W)

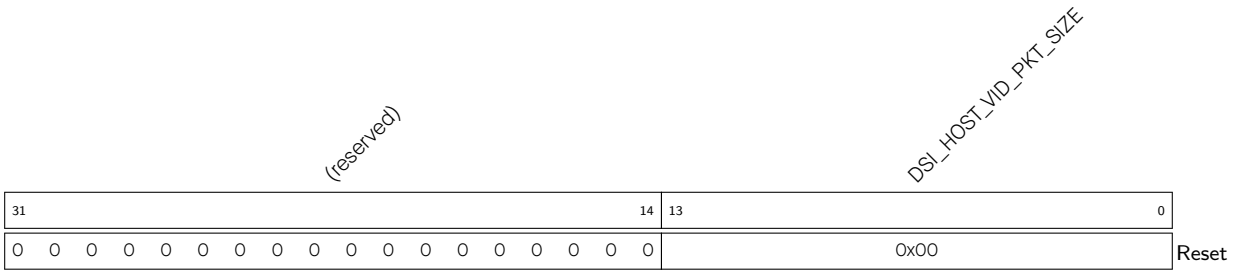
DSI_HOST_VPG_ORIENTATION Configures the color bar orientation.

0: vertical

1: horizontal

(R/W)

Register 43.50. DSI_HOST_VID_PKT_SIZE_REG (0x003C)

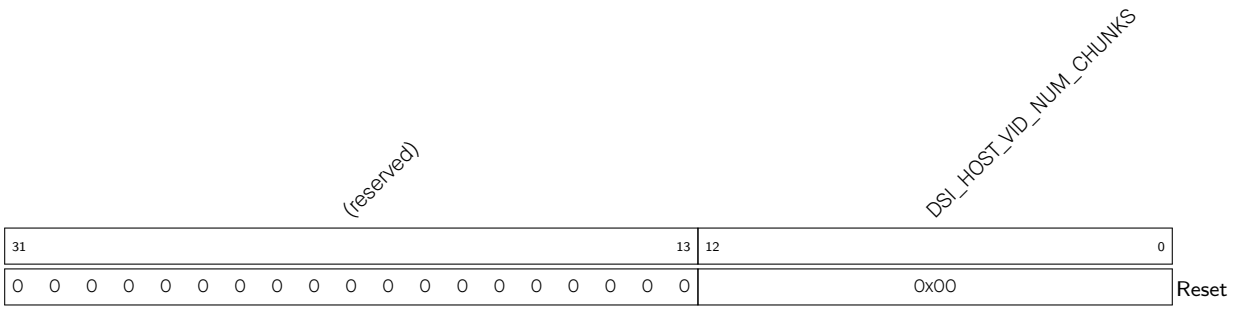


DSI_HOST_VID_PKT_SIZE Configures the number of pixels in a single video packet

For 18-bit not loosely packed data, the value must be a multiple of 4

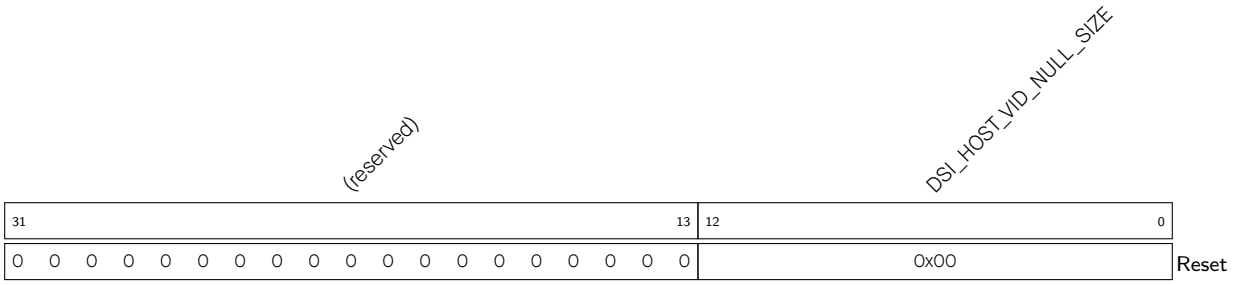
(R/W)

Register 43.51. DSI_HOST_VID_NUM_CHUNKS_REG (0x0040)



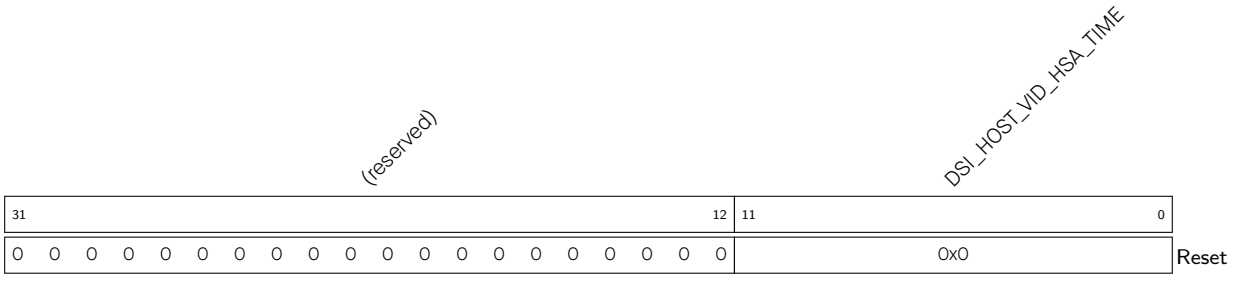
DSI_HOST_VID_NUM_CHUNKS Configures the number of chunks to be transmitted during a line. A chunk is a pair made of a video packet and a null packet. (R/W)

Register 43.52. DSI_HOST_VID_NULL_SIZE_REG (0x0044)



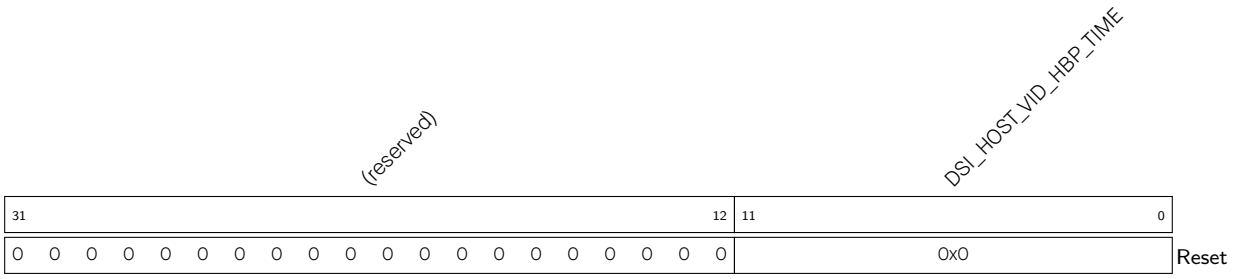
DSI_HOST_VID_NULL_SIZE Configures the number of bytes inside a null packet. Set to 0 to disable null packets. (R/W)

Register 43.53. DSI_HOST_VID_HSA_TIME_REG (0x0048)



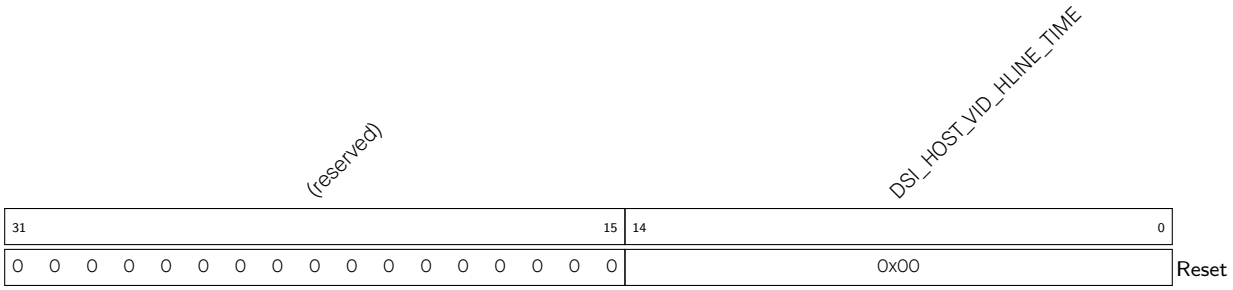
DSI_HOST_VID_HSA_TIME Configures the HSA period in lane byte clock cycles. (R/W)

Register 43.54. DSI_HOST_VID_HBP_TIME_REG (0x004C)



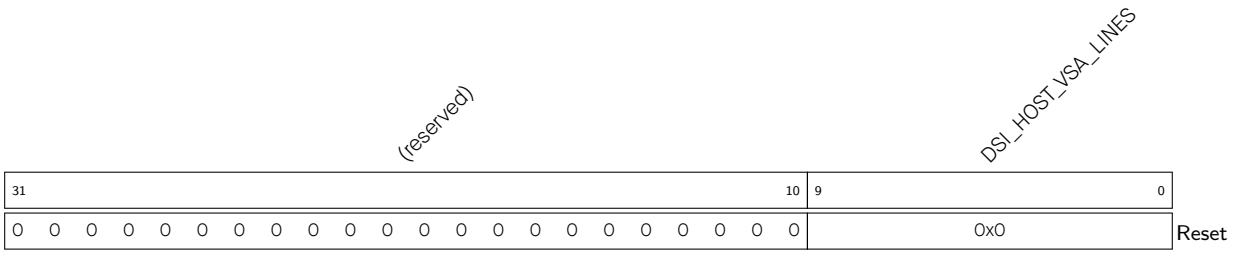
DSI_HOST_VID_HBP_TIME Configures the HBP period in lane byte clock cycles. (R/W)

Register 43.55. DSI_HOST_VID_HLINE_TIME_REG (0x0050)



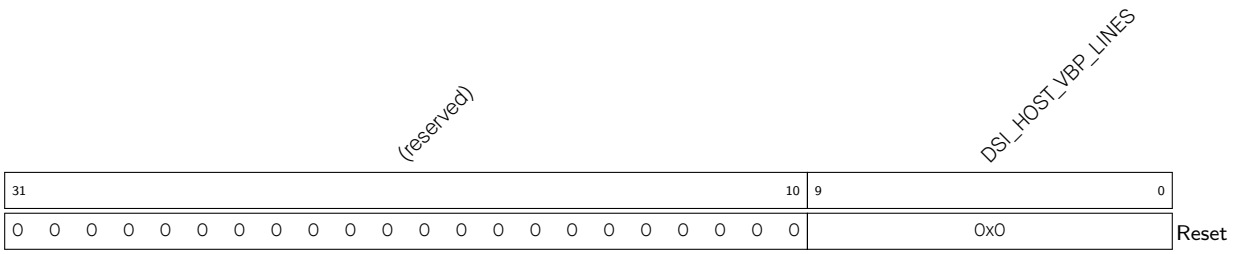
DSI_HOST_VID_HLINE_TIME Configures the HTOTAL time (HSA+HBP+HACT+HFP) in lane byte clock cycles. (R/W)

Register 43.56. DSI_HOST_VID_VSA_LINES_REG (0x0054)



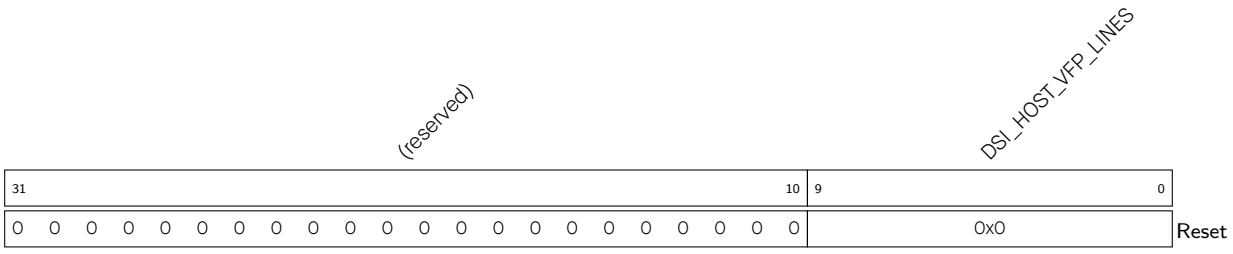
DSI_HOST_VSA_LINES Configures the VSA period in number of lines. (R/W)

Register 43.57. DSI_HOST_VID_VBP_LINES_REG (0x0058)



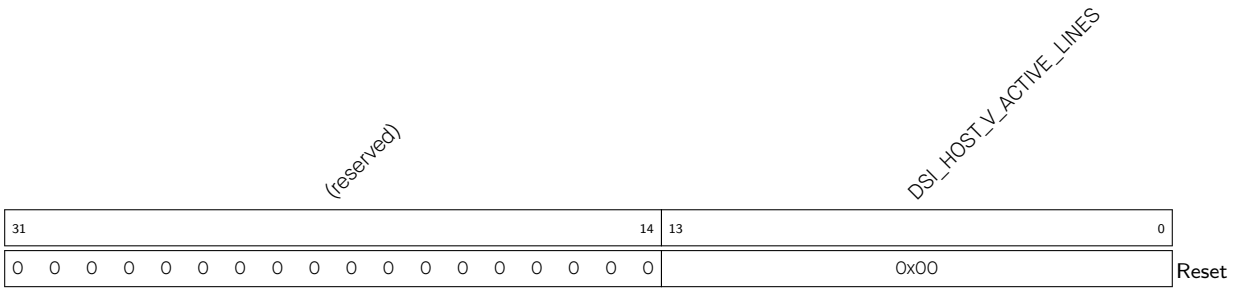
DSI_HOST_VBP_LINES Configures the VBP period in number of lines. (R/W)

Register 43.58. DSI_HOST_VID_VFP_LINES_REG (0x005C)



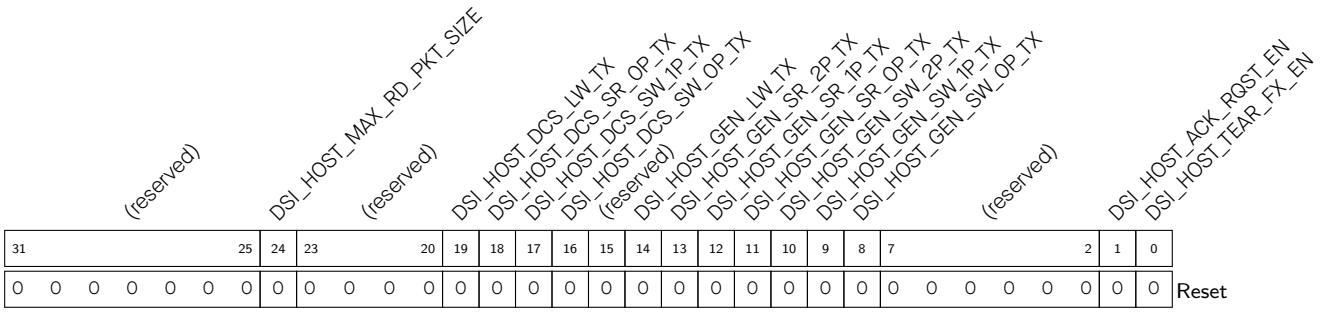
DSI_HOST_VFP_LINES Configures the VFP period in number of lines. (R/W)

Register 43.59. DSI_HOST_VID_VACTIVE_LINES_REG (0x0060)



DSI_HOST_V_ACTIVE_LINES Configures the VACT period in number of lines. (R/W)

Register 43.60. DSI_HOST_CMD_MODE_CFG_REG (0x0068)


(reserved)
DSI_HOST_MAX_RD_PKT_SIZE
(reserved)
DSI_HOST_DCS_LW_TX
DSI_HOST_DCS_SR_0P_TX
DSI_HOST_DCS_SW_1P_TX
DSI_HOST_DCS_SW_0P_TX
(reserved)
DSI_HOST_GEN_LW_TX
DSI_HOST_GEN_SR_2P_TX
DSI_HOST_GEN_SR_1P_TX
DSI_HOST_GEN_SR_0P_TX
DSI_HOST_GEN_SW_2P_TX
DSI_HOST_GEN_SW_1P_TX
DSI_HOST_GEN_SW_0P_TX
(reserved)
DSI_HOST_ACK_RQST_EN
DSI_HOST_TEAR_FX_EN
31 25 24 23 20 19 18 17 16 15 14 13 12 11 10 9 8 7 2 1 0
0 0 0

Register 43.60. DSI_HOST_CMD_MODE_CFG_REG (0x0068)

Continued from the previous page...

DSI_HOST_GEN_SR_1P_TX Configures the Generic short read packet with one parameter command transmission type.
0: HS
1: LP
(R/W)

DSI_HOST_GEN_SR_2P_TX Configures the Generic short read packet with two parameter command transmission type.
0: HS
1: LP
(R/W)

DSI_HOST_GEN_LW_TX Configures the Generic long write packet command transmission type.
0: HS
1: LP
(R/W)

DSI_HOST_DCS_SW_0P_TX Configures the DCS short write packet with zero parameter command transmission type.
0: HS
1: LP
(R/W)

DSI_HOST_DCS_SW_1P_TX Configures the DCS short write packet with one parameter command transmission type.
0: HS
1: LP
(R/W)

DSI_HOST_DCS_SR_0P_TX Configures the DCS short read packet with zero parameter command transmission type.
0: HS
1: LP
(R/W)

DSI_HOST_DCS_LW_TX Configures the DCS long write packet command transmission type.
0: HS
1: LP
(R/W)

DSI_HOST_MAX_RD_PKT_SIZE Configures the max read packet size command transmission type.
0: HS
1: LP
(R/W)

Register 43.61. DSI_HOST_GEN_HDR_REG (0x006C)



DSI_HOST_GEN_DT Configures the Data Type of packet header. (R/W)

DSI_HOST_GEN_VC Configures the Virtual Channel ID of packet header. (R/W)

DSI_HOST_GEN_WC_LSBYTE Configures the least significant byte of Word Count for long packets or data 0 for short packets of packet header. (R/W)

DSI_HOST_GEN_WC_MSBYTE Configures the most significant byte of Word Count for long packets or data 1 for short packets of packet header. (R/W)

Register 43.62. DSI_HOST_GEN_PLD_DATA_REG (0x0070)



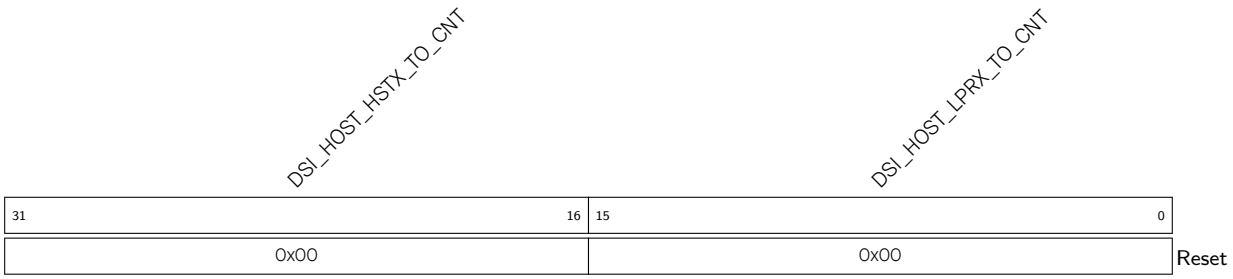
DSI_HOST_GEN_PLD_B1 Configures the byte 1 of the packet payload. (R/W)

DSI_HOST_GEN_PLD_B2 Configures the byte 2 of the packet payload. (R/W)

DSI_HOST_GEN_PLD_B3 Configures the byte 3 of the packet payload. (R/W)

DSI_HOST_GEN_PLD_B4 Configures the byte 4 of the packet payload. (R/W)

Register 43.63. DSI_HOST_TO_CNT_CFG_REG (0x0078)



DSI_HOST_LPRX_TO_CNT Configures the timeout counter that triggers a LP reception timeout contention detection (measured in TO_CLK_DIVISION cycles). (R/W)

DSI_HOST_HSTX_TO_CNT Configures the timeout counter that triggers a HS transmission timeout contention detection (measured in TO_CLK_DIVISION cycles).
For non-burst mode, if there is not sufficient time to switch from HS to LP and back during HFP, the DSI link returns to LP once per frame. The configuration should meet:

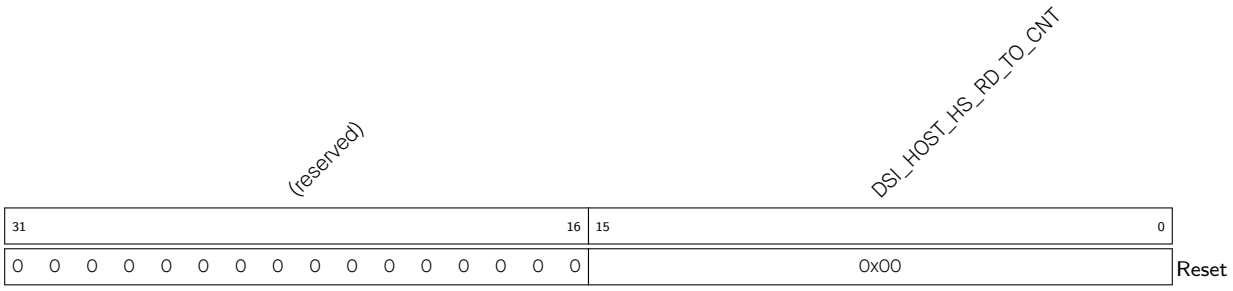
$$\text{TO_CLK_DIVISION} \times \text{DSI_HOST_LPRX_TO_CNT} \times \text{lane_byte_clk_period} \geq 1.1 \times \text{frame data transmission time}$$

For burst mode, pixel packets are time-compressed, leaving more time during HFP to switch from HS to LP and back. If this mode is chosen, the configuration should meet:

$$\text{TO_CLK_DIVISION} \times \text{DSI_HOST_LPRX_TO_CNT} \times \text{lane_byte_clk_period} \geq 1.1 \times \text{line data transmission time}$$

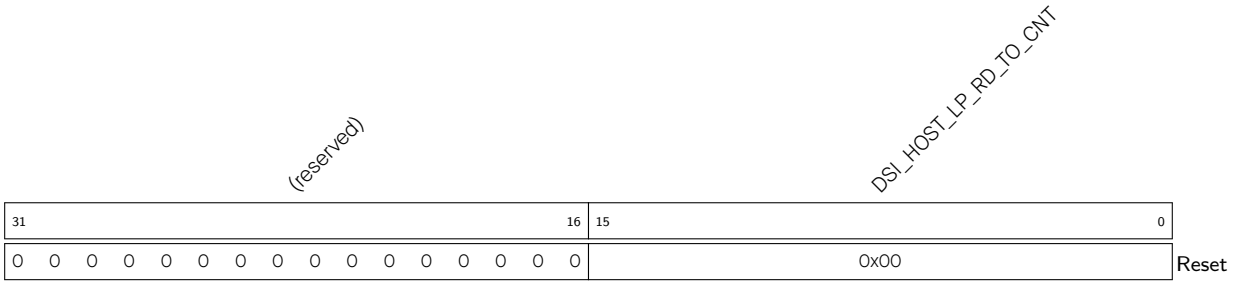
(R/W)

Register 43.64. DSI_HOST_HS_RD_TO_CNT_REG (0x007C)



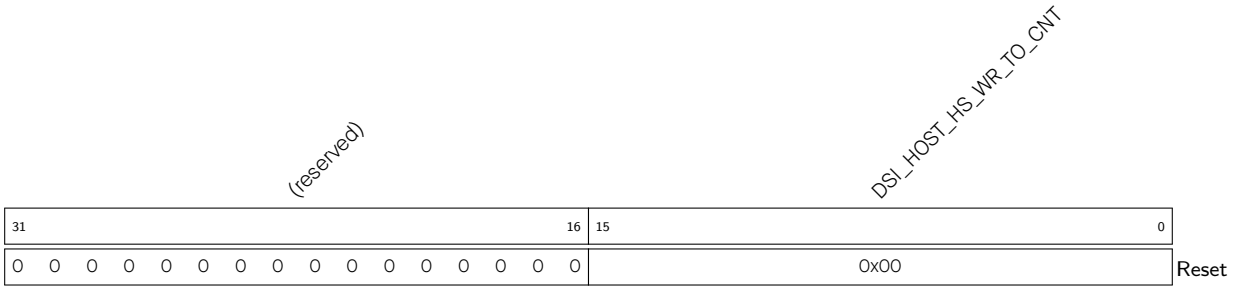
DSI_HOST_HS_RD_TO_CNT Configures the time that DSI Host keeps the link still after sending a HS read. Measured in lane byte clk cycles. Count starts when DPHY enters stop state and causes no interrupts. (R/W)

Register 43.65. DSI_HOST_LP_RD_TO_CNT_REG (0x0080)



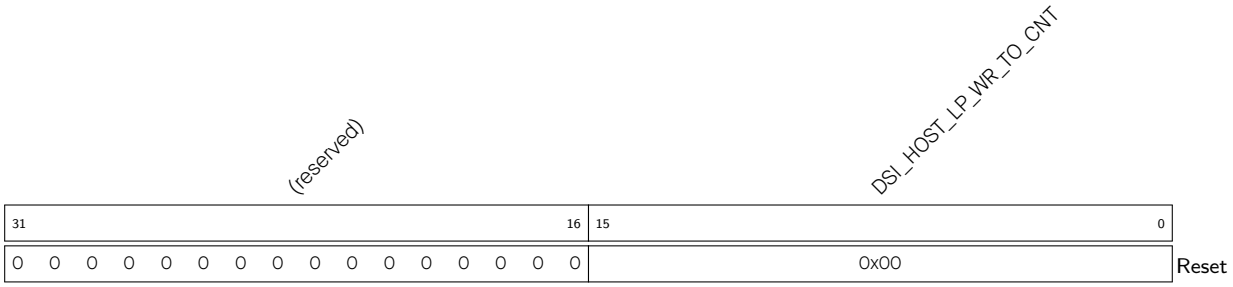
DSI_HOST_LP_RD_TO_CNT Configures the time that DSI Host keeps the link still after sending a LP read. Measured in lane byte clk cycles. Count starts when DPHY enters stop state and causes no interrupts. (R/W)

Register 43.66. DSI_HOST_HS_WR_TO_CNT_REG (0x0084)



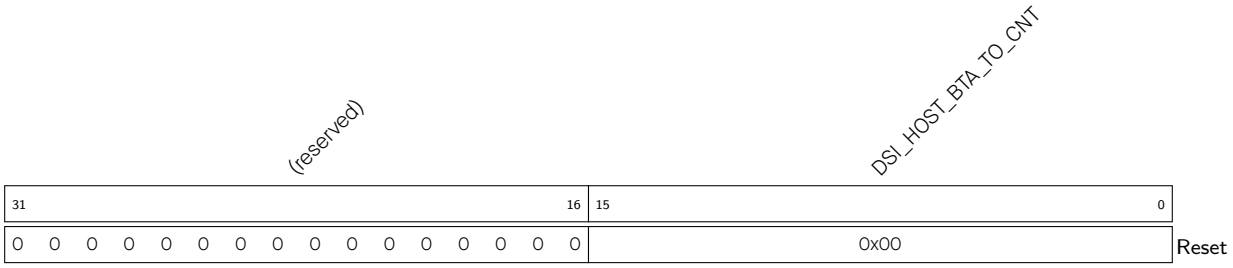
DSI_HOST_HS_WR_TO_CNT Configures the time that DSI Host keeps the link still after sending a HS write. Measured in lane byte clk cycles. Count starts when DPHY enters stop state and causes no interrupts. (R/W)

Register 43.67. DSI_HOST_LP_WR_TO_CNT_REG (0x0088)



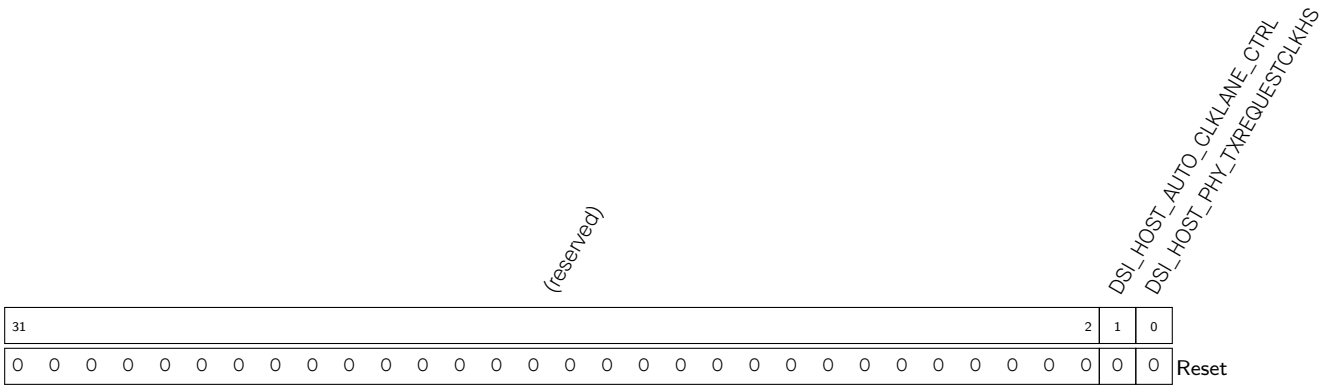
DSI_HOST_LP_WR_TO_CNT Configures the time that DSI Host keeps the link still after sending a LP write. Measured in lane byte clk cycles. Count starts when DPHY enters stop state and causes no interrupts. (R/W)

Register 43.68. DSI_HOST_BTA_TO_CNT_REG (0x008C)



DSI_HOST_BTA_TO_CNT Configures the time that DSI Host keeps the link still after sending a BTA. Measured in lane byte clk cycles. Count starts when DPHY enters stop state and causes no interrupts. (R/W)

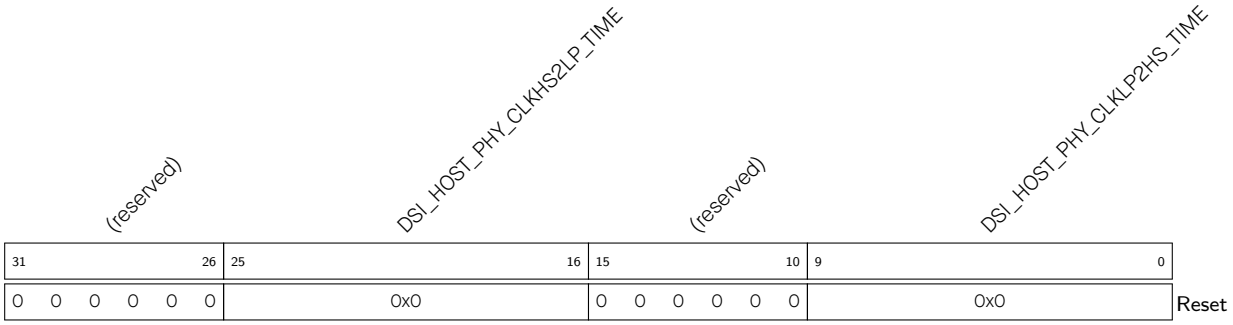
Register 43.69. DSI_HOST_LPCLK_CTRL_REG (0x0094)



DSI_HOST_PHY_TXREQUESTCLKHS Controls the DPY PPI txrequestclkhs signal. (R/W)

DSI_HOST_AUTO_CLKLANE_CTRL Configures whether to enable the automatic mechanism to stop providing clock in the clock lane when time allows. (R/W)

Register 43.70. DSI_HOST_PHY_TMR_LPCLK_CFG_REG (0x0098)



DSI_HOST_PHY_CLKLP2HS_TIME Configures the max time that DPHY clock lane takes to go from LP to HS, mearured in lane byte clk cycles. (R/W)

DSI_HOST_PHY_CLKHS2LP_TIME Configures the max time that DPHY clock lane takes to go from HS to LP, mearured in lane byte clk cycles. (R/W)

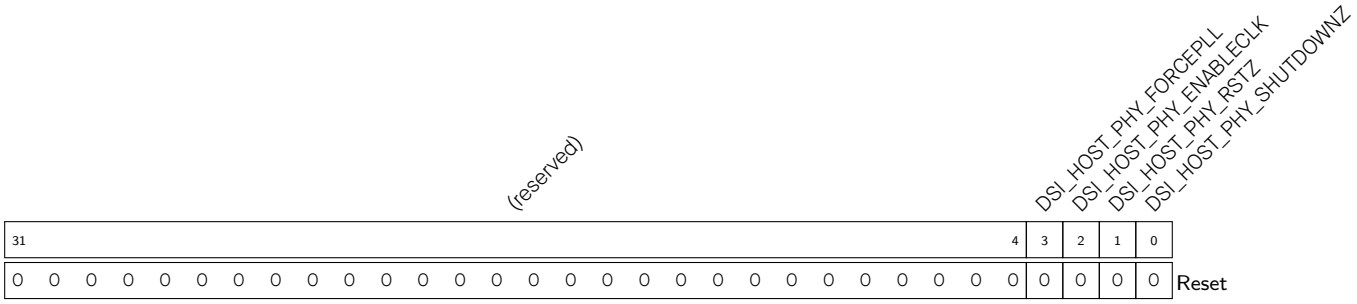
Register 43.71. DSI_HOST_PHY_TMR_CFG_REG (0x009C)



DSI_HOST_PHY_LP2HS_TIME Configures the max time that DPHY data lane takes to go from LP to HS, mearured in lane byte clk cycles. (R/W)

DSI_HOST_PHY_HS2LP_TIME Configures the max time that DPHY data lane takes to go from HS to LP, mearured in lane byte clk cycles. (R/W)

Register 43.72. DSI_HOST_PHY_RSTZ_REG (0x00A0)



DSI_HOST_PHY_SHUTDOWNZ Configures DPHY macro power state.

0: power dowm

1: power up

(R/W)

DSI_HOST_PHY_RSTZ Configures DPHY digital logic reset.

0: reset

1: release reset

(R/W)

DSI_HOST_PHY_ENABLECLK Configures DPHY clock lane module enable.

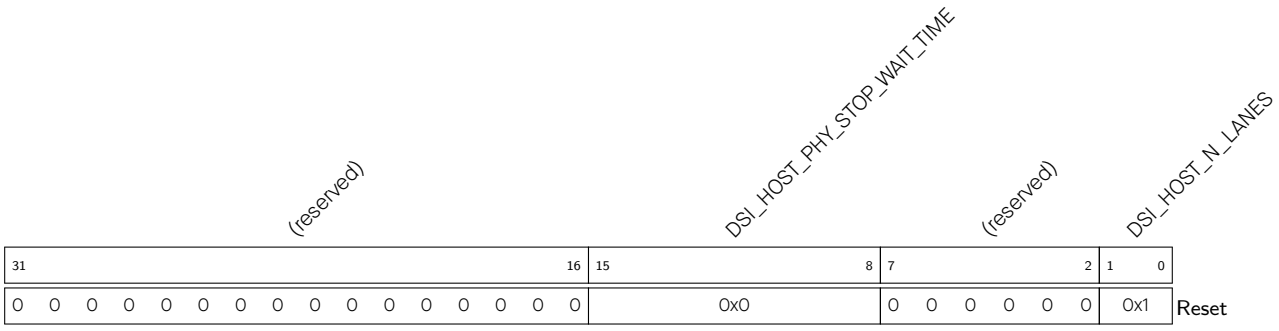
0: disable

1: enable

(R/W)

DSI_HOST_PHY_FORCEPLL Configures DPHY PLL when DPHY is in ULPS. (R/W)

Register 43.73. DSI_HOST_PHY_IF_CFG_REG (0x00A4)



DSI_HOST_N_LANES Configures the number of active data lane.

0: lane 0

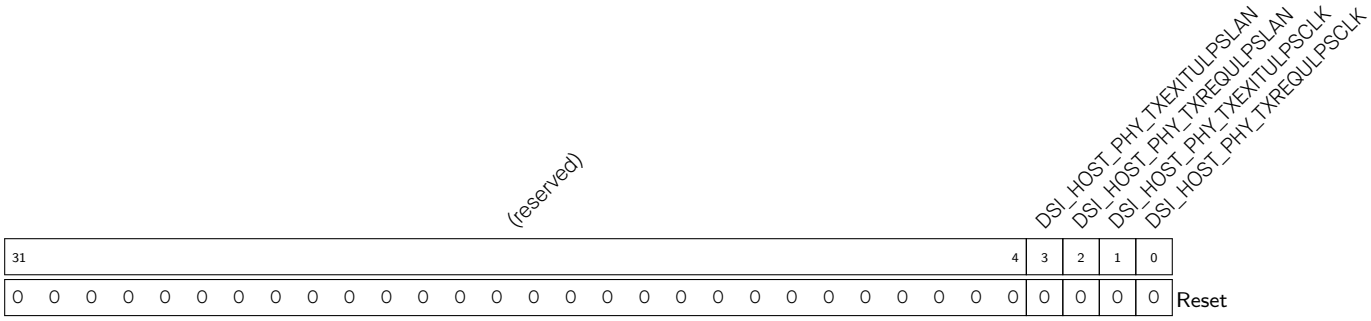
1: lane 0 and 1

others: reserved

(R/W)

DSI_HOST_PHY_STOP_WAIT_TIME NA. (R/W)

Register 43.74. DSI_HOST_PHY_ULPS_CTRL_REG (0x00A8)



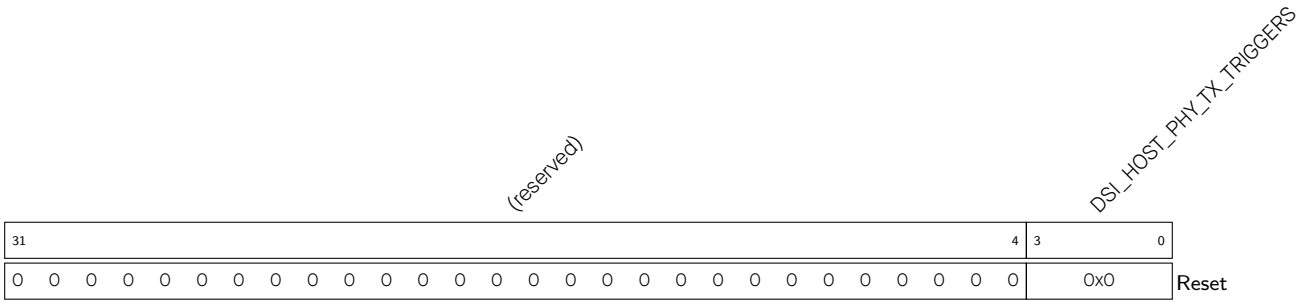
DSI_HOST_PHY_TXREQULPSCLK Controls ULPS mode request on clock lane. (R/W)

DSI_HOST_PHY_TXEXITULPSCLK Controls ULPS mode exit on clock lane. (R/W)

DSI_HOST_PHY_TXREQULPSLAN Controls ULPS mode request on all active data lane. (R/W)

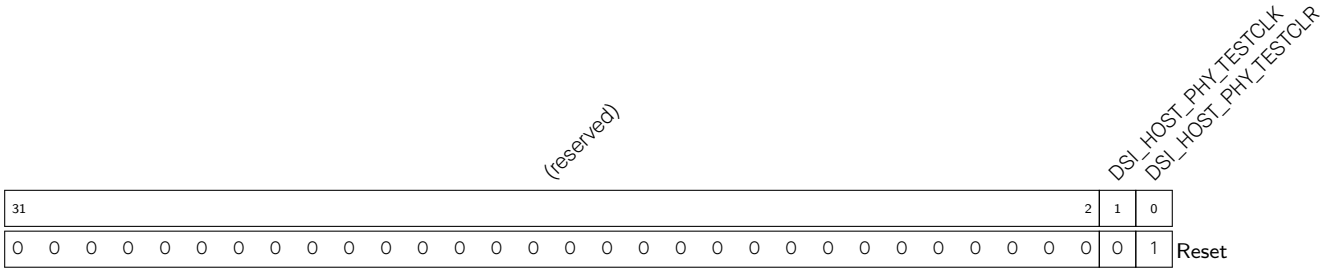
DSI_HOST_PHY_TXEXITULPSLAN Controls ULPS mode exit on all active data lane. (R/W)

Register 43.75. DSI_HOST_PHY_TX_TRIGGERS_REG (0x00AC)



DSI_HOST_PHY_TX_TRIGGERS Controls the trigger transmission. (R/W)

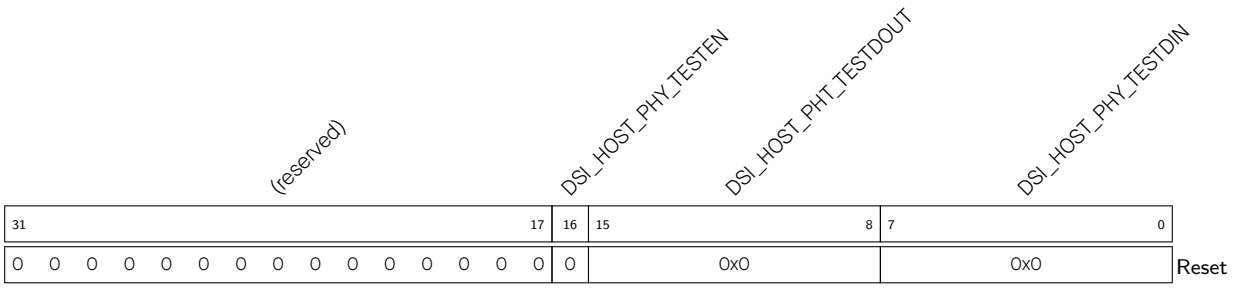
Register 43.76. DSI_HOST_PHY_TST_CTRL0_REG (0x00B4)



DSI_HOST_PHY_TESTCLR Controls test code interface TESTCLR. (R/W)

DSI_HOST_PHY_TESTCLK Controls test code interface TESTCLK. (R/W)

Register 43.77. DSI_HOST_PHY_TST_CTRL1_REG (0x00B8)

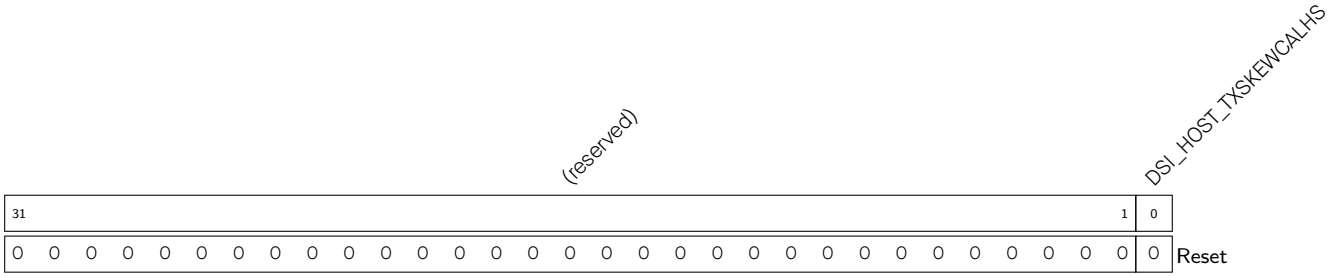


DSI_HOST_PHY_TESTDIN Controls test code interface TESTDIN. (R/W)

DSI_HOST_PHT_TESTDOUT Represents test code interface TESTDOUT. (RO)

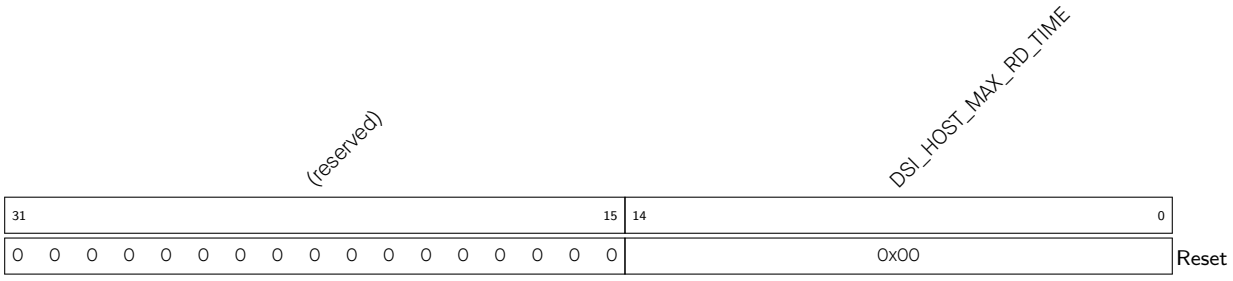
DSI_HOST_PHY_TESTEN Controls test code interface TESTEN. (R/W)

Register 43.78. DSI_HOST_PHY_CAL_REG (0x00CC)



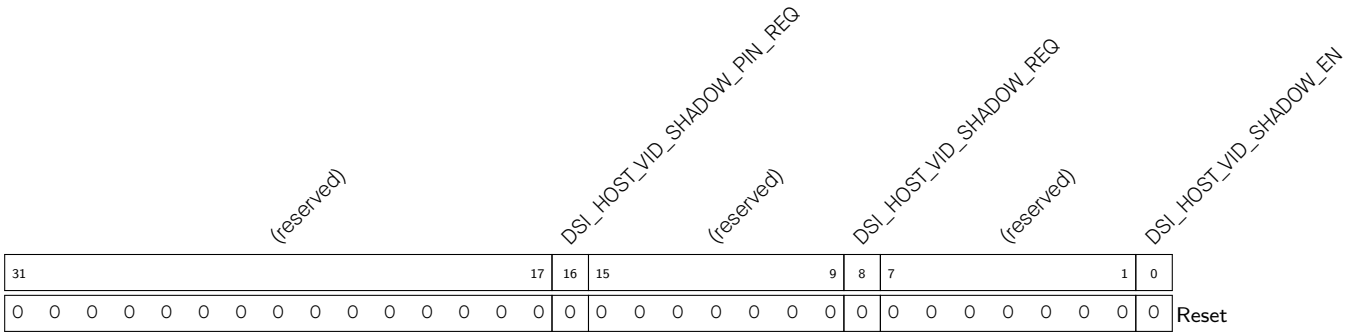
DSI_HOST_TXSKEWCALHS Configures HS skew calibration (assuming that DPHY is in stop state). (R/W)

Register 43.79. DSI_HOST_PHY_TMR_RD_CFG_REG (0x00F4)



DSI_HOST_MAX_RD_TIME Configures the max time required to perform a read command in lane byte clock cycle. This register can only be modified when no read command is in prograss. (R/W)

Register 43.80. DSI_HOST_VID_SHADOW_CTRL_REG (0x0100)

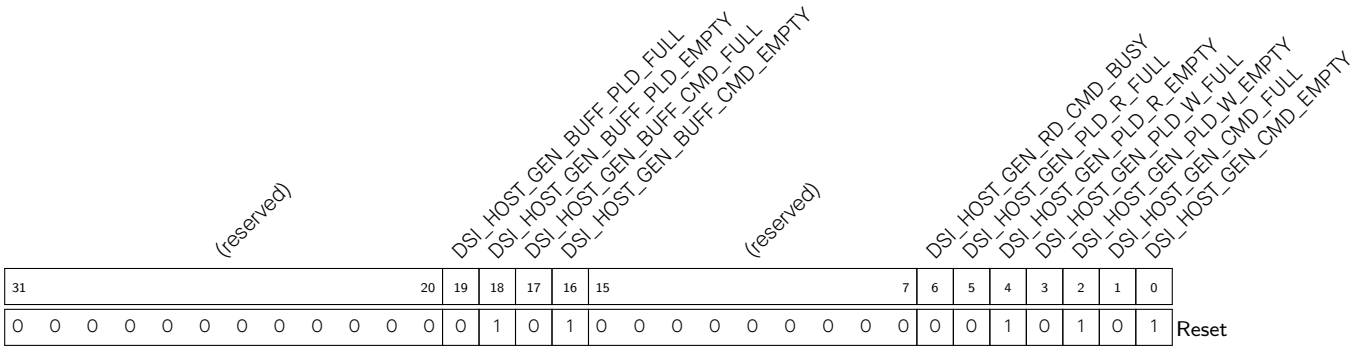


DSI_HOST_VID_SHADOW_EN Controls whether the DPI receives the active configuration from shadow register. (R/W)

DSI_HOST_VID_SHADOW_REQ Controls to update DPI register from regbank to shadow register. AUto cleared after request done. (R/W)

DSI_HOST_VID_SHADOW_PIN_REQ Controls the video request is done by external pin. When set to 1, DSI_HOST_VID_SHADOW_REQ is ignored. (R/W)

Register 43.81. DSI_HOST_CMD_PKT_STATUS_REG (0x0074)



DSI_HOST_GEN_CMD_EMPTY Represents the empty status of generic command FIFO. (RO)

DSI_HOST_GEN_CMD_FULL Represents the full status of generic command FIFO. (RO)

DSI_HOST_GEN_PLD_W_EMPTY Represents the empty status of generic write payload FIFO. (RO)

DSI_HOST_GEN_PLD_W_FULL Represents the full status of generic write payload FIFO. (RO)

DSI_HOST_GEN_PLD_R_EMPTY Represents the empty status of generic read payload FIFO. (RO)

DSI_HOST_GEN_PLD_R_FULL Represents the full status of generic read payload FIFO. (RO)

DSI_HOST_GEN_RD_CMD_BUSY Represents that a read command is issued and cleared when the entire response is stored in the FIFO for Generic Interface. (RO)

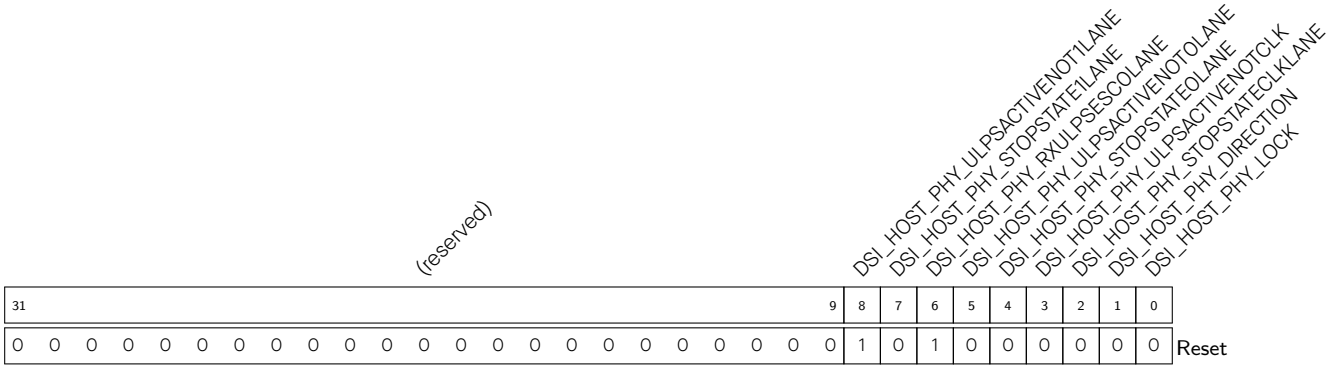
DSI_HOST_GEN_BUFF_CMD_EMPTY Represents the empty status of generic command internal buffer. (RO)

DSI_HOST_GEN_BUFF_CMD_FULL Represents the full status of generic command internal buffer. (RO)

DSI_HOST_GEN_BUFF_PLD_EMPTY Represents the empty status of generic payload internal buffer. (RO)

DSI_HOST_GEN_BUFF_PLD_FULL Represents the full status of generic payload internal buffer. (RO)

Register 43.82. DSI_HOST_PHY_STATUS_REG (0x00B0)



DSI_HOST_PHY_LOCK Represents the status of DPHY phylock. (RO)

DSI_HOST_PHY_DIRECTION Represents the status of phydirection. (RO)

DSI_HOST_PHY_STOPSTATECLKLANE Represents the status of DPHY phystopstateclklane. (RO)

DSI_HOST_PHY_ULPSACTIVENOTCLK Represents the status of DPHY phyulpsactivenotclk. (RO)

DSI_HOST_PHY_STOPSTATEOLANE Represents the status of DPHY phystopstate0lane. (RO)

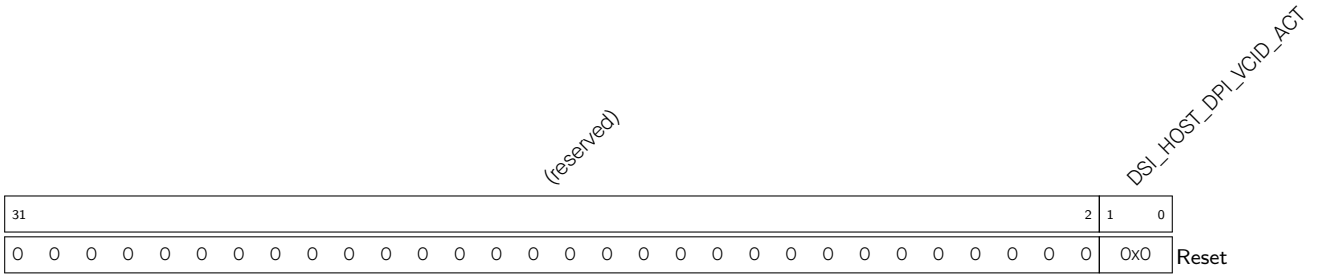
DSI_HOST_PHY_ULPSACTIVENOTOLANE Represents the status of DPHY ulpsactivenot0lane. (RO)

DSI_HOST_PHY_RXULPSESCOLANE Represents the status of DPHY rxulpsesc0lane. (RO)

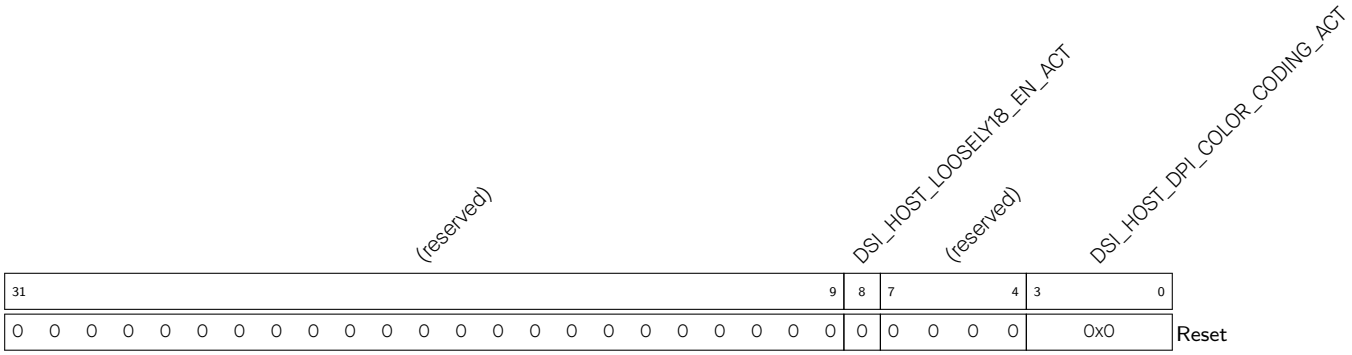
DSI_HOST_PHY_STOPSTATE1LANE Represents the status of DPHY phystopstate1lane. (RO)

DSI_HOST_PHY_ULPSACTIVENOT1LANE Represents the status of DPHY ulpsactivenot1lane. (RO)

Register 43.83. DSI_HOST_DPI_VCID_ACT_REG (0x010C)


(reserved)
DSI_HOST_DPI_VCID_ACT
31 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

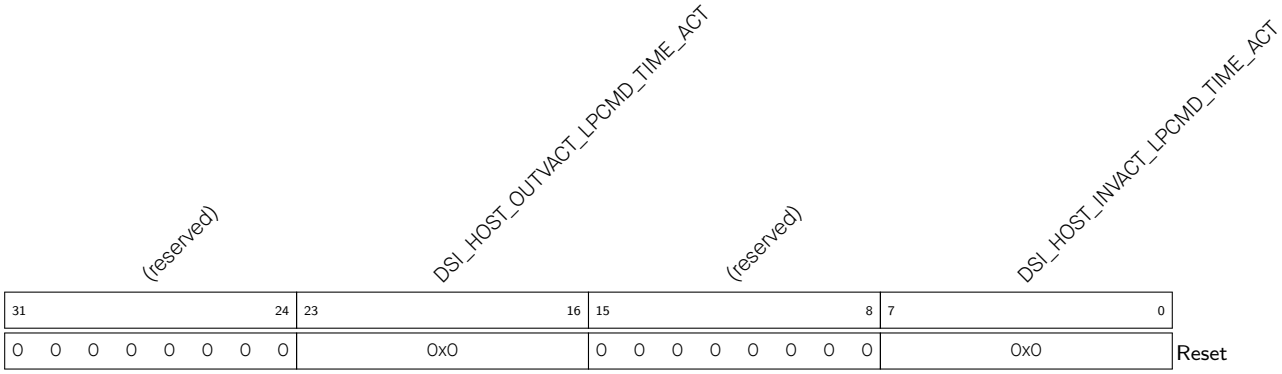
Register 43.84. DSI_HOST_DPI_COLOR_CODING_ACT_REG (0x0110)



DSI_HOST_DPI_COLOR_CODING_ACT Represent the actual DPI color coding. (RO)

DSI_HOST_LOOSELY18_EN_ACT Represent the actual 18-bit loosely packed enable. (RO)

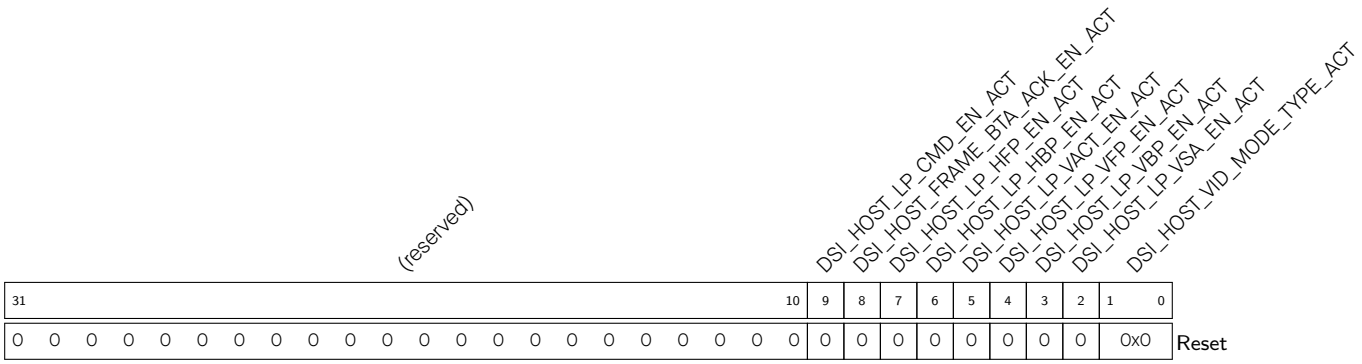
Register 43.85. DSI_HOST_DPI_LP_CMD_TIM_ACT_REG (0x0118)



DSI_HOST_INVACT_LPCMD_TIME_ACT Represents the actual value of INVACT_LPCMD_TIME. (RO)

DSI_HOST_OUTVACT_LPCMD_TIME_ACT Represents the actual value of OUT_VACT_LPCMD_TIME. (RO)

Register 43.86. DSI_HOST_VID_MODE_CFG_ACT_REG (0x0138)



DSI_HOST_VID_MODE_TYPE_ACT Represents the actual value of VID_MODE_TYPE. (RO)

DSI_HOST_LP_VSA_EN_ACT Represents the actual value of LP_VSA_EN. (RO)

DSI_HOST_LP_VBP_EN_ACT Represents the actual value of LP_VBP_EN. (RO)

DSI_HOST_LP_VFP_EN_ACT Represents the actual value of LP_VFP_EN. (RO)

DSI_HOST_LP_VACT_EN_ACT Represents the actual value of LP_VACT_EN. (RO)

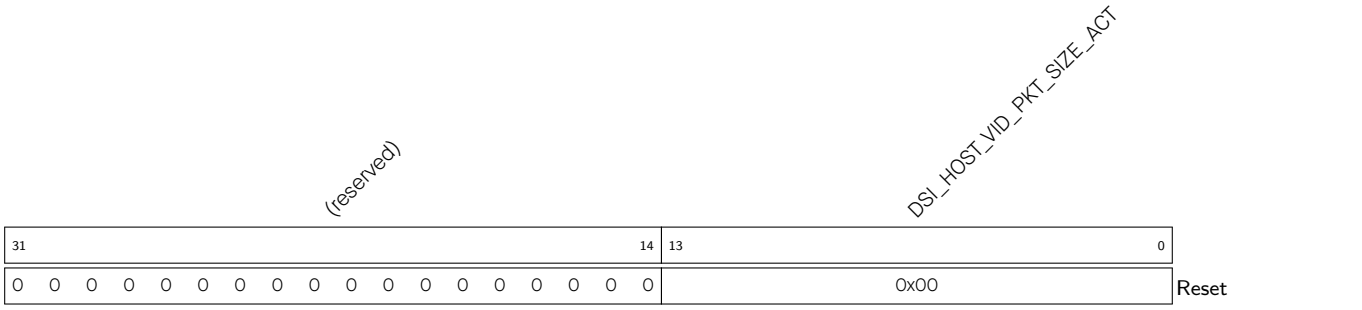
DSI_HOST_LP_HBP_EN_ACT Represents the actual value of LP_HBP_EN. (RO)

DSI_HOST_LP_HFP_EN_ACT Represents the actual value of LP_HFP_EN. (RO)

DSI_HOST_FRAME_BTA_ACK_EN_ACT Represents the actual value of FRAME_BTA_ACK_EN. (RO)

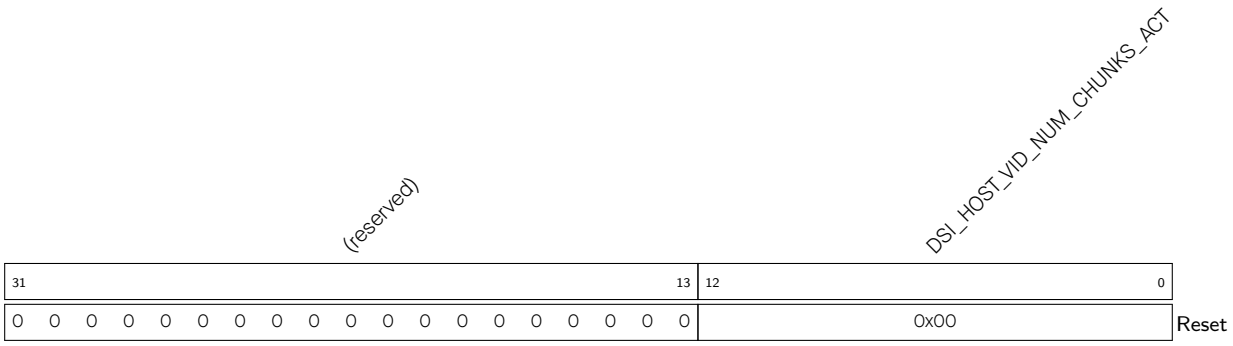
DSI_HOST_LP_CMD_EN_ACT Represents the actual value of LP_CMD_EN. (RO)

Register 43.87. DSI_HOST_VID_PKT_SIZE_ACT_REG (0x013C)



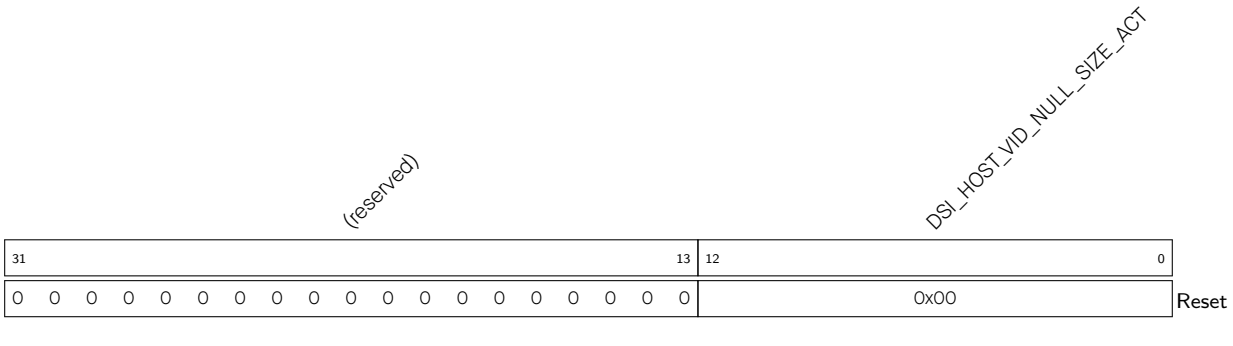
DSI_HOST_VID_PKT_SIZE_ACT Represents the actual value of VID_PKT_SIZE. (RO)

Register 43.88. DSI_HOST_VID_NUM_CHUNKS_ACT_REG (0x0140)



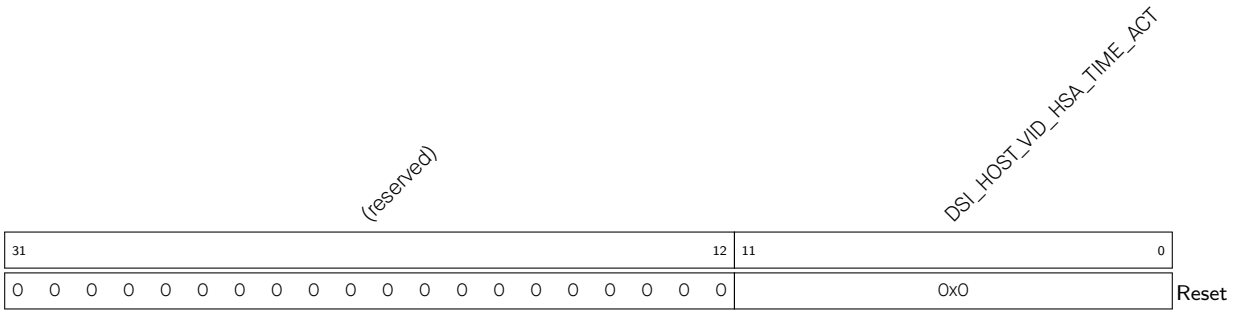
DSI_HOST_VID_NUM_CHUNKS_ACT Represents the actual value of VID_NUM_CHUNKS_ACT. (RO)

Register 43.89. DSI_HOST_VID_NULL_SIZE_ACT_REG (0x0144)



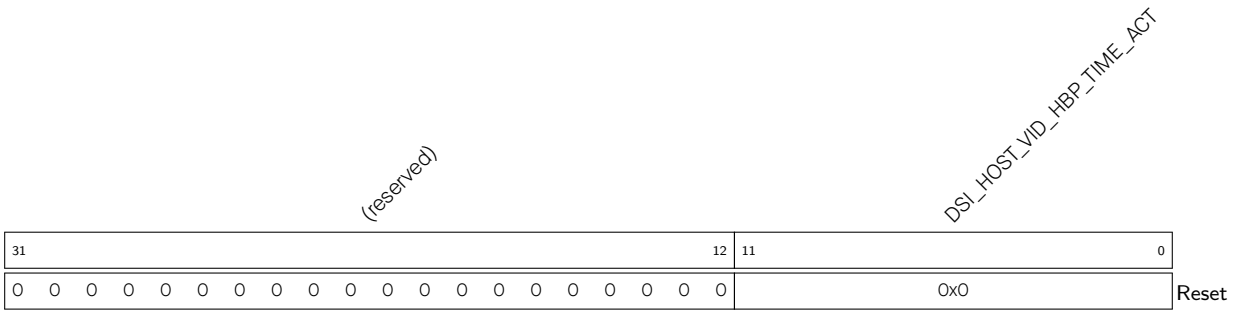
DSI_HOST_VID_NULL_SIZE_ACT Represents the actual value of VID_NULL_SIZE. (RO)

Register 43.90. DSI_HOST_VID_HSA_TIME_ACT_REG (0x0148)



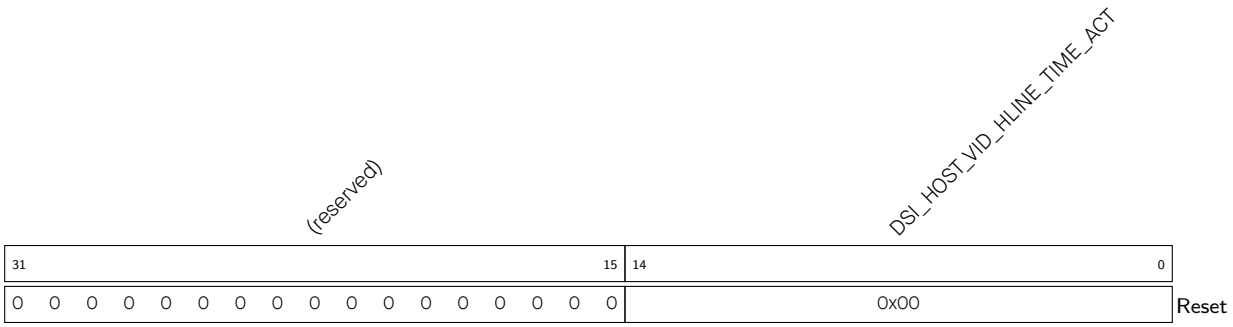
DSI_HOST_VID_HSA_TIME_ACT Represents the actual value of VID_HSA_TIME. (RO)

Register 43.91. DSI_HOST_VID_HBP_TIME_ACT_REG (0x014C)



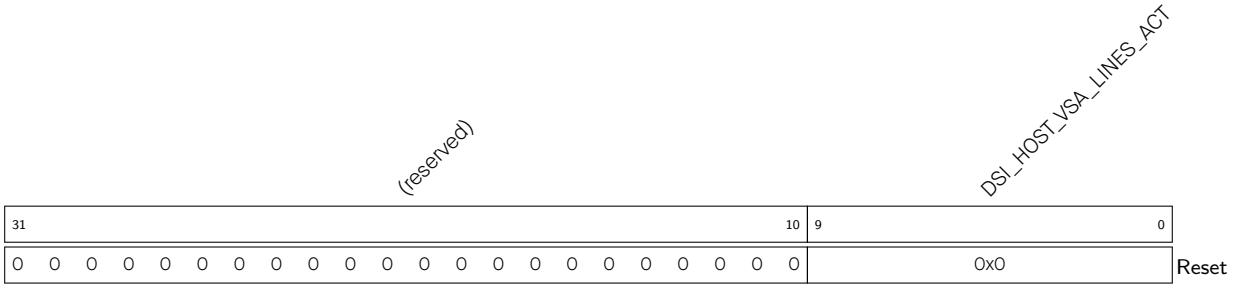
DSI_HOST_VID_HBP_TIME_ACT Represents the actual value of VID_HBP_TIME. (RO)

Register 43.92. DSI_HOST_VID_HLINE_TIME_ACT_REG (0x0150)



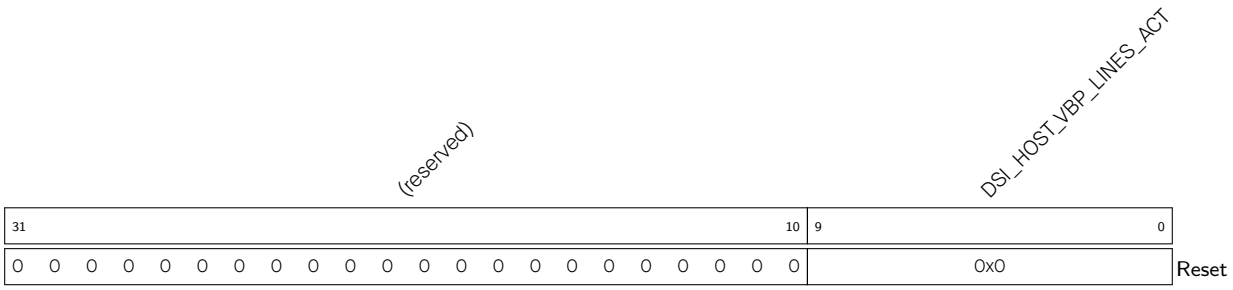
DSI_HOST_VID_HLINE_TIME_ACT Represents the actual value of VID_HLINE_TIME. (RO)

Register 43.93. DSI_HOST_VID_VSA_LINES_ACT_REG (0x0154)



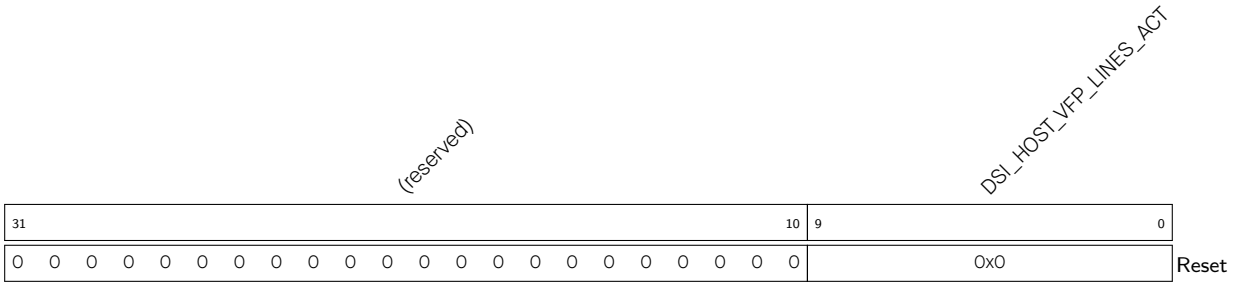
DSI_HOST_VSA_LINES_ACT Represents the actual value of VID_VSA_TIME. (RO)

Register 43.94. DSI_HOST_VID_VBP_LINES_ACT_REG (0x0158)



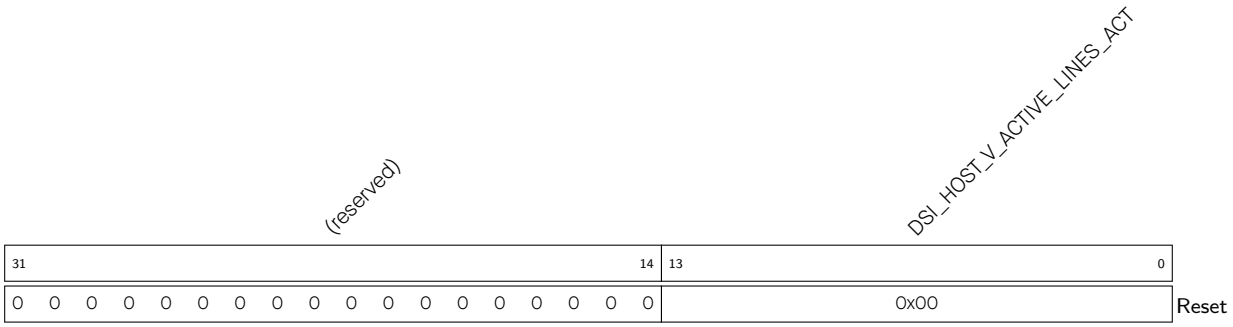
DSI_HOST_VBP_LINES_ACT Represents the actual value of VID_VBP_TIME. (RO)

Register 43.95. DSI_HOST_VID_VFP_LINES_ACT_REG (0x015C)



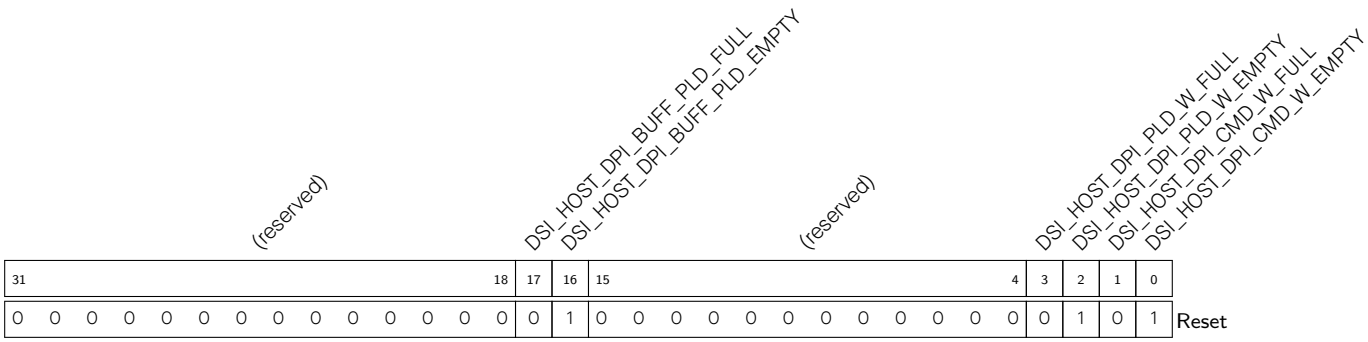
DSI_HOST_VFP_LINES_ACT Represents the actual value of VID_VFP_TIME. (RO)

Register 43.96. DSI_HOST_VID_VACTIVE_LINES_ACT_REG (0x0160)



DSI_HOST_V_ACTIVE_LINES_ACT Represents the actual value of VID_VACTIVE_TIME. (RO)

Register 43.97. DSI_HOST_VID_PKT_STATUS_REG (0x0168)



DSI_HOST_DPI_CMD_W_EMPTY Represents the empty status of write command FIFO for video mode. (RO)

DSI_HOST_DPI_CMD_W_FULL Represents the full status of write command FIFO for video mode. (RO)

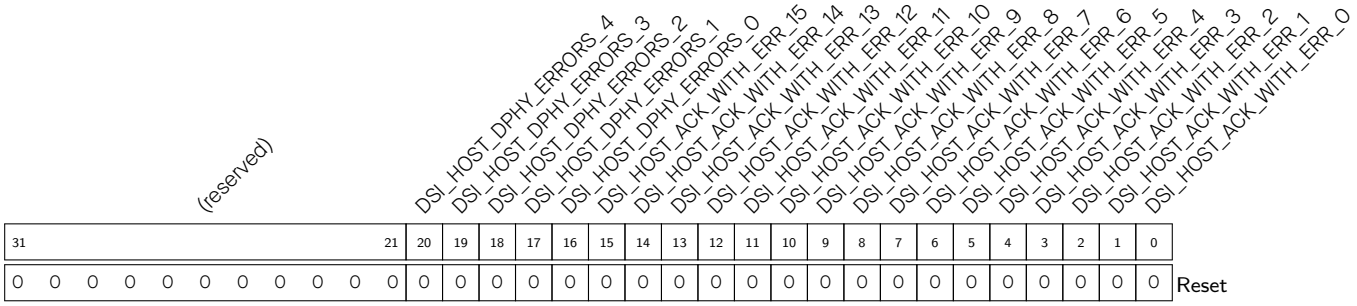
DSI_HOST_DPI_PLD_W_EMPTY Represents the empty status of write payload FIFO for video mode. (RO)

DSI_HOST_DPI_PLD_W_FULL Represents the full status of write payload FIFO for video mode. (RO)

DSI_HOST_DPI_BUFF_PLD_EMPTY Represents the empty status of payload internal buffer for video mode. (RO)

DSI_HOST_DPI_BUFF_PLD_FULL Represents the full status of payload internal buffer for video mode. (RO)

Register 43.98. DSI_HOST_INT_STO_REG (0x00BC)



DSI_HOST_ACK_WITH_ERR_0 Represents SoT error from the Acknowledge report. (RO)

DSI_HOST_ACK_WITH_ERR_1 Represents SoT Sync error from the Acknowledge report. (RO)

DSI_HOST_ACK_WITH_ERR_2 Represents EoT error from the Acknowledge report. (RO)

DSI_HOST_ACK_WITH_ERR_3 Represents Escape Mode Entry Command error from the Acknowledge report. (RO)

DSI_HOST_ACK_WITH_ERR_4 Represents LP Transmit Sync error from the Acknowledge report. (RO)

DSI_HOST_ACK_WITH_ERR_5 Represents Periphal timeout error from the Acknowledge report. (RO)

DSI_HOST_ACK_WITH_ERR_6 Represents False Control error from the Acknowledge report. (RO)

DSI_HOST_ACK_WITH_ERR_7 Represents reserved from the Acknowledge report. (RO)

DSI_HOST_ACK_WITH_ERR_8 Represents ECCsingle-bit error from the Acknowledge report. (RO)

DSI_HOST_ACK_WITH_ERR_9 Represents ECC multi-bit error from the Acknowledge report. (RO)

DSI_HOST_ACK_WITH_ERR_10 Represents checksum error (long packet only) from the Acknowledge report. (RO)

DSI_HOST_ACK_WITH_ERR_11 Represents not recognized DSI data type error from the Acknowledge report. (RO)

DSI_HOST_ACK_WITH_ERR_12 Represents invalid DSI VCID error from the Acknowledge report. (RO)

DSI_HOST_ACK_WITH_ERR_13 Represents invalid transmission length error from the Acknowledge report. (RO)

DSI_HOST_ACK_WITH_ERR_14 Represents reserved from the Acknowledge report. (RO)

DSI_HOST_ACK_WITH_ERR_15 Represents DSI protocol violation error from the Acknowledge report. (RO)

Continued on the next page...

Register 43.98. DSI_HOST_INT_ST0_REG (0x00BC)

Continued from the previous page...

DSI_HOST_DPHY_ERRORS_0 Represents ErrEsc eacape entry error from Lane 0. (RO)

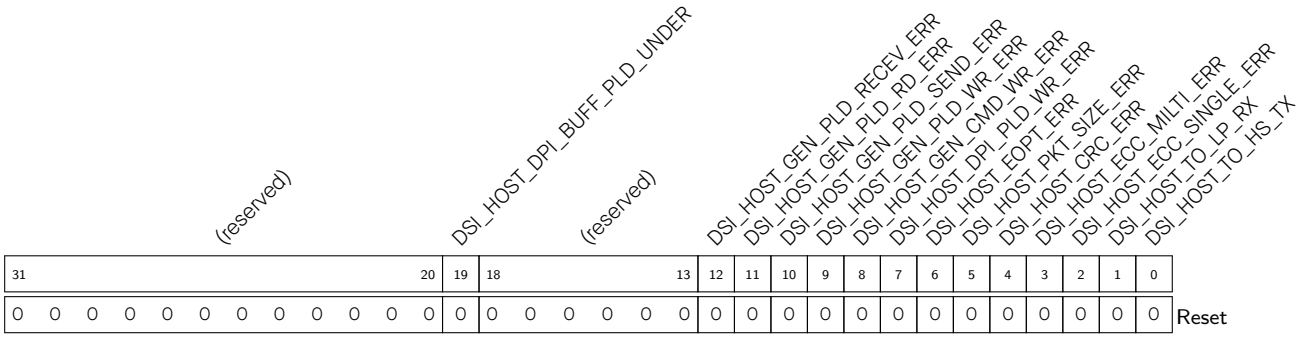
DSI_HOST_DPHY_ERRORS_1 Represents ErrSyncEsc LP data transmission synchronization error from Lane 0. (RO)

DSI_HOST_DPHY_ERRORS_2 Represents ErrControl control error from Lane 0. (RO)

DSI_HOST_DPHY_ERRORS_3 Represents ErrContentionLP0 LP0 contention error from Lane 0. (RO)

DSI_HOST_DPHY_ERRORS_4 Represents ErrContentionLP1 LP1 contention error from Lane 0. (RO)

Register 43.99. DSI_HOST_INT_ST1_REG (0x00C0)



DSI_HOST_TO_HS_TX Represents HP transmission timeout couinter reached the end and contention has been detected. (RO)

DSI_HOST_TO_LP_RX Represents LP reception timeout couinter reached the end and contention has been detected. (RO)

DSI_HOST_ECC_SINGLE_ERR Represents ECC single error has been detected and corrected in a received packet. (RO)

DSI_HOST_ECC_MILTI_ERR Represents ECC multiple error has been detected in a received packet. (RO)

DSI_HOST_CRC_ERR Represents CRC error has been detected in a received packet. (RO)

DSI_HOST_PKT_SIZE_ERR Represents the packet size error has been detected during packet reception. (RO)

DSI_HOST_EOPT_ERR Represents the EoTp packet has not been received at the end of the incommign peripheral transmission. (RO)

DSI_HOST_DPI_PLD_WR_ERR Represents payload FIFO becomes full and data corrupted during DPI pixel line storage. (RO)

DSI_HOST_GEN_CMD_WR_ERR Represents the system tried to write a command through Generic Interface and the FIFO is full. (RO)

DSI_HOST_GEN_PLD_WR_ERR Represents the system tried to write a payload data through Generic Interface and the FIFO is full. (RO)

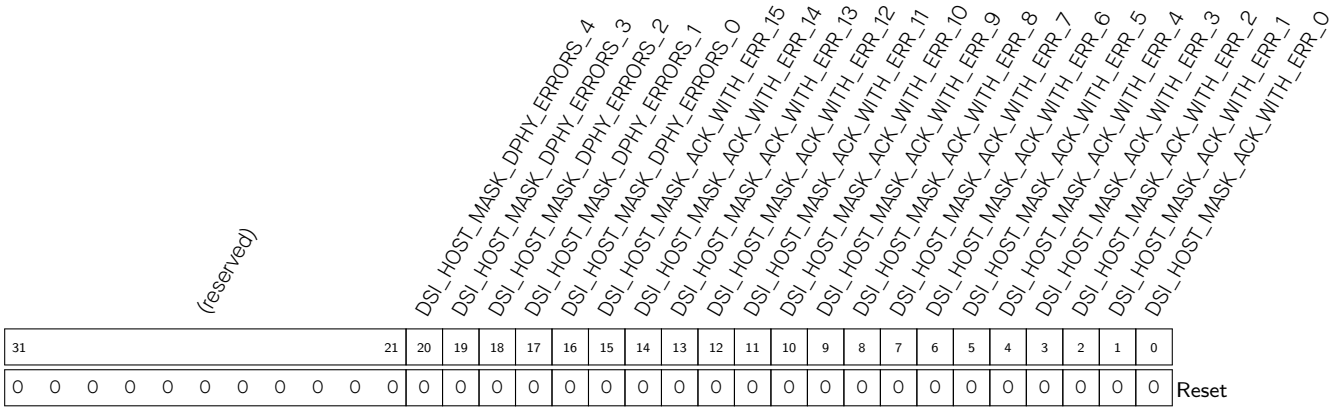
DSI_HOST_GEN_PLD_SEND_ERR Represents the payload FIFO becomes empty and corrupted data is sent during Generic Interface packet build. (RO)

DSI_HOST_GEN_PLD_RD_ERR Represents that payload FIFO becomes empty and the data sent to the interface is corrupted during a DCS read data. (RO)

DSI_HOST_GEN_PLD_RECEV_ERR Represents the payload FIFO becomes full and the received data is corrupted during a Generic Interface packet read back. (RO)

DSI_HOST_DPI_BUFF_PLD_UNDER Represents an underflow has occued when reading payload to build DSI packet for video mode. (RO)

Register 43.100. DSI_HOST_INT_MSK0_REG (0x00C4)



DSI_HOST_MASK_ACK_WITH_ERR_0 Controls the mask for ACK_WITH_ERR_0. (R/W)

DSI_HOST_MASK_ACK_WITH_ERR_1 Controls the mask for ACK_WITH_ERR_1. (R/W)

DSI_HOST_MASK_ACK_WITH_ERR_2 Controls the mask for ACK_WITH_ERR_2. (R/W)

DSI_HOST_MASK_ACK_WITH_ERR_3 Controls the mask for ACK_WITH_ERR_3. (R/W)

DSI_HOST_MASK_ACK_WITH_ERR_4 Controls the mask for ACK_WITH_ERR_4. (R/W)

DSI_HOST_MASK_ACK_WITH_ERR_5 Controls the mask for ACK_WITH_ERR_5. (R/W)

DSI_HOST_MASK_ACK_WITH_ERR_6 Controls the mask for ACK_WITH_ERR_6. (R/W)

DSI_HOST_MASK_ACK_WITH_ERR_7 Controls the mask for ACK_WITH_ERR_7. (R/W)

DSI_HOST_MASK_ACK_WITH_ERR_8 Controls the mask for ACK_WITH_ERR_8. (R/W)

DSI_HOST_MASK_ACK_WITH_ERR_9 Controls the mask for ACK_WITH_ERR_9. (R/W)

DSI_HOST_MASK_ACK_WITH_ERR_10 Controls the mask for ACK_WITH_ERR_10. (R/W)

DSI_HOST_MASK_ACK_WITH_ERR_11 Controls the mask for ACK_WITH_ERR_11. (R/W)

DSI_HOST_MASK_ACK_WITH_ERR_12 Controls the mask for ACK_WITH_ERR_12. (R/W)

DSI_HOST_MASK_ACK_WITH_ERR_13 Controls the mask for ACK_WITH_ERR_13. (R/W)

DSI_HOST_MASK_ACK_WITH_ERR_14 Controls the mask for ACK_WITH_ERR_14. (R/W)

DSI_HOST_MASK_ACK_WITH_ERR_15 Controls the mask for ACK_WITH_ERR_15. (R/W)

DSI_HOST_MASK_DPHY_ERRORS_0 Controls the mask for DPHY_ERRORS_0. (R/W)

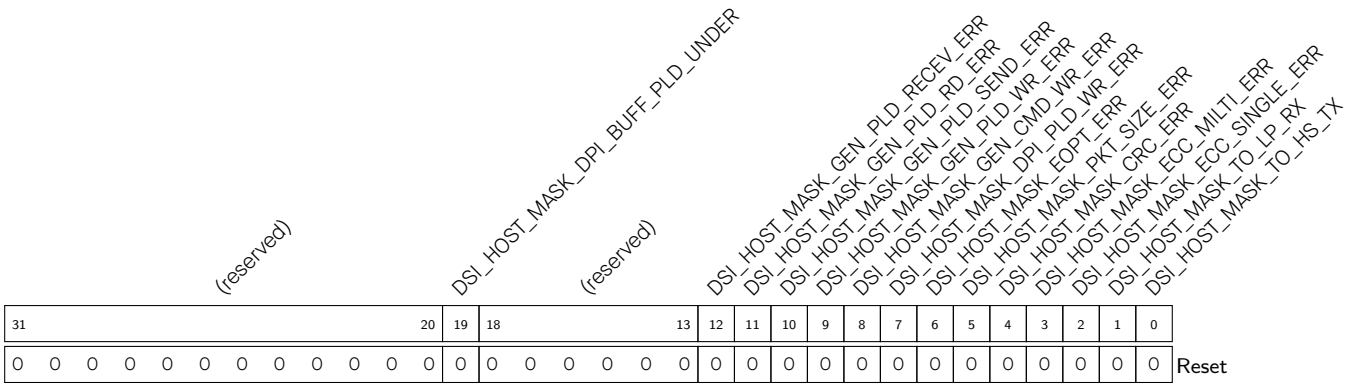
DSI_HOST_MASK_DPHY_ERRORS_1 Controls the mask for DPHY_ERRORS_1. (R/W)

DSI_HOST_MASK_DPHY_ERRORS_2 Controls the mask for DPHY_ERRORS_2. (R/W)

DSI_HOST_MASK_DPHY_ERRORS_3 Controls the mask for DPHY_ERRORS_3. (R/W)

DSI_HOST_MASK_DPHY_ERRORS_4 Controls the mask for DPHY_ERRORS_4. (R/W)

Register 43.101. DSI_HOST_INT_MSK1_REG (0x00C8)



DSI_HOST_MASK_TO_HS_TX Controls the mask for TO_HS_TX. (R/W)

DSI_HOST_MASK_TO_LP_RX Controls the mask for TO_LP_RX. (R/W)

DSI_HOST_MASK_ECC_SINGLE_ERR Controls the mask for ECC_SINGLE_ERR. (R/W)

DSI_HOST_MASK_ECC_MILTI_ERR Controls the mask for ECC_MULTI_ERR. (R/W)

DSI_HOST_MASK_CRC_ERR Controls the mask for CRC_ERR. (R/W)

DSI_HOST_MASK_PKT_SIZE_ERR Controls the mask for PKT_SIZE_ERR. (R/W)

DSI_HOST_MASK_EOPT_ERR Controls the mask for EOPT_ERR. (R/W)

DSI_HOST_MASK_DPI_PLD_WR_ERR Controls the mask for DPI_PLD_WR_ERR. (R/W)

DSI_HOST_MASK_GEN_CMD_WR_ERR Controls the mask for GEN_CMD_WR_ERR. (R/W)

DSI_HOST_MASK_GEN_PLD_WR_ERR Controls the mask for GEN_PLD_WR_ERR. (R/W)

DSI_HOST_MASK_GEN_PLD_SEND_ERR Controls the mask for GEN_PLD_SEND_ERR. (R/W)

DSI_HOST_MASK_GEN_PLD_RD_ERR Controls the mask for GEN_PLD_RD_ERR. (R/W)

DSI_HOST_MASK_GEN_PLD_RECEV_ERR Controls the mask for GEN_PLD_RECEV_ERR. (R/W)

DSI_HOST_MASK_DPI_BUFF_PLD_UNDER Controls the mask for DPI_BUFF_PLD_UNDER. (R/W)

Chapter 44

Voice Activity Detection (VAD)

44.1 Introduction

ESP32-P4 integrates a Voice Activity Detection (VAD) module. This module facilitates the hardware implementation of the first-stage algorithm for voice wake-up and other multimedia functions. Additionally, it provides hardware support for low-power voice wake-up solutions.

44.2 Feature List

The VAD module has the following features:

- VAD algorithm processes voice data frame by frame, with each frame containing 256 data points. The data sampling rate is 8 kHz, and the bit width is 16 bits
- 2 KB buffer that stores up to 4 frames of data
- Independent system wake-up source
- Configurable interrupt sources
- Flexible configuration of algorithm parameters

44.3 Architectural Overview

Figure 44.3-1 is the block diagram of the ESP32-P4 VAD module. The VAD module includes:

- Energy threshold check
- FFT (Fast Fourier Transform) calculation
- LTSD (Long-Term Spectral Divergence) calculation

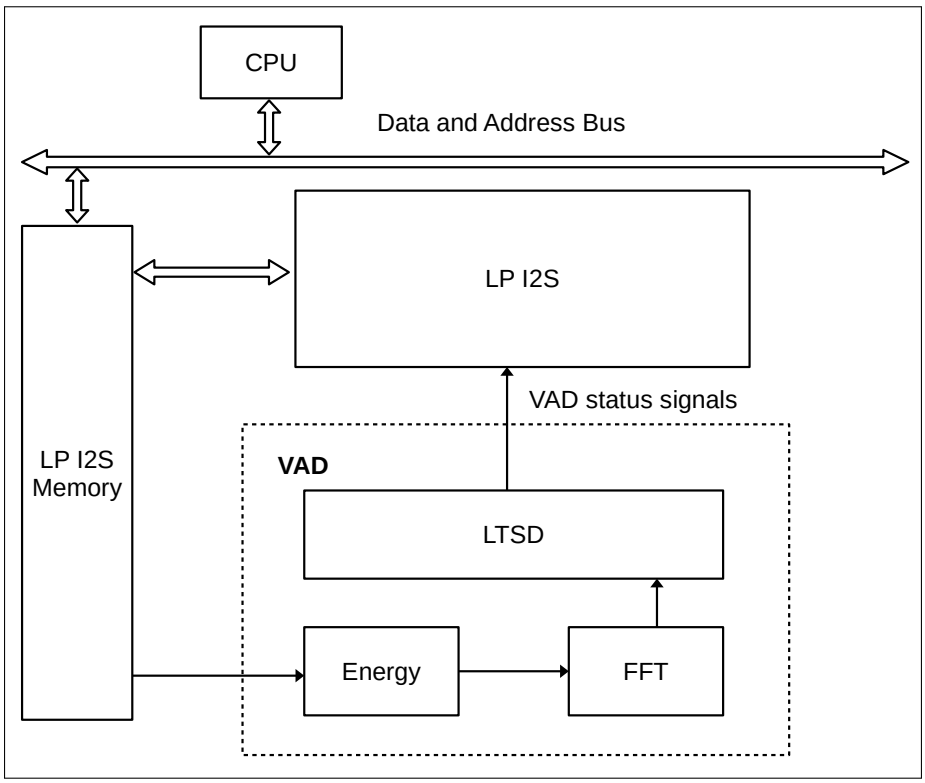


Figure 44.3-1. ESP32-P4 VAD Architecture

The VAD module reads 8 kHz, 16-bit wide audio data from LP I2S memory. It processes the data through three stages: energy detection, FFT, and LTSD. Then the module detects the voice activity status and sends the VAD status signal to the LP I2S module, which is used to generate the corresponding interrupt and wake-up signal.

44.4 Functional Description

The ESP32-P4's VAD module offers flexible and configurable functionality. Users can adjust algorithm parameters and wake-up sources to meet their specific needs.

44.4.1 Algorithm Parameter Configuration

Number of Initial Frames

When the VAD module starts running, it first enters an initialization phase before normal operation. During this phase, it automatically learns noise parameters. The duration of the initialization phase depends on the number of initial frames configured by the user. Proper setting of initial frames can effectively suppress noise and significantly reduce false positives. However, if the initialization phase is extended over too many frames, voice activity within these frames may be missed. Therefore, please configure an appropriate number of initial frames based on actual conditions.

It is recommended to place the device in the target environment noise during the initialization, and set the number of initial frames to 100-200 frames.

Users can configure the number of initial frames via LP_I2S_PARAM_INIT_FRAME_NUM.

Note:
It is recommended to let the VAD module learn noise parameters during the initialization. However, the VAD module will also continuously learn and update noise parameters during normal operation. Therefore, in scenarios where the false positive rate during initial frames is not a concern, the number of initial frames can be reduced, allowing the VAD module to enter the normal operation more quickly.

Energy Threshold

The VAD module sets an energy threshold before the voice activity detection. Setting the threshold can filter out low-energy voice activities (e.g., distant voices) from triggering a wake-up. Appropriate energy threshold setting can effectively reduce false positives and conserve power.

It defines the energy value of a speech frame as the mean square of one-frame data. The VAD module calculates the sum of squares of one-frame data in real-time. If this sum is less than the energy threshold multiplied by 256, subsequent calculations will not be performed. Users can configure the energy threshold via LP_I2S_PARAM_MIN_ENERGY.

Band Energy Check

The band energy check assesses the proportion of band energy within the overall frequency spectrum. Enabling the band energy check in varying environments can help reduce false positives; however, it may also reduce true positives. Users should determine whether to enable or disable the band energy check based on the environment.

Users can enable or disable the band energy check via LP_I2S_PARAM_SKIP_BAND_ENERGY.

- 0: Enable band energy check
- 1: Disable band energy check

Voice Activity Detection

The VAD module has two voice activity status: voice activity status and non-voice activity status, managed by voice frame counter speech_count and non-voice frame counter silent_count. These counters control transitions between the status as follows:

- **Non-voice activity status:**
  - When voice activities are detected in the current frame, the count value of speech_count increases by 1.
  - When voice activities are not detected in the current frame, the count value of speech_count decreases by 1 (down to a minimum of 0).
  - If the count value of speech_count is greater than LP_I2S_PARAM_HANGOVER_SPEECH, the VAD module enters the voice activity status and resets silent_count to 0.
  - Under the non-voice activity status, the VAD module continuously learns and updates noise information.
- **Voice activity status:**
  - When voice activities are detected in the current frame, the count value of speech_count increases by 1, and the count value of silent_count decreases by 1 (down to a minimum of 0).

- When voice activities are not detected in the current frame, the count value of silent_count increases by 1:
  - If the count value of speech_count is greater than LP_I2S_PARAM_MIN_SPEECH_COUNT, and
    - The count value of silent_count is greater than LP_I2S_PARAM_HANGOVER_SILENT, the VAD module transitions to the non-voice activity status and resets both speech_count and silent_count to 0;
    - The count value of silent_count is less than LP_I2S_PARAM_HANGOVER_SILENT, the VAD module maintains the voice activity status.
  - If the count value of speech_count is less than or equal to LP_I2S_PARAM_MIN_SPEECH_COUNT, the VAD module transitions directly to the non-voice activity status and resets speech_count to 0.
- Under the voice activity status, the VAD module pauses updating noise information. When the count value of speech_count exceeds LP_I2S_PARAM_MAX_SPEECH_COUNT, indicating stability in the new environment, it resumes learning and updating noise information.

Status Registers

The VAD module provides several key operational variables as status registers (read-only) to facilitate user debugging. The status registers include:

- LP_I2S_VAD_FLAG: Represents the current voice activity detection status. 1 represents the VAD module is in the voice activity status, while 0 represents it is in the non-voice activity status.
- LP_I2S_ENERGY_ENOUGH: Represents whether the current frame passes the energy threshold check. 1 represents the current frame passes the energy threshold check, while 0 represents not pass.
- LP_I2S_SPEECH_COUNT_OB: Represents the count value of the voice frame counter, speech_count.
- LP_I2S_SILENT_COUNT_OB: Represents the count value of the non-voice frame counter, silent_count.

44.4.2 Wake-up Source Configuration

When the VAD module is in the voice activity status, it triggers the wake-up source LP_I2S_WAKEUP. Users can configure the wake-up source (please refer to Chapter 50 *LP I2S Controller*), enabling the VAD to become one of the wake-up sources for ESP32-P4.

44.5 Interrupts

ESP32-P4's VAD module shares the same interrupt signal LP_I2S_INTR with the LP I2S module. The following interrupt sources can generate the interrupt signal:

- LP_I2S_VAD_DONE_INT: Triggered when the VAD completes processing one frame of data.
- LP_I2S_VAD_RESET_DONE_INT: Triggered when the VAD is reset.

Each interrupt source can be configured by a common set of registers that are described in Section *Interrupt Configuration Registers*. The specific registers can be found in Section 44.7 *Register Summary*.

44.6 Programming Procedure

44.6.1 Automatic Operation Mode

1. Configure the VAD parameters as described in Section 44.4.1.
2. Enable the VAD automatic detection mode by writing 1 to LP_I2S_VAD_EN.
3. Configure the LP I2S to operate in 16-bit receive mode and start receiving data in 16-bit format. The received data will be stored in the LP I2S memory.
4. The VAD module will automatically perform operations on each frame of data after the LP I2S receives the complete frame.
5. (Optional) When the VAD module triggers a wake-up source/interrupt signal, the LP I2S memory already contains the voice frame that triggered the VAD wake-up, along with the previous three voice frames. Users can read these four frames of voice data from the LP I2S memory via the LP core for subsequent processing. For details on the size of the LP I2S memory and data reading methods, please refer to Chapter 50 *LP I2S Controller* > Section 50.8.3 *Internal Memory*.

44.6.2 Manual Operation Mode

1. Configure the VAD parameters as described in Section 44.4.1.
2. Manually store the voice data in the LP I2S memory. For details, refer to Chapter 50 *LP I2S Controller* > Section 50.8.3 *Internal Memory*.
3. With all the voice data stored in the memory, write 1 to LP_I2S_VAD_FORCE_START, then the VAD will perform operations on one frame of data.
4. When the VAD module completes processing one frame of data, LP_I2S_VAD_DONE_INT interrupt will be generated.

44.7 Register Summary

The addresses in this section are relative to VAD base address provided in Table 9.3-2 in Chapter 9 *System and Memory*.

The abbreviations given in Column **Access** are explained in Section *Access Types for Registers*.

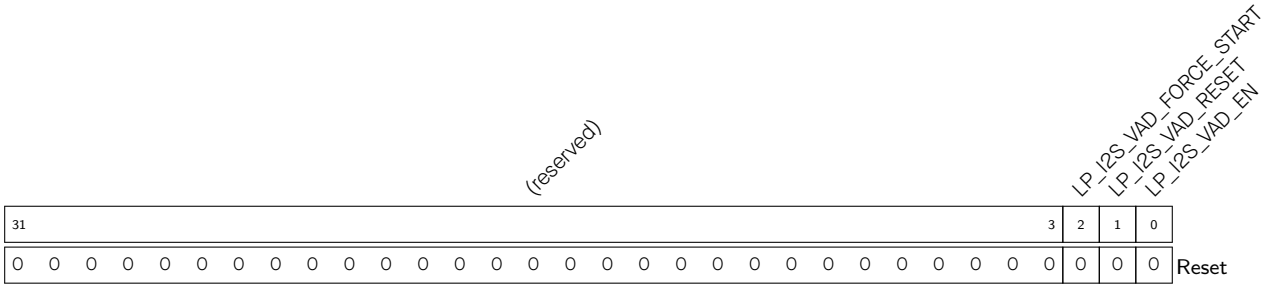
Name	Description	Address	Access
VAD Registers			
LP_I2S_VAD_CONF_REG	Operation control register	0x0000	varies
LP_I2S_VAD_RESULT_REG	Voice activity status register	0x0004	RO
LP_I2S_VAD_PARAM0_REG	Parameter configuration register 0	0x0080	R/W
LP_I2S_VAD_PARAM1_REG	Parameter configuration register 1	0x0084	R/W
LP_I2S_VAD_OBO_REG	Status register for key operational variables	0x00B0	RO

44.8 Registers

The addresses in this section are relative to VAD base address provided in Table 9.3-2 in Chapter 9 *System and Memory*.

For how to program reserved fields, please refer to Section *Programming Reserved Register Field*.

Register 44.1. LP_I2S_VAD_CONF_REG (0x0000)

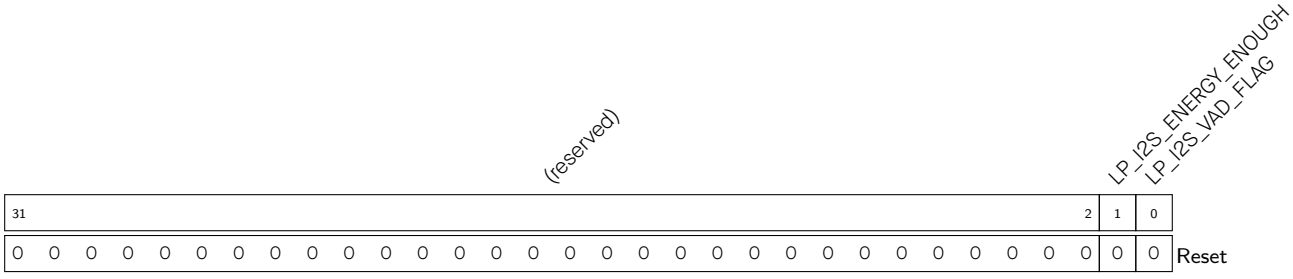


LP_I2S_VAD_EN Configures whether to enable the automatic operation mode.
0: Disable
1: Enable
(R/W)

LP_I2S_VAD_RESET Configures whether to reset the VAD module.
0: Not reset
1: Reset
(WT)

LP_I2S_VAD_FORCE_START Configures whether to trigger the manual operation on one frame of data.
0: Not trigger
1: Trigger
(WT)

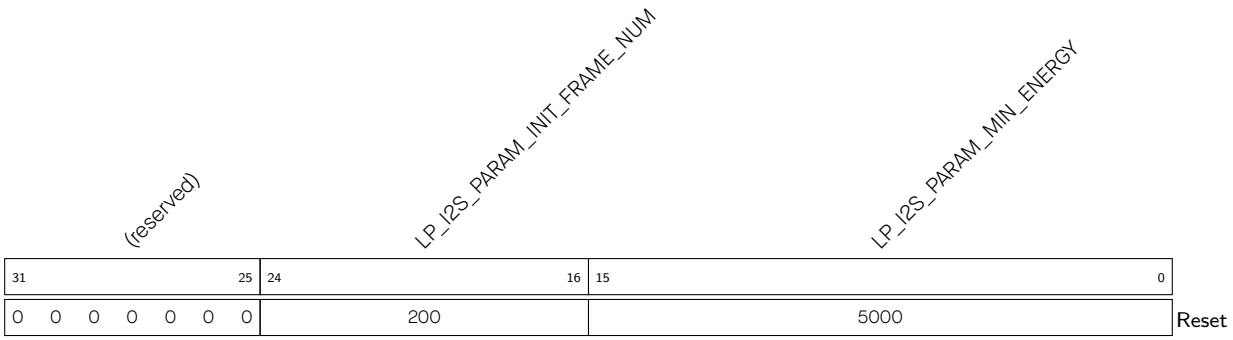
Register 44.2. LP_I2S_VAD_RESULT_REG (0x0004)



LP_I2S_VAD_FLAG Represents the voice activity status. For details, see Section 44.4.1. (RO)

LP_I2S_ENERGY_ENOUGH Represents whether the current frame passes the energy threshold check. For details, see Section 44.4.1. (RO)

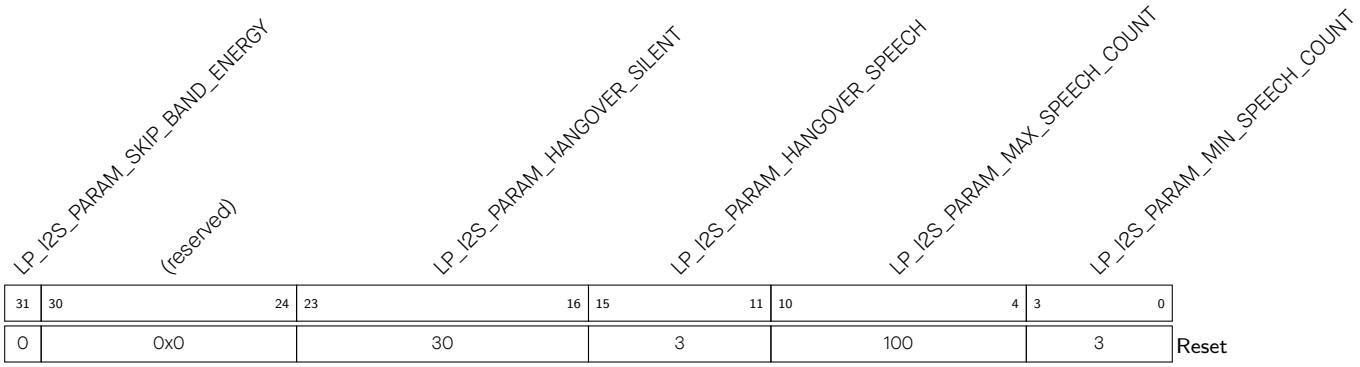
Register 44.3. LP_I2S_VAD_PARAM0_REG (0x0080)



LP_I2S_PARAM_MIN_ENERGY Configures the energy threshold. For details, see Section 44.4.1. (R/W)

LP_I2S_PARAM_INIT_FRAME_NUM Configures the number of initial frames. For details, see Section 44.4.1. (R/W)

Register 44.4. LP_I2S_VAD_PARAM1_REG (0x0084)



LP_I2S_PARAM_MIN_SPEECH_COUNT Configures the transition between voice activity status. For details, see Section 44.4.1. (R/W)

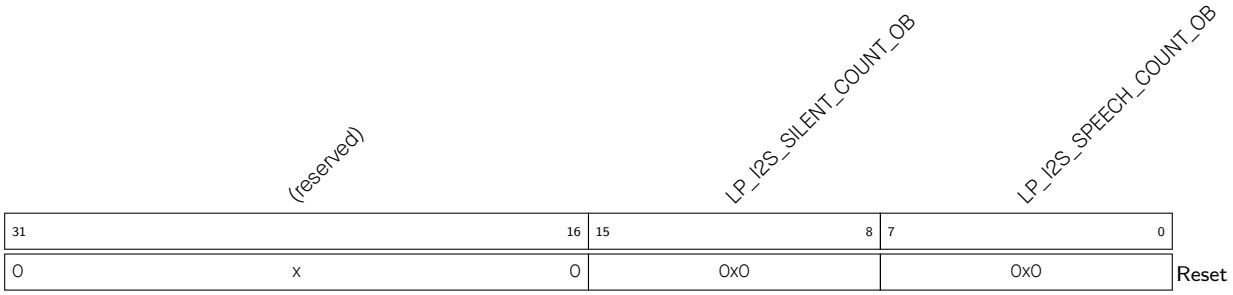
LP_I2S_PARAM_MAX_SPEECH_COUNT Configures the transition between voice activity status. For details, see Section 44.4.1. (R/W)

LP_I2S_PARAM_HANGOVER_SPEECH Configures the transition between voice activity status. For details, see Section 44.4.1. (R/W)

LP_I2S_PARAM_HANGOVER_SILENT Configures the transition between voice activity status. For details, see Section 44.4.1. (R/W)

LP_I2S_PARAM_SKIP_BAND_ENERGY Configures the transition between voice activity status. For details, see Section 44.4.1. (R/W)

Register 44.5. LP_I2S_VAD_OB0_REG (0x00B0)



LP_I2S_SPEECH_COUNT_OB Represents the count value of the voice frame counter, speech_count. For details, see Section 44.4.1. (RO)

LP_I2S_SILENT_COUNT_OB Represents the count value of the non-voice frame counter, silent_count. For details, see Section 44.4.1. (RO)

Part VII

Connectivity Interface

This part addresses the connectivity aspects of the system, describing components related to various communication interfaces like I2C, I2S, SPI, UART, USB, and more. The part also covers interfaces to generate signals used in remote control, motor control, LED control, etc.

Chapter 45

UART Controller (UART)

45.1 Overview

A UART is a character-oriented data link for asynchronous communication between devices. Such communication does not add clock signals to the data sent. Therefore, in order to communicate successfully, the transmitter and the receiver must operate at the same baud rate with the same stop bit(s) and parity bit(s).

A UART data frame usually begins with one start bit, followed by data bits, one parity bit (optional), and one or more stop bits. UART controllers on ESP32-P4 support various lengths of data bits and stop bits. These controllers also support software and hardware flow control as well as GDMA for high-speed data transfer. This allows developers to use multiple UART ports at minimal software cost.

ESP32-P4 has six UART controllers, including five regular UARTs and one low-power (LP) UART. These UARTs are compatible with various UART devices, and support Infrared Data Association (IrDA) and RS485 communication. In this chapter, the five regular UART controllers are referred to as UART*n*, in which *n* denotes 0, 1, 2, 3, or 4. LP UART is the cut-down version of the regular UART, with a separate group of registers. For differences between UART and LP UART, please refer to Table 45.2-1.

45.2 Features

Table 45.2-1 lists the feature comparison between UART and LP UART:

Table 45.2-1. UART and LP UART Feature Comparison

UART Feature	LP UART Feature
Programmable baud rate up to 5 MBaud	
128 x 8 bit RAM respectively for the TX channel and RX channel of a UART controller	20 x 8-bit RAM, shared by the TX FIFO and RX FIFO of LP UART
Full-duplex asynchronous communication	
Data bits (5 to 8 bits)	
Stop bits (1, 1.5, or 2 bits)	
Parity bit	
Special character AT_CMD detection	
RS485 protocol	—
IrDA protocol	—

Cont'd on next page

Table 45.2-1 – cont'd from previous page

UART Feature	LP_UART Feature
High-speed data communication using GDMA	—
Receive timeout	
UART as wakeup source	
Software and hardware flow control	
Three prescalable clock sources: 1. XTAL_CLK 2. RC_FAST_CLK 3. PLL_F80M_CLK	Three prescalable clock sources 1. RC_FAST_CLK 2. XTAL_DIV_CLK 3. PLL_F8M_CLK

The following description mainly covers regular UART controllers.

45.3 UART Structure

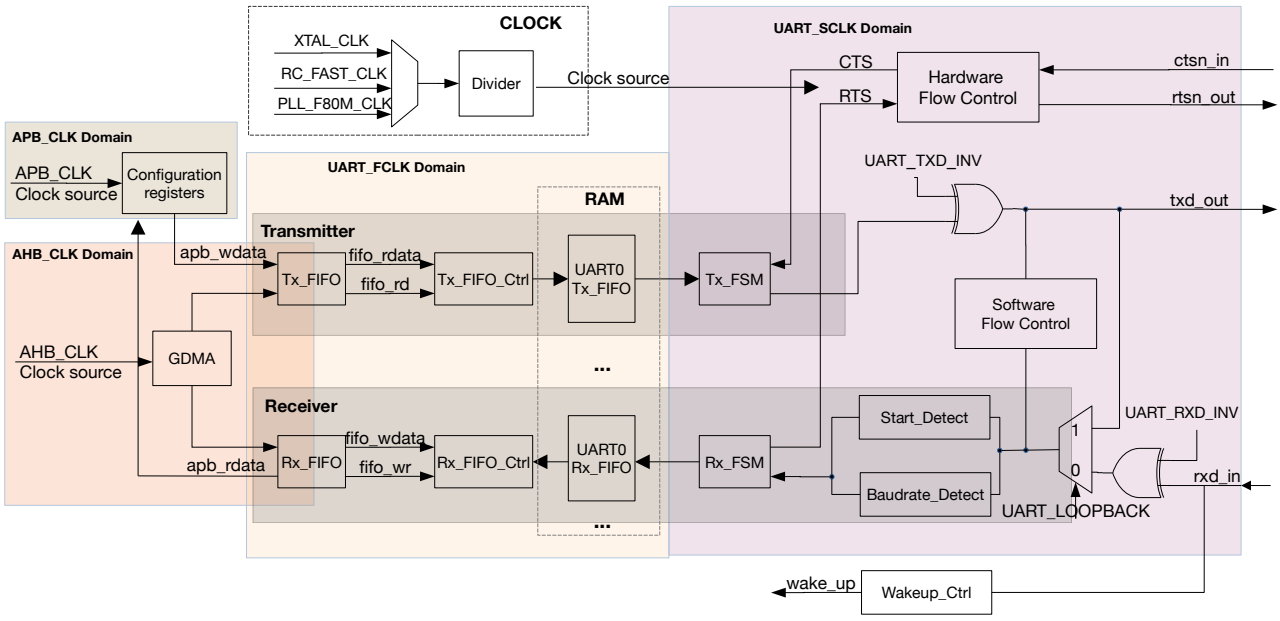


Figure 45.3-1. UART Structure

Figure 45.3-1 shows the basic structure of a UART controller. A UART controller works in four clock domains, namely APB_CLK, AHB_CLK, UART_SCLK, and UART_FCLK. APB_CLK and AHB_CLK are synchronized but with different frequencies (APB_CLK is derived from AHB_CLK by division), and likewise UART_SCLK and UART_FCLK are synchronized but with different frequencies (UART_SCLK is derived from UART_FCLK by division). UART_FCLK has three clock sources: PLL_F80M_CLK, RC_FAST_CLK, and crystal clock XTAL_CLK (for details, please refer to Chapter 11 *Reset and Clock*), which are selected by configuring HP_SYS_UART*n*_CLK_SRC_SEL. The selected clock source is divided by a divider to generate UART_SCLK clock signals. The divisor is configured by HP_SYS_CLKRST_UART*n*_SCLK_DIV_NUM for the integral part, HP_SYS_CLKRST_UART*n*_SCLK_DIV_DENOMINATOR for the denominator of the fractional part, and HP_SYS_CLKRST_UART*n*_SCLK_DIV_NUMERATOR for the numerator of the fractional part. The divisor ranges from 1 ~ 256. Only regular UART has such a divider; LP UART does not.

A UART controller can be broken down into two parts based on functions: a transmitter and a receiver.

The transmitter contains a TX FIFO (i.e., Tx_FIFO in Figure 45.3-1), which buffers data to be sent. Software can write data to Tx_FIFO via the APB bus, or move data to Tx_FIFO using GDMA. Tx_FIFO_Ctrl controls writing and reading Tx_FIFO. When Tx_FIFO is not empty, Tx_FSM reads data bits in the data frame via Tx_FIFO_Ctrl, and converts them into a bitstream. The levels of output bitstream signal txd_out can be inverted by configuring the UART_TXD_INV field.

The receiver contains an RX FIFO (i.e., Rx_FIFO in Figure 45.3-1), which buffers data to be processed. The input bitstream signal rxd_in is transferred to the UART controller, and its level can be inverted by configuring UART_RXD_INV field. Baudrate_Detect measures the baud rate of input bitstream signal rxd_in by detecting its minimum pulse width. Start_Detect detects the start bit in a data frame. If the start bit is detected, Rx_FSM stores data bits in the data frame into Rx_FIFO by Rx_FIFO_Ctrl. Software can read data from Rx_FIFO via the APB bus, or receive data using GDMA.

HW_Flow_Ctrl controls rxd_in and txd_out data flows by standard UART RTS and CTS flow control signals (rtsn_out and ctsn_in). SW_Flow_Ctrl controls data flows by adding special characters to outgoing data and detecting special characters in incoming data. When a UART controller is in Light-sleep mode (see Chapter 15 *Low-Power Management* for more details), a wake_up signal can be generated in four ways and sent to RTC, which then wakes up the ESP32-P4 chip. For more information about wakeup, please refer to Section 45.4.8.

45.4 Functional Description

45.4.1 Clock and Reset

UART controllers are asynchronous. Their register configuration module works in the APB_CLK domain. TX FIFO and RX FIFO work across the AHB_CLK and UART_FCLK domains. The UART RAM control unit works in the UART_FCLK domain. The UART transmission and reception control module works in the UART_SCLK domain, i.e., UART Core's clock domain.

When the frequency of the UART_SCLK is higher than the frequency needed to generate the baud rate, the UART Core can be clocked at a lower frequency by the divider, in order to reduce power consumption. Usually, the UART Core's clock frequency is lower than the APB_CLK's frequency, and can be divided by the largest divisor when higher than the frequency needed to generate the baud rate. The frequency of the UART Core's clock can also be at most twice higher than the APB_CLK. The clock for the UART transmitter and the UART receiver can be controlled independently. To enable the clock for the UART transmitter, UART_TX_SCLK_EN shall be set; to enable the clock for the UART receiver, UART_RX_SCLK_EN shall be set.

To ensure that the configured register values are synchronized from APB_CLK domain to the UART_SCLK domain, please follow the procedures in Section45.5.

To reset the whole UART, please:

- Enable the UART Core's clock by setting HP_SYS_CLKRST_UART*n*_CLK_EN to 1.
- Write 1 to HP_SYS_CLKRST_RST_EN_UART*n*_CORE.
- Clear HP_SYS_CLKRST_RST_EN_UART*n*_CORE to 0.

45.4.2 UART FIFO

The transmitter and the receiver on the UART controller each use a 128 x 8-bit RAM, and access their respective RAM through a separate 4 x 8-bit asynchronous FIFO interface. The RAM and asynchronous FIFO interface for the transmitter and the receiver are independent and cannot be shared.

UART Tx_FIFO is reset by setting UART_TXFIFO_RST. UART Rx_FIFO is reset by setting UART_RXFIFO_RST.

Data to be sent is written to TX FIFO via the APB bus or using GDMA, read automatically, and converted from a frame into a bitstream by hardware Tx_FSM. Data received is converted from a bitstream into a frame by hardware Rx_FSM, written into RX FIFO, and then stored into RAM via the APB bus or using GDMA. The five UART controllers share one GDMA channel.

The empty signal threshold for Tx_FIFO is configured by setting UART_TXFIFO_EMPTY_THRHD. When data stored in Tx_FIFO is less than UART_TXFIFO_EMPTY_THRHD, a UART_TXFIFO_EMPTY_INT interrupt is generated. The full signal threshold for Rx_FIFO is configured by setting UART_RXFIFO_FULL_THRHD. When data stored in Rx_FIFO is greater than or equal to UART_RXFIFO_FULL_THRHD, a UART_RXFIFO_FULL_INT interrupt is generated. In addition, when Rx_FIFO receives more data than its capacity, a UART_RXFIFO_OVF_INT interrupt is generated.

UART*n* can access FIFO via register UART_FIFO_REG. You can put data into TX FIFO by writing UART_RXFIFO_RD_BYTE, and get data in RX FIFO by reading UART_RXFIFO_RD_BYTE.

45.4.3 Baud Rate Generation and Detection

45.4.3.1 Baud Rate Generation

Before a UART controller sends or receives data, the baud rate should be configured by setting corresponding registers. The baud rate generator of a UART controller functions by dividing the input clock source. It can divide the clock source by a fractional amount. The divisor is configured by UART_CLKDIV_SYNC_REG: UART_CLKDIV for the integral part, and UART_CLKDIV_FRAG for the fractional part. When using the 80 MHz input clock, the UART controller supports a maximum baud rate of 5 MBaud.

The divisor of the baud rate divider is equal to

$$UART_CLKDIV + \frac{UART_CLKDIV_FRAG}{16}$$

meaning that the final baud rate is equal to

$$\frac{INPUT_FREQ}{UART_CLKDIV + \frac{UART_CLKDIV_FRAG}{16}}$$

where INPUT_FREQ is the frequency of UART Core's source clock. For example, if UART_CLKDIV = 694 and UART_CLKDIV_FRAG = 7, then the divisor value is

$$694 + \frac{7}{16} = 694.4375$$

When UART_CLKDIV_FRAG is 0, the baud rate generator is an integer clock divider where an output pulse is generated every UART_CLKDIV input pulses.

When UART_CLKDIV_FRAG is not 0, the divider is fractional and the output baud rate clock pulses are not strictly uniform. As shown in Figure 45.4-1, for every 16 output pulses, the generator divides either

(UART_CLKDIV + 1) input pulses or UART_CLKDIV input pulses per output pulse. A total of UART_CLKDIV_FRAG output pulses are generated by dividing (UART_CLKDIV + 1) input pulses, and the remaining (16 - UART_CLKDIV_FRAG) output pulses are generated by dividing UART_CLKDIV input pulses.

The output pulses are interleaved as shown in Figure 45.4-1 below, to make the output timing more uniform:

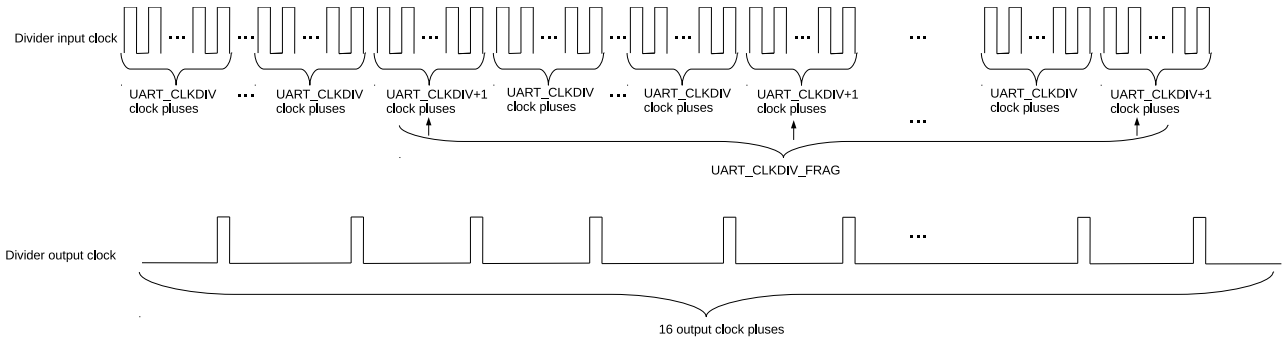


Figure 45.4-1. UART Controllers Division

To support IrDA (see Section 45.4.7 for details), the fractional clock divider for IrDA data transmission generates clock signals divided by 16 × UART_CLKDIV_SYNC_REG. This divider works similarly as the one elaborated above: it takes UART_CLKDIV/16 as the integer value and the lowest four bits of UART_CLKDIV as the fractional value.

45.4.3.2 Baud Rate Detection

Automatic baud rate detection (Autobaud) on UARTs is enabled by setting UART_AUTOBAUD_EN. The Baudrate_Detect module shown in Figure 45.4-2 filters any noise whose pulse width is shorter than UART_GLITCH_FILT.

Before communication starts, the transmitter could send random data to the receiver for baud rate detection. UART_LOWPULSE_MIN_CNT stores the minimum low pulse width, UART_HIGHPULSE_MIN_CNT stores the minimum high pulse width, UART_POSEDGE_MIN_CNT stores the minimum pulse width between two positive edges, and UART_NEGEDGE_MIN_CNT stores the minimum pulse width between two negative edges. These four fields are read by software to determine the transmitter's baud rate.

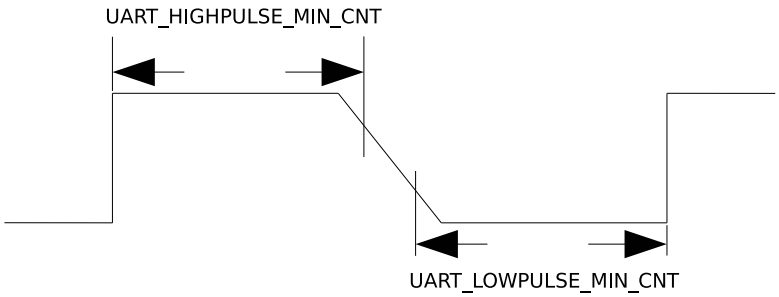


Figure 45.4-2. The Timing Diagram of Weak UART Signals Along Negative Edges

The baud rate can be determined in the following three ways:

1. Normally, to avoid sampling erroneous data along positive or negative edges in a metastable state, which results in the inaccuracy of UART_LOWPULSE_MIN_CNT or UART_HIGHPULSE_MIN_CNT, use a

weighted average of these two values to eliminate errors for 1-bit pulses. In this case, the baud rate is calculated as follows:

$$B_{uart} = \frac{f_{clk}}{(UART_LOWPULSE_MIN_CNT + UART_HIGHPULSE_MIN_CNT + 2)/2}$$

2. If UART signals are weak along negative edges as shown in Figure 45.4-2, which leads to an inaccurate average of UART_LOWPULSE_MIN_CNT and UART_HIGHPULSE_MIN_CNT, use UART_POSEDGE_MIN_CNT to determine the transmitter's baud rate as follows:

$$B_{uart} = \frac{f_{clk}}{(UART_POSEDGE_MIN_CNT + 1)/2}$$

3. If UART signals are weak along positive edges, use UART_NEGEDGE_MIN_CNT to determine the transmitter's baud rate as follows:

$$B_{uart} = \frac{f_{clk}}{(UART_NEGEDGE_MIN_CNT + 1)/2}$$

45.4.4 UART Data Frame

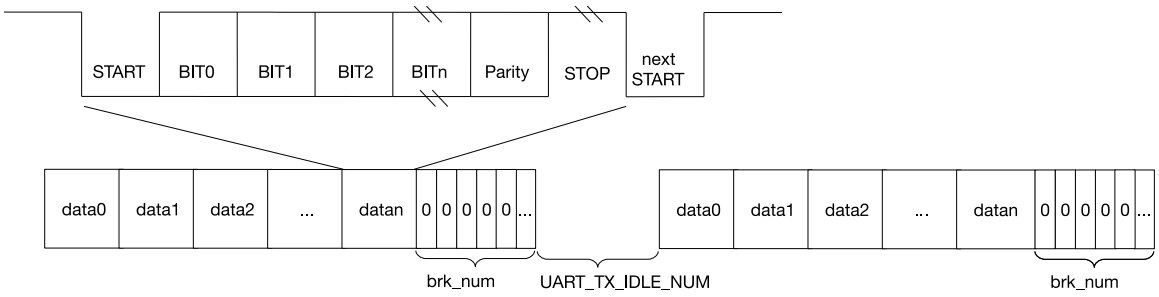


Figure 45.4-3. Structure of UART Data Frame

Figure 45.4-3 shows the basic structure of a data frame. A frame starts with one start bit, and ends with stop bits which can be 1, 1.5, or 2 bit-long, configured by UART_STOP_BIT_NUM (in RS485 mode turnaround delay may be added. See details in Section 45.4.6.2). The start bit is logical low, whereas stop bits are logical high.

The actual data length can be anywhere between 5 ~ 8 bit, configured by UART_BIT_NUM. When UART_PARITY_EN is set, a parity bit is added after data bits. UART_PARITY is used to choose even parity or odd parity. When the receiver detects a parity bit error in the data received, a UART_PARITY_ERR_INT interrupt is generated, and the data received will still be stored into RX FIFO. When the receiver detects a data frame error, a UART_FRM_ERR_INT interrupt is generated, and the data received by default is stored into RX FIFO.

If all data in Tx_FIFO has been sent, a UART_TX_DONE_INT interrupt is generated. After this, if the UART_TXD_BRK bit is set, then the transmitter will enter the break condition and send several NULL characters in which the TX data line is logical low. The number of NULL characters is configured by UART_TX_BRK_NUM. Once the transmitter has sent all NULL characters, a UART_TX_BRK_DONE_INT interrupt is generated. The minimum interval between data frames can be configured using UART_TX_IDLE_NUM. If the transmitter stays idle for UART_TX_IDLE_NUM or more time, a UART_TX_BRK_IDLE_DONE_INT interrupt is generated.

The receiver can also detect the Break conditions when the RX data line detects any low logical level for one NULL character transmission, and a UART_BRK_DET_INT interrupt will be triggered to detect when a break condition has been completed.

The receiver can detect the current bus state through the timeout interrupt UART_RXFIFO_TOUT_INT. The UART_RXFIFO_TOUT_INT interrupt will be triggered when the bus is in the idle state for more than UART_RX_TOUT_THRHD bit time on current baud rate after the receiver has received at least one byte. You can use this interrupt to detect whether all the data from the transmitter has been sent.

45.4.5 AT_CMD Character Structure

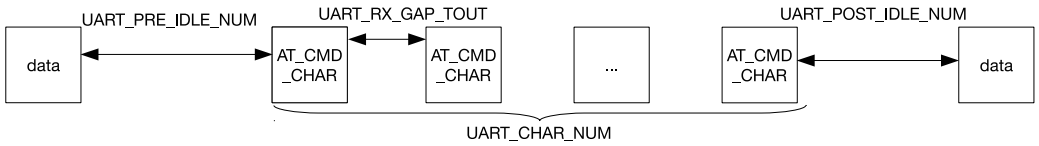


Figure 45.4-4. AT_CMD Character Structure

Figure 45.4-4 is the structure of a special character AT_CMD. If the receiver constantly receives AT_CMD_CHAR and the following conditions are met, a UART_AT_CMD_CHAR_DET_INT interrupt is generated.

- The interval between the first AT_CMD_CHAR and the last non-AT_CMD_CHAR character is at least UART_PRE_IDLE_NUM cycles.
- The interval between two AT_CMD_CHAR characters is less than UART_RX_GAP_TOUT in the unit of baud rate cycles.
- The number of AT_CMD_CHAR characters is equal to or greater than UART_CHAR_NUM.
- The interval between the last AT_CMD_CHAR character and next non-AT_CMD_CHAR character is at least UART_POST_IDLE_NUM cycles.

Note: Given that the interval between AT_CMD_CHAR characters is less than UART_RX_GAP_TOUT in the unit of baud rate cycles, the APB_CLK frequency is suggested not to be lower than 8 MHz.

45.4.6 RS485

The five regular UART controllers support RS485 communication mode. In this mode differential signals are used to transmit data, so it can communicate over longer distances at higher bit rates than RS232. RS485 has two-wire half-duplex and four-wire full-duplex options. UART controllers support two-wire half-duplex transmission and bus snooping.

45.4.6.1 Driver Control

As shown in Figure 45.4-5, in a two-wire multidrop network, an external RS485 transceiver is needed for differential to single-ended conversion or the other way around. An RS485 transceiver contains a driver and a receiver. When a UART controller is not in transmitter mode, the connection to the differential line can be broken by disabling the driver. When DE is 1, the driver is enabled; when DE is 0, the driver is disabled.

The UART receiver converts differential signals to single-ended signals via an external receiver. RE is the enable control signal for the receiver. When RE is 0, the receiver is enabled; when RE is 1, the receiver is disabled. If RE is configured as 0, the UART controller is allowed to snoop data on the bus, including the data sent by itself.

DE can be controlled by either software or hardware. To reduce the cost of software, in our design DE is controlled by hardware. As shown in Figure 45.4-5, DE is connected to dtrn_out of UART (please refer to Section 45.4.9.1 for more details).

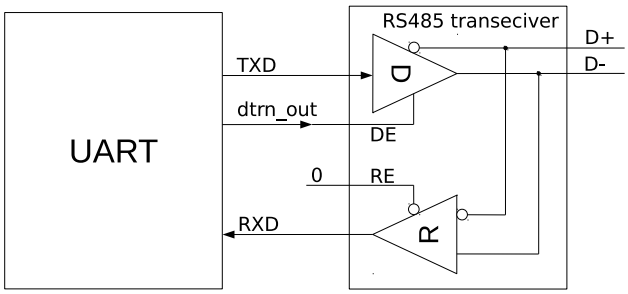


Figure 45.4-5. Driver Control Diagram in RS485 Mode

45.4.6.2 Turnaround Delay

By default, the UART controllers work in receiver mode. When a UART controller is switched from transmitter mode to receiver mode, the RS485 protocol requires a turnaround delay of one cycle after the stop bit. The UART transmitter supports adding a turnaround delay of one cycle before the start bit or after the stop bit. When UART_DL0_EN is set, a turnaround delay of one cycle is added before the start bit; when UART_DL1_EN is set, a turnaround delay of one cycle is added after the stop bit.

45.4.6.3 Bus Snooping

In a two-wire multidrop network, UART controllers support bus snooping if RE of the external RS485 transceiver is 0. By default, a UART controller is not allowed to transmit and receive data simultaneously. If UART_RS485TX_RX_EN is set and the external RS485 transceiver is configured as in Figure 45.4-5, a UART controller may receive data in transmitter mode and snoop the bus. If UART_RS485RXBY_TX_EN is set, a UART controller may transmit data in receiver mode.

The two UART controllers can snoop the data sent by themselves. In transmitter mode, when a UART controller monitors a collision between the data sent and the data received, a UART_RS485_CLASH_INT is generated; when a UART controller monitors a data frame error, a UART_RS485_FRM_ERR_INT interrupt is generated; when a UART controller monitors a parity bit error, a UART_RS485_PARITY_ERR_INT is generated.

45.4.7 IrDA

IrDA protocol consists of three layers, namely the physical layer, the link access protocol, and the link management protocol. The UART controllers implement IrDA's physical layer. In IrDA encoding, a UART controller supports data rates up to 115.2 kbit/s (SIR, or serial infrared mode). As shown in Figure 45.4-6, the IrDA encoder converts a non-return to zero code (NRZ) signal to a return to zero inverted code (RZI) signal and sends it to the external driver and infrared LED. This encoder uses modulated signals whose pulse width is 3/16 bits to indicate logic "0", and low levels to indicate logic "1". The IrDA decoder receives signals from the infrared receiver and converts them to NRZ signals. In most cases, the receiver is high when it is idle, and the parity bit of the encoder output is opposite to that of the decoder input. If a low pulse is detected, it indicates that a start bit has been received.

When IrDA function is enabled, one bit is divided into 16 clock cycles. If the bit to be sent is zero, then the 9th, 10th, and 11th clock cycle are high.

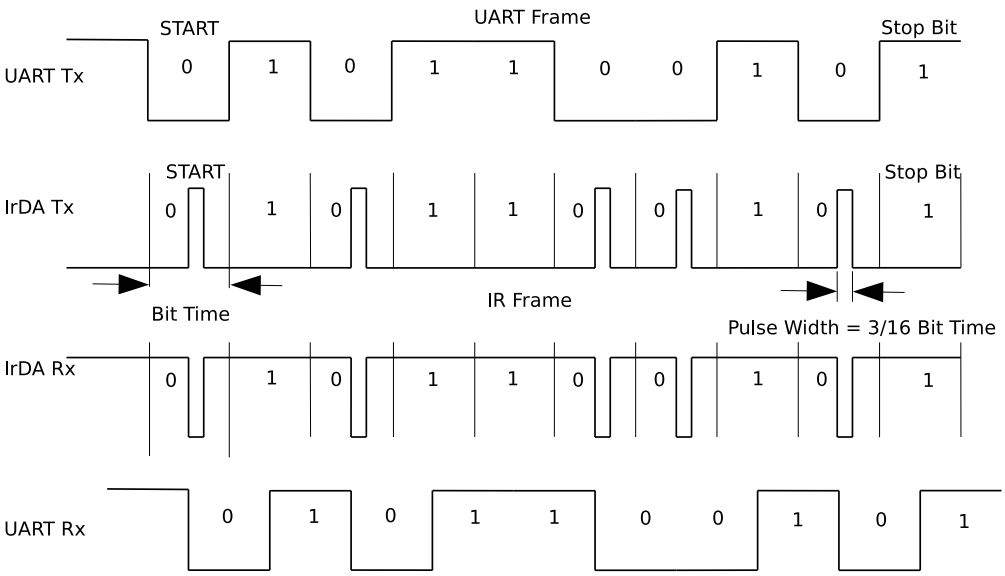


Figure 45.4-6. The Timing Diagram of Encoding and Decoding in SIR mode

The IrDA transceiver is half-duplex, meaning that it cannot send and receive data simultaneously. As shown in Figure 45.4-7, IrDA function is enabled by setting UART_IRDA_EN. When UART_IRDA_TX_EN is set to 1, the IrDA transceiver is enabled to send data and not allowed to receive data; when UART_IRDA_TX_EN is reset to 0, the IrDA transceiver is enabled to receive data and not allowed to send data.

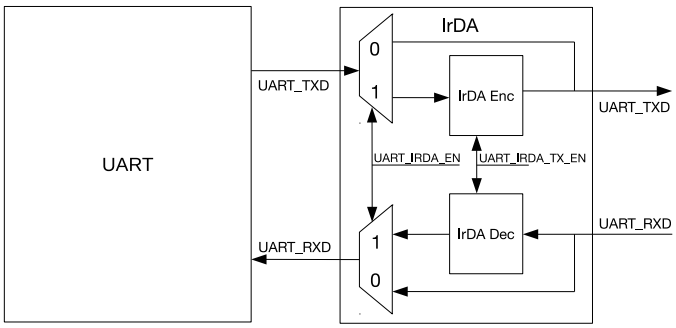


Figure 45.4-7. IrDA Encoding and Decoding Diagram

45.4.8 Wakeup

UART can be set as wakeup source. When a UART controller is in Light-sleep mode, a wake_up signal can be generated in four ways and be sent to the RTC module, which then wakes up ESP32-P4.

- UART_WK_MODE_SEL = 0: When all the clocks are disabled, the chip can be woken up by reverting RXD for multiple cycles until the number of positive edges is greater than or equal to UART_ACTIVE_THRESHOLD + 3.
- UART_WK_MODE_SEL = 1: UART Core keeps working, so the UART receiver can still receive data and store the received data in RX FIFO. When the number of data bytes in RX FIFO is greater than UART_RX_WAKE_UP_THRHD, the chip can be woken up from the Light-sleep mode.
- UART_WK_MODE_SEL = 2: When the UART receiver detects a start bit, the chip will be woken up.

- UART_WK_MODE_SEL = 3: When the UART receiver receives a specific character sequence, the chip will be woken up. The wakeup characters can be defined by configuring UART_WK_CHAR0, UART_WK_CHAR1, UART_WK_CHAR2, UART_WK_CHAR3, and UART_WK_CHAR4. These characters can be formed into different character sequences by configuring UART_CHAR_NUM and UART_WK_CHAR_MASK, as shown in Table 45.4-1. Once the sequence is detected, the chip will be woken up. For the last configuration in Table 45.4-1, UART will detect for CHAR0 ~ CHAR4 in order.

Table 45.4-1. UART_CHAR_WAKEUP Mode Configuration

UART_CHAR_NAME	UART_WP_CHAR_MASK	Character Sequence
1	0xF	CHAR4
2	0x7	CHAR3/CHAR4
3	0x3	CHAR2/CHAR3/CHAR4
4	0x1	CHAR1/CHAR2/CHAR3/CHAR4
5	0x0	CHAR0/CHAR1/CHAR2/CHAR3/CHAR4

After the chip is woken up by UART, it is necessary to clear the wake_up signal by transmitting data to UART in Active mode or resetting the whole UART, otherwise the number of rising edges required for the next wakeup will be reduced.

45.4.9 Flow Control

UART controllers have two ways to control data flow, namely hardware flow control and software flow control. Hardware flow control is achieved using output signal rtsn_out and input signal ctsn_in. Software flow control is achieved by inserting special characters in the data flow sent and detecting special characters in the data flow received.

45.4.9.1 Hardware Flow Control

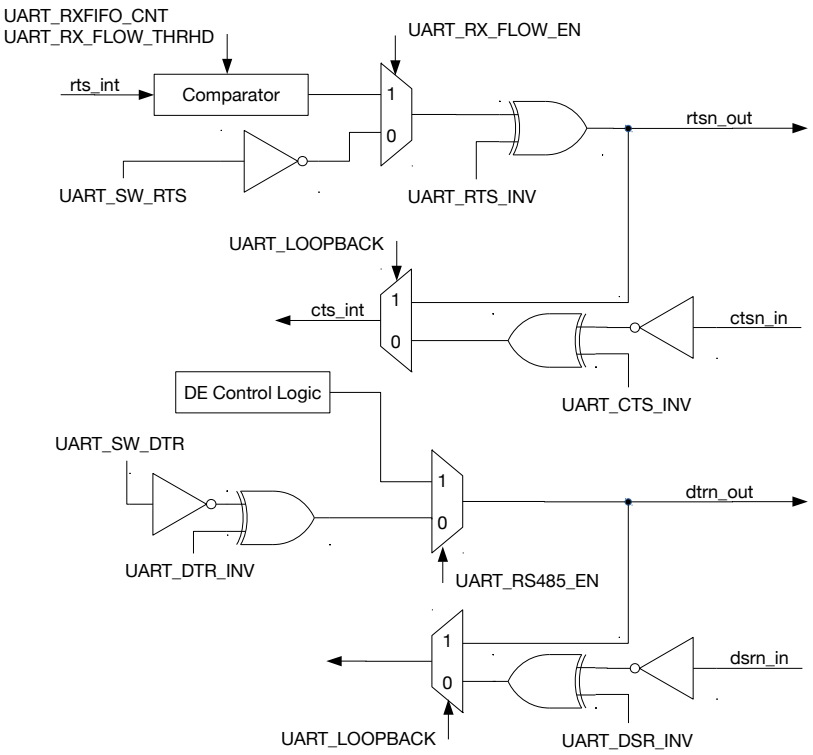


Figure 45.4-8. Hardware Flow Control Diagram

Figure 45.4-8 shows the hardware flow control of a UART controller. Hardware flow control uses output signal rtsn_out and input signal dsrn_in. Figure 45.4-9 illustrates how these signals are connected between UART on ESP32-P4 (hereinafter referred to as IUO) and the external UART (hereinafter referred to as EUO).

When rtsn_out of IUO is low, EUO is allowed to send data. When rtsn_out of IUO is high, EUO is notified to stop sending data until rtsn_out of IUO returns to low. The output signal rtsn_out can be controlled in two ways.

- Software control: Enter this mode by clearing UART_RX_FLOW_EN to 0. In this mode, the level of rtsn_out is changed by configuring UART_SW_RTS.
- Hardware control: Enter this mode by setting UART_RX_FLOW_EN to 1. In this mode, rtsn_out is pulled high when data in Rx_FIFO exceeds UART_RX_FLOW_THRHD.

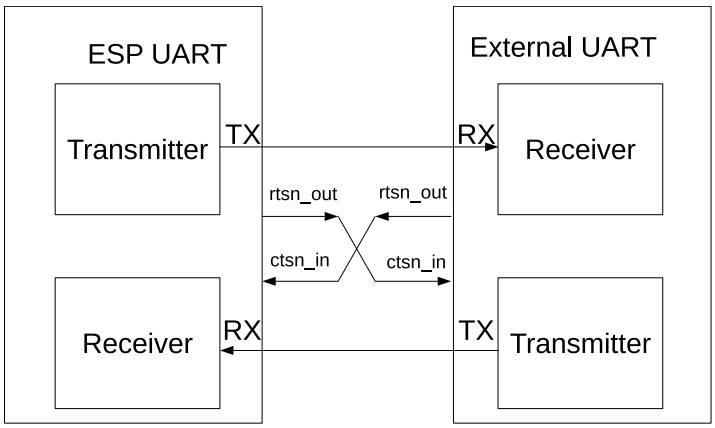


Figure 45.4-9. Connection between Hardware Flow Control Signals

When ctsn_in of IUO is low, IUO is allowed to send data; when ctsn_in is high, IUO is not allowed to send data. When IUO detects an edge change of ctsn_in, a UART_CTS_CHG_INT interrupt is generated.

If dtrn_out of IUO is high, it indicates that IUO is ready to transmit data. dtrn_out is generated by configuring the UART_SW_DTR field. When the IUO transmitter detects an edge change of dsrn_in, a UART_DSR_CHG_INT interrupt is generated. After this interrupt is detected, software can obtain the level of input signal dsrn_in by reading UART_DSRN. If dsrn_in is high, it indicates that EUO is ready to transmit data.

In a two-wire RS485 multidrop network enabled by setting UART_RS485_EN, dtrn_out is generated by hardware and used for transmit/receive turnaround. When data transmission starts, dtrn_out is pulled high and the external driver is enabled; when data transmission completes, dtrn_out is pulled low and the external driver is disabled. Please note that when there is a turnaround delay of one cycle added after the stop bit, dtrn_out is pulled low after the delay.

UART loopback test is enabled by setting UART_LOOPBACK. In the test, UART output signal txd_out is connected to its input signal rxd_in, rtsn_out is connected to ctsn_in, and dtrn_out is connected to dsrn_out. If the data sent matches the data received, it indicates that UART controllers are working properly.

45.4.9.2 Software Flow Control

Instead of CTS/RTS lines, software flow control uses XON/XOFF characters to start or stop data transmission. Such flow control is enabled by setting UART_SW_FLOW_CON_EN to 1.

When using software flow control, hardware automatically detects if there are XON/XOFF characters in the data flow received, and generate a UART_SW_XOFF_INT or a UART_SW_XON_INT interrupt accordingly. If an XOFF character is detected, the transmitter stops data transmission once the current byte has been transmitted; if an XON character is detected, the transmitter starts data transmission. In addition, software can force the transmitter to stop sending data by setting UART_FORCE_XOFF, or to start sending data by setting UART_FORCE_XON.

Software determines whether to insert flow control characters based on the remaining room in RX FIFO. When UART_SEND_XOFF is set, the transmitter sends an XOFF character configured by UART_XOFF_CHAR after the current byte in transmission; when UART_SEND_XON is set, the transmitter sends an XON character configured by UART_XON_CHAR after the current byte in transmission. If the RX FIFO of a UART controller stores more data than UART_XOFF_THRESHOLD, UART_SEND_XOFF is set by hardware. As a result, the transmitter sends an XOFF character configured by UART_XOFF_CHAR after the current byte in transmission. If the RX FIFO of a

UART controller stores less data than UART_XON_THRESHOLD, UART_SEND_XON is set by hardware. As a result, the transmitter sends an XON character configured by UART_XON_CHAR after the current byte in transmission.

In full-duplex mode, when the UART receiver receives an XOFF character, the UART transmitter is not allowed to send any data including XOFF even if the UART receiver receives more data than its threshold. To avoid deadlocks in software flow control or overflow caused thereby, you can set UART_XON_XOFF_STILL_SEND. In this way, the UART transmitter can still send an XOFF character when it is not allowed to send any data.

45.4.10 GDMA Mode

The five UART controllers on ESP32-P4 share one TX/RX GDMA (General Direct Memory Access) channel via UHCI (Universal Host Controller Interface). In GDMA mode, UART controllers support the decoding and encoding of HCI data packets. The UHCI_UART_SEL field determines which UART controller occupies the GDMA TX/RX channel.

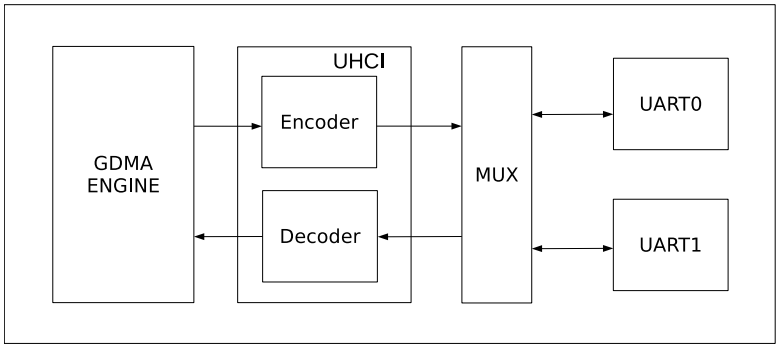


Figure 45.4-10. Data Transfer in GDMA Mode

Figure 45.4-10 shows how data is transferred using GDMA. Before GDMA receives data, software prepares an inlink. GDMA_INLINK_ADDR_CH*n* points to the first receive descriptor in the inlink. After GDMA_INLINK_START_CH*n* is set, UHCI sends data that UART has received to the decoder. The decoded data is then stored into the RAM pointed by the inlink under the control of GDMA.

Before GDMA sends data, software prepares an outlink and data to be sent. GDMA_OUTLINK_ADDR_CH*n* points to the first transmit descriptor in the outlink. After GDMA_OUTLINK_START_CH*n* is set, GDMA reads data from the RAM pointed by outlink. The data is then encoded by the encoder, and sent sequentially by the UART transmitter.

HCI data packets have separators at the beginning and the end, with data bits in the middle (separators + data bits + separators). The encoder inserts separators in front of and after data bits, and replaces data bits identical to separators with special characters. The decoder removes separators in front of and after data bits, and replaces special characters with separators. There can be more than one continuous separator at the beginning and the end of a data packet. The separator is configured by UHCI_SEPER_CHAR, 0xC0 by default. The special character is configured by UHCI_ESC_SEQ0_CHAR0 (0xDB by default) and UHCI_ESC_SEQ0_CHAR1 (0xDD by default). When all data has been sent, a GDMA_OUT_TOTAL_EOF_CH*n*_INT interrupt is generated. When all data has been received, a GDMA_IN_SUC_EOF_CH*n*_INT is generated.

45.4.11 Interrupts

ESP32-P4's UART*n* and UHCI can generate the following interrupt signals that will be sent to the Interrupt Matrix.

- UART*n*_INTR
- UHCI_INTR

There are several internal interrupt sources from UART*n* and UHCI that can generate the above interrupt signals.

UART*n* interrupt sources are listed as follows:

- UART_AT_CMD_CHAR_DET_INT: Triggered when the receiver detects an AT_CMD character.
- UART_RS485_CLASH_INT: Triggered when a collision is detected between the transmitter and the receiver in RS485 mode.
- UART_RS485_FRM_ERR_INT: Triggered when an error is detected in the data frame sent by the transmitter in RS485 mode.
- UART_RS485_PARITY_ERR_INT: Triggered when an error is detected in the parity bit sent by the transmitter in RS485 mode.
- UART_TX_DONE_INT: Triggered when all data in the transmitter's TX FIFO has been sent.
- UART_TX_BRK_IDLE_DONE_INT: Triggered when the transmitter stays idle for the minimum interval (threshold) after sending the last data bit.
- UART_TX_BRK_DONE_INT: Triggered when the transmitter has sent all NULL characters after all data in TX FIFO had been sent.
- UART_GLITCH_DET_INT: Triggered when the receiver detects a glitch in the middle of the start bit.
- UART_SW_XOFF_INT: Triggered when UART_SW_FLOW_CON_EN is set and the receiver receives a XOFF character.
- UART_SW_XON_INT: Triggered when UART_SW_FLOW_CON_EN is set and the receiver receives a XON character.
- UART_RXFIFO_TOUT_INT: Triggered when the receiver has received at least one byte, and the bus remains idle for UART_RX_TOUT_THRHD bit time.
- UART_BRK_DET_INT: Triggered when the receiver detects a NULL character (i.e., logic 0 for one NULL character transmission) after stop bits.
- UART_CTS_CHG_INT: Triggered when the receiver detects an edge change of CTSn signals.
- UART_DSR_CHG_INT: Triggered when the receiver detects an edge change of DSRn signals.
- UART_RXFIFO_OVF_INT: Triggered when the amount of data received by the receiver exceeds the storage capacity of the FIFO.
- UART_FRM_ERR_INT: Triggered when the receiver detects a data frame error.
- UART_PARITY_ERR_INT: Triggered when the receiver detects a parity error.
- UART_TXFIFO_EMPTY_INT: Triggered when TX FIFO stores less data than what UART_TXFIFO_EMPTY_THRHD specifies.

- UART_RXFIFO_FULL_INT: Triggered when the receiver receives more data than what UART_RXFIFO_FULL_THRHD specifies.
- UART_WAKEUP_INT: Triggered when UART is woken up.

UHCI interrupt sources are listed as follows:

- UHCI_APP_CTRL1_INT: Triggered when software sets UHCI_APP_CTRL1_INT_RAW.
- UHCI_APP_CTRL0_INT: Triggered when software sets UHCI_APP_CTRL0_INT_RAW.
- UHCI_OUTLINK_EOF_ERR_INT: Triggered when an EOF error is detected in a transmit descriptor.
- UHCI_SEND_A_REG_Q_INT: Triggered when UHCI has sent a series of short packets using always_send.
- UHCI_SEND_S_REG_Q_INT: Triggered when UHCI has sent a series of short packets using single_send.
- UHCI_TX_HUNG_INT: Triggered when UHCI takes too long to read RAM using a GDMA transmit channel.
- UHCI_RX_HUNG_INT: Triggered when UHCI takes too long to receive data using a GDMA receive channel.
- UHCI_TX_START_INT: Triggered when GDMA detects a separator character.
- UHCI_RX_START_INT: Triggered when a separator character has been sent.

Each interrupt source can be configured by a common set of registers that are described in Section *Interrupt Configuration Registers*. The specific registers can be found in Section 45.6 *Register Summary*.

45.5 Programming Procedures

45.5.1 Register Type

All UART registers are in the APB_CLK domain.

UART configuration registers can be classified into two groups. One group of registers are read in the APB_CLK or AHB_CLK domains, so once such registers are configured no extra operations are required. The other group of registers are read in the UART_SCLK domain, and therefore need to implement the clock domain crossing design. Once these registers are configured, the configured values need to be synchronized to the UART Core's clock domain by writing to UART_REG_UPDATE. Once all values have been synchronized, UART_REG_UPDATE will be automatically cleared by hardware. After configuring registers that need synchronization, it is recommended to check whether UART_REG_UPDATE is 0. This is to ensure that register values configured before have already been synchronized.

To distinguish between these two groups of registers easily, all registers that implement the clock domain crossing design have the _SYNC suffix, and are put together in Section 45.6. Those without the _SYNC suffix in Section 45.6 are configuration registers that require no clock domain crossing.

45.5.2 Detailed Steps

Figure 45.5-1 illustrates the process to program UART controllers, namely initialize UART, configure registers, enable the UART transmitter or receiver, and finish data transmission.

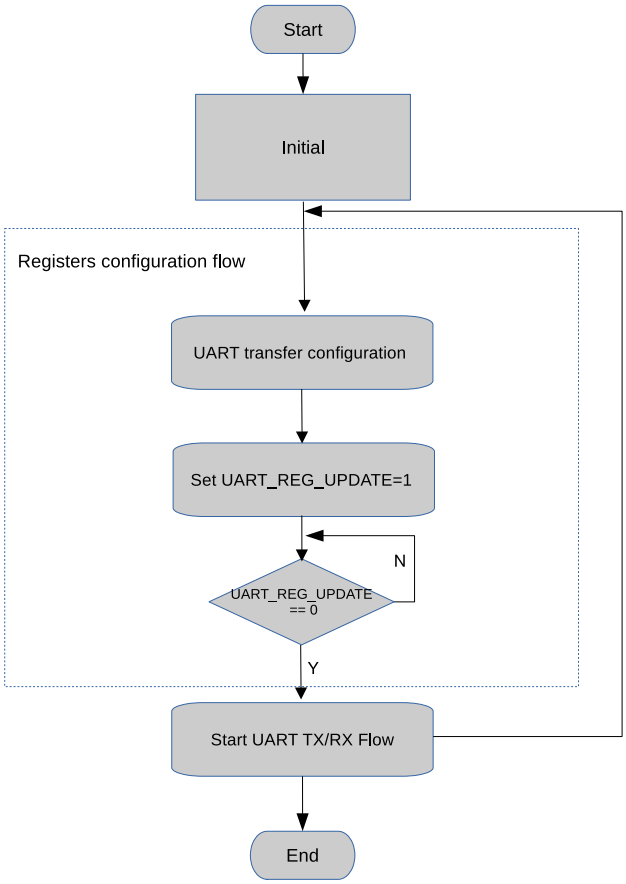


Figure 45.5-1. UART Programming Procedures

45.5.2.1 Initializing UART*n*

To initialize UART*n*:

- Write 1 to HP_SYS_CLKRST_RST_EN_UART*n*_APB.
- Clear HP_SYS_CLKRST_RST_EN_UART*n*_APB to 0.
- Write 1 to HP_SYS_CLKRST_RST_EN_UART*n*_CORE.
- Clear HP_SYS_CLKRST_RST_EN_UART*n*_CORE to 0.

45.5.2.2 Configuring UART*n* Communication

To configure UART*n* communication:

- Wait for UART_REG_UPDATE to become 0, which indicates the completion of the last synchronization.
- Select the clock source via HP_SYS_CLKRST_UART*n*_CLK_SRC_SEL.
- Configure divisor of the divider via HP_SYS_CLKRST_UART*n*_SCLK_DIV_NUM, HP_SYS_CLKRST_UART*n*_SCLK_DIV_DENOMINATOR, and

HP_SYS_CLKRST_UART*n*_SCLK_DIV_NUMERATOR.

- Configure the baud rate for transmission via UART_CLKDIV and UART_CLKDIV_FRAG.
- Configure data length via UART_BIT_NUM.
- Configure odd or even parity check via UART_PARITY_EN and UART_PARITY.
- Optional steps depending on application ...
- Synchronize the configured values to the Core Clock domain by writing 1 to UART_REG_UPDATE.

45.5.2.3 Enabling UART*n*

To enable UART*n* transmitter:

- Configure TX FIFO's empty threshold via UART_TXFIFO_EMPTY_THRHD.
- Disable UART_TXFIFO_EMPTY_INT interrupt by clearing UART_TXFIFO_EMPTY_INT_ENA.
- Write data to be sent to UART_RXFIFO_RD_BYTE.
- Clear UART_TXFIFO_EMPTY_INT interrupt by setting UART_TXFIFO_EMPTY_INT_CLR.
- Enable UART_TXFIFO_EMPTY_INT interrupt by setting UART_TXFIFO_EMPTY_INT_ENA.
- Check UART_TXFIFO_EMPTY_INT_ST and wait for the completion of data transmission.

To enable UART*n* receiver:

- Configure RX FIFO's full threshold via UART_RXFIFO_FULL_THRHD.
- Enable UART_RXFIFO_FULL_INT interrupt by setting UART_RXFIFO_FULL_INT_ENA.
- Check UART_RXFIFO_FULL_INT_ST and wait until the RX FIFO is full.
- Read data from RX FIFO via UART_RXFIFO_RD_BYTE, and obtain the number of bytes received in RX FIFO via UART_RXFIFO_CNT.

45.6 Register Summary

45.6.1 UART Register Summary

The addresses in this section are relative to UART Controller base address provided in Table 9.3-2 in Chapter 9 *System and Memory*.

The abbreviations given in Column **Access** are explained in Section *Access Types for Registers*.

Name	Description	Address	Access
FIFO Configuration			
UART_FIFO_REG	FIFO data register	0x0000	RO
UART_TOUT_CONF_SYNC_REG	UART threshold and allocation configuration	0x0064	R/W
UART Interrupt Register			
UART_INT_RAW_REG	Raw interrupt status	0x0004	R/WTC/SS
UART_INT_ST_REG	Masked interrupt status	0x0008	RO
UART_INT_ENA_REG	Interrupt enable bits	0x000C	R/W
UART_INT_CLR_REG	Interrupt clear bits	0x0010	WT
Configuration Register			
UART_CLKDIV_SYNC_REG	Clock divider configuration	0x0014	R/W
UART_RX_FILT_REG	RX filter configuration	0x0018	R/W
UART_CONF0_SYNC_REG	Configuration register 0	0x0020	R/W
UART_CONF1_REG	Configuration register 1	0x0024	R/W
UART_HWFC_CONF_SYNC_REG	Hardware flow control configuration	0x002C	R/W
UART_SLEEP_CONF0_REG	UART sleep configuration register 0	0x0030	R/W
UART_SLEEP_CONF1_REG	UART sleep configuration register 1	0x0034	R/W
UART_SLEEP_CONF2_REG	UART sleep configuration register 2	0x0038	R/W
UART_SWFC_CONF0_SYNC_REG	Software flow control character configuration	0x003C	varies
UART_SWFC_CONF1_REG	Software flow control character configuration	0x0040	R/W
UART_TXBRK_CONF_SYNC_REG	TX break character configuration	0x0044	R/W
UART_IDLE_CONF_SYNC_REG	Frame end idle time configuration	0x0048	R/W
UART_RS485_CONF_SYNC_REG	RS485 mode configuration	0x004C	R/W
UART_CLK_CONF_REG	UART core clock configuration	0x0088	R/W
UART_REG_UPDATE_REG	UART register configuration update	0x0098	R/W/SC
UART_ID_REG	UART ID register	0x009C	R/W
Status Register			
UART_STATUS_REG	UART status register	0x001C	RO
UART_MEM_TX_STATUS_REG	TX FIFO write and read offset address	0x0068	RO
UART_MEM_RX_STATUS_REG	Rx FIFO write and read offset address	0x006C	RO
UART_FSM_STATUS_REG	UART transmit and receive status	0x0070	RO
UART_AFIFO_STATUS_REG	UART asynchronous FIFO status	0x0090	RO
AT Escape Sequence Selection Configuration			
UART_AT_CMD_PRECNT_SYNC_REG	Pre-sequence timing configuration	0x0050	R/W
UART_AT_CMD_POSTCNT_SYNC_REG	Post-sequence timing configuration	0x0054	R/W
UART_AT_CMD_GAPTOUT_SYNC_REG	Timeout configuration	0x0058	R/W

Name	Description	Address	Access
UART_AT_CMD_CHAR_SYNC_REG	AT escape sequence detection configuration	0x005C	R/W
Autobaud Register			
UART_POSPULSE_REG	Autobaud high pulse register	0x0074	RO
UART_NEGPULSE_REG	Autobaud low pulse register	0x0078	RO
UART_LOWPULSE_REG	Autobaud minimum low pulse duration register	0x007C	RO
UART_HIGHPULSE_REG	Autobaud minimum high pulse duration register	0x0080	RO
UART_RXD_CNT_REG	Autobaud edge change count register	0x0084	RO
Version Register			
UART_DATE_REG	UART version control register	0x008C	R/W

45.6.2 LP UART Register Summary

The addresses in this section are relative to LP UART base address provided in Table 9.3-2 in Chapter 9 *System and Memory*.

The abbreviations given in Column **Access** are explained in Section *Access Types for Registers*.

Name	Description	Address	Access
FIFO Configuration			
LP_UART_FIFO_REG	FIFO data register	0x0000	RO
LP_UART_TOUT_CONF_SYNC_REG	LP UART threshold and allocation configuration	0x0064	R/W
LP UART Interrupt Register			
LP_UART_INT_RAW_REG	Raw interrupt status	0x0004	R/WTC/SS
LP_UART_INT_ST_REG	Masked interrupt status	0x0008	RO
LP_UART_INT_ENA_REG	Interrupt enable bits	0x000C	R/W
LP_UART_INT_CLR_REG	Interrupt clear bits	0x0010	WT
Configuration Register			
LP_UART_CLKDIV_SYNC_REG	Clock divider configuration	0x0014	R/W
LP_UART_RX_FILT_REG	RX filter configuration	0x0018	R/W
LP_UART_CONF0_SYNC_REG	Configuration register 0	0x0020	R/W
LP_UART_CONF1_REG	Configuration register 1	0x0024	R/W
LP_UART_HWFC_CONF_SYNC_REG	Hardware flow control configuration	0x002C	R/W
LP_UART_SLEEP_CONF0_REG	LP UART sleep configuration register 0	0x0030	R/W
LP_UART_SLEEP_CONF1_REG	LP UART sleep configuration register 1	0x0034	R/W
LP_UART_SLEEP_CONF2_REG	LP UART sleep configuration register 2	0x0038	R/W
LP_UART_SWFC_CONF0_SYNC_REG	Software flow control character configuration	0x003C	varies
LP_UART_SWFC_CONF1_REG	Software flow control character configuration	0x0040	R/W
LP_UART_TXBRK_CONF_SYNC_REG	TX break character configuration	0x0044	R/W
LP_UART_IDLE_CONF_SYNC_REG	Frame end idle time configuration	0x0048	R/W
LP_UART_CLK_CONF_REG	LP UART core clock configuration	0x0088	R/W

Name	Description	Address	Access
LP_UART_REG_UPDATE_REG	LP UART register configuration update register	0x0098	R/W/SC
LP_UART_ID_REG	LP UART ID register	0x009C	R/W
Status Register			
LP_UART_STATUS_REG	LP UART status register	0x001C	RO
LP_UART_MEM_TX_STATUS_REG	TX FIFO write and read offset address	0x0068	RO
LP_UART_MEM_RX_STATUS_REG	RX FIFO write and read offset address	0x006C	RO
LP_UART_FSM_STATUS_REG	LP UART transmit and receive status	0x0070	RO
LP_UART_AFIFO_STATUS_REG	LP UART asynchronous FIFO Status	0x0090	RO
AT Escape Sequence Selection Configuration			
LP_UART_AT_CMD_PRECNT_SYNC_REG	Pre-sequence timing configuration	0x0050	R/W
LP_UART_AT_CMD_POSTCNT_SYNC_REG	Post-sequence timing configuration	0x0054	R/W
LP_UART_AT_CMD_GAPTOUT_SYNC_REG	Timeout configuration	0x0058	R/W
LP_UART_AT_CMD_CHAR_SYNC_REG	AT escape sequence detection configuration	0x005C	R/W
Version Register			
LP_UART_DATE_REG	LP UART version register	0x008C	R/W

45.6.3 UHCI Register Summary

The addresses in this section are relative to UHCI base address provided in Table 9.3-2 in Chapter 9 *System and Memory*.

The abbreviations given in Column **Access** are explained in Section *Access Types for Registers*.

Name	Description	Address	Access
Configuration Register			
UHCI_CONF0_REG	UHCI configuration register	0x0000	R/W
UHCI_CONF1_REG	UHCI configuration register	0x0014	varies
UHCI_ESCAPE_CONF_REG	Escape character configuration	0x0020	R/W
UHCI_HUNG_CONF_REG	Timeout configuration	0x0024	R/W
UHCI_ACK_NUM_REG	UHCI ACK number configuration	0x0028	varies
UHCI_QUICK_SENT_REG	UHCI quick send configuration register	0x0030	varies
UHCI_REG_Q0_WORD0_REG	Q0 WORD0 quick send register	0x0034	R/W
UHCI_REG_Q0_WORD1_REG	Q0 WORD1 quick send register	0x0038	R/W
UHCI_REG_Q1_WORD0_REG	Q1 WORD0 quick send register	0x003C	R/W
UHCI_REG_Q1_WORD1_REG	Q1 WORD1 quick send register	0x0040	R/W
UHCI_REG_Q2_WORD0_REG	Q2 WORD0 quick send register	0x0044	R/W
UHCI_REG_Q2_WORD1_REG	Q2 WORD1 quick send register	0x0048	R/W
UHCI_REG_Q3_WORD0_REG	Q3 WORD0 quick send register	0x004C	R/W
UHCI_REG_Q3_WORD1_REG	Q3 WORD1 quick send register	0x0050	R/W
UHCI_REG_Q4_WORD0_REG	Q4 WORD0 quick send register	0x0054	R/W
UHCI_REG_Q4_WORD1_REG	Q4 WORD1 quick send register	0x0058	R/W
UHCI_REG_Q5_WORD0_REG	Q5 WORD0 quick send register	0x005C	R/W

Name	Description	Address	Access
UHCI_REG_Q5_WORD1_REG	Q5 WORD1 quick send register	0x0060	R/W
UHCI_REG_Q6_WORD0_REG	Q6 WORD0 quick send register	0x0064	R/W
UHCI_REG_Q6_WORD1_REG	Q6 WORD1 quick register	0x0068	R/W
UHCI_ESC_CONF0_REG	Escape sequence configuration register 0	0x006C	R/W
UHCI_ESC_CONF1_REG	Escape sequence configuration register 1	0x0070	R/W
UHCI_ESC_CONF2_REG	Escape sequence configuration register 2	0x0074	R/W
UHCI_ESC_CONF3_REG	Escape sequence configuration register 3	0x0078	R/W
UHCI_PKT_THRES_REG	Configuration register for packet length	0x007C	R/W
UHCI Interrupt Register			
UHCI_INT_RAW_REG	Raw interrupt status	0x0004	varies
UHCI_INT_ST_REG	Masked interrupt status	0x0008	RO
UHCI_INT_ENA_REG	Interrupt enable bits	0x000C	R/W
UHCI_INT_CLR_REG	Interrupt clear bits	0x0010	WT
UHCI Status Register			
UHCI_STATE0_REG	UHCI receive status	0x0018	RO
UHCI_STATE1_REG	UHCI transmit status	0x001C	RO
UHCI_RX_HEAD_REG	UHCI packet header register	0x002C	RO
Version Register			
UHCI_DATE_REG	UHCI version control register	0x0080	R/W

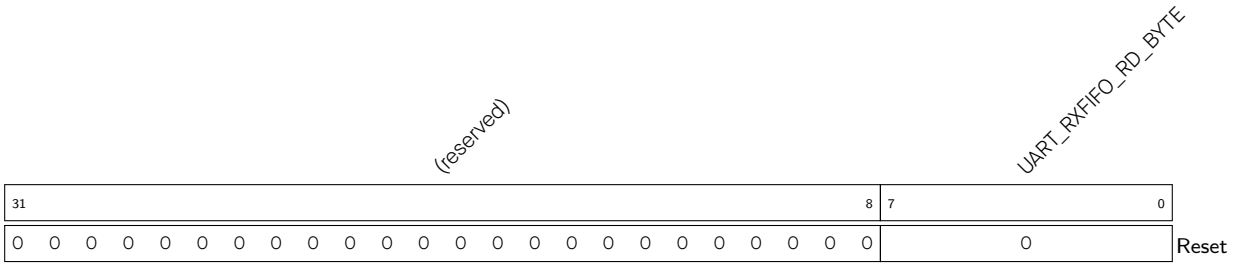
45.7 Registers

45.7.1 UART Registers

The addresses in this section are relative to UART Controller base address provided in Table 9.3-2 in Chapter 9 *System and Memory*.

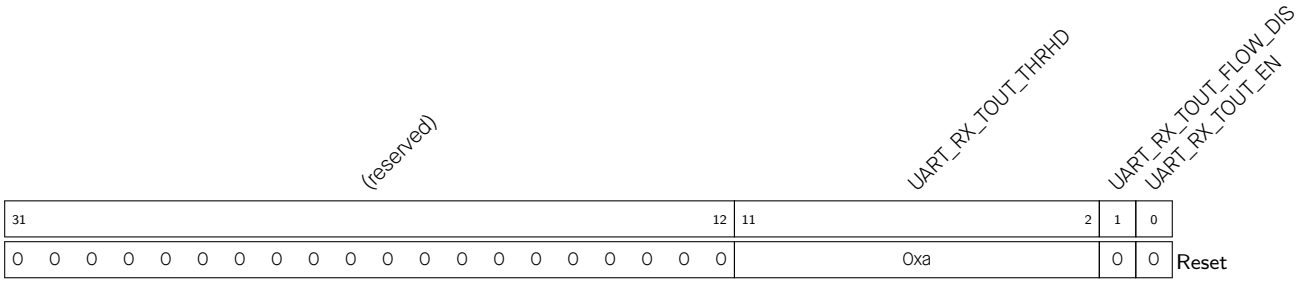
For how to program reserved fields, please refer to Section *Programming Reserved Register Field*.

Register 45.1. UART_FIFO_REG (0x0000)



UART_RXFIFO_RD_BYTE Represents the data UART *n* read from FIFO.
Measurement unit: byte. (RO)

Register 45.2. UART_TOUT_CONF_SYNC_REG (0x0064)

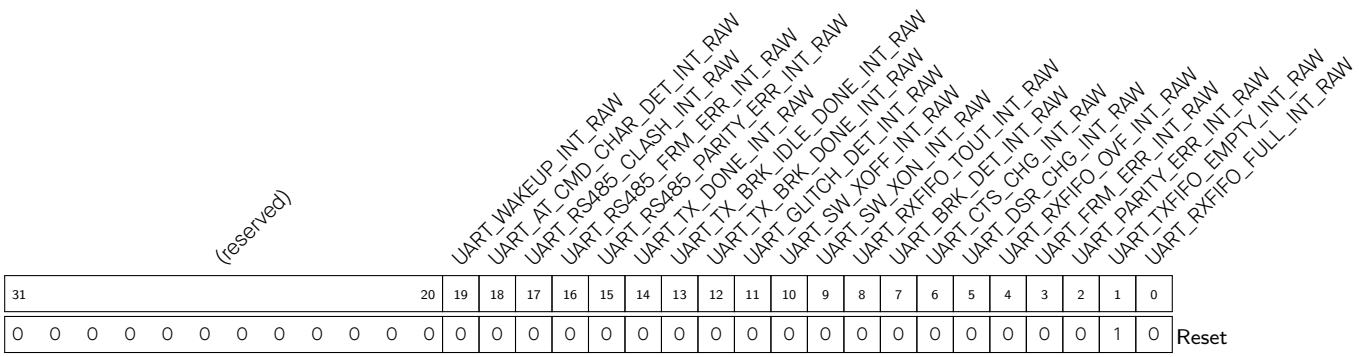


UART_RX_TOUT_EN Configures whether or not to enable UART receiver's timeout function.
0: Disable
1: Enable
(R/W)

UART_RX_TOUT_FLOW_DIS Configures whether or not to disable the idle status counter when hardware flow control is enabled.
0: Enable
1: Disable
(R/W)

UART_RX_TOUT_THRHD Configures the amount of time that the bus can remain idle before timeout.
Measurement unit: bit time (the time to transmit 1 bit). (R/W)

Register 45.3. UART_INT_RAW_REG (0x0004)



UART_RXFIFO_FULL_INT_RAW The raw interrupt status of UART_RXFIFO_FULL_INT. (R/WTC/SS)

UART_TXFIFO_EMPTY_INT_RAW The raw interrupt status of UART_TXFIFO_EMPTY_INT. (R/WTC/SS)

UART_PARITY_ERR_INT_RAW The raw interrupt status of UART_PARITY_ERR_INT. (R/WTC/SS)

UART_FRM_ERR_INT_RAW The raw interrupt status of UART_FRM_ERR_INT. (R/WTC/SS)

UART_RXFIFO_OVF_INT_RAW The raw interrupt status of UART_RXFIFO_OVF_INT. (R/WTC/SS)

UART_DSR_CHG_INT_RAW The raw interrupt status of UART_DSR_CHG_INT. (R/WTC/SS)

UART_CTS_CHG_INT_RAW The raw interrupt status of UART_CTS_CHG_INT. (R/WTC/SS)

UART_BRK_DET_INT_RAW The raw interrupt status of UART_BRK_DET_INT. (R/WTC/SS)

UART_RXFIFO_TOUT_INT_RAW The raw interrupt status of UART_RXFIFO_TOUT_INT. (R/WTC/SS)

UART_SW_XON_INT_RAW The raw interrupt status of UART_SW_XON_INT. (R/WTC/SS)

UART_SW_XOFF_INT_RAW UART_SW_XOFF_INT. (R/WTC/SS)

UART_GLITCH_DET_INT_RAW The raw interrupt status of UART_GLITCH_DET_INT. (R/WTC/SS)

UART_TX_BRK_DONE_INT_RAW The raw interrupt status of UART_TX_BRK_DONE_INT. (R/WTC/SS)

UART_TX_BRK_IDLE_DONE_INT_RAW The raw interrupt status of UART_TX_BRK_IDLE_DONE_INT. (R/WTC/SS)

UART_TX_DONE_INT_RAW The raw interrupt status of UART_TX_DONE_INT. (R/WTC/SS)

UART_RS485_PARITY_ERR_INT_RAW The raw interrupt status of UART_RS485_PARITY_ERR_INT. (R/WTC/SS)

UART_RS485_FRM_ERR_INT_RAW The raw interrupt status of UART_RS485_FRM_ERR_INT. (R/WTC/SS)

Continued on the next page...

Register 45.3. UART_INT_RAW_REG (0x0004)

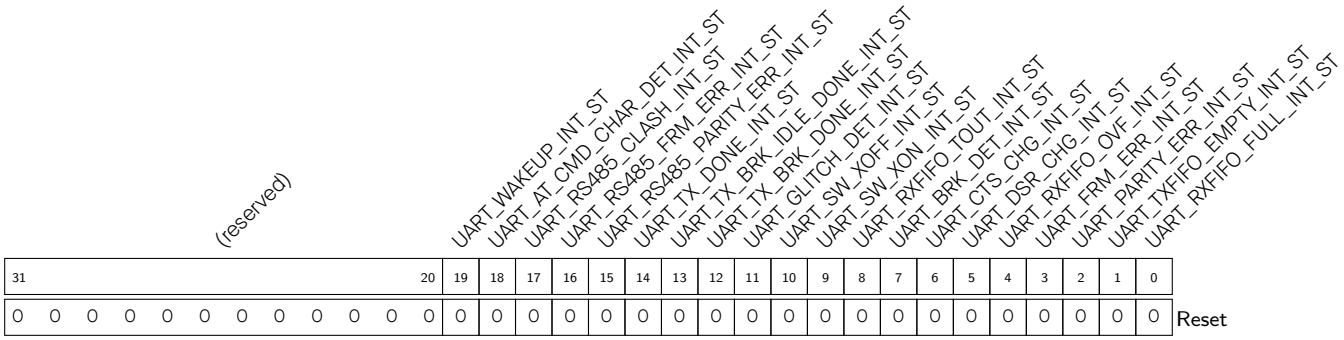
Continued from the previous page...

UART_RS485_CLASH_INT_RAW The raw interrupt status of UART_RS485_CLASH_INT. (R/WTC/SS)

UART_AT_CMD_CHAR_DET_INT_RAW The raw interrupt status of UART_AT_CMD_CHAR_DET_INT. (R/WTC/SS)

UART_WAKEUP_INT_RAW The raw interrupt status of UART_WAKEUP_INT. (R/WTC/SS)

Register 45.4. UART_INT_ST_REG (0x0008)



UART_RXFIFO_FULL_INT_ST The masked interrupt status of UART_RXFIFO_FULL_INT.(RO)

UART_TXFIFO_EMPTY_INT_ST The masked interrupt status of UART_TXFIFO_EMPTY_INT. (RO)

UART_PARITY_ERR_INT_ST The masked interrupt status of UART_PARITY_ERR_INT. (RO)

UART_FRM_ERR_INT_ST The masked interrupt status of UART_FRM_ERR_INT. (RO)

UART_RXFIFO_OVF_INT_ST The masked interrupt status of UART_RXFIFO_OVF_INT. (RO)

UART_DSR_CHG_INT_ST The masked interrupt status of UART_DSR_CHG_INT. (RO)

UART_CTS_CHG_INT_ST The masked interrupt status of UART_CTS_CHG_INT. (RO)

UART_BRK_DET_INT_ST The masked interrupt status of UART_BRK_DET_INT. (RO)

UART_RXFIFO_TOUT_INT_ST The masked interrupt status of UART_RXFIFO_TOUT_INT. (RO)

UART_SW_XON_INT_ST The masked interrupt status of UART_SW_XON_INT. (RO)

UART_SW_XOFF_INT_ST The masked interrupt status of UART_SW_XOFF_INT. (RO)

UART_GLITCH_DET_INT_ST The masked interrupt status of UART_GLITCH_DET_INT. (RO)

UART_TX_BRK_DONE_INT_ST The masked interrupt status of UART_TX_BRK_DONE_INT. (RO)

UART_TX_BRK_IDLE_DONE_INT_ST The masked interrupt status of UART_TX_BRK_IDLE_DONE_INT. (RO)

UART_TX_DONE_INT_ST The masked interrupt status of UART_TX_DONE_INT. (RO)

UART_RS485_PARITY_ERR_INT_ST The masked interrupt status of UART_RS485_PARITY_ERR_INT. (RO)

UART_RS485_FRM_ERR_INT_ST The masked interrupt status of UART_RS485_FRM_ERR_INT. (RO)

UART_RS485_CLASH_INT_ST The masked interrupt status of UART_RS485_CLASH_INT. (RO)

UART_AT_CMD_CHAR_DET_INT_ST The masked interrupt status of UART_AT_CMD_CHAR_DET_INT. (RO)

UART_WAKEUP_INT_ST The masked interrupt status of UART_WAKEUP_INT. (RO)

Register 45.5. UART_INT_ENA_REG (0x000C)



UART_RXFIFO_FULL_INT_ENA Write 1 to enable UART_RXFIFO_FULL_INT. (R/W)

UART_TXFIFO_EMPTY_INT_ENA Write 1 to enable UART_TXFIFO_EMPTY_INT. (R/W)

UART_PARITY_ERR_INT_ENA Write 1 to enable UART_PARITY_ERR_INT. (R/W)

UART_FRM_ERR_INT_ENA Write 1 to enable UART_FRM_ERR_INT. (R/W)

UART_RXFIFO_OVF_INT_ENA Write 1 to enable UART_RXFIFO_OVF_INT. (R/W)

UART_DSR_CHG_INT_ENA Write 1 to enable UART_DSR_CHG_INT. (R/W)

UART_CTS_CHG_INT_ENA Write 1 to enable UART_CTS_CHG_INT. (R/W)

UART_BRK_DET_INT_ENA Write 1 to enable UART_BRK_DET_INT. (R/W)

UART_RXFIFO_TOUT_INT_ENA Write 1 to enable UART_RXFIFO_TOUT_INT. (R/W)

UART_SW_XON_INT_ENA Write 1 to enable UART_SW_XON_INT.(R/W)

UART_SW_XOFF_INT_ENA Write 1 to enable UART_SW_XOFF_INT. (R/W)

UART_GLITCH_DET_INT_ENA Write 1 to enable UART_GLITCH_DET_INT. (R/W)

UART_TX_BRK_DONE_INT_ENA Write 1 to enable UART_TX_BRK_DONE_INT. (R/W)

UART_TX_BRK_IDLE_DONE_INT_ENA Write 1 to enable UART_TX_BRK_IDLE_DONE_INT. (R/W)

UART_TX_DONE_INT_ENA Write 1 to enable UART_TX_DONE_INT. (R/W)

UART_RS485_PARITY_ERR_INT_ENA Write 1 to enable UART_RS485_PARITY_ERR_INT. (R/W)

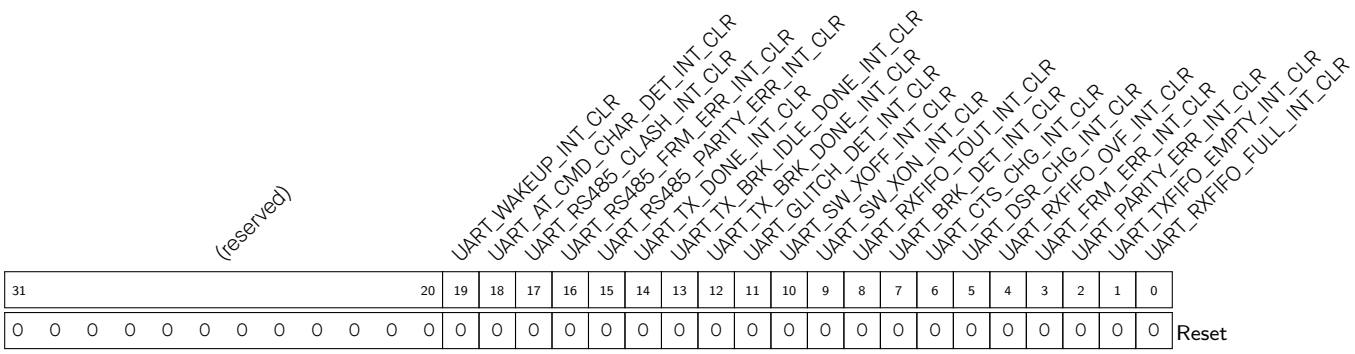
UART_RS485_FRM_ERR_INT_ENA Write 1 to enable UART_RS485_FRM_ERR_INT. (R/W)

UART_RS485_CLASH_INT_ENA Write 1 to enable UART_RS485_CLASH_INT. (R/W)

UART_AT_CMD_CHAR_DET_INT_ENA Write 1 to enable UART_AT_CMD_CHAR_DET_INT. (R/W)

UART_WAKEUP_INT_ENA Write 1 to enable UART_WAKEUP_INT. (R/W)

Register 45.6. UART_INT_CLR_REG (0x0010)



UART_RXFIFO_FULL_INT_CLR Write 1 to clear UART_RXFIFO_FULL_INT. (WT)

UART_TXFIFO_EMPTY_INT_CLR Write 1 to clear UART_TXFIFO_EMPTY_INT. (WT)

UART_PARITY_ERR_INT_CLR Write 1 to clear UART_PARITY_ERR_INT. (WT)

UART_FRM_ERR_INT_CLR Write 1 to clear UART_FRM_ERR_INT. (WT)

UART_RXFIFO_OVF_INT_CLR Write 1 to clear UART_RXFIFO_OVF_INT. (WT)

UART_DSR_CHG_INT_CLR Write 1 to clear UART_DSR_CHG_INT. (WT)

UART_CTS_CHG_INT_CLR Write 1 to clear UART_CTS_CHG_INT. (WT)

UART_BRK_DET_INT_CLR Write 1 to clear UART_BRK_DET_INT. (WT)

UART_RXFIFO_TOUT_INT_CLR Write 1 to clear UART_RXFIFO_TOUT_INT. (WT)

UART_SW_XON_INT_CLR Write 1 to clear UART_SW_XON_INT. (WT)

UART_SW_XOFF_INT_CLR Write 1 to clear UART_SW_XOFF_INT. (WT)

UART_GLITCH_DET_INT_CLR Write 1 to clear UART_GLITCH_DET_INT. (WT)

UART_TX_BRK_DONE_INT_CLR Write 1 to clear UART_TX_BRK_DONE_INT. (WT)

UART_TX_BRK_IDLE_DONE_INT_CLR Write 1 to clear UART_TX_BRK_IDLE_DONE_INT. (WT)

UART_TX_DONE_INT_CLR Write 1 to clear UART_TX_DONE_INT. (WT)

UART_RS485_PARITY_ERR_INT_CLR Write 1 to clear UART_RS485_PARITY_ERR_INT. (WT)

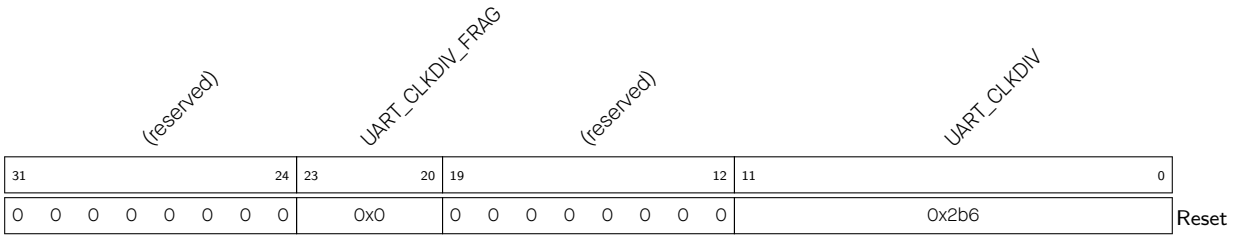
UART_RS485_FRM_ERR_INT_CLR Write 1 to clear UART_RS485_FRM_ERR_INT. (WT)

UART_RS485_CLASH_INT_CLR Write 1 to clear UART_RS485_CLASH_INT. (WT)

UART_AT_CMD_CHAR_DET_INT_CLR Write 1 to clear UART_AT_CMD_CHAR_DET_INT. (WT)

UART_WAKEUP_INT_CLR Write 1 to clear UART_WAKEUP_INT. (WT)

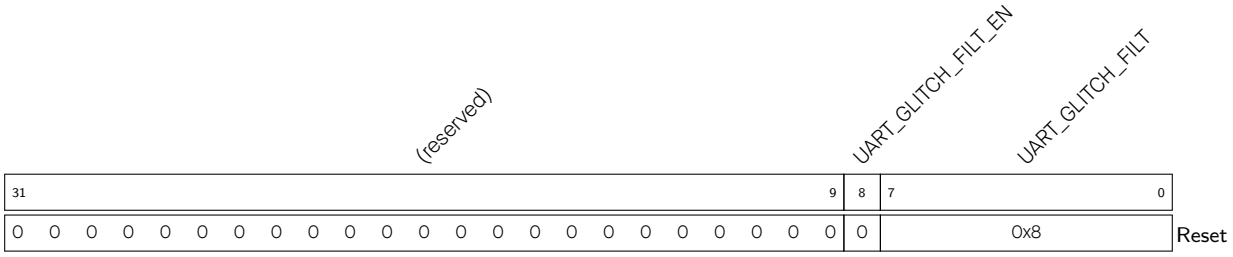
Register 45.7. UART_CLKDIV_SYNC_REG (0x0014)



UART_CLKDIV Configures the integral part of the divisor for baud rate generation. (R/W)

UART_CLKDIV_FRAG Configures the fractional part of the divisor for baud rate generation. (R/W)

Register 45.8. UART_RX_FILT_REG (0x0018)



UART_GLITCH_FILT Configures the width of a pulse to be filtered.

Measurement unit: UART Core's clock cycle.

Pulses whose width is lower than this value will be ignored. (R/W)

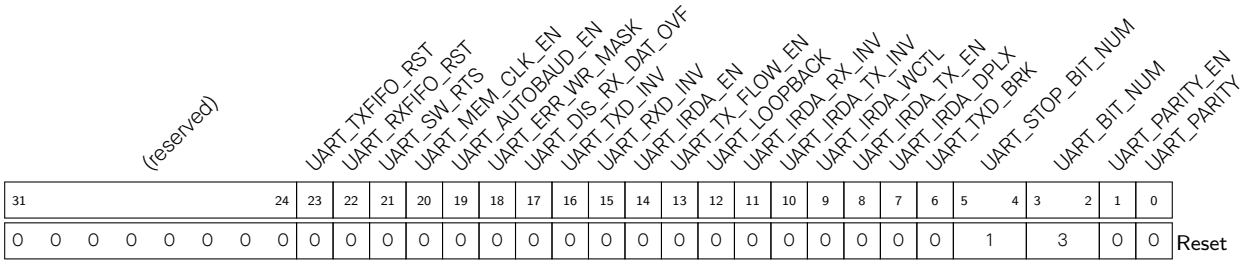
UART_GLITCH_FILT_EN Configures whether or not to enable RX signal filter.

0: Disable

1: Enable

(R/W)

Register 45.9. UART_CONF0_SYNC_REG (0x0020)



UART_PARITY Configures the parity check mode.

0: Even parity

1: Odd parity

(R/W)

UART_PARITY_EN Configures whether or not to enable UART parity check.

0: Disable

1: Enable

(R/W)

UART_BIT_NUM Configures the number of data bits.

0: 5 bits

1: 6 bits

2: 7 bits

3: 8 bits

(R/W)

UART_STOP_BIT_NUM Configures the number of stop bits.

0: Invalid. No effect

1: 1 bit

2: 1.5 bits

3: 2 bits

(R/W)

UART_TXD_BRK Configures whether or not to send NULL characters when finishing data transmission.

0: Not send

1: Send

(R/W)

UART_IRDA_DPLX Configures whether or not to enable IrDA loopback test.

0: Disable

1: Enable

(R/W)

UART_IRDA_TX_EN Configures whether or not to enable the IrDA transmitter.

0: Disable

1: Enable

(R/W)

Continued on the next page...

Register 45.9. UART_CONF0_SYNC_REG (0x0020)

Continued from the previous page...

UART_IRDA_WCTL Configures the 11th bit of the IrDA transmitter.

0: This bit is 0.

1: This bit is the same as the 10th bit.

(R/W)

UART_IRDA_TX_INV Configures whether or not to invert the level of the IrDA transmitter.

0: Not invert

1: Invert

(R/W)

UART_IRDA_RX_INV Configures whether or not to invert the level of the IrDA receiver.

0: Not invert

1: Invert

(R/W)

UART_LOOPBACK Configures whether or not to enable UART loopback test.

0: Disable

1: Enable

(R/W)

UART_TX_FLOW_EN Configures whether or not to enable flow control for the transmitter.

0: Disable

1: Enable

(R/W)

UART_IRDA_EN Configures whether or not to enable IrDA protocol.

0: Disable

1: Enable

(R/W)

UART_RXD_INV Configures whether or not to invert the level of UART RXD signal.

0: Not invert

1: Invert

(R/W)

UART_TXD_INV Configures whether or not to invert the level of UART TXD signal.

0: Not invert

1: Invert

(R/W)

UART_DIS_RX_DAT_OVF Configures whether or not to disable data overflow detection for the UART receiver.

0: Enable

1: Disable

(R/W)

Continued on the next page...

Register 45.9. UART_CONF0_SYNC_REG (0x0020)

Continued from the previous page...

UART_ERR_WR_MASK Configures whether or not to store the received data with errors into FIFO.

0: Store

1: Not store

(R/W)

UART_AUTOBAUD_EN Configures whether or not to enable baud rate detection.

0: Disable

1: Enable

(R/W)

UART_MEM_CLK_EN Configures whether or not to enable clock gating for UART memory.

0: Disable

1: Enable

(R/W)

UART_SW_RTS Configures the RTS signal used in software flow control.

0: The UART transmitter is not allowed to send data.

1: The UART transmitted is allowed to send data.

(R/W)

UART_RXFIFO_RST Configures whether or not to reset the UART RX FIFO.

0: Not reset

1: Reset

(R/W)

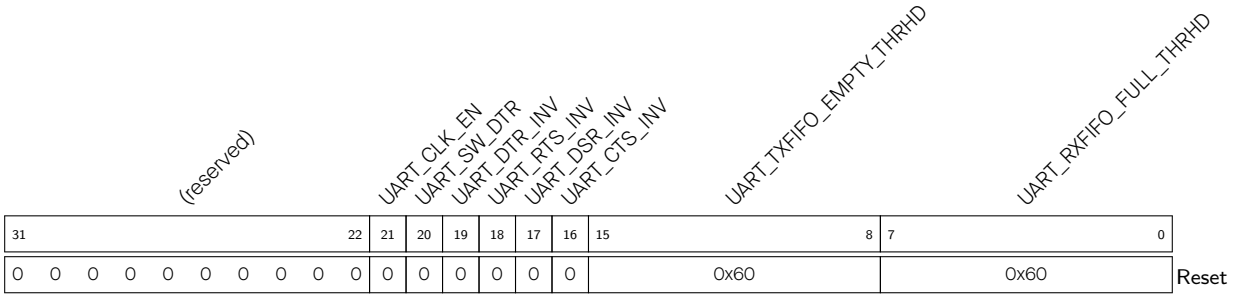
UART_TXFIFO_RST Configures whether or not to reset the UART TX FIFO.

0: Not reset

1: Reset

(R/W)

Register 45.10. UART_CONF1_REG (0x0024)



UART_RXFIFO_FULL_THRHD Configures the threshold for RX FIFO being full.

Measurement unit: byte. (R/W)

UART_TXFIFO_EMPTY_THRHD Configures the threshold for TX FIFO being empty.

Measurement unit: byte. (R/W)

UART_CTS_INV Configures whether or not to invert the level of UART CTS signal.

0: Not invert

1: Invert

(R/W)

UART_DSR_INV Configures whether or not to invert the level of UART DSR signal.

0: Not invert

1: Invert

(R/W)

UART_RTS_INV Configures whether or not to invert the level of UART RTS signal.

0: Not invert

1: Invert

(R/W)

UART_DTR_INV Configures whether or not to invert the level of UART DTR signal.

0: Not invert

1: Invert

(R/W)

UART_SW_DTR Configures the DTR signal used in software flow control.

0: Data to be transmitted is not ready.

1: Data to be transmitted is ready.

(R/W)

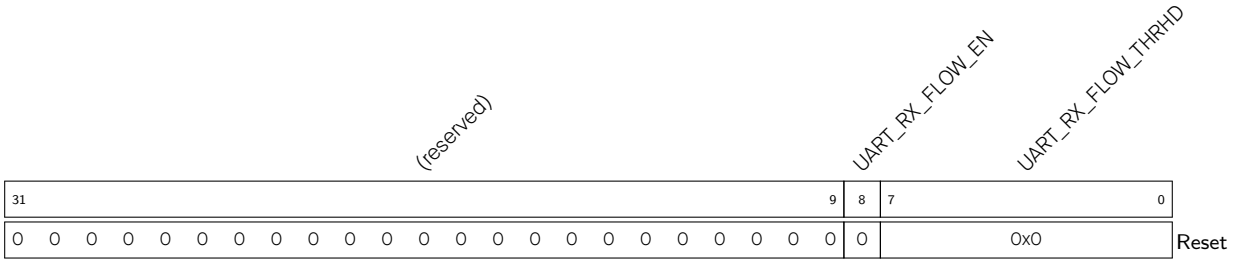
UART_CLK_EN Configures clock gating.

0: Support clock only when the application writes registers.

1: Always force the clock on for registers.

(R/W)

Register 45.11. UART_HWFC_CONF_SYNC_REG (0x002C)



UART_RX_FLOW_THRHD Configures the maximum number of data bytes that can be received during hardware flow control.

Measurement unit: byte. (R/W)

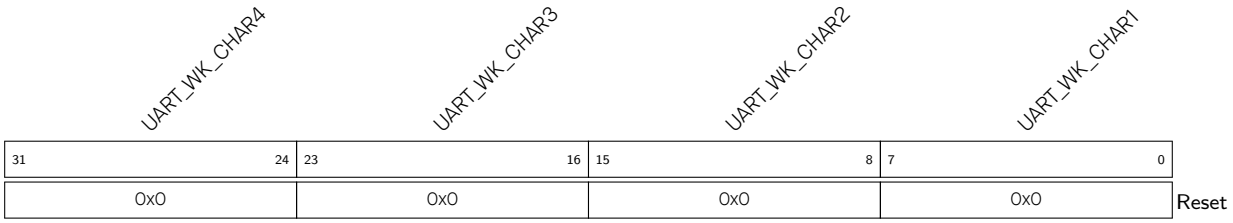
UART_RX_FLOW_EN Configures whether or not to enable the UART receiver.

0: Disable

1: Enable

(R/W)

Register 45.12. UART_SLEEP_CONF0_REG (0x0030)



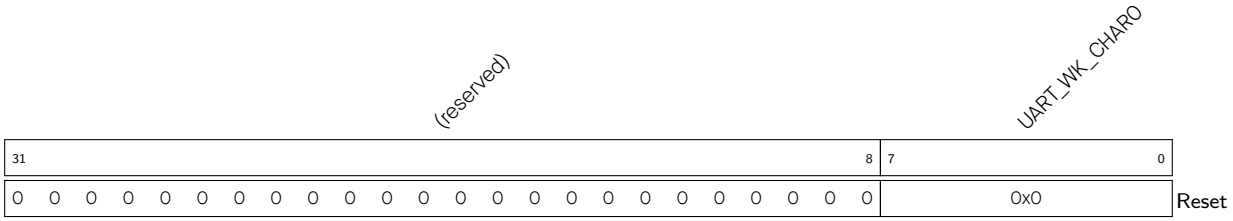
UART_WK_CHAR1 Configures wakeup character 1. (R/W)

UART_WK_CHAR2 Configures wakeup character 2. (R/W)

UART_WK_CHAR3 Configures wakeup character 3. (R/W)

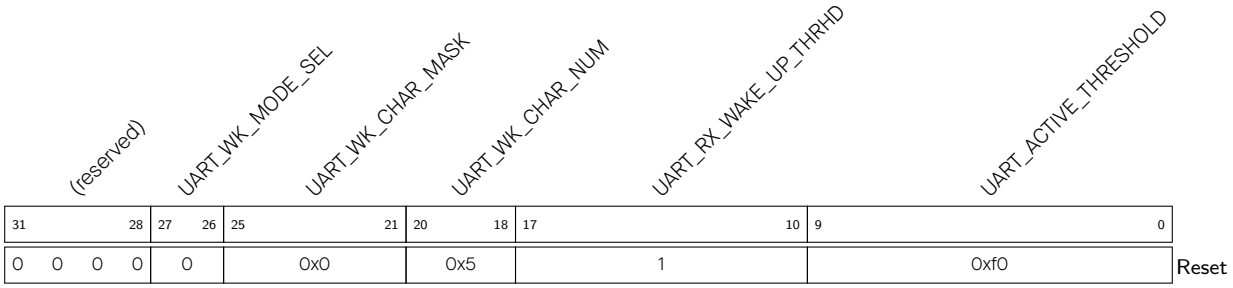
UART_WK_CHAR4 Configures wakeup character 4. (R/W)

Register 45.13. UART_SLEEP_CONF1_REG (0x0034)



UART_WK_CHAR0 Configures wakeup character 0. (R/W)

Register 45.14. UART_SLEEP_CONF2_REG (0x0038)



UART_ACTIVE_THRESHOLD Configures the number of RXD edge changes to wake up the chip in wakeup mode 0. (R/W)

UART_RX_WAKE_UP_THRHD Configures the number of received data bytes to wake up the chip in wakeup mode 1. (R/W)

UART_WK_CHAR_NUM Configures the number of wakeup characters. (R/W)

UART_WK_CHAR_MASK Configures whether or not to mask wakeup characters.

0: Not mask

1: Mask

(R/W)

UART_WK_MODE_SEL Configures which wakeup mode to select. See Section 45.4.8 *Wakeup* for the explanation of each mode.

0: Mode 0

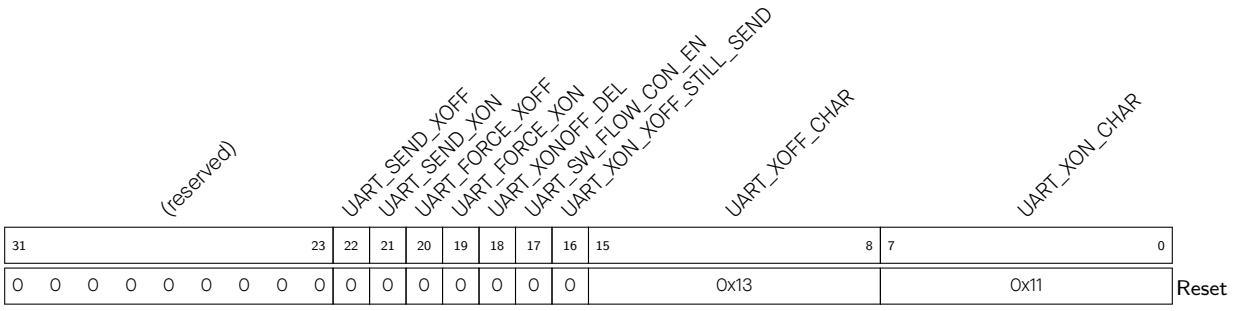
1: Mode 1

2: Mode 2

3: Mode 3

(R/W)

Register 45.15. UART_SWFC_CONF0_SYNC_REG (0x003C)



UART_XON_CHAR Configures the XON character for flow control. (R/W)

UART_XOFF_CHAR Configures the XOFF character for flow control. (R/W)

UART_XON_XOFF_STILL_SEND Configures whether the UART transmitter can send XON or XOFF characters when it is disabled.

0: Cannot send

1: Can send

(R/W)

UART_SW_FLOW_CON_EN Configures whether or not to enable software flow control.

0: Disable

1: Enable

(R/W)

UART_XONOFF_DEL Configures whether or not to remove flow control characters from the received data.

0: Not move

1: Move

(R/W)

UART_FORCE_XON Configures whether the transmitter continues to sending data.

0: Not send

1: Send

(R/W)

UART_FORCE_XOFF Configures whether or not to stop the transmitter from sending data.

0: Not stop

1: Stop

(R/W)

UART_SEND_XON Configures whether or not to send XON characters.

0: Not send

1: Send

(R/W/SS/SC)

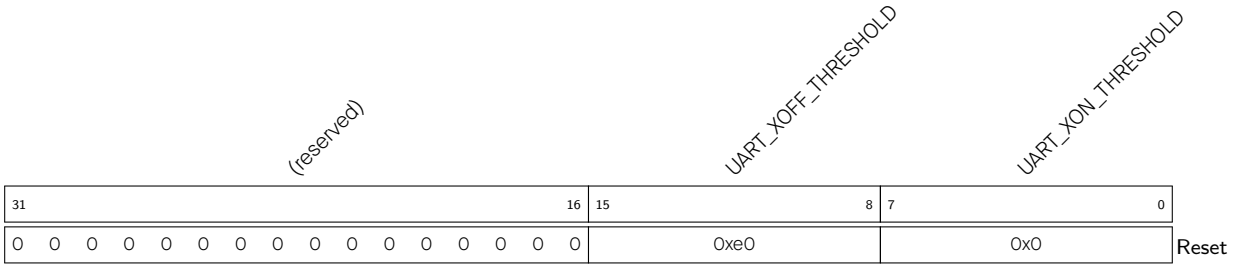
UART_SEND_XOFF Configures whether or not to send XOFF characters.

0: Not send

1: Send

(R/W/SS/SC)

Register 45.16. UART_SWFC_CONF1_REG (0x0040)



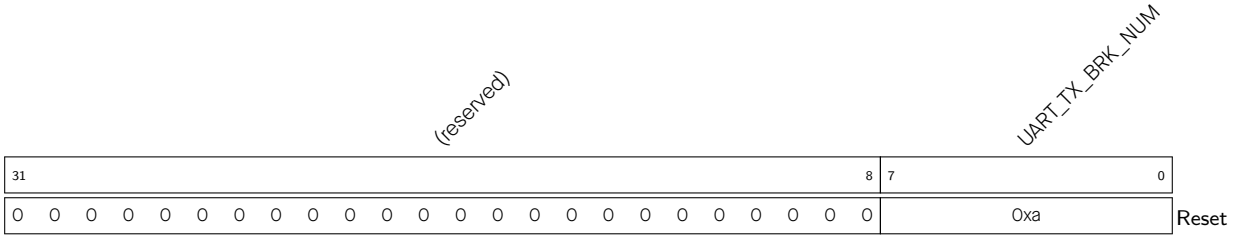
UART_XON_THRESHOLD Configures the threshold for data in RX FIFO to send XON characters in software flow control.

Measurement unit: byte. (R/W)

UART_XOFF_THRESHOLD Configures the threshold for data in RX FIFO to send XOFF characters in software flow control.

Measurement unit: byte. (R/W)

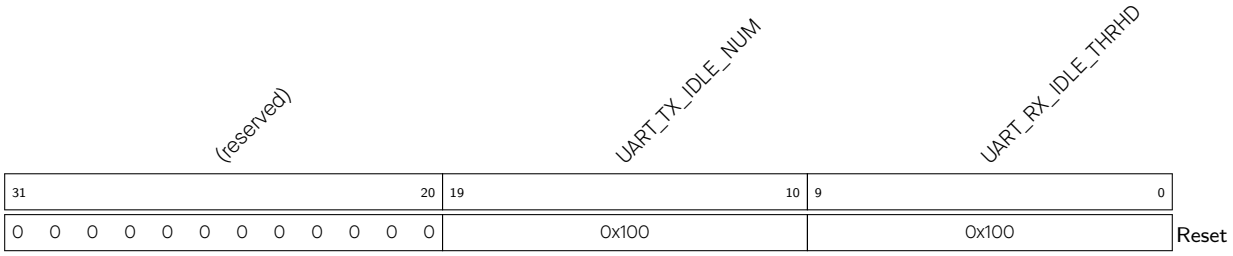
Register 45.17. UART_TXBRK_CONF_SYNC_REG (0x0044)



UART_TX_BRK_NUM Configures the number of NULL characters to be sent after finishing data transmission.

Valid only when UART_TXD_BRK is 1. (R/W)

Register 45.18. UART_IDLE_CONF_SYNC_REG (0x0048)



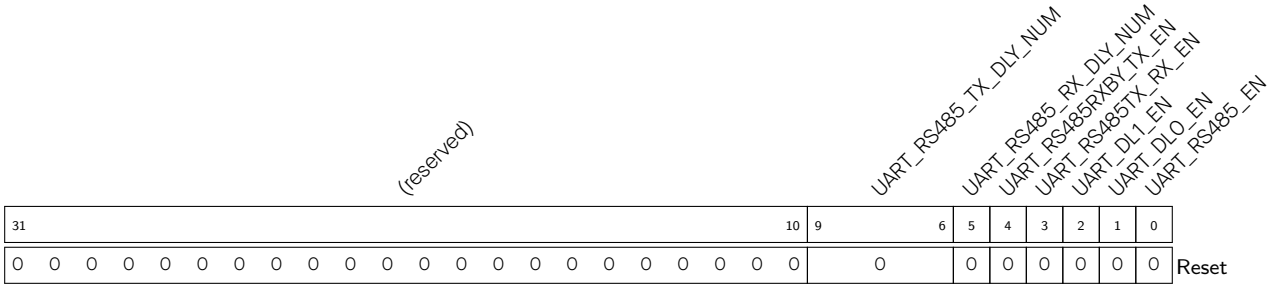
UART_RX_IDLE_THRHD Configures the threshold to generate a frame end signal when the receiver takes more time to receive one data byte data.

Measurement unit: bit time (the time to transmit 1 bit). (R/W)

UART_TX_IDLE_NUM Configures the interval between two data transfers.

Measurement unit: bit time (the time to transmit 1 bit). (R/W)

Register 45.19. UART_RS485_CONF_SYNC_REG (0x004C)



UART_RS485_EN Configures whether or not to enable RS485 mode.

0: Disable

1: Enable

(R/W)

UART_DL0_EN Configures whether or not to add a turnaround delay of 1 bit before the start bit.

0: Not add

1: Add

(R/W)

UART_DL1_EN Configures whether or not to add a turnaround delay of 1 bit after the stop bit.

0: Not add

1: Add

(R/W)

UART_RS485TX_RX_EN Configures whether or not to enable the receiver for data reception when the transmitter is transmitting data in RS485 mode.

0: Disable

1: Enable

(R/W)

UART_RS485RXBY_TX_EN Configures whether to enable the RS485 transmitter for data transmission when the RS485 receiver is busy.

0: Disable

1: Enable

(R/W)

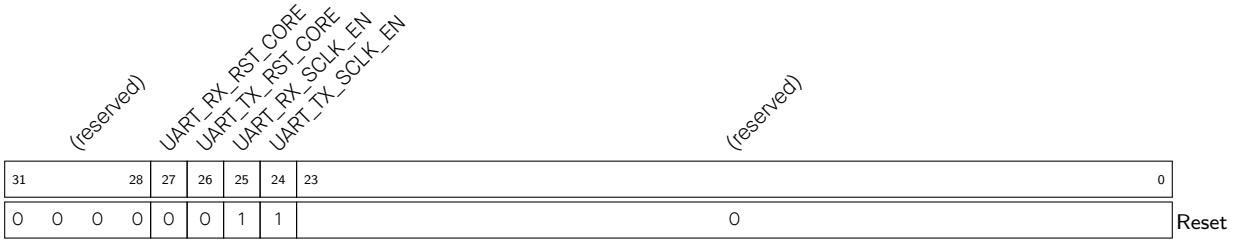
UART_RS485_RX_DLY_NUM Configures the delay of internal data signals in the receiver.

Measurement unit: bit time (the time to transmit 1 bit).. (R/W)

UART_RS485_TX_DLY_NUM Configures the delay of internal data signals in the transmitter.

Measurement unit: bit time (the time to transmit 1 bit). (R/W)

Register 45.20. UART_CLK_CONF_REG (0x0088)



UART_TX_SCLK_EN Configures whether or not to enable UART TX clock.

0: Disable

1: Enable

(R/W)

UART_RX_SCLK_EN Configures whether or not to enable UART RX clock.

0: Disable

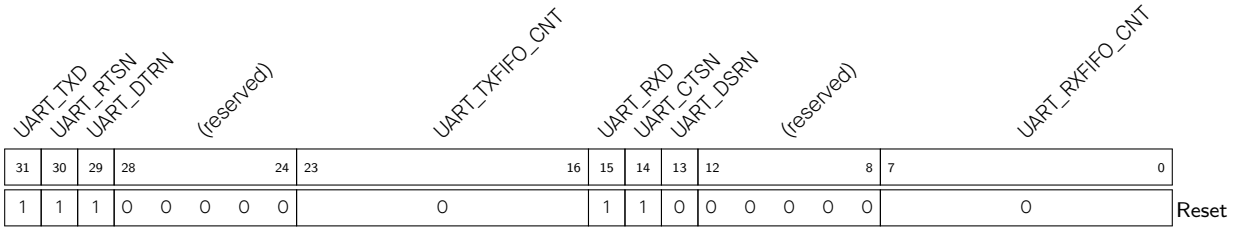
1: Enable

(R/W)

UART_TX_RST_CORE Write 1 and then write 0 to reset UART TX. (R/W)

UART_RX_RST_CORE Write 1 and then write 0 to reset UART RX. (R/W)

Register 45.21. UART_STATUS_REG (0x001C)



UART_RXFIFO_CNT Represents the number of valid data bytes in RX FIFO. (RO)

UART_DSRN Represents the level of the internal UART DSR signal. (RO)

UART_CTSN Represents the level of the internal UART CTS signal. (RO)

UART_RXD Represents the level of the internal UART RXD signal. (RO)

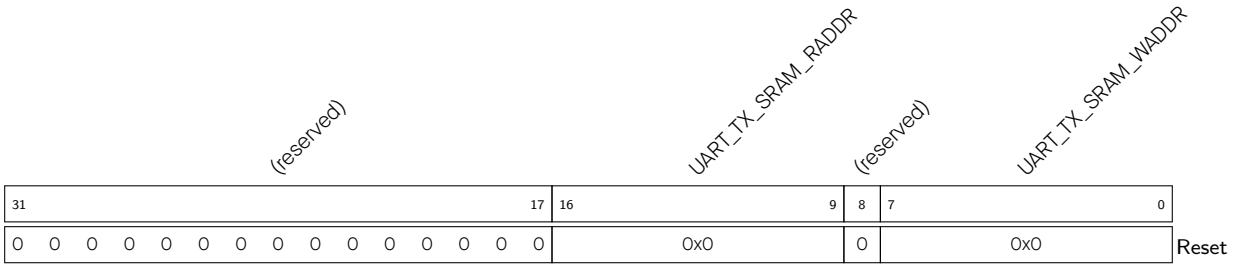
UART_TXFIFO_CNT Represents the number of valid data bytes in RX FIFO. (RO)

UART_DTRN Represents the level of the internal UART DTR signal. (RO)

UART_RTSN Represents the level of the internal UART RTS signal. (RO)

UART_TXD Represents the level of the internal UART TXD signal. (RO)

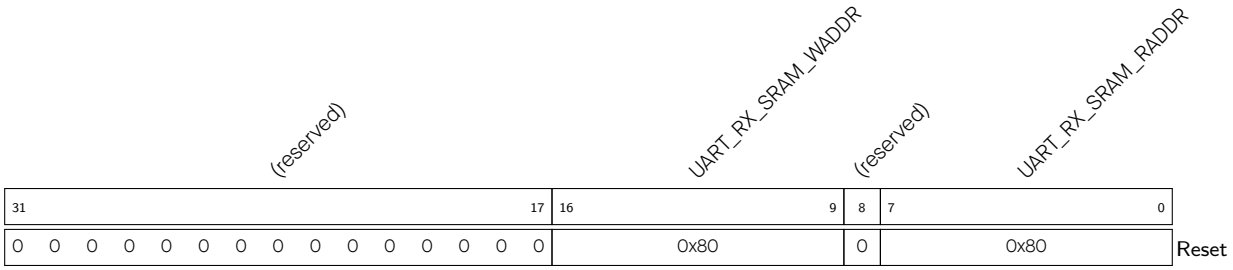
Register 45.22. UART_MEM_TX_STATUS_REG (0x0068)



UART_TX_SRAM_WADDR Represents the offset address to write TX FIFO. (RO)

UART_TX_SRAM_RADDR Represents the offset address to read TX FIFO. (RO)

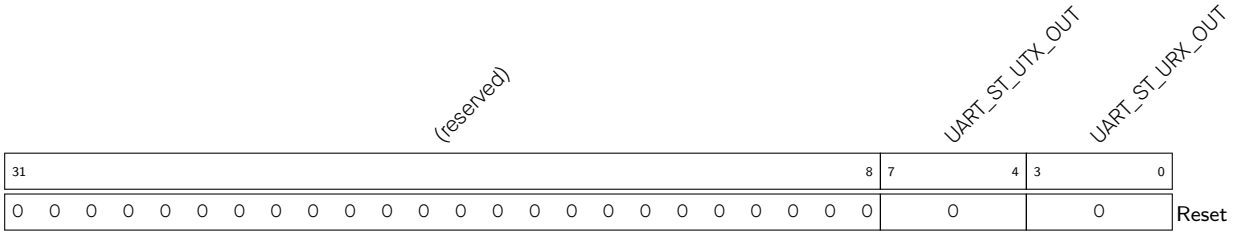
Register 45.23. UART_MEM_RX_STATUS_REG (0x006C)



UART_RX_SRAM_RADDR Represents the offset address to read RX FIFO. (RO)

UART_RX_SRAM_WADDR Represents the offset address to write RX FIFO. (RO)

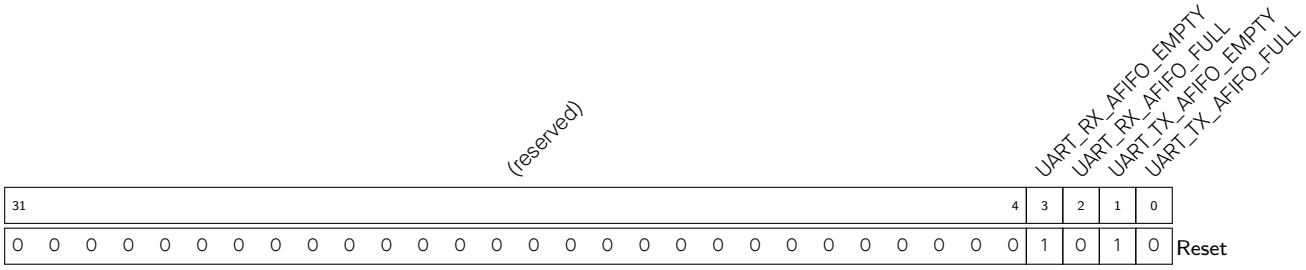
Register 45.24. UART_FSM_STATUS_REG (0x0070)



UART_ST_URX_OUT Represents the status of the receiver. (RO)

UART_ST_UTX_OUT Represents the status of the transmitter. (RO)

Register 45.25. UART_AFIFO_STATUS_REG (0x0090)



UART_TX_AFIFO_FULL Represents whether or not the APB TX asynchronous FIFO is full.

0: Not full

1: Full

(RO)

UART_TX_AFIFO_EMPTY Represents whether or not the APB TX asynchronous FIFO is empty.

0: Not empty

1: Empty

(RO)

UART_RX_AFIFO_FULL Represents whether or not the APB RX asynchronous FIFO is full.

0: Not full

1: Full

(RO)

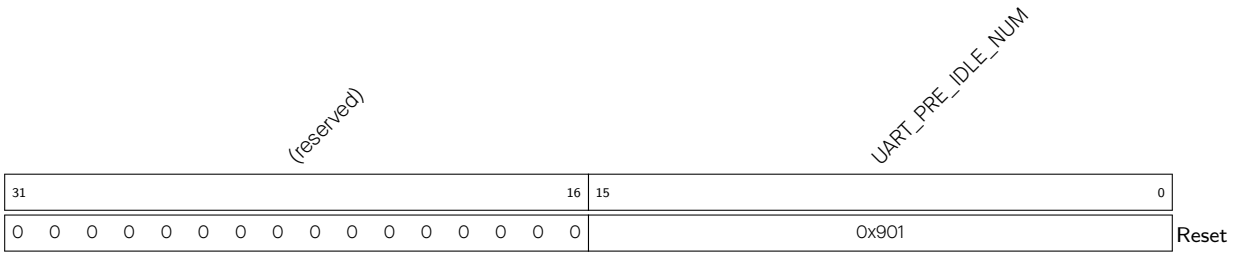
UART_RX_AFIFO_EMPTY Represents whether or not the APB RX asynchronous FIFO is empty.

0: Not empty

1: Empty

(RO)

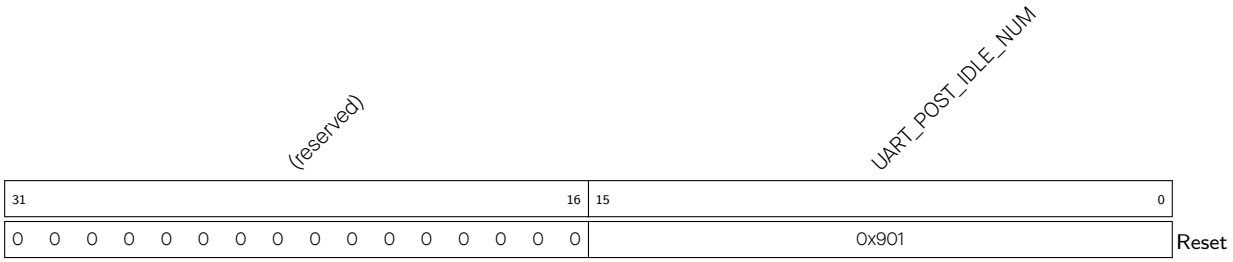
Register 45.26. UART_AT_CMD_PRECNT_SYNC_REG (0x0050)



UART_PRE_IDLE_NUM Configures the idle time before the receiver receives the first AT_CMD.

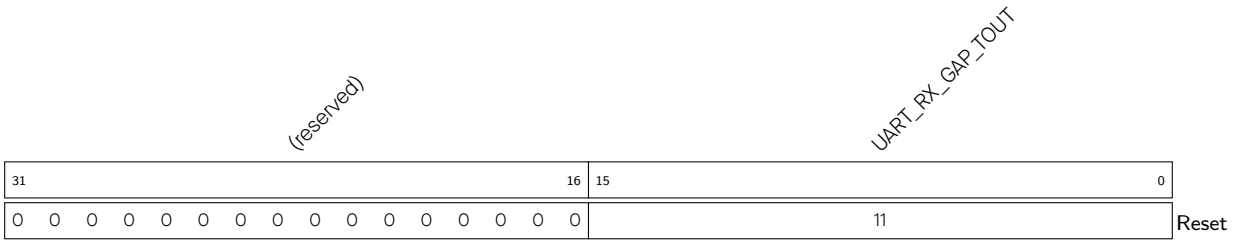
Measurement unit: bit time (the time to transmit 1 bit). (R/W)

Register 45.27. UART_AT_CMD_POSTCNT_SYNC_REG (0x0054)



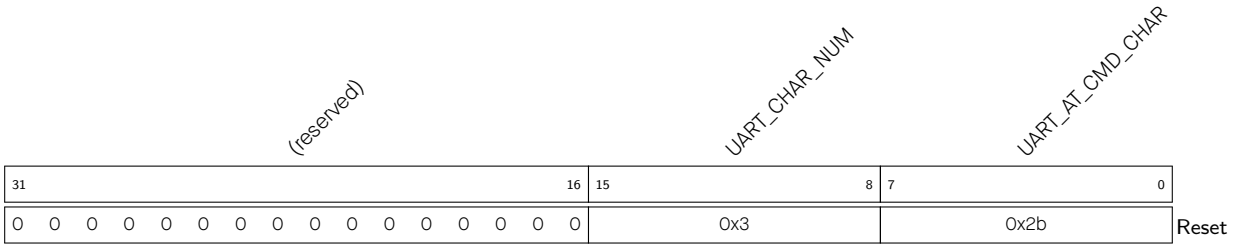
UART_POST_IDLE_NUM Configures the interval between the last AT_CMD and subsequent data. Measurement unit: bit time (the time to transmit 1 bit). (R/W)

Register 45.28. UART_AT_CMD_GAPTOUT_SYNC_REG (0x0058)



UART_RX_GAP_TOUT Configures the interval between two AT_CMD characters. Measurement unit: bit time (the time to transmit 1 bit). (R/W)

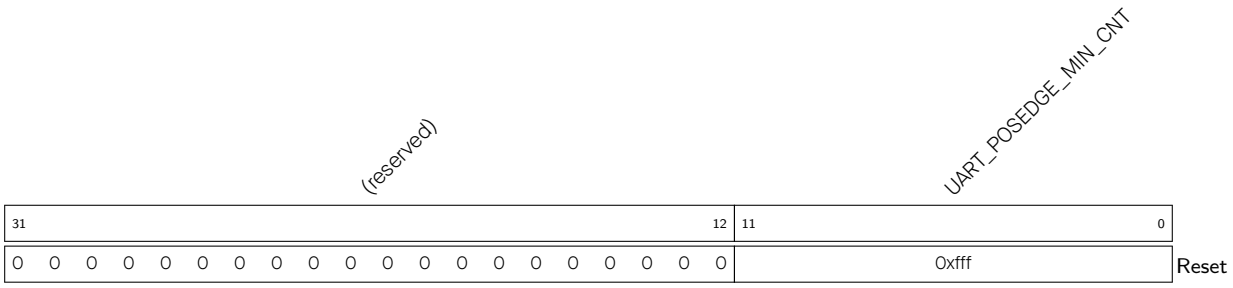
Register 45.29. UART_AT_CMD_CHAR_SYNC_REG (0x005C)



UART_AT_CMD_CHAR Configures the AT_CMD character. (R/W)

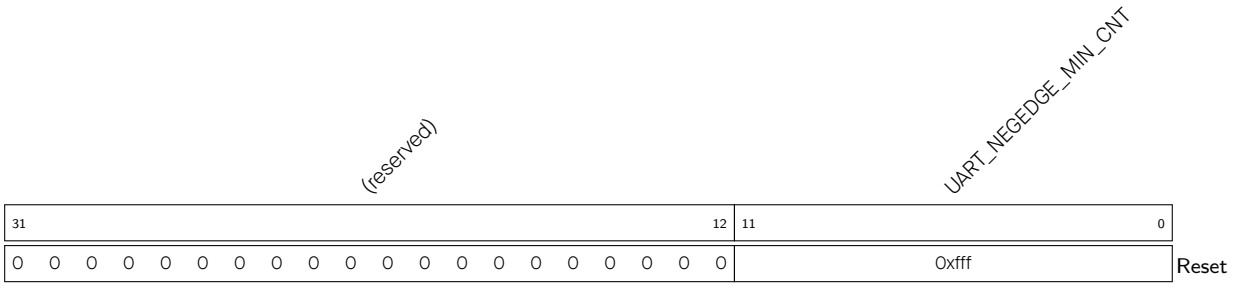
UART_CHAR_NUM Configures the number of continuous AT_CMD characters a receiver can receive. (R/W)

Register 45.30. UART_POSPULSE_REG (0x0074)



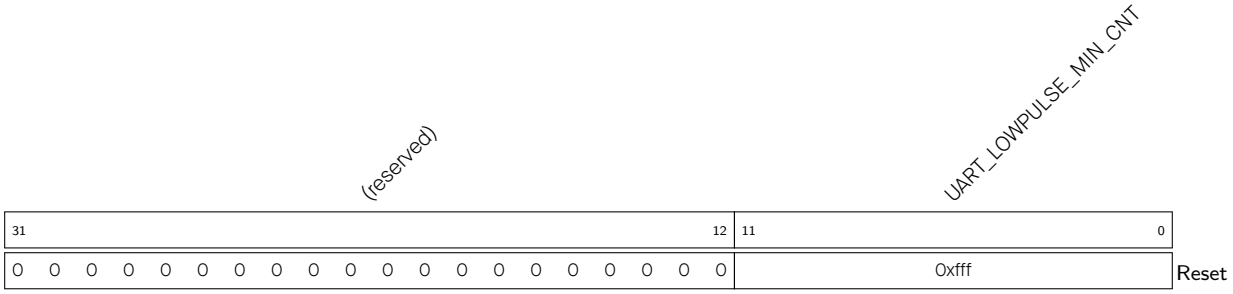
UART_POSEDGE_MIN_CNT Represents the minimal input clock counter value between two positive edges. It is used for baud rate detection. (RO)

Register 45.31. UART_NEGPULSE_REG (0x0078)



UART_NEGEDGE_MIN_CNT Represents the minimal input clock counter value between two negative edges. It is used for baud rate detection. (RO)

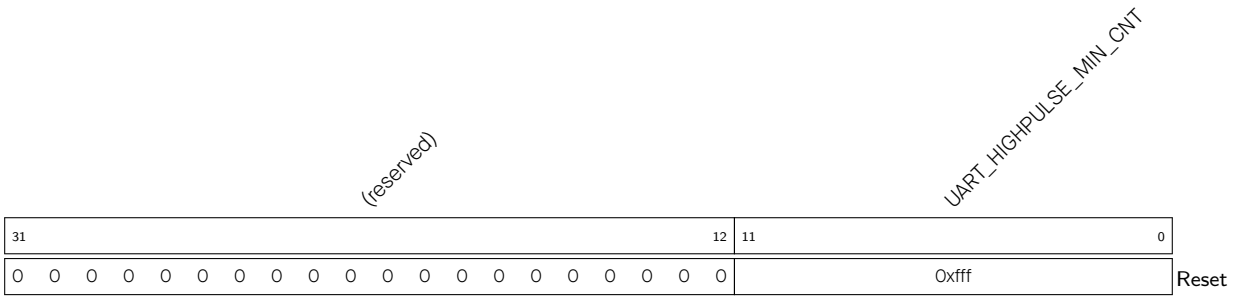
Register 45.32. UART_LOWPULSE_REG (0x007C)



UART_LOWPULSE_MIN_CNT Represents the minimum duration time of a low-level pulse. It is used for baud rate detection.

Measurement unit: APB_CLK clock cycle. (RO)

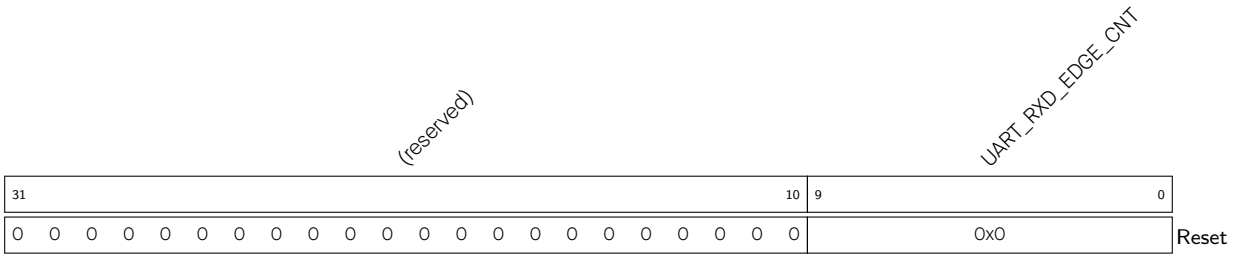
Register 45.33. UART_HIGHPULSE_REG (0x0080)



UART_HIGHPULSE_MIN_CNT Represents the maximum duration time for a high-level pulse. It is used for baud rate detection.

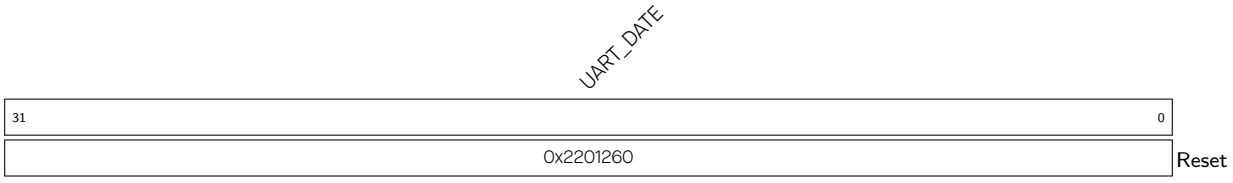
Measurement unit: APB_CLK clock cycle. (RO)

Register 45.34. UART_RXD_CNT_REG (0x0084)



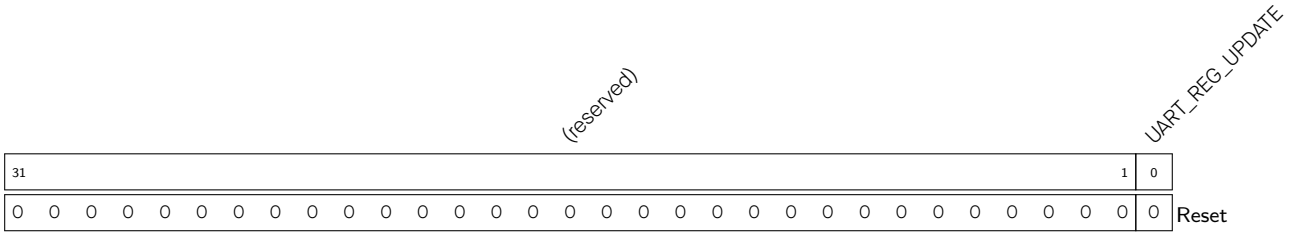
UART_RXD_EDGE_CNT Represents the number of RXD edge changes. It is used for baud rate detection. (RO)

Register 45.35. UART_DATE_REG (0x008C)



UART_DATE Version control register. (R/W)

Register 45.36. UART_REG_UPDATE_REG (0x0098)



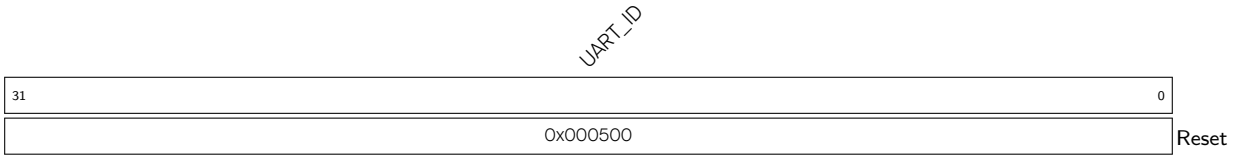
UART_REG_UPDATE Configures whether or not to synchronize registers.

0: Not synchronize

1: Synchronize

(R/W/SC)

Register 45.37. UART_ID_REG (0x009C)



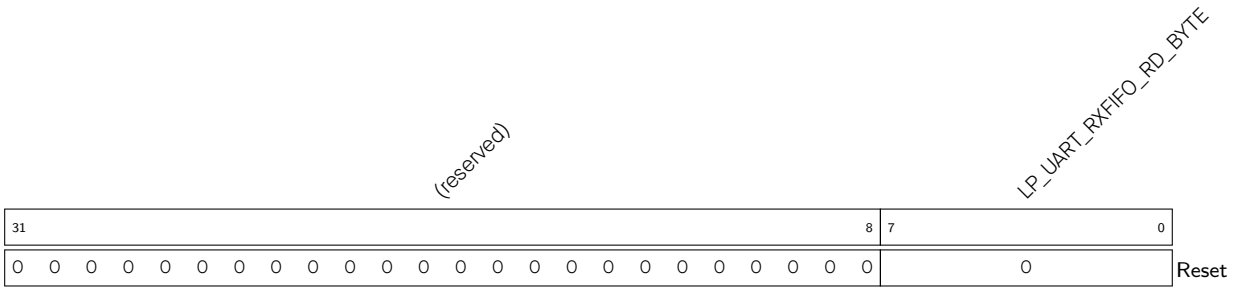
UART_ID Configures the UART ID. (R/W)

45.7.2 LP UART Registers

The addresses in this section are relative to LP UART base address provided in Table 9.3-2 in Chapter 9 *System and Memory*.

For how to program reserved fields, please refer to Section *Programming Reserved Register Field*.

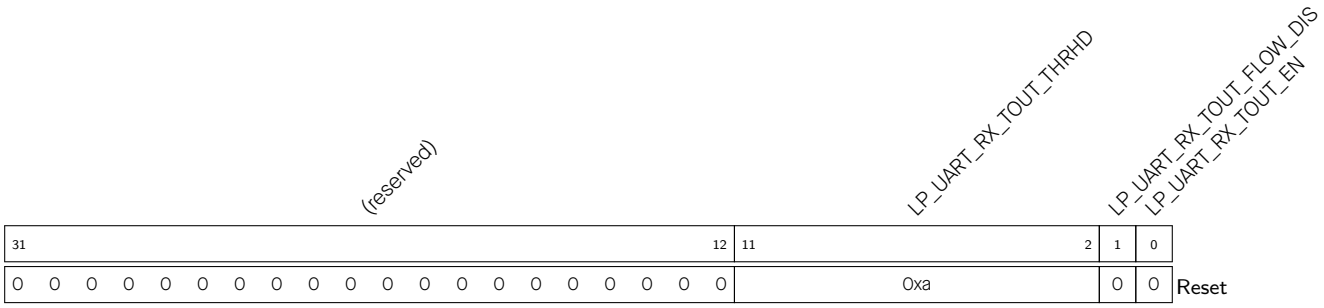
Register 45.38. LP_UART_FIFO_REG (0x0000)



LP_UART_RXFIFO_RD_BYTE Represents the data LP UART *n* read from FIFO.

Measurement unit: byte. (RO)

Register 45.39. LP_UART_TOUT_CONF_SYNC_REG (0x0064)



LP_UART_RX_TOUT_EN Configures whether or not to enable LP UART receiver's timeout function.

0: Disable

1: Enable

(R/W)

LP_UART_RX_TOUT_FLOW_DIS Configures whether or not to disable the idle status counter when hardware flow control is enabled.

0: Invalid. No effect

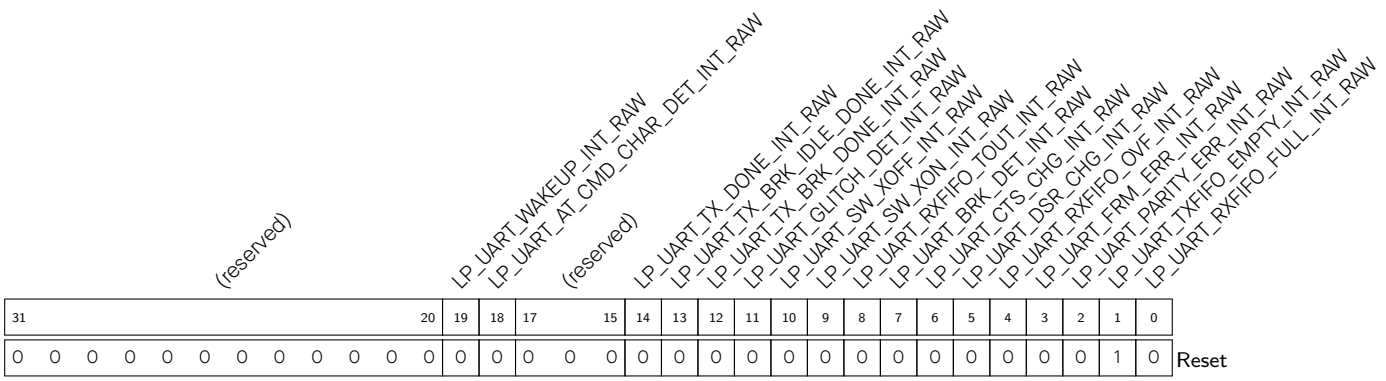
1: Disable

(R/W)

LP_UART_RX_TOUT_THRHD Configures the amount of time that the bus can remain idle before time-out.

Measurement unit: bit time (the time to transmit 1 bit). (R/W)

Register 45.40. LP_UART_INT_RAW_REG (0x0004)



LP_UART_RXFIFO_FULL_INT_RAW The raw interrupt status of LP_UART_RXFIFO_FULL_INT. (R/WTC/SS)

LP_UART_TXFIFO_EMPTY_INT_RAW The raw interrupt status of LP_UART_TXFIFO_EMPTY_INT. (R/WTC/SS)

LP_UART_PARITY_ERR_INT_RAW The raw interrupt status of LP_UART_PARITY_ERR_INT. (R/WTC/SS)

LP_UART_FRM_ERR_INT_RAW The raw interrupt status of LP_UART_FRM_ERR_INT. (R/WTC/SS)

LP_UART_RXFIFO_OVF_INT_RAW The raw interrupt status of LP_UART_RXFIFO_OVF_INT. (R/WTC/SS)

LP_UART_DSR_CHG_INT_RAW The raw interrupt status of LP_UART_DSR_CHG_INT. (R/WTC/SS)

LP_UART_CTS_CHG_INT_RAW The raw interrupt status of LP_UART_CTS_CHG_INT. (R/WTC/SS)

LP_UART_BRK_DET_INT_RAW The raw interrupt status of LP_UART_BRK_DET_INT. (R/WTC/SS)

LP_UART_RXFIFO_TOUT_INT_RAW The raw interrupt status of LP_UART_RXFIFO_TOUT_INT. (R/WTC/SS)

LP_UART_SW_XON_INT_RAW The raw interrupt status of LP_UART_SW_XON_INT. (R/WTC/SS)

LP_UART_SW_XOFF_INT_RAW LP_UART_SW_XOFF_INT. (R/WTC/SS)

LP_UART_GLITCH_DET_INT_RAW The raw interrupt status of LP_UART_GLITCH_DET_INT. (R/WTC/SS)

LP_UART_TX_BRK_DONE_INT_RAW The raw interrupt status of LP_UART_TX_BRK_DONE_INT. (R/WTC/SS)

LP_UART_TX_BRK_IDLE_DONE_INT_RAW The raw interrupt status of LP_UART_TX_BRK_IDLE_DONE_INT. (R/WTC/SS)

Continued on the next page...

Register 45.40. LP_UART_INT_RAW_REG (0x0004)

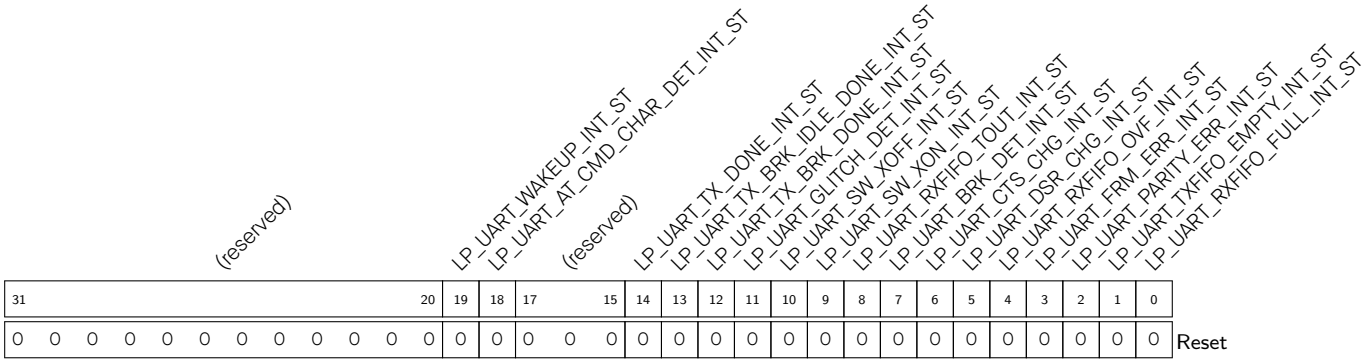
Continued from the previous page...

LP_UART_TX_DONE_INT_RAW The raw interrupt status of LP_UART_TX_DONE_INT. (R/WTC/SS)

LP_UART_AT_CMD_CHAR_DET_INT_RAW The raw interrupt status of LP_UART_AT_CMD_CHAR_DET_INT. (R/WTC/SS)

LP_UART_WAKEUP_INT_RAW The raw interrupt status of LP_UART_WAKEUP_INT. (R/WTC/SS)

Register 45.41. LP_UART_INT_ST_REG (0x0008)



LP_UART_RXFIFO_FULL_INT_ST The masked interrupt status of LP_UART_RXFIFO_FULL_INT.(RO)

LP_UART_TXFIFO_EMPTY_INT_ST The masked interrupt status of LP_UART_TXFIFO_EMPTY_INT. (RO)

LP_UART_PARITY_ERR_INT_ST The masked interrupt status of LP_UART_PARITY_ERR_INT. (RO)

LP_UART_FRM_ERR_INT_ST The masked interrupt status of LP_UART_FRM_ERR_INT. (RO)

LP_UART_RXFIFO_OVF_INT_ST The masked interrupt status of LP_UART_RXFIFO_OVF_INT. (RO)

LP_UART_DSR_CHG_INT_ST The masked interrupt status of LP_UART_DSR_CHG_INT. (RO)

LP_UART_CTS_CHG_INT_ST The masked interrupt status of LP_UART_CTS_CHG_INT. (RO)

LP_UART_BRK_DET_INT_ST The masked interrupt status of LP_UART_BRK_DET_INT. (RO)

LP_UART_RXFIFO_TOUT_INT_ST The masked interrupt status of LP_UART_RXFIFO_TOUT_INT. (RO)

LP_UART_SW_XON_INT_ST The masked interrupt status of LP_UART_SW_XON_INT. (RO)

LP_UART_SW_XOFF_INT_ST The masked interrupt status of LP_UART_SW_XOFF_INT. (RO)

LP_UART_GLITCH_DET_INT_ST The masked interrupt status of LP_UART_GLITCH_DET_INT. (RO)

LP_UART_TX_BRK_DONE_INT_ST The masked interrupt status of LP_UART_TX_BRK_DONE_INT. (RO)

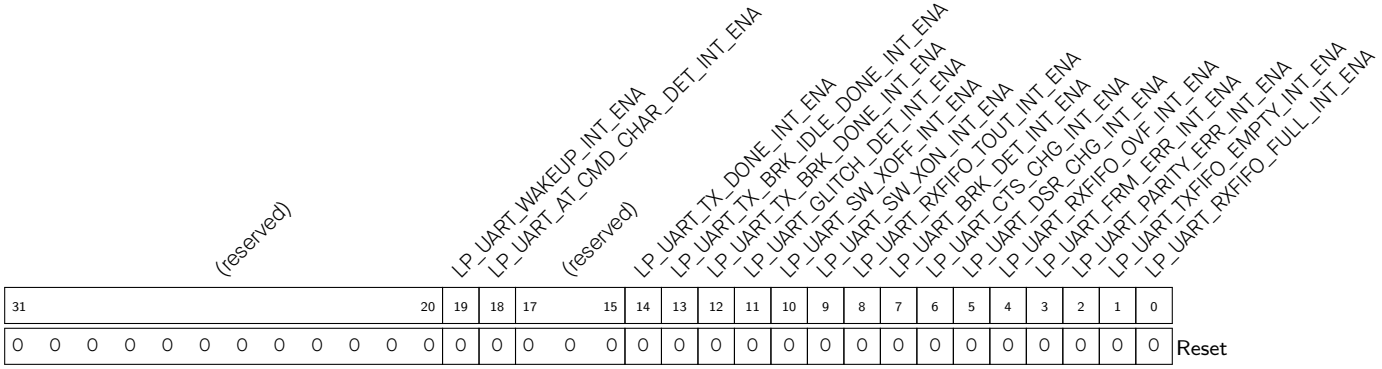
LP_UART_TX_BRK_IDLE_DONE_INT_ST The masked interrupt status of LP_UART_TX_BRK_IDLE_DONE_INT. (RO)

LP_UART_TX_DONE_INT_ST The masked interrupt status of LP_UART_TX_DONE_INT. (RO)

LP_UART_AT_CMD_CHAR_DET_INT_ST The masked interrupt status of LP_UART_AT_CMD_CHAR_DET_INT. (RO)

LP_UART_WAKEUP_INT_ST The masked interrupt status of LP_UART_WAKEUP_INT. (RO)

Register 45.42. LP_UART_INT_ENA_REG (0x000C)



LP_UART_RXFIFO_FULL_INT_ENA Write 1 to enable LP_UART_RXFIFO_FULL_INT. (R/W)

LP_UART_TXFIFO_EMPTY_INT_ENA Write 1 to enable LP_UART_TXFIFO_EMPTY_INT. (R/W)

LP_UART_PARITY_ERR_INT_ENA Write 1 to enable LP_UART_PARITY_ERR_INT. (R/W)

LP_UART_FRM_ERR_INT_ENA Write 1 to enable LP_UART_FRM_ERR_INT. (R/W)

LP_UART_RXFIFO_OVF_INT_ENA Write 1 to enable LP_UART_RXFIFO_OVF_INT. (R/W)

LP_UART_DSR_CHG_INT_ENA Write 1 to enable LP_UART_DSR_CHG_INT. (R/W)

LP_UART_CTS_CHG_INT_ENA Write 1 to enable LP_UART_CTS_CHG_INT. (R/W)

LP_UART_BRK_DET_INT_ENA Write 1 to enable LP_UART_BRK_DET_INT. (R/W)

LP_UART_RXFIFO_TOUT_INT_ENA Write 1 to enable LP_UART_RXFIFO_TOUT_INT. (R/W)

LP_UART_SW_XON_INT_ENA Write 1 to enable LP_UART_SW_XON_INT.(R/W)

LP_UART_SW_XOFF_INT_ENA Write 1 to enable LP_UART_SW_XOFF_INT. (R/W)

LP_UART_GLITCH_DET_INT_ENA Write 1 to enable LP_UART_GLITCH_DET_INT. (R/W)

LP_UART_TX_BRK_DONE_INT_ENA Write 1 to enable LP_UART_TX_BRK_DONE_INT. (R/W)

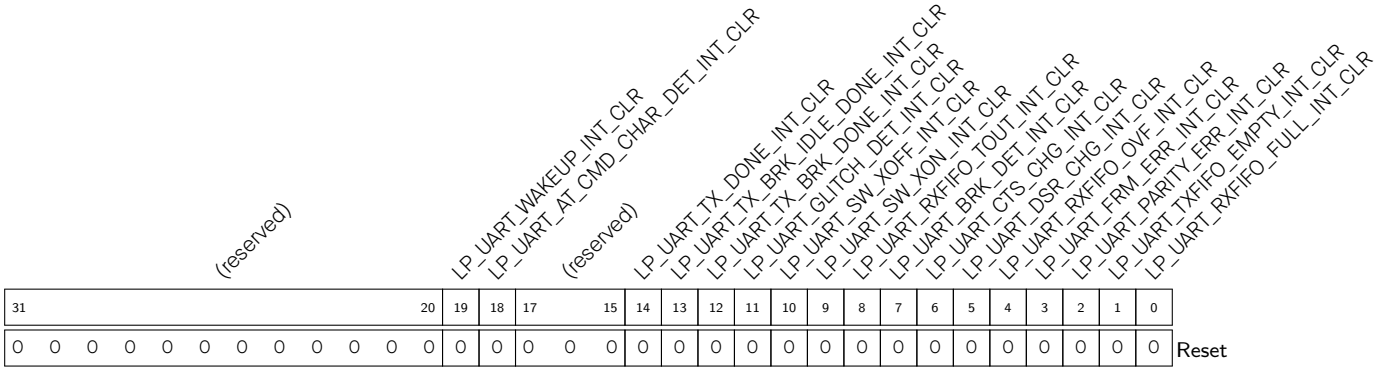
LP_UART_TX_BRK_IDLE_DONE_INT_ENA Write 1 to enable LP_UART_TX_BRK_IDLE_DONE_INT. (R/W)

LP_UART_TX_DONE_INT_ENA Write 1 to enable LP_UART_TX_DONE_INT. (R/W)

LP_UART_AT_CMD_CHAR_DET_INT_ENA Write 1 to enable LP_UART_AT_CMD_CHAR_DET_INT. (R/W)

LP_UART_WAKEUP_INT_ENA Write 1 to enable LP_UART_WAKEUP_INT. (R/W)

Register 45.43. LP_UART_INT_CLR_REG (0x0010)



LP_UART_RXFIFO_FULL_INT_CLR Write 1 to clear LP_UART_RXFIFO_FULL_INT. (WT)

LP_UART_TXFIFO_EMPTY_INT_CLR Write 1 to clear LP_UART_TXFIFO_EMPTY_INT. (WT)

LP_UART_PARITY_ERR_INT_CLR Write 1 to clear LP_UART_PARITY_ERR_INT. (WT)

LP_UART_FRM_ERR_INT_CLR Write 1 to clear LP_UART_FRM_ERR_INT. (WT)

LP_UART_RXFIFO_OVF_INT_CLR Write 1 to clear LP_UART_RXFIFO_OVF_INT. (WT)

LP_UART_DSR_CHG_INT_CLR Write 1 to clear LP_UART_DSR_CHG_INT. (WT)

LP_UART_CTS_CHG_INT_CLR Write 1 to clear LP_UART_CTS_CHG_INT. (WT)

LP_UART_BRK_DET_INT_CLR Write 1 to clear LP_UART_BRK_DET_INT. (WT)

LP_UART_RXFIFO_TOUT_INT_CLR Write 1 to clear LP_UART_RXFIFO_TOUT_INT. (WT)

LP_UART_SW_XON_INT_CLR Write 1 to clear LP_UART_SW_XON_INT. (WT)

LP_UART_SW_XOFF_INT_CLR Write 1 to clear LP_UART_SW_XOFF_INT. (WT)

LP_UART_GLITCH_DET_INT_CLR Write 1 to clear LP_UART_GLITCH_DET_INT. (WT)

LP_UART_TX_BRK_DONE_INT_CLR Write 1 to clear LP_UART_TX_BRK_DONE_INT. (WT)

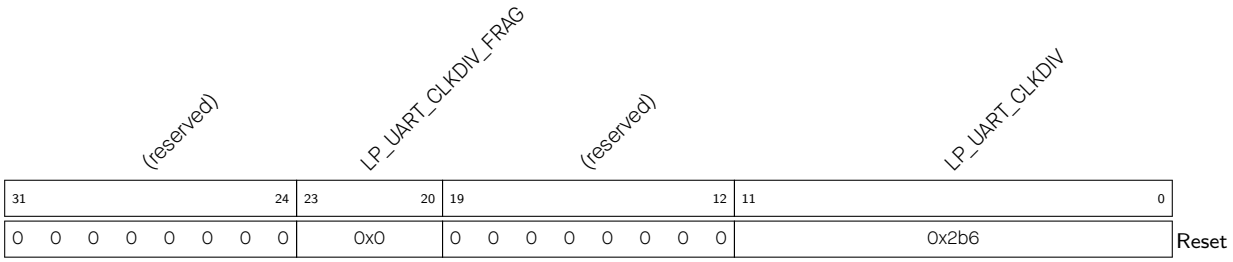
LP_UART_TX_BRK_IDLE_DONE_INT_CLR Write 1 to clear LP_UART_TX_BRK_IDLE_DONE_INT. (WT)

LP_UART_TX_DONE_INT_CLR Write 1 to clear LP_UART_TX_DONE_INT. (WT)

LP_UART_AT_CMD_CHAR_DET_INT_CLR Write 1 to clear LP_UART_AT_CMD_CHAR_DET_INT. (WT)

LP_UART_WAKEUP_INT_CLR Write 1 to clear LP_UART_WAKEUP_INT. (WT)

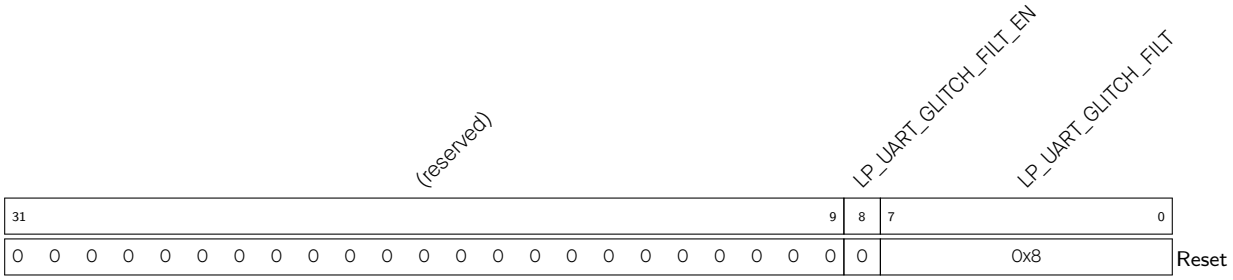
Register 45.44. LP_UART_CLKDIV_SYNC_REG (0x0014)



LP_UART_CLKDIV Configures the integral part of the divisor for baud rate generation. (R/W)

LP_UART_CLKDIV_FRAG Configures the fractional part of the divisor for baud rate generation. (R/W)

Register 45.45. LP_UART_RX_FILT_REG (0x0018)



LP_UART_GLITCH_FILT Configures the width of a pulse to be filtered.

Measurement unit: UART Core's clock cycle.

Pulses whose width is lower than this value will be ignored. (R/W)

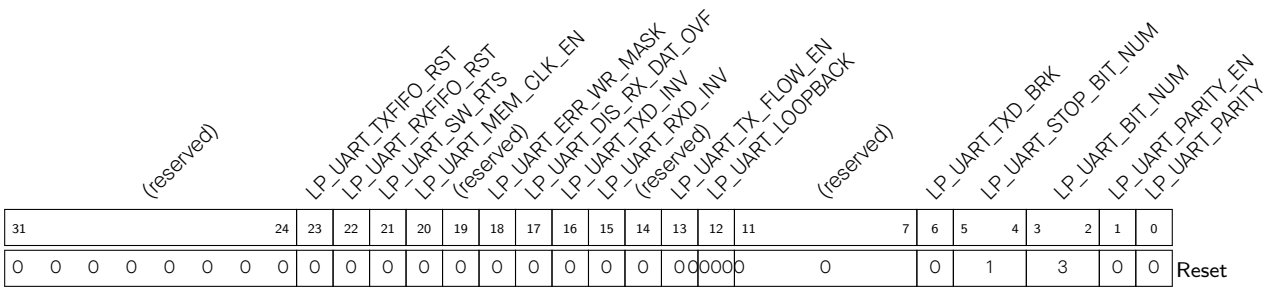
LP_UART_GLITCH_FILT_EN Configures whether or not to enable RX signal filter.

0: Disable

1: Enable

(R/W)

Register 45.46. LP_UART_CONF0_SYNC_REG (0x0020)


(reserved)
LP_UART_TXFIFO_RST
LP_UART_RXFIFO_RST
LP_UART_SW_RTS
LP_UART_MEM_CLK_EN
(reserved)
LP_UART_ERR_WR_MASK
LP_UART_DIS_RX_DAT_OVF
LP_UART_TXD_INV
LP_UART_RXD_INV
(reserved)
LP_UART_TX_FLOW_EN
LP_UART_LOOPBACK
(reserved)
LP_UART_TXD_BRK
LP_UART_STOP_BIT_NUM
LP_UART_BIT_NUM
LP_UART_PARITY_EN
LP_UART_PARITY
31 24 23 22 21 20 19 18 17 16 15 14 13 12 11 7 6 5 4 3 2 1 0
0 0 0

Register 45.46. LP_UART_CONF0_SYNC_REG (0x0020)

Continued from the previous page...

LP_UART_TX_FLOW_EN Configures whether or not to enable flow control for the transmitter.

0: Disable

1: Enable

(R/W)

LP_UART_RXD_INV Configures whether or not to invert the level of LP UART RXD signal.

0: Not invert

1: Invert

(R/W)

LP_UART_TXD_INV Configures whether or not to invert the level of LP UART TXD signal.

0: Not invert

1: Invert

(R/W)

LP_UART_DIS_RX_DAT_OVF Configures whether or not to disable data overflow detection for the LP UART receiver.

0: Enable

1: Disable

(R/W)

LP_UART_ERR_WR_MASK Configures whether or not to store the received data with errors into FIFO.

0: Store

1: Not store

(R/W)

LP_UART_MEM_CLK_EN Configures whether or not to enable clock gating for LP UART memory.

0: Disable

1: Enable

(R/W)

LP_UART_SW_RTS Configures the RTS signal used in software flow control.

0: The LP UART transmitter is not allowed to send data.

1: The LP UART transmitted is allowed to send data.

(R/W)

LP_UART_RXFIFO_RST Configures whether or not to reset the LP UART RX FIFO.

0: Not reset

1: Reset

(R/W)

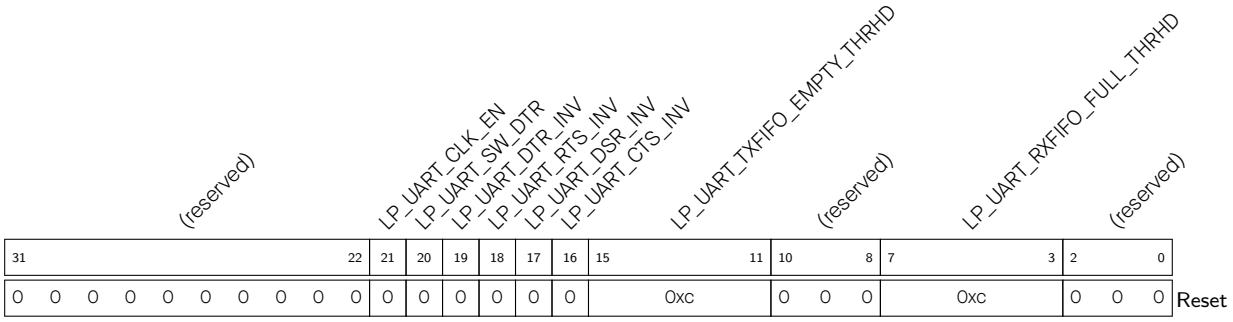
LP_UART_TXFIFO_RST Configures whether or not to reset the LP UART TX FIFO.

0: Not reset

1: Reset

(R/W)

Register 45.47. LP_UART_CONF1_REG (0x0024)



LP_UART_RXFIFO_FULL_THRHD Configures the threshold for RX FIFO being full.
Measurement unit: byte. (R/W)

LP_UART_TXFIFO_EMPTY_THRHD Configures the threshold for TX FIFO being empty.
Measurement unit: byte. (R/W)

LP_UART_CTS_INV Configures whether or not to invert the level of LP UART CTS signal.
0: Not invert
1: Invert
(R/W)

LP_UART_DSR_INV Configures whether or not to invert the level of LP UART DSR signal.
0: Not invert
1: Invert
(R/W)

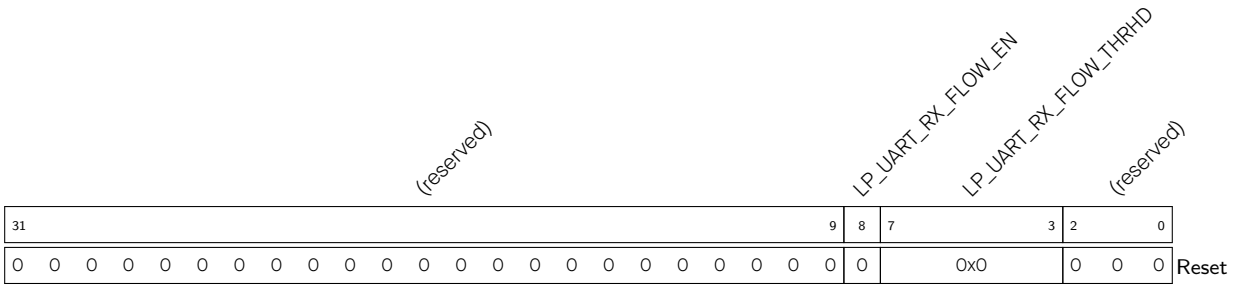
LP_UART_RTS_INV Configures whether or not to invert the level of LP UART RTS signal.
0: Not invert
1: Invert
(R/W)

LP_UART_DTR_INV Configures whether or not to invert the level of LP UART DTR signal.
0: Not invert
1: Invert
(R/W)

LP_UART_SW_DTR Configures the DTR signal used in software flow control.
0: Data to be transmitted is not ready.
1: Data to be transmitted is ready.
(R/W)

LP_UART_CLK_EN Configures clock gating.
0: Support clock only when the application writes registers.
1: Always force the clock on for registers.
(R/W)

Register 45.48. LP_UART_HWFC_CONF_SYNC_REG (0x002C)



LP_UART_RX_FLOW_THRHD Configures the maximum number of data bytes that can be received during hardware flow control.

Measurement unit: byte. (R/W)

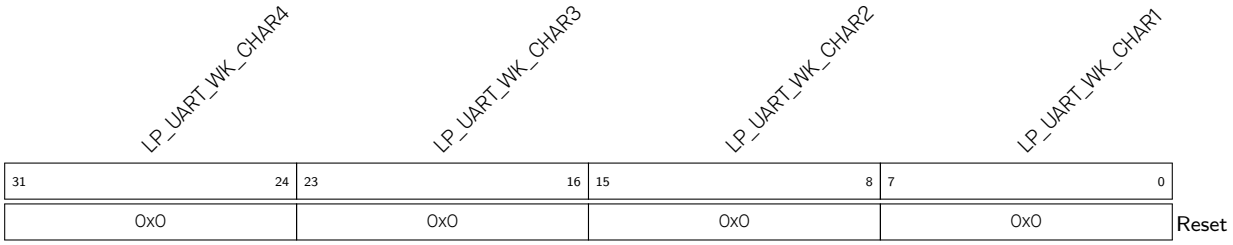
LP_UART_RX_FLOW_EN Configures whether or not to enable the LP UART receiver.

0: Disable

1: Enable

(R/W)

Register 45.49. LP_UART_SLEEP_CONF0_REG (0x0030)



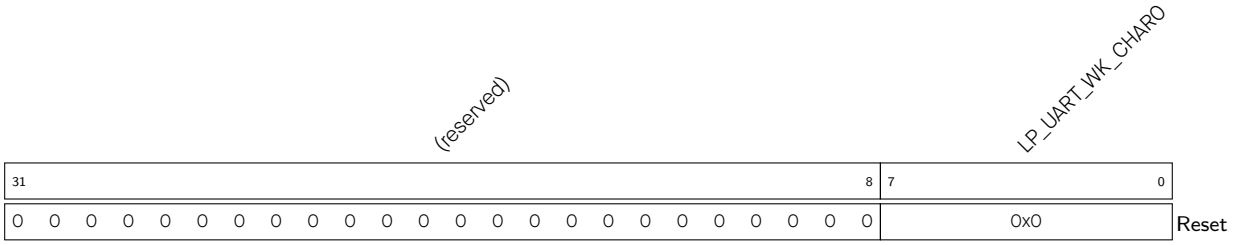
LP_UART_WK_CHAR1 Configures wakeup character 1. (R/W)

LP_UART_WK_CHAR2 Configures wakeup character 2. (R/W)

LP_UART_WK_CHAR3 Configures wakeup character 3. (R/W)

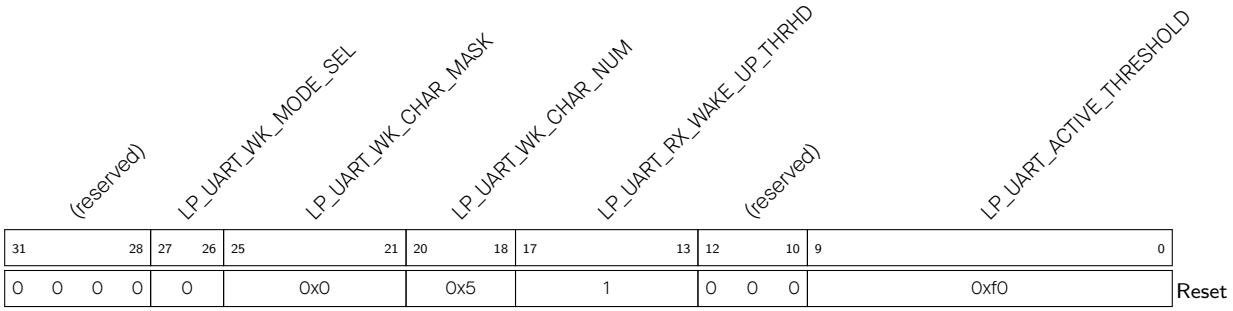
LP_UART_WK_CHAR4 Configures wakeup character 4. (R/W)

Register 45.50. LP_UART_SLEEP_CONF1_REG (0x0034)



LP_UART_WK_CHAR0 Configures wakeup character 0. (R/W)

Register 45.51. LP_UART_SLEEP_CONF2_REG (0x0038)



LP_UART_ACTIVE_THRESHOLD Configures the number of RXD edge changes to wake up the chip in wakeup mode 0. (R/W)

LP_UART_RX_WAKE_UP_THRHD Configures the number of received data bytes to wake up the chip in wakeup mode 1. (R/W)

LP_UART_WK_CHAR_NUM Configures the number of wakeup characters. (R/W)

LP_UART_WK_CHAR_MASK Configures whether or not to mask wakeup characters.

0: Not mask

1: Mask

(R/W)

LP_UART_WK_MODE_SEL Configures which wakeup mode to select.

0: Mode 0

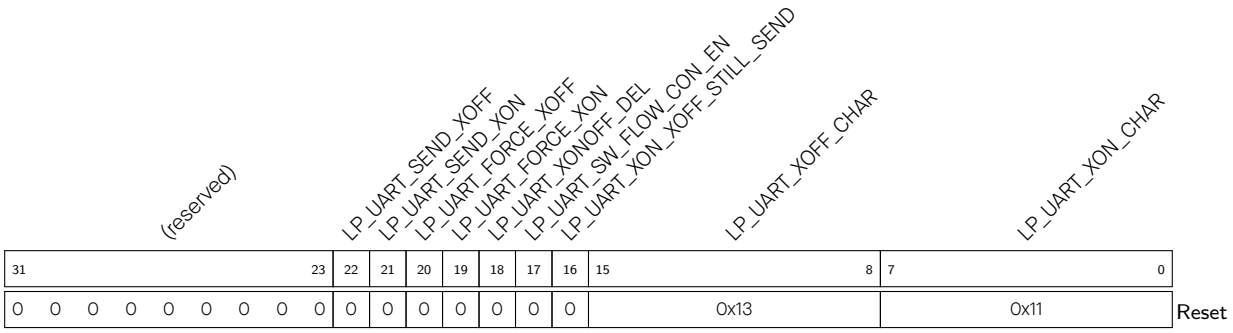
1: Mode 1

2: Mode 2

3: Mode 3

(R/W)

Register 45.52. LP_UART_SWFC_CONF0_SYNC_REG (0x003C)



LP_UART_XON_CHAR Configures the XON character for flow control. (R/W)

LP_UART_XOFF_CHAR Configures the XOFF character for flow control. (R/W)

LP_UART_XON_XOFF_STILL_SEND Configures whether the LP UART transmitter can send XON or XOFF characters when it is disabled.

0: Cannot send

1: Can send

(R/W)

LP_UART_SW_FLOW_CON_EN Configures whether or not to enable software flow control.

0: Disable

1: Enable

(R/W)

LP_UART_XONOFF_DEL Configures whether or not to remove flow control characters from the received data.

0: Not move

1: Move

(R/W)

LP_UART_FORCE_XON Configures whether the transmitter continues to sending data.

0: Not send

1: Send

(R/W)

LP_UART_FORCE_XOFF Configures whether or not to stop the transmitter from sending data.

0: Not stop

1: Stop

(R/W)

LP_UART_SEND_XON Configures whether or not to send XON characters.

0: Not send

1: Send

(R/W/SS/SC)

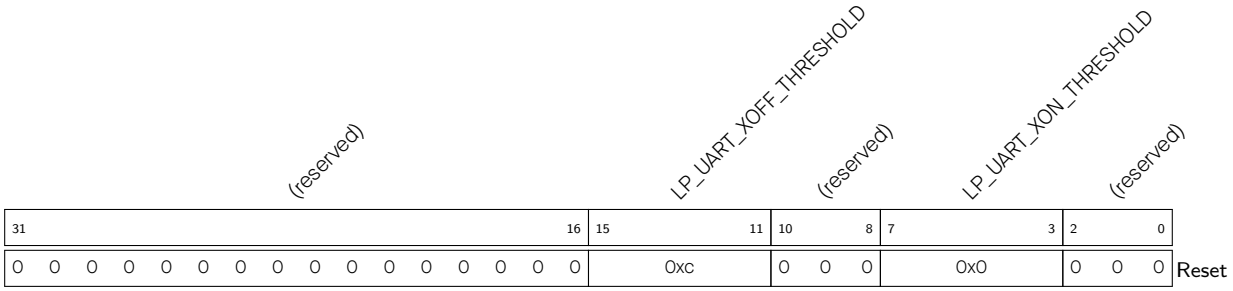
LP_UART_SEND_XOFF Configures whether or not to send XOFF characters.

0: Not send

1: Send

(R/W/SS/SC)

Register 45.53. LP_UART_SWFC_CONF1_REG (0x0040)



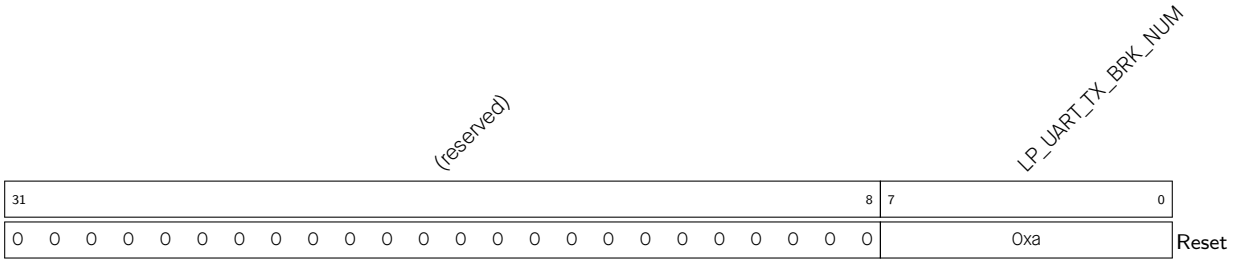
LP_UART_XON_THRESHOLD Configures the threshold for data in RX FIFO to send XON characters in software flow control.

Measurement unit: byte. (R/W)

LP_UART_XOFF_THRESHOLD Configures the threshold for data in RX FIFO to send XOFF characters in software flow control.

Measurement unit: byte. (R/W)

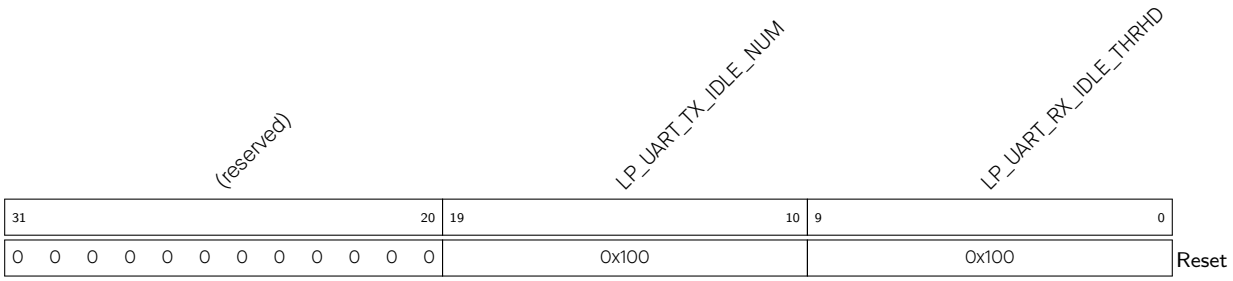
Register 45.54. LP_UART_TXBRK_CONF_SYNC_REG (0x0044)



LP_UART_TX_BRK_NUM Configures the number of NULL characters to be sent after finishing data transmission.

Valid only when LP_UART_TXD_BRK is 1. (R/W)

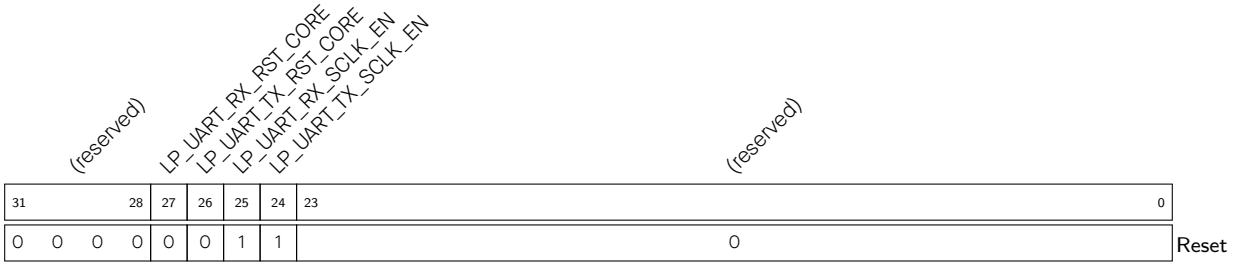
Register 45.55. LP_UART_IDLE_CONF_SYNC_REG (0x0048)



LP_UART_RX_IDLE_THRHD Configures the threshold to generate a frame end signal when the receiver takes more time to receive one data byte data.
Measurement unit: bit time (the time to transmit 1 bit). (R/W)

LP_UART_TX_IDLE_NUM Configures the interval between two data transfers.
Measurement unit: bit time (the time to transmit 1 bit). (R/W)

Register 45.56. LP_UART_CLK_CONF_REG (0x0088)



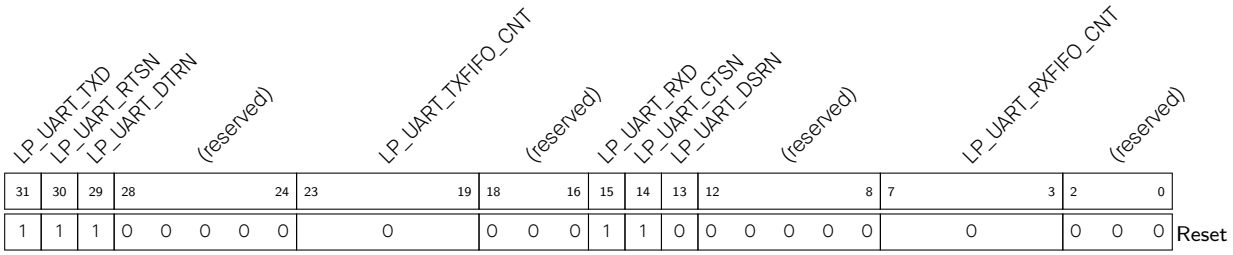
LP_UART_TX_SCLK_EN Configures whether or not to enable LP UART TX clock.
0: Disable
1: Enable
(R/W)

LP_UART_RX_SCLK_EN Configures whether or not to enable LP UART RX clock.
0: Disable
1: Enable
(R/W)

LP_UART_TX_RST_CORE Write 1 and then write 0 to reset LP UART TX. (R/W)

LP_UART_RX_RST_CORE Write 1 and then write 0 to reset LP UART RX. (R/W)

Register 45.57. LP_UART_STATUS_REG (0x001C)



LP_UART_RXFIFO_CNT Represents the number of valid data bytes in RX FIFO. (RO)

LP_UART_DSRN Represents the level of the internal LP UART DSR signal. (RO)

LP_UART_CTSN Represents the level of the internal LP UART CTS signal. (RO)

LP_UART_RXD Represents the level of the internal LP UART RXD signal. (RO)

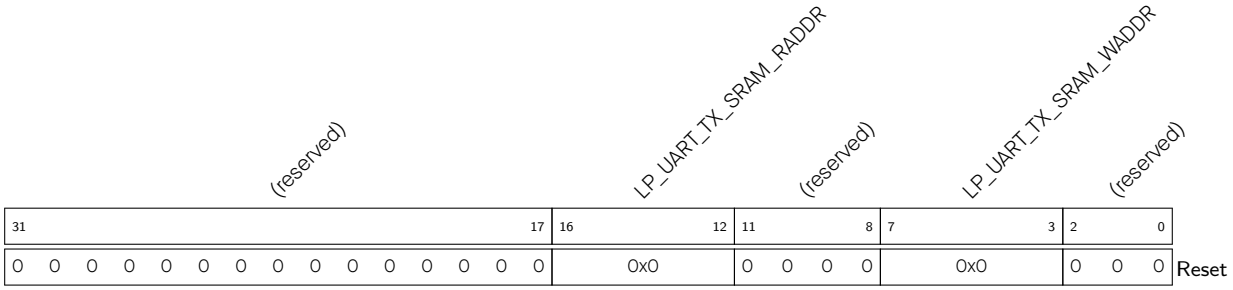
LP_UART_TXFIFO_CNT Represents the number of valid data bytes in RX FIFO. (RO)

LP_UART_DTRN Represents the level of the internal LP UART DTR signal. (RO)

LP_UART_RTSN Represents the level of the internal LP UART RTS signal. (RO)

LP_UART_TXD Represents the level of the internal LP UART TXD signal. (RO)

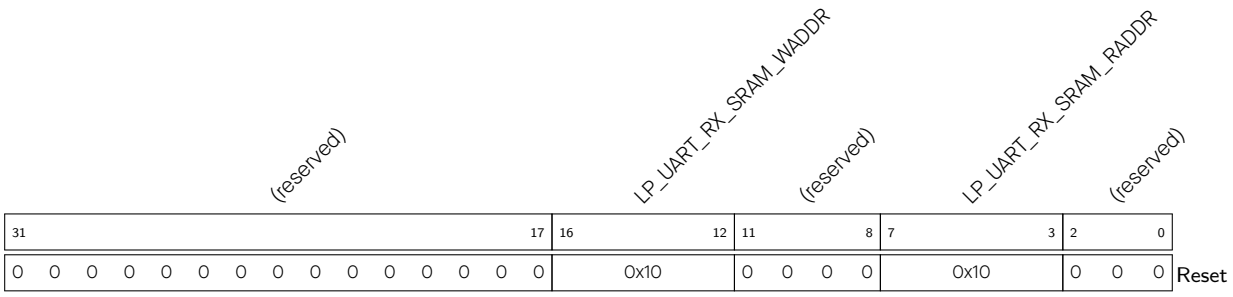
Register 45.58. LP_UART_MEM_TX_STATUS_REG (0x0068)



LP_UART_TX_SRAM_WADDR Represents the offset address to write TX FIFO. (RO)

LP_UART_TX_SRAM_RADDR Represents the offset address to read TX FIFO. (RO)

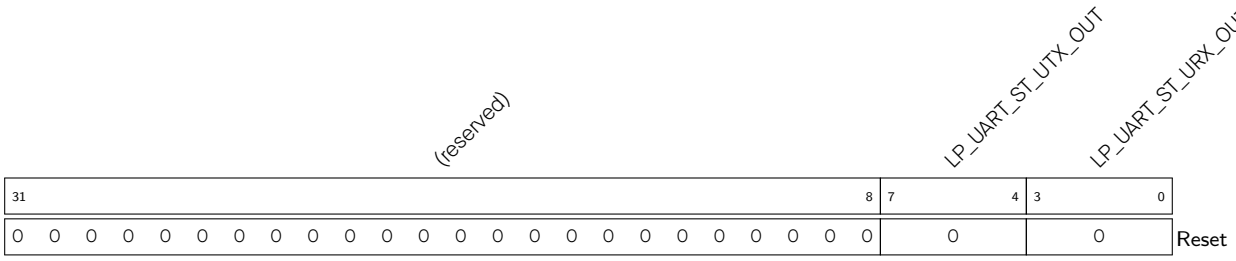
Register 45.59. LP_UART_MEM_RX_STATUS_REG (0x006C)



LP_UART_RX_SRAM_RADDR Represents the offset address to read RX FIFO. (RO)

LP_UART_RX_SRAM_WADDR Represents the offset address to write RX FIFO. (RO)

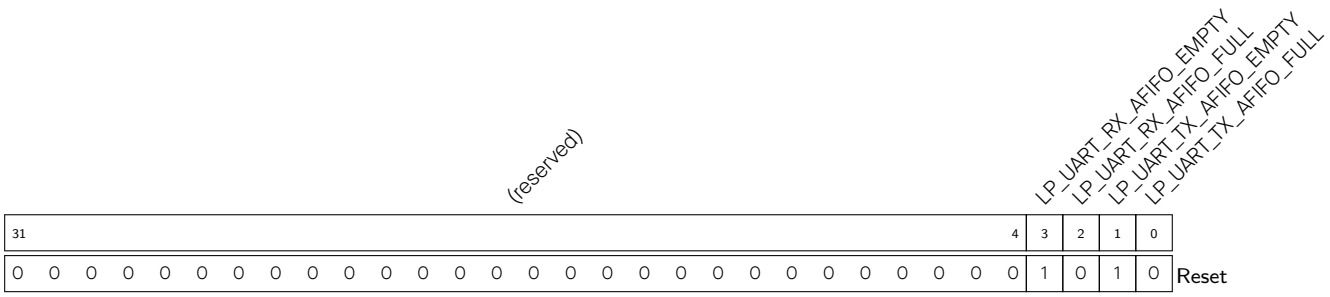
Register 45.60. LP_UART_FSM_STATUS_REG (0x0070)



LP_UART_ST_URX_OUT Represents the status of the receiver. (RO)

LP_UART_ST_UTX_OUT Represents the status of the transmitter. (RO)

Register 45.61. LP_UART_AFIFO_STATUS_REG (0x0090)



LP_UART_TX_AFIFO_FULL Represents whether or not the APB TX asynchronous FIFO is full.

0: Not full

1: Full

(RO)

LP_UART_TX_AFIFO_EMPTY Represents whether or not the APB TX asynchronous FIFO is empty.

0: Not empty

1: Empty

(RO)

LP_UART_RX_AFIFO_FULL Represents whether or not the APB RX asynchronous FIFO is full.

0: Not full

1: Full

(RO)

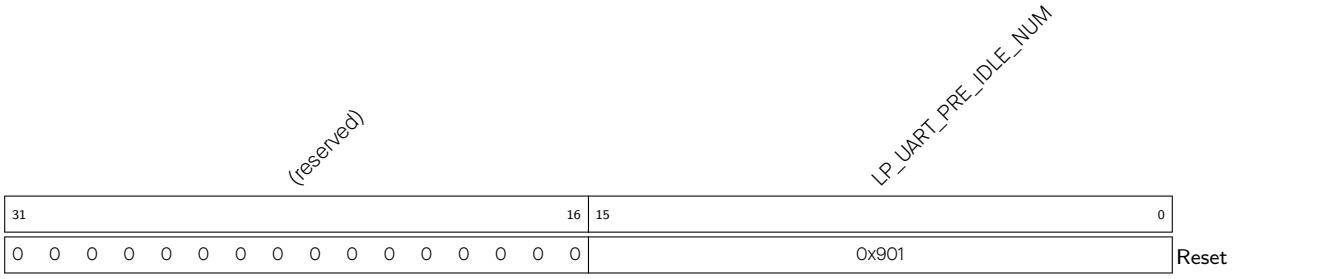
LP_UART_RX_AFIFO_EMPTY Represents whether or not the APB RX asynchronous FIFO is empty.

0: Not empty

1: Empty

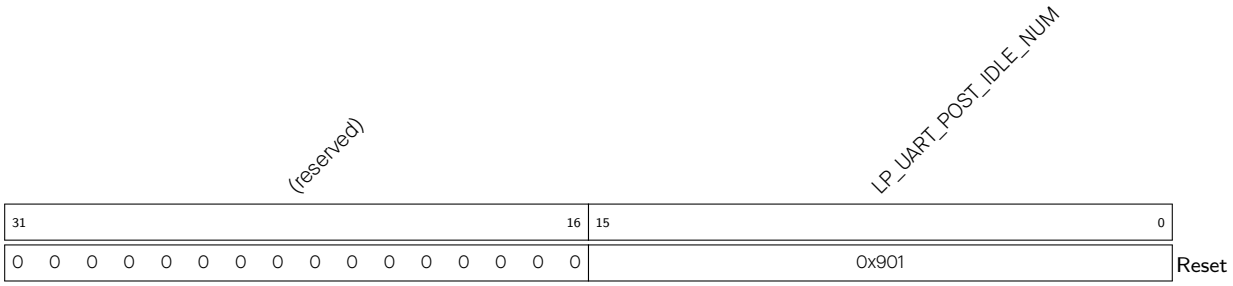
(RO)

Register 45.62. LP_UART_AT_CMD_PRECNT_SYNC_REG (0x0050)



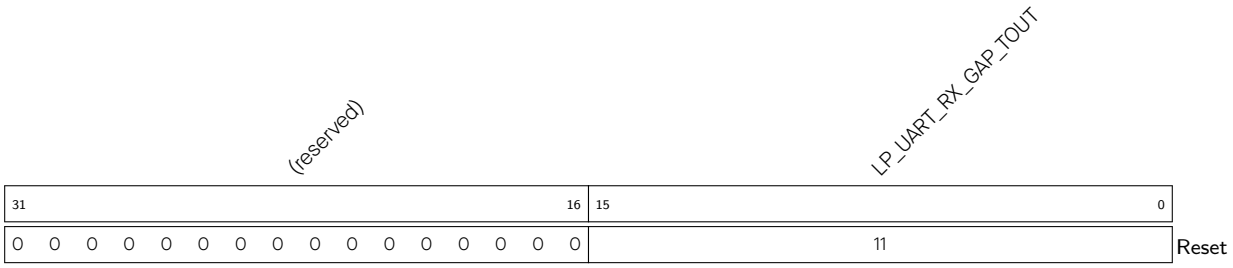
LP_UART_PRE_IDLE_NUM Configures the idle time before the receiver receives the first AT_CMD. Measurement unit: bit time (the time to transmit 1 bit). (R/W)

Register 45.63. LP_UART_AT_CMD_POSTCNT_SYNC_REG (0x0054)



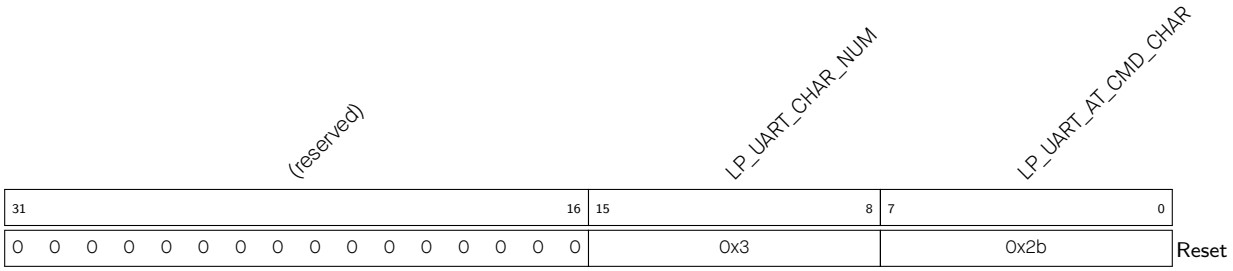
LP_UART_POST_IDLE_NUM Configures the interval between the last AT_CMD and subsequent data. Measurement unit: bit time (the time to transmit 1 bit). (R/W)

Register 45.64. LP_UART_AT_CMD_GAPTOUT_SYNC_REG (0x0058)



LP_UART_RX_GAP_TOUT Configures the interval between two AT_CMD characters. Measurement unit: bit time (the time to transmit 1 bit). (R/W)

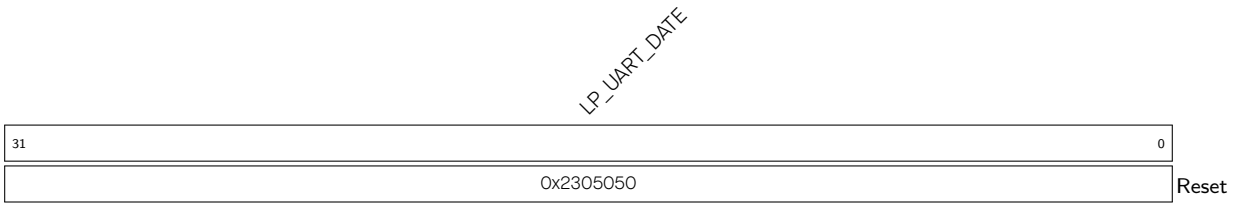
Register 45.65. LP_UART_AT_CMD_CHAR_SYNC_REG (0x005C)



LP_UART_AT_CMD_CHAR Configures the AT_CMD character. (R/W)

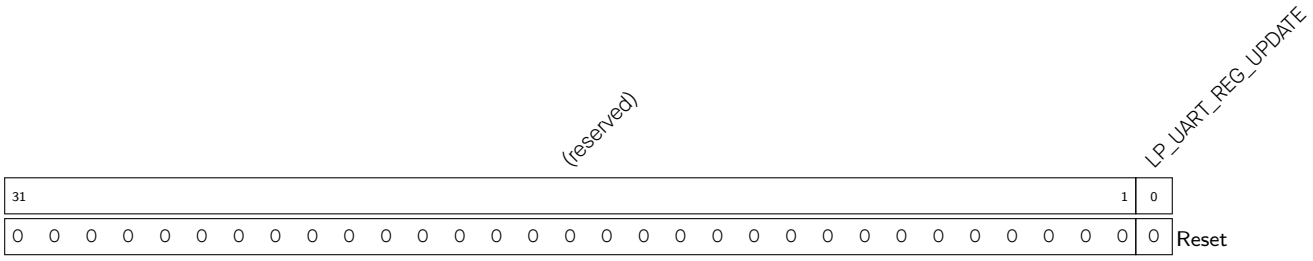
LP_UART_CHAR_NUM Configures the number of continuous AT_CMD characters a receiver can receive. (R/W)

Register 45.66. LP_UART_DATE_REG (0x008C)



LP_UART_DATE Version control register. (R/W)

Register 45.67. LP_UART_REG_UPDATE_REG (0x0098)



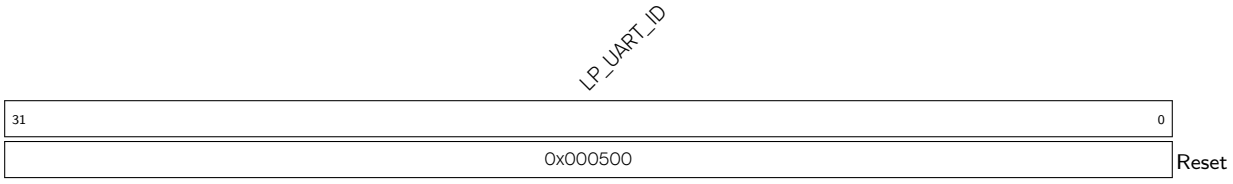
LP_UART_REG_UPDATE Configures whether or not to synchronize registers.

0: Not synchronize

1: Synchronize

(R/W/SC)

Register 45.68. LP_UART_ID_REG (0x009C)



LP_UART_ID Configures the LP UART ID. (R/W)

45.7.3 UHCI Registers

The addresses in this section are relative to UHCI base address provided in Table 9.3-2 in Chapter 9 *System and Memory*.

For how to program reserved fields, please refer to Section *Programming Reserved Register Field*.

Register 45.69. UHCI_CONF0_REG (0x0000)



UHCI_TX_RST Write 1 and then write 0 to reset the decoder state machine. (R/W)

UHCI_RX_RST Write 1 and then write 0 to reset the encoder state machine. (R/W)

UHCI_UART0_CE Select one UART from UART 0 ~ 4 to connect with UHCI.

0: Select UART0
1: Select UART1
2: Select UART2
3: Select UART3
4: Select UART4
5 ~ 7: No effect, as no UART will connect with UHCI
(R/W)

UHCI_UART1_CE Configures whether or not to connect UHCI with UART1.

0: Not connect
1: Connect
(R/W)

UHCI_SEPER_EN Configures whether or not to separate the data frame with a special character.

0: Not separate
1: Separate
(R/W)

UHCI_HEAD_EN Configures whether or not to encode the data packet with a formatting header.

0: Not use formatting header
1: Use formatting header
(R/W)

Continued on the next page...

Register 45.69. UHCI_CONF0_REG (0x0000)

Continued from the previous page...

UHCI_CRC_REC_EN Configures whether or not to enable the reception of the 16-bit CRC.

0: Disable

1: Enable

(R/W)

UHCI_UART_IDLE_EOF_EN Configures whether or not to stop receiving data when UART is idle.

0: Not stop

1: Stop

(R/W)

UHCI_LEN_EOF_EN Configures when the UHCI decoder stops receiving data.

0: Stops after receiving 0xC0

1: Stops when the number of received data bytes reach the specified value. When UHCI_HEAD_EN is 1, the specified value is the data length indicated by the UHCI packet header; when UHCI_HEAD_EN is 0, the specified value is the configured value.

(R/W)

UHCI_ENCODE_CRC_EN Configures whether or not to enable data integrity check by appending a 16 bit CCITT-CRC to the end of the data.

0: Disable

1: Enable

(R/W)

UHCI_CLK_EN Configures clock gating.

0: Support clock only when the application writes registers.

1: Always force the clock on for registers.

(R/W)

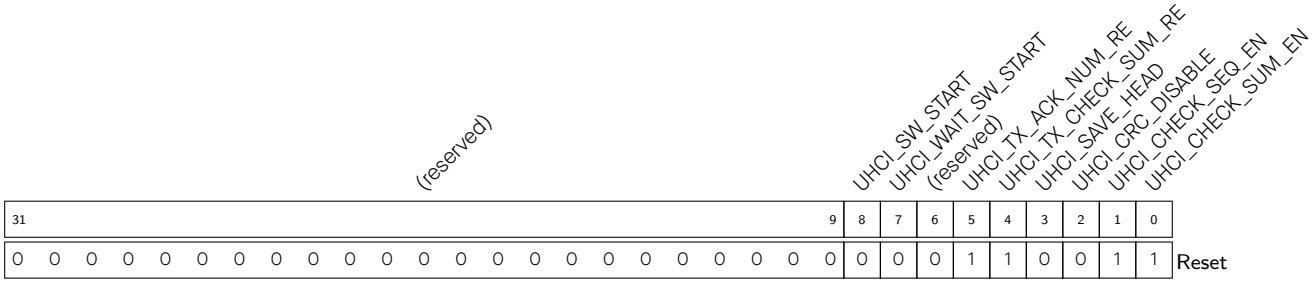
UHCI_UART_RX_BRK_EOF_EN Configures whether or not to stop UHCI from receiving data after UART has received a NULL frame.

0: Not stop

1: Stop

(R/W)

Register 45.70. UHCI_CONF1_REG (0x0014)



UHCI_CHECK_SUM_EN Configures whether or not to enable header checksum validation when UHCI receives a data packet.

0: Disable

1: Enable

(R/W)

UHCI_CHECK_SEQ_EN Configures whether or not to enable the sequence number check when UHCI receives a data packet.

0: Disable

1: Enable

(R/W)

UHCI_CRC_DISABLE Configures whether or not to enable CRC calculation.

0: Disable

1: Enable

Valid only when the Data Integrity Check Present bit in UHCI packet is 1.

(R/W)

UHCI_SAVE_HEAD Configures whether or not to save the packet header when UHCI receives a data packet.

0: Not save

1: Save

(R/W)

UHCI_TX_CHECK_SUM_RE Configures whether or not to encode the data packet with a checksum.

0: Not use checksum

1: Use checksum

(R/W)

UHCI_TX_ACK_NUM_RE Configures whether or not to encode the data packet with an acknowledgment when a reliable packet is to be transmitted.

0: Not use acknowledgement

1: Use acknowledgement

(R/W)

Continued on the next page...

Register 45.70. UHCI_CONF1_REG (0x0014)

Continued from the previous page...

UHCI_WAIT_SW_START Configures whether or not to put the UHCI encoder state machine to ST_SW_WAIT state.

0: No

1: Yes

(R/W)

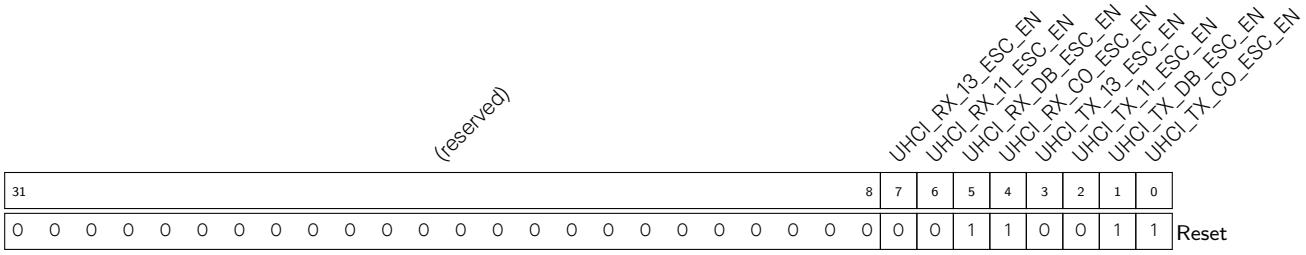
UHCI_SW_START Configures whether or not to send data packets when the encoder state machine is in ST_SW_WAIT state.

0: Not send

1: Send

(R/W/SC)

Register 45.71. UHCI_ESCAPE_CONF_REG (0x0020)



UHCI_TX_C0_ESC_EN Configures whether or not to decode character 0xC0 when DMA receives data.

0: Not decode

1: Decode

(R/W)

UHCI_TX_DB_ESC_EN Configures whether or not to decode character 0xDB when DMA receives data.

0: Not decode

1: Decode

(R/W)

UHCI_TX_11_ESC_EN Configures whether or not to decode flow control character 0x11 when DMA receives data.

0: Not decode

1: Decode

(R/W)

UHCI_TX_13_ESC_EN Configures whether or not to decode flow control character 0x13 when DMA receives data.

0: Not decode

1: Decode

(R/W)

UHCI_RX_C0_ESC_EN Configures whether or not to replace 0xC0 by special characters when DMA sends data.

0: Not replace

1: Replace

(R/W)

UHCI_RX_DB_ESC_EN Configures whether or not to replace 0xDB by special characters when DMA sends data.

0: Not replace

1: Replace

(R/W)

Continued on the next page...

Register 45.71. UHCI_ESCAPE_CONF_REG (0x0020)

Continued from the previous page...

UHCI_RX_11_ESC_EN Configures whether or not to replace flow control character 0x11 by special characters when DMA sends data.

0: Not replace

1: Replace

(R/W)

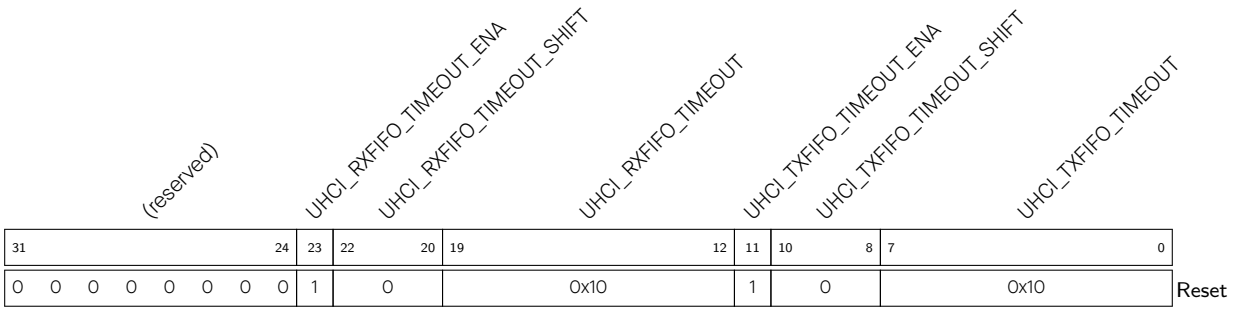
UHCI_RX_13_ESC_EN Configures whether or not to replace flow control character 0x13 by special characters when DMA sends data.

0: Not replace

1: Replace

(R/W)

Register 45.72. UHCI_HUNG_CONF_REG (0x0024)



UHCI_TXFIFO_TIMEOUT Configures the timeout value for DMA data reception.
Measurement unit: ms. (R/W)

UHCI_TXFIFO_TIMEOUT_SHIFT Configures the upper limit of the timeout counter for TX FIFO. (R/W)

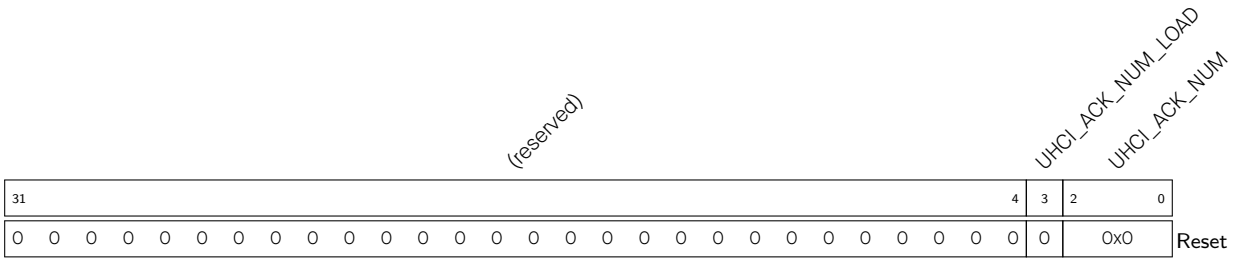
UHCI_TXFIFO_TIMEOUT_ENA Configures whether or not to enable the data reception timeout for TX FIFO.
0: Disable
1: Enable
(R/W)

UHCI_RXFIFO_TIMEOUT Configures the timeout value for DMA to read data from RAM.
Measurement unit: ms. (R/W)

UHCI_RXFIFO_TIMEOUT_SHIFT Configures the upper limit of the timeout counter for RX FIFO. (R/W)

UHCI_RXFIFO_TIMEOUT_ENA Configures whether or not to enable the DMA data transmission timeout.
0: Disable
1: Enable
(R/W)

Register 45.73. UHCI_ACK_NUM_REG (0x0028)



UHCI_ACK_NUM Configures the number of acknowledgements used in software flow control. (R/W)

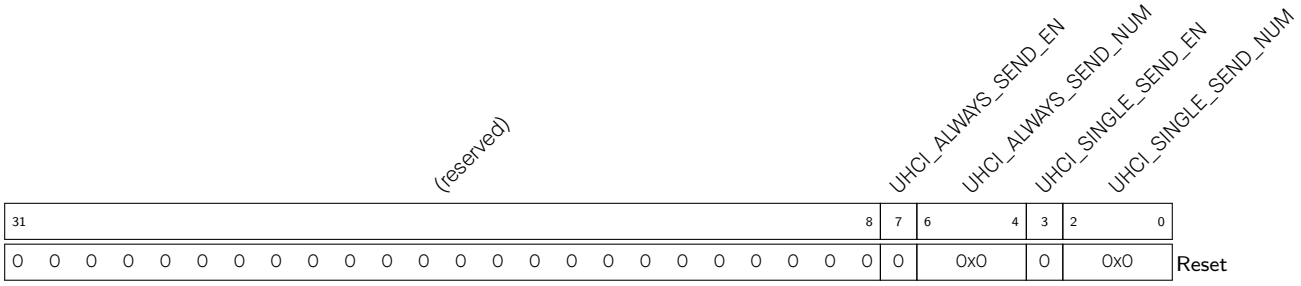
UHCI_ACK_NUM_LOAD Configures whether or not load acknowledgements.

0: Not load

1: Load

(WT)

Register 45.74. UHCI_QUICK_SENT_REG (0x0030)



UHCI_SINGLE_SEND_NUM Configures the source of data to be transmitted in single_send mode.

0: Q0 register

1: Q1 register

2: Q2 register

3: Q3 register

4: Q4 register

5: Q5 register

6: Q6 register

7: Invalid. No effect

(R/W)

UHCI_SINGLE_SEND_EN Configures whether or not to enable single_send mode.

0: Disable

1: Enable

(R/W/SC)

UHCI_ALWAYS_SEND_NUM Configures the source of data to be transmitted in always_send mode.

0: Q0 register

1: Q1 register

2: Q2 register

3: Q3 register

4: Q4 register

5: Q5 register

6: Q6 register

7: Invalid. No effect

(R/W)

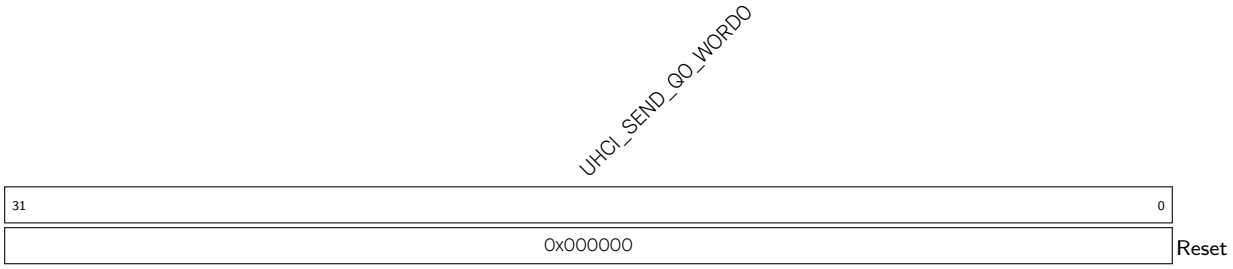
UHCI_ALWAYS_SEND_EN Configures whether or not to enable always_send mode.

0: Disable

1: Enable

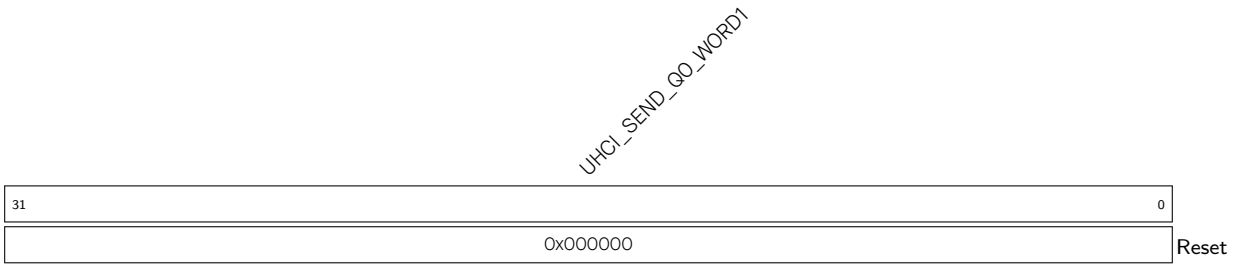
(R/W)

Register 45.75. UHCI_REG_Q0_WORD0_REG (0x0034)



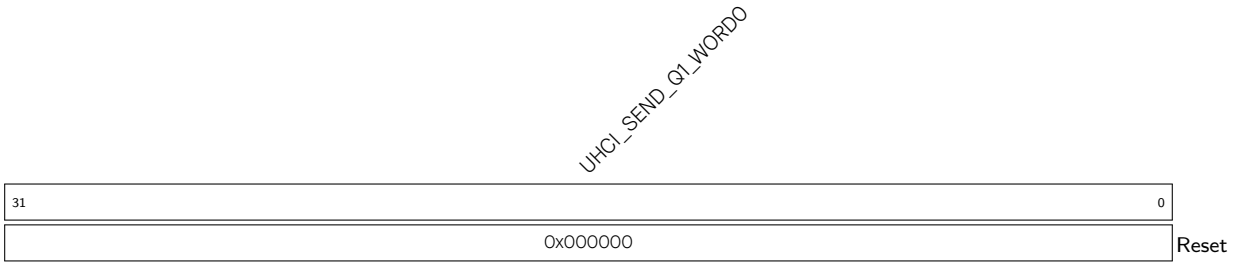
UHCI_SEND_Q0_WORD0 Data to be transmitted in Q0 register. (R/W)

Register 45.76. UHCI_REG_Q0_WORD1_REG (0x0038)



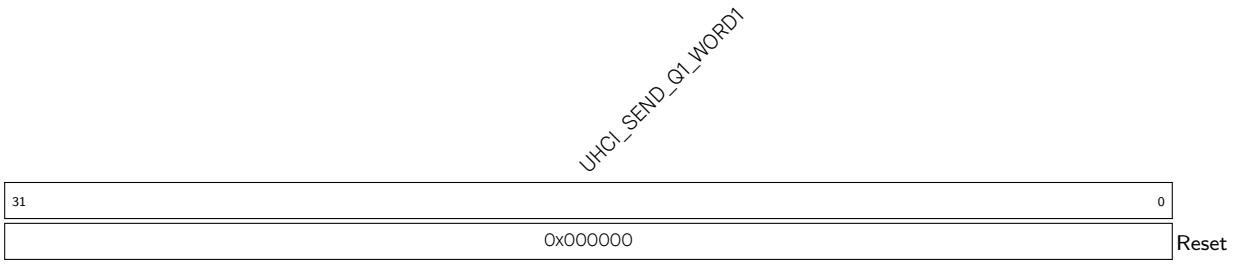
UHCI_SEND_Q0_WORD1 Data to be transmitted in Q0 register. (R/W)

Register 45.77. UHCI_REG_Q1_WORD0_REG (0x003C)



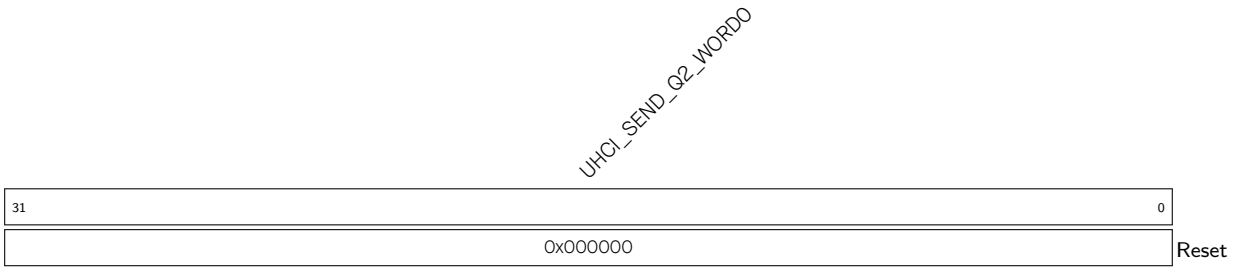
UHCI_SEND_Q1_WORD0 Data to be transmitted in Q1 register. (R/W)

Register 45.78. UHCI_REG_Q1_WORD1_REG (0x0040)



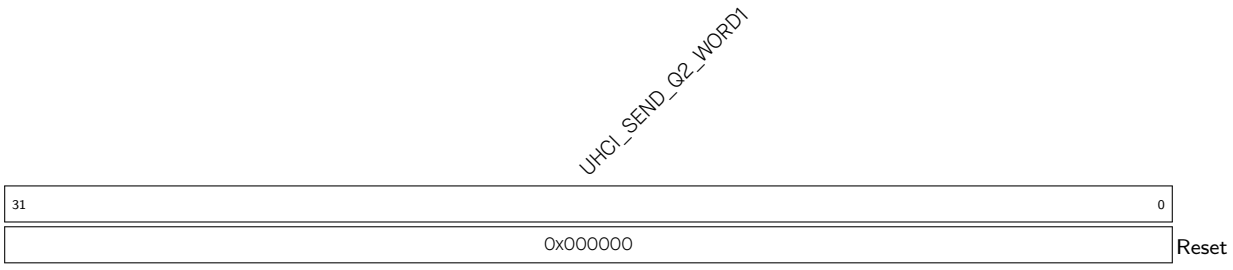
UHCI_SEND_Q1_WORD1 Data to be transmitted in Q1 register. (R/W)

Register 45.79. UHCI_REG_Q2_WORD0_REG (0x0044)



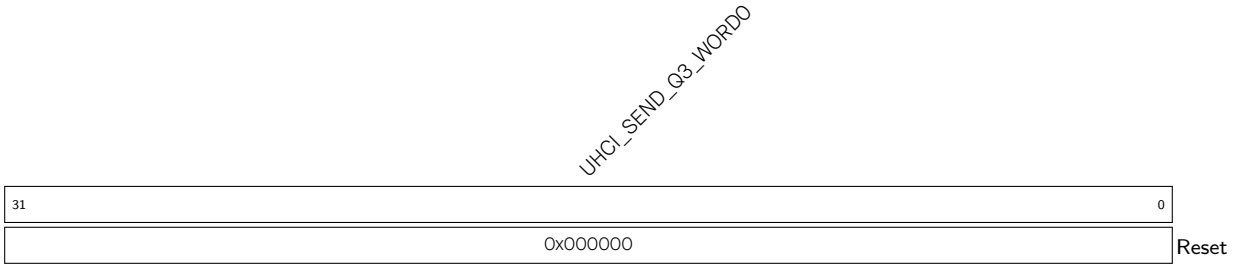
UHCI_SEND_Q2_WORD0 Data to be transmitted in Q2 register. (R/W)

Register 45.80. UHCI_REG_Q2_WORD1_REG (0x0048)



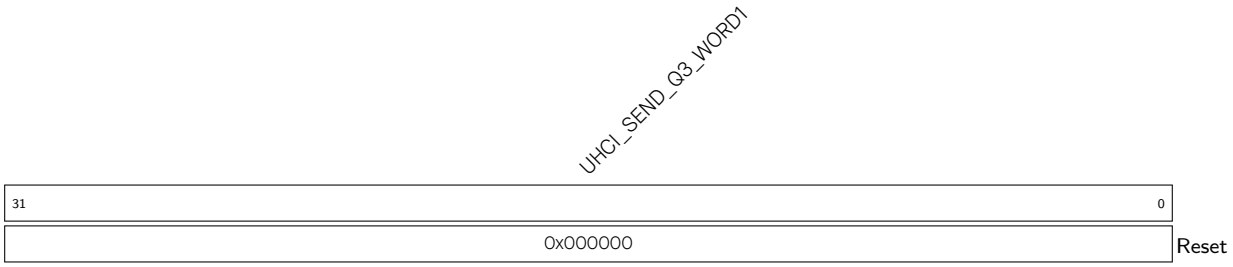
UHCI_SEND_Q2_WORD1 Data to be transmitted in Q2 register. (R/W)

Register 45.81. UHCI_REG_Q3_WORD0_REG (0x004C)



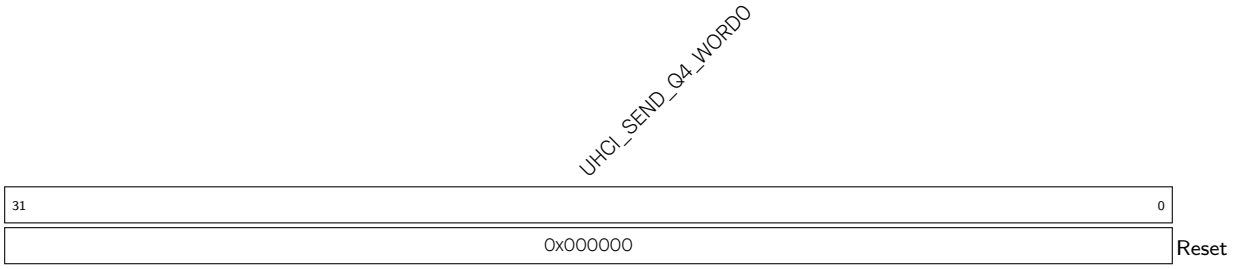
UHCI_SEND_Q3_WORD0 Data to be transmitted in Q3 register. (R/W)

Register 45.82. UHCI_REG_Q3_WORD1_REG (0x0050)



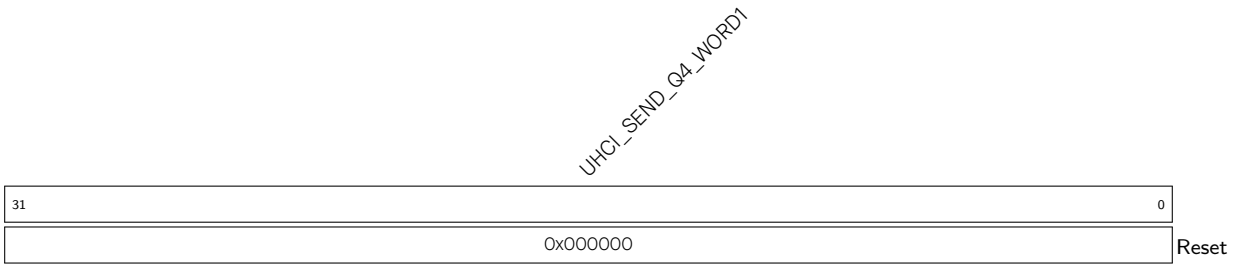
UHCI_SEND_Q3_WORD1 Data to be transmitted in Q3 register. (R/W)

Register 45.83. UHCI_REG_Q4_WORD0_REG (0x0054)



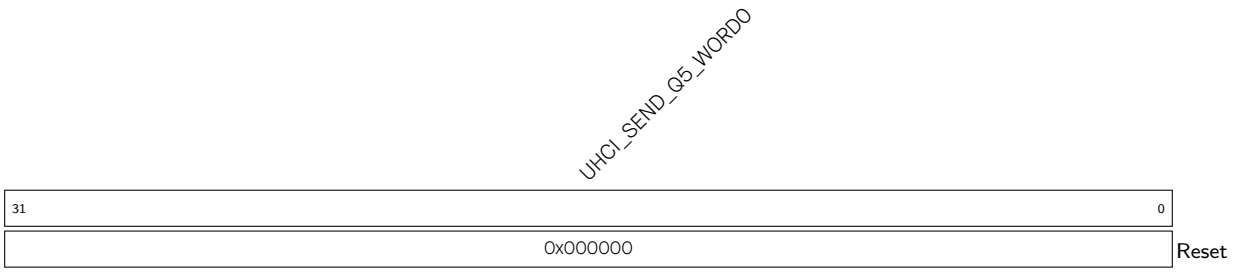
UHCI_SEND_Q4_WORD0 Data to be transmitted in Q4 register. (R/W)

Register 45.84. UHCI_REG_Q4_WORD1_REG (0x0058)



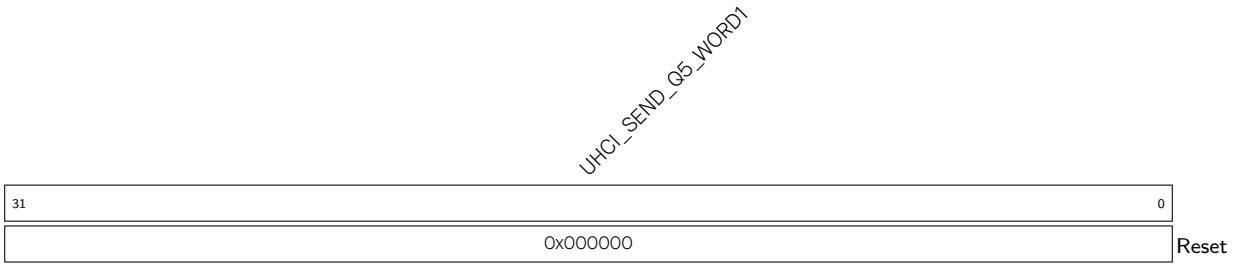
UHCI_SEND_Q4_WORD1 Data to be transmitted in Q4 register. (R/W)

Register 45.85. UHCI_REG_Q5_WORD0_REG (0x005C)



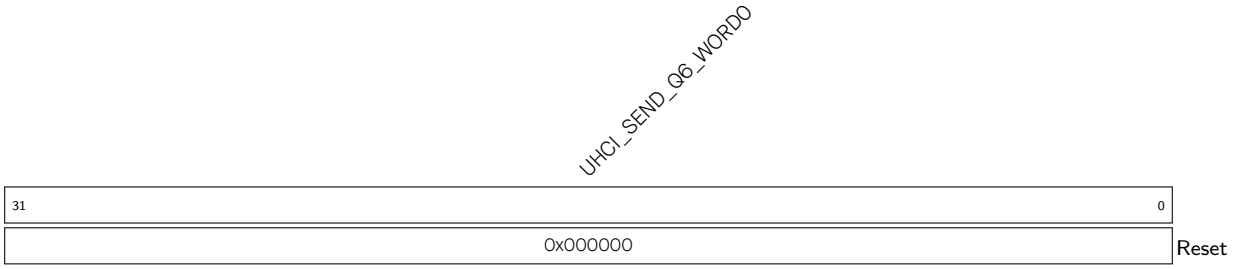
UHCI_SEND_Q5_WORD0 Data to be transmitted in Q5 register. (R/W)

Register 45.86. UHCI_REG_Q5_WORD1_REG (0x0060)



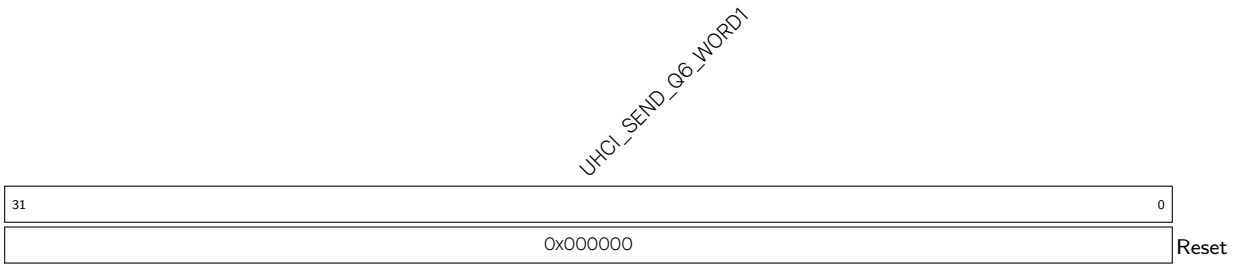
UHCI_SEND_Q5_WORD1 Data to be transmitted in Q5 register. (R/W)

Register 45.87. UHCI_REG_Q6_WORD0_REG (0x0064)



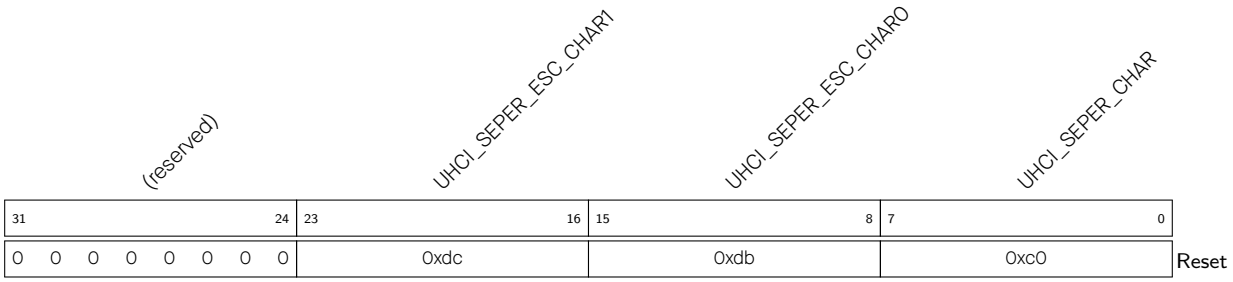
UHCI_SEND_Q6_WORD0 Data to be transmitted in Q6 register. (R/W)

Register 45.88. UHCI_REG_Q6_WORD1_REG (0x0068)



UHCI_SEND_Q6_WORD1 Data to be transmitted in Q6 register. (R/W)

Register 45.89. UHCI_ESC_CONF0_REG (0x006C)

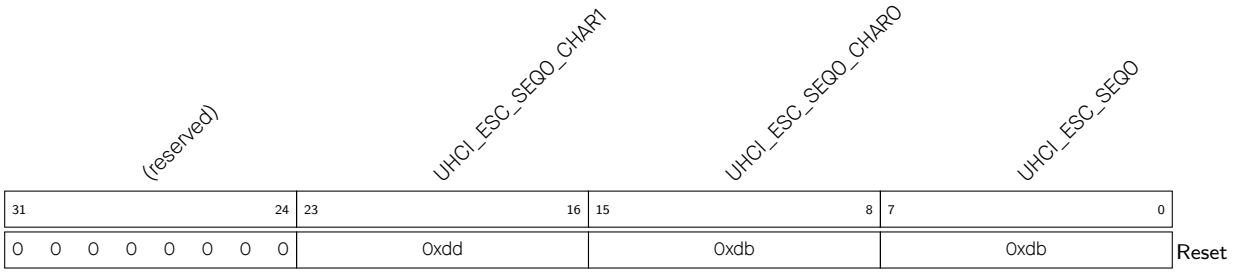


UHCI_SEPER_CHAR Configures separators to encode data packets. The default value is 0xC0. (R/W)

UHCI_SEPER_ESC_CHAR0 Configures the first character of SLIP escape sequence. The default value is 0xDB. (R/W)

UHCI_SEPER_ESC_CHAR1 Configures the second character of SLIP escape sequence. The default value is 0xDC. (R/W)

Register 45.90. UHCI_ESC_CONF1_REG (0x0070)

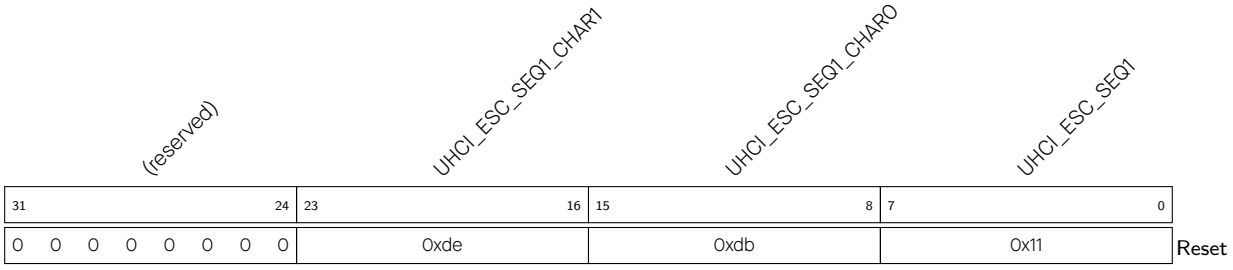


UHCI_ESC_SEQ0 Configures the character that needs to be encoded. The default value is 0xDB used as the first character of SLIP escape sequence. (R/W)

UHCI_ESC_SEQ0_CHAR0 Configures the first character of SLIP escape sequence. The default value is 0xDB. (R/W)

UHCI_ESC_SEQ0_CHAR1 Configures the second character of SLIP escape sequence. The default value is 0xDD. (R/W)

Register 45.91. UHCI_ESC_CONF2_REG (0x0074)

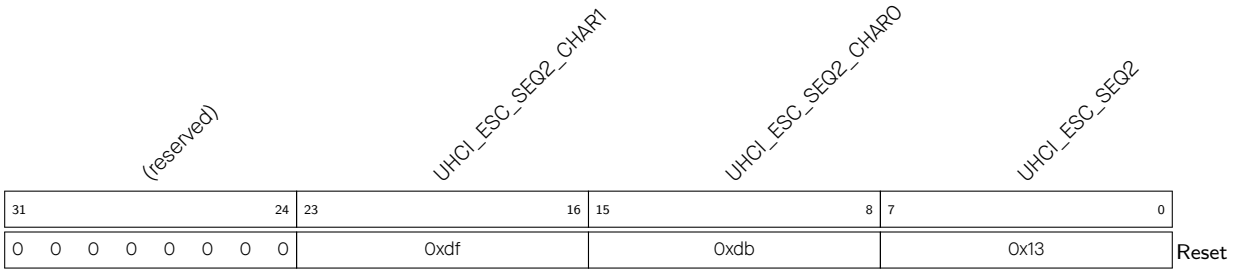


UHCI_ESC_SEQ1 Configures a character that need to be encoded. The default value is 0x11 used as a flow control character. (R/W)

UHCI_ESC_SEQ1_CHAR0 Configures the first character of SLIP escape sequence. The default value is 0xDB. (R/W)

UHCI_ESC_SEQ1_CHAR1 Configures the second character of SLIP escape sequence. The default value is 0xDE. (R/W)

Register 45.92. UHCI_ESC_CONF3_REG (0x0078)

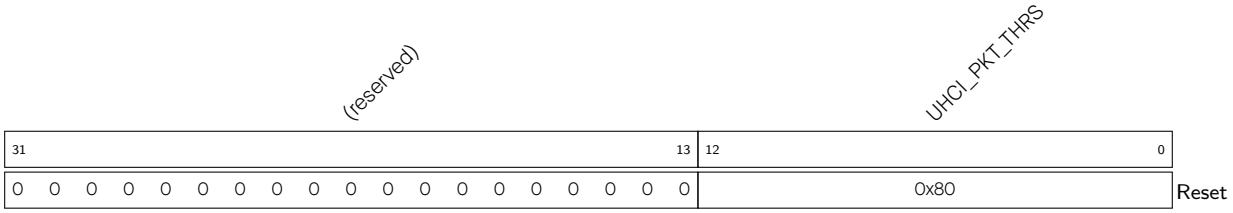


UHCI_ESC_SEQ2 Configures the character that needs to be decoded. The default value is 0x13 used as a flow control character. (R/W)

UHCI_ESC_SEQ2_CHAR0 Configures the first character of SLIP escape sequence. The default value is 0xDB. (R/W)

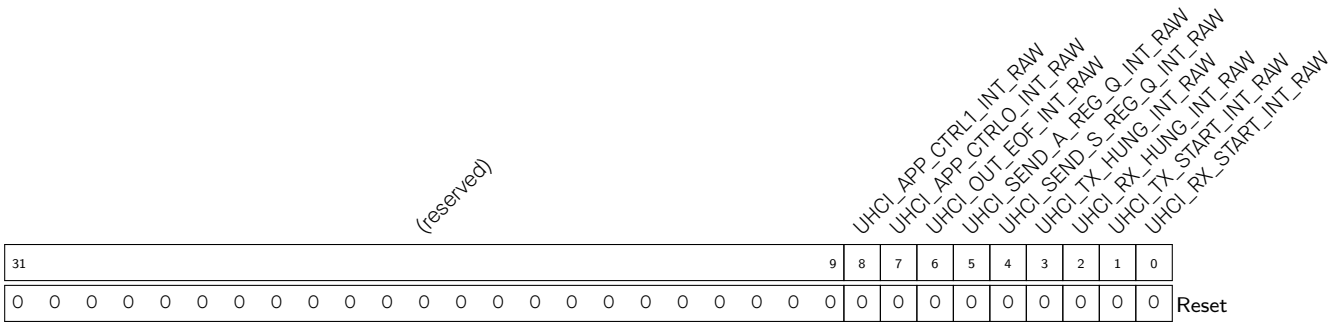
UHCI_ESC_SEQ2_CHAR1 Configures the second character of SLIP escape sequence. The default value is 0xDF. (R/W)

Register 45.93. UHCI_PKT_THRES_REG (0x007C)



UHCI_PKT_THRS Configures the maximum value of the packet length.
Measurement unit: byte.
Valid only when UHCI_HEAD_EN is 0. (R/W)

Register 45.94. UHCI_INT_RAW_REG (0x0004)



UHCI_RX_START_INT_RAW The raw interrupt status of UHCI_RX_START_INT. (R/WTC/SS)

UHCI_TX_START_INT_RAW The raw interrupt status of UHCI_TX_START_INT. (R/WTC/SS)

UHCI_RX_HUNG_INT_RAW The raw interrupt status of UHCI_RX_HUNG_INT. (R/WTC/SS)

UHCI_TX_HUNG_INT_RAW The raw interrupt status of UHCI_TX_HUNG_INT. (R/WTC/SS)

UHCI_SEND_S_REG_Q_INT_RAW The raw interrupt status of UHCI_SEND_S_REG_Q_INT. (R/WTC/SS)

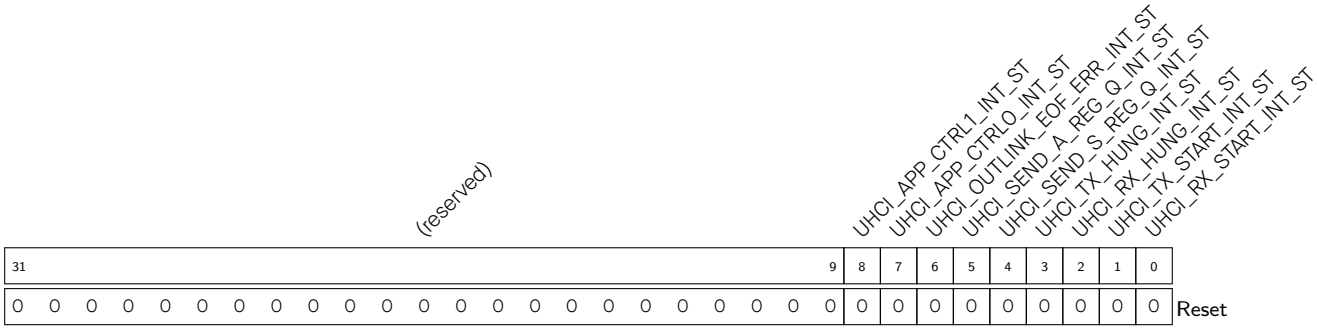
UHCI_SEND_A_REG_Q_INT_RAW The raw interrupt status of UHCI_SEND_A_REG_Q_INT. (R/WTC/SS)

UHCI_OUT_EOF_INT_RAW The raw interrupt status of UHCI_OUT_EOF_INT. (R/WTC/SS)

UHCI_APP_CTRL0_INT_RAW The raw interrupt status of UHCI_APP_CTRL0_INT. (R/W)

UHCI_APP_CTRL1_INT_RAW The raw interrupt status of UHCI_APP_CTRL1_INT. (R/W)

Register 45.95. UHCI_INT_ST_REG (0x0008)



UHCI_RX_START_INT_ST The masked interrupt status of UHCI_RX_START_INT. (RO)

UHCI_TX_START_INT_ST The masked interrupt status of UHCI_TX_START_INT. (RO)

UHCI_RX_HUNG_INT_ST The masked interrupt status of UHCI_RX_HUNG_INT. (RO)

UHCI_TX_HUNG_INT_ST The masked interrupt status of UHCI_TX_HUNG_INT. (RO)

UHCI_SEND_S_REG_Q_INT_ST The masked interrupt status of UHCI_SEND_S_REG_Q_INT. (RO)

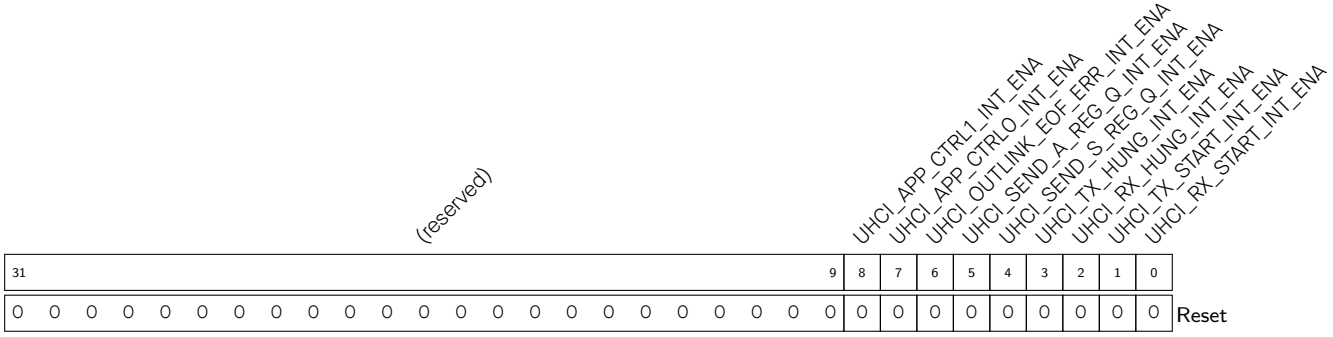
UHCI_SEND_A_REG_Q_INT_ST The masked interrupt status of UHCI_SEND_A_REG_Q_INT. (RO)

UHCI_OUTLINK_EOF_ERR_INT_ST The masked interrupt status of UHCI_OUTLINK_EOF_ERR_INT. (RO)

UHCI_APP_CTRL0_INT_ST The masked interrupt status of UHCI_APP_CTRL0_INT. (RO)

UHCI_APP_CTRL1_INT_ST The masked interrupt status of UHCI_APP_CTRL1_INT. (RO)

Register 45.96. UHCI_INT_ENA_REG (0x000C)



UHCI_RX_START_INT_ENA Write 1 to enable UHCI_RX_START_INT. (R/W)

UHCI_TX_START_INT_ENA Write 1 to enable UHCI_TX_START_INT. (R/W)

UHCI_RX_HUNG_INT_ENA Write 1 to enable UHCI_RX_HUNG_INT. (R/W)

UHCI_TX_HUNG_INT_ENA Write 1 to enable UHCI_TX_HUNG_INT. (R/W)

UHCI_SEND_S_REG_Q_INT_ENA Write 1 to enable UHCI_SEND_S_REG_Q_INT. (R/W)

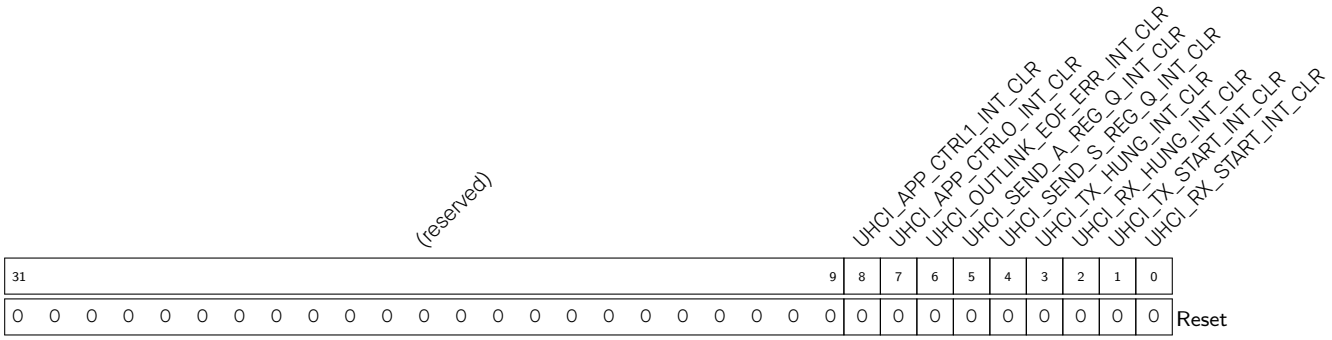
UHCI_SEND_A_REG_Q_INT_ENA Write 1 to enable UHCI_SEND_A_REG_Q_INT. (R/W)

UHCI_OUTLINK_EOF_ERR_INT_ENA Write 1 to enable UHCI_OUTLINK_EOF_ERR_INT. (R/W)

UHCI_APP_CTRL0_INT_ENA Write 1 to enable UHCI_APP_CTRL0_INT. (R/W)

UHCI_APP_CTRL1_INT_ENA Write 1 to enable UHCI_APP_CTRL1_INT. (R/W)

Register 45.97. UHCI_INT_CLR_REG (0x0010)



UHCI_RX_START_INT_CLR Write 1 to clear UHCI_RX_START_INT. (WT)

UHCI_TX_START_INT_CLR Write 1 to clear UHCI_TX_START_INT. (WT)

UHCI_RX_HUNG_INT_CLR Write 1 to clear UHCI_RX_HUNG_INT. (WT)

UHCI_TX_HUNG_INT_CLR Write 1 to clear UHCI_TX_HUNG_INT. (WT)

UHCI_SEND_S_REG_Q_INT_CLR Write 1 to clear UHCI_SEND_S_REG_Q_INT. (WT)

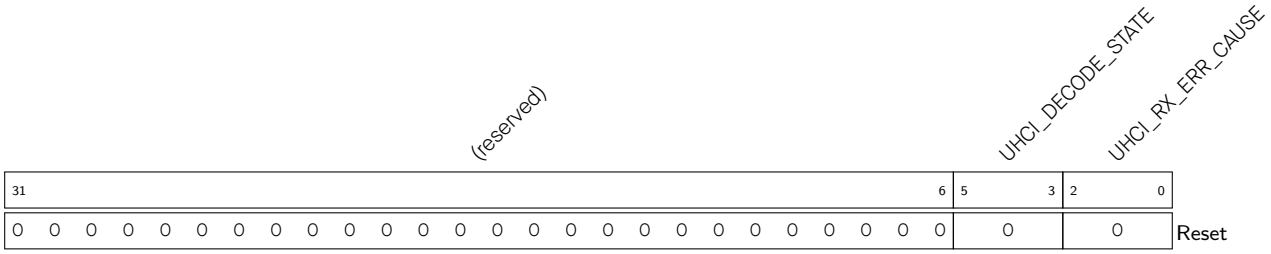
UHCI_SEND_A_REG_Q_INT_CLR Write 1 to clear UHCI_SEND_A_REG_Q_INT. (WT)

UHCI_OUTLINK_EOF_ERR_INT_CLR Write 1 to clear UHCI_OUTLINK_EOF_ERR_INT. (WT)

UHCI_APP_CTRL0_INT_CLR Write 1 to clear UHCI_APP_CTRL0_INT. (WT)

UHCI_APP_CTRL1_INT_CLR Write 1 to clear UHCI_APP_CTRL1_INT. (WT)

Register 45.98. UHCI_STATE0_REG (0x0018)



UHCI_RX_ERR_CAUSE Represents the error type when DMA has received a packet with error.

0: Invalid. No effect

1: Checksum error in the HCI packet

2: Sequence number error in the HCI packet

3: CRC bit error in the HCI packet

4: 0xC0 is found but the received HCI packet is not complete 5: 0xC0 is not found when the HCI packet has been received

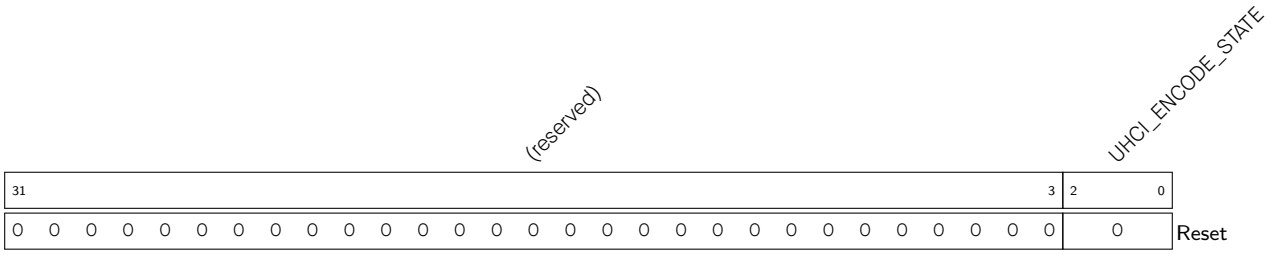
6: CRC check error

7: Invalid. No effect

(RO)

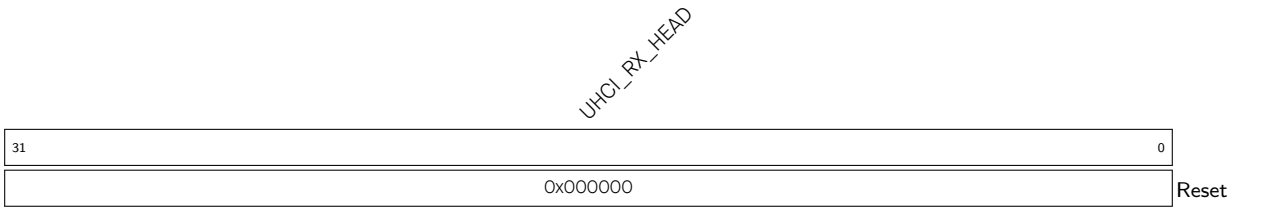
UHCI_DECODE_STATE Represents the UHCI decoder status. (RO)

Register 45.99. UHCI_STATE1_REG (0x001C)



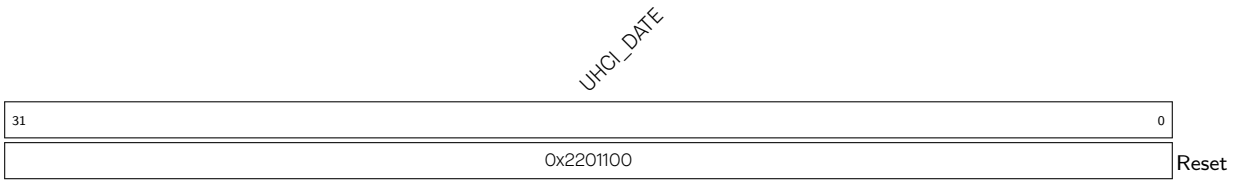
UHCI_ENCODE_STATE Represents the UHCI encoder status. (RO)

Register 45.100. UHCI_RX_HEAD_REG (0x002C)



UHCI_RX_HEAD Represents the header of the current received packet. (RO)

Register 45.101. UHCI_DATE_REG (0x0080)



UHCI_DATE Version control register. (R/W)

Chapter 46

SPI Controller (SPI)

Note:
This chapter applies only to ESP32-P4 chip revision v1.3. Content for the latest chip revision is still under development and will be published in a future TRM version.

46.1 Overview

The Serial Peripheral Interface (SPI) is a synchronous serial interface useful for communication with external peripherals. The ESP32-P4 chip integrates four SPI controllers:

- MSPI controller, including two sub-controllers
  - FLASH MSPI controller
    - FLASH MSPI SPI0
    - FLASH MSPI SPI1
  - PSRAM MSPI controller
    - PSRAM MSPI SPI0
    - PSRAM MSPI SPI1
- General Purpose SPI2 (GP-SPI2)
- General Purpose SPI3 (GP-SPI3)
- Low-Power SPI (LP-SPI)

MSPI controller is primarily reserved for internal use to communicate with external flash and PSRAM memory. This chapter mainly focuses on the GP-SPI and LP-SPI. **In this chapter unless otherwise stated, GP-SPI refers to both GP-SPI2 and GP-SPI3.**

46.2 Glossary

To better illustrate the functions of GP-SPI and LP-SPI, the following terms are used in this chapter.

Master Mode	GP-SPI or LP-SPI acts as an SPI master and initiates SPI transactions.
Slave Mode	GP-SPI or LP-SPI acts as an SPI slave and exchanges data with its master when its CS is asserted.
MISO	Master in, slave out, data transmission from a slave to a master.
MOSI	Master out, slave in, data transmission from a master to a slave.

Transaction One instance of a master asserting a CS line, transferring data to and from a slave, and de-asserting the CS line. Transactions are atomic, which means they can never be interrupted by another transaction.

SPI Transfer The whole process of an SPI master exchanging data with a slave. One SPI transfer consists of one or more SPI transactions.

Single Transfer An SPI transfer that consists of only one transaction.

CPU-Controlled Transfer A data transfer that happens between CPU configured buffer SPI_W0_REG~SPI_W15_REG (LP_SPI_W0_REG~LP_SPI_W15_REG) and SPI peripherals.

DMA-Controlled Transfer A data transfer that happens between DMA and SPI peripherals, controlled by the DMA engine.

Configurable Segmented Transfer A data transfer controlled by DMA when GP-SPI2 works as a master. Such transfer consists of multiple transactions (segments), and each of transaction can be configured independently.

Slave Segmented Transfer A data transfer controlled by DMA when GP-SPI works as a slave. Such transfer consists of multiple transactions (segments).

Full-duplex The sending line and receiving line between the master and the slave are independent. Sending data and receiving data happen at the same time.

Half-duplex Only one side, the master or the slave, sends data first, and the other side receives data. Sending data and receiving data can not happen at the same time.

4-line full-duplex 4-line here means: clock line, CS line, and two data lines. The two data lines can be used to send or receive data simultaneously.

4-line half-duplex 4-line here means: clock line, CS line, and two data lines. The two data lines can not be used simultaneously.

3-line half-duplex 3-line here means: clock line, CS line, and one data line. The data line is used to transmit or receive data.

1-bit SPI In one clock cycle, one bit can be transferred.

(2-bit) Dual SPI In one clock cycle, two bits can be transferred.

Dual Output Read A data mode of Dual SPI. In one clock cycle, one bit of a command, or one bit of an address, or two bits of data can be transferred.

Dual I/O Read Another data mode of Dual SPI. In one clock cycle, one bit of a command, or two bits of an address, or two bits of data can be transferred.

(4-bit) Quad SPI In one clock cycle, four bits can be transferred.

Quad Output Read A data mode of Quad SPI. In one clock cycle, one bit of a command, or one bit of an address, or four bits of data can be transferred.

Quad I/O Read Another data mode of Quad SPI. In one clock cycle, one bit of a command, or four bits of an address, or four bits of data can be transferred.

QPI In one clock cycle, four bits of a command, or four bits of an address, or four bits of data can be transferred.

(8-bit) Octal SPI In one clock cycle, eight bits can be transferred.

Octal Output Read A data mode of Octal SPI. In one clock cycle, one bit of a command, or one bit of an address, or eight bits of data can be transferred.

Octal I/O Read Another data mode of Octal SPI. In one clock cycle, one bit of a command, or eight bits of an address, or eight bits of data can be transferred.

OPI In one clock cycle, eight bits of a command, or eight bits of an address, or eight bits of data can be transferred.

SPI2 The prefix of the signals for GP-SPI2. SPI2 bus signals are routed to GPIO pins via HP GPIO matrix or HP IO MUX.

SPI3 The prefix of the signals for GP-SPI3. SPI3 bus signals are routed to GPIO pins via HP GPIO matrix only.

LP_SPI The prefix of the signals for LP-SPI. LP_SPI bus signals are routed to GPIO pins via LP GPIO matrix only.

46.3 Features

GP-SPI has the following features:

- Works as master or as slave
- Half- and full-duplex communications
- CPU- and DMA-controlled transfers
- Various data modes
  - **GP-SPI2**
    * 1-bit SPI mode
    * 2-bit Dual SPI mode
    * 4-bit Quad SPI mode
    * QPI mode
    * 8-bit Octal SPI mode (available only when GP-SPI2 works as a master)
    * OPI mode (available only when GP-SPI2 works as a master)
  - **GP-SPI3**
    * 1-bit SPI mode
    * 2-bit Dual SPI mode
    * 4-bit Quad SPI mode
    * QPI mode
- Configurable module clock frequency
  - Master: up to 80 MHz
  - Slave: up to 60 MHz

- Configurable data length
  - CPU-controlled transfer as master or as slave: 1∼64 bytes
  - DMA-controlled single transfer as master: 1∼32 KB
  - DMA-controlled configurable segmented transfer as master: data length is unlimited
  - DMA-controlled single transfer or segmented transfer as slave: data length is unlimited
- Configurable bit read/write order
- Independent interrupts for CPU-controlled transfer and DMA-controlled transfer
- Configurable clock polarity and phase
- Four SPI clock modes: mode 0∼mode 3
- Multiple CS lines as master
  - **GP-SPI2**: CS0∼CS5
  - **GP-SPI3**: CS0∼CS2
- Able to communicate with SPI devices, such as a sensor, a screen controller, as well as a flash or RAM chip

LP-SPI is a simplified version of GP-SPI and has a subset of GP-SPI's features:

- Works as a master or as a slave
- Half- and full-duplex communications
- CPU-controlled transfer
- 1-bit SPI data mode
- Configurable module clock frequency:
  - Master: up to 40 MHz
  - Slave: up to 40 MHz
- Configurable data length:
  - CPU-controlled transfer as master or as slave: 1∼64 bytes
- Configurable bit read/write order
- Interrupts for CPU-controlled transfer
- Configurable clock polarity and phase
- Four SPI clock modes: mode 0∼mode 3
- One CS line as master: CS0
- Wake-up feature as slave (the only new feature compared with GP-SPI)

46.4 Architectural Overview

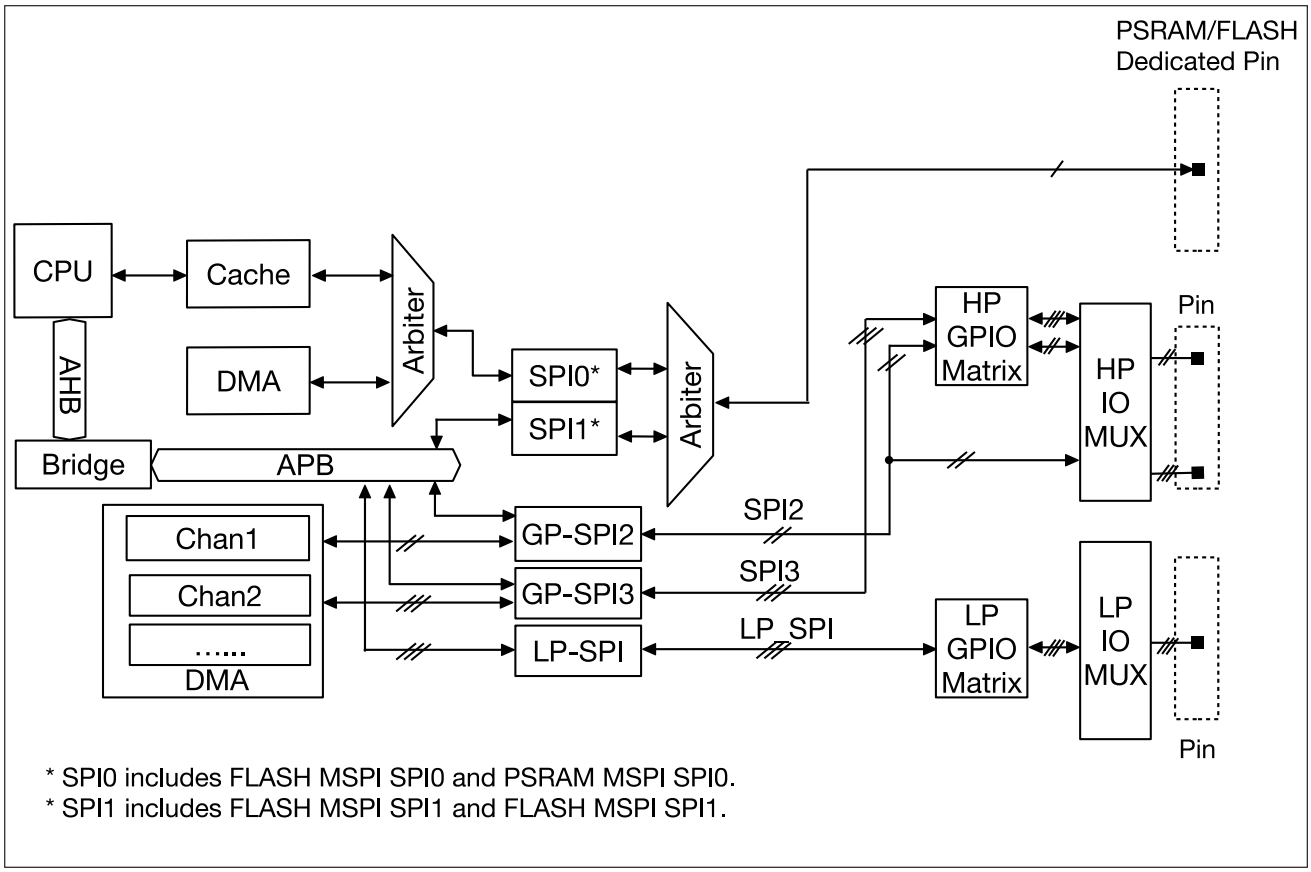


Figure 46.4-1. SPI Module Overview

Figure 46.4-1 shows an overview of this SPI module. GP-SPI2, GP-SPI3, and LP-SPI exchange data with SPI devices by the following ways:

- CPU-controlled transfer:
  - CPU ↔ LP-SPI ↔ SPI devices
  - CPU ↔ GP-SPI2 (GP-SPI3) ↔ SPI devices
- DMA-controlled transfer: DMA ↔ GP-SPI2 (GP-SPI3) ↔ SPI devices

In this chip, "SPI2", "SPI3", and "LP_SPI" are used as prefixes of GP-SPI2, GP-SPI3, and LP-SPI input and output signal buses to illustrate the SPI bus functions.

SPI2 bus signals are routed to GPIO pins via either the HP GPIO matrix or the HP IO MUX, while SPI3 bus signals are routed to GPIO pins via the HP GPIO matrix only. LP_SPI bus signals are routed via the LP GPIO matrix only. For more information, see Chapter 10 *GPIO Matrix and IO MUX*.

The functionalities of GP-SPI3 are nearly the same as those of GP-SPI2. The functionalities of LP-SPI are a subset of the GP-SPI functionalities. GP-SPI2's functionalities are described in Section 46.5. The differences among GP-SPI2, GP-SPI3, and LP-SPI are described in Section 46.5.1 and Section 46.10.

46.5 Functional Description

46.5.1 Data Modes

GP-SPI and LP-SPI can be configured as either a master or a slave to communicate with other SPI devices in the following data modes, see Table 46.5-1. For more information about the data modes used when GP-SPI or LP-SPI works as a master, see Section 46.5.9, and Section 46.5.10 for GP-SPI working as a slave.

Table 46.5-1. Data Modes Supported by GP-SPI and LP-SPI

Data Mode		CMD State	Address State	Data State	GP-SPI2	GP-SPI3	LP-SPI
1-bit SPI		1-bit	1-bit	1-bit	Y	Y	Y
Dual SPI	Dual Output Read	1-bit	1-bit	2-bit	Y	Y	—
	Dual I/O Read	1-bit	2-bit	2-bit	Y	Y	—
Quad SPI	Quad Output Read	1-bit	1-bit	4-bit	Y	Y	—
	Quad I/O Read	1-bit	4-bit	4-bit	Y	Y	—
Octal SPI	Octal Output Read	1-bit	1-bit	8-bit	Y	—	—
	Octal I/O Read	1-bit	8-bit	8-bit	Y	—	—
QPI		4-bit	4-bit	4-bit	Y	Y	—
OPI		8-bit	8-bit	8-bit	Y	—	—

46.5.2 Introduction to Bus Signals

The functional description of SPI2/SPI3 and LP_SPI bus signals is shown in Table 46.5-2. Tables 46.5-3, 46.5-4, and 46.5-5 list the signals used in various SPI modes.

Table 46.5-2. Functional Description of SPI2/SPI3 and LP_SPI Bus Signals

SPI2 Bus Signal	SPI3 Bus Signal	LP-SPI Bus Signal	Function
SPI2CLK	SPI3_CLK	LP_SPI_CLK	Input and output clock as master/slave
SPI2CS0	SPI3_CS0	LP_SPI_CS	Input and output CS signal as master/slave
SPI2CS1 ~5	SPI3_CS1~2	—	Output CS signal as master
SPI2D	SPI3_D	LP_SPI_D	MOSI/SIO0 (serial data input and output, bit0)
SPI2Q	SPI3_Q	LP_SPI_Q	MISO/SIO1 (serial data input and output, bit1)
SPI2WP	SPI3_WP	—	SIO2 (serial data input and output, bit2)
SPI2HD	SPI3_HD	—	SIO3 (serial data input and output, bit3)
SPI2D4~7	—	—	SIO4~7 (serial data input and output, bit4~7)
SPI2DQS	—	—	Output data mask signal as master

Table 46.5-3. SPI2 Bus Signals Used in Various SPI Modes

SPI2 Bus Signal	Master								Slave					
	1-bit SPI			Dual SPI	Quad SPI	QPI	Octal SPI	OPI	1-bit SPI			Dual SPI	Quad SPI	QPI
	FD[1]	3-line HD[2]	4-line HD						FD	3-line HD	4-line HD			
SPI2CLK	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
SPI2CS0	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
SPI2CS1	Y	Y	Y	Y	Y	Y	Y	Y						
SPI2CS2	Y	Y	Y	Y	Y	Y	Y	Y						
SPI2CS3	Y	Y	Y	Y	Y	Y	Y	Y						
SPI2CS4	Y	Y	Y	Y	Y	Y	Y	Y						
SPI2CS5	Y	Y	Y	Y	Y	Y	Y	Y						
SPI2D	Y	Y	(Y)[3]	Y[4]	Y[5]	Y	Y	Y	Y	Y	(Y)[6]	Y[7]	Y[8]	Y
SPI2Q	Y		(Y)[3]	Y[4]	Y[5]	Y	Y	Y	Y		(Y)[6]	Y[7]	Y[8]	Y
SPI2WP					Y[5]	Y	Y	Y					Y[8]	Y
SPI2HD					Y[5]	Y	Y	Y					Y[8]	Y
SPI2D4~7							Y	Y						
SPI2DQS							Y	Y						

[1] FD: full-duplex

[2] HD: half-duplex

[3] Only one of the two signals is used at a time.

[4] The two signals are used in parallel.

[5] The four signals are used in parallel.

[6] Only one of the two signals is used at a time.

[7] The two signals are used in parallel.

[8] The four signals are used in parallel.

GoBack

Table 46.5-4. SPI3 Bus Signals Used in Various SPI Modes

SPI3 Bus Signal	Master						Slave					
	1-bit SPI			Dual SPI	Quad SPI	QPI	1-bit SPI			Dual SPI	Quad SPI	QPI
	FD[1]	3-line HD[2]	4-line HD				FD	3-line HD	4-line HD			
SPI3_CLK	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
SPI3_CS0	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
SPI3_CS1	Y	Y	Y	Y	Y	Y						
SPI3_CS2	Y	Y	Y	Y	Y	Y						
SPI3_D	Y	Y	(Y)[3]	Y[4]	Y[5]	Y	Y	Y	(Y)[6]	Y[7]	Y[8]	Y
SPI3_Q	Y		(Y)[3]	Y[4]	Y[5]	Y	Y		(Y)[6]	Y[7]	Y[8]	Y
SPI3_WP					Y[5]	Y					Y[8]	Y
SPI3_HD					Y[5]	Y					Y[8]	Y

[1] FD: full-duplex

[2] HD: half-duplex

[3] Only one of the two signals is used at a time.

[4] The two signals are used in parallel.

[5] The four signals are used in parallel.

[6] Only one of the two signals is used at a time.

[7] The two signals are used in parallel.

[8] The four signals are used in parallel.

Table 46.5-5. LP-SPI Bus Signals Used in Various SPI Modes

LP_SPI Bus Signal	Master (1-bit SPI)			Slave (1-bit SPI)		
	FD[1]	3-line HD[2]	4-line HD	FD	3-line HD	4-line HD
LP_SPI_CLK	Y	Y	Y	Y	Y	Y
LP_SPI_CS	Y	Y	Y	Y	Y	Y
LP_SPI_D	Y	Y	(Y)[3]	Y	Y	(Y)[3]
LP_SPI_Q	Y		(Y)[3]	Y		(Y)[3]

[1] FD: full-duplex

[2] HD: half-duplex

[3] Only one of the two signals is used at a time.

46.5.3 Bit Read/Write Order Control

- GP-SPI
  - When operating as a master:
    * The bit order of the command, address and data sent by the GP-SPI master is controlled by SPI_WR_BIT_ORDER.
    * The bit order of the data received by the master is controlled by SPI_RD_BIT_ORDER.
  - When operating as a slave:
    * The bit order of the data sent by the GP-SPI slave is controlled by SPI_WR_BIT_ORDER.
    * The bit order of the command, address and data received by the slave is controlled by SPI_RD_BIT_ORDER.
- LP-SPI
  - When operating as a master:
    * The bit order of the command, address and data sent by the LP-SPI master is controlled by LP_SPI_WR_BIT_ORDER.
    * The bit order of the data received by the master is controlled by LP_SPI_RD_BIT_ORDER.
  - When operating as a slave:
    * The bit order of the data sent by the LP-SPI slave is controlled by LP_SPI_WR_BIT_ORDER.
    * The bit order of the command, address and data received by the slave is controlled by LP_SPI_RD_BIT_ORDER.

Tables 46.5-6 and 46.5-7 show the functions of SPI_RD/WR_BIT_ORDER and LP_SPI_RD/WR_BIT_ORDER.

GoBack

Table 46.5-6. Bit Order Control in GP-SPI (Take GP-SPI2 as an Example)

Bit Mode	SPI2 Bus Data	SPI_RD/WR_BIT_ORDER = 0 (MSB)	SPI_RD/WR_BIT_ORDER = 2 (MSB)	SPI_RD/WR_BIT_ORDER = 1 (LSB)	SPI_RD/WR_BIT_ORDER = 3 (LSB)
1-bit mode	SPI2D or SPI2Q	B7→B6→B5→B4→B3→B2→B1→B0	B7→B6→B5→B4→B3→B2→B1→B0	B0→B1→B2→B3→B4→B5→B6→B7	B0→B1→B2→B3→B4→B5→B6→B7
2-bit mode	SPI2Q	B7→B5→B3→B1	B6→B4→B2→B0	B1→B3→B5→B7	B0→B2→B4→B6
	SPI2D	B6→B4→B2→B0	B7→B5→B3→B1	B0→B2→B4→B6	B1→B3→B5→B7
4-bit mode	SPI2HD	B7→B3	B4→B0	B3→B7	B0→B4
	SPI2WP	B6→B2	B5→B1	B2→B6	B1→B5
	SPI2Q	B5→B1	B6→B2	B1→B5	B2→B6
	SPI2D	B4→B0	B7→B3	B0→B4	B3→B7
8-bit mode	SPI2D7	B7	B7	B0	B0
	SPI2D6	B6	B6	B1	B1
	SPI2D5	B5	B5	B2	B2
	SPI2D4	B4	B4	B3	B3
	SPI2HD	B3	B3	B4	B4
	SPI2WP	B2	B2	B5	B5
	SPI2Q	B1	B1	B6	B6
	SPI2D	B0	B0	B7	B7

Table 46.5-7. Bit Order Control in LP-SPI

Bit Mode	LP_SPI Bus Signal	LP_SPI_RD/WR_BIT_ORDER = 0 (MSB)	LP_SPI_RD/WR_BIT_ORDER = 1 (LSB)
1-bit mode	LP_SPI_D or LP_SPI_Q	B7→B6→B5→B4→B3→B2→B1→B0	B0→B1→B2→B3→B4→B5→B6→B7

46.5.4 Unaligned Byte Transfer

- When operating as a master, GP-SPI sends and receives data in bits. The bit length is equal to SPI_MS_DATA_BITLEN + 1, a multiple of 1/2/4/8 in 1/2/4/8-bit mode.
  - When sending data whose length is not an integer multiple of 8 bits, the software needs to fill the last part of the data less than 8 bits into a full 1-byte data.
  - When receiving data whose length is not an integer multiple of 8 bits, the part less than 1 byte is still received as 1 byte.
- When operating as slave, GP-SPI sends and receives data in bits. The bit length is a multiple of 1/2/4/8 in 1/2/4/8-bit mode.
  - When sending data whose length is not an integer multiple of 8 bits, the software needs to fill the last part of the data less than 8 bits into a full 1-byte data.
  - When receiving data whose length is not an integer multiple of 8 bits, the part less than 1 byte is still received as 1 byte. The total bit length received can be read from SPI_SLV_DATA_BITLEN. The valid bits in the last 1 byte is indicated by SPI_SLV_LAST_BYTE_STRB.

Note:

- No matter working as master or slave, GP-SPI sends and receives the data parts less than 1 byte following the same bit order as the other data bits as configured.
- LP-SPI does not support this unaligned byte transfer.

46.5.5 Transfer Types

The transfer types supported by GP-SPI working as a master or a slave are shown in Table 46.5-8.

Table 46.5-8. Supported Transfer Types as Master or Slave

Mode		CPU-Controlled Single Transfer [1]	DMA-Controlled Single Transfer	DMA-Controlled Configurable Segmented Transfer [2]	DMA-Controlled Slave Segmented Transfer
Master	Full-Duplex	Y	Y	Y	—
	Half-Duplex	Y	Y	Y	—
Slave	Full-Duplex	Y	Y	—	Y
	Half-Duplex	Y	Y	—	Y

[1] LP-SPI supports only the CPU-controlled single transfer.

[2] GP-SPI3 does not support DMA-controlled configurable segmented transfer.

The following sections provide detailed information about the transfer types listed in the table above.

46.5.6 CPU-Controlled Data Transfer

Note:
Both GP-SPI and LP-SPI provide 16 x 32-bit data buffers:

- GP-SPI: SPI_W0_REG~SPI_W15_REG
- LP-SPI: LP_SPI_W0_REG~LP_SPI_W15_REG

The following section illustrates the buffer's structure using GP-SPI data buffer as an example.

Figure 46.5-1 shows the buffer's structure. CPU-controlled transfer indicates the transfer, in which the data to send is from GP-SPI data buffer and the received data is stored to GP-SPI data buffer. In such transfer, every single transaction needs to be triggered by the CPU, after its related registers are configured. For such reason, the CPU-controlled transfer is always single transfer (consisting of only one transaction). CPU-controlled transfer supports full-duplex communication and half-duplex communication.

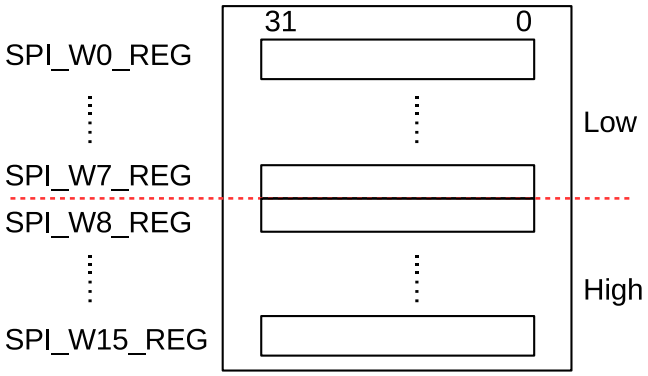


Figure 46.5-1. Data Buffer Used in CPU-Controlled Transfer

46.5.6.1 CPU-Controlled Master Transfer

In a CPU-controlled master full-duplex or half-duplex transfer, the RX or TX data is saved to or sent from SPI_W0_REG~SPI_W15_REG. The bits SPI_USR_MOSI_HIGHPART and SPI_USR_MISO_HIGHPART control which buffers are used. See the description below.

- **TX data**
  - When SPI_USR_MOSI_HIGHPART is cleared, i.e., high part mode is disabled, TX data is read from SPI_W0_REG~SPI_W15_REG and the data address is incremented by 1 on each byte transferred. **If the data byte length is larger than 64, the data in SPI_W0_REG~SPI_W15_REG may be sent more than once.**
    Take each 256 bytes as a cycle:
    * The first 64 bytes (*Byte 0~Byte 63*) are read from SPI_W0_REG~SPI_W15_REG, sequentially.
    * *Byte 64~Byte 255* are read from SPI_W15_REG[31:24] repeatedly.
    * *Byte 256~Byte 319* (the first 64 bytes in another 256 bytes) are read from SPI_W0_REG~SPI_W15_REG again, sequentially, same as the behaviors described above.

    For instance: to send 258 bytes (*Byte 0~Byte 257*), the data is read from the registers as follows:
    * The first 64 bytes (*Byte 0~Byte 63*) are read from SPI_W0_REG~SPI_W15_REG, sequentially.
    * *Byte 64~Byte 255* are read from SPI_W15_REG[31:24] repeatedly.

* The other bytes (*Byte 256* and *Byte 257*) are read from SPI_W0_REG[7:0] and SPI_W0_REG[15:8] again, sequentially. The logic is:
  - The address to read data for *Byte 256* is the result of (256 % 64 = 0), i.e., SPI_W0_REG[7:0].
  - The address to read data for *Byte 257* is the result of (257 % 64 = 1), i.e., SPI_W0_REG[15:8].

– When SPI_USR_MOSI_HIGHPART is set, i.e., high part mode is enabled, TX data is read from SPI_W8_REG~SPI_W15_REG and the data address is incremented by 1 on each byte transferred. **If the data byte length is larger than 32, the data in SPI_W8_REG~SPI_W15_REG may be sent more than once.**
Take each 256 bytes as a cycle:

* The first 32 bytes (*Byte 0~Byte 31*) are read from SPI_W8_REG~SPI_W15_REG, sequentially.
* *Byte 32~Byte 255* are read from SPI_W15_REG[31:24] repeatedly.
* *Byte 256~Byte 287* (the first 32 bytes in the another 256 bytes) are read from SPI_W8_REG~SPI_W15_REG again, sequentially, same as the behaviors described above.

For instance: to send 258 bytes (*Byte 0~Byte 257*), the data is read from the registers as follows:

* The first 32 bytes (*Byte 0~Byte 31*) are read from SPI_W8_REG~SPI_W15_REG, sequentially.
* *Byte 32~Byte 255* are read from SPI_W15_REG[31:24] repeatedly.
* The other bytes (*Byte 256* and *Byte 257*) are read from SPI_W8_REG[7:0] and SPI_W8_REG[15:8] again, sequentially. The logic is:
  - The address to read data for *Byte 256* is the result of (256 % 32 = 0), i.e., SPI_W8_REG[7:0].
  - The address to read data for *Byte 257* is the result of (257 % 32 = 1), i.e., SPI_W8_REG[15:8].

- **RX data**

– When SPI_USR_MISO_HIGHPART is cleared, i.e., high part mode is disabled, RX data is saved to SPI_W0_REG~SPI_W15_REG, and the data address is incremented by 1 on each byte transferred. **If the data byte length is larger than 64, the data in SPI_W0_REG~SPI_W15_REG may be overwritten.**
Take each 256 bytes as a cycle:

* The first 64 bytes (*Byte 0~Byte 63*) are saved to SPI_W0_REG~SPI_W15_REG, sequentially.
* *Byte 64~Byte 255* are saved to SPI_W15_REG[31:24] repeatedly.
* *Byte 255~Byte 319* (the first 64 bytes in the another 256 bytes) are saved to SPI_W0_REG~SPI_W15_REG again, sequentially, same as the behaviors described above.

For instance: to receive 258 bytes (*Byte 0~Byte 257*), the data is saved to the registers as follows:

* The first 64 bytes (*Byte 0~Byte 63*) are saved to SPI_W0_REG~SPI_W15_REG, sequentially.
* *Byte 64~Byte 255* are saved to SPI_W15_REG[31:24] repeatedly.

* The other bytes (*Byte 256* and *Byte 257*) are saved to SPI_W0_REG[7:0] and SPI_W0_REG[15:8] again, sequentially. The logic is:
  - The address to save *Byte 256* is the result of (256 % 64 = 0), i.e., SPI_W0_REG[7:0].
  - The address to save *Byte 257* is the result of (257 % 64 = 1), i.e., SPI_W0_REG[15:8].

– When SPI_USR_MISO_HIGHPART is set, i.e., high part mode is enabled, the RX data is saved to SPI_W8_REG~SPI_W15_REG, and the data address is incremented by 1 on each byte transferred. **If the data byte length is larger than 32, the content of SPI_W8_REG~SPI_W15_REG may be overwritten.**
Take each 256 bytes as a cycle:

* *Byte 0~Byte 31* are saved to SPI_W8_REG~SPI_W15_REG, sequentially.
* *Byte 32~Byte 255* are saved to SPI_W15_REG[31:24] repeatedly.
* *Byte 256~Byte 287* (the first 32 bytes in the another 256 bytes) are saved to SPI_W8_REG~SPI_W15_REG again, sequentially.

For instance: to receive 258 bytes (*Byte 0~Byte 257*), the data is saved to the registers as follows:

* The first 32 bytes (*Byte 0~Byte 31*) are saved to SPI_W8_REG~SPI_W15_REG, sequentially.
* *Byte 32~Byte 255* are saved to SPI_W15_REG[31:24] repeatedly.
* The other bytes (*Byte 256* and *Byte 257*) are saved to SPI_W8_REG[7:0] and SPI_W8_REG[15:8] again, sequentially. The logic is:
  - The address to save *Byte 256* is the result of (256 % 32 = 0), i.e., SPI_W8_REG[7:0].
  - The address to save *Byte 257* is the result of (257 % 32 = 1), i.e., SPI_W8_REG[15:8].

Note:

- TX/RX data address mentioned above both are byte-addressable.
  - If high part mode is disabled, Address 0 stands for SPI_W0_REG[7:0], and Address 1 for SPI_W0_REG[15:8], and so on.
  - If high part mode is enabled, Address 0 stands for SPI_W8_REG[7:0], and Address 1 for SPI_W8_REG[15:8], and so on.

  The largest address points to SPI_W15_REG[31:24].
- To avoid any possible error in TX/RX data, such as TX data being sent more than once or RX data being overwritten, please make sure the registers are configured correctly.
- LP-SPI functions similarly, but uses LP-SPI registers and LP-SPI data buffers, as shown in Section 46.12.3.

46.5.6.2 CPU-Controlled Slave Transfer

In a CPU-controlled slave full-duplex or half-duplex transfer, the RX or TX data is saved to or sent from SPI_W0_REG~SPI_W15_REG, which are byte-addressable.

- In full-duplex communication, the address of SPI_W0_REG~SPI_W15_REG starts from 0 and is incremented by 1 on each byte transferred. If the data address is larger than 63, the data in SPI_W0_REG~SPI_W15_REG will be overwritten, same as the behaviors described in the master transfer when high part mode is disabled.

- In half-duplex communication, the *ADDR* value in transmission format is the start address of the RX or TX data, corresponding to the registers SPI_W0_REG~SPI_W15_REG. The RX or TX address is incremented by 1 on each byte transferred. If the address is larger than 63 (the highest byte address, i.e., SPI_W15_REG[31:24]), the data in SPI_W8_REG~SPI_W15_REG will be overwritten, same as the behaviors described in the master transfer when high part mode is enabled.

According to your applications, the registers SPI_W0_REG~SPI_W15_REG can be used as:

- data buffers only
- data buffers and status buffers
- status buffers only

46.5.7 DMA-Controlled Data Transfer

Note:
LP-SPI does not support this transfer. The description below is for GP-SPI only.

DMA-controlled transfer refers to the transfer in which the DMA RX module receives data and the DMA TX module sends data. This transfer is supported both as master and as slave.

A DMA-controlled transfer can be:

- a single transfer, consisting of only one transaction. GP-SPI supports this transfer both as master and as slave.
- a configurable segmented transfer, consisting of several transactions (segments). Only GP-SPI2 supports this transfer when working as a master. For more information, see Section 46.5.9.5.
- a slave segmented transfer, consisting of several transactions (segments). GP-SPI supports this transfer only when working as a slave. For more information, see Section 46.5.10.3.

A DMA-controlled transfer only needs to be triggered once by CPU. When such a transfer is triggered, data is transferred by the DMA engine from or to the DMA-linked memory, without CPU operation.

DMA-controlled transfer supports full-duplex communication, half-duplex communication and functions described in Section 46.5.9 and Section 46.5.10. Meanwhile, the DMA RX module is independent from the DMA TX module, which means that there are four kinds of full-duplex communications:

- Data is received in DMA-controlled mode and sent in DMA-controlled mode.
- Data is received in DMA-controlled mode but sent in CPU-controlled mode.
- Data is received in CPU-controlled mode but sent in DMA-controlled mode.
- Data is received in CPU-controlled mode and sent in CPU-controlled mode.

46.5.7.1 DMA Configuration

- Select a DMA channel *n* and configure DMA TX/RX descriptor. See Chapter 6 *GDMA Controller (GDMA-AHB, GDMA-AXI)*.

- Set AXI_DMA_INLINK_START_CH*n* or AXI_DMA_OUTLINK_START_CH*n* to start DMA RX or TX engine, respectively.
- Before all the DMA TX buffer is used or the DMA TX engine is reset, if AXI_DMA_OUTLINK_RESTART_CH*n* is set, a new TX buffer will be added to the end of the last TX buffer in use.
- DMA RX buffer is linked in the same way as the DMA TX buffer, by setting AXI_DMA_INLINK_START_CH*n* or AXI_DMA_INLINK_RESTART_CH*n*.
- The TX and RX data lengths are determined by the configured DMA TX and RX buffer respectively, both of which are 0~32 KB.
- Initialize DMA inlink and outlink before DMA starts. The bits SPI_DMA_RX_ENA and SPI_DMA_TX_ENA in register SPI_DMA_CONF_REG should be set, otherwise the read/write data will be stored to/sent from the registers SPI_W0_REG~SPI_W15_REG.

When GP-SPI operating as master, if AXI_DMA_IN_SUC_EOF_CH*n*_INT_ENA is set, then the interrupt AXI_DMA_IN_SUC_EOF_CH*n*_INT will be triggered when one single transfer or one configurable segmented transfer is finished.

When GP-SPI operating as slave, if AXI_DMA_IN_SUC_EOF_CH*n*_INT_ENA is set, then the interrupt AXI_DMA_IN_SUC_EOF_CH*n*_INT will be triggered when one of the following conditions are met.

Table 46.5-9. Interrupt Trigger Condition on GP-SPI Data Transfer as Slave

Transfer Type	Control Bit [1]	Control Bit [2]	Condition
Slave Single Transfer	0	0	A single transfer is done.
	1	0	A single transfer is done. Or the length of the received data is equal to (SPI_MS_DATA_BITLEN + 1).
Slave Segmented Transfer	0	1	CMD7 or End_SEG_TRANS is received correctly.
	1	1	CMD7 or End_SEG_TRANS is received correctly. Or the length of the received data is equal to (SPI_MS_DATA_BITLEN + 1).

[1] SPI_RX_EOF_EN
[2] SPI_DMA_SLV_SEG_TRANS_EN

46.5.7.2 DMA TX/RX Buffer Length Control

It is recommended that the length of the configured DMA TX/RX buffer is equal to the length of actual data transferred.

- If the length of the configured DMA TX buffer is shorter than that of the actual data transferred, the extra data will be the same as the last transferred data. SPI_OUTFIFO_EMPTY_ERR_INT and AXI_DMA_OUT_EOF_CH*n*_INT are triggered.
- If the length of the configured DMA TX buffer is longer than that of actual data transferred, the TX buffer is not fully used, and the remaining buffer will be used for the following transaction even if a new TX buffer is linked later. Please keep it in mind. Or save the unused data and reset the DMA.
- If the length of the configured DMA RX buffer is shorter than that of the actual data transferred, the extra data will be lost. SPI_INFIFO_FULL_ERR_INT and SPI_TRANS_DONE_INT are triggered. But AXI_DMA_IN_SUC_EOF_CH*n*_INT is not triggered.

- If the length of the configured DMA RX buffer is longer than that of the actual data transferred, the RX buffer is not fully used, and the remaining buffer is discarded. In the following transaction, a new linked buffer will be used directly.

46.5.8 Data Flow Control (Take GP-SPI as an Example)

CPU-controlled and DMA-controlled transfers are supported in GP-SPI both as master and as slave. CPU-controlled transfer means that data is transferred between registers SPI_W0_REG~SPI_W15_REG and the SPI device. DMA-controlled transfer means that data is transferred between the configured DMA TX/RX buffer and the SPI device. To select between the two transfer types, configure SPI_DMA_RX_ENA and SPI_DMA_TX_ENA before the transfer starts.

46.5.8.1 GP-SPI Functional Blocks

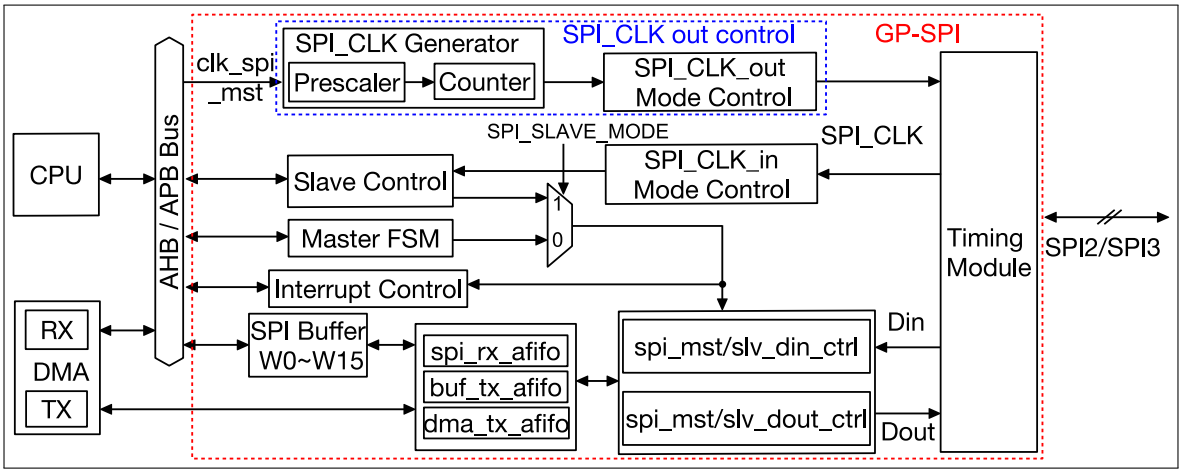


Figure 46.5-2. GP-SPI Functional Blocks

Figure 46.5-2 shows the main functional blocks in GP-SPI, including:

- Master FSM: all the features supported in GP-SPI as master are controlled by this state machine together with register configuration.
- SPI Buffer: SPI_W0_REG~SPI_W15_REG. See Figure 46.5-1. The data in CPU-controlled transfer is prepared in this buffer.
- Timing Module: captures data on SPI2/SPI3 bus.
- spi_mst/slv_din_ctrl and spi_mst/slv_dout_ctrl: converts the TX/RX data into bytes.
- spi_rx_afifo: stores the received data.
- buf_tx_afifo: stores the data to send.
- dma_tx_afifo: stores the data from DMA.
- clk_spi_mst: this clock is the module clock of GP-SPI and is used in GP-SPI as master to generate SPI_CLK signal for data transfer and for slaves.
- SPI_CLK Generator: generates SPI_CLK by dividing clk_spi_mst. The divider is determined by SPI_CLKCNT_N and SPI_CLKDIV_PRE.

- SPI_CLK_out Mode Control: outputs the SPI_CLK signal for data transfer and for slaves.
- SPI_CLK_in Mode Control: captures the SPI_CLK signal from SPI master when GP-SPI works as a slave.

46.5.8.2 Data Flow Control as Master

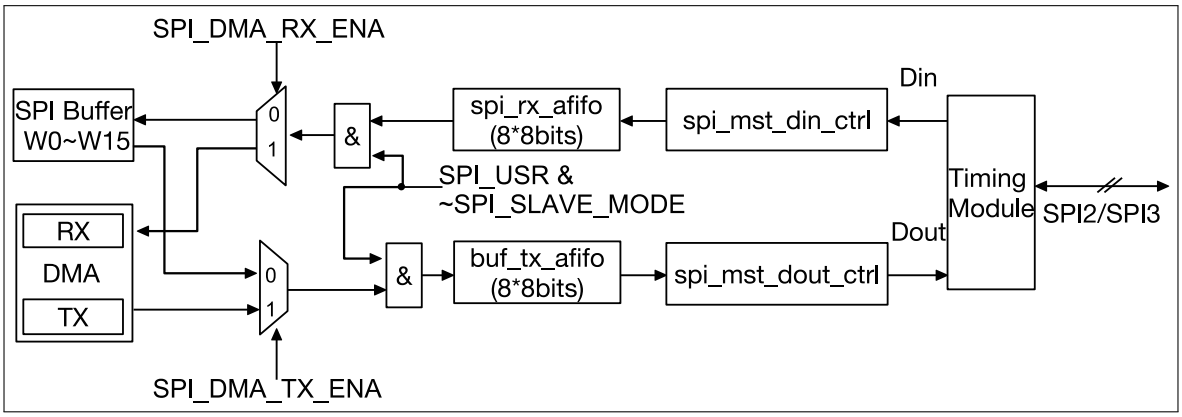


Figure 46.5-3. Data Flow Control in GP-SPI as Master

Figure 46.5-3 shows the data flow of GP-SPI as master. Its control logic is as follows:

- RX data: data bits in SPI2/SPI3 bus is captured by *Timing Module*, converted into bytes by *spi_mst_din_ctrl* module, then buffered in *spi_rx_afifo*, and finally stored in corresponding addresses according to the transfer types.
  - CPU-controlled transfer: the data is stored to registers SPI_W0_REG ~SPI_W15_REG.
  - DMA-controlled transfer: the data is stored to DMA RX buffer.
- TX data: the TX data is from corresponding addresses according to transfer modes and is saved to *buf_tx_afifo*.
  - CPU-controlled transfer: TX data is from SPI_W0_REG~SPI_W15_REG.
  - DMA-controlled transfer: TX data is from DMA TX buffer.

The data in *buf_tx_afifo* is sent out to Timing Module in 1/2/4/(8)-bit modes, controlled by GP-SPI state machine. The Timing Module can be used for timing compensation. For more information, see Section 46.8.

46.5.8.3 Data Flow Control as Slave

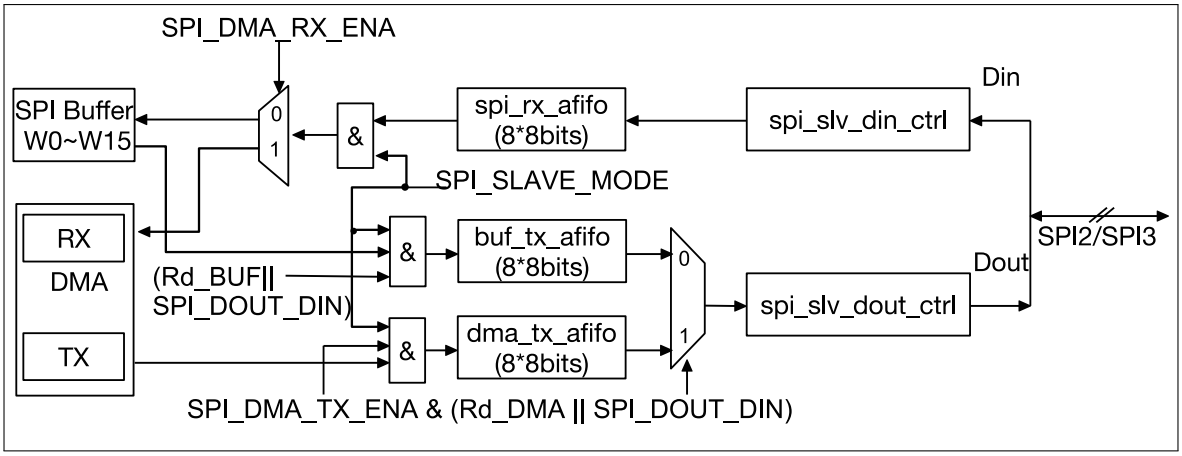


Figure 46.5-4. Data Flow Control in GP-SPI as Slave

Figure 46.5-4 shows the data flow of GP-SPI as slave. Its control logic is as follows:

- In CPU/DMA-controlled full-/half-duplex transfer, when an external SPI master starts the SPI transfer, data on the SPI2/SPI3 bus is captured, converted into bytes by the *spi_slv_din_ctrl* module, and then is stored in *spi_rx_afifo*.
  - In CPU-controlled full-duplex transfer, the received data in *spi_rx_afifo* will be later stored into registers SPI_W0_REG~SPI_W15_REG, successively.
  - In half-duplex Wr_BUF transfer, when the value of address (SLV_ADDR[7:0]) is received, the received data in *spi_rx_afifo* will be stored in the related address of registers SPI_W0_REG ~SPI_W15_REG.
  - In DMA-controlled full-duplex transfer or in half-duplex Wr_DMA transfer, the received data in spi_rx_afifo will be stored in the configured DMA RX buffer.
- In CPU-controlled full-/half-duplex transfer, the data to send is stored in *buf_tx_afifo*. In DMA-controlled full-/half-duplex transfer, the data to send is stored in *dma_tx_afifo*. Therefore, Rd_BUF transaction controlled by CPU and Rd_DMA transaction controlled by DMA can be done in one slave segmented transfer.
  - In CPU-controlled full-duplex transfer, when SPI_SLAVE_MODE and SPI_DOUTDIN are set and SPI_DMA_TX_ENA is cleared, the data in SPI_W0_REG~SPI_W15_REG will be stored into *buf_tx_afifo*.
  - In CPU-controlled half-duplex transfer, when SPI_SLAVE_MODE is set, SPI_DOUTDIN is cleared, Rd_BUF command and SLV_ADDR[7:0] are received, the data started from the related address of SPI_W0_REG~SPI_W15_REG will be stored into *buf_tx_afifo*.
  - In DMA-controlled full-duplex transfer, when SPI_SLAVE_MODE, SPI_DOUTDIN, and SPI_DMA_TX_ENA are set, the data in the configured DMA TX buffer will be stored into *dma_tx_afifo*.
  - In DMA-controlled half-duplex transfer, when SPI_SLAVE_MODE is set, SPI_DOUTDIN is cleared, and Rd_DMA command is received, the data in the configured DMA TX buffer will be stored into *dma_tx_afifo*.

The data in *buf_tx_afifo* or *dma_tx_afifo* is sent out by *spi_slv_dout_ctrl* module in 1/2/4(8)-bit modes.

46.5.9 GP-SPI as a Master

GP-SPI can be configured as a SPI master by clearing the bit SPI_SLAVE_MODE in SPI_SLAVE_REG. In this operation mode, GP-SPI provides clock signal (the divided clock from GP-SPI module clock) and CS signal.

46.5.9.1 State Machine

When GP-SPI works as a master, the state machine controls GP-SPI's various states during data transfer, including configuration (CONF), preparation (PREP), command (CMD), address (ADDR), dummy (DUMMY), data out (DOUT), and data in (DIN) states. GP-SPI is mainly used to access 1/2/4/8-bit SPI devices, such as flash and external RAM, thus the naming of GP-SPI states keeps consistent with the sequence naming of flash and external RAM. The meaning of each state is described as follows and Figure 46.5-5 shows the workflow of GP-SPI state machine.

1. IDLE: GP-SPI is not active or is operating as a slave.
2. CONF (valid for GP-SPI2 only): only used in DMA-controlled configurable segmented transfer. Set SPI_USR and SPI_USR_CONF to enable this state. If this state is not enabled, it means the current transfer is a single transfer.
3. PREP: prepare an SPI transaction and control SPI CS setup time. Set SPI_USR and SPI_CS_SETUP to enable this state.
4. CMD: send command sequence. Set SPI_USR and SPI_USR_COMMAND to enable this state.
5. ADDR: send address sequence. Set SPI_USR and SPI_USR_ADDR to enable this state.
6. DUMMY (wait cycle): send dummy sequence. set SPI_USR and SPI_USR_DUMMY to enable this state.
7. DATA: transfer data.
   - DOUT: send data sequence. Set SPI_USR and SPI_USR_MOSI to enable this state.
   - DIN: receive data sequence. Set SPI_USR and SPI_USR_MISO to enable this state.
8. DONE: control SPI CS hold time. Set SPI_USR to enable this state.

Note:

To start this state machine, set SPI_USR first. SPI_MST_FD_WAIT_DMA_TX_DATA controls when SPI_USR takes effect:

- 0: the configured state takes effect immediately after SPI_USR and other control registers are configured.
- 1: if DOUT state is configured, the SPI_USR and other control registers will take effect, and the state machine will start, only when the data is ready in *buf_tx_afifo*.

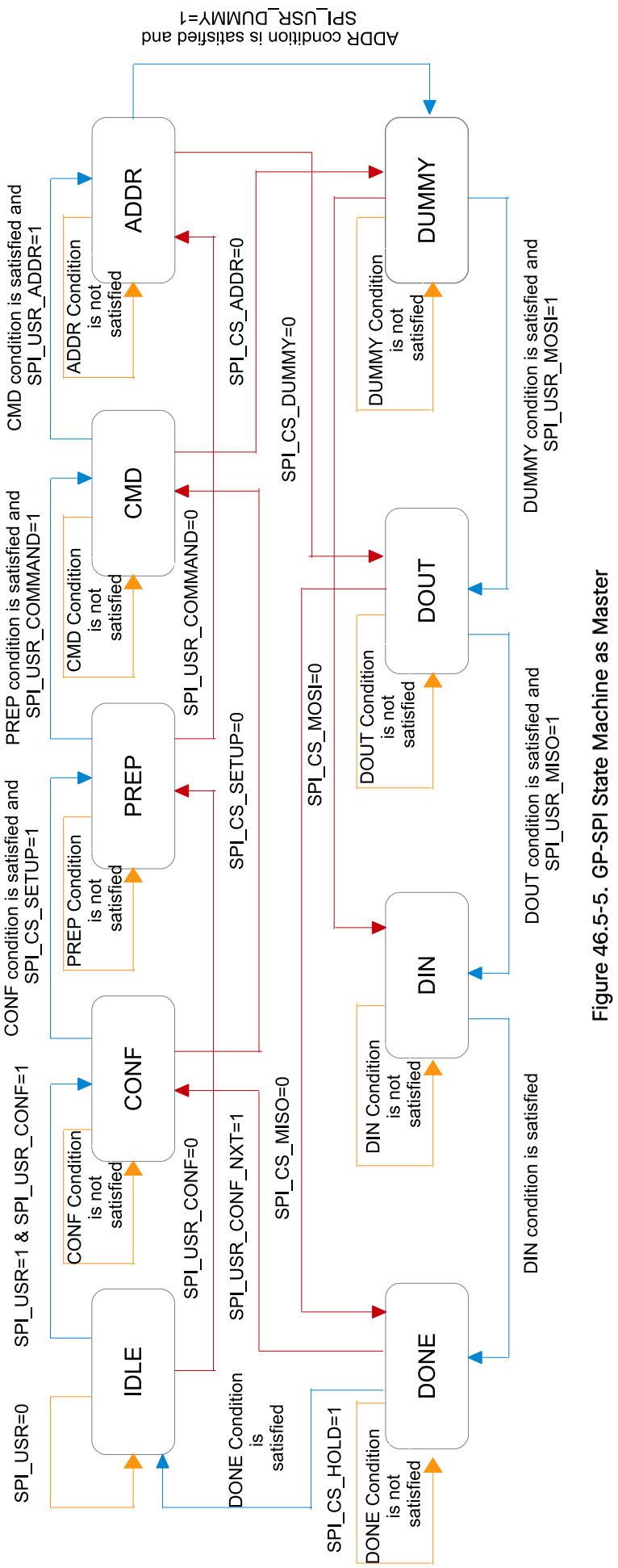


Figure 46.5-5. GP-SPI State Machine as Master

Legend to state flow:

- —: indicates corresponding state condition is not satisfied; repeats current state.
- —: corresponding registers are set and conditions are satisfied; goes to next state.
- —: state registers are not set; skips one or more following states, depending on the registers of the following states are set or not.

Explanation to the conditions listed in the figure above:

- CONF condition: gpc[17:0] >= SPI_CONF_BITLEN[17:0]
- PREP condition: gpc[4:0] >= SPI_CS_SETUP_TIME[4:0]
- CMD condition: gpc[3:0] >= SPI_USR_COMMAND_BITLEN[3:0]
- ADDR condition: gpc[4:0] >= SPI_USR_ADDR_BITLEN[4:0]
- DUMMY condition: gpc[7:0] >= SPI_USR_DUMMY_CYCLELEN[7:0]
- DOUT condition: gpc[17:0] >= SPI_MS_DATA_BITLEN[17:0]
- DIN condition: gpc[17:0] >= SPI_MS_DATA_BITLEN[17:0]
- DONE condition: (gpc[4:0] >= SPI_CS_HOLD_TIME[4:0] || SPI_CS_HOLD == 1'b0)

A counter (gpc[17:0]) is used in the state machine to control the cycle length of each state. The states CONF, PREP, CMD, ADDR, DUMMY, DOUT, and DIN can be enabled or disabled independently. The cycle length of each state can also be configured independently.

46.5.9.2 Register Configuration for State and Bit Mode Control

Introduction

The registers, related to GP-SPI state control, are listed in Table 46.5-10. Users can enable QPI mode for GP-SPI by setting the bit SPI_QPI_MODE in register SPI_USER_REG.

Table 46.5-10. Registers Used for State Control in 1/2/4/8-bit Modes

State	Control Registers for 1-bit Mode SPI2/SPI3 Bus	Control Registers for 2-bit Mode SPI2/SPI3 Bus	Control Registers for 4-bit Mode SPI2/SPI3 Bus	Control Registers for 8-bit Mode SPI2 Bus
CMD	SPI_USR_COMMAND_VALUE SPI_USR_COMMAND_BITLEN SPI_USR_COMMAND	SPI_USR_COMMAND_VALUE SPI_USR_COMMAND_BITLEN SPI_FCMD_DUAL SPI_USR_COMMAND	SPI_USR_COMMAND_VALUE SPI_USR_COMMAND_BITLEN SPI_FCMD_QUAD SPI_USR_COMMAND	SPI_USR_COMMAND_VALUE SPI_USR_COMMAND_BITLEN SPI_FCMD_OCT SPI_USR_COMMAND
ADDR	SPI_USR_ADDR_VALUE SPI_USR_ADDR_BITLEN SPI_USR_ADDR	SPI_USR_ADDR_VALUE SPI_USR_ADDR_BITLEN SPI_USR_ADDR SPI_FADDR_DUAL	SPI_USR_ADDR_VALUE SPI_USR_ADDR_BITLEN SPI_USR_ADDR SPI_FADDR_QUAD	SPI_USR_ADDR_VALUE SPI_USR_ADDR_BITLEN SPI_USR_ADDR SPI_FADDR_OCT
DUMMY	SPI_USR_DUMMY_CYCLELEN SPI_USR_DUMMY	SPI_USR_DUMMY_CYCLELEN SPI_USR_DUMMY	SPI_USR_DUMMY_CYCLELEN SPI_USR_DUMMY	SPI_USR_DUMMY_CYCLELEN SPI_USR_DUMMY
DIN	SPI_USR_MISO SPI_MS_DATA_BITLEN	SPI_USR_MISO SPI_MS_DATA_BITLEN SPI_FREAD_DUAL	SPI_USR_MISO SPI_MS_DATA_BITLEN SPI_FREAD_QUAD	SPI_USR_MISO SPI_MS_DATA_BITLEN SPI_FREAD_OCT
DOUT	SPI_USR_MOSI SPI_MS_DATA_BITLEN	SPI_USR_MOSI SPI_MS_DATA_BITLEN SPI_FWRITE_DUAL	SPI_USR_MOSI SPI_MS_DATA_BITLEN SPI_FWRITE_QUAD	SPI_USR_MOSI SPI_MS_DATA_BITLEN SPI_FWRITE_OCT

As shown in Table 46.5-10, the registers in each cell should be configured to set the SPI2/SPI3 bus to corresponding bit mode, i.e., the mode shown in the table header, at a specific state (corresponding to the first column).

Configuration

For instance, when GP-SPI2 reads data, and

- CMD is in 1-bit mode
- ADDR is in 2-bit mode
- DUMMY lasts for 8 clock cycles
- DIN is in 4-bit mode

The register configuration can be as follows:

1. Configure CMD state related registers.
   - Configure the required command value in SPI_USR_COMMAND_VALUE.
   - Configure command bit length in SPI_USR_COMMAND_BITLEN. SPI_USR_COMMAND_BITLEN = expected bit length - 1.
   - Set SPI_USR_COMMAND.
   - Clear SPI_FCMD_DUAL, SPI_FCMD_QUAD, and SPI_FCMD_OCT.
2. Configure ADDR state related registers.
   - Configure the required address value in SPI_USR_ADDR_VALUE.
   - Configure address bit length in SPI_USR_ADDR_BITLEN. SPI_USR_ADDR_BITLEN = expected bit length - 1.
   - Set SPI_USR_ADDR and SPI_FADDR_DUAL.
   - Clear SPI_FADDR_QUAD and SPI_FCMD_OCT.
3. Configure DUMMY state related registers.
   - Configure DUMMY cycles in SPI_USR_DUMMY_CYCLELEN. SPI_USR_DUMMY_CYCLELEN = expected clock cycles - 1.
   - Set SPI_USR_DUMMY.
4. Configure DIN state related registers.
   - Configure read data bit length in SPI_MS_DATA_BITLEN. SPI_MS_DATA_BITLEN = expected bit length - 1.
   - Set SPI_FREAD_QUAD and SPI_USR_MISO.
   - Clear SPI_FREAD_DUAL and SPI_FCMD_OCT.
   - Configure DMA for DMA-controlled transfer. For CPU controlled transfer, no action is needed.
5. Clear SPI_USR_MOSI.
6. Set SPI_DMA_AFIFO_RST, SPI_BUF_AFIFO_RST, and SPI_RX_AFIFO_RST to reset these buffers.
7. Set SPI_USR to start GP-SPI2 transfer.

Note:
Updating the configuration when the GP-SPI works as master described in this and subsequent sections requires to set SPI_UPDATE accordingly to synchronize the configuration from AHB_CLK domain to clk_spi_mst domain. For more detailed configuration, see the sections above. No operation is required when the GP-SPI works as slave.

When writing data (DOUT state), SPI_USR_MOSI should be configured instead, while SPI_USR_MISO should be cleared. The output data bit length is the value of SPI_MS_DATA_BITLEN + 1. Output data should be configured in GP-SPI2 data buffer (SPI_W0_REG~SPI_W15_REG) for CPU-controlled transfer, or DMA TX buffer for DMA-controlled transfer.

Pay special attention to the command value in SPI_USR_COMMAND_VALUE and to address value in SPI_USR_ADDR_VALUE.

The configuration of command value is as follows:

Table 46.5-11. Sending Sequence of Command Value

COMMAND_BITLEN[1]	COMMAND_VALUE[2]	BIT_ORDER[3]	Sending Sequence of Command Value
0~7	[7:0]	1	COMMAND_VALUE[COMMAND_BITLEN:0] is sent first.
		0	COMMAND_VALUE[7:7−COMMAND_BITLEN] is sent first.
8~15	[15:0]	1	COMMAND_VALUE[7:0] is sent first, and then COMMAND_VALUE[COMMAND_BITLEN:8].
		0	COMMAND_VALUE[7:0] is sent first, and then COMMAND_VALUE[15:15−COMMAND_BITLEN].

[1] SPI_USR_COMMAND_BITLEN: this field is used to configure the bit length of the command.
[2] SPI_USR_COMMAND_VALUE: command value is written into this field. For which part of this field is used, see the table above.
[3] SPI_WR_BIT_ORDER: 0: LSB first; 1: MSB first.

The configuration of address value is as follows:

Table 46.5-12. Sending Sequence of Address Value

ADDR_BITLEN[1]	ADDR_VALUE[2]	BIT_ORDER[3]	Sending Sequence of Address Value
0~7	[31:24]	1	COMMAND_VALUE[ADDR_BITLEN + 24:24] is sent first.
		0	ADDR_VALUE[31:31−ADDR_BITLEN] is sent first.
8~15	[31:16]	1	ADDR_VALUE[31:24] is sent first, and then ADDR_VALUE[ADDR_BITLEN + 8:16].
		0	ADDR_VALUE[31:24] is sent first, and then ADDR_VALUE[23:31−ADDR_BITLEN].
16~23	[31:8]	1	ADDR_VALUE[31:16] is sent first, and then ADDR_VALUE[ADDR_BITLEN−8:8].

		0	ADDR_VALUE[31:16] is sent first, and then ADDR_VALUE[15:31−ADDR_BITLEN].
24∼31	[31:0]	1	ADDR_VALUE[31:8] is sent first, and then ADDR_VALUE[ADDR_BITLEN−24:0].
		0	ADDR_VALUE[31:8] is sent first, and then ADDR_VALUE[7:31−ADDR_BITLEN].

[1] SPI_USR_ADDR_BITLEN: this field is used to configure the bit length of the address.
[2] SPI_USR_ADDR_VALUE: address value is written into this field. For which part of this field is used, see the table above.
[3] SPI_WR_BIT_ORDER: 0: LSB first; 1: MSB first.

46.5.9.3 Full-Duplex Communication (1-bit Mode Only)

Introduction

GP-SPI supports SPI full-duplex communication. In this mode, SPI master provides CLK and CS signals, exchanging data with SPI slave in 1-bit mode via MOSI (SPI2D/SPI3_D, sending) and MISO (SPI2Q/SPI3_Q, receiving) at the same time. To enable this communication mode, set the bit SPI_DOUTDIN in register SPI_USER_REG. Figure 46.5-6 illustrates the connection of GP-SPI2 with its slave in full-duplex communication.

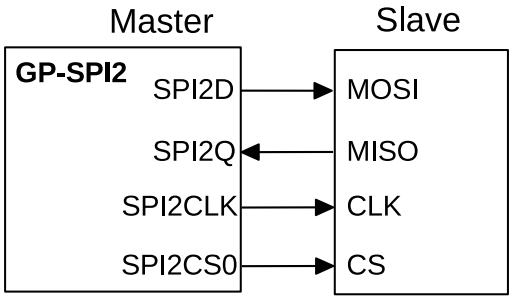


Figure 46.5-6. Full-Duplex Communication Between GP-SPI2 Master and a Slave

In full-duplex communication, the behavior of states CMD, ADDR, DUMMY, DOUT and DIN are configurable. Usually, the states CMD, ADDR and DUMMY are not used in this communication. The bit length of transferred data is configured in SPI_MS_DATA_BITLEN. The actual bit length used in communication equals to (SPI_MS_DATA_BITLEN + 1).

Configuration (Take GP-SPI2 as an example)

To start a data transfer, follow the steps below:

- Configure the IO path via IO MUX or GPIO matrix between GP-SPI2 and an external SPI device.
- Configure AHB clock (AHB_CLK, see Chapter 11 *Reset and Clock*) and module clock (clk_spi_mst) for the GP-SPI2 module.
- Set SPI_DOUTDIN and clear SPI_SLAVE_MODE, to enable master full-duplex communication.
- Configure GP-SPI2 registers listed in Table 46.5-10.
- Configure SPI CS setup time and hold time according to Section 46.6.

- Set the property of SPI2CLK according to Section 46.7.
- Prepare data according to the selected transfer type:
  - In CPU-controlled MOSI transfer, prepare data in registers SPI_W0_REG~SPI_W15_REG.
  - In DMA-controlled transfer,
    * configure SPI_DMA_TX_ENA/SPI_DMA_RX_ENA,
    * configure DMA TX/RX link,
    * and start DMA TX/RX engine, as described in Section 46.5.7 and Section 46.5.8.
- Configure interrupts and wait for SPI slave to get ready for transfer.
- Set SPI_DMA_AFIFO_RST, SPI_BUF_AFIFO_RST, and SPI_RX_AFIFO_RST to reset these buffers.
- Set SPI_USR in register SPI_CMD_REG to start the transfer and wait for the configured interrupts.

46.5.9.4 Half-Duplex Communication (1/2/4/(8)-bit Mode)

Introduction

In this mode, GP-SPI provides CLK and CS signals. Only one side (SPI master or slave) can send data at a time, while the other side receives the data. To enable this communication mode, clear the bit SPI_DOUTDIN in register SPI_USER_REG. The standard format of SPI half-duplex communication is CMD + [ADDR +] [DUMMY +] [DOUT or DIN]. The states ADDR, DUMMY, DOUT, and DIN are optional, and can be disabled or enabled independently.

As described in Section 46.5.9.2, the properties of GP-SPI states: CMD, ADDR, DUMMY, DOUT and DIN, such as cycle length, value, and parallel bus bit mode, can be set independently. For the register configuration, see Table 46.5-10.

The detailed properties of half-duplex GP-SPI are as follows:

1. CMD: 0~16 bits, master output, slave input.
2. ADDR: 0~32 bits, master output, slave input.
3. DUMMY: 0~256 SPI2CLK/SPI3_CLK cycles, master output, slave input.
4. DOUT: 0~512 bits (64 bytes) in CPU-controlled transfer, 0~256 Kbits (32 KB) in DMA-controlled single transfer, and unlimited length in DMA-controlled configurable segmented transfer, master output, slave input.
5. DIN: 0~512 bits (64 bytes) in CPU-controlled transfer, 0~256 Kbits (32 KB) in DMA-controlled single transfer, and unlimited length in DMA-controlled configurable segmented transfer, slave output, master input.

GP-SPI2 also supports Double Transfer Rate (DTR) in 1/2/4/8-bit modes, in which data is sent and received at the rising and the falling edges of SPI clock:

- If SPI_CMD_DTR_EN is set, the CMD value will be sent in DTR mode; otherwise CMD phase will be in Single Transfer Rate (STR) mode.
- If SPI_ADDR_DTR_EN is set, the ADDR value will be sent in DTR mode; otherwise ADDR phase will be in STR mode.

- If SPI_DATA_DTR_EN is set, the input data in state DIN or output data in state DOUT will be sent in DTR mode; otherwise the data will be in STR mode.

The control bits SPI_CMD_DTR_EN, SPI_ADDR_DTR_EN, and SPI_DATA_DTR_EN can be configured independently, which means CMD in STR mode, ADDR and DOUT or DIN in DTR mode are supported.

GP-SPI2 can only output SPI2DQS signal, but can not receive the signal. Therefore, only flash or external RAM working in fixed dummy mode (dummy cycles in a read sequence is fixed) are supported.

Configuration (Take GP-SPI2 as an example)

The register configuration can be as follows:

1. Configure the IO path via HP IO MUX or HP GPIO matrix between GP-SPI2 and an external SPI device.
2. Configure AHB clock (AHB_CLK) and module clock (clk_spi_mst) for the GP-SPI2 module.
3. Clear SPI_DOUTDIN and SPI_SLAVE_MODE, to enable master half-duplex communication.
4. Configure GP-SPI2 registers listed in Table 46.5-10.
5. Configure SPI CS setup time and hold time according to Section 46.6.
6. Set the property of SPI2CLK according to Section 46.7.
7. Prepare data according to the selected transfer type:
   - In CPU-controlled MOSI transfer, prepare data in registers SPI_W0_REG~SPI_W15_REG.
   - In DMA-controlled transfer,
     - configure SPI_DMA_TX_ENA/SPI_DMA_RX_ENA,
     - configure DMA TX/RX link,
     - and start DMA TX/RX engine, as described in Section 46.5.7 and Section 46.5.8.
8. Configure interrupts and wait for SPI slave to get ready for transfer.
9. Set SPI_DMA_AFIFO_RST, SPI_BUF_AFIFO_RST, and SPI_RX_AFIFO_RST to reset these buffers.
10. Set SPI_USR in register SPI_CMD_REG to start the transfer and wait for the configured interrupts.

Application Example (Take GP-SPI2 as an example)

The following example shows how GP-SPI2 accesses flash and external RAM in master half-duplex communication.

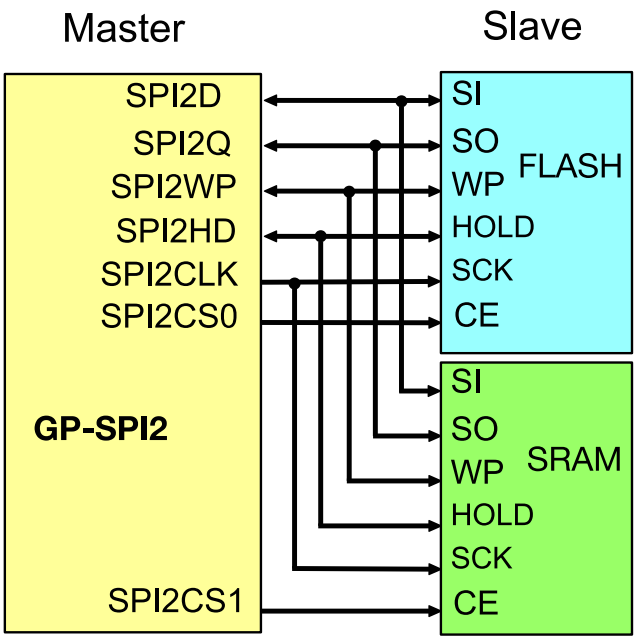


Figure 46.5-7. Connection of GP-SPI2 to Flash and External RAM in 4-bit Mode

Figure 46.5-8 indicates GP-SPI2 Quad I/O Read sequence according to standard flash specification. Other GP-SPI2 command sequences are implemented in accordance with the requirements of SPI slaves.

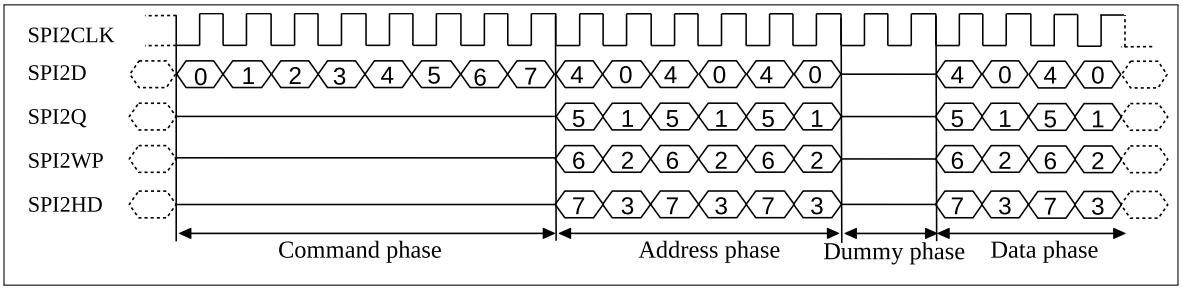


Figure 46.5-8. SPI Quad I/O Read Command Sequence Sent by GP-SPI2 to Flash

46.5.9.5 DMA-Controlled Configurable Segmented Transfer

Note:

- LP-SPI and GP-SPI3 do not support DMA-controlled configurable segmented transfer.
- Note that there is no separate section on how to configure a single transfer as master, since the CONF state of a configurable segmented transfer can be skipped to implement a single transfer.

Introduction

When GP-SPI2 works as a master, it provides a feature named configurable segmented transfer controlled by DMA.

A DMA-controlled transfer as master can be:

- a single transfer, consisting of only one transaction;
- or a configurable segmented transfer, consisting of several transactions (segments).

In a configurable segmented transfer, the registers of each single transaction (segment) are configurable. This feature enables GP-SPI2 to do as many transactions (segments) as configured after such transfer is triggered once by the CPU. Figure 46.5-9 shows how this feature works.

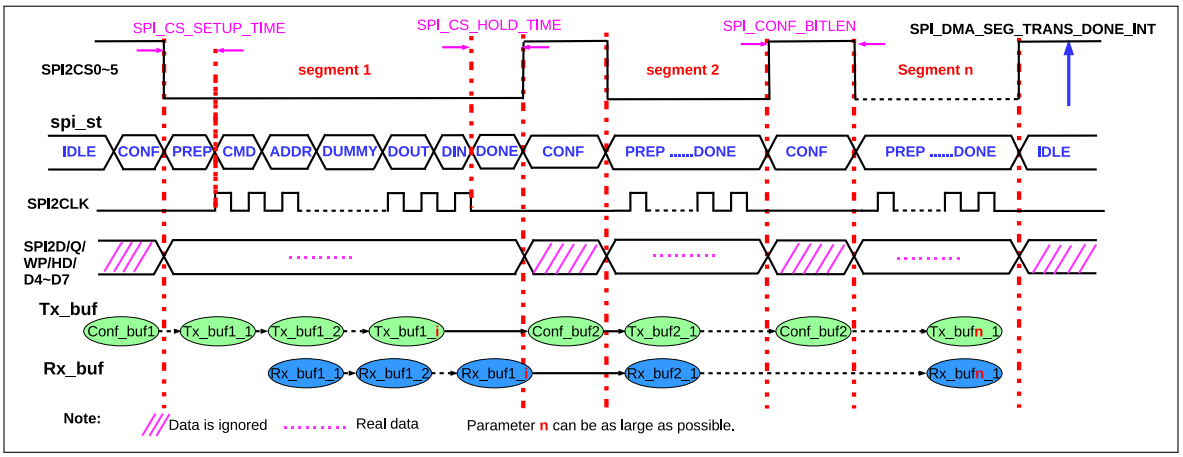


Figure 46.5-9. Configurable Segmented Transfer as Master

As shown in Figure 46.5-9, the registers for one transaction (segment *n*) can be reconfigured by GP-SPI2 hardware according to the content in its Conf_buf*n* during the CONF state, before this segment starts.

It is recommended to provide separate DMA CONF links and CONF buffers (Conf_buf*i* in Figure 46.5-9) for each CONF state. A DMA TX link is used to connect all the CONF buffers and TX data buffers (Tx_buf*i* in Figure 46.5-9) into a chain. Hence, the behavior of the SPI2 bus in each segment can be controlled independently.

For example, in a configurable segmented transfer, its segment *i*, segment *j*, and segment *k* can be configured to full-duplex, half-duplex MISO, and half-duplex MOSI, respectively. *i*, *j*, and *k* represent different segment numbers.

Meanwhile, the state of GP-SPI2, the data length and cycle length of the SPI2 bus, and the behavior of the DMA, can be configured independently for each segment. When this whole DMA-controlled transfer (consisting of several segments) has finished, a GP-SPI2 interrupt, SPI_DMA_SEG_TRANS_DONE_INT, is triggered.

Configuration

1. Configure the IO path via HP IO MUX or HP GPIO matrix between GP-SPI2 and an external SPI device.
2. Configure AHB clock (AHB_CLK) and module clock (clk_spi_mst) for the GP-SPI2 module.
3. Clear SPI_DOUTDIN and SPI_SLAVE_MODE, to enable master half-duplex communication.
4. Configure GP-SPI2 registers listed in Table 46.5-10.
5. Configure SPI CS setup time and hold time according to Section 46.6.
6. Set the property of SPI2CLK according to Section 46.7.
7. Prepare descriptors for DMA CONF buffer and TX data (optional) for each segment. Chain the descriptors of CONF buffer and TX buffers of several segments into one linked list.
8. Similarly, prepare descriptors for RX buffers for each segment and chain them into one linked list.

9. Configure all the needed CONF buffers, TX buffers and RX buffers, respectively for each segment before this DMA-controlled transfer begins.
10. Point AXI_DMA_OUTLINK_ADDR_CH*n* to the head address of the CONF and TX buffer descriptor linked list, and then set AXI_DMA_OUTLINK_START_CH*n* to start the TX DMA.
11. Clear the bit SPI_RX_EOF_EN in register SPI_DMA_CONF_REG. Point AXI_DMA_INLINK_ADDR_CH*n* to the head address of the CONF and RX buffer descriptor linked list, and then set AXI_DMA_INLINK_START_CH*n* to start the RX DMA.
12. Set SPI_USR_CONF to enable CONF state.
13. Set SPI_DMA_SEG_TRANS_DONE_INT_ENA to enable the SPI_DMA_SEG_TRANS_DONE_INT interrupt. Configure other interrupts if needed according to Section 46.11.
14. Wait for all the slaves to get ready for transfer.
15. Set SPI_DMA_AFIFO_RST, SPI_BUF_AFIFO_RST, and SPI_RX_AFIFO_RST to reset these buffers.
16. Set SPI_USR to start this DMA-controlled transfer.
17. Wait for SPI_DMA_SEG_TRANS_DONE_INT interrupt, which means this transfer has finished and the data has been stored into corresponding memory.

Note:

Prepare the data to send for each segment in its PREP state. Shall ensure that:

$$(SPI_CS_SETUP_TIME + 1) \times T_{SPI_CLK} >= 4 \times (T_{AHB_CLK} + Tclk_spi_mst)$$

Configuration of CONF Buffer and Magic Value

In a configurable segmented transfer, only registers which change from the last transaction (segment) need to be re-configured to new values in CONF state. The configuration of other registers can be skipped (i.e., kept the same) to save time and chip resources.

The first word in DMA CONF buffer*i*, called SPI_BIT_MAP_WORD, defines whether given GP-SPI2 register is to be updated or not in segment *i*. The relation of SPI_BIT_MAP_WORD and GP-SPI2 registers to update can be seen in Table 46.5-13 Bitmap (BM) Table. If a bit in the BM table is set to 1, its corresponding register value will be updated in this segment. Otherwise, if some registers should be kept from being changed, the related bits should be set to 0.

Table 46.5-13. BM Table for CONF State

BM Bit	Register	BM Bit	Register
0	SPI_ADDR_REG	7	SPI_MISC_REG
1	SPI_CTRL_REG	8	SPI_DIN_MODE_REG
2	SPI_CLOCK_REG	9	SPI_DIN_NUM_REG
3	SPI_USER_REG	10	SPI_DOUT_MODE_REG
4	SPI_USER1_REG	11	SPI_DMA_CONF_REG
5	SPI_USER2_REG	12	SPI_DMA_INT_ENA_REG
6	SPI_MS_DLEN_REG	13	SPI_DMA_INT_CLR_REG

Then new values of all the registers to be modified should be placed right after SPI_BIT_MAP_WORD, in consecutive words in the CONF buffer.

To ensure the correctness of the content in each CONF buffer, the value in SPI_BIT_MAP_WORD[31:28] is used as a "magic value", and will be compared with SPI_DMA_SEG_MAGIC_VALUE in register SPI_SLAVE_REG. The value of SPI_DMA_SEG_MAGIC_VALUE should be configured before this DMA-controlled transfer starts, and can not be changed during these segments.

- If SPI_BIT_MAP_WORD[31:28] == SPI_DMA_SEG_MAGIC_VALUE, this DMA-controlled transfer continues normally. SPI_DMA_SEG_TRANS_DONE_INT is triggered at the end of this DMA-controlled transfer.
- If SPI_BIT_MAP_WORD[31:28] != SPI_DMA_SEG_MAGIC_VALUE, GP-SPI2 state (spi_st) goes back to IDLE and the transfer is ended immediately. The interrupt SPI_DMA_SEG_TRANS_DONE_INT is still triggered, with SPI_SEG_MAGIC_ERR_INT_RAW bit set to 1.

CONF Buffer Configuration Example

Table 46.5-14 and Table 46.5-15 provide an example to show how to configure a CONF buffer for a transaction (segment*i*) in which SPI_ADDR_REG, SPI_CTRL_REG, SPI_CLOCK_REG, SPI_USER_REG, and SPI_USER1_REG need to be updated.

Table 46.5-14. An Example of CONF buffer*i* in Segment *i*

CONF buffer*i*	Note
SPI_BIT_MAP_WORD	The first word in this buffer. Its value is 0xA000001F in this example when the SPI_DMA_SEG_MAGIC_VALUE is set to 0xA. As shown in Table 46.5-15, bits 0, 1, 2, 3, and 4 are set, indicating the following registers will be updated.
SPI_ADDR_REG	The second word, stores the new value to SPI_ADDR_REG.
SPI_CTRL_REG	The third word, stores the new value to SPI_CTRL_REG.
SPI_CLOCK_REG	The fourth word, stores the new value to SPI_CLOCK_REG.
SPI_USER_REG	The fifth word, stores the new value to SPI_USER_REG.
SPI_USER1_REG	The sixth word, stores the new value to SPI_USER1_REG.

Table 46.5-15. BM Bit Value and Register to Be Updated in This Example

BM Bit	Value	Register	BM Bit	Value	Register
0	1	SPI_ADDR_REG	7	0	SPI_MISC_REG
1	1	SPI_CTRL_REG	8	0	SPI_DIN_MODE_REG
2	1	SPI_CLOCK_REG	9	0	SPI_DIN_NUM_REG
3	1	SPI_USER_REG	10	0	SPI_DOUT_MODE_REG
4	1	SPI_USER1_REG	11	0	SPI_DMA_CONF_REG
5	0	SPI_USER2_REG	12	0	SPI_DMA_INT_ENA_REG
6	0	SPI_MS_DLEN_REG	13	0	SPI_DMA_INT_CLR_REG

Notes

In a DMA-controlled configurable segmented transfer, please pay special attention to the following bits:

- SPI_USR_CONF: set SPI_USR_CONF before SPI_USR is set, to enable this transfer.
- SPI_USR_CONF_NXT: if segment *i* is not the final transaction of this whole DMA-controlled transfer, its SPI_USR_CONF_NXT bit should be set to 1.
- SPI_CONF_BITLEN: GP-SPI2 CS setup time and hold time are programmable independently in each segment, see Section 46.6 for detailed configuration. The CS high time in each segment is about:

$$(SPI_CONF_BITLEN + 5) \times T_{AHB_CLK}$$

  The CS high time in CONF state can be set from 62.5 ns~3.2768 ms when $f_{\text{AHB_CLK}}$ is 80 MHz. (SPI_CONF_BITLEN + 5) will overflow from (0x40000 - SPI_CONF_BITLEN - 5) if SPI_CONF_BITLEN is larger than 0x3FFFA.

46.5.10 GP-SPI Works as a Slave

GP-SPI can be used as a slave to communicate with an SPI master. As a slave, GP-SPI supports 1-bit SPI, 2-bit dual SPI, 4-bit quad SPI, and QPI modes, with specific communication formats. To enable this mode, set SPI_SLAVE_MODE in register SPI_SLAVE_REG.

The CS signal must be held low during the transfer, and its falling/rising edges indicate the start/end of a single or segmented transfer. Take CS low-level effective as an example, when GP-SPI is used as a slave, the CS invalid duration (high-level) between each SPI transfer should not be less than 8 T_{AHB_CLK} to ensure that each transfer can end normally.

46.5.10.1 Configurable Communication Formats

When GP-SPI works as a slave, SPI full-duplex and half-duplex communications are available. To select from the two communications, configure SPI_DOUTDIN in register SPI_USER_REG.

Full-duplex communication means that input data and output data are transmitted simultaneously throughout the entire transaction. All bits are treated as input or output data, which means no command, address or dummy states are expected. The interrupt SPI_TRANS_DONE_INT is triggered once the transaction ends.

In half-duplex communication, the format is CMD+ADDR+DUMMY+DATA (DIN or DOUT).

- "DIN" means that an SPI master reads data from GP-SPI.
- "DOUT" means that an SPI master writes data to GP-SPI.

The detailed properties of each state are as follows:

1. CMD:
   - Indicate the function of SPI slave.
   - One byte from master to slave.
   - Only the values in Table 46.5-16 and Table 46.5-17 are valid.
   - Can be sent in 1-bit SPI mode or 4-bit QPI mode.
2. ADDR:
   - The address for Wr_BUF and Rd_BUF commands in CPU-controlled transfer, or placeholder bits in other transfers and can be defined by application.

- One byte from master to slave.
- Can be sent in 1-bit, 2-bit or 4-bit modes according to the command.

3. DUMMY:
   - Its value is meaningless. SPI slave prepares data in this state.
   - Bit mode of SPI2 bus is also meaningless here.
   - Last for eight SPI_CLK cycles.
4. DIN or DOUT:
   - Data length can be 0~64 bytes in CPU-controlled transfer and unlimited in DMA-controlled transfer.
   - Can be sent in 1-bit, 2-bit or 4-bit modes according to the CMD value.

Note:
The states of ADDR and DUMMY can never be skipped in any half-duplex communications.

When a half-duplex transaction is complete, the transferred CMD and ADDR values are latched into SPI_SLV_LAST_COMMAND and SPI_SLV_LAST_ADDR, respectively. SPI_SLV_CMD_ERR_INT_RAW will be set if the transferred CMD value is not supported by GP-SPI as slave. SPI_SLV_CMD_ERR_INT_RAW can only be cleared by software.

46.5.10.2 CMD Values Supported in Half-Duplex Communication

In half-duplex communication, the defined values of CMD determine the transfer types. Unsupported CMD values are disregarded, meanwhile the related transfer is ignored and SPI_SLV_CMD_ERR_INT_RAW is set. The transfer format is CMD (8 bits) + ADDR (8 bits) + DUMMY (8 SPI_CLK cycles) + DATA (unit in bytes). The detailed description of CMD[3:0] is as follows:

- 0x1 (Wr_BUF): CPU-controlled write operation. Master sends data and GP-SPI receives data. The data is stored in the related address of SPI_W0_REG~SPI_W15_REG.
- 0x2 (Rd_BUF): CPU-controlled read operation. Master receives the data sent by GP-SPI. The data comes from the related address of SPI_W0_REG~SPI_W15_REG.
- 0x3 (Wr_DMA): DMA-controlled write operation. Master sends data and GP-SPI receives data. The data is stored in GP-SPI DMA RX buffer.
- 0x4 (Rd_DMA): DMA-controlled read operation. Master receives the data sent by GP-SPI. The data comes from GP-SPI DMA TX buffer.
- 0x7 (CMD7): used to generate an SPI_SLV_CMD7_INT interrupt. It can also generate a AXI_DMA_IN_SUC_EOF_CH*n*_INT interrupt in a slave segmented transfer when DMA RX link is used. But it will not end GP-SPI's slave segmented transfer.
- 0x8 (CMD8): only used to generate an SPI_SLV_CMD8_INT interrupt, which will not end GP-SPI's slave segmented transfer.
- 0x9 (CMD9): only used to generate an SPI_SLV_CMD9_INT interrupt, which will not end GP-SPI's slave segmented transfer.

- 0xA (CMDA): only used to generate an SPI_SLV_CMDA_INT interrupt, which will not end GP-SPI's slave segmented transfer.

CMD7, CMD8, CMD9, and CMDA commands are reserved for user definition. These commands can be used as handshake signals, as passwords of some specific functions, as trigger signals of some user defined actions, and so on.

1/2/4-bit modes in states of CMD, ADDR, DATA are supported, which are determined by value of CMD[7:4]. The DUMMY state is always in 1-bit mode and lasts for eight SPI_CLK cycles. The definition of CMD[7:4] is as follows:

- 0x0: CMD, ADDR, and DATA states all are in 1-bit mode.
- 0x1: CMD and ADDR are in 1-bit mode. DATA is in 2-bit mode.
- 0x2: CMD and ADDR are in 1-bit mode. DATA is in 4-bit mode.
- 0x5: CMD is in 1-bit mode. ADDR and DATA are in 2-bit mode.
- 0xA: CMD is in 1-bit mode, ADDR and DATA are in 4-bit mode or in QPI mode.

In addition, if the value of CMD[7:0] is 0x05, 0xA5, 0x06, or 0xDD, DUMMY and DATA states are skipped. The definition of CMD[7:0] is as follows:

- 0x05 (End_SEG_TRANS): master sends this command to end slave segmented transfer in SPI mode.
- 0xA5 (End_SEG_TRANS): master sends this command to end slave segmented transfer in QPI mode.
- 0x06 (En_QPI): GP-SPI enters QPI mode when receiving this command and the bit SPI_QPI_MODE in register SPI_USER_REG is set.
- 0xDD (Ex_QPI): GP-SPI exits QPI mode when receiving this command and the bit SPI_QPI_MODE is cleared.

All the CMD values supported by GP-SPI are listed in Table 46.5-16 and Table 46.5-17. Note that the DUMMY state is always in 1-bit mode and lasts for eight SPI_CLK cycles.

Table 46.5-16. CMD Values Supported in SPI Mode

Transfer Type	CMD[7:0]	CMD State	ADDR State	DATA State
Wr_BUF	0x01	1-bit mode	1-bit mode	1-bit mode
	0x11	1-bit mode	1-bit mode	2-bit mode
	0x21	1-bit mode	1-bit mode	4-bit mode
	0x51	1-bit mode	2-bit mode	2-bit mode
	0xA1	1-bit mode	4-bit mode	4-bit mode
Rd_BUF	0x02	1-bit mode	1-bit mode	1-bit mode
	0x12	1-bit mode	1-bit mode	2-bit mode
	0x22	1-bit mode	1-bit mode	4-bit mode
	0x52	1-bit mode	2-bit mode	2-bit mode
	0xA2	1-bit mode	4-bit mode	4-bit mode
Wr_DMA	0x03	1-bit mode	1-bit mode	1-bit mode
	0x13	1-bit mode	1-bit mode	2-bit mode
	0x23	1-bit mode	1-bit mode	4-bit mode
	0x53	1-bit mode	2-bit mode	2-bit mode

Table 46.5-16. CMD Values Supported in SPI Mode

Transfer Type	CMD[7:0]	CMD State	ADDR State	DATA State
	0xA3	1-bit mode	4-bit mode	4-bit mode
Rd_DMA	0x04	1-bit mode	1-bit mode	1-bit mode
	0x14	1-bit mode	1-bit mode	2-bit mode
	0x24	1-bit mode	1-bit mode	4-bit mode
	0x54	1-bit mode	2-bit mode	2-bit mode
	0xA4	1-bit mode	4-bit mode	4-bit mode
CMD7	0x07	1-bit mode	1-bit mode	-
	0x17	1-bit mode	1-bit mode	-
	0x27	1-bit mode	1-bit mode	-
	0x57	1-bit mode	2-bit mode	-
	0xA7	1-bit mode	4-bit mode	-
CMD8	0x08	1-bit mode	1-bit mode	-
	0x18	1-bit mode	1-bit mode	-
	0x28	1-bit mode	1-bit mode	-
	0x58	1-bit mode	2-bit mode	-
	0xA8	1-bit mode	4-bit mode	-
CMD9	0x09	1-bit mode	1-bit mode	-
	0x19	1-bit mode	1-bit mode	-
	0x29	1-bit mode	1-bit mode	-
	0x59	1-bit mode	2-bit mode	-
	0xA9	1-bit mode	4-bit mode	-
CMDA	0x0A	1-bit mode	1-bit mode	-
	0x1A	1-bit mode	1-bit mode	-
	0x2A	1-bit mode	1-bit mode	-
	0x5A	1-bit mode	2-bit mode	-
	0xAA	1-bit mode	4-bit mode	-
End_SEG_TRANS	0x05	1-bit mode	-	-
En_QPI	0x06	1-bit mode	-	-

Table 46.5-17. CMD Values Supported in QPI Mode

Transfer Type	CMD[7:0]	CMD State	ADDR State	DATA State
Wr_BUF	0xA1	4-bit mode	4-bit mode	4-bit mode
Rd_BUF	0xA2	4-bit mode	4-bit mode	4-bit mode
Wr_DMA	0xA3	4-bit mode	4-bit mode	4-bit mode
Rd_DMA	0xA4	4-bit mode	4-bit mode	4-bit mode
CMD7	0xA7	4-bit mode	4-bit mode	-
CMD8	0xA8	4-bit mode	4-bit mode	-
CMD9	0xA9	4-bit mode	4-bit mode	-
CMDA	0xAA	4-bit mode	4-bit mode	-
End_SEG_TRANS	0xA5	4-bit mode	4-bit mode	-
Ex_QPI	0xDD	4-bit mode	4-bit mode	-

Master sends 0x06 CMD (En_QPI) to set GP-SPI slave to QPI mode and all the states of supported transfer will be in 4-bit mode afterwards. If 0xDD CMD (Ex_QPI) is received, GP-SPI slave will be back to SPI mode.

Other transfer types than these described in Table 46.5-16 and Table 46.5-17 are ignored. But if the CS low-level duration is longer than two APB_CLK cycles, SPI_TRANS_DONE_INT will be triggered. For more information on interrupts triggered at the end of transmissions, please refer to Section 46.11.

46.5.10.3 Slave Single Transfer and Slave Segmented Transfer

When GP-SPI works as a slave, it supports full-duplex and half-duplex communications controlled by DMA and by CPU. DMA-controlled transfer can be a single transfer, or a slave segmented transfer consisting of several transactions (segments). The CPU-controlled transfer can only be one single transfer, since each CPU-controlled transaction needs to be triggered by CPU.

In a slave segmented transfer, all transfer types listed in Table 46.5-16 and Table 46.5-17 are supported in a single transaction (segment). It means that CPU-controlled transaction and DMA-controlled transaction can be mixed in one slave segmented transfer.

It is recommended that in a slave segmented transfer:

- CPU-controlled transaction is used for handshake communication and short data transfers.
- DMA-controlled transaction is used for large data transfers.

46.5.10.4 Configuration of Slave Single Transfer

When operating as slave, GP-SPI supports CPU/DMA-controlled full-duplex/half-duplex single transfers.

The register configuration procedure is as follows (take GP-SPI2 as an example):

1. Configure the IO path via HP IO MUX or HP GPIO matrix between GP-SPI2 and an external SPI device.
2. Configure AHB clock (AHB_CLK).
3. Set SPI_SLAVE_MODE to enable slave mode.
4. Configure SPI_DOUTDIN:
    - 1: enable full-duplex communication.
    - 0: enable half-duplex communication.
5. Prepare data:
    - if CPU-controlled transfer is selected and GP-SPI2 is used to send data, then prepare data in registers SPI_W0_REG~SPI_W15_REG.
    - if DMA-controlled transfer is selected,
        - configure SPI_DMA_TX_ENA/SPI_DMA_RX_ENA and SPI_RX_EOF_EN.
        - configure DMA TX/RX link,
        - and start DMA TX/RX engine, as described in Section 46.5.7 and Section 46.5.8.
6. Set SPI_DMA_AFIFO_RST, SPI_BUF_AFIFO_RST, and SPI_RX_AFIFO_RST to reset these buffers.

7. Clear SPI_DMA_SLV_SEG_TRANS_EN in register SPI_DMA_CONF_REG to enable slave single transfer.
8. Set SPI_TRANS_DONE_INT_ENA in register SPI_DMA_INT_ENA_REG and wait for the interrupt SPI_TRANS_DONE_INT. In DMA-controlled mode, it is recommended to wait for the interrupt AXI_DMA_IN_SUC_EOF_CH*n*_INT when DMA RX buffer is used, which means that data has been stored in the related memory. Other interrupts described in Section 46.11 are optional.

46.5.10.5 Configuration of Slave Segmented Transfer in Half-Duplex

DMA must be used in this mode. The register configuration procedure is as follows (take GP-SPI2 as an example):

1. Configure the IO path via HP IO MUX or HP GPIO matrix between GP-SPI2 and an external SPI device.
2. Configure APB clock (APB_CLK).
3. Set SPI_SLAVE_MODE to enable slave mode.
4. Clear SPI_DOUTDIN to enable half-duplex communication.
5. Prepare data in registers SPI_W0_REG~SPI_W15_REG, if needed.
6. Set SPI_DMA_AFIFO_RST, SPI_BUF_AFIFO_RST, and SPI_RX_AFIFO_RST to reset these buffers.
7. Set bits SPI_DMA_RX_ENA and SPI_DMA_TX_ENA. Clear the bit SPI_RX_EOF_EN. Configure DMA TX/RX link and start DMA TX/RX engine, as shown in Section 46.5.7 and Section 46.5.8.
8. Set SPI_DMA_SLV_SEG_TRANS_EN in register SPI_DMA_CONF_REG to enable slave segmented transfer.
9. Set SPI_DMA_SEG_TRANS_DONE_INT_ENA in register SPI_DMA_INT_ENA_REG and wait for the interrupt SPI_DMA_SEG_TRANS_DONE_INT, which means that the segmented transfer has finished and data has been put into the related memory. Other interrupts described in Section 46.11 are optional.

When End_SEG_TRANS (0x05 in SPI mode, 0xA5 in QPI mode) is received by GP-SPI, this slave segmented transfer is ended and the interrupt SPI_DMA_SEG_TRANS_DONE_INT is triggered.

46.5.10.6 Configuration of Slave Segmented Transfer in Full-Duplex

DMA must be used in this mode. In such transfer, the data is transferred from and to the DMA buffer. The interrupt AXI_DMA_IN_SUC_EOF_CH*n*_INT is triggered when the transfer ends.

The register configuration procedure is as follows (take GP-SPI2 as an example):

1. Configure the IO path via HP IO MUX or HP GPIO matrix between GP-SPI2 and an external SPI device.
2. Configure APB clock (APB_CLK).
3. Set SPI_SLAVE_MODE and SPI_DOUTDIN, to enable slave full-duplex communication.
4. Set SPI_DMA_AFIFO_RST, SPI_BUF_AFIFO_RST, and SPI_RX_AFIFO_RST to reset these buffers.
5. Set SPI_DMA_TX_ENA/SPI_DMA_RX_ENA. Configure DMA TX/RX link and start DMA TX/RX engine, as shown in Section 46.5.7 and Section 46.5.8.
6. Set SPI_RX_EOF_EN in register SPI_DMA_CONF_REG. Configure SPI_MS_DATA_BITLEN[17:0] in register SPI_MS_DLEN_REG to the bit length of the received DMA data.
7. Set SPI_DMA_SLV_SEG_TRANS_EN in register SPI_DMA_CONF_REG to enable slave segmented transfer.

8. Set AXI_DMA_IN_SUC_EOF_CH*n*_INT_ENA and wait for the interrupt AXI_DMA_IN_SUC_EOF_CH*n*_INT.

46.6 CS Setup Time and Hold Time Control

SPI bus CS (SPI_CS) setup time and hold time are very important to meet the timing requirements of various SPI devices (e.g., flash or PSRAM).

CS setup time is the time between the CS falling edge and the first latch edge of SPI bus CLK (SPI_CLK). The first latch edge for mode 0 and mode 3 is rising edge, and falling edge for mode 1 and mode 2.

CS hold time is the time between the last latch edge of SPI_CLK and the CS rising edge.

When operating as slave, the CS setup time and hold time should be longer than 0.5 x T_SPI_CLK, otherwise the SPI transfer may be incorrect. T_SPI_CLK is one cycle of SPI_CLK.

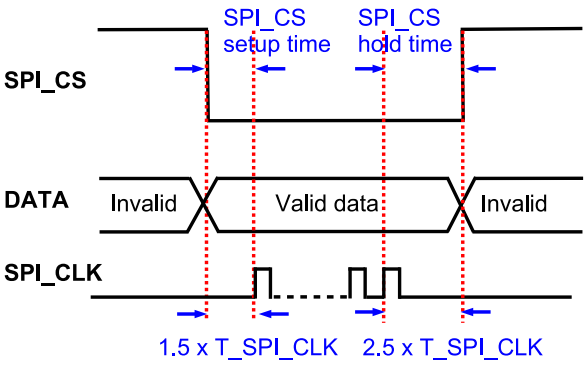
When operating as master, set the CS setup time by specifying SPI_CS_SETUP in register SPI_USER_REG and SPI_CS_SETUP_TIME in register SPI_USER1_REG.

- If SPI_CS_SETUP is cleared, the SPI CS setup time is 0.5 x T_SPI_CLK.
- If SPI_CS_SETUP is set, the SPI CS setup time is (SPI_CS_SETUP_TIME + 1.5) x T_SPI_CLK.

Set the CS hold time by specifying SPI_CS_HOLD in register SPI_USER_REG and SPI_CS_HOLD_TIME in register SPI_USER1_REG.

- If SPI_CS_HOLD is cleared, the SPI CS hold time is 0.5 x T_SPI_CLK.
- If SPI_CS_HOLD is set, the SPI CS hold time is (SPI_CS_HOLD_TIME + 1.5) x T_SPI_CLK.

Figure 46.6-1 and Figure 46.6-2 show the recommended CS timing and register configuration to access external RAM and flash.



Register Configurations:

SPI_CS_SETUP = 1; SPI_CS_SETUP_TIME = 0;
SPI_CS_HOLD = 1; SPI_CS_HOLD_TIME = 1.

Figure 46.6-1. Recommended CS Timing and Settings When Accessing External RAM

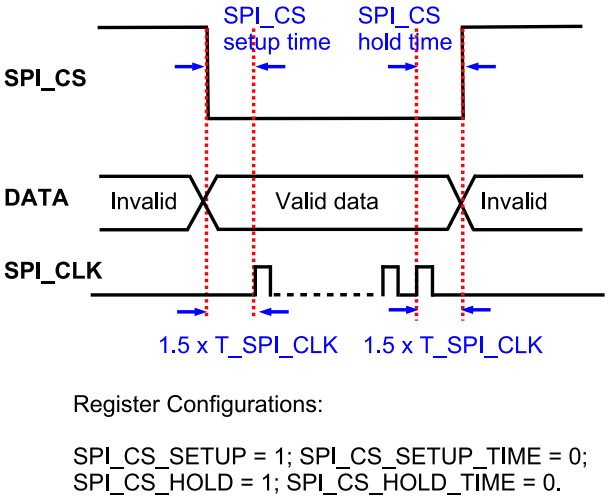


Figure 46.6-2. Recommended CS Timing and Settings When Accessing Flash

46.7 Clock Control

46.7.1 GP-SPI Clock Control

GP-SPI has the following clocks:

- clk_spi_mst: module clock of GP-SPI, derived from the following clock sources, depending on the configuration of HP_SYS_CLKRST_GPSPI2/3_CLK_SRC_SEL:
  - 0: XTAL_CLK
  - 1: RC_FAST_CLK
  - 2: SDIO_PLLO_CLK
  - 3: APLL_CLK
  - 4: SPLL_CLK (480 MHz)

  The divider value is equal to HP_SYS_CLKRST_GPSPI2/3_MST_CLK_DIV_NUM + 1. HP_SYS_CLKRST_GPSPI2/3_SYS_CLK_EN is used to enable the selected clock source. clk_spi_mst is used in GP-SPI as master to generate SPI_CLK signal for data transfer and for slaves.
- clk_hclk: module timing compensation clock of GP-SPI, which is a frequency-doubled clock derived from the same source as clk_spi_mst. Its clock source is also determined by HP_SYS_CLKRST_GPSPI2/3_CLK_SRC_SEL. To use this clock, HP_SYS_CLKRST_GPSPI2/3_HS_CLK_EN must be set, and the divider value is equal to HP_SYS_CLKRST_GPSPI2/3_HS_CLK_DIV_NUM + 1.
- SPI_CLK: this is the output clock when the GP-SPI works as master, or IO input clock when the GP-SPI works as slave.
- AHB_CLK: clock for register configuration.

When operating as master, the maximum output clock frequency of GP-SPI is $f_{clk_spi_mst}$. To have slower frequencies, the output clock frequency can be divided as follows:

$$f_{SPI_CLK} = \frac{f_{clk_spi_mst}}{(SPI_CLKCNT_N + 1)(SPI_CLKDIV_PRE + 1)}$$

The divider is configured by SPI_CLKCNT_N and SPI_CLKDIV_PRE in register SPI_CLOCK_REG. When the bit SPI_CLK_EQU_SYSCLK in register SPI_CLOCK_REG is set, the output clock frequency of GP-SPI will be $f_{clk_spi_mst}$. For other integral clock divisions, SPI_CLK_EQU_SYSCLK should be cleared.

When operating as slave, the supported input clock frequency (f_{SPI_CLK}) of GP-SPI is:

- If f_{AHB_CLK} >= 60 MHz, f_{SPI_CLK} <= 60 MHz.
- If f_{AHB_CLK} < 60 MHz, f_{SPI_CLK} <= f_{AHB_CLK}.

46.7.2 LP-SPI Clock Control

LP-SPI has the following clocks:

- lp_spi_pclk: LP-SPI module clock and clock for register configuration. lp_spi_pclk is used to generate LP_SPI_CLK for data transfer and for slaves when LP-SPI works as master. LPPERI_CK_EN_LP_SPI is used to enable this lp_spi_pclk.
- LP_SPI_CLK: output clock as master.

46.7.3 Clock Phase and Polarity

SPI protocol has four clock modes, i.e., modes 0~3. See Figure 46.7-1 and Figure 46.7-2 (excerpted from SPI protocol):

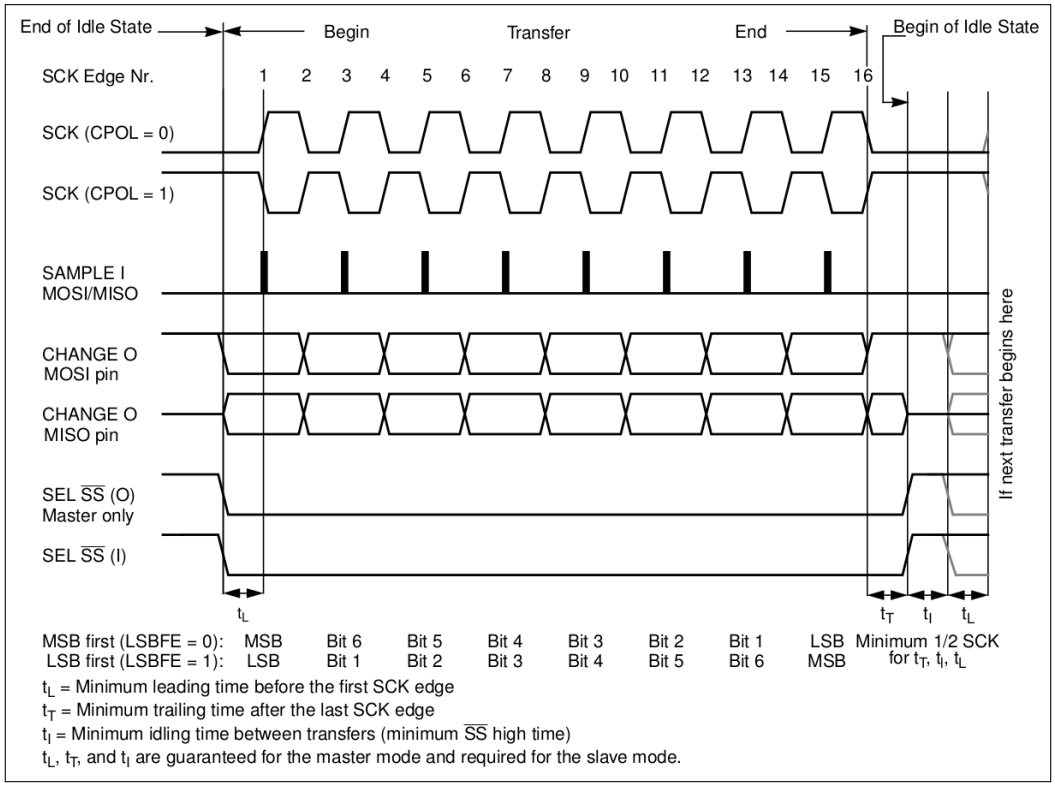


Figure 46.7-1. SPI Clock Mode 0 or 2

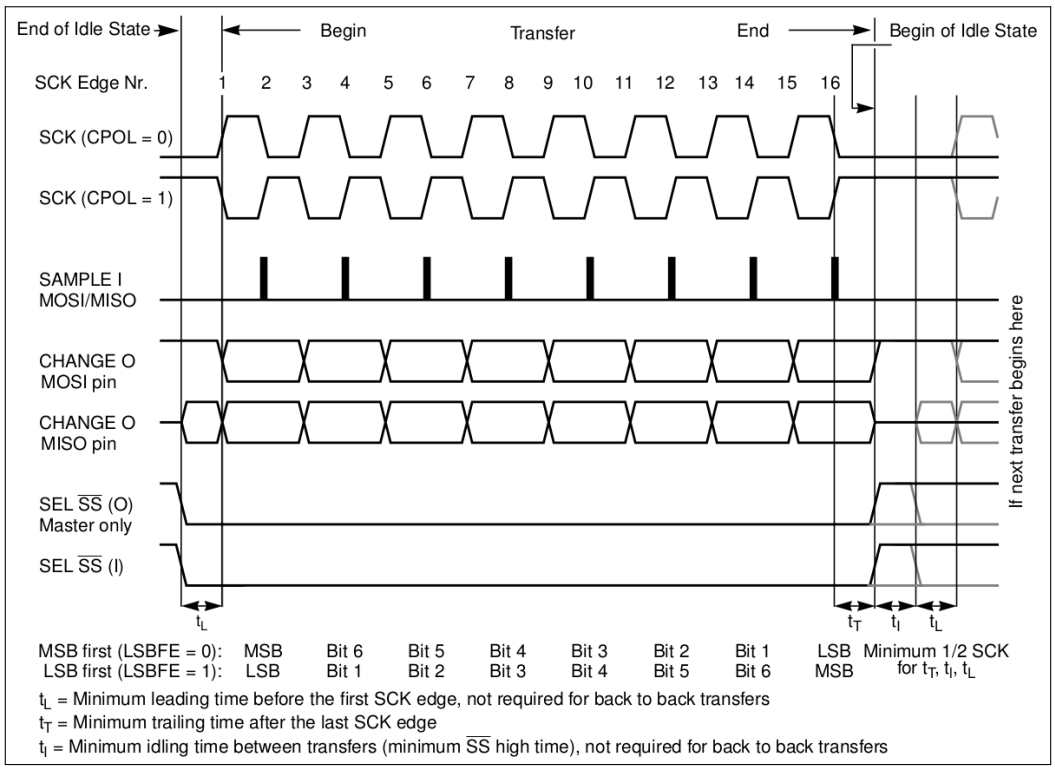


Figure 46.7-2. SPI Clock Mode 1 or 3

1. Mode 0: CPOL = 0, CPHA = 0; SCK is 0 when the SPI is in idle state; data is changed on the falling edge of SCK and sampled on the rising edge. The first data is shifted out before the first falling edge of SCK.
2. Mode 1: CPOL = 0, CPHA = 1; SCK is 0 when the SPI is in idle state; data is changed on the rising edge of SCK and sampled on the falling edge.
3. Mode 2: CPOL = 1, CPHA = 0; SCK is 1 when the SPI is in idle state; data is changed on the rising edge of SCK and sampled on the falling edge. The first data is shifted out before the first rising edge of SCK.
4. Mode 3: CPOL = 1, CPHA = 1; SCK is 1 when the SPI is in idle state; data is changed on the falling edge of SCK and sampled on the rising edge.

46.7.4 Clock Control as Master

The four clock modes 0~3 are supported in GP-SPI as master. The polarity and phase of GP-SPI clock are controlled by the bit SPI_CK_IDLE_EDGE in register SPI_MISC_REG and the bit SPI_CK_OUT_EDGE in register SPI_USER_REG. The register configuration for SPI clock modes 0~3 is provided in Table 46.7-1, and can be changed according to the path delay in the application.

Table 46.7-1. Clock Phase and Polarity Configuration as Master

Control Bit	Mode 0	Mode 1	Mode 2	Mode 3
SPI_CK_IDLE_EDGE	0	0	1	1
SPI_CK_OUT_EDGE	0	1	1	0

SPI_CLK_MODE is used to select the number of rising edges of SPI_CLK when SPI_CS raises high to be 0, 1, 2 or SPI_CLK always on.

Note:
When SPI_CLK_MODE is configured to 1 or 2, the bit SPI_CS_HOLD must be set and the value of SPI_CS_HOLD_TIME should be larger than 1.

46.7.5 Clock Control as Slave

GP-SPI as slave also supports clock modes 0~3. The polarity and phase are configured by the bits SPI_TSCK_I_EDGE and SPI_RSCK_I_EDGE in register SPI_USER_REG. The output edge of data is controlled by SPI_CLK_MODE_13 in register SPI_SLAVE_REG. The detailed register configuration is shown in Table 46.7-2:

Table 46.7-2. Clock Phase and Polarity Configuration as Slave

Control Bit	Mode 0	Mode 1	Mode 2	Mode 3
SPI_TSCK_I_EDGE	0	1	1	0
SPI_RSCK_I_EDGE	0	1	1	0
SPI_CLK_MODE_13	0	1	0	1

46.8 GP-SPI Timing Compensation

Introduction

The I/O lines are mapped via GPIO matrix or IO MUX. But there is no timing adjustment in IO MUX. The input data and output data can be delayed for 1 or 2 IO MUX operating clock cycles at the rising or falling edge in GPIO matrix. For detailed register configuration, see Chapter 10 *GPIO Matrix and IO MUX*. Timing compensation is supported only by GP-SPI.

Figure 46.8-1 shows the timing compensation control for GP-SPI as master, including the following paths:

- "CLK": the output path of GP-SPI bus clock. The clock is sent out by SPI_CLK out control module, passes through GPIO matrix or IO MUX and then goes to an external SPI device.
- "IN": data input path of GP-SPI (see line 3 path in color purple in Figure 46.8-1). The input data from an external SPI device passes through GPIO matrix or IO MUX, then is adjusted by the Timing Module (see Figure 46.5-2) and finally is stored into spi_rx_afifo.
- "OUT": data output path of GP-SPI (see line 2 path in color rose-red in Figure 46.8-1). The output data is sent out to the Timing Module, passes through GPIO matrix or IO MUX and is then captured by an external SPI device.

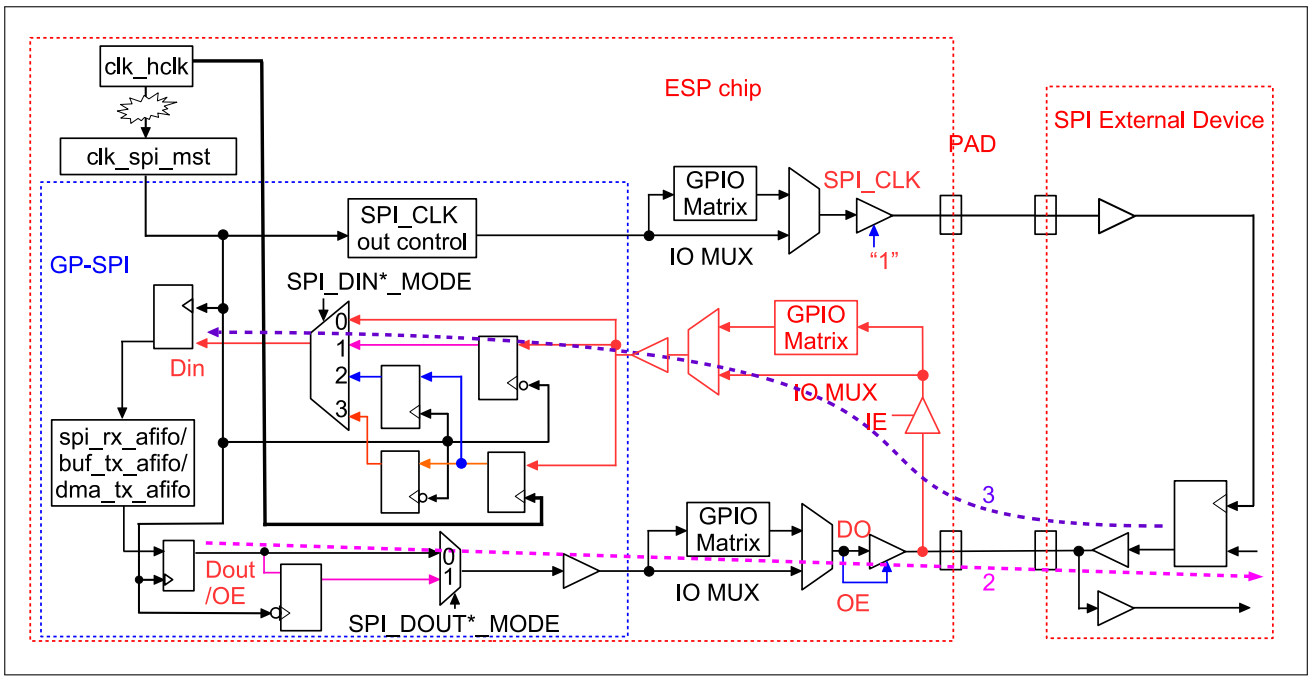


Figure 46.8-1. Timing Compensation Control Diagram in GP-SPI as Master

Every input and output data is passing through the Timing Module and the module can be used to apply delay in units of $T_{clk_spi_mst}$ (one cycle of clk_spi_mst) on rising or falling edge.

Key Registers

- SPI_DIN_MODE_REG: select the latch edge of input data
- SPI_DIN_NUM_REG: select the delay cycles of input data
- SPI_DOUT_MODE_REG: select the latch edge of output data

Timing Compensation Example

Figure 46.8-2 shows a timing compensation example in GP-SPI2 as master. Note that DUMMY cycle length is configurable to compensate the delay in I/O lines, so as to enhance the performance of GP-SPI2.

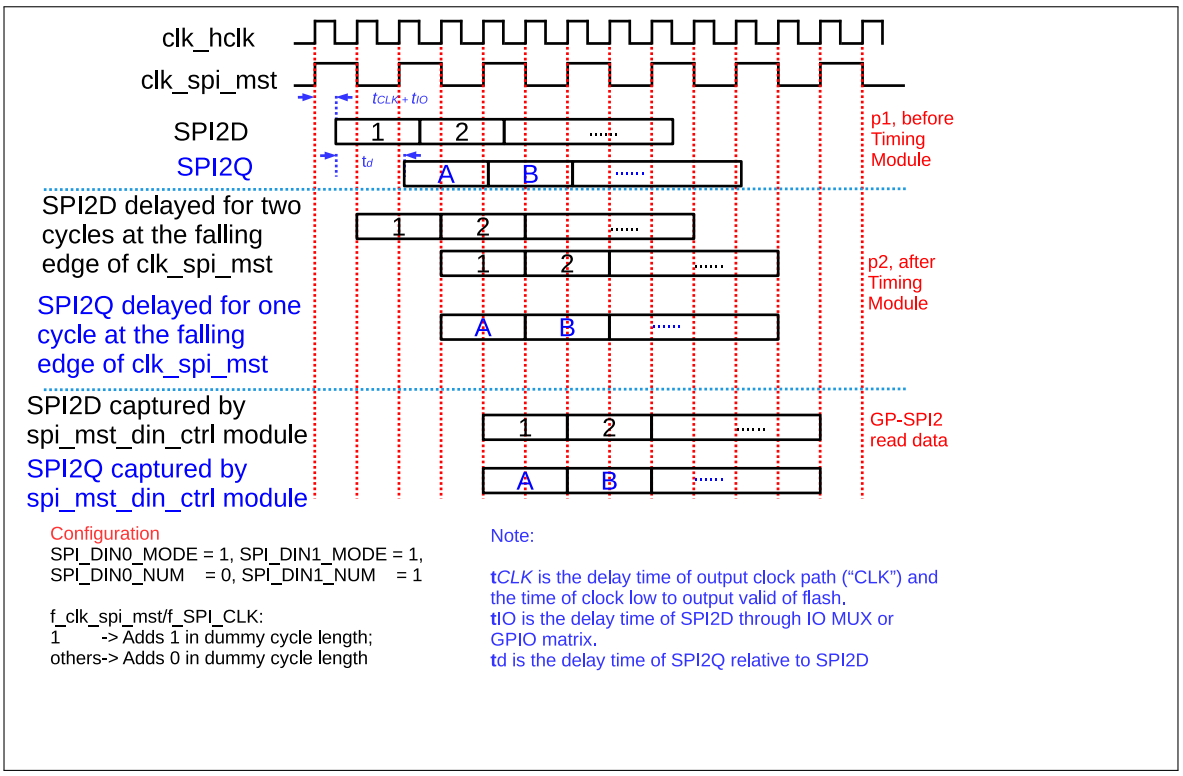


Figure 46.8-2. Timing Compensation Example in GP-SPI2 as Master

In Figure 46.8-2, "p1" is the point of input data of Timing Module, "p2" is the point of output data of Timing Module. Since the input data SPI2Q is unaligned to SPI2D, the read data of GP-SPI2 will be wrong without the timing compensation.

To get the correct read data, follow the settings below. Assuming $f_{clk_spi_mst}$ equals to f_{SPI_CLK}:

- Delay SPI2D for two cycles at the falling edge of clk_spi_mst.
- Delay SPI2Q for one cycle at the falling edge of clk_spi_mst.
- Add one extra dummy cycle.

When GP-SPI works as slave, if the bit SPI_RSCK_DATA_OUT in register SPI_SLAVE_REG is set to 1, the output data is sent at latch edge, which is half an SPI clock cycle earlier. This can be used for slave mode timing compensation.

46.9 LP-SPI Wake-Up

LP-SPI supports wake-up feature when working as a slave. When LP-SPI is in a sleep state (set LP_SPI_SLEEP_EN to 1), it can generate a wake_up signal and trigger PMU to wake up the chip by configuring LP_SPI_SLV_WK_MODE_SEL to select the wake up mode. For the detailed wake-up flow, see Chapter 15 *Low-Power Management*.

- When LP_SPI_SLV_WK_MODE_SEL = 0, LP-SPI wakes up the chip after detecting the start bit.
- When LP_SPI_SLV_WK_MODE_SEL = 1, LP-SPI wakes up the chip after receiving a specific character sequence.

- LP_SPI_SLEEP_WK_DATA_SEL selects to monitor the specific character starting from the CMD state or DIN state.
- LP_SPI_SLEEP_DIS_RXFIFO_WR_EN selects whether to save data before waking up.
- LP_SPI_SLV_WK_CHAR0~LP_SPI_SLV_WK_CHAR4 define wake up characters. LP-SPI wakes up the chip once it detects the character sequence consisting of CHAR0/CHAR1/CHAR2/CHAR3/CHAR4.
- LP_SPI_SLV_WK_CHAR_NUM and LP_SPI_SLV_WK_CHAR_MASK configure various character sequences, see Table 46.9-1. LP-SPI also supports fuzzy detection. For example, LP_SPI_WK_SLV_CHAR_NUM = 5 and LP_SPI_SLV_WK_MASK = 0x15 (binary 10101), then the character sequence will be */CHAR1/*/CHAR3/*.

Table 46.9-1. LP-SPI Wake-Up Configuration When Working as a Slave

LP_SPI_WK_SLV_CHAR_NUM	LP_SPI_SLV_WK_MASK	Character Sequence
1	0xF	CHAR4
2	0x7	CHAR3/CHAR4
3	0x3	CHAR2/CHAR3/CHAR4
4	0x1	CHAR1/CHAR2/CHAR3/CHAR4
5	0x0	CHAR0/CHAR1/CHAR2/CHAR3/CHAR4

46.10 Differences Among LP-SPI, GP-SPI2, and GP-SPI3

46.10.1 Feature Differences

Table 46.10-1 lists the differences among GP-SPI2, GP-SPI3, and LP-SPI on their supported features.

Table 46.10-1. Feature Differences Among GP-SPI2, GP-SPI3, and LP-SPI

Feature	GP-SPI2	GP-SPI3	LP-SPI
Data modes supported in CMD/ADDR/DOUT/DIN states when working as a master	1/2/4/8-bit	1/2/4-bit	1-bit
Data modes supported in CMD/ADDR/DOUT/DIN states when working as a slave	1/2/4-bit	1/2/4-bit	1-bit
DMA-controlled transfers	Support all DMA-controlled transfers	Not support DMA-controlled configurable segmented transfer	Not support any DMA related transfers
I/O mapping	Via HP GPIO matrix or HP IO MUX	Via HP GPIO matrix	Via LP GPIO matrix or LP IO MUX
CS line amount	6	3	1
Unaligned byte transfer	Support	Support	Not support
DTR feature	Support	Not support	Not support

cont'd on next page

Table 46.10-1 – Cont'd from previous page

Feature	GP-SPI2	GP-SPI3	LP-SPI
Wake-up feature when working as a slave	Not support	Not support	Support
Timing compensation	Support	Support	Not support

46.10.2 Register Differences

Table 46.10-2 lists the registers and fields that differs in GP-SPI2, GP-SPI3, and LP-SPI.

Table 46.10-2. Register Differences

Register	Field	GP-SPI2	GP-SPI3	LP-SPI
CMD_REG	CONF_BITLEN	Y	—	—
USER_REG	OPI_MODE	Y	Y	—
	FWRITE_DUAL	Y	Y	—
	FWRITE_DUAD	Y	Y	—
	FWRITE_OCT	Y	—	—
	USR_CONF_NEXT	Y	—	—
CTRL_REG	FADDR_DUAL	Y	Y	—
	FADDR_QUAD	Y	Y	—
	FADDR_OCT	Y	—	—
	FCMD_DUAL	Y	Y	—
	FCMD_QUAD	Y	Y	—
	FCMD_OCT	Y	—	—
	FREAD_DUAL	Y	Y	—
	FREAD_QUAD	Y	Y	—
	FREAD_OCT	Y	—	—
	HOLD_POL	Y	Y	—
	WP_POL	Y	Y	—
MISC_REG	CS1_DIS	Y	Y	—
	CS2_DIS	Y	Y	—
	CS3_DIS	Y	—	—
	CS4_DIS	Y	—	—
	CS5_DIS	Y	—	—
	CLK_DATA_DTR_EN	Y	—	—
	DATA_DTR_EN	Y	—	—
	ADDR_DTR_EN	Y	—	—
	DQS_IDLE_EDGE	Y	—	—
	QUAD_DIN_PIN_SWAP	Y	—	—

Cont'd on next page

Table 46.10-2 – cont'd from previous page

Register	Field	GP-SPI2	GP-SPI3	LP-SPI
DMA_CONF_REG	DMA_OUTFIFO_EMPTY	Y	Y	—
	DMA_INFIFO_FULL	Y	Y	—
	DMA_SLV_SEG_TRANS_EN	Y	Y	—
	SLV_RX_SEG_TRANS_CLR_EN	Y	Y	—
	SLV_TX_SEG_TRANS_CLR_EN	Y	Y	—
	RX_EOF_EN	Y	Y	—
	DMA_RX_ENA	Y	Y	—
	DMA_TX_ENA	Y	Y	—
	DMA_AFIFO_RST	Y	Y	—
SLEEP_CONF0_REG	SLV_WK_CHAR0	—	—	Y
	SLV_WK_CHAR_NUM	—	—	Y
	SLV_WK_CHAR_MASK	—	—	Y
	SLV_WK_MODE_SEL	—	—	Y
	SLEEP_EN	—	—	Y
	SLEEP_DIS_RXFIFO_WR_EN	—	—	Y
	SLEEP_WK_DATA_SEL	—	—	Y
SLEEP_CONF1_REG	SLV_WK_CHAR1	—	—	Y
	SLV_WK_CHAR2	—	—	Y
	SLV_WK_CHAR3	—	—	Y
	SLV_WK_CHAR4	—	—	Y
SLAVE_REG	SLV_RDDMA_BITLEN_EN	Y	Y	—
	SLV_WRDMA_BITLEN_EN	Y	Y	—
	SLV_LAST_BYTE_STRB	Y	Y	—
	DMA_SEG_MAGIC_VALUE	Y	—	—
	USR_CONF	Y	—	—
	MST_FD_WAIT_DMA_TX_DATA	Y	Y	—
DIN_MODE_REG	DIN4_MODE	Y	—	—
	DIN5_MODE	Y	—	—
	DIN6_MODE	Y	—	—
	DIN7_MODE	Y	—	—
DIN_NUM_REG	DIN4_NUM	Y	—	—
	DIN5_NUM	Y	—	—
	DIN6_NUM	Y	—	—
	DIN7_NUM	Y	—	—
DOUT_MODE_REG	DOUT4_MODE	Y	—	—
	DOUT5_MODE	Y	—	—
	DOUT6_MODE	Y	—	—
	DOUT7_MODE	Y	—	—
	D_DQS_MODE	Y	—	—

Cont'd on next page

Table 46.10-2 – cont'd from previous page

Register	Field	GP-SPI2	GP-SPI3	LP-SPI
DMA_INT_ENA_REG	DMA_INFIFO_FULL_ERR_INT_ENA	Y	Y	—
	DMA_OUTFIFO_EMPTY_ERR_INT_ENA	Y	Y	—
	SLV_EX_QPI_INT_ENA	Y	Y	—
	SLV_EN_QPI_INT_ENA	Y	Y	—
	SLV_CMD7_INT_ENA	Y	Y	—
	SLV_CMD8_INT_ENA	Y	Y	—
	SLV_CMD9_INT_ENA	Y	Y	—
	SLV_CMDA_INT_ENA	Y	Y	—
	SLV_RD_DMA_DONE_INT_ENA	Y	Y	—
	SLV_WR_DMA_DONE_INT_ENA	Y	Y	—
	LP_SPI_WAKEUP_INT_ENA	—	—	Y
	DMA_SEG_TRANS_DONE_INT_ENA	Y	Y	—
	SPI_SEG_MAGIC_ERR_INT_ENA	Y	—	—
DMA_INT_CLR_REG	DMA_INFIFO_FULL_ERR_INT_CLR	Y	Y	—
	DMA_OUTFIFO_EMPTY_ERR_INT_CLR	Y	Y	—
	SLV_EX_QPI_INT_CLR	Y	Y	—
	SLV_EN_QPI_INT_CLR	Y	Y	—
	SLV_CMD7_INT_CLR	Y	Y	—
	SLV_CMD8_INT_CLR	Y	Y	—
	SLV_CMD9_INT_CLR	Y	Y	—
	SLV_CMDA_INT_CLR	Y	Y	—
	SLV_RD_DMA_DONE_INT_CLR	Y	Y	—
	SLV_WR_DMA_DONE_INT_CLR	Y	Y	—
	LP_SPI_WAKEUP_INT_CLR	—	—	Y
	DMA_SEG_TRANS_DONE_INT_CLR	Y	Y	—
	SPI_SEG_MAGIC_ERR_INT_CLR	Y	—	—
DMA_INT_RAW_REG	DMA_INFIFO_FULL_ERR_INT_RAW	Y	Y	—
	DMA_OUTFIFO_EMPTY_ERR_INT_RAW	Y	Y	—
	SLV_EX_QPI_INT_RAW	Y	Y	—
	SLV_EN_QPI_INT_RAW	Y	Y	—
	SLV_CMD7_INT_RAW	Y	Y	—
	SLV_CMD8_INT_RAW	Y	Y	—
	SLV_CMD9_INT_RAW	Y	Y	—
	SLV_CMDA_INT_RAW	Y	Y	—
	SLV_RD_DMA_DONE_INT_RAW	Y	Y	—
	SLV_WR_DMA_DONE_INT_RAW	Y	Y	—
	LP_SPI_WAKEUP_INT_RAW	—	—	Y
	DMA_SEG_TRANS_DONE_INT_RAW	Y	Y	—
	SPI_SEG_MAGIC_ERR_INT_RAW	Y	—	—

Cont'd on next page

Table 46.10-2 – cont'd from previous page

Register	Field	GP-SPI2	GP-SPI3	LP-SPI
DMA_INT_ST_REG	DMA_INFIFO_FULL_ERR_INT_ST	Y	Y	—
	DMA_OUTFIFO_EMPTY_ERR_INT_ST	Y	Y	—
	SLV_EX_QPI_INT_ST	Y	Y	—
	SLV_EN_QPI_INT_ST	Y	Y	—
	SLV_CMD7_INT_ST	Y	Y	—
	SLV_CMD8_INT_ST	Y	Y	—
	SLV_CMD9_INT_ST	Y	Y	—
	SLV_CMDA_INT_ST	Y	Y	—
	SLV_RD_DMA_DONE_INT_ST	Y	Y	—
	SLV_WR_DMA_DONE_INT_ST	Y	Y	—
	LP_SPI_WAKEUP_INT_ST	—	—	Y
	DMA_SEG_TRANS_DONE_INT_ST	Y	Y	—
	SPI_SEG_MAGIC_ERR_INT_ST	Y	—	—
DMA_INT_SET_REG	DMA_INFIFO_FULL_ERR_INT_SET	Y	Y	—
	DMA_OUTFIFO_EMPTY_ERR_INT_SET	Y	Y	—
	SLV_EX_QPI_INT_SET	Y	Y	—
	SLV_EN_QPI_INT_SET	Y	Y	—
	SLV_CMD7_INT_SET	Y	Y	—
	SLV_CMD8_INT_SET	Y	Y	—
	SLV_CMD9_INT_SET	Y	Y	—
	SLV_CMDA_INT_SET	Y	Y	—
	SLV_RD_DMA_DONE_INT_SET	Y	Y	—
	SLV_WR_DMA_DONE_INT_SET	Y	Y	—
	LP_SPI_WAKEUP_INT_SET	—	—	Y
	DMA_SEG_TRANS_DONE_INT_SET	Y	Y	—
	SPI_SEG_MAGIC_ERR_INT_SET	Y	—	—

46.10.3 Interrupt Differences

Table 46.10-3 lists the interrupts supported in GP-SPI2, GP-SPI3, and LP-SPI.

Table 46.10-3. Interrupt Differences

Interrupt	GP-SPI2	GP-SPI3	LP-SPI
SPI_DMA_INFIFO_FULL_ERR_INT	Y	Y	—
SPI_DMA_OUTFIFO_EMPTY_ERR_INT	Y	Y	—
SPI_SLV_EX_QPI_INT	Y	Y	—
SPI_SLV_EN_QPI_INT	Y	Y	—
SPI_SLV_CMD7_INT	Y	Y	—
SPI_SLV_CMD8_INT	Y	Y	—
SPI_SLV_CMD9_INT	Y	Y	—
SPI_SLV_CMDA_INT	Y	Y	—

Table 46.10-3. Interrupt Differences

Interrupt	GP-SPI2	GP-SPI3	LP-SPI
SPI_SLV_RD_DMA_DONE_INT	Y	Y	—
SPI_SLV_WR_DMA_DONE_INT	Y	Y	—
SPI_SLV_RD_BUF_DONE_INT	Y	Y	Y
SPI_SLV_WR_BUF_DONE_INT	Y	Y	Y
SPI_TRANS_DONE_INT	Y	Y	Y
SPI_DMA_SEG_TRANS_DONE_INT	Y	Y	—
SPI_SEG_MAGIC_ERR_INT	Y	—	—
SPI_MST_RX_AFIFO_WFULL_ERR_INT	Y	Y	Y
SPI_MST_TX_AFIFO_REMPTY_ERR_INT	Y	Y	Y
SPI_SLV_CMD_ERR_INT	Y	Y	Y
SPI_APP2_INT	Y	Y	Y
SPI_APP1_INT	Y	Y	Y
LP_SPI_WAKEUP_INT	—	—	Y

46.11 Interrupt

46.11.1 Interrupt Description

ESP32-P4 LP-SPI and GP-SPI can generate the following external interrupt signals:

- LP-SPI: LP_SPI_INTR
- GP-SPI2: SPI_INTR_2
- GP-SPI3: SPI_INTR_3

LP_SPI_INTR is directly mapped to LP CPU (see *Low-Power CPU*), and SPI_INTR_2/SPI_INTR_3 can be mapped to HP CPU0/1 through the Interrupt Matrix. These interrupt signals are generated by internal interrupt sources of LP-SPI and GP-SPI. Table 46.11-1 lists the internal interrupt sources, interrupt trigger conditions, and generated interrupt signals of LP-SPI and GP-SPI.

Table 46.11-1. Internal Interrupt Sources of LP-SPI and GP-SPI

Internal Interrupt Source	Trigger Condition	Interrupt Signal
SPI_DMA_INFIFO_FULL_ERR_INT	The length of DMA RX FIFO is shorter than that of actual data transferred	SPI_INTR_2/SPI_INTR_3
SPI_DMA_OUTFIFO_EMPTY_ERR_INT	The length of DMA TX FIFO is shorter than that of actual data transferred	SPI_INTR_2/SPI_INTR_3
SPI_SLV_EX_QPI_INT	Ex_QPI is received correctly in GP-SPI2 as slave and the SPI transfer ends	SPI_INTR_2/SPI_INTR_3
SPI_SLV_EN_QPI_INT	En_QPI is received correctly in GP-SPI2 as slave and the SPI transfer ends	SPI_INTR_2/SPI_INTR_3

Cont'd on next page

Table 46.11-1 – cont'd from previous page

Internal Interrupt Source	Trigger Condition	Interrupt Signal
SPI_SLV_CMD7_INT	CMD7 is received correctly in GP-SPI2 as slave and the SPI transfer ends	SPI_INTR_2/SPI_INTR_3
SPI_SLV_CMD8_INT	CMD8 is received correctly in GP-SPI2 as slave and the SPI transfer ends	SPI_INTR_2/SPI_INTR_3
SPI_SLV_CMD9_INT	CMD9 is received correctly in GP-SPI2 as slave and the SPI transfer ends	SPI_INTR_2/SPI_INTR_3
SPI_SLV_CMDA_INT	CMDA is received correctly in GP-SPI2 as slave and the SPI transfer ends	SPI_INTR_2/SPI_INTR_3
SPI_SLV_RD_DMA_DONE_INT	At the end of Rd_DMA transfer as slave	SPI_INTR_2/SPI_INTR_3
SPI_SLV_WR_DMA_DONE_INT	At the end of Wr_DMA transfer as slave	SPI_INTR_2/SPI_INTR_3
SPI_SLV_RD_BUF_DONE_INT	At the end of Rd_BUF transfer as slave	LP_SPI_INTR/SPI_INTR_2/ SPI_INTR_3
SPI_SLV_WR_BUF_DONE_INT	At the end of Wr_BUF transfer as slave	LP_SPI_INTR/SPI_INTR_2/ SPI_INTR_3
SPI_TRANS_DONE_INT	At the end of SPI bus transfer in both as master and as slave.	LP_SPI_INTR/SPI_INTR_2/ SPI_INTR_3
SPI_DMA_SEG_TRANS_DONE _INT	At the end of End_SEG_TRANS transfer in GP-SPI slave segmented transfer mode or at the end of configurable segmented transfer as master	SPI_INTR_2/SPI_INTR_3
SPI_SEG_MAGIC_ERR_INT	A Magic error occurs in CONF buffer during configurable segmented transfer as master	SPI_INTR_2
SPI_MST_RX_AFIFO_WFULL _ERR_INT	RX AFIFO write-full error in GP-SPI as master	LP_SPI_INTR/SPI_INTR_2/ SPI_INTR_3
SPI_MST_TX_AFIFO_REMPTY _ERR_INT	TX AFIFO read-empty error in GP-SPI as master	LP_SPI_INTR/SPI_INTR_2/ SPI_INTR_3
SPI_SLV_CMD_ERR_INT	A received command value is not supported in GP-SPI as slave	LP_SPI_INTR/SPI_INTR_2/ SPI_INTR_3
SPI_APP2_INT	Set SPI_APP2_INT_SET (Only used for user defined function)	LP_SPI_INTR/SPI_INTR_2/ SPI_INTR_3
SPI_APP1_INT	Set SPI_APP1_INT_SET (Only used for user defined function)	LP_SPI_INTR/SPI_INTR_2/ SPI_INTR_3

Cont'd on next page

Table 46.11-1 – cont'd from previous page

Internal Interrupt Source	Trigger Condition	Interrupt Signal
LP_SPI_WAKEUP_INT	Pre-defined chars are received and wake_up signal is generated	LP_SPI_INTR

LP-SPI is a simplified version of GP-SPI, which contains only some GP-SPI interrupt sources. Each interrupt source has a common set of configuration registers, see the *Interrupt Registers* in Section *Register Summary*.

46.11.2 Interrupts Used in Master and in Slave (Take GP-SPI as an Example)

Table 46.11-2 and Table 46.11-3 show the interrupts used in GP-SPI master and slave modes. Set the interrupt enable bit SPI_*_INT_ENA in SPI_DMA_INT_ENA_REG and wait for the interrupt. When the transfer ends, the related interrupt is triggered and should be cleared by software before the next transfer.

Table 46.11-2. GP-SPI Master Mode Interrupts

Transfer Type	Communication Mode	Controlled by	Interrupt
Single Transfer	Full-duplex	DMA	AXI_DMA_IN_SUC_EOF_CH*n*_INT [1]
		CPU	SPI_TRANS_DONE_INT [2]
	Half-duplex MOSI Mode	DMA	SPI_TRANS_DONE_INT
		CPU	SPI_TRANS_DONE_INT
	Half-duplex MISO Mode	DMA	AXI_DMA_IN_SUC_EOF_CH*n*_INT
		CPU	SPI_TRANS_DONE_INT
Configurable Segmented Transfer	Full-duplex	DMA	SPI_DMA_SEG_TRANS_DONE_INT [3]
		CPU	Not supported
	Half-duplex MOSI Mode	DMA	SPI_DMA_SEG_TRANS_DONE_INT
		CPU	Not supported
	Half-duplex MISO	DMA	SPI_DMA_SEG_TRANS_DONE_INT
		CPU	Not supported

[1] If AXI_DMA_IN_SUC_EOF_CH*n*_INT is triggered, it means all the RX data of GP-SPI has been stored in the RX buffer, and the TX data has been transferred to the slave.

[2] SPI_TRANS_DONE_INT is triggered when CS is high, which indicates that master has completed the data exchange in SPI_W0_REG~SPI_W15_REG with slave in this mode.

[3] If SPI_DMA_SEG_TRANS_DONE_INT is triggered, it means that the whole configurable segmented transfer (consisting of several segments) has finished, i.e., the RX data has been stored in the RX buffer completely and all the TX data has been sent out.

Table 46.11-3. GP-SPI Slave Mode Interrupts

Transfer Type	Communication Mode	Controlled by	Interrupt
Single Transfer	Full-duplex	DMA	AXI_DMA_IN_SUC_EOF_CH*n*_INT [1]
		CPU	SPI_TRANS_DONE_INT [2]
	Half-duplex MOSI Mode	DMA (Wr_DMA)	AXI_DMA_IN_SUC_EOF_CH*n*_INT [3]
		CPU (Wr_BUF)	SPI_TRANS_DONE_INT [4]
	Half-duplex MISO Mode	DMA (Rd_DMA)	SPI_TRANS_DONE_INT [5]
		CPU (Rd_BUF)	SPI_TRANS_DONE_INT [6]
Slave Segmented Transfer	Full-duplex	DMA	AXI_DMA_IN_SUC_EOF_CH*n*_INT [7]
		CPU	Not supported [8]
	Half-duplex MOSI Mode	DMA (Wr_DMA)	SPI_DMA_SEG_TRANS_DONE_INT [9]
		CPU (Wr_BUF)	Not supported [10]
	Half-duplex MISO Mode	DMA (Rd_DMA)	SPI_DMA_SEG_TRANS_DONE_INT [11]
		CPU (Rd_BUF)	Not supported [12]

[1] If AXI_DMA_IN_SUC_EOF_CH*n*_INT is triggered, it means all the RX data has been stored in the RX buffer, and the TX data has been sent to the slave.

[2] SPI_TRANS_DONE_INT is triggered when CS is high, which indicates that master has completed the data exchange in SPI_W0_REG~SPI_W15_REG with slave in this mode.

[3] SPI_SLV_WR_DMA_DONE_INT just means that the transmission on the SPI bus is done, but can not ensure that all the push data has been stored in the RX buffer. For this reason, AXI_DMA_IN_SUC_EOF_CH*n*_INT is recommended.

[4] Or wait for SPI_SLV_WR_BUF_DONE_INT.

[5] Or wait for SPI_SLV_RD_DMA_DONE_INT.

[6] Or wait for SPI_SLV_RD_BUF_DONE_INT.

[7] Slave should set the total read data byte length in SPI_MS_DATA_BITLEN before the transfer begins. Set SPI_RX_EOF_EN to 1 before the end of the interrupt program.

[8] Master and slave should define a method to end the segmented transfer, such as via GPIO interrupt.

[9] Master sends End_SEG_TRAN to end the segmented transfer or slave sets the total read data byte length in SPI_MS_DATA_BITLEN and waits for AXI_DMA_IN_SUC_EOF_CH*n*_INT.

[10] Half-duplex Wr_BUF single transfer can be used in a slave segmented transfer.

[11] Master sends End_SEG_TRAN to end the segmented transfer.

[12] Half-duplex Rd_BUF single transfer can be used in a slave segmented transfer.

46.12 Register Summary

46.12.1 GP-SPI2 Register Summary

The addresses in this section are relative to GP-SPI2 base address provided in Table 9.3-2 in Chapter 9 *System and Memory*.

The abbreviations given in Column Access are explained in Section *Access Types for Registers*.

Name	Description	Address	Access
User-defined control registers			

Name	Description	Address	Access
SPI_CMD_REG	Command control register	0x0000	varies
SPI_ADDR_REG	Address value register	0x0004	R/W
SPI_USER_REG	SPI USER control register	0x0010	varies
SPI_USER1_REG	SPI USER control register 1	0x0014	R/W
SPI_USER2_REG	SPI USER control register 2	0x0018	R/W
Control and configuration registers			
SPI_CTRL_REG	SPI control register	0x0008	R/W
SPI_MS_DLEN_REG	SPI data bit length control register	0x001C	R/W
SPI_MISC_REG	SPI misc register	0x0020	R/W
SPI_DMA_CONF_REG	SPI DMA control register	0x0030	varies
SPI_SLAVE_REG	SPI slave control register	0x00E0	varies
SPI_SLAVE1_REG	SPI slave control register 1	0x00E4	R/W/SS
Clock control registers			
SPI_CLOCK_REG	SPI clock control register	0x000C	R/W
SPI_CLK_GATE_REG	SPI module clock and register clock control	0x00E8	R/W
Timing registers			
SPI_DIN_MODE_REG	SPI input delay mode configuration	0x0024	R/W
SPI_DIN_NUM_REG	SPI input delay number configuration	0x0028	R/W
SPI_DOUT_MODE_REG	SPI output delay mode configuration	0x002C	R/W
Interrupt registers			
SPI_DMA_INT_ENA_REG	SPI interrupt enable register	0x0034	R/W
SPI_DMA_INT_CLR_REG	SPI interrupt clear register	0x0038	WT
SPI_DMA_INT_RAW_REG	SPI interrupt raw register	0x003C	R/WTC/SS
SPI_DMA_INT_ST_REG	SPI interrupt status register	0x0040	RO
SPI_DMA_INT_SET_REG	SPI interrupt software set register	0x0044	WT
CPU-controlled data buffer			
SPI_W0_REG	SPI CPU-controlled buffer 0	0x0098	R/W/SS
SPI_W1_REG	SPI CPU-controlled buffer 1	0x009C	R/W/SS
SPI_W2_REG	SPI CPU-controlled buffer 2	0x00A0	R/W/SS
SPI_W3_REG	SPI CPU-controlled buffer 3	0x00A4	R/W/SS
SPI_W4_REG	SPI CPU-controlled buffer 4	0x00A8	R/W/SS
SPI_W5_REG	SPI CPU-controlled buffer 5	0x00AC	R/W/SS
SPI_W6_REG	SPI CPU-controlled buffer 6	0x00B0	R/W/SS
SPI_W7_REG	SPI CPU-controlled buffer 7	0x00B4	R/W/SS
SPI_W8_REG	SPI CPU-controlled buffer 8	0x00B8	R/W/SS
SPI_W9_REG	SPI CPU-controlled buffer 9	0x00BC	R/W/SS
SPI_W10_REG	SPI CPU-controlled buffer 10	0x00C0	R/W/SS
SPI_W11_REG	SPI CPU-controlled buffer 11	0x00C4	R/W/SS
SPI_W12_REG	SPI CPU-controlled buffer 12	0x00C8	R/W/SS
SPI_W13_REG	SPI CPU-controlled buffer 13	0x00CC	R/W/SS
SPI_W14_REG	SPI CPU-controlled buffer 14	0x00D0	R/W/SS
SPI_W15_REG	SPI CPU-controlled buffer 15	0x00D4	R/W/SS
Version register			

Name	Description	Address	Access
SPI_DATE_REG	Version control	0x00F0	R/W

46.12.2 GP-SPI3 Register Summary

The addresses in this section are relative to GP-SPI3 base address provided in Table 9.3-2 in Chapter 9 *System and Memory*.

The abbreviations given in Column Access are explained in Section *Access Types for Registers*.

Name	Description	Address	Access
User-defined control registers			
SPI_CMD_REG	Command control register	0x0000	varies
SPI_ADDR_REG	Address value register	0x0004	R/W
SPI_USER_REG	SPI USER control register	0x0010	varies
SPI_USER1_REG	SPI USER control register 1	0x0014	R/W
SPI_USER2_REG	SPI USER control register 2	0x0018	R/W
Control and configuration registers			
SPI_CTRL_REG	SPI control register	0x0008	R/W
SPI_MS_DLEN_REG	SPI data bit length control register	0x001C	R/W
SPI_MISC_REG	SPI misc register	0x0020	R/W
SPI_DMA_CONF_REG	SPI DMA control register	0x0030	varies
SPI_SLAVE_REG	SPI slave control register	0x00E0	varies
SPI_SLAVE1_REG	SPI slave control register 1	0x00E4	R/W/SS
Clock control registers			
SPI_CLOCK_REG	SPI clock control register	0x000C	R/W
SPI_CLK_GATE_REG	SPI module clock and register clock control	0x00E8	R/W
Timing registers			
SPI_DIN_MODE_REG	SPI input delay mode configuration	0x0024	R/W
SPI_DIN_NUM_REG	SPI input delay number configuration	0x0028	R/W
SPI_DOUT_MODE_REG	SPI output delay mode configuration	0x002C	R/W
Interrupt registers			
SPI_DMA_INT_ENA_REG	SPI interrupt enable register	0x0034	R/W
SPI_DMA_INT_CLR_REG	SPI interrupt clear register	0x0038	WT
SPI_DMA_INT_RAW_REG	SPI interrupt raw register	0x003C	R/WTC/SS
SPI_DMA_INT_ST_REG	SPI interrupt status register	0x0040	RO
SPI_DMA_INT_SET_REG	SPI interrupt software set register	0x0044	WT
CPU-controlled data buffer			
SPI_W0_REG	SPI CPU-controlled buffer 0	0x0098	R/W/SS
SPI_W1_REG	SPI CPU-controlled buffer 1	0x009C	R/W/SS
SPI_W2_REG	SPI CPU-controlled buffer 2	0x00A0	R/W/SS
SPI_W3_REG	SPI CPU-controlled buffer 3	0x00A4	R/W/SS
SPI_W4_REG	SPI CPU-controlled buffer 4	0x00A8	R/W/SS
SPI_W5_REG	SPI CPU-controlled buffer 5	0x00AC	R/W/SS
SPI_W6_REG	SPI CPU-controlled buffer 6	0x00B0	R/W/SS

Name	Description	Address	Access
SPI_W7_REG	SPI CPU-controlled buffer 7	0x00B4	R/W/SS
SPI_W8_REG	SPI CPU-controlled buffer 8	0x00B8	R/W/SS
SPI_W9_REG	SPI CPU-controlled buffer 9	0x00BC	R/W/SS
SPI_W10_REG	SPI CPU-controlled buffer 10	0x00C0	R/W/SS
SPI_W11_REG	SPI CPU-controlled buffer 11	0x00C4	R/W/SS
SPI_W12_REG	SPI CPU-controlled buffer 12	0x00C8	R/W/SS
SPI_W13_REG	SPI CPU-controlled buffer 13	0x00CC	R/W/SS
SPI_W14_REG	SPI CPU-controlled buffer 14	0x00D0	R/W/SS
SPI_W15_REG	SPI CPU-controlled buffer 15	0x00D4	R/W/SS
Version register			
SPI_DATE_REG	Version control	0x00F0	R/W

46.12.3 LP-SPI Register Summary

The addresses in this section are relative to LP-SPI base address provided in Table 9.3-2 in Chapter 9 *System and Memory*.

The abbreviations given in Column Access are explained in Section *Access Types for Registers*.

Name	Description	Address	Access
User-defined control registers			
LP_SPI_CMD_REG	Command control register	0x0000	varies
LP_SPI_ADDR_REG	Address value register	0x0004	R/W
LP_SPI_USER_REG	LP-SPI USER control register	0x0010	R/W
LP_SPI_USER1_REG	LP-SPI USER control register 1	0x0014	R/W
LP_SPI_USER2_REG	LP-SPI USER control register 2	0x0018	R/W
Control and configuration registers			
LP_SPI_CTRL_REG	LP-SPI control register	0x0008	R/W
LP_SPI_MS_DLEN_REG	LP-SPI data bit length control register	0x001C	R/W
LP_SPI_MISC_REG	LP-SPI misc register	0x0020	R/W
LP_SPI_DMA_CONF_REG	LP-SPI DMA control register	0x0030	WT
LP_SPI_SLEEP_CONF0_REG	LP-SPI sleep configure register 0	0x0044	R/W
LP_SPI_SLEEP_CONF1_REG	LP-SPI sleep configure register 1	0x0048	R/W
LP_SPI_SLAVE_REG	LP-SPI slave control register	0x00E0	varies
LP_SPI_SLAVE1_REG	LP-SPI slave control register 1	0x00E4	R/W/SS
Clock control registers			
LP_SPI_CLOCK_REG	LP-SPI clock control register	0x000C	R/W
LP_SPI_CLK_GATE_REG	LP-SPI module clock and register clock control	0x00E8	R/W
Timing registers			
LP_SPI_DIN_MODE_REG	LP-SPI input delay mode configuration	0x0024	R/W
LP_SPI_DIN_NUM_REG	LP-SPI input delay number configuration	0x0028	R/W
LP_SPI_DOUT_MODE_REG	LP-SPI output delay mode configuration	0x002C	R/W
Interrupt registers			
LP_SPI_DMA_INT_ENA_REG	LP-SPI interrupt enable register	0x0034	R/W

Name	Description	Address	Access
LP_SPI_DMA_INT_CLR_REG	LP-SPI interrupt clear register	0x0038	WT
LP_SPI_DMA_INT_RAW_REG	LP-SPI interrupt raw register	0x003C	R/WTC/SS
LP_SPI_DMA_INT_ST_REG	LP-SPI interrupt status register	0x0040	RO
LP_SPI_DMA_INT_SET_REG	LP-SPI interrupt software set register	0x004C	WT
CPU-controlled data buffer			
LP_SPI_W0_REG	LP-SPI CPU-controlled buffer 0	0x0098	R/W/SS
LP_SPI_W1_REG	LP-SPI CPU-controlled buffer 1	0x009C	R/W/SS
LP_SPI_W2_REG	LP-SPI CPU-controlled buffer 2	0x00A0	R/W/SS
LP_SPI_W3_REG	LP-SPI CPU-controlled buffer 3	0x00A4	R/W/SS
LP_SPI_W4_REG	LP-SPI CPU-controlled buffer 4	0x00A8	R/W/SS
LP_SPI_W5_REG	LP-SPI CPU-controlled buffer 5	0x00AC	R/W/SS
LP_SPI_W6_REG	LP-SPI CPU-controlled buffer 6	0x00B0	R/W/SS
LP_SPI_W7_REG	LP-SPI CPU-controlled buffer 7	0x00B4	R/W/SS
LP_SPI_W8_REG	LP-SPI CPU-controlled buffer 8	0x00B8	R/W/SS
LP_SPI_W9_REG	LP-SPI CPU-controlled buffer 9	0x00BC	R/W/SS
LP_SPI_W10_REG	LP-SPI CPU-controlled buffer 10	0x00C0	R/W/SS
LP_SPI_W11_REG	LP-SPI CPU-controlled buffer 11	0x00C4	R/W/SS
LP_SPI_W12_REG	LP-SPI CPU-controlled buffer 12	0x00C8	R/W/SS
LP_SPI_W13_REG	LP-SPI CPU-controlled buffer 13	0x00CC	R/W/SS
LP_SPI_W14_REG	LP-SPI CPU-controlled buffer 14	0x00D0	R/W/SS
LP_SPI_W15_REG	LP-SPI CPU-controlled buffer 15	0x00D4	R/W/SS
Version register			
LP_SPI_DATE_REG	Version control	0x00F0	R/W

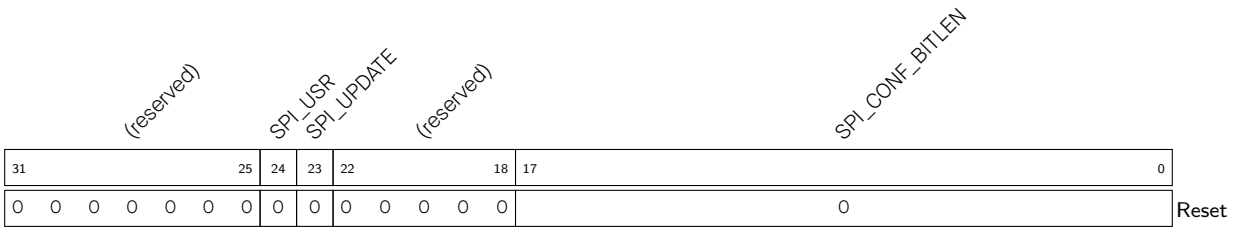
46.13 Register

46.13.1 GP-SPI2 Register

The addresses in this section are relative to GP-SPI2 base address provided in Table 9.3-2 in Chapter 9 *System and Memory*.

For how to program reserved fields, please refer to Section *Programming Reserved Register Field*.

Register 46.1. SPI_CMD_REG (0x0000)



SPI_CONF_BITLEN Configures the SPI_CLK cycles of SPI CONF state.

Measurement unit: SPI_CLK clock cycle.

Can be configured in CONF state.

(R/W)

SPI_UPDATE Configures whether or not to synchronize SPI registers from APB clock domain into SPI module clock domain.

0: Not synchronize

1: Synchronize

This bit is only used in SPI master transfer.

(WT)

SPI_USR Configures whether or not to enable user-defined command.

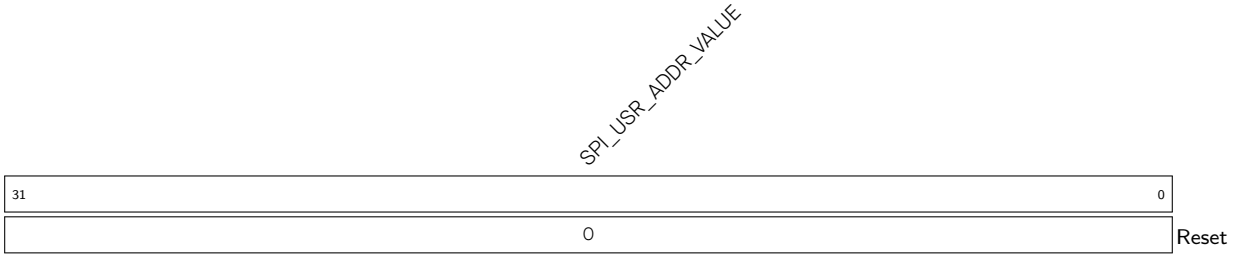
0: Not enable

1: Enable

An SPI operation will be triggered when the bit is set. This bit will be cleared once the operation is done. Can not be changed by CONF_buf.

(R/W/SC)

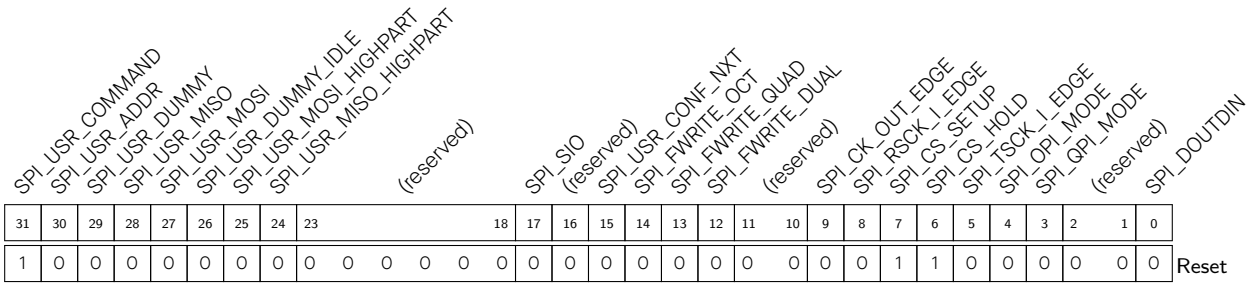
Register 46.2. SPI_ADDR_REG (0x0004)



SPI_USR_ADDR_VALUE Configures the address to slave. Can be configured in CONF state.

(R/W)

Register 46.3. SPI_USER_REG (0x0010)



SPI_DOUTDIN Configures whether or not to enable full-duplex communication.

0: Disable

1: Enable

Can be configured in CONF state.

(R/W)

SPI_QPI_MODE Configures whether or not to enable QPI mode.

0: Disable

1: Enable

This configuration is applicable when the SPI controller works as master or slave. Can be configured in CONF state.

(R/W/SS/SC)

SPI_OPI_MODE Configures whether or not to enable OPI mode.

0: Disable. SPI controller works in other modes.

1: Enable. SPI controller works in OPI mode, all phases of which are in 8-bit mode.

This configuration is only applicable when the SPI controller works as master. Can be configured in CONF state.

(R/W)

SPI_TSCK_I_EDGE Configures whether or not to change the polarity of TSCK in slave transfer.

0: TSCK = SPI_CK_I

1: TSCK = !SPI_CK_I

(R/W)

SPI_CS_HOLD Configures whether or not to keep SPI CS low when SPI is in DONE state.

0: Not keep low

1: Keep low

Can be configured in CONF state.

(R/W)

SPI_CS_SETUP Configures whether or not to enable SPI CS when SPI is in prepare (PREP) state.

0: Disable

1: Enable

Can be configured in CONF state.

(R/W)

Continued on the next page...

Register 46.3. SPI_USER_REG (0x0010)

Continued from the previous page...

SPI_RSCK_I_EDGE Configures whether or not to change the polarity of RSCK in slave transfer.
0: RSCK = !SPI_CK_I
1: RSCK = SPI_CK_I
(R/W)

SPI_CK_OUT_EDGE Configures SPI clock mode together with SPI_CK_IDLE_EDGE. Can be configured in CONF state. For more information, see Section 46.7.4.
(R/W)

SPI_FWRITE_DUAL Configures whether or not to enable the 2-bit mode of read-data phase in write operations.
0: Not enable
1: Enable
Can be configured in CONF state.
(R/W)

SPI_FWRITE_QUAD Configures whether or not to enable the 4-bit mode of read-data phase in write operations.
0: Not enable
1: Enable
Can be configured in CONF state.
(R/W)

SPI_FWRITE_OCT Configures whether or not to enable the 8-bit mode of read-data phase in write operations.
0: Not enable
1: Enable
Can be configured in CONF state.
(R/W)

SPI_USR_CONF_NXT Configures whether or not to enable the CONF state for the next transaction (segment) in a configurable segmented transfer.
0: This transfer will end after the current transaction (segment) is finished. Or this is not a configurable segmented transfer.
1: This configurable segmented transfer will continue its next transaction (segment).
Can be configured in CONF state.
(R/W)

SPI_SIO Configures whether or not to enable 3-line half-duplex communication, where MOSI and MISO signals share the same pin.
0: Disable
1: Enable
Can be configured in CONF state.
(R/W)

Continued on the next page...

Register 46.3. SPI_USER_REG (0x0010)

Continued from the previous page...

SPI_USR_MISO_HIGHPART Configures whether or not to enable High-Part mode in read-data phase, i.e., only access to high-part of the buffers: SPI_W8_REG ~ SPI_W15_REG.
0: Disable
1: Enable
Can be configured in CONF state.
(R/W)

SPI_USR_MOSI_HIGHPART Configures whether or not to enable High-Part mode in write-data phase, i.e., only access to high-part of the buffers: SPI_W8_REG ~ SPI_W15_REG.
0: Disable
1: Enable
Can be configured in CONF state.
(R/W)

SPI_USR_DUMMY_IDLE Configures whether or not to disable SPI clock in DUMMY state.
0: Not disable
1: Disable
Can be configured in CONF state.
(R/W)

SPI_USR_MOSI Configures whether or not to enable the write-data (DOUT) state of an operation.
0: Disable
1: Enable
Can be configured in CONF state.
(R/W)

SPI_USR_MISO Configures whether or not to enable the read-data (DIN) state of an operation.
0: Disable
1: Enable
Can be configured in CONF state.
(R/W)

SPI_USR_DUMMY Configures whether or not to enable the DUMMY state of an operation.
0: Disable
1: Enable
Can be configured in CONF state.
(R/W)

SPI_USR_ADDR Configures whether or not to enable the address (ADDR) state of an operation.
0: Disable
1: Enable
Can be configured in CONF state.
(R/W)

Continued on the next page...

Register 46.3. SPI_USER_REG (0x0010)

Continued from the previous page...

SPI_USR_COMMAND Configures whether or not to enable the command (CMD) state of an operation.

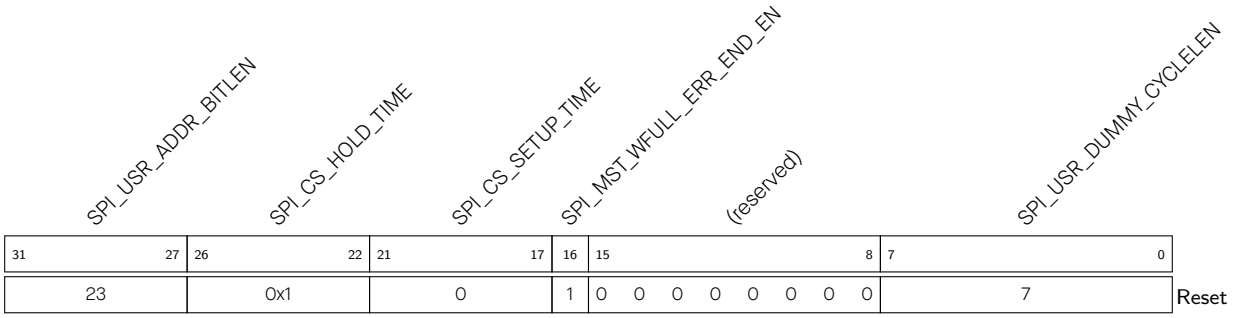
0: Disable

1: Enable

Can be configured in CONF state.

(R/W)

Register 46.4. SPI_USER1_REG (0x0014)



SPI_USR_DUMMY_CYCLELEN Configures the length of DUMMY state.

Measurement unit: SPI_CLK clock cycles.

This value is (the expected cycle number - 1).

Can be configured in CONF state.

(R/W)

SPI_MST_WFULL_ERR_END_EN Configures whether or not to end the SPI transfer when SPI RX AFIFO wfull error occurs in master full-/half-duplex transfers.

0: Not end

1: End

(R/W)

SPI_CS_SETUP_TIME Configures the length of prepare (PREP) state.

Measurement unit: SPI_CLK clock cycles.

This value is equal to the expected cycles - 1. This field is used together with SPI_CS_SETUP.

Can be configured in CONF state.

(R/W)

SPI_CS_HOLD_TIME Configures the delay cycles of CS pin.

Measurement unit: SPI_CLK clock cycles.

This field is used together with SPI_CS_HOLD.

Can be configured in CONF state.

(R/W)

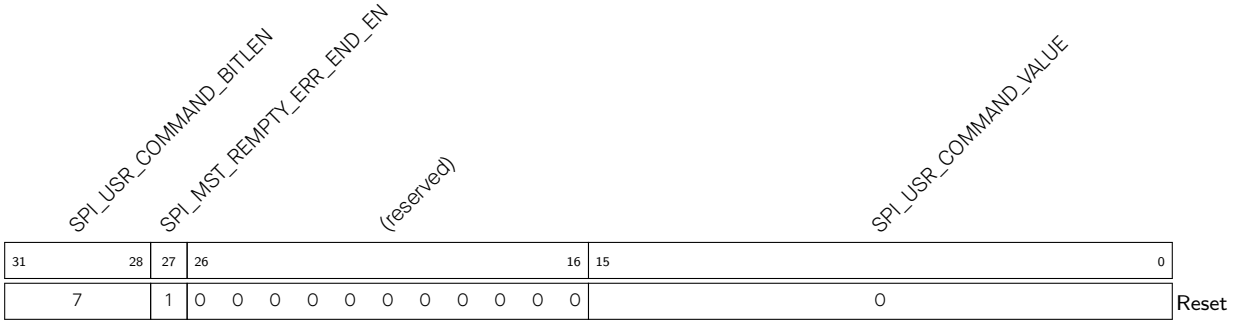
SPI_USR_ADDR_BITLEN Configures the bit length in address state.

This value is (expected bit number - 1).

Can be configured in CONF state.

(R/W)

Register 46.5. SPI_USER2_REG (0x0018)



SPI_USR_COMMAND_VALUE Configures the command value.

Can be configured in CONF state.

(R/W)

SPI_MST_REMPTY_ERR_END_EN Configures whether or not to end the SPI transfer when SPI TX AFIFO read empty error occurs in master full-/half-duplex transfers.

0: Not end

1: End

(R/W)

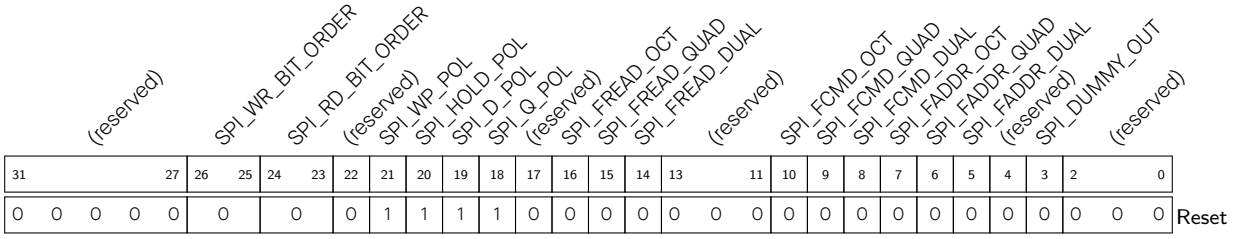
SPI_USR_COMMAND_BITLEN Configures the bit length of command state.

This value is (expected bit number - 1).

Can be configured in CONF state.

(R/W)

Register 46.6. SPI_CTRL_REG (0x0008)



SPI_DUMMY_OUT Configures whether or not to output the SPI2 bus signals in DUMMY state.

0: Not output

1: Output

Can be configured in CONF state.

(R/W)

SPI_FADDR_DUAL Configures whether or not to enable 2-bit mode during address (ADDR) state.

0: Disable

1: Enable

Can be configured in CONF state.

(R/W)

SPI_FADDR_QUAD Configures whether or not to enable 4-bit mode during address (ADDR) state.

0: Disable

1: Enable

Can be configured in CONF state.

(R/W)

SPI_FADDR_OCT Configures whether or not to enable 8-bit mode during address (ADDR) state.

0: Disable

1: Enable

Can be configured in CONF state.

(R/W)

SPI_FCMD_DUAL Configures whether or not to enable 2-bit mode during command (CMD) state.

0: Disable

1: Enable

Can be configured in CONF state.

(R/W)

SPI_FCMD_QUAD Configures whether or not to enable 4-bit mode during command (CMD) state.

0: Disable

1: Enable

Can be configured in CONF state.

(R/W)

Continued on the next page...

Register 46.6. SPI_CTRL_REG (0x0008)

Continued from the previous page...

SPI_FCMD_OCT Configures whether or not to enable 8-bit mode during command (CMD) state.
0: Disable
1: Enable
Can be configured in CONF state.
(R/W)

SPI_FREAD_DUAL Configures whether or not to enable the 2-bit mode of read-data (DIN) state in read operations.
0: Disable
1: Enable
Can be configured in CONF state.
(R/W)

SPI_FREAD_QUAD Configures whether or not to enable the 4-bit mode of read-data (DIN) state in read operations.
0: Disable
1: Enable
Can be configured in CONF state.
(R/W)

SPI_FREAD_OCT Configures whether or not to enable the 8-bit mode of read-data (DIN) state in read operations.
0: Disable
1: Enable
Can be configured in CONF state. (R/W)

SPI_Q_POL Configures MISO line polarity.
0: Low
1: High
Can be configured in CONF state.
(R/W)

SPI_D_POL Configures MOSI line polarity.
0: Low
1: High
Can be configured in CONF state.
(R/W)

SPI_HOLD_POL Configures SPI_HOLD output value when SPI is in idle.
0: Output low
1: Output high
Can be configured in CONF state.
(R/W)

Continued on the next page...

Register 46.6. SPI_CTRL_REG (0x0008)

Continued from the previous page...

SPI_WP_POL Configures the output value of write-protect signal when SPI is in idle.

0: Output low

1: Output high

Can be configured in CONF state.

(R/W)

SPI_RD_BIT_ORDER Configures the bit order in read-data (MISO) state.

0: MSB first

1: LSB first

2: MSB first

3: LSB first

Can be configured in CONF state. For more information, see Table 46.5-6.

(R/W)

SPI_WR_BIT_ORDER Configures the bit order in command (CMD), address (ADDR), and write-data (MOSI) states.

0: MSB first

1: LSB first

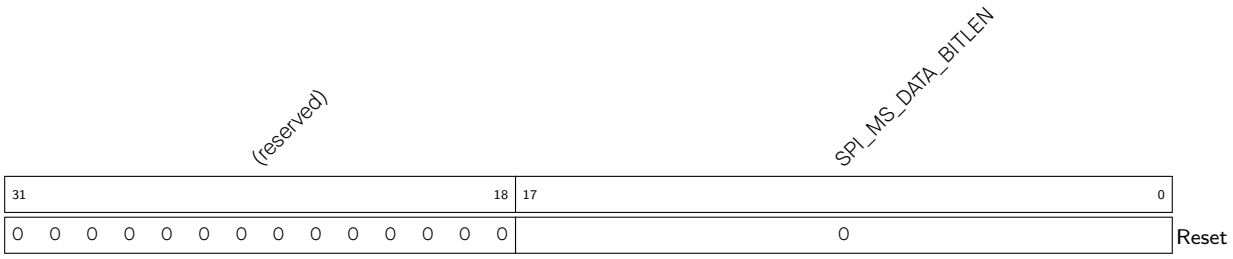
2: MSB first

3: LSB first

Can be configured in CONF state. For more information, see Table 46.5-6.

(R/W)

Register 46.7. SPI_MS_DLEN_REG (0x001C)



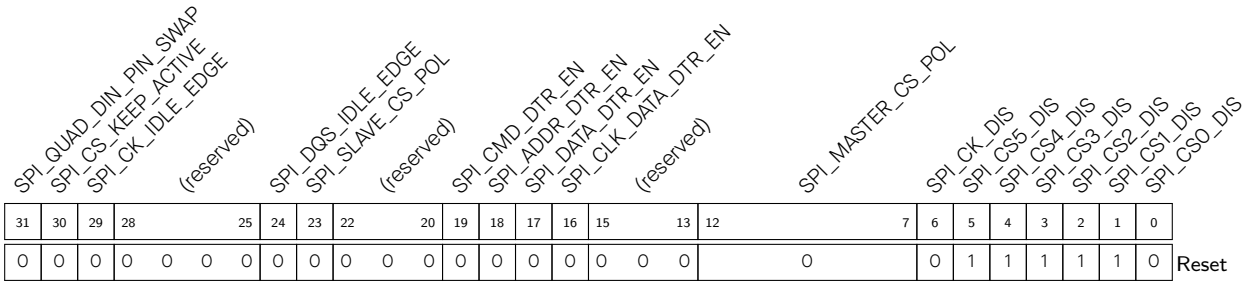
SPI_MS_DATA_BITLEN Configures the data bit length of SPI transfer in DMA-controlled master transfer or in CPU-controlled master transfer. Or configures the bit length of SPI RX transfer in DMA-controlled slave transfer.

This value shall be (expected bit_num - 1).

Can be configured in CONF state.

(R/W)

Register 46.8. SPI_MISC_REG (0x0020)



SPI_CS0_DIS Configures whether or not to disable SPI_CS0 pin.

0: SPI_CS0 signal is from/to SPI_CS0 pin.

1: Disable SPI_CS0 pin.

Can be configured in CONF state.

(R/W)

SPI_CS1_DIS Configures whether or not to disable SPI_CS1 pin.

0: SPI_CS1 signal is from/to SPI_CS1 pin.

1: Disable SPI_CS1 pin.

Can be configured in CONF state.

(R/W)

SPI_CS2_DIS Configures whether or not to disable SPI_CS2 pin.

0: SPI_CS2 signal is from/to SPI_CS2 pin.

1: Disable SPI_CS2 pin.

Can be configured in CONF state.

(R/W)

SPI_CS3_DIS Configures whether or not to disable SPI_CS3 pin.

0: SPI_CS3 signal is from/to SPI_CS3 pin.

1: Disable SPI_CS3 pin.

Can be configured in CONF state.

(R/W)

SPI_CS4_DIS Configures whether or not to disable SPI_CS4 pin.

0: SPI_CS4 signal is from/to SPI_CS4 pin.

1: Disable SPI_CS4 pin.

Can be configured in CONF state.

(R/W)

SPI_CS5_DIS Configures whether or not to disable SPI_CS5 pin.

0: SPI_CS5 signal is from/to SPI_CS5 pin.

1: Disable SPI_CS5 pin.

Can be configured in CONF state.

(R/W)

Continued on the next page...

Register 46.8. SPI_MISC_REG (0x0020)

Continued from the previous page...

SPI_CK_DIS Configures whether or not to disable SPI_CLK output.

0: Enable

1: Disable

Can be configured in CONF state.

(R/W)

SPI_MASTER_CS_POL Configures the polarity of SPI_CS*n* (*n* = 0∼5) line in master transfer.

0: SPI_CS*n* is low active.

1: SPI_CS*n* is high active.

Can be configured in CONF state.

(R/W)

SPI_CLK_DATA_DTR_EN Configures whether or not to enable DTR mode for SPI_CLK, DATA, and SPI_DQS when the SPI works as master.

0: Enable DTR mode only for SPI_DQS

1: Enable DTR mode for SPI_CLK, DATA, and SPI_DQS.

This bit should be used with SPI_DATA_DTR_EN, SPI_ADDR_DTR_EN, and SPI_CMD_DTR_EN.

(R/W)

SPI_DATA_DTR_EN Configures whether or not to enable DTR mode for SPI_CLK and DATA in DOUT and DIN states when the SPI works as master in 1/2/4/8-bit mode.

0: Not enable

1: Enable

Can be configured in CONF state.

(R/W)

SPI_ADDR_DTR_EN Configures whether or not to enable DTR mode for SPI_CLK and DATA in ADDR state when the SPI works as master in 1/2/4/8-bit mode.

0: Not enable

1: Enable

Can be configured in CONF state.

(R/W)

SPI_CMD_DTR_EN Configures whether or not to enable DTR mode for SPI_CLK and DATA in CMD state when the SPI works as master in 1/2/4/8-bit mode.

0: Not enable

1: Enable

Can be configured in CONF state.

(R/W)

SPI_SLAVE_CS_POL Configures whether or not to invert SPI slave input CS polarity.

0: Not change

1: Invert

Can be configured in CONF state.

(R/W)

Continued on the next page...

Register 46.8. SPI_MISC_REG (0x0020)

Continued from the previous page...

SPI_DQS_IDLE_EDGE Configured the default value of SPI_DQS.

0: Low

1: High

Can be configured in CONF state.

(R/W)

SPI_CK_IDLE_EDGE Configures the level of SPI_CLK line when GP-SPI2 is in idle.

0: Low

1: High

Can be configured in CONF state.

(R/W)

SPI_CS_KEEP_ACTIVE Configures whether or not to keep the SPI_CS line low.

0: Not keep low

1: Keep low

Can be configured in CONF state.

(R/W)

SPI_QUAD_DIN_PIN_SWAP Configures whether or not to swap SPI Quad input pins.

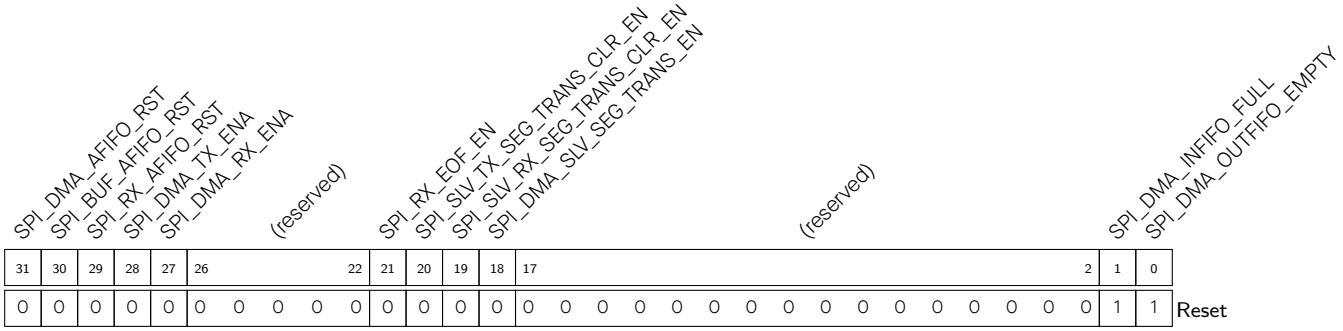
0: Not swap

1: Swap SPI2D with SPI2Q, and SPI2WP with SPI2HD

Can be configured in CONF state.

(R/W)

Register 46.9. SPI_DMA_CONF_REG (0x0030)



SPI_DMA_OUTFIFO_EMPTY Represents whether or not the DMA TX FIFO is ready for sending data.

0: Ready

1: Not ready

(RO)

SPI_DMA_INFIFO_FULL Represents whether or not the DMA RX FIFO is ready for receiving data.

0: Ready

1: Not ready

(RO)

SPI_DMA_SLV_SEG_TRANS_EN Configures whether or not to enable DMA-controlled segmented transfer in slave half-duplex communication.

0: Disable

1: Enable

(R/W)

SPI_SLV_RX_SEG_TRANS_CLR_EN In slave segmented transfer, if the size of the DMA RX buffer is smaller than the size of the received data,

1: the data in all the following Wr_DMA transactions will not be received.

0: the data in this Wr_DMA transaction will not be received, but in the following transactions,

- if the size of DMA RX buffer is not 0, the data in following Wr_DMA transactions will be received.
- if the size of DMA RX buffer is 0, the data in following Wr_DMA transactions will not be received.

(R/W)

SPI_SLV_TX_SEG_TRANS_CLR_EN In slave segmented transfer, if the size of the DMA TX buffer is smaller than the size of the transmitted data,

1: the data in the following transactions will not be updated, i.e. the old data is transmitted repeatedly.

0: the data in this transaction will not be updated. But in the following transactions,

- if new data is filled in DMA TX FIFO, new data will be transmitted.
- if no new data is filled in DMA TX FIFO, no new data will be transmitted.

(R/W)

Continued on the next page...

Register 46.9. SPI_DMA_CONF_REG (0x0030)

Continued from the previous page...

SPI_RX_EOF_EN Configures the trigger source of GDMA_IN_SUC_EOF_CH*n*_INT_RAW.

0: GDMA_IN_SUC_EOF_CH*n*_INT_RAW is set by SPI_TRANS_DONE_INT event in a single transfer, or by an SPI_DMA_SEG_TRANS_DONE_INT event in a segmented transfer.

1: In a DAM-controlled transfer, if the bit number of transferred data is equal to (SPI_MS_DATA_BITLEN + 1), then GDMA_IN_SUC_EOF_CH*n*_INT_RAW will be set by hardware. (R/W)

SPI_DMA_RX_ENA Configures whether or not to enable DMA-controlled receive data transfer.

0: Disable

1: Enable

(R/W)

SPI_DMA_TX_ENA Configures whether or not to enable DMA-controlled send data transfer.

0: Disable

1: Enable

(R/W)

SPI_RX_AFIFO_RST Configures whether or not to reset RX AFIFO (i.e., spi_rx_afifo in Figure 46.5-3 and Figure 46.5-4), which is used to receive data in SPI master and slave transfer.

0: Not reset

1: Reset

(WT)

SPI_BUF_AFIFO_RST Configures whether or not to reset BUF TX AFIFO (i.e., buf_tx_afifo in Figure 46.5-3 and Figure 46.5-4), which is used to send data out in SPI slave CPU-controlled transfer and master transfer.

0: Not reset

1: Reset

(WT)

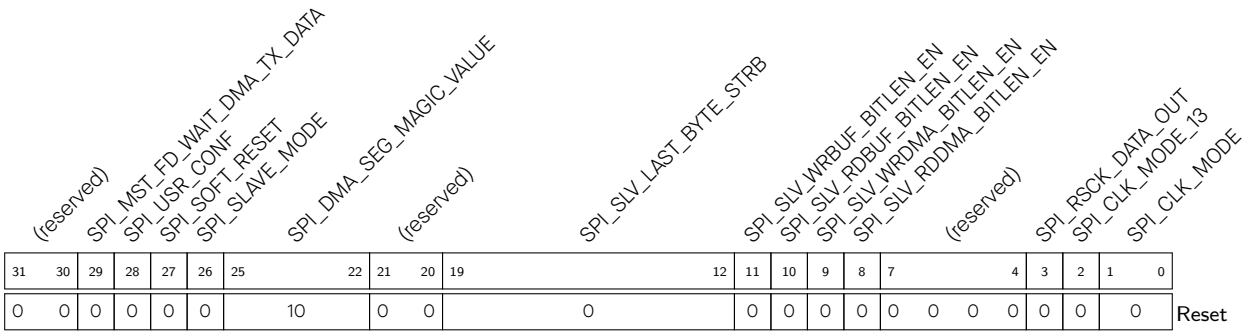
SPI_DMA_AFIFO_RST Configures whether or not to reset DMA TX AFIFO (i.e., dma_tx_afifo in Figure 46.5-3 and Figure 46.5-4), which is used to send data out in SPI slave DMA-controlled transfer.

0: Not reset

1: Reset

(WT)

Register 46.10. SPI_SLAVE_REG (0x00E0)



SPI_CLK_MODE Configures SPI clock mode.

0: SPI clock is off when CS becomes inactive.

1: SPI clock is delayed one cycle after CS becomes inactive.

2: SPI clock is delayed two cycles after CS becomes inactive.

3: SPI clock is always on.

Can be configured in CONF state.

(R/W)

SPI_CLK_MODE_13 Configures clock mode.

0: SPI clock mode 1 and mode 3. See Table 46.7-2.

1: SPI clock mode 0 and mode 2. See Table 46.7-2.

(R/W)

SPI_RSCK_DATA_OUT Configures the edge of output data.

0: Output data at TSCK rising edge.

1: Output data at RSCK rising edge.

(R/W)

SPI_SLV_RDDMA_BITLEN_EN Configures whether or not to use SPI_SLV_DATA_BITLEN to store the data bit length of Rd_DMA transfer.

0: Not use

1: Use

(R/W)

SPI_SLV_WRDMA_BITLEN_EN Configures whether or not to use SPI_SLV_DATA_BITLEN to store the data bit length of Wr_DMA transfer.

0: Not use

1: Use

(R/W)

SPI_SLV_RDBUF_BITLEN_EN Configures whether or not to use SPI_SLV_DATA_BITLEN to store the data bit length of Rd_BUF transfer.

0: Not use

1: Use

(R/W)

Continued on the next page...

Register 46.10. SPI_SLAVE_REG (0x00E0)

Continued from the previous page...

SPI_SLV_WRBUF_BITLEN_EN Configures whether or not to use SPI_SLV_DATA_BITLEN to store the data bit length of Wr_BUF transfer.
0: Not use
1: Use
(R/W)

SPI_SLV_LAST_BYTE_STRB Represents the valid bits of the last received data byte in SPI slave full-duplex and half-duplex transfer.
(R/SS)

SPI_DMA_SEG_MAGIC_VALUE Configures the magic value of BM table in DMA-controlled configurable segmented transfer.
(R/W)

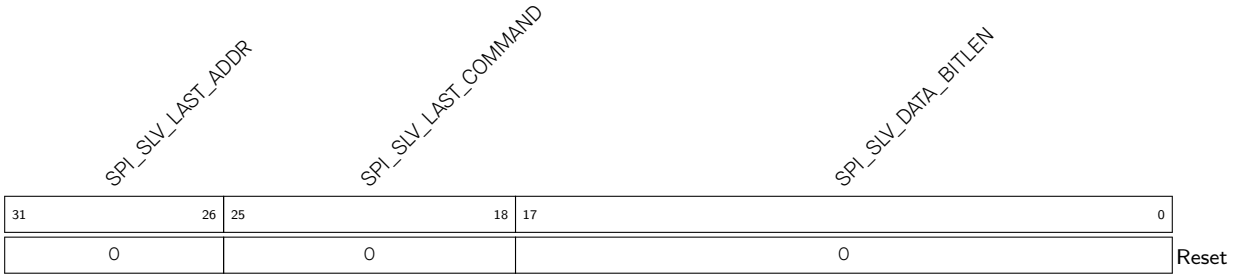
SPI_SLAVE_MODE Configures SPI work mode.
0: Master
1: Slave
(R/W)

SPI_SOFT_RESET Configures whether or not to reset the SPI clock line, CS line, and data line via software.
0: Not reset
1: Reset
Can be configured in CONF state.
(WT)

SPI_USR_CONF Configures whether or not to enable the CONF state of current DMA-controlled configurable segmented transfer.
0: No effect, which means the current transfer is not a configurable segmented transfer.
1: Enable, which means a configurable segmented transfer is started.
(R/W)

SPI_MST_FD_WAIT_DMA_TX_DATA Configures whether or not to wait DMA TX data gets ready before starting SPI transfer in master full-duplex transfer.
0: Not wait
1: Wait
(R/W)

Register 46.11. SPI_SLAVE1_REG (0x00E4)

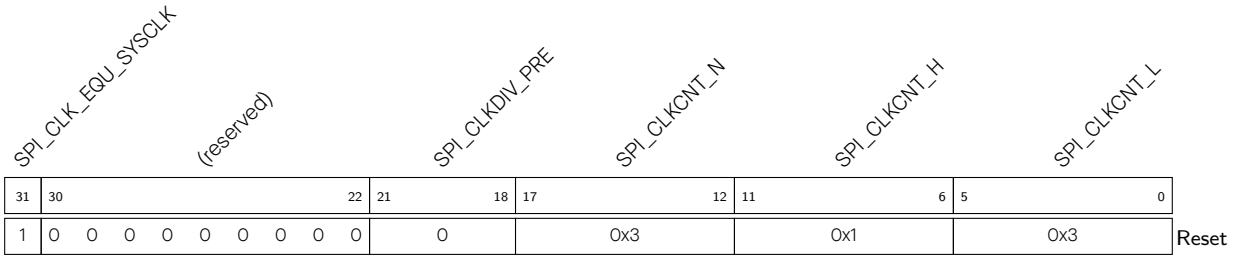


SPI_SLV_DATA_BITLEN Configures the transferred data bit length in SPI slave full-/half-duplex modes.
(R/W/SS)

SPI_SLV_LAST_COMMAND Configures the command value in slave mode.
(R/W/SS)

SPI_SLV_LAST_ADDR Configures the address value in slave mode.
(R/W/SS)

Register 46.12. SPI_CLOCK_REG (0x000C)



SPI_CLKCNT_L Configures clock duty cycles, together with SPI_CLKCNT_H and SPI_CLKCNT_N.

In master transfer, this field must be equal to SPI_CLKCNT_N.

In slave mode, it must be 0.

Can be configured in CONF state.

(R/W)

SPI_CLKCNT_H Configures the duty cycle of SPI_CLK (high level) in master transfer.

It's recommended to configure this value to floor((SPI_CLKCNT_N + 1)/2 - 1). floor() here is to round a number down, e.g., floor(2.2) = 2.

In slave mode, it must be 0.

Can be configured in CONF state.

(R/W)

SPI_CLKCNT_N Configures the divider of SPI_CLK in master transfer.

SPI_CLK frequency is $f_{clk_spi_mst}$/(SPI_CLKDIV_PRE + 1)/(SPI_CLKCNT_N + 1).

Can be configured in CONF state.

(R/W)

SPI_CLKDIV_PRE Configures the pre-divider of SPI_CLK in master transfer.

Can be configured in CONF state.

(R/W)

SPI_CLK_EQU_SYSCLK Configures whether or not the SPI_CLK is equal to clk_spi_mst in master transfer.

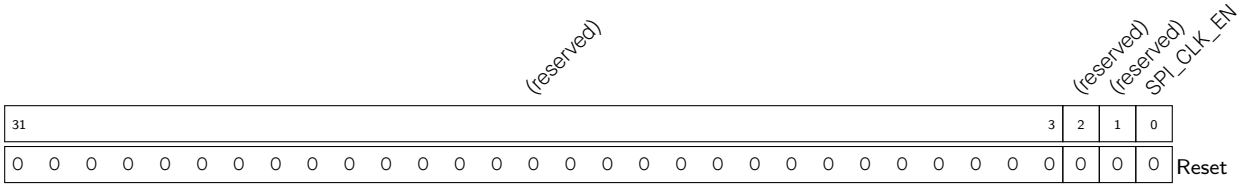
0: SPI_CLK is divided from clk_spi_mst.

1: SPI_CLK is equal to clk_spi_mst.

Can be configured in CONF state.

(R/W)

Register 46.13. SPI_CLK_GATE_REG (0x00E8)



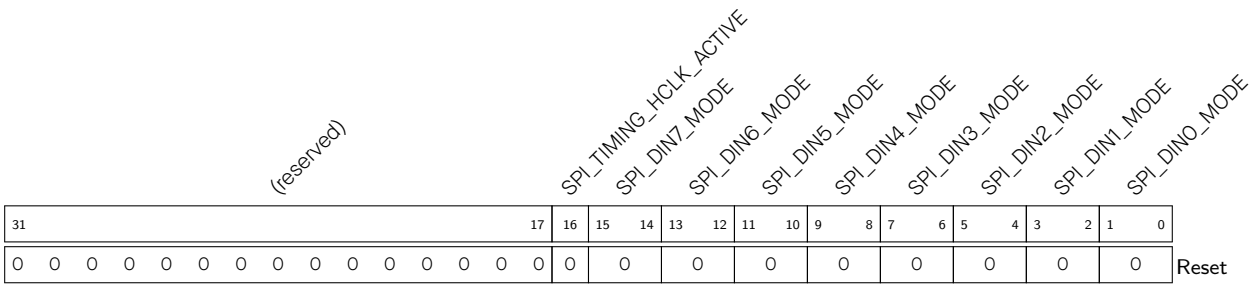
SPI_CLK_EN Configures whether or not to enable clock gate.

0: Disable

1: Enable

(R/W)

Register 46.14. SPI_DIN_MODE_REG (0x0024)



SPI_DIN0_MODE Configures the input mode for SPI2D signal.

0: Input without delay

1: Input at the (SPI_DIN0_NUM + 1)th falling edge of clk_spi_mst

2: Input at the (SPI_DIN0_NUM + 1)th rising edge of clk_hclk plus one clk_spi_mst rising edge cycle

3: Input at the (SPI_DIN0_NUM + 1)th rising edge of clk_hclk plus one clk_spi_mst falling edge cycle

Can be configured in CONF state.

(R/W)

SPI_DIN1_MODE Configures the input mode for SPI2Q signal.

0: Input without delay

1: Input at the (SPI_DIN1_NUM+1)th falling edge of clk_spi_mst

2: Input at the (SPI_DIN1_NUM + 1)th rising edge of clk_hclk plus one clk_spi_mst rising edge cycle

3: Input at the (SPI_DIN1_NUM + 1)th rising edge of clk_hclk plus one clk_spi_mst falling edge cycle

Can be configured in CONF state.

(R/W)

SPI_DIN2_MODE Configures the input mode for SPI2WP signal.

0: Input without delay

1: Input at the (SPI_DIN2_NUM + 1)th falling edge of clk_spi_mst

2: Input at the (SPI_DIN2_NUM + 1)th rising edge of clk_hclk plus one clk_spi_mst rising edge cycle

3: Input at the (SPI_DIN2_NUM + 1)th rising edge of clk_hclk plus one clk_spi_mst falling edge cycle

Can be configured in CONF state.

(R/W)

Continued on the next page...

Register 46.14. SPI_DIN_MODE_REG (0x0024)

Continued from the previous page...

SPI_DIN3_MODE Configures the input mode for SPI2HD signal.

0: Input without delay

1: Input at the (SPI_DIN3_NUM + 1)th falling edge of clk_spi_mst

2: Input at the (SPI_DIN3_NUM + 1)th rising edge of clk_hclk plus one clk_spi_mst rising edge cycle

3: Input at the (SPI_DIN3_NUM + 1)th rising edge of clk_hclk plus one clk_spi_mst falling edge cycle

Can be configured in CONF state.

(R/W)

SPI_DIN4_MODE Configures the input mode for SPI2D4 signal.

0: Input without delay

1: Input at the (SPI_DIN4_NUM + 1)th falling edge of clk_spi_mst

2: Input at the (SPI_DIN4_NUM + 1)th rising edge of clk_hclk plus one clk_spi_mst rising edge cycle

3: Input at the (SPI_DIN4_NUM + 1)th rising edge of clk_hclk plus one clk_spi_mst falling edge cycle

Can be configured in CONF state.

(R/W)

SPI_DIN5_MODE Configures the input mode for SPI2D5 signal.

0: Input without delay

1: Input at the (SPI_DIN5_NUM + 1)th falling edge of clk_spi_mst

2: Input at the (SPI_DIN5_NUM + 1)th rising edge of clk_hclk plus one clk_spi_mst rising edge cycle

3: Input at the (SPI_DIN5_NUM + 1)th rising edge of clk_hclk plus one clk_spi_mst falling edge cycle

Can be configured in CONF state.

(R/W)

SPI_DIN6_MODE Configures the input mode for SPI2D6 signal.

0: Input without delay

1: Input at the (SPI_DIN6_NUM + 1)th falling edge of clk_spi_mst

2: Input at the (SPI_DIN6_NUM + 1)th rising edge of clk_hclk plus one clk_spi_mst rising edge cycle

3: Input at the (SPI_DIN6_NUM + 1)th rising edge of clk_hclk plus one clk_spi_mst falling edge cycle

Can be configured in CONF state.

(R/W)

Continued on the next page...

Register 46.14. SPI_DIN_MODE_REG (0x0024)

Continued from the previous page...

SPI_DIN7_MODE Configures the input mode for SPI2D7 signal.

0: Input without delay

1: Input at the (SPI_DIN7_NUM + 1)th falling edge of clk_spi_mst

2: Input at the (SPI_DIN7_NUM + 1)th rising edge of clk_hclk plus one clk_spi_mst rising edge cycle

3: Input at the (SPI_DIN7_NUM + 1)th rising edge of clk_hclk plus one clk_spi_mst falling edge cycle

Can be configured in CONF state.

(R/W)

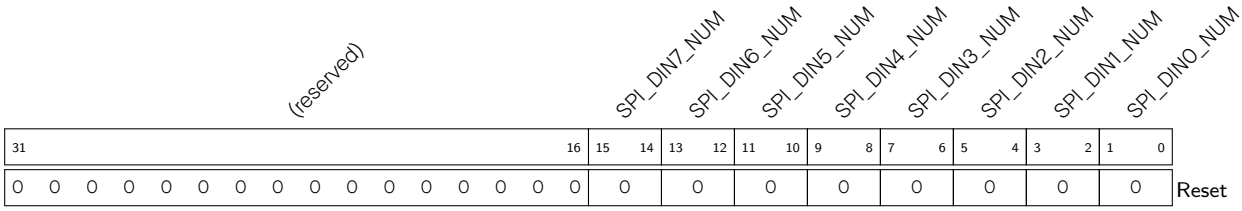
SPI_TIMING_HCLK_ACTIVE Configures whether or not to enable HCLK (high-frequency clock) in SPI input timing module.

0: Disable

1: Enable

Can be configured in CONF state. (R/W)

Register 46.15. SPI_DIN_NUM_REG (0x0028)



SPI_DIN0_NUM Configures the delays to input signal SPI2D based on the setting of SPI_DIN0_MODE.
0: Delayed by 1 clock cycle
1: Delayed by 2 clock cycles
2: Delayed by 3 clock cycles
3: Delayed by 4 clock cycles
Can be configured in CONF state.
(R/W)

SPI_DIN1_NUM Configures the delays to input signal SPI2Q based on the setting of SPI_DIN1_MODE.
0: Delayed by 1 clock cycle
1: Delayed by 2 clock cycles
2: Delayed by 3 clock cycles
3: Delayed by 4 clock cycles
Can be configured in CONF state.
(R/W)

SPI_DIN2_NUM Configures the delays to input signal SPI2WP based on the setting of SPI_DIN2_MODE.
0: Delayed by 1 clock cycle
1: Delayed by 2 clock cycles
2: Delayed by 3 clock cycles
3: Delayed by 4 clock cycles
Can be configured in CONF state.
(R/W)

SPI_DIN3_NUM Configures the delays to input signal SPI2HD based on the setting of SPI_DIN3_MODE.
0: Delayed by 1 clock cycle
1: Delayed by 2 clock cycles
2: Delayed by 3 clock cycles
3: Delayed by 4 clock cycles
Can be configured in CONF state.
(R/W)

Continued on the next page...

Register 46.15. SPI_DIN_NUM_REG (0x0028)

Continued from the previous page...

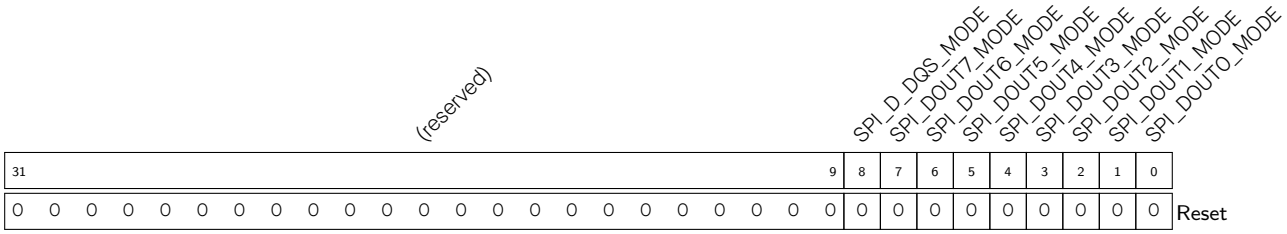
SPI_DIN4_NUM Configures the delays to input signal SPI2D4 based on the setting of SPI_DIN4_MODE.
0: Delayed by 1 clock cycle
1: Delayed by 2 clock cycles
2: Delayed by 3 clock cycles
3: Delayed by 4 clock cycles
Can be configured in CONF state.
(R/W)

SPI_DIN5_NUM Configures the delays to input signal SPI2D5 based on the setting of SPI_DIN5_MODE.
0: Delayed by 1 clock cycle
1: Delayed by 2 clock cycles
2: Delayed by 3 clock cycles
3: Delayed by 4 clock cycles
Can be configured in CONF state.
(R/W)

SPI_DIN6_NUM Configures the delays to input signal SPI2D6 based on the setting of SPI_DIN6_MODE.
0: Delayed by 1 clock cycle
1: Delayed by 2 clock cycles
2: Delayed by 3 clock cycles
3: Delayed by 4 clock cycles
Can be configured in CONF state.
(R/W)

SPI_DIN7_NUM Configures the delays to input signal SPI2D7 based on the setting of SPI_DIN7_MODE.
0: Delayed by 1 clock cycle
1: Delayed by 2 clock cycles
2: Delayed by 3 clock cycles
3: Delayed by 4 clock cycles
Can be configured in CONF state.
(R/W)

Register 46.16. SPI_DOUT_MODE_REG (0x002C)



SPI_DOUT0_MODE Configures the output mode for SPI2D signal.

0: Output without delay

1: Output with a delay of a SPI module clock cycle at its falling edge

Can be configured in CONF state.

(R/W)

SPI_DOUT1_MODE Configures the output mode for SPI2Q signal.

0: Output without delay

1: Output with a delay of a SPI module clock cycle at its falling edge

Can be configured in CONF state.

(R/W)

SPI_DOUT2_MODE Configures the output mode for SPI2WP signal.

0: Output without delay

1: Output with a delay of a SPI module clock cycle at its falling edge

Can be configured in CONF state.

(R/W)

SPI_DOUT3_MODE Configures the output mode for SPI2HD signal.

0: Output without delay

1: Output with a delay of a SPI module clock cycle at its falling edge

Can be configured in CONF state.

(R/W)

SPI_DOUT4_MODE Configures the output mode for SPI2D4 signal.

0: Output without delay

1: Output with a delay of a SPI module clock cycle at its falling edge

Can be configured in CONF state.

(R/W)

SPI_DOUT5_MODE Configures the output mode for SPI2D5 signal.

0: Output without delay

1: Output with a delay of a SPI module clock cycle at its falling edge

Can be configured in CONF state.

(R/W)

SPI_DOUT6_MODE Configures the output mode for SPI2D6 signal.

0: Output without delay

1: Output with a delay of a SPI module clock cycle at its falling edge

Can be configured in CONF state.

(R/W)

Continued on the next page...

Register 46.16. SPI_DOUT_MODE_REG (0x002C)

Continued from the previous page...

SPI_DOUT7_MODE Configures the output mode for SPI2D7 signal.

0: Output without delay

1: Output with a delay of a SPI module clock cycle at its falling edge

Can be configured in CONF state.

(R/W)

SPI_D_DQS_MODE Configures the output mode for output SPI_DQS signal.

0: Output without delay

1: Output is delayed for a SPI clock cycle at its falling edge.

Can be configured in CONF state.

(R/W)

Register 46.17. SPI_DMA_INT_ENA_REG (0x0034)



SPI_DMA_INFIFO_FULL_ERR_INT_ENA Write 1 to enable SPI_DMA_INFIFO_FULL_ERR_INT interrupt.
(R/W)

SPI_DMA_OUTFIFO_EMPTY_ERR_INT_ENA Write 1 to enable SPI_DMA_OUTFIFO_EMPTY_ERR_INT interrupt.
(R/W)

SPI_SLV_EX_QPI_INT_ENA Write 1 to enable SPI_SLV_EX_QPI_INT interrupt.
(R/W)

SPI_SLV_EN_QPI_INT_ENA Write 1 to enable SPI_SLV_EN_QPI_INT interrupt.
(R/W)

SPI_SLV_CMD7_INT_ENA Write 1 to enable SPI_SLV_CMD7_INT interrupt.
(R/W)

SPI_SLV_CMD8_INT_ENA Write 1 to enable SPI_SLV_CMD8_INT interrupt.
(R/W)

SPI_SLV_CMD9_INT_ENA Write 1 to enable SPI_SLV_CMD9_INT interrupt.
(R/W)

SPI_SLV_CMDA_INT_ENA Write 1 to enable SPI_SLV_CMDA_INT interrupt.
(R/W)

SPI_SLV_RD_DMA_DONE_INT_ENA Write 1 to enable SPI_SLV_RD_DMA_DONE_INT interrupt.
(R/W)

SPI_SLV_WR_DMA_DONE_INT_ENA Write 1 to enable SPI_SLV_WR_DMA_DONE_INT interrupt.
(R/W)

SPI_SLV_RD_BUF_DONE_INT_ENA Write 1 to enable SPI_SLV_RD_BUF_DONE_INT interrupt.
(R/W)

SPI_SLV_WR_BUF_DONE_INT_ENA Write 1 to enable SPI_SLV_WR_BUF_DONE_INT interrupt.
(R/W)

SPI_TRANS_DONE_INT_ENA Write 1 to enable SPI_TRANS_DONE_INT interrupt.
(R/W)

Continued on the next page...

Register 46.17. SPI_DMA_INT_ENA_REG (0x0034)

Continued from the previous page...

SPI_DMA_SEG_TRANS_DONE_INT_ENA Write 1 to enable SPI_DMA_SEG_TRANS_DONE_INT interrupt.
(R/W)

SPI_SEG_MAGIC_ERR_INT_ENA Write 1 to enable SPI_SEG_MAGIC_ERR_INT interrupt.
(R/W)

SPI_SLV_BUF_ADDR_ERR_INT_ENA Write 1 to enable SPI_SLV_BUF_ADDR_ERR_INT interrupt.
(R/W)

SPI_SLV_CMD_ERR_INT_ENA Write 1 to enable SPI_SLV_CMD_ERR_INT interrupt.
(R/W)

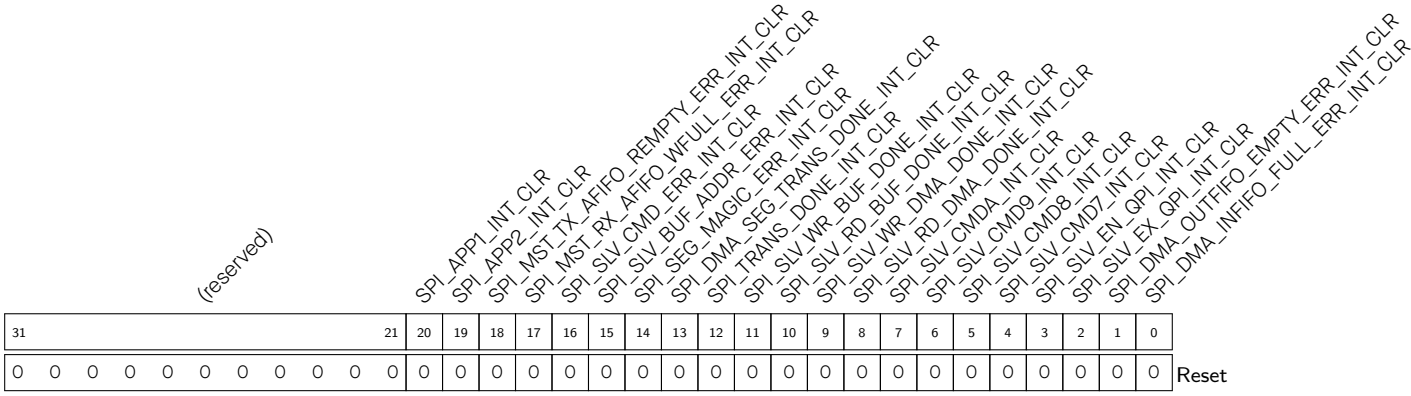
SPI_MST_RX_AFIFO_WFULL_ERR_INT_ENA Write 1 to enable SPI_MST_RX_AFIFO_WFULL_ERR_INT interrupt.
(R/W)

SPI_MST_TX_AFIFO_REMPTY_ERR_INT_ENA Write 1 to enable SPI_MST_TX_AFIFO_REMPTY_ERR_INT interrupt.
(R/W)

SPI_APP2_INT_ENA Write 1 to enable SPI_APP2_INT interrupt.
(R/W)

SPI_APP1_INT_ENA Write 1 to enable SPI_APP1_INT interrupt.
(R/W)

Register 46.18. SPI_DMA_INT_CLR_REG (0x0038)



SPI_DMA_INFIFO_FULL_ERR_INT_CLR Write 1 to clear SPI_DMA_INFIFO_FULL_ERR_INT interrupt. (WT)

SPI_DMA_OUTFIFO_EMPTY_ERR_INT_CLR Write 1 to clear SPI_DMA_OUTFIFO_EMPTY_ERR_INT interrupt. (WT)

SPI_SLV_EX_QPI_INT_CLR Write 1 to clear SPI_SLV_EX_QPI_INT interrupt. (WT)

SPI_SLV_EN_QPI_INT_CLR Write 1 to clear SPI_SLV_EN_QPI_INT interrupt. (WT)

SPI_SLV_CMD7_INT_CLR Write 1 to clear SPI_SLV_CMD7_INT interrupt. (WT)

SPI_SLV_CMD8_INT_CLR Write 1 to clear SPI_SLV_CMD8_INT interrupt. (WT)

SPI_SLV_CMD9_INT_CLR Write 1 to clear SPI_SLV_CMD9_INT interrupt. (WT)

SPI_SLV_CMDA_INT_CLR Write 1 to clear SPI_SLV_CMDA_INT interrupt. (WT)

SPI_SLV_RD_DMA_DONE_INT_CLR Write 1 to clear SPI_SLV_RD_DMA_DONE_INT interrupt. (WT)

SPI_SLV_WR_DMA_DONE_INT_CLR Write 1 to clear SPI_SLV_WR_DMA_DONE_INT interrupt. (WT)

SPI_SLV_RD_BUF_DONE_INT_CLR Write 1 to clear SPI_SLV_RD_BUF_DONE_INT interrupt. (WT)

SPI_SLV_WR_BUF_DONE_INT_CLR Write 1 to clear SPI_SLV_WR_BUF_DONE_INT interrupt. (WT)

SPI_TRANS_DONE_INT_CLR Write 1 to clear SPI_TRANS_DONE_INT interrupt. (WT)

Continued on the next page...

Register 46.18. SPI_DMA_INT_CLR_REG (0x0038)

Continued from the previous page...

SPI_DMA_SEG_TRANS_DONE_INT_CLR Write 1 to clear SPI_DMA_SEG_TRANS_DONE_INT interrupt.
(WT)

SPI_SEG_MAGIC_ERR_INT_CLR Write 1 to clear SPI_SEG_MAGIC_ERR_INT interrupt.
(WT)

SPI_SLV_BUF_ADDR_ERR_INT_CLR Write 1 to clear SPI_SLV_BUF_ADDR_ERR_INT interrupt.
(WT)

SPI_SLV_CMD_ERR_INT_CLR Write 1 to clear SPI_SLV_CMD_ERR_INT interrupt.
(WT)

SPI_MST_RX_AFIFO_WFULL_ERR_INT_CLR Write 1 to clear SPI_MST_RX_AFIFO_WFULL_ERR_INT interrupt.
(WT)

SPI_MST_TX_AFIFO_REMPTY_ERR_INT_CLR Write 1 to clear SPI_MST_TX_AFIFO_REMPTY_ERR_INT interrupt.
(WT)

SPI_APP2_INT_CLR Write 1 to clear SPI_APP2_INT interrupt.
(WT)

SPI_APP1_INT_CLR Write 1 to clear SPI_APP1_INT interrupt.
(WT)

Register 46.19. SPI_DMA_INT_RAW_REG (0x003C)



SPI_DMA_INFIFO_FULL_ERR_INT_RAW The raw interrupt status of SPI_DMA_INFIFO_FULL_ERR_INT interrupt.
(R/WTC/SS)

SPI_DMA_OUTFIFO_EMPTY_ERR_INT_RAW The raw interrupt status of SPI_DMA_OUTFIFO_EMPTY_ERR_INT interrupt.
(R/WTC/SS)

SPI_SLV_EX_QPI_INT_RAW The raw interrupt status of SPI_SLV_EX_QPI_INT interrupt.
(R/WTC/SS)

SPI_SLV_EN_QPI_INT_RAW The raw interrupt status of SPI_SLV_EN_QPI_INT interrupt.
(R/WTC/SS)

SPI_SLV_CMD7_INT_RAW The raw interrupt status of SPI_SLV_CMD7_INT interrupt.
(R/WTC/SS)

SPI_SLV_CMD8_INT_RAW The raw interrupt status of SPI_SLV_CMD8_INT interrupt.
(R/WTC/SS)

SPI_SLV_CMD9_INT_RAW The raw interrupt status of SPI_SLV_CMD9_INT interrupt.
(R/WTC/SS)

SPI_SLV_CMDA_INT_RAW The raw interrupt status of SPI_SLV_CMDA_INT interrupt.
(R/WTC/SS)

SPI_SLV_RD_DMA_DONE_INT_RAW The raw interrupt status of SPI_SLV_RD_DMA_DONE_INT interrupt.
(R/WTC/SS)

SPI_SLV_WR_DMA_DONE_INT_RAW The raw interrupt status of SPI_SLV_WR_DMA_DONE_INT interrupt.
(R/WTC/SS)

SPI_SLV_RD_BUF_DONE_INT_RAW The raw interrupt status of SPI_SLV_RD_BUF_DONE_INT interrupt.
(R/WTC/SS)

Continued on the next page...

Register 46.19. SPI_DMA_INT_RAW_REG (0x003C)

Continued from the previous page...

SPI_SLV_WR_BUF_DONE_INT_RAW The raw interrupt status of SPI_SLV_WR_BUF_DONE_INT interrupt.
(R/WTC/SS)

SPI_TRANS_DONE_INT_RAW The raw interrupt status of SPI_TRANS_DONE_INT interrupt.
(R/WTC/SS)

SPI_DMA_SEG_TRANS_DONE_INT_RAW The raw interrupt status of SPI_DMA_SEG_TRANS_DONE_INT interrupt.
(R/WTC/SS)

SPI_SEG_MAGIC_ERR_INT_RAW The raw interrupt status of SPI_SEG_MAGIC_ERR_INT interrupt.
(R/WTC/SS)

SPI_SLV_BUF_ADDR_ERR_INT_RAW The raw interrupt status of SPI_SLV_BUF_ADDR_ERR_INT interrupt.
(R/WTC/SS)

SPI_SLV_CMD_ERR_INT_RAW The raw interrupt status of SPI_SLV_CMD_ERR_INT interrupt.
(R/WTC/SS)

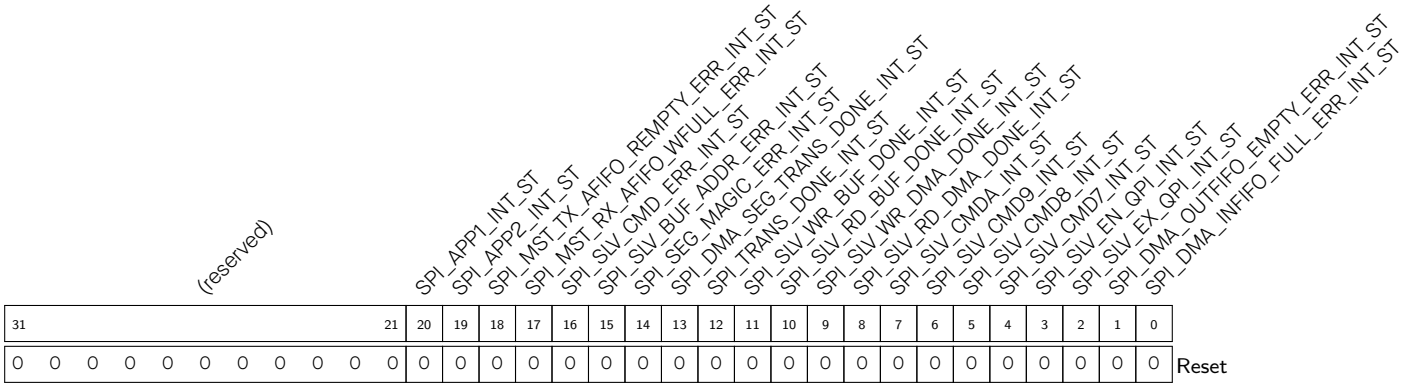
SPI_MST_RX_AFIFO_WFULL_ERR_INT_RAW The raw interrupt status of SPI_MST_RX_AFIFO_WFULL_ERR_INT interrupt.
(R/WTC/SS)

SPI_MST_TX_AFIFO_REMPTY_ERR_INT_RAW The raw interrupt status of SPI_MST_TX_AFIFO_REMPTY_ERR_INT interrupt.
(R/WTC/SS)

SPI_APP2_INT_RAW The raw interrupt status of SPI_APP2_INT interrupt. The value is only controlled by the application.
(R/WTC/SS)

SPI_APP1_INT_RAW The raw interrupt status of SPI_APP1_INT interrupt. The value is only controlled by the application.
(R/WTC/SS)

Register 46.20. SPI_DMA_INT_ST_REG (0x0040)



SPI_DMA_INFIFO_FULL_ERR_INT_ST The interrupt status of SPI_DMA_INFIFO_FULL_ERR_INT interrupt.
(RO)

SPI_DMA_OUTFIFO_EMPTY_ERR_INT_ST The interrupt status of SPI_DMA_OUTFIFO_EMPTY_ERR_INT interrupt.
(RO)

SPI_SLV_EX_QPI_INT_ST The interrupt status of SPI_SLV_EX_QPI_INT interrupt.
(RO)

SPI_SLV_EN_QPI_INT_ST The interrupt status of SPI_SLV_EN_QPI_INT interrupt.
(RO)

SPI_SLV_CMD7_INT_ST The interrupt status of SPI_SLV_CMD7_INT interrupt.
(RO)

SPI_SLV_CMD8_INT_ST The interrupt status of SPI_SLV_CMD8_INT interrupt.
(RO)

SPI_SLV_CMD9_INT_ST The interrupt status of SPI_SLV_CMD9_INT interrupt.
(RO)

SPI_SLV_CMDA_INT_ST The interrupt status of SPI_SLV_CMDA_INT interrupt.
(RO)

SPI_SLV_RD_DMA_DONE_INT_ST The interrupt status of SPI_SLV_RD_DMA_DONE_INT interrupt.
(RO)

SPI_SLV_WR_DMA_DONE_INT_ST The interrupt status of SPI_SLV_WR_DMA_DONE_INT interrupt.
(RO)

SPI_SLV_RD_BUF_DONE_INT_ST The interrupt status of SPI_SLV_RD_BUF_DONE_INT interrupt.
(RO)

SPI_SLV_WR_BUF_DONE_INT_ST The interrupt status of SPI_SLV_WR_BUF_DONE_INT interrupt.
(RO)

Continued on the next page...

Register 46.20. SPI_DMA_INT_ST_REG (0x0040)

Continued from the previous page...

SPI_TRANS_DONE_INT_ST The interrupt status of SPI_TRANS_DONE_INT interrupt.
(RO)

SPI_DMA_SEG_TRANS_DONE_INT_ST The interrupt status of SPI_DMA_SEG_TRANS_DONE_INT interrupt.
(RO)

SPI_SEG_MAGIC_ERR_INT_ST The interrupt status of SPI_SEG_MAGIC_ERR_INT interrupt.
(RO)

SPI_SLV_BUF_ADDR_ERR_INT_ST The status bit for SPI_SLV_BUF_ADDR_ERR_INT interrupt.
(RO)

SPI_SLV_CMD_ERR_INT_ST The interrupt status of SPI_SLV_CMD_ERR_INT interrupt.
(RO)

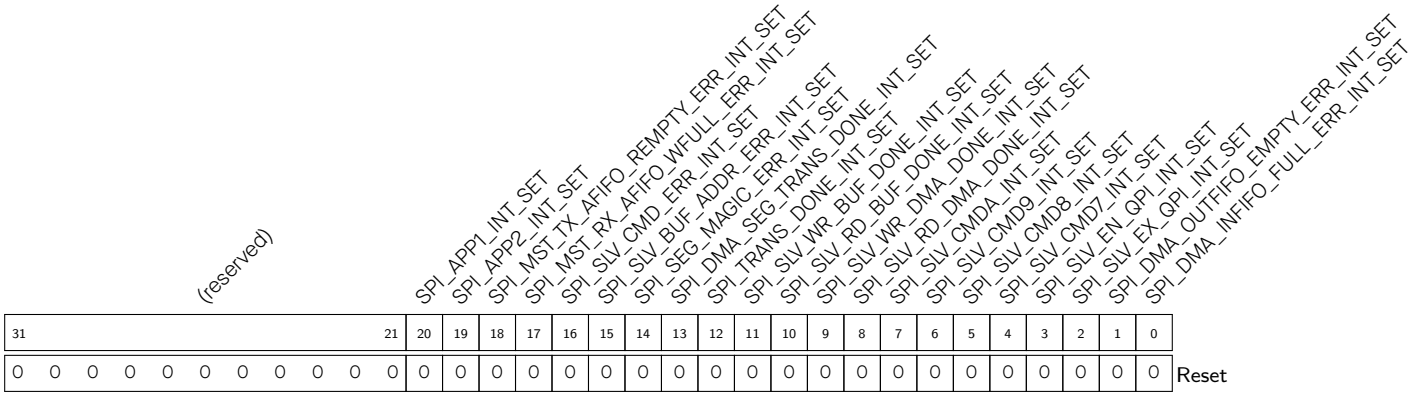
SPI_MST_RX_AFIFO_WFULL_ERR_INT_ST The interrupt status of SPI_MST_RX_AFIFO_WFULL_ERR_INT interrupt.
(RO)

SPI_MST_TX_AFIFO_REMPTY_ERR_INT_ST The interrupt status of SPI_MST_TX_AFIFO_REMPTY_ERR_INT interrupt.
(RO)

SPI_APP2_INT_ST The interrupt status of SPI_APP2_INT interrupt.
(RO)

SPI_APP1_INT_ST The interrupt status of SPI_APP1_INT interrupt.
(RO)

Register 46.21. SPI_DMA_INT_SET_REG (0x0044)



SPI_DMA_INFIFO_FULL_ERR_INT_SET Write 1 to set SPI_DMA_INFIFO_FULL_ERR_INT interrupt. (WT)

SPI_DMA_OUTFIFO_EMPTY_ERR_INT_SET Write 1 to set SPI_DMA_OUTFIFO_EMPTY_ERR_INT interrupt. (WT)

SPI_SLV_EX_QPI_INT_SET Write 1 to set SPI_SLV_EX_QPI_INT interrupt. (WT)

SPI_SLV_EN_QPI_INT_SET Write 1 to set SPI_SLV_EN_QPI_INT interrupt. (WT)

SPI_SLV_CMD7_INT_SET Write 1 to set SPI_SLV_CMD7_INT interrupt. (WT)

SPI_SLV_CMD8_INT_SET Write 1 to set SPI_SLV_CMD8_INT interrupt. (WT)

SPI_SLV_CMD9_INT_SET Write 1 to set SPI_SLV_CMD9_INT interrupt. (WT)

SPI_SLV_CMDA_INT_SET Write 1 to set SPI_SLV_CMDA_INT interrupt. (WT)

SPI_SLV_RD_DMA_DONE_INT_SET Write 1 to set SPI_SLV_RD_DMA_DONE_INT interrupt. (WT)

SPI_SLV_WR_DMA_DONE_INT_SET Write 1 to set SPI_SLV_WR_DMA_DONE_INT interrupt. (WT)

SPI_SLV_RD_BUF_DONE_INT_SET Write 1 to set SPI_SLV_RD_BUF_DONE_INT interrupt. (WT)

SPI_SLV_WR_BUF_DONE_INT_SET Write 1 to set SPI_SLV_WR_BUF_DONE_INT interrupt. (WT)

SPI_TRANS_DONE_INT_SET Write 1 to set SPI_TRANS_DONE_INT interrupt. (WT)

Continued on the next page...

Register 46.21. SPI_DMA_INT_SET_REG (0x0044)

Continued from the previous page...

SPI_DMA_SEG_TRANS_DONE_INT_SET Write 1 to set SPI_DMA_SEG_TRANS_DONE_INT interrupt. (WT)

SPI_SEG_MAGIC_ERR_INT_SET Write 1 to set SPI_SEG_MAGIC_ERR_INT interrupt. (WT)

SPI_SLV_BUF_ADDR_ERR_INT_SET Write 1 to set SPI_SLV_BUF_ADDR_ERR_INT interrupt. (WT)

SPI_SLV_CMD_ERR_INT_SET Write 1 to set SPI_SLV_CMD_ERR_INT interrupt. (WT)

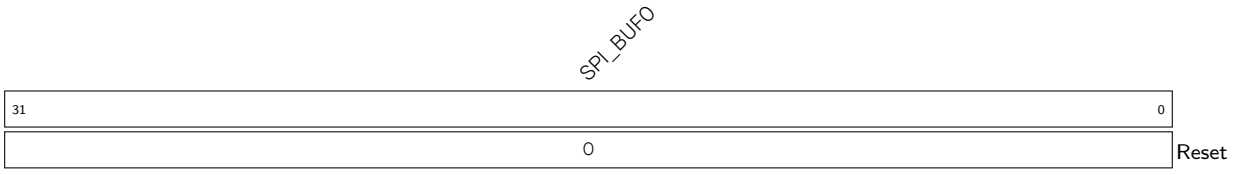
SPI_MST_RX_AFIFO_WFULL_ERR_INT_SET Write 1 to set SPI_MST_RX_AFIFO_WFULL_ERR_INT interrupt. (WT)

SPI_MST_TX_AFIFO_REMPTY_ERR_INT_SET Write 1 to set SPI_MST_TX_AFIFO_REMPTY_ERR_INT interrupt. (WT)

SPI_APP2_INT_SET Write 1 to set SPI_APP2_INT interrupt. (WT)

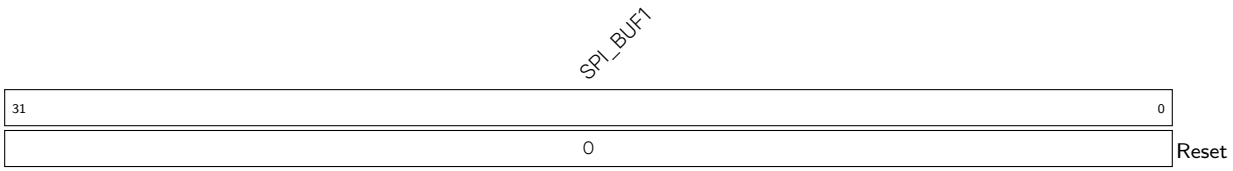
SPI_APP1_INT_SET Write 1 to set SPI_APP1_INT interrupt. (WT)

Register 46.22. SPI_W0_REG (0x0098)



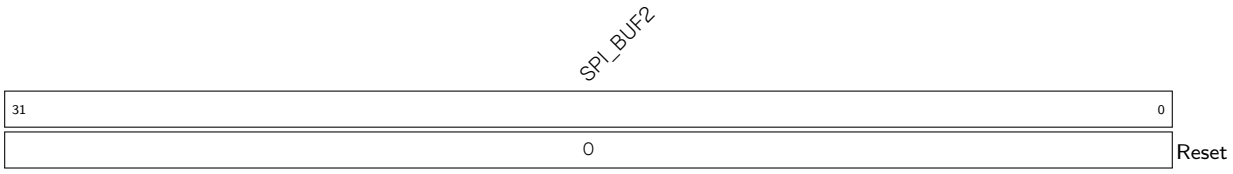
SPI_BUF0 32-bit data buffer 0. (R/W/SS)

Register 46.23. SPI_W1_REG (0x009C)



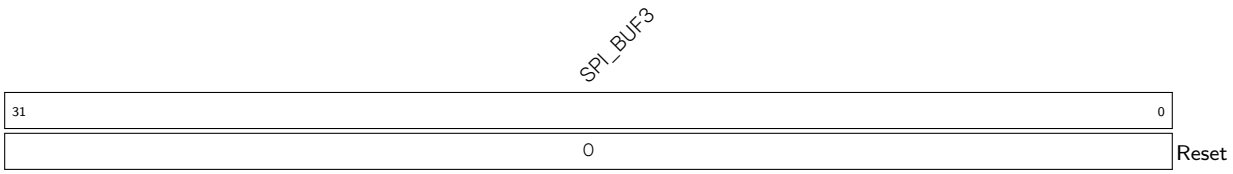
SPI_BUF1 32-bit data buffer 1.
(R/W/SS)

Register 46.24. SPI_W2_REG (0x00A0)



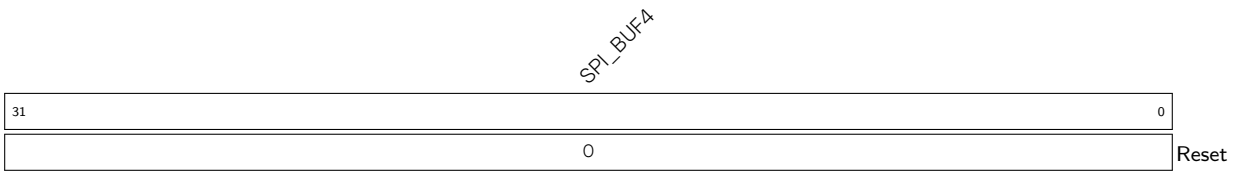
SPI_BUF2 32-bit data buffer 2.
(R/W/SS)

Register 46.25. SPI_W3_REG (0x00A4)



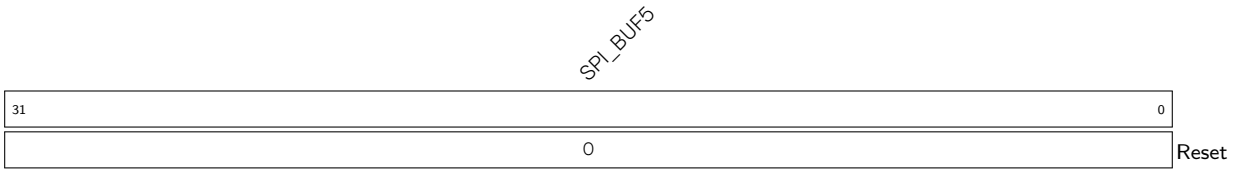
SPI_BUF3 32-bit data buffer 3.
(R/W/SS)

Register 46.26. SPI_W4_REG (0x00A8)



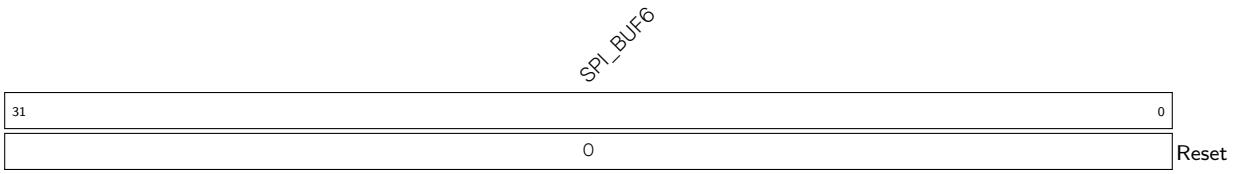
SPI_BUF4 32-bit data buffer 4.
(R/W/SS)

Register 46.27. SPI_W5_REG (0x00AC)



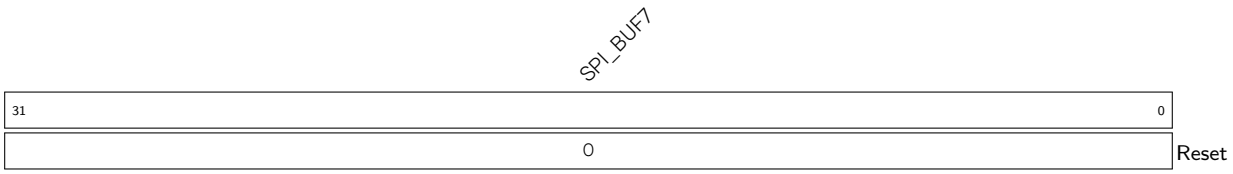
SPI_BUF5 32-bit data buffer 5.
(R/W/SS)

Register 46.28. SPI_W6_REG (0x00B0)



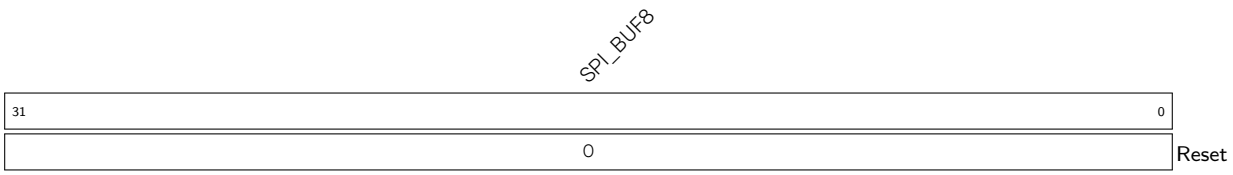
SPI_BUF6 32-bit data buffer 6.
(R/W/SS)

Register 46.29. SPI_W7_REG (0x00B4)



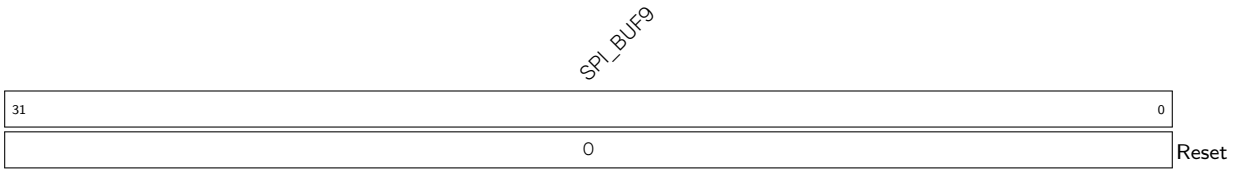
SPI_BUF7 32-bit data buffer 7.
(R/W/SS)

Register 46.30. SPI_W8_REG (0x00B8)



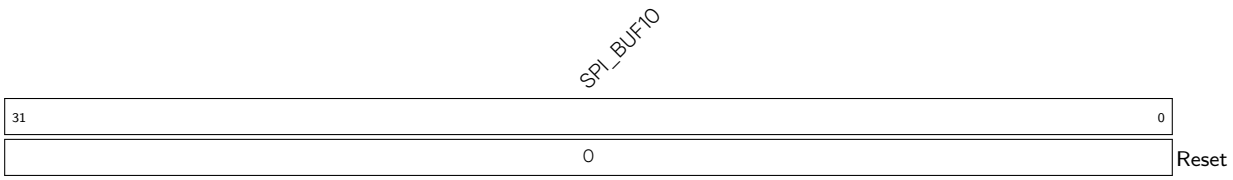
SPI_BUF8 32-bit data buffer 8.
(R/W/SS)

Register 46.31. SPI_W9_REG (0x00BC)



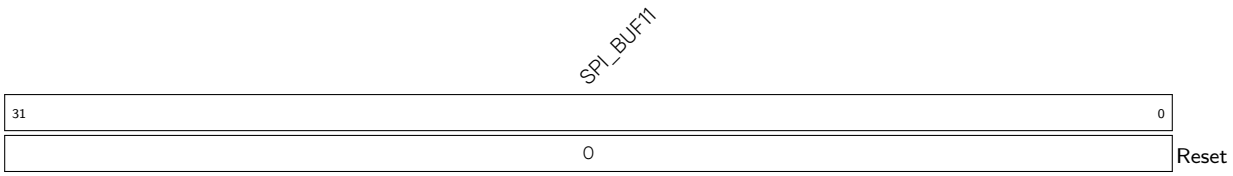
SPI_BUF9 32-bit data buffer 9.
(R/W/SS)

Register 46.32. SPI_W10_REG (0x00C0)



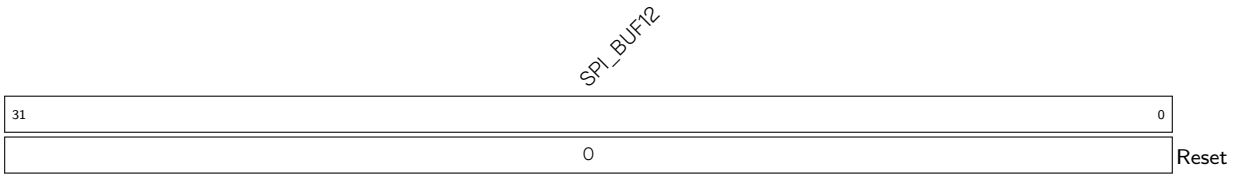
SPI_BUF10 32-bit data buffer 10.
(R/W/SS)

Register 46.33. SPI_W11_REG (0x00C4)



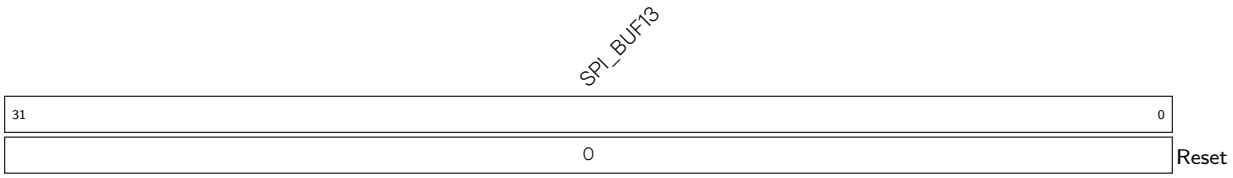
SPI_BUF11 32-bit data buffer 11.
(R/W/SS)

Register 46.34. SPI_W12_REG (0x00C8)



SPI_BUF12 32-bit data buffer 12.
(R/W/SS)

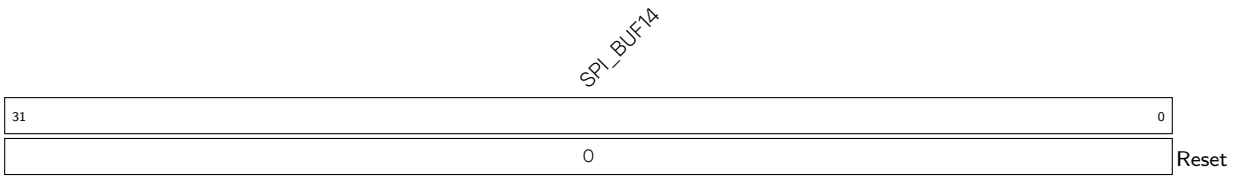
Register 46.35. SPI_W13_REG (0x00CC)



SPI_BUF13 32-bit data buffer 13.

(R/W/SS)

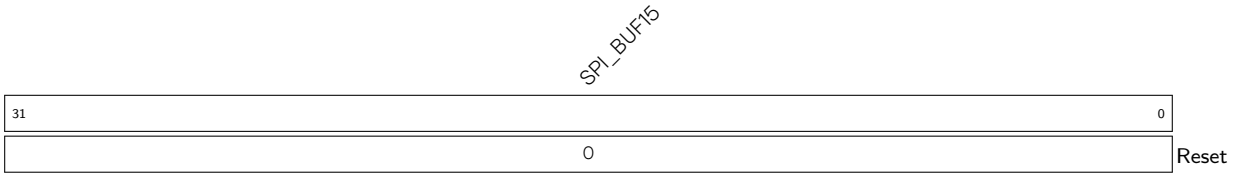
Register 46.36. SPI_W14_REG (0x00D0)



SPI_BUF14 32-bit data buffer 14.

(R/W/SS)

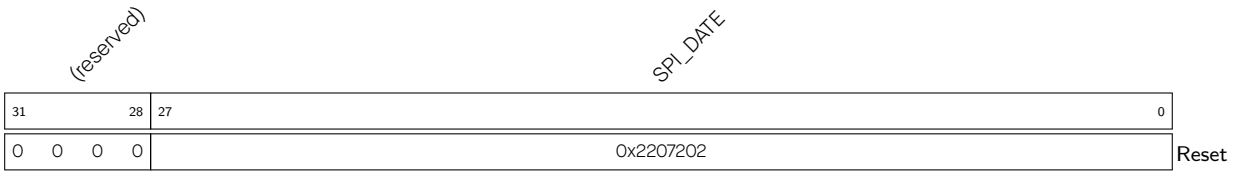
Register 46.37. SPI_W15_REG (0x00D4)



SPI_BUF15 32-bit data buffer 15.

(R/W/SS)

Register 46.38. SPI_DATE_REG (0x00F0)



SPI_DATE Version control register.

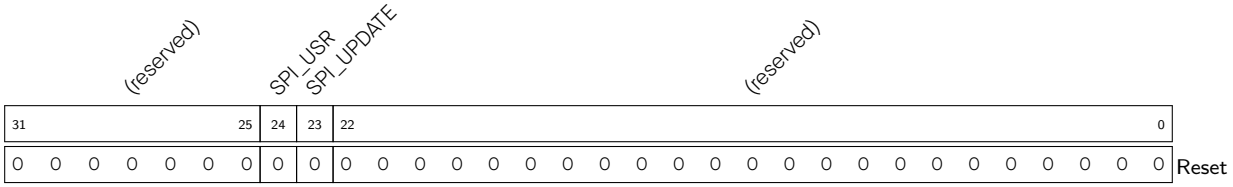
(R/W)

46.13.2 GP-SPI3 Register

The addresses in this section are relative to GP-SPI3 base address provided in Table 9.3-2 in Chapter 9 *System and Memory*.

For how to program reserved fields, please refer to Section *Programming Reserved Register Field*.

Register 46.39. SPI_CMD_REG (0x0000)



SPI_UPDATE Configures whether or not to synchronize SPI registers from APB clock domain into SPI module clock domain.

0: Not synchronize

1: Synchronize

This bit is only used in SPI master transfer.

(WT)

SPI_USR Configures whether or not to enable user-defined command.

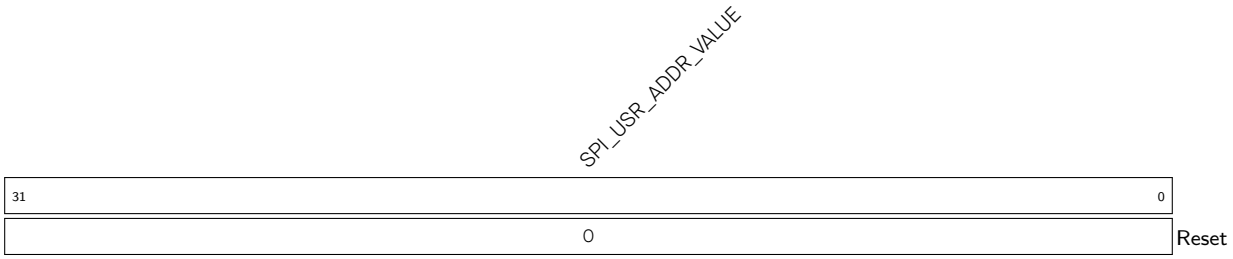
0: Not enable

1: Enable

An SPI operation will be triggered when the bit is set. This bit will be cleared once the operation is done.

(R/W/SC)

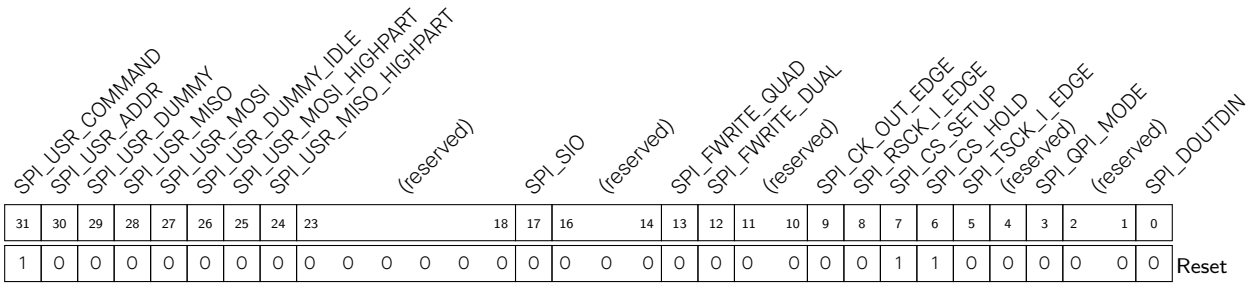
Register 46.40. SPI_ADDR_REG (0x0004)



SPI_USR_ADDR_VALUE Configures the address to slave.

(R/W)

Register 46.41. SPI_USER_REG (0x0010)



SPI_DOUTDIN Configures whether or not to enable full-duplex communication.

0: Disable

1: Enable

(R/W)

SPI_QPI_MODE Configures whether or not to enable QPI mode.

0: Disable

1: Enable

This configuration is applicable when the SPI controller works as master or slave.

(R/W/SS/SC)

SPI_TSCK_I_EDGE Configures whether or not to change the polarity of TSCK in slave transfer.

0: TSCK = SPI_CK_I

1: TSCK = !SPI_CK_I

(R/W)

SPI_CS_HOLD Configures whether or not to keep SPI CS low when SPI is in DONE state.

0: Not keep low

1: Keep low

(R/W)

SPI_CS_SETUP Configures whether or not to enable SPI CS when SPI is in prepare (PREP) state.

0: Disable

1: Enable

(R/W)

SPI_RSCK_I_EDGE Configures whether or not to change the polarity of RSCK in slave transfer.

0: RSCK = !SPI_CK_I

1: RSCK = SPI_CK_I

(R/W)

SPI_CK_OUT_EDGE Configures SPI clock mode together with SPI_CK_IDLE_EDGE. For more information, see Section 46.7.4.

(R/W)

Continued on the next page...

Register 46.41. SPI_USER_REG (0x0010)

Continued from the previous page...

SPI_FWRITE_DUAL Configures whether or not to enable the 2-bit mode of read-data phase in write operations.
0: Not enable
1: Enable
(R/W)

SPI_FWRITE_QUAD Configures whether or not to enable the 4-bit mode of read-data phase in write operations.
0: Not enable
1: Enable
(R/W)

SPI_SIO Configures whether or not to enable 3-line half-duplex communication, where MOSI and MISO signals share the same pin.
0: Disable
1: Enable
(R/W)

SPI_USR_MISO_HIGHPART Configures whether or not to enable High-Part mode in read-data phase, i.e., only access to high-part of the buffers: SPI_W8_REG ~ SPI_W15_REG.
0: Disable
1: Enable
Can be configured in CONF state.
(R/W)

SPI_USR_MOSI_HIGHPART Configures whether or not to enable High-Part mode in write-data phase, i.e., only access to high-part of the buffers: SPI_W8_REG ~ SPI_W15_REG.
0: Disable
1: Enable
Can be configured in CONF state.
(R/W)

SPI_USR_DUMMY_IDLE Configures whether or not to disable SPI clock in DUMMY state.
0: Not disable
1: Disable
(R/W)

SPI_USR_MOSI Configures whether or not to enable the write-data (DOUT) state of an operation.
0: Disable
1: Enable
(R/W)

Continued on the next page...

Register 46.41. SPI_USER_REG (0x0010)

Continued from the previous page...

SPI_USR_MISO Configures whether or not to enable the read-data (DIN) state of an operation.

0: Disable

1: Enable

(R/W)

SPI_USR_DUMMY Configures whether or not to enable the DUMMY state of an operation.

0: Disable

1: Enable

(R/W)

SPI_USR_ADDR Configures whether or not to enable the address (ADDR) state of an operation.

0: Disable

1: Enable

(R/W)

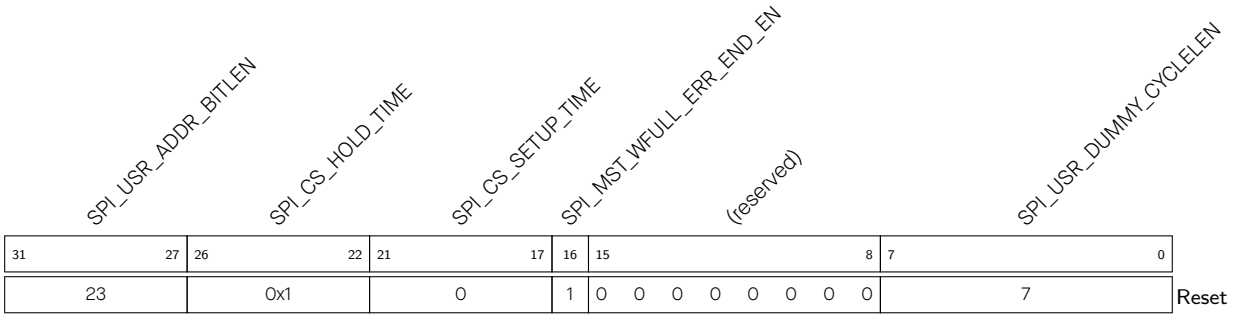
SPI_USR_COMMAND Configures whether or not to enable the command (CMD) state of an operation.

0: Disable

1: Enable

(R/W)

Register 46.42. SPI_USER1_REG (0x0014)



SPI_USR_DUMMY_CYCLELEN Configures the length of DUMMY state.

Measurement unit: SPI_CLK clock cycles.

This value is (the expected cycle number - 1).

(R/W)

SPI_MST_WFULL_ERR_END_EN Configures whether or not to end the SPI transfer when SPI RX AFIFO wfull error occurs in master full-/half-duplex transfers.

0: Not end

1: End

(R/W)

SPI_CS_SETUP_TIME Configures the length of prepare (PREP) state.

Measurement unit: SPI_CLK clock cycles.

This value is equal to the expected cycles - 1. This field is used together with SPI_CS_SETUP.

(R/W)

SPI_CS_HOLD_TIME Configures the delay cycles of CS pin.

Measurement unit: SPI_CLK clock cycles.

This field is used together with SPI_CS_HOLD.

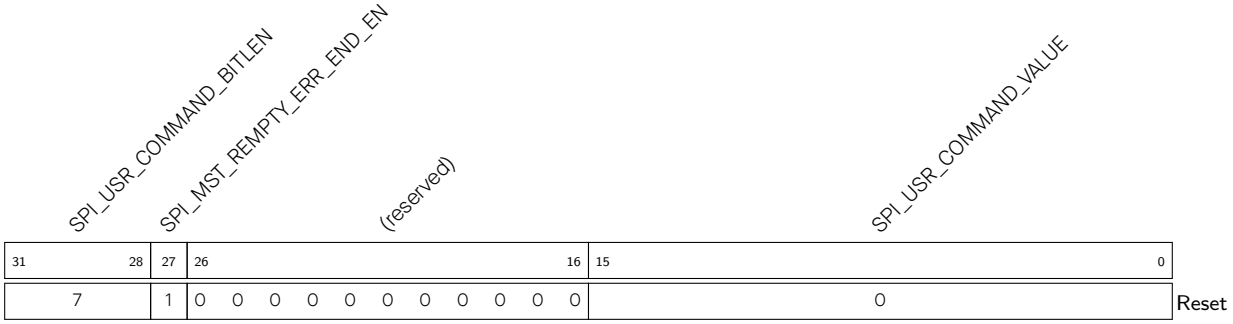
(R/W)

SPI_USR_ADDR_BITLEN Configures the bit length in address state.

This value is (expected bit number - 1).

(R/W)

Register 46.43. SPI_USER2_REG (0x0018)



SPI_USR_COMMAND_VALUE Configures the command value.

(R/W)

SPI_MST_REMPTY_ERR_END_EN Configures whether or not to end the SPI transfer when SPI TX AFIFO read empty error occurs in master full-/half-duplex transfers.

0: Not end

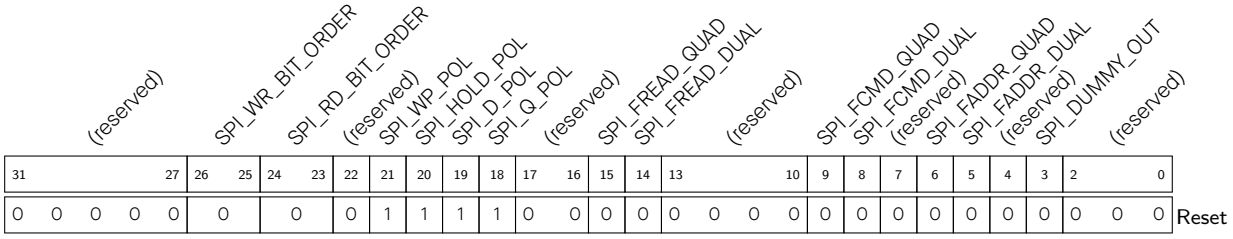
1: End

(R/W)

SPI_USR_COMMAND_BITLEN Configures the bit length of command state. This value is (expected bit number - 1).

(R/W)

Register 46.44. SPI_CTRL_REG (0x0008)



SPI_DUMMY_OUT Configures whether or not to output the SPI3 bus signals in DUMMY state.

0: Not output

1: Output

(R/W)

SPI_FADDR_DUAL Configures whether or not to enable 2-bit mode during address (ADDR) state.

0: Disable

1: Enable

(R/W)

SPI_FADDR_QUAD Configures whether or not to enable 4-bit mode during address (ADDR) state.

0: Disable

1: Enable

(R/W)

SPI_FCMD_DUAL Configures whether or not to enable 2-bit mode during command (CMD) state.

0: Disable

1: Enable

(R/W)

SPI_FCMD_QUAD Configures whether or not to enable 4-bit mode during command (CMD) state.

0: Disable

1: Enable

(R/W)

SPI_FREAD_DUAL Configures whether or not to enable the 2-bit mode of read-data (DIN) state in read operations.

0: Disable

1: Enable

(R/W)

SPI_FREAD_QUAD Configures whether or not to enable the 4-bit mode of read-data (DIN) state in read operations.

0: Disable

1: Enable

(R/W)

Continued on the next page...

Register 46.44. SPI_CTRL_REG (0x0008)

Continued from the previous page...

SPI_Q_POL Configures MISO line polarity.

0: Low

1: High

(R/W)

SPI_D_POL Configures MOSI line polarity.

0: Low

1: High

(R/W)

SPI_HOLD_POL Configures SPI_HOLD output value when SPI is in idle.

0: Output low

1: Output high

(R/W)

SPI_WP_POL Configures the output value of write-protect signal when SPI is in idle.

0: Output low

1: Output high

(R/W)

SPI_RD_BIT_ORDER Configures the bit order in read-data (MISO) state.

0: MSB first

1: LSB first

2: MSB first

3: LSB first

(R/W)

SPI_WR_BIT_ORDER Configures the bit order in command (CMD), address (ADDR), and write-data (MOSI) states.

0: MSB first

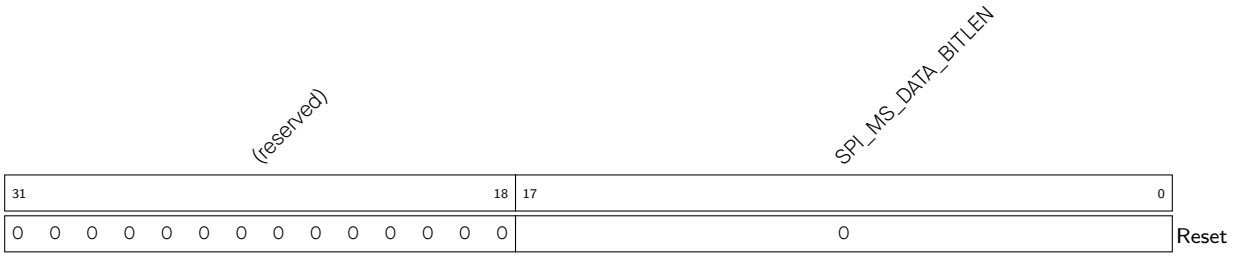
1: LSB first

2: MSB first

3: LSB first

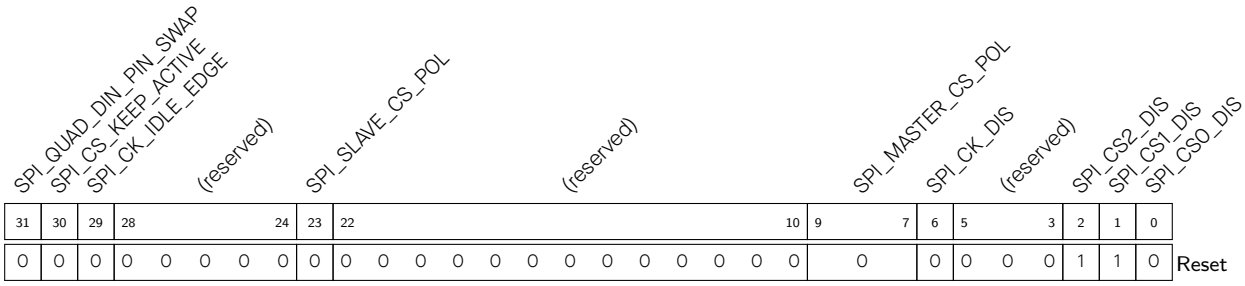
(R/W)

Register 46.45. SPI_MS_DLEN_REG (0x001C)



SPI_MS_DATA_BITLEN Configures the data bit length of SPI transfer in CPU-controlled master transfer. Or configures the bit length of SPI RX transfer in DMA-controlled slave transfer.
This value shall be (expected bit_num - 1).
(R/W)

Register 46.46. SPI_MISC_REG (0x0020)


SPI_QUAD_DIN_PIN_SWAP
SPI_CS_KEEP_ACTIVE
SPI_CK_IDLE_EDGE
(reserved)
SPI_SLAVE_CS_POL
(reserved)
SPI_MASTER_CS_POL
SPI_CK_DIS
(reserved)
SPI_CS2_DIS
SPI_CS1_DIS
SPI_CS0_DIS
31 30 29 28 24 23 22 10 9 7 6 5 3 2 1 0
0 0 0

Register 46.46. SPI_MISC_REG (0x0020)

Continued from the previous page...

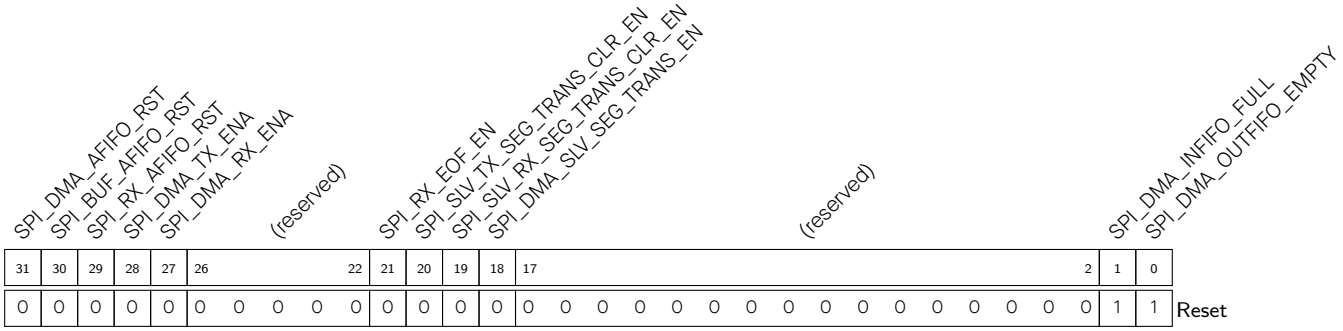
SPI_QUAD_DIN_PIN_SWAP Configures whether or not to swap SPI Quad input pins.

0: Not swap

1: Swap SPI3_D with SPI3_Q, and SPI3_WP with SPI3_HD

(R/W)

Register 46.47. SPI_DMA_CONF_REG (0x0030)



SPI_DMA_OUTFIFO_EMPTY Represents whether or not the DMA TX FIFO is ready for sending data.

0: Ready

1: Not ready

(RO)

SPI_DMA_INFIFO_FULL Represents whether or not the DMA RX FIFO is ready for receiving data.

0: Ready

1: Not ready

(RO)

SPI_DMA_SLV_SEG_TRANS_EN Configures whether or not to enable DMA-controlled segmented transfer in slave half-duplex communication.

0: Disable

1: Enable

(R/W)

SPI_SLV_RX_SEG_TRANS_CLR_EN In slave segmented transfer, if the size of the DMA RX buffer is smaller than the size of the received data,

1: the data in all the following Wr_DMA transactions will not be received

0: the data in this Wr_DMA transaction will not be received, but in the following transactions,

- if the size of DMA RX buffer is not 0, the data in following Wr_DMA transactions will be received.

- if the size of DMA RX buffer is 0, the data in following Wr_DMA transactions will not be received.

(R/W)

SPI_SLV_TX_SEG_TRANS_CLR_EN In slave segmented transfer, if the size of the DMA TX buffer is smaller than the size of the transmitted data,

1: the data in the following transactions will not be updated, i.e. the old data is transmitted repeatedly.

0: the data in this transaction will not be updated. But in the following transactions,

- if new data is filled in DMA TX FIFO, new data will be transmitted. - if no new data is filled in DMA TX FIFO, no new data will be transmitted.

(R/W)

Continued on the next page...

Register 46.47. SPI_DMA_CONF_REG (0x0030)

Continued from the previous page...

SPI_RX_EOF_EN Configures the trigger source of GDMA_IN_SUC_EOF_CH*n*_INT_RAW.

0: GDMA_IN_SUC_EOF_CH*n*_INT_RAW is set by SPI_TRANS_DONE_INT event in a single transfer, or by an SPI_DMA_SEG_TRANS_DONE_INT event in a segmented transfer.

1: In a DAM-controlled transfer, if the bit number of transferred data is equal to (SPI_MS_DATA_BITLEN + 1), then GDMA_IN_SUC_EOF_CH*n*_INT_RAW will be set by hardware. (R/W)

SPI_DMA_RX_ENA Configures whether or not to enable DMA-controlled receive data transfer.

0: Disable

1: Enable

(R/W)

SPI_DMA_TX_ENA Configures whether or not to enable DMA-controlled send data transfer.

0: Disable

1: Enable

(R/W)

SPI_RX_AFIFO_RST Configures whether or not to reset RX AFIFO (i.e., spi_rx_afifo in Figure 46.5-3 and Figure 46.5-4), which is used to receive data in SPI master and slave transfer.

0: Not reset

1: Reset

(WT)

SPI_BUF_AFIFO_RST Configures whether or not to reset BUF TX AFIFO (i.e., buf_tx_afifo in Figure 46.5-3 and Figure 46.5-4), which is used to send data out in SPI slave CPU-controlled transfer and master transfer.

0: Not reset

1: Reset

(WT)

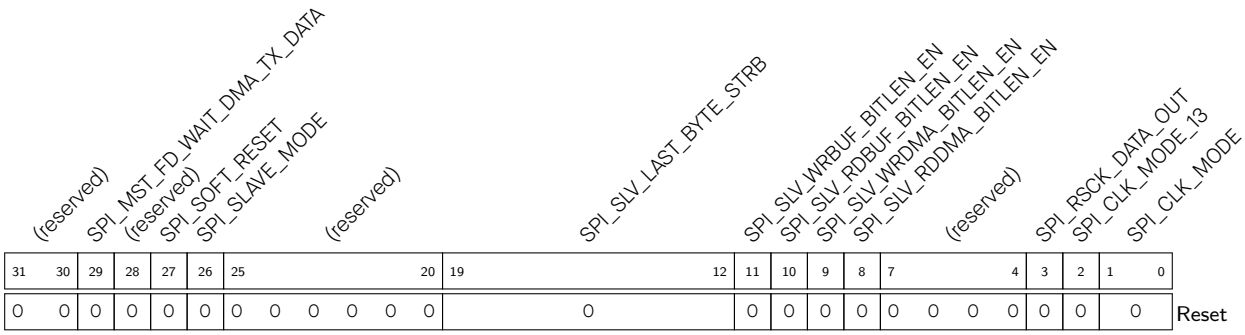
SPI_DMA_AFIFO_RST Configures whether or not to reset DMA TX AFIFO (i.e., dma_tx_afifo in Figure 46.5-3 and Figure 46.5-4), which is used to send data out in SPI slave DMA-controlled transfer.

0: Not reset

1: Reset

(WT)

Register 46.48. SPI_SLAVE_REG (0x00E0)



SPI_CLK_MODE Configures SPI clock mode.

0: SPI clock is off when CS becomes inactive.

1: SPI clock is delayed one cycle after CS becomes inactive.

2: SPI clock is delayed two cycles after CS becomes inactive.

3: SPI clock is always on.

(R/W)

SPI_CLK_MODE_13 Configures clock mode.

0: SPI clock mode 1 and mode 3.See Table 46.7-2.

1: SPI clock mode 0 and mode 2. See Table 46.7-2.

(R/W)

SPI_RSCK_DATA_OUT Configures the edge of output data.

0: Output data at TSCK rising edge.

1: Output data at RSCK rising edge.

(R/W)

SPI_SLV_RDDMA_BITLEN_EN Configures whether or not to use SPI_SLV_DATA_BITLEN to store the data bit length of Rd_DMA transfer.

0: Not use

1: Use

(R/W)

SPI_SLV_WRDMA_BITLEN_EN Configures whether or not to use SPI_SLV_DATA_BITLEN to store the data bit length of Wr_DMA transfer.

0: Not use

1: Use

(R/W)

SPI_SLV_RDBUF_BITLEN_EN Configures whether or not to use SPI_SLV_DATA_BITLEN to store the data bit length of Rd_BUF transfer.

0: Not use

1: Use

(R/W)

Continued on the next page...

Register 46.48. SPI_SLAVE_REG (0x00E0)

Continued from the previous page...

SPI_SLV_WRBUF_BITLEN_EN Configures whether or not to use SPI_SLV_DATA_BITLEN to store the data bit length of Wr_BUF transfer.

0: Not use

1: Use

(R/W)

SPI_SLV_LAST_BYTE_STRB Represents the valid bits of the last received data byte in SPI slave full-duplex and half-duplex transfer.

(R/SS)

SPI_SLAVE_MODE Configures SPI work mode.

0: Master

1: Slave

(R/W)

SPI_SOFT_RESET Configures whether to reset the SPI clock line, CS line, and data line via software.

0: Not reset

1: Reset

(WT)

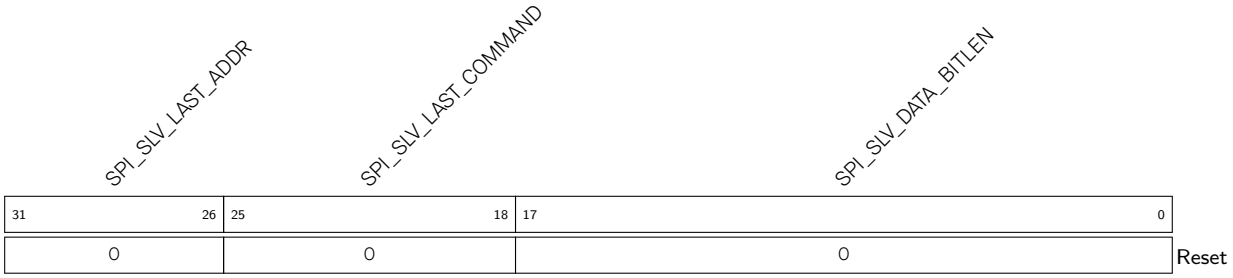
SPI_MST_FD_WAIT_DMA_TX_DATA Configures whether or not to wait DMA TX data gets ready before starting SPI transfer in master full-duplex transfer.

0: Not wait

1: Wait

(R/W)

Register 46.49. SPI_SLAVE1_REG (0x00E4)

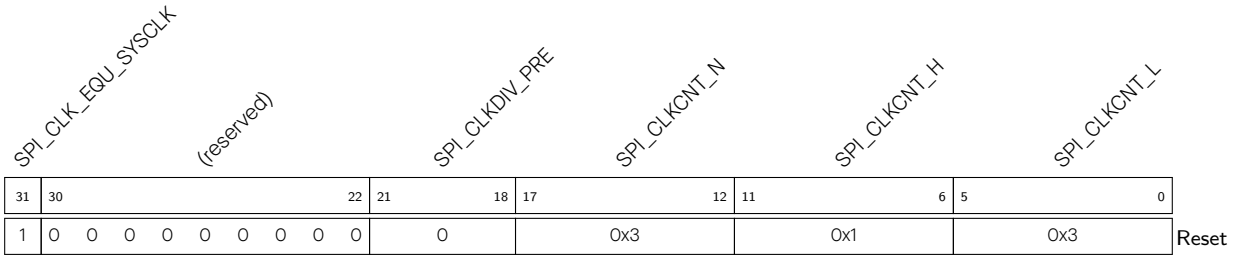


SPI_SLV_DATA_BITLEN Configures the transferred data bit length in SPI slave full-/half-duplex modes.
(R/W/SS)

SPI_SLV_LAST_COMMAND Configures the command value in slave mode.
(R/W/SS)

SPI_SLV_LAST_ADDR Configures the address value in slave mode.
(R/W/SS)

Register 46.50. SPI_CLOCK_REG (0x000C)



SPI_CLKCNT_L Configures clock duty cycles, together with SPI_CLKCNT_H and SPI_CLKCNT_N.

In master transfer, this field must be equal to SPI_CLKCNT_N.

In slave mode, it must be 0.

Can be configured in CONF state.

(R/W)

SPI_CLKCNT_H Configures the duty cycle of SPI_CLK (high level) in master transfer.

It's recommended to configure this value to floor((SPI_CLKCNT_N + 1)/2 - 1). floor() here is to round a number down, e.g., floor(2.2) = 2.

In slave mode, it must be 0.

(R/W)

SPI_CLKCNT_N Configures the divider of SPI_CLK in master transfer.

SPI_CLK frequency is $f_{clk_spi_mst}$/(SPI_CLKDIV_PRE + 1)/(SPI_CLKCNT_N + 1).

(R/W)

SPI_CLKDIV_PRE Configures the pre-divider of SPI_CLK in master transfer.

(R/W)

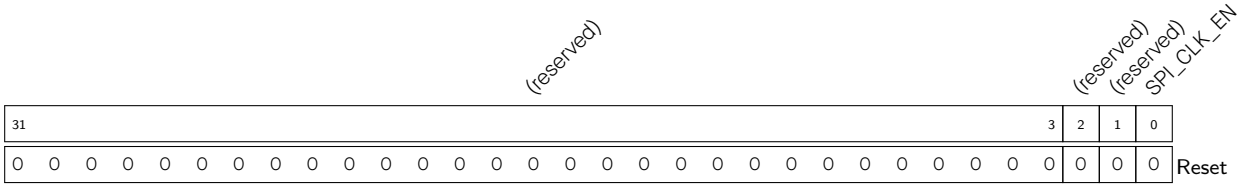
SPI_CLK_EQU_SYSCLK Configures whether or not the SPI_CLK is equal to clk_spi_mst in master transfer.

0: SPI_CLK is divided from clk_spi_mst.

1: SPI_CLK is equal to clk_spi_mst.

(R/W)

Register 46.51. SPI_CLK_GATE_REG (0x00E8)



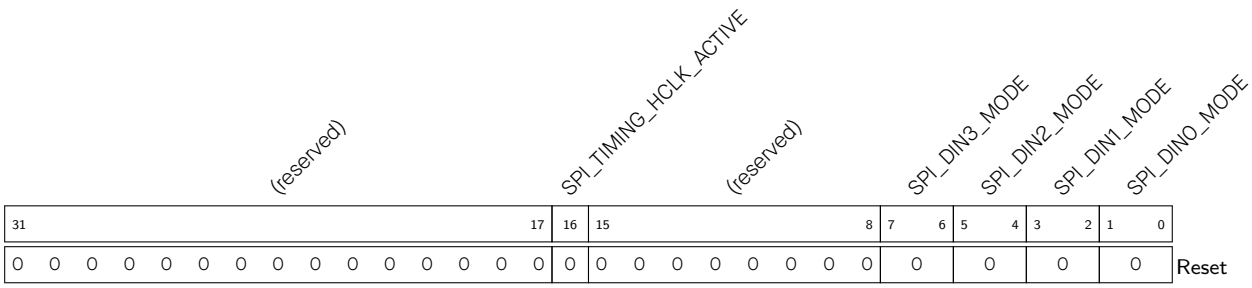
SPI_CLK_EN Configures whether or not to enable clock gate.

0: Disable

1: Enable

(R/W)

Register 46.52. SPI_DIN_MODE_REG (0x0024)



SPI_DIN0_MODE Configures the input mode for SPI3_D signal.

0: Input without delay

1: Input at the (SPI_DIN0_NUM + 1)th falling edge of clk_spi_mst

2: Input at the (SPI_DIN0_NUM + 1)th rising edge of clk_hclk plus one clk_spi_mst rising edge cycle

3: Input at the (SPI_DIN0_NUM + 1)th rising edge of clk_hclk plus one clk_spi_mst falling edge cycle

(R/W)

SPI_DIN1_MODE Configures the input mode for SPI3_Q signal.

0: Input without delay

1: Input at the (SPI_DIN1_NUM+1)th falling edge of clk_spi_mst

2: Input at the (SPI_DIN1_NUM + 1)th rising edge of clk_hclk plus one clk_spi_mst rising edge cycle

3: Input at the (SPI_DIN1_NUM + 1)th rising edge of clk_hclk plus one clk_spi_mst falling edge cycle

(R/W)

SPI_DIN2_MODE Configures the input mode for SPI3_WP signal.

0: Input without delay

1: Input at the (SPI_DIN2_NUM + 1)th falling edge of clk_spi_mst

2: Input at the (SPI_DIN2_NUM + 1)th rising edge of clk_hclk plus one clk_spi_mst rising edge cycle

3: Input at the (SPI_DIN2_NUM + 1)th rising edge of clk_hclk plus one clk_spi_mst falling edge cycle

(R/W)

SPI_DIN3_MODE Configures the input mode for SPI3_HD signal.

0: Input without delay

1: Input at the (SPI_DIN3_NUM + 1)th falling edge of clk_spi_mst

2: Input at the (SPI_DIN3_NUM + 1)th rising edge of clk_hclk plus one clk_spi_mst rising edge cycle

3: Input at the (SPI_DIN3_NUM + 1)th rising edge of clk_hclk plus one clk_spi_mst falling edge cycle

(R/W)

Continued on the next page...

Register 46.52. SPI_DIN_MODE_REG (0x0024)

Continued from the previous page...

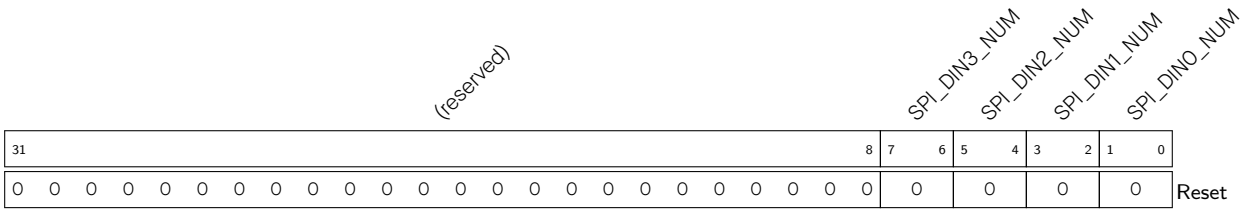
SPI_TIMING_HCLK_ACTIVE Configures whether or not to enable HCLK (high-frequency clock) in SPI input timing module.

0: Disable

1: Enable

(R/W)

Register 46.53. SPI_DIN_NUM_REG (0x0028)



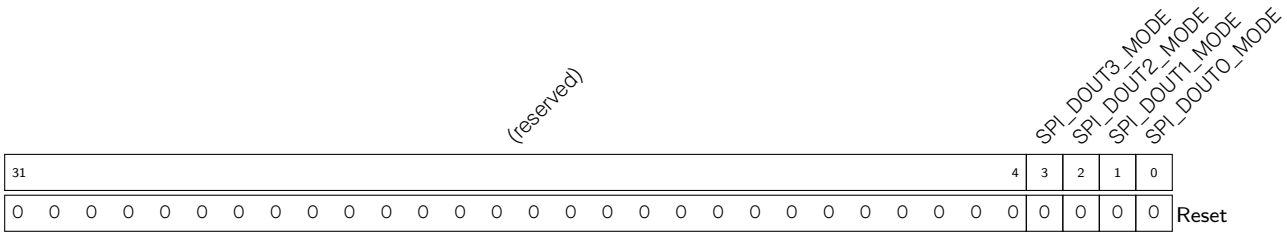
SPI_DIN0_NUM Configures the delays to input signal SPI3_D based on the setting of SPI_DIN0_MODE.
0: Delayed by 1 clock cycle
1: Delayed by 2 clock cycles
2: Delayed by 3 clock cycles
3: Delayed by 4 clock cycles
(R/W)

SPI_DIN1_NUM Configures the delays to input signal SPI3_Q based on the setting of SPI_DIN1_MODE.
0: Delayed by 1 clock cycle
1: Delayed by 2 clock cycles
2: Delayed by 3 clock cycles
3: Delayed by 4 clock cycles
(R/W)

SPI_DIN2_NUM Configures the delays to input signal SPI3_WP based on the setting of SPI_DIN2_MODE.
0: Delayed by 1 clock cycle
1: Delayed by 2 clock cycles
2: Delayed by 3 clock cycles
3: Delayed by 4 clock cycles
(R/W)

SPI_DIN3_NUM Configures the delays to input signal SPI3_HD based on the setting of SPI_DIN3_MODE.
0: Delayed by 1 clock cycle
1: Delayed by 2 clock cycles
2: Delayed by 3 clock cycles
3: Delayed by 4 clock cycles
(R/W)

Register 46.54. SPI_DOUT_MODE_REG (0x002C)



SPI_DOUT0_MODE Configures the output mode for SPI3_D signal.

0: Output without delay

1: Output with a delay of a SPI module clock cycle at its falling edge

(R/W)

SPI_DOUT1_MODE Configures the output mode for SPI3_Q signal.

0: Output without delay

1: Output with a delay of a SPI module clock cycle at its falling edge

(R/W)

SPI_DOUT2_MODE Configures the output mode for SPI3_WP signal.

0: Output without delay

1: Output with a delay of a SPI module clock cycle at its falling edge

(R/W)

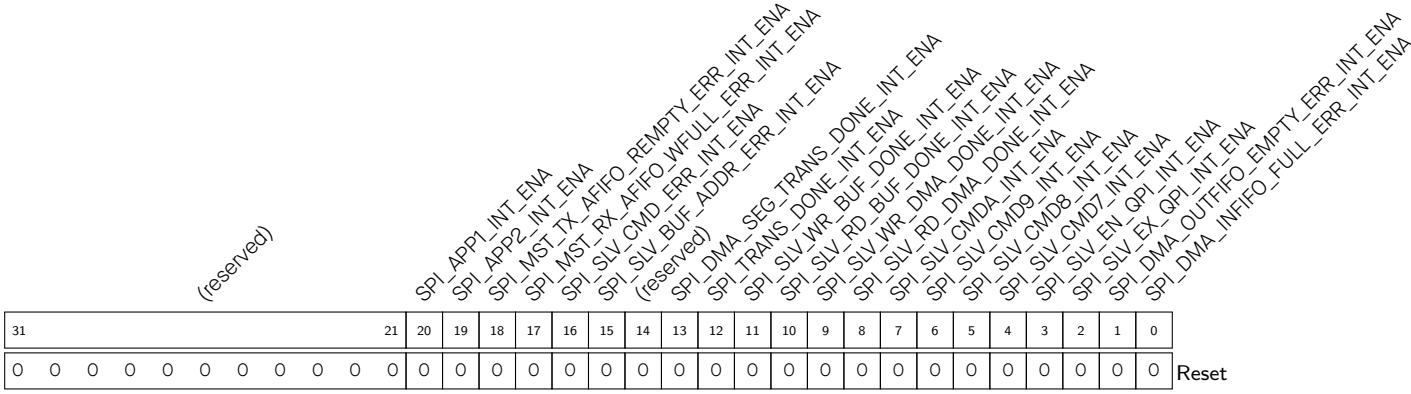
SPI_DOUT3_MODE Configures the output mode for SPI3_HD signal.

0: Output without delay

1: Output with a delay of a SPI module clock cycle at its falling edge

(R/W)

Register 46.55. SPI_DMA_INT_ENA_REG (0x0034)



SPI_DMA_INFIFO_FULL_ERR_INT_ENA Write 1 to enable SPI_DMA_INFIFO_FULL_ERR_INT interrupt.
(R/W)

SPI_DMA_OUTFIFO_EMPTY_ERR_INT_ENA Write 1 to enable SPI_DMA_OUTFIFO_EMPTY_ERR_INT interrupt.
(R/W)

SPI_SLV_EX_QPI_INT_ENA Write 1 to enable SPI_SLV_EX_QPI_INT interrupt.
(R/W)

SPI_SLV_EN_QPI_INT_ENA Write 1 to enable SPI_SLV_EN_QPI_INT interrupt.
(R/W)

SPI_SLV_CMD7_INT_ENA Write 1 to enable SPI_SLV_CMD7_INT interrupt.
(R/W)

SPI_SLV_CMD8_INT_ENA Write 1 to enable SPI_SLV_CMD8_INT interrupt.
(R/W)

SPI_SLV_CMD9_INT_ENA Write 1 to enable SPI_SLV_CMD9_INT interrupt.
(R/W)

SPI_SLV_CMDA_INT_ENA Write 1 to enable SPI_SLV_CMDA_INT interrupt.
(R/W)

SPI_SLV_RD_DMA_DONE_INT_ENA Write 1 to enable SPI_SLV_RD_DMA_DONE_INT interrupt.
(R/W)

SPI_SLV_WR_DMA_DONE_INT_ENA Write 1 to enable SPI_SLV_WR_DMA_DONE_INT interrupt.
(R/W)

SPI_SLV_RD_BUF_DONE_INT_ENA Write 1 to enable SPI_SLV_RD_BUF_DONE_INT interrupt.
(R/W)

SPI_SLV_WR_BUF_DONE_INT_ENA Write 1 to enable SPI_SLV_WR_BUF_DONE_INT interrupt.
(R/W)

Continued on the next page...

Register 46.55. SPI_DMA_INT_ENA_REG (0x0034)

Continued from the previous page...

SPI_TRANS_DONE_INT_ENA Write 1 to enable SPI_TRANS_DONE_INT interrupt. (R/W)

SPI_DMA_SEG_TRANS_DONE_INT_ENA Write 1 to enable SPI_DMA_SEG_TRANS_DONE_INT interrupt. (R/W)

SPI_SLV_BUF_ADDR_ERR_INT_ENA Write 1 to enable SPI_SLV_BUF_ADDR_ERR_INT interrupt. (R/W)

SPI_SLV_CMD_ERR_INT_ENA Write 1 to enable SPI_SLV_CMD_ERR_INT interrupt. (R/W)

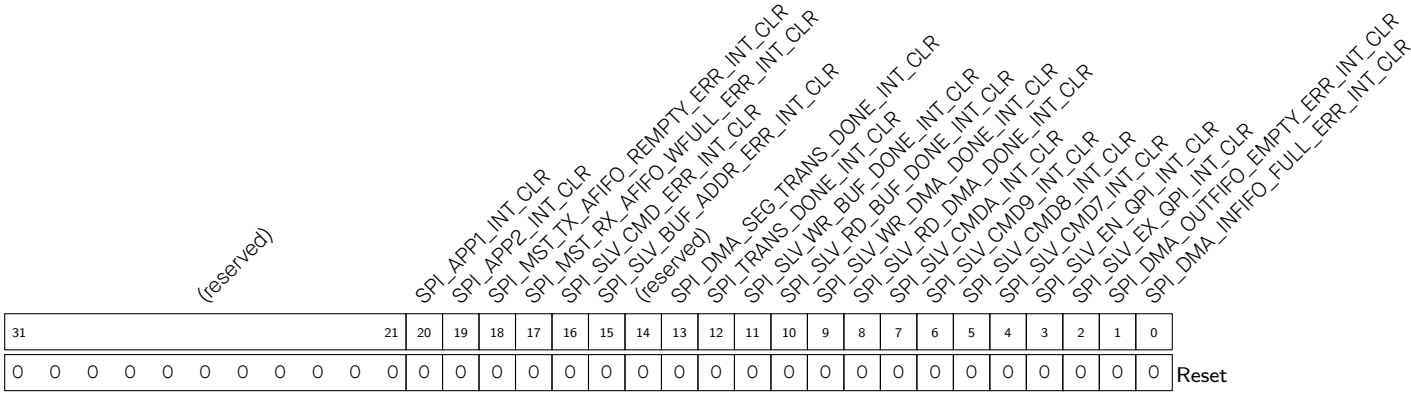
SPI_MST_RX_AFIFO_WFULL_ERR_INT_ENA Write 1 to enable SPI_MST_RX_AFIFO_WFULL_ERR_INT interrupt. (R/W)

SPI_MST_TX_AFIFO_REMPTY_ERR_INT_ENA Write 1 to enable SPI_MST_TX_AFIFO_REMPTY_ERR_INT interrupt. (R/W)

SPI_APP2_INT_ENA Write 1 to enable SPI_APP2_INT interrupt. (R/W)

SPI_APP1_INT_ENA Write 1 to enable SPI_APP1_INT interrupt. (R/W)

Register 46.56. SPI_DMA_INT_CLR_REG (0x0038)



SPI_DMA_INFIFO_FULL_ERR_INT_CLR Write 1 to clear SPI_DMA_INFIFO_FULL_ERR_INT interrupt. (WT)

SPI_DMA_OUTFIFO_EMPTY_ERR_INT_CLR Write 1 to clear SPI_DMA_OUTFIFO_EMPTY_ERR_INT interrupt. (WT)

SPI_SLV_EX_QPI_INT_CLR Write 1 to clear SPI_SLV_EX_QPI_INT interrupt. (WT)

SPI_SLV_EN_QPI_INT_CLR Write 1 to clear SPI_SLV_EN_QPI_INT interrupt. (WT)

SPI_SLV_CMD7_INT_CLR Write 1 to clear SPI_SLV_CMD7_INT interrupt. (WT)

SPI_SLV_CMD8_INT_CLR Write 1 to clear SPI_SLV_CMD8_INT interrupt. (WT)

SPI_SLV_CMD9_INT_CLR Write 1 to clear SPI_SLV_CMD9_INT interrupt. (WT)

SPI_SLV_CMDA_INT_CLR Write 1 to clear SPI_SLV_CMDA_INT interrupt. (WT)

SPI_SLV_RD_DMA_DONE_INT_CLR Write 1 to clear SPI_SLV_RD_DMA_DONE_INT interrupt. (WT)

SPI_SLV_WR_DMA_DONE_INT_CLR Write 1 to clear SPI_SLV_WR_DMA_DONE_INT interrupt. (WT)

SPI_SLV_RD_BUF_DONE_INT_CLR Write 1 to clear SPI_SLV_RD_BUF_DONE_INT interrupt. (WT)

SPI_SLV_WR_BUF_DONE_INT_CLR Write 1 to clear SPI_SLV_WR_BUF_DONE_INT interrupt. (WT)

Continued on the next page...

Register 46.56. SPI_DMA_INT_CLR_REG (0x0038)

Continued from the previous page...

SPI_TRANS_DONE_INT_CLR Write 1 to clear SPI_TRANS_DONE_INT interrupt. (WT)

SPI_DMA_SEG_TRANS_DONE_INT_CLR Write 1 to clear SPI_DMA_SEG_TRANS_DONE_INT interrupt. (WT)

SPI_SLV_BUF_ADDR_ERR_INT_CLR Write 1 to clear SPI_SLV_BUF_ADDR_ERR_INT interrupt. (WT)

SPI_SLV_CMD_ERR_INT_CLR Write 1 to clear SPI_SLV_CMD_ERR_INT interrupt. (WT)

SPI_MST_RX_AFIFO_WFULL_ERR_INT_CLR Write 1 to clear SPI_MST_RX_AFIFO_WFULL_ERR_INT interrupt. (WT)

SPI_MST_TX_AFIFO_REMPTY_ERR_INT_CLR Write 1 to clear SPI_MST_TX_AFIFO_REMPTY_ERR_INT interrupt. (WT)

SPI_APP2_INT_CLR Write 1 to clear SPI_APP2_INT interrupt. (WT)

SPI_APP1_INT_CLR Write 1 to clear SPI_APP1_INT interrupt. (WT)

Register 46.57. SPI_DMA_INT_RAW_REG (0x003C)



SPI_DMA_INFIFO_FULL_ERR_INT_RAW The raw interrupt status of SPI_DMA_INFIFO_FULL_ERR_INT interrupt.
(R/WTC/SS)

SPI_DMA_OUTFIFO_EMPTY_ERR_INT_RAW The raw interrupt status of SPI_DMA_OUTFIFO_EMPTY_ERR_INT interrupt.
(R/WTC/SS)

SPI_SLV_EX_QPI_INT_RAW The raw interrupt status of SPI_SLV_EX_QPI_INT interrupt.
(R/WTC/SS)

SPI_SLV_EN_QPI_INT_RAW The raw interrupt status of SPI_SLV_EN_QPI_INT interrupt.
(R/WTC/SS)

SPI_SLV_CMD7_INT_RAW The raw interrupt status of SPI_SLV_CMD7_INT interrupt.
(R/WTC/SS)

SPI_SLV_CMD8_INT_RAW The raw interrupt status of SPI_SLV_CMD8_INT interrupt.
(R/WTC/SS)

SPI_SLV_CMD9_INT_RAW The raw interrupt status of SPI_SLV_CMD9_INT interrupt.
(R/WTC/SS)

SPI_SLV_CMDA_INT_RAW The raw interrupt status of SPI_SLV_CMDA_INT interrupt.
(R/WTC/SS)

SPI_SLV_RD_DMA_DONE_INT_RAW The raw interrupt status of SPI_SLV_RD_DMA_DONE_INT interrupt.
(R/WTC/SS)

SPI_SLV_WR_DMA_DONE_INT_RAW The raw interrupt status of SPI_SLV_WR_DMA_DONE_INT interrupt.
(R/WTC/SS)

SPI_SLV_RD_BUF_DONE_INT_RAW The raw interrupt status of SPI_SLV_RD_BUF_DONE_INT interrupt.
(R/WTC/SS)

Continued on the next page...

Register 46.57. SPI_DMA_INT_RAW_REG (0x003C)

Continued from the previous page...

SPI_SLV_WR_BUF_DONE_INT_RAW The raw interrupt status of SPI_SLV_WR_BUF_DONE_INT interrupt.
(R/WTC/SS)

SPI_TRANS_DONE_INT_RAW The raw interrupt status of SPI_TRANS_DONE_INT interrupt.
(R/WTC/SS)

SPI_DMA_SEG_TRANS_DONE_INT_RAW The raw interrupt status of SPI_DMA_SEG_TRANS_DONE_INT interrupt.
(R/WTC/SS)

SPI_SLV_BUF_ADDR_ERR_INT_RAW The raw interrupt status of SPI_SLV_BUF_ADDR_ERR_INT interrupt.
(R/WTC/SS)

SPI_SLV_CMD_ERR_INT_RAW The raw interrupt status of SPI_SLV_CMD_ERR_INT interrupt.
(R/WTC/SS)

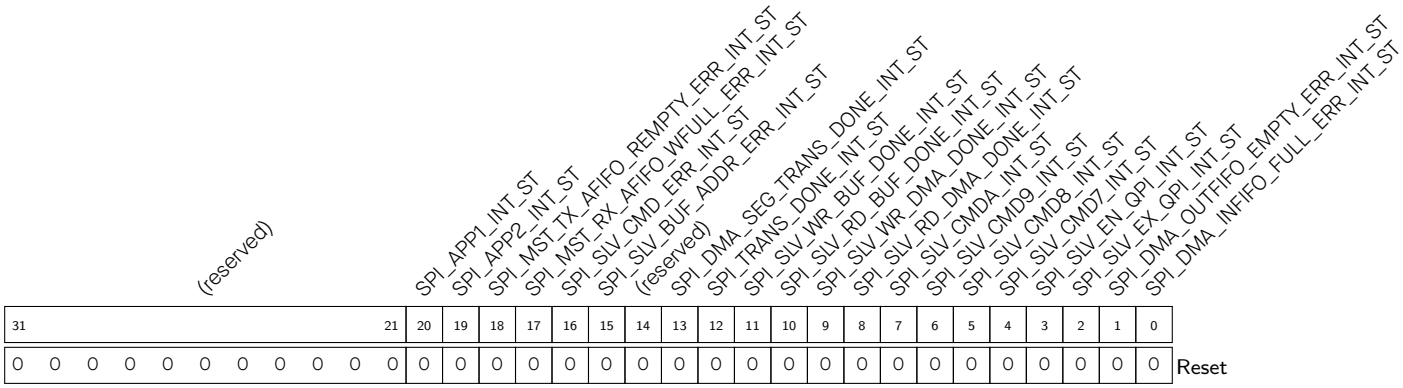
SPI_MST_RX_AFIFO_WFULL_ERR_INT_RAW The raw interrupt status of SPI_MST_RX_AFIFO_WFULL_ERR_INT interrupt.
(R/WTC/SS)

SPI_MST_TX_AFIFO_REMPTY_ERR_INT_RAW The raw interrupt status of SPI_MST_TX_AFIFO_REMPTY_ERR_INT interrupt.
(R/WTC/SS)

SPI_APP2_INT_RAW The raw interrupt status of SPI_APP2_INT interrupt. The value is only controlled by the application.
(R/WTC/SS)

SPI_APP1_INT_RAW The raw interrupt status of SPI_APP1_INT interrupt. The value is only controlled by the application.
(R/WTC/SS)

Register 46.58. SPI_DMA_INT_ST_REG (0x0040)



SPI_DMA_INFIFO_FULL_ERR_INT_ST The interrupt status of SPI_DMA_INFIFO_FULL_ERR_INT interrupt.
(RO)

SPI_DMA_OUTFIFO_EMPTY_ERR_INT_ST The interrupt status of SPI_DMA_OUTFIFO_EMPTY_ERR_INT interrupt.
(RO)

SPI_SLV_EX_QPI_INT_ST The interrupt status of SPI_SLV_EX_QPI_INT interrupt.
(RO)

SPI_SLV_EN_QPI_INT_ST The interrupt status of SPI_SLV_EN_QPI_INT interrupt.
(RO)

SPI_SLV_CMD7_INT_ST The interrupt status of SPI_SLV_CMD7_INT interrupt.
(RO)

SPI_SLV_CMD8_INT_ST The interrupt status of SPI_SLV_CMD8_INT interrupt.
(RO)

SPI_SLV_CMD9_INT_ST The interrupt status of SPI_SLV_CMD9_INT interrupt.
(RO)

SPI_SLV_CMDA_INT_ST The interrupt status of SPI_SLV_CMDA_INT interrupt.
(RO)

SPI_SLV_RD_DMA_DONE_INT_ST The interrupt status of SPI_SLV_RD_DMA_DONE_INT interrupt.
(RO)

SPI_SLV_WR_DMA_DONE_INT_ST The interrupt status of SPI_SLV_WR_DMA_DONE_INT interrupt.
(RO)

SPI_SLV_RD_BUF_DONE_INT_ST The interrupt status of SPI_SLV_RD_BUF_DONE_INT interrupt.
(RO)

SPI_SLV_WR_BUF_DONE_INT_ST The interrupt status of SPI_SLV_WR_BUF_DONE_INT interrupt.
(RO)

Continued on the next page...

Register 46.58. SPI_DMA_INT_ST_REG (0x0040)

Continued from the previous page...

SPI_TRANS_DONE_INT_ST The interrupt status of SPI_TRANS_DONE_INT interrupt.
(RO)

SPI_DMA_SEG_TRANS_DONE_INT_ST The interrupt status of SPI_DMA_SEG_TRANS_DONE_INT interrupt.
(RO)

SPI_SLV_BUF_ADDR_ERR_INT_ST The interrupt status of SPI_SLV_BUF_ADDR_ERR_INT interrupt.
(RO)

SPI_SLV_CMD_ERR_INT_ST The interrupt status of SPI_SLV_CMD_ERR_INT interrupt.
(RO)

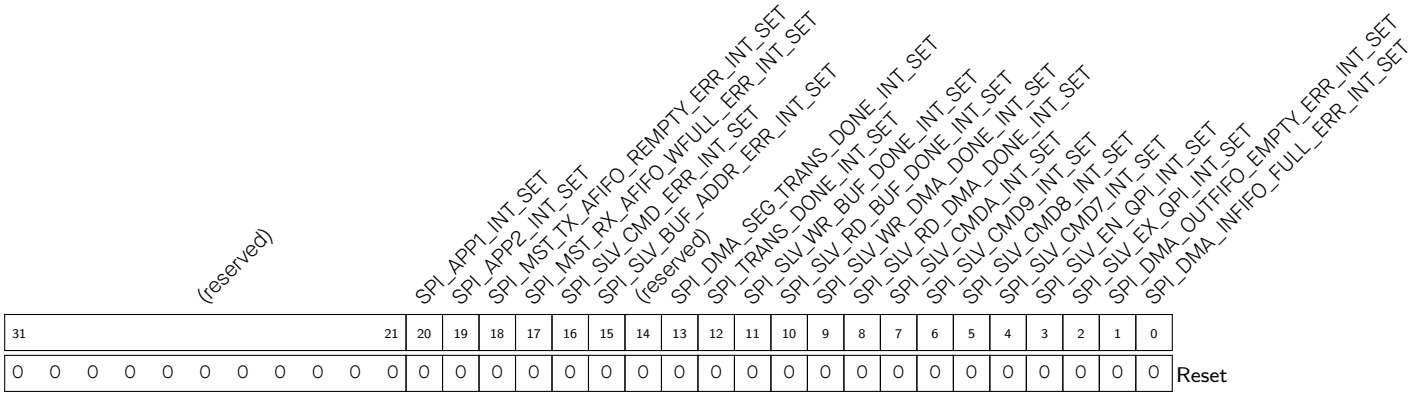
SPI_MST_RX_AFIFO_WFULL_ERR_INT_ST The interrupt status of SPI_MST_RX_AFIFO_WFULL_ERR_INT interrupt.
(RO)

SPI_MST_TX_AFIFO_REMPTY_ERR_INT_ST The interrupt status of SPI_MST_TX_AFIFO_REMPTY_ERR_INT interrupt.
(RO)

SPI_APP2_INT_ST The interrupt status of SPI_APP2_INT interrupt.
(RO)

SPI_APP1_INT_ST The interrupt status of SPI_APP1_INT interrupt.
(RO)

Register 46.59. SPI_DMA_INT_SET_REG (0x0044)



SPI_DMA_INFIFO_FULL_ERR_INT_SET Write 1 to set SPI_DMA_INFIFO_FULL_ERR_INT interrupt. (WT)

SPI_DMA_OUTFIFO_EMPTY_ERR_INT_SET Write 1 to set SPI_DMA_OUTFIFO_EMPTY_ERR_INT interrupt. (WT)

SPI_SLV_EX_QPI_INT_SET Write 1 to set SPI_SLV_EX_QPI_INT interrupt. (WT)

SPI_SLV_EN_QPI_INT_SET Write 1 to set SPI_SLV_EN_QPI_INT interrupt. (WT)

SPI_SLV_CMD7_INT_SET Write 1 to set SPI_SLV_CMD7_INT interrupt. (WT)

SPI_SLV_CMD8_INT_SET Write 1 to set SPI_SLV_CMD8_INT interrupt. (WT)

SPI_SLV_CMD9_INT_SET Write 1 to set SPI_SLV_CMD9_INT interrupt. (WT)

SPI_SLV_CMDA_INT_SET Write 1 to set SPI_SLV_CMDA_INT interrupt. (WT)

SPI_SLV_RD_DMA_DONE_INT_SET Write 1 to set SPI_SLV_RD_DMA_DONE_INT interrupt. (WT)

SPI_SLV_WR_DMA_DONE_INT_SET Write 1 to set SPI_SLV_WR_DMA_DONE_INT interrupt. (WT)

SPI_SLV_RD_BUF_DONE_INT_SET Write 1 to set SPI_SLV_RD_BUF_DONE_INT interrupt. (WT)

SPI_SLV_WR_BUF_DONE_INT_SET Write 1 to set SPI_SLV_WR_BUF_DONE_INT interrupt. (WT)

SPI_TRANS_DONE_INT_SET Write 1 to set SPI_TRANS_DONE_INT interrupt. (WT)

Continued on the next page...

Register 46.59. SPI_DMA_INT_SET_REG (0x0044)

Continued from the previous page...

SPI_DMA_SEG_TRANS_DONE_INT_SET Write 1 to set SPI_DMA_SEG_TRANS_DONE_INT interrupt. (WT)

SPI_SLV_BUF_ADDR_ERR_INT_SET Write 1 to set SPI_SLV_BUF_ADDR_ERR_INT interrupt. (WT)

SPI_SLV_CMD_ERR_INT_SET Write 1 to set SPI_SLV_CMD_ERR_INT interrupt. (WT)

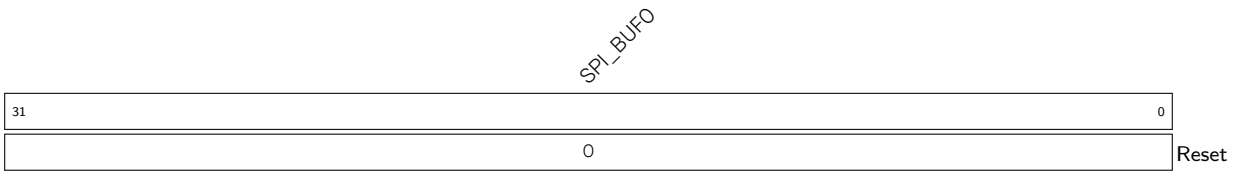
SPI_MST_RX_AFIFO_WFULL_ERR_INT_SET Write 1 to set SPI_MST_RX_AFIFO_WFULL_ERR_INT interrupt. (WT)

SPI_MST_TX_AFIFO_REMPTY_ERR_INT_SET Write 1 to set SPI_MST_TX_AFIFO_REMPTY_ERR_INT interrupt. (WT)

SPI_APP2_INT_SET Write 1 to set SPI_APP2_INT interrupt. (WT)

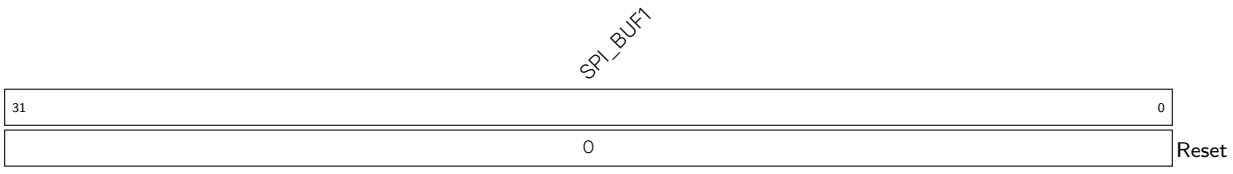
SPI_APP1_INT_SET Write 1 to set SPI_APP1_INT interrupt. (WT)

Register 46.60. SPI_W0_REG (0x0098)



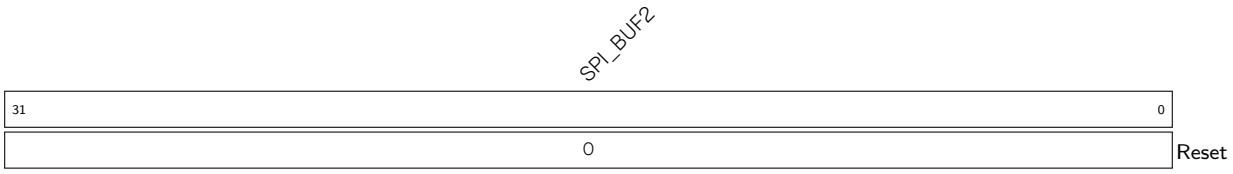
SPI_BUF0 32-bit data buffer 0. (R/W/SS)

Register 46.61. SPI_W1_REG (0x009C)



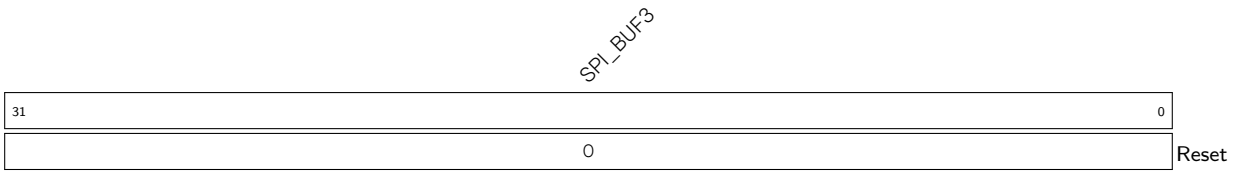
SPI_BUF1 32-bit data buffer 1. (R/W/SS)

Register 46.62. SPI_W2_REG (0x00A0)



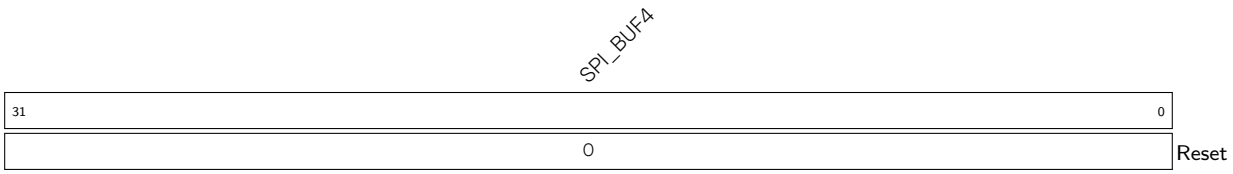
SPI_BUF2 32-bit data buffer 2.
(R/W/SS)

Register 46.63. SPI_W3_REG (0x00A4)



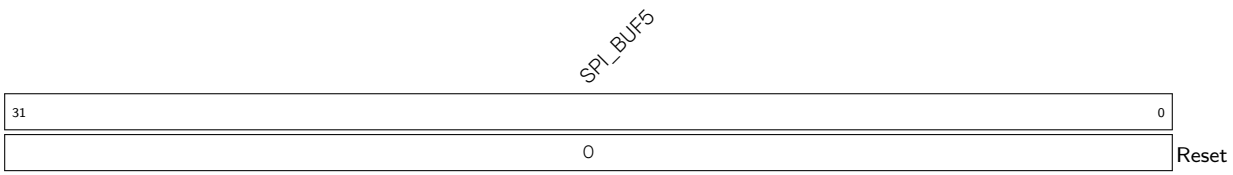
SPI_BUF3 32-bit data buffer 3.
(R/W/SS)

Register 46.64. SPI_W4_REG (0x00A8)



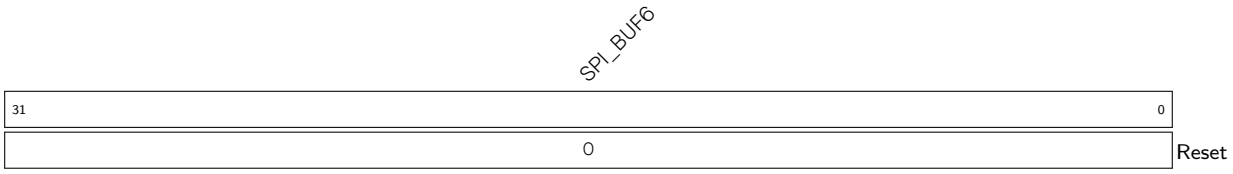
SPI_BUF4 32-bit data buffer 4.
(R/W/SS)

Register 46.65. SPI_W5_REG (0x00AC)



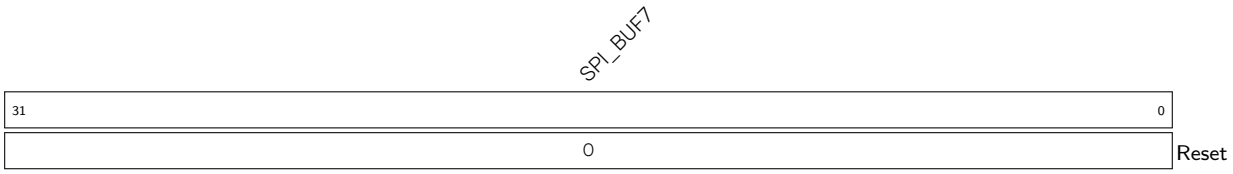
SPI_BUF5 32-bit data buffer 5.
(R/W/SS)

Register 46.66. SPI_W6_REG (0x00B0)



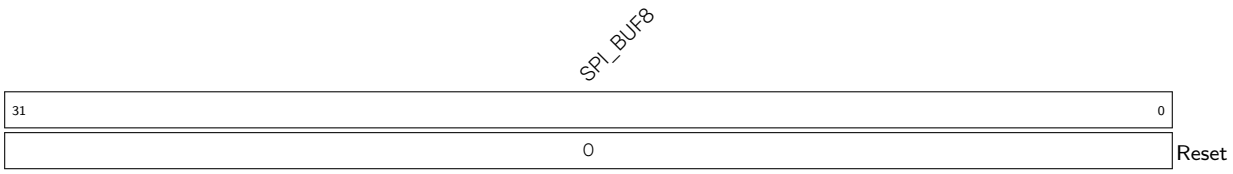
SPI_BUF6 32-bit data buffer 6.
(R/W/SS)

Register 46.67. SPI_W7_REG (0x00B4)



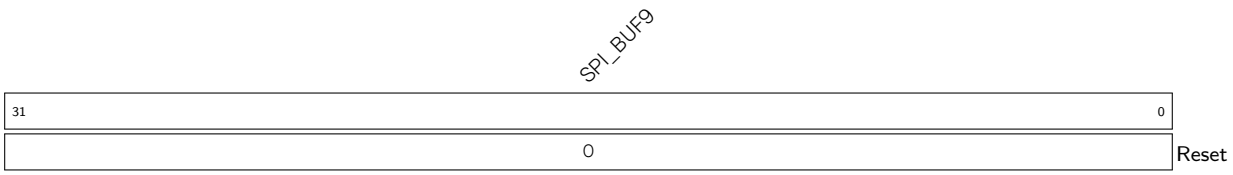
SPI_BUF7 32-bit data buffer 7.
(R/W/SS)

Register 46.68. SPI_W8_REG (0x00B8)



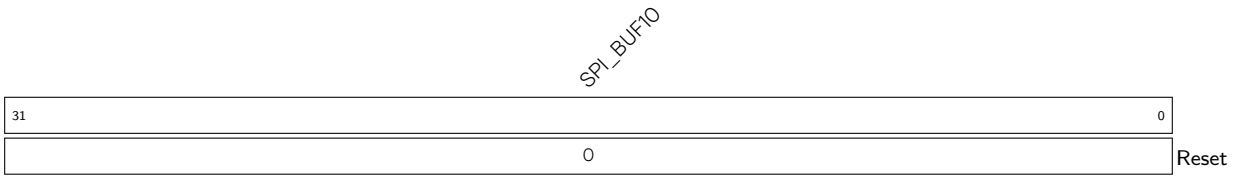
SPI_BUF8 32-bit data buffer 8.
(R/W/SS)

Register 46.69. SPI_W9_REG (0x00BC)



SPI_BUF9 32-bit data buffer 9.
(R/W/SS)

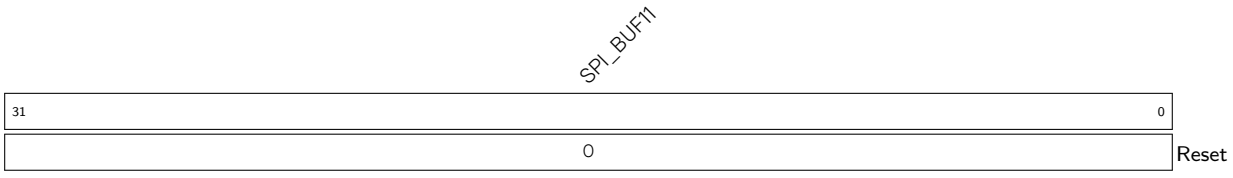
Register 46.70. SPI_W10_REG (0x00C0)



SPI_BUF10 32-bit data buffer 10.

(R/W/SS)

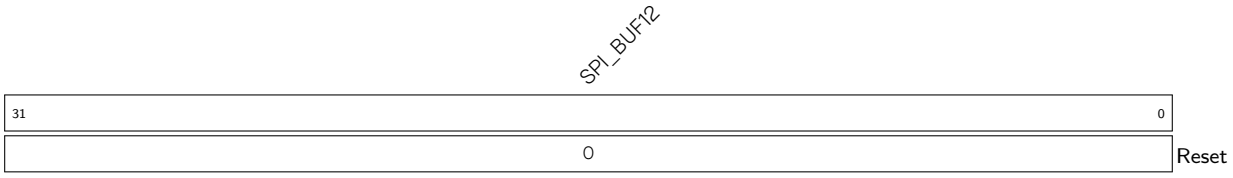
Register 46.71. SPI_W11_REG (0x00C4)



SPI_BUF11 32-bit data buffer 11.

(R/W/SS)

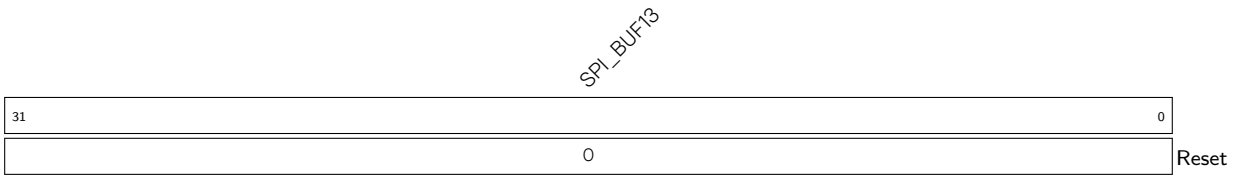
Register 46.72. SPI_W12_REG (0x00C8)



SPI_BUF12 32-bit data buffer 12.

(R/W/SS)

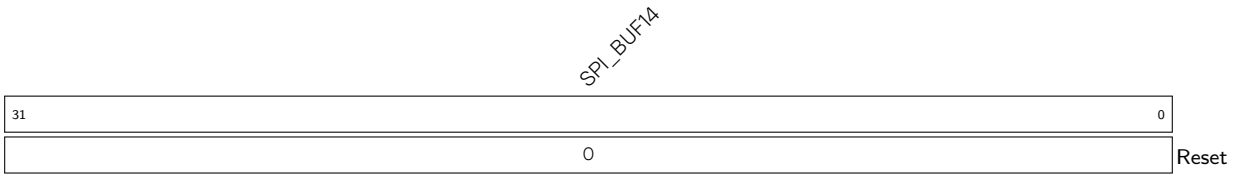
Register 46.73. SPI_W13_REG (0x00CC)



SPI_BUF13 32-bit data buffer 13.

(R/W/SS)

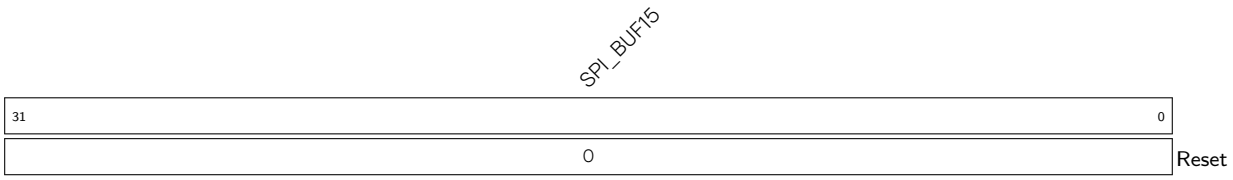
Register 46.74. SPI_W14_REG (0x00D0)



SPI_BUF14 32-bit data buffer 14.

(R/W/SS)

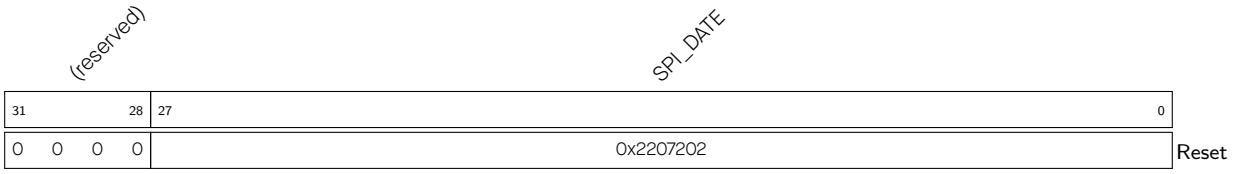
Register 46.75. SPI_W15_REG (0x00D4)



SPI_BUF15 32-bit data buffer 15.

(R/W/SS)

Register 46.76. SPI_DATE_REG (0x00F0)



SPI_DATE Version control register.

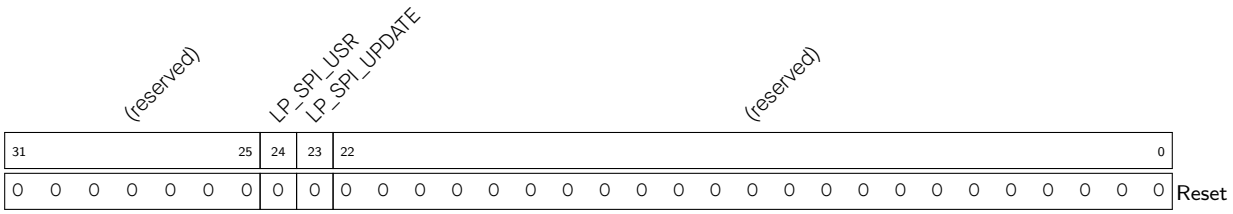
(R/W)

46.13.3 LP-SPI Register

The addresses in this section are relative to LP-SPI base address provided in Table 9.3-2 in Chapter 9 *System and Memory*.

For how to program reserved fields, please refer to Section *Programming Reserved Register Field*.

Register 46.77. LP_SPI_CMD_REG (0x0000)



LP_SPI_UPDATE Configures whether or not to synchronize SPI registers from APB clock domain into SPI module clock domain.

0: Not synchronize

1: Synchronize

This bit is only used in SPI master transfer.

(WT)

LP_SPI_USR Configures whether or not to enable user-defined command.

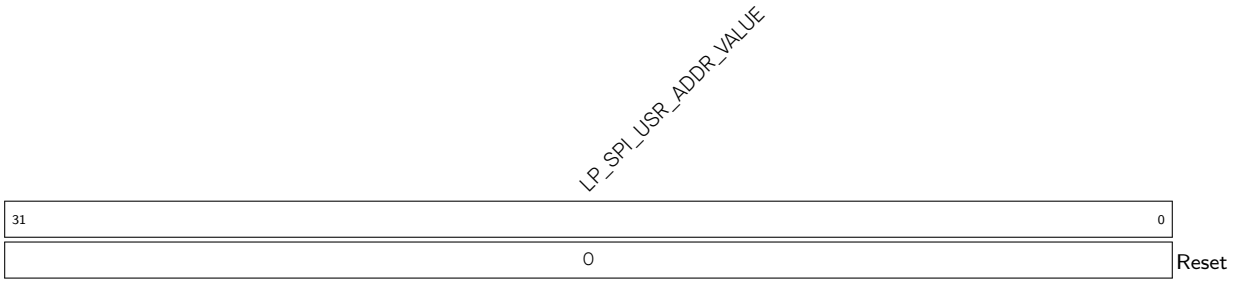
0: Not enable

1: Enable

An SPI operation will be triggered when the bit is set. This bit will be cleared once the operation is done.

(R/W/SC)

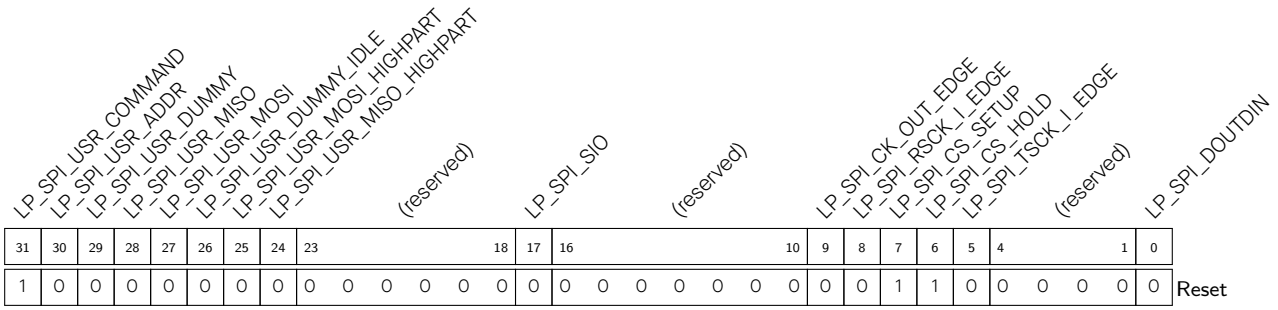
Register 46.78. LP_SPI_ADDR_REG (0x0004)



LP_SPI_USR_ADDR_VALUE Configures the address to slave.

(R/W)

Register 46.79. LP_SPI_USER_REG (0x0010)



LP_SPI_DOUTDIN Configures whether or not to enable full-duplex communication.

0: Disable

1: Enable

(R/W)

LP_SPI_TSCK_I_EDGE Configures whether or not to change the polarity of TSCK in slave transfer.

0: TSCK = SPI_CK_I

1: TSCK = !SPI_CK_I

(R/W)

LP_SPI_CS_HOLD Configures whether or not to keep SPI CS low when SPI is in DONE state.

0: Not keep low

1: Keep low

(R/W)

LP_SPI_CS_SETUP Configures whether or not to enable SPI CS when SPI is in prepare (PREP) state.

0: Disable

1: Enable

(R/W)

LP_SPI_RSCK_I_EDGE Configures whether or not to change the polarity of RSCK in slave transfer.

0: RSCK = !SPI_CK_I

1: RSCK = SPI_CK_I

(R/W)

LP_SPI_CK_OUT_EDGE Configures SPI clock mode together with LP_SPI_CK_IDLE_EDGE. For more information, see Section 46.7.4. (R/W)

LP_SPI_SIO Configures whether or not to enable 3-line half-duplex communication, where MOSI and MISO signals share the same pin.

0: Disable

1: Enable

(R/W)

Continued on the next page...

Register 46.79. LP_SPI_USER_REG (0x0010)

Continued from the previous page...

LP_SPI_USR_MISO_HIGHPART Configures whether or not to enable High-Part mode in read-data phase, i.e., only access to high-part of the buffers: LP_SPI_W8_REG ~ LP_SPI_W15_REG.
0: Disable
1: Enable
(R/W)

LP_SPI_USR_MOSI_HIGHPART Configures whether or not to enable High-Part mode in write-data phase, i.e., only access to high-part of the buffers: LP_SPI_W8_REG ~ LP_SPI_W15_REG.
0: Disable
1: Enable
(R/W)

LP_SPI_USR_DUMMY_IDLE Configures whether or not to disable SPI clock in DUMMY state.
0: Not disable
1: Disable
(R/W)

LP_SPI_USR_MOSI Configures whether or not to enable the write-data (DOUT) state of an operation.
0: Disable
1: Enable
(R/W)

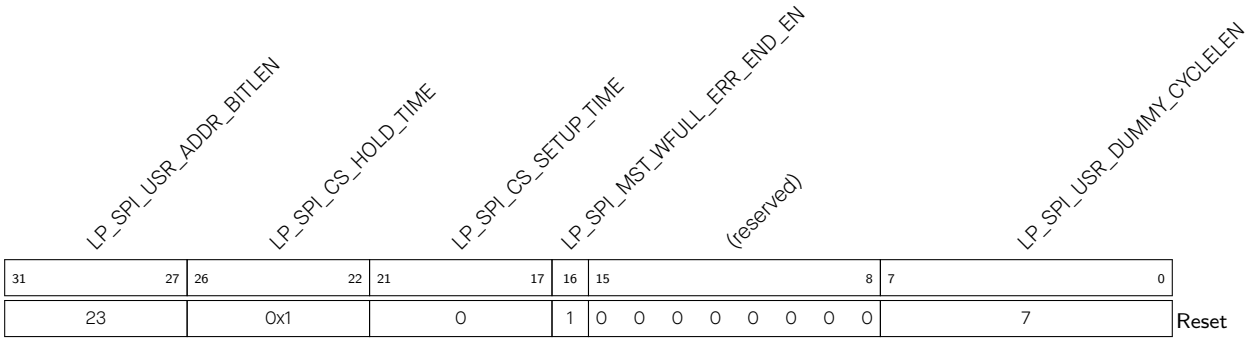
LP_SPI_USR_MISO Configures whether or not to enable the read-data (DIN) state of an operation.
0: Disable
1: Enable
(R/W)

LP_SPI_USR_DUMMY Configures whether or not to enable the DUMMY state of an operation.
0: Disable
1: Enable
(R/W)

LP_SPI_USR_ADDR Configures whether or not to enable the address (ADDR) state of an operation.
0: Disable
1: Enable
(R/W)

LP_SPI_USR_COMMAND Configures whether or not to enable the command (CMD) state of an operation.
0: Disable
1: Enable
(R/W)

Register 46.80. LP_SPI_USER1_REG (0x0014)



LP_SPI_USR_DUMMY_CYCLELEN Configures the length of DUMMY state.

Measurement unit: SPI_CLK clock cycles.

This value is (the expected cycle number - 1). (R/W)

LP_SPI_MST_WFULL_ERR_END_EN Configures whether or not to end the LP-SPI transfer when SPI RX AFIFO wfull error occurs in master full-/half-duplex transfers.

0: Not end

1: End

(R/W)

LP_SPI_CS_SETUP_TIME Configures the length of prepare (PREP) state.

Measurement unit: SPI_CLK clock cycles.

This value is equal to the expected cycles - 1. This field is used together with LP_SPI_CS_SETUP. (R/W)

LP_SPI_CS_HOLD_TIME Configures the delay cycles of CS pin.

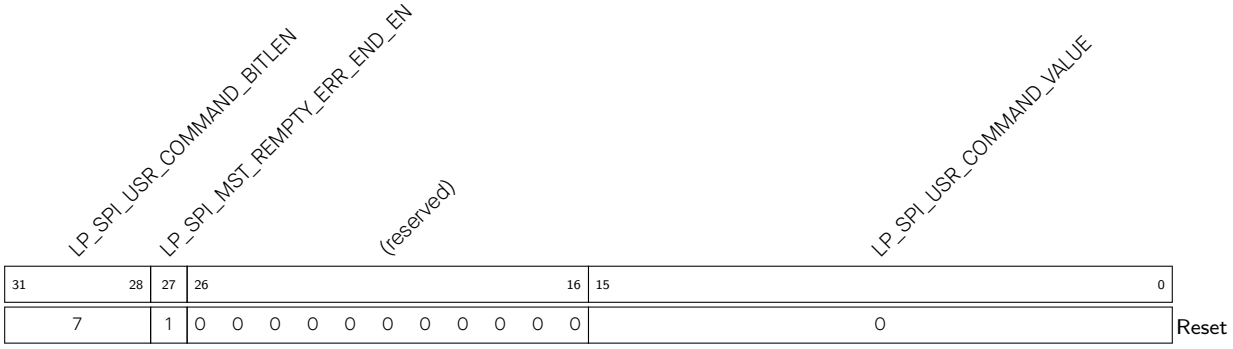
Measurement unit: SPI_CLK clock cycles.

This field is used together with LP_SPI_CS_HOLD. (R/W)

LP_SPI_USR_ADDR_BITLEN Configures the bit length in address state.

This value is (expected bit number - 1). (R/W)

Register 46.81. LP_SPI_USER2_REG (0x0018)



LP_SPI_USR_COMMAND_VALUE Configures the command value. (R/W)

LP_SPI_MST_REMPTY_ERR_END_EN Configures whether or not to end the LP-SPI transfer when SPI TX AFIFO read empty error occurs in master full-/half-duplex transfers.

0: Not end

1: End

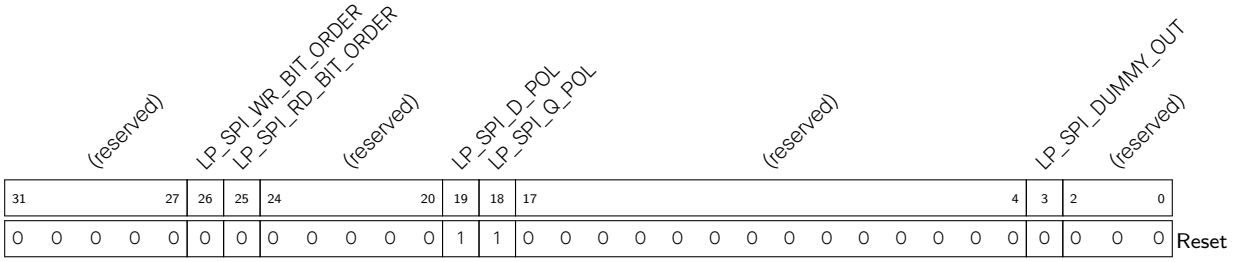
(R/W)

LP_SPI_USR_COMMAND_BITLEN Configures the bit length of command state.

This value is (expected bit number - 1).

(R/W)

Register 46.82. LP_SPI_CTRL_REG (0x0008)



LP_SPI_DUMMY_OUT Configures whether or not to output the LP_SPI bus signals in DUMMY state.

0: Not output

1: Output

(R/W)

LP_SPI_Q_POL Configures MISO line polarity.

0: Low

1: High

(R/W)

LP_SPI_D_POL Configures MOSI line polarity.

0: Low

1: High

(R/W)

LP_SPI_RD_BIT_ORDER Configures the bit order in read-data (MISO) state.

0: MSB first

1: LSB first

(R/W)

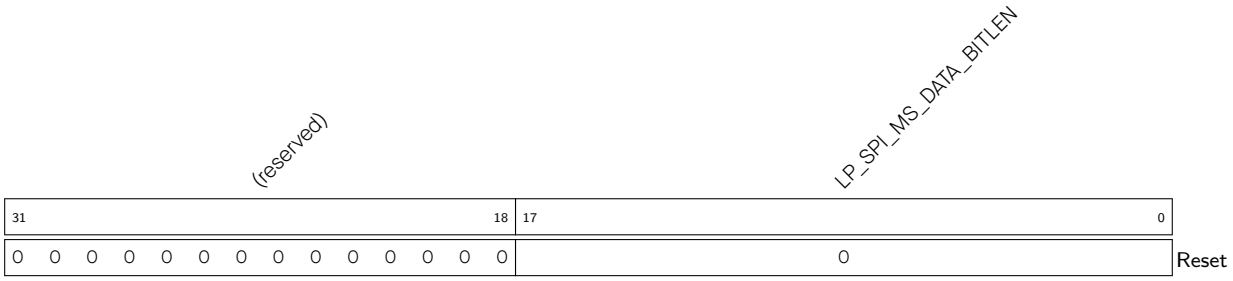
LP_SPI_WR_BIT_ORDER Configures the bit order in command (CMD), address (ADDR), and write-data (MOSI) states.

0: MSB first

1: LSB first

(R/W)

Register 46.83. LP_SPI_MS_DLEN_REG (0x001C)

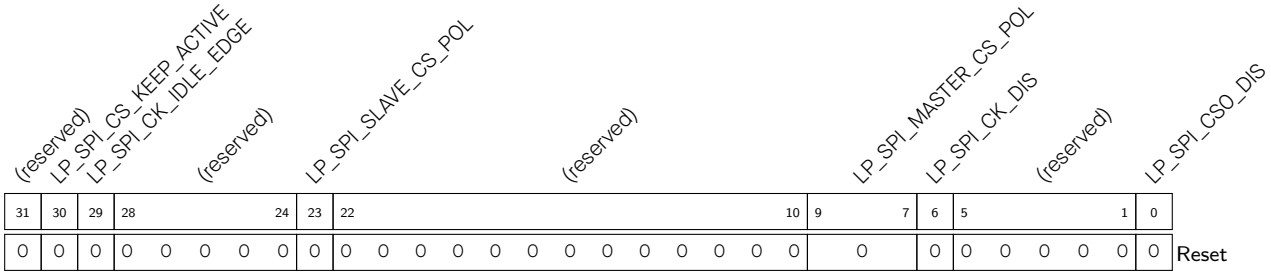


LP_SPI_MS_DATA_BITLEN Configures the data bit length of SPI transfer in CPU-controlled master transfer.

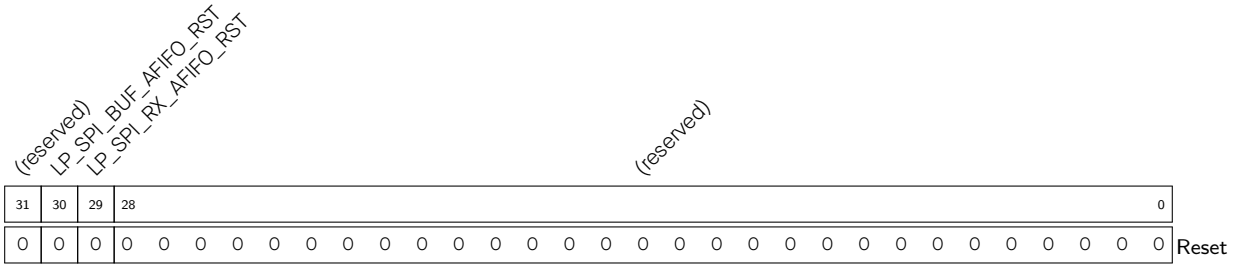
This value shall be (expected bit_num - 1).

(R/W)

Register 46.84. LP_SPI_MISC_REG (0x0020)


(reserved)
LP_SPI_CS_KEEP_ACTIVE
LP_SPI_CK_IDLE_EDGE
(reserved)
LP_SPI_SLAVE_CS_POL
(reserved)
LP_SPI_MASTER_CS_POL
LP_SPI_CK_DIS
(reserved)
LP_SPI_CSO_DIS
31 30 29 28 24 23 22 10 9 7 6 5 1 0
0 0 0

Register 46.85. LP_SPI_DMA_CONF_REG (0x0030)



LP_SPI_RX_AFIFO_RST Configures to reset spi_rx_afifo as shown in Figure 46.5-3 and Figure 46.5-4.

0: Not reset

1: Reset

spi_rx_afifo is used to receive data in SPI master and slave transfer.

(WT)

LP_SPI_BUF_AFIFO_RST Configures to reset buf_tx_afifo as shown in Figure 46.5-3 and Figure 46.5-4.

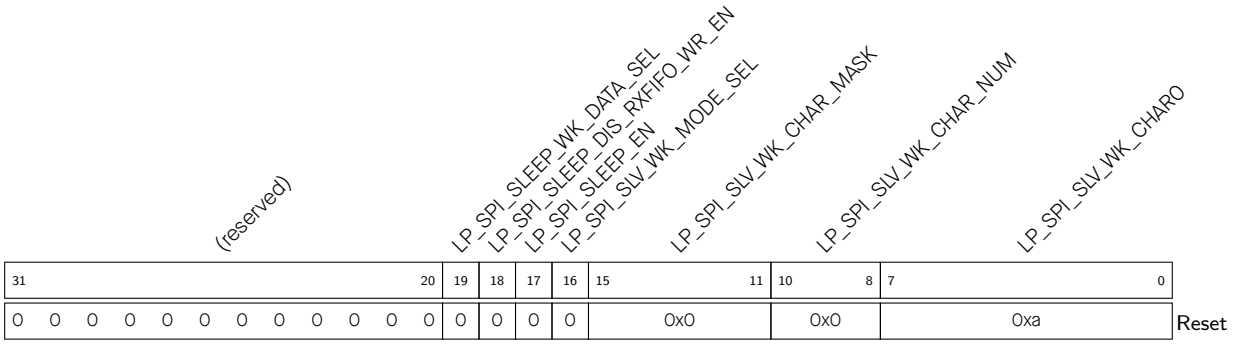
0: Not reset

1: Reset

buf_tx_afifo is used to send data out in CPU-controlled master and slave transfer.

(WT)

Register 46.86. LP_SPI_SLEEP_CONF0_REG (0x0044)



LP_SPI_SLV_WK_CHAR0 Configures wake-up character 0.

(R/W)

LP_SPI_SLV_WK_CHAR_NUM Configures the amount of the enabled wake-up characters.

(R/W)

LP_SPI_SLV_WK_CHAR_MASK Configures the bit map for the characters to be masked.

Bit 0 ~ bit 4 correspond to wake-up character 0 ~ character 4, respectively.

If the bit value is:

- 0: The corresponding wake-up character is not masked.
- 1: The corresponding wake-up character is masked.

(R/W)

LP_SPI_SLV_WK_MODE_SEL Configures the wake-up mode.

0: Wakes up the chip once the LP-SPI slave detects a start bit.

1: Wakes up the chip once the LP-SPI slave detects a specific sequence of characters.

(R/W)

LP_SPI_SLEEP_EN Configures whether or not to enable sleep mode.

0: Not enable

1: Enable

(R/W)

LP_SPI_SLEEP_DIS_RXFIFO_WR_EN Configures whether or not to store the data before wake-up.

0: Store the data

1: Not store the data

(R/W)

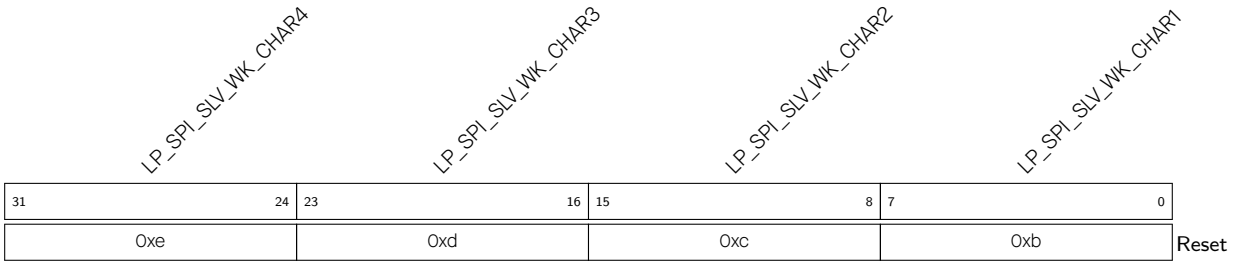
LP_SPI_SLEEP_WK_DATA_SEL Configures in which states to detect the wake-up characters.

0: Only detect wake-up characters from RX data in DATA state.

1: Detect wake-up characters from RX data in all states.

(R/W)

Register 46.87. LP_SPI_SLEEP_CONF1_REG (0x0048)



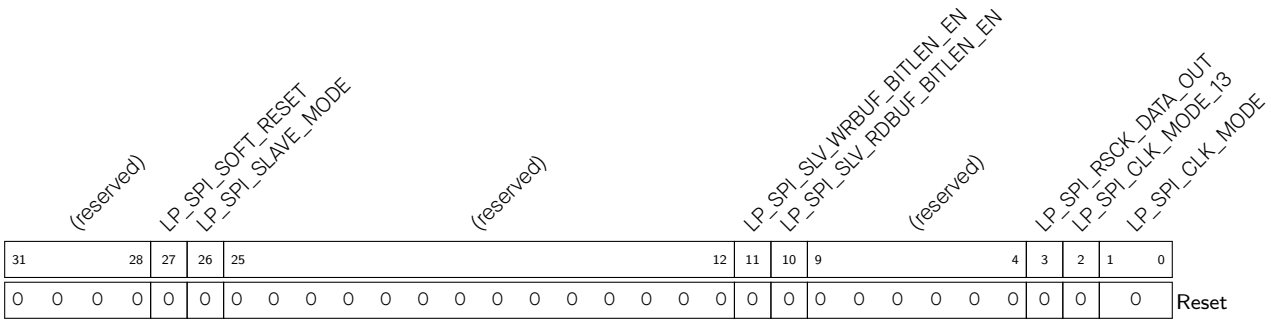
LP_SPI_SLV_WK_CHAR1 Configures wake-up character 1. (R/W)

LP_SPI_SLV_WK_CHAR2 Configures wake-up character 2. (R/W)

LP_SPI_SLV_WK_CHAR3 Configures wake-up character 3. (R/W)

LP_SPI_SLV_WK_CHAR4 Configures wake-up character 4. (R/W)

Register 46.88. LP_SPI_SLAVE_REG (0x00E0)



LP_SPI_CLK_MODE Configures LP-SPI clock mode.

0: LP-SPI clock is off when CS becomes inactive.

1: LP-SPI clock is delayed one cycle after CS becomes inactive.

2: LP-SPI clock is delayed two cycles after CS becomes inactive.

3: LP-SPI clock is always on.

(R/W)

LP_SPI_CLK_MODE_13 Configures clock mode.

0: SPI clock mode 1 and mode 3. See Table 46.7-2.

1: SPI clock mode 0 and mode 2. See Table 46.7-2.

(R/W)

LP_SPI_RSCK_DATA_OUT Configures the edge of output data.

0: Output data at TSCK rising edge.

1: Output data at RSCK rising edge.

(R/W)

LP_SPI_SLV_RDBUF_BITLEN_EN Configures whether or not to use SPI_SLV_DATA_BITLEN to store the data bit length of Rd_BUF transfer.

0: Not use

1: Use

(R/W)

LP_SPI_SLV_WRBUF_BITLEN_EN Configures whether or not to use SPI_SLV_DATA_BITLEN to store the data bit length of Wr_BUF transfer.

0: Not use

1: Use

(R/W)

LP_SPI_SLAVE_MODE Configures SPI work mode.

0: Master

1: Slave

(R/W)

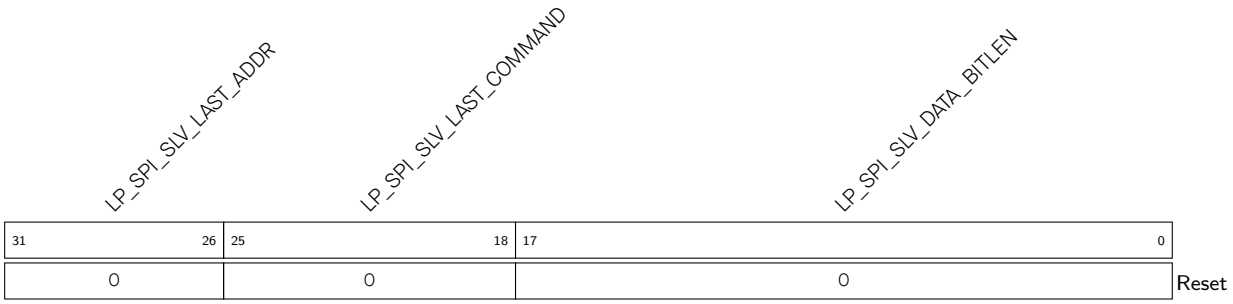
LP_SPI_SOFT_RESET Configures whether to reset the SPI clock line, CS line, and data line via software.

0: Not reset

1: Reset

(WT)

Register 46.89. LP_SPI_SLAVE1_REG (0x00E4)

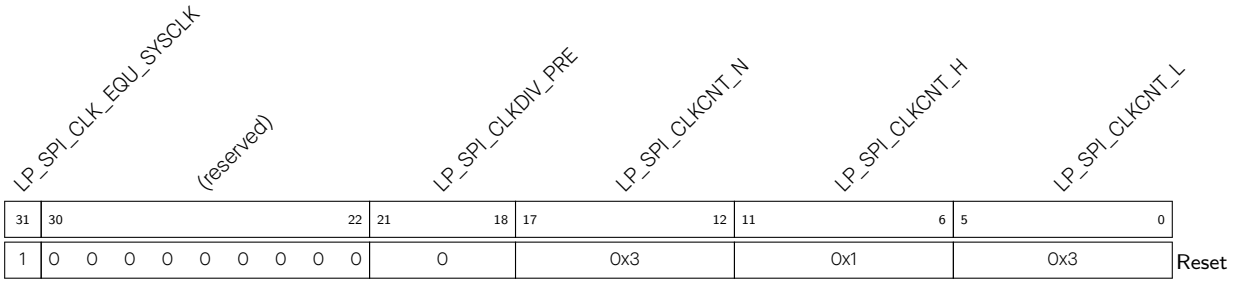


LP_SPI_SLV_DATA_BITLEN Configures the transferred data bit length in SPI slave full-/half-duplex modes.
(R/W/SS)

LP_SPI_SLV_LAST_COMMAND Configures the command value in slave mode.
(R/W/SS)

LP_SPI_SLV_LAST_ADDR Configures the address value in slave mode.
(R/W/SS)

Register 46.90. LP_SPI_CLOCK_REG (0x000C)



LP_SPI_CLKCNT_L Configures clock duty cycles, together with LP_SPI_CLKCNT_H and LP_SPI_CLKCNT_N.

In master transfer, this field must be equal to LP_SPI_CLKCNT_N.

In slave mode, it must be 0.

(R/W)

LP_SPI_CLKCNT_H Configures the duty cycle of LP_SPI_CLK (high level) in master transfer.

It's recommended to configure this value to floor((LP_SPI_CLKCNT_N + 1)/2 - 1). floor() here is to round a number down, e.g., floor(2.2) = 2.

In slave mode, it must be 0.

(R/W)

LP_SPI_CLKCNT_N Configures the divider of LP_SPI_CLK in master transfer.

LP_SPI_CLK frequency is $f_{lp_spi_pclk}$/(LP_SPI_CLKDIV_PRE + 1)/(LP_SPI_CLKCNT_N + 1).

(R/W)

LP_SPI_CLKDIV_PRE Configures the pre-divider of LP_SPI_CLK in master transfer.

(R/W)

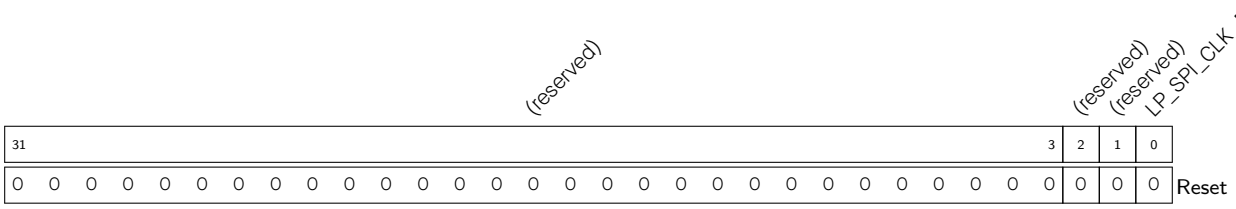
LP_SPI_CLK_EQU_SYSCLK Configures whether or not the LP_SPI_CLK is equal to lp_spi_pclk in master transfer.

0: LP_SPI_CLK is divided from lp_spi_pclk.

1: LP_SPI_CLK is equal to lp_spi_pclk.

(R/W)

Register 46.91. LP_SPI_CLK_GATE_REG (0x00E8)



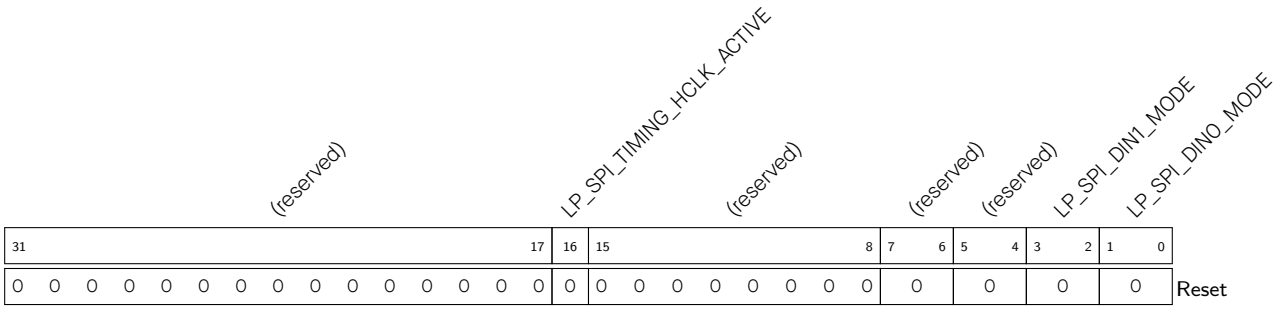
LP_SPI_CLK_EN Configures whether or not to enable clock gate.

0: Disable

1: Enable

(R/W)

Register 46.92. LP_SPI_DIN_MODE_REG (0x0024)



LP_SPI_DIN0_MODE Configures the input mode for LP_SPI_D signal.

0: Input without delay

1: Input at the (LP_SPI_DIN0_NUM + 1)th falling edge of clk_spi_mst

2: Input at the (LP_SPI_DIN0_NUM + 1)th rising edge of clk_hclk plus one clk_spi_mst rising edge cycle

3: Input at the (LP_SPI_DIN0_NUM + 1)th rising edge of clk_hclk plus one clk_spi_mst falling edge cycle

(R/W)

LP_SPI_DIN1_MODE Configures the input mode for LP_SPI_Q signal.

0: Input without delay

1: Input at the (LP_SPI_DIN1_NUM+1)th falling edge of clk_spi_mst

2: Input at the (LP_SPI_DIN1_NUM + 1)th rising edge of clk_hclk plus one clk_spi_mst rising edge cycle

3: Input at the (LP_SPI_DIN1_NUM + 1)th rising edge of clk_hclk plus one clk_spi_mst falling edge cycle

(R/W)

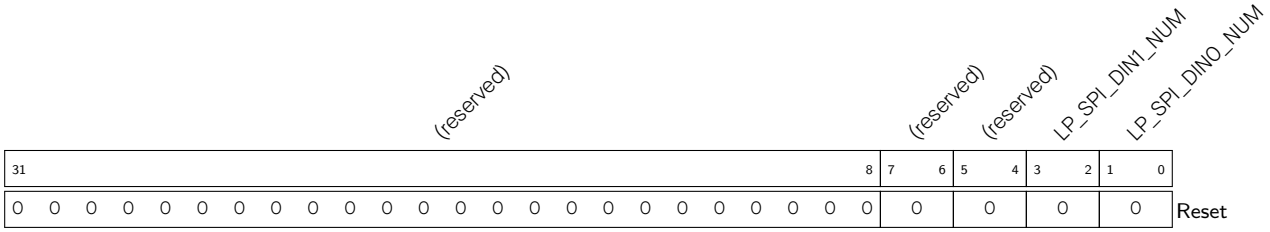
LP_SPI_TIMING_HCLK_ACTIVE Configures whether or not to enable HCLK (high-frequency clock) in LP-SPI input timing module.

0: Disable

1: Enable

(R/W)

Register 46.93. LP_SPI_DIN_NUM_REG (0x0028)



LP_SPI_DIN0_NUM Configures the delays to input signal LP_SPI_D based on the setting of LP_SPI_DIN0_MODE.

0: Delayed by 1 clock cycle

1: Delayed by 2 clock cycles

2: Delayed by 3 clock cycles

3: Delayed by 4 clock cycles

(R/W)

LP_SPI_DIN1_NUM Configures the delays to input signal LP_SPI_Q based on the setting of LP_SPI_DIN1_MODE.

0: Delayed by 1 clock cycle

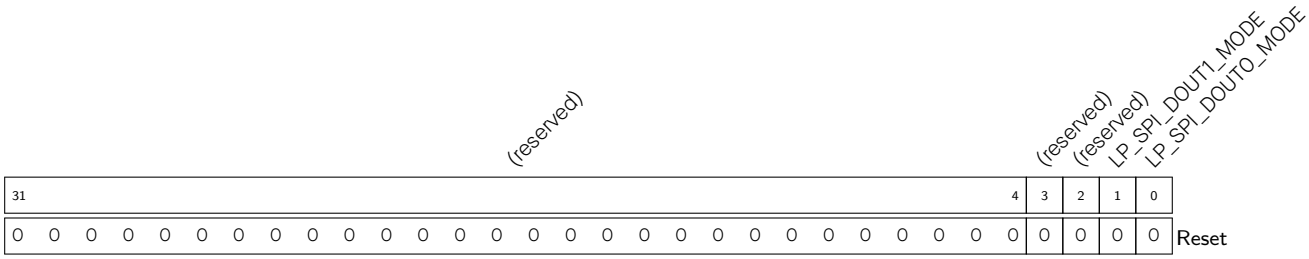
1: Delayed by 2 clock cycles

2: Delayed by 3 clock cycles

3: Delayed by 4 clock cycles

(R/W)

Register 46.94. LP_SPI_DOUT_MODE_REG (0x002C)



LP_SPI_DOUT0_MODE Configures the output mode for LP_SPI_D signal.

0: Output without delay

1: Output with a delay of an LP-SPI module clock cycle at its falling edge

(R/W)

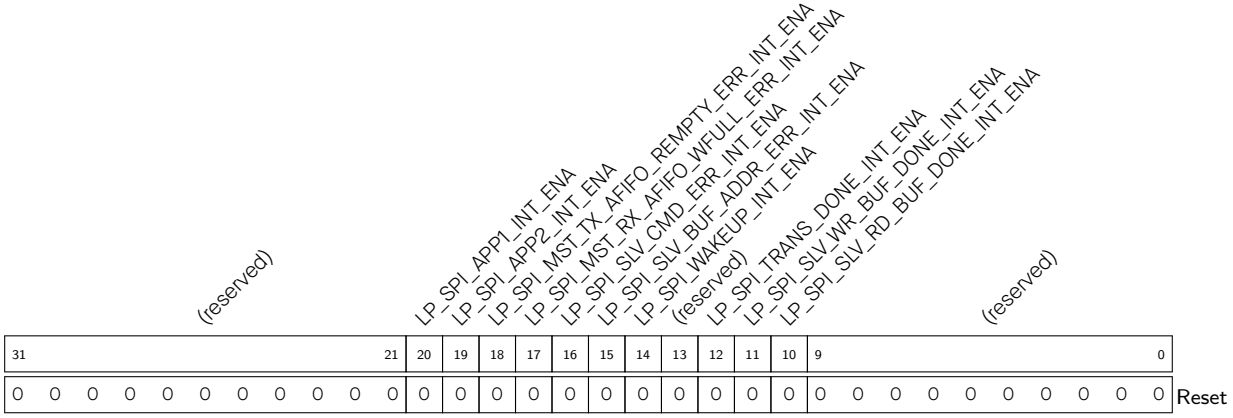
LP_SPI_DOUT1_MODE Configures the output mode for LP_SPI_Q signal.

0: Output without delay

1: Output with a delay of an LP-SPI module clock cycle at its falling edge

(R/W)

Register 46.95. LP_SPI_DMA_INT_ENA_REG (0x0034)



LP_SPI_SLV_RD_BUF_DONE_INT_ENA Write 1 to enable SPI_SLV_RD_BUF_DONE_INT interrupt. (R/W)

LP_SPI_SLV_WR_BUF_DONE_INT_ENA Write 1 to enable SPI_SLV_WR_BUF_DONE_INT interrupt. (R/W)

LP_SPI_TRANS_DONE_INT_ENA Write 1 to enable SPI_TRANS_DONE_INT interrupt. (R/W)

LP_SPI_WAKEUP_INT_ENA Write 1 to enable LP_SPI_WAKEUP_INT interrupt. (R/W)

LP_SPI_SLV_BUF_ADDR_ERR_INT_ENA Write 1 to enable SPI_SLV_BUF_ADDR_ERR_INT interrupt. (R/W)

LP_SPI_SLV_CMD_ERR_INT_ENA Write 1 to enable SPI_SLV_CMD_ERR_INT interrupt. (R/W)

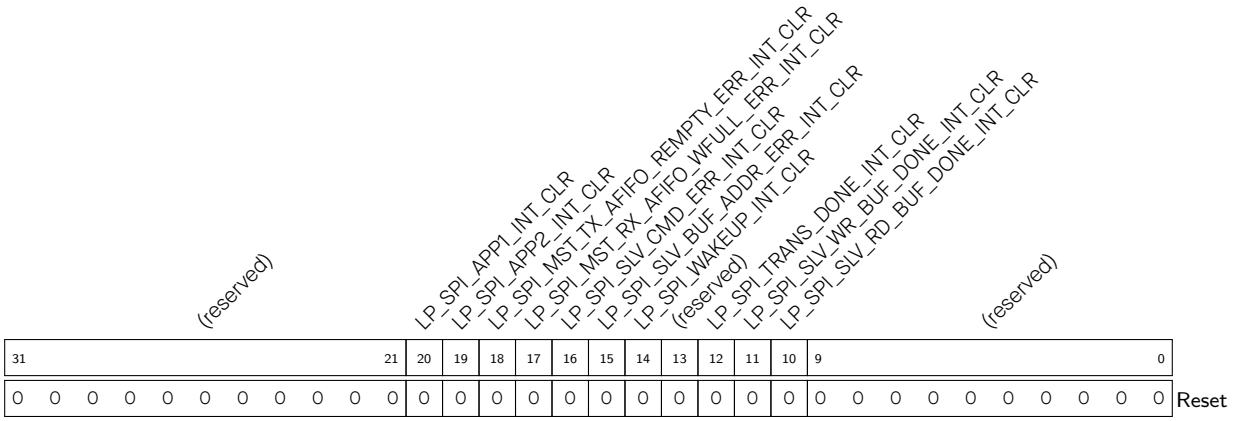
LP_SPI_MST_RX_AFIFO_WFULL_ERR_INT_ENA Write 1 to enable SPI_MST_RX_AFIFO_WFULL_ERR_INT interrupt. (R/W)

LP_SPI_MST_TX_AFIFO_REMPTY_ERR_INT_ENA Write 1 to enable SPI_MST_TX_AFIFO_REMPTY_ERR_INT interrupt. (R/W)

LP_SPI_APP2_INT_ENA Write 1 to enable SPI_APP2_INT interrupt. (R/W)

LP_SPI_APP1_INT_ENA Write 1 to enable SPI_APP1_INT interrupt. (R/W)

Register 46.96. LP_SPI_DMA_INT_CLR_REG (0x0038)



LP_SPI_SLV_RD_BUF_DONE_INT_CLR Write 1 to clear SPI_SLV_RD_BUF_DONE_INT interrupt. (WT)

LP_SPI_SLV_WR_BUF_DONE_INT_CLR Write 1 to clear SPI_SLV_WR_BUF_DONE_INT interrupt. (WT)

LP_SPI_TRANS_DONE_INT_CLR Write 1 to clear SPI_TRANS_DONE_INT interrupt. (WT)

LP_SPI_WAKEUP_INT_CLR Write 1 to clear LP SPI_WAKEUP_INT interrupt. (WT)

LP_SPI_SLV_BUF_ADDR_ERR_INT_CLR Write 1 to clear SPI_SLV_BUF_ADDR_ERR_INT interrupt. (WT)

LP_SPI_SLV_CMD_ERR_INT_CLR Write 1 to clear SPI_SLV_CMD_ERR_INT interrupt. (WT)

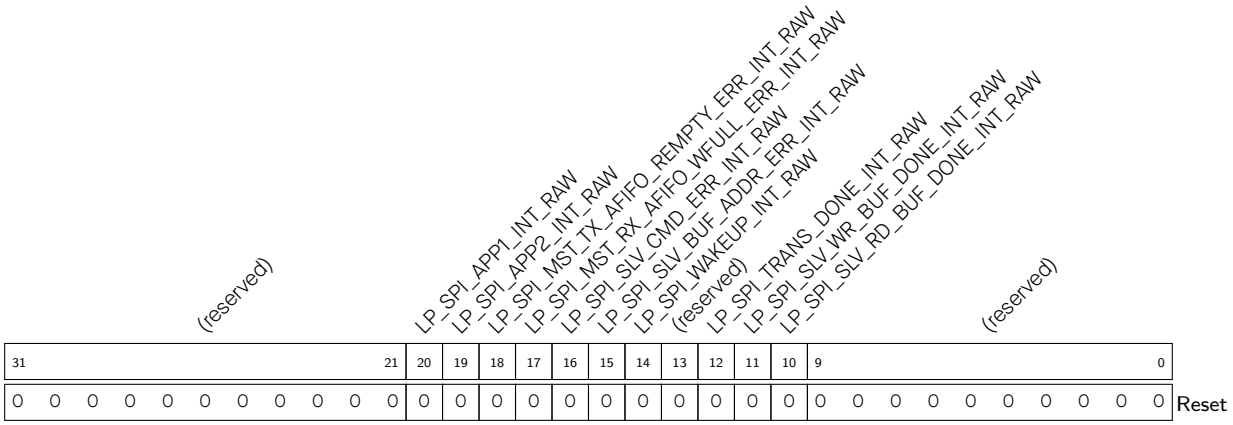
LP_SPI_MST_RX_AFIFO_WFULL_ERR_INT_CLR Write 1 to clear SPI_MST_RX_AFIFO_WFULL_ERR_INT interrupt. (WT)

LP_SPI_MST_TX_AFIFO_REMPTY_ERR_INT_CLR Write 1 to clear SPI_MST_TX_AFIFO_REMPTY_ERR_INT interrupt. (WT)

LP_SPI_APP2_INT_CLR Write 1 to clear SPI_APP2_INT interrupt. (WT)

LP_SPI_APP1_INT_CLR Write 1 to clear SPI_APP1_INT interrupt. (WT)

Register 46.97. LP_SPI_DMA_INT_RAW_REG (0x003C)



LP_SPI_SLV_RD_BUF_DONE_INT_RAW The raw interrupt status of SPI_SLV_RD_BUF_DONE_INT interrupt.
(R/WTC/SS)

LP_SPI_SLV_WR_BUF_DONE_INT_RAW The raw interrupt status of SPI_SLV_WR_BUF_DONE_INT interrupt.
(R/WTC/SS)

LP_SPI_TRANS_DONE_INT_RAW The raw interrupt status of SPI_TRANS_DONE_INT interrupt.
(R/WTC/SS)

LP_SPI_WAKEUP_INT_RAW The raw interrupt status of LP SPI_LP SPI_WAKEUP_INT interrupt.
(R/WTC/SS)

LP_SPI_SLV_BUF_ADDR_ERR_INT_RAW The raw interrupt status of SPI_SLV_BUF_ADDR_ERR_INT interrupt.
(R/WTC/SS)

LP_SPI_SLV_CMD_ERR_INT_RAW The raw interrupt status of SPI_SLV_CMD_ERR_INT interrupt.
(R/WTC/SS)

LP_SPI_MST_RX_AFIFO_WFULL_ERR_INT_RAW The raw interrupt status of SPI_MST_RX_AFIFO_WFULL_ERR_INT interrupt.
(R/WTC/SS)

LP_SPI_MST_TX_AFIFO_REMPTY_ERR_INT_RAW The raw interrupt status of SPI_MST_TX_AFIFO_REMPTY_ERR_INT interrupt.
(R/WTC/SS)

LP_SPI_APP2_INT_RAW The raw interrupt status of SPI_APP2_INT interrupt. The value is only controlled by the application.
(R/WTC/SS)

LP_SPI_APP1_INT_RAW The raw interrupt status of SPI_APP1_INT interrupt. The value is only controlled by the application.
(R/WTC/SS)

Register 46.98. LP_SPI_DMA_INT_ST_REG (0x0040)



LP_SPI_SLV_RD_BUF_DONE_INT_ST The interrupt status of SPI_SLV_RD_BUF_DONE_INT interrupt.
(RO)

LP_SPI_SLV_WR_BUF_DONE_INT_ST The interrupt status of SPI_SLV_WR_BUF_DONE_INT interrupt.
(RO)

LP_SPI_TRANS_DONE_INT_ST The interrupt status of SPI_TRANS_DONE_INT interrupt.
(RO)

LP_SPI_WAKEUP_INT_ST The interrupt status bit of LP_SPI_WAKEUP_INT interrupt.
(RO)

LP_SPI_SLV_BUF_ADDR_ERR_INT_ST The interrupt status bit of SPI_SLV_BUF_ADDR_ERR_INT interrupt.
(RO)

LP_SPI_SLV_CMD_ERR_INT_ST The interrupt status of SPI_SLV_CMD_ERR_INT interrupt.
(RO)

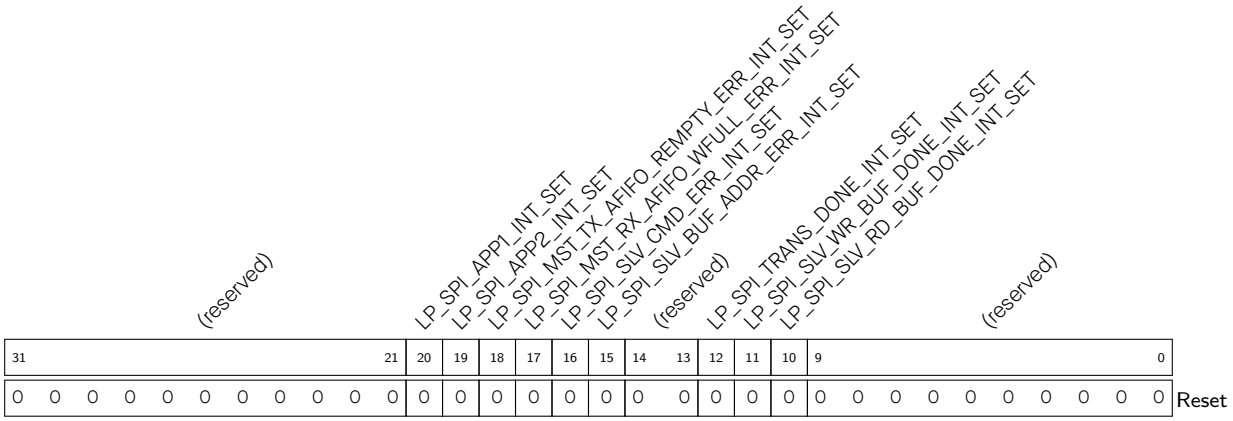
LP_SPI_MST_RX_AFIFO_WFULL_ERR_INT_ST The interrupt status of SPI_MST_RX_AFIFO_WFULL_ERR_INT interrupt.
(RO)

LP_SPI_MST_TX_AFIFO_REMPTY_ERR_INT_ST The interrupt status of SPI_MST_TX_AFIFO_REMPTY_ERR_INT interrupt.
(RO)

LP_SPI_APP2_INT_ST The interrupt status of SPI_APP2_INT interrupt.
(RO)

LP_SPI_APP1_INT_ST The interrupt status of SPI_APP1_INT interrupt.
(RO)

Register 46.99. LP_SPI_DMA_INT_SET_REG (0x004C)



LP_SPI_SLV_RD_BUF_DONE_INT_SET Write 1 to set SPI_SLV_RD_BUF_DONE_INT interrupt. (WT)

LP_SPI_SLV_WR_BUF_DONE_INT_SET Write 1 to set SPI_SLV_WR_BUF_DONE_INT interrupt. (WT)

LP_SPI_TRANS_DONE_INT_SET Write 1 to set SPI_TRANS_DONE_INT interrupt. (WT)

LP_SPI_SLV_BUF_ADDR_ERR_INT_SET Write 1 to set SPI_SLV_BUF_ADDR_ERR_INT interrupt. (WT)

LP_SPI_SLV_CMD_ERR_INT_SET Write 1 to set SPI_SLV_CMD_ERR_INT interrupt. (WT)

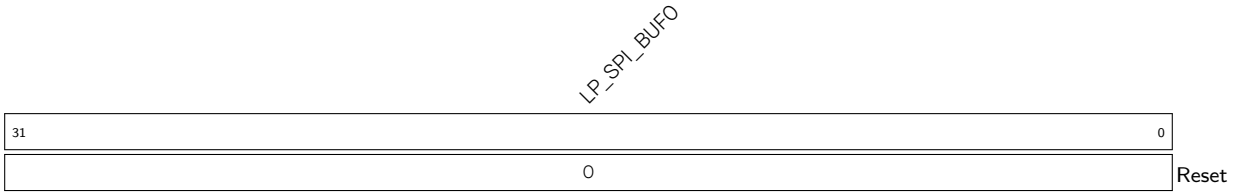
LP_SPI_MST_RX_AFIFO_WFULL_ERR_INT_SET Write 1 to set SPI_MST_RX_AFIFO_WFULL_ERR_INT interrupt. (WT)

LP_SPI_MST_TX_AFIFO_REMPTY_ERR_INT_SET Write 1 to set SPI_MST_TX_AFIFO_REMPTY_ERR_INT interrupt. (WT)

LP_SPI_APP2_INT_SET Write 1 to set SPI_APP2_INT interrupt. (WT)

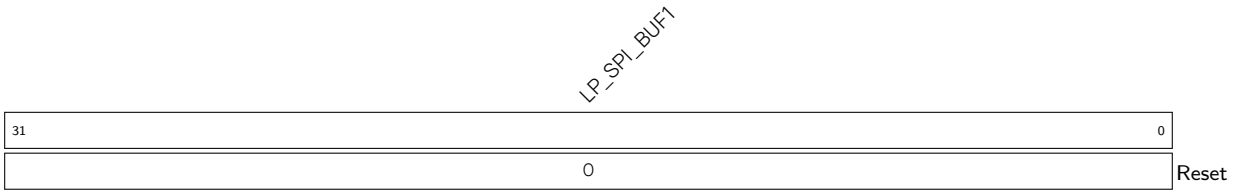
LP_SPI_APP1_INT_SET Write 1 to set SPI_APP1_INT interrupt. (WT)

Register 46.100. LP_SPI_W0_REG (0x0098)



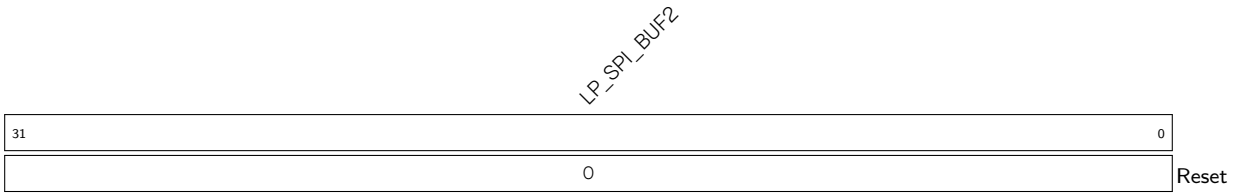
LP_SPI_BUF0 32-bit data buffer 0.
(R/W/SS)

Register 46.101. LP_SPI_W1_REG (0x009C)



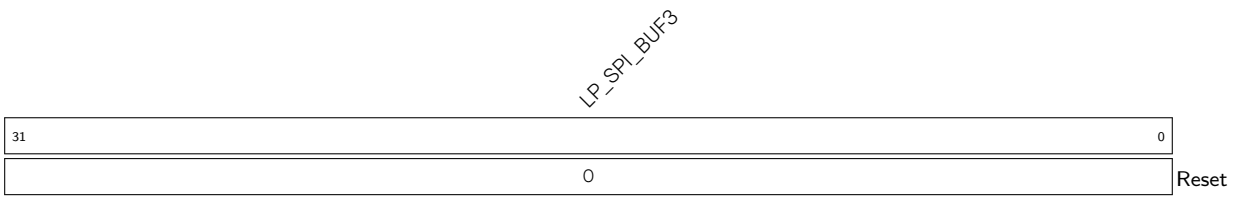
LP_SPI_BUF1 32-bit data buffer 1.
(R/W/SS)

Register 46.102. LP_SPI_W2_REG (0x00A0)



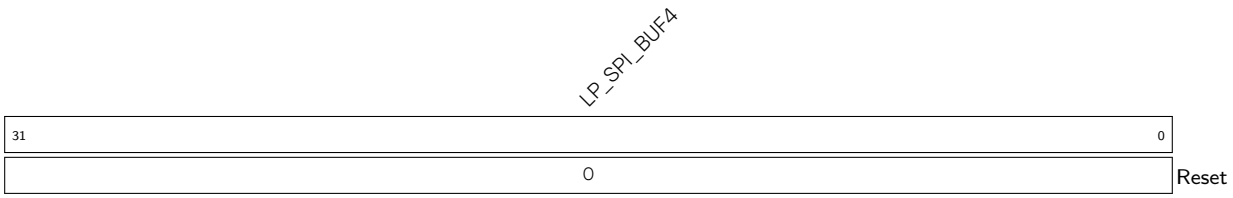
LP_SPI_BUF2 32-bit data buffer 2.
(R/W/SS)

Register 46.103. LP_SPI_W3_REG (0x00A4)



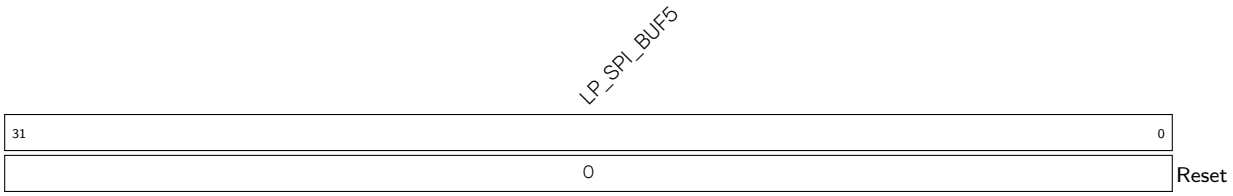
LP_SPI_BUF3 32-bit data buffer 3.
(R/W/SS)

Register 46.104. LP_SPI_W4_REG (0x00A8)



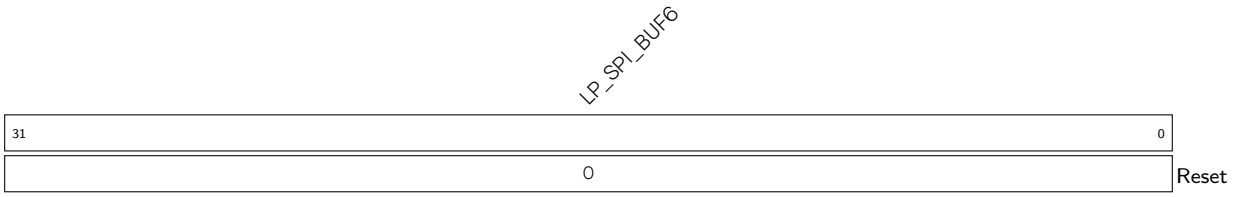
LP_SPI_BUF4 32-bit data buffer 4.
(R/W/SS)

Register 46.105. LP_SPI_W5_REG (0x00AC)



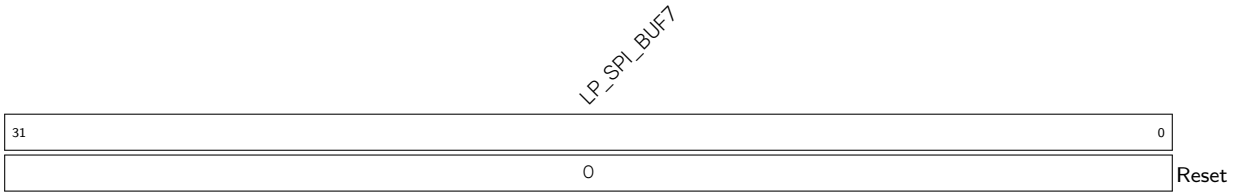
LP_SPI_BUF5 32-bit data buffer 5.
(R/W/SS)

Register 46.106. LP_SPI_W6_REG (0x00B0)



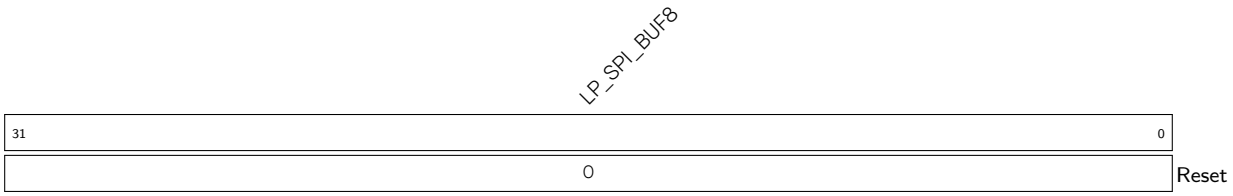
LP_SPI_BUF6 32-bit data buffer 6.
(R/W/SS)

Register 46.107. LP_SPI_W7_REG (0x00B4)



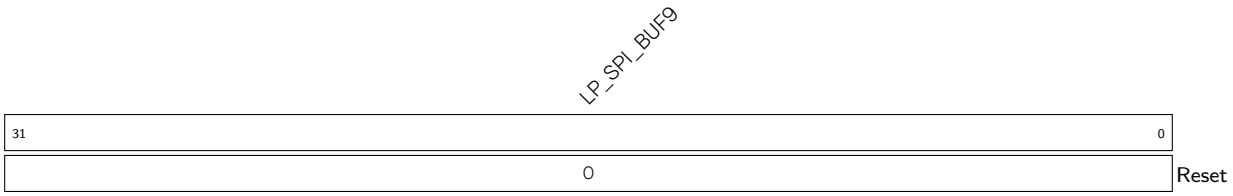
LP_SPI_BUF7 32-bit data buffer 7.
(R/W/SS)

Register 46.108. LP_SPI_W8_REG (0x00B8)



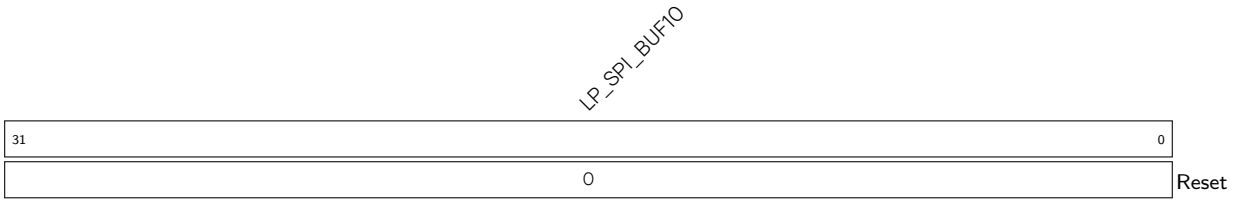
LP_SPI_BUF8 32-bit data buffer 8.
(R/W/SS)

Register 46.109. LP_SPI_W9_REG (0x00BC)



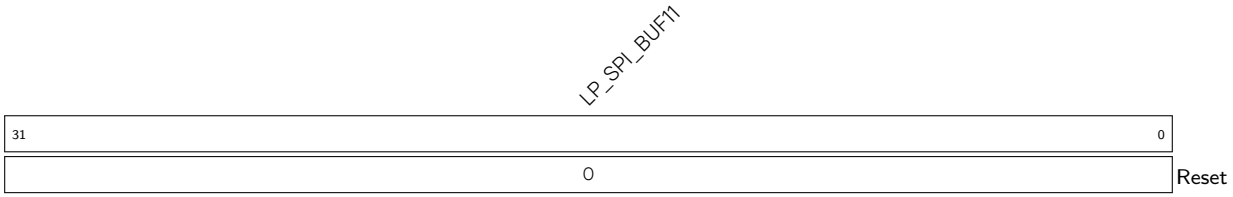
LP_SPI_BUF9 32-bit data buffer 9.
(R/W/SS)

Register 46.110. LP_SPI_W10_REG (0x00C0)



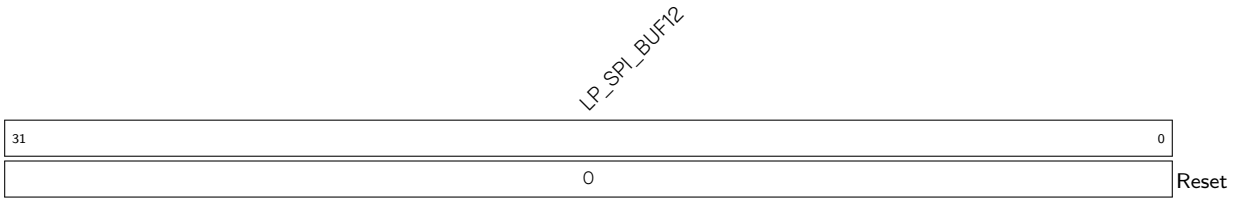
LP_SPI_BUF10 32-bit data buffer 10.
(R/W/SS)

Register 46.111. LP_SPI_W11_REG (0x00C4)



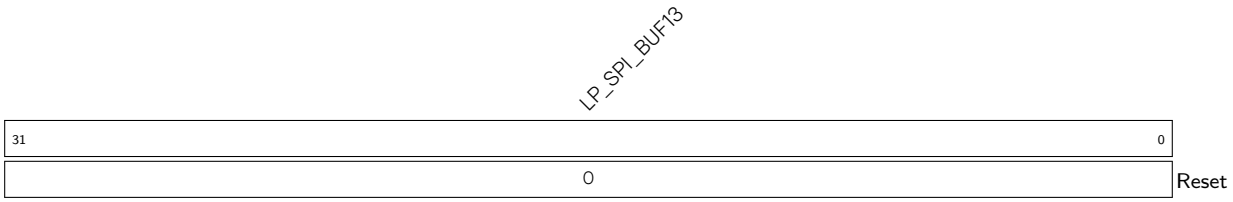
LP_SPI_BUF11 32-bit data buffer 11.
(R/W/SS)

Register 46.112. LP_SPI_W12_REG (0x00C8)



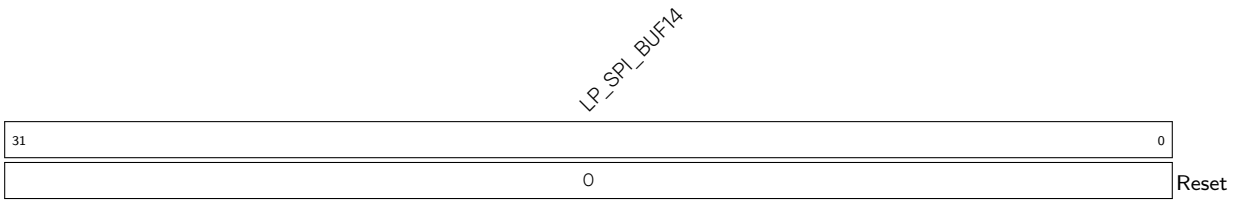
LP_SPI_BUF12 32-bit data buffer 12.
(R/W/SS)

Register 46.113. LP_SPI_W13_REG (0x00CC)



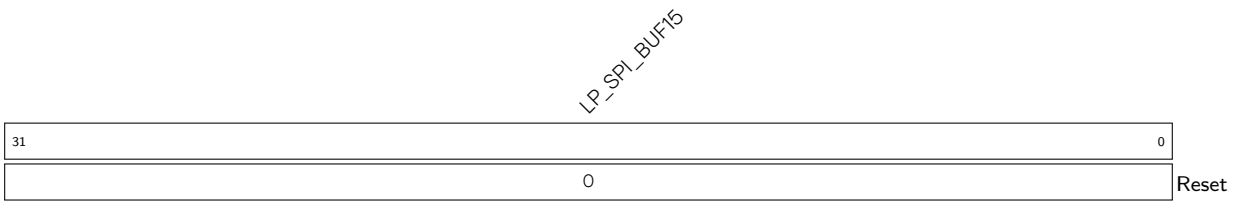
LP_SPI_BUF13 32-bit data buffer 13.
(R/W/SS)

Register 46.114. LP_SPI_W14_REG (0x00D0)



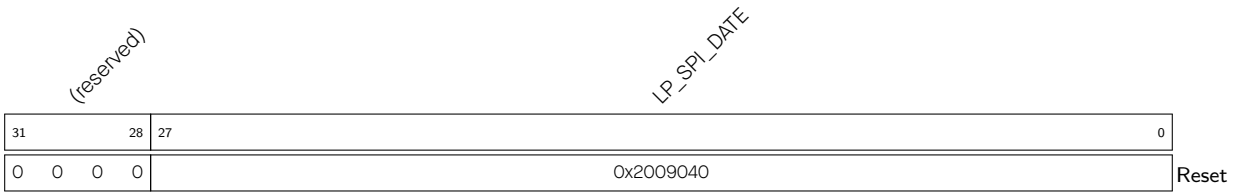
LP_SPI_BUF14 32-bit data buffer 14.
(R/W/SS)

Register 46.115. LP_SPI_W15_REG (0x00D4)



LP_SPI_BUF15 32-bit data buffer 15.
(R/W/SS)

Register 46.116. LP_SPI_DATE_REG (0x00F0)



LP_SPI_DATE Version control register.

(R/W)

Chapter 47

I2C Controller (I2C)

The I2C (Inter-Integrated Circuit) bus allows ESP32-P4 to communicate with multiple external devices. These external devices can share one I2C bus. ESP32-P4 has three I2C controllers: two in the main system and one in the low-power system. The two I2C controllers in the main system can act as a master or a slave (referred to as I2C below), while the one in the low-power system can only act as a master (referred to as LP_I2C below), which can still work when the main system sleeps.

47.1 Overview

The I2C bus has two lines, namely a serial data line (SDA) and a serial clock line (SCL). Both SDA and SCL lines are open-drain. The I2C bus can be connected to a single or multiple master devices and a single or multiple slave devices. However, only one master device can access a slave at a time via the bus.

The master initiates communication by generating a START condition: pulling the SDA line low while SCL is high. Then it issues nine clock pulses via SCL. The first eight pulses are used to transmit a 7-bit address followed by a read/write ($R/\overline{W}$) bit. If the address of an I2C slave matches the 7-bit address transmitted, this matching slave can respond by pulling SDA low on the ninth clock pulse. The master and the slave can send or receive data according to the $R/\overline{W}$ bit. Whether to terminate the data transfer or not is determined by the logic level of the acknowledge (ACK) bit. During data transfer, SDA changes only when SCL is low. Once the communication has finished, the master sends a STOP condition: pulling SDA up while SCL is high. If a master both reads and writes data in one transfer, then it should send a RSTART condition, a slave address, and a $R/\overline{W}$ bit before changing its operation. The RSTART condition is used to change the transfer direction and the mode of the devices (master mode or slave mode).

47.2 Features

The I2C controller of ESP32-P4 has the following features:

- Master mode and slave mode
- Communication between multiple masters and slaves
- Standard mode (100 Kbit/s)
- Fast mode (400 Kbit/s)
- 7-bit addressing and 10-bit addressing
- Continuous data transfer achieved by pulling SCL low in slave mode
- Programmable digital noise filtering
- Dual address mode, which uses slave address and slave memory or register address

47.3 I2C Architecture

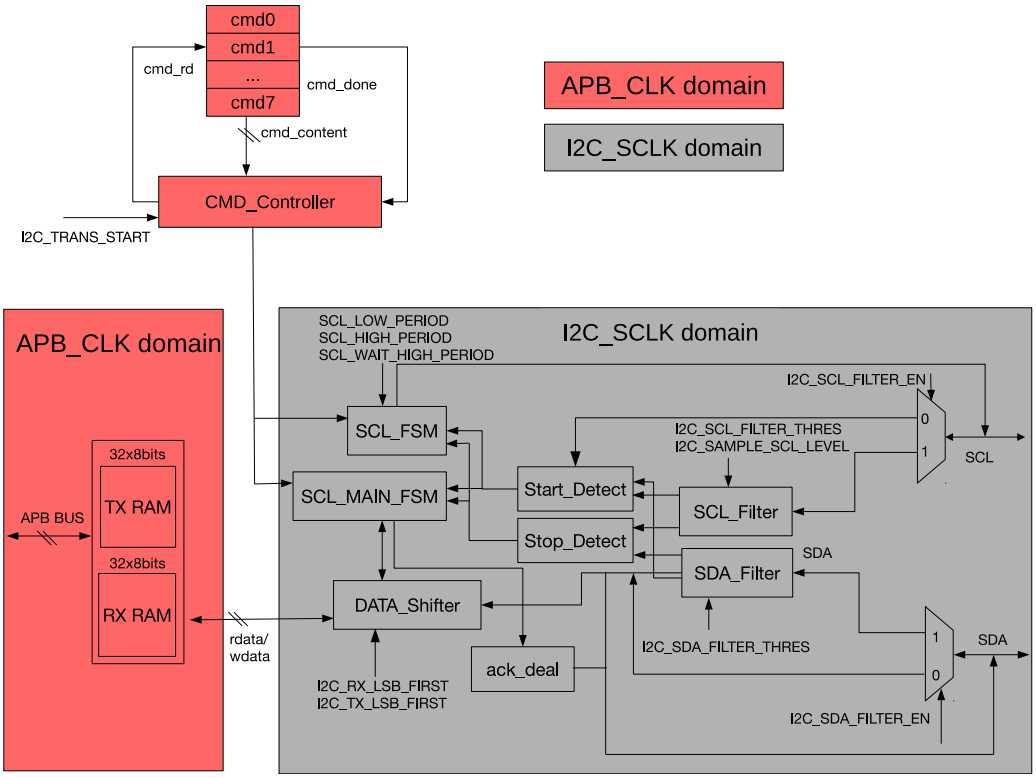


Figure 47.3-1. I2C Master Architecture

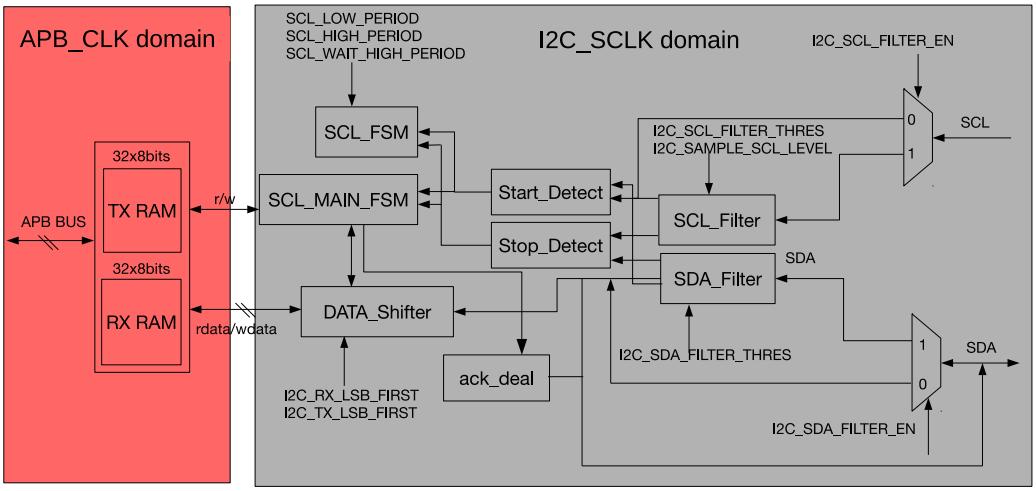


Figure 47.3-2. I2C Slave Architecture

The I2C controller runs either in master mode or slave mode, which is determined by I2C_MS_MODE. Figure 47.3-1 shows the architecture of a master, while Figure 47.3-2 shows that of a slave. The I2C controller has the following main parts:

- Transmit and receive memory (TX/RX RAM): store data to be transmitted and data received respectively.
- Command controller (CMD_Controller): generate (R)START, STOP, WRITE, READ, and END commands

- SCL clock controller (SCL_FSM): generate the timing sequence conforming to the I2C protocol. Figure 47.3-3 and Table 47.3-1 are the timing diagram and corresponding parameters of the I2C protocol.
- SDA data controller (SCL_MAIN_FSM): control the execution of I2C commands and the data sequence of the SDA line. It also controls the ACK_deal module to generate the ACK bit and detect the level of the ACK bit on the SDA line.
- Serial/parallel data converter (DATA_Shifter): shift data between serial and parallel form
- Filter for SCL (SCL_Filter): remove noises on SCL input signals
- Filter for SDA (SDA_Filter): remove noises on SDA input signals
- ACK bit controller (ack_deal): generate the ACK bit and detect the level of the ACK bit on the SDA line under the control of SCL_MAIN_FSM.

Besides, the I2C controller also has a clock module that generates I2C clocks, and a synchronization module that synchronizes the APB bus and the I2C controller.

The clock module is used to select clock sources, turn on and off clocks, and divide clocks. The synchronization module synchronizes signal transfer between different clock domains.

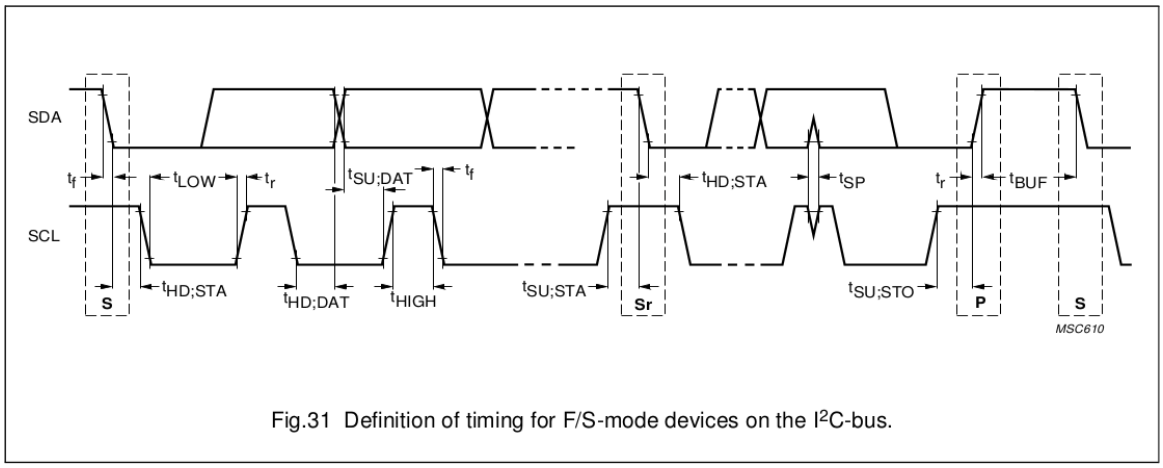


Fig.31 Definition of timing for F/S-mode devices on the I²C-bus.

Figure 47.3-3. I2C Protocol Timing (Cited from Fig.31 in The I2C-bus specification Version 2.1)

PARAMETER	SYMBOL	STANDARD-MODE		FAST-MODE		UNIT
		MIN.	MAX.	MIN.	MAX.	
SCL clock frequency	f_{SCL}	0	100	0	400	kHz
Hold time (repeated) START condition. After this period, the first clock pulse is generated	$t_{HD;STA}$	4.0	–	0.6	–	µs
LOW period of the SCL clock	t_{LOW}	4.7	–	1.3	–	µs
HIGH period of the SCL clock	t_{HIGH}	4.0	–	0.6	–	µs
Set-up time for a repeated START condition	$t_{SU;STA}$	4.7	–	0.6	–	µs
Data hold time: for CBUS compatible masters (see NOTE, Section 10.1.3) for I²C-bus devices	$t_{HD;DAT}$	 5.0 0(2)	 – 3.45(3)	 – 0(2)	 – 0.9(3)	 µs µs
Data set-up time	$t_{SU;DAT}$	250	–	100(4)	–	ns
Rise time of both SDA and SCL signals	t_r	–	1000	$20 + 0.1C_b$(5)	300	ns
Fall time of both SDA and SCL signals	t_f	–	300	$20 + 0.1C_b$(5)	300	ns
Set-up time for STOP condition	$t_{SU;STO}$	4.0	–	0.6	–	µs
Bus free time between a STOP and START condition	t_{BUF}	4.7	–	1.3	–	µs

Table 47.3-1. I2C Timing Parameters (Cited from Table 5 in The I2C-bus specification Version 2.1)

47.4 Functional Description

As mentioned above, one or more masters and one or more slaves can be connected on the I2C bus. The following sections describe the operations of the ESP32-P4 I2C controllers. Note that operations may differ between the I2C controllers in ESP32-P4 and other masters or slaves on the bus. Please refer to datasheets of individual I2C devices for specific information.

47.4.1 Clock Configuration

Registers, TX RAM, and RX RAM are configured and accessed in the APB_CLK clock domain. The main logic of the I2C controller, including SCL_FSM, SCL_MAIN_FSM, SCL_FILTER, SDA_FILTER, and DATA_SHIFTER, are in the I2C_SCLK clock domain.

You can configure the clock source for I2C_SCLK of I2C0 in the main system (HP) to FOSC_20M_CLK or XTAL_40M_CLK via HP_SYS_CLKRST_I2C0_CLK_SRC_SEL and configure that of I2C1 in the main system via HP_SYS_CLKRST_I2C1_CLK_SRC_SEL. For the clock source for I2C_SCLK of LP_I2C in the low-power system (LP), you can configure it to XTAL_D2_CLK, LP_DYN_FAST_CLK, or PLL_LP_CLK 8 MHz via LPPERI_LP_I2C_CLK_SEL.

The steps to configure the clock source for HP I2C0 are as follows:

- Enable the clock source for I2C_SCLK of I2C0 by configuring HP_SYS_CLKRST_I2C0_CLK_EN to 1.
- When HP_SYS_CLKRST_I2C0_SRC_SEL is 0, the clock source is XTAL_40M_CLK.
- When HP_SYS_CLKRST_I2C0_SRC_SEL is 1, the clock source is FOSC_20M_CLK.

The steps to configure the clock source for HP I2C1 are as follows:

- Enable the clock source for I2C_SCLK of I2C1 by configuring HP_SYS_CLKRST_I2C1_CLK_EN to 1.

- When HP_SYS_CLKRST_I2C1_SRC_SEL is 0, the clock source is XTAL_40M_CLK.
- When HP_SYS_CLKRST_I2C1_SRC_SEL is 1, the clock source is FOSC_20M_CLK.

The steps to configure the clock source for LP_I2C are as follows:

- Enable the clock source for I2C_SCLK of LP_I2C by configuring LPPERI_CK_EN_LP_I2C to 1.
- When LPPERI_LP_I2C_CLK_SEL is 0, the clock source is LP_DYN_FAST_CLK.
- When LPPERI_LP_I2C_CLK_SEL is 1, the clock source is XTAL_D2_CLK.
- When LPPERI_LP_I2C_CLK_SEL is 2, the clock source is PLL_LP_CLK 8 MHz.

The clock source then passes through a fractional divider to generate I2C_SCLK according to the following equation:

$$I2C_SCLK_DIV_NUM + 1 + \frac{I2C_SCLK_DIV_A}{I2C_SCLK_DIV_B}$$

In the equation, I2C_SCLK_DIV_NUM represents the integer part of the divisor, I2C_SCLK_DIV_A represents the numerator of the fractional part of the divisor, and I2C_SCLK_DIV_B represents the denominator of the fractional part of the divisor. Limited by timing parameters, the derived clock I2C_SCLK should operate at a frequency 20 times larger than SCL's frequency.
For I2C0:

- Configure I2C_SCLK_DIV_NUM via HP_SYS_CLKRST_I2C0_CLK_DIV_NUM.
- Configure I2C_SCLK_DIV_A via HP_SYS_CLKRST_I2C0_CLK_DIV_NUMERATOR.
- Configure I2C_SCLK_DIV_B via HP_SYS_CLKRST_I2C0_CLK_DIV_DENOMINATOR.

For I2C1:

- Configure I2C_SCLK_DIV_NUM via HP_SYS_CLKRST_I2C1_CLK_DIV_NUM.
- Configure I2C_SCLK_DIV_A via HP_SYS_CLKRST_I2C1_CLK_DIV_NUMERATOR.
- Configure I2C_SCLK_DIV_B via HP_SYS_CLKRST_I2C1_CLK_DIV_DENOMINATOR.

47.4.2 SCL and SDA Noise Filtering

SCL_Filter and SDA_Filter modules are identical and are used to filter signal noise on SCL and SDA, respectively. These filters can be enabled or disabled by configuring I2C_SCL_FILTER_EN and I2C_SDA_FILTER_EN.

Take SCL_Filter as an example. When enabled, SCL_Filter samples input signals on the SCL line continuously. These input signals are valid only if they remain unchanged for consecutive I2C_SCL_FILTER_THRES I2C_SCLK clock cycles. Given that only valid input signals can pass through the filter, SCL_Filter can remove glitches whose pulse width is shorter than I2C_SCL_FILTER_THRES I2C_SCLK clock cycles, while SDA_Filter can remove glitches whose pulse width is shorter than I2C_SDA_FILTER_THRES I2C_SCLK clock cycles.

47.4.3 SCL Clock Stretching

The I2C controller in slave mode (i.e., slave) can realize the function called clock stretching by holding the SCL line low to suspend data transmission in exchange for more time to process data. This function is enabled

by setting the I2C_SLAVE_SCL_STRETCH_EN bit. The time period to release the SCL line from stretching is configured by setting the I2C_STRETCH_PROTECT_NUM field, in order to avoid timing sequence errors. The slave can choose to achieve clock stretching by holding the SCL line low when one of the following four events occurs:

1. Address match: The address of the slave matches the address sent by the master via the SDA line, and the $R/\overline{W}$ bit is 1.
2. RAM being full: RX RAM of the slave is full. Note that when the slave receives less than the FIFO depth, which is 32 bytes in ESP32-P4 I2C, it is not necessary to enable clock stretching; when the slave receives FIFO depth bytes or more, you may interrupt data transmission to wrapped around RAM via the FIFO threshold, or enable clock stretching for more time to process data. When clock stretching is enabled, I2C_RX_FULL_ACK_LEVEL must be cleared, otherwise there will be unpredictable consequences.
3. RAM being empty: The slave is sending data, but its TX RAM is empty.
4. Sending an ACK: If I2C_SLAVE_BYTE_ACK_CTL_EN is set, the slave pulls SCL low when sending an ACK bit. At this stage, software validates data and configures I2C_SLAVE_BYTE_ACK_LVL to control the level of the ACK bit. Note that when RX RAM of the slave is full, the level of the ACK bit to be sent is determined by I2C_RX_FULL_ACK_LEVEL, instead of I2C_SLAVE_BYTE_ACK_LVL. In this case, I2C_RX_FULL_ACK_LEVEL should also be cleared to ensure proper functioning of clock stretching.

When clock stretching occurs, the cause of stretching can be read from the I2C_STRETCH_CAUSE bit. Clock stretching can be disabled by setting the I2C_SLAVE_SCL_STRETCH_CLR bit.

47.4.4 Generating SCL Pulses in Idle State

Usually, when the I2C bus is idle, the SCL line is held high. The I2C controller in ESP32-P4 can be programmed to generate SCL pulses in an idle state. This function only works when the I2C controller is configured as master. If the I2C_SCL_RST_SLV_EN bit is set, hardware will send I2C_SCL_RST_SLV_NUM SCL pulses, and then automatically clear the I2C_SCL_RST_SLV_EN bit.

47.4.5 Synchronization

I2C registers are configured in the APB_CLK domain, whereas the I2C controller is configured in the asynchronous I2C_SCLK domain. Therefore, before being used by the I2C controller, register values should be synchronized by first writing configuration registers and then writing 1 to I2C_CONF_UPGATE. Registers that need synchronization are listed in Table 47.4-1.

Table 47.4-1. I2C Synchronous Registers

Register	Field	Address
I2C_CTR_REG	I2C_SLV_TX_AUTO_START_EN	0x0004
	I2C_ADDR_10BIT_RW_CHECK_EN	
	I2C_ADDR_BROADCASTING_EN	
	I2C_SDA_FORCE_OUT	
	I2C_SCL_FORCE_OUT	
	I2C_SAMPLE_SCL_LEVEL	
	I2C_RX_FULL_ACK_LEVEL	
	I2C_MS_MODE	
	I2C_TX_LSB_FIRST	
	I2C_RX_LSB_FIRST	
	I2C_ARBITRATION_EN	
I2C_TO_REG	I2C_TIME_OUT_EN	0x000C
	I2C_TIME_OUT_VALUE	
I2C_SLAVE_ADDR_REG	I2C_ADDR_10BIT_EN	0x0010
	I2C_SLAVE_ADDR	
I2C_FIFO_CONF_REG	I2C_FIFO_ADDR_CFG_EN	0x0018
I2C_SCL_SP_CONF_REG	I2C_SDA_PD_EN	0x0080
	I2C_SCL_PD_EN	
	I2C_SCL_RST_SLV_NUM	
	I2C_SCL_RST_SLV_EN	
I2C_SCL_STRETCH_CONF_REG	I2C_SLAVE_BYTE_ACK_CTL_EN	0x0084
	I2C_SLAVE_BYTE_ACK_LVL	
	I2C_SLAVE_SCL_STRETCH_EN	
	I2C_STRETCH_PROTECT_NUM	
I2C_SCL_LOW_PERIOD_REG	I2C_SCL_LOW_PERIOD	0x0000
I2C_SCL_HIGH_PERIOD_REG	I2C_WAIT_HIGH_PERIOD	0x0038
	I2C_HIGH_PERIOD	
I2C_SDA_HOLD_REG	I2C_SDA_HOLD_TIME	0x0030
I2C_SDA_SAMPLE_REG	I2C_SDA_SAMPLE_TIME	0x0034
I2C_SCL_START_HOLD_REG	I2C_SCL_START_HOLD_TIME	0x0040
I2C_SCL_RSTART_SETUP_REG	I2C_SCL_RSTART_SETUP_TIME	0x0044
I2C_SCL_STOP_HOLD_REG	I2C_SCL_STOP_HOLD_TIME	0x0048
I2C_SCL_STOP_SETUP_REG	I2C_SCL_STOP_SETUP_TIME	0x004C
I2C_SCL_ST_TIME_OUT_REG	I2C_SCL_ST_TO_I2C	0x0078
I2C_SCL_MAIN_ST_TIME_OUT_REG	I2C_SCL_MAIN_ST_TO_I2C	0x007C
I2C_FILTER_CFG_REG	I2C_SCL_FILTER_EN	0x0050
	I2C_SCL_FILTER_THRES	
	I2C_SDA_FILTER_EN	
	I2C_SDA_FILTER_THRES	

47.4.6 Open-Drain Output

SCL and SDA output drivers must be configured as open-drain. There are two ways to achieve this:

1. Set I2C_SCL_FORCE_OUT and I2C_SDA_FORCE_OUT, and configure GPIO_PIN*n*_PAD_DRIVER for corresponding SCL and SDA pads as open-drain.
2. Clear I2C_SCL_FORCE_OUT and I2C_SDA_FORCE_OUT.

Because these lines are configured as open-drain, the low-to-high transition time of each line is longer, determined together by the pull-up resistor and line capacitance. The output duty cycle of I2C is limited by the SDA and SCL line's pull-up speed, mainly SCL's speed.

In addition, when I2C_SCL_FORCE_OUT and I2C_SCL_PD_EN are set to 1, SCL can be forced low; when I2C_SDA_FORCE_OUT and I2C_SDA_PD_EN are set to 1, SDA can be forced low.

47.4.7 Timing Parameter Configuration

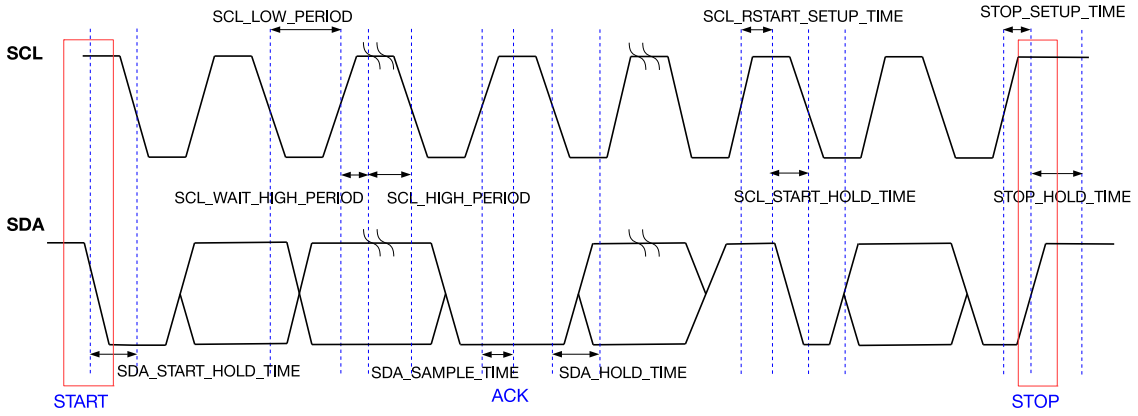


Figure 47.4-1. I2C Timing Diagram

Figure 47.4-1 shows the timing diagram of an I2C master. This figure also specifies registers used to configure the START bit, STOP bit, data hold time, data sample time, waiting time on the rising SCL edge, etc. Timing parameters are calculated as follows in I2C_SCLK clock cycles:

1. $t_{LOW} = (I2C_SCL_LOW_PERIOD + 1) \cdot T_{I2C_SCLK}$
2. $t_{HIGH} = (I2C_SCL_HIGH_PERIOD + 1) \cdot T_{I2C_SCLK}$
3. $t_{SU:STA} = (I2C_SCL_RSTART_SETUP_TIME + 1) \cdot T_{I2C_SCLK}$
4. $t_{HD:STA} = (I2C_SCL_START_HOLD_TIME + 1) \cdot T_{I2C_SCLK}$
5. $t_{r} = (I2C_SCL_WAIT_HIGH_PERIOD + 1) \cdot T_{I2C_SCLK}$
6. $t_{SU:STO} = (I2C_SCL_STOP_SETUP_TIME + 1) \cdot T_{I2C_SCLK}$
7. $t_{BUF} = (I2C_SCL_STOP_HOLD_TIME + 1) \cdot T_{I2C_SCLK}$
8. $t_{HD:DAT} = (I2C_SDA_HOLD_TIME + 1) \cdot T_{I2C_SCLK}$
9. $t_{SU:DAT} = (I2C_SCL_LOW_PERIOD - I2C_SDA_HOLD_TIME) \cdot T_{I2C_SCLK}$

Timing registers below are divided into two groups, depending on the mode in which these registers are active:

- Master mode only:

1. I2C_SCL_START_HOLD_TIME: Specifies the interval between the moment SDA is pulled low and the moment SCL is pulled low when the master generates a START condition. This interval is (I2C_SCL_START_HOLD_TIME +1) in I2C_SCLK cycles. This register is active only when the I2C controller works in master mode.
2. I2C_SCL_LOW_PERIOD: Specifies the low period of SCL. This period lasts (I2C_SCL_LOW_PERIOD +1) in I2C_SCLK cycles. This register is active only when the I2C controller works in master mode.

   However, this period could be extended in the following scenarios:

   - SCL is pulled low by peripheral devices when I2C acts as a master.
   - SCL is pulled low by an END command executed by the I2C controller.
   - SCL is pulled low by clock stretching when I2C acts as a slave.
3. I2C_SCL_WAIT_HIGH_PERIOD: Specifies the time for SCL to switch from low to high in I2C_SCLK cycles. Please make sure that SCL can be pulled high within this time period. Otherwise, the high period of SCL may be incorrect. This register is active only when the I2C controller works in master mode.
4. I2C_SCL_HIGH_PERIOD: Specifies the high period of SCL in I2C_SCLK cycles. This register is active only when the I2C controller works in master mode. When SCL goes high within (I2C_SCL_WAIT_HIGH_PERIOD + 1) in I2C_SCLK cycles, its frequency is:

   $$f_{scl} = \frac{f_{\text{I2C_SCLK}}}{\text{I2C_SCL_LOW_PERIOD} + \text{I2C_SCL_HIGH_PERIOD} + \text{I2C_SCL_WAIT_HIGH_PERIOD} + 3 + \text{I2C_SCL_FILTER_THRES}}$$

   where 3 represents the amount of clock cycles required to synchronize the SCL. If the SCL filtering function is turned on, the delay caused by I2C_SCL_FILTER_THRES needs to be added. As the SCL low-to-high transition time represented by I2C_SCL_WAIT_HIGH_PERIOD + 1 module clock can be affected by the pull-up resistor, IO drive capability, SCL line capacitance, etc., deviation may occur between the actual frequency of the test and the theoretical frequency. At this point, deviations can be reduced by adjusting the value of I2C_SCL_WAIT_HIGH_PERIOD.

- Master mode and slave mode:
  1. I2C_SDA_SAMPLE_TIME: Specifies the interval between the rising edge of SCL and the level sampling time of SDA. It is advised to set a value in the middle of SCL's high period, so as to correctly sample the level of SCL. This register is active both in master mode and slave mode.
  2. I2C_SDA_HOLD_TIME: Specifies the interval between changing the SDA output level and the falling edge of SCL. This register is active both in master mode and slave mode.

Timing parameters limits corresponding register configuration.

1. $\frac{f_{I2C_SCLK}}{f_{SCL}} > 20$
2. $3 \times f_{I2C_SCLK} \leq (I2C_SDA_HOLD_TIME - 4) \times f_{APB_CLK}$
3. I2C_SDA_HOLD_TIME + I2C_SCL_START_HOLD_TIME > SDA_FILTER_THRES + 3
4. I2C_SCL_WAIT_HIGH_PERIOD < I2C_SDA_SAMPLE_TIME < I2C_SCL_HIGH_PERIOD
5. I2C_SDA_SAMPLE_TIME < I2C_SCL_WAIT_HIGH_PERIOD + I2C_SCL_START_HOLD_TIME + I2C_SCL_RSTART_SETUP_TIME
6. I2C_STRETCH_PROTECT_NUM + I2C_SDA_HOLD_TIME > I2C_SCL_LOW_PERIOD

47.4.8 Timeout Control

The I2C controller has three types of timeout control, namely timeout control for SCL_FSM, for SCL_MAIN_FSM, and for the SCL line. The first two are always enabled, while the third is configurable.

When SCL_FSM remains unchanged for more than $2^{I2C_SCL_ST_TO_I2C+1}$ –1 clock cycles, an I2C_SCL_ST_TO_INT interrupt is triggered, and then SCL_FSM goes to idle state. The value of I2C_SCL_ST_TO_I2C should be less than or equal to 23, which means SCL_FSM could remain unchanged for 2^{24} –1 I2C_SCLK clock cycles at most before the interrupt is generated.

When SCL_MAIN_FSM remains unchanged for more than $2^{I2C_SCL_MAIN_ST_TO_I2C+1}$ –1 I2C_SCLK clock cycles, an I2C_SCL_MAIN_ST_TO_INT interrupt is triggered, and then SCL_MAIN_FSM goes to idle state. The value of I2C_SCL_MAIN_ST_TO_I2C should be less than or equal to 23, which means SCL_MAIN_FSM could remain unchanged for 2^{24} –1 clock cycles at most before the interrupt is generated.

Timeout control for SCL is enabled by setting I2C_TIME_OUT_EN. When the level of SCL remains unchanged for more than $2^{I2C_TIME_OUT_VALUE+1}$ –1 clock cycles, an I2C_TIME_OUT_INT interrupt is triggered, and then the I2C bus goes to idle state.

47.4.9 Command Configuration

When the I2C controller works in master mode, CMD_Controller reads commands from 8 sequential command registers and controls SCL_FSM and SCL_MAIN_FSM accordingly.

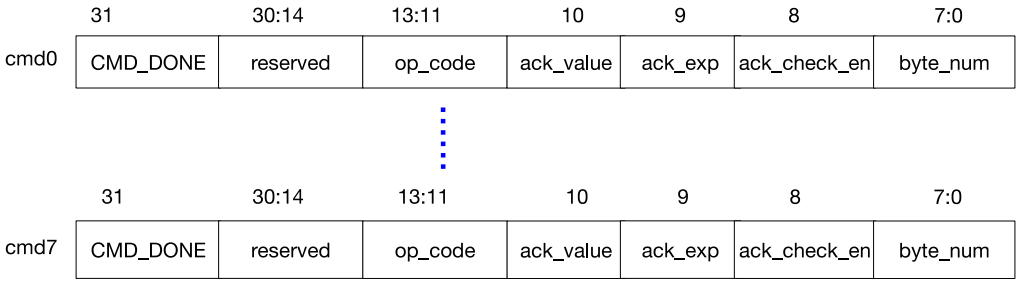


Figure 47.4-2. Structure of I2C Command Registers

Command registers, whose structure is illustrated in Figure 47.4-2, are active only when the I2C controller works in master mode. Fields of command registers are:

1. CMD_DONE: Indicates that a command has been executed. After each command has been executed, the CMD_DONE bit in the corresponding command register is set to 1 by hardware. By reading this bit, the software can tell if the command has been executed. When writing new commands, this bit must be cleared by software.
2. op_code: Indicates the command. The I2C controller supports five commands:
   - WRITE: op_code = 1. The I2C controller sends a slave address, a register address (only in dual address mode), and data to the slave.
   - STOP: op_code = 2. The I2C controller sends a STOP bit defined by the I2C protocol. This code also indicates that the command sequence has been executed, and the CMD_Controller stops reading commands. After restarted by software, the CMD_Controller resumes reading commands from command register 0.

- READ: op_code = 3. The I2C controller reads data from the slave.
- END: op_code = 4. The I2C controller pulls the SCL line down and suspends I2C communication. This code also indicates that the command sequence has been completed, and the CMD_Controller stops executing commands. Once the software refreshes data in command registers and the RAM, the CMD_Controller can be restarted to execute commands from command register 0 again.
- RSTART: op_code = 6. The I2C controller sends a START bit or a RSTART bit defined by the I2C protocol.

3. ack_value: Used to configure the level of the ACK bit sent by the I2C controller during a read operation. This bit is ignored in RSTART, STOP, END, and WRITE conditions.
4. ack_exp: Used to configure the level of the ACK bit expected by the I2C controller during a write operation. This bit is ignored during RSTART, STOP, END, and READ conditions.
5. ack_check_en: Used to enable the I2C controller during a write operation to check whether the ACK level sent by the slave matches ack_exp in the command. If this bit is set and the level received does not match ack_exp in the WRITE command, the master will generate an I2C_NACK_INT interrupt and a STOP condition for data transfer. If this bit is cleared, the controller will not check the ACK level sent by the slave. This bit is ignored during RSTART, STOP, END, and READ conditions.
6. byte_num: Specifies the length of data (in bytes) to be read or written. It can range from 1 to 255 bytes. This bit is ignored during RSTART, STOP, and END conditions.

Each command sequence is executed starting from command register 0 and terminated by a STOP or an END. Therefore, there must be a STOP or an END command in the eight command registers.

A complete data transfer on the I2C bus should be initiated by a START and terminated by a STOP. The transfer process may be completed using multiple sequences, separated by END commands. Each sequence may differ in the direction of data transfer, clock frequency, slave addresses, data length, etc. This allows efficient use of available peripheral RAM and also achieves more flexible I2C communication.

47.4.10 TX/RX RAM Data Storage

Both TX RAM and RX RAM are 32 × 8 bits and can be accessed in FIFO or non-FIFO mode. If I2C_NONFIFO_EN bit is cleared, both RAMs are accessed in FIFO mode; if I2C_NONFIFO_EN bit is set, both RAMs are accessed in non-FIFO mode.

TX RAM stores data that the I2C controller needs to send. During communication, when the I2C controller needs to send data (except acknowledgment bits), it reads data from TX RAM and sends them sequentially via SDA. When the I2C controller works in master mode, all data must be stored in TX RAM in the order they need to be sent to slaves. The data stored in TX RAM include slave addresses, read/write bits, register addresses (only in dual address mode) and data to be sent. When the I2C controller works in slave mode, TX RAM only stores data to be sent.

TX RAM can be read and written by the CPU. The CPU writes to TX RAM either in FIFO mode or in non-FIFO mode (direct address). In FIFO mode, the CPU writes to TX RAM via the fixed address I2C_DATA_REG, with addresses for writing in TX RAM incremented automatically by hardware. In non-FIFO mode, the CPU accesses TX RAM directly via address fields (I2C Base Address + 0x100) ~(I2C Base Address + 0x17C). Each byte in TX RAM occupies an entire word in the address space. Therefore, the address of the first byte is I2C Base

Address + 0x100, the second byte is I2C Base Address + 0x104, the third byte is I2C Base Address + 0x108, and so on. The CPU can only read TX RAM via direct addresses. Bytes written to the TX RAM can be read back by the CPU, via the direct addresses. Addresses for reading TX RAM are the same as addresses for writing TX RAM.

RX RAM stores data the I2C controller receives during communication. When the I2C controller works in slave mode, neither slave addresses sent by the master nor register addresses (only in dual address mode) will be stored in RX RAM. Values of RX RAM can be read by software after I2C communication is completed.

RX RAM can only be read by the CPU. The CPU reads RX RAM either in FIFO mode or in non-FIFO mode (direct address). In FIFO mode, the CPU reads RX RAM via the fixed address I2C_DATA_REG, with addresses for reading RX RAM incremented automatically by hardware. In non-FIFO mode, the CPU accesses RX RAM directly via address fields (I2C Base Address + 0x180) ~(I2C Base Address + 0x1FC). Each byte in RX RAM occupies an entire word in the address space. Therefore, the address of the first byte is I2C Base Address + 0x180, the second byte is I2C Base Address + 0x184, the third byte is I2C Base Address + 0x188 and so on.

In FIFO mode, the TX RAM of a master may wrap around to send data larger than the FIFO depth. Set I2C_FIFO_PRT_EN. If the size of data to be sent is smaller than I2C_TXFIFO_WM_THRHD (master), an I2C_TXFIFO_WM_INT (master) interrupt is generated. After receiving the interrupt, the software continues writing to I2C_DATA_REG (master). Please ensure that software writes to or refreshes TX RAM before the master sends data, otherwise it may result in unpredictable consequences.

In FIFO mode, the RX RAM of a slave may also wrap around to receive data larger than the FIFO depth. Set I2C_FIFO_PRT_EN and clear I2C_RX_FULL_ACK_LEVEL. If data already received (to be overwritten) is larger than I2C_RXFIFO_WM_THRHD (slave), an I2C_RXFIFO_WM_INT (slave) interrupt is generated. After receiving the interrupt, the software continues reading from I2C_DATA_REG (slave).

47.4.11 Data Conversion

DATA_Shifter is used for serial/parallel conversion, converting byte data in TX RAM to an outgoing serial bitstream or an incoming serial bitstream to byte data in RX RAM. I2C_RX_LSB_FIRST and I2C_TX_LSB_FIRST can be used to select LSB- or MSB-first storage and transmission of data.

47.4.12 Addressing Mode

The ESP32-P4 I2C controller supports 7-bit and 10-bit addressing. 10-bit addressing can be mixed with 7-bit addressing. Besides, the ESP32-P4 I2C controller also supports dual address mode.

Define the slave address as SLV_ADDR. In 7-bit addressing mode, the slave address is SLV_ADDR[6:0]; in 10-bit addressing mode, the slave address is SLV_ADDR[9:0].

In 7-bit addressing mode, the master only needs to send one byte of the address, which comprises SLV_ADDR[6:0] and a $R/\overline{W}$ bit. In the 7-bit addressing mode, there is a special case called general call addressing (broadcast). It is enabled by setting I2C_ADDR_BROADCASTING_EN in a slave. When the slave receives the general call address (0x00) from the master and the $R/\overline{W}$ bit followed is 0, it responds to the master regardless of its own address.

In 10-bit addressing mode, the master needs to send two bytes of address. The first byte is slave_addr_first_7bits followed by a $R/\overline{W}$ bit, and slave_addr_first_7bits should be configured as (0x78 | SLV_ADDR[9:8]). The second byte is slave_addr_second_byte, which should be configured as SLV_ADDR[7:0].

The slave can enable 10-bit addressing by configuring I2C_ADDR_10BIT_EN. I2C_SLAVE_ADDR is used to configure I2C slave address. Specifically, I2C_SLAVE_ADDR[14:7] should be configured as SLV_ADDR[7:0], and I2C_SLAVE_ADDR[6:0] should be configured as (0x78 | SLV_ADDR[9:8]). Since a 10-bit slave address has one more byte than a 7-bit address, byte_num of the WRITE command and the number of bytes in the RAM increase by one. Please refer to Programming Example for detailed descriptions.

When working in slave mode, the I2C controller supports dual address mode, where the first address is the address of an I2C slave, and the second one is the slave's memory address. When using dual address mode, RAM must be accessed in non-FIFO mode. Dual address mode is enabled by setting I2C_FIFO_ADDR_CFG_EN. When the slave address received by the slave is inconsistent with the internally configured slave address, the I2C_SLAVE_ADDR_UNMATCH interrupt will be generated.

47.4.13 $R/\overline{W}$ Bit Check in 10-bit Addressing Mode

In 10-bit addressing mode, when I2C_ADDR_10BIT_RW_CHECK_EN is set to 1, the I2C controller performs a check on the first byte, which consists of slave_addr_first_7bits and a $R/\overline{W}$ bit. When the $R/\overline{W}$ bit does not indicate a WRITE operation, i.e., not in line with the I2C protocol, the data transfer ends. If the check feature is not enabled, when the $R/\overline{W}$ bit does not indicate a WRITE, the data transfer still continues, but transfer failure may occur.

47.4.14 To Start the I2C Controller

To start the I2C controller in master mode, after configuring the controller to master mode and command registers, write 1 to I2C_TRANS_START in order to let the master start to parse and execute command sequences. The master always executes a command sequence starting from the command register 0 to a STOP or an END. To execute another command sequence starting from command register 0, refresh commands by writing 1 again to I2C_TRANS_START.

There are two ways to start the I2C controller in slave mode:

- **Automatic Start Transmission:** When the register I2C_SLV_TX_AUTO_START_EN is set, the slave automatically starts transmission after being addressed by the master. This mode is suitable when the master needs to continuously access slave data.
- **Manual Start Transmission:** When the register I2C_SLV_TX_AUTO_START_EN is cleared, the slave must explicitly set I2C_TRANS_START before each transmission. This mode is suitable when the master needs to selectively read slave data.

  For example:

  1. The slave prepares 5 bytes of data;
  2. The master reads only the first 3 bytes;
  3. After the master finishes reading, the slave sets I2C_TX_FIFO_RST to clear the remaining data in TXFIFO, and then sets I2C_TRANS_START to reinitialize the slave state.

Notes: In manual start mode, I2C_TRANS_START must be set before the next address match. If the master addresses the slave while I2C_TRANS_START is not set, the slave may acknowledge the address but fail to progress its internal state machines, which can lead to undefined behavior or timing disorder in the slave controller.

47.5 Functional Differences Between LP_I2C and I2C

LP_I2C can be used as a master to communicate with external devices when the main system sleeps. LP_I2C includes all the functions of the ESP32-P4 $I2C_{master}$, but doesn't include any functions of ESP32-P4 $I2C_{slave}$. It does not contain any registers related to the $I2C_{slave}$. For detailed register list, see 47.8.2 LP_I2C Register Summary.

The design differences between LP_I2C and I2C master are as follows:

- The size of TX/RX RAM in LP_I2C is 16*8 bit, which means the TX/RX FIFO depth is 16 bytes.
- The clock source of APB_CLK in LP_I2C is CLK_AON_FAST. The clock source for I2C_SCLK is configured via LPPERI_LP_I2C_CLK_SEL. The corresponding bits are:
  - 0: CLK_ROOT_FAST
  - 1: CLK_XTALD2
  - 2: PLL_F8M_CLK

  Configuring LPPERI_CK_EN_LP_I2C to 1 enables the clock source of I2C_SCLK. Adjust the timing registers accordingly when the clock frequency changes.

See the programming examples of ESP32-P4 $I2C_{slave}$ in 47.6 for that of LP_I2C.

47.6 Programming Example

This section provides programming examples for typical communication scenarios. ESP32-P4 has two I2C controllers. For the convenience of description, I2C masters and slaves in all subsequent figures are ESP32-P4 I2C controllers. I2C master is referred to as $I2C_{master}$, and I2C slave is referred to as $I2C_{slave}$.

47.6.1 $I2C_{master}$ Writes to $I2C_{slave}$ with a 7-bit Address in One Command Sequence

47.6.1.1 Introduction

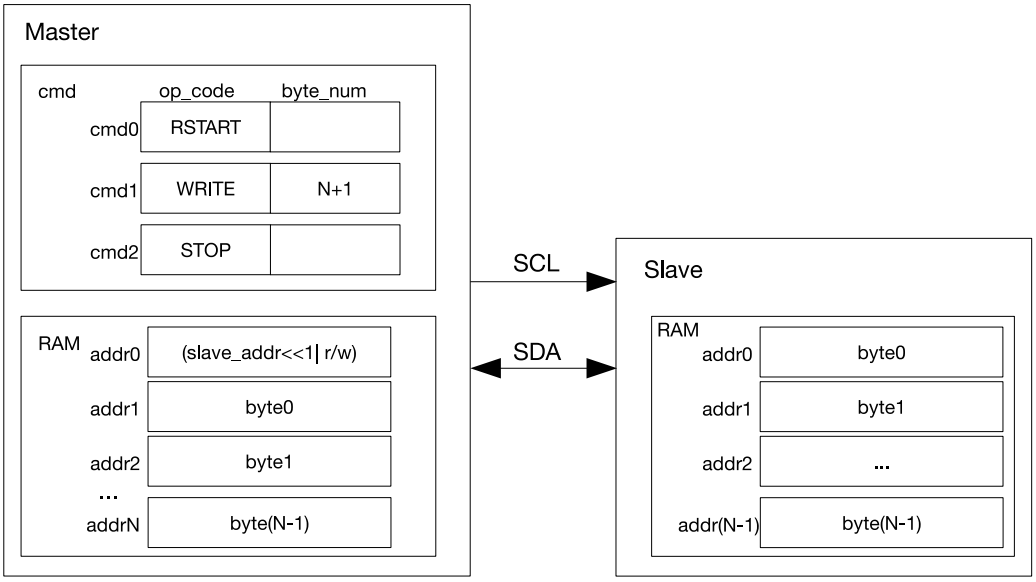


Figure 47.6-1. $I2C_{master}$ Writing to $I2C_{slave}$ with a 7-bit Address

Figure 47.6-1 shows how $I2C_{master}$ writes N bytes of data to $I2C_{slave}$ registers or RAM using 7-bit addressing. As shown in figure 47.6-1, the first byte in the RAM of $I2C_{master}$ is a 7-bit $I2C_{slave}$ address followed by a $R/\overline{W}$ bit. When the $R/\overline{W}$ bit is 0, it indicates a WRITE operation. The remaining bytes are used to store data ready for transfer. The cmd box contains related command sequences.

After the command sequence is configured and data in RAM is ready, $I2C_{master}$ enables the controller and initiates data transfer by setting the I2C_TRANS_START bit. The controller has four steps to take:

1. Wait for SCL to go high, to avoid SCL being used by other masters or slaves.
2. Execute a RSTART command by sending a START bit.
3. Execute a WRITE command by taking N+1 bytes from the RAM in order and sending them to $I2C_{slave}$ in the same order. The first byte is the address of $I2C_{slave}$.
4. Execute a STOP command. Once the $I2C_{master}$ transfers a STOP bit, an I2C_TRANS_COMPLETE_INT interrupt is generated.

47.6.1.2 Configuration Example

1. Configure the timing parameter registers of $I2C_{master}$ and $I2C_{slave}$ according to Section 47.4.7.
2. Set I2C_MS_MODE (master) to 1, and I2C_MS_MODE (slave) to 0.
3. Write 1 to I2C_CONF_UPGATE (master) and I2C_CONF_UPGATE (slave) to synchronize registers.
4. Configure command registers of $I2C_{master}$.

Command register	op_code	ack_value	ack_exp	ack_check_er	byte_num
I2C_COMMAND0 (master)	RSTART	—	—	—	—
I2C_COMMAND1 (master)	WRITE	—	ack_exp	1	N+1

I2C_COMMAND2 (master)	STOP	—	—	—	—

5. Write the address of $I2C_{slave}$ and data to be sent to TX RAM of $I2C_{master}$ in either FIFO mode or non-FIFO mode according to Section 47.4.10.
6. Write the address of $I2C_{slave}$ to I2C_SLAVE_ADDR (slave) in I2C_SLAVE_ADDR_REG (slave) register.
7. Write 1 to I2C_CONF_UPGATE (master) and I2C_CONF_UPGATE (slave) to synchronize registers.
8. Write 1 to I2C_TRANS_START (master) and I2C_TRANS_START (slave) to start transfer.
9. $I2C_{slave}$ compares the slave address sent by $I2C_{master}$ with its own address in I2C_SLAVE_ADDR (slave). When ack_check_en (master) in $I2C_{master}$'s WRITE command is 1, $I2C_{master}$ checks ACK value each time it sends a byte. When ack_check_en (master) is 0, $I2C_{master}$ does not check the ACK value and take $I2C_{slave}$ as a matching slave by default.
   - Match: If the received ACK value matches ack_exp (master) (the expected ACK value), $I2C_{master}$ continues data transfer.
   - Not match: If the received ACK value does not match ack_exp, $I2C_{master}$ generates an I2C_NACK_INT (master) interrupt and stops data transfer.
10. $I2C_{master}$ sends data, and determines whether to check ACK value according to ack_check_en (master).
11. If data to be sent (N) is larger than TX FIFO depth, TX RAM of $I2C_{master}$ may wrap around in FIFO mode. For details, please refer to Section 47.4.10.
12. If data to be received (N) is larger than RX FIFO depth, RX RAM of $I2C_{slave}$ may wrap around in FIFO mode. For details, please refer to Section 47.4.10.

    If data to be received (N) is larger than RX FIFO depth, the other way is to enable clock stretching by setting the I2C_SLAVE_SCL_STRETCH_EN (slave), and clearing I2C_RX_FULL_ACK_LEVEL. When RX RAM is full, an I2C_SLAVE_STRETCH_INT (slave) interrupt is generated. In this way, $I2C_{slave}$ can hold SCL low, in exchange for more time to read data. After the software has finished reading, you can set I2C_SLAVE_STRETCH_INT_CLR (slave) to 1 to clear interrupt, and set I2C_SLAVE_SCL_STRETCH_CLR (slave) to release the SCL line.
13. After data transfer completes, $I2C_{master}$ executes the STOP command, and generates an I2C_TRANS_COMPLETE_INT (master) interrupt.

47.6.2 $I2C_{master}$ Writes to $I2C_{slave}$ with a 10-bit Address in One Command Sequence

47.6.2.1 Introduction

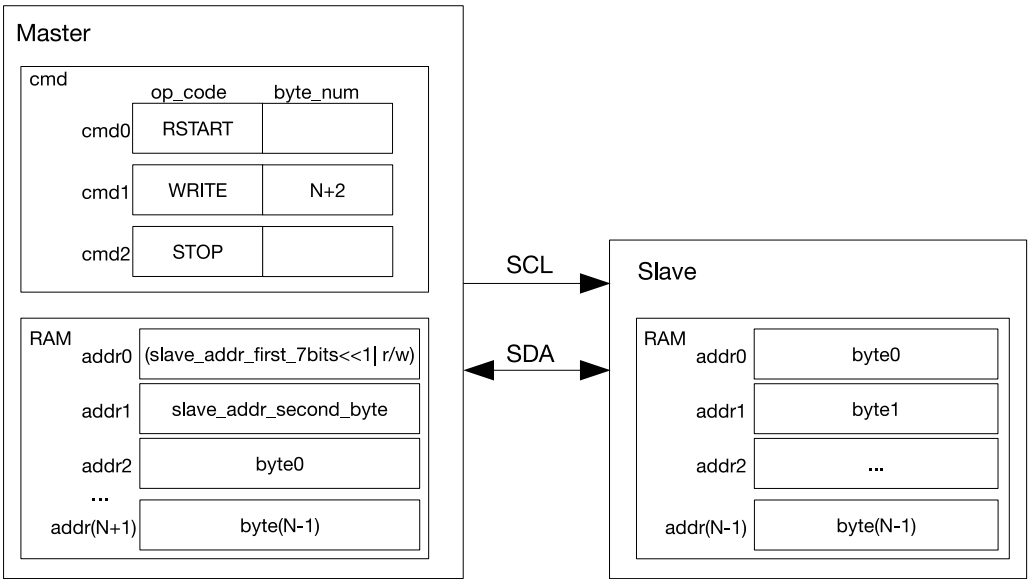


Figure 47.6-2. $I2C_{master}$ Writing to a Slave with a 10-bit Address

Figure 47.6-2 shows how $I2C_{master}$ writes N bytes of data using 10-bit addressing to an I2C slave. The configuration and transfer process is similar to what is described in 47.6.1, except that a 10-bit $I2C_{slave}$ address is formed from two bytes. Since a 10-bit $I2C_{slave}$ address has one more byte than a 7-bit $I2C_{slave}$ address, byte_num and length of data in TX RAM increase by 1 accordingly.

47.6.2.2 Configuration Example

1. Set I2C_MS_MODE (master) to 1, and I2C_MS_MODE (slave) to 0.
2. Write 1 to I2C_CONF_UPGATE (master) and I2C_CONF_UPGATE (slave) to synchronize registers.
3. Configure command registers of $I2C_{master}$.

Command registers	op_code	ack_value	ack_exp	ack_check_er	byte_num
I2C_COMMAND0 (master)	RSTART	—	—	—	—
I2C_COMMAND1 (master)	WRITE	—	ack_exp	1	N+2
I2C_COMMAND2 (master)	STOP	—	—	—	—

4. Configure I2C_SLAVE_ADDR (slave) in I2C_SLAVE_ADDR_REG (slave) as $I2C_{slave}$'s 10-bit address, and set I2C_ADDR_10BIT_EN (slave) to 1 to enable 10-bit addressing.
5. Write the address of $I2C_{slave}$ and data to be sent to TX RAM of $I2C_{master}$. The first byte of the address of $I2C_{slave}$ comprises ((0x78 | I2C_SLAVE_ADDR[9:8])«1) and a $R/\overline{W}$ bit. The second byte of the address of $I2C_{slave}$ is I2C_SLAVE_ADDR[7:0]. These two bytes are followed by data to be sent in FIFO or non-FIFO mode.
6. Write 1 to I2C_CONF_UPGATE (master) and I2C_CONF_UPGATE (slave) to synchronize registers.
7. Write 1 to I2C_TRANS_START (master) and I2C_TRANS_START (slave) to start transfer.

8. $I2C_{slave}$ compares the slave address sent by $I2C_{master}$ with its own address in I2C_SLAVE_ADDR (slave). When ack_check_en (master) in $I2C_{master}$'s WRITE command is 1, $I2C_{master}$ checks ACK value each time it sends a byte. When ack_check_en (master) is 0, $I2C_{master}$ does not check the ACK value and take $I2C_{slave}$ as a matching slave by default.
   - Match: If the received ACK value matches ack_exp (master) (the expected ACK value), $I2C_{master}$ continues data transfer.
   - Not match: If the received ACK value does not match ack_exp, $I2C_{master}$ generates an I2C_NACK_INT (master) interrupt and stops data transfer.
9. $I2C_{master}$ sends data, and determines whether to check ACK value according to ack_check_en (master).
10. If data to be sent is larger than TX FIFO depth, TX RAM of $I2C_{master}$ may wrap around in FIFO mode. For details, please refer to Section 47.4.10.
11. If data to be received is larger than RX FIFO depth, RX RAM of $I2C_{slave}$ may wrap around in FIFO mode. For details, please refer to Section 47.4.10.

    If data to be received is larger than RX FIFO depth, the other way is to enable clock stretching by setting I2C_SLAVE_SCL_STRETCH_EN (slave), and clearing I2C_RX_FULL_ACK_LEVEL to 0. When RX RAM is full, an I2C_SLAVE_STRETCH_INT (slave) interrupt is generated. In this way, $I2C_{slave}$ can hold SCL low, in exchange for more time to read data. After the software has finished reading, you can set I2C_SLAVE_STRETCH_INT_CLR (slave) to 1 to clear interrupt, and set I2C_SLAVE_SCL_STRETCH_CLR (slave) to release the SCL line.
12. After data transfer completes, $I2C_{master}$ executes the STOP command, and generates an I2C_TRANS_COMPLETE_INT (master) interrupt.

47.6.3 $I2C_{master}$ Writes to $I2C_{slave}$ with Two 7-bit Addresses in One Command Sequence

47.6.3.1 Introduction

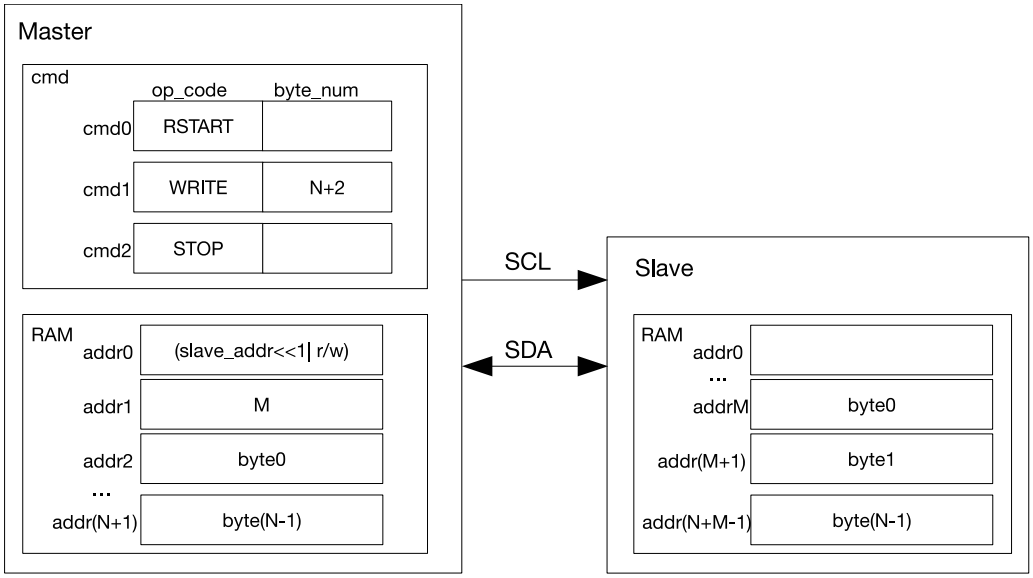


Figure 47.6-3. $I2C_{master}$ Writing to $I2C_{slave}$ with Two 7-bit Addresses

Figure 47.6-3 shows how $I2C_{master}$ writes N bytes of data to $I2C_{slave}$ registers or RAM using 7-bit double addressing. The configuration and transfer process is similar to what is described in Section 47.6.1, except that in 7-bit dual address mode $I2C_{master}$ sends two 7-bit addresses. The first address is the address of an I2C slave, and the second one is $I2C_{slave}$'s memory address (i.e., addrM in Figure 47.6-3). When using double addressing, the slave's RX RAM must be accessed in the non-FIFO mode, and clock stretching by the slave must be disabled. In the non-FIFO mode, the write pointer wraps around to the initial position of the FIFO once its depth is exceeded. As a consequence, data in the slave's RX RAM is overwritten cyclically every 32 bytes. When new data is written to an address that already contains data in the slave, the existing data is directly overwritten without any warning or error indication.

47.6.3.2 Configuration Example

1. Set I2C_MS_MODE (master) to 1, and I2C_MS_MODE (slave) to 0.
2. Set I2C_FIFO_ADDR_CFG_EN (slave) to 1 to enable dual address mode.
3. Write 1 to I2C_CONF_UPGATE (master) and I2C_CONF_UPGATE (slave) to synchronize registers.
4. Configure command registers of $I2C_{master}$.

Command registers	op_code	ack_value	ack_exp	ack_check_er	byte_num
I2C_COMMAND0 (master)	RSTART	—	—	—	—
I2C_COMMAND1 (master)	WRITE	—	ack_exp	1	N+2
I2C_COMMAND2 (master)	STOP	—	—	—	—

5. Write the address of $I2C_{slave}$ and data to be sent to TX RAM of $I2C_{master}$ in FIFO or non-FIFO mode.
6. Write the address of $I2C_{slave}$ to I2C_SLAVE_ADDR (slave) in I2C_SLAVE_ADDR_REG (slave) register.
7. Write 1 to I2C_CONF_UPGATE (master) and I2C_CONF_UPGATE (slave) to synchronize registers.
8. Write 1 to I2C_TRANS_START (master) and I2C_TRANS_START (slave) to start transfer.
9. $I2C_{slave}$ compares the slave address sent by $I2C_{master}$ with its own address in I2C_SLAVE_ADDR (slave). When ack_check_en (master) in $I2C_{master}$'s WRITE command is 1, $I2C_{master}$ checks ACK value each time it sends a byte. When ack_check_en (master) is 0, $I2C_{master}$ does not check ACK value and takes $I2C_{slave}$ as a matching slave by default.
   - Match: If the received ACK value matches ack_exp (master) (the expected ACK value), $I2C_{master}$ continues data transfer.
   - Not match: If the received ACK value does not match ack_exp, $I2C_{master}$ generates an I2C_NACK_INT (master) interrupt and stops data transfer.
10. $I2C_{slave}$ receives the RX RAM address sent by $I2C_{master}$ and adds the offset.
11. $I2C_{master}$ sends data, and determines whether to check ACK value according to ack_check_en (master).
12. If data to be sent is larger than TX FIFO depth, TX RAM of $I2C_{master}$ may wrap around in FIFO mode. For details, please refer to Section 47.4.10.
13. After data transfer completes, $I2C_{master}$ executes the STOP command, and generates an I2C_TRANS_COMPLETE_INT (master) interrupt.

47.6.4 $I2C_{master}$ Writes to $I2C_{slave}$ with a 7-bit Address in Multiple Command Sequences

47.6.4.1 Introduction

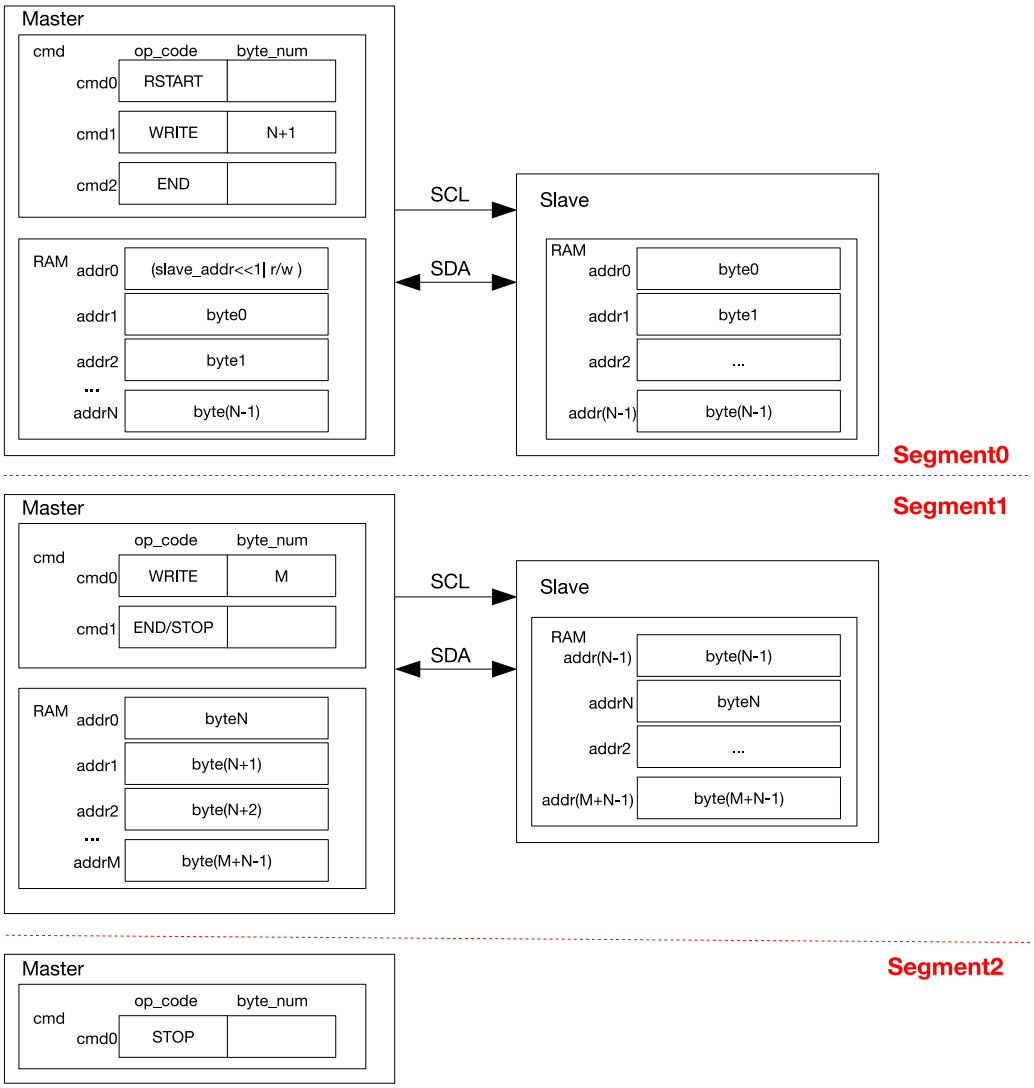


Figure 47.6-4. $I2C_{master}$ Writing to $I2C_{slave}$ with a 7-bit Address in Multiple Sequences

Given that the I2C Controller RAM holds only the size of TX/RX FIFO depth, when data are too large to be processed, it is advised to transmit them in multiple command sequences. At the end of every command sequence is an END command. When the controller executes this END command, SCL will be pulled low, and the software can refresh command sequence registers and the RAM for the next transfer.

Figure 47.6-4 shows how $I2C_{master}$ writes to an I2C slave in two or three segments as an example. For the first segment, the CMD_Controller registers are configured as shown in Segment0. Once data in $I2C_{master}$'s RAM is ready and I2C_TRANS_START is set, $I2C_{master}$ initiates data transfer. After executing the END command, $I2C_{master}$ turns off the SCL clock and pulls SCL low to reserve the bus. Meanwhile, the controller generates an I2C_END_DETECT_INT interrupt.

For the second segment, after detecting the I2C_END_DETECT_INT interrupt, the software refreshes the

CMD_Controller registers, reloads the RAM, and clears this interrupt, as shown in Segment1. If cmd1 in the second segment is a STOP, then data is transmitted to $I2C_{slave}$ in two segments. $I2C_{master}$ resumes data transfer after I2C_TRANS_START is set, and terminates the transfer by sending a STOP bit.

For the third segment, after the second data transfer finishes and an I2C_END_DETECT_INT is detected, the CMD_Controller registers of $I2C_{master}$ are configured as shown in Segment2. Once I2C_TRANS_START is set, $I2C_{master}$ generates a STOP bit and terminates the transfer.

Note that other $I2C_{master}$s will not transact on the bus between two segments. The bus is only released after a STOP command is sent. The I2C controller can be reset by setting I2C_FSM_RST field at any time. This field will later be cleared automatically by hardware.

47.6.4.2 Configuration Example

1. Set I2C_MS_MODE (master) to 1, and I2C_MS_MODE (slave) to 0.
2. Write 1 to I2C_CONF_UPGATE (master) and I2C_CONF_UPGATE (slave) to synchronize registers.
3. Configure command registers of $I2C_{master}$.

Command registers	op_code	ack_value	ack_exp	ack_check_er	byte_num
I2C_COMMAND0 (master)	RSTART	—	—	—	—
I2C_COMMAND1 (master)	WRITE	—	ack_exp	1	N+1
I2C_COMMAND2 (master)	END	—	—	—	—

4. Write the address of $I2C_{slave}$ and data to be sent to TX RAM of $I2C_{master}$ in either FIFO mode or non-FIFO mode according to Section 47.4.10.
5. Write the address of $I2C_{slave}$ to I2C_SLAVE_ADDR (slave) in I2C_SLAVE_ADDR_REG (slave) register
6. Write 1 to I2C_CONF_UPGATE (master) and I2C_CONF_UPGATE (slave) to synchronize registers.
7. Write 1 to I2C_TRANS_START (master) and I2C_TRANS_START (slave) to start transfer.
8. $I2C_{slave}$ compares the slave address sent by $I2C_{master}$ with its own address in I2C_SLAVE_ADDR (slave). When ack_check_en (master) in $I2C_{master}$'s WRITE command is 1, $I2C_{master}$ checks ACK value each time it sends a byte. When ack_check_en (master) is 0, $I2C_{master}$ does not check ACK value and takes $I2C_{slave}$ as a matching slave by default.
   - Match: If the received ACK value matches ack_exp (master) (the expected ACK value), $I2C_{master}$ continues data transfer.
   - Not match: If the received ACK value does not match ack_exp, $I2C_{master}$ generates an I2C_NACK_INT (master) interrupt and stops data transfer.
9. $I2C_{master}$ sends data, and checks ACK value or not according to ack_check_en (master).
10. After the I2C_END_DETECT_INT (master) interrupt is generated, set I2C_END_DETECT_INT_CLR (master) to 1 to clear this interrupt.
11. Update $I2C_{master}$'s command registers.

Command registers	op_code	ack_value	ack_exp	ack_check_er	byte_num
I2C_COMMAND0 (master)	WRITE	—	ack_exp	1	M
I2C_COMMAND1 (master)	END/STOP	—	—	—	—

12. Write M bytes of data to be sent to TX RAM of $I2C_{master}$ in FIFO or non-FIFO mode.
13. Write 1 to I2C_TRANS_START (master) bit to start the transfer and repeat step 9.
14. If the command is a STOP, I2C stops the transfer and generates an I2C_TRANS_COMPLETE_INT (master) interrupt.
15. If the command is an END, repeat step 10.
16. Update $I2C_{master}$'s command registers.

Command registers of $I2C_{master}$	op_code	ack_value	ack_exp	ack_check_er	byte_num
I2C_COMMAND1 (master)	STOP	—	—	—	—

17. Write 1 to I2C_TRANS_START (master) bit to start transfer.
18. $I2C_{master}$ executes the STOP command and generates an I2C_TRANS_COMPLETE_INT (master) interrupt.

47.6.5 $I2C_{master}$ Reads $I2C_{slave}$ with a 7-bit Address in One Command Sequence

47.6.5.1 Introduction

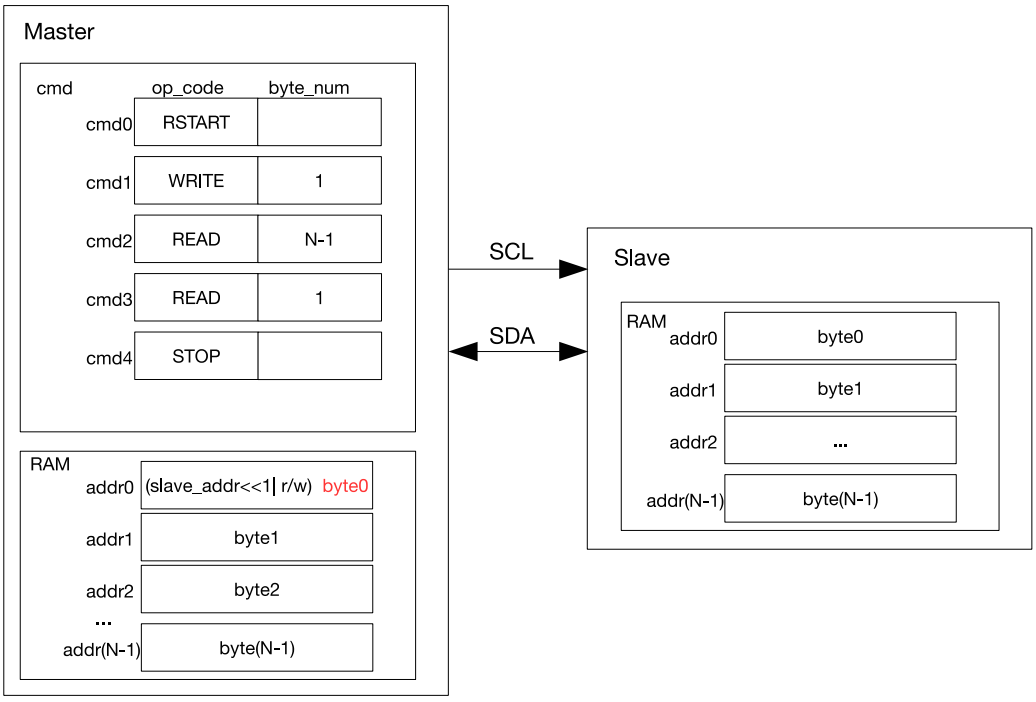


Figure 47.6-5. $I2C_{master}$ Reading $I2C_{slave}$ with a 7-bit Address

Figure 47.6-5 shows how I2C$_{master}$ reads N bytes of data from an I2C slave using 7-bit addressing. cmd1 is a WRITE command, and when this command is executed I2C$_{master}$ sends the address of I2C$_{slave}$. The byte sent comprises a 7-bit I2C$_{slave}$ address and a $R/\overline{W}$ bit. When the $R/\overline{W}$ bit is 1, it indicates a READ operation. If the address of an I2C slave matches the sent address, this matching slave starts sending data to I2C$_{master}$. I2C$_{master}$ generates acknowledgments according to ack_value defined in the READ command upon receiving a byte.

As illustrated in Figure 47.6-5, I2C$_{master}$ executes two READ commands: it generates ACKs for (N-1) bytes of data in cmd2, and a NACK for the last byte of data in cmd 3. This configuration may be changed as required. I2C$_{master}$ writes received data into the controller RAM from addr0, whose original content (the address of I2C$_{slave}$ and a $R/\overline{W}$ bit) is overwritten by byte0 marked red in Figure 47.6-5.

47.6.5.2 Configuration Example

1. Set I2C_MS_MODE (master) to 1, and I2C_MS_MODE (slave) to 0.
2. We recommend setting I2C_SLAVE_SCL_STRETCH_EN (slave) to 1, so that SCL can be held low for more processing time when I2C$_{slave}$ needs to send data. If this bit is not set, the software should write data to be sent to I2C$_{slave}$'s TX RAM before I2C$_{master}$ initiates the transfer. The configuration below is applicable to the scenario where I2C_SLAVE_SCL_STRETCH_EN (slave) is 1.
3. Write 1 to I2C_CONF_UPGATE (master) and I2C_CONF_UPGATE (slave) to synchronize registers.
4. Configure command registers of I2C$_{master}$.

Command registers of I2C$_{master}$	op_code	ack_value	ack_exp	ack_check_er	byte_num
I2C_COMMAND0 (master)	RSTART	—	—	—	—
I2C_COMMAND1 (master)	WRITE	0	0	1	1
I2C_COMMAND2 (master)	READ	0	0	1	N-1
I2C_COMMAND3 (master)	READ	1	0	1	1
I2C_COMMAND4 (master)	STOP	—	—	—	—

5. Write the address of I2C$_{slave}$ to TX RAM of I2C$_{master}$ in either FIFO mode or non-FIFO mode according to Section 47.4.10.
6. Write the address of I2C$_{slave}$ to I2C_SLAVE_ADDR (slave) in I2C_SLAVE_ADDR_REG (slave) register.
7. Write 1 to I2C_CONF_UPGATE (master) and I2C_CONF_UPGATE (slave) to synchronize registers.
8. Write 1 to I2C_TRANS_START (master) bit to start I2C$_{master}$'s transfer.
9. Start I2C$_{slave}$'s transfer according to Section 47.4.14.
10. I2C$_{slave}$ compares the slave address sent by I2C$_{master}$ with its own address in I2C_SLAVE_ADDR (slave). When ack_check_en (master) in I2C$_{master}$'s WRITE command is 1, I2C$_{master}$ checks ACK value each time it sends a byte. When ack_check_en (master) is 0, I2C$_{master}$ does not check ACK value and takes I2C$_{slave}$ as a matching slave by default.

- Match: If the received ACK value matches ack_exp (master) (the expected ACK value), $I2C_{master}$ continues data transfer.
- Not match: If the received ACK value does not match ack_exp, $I2C_{master}$ generates an I2C_NACK_INT (master) interrupt and stops data transfer.

11. After I2C_SLAVE_STRETCH_INT (slave) is generated, the I2C_STRETCH_CAUSE bit is 0. The address of $I2C_{slave}$ matches the address sent over SDA, and $I2C_{slave}$ needs to send data.
12. Write data to be sent to TX RAM of $I2C_{slave}$ in either FIFO mode or non-FIFO mode according to Section 47.4.10.
13. Set I2C_SLAVE_SCL_STRETCH_CLR (slave) to 1 to release SCL.
14. $I2C_{slave}$ sends data, and $I2C_{master}$ checks ACK value or not according to ack_check_en (master) in the READ command.
15. If data to be read by $I2C_{master}$ is larger than the TX FIFO depth of $I2C_{slave}$, an I2C_SLAVE_STRETCH_INT (slave) interrupt will be generated when TX RAM of $I2C_{slave}$ becomes empty. In this way, $I2C_{slave}$ can hold SCL low, so that software has more time to pad data in TX RAM of $I2C_{slave}$ and read data in RX RAM of $I2C_{master}$. After software has finished reading, you can set I2C_SLAVE_STRETCH_INT_CLR (slave) to 1 to clear interrupt, and set I2C_SLAVE_SCL_STRETCH_CLR (slave) to 1 to release the SCL line.
16. After $I2C_{master}$ has received the last byte of data, set ack_value (master) to 1. $I2C_{slave}$ will stop the transfer once receiving the I2C_NACK_INT interrupt.
17. After data transfer completes, $I2C_{master}$ executes the STOP command, and generates an I2C_TRANS_COMPLETE_INT (master) interrupt.

47.6.6 $I2C_{master}$ Reads $I2C_{slave}$ with a 10-bit Address in One Command Sequence

47.6.6.1 Introduction

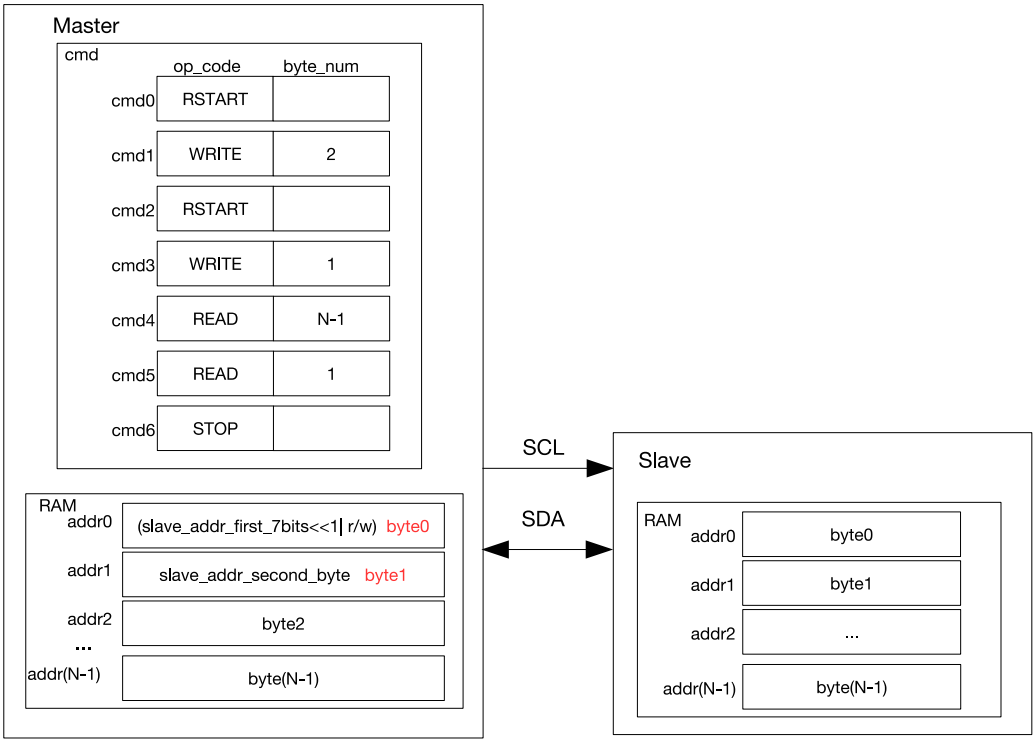


Figure 47.6-6. $I2C_{master}$ Reading $I2C_{slave}$ with a 10-bit Address

Figure 47.6-6 shows how $I2C_{master}$ reads data from an I2C slave using 10-bit addressing. Unlike 7-bit addressing, in 10-bit addressing the WRITE command of the $I2C_{master}$ is formed from two bytes, and correspondingly TX RAM of this master stores a 10-bit address of two bytes. The $R/\overline{W}$ bit in the first byte is 0, which indicates a WRITE operation. After a RSTART condition, $I2C_{master}$ sends the first byte of address again to read data from $I2C_{slave}$, but the $R/\overline{W}$ bit is 1, which indicates a READ operation. The two address bytes can be configured as described in Section 47.6.2.

47.6.6.2 Configuration Example

1. Set I2C_MS_MODE (master) to 1, and I2C_MS_MODE (slave) to 0.
2. We recommend setting I2C_SLAVE_SCL_STRETCH_EN (slave) to 1, so that SCL can be held low for more processing time when $I2C_{slave}$ needs to send data. If this bit is not set, the software should write data to be sent to $I2C_{slave}$'s TX RAM before $I2C_{master}$ initiates the transfer. The configuration below is applicable to a scenario where I2C_SLAVE_SCL_STRETCH_EN (slave) is 1.
3. Write 1 to I2C_CONF_UPGATE (master) and I2C_CONF_UPGATE (slave) to synchronize registers.
4. Configure command registers of $I2C_{master}$.

Command registers of $I2C_{master}$	op_code	ack_value	ack_exp	ack_check_er	byte_num
I2C_COMMAND0 (master)	RSTART	—	—	—	—
I2C_COMMAND1 (master)	WRITE	0	0	1	2

I2C_COMMAND2 (master)	RSTART	—	—	—	—
I2C_COMMAND3 (master)	WRITE	0	0	1	1
I2C_COMMAND4 (master)	READ	0	0	1	N-1
I2C_COMMAND5 (master)	READ	1	0	1	1
I2C_COMMAND6 (master)	STOP	—	—	—	—

5. Configure I2C_SLAVE_ADDR (slave) in I2C_SLAVE_ADDR_REG (slave) as $I2C_{slave}$'s 10-bit address, and set I2C_ADDR_10BIT_EN (slave) to 1 to enable 10-bit addressing.
6. Write the address of $I2C_{slave}$ and data to be sent to TX RAM of $I2C_{master}$ in either FIFO or non-FIFO mode. The first byte of address comprises ((0x78 | I2C_SLAVE_ADDR[9:8])«1) and a $R/\overline{W}$ bit, which is 1 and indicates a WRITE operation. The second byte of address is I2C_SLAVE_ADDR[7:0]. The third byte is ((0x78 | I2C_SLAVE_ADDR[9:8])«1) and a $R/\overline{W}$ bit, which is 1 and indicates a READ operation.
7. Write 1 to I2C_CONF_UPGATE (master) and I2C_CONF_UPGATE (slave) to synchronize registers.
8. Write 1 to I2C_TRANS_START (master) to start $I2C_{master}$'s transfer.
9. Start $I2C_{slave}$'s transfer according to Section 47.4.14.
10. $I2C_{slave}$ compares the slave address sent by $I2C_{master}$ with its own address in I2C_SLAVE_ADDR (slave). When ack_check_en (master) in $I2C_{master}$'s WRITE command is 1, $I2C_{master}$ checks ACK value each time it sends a byte. When ack_check_en (master) is 0, $I2C_{master}$ does not check ACK value and takes $I2C_{slave}$ as a matching slave by default.
    - Match: If the received ACK value matches ack_exp (master) (the expected ACK value), $I2C_{master}$ continues data transfer.
    - Not match: If the received ACK value does not match ack_exp, $I2C_{master}$ generates an I2C_NACK_INT (master) interrupt and stops data transfer.
11. $I2C_{master}$ sends a RSTART and the third byte in TX RAM, which is ((0x78 | I2C_SLAVE_ADDR[9:8])«1) and a $R/\overline{W}$ bit that indicates READ.
12. $I2C_{slave}$ repeats step 10. If its address matches the address sent by $I2C_{master}$, $I2C_{slave}$ proceed on to the next steps.
13. After I2C_SLAVE_STRETCH_INT (slave) is generated, the I2C_STRETCH_CAUSE bit is 0. The address of $I2C_{slave}$ matches the address sent over SDA, and $I2C_{slave}$ needs to send data.
14. Write data to be sent to TX RAM of $I2C_{slave}$ in either FIFO mode or non-FIFO mode according to Section 47.4.10.
15. Set I2C_SLAVE_SCL_STRETCH_CLR (slave) to 1 to release SCL.
16. $I2C_{slave}$ sends data, and $I2C_{master}$ checks ACK value or not according to ack_check_en (master) in the READ command.

17. If data to be read by $I2C_{master}$ is larger than the TX FIFO depth of $I2C_{slave}$, an I2C_SLAVE_STRETCH_INT (slave) interrupt will be generated when TX RAM of $I2C_{slave}$ becomes empty. In this way, $I2C_{slave}$ can hold SCL low, so that software has more time to pad data in TX RAM of $I2C_{slave}$ and read data in RX RAM of $I2C_{master}$. After the software has finished reading, you can set I2C_SLAVE_STRETCH_INT_CLR (slave) to 1 to clear interrupt, and set I2C_SLAVE_SCL_STRETCH_CLR (slave) to release the SCL line.
18. After $I2C_{master}$ has received the last byte of data, set ack_value (master) to 1. $I2C_{slave}$ will stop transfer once receiving the I2C_NACK_INT interrupt.
19. After data transfer completes, $I2C_{master}$ executes the STOP command, and generates an I2C_TRANS_COMPLETE_INT (master) interrupt.

47.6.7 $I2C_{master}$ Reads $I2C_{slave}$ with Two 7-bit Addresses in One Command Sequence

47.6.7.1 Introduction

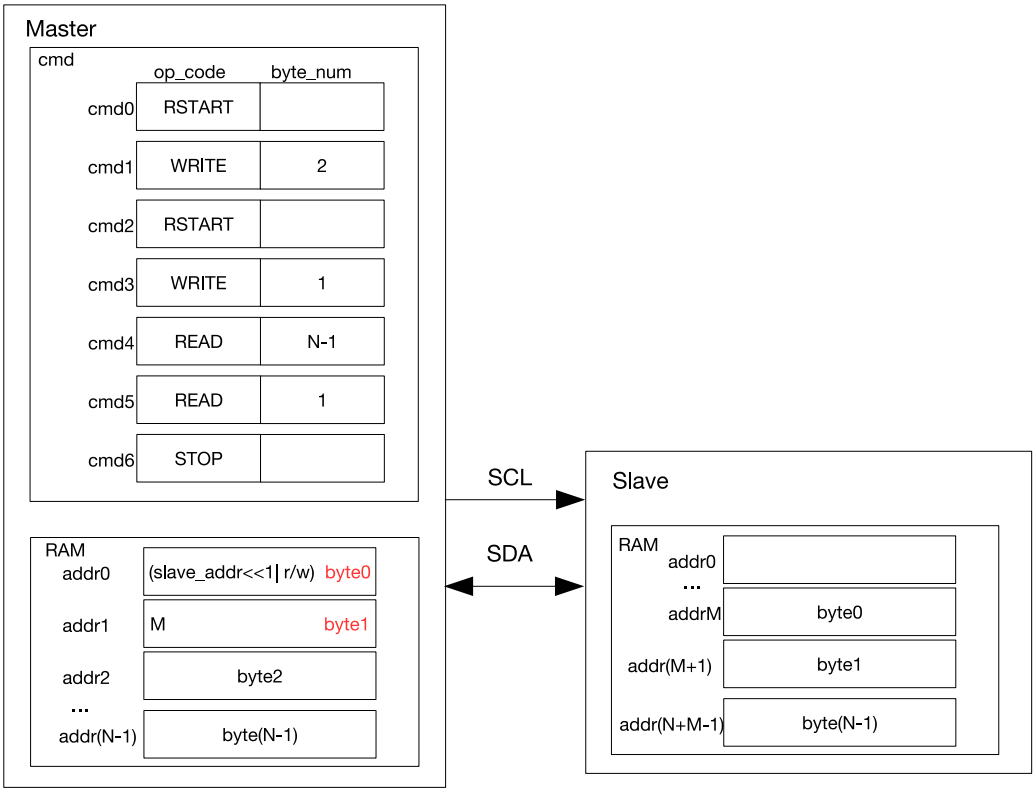


Figure 47.6-7. $I2C_{master}$ Reading N Bytes of Data from addrM of $I2C_{slave}$ with Two 7-bit Addresses

Figure 47.6-7 shows how $I2C_{master}$ reads data from specified addresses in an I2C slave. $I2C_{master}$ sends two bytes of addresses: the first byte is a 7-bit $I2C_{slave}$ address followed by a $R/\overline{W}$ bit, which is 0 and indicates a WRITE; the second byte is $I2C_{slave}$'s memory address. After a RSTART condition, $I2C_{master}$ sends the first byte of address again, but the $R/\overline{W}$ bit is 1 which indicates a READ. Then, $I2C_{master}$ reads data starting from addrM. When double addressing mode is used, the slave's TX RAM must be accessed in non-FIFO mode, and clock stretching by the slave must be disabled. In this configuration, the slave's TX FIFO operates in a non-FIFO manner. Even in the case of an underflow condition where the master attempts to read more data than available, the slave continues to return data corresponding to the current address pointer without generating

warnings. The address pointer wraps around to the beginning of the memory space once the end is reached.

47.6.7.2 Configuration Example

1. Set I2C_MS_MODE (master) to 1, and I2C_MS_MODE (slave) to 0.
2. We recommend setting I2C_SLAVE_SCL_STRETCH_EN (slave) to 1, so that SCL can be held low for more processing time when $I2C_{slave}$ needs to send data. If this bit is not set, the software should write data to be sent to $I2C_{slave}$'s TX RAM before $I2C_{master}$ initiates the transfer. The configuration below is applicable to the scenario where I2C_SLAVE_SCL_STRETCH_EN (slave) is 1.
3. Set I2C_FIFO_ADDR_CFG_EN (slave) to 1 to enable dual address mode.
4. Write 1 to I2C_CONF_UPGATE (master) and I2C_CONF_UPGATE (slave) to synchronize registers.
5. Configure command registers of $I2C_{master}$.

Command registers of $I2C_{master}$	op_code	ack_value	ack_exp	ack_check_er	byte_num
I2C_COMMAND0 (master)	RSTART	—	—	—	—
I2C_COMMAND1 (master)	WRITE	0	0	1	2
I2C_COMMAND2 (master)	RSTART	—	—	—	—
I2C_COMMAND3 (master)	WRITE	0	0	1	1
I2C_COMMAND4 (master)	READ	0	0	1	N-1
I2C_COMMAND5 (master)	READ	1	0	1	1
I2C_COMMAND6 (master)	STOP	—	—	—	—

6. Configure I2C_SLAVE_ADDR (slave) in I2C_SLAVE_ADDR_REG (slave) register as $I2C_{slave}$'s 7-bit address, and set I2C_ADDR_10BIT_EN (slave) to 0 to enable 7-bit addressing.
7. Write the address of $I2C_{slave}$ and data to be sent to TX RAM of $I2C_{master}$ in either FIFO or non-FIFO mode according to Section 47.4.10. The first byte of address comprises (I2C_SLAVE_ADDR[6:0])«1) and a $R/\overline{W}$ bit, which is 0 and indicates a WRITE. The second byte of the address is memory address M of $I2C_{slave}$. The third byte is (I2C_SLAVE_ADDR[6:0])«1) and a $R/\overline{W}$ bit, which is 1 and indicates a READ.
8. Write 1 to I2C_CONF_UPGATE (master) and I2C_CONF_UPGATE (slave) to synchronize registers.
9. Write 1 to I2C_TRANS_START (master) to start $I2C_{master}$'s transfer.
10. Start $I2C_{slave}$'s transfer according to Section 47.4.14.
11. $I2C_{slave}$ compares the slave address sent by $I2C_{master}$ with its own address in I2C_SLAVE_ADDR (slave). When ack_check_en (master) in $I2C_{master}$'s WRITE command is 1, $I2C_{master}$ checks ACK value each time it sends a byte. When ack_check_en (master) is 0, $I2C_{master}$ does not check ACK value and takes $I2C_{slave}$ as a matching slave by default.

- Match: If the received ACK value matches ack_exp (master) (the expected ACK value), $I2C_{master}$ continues data transfer.
- Not match: If the received ACK value does not match ack_exp, $I2C_{master}$ generates an I2C_NACK_INT (master) interrupt and stops data transfer.

12. $I2C_{slave}$ receives memory address sent by $I2C_{master}$ and adds the offset.
13. $I2C_{master}$ sends a RSTART and the third byte in TX RAM, which is ((0x78 | I2C_SLAVE_ADDR[6:0])«1)and an R bit.
14. $I2C_{slave}$ repeats step 11. If its address matches the address sent by $I2C_{master}$, $I2C_{slave}$ proceed on to the next steps.
15. Write data to be sent to TX RAM of $I2C_{slave}$ in non-FIFO mode.
16. $I2C_{slave}$ sends data, and $I2C_{master}$ checks ACK value or not according to ack_check_en (master) in the READ command.
17. After $I2C_{master}$ has received the last byte of data, set ack_value (master) to 1. $I2C_{slave}$ will stop transfer once receiving the I2C_NACK_INT interrupt.
18. After data transfer completes, $I2C_{master}$ executes the STOP command, and generates an I2C_TRANS_COMPLETE_INT (master) interrupt.

47.6.8 $I2C_{master}$ Reads $I2C_{slave}$ with a 7-bit Address in Multiple Command Sequences

47.6.8.1 Introduction

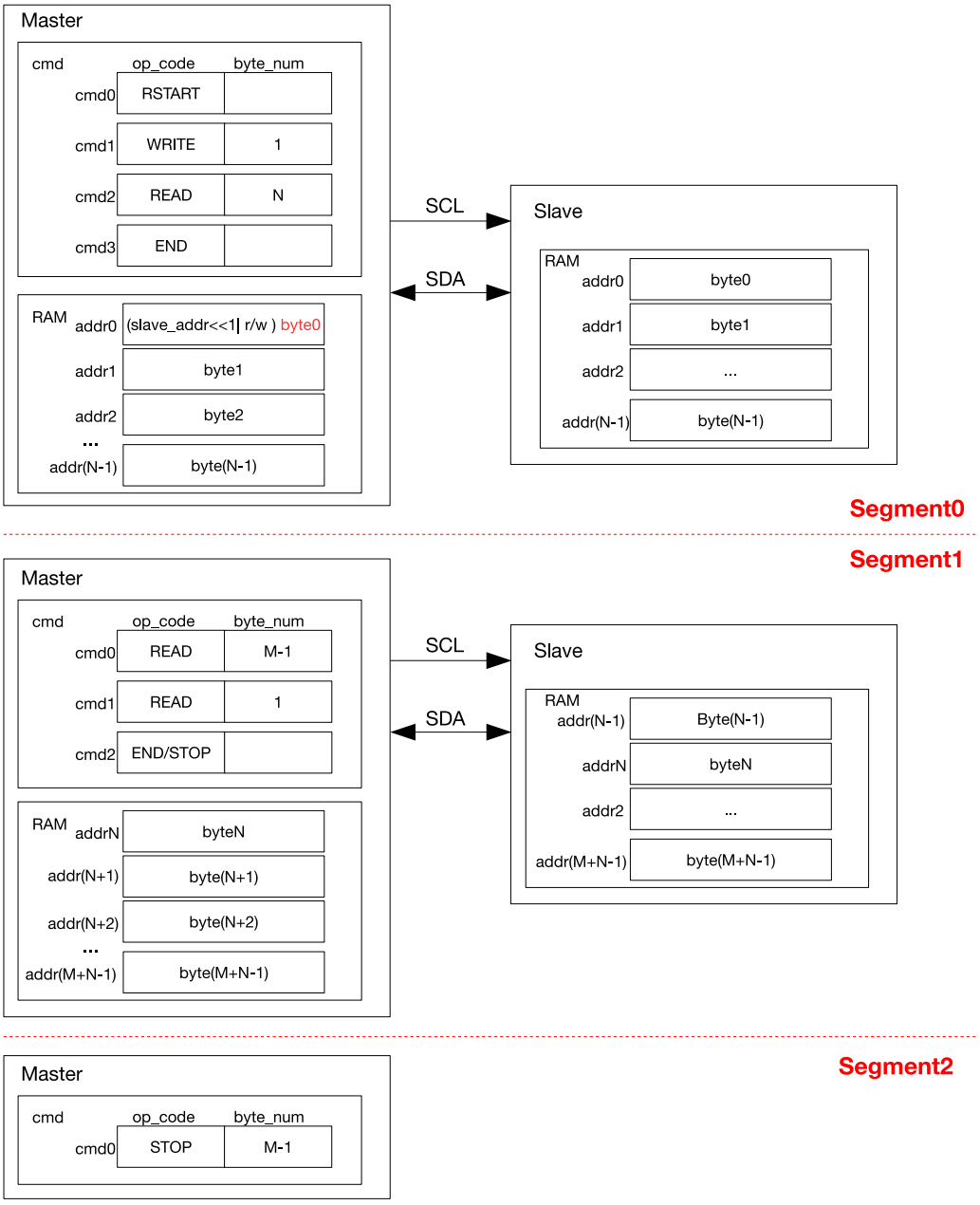


Figure 47.6-8. $I2C_{master}$ Reading $I2C_{slave}$ with a 7-bit Address in Segments

Figure 47.6-8 shows how $I2C_{master}$ reads (N+M) bytes of data from an I2C slave in two/three segments separated by END commands. Configuration procedures are described as follows:

1. The procedures for Segment0 is similar to 47.6-5, except that the last command is an END.
2. Prepare data in the TX RAM of $I2C_{slave}$, and set I2C_TRANS_START to start data transfer. After executing the END command, $I2C_{master}$ refreshes command registers and the RAM as shown in Segment1, and clears the corresponding I2C_END_DETECT_INT interrupt. If cmd2 in Segment1 is a STOP, then data is read from $I2C_{slave}$ in two segments. $I2C_{master}$ resumes data transfer by setting I2C_TRANS_START and terminates the transfer by sending a STOP bit.
3. If cmd2 in Segment1 is an END, then data is read from $I2C_{slave}$ in three segments. After the second data

transfer finishes and an I2C_END_DETECT_INT interrupt is detected, the cmd box is configured as shown in Segment2. Once I2C_TRANS_START is set, $I2C_{master}$ terminates the transfer by sending a STOP bit.

47.6.8.2 Configuration Example

1. Set I2C_MS_MODE (master) to 1, and I2C_MS_MODE (slave) to 0.
2. We recommend setting I2C_SLAVE_SCL_STRETCH_EN (slave) to 1, so that SCL can be held low for more processing time when $I2C_{slave}$ needs to send data. If this bit is not set, the software should write data to be sent to $I2C_{slave}$'s TX RAM before $I2C_{master}$ initiates the transfer. The configuration below is applicable to a scenario where I2C_SLAVE_SCL_STRETCH_EN (slave) is 1.
3. Write 1 to I2C_CONF_UPGATE (master) and I2C_CONF_UPGATE (slave) to synchronize registers.
4. Configure command registers of $I2C_{master}$.

Command registers of $I2C_{master}$	op_code	ack_value	ack_exp	ack_check_er	byte_num
I2C_COMMAND0 (master)	RSTART	—	—	—	—
I2C_COMMAND1 (master)	WRITE	0	0	1	1
I2C_COMMAND2 (master)	READ	0	0	1	N
I2C_COMMAND3 (master)	END	—	—	—	—

5. Write the address of $I2C_{slave}$ to TX RAM of $I2C_{master}$ in FIFO or non-FIFO mode.
6. Write the address of $I2C_{slave}$ to I2C_SLAVE_ADDR (slave) in I2C_SLAVE_ADDR_REG (slave) register.
7. Write 1 to I2C_CONF_UPGATE (master) and I2C_CONF_UPGATE (slave) to synchronize registers.
8. Write 1 to I2C_TRANS_START (master) to start $I2C_{master}$'s transfer.
9. Start $I2C_{slave}$'s transfer according to Section 47.4.14.
10. $I2C_{slave}$ compares the slave address sent by $I2C_{master}$ with its own address in I2C_SLAVE_ADDR (slave). When ack_check_en (master) in $I2C_{master}$'s WRITE command is 1, $I2C_{master}$ checks ACK value each time it sends a byte. When ack_check_en (master) is 0, $I2C_{master}$ does not check the ACK value and takes $I2C_{slave}$ as a matching slave by default.
    - Match: If the received ACK value matches ack_exp (master) (the expected ACK value), $I2C_{master}$ continues data transfer.
    - Not match: If the received ACK value does not match ack_exp, $I2C_{master}$ generates an I2C_NACK_INT (master) interrupt and stops data transfer.
11. After I2C_SLAVE_STRETCH_INT (slave) is generated, the I2C_STRETCH_CAUSE bit is 0. The address of $I2C_{slave}$ matches the address sent over SDA, and $I2C_{slave}$ needs to send data.
12. Write data to be sent to TX RAM of $I2C_{slave}$ in either FIFO mode or non-FIFO mode according to Section 47.4.10.
13. Set I2C_SLAVE_SCL_STRETCH_CLR (slave) to 1 to release SCL.

14. $I2C_{slave}$ sends data, and $I2C_{master}$ checks ACK value or not according to ack_check_en (master) in the READ command.
15. If data to be read by $I2C_{master}$ in one READ command (N or M) is larger than the TX FIFO depth of $I2C_{slave}$, an I2C_SLAVE_STRETCH_INT (slave) interrupt will be generated when TX RAM of $I2C_{slave}$ becomes empty. In this way, $I2C_{slave}$ can hold SCL low so that software has more time to pad data in TX RAM of $I2C_{slave}$ and read data in RX RAM of $I2C_{master}$. After the software has finished reading, you can set I2C_SLAVE_STRETCH_INT_CLR (slave) to 1 to clear interrupt, and set I2C_SLAVE_SCL_STRETCH_CLR (slave) to release the SCL line.
16. Once finishing reading data in the first READ command, $I2C_{master}$ executes the END command and triggers an I2C_END_DETECT_INT (master) interrupt, which is cleared by setting I2C_END_DETECT_INT_CLR (master) to 1.
17. Update $I2C_{master}$'s command registers using one of the following two methods:

Command registers of $I2C_{master}$	op_code	ack_value	ack_exp	ack_check_er	byte_num
I2C_COMMAND0 (master)	READ	ack_value	ack_exp	1	M
I2C_COMMAND1 (master)	END	—	—	—	—

Or

Command registers of $I2C_{master}$	op_code	ack_value	ack_exp	ack_check_er	byte_num
I2C_COMMAND0 (master)	READ	0	0	1	M-1
I2C_COMMAND1 (master)	READ	1	0	1	1
I2C_COMMAND2 (master)	STOP	—	—	—	—

18. Write M bytes of data to be sent to TX RAM of $I2C_{slave}$. If M is larger than the TX FIFO depth, then repeat step 12 in FIFO or non-FIFO mode.
19. Write 1 to I2C_TRANS_START (master) bit to start the transfer and repeat step 14.
20. If the last command is a STOP, then set ack_value (master) to 1 after $I2C_{master}$ has received the last byte of data. $I2C_{slave}$ stops transfer upon the I2C_NACK_INT interrupt. $I2C_{master}$ executes the STOP command to stop the transfer and generates an I2C_TRANS_COMPLETE_INT (master) interrupt.
21. If the last command is an END, then repeat step 16 and proceed on to the next steps.
22. Update $I2C_{master}$'s command registers.

Command registers of $I2C_{master}$	op_code	ack_value	ack_exp	ack_check_er	byte_num
I2C_COMMAND1 (master)	STOP	—	—	—	—

23. Write 1 to I2C_TRANS_START (master) bit to start transfer.

24. $I2C_{master}$ executes the STOP command to stop transfer, and generates an I2C_TRANS_COMPLETE_INT (master) interrupt.

47.7 Interrupts

ESP32-P4's I2C*n* can generate the I2C*n*_INTR interrupt signal that will be sent to the Interrupt Matrix. There are several internal interrupt sources from I2C*n* that can generate the above interrupt signal(s) as follows:

- I2C_SLAVE_STRETCH_INT: Triggered when one of the four stretching events occurs in slave mode.
- I2C_DET_START_INT: Triggered when the master or the slave detects a START signal.
- I2C_SCL_MAIN_ST_TO_INT: Triggered when the main state machine SCL_MAIN_FSM remains unchanged for over I2C_SCL_MAIN_ST_TO_I2C[23:0] clock cycles.
- I2C_SCL_ST_TO_INT: Triggered when the state machine SCL_FSM remains unchanged for over I2C_SCL_ST_TO_I2C[23:0] clock cycles.
- I2C_RXFIFO_UDF_INT: Triggered when the I2C controller reads RX FIFO via the APB bus, but RX FIFO is empty.
- I2C_TXFIFO_OVF_INT: Triggered when the I2C controller writes TX FIFO via the APB bus, but TX FIFO is full.
- I2C_NACK_INT: Triggered when the ACK value received by the master is not as expected, or when the ACK value received by the slave is 1.
- I2C_TRANS_START_INT: Triggered when the I2C controller sends a START bit.
- I2C_TIME_OUT_INT: Triggered when SCL stays high or low for more than $2^{I2C_TIME_OUT_VALUE}$ clock cycles during data transfer.
- I2C_TRANS_COMPLETE_INT: Triggered when the I2C controller detects a STOP bit.
- I2C_MST_TXFIFO_UDF_INT: Triggered when TX FIFO of the master underflows.
- I2C_ARBITRATION_LOST_INT: Triggered when the SDA's output value does not match its input value while the master's SCL is high.
- I2C_BYTE_TRANS_DONE_INT: Triggered when the I2C controller sends or receives a byte.
- I2C_END_DETECT_INT: Triggered when op_code of the master indicates an END command and an END condition is detected.
- I2C_RXFIFO_OVF_INT: Triggered when RX FIFO of the I2C controller overflows.
- I2C_TXFIFO_WM_INT: I2C TX FIFO watermark interrupt. Triggered when I2C_FIFO_PRT_EN is 1 and the pointers of TX FIFO are less than I2C_TXFIFO_WM_THRHD[4:0].
- I2C_RXFIFO_WM_INT: I2C RX FIFO watermark interrupt. Triggered when I2C_FIFO_PRT_EN is 1 and the pointers of RX FIFO are greater than I2C_RXFIFO_WM_THRHD[4:0].
- I2C_SLAVE_ADDR_UNMATCH_INT: Triggered when the received slave address is inconsistent with the internally configured slave address in slave mode.

Note:
For definitions of *interrupt*, *interrupt signal*, *interrupt source*, and their correlations, please refer to Chapter 13 *Interrupt Matrix* > Section 13.2 *Interrupt Terminology in ESP32-P4*.

Each interrupt source can be configured by a common set of registers that are described in Section *Interrupt Configuration Registers*. The specific registers can be found in Section 47.8.1 *I2C Register Summary*.

47.8 Register Summary

47.8.1 I2C Register Summary

The addresses in this section are relative to the I2C Controller base address provided in Table 9.3-2 in Chapter 9 *System and Memory*.

The abbreviations given in Column **Access** are explained in Section *Access Types for Registers*.

Name	Description	Address	Access
Timing registers			
I2C_SCL_LOW_PERIOD_REG	Configures the low level width of the SCL Clock	0x0000	R/W
I2C_SDA_HOLD_REG	Configures the hold time after a negative SCL edge	0x0030	R/W
I2C_SDA_SAMPLE_REG	Configures the sample time after a positive SCL edge	0x0034	R/W
I2C_SCL_HIGH_PERIOD_REG	Configures the high level width of SCL	0x0038	R/W
I2C_SCL_START_HOLD_REG	Configures the delay between the SDA and SCL negative edge for a start condition	0x0040	R/W
I2C_SCL_RSTART_SETUP_REG	Configures the delay between the positive edge of SCL and the negative edge of SDA	0x0044	R/W
I2C_SCL_STOP_HOLD_REG	Configures the delay after the SCL clock edge for a stop condition	0x0048	R/W
I2C_SCL_STOP_SETUP_REG	Configures the delay between the SDA and SCL rising edge for a stop condition Measurement unit: i2c_sclk	0x004C	R/W
I2C_SCL_ST_TIME_OUT_REG	SCL status time out register	0x0078	R/W
I2C_SCL_MAIN_ST_TIME_OUT_REG	SCL main status time out register	0x007C	R/W
Configuration registers			
I2C_CTR_REG	Transmission setting	0x0004	varies
I2C_TO_REG	Setting time out control for receiving data	0x000C	R/W
I2C_SLAVE_ADDR_REG	Local slave address setting	0x0010	R/W
I2C_FIFO_CONF_REG	FIFO configuration register	0x0018	R/W
I2C_FILTER_CFG_REG	SCL and SDA filter configuration register	0x0050	R/W
I2C_SCL_SP_CONF_REG	Power configuration register	0x0080	varies
I2C_SCL_STRETCH_CONF_REG	Set SCL stretch of I2C slave	0x0084	varies
Status registers			
I2C_SR_REG	Describe I2C work status	0x0008	RO
I2C_FIFO_ST_REG	FIFO status register	0x0014	RO
I2C_DATA_REG	Rx FIFO read data	0x001C	HRO
Interrupt registers			
I2C_INT_RAW_REG	Raw interrupt status	0x0020	R/SS/WTC
I2C_INT_CLR_REG	Interrupt clear bits	0x0024	WT
I2C_INT_ENA_REG	Interrupt enable bits	0x0028	R/W

Name	Description	Address	Access
I2C_INT_STATUS_REG	Status of captured I2C communication events	0x002C	RO
Command registers			
I2C_COMD0_REG	I2C command register 0	0x0058	varies
I2C_COMD1_REG	I2C command register 1	0x005C	varies
I2C_COMD2_REG	I2C command register 2	0x0060	varies
I2C_COMD3_REG	I2C command register 3	0x0064	varies
I2C_COMD4_REG	I2C command register 4	0x0068	varies
I2C_COMD5_REG	I2C command register 5	0x006C	varies
I2C_COMD6_REG	I2C command register 6	0x0070	varies
I2C_COMD7_REG	I2C command register 7	0x0074	varies
Version register			
I2C_DATE_REG	Version register	0x00F8	R/W
Address register			
I2C_TXFIFO_START_ADDR_REG	I2C TX FIFO base address register	0x0100	HRO
I2C_RXFIFO_START_ADDR_REG	I2C RX FIFO base address register	0x0180	HRO

47.8.2 LP_I2C Register Summary

The abbreviations given in Column **Access** are explained in Section *Access Types for Registers*.

Name	Description	Address	Access
Timing Registers			
LP_I2C_SCL_LOW_PERIOD_REG	Configures the low level width of the SCL Clock	0x0000	R/W
LP_I2C_SDA_HOLD_REG	Configures the hold time after a negative SCL edge	0x0030	R/W
LP_I2C_SDA_SAMPLE_REG	Configures the sample time after a positive SCL edge	0x0034	R/W
LP_I2C_SCL_HIGH_PERIOD_REG	Configures the high level width of SCL	0x0038	R/W
LP_I2C_SCL_START_HOLD_REG	Configures the delay between the SDA and SCL negative edge for a start condition	0x0040	R/W
LP_I2C_SCL_RSTART_SETUP_REG	Configures the delay between the positive edge of SCL and the negative edge of SDA	0x0044	R/W
LP_I2C_SCL_STOP_HOLD_REG	Configures the delay after the SCL clock edge for a stop condition	0x0048	R/W
LP_I2C_SCL_STOP_SETUP_REG	Configures the delay between the SDA and SCL positive edge for a stop condition	0x004C	R/W
LP_I2C_SCL_ST_TIME_OUT_REG	SCL status time out register	0x0078	R/W
LP_I2C_SCL_MAIN_ST_TIME_OUT_REG	SCL main status time out register	0x007C	R/W
Configuration Registers			
LP_I2C_CTR_REG	Transmission setting	0x0004	varies
LP_I2C_TO_REG	Setting time out control for receiving data	0x000C	R/W
LP_I2C_FIFO_CONF_REG	FIFO configuration register	0x0018	R/W
LP_I2C_FILTER_CFG_REG	SCL and SDA filter configuration register	0x0050	R/W
LP_I2C_SCL_SP_CONF_REG	Power configuration register	0x0080	varies

Name	Description	Address	Access
Status Registers			
LP_I2C_SR_REG	Describe I2C work status	0x0008	RO
LP_I2C_FIFO_ST_REG	FIFO status register	0x0014	RO
LP_I2C_DATA_REG	Rx FIFO read data	0x001C	RO
Interrupt Registers			
LP_I2C_INT_RAW_REG	Raw interrupt status	0x0020	R/SS/WTC
LP_I2C_INT_CLR_REG	Interrupt clear bits	0x0024	WT
LP_I2C_INT_ENA_REG	Interrupt enable bits	0x0028	R/W
LP_I2C_INT_STATUS_REG	Status of captured I2C communication events	0x002C	RO
Command Registers			
LP_I2C_COMD0_REG	I2C command register 0	0x0058	varies
LP_I2C_COMD1_REG	I2C command register 1	0x005C	varies
LP_I2C_COMD2_REG	I2C command register 2	0x0060	varies
LP_I2C_COMD3_REG	I2C command register 3	0x0064	varies
LP_I2C_COMD4_REG	I2C command register 4	0x0068	varies
LP_I2C_COMD5_REG	I2C command register 5	0x006C	varies
LP_I2C_COMD6_REG	I2C command register 6	0x0070	varies
LP_I2C_COMD7_REG	I2C command register 7	0x0074	varies
Version Register			
LP_I2C_DATE_REG	Version register	0x00F8	R/W
Address Registers			
LP_I2C_TXFIFO_START_ADDR_REG	I2C TXFIFO base address register	0x0100	HRO
LP_I2C_RXFIFO_START_ADDR_REG	I2C RXFIFO base address register	0x0180	HRO

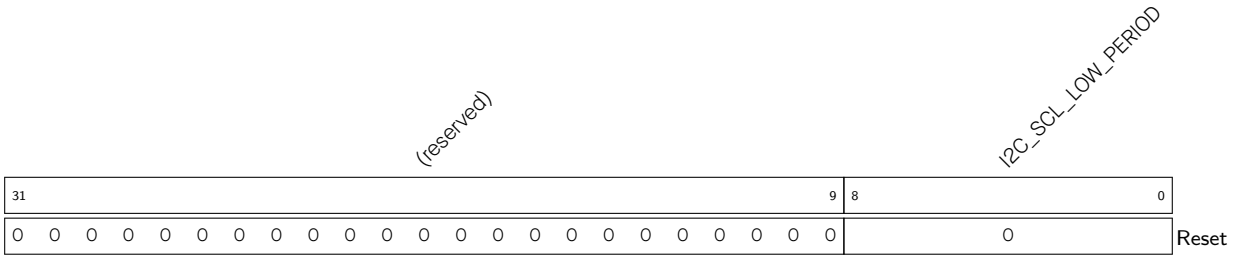
47.9 Registers

47.9.1 I2C Registers

The addresses in this section are relative to the I2C Controller base address provided in Table 9.3-2 in Chapter 9 *System and Memory*.

For how to program reserved fields, please refer to Section Programming Reserved Register Field.

Register 47.1. I2C_SCL_LOW_PERIOD_REG (0x0000)

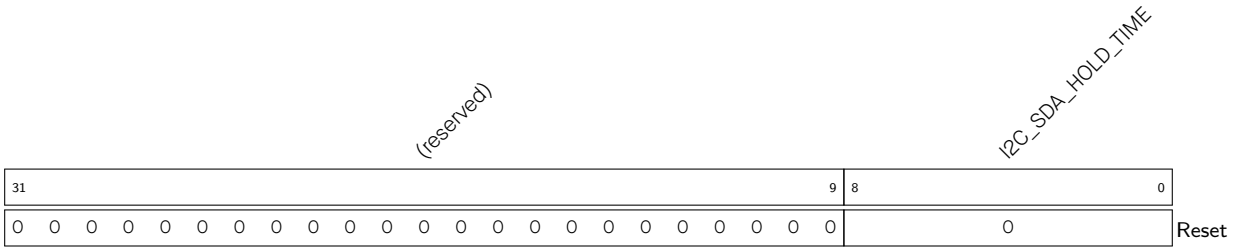


I2C_SCL_LOW_PERIOD Configures the low level width of the SCL Clock in master mode.

Measurement unit: I2C_SCLK clock cycles

(R/W)

Register 47.2. I2C_SDA_HOLD_REG (0x0030)

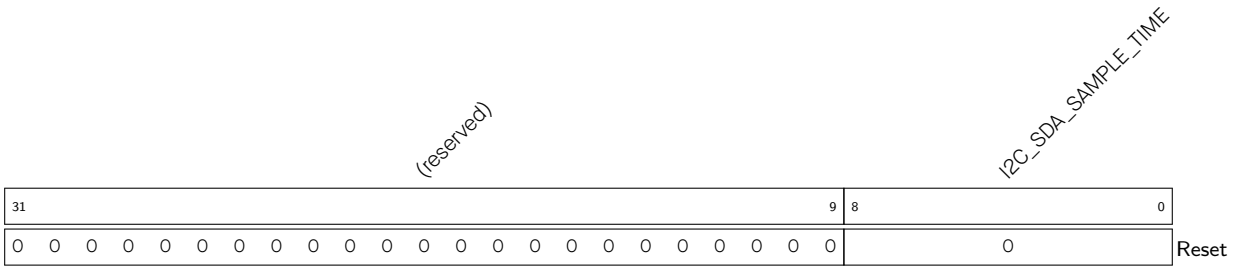


I2C_SDA_HOLD_TIME Configures the time to hold the data after the falling edge of SCL.

Measurement unit: I2C_SCLK clock cycles

(R/W)

Register 47.3. I2C_SDA_SAMPLE_REG (0x0034)



I2C_SDA_SAMPLE_TIME Configures the time for sampling SDA.

Measurement unit: I2C_SCLK clock cycles

(R/W)

Register 47.4. I2C_SCL_HIGH_PERIOD_REG (0x0038)



I2C_SCL_HIGH_PERIOD Configures for how long SCL remains high in master mode.

Measurement unit: I2C_SCLK clock cycles

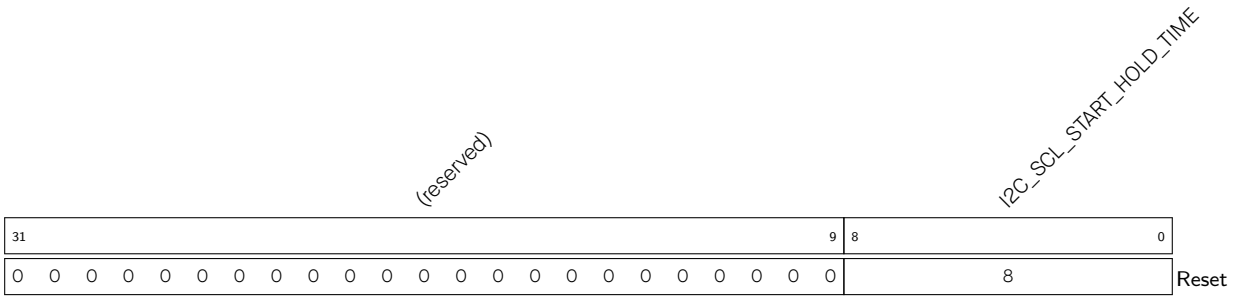
(R/W)

I2C_SCL_WAIT_HIGH_PERIOD Configures the SCL_FSM's waiting period for SCL high level in master mode.

Measurement unit: I2C_SCLK clock cycles

(R/W)

Register 47.5. I2C_SCL_START_HOLD_REG (0x0040)

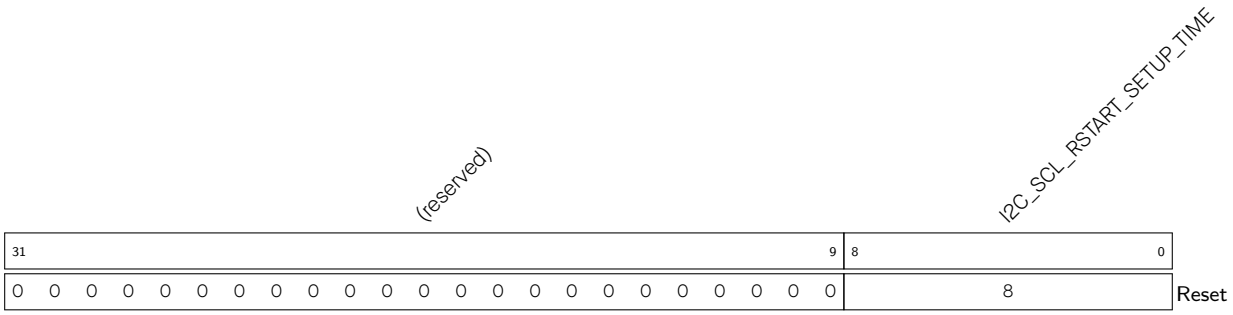


I2C_SCL_START_HOLD_TIME Configures the time between the falling edge of SDA and the falling edge of SCL for a START condition.

Measurement unit: I2C_SCLK clock cycles

(R/W)

Register 47.6. I2C_SCL_RSTART_SETUP_REG (0x0044)

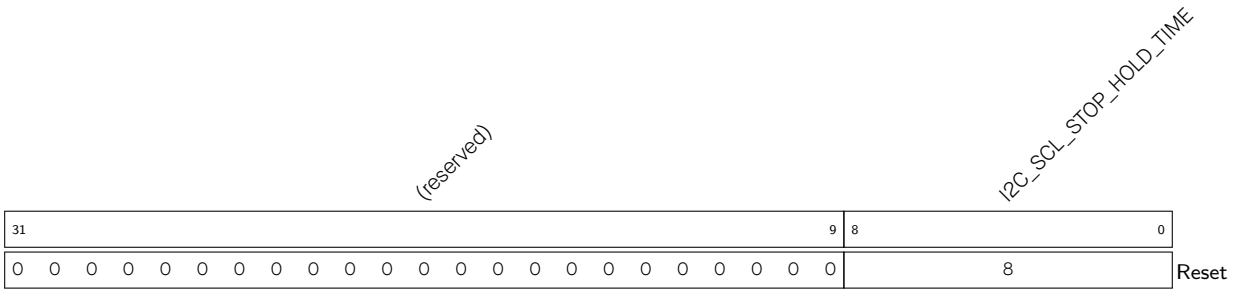


I2C_SCL_RSTART_SETUP_TIME Configures the time between the positive edge of SCL and the negative edge of SDA for a RESTART condition.

Measurement unit: I2C_SCLK clock cycles

(R/W)

Register 47.7. I2C_SCL_STOP_HOLD_REG (0x0048)

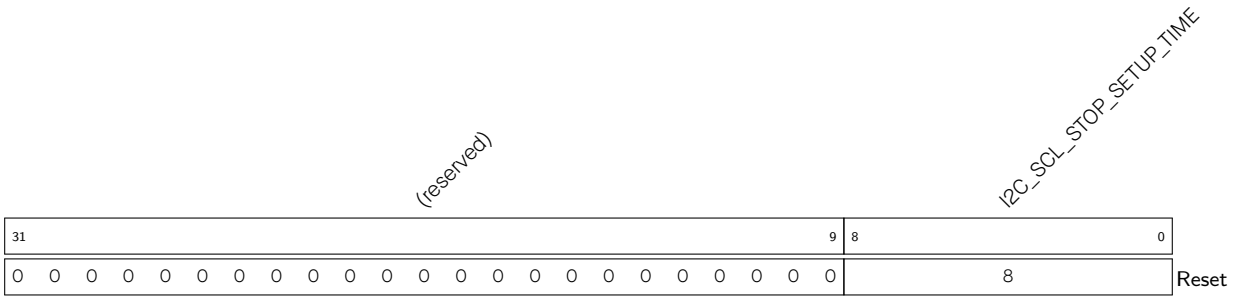


I2C_SCL_STOP_HOLD_TIME Configures the delay after the STOP condition.

Measurement unit: I2C_SCLK clock cycles

(R/W)

Register 47.8. I2C_SCL_STOP_SETUP_REG (0x004C)

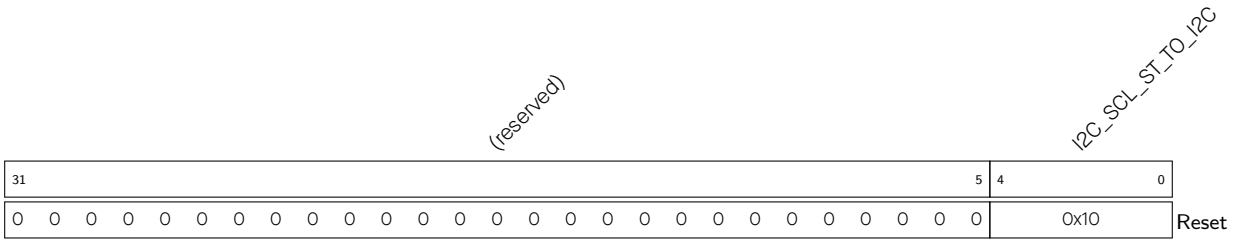


I2C_SCL_STOP_SETUP_TIME Configures the time between the rising edge of SCL and the rising edge of SDA.

Measurement unit: I2C_SCLK clock cycles

(R/W)

Register 47.9. I2C_SCL_ST_TIME_OUT_REG (0x0078)



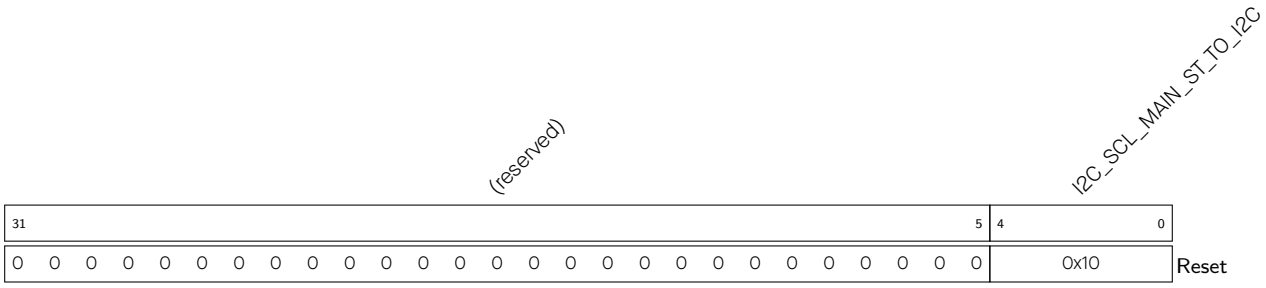
I2C_SCL_ST_TO_I2C Configures the threshold value of SCL_FSM state unchanged period. It should be no more than 23, more than 1.

$$\text{Maximum Clock Cycle Threshold} = 2^{\text{I2C_SCL_ST_TO_I2C}+1}-1$$

Measurement unit: I2C_SCLK clock cycles

(R/W)

Register 47.10. I2C_SCL_MAIN_ST_TIME_OUT_REG (0x007C)

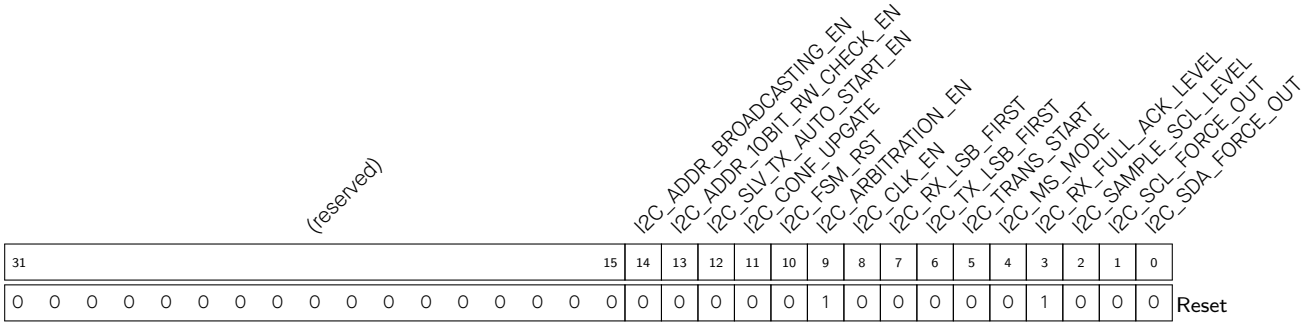


I2C_SCL_MAIN_ST_TO_I2C Configures the threshold value of SCL_MAIN_FSM state unchanged period. It should be no more than 23.

Measurement unit: I2C_SCLK clock cycles

(R/W)

Register 47.11. I2C_CTR_REG (0x0004)



I2C_SDA_FORCE_OUT Configures the SDA output mode.

0: Open drain output

1: Direct output

(R/W)

I2C_SCL_FORCE_OUT Configures the SCL output mode.

0: Open drain output

1: Direct output

(R/W)

I2C_SAMPLE_SCL_LEVEL Configures the sample mode for SDA.

0: Sample SDA data on the SCL high level

1: Sample SDA data on the SCL low level

(R/W)

I2C_RX_FULL_ACK_LEVEL Configures the ACK value that needs to be sent by the master when the received RX RAM has reached the threshold.

(R/W)

I2C_MS_MODE Configures the module as an I2C Master or Slave.

0: Slave

1: Master

(R/W)

I2C_TRANS_START Configures whether the slave starts sending the data in TX FIFO.

0: No effect

1: Start

(WT)

I2C_TX_LSB_FIRST Configures to control the sending order for data needing to be sent.

0: send data from the most significant bit

1: send data from the least significant bit

(R/W)

I2C_RX_LSB_FIRST Configures to control the storage order for received data.

0: receive data from the most significant bit

1: receive data from the least significant bit

(R/W)

Continued on the next page...

Register 47.11. I2C_CTR_REG (0x0004)

Continued from the previous page...

I2C_CLK_EN Configures whether to gate clock signal for registers.

0: Support clock only when registers are read or written to by software

1: Force clock on for registers

(R/W)

I2C_ARBITRATION_EN Configures to enable I2C bus arbitration detection.

0: No effect

1: Enable

(R/W)

I2C_FSM_RST Configures to reset the SCL_FSM.

0: No effect

1: Reset (WT)

I2C_CONF_UPGATE Configures this bit for synchronization.

0: No effect

1: Synchronize (WT)

I2C_SLV_TX_AUTO_START_EN Configures to enable slave to send data automatically

0: Disable

1: Enable

(R/W)

I2C_ADDR_10BIT_RW_CHECK_EN Configures to check if the R/W bit of 10-bit addressing consists with I2C protocol.

0: Not check

1: Check (R/W)

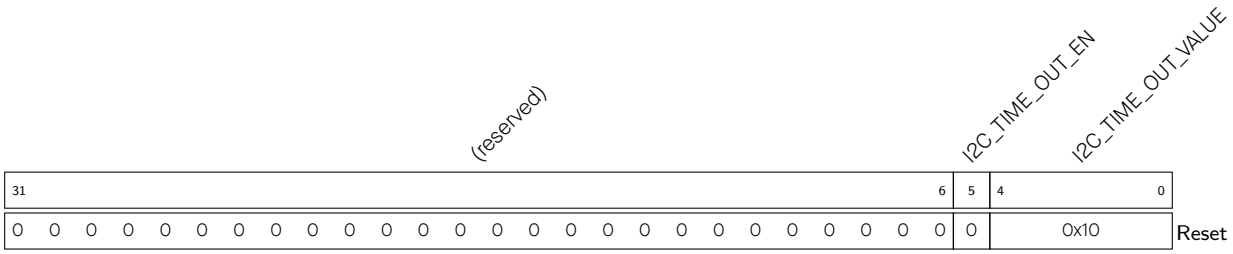
I2C_ADDR_BROADCASTING_EN Configures to support the 7-bit general call function.

0: Not support

1: Support

(R/W)

Register 47.12. I2C_TO_REG (0x000C)



I2C_TIME_OUT_VALUE Configures the timeout threshold period for SCL stuck at high or low level. The actual period is 2^(I2C_TIME_OUT_VALUE+1)−1. Measurement unit: I2C_SCLK clock cycles (R/W)

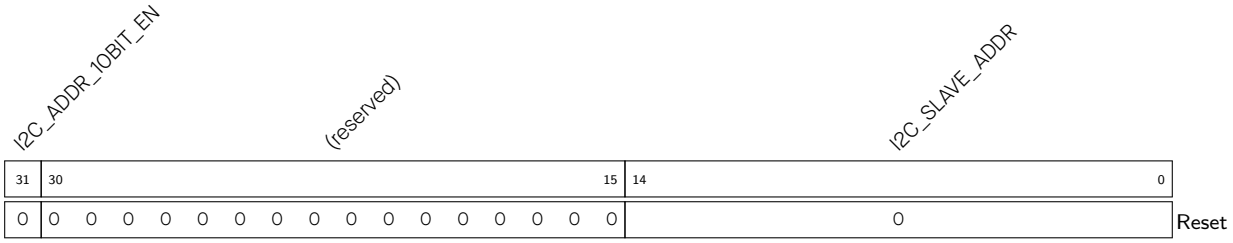
I2C_TIME_OUT_EN Configures to enable time out control.

0: No effect

1: Enable

(R/W)

Register 47.13. I2C_SLAVE_ADDR_REG (0x0010)



I2C_SLAVE_ADDR Configure the slave address of I2C Slave. (R/W)

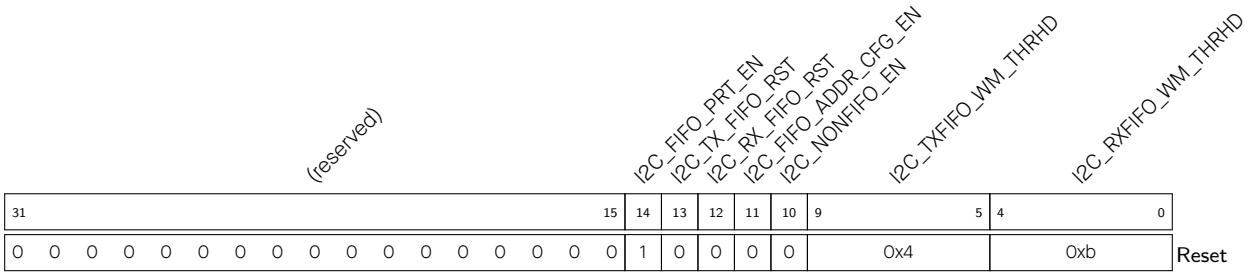
I2C_ADDR_10BIT_EN Configures to enable the slave 10-bit addressing mode in master mode.

0: No effect

1: Enable

(R/W)

Register 47.14. I2C_FIFO_CONF_REG (0x0018)



I2C_RXFIFO_WM_THRHD Configures the watermark threshold of RX FIFO in non-FIFO access mode. When I2C_FIFO_PRT_EN is 1 and RX FIFO counter is bigger than I2C_RXFIFO_WM_THRHD[4:0], I2C_RXFIFO_WM_INT_RAW bit will be valid. (R/W)

I2C_TXFIFO_WM_THRHD Configures the watermark threshold of TX FIFO in non-FIFO access mode. When I2C_FIFO_PRT_EN is 1 and TC FIFO counter is bigger than I2C_TXFIFO_WM_THRHD[4:0], I2C_TXFIFO_WM_INT_RAW bit will be valid. (R/W)

I2C_NONFIFO_EN Configures to enable APB non-FIFO access. (R/W)

I2C_FIFO_ADDR_CFG_EN Configures the slave to enable dual address mode. When this mode is enabled, the byte received after the I2C address byte represents the offset address in the I2C Slave RAM.

0: Disable

1: Enable

(R/W)

I2C_RX_FIFO_RST Configures to reset RX FIFO.

0: No effect

1: Reset (R/W)

I2C_TX_FIFO_RST Configures to reset TX FIFO.

0: No effect

1: Reset (R/W)

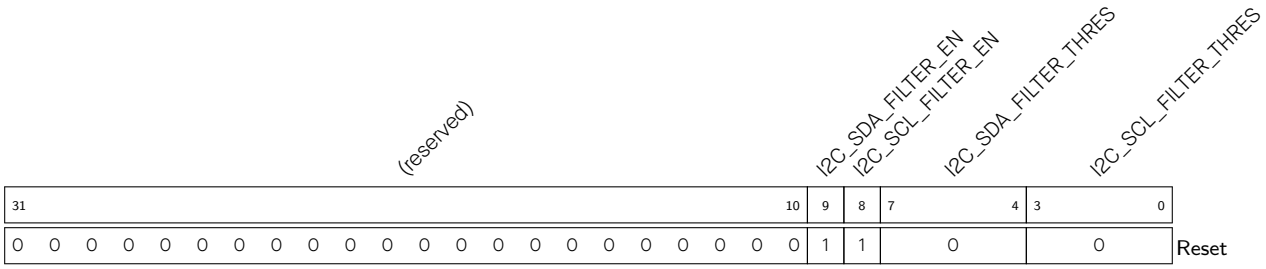
I2C_FIFO_PRT_EN Configures to enable FIFO pointer in non-FIFO access mode. This bit controls the valid bits and the TX/RX FIFO overflow, underflow, full and empty interrupts.

0: No effect

1: Enable

(R/W)

Register 47.15. I2C_FILTER_CFG_REG (0x0050)



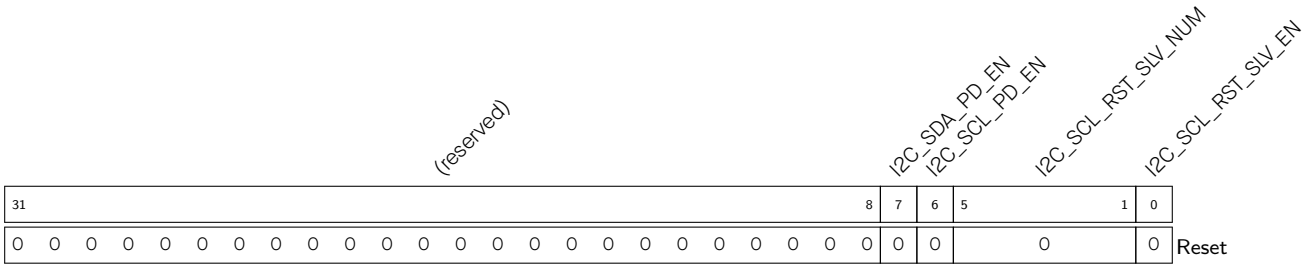
I2C_SCL_FILTER_THRES Configures the threshold pulse width to be filtered on SCL. When a pulse on the SCL input has smaller width than this register value, the I2C controller will ignore that pulse. Measurement unit: I2C_SCLK clock cycles
(R/W)

I2C_SDA_FILTER_THRES Configures the threshold pulse width to be filtered on SDA. When a pulse on the SDA input has smaller width than this register value, the I2C controller will ignore that pulse. Measurement unit: I2C_SCLK clock cycles
(R/W)

I2C_SCL_FILTER_EN Configures to enable the filter function for SCL.
0: No effect
1: Enable
(R/W)

I2C_SDA_FILTER_EN Configures to enable the filter function for SDA.
0: No effect
1: Enable
(R/W)

Register 47.16. I2C_SCL_SP_CONF_REG (0x0080)



I2C_SCL_RST_SLV_EN Configures to send out SCL pulses when I2C master is IDLE. The number of pulses equals to I2C_SCL_RST_SLV_NUM[4:0].

0: Invalid

1: Send out SCL pulses

(R/W/SC)

I2C_SCL_RST_SLV_NUM Configure the pulses of SCL generated in I2C master mode.

Valid when I2C_SCL_RST_SLV_EN is 1.

Measurement unit: I2C_SCLK clock cycles

(R/W)

I2C_SCL_PD_EN Configures to power down the I2C output SCL line.

0: Not power down.

1: Not work and power down.

Valid only when I2C_SCL_FORCE_OUT is 1. (R/W)

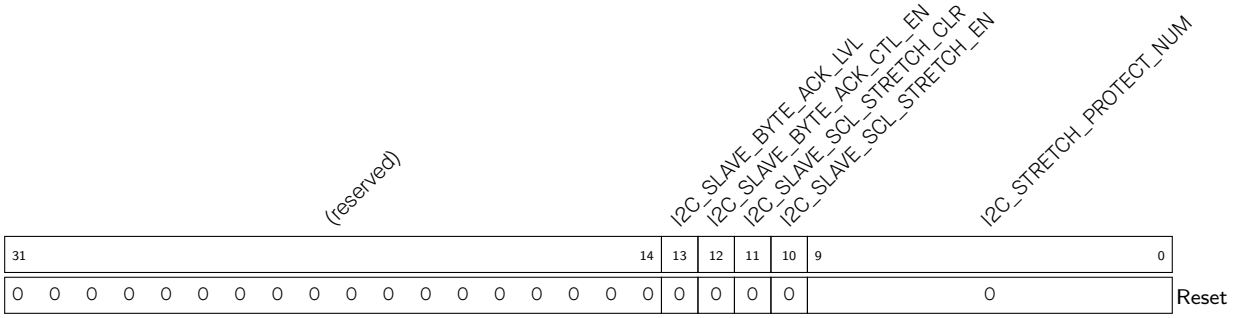
I2C_SDA_PD_EN Configures to power down the I2C output SDA line.

0: Not power down.

1: Not work and power down.

Valid only when I2C_SDA_FORCE_OUT is 1. (R/W)

Register 47.17. I2C_SCL_STRETCH_CONF_REG (0x0084)



I2C_STRETCH_PROTECT_NUM Configures the time period to release the SCL line from stretching to avoid timing violation. Usually it should be larger than the SDA setup time.

Measurement unit: I2C_SCLK clock cycles

(R/W)

I2C_SLAVE_SCL_STRETCH_EN Configures to enable slave SCL stretch function. The SCL output line will be stretched low when I2C_SLAVE_SCL_STRETCH_EN is 1 and a stretch event happens. The stretch cause can be seen in I2C_STRETCH_CAUSE.

0: Disable

1: Enable

(R/W)

I2C_SLAVE_SCL_STRETCH_CLR Configures to clear the I2C slave SCL stretch function.

0: No effect

1: Clear

(WT)

I2C_SLAVE_BYTE_ACK_CTL_EN Configures to enable the function for the slave to control ACK level.

0: Disable

1: Enable

(R/W)

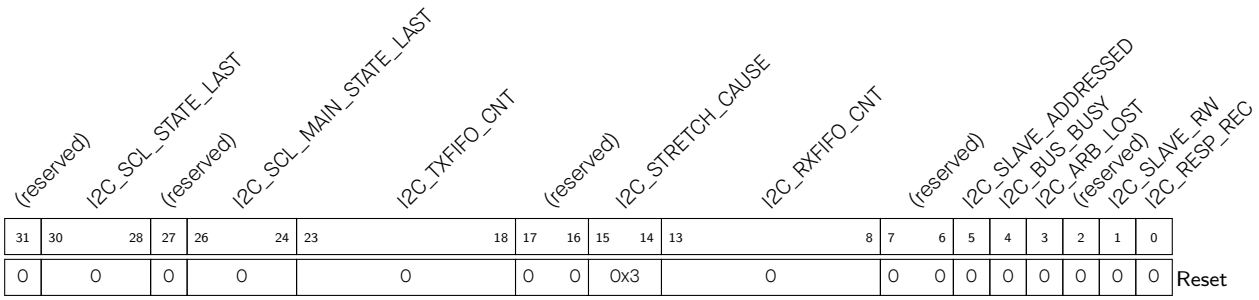
I2C_SLAVE_BYTE_ACK_LVL Configures the ACK level when slave controlling ACK level function is enabled.

0: Low level

1: High level

(R/W)

Register 47.18. I2C_SR_REG (0x0008)



I2C_RESP_REC Represents the received ACK value in master mode or slave mode.

0: ACK

1: NACK. (RO)

I2C_SLAVE_RW Represents the transfer direction in slave mode.

0: Master writes to slave. 1: Master reads from slave

(RO)

I2C_ARB_LOST Represents whether the I2C controller loses control of SCL line.

0: No arbitration lost

1: Arbitration lost

(RO)

I2C_BUS_BUSY Represents the I2C bus state.

0: The I2C bus is in an idle state.

1: The I2C bus is busy transferring data

(RO)

I2C_SLAVE_ADDRESSED Represents whether the address sent by the master is equal to the address of the slave.

Valid only when the module is configured as an I2C Slave.

0: Not equal

1: Equal

(RO)

I2C_RXFIFO_CNT Represents the number of data bytes received in RAM. (RO)

I2C_STRETCH_CAUSE Represents the cause of SCL clocking stretching in slave mode.

0: Stretching SCL low when the master starts to read data.

1: Stretching SCL low when I2C TX FIFO is empty in slave mode.

2: Stretching SCL low when I2C RX FIFO is full in slave mode.

3: Invalid

(RO)

I2C_TXFIFO_CNT Represents the number of data bytes to be sent. (RO)

Continued on the next page...

Register 47.18. I2C_SR_REG (0x0008)

Continued from the previous page...

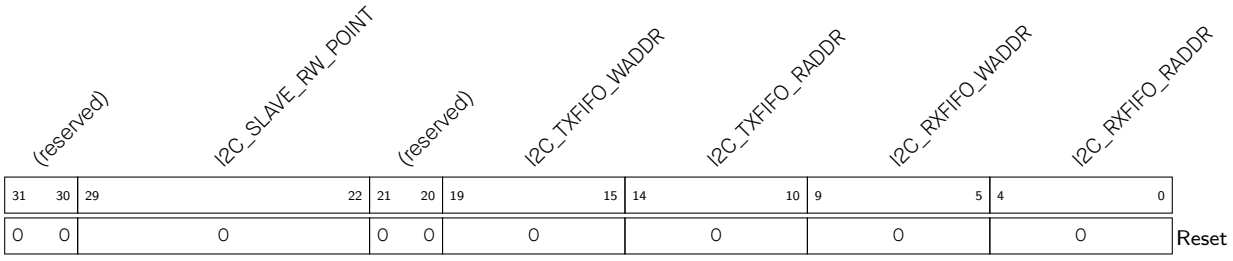
I2C_SCL_MAIN_STATE_LAST Represents the states of the I2C module state machine.

0: Idle
1: Address shift
2: ACK address
3: Rx data
4: Tx data
5: Send ACK
6: Wait ACK
7: Invalid
(RO)

I2C_SCL_STATE_LAST Represents the states of the state machine used to produce SCL.

0: Idle
1: Start
2: Negative edge
3: Low
4: Positive edge
5: High
6: Stop
7: Invalid
(RO)

Register 47.19. I2C_FIFO_ST_REG (0x0014)



I2C_RXFIFO_RADDR Represents the offset address of the APB reading from RX FIFO. (RO)

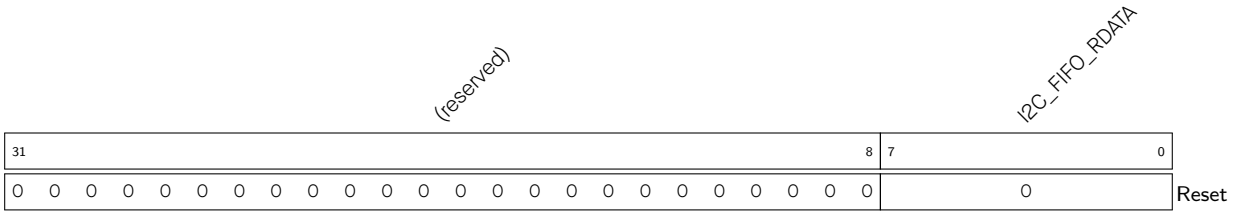
I2C_RXFIFO_WADDR Represents the offset address of the I2C module receiving data and writing to RX FIFO. (RO)

I2C_TXFIFO_RADDR Represents the offset address of the I2C module reading from TX FIFO. (RO)

I2C_TXFIFO_WADDR Represents the offset address of the APB bus writing to TX FIFO. (RO)

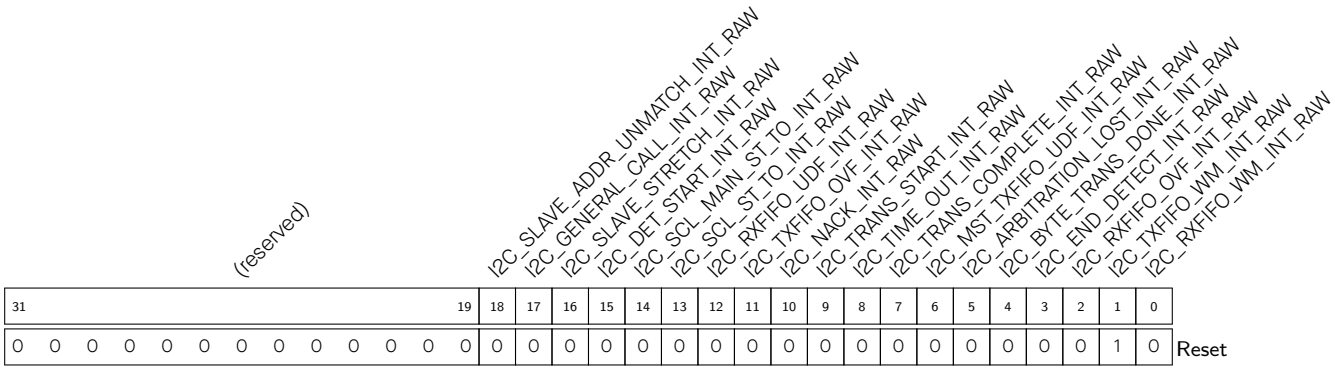
I2C_SLAVE_RW_POINT Represents the offset address in the I2C Slave RAM addressed by I2C Master when in I2C slave mode. (RO)

Register 47.20. I2C_DATA_REG (0x001C)



I2C_FIFO_RDATA Represents the value of RX FIFO read data. (RO)

Register 47.21. I2C_INT_RAW_REG (0x0020)



I2C_RXFIFO_WM_INT_RAW The raw interrupt status of I2C_RXFIFO_WM_INT interrupt. (R/SS/WTC)

I2C_TXFIFO_WM_INT_RAW The raw interrupt status of I2C_TXFIFO_WM_INT interrupt. (R/SS/WTC)

I2C_RXFIFO_OVF_INT_RAW The raw interrupt status of I2C_RXFIFO_OVF_INT interrupt. (R/SS/WTC)

I2C_END_DETECT_INT_RAW The raw interrupt status of the I2C_END_DETECT_INT interrupt. (R/SS/WTC)

I2C_BYTE_TRANS_DONE_INT_RAW The raw interrupt status of the I2C_END_DETECT_INT interrupt. (R/SS/WTC)

I2C_ARBITRATION_LOST_INT_RAW The raw interrupt status of the I2C_ARBITRATION_LOST_INT interrupt. (R/SS/WTC)

I2C_MST_TXFIFO_UDF_INT_RAW The raw interrupt status of I2C_TRANS_COMPLETE_INT interrupt. (R/SS/WTC)

I2C_TRANS_COMPLETE_INT_RAW The raw interrupt status of the I2C_TRANS_COMPLETE_INT interrupt. (R/SS/WTC)

Continued on the next page...

Register 47.21. I2C_INT_RAW_REG (0x0020)

Continued from the previous page...

I2C_TIME_OUT_INT_RAW The raw interrupt status of the I2C_TIME_OUT_INT interrupt. (R/SS/WTC)

I2C_TRANS_START_INT_RAW The raw interrupt status of the I2C_TRANS_START_INT interrupt. (R/SS/WTC)

I2C_NACK_INT_RAW The raw interrupt status of I2C_SLAVE_STRETCH_INT interrupt. (R/SS/WTC)

I2C_TXFIFO_OVF_INT_RAW The raw interrupt status of I2C_TXFIFO_OVF_INT interrupt. (R/SS/WTC)

I2C_RXFIFO_UDF_INT_RAW The raw interrupt status of I2C_RXFIFO_UDF_INT interrupt. (R/SS/WTC)

I2C_SCL_ST_TO_INT_RAW The raw interrupt status of I2C_SCL_ST_TO_INT interrupt. (R/SS/WTC)

I2C_SCL_MAIN_ST_TO_INT_RAW The raw interrupt status of I2C_SCL_MAIN_ST_TO_INT interrupt. (R/SS/WTC)

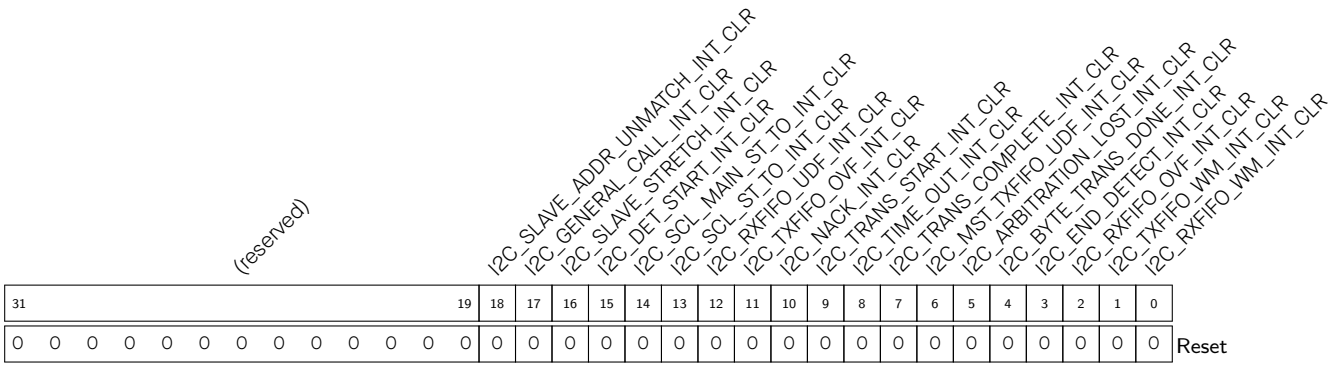
I2C_DET_START_INT_RAW The raw interrupt status of I2C_DET_START_INT interrupt. (R/SS/WTC)

I2C_SLAVE_STRETCH_INT_RAW The raw interrupt status of I2C_SLAVE_STRETCH_INT interrupt. (R/SS/WTC)

I2C_GENERAL_CALL_INT_RAW The raw interrupt status of I2C_GENARAL_CALL_INT interrupt. (R/SS/WTC)

I2C_SLAVE_ADDR_UNMATCH_INT_RAW The raw interrupt status of I2C_SLAVE_ADDR_UNMATCH_INT_RAW interrupt. (R/SS/WTC)

Register 47.22. I2C_INT_CLR_REG (0x0024)



I2C_RXFIFO_WM_INT_CLR Write 1 to clear I2C_RXFIFO_WM_INT interrupt. (WT)

I2C_TXFIFO_WM_INT_CLR Write 1 to clear I2C_TXFIFO_WM_INT interrupt. (WT)

I2C_RXFIFO_OVF_INT_CLR Write 1 to clear I2C_RXFIFO_OVF_INT interrupt. (WT)

I2C_END_DETECT_INT_CLR Write 1 to clear the I2C_END_DETECT_INT interrupt. (WT)

I2C_BYTE_TRANS_DONE_INT_CLR Write 1 to clear the I2C_BYTE_TRANS_DONE_INT interrupt. (WT)

I2C_ARBITRATION_LOST_INT_CLR Write 1 to clear the I2C_ARBITRATION_LOST_INT interrupt. (WT)

I2C_MST_TXFIFO_UDF_INT_CLR Write 1 to clear I2C_MST_TXFIFO_UDF_INT interrupt. (WT)

I2C_TRANS_COMPLETE_INT_CLR Write 1 to clear the I2C_TRANS_COMPLETE_INT interrupt. (WT)

I2C_TIME_OUT_INT_CLR Write 1 to clear the I2C_TIME_OUT_INT interrupt. (WT)

I2C_TRANS_START_INT_CLR Write 1 to clear the I2C_TRANS_START_INT interrupt. (WT)

I2C_NACK_INT_CLR Write 1 to clear I2C_NACK_INT interrupt. (WT)

I2C_TXFIFO_OVF_INT_CLR Write 1 to clear I2C_TXFIFO_OVF_INT interrupt. (WT)

I2C_RXFIFO_UDF_INT_CLR Write 1 to clear I2C_RXFIFO_UDF_INT interrupt. (WT)

I2C_SCL_ST_TO_INT_CLR Write 1 to clear I2C_SCL_ST_TO_INT interrupt. (WT)

I2C_SCL_MAIN_ST_TO_INT_CLR Write 1 to clear I2C_SCL_MAIN_ST_TO_INT interrupt. (WT)

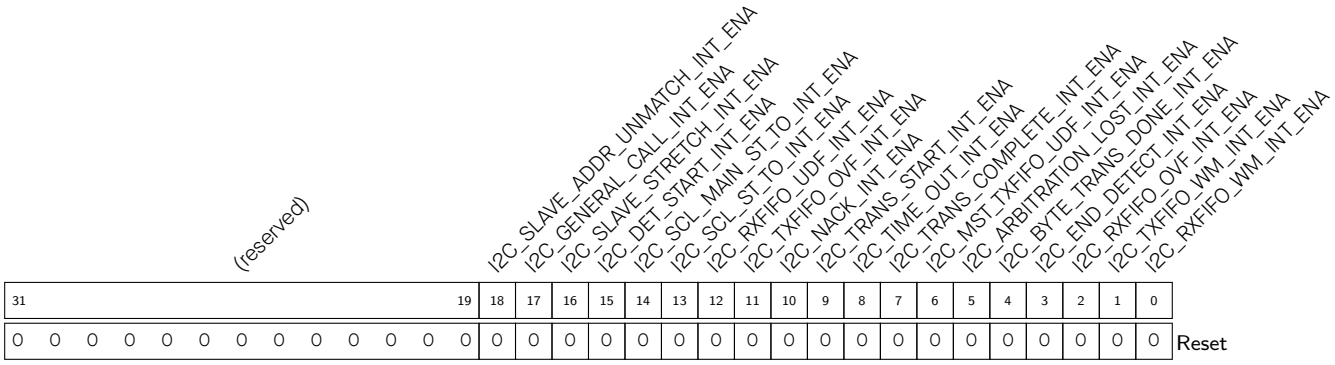
I2C_DET_START_INT_CLR Write 1 to clear I2C_DET_START_INT interrupt. (WT)

I2C_SLAVE_STRETCH_INT_CLR Write 1 to clear I2C_SLAVE_STRETCH_INT interrupt. (WT)

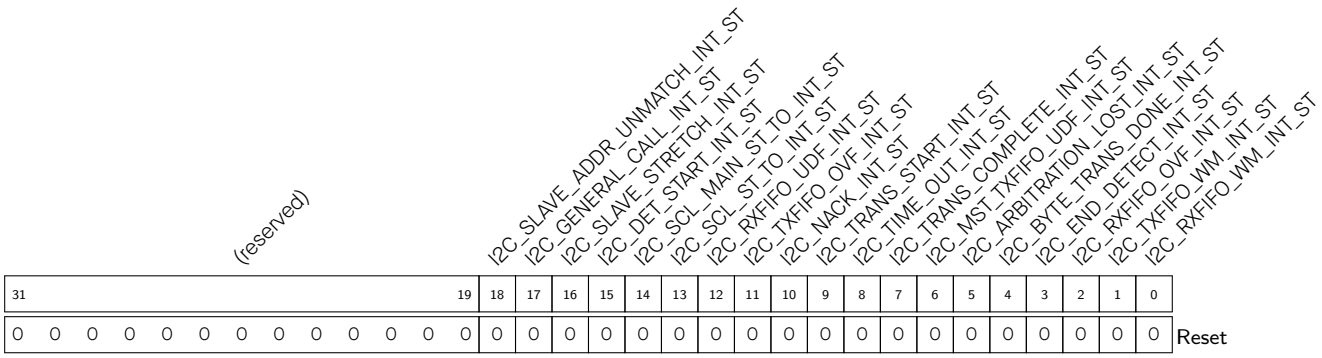
I2C_GENERAL_CALL_INT_CLR Write 1 to clear I2C_GENARAL_CALL_INT interrupt. (WT)

I2C_SLAVE_ADDR_UNMATCH_INT_CLR Write 1 to clear I2C_SLAVE_ADDR_UNMATCH_INT_RAW interrupt. (WT)

Register 47.23. I2C_INT_ENA_REG (0x0028)


(reserved)
I2C_SLAVE_ADDR_UNMATCH_INT_ENA
I2C_GENERAL_CALL_INT_ENA
I2C_SLAVE_STRETCH_INT_ENA
I2C_DET_START_INT_ENA
I2C_SCL_MAIN_ST_TO_INT_ENA
I2C_SCL_ST_TO_INT_ENA
I2C_RXFIFO_UDF_INT_ENA
I2C_TXFIFO_OVF_INT_ENA
I2C_NACK_INT_ENA
I2C_TRANS_START_INT_ENA
I2C_TIME_OUT_INT_ENA
I2C_TRANS_COMPLETE_INT_ENA
I2C_MST_TXFIFO_UDF_INT_ENA
I2C_ARBITRATION_LOST_INT_ENA
I2C_BYTE_TRANS_DONE_INT_ENA
I2C_END_DETECT_INT_ENA
I2C_RXFIFO_OVF_INT_ENA
I2C_TXFIFO_WM_INT_ENA
I2C_RXFIFO_WM_INT_ENA
31 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0

Register 47.24. I2C_INT_STATUS_REG (0x002C)


(reserved)
I2C_SLAVE_ADDR_UNMATCH_INT_ST
I2C_GENERAL_CALL_INT_ST
I2C_SLAVE_STRETCH_INT_ST
I2C_DET_START_INT_ST
I2C_SCL_MAIN_ST_TO_INT_ST
I2C_SCL_ST_TO_INT_ST
I2C_RXFIFO_UDF_INT_ST
I2C_TXFIFO_OVF_INT_ST
I2C_NACK_INT_ST
I2C_TRANS_START_INT_ST
I2C_TIME_OUT_INT_ST
I2C_TRANS_COMPLETE_INT_ST
I2C_MST_TXFIFO_UDF_INT_ST
I2C_ARBITRATION_LOST_INT_ST
I2C_BYTE_TRANS_DONE_INT_ST
I2C_END_DETECT_INT_ST
I2C_RXFIFO_OVF_INT_ST
I2C_TXFIFO_WM_INT_ST
I2C_RXFIFO_WM_INT_ST
31 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0

Register 47.24. I2C_INT_STATUS_REG (0x002C)

Continued from the previous page...

I2C_RXFIFO_UDF_INT_ST The masked interrupt status of I2C_RXFIFO_UDF_INT interrupt. (RO)

I2C_SCL_ST_TO_INT_ST The masked interrupt status of I2C_SCL_ST_TO_INT interrupt. (RO)

I2C_SCL_MAIN_ST_TO_INT_ST The masked interrupt status of I2C_SCL_MAIN_ST_TO_INT interrupt. (RO)

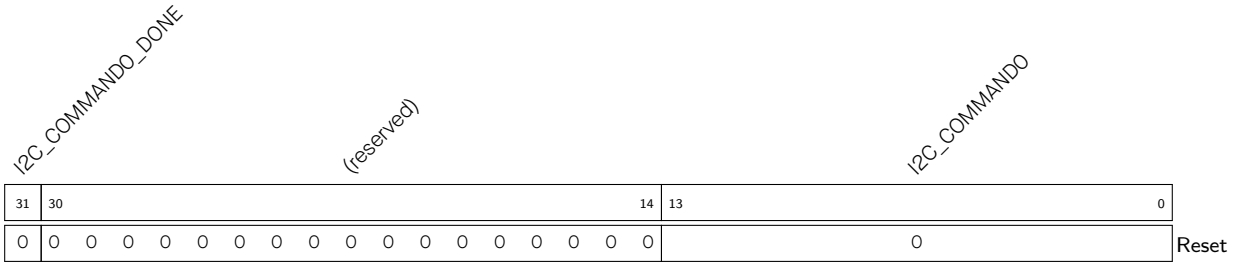
I2C_DET_START_INT_ST The masked interrupt status of I2C_DET_START_INT interrupt. (RO)

I2C_SLAVE_STRETCH_INT_ST The masked interrupt status of I2C_SLAVE_STRETCH_INT interrupt. (RO)

I2C_GENERAL_CALL_INT_ST The masked interrupt status of I2C_GENARAL_CALL_INT interrupt. (RO)

I2C_SLAVE_ADDR_UNMATCH_INT_ST The masked interrupt status of I2C_SLAVE_ADDR_UNMATCH_INT interrupt. (RO)

Register 47.25. I2C_COMD0_REG (0x0058)



I2C_COMMAND0 Configures command 0.

It consists of three parts:

op_code is the command

1: WRITE

2: STOP

3: READ

4: END

6: RSTART

Byte_num represents the number of bytes that need to be sent or received.

ack_check_en, ack_exp, and ack are used to control the ACK bit. See I2C cmd structure 47.4-2 for more information.

(R/W)

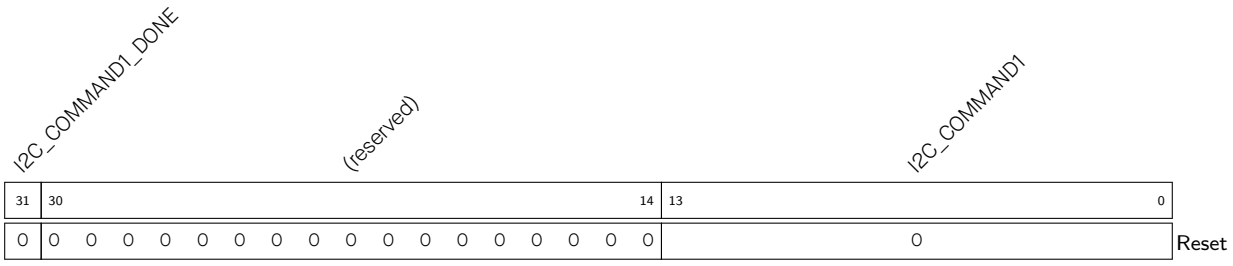
I2C_COMMAND0_DONE Represents whether command 0 is done in I2C Master mode.

0: Not done

1: Done

(R/W/SS)

Register 47.26. I2C_COMD1_REG (0x005C)



I2C_COMMAND1 Configures command 1.

See details in I2C_CMD0_REG[13:0]. (R/W)

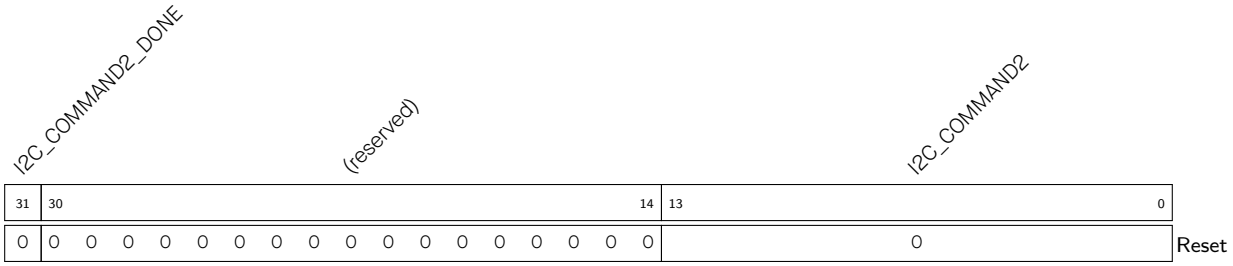
I2C_COMMAND1_DONE Represents whether command 1 is done in I2C Master mode.

0: Not done

1: Done

(R/W/SS)

Register 47.27. I2C_COMD2_REG (0x0060)



I2C_COMMAND2 Configures command 2. See details in I2C_CMD0_REG[13:0]. (R/W)

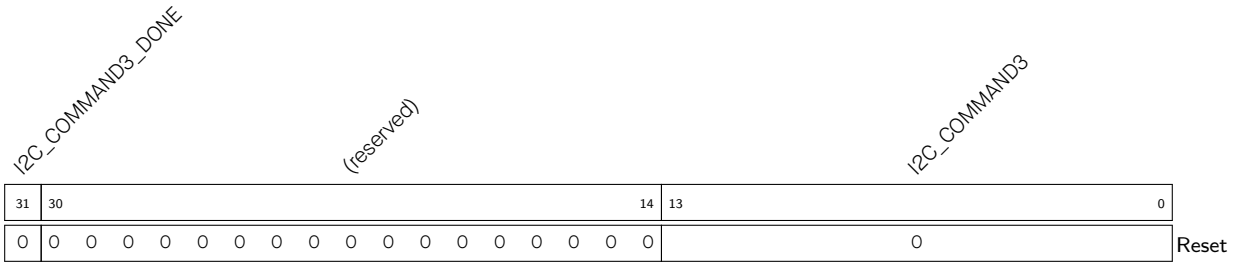
I2C_COMMAND2_DONE Represents whether command 2 is done in I2C Master mode.

0: Not done

1: Done

(R/W/SS)

Register 47.28. I2C_COMD3_REG (0x0064)



I2C_COMMAND3 Configures command 3. See details in I2C_CMD0_REG[13:0]. (R/W)

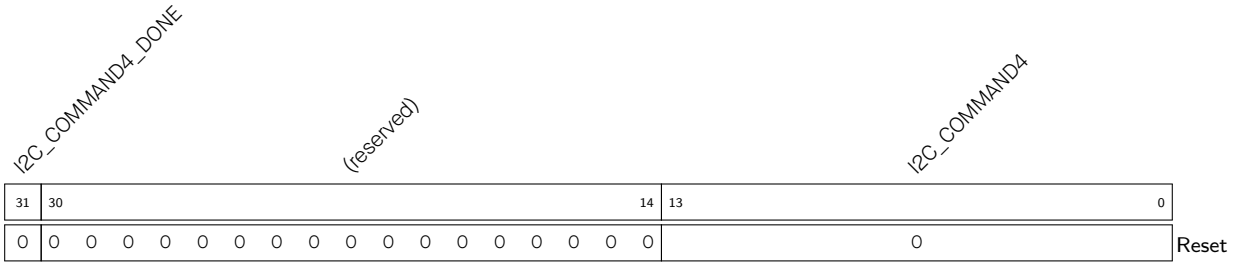
I2C_COMMAND3_DONE Represents whether command 3 is done in I2C Master mode.

0: Not done

1: Done

(R/W/SS)

Register 47.29. I2C_COMD4_REG (0x0068)



I2C_COMMAND4 Configures command 4. See details in I2C_CMD0_REG[13:0]. (R/W)

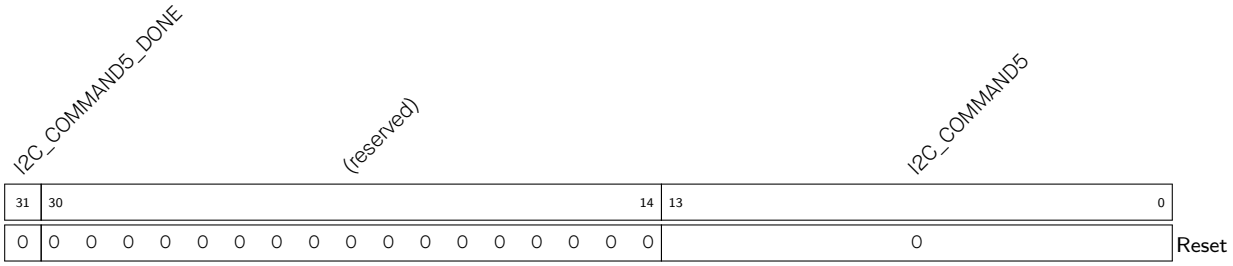
I2C_COMMAND4_DONE Represents whether command 4 is done in I2C Master mode.

0: Not done

1: Done

(R/W/SS)

Register 47.30. I2C_COMD5_REG (0x006C)



I2C_COMMAND5 Configures command 5. See details in I2C_CMD0_REG[13:0]. (R/W)

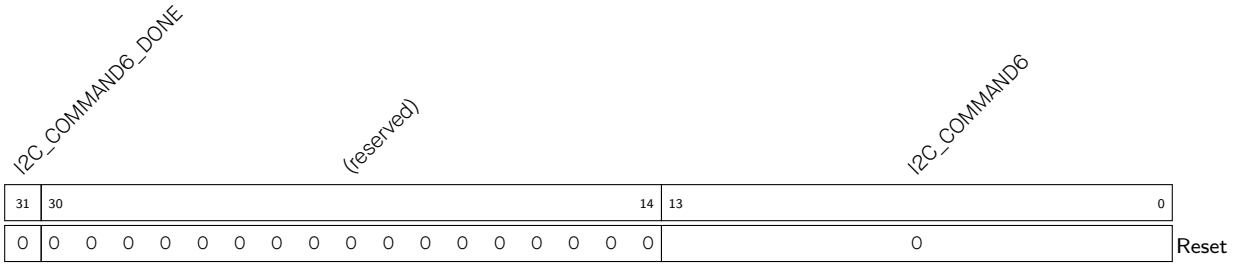
I2C_COMMAND5_DONE Represents whether command 5 is done in I2C Master mode.

0: Not done

1: Done

(R/W/SS)

Register 47.31. I2C_COMD6_REG (0x0070)



I2C_COMMAND6 Configures command 6. See details in I2C_CMD0_REG[13:0]. (R/W)

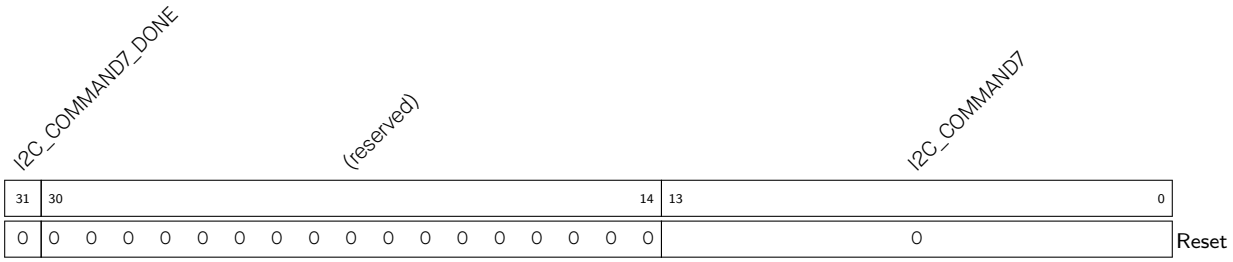
I2C_COMMAND6_DONE Represents whether command 6 is done in I2C Master mode.

0: Not done

1: Done

(R/W/SS)

Register 47.32. I2C_COMD7_REG (0x0074)



I2C_COMMAND7 Configures command 7. See details in I2C_CMD0_REG[13:0]. (R/W)

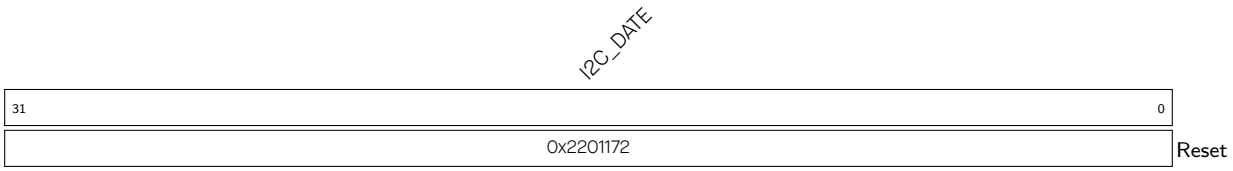
I2C_COMMAND7_DONE Represents whether command 7 is done in I2C Master mode.

0: Not done

1: Done

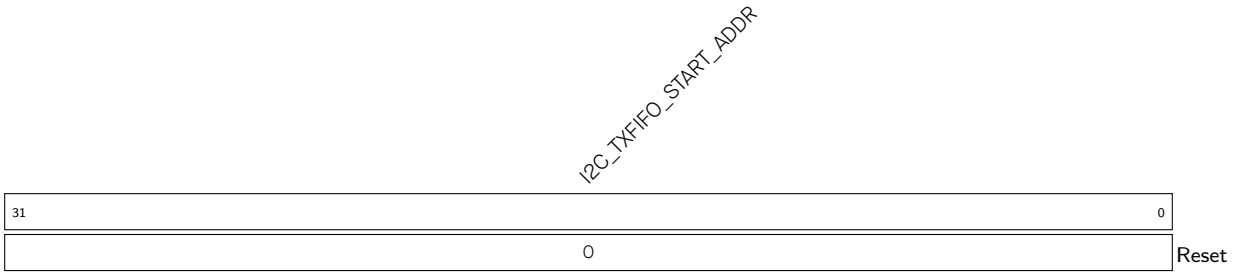
(R/W/SS)

Register 47.33. I2C_DATE_REG (0x00F8)



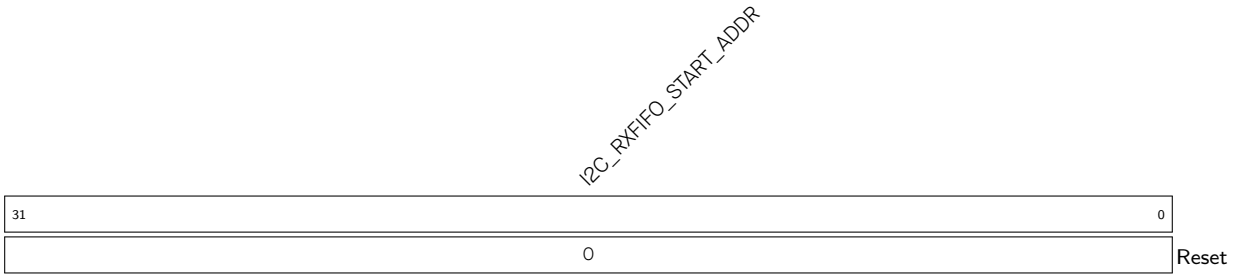
I2C_DATE Version control register. (R/W)

Register 47.34. I2C_TXFIFO_START_ADDR_REG (0x0100)



I2C_TXFIFO_START_ADDR Represents the I2C TX FIFO first address. (HRO)

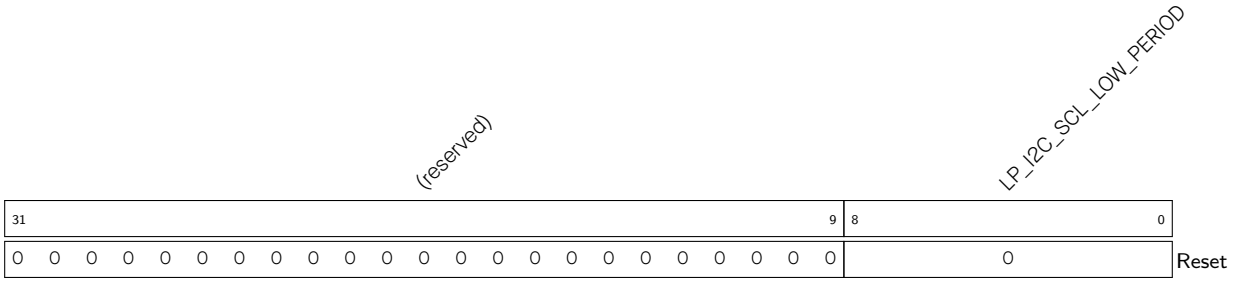
Register 47.35. I2C_RXFIFO_START_ADDR_REG (0x0180)



I2C_RXFIFO_START_ADDR Represents the I2C RX FIFO first address. (HRO)

47.9.2 LP_I2C Registers

Register 47.36. LP_I2C_SCL_LOW_PERIOD_REG (0x0000)

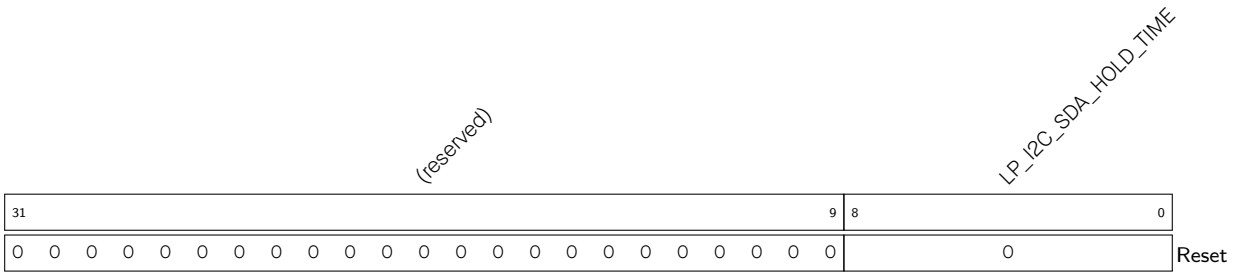


LP_I2C_SCL_LOW_PERIOD Configures the low level width of the SCL Clock in master mode.

Measurement unit: i2c_sclk

(R/W)

Register 47.37. LP_I2C_SDA_HOLD_REG (0x0030)

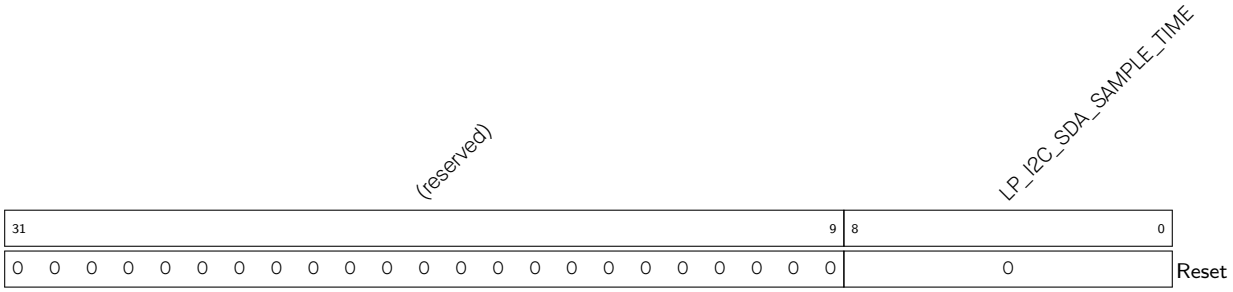


LP_I2C_SDA_HOLD_TIME Configures the time to hold the data after the falling edge of SCL.

Measurement unit: i2c_sclk

(R/W)

Register 47.38. LP_I2C_SDA_SAMPLE_REG (0x0034)

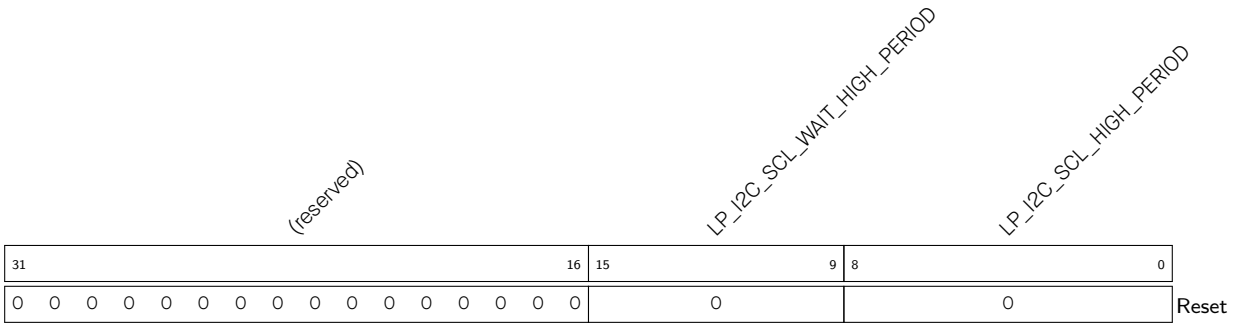


LP_I2C_SDA_SAMPLE_TIME Configures the time for sampling SDA.

Measurement unit: i2c_sclk

(R/W)

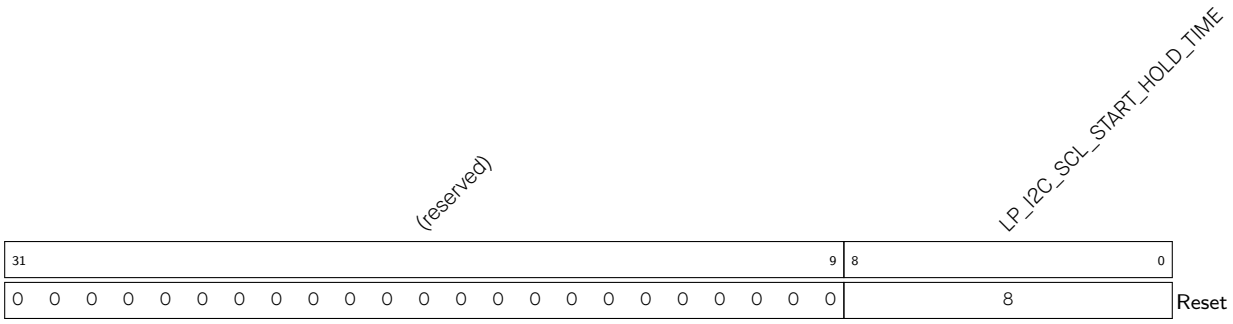
Register 47.39. LP_I2C_SCL_HIGH_PERIOD_REG (0x0038)



LP_I2C_SCL_HIGH_PERIOD Configures for how long SCL remains high in master mode.
Measurement unit: i2c_sclk
(R/W)

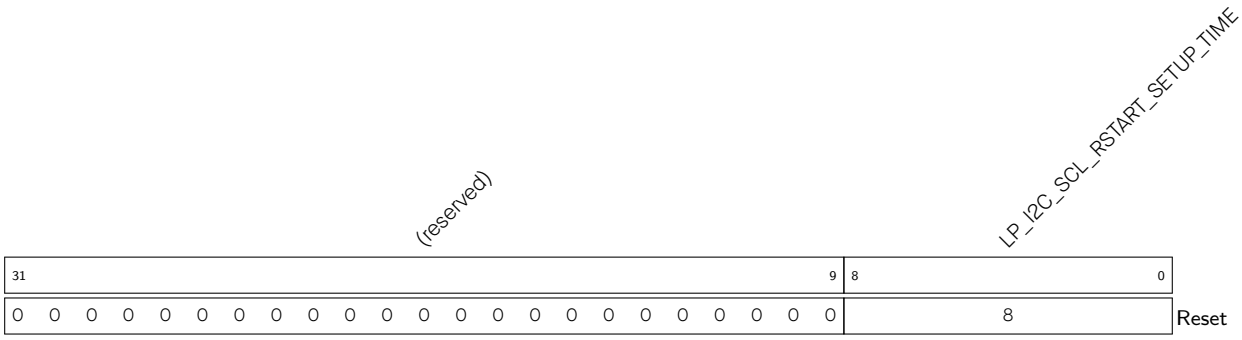
LP_I2C_SCL_WAIT_HIGH_PERIOD Configures the SCL_FSM's waiting period for SCL high level in master mode.
Measurement unit: i2c_sclk
(R/W)

Register 47.40. LP_I2C_SCL_START_HOLD_REG (0x0040)



LP_I2C_SCL_START_HOLD_TIME Configures the time between the falling edge of SDA and the falling edge of SCL for a START condition.
Measurement unit: i2c_sclk
(R/W)

Register 47.41. LP_I2C_SCL_RSTART_SETUP_REG (0x0044)

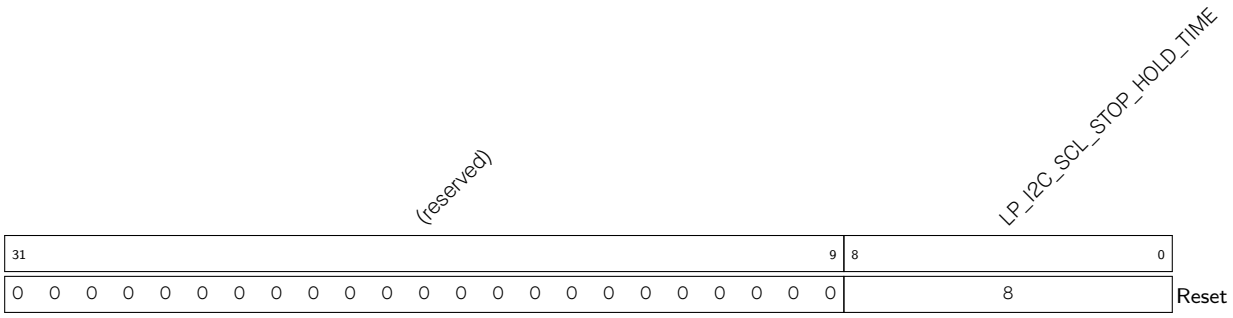


LP_I2C_SCL_RSTART_SETUP_TIME Configures the time between the positive edge of SCL and the negative edge of SDA for a RESTART condition.

Measurement unit: i2c_sclk

(R/W)

Register 47.42. LP_I2C_SCL_STOP_HOLD_REG (0x0048)

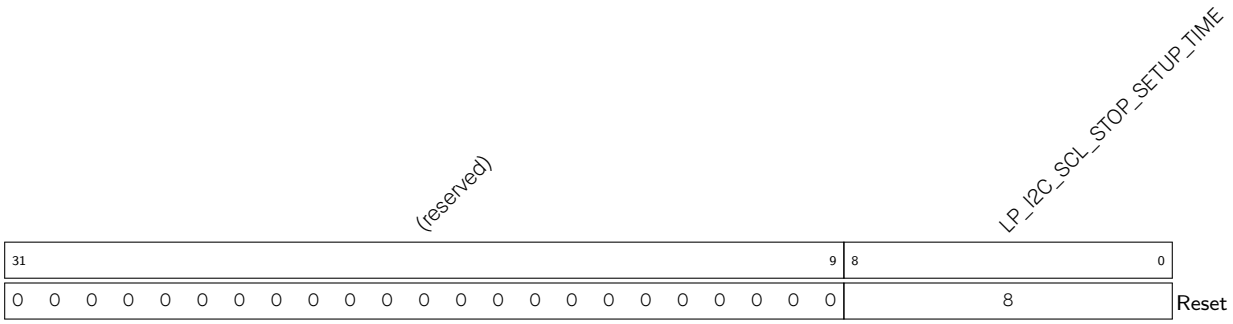


LP_I2C_SCL_STOP_HOLD_TIME Configures the delay after the STOP condition.

Measurement unit: i2c_sclk

(R/W)

Register 47.43. LP_I2C_SCL_STOP_SETUP_REG (0x004C)

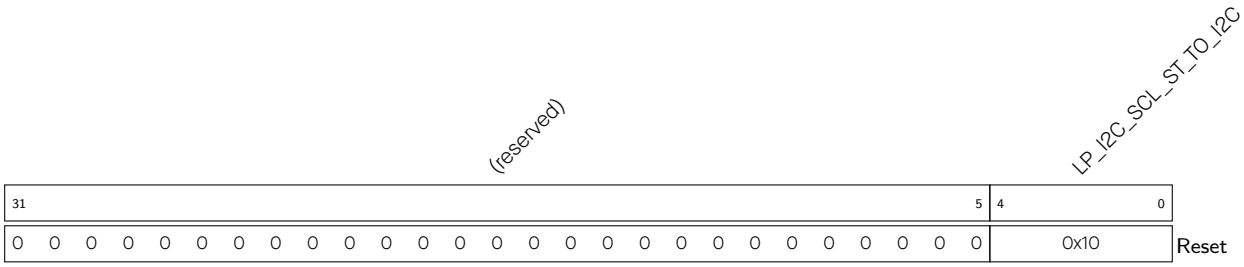


LP_I2C_SCL_STOP_SETUP_TIME Configures the time between the rising edge of SCL and the rising edge of SDA.

Measurement unit: i2c_sclk

(R/W)

Register 47.44. LP_I2C_SCL_ST_TIME_OUT_REG (0x0078)

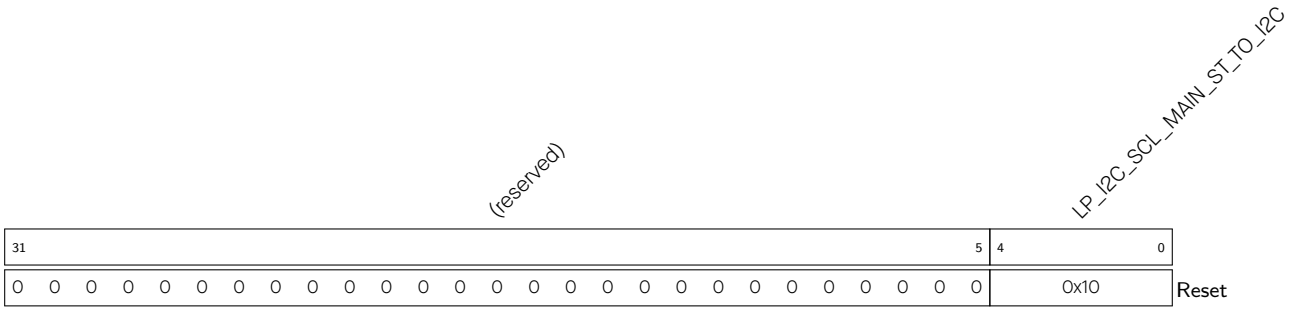


LP_I2C_SCL_ST_TO_I2C Configures the threshold value of SCL_FSM state unchanged period. It should be no more than 23.

Measurement unit: i2c_sclk

(R/W)

Register 47.45. LP_I2C_SCL_MAIN_ST_TIME_OUT_REG (0x007C)

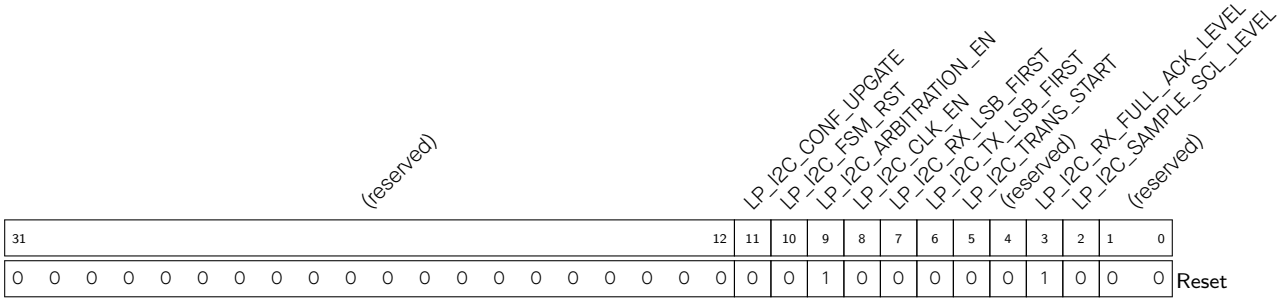


LP_I2C_SCL_MAIN_ST_TO_I2C Configures the threshold value of SCL_MAIN_FSM state unchanged period. It should be no more than 23.

Measurement unit: i2c_sclk

(R/W)

Register 47.46. LP_I2C_CTR_REG (0x0004)



LP_I2C_SAMPLE_SCL_LEVEL Configures the sample mode for SDA.

1: Sample SDA data on the SCL low level.

0: Sample SDA data on the SCL high level.

(R/W)

LP_I2C_RX_FULL_ACK_LEVEL Configures the ACK value that needs to be sent by master when the rx_fifo_cnt has reached the threshold. (R/W)

LP_I2C_TRANS_START Configures to start sending the data in txfifo for slave.

0: No effect

1: Start

(WT)

LP_I2C_TX_LSB_FIRST Configures to control the sending order for data to be sent.

1: send data from the least significant bit

0: send data from the most significant bit

(R/W)

LP_I2C_RX_LSB_FIRST Configures to control the storage order for received data.

1: receive data from the least significant bit

0: receive data from the most significant bit

(R/W)

LP_I2C_CLK_EN Configures whether to gate clock signal for registers.

0: Support clock only when registers are read or written to by software

1: Force clock on for registers.

(R/W)

LP_I2C_ARBITRATION_EN Configures to enable I2C bus arbitration detection.

0: No effect

1: Enable

(R/W)

LP_I2C_FSM_RST Configures to reset the SCL_FSM.

0: No effect

1: Reset

(WT)

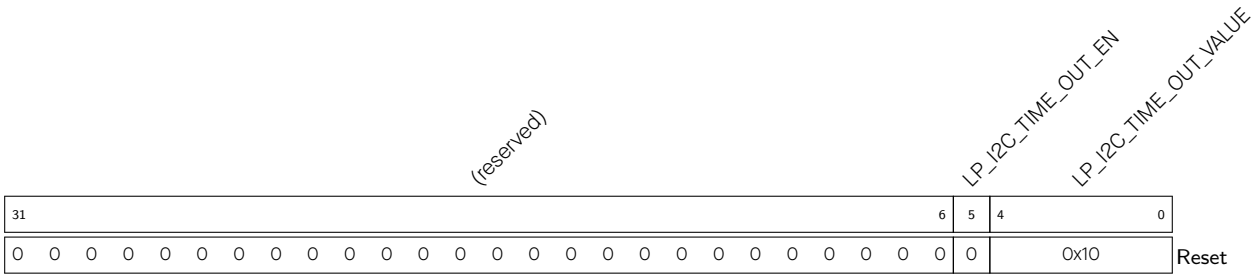
LP_I2C_CONF_UPGATE Configures this bit for synchronization.

0: No effect

1: Synchronize

(WT)

Register 47.47. LP_I2C_TO_REG (0x000C)



LP_I2C_TIME_OUT_VALUE Configures the timeout threshold period for SCL stucking at high or low level. The actual period is 2^(reg_time_out_value +1)−1.

Measurement unit: i2c_sclk.

(R/W)

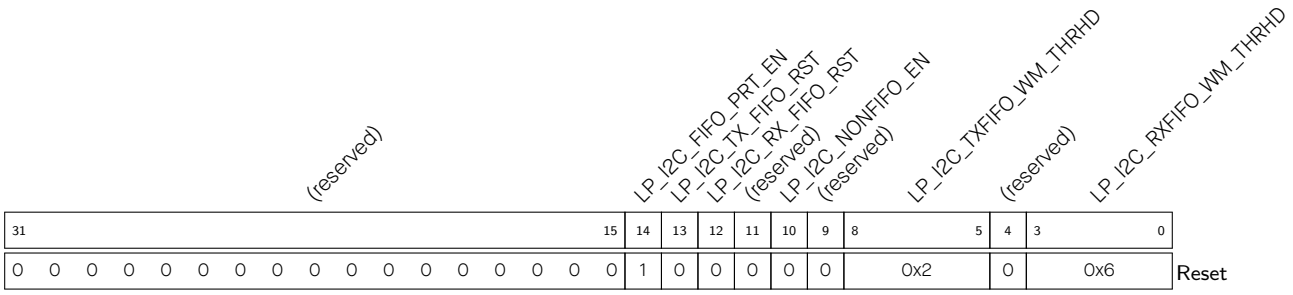
LP_I2C_TIME_OUT_EN Configures to enable time out control.

0: No effect

1: Enable

(R/W)

Register 47.48. LP_I2C_FIFO_CONF_REG (0x0018)



LP_I2C_RXFIFO_WM_THRHD Configures the water mark threshold of RXFIFO in nonfifo access mode. When LP_I2C_FIFO_PRT_EN is 1 and rx FIFO counter is bigger than LP_I2C_RXFIFO_WM_THRHD[4:0], LP_I2C_RXFIFO_WM_INT_RAW bit will be valid. (R/W)

LP_I2C_TXFIFO_WM_THRHD Configures the water mark threshold of TXFIFO in nonfifo access mode. When LP_I2C_FIFO_PRT_EN is 1 and rx FIFO counter is bigger than LP_I2C_TXFIFO_WM_THRHD[4:0], LP_I2C_TXFIFO_WM_INT_RAW bit will be valid. (R/W)

LP_I2C_NONFIFO_EN Configures to enable APB nonfifo access. (R/W)

LP_I2C_RX_FIFO_RST Configures to reset RXFIFO.

0: No effect

1: Reset

(R/W)

LP_I2C_TX_FIFO_RST Configures to reset TXFIFO. 0: No effect

1: Reset

(R/W)

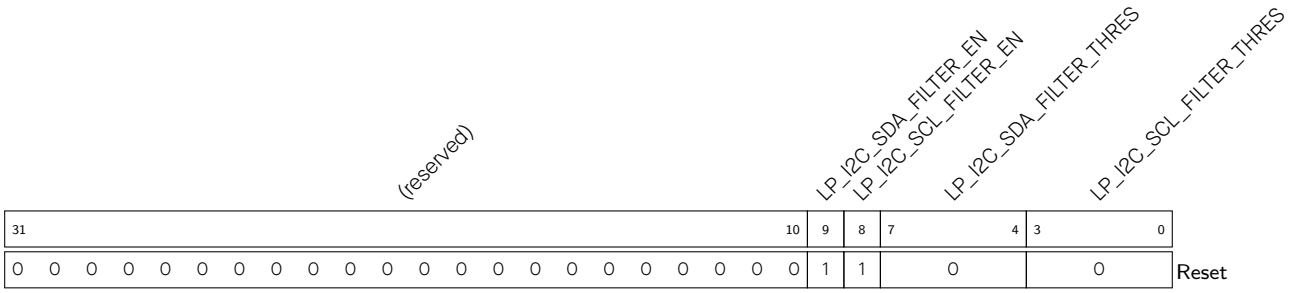
LP_I2C_FIFO_PRT_EN Configures to enable FIFO pointer in non-fifo access mode. This bit controls the valid bits and the TX/RX FIFO overflow, underflow, full and empty interrupts.

0: No effect

1: Enable

(R/W)

Register 47.49. LP_I2C_FILTER_CFG_REG (0x0050)



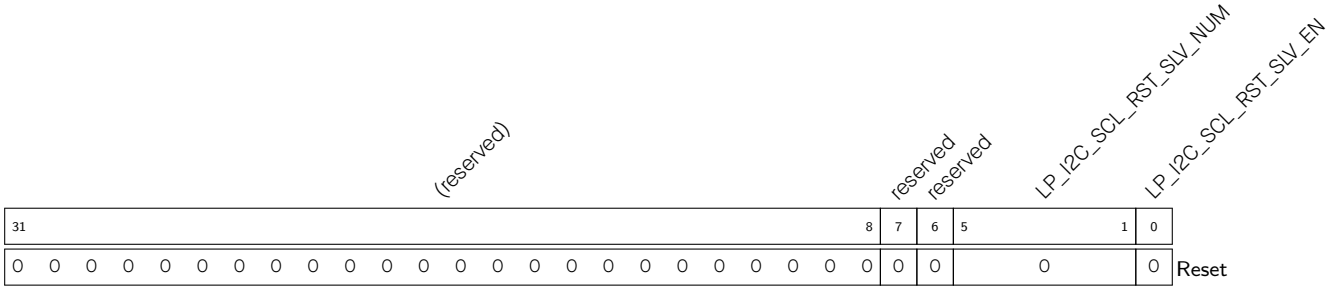
LP_I2C_SCL_FILTER_THRES Configures the threshold pulse width to be filtered on SCL. When a pulse on the SCL input has smaller width than this value, the I2C controller will ignore that pulse. Measurement unit: i2c_sclk
(R/W)

LP_I2C_SDA_FILTER_THRES Configures the threshold pulse width to be filtered on SDA. When a pulse on the SDA input has smaller width than this value, the I2C controller will ignore that pulse. Measurement unit: i2c_sclk
(R/W)

LP_I2C_SCL_FILTER_EN Configures to enable the filter function for SCL.
0: No effect
1: Enable
(R/W)

LP_I2C_SDA_FILTER_EN Configures to enable the filter function for SDA.
0: No effect
1: Enable
(R/W)

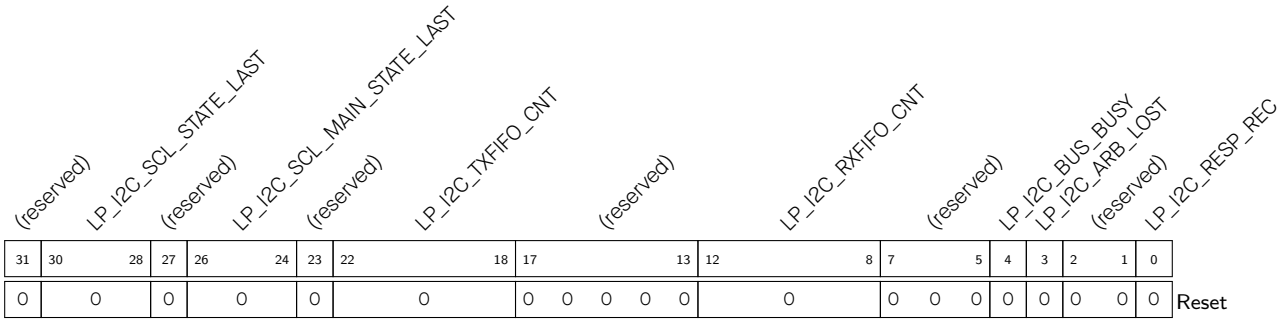
Register 47.50. LP_I2C_SCL_SP_CONF_REG (0x0080)



LP_I2C_SCL_RST_SLV_EN Configures to send out SCL pulses when I2C master is IDLE. The number of pulses equals to LP_I2C_SCL_RST_SLV_NUM[4:0]. (R/W/SC)

LP_I2C_SCL_RST_SLV_NUM Configure the pulses of SCL generated in I2C master mode. Valid when LP_I2C_SCL_RST_SLV_EN is 1.
Measurement unit: i2c_sclk
(R/W)

Register 47.51. LP_I2C_SR_REG (0x0008)



LP_I2C_RESP_REC Represents the received ACK value in master mode or slave mode.

0: ACK

1: NACK

(RO)

LP_I2C_ARB_LOST Represents whether the I2C controller loses control of SCL line.

0: No arbitration lost

1: Arbitration lost

(RO)

LP_I2C_BUS_BUSY Represents the I2C bus state.

1: The I2C bus is busy transferring data

0: The I2C bus is in idle state

(RO)

LP_I2C_RXFIFO_CNT Represents the number of data bytes received in RAM. (RO)

LP_I2C_TXFIFO_CNT Represents the number of data bytes to be sent. (RO)

LP_I2C_SCL_MAIN_STATE_LAST Represents the states of the I2C module state machine.

0: Idle

1: Address shift

2: ACK address

3: Rx data

4: Tx data

5: Send ACK

6: Wait ACK

(RO)

LP_I2C_SCL_STATE_LAST Represents the states of the state machine used to produce SCL.

0: Idle

1: Start

2: Negative edge

3: Low

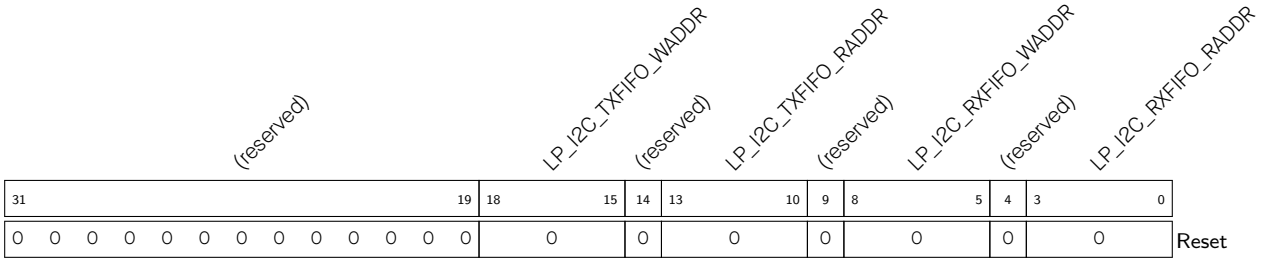
4: Positive edge

5: High

6: Stop

(RO)

Register 47.52. LP_I2C_FIFO_ST_REG (0x0014)



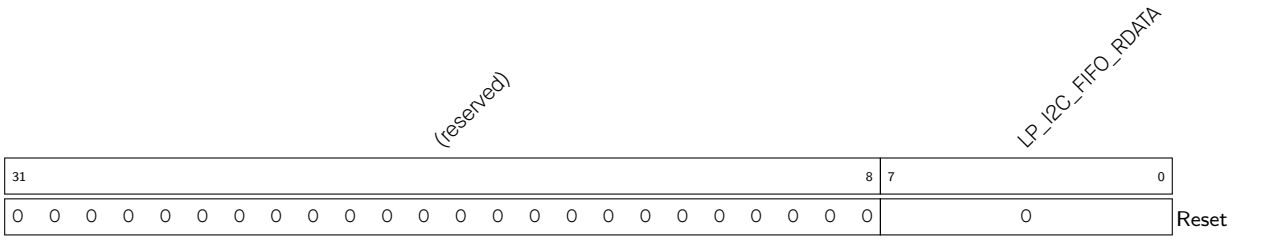
LP_I2C_RXFIFO_RADDR Represents the offset address of the APB reading from RXFIFO (RO)

LP_I2C_RXFIFO_WADDR Represents the offset address of i2c module receiving data and writing to RXFIFO. (RO)

LP_I2C_TXFIFO_RADDR Represents the offset address of i2c module reading from TXFIFO. (RO)

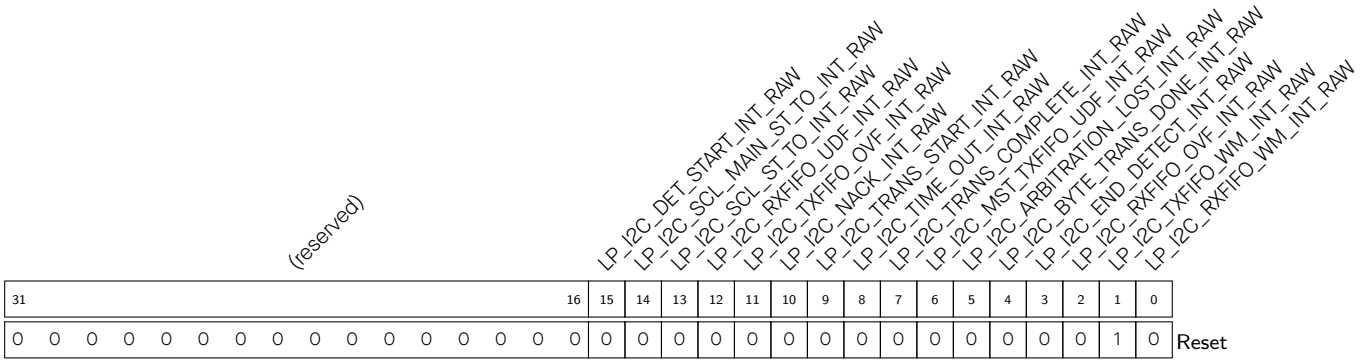
LP_I2C_TXFIFO_WADDR Represents the offset address of APB bus writing to TXFIFO. (RO)

Register 47.53. LP_I2C_DATA_REG (0x001C)



LP_I2C_FIFO_RDATA Represents the value of RXFIFO read data. (RO)

Register 47.54. LP_I2C_INT_RAW_REG (0x0020)



LP_I2C_RXFIFO_WM_INT_RAW The raw interrupt status of LP_I2C_RXFIFO_WM_INT interrupt. (R/SS/WTC)

LP_I2C_TXFIFO_WM_INT_RAW The raw interrupt status of LP_I2C_TXFIFO_WM_INT interrupt. (R/SS/WTC)

LP_I2C_RXFIFO_OVF_INT_RAW The raw interrupt status of LP_I2C_RXFIFO_OVF_INT interrupt. (R/SS/WTC)

LP_I2C_END_DETECT_INT_RAW The raw interrupt status of the LP_I2C_END_DETECT_INT interrupt. (R/SS/WTC)

LP_I2C_BYTE_TRANS_DONE_INT_RAW The raw interrupt status of the LP_I2C_END_DETECT_INT interrupt. (R/SS/WTC)

LP_I2C_ARBITRATION_LOST_INT_RAW The raw interrupt status of the LP_I2C_ARBITRATION_LOST_INT interrupt. (R/SS/WTC)

LP_I2C_MST_TXFIFO_UDF_INT_RAW The raw interrupt status of LP_I2C_TRANS_COMPLETE_INT interrupt. (R/SS/WTC)

LP_I2C_TRANS_COMPLETE_INT_RAW The raw interrupt status of the LP_I2C_TRANS_COMPLETE_INT interrupt. (R/SS/WTC)

LP_I2C_TIME_OUT_INT_RAW The raw interrupt status of the LP_I2C_TIME_OUT_INT interrupt. (R/SS/WTC)

Continued on the next page...

Register 47.54. LP_I2C_INT_RAW_REG (0x0020)

Continued from the previous page...

LP_I2C_TRANS_START_INT_RAW The raw interrupt status of the LP_I2C_TRANS_START_INT interrupt. (R/SS/WTC)

LP_I2C_NACK_INT_RAW The raw interrupt status of LP_I2C_SLAVE_STRETCH_INT interrupt. (R/SS/WTC)

LP_I2C_TXFIFO_OVF_INT_RAW The raw interrupt status of LP_I2C_TXFIFO_OVF_INT interrupt. (R/SS/WTC)

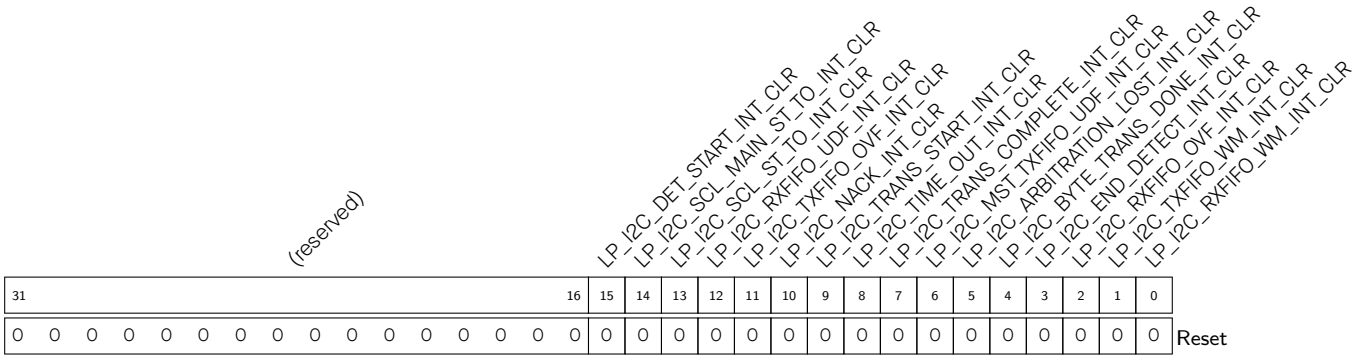
LP_I2C_RXFIFO_UDF_INT_RAW The raw interrupt status of LP_I2C_RXFIFO_UDF_INT interrupt. (R/SS/WTC)

LP_I2C_SCL_ST_TO_INT_RAW The raw interrupt status of LP_I2C_SCL_ST_TO_INT interrupt. (R/SS/WTC)

LP_I2C_SCL_MAIN_ST_TO_INT_RAW The raw interrupt status of LP_I2C_SCL_MAIN_ST_TO_INT interrupt. (R/SS/WTC)

LP_I2C_DET_START_INT_RAW The raw interrupt status of LP_I2C_DET_START_INT interrupt. (R/SS/WTC)

Register 47.55. LP_I2C_INT_CLR_REG (0x0024)



LP_I2C_RXFIFO_WM_INT_CLR Write 1 to clear LP_I2C_RXFIFO_WM_INT interrupt. (WT)

LP_I2C_TXFIFO_WM_INT_CLR Write 1 to clear LP_I2C_TXFIFO_WM_INT interrupt. (WT)

LP_I2C_RXFIFO_OVF_INT_CLR Write 1 to clear LP_I2C_RXFIFO_OVF_INT interrupt. (WT)

LP_I2C_END_DETECT_INT_CLR Write 1 to clear the LP_I2C_END_DETECT_INT interrupt. (WT)

LP_I2C_BYTE_TRANS_DONE_INT_CLR Write 1 to clear the LP_I2C_END_DETECT_INT interrupt. (WT)

LP_I2C_ARBITRATION_LOST_INT_CLR Write 1 to clear the LP_I2C_ARBITRATION_LOST_INT interrupt. (WT)

LP_I2C_MST_TXFIFO_UDF_INT_CLR Write 1 to clear LP_I2C_TRANS_COMPLETE_INT interrupt. (WT)

LP_I2C_TRANS_COMPLETE_INT_CLR Write 1 to clear the LP_I2C_TRANS_COMPLETE_INT interrupt. (WT)

LP_I2C_TIME_OUT_INT_CLR Write 1 to clear the LP_I2C_TIME_OUT_INT interrupt. (WT)

LP_I2C_TRANS_START_INT_CLR Write 1 to clear the LP_I2C_TRANS_START_INT interrupt. (WT)

LP_I2C_NACK_INT_CLR Write 1 to clear LP_I2C_SLAVE_STRETCH_INT interrupt. (WT)

LP_I2C_TXFIFO_OVF_INT_CLR Write 1 to clear LP_I2C_TXFIFO_OVF_INT interrupt. (WT)

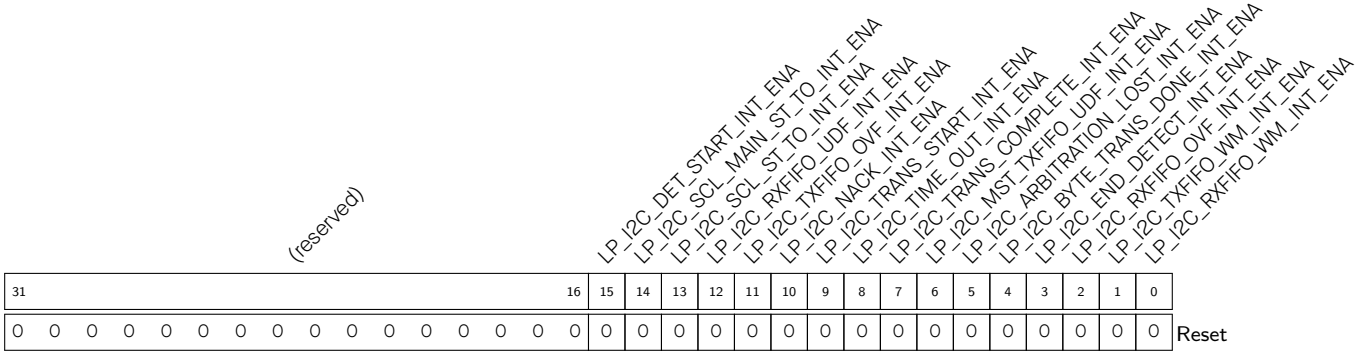
LP_I2C_RXFIFO_UDF_INT_CLR Write 1 to clear LP_I2C_RXFIFO_UDF_INT interrupt. (WT)

LP_I2C_SCL_ST_TO_INT_CLR Write 1 to clear LP_I2C_SCL_ST_TO_INT interrupt. (WT)

LP_I2C_SCL_MAIN_ST_TO_INT_CLR Write 1 to clear LP_I2C_SCL_MAIN_ST_TO_INT interrupt. (WT)

LP_I2C_DET_START_INT_CLR Write 1 to clear LP_I2C_DET_START_INT interrupt. (WT)

Register 47.56. LP_I2C_INT_ENA_REG (0x0028)



LP_I2C_RXFIFO_WM_INT_ENA Write 1 to enable LP_I2C_RXFIFO_WM_INT interrupt. (R/W)

LP_I2C_TXFIFO_WM_INT_ENA Write 1 to enable LP_I2C_TXFIFO_WM_INT interrupt. (R/W)

LP_I2C_RXFIFO_OVF_INT_ENA Write 1 to enable LP_I2C_RXFIFO_OVF_INT interrupt. (R/W)

LP_I2C_END_DETECT_INT_ENA Write 1 to enable the LP_I2C_END_DETECT_INT interrupt. (R/W)

LP_I2C_BYTE_TRANS_DONE_INT_ENA Write 1 to enable the LP_I2C_END_DETECT_INT interrupt. (R/W)

LP_I2C_ARBITRATION_LOST_INT_ENA Write 1 to enable the LP_I2C_ARBITRATION_LOST_INT interrupt. (R/W)

LP_I2C_MST_TXFIFO_UDF_INT_ENA Write 1 to enable LP_I2C_TRANS_COMPLETE_INT interrupt. (R/W)

LP_I2C_TRANS_COMPLETE_INT_ENA Write 1 to enable the LP_I2C_TRANS_COMPLETE_INT interrupt. (R/W)

LP_I2C_TIME_OUT_INT_ENA Write 1 to enable the LP_I2C_TIME_OUT_INT interrupt. (R/W)

LP_I2C_TRANS_START_INT_ENA Write 1 to enable the LP_I2C_TRANS_START_INT interrupt. (R/W)

LP_I2C_NACK_INT_ENA Write 1 to enable LP_I2C_SLAVE_STRETCH_INT interrupt. (R/W)

LP_I2C_TXFIFO_OVF_INT_ENA Write 1 to enable LP_I2C_TXFIFO_OVF_INT interrupt. (R/W)

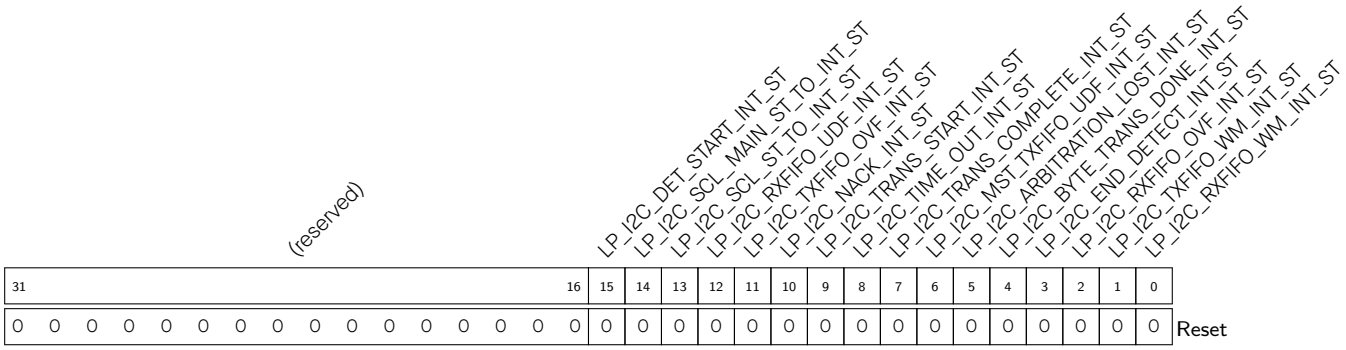
LP_I2C_RXFIFO_UDF_INT_ENA Write 1 to enable LP_I2C_RXFIFO_UDF_INT interrupt. (R/W)

LP_I2C_SCL_ST_TO_INT_ENA Write 1 to enable LP_I2C_SCL_ST_TO_INT interrupt. (R/W)

LP_I2C_SCL_MAIN_ST_TO_INT_ENA Write 1 to enable LP_I2C_SCL_MAIN_ST_TO_INT interrupt. (R/W)

LP_I2C_DET_START_INT_ENA Write 1 to enable LP_I2C_DET_START_INT interrupt. (R/W)

Register 47.57. LP_I2C_INT_STATUS_REG (0x002C)



LP_I2C_RXFIFO_WM_INT_ST The masked interrupt status of LP_I2C_RXFIFO_WM_INT interrupt. (RO)

LP_I2C_TXFIFO_WM_INT_ST The masked interrupt status of LP_I2C_TXFIFO_WM_INT interrupt. (RO)

LP_I2C_RXFIFO_OVF_INT_ST The masked interrupt status of LP_I2C_RXFIFO_OVF_INT interrupt. (RO)

LP_I2C_END_DETECT_INT_ST The masked interrupt status of the LP_I2C_END_DETECT_INT interrupt. (RO)

LP_I2C_BYTE_TRANS_DONE_INT_ST The masked interrupt status of the LP_I2C_END_DETECT_INT interrupt. (RO)

LP_I2C_ARBITRATION_LOST_INT_ST The masked interrupt status of the LP_I2C_ARBITRATION_LOST_INT interrupt. (RO)

LP_I2C_MST_TXFIFO_UDF_INT_ST The masked interrupt status of LP_I2C_TRANS_COMPLETE_INT interrupt. (RO)

LP_I2C_TRANS_COMPLETE_INT_ST The masked interrupt status of the LP_I2C_TRANS_COMPLETE_INT interrupt. (RO)

LP_I2C_TIME_OUT_INT_ST The masked interrupt status of the LP_I2C_TIME_OUT_INT interrupt. (RO)

LP_I2C_TRANS_START_INT_ST The masked interrupt status of the LP_I2C_TRANS_START_INT interrupt. (RO)

Continued on the next page...

Register 47.57. LP_I2C_INT_STATUS_REG (0x002C)

Continued from the previous page...

LP_I2C_NACK_INT_ST The masked interrupt status of LP_I2C_SLAVE_STRETCH_INT interrupt. (RO)

LP_I2C_TXFIFO_OVF_INT_ST The masked interrupt status of LP_I2C_TXFIFO_OVF_INT interrupt. (RO)

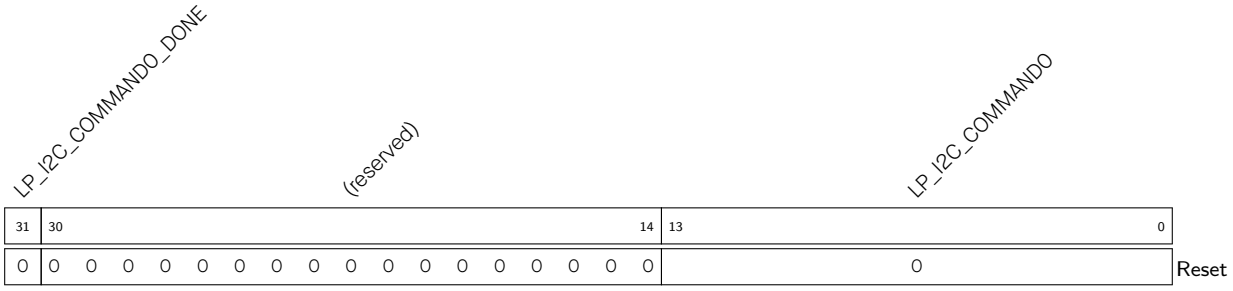
LP_I2C_RXFIFO_UDF_INT_ST The masked interrupt status of LP_I2C_RXFIFO_UDF_INT interrupt. (RO)

LP_I2C_SCL_ST_TO_INT_ST The masked interrupt status of LP_I2C_SCL_ST_TO_INT interrupt. (RO)

LP_I2C_SCL_MAIN_ST_TO_INT_ST The masked interrupt status of LP_I2C_SCL_MAIN_ST_TO_INT interrupt. (RO)

LP_I2C_DET_START_INT_ST The masked interrupt status of LP_I2C_DET_START_INT interrupt. (RO)

Register 47.58. LP_I2C_COMD0_REG (0x0058)



LP_I2C_COMMAND0 Configures command 0.

It consists of three parts:

op_code is the command

1: WRITE

2: STOP

3: READ

4: END

6: RSTART

Byte_num represents the number of bytes that need to be sent or received.

ack_check_en, ack_exp and ack are used to control the ACK bit. See I2C cmd structure 47.4-2 for more information. (R/W)

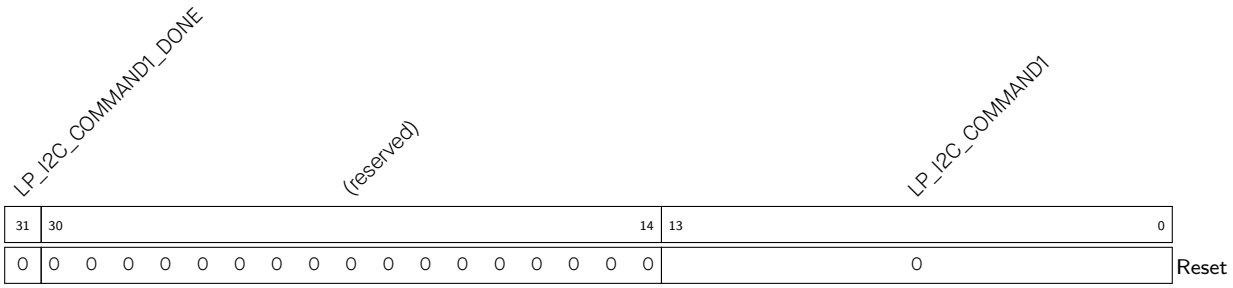
LP_I2C_COMMAND0_DONE Represents whether command 0 is done in I2C Master mode.

0: Not done

1: Done

(R/W/SS)

Register 47.59. LP_I2C_COMD1_REG (0x005C)



LP_I2C_COMMAND1 Configures command 1.

See details in I2C_CMD0_REG[13:0]. (R/W)

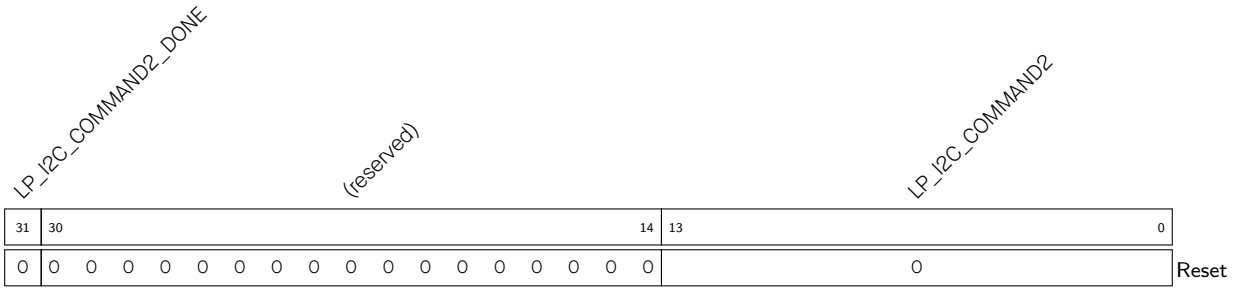
LP_I2C_COMMAND1_DONE Represents whether command 1 is done in I2C Master mode.

0: Not done

1: Done

(R/W/SS)

Register 47.60. LP_I2C_COMD2_REG (0x0060)



LP_I2C_COMMAND2 Configures command 2. See details in I2C_CMD0_REG[13:0]. (R/W)

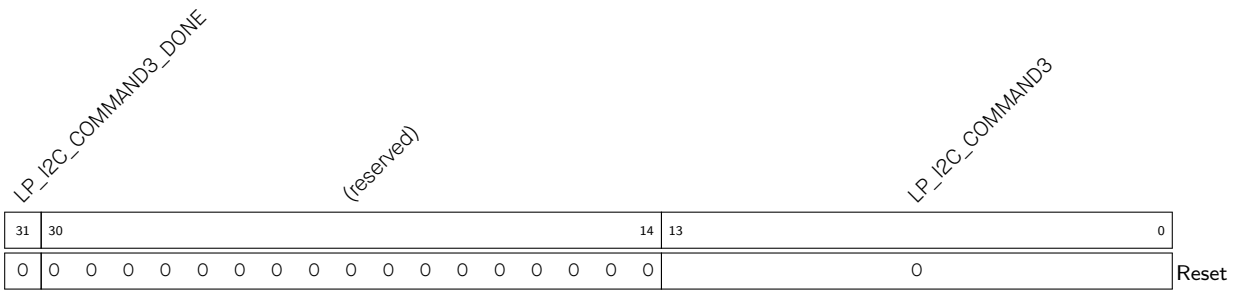
LP_I2C_COMMAND2_DONE Represents whether command 2 is done in I2C Master mode.

0: Not done

1: Done

(R/W/SS)

Register 47.61. LP_I2C_COMD3_REG (0x0064)



LP_I2C_COMMAND3 Configures command 3. See details in I2C_CMD0_REG[13:0]. (R/W)

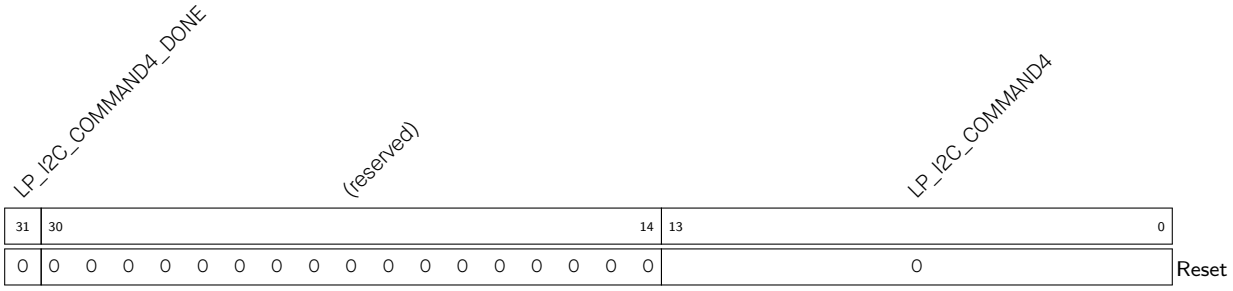
LP_I2C_COMMAND3_DONE Represents whether command 3 is done in I2C Master mode.

0: Not done

1: Done

(R/W/SS)

Register 47.62. LP_I2C_COMD4_REG (0x0068)



LP_I2C_COMMAND4 Configures command 4. See details in I2C_CMD0_REG[13:0]. (R/W)

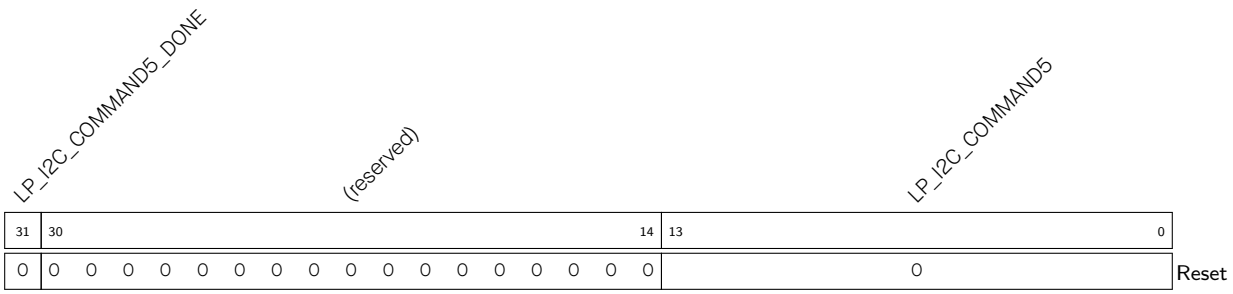
LP_I2C_COMMAND4_DONE Represents whether command 4 is done in I2C Master mode.

0: Not done

1: Done

(R/W/SS)

Register 47.63. LP_I2C_COMD5_REG (0x006C)



LP_I2C_COMMAND5 Configures command 5. See details in I2C_CMD0_REG[13:0]. (R/W)

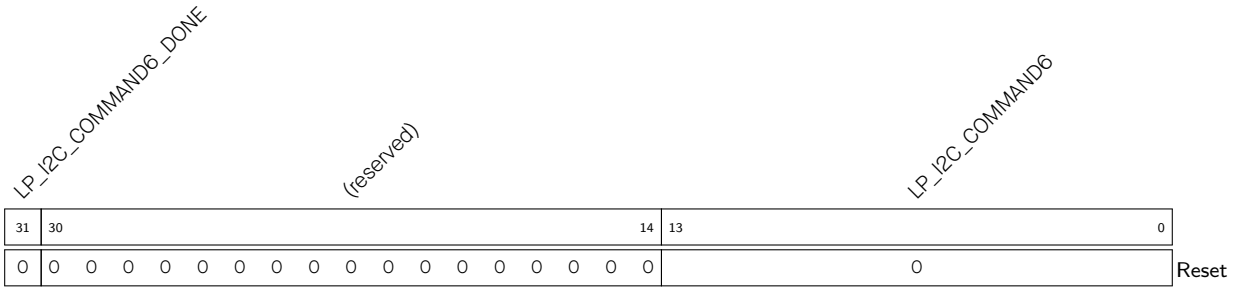
LP_I2C_COMMAND5_DONE Represents whether command 5 is done in I2C Master mode.

0: Not done

1: Done

(R/W/SS)

Register 47.64. LP_I2C_COMD6_REG (0x0070)



LP_I2C_COMMAND6 Configures command 6. See details in I2C_CMD0_REG[13:0]. (R/W)

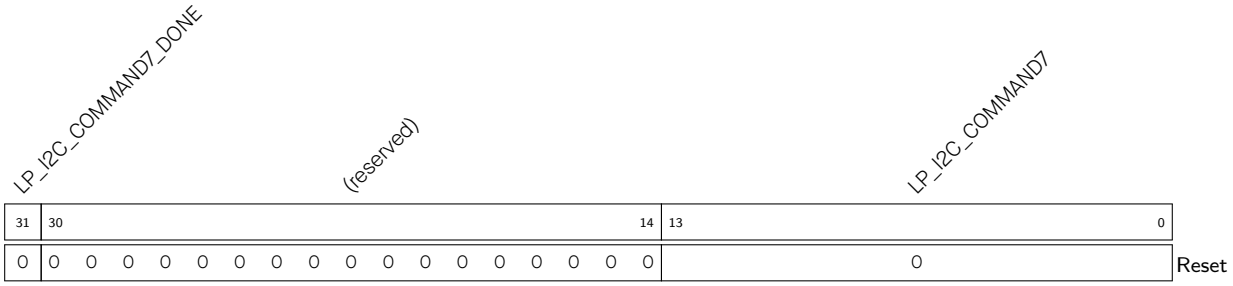
LP_I2C_COMMAND6_DONE Represents whether command 6 is done in I2C Master mode.

0: Not done

1: Done

(R/W/SS)

Register 47.65. LP_I2C_COMD7_REG (0x0074)



LP_I2C_COMMAND7 Configures command 7. See details in I2C_CMD0_REG[13:0]. (R/W)

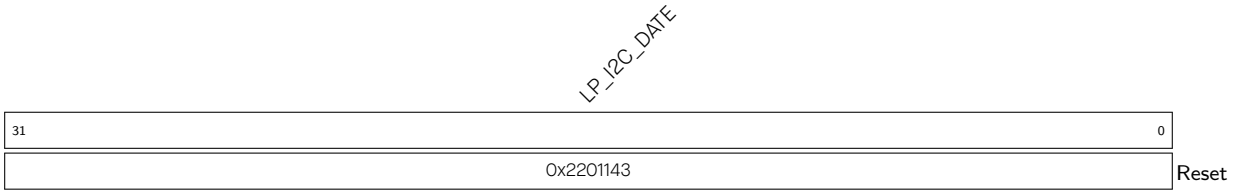
LP_I2C_COMMAND7_DONE Represents whether command 7 is done in I2C Master mode.

0: Not done

1: Done

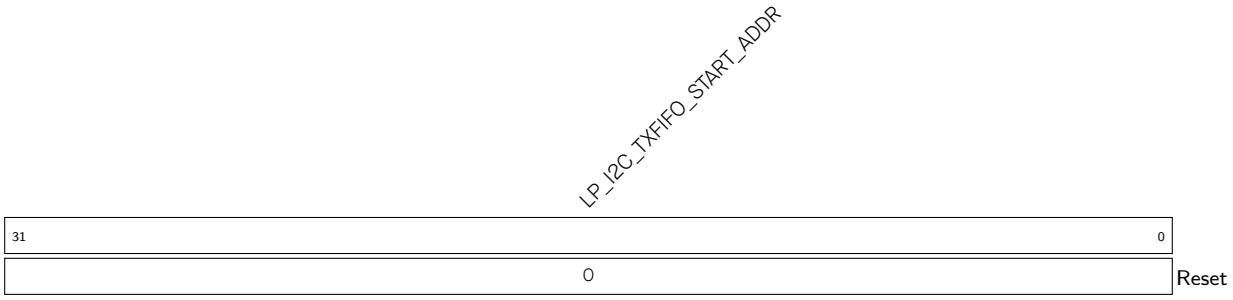
(R/W/SS)

Register 47.66. LP_I2C_DATE_REG (0x00F8)



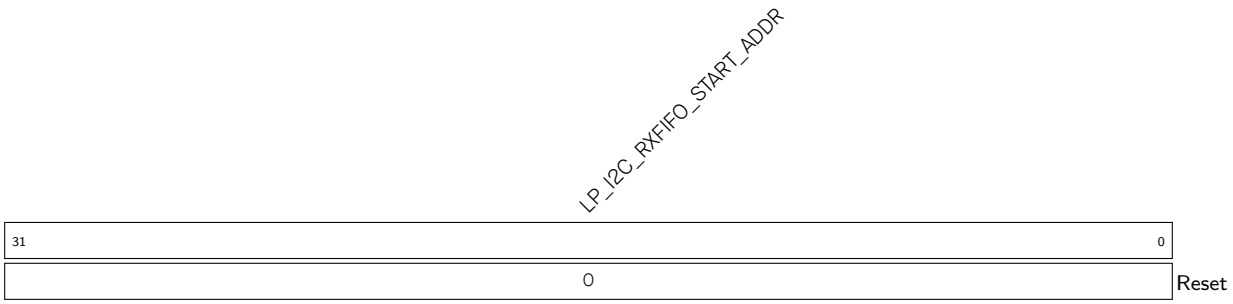
LP_I2C_DATE Version control register. (R/W)

Register 47.67. LP_I2C_TXFIFO_START_ADDR_REG (0x0100)



LP_I2C_TXFIFO_START_ADDR Represents the I2C txfifo first address. (HRO)

Register 47.68. LP_I2C_RXFIFO_START_ADDR_REG (0x0180)



LP_I2C_RXFIFO_START_ADDR Represents the I2C rxfifo first address. (HRO)

Chapter 48

Analog I2C Controller

48.1 Introduction

The Analog I2C Controller contains two I2C masters dedicated to configuring and communicating with certain analog circuit modules. Those modules can adjust their operating states by configuring their internal registers to better suit the current application scenario of the chip. For example, the Analog I2C Controller can configure an analog sensor to add a uniform offset to the ADC sampling values, thereby extending the sampling voltage range. Each configurable analog module has an I2C slave internally, which is assigned with an independent address. The Analog I2C Controller supports the configuration of the following analog modules:

- CPU_PLL: This mainly includes the configuration register for CPLL_CLK, with a slave address of 0x67. For more information on CPLL_CLK, refer to Chapter 11 *Reset and Clock*.
- SYS_PLL: This mainly includes the configuration register for SPLL_CLK, with a slave address of 0x66. For more information on SPLL_CLK, refer to Chapter 11 *Reset and Clock*.
- MSPI: This mainly includes the configuration register for MPLL_CLK , with a slave address of 0x63. For more information on MPLL_CLK, refer to Chapter 11 *Reset and Clock*.
- PLLA: This mainly includes the configuration register for APLL_CLK , with a slave address of 0x6f. For more information on APLL_CLK, refer to Chapter 11 *Reset and Clock*.
- ANA_SENSOR: This mainly includes the configuration register for analog sensors (such as *Temperature Sensor (TSENS)* and *ADC Controller (ADC)*), with a slave address of 0x69.
- BIAS: This mainly includes the voltage detection configuration register, with a slave address of 0x6a. For more information, refer to Chapter 25 *Brown-out Detector*.

48.2 Feature List

The Analog I2C Controller has the following features:

- Master mode only
- 7-bit addressing
- Adjustable transmission rate
- Communication in the sleep modes supported by the Low-Power CPU
- Dual master operation mode

48.3 Architectural Overview

The figure below shows the architecture of the Analog I2C Controller.

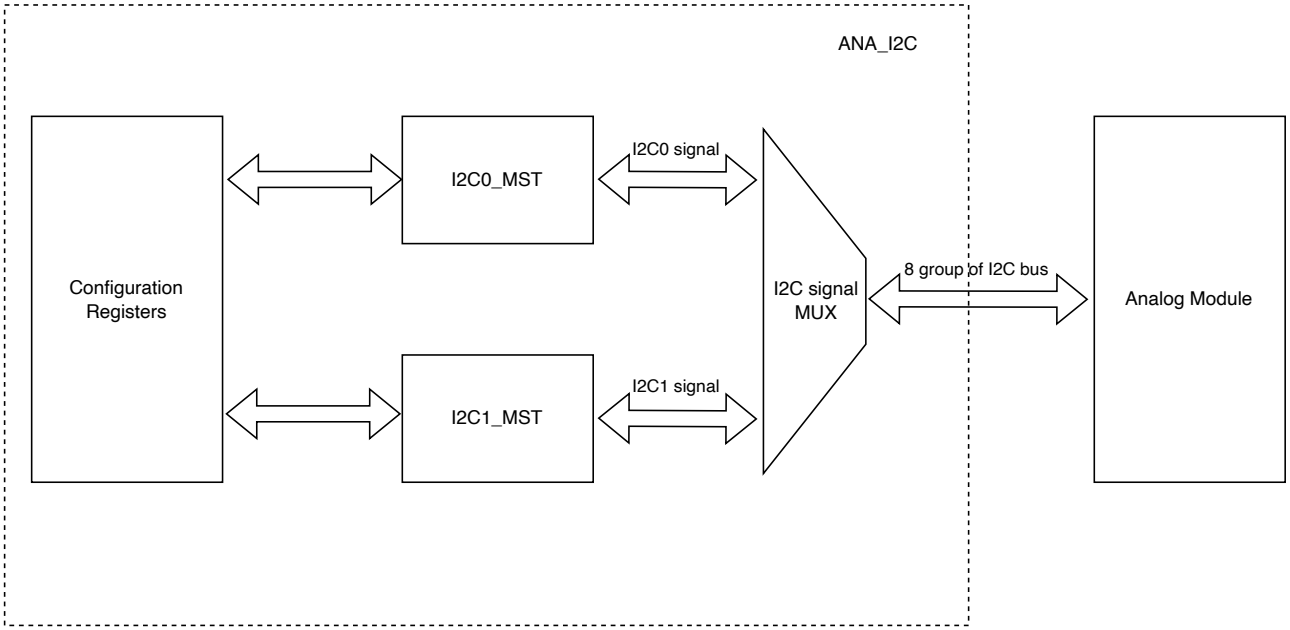


Figure 48.3-1. Analog I2C Controller Architecture

- Configuration Registers: A register module interfacing with the software, providing relevant configurations for the operation of the I2C master. This module controls the transmission address, data, etc., and monitors this transmission process by reading the status registers of I2C0/1_MST.
- I2C0/1_MST: Two I2C masters that can independently configure analog modules in parallel. The I2C signals they generate will be selected by a subsequent MUX and output to the analog modules.
- I2C signal MUX: A module that determines which analog master, I2C0_MST or I2C1_MST, configures a particular analog module.

48.4 Functional Description

Transmission Rate Adjustment

The transmission rate and signal phase of the I2C signal can be configured by ANA_I2C_MST_I2C*x*_SDA_SIDE_GUARD and ANA_I2C_MST_I2C*x*_SCL_PULSE_DUR. The relationship between the register values and the waveform is shown in the following figure.

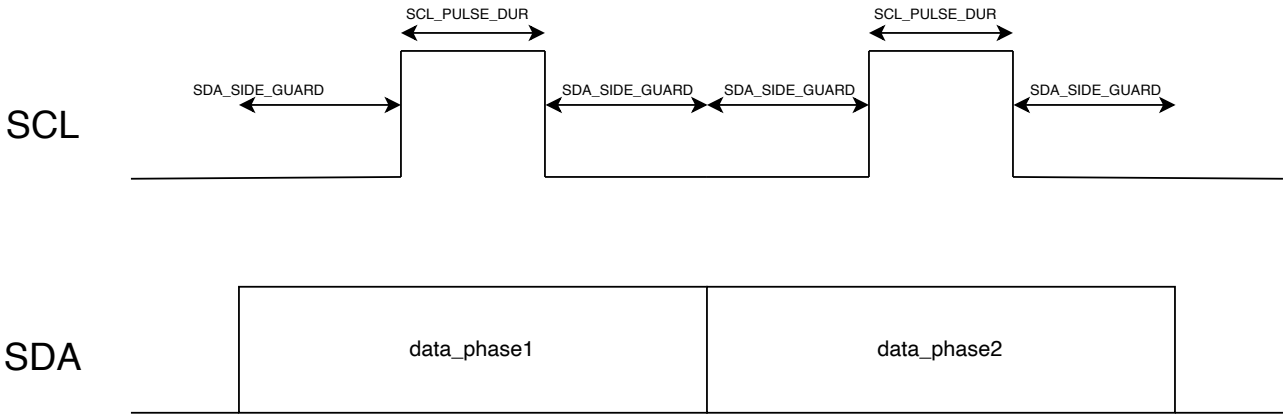


Figure 48.4-1. Signal Phase

Operation in Sleep Modes

In chip sleep modes, the operation of the Analog I2C Controller can be driven by the Low-Power CPU (see Chapter 5 *Low-Power CPU*).

Dual Master Operation Mode

The module integrates two independent master modules, each with its own set of configurations. The two masters can operate independently, configuring different analog modules, thereby improving operational efficiency.

48.5 Programming Procedures

The following is the process for configuring analog modules using the Analog I2C Controller.

- Write 1 to LPPERI_CK_EN_LP_I2CMST to enable the clock of the Controller.
- Configure ANA_I2C_MST_ANA_CONF2 to select the module for communication.
- Configure ANA_I2C_MST_I2C*x*_SDA_SIDE_GUARD and ANA_I2C_MST_I2C*x*_SCL_PULSE_DUR to determine the transmission rate and signal phase of the I2C signal.
- Configure ANA_I2C_MST_I2C*x*_CTRL to select relevant information for the current communication.
- Read ANA_I2C_MST_I2C*x*_BUSY. A low state indicates the end of the current communication for I2C*x*.

48.6 Register Summary

The addresses in this section are relative to Analog I2C Controller base address provided in Table 9.3-2 in Chapter 9 *System and Memory*.

The abbreviations given in Column **Access** are explained in Section *Access Types for Registers*.

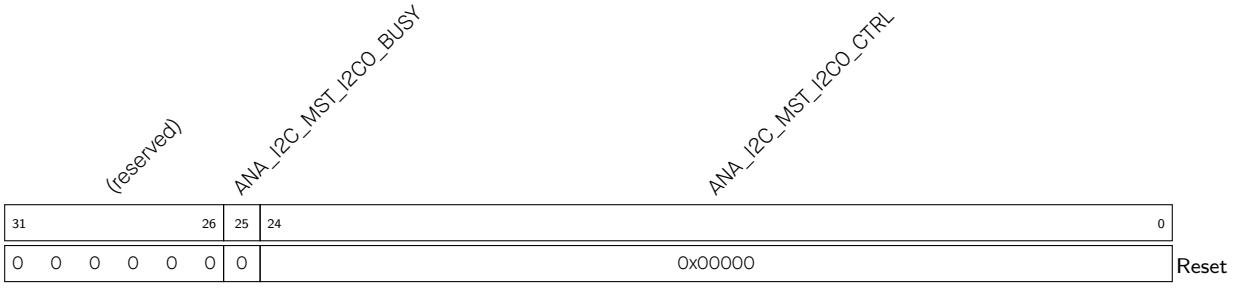
Name	Description	Address	Access
Configuration Registers			
ANA_I2C_MST_I2C0_CTRL_REG	I2C0 transmission configuration register	0x0000	varies
ANA_I2C_MST_I2C1_CTRL_REG	I2C1 transmission configuration register	0x0004	varies
ANA_I2C_MST_ANA_CONF2_REG	I2C master selection register	0x0020	varies
ANA_I2C_MST_I2C0_CTRL1_REG	I2C0 transmission rate and signal phase configuration register	0x0024	R/W
ANA_I2C_MST_I2C1_CTRL1_REG	I2C1 transmission rate and signal phase configuration register	0x0028	R/W
ANA_I2C_MST_DATE_REG	Version control register	0x0038	R/W

48.7 Registers

The addresses in this section are relative to Analog I2C Controller base address provided in Table 9.3-2 in Chapter 9 *System and Memory*.

For how to program registers that contain reserved fields, please refer to Section Programming Reserved Register Field.

Register 48.1. ANA_I2C_MST_I2C0_CTRL_REG (0x0000)



ANA_I2C_MST_I2C0_CTRL Configures the transmission information for I2C0.

Bit[0:7]: Configures the slave address

Bit[8:15]: Configures the slave register address

Bit[16:23]: Configures the transmitted data

Bit[24]: Configures the read or write operation

0: Write

1: Read

Once this register is configured, the I2C0 master will generate I2C read or write signals. (R/W)

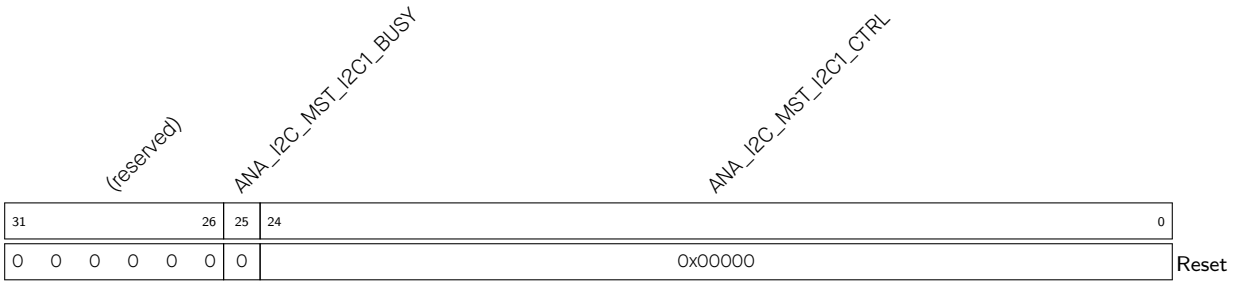
ANA_I2C_MST_I2C0_BUSY Represents whether I2C0 is currently transferring data.

0: I2C0 is not transferring data

1: I2C0 is transferring data

(RO)

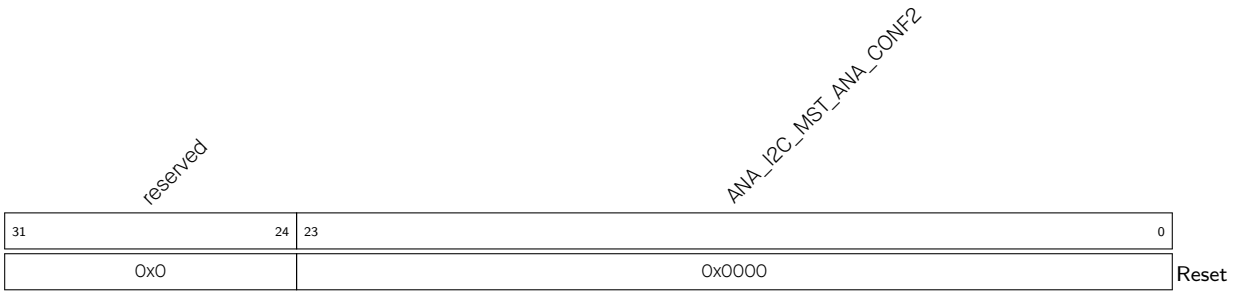
Register 48.2. ANA_I2C_MST_I2C1_CTRL_REG (0x0004)



ANA_I2C_MST_I2C1_CTRL Configures the transmission information for I2C1. Its subfields have the same meaning with ANA_I2C_MST_I2C0_CTRL. (R/W)

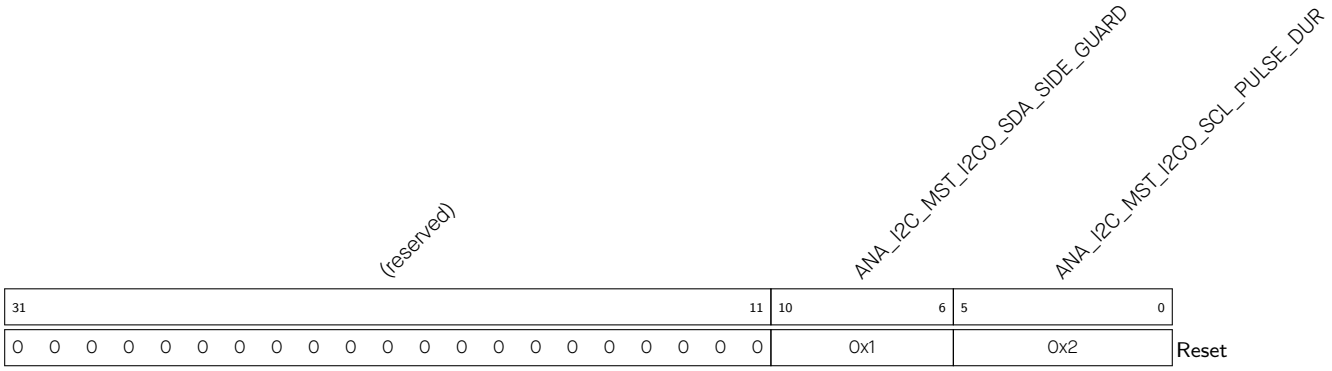
ANA_I2C_MST_I2C1_BUSY Represents whether I2C1 is currently transferring data
0: I2C1 is not transferring data
1: I2C1 is transferring data
(RO)

Register 48.3. ANA_I2C_MST_ANA_CONF2_REG (0x0020)



ANA_I2C_MST_ANA_CONF2 Configures which I2C master the following analog modules uses. If the corresponding bit is set to 1, I2C0 master is used for communication; if set to 0, I2C1 master is used. Since the slave address has already been defined, erroneous communication will not be caused.
Bit5: SYS_PLL
Bit6: SDIO_PLL
Bit7: ANA_SENSOR
Bit8: PLLA
Bit9: MSPI
Bit10: DIG_REG
Bit11: CPU_PLL
Bit12: BIAS
(R/W)

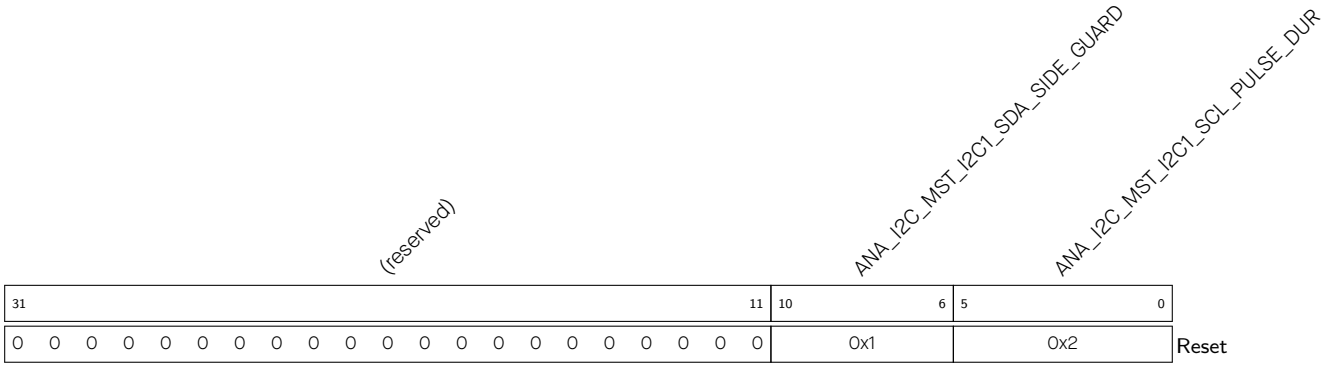
Register 48.4. ANA_I2C_MST_I2C0_CTRL1_REG (0x0024)



ANA_I2C_MST_I2C0_SCL_PULSE_DUR Configures the duration of the high-level period of the SCL driven by I2C0. The duration is measured by the counter operating on the LP_FAST_CLK clock. (R/W)

ANA_I2C_MST_I2C0_SDA_SIDE_GUARD Configures the duration of the low-level period of the SCL driven by I2C0. The duration is measured by the counter operating on the LP_FAST_CLK clock. (R/W)

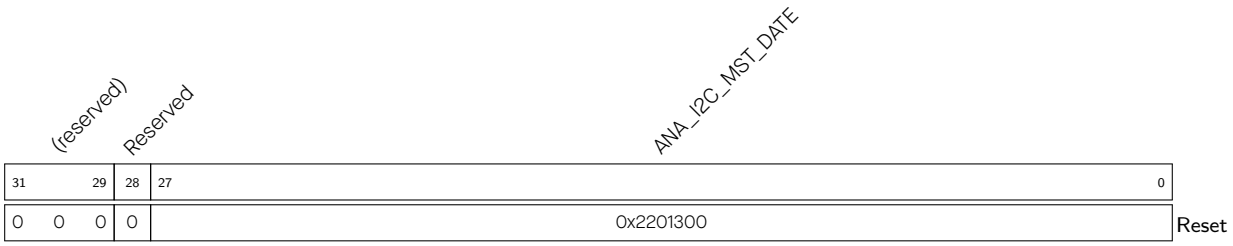
Register 48.5. ANA_I2C_MST_I2C1_CTRL1_REG (0x0028)



ANA_I2C_MST_I2C1_SCL_PULSE_DUR Configures the duration of the high-level period of the SCL driven by I2C1. The duration is measured by the counter operating on the LP_FAST_CLK clock. (R/W)

ANA_I2C_MST_I2C1_SDA_SIDE_GUARD Configures the duration of the low-level period of the SCL driven by I2C1. The duration is measured by the counter operating on the LP_FAST_CLK clock. (R/W)

Register 48.6. ANA_I2C_MST_DATE_REG (0x0038)



ANA_I2C_MST_DATE Version control register. (R/W)

Chapter 49

I2S Controller (I2S)

49.1 Overview

ESP32-P4 has three built-in I2S interfaces (i.e., I2S*0*, I2S*1*, and I2S*2*), which provide flexible communication interfaces for streaming digital data in multimedia applications, especially digital audio applications.

The I2S standard bus defines three signals, namely, a bit clock signal (BCK), a channel/word select signal (WS), and a serial data signal (SD). A basic I2S data bus has one master and one slave. The roles remain unchanged throughout the communication. The I2S module on ESP32-P4 provides separate transmit (TX) and receive (RX) units for high performance.

Note:
The information provided in this chapter applies to I2S*0*, I2S*1*, and I2S*2*. Unless otherwise indicated, I2S or I2S*n* in this chapter refer to I2S*0*, I2S*1*, and I2S*2*.

49.2 Terminology

To better illustrate the functionality of I2S, the following terms are used in this chapter.

Master mode	As a master, I2S*n* drives BCK/WS signals and transmits data to or receives data from a slave.
Slave mode	As a slave, I2S*n* is driven by BCK/WS signals and receives data from or transmits data to a master.
Full-duplex	There are two separate data lines. The transmitted and received data is carried simultaneously.
Half-duplex	Only one side, the master or the slave, transmits data first, and the other side receives data. Data transmission and reception cannot occur simultaneously.
A-law and μ-law	A-law and μ-law are compression/decompression algorithms in digital pulse code modulated (PCM) non-uniform quantization, which can effectively improve the signal-to-quantization noise ratio.
TDM RX mode	In this mode, pulse code modulated (PCM) data is received utilizing time division multiplexing (TDM). The signal lines include BCK, WS, and SD. Data from 16 channels at most can be received. TDM Philips standard, TDM MSB alignment standard, and TDM PCM standard are supported in this mode, depending on user configuration.

PDM RX mode In this mode, pulse density modulation (PDM) data is received. Used signals include WS and DATA. PDM standard is supported in this mode by user configuration.

TDM TX mode In this mode, pulse code modulated (PCM) data is sent in a way of time division multiplexing (TDM). The signal lines include BCK, WS, and DATA. Data up to 16 channels can be sent. TDM Philips standard, TDM MSB alignment standard, and TDM PCM standard are supported in this mode, depending on user configuration.

PDM TX mode In this mode, pulse density modulation (PDM) data is sent. The signal lines include WS and DATA. PDM standard is supported in this mode by user configuration.

PCM-to-PDM Data Format Converter (for I2S*0* only) A data format converter that converts PCM data into PDM data (hereinafter referred to as PCM-to-PDM converter), which can be enabled in PDM TX master mode. When the converter is enabled, I2S fetches pulse code modulated (PCM) data from storage via GDMA, converts it to pulse density modulation (PDM) data, and then transmits it.

PDM-to-PCM Data Format Converter (for I2S*0* only) A data format converter that converts PDM data into PCM data (hereinafter referred to as PDM-to-PCM converter), which can be enabled in PDM RX master or slave mode. When the converter is enabled, I2S receives pulse density modulation (PDM) data, converts it into pulse code modulated (PCM) data, and stores it into memory via GDMA.

49.3 Features

The I2S module has the following features:

- Master mode and slave mode
- Full-duplex and half-duplex communications
- Separate TX and RX units that can work independently or simultaneously
- A variety of audio standards supported:
  - TDM Philips standard
  - TDM MSB alignment standard
  - TDM PCM standard
  - PDM standard
- Various TX/RX modes supported:
  - TDM TX mode, up to 16 channels supported
  - TDM RX mode, up to 16 channels supported

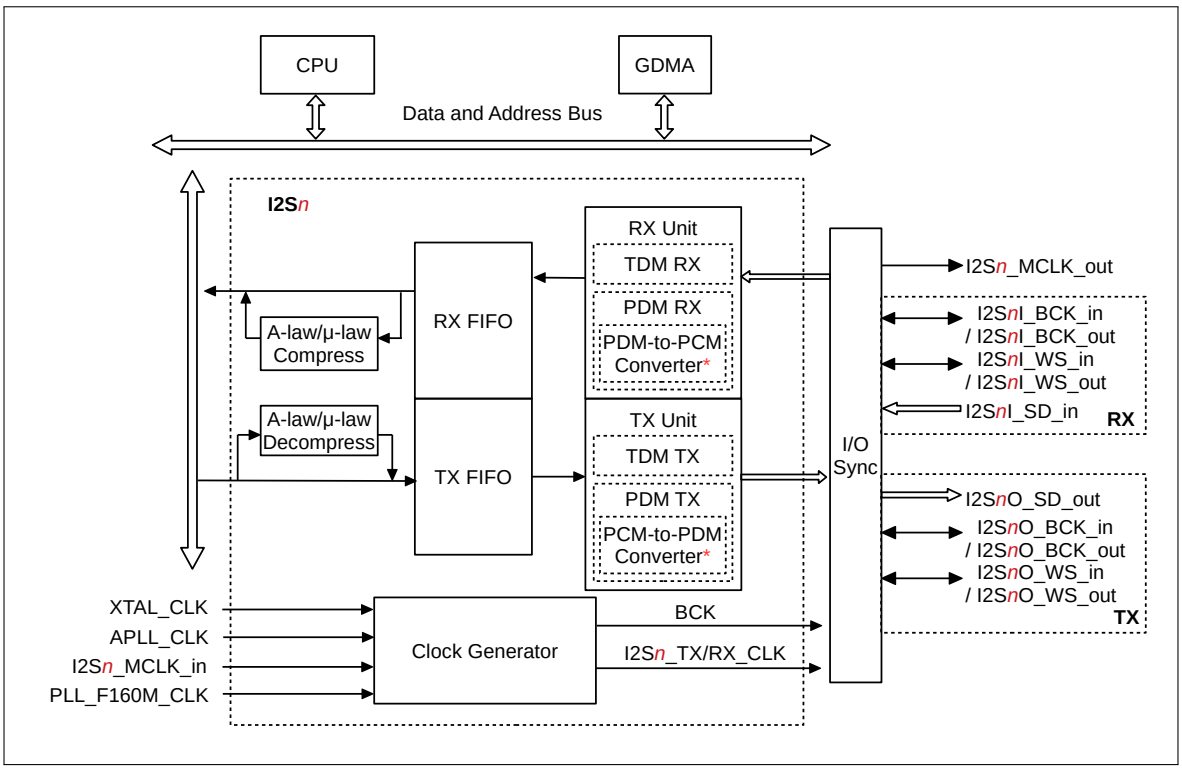
  - PDM TX mode
    * Raw PDM data transmission
    * PCM-to-PDM data format conversion (for I2S*0* only), up to 2 channels supported
  - PDM RX mode
    * Raw PDM data reception
    * PDM-to-PCM data format conversion (for I2S*0* only), up to 8 channels supported
- Configurable APLL clock with frequencies up to 125 MHz
- Configurable high-precision sample clock with a variety of sampling frequencies supported (refer to the note below for more details)
- 8/16/24/32-bit data width
- Synchronous counter in TX mode
- ETM feature
- Direct Memory Access (GDMA-AHB only)
- Standard I2S interface interrupts

Note:
In slave mode, to ensure the sampling accuracy, the module clock frequency must be greater than or equal to 8 times the BCK clock frequency. Therefore, the maximum sampling frequency of I2S*n* is limited by the data bit width and the number of channels. For example, since the clock source frequency is up to 125 MHz, the module clock can be configured up to 62.5 MHz, and the BCK clock can be configured up to 7.8125 MHz. Therefore, when transmitting dual-channel 32-bit width data, I2S*n* supports sample frequencies up to 122.070 kHz, e.g., 8 kHz, 16 kHz, 32 kHz, 44.1 kHz, 48 kHz, 88.2 kHz, and 96 kHz. Please refer to Section 49.6 *I2S TX/RX Clock* for detailed information.

49.4 System Architecture

Figure 49.4-1 shows the structure of the ESP32-P4 I2S*n* module, consisting of:

- Transmit control unit (TX Unit)
- Receive control unit (RX Unit)
- Input and output timing unit (I/O Sync)
- Clock divider (Clock Generator)
- 64 x 32-bit TX FIFO
- 64 x 32-bit RX FIFO
- Compress/Decompress units



Note: The PDM-to-PCM converter and PCM-to-PDM converter are only supported by I2S*O*.

Figure 49.4-1. ESP32-P4 I2S System Diagram

The I2S*n* module supports direct memory access (DMA) to internal memory. For more information, see Chapter 6 *GDMA Controller (GDMA-AHB, GDMA-AXI)*.

Both the TX unit and the RX unit have a three-line interface that uses a bit clock line (BCK), a word select line (WS), and a serial data line (SD). The SD line of the TX unit is used for data output and the SD line of the RX unit for data input. The BCK and WS signal lines of the TX and RX units are used for data output in master mode, and the BCK and WS signal lines of the TX and RX units are used for data input in slave mode.

The signal bus of the I2S*n* module is shown at the right part of Figure 49.4-1. The naming of these signals in RX and TX units follows the pattern of I2S*n*A_B_C such as I2S*n*I_BCK_in.

- "A" indicates that the signal belongs to the I2S*n* TX or RX unit, which includes:
  - "I": Signal input to/output from the RX unit
  - "O": Signal input to/output from the TX unit
- "B" represents the signal function, which includes:
  - BCK
  - WS
  - SD
- "C" represents the signal direction, which includes:
  - "in": Input signal into the I2S*n* module
  - "out": Output signal from the I2S*n* module

Table 49.4-1 provides a detailed description of I2S signals.

Table 49.4-1. I2S*n* Signal Description

Signal	Direction	Function
I2S*n*I_BCK_in	Input	In I2S*n* slave mode, inputs BCK signal for RX unit.
I2S*n*I_BCK_out	Output	In I2S*n* master mode, outputs BCK signal for RX unit.
I2S*n*I_WS_in	Input	In I2S*n* slave mode, inputs WS signal for RX unit.
I2S*n*I_WS_out	Output	In I2S*n* master mode, outputs WS signal for RX unit.
I2S*n*I_Data_in	Input	Works as the serial input data bus for I2S*n* RX unit.
I2S*n*O_Data_out	Output	Works as the serial output data bus for I2S*n* TX unit.
I2S*n*O_BCK_in	Input	In I2S*n* slave mode, inputs BCK signal for TX unit.
I2S*n*O_BCK_out	Output	In I2S*n* master mode, outputs BCK signal for TX unit.
I2S*n*O_WS_in	Input	In I2S*n* slave mode, inputs WS signal for TX unit.
I2S*n*O_WS_out	Output	In I2S*n* master mode, outputs WS signal for TX unit.
I2S*n*_MCLK_in	Input	In I2S*n* slave mode, works as a clock source from the external master.
I2S*n*_MCLK_out	Output	In I2S*n* master mode, works as a clock source for the external slave.
I2S*O*I_Data1_in	Input	When the PDM-to-PCM converter is enabled, works as the serial input data line for RX unit.
I2S*O*I_Data2_in	Input	When the PDM-to-PCM converter is enabled, works as the serial input data line for RX unit.
I2S*O*I_Data3_in	Input	When the PDM-to-PCM converter is enabled, works as the serial input data line for RX unit.
I2S*O*O_Data1_out	Output	When the PCM-to-PDM converter is enabled, works as the serial output data line for TX unit.

* Any required signals of I2S*n* must be mapped to the chip's pins via HP GPIO matrix, see Chapter 10 *GPIO Matrix and IO MUX*.

49.5 Supported Audio Standards

ESP32-P4 I2S*n* supports multiple audio standards, including TDM Philips standard, TDM MSB alignment standard, TDM PCM standard, and PDM standard.

Select the needed standard by configuring the following bits:

- I2S_TX/RX_TDM_EN
  - 0: Disable TDM mode
  - 1: Enable TDM mode
- I2S_TX/RX_PDM_EN
  - 0: Disable PDM mode
  - 1: Enable PDM mode
- I2S_TX/RX_MSB_SHIFT
  - 0: WS and SD signals change simultaneously, i.e., enable MSB alignment standard

– 1: WS signal changes one BCK clock cycle earlier than SD signal, i.e., enable Philips standard or select PCM standard

Note:
I2S_TX/RX_TDM_EN and I2S_TX/RX_PDM_EN must not be configured to 1 or 0 at the same time, otherwise ESP32-P4 I2S*n* will transmit data incorrectly in a mode that is neither TDM nor PDM.

49.5.1 TDM Philips Standard

Philips specifications require that WS signal changes one BCK clock cycle earlier than SD signal on BCK falling edge, which means that WS signal is valid from one clock cycle before transmitting the first bit of channel data and changes one clock before the end of channel data transfer. SD signal line transmits the most significant bit of audio data first.

Compared with the Philips standard, TDM Philips standard supports multiple channels. See Figure 49.5-1.

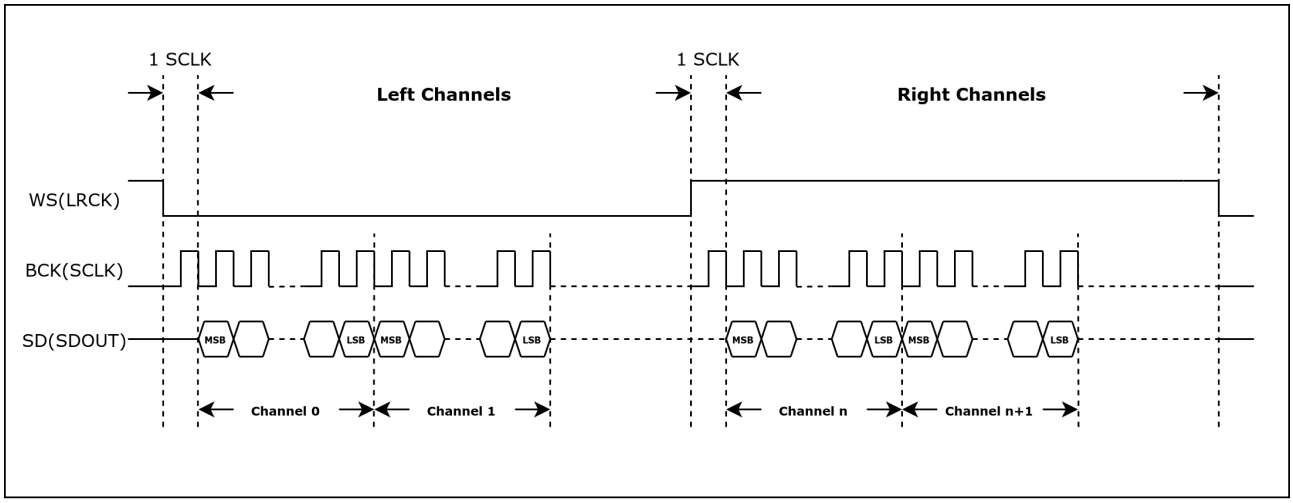


Figure 49.5-1. TDM Philips Standard Timing Diagram

49.5.2 TDM MSB Alignment Standard

MSB alignment specifications require that WS and SD signals change simultaneously on the falling edge of BCK. The WS signal is valid until the end of the channel data transfer. The SD signal line transmits the most significant bit of audio data first.

Compared with MSB alignment standard, TDM MSB alignment standard supports multiple channels. See Figure 49.5-2.

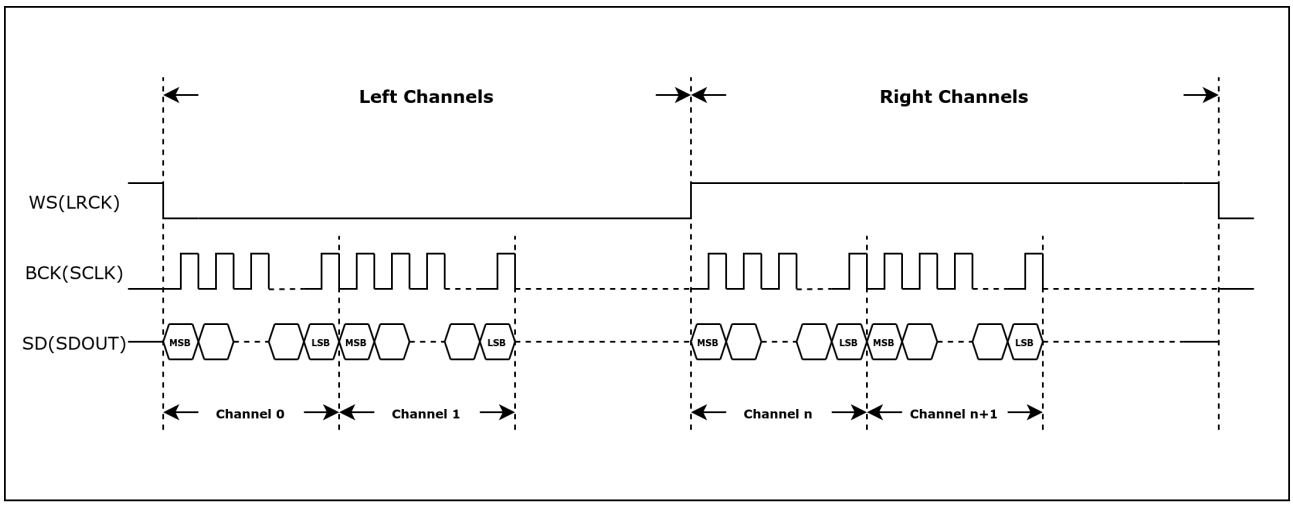


Figure 49.5-2. TDM MSB Alignment Standard Timing Diagram

49.5.3 TDM PCM Standard

Short frame synchronization under the PCM standard requires that the WS signal changes one BCK clock cycle earlier than the SD signal on the falling edge of BCK, which means that the WS signal becomes valid one clock cycle before transferring the first bit of channel data and remains unchanged in this BCK clock cycle. SD signal line first transmits the most significant bit of audio data.

Compared with PCM standard, TDM PCM standard supports multiple channels. See Figure 49.5-3.

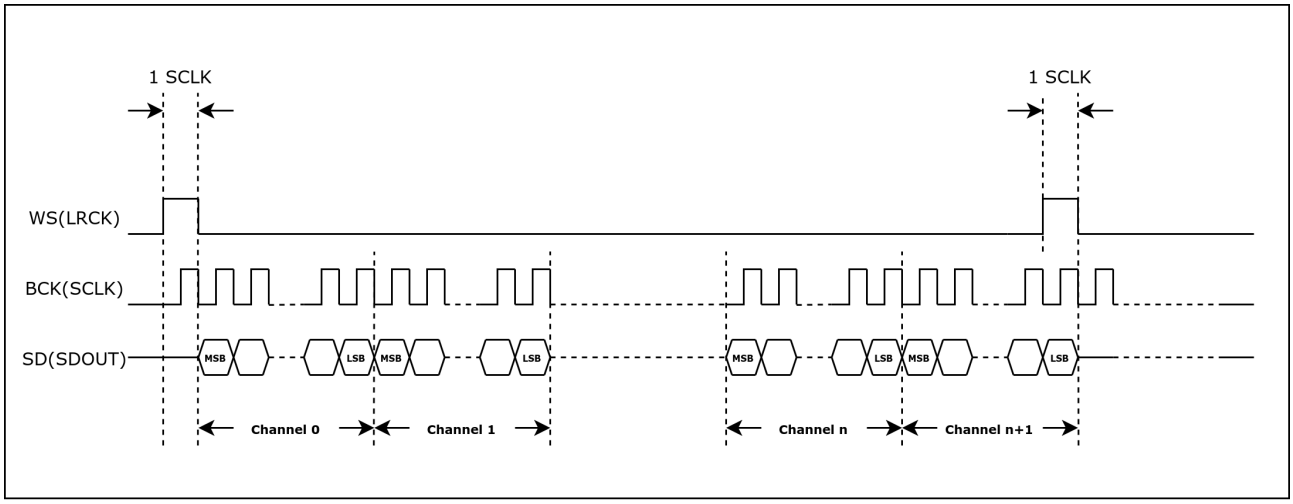


Figure 49.5-3. TDM PCM Standard Timing Diagram

49.5.4 PDM Standard

Under PDM standard, WS signal changes continuously during data transmission. The low-level and high-level of this signal indicates the left channel and right channel respectively. WS and SD signals change simultaneously. See Figure 49.5-4.

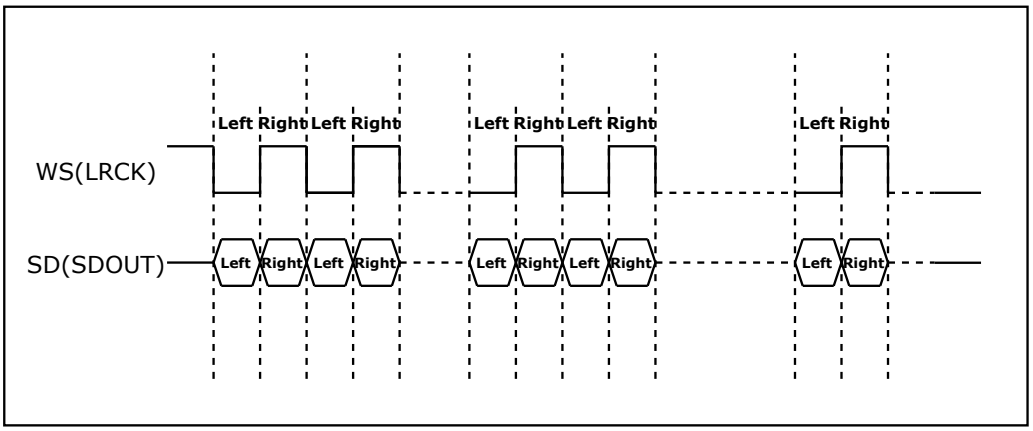


Figure 49.5-4. PDM Standard Timing Diagram

49.6 I2S TX/RX Clock

I2S*n*_TX/RX_CLK is the master clock of I2S*n* TX/RX unit, divided from:

- 40 MHz XTAL_CLK
- 160 MHz PLL_F160M_CLK
- APLL_CLK with adjustable frequency
- I2S*n*_MCLK_in (external input clock)

The serial clock (BCK) of the I2S*n* TX/RX unit is divided from I2S*n*_TX/RX_CLK, as shown in Figure 49.6-1. HP_SYS_CLKRST_I2S*n*_TX/RX_CLK_SRC_SEL is used to select clock source for TX/RX unit, and HP_SYS_CLKRST_I2S*n*_TX/RX_CLK_EN to enable or disable the clock source.

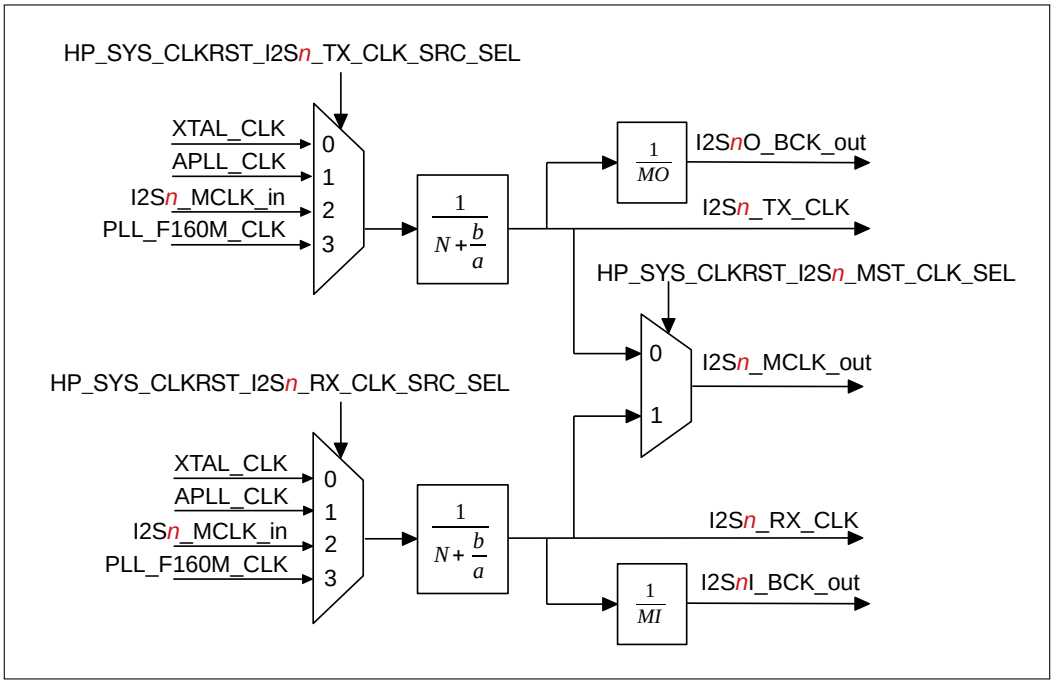


Figure 49.6-1. I2S*n* Clock Generator

The following formula shows the relation between I2S*n*_TX/RX_CLK frequency f_{I2Sn_TX/RX_CLK} and the divider clock source frequency $f_{I2Sn_CLK_S}$:

$$f_{I2Sn_TX/RX_CLK} = \frac{f_{I2Sn_CLK_S}}{N + \frac{b}{a}}$$

N is an integer value between 2 and 256. The value of N is mapped to that of HP_SYS_CLKRST_I2S*n*_TX/RX_DIV_N as follows:

- When HP_SYS_CLKRST_I2S*n*_TX/RX_DIV_N = 0, N = 256;
- When HP_SYS_CLKRST_I2S*n*_TX/RX_DIV_N = 1, N = 2;
- When HP_SYS_CLKRST_I2S*n*_TX/RX_DIV_N has any other value, N = HP_SYS_CLKRST_I2S*n*_TX/RX_DIV_N.

The values of "a" and "b" in fractional divider depend only on x, y, z, and yn1. The corresponding formulas are as follows:

- When b $<= \frac{a}{2}$, yn1 $= 0$, x $= floor([\frac{a}{b}]) - 1$, y $=$ a%b, z $=$ b;
- When b $> \frac{a}{2}$, yn1 $= 1$, x $= floor([\frac{a}{a - b}]) - 1$, y $=$ a%(a - b), z $=$ a - b.

The values of x, y, z, and yn1 are configured in HP_SYS_CLKRST_I2S*n*_TX/RX_DIV_X, HP_SYS_CLKRST_I2S*n*_TX/RX_DIV_Y, HP_SYS_CLKRST_I2S*n*_TX/RX_DIV_Z, and HP_SYS_CLKRST_I2S*n*_TX/RX_DIV_YN1.

To configure the integer divider, clear HP_SYS_CLKRST_I2S*n*_TX/RX_DIV_X and HP_SYS_CLKRST_I2S*n*_TX/RX_DIV_Z, then set HP_SYS_CLKRST_I2S*n*_TX/RX_DIV_Y to 1.

Note:
Using fractional divider may introduce some clock jitter.

In master TX mode, the serial clock BCK for I2S*n* TX unit is I2S*n*O_BCK_out divided from I2S*n*_TX_CLK which is:

$$f_{I2SnO_BCK_out} = \frac{f_{I2Sn_TX_CLK}}{MO}$$

"MO" is an integer value:

$$MO = I2Sn_TX_BCK_DIV_NUM + 1$$

Note:
Note that I2S_TX_BCK_DIV_NUM must not be configured as 1.

In master RX mode, the serial clock BCK for I2S*n* RX unit is I2S*n*I_BCK_out divided from I2S*n*_RX_CLK, which is:

$$f_{I2SnI_BCK_out} = \frac{f_{I2Sn_RX_CLK}}{MI}$$

"MI" is an integer value:

$$MI = I2S_RX_BCK_DIV_NUM + 1$$

Note:

- I2S_RX_BCK_DIV_NUM must not be configured as 1.
- In I2S slave mode, make sure f_{I2S_TX/RX_CLK} >= 8 × f_{BCK}. The I2S*n* module can output I2S*n*_MCLK_out as the master clock for peripherals.

49.7 I2S Reset

The units and FIFOs in the I2S*n* module are reset by the following bits.

- I2S*n* TX/RX units: Reset by the bits I2S_TX_RESET and I2S_RX_RESET;
- I2S*n* TX/RX FIFO: Reset by the bits I2S_TX_FIFO_RESET and I2S_RX_FIFO_RESET.

Note:
The I2S*n* module clock must be configured first before the module and FIFO are reset.

49.8 I2S Master/Slave Mode

The ESP32-P4 I2S*n* module can operate as a master or a slave in half-duplex and full-duplex communications, depending on the configuration of I2S_RX_SLAVE_MOD and I2S_TX_SLAVE_MOD.

- I2S_TX_SLAVE_MOD
  - 0: Master TX mode
  - 1: Slave TX mode
- I2S_RX_SLAVE_MOD
  - 0: Master RX mode
  - 1: Slave RX mode

49.8.1 Master/Slave TX Mode

- I2S*n* works as a master transmitter:
  - Set I2S_TX_START to start transmitting data. When this bit is set, the TX unit keeps driving the clock signal and serial data.
  - If I2S_TX_STOP_EN is set and all the data in FIFO is transmitted, the master stops transmitting data and clock signals.
  - If I2S_TX_STOP_EN is cleared and all the data in FIFO is transmitted, meanwhile no new data is filled into FIFO, the TX unit keeps transmitting the last data frame and clock signal.
  - Master stops transmitting data when the bit I2S_TX_START is cleared.
- I2S*n* works as a slave transmitter:
  - Set I2S_TX_START. Wait for the master BCK clock to enable a transmit operation.

- If I2S_TX_STOP_EN is set and all the data in FIFO is transmitted, then the slave keeps transmitting zeros, till the master stops providing BCK signal.
- If I2S_TX_STOP_EN is cleared and all the data in FIFO is transmitted, meanwhile no new data is filled into FIFO, the TX unit keeps transmitting the last data frame.
- If I2S_TX_START is cleared, slave keeps transmitting zeros till the master stops providing BCK clock signal.

49.8.2 Master/Slave RX Mode

- I2S*n* works as a master receiver:
  - Set I2S_RX_START to start receiving data. When this bit is set, the RX unit keeps outputting clock signal and sampling input data.
  - Configure I2S_RX_STOP_MODE to control the suspension of data reception:
    * 0: The RX unit only suspends data reception when I2S_RX_START is cleared.
    * 1: The RX unit suspends data reception when I2S_RX_START is cleared or the number of received bytes is greater than the value configured in I2S_RX_EOF_NUM_REG. When I2S_RX_START is not cleared, data reception can be restarted by setting I2S_RX_RESET to 1.
    * 2: The RX unit suspends data reception when I2S_RX_START is cleared or GDMA RX FIFO is full. When I2S_RX_START is not cleared and the GDMA RX FIFO is no longer full, data reception can be restarted by setting I2S_RX_RESET to 1.
  - RX unit stops receiving data when the bit I2S_RX_START is cleared.
- I2S*n* works as a slave receiver:
  - Set I2S_RX_START. Wait for the master BCK signal to start receiving data.
  - Configure I2S_RX_STOP_MODE to control data reception suspension:
    * 0: The RX unit only suspends data reception when I2S_RX_START is cleared.
    * 1: The RX unit suspends data reception when the number of received bytes is greater than the value configured in I2S_RX_EOF_NUM_REG. When I2S_RX_START is not cleared, data reception can be restarted by setting I2S_RX_RESET to 1.
    * 2: The RX unit suspends data reception when I2S_RX_START is cleared or GDMA RX FIFO is full. When I2S_RX_START is not cleared and the GDMA RX FIFO is no longer full, data reception can be restarted by setting I2S_RX_RESET to 1.
  - The RX unit stops receiving data when the bit I2S_RX_START is cleared.

49.9 Transmitting Data

Note:
Updating the configuration described in this and subsequent sections requires to set I2S_TX_UPDATE accordingly to synchronize registers from APB clock domain to TX clock domain. For more detailed configurations, see Section 49.13.1.

In TX mode, I2S*n* first reads data through GDMA and transmits these data out via output signals according to the configured data mode and channel mode.

49.9.1 Data Format Control

Data format is controlled in the following phases:

- Phase I: Read data from memory and write it to TX FIFO;
- Phase II: Read the TX data from TX FIFO and convert the data according to the output data mode;
- Phase III: Clock out the TX data serially.

49.9.1.1 Bit Width Control of Channel Valid Data

The bit width of the valid data in each channel is determined by I2S_TX_BITS_MOD and I2S_TX_24_FILL_EN. For details, see the table below.

Table 49.9-1. Bit Width of Channel Valid Data

Channel Valid Data Width	I2S_TX_BITS_MOD	I2S_TX_24_FILL_EN
32	31	x*
	23	1
24	23	0
16	15	x
8	7	x

* "x" represents that this value is ignored.

49.9.1.2 Endian Control of Channel Valid Data

When I2S*n* reads data through GDMA, the data endian under various data width is controlled by I2S_TX_BIG_ENDIAN. Table 49.9-2 shows how I2S_TX_BIG_ENDIAN controls the reading of the data with different valid data widths of the channel.

Table 49.9-2. Endian of Channel Valid Data

Channel Valid Data Width	Original Data	Endian of Processed Data	I2S_TX_BIG_ENDIAN
32	{B3, B2, B1, B0}	{B3, B2, B1, B0}	0
		{B0, B1, B2, B3}	1
24	{B2, B1, B0}	{B2, B1, B0}	0
		{B0, B1, B2}	1
16	{B1, B0}	{B1, B0}	0
		{B0, B1}	1
8	{B0}	{B0}	x

Note:
B0, B1, B2, B3 each represents an 8-bit data, and the symbol {} indicates that the bytes are combined together. For

example, {B3, B2, B1, B0} represents a 32-bit data, wherein B0 represents bit 0-7, B1 represents bit 8-15, B2 represents bit 16-23, and B3 represents bit 24-31.

49.9.1.3 A-law/μ-law Compression and Decompression

ESP32-P4 I2S*n* compresses/decompresses the valid data into 32-bit by A-law or by μ-law. If the bit width of valid data is smaller than 32, zeros are filled to the extra high bits of the data to be compressed/decompressed by default.

Note:
Extra high bits here mean the bits[31: channel valid data width] of the data to be compressed/decompressed.

Configure I2S_TX_PCM_BYPASS:

- 0: Compress or decompress the data
- 1: Do not compress or decompress the data

Configure I2S_TX_PCM_CONF:

- 0: Decompress the data using A-law
- 1: Compress the data using A-law
- 2: Decompress the data using μ-law
- 3: Compress the data using μ-law

At this point, the first phase of data format control is completed.

49.9.1.4 Bit Width Control of Channel TX Data

The TX data width in each channel is determined by I2S_TX_TDM_CHAN_BITS.

- If TX data width in each channel is larger than the valid data width, zeros will be filled to these extra bits. Configure I2S_TX_LEFT_ALIGN:
  - 0: The valid data is at the lower bits of TX data. Zeros are filled into higher bits of TX data;
  - 1: The valid data is at the higher bits of TX data. Zeros are filled into lower bits of TX data.
- If the TX data width in each channel is smaller than the valid data width, only the lower bits of valid data are sent out, and the higher bits are discarded.

At this point, the second phase of data format control is completed.

49.9.1.5 Bit Order Control of Channel Data

The data bit order in each channel is controlled by I2S_TX_BIT_ORDER:

- 0: Not reverse the valid data bit order;
- 1: Reverse the valid data bit order.

At this point, the data format control is completed. The data after format control will be sent sequentially from high to low. Figure 49.9-1 shows the complete process of TX data format control.

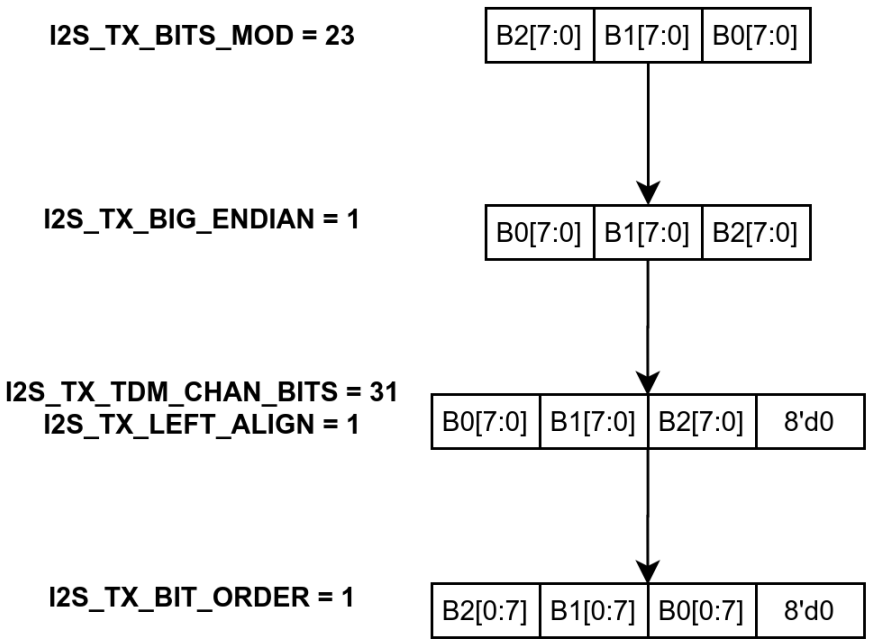


Figure 49.9-1. TX Data Format Control

49.9.2 Channel Mode Control

ESP32-P4 I2S*n* supports both TDM TX mode and PDM TX mode. Set I2S_TX_TDM_EN to enable TDM TX mode, or set I2S_TX_PDM_EN to enable PDM TX mode.

Note:
I2S_TX_TDM_EN and I2S_TX_PDM_EN must not be cleared or set simultaneously.

49.9.2.1 TDM TX Mode

In TDM TX mode, I2S*n* supports up to 16 channels to transmit data. The total number of TX channels in use is controlled by I2S_TX_TDM_TOT_CHAN_NUM. For example, If I2S_TX_TDM_TOT_CHAN_NUM is set to 5, six channels in total (channel 0 ~ 5) will be used to transmit data. See Figure 49.9-2.

In these TX channels, if I2S_TX_TDM_CHANn_EN is set to:

- 1: This channel transmits the channel data out;
- 0: The TX data to be sent by this channel is controlled by I2S_TX_CHAN_EQUAL:
  - 1: The data of the previous channel is sent out;
  - 0: The data stored in I2S_SINGLE_DATA is sent out.

Note:
Most stereo I2S*n* codecs can be controlled by setting the I2S*n* module into 2-channel mode under TDM standard.

In TDM TX master mode, WS signal is controlled by I2S_TX_WS_IDLE_POL and I2S_TX_TDM_WS_WIDTH:

- I2S_TX_WS_IDLE_POL: The default level of WS signal;
- I2S_TX_TDM_WS_WIDTH: The cycles the WS default level lasts for when transmitting all channel data.

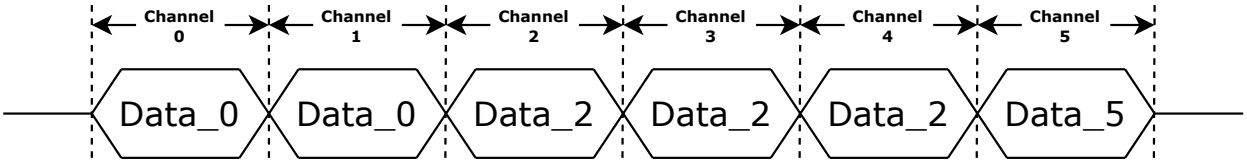
I2S_TX_HALF_SAMPLE_BITS x 2 is equal to the BCK cycles in one WS period.

TDM Channel Configuration Example

In this example, the register configuration is as follows:

- I2S_TX_TDM_CHAN_NUM = 5, i.e., channel 0 ~ 5 are used to transmit data.
- I2S_TX_CHAN_EQUAL = 1, i.e., data of the previous channel will be transmitted if the bit I2S_TX_TDM_CHAN*n*_EN (*n* = 0 ~ 5) is cleared.
- I2S_TX_TDM_CHAN0/2/5_EN = 1, i.e., these channels transmit their channel data out.
- I2S_TX_TDM_CHAN1/3/4_EN = 0, i.e., these channels transmit the previous channel's data out.

Once the configuration is done, data is transmitted as follows.



I2S_TX_TDM_CHAN_NUM = 5; I2S_TX_CHAN_EQUAL = 1;
I2S_TX_TDM_CHAN0_EN = 1; I2S_TX_TDM_CHAN1_EN = 0; I2S_TX_TDM_CHAN2_EN = 1;
I2S_TX_TDM_CHAN3_EN = 0; I2S_TX_TDM_CHAN4_EN = 0; I2S_TX_TDM_CHAN5_EN = 1;

Figure 49.9-2. TDM Channel Control

49.9.2.2 PDM TX Mode

In PDM TX mode, I2S*n* supports both PDM raw data transmission and PCM-to-PDM data format conversion (the latter only supported by I2S*0*).

In PDM TX mode, fetching data through GDMA is controlled by I2S_TX_MONO and I2S_TX_MONO_FST_VLD. See Table 49.9-3. Configure the two bits according to the data stored in memory, be it the single-channel or dual-channel data.

Table 49.9-3. Data-Fetching Control in PDM Mode

Data-Fetching Control Option	Mode	I2S_TX_MONO	I2S_TX_MONO_FST_VLD
Post data-fetching request to GDMA at any edge of WS signal	Stereo mode	0	x
Post data-fetching request to GDMA only at the second half period of WS signal	Mono mode	1	0
Post data-fetching request to GDMA only at the first half period of WS signal	Mono mode	1	1

When I2S*n* is in PDM TX master mode, the default level of WS signal is controlled by I2S_TX_WS_IDLE_POL, and the WS signal frequency is half of the BCK signal frequency. The configuration of WS signal is similar to that of BCK signal. Please refer to Section 49.6 and Figure 49.6.

When **transmitting raw PDM data**, the I2S*n* channel mode is controlled by I2S_TX_CHAN_MOD and I2S_TX_WS_IDLE_POL. See the table below.

Table 49.9-4. I2S*n* Channel Control for Raw PDM Data

Channel Control Option	Left Channel	Right Channel	Mode Control Field[1]	Channel Select Bit[2]
Stereo mode	Transmit the left channel data	Transmit the right channel data	0	x
Mono mode	Transmit the left channel data	Transmit the left channel data	1	0
	Transmit the right channel data	Transmit the right channel data	1	1
	Transmit the right channel data	Transmit the right channel data	2	0
	Transmit the left channel data	Transmit the left channel data	2	1
	Transmit the value of "single"[3]	Transmit the right channel data	3	0
	Transmit the left channel data	Transmit the value of "single"	3	1
	Transmit the left channel data	Transmit the value of "single"	4	0
	Transmit the value of "single"	Transmit the right channel data	4	1

[1] I2S_TX_CHAN_MOD

[2] I2S_TX_WS_IDLE_POL

[3] The "single" value is equal to the value of I2S_SINGLE_DATA.

If the **PCM-to-PDM converter is enabled** for I2S*O* , the PCM data through GDMA is converted to PDM data and then output in PDM signal format. Configure I2S_PCM2PDM_CONV_EN to enable the converter. The register configuration for the PCM-to-PDM converter is as follows:

- Configure 1-line PDM output format or 1-/2-line DAC output mode as the table below:

Table 49.9-5. I2S*O* PCM-to-PDM Data Output

Channel Output Format	I2S_TX_PDM_DAC_MODE_EN	I2S_TX_PDM_DAC_2OUT_EN
1-line PDM output format[1]	0	x
1-line DAC output format[2]	1	0
2-line DAC output format	1	1

Note:

1. In PDM output format, SD data of two channels is sent out in one WS period.
2. In DAC output format, SD data of one channel is sent out in one WS period.
3. 1-line PDM/DAC output format uses I2S*n*O_Data_out as the data output line.
4. 2-line DAC output format uses I2S*n*O_Data_out and I2S*n*O_Data1_out as data output lines.
5. In terms of application, the PDM output format is targeted at applications that require clock signals and can decode PDM format codecs, while the DAC output format is targeted at applications that do not rely on clock signals, do not have PDM format codecs, and can recover analog waveforms directly through low-pass filtering.

- Configure sampling frequency and upsampling rate as below:

When the I2S0 PCM-to-PDM converter is enabled, PDM clock frequency is equal to BCK frequency. The relation of sampling frequency ($f_{Sampling}$, the sampling frequency of PCM data) and BCK frequency (the sampling frequency of PDM data) is as follows:

$$f_{Sampling} = \frac{f_{BCK}}{OSR}$$

Upsampling rate (OSR) is related to I2S_TX_PDM_SINC_OSR2 as follows:

$$OSR = I2S_TX_PDM_SINC_OSR2 \times 64$$

Sampling frequency $f_{Sampling}$ is related to I2S_TX_PDM_FS as follows:

$$f_{Sampling} = I2S_TX_PDM_FS \times 100$$

Configure the registers according to the needed sampling frequency, upsampling rate, and PDM clock frequency.

PDM Channel Configuration Example

In this example of I2S0, the register configuration is as follows.

- I2S_PCM2PDM_CONV_EN = 0, i.e., transmit the raw PDM data.
- I2S_TX_MONO = 0, i.e., data is fetched from memory via GDMA in both the high and low levels of WS.
- I2S_TX_CHAN_MOD = 2, i.e., mono mode is selected.
- I2S_TX_WS_IDLE_POL = 1, i.e., both the left and right channels transmit the left channel data, and the right channel data will be discarded.

Once the configuration is done, assume that the data in memory **after data format control** is:

Left	Right	Left	Right	...	Left	Right

Note:

1. The data above refers to the processed data after data format control instead of the original data.
2. "Left" and "Right" represent channel data, and their bit widths are channel valid data width. Please refer to Section 49.9.1.

Then the channel data is transmitted after channel mode control as follows.

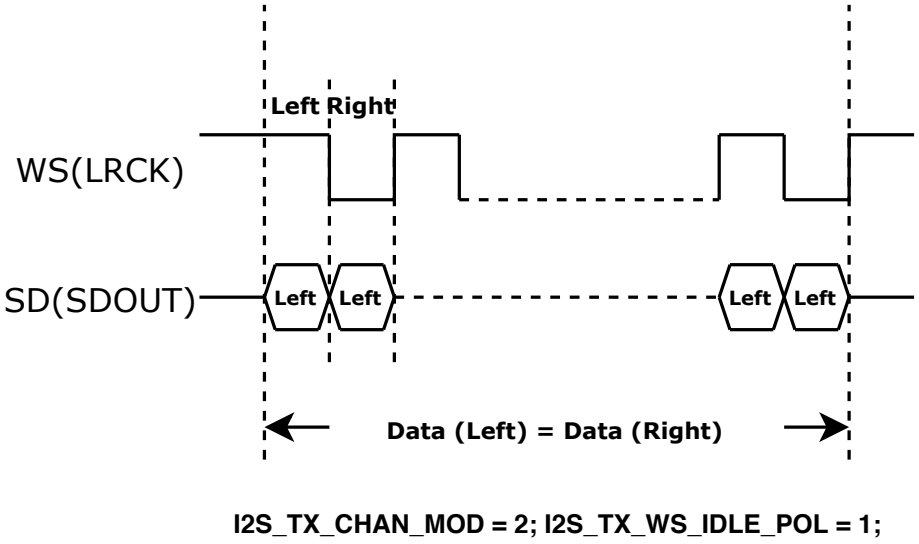


Figure 49.9-3. PDM Channel Control Example

49.9.3 Synchronous Counter

ESP32-P4 I2S*n* TX unit contains synchronous counters to reflect the current I2S*n* data transmission status. The counters can be used to detect I2S*n* transmit data errors or to synchronize data when multiple devices are using I2S*n* for data transmission.

ESP32-P4 I2S*n* contains two types of synchronous counters:

- I2S_TX_FIFO_CNT
  - Each time the I2S*n* TX FIFO reads data, the counter is incremented by 1.
  - The counter can be reset by setting I2S_TX_FIFO_CNT_RST.
  - The counter is 31-bit wide and is automatically reset to zero on overflow.
- I2S_TX_BCK_CNT
  - Each time I2S*n* TX transmits a BCK cycle, the counter is incremented by 1.
  - The counter can be reset by setting I2S_TX_BCK_CNT_RST.
  - The counter is 31-bit wide and is automatically reset to zero on overflow.

Note:

- Each time the I2S*n* TX FIFO transmits data, a channel data is sent. Therefore, when I2S*n* transmits the data correctly, I2S_TX_BCK_CNT = I2S_TX_FIFO_CNT × the bit width of channel TX data (see Section 49.9.1.4).
- When the I2S*n* TX unit transmits the data, there is some delay between fetching data from the FIFO and transmitting it to the line, so the value of I2S_TX_BCK_CNT might be a little different from that of I2S_TX_FIFO_CNT.

Application scenarios

1. I2S TX synchronization for multiple devices
   When multiple devices transmit data through the I2S TX at the same time, desynchronization may occur after a long period of time due to small frequency differences in the clock sources. In this case, read the

I2S_TX_FIFO_CNT value of each device to determine the location of the audio data being sent to perform synchronization.

2. I2S TX stability check
   When the device is in an application scenario with high GDMA usage, there may be numbers missing in I2S TX occasionally due to multiplexing arbitration. In this case, check the ratio of I2S_TX_BCK_CNT to I2S_TX_FIFO_CNT to confirm the stability of the device when running I2S TX.

49.10 Receiving Data

In RX mode, I2S*n* first reads data from the peripheral interface and then stores the data in memory via GDMA according to the configured channel mode and data mode.

49.10.1 Channel Mode Control

ESP32-P4 I2S*n* supports both TDM RX mode and PDM RX mode. Set I2S_RX_TDM_EN to enable TDM RX mode, or set I2S_RX_PDM_EN to enable PDM RX mode.

Note:
I2S_RX_TDM_EN and I2S_RX_PDM_EN must not be cleared or set simultaneously.

49.10.1.1 TDM RX Mode

In TDM RX mode, I2S*n* supports up to 16 channels to input data. The total number of RX channels in use is controlled by I2S_RX_TDM_TOT_CHAN_NUM. For example, if I2S_RX_TDM_TOT_CHAN_NUM is set to 5, channel 0 ~ 5 will be used to receive data.

In these RX channels, if I2S_RX_TDM_CHAN*n*_EN is set to:

- 1: The channel data is valid and will be stored into RX FIFO;
- 0: The channel data is invalid and will not be stored into RX FIFO.

In TDM master mode, WS signal is controlled by I2S_RX_WS_IDLE_POL and I2S_RX_TDM_WS_WIDTH.

- I2S_RX_WS_IDLE_POL: The default level of WS signal;
- I2S_RX_TDM_WS_WIDTH: The cycles the WS default level lasts for when receiving all channel data.

I2S_RX_HALF_SAMPLE_BITS x 2 is equal to the BCK cycles in one WS period.

49.10.1.2 PDM RX Mode

In PDM RX mode, I2S*n* supports both PDM raw data reception and PDM-to-PCM data format conversion (the latter only supported by I2S*0*).

In PDM RX mode, I2S*n* converts the serial data from channels to the data to be entered into memory.

In PDM RX master mode, the default level of the WS signal is controlled by I2S_RX_WS_IDLE_POL. WS frequency is half of the BCK frequency. The configuration of the BCK signal is similar to that of WS signal as described in Section 49.6. Note, in PDM RX mode, the value of I2S_RX_HALF_SAMPLE_BITS must be same as that of I2S_RX_BITS_MOD.

When the **PDM-to-PCM converter is enabled** for I2S*0*, the received PDM data is converted to PCM data and controlled according to the data mode. Configure I2S_RX_PDM2PCM_EN to enable this converter. The register configuration for PDM-to-PCM converter is as follows:

- Configure sampling frequency and downsampling rate.
  When I2S*0* PDM-to-PCM converter is enabled, PDM clock frequency is:
  - in master mode: PDM clock frequency is equal to BCK frequency.
  - in slave mode: PDM clock is provided by external device.

  The sampling frequency ($f_{Sampling}$) is related to PDM clock frequency as follows:

$$f_{Sampling} = \frac{f_{PDM}}{DSR}$$

  Downsampling rate (DSR) is related to I2S_RX_PDM_SINC_DSR_16_EN as follows:

$$DSR = I2S_RX_PDM_SINC_DSR_16_EN \times 64$$

  Configure the registers according to needed master/slave mode, sampling frequency, and downsampling rate.
- Configure valid channels.
  When the PDM-to-PCM converter is enabled, input signals from eight channels are supported at most. See Table 49.10-1 for the register configuration and related channels.

Table 49.10-1. PDM-to-PCM Data Input

Input Data Signal	Channel	Enable Register
I2S*0*I_Data_in	Left channel	I2S_RX_TDM_PDM_CHAN0_EN
	Right channel	I2S_RX_TDM_PDM_CHAN1_EN
I2S*0*I1_Data_in	Left channel	I2S_RX_TDM_PDM_CHAN2_EN
	Right channel	I2S_RX_TDM_PDM_CHAN3_EN
I2S*0*I2_Data_in	Left channel	I2S_RX_TDM_PDM_CHAN4_EN
	Right channel	I2S_RX_TDM_PDM_CHAN5_EN
I2S*0*I3_Data_in	Left channel	I2S_RX_TDM_PDM_CHAN6_EN
	Right channel	I2S_RX_TDM_PDM_CHAN7_EN

49.10.2 Data Format Control

The data format of I2S*n* is controlled in the following phases:

- Phase I: Serial input data is converted into the data to be saved to RX FIFO;
- Phase II: The data is read from RX FIFO and converted according to the input data mode.

49.10.2.1 Bit Order Control of Channel Data

The channel data will be stored as the data to be input in order from high to low. The data bit order in each channel is controlled by I2S_RX_BIT_ORDER:

- 0: The bit order of the data to be input is not reversed;
- 1: The bit order of the data to be input is reversed.

At this point, the first phase of data format control is completed. The data to be input after bit order control is stored in the RX FIFO.

49.10.2.2 Bit Width Control of Channel Storage (Valid) Data

The storage data width in each channel is controlled by I2S_RX_BITS_MOD and I2S_RX_24_FILL_EN. See the table below.

Table 49.10-2. Channel Storage Data Width

Channel Storage Data Width	I2S_RX_BITS_MOD	I2S_RX_24_FILL_EN
32	31	x
	23	1
24	23	0
16	15	x
8	7	x

49.10.2.3 Bit Width Control of Channel RX Data

The RX data width in each channel is determined by I2S_RX_TDM_CHAN_BITS.

- If the storage data width in each channel is smaller than the received (RX) data width, then only the bits within the storage data width is saved into memory. Configure I2S_RX_LEFT_ALIGN to:
  - 0: Only the lower bits of the received data within the storage data width is stored to memory;
  - 1: Only the higher bits of the received data within the storage data width is stored to memory.
- If the received data width is smaller than the storage data width in each channel, the higher bits of the received data will be filled with zeros and then the data is saved to memory.

49.10.2.4 Endian Control of Channel Storage Data

The received data is then converted into storage data (to be stored to memory) after some processing, such as discarding extra bits or filling zeros in missing bits. The endian of the storage data is controlled by I2S_RX_BIG_ENDIAN under various data width. See the table below.

Table 49.10-3. Channel Storage Data Endian

Channel Storage Data Width	Original Data	Endian of Processed Data	I2S_RX_BIG_ENDIAN
32	{B3, B2, B1, B0}	{B3, B2, B1, B0}	0
		{B0, B1, B2, B3}	1
24	{B2, B1, B0}	{B2, B1, B0}	0
		{B0, B1, B2}	1
16	{B1, B0}	{B1, B0}	0
		{B0, B1}	1
8	{B0}	{B0}	x

49.10.2.5 A-law/μ-law Compression and Decompression

ESP32-P4 I2S*n* compresses/decompresses the storage data in 32-bit by A-law or by μ-law. By default, zeros are filled into high bits.

Configure I2S_RX_PCM_BYPASS:

- 0: Compress or decompress the data
- 1: Do not compress or decompress the data

Configure I2S_RX_PCM_CONF:

- 0: Decompress the data using A-law
- 1: Compress the data using A-law
- 2: Decompress the data using μ-law
- 3: Compress the data using μ-law

At this point, the data format control is completed. Data then is stored into memory via GDMA.

49.11 Event Task Matrix Feature

ESP32-P4 I2S*n* supports the Event Task Matrix (ETM) function, which allows I2S*n*'s ETM tasks to be triggered by any peripherals' ETM events, or I2S*n*'s ETM events to trigger any peripherals' ETM tasks. This section introduces the ETM tasks and events related to I2S*n*. For more information, please refer to Chapter 14 Event Task Matrix (ETM).

I2S*n* can receive the following ETM tasks:

- I2S*n*_TASK_START_TX: Enables I2S*n* TX for data transfer.
- I2S*n*_TASK_START_RX: Enables I2S*n* RX for data transfer.
- I2S*n*_TASK_STOP_TX: Stops I2S*n* TX data transfer.
- I2S*n*_TASK_STOP_RX: Stops I2S*n* RX data transfer.

I2S*n* can generate the following ETM events:

- I2S*n*_EVT_TX_DONE: Indicates that I2S*n* TX has completed data transmission.
- I2S*n*_EVT_RX_DONE: Indicates that I2S*n* RX has completed data reception.
- I2S*n*_EVT_X_WORDS_SENT: Indicates that the word number sent by I2S*n* TX is equal to or larger than the value set by I2S_ETM_TX_SEND_WORD_NUM.
- I2S*n*_EVT_X_WORDS_RECEIVED: Indicates that the word number received by I2S*n* RX is equal to or larger than the value set by I2S_ETM_RX_RECEIVE_WORD_NUM.

In practical applications, I2S*n*'s ETM events can trigger its own ETM tasks. For example, the I2S*n*_EVT_X_WORDS_SENT event can trigger the I2S*n*_TASK_STOP_TX task, and in this way stop the I2S*n* operation through ETM.

49.12 I2S Interrupts

ESP32-P4's I2S*n* can generate the I2S*n*_INTR interrupt signal that will be sent to the Interrupt Matrix. There are several internal interrupt sources from I2S*n* that can generate the above interrupt signal(s) as follows:

- I2S*n*_TX_HUNG_INT: Triggered when data transmission is timed out. For example, if the I2S*n* module is configured as TX slave mode, but the master does not provide BCK or WS signals for a long time (specified in I2S_LC_HUNG_CONF_REG), this interrupt will be triggered.
- I2S*n*_RX_HUNG_INT: Triggered when the data reception is timed out. For example, if the I2S*n* module is configured as RX slave mode, but the master does not transmit data for a long time (specified in I2S_LC_HUNG_CONF_REG), this interrupt will be triggered.
- I2S*n*_TX_DONE_INT: Triggered when data transmission is completed.
- I2S*n*_RX_DONE_INT: Triggered when the data reception is completed.

Note:
For definitions of *interrupt*, *interrupt signal*, *interrupt source*, and their correlations, please refer to Chapter 13 *Interrupt Matrix* > Section 13.2 *Interrupt Terminology in ESP32-P4*.

Each interrupt source can be configured by a common set of registers that are described in Section *Interrupt Configuration Registers*. The specific registers can be found in Section 49.14 *Register Summary*.

49.13 Software Configuration Process

49.13.1 Configure I2S as TX Mode

Follow the steps below to configure I2S*n* as TX mode via software:

1. Configure the clock as described in Section 49.6.
2. Configure signal pins according to Table 49.4-1.
3. Select the mode needed by configuring I2S_TX_SLAVE_MOD.
   - 0: Master TX mode
   - 1: Slave TX mode
4. Set needed TX data mode and TX channel mode as described in Section 49.9, and then set I2S_TX_UPDATE.
5. Reset TX unit and TX FIFO as described in Section 49.7.
6. Enable corresponding interrupts. See Section 49.12.
7. Configure GDMA outlink.
8. Set I2S_TX_STOP_EN if needed. For more information, please refer to Section 49.8.1.
9. Start transmitting data:
   - In master mode, wait till I2S*n* slave gets ready, then set I2S_TX_START to start transmitting data;

- In slave mode, set I2S_TX_START. When the I2S*n* master supplies BCK and WS signals, I2S*n* slave starts transmitting data.

10. Wait for the interrupt signals set in Step 6, or check whether the transfer is completed by querying I2S_TX_IDLE:
    - 0: Transmitter is working;
    - 1: Transmitter is in idle state.
11. Clear I2S_TX_START to stop data transfer.

49.13.2 Configure I2S as RX Mode

Follow the steps below to configure I2S*n* as RX mode via software:

1. Configure the clock as described in Section 49.6.
2. Configure signal pins according to Table 49.4-1.
3. Select the mode needed by configuring I2S_RX_SLAVE_MOD.
    - 0: Master RX mode
    - 1: Slave RX mode
4. Set needed RX data mode and RX channel mode as described in Section 49.10, and then set I2S_RX_UPDATE.
5. Reset the RX unit and its FIFO according to Section 49.7.
6. Enable the corresponding interrupts. See Section 49.12.
7. Configure GDMA inlink and set the length of RX data by configuring I2S_RX_EOF_NUM_REG.
8. Start receiving data:
    - In master mode, when the slave is ready, set I2S_RX_START to start receiving data.
    - In slave mode, set I2S_RX_START to start receiving data when BCK and WS signals are received from the master.
9. The received data is then stored to the specified memory address according to the configuration of GDMA. Then the corresponding interrupt set in step 6 is generated.

49.14 Register Summary

The addresses in this section are relative to I2S*n* base address provided in Table 9.3-2 in Chapter 9 *System and Memory*.

The abbreviations given in Column **Access** are explained in Section *Access Types for Registers*.

Name	Description	Address	Access
Interrupt registers			
I2S_INT_RAW_REG	I2S interrupt raw register	0x000C	R/SS/WTC
I2S_INT_ST_REG	I2S interrupt status register	0x0010	RO
I2S_INT_ENA_REG	I2S interrupt enable register	0x0014	R/W
I2S_INT_CLR_REG	I2S interrupt clear register	0x0018	WT
RX control and configuration registers			
I2S_RX_CONF_REG	I2S RX configuration register	0x0020	varies
I2S_RX_CONF1_REG	I2S RX configuration register 1	0x0028	R/W
I2S_RX_PDM2PCM_CONF_REG	I2S RX PDM-to-PCM configuration register	0x0048	R/W
I2S_RX_TDM_CTRL_REG	I2S TX TDM mode control register	0x0050	R/W
I2S_RXEOF_NUM_REG	I2S RX data number control register	0x0064	R/W
TX control and configuration registers			
I2S_TX_CONF_REG	I2S TX configuration register	0x0024	varies
I2S_TX_CONF1_REG	I2S TX configuration register 1	0x002C	R/W
I2S_TX_PCM2PDM_CONF_REG	I2S TX PCM-to-PDM configuration register (for I2S*0* only)	0x0040	R/W
I2S_TX_PCM2PDM_CONF1_REG	I2S TX PCM-to-PDM configuration register (for I2S*0* only)	0x0044	R/W
I2S_TX_TDM_CTRL_REG	I2S TX TDM mode control register	0x0054	R/W
RX clock and timing register			
I2S_RX_TIMING_REG	I2S RX timing control register (some fields for I2S*0* only)	0x0058	R/W
TX clock and timing register			
I2S_TX_TIMING_REG	I2S TX timing control register (some fields for I2S*0* only)	0x005C	R/W
Control and configuration registers			
I2S_LC_HUNG_CONF_REG	I2S timeout configuration register	0x0060	R/W
I2S_CONF_SIGLE_DATA_REG	I2S single data register	0x0068	R/W
TX status register			
I2S_STATE_REG	I2S TX status register	0x006C	RO
ETM register			
I2S_ETM_CONF_REG	I2S ETM configuration register	0x0070	R/W
Sync counter registers			
I2S_FIFO_CNT_REG	I2S sync counter register	0x0074	varies
I2S_BCK_CNT_REG	I2S sync counter register	0x0078	varies
Clock registers			
I2S_CLK_GATE_REG	Clock gate register	0x007C	R/W

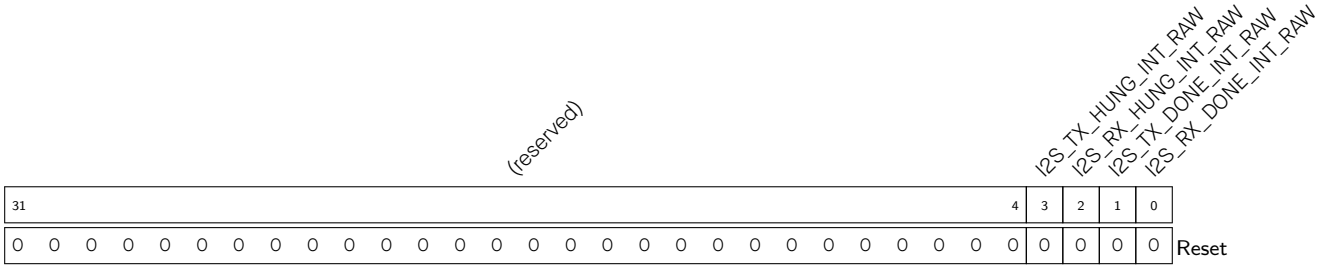
Name	Description	Address	Access
Version register			
I2S_DATE_REG	Version control register	0x0080	R/W

49.15 Registers

The addresses in this section are relative to I2S*n* base address provided in Table 9.3-2 in Chapter 9 *System and Memory*.

For how to program reserved fields, please refer to Section *Programming Reserved Register Field*.

Register 49.1. I2S_INT_RAW_REG (0x000C)



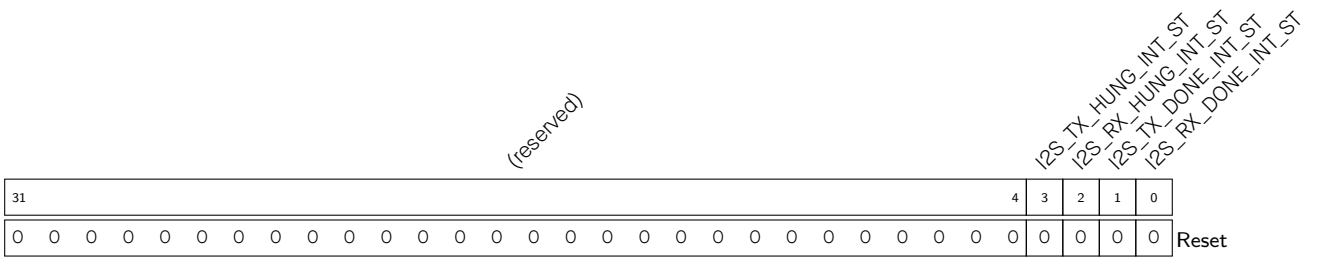
I2S_RX_DONE_INT_RAW The raw interrupt status of the I2S_RX_DONE_INT interrupt. (R/SS/WTC)

I2S_TX_DONE_INT_RAW The raw interrupt status of the I2S_TX_DONE_INT interrupt. (R/SS/WTC)

I2S_RX_HUNG_INT_RAW The raw interrupt status of the I2S_RX_HUNG_INT interrupt. (R/SS/WTC)

I2S_TX_HUNG_INT_RAW The raw interrupt status of the I2S_TX_HUNG_INT interrupt. (R/SS/WTC)

Register 49.2. I2S_INT_ST_REG (0x0010)



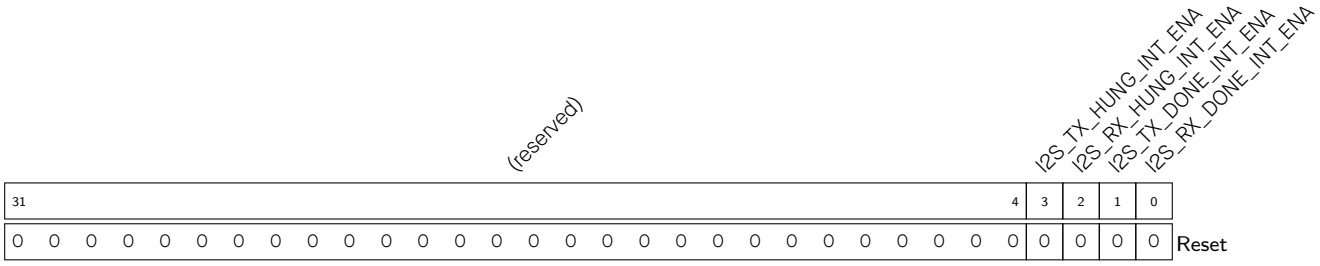
I2S_RX_DONE_INT_ST The masked interrupt status of the I2S_RX_DONE_INT interrupt. (RO)

I2S_TX_DONE_INT_ST The masked interrupt status of the I2S_TX_DONE_INT interrupt. (RO)

I2S_RX_HUNG_INT_ST The masked interrupt status of the I2S_RX_HUNG_INT interrupt. (RO)

I2S_TX_HUNG_INT_ST The masked interrupt status of the I2S_TX_HUNG_INT interrupt. (RO)

Register 49.3. I2S_INT_ENA_REG (0x0014)



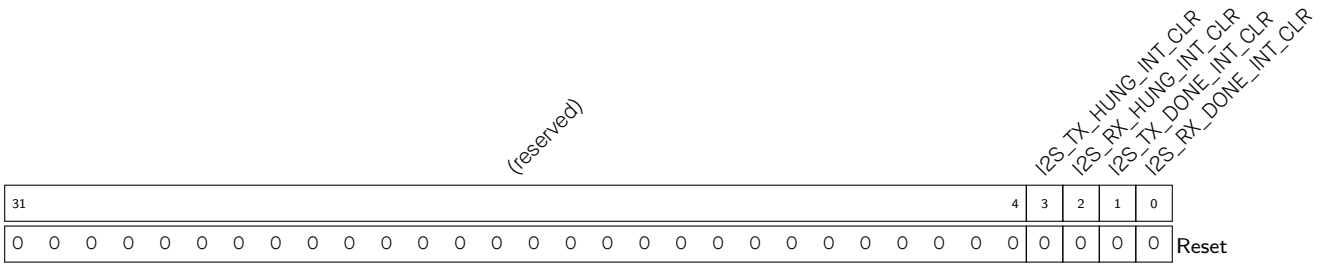
I2S_RX_DONE_INT_ENA Write 1 to enable the I2S_RX_DONE_INT interrupt. (R/W)

I2S_TX_DONE_INT_ENA Write 1 to enable the I2S_TX_DONE_INT interrupt. (R/W)

I2S_RX_HUNG_INT_ENA Write 1 to enable the I2S_RX_HUNG_INT interrupt. (R/W)

I2S_TX_HUNG_INT_ENA Write 1 to enable the I2S_TX_HUNG_INT interrupt. (R/W)

Register 49.4. I2S_INT_CLR_REG (0x0018)



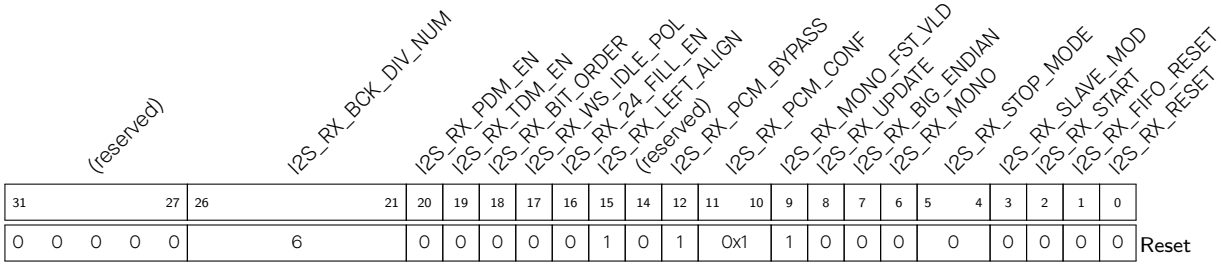
I2S_RX_DONE_INT_CLR Write 1 to clear the I2S_RX_DONE_INT interrupt. (WT)

I2S_TX_DONE_INT_CLR Write 1 to clear the I2S_TX_DONE_INT interrupt. (WT)

I2S_RX_HUNG_INT_CLR Write 1 to clear the I2S_RX_HUNG_INT interrupt. (WT)

I2S_TX_HUNG_INT_CLR Write 1 to clear the I2S_TX_HUNG_INT interrupt. (WT)

Register 49.5. I2S_RX_CONF_REG (0x0020)



I2S_RX_RESET Configures whether to reset RX unit.

0: No effect

1: Reset

(WT)

I2S_RX_FIFO_RESET Configures whether to reset RX FIFO.

0: No effect

1: Reset

(WT)

I2S_RX_START Configures whether to start receiving data.

0: No effect

1: Start

(R/W/SC)

I2S_RX_SLAVE_MOD Configures whether to enable slave RX mode.

0: Enable master mode

1: Enable slave mode

(R/W)

I2S_RX_STOP_MODE Configures when I2S RX stop data reception.

0: Only stops when I2S_RX_START is cleared

1: Stops when I2S_RX_START is cleared or the number of received bytes is greater than the value configured in I2S_RX_EOF_NUM_REG

2: Stops when I2S_RX_START is cleared or GDMA RX FIFO is full

(R/W)

I2S_RX_MONO Configures whether to enable RX unit in mono mode.

0: Disable

1: Enable

(R/W)

I2S_RX_BIG_ENDIAN Configures I2S RX byte endian.

0: Low address data is saved to low address

1: Low address data is saved to high address

(R/W)

Continued on the next page...

Register 49.5. I2S_RX_CONF_REG (0x0020)

Continued from the previous page...

I2S_RX_UPDATE Configures whether to update I2S RX registers from APB clock domain to I2S RX clock domain. This bit will be cleared by hardware after the register update is done.
0: No effect
1: Update
(R/W/SC)

I2S_RX_MONO_FST_VLD Configures the valid data channel in I2S RX mono mode.
0: The second channel data valid
1: The first channel data valid
(R/W)

I2S_RX_PCM_CONF Configures I2S RX compress/decompress mode.
0 (atol): A-law decompress
1 (ltoa): A-law compress
2 (utol): μ-law decompress
3 (ltou): μ-law compress
(R/W)

I2S_RX_PCM_BYPASS Configures whether to bypass the Compress/Decompress units for received data.
0: No effect
1: Bypass
(R/W)

I2S_RX_MSB_SHIFT Configures the timing between the WS signal and the MSB of data.
0: Align at the rising edge
1: The WS signal changes one BCK clock earlier
(R/W)

I2S_RX_LEFT_ALIGN Configures I2S RX alignment mode.
0: Right alignment mode
1: Left alignment mode
(R/W)

Continued on the next page...

Register 49.5. I2S_RX_CONF_REG (0x0020)

Continued from the previous page...

I2S_RX_24_FILL_EN Configures the bit number that the 24-bit channel data is stored to.

0: Store 24-bit channel data to 24 bits

1: Store 24-bit channel data to 32 bits (Extra bits are filled with zeros)

(R/W)

I2S_RX_WS_IDLE_POL Configures the relationship between WS level and which channel data to receive.

0: WS remains low when receiving left channel data and high when receiving right channel data

1: WS remains high when receiving left channel data and low when receiving right channel data

(R/W)

I2S_RX_BIT_ORDER Configures whether to reverse the bit order of the I2S RX data to be received.

0: Not reverse

1: Reverse

(R/W)

I2S_RX_TDM_EN Configures whether to enable I2S TDM RX mode.

0: Disable

1: Enable

(R/W)

I2S_RX_PDM_EN Configures whether to enable I2S PDM RX mode.

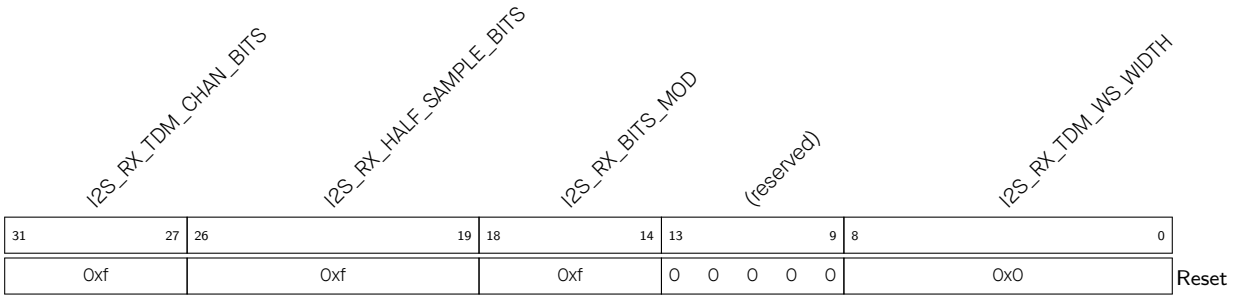
0: Disable

1: Enable

(R/W)

I2S_RX_BCK_DIV_NUM Configures the divider of BCK in RX mode. Note this divider must not be configured to 1. (R/W)

Register 49.6. I2S_RX_CONF1_REG (0x0028)



I2S_RX_TDM_WS_WIDTH Configures the width of I2S*n*I_WS_out (WS default level) at idle level in TDM mode. Width of I2S*n*I_WS_out at idle level in TDM mode = (I2S_RX_TDM_WS_WIDTH[8:0] +1) x T_BCK. (R/W)

I2S_RX_BITS_MOD Configures the valid data bit length of I2S RX channel.

7: All the valid channel data is in 8-bit mode

15: All the valid channel data is in 16-bit mode

23: All the valid channel data is in 24-bit mode

31: All the valid channel data is in 32-bit mode

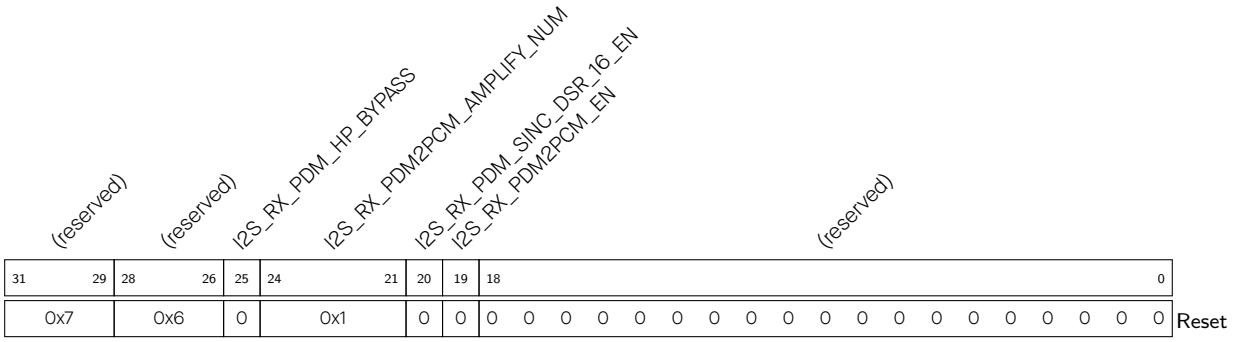
Other values are invalid.

(R/W)

I2S_RX_HALF_SAMPLE_BITS Configures I2S RX sample bits. BCK cycles in one WS period = I2S_RX_HALF_SAMPLE_BITS x 2. (R/W)

I2S_RX_TDM_CHAN_BITS Configures RX bit number for each channel in TDM mode. Bit number expected = I2S_RX_TDM_CHAN_BITS + 1. (R/W)

Register 49.7. I2S_RX_PDM2PCM_CONF_REG (for I2S0 only) (0x0048)



I2S_RX_PDM2PCM_EN Configures whether to enable the PDM-to-PCM converter in RX mode.

0: Disable

1: Enable

(R/W)

I2S_RX_PDM_SINC_DSR_16_EN Configures the downsampling rate of PDM RX filter group 1 module.

0: 64

1: 128

(R/W)

I2S_RX_PDM2PCM_AMPLIFY_NUM Configures the PDM-to-PCM RX amplify number. PCM data will be multiplied by this value before outputting.

(R/W)

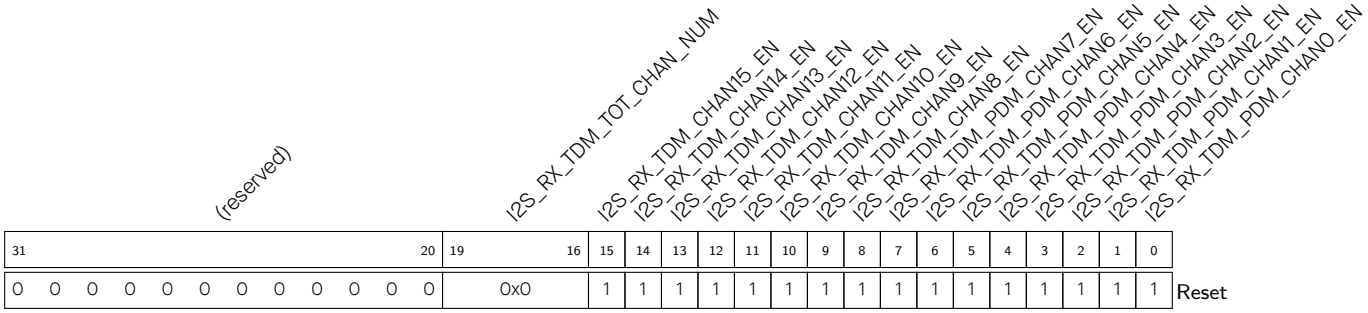
I2S_RX_PDM_HP_BYPASS Configures whether PDM-to-PCM RX bypasses the HP filter.

0: Not bypass

1: Bypass

(R/W)

Register 49.8. I2S_RX_TDM_CTRL_REG (0x0050)



I2S_RX_TDM_PDM_CHAN*n*_EN (*n*: 0-7) Configures whether to enable the valid data input of I2S RX TDM or PDM channel *n*.

0: Disable. Channel *n* only inputs 0

1: Enable

(R/W)

I2S_RX_TDM_CHAN*n*_EN (*n*: 8-15) Configures whether to enable the valid data input of I2S RX TDM channel *n*.

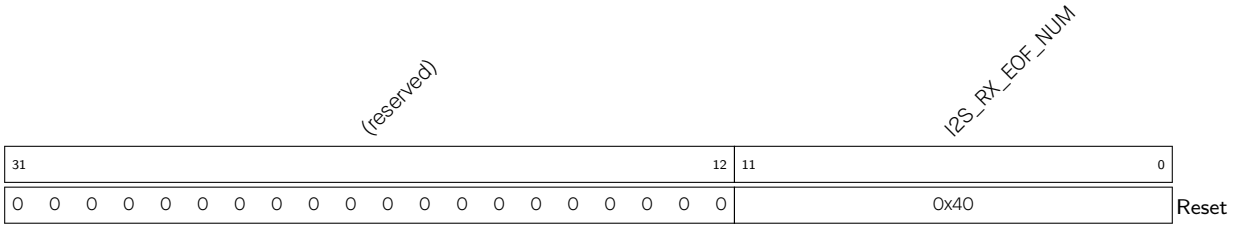
0: Disable. Channel *n* only inputs 0

1: Enable

(R/W)

I2S_RX_TDM_TOT_CHAN_NUM Configures the total number of channels in use in I2S RX TDM mode. Total channel number in use = I2S_RX_TDM_TOT_CHAN_NUM + 1. (R/W)

Register 49.9. I2S_RXEOF_NUM_REG (0x0064)



I2S_RX_EOF_NUM Configures the bit length of RX data. Bit length of RX data = (I2S_RX_BITS_MOD + 1) x (I2S_RX_EOF_NUM + 1). Once the received data reaches such bit length, an AHB_DMA_IN_SUC_EOF_CH*n*_INT interrupt is triggered in the configured GDMA RX channel. (R/W)

Register 49.10. I2S_TX_CONF_REG (0x0024)



I2S_TX_RESET Configures whether to reset TX unit.

0: No effect

1: Reset

(WT)

I2S_TX_FIFO_RESET Configures whether to reset TX FIFO.

0: No effect

1: Reset

(WT)

I2S_TX_START Configures whether to start transmitting data.

0: No effect

1: Start

(R/W/SC)

I2S_TX_SLAVE_MOD Configures whether to enable slave TX mode.

0: Enable master mode

1: Enable slave mode

(R/W)

I2S_TX_STOP_EN Configures whether to stop outputting the BCK signal and the WS signal when TX FIFO is empty.

0: No effect

1: Stop

(R/W)

I2S_TX_CHAN_EQUAL Configures whether to equalize left channel data and right channel data in I2S TX mono mode or TDM mode.

0: The I2S_SINGLE_DATA is invalid channel data in I2S TX mono mode or TDM mode

1: The left channel data is equal to right channel data in I2S TX mono mode or TDM mode

(R/W)

I2S_TX_MONO Configures whether to enable TX unit in mono mode.

0: Disable

1: Enable

(R/W)

Continued on the next page...

Register 49.10. I2S_TX_CONF_REG (0x0024)

Continued from the previous page...

I2S_TX_BIG_ENDIAN Configures I2S TX byte endian.
0: Low address with low address value
1: Low address value to high address
(R/W)

I2S_TX_UPDATE Configures whether to update I2S TX registers from APB clock domain to I2S TX clock domain.
0: No effect
1: Update
This bit will be cleared by hardware after update register done.
(R/W/SC)

I2S_TX_MONO_FST_VLD Configures the valid data channel in I2S TX mono mode.
0: The second channel data valid
1: The first channel data valid
(R/W)

I2S_TX_PCM_CONF Configures the I2S TX compress/decompress mode.
0 (atol): A-law decompress
1 (ltoa): A-law compress
2 (utol): μ-law decompress
3 (ltou): μ-law compress
(R/W)

I2S_TX_PCM_BYPASS Configures whether to bypass Compress/Decompress units for transmitted data.
0: No effect
1: Bypass
(R/W)

I2S_TX_MSB_SHIFT Configures the timing between the WS signal and the MSB of data.
0: Align at the rising edge
1: WS signal changes one BCK clock earlier
(R/W)

I2S_TX_BCK_NO_DLY Configures the source of the BCK rising and falling edges in master mode.
0: Rising and falling edges are constructed based on the BCK input from I2S*n*O_BCK_in
1: Rising and falling edges are constructed by dividing the clock of I2S*n* TX
(R/W)

I2S_TX_LEFT_ALIGN Configures I2S TX alignment mode.
0: Right alignment mode
1: Left alignment mode
(R/W)

Continued on the next page...

Register 49.10. I2S_TX_CONF_REG (0x0024)

Continued from the previous page...

I2S_TX_24_FILL_EN Configures the bit number that the 24 channel bits are stored to.
0: Store 24-bit channel data to 24 bits
1: Store 24-bit channel data to 32 bits (Extra bits are filled with zeros)
(R/W)

I2S_TX_WS_IDLE_POL Configures the relationship between WS and which channel data to transmit.
0: WS remains low when transmitting left channel data and high when transmitting right channel data
1: WS remains high when transmitting left channel data and low when transmitting right channel data
(R/W)

I2S_TX_BIT_ORDER Configures whether to reverse the bit order of valid data to be sent by the I2S TX.
0: Not reverse
1: Reverse
(R/W)

I2S_TX_TDM_EN Configures whether to enable I2S TDM TX mode.
0: Disable
1: Enable
(R/W)

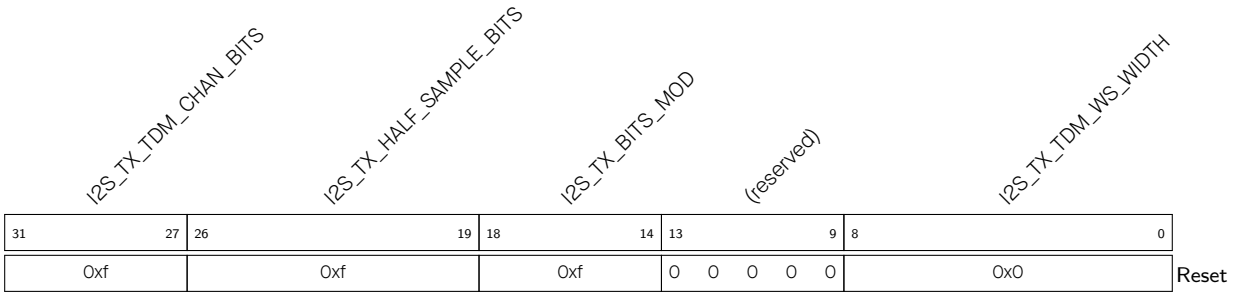
I2S_TX_PDM_EN Configures whether to enable I2S PDM TX mode.
0: Disable
1: Enable
(R/W)

I2S_TX_BCK_DIV_NUM Configures the divider of BCK in TX mode. Note this divider must not be configured to 1. (R/W)

I2S_TX_CHAN_MOD Configures I2S TX channel mode. For more information, see Table 49.9-4. (R/W)

I2S_SIG_LOOPBACK Configures whether to enable TX unit and RX unit sharing the same WS and BCK signals.
0: Disable
1: Enable
(R/W)

Register 49.11. I2S_TX_CONF1_REG (0x002C)



I2S_TX_TDM_WS_WIDTH Configures the width of I2S*n*O_WS_out (WS default level) at idle level in TDM mode. The width of I2S*n*O_WS_out at idle level in TDM mode = (I2S_TX_TDM_WS_WIDTH[8:0] +1) x T_BCK. (R/W)

I2S_TX_BITS_MOD Configures the valid data bit length of I2S TX channel.

7: All the valid channel data is in 8-bit mode

15: All the valid channel data is in 16-bit mode

23: All the valid channel data is in 24-bit mode

31: All the valid channel data is in 32-bit mode

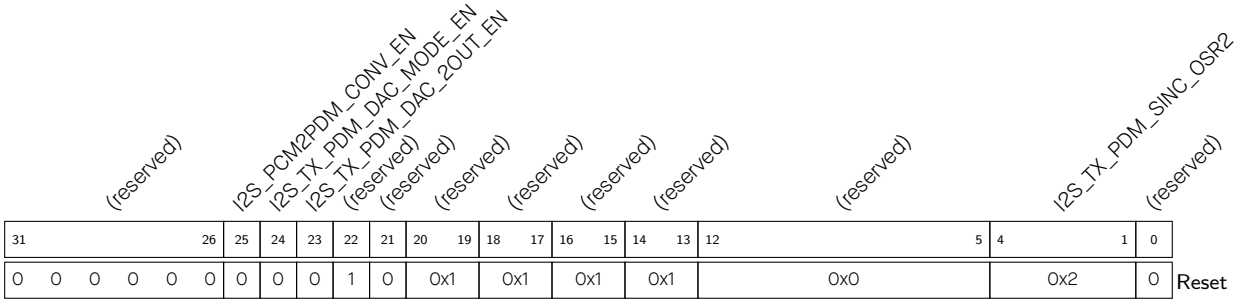
Other values are invalid.

(R/W)

I2S_TX_HALF_SAMPLE_BITS Configures I2S TX sample bits. BCK cycles in one WS period = I2S_TX_HALF_SAMPLE_BITS x 2. (R/W)

I2S_TX_TDM_CHAN_BITS Configures TX bit number for each channel in TDM mode. Bit number expected = I2S_TX_TDM_CHAN_BITS + 1. (R/W)

Register 49.12. I2S_TX_PCM2PDM_CONF_REG (for I2S*0* only) (0x0040)



I2S_TX_PDM_SINC_OSR2 Configures I2S TX PDM OSR value. (R/W)

I2S_TX_PDM_DAC_2OUT_EN Configures whether to enable I2S TX PDM DAC mode.

0: Enable 1-line DAC output mode

1: Enable 2-line DAC output mode

Only valid when I2S_TX_PDM_DAC_MODE_EN is set.

(R/W)

I2S_TX_PDM_DAC_MODE_EN Configures whether to enable 1-line PDM output mode or DAC output mode.

0: Enable 1-line PDM output mode

1: Enable DAC output mode

(R/W)

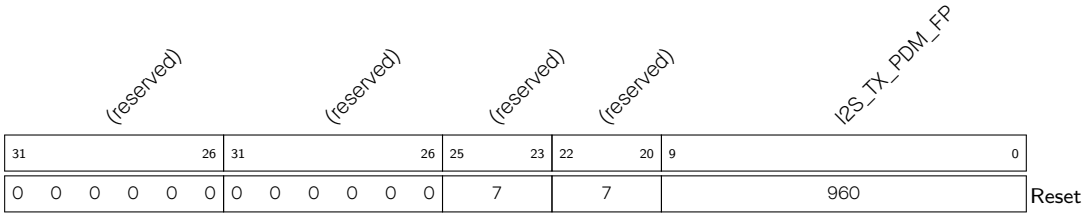
I2S_PCM2PDM_CONV_EN Configures whether to enable the I2S TX PCM-to-PDM converter.

0: Disable

1: Enable

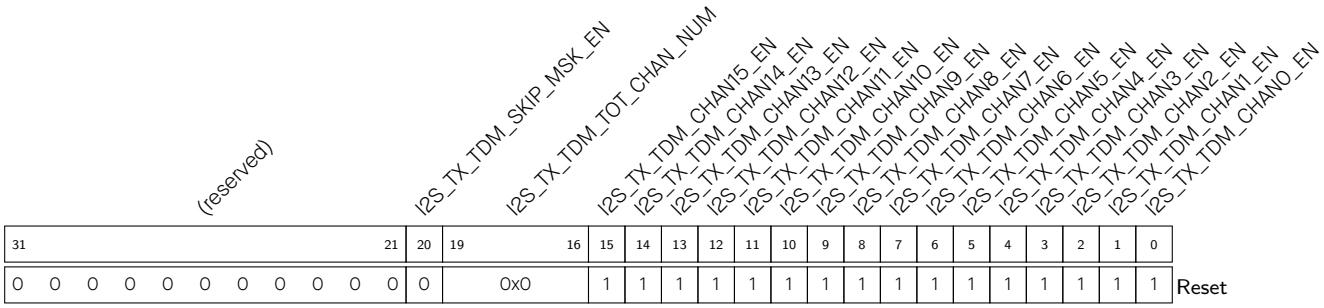
(R/W)

Register 49.13. I2S_TX_PCM2PDM_CONF1_REG (0x0044)



I2S_TX_PDM_FS Configures I2S PDM TX upsampling parameter. (R/W)

Register 49.14. I2S_TX_TDM_CTRL_REG (0x0054)



I2S_TX_TDM_CHAN*n*_EN (*n*: 0-15) Configures whether to enable the valid data output of I2S TX TDM channel *n*.

0: Channel TX data is controlled by I2S_TX_CHAN_EQUAL and I2S_SINGLE_DATA. See Section 49.9.2.1

1: Enable

(R/W)

I2S_TX_TDM_TOT_CHAN_NUM Configures the total number of channels in use in I2S TX TDM mode. Total channel number in use = I2S_TX_TDM_TOT_CHAN_NUM + 1. (R/W)

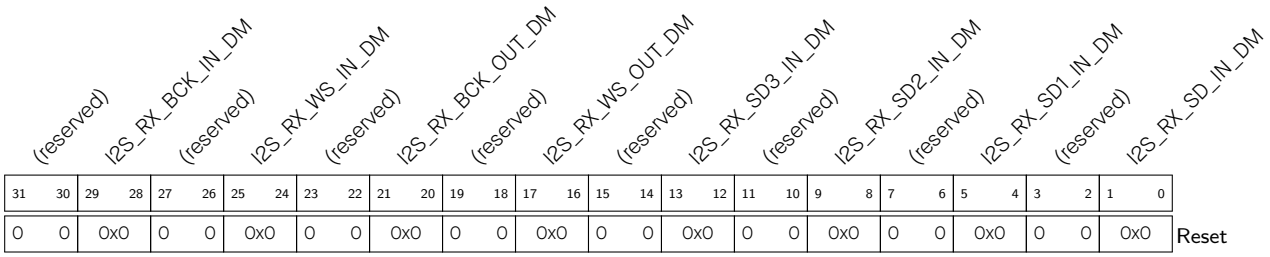
I2S_TX_TDM_SKIP_MSK_EN Configures the data to be sent in GDMA TX buffer.

0: Data stored in GDMA TX buffer is used by enabled channels and will not be read by channels that are not enabled.

1: Data stored in GDMA TX buffer is read by all channels and will be skipped by channels that are not enabled.

(R/W)

Register 49.15. I2S_RX_TIMING_REG (0x0058)



I2S_RX_SD_IN_DM Configures the delay mode of I2S RX SD input signal.

0: Bypass

1: Delay by rising edge

2: Delay by falling edge

3: Reserved

(R/W)

I2S_RX_SD1_IN_DM (for I2S*0* only) Configures the delay mode of I2S RX SD1 input signal. For detailed configuration values, please refer to I2S_RX_SD_IN_DM. (R/W)

I2S_RX_SD2_IN_DM (for I2S*0* only) Configures the delay mode of I2S RX SD2 input signal. For detailed configuration values, please refer to I2S_RX_SD_IN_DM. (R/W)

I2S_RX_SD3_IN_DM (for I2S*0* only) Configures the delay mode of I2S RX SD3 input signal.For detailed configuration values, please refer to I2S_RX_SD_IN_DM.(R/W)

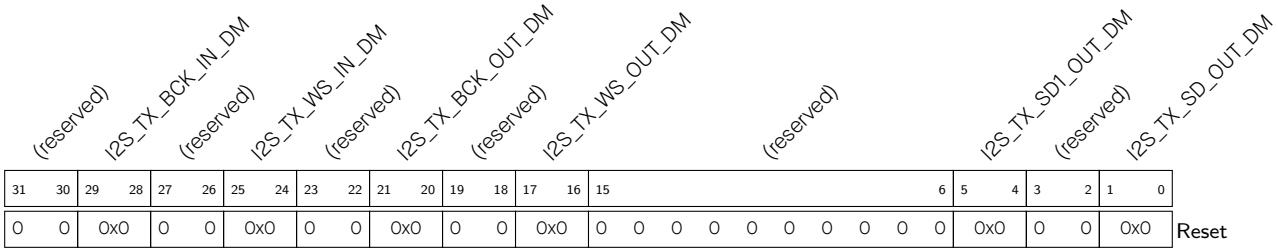
I2S_RX_WS_OUT_DM Configures the delay mode of I2S RX WS output signal. For detailed configuration values, please refer to I2S_RX_SD_IN_DM. (R/W)

I2S_RX_BCK_OUT_DM Configures the delay mode of I2S RX BCK output signal. For detailed configuration values, please refer to I2S_RX_SD_IN_DM. (R/W)

I2S_RX_WS_IN_DM Configures the delay mode of I2S RX WS input signal. For detailed configuration values, please refer to I2S_RX_SD_IN_DM. (R/W)

I2S_RX_BCK_IN_DM Configures the delay mode of I2S RX BCK input signal. For detailed configuration values, please refer to I2S_RX_SD_IN_DM. (R/W)

Register 49.16. I2S_TX_TIMING_REG (0x005C)



I2S_TX_SD_OUT_DM Configures the delay mode of I2S TX SD output signal.

0: Bypass

1: Delay by rising edge

2: Delay by falling edge

3: Reserved

(R/W)

I2S_TX_SD1_OUT_DM (for I2S*0* only) Configures the delay mode of I2S TX SD1 output signal. For detailed configuration values, please refer to I2S_TX_SD_OUT_DM. (R/W)

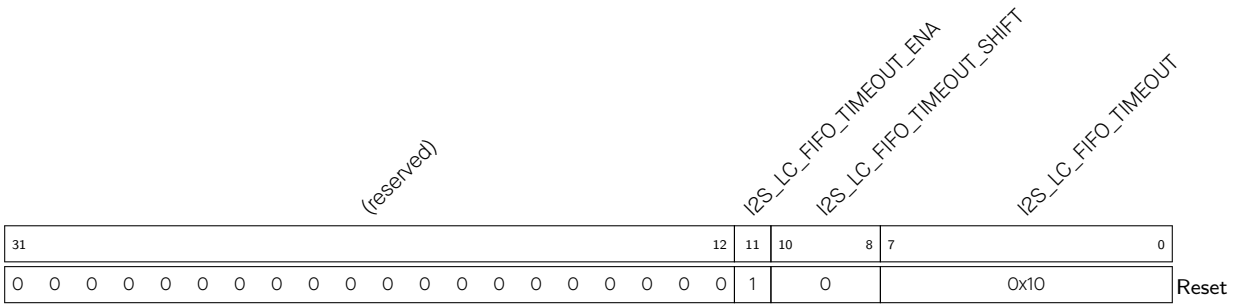
I2S_TX_WS_OUT_DM Configures the delay mode of I2S TX WS output signal. For detailed configuration values, please refer to I2S_TX_SD_OUT_DM. (R/W)

I2S_TX_BCK_OUT_DM Configures the delay mode of I2S TX BCK output signal. For detailed configuration values, please refer to I2S_TX_SD_OUT_DM. (R/W)

I2S_TX_WS_IN_DM Configures the delay mode of I2S TX WS input signal. For detailed configuration values, please refer to I2S_TX_SD_OUT_DM. (R/W)

I2S_TX_BCK_IN_DM Configures the delay mode of I2S TX BCK input signal. For detailed configuration values, please refer to I2S_TX_SD_OUT_DM. (R/W)

Register 49.17. I2S_LC_HUNG_CONF_REG (0x0060)



I2S_LC_FIFO_TIMEOUT Configures FIFO timeout threshold. The FIFO hung counter is incremented by one each time I2S is in an error state and the tick counter reaches its threshold. I2S_TX_HUNG_INT or I2S_RX_HUNG_INT interrupt will be triggered when FIFO hung counter is equal to the FIFO timeout threshold. (R/W)

I2S_LC_FIFO_TIMEOUT_SHIFT Configures tick counter threshold. The tick counter is incremented by one each time I2S is in an error state and it is on the rising edge in each system clock (APB). The tick counter is reset when counter value >= $88000/2^{I2S_LC_FIFO_TIMEOUT_SHIFT}$. (R/W)

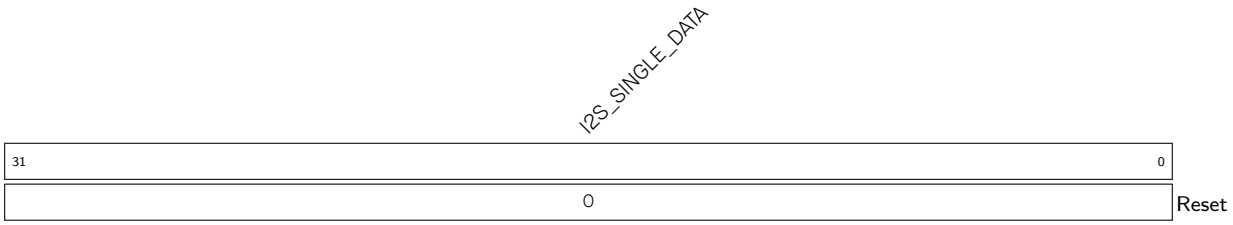
I2S_LC_FIFO_TIMEOUT_ENA Configures whether to enable FIFO timeout.

0: Disable

1: Enable

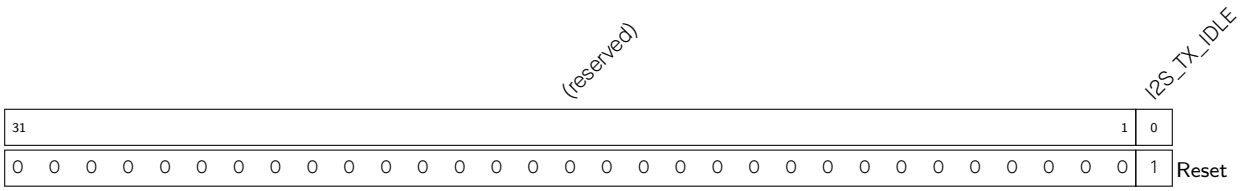
(R/W)

Register 49.18. I2S_CONF_SIGLE_DATA_REG (0x0068)



I2S_SINGLE_DATA Configures constant channel data to be sent out. (R/W)

Register 49.19. I2S_STATE_REG (0x006C)



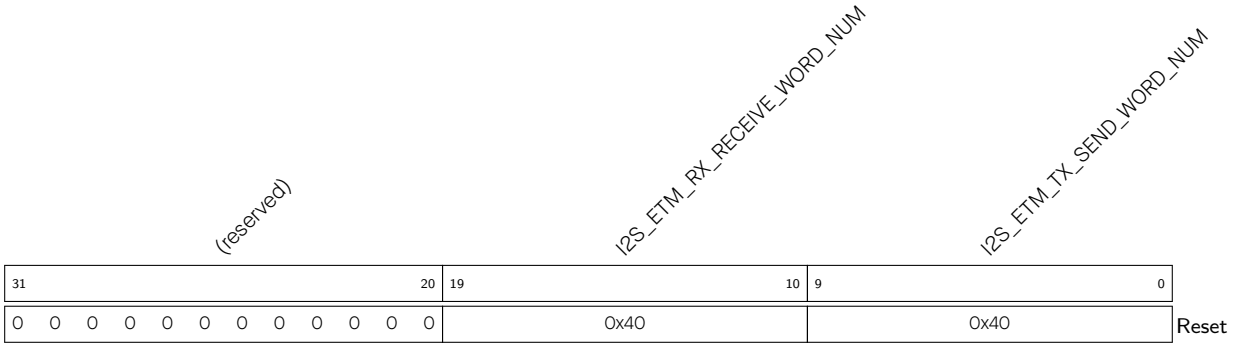
I2S_TX_IDLE Represents the TX unit state.

0: I2S TX unit is working

1: I2S TX unit is in idle state

(RO)

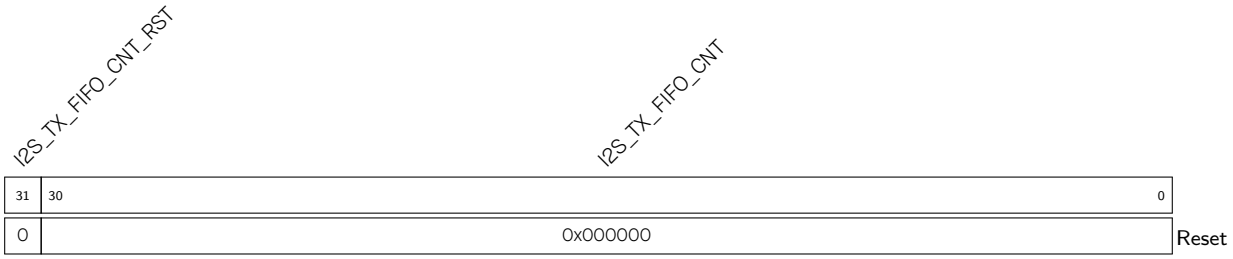
Register 49.20. I2S_ETM_CONF_REG (0x0070)



I2S_ETM_TX_SEND_WORD_NUM Configures the threshold of triggering ETM I2S_TX_X_WORDS_SENT event. When transmitting word number of I2S_ETM_TX_SEND_WORD_NUM [9:0], I2S will trigger the corresponding ETM event. (R/W)

I2S_ETM_RX_RECEIVE_WORD_NUM Configures the threshold of triggering ETM I2S_RX_X_WORDS_RECEIVED event. When receiving word number of I2S_ETM_RX_RECEIVE_WORD_NUM [9:0], I2S will trigger the corresponding ETM event. (R/W)

Register 49.21. I2S_FIFO_CNT_REG (0x0074)



I2S_TX_FIFO_CNT Configures the TX FIFO counter value. (RO)

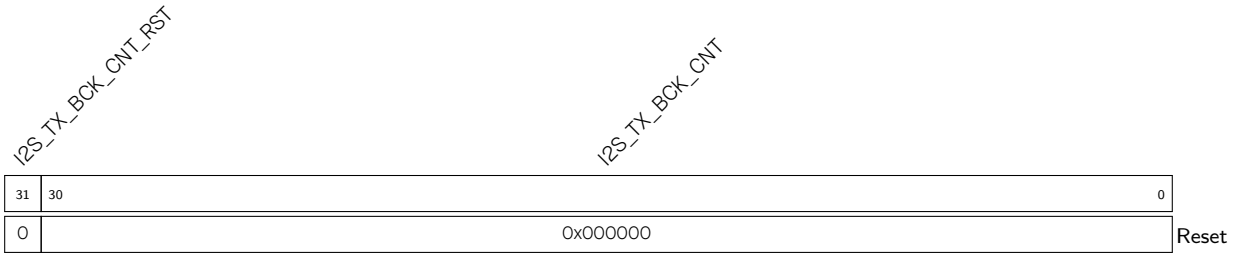
I2S_TX_FIFO_CNT_RST Configure whether to reset the TX FIFO counter.

0: No effect

1: Reset

(WT)

Register 49.22. I2S_BCK_CNT_REG (0x0078)



I2S_TX_BCK_CNT Configures the TX BCK counter value. (RO)

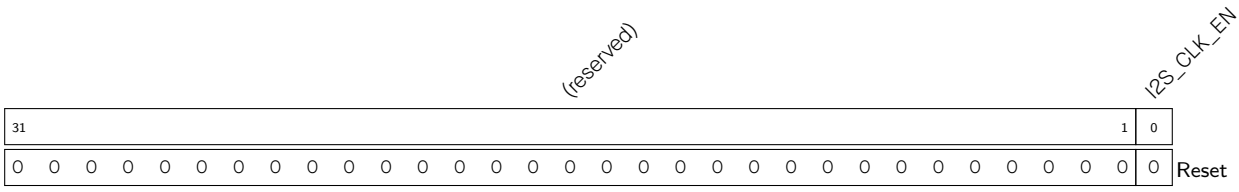
I2S_TX_BCK_CNT_RST Configures whether to reset TC BCK counter.

0: No effect

1: Reset

(WT)

Register 49.23. I2S_CLK_GATE_REG (0x007C)



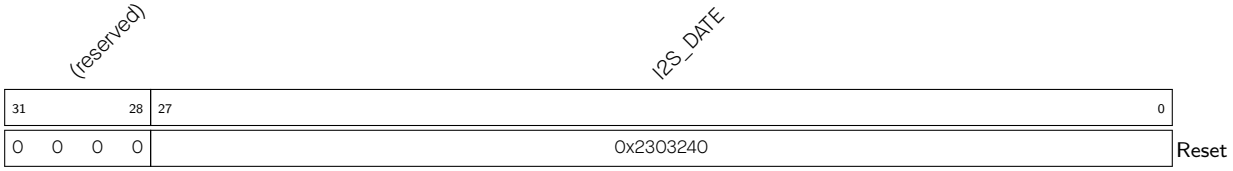
I2S_CLK_EN Configures whether to enable clock gate.

0: Disable

1: Enable

(R/W)

Register 49.24. I2S_DATE_REG (0x0080)



I2S_DATE Version control register. (R/W)

Chapter 50

LP I2S Controller

50.1 Introduction

ESP32-P4 has a built-in LP I2S interface, which provides a data reception communication interface for Voice Activity Detection (VAD) and some digital audio applications in low power mode.

The I2S standard bus defines three signals, namely, a bit clock signal (BCK), a channel/word select signal (WS), and a serial data signal (SD). A basic I2S data bus has one master and one slave. The roles remain unchanged throughout the communication.

The LP I2S module on ESP32-P4 provides an independent RX unit, which supports receiving data when the chip is running with the lowest power consumption. Compared to HP I2S, i.e., I2S0, I2S1, and I2S2, LP I2S does not support DMA access. Instead, it uses a separate internal memory to store data.

Note:
For I2S-related terminology, refer to the Section 49.2 *Terminology* in Chapter 49 *I2S Controller (I2S)*.

50.2 Feature List

The LP I2S module has the following features:

- RX master mode and slave mode
- A variety of audio standards supported:
  - TDM Philips standard
  - TDM MSB alignment standard
  - TDM PCM standard
  - PDM standard
- Various RX modes supported:
  - TDM RX mode, up to 2 channels supported
  - PDM RX mode
    * Raw PDM data reception
    * PDM-to-PCM data format conversion, up to 2 channels supported
- Configurable sample clock with a variety of sampling frequencies supported
- 16-bit data communication

- Standard LP I2S interface interrupts

50.3 Architectural Overview

Figure 50.3-1 shows the structure of the ESP32-P4 LP I2S module, consisting of:

- Receive control unit (RX Unit)
- Input and output timing unit (I/O Sync)
- Clock divider (Clock Generator)
- 64 x 32-bit RX FIFO
- 256 x 32-bit LP I2S Memory
- Communication interface for Voice Activity Detection (VAD)

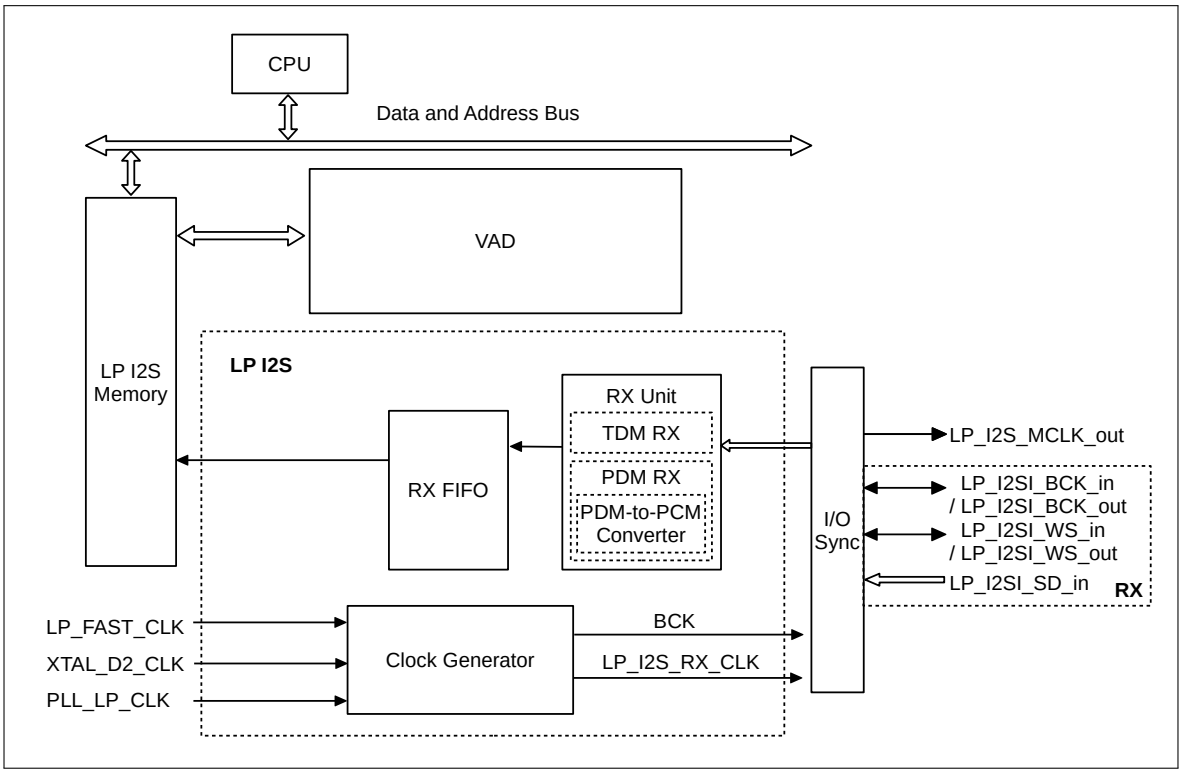


Figure 50.3-1. ESP32-P4 LP I2S Architecture

The RX unit have a three-line interface that uses a bit clock line (BCK), a word select line (WS), and a serial data line (SD). The SD line of the RX unit is dedicated for data input. BCK and WS signal lines for the RX unit can be configured as master output mode or slave input mode.

The signal bus of the LP I2S module is shown at the right part of Figure 50.3-1. The naming of these signals in RX units follows the pattern of LP_I2SA_B_C, for example, LP_I2SI_BCK_in.

- "A" is fixed to "I", indicating that signals are input to the RX unit
- "B" represents the signal function, which includes:
  - BCK

  - WS
  - SD
- "C" represents the signal direction, which includes:
  - "in": Input signal into the LP I2S module
  - "out": Output signal from the LP I2S module

Table 50.3-1 provides a detailed description of LP I2S signals.

Table 50.3-1. LP I2S Signal Description

Signal *	Direction	Function
LP_I2SI_BCK_in	Input	In LP I2S slave mode, inputs BCK signal for RX unit
LP_I2SI_BCK_out	Output	In LP I2S master mode, outputs BCK signal for RX unit
LP_I2SI_WS_in	Input	In LP I2S slave mode, inputs WS signal for RX unit
LP_I2SI_WS_out	Output	In LP I2S master mode, outputs WS signal for RX unit
LP_I2SI_SD_in	Input	Serial input data bus for LP I2S RX unit
LP_I2S_MCLK_out	Output	LP I2S master clock output

* Any required signals of LP I2S must be mapped to the chip's pins via LP GPIO matrix. See Chapter 10 *GPIO Matrix and IO MUX*.

50.4 Supported Audio Standards

ESP32-P4 I2S supports multiple audio standards, including TDM Philips standard, TDM MSB alignment standard, TDM PCM standard, and PDM standard.

Select the needed standard by configuring the following bits:

- LP_I2S_RX_TDM_EN
  - 0: Disable TDM mode
  - 1: Enable TDM mode
- LP_I2S_RX_PDM_EN
  - 0: Disable PDM mode
  - 1: Enable PDM mode
- LP_I2S_RX_MSB_SHIFT
  - 0: WS and SD signals change simultaneously, i.e., enable MSB alignment standard
  - 1: WS signal changes one BCK clock cycle earlier than SD signal, i.e., enable Philips standard or select PCM standard

Note:
LP_I2S_RX_TDM_EN and LP_I2S_RX_PDM_EN must not be configured to 1 or 0 at the same time, otherwise LP I2S will transmit data incorrectly in a mode that is neither TDM nor PDM.

50.4.1 TDM Philips Standard

Philips standards require that WS signal changes one BCK clock cycle earlier than SD signal on BCK falling edge, which means that WS signal is valid from one clock cycle before transmitting the first bit of channel data and changes one clock before the end of channel data transfer. SD signal line transmits the most significant bit of audio data first.

Compared with the basic Philips standard, TDM Philips standard supports multiple channels. See Figure 50.4-1.

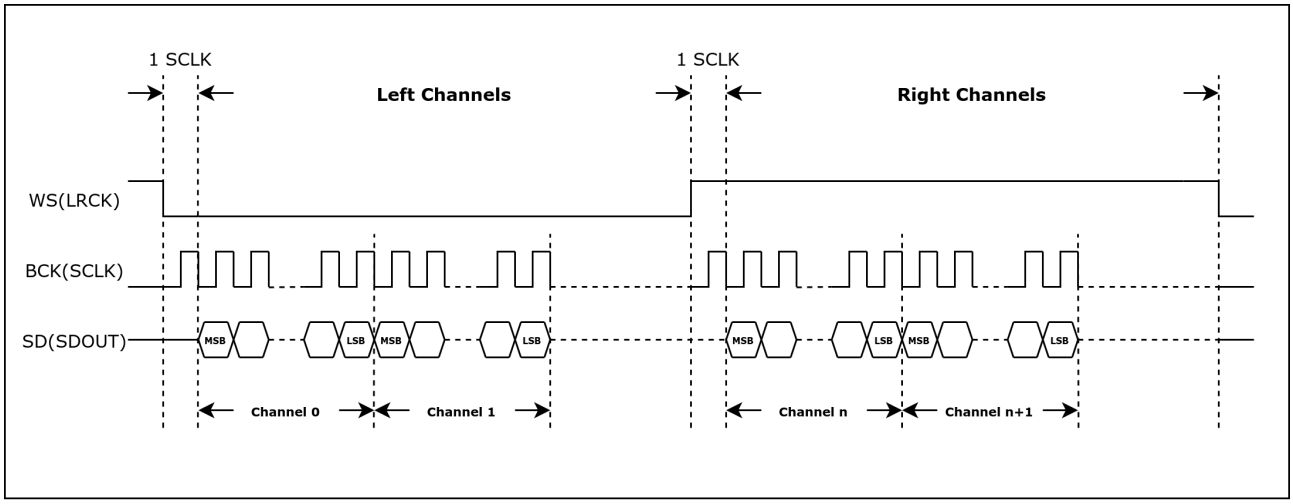


Figure 50.4-1. TDM Philips Standard Timing Diagram

50.4.2 TDM MSB Alignment Standard

MSB alignment standards require that WS and SD signals change simultaneously on the falling edge of BCK. The WS signal is valid until the end of the channel data transfer. The SD signal line transmits the most significant bit of audio data first.

Compared with the basic MSB alignment standard, TDM MSB alignment standard supports multiple channels. See Figure 50.4-2.

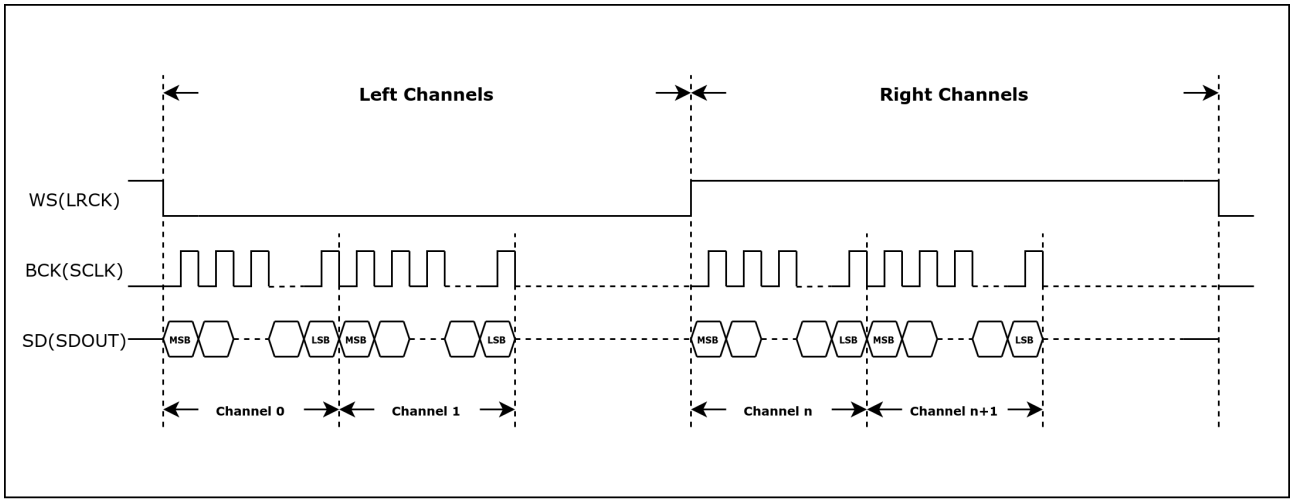


Figure 50.4-2. TDM MSB Alignment Standard Timing Diagram

50.4.3 TDM PCM Standard

Short frame synchronization under the PCM standards requires that the WS signal changes one BCK clock cycle earlier than the SD signal on the falling edge of BCK, which means that the WS signal becomes valid one clock cycle before transferring the first bit of channel data and remains unchanged in this BCK clock cycle. SD signal line first transmits the most significant bit of audio data.

Compared with the basic PCM standard, TDM PCM standard supports multiple channels. See Figure 50.4-3.

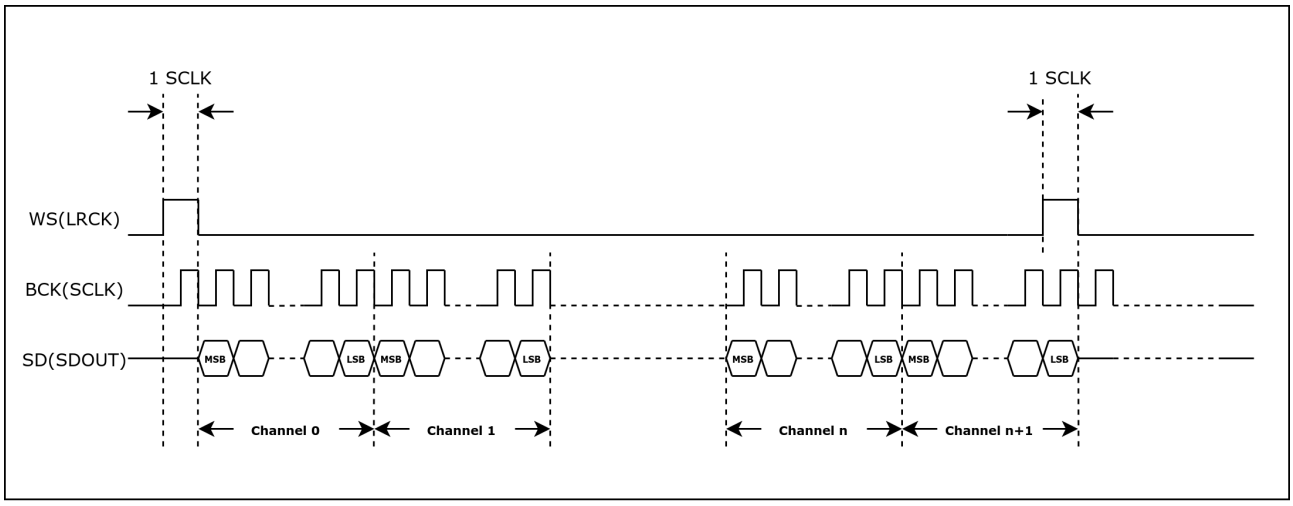


Figure 50.4-3. TDM PCM Standard Timing Diagram

50.4.4 PDM Standard

Under PDM standard, WS signal changes continuously during data transmission. The low-level and high-level of this signal indicates the left channel and right channel respectively. WS and SD signals change simultaneously on the falling edge of BCK. See Figure 50.4-4.

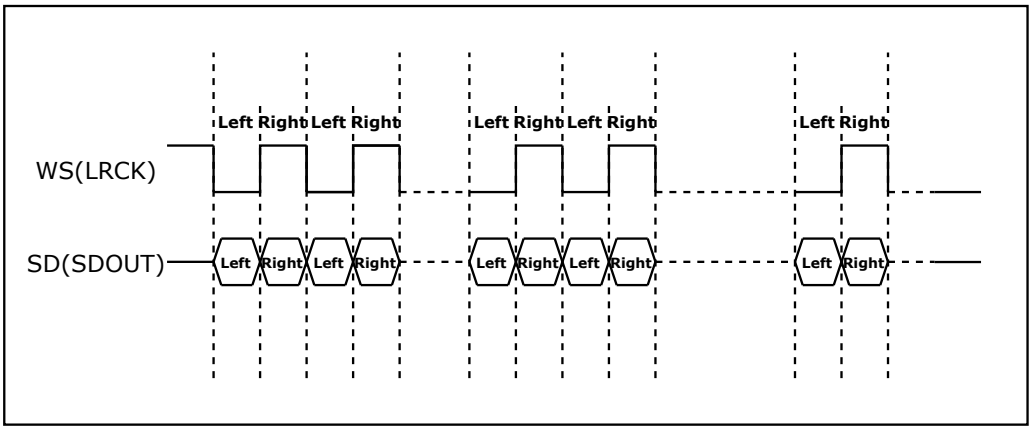


Figure 50.4-4. PDM Standard Timing Diagram

50.5 RX Clock

LP_I2S_RX_CLK is the master clock of LP I2S RX unit, divided from:

- Up to 40 MHz LP_FAST_CLK with configurable clock source
- 20 MHz XTAL_D2_CLK
- 8 MHz PLL_LP_CLK

The serial clock (BCK) of the LP I2S RX unit is divided from LP_I2S_RX_CLK, as shown in Figure 50.5-1. LPPERI_LP_I2S_RX_CLK_SEL is used to select clock source for the LP I2S RX unit, and LPPERI_CK_EN_LP_I2S_RX to enable or disable the clock source.

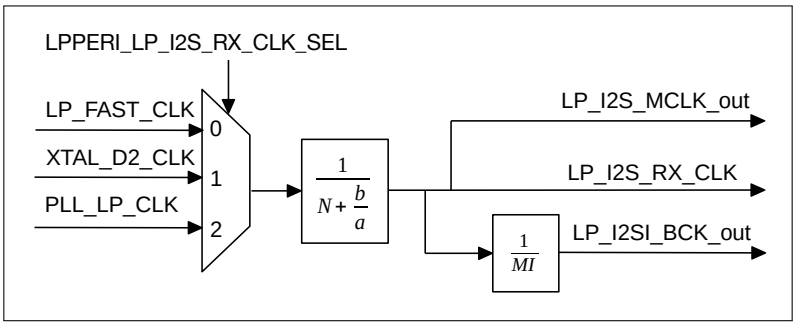


Figure 50.5-1. LP I2S Clock Generator

The following formula shows the relation between LP_I2S_RX_CLK frequency $f_{\text{LP_I2S_RX_CLK}}$ and the divider clock source frequency $f_{\text{LP_I2S_CLK_S}}$:

$$f_{\text{LP_I2S_RX_CLK}} = \frac{f_{\text{LP_I2S_CLK_S}}}{\text{N} + \frac{\text{b}}{\text{a}}}$$

N is an integer value between 2 and 256. The value of N is mapped to that of LPPERI_LP_I2S_RX_CLKM_DIV_N as follows:

- When LPPERI_LP_I2S_RX_CLKM_DIV_N = 0, N = 256;
- When LPPERI_LP_I2S_RX_CLKM_DIV_N = 1, N = 2;
- When LPPERI_LP_I2S_RX_CLKM_DIV_N has any other value, N = LPPERI_LP_I2S_RX_CLKM_DIV_NUM.

The values of "a" and "b" in fractional divider depend only on x, y, z, and yn1. The corresponding formulas are as follows:

- When b $<= \frac{a}{2}$, yn1 $= 0$, x $= floor([\frac{a}{b}]) - 1$, y $=$ a%b, z $=$ b;
- When b $> \frac{a}{2}$, yn1 $= 1$, x $= floor([\frac{a}{a-b}]) - 1$, y $=$ a%(a - b), z $=$ a - b.

The values of x, y, z, and yn1 are configured by LPPERI_LP_I2S_RX_CLKM_DIV_X, LPPERI_LP_I2S_RX_CLKM_DIV_Y, LPPERI_LP_I2S_RX_CLKM_DIV_Z and LPPERI_LP_I2S_RX_CLKM_DIV_YN1.

To configure the integer divider, clear LPPERI_LP_I2S_RX_CLKM_DIV_X and LPPERI_LP_I2S_RX_CLKM_DIV_Z , then set LPPERI_LP_I2S_RX_CLKM_DIV_Y to 1.

Note:
Using fractional divider may introduce some clock jitter.

In master RX mode, the serial clock BCK for LP I2S RX unit is LP_I2SI_BCK_out divided from LP_I2S_RX_CLK, which is:

$$f_{\text{LP_I2SI_BCK_out}} = \frac{f_{\text{LP_I2S_RX_CLK}}}{\text{MI}}$$

"MI" is an integer value:

$$\text{MI} = \text{LP_I2S_RX_BCK_DIV_NUM} + 1$$

Note:
LP_I2S_RX_BCK_DIV_NUM must not be configured as 1.

In LP I2S slave mode, make sure $f_{\text{LP_I2S_RX_CLK}}$ >= 8 x f_{BCK} to ensure the sampling accuracy. The LP I2S module can output LP_I2S_MCLK_out as the master clock for peripherals.

Since in slave mode, the module clock frequency must be greater than or equal to 8 times the BCK clock frequency, the maximum sampling frequency of LP I2S is limited by the data bit width and the number of channels. For example, since the clock source frequency is up to 40 MHz, the module clock can be configured up to 20 MHz, and the BCK clock can be configured up to 5 MHz. Therefore, when transmitting dual-channel 16-bit width data, LP I2S supports sample frequencies up to 78.125 kHz, e.g., 8 kHz, 16 kHz, 32 kHz, 44.1 kHz, 48 kHz. For detailed information, please refer to Section 50.5 *RX Clock*.

In addition to the clock frequency limitation, LP I2S uses LP I2S internal memory for data storage instead of DMA. Therefore, users need to read data from the LP I2S internal memory in a timely manner. The reading rate may limit the sample frequencies that LP I2S supports. For detailed information, please refer to Section 50.8.3 *Internal Memory*.

50.6 Reset

The units and FIFOs in the LP I2S module are reset by the following bits.

- LP I2S RX units: Reset by the bit LP_I2S_RX_RESET;
- LP I2S RX FIFO: Reset by the bits LP_I2S_RX_FIFO_RESET.

Note:
The LP I2S module clock must be configured first before the units and FIFO are reset.

50.7 Master/Slave RX Mode

The LP I2S supports receiving data either as a master or as a slave in RX mode.

- LP I2S works as a master receiver:
  - Set LP_I2S_RX_START to start receiving data. When this bit is set, the RX unit keeps outputting clock signal and sampling input data.
  - Configure LP_I2S_RX_STOP_MODE to control the suspension of data reception:
    * 0: The RX unit only suspends data reception when LP_I2S_RX_START is cleared.
    * 1: The RX unit suspends data reception when LP_I2S_RX_START is cleared or the number of received bytes is greater than the value configured in LP_I2S_RX_EOF_NUM_REG. After data

reception is suspended, if LP_I2S_RX_START is not cleared, data reception can be restarted by setting LP_I2S_RX_RESET to 1.

- RX unit stops receiving data when the bit LP_I2S_RX_START is cleared.

- LP I2S works as a slave receiver:
  - Set LP_I2S_RX_START. Wait for the master BCK signal to start receiving data.
  - Configure LP_I2S_RX_STOP_MODE to control data reception suspension:
    * 0: The RX unit only suspends data reception when LP_I2S_RX_START is cleared.
    * 1: The RX unit suspends data reception when LP_I2S_RX_START is cleared or the number of received bytes is greater than the value configured in LP_I2S_RX_EOF_NUM_REG. After data reception is suspended, if LP_I2S_RX_START is not cleared, data reception can be restarted by setting LP_I2S_RX_RESET to 1.
  - The RX unit stops receiving data when the bit LP_I2S_RX_START is cleared.

50.8 Receiving Data

In RX mode, LP I2S first reads data from the peripheral interface and then stores the data in the LP I2S memory according to the configured channel mode and data mode.

50.8.1 Channel Mode Control

ESP32-P4 LP I2S supports both TDM RX mode and PDM RX mode. Set LP_I2S_RX_TDM_EN to enable TDM RX mode, or set LP_I2S_RX_PDM_EN to enable PDM RX mode.

Note:
LP_I2S_RX_TDM_EN and LP_I2S_RX_PDM_EN must not be cleared or set simultaneously.

50.8.1.1 TDM RX Mode

In TDM RX mode, LP I2S supports up to 2 channels to input data. The total number of RX channels in use is controlled by LP_I2S_RX_TDM_TOT_CHAN_NUM. For example, if LP_I2S_RX_TDM_TOT_CHAN_NUM is set to 1, channel 0 ~ 1 will be used to receive data.

In these RX channels, if LP_I2S_RX_TDM_PDM_CHANn_EN is set to:

- 0: The channel data is invalid and will not be stored into RX FIFO;
- 1: The channel data is valid and will be stored into RX FIFO.

In TDM master mode, WS signal is controlled by LP_I2S_RX_WS_IDLE_POL and LP_I2S_RX_TDM_WS_WIDTH.

- LP_I2S_RX_WS_IDLE_POL: The default level of WS signal;
- LP_I2S_RX_TDM_WS_WIDTH: The cycles the WS default level lasts for when receiving all channel data.

LP_I2S_RX_HALF_SAMPLE_BITS x 2 is equal to the BCK cycles in one WS period.

50.8.1.2 PDM RX Mode

In PDM RX mode, LP I2S converts the serial data from channels to the data to be entered into memory. LP I2S supports both PDM raw data reception and PDM-to-PCM data format conversion.

In PDM RX master mode, the default level of the WS signal is controlled by LP_I2S_RX_WS_IDLE_POL. WS frequency is half of the BCK frequency. The configuration of the BCK signal is similar to that of WS signal as described in Section 50.5. Note, in PDM RX mode, the value of LP_I2S_RX_HALF_SAMPLE_BITS must be same as that of LP_I2S_RX_BITS_MOD.

When the PDM-to-PCM converter is enabled for LP I2S, the received PDM data is converted to PCM data and controlled according to the data mode. Configure LP_I2S_RX_PDM2PCM_EN to enable this converter. The register configuration for PDM-to-PCM converter is as follows:

- Configure sampling frequency and downsampling rate.
  When LP I2S PDM-to-PCM converter is enabled, PDM clock frequency is:
  - in master mode: PDM clock frequency is equal to BCK frequency.
  - in slave mode: PDM clock is provided by external device.

  The sampling frequency ($f_{Sampling}$) is related to PDM clock frequency as follows:

$$f_{Sampling} = \frac{f_{PDM}}{DSR}$$

  Downsampling rate (DSR) is related to LP_I2S_RX_PDM_SINC_DSR_16_EN as follows:

$$DSR = LP_I2S_RX_PDM_SINC_DSR_16_EN \times 64$$

  Configure the registers according to needed master/slave mode, sampling frequency, and downsampling rate.
- Configure valid channels.
  When the PDM-to-PCM converter is enabled, input signals from eight channels are supported at most. See Table 50.8-1 for the register configuration and related channels.

Table 50.8-1. PDM-to-PCM Data Input

Input Data Signal	Channel	Enable Register
LP_I2SI_SD_in	Left channel	LP_I2S_RX_TDM_PDM_CHAN0_EN
	Right channel	LP_I2S_RX_TDM_PDM_CHAN1_EN

50.8.2 Data Format Control

The data format of LP I2S is controlled in the following phases:

- Phase I: Serial input data is converted into the data to be saved to RX FIFO;
- Phase II: The data is read from RX FIFO and converted according to the input data mode.

50.8.2.1 Bit Order Control of Channel Data

The channel data will be stored as the data to be input in order from high to low. The data bit order in each channel is controlled by LP_I2S_RX_BIT_ORDER:

- 0: The bit order of the data to be input is not reversed;
- 1: The bit order of the data to be input is reversed.

At this point, the first phase of data format control is completed.

50.8.2.2 Bit Width Control of Channel RX Data

The storage data width in each channel is controlled by LP_I2S_RX_BITS_MOD. For ESP32-P4 LP I2S, the value of LP_I2S_RX_BITS_MOD is fixed as 15. The corresponding channel storage data width is 16 bit.

50.8.2.3 Bit Width Control of Channel RX Data

The RX data width in each channel is determined by LP_I2S_RX_TDM_CHAN_BITS.

- If the storage data width in each channel is smaller than the received (RX) data width, then only the bits within the storage data width is saved into memory. Configure LP_I2S_RX_LEFT_ALIGN to:
  - 0: Only the lower bits of the received data within the storage data width is stored to memory;
  - 1: Only the higher bits of the received data within the storage data width is stored to memory.
- If the received data width is smaller than the storage data width in each channel, the higher bits of the received data will be filled with zeros and then the data is saved to memory.

50.8.2.4 Endian Control of Channel Storage Data

The received data is then converted into storage data (to be stored to memory) after some processing, such as discarding extra bits or filling zeros in missing bits. The endian of the storage data is controlled by LP_I2S_RX_BIG_ENDIAN (the data bit width is fixed as 16). See the table below.

Table 50.8-2. Channel Storage Data Endian

Original Data	Endian of Processed Data	LP_I2S_RX_BIG_ENDIAN
{B1, B0}	{B1, B0}	0
	{B0, B1}	1

At this point, the data format control is completed. Data then is stored into memory.

50.8.3 Internal Memory

All data after format control is stored in the LP I2S internal memory, which is a 16-bit wide circular buffer that supports RX unit write operations and system read operations. The LP I2S internal memory includes a pair of hardware-maintained read and write pointers to manage write and read operations.

1. When LP I2S writes data:
   - If the memory is full: Overwrites the oldest data and increments the write and read pointers by 1 (the value of the read pointer is the value of the write pointer plus one at this point).
   - If the memory is not full: Writes the data and increments the write pointer by 1. The read pointer remains unchanged.
2. When the system reads from the LP I2S memory:

- If the memory is empty: The read and write pointers remain unchanged (the value of the read pointer equals to the value of the write pointer at this point).
- If the memory is not empty: Increments the read pointer by 1. The write pointer remains unchanged.

Note:

- The incremental operation for the read and write pointers is in a circular manner, i.e., when the value of the read or write pointer reaches the boundary of the memory, the incremental operation resets the value of the pointer to zero.
- Within the LP I2S internal memory architecture, the system always reads the oldest data in the memory.

50.9 Interrupts

ESP32-P4's LP I2S can generate the LP_I2S_INTR interrupt signal that will be sent to the Interrupt Matrix. There are several internal interrupt sources from LP I2S that can generate the above interrupt signals as follows:

- LP_I2S_RX_HUNG_INT: Triggered when the data reception is timed out. For example, if the LP I2S module is configured as RX slave mode, but the master does not transmit data for a long time (specified in LP_I2S_LC_HUNG_CONF_REG), this interrupt will be triggered.
- LP_I2S_RX_DONE_INT: Triggered when the data reception is completed.
- LP_I2S_RX_FIFOMEM_UDF_INT: Triggered when the LP I2S memory is read empty.
- LP_I2S_RX_MEM_THRESHOLD_INT: Triggered when the data in LP I2S memory is larger than the value configured in LP_I2S_RX_MEM_THRESHOLD.

Note:
For definitions of *interrupt*, *interrupt signal*, *interrupt source*, and their correlations, please refer to Chapter 13 *Interrupt Matrix* > Section 13.2 *Interrupt Terminology in ESP32-P4*.

Each interrupt source can be configured by a common set of registers that are described in Section *Interrupt Configuration Registers*. The specific registers can be found in Section 50.10 *Register Summary*.

50.10 Register Summary

The addresses in this section are relative to LP I2S base address provided in Table 9.3-2 in Chapter 9 *System and Memory*.

The abbreviations given in Column **Access** are explained in Section *Access Types for Registers*.

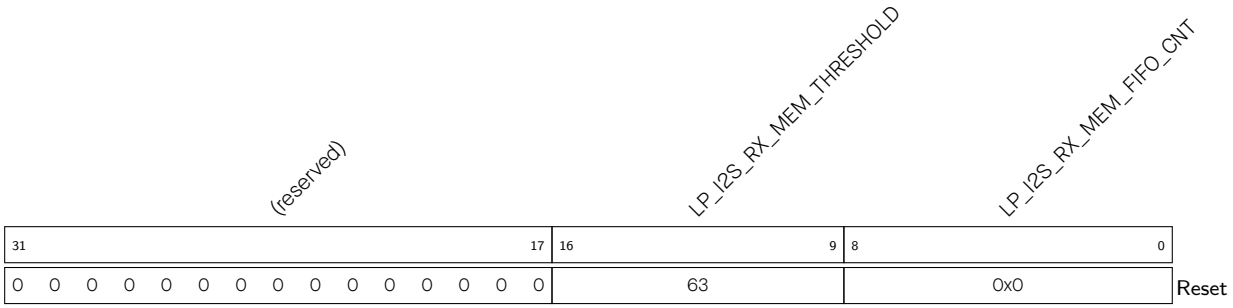
Name	Description	Address	Access
RX control and configuration registers			
LP_I2S_RX_MEM_CONF_REG	LP I2S memory configuration register	0x0008	varies
LP_I2S_RX_CONF_REG	LP I2S RX configuration register	0x0020	varies
LP_I2S_RX_CONF1_REG	LP I2S RX configuration register 1	0x0028	R/W
LP_I2S_RX_TDM_CTRL_REG	LP I2S RX TDM mode configuration register	0x0050	R/W
LP_I2S_RXEOF_NUM_REG	LP I2S RX data number control register	0x0064	R/W
LP_I2S_RX_PDM_CONF_REG	LP I2S RX PDM mode configuration register	0x0070	R/W
Interrupt registers			
LP_I2S_INT_RAW_REG	LP I2S interrupt raw register	0x000C	RO/WTC/SS
LP_I2S_INT_ST_REG	LP I2S interrupt status register	0x0010	RO
LP_I2S_INT_ENA_REG	LP I2S interrupt enable register	0x0014	R/W
LP_I2S_INT_CLR_REG	LP I2S interrupt clear register	0x0018	WT
RX clock and timing register			
LP_I2S_RX_TIMING_REG	LP I2S RX timing control register	0x0058	R/W
Control and configuration registers			
LP_I2S_LC_HUNG_CONF_REG	LP I2S timeout configuration register	0x0060	R/W
LP_I2S_CONF_SIGLE_DATA_REG	LP I2S single data register	0x0068	R/W
Clock register			
LP_I2S_CLK_GATE_REG	Clock gate register	0x00F8	R/W
Version register			
LP_I2S_DATE_REG	Version control register	0x00FC	R/W

50.11 Registers

The addresses in this section are relative to LP I2S base address provided in Table 9.3-2 in Chapter 9 *System and Memory*.

For how to program reserved fields, please refer to Section *Programming Reserved Register Field*.

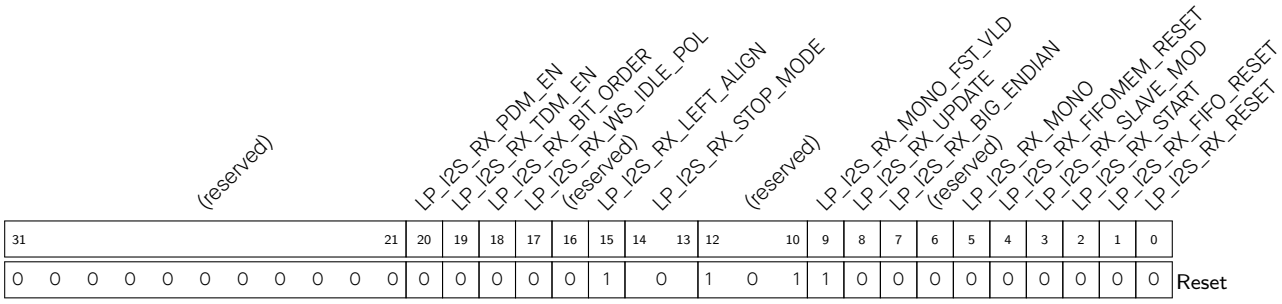
Register 50.1. LP_I2S_RX_MEM_CONF_REG (0x0008)



LP_I2S_RX_MEM_FIFO_CNT Configures the number of data in the RX memory. (RO)

LP_I2S_RX_MEM_THRESHOLD Configures the RX memory threshold. LP I2S RX memory triggers an interrupt when the data in the memory exceeds (not including equals) this threshold. (R/W)

Register 50.2. LP_I2S_RX_CONF_REG (0x0020)



LP_I2S_RX_RESET Configures whether to reset RX.

0: No effect

1: Reset

(WT)

LP_I2S_RX_FIFO_RESET Configures whether to reset RX FIFO.

0: No effect

1: Reset

(WT)

LP_I2S_RX_START Configures whether to start receiving data.

0: No effect

1: Start

(R/W)

Continued on the next page...

Register 50.2. LP_I2S_RX_CONF_REG (0x0020)

Continued from the previous page...

LP_I2S_RX_SLAVE_MOD Configures whether to enable slave RX mode.

0: Disable

1: Enable

(R/W)

LP_I2S_RX_FIFOMEM_RESET Configures whether to reset RX Sync FIFO memory.

0: No effect

1: Reset

(WT)

LP_I2S_RX_MONO Configures whether to enable RX unit in mono mode.

0: Disable

1: Enable

(R/W)

LP_I2S_RX_BIG_ENDIAN Configures LP I2S RX byte endian.

0: Low address data is saved to low address

1: Low address data is saved to high address

(R/W)

LP_I2S_RX_UPDATE Configures whether to update LP I2S RX registers from APB clock domain to LP I2S RX clock domain. This bit will be cleared by hardware after the register update is done.

0: No effect

1: Update

(R/W/SC)

LP_I2S_RX_MONO_FST_VLD Configures which channel data value is valid in LP I2S RX mono mode.

0: The second channel data value is valid

1: The first channel data value is valid

(R/W)

LP_I2S_RX_STOP_MODE Configures when LP I2S RX stops data reception.

0: Only stops when LP_I2S_RX_START is cleared

1: Stops when LP_I2S_RX_START is cleared or the number of received bytes is greater than the value configured in LP_I2S_RX_EOF_NUM_REG

Other values: Invalid

(R/W)

LP_I2S_RX_LEFT_ALIGN Configures the RX alignment mode

0: Right alignment

1: Left alignment

(R/W)

Continued on the next page...

Register 50.2. LP_I2S_RX_CONF_REG (0x0020)

Continued from the previous page...

LP_I2S_RX_WS_IDLE_POL Configures the relationship between WS level and which channel data to receive.

0: WS remains low when receiving left channel data and high when receiving right channel data

1: WS remains high when receiving left channel data and low when receiving right channel data

(R/W)

LP_I2S_RX_BIT_ORDER Configures whether to reverse the bit order of the LP I2S RX data to be received.

0: Not reverse

1: Reverse

(R/W)

LP_I2S_RX_TDM_EN Configures whether to enable LP I2S TDM RX mode.

0: Disable

1: Enable

(R/W)

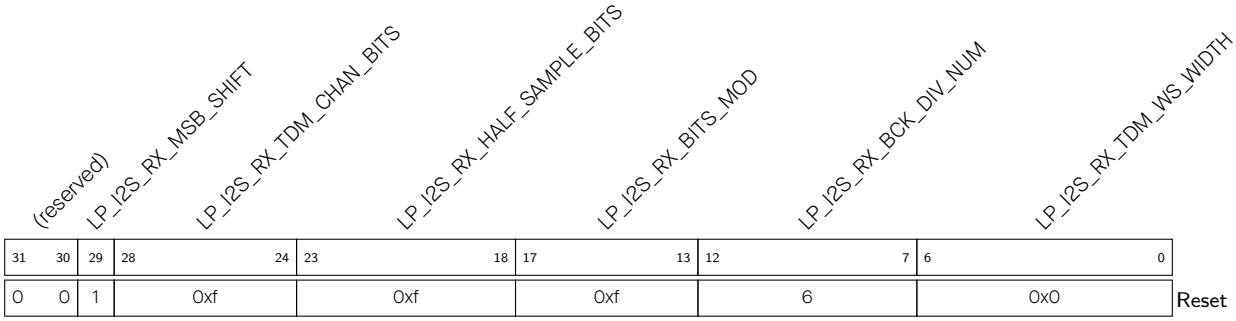
LP_I2S_RX_PDM_EN Configures whether to enable LP I2S PDM RX mode.

0: Disable

1: Enable

(R/W)

Register 50.3. LP_I2S_RX_CONF1_REG (0x0028)



LP_I2S_RX_TDM_WS_WIDTH Configures the width of LP_I2SI_WS_out (WS default level) at idle level in TDM mode. Width of LP_I2SI_WS_out at idle level in TDM mode = (LP_I2S_RX_TDM_WS_WIDTH[6:0] +1) x T_BCK. (R/W)

LP_I2S_RX_BCK_DIV_NUM Configures the divider of BCK in RX mode. Note this divider must not be configured to 1. (R/W)

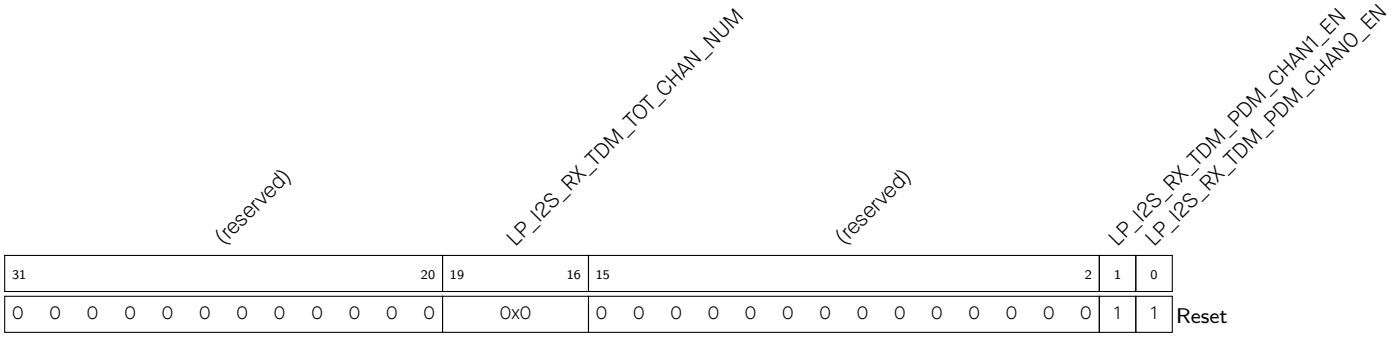
LP_I2S_RX_BITS_MOD Configures the valid data bit length of LP I2S RX channel. Since the LP I2S only supports 16-bit mode, this field must be configured to 15. LP I2S cannot function properly if this field is configured to any other value. (R/W)

LP_I2S_RX_HALF_SAMPLE_BITS Configures LP I2S RX sample bits. BCK cycles in one WS period = I2S_RX_HALF_SAMPLE_BITS x 2. (R/W)

LP_I2S_RX_TDM_CHAN_BITS Configures RX bit number for each channel in TDM mode. Bit number expected = LP_I2S_RX_TDM_CHAN_BITS + 1. (R/W)

LP_I2S_RX_MSB_SHIFT Configures the timing between the WS signal and the MSB of data.
0: Align at the rising edge
1: The WS signal changes one BCK clock earlier
(R/W)

Register 50.4. LP_I2S_RX_TDM_CTRL_REG (0x0050)



LP_I2S_RX_TDM_PDM_CHAN0_EN Configures whether to enable the valid data input of LP I2S RX TDM or PDM channel 0.

0: Disable. Channel 0 only inputs 0

1: Enable

(R/W)

LP_I2S_RX_TDM_PDM_CHAN1_EN Configures whether to enable the valid data input of LP I2S RX TDM or PDM channel 1.

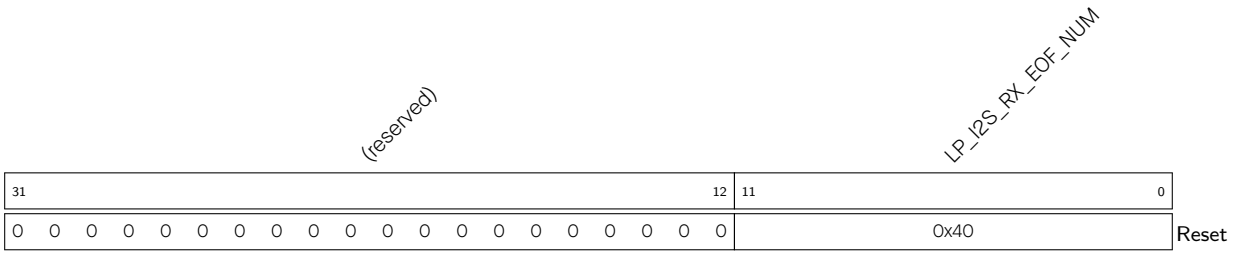
0: Disable. Channel 1 only inputs 0

1: Enable

(R/W)

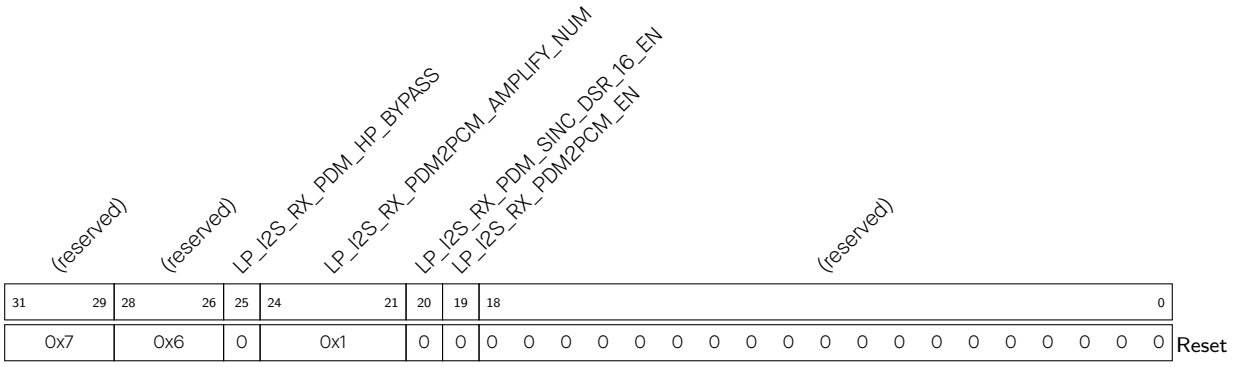
LP_I2S_RX_TDM_TOT_CHAN_NUM Configures the total channel number of LP I2S RX TDM mode. Total channel number in use = LP_I2S_RX_TDM_TOT_CHAN_NUM + 1. (R/W)

Register 50.5. LP_I2S_RXEOF_NUM_REG (0x0064)



LP_I2S_RX_EOF_NUM Configures the bit length of RX data. Bit length of RX data = (LP_I2S_RX_BITS_MOD[4:0] + 1) x (LP_I2S_RX_EOF_NUM[11:0] + 1). Valid when LP_I2S_RX_STOP_MODE is set to 1. (R/W)

Register 50.6. LP_I2S_RX_PDM_CONF_REG (0x0070)



LP_I2S_RX_PDM2PCM_EN Configures whether to enable the PDM-to-PCM converter in RX mode.

0: Disable

1: Enable

(R/W)

LP_I2S_RX_PDM_SINC_DSR_16_EN Configures the downsampling rate of PDM RX filter group 1 module.

0: 64

1: 128

(R/W)

LP_I2S_RX_PDM2PCM_AMPLIFY_NUM Configures the PDM-to-PCM RX amplification coefficient. PCM data will be multiplied by this value before outputting. (R/W)

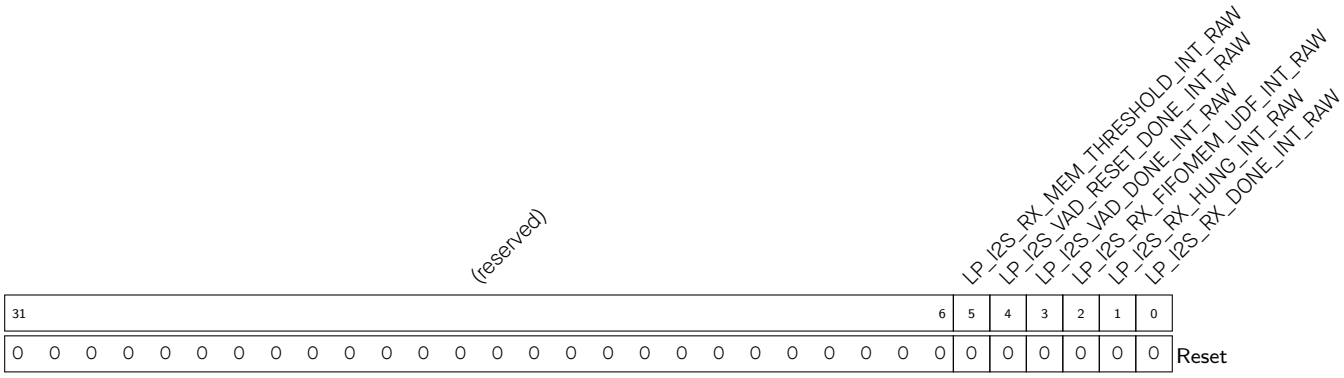
LP_I2S_RX_PDM_HP_BYPASS Configures whether PDM-to-PCM RX bypasses the HP filter.

0: Not bypass

1: Bypass

(R/W)

Register 50.7. LP_I2S_INT_RAW_REG (0x000C)



LP_I2S_RX_DONE_INT_RAW The raw interrupt status of the LP_I2S_RX_DONE_INT interrupt. (R/SS/WTC)

LP_I2S_RX_HUNG_INT_RAW The raw interrupt status of the LP_I2S_RX_DONE_INT interrupt. (R/SS/WTC)

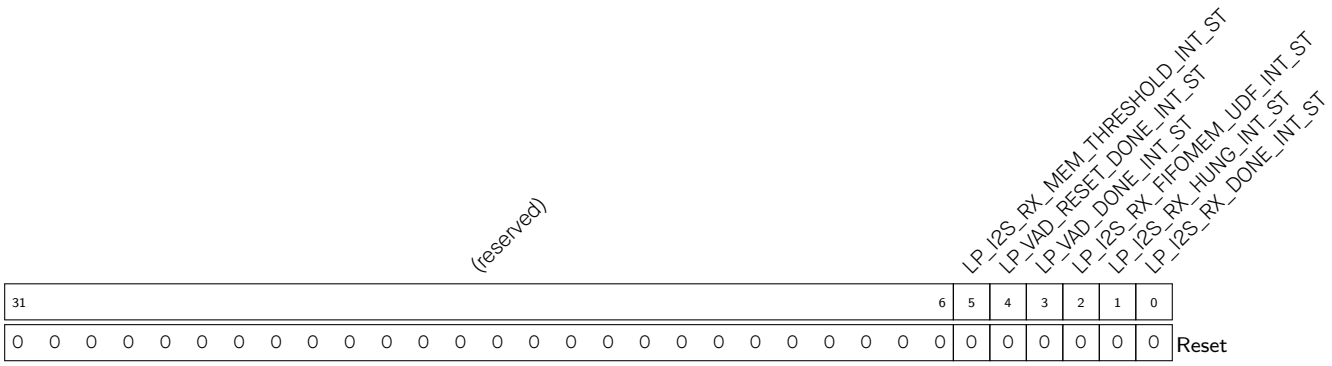
LP_I2S_RX_FIFOMEM_UDF_INT_RAW The raw interrupt status of the LP_I2S_RX_FIFOMEM_UDF_INT interrupt. (R/SS/WTC)

LP_I2S_VAD_DONE_INT_RAW The raw interrupt status of the LP_I2S_VAD_DONE_INT interrupt. (R/SS/WTC)

LP_I2S_VAD_RESET_DONE_INT_RAW The raw interrupt status of the LP_I2S_VAD_RESET_DONE_INT interrupt. (R/SS/WTC)

LP_I2S_RX_MEM_THRESHOLD_INT_RAW The raw interrupt status of the LP_I2S_RX_MEM_THRESHOLD_INT interrupt. (R/SS/WTC)

Register 50.8. LP_I2S_INT_ST_REG (0x0010)



LP_I2S_RX_DONE_INT_ST The masked interrupt status of the LP_I2S_RX_DONE_INT interrupt. (RO)

LP_I2S_RX_HUNG_INT_ST The masked interrupt status of the LP_I2S_RX_DONE_INT interrupt. (RO)

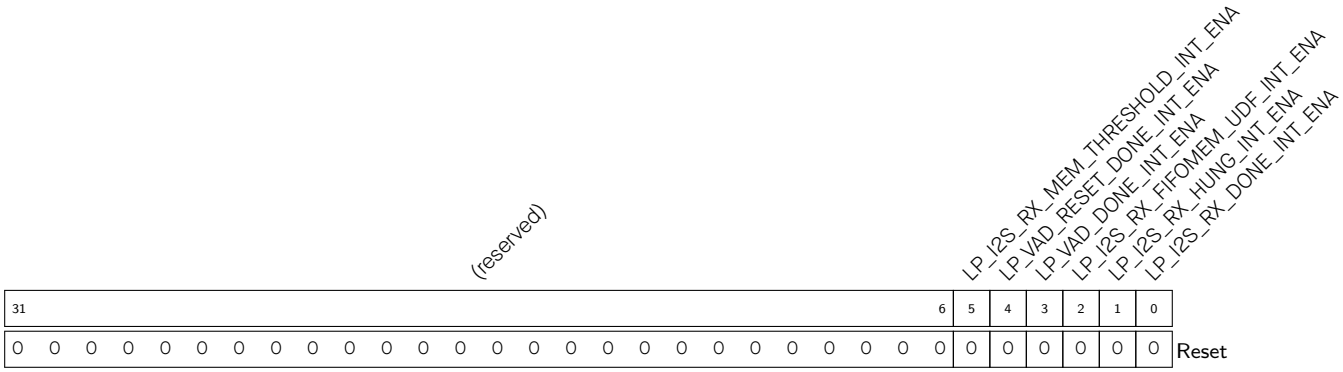
LP_I2S_RX_FIFOMEM_UDF_INT_ST The masked interrupt status of the LP_I2S_RX_FIFOMEM_UDF_INT interrupt. (RO)

LP_VAD_DONE_INT_ST The masked interrupt status of the LP_I2S_VAD_DONE_INT interrupt. (RO)

LP_VAD_RESET_DONE_INT_ST The masked interrupt status of the LP_I2S_VAD_RESET_DONE_INT interrupt. (RO)

LP_I2S_RX_MEM_THRESHOLD_INT_ST The masked interrupt status of the LP_I2S_RX_MEM_THRESHOLD_INT interrupt. (RO)

Register 50.9. LP_I2S_INT_ENA_REG (0x0014)



LP_I2S_RX_DONE_INT_ENA Write 1 to enable the LP_I2S_RX_DONE_INT interrupt. (R/W)

LP_I2S_RX_HUNG_INT_ENA Write 1 to enable the LP_I2S_RX_DONE_INT interrupt. (R/W)

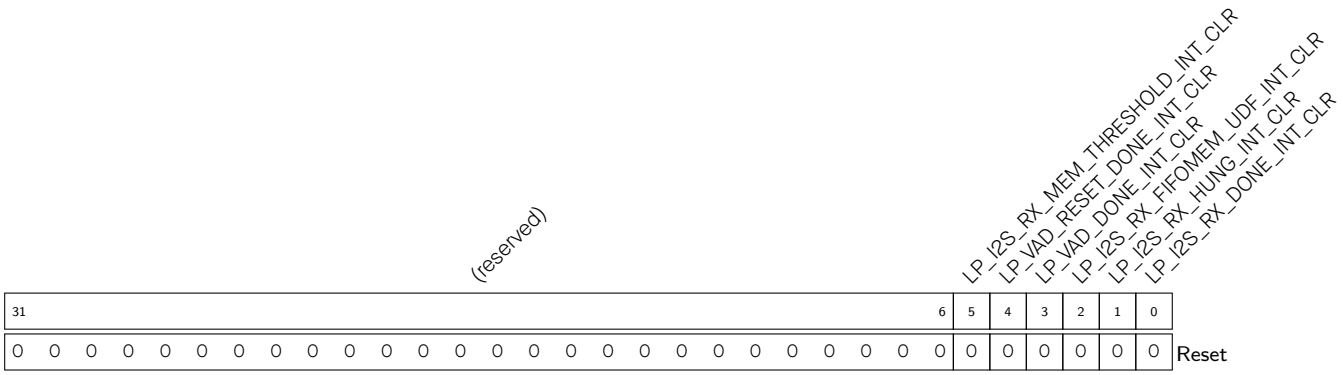
LP_I2S_RX_FIFOMEM_UDF_INT_ENA Write 1 to enable the LP_I2S_RX_FIFOMEM_UDF_INT interrupt. (R/W)

LP_VAD_DONE_INT_ENA Write 1 to enable the LP_I2S_VAD_DONE_INT interrupt. (R/W)

LP_VAD_RESET_DONE_INT_ENA Write 1 to enable the LP_I2S_VAD_RESET_DONE_INT interrupt. (R/W)

LP_I2S_RX_MEM_THRESHOLD_INT_ENA Write 1 to enable the LP_I2S_RX_MEM_THRESHOLD_INT interrupt. (R/W)

Register 50.10. LP_I2S_INT_CLR_REG (0x0018)



LP_I2S_RX_DONE_INT_CLR Write 1 to clear the LP_I2S_RX_DONE_INT interrupt. (WT)

LP_I2S_RX_HUNG_INT_CLR Write 1 to clear the LP_I2S_RX_DONE_INT interrupt. (WT)

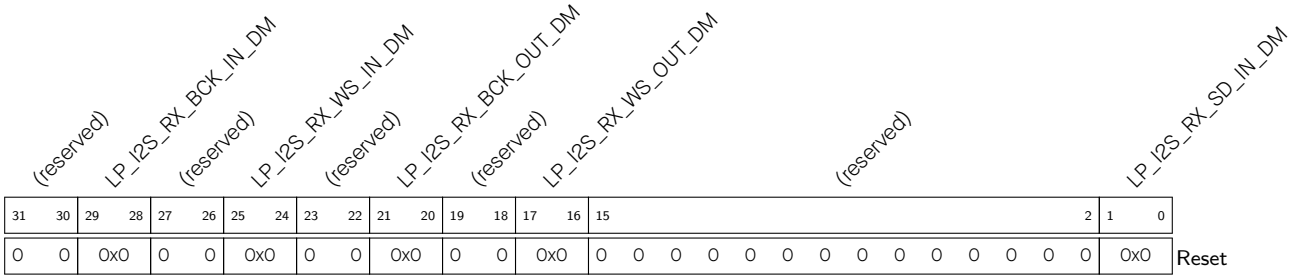
LP_I2S_RX_FIFOMEM_UDF_INT_CLR Write 1 to clear the LP_I2S_RX_FIFOMEM_UDF_INT interrupt. (WT)

LP_VAD_DONE_INT_CLR Write 1 to clear the LP_I2S_VAD_DONE_INT interrupt. (WT)

LP_VAD_RESET_DONE_INT_CLR Write 1 to clear the LP_I2S_VAD_RESET_DONE_INT interrupt. (WT)

LP_I2S_RX_MEM_THRESHOLD_INT_CLR Write 1 to clear the LP_I2S_RX_MEM_THRESHOLD_INT interrupt. (WT)

Register 50.11. LP_I2S_RX_TIMING_REG (0x0058)



LP_I2S_RX_SD_IN_DM Configures the delay mode of LP I2S RX SD input signal.

0: Bypass

1: Delay by positive edge

2: Delay by negative edge

3: Invalid

(R/W)

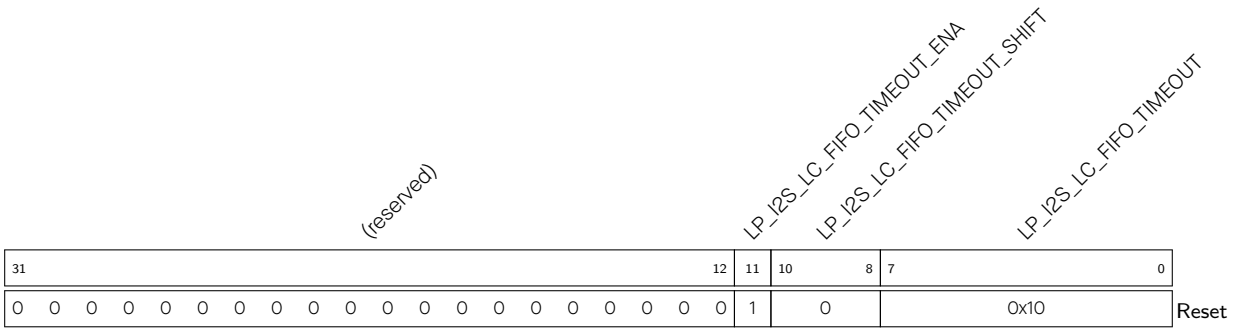
LP_I2S_RX_WS_OUT_DM Configures the delay mode of LP I2S RX WS output signal. For detailed configuration values, please refer to LP_I2S_RX_SD_IN_DM. (R/W)

LP_I2S_RX_BCK_OUT_DM Configures the delay mode of LP I2S RX BCK output signal. For detailed configuration values, please refer to LP_I2S_RX_SD_IN_DM. (R/W)

LP_I2S_RX_WS_IN_DM Configures the delay mode of LP I2S RX WS input signal. For detailed configuration values, please refer to LP_I2S_RX_SD_IN_DM.(R/W)

LP_I2S_RX_BCK_IN_DM Configures the delay mode of LP I2S RX BCK input signal. For detailed configuration values, please refer to LP_I2S_RX_SD_IN_DM. (R/W)

Register 50.12. LP_I2S_LC_HUNG_CONF_REG (0x0060)



LP_I2S_LC_FIFO_TIMEOUT Configures FIFO timeout threshold. The FIFO hung counter is incremented by one each time LP I2S is in an error state and the tick counter reaches its threshold. LP_I2S_RX_HUNG_INT interrupt will be triggered when FIFO hung counter is equal to the FIFO timeout threshold. (R/W)

LP_I2S_LC_FIFO_TIMEOUT_SHIFT Configures tick counter threshold. The tick counter is incremented by one each time LP I2S is in an error state and it is on the rising edge in each system clock (APB). The tick counter is reset when counter value >= $88000/2^{LP_I2S_LC_FIFO_TIMEOUT_SHIFT}$. (R/W)

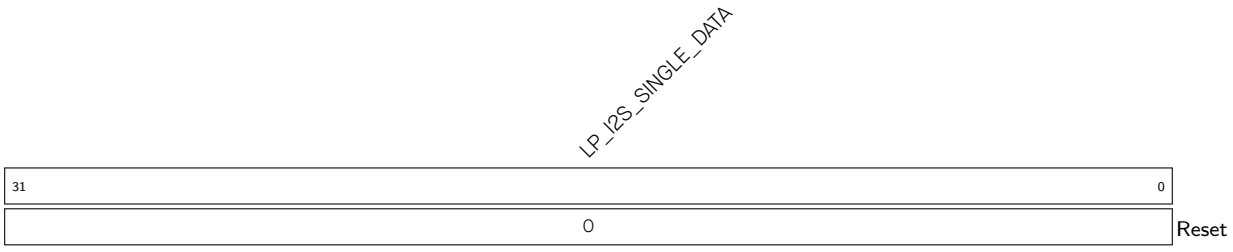
LP_I2S_LC_FIFO_TIMEOUT_ENA The enable bit for FIFO timeout. Configures whether to enable FIFO timeout.

0: Disable

1: Enable

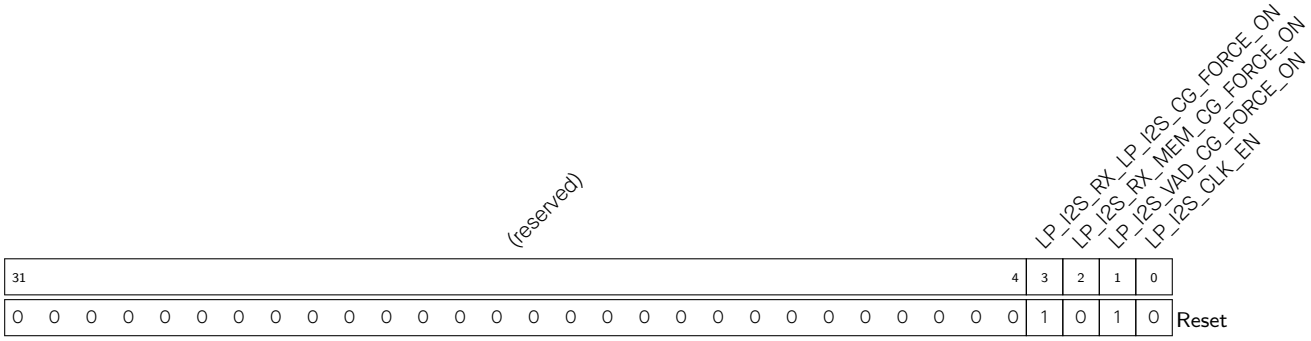
(R/W)

Register 50.13. LP_I2S_CONF_SIGLE_DATA_REG (0x0068)



LP_I2S_SINGLE_DATA Configures the constant channel data to be sent out. (R/W)

Register 50.14. LP_I2S_CLK_GATE_REG (0x00F8)



LP_I2S_CLK_EN Configures whether to enable clock gate.

0: Disable

1: Enable

(R/W)

LP_I2S_VAD_CG_FORCE_ON Configures whether to force the VAD clock gate on.

0: No effect

1: Force on

(R/W)

LP_I2S_RX_MEM_CG_FORCE_ON Configures whether to force the LP I2S RX memory clock gate on.

0: No effect

1: Force on

(R/W)

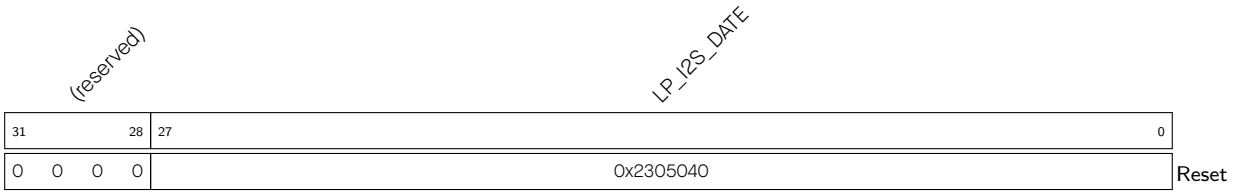
LP_I2S_RX_LP_I2S_CG_FORCE_ON Configures whether to force the LP I2S RX register clock gate on.

0: No effect

1: Force on

(R/W)

Register 50.15. LP_I2S_DATE_REG (0x00FC)



LP_I2S_DATE Version control register. (R/W)

Chapter 51

Pulse Count Controller (PCNT)

The pulse count controller (PCNT) is designed to count input pulses.

51.1 Introduction

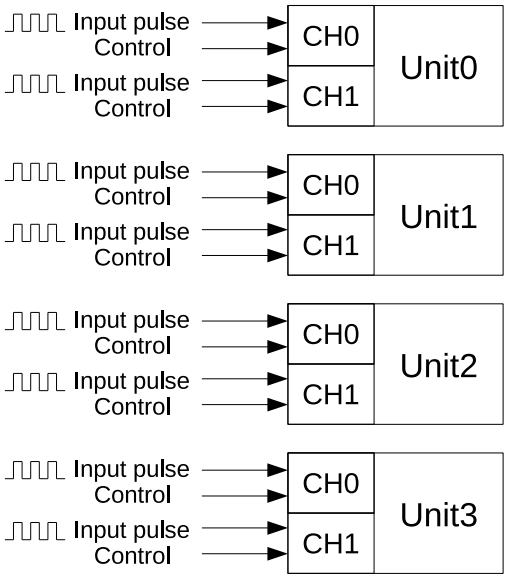


Figure 51.1-1. PCNT Overview

As shown in Figure 51.1-1, PCNT has four independent pulse counters called units, which have their groups of registers. Each unit includes two channels (ch0 and ch1) and a 16-bit signed counter. Each channel has an input pulse signal and an input signal filtering module. The 16-bit signed counter can be configured for incremental or decremental counting. The clock for PCNT module is APB_CLK.

51.2 Feature List

A PCNT has the following features:

- Four independent pulse counters (units) that count from 1 to 65535
- Each unit consists of two independent channels sharing one pulse counter
- All channels have input pulse signals (e.g., sig_ch0_u*n*) with their corresponding control signals (e.g., ctrl_ch0_u*n*)
- Independently filter glitches of input pulse signals (sig_ch0_u*n* and sig_ch1_u*n*) and control signals (ctrl_ch0_u*n* and ctrl_ch1_u*n*) on each unit

- Each channel has the following parameters:
  1. Selection between counting on rising or falling edges of the input pulse signal
  2. Configuration to Increment, Decrement, or Disable counter mode for control signal's high and low states
- Maximum frequency of pulses: $\frac{f_{APB_CLK}}{2}$

51.3 Architectural Overview

Each unit of PCNT includes two channels (ch0 and ch1) and the functionality of the two channels is identical. The remainder of the chapter will take channel 0 (ch0) as example and *n* denotes the number of a unit from 0 ~ 3. Each channel of PCNT has two input signals:

1. One input pulse signal (e.g., sig_ch0_u*n*, the input pulse signal for ch0 of unit *n* ch0)
2. One control signal (e.g., ctrl_ch0_u*n*, the control signal for ch0 of unit *n* ch0)

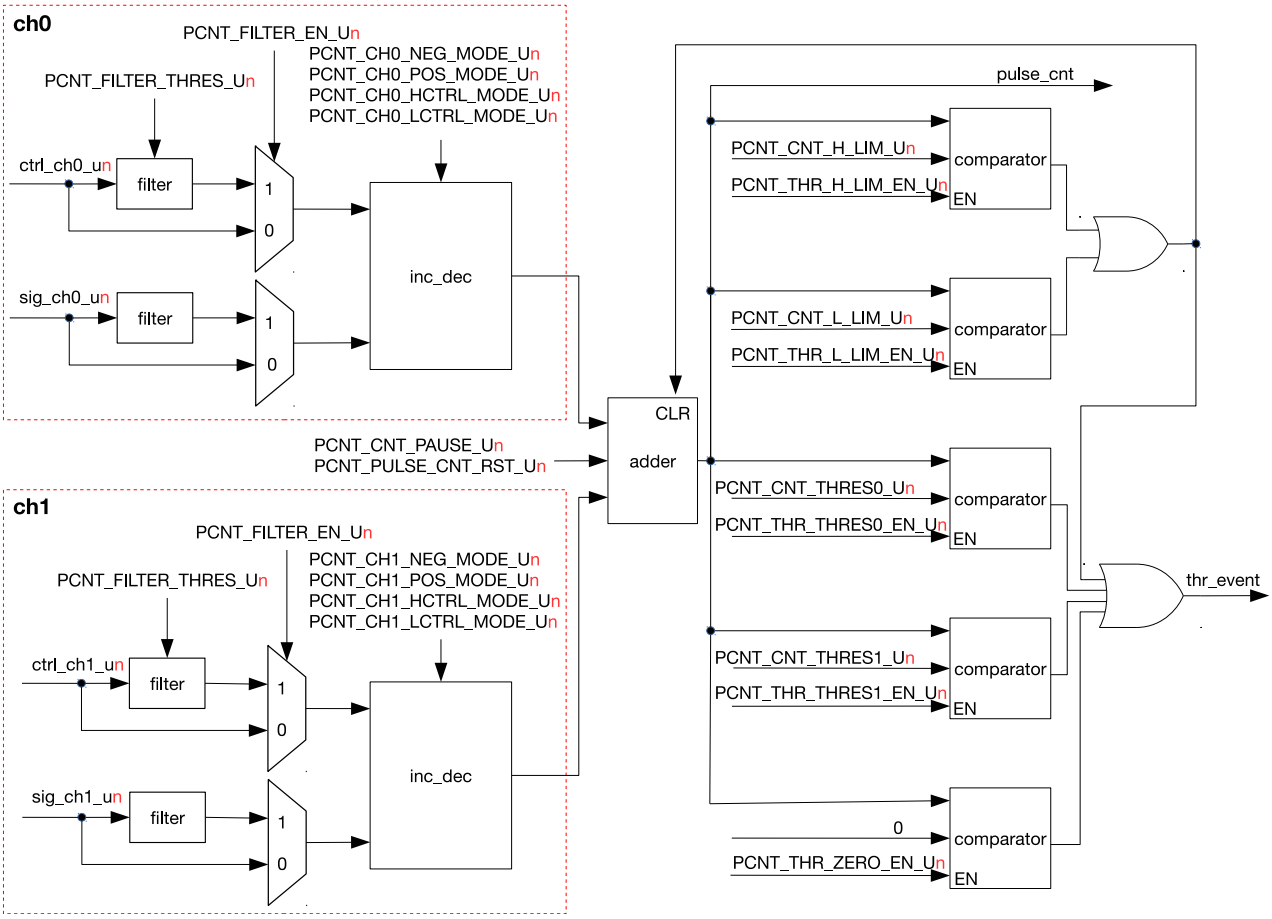


Figure 51.3-1. PCNT Architectural Overview

Figure 51.3-1 shows PCNT's architecture. As stated above, ctrl_ch0_u*n* is the control signal for ch0 of unit *n*. Its high and low states can be assigned in different counter modes and used for pulse counting of the channel's input pulse signal sig_ch0_u*n* on falling or rising edges.

51.4 Functional Description

The available counter modes of PCNT are as follows:

- Increment mode: When a channel detects an active edge of sig_ch0_u*n* (can be configured by software), the counter value pulse_cnt increases by 1. Upon reaching PCNT_CNT_H_LIM_U*n*, pulse_cnt is cleared. If PCNT_CNT_PAUSE_U*n* is set to 1, then pulse_cnt freezes.
- Decrement mode: When a channel detects an active edge of sig_ch0_u*n* (can be configured by software), the counter value pulse_cnt decreases by 1. Upon reaching PCNT_CNT_L_LIM_U*n*, pulse_cnt is cleared. If PCNT_CNT_PAUSE_U*n* is set to 1, then pulse_cnt freezes.
- Disable mode: Counting is disabled, and the counter value pulse_cnt freezes.

Table 51.4-1 to Table 51.4-4 provide information on how to configure the counter mode for channel 0.

Table 51.4-1. Counter Mode. Rising edge of Input Pulse Signal. Control Signal in Low State

PCNT_CH0_POS_MODE_U*n*	PCNT_CH0_LCTRL_MODE_U*n*	Counter Mode
1	0	Increment
	1	Decrement
	Others	Disable
2	0	Decrement
	1	Increment
	Others	Disable
Others	N/A	Disable

Table 51.4-2. Counter Mode. Rising edge of Input Pulse Signal. Control Signal in High State

PCNT_CH0_POS_MODE_U*n*	PCNT_CH0_HCTRL_MODE_U*n*	Counter Mode
1	0	Increment
	1	Decrement
	Others	Disable
2	0	Decrement
	1	Increment
	Others	Disable
Others	N/A	Disable

Table 51.4-3. Counter Mode. Falling Edge of Input Pulse Signal. Control Signal in Low State

PCNT_CH0_NEG_MODE_U*n*	PCNT_CH0_LCTRL_MODE_U*n*	Counter Mode
1	0	Increment
	1	Decrement
	Others	Disable
2	0	Decrement
	1	Increment
	Others	Disable
Others	N/A	Disable

Table 51.4-4. Counter Mode. Falling Edge of Input Pulse Signal. Control Signal in High State

PCNT_CHO_NEG_MODE_U*n*	PCNT_CHO_HCTRL_MODE_U*n*	Counter Mode
1	0	Increment
	1	Decrement
	Others	Disable
2	0	Decrement
	1	Increment
	Others	Disable
Others	N/A	Disable

Each unit has one filter for all its control and input pulse signals. A filter can be enabled by setting the bit PCNT_FILTER_EN_U*n*. The filter monitors the signals and ignores all the noise, i.e., the glitches with pulse widths shorter than PCNT_FILTER_THRES_U*n* APB clock cycles in length.

As shown on Figure 51.3-1, each unit has two channels which process different input pulse signals and increase or decrease values via their respective inc_dec modules, then the two channels send these values to the counter module which has a 16-bit wide signed register. This counter can be suspended by setting PCNT_CNT_PAUSE_U*n*, and cleared by setting PCNT_PULSE_CNT_RST_U*n*.

The PCNT has five watchpoints that share one interrupt. The interrupt can be enabled or disabled by interrupt enable signals of each individual watchpoint.

- Maximum count value: When pulse_cnt reaches PCNT_CNT_H_LIM_U*n*, a high limit interrupt is triggered and PCNT_CNT_THR_H_LIM_LAT_U*n* is high.
- Minimum count value: When pulse_cnt reaches PCNT_CNT_L_LIM_U*n*, a low limit interrupt is triggered and PCNT_CNT_THR_L_LIM_LAT_U*n* is high.
- Two threshold values: When pulse_cnt equals either PCNT_CNT_THRESO_U*n* or PCNT_CNT_THRES1_U*n*, an interrupt is triggered and either PCNT_CNT_THR_THRESO_LAT_U*n* or PCNT_CNT_THR_THRES1_LAT_U*n* is high respectively.
- Zero: When pulse_cnt is 0, an interrupt is triggered and PCNT_CNT_THR_ZERO_LAT_U*n* is valid.

If PCNT_CNT_H_LIM_U*n* and/or PCNT_CNT_L_LIM_U*n* are reconfigured by software when PCNT is working, the new configuration will take effect after pulse_cnt counts to any of the above five watchpoints; If PCNT_CNT_THRESO_U*n* and/or PCNT_CNT_THRES1_U*n* are reconfigured by software, the new configuration will take effect immediately.

51.5 Interrupts

ESP32-P4's PCNT can generate the following interrupt signal that will be sent to the Interrupt Matrix.

- PCNT_INT

There are several internal interrupt sources from PCNT that can generate the above interrupt signal. The interrupt sources from PCNT are listed with their trigger conditions and the resulted interrupt signal in Table 51.5-1.

Table 51.5-1. PCNT's Internal Interrupt Sources

Internal Interrupt Source	Trigger Condition	Interrupt Signal
PCNT_CNT_THR_EVENT_U0_INT	reaching the watchpoint in PCNT unit 0	PCNT_INT
PCNT_CNT_THR_EVENT_U1_INT	reaching the watchpoint in PCNT unit 1	PCNT_INT
PCNT_CNT_THR_EVENT_U2_INT	reaching the watchpoint in PCNT unit 2	PCNT_INT
PCNT_CNT_THR_EVENT_U3_INT	reaching the watchpoint in PCNT unit 3	PCNT_INT

Note:
For definitions of *interrupt*, *interrupt signal*, *interrupt source*, and their correlations, please refer to Chapter 13 *Interrupt Matrix* > Section 13.2 *Interrupt Terminology in ESP32-P4*.

51.6 Programming Procedures

In each unit, channel 0 and channel 1 can be configured to work independently or together. The three subsections below provide details of channel 0 incrementing independently, channel 0 decrementing independently, and channel 0 and channel 1 incrementing together. For other working modes not elaborated in this section (e.g., channel 1 incrementing/decremeting independently, or one channel incrementing while the other decrementing), reference can be made to these three subsections.

51.6.1 Channel 0 Incrementing Independently

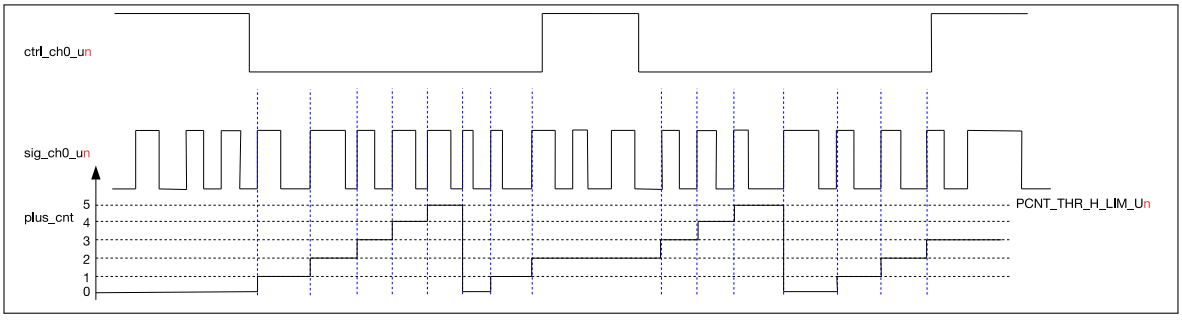


Figure 51.6-1. Channel 0 Up Counting Diagram

Figure 51.6-1 illustrates how channel 0 is configured to increment independently on the rising edge of sig_ch0_u*n* while channel 1 is disabled (see Subsection 51.4 for how to disable channel 1). The configuration of channel 0 is shown below.

- PCNT_CH0_LCTRL_MODE_U*n*=0: When ctrl_ch0_u*n* is low, the counter mode specified for the low state turns on, in this case it is Increment mode.
- PCNT_CH0_HCTRL_MODE_U*n*=2: When ctrl_ch0_u*n* is high, the counter mode specified for the low state turns on, in this case it is Disable mode.
- PCNT_CH0_POS_MODE_U*n*=1: The counter increments on the rising edge of sig_ch0_u*n*.

- PCNT_CH0_NEG_MODE_U*n*=0: The counter idles on the falling edge of sig_ch0_u*n*.
- PCNT_CNT_H_LIM_U*n*=5: When pulse_cnt counts up to PCNT_CNT_H_LIM_U*n*, it is cleared.

51.6.2 Channel 0 Decrementing Independently

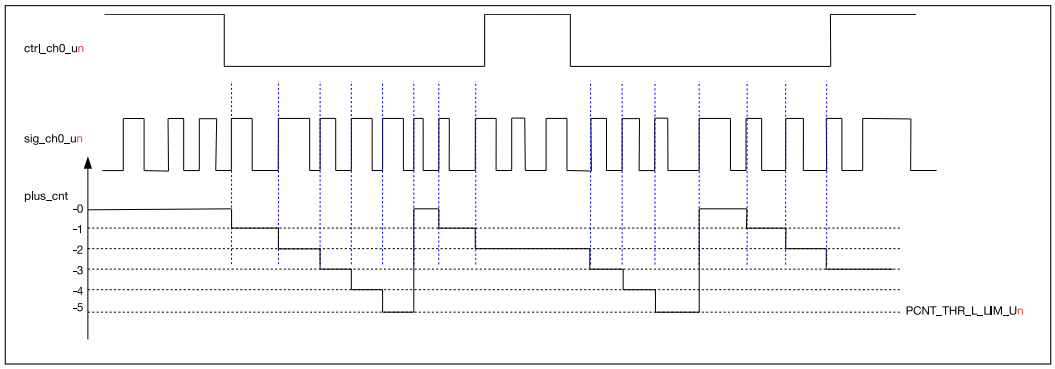


Figure 51.6-2. Channel 0 Down Counting Diagram

Figure 51.6-2 illustrates how channel 0 is configured to decrement independently on the rising edge of sig_ch0_u*n* while channel 1 is disabled. The configuration of channel 0 in this case differs from that in Figure 51.6-1 in the following aspects:

- PCNT_CH0_POS_MODE_U*n*=2: the counter decrements on the rising edge of sig_ch0_u*n*.
- PCNT_CNT_L_LIM_U*n*=-5: when pulse_cnt counts down to PCNT_CNT_L_LIM_U*n*, it is cleared.

51.6.3 Channel 0 and Channel 1 Incrementing Together

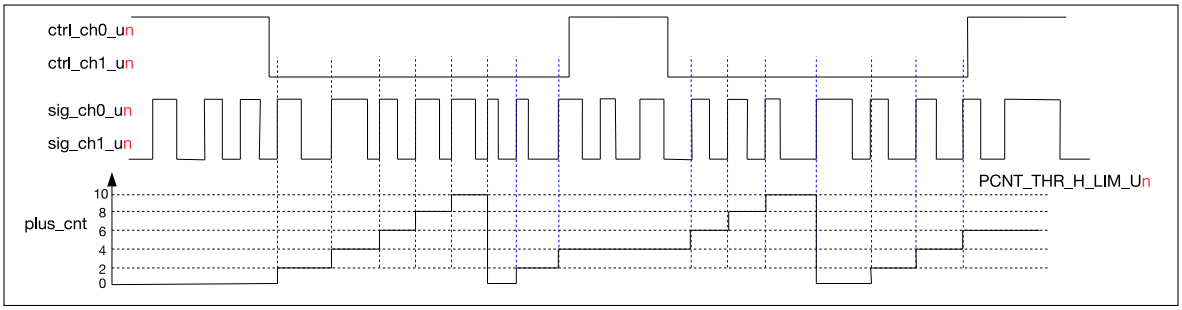


Figure 51.6-3. Two Channels Up Counting Diagram

Figure 51.6-3 illustrates how channel 0 and channel 1 are configured to increment on the rising edge of sig_ch0_u*n* and sig_ch1_u*n* respectively at the same time. It can be seen in Figure 51.6-3 that control signal ctrl_ch0_u*n* and ctrl_ch1_u*n* have the same waveform, so as input pulse signal sig_ch0_u*n* and sig_ch1_u*n*. The configuration procedure is shown below.

- For channel 0:
  - PCNT_CH0_LCTRL_MODE_U*n*=0: When ctrl_ch0_u*n* is low, the counter mode specified for the low state turns on, in this case it is Increment mode.

  - PCNT_CH0_HCTRL_MODE_U*n*=2: When ctrl_ch0_u*n* is high, the counter mode specified for the low state turns on, in this case it is Disable mode.
  - PCNT_CH0_POS_MODE_U*n*=1: The counter increments on the rising edge of sig_ch0_u*n*.
  - PCNT_CH0_NEG_MODE_U*n*=0: The counter idles on the falling edge of sig_ch0_u*n*.
- For channel 1:
  - PCNT_CH1_LCTRL_MODE_U*n*=0: When ctrl_ch1_u*n* is low, the counter mode specified for the low state turns on, in this case it is Increment mode.
  - PCNT_CH1_HCTRL_MODE_U*n*=2: When ctrl_ch1_u*n* is high, the counter mode specified for the low state turns on, in this case it is Disable mode.
  - PCNT_CH1_POS_MODE_U*n*=1: The counter increments on the rising edge of sig_ch1_u*n*.
  - PCNT_CH1_NEG_MODE_U*n*=0: The counter idles on the falling edge of sig_ch1_u*n*.
- PCNT_CNT_H_LIM_U*n*=10: When pulse_cnt counts up to PCNT_CNT_H_LIM_U*n*, it is cleared.

51.7 Register Summary

The addresses in this section are relative to Pulse Count Controller base address provided in Table 9.3-2 in Chapter 9 *System and Memory*.

The abbreviations given in Column **Access** are explained in Section *Access Types for Registers*.

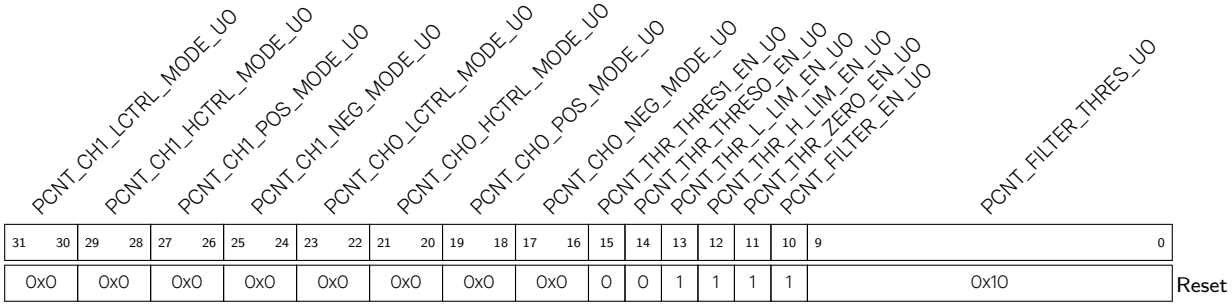
Name	Description	Address	Access
Configuration Register			
PCNT_U0_CONF0_REG	Configuration register 0 for unit 0	0x0000	R/W
PCNT_U0_CONF1_REG	Configuration register 1 for unit 0	0x0004	R/W
PCNT_U0_CONF2_REG	Configuration register 2 for unit 0	0x0008	R/W
PCNT_U1_CONF0_REG	Configuration register 0 for unit 1	0x000C	R/W
PCNT_U1_CONF1_REG	Configuration register 1 for unit 1	0x0010	R/W
PCNT_U1_CONF2_REG	Configuration register 2 for unit 1	0x0014	R/W
PCNT_U2_CONF0_REG	Configuration register 0 for unit 2	0x0018	R/W
PCNT_U2_CONF1_REG	Configuration register 1 for unit 2	0x001C	R/W
PCNT_U2_CONF2_REG	Configuration register 2 for unit 2	0x0020	R/W
PCNT_U3_CONF0_REG	Configuration register 0 for unit 3	0x0024	R/W
PCNT_U3_CONF1_REG	Configuration register 1 for unit 3	0x0028	R/W
PCNT_U3_CONF2_REG	Configuration register 2 for unit 3	0x002C	R/W
PCNT_CTRL_REG	Control register for all counters	0x0060	R/W
Status Register			
PCNT_U0_CNT_REG	Counter value for unit 0	0x0030	RO
PCNT_U1_CNT_REG	Counter value for unit 1	0x0034	RO
PCNT_U2_CNT_REG	Counter value for unit 2	0x0038	RO
PCNT_U3_CNT_REG	Counter value for unit 3	0x003C	RO
PCNT_U0_STATUS_REG	PNCT UNIT0 status register	0x0050	RO
PCNT_U1_STATUS_REG	PNCT UNIT1 status register	0x0054	RO
PCNT_U2_STATUS_REG	PNCT UNIT2 status register	0x0058	RO
PCNT_U3_STATUS_REG	PNCT UNIT3 status register	0x005C	RO
Interrupt Register			
PCNT_INT_RAW_REG	Interrupt raw status register	0x0040	RO
PCNT_INT_ST_REG	Interrupt status register	0x0044	RO
PCNT_INT_ENA_REG	Interrupt enable register	0x0048	R/W
PCNT_INT_CLR_REG	Interrupt clear register	0x004C	WO
Version Register			
PCNT_DATE_REG	PCNT version control register	0x00FC	R/W

51.8 Registers

The addresses in this section are relative to Pulse Count Controller base address provided in Table 9.3-2 in Chapter 9 *System and Memory*.

For how to program reserved fields, please refer to Section Programming Reserved Register Field.

Register 51.1. PCNT_U*n*_CONF0_REG (*n*: 0-3) (0x0000+0xCn*)**



PCNT_FILTER_THRES_U*n* Configures the maximum threshold for the filter. Any pulses with width less than this will be ignored when the filter is enabled.

Measurement unit: APB_CLK cycles.

(R/W)

PCNT_FILTER_EN_U*n* This is the enable bit for unit *n*'s input filter. (R/W)

PCNT_THR_ZERO_EN_U*n* This is the enable bit for unit *n*'s zero comparator. (R/W)

PCNT_THR_H_LIM_EN_U*n* This is the enable bit for unit *n*'s thr_h_lim comparator. Configures it to enable the high limit interrupt. (R/W)

PCNT_THR_L_LIM_EN_U*n* This is the enable bit for unit *n*'s thr_l_lim comparator. Configures it to enable the low limit interrupt. (R/W)

PCNT_THR_THRES0_EN_U*n* This is the enable bit for unit *n*'s thres0 comparator. (R/W)

PCNT_THR_THRES1_EN_U*n* This is the enable bit for unit *n*'s thres1 comparator. (R/W)

PCNT_CH0_NEG_MODE_U*n* Configures the behavior when the signal input of channel 0 detects a falling edge.

1: Increment the counter

2: Decrement the counter

0, 3: No effect

(R/W)

PCNT_CH0_POS_MODE_U*n* Configures the behavior when the signal input of channel 0 detects a rising edge.

1: Increment the counter

2: Decrement the counter

0, 3: No effect

(R/W)

PCNT_CH0_HCTRL_MODE_U*n* Configures how the CH*n*_POS_MODE/CH*n*_NEG_MODE settings will be modified when the control signal is high.

0: No modification

1: Invert behavior (increase → decrease, decrease → increase)

2, 3: Inhibit counter modification

(R/W)

Continued on the next page...

Register 51.1. PCNT_U*n*_CONF0_REG (*n*: 0-3) (0x0000+0xC**n*)

Continued from the previous page...

PCNT_CH0_LCTRL_MODE_U*n* Configures how the CH*n*_POS_MODE/CH*n*_NEG_MODE settings will be modified when the control signal is low.
0: No modification
1: Invert behavior (increase → decrease, decrease → increase)
2, 3: Inhibit counter modification
(R/W)

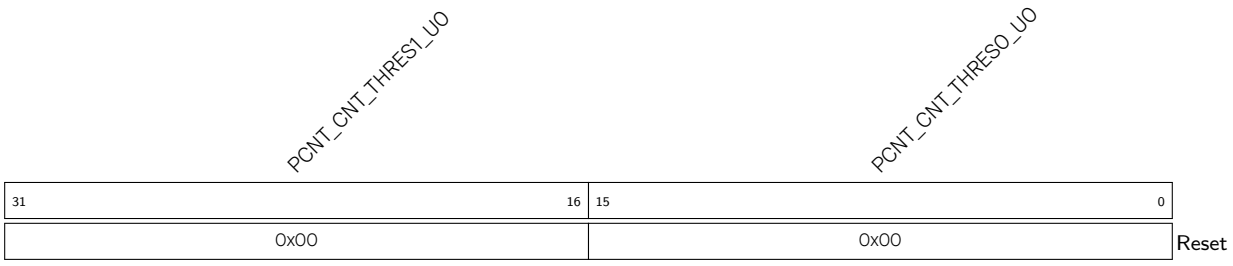
PCNT_CH1_NEG_MODE_U*n* Configures the behavior when the signal input of channel 1 detects a falling edge.
1: Increment the counter
2: Decrement the counter
0, 3: No effect
(R/W)

PCNT_CH1_POS_MODE_U*n* Configures the behavior when the signal input of channel 1 detects a rising edge.
1: Increment the counter
2: Decrement the counter
0, 3: No effect
(R/W)

PCNT_CH1_HCTRL_MODE_U*n* Configures how the CH*n*_POS_MODE/CH*n*_NEG_MODE settings will be modified when the control signal is high.
0: No modification
1: Invert behavior (increase → decrease, decrease → increase)
2, 3: Inhibit counter modification
(R/W)

PCNT_CH1_LCTRL_MODE_U*n* Configures how the CH*n*_POS_MODE/CH*n*_NEG_MODE settings will be modified when the control signal is low.
0: No modification
1: Invert behavior (increase → decrease, decrease → increase)
2, 3: Inhibit counter modification
(R/W)

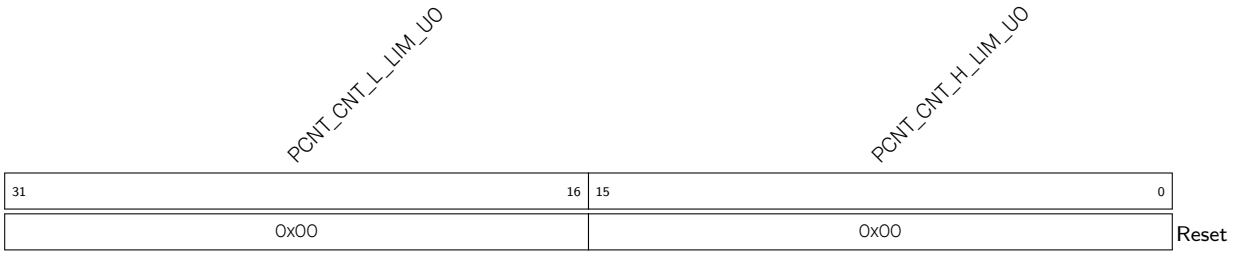
Register 51.2. PCNT_U*n*_CONF1_REG (*n*: 0-3) (0x0004+0xC**n*)



PCNT_CNT_THRES0_U*n* Configures the thres0 value for unit *n*. (R/W)

PCNT_CNT_THRES1_U*n* Configures the thres1 value for unit *n*. (R/W)

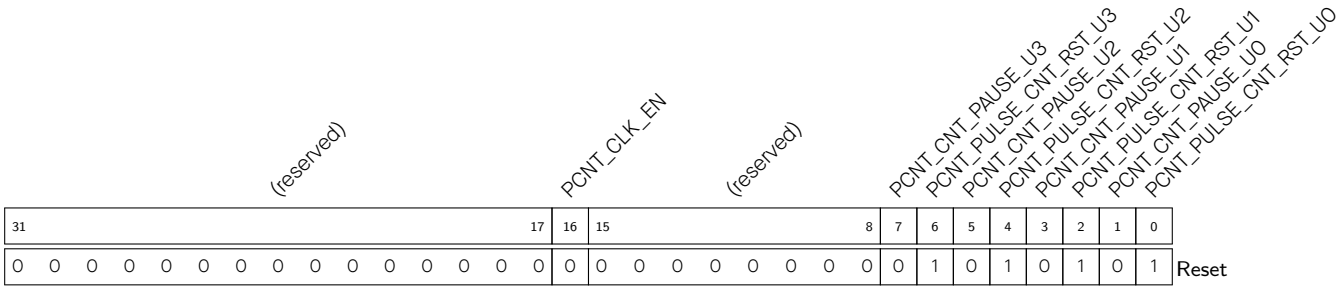
Register 51.3. PCNT_U*n*_CONF2_REG (*n*: 0-3) (0x0008+0xC**n*)



PCNT_CNT_H_LIM_U*n* Configures the thr_h_lim value for unit *n*. When pulse_cnt reaches this value, the counter will be cleared to 0. (R/W)

PCNT_CNT_L_LIM_U*n* Configures the thr_l_lim value for unit *n*. When pulse_cnt reaches this value, the counter will be cleared to 0.(R/W)

Register 51.4. PCNT_CTRL_REG (0x0060)



PCNT_PULSE_CNT_RST_U*n* Write 1 to clear unit *n*'s counter.

0: No effect

1: Clear

(R/W)

PCNT_CNT_PAUSE_U*n* Write 1 to freeze unit *n*'s counter.

0: No effect

1: Freeze

(R/W)

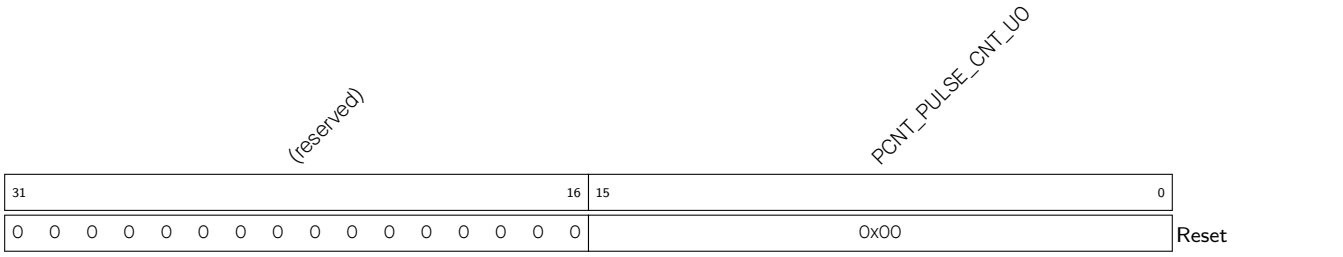
PCNT_CLK_EN Configures whether or not to enable the registers clock gate of PCNT module.

0: the clock for registers is enabled when registers are read and written

1: the clock for registers is always on

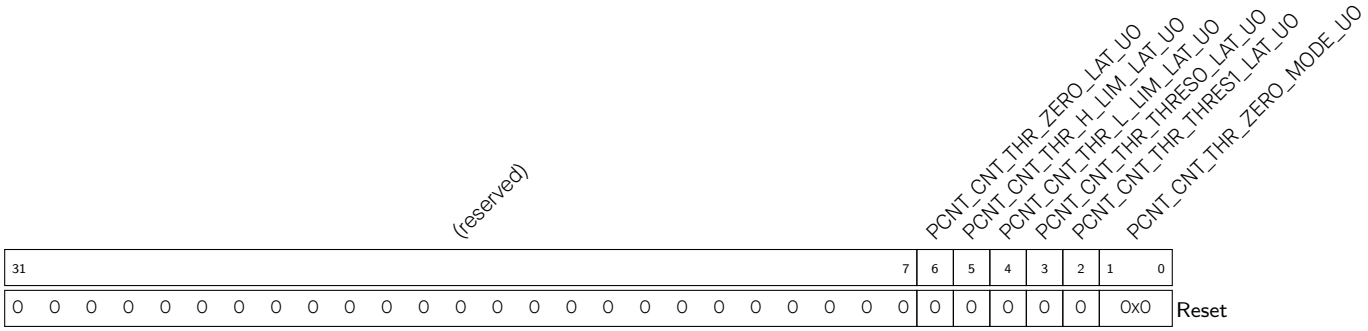
(R/W)

Register 51.5. PCNT_U*n*_CNT_REG (*n*: 0-3) (0x0030+0x4**n*)



PCNT_PULSE_CNT_U*n* Represents the current pulse count value for unit *n*. (RO)

Register 51.6. PCNT_U*n*_STATUS_REG (*n*: 0-3) (0x0050+0x4**n*)



PCNT_CNT_THR_ZERO_MODE_U*n* Represents the pulse counter status of PCNT_U*n* corresponding to 0.

0: pulse counter decreases from positive to 0

1: pulse counter increases from negative to 0

2: pulse counter is negative

3: pulse counter is positive

(RO)

PCNT_CNT_THR_THRES1_LAT_U*n* Represents whether the current value of PCNT_U*n* equals to thres1 when thres1 comparator is enabled.

0: others

1: the current pulse counter equals to thres1 and thres1 event is valid

(RO)

PCNT_CNT_THR_THRES0_LAT_U*n* Represents whether the current value of PCNT_U*n* equals to thres0 when thres0 comparator is enabled.

0: others

1: the current pulse counter equals to thres0 and thres0 event is valid

(RO)

PCNT_CNT_THR_L_LIM_LAT_U*n* Represents whether the current value of PCNT_U*n* is equal to the low limit threshold when the low limit comparator is enabled.

0: others

1: the current pulse counter equals to thr_l_lim and low limit event is valid.

(RO)

PCNT_CNT_THR_H_LIM_LAT_U*n* Represents whether the current value of PCNT_U*n* is equal to the high limit threshold when the high limit comparator is enabled.

0: others

1: the current pulse counter equals to thr_h_lim and high limit event is valid.

(RO)

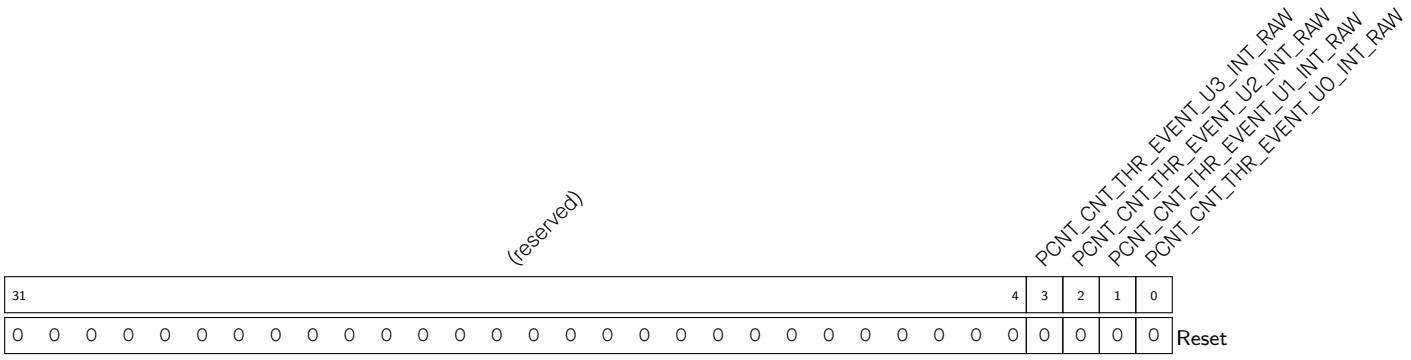
PCNT_CNT_THR_ZERO_LAT_U*n* Represents whether the current value of PCNT_U*n* is 0 when the over-zero comparator is enabled.

0: others

1: the current pulse counter equals to 0 and zero threshold event is valid.

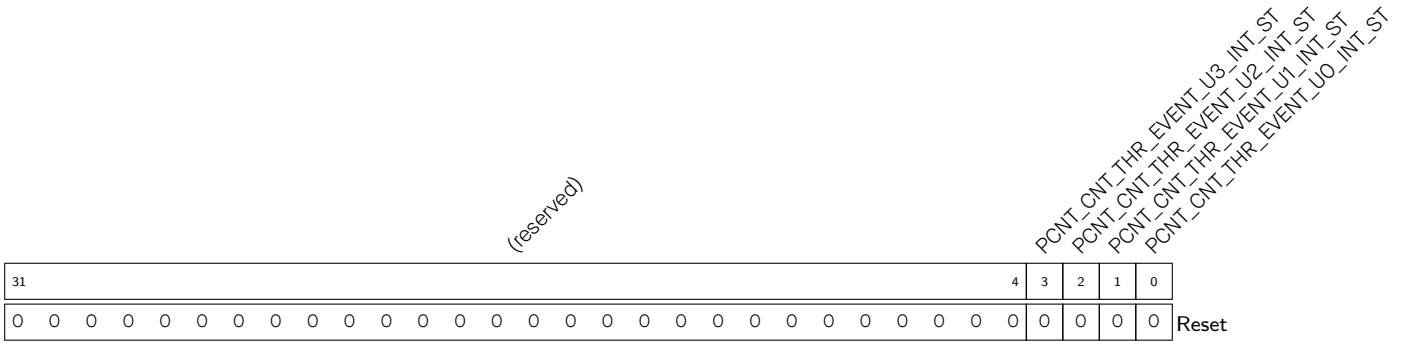
(RO)

Register 51.7. PCNT_INT_RAW_REG (0x0040)



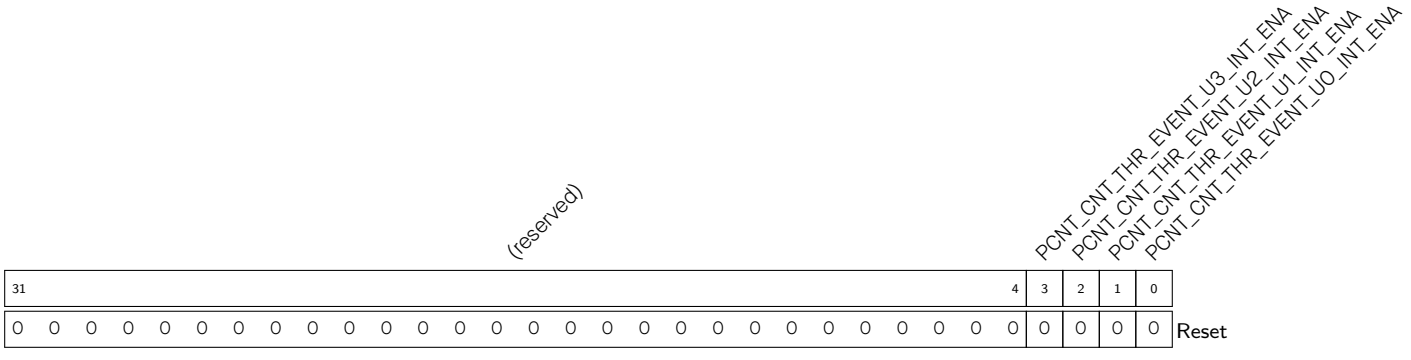
PCNT_CNT_THR_EVENT_U*n*_INT_RAW The raw interrupt status of the PCNT_CNT_THR_EVENT_U*n*_INT interrupt. (RO)

Register 51.8. PCNT_INT_ST_REG (0x0044)



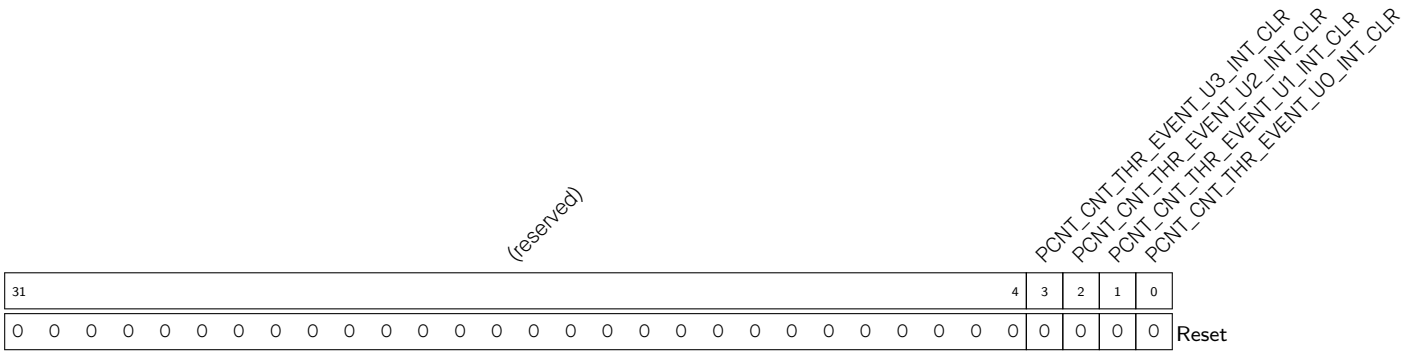
PCNT_CNT_THR_EVENT_U*n*_INT_ST The masked interrupt status of the PCNT_CNT_THR_EVENT_U*n*_INT interrupt. (RO)

Register 51.9. PCNT_INT_ENA_REG (0x0048)



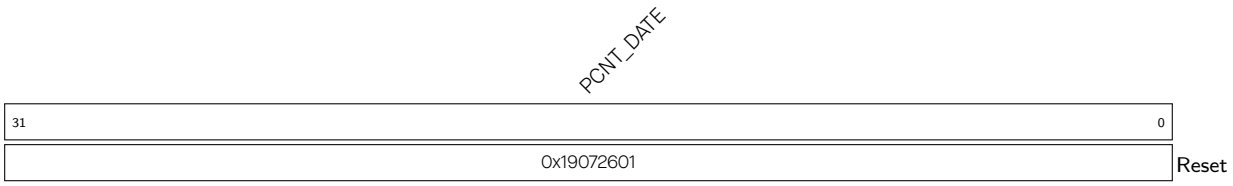
PCNT_CNT_THR_EVENT_U*n*_INT_ENA Write 1 to enable the PCNT_CNT_THR_EVENT_U*n*_INT interrupt. (R/W)

Register 51.10. PCNT_INT_CLR_REG (0x004C)



PCNT_CNT_THR_EVENT_U*n*_INT_CLR Write 1 to clear the PCNT_CNT_THR_EVENT_U*n*_INT interrupt. (WO)

Register 51.11. PCNT_DATE_REG (0x00FC)



PCNT_DATE Version control register. (R/W)

Chapter 52

USB 2.0 High-Speed OTG

52.1 Overview

The ESP32-P4 chip features a USB 2.0 High-Speed On-The-Go peripheral (OTG_HS) with an integrated transceiver. This OTG_HS complies with the USB 2.0 specification, OTG Revision 1.3, and OTG Revision 2.0 specifications. The interface supports USB 2.0 High-Speed mode (480 Mbit/s), Full-Speed mode (12 Mbit/s), and Low-Speed mode (1.5 Mbit/s).

- When OTG_HS operates in High-Speed or Full-Speed modes, it can be configured as either a Host or a Device.
- When OTG_HS operates in Low-Speed mode, it can only be configured as a Host.

52.2 Glossary

The following abbreviations and terms are used in this chapter.

Host	The host computer system where the USB Host Controller is installed. This includes the host hardware platform (CPU, bus, etc.) and the operating system in use.
Device	A logical or physical entity that performs a function. The actual entity described depends on the context of the reference. At the lowest level, device may refer to a single hardware component, as in a memory device. At a higher level, it may refer to a collection of hardware components that perform a particular function, such as a USB interface device. At an even higher level, device may refer to the function performed by an entity attached to the USB; for example, a data/FAX modem device. When used as a non-specific reference, a USB device is either a hub or a function.
High-speed	USB operation at 480 Mbit/s.
Full-speed	USB operation at 12 Mbit/s.
Low-speed	USB operation at 1.5 Mbit/s.
Endpoint	A uniquely addressable portion of a USB device that is the source or sink of information in a communication flow between the host and device.
Scatter/Gather DMA mode	Accesses discontinuous memory areas via DMA.
Buffer DMA mode	Accesses contiguous memory areas via DMA.
Slave mode	Accesses memory area through the CPU.
RX FIFO	Stores the received data.

TX FIFO	Stores the data to be transmitted.
Periodic FIFO	Stores Isochronous transfer data and Interrupt transfer data to be sent.
Non-periodic FIFO	Stores Control transfer data and Bulk transfer data to be sent.

52.3 Features

52.3.1 General Features

- USB 2.0 specification, OTG Revision 1.3 and OTG Revision 2.0 specifications
- High-Speed, Full-Speed, and Low-Speed data rates
- As a host and a device in High-Speed mode and Full-Speed mode
- Dynamic FIFO (DFIFO) sizing, each device EP/host channel can dynamically allocate a maximum of 4 KB FIFO.
- Multiple modes of memory access
  - Scatter/Gather DMA mode
  - Buffer DMA mode
  - Slave Mode
- Integrated UTMI High-Speed transceiver

52.3.2 Device Mode Features

- Endpoint 0 always present, bi-directional, consisting of EP0 IN and EP0 OUT
- 15 additional endpoints 1~15, configurable as IN or OUT
- Maximum of eight IN endpoints concurrently active at any time, including EP0 IN
- All OUT endpoints share a single RX FIFO
- Each IN endpoint has a dedicated TX FIFO

52.3.3 Host Mode Features

- 16 host channels
- RX FIFO: shared by all periodic and non-periodic transactions
- Two TX FIFO:
  - One shared by all non-periodic transactions
  - One shared by all periodic transactions
- All of the above FIFOs share a 4 KB RAM.
- The size of each FIFO is configurable, with a maximum of 4 KB.

52.4 Functional Description

52.4.1 Controller Core and Interfaces

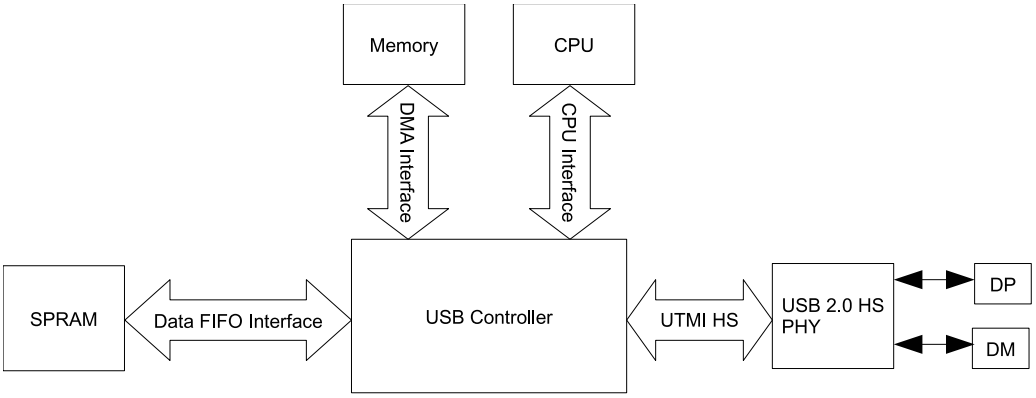


Figure 52.4-1. OTG_HS System Architecture

OTG_HS uses the same controller core as OTG_FS, i.e., USB Controller Core. The controller core has the following interfaces, see Figure 52.4-1:

- **CPU Interface**
  Provides the CPU with read/write access to the controller core's various registers and FIFOs. This interface is internally implemented as an AHB slave interface. The way to access the FIFOs through the CPU interface is called Slave mode.
- **DMA Interface**
  Provides the controller core's internal DMA with read/write access to system memory, e.g., fetching and writing data payloads in DMA mode. This interface is internally implemented as an AHB master interface.
- **USB 2.0 Interface (USB 2.0 HS PHY)**
  This interface is used to connect the controller core to a USB 2.0 UTMI serial transceiver. With this serial transceiver, OTG_HS is able to support High-Speed data rate and Full-Speed data rate.
- **Data FIFO Interface**
  The multiple FIFOs used by the controller core are not actually located within the controller core itself, but on the Single-Port RAM (SPRAM). FIFOs are dynamically sized, thus are allocated at run-time in the SPRAM. When the CPU, DMA, or the controller core attempts to read/write to FIFOs, those accesses are routed through the data FIFO RAM interface.

52.4.2 Memory Layout

Figure 52.4-2 illustrates the memory layout of the registers which are used to configure and control the USB Controller Core.

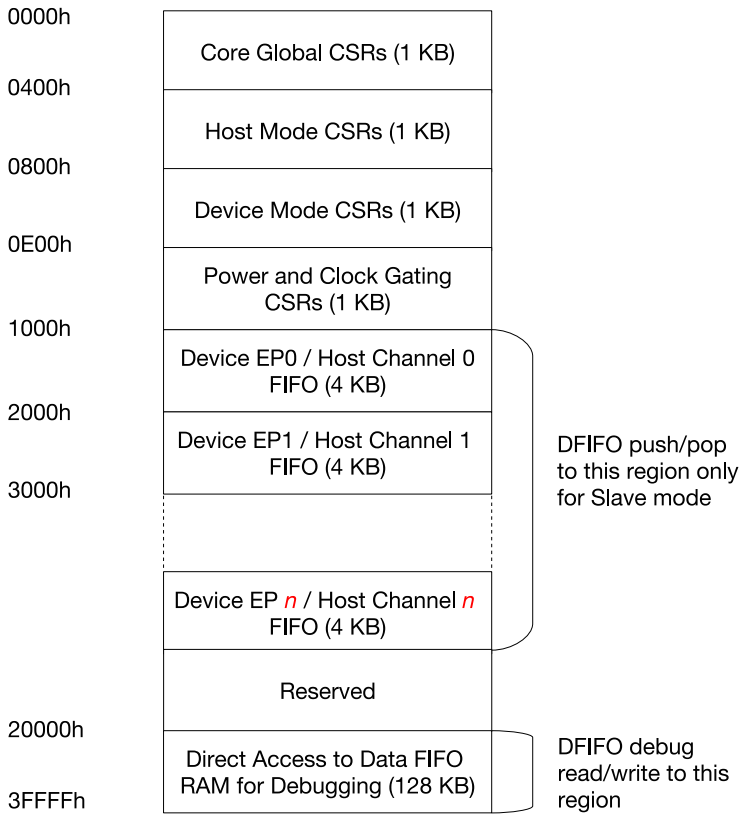


Figure 52.4-2. OTG_HS Register Layout

52.4.2.1 Control & Status Registers (CSRs)

- **Core Global CSRs**
  These registers are responsible for the configuration/control/status of the global features of OTG_HS, i.e., features which are common to both Host and Device modes. These features include OTG control and system-level interrupts. Software can access these registers whilst in Host or Device modes.
- **Host Mode CSRs**
  These registers are responsible for the configuration/control/status in Host mode, thus should only be accessed in Host mode. Each channel has its own set of registers within the Host mode CSRs.
- **Device Mode CSRs**
  These registers are responsible for the configuration/control/status in Device mode, thus should only be accessed in Device mode. Each endpoint has its own set of registers within the Device mode CSRs.
- **Power and Clock Gating Register**
  A single register controls power-down and gates various clocks.

52.4.2.2 FIFO Access

The OTG_HS makes use of multiple FIFOs to buffer the data payloads to be transmitted or to buffer the received data payloads. The number and type of FIFOs are dependent on Host or Device mode, and the number of channels or endpoints used, see Section 52.4.3. There are two ways to access the FIFOs: DMA mode and Slave mode. In Slave mode, the CPU will need to access these FIFOs by reading and writing to either the DFIFO push/pop regions or the DFIFO read/write debug region. FIFO access is governed by the

following rules:

- Read access to any address in any one of the 4 KB push/pop regions will result in a pop from the shared RX FIFO.
- Write access to a particular 4 KB push/pop region will result in a push to the corresponding endpoint or channel's TX FIFO given that the endpoint is an IN endpoint, or the channel is an OUT channel.
  - In Device mode, data is pushed to the corresponding IN endpoint's dedicated TX FIFO.
  - In Host mode, data is pushed to the non-periodic TX FIFO or the periodic TX FIFO depending on whether the channel is a non-periodic channel, or a periodic channel.
- Access to the 128 KB read/write region will result in direct read/write instead of a push/pop. This is generally used for debugging purposes only.

Note that pushing and popping data to and from the FIFOs by the CPU is only required in Slave mode. In DMA mode, the internal DMA will handle all pushing/popping of data to and from the TX and RX FIFOs.

52.4.3 FIFO and Queue Organization

The FIFOs in OTG_HS are primarily used to hold data packet payloads, i.e., the data field of USB Data packets. TX FIFOs are used to store data payloads that will be transmitted by OUT transactions in Host mode or IN transactions in Device mode. RX FIFOs are used to store received data payloads of IN transactions in Host mode or OUT transactions in Device mode. In addition to storing data payloads, RX FIFOs also store a **status entry** for each data payload. Each status entry contains information about a data payload such as channel number, byte count, and validity status. In Slave mode, status entries are also used to indicate various channel events.

The portion of SPRAM that can be used for FIFO allocation has a depth of 1024 and a width of 35 bits (32 data bits plus 3 control bits). The multiple FIFOs used by each channel (in Host mode) or endpoint (in Device mode) are allocated into the SPRAM and can be dynamically sized.

52.4.3.1 Host Mode FIFOs and Queues

The following FIFOs are used in Host mode, see Figure 52.4-3:

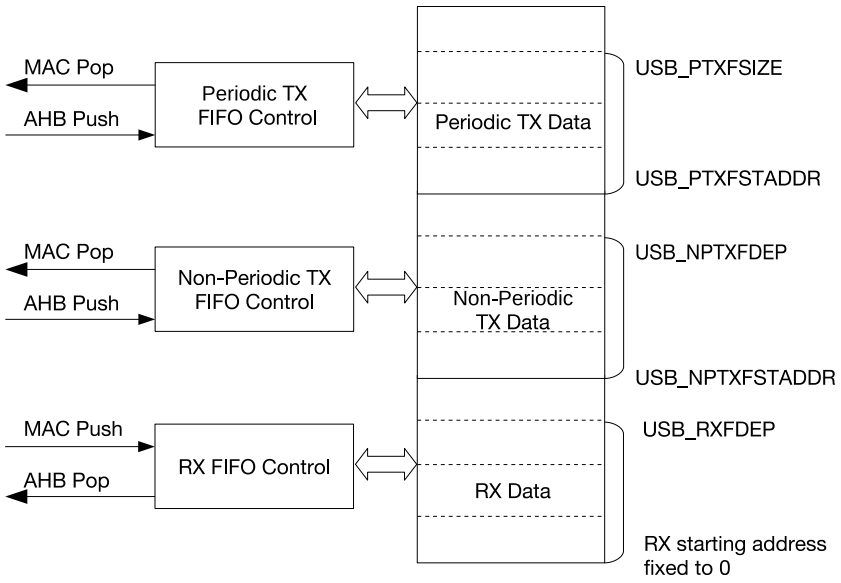


Figure 52.4-3. Host Mode FIFOs

- **Non-periodic TX FIFO**: Stores data payloads of bulk and control OUT transactions for all channels.
- **Periodic TX FIFO**: Stores data payloads of interrupt or isochronous OUT transactions for all channels.
- **RX FIFO**: Stores data payloads of all IN transactions, and status entries that are used to indicate size of data payloads and transaction/channel events such as transfer complete or channel halted.

In addition to FIFOs, Host mode also contains two request queues used to queue up the various transaction request from the multiple channels. Each entry in a request queue holds the IN/OUT channel number along with other information to perform the transaction, such as transaction type. Request queues are also used to queue other types of requests such as a channel halt request.

Unlike FIFOs, request queues are fixed in size and cannot be accessed directly by software. Rather, once a channel is enabled, requests will be automatically written to the request queue by the Host core. The order in which the requests are written into the queue determines the sequence of transactions on the USB.

Host mode contains the following request queues:

- **Non-periodic request queue**: Request queue for all non-periodic Bulk and Control channels. The queue has a depth of four entries.
- **Periodic request queue**: Request queue for all periodic Interrupt and Isochronous channels. The queue has a depth of four entries.

When scheduling transactions, hardware will execute all requests on the periodic request queue first before executing requests on the non-periodic request queue.

52.4.3.2 Device Mode FIFOs

The following FIFOs are used in Device mode, see Figure 52.4-4:

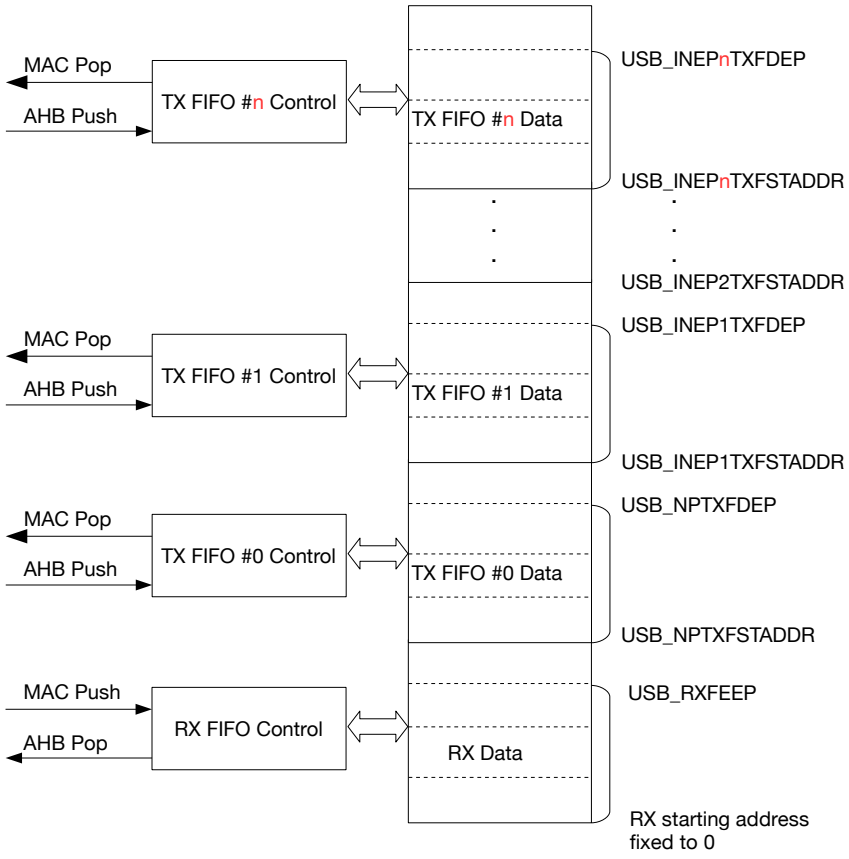


Figure 52.4-4. Device Mode FIFOs

- **RX FIFO**: Stores data payloads received in Data packet, and status entries used to indicate size of those data payloads.
- **Dedicated TX FIFO**: Each active IN endpoint will have a dedicated TX FIFO used to store all IN data payloads of that endpoint, regardless of the transaction type, both periodic and non-periodic IN transactions.

Due to the dedicated FIFOs, Device mode does not use any request queues. Instead, the order of IN transactions are determined by the Host.

52.4.4 Interrupt Hierarchy

OTG_HS provides a single interrupt line which can be routed via the interrupt matrix to one of the CPUs. The interrupt signal can be unmasked by setting USB_GLBLINTRMSK. The OTG_HS interrupt is an OR of all bits in the USB_GINTSTS_REG register, and the bits in USB_GINTSTS_REG can be unmasked by setting the corresponding bits in the USB_GINTMSK_REG register. USB_GINTSTS_REG contains system level interrupts, and also specific bits for Host or Device mode interrupts, and OTG specific interrupts. OTG_HS interrupt sources are organized as Figure 52.4-5 shows.

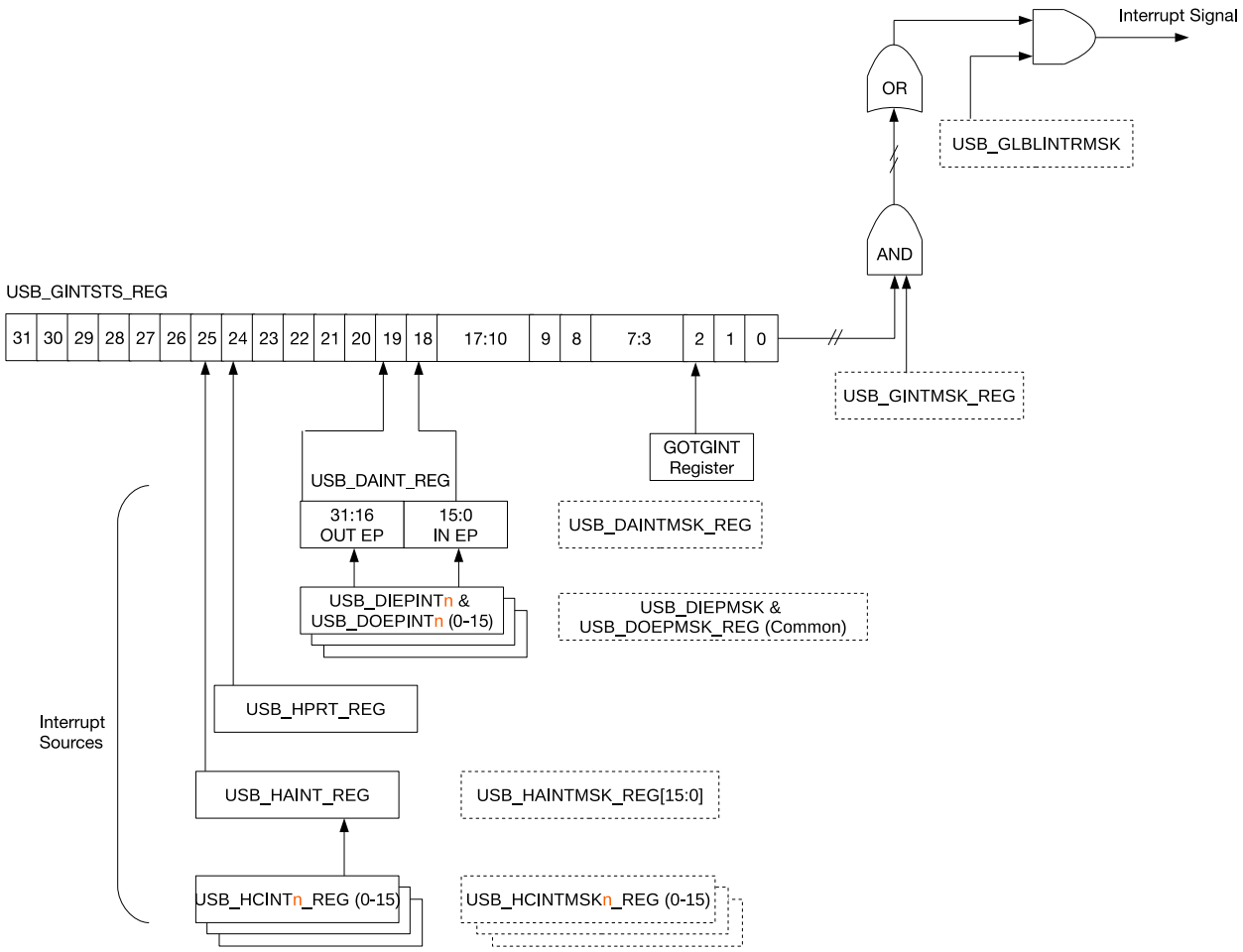


Figure 52.4-5. OTG_HS Interrupt Hierarchy

The following bits of the USB_GINTSTS_REG register indicate an interrupt source lower in the hierarchy:

- **USB_PRTINT** indicates that the Host port has a pending interrupt. The USB_HPRT_REG register indicates the interrupt source.
- **USB_HCHINT** indicates that one or more Host channels have a pending interrupt. Read the USB_HAINT_REG register to determine which channel(s) have a pending interrupt, then read the pending channel's USB_HCINT*n*_REG register to determine the interrupt source.
- **USB_OEPINT** indicates that one or more OUT endpoints have a pending interrupt. Read the USB_DAINT_REG register to determine which OUT endpoint(s) have a pending interrupt, then read the USB_DOEPINT*n*_REG register to determine the interrupt source.
- **USB_IEPINT** indicates that one or more IN endpoints have a pending interrupt. Read the USB_DAINT_REG register to determine which IN endpoint(s) are pending, then read the pending IN endpoint's USB_DIEPINT*n*_REG register to determine the interrupt source.
- **USB_OTGINT** indicates an On-The-Go event has triggered an interrupt. Read the USB_GOTGINT_REG register to determine which OTG event(s) triggered the interrupt.

52.4.5 DMA Modes and Slave Mode

USB OTG supports three ways to access memory: Slave mode, Buffer DMA mode, and Scatter/Gather DMA mode.

52.4.5.1 Slave Mode

In Slave mode, all data payloads must be pushed/popped to and from the FIFOs by the CPU.

- When transmitting a packet using IN endpoints or OUT channels, the data payload must be pushed into the corresponding endpoint or channel's TX FIFO.
- When receiving a packet, the packet's status entry must first be popped off the RX FIFO by reading USB_GRXSTSP_REG. The status entry should be used to determine the length of the packet's payload in bytes. The corresponding number of bytes must then be manually popped off the RX FIFO by reading from the RX FIFO's memory region.

52.4.5.2 Buffer DMA Mode

Buffer mode is similar to Slave mode but utilizes the internal DMA to push and pop data payloads to the FIFOs.

- When transmitting a packet using IN endpoints or OUT channels, the data payload's address in memory should be written to the USB_HCDMA*n*_REG (in Host mode) or USB_DIEPDMA*n*_REG (in Device mode) registers. When the endpoint or channel is enabled, the internal DMA will push the data payload from memory into the TX FIFO of the channel or endpoint.
- When receiving a packet using OUT endpoints or IN channels, the address of an empty buffer in memory should be written to the USB_HCDMA*n*_REG (in Host mode) or USB_DOEPDMA*n*_REG (in Device mode) registers. When the endpoint or channel is enabled, the internal DMA will pop the data payload from RX FIFO into the buffer.

52.4.5.3 Scatter/Gather DMA Mode

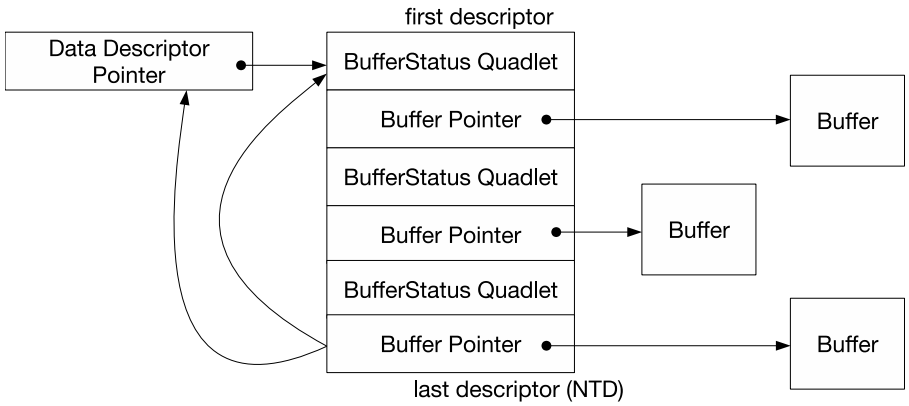


Figure 52.4-6. Scatter/Gather DMA Descriptor List

In Scatter/Gather DMA mode, buffers containing data payloads can be scattered throughout memory. Each endpoint or channel will have a contiguous DMA descriptor list, where each descriptor contains a 32-bit

pointer to the data payload or buffer and a 32-bit buffer descriptor (BufferStatus Quadlet). The data payloads and buffers can correspond to a single transaction (i.e., < 1 MPS bytes) or an entire transfer (> 1 MPS bytes). MPS: maximum packet size. The list is implemented as a ring buffer meaning that the DMA will return to the first entry when it encounters the last entry on the list.

- When transmitting a transfer/transaction using IN endpoints or OUT channels, the DMA will gather the data payloads from the multiple buffers and push them into a TX FIFO.
- When receiving a transfer/transaction using OUT endpoints or IN channels, the DMA will pop the received data payloads from the RX FIFO and scatter them to the multiple buffers pointed to by the DMA list entries.

52.4.6 Transaction and Transfer Level Operation

In either Host or Device mode, communication can operate either at the transaction level or the transfer level.

52.4.6.1 Transaction and Transfer Level Operation in DMA Mode

When operating at the transfer level in DMA Host mode, software is interrupted only when a channel has been halted. Channels are halted when their programmed transfer size has completed successfully, has received a STALL, or if there are excessive transaction errors, e.g., 3 consecutive transaction errors. In DMA Device mode, all errors are handled by the controller core itself.

When operating at the transaction level in DMA mode, the transfer size is set to the size of one data packet, either a maximum packet size or a short packet size.

52.4.6.2 Transaction and Transfer Level Operation in Slave Mode

When operating at the transaction level in Slave Mode, transfers are handled one transaction at a time. Each data payload should correspond to a single data packet, and software must determine whether a retry of the transaction is necessary based on the handshake response received on the USB, e.g., ACK or NAK.

The following table describes transaction level operation in Slave mode for both IN and OUT transactions.

Table 52.4-1. IN and OUT Transactions in Slave Mode

Host Mode	Device Mode
OUT Transactions	

Host Mode	Device Mode
1. Software specifies the size of the data packet and the number of data packets (1 data packet) in the USB_HCTSIZ*n*_REG register, enables the channel, then copies the packet's data payload into the TX FIFO. 2. When the last DWORD of the data payload has been pushed, the controller core will automatically write a request into the appropriate request queue. 3. If the transaction is successful, the USB_XFERCOMPL interrupt will be generated. If the transaction is unsuccessful, an error interrupt, e.g., USB_H_NACK*n* will occur.	1. Software specifies the expected size of the data packet (1 MPS) and the number of data packets (1 data packet) in the USB_DOEPTSIZ*n*_REG register. Once the endpoint is enabled, it will wait for the host to transmit a packet to it. 2. The received packet will be pushed into the RX FIFO along with a packet status entry. 3. If the transaction is unsuccessful, e.g., due to a full RX FIFO, the endpoint will automatically NAK the incoming packet.
IN Transactions	
1. Software specifies the expected size of the data packet and the number of packets (1 data packet) in the USB_HCTSIZ*n*_REG register, then enables the channel. 2. The controller core will automatically write a request into the appropriate request queue. 3. If the transaction is successful, the received data along with a status entry should be written to the RX FIFO. If the transaction is unsuccessful, an error interrupt, e.g., USB_H_NACK*n* will occur.	1. Software specifies the size of the data packet and the number of data packets (1 data packet) in the USB_DIEPTSIZ*n*_REG register. Once the endpoint is enabled, it will wait for the host to read the packet. 2. When the packet has been transmitted, the USB_XFERCOMPL interrupt will be generated.

When operating at the transfer level in Slave mode, one or more transaction-level operations can be pipelined thus being analogous to transfer level operation in DMA mode. Within pipelined transactions, multiple packets of the same transfer can be read/written from the FIFOs in single instance, thus preventing the need for interrupting the software on a per-packet basis.

Operating on a transfer level in Slave mode is similar to operating on the transaction-level, except the transfer size and packet count for each transfer in the USB_HCTSIZ*n*_REG or USB_DOEPTSIZ*n*_REG register will need to be set to reflect the entire transfer. After the channel or endpoint is enabled, multiple data packets worth of payloads should be written to or read from the TX or RX FIFOs respectively (given that there is enough space or enough data).

52.4.7 OTG

Internal ID/VBUS is left floating in OTG_HS, thus SRP/HNP is not supported. But OTG_HS still supports Dual-Role-Devices (DRD), i.e, it can be used both as a host and a device. Software can force setting this

controller to work as host or device by configuring ForceDevMode or ForceHstMode in register USB_GUSBCFG, regardless of the value of ID pin.

52.5 Registers

The catalog and comprehensive specifications of USB OTG registers are subject to a Non-Disclosure Agreement (NDA) as mandated by the IP provider. To obtain support information for a particular register, please contact Espressif Technical Support Team via Technical Inquires.

Chapter 53

USB 2.0 Full-Speed OTG

53.1 Overview

The ESP32-P4 features a USB 2.0 Full-Speed On-The-Go peripheral (henceforth referred to as OTG_FS) along with integrated transceivers. This OTG_FS conforms to USB 2.0 specification, OTG Revision 1.3, and OTG Revision 2.0 specifications, OTG_FS can operate as either a USB Host or Device and supports 12 Mbit/s full-speed (FS) and 1.5 Mbit/s low-speed (LS) data rates of the USB 2.0 specification. The Host Negotiation Protocol (HNP) and the Session Request Protocol (SRP) are also supported.

53.2 Glossary

The following abbreviations and terms are used in this chapter.

Host The host computer system where the USB Host Controller is installed. This includes the host hardware platform (CPU, bus, etc.) and the operating system in use.

Device A logical or physical entity that performs a function. The actual entity described depends on the context of the reference. At the lowest level, device may refer to a single hardware component, as in a memory device. At a higher level, it may refer to a collection of hardware components that perform a particular function, such as a USB interface device. At an even higher level, device may refer to the function performed by an entity attached to the USB; for example, a data/FAX modem device. When used as a non-specific reference, a USB device is either a hub or a function.

Full-speed USB operation at 12 Mbit/s.

Low-speed USB operation at 1.5 Mbit/s.

Host Negotiation Protocol (HNP) Allows the host function to be transferred between two directly connected OTG devices and eliminates the need for a user to switch the cable connections in order to allow a change in control of communications between the devices.

Session Request Protocol (SRP) Allows a B-device to request the A-device to turn on the power supply to the USB interface (VBUS) and start a session.

Endpoint A uniquely addressable portion of a USB device that is the source or sink of information in a communication flow between the host and device.

Scatter/Gather DMA mode Accesses discontinuous memory areas via DMA.

Buffer DMA mode	Accesses contiguous memory areas via DMA.
Slave mode	Accesses memory area through the CPU.
RX FIFO	Stores the received data.
TX FIFO	Stores the data to be transmitted.
Periodic FIFO	Stores Isochronous transfer data and Interrupt transfer data to be sent.
Non-periodic FIFO	Stores Control transfer data and Bulk transfer data to be sent.

53.3 Features

53.3.1 General Features

- USB 2.0 specification, OTG Revision 1.3 and OTG Revision 2.0 specifications
- USB 2.0 full-speed and low-speed data rates
- HNP and SRP as A-device or B-device
- Dynamic FIFO (DFIFO) sizing, maximum to 1 KB
- Multiple modes of memory access
  - Scatter/Gather DMA mode
  - Buffer DMA mode
  - Slave mode
- Two integrated transceivers

53.3.2 Device Mode Features

- Endpoint 0 always present, bi-directional, consisting of EP0 IN and EP0 OUT
- Six additional endpoints 1~6, configurable as IN or OUT
- Maximum of five IN endpoints concurrently active at any time, including EP0 IN
- All OUT endpoints share a single RX FIFO
- Each IN endpoint has a dedicated TX FIFO

53.3.3 Host Mode Features

- Eight host channels
- RX FIFO: shared by all periodic and non-periodic transactions
- Two TX FIFO:
  - One shared by all non-periodic transactions
  - One shared by all periodic transactions
- All of the above FIFOs share a 1 KB RAM.
- The size of each FIFO is configurable, with a maximum of 1 KB.

53.4 Functional Description

53.4.1 Controller Core and Interfaces

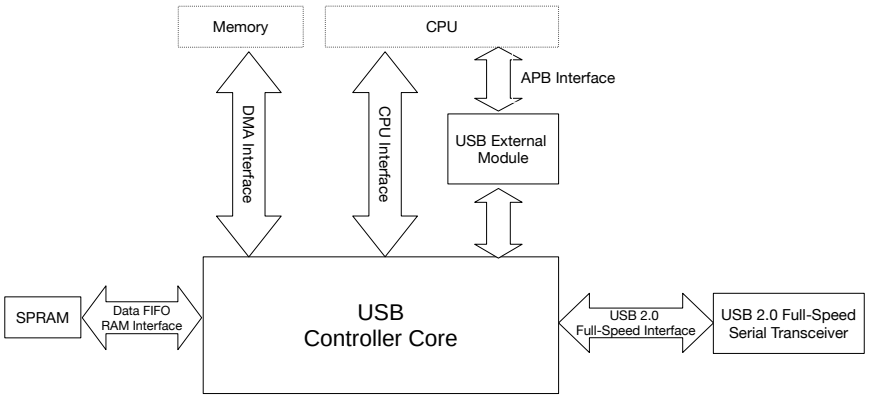


Figure 53.4-1. OTG_FS System Architecture

The core part of the OTG_FS peripheral is the USB Controller Core. The controller core has the following interfaces, see Figure 53.4-1:

- **CPU Interface**
  Provides the CPU with read/write access to the controller core's various registers and FIFOs. This interface is internally implemented as an AHB slave interface. The way to access the FIFOs through the CPU interface is called Slave mode.
- **APB Interface**
  Allows the CPU to control the USB controller core via the USB external controller.
- **DMA Interface**
  Provides the controller core's internal DMA with read/write access to system memory, e.g., fetching and writing data payloads in DMA mode. This interface is internally implemented as an AHB master interface.
- **USB 2.0 Full-Speed Interface**
  ESP32-P4 has two full-speed transceivers. This interface is used to connect the controller core to one of USB 2.0 full-speed serial transceivers. Aside from USB OTG, ESP32-P4 also includes a USB Serial/JTAG controller, see Chapter *54USB Serial/JTAG Controller (USB_SERIAL_JTAG)*. These two USB controllers can utilize the integrated internal transceivers by time-division multiplexing. In the following sections, GPIO24 and GPIO25 are referred to as FS_PHY1, while GPIO26 and GPIO27 as FS_PHY2.

  By default FS_PHY1 connects to USB Serial/JTAG controller and FS_PHY2 to OTG_FS. The connection is configurable via eFuse settings:

  - 0: FS_PHY1 is connected to USB Serial/JTAG controller, while FS_PHY2 to OTG_FS.
  - 1: FS_PHY2 is connected to USB Serial/JTAG controller, while FS_PHY1 to OTG_FS.
- **USB External Controller**
  The USB External Controller is primarily used to control the routing of the USB 2.0 full-speed serial interface to the internal registers, thus allowing the CPU to perform query operations, etc. The External Controller can also enable a power saving mode by gating the controller core's clock, i.e., AHB clock, or

powering down the connected SPRAM clock. Note that this power saving mode is different from the power savings via SRP.

- **Data FIFO RAM Interface**
  The multiple FIFOs used by the controller core are not actually located within the controller core itself, but on the Single-Port RAM (SPRAM). FIFOs are dynamically sized, thus are allocated at run-time in the SPRAM. When the CPU, DMA, or the controller core attempts to read/write to FIFOs, those accesses are routed through the data FIFO RAM interface.

53.4.2 Memory Layout

Figure 53.4-2 illustrates the memory layout of the registers which are used to configure and control the USB Controller Core. Note that USB External Controller uses a separate set of registers called wrap registers.

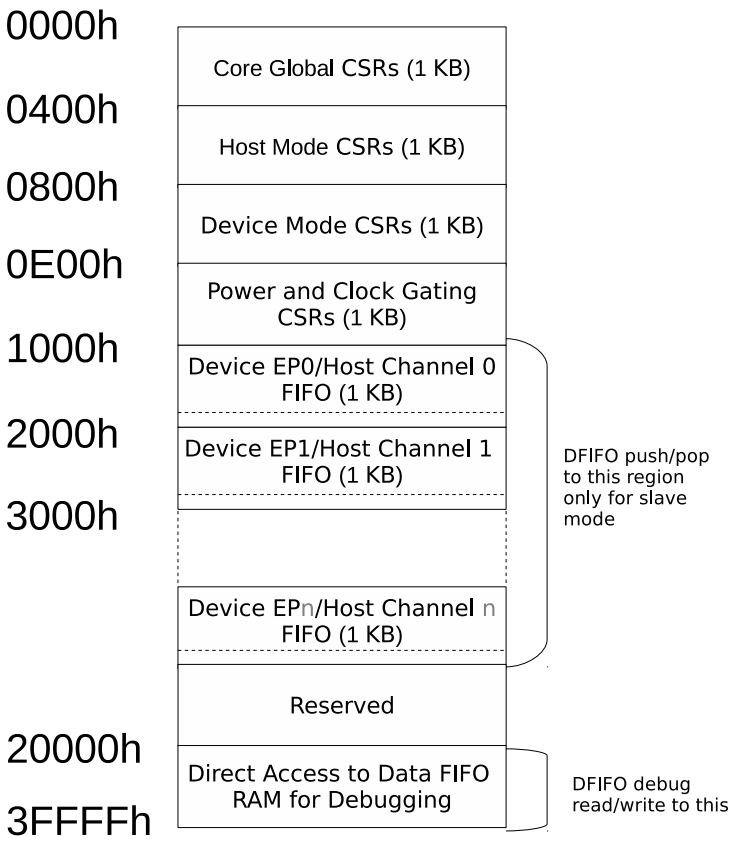


Figure 53.4-2. OTG_FS Register Layout

53.4.2.1 Control & Status Registers (CSRs)

- **Core Global CSRs**
  These registers are responsible for the configuration/control/status of the global features of OTG_FS, i.e., features which are common to both Host and Device modes. These features include:
  - OTG control (HNP, SRP, and A/B-device detection);
  - USB configuration (Host or Device mode selection and PHY selection);
  - and system-level interrupts.

Software can access these registers whilst in Host or Device modes.

- **Host Mode CSRs**
  These registers are responsible for the configuration/control/status in Host mode, thus should only be accessed in Host mode. Each channel has its own set of registers within the Host mode CSRs.
- **Device Mode CSRs**
  These registers are responsible for the configuration/control/status in Device mode, thus should only be accessed in Device mode. Each endpoint has its own set of registers within the Device mode CSRs.
- **Power and Clock Gating Register**
  A single register controls power-down and gates various clocks.

53.4.2.2 FIFO Access

The OTG_FS makes use of multiple FIFOs to buffer the data payloads to be transmitted or to buffer the received data payloads. The number and type of FIFOs are dependent on Host or Device mode, and the number of channels or endpoints used, see Section 53.4.3. There are two ways to access the FIFOs: DMA mode and Slave mode. In Slave mode, the CPU accesses these FIFOs by reading and writing to either the DFIFO push/pop regions or the DFIFO read/write debug region. FIFO access is governed by the following rules:

- Read access to any address in any one of the 1 KB push/pop regions will result in a pop from the shared RX FIFO.
- Write access to a particular 1 KB push/pop region will result in a push to the corresponding endpoint or channel's TX FIFO given that the endpoint is an IN endpoint, or the channel is an OUT channel.
  - In Device mode, data is pushed to the corresponding IN endpoint's dedicated TX FIFO.
  - In Host mode, data is pushed to the non-periodic TX FIFO or the periodic TX FIFO depending on whether the channel is a non-periodic channel, or a periodic channel.
- Access to the 128 KB read/write region will result in direct read/write instead of a push/pop. This is generally used for debugging purposes only.

Note that pushing and popping data to and from the FIFOs by the CPU is only required in Slave mode. In DMA mode, the internal DMA will handle all pushing/popping of data to and from the TX and RX FIFOs.

53.4.3 FIFO and Queue Organization

The FIFOs in OTG_FS are primarily used to hold data packet payloads, i.e., the data field of USB Data packets. TX FIFOs are used to store data payloads that will be transmitted by OUT transactions in Host mode or IN transactions in Device mode. RX FIFOs are used to store received data payloads of IN transactions in Host mode or OUT transactions in Device mode. In addition to storing data payloads, RX FIFOs also store a **status entry** for each data payload. Each status entry contains information about a data payload such as channel number, byte count, and validity status. In Slave mode, status entries are also used to indicate various channel events.

The portion of SPRAM that can be used for FIFO allocation has a depth of 256 and a width of 35 bits (32 data bits plus 3 control bits). The multiple FIFOs used by each channel in Host mode or endpoint in Device mode are allocated into the SPRAM and can be dynamically sized.

53.4.3.1 Host Mode FIFOs and Queues

The following FIFOs are used in Host mode, see Figure 53.4-3:

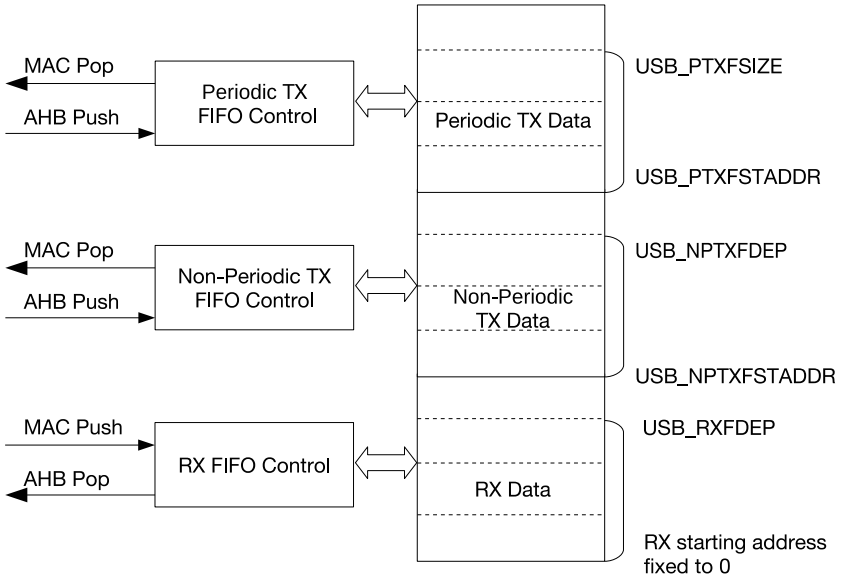


Figure 53.4-3. Host Mode FIFOs

- **Non-periodic TX FIFO**: Stores data payloads of Bulk and Control OUT transactions for all channels.
- **Periodic TX FIFO**: Stores data payloads of Interrupt or Isochronous OUT transactions for all channels.
- **RX FIFO**: Stores data payloads of all IN transactions, and status entries that are used to indicate size of data payloads and transaction/channel events such as transfer complete or channel halted.

In addition to FIFOs, Host mode also contains two request queues used to queue up the various transaction request from the multiple channels. Each entry in a request queue holds the IN/OUT channel number along with other information to perform the transaction, such as transaction type. Request queues are also used to queue other types of requests such as a channel halt request.

Unlike FIFOs, request queues are fixed in size and cannot be accessed directly by software. Rather, once a channel is enabled, requests will be automatically written to the request queue by the Host core. The order in which the requests are written into the queue determines the sequence of transactions on the USB.

Host mode contains the following request queues:

- **Non-periodic request queue**: Request queue for all non-periodic Bulk and Control channels. The queue has a depth of four entries.
- **Periodic request queue**: Request queue for all periodic Interrupt and Isochronous channels. The queue has a depth of eight entries.

When scheduling transactions, hardware will execute all requests on the periodic request queue first before executing requests on the non-periodic request queue.

53.4.3.2 Device Mode FIFOs

The following FIFOs are used in Device mode, see Figure 53.4-4:

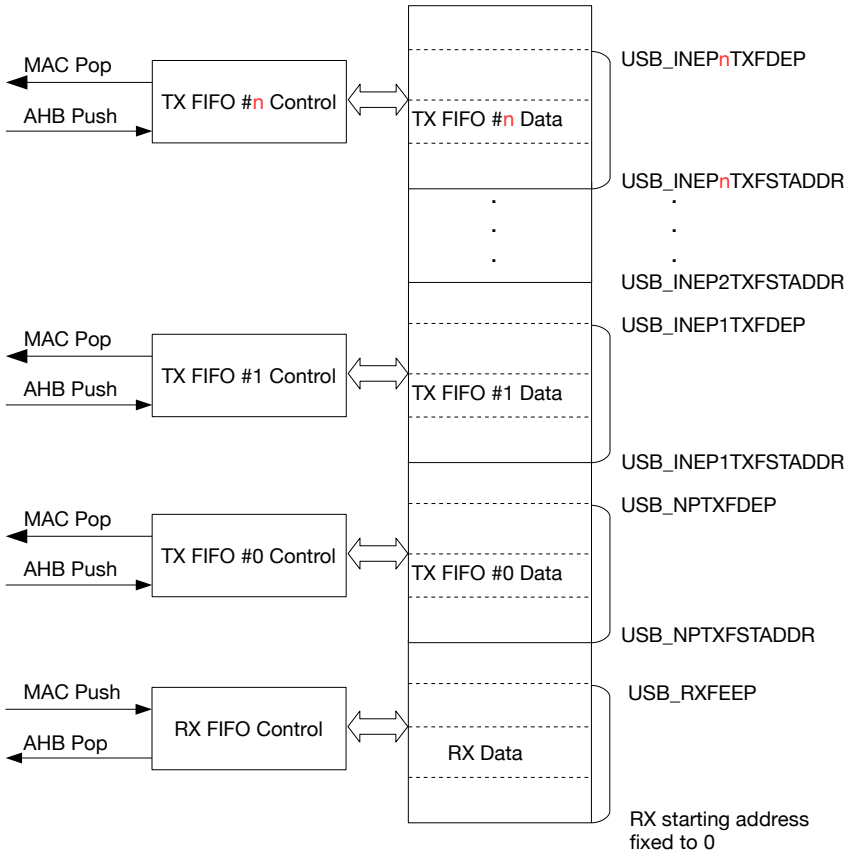


Figure 53.4-4. Device Mode FIFOs

- **RX FIFO**: Stores data payloads received in Data packet, and status entries used to indicate size of those data payloads.
- **Dedicated TX FIFO**: Each active IN endpoint will have a dedicated TX FIFO used to store all IN data payloads of that endpoint, regardless of the transaction type, both periodic and non-periodic IN transactions.

Due to the dedicated FIFOs, Device mode does not use any request queues. Instead, the order of IN transactions are determined by the Host.

53.4.4 Interrupt Hierarchy

OTG_FS provides a single interrupt line which can be routed via the interrupt matrix to one of the CPUs. The interrupt signal can be unmasked by setting USB_GLBLINTRMSK. The OTG_FS interrupt is an OR of all bits in the USB_GINTSTS_REG register, and the bits in USB_GINTSTS_REG can be unmasked by setting the corresponding bits in the USB_GINTMSK_REG register. USB_GINTSTS_REG contains system level interrupts, and also specific bits for Host or Device mode interrupts, and OTG specific interrupts. OTG_FS interrupt sources are organized as Figure 53.4-5 shows.

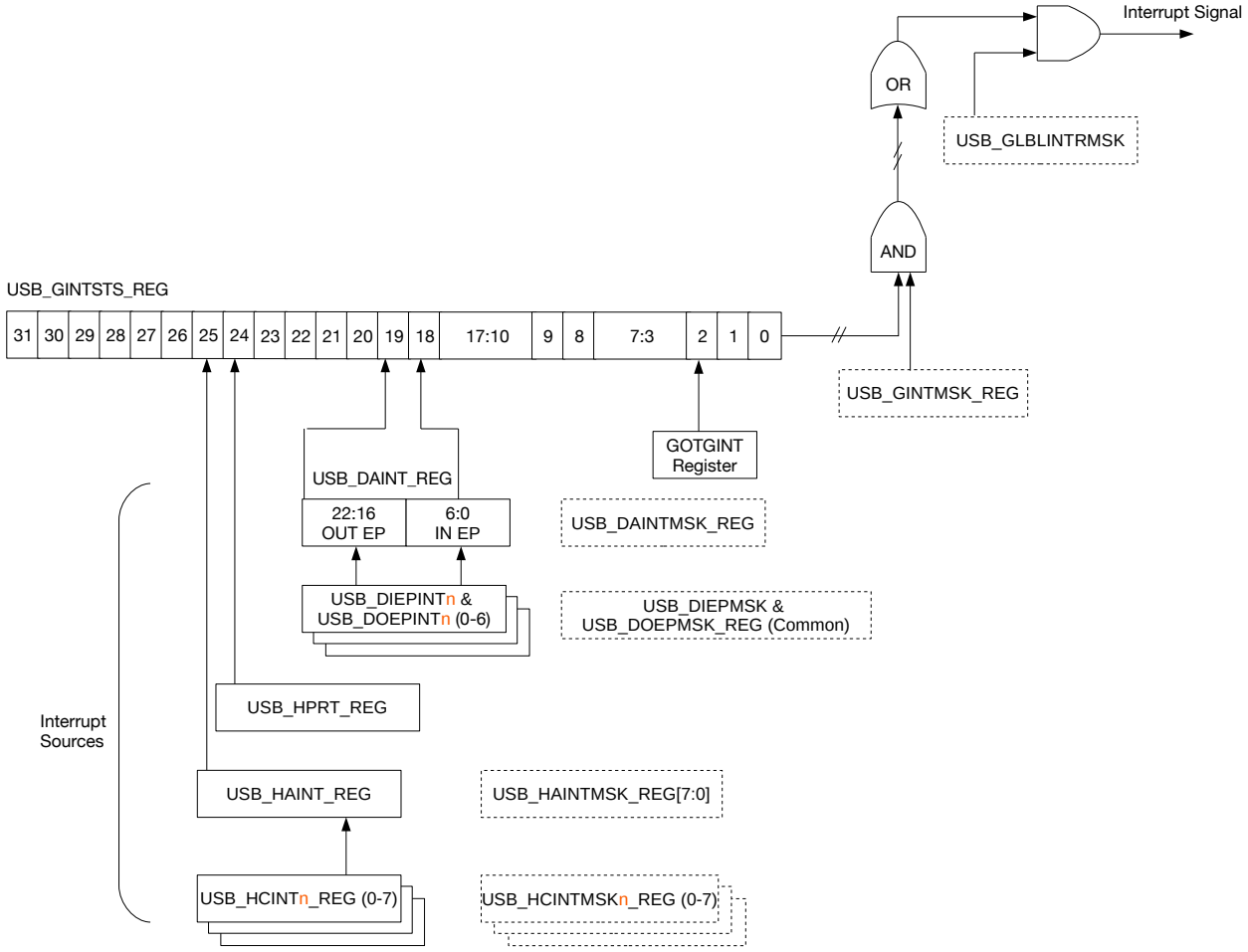


Figure 53.4-5. OTG_FS Interrupt Hierarchy

The following bits of the USB_GINTSTS_REG register indicate an interrupt source lower in the hierarchy:

- **USB_PRTINT** indicates that the Host port has a pending interrupt. The USB_HPRT_REG register indicates the interrupt source.
- **USB_HCHINT** indicates that one or more Host channels have a pending interrupt. Read the USB_HAINT_REG register to determine which channel(s) have a pending interrupt, then read the pending channel's USB_HCINT*n*_REG register to determine the interrupt source.
- **USB_OEPINT** indicates that one or more OUT endpoints have a pending interrupt. Read the USB_DAINT_REG register to determine which OUT endpoint(s) have a pending interrupt, then read the USB_DOEPINT*n*_REG register to determine the interrupt source.
- **USB_IEPINT** indicates that one or more IN endpoints have a pending interrupt. Read the USB_DAINT_REG register to determine which IN endpoint(s) are pending, then read the pending IN endpoint's USB_DIEPINT*n*_REG register to determine the interrupt source.
- **USB_OTGINT** indicates an On-The-Go event has triggered an interrupt. Read the USB_GOTGINT_REG register to determine which OTG event(s) triggered the interrupt.

53.4.5 Slave Mode and DMA Modes

OTG_FS supports three ways to access memory: Slave mode, Buffer DMA mode, and Scatter/Gather DMA mode.

53.4.5.1 Slave Mode

In Slave mode, all data payloads must be pushed/popped to and from the FIFOs by the CPU.

- When transmitting a packet using IN endpoints or OUT channels, the data payload must be pushed into the corresponding endpoint or channel's TX FIFO.
- When receiving a packet, the packet's status entry must first be popped off the RX FIFO by reading USB_GRXSTSP_REG. The status entry should be used to determine the length of the packet's payload (in bytes). The corresponding number of bytes must then be manually popped off the RX FIFO by CPU reading from the RX FIFO's memory region.

53.4.5.2 Buffer DMA Mode

Buffer DMA mode is similar to Slave mode but utilizes the internal DMA to push and pop data payloads to the FIFOs.

- When transmitting a packet using IN endpoints or OUT channels, the data payload's address in memory should be written to USB_HCDMA*n*_REG in Host mode or USB_DIEPDMA*n*_REG in Device mode. When the endpoint or channel is enabled, the internal DMA will push the data payload from memory into the TX FIFO of the channel or endpoint.
- When receiving a packet using OUT endpoints or IN channels, the address of an empty buffer in memory should be written to USB_HCDMA*n*_REG in Host mode or USB_DOEPDMA*n*_REG in Device mode. When the endpoint or channel is enabled, the internal DMA will pop the data payload from RX FIFO into the buffer.

53.4.5.3 Scatter/Gather DMA Mode

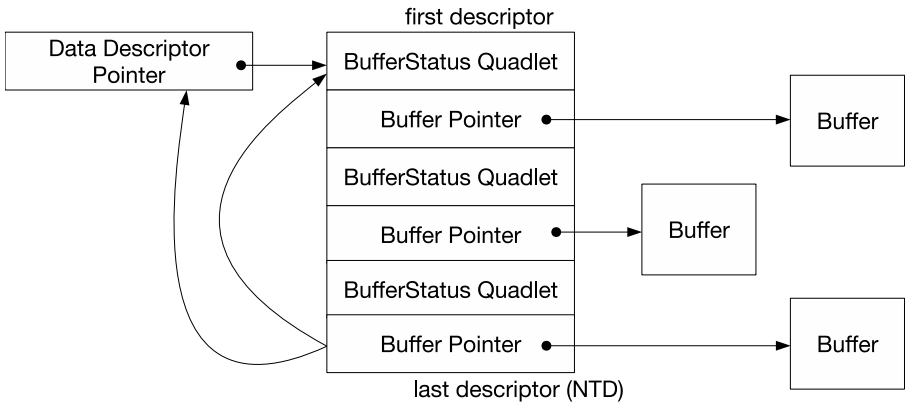


Figure 53.4-6. Scatter/Gather DMA Descriptor List

In Scatter/Gather DMA mode, buffers containing data payloads can be scattered throughout memory. Each endpoint or channel will have a contiguous DMA descriptor list, where each descriptor contains a 32-bit

pointer to the data payload or buffer and a 32-bit buffer descriptor (BufferStatus Quadlet). The data payloads and buffers can correspond to a single transaction (i.e., < 1 MPS bytes) or an entire transfer (> 1 MPS bytes). MPS: maximum packet size. The list is implemented as a ring buffer meaning that the DMA will return to the first entry when it encounters the last entry on the list.

- When transmitting a transfer/transaction using IN endpoints or OUT channels, the DMA will gather the data payloads from the multiple buffers and push them into a TX FIFO.
- When receiving a transfer/transaction using OUT endpoints or IN channels, the DMA will pop the received data payloads from the RX FIFO and scatter them to the multiple buffers pointed to by the DMA list entries.

53.4.6 Transaction and Transfer Level Operation

In either Host or Device mode, communication can operate either at the transaction level or the transfer level.

53.4.6.1 Transaction and Transfer Level Operation in DMA Mode

When operating at the transaction level in DMA mode, the transfer size is set to the size of one data packet, either a maximum packet size or a short packet size.

When operating at the transfer level in DMA Host mode, software is interrupted only when a channel has been halted. Channels are halted when their programmed transfer size has completed successfully, has received a STALL, or if there are excessive transaction errors, e.g., 3 consecutive transaction errors. In DMA Device mode, all errors are handled by the controller core itself.

53.4.6.2 Transaction and Transfer Level Operation in Slave Mode

When operating at the transaction level in Slave Mode, transfers are handled one transaction at a time. Each data payload should correspond to a single data packet, and software must determine whether a retry of the transaction is necessary based on the handshake response received on the USB, e.g., ACK or NAK.

The following table describes transaction level operation in Slave mode for both IN and OUT transactions.

Table 53.4-1. IN and OUT Transactions in Slave Mode

Host Mode	Device Mode
OUT Transactions	

Host Mode	Device Mode
1. Software specifies the size of the data packet and the number of data packets (1 data packet) in the USB_HCTSIZ*n*_REG register, enables the channel, then copies the packet's data payload into the TX FIFO. 2. When the last DWORD of the data payload has been pushed, the controller core will automatically write a request into the appropriate request queue. 3. If the transaction is successful, the USB_XFERCOMPL interrupt will be generated. If the transaction is unsuccessful, an error interrupt, e.g. USB_H_NACK*n* will occur.	1. Software specifies the expected size of the data packet (1 MPS) and the number of data packets (1 data packet) in the USB_DOEPTSIZ*n*_REG. Once the endpoint is enabled, it will wait for the host to transmit a packet to it. 2. The received packet will be pushed into the RX FIFO along with a packet status entry. 3. If the transaction is unsuccessful, e.g., due to a full RX FIFO, the endpoint will automatically NAK the incoming packet.
IN Transactions	
1. Software specifies the expected size of the data packet and the number of packets (1 data packet) in the USB_HCTSIZ*n*_REG register, then enables the channel. 2. The controller core will automatically write a request into the appropriate request queue. 3. If the transaction is successful, the received data along with a status entry should be written to the RX FIFO. If the transaction is unsuccessful, an error interrupt, e.g., USB_H_NACK*n* will occur.	1. Software specifies the size of the data packet and the number of data packets (1 data packet) in the USB_DIEPTSIZ*n*_REG register. Once the endpoint is enabled, it will wait for the host to read the packet. 2. When the packet has been transmitted, the USB_XFERCOMPL interrupt will be generated.

When operating at the transfer level in Slave mode, one or more transaction-level operations can be pipelined thus being analogous to transfer level operation in DMA mode. Within pipelined transactions, multiple packets of the same transfer can be read/written from the FIFOs in single instance, thus preventing the need for interrupting the software on a per-packet basis.

Operating on a transfer level in Slave mode is similar to operating on the transaction-level, except the transfer size and packet count for each transfer in the USB_HCTSIZ*n*_REG or USB_DOEPTSIZ*n*_REG register will need to be set to reflect the entire transfer. After the channel or endpoint is enabled, multiple data packets worth of payloads should be written to or read from the TX or RX FIFOs respectively (given that there is enough space or enough data).

53.5 OTG

USB OTG allows OTG devices to act in the USB Host role or the USB Device role. Thus, OTG devices will typically have a Mini-AB or Micro-AB receptacle so that it can receive an A-plug or B-plug. OTG devices will

become either an A-device or a B-device depending on whether an A-plug or a B-plug is connected.

- A-device defaults to the Host role (A-Host) whilst B-device defaults to the Device role (B-Peripheral).
- A-device and B-device may exchange roles by using the Host Negotiation Protocol (HNP), thus becoming A-peripheral and B-Host.
- A-device can turn off Vbus to save power. B-device can then wake up the A-device by requesting it to turn on Vbus and start a new session. This mechanism is called session request protocol (SRP).
- A-device always powers Vbus even if it is an A-peripheral.

OTG devices are able to determine whether they are connected to an A plug or a B plug using the ID pin of the plugs. The ID pin in A-plugs are pulled to ground whilst B-plugs have the ID pin left floating.

53.5.1 OTG Interface

The OTG_FS supports both the Session Request Protocol (SRP) and Host Negotiation Protocol (HNP) of the OTG Revision 1.3 specification. The OTG_FS controller core interfaces with the internal transceiver using the UTMI+ OTG interface. The UTMI+ OTG interface allows the controller core to manipulate the transceiver for OTG purposes, e.g., enabling/disabling pull-ups and pull-downs in HNP, and also allows the transceiver to indicate OTG related events. If an external transceiver is used instead, the UTMI+ OTG interface signals will be routed to the ESP32-P4's GPIOs instead through GPIO Matrix. Please refer to Chapter 10 *GPIO Matrix and IO MUX*. The UTMI+ OTG interface signals are described in Table 53.5-1.

Table 53.5-1. UTMI+ OTG Interface

Signal Name	I/O	Description
usb_otg_iddig_in	I	Mini A/B Plug Indicator. Indicates whether the connected plug is mini-A or mini-B. Valid only when usb_otg_idpullup is sampled asserted. 0: Mini-A connected 1: Mini-B connected
usb_otg_avalid_in	I	A-Peripheral Session Valid. Indicates if the voltage Vbus is at a valid level for an A-peripheral session. The comparator thresholds are: 0: Vbus <0.8 V 1: Vbus = 0.2 V to 2.0 V
usb_otg_bvalid_in	I	B-Peripheral Session Valid. Indicates if the voltage Vbus is at a valid level for a B-peripheral session. The comparator thresholds are: 0: Vbus <0.8 V 1: Vbus = 0.8 V to 4 V
usb_otg_vbusvalid_in	I	Vbus Valid. Indicates if the voltage Vbus is valid for A/B-device/peripheral operation. The comparator thresholds are: 0: Vbus <4.4 V 1: Vbus >4.75 V
usb_srp_sessend_in	I	B-device Session End. Indicates if the voltage Vbus is below the B-device Session End threshold. The comparator thresholds are: 0: Vbus >0.8 V 1: Vbus <0.2 V

Signal Name	I/O	Description
usb_otg_idpullup	O	Analog ID input Sample Enable. Enables sampling the analog ID line. 0: ID pin sampling disabled 1: ID pin sampling enabled
usb_otg_dppulldown	O	D+ Pull-down Resistor Enable. Enables the 15 kΩ pull-down resistor on the D+ line.
usb_otg_dmpulldown	O	D- Pull-down Resistor Enable. Enables the 15 kΩ pull-down resistor on the D- line.
usb_otg_drvvbus	O	Drive Vbus. Enables driving Vbus to 5 V. 0: Do not drive Vbus 1: Drive Vbus
usb_srp_chrgvbus	O	Vbus Input Charge Enable. Directs the PHY to charge Vbus. 0: Do not charge Vbus through a resistor 1: Charge Vbus through a resistor (must be active for at least 30 ms)
usb_srp_dischrgvbus	O	Vbus Input Discharge Enable. Directs the PHY to discharge Vbus. 0: Do not discharge Vbus through a resistor. 1: Discharge Vbus through a resistor (must be active for at least 50 ms).

53.5.2 ID Pin Detection

Bit USB_CONIDSTS in register USB_GOTGCTL_REG indicates whether the OTG controller is an A-device (1'b0) or a B-device (1'b1). The USB_CONIDSTSCHNG interrupt will trigger whenever there is a change to USB_CONIDSTS, i.e., when a plug is connected or disconnected.

53.5.3 Session Request Protocol (SRP)

53.5.3.1 A-Device SRP

Figure 53.5-1 illustrates the flow of SRP when the OTG_FS is acting as an A-device, i.e., default host and the device that powers Vbus.

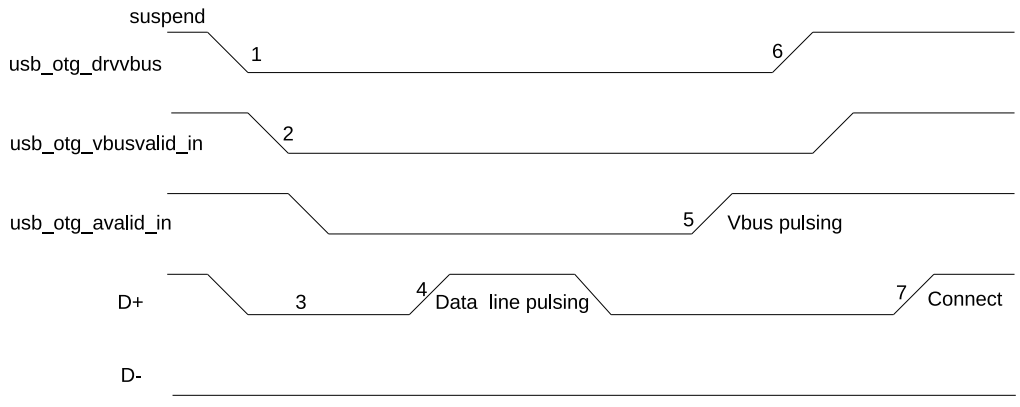


Figure 53.5-1. A-Device SRP

1. To save power, the application suspends and turns off port power when the bus is idle by writing to the Port Suspend (USB_PRTSUSP to 1'b0) and Port Power (USB_PRTPWR to 1'b0) bits in the Host Port Control and Status register.

2. PHY indicates port power off by deasserting the usb_otg_vbusvalid_in signal.
3. The A-device must detect SE0 for at least 2 ms to start SRP when Vbus power is off.
4. To initiate SRP, the B-device turns on its data line pull-up resistor for 5 to 10 ms. The OTG_FS core detects data-line pulsing.
5. The device drives Vbus above the A-device session valid (2.0 V minimum) for Vbus pulsing. The OTG_FS core interrupts the application on detecting SRP. The Session Request Detected bit (USB_SESSREQINT) is set in Global Interrupt Status register.
6. The application must service the Session Request Detected interrupt and turn on the Port Power bit by writing the Port Power bit in the Host Port Control and Status register. The PHY indicates port power-on by asserting usb_otg_vbusvalid_in signal.
7. When the USB is powered, the B-Device connects, completing the SRP process.

53.5.3.2 B-Device SRP

Figure 53.5-2 illustrates the flow of SRP when the OTG_FS is acting as a B-device, i.e., does not power Vbus.

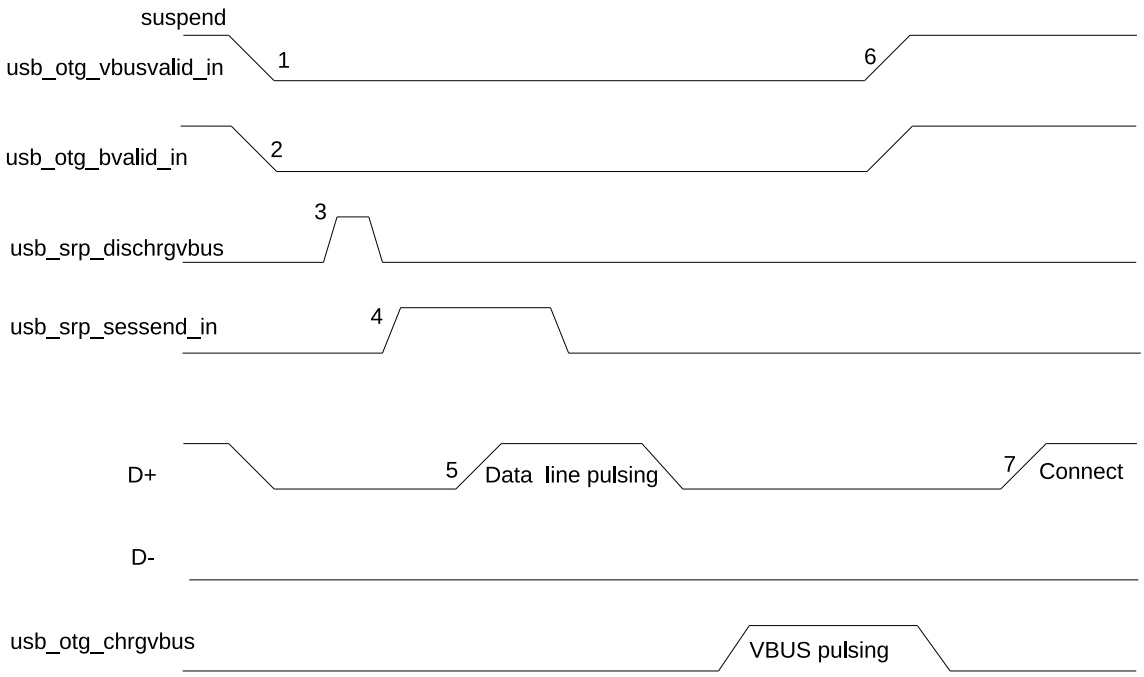


Figure 53.5-2. B-Device SRP

1. To save power, the host (A-device) suspends and turns off port power when the bus is idle. PHY indicates port power off by deasserting the usb_otg_vbusvalid_in signal. The OTG_FS core sets the Early Suspend bit (USB_ERLYSUSP) in the Core Interrupt register after detecting 3 ms of bus idleness. Following this, the OTG_FS core sets the USB Suspend bit (USB_USBSUSP) in the Core Interrupt register. PHY indicates the end of the B-device session by deasserting the usb_otg_bvalid_in signal.
2. The OTG_FS core asserts the usb_otg_dischrgvbus signal to indicate to the PHY to speed up Vbus discharge.
3. PHY indicates the session's end by asserting the usb_otg_sessend_in signal. This is the initial condition

for SRP. The OTG_FS core requires 2 ms of SE0 before initiating SRP. For a USB 2.0 full-speed serial transceiver, the application must wait until Vbus discharges to 0.2 V after USB_BSESVLD is deasserted.

4. The application waits for 1.5 seconds (TB_SE0_SRP time) before initiating SRP by writing the Session Request bit (USB_SESREQ) in the OTG Control and Status register. The OTG_FS core performs data-line pulsing followed by Vbus pulsing.
5. The host (A-device) detects SRP from either the data-line or Vbus pulsing, and turns on Vbus. The PHY indicates Vbus power-on by asserting usb_otg_vbusvalid_in.
6. The OTG_FS core performs Vbus pulsing by asserting usb_srp_chrgvbus. The host (A-device) starts a new session by turning on Vbus, indicating SRP success. The OTG_FS core interrupts the application by setting the Session Request Success Status Change bit (USB_SESREQSC) in the OTG Interrupt Status register. The application reads the Session Request Success bit in the OTG Control and Status register.
7. When the USB is powered, the OTG_FS core connects, completing the SRP process.

53.5.4 Host Negotiation Protocol (HNP)

53.5.4.1 A-Device HNP

Figure 53.5-3 illustrates the flow of HNP when the OTG_FS is acting as an A-device.

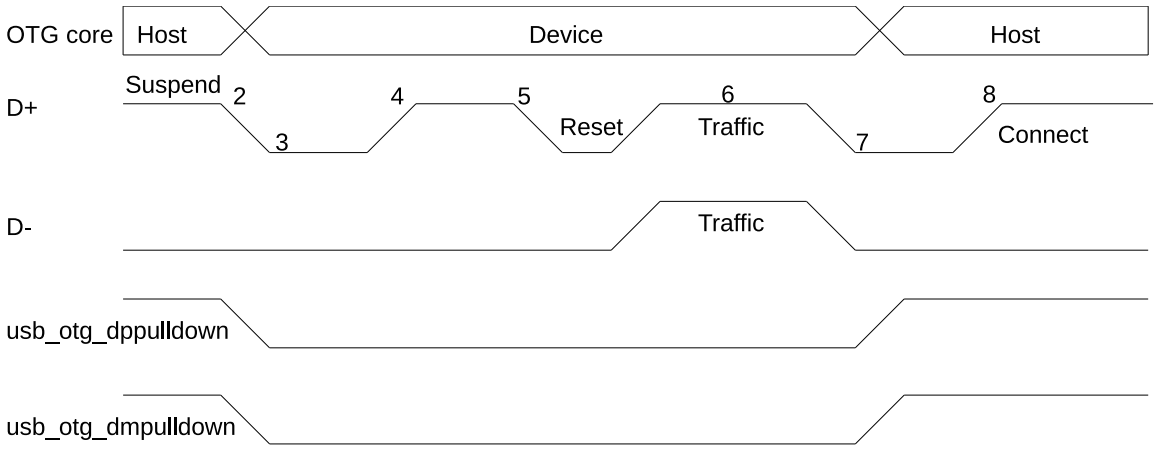


Figure 53.5-3. A-Device HNP

1. The OTG_FS core sends the B-device a SetFeature b_hnp_enable descriptor to enable HNP support. The B-device's ACK response indicates that the B-device supports HNP. The application must set Host Set HNP Enable bit (USB_HSTSETHNPEN) in the OTG Control and Status register to indicate to the OTG_FS core that the B-device supports HNP.
2. When it has finished using the bus, the application suspends by writing the Port Suspend bit (USB_PRTSUSP) in the Host Port Control and Status register.
3. When the B-device observes a USB suspend, it disconnects, indicating the initial condition for HNP. The B-device initiates HNP only when it must switch to the host role; otherwise, the bus continues to be suspended. The OTG_FS core sets the Host Negotiation Detected interrupt (USB_HSTNEGDET) in the OTG Interrupt Status register, indicating the start of HNP. The OTG_FS core deasserts the usb_otg_dppulldown and usb_otg_dmpulldown signals to indicate a device role. The PHY enables the D+ pull-up resistor, thus indicates a connection for the B-device. The application must read the Current

Mode bit (USB_CURMOD_INT) in the OTG Control and Status register to determine Device mode operation.

4. The B-device detects the connection, issues a USB reset, and enumerates the OTG_FS core for data traffic.
5. The B-device continues the host role, initiating traffic, and suspends the bus when done. The OTG_FS core sets the Early Suspend bit (USB_ERLYSUSP) in the Core Interrupt register after detecting 3 ms of bus idleness. Following this, the OTG_FS core sets the USB Suspend bit (USB_USBSUSP) in the Core Interrupt register.
6. In Negotiated mode, the OTG_FS core detects the suspend, disconnects, and switches back to the host role. The OTG_FS core asserts the usb_otg_dppulldown and usb_otg_dmpulldown signals to indicate its assumption of the host role.
7. The OTG_FS core sets the Connector ID Status Change bit (USB_CONIDSTS) in the OTG Interrupt Status register. The application must read the connector ID status in the OTG Control and Status register to determine the OTG_FS core's operation as an A-device. This indicates the completion of HNP to the application. The application must read the Current Mode bit (USB_CURMOD_INT) in the OTG Control and Status register to determine Host mode operation.
8. The B-device connects, completing the HNP process.

53.5.4.2 B-Device HNP

Figure 53.5-4 illustrates the flow of HNP when the OTG_FS is acting as a B-device.

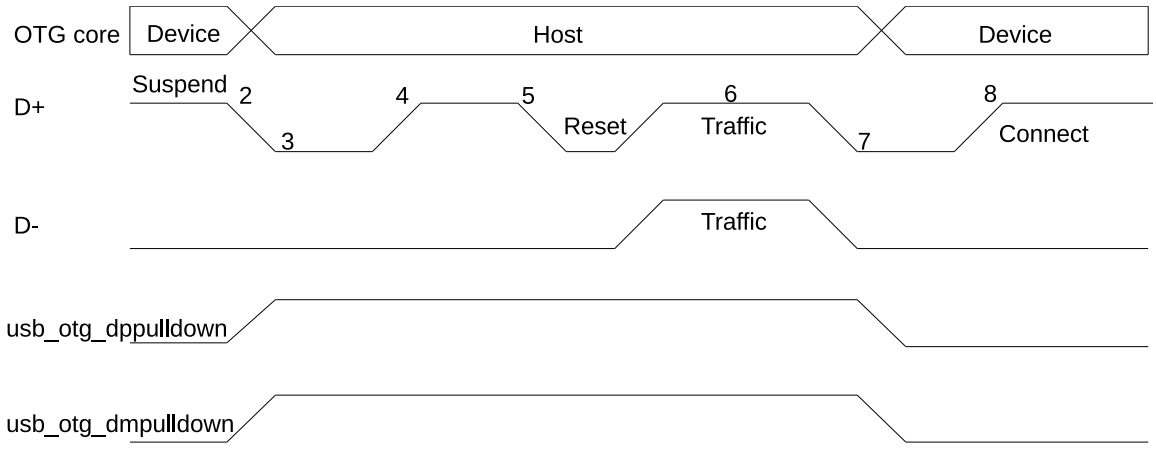


Figure 53.5-4. B-Device HNP

1. The A-device sends the SetFeature b_hnp_enable descriptor to enable HNP support. The OTG_FS core's ACK response indicates that it supports HNP. The application must set the Device HNP Enable bit (USB_DEVHNPEN) in the OTG Control and Status register to indicate HNP support. The application sets the HNP Request bit (USB_DEVHNPEN) in the OTG Control and Status register to indicate to the OTG_FS core to initiate HNP.
2. When A-device has finished using the bus, it suspends the bus.
   (a) The OTG_FS core sets the Early Suspend bit (USB_ERLYSUSP) in the Core Interrupt register after 3 ms of bus idleness. Following this, the OTG_FS core sets the USB Suspend bit (USB_USBSUSP) in

the Core Interrupt register. The OTG_FS core disconnects and the A-device detects SE0 on the bus, indicating HNP.

(b) The OTG_FS core asserts the usb_otg_dppulldown and usb_otg_dmpulldown signals to indicate its assumption of the host role.

(c) The A-device responds by activating its D+ pull-up resistor within 3 ms of detecting SE0. The OTG_FS core detects this as a connect.

(d) The OTG_FS core sets the Host Negotiation Success Status Change interrupt in the OTG Interrupt Status register (USB_CONIDSTS), indicating the HNP status. The application must read the Host Negotiation Success bit (USB_HSTNEGSCS) in the OTG Control and Status register to determine host negotiation success. The application must read the Current Mode bit (USB_CURMOD_INT) in the Core Interrupt register to determine Host mode operation.

3. Program the USB_PRTPWR bit to 1'b1. This drives Vbus on the USB.
4. Wait for the USB_PRTCONNDET interrupt. This indicates that a device is connected to the port.
5. The application sets the reset bit (USB_PRTRST) and the OTG_FS core issues a USB reset and enumerates the A-device for data traffic.
6. Wait for the USB_PRTENCHNG interrupt.
7. The OTG_FS core continues the host role of initiating traffic, and when done, suspends the bus by writing the Port Suspend bit (USB_PRTSUSP) in the Host Port Control and Status register.
8. In Negotiated mode, when the A-device detects a suspend, it disconnects and switches back to the host role. The OTG_FS core deasserts the usb_otg_dppulldown and usb_otg_dmpulldown signals to indicate the assumption of the device role.
9. The application must read the Current Mode bit (USB_CURMOD_INT) in the Core Interrupt register to determine the Host mode operation.
10. The OTG_FS core connects, completing the HNP process.

53.6 Registers

The catalog and comprehensive specifications of USB OTG registers are subject to a Non-Disclosure Agreement (NDA) as mandated by the IP provider. To obtain support information for a particular register, please contact Espressif Technical Support Team via Technical Inquires.

Chapter 54

USB Serial/JTAG Controller (USB_SERIAL_JTAG)

ESP32-P4 contains a USB Serial/JTAG Controller. This unit can be used to program the SoC's flash, read program output, as well as attach a debugger to the running program. All of these are possible for any computer with a USB host (hereafter referred to as 'host') without any active external components.

54.1 Overview

While programming and debugging an ESP32-P4 project using the UART and JTAG functionality is certainly possible, it has a few downsides. First of all, both UART and JTAG take up IO pins and as such, fewer pins are left usable for controlling external signals in software. Additionally, an external chip or adapter is needed for both UART and JTAG to interface with a host computer, which means it will be necessary to integrate these two functionalities in the form of external chips or debugging adapters.

In order to alleviate these issues, ESP32-P4 provides a USB Serial/JTAG Controller, which integrates the functionality of both a USB-to-serial converter as well as a USB-to-JTAG adapter. As this device directly interfaces with an external USB host using only the two data lines required by USB 2.0, only two pins are required to be dedicated to this functionality for debugging ESP32-P4.

54.2 Features

The USB Serial/JTAG controller has the following features:

- USB Full-speed device; Hardwired for CDC-ACM (Communication Device Class - Abstract Control Model) and JTAG adapter functionality
- CDC-ACM:
  - Integrates CDC-ACM adherent serial port emulation (plug-and-play on most modern OSes)
  - Supports host controllable chip reset and entry into download mode
- JTAG adapter functionality:
  - Allows fast communication with CPU debugging core using a compact representation of JTAG instructions
- A control endpoint, a dummy interrupt endpoint, two bulk input endpoints, and two bulk output endpoints; Up to 64-byte data payload size
- Internal PHY: very few or no external components needed to connect to a host computer

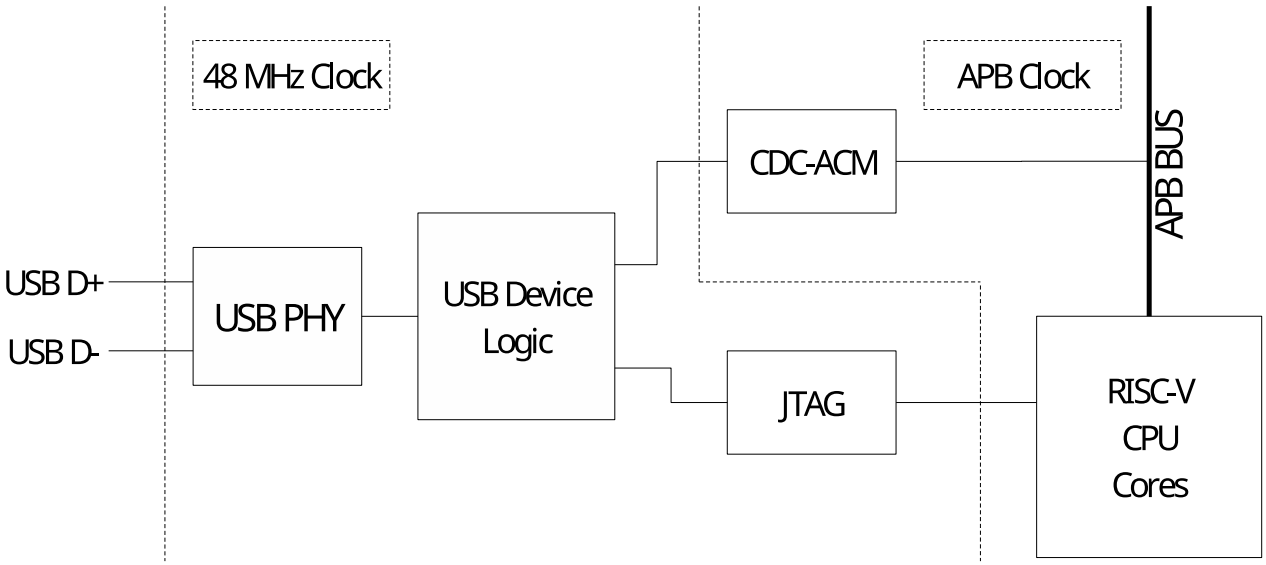


Figure 54.2-1. USB Serial/JTAG High Level Diagram

As shown in Figure 54.2-1, the USB Serial/JTAG controller consists of a USB PHY, a USB device interface, a JTAG command processor, a response capture unit, and the CDC-ACM registers. The PHY and device interface are clocked from a 48 MHz clock derived from the baseband PLL (BBPLL); the software-accessible side of the CDC-ACM block is clocked from APB_CLK. The JTAG command processor is connected to the JTAG debugging unit of the main processor; the CDC-ACM registers are connected to the APB bus and as such can be read from and written to by software running on the main CPU.

Note that while the USB Serial/JTAG device supports USB 2.0 standard, it only supports Full-speed (12 Mbps) mode but not other modes that the USB 2.0 standard introduced, e.g., the High-speed (480 Mbps) mode.

Figure 54.2-2 shows the internal details of the USB Serial/JTAG controller on the USB side. The USB Serial/JTAG controller consists of a USB 2.0 Full-speed device. It contains a control endpoint, a dummy interrupt endpoint, two bulk input endpoints, and two bulk output endpoints. Together, these form a USB composite device, which consists of a CDC-ACM USB class device as well as a vendor-specific device implementing the JTAG interface. On the SoC side, the JTAG interface is directly connected to the RISC-V CPU's debugging interface, allowing debugging of programs running on that core. Meanwhile, the CDC-ACM device is exposed as a set of registers, allowing a program on the CPU to read and write from it. Additionally, the ROM startup code of the SoC contains code that allows the user to reprogram attached flash memory using this interface.

Note:
To get specific information about what endpoint numbers are used for what functionality, please parse the relevant USB descriptors as returned by the device. This guarantees compatibility with other devices with the same functionality but a different implementation.

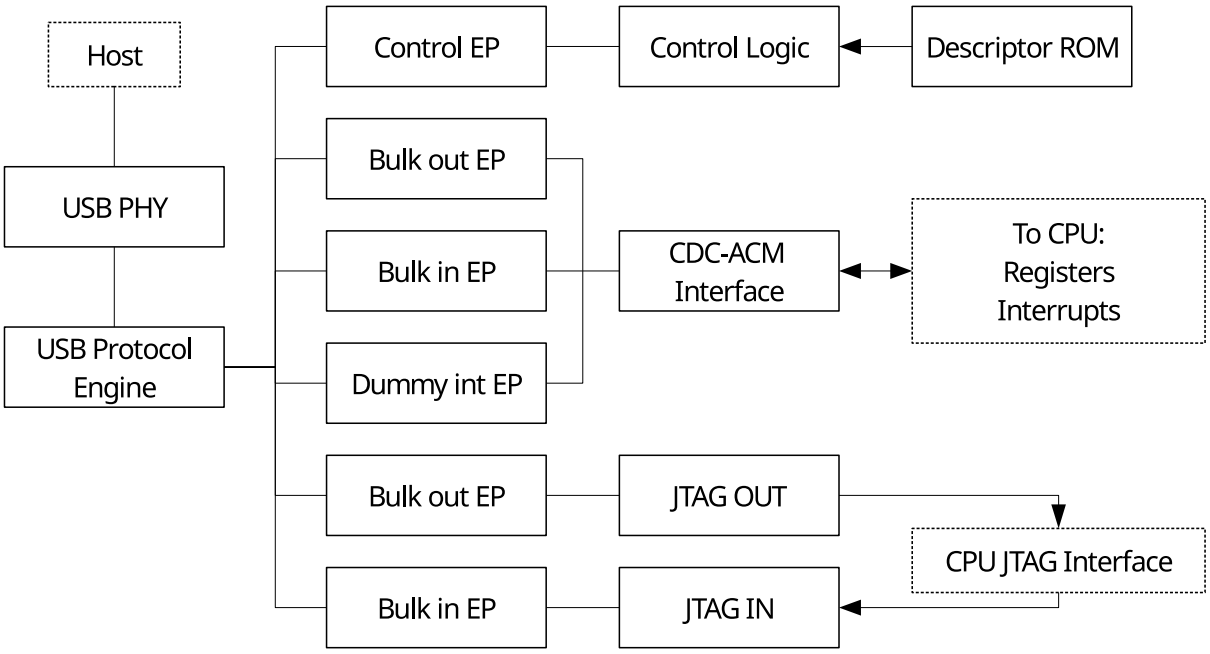


Figure 54.2-2. USB Serial/JTAG Block Diagram

54.3 Functional Description

The USB Serial/JTAG controller interfaces with a USB host processor on one side, and with the CPU debugging hardware as well as the software that communicates through the CDC-ACM port on the other side.

54.3.1 CDC-ACM USB Interface Functional Description

The CDC-ACM interface adheres to the standard USB CDC-ACM class for serial port emulation. It contains a dummy interrupt endpoint (which will never send any events, as they are not implemented nor needed) and a Bulk IN as well as a Bulk OUT endpoint for the host's received and sent serial data respectively. These endpoints can handle 64-bytes packet at a time, allowing high throughput. As CDC-ACM is a standard USB device class, a host generally can function without any special installation procedures. That is to say, when the USB debugging device is properly connected to a host, the operating system should show a new serial port moments later.

The CDC-ACM interface accepts the following standard CDC-ACM control requests:

Table 54.3-1. Standard CDC-ACM Control Requests

Command	Action
SEND_BREAK	Accepted but ignored (dummy)
SET_LINE_CODING	Accepted, value sent is readable in software
GET_LINE_CODING	By default, returns 9600 baud, no parity, 8 data bits, 1 stop bit (Can be changed through software)
SET_CONTROL_LINE_STATE	Set the state of the RTS/DTR lines. See Table 54.3-2

Aside from general-purpose communication, the CDC-ACM interface can also be used to reset ESP32-P4 and optionally make it enter download mode to flash new firmware. This can be realized by setting the RTS and DTR lines on the virtual serial port.

Table 54.3-2. CDC-ACM Settings with RTS and DTR

RTS	DTR	Action
0	0	Clear download mode flag
0	1	Set download mode flag
1	0	Reset ESP32-P4
1	1	No action

Note that if the download mode flag is set when ESP32-P4 is reset, ESP32-P4 will reboot into download mode. When this flag is cleared and the chip is reset, ESP32-P4 will boot from flash. For specific sequences, please refer to Section 54.4. All these functions can also be disabled by programming various eFuses. Please refer to Chapter 9.4 *eFuse Controller (EFUSE)* for more details.

54.3.2 CDC-ACM Firmware Interface Functional Description

The CPU can interact with the USB Serial/JTAG controller as the module is connected to the internal APB bus of ESP32-P4. This is mainly used to read and write data from and to the virtual serial port on the attached host.

USB CDC-ACM serial data is sent to and received from the host in packets of 0 to 64 bytes in size. When enough CDC-ACM data has accumulated in the host, the host sends a packet to the CDC-ACM receive endpoint, and the USB Serial/JTAG controller accepts this packet if it has a free buffer. Conversely, the host checks periodically if the USB Serial/JTAG controller has a packet ready to be sent to the host, and if so, receives this packet.

Firmware can get notified of new data from the host in one of the following two ways. First of all, the USB_SERIAL_JTAG_SERIAL_OUT_EP_DATA_AVAIL bit will remain set as long as there still is unread host data in the buffer. Secondly, the availability of data will trigger the USB_SERIAL_JTAG_SERIAL_OUT_RECV_PKT_INT interrupt. When data is available, it can be read by firmware through repeatedly reading bytes from USB_SERIAL_JTAG_EP1_REG. The amount of bytes to read can be determined by checking the USB_SERIAL_JTAG_SERIAL_OUT_EP_DATA_AVAIL bit after reading each byte to see if there is more data to read. After all data is read, the USB device is automatically readied to receive a new data packet from the host.

When the firmware has data to send, it can put the data in the send buffer and trigger a flush to allow the host to receive the data in a USB packet. In order to do so, there needs to be space available in the send buffer. Firmware can check this by reading USB_REG_SERIAL_IN_EP_DATA_FREE. A 1 in this register field indicates there is still free room in the buffer, and firmware can fill the buffer by writing bytes to the USB_SERIAL_JTAG_EP1_REG register. Writing the buffer does not immediately trigger sending data to the host until the buffer is flushed. After the flush, the entire buffer will be ready to be received by the USB host at once. A flush can be triggered in two ways: 1) after the 64th byte is written to the buffer, the USB hardware will automatically flush the buffer to the host; or 2) firmware can trigger a flush by writing 1 to USB_REG_SERIAL_WR_DONE.

Regardless of how a flush is triggered, the send buffer will be unavailable for firmware to write into until it has

been fully read by the host. As soon as the send buffer has been fully read, the USB_SERIAL_JTAG_SERIAL_IN_EMPTY_INT interrupt will be triggered, indicating that the send buffer can receive another 64 bytes.

It is possible to handle some out-of-band serial requests in software, specifically, the host setting DTR and RTS and changing the line state. If the CDC-ACM interface receives a SET_LINE_CODING request, the peripheral can be configured to trigger a USB_SERIAL_JTAG_SET_LINE_CODE_INT interrupt, at which point the line coding can be read from the USB_SERIAL_JTAG_SET_LINE_CODE_WO_REG register. Similarly, SET_CONTROL_LINE_STATE requests will trigger USB_SERIAL_JTAG_RTS_CHG_INT and USB_SERIAL_JTAG_DTR_CHG_INT interrupts if they change the state of these lines. Software can then read the specific state through the USB_SERIAL_JTAG_RTS and USB_SERIAL_JTAG_DTR bits. Note that as described earlier, certain RTS/DTR sequences lead to hardware reset of ESP32-P4. Software can disable hardware recognition of these DTR/RTS sequences by setting the USB_SERIAL_JTAG_USB_UART_CHIP_RST_DIS bit, allowing software to interpret these signals freely.

Finally, the host can read the current line state using GET_LINE_CODING. This event sends back the data in the USB_SERIAL_JTAG_GET_LINE_CODE_WO_REG register and triggers a USB_SERIAL_JTAG_GET_LINE_CODE_INT interrupt.

54.3.3 USB-to-JTAG Interface: JTAG Command Processor

The USB-to-JTAG interface uses a vendor-specific class for its implementation. It consists of two endpoints, one to receive commands and another to send responses. Additionally, some less time-sensitive commands can be given as control requests.

Commands from the host to the JTAG interface are interpreted by the JTAG command processor. Internally, the JTAG command processor implements a full four-wire JTAG bus, consisting of the TCK, TMS and TDI output lines to the RISC-V CPU, as well as the TDO line signalling back from the CPU to the JTAG response capture unit. These signals adhere to the IEEE 1149.1 JTAG standards. Additionally, there is an SRST line to reset ESP32-P4.

Optionally, software can set USB_SERIAL_JTAG_USB_JTAG_BRIDGE_EN in order to redirect these signals to the GPIO matrix instead, where they can be routed to IO pads on ESP32-P4. This also allows external devices to be debugged via the USB Serial/JTAG peripheral.

The JTAG command processor parses each received nibble (4-bit value) as a command. As USB data is received in 8-bit bytes, this means each byte contains two commands. The USB command processor will execute high-nibble first and low-nibble second. The commands are used to control the TCK, TMS, TDI, and SRST lines of the internal JTAG bus, as well as to signal the JTAG response capture unit the state of the TDO line (which is driven by the CPU debugging logic) that needs to be captured.

In the internal JTAG bus, TCK, TMS, TDI, and TDO are connected directly to the JTAG debugging logic of the RISC-V CPU. SRST is connected to the reset logic of the digital circuitry in ESP32-P4 and a high level on this line will cause a digital system reset. Note that the USB Serial/JTAG controller itself is not affected by SRST.

A nibble can contain the following commands:

Table 54.3-3. Commands of a Nibble

bit	3	2	1	0
CMD_CLK	0	cap	tms	tdi
CMD_RST	1	0	0	srst
CMD_FLUSH	1	0	1	0
CMD_RSV	1	0	1	1
CMD_REP	1	1	R1	R0

- CMD_CLK will set the TDI and TMS as the indicated values and emit one clock pulse on TCK. If the CAP bit is 1, it will instruct the JTAG response capture unit to capture the state of the TDO line. This instruction forms the basis of JTAG communication.
- CMD_RST will set the state of the SRST line as the indicated value. This can be used to reset ESP32-P4.
- CMD_FLUSH will instruct the JTAG response capture unit to flush the buffer of all bits it collected so the host is able to read them. Note that in some cases, a JTAG transaction will end in an odd number of commands and as such an odd number of nibbles. In this case, it is allowed to repeat the CMD_FLUSH command to get an even number of nibbles fitting an integer number of bytes.
- CMD_RSV is reserved in the current implementation. This command will be ignored when received by ESP32-P4.
- CMD_REP repeats the last (non-CMD_REP) command for a certain number of times. The purpose is to compress command streams which repeat the CMD_CLK instruction for multiple times. A command such as CMD_CLK can be followed by multiple CMD_REP commands. The number of repetitions done by one CMD_REP can be expressed as $repetition_count = (R1 \times 2 + R0) \times (4^{cmd_rep_count})$, where cmd_rep_count indicates the number of the CMD_REP instruction that went directly before it. Note that the CMD_REP command is only intended to repeat a CMD_CLK command. Specifically, using it on a CMD_FLUSH command may lead to an unresponsive USB device, and a USB reset will be required to recover it.

54.3.4 USB-to-JTAG Interface: CMD_REP Usage Example

Here is a list of commands as an illustration of the usage of CMD_REP. Note that each command is a nibble, and in this example, the bytewise command stream would be 0x0D 0x5E 0xCF.

1. 0x0 (CMD_CLK: cap=0, tdi=0, tms=0)
2. 0xD (CMD_REP: R1=0, R0=1)
3. 0x5 (CMD_CLK: cap=1, tdi=0, tms=1)
4. 0xE (CMD_REP: R1=1, R0=0)
5. 0xC (CMD_REP: R1=0, R0=0)
6. 0xF (CMD_REP: R1=1, R0=1)

The following shows what happens at every step:

1. TCK is clocked with the TDI and TMS lines set to 0. No data is captured.
2. TCK is clocked another $(0 \times 2 + 1) \times (4^0) = 1$ time with the same settings as step 1.

3. TCK is clocked with the TDI line set to 0 and TMS set to 1. Data on the TDO line is captured.
4. TCK is clocked another $(1 \times 2 + 0) \times (4^0) = 2$ times with the same settings as step 3.
5. Nothing happens: $(0 \times 2 + 0) \times (4^1) = 0$. Note that this increases cmd_rep_count in the next step.
6. TCK is clocked another $(1 \times 2 + 1) \times (4^2) = 48$ times with the same settings as step 3.

In other words, this example stream has the same net effect as that of executing command 1 twice, then repeating command 3 for 51 times.

54.3.5 USB-to-JTAG Interface: Response Capture Unit

The response capture unit reads the TDO line of the internal JTAG bus and captures its value when the command parser executes a CMD_CLK with cap=1. It puts this bit into an internal shift register, and writes a byte into the USB buffer when 8 bits have been collected. Of these 8 bits, the least significant one is the one that is read from TDO the earliest.

As soon as either 64 bytes (512 bits) have been collected or a CMD_FLUSH command is executed, the response capture unit will make the buffer available for the host to receive. Note that the interface to the USB logic is double-buffered. Therefore, as long as the USB throughput is sufficient, the response capture unit can always receive more data. That is to say, while one of the buffers is waiting to be sent to the host, the other can receive more data. When the host has received data from its buffer and the response capture unit flushes its buffer, the two buffers exchange position.

This also means that a command stream can cause at most 128 bytes of capture data generated (less if there are flush commands in the stream) without the host acting to receive the generated data. If more data is generated anyway, the command stream will pause and the device will not accept more commands until the generated capture data is read out.

Note that in general, the logic of the response capture unit tries not to send zero-byte responses. For instance, sending a series of CMD_FLUSH commands will not cause a series of 0-byte USB responses to be sent. However, in the current implementation, some zero-0 responses may be generated in extraordinary circumstances. It is recommended to ignore these responses.

54.3.6 USB-to-JTAG Interface: Control Transfer Requests

Aside from the command processor and the response capture unit, the USB-to-JTAG interface also understands some control requests, as documented in the table below:

Table 54.3-4. USB-to-JTAG Control Requests

bmRequestType	bRequest	wValue	wIndex	wLength	Data
01000000b	0 (VEND_JTAG_SETDIV)	[divider]	interface	0	None
01000000b	1 (VEND_JTAG_SETIO)	[iobits]	interface	0	None
11000000b	2 (VEND_JTAG_GETTDO)	0	interface	1	[iostate]
10000000b	6 (GET_DESCRIPTOR)	0x2000	0	256	[jtag cap desc]

- VEND_JTAG_SETDIV sets the divider used. This directly affects the duration of a TCK clock pulse. The TCK clock pulses are derived from a base clock of 48 MHz, which is divided down using an internal

divider. This control request allows the host to set this divider. Note that on startup, the divider is set to 2, which means the TCK clock rate will generally be 24 MHz.

- VEND_JTAG_SETIO can bypass the JTAG command processor to set the internal TDI, TDO, TMS, and SRST lines to given values. These values are encoded in the wValue field in the format of 11'b0, srst, trst, tck, tms, tdi.
- VEND_JTAG_GETTDO can bypass the JTAG response capture unit to read the internal TDO signal directly. This request returns one byte of data, of which the least significant bit represents the status of the TDO line.
- GET_DESCRIPTOR is a standard USB request. However, it can also be used with a vendor-specific wValue of 0x2000 to get the JTAG capabilities descriptor. This returns a certain amount of bytes representing the following fixed structure, which describes the capabilities of the USB-to-JTAG adapter (as shown in Table 54.3-5). This structure allows host software to automatically support future revisions of the hardware without the need for an update.

The JTAG capability descriptors of ESP32-P4 are as follows. Note that all 16-bit values are little-endian.

Table 54.3-5. JTAG Capability Descriptors

Byte	Value	Description
0	1	JTAG protocol capability structure version
1	10	Total length of JTAG protocol capabilities
2	1	Type of this struct: 1 for speed capability struct
3	8	Length of this speed capabilities struct
4 ~ 5	4800	JTAG base clock speed in 10 kHz increments. Note that the maximum TCK speed is half of this value
6 ~ 7	1	Minimum divider value settable by the VEND_JTAG_SETDIV request
8 ~ 9	255	Maximum divider value settable by the VEND_JTAG_SETDIV request

54.4 Recommended Operation

Little setup is needed for using the USB Serial/JTAG device. The USB-to-JTAG hardware itself does not need any setup aside from the standard USB initialization that the host operating system already does. Apart from that, the CDC-ACM emulation on the host side is also plug-and-play.

On the firmware side, very little initialization is needed either. The USB hardware is self-initialized and after boot-up, if a host is connected and listening on the CDC-ACM interface, data can be exchanged as described above without any specific setup except for the situation when the firmware optionally sets up an interrupt service handler.

One thing to note is that there may be situations where either the host is not attached or the CDC-ACM virtual port is not opened. In such cases, the packets that are flushed to the host will never be picked up and the send buffer will never be empty. It is important to detect these situations and implement timeout, as this is the only way to reliably detect whether the port on the host side is closed or not.

Another thing to note is that the USB device is dependent on the BBPLL for the 48 MHz USB PHY clock. If this PLL is disabled, the USB communication will cease to function.

One scenario where this happens is Deep-sleep. The USB Serial/JTAG controller (as well as the attached RISC-V CPU) will be entirely powered down in Deep-sleep mode. If a device needs to be debugged in this mode, it may be preferable to use an external JTAG debugger and a serial interface instead.

The CDC-ACM interface can also be used to reset the SoC and take it into or out of download mode. Generating the correct sequence of handshake signals can be a bit complicated, since most operating systems only allow setting or resetting DTR and RTS separately, but not in tandem. Additionally, some drivers (e.g., the standard CDC-ACM driver on Windows) do not set DTR until RTS is set and the user needs to explicitly set RTS in order to 'propagate' the DTR value. The recommended procedures are introduced below.

To reset the SoC into download mode:

Table 54.4-1. Reset SoC into Download Mode

Action	Internal state	Note
Clear DTR	RTS=?, DTR=0	Initialize to known values
Clear RTS	RTS=0, DTR=0	-
Set DTR	RTS=0, DTR=1	Set download mode flag
Clear RTS	RTS=0, DTR=1	Propagate DTR
Set RTS	RTS=1, DTR=1	-
Clear DTR	RTS=1, DTR=0	Reset SoC
Set RTS	RTS=1, DTR=0	Propagate DTR
Clear RTS	RTS=0, DTR=0	Clear download flag

To reset the SoC into booting from flash:

Table 54.4-2. Reset SoC into Booting from flash

Action	Internal state	Note
Clear DTR	RTS=?, DTR=0	-
Clear RTS	RTS=0, DTR=0	Clear download flag
Set RTS	RTS=1, DTR=0	Reset SoC
Clear RTS	RTS=0, DTR=0	Exit reset

54.5 Interrupts

ESP32-P4's USB Serial/JTAG Controller can generate the USB_SERIAL_JTAG_INTR interrupt signal that will be sent to the Interrupt Matrix. There are several internal interrupt sources from the USB Serial/JTAG Controller that can generate this interrupt signal.

- USB_SERIAL_JTAG_JTAG_IN_FLUSH_INT: triggered when flush cmd is received for the JTAG bulk out endpoint.
- USB_SERIAL_JTAG_SOF_INT: triggered when SOF frame is received.
- USB_SERIAL_JTAG_SERIAL_OUT_RECV_PKT_INT: triggered when Serial Port OUT Endpoint receives one packet.

- USB_SERIAL_JTAG_SERIAL_IN_EMPTY_INT: triggered when Serial Port IN Endpoint is empty.
- USB_SERIAL_JTAG_PID_ERR_INT: triggered when PID error is detected.
- USB_SERIAL_JTAG_CRC5_ERR_INT: triggered when CRC5 error is detected.
- USB_SERIAL_JTAG_CRC16_ERR_INT: triggered when CRC16 error is detected.
- USB_SERIAL_JTAG_STUFF_ERR_INT: triggered when a bit stuffing error is detected.
- USB_SERIAL_JTAG_IN_TOKEN_REC_IN_EP1_INT: triggered when IN token for IN endpoint 1 is received.
- USB_SERIAL_JTAG_USB_BUS_RESET_INT: triggered when USB bus reset is detected.
- USB_SERIAL_JTAG_OUT_EP1_ZERO_PAYLOAD_INT: triggered when OUT endpoint 1 receives packet with zero payload.
- USB_SERIAL_JTAG_OUT_EP2_ZERO_PAYLOAD_INT: triggered when OUT endpoint 2 receives packet with zero payload.
- USB_SERIAL_JTAG_RTS_CHG_INT: triggered when level of RTS from USB serial channel is changed.
- USB_SERIAL_JTAG_DTR_CHG_INT: triggered when level of DTR from USB serial channel is changed.
- USB_SERIAL_JTAG_GET_LINE_CODE_INT: triggered when level of GET LINE CODING request is received.
- USB_SERIAL_JTAG_SET_LINE_CODE_INT: triggered when level of SET LINE CODING request is received.

54.6 Register Summary

The addresses in this section are relative to USB Serial/JTAG controller base address provided in Table 9.3-2 in Chapter 9 *System and Memory*.

The abbreviations given in Column **Access** are explained in Section *Access Types for Registers*.

Name	Description	Address	Access
Configuration Registers			
USB_SERIAL_JTAG_EP1_REG	FIFO access for the CDC-ACM data IN and OUT endpoints	0x0000	R/W
USB_SERIAL_JTAG_EP1_CONF_REG	Configuration and control registers for the CDC-ACM FIFOs	0x0004	varies
USB_SERIAL_JTAG_CONF0_REG	PHY hardware configuration	0x0018	R/W
USB_SERIAL_JTAG_TEST_REG	Registers used for debugging the PHY	0x001C	varies
USB_SERIAL_JTAG_MISC_CONF_REG	Clock enable control	0x0044	R/W
USB_SERIAL_JTAG_MEM_CONF_REG	Memory power control	0x0048	R/W
USB_SERIAL_JTAG_CHIP_RST_REG	CDC-ACM chip reset control	0x004C	varies
USB_SERIAL_JTAG_GET_LINE_CODE_W0_REG	W0 of GET_LINE_CODING command	0x0058	R/W
USB_SERIAL_JTAG_GET_LINE_CODE_W1_REG	W1 of GET_LINE_CODING command	0x005C	R/W
USB_SERIAL_JTAG_CONFIG_UPDATE_REG	Configuration registers' value update	0x0060	WT
USB_SERIAL_JTAG_SER_AFIFO_CONFIG_REG	Serial AFIFO configure register	0x0064	varies
Interrupt Registers			
USB_SERIAL_JTAG_INT_RAW_REG	Interrupt raw status register	0x0008	R/WTC/SS
USB_SERIAL_JTAG_INT_ST_REG	Interrupt status register	0x000C	RO
USB_SERIAL_JTAG_INT_ENA_REG	Interrupt enable status register	0x0010	R/W
USB_SERIAL_JTAG_INT_CLR_REG	Interrupt clear status register	0x0014	WT
Status Registers			
USB_SERIAL_JTAG_JFIFO_ST_REG	JTAG FIFO status and control registers	0x0020	varies
USB_SERIAL_JTAG_FRAM_NUM_REG	Last received SOF frame index register	0x0024	RO
USB_SERIAL_JTAG_IN_EP0_ST_REG	Control IN endpoint status information	0x0028	RO
USB_SERIAL_JTAG_IN_EP1_ST_REG	CDC-ACM IN endpoint status information	0x002C	RO
USB_SERIAL_JTAG_IN_EP2_ST_REG	CDC-ACM interrupt IN endpoint status information	0x0030	RO
USB_SERIAL_JTAG_IN_EP3_ST_REG	JTAG IN endpoint status information	0x0034	RO
USB_SERIAL_JTAG_OUT_EP0_ST_REG	Control OUT endpoint status information	0x0038	RO
USB_SERIAL_JTAG_OUT_EP1_ST_REG	CDC-ACM OUT endpoint status information	0x003C	RO
USB_SERIAL_JTAG_OUT_EP2_ST_REG	JTAG OUT endpoint status information	0x0040	RO

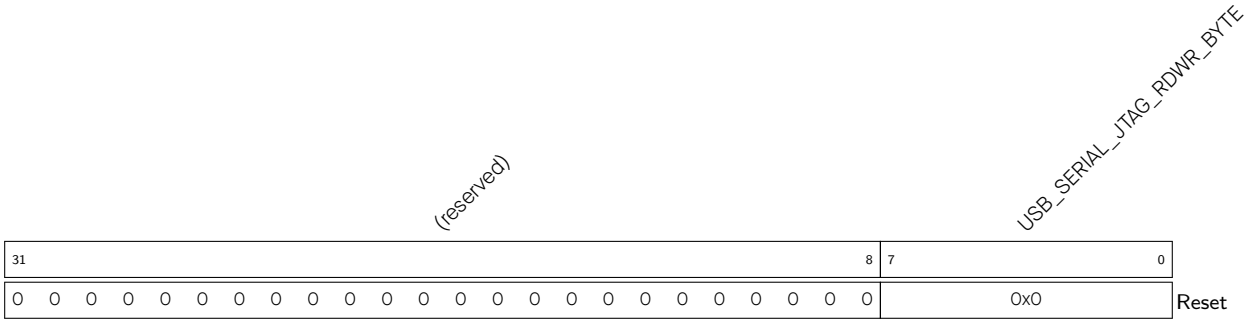
Name	Description	Address	Access
USB_SERIAL_JTAG_SET_LINE_CODE_W0_REG	W0 of SET_LINE_CODING command	0x0050	RO
USB_SERIAL_JTAG_SET_LINE_CODE_W1_REG	W1 of SET_LINE_CODING command	0x0054	RO
USB_SERIAL_JTAG_BUS_RESET_ST_REG	USB Bus reset status register	0x0068	RO
Version Registers			
USB_SERIAL_JTAG_DATE_REG	Date register	0x0088	R/W

54.7 Registers

The addresses in this section are relative to USB Serial/JTAG controller base address provided in Table 9.3-2 in Chapter 9 *System and Memory*.

For how to program reserved fields, please refer to Section Programming Reserved Register Field.

Register 54.1. USB_SERIAL_JTAG_EP1_REG (0x0000)



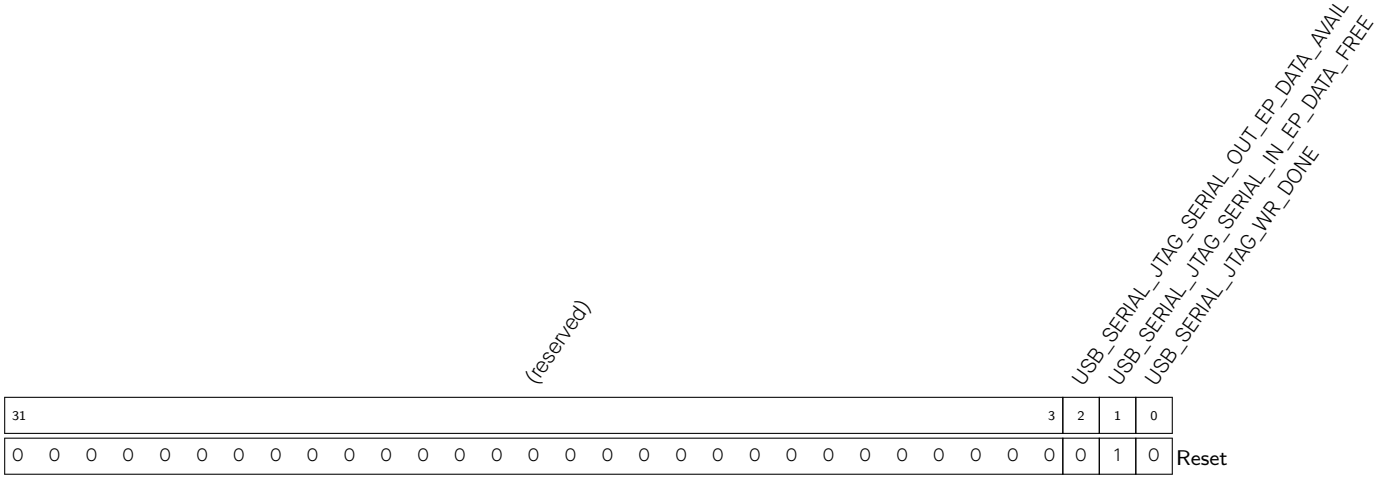
USB_SERIAL_JTAG_RDWR_BYTE A write to this register pushes the written data into the CDC TX FIFO; a read from this register pops a byte from the CDC RX FIFO and returns it.

When USB_SERIAL_JTAG_SERIAL_IN_EMPTY_INT is set, users can write data (up to 64 bytes) into CDC-ACM TX FIFO through this register.

When USB_SERIAL_JTAG_SERIAL_OUT_RECV_PKT_INT is set, users can check how many data is received through USB_SERIAL_JTAG_OUT_EP1_WR_ADDR, then read data from CDC-ACM RX FIFO through this register.

(R/W)

Register 54.2. USB_SERIAL_JTAG_EP1_CONF_REG (0x0004)



USB_SERIAL_JTAG_WR_DONE Configures whether to represent writing byte data to CDC-ACM TX FIFO is done.

0: No effect

1: Represents writing byte data to CDC-ACM TX FIFO is done

This bit then stays 0 until data in CDC-ACM TX FIFO is read by the USB Host.

(WT)

USB_SERIAL_JTAG_SERIAL_IN_EP_DATA_FREE Represents whether CDC-ACM TX FIFO has space available.

0: CDC-ACM TX FIFO is full and no data should be written into it

1: CDC-ACM TX FIFO is not full and data can be written into it

After writing USB_SERIAL_JTAG_WR_DONE, this bit will be 0 until data in CDC-ACM TX FIFO is read by USB Host.

(RO)

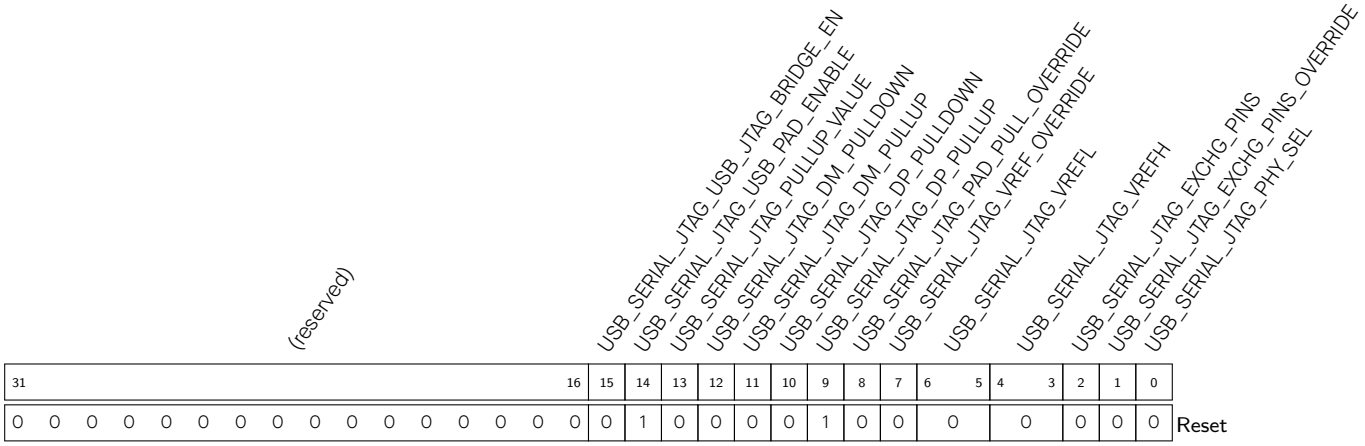
USB_SERIAL_JTAG_SERIAL_OUT_EP_DATA_AVAIL Represents whether there is data in CDC-ACM RX FIFO.

0: There is no data in CDC-ACM RX FIFO

1: There is data in CDC-ACM RX FIFO

(RO)

Register 54.3. USB_SERIAL_JTAG_CONF0_REG (0x0018)



USB_SERIAL_JTAG_PHY_SEL Configures whether to select internal or external PHY.

0: Internal PHY

1: External PHY

(R/W)

USB_SERIAL_JTAG_EXCHG_PINS_OVERRIDE Configures whether to enable software control USB D+ D- exchange.

0: Disable

1: Enable

(R/W)

USB_SERIAL_JTAG_EXCHG_PINS Configures whether to enable USB D+ D- exchange.

0: Disable

1: Enable

(R/W)

USB_SERIAL_JTAG_VREFH Configures single-end input high threshold.

0: 1.76 V

1: 1.84 V

2: 1.92 V

3: 2.00 V

(R/W)

USB_SERIAL_JTAG_VREFL Configures single-end input low threshold.

0: 0.80 V

1: 0.88 V

2: 0.96 V

3: 1.04 V

(R/W)

USB_SERIAL_JTAG_VREF_OVERRIDE Configures whether to enable software control input threshold.

0: Disable

1: Enable

(R/W)

Continued on the next page...

Register 54.3. USB_SERIAL_JTAG_CONF0_REG (0x0018)

Continued from the previous page...

USB_SERIAL_JTAG_PAD_PULL_OVERRIDE Configures whether to enable software to control USB D+ D- pullup and pulldown.
0: Disable
1: Enable
(R/W)

USB_SERIAL_JTAG_DP_PULLUP Configures whether to enable USB D+ pull up when USB_SERIAL_JTAG_PAD_PULL_OVERRIDE is 1.
0: Disable
1: Enable
(R/W)

USB_SERIAL_JTAG_DP_PULLDOWN Configures whether to enable USB D+ pull down when USB_SERIAL_JTAG_PAD_PULL_OVERRIDE is 1.
0: Disable
1: Enable
(R/W)

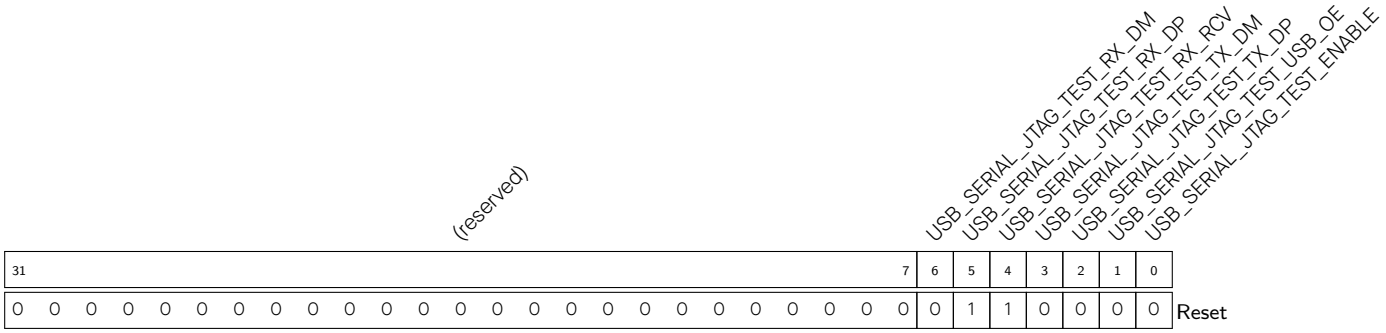
USB_SERIAL_JTAG_DM_PULLDOWN Configures whether to enable USB D- pull down when USB_SERIAL_JTAG_PAD_PULL_OVERRIDE is 1.
0: Disable
1: Enable
(R/W)

USB_SERIAL_JTAG_PULLUP_VALUE Configures the pull up value when USB_SERIAL_JTAG_PAD_PULL_OVERRIDE is 1.
0: 2.2 K
1: 1.1 K
(R/W)

USB_SERIAL_JTAG_USB_PAD_ENABLE Configures whether to enable USB pad function.
0: Disable
1: Enable
(R/W)

USB_SERIAL_JTAG_USB_JTAG_BRIDGE_EN Configures whether to disconnect USB_JTAG and internal JTAG.
0: USB_JTAG is connected to the internal JTAG port of CPU
1: USB_JTAG and the internal JTAG are disconnected, MTMS, MTDI, and MTCK are output through GPIO Matrix, and MTDO is input through GPIO Matrix
(R/W)

Register 54.4. USB_SERIAL_JTAG_TEST_REG (0x001C)



USB_SERIAL_JTAG_TEST_ENABLE Configures whether to enable the test mode of the USB pad.

0: Resume normal operation

1: Enable the test mode of the USB pad

Enabling the test mode of the USB pad allows the USB pad to be controlled/read using the other bits in this register.

(R/W)

USB_SERIAL_JTAG_TEST_USB_OE Configures whether to enable USB pad output.

0: Set D+ and D- to high impedance

1: Output the values set in USB_SERIAL_JTAG_TEST_TX_DP and USB_SERIAL_JTAG_TEST_TX_DM on the D+ and D- pins

(R/W)

USB_SERIAL_JTAG_TEST_TX_DP Configures value of USB D+ in test mode when USB_SERIAL_JTAG_TEST_USB_OE is 1. (R/W)

USB_SERIAL_JTAG_TEST_TX_DM Configures value of USB D- in test mode when USB_SERIAL_JTAG_TEST_USB_OE is 1. (R/W)

USB_SERIAL_JTAG_TEST_RX_RCV Represents the current logical level of the voltage difference between USB D- and USB D+ pads in test mode.

0: USB D- voltage is higher than USB D+

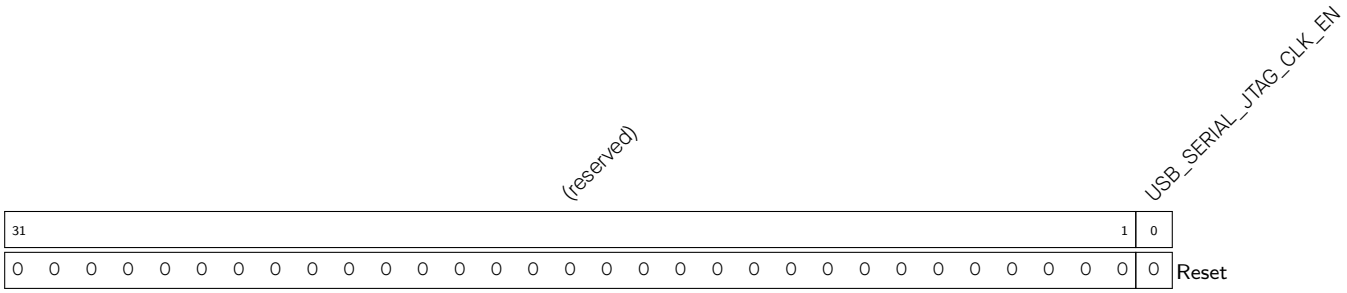
1: USB D+ voltage is higher than USB D-

(RO)

USB_SERIAL_JTAG_TEST_RX_DP Represents the logical level of the USB D+ pad in test mode. (RO)

USB_SERIAL_JTAG_TEST_RX_DM Represents the logical level of the USB D- pad in test mode. (RO)

Register 54.5. USB_SERIAL_JTAG_MISC_CONF_REG (0x0044)



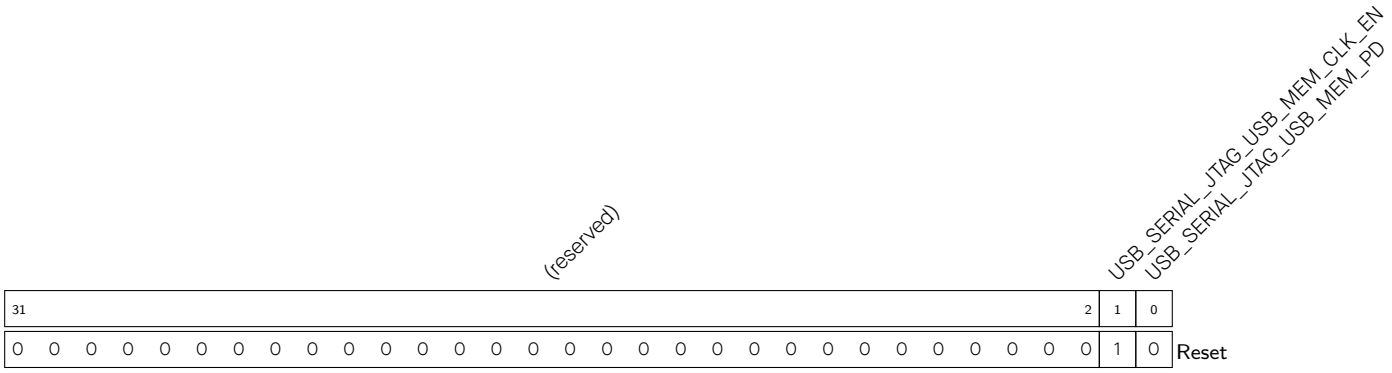
USB_SERIAL_JTAG_CLK_EN Configures whether to force clock on for register.

0: Support clock only when application writes registers

1: Force clock on for register

(R/W)

Register 54.6. USB_SERIAL_JTAG_MEM_CONF_REG (0x0048)



USB_SERIAL_JTAG_USB_MEM_PD Configures whether to power down USB memory.

0: No effect

1: Power down

(R/W)

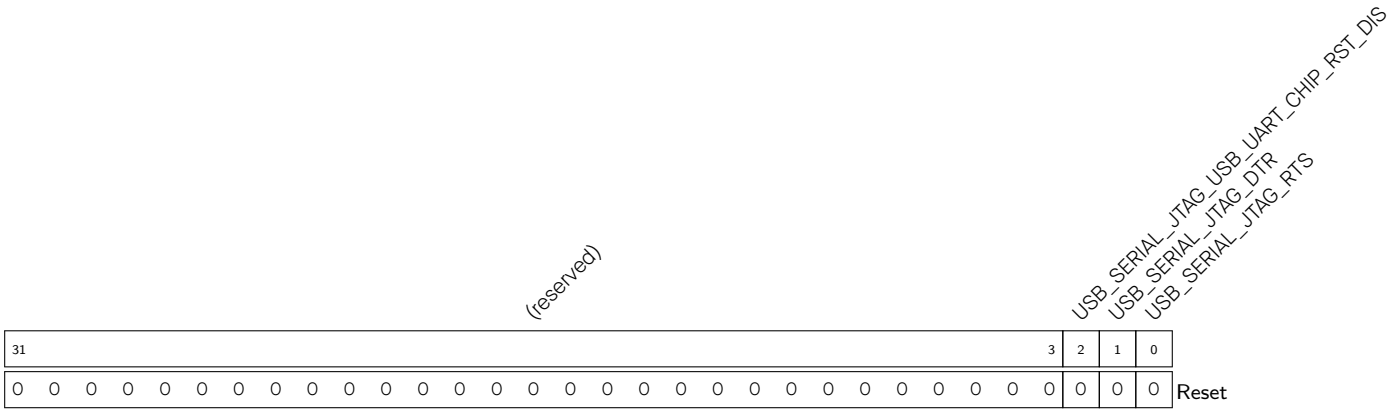
USB_SERIAL_JTAG_USB_MEM_CLK_EN Configures whether to force clock on for USB memory.

0: No effect

1: Force

(R/W)

Register 54.7. USB_SERIAL_JTAG_CHIP_RST_REG (0x004C)



USB_SERIAL_JTAG_RTS Represents the state of RTS signal as set by the most recent SET_LINE_CODING command. (RO)

USB_SERIAL_JTAG_DTR Represents the state of DTR signal as set by the most recent SET_LINE_CODING command. (RO)

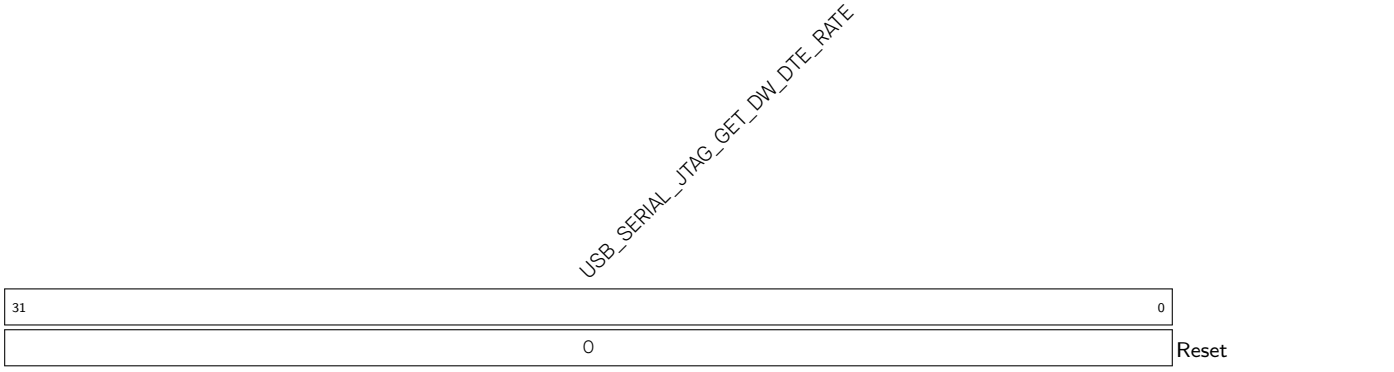
USB_SERIAL_JTAG_USB_UART_CHIP_RST_DIS Configures whether to disable chip reset from USB serial channel.

0: No effect

1: Disable

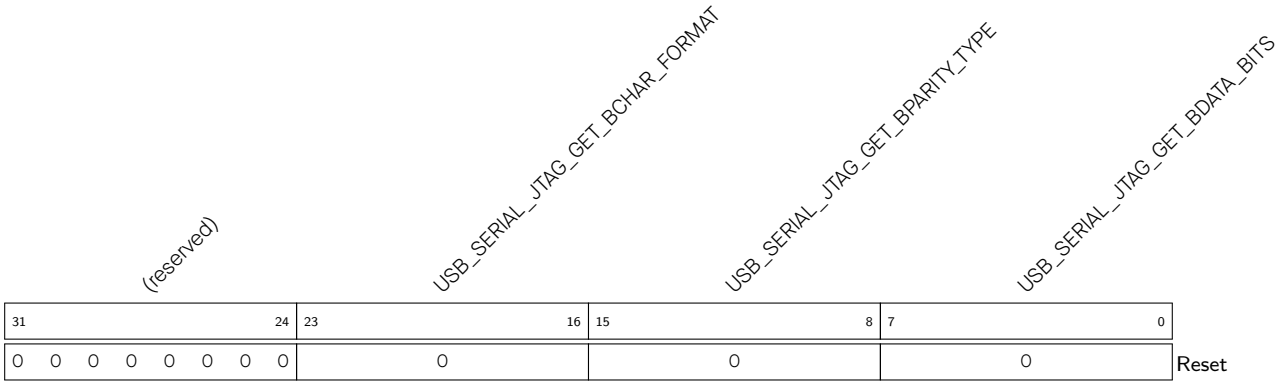
(R/W)

Register 54.8. USB_SERIAL_JTAG_GET_LINE_CODE_W0_REG (0x0058)



USB_SERIAL_JTAG_GET_DW_DTE_RATE Configures the value of dwDTERate set by software, which is requested by GET_LINE_CODING command. (R/W)

Register 54.9. USB_SERIAL_JTAG_GET_LINE_CODE_W1_REG (0x005C)

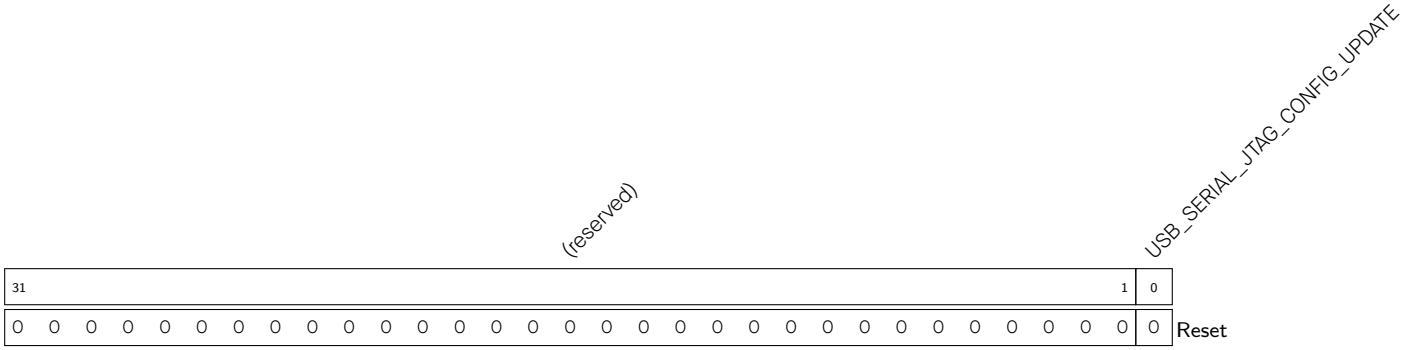


USB_SERIAL_JTAG_GET_BDATA_BITS Configures the value of bDataBits set by software, which is requested by GET_LINE_CODING command. (R/W)

USB_SERIAL_JTAG_GET_BPARITY_TYPE Configures the value of bParityType set by software, which is requested by GET_LINE_CODING command. (R/W)

USB_SERIAL_JTAG_GET_BCHAR_FORMAT Configures the value of bCharFormat set by software, which is requested by GET_LINE_CODING command. (R/W)

Register 54.10. USB_SERIAL_JTAG_CONFIG_UPDATE_REG (0x0060)



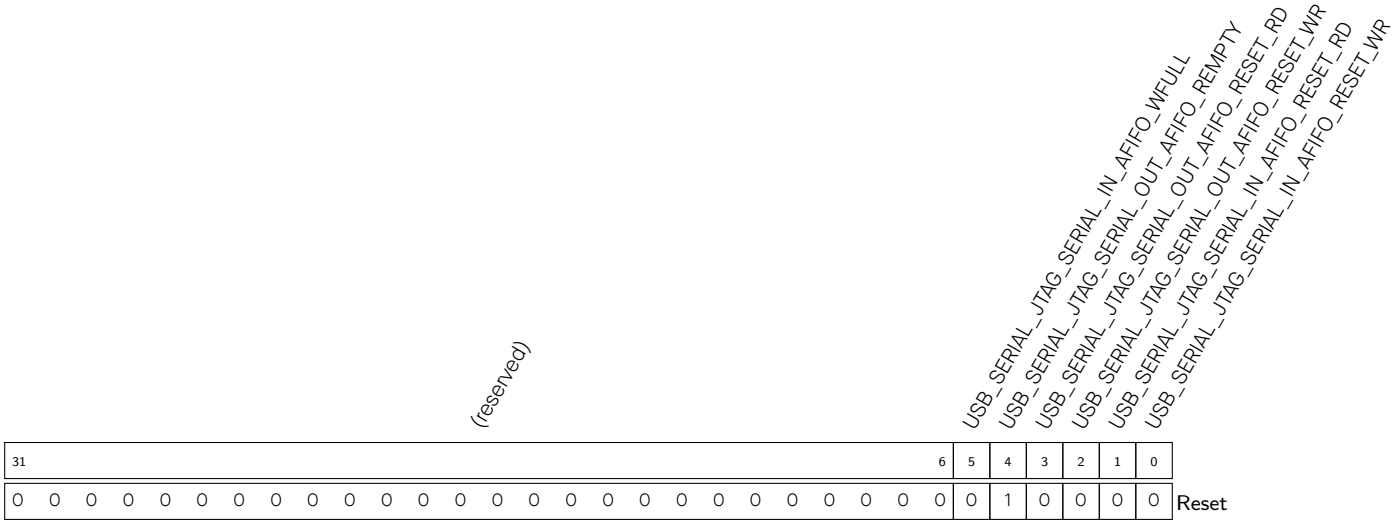
USB_SERIAL_JTAG_CONFIG_UPDATE Configures whether to update the value of configuration registers from APB clock domain to 48 MHz clock domain.

0: No effect

1: Update

(WT)

Register 54.11. USB_SERIAL_JTAG_SER_AFIFO_CONFIG_REG (0x0064)



USB_SERIAL_JTAG_SERIAL_IN_AFIFO_RESET_WR Configures whether to reset CDC_ACM IN async FIFO write clock domain.

0: No effect

1: Reset

(R/W)

USB_SERIAL_JTAG_SERIAL_IN_AFIFO_RESET_RD Configures whether to reset CDC_ACM IN async FIFO read clock domain.

0: No effect

1: Reset

(R/W)

USB_SERIAL_JTAG_SERIAL_OUT_AFIFO_RESET_WR Configures whether to reset CDC_ACM OUT async FIFO write clock domain.

0: No effect

1: Reset

(R/W)

USB_SERIAL_JTAG_SERIAL_OUT_AFIFO_RESET_RD Configures whether to reset CDC_ACM OUT async FIFO read clock domain.

0: No effect

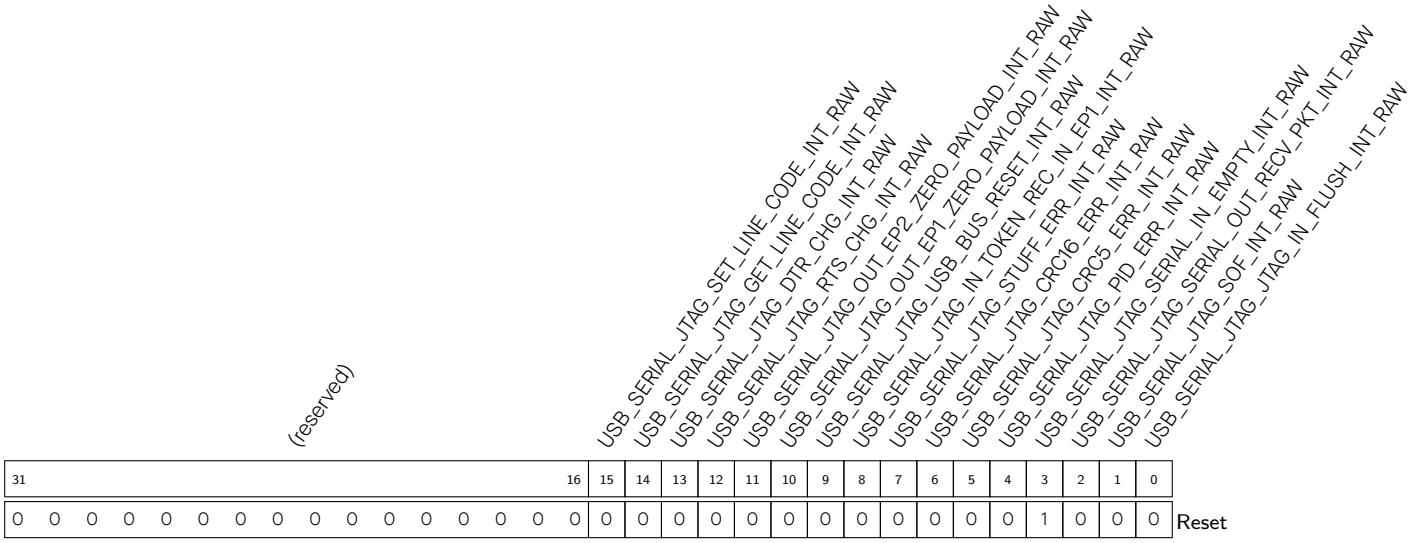
1: Reset

(R/W)

USB_SERIAL_JTAG_SERIAL_OUT_AFIFO_REMPTY Represents CDC_ACM OUT async FIFO empty signal in read clock domain. (RO)

USB_SERIAL_JTAG_SERIAL_IN_AFIFO_WFULL Represents CDC_ACM IN async FIFO full signal in write clock domain. (RO)

Register 54.12. USB_SERIAL_JTAG_INT_RAW_REG (0x0008)



USB_SERIAL_JTAG_JTAG_IN_FLUSH_INT_RAW The raw interrupt status of USB_SERIAL_JTAG_JTAG_IN_FLUSH_INT. (R/WTC/SS)

USB_SERIAL_JTAG_SOF_INT_RAW The raw interrupt status of USB_SERIAL_JTAG_SOF_INT. (R/WTC/SS)

USB_SERIAL_JTAG_SERIAL_OUT_RECV_PKT_INT_RAW The raw interrupt status of USB_SERIAL_JTAG_SERIAL_OUT_RECV_PKT_INT. (R/WTC/SS)

USB_SERIAL_JTAG_SERIAL_IN_EMPTY_INT_RAW The raw interrupt status of USB_SERIAL_JTAG_SERIAL_IN_EMPTY_INT. (R/WTC/SS)

USB_SERIAL_JTAG_PID_ERR_INT_RAW The raw interrupt status of USB_SERIAL_JTAG_PID_ERR_INT. (R/WTC/SS)

USB_SERIAL_JTAG_CRC5_ERR_INT_RAW The raw interrupt status of USB_SERIAL_JTAG_CRC5_ERR_INT. (R/WTC/SS)

USB_SERIAL_JTAG_CRC16_ERR_INT_RAW The raw interrupt status of USB_SERIAL_JTAG_CRC16_ERR_INT. (R/WTC/SS)

USB_SERIAL_JTAG_STUFF_ERR_INT_RAW The raw interrupt status of USB_SERIAL_JTAG_STUFF_ERR_INT. (R/WTC/SS)

USB_SERIAL_JTAG_IN_TOKEN_REC_IN_EP1_INT_RAW The raw interrupt status of USB_SERIAL_JTAG_IN_TOKEN_REC_IN_EP1_INT. (R/WTC/SS)

USB_SERIAL_JTAG_USB_BUS_RESET_INT_RAW The raw interrupt status of USB_SERIAL_JTAG_USB_BUS_RESET_INT. (R/WTC/SS)

USB_SERIAL_JTAG_OUT_EP1_ZERO_PAYLOAD_INT_RAW The raw interrupt status of USB_SERIAL_JTAG_OUT_EP1_ZERO_PAYLOAD_INT. (R/WTC/SS)

Continued on the next page...

Register 54.12. USB_SERIAL_JTAG_INT_RAW_REG (0x0008)

Continued from the previous page...

USB_SERIAL_JTAG_OUT_EP2_ZERO_PAYLOAD_INT_RAW The raw interrupt status of USB_SERIAL_JTAG_OUT_EP2_ZERO_PAYLOAD_INT. (R/WTC/SS)

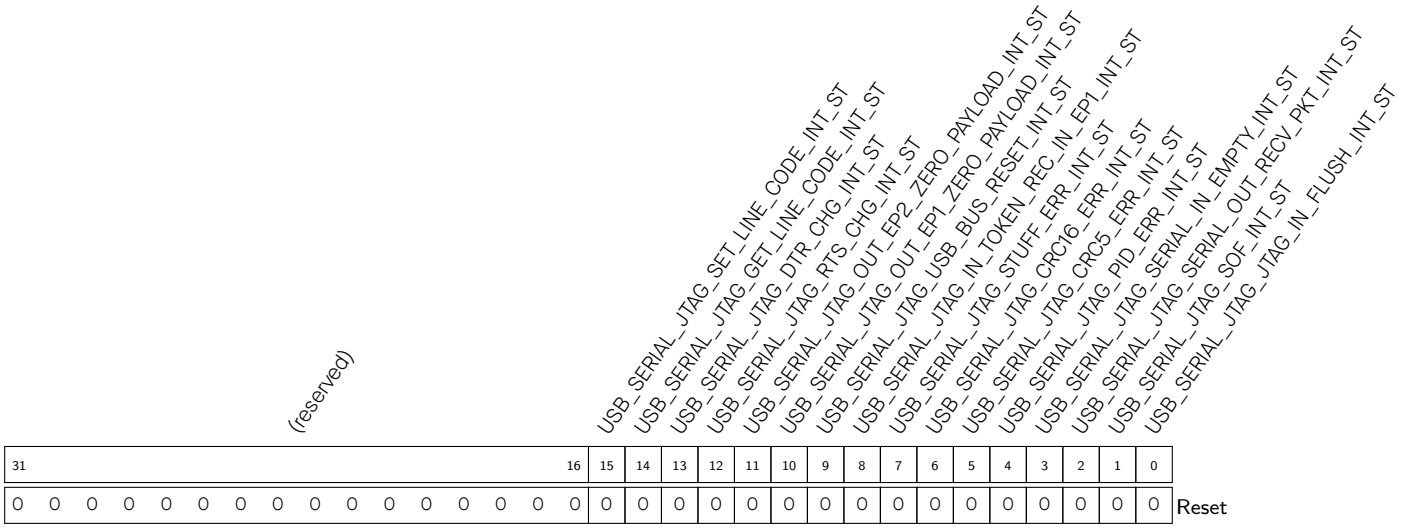
USB_SERIAL_JTAG_RTS_CHG_INT_RAW The raw interrupt status of USB_SERIAL_JTAG_RTS_CHG_INT. (R/WTC/SS)

USB_SERIAL_JTAG_DTR_CHG_INT_RAW The raw interrupt status of USB_SERIAL_JTAG_DTR_CHG_INT. (R/WTC/SS)

USB_SERIAL_JTAG_GET_LINE_CODE_INT_RAW The raw interrupt status of USB_SERIAL_JTAG_GET_LINE_CODE_INT. (R/WTC/SS)

USB_SERIAL_JTAG_SET_LINE_CODE_INT_RAW The raw interrupt status of USB_SERIAL_JTAG_SET_LINE_CODE_INT. (R/WTC/SS)

Register 54.13. USB_SERIAL_JTAG_INT_ST_REG (0x000C)



USB_SERIAL_JTAG_JTAG_IN_FLUSH_INT_ST The masked interrupt status of USB_SERIAL_JTAG_JTAG_IN_FLUSH_INT. (RO)

USB_SERIAL_JTAG_SOF_INT_ST The masked interrupt status of USB_SERIAL_JTAG_SOF_INT. (RO)

USB_SERIAL_JTAG_SERIAL_OUT_RECV_PKT_INT_ST The masked interrupt status of the USB_SERIAL_JTAG_SERIAL_OUT_RECV_PKT_INT. (RO)

USB_SERIAL_JTAG_SERIAL_IN_EMPTY_INT_ST The masked interrupt status of USB_SERIAL_JTAG_SERIAL_IN_EMPTY_INT. (RO)

USB_SERIAL_JTAG_PID_ERR_INT_ST The masked interrupt status of USB_SERIAL_JTAG_PID_ERR_INT. (RO)

USB_SERIAL_JTAG_CRC5_ERR_INT_ST The masked interrupt status of USB_SERIAL_JTAG_CRC5_ERR_INT. (RO)

USB_SERIAL_JTAG_CRC16_ERR_INT_ST The masked interrupt status of USB_SERIAL_JTAG_CRC16_ERR_INT. (RO)

USB_SERIAL_JTAG_STUFF_ERR_INT_ST The masked interrupt status of USB_SERIAL_JTAG_STUFF_ERR_INT. (RO)

USB_SERIAL_JTAG_IN_TOKEN_REC_IN_EP1_INT_ST The masked interrupt status of USB_SERIAL_JTAG_IN_TOKEN_REC_IN_EP1_INT. (RO)

USB_SERIAL_JTAG_USB_BUS_RESET_INT_ST The masked interrupt status of USB_SERIAL_JTAG_USB_BUS_RESET_INT. (RO)

USB_SERIAL_JTAG_OUT_EP1_ZERO_PAYLOAD_INT_ST The masked interrupt status of USB_SERIAL_JTAG_OUT_EP1_ZERO_PAYLOAD_INT (RO)

USB_SERIAL_JTAG_OUT_EP2_ZERO_PAYLOAD_INT_ST The masked interrupt status of USB_SERIAL_JTAG_OUT_EP2_ZERO_PAYLOAD_INT. (RO)

Continued on the next page...

Register 54.13. USB_SERIAL_JTAG_INT_ST_REG (0x000C)

Continued from the previous page...

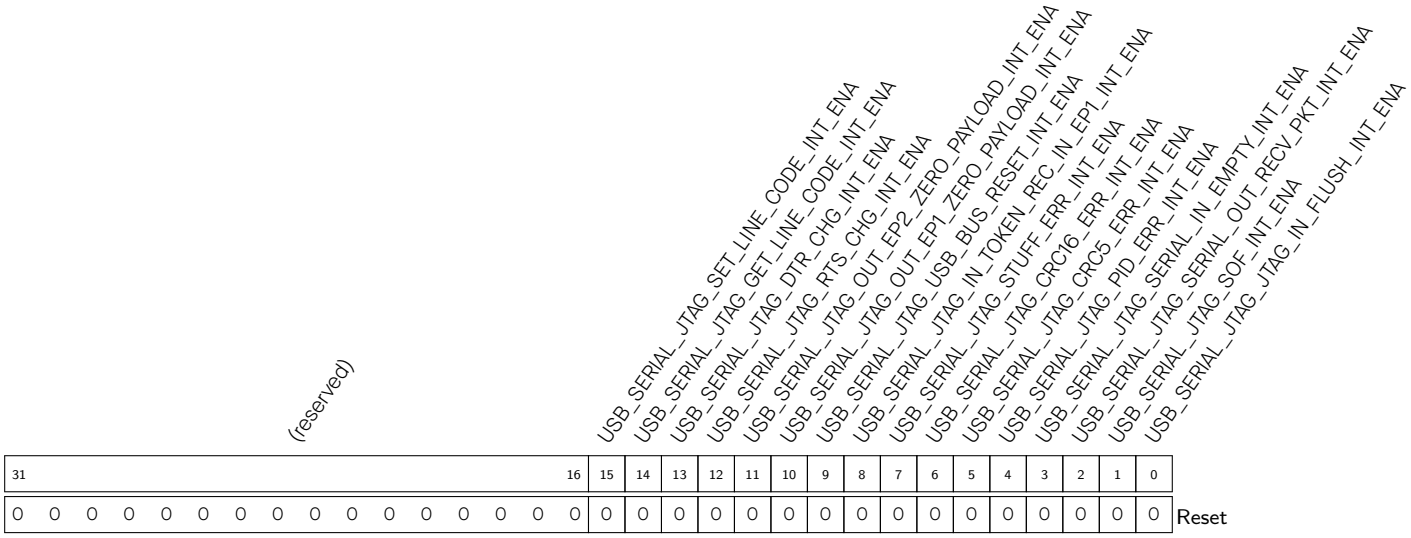
USB_SERIAL_JTAG_RTS_CHG_INT_ST The masked interrupt status of USB_SERIAL_JTAG_RTS_CHG_INT. (RO)

USB_SERIAL_JTAG_DTR_CHG_INT_ST The masked interrupt status of USB_SERIAL_JTAG_DTR_CHG_INT. (RO)

USB_SERIAL_JTAG_GET_LINE_CODE_INT_ST The masked interrupt status of USB_SERIAL_JTAG_GET_LINE_CODE_INT. (RO)

USB_SERIAL_JTAG_SET_LINE_CODE_INT_ST The masked interrupt status of USB_SERIAL_JTAG_SET_LINE_CODE_INT. (RO)

Register 54.14. USB_SERIAL_JTAG_INT_ENA_REG (0x0010)



USB_SERIAL_JTAG_JTAG_IN_FLUSH_INT_ENA Write 1 to enable USB_SERIAL_JTAG_JTAG_IN_FLUSH_INT. (R/W)

USB_SERIAL_JTAG_SOF_INT_ENA Write 1 to enable USB_SERIAL_JTAG_SOF_INT. (R/W)

USB_SERIAL_JTAG_SERIAL_OUT_RECV_PKT_INT_ENA Write 1 to enable USB_SERIAL_JTAG_SERIAL_OUT_RECV_PKT_INT. (R/W)

USB_SERIAL_JTAG_SERIAL_IN_EMPTY_INT_ENA Write 1 to enable USB_SERIAL_JTAG_SERIAL_IN_EMPTY_INT. (R/W)

USB_SERIAL_JTAG_PID_ERR_INT_ENA Write 1 to enable USB_SERIAL_JTAG_PID_ERR_INT. (R/W)

USB_SERIAL_JTAG_CRC5_ERR_INT_ENA Write 1 to enable USB_SERIAL_JTAG_CRC5_ERR_INT. (R/W)

USB_SERIAL_JTAG_CRC16_ERR_INT_ENA Write 1 to enable USB_SERIAL_JTAG_CRC16_ERR_INT. (R/W)

USB_SERIAL_JTAG_STUFF_ERR_INT_ENA Write 1 to enable USB_SERIAL_JTAG_STUFF_ERR_INT. (R/W)

USB_SERIAL_JTAG_IN_TOKEN_REC_IN_EP1_INT_ENA Write 1 to enable USB_SERIAL_JTAG_IN_TOKEN_REC_IN_EP1_INT. (R/W)

USB_SERIAL_JTAG_USB_BUS_RESET_INT_ENA Write 1 to enable USB_SERIAL_JTAG_USB_BUS_RESET_INT. (R/W)

USB_SERIAL_JTAG_OUT_EP1_ZERO_PAYLOAD_INT_ENA Write 1 to enable USB_SERIAL_JTAG_OUT_EP1_ZERO_PAYLOAD_INT. (R/W)

USB_SERIAL_JTAG_OUT_EP2_ZERO_PAYLOAD_INT_ENA Write 1 to enable USB_SERIAL_JTAG_OUT_EP2_ZERO_PAYLOAD_INT. (R/W)

Continued on the next page...

Register 54.14. USB_SERIAL_JTAG_INT_ENA_REG (0x0010)

Continued from the previous page...

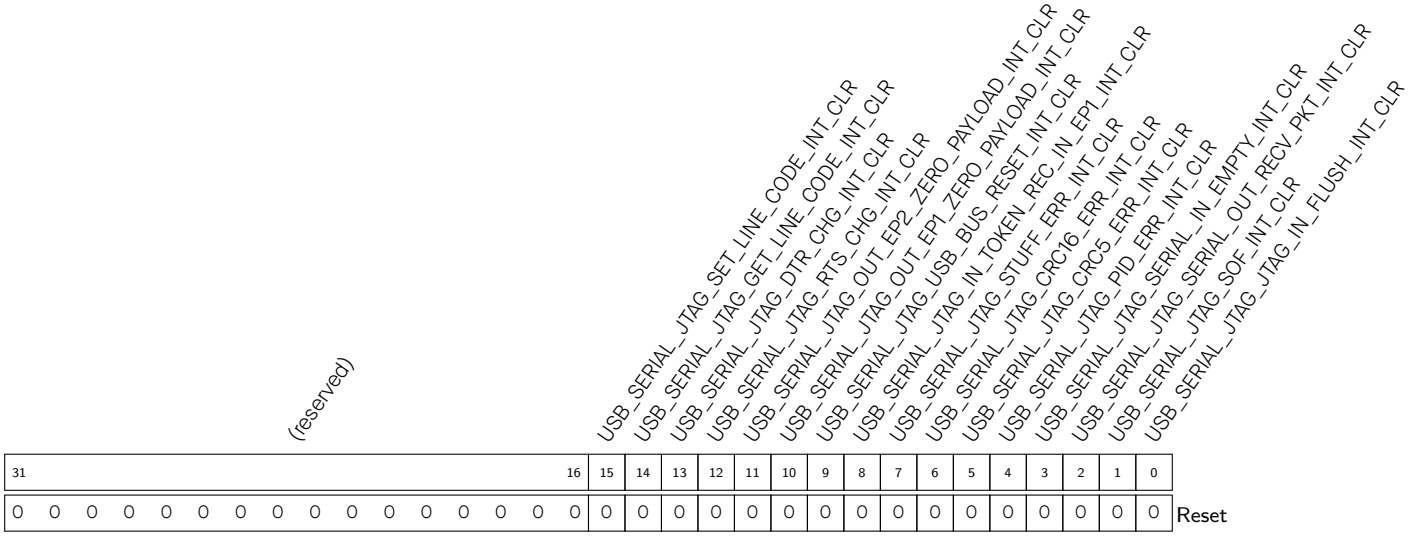
USB_SERIAL_JTAG_RTS_CHG_INT_ENA Write 1 to enable USB_SERIAL_JTAG_RTS_CHG_INT. (R/W)

USB_SERIAL_JTAG_DTR_CHG_INT_ENA Write 1 to enable USB_SERIAL_JTAG_DTR_CHG_INT. (R/W)

USB_SERIAL_JTAG_GET_LINE_CODE_INT_ENA Write 1 to enable USB_SERIAL_JTAG_GET_LINE_CODE_INT. (R/W)

USB_SERIAL_JTAG_SET_LINE_CODE_INT_ENA Write 1 to enable USB_SERIAL_JTAG_SET_LINE_CODE_INT. (R/W)

Register 54.15. USB_SERIAL_JTAG_INT_CLR_REG (0x0014)



USB_SERIAL_JTAG_JTAG_IN_FLUSH_INT_CLR Write 1 to clear USB_SERIAL_JTAG_JTAG_IN_FLUSH_INT. (WT)

USB_SERIAL_JTAG_SOF_INT_CLR Write 1 to clear USB_SERIAL_JTAG_SOF_INT. (WT)

USB_SERIAL_JTAG_SERIAL_OUT_RECV_PKT_INT_CLR Write 1 to clear USB_SERIAL_JTAG_SERIAL_OUT_RECV_PKT_INT. (WT)

USB_SERIAL_JTAG_SERIAL_IN_EMPTY_INT_CLR Write 1 to clear USB_SERIAL_JTAG_SERIAL_IN_EMPTY_INT. (WT)

USB_SERIAL_JTAG_PID_ERR_INT_CLR Write 1 to clear USB_SERIAL_JTAG_PID_ERR_INT. (WT)

USB_SERIAL_JTAG_CRC5_ERR_INT_CLR Write 1 to clear USB_SERIAL_JTAG_CRC5_ERR_INT. (WT)

USB_SERIAL_JTAG_CRC16_ERR_INT_CLR Write 1 to clear USB_SERIAL_JTAG_CRC16_ERR_INT. (WT)

USB_SERIAL_JTAG_STUFF_ERR_INT_CLR Write 1 to clear USB_SERIAL_JTAG_STUFF_ERR_INT. (WT)

USB_SERIAL_JTAG_IN_TOKEN_REC_IN_EP1_INT_CLR Write 1 to clear USB_SERIAL_JTAG_IN_TOKEN_REC_IN_EP1_INT. (WT)

USB_SERIAL_JTAG_USB_BUS_RESET_INT_CLR Write 1 to clear USB_SERIAL_JTAG_USB_BUS_RESET_INT. (WT)

USB_SERIAL_JTAG_OUT_EP1_ZERO_PAYLOAD_INT_CLR Write 1 to clear USB_SERIAL_JTAG_OUT_EP1_ZERO_PAYLOAD_INT. (WT)

USB_SERIAL_JTAG_OUT_EP2_ZERO_PAYLOAD_INT_CLR Write 1 to clear USB_SERIAL_JTAG_OUT_EP2_ZERO_PAYLOAD_INT. (WT)

Continued on the next page...

Register 54.15. USB_SERIAL_JTAG_INT_CLR_REG (0x0014)

Continued from the previous page...

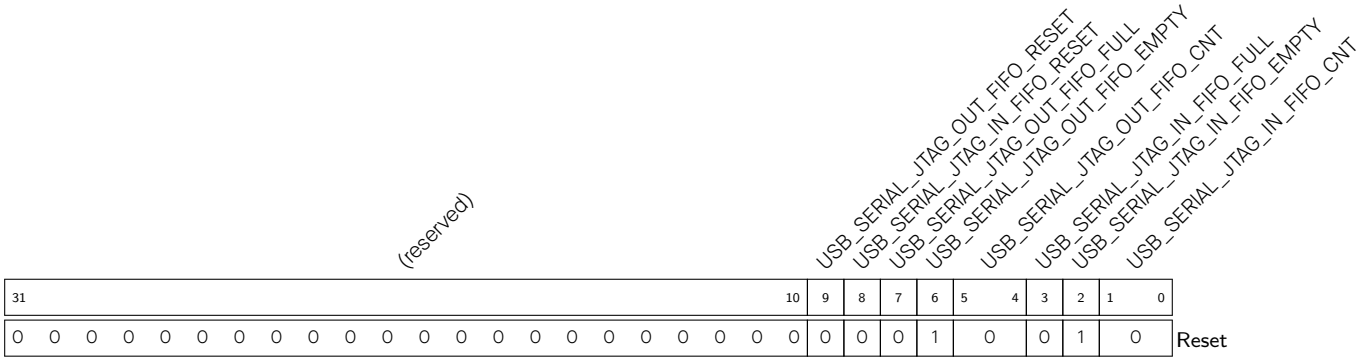
USB_SERIAL_JTAG_RTS_CHG_INT_CLR Write 1 to clear USB_SERIAL_JTAG_RTS_CHG_INT. (WT)

USB_SERIAL_JTAG_DTR_CHG_INT_CLR Write 1 to clear USB_SERIAL_JTAG_DTR_CHG_INT. (WT)

USB_SERIAL_JTAG_GET_LINE_CODE_INT_CLR Write 1 to clear USB_SERIAL_JTAG_GET_LINE_CODE_INT. (WT)

USB_SERIAL_JTAG_SET_LINE_CODE_INT_CLR Write 1 to clear USB_SERIAL_JTAG_SET_LINE_CODE_INT. (WT)

Register 54.16. USB_SERIAL_JTAG_JFIFO_ST_REG (0x0020)



USB_SERIAL_JTAG_IN_FIFO_CNT Represents JTAG IN FIFO counter. (RO)

USB_SERIAL_JTAG_IN_FIFO_EMPTY Represents whether JTAG IN FIFO is empty.

0: Not empty

1: Empty

(RO)

USB_SERIAL_JTAG_IN_FIFO_FULL Represents whether JTAG IN FIFO is full.

0: Not full

1: Full

(RO)

USB_SERIAL_JTAG_OUT_FIFO_CNT Represents JTAG OUT FIFO counter. (RO)

USB_SERIAL_JTAG_OUT_FIFO_EMPTY Represents whether JTAG OUT FIFO is empty.

0: Not empty

1: Empty

(RO)

USB_SERIAL_JTAG_OUT_FIFO_FULL Represents whether JTAG OUT FIFO is full.

0: Not full

1: Full

(RO)

USB_SERIAL_JTAG_IN_FIFO_RESET Configures whether to reset JTAG IN FIFO.

0: No effect

1: Reset

(R/W)

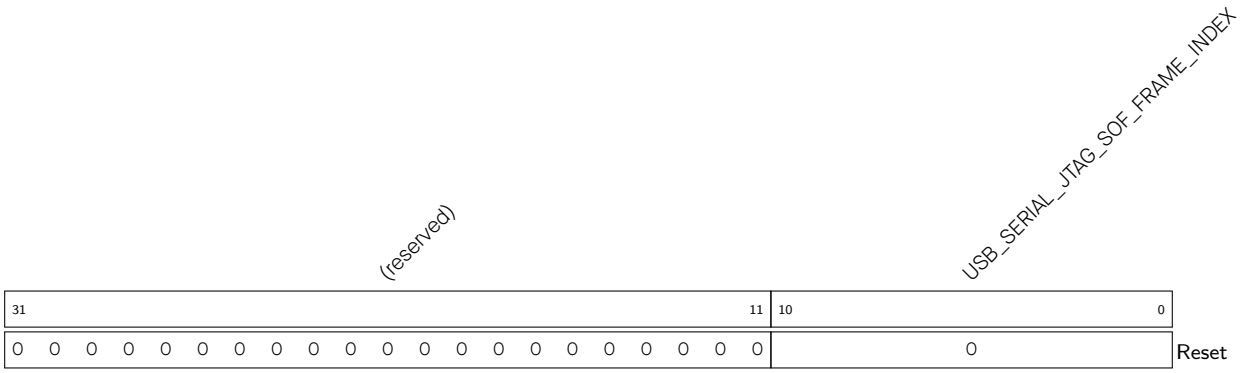
USB_SERIAL_JTAG_OUT_FIFO_RESET Configures whether to reset JTAG OUT FIFO.

0: No effect

1: Reset

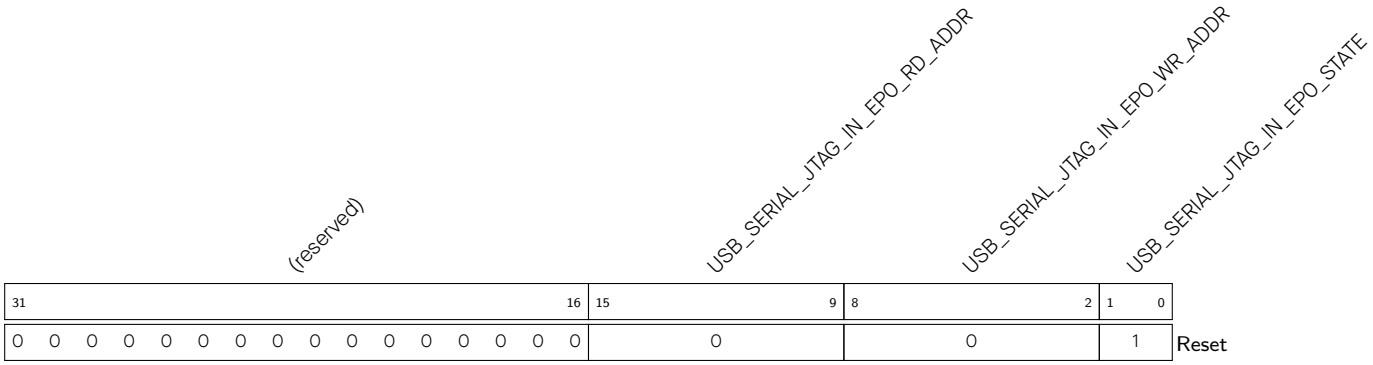
(R/W)

Register 54.17. USB_SERIAL_JTAG_FRAM_NUM_REG (0x0024)



USB_SERIAL_JTAG_SOF_FRAME_INDEX Represents frame index of received SOF frame. (RO)

Register 54.18. USB_SERIAL_JTAG_IN_EP0_ST_REG (0x0028)

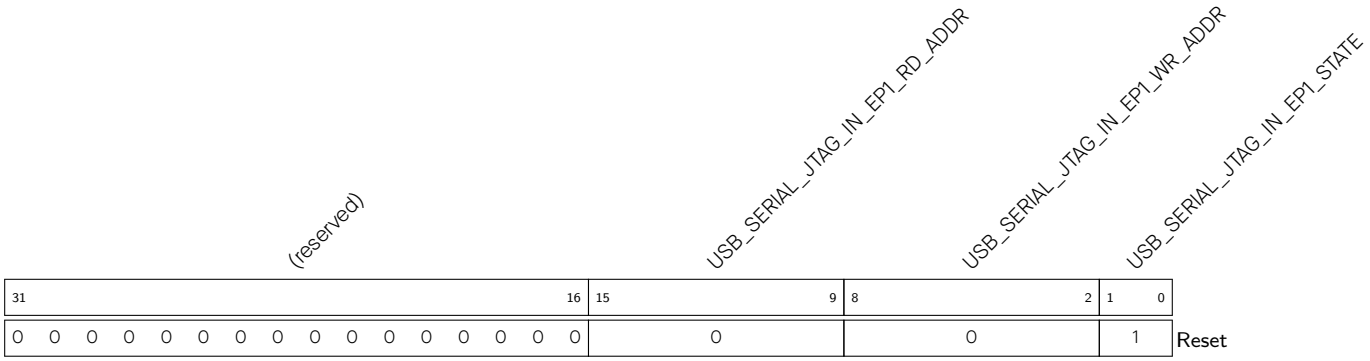


USB_SERIAL_JTAG_IN_EP0_STATE Represents state of IN Endpoint 0. (RO)

USB_SERIAL_JTAG_IN_EP0_WR_ADDR Represents write data address of IN endpoint 0. (RO)

USB_SERIAL_JTAG_IN_EP0_RD_ADDR Represents read data address of IN endpoint 0. (RO)

Register 54.19. USB_SERIAL_JTAG_IN_EP1_ST_REG (0x002C)

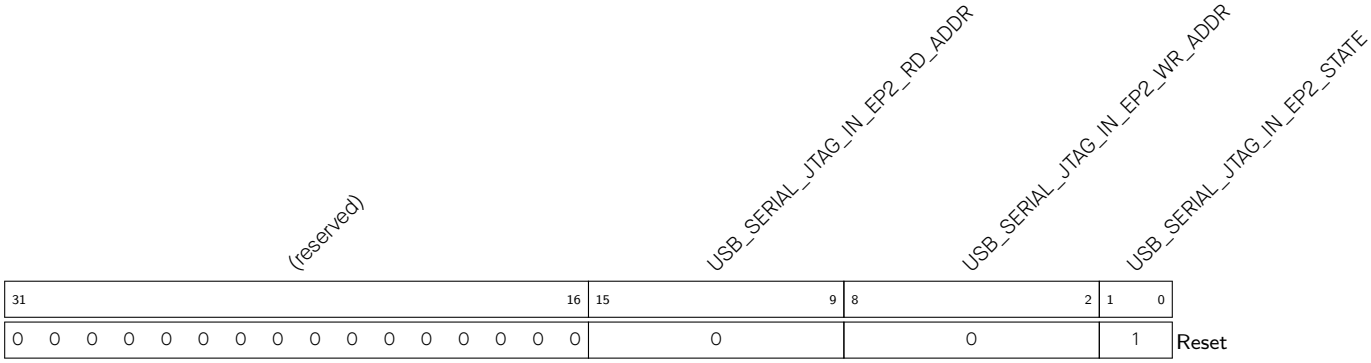


USB_SERIAL_JTAG_IN_EP1_STATE Represents state of IN Endpoint 1. (RO)

USB_SERIAL_JTAG_IN_EP1_WR_ADDR Represents write data address of IN endpoint 1. (RO)

USB_SERIAL_JTAG_IN_EP1_RD_ADDR Represents read data address of IN endpoint 1. (RO)

Register 54.20. USB_SERIAL_JTAG_IN_EP2_ST_REG (0x0030)

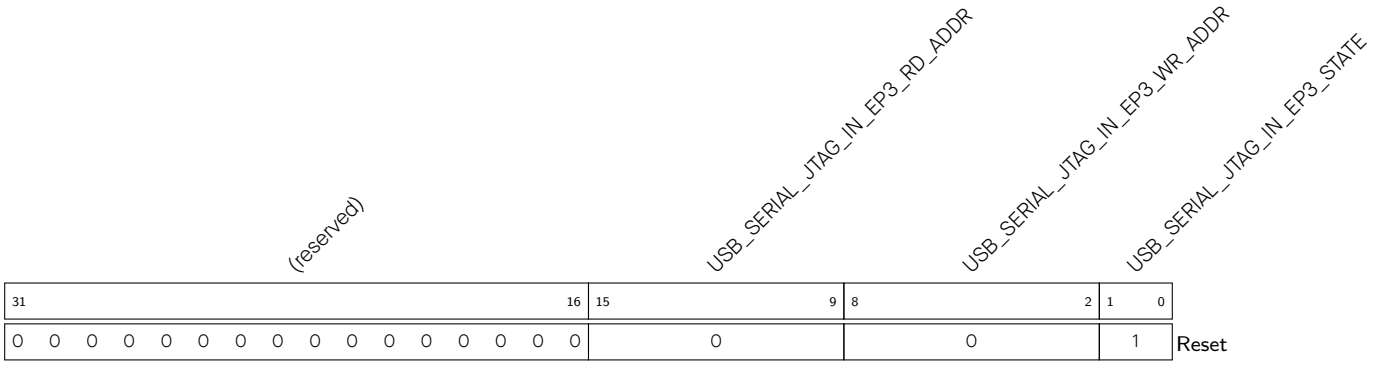


USB_SERIAL_JTAG_IN_EP2_STATE Represents state of IN Endpoint 2. (RO)

USB_SERIAL_JTAG_IN_EP2_WR_ADDR Represents write data address of IN endpoint 2. (RO)

USB_SERIAL_JTAG_IN_EP2_RD_ADDR Represents read data address of IN endpoint 2. (RO)

Register 54.21. USB_SERIAL_JTAG_IN_EP3_ST_REG (0x0034)

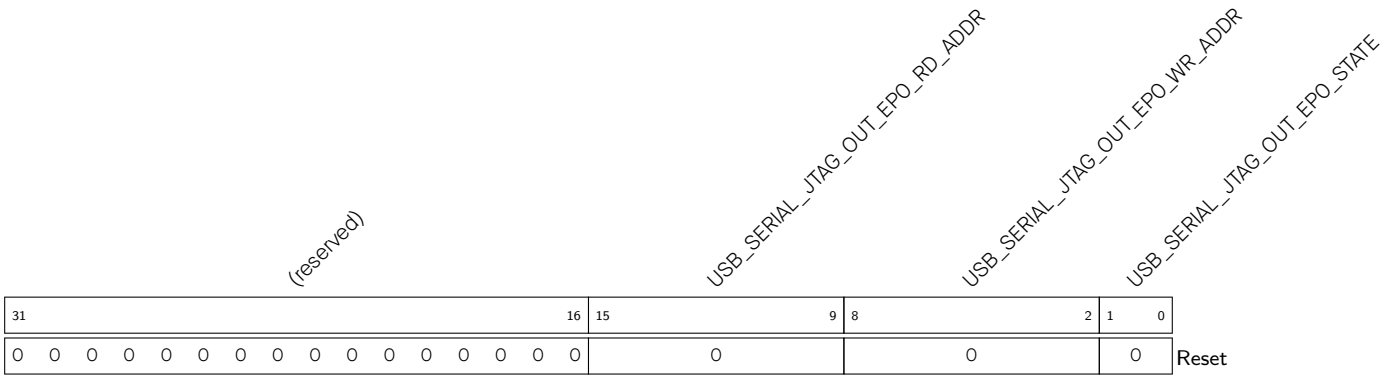


USB_SERIAL_JTAG_IN_EP3_STATE Represents state of IN Endpoint 3. (RO)

USB_SERIAL_JTAG_IN_EP3_WR_ADDR Represents write data address of IN endpoint 3. (RO)

USB_SERIAL_JTAG_IN_EP3_RD_ADDR Represents read data address of IN endpoint 3. (RO)

Register 54.22. USB_SERIAL_JTAG_OUT_EP0_ST_REG (0x0038)



USB_SERIAL_JTAG_OUT_EP0_STATE Represents state of OUT Endpoint 0. (RO)

USB_SERIAL_JTAG_OUT_EP0_WR_ADDR Represents write data address of OUT endpoint 0. When USB_SERIAL_JTAG_SERIAL_OUT_RECV_PKT_INT is detected, there are (USB_SERIAL_JTAG_OUT_EP0_WR_ADDR − 2) bytes data in OUT endpoint 0. (RO)

USB_SERIAL_JTAG_OUT_EP0_RD_ADDR Represents read data address of OUT endpoint 0. (RO)

Register 54.23. USB_SERIAL_JTAG_OUT_EP1_ST_REG (0x003C)



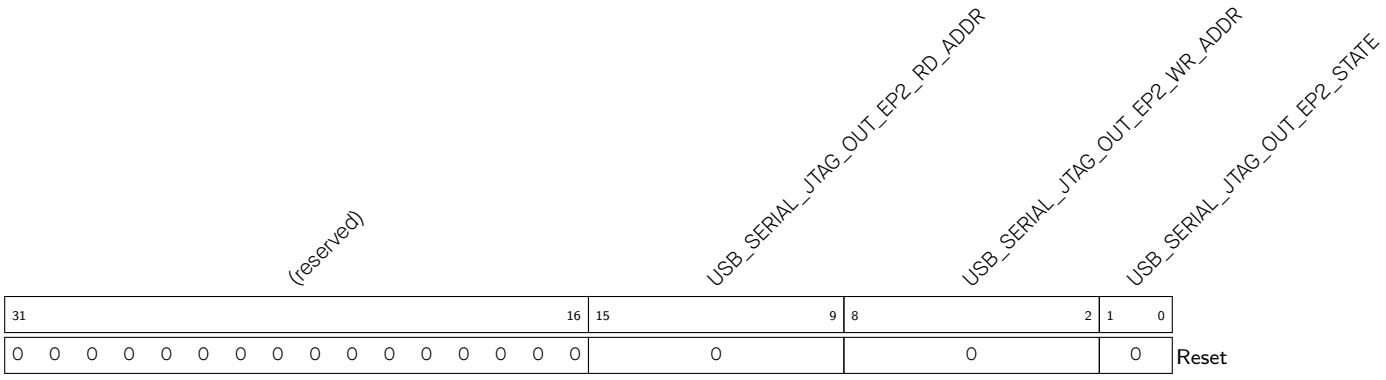
USB_SERIAL_JTAG_OUT_EP1_STATE Represents state of OUT Endpoint 1. (RO)

USB_SERIAL_JTAG_OUT_EP1_WR_ADDR Represents write data address of OUT endpoint 1. When USB_SERIAL_JTAG_SERIAL_OUT_RECV_PKT_INT is detected, there are (USB_SERIAL_JTAG_OUT_EP1_WR_ADDR − 2) bytes data in OUT endpoint 1. (RO)

USB_SERIAL_JTAG_OUT_EP1_RD_ADDR Represents read data address of OUT endpoint 1. (RO)

USB_SERIAL_JTAG_OUT_EP1_REC_DATA_CNT Represents data count in OUT endpoint 1 when one packet is received. (RO)

Register 54.24. USB_SERIAL_JTAG_OUT_EP2_ST_REG (0x0040)

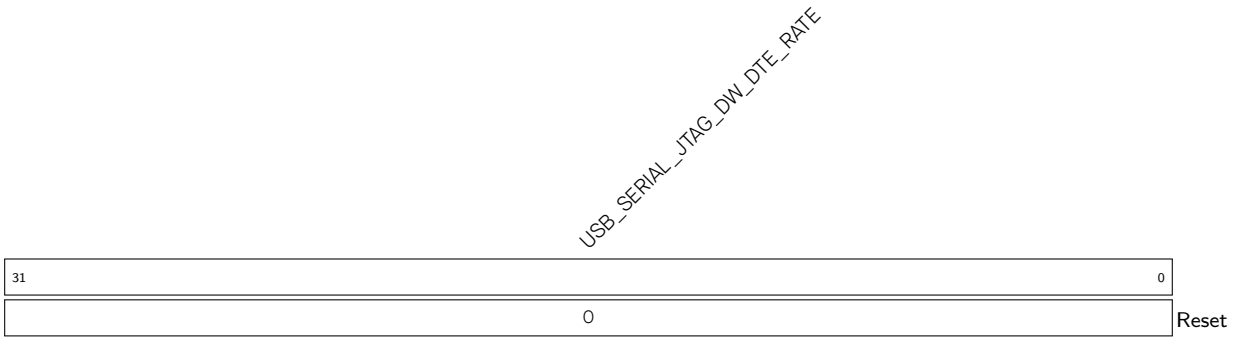


USB_SERIAL_JTAG_OUT_EP2_STATE Represents state of OUT Endpoint 2. (RO)

USB_SERIAL_JTAG_OUT_EP2_WR_ADDR Represents write data address of OUT endpoint 2. When USB_SERIAL_JTAG_SERIAL_OUT_RECV_PKT_INT is detected there are (USB_SERIAL_JTAG_OUT_EP2_WR_ADDR − 2) bytes data in OUT endpoint 2. (RO)

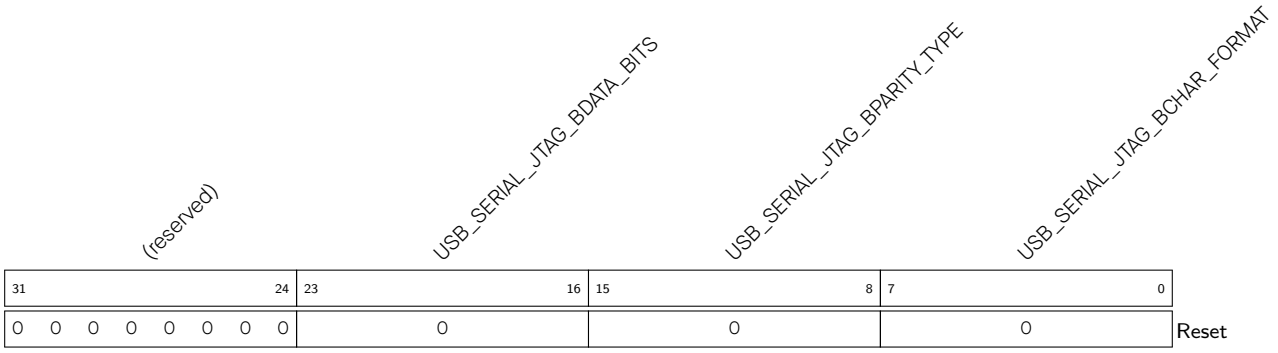
USB_SERIAL_JTAG_OUT_EP2_RD_ADDR Represents read data address of OUT endpoint 2. (RO)

Register 54.25. USB_SERIAL_JTAG_SET_LINE_CODE_W0_REG (0x0050)



USB_SERIAL_JTAG_DW_DTE_RATE Represents the value of dwDTERate set by host through SET_LINE_CODING command. (RO)

Register 54.26. USB_SERIAL_JTAG_SET_LINE_CODE_W1_REG (0x0054)

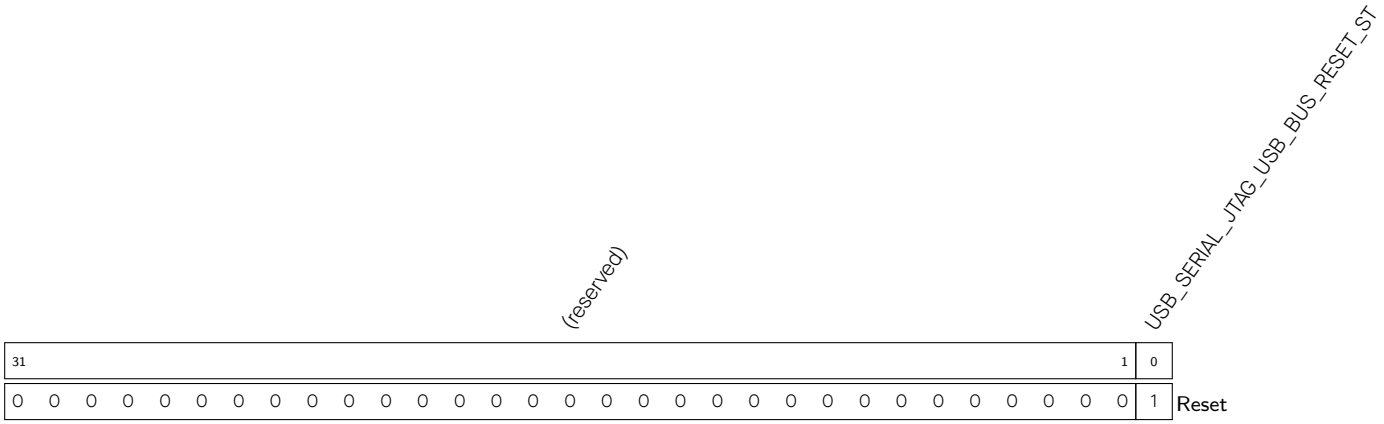


USB_SERIAL_JTAG_BCHAR_FORMAT Represents the value of bCharFormat set by host through SET_LINE_CODING command. (RO)

USB_SERIAL_JTAG_BPARITY_TYPE Represents the value of bParityTpye set by host through SET_LINE_CODING command. (RO)

USB_SERIAL_JTAG_BDATA_BITS Represents the value of bDataBits set by host through SET_LINE_CODING command. (RO)

Register 54.27. USB_SERIAL_JTAG_BUS_RESET_ST_REG (0x0068)



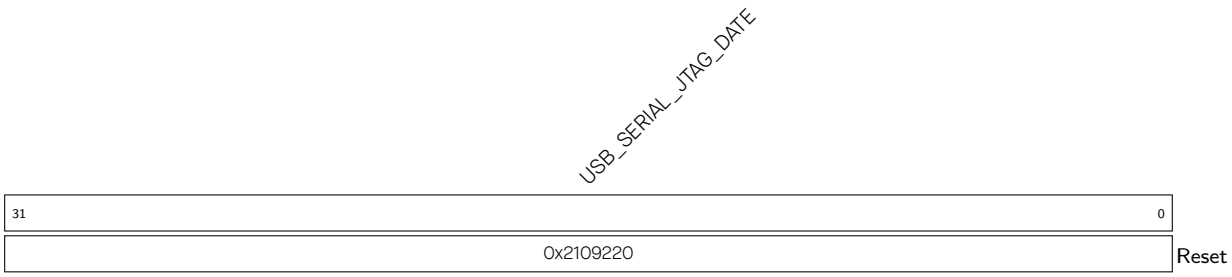
USB_SERIAL_JTAG_USB_BUS_RESET_ST Represents whether USB bus reset is released.

0: USB Serial/JTAG is in USB bus reset status

1: USB bus reset is released

(RO)

Register 54.28. USB_SERIAL_JTAG_DATE_REG (0x0088)



USB_SERIAL_JTAG_DATE Version control register. (R/W)

Chapter 55

Ethernet Media Access Controller (EMAC)

55.1 Overview

By using the external Ethernet PHY (physical layer), ESP32-P4 can send and receive data via Ethernet MAC (Media Access Controller) according to the IEEE 802.3 standard, as Figure 55.1-1 shows. Ethernet is currently the most commonly used network protocol that controls how data is transmitted over local- and wide-area networks, abbreviated as LAN and WAN, respectively.

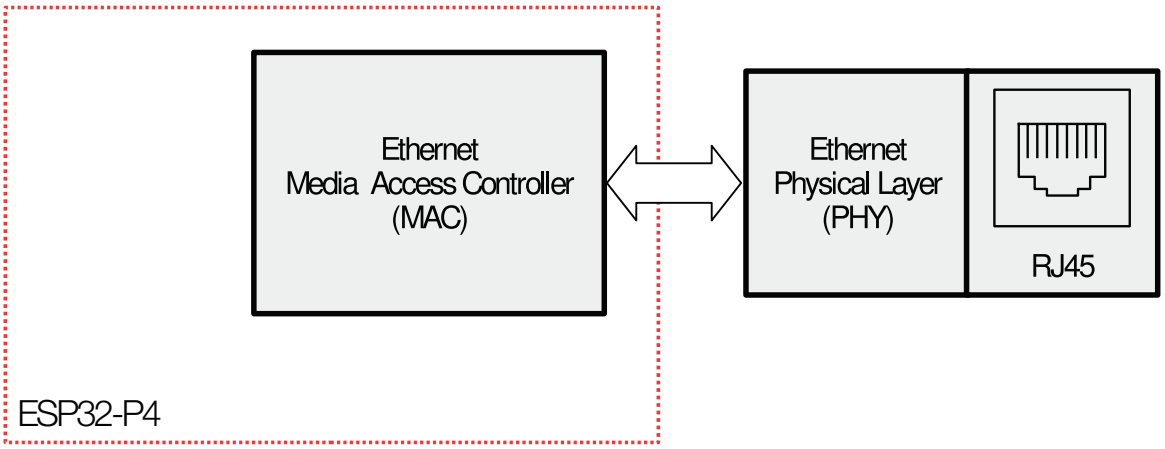


Figure 55.1-1. Ethernet MAC Functionality Overview

ESP32-P4 Ethernet MAC complies with the following standards:

- IEEE 802.3-2002 for Ethernet MAC
- IEEE 1588-2008 standard for precise networked clock synchronization
- IEEE 802.3 standard Media Independent Interface (MII) and Reduced Media Independent Interface (RMII)
- IEEE 802.3az-2010 for Energy Efficient Ethernet
- IEEE 802.1Q for VLAN frame format

55.2 Features

- Data rates of 10/100 Mbit/s through an external PHY interface
- Communication with an external Fast Ethernet PHY through IEEE 802.3-compliant MII and RMII interfaces
- Full-duplex and half-duplex modes

  - Carrier Sense Multiple Access or Collision Detection (CSMA/CD) protocol in half-duplex mode
  - IEEE 802.3x flow control in full-duplex mode
  - Optional forwarding of received pause control frame to the user application in full-duplex mode
  - Back-pressure flow control in half-duplex mode
  - Automatic transmission of zero-quanta pause frame on deassertion of flow control input in full-duplex mode
- Preamble and start-of-frame data (SFD) insertion in Transmit, and deletion in Receive paths
- Automatic CRC and padding (all 0) generation controllable on a per-frame basis
- Options for automatic padding generation for data below the minimum frame length
- Programmable frame length supporting jumbo frames of up to 16 KB
- Programmable inter-frame gap (IFG) from 40 to 96 bit times in steps of 8)
- Supports a variety of flexible address filtering modes:
  - Up to eight 48-bit perfect address filters with masks for each byte
  - Up to eight 48-bit source address (SA) address comparison check with masks for each byte
  - Option to pass all multicast addressed frames
  - Promiscuous mode support to pass all frames without any filtering for network monitoring
  - Passes all incoming packets (as per filter) with a status report
- Separate 32-bit status returned for transmission and reception packets
- Supports IEEE 802.1Q VLAN tag detection for reception frames
- Separate transmission, reception, and control interfaces for the application
- Management Data Input/Output (MDIO) interface for PHY device configuration and management
- Supports checksum off-load for received IPv4 and TCP packets encapsulated by the Ethernet frame
- Supports checking IPv4 header checksum and TCP, UDP, or ICMP checksum encapsulated in IPv4 or IPv6 datagrams
- Supports 64-bit timestamp on each transmit and receive frame (see IEEE 1588-2008)
- Supports Energy Efficient Ethernet (see IEEE 802.3az-2010)
- CRC replacement, SA insertion or replacement, and VLAN insertion, replacement or deletion in transmit frames
- Two FIFOs: 256-byte TX FIFO and 256-byte RX FIFO
- Receive status vectors inserted into RX FIFO after the EOF (end of frame) transfer, which enables multiple-frame storage in RX FIFO without requiring another FIFO to store those frames' receive status
- Option to forward good runt frames
- Supports statistics by generating pulses for frames dropped or corrupted due to overflow in RX FIFO
- Automatic re-transmission of collision frames (subject to certain conditions, see Section 55.4.1.1)

- Discarding frames in cases of late collisions, excessive collisions, excessive deferrals, and underflow conditions
- Software control to flush TX FIFO

55.3 Ethernet MAC Architecture

Figure 55.3-1 shows the block diagram of the Ethernet MAC.

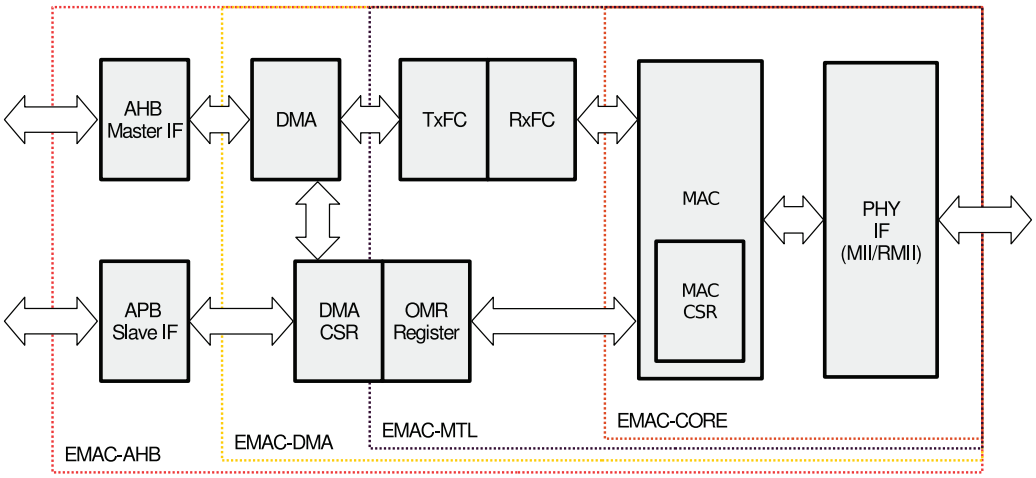


Figure 55.3-1. Ethernet MAC Block Diagram

Ethernet MAC consists of three layers: EMAC_CORE (MAC Core Layer), EMAC_MTL (MAC Transition Layer), and EMAC_DMA (Direct Memory Access). Each of these three layers has two directions: TX and RX. They are connected to the system through the Advanced High Performance Bus (AHB) and the Advanced Peripheral Bus (APB) on the chip. Off the chip, they communicate with the external PHY through the MII and RMII interfaces to establish an Ethernet connection.

55.4 Functional Description

55.4.1 EMAC_CORE

The MAC supports two interfaces (see Section 55.4.4) towards the PHY chip. The PHY interface can be selected only once after the chip reset. The MAC core communicates with the application side (DMA side) using the MAC Transmit Interface (MTI), MAC Receive Interface (MRI), and the MAC Control Interface (MCI).

55.4.1.1 Transmission

Transmission is initiated when the MTL application pushes in data with the SOF signal (start of frame) asserted. When the SOF signal is detected, the MAC accepts the data and begins transmitting to the RMII or MII. The time required to transmit the frame data to the RMII or MII after the application initiates transmission is variable, depending on delay factors like IFG delay, time to transmit preamble or SFD (Start Frame Delimiter), and any back-off delays for half-duplex mode. Until then, the MAC does not accept the data received from MTL by deasserting the ready signal.

After the EOF (end of frame) is transmitted to the MAC, the MAC completes normal transmission and then gives the Transmit Status to the MTL. If a normal collision (in half-duplex mode) occurs during transmission, the MAC core makes valid the Transmit Status to the MTL. It will then accept and drop all further data until the next SOF is received. The MTL block should retransmit the same frame from SOF on observing a retry request (in the Status) from the MAC.

The MAC issues an underflow status if the MTL is not able to provide the data continuously during the transmission. During the normal transmission of a frame from the MTL, if the MAC receives an SOF without getting an EOF for the previous frame, then the SOF is ignored and the new frame is considered as a continuation of the previous frame.

Transmit Flow Control

In full-duplex mode, when the Transmit Flow Control Enable bit (TFCE) is set to 1, the MAC generates pause frames and transmits them as necessary, with the calculated CRC appended.

Pause frame generation can be initiated in two ways. When the application sets the Flow Control Busy bit (FCBBA) to 1, or when the RX FIFO is full, a pause frame is transmitted.

- If the application has requested flow control by setting the FCBBA bit to 1, the MAC will generate and transmit a single pause frame. The value of the pause time in the generated frame is the pause time value programmed in the PAUSE_TIME field. If the application wants to extend or end the pause prior to the time specified in the previously transmitted pause frame, it must request another pause frame transmission after programming the pause time field PAUSE_TIME with an appropriate value.
- If the application has requested flow control when the RX FIFO is full, the MAC will generate and transmit a pause frame. The value of the pause time in the generated frame is the pause time value programmed in the PAUSE_TIME field. If the RX FIFO remains full at a configurable number of slot times (PLT) before the pause time runs out, a second pause frame will be transmitted. The process will be repeated as long as the RX FIFO remains full. If the FIFO is no longer full prior to the sampling time, the MAC will send a pause frame with zero pause time to indicate to the remote end that the RX buffer is ready to receive new data frames.

Retransmission During Collision

In half-duplex mode, a collision may occur on the MAC line interface when frames are transmitted to the MAC. The MAC may indicate a retry attempt by giving the status even before the EOF is transferred. Then the MAC will enable the retransmission by popping out the frame again from the FIFO. After more than 96 bytes are popped towards the MAC core, the FIFO controller frees the space in the FIFO, and makes it available to the DMA to push in more data. This means that the retransmission is not possible after this threshold is crossed or when the MAC core indicates a late-collision event.

The MAC transmitter may abort the transmission of a frame because of collision, TX FIFO underflow, loss of carrier, jabber timeout, no carrier, excessive deferral, or late collision. When frame transmission is aborted because of collision, the MAC requests retransmission of the frame.

Transmit Status Word

At the end of the Ethernet frame transfer, the MAC outputs the transmit status to the application. The detailed description of the Transmit Status is the same as for TDES0.

55.4.1.2 Reception

A receive operation is initiated when the MAC detects an SFD on the RMII or MII. The MAC strips the preamble and SFD before proceeding to process the frame. The header fields are checked for filtering and the FCS (Frame Check Sequence) field used to verify the CRC for the frame. The received frame is stored in a shallow buffer until the address filtering is performed. The frame is dropped in the MAC if it fails the address filtering.

The frames received by the MAC are pushed into the RX FIFO. Once the RX FIFO crosses the configured Receive threshold (RX_THRESH_CTRL), its status is indicated to the DMA, so that the DMA can initiate a pre-configured burst transfers towards the AHB interface.

In the default cut-through mode, when the FIFO receives 64 bytes or a full packet of data, it pops out the data and indicates the availability to the DMA. Once the DMA initiates the transfer to the AHB interface, the data transmission continues from the FIFO until a complete packet has been transferred. Upon the completion of the EOF frame transfer, the status word will be popped out and transmitted to the DMA controller.

Receive Protocol

The receive module strips the preamble and SFD of the received frame. Once the SFD is detected, the MAC begins sending the Ethernet frame data to the RX FIFO, starting from the first byte following the SFD (destination address). If the IEEE 1588 timestamp feature is enabled, the snapshot of system time will be captured whenever an SFD of any frame is detected on MII, and sent to the application unless the MAC filters out and drops the frame.

If the Length/Type field of the receiving frame is less than 0x600 and if the MAC is programmed for the automatic CRC/paddingstripping option, the MAC sends the frame data up to the count specified in the Length/Type field to the RX FIFO, and then starts dropping bytes (including the FCS field). If the Length/Type field is greater than or equal to 0x600, the MAC will send all received Ethernet frame data to the RX FIFO, regardless of the value on the programmed automatic CRC stripping option. By default, the MAC is programmed for the watchdog timer to be enabled, that is frames above 2048 bytes (including DA, SA, LT, data, padding, and FCS) are cut off. This feature can be disabled by programming the Watchdog Disable bit EMACWATCHDOG. However, even if the watchdog timer is disabled, frames longer than 16 KB will still be cut off and the watchdog timeout status will be indicated.

Receive Frame Controller

If the RECEIVE_ALL bit in the MAC Frame Filter Register is reset, the MAC performs frame filtering based on the destination and source addresses. The application still needs to perform another level of filtering if it decides not to receive any bad frames like runt, CRC error frames, etc. On detecting a filter failure, the frame is dropped and not transmitted to the application. When the filter parameters change dynamically, if a DA and SA filter failure occurs, the reset of the frame is dropped and the Receive Status word (with zero frame length, CRC Error, and Runt Error bits set) is updated immediately indicating the filter failure.

Receive Flow Control

The MAC detects the receiving pause frame and pauses the frame transmission for the delay specified within the received pause frame (in full-duplex mode only). The Pause Frame Detection Function can be enabled or disabled with the RFCE bit. Once the receive flow control is enabled, the MAC starts monitoring the received frame destination address for any match with the multicast address of the control frame (0x0180 C200 0001).

If a match is detected (i.e., the destination address of the received frame matches the destination address of the reserved control frame), the MAC decides whether to transmit the received control frame to the application, according to the PCF field.

The MAC also decodes the type, the opcode, and the pause timer field of the receiving control Frame. If the byte count of the status indicates 64 bytes, and if there is no CRC error, the MAC transmitter will pause the transmission of any data frame for the duration of the decoded pause time value multiplied by the slot time (64 byte times for 10/100 Mbit/s mode). Meanwhile, if another pause frame is detected with a zero pause time, the MAC will reset the pause time and give another pause request.

If the received control frame matches neither the type field (0x8808), the opcode (0x00001), nor the byte length (64 bytes), or if there is a CRC error, the MAC will not generate a pause.

In the case of a pause frame with a multicast destination address, the MAC filters the frame according to the address match.

For a pause frame with a unicast destination address, the filtering depends on whether the DA matched the contents of the MAC Address Register 0 and the UPFD (detecting a pause frame even with a unicast destination address) is set. The PCF register controls the filtering for control frames in addition to the address filter.

Receive Operation Multiframe Handling

Since the status is available immediately following the data, the MAC is capable of storing any number of frames into the FIFO, as long as it is not full.

Error Handling

If the RX FIFO is full before it receives the EOF data from the MAC, an overflow will be declared and the whole frame will be dropped. The status bit RDESO[11] will reflect the fact that this frame is a partial frame due to overflow. The RX FIFO can perform the filtering of error frames and runt frames, if this function is enabled via FLUSH_TX_FIFO and FWD_UNDER_GF.

In cut-through mode, if a frame's status and length are available when reading an SOF from the RX FIFO, the whole error frame can be dropped. The DMA can flush the error frame being read from the FIFO by clearing FWD_ERR_FRAME to 0. The DMA then stops transferring data to the application, internally reads out the rest of the frame, and drops it. The MTL will then start the transfer of the next frame, if it is available. If FIFO is available, the transmission of the next frame will be initiated.

Receive Status Word

At the end of the Ethernet frame transfer, the MAC outputs the receive status to the application via EMAC_DMA. The detailed description of the receive status is the same for bit[31:0] in RDESO.

55.4.2 EMAC_MTL (MAC Transaction Layer)

The MAC Transaction Layer provides FIFO memory to buffer and regulate the frames between the application system memory and the MAC. It also enables the data to be transmitted between the application clock domain and the MAC clock domains. The MTL layer has two data paths, namely the Transmit path and the Receive path. The data path for both directions is 32-bit wide and operates with a simple FIFO protocol.

55.4.3 EMAC_DMA

EMAC has its own DMA module, with independent Transmit and Receive engines and a CSR (Control and Status Registers) space. The Transmit engine transfers data from the system memory to the device port (MTL), while the Receive engine transmits data from the device port to the system memory. The controller uses descriptors to efficiently move data from source to destination with minimal Host CPU intervention. The DMA is designed for packet-oriented data transfer such as frames in Ethernet. The controller can be programmed to interrupt the Host CPU for normal situations, such as the completion of frame transmission or reception, or error conditions.

55.4.3.1 Transmit Descriptors

Figure 55.4-1 shows the structure of the transmit descriptor linked list. Table 55.4-1 through Table 55.4-4 show the descriptions of each transmit descriptor.

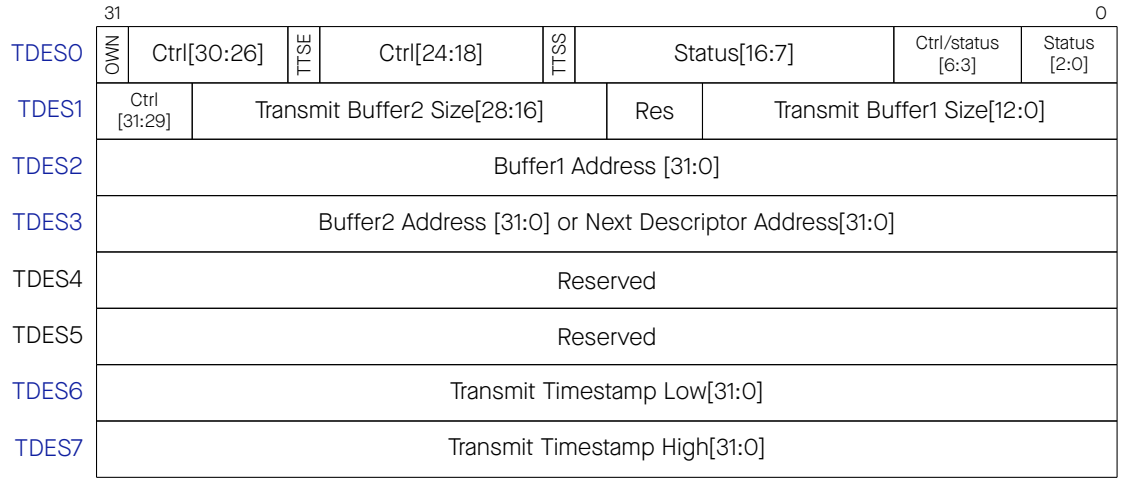


Figure 55.4-1. Transmit Descriptor Linked List

Table 55.4-1. Transmit Descriptor 0 (TDES0)

Bits	Name	Description
[31]	OWN: Own Bit	When set, this bit indicates that the descriptor is owned by the DMA. When reset, it indicates that the descriptor is owned by the Host. The DMA clears this bit, either when it completes the frame transmission or when the buffers allocated to the descriptor are empty. The ownership bit of the First Descriptor of the frame should be set after all subsequent descriptors belonging to the same frame have been set. This avoids a possible race condition between fetching a descriptor and the driver setting an ownership bit.
[30]	IC: Interrupt on Completion	When set, this bit sets the Transmit Interrupt TRANS_INT after the present frame has been transmitted. This bit is valid only when the last segment bit TDES0[29] is set.
[29]	LS: Last Segment	When set, this bit indicates that the buffer contains the last segment of the frame, and the TBS1 or TBS2 field in TDES1 should have a non-zero value.

Bits	Name	Description
[28]	FS: First Segment	When set, this bit indicates that the buffer contains the first segment of a frame.
[27]	DC: Disable CRC	When this bit is set, the MAC does not append a cyclic redundancy check (CRC) to the end of the transmitted frame. This is valid only when the first segment (TDES0[28]) is set.
[26]	DP: Disable Padding	When this bit is set, the MAC does not automatically add padding to a frame shorter than 64 bytes. When this bit is reset, the DMA automatically adds padding and CRC to a frame shorter than 64 bytes, and the CRC field is added despite the state of the DC (TDES0[27]) bit. This is valid only when the first segment (TDES0[28]) is set.
[25]	TTSE: Transmit Time Stamp Enable	When this bit is set, the MAC enables IEEE1588 hardware timestamping for the transmit frame.
[24]	CRCR: CRC Replacement Control	When set, the MAC replaces the last four bytes of the transmitted packet with recalculated CRC bytes. The Host should ensure that the CRC bytes are present in the frame being transmitted from the Transmit Buffer. This bit is valid when the First Segment control bit (TDES0[28]) is set. In addition, CRC replacement is done only when Bit TDES0[27] is set to 1.
[23:22]	CIC: Checksum Insertion Control	These bits control the checksum calculation and insertion. Bit encodings are shown below: • 00: Checksum insertion is disabled. • 01: Only IP header checksum calculation and insertion are enabled. • 10: IP header checksum and payload checksum calculation and insertion are enabled, but pseudo-header checksum is not calculated in hardware. • 11: IP Header checksum and payload checksum calculation and insertion are enabled, and pseudo-header checksum is calculated in hardware. This field is valid when the First Segment control bit (TDES0[28]) is set.
[21]	TER: Transmit End of Ring	When set, this bit indicates that the descriptor list reached its final descriptor. The DMA returns to the base address of the list, creating a Descriptor Ring.
[20]	TCH: Second Address Chained	When set, this bit indicates that the second address in the descriptor is the Next Descriptor address, rather than the second buffer address. When TDES0[20] is set, TBS2 (TDES1[28:16]) is a “don’t care” value. TDES0[21] takes precedence over TDES0[20]. This bit must be set to 1.

Bits	Name	Description
[19:18]	VLIC: VLAN Insertion Control	When set, these bits request the MAC to perform VLAN tagging or untagging before transmitting the frames. If the frame is modified for VLAN tags, the MAC automatically recalculates and replaces the CRC bytes. The following list describes the values of these bits: • 00: Do not add a VLAN tag. • 01: Remove the VLAN tag from the frames before transmission. This option should be used only with the VLAN frames. • 10: Insert a VLAN tag with the tag value programmed in VLAN Tag Inclusion or Replacement Register. • 11: Replace the VLAN tag in frames with the Tag value programmed in VLAN Tag Inclusion or Replacement Register. This option should be used only with the VLAN frames.
[17]	TTSS: Transmit Timestamp Status	This bit is used as a status bit to indicate that a timestamp was captured for the described transmit frame. When this bit is set, TDES6 and TDES7 have a timestamp value captured for the transmit frame. This bit is valid only when the descriptor's LS (TDES0[29]) is set.
[16]	IHE: IP Header Error	When set, this bit indicates that the MAC transmitter detected an error in the IP datagram header. The transmitter checks the header length in the IPv4 packet against the number of header bytes received from the application, and indicates an error status if there is a mismatch. For IPv6 frames, a header error is reported if the main header length is not 40 bytes. Furthermore, the Ethernet Length/Type field value for an IPv4 or IPv6 frame must match the IP header version received with the packet. For IPv4 frames, an error status is also indicated if the Header Length field has a value less than 0x5.
[15]	ES: Error Summary	Indicates the logical OR of the following bits: • TDES0[16]: IP Header Error • TDES0[14]: Jabber Timeout • TDES0[13]: Frame Flush • TDES0[12]: IP Payload Error • TDES0[11]: Loss of Carrier • TDES0[10]: No Carrier • TDES0[9]: Late Collision • TDES0[8]: Excessive Collision • TDES0[2]: Excessive Deferral • TDES0[1]: Underflow Error
[14]	JT: Jabber Timeout	When set, this bit indicates the MAC transmitter has experienced a jabber timeout. This bit is only set when the EMACJABBER bit (disable jabber) of EMACCONFIG_REG is not set.
[13]	FF: Frame Flushed	When set, this bit indicates that the DMA or MTL flushed the frame because of a software Flush command given by the CPU.

Bits	Name	Description
[12]	IPE: IP Payload Error	When set, this bit indicates that the MAC transmitter detected an error in the TCP, UDP, or ICMP IP datagram payload. The transmitter checks the payload length received in the IPv4 or IPv6 header against the actual number of TCP, UDP, or ICMP packet bytes received from the application, and issues an error status in case of a mismatch.
[11]	LOC: Loss of Carrier	When set, this bit indicates that a Loss of Carrier occurred during frame transmission (that is, the MII_CRS signal was inactive for one or more transmit clock periods during frame transmission). This is valid only for the frames transmitted without collision and when the MAC operates in the half-duplex mode.
[10]	NC: No Carrier	When set, this bit indicates that the carrier sense signal from the PHY was not asserted during transmission.
[9]	LC: Late Collision	When set, this bit indicates that frame transmission was aborted because of a collision occurring after the collision window (in MII mode, 64 byte times including Preamble, and 512 byte times including Preamble and Carrier Extension). This bit is not valid if the Underflow Error bit is set.
[8]	EC: Excessive Collision	When set, this bit indicates that the transmission was aborted after 16 successive collisions while attempting to transmit the current frame. If the EMACRETRY bit of EMACCONFIG_REG is set, this bit is set after the first collision, and the transmission of the frame is aborted.
[7]	VF: VLAN Frame	When set, this bit indicates that the transmitted frame was a VLAN-type frame.
[6:3]	Ctrl/status	These status bits indicate the number of collisions occurring before the frame was transmitted. This count is not valid when the Excessive Collisions bit (TDES0[8]) is set. The CPU updates this status field only in the half-duplex mode.
[2]	ED: Excessive Deferral	When set, this bit indicates that the transmission has ended because of excessive deferral of over 24,288 bit times (if Jumbo Frame is enabled) if the EMACDEFERRAL bit (deferral check) of EMACCONFIG_REG is set high.
[1]	UF: Underflow Error	When set, this bit indicates that the MAC aborted the frame because the data arrived late from the Host memory. Underflow Error indicates that the DMA encountered an empty transmit buffer while transmitting the frame. The transmission process enters the Suspended state and sets both Transmit Underflow Register (Status Register TRANS_UNDFLOW) and Transmit Interrupt Register (Status Register TRANS_INT).
[0]	DB: Deferred Bit	When set, this bit indicates that the MAC defers before transmission because of the presence of a carrier. This bit is valid only in the half-duplex mode.

Table 55.4-2. Transmit Descriptor 1 (TDES1)

Bits	Name	Description
[31:29]	SAIC: SA Insertion Control	These bits request the MAC to add or replace the Source Address field in the Ethernet frame with the value given in EMACADDR0HIGH_REG, EMACADDR0LOW_REG, EMACADDR1HIGH_REG, and EMACADDR1LOW_REG. If the Source Address field is modified in a frame, the MAC automatically recalculates and replaces the CRC bytes. The Bit[31] specifies the MAC Address Register value (1 or 0) that is used for Source Address insertion or replacement. The following list describes the values of Bits[30:29]: • 00: Do not include the source address. • 01: Include or insert the source address. For reliable transmission, the application must provide frames without source addresses. • 10: Replace the source address. For reliable transmission, the application must provide frames with source addresses. • 11: Reserved. These bits are valid when the First Segment control bit (TDES0[28]) is set.
[28:16]	TBS2: Transmit Buffer2 Size	These bits specify the transmit Buffer 2 size in bytes. This field is not valid if TDES0[20] is set.
[15:13]	Reserved	Reserved
[12:0]	TBS1: Transmit Buffer 1 Size	These bits indicate the data buffer byte size in bytes. If this field is 0, the DMA ignores this buffer and uses Buffer 2 or the next descriptor, depending on the value of TCH (TDES0[20]).

Table 55.4-3. Transmit Descriptor 2 (TDES2)

Bits	Name	Description
[31:0]	Buffer 1 Address Pointer	These bits indicate the physical address of Buffer 1.

Table 55.4-4. Transmit Descriptor 3 (TDES3)

Bits	Name	Description
[31:0]	Buffer2 Address Pointer (Next Descriptor Address)	These bits specify the physical address of Buffer 2 when a descriptor ring structure is used. If the Second Address Chained (TDES0[20]) bit is set, this address specifies the physical memory where the Next Descriptor is present.

Table 55.4-5. Transmit Descriptor 6 (TDES6)

Bits	Name	Description
[31:0]	TTSL: Transmit Frame Time Stamp Low	This field is updated by DMA with the least significant 32 bits of the timestamp captured for the corresponding transmit frame. This field has the timestamp only if the Last Segment (TDES0[29]) bit in the descriptor is set, and the Timestamp Status (TDES0[17]) bit is set.

Table 55.4-6. Transmit Descriptor 7 (TDES7)

Bits	Name	Description
[31:0]	TTSH: Transmit Frame Time Stamp High	This field is updated by DMA with the most significant 32 bits of the timestamp captured for the corresponding receive frame. This field has the timestamp only if the Last Segment (TDES0[29]) bit in the descriptor is set, and the Timestamp Status (TDES0[17]) bit is set.

55.4.3.2 Receive Descriptors

Figure 55.4-2 shows the structure of the receive descriptor linked list. Table 55.4-7 through Table 55.4-11 show the descriptions of each receive descriptor.

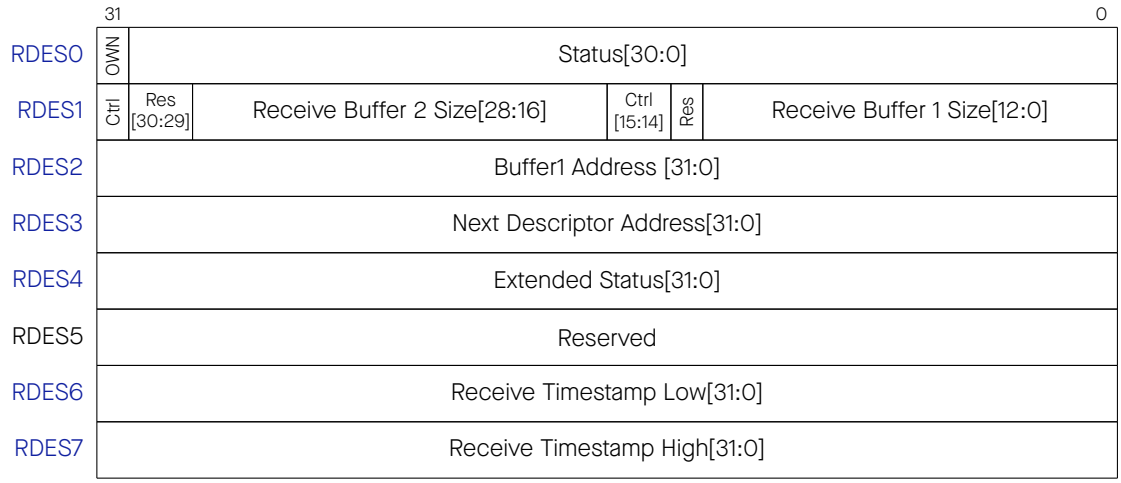


Figure 55.4-2. Receive Descriptor Linked List

Table 55.4-7. Receive Descriptor 0 (RDES0)

Bits	Name	Description
[31]	OWN: Own Bit	When set, this bit indicates that the descriptor is owned by the DMA. When reset, it indicates that the descriptor is owned by the Host. The DMA clears this bit either when it completes the frame reception or when the buffers that are associated with this descriptor are full.

Bits	Name	Description
[30]	AFM: Destination Address Filter Fail	When set, this bit indicates a frame that failed in the DA Filter in the MAC.
[29:16]	FL: Frame Length	These bits indicate the byte length of the received frame that was transmitted to Host memory. This field is valid when Last Descriptor (RDESO[8]) is set and either the Descriptor Error (RDESO[14]) or Overflow Error bit RDESO[11] are reset. The frame length also includes the two bytes appended to the Ethernet frame when IP checksum calculation (Type 1) is enabled and the received frame is not a MAC control frame.
[15]	ES: Error Summary	Indicates the logical OR of the following bits: • RDESO[1]: CRC Error • RDESO[3]: Receive Error • RDESO[4]: Watchdog Timeout • RDESO[6]: Late Collision • RDESO[7]: Giant Frame • RDES4[4:3]: IP Header or Payload Error • RDESO[11]: Overflow Error • RDESO[14]: Descriptor Error This field is valid only when RDESO[8] is set.
[14]	DE: Descriptor Error	When set, this bit indicates a frame truncation caused by a frame that does not fit within the current descriptor buffers, and that the DMA does not own the Next Descriptor. The frame is truncated. This field is valid only when This field is valid only when RDESO[8] is set.
[13]	SAF: Source Address Filter Fail	When set, this bit indicates that the SA field of frame failed the SA Filter in the MAC.
[12]	LE: Length Error	When set, this bit indicates that the actual length of the frame received and that the Length/Type field does not match. This bit is valid only when the Frame Type (RDESO[5]) bit is reset.
[11]	OE: Overflow Error	When set, this bit indicates that the received frame was damaged because of buffer overflow in MTL.
[10]	VLAN: VLAN Tag	When set, this bit indicates that the frame pointed to by this descriptor is a VLAN frame tagged by the MAC. The VLAN tagging depends on checking the VLAN fields of the received frame based on the VLAN Tag Register) configuration.
[9]	FS: First Descriptor	When set, this bit indicates that this descriptor contains the first buffer of the frame. If the size of the first buffer is 0, the second buffer contains the beginning of the frame. If the size of the second buffer is also 0, the next Descriptor contains the beginning of the frame.
[8]	LS: Last Descriptor	When set, this bit indicates that the buffers pointed to by this descriptor are the last buffers of the frame.

Bits	Name	Description
[7]	IP Checksum Error (Type1), or Giant Frame	When the Advanced Timestamp feature is present, and when this bit set, it indicates that a snapshot of the Timestamp is written in descriptor word 6 (RDES6) and 7 (RDES7). This is valid only when the Last Descriptor bit (RDES0[8]) is set. When IP Checksum Engine (Type 1) is selected, this bit, if set, indicates one of the following: • The 16-bit IPv4 header checksum calculated by the core did not match the received checksum bytes. • The header checksum checking is bypassed for non-IPv4 frames. Otherwise, this bit, when set, indicates the Giant Frame Status. When normal frame processing is enabled, Giant frames are larger than 1,518 bytes (or 1,522 bytes for VLAN or 2,000 bytes when ASS2KP is set). When Jumbo Frame processing is enabled, Giant frames are larger than 9,018 bytes (or 9,022 bytes for VLAN).
[6]	LC: Late Collision	When set, this bit indicates that a late collision has occurred while receiving the frame in the half-duplex mode.
[5]	FT: Frame Type	When set, this bit indicates that the Receive Frame is an Ethernet-type frame (the LT field is greater than, or equal to, 1,536 bytes). When reset, it indicates that the received frame is an IEEE 802.3 frame. This bit is not valid for Runt frames which are less than 14 bytes.
[4]	RWT: Receive Watchdog Timeout	When set, this bit indicates that the Receive Watchdog Timer has expired while receiving the current frame and the current frame is truncated after the Watchdog Timeout.
[3]	RE: Receive Error	When set, this bit indicates that the MII_RXER signal is asserted while MII_RXDV is asserted during frame reception.
[2]	DE: Dribble Bit Error	When set, this bit indicates that the received frame has a non-integer multiple of bytes (odd nibbles). This bit is valid only in the MII Mode.
[1]	CE: CRC Error	When set, this bit indicates that a Cyclic Redundancy Check (CRC) Error occurred on the received frame. This field is valid only when the Last Descriptor (RDES0[8]) is set.

Bits	Name	Description
[0]	Extended Status Available/ RX MAC Address	When the IP Checksum Offload (Type 2) is present, this bit, when set, indicates that the extended status is available in descriptor word 4 (RDES4). This is valid only when the Last Descriptor bit (RDES0[8]) is set. This bit is invalid when Bit 30 is set. When IP Checksum Offload (Type 2) is present, this bit is set even when the IP Checksum Offload engine bypasses the processing of the received frame. The bypassing may be because of a non-IP frame or an IP frame with a non-TCP/UDP/ICMP payload. When the IPC Full Offload is not selected, this bit indicates an RX MAC Address status. When set, this bit indicates that the RX MAC Address registers value (1 to 15) matched the frame's DA field. When reset, this bit indicates that the RX MAC Address Register 0 value matched the DA field.

Table 55.4-8. Receive Descriptor 1 (RDES1)

Bits	Name	Description
[31]	Ctrl	When set, this bit prevents setting the Status Register's RI bit (RECV_INT) for the received frame that ends in the buffer indicated by this descriptor. This, in turn, disables the assertion of the interrupt to Host because of the RI for that frame.
[30:29]	Reserved	Reserved.
[28:16]	RBS2: Receive Buffer 2 Size	These bits indicate the size of Buffer 2 in bytes. The buffer size must be a multiple of 4, even if the value of RDES3 (Buffer 2 address pointer) is not aligned to bus width. In the case where the buffer size is not a multiple of 4, the resulting behavior is undefined. This field is not valid if RDES1[14] is set.
[15]	RER: Receive End of Ring	When set, this bit indicates that the descriptor list reached its final descriptor. The DMA returns to the base address of the list, creating a Descriptor Ring.
[14]	RCH: Second Address Chained	When set, this bit indicates that the second address in the descriptor is the Next Descriptor address rather than the second buffer address. When this bit is set, RBS2 (RDES1[28:16]) is a "don't care" value. RDES1[15] takes precedence over RDES1[14].
[13]	Reserved	Reserved.
[12:0]	RBS1: Receive Buffer 1 Size	These bits indicate the size of Buffer 1 in bytes. The buffer size must be a multiple of 4, even if the value of RDES2 (buffer1 address pointer) is not aligned to bus width. When the buffer size is not a multiple of 4, the resulting behavior is undefined. If this field is 0, the DMA ignores this buffer and uses Buffer 2 or the next descriptor depending on the value of RCH (Bit[14]).

Table 55.4-9. Receive Descriptor 2 (RDES2)

Bits	Name	Description
[31:0]	Buffer 1 Address Pointer	These bits indicate the physical address of Buffer 1.

Table 55.4-10. Receive Descriptor 3 (RDES3)

Bits	Name	Description
[31:0]	Buffer 2 Address Pointer (Next Descriptor Address)	These bits indicate the physical address of Buffer 2 when descriptor chaining is used. If the Second Address Chained (RDES1[14]) bit is set, then this address contains the pointer to the physical memory where the Next Descriptor is present.

Table 55.4-11. Receive Descriptor 4 (RDES4)

Bits	Name	Description
[31:28]	Reserved	Reserved.
[27:26]	Reserved	Reserved.
[25]	Reserved	Reserved.
[24]	Reserved	Reserved.
[23:21]	Reserved	Reserved.
[20:18]	Reserved	Reserved.
[17]	Reserved	Reserved.
[16]	Reserved	Reserved.
[15]	Reserved	Reserved.
[14]	Timestamp Dropped	When set, this bit indicates that the timestamp was captured for the receive frame but got dropped in the MTL RX FIFO because of overflow.
[13]	PTP Version	When set, this bit indicates that the received PTP message is having the IEEE 1588 version 2 format. When reset, it has the version 1 format.
[12]	PTP Frame Type	When set, this bit indicates that the PTP message over Ethernet is sent directly over Ethernet. When this bit is not set and the MT (message type) is not 0, it indicates that the PTP message over UDP-IPv4 or UDP-IPv6 is received.

Bits	Name	Description
[11:8]	Message Type	These bits are encoded to give the type of the message received. • 0000: Reserved • 0001: SYNC (all clock types) • 0010: Follow_Up (all clock types) • 0011: Delay_Req (all clock types) • 0100: Delay_Resp (all clock types) • 0101: Pdelay_Req (in peer-to-peer transparent clock) • 0110: Pdelay_Resp (in peer-to-peer transparent clock) • 0111: Pdelay_Resp_Follow_Up (in peer-to-peer transparent clock) • 1000: Announce • 1001: Management • 1010: Signaling • 1011-1110: Reserved • 1111: PTP packet with Reserved message type
[7]	IPv6 Packet Received	When set, this bit indicates that the received packet is an IPv6 packet. This bit is updated only when Bit[10] (IPC) of MAC Configuration Register (EMACRXIPCOFFLOAD) is set.
[6]	IPv4 Packet Received	When set, this bit indicates that the received packet is an IPv4 packet. This bit is updated only when Bit[10] (IPC) of MAC Configuration Register (EMACRXIPCOFFLOAD) is set.
[5]	IP Checksum Bypassed	When set, this bit indicates that the checksum offload engine is bypassed.
[4]	IP Payload Error	When set, this bit indicates that the 16-bit IP payload checksum (that is, the TCP, UDP, or ICMP checksum) that the MAC core calculated does not match the corresponding checksum field in the received segment. It is also set when the TCP, UDP, or ICMP segment length does not match the payload length value in the IP Header field. This bit is valid when either Bit[7] or Bit[6] is set.
[3]	IP Header Error	When set, this bit indicates that either the 16-bit IPv4 header checksum calculated by the MAC core does not match the received checksum bytes, or the IP datagram version is not consistent with the Ethernet Type value. This bit is valid when either Bit[7] or Bit[6] is set.

Bits	Name	Description
[2:0]	IP Payload Type	These bits indicate the type of payload encapsulated in the IP datagram processed by the Receive Checksum Offload Engine (COE). The COE also sets these bits to 00 if it does not process the IP datagram's payload due to an IP header error or fragmented IP. • 000: Unknown or did not process IP payload • 001: UDP • 010: TCP • 011: ICMP • 1xx: Reserved This bit is valid when either Bit[7] or Bit[6] is set.

Table 55.4-12. Receive Descriptor 6 (RDES6)

Bits	Name	Description
[31:0]	RTSH: Receive Frame Time Stamp Low	This field is updated by DMA with the least significant 32 bits of the timestamp captured for the corresponding receive frame. This field is updated by DMA only for the last descriptor of the receive frame which is indicated by the Last Descriptor status bit (RDES0[8]).

Table 55.4-13. Receive Descriptor 7 (RDES7)

Bits	Name	Description
[31:0]	RTSH: Receive Frame Timestamp High	This field is updated by DMA with the most significant 32 bits of the timestamp captured for the corresponding receive frame. This field is updated by DMA only for the last descriptor of the receive frame which is indicated by the Last Descriptor status bit (RDES0[8]).

55.4.4 PHY Interface

The MAC and the external PHY communicate through two interfaces:

- Media Independent Interface (MII)
- Reduced Media Independent Interface (RMII)

55.4.4.1 Media Independent Interface: MII

The Media Independent Interface (MII) defines the interconnection between MAC sublayers and PHYs at 10 Mbit/s and 100 Mbit/s.

MII Signals to PHY

The MII interface signals are shown in Figure 55.4-3.

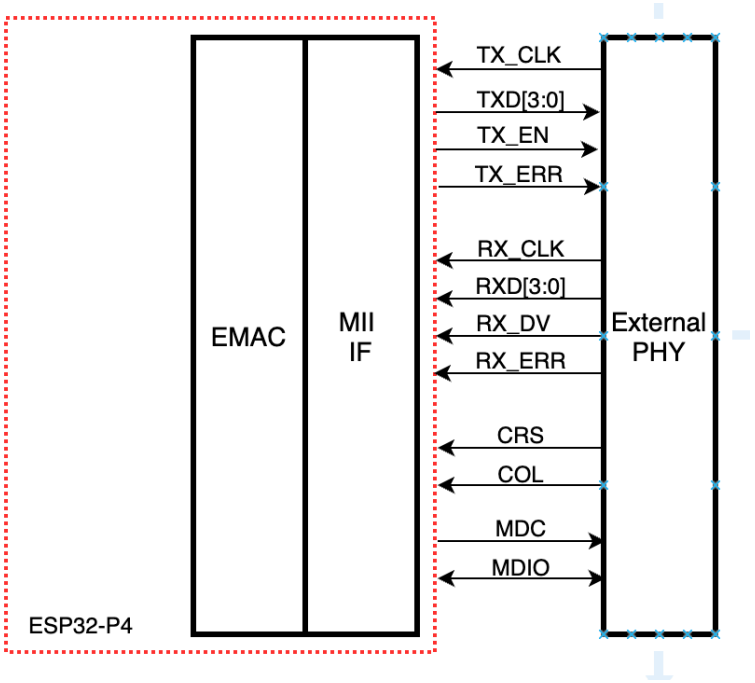


Figure 55.4-3. MII Interface

The MII interface signals are described below:

- MII_TX_CLK: TX clock signal. This signal provides the reference timing for TX data transmission. It has two frequencies: 2.5 MHz at 10 Mbit/s, and 25 MHz at 100 Mbit/s.
- MII_TXD[3:0]: A bundle of four transmit data signals driven by the MAC sublayer, and qualified (valid data) on the assertion of the MII_TX_EN signal. MII_TXD[0] is the lowest significant bit and MII_TXD[3] is the highest significant bit. When the MII_TX_EN signal is deasserted, the transmit data have no effect on the PHY.
- MII_TX_EN: Transmission enable signal. This signal indicates that the MAC is presenting nibbles (4 bits) on the MII for transmission. It must be asserted synchronously (MII_TX_CLK) with the first nibble of the preamble and must remain asserted while all nibbles to be transmitted are presented to the MII.
- MII_TX_ERR: Transmit error signal. The signal must be asserted for one or more clock periods (MII_TX_CLK) to indicate to the PHY layer that an error was detected somewhere in the frame.
- MII_RX_CLK: RX clock signal. This signal provides the reference timing for RX data transmission. It has two frequencies: 2.5 MHz at 10 Mbit/s, and 25 MHz at 100 Mbit/s.
- MII_RXD[3:0]: A bundle of four receive data signals driven by the MAC sublayer, and qualified (valid data) on the assertion of the MII_RX_DV signal. MII_RXD[0] is the lowest significant bit and MII_RXD[3] is the highest significant bit. When MII_RX_DV is deasserted and MII_RX_ERR is asserted, a specific MII_RXD[3:0] value is used to transfer specific information from the PHY.
- MII_RX_DV: Receive data valid signal. This signal indicates that the PHY is presenting recovered and decoded nibbles on the MII for reception. It must be asserted synchronously (MII_RX_CLK) with the first recovered nibble of the frame and must remain asserted through the final recovered nibble. It must be deasserted prior to the first clock cycle that follows the final nibble. In order to receive the frame correctly, the MII_RX_DV signal must encompass the frame, starting no later than the SFD field.

- MII_CRS: Carrier sense signal. When the transmit or receive medium is non-idle, the signal is asserted by the PHY. When the transmit or receive medium is idle, the signal is deasserted by the PHY. The PHY must ensure that the MII_CRS signal remains asserted throughout the duration of a collision condition. This signal does not need to be synchronized with the TX and RX clocks. In full duplex mode the state of this signal is don't care for the MAC sublayer.
- MII_COL: Collision detection signal. This signal must be asserted by the PHY upon detection of a collision on the medium and must remain asserted while the collision condition persists. This signal is not required to transition synchronously with respect to the TX and RX clocks. In full duplex mode, the state of this signal is don't care for the MAC sublayer.
- MII_RX_ERR: Receive error signal. The signal must be asserted for one or more clock periods (MII_RX_CLK) to indicate to the MAC sublayer that an error was detected somewhere in the frame.
- MDIO and MDC: Management data input/output and management data clock signal. The two signals constitute a serial bus defined for the Ethernet family of IEEE 802.3 standards, used to transfer control and data information to the PHY, see section Station Management Agent (SMA) Interface.

MII Clock

In MII mode, there are clock signals in two directions between MII and the PHY, namely clk_tx and clk_rx. clk_tx is used to synchronize the TX data, and clk_rx is used to synchronize the RX data. Both clocks are provided by the PHY.

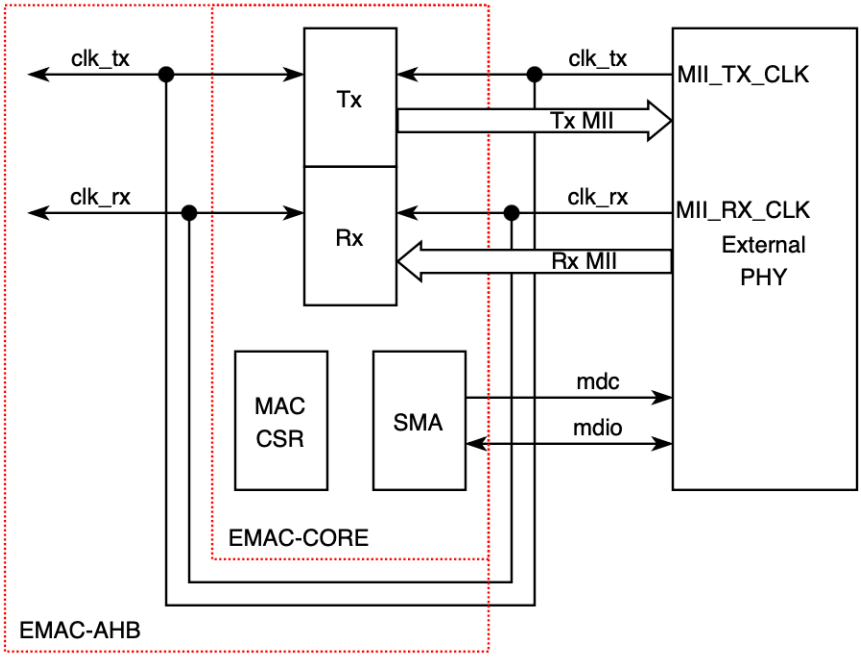


Figure 55.4-4. MII Clock

55.4.4.2 Reduced Media Independent Interface: RMII

RMII interface signals are shown in Figure 55.4-5.

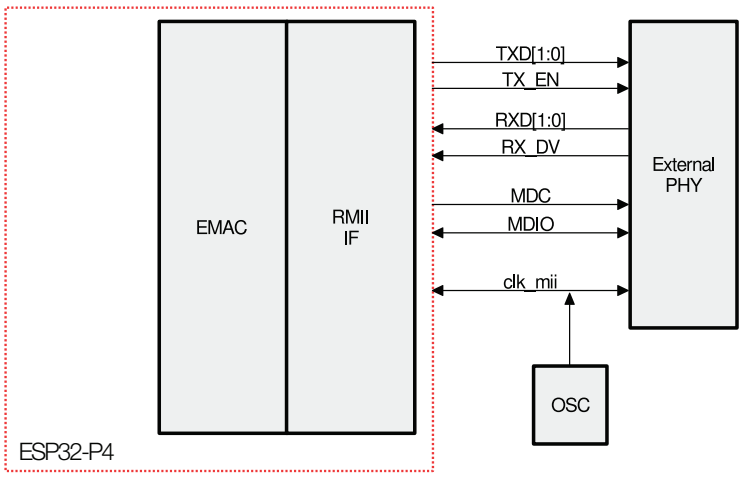


Figure 55.4-5. RMII Interface

Note:

Figure 55.4-3 and Figure 55.4-5 uniformly use the signal name RX_DV to represent the receive-valid signal. In the MII interface, this corresponds to RX_DV, while in the RMII interface it corresponds to CRS_DV. The actual signal semantics depend on the selected Ethernet interface mode.

RMII Signals

The Reduced Media Independent Interface (RMII) specification reduces the number of pins between the microcontroller's Ethernet MAC and the external PHY at 10 Mbit/s or 100 Mbit/s. According to the IEEE 802.3u standard, an MII contains 16 pins for data and control. The RMII specification reduces the pin count to 7.

RMII has the following features:

- Supports an operating rate of 10 Mbit/s or 100 Mbit/s
- The clock reference must be 50 MHz.
- The same clock reference must be sourced externally both to the MAC and the external Ethernet PHY.
- Provides independent 2-bit wide TX and RX data paths.

RMII Clock

The 50 MHz RMII reference clock can be:

- Generated internally by ESP32-P4 and looped back via GPIOs. For GPIO configurations, please refer to Chapter 10 *GPIO Matrix and IO MUX*.
- Provided by the external crystal and obtained at GPIOs.

The RMII clock is shown in Figure 55.4-6.

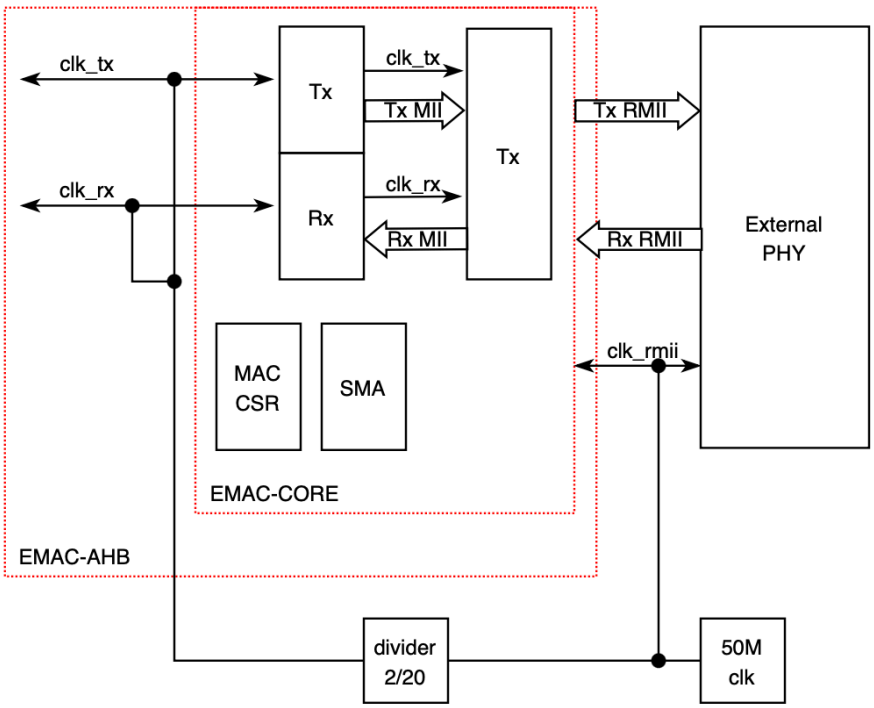


Figure 55.4-6. RMII Clock

55.4.4.3 Station Management Agent (SMA) Interface

As Figure 55.4-3 and Figure 55.4-5 show, the MAC uses MDC and MDIO signals to transfer control and data information to the PHY. The MDC clock is generated from the application clock through a clock divider and operates at a maximum frequency of 2.5 MHz. The MDIO data signal transfers to and from PHY synchronously with the MDC clock signal. Meanwhile, the PHY transmits register data.

- Read a PHY register:
  - Read MIIBUSY until the MII is in an idle state.
  - Configure MIIDEV to specify the PHY device to access.
  - Configure MIIREG to specify the address of the PHY register to access.
  - Configure MIICSRCLK to set the APB clock frequency.
  - Set MIIWRITE to 0 for read access.
  - Read MIIBUSY until the MII is in an idle state.
  - Read EMACMIIDATA_REG to retrieve the value of the PHY register.
- Write a PHY register:
  - Read MIIBUSY until the MII is in an idle state.
  - Configure EMACMIIDATA_REG to set the value to be written to the PHY register.
  - Configure MIIDEV to specify the PHY device to access.
  - Configure MIIREG to specify the address of the PHY register to access.
  - Configure MIICSRCLK to set the APB clock frequency.

- Set MIIWRITE to 1 for write access.
- Read MIIBUSY until the MII is in an idle state, which indicates the completion of the write operation.

55.4.5 MAC Address Filtering

Address filtering checks the destination and source addresses on all received frames and reports the address filtering status accordingly. Address checking is based on different parameters (Frame filter register) chosen by the application. The filtered frame can also be identified: multicast or broadcast frame. Address filtering uses the station's physical (MAC) address for address checking.

55.4.5.1 Unicast Destination Address Filtering

The MAC supports up to 9 MAC addresses for unicast perfect filtering. MAC compares all 48 bits of the received unicast address with the programmed MAC address for any match. By default, EMACADDR0 is always enabled, and the other addresses (EMACADDR0 ~ EMACADDR8) are selected with an individual enable bit. Each byte of these other addresses (EMACADDR0 ~ EMACADDR8) can be masked during comparison with the corresponding received DA byte by setting the corresponding Mask Byte Control bit in the register. This helps group address filtering for the DA.

55.4.5.2 Multicast Destination Address Filtering

The MAC can be programmed to pass all multicast frames by setting the Pass All Multicast (PAM) bit in the Frame Filter Register to 1. If the PAM bit is reset, the MAC the filtering for multicast addresses.

In perfect filtering mode, the multicast address is compared with the programmed MAC Destination Address Registers (EMACADDR0 ~ EMACADDR8). Group address filtering is also supported.

55.4.5.3 Broadcast Address Filter

The MAC does not filter any broadcast frames in the default mode. However, if the MAC is programmed to reject all broadcast frames by setting the Disable Broadcast Frames (DBF) bit in the Frame Filter Register to 1, all broadcast frames will be dropped.

55.4.5.4 Unicast Source Address Filter

The MAC can also perform perfect filtering based on the source address field of the received frames. By default, the Address Filtering Module (AFM) compares the Source Address (SA) field with the values programmed in the SA register. The MAC address registers [1:3] can be configured to contain SA instead of DA for comparison, by setting bit[30] in the corresponding register. Group filtering with SA is also supported. The frames that fail the SA filter are dropped by the MAC if the Source Address Filter Enable (SAFE) bit in the Frame filter register is set. Otherwise, the result of the SA filter is given as a status bit in the Receive Status word (see Table 55.4-7).

When the SAFE bit is set to 1, the result of the SA filtering and DA filtering is AND'ed to decide whether the frame needs to be forwarded. This means that either of the filter failure results will drop the frame. Both filters have to pass the frame for the frame to be forwarded to the application.

55.4.5.5 Inverse Filtering Operation

For both destination and source address filtering, there is an option to invert the filter-match result at the final output. These are controlled by the DAIF and SAIF bits in the Frame filter register, respectively. The DAIF bit is applicable for both Unicast and Multicast DA frames. When the DAIF bit is set, the result of the unicast/multicast destination address filter is inverted. Similarly, when the SAIF bit is set, the result of the unicast SA filter is inverted.

The following two tables summarize the destination address and source address filtering based on the type of the frames received.

Table 55.4-14. Destination Address Filtering

Frame Type	PM	PF	DAIF	PAM	DB	DA Filter Operation
Broadcast	1	X	X	X	X	Pass
	0	X	X	X	0	Pass
	0	X	X	X	1	Fail
Unicast	1	X	X	X	X	Pass all frames
	0	X	0	X	X	Pass on perfect/group filter match
	0	X	1	X	X	Fail on perfect/group filter match
	0	1	0	X	X	Pass on perfect/group filter match
	0	1	1	X	X	Fail on perfect/group filter match
Multicast	1	X	X	X	X	Pass all frames
	X	X	X	1	X	Pass all frames
	0	X	0	0	X	Pass on perfect/group filter match and drop pause control frames if PCF = 0x.
	0	1	0	0	X	Pass on perfect/group filter match and drop pause control frames if PCF = 0x.
	0	X	1	0	X	Fail on perfect/group filter match and drop pause control frames if PCF = 0x.
	0	1	1	0	X	Fail on perfect/group filter match and drop pause control frames if PCF = 0x.

The filtering parameters in the MAC Frame Filter Register described in Table 55.4-14 are as follows.

Parameter name		Parameter setting	
PM:	Pass All Multicast	1:	Set
PF:	Perfect Filter	0:	Cleared
DAIF:	Destination Address Inverse Filtering	X:	Don't care
PAM:	Pass All Multicast		
DB:	Disable Broadcast Frames		

Table 55.4-15. Source Address Filtering

Frame Type	PM	SAIF	SAF	Source Address Filter Operation
	1	X	X	Pass all frames
	0	0	0	Pass on perfect/group filter match but do not drop frames that fail
	0	0	0	Fail on perfect/group filter match but do not drop frames that fail
	0	0	0	Pass on perfect/group filter match and drop frames that fail
Unicast	0	0	0	Fail on perfect/group filter match and drop frames that fail

The filtering parameters in the MAC Frame Filter Register described in Table 55.4-15 are as follows.

Parameter name		Parameter setting	
PM:	Pass All Multicast	1:	Set
SAF:	Source Address Filtering	0:	Cleared
SAIF:	Source Address Inverse Filtering	X:	Don't care

55.4.6 Energy Efficient Ethernet (EEE)

Energy Efficient Ethernet (EEE) is an optional operating mode that enables the MAC sublayer along with a series of physical layers to operate in the Low Power Idle (LPI) mode. The EEE mode supports 100 Mbps MAC operations compliant with the IEEE 802.3az-2010 standard. The EEE is supported only in the full-duplex mode when MAC uses MII.

The LPI mode allows power saving by switching off parts of the communication device functionality when there is no data to be transmitted and received. The system on both sides of the link can disable some functionalities and save power during the period of low-link utilization. The MAC controls whether the system should enter or exit the LPI mode and communicate this to the PHY.

55.4.6.1 Transmission

To enter the LPI mode, the software must set the LPIEN bit to indicate to the MAC to stop transmission and initiate the LPI protocol. The MAC completes the transmission in progress, generates its transmission status, and then starts transmitting the LPI pattern instead of the IDLE pattern during the Interframe gap (IFG). The MAC then updates TLPIEN and generates an interrupt.

To exit the LPI mode, the software must clear the LPIEN bit to indicate to the MAC. The MAC stops transmitting the LPI pattern, and starts transmitting the IDLE pattern. When the LPI_TW_TIMER expires, the MAC updates TLPIEX and generates an interrupt.

55.4.6.2 Reception

When the PHY receives the signals from the link partner to enter into the LPI state, the PHY starts transmitting the LPI pattern. The MAC updates RLPIEN and generates an interrupt.

When the PHY receives signals from the link partner to exit the LPI state, the PHY stops transmitting the LPI pattern. The MAC updates RLPIEX and generates an interrupt.

55.4.7 Source Address, VLAN, and CRC Control

55.4.7.1 Source Address Control

The software can enable source address insertion or replacement for all or some frames:

- Enable source address insertion or replacement for all frames by configuring SAIRC.
- Enable source address insertion or replacement for the current frame by configuring TDES1[31:29] of the frame. When TDES1[31] is set, MAC_ADDRESS1_HI and EMACADDR1LOW_REG are used for insertion and replacement. When TDES1[31] is cleared, MAC_ADDRESSO_HI and EMACADDROLOW_REG are used for insertion and replacement. When ADDRESS_ENABLE1 is not set, MAC_ADDRESSO_HI and EMACADDROLOW_REG are used for insertion and replacement irrespective of TDES1[31].

55.4.7.2 VLAN Control

The software can enable VLAN insertion, replacement, or deletion for all or some frames:

- Enable VLAN insertion, replacement, or deletion for all frames by configuring VLC.
- Enable VLAN insertion, replacement, or deletion for the current frame by configuring TDESO[19:18] of the frame.

55.4.7.3 CRC Control

CRC replacement is a feature that the software can use to indicate the MAC to replace the FCS field of the frame being transmitted with the calculated CRC. This feature can be enabled for the current frame by configuring TDESO[24].

CRC replacement is valid only when MAC does not append CRC (TDESO[27] is set), i.e., when FCS is provided by software in frames transmitted. If source address or VLAN control is enabled, the MAC will replace or append FCS fields with CRC when TDESO[27] is set or cleared respectively.

55.4.8 Time Stamp

The IEEE 1588-2008 standard defines the Precision Time Protocol (PTP) that allows precise clock synchronization in measurement and control systems implemented with technologies such as local computing, network communication and distributed objects. it supports system-wide synchronization accuracy in the sub-microsecond range with a minimum network and local clock computing resources.

55.4.8.1 Transmit Path

The MAC captures a timestamp when the start-of-frame data (SFD) is sent on the MII interface. The frames for which timestamps are captured can be controlled on a per-frame basis. In other words, each transmit frame can be marked to indicate whether a timestamp should be captured for that frame. The MAC does not process the transmitted to identity PTP frames. Rather, it captures the timestamp or not according to the configurations of TDESO[25] by software. The MAC returns the timestamp to the software inside the corresponding transmit descriptor, thus connecting the timestamp automatically to the specific PTP frame. The 64-bit timestamp information is written to the TDES6 and TDES7 fields.

55.4.8.2 Receive Path

The MAC processes the received frames to identify valid PTP frames. The snapshot of the time to be sent to the application can be controlled by software via EMACTSTPCTRL_REG.

The MAC returns the timestamp to the software inside the corresponding receive descriptor. The extended status (including the timestamp status and IPC status) is written to RDES4, and the timestamp snapshot is written to RDES6 and RDES7.

55.4.9 Remote Wakeup

The MAC supports remote wakeup. When the receiver receives a remote wakeup frame or a magic packet, an interrupt will be triggered, and the PMU will be notified to wake up the SoC system.

The software can control whether to use remote wakeup frames to trigger interrupts by configuring RWKPKTEN , whether to use magic packets to trigger interrupts by configuring MGKPKTEN, and whether the MAC drops all received frames in power-down mode by configuring PWRDWN.

55.4.10 Good Transmitted and Received Frames

A frame that has been successfully transmitted is considered a "good frame". In other words, a transmitted frame is considered to be "good", if the frame transmission is not aborted due to the following errors:

- Jabber timeout (TDES0[14])
- No carrier or loss of carrier (TDES0[11:10])
- Late collision (TDES0[9])
- Frame underflow (TDES0[1])
- Excessive deferral (TDES0[2])
- Excessive collision (TDES0[8])

A received frame is considered to be "good" if the following errors do not occur:

- CRC error (RDES0[1])
- Runt frames (frames shorter than 64 bytes)
- Alignment error (in 10/100 Mbps modes only) (RDES0[2])
- Length error (non-type frames only) (RDES0[12])
- Frame size over the maximum size (for non-type frames over the maximum frame size only)) (RDES0[7])
- MII_RXER input error (RDES0[3])

The maximum frame size depends on the frame type:

- The maximum size of untagged frames = 1518 bytes
- The maximum size of VLAN frames = 1522 bytes

55.5 Programming Procedures

55.5.1 MAC System Layer Configuration

1. Configure the IO interfaces. For detailed configurations about the MII and the RMII interface, see Chapter 10 *GPIO Matrix and IO MUX*.

2. Configure the EMAC system clock by setting HP_SYS_CLKRST_EMAC_SYS_CLK_EN to 1.
3. Configure the PHY interface clock:
   - RMII interface: select the RMII interface by configuring HP_SYS_PHY_INTF_SEL to 4.
     - Select RMII reference clock source:
       * Reference clock sourced from the external crystal:
         · Configure HP_SYS_CLKRST_PAD_EMAC_REF_CLK_EN to disable PAD_EMAC_REF_CLK outputted to GPIO.
         · Configure HP_SYS_CLKRST_REF_50M_CLK_EN to disable PLL_F50M_CLK.
       * Reference clock sourced from ESP32-P4:
         · Configure HP_SYS_CLKRST_PAD_EMAC_REF_CLK_EN to enable PAD_EMAC_REF_CLK outputted to GPIO.
         · Configure HP_SYS_CLKRST_REF_50M_CLK_DIV_NUM to specify the clock divisor for PLL_F50M_CLK.
         · Configure HP_SYS_CLKRST_REF_50M_CLK_EN to enable PLL_F50M_CLK.
     - Configure HP_SYS_CLKRST_EMAC_RMII_CLK_SRC_SEL to specify the clock source of EMAC_RMII_CLK as PAD_EMAC_TXRX_CLK;
     - Configure HP_SYS_CLKRST_EMAC_RX_CLK_SRC_SEL to specify the clock source of EMAC_RX_CLK as PAD_EMAC_TXRX_CLK;
     - Configure HP_SYS_CLKRST_EMAC_RX_CLK_DIV_NUM to specify the clock divisor for EMAC_RX_CLK according to the EMAC line speed (10 Mbit/s or 100 Mbit/s);
     - Configure HP_SYS_CLKRST_EMAC_TX_CLK_SRC_SEL to specify the clock source of EMAC_TX_CLK as PAD_EMAC_TXRX_CLK;
     - Configure HP_SYS_CLKRST_EMAC_TX_CLK_DIV_NUM to specify the clock divisor for EMAC_TX_CLK according to the EMAC line speed (10 Mbit/s or 100 Mbit/s);
     - Configure LP_AONCLKRST_HP_PAD_EMAC_TXRX_CLK_EN to enable the clock from IO pin PAD_EMAC_TXRX_CLK;
     - Configure HP_SYS_CLKRST_EMAC_RMII_CLK_EN to enable EMAC_RMII_CLK;
     - Configure HP_SYS_CLKRST_EMAC_RX_CLK_EN to enable EMAC_RX_CLK;
     - Configure HP_SYS_CLKRST_EMAC_TX_CLK_EN to enable EMAC_TX_CLK;
   - MII interface: select the MII interface by configuring HP_SYS_PHY_INTF_SEL to 0.
     - Configure HP_SYS_CLKRST_EMAC_RX_CLK_SRC_SEL to specify EMAC_RX_CLK as PAD_EMAC_RX_CLK;
     - Configure HP_SYS_CLKRST_EMAC_RX_CLK_DIV_NUM to specify the clock divisor for EMAC_RX_CLK according to the EMAC line speed (10 Mbit/s or 100 Mbit/s);
     - Configure HP_SYS_CLKRST_EMAC_TX_CLK_SRC_SEL to specify EMAC_TX_CLK as PAD_EMAC_TX_CLK;

  - Configure HP_SYS_CLKRST_EMAC_TX_CLK_DIV_NUM to specify the clock divisor for EMAC_TX_CLK according to the EMAC line speed (10 Mbit/s or 100 Mbit/s);
  - Configure LP_AONCLKRST_HP_PAD_EMAC_RX_CLK_EN to enable the clock from IO pin PAD_EMAC_RX_CLK;
  - Configure LP_AONCLKRST_HP_PAD_EMAC_TX_CLK_EN to enable the clock from IO pin PAD_EMAC_TX_CLK;
  - Configure HP_SYS_CLKRST_EMAC_RX_CLK_EN to enable EMAC_RX_CLK;
  - Configure HP_SYS_CLKRST_EMAC_TX_CLK_EN to enable EMAC_TX_CLK;

55.5.2 EMAC Initial Configuration

Initialize EMAC_DMA:

- Reset EMAC by setting SW_RST to 1.
- Wait for the completion of reset until reading 0 from SW_RST, which indicates that the reset is done.
- Read AHB_ST; a value of 0 indicates that all AHB bus transactions have been completed.
- Configure the AHB bus mode for EMAC_DMA via DMABUSMODE_REG.
- Prepare the linked list and configure the linked list base address via DMARXBASEADDR_REG and DMATXBASEADDR_REG.
- Configure the operating mode for EMAC_DMA and EMAC_MTL via DMAOPERATION_MODE_REG.
- Configure whether to enable DMA interrupts via DMAIN_EN_REG.

Initialize EMAC_CORE:

- Read EMACMIIADDR_REG and EMACMIIDATA_REG for the PHY's connection status, operating frequency, and operating mode.
- Configure EMACADDROHIGH_REG and EMACADDROLOW_REG to specify the MAC address.
- Configure EMACFF_REG to select the packet filtering mode.
- Configure EMACFC_REG for flow control.
- Configure whether to mask MAC interrupts via EMACINTMASK_REG.
- Configure the MAC's operating mode for transmission and reception via EMACCONFIG_REG.

55.5.3 Starting Transmission

- Start EMAC_DMA transmission by setting START_STOP_TRANSMISSION_COMMAND to 1.
- Start EMAC_CORE transmission by setting EMACTX to 1.
- Detect the TRANS_INT interrupt if enabled, and wait for the frame transmission to complete.

55.5.4 Starting Reception

- Start EMAC_DMA reception by setting START_STOP_RX to 1.

- Start EMAC_CORE reception by setting EMACRX to 1.
- Detect the RECV_INT interrupt if enabled, and wait for the frame reception to complete.

55.5.5 TX Entering and Exiting the LPI State

- Configure the LPI timer counter via EMACLPITIMERSCONTROL_REG
- Make the transmitter to enter the LPI state via EMACLPI_CSR_REG
- Detect the LPI_INT interrupt, and wait for TLPIEN to be set, which indicates that the transmitter has entered the LPI state
- Bring the transmitter out of the LPI state by clearing LPIEN
- Detect the LPI_INT interrupt, and wait for TLPIEX to be set, which indicates that the transmitter has exited the LPI state

55.5.6 RX Entering and Exiting the LPI State

- Detect the LPI_INT interrupt, and wait for RLPIEN to be set, which indicates that the receiver has entered the LPI state
- Detect the LPI_INT interrupt, and wait for RLPIEX to be set, which indicates that the receiver has exited the LPI state

55.6 Interrupts

ESP32-P4's EMAC can generate the following interrupt signals that will be sent to the Interrupt Matrix.

- ETH_MAC_INTR
- PMT_INTR
- LPT_INTR

There are several internal interrupt sources from EMAC that can generate the above interrupt signals.

The following interrupt sources can generate the ETH_MAC_INTR interrupt signal:

- ABN_INT_SUMM: Triggered when DMAIN_AISE is 1 and the following interrupts are triggered.
  - TRANS_PROC_STOP: Triggered when DMAIN_TSE is 1 and the transmit process stops.
  - TRANS_JABBER_TO: Triggered when DMAIN_TJTE is 1 and the frame size exceeds 2048 bytes (10240 bytes if the jumbo frame is enabled).
  - RECV_OVFLOW: Triggered whenDMAIN_OIE is 1 and the receive buffer overflows during frame reception.
  - TRANS_UNDFLOW: Triggered whenDMAIN_UIE is 1 and the transmit buffer underflows during frame transmission.
  - RECV_BUF_UNAVAIL: Triggered when DMAIN_RBUE is 1, the Host owns the Next Descriptor in the Receive List and the DMA cannot acquire it.
  - RECV_PROC_STOP: Triggered when DMAIN_RSE is 1 and the receive process stops.

  - RECV_WDT_TO: Triggered when DMAIN_RWTE is 1 and the Receive Watchdog Timer has expired while receiving the current frame.
  - EARLY_TRANS_INT: Triggered when DMAIN_ETIE is 1 and the frame has been completely transmitted to MTL TX FIFO.
  - FATAL_BUS_ERR_INT: Triggered when DMAIN_FBEE is 1 and a bus error described in ERROR_BITS occurs.
- NORM_INT_SUMM: Triggered when DMAIN_NISE is 1 and the following interrupts are triggered.
  - TRANS_INT: Triggered when DMAIN_TIE is 1 and the transmission completes.
  - TRANS_BUF_UNAVAIL: Triggered when DMAIN_TBUE is 1, the Host owns the Next Descriptor in the Transmit List and the DMA cannot acquire it.
  - RECV_INT: Triggered when DMAIN_RIE is 1 and the frame has been received.
  - EARLY_RECV_INT: Triggered when DMAIN_ERIE is 1 and the DMA has received the first buffer of the data packet.

  EMAC_PMT_INT: Triggered when PMTINTMASK is 0, and a magic packet or a remote wakeup frame has been received in power-down mode.

  TS_TRI_INT: Triggered when TSINTMASK is 0 and any of the following conditions is true:
  - The system time value equals or exceeds the value specified in the Target Time registers
  - There is an overflow in the System Time Seconds Register

  EMAC_LPI_INT: Triggered when LPIINTMASK is 0 and any of the following conditions is true:
  - The MAC Transmitter enters the LPI state
  - The MAC Transmitter exits the LPI state
  - The MAC Receiver enters the LPI state
  - The MAC Receiver exits the LPI state

The following interrupt sources can generate the PMT_INTR interrupt signal:

- RWKPRCVD: Triggered when a remove wakeup frame is received.
- MGKPRCVD: Triggered when a magic packet is received.

The following interrupt source can generate the LPI_INTR interrupt signal:

- RLPIEX: Triggered when the MAC receiver has stopped receiving the LPI pattern on PHY and has exited the LPI state.

55.7 Register Summary

The addresses in this section are relative to EMAC base address provided in Table 9.3-2 in Chapter 9 *System and Memory*.

The abbreviations given in Column **Access** are explained in Section *Access Types for Registers*.

Name	Description	Address	Access
DMA configuration and control registers			
DMABUSMODE_REG	Bus mode register	0x1000	R/WS/SC
DMATXPOLLDEMAND_REG	Transmit poll demand register	0x1004	RO/WT
DMARXPOLLDEMAND_REG	Receive poll demand register	0x1008	RO/WT
DMARXBASEADDR_REG	Base address of the first receive descriptor	0x100C	R/W
DMATXBASEADDR_REG	Base address of the first transmit descriptor	0x1010	R/W
DMASTATUS_REG	Base address of interrupt, error and other events	0x1014	R/SS/WC
DMAOPERATION_MODE_REG	Operation mode and command register	0x1018	R/SS/WC
DMAIN_EN_REG	Interrupt disable and enable register	0x101C	R/W
DMAMISSEDFR_REG	Missed frame and buffer overflow counter register	0x1020	R/W
DMARINTWDTIMER_REG	Receive watchdog timer counter register	0x1024	R/W
DMAAHBSTATUS_REG	AHB master interface status register	0x102C	RO
DMATXCURRDESC_REG	Current transmit descriptor register	0x1048	RO
DMARXCURRDESC_REG	Current receive descriptor register	0x104C	RO
DMATXCURRADDR_BUF_REG	Current transmit buffer address register	0x1050	RO
DMARXCURRADDR_BUF_REG	Current receive buffer address register	0x1054	RO
MAC configuration and control registers			
EMACCONFIG_REG	MAC configuration register	0x0000	R/W
EMACFF_REG	Frame filter register	0x0004	R/W
EMACMIIADDR_REG	PHY access permission configuration register	0x0010	R/WS/SC
EMACMIIDATA_REG	PHY write and read data register	0x0014	R/W
EMACFC_REG	Frame flow control register	0x0018	varies
EMACVLANTAG_REG	802.1Q VLAN tag register	0x001C	R/W
EMACDEBUG_REG	Status and debug register	0x0024	RO
PMT_RWUFFR_REG	Wake-up frame filter register	0x0028	RO

Name	Description	Address	Access
PMT_CSR_REG	Power management control and status register	0x002C	RO
EMACLPI_CSR_REG	LPI control and status register	0x0030	RO
EMACLPITIMERSCONTROL_REG	LPI timer control register	0x0034	RO
EMACINTS_REG	Interrupt status register	0x0038	RO
EMACINTMASK_REG	Interrupt mask register	0x003C	R/W
EMACADDR0HIGH_REG	The higher 16 bits of the first 6-byte MAC address	0x0040	R/W
EMACADDR0LOW_REG	The lower 32 bits of the first 6-byte MAC address	0x0044	R/W
EMACADDR1HIGH_REG	The higher 16 bits of the second 6-byte MAC address	0x0048	R/W
EMACADDR1LOW_REG	The lower 32 bits of the second 6-byte MAC address	0x004C	R/W
EMACADDR2HIGH_REG	The higher 16 bits of the third 6-byte MAC address	0x0050	R/W
EMACADDR2LOW_REG	The lower 32 bits of the third 6-byte MAC address	0x0054	R/W
EMACADDR3HIGH_REG	The higher 16 bits of the fourth 6-byte MAC address	0x0058	R/W
EMACADDR3LOW_REG	The lower 32 bits of the fourth 6-byte MAC address	0x005C	R/W
EMACADDR4HIGH_REG	The higher 16 bits of the fifth 6-byte MAC address	0x0060	R/W
EMACADDR4LOW_REG	The lower 32 bits of the fifth 6-byte MAC address	0x0064	R/W
EMACADDR5HIGH_REG	The higher 16 bits of the sixth 6-byte MAC address	0x0068	R/W
EMACADDR5LOW_REG	The lower 32 bits of the sixth 6-byte MAC address	0x006C	R/W
EMACADDR6HIGH_REG	The higher 16 bits of the seventh 6-byte MAC address	0x0070	R/W
EMACADDR6LOW_REG	The lower 32 bits of the seventh 6-byte MAC address	0x0074	R/W
EMACADDR7HIGH_REG	The higher 16 bits of the eighth 6-byte MAC address	0x0078	R/W
EMACADDR7LOW_REG	The lower 32 bits of the eighth 6-byte MAC address	0x007C	R/W
EMACADDR8HIGH_REG	The higher 16 bits of the ninth 6-byte MAC address	0x0080	R/W
EMACADDR8LOW_REG	The lower 32 bits of the ninth 6-byte MAC address	0x0084	R/W
EMACCSTATUS_REG	Interconnection status	0x00D8	RO

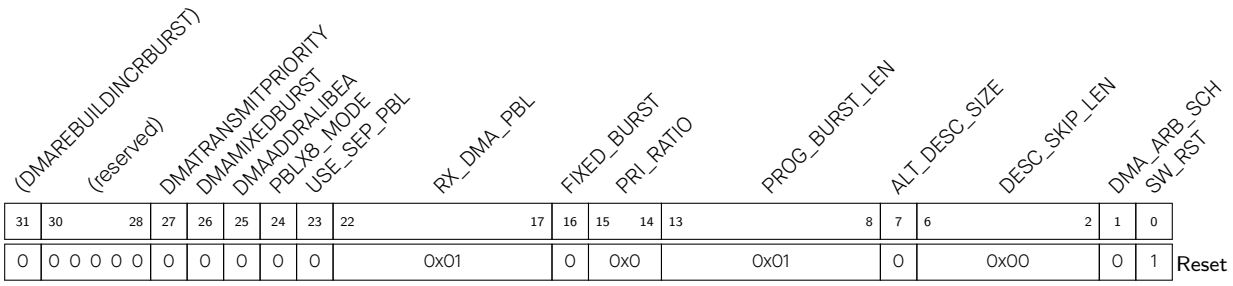
Name	Description	Address	Access
EMACWDOGTO_REG	Watchdog timer timeout control register	0x00DC	R/W
EMACTXVLTCTRL_REG	TX VLAN tag control register	0x0584	R/W
EMACTSTPCTRL_REG	Timestamp control register	0x0700	R/W
EMACSUB2NDINCR_REG	Sub-second increment register	0x0704	R/W
EMACSYSTIM2ND_REG	System time second register	0x0708	RO
EMACSYSTIMNS_REG	System time nanosecond register	0x070C	RO
EMAC2NDUPT_REG	System time second update register	0x0710	R/W
EMACNSUPT_REG	System time nanosecond update register	0x0714	R/W
EMACTSTPADDEND_REG	Fine update mode addend	0x0718	R/W
EMACTGTIME2ND_REG	Target time second register	0x071C	R/W
EMACTGTIMENS_REG	Target time nanosecond register	0x0720	R/W
EMACSYSTIMHIGHWORD2ND_REG	System time nanosecond register (bit 32 to bit 47)	0x0724	R/W/SU
EMACTSTPSTATUS_REG	Timestamp status register	0x0728	R/SS/RC

55.8 Registers

The addresses in this section are relative to EMAC base address provided in Table 9.3-2 in Chapter 9 *System and Memory*.

For how to program reserved fields, please refer to Section Programming Reserved Register Field.

Register 55.1. DMABUSMODE_REG (0x1000)



DMAREBUILDINCRBURST Configures whether to rebuild INCRx burst.

0: Not rebuild

1: If the AHB master gets an EBT (retry, split, or losing bus grant), the AHB master interface rebuilds the pending beats of any burst transfer initiated with INCRx using INCRx and SINGLE. (R/W)

DMATRANSMITPRIORITY Configures whether the transmit DMA has higher priority than the receive DMA during arbitration.

0: Lower priority

1: Higher priority

(R/W)

DMAMIXEDBURST Configures whether to use mixed burst.

0: Not use

1: If the FIXED_BURST bit is 1, the AHB Master interface starts all bursts of length more than 16 with INCR (undefined burst) whereas it reverts to fixed burst transfers (INCRx and SINGLE) for burst length of 16 and less.

(R/W)

DMAADDRALIBEA Configures whether bursts are aligned to address.

0: Not aligned to address

1: If the FIXED_BURST bit is 1, the AHB interface generates all bursts aligned to the start address LS bits. If the FIXED_BURST bit is 0, the first burst (accessing the data buffer's start address) is not aligned, but subsequent bursts are aligned to the address.

(R/W)

Continued on the next page...

Register 55.1. DMABUSMODE_REG (0x1000)

Continued from the previous page...

PBLX8_MODE Configures whether to multiply the PBL (programmable burst length) value by 8.
0: Disable
1: The programmed PBL value (PROG_BURST_LEN and RX_DMA_PBL) is multiplied by 8. Therefore, the DMA transfers the data in 8, 16, 32, 64, 128, and 256 beats depending on the PBL value.
(R/W)

USE_SEP_PBL Configures whether to use separate PBL.
0: PROG_BURST_LEN is applicable only for TX DMA, and RX DMA uses RX_DMA_PBL.
1: PROG_BURST_LEN is applicable for both TX and RX DMA.
(R/W)

RX_DMA_PBL Configure the maximum number of beats to be transferred in one RX DMA transaction, namely the maximum value that is used in a single block Read or Write.
Value range: 1, 2, 4, 8, 16 and 32. Any other value results in undefined behavior.
Valid only when USE_SEP_PBL is set.
(R/W)

FIXED_BURST Configures whether the AHB Master interface performs fixed burst transfers.
0: The AHB interface uses SINGLE and INCR burst transfer operations.
1: The AHB interface uses only SINGLE, INCR4, INCR8, or INCR16 during start of the normal burst transfers.
(R/W)

PRI_RATIO Configures the priority ratio in the weighted round-robin arbitration between the RX DMA and TX DMA.
0: 1:1
1: 2:0
2: 3:1
3: 4:1
Valid only when DMA_ARB_SCH is 0. (R/W)

PROG_BURST_LEN Configures the maximum number of beats to be transferred in one DMA transaction.
If the number of beats to be transferred is more than 32, then perform the following steps:
1. Set the PBLx8 mode with PBLX8_MODE
2. Set the PBL with PROG_BURST_LEN
(R/W)

ALT_DESC_SIZE Configures whether to increase the descriptor size to 32 bytes.
0: Keep 16 bytes
1: Increase to 32 bytes
(R/W)

DESC_SKIP_LEN Configures the number of Word, Dword, or Lword (depending on the 32-bit, 64-bit, or 128-bit bus) to skip between two unchained descriptors.
The address skipping starts from the end of the current descriptor to the start of next descriptor.
When this field is 0, the descriptor table is taken as contiguous by the DMA in Ring mode. (R/W)

Continued on the next page...

Register 55.1. DMABUSMODE_REG (0x1000)

Continued from the previous page...

DMA_ARB_SCH Configures the arbitration scheme between the transmit and receive paths.

0: Weighted round-robin with RX:TX or TX:RX. In this case, the priority between the paths is according to the priority specified in PRI_RATIO.

1: Fixed priority (The transmit path has priority over receive path).

(R/W)

SW_RST Configures whether the MAC DMA Controller resets the logic and all internal registers of the MAC.

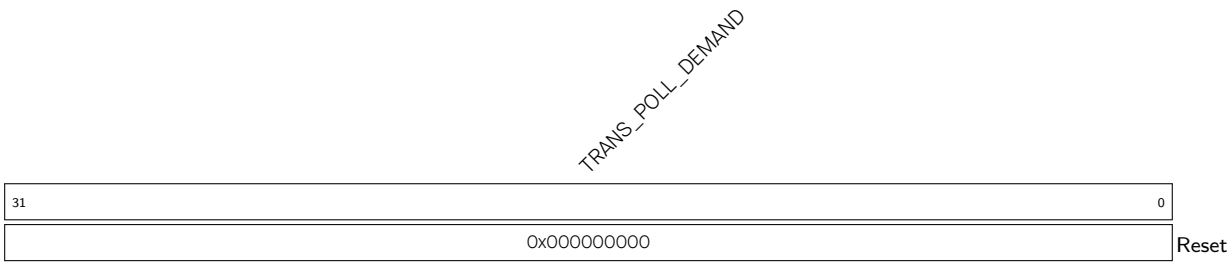
0: Release from reset

1: Reset

This bit is cleared automatically after the reset operation has completed in all of the ETH_MAC clock domains.

(R/WS/SC)

Register 55.2. DMATXPOLLDEMAND_REG (0x1004)

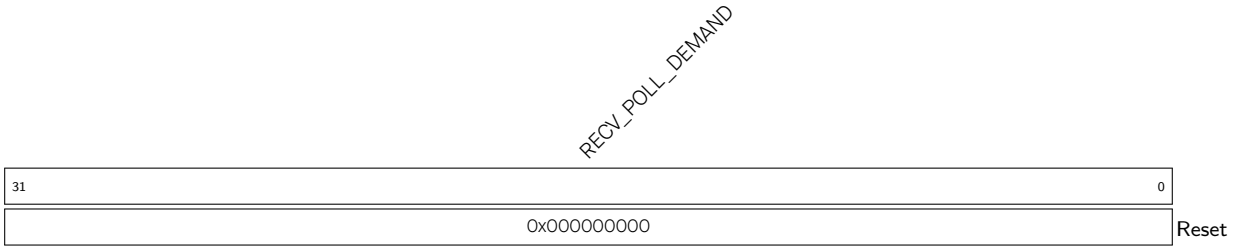


TRANS_POLL_DEMAND Configures whether to enable the TX DMA to check if the DMA owns the current descriptor.

Any value: Enable

When this field is written, the DMA reads the current descriptor pointed to by DMATXCURRDESC_REG. If that descriptor is not available (owned by the Host), the transmission returns to the Suspend state and the Bit 2 (TU) of Register 5 (Status Register) is asserted. If the descriptor is available, the transmission resumes. (RO/WT)

Register 55.3. DMARXPOLLDEMAND_REG (0x1008)

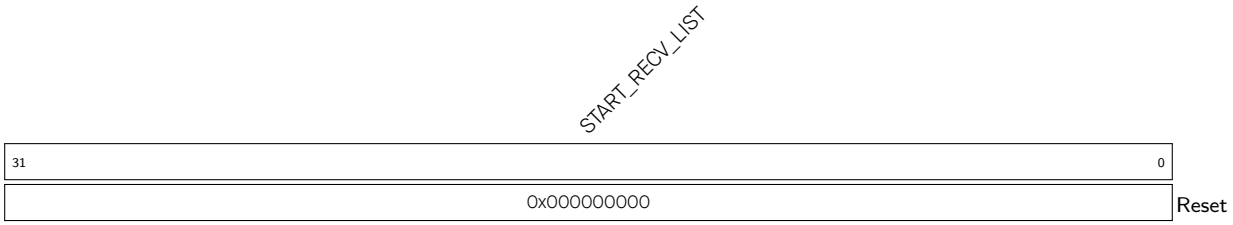


RECV_POLL_DEMAND Configures whether to enable the RX DMA to check if the DMA owns the current descriptor.

Any value: Enable

When this field is written, the DMA reads the current descriptor pointed to by DMARXCURRDESC_REG. If that descriptor is not available (owned by the Host), the reception returns to the Suspended state and the Bit 7 (RU) of Register 5 (Status Register) is not asserted. If the descriptor is available, the RX DMA returns to the active state.(RO/WT)

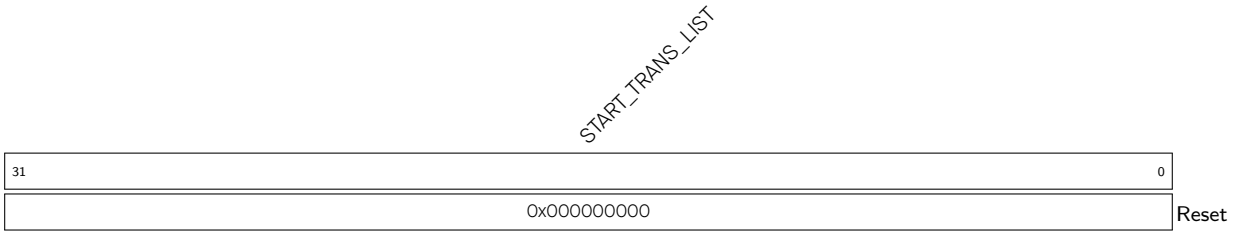
Register 55.4. DMARXBASEADDR_REG (0x100C)



START_RECV_LIST Configures the base address of the first descriptor in the receive descriptor linked list.

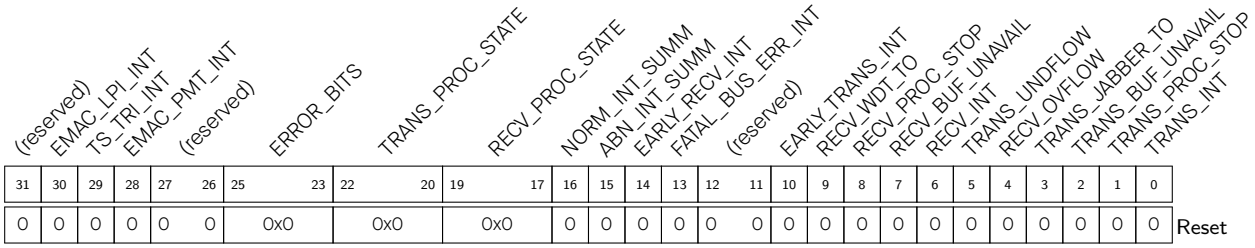
The LSB bits (1:0) are ignored (32-bit wide bus) and internally taken as all-zero by the DMA. Therefore, these LSB bits are read-only (RO). (R/W)

Register 55.5. DMATXBASEADDR_REG (0x1010)



START_TRANS_LIST Configures the base address of the first descriptor in the Transmit Descriptor list. The LSB bits (1:0) are ignored (32-bit wide bus) and are internally taken as all-zero by the DMA. Therefore, these LSB bits are read-only (RO). (R/W)

Register 55.6. DMASTATUS_REG (0x1014)



EMAC_LPI_INT The raw interrupt status of EMAC_LPI_INT. (RO)

TS_TRI_INT The raw interrupt status of TS_TRI_INT. (RO)

EMAC_PMT_INT The raw interrupt status of EMAC_PMT_INT. (RO)

ERROR_BITS Represents the type of error that caused a Bus Error, for example, error response on the AHB interface.

0: Error during RX DMA Write Data Transfer

1: Error during TX DMA Read Data Transfer

2: Error during RX DMA Descriptor Write Access

3: Error during TX DMA Descriptor Write Access

4: Error during RX DMA Descriptor Read Access

5: Error during TX DMA Descriptor Read Access

Valid only when Bit 13 (FBI) is 1. This field does not generate an interrupt. (RO)

TRANS_PROC_STATE Represents the Transmit DMA FSM state.

0: Stopped; Reset or Stop Transmit Command issued

1: Running; Fetching Transmit Transfer Descriptor

2: Running; Waiting for status

3: Running; Reading Data from Host memory buffer and queuing it to transmit buffer (TX FIFO)

4: TIME_STAMP write state

5: Reserved for future use

6: Suspended; Transmit Descriptor Unavailable or Transmit Buffer Underflow

7: Running; Closing Transmit Descriptor

This field does not generate an interrupt. (RO)

Continued on the next page...

Register 55.6. DMASTATUS_REG (0x1014)

Continued from the previous page...

NORM_INT_SUMM Represents the logical OR of the following when the corresponding interrupt bits are enabled in Interrupt Enable Register.
Bit[0]: Transmit Interrupt
Bit[2]: Transmit Buffer Unavailable
Bit[6]: Receive Interrupt
Bit[14]: Early Receive Interrupt
Only unmasked interrupts affect this bit.
This is a sticky bit and must be cleared (by writing 1 to this bit) each time a corresponding bit, which causes this set to be set, is cleared. (R/SS/WC)

ABN_INT_SUMM Represents the logical OR of the following when the corresponding interrupt bits are enabled in Interrupt Enable Register.
Bit[1]: Transmit Process Stopped
Bit[3]: Transmit Jabber Timeout
Bit[4]: Receive FIFO Overflow
Bit[5]: Transmit Underflow
Bit[7]: Receive Buffer Unavailable
Bit[8]: Receive Process Stopped
Bit[9]: Receive Watchdog Timeout
Bit[10]: Early Transmit Interrupt
Bit[13]: Fatal Bus Error
Only unmasked interrupts affect this bit.
This is a sticky bit and must be cleared each time a corresponding bit, which causes this bit to be set, is cleared. (R/SS/WC)

EARLY_RECV_INT The raw interrupt status of EARLY_RECV_INT.(R/SS/WC)

FATAL_BUS_ERR_INT The raw interrupt status of FATAL_BUS_ERR_INT. (R/SS/WC)

EARLY_TRANS_INT The raw interrupt status of EARLY_TRANS_INT. (R/SS/WC)

RECV_WDT_TO Represents whether the Receive Watchdog Timer expired while receiving the current frame and the current frame is truncated after the watchdog timeout.
0: Not expired
1: Expired
(R/SS/WC)

RECV_PROC_STOP Represents whether the Receive Process enters the Stopped state.
0: Not stopped
1: Stopped
(R/SS/WC)

Continued on the next page...

Register 55.6. DMASTATUS_REG (0x1014)

Continued from the previous page...

RECV_BUF_UNAVAIL Represents whether the Receive Buffer is available.

0: Available

1: Unavailable

This bit is set only when the previous Receive Descriptor is owned by the DMA.

When set, the Host owns the Next Descriptor in the Receive List and the DMA cannot acquire it. The Receive Process is suspended.

To resume processing Receive descriptors, the Host should change the ownership of the descriptor and issue a Receive Poll Demand command. If no Receive Poll Demand is issued, the Receive Process resumes when the next recognized incoming frame is received. (R/SS/WC)

RECV_INT The raw interrupt status of RECV_INT. (R/SS/WC)

TRANS_UNDFLOW Represents whether the Transmit Buffer had an Underflow during frame transmission.

0: No underflow

1: Underflow

When this bit is 1, transmission is suspended and an Underflow Error TDES0[1] is set. (R/SS/WC)

RECV_OVFLOW Represents whether the Receive Buffer had an Overflow during frame reception.

0: No overflow

1: Overflow

If the partial frame is transferred to the application, the overflow status is set in RDES0[11]. (R/SS/WC)

TRANS_JABBER_TO Represents whether the Transmit Jabber Timer expired, which happens when the frame size exceeds 2,048 (10,240 bytes when the Jumbo frame is enabled).

0: Not expired

1: Expired

When the Jabber Timeout occurs, the transmission process is aborted and placed in the Stopped state. This causes the Transmit Jabber Timeout TDES0[14] flag to assert. (R/SS/WC)

TRANS_BUF_UNAVAIL Represents whether the Transmit Buffer is available.

0: Available

1: Unavailable

When this bit is 1, the Host owns the Next Descriptor in the Transmit List and the DMA cannot acquire it. Transmission is suspended. Bits[22:20] explain the Transmit Process state transitions.

To resume processing Transmit descriptors, the Host should change the ownership of the descriptor by setting TDES0[31] and then issue a Transmit Poll Demand command. (R/SS/WC)

TRANS_PROC_STOP Represents whether the transmission is stopped.

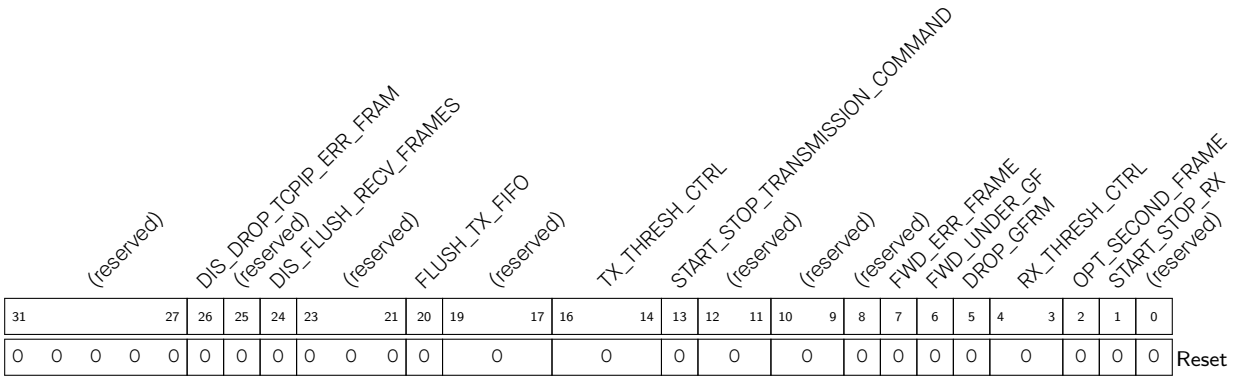
0: Not stopped

1: Stopped

(R/SS/WC)

TRANS_INT The raw interrupt status of TRANS_INT. (R/SS/WC)

Register 55.7. DMAOPERATION_MODE_REG (0x1018)



DIS_DROP_TCPIP_ERR_FRAM Configures whether to disable dropping of TCP/IP checksum error frames.

0: Enable

1: Disable

When FWD_ERR_FRAME is 0, this bit is ignored and all error frames are dropped. (R/W)

DIS_FLUSH_RECV_FRAMES Configures whether to disable flushing of received frames because of the unavailability of receive descriptors or buffers.

0: Enable

1: Disable

(R/W)

FLUSH_TX_FIFO Configures whether to flush TX FIFO to default values.

0: Not flush

1: Flush

(R/WS/SC)

TX_THRESH_CTRL Configures the threshold of TX FIFO. Transmission starts when the frame size within the TX FIFO is larger than the threshold. In addition, full frames with a length less than the threshold are also transmitted.

0: 64

1: 128

2: 192

3: 256

4: 40

5: 32

6: 24

7: 16

Transmission starts when the frame size within the TX FIFO is larger than the threshold. In addition, full frames with a length less than the threshold are also transmitted.

(R/W)

Continued on the next page...

Register 55.7. DMAOPERATION_MODE_REG (0x1018)

Continued from the previous page...

START_STOP_TRANSMISSION_COMMAND Configures whether to start or stop the transmission.

0: Stop

1: Start

(R/W)

FWD_ERR_FRAME Configures whether RX FIFO forward frames with error status (CRC error, collision error, giant frame, watchdog timeout, or overflow)

0: Drop

1: Forward

(R/W)

FWD_UNDER_GF Configures whether the RX FIFO forward undersized (frames with no Error and length less than 64 bytes) good frames including PAD and CRC.

0: Drop

1: Forward

(R/W)

DROP_GFRM Configures whether to the ETH_MAC drops the received giant frames in the RX FIFO.

0: Not drop

1: Drop

(R/W)

RX_THRESH_CTRL Configures the threshold of the RX FIFO.

0: 64

1: 32

2: 96

3: 128

Transfer (request) to DMA starts when the frame size within the RX FIFO is larger than the threshold. (R/W)

OPT_SECOND_FRAME Configures whether the DMA processes the second frame of the Transmit data even before the status for the first frame is obtained.

0: Not process

1: Process

(R/W)

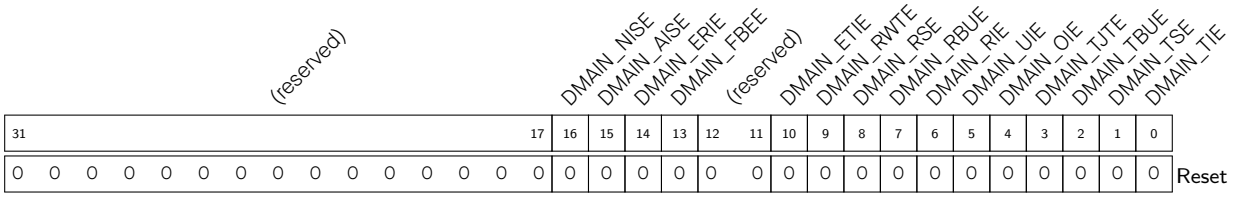
START_STOP_RX Configures whether to start or stop RX DMA reception.

0: Stop reception after the transfer of the current frame

1: Start reception

(R/W)

Register 55.8. DMAIN_EN_REG (0x101C)



DMAIN_NISE Write 1 to enable and write 0 to disable the summary of the following normal interrupt enable bits:
Bit[0]: Transmit Interrupt
Bit[2]: Transmit Buffer Unavailable
Bit[6]: Receive Interrupt
Bit[14]: Early Receive Interrupt
(R/W)

DMAIN_AISE Write 1 to enable and write 0 to disable the summary of the following abnormal interrupt enable bis:
Bit[1]: Transmit Process Stopped
Bit[3]: Transmit Jabber Timeout
Bit[4]: Receive FIFO Overflow
Bit[5]: Transmit Underflow
Bit[7]: Receive Buffer Unavailable
Bit[8]: Receive Process Stopped
Bit[9]: Receive Watchdog Timeout
Bit[10]: Early Transmit Interrupt
Bit[13]: Fatal Bus Error
(R/W)

DMAIN_ERIE Write 1 to enable and write 0 to disable EARLY_RECV_INT. (R/W)

DMAIN_FBEE Write 1 to enable and write 0 to disable FATAL_BUS_ERR_INT.(R/W)

DMAIN_ETIE Write 1 to enable and write 0 to disable EARLY_TRANS_INT. (R/W)

DMAIN_RWTE Write 1 to enable and write 0 to disable RECV_WDT_TO_INT. (R/W)

DMAIN_RSE Write 1 to enable and write 0 to disable RECV_PROC_STOP_INT. (R/W)

DMAIN_RBUE Write 1 to enable and write 0 to disable RECV_BUF_UNAVAIL_INT. (R/W)

DMAIN_RIE Write 1 to enable and write 0 to disable RECV_INT. (R/W)

Continued on the next page...

Register 55.8. DMAIN_EN_REG (0x101C)

Continued from the previous page...

DMAIN_UIE Write 1 to enable and write 0 to disable TRANS_UNDFLOW_INT. (R/W)

DMAIN_OIE Write 1 to enable and write 0 to disable RECV_OVFLOW_INT. (R/W)

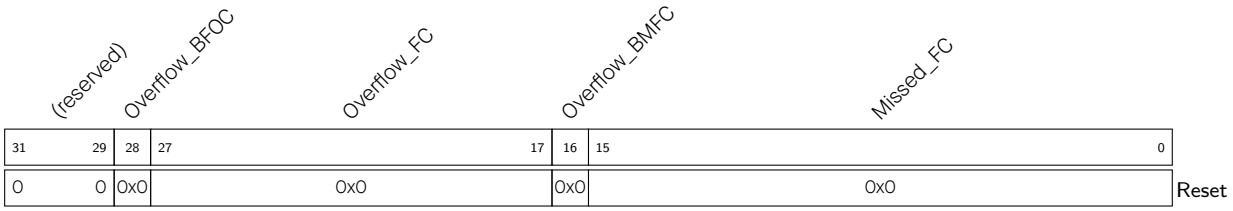
DMAIN_TJTE Write 1 to enable and write 0 to disable TRANS_JABBER_TO_INT. (R/W)

DMAIN_TBUE Write 1 to enable and write 0 to disable TRANS_BUF_UNAVAIL_INT. (R/W)

DMAIN_TSE Write 1 to enable and write 0 to disable TRANS_PROC_STOP_INT.(R/W)

DMAIN_TIE Write 1 to enable and write 0 to disable TRANS_INT.(R/W)

Register 55.9. DMAMISSEDFR_REG (0x1020)



Overflow_BFOC Represents the status of the overflow frame counter.

0: No overflow.

1: The overflow frame counter (Overflow_FC) overflows, that is, the RX FIFO overflows with the overflow frame counter at maximum value. In such a scenario, the overflow frame counter is reset to all-zeros.

(R/SS/RC)

Overflow_FC Represents the number of frames missed by the application.

This counter is incremented each time the MTL FIFO overflows. (R/SS/RC)

Overflow_BMFC Represents the status of the missed frame counter.

0: No overflow.

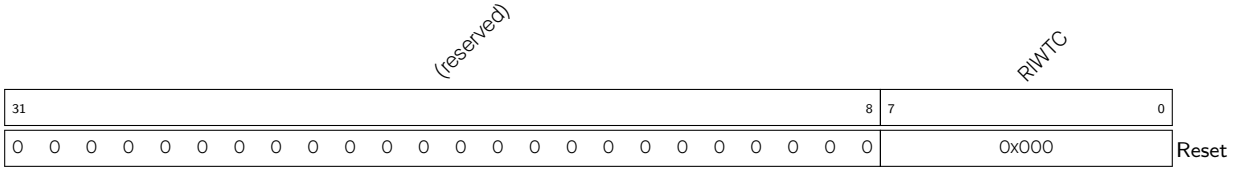
1: The missed frame counter (Missed_FC) overflows, that is, the Host Receive Buffer being unavailable with the missed frame counter at maximum value. In such a scenario, the Missed frame counter is reset to all-zeros.

(R/SS/RC)

Missed_FC Represents the number of frames missed by the controller because of the Host Receive Buffer being unavailable.

This counter is incremented each time the DMA discards an incoming frame. (R/SS/RC)

Register 55.10. DMARINTWDTIMER_REG (0x1024)

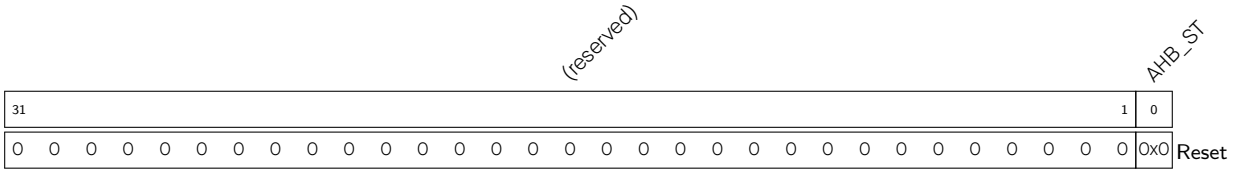


RIWTC Configures the number of system clock cycles multiplied by 256.

The watchdog timer gets triggered with the programmed value after the RX DMA completes the transfer of a frame for which the RI (RECV_INT) status bit is not set because of the setting in the corresponding descriptor RDES1[31].

When the watchdog timer runs out, the RI bit is set and the timer is stopped. The watchdog timer is reset when the RI bit is set high because of the automatic setting of RI as per RDES1[31] of any received frame. (R/W)

Register 55.11. DMAAHBSTATUS_REG (0x102C)



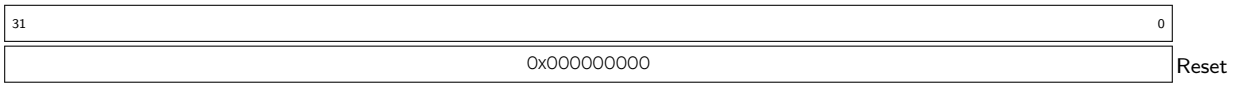
AHB_ST Configures the state of the AHB master interface.

0: Idle

1: Active

(R/W)

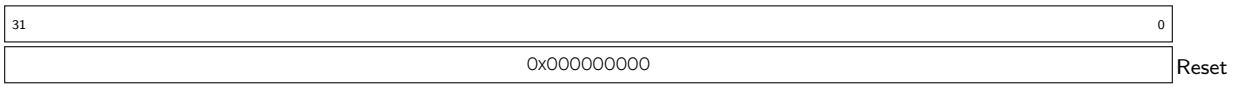
Register 55.12. DMATXCURRDESC_REG (0x1048)



TRANS_DECR_ADDR_PTR Represents the address of the current transmit descriptor list updated by the DMA during operation.

This field is cleared on reset. (RO)

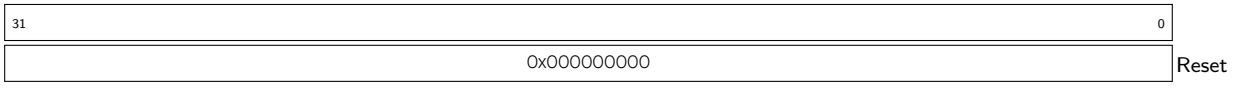
Register 55.13. DMARXCURRDESC_REG (0x104C)



RECV_DECR_ADDR_PTR Represents the address of the current receive descriptor list updated by the DMA during operation.

This field is cleared on reset. (RO)

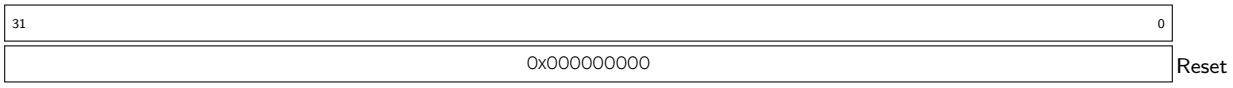
Register 55.14. DMATXCURRADDR_BUF_REG (0x1050)



TRANS_BUFF_ADDR_PTR Represents the address of the Transmit Buffer updated by the DMA during operation.

This field is cleared on reset. (RO)

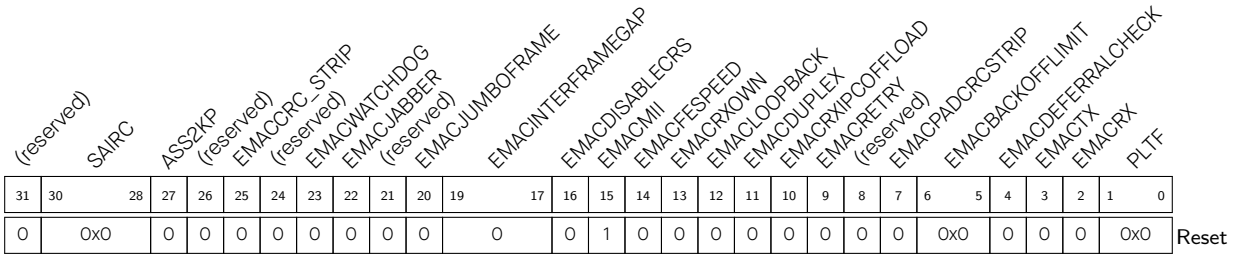
Register 55.15. DMARXCURRADDR_BUF_REG (0x1054)



RECV_BUFF_ADDR_PTR Represents the address of the Receive Buffer updated by the DMA during operation.

This field is cleared on reset. (RO)

Register 55.16. EMACCONFIG_REG (0x0000)



SAIRC Configures the source address insertion or replacement for all transmitted frames.

Bit 30 specifies which MAC Address register (0 or 1) is used for source address insertion or replacement based on the values of Bits [29:28].

Bits [29:28]:

0, 1: The input mti_sa_ctrl_i and ati_sa_ctrl_i control the SA field generation

2: Insert the content of the MAC Address register (0 or 1) in the SA field of all transmitted frames

3: Replace the the content of MAC Address register (0 or 1) in the SA field of all transmitted frames

Bit 30:

0: Use MAC Address 0 register

1: Use MAC Address 1 register

(R/W)

ASS2KP Configures whether the MAC considers received frames of more than 2000 bytes as normal packets or giant frames.

0: Giant frames. In this case, if Bit[20] (JE) is 0, the MAC considers all received frames of size more than 1,518 bytes (1,522 bytes for tagged) as Giant frames.

1: Normal packets.

(R/W)

EMACCRC_STRIP Configures whether to strip and drop the last 4 bytes (FCS) of all frames of Ether type (Length/Type field greater than or equal to 1,536) before forwarding the frame to the application.

0: Not strip and drop

1: Strip and drop

(R/W)

EMACWATCHDOG Configures whether to disable the watchdog timer.

0: Enable. In this case, the MAC does not allow a receive frame with more than 2,048 bytes (10,240 if JE is set high) or the value programmed in Watchdog Timeout Register. The MAC cuts off any bytes received after the watchdog limit number of bytes.

1: Disable. In this case, the MAC can receive frames of up to 16,384 bytes.

(R/W)

EMACJABBER Configures whether to disable the jabber timer on the transmitter.

0: Enable. In this case, the MAC cuts off the transmitter if the application sends out more than 2,048 bytes of data (10,240 if JE is set high) during transmission.

1: Disable. In this case, the MAC can transfer frames of up to 16,384 bytes.

(R/W)

Continued on the next page...

Register 55.16. EMACCONFIG_REG (0x0000)

Continued from the previous page...

EMACJUMBOFRAME Configures whether the MAC allows Jumbo frames of 9,018 bytes (9,022 bytes for VLAN tagged frames) without reporting a giant frame error in the receive frame status.
0: Not allow
1: Allow
(R/W)

EMACINTERFRAMEGAP Configures the minimum interframe gap (IFG) between frames during transmission.
0: 96 bit times
1: 88 bit times
2: 80 bit times
3: 72 bit times
4: 64 bit times
5: 56 bit times
6: 48 bit times
7: 40 bit times
In the half-duplex mode, the minimum IFG can be configured only for 64 bit times (IFG = 100).
(R/W)

EMACDISABLECRS Configures whether to disable Carrier Sense during transmission.
0: The MAC transmitter generates Loss of Carrier or No Carrier errors because of Carrier Sense and can even abort the transmissions.
1: The MAC transmitter ignores the MII CRS signal during frame transmission in the half-duplex mode. This request results in no errors generated because of Loss of Carrier or No Carrier during such transmission.
(R/W)

EMACMII Configures the Ethernet line speed.
0: For 1000 Mbps operations
1: For 10 or 100 Mbps operations
In 10 or 100 Mbps operations, the exact line speed is determined by both this bit and the FES (EMACFESPEED) bit. (R/W)

EMACFESPEED Configures the speed in the MII and RMII interface.
0: 10 Mbps
1: 100 Mbps
(R/W)

EMACRXOWN Configures whether to disable receive own.
0: The MAC receives all packets that are given by the PHY while transmitting.
1: the MAC disables the reception of frames when TX_EN triggers an interrupt in the half-duplex mode.
This bit is not applicable if the MAC is operating in the full-duplex mode. (R/W)

Continued on the next page...

Register 55.16. EMACCONFIG_REG (0x0000)

Continued from the previous page...

EMACLOOPBACK Configures whether to enable the loopback mode.

0: Disable.

1: Enable. In this case, the MII Receive clock input (CLK_RX) is required for the loopback to work properly, because the Transmit clock is not looped-back internally.

(R/W)

EMACDUPLEX Configures whether to enable full-duplex mode, so that the MAC can transmit and receive simultaneously.

0: Disable

1: Enable

This bit is RO with a default value of 1 in the full-duplex-only configuration. (R/W)

EMACRXIPCOFFLOAD Configures whether to enable the checksum offload function

0: Disable

1: Enable

When the checksum offload function is enabled, the MAC calculates the 16-bit one's complement of the one's complement sum of all received Ethernet frame payloads. It also checks whether the IPv4 Header checksum (assumed to be bytes 25/26 or 29/30 (VLAN-tagged) of the received Ethernet frame) is correct for the received frame and gives the status in the receive status word. The MAC also appends the 16-bit checksum calculated for the IP header datagram payload (bytes after the IPv4 header) and appends it to the Ethernet frame transferred to the application (when Type 2 COE is deselected). (R/W)

EMACRETRY Configures whether to disable retry, so that the MAC attempts only one transmission, and when a collision occurs on the MII interface the MAC ignores the current frame transmission and reports a Frame Abort with excessive collision error in the transmit frame status.

0: Enable

1: Disable

Valid only in the half-duplex mode. (R/W)

EMACPADCRCSTRIP Configures whether to enable automatic Pad or CRC stripping.

0: Disable. In this case, the MAC passes all incoming frames, without modifying them, to the Host.

1: Enable. In this case, the MAC strips the Pad or FCS field on the incoming frames only if the value of the length field is less than 1,536 bytes. All received frames with length field greater than or equal to 1,536 bytes are passed to the application without stripping the Pad or FCS field. (R/W)

Continued on the next page...

Register 55.16. EMACCONFIG_REG (0x0000)

Continued from the previous page...

EMACBACKOFFLIMIT Configures the back-off limit determining the random integer number (r) of slot time delays (4,096 bit times for 1000 Mbps and 512 bit times for 10/100 Mbps) for which the MAC waits before rescheduling a transmission attempt during retries after a collision.
00: k = min (n, 10)
01: k = min (n, 8)
10: k = min (n, 4)
11: k = min (n, 1)
where n = retransmission attempt.
The random integer r takes the value in the range 0 <= r < kth power of 2
Valid only in the half-duplex mode. (R/W)

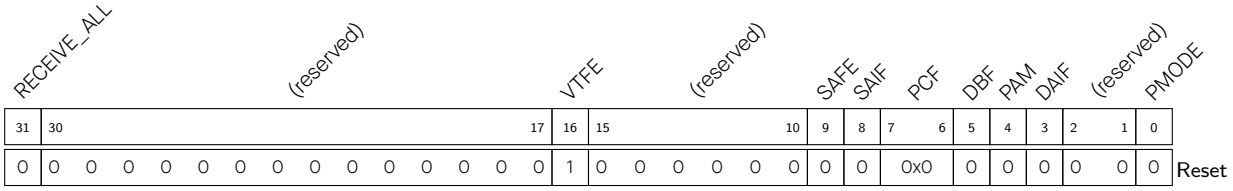
EMACDEFERRALCHECK Configures whether to enable the deferral check function.
0: Disable
1: Enable
(R/W)

EMACTX Configures whether to enable the transmit state machine of the MAC.
0: Disabled after the completion of the transmission of the current frame, and does not transmit any further frames
1: Enabled for transmission on the MII
(R/W)

EMACRX Configures whether to enable the receiver state machine of the MAC. 0: Disabled after the completion of the reception of the current frame, and does not receive any further frames from the MII
1: Enabled for receiving frames from the MII
(R/W)

PLTF Configures the number of preamble bytes that are added to the beginning of every Transmit frame.
0: 7 bytes of preamble
1: 5 byte of preamble
2: 3 bytes of preamble
3: Reserved
The preamble reduction occurs only when the MAC is operating in the full-duplex mode. (R/W)

Register 55.17. EMACFF_REG (0x0004)



RECEIVE_ALL Configures whether to pass all received modules.

0: The MAC Receiver module passes only those frames to the Application that pass the SA or DA address filter.

1: The MAC Receiver module passes all received frames, irrespective of whether they pass the address filter or not, to the Application. The result of the SA or DA filtering is updated (pass or fail) in the corresponding bits in the Receive Status Word.

(R/W)

VTFE Configures whether to enable the VLAN Tag filter.

0: Disable. In this case, the MAC forwards all frames irrespective of the match status of the VLAN Tag.

1: Enable. In this case, the MAC drops VLAN-tagged frames that do not match the VLAN Tag comparison.

(R/W)

SAFE Configures whether to enable the source address filter.

0: Disable. In this case, the MAC forwards the received frame to the application with an updated SAF bit of the RX Status depending on the SA address comparison.

1: Enable. In this case, the MAC compares the SA field of the received frames with the values programmed in the enabled SA registers. If the comparison fails, the MAC drops the frame.

(R/W)

SAIF Configures whether to enable SA inverse filtering.

0: Disable. In this case, frames whose SA does not match the SA registers are marked as failing the SA Address filter.

1: Enable. In this case, the Address Check block operates in inverse filtering mode for the SA address comparison. The frames whose SA matches the SA registers are marked as failing the SA Address filter.

(R/W)

Continued on the next page...

Register 55.17. EMACFF_REG (0x0004)

Continued from the previous page...

PCF Configures the forwarding of all control frames (including unicast and multicast PAUSE frames).

0: The MAC filters all control frames.

1: The MAC forwards all control frames except PAUSE control frames to application even if they fail the Address filter.

2: The MAC forwards all control frames to application even if they fail the Address Filter.

3: The MAC forwards control frames that pass the Address Filter.

The following conditions should be true for the PAUSE control frames processing:

Condition 1: The MAC is in the full-duplex mode and flow control is enabled by setting Bit 2 (RFE) of Register 6 (Flow Control Register) to 1.

Condition 2: The destination address (DA) of the received frame matches the special multicast address or EMACADDR0 when Bit 3 (UP) of the Register 6 (Flow Control Register) is set.

Condition 3: The Type field of the received frame is 0x8808 and the OPCODE field is 0x0001.

(R/W)

DBF Configures whether to disable broadcast frames.

0: Enable. The AFM (Address Filtering Module) module passes all received broadcast frames.

1: Disable. The AFM module filters all incoming broadcast frames. In addition, it overrides all other filter settings.

(R/W)

PAM Configures whether to pass all multicast frames.

0: Filtering of multicast frame depends on HMC bit

1: All received frames with a multicast destination address (first bit in the destination address field is '1') are passed

(R/W)

DAIF Configures whether to enable DA inverse filtering for both unicast and multicast frames.

0: Disable

1: Enable

(R/W)

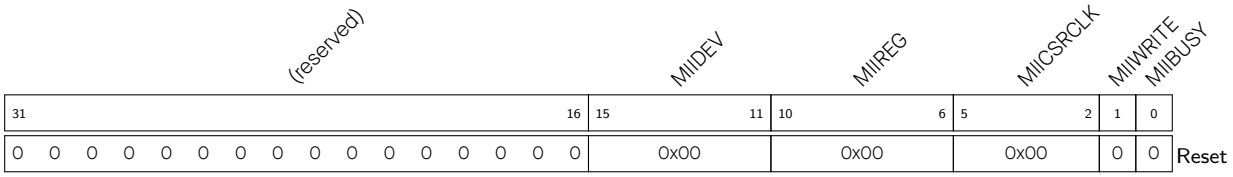
PMODE Configures whether to enable the promiscuous mode.

0: Disable.

1: Enabled. In this case, the Address Filter module passes all incoming frames regardless of its destination or source address. The SA or DA Filter Fails status bits of the Receive Status Word are always cleared when PR (PRI_RATIO) is set.

(R/W)

Register 55.18. EMACMIIADDR_REG (0x0010)



MIIDEV Configures which of the 32 possible PHY devices are being accessed. (R/W)

MIIREG Configures the desired address register in the selected PHY device. (R/W)

MIICSRCLK Configures the CSR clock frequency.

0: APB clock frequency is 80 MHz, and MDC clock frequency is APB_CLK/42

3: APB clock frequency is 40 MHz, and MDC clock frequency is APB_CLK/26

Other values are reserved

(R/W)

MIIWRITE Configures the direction of the operation that uses MII_DATA.

0: Read operation

1: Write operation

(R/W)

MIIBUSY Configures the state of the MII.

0: Idle

1: Busy

This bit is used in combination with MIIREG and MII_DATA.

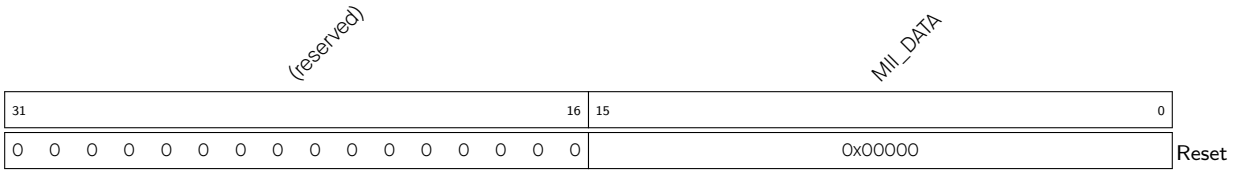
This bit should read logic 0 (default) before writing to MIIREG and MII_DATA.

To read or write to MIIREG and MII_DATA, software (the user) should set this field to 1.

MII_DATA should be kept valid (data remains unchanged) when it is accessed until this field is cleared by hardware (the MAC).

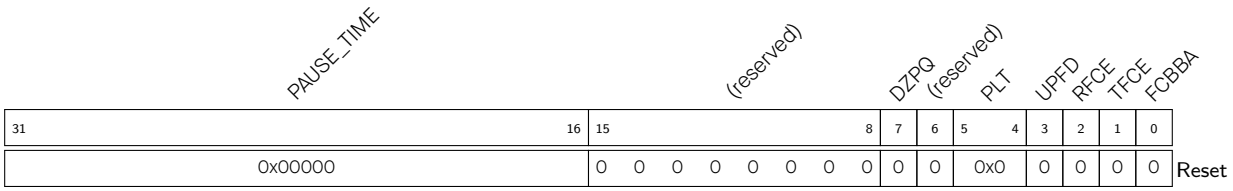
Note that ESP32-P4 MAC does not receive ACK from PHY during a read or write access to MIIREG and MII_DATA. (R/WS/SC)

Register 55.19. EMACMIIDATA_REG (0x0014)



MII_DATA Configures the 16-bit data value read from the PHY after a Management Read operation or the 16-bit data value to be written to the PHY before a Management Write operation.(R/W)

Register 55.20. EMACFC_REG (0x0018)



PAUSE_TIME Configures the value to be used in the Pause Time field in the transmit control frame. If the Pause Time bits is configured to be double-synchronized to the (G)MII clock domain, then consecutive writes to this register should be performed only after at least four clock cycles in the destination clock domain. (R/W)

DZPQ Configures whether to disable the automatic generation of the Zero-Quanta Pause Control frames.

0: Enable

1: Disable

(R/W)

PLT Configures the threshold of the PAUSE timer at which the PAUSE Frame is automatically retransmitted.

0: The threshold is Pause time minus 4 slot times (PT - 4 slot times).

1: The threshold is Pause time minus 28 slot times (PT - 28 slot times).

2: The threshold is Pause time minus 144 slot times (PT - 144 slot times).

3: The threshold is Pause time minus 256 slot times (PT - 256 slot times).

The threshold values should be always less than the Pause Time configured in Bits[31:16]. For example, if PT = 100H (256 slot-times), and PLT = 01, then a second PAUSE frame is automatically transmitted 228 (256 - 28) slot times after the first PAUSE frame is transmitted.

(R/W)

UPFD Configures whether to enable unicast pause frame detection.

0: Disable. The MAC only detects Pause frames with a unique multicast address.

1: Enable. The MAC can also detect Pause frames with unicast address of the station. This unicast address should be as specified in the EMACADDR0 High Register and EMACADDR0 Low Register.

(R/W)

RFCE Configures whether to enable the decode function of the Pause time.

0: Disable

1: Enable. The MAC decodes the received Pause frame and disables its transmitter for a specified (Pause) time

(R/W)

TFCE Configures whether to enable transmit flow control in full-duplex mode, and whether to enable the back-pressure feature in half-duplex mode.

0: Disable

1: Enable

(R/W)

Continued on the next page...

Register 55.20. EMACFC_REG (0x0018)

Continued from the previous page...

FCBBA Configures whether to enable a Pause Control frame in the full-duplex mode, and whether to activate the back-pressure function in the half-duplex mode if the TFCE bit is set.

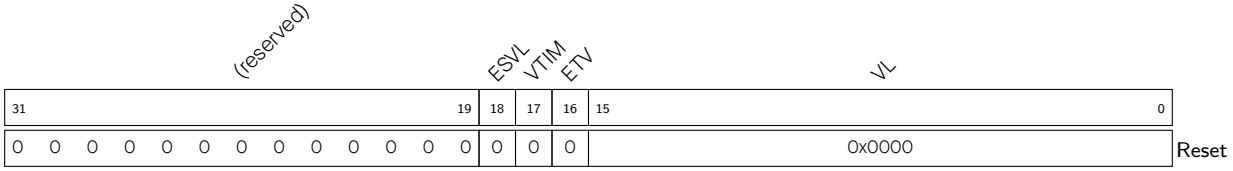
0: Disable

1: Enable

In the full-duplex mode, this bit should be read as 0 before writing to the Flow Control register. To initiate a Pause control frame, the Application must set this bit to 1. During a transfer of the Control Frame, this bit continues to be set to signify that a frame transmission is in progress. After the completion of Pause control frame transmission, the MAC resets this bit to 0. The Flow Control register should not be written to until this bit is cleared. (R/WS/SC)

In the half-duplex mode, when this bit is set (and TFCE is set), then backpressure is asserted by the MAC. During backpressure, when the MAC receives a new frame, the transmitter starts sending a JAM pattern resulting in a collision. When the MAC is configured for the full-duplex mode, the BPA is automatically disabled. (R/W)

Register 55.21. EMACVLANTAG_REG (0x001C)



ESVL Configures whether to enable S-VLAN.

0: Disable.

1: Enable. The MAC transmitter and receiver also consider the S-VLAN (Type = 0x88A8) frames as valid VLAN tagged frames.

(R/W)

VTIM Configures whether to enable VLAN Tag inverse match.

0: Disable. The frames with matched VLAN Tag are marked as matched.

1: Enable. The frames that do not have matching VLAN Tag are marked as matched.

(R/W)

ETV Configures whether to enable 12-bit VLAN Tag comparison.

0: Disable. All 16 bits of the 15th and 16th bytes of the received VLAN frame are used for comparison and VLAN hash filtering.

1: Enable. A 12-bit VLAN identifier is used for comparing and filtering instead of the complete 16-bit VLAN tag. Bits [11:0] of VLAN tag are compared with the corresponding field in the received VLAN-tagged frame.

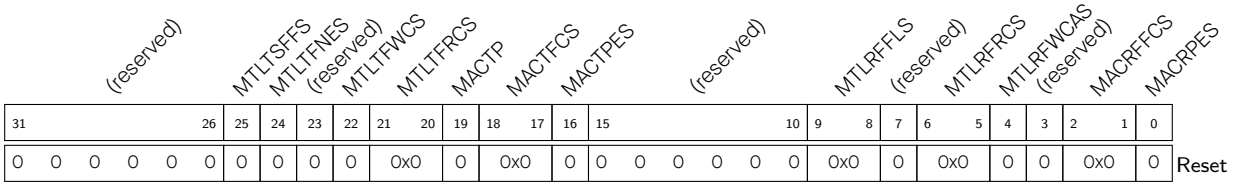
(R/W)

VL Configures the 802.1Q VLAN tag to identify the VLAN frames and is compared to the 15th and 16th bytes of the frames being received for VLAN frames.

When the ETV bit is set, only the VID (Bits[11:0]) is used for comparison.

If VL (VL[11:0] if ETV is set) is all zeros, the MAC does not check the fifteenth and 16th bytes for VLAN tag comparison, and declares all frames with a Type field value of 0x8100 or 0x88a8 as VLAN frames.(R/W)

Register 55.22. EMACDEBUG_REG (0x0024)



MTLTSFFS Represents whether MTL TX Status FIFO is full and can accept any more frames for transmission.

0: Not Full

1: Full

(RO)

MTLTFNES Represents whether MTL TX FIFO is not empty and some data is left for transmission.

0: Empty

1: Not empty

(RO)

MTLTFWCS Represents whether MTL TX FIFO Write Controller is active and transferring data to the TX FIFO.

0: Inactive

1: Active

(RO)

MTLTFRCS Represents the state of the TX FIFO Read Controller.

0: IDLE state

1: READ state (transferring data to MAC transmitter)

2: Waiting for TXStatus from MAC transmitter

3: Writing the received TXStatus or flushing the TX FIFO

(RO)

MACTP Represents whether the MAC transmitter is in the PAUSE condition (in the full-duplex only mode) and hence does not schedule any frame for transmission.

0: Not in pause

1: In pause

(RO)

MACTFCS Represents the state of the MAC Transmit Frame Controller module.

0: IDLE state

1: Waiting for Status of the previous frame or IFG or backoff period to be over

2: Generating and transmitting a PAUSE control frame (in the full-duplex mode)

3: Transferring input frame for transmission

(RO)

MACTPES Represents the status of the MAC MII transmit protocol engine.

0: IDLE state

1: Actively transmitting data

(RO)

Continued on the next page...

Register 55.22. EMACDEBUG_REG (0x0024)

Continued from the previous page...

MTLRFFLS Represents the status of the fill-level of the RX FIFO.

0: RX FIFO Empty

1: RX FIFO fill level is below the flow-control deactivate threshold

2: RX FIFO fill level is above the flow-control activate threshold

3: RX FIFO Full

(RO)

MTLRFRCS Represents the state of the RX FIFO read Controller.

0: IDLE state

1: Reading frame data

2: Reserved

3: Flushing the frame data and status

(RO)

MTLRFWCAS Represents the state of the MTL RX FIFO Write Controller.

0: Inactive

1: Active and is transferring a received frame to the FIFO.

(RO)

MACRFFCS Represents the state of the small FIFO Read (bit 1) and Write (bit 0) controllers of the MAC Receive Frame Controller Module.

0: Inactive

1: Active

(RO)

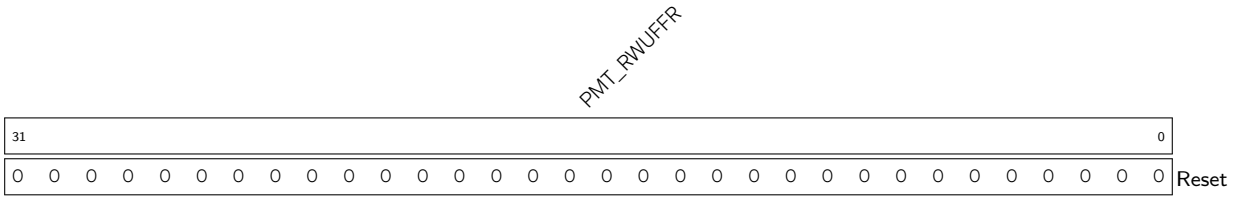
MACRPES Represents the state of the MAC GMII or MII receive protocol engine.

0: IDLE

1: Actively receiving data

(RO)

Register 55.23. PMT_RWUFFR_REG (0x0028)



WKUPPKTFILTER When RWKPTR is 0 ~ 3, configures the masking of filter 0 ~ 3 wakeup frame.

Bit[30:0] corresponds to offset+j (j = 0 ~ 30).

Bit[31] has to be 0. 0: Unmask

1: Mask

When RWKPTR is 4, bit[3]/bit[11]/bit[19]/bit[27] configure the target address type, and bit[0]/bit[8]/bit[16]/bit[24] configure whether to enable filter 0 ~ 3.

0: Unicast/Disable

1: Multicast/Enable

When RWKPTR is 5, configures the offset of the filter 0 ~ 3 masked byte.

Bit[7:0] configures the offset of filter 0 masked byte, and [31:24] configures the offset of filter 3 masked byte.

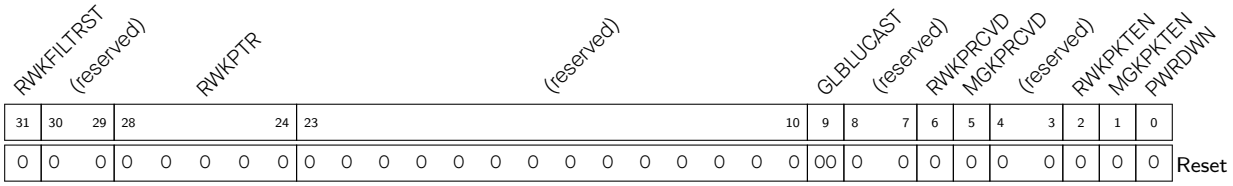
When RWKPTR is 6 ~ 7, configures the 16-bit CRC value of filter 0 ~ 3 masked byte starting from offset based on the formula:

$G(x) = x^{16} + x^{15} + x^2 + 1$

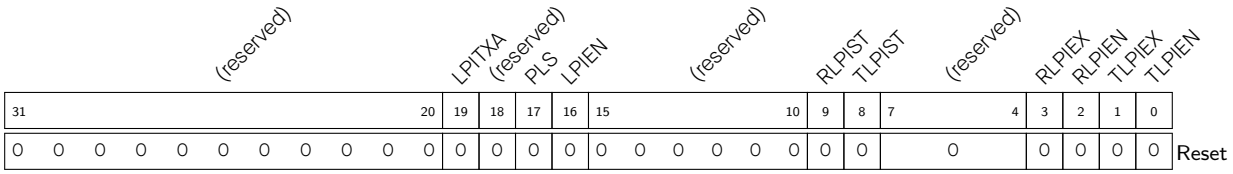
Bit[7:0] configures the CRC value of filter 0 or filter 2 masked byte when RWKPTR is 6 or 7 respectively, and bit[31:16] configures the CRC value of filter 1 or filter 3 masked byte when RWKPTR is 6 or 7 respectively.

(R/W)

Register 55.24. PMT_CSR_REG (0x002C)


RWKFILTRST
(reserved)
RWKPTR
(reserved)
GLBLUCAST
(reserved)
RWKPRCVD
MGKPRCVD
(reserved)
RWKPKTEN
MGKPKTEN
PWRDWN
31 30 29 28 24 23 10 9 8 7 6 5 4 3 2 1 0
0 0 0

Register 55.25. EMACLPI_CSR_REG (0x0030)



LPITXA Configures the behavior of the MAC when it is entering or coming out of the LPI mode on the transmit side.

0: This bit directly controls behavior of the MAC when it is entering or coming out of the LPI mode.

1: This bit controls behavior of the MAC when it is entering or coming out of the LPI mode in combination with LPIEN. When LPIEN is also 1, the MAC enters the LPI mode only after all outstanding frames (in the core) and pending frames (in the application interface) have been transmitted. The MAC comes out of the LPI mode when the application sends any frame for transmission

(R/W)

PLS Configures the link status of the PHY.

0: Link up

1: Link down

(R/W)

LPIEN Configures whether the EMAC transmitter enters the LPI state.

0: Exit

1: Enter

This bit is cleared when the LPITXA bit is set and the MAC exits the LPI state because of the arrival of a new packet for transmission.

(R/W/SC)

RLPIST Represents the receive LPI state.

0: Inactive

1: The EMAC is receiving the LPI pattern

(RO)

TLPIST Represents the transmit LPI state.

0: Inactive

1: The EMAC is transmitting the LPI pattern

(RO)

RLPIEX Represents the receive LPI exit state.

0: Inactive

1: The EMAC Receiver has stopped receiving the LPI pattern, exited the LPI state, and resumed the normal reception

(R/SS/RC)

Continued on the next page...

Register 55.25. EMACLPI_CSR_REG (0x0030)

Continued from the previous page...

RLPIEN Represents the receive LPI entry state.

0: Inactive

1: The EMAC Receiver has received an LPI pattern and entered the LPI state

(R/SS/RC)

TLPIEX Represents the transmit LPI exit state.

0: Inactive

1: The EMAC transmitter has exited the LPI state after the user has cleared the LPIEN bit and the LPI_TW_TIMER has expired

(R/SS/RC)

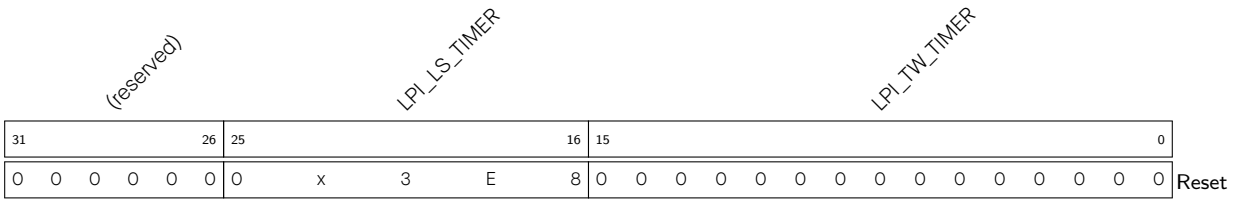
TLPIEN Represents the transmit LPI entry state.

0: Inactive

1: The EMAC Transmitter has entered the LPI state because of the setting of the LPIEN bit

(R/SS/RC)

Register 55.26. EMACLPITIMERSCONTROL_REG (0x0034)



LPI_LS_TIMER Configures the minimum time for which the link status from the PHY should be up.

Measurement unit: millisecond.

The default value is 1000 (1 second) as defined in the IEEE standard.

(R/W)

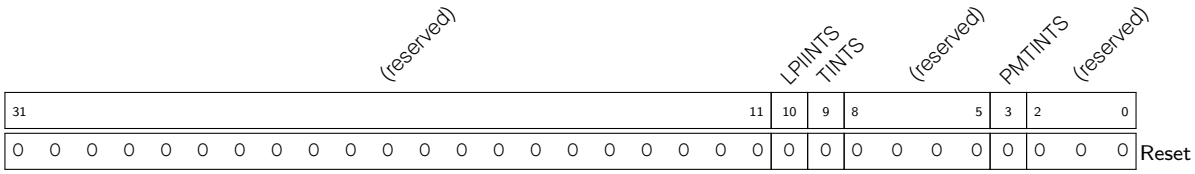
LPI_TW_TIMER Configures the minimum time for which the MAC waits after it stops transmitting the LPI pattern to the PHY and before it resumes the normal transmission.

Measurement unit: millisecond.

The TLPIEX status bit is set after the expiry of this timer.

(R/W)

Register 55.27. EMACINTS_REG (0x0038)

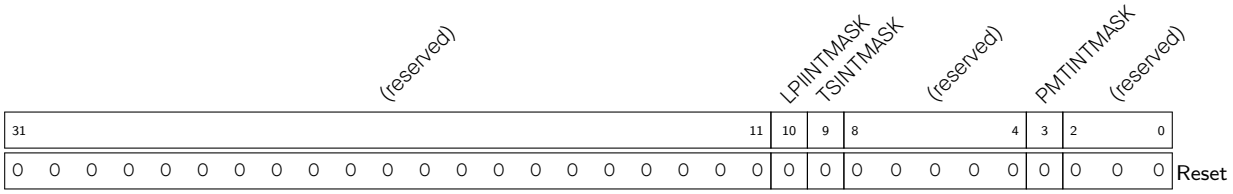


LPIINTS The raw interrupt status of EMAC_LPI_INT. (RO)

TINTS The raw interrupt status of TS_TRI_INT. (R/SS/RC)

PMTINTS The raw interrupt status of EMAC_PMP_INT. (RO)

Register 55.28. EMACINTMASK_REG (0x003C)

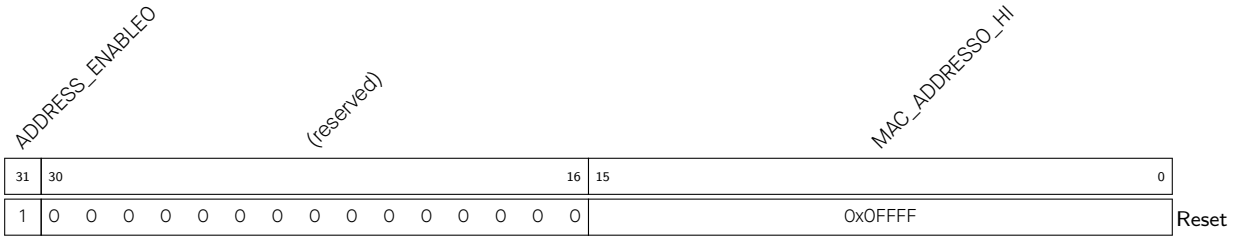


LPIINTMASK Write 1 to mask EMAC_LPI_INT.(R/W)

TSINTMASK Write 1 to mask TS_TRI_INT. (R/W)

PMTINTMASK Write 1 to mask EMAC_PMP_INT. (R/W)

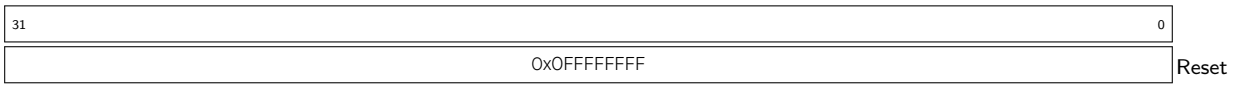
Register 55.29. EMACADDROHIGH_REG (0x0040)



ADDRESS_ENABLE0 This bit is always set to 1. (RO)

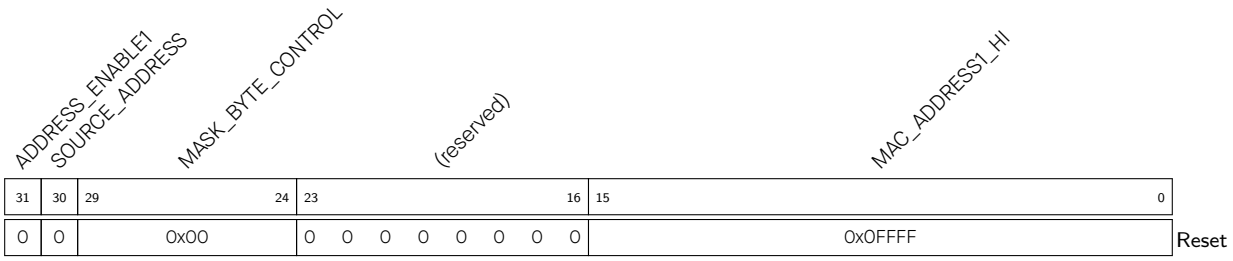
MAC_ADDRESS0_HI Configures the upper 16 bits of the first 6-byte MAC address [47:32]. The MAC uses this field for filtering the received frames and inserting the MAC address in the Transmit Flow Control (PAUSE) Frames. (R/W)

Register 55.30. EMACADDR0LOW_REG (0x0044)



EMACADDR0LOW_REG Configures the lower 32 bits of the first 6-byte MAC address. The MAC uses this field for filtering the received frames and inserting the MAC address in the Transmit Flow Control (PAUSE) Frames. (R/W)

Register 55.31. EMACADDR1HIGH_REG (0x0048)



ADDRESS_ENABLE1 Configures whether to enable perfect filtering using the second MAC address.

0: Disable

1: Enable

(R/W)

SOURCE_ADDRESS Configures whether to which fields of the received frame EMACADDR1 [47:0] compares.

0: The DA fields

1: The SA fields

(R/W)

MASK_BYTE_CONTROL Configures whether to mask MAC address byte bytes when comparing the received DA or SA with the contents of EMACADDR1.

0: Unmask

1: Mask

The correspondence between the mask control bits and the bytes:

Bit[29]: EMACADDR1HIGH_REG[15:8]

Bit[28]: EMACADDR1HIGH_REG[7:0]

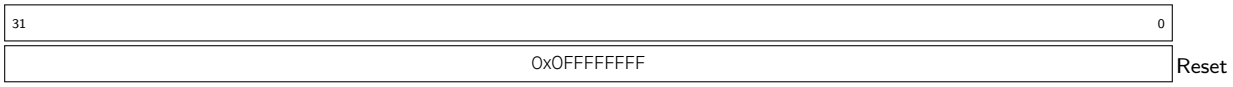
Bit[27]: EMACADDR1LOW_REG[31:24]

Bit[24]: EMACADDR1LOW_REG[7:0]

You can filter a group of addresses (known as group address filtering) by masking one or more bytes of the address.(R/W)

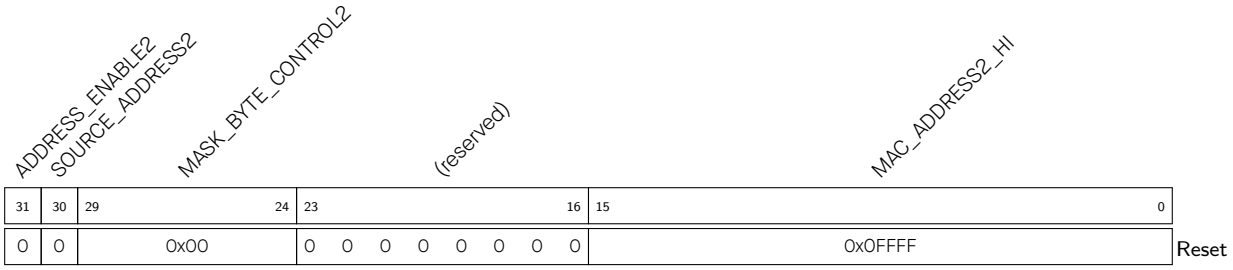
MAC_ADDRESS1_HI Configures the upper 16 bits of the second 6-byte MAC address [47:32]. (R/W)

Register 55.32. EMACADDR1LOW_REG (0x004C)



EMACADDR1LOW_REG Configures the lower 32 bits of the second 6-byte MAC address. The content of this field is undefined until loaded by the Application after the initialization process. (R/W)

Register 55.33. EMACADDR2HIGH_REG (0x0050)



ADDRESS_ENABLE2 Configures whether to enable perfect filtering using the third MAC address.

0: Disable

1: Enable

(R/W)

SOURCE_ADDRESS2 Configures whether to which fields of the received frame EMACADDR2 [47:0] compares.

0: The DA fields

1: The SA fields

(R/W)

MASK_BYTE_CONTROL2 Configures whether to mask MAC address byte bytes when comparing the received DA or SA with the contents of EMACADDR2.

0: Unmask

1: Mask

The correspondence between the mask control bits and the bytes:

Bit[29]: EMACADDR2HIGH_REG[15:8]

Bit[28]: EMACADDR2HIGH_REG[7:0]

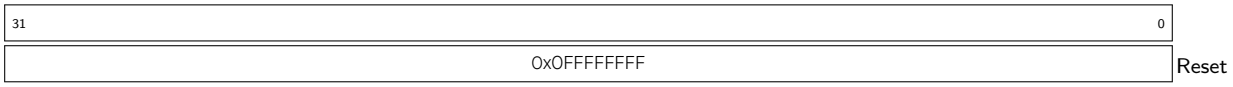
Bit[27]: EMACADDR2LOW_REG[31:24]

Bit[24]: EMACADDR2LOW_REG[7:0]

You can filter a group of addresses (known as group address filtering) by masking one or more bytes of the address.(R/W)

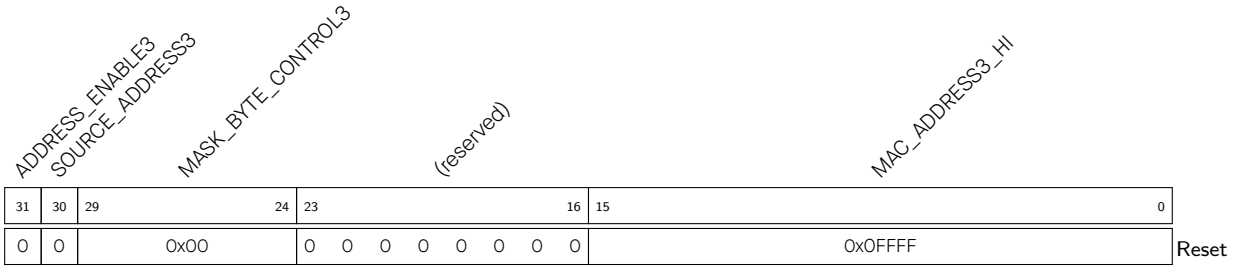
MAC_ADDRESS2_HI Configures the upper 16 bits of the third 6-byte MAC address [47:32]. (R/W)

Register 55.34. EMACADDR2LOW_REG (0x0054)



EMACADDR2LOW_REG Configures the lower 32 bits of the third 6-byte MAC address.
The content of this field is undefined until loaded by the Application after the initialization process.(R/W)

Register 55.35. EMACADDR3HIGH_REG (0x0058)



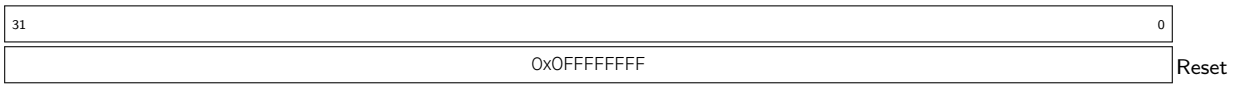
ADDRESS_ENABLE3 Configures whether to enable perfect filtering using the fourth MAC address.
0: Disable
1: Enable
(R/W)

SOURCE_ADDRESS3 Configures whether to which fields of the received frame EMACADDR3 [47:0] compares.
0: The DA fields
1: The SA fields
(R/W)

MASK_BYTE_CONTROL3 Configures whether to mask MAC address byte bytes when comparing the received DA or SA with the contents of EMACADDR3.
0: Unmask
1: Mask
The correspondence between the mask control bits and the bytes:
Bit[29]: EMACADDR3HIGH_REG[15:8]
Bit[28]: EMACADDR3HIGH_REG[7:0]
Bit[27]: EMACADDR3LOW_REG[31:24]
Bit[24]: EMACADDR3LOW_REG[7:0]
You can filter a group of addresses (known as group address filtering) by masking one or more bytes of the address. (R/W)

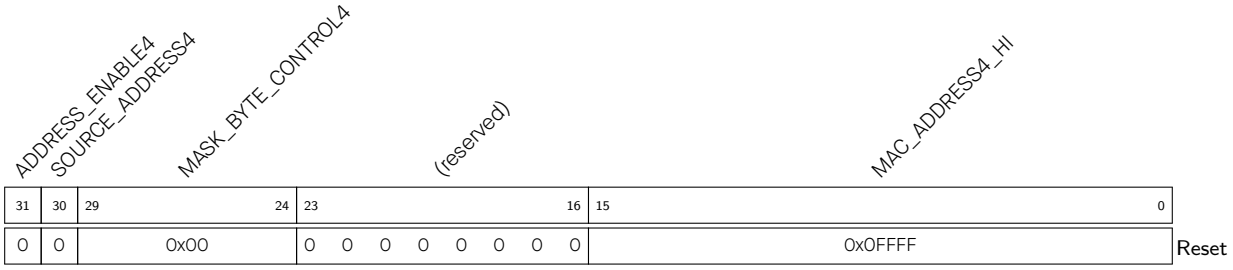
MAC_ADDRESS3_HI Configures the upper 16 bits of the fourth 6-byte MAC address [47:32]. (R/W)

Register 55.36. EMACADDR3LOW_REG (0x005C)



EMACADDR3LOW_REG Configures the lower 32 bits of the fourth 6-byte MAC address. The content of this field is undefined until loaded by the Application after the initialization process. (R/W)

Register 55.37. EMACADDR4HIGH_REG (0x0060)



ADDRESS_ENABLE4 Configures whether to enable perfect filtering using the fifth MAC address.

0: Disable

1: Enable

(R/W)

SOURCE_ADDRESS4 Configures whether to which fields of the received frame EMACADDR4 [47:0] compares.

0: The DA fields

1: The SA fields

(R/W)

MASK_BYTE_CONTROL4 Configures whether to mask MAC address byte bytes when comparing the received DA or SA with the contents of EMACADDR4.

0: Unmask

1: Mask

The correspondence between the mask control bits and the bytes:

Bit[29]: EMACADDR4HIGH_REG[15:8]

Bit[28]: EMACADDR4HIGH_REG[7:0]

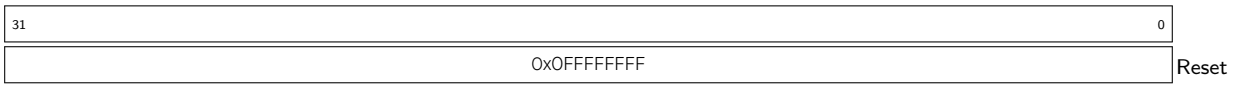
Bit[27]: EMACADDR4LOW_REG[31:24]

Bit[24]: EMACADDR4LOW_REG[7:0]

You can filter a group of addresses (known as group address filtering) by masking one or more bytes of the address.(R/W)

MAC_ADDRESS4_HI Configures the upper 16 bits of the fifth 6-byte MAC address [47:32]. (R/W)

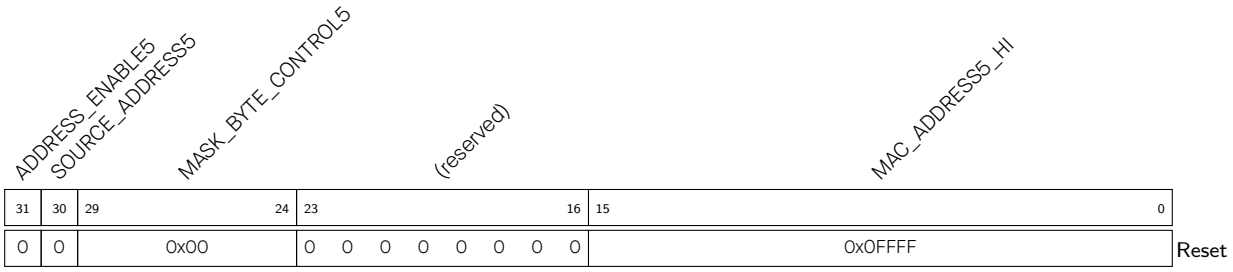
Register 55.38. EMACADDR4LOW_REG (0x0064)



EMACADDR4LOW_REG Configures the lower 32 bits of the fifth 6-byte MAC address.

The content of this field is undefined until loaded by the Application after the initialization process. (R/W)

Register 55.39. EMACADDR5HIGH_REG (0x0068)



ADDRESS_ENABLE5 Configures whether to enable perfect filtering using the sixth MAC address.

0: Disable

1: Enable

(R/W)

SOURCE_ADDRESS5 Configures whether to which fields of the received frame EMACADDR5 [47:0] compares.

0: The DA fields

1: The SA fields

(R/W)

MASK_BYTE_CONTROL5 Configures whether to mask MAC address byte bytes when comparing the received DA or SA with the contents of EMACADDR5.

0: Unmask

1: Mask

The correspondence between the mask control bits and the bytes:

Bit[29]: EMACADDR5HIGH_REG[15:8]

Bit[28]: EMACADDR5HIGH_REG[7:0]

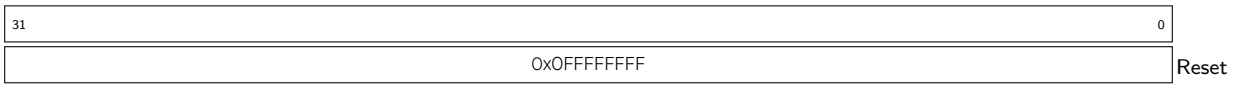
Bit[27]: EMACADDR5LOW_REG[31:24]

Bit[24]: EMACADDR5LOW_REG[7:0]

You can filter a group of addresses (known as group address filtering) by masking one or more bytes of the address.(R/W)

MAC_ADDRESS5_HI Configures the upper 16 bits of the sixth 6-byte MAC address [47:32]. (R/W)

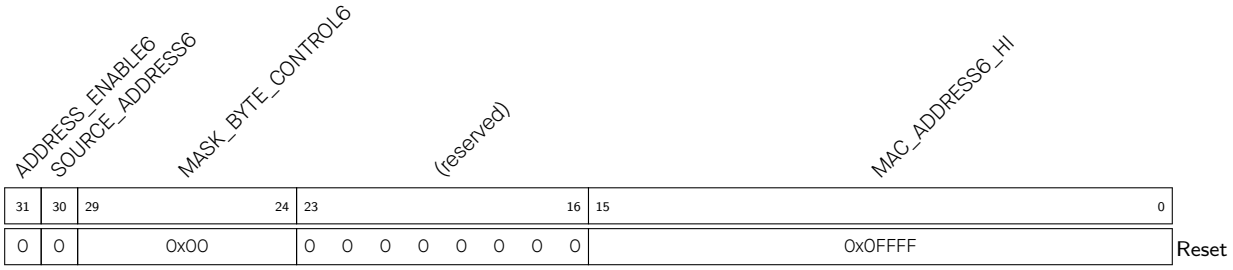
Register 55.40. EMACADDR5LOW_REG (0x006C)



EMACADDR5LOW_REG Configures the lower 32 bits of the sixth 6-byte MAC address.

The content of this field is undefined until loaded by the Application after the initialization process. (R/W)

Register 55.41. EMACADDR6HIGH_REG (0x0070)



ADDRESS_ENABLE6 Configures whether to enable perfect filtering using the seventh MAC address.

0: Disable

1: Enable

(R/W)

SOURCE_ADDRESS6 Configures whether to which fields of the received frame EMACADDR 6[47:0] compares.

0: The DA fields

1: The SA fields

(R/W)

MASK_BYTE_CONTROL6 Configures whether to mask MAC address byte bytes when comparing the received DA or SA with the contents of EMACADDR6.

0: Unmask

1: Mask

The correspondence between the mask control bits and the bytes:

Bit[29]: EMACADDR6HIGH_REG[15:8]

Bit[28]: EMACADDR6HIGH_REG[7:0]

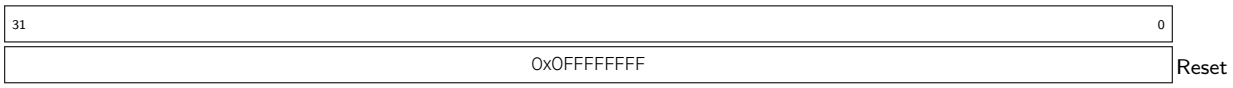
Bit[27]: EMACADDR6LOW_REG[31:24]

Bit[24]: EMACADDR6LOW_REG[7:0]

You can filter a group of addresses (known as group address filtering) by masking one or more bytes of the address. (R/W)

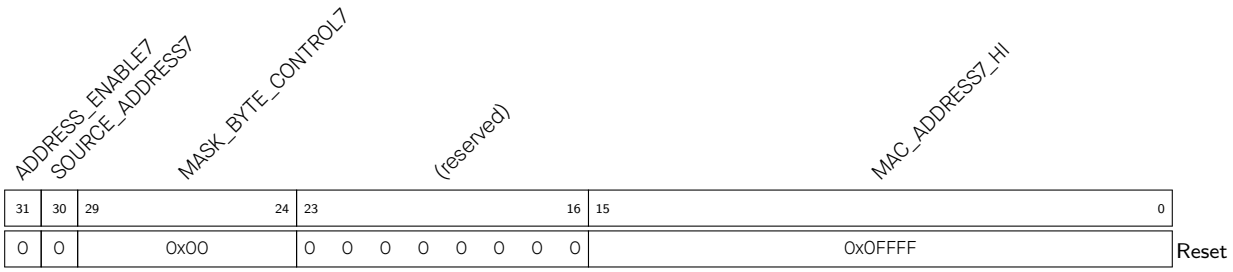
MAC_ADDRESS6_HI Configures the upper 16 bits of the seventh 6-byte MAC address [47:32]. (R/W)

Register 55.42. EMACADDR6LOW_REG (0x0074)



EMACADDR6LOW_REG Configures the lower 32 bits of the seventh 6-byte MAC address. The content of this field is undefined until loaded by the Application after the initialization process.(R/W)

Register 55.43. EMACADDR7HIGH_REG (0x0078)



ADDRESS_ENABLE7 Configures whether to enable perfect filtering using the eighth MAC address.

0: Disable

1: Enable

(R/W)

SOURCE_ADDRESS7 Configures whether to which fields of the received frame EMACADDR7 [47:0] compares.

0: The DA fields

1: The SA fields

(R/W)

MASK_BYTE_CONTROL7 Configures whether to mask MAC address byte bytes when comparing the received DA or SA with the contents of EMACADDR7.

0: Unmask

1: Mask

The correspondence between the mask control bits and the bytes:

Bit[29]: EMACADDR7HIGH_REG[15:8]

Bit[28]: EMACADDR7HIGH_REG[7:0]

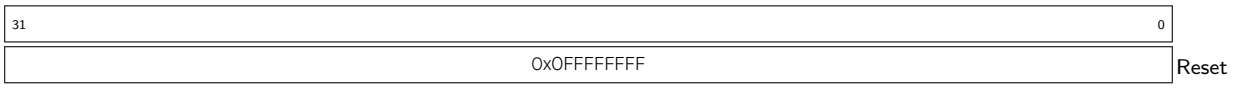
Bit[27]: EMACADDR7LOW_REG[31:24]

Bit[24]: EMACADDR7LOW_REG[7:0]

You can filter a group of addresses (known as group address filtering) by masking one or more bytes of the address. (R/W)

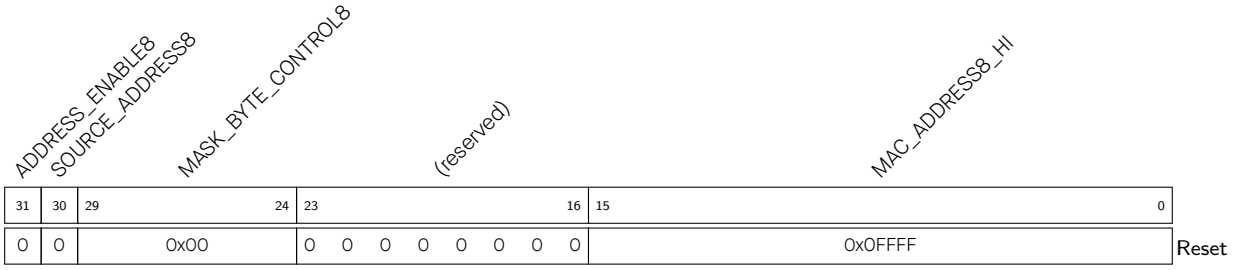
MAC_ADDRESS7_HI Configures the upper 16 bits of the eighth 6-byte MAC address [47:32]. (R/W)

Register 55.44. EMACADDR7LOW_REG (0x007C)



EMACADDR7LOW_REG Configures the lower 32 bits of the eighth 6-byte MAC address. The content of this field is undefined until loaded by the Application after the initialization process.(R/W)

Register 55.45. EMACADDR8HIGH_REG (0x0080)



ADDRESS_ENABLE8 Configures whether to enable perfect filtering using the ninth MAC address.

0: Disable

1: Enable

(R/W)

SOURCE_ADDRESS8 Configures whether to which fields of the received frame EMACADDR8 [47:0] compares.

0: The DA fields

1: The SA fields

(R/W)

MASK_BYTE_CONTROL8 Configures whether to mask MAC address byte bytes when comparing the received DA or SA with the contents of EMACADDR8.

0: Unmask

1: Mask

The correspondence between the mask control bits and the bytes:

Bit[29]: EMACADDRH8IGH_REG[15:8]

Bit[28]: EMACADDR8HIGH_REG[7:0]

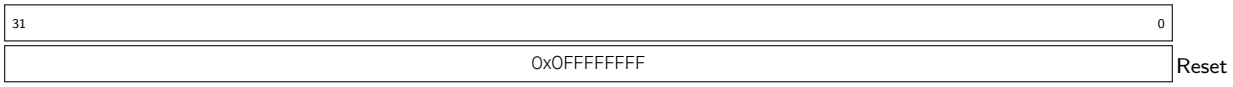
Bit[27]: EMACADDR8LOW_REG[31:24]

Bit[24]: EMACADDR8LOW_REG[7:0]

You can filter a group of addresses (known as group address filtering) by masking one or more bytes of the address. (R/W)

MAC_ADDRESS8_HI Configures the upper 16 bits of the ninth 6-byte MAC address [47:32]. (R/W)

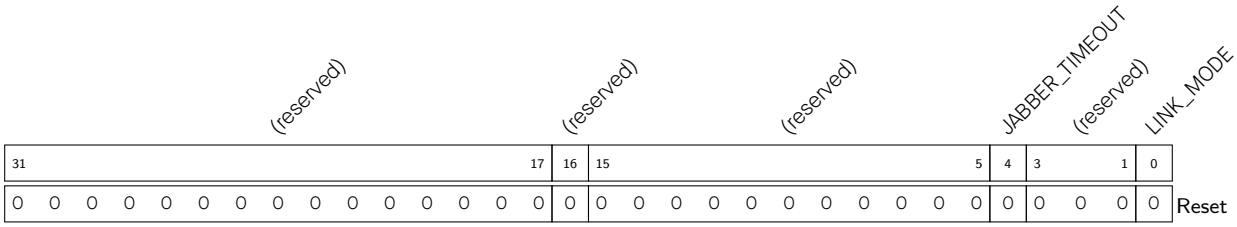
Register 55.46. EMACADDR8LOW_REG (0x0084)



EMACADDR8LOW_REG Configures the lower 32 bits of the ninth 6-byte MAC address.

The content of this field is undefined until loaded by the Application after the initialization process. (R/W)

Register 55.47. EMACCSTATUS_REG (0x00D8)



JABBER_TIMEOUT Represents whether a receive jabber error occurs.

0: Not occur

1: Occur

(RO)

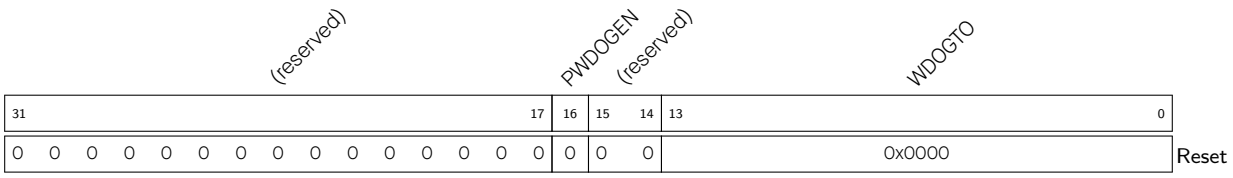
LINK_MODE Represents the current mode of operation of the link.

0: Half-duplex

1: Full-duplex

(RO)

Register 55.48. EMACWDOGTO_REG (0x00DC)



PWDOGEN Configures whether the watchdog timeout is programmble.

0: Not programmable. The watchdog timeout for a received frame is controlled by the setting of EMACCONFIG_REG[23] (WD) and EMACCONFIG_REG[20] (JE).

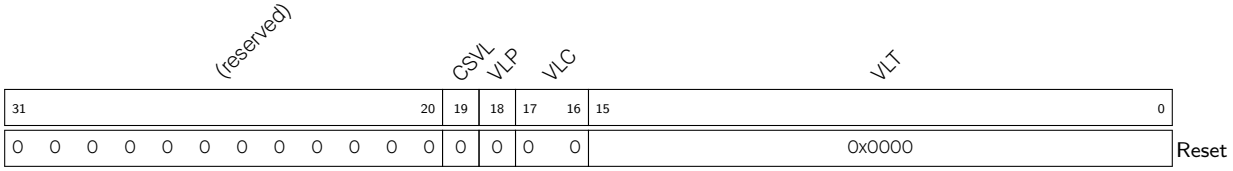
1: Programmable via the WDOGTO field if EMACCONFIG_REG[23] (WD) is 0.

(R/W)

WDOGTO Configures the watchdog timeout for a received frame when PWDOGEN is 1 and EMACCONFIG_REG[23] (WD) is 0.

If the length of a received frame exceeds the value of this field, such frame is terminated and declared as an error frame. (R/W)

Register 55.49. EMACTXVLTCTRL_REG (0x0584)



CSVL Configures which type is inserted or replaced in the 13th and 14th bytes of transmitted frames.

0: C-VLAN type (0x8100)

1: S-VLAN type (0x88A8)

(R/W)

VLP Configures what controls VLAN deletion, insertion, or replacement.

0: The control input signal

1: VLC

(R/W)

VLC Configures the VLAN tag in transmit frames.

0: No VLAN tag deletion, insertion, or replacement.

1: VLAN tag deletion. The MAC removes the VLAN type (bytes 13 and 14) and VLAN tag (bytes 15 and 16) of all transmitted frames with VLAN tags.

2: VLAN tag insertion. The MAC inserts VLT in bytes 15 and 16 of the frame after inserting the Type value (0x8100/0x88a8) in bytes 13 and 14. This operation is performed on all transmitted frames, irrespective of whether they already have a VLAN tag.

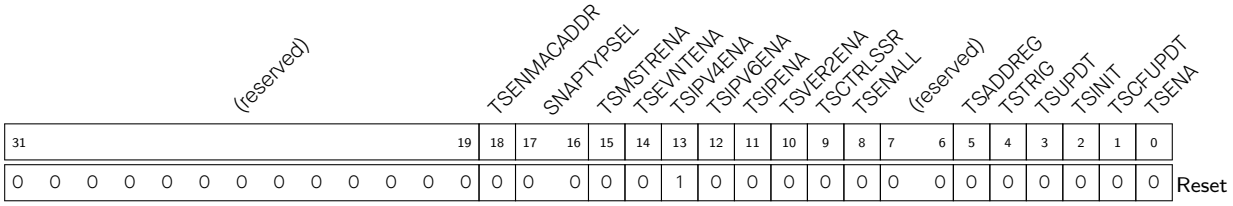
3: VLAN tag replacement. The MAC replaces VLT in bytes 15 and 16 of all VLAN-type transmitted frames (Bytes 13 and 14 are 0x8100/0x88a8).

(R/W)

VLT Configures the value of the VLAN tag to be inserted or replaced.

The value must only be changed when the transmit lines are inactive or during the initialization phase. (R/W)

Register 55.50. EMACTSTPCTRL_REG (0x0700)



TSENMACADDR Configures whether to enable the DA MAC address (that matches any MAC Address register) for PTP frame filtering when PTP is directly sent over Ethernet.

0: Disable

1: Enable

(R/W)

SNAPTYPSEL Configures the set of PTP packet types for which snapshot needs to be taken, in combination with TSMSTRENA and TSEVNTENA.

SNAPTYPSEL, TSMSTRENA, TSEVNTENA:

00x0: SYNC, Follow_Up, Delay_Req, Delay_Resp

0001: SYNC

0011: Delay_Req

01x0: SYNC, Follow_Up, Delay_Req, Delay_Resp, Pdelay_Req, Pdelay_Resp, Pdelay_Resp_Follow_Up

0101: SYNC, Pdelay_Req, Pdelay_Resp

0111: Delay_Req, Pdelay_Req, Pdelay_Resp

10xx: SYNC, Delay_Req

11xx: Pdelay_Req, Pdelay_Resp

(R/W)

TSMSTRENA Configures whether to enable snapshot only for messages relevant to the master node.

0: Disable. The snapshot is taken for the messages relevant to the slave node

1: Enable

(R/W)

TSEVNTENA Configures whether to enable timestamp snapshot only for event messages (SYNC, Delay_Req, Pdelay_Req, Pdelay_Resp).

0: Disable. The snapshot is taken for all messages except Announce, Management, and Signaling

1: Enable

(R/W)

TSIPV4ENA Configures whether to enable the processing of PTP frames sent over IPv4-UDP.

0: Disable. The MAC ignores the PTP transported over UDP-IPv4 packets

1: Enable. The MAC receiver processes the PTP packets encapsulated in UDP over IPv4 packets

(R/W)

Continued on the next page...

Register 55.50. EMACTSTPCTRL_REG (0x0700)

Continued from the previous page...

TSIPV6ENA Configures whether to enable the processing of PTP frames sent over IPv6-UDP.
0: Disable. The MAC ignores the PTP transported over UDP-IPv6 packets
1: Enable. The MAC receiver processes the PTP packets encapsulated in UDP over IPv6 packets
(R/W)

TSIPENA Configures whether to enable the processing of PTP sent over Ethernet frames.
0: Disable. the MAC ignores the PTP over Ethernet packets
1: Enable. The MAC receiver processes the PTP packets encapsulated directly in the Ethernet frames
(R/W)

TSVER2ENA Configures whether to enable PTP packet processing for version 2 format.
0: Disable. The PTP packets are processed using the version 1 format
1: Enable. The PTP packets are processed using the 1588 version 2 format
(R/W)

TSCTRLSSR Configures the timeout digital or binary rollover.
0: The Timestamp Low register rolls over after the value of the sub-second register 0x7FFF_FFFF. The sub-second increment has to be programmed correctly depending on the PTP reference clock frequency and the value of this bit.
1: The Timestamp Low register rolls over after 0x3B9A_C9FF value (that is, 1 nanosecond accuracy) and increments the timestamp (High) seconds.
(R/W)

TSENALL Configures whether to enable timestamp for all frames received by the MAC.
0: Disable
1: Enable
(R/W)

TSADDREG Configures whether to update the content of the EMACTSTPADDEND_REG register in the PTP block for fine correction.
0: Not update
1: Update
This bit should be zero before setting it. (R/WS/SC)

Continued on the next page...

Register 55.50. EMACTSTPCTRL_REG (0x0700)

Continued from the previous page...

TSTRIG Configures whether to enable the timestamp interrupt trigger.

0: Disable

1: Enable. The timestamp interrupt is generated when the System Time becomes greater than the value written in the Target Time register

(R/WS/SC)

TSUPDT Configures whether to update (add or subtract) the system time with the value specified in EMAC2NDUPT_REG and EMACNSUPT_REG.

0: Not update

1: Update

This bit should be read as zero before updating it.

EMACSYSTIMHIGHWORD2ND_REG is not updated. (R/WS/SC)

TSINIT Configures whether to initialize (overwrite) the system time with the value specified in EMAC2NDUPT_REG and EMACNSUPT_REG.

0: Not initialize

1: Initialize

This bit should be read as zero before updating it.

EMACSYSTIMHIGHWORD2ND_REG will be initialized. (R/WS/SC)

TSCFUPDT Configures the method used to update the system time.

0: Coarse method

1: Find update method

(R/W)

TSENA Configures whether to add the timestamp for the transmit and receive frames.

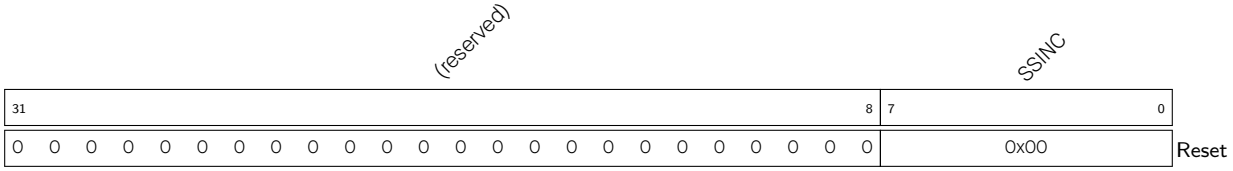
0: Not add the timestamp, and receive frames and the Timestamp Generator is also suspended

1: Add the timestamp

You need to initialize the Timestamp (system time) after setting this bit to 1.

On the receive side, the MAC processes the 1588 frames only if this bit is set. (R/W)

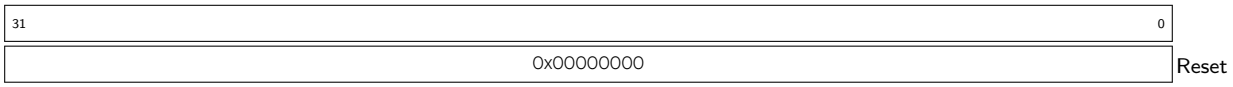
Register 55.51. EMACSUB2NDINCR_REG (0x0704)



SSINC Configures the value accumulated every clock cycle (PTP clock) with the contents of the sub-second register.

For example, when PTP clock is 50 MHz (the period is 20 ns), you should program 20 (0x14) when the System Time-Nanoseconds register has an accuracy of 1 ns (EMACTSTPCTRL_REG[9], i.e., the TSCTRLSSR bit is set). When TSCTRLSSR is cleared, the Nanoseconds register has a resolution of 0.465 ns. In this case, you should program a value of 43 (0x2B) that is derived by 20 ns/0.465. (R/W)

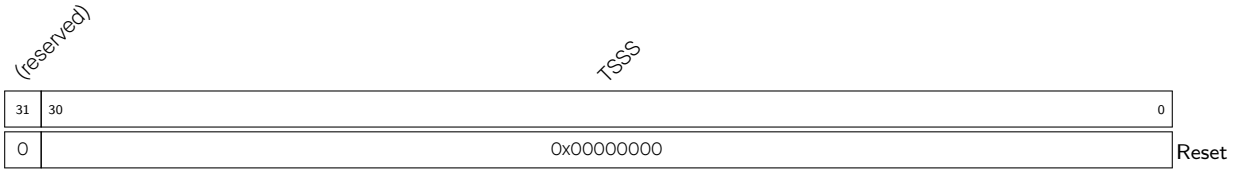
Register 55.52. EMACSYSTIM2ND_REG (0x0708)



TSS Represents the current value of the System Time maintained by the MAC.

Measurement unit: second. (RO)

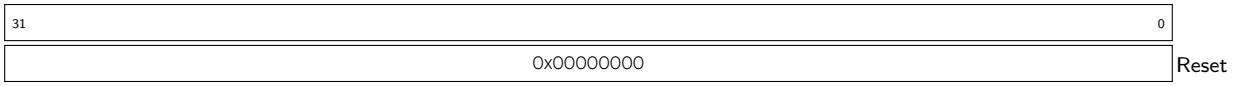
Register 55.53. EMACSYSTIMNS_REG (0x070C)



TSSS Represents the sub-second representation of time, with an accuracy of 0.46 ns.

When EMACTSTPCTRL_REG[9] (TSCTRLSSR) is set, each bit represents 1 ns and the maximum value is 0x3B9A_C9FF. (RO)

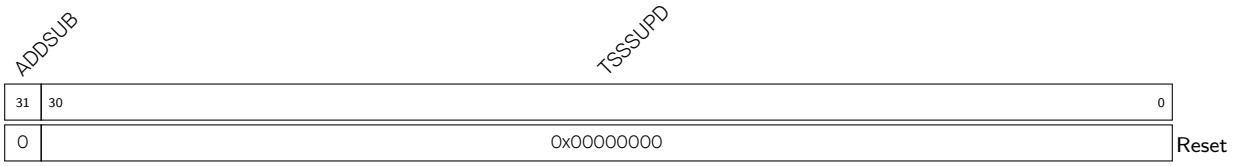
Register 55.54. EMAC2NDUPT_REG (0x0710)



TSSSUPD Configures the time to be initialized or added to the system time.

Measurement unit: second. (R/W)

Register 55.55. EMACNSUPT_REG (0x0714)



ADDSUB with Configures whether to subtract or add the system time value the contents of the update register.

0: Add

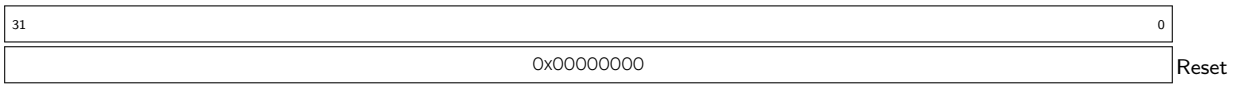
1: Subtract

(R/W)

TSSSUPD Configures the sub-second representation of time, with an accuracy of 0.46 ns.

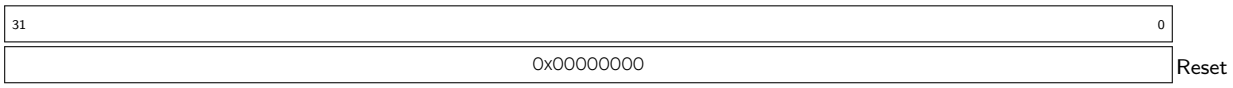
When EMACTSTPCTRL_REG[9] (TSCTRLSSR) is set, each bit represents 1 ns and the programmed value should not exceed 0x3B9A_C9FF. (R/W)

Register 55.56. EMACTSTPADDEND_REG (0x0718)



TSAR Configures the 32-bit time value to be added to the Accumulator register to achieve time synchronization. (R/W)

Register 55.57. EMACTGTIME2ND_REG (0x071C)

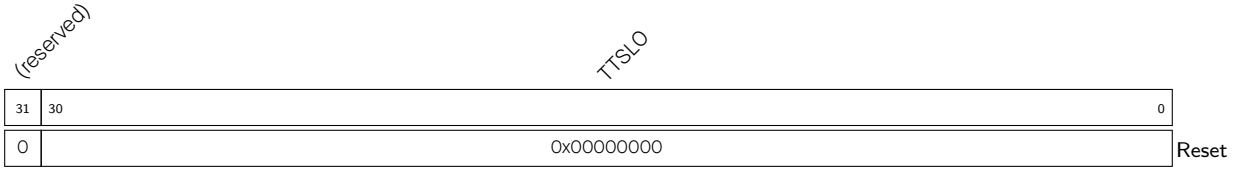


TSTR Configures the target time.

Measurement unit: second.

When the system time matches or exceeds the target time, an interrupt is generated (if enabled). (R/W)

Register 55.58. EMACTGTIMENS_REG (0x0720)

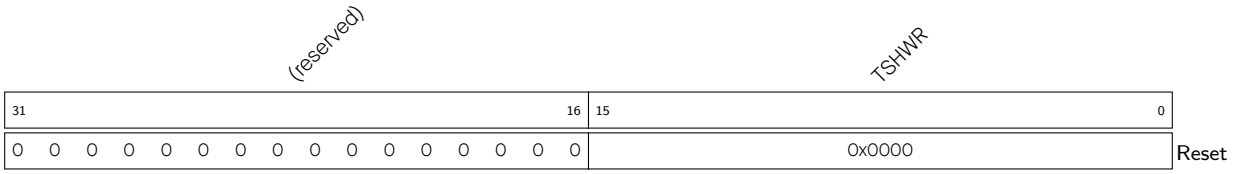


TTSLO Configures the signed target time.

Measurement unit: nanosecond.

When the system time matches or exceeds the target time, an interrupt is generated (if enabled). (R/W)

Register 55.59. EMACSYSTIMHIGHWORD2ND_REG (0x0724)

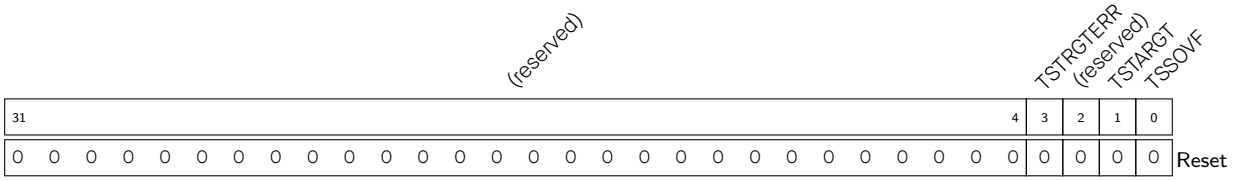


TSHWR Configures the most significant 16-bits of the timestamp seconds value.

The register is directly written to initialize the value.

This register is incremented when there is an overflow from the 32 bits of the System Time Seconds register. (R/W/SU)

Register 55.60. EMACTSTPSTATUS_REG (0x0728)



TSTRGTERR Represents whether the target time, being programmed in EMACTGTIME2ND_REG and EMACTGTIMENS_REG, is already elapsed.

0: Not elapsed

1: Elapsed

(R/SS/RC)

TSTARGT Represents whether the system time is greater or equal to the value specified in EMACTGTIME2ND_REG and EMACTGTIMENS_REG.

0: Less than the value

1: Greater or equal to the value

(R/SS/RC)

TSSOVF Represents whether the second value of the timestamp has overflowed beyond 0xFFFF_FFFF.

0: No overflow

1: Overflow

(R/SS/RC)

Chapter 56

Two-Wire Automotive Interface (TWAI)

The Two-wire Automotive Interface (TWAI®) is a multi-master, multi-cast communication protocol with functions such as error detection and signaling and inbuilt message priorities and arbitration. The TWAI protocol is suited for automotive and industrial applications (see Section 56.2 for more details).

ESP32-P4 contains three TWAI controllers, TWAI 0, TWAI 1, and TWAI 2. Each controller can individually be connected to a TWAI bus via an external transceiver. The TWAI controllers have many advanced features, and can be utilized in a wide range of use cases, such as automotive products, industrial automation controls, building automation, etc.

56.1 Features

Each TWAI controller on ESP32-P4 supports the following features:

- Compatibility with ISO 11898-1 protocol (CAN Specification 2.0)
- Standard Frame Format (11-bit ID) and Extended Frame Format (29-bit ID)
- Bit rates from 1 Kbit/s to 1 Mbit/s
- Multiple modes of operation:
  - Normal
  - Listen-only (no influence on bus)
  - Self-test (no acknowledgment required during data transmission)
- 64-byte Receive FIFO
- Special transmissions:
  - Single-shot transmissions (does not automatically re-transmit upon error)
  - Self-reception (the TWAI controller transmits and receives messages simultaneously)
- Acceptance Filter (supports Single and Dual-filter modes)
- Error detection and handling:
  - Error counters
  - Configurable error warning limit
  - Error code capture
  - Arbitration lost capture
  - Automatic transceiver standby

56.2 Protocol Overview

56.2.1 TWAI Properties

The TWAI protocol connects two or more nodes in a bus network, and allows nodes to exchange messages with deterministic delay. A TWAI bus has the following properties:

Single Channel and Non-Return-to-Zero: The bus has only one transmission line for single-channel communication, which is half-duplex. Synchronization is also implemented in this channel, so extra channels (e.g., clock or enable) are not required. The bit stream of a TWAI message is encoded using the Non-Return-to-Zero (NRZ) method.

Bit Values: The single channel can either be in a dominant or recessive state, representing a logical 0 and a logical 1 respectively. A node transmitting data in a dominant state always overrides the other node transmitting data in a recessive state. The physical implementation on the bus is left to the application level to decide (e.g., differential pair or a single wire).

Bit Stuffing: Certain fields of TWAI messages are bit-stuffed. A transmitter that transmits five consecutive bits of the same value (e.g., dominant value or recessive value) should automatically insert a complementary bit. Likewise, a receiver that receives five consecutive bits of the same value should treat the next bit as a stuffed bit. Bit stuffing is applied to the following fields: SOF, arbitration field, control field, data field, and CRC sequence (see Section 56.2.2 for more details).

Multi-Cast: All nodes receive the same bits as they are connected to the same bus. Data is consistent across all nodes unless there is a bus error (see Section 56.2.3 for more details).

Multi-Master: Any node can initiate a transmission. If a transmission is already ongoing, a node will wait until the current transmission is over before initiating a new transmission.

Message Priority and Arbitration: If two or more nodes simultaneously initiate a transmission, the TWAI protocol ensures that one node will win arbitration of the bus. The arbitration field of the message transmitted by each node is used to determine which node will win the arbitration.

Error Detection and Signaling: Each node actively monitors the bus for errors, and signals the detected errors by transmitting an error frame.

Fault Confinement: Each node maintains a set of error counters that are incremented/decremented according to a set of rules. When the error counters surpass a certain threshold, the node will automatically eliminate itself from the network by switching itself off.

Configurable Bit Rate: The bit rate for a single TWAI bus is configurable. However, all nodes on the same bus must operate at the same bit rate.

Transmitters and Receivers: At any point in time, a TWAI node can either be a transmitter or a receiver.

- A node generating a message is a transmitter. The node remains a transmitter until the bus is idle or until the node loses arbitration. Please note that there could be multiple nodes that act as transmitters during arbitration.
- All nodes that are not transmitters can work as receivers.

56.2.2 TWAI Messages

TWAI nodes use messages to transmit data, and signal errors to other nodes when detecting errors on the bus. Messages are split into various frame types, and different frame types have different frame formats.

The TWAI protocol has the following frame types:

- Data frame
- Remote frame
- Error frame
- Overload frame
- Interframe space

The TWAI protocol supports the following frame formats:

- Standard Frame Format (SFF) that uses a 11-bit identifier
- Extended Frame Format (EFF) that uses a 29-bit identifier

56.2.2.1 Data Frames and Remote Frames

Data frames are used by nodes to send data to other nodes, and can have a payload of 0 to 8 data bytes. Remote frames are used for nodes to request a data frame with the same identifier from other nodes, and thus they do not contain any data bytes. However, data frames and remote frames share many fields. Figure 56.2-1 illustrates the fields and sub-fields of different frames and formats.

Arbitration Field

When two or more nodes transmit data or remote frames simultaneously, the arbitration field is used to determine which node will win arbitration of the bus. In the arbitration field, if a node transmits a recessive bit while detecting a dominant bit, it indicates that another node has overridden its recessive bit. Therefore, the node transmitting the recessive bit has lost arbitration of the bus and should immediately switch to be a receiver.

The arbitration field primarily consists of a frame identifier that is transmitted from the most significant bit first. Given that a dominant bit represents a logical 0, and a recessive bit represents a logical 1:

- A frame with the smallest ID value always wins arbitration.
- Given the same ID and format, data frames always prevail over remote frames due to their RTR bits being dominant.
- Given the same first 11 bits of ID, a Standard Format Data Frame always prevails over an Extended Format Data Frame due to its SRR bits being recessive.

Control Field

The control field primarily consists of the Data Length Code (DLC) which indicates the number of payload data bytes for a data frame, or the number of requested data bytes for a remote frame. The DLC is transmitted from the most significant bit first.

Data Field

The data field contains the actual payload data bytes of a data frame. Remote frames do not contain any data field.

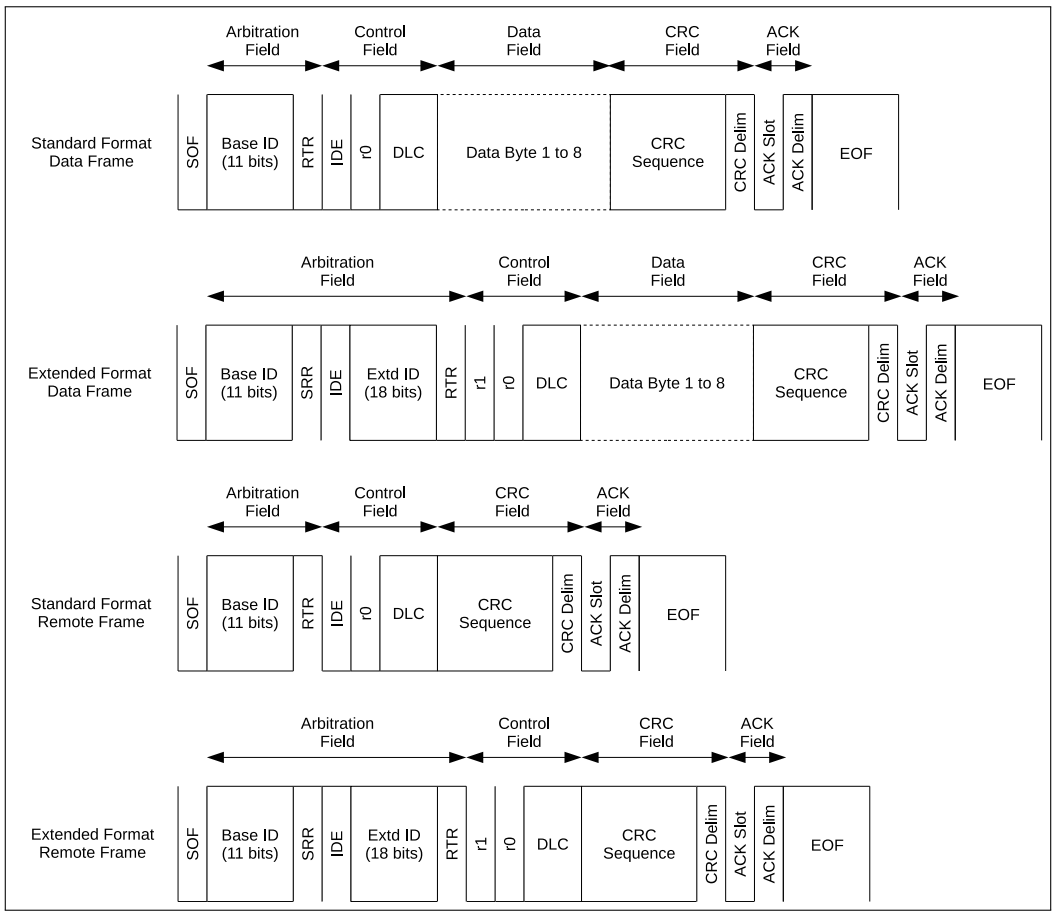


Figure 56.2-1. Bit Fields in Data Frames and Remote Frames

CRC Field

The CRC field primarily consists of a CRC sequence. The CRC sequence is a 15-bit cyclic redundancy code calculated from the de-stuffed contents (everything from the SOF to the end of the data field) of a data or remote frame.

ACK Field

The ACK field primarily consists of an ACK Slot and an ACK Delim. The ACK field indicates that the receiver has received an effective message from the transmitter.

Table 56.2-1. Data Frames and Remote Frames in SFF and EFF

Data/Remote Frames	Description
SOF	The SOF (Start of Frame) is a single dominant bit used to synchronize nodes on the bus.
Base ID	The Base ID (ID.28 to ID.18) is the 11-bit identifier for SFF, or the first 11 bits of the 29-bit identifier for EFF.
RTR	The RTR (Remote Transmission Request) bit indicates whether the message is a data frame (dominant) or a remote frame (recessive). This means that a remote frame will always lose arbitration to a data frame if they have the same ID.

Cont'd on next page

Table 56.2-1 – cont'd from previous page

Data/Remote Frames	Description
SRR	The SRR (Substitute Remote Request) bit is transmitted in EFF to substitute for the RTR bit at the same position in SFF.
IDE	The IDE (Identifier Extension) bit indicates whether the message is SFF (dominant) or EFF (recessive). This means that a SFF frame will always win arbitration over an EFF frame if they have the same Base ID.
Extd ID	The Extended ID (ID.17 to ID.0) is the remaining 18 bits of the 29-bit identifier for EFF.
r1	The r1 bit (reserved bit 1) is always dominant.
r0	The r0 bit (reserved bit 0) is always dominant.
DLC	The DLC (Data Length Code) is 4-bit long and should contain any value from 0 to 8. Data frames use the DLC to indicate the number of data bytes in the data frame. Remote frames used the DLC to indicate the number of data bytes to request from another node.
Data Bytes	The data payload of data frames. The number of bytes should match the value of DLC. Data byte 0 is transmitted first, and each data byte is transmitted from the most significant bit first.
CRC Sequence	The CRC sequence is a 15-bit cyclic redundancy code.
CRC Delim	The CRC Delim (CRC Delimiter) is a single recessive bit that follows the CRC sequence.
ACK Slot	The ACK Slot (Acknowledgment Slot) is intended for receiver nodes to indicate that the data or remote frame was received without any issue. The transmitter node will send a recessive bit in the ACK Slot and receiver nodes should override the ACK Slot with a dominant bit if the frame was received without errors.
ACK Delim	The ACK Delim (Acknowledgment Delimiter) is a single recessive bit.
EOF	The EOF (End of Frame) marks the end of a data or remote frame, and consists of seven recessive bits.

56.2.2.2 Error and Overload Frames

Error Frames

Error frames are transmitted when a node detects a bus error. Error frames notably include an Error Flag which is made up of six consecutive bits of the same value, thus violating the bit-stuffing rule. Therefore, when a particular node detects a bus error and transmits an error frame, all other nodes will then detect a stuff error and transmit their own error frames in response. This has the effect of propagating the detection of a bus error across all nodes on the bus.

When a node detects a bus error, it will transmit an error frame starting from the next bit. However, when a node detects a CRC error, the error frame will start at the bit following the ACK Delim (see Section 56.2.3 for more details). The following Figure 56.2-2 shows different fields of an error frame:

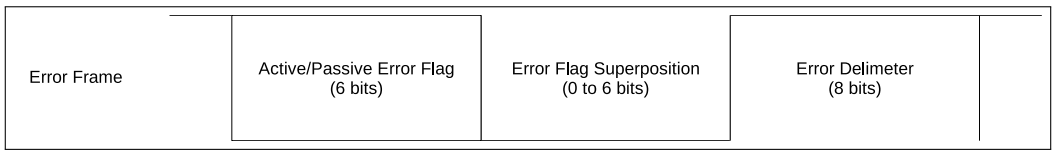


Figure 56.2-2. Fields of an Error Frame

Table 56.2-2. Error Frame

Error Frame	Description
Error Flag	The Error Flag has two forms, the Active Error Flag consisting of six dominant bits and the Passive Error Flag consisting of six recessive bits (unless overridden by dominant bits of other nodes). Active Error Flags are sent by error active nodes, whilst Passive Error Flags are sent by error passive nodes.
Error Flag Superposition	The Error Flag Superposition field meant to allow for other nodes on the bus to transmit their respective Active Error Flags. The superposition field can range from 0 to 6 bits, and ends when the first recessive bit is detected (i.e., the first it of the Delimiter).
Error Delimiter	The Delimiter field marks the end of the error/overload frame, and consists of eight recessive bits.

Overload Frames

An overload frame has the same bit fields as an error frame containing an Active Error Flag. The key difference is in the cases that can trigger the transmission of an overload frame. Figure 56.2-3 below shows the bit fields of an overload frame.

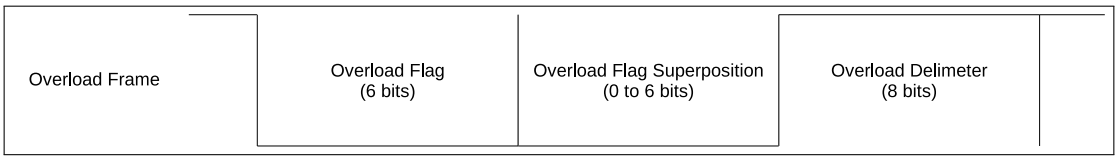


Figure 56.2-3. Fields of an Overload Frame

Table 56.2-3. Overload Frame

Overload Flag	Description
Overload Flag	Consists of six dominant bits. Same as an Active Error Flag.
Overload Flag Superposition	Allows for the superposition of Overload Flags from other nodes, similar to an Error Flag Superposition.
Overload Delimiter	Consists of eight recessive bits. Same as an Error Delimiter.

Overload frames will be transmitted in the following cases:

1. A receiver requires a delay of the next data or remote frame.
2. A dominant bit is detected at the first and second bit of intermission.

3. A dominant bit is detected at the eighth (last) bit of an Error Delimiter. Note that in this case, TEC and REC will not be incremented (see Section 56.2.3 for more details).

Transmitting an overload frame due to one of the above cases must also satisfy the following rules:

- The start of an overload frame due to case 1 is only allowed to be started at the first bit time of an expected intermission. In this case, a maximum of two overload frames may be generated.
- The start of an overload frame due to case 2 and 3 is only allowed to be started one bit after detecting the dominant bit.

56.2.2.3 Interframe Space

The Interframe Space acts as a separator between frames. Before sending, data frames and remote frames must be separated from preceding frames by an Interframe Space, regardless of the preceding frame's type (data frame, remote frame, error frame, or overload frame). However, error frames and overload frames do not need to be separated from preceding frames.

Figure 56.2-4 shows the fields within an Interframe Space:

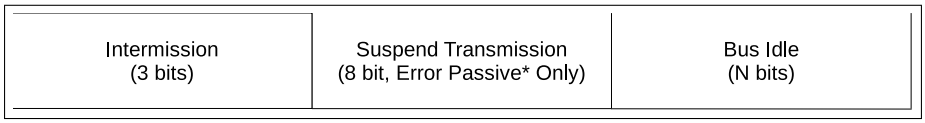


Figure 56.2-4. The Fields within an Interframe Space (*Error Passive)

Table 56.2-4. Interframe Space

Interframe Space	Description
Intermission	The Intermission consists of three recessive bits.
Suspend Transmission	An Error Passive node that has just transmitted a message must include a Suspend Transmission field. This field consists of eight recessive bits. Error Active nodes should not include this field.
Bus Idle	The Bus Idle field is of arbitrary length. Bus Idle ends when an SOF is transmitted. If a node has a pending transmission, the SOF should be transmitted at the first bit following Suspend Transmission.

56.2.3 TWAI Errors

56.2.3.1 Error Types

Bus Errors in TWAI are categorized into the following types:

Bit Error

A Bit Error occurs when a node transmits a bit value (i.e., dominant or recessive) but detects an opposite bit (e.g., a dominant bit is transmitted but a recessive is detected). However, if the transmitted bit is recessive and is located in the Arbitration Field, ACK Slot, or Passive Error Flag, then detecting a dominant bit will not be considered as a Bit Error.

Stuff Error

A stuff error occurs when six consecutive bits of the same value are detected (which violates the bit-stuffing encoding rules).

CRC Error

The receiver will calculate the CRC value based on the valid bits of received data frames and remote frames (before bit stuffing). A CRC Error occurs when the CRC calculated by the receiver does not match the CRC sequence in the received data frames and remote frames.

Format Error

A Format Error occurs when a format-fixed bit field of a message contains an illegal bit. For example, the r1 and r0 fields must be dominant.

ACK Error

An ACK Error occurs when a transmitter does not detect a dominant bit at the ACK Slot.

56.2.3.2 Error States

TWAI nodes implement fault confinement by maintaining two error counters in each node, where the counter values determine the error state. The two error counters are known as the Transmit Error Counter (TEC) and Receive Error Counter (REC). TWAI has the following error states:

Error Active

An Error Active node is able to participate in bus communication and transmit an Active Error Flag when it detects an error.

Error Passive

An Error Passive node is able to participate in bus communication and transmit a Passive Error Flag when it detects an error. Error Passive nodes that have transmitted data or remote frames must also include the Suspend Transmission field in the subsequent Interframe Space.

Bus Off

A Bus Off node is not permitted to influence the bus in any way (i.e., is not allowed to transmit data).

56.2.3.3 Error State Transition

1. A node becomes Error Passive when its TEC and/or REC is greater than or equal to 128. Though the node becomes Error Passive, it still sends an Active Error Flag. Note that once the REC has reached 128, any further increases to its value are invalid until the REC returns to a value less than 128.
2. A node becomes Bus Off when its TEC is greater than or equal to 256.
3. An Error Passive node becomes Error Active when both the TEC and REC are less than or equal to 127.
4. A Bus Off node can become Error Active (with both its TEC and REC reset to 0) after it monitors 128 occurrences of 11 consecutive recessive bits on the bus.

56.2.3.4 Error Counter Rules

The TEC and REC are incremented/decremented according to the following rules. **Note that more than one rule can apply to a given message transfer.**

TEC increment/decrement rules:

1. When a transmitter sends an Error Flag, the TEC is increased by 8. However, the following scenarios are exempt from this rule:
   - A transmitter is Error Passive and no dominant bit is detected when an Acknowledgment Error is detected and the Passive Error Flag is sent. In this case, the TEC should not be increased.
   - A transmitter transmits an Error Flag due to a Stuff Error during Arbitration. If the stuffed bit should have been recessive but was monitored as dominant, then the TEC should not be increased.
2. If a transmitter detects a Bit Error while sending an Active Error Flag or Overload Flag, the TEC is increased by 8.
3. A node can tolerate up to 7 consecutive dominant bits after sending an Active/Passive Error Flag, or Overload Flag. After detecting the 14th consecutive dominant bit when sending an Active Error Flag or Overload Flag, or the 8th consecutive dominant bit following a Passive Error Flag, a transmitter will increase its TEC by 8 and a receiver will increase its REC by 8. Every additional 8 consecutive dominant bits will also increase the TEC for transmitters or REC for receivers by 8 as well.
4. When a transmitter has transmitted a message, which means getting ACK and no errors until the EOF is completed, the TEC is decremented by 1, unless the TEC is already at 0.

REC increment/decrement rules:

1. When a receiver detects a dominant bit as the first bit after sending an Error Flag, the REC is increased by 8.
2. When a receiver detects an error, the REC is increased by 1, except when the detected error was a Bit Error during the transmission of an Active Error Flag or an Overload Flag.
3. If a receiver detects a Bit Error while sending an Active Error Flag or Overload Flag, the REC is increased by 8.
4. When a receiver successfully receives a message, which means getting no errors before ACK Slot and successfully sending ACK, the REC is decremented accordingly.
   - If the REC is between 1 and 127, it will be decremented by 1.
   - If the REC is greater than 127, it will be set to 127.
   - If the REC is 0, it will remain 0.

56.2.4 TWAI Bit Timing

56.2.4.1 Nominal Bit

The TWAI protocol allows a TWAI bus to operate at a particular bit rate. However, all nodes within a TWAI bus must operate at the same bit rate.

- **The Nominal Bit Rate** is defined as the number of bits transmitted per second.
- **The Nominal Bit Time** is defined as **1/Nominal Bit Rate**.

A single Nominal Bit Time is divided into multiple segments, and each segment is made up of multiple Time Quanta. A **Time Quantum** is a minimum unit of time, and is implemented as some form of prescaled clock signal in each node. Figure 56.2-5 illustrates the segments within a single Nominal Bit Time.

TWAI controllers will operate in time steps of one Time Quanta where the state of the TWAI bus is analyzed. If the bus states in two consecutive Time Quanta are different (i.e., recessive to dominant or vice versa), an edge is generated. The intersection of PBS1 and PBS2 is considered the Sample Point and the sampled bus value is considered the value of that bit.

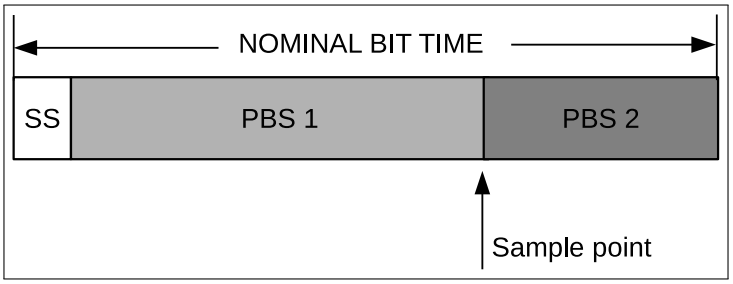


Figure 56.2-5. Layout of a Bit

Table 56.2-5. Segments of a Nominal Bit Time

Segment	Description
SS	The SS (Synchronization Segment) is 1 Time Quantum long. If all nodes are perfectly synchronized, the edge of a bit will lie in the SS.
PBS1	PBS1 (Phase Buffer Segment 1) can be 1 to 16 Time Quanta long. PBS1 is meant to compensate for the physical delay times within the network. PBS1 can also be lengthened for synchronization purposes.
PBS2	PBS2 (Phase Buffer Segment 2) can be 1 to 8 Time Quanta long. PBS2 is meant to compensate for the information processing time of nodes. PBS2 can also be shortened for synchronization purposes.

56.2.4.2 Hard Synchronization and Resynchronization

Due to clock skew and jitter, the bit timing of nodes on the same bus may become out of phase. Therefore, a bit edge may come before or after the SS. To ensure that the internal bit timing clocks of each node are kept in phase, TWAI has various methods of synchronization. The **Phase Error "e"** is measured in the number of Time Quanta and relative to the SS.

- A positive Phase Error (e > 0) is when the edge lies after the SS and before the Sample Point (i.e., the edge is late).
- A negative Phase Error (e < 0) is when the edge lies after the Sample Point of the previous bit and before SS (i.e., the edge is early).

To correct Phase Errors, there are two forms of synchronization, known as **Hard Synchronization** and **Resynchronization**. **Hard Synchronization** and **Resynchronization** obey the following rules:

- Only one synchronization may occur in a single bit time.
- Synchronizations only occur on recessive to dominant edges.

Hard Synchronization

Hard Synchronization occurs on the recessive to dominant (i.e., the first SOF bit after Bus Idle) edges when the bus is idle. All nodes will restart their internal bit timings so that the recessive to dominant edge lies within the SS of the restarted bit timing.

Resynchronization

Resynchronization occurs on recessive to dominant edges when the bus is not idle. If the edge has a positive Phase Error (e > 0), PBS1 is lengthened by a certain number of Time Quanta. If the edge has a negative Phase Error (e < 0), PBS2 will be shortened by a certain number of Time Quanta.

The number of Time Quanta to lengthen or shorten depends on the magnitude of the Phase Error, and is also limited by the Synchronization Jump Width (SJW) value which is programmable.

- When the magnitude of the Phase Error (**e**) is less than or equal to the SJW, PBS1/PBS2 are lengthened/shortened by the **e** number of Time Quanta. This has the same effect as Hard Synchronization.
- When the magnitude of the Phase Error is greater than the SJW, PBS1/PBS2 are lengthened/shortened by the SJW number of Time Quanta. This means it may take multiple bits of synchronization before the Phase Error is entirely corrected.

56.3 Architectural Overview

The major functional blocks of the TWAI controller are shown in Figure 56.3-1.

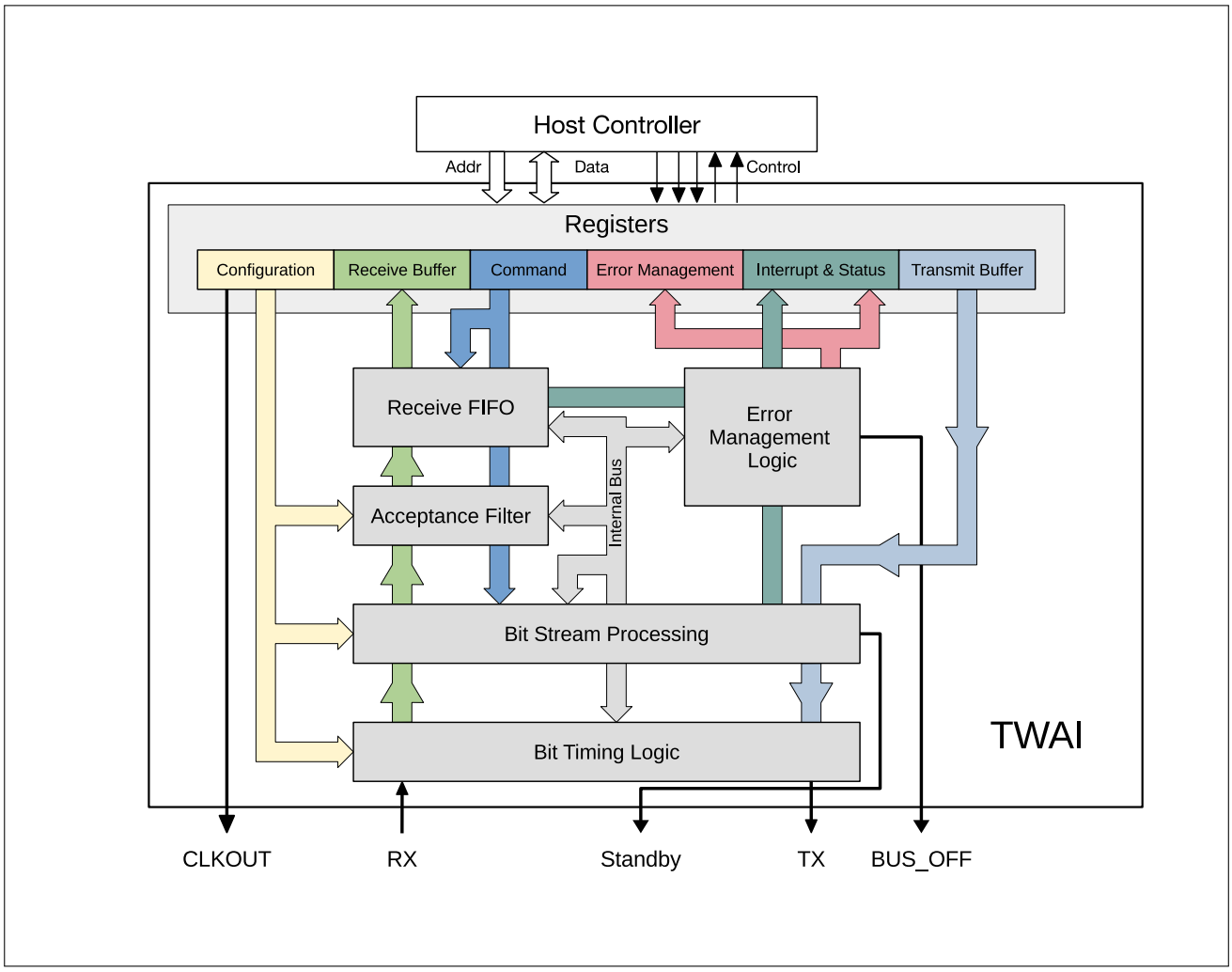


Figure 56.3-1. TWAI Overview Diagram

56.3.1 Bit Timing Logic

The Bit Timing Logic (BTL) module transmits and receives messages at the configured bit rate. The BTL module also handles bit timing synchronization so that communication remains stable. A single bit rate consists of multiple programmable segments that allows users to set the number of time quanta to adjust propagation delay and controller processing time, etc.

56.3.2 Bit Stream Processor

The Bit Stream Processing (BSP) module handles the frame processing of data from the Transmit Buffer (e.g., bit stuffing and additional CRC fields) and generates a bit stream for the Bit Timing Logic (BTL) module. At the same time, the BSP module is also responsible for processing the received bit stream (e.g., de-stuffing and verifying CRC) from the BTL module and placing the message into the Receive FIFO. The BSP will also detect errors on the TWAI bus and report them to the Error Management Logic (EML).

56.3.3 Acceptance Filter

The Acceptance Filter is a programmable message filtering unit that allows the TWAI controller to accept or reject a received message based on the message's ID field. Only accepted messages will be stored in the Receive FIFO. The Acceptance Filter can be configured to Single-filter mode or Dual-filter mode through registers.

56.3.4 Receive FIFO

The Receive FIFO is a 64-byte buffer (inside the TWAI controller) that stores received messages accepted by the Acceptance Filter. Messages in the Receive FIFO can vary in size (between 3 to 13 bytes). When the Receive FIFO is full (or does not have enough space to store the currently received message in its entirety), the Overrun Interrupt will be triggered, and any subsequently received messages will be lost until adequate space is cleared in the Receive FIFO. The first message in the Receive FIFO will be mapped to the 13-byte Receive Buffer until that message is cleared (using the Release Receive Buffer command bit). After being cleared, the Receive Buffer will map to the next message in the Receive FIFO, and the space occupied by the previous message in the Receive FIFO can be used to receive new messages.

56.3.5 Error Management Logic

The Error Management Logic (EML) module updates the TEC and REC, records error information like error types and positions, and updates the error state of the TWAI controller to ensure that the BSP module generates the correct Error Flags. Furthermore, this module also records the bit position when the TWAI controller loses arbitration.

56.3.6 Registers Block

The ESP32-P4 CPU accesses peripherals using 32-bit aligned words. However, the majority of registers in the TWAI controller only contain useful data at the least significant byte (bits [7:0]). Therefore, in these registers, bits [31:8] are ignored on writes, and return 0 on reads.

Configuration Registers

The configuration registers store various configuration items for the TWAI controller such as bit rates, Operation mode, Acceptance Filter, etc. Configuration registers can only be modified whilst the TWAI controller is in Reset mode (See Section 56.4.1).

Command Registers

The command register is used by the CPU to drive the TWAI controller to initiate certain actions such as transmitting a message or clearing the Receive Buffer. The command register can only be modified when the TWAI controller is in Operation mode (see Section 56.4.1).

Interrupt & Status Registers

The interrupt register indicates what events have occurred in the TWAI controller (each event is represented by a separate bit). The status register indicates the current status of the TWAI controller.

Error Management Registers

The error management registers include error counters and capture registers. The error counter registers represent TEC and REC values. The capture registers will record information about instances where the TWAI controller detects a bus error, or when it loses arbitration.

Transmit Buffer Registers

The transmit buffer is a 13-byte buffer used to store a TWAI message to be transmitted.

Receive Buffer Registers

The Receive Buffer is a 13-byte buffer which stores a single message. The Receive Buffer acts as a window of Receive FIFO, whose first message will be mapped into the Receive Buffer.

Note that the Transmit Buffer registers, Receive Buffer registers, and the Acceptance Filter registers share the same address range (offset 0x0040 to 0x0070). Their access is governed by the following rules:

- When the TWAI controller is in Reset mode, all reads and writes to the address range maps to the Acceptance Filter registers.
- When the TWAI controller is in Operation mode:
  - All reads to the address range maps to the Receive Buffer registers.
  - All writes to the address range maps to the Transmit Buffer registers.

56.4 Functional Description

56.4.1 Modes

The ESP32-P4 TWAI controller has two working modes: Reset mode and Operation mode. Reset mode and Operation mode are entered by setting or clearing the TWAI_RESET_MODE bit.

56.4.1.1 Reset Mode

Entering Reset mode is required in order to modify the various configuration registers of the TWAI controller. When entering Reset mode, the TWAI controller is essentially disconnected from the TWAI bus. When in Reset mode, the TWAI controller will not be able to transmit any messages (including error signals). Any transmission in progress is immediately terminated. Likewise, the TWAI controller will not be able to receive any messages either.

56.4.1.2 Operation Mode

In Operation mode, the TWAI controller connects to the bus and write-protects all configuration registers to ensure consistency during operation. When in Operation mode, the TWAI controller can transmit and receive messages (including error signaling) depending on which operation sub-mode the TWAI controller was configured with. The TWAI controller supports the following operation sub-modes:

- **Normal mode:** The TWAI controller can transmit and receive messages including error signals (such as Error and Overload Frames).
- **Self-test mode:** Self-test mode is similar to Normal mode, but the TWAI controller will consider the transmission of a data or remote frame successful and do not generate an ACK error even if it was not acknowledged. This mode is commonly used during the self-test of a TWAI controller.
- **Listen-only mode:** The TWAI controller will be able to receive messages, but will remain completely passive on the TWAI bus. Thus, the TWAI controller will not be able to transmit any messages, acknowledgments, or error signals. The error counters will remain frozen. This mode is useful for TWAI bus monitoring.

Note that when exiting Reset mode (i.e., entering Operation mode), the TWAI controller must wait for 11 consecutive recessive bits to occur before fully connecting to the TWAI bus (i.e., being able to transmit or receive).

56.4.2 Bit Timing

The operating bit rate of the TWAI controller must be configured whilst the TWAI controller is in Reset mode. The bit rate is configured using TWAI_BUS_TIMING_0_REG and TWAI_BUS_TIMING_1_REG.

The following Table 56.4-1 illustrates the bit fields of TWAI_BUS_TIMING_0_REG. The frequency of the TWAI core clock has multiple clock sources that can be configured by the user as needed. See Chapter 11 *Reset and Clock* for detailed configuration instructions.

Table 56.4-1. Bit Information of TWAI_BUS_TIMING_0_REG (0x18)

Bit 31-16	Bit 15	Bit 14	Bit 13	Bit 12	.....	Bit 1	Bit 0
Reserved	SJW.1	SJW.0	BRP.13	BRP.12	.....	BRP.1	BRP.0

Notes:

- BRP: The TWAI Time Quanta clock is derived from the XTAL clock (the default is 40 MHz and is configured). The Baud Rate Prescaler (BRP) field is used to define the prescaler according to the equation below, where t_{Tq} is the Time Quanta clock cycle and t_{CLK} is TWAI core clock cycle:
  $t_{Tq} = 2 \times t_{CLK} \times (BRP + 1)$
- SJW (t_{SJW}): Synchronization Jump Width (SJW) t_{SJW} is configured in SJW[1:0] where t_{SJW} = (SJW[1:0] + 1).

The following Table 56.4-2 illustrates the bit fields of TWAI_BUS_TIMING_1_REG.

Table 56.4-2. Bit Information of TWAI_BUS_TIMING_1_REG (0x1c)

Bit 31-8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Reserved	SAM	PBS2.2	PBS2.1	PBS2.0	PBS1.3	PBS1.2	PBS1.1	PBS1.0

Notes:

- PBS1: The number of Time Quanta in Phase Buffer Segment 1 is defined based on PBS1 + 1.
- PBS2: The number of Time Quanta in Phase Buffer Segment 2 is defined based on PBS2 + 1.
- SAM: Enables triple sampling if set to 1. This is useful for low/medium speed buses to filter spikes on the bus line.

56.4.3 Transmit and Receive Buffers

56.4.3.1 Overview of Buffers

Table 56.4-3. Buffer Layout for Standard Frame Format and Extended Frame Format

Standard Frame Format (SFF)		Extended Frame Format (EFF)	
Offset Address	Content	Offset Address	Content
0x40	TX/RX frame information	0x40	TX/RX frame information
0x44	TX/RX identifier 1	0x44	TX/RX identifier 1
0x48	TX/RX identifier 2	0x48	TX/RX identifier 2
0x4c	TX/RX data byte 1	0x4c	TX/RX identifier 3
0x50	TX/RX data byte 2	0x50	TX/RX identifier 4
0x54	TX/RX data byte 3	0x54	TX/RX data byte 1
0x58	TX/RX data byte 4	0x58	TX/RX data byte 2
0x5c	TX/RX data byte 5	0x5c	TX/RX data byte 3
0x60	TX/RX data byte 6	0x60	TX/RX data byte 4
0x64	TX/RX data byte 7	0x64	TX/RX data byte 5
0x68	TX/RX data byte 8	0x68	TX/RX data byte 6
0x6c	reserved	0x6c	TX/RX data byte 7
0x70	reserved	0x70	TX/RX data byte 8

Table 56.4-3 illustrates the layout of the Transmit Buffer and Receive Buffer registers. Both the Transmit and Receive Buffer registers share the same address space and are only accessible when the TWAI controller is in Operation mode. The CPU accesses Transmit Buffer registers for write operations, and Receive Buffer registers for read operations. Both buffers share the exact same register layout and fields to store a message (received or to be transmitted). The Transmit Buffer registers are used to configure a TWAI message to be transmitted. The CPU would write to the Transmit Buffer registers specifying the message's frame type, frame format, frame ID, and frame data (payload). Once the Transmit Buffer is configured, the CPU would then initiate the transmission by setting the TWAI_TX_REQUEST bit in TWAI_CMD_REG.

- For a self-reception request, set the TWAI_SELF_RX_REQUEST bit instead.
- For a single-shot transmission, set both the TWAI_TX_REQUEST and the TWAI_ABORT_TX simultaneously.

The Receive Buffer registers map the first message in the Receive FIFO. The CPU would read the Receive Buffer registers to obtain the first message's frame type, frame format, frame ID, and frame data (payload). Once the message has been read from the Receive Buffer registers, the CPU can set the TWAI_RELEASE_BUFFER bit in TWAI_CMD_REG to clear the Receive Buffer registers. If there are still messages in the Receive FIFO, the Receive Buffer registers will map the first of the remaining messages again.

56.4.3.2 Frame Information

The frame information is one byte long and specifies a message's frame type, frame format, and length of data. The frame information fields are shown in Table 56.4-4.

Table 56.4-4. TX/RX Frame Information (SFF/EFF); TWAI Address 0x40

Bit 31-8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Reserved	FF[1]	RTR[2]	X[3]	X[3]	DLC.3[4]	DLC.2[4]	DLC.1[4]	DLC.0[4]

Notes:

1. FF: The Frame Format (FF) bit specifies whether the message is Extended Frame Format (EFF) or Standard Frame Format (SFF). The message is EFF when the FF bit is 1, and SFF when the FF bit is 0.
2. RTR: The Remote Transmission Request (RTR) bit specifies whether the message is a data frame or a remote frame. The message is a remote frame when the RTR bit is 1, and a data frame when the RTR bit is 0.
3. X: Don't care, can be any value.
4. DLC: The Data Length Code (DLC) field specifies the number of data bytes for a data frame, or the number of data bytes to request in a remote frame. TWAI data frames are limited to a maximum payload of 8 data bytes, and thus the DLC should range from 0 to 8.

56.4.3.3 Frame Identifier

The Frame Identifier fields occupy two-byte (11-bit) long if the message is SFF, and four-byte (29-bit) long if the message is EFF.

The Frame Identifier fields for an SFF (11-bit) message are shown in Table 56.4-5 ~ 56.4-6.

Table 56.4-5. TX/RX Identifier 1 (SFF); TWAI Address 0x44

Bit 31-8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Reserved	ID.10	ID.9	ID.8	ID.7	ID.6	ID.5	ID.4	ID.3

Table 56.4-6. TX/RX Identifier 2 (SFF); TWAI Address 0x48

Bit 31-8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Reserved	ID.2	ID.1	ID.0	X[1]	X[2]	X[2]	X[2]	X[2]

Notes:

1. Don't care. Recommended to be compatible with receive buffer (i.e., set to RTR) in case of using the self-reception functionality (or together with self-test functionality).

2. Don't care. Recommended to be compatible with receive buffer (i.e., set to 0) in case of using the self-reception functionality (or together with self-test functionality).

The Frame Identifier fields for an EFF (29-bits) message is shown in Table 56.4-7 ~ 56.4-10.

Table 56.4-7. TX/RX Identifier 1 (EFF); TWAI Address 0x44

Bit 31-8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Reserved	ID.28	ID.27	ID.26	ID.25	ID.24	ID.23	ID.22	ID.21

Table 56.4-8. TX/RX Identifier 2 (EFF); TWAI Address 0x48

Bit 31-8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Reserved	ID.20	ID.19	ID.18	ID.17	ID.16	ID.15	ID.14	ID.13

Table 56.4-9. TX/RX Identifier 3 (EFF); TWAI Address 0x4c

Bit 31-8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Reserved	ID.12	ID.11	ID.10	ID.9	ID.8	ID.7	ID.6	ID.5

Table 56.4-10. TX/RX Identifier 4 (EFF); TWAI Address 0x50

Bit 31-8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Reserved	ID.4	ID.3	ID.2	ID.1	ID.0	X[1]	X[2]	X[2]

Notes:

1. Don't care. Recommended to be compatible with receive buffer (i.e., set to RTR) in case of using the self-reception functionality (or together with self-test functionality).
2. Don't care. Recommended to be compatible with receive buffer (i.e., set to 0) in case of using the self-reception functionality (or together with self-test functionality).

56.4.3.4 Frame Data

The Frame Data field contains the payloads of transmitted or received data frame, and can range from 0 to 8 bytes. The number of valid bytes should be equal to the DLC. However, if the DLC is larger than eight bytes, the number of valid bytes would still be limited to eight. Remote frames do not have data payloads, so their Frame Data fields will be unused.

For example, when transmitting a data frame with five bytes, the CPU should write five to the DLC field, and then write data to the corresponding register of the first to the fifth data field. Likewise, when the CPU receives a data frame with a DLC of five data bytes, only the first to the fifth data byte will contain valid payload data for the CPU to read.

56.4.4 Receive FIFO and Data Overruns

The Receive FIFO is a 64-byte internal buffer used to store received messages in First In First Out order. A single received message can occupy between 3 to 13 bytes of space in the Receive FIFO, and their

endianness is identical to the register layout of the Receive Buffer registers. The Receive Buffer registers are mapped to the bytes of the first message in the Receive FIFO.

When the TWAI controller receives the first message, the value of TWAI_RX_MESSAGE_COUNTER increases to 1, and the RXI interrupt is activated, automatically updating the first message into the buffer. Subsequently, with each new message received by the TWAI controller, the value of TWAI_RX_MESSAGE_COUNTER increments by 1, reaching a maximum of 64. When this limit is reached and there is sufficient space remaining in the Receive FIFO, the message will be written into the FIFO.

Once the software reads a message from the Receive Buffer, it can free up the space occupied by the message in the Receive FIFO by setting TWAI_RELEASE_BUFFER to 1. This will also decrement TWAI_RX_MESSAGE_COUNTER by 1. The Receive Buffer will then map to the next message in the Receive FIFO. The software should repeat the above operations until TWAI_RX_MESSAGE_COUNTER reaches 0. At this point, the RXI interrupt stops being triggered, the data in the receive buffer becomes invalid, and TWAI_STATUS_RECEIVE_BUFFER is set to 0.

A data overrun occurs when the TWAI controller receives a message, but the Receive FIFO lacks adequate free space to store the received message in its entirety (either due to the message contents being larger than the free space in the Receive FIFO, or the Receive FIFO being completely full).

When a data overrun occurs:

- The free space left in the Receive FIFO is filled with the partial contents of the overrun message. If the Receive FIFO is already full, then none of the overrun message's contents will be stored.
- When data in the Receive FIFO overruns for the first time, a Data Overrun Interrupt will be triggered.
- Each overrun message will still increment the TWAI_RX_MESSAGE_COUNTER up to a maximum of 64.
- The Receive FIFO will internally mark overrun messages as invalid. The TWAI_STATUS_MISS bit can be used to determine whether the message currently mapped to by the Receive Buffer is valid or overrun.

To clear an overrun Receive FIFO, the TWAI_RELEASE_BUFFER must be called repeatedly until TWAI_RX_MESSAGE_COUNTER is 0. This requires users to read all valid messages in the Receive FIFO and clear all overrun messages.

56.4.5 Acceptance Filter

The Acceptance Filter allows the TWAI controller to filter out received messages based on their ID (and optionally their first data byte and frame type). Only accepted messages are passed on to the Receive FIFO. The use of Acceptance Filters allows a more lightweight operation of the TWAI controller (e.g., less use of Receive FIFO, fewer Receive Interrupts) since the TWAI Controller only needs to handle a subset of messages.

The Acceptance Filter configuration registers can only be accessed whilst the TWAI controller is in Reset mode, since they share the same address spaces with the Transmit Buffer and Receive Buffer registers.

The configuration registers consist of a 32-bit Acceptance Code Value and a 32-bit Acceptance Mask Value. The Acceptance Code value specifies a bit pattern which each filtered bit of the message must match in order for the message to be accepted. The Acceptance Mask Value is able to mask out certain bits of the Code value (i.e., set as "Don't Care" bits). Each filtered bit of the message must either match the acceptance code or be masked in order for the message to be accepted, as demonstrated in Figure 56.4-1.

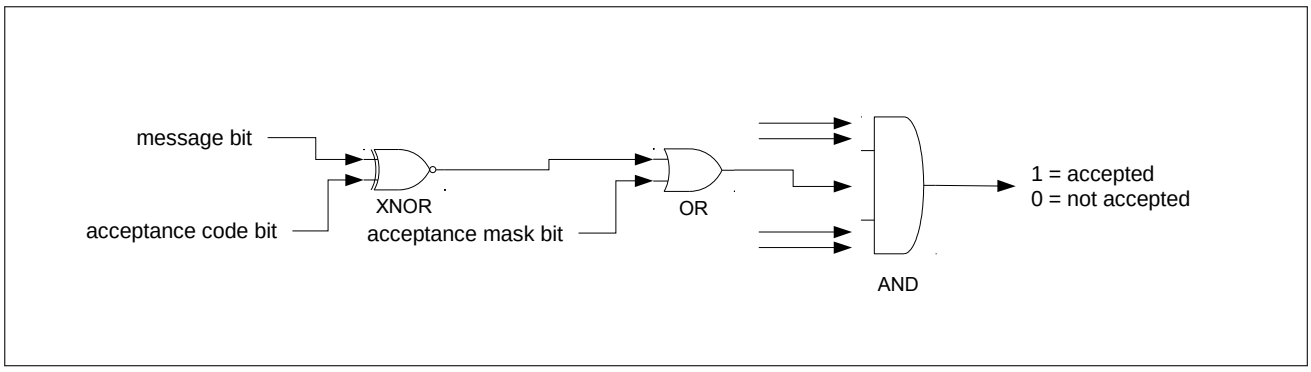


Figure 56.4-1. Acceptance Filter

The TWAI controller Acceptance Filter allows the 32-bit Acceptance Code and Mask Values to either define a single filter (i.e., Single-filter mode), or two filters (i.e., Dual-filter mode). How the Acceptance Filter interprets the 32-bit code and mask values is dependent on the filter mode and the format of received messages (i.e., SFF or EFF).

56.4.5.1 Single-Filter Mode

Single Filter mode is enabled by setting the TWAI_ACCEPTANCE_FILTER_MODE bit to 1. This will cause the 32-bit code and mask values to define a single filter. The single filter can filter the following bits of data or remote frames:

- SFF
  - The entire 11-bit ID
  - RTR bit
  - Data byte 1 and Data byte 2
- EFF
  - The entire 29-bit ID
  - RTR bit

The following Figure 56.4-2 illustrates how the 32-bit code and mask values will be interpreted under Single-filter mode.

56.4.5.2 Dual-Filter Mode

Dual-filter mode is enabled by clearing the TWAI_ACCEPTANCE_FILTER_MODE bit to 0. This will cause the 32-bit code and mask values to define a two separate filters referred to as filter 1 or filter 2. Under Dual-filter mode, a message will be accepted if it is accepted by one of the two filters.

The two filters can filter the following bits of data or remote frames:

- SFF
  - The entire 11-bit ID
  - RTR bit
  - Data byte 1 (for filter 1 only)

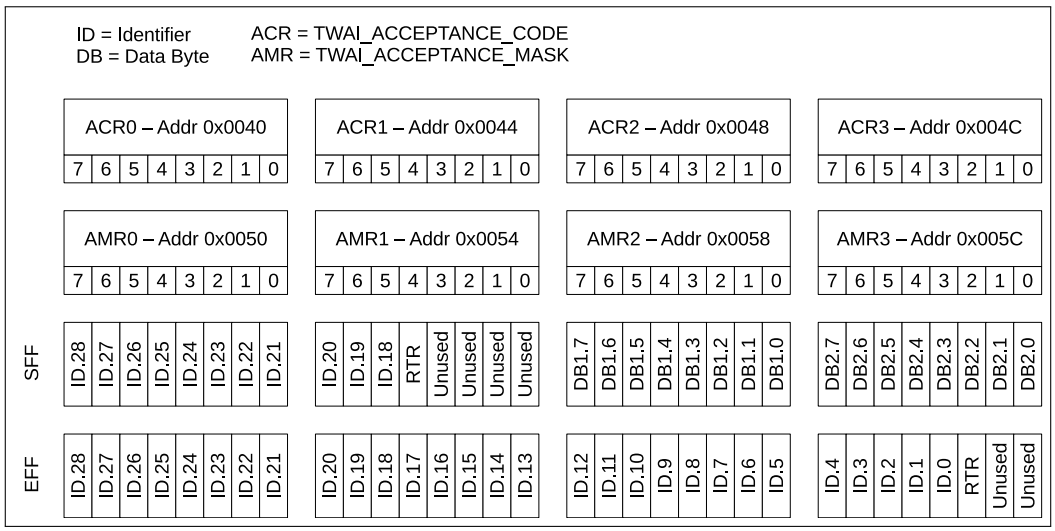


Figure 56.4-2. Single-Filter Mode

- EFF
  - The first 16 bits of the 29-bit ID

The following Figure 56.4-3 illustrates how the 32-bit code and mask values will be interpreted in Dual-filter mode.

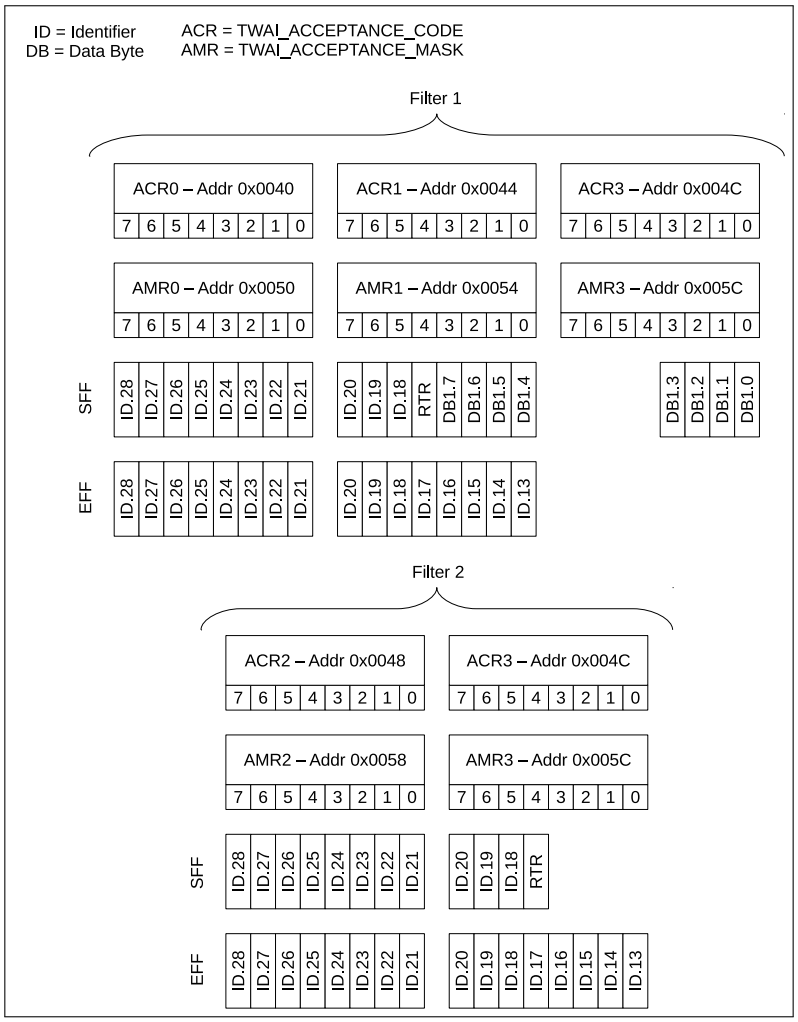


Figure 56.4-3. Dual-Filter Mode

56.4.6 Error Management

The TWAI protocol requires that each TWAI node maintains the Transmit Error Counter (TEC) and Receive Error Counter (REC). The value of both error counters determines the current error state of the TWAI controller (i.e., Error Active, Error Passive, Bus-Off). The TWAI controller stores the TEC and REC values in TWAI_TX_ERR_CNT_REG and TWAI_RX_ERR_CNT_REG respectively, and they can be read by the CPU anytime. In addition to the error states, the TWAI controller also offers an Error Warning Limit (EWL) feature that can warn users of the occurrence of severe bus errors before the TWAI controller enters the Error Passive state.

The current error state of the TWAI controller is indicated via a combination of the following values and status bits: TEC, REC, TWAI_STATUS_ERR, and TWAI_STATUS_NODE_BUS_OFF. Certain changes to these values and bits will also trigger interrupts, so that users are notified of error state transitions (see section 56.5). The following figure 56.4-4 shows the relation between the error states, values and bits, and error state related interrupts.

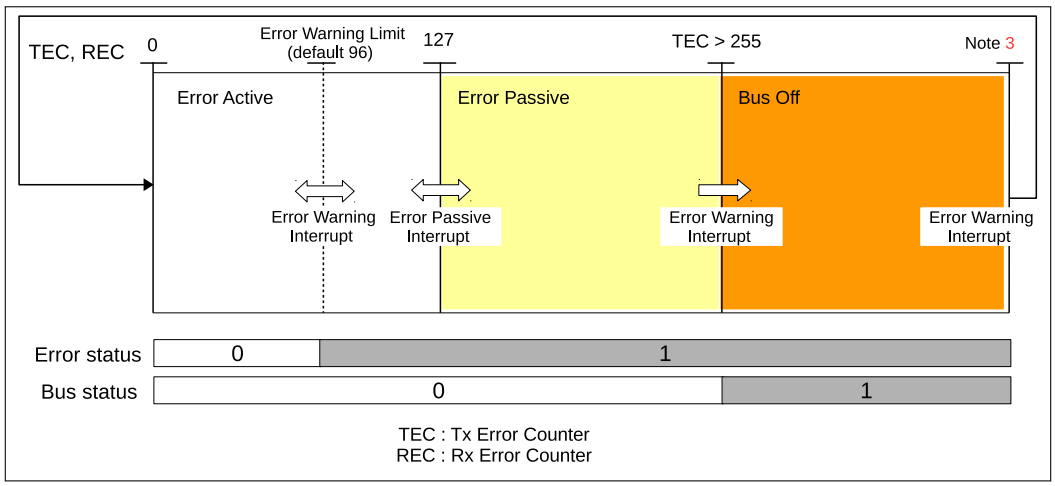


Figure 56.4-4. Error State Transition

56.4.6.1 Error Warning Limit

The Error Warning Limit (EWL) is a configurable threshold value for the TEC and REC, which will trigger an interrupt when exceeded. The EWL is intended to serve as a warning about severe TWAI bus errors, and is triggered before the TWAI controller enters the Error Passive state. The EWL is configured in TWAI_ERR_WARNING_LIMIT_REG and can only be configured whilst the TWAI controller is in Reset mode. The TWAI_ERR_WARNING_LIMIT_REG has a default value of 96.

When the values of TEC and/or REC are larger than or equal to the EWL value, the TWAI_STATUS_ERR bit is immediately set to 1. Likewise, when the values of both the TEC and REC are smaller than the EWL value, the TWAI_STATUS_ERR bit is immediately reset to 0. The Error Warning Interrupt is triggered whenever the value of the TWAI_STATUS_ERR bit (or the TWAI_STATUS_NODE_BUS_OFF) changes.

56.4.6.2 Error Passive

The TWAI controller is in the Error Passive state when the TEC or REC value exceeds 127. Likewise, when both the TEC and REC are less than or equal to 127, the TWAI controller enters the Error Active state. The Error Passive Interrupt is triggered whenever the TWAI controller transitions from the Error Active state to the Error Passive state or vice versa.

56.4.6.3 Bus-Off and Bus-Off Recovery

The TWAI controller enters the Bus-Off state when the TEC value exceeds 255. On entering the Bus-Off state, the TWAI controller will automatically do the following:

- Set REC to 0
- Set TEC to 127
- Set the TWAI_STATUS_NODE_BUS_OFF bit to 1
- Enter Reset mode

The Error Warning Interrupt is triggered whenever the value of the TWAI_STATUS_NODE_BUS_OFF bit (or the TWAI_STATUS_ERR bit) changes.

To return to the Error Active state, the TWAI controller must undergo Bus-Off Recovery. Bus-Off Recovery requires the TWAI controller to observe 128 occurrences of 11 consecutive recessive bits on the bus. To initiate Bus-Off Recovery (after entering the Bus-Off state), the TWAI controller should enter Operation mode by setting the TWAI_RESET_MODE bit to 0. The TEC tracks the progress of Bus-Off Recovery by decrementing the TEC each time when the TWAI controller observes 11 consecutive recessive bits. When Bus-Off Recovery has completed (i.e., TEC has decremented from 127 to 0), the TWAI_STATUS_NODE_BUS_OFF bit will automatically be reset to 0, thus triggering the Error Warning Interrupt.

56.4.7 Error Code Capture

The Error Code Capture (ECC) feature allows the TWAI controller to record the error type and bit position of a TWAI bus error in the form of an error code. Upon detecting a TWAI bus error, the Bus Error Interrupt is triggered and the error code is recorded in TWAI_ERR_CODE_CAP_REG. Subsequent bus errors will trigger the Bus Error Interrupt, but their error codes will not be recorded until the current error code is read from TWAI_ERR_CODE_CAP_REG.

The following Table 56.4-11 shows the fields of the TWAI_ERR_CODE_CAP_REG:

Table 56.4-11. Bit Information of TWAI_ERR_CODE_CAP_REG (0x30)

Bit 31-8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Reserved	ERRC.1[1]	ERRC.0[1]	DIR[2]	SEG.4[3]	SEG.3[3]	SEG.2[3]	SEG.1[3]	SEG.0[3]

Notes:

- ERRC: The Error Code (ERRC) indicates the type of bus error; 00 for bit error, 01 for format error, 10 for stuff error, and 11 for other types of error.
- DIR: The Direction (DIR) indicates whether the TWAI controller was transmitting or receiving when the bus error occurred; 0 for the transmitter, 1 for the receiver.
- SEG: The Error Segment (SEG) indicates the segment of the TWAI message at which the bus error occurred.

The following Table 56.4-12 shows how to interpret the SEG.0 to SEG.4 bits.

Table 56.4-12. Bit Information of Bits SEG.4 - SEG.0

Bit SEG.4	Bit SEG.3	Bit SEG.2	Bit SEG.1	Bit SEG.0	Description
0	0	0	1	1	start of frame
0	0	0	1	0	ID.28 ~ ID.21
0	0	1	1	0	ID.20 ~ ID.18
0	0	1	0	0	bit SRTR
0	0	1	0	1	bit IDE
0	0	1	1	1	ID.17 ~ ID.13
0	1	1	1	1	ID.12 ~ ID.5
0	1	1	1	0	ID.4 ~ ID.0
0	1	1	0	0	bit RTR
0	1	1	0	1	reserved bit 1
0	1	0	0	1	reserved bit 0

Cont'd on next page

Table 56.4-12 – cont'd from previous page

Bit SEG.4	Bit SEG.3	Bit SEG.2	Bit SEG.1	Bit SEG.0	Description
0	1	0	1	1	data length code
0	1	0	1	0	data field
0	1	0	0	0	CRC sequence
1	1	0	0	0	CRC delimiter
1	1	0	0	1	ACK slot
1	1	0	1	1	ACK delimiter
1	1	0	1	0	end of frame
1	0	0	1	0	intermission
1	0	0	0	1	active error flag
1	0	1	1	0	passive error flag
1	0	0	1	1	tolerate dominant bits
1	0	1	1	1	error delimiter
1	1	1	0	0	overload flag

Notes:

- Bit SRTR: under Standard Frame Format.
- Bit IDE: Identifier Extension Bit, 0 for Standard Frame Format.

56.4.8 Arbitration Lost Capture

The Arbitration Lost Capture (ALC) feature allows the TWAI controller to record the bit position where it loses arbitration. When the TWAI controller loses arbitration, the bit position is recorded in TWAI_ARB_LOST_CAP_REG and the Arbitration Lost Interrupt is triggered.

Subsequent losses in arbitration will trigger the Arbitration Lost Interrupt, but will not be recorded in TWAI_ARB_LOST_CAP_REG until the current Arbitration Lost Capture is read from the TWAI_ERR_CODE_CAP_REG.

Table 56.4-13 illustrates bits and fields of TWAI_ERR_CODE_CAP_REG whilst Figure 56.4-5 illustrates the bit positions of a TWAI message.

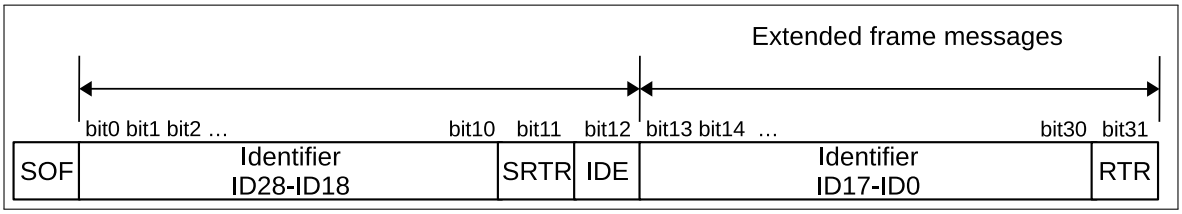


Figure 56.4-5. Positions of Arbitration Lost Bits

Table 56.4-13. Bit Information of TWAI_ARB_LOST_CAP_REG (0x2c)

Bit 31-5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Reserved	BITNO.4[1]	BITNO.3[1]	BITNO.2[1]	BITNO.1[1]	BITNO.0[1]

Notes:

- BITNO: Bit Number (BITNO) indicates the nth bit of a TWAI message where arbitration was lost.

56.4.9 Transceiver Auto-Standby

It is common for TWAI transceivers to support a Standby mode to lower power consumption. TWAI transceivers will usually generate a standby signal that is asserted by the connected TWAI controller, thus allowing the controller to place the transceiver into standby when appropriate (e.g., when the bus will be idle for an extended period of time). Transceivers will exit Standby mode if the controller de-asserts the standby signal, or if the transceiver detects bus activity (also known as a wake-up feature).

ESP32-P4's TWAI controller supports both hardware control (i.e., automatic) and software control (i.e., manual) of the standby signal to control the switching of TWAI transceivers connected to the chip. When hardware controlled, the TWAI controller will automatically assert the standby signal when the bus remains idle for longer than a configurable amount of time. When software controlled, the standby signal can be manually asserted/de-asserted directly by the software.

- Hardware output:
  1. Set the TWAI_HW_STANDBY_EN field in the TWAI_HW_CFG_REG register to enable standby function for hardware.
  2. Configure the TWAI_HW_STANDB_CNT_REG register. This register indicates the time required before hardware triggers the standby signal after entering idle status, in which the value indicates the number of cycles of the TWAI controller operating clock (40 MHz by default).
- Software output:
  1. Set the TWAI_SW_STANDBY_EN field in the TWAI_SW_STANDBY_CFG_REG register to generate standby signals in the TWAI controller.

The standby signal generated using either of the above methods will be pulled down (cleared) when either of the following conditions is met:

1. The standby signal will be automatically cleared when the TWAI controller's receiver exits the idle status (such as when receiving a new message).
2. Users can also pull down the standby signal by setting the TWAI_SW_STANDBY_CLR field in the TWAI_SW_STANDBY_CFG_REG register.

56.5 Interrupts

ESP32-P4's TWAI can generate the following interrupt signal that will be sent to the Interrupt Matrix.

- TWAI_INT

There are several internal interrupt sources from the TWAI that can generate the above interrupt signal. The interrupt sources from the TWAI are listed with their trigger conditions and the resulted interrupt signal in Table 56.5-1.

Table 56.5-1. TWAI's Internal Interrupt Sources

Internal Interrupt Source	Trigger Condition	Interrupt Signal
TWAI_RECEIVE_INT (RXI)	The Receive Interrupt (RXI) is asserted whenever the TWAI controller has received messages that are pending to be read from the Receive Buffer (i.e., when TWAI_RX_MESSAGE_COUNTER_REG > 0). Pending received messages includes valid messages in the Receive FIFO and also overrun messages. The RXI will not be deasserted until all pending received messages are cleared using the TWAI_RELEASE_BUFFER command bit.	TWAI_INT
TWAI_TRANSMIT_INT (TXI)	The Transmit Interrupt (TXI) is triggered whenever Transmit Buffer becomes free, indicating another message can be loaded into the Transmit Buffer for transmission. The Transmit Buffer becomes free under the following scenarios: • A message transmission has completed successfully, i.e., acknowledged without any errors. Any failed messages will automatically be resent. • A single shot transmission has completed (successfully or unsuccessfully, indicated by the TWAI_STATUS_TRANSMISSION_COMPLETE bit). • A message transmission was aborted using the TWAI_ABORT_TX command bit.	TWAI_INT

Cont'd on next page

Table 56.5-1 – cont'd from previous page

Internal Interrupt Source	Trigger Condition	Interrupt Signal
TWAI_ERR_WARNING_INT (EWI)	The Error Warning Interrupt (EWI) is triggered whenever there is a change to the TWAI_STATUS_ERR and TWAI_STATUS_NODE_BUS_OFF bits of the TWAI_STATUS_REG (i.e., transition from 0 to 1 or vice versa). Thus, an EWI could indicate one of the following events, depending on the values TWAI_STATUS_ERR and TWAI_STATUS_NODE_BUS_OFF at the moment when the EWI is triggered. • If TWAI_STATUS_ERR = 0 and TWAI_STATUS_NODE_BUS_OFF = 0: – If the TWAI controller was in the Error Active state, it indicates both the TEC and REC have returned below the threshold value set by TWAI_ERR_WARNING_LIMIT_REG. – If the TWAI controller was previously in the Bus Off Recovery state, it indicates that Bus Recovery has completed successfully. • If TWAI_STATUS_ERR = 1 and TWAI_STATUS_NODE_BUS_OFF = 0: The TEC or REC error counters have exceeded the threshold value set by TWAI_ERR_WARNING_LIMIT_REG. • If TWAI_STATUS_ERR = 1 and TWAI_STATUS_NODE_BUS_OFF = 1: The TWAI controller has entered the BUS_OFF state (due to the TEC >= 256). • If TWAI_STATUS_ERR = 0 and TWAI_STATUS_NODE_BUS_OFF = 1: The TWAI controller's TEC has dropped below the threshold value set by TWAI_ERR_WARNING_LIMIT_REG during BUS_OFF recovery.	TWAI_INT

Cont'd on next page

Table 56.5-1 – cont'd from previous page

Internal Interrupt Source	Trigger Condition	Interrupt Signal
TWAI_DATA_OVERRUN_INT (DOI)	The Data Overrun Interrupt (DOI) is triggered whenever the Receive FIFO has overrun. The DOI indicates that the Receive FIFO is full and should be cleared immediately to prevent any further overrun messages. The DOI is only triggered by the first message that causes the Receive FIFO to overrun (i.e., the transition from the Receive FIFO not being full to the Receive FIFO overrunning). Any subsequent overrun messages will not trigger the DOI again. The DOI could be triggered again when all received messages (valid or overrun) have been cleared.	TWAI_INT
TWAI_TS_COUNTER_OVFL_INT	The Timestamp Overflow Interrupt is triggered whenever the 32-bit counter of the TWAI controller timestamp has overflowed.	TWAI_INT
TWAI_ERR_PASSIVE_INT (EPI)	The Error Passive Interrupt (EPI) is triggered whenever the TWAI controller switches from Error Active to Error Passive, or vice versa.	TWAI_INT
TWAI_ARBITRATION_LOST_INT (ALI)	The Arbitration Lost Interrupt (ALI) is triggered whenever the TWAI controller is attempting to transmit a message and loses arbitration. The bit position where the TWAI controller lost arbitration is automatically recorded in Arbitration Lost Capture register (TWAI_ARB_LOST_CAP_REG). When the ALI occurs again, the Arbitration Lost Capture register will no longer record the new bit location until it is cleared (via CPU reading this register).	TWAI_INT
TWAI_BUS_ERR_INT (BEI)	The Bus Error Interrupt (BEI) is triggered whenever the TWAI controller detects an error on the TWAI bus. When a bus error occurs, the Bus Error type and its bit position are automatically recorded in the Error Code Capture register (TWAI_ERR_CODE_CAP_REG). When the BEI occurs again, the Error Code Capture register will no longer record new error information until it is cleared (via a read from the CPU).	TWAI_INT
TWAI_IDLE_INT (BISI)	The Bus Idle Status Interrupt (BISI) is triggered when the number of clock cycles of the TWAI controller in the idle status exceeds the pre-configured value in the TWAI_IDLE_INTR_CNT_REG register. Users can configure this interrupt to get the TWAI controller idle status and further decide whether to turn off the external TWAI receiver to reduce the overall power consumption (see Section 56.4.9).	TWAI_INT

The TWAI controller's interrupt signal to the interrupt matrix will be asserted whenever one or more interrupt bits are set in the TWAI_INTERRUPT_REG, and de-asserted when all bits in TWAI_INTERRUPT_REG are cleared. The majority of interrupt bits in TWAI_INTERRUPT_REG are automatically cleared when the register is read, except for the Receive Interrupt which can only be cleared when all the messages are released by setting the TWAI_RELEASE_BUFFER bit.

Note:
For definitions of *interrupt*, *interrupt signal*, *interrupt source*, and their correlations, please refer to Chapter 13 *Interrupt Matrix* > Section 13.2 *Interrupt Terminology in ESP32-P4*.

Each interrupt source can be configured by a common set of registers that are described in Section *Interrupt Configuration Registers*. The specific registers can be found in Section 56.6 *Register Summary*.

56.6 Register Summary

'|' here means a separate line to distinguish between TWAI working modes discussed in Section 56.4.1 *Modes*. The left describes the access in Operation mode. The right belongs to Reset mode and is marked in red. The addresses in this section are relative to Two-wire Automotive Interface base address provided in Table 9.3-2 in Chapter 9 *System and Memory*.

The abbreviations given in Column **Access** are explained in Section *Access Types for Registers*.

Name	Description	Address	Access
Configuration Registers			
TWAI_MODE_REG	Mode Register	0x0000	R/W
TWAI_CMD_REG	Command Register	0x0004	WO
TWAI_BUS_TIMING_0_REG	Bus Timing Register 0	0x0018	R/W
TWAI_BUS_TIMING_1_REG	Bus Timing Register 1	0x001C	R/W
TWAI_ERR_WARNING_LIMIT_REG	Error Warning Limit Register	0x0034	R/W
TWAI_CLOCK_DIVIDER_REG	Clock Divider Register	0x007C	R/W
TWAI_SW_STANDBY_CFG_REG	Software Standby Register	0x0080	R/W
TWAI_HW_CFG_REG	Hardware Standby Register	0x0084	R/W
TWAI_HW_STANDBY_CNT_REG	Standby Time Length Register	0x0088	R/W
TWAI_IDLE_INTR_CNT_REG	Idle Status Time Length Register	0x008C	R/W
Status Registers			
TWAI_STATUS_REG	Status Register	0x0008	RO
TWAI_ARB_LOST_CAP_REG	Arbitration Lost Capture Register	0x002C	RO
TWAI_ERR_CODE_CAP_REG	Error Code Capture Register	0x0030	RO
TWAI_RX_ERR_CNT_REG	Receive Error Counter Register	0x0038	R/W
TWAI_TX_ERR_CNT_REG	Transmit Error Counter Register	0x003C	R/W
TWAI_RX_MESSAGE_COUNTER_REG	Receive Message Counter Register	0x0074	RO
Interrupt Registers			
TWAI_INTERRUPT_REG	Interrupt Mask Register	0x000C	RO
TWAI_INTERRUPT_ENABLE_REG	Interrupt Enable Register	0x0010	varies
Data Registers			
TWAI_DATA_0_REG	Data Register 0	0x0040	R/W
TWAI_DATA_1_REG	Data Register 1	0x0044	R/W
TWAI_DATA_2_REG	Data Register 2	0x0048	R/W
TWAI_DATA_3_REG	Data Register 3	0x004C	R/W
TWAI_DATA_4_REG	Data Register 4	0x0050	R/W
TWAI_DATA_5_REG	Data Register 5	0x0054	R/W
TWAI_DATA_6_REG	Data Register 6	0x0058	R/W
TWAI_DATA_7_REG	Data Register 7	0x005C	R/W
TWAI_DATA_8_REG	Data Register 8	0x0060	R/W | R/O
TWAI_DATA_9_REG	Data Register 9	0x0064	R/W | R/O
TWAI_DATA_10_REG	Data Register 10	0x0068	R/W | R/O
TWAI_DATA_11_REG	Data Register 11	0x006C	R/W | R/O
TWAI_DATA_12_REG	Data Register 12	0x0070	R/W | R/O

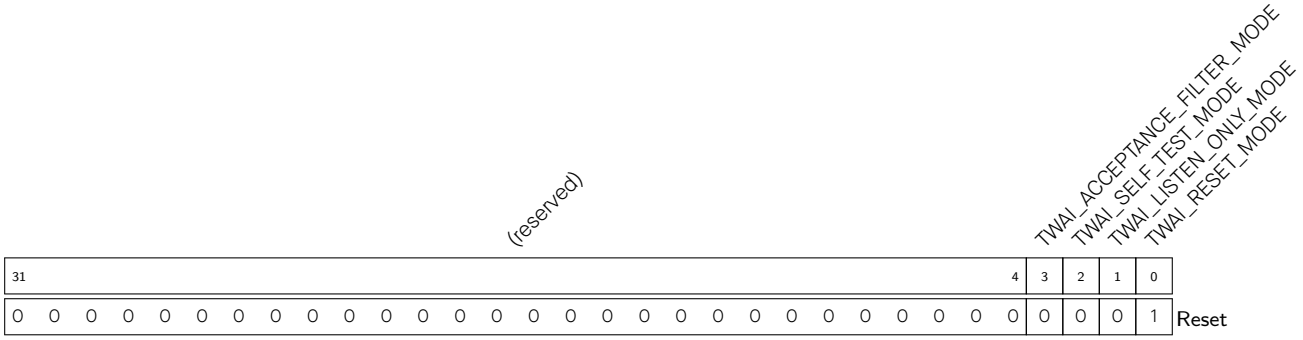
Name	Description	Address	Access
Timestamp Register			
TWAI_TIMESTAMP_DATA_REG	Timestamp Data Register	0x0094	RO
TWAI_TIMESTAMP_PRESCALER_REG	Timestamp Configuration Register 0	0x0098	R/W
TWAI_TIMESTAMP_CFG_REG	Timestamp Configuration Register 1	0x009C	R/W

56.7 Registers

'|' here means a separate line. The left describes the access in Operation mode. The right belongs to Reset mode with red color. The addresses in this section are relative to Two-wire Automotive Interface base address (each TWAI 0 and TWAI 1 has an individual base address) provided in Table 9.3-2 in Chapter 9 *System and Memory*.

For how to program reserved fields, please refer to Section *Programming Reserved Register Field*.

Register 56.1. TWAI_MODE_REG (0x0000)



TWAI_RESET_MODE Configures the Operation mode of the TWAI Controller.

0: Operation mode

1: Reset mode

(R/W)

TWAI_LISTEN_ONLY_MODE Configures whether to enter the Listen-only mode.

0: No effect

1: Listen-only mode. In this mode, the nodes will only receive messages from the bus, without generating the acknowledge signal or updating the RX error counter.

(R/W)

TWAI_SELF_TEST_MODE Configures whether to enter the Self-test mode.

0: No effect

1: Enter the Self-test mode. In this mode, the TX nodes can perform a successful transmission without receiving the acknowledge signal. This mode is often used to test a single node with the self-reception request command.

(R/W)

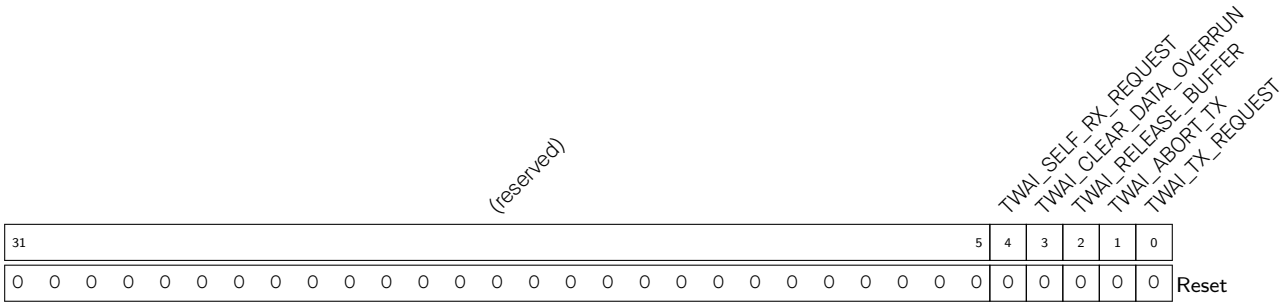
TWAI_ACCEPTANCE_FILTER_MODE Configures the filter mode.

0: Dual-filter mode

1: Single-filter mode

(R/W)

Register 56.2. TWAI_CMD_REG (0x0004)



TWAI_TX_REQUEST Configures whether to drive nodes to start transmission.

0: No effect

1: Drive nodes to start transmission

(WO)

TWAI_ABORT_TX Configures whether to cancel a pending transmission request.

0: No effect

1: Cancel a pending transmission request

(WO)

TWAI_RELEASE_BUFFER Configures whether to release the RX buffer.

0: No effect

1: Release the RX buffer

(WO)

TWAI_CLEAR_DATA_OVERRUN Configures whether to clear the data overrun status bit.

0: No effect

1: Clear the data overrun status bit

(WO)

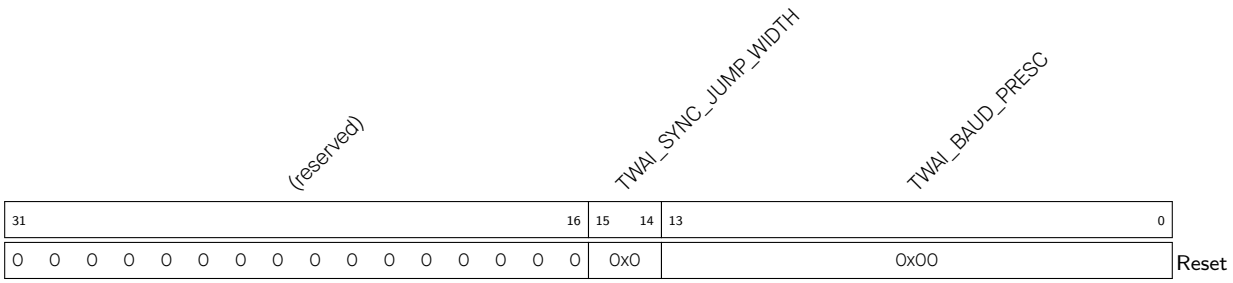
TWAI_SELF_RX_REQUEST Configures whether to drive nodes to start transmitting messages while also receiving messages on the bus simultaneously.

0: No effect

1: Drive nodes to start transmitting messages while also receiving messages on the bus (set the value of TWAI_TX_REQUEST to 0, or only transmission is allowed)

(WO)

Register 56.3. TWAI_BUS_TIMING_0_REG (0x0018)



TWAI_BAUD_PRESC Configures baud rate prescaler value, determining the frequency dividing ratio.

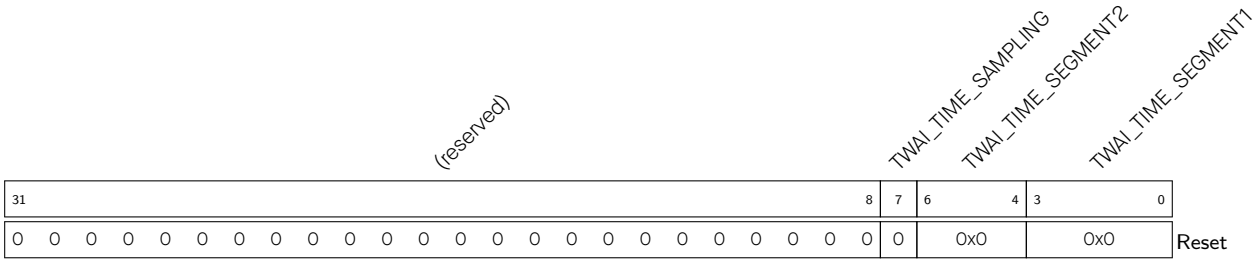
0: Low

1: High

(R/W)

TWAI_SYNC_JUMP_WIDTH Configures Synchronization Jump Width (SJW), ranging from 1 ~ 4 Tq wide. (R/W)

Register 56.4. TWAI_BUS_TIMING_1_REG (0x001C)



TWAI_TIME_SEGMENT1 Configures the width of PBS1. (R/W)

TWAI_TIME_SEGMENT2 Configures the width of PBS2. (R/W)

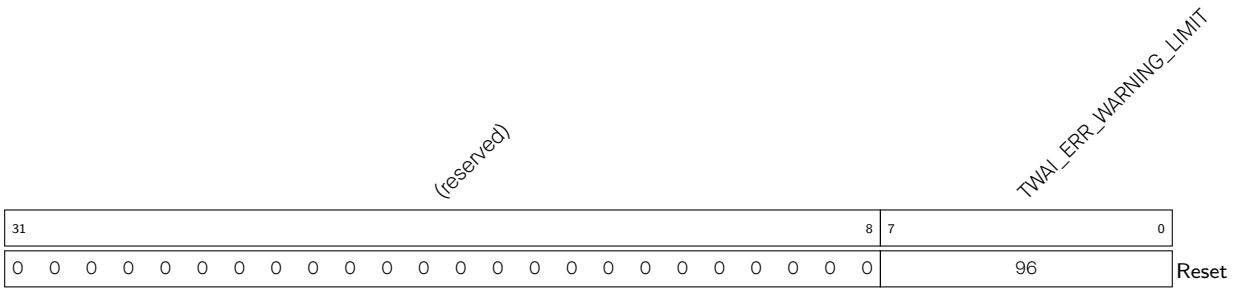
TWAI_TIME_SAMPLING Configures the number of sample points.

0: The bus is sampled once

1: The bus is sampled three times

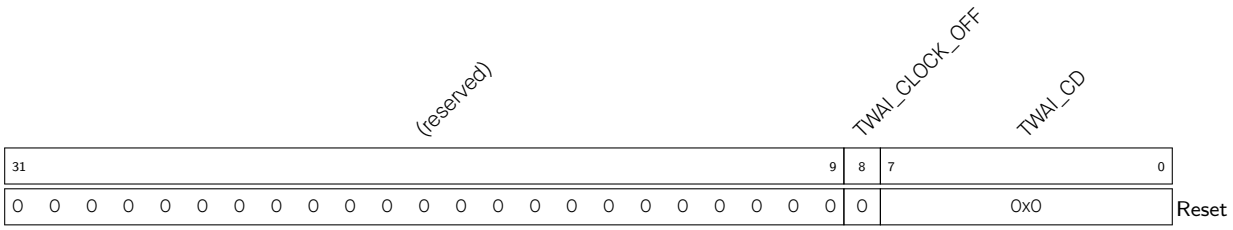
(R/W)

Register 56.5. TWAI_ERR_WARNING_LIMIT_REG (0x0034)



TWAI_ERR_WARNING_LIMIT Configures error warning threshold. When any of the error counter values (TWAI_RX_ERR_CNT or TWAI_TX_ERR_CNT) exceeds the threshold, the TWAI controller enters an error state; when all the error counter values are below the threshold, the TWAI controller exits the error state. An error warning interrupt will be triggered if the TWAI controller enters or exits an error state. (R/W)

Register 56.6. TWAI_CLOCK_DIVIDER_REG (0x007C)



TWAI_CD Configures the divisor of the external CLKOUT pin. (R/W)

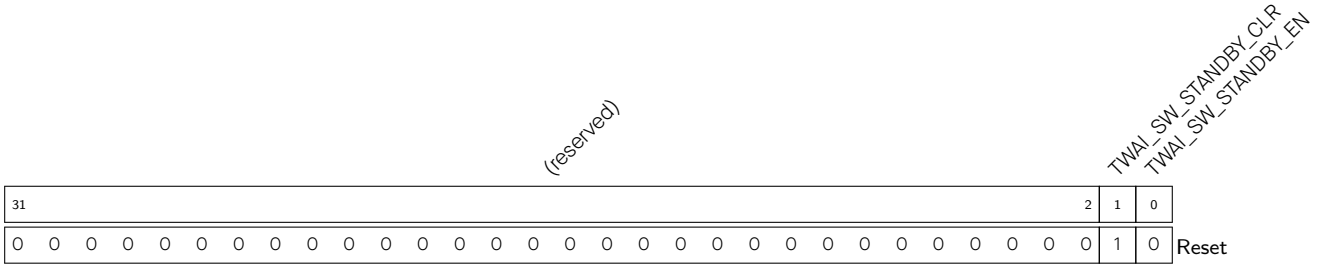
TWAI_CLOCK_OFF Configures whether or not to enable the external CLKOUT pin in Reset mode.

0: Enable the external CLKOUT pin

1: Disable the external CLKOUT pin

(R/W)

Register 56.7. TWAI_SW_STANDBY_CFG_REG (0x0080)



TWAI_SW_STANDBY_EN Configures whether to set standby signals with software.

0: No effect

1: Set standby signals

(R/W)

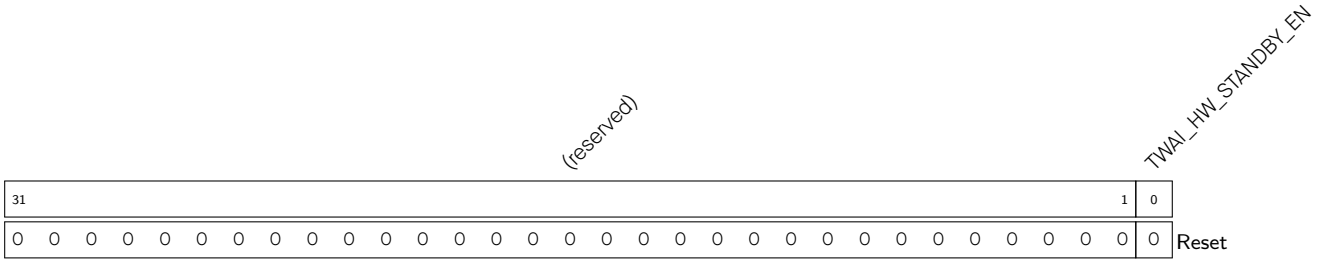
TWAI_SW_STANDBY_CLR Configures whether to clear standby signals with software.

0: No effect

1: Clear standby signals

(R/W)

Register 56.8. TWAI_HW_CFG_REG (0x0084)



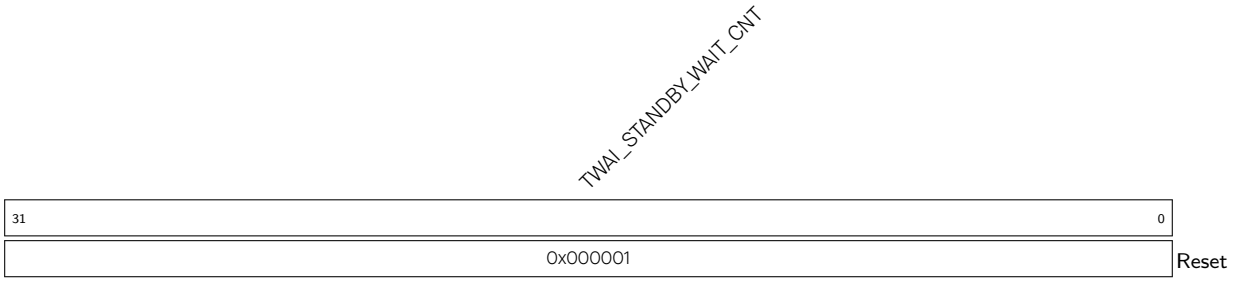
TWAI_HW_STANDBY_EN Configures whether to enable the automatic standby function for hardware.

0: No effect

1: Enable the automatic standby function for hardware

(R/W)

Register 56.9. TWAI_HW_STANDBY_CNT_REG (0x0088)

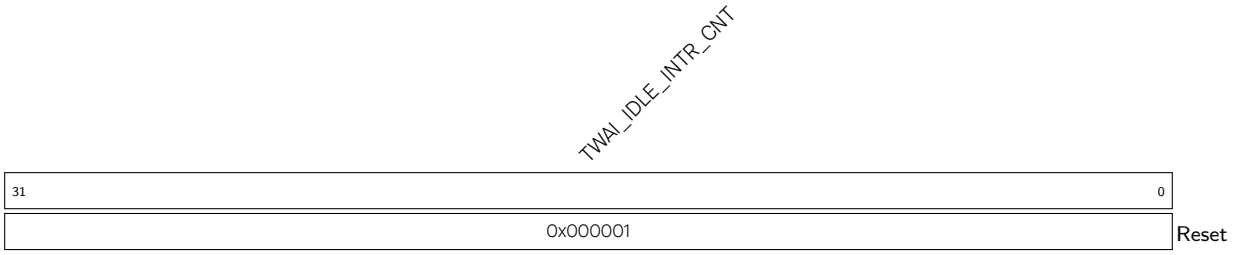


TWAI_STANDBY_WAIT_CNT Configures the time required before hardware triggers the standby signal after entering idle status.

Measurement unit: TWAI controller clock cycles.

(R/W)

Register 56.10. TWAI_IDLE_INTR_CNT_REG (0x008C)

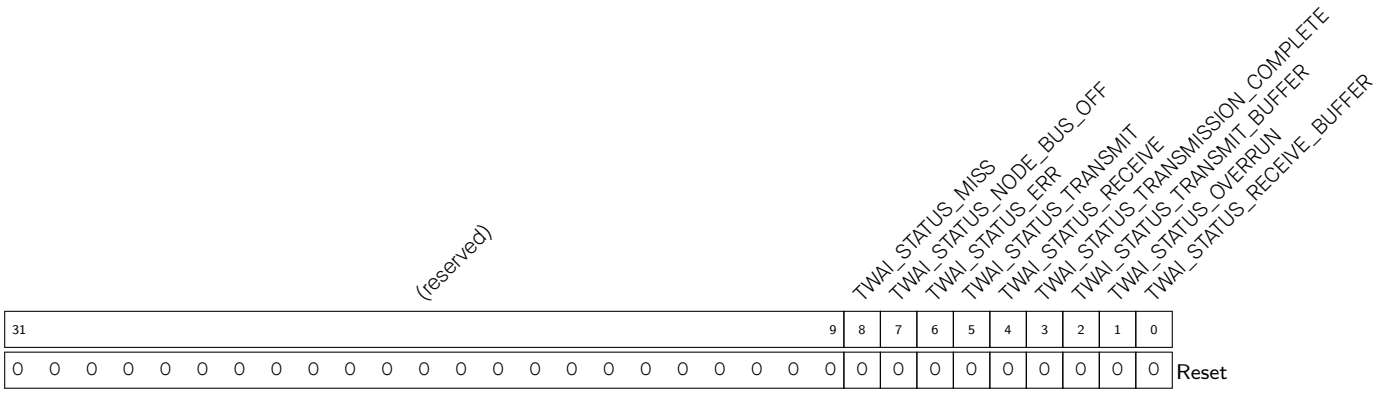


TWAI_IDLE_INTR_CNT Configures the time required before hardware generates the bus idle status interrupt signal after entering idle status.

Measurement unit: TWAI controller clock cycles.

(R/W)

Register 56.11. TWAI_STATUS_REG (0x0008)



TWAI_STATUS_RECEIVE_BUFFER Represents whether or not the RX buffer is empty.

0: Empty

1: Not empty, with at least one received data packet.

(RO)

TWAI_STATUS_OVERRUN Represents whether or not the RX FIFO is full.

0: Not full

1: Full, and data overrun has occurred

(RO)

TWAI_STATUS_TRANSMIT_BUFFER Represents whether or not the TX buffer is empty.

0: Not empty

1: Empty, and the CPU may write a message into it

(RO)

TWAI_STATUS_TRANSMISSION_COMPLETE Represents whether or not the TWAI controller has sent an entire packet to the bus.

0: Not sent

1: Sent

(RO)

TWAI_STATUS_RECEIVE Represents whether or not the TWAI Controller is receiving a message from the bus.

0: Not receiving

1: Receiving

(RO)

TWAI_STATUS_TRANSMIT Represents whether or not the TWAI Controller is transmitting a message to the bus.

0: Not transmitting

1: Transmitting

(RO)

TWAI_STATUS_ERR Represents at least one of the RX/TX error counter has reached or exceeded the value set in register TWAI_ERR_WARNING_LIMIT_REG. (RO)

Continued on the next page...

Register 56.11. LP_UART_INT_RAW_REG (0x0004)

Continued from the previous page...

TWAI_STATUS_NODE_BUS_OFF Represents whether or not the TWAI Controller involves in bus activities in bus-off status.

0: Involved

1: No longer involved

(RO)

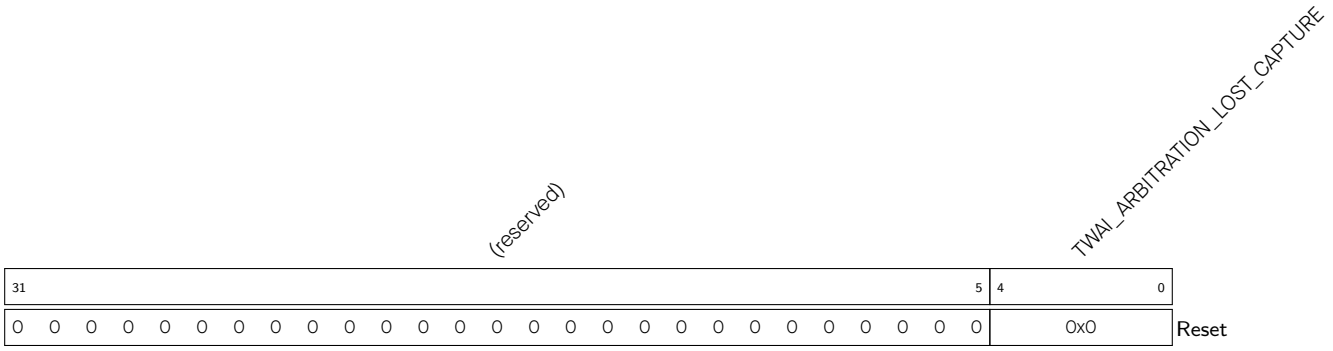
TWAI_STATUS_MISS Represents whether or not the data packet in the RX FIFO is complete.

0: The current packet is complete

1: The current packet is missing

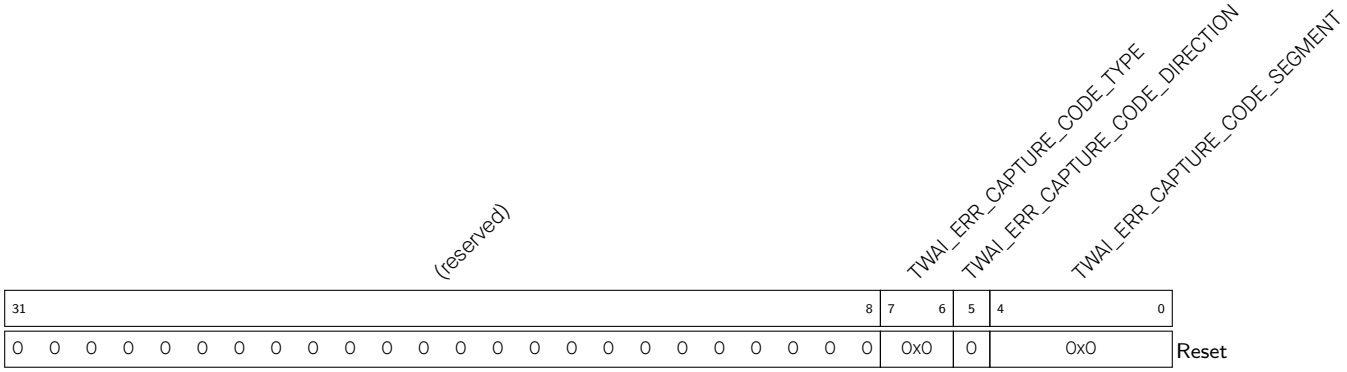
(RO)

Register 56.12. TWAI_ARB_LOST_CAP_REG (0x002C)



TWAI_ARBITRATION_LOST_CAPTURE Represents the bit position of lost arbitration. (RO)

Register 56.13. TWAI_ERR_CODE_CAP_REG (0x0030)



TWAI_ERR_CAPTURE_CODE_SEGMENT Represents the location of errors, see Table 56.4-11 for details. (RO)

TWAI_ERR_CAPTURE_CODE_DIRECTION Represents transmission direction of the node when an error occurs.

0: Error occurs when transmitting a message

1: Error occurs when receiving a message

(RO)

TWAI_ERR_CAPTURE_CODE_TYPE Represents error types.

0: Bit error

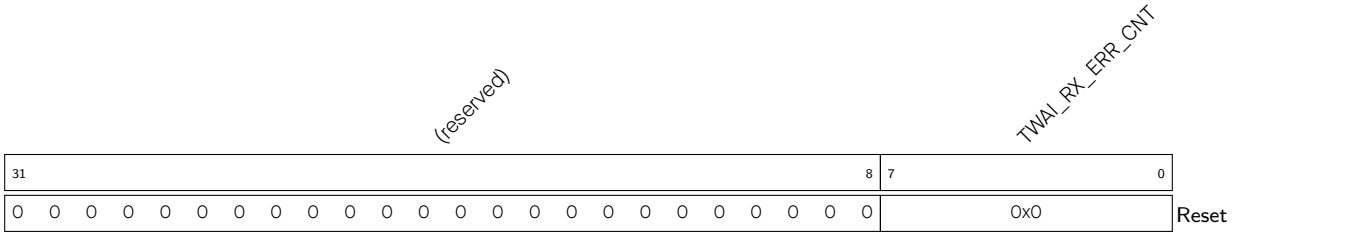
1: Form error

2: Stuff error

3: Others

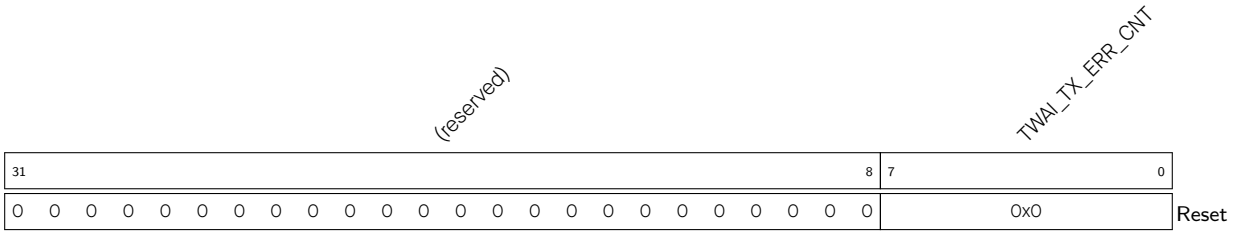
(RO)

Register 56.14. TWAI_RX_ERR_CNT_REG (0x0038)



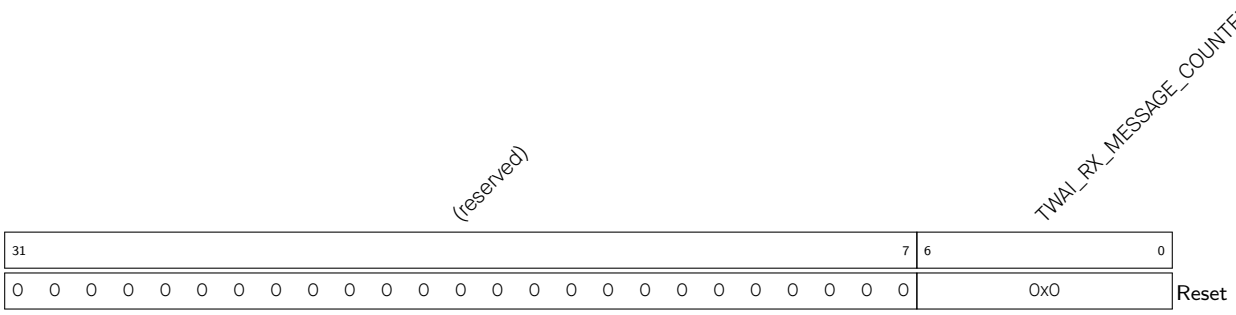
TWAI_RX_ERR_CNT The RX error counter register, which reflects value changes in reception status. (R/W)

Register 56.15. TWAI_TX_ERR_CNT_REG (0x003C)



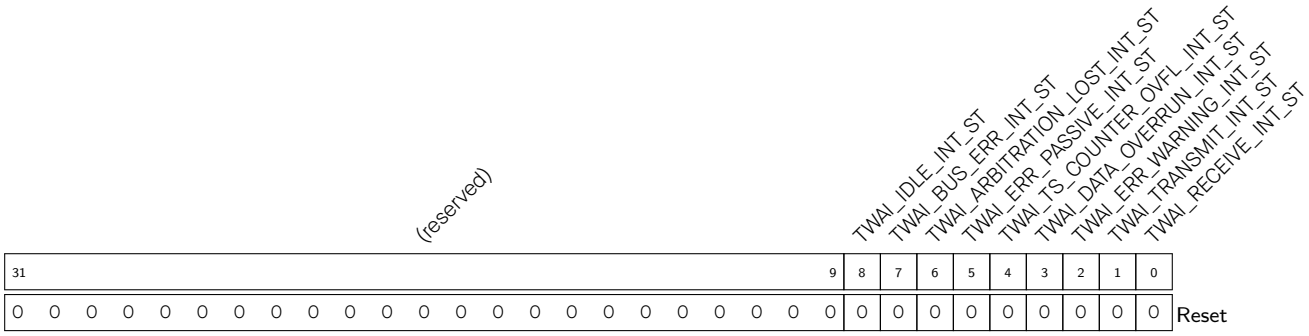
TWAI_TX_ERR_CNT The TX error counter register, which reflects value changes in transmission status. (R/W)

Register 56.16. TWAI_RX_MESSAGE_COUNTER_REG (0x0074)



TWAI_RX_MESSAGE_COUNTER Represents the number of messages available within the RX FIFO. (RO)

Register 56.17. TWAI_INTERRUPT_REG (0x000C)



TWAI_RECEIVE_INT_ST The masked interrupt status of TWAI_RX_INT. (RO)

TWAI_TRANSMIT_INT_ST The masked interrupt status of TWAI_TX_INT. (RO)

TWAI_ERR_WARNING_INT_ST The masked interrupt status of TWAI_ERR_WARN_INT. (RO)

TWAI_DATA_OVERRUN_INT_ST The masked interrupt status of TWAI_OVERRUN_INT. (RO)

TWAI_TS_COUNTER_OVFL_INT_ST The masked interrupt status of TWAI_TS_COUNTER_OVFL_INT. (RO)

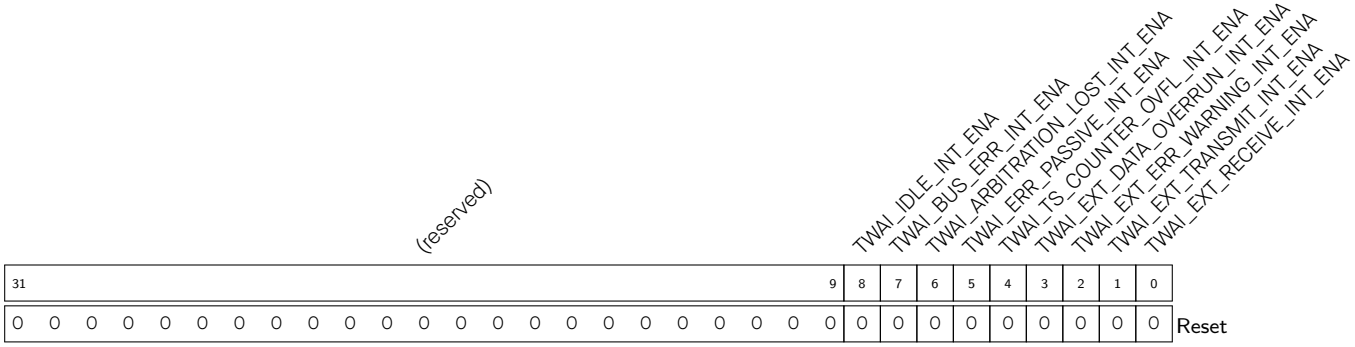
TWAI_ERR_PASSIVE_INT_ST The masked interrupt status of TWAI_ERR_PASSIVE_INT. (RO)

TWAI_ARBITRATION_LOST_INT_ST The masked interrupt status of TWAI_ARB_LOST_INT. (RO)

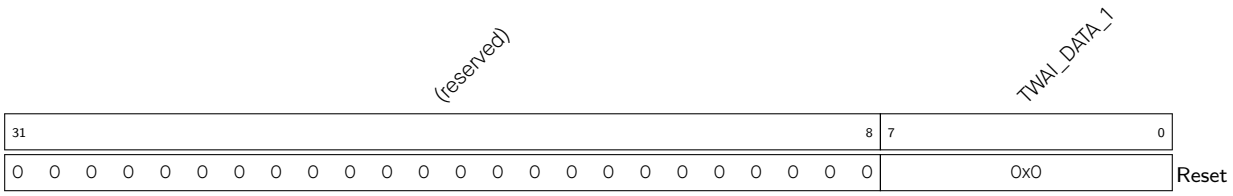
TWAI_BUS_ERR_INT_ST The masked interrupt status of TWAI_BUS_ERR_INT. (RO)

TWAI_IDLE_INT_ST The masked interrupt status of TWAI_BUS_STATE_INT. (RO)

Register 56.18. TWAI_INTERRUPT_ENABLE_REG (0x0010)

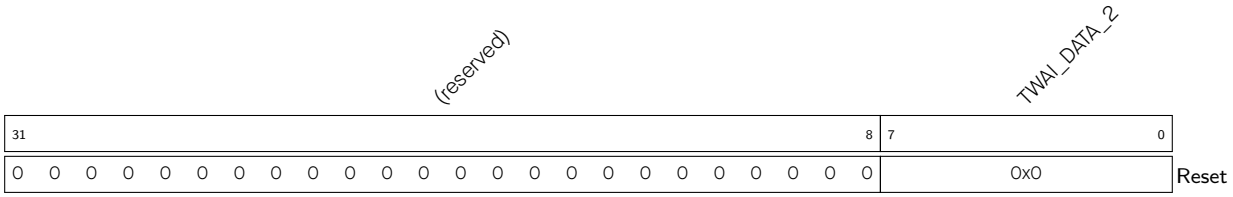

(reserved)
TWAI_IDLE_INT_ENA
TWAI_BUS_ERR_INT_ENA
TWAI_ARBITRATION_LOST_INT_ENA
TWAI_ERR_PASSIVE_INT_ENA
TWAI_TS_COUNTER_OVFL_INT_ENA
TWAI_EXT_DATA_OVERRUN_INT_ENA
TWAI_EXT_ERR_WARNING_INT_ENA
TWAI_EXT_TRANSMIT_INT_ENA
TWAI_EXT_RECEIVE_INT_ENA
31 9 8 7 6 5 4 3 2 1 0
0 0 0

Register 56.20. TWAI_DATA_1_REG (0x0044)



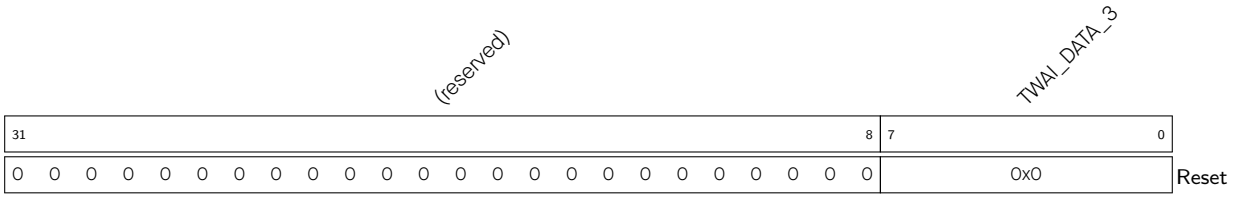
TWAI_DATA_1 In Operation mode, configures the 1st byte information of the data to be transmitted, or reads the 1st byte information of the data received. In Reset mode, configures the 2nd byte of the filter code. (R/W)

Register 56.21. TWAI_DATA_2_REG (0x0048)



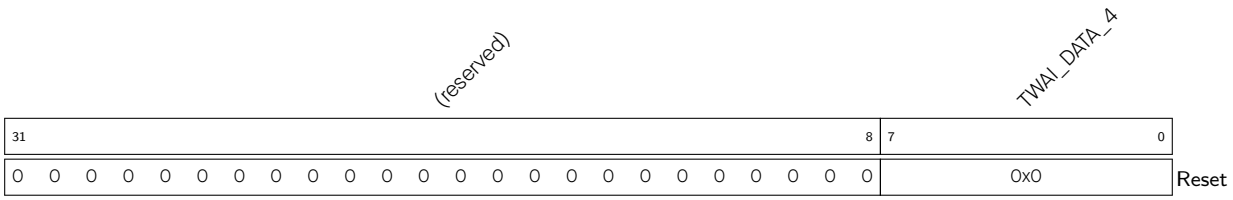
TWAI_DATA_2 In Operation mode, configures the 2nd byte information of the data to be transmitted, or reads the 2nd byte information of the data received. In Reset mode, configures the 3rd byte of the filter code. (R/W)

Register 56.22. TWAI_DATA_3_REG (0x004C)



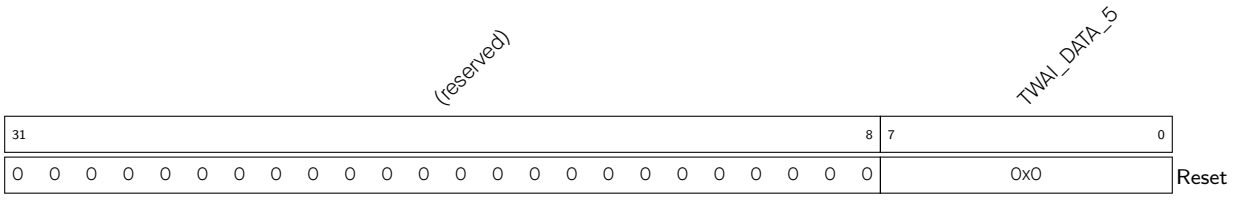
TWAI_DATA_3 In Operation mode, configures the 3rd byte information of the data to be transmitted, or reads the 3rd byte information of the data received. In Reset mode, configures the 4th byte of the filter code. (R/W)

Register 56.23. TWAI_DATA_4_REG (0x0050)



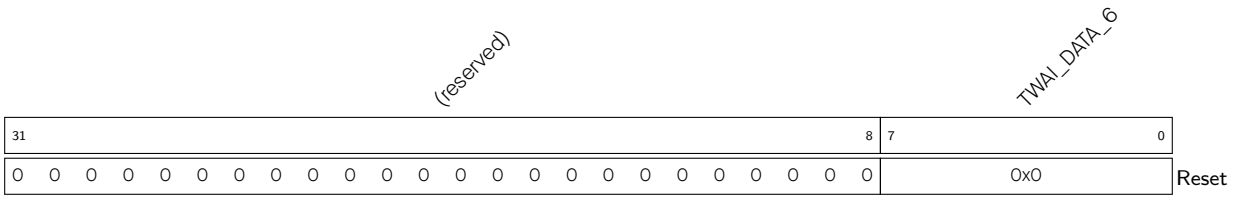
TWAI_DATA_4 In Operation mode, configures the 4th byte information of the data to be transmitted, or reads the 4th byte information of the data received. In Reset mode, configures the 1st byte of the masked filter code. (R/W)

Register 56.24. TWAI_DATA_5_REG (0x0054)



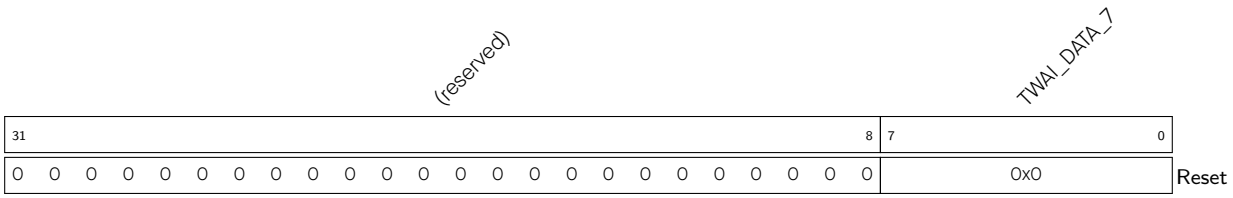
TWAI_DATA_5 In Operation mode, configures the 5th byte information of the data to be transmitted, or reads the 5th byte information of the data received. In Reset mode, configures the 2nd byte of the masked filter code. (R/W)

Register 56.25. TWAI_DATA_6_REG (0x0058)



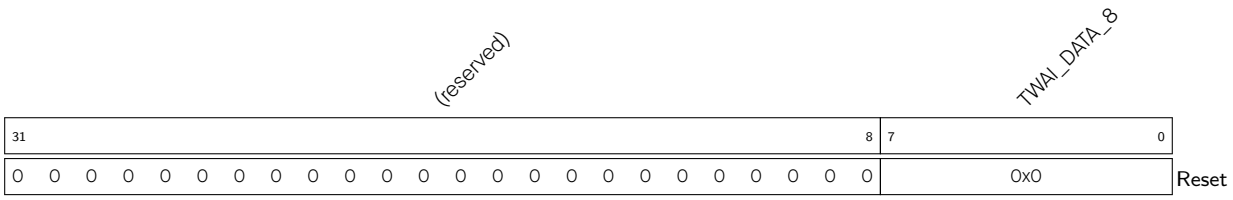
TWAI_DATA_6 In Operation mode, configures the 6th byte information of the data to be transmitted, or reads the 6th byte information of the data received. In Reset mode, configures the 3rd byte of the masked filter code. (R/W)

Register 56.26. TWAI_DATA_7_REG (0x005C)



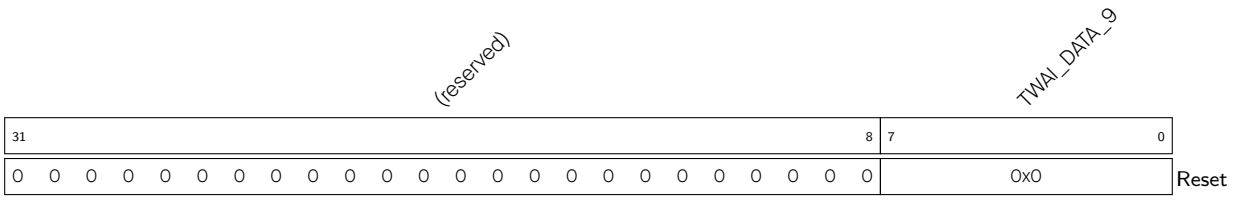
TWAI_DATA_7 In Operation mode, configures the 7th byte information of the data to be transmitted, or reads the 7th byte information of the data received. In Reset mode, configures the 4th byte of the masked filter code. (R/W)

Register 56.27. TWAI_DATA_8_REG (0x0060)



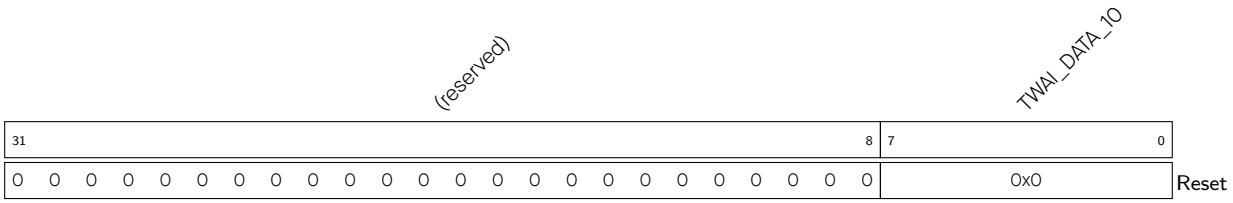
TWAI_DATA_8 In Operation mode, configures the 8th byte information of the data to be transmitted, or reads the 8th byte information of the data received. (R/W)
In Reset mode, reserved. (RO)

Register 56.28. TWAI_DATA_9_REG (0x0064)



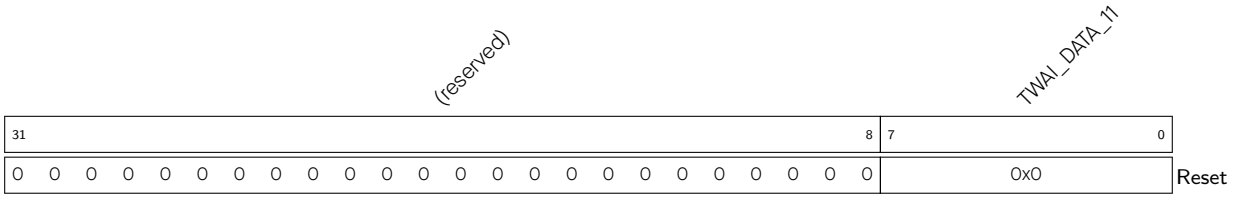
TWAI_DATA_9 In Operation mode, configures the 9th byte information of the data to be transmitted, or reads the 9th byte information of the data received. (R/W)
In Reset mode, reserved. (RO)

Register 56.29. TWAI_DATA_10_REG (0x0068)



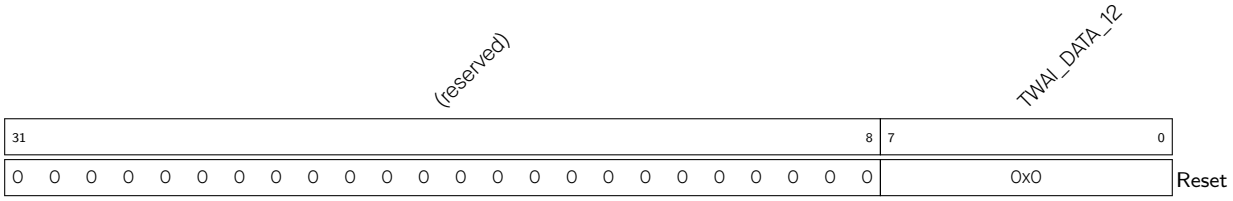
TWAI_DATA_10 In Operation mode, configures the 10th byte information of the data to be transmitted, or reads the 10th byte information of the data received. (R/W)
In Reset mode, reserved. (RO)

Register 56.30. TWAI_DATA_11_REG (0x006C)



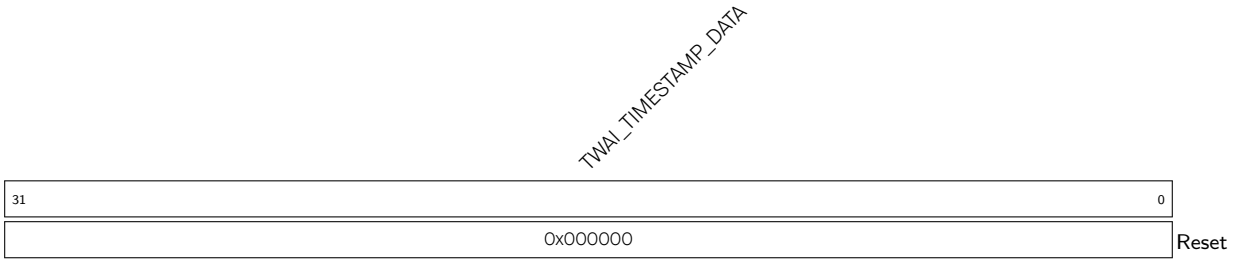
TWAI_DATA_11 In Operation mode, configures the 11th byte information of the data to be transmitted, or reads the 11th byte information of the data received. (R/W)
In Reset mode, reserved. (RO)

Register 56.31. TWAI_DATA_12_REG (0x0070)



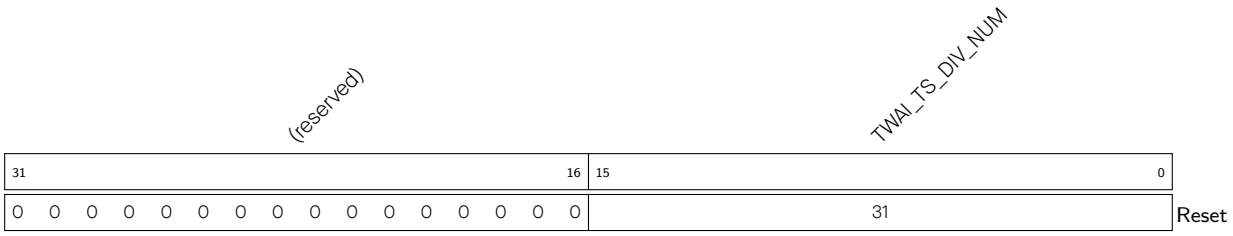
TWAI_DATA_12 In Operation mode, configures the 12th byte information of the data to be transmitted, or reads the 12th byte information of the data received. (R/W)
In Reset mode, reserved. (RO)

Register 56.32. TWAI_TIMESTAMP_DATA_REG (0x0094)



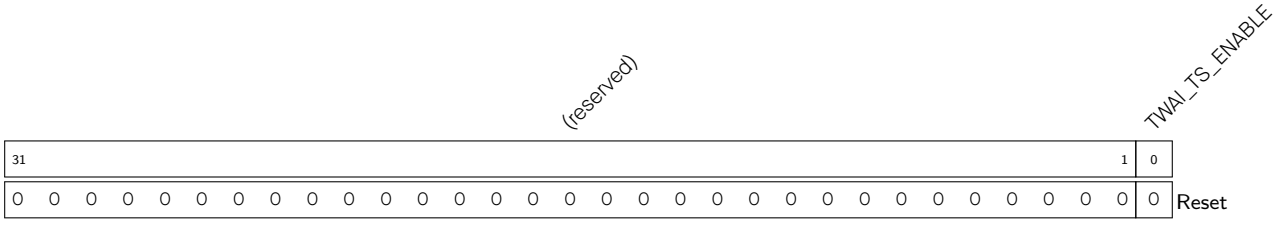
TWAI_TIMESTAMP_DATA The timestamp of the TWAI frame indicated by the RX FIFO read pointer corresponds to the time when the current TWAI frame was received. Valid only when TWAI_TS_ENABLE is enabled. (RO)

Register 56.33. TWAI_TIMESTAMP_PRESCALER_REG (0x0098)



TWAI_TS_DIV_NUM Configures the clock divisor of the timestamp counter. (R/W)

Register 56.34. TWAI_TIMESTAMP_CFG_REG (0x009C)



TWAI_TS_ENABLE Configures whether to enable the timestamp collection function.

0: Disable

1: Enable

(R/W)

Chapter 57

SD/MMC Host Controller (SDHOST)

57.1 Overview

The ESP32-P4 memory card interface controller provides a hardware interface between the Advanced Peripheral Bus (APB) and an external memory device. The memory card interface allows the ESP32-P4 to be connected to SDIO (secure digital I/O) memory cards, MMC (multimedia cards) and devices with a CE-ATA (Consumer Electronics Advanced Transport Architecture) interface. It supports two external cards (Card0 and Card1). All SD/MMC module interface signals only connect to GPIO pins via GPIO matrix.

57.2 Features

This module supports the following features:

- Two external cards
- SD memory Card specification V3.0 and V3.01, with a maximum transfer rate of DDR50
- MMC: V4.41, V4.5, and V4.51, with a maximum transfer rate of DDR50
- CE-ATA: V1.1
- 1-bit, 4-bit, and 8-bit modes

The SD/MMC controller topology is shown in Figure 57.2-1. The controller supports two peripherals, but they cannot function at the same time.

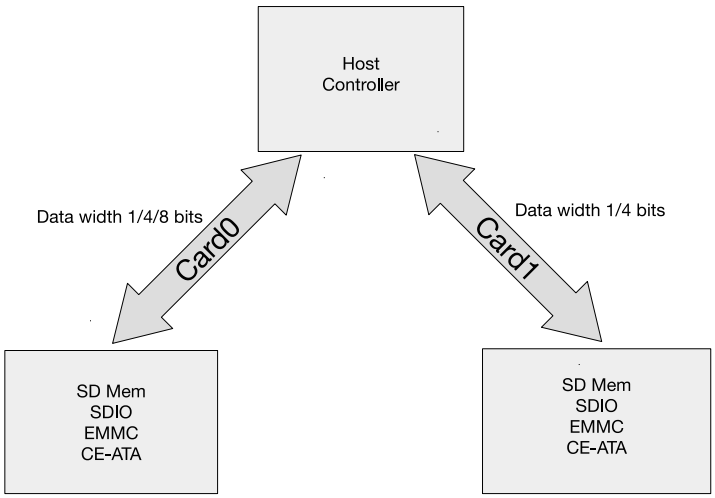


Figure 57.2-1. SD/MMC Controller Topology

57.3 SD/MMC External Interface Signals

The primary external interface signals, which enable the SD/MMC controller to communicate with an external device, are clock (sdhost_cclk_out_1.eg:card1), command (sdhost_ccmd_out_1), and data signals (sdhost_cdata_in_1[7:0]/sdhost_cdata_out_1[7:0]). Additional signals include the card interrupt, card detect, and write protect signals. The direction of each signal is shown in Figure 57.3-1. The direction and description of each pin are listed in Table 57.3-1.

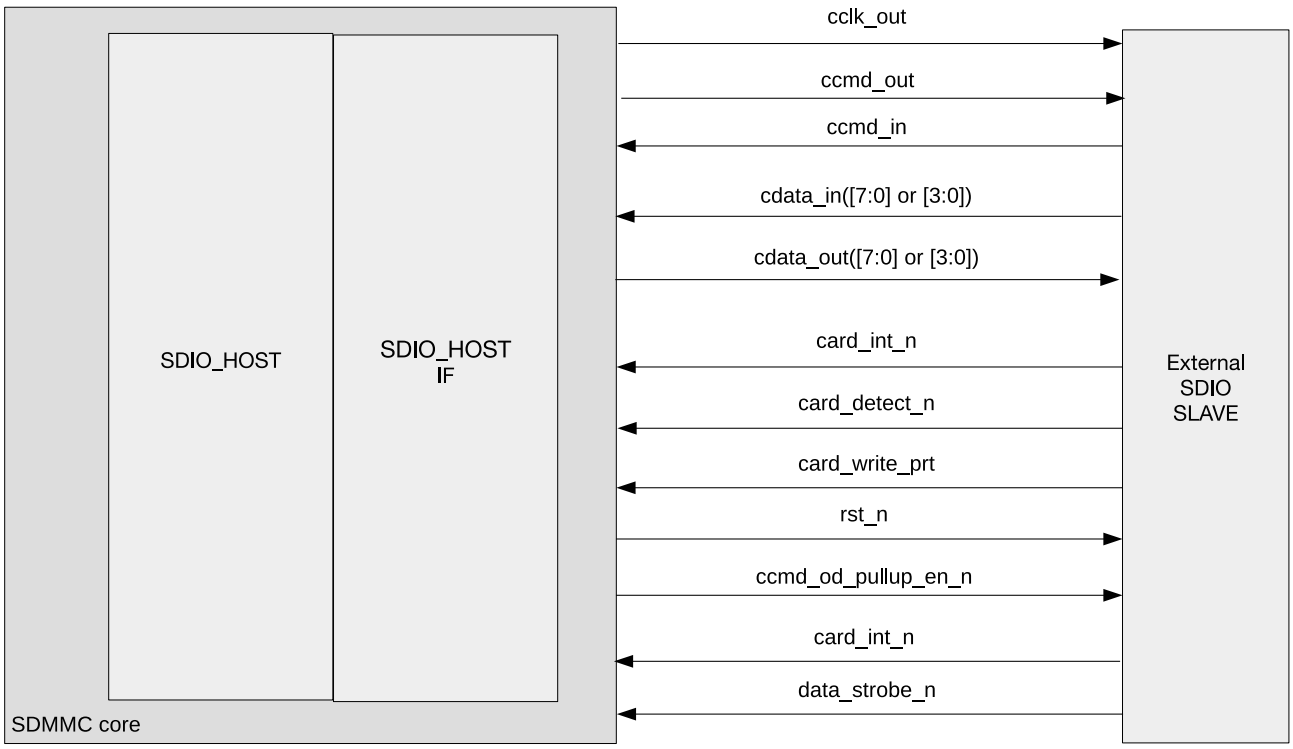


Figure 57.3-1. SD/MMC Controller External Interface Signals

Table 57.3-1. SD/MMC Signal Description

Signal	Direction	Description
sdhost_cclk_out	Output	Clock signals for slave device
sdhost_ccmd	Duplex	Duplex command/response lines
sdhost_cdata	Duplex	Duplex data read/write lines
sdhost_card_detect_n	Input	Card detection input line
sdhost_card_write_prt	Input	Card write protection status input

57.4 Functional Description

57.4.1 SD/MMC Host Controller Architecture

The SD/MMC host controller consists of two main functional blocks, as shown in Figure 57.4-1:

- Bus Interface Unit (BIU): Provides the APB interface for registers, access to RAM data, and DMA data read and write operations.

- Card Interface Unit (CIU): Handles external memory card interface protocols and provides clock control.

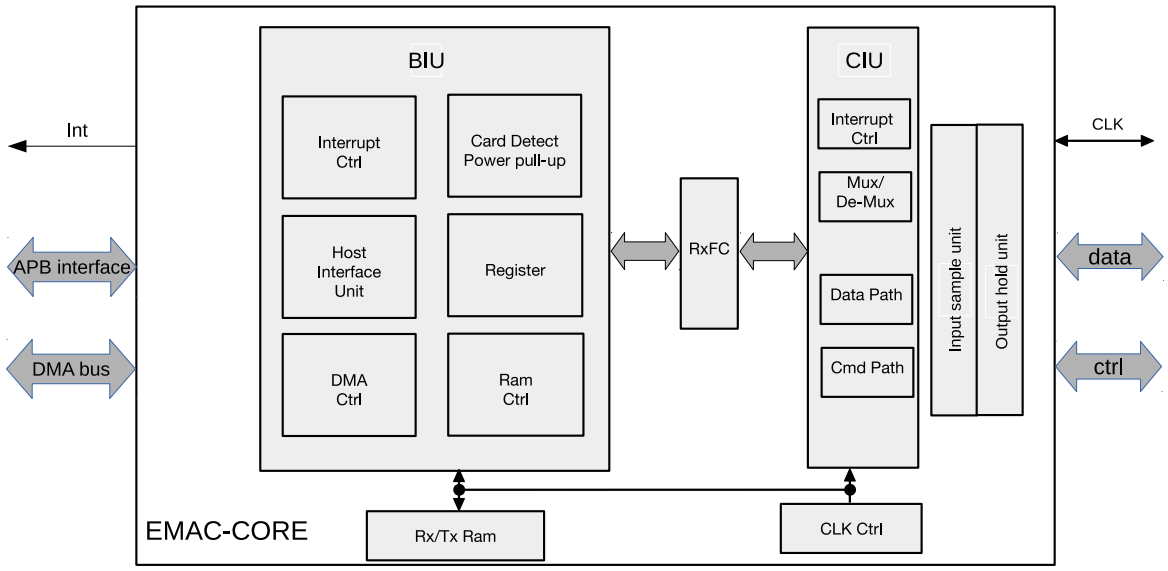


Figure 57.4-1. SDIO Host Block Diagram

57.4.1.1 Bus Interface Unit (BIU)

The BIU provides access to registers and RAM data through the Host Interface Unit (HIU). Additionally, it provides a method to access to memory data through a DMA interface. Figure 57.4-1 illustrates the internal components of the BIU.

The BIU provides the following functions:

- Host interface
- DMA interface
- Interrupt control
- Register access
- FIFO access
- Power-up/pull-up control and card detection

57.4.1.2 Card Interface Unit (CIU)

The CIU module implements the dedicated card slot specifications. Within the CIU, the command path control unit and data path control unit are used to interface with the command and data ports, respectively, of the SD/MMC/CE-ATA cards. The CIU also provides clock control. Figure 57.4-1 illustrates the internal structure of the CIU, which consists of the following primary functional blocks:

- Command path
- Data path
- SDIO interrupt control
- Clock control
- MUX (multiplexer) and De-MUX (de-multiplexer) unit

57.4.2 Command Path

The command path performs the following functions:

- Configures clock parameters
- Configures card command parameters
- Sends commands to card bus (sdhost_ccmd_out line)
- Receives responses from card bus (sdhost_ccmd_in line)
- Sends responses to BIU
- Drives the P-bit on the command line

The command path state machine is shown in Figure 57.4-2.

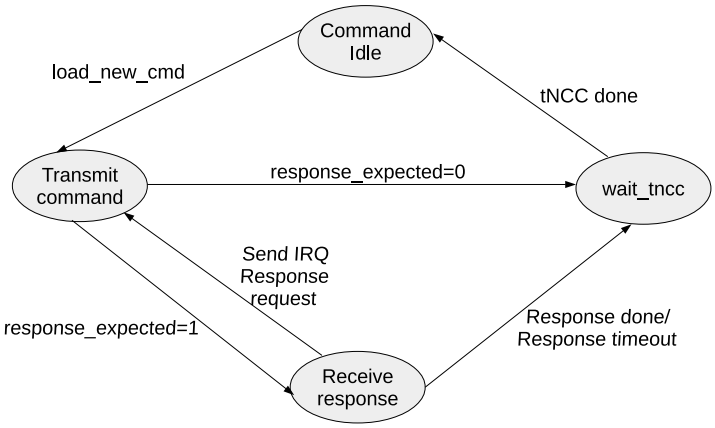


Figure 57.4-2. Command Path State Machine

57.4.3 Data Path

The data path block pops RAM data and transmits them on sdhost_cdata_out during a write-data transfer, or it receives data on sdhost_cdata_in and pushes them into RAM during a read-data transfer. The data path loads new data parameters, i.e., expected data, read/write data transfer, stream/block transfer, block size, byte count, card type, timeout registers, etc., whenever a data transfer command is not in progress.

If the SDHOST_DATA_EXPECTED bit is set in the SDHOST_CMD_REG register, the new command is a data transfer command and the data path starts one of the following operations:

- Transmitting data if the SDHOST_READ_WRITE bit is 1
- Receiving data if the SDHOST_READ_WRITE bit is 0

57.4.3.1 Data Transmit

The module starts data transmission two clock cycles after a response for the data write command is received. This occurs even if the command path detects a response error or a cyclic redundancy check (CRC) error in a response. If no response is received from the card until the response timeout, no data will be transmitted. Depending on the value of the SDHOST_TRANSFER_MODE bit in the SDHOST_CMD_REG register, the data transmit state machine adds data to the card's data bus in a stream or in block(s). The data transmit state machine is shown in Figure 57.4-3.

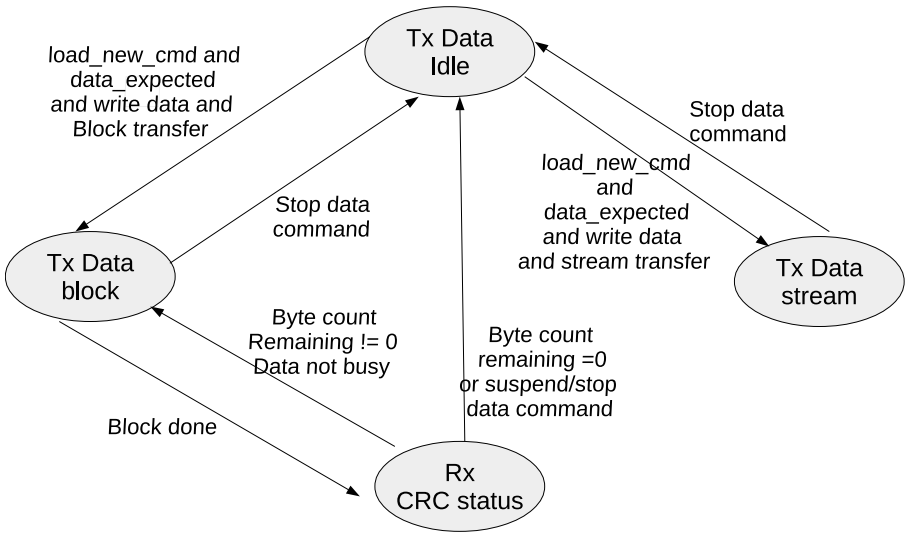


Figure 57.4-3. Data Transmit State Machine

57.4.3.2 Data Receive

The module receives data two clock cycles after the end bit of a data read command, even if the command path detects a response error or a CRC error. If no response is received from the card and a response timeout occurs, the BIU does not receive a signal about the completion of the data transfer. If the command sent by the CIU is an illegal operation for the card, it would prevent the card from starting a read-data transfer, and the BIU will not receive a signal about the completion of the data transfer.

If no data is received by the data timeout, the data path signals a data timeout to the BIU, which marks an end to the data transfer. Based on the value of the SDHOST_TRANSFER_MODE bit in the SDHOST_CMD_REG register, the data receive state machine gets data from the card's data bus in a stream or block(s). The data receive state machine is shown in Figure 57.4-4.

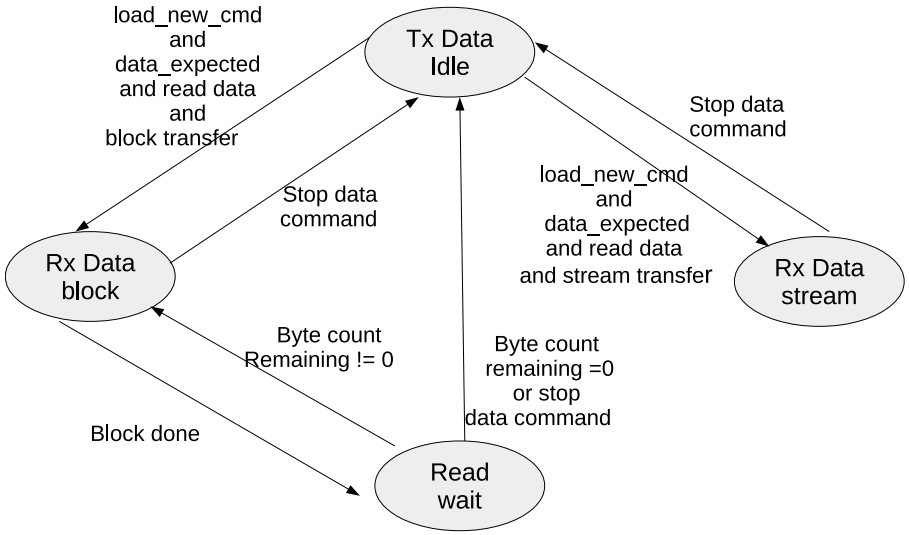


Figure 57.4-4. Data Receive State Machine

57.5 Software Restrictions for CIU Operations

- Only one card at a time can be selected to execute a command or data transfer. For example, when data are being transferred to or from a card, a new command must not be issued to another card. A new command, however, can be issued to the same card, allowing it to read the device status or stop the transfer.
- Only one data transfer command can be issued at a time.
- During an open-ended card write operation, if the card clock is stopped because RAM is empty, the software must first fill RAM with data, and start the card clock. Only then can it issue a stop/abort command to the card.
- During an SDIO/Combo card transfer, if the card function is suspended and the software wants to resume the suspended transfer, it must first reset and then release RAM by configuring the SDHOST_FIFO_RESET bit, and then issue the resume command as if it were a new data transfer command.
- When issuing card reset commands (CMD0, CMD15 or CMD52_reset) while a card data transfer is in progress, the software must set the SDHOST_STOP_ABORT_CMD bit in the SDHOST_CMD_REG register, so that the CIU can stop the data transfer after issuing the card reset command.
- When the controlling bit for end bit error is set in the SDHOST_RINTSTS_REG register, the CIU cannot control SDIO interrupts. In such a case, the software must ignore SDIO interrupts and issue a stop/abort command to the card, so that the card stops transmitting data.
- If the card clock is stopped because RAM is full during a card read, the software must read at least two RAM addresses to restart the card clock.
- Only one CE-ATA device at a time can be selected for a command or data transfer. For example, when data are transferred from a CE-ATA device, a new command must not be sent to another CE-ATA device.
- If the CE-ATA device interrupts are enabled (nIEN=0), a new SDHOST_RW_BLK command must not be sent to the same device if the execution of a SDHOST_RW_BLK command is already in progress. Only the Command Completion Signal Disable (CCSD) command can be sent while waiting for the Command Completion Signal (CCS).
- If, however, the CE-ATA device interrupts are disabled (nIEN=1), a new command can be issued to the same device, allowing it to read status information.
- Open-ended transfers are not supported for CE-ATA devices.
- The sdhost_send_auto_stop signal is not supported (software must not set the SDHOST_SEND_AUTO_STOP bit) for CE-ATA transfers.

After configuring the command start bit to 1, the values of the following registers cannot be changed before a command has been issued:

- CMD - command
- CMDARG - command argument
- BYTCNT - byte count
- BLKSIZ - block size
- CLKDIV - clock divider

- CKLENA - clock enable
- CLKSRC - clock source
- TMOUT - timeout
- CTYPE - card type

57.6 RAM for Receiving and Transmitting Data

The submodule RAM is a buffer area for transmitting and receiving data. It can be divided into two units: one is for transmitting data, and the other is for receiving data. The process of transmitting and receiving data can be achieved by the CPU and DMA for reading and writing. The latter method is described in detail in Section 57.8.

57.6.1 TX RAM Module

There are two ways to enable a write operation: CPU and DMA write.

For CPU write operations to TX RAM, write the data directly into the SDHOST_BUFFIFO_REG register via the APB interface. The corresponding FIFO width is 32 bits and the depth is 512.

For DMA write operations to TX RAM, set up DMA linked list descriptors as described in Section 57.8.

57.6.2 RX RAM Module

There are two ways to enable a read operation: CPU and DMA read.

For CPU read operations from RX RAM, read the data directly from the SDHOST_BUFFIFO_REG register via the APB interface. The corresponding FIFO width is 32 bits and the depth is 512.

For DMA read operations from RX RAM, set up DMA linked list descriptors as described in Section 57.8.

57.7 DMA Linked List

Each unit in the linked list structure consists of two parts: a descriptor and a data buffer. In other words, each descriptor points to a unique data buffer and to the next descriptor in the linked list. Figure 57.7-1 shows the linked list.

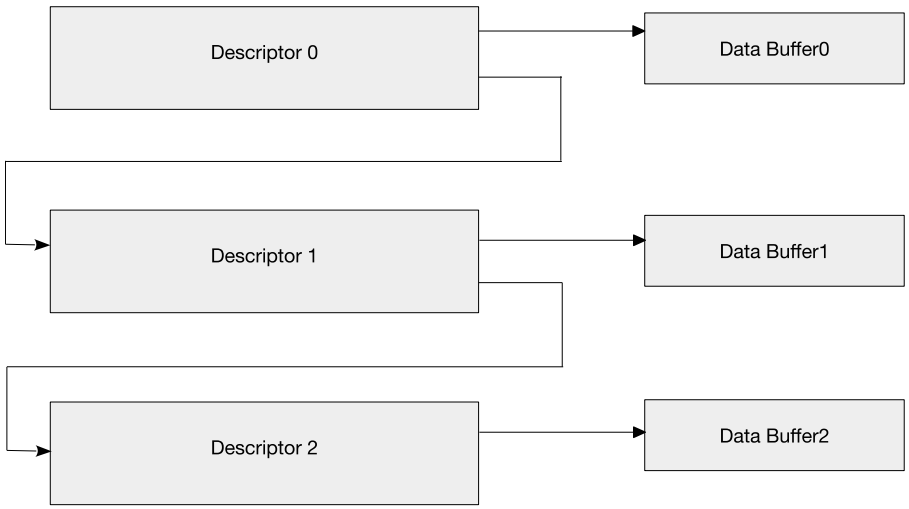


Figure 57.7-1. Linked List Structure

57.8 DMA Descriptor Format

Each descriptor consists of four words as shown in 57.8-1. Table 57.8-1, Table 57.8-2, Table 57.8-3, Table 57.8-4 provide the descriptions of these four words.

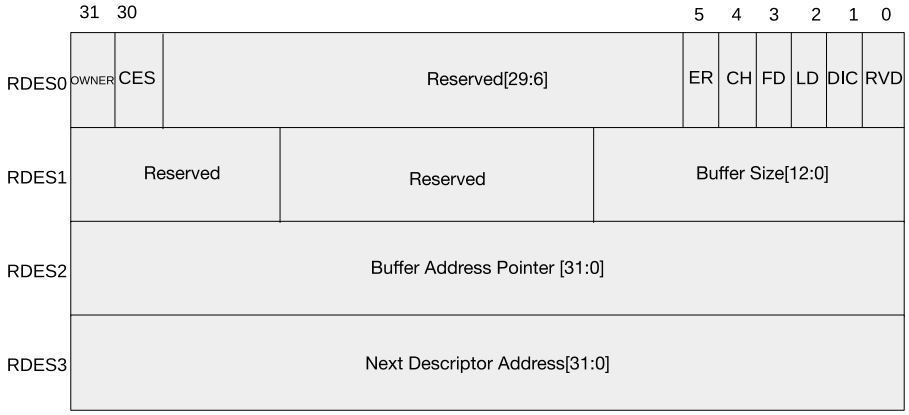


Figure 57.8-1. Descriptor Format

The DES0 field contains control and status information.

Table 57.8-1. DES0 Descriptor Field

Bits	Name	Description
31	OWNER	When set to 1, this bit indicates that the descriptor is owned by the DMA Controller. When reset to 0, it indicates that the descriptor is owned by the host. The DMA clears this bit when it completes the data transfer.

Bits	Name	Description
30	CES (Card Error Summary)	This bit indicates the status of the card's read/write operations. This bit is the logical OR of the following bits in the SDHOST_RINTSTS_REG register. • EBE: End Bit Error • RTO: Response Timeout • RCRC: Response CRC • SBE: Start Bit Error • DRTO: Data Read Timeout • DCRC: Data CRC for Receive • RE: Response Error
29:6	Reserved	Reserved
5	ER (End of Ring)	When set to 1, this bit indicates that the linked list has reached its final descriptor. The DMA Controller then returns to the base address of the linked list, creating a linked list ring.
4	CH (Second Address Chained)	When set to 1, this bit indicates that the second address in the descriptor is the next descriptor address. When this bit is set to 1, BS2 (DES1[25:13]) must be all zeros.
3	FD (First Descriptor)	When set to 1, this bit indicates that this descriptor contains the first buffer of the data. If the size of the first buffer is 0, the next descriptor contains the beginning of the data.
2	LD (Last Descriptor)	This bit is associated with the last data block of a DMA transfer. When set to 1, the bit indicates that the buffers pointed by this descriptor are the last buffers of the data. After this descriptor is completed, the remaining byte count is 0. In other words, after the descriptor with the LD bit set is completed, the remaining byte count should be 0.
1	DIC (Disable Interrupt on Completion)	When set to 1, this bit prevents the setting of the TI/RI bit of the DMA Status Register (SDHOST_IDSTS_REG) for the data that ends in the buffer pointed by this descriptor.
0	Reserved	Reserved

The DES1 field contains the buffer size.

Table 57.8-2. DES1 Descriptor Field

Bits	Name	Description
31:26	Reserved	Reserved
25:13	Reserved	Reserved

Bits	Name	Description
12:0	BS (Buffer Size)	These bits indicate the size of the data buffer byte size, which must be a multiple of four. When the buffer size is not a multiple of four, the resulting behavior is undefined. This field must not be zero.

The DES2 field contains the address pointer to the data buffer.

Table 57.8-3. DES2 Descriptor Field

Bits	Name	Description
31:0	Buffer Address Pointer	These bits indicate the address of the data buffer. The buffer address must be word-aligned.

The DES3 field contains the address pointer to the next descriptor if the present descriptor is not the last one in a linked list structure.

Table 57.8-4. DES3 Descriptor Field

Bits	Name	Description
31:0	Next Descriptor Address	If CH (DES0[4]) is set to 1, these bits contains the pointer to the next descriptor. If this is not the last descriptor in a linked list structure, bits 1 and 0 must be zero, i.e., DES3[1:0] = 0.

57.9 Programming Procedures

57.9.1 Initializing Registers

To initialize registers, perform the following steps:

1. Write to the control register SDHOST_CTRL_REG, the raw interrupt register SDHOST_RINTSTS_REG, and SDIO interrupt mask register SDHOST_INTMASK_REG to clear pending interrupts and configure controlling and interrupt parameters.
2. Write clock divider configuration register SDHOST_CLKDIV_REG, clock source selection register SDHOST_CLKSRC_REG, and clock enable register SDHOST_CLKENA_REG to configure the card clock.
3. Write other configuration registers based on card parameters. For example, configure
   - card bus width (SDHOST_CTYPE_REG)
   - user ID (SDHOST_USRID_REG)
   - UHS mode voltage and DDR (SDHOST_UHS_REG)
   - EMMC mode start bit (SDHOST_EMMCDDR_REG)
   - SDIO mode (SDHOST_CLK_EDGE_SEL_REG)
   - timeout value (SDHOST_TMOUT_REG)

- debounce filter (SDHOST_DEBNCE_REG)
- internal phase shift (SDHOST_ENSHIFT_REG)

You can reset the card with SDHOST_RST_N_REG.

57.9.2 Sending Commands

To send commands, perform the following steps:

1. Write the command argument register SDHOST_CMDARG_REG with the appropriate command argument parameter.
2. For data transfer, write the data size register SDHOST_BYTCNT_REG, the block size register SDHOST_BLKSIZ_REG register, and the card threshold control register SDHOST_CARDTHRCTL_REG to configure data transfer parameters.
3. Data transfer can be achieved by DMA or FIFO, and DMA is preferable.
   - To transmit and receive data via DMA, configure the DMA Controller according to Section 57.9.3, Section 57.9.4, and Section 57.9.5.
   - To transmit and receive data via FIFO, configure the FIFO threshold via FIFO configuration register SDHOST_FIFOTH_REG. For writing to a card, software should write data to the FIFO via SDHOST_BUFFIFO_REG continuously during command execution.
4. Configure and start the command via the SDHOST_CMD_REG.
5. Wait until the SDHOST_START_CMD bit in the SDHOST_CMD_REG register is cleared by hardware upon command reception. When hardware cannot receive the command, it will set the 12th bit HLE in the SDHOST_RINTSTS_REG register to 1.
6. Read the SDHOST_RINTSTS_REG register or enable interrupts to check command execution status. If needed, read the SDHOST_RESP*n*_REG (*n* ranges from 0 ~ 3) register to get command response from the card.
7. If data is read from the card via FIFO, software can get the data returned by the card via reading the SDHOST_BUFFIFO_REG register.

Please refer to the card specification for the command format, and configure register field values depending on the command format.

57.9.3 Initializing DMA

To initialize the DMA Controller, perform the following steps:

1. Write to the DMA bus mode register (SDHOST_BMOD_REG) to configure the host bus's access parameters.
2. Write to the DMA interrupt enable register (SDHOST_IDINTEN_REG) to mask unnecessary interrupt causes.
3. Create either a transmit or receive linked list. Then write the starting address of the first descriptor in the linked list to the DMA Controller's linked list base address register (SDHOST_DBADDR_REG).
4. The DMA Controller engine proceeds to get descriptors from the linked list.

57.9.4 Initializing DMA Transmission

To initialize DMA transmission, perform the following steps:

1. The host sets up the descriptor fields (DES0-DES3) for transmission, and sets the OWNER bit (DES0[31]) to 1. The host also prepares the data buffer.
2. The host writes the write data command to the command register SDHOST_CMD_REG.
3. The host sets the required transmit threshold via the SDHOST_TX_WMARK field in the SDHOST_FIFOTH_REG register.
4. The DMA Controller engine fetches the descriptor and checks the OWNER bit. If the OWNER bit is not set, it means that the host owns the descriptor. In this case, the DMA Controller enters a suspend state and asserts the Descriptor Unable interrupt via the SDHOST_IDSTS_REG register. The host then needs to release the DMA Controller by writing any value to SDHOST_PLDMND_REG.
5. The DMA Controller waits for the Command Done (CMDD) bit in the SDHOST_RINTSTS_REG register to be set to 1 with no errors from BIU, which indicates that a transfer has completed.
6. The DMA Controller engine waits for a DMA interface request from BIU. This request is generated based on the configured transmit threshold value. For the last byte of data that cannot be accessed using a burst, a single transfer is performed on the AHB bus.
7. The DMA Controller fetches the transmit data from the data buffer in the host memory and transfers them to RAM in preparation for transmission to the card.
8. When data span across multiple descriptors, the DMA Controller fetches the next descriptor and continues with its operation with the next descriptor. The Last Descriptor bit indicates whether the data span multiple descriptors or not.
9. When data transmission is complete, the status information is updated in the SDHOST_IDSTS_REG register by setting the SDHOST_IDSTS_TI bit to 1 if it has already been enabled. Also, the OWNER bit is cleared by the DMA Controller by updating the DES0 field.

57.9.5 Initializing DMA Reception

To initialize DMA reception, perform the following steps:

1. The host sets up the descriptor fields (DES0-DES3) for reception, and sets the OWNER bit (DES0[31]) to 1.
2. The host writes the read data command in the SDHOST_CMD_REG register in BIU.
3. The host sets the required receive threshold level via the SDHOST_RX_WMARK field in the SDHOST_FIFOTH_REG register.
4. The DMA Controller engine fetches the descriptor and checks the OWNER bit. If the OWNER bit is not set, it means that the host owns the descriptor. In this case, the DMA enters suspend state and asserts the Descriptor Unable interrupt via the SDHOST_IDSTS_REG register. The host then needs to release the DMA Controller by writing any value to SDHOST_PLDMND_REG.
5. The DMA Controller waits for the Command Done (CMDD) bit in the SDHOST_RINTSTS_REG register to be set to 1 with no errors from BIU, which indicates that a reception has been completed.

6. The DMA Controller engine waits for a DMA interface request from BIU. This request is generated based on the configured receive threshold value. For the last byte of data that cannot be accessed using a burst, a single transfer is performed on the AHB bus.
7. The DMA Controller fetches the data from RAM and transfers them to the host memory.
8. When data span across multiple descriptors, the DMA Controller fetches the next descriptor and continues with its operation with the next descriptor. The Last Descriptor bit indicates whether the data span multiple descriptors or not.
9. When data reception is complete, the status information is updated in the SDHOST_IDSTS_REG register by setting the SDHOST_IDSTS_RI bit to 1, if it has already been enabled. Also, the OWNER bit is cleared by the DMA Controller by updating the DESO field.

57.10 Clock Phase Selection

The phase of the SD/MMC Host Controller clock source is configurable, so as to adjust the setup time sequence for the input or output data signals.

The clock source of the SD/MMC Host Controller can be a high-performance clock at a high frequency or a low-power clock at a low frequency. When the clock source is a high-performance clock, the internal signal clock phase, output signal driving clock phase, and input signal sampling clock phase can be configured via the SDHOST_DLL_CLK_CONF_REG, in the unit of 1/64 clock source period. When the clock source is a low-power clock, the internal signal clock phase, output signal driving clock phase, and input signal sampling clock phase can be configured via the system clock register. The phase options are 0 degrees, 90 degrees, 180 degrees, and 270 degrees, or in other words the unit is 1/4 clock source period.

For details, please see Chapter 11 *Reset and Clock*.

57.11 Interrupt

The SD/MMC Host Controller can generate interrupt signal SDIO_HOST_INTR triggered by the following interrupt sources, and send the interrupt signals to the Interrupt Matrix.

- SDIO_INT: Interrupts from SDIO cards
- EBE: Triggered when an error in end bit occurs, or no data CRC received
- ACD: Triggered when an automatically sent command has been executed
- SBE/BCI: Triggered when an error in the start bit occurs
- HLE: Triggered when a write error occurs during hardware-lock period
- FRUN: Triggered when the FIFO is empty or full
- HTO: Triggered when the host does not fill data before timeout period
- DRTO: Triggered when a data read operation times out
- RTO: Triggered when card response times out
- DCRC: Triggered when data CRC error occurs
- RCRC: Triggered when a card response CRC error occurs

- RXDR: Triggered during read operations from card when FIFO level is greater than Receive-Threshold level
- TXDR: Triggered during write operations to card when FIFO level reaches less than or equal to Transmit-Threshold level.
- DTO: Triggered when data transfer completes
- CMDD: Triggered when a command has been executed
- RE: Triggered when response error occurs
- CD: Triggered when a card is detected
- IDSTS_RI: Triggered when data pointed by a DMA descriptor list have been received
- IDSTS_TI: Triggered when data pointed by a DMA descriptor list have been transmitted
- IDSTS_FBE: Triggered when a fatal bus error occurs and the host is stopped during data transmission or reception. When triggered, the DMA disables all bus access.
- IDSTS_DU: Triggered when a DMA descriptor list is not available because the OWNER bit is 0 (DES0 [31] = 0)

The SDHOST_RINTSTS_REG and SDHOST_IDSTS_REG register contains all the bits that might cause an interrupt. The SDHOST_INTMASK_REG and SDHOST_IDINTEN_REG register contains an enable bit for each of the events that can cause an interrupt.

There are three interrupt summary bits in the SDHOST_IDSTS_REG register: the card error interrupt summary bit (bit 5 SDHOST_IDSTS_CES), the normal interrupt summary bit (bit 8 SDHOST_IDSTS_NIS), and the abnormal interrupt summary bit (bit 9 SDHOST_IDSTS_AIS). Interrupts are cleared by writing 1 to the corresponding bit. When all the enabled interrupts within a group are cleared, the corresponding summary bit is also cleared.

Interrupts are not queued. If another interrupt event occurs before the driver has responded to the previous interrupt, no additional interrupts are generated. For example, the SDHOST_IDSTS_RI indicates that one or more data were transferred to the host buffer.

An interrupt is generated only once for simultaneous multiple events. The driver must scan the SDHOST_RINTSTS_REG and SDHOST_IDSTS_REG register for the interrupt cause.

Note:
For definitions of *interrupt*, *interrupt signal*, *interrupt source*, and their correlations, please refer to Chapter 13 *Interrupt Matrix* > Section 13.2 *Interrupt Terminology in ESP32-P4*.

57.12 Register Summary

The addresses in this section are relative to the SD/MMC Host Controller base address provided in Table 9.3-2 in Chapter 9 *System and Memory*.

The abbreviations given in Column **Access** are explained in Section *Access Types for Registers*.

Name	Description	Address	Access
Configuration registers			
SDHOST_CTRL_REG	Control register	0x0000	R/W
SDHOST_CLKDIV_REG	Clock divider configuration	0x0008	R/W
SDHOST_CLKSRC_REG	Clock source selection	0x000C	R/W
SDHOST_CLKENA_REG	Clock enable register	0x0010	R/W
SDHOST_TMOUT_REG	Data and response timeout configuration	0x0014	R/W
SDHOST_CTYPE_REG	Card bus width configuration	0x0018	R/W
SDHOST_BLKSIZ_REG	Card data block size configuration	0x001C	R/W
SDHOST_BYTCNT_REG	Data transfer length configuration	0x0020	R/W
SDHOST_CMDARG_REG	Command argument data	0x0028	R/W
SDHOST_CMD_REG	Command and boot configuration	0x002C	R/W
SDHOST_FIFOTH_REG	FIFO configuration	0x004C	R/W
SDHOST_DEBNCE_REG	Debounce filter time configuration	0x0064	R/W
SDHOST_USRID_REG	User ID (scratchpad)	0x0068	R/W
SDHOST_VERID_REG	Version ID (scratchpad) register	0x006C	RO
SDHOST_HCON_REG	Hardware feature register	0x0070	RO
SDHOST_UHS_REG	UHS-1	0x0074	R/W
SDHOST_RST_N_REG	Card reset	0x0078	R/W
SDHOST_BMOD_REG	Burst mode transfer configuration	0x0080	R/W
SDHOST_PLDMND_REG	Poll demand configuration	0x0084	WO
SDHOST_DBADDR_REG	Linked-list base address	0x0088	R/W
SDHOST_CARDTHRCTL_REG	Card Threshold Control	0x0100	R/W
SDHOST_EMMCDDR_REG	eMMC DDR register	0x010C	R/W
SDHOST_ENSHIFT_REG	Enable Phase Shift	0x0110	R/W
SDHOST_CLK_EDGE_SEL_REG	SDIO control register	0x0800	R/W
SDHOST_DLL_CLK_CONF_REG	SDIO DLL clock control	0x0808	R/W
Interrupt registers			
SDHOST_INTMASK_REG	SDIO interrupt mask register	0x0024	R/W
SDHOST_MINTSTS_REG	Masked interrupt status register	0x0040	RO
SDHOST_RINTSTS_REG	Raw interrupt register	0x0044	R/W1C
SDHOST_IDSTS_REG	DMA interrupt register	0x008C	R/W1C
SDHOST_IDINTEN_REG	DMA interrupt enable register	0x0090	R/W
Status registers			
SDHOST_RESP0_REG	Response data	0x0030	RO
SDHOST_RESP1_REG	Long response data	0x0034	RO
SDHOST_RESP2_REG	Long response data	0x0038	RO
SDHOST_RESP3_REG	Long response data	0x003C	RO

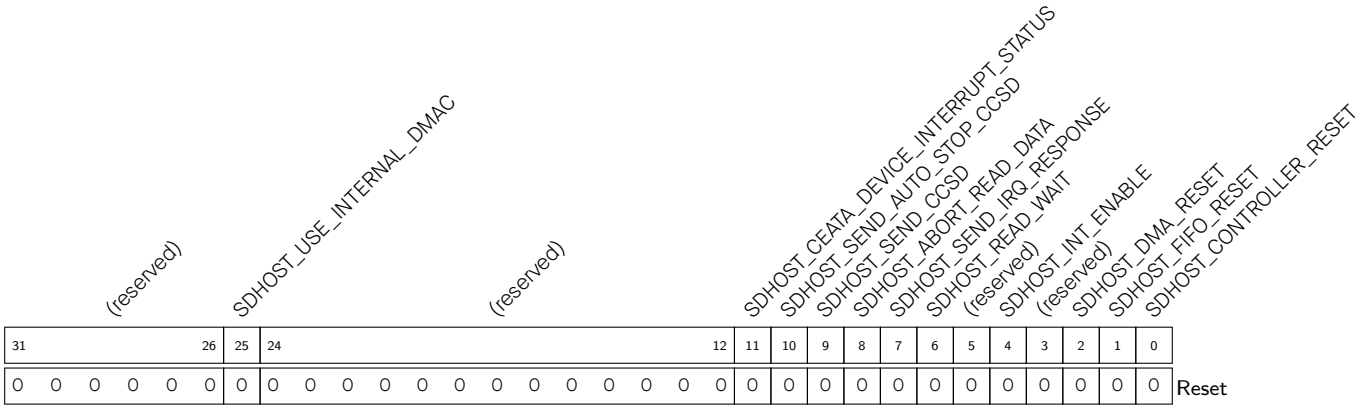
Name	Description	Address	Access
SDHOST_STATUS_REG	SD/MMC status	0x0048	RO
SDHOST_CDETECT_REG	Card detect	0x0050	RO
SDHOST_WRTPRT_REG	Card write protection (WP) status	0x0054	RO
SDHOST_TCBCNT_REG	Transferred byte count	0x005C	RO
SDHOST_TBBCNT_REG	Transferred byte count	0x0060	RO
SDHOST_DSCADDR_REG	Host descriptor address pointer	0x0094	RO
SDHOST_BUFADDR_REG	Host buffer address pointer	0x0098	RO
Transmit and receive data registers			
SDHOST_BUFFIFO_REG	Software write and read transmit data by FIFO	0x0200	R/W

57.13 Registers

The addresses in this section are relative to the SD/MMC Host Controller base address provided in Table 9.3-2 in Chapter 9 *System and Memory*.

For how to program reserved fields, please refer to Section Programming Reserved Register Field.

Register 57.1. SDHOST_CTRL_REG (0x0000)



SDHOST_CONTROLLER_RESET Configures whether to reset the controller. This bit is auto-cleared after controller reset.

0: No effect

1: Reset

(R/W)

SDHOST_FIFO_RESET Configures whether to reset FIFO. This bit is auto-cleared after FIFO reset.

0: No effect

1: Reset

(R/W)

SDHOST_DMA_RESET Configures whether to reset DMA interface. This bit is auto-cleared after DMA reset.

0: No effect

1: Reset

(R/W)

SDHOST_INT_ENABLE Write 1 to enable global interrupt port. (R/W)

SDHOST_READ_WAIT Configures whether to send read-wait to SDIO cards.

0: Clear read wait

1: Assert read wait

(R/W)

Continued on the next page...

Register 57.1. CTRL_REG (0x0000)

Continued from the previous page...

SDHOST_SEND_IRQ_RESPONSE Configures whether to send auto interrupt request (IRQ) response. This bit automatically clears once a response is sent.

To wait for MMC card interrupts, the host issues a CMD40 command and waits for an interrupt response from MMC card(s). In the meantime, if the host wants SD/MMC to exit waiting for the interrupt state, it can set this bit, at which time SD/MMC command state-machine sends a CMD40 response on the bus and returns to the idle state.

0: No effect

1: Send auto IRQ response

(R/W)

SDHOST_ABORT_READ_DATA Configures whether to abort read data. This bit is automatically cleared once the data state machine is reset to idle.

After a suspend-command is issued during a read-operation, software polls the card to find when the suspend-event occurred. Once the suspend-event has occurred, software sets the bit which will reset the data state machine that is waiting for the next block of data.

0: No effect

1: Abort read data

(R/W)

SDHOST_SEND_CCSD Configures whether to send Command Completion Signal Disable (CCSD) to CE-ATA device. Once the CCSD pattern is sent to the device, SD/MMC automatically clears the SDHOST_SEND_CCSD bit. It also sets the Command Done (CD) bit in the SDHOST_RINTSTS_REG register.

Software sets this bit only if the current command is expecting CCS (that is, RW_BLK), and if interrupts are enabled for the CE-ATA device.

0:No effect

1: Send CCSD to CE-ATA device

(R/W)

SDHOST_SEND_AUTO_STOP_CCSD Configures whether to send an internally-generated STOP command (CMD12) to the CE-ATA device. After sending the CCSD, SD/MMC automatically clears the SDHOST_SEND_AUTO_STOP_CCSD bit. After sending this internally-generated STOP command, the Auto Command Done (ACD) bit in SDHOST_RINTSTS_REG is set.

Always Set SDHOST_SEND_AUTO_STOP_CCSD and SDHOST_SEND_CCSD bits together, SDHOST_SEND_AUTO_STOP_CCSD should not be set independently of SDHOST_SEND_CCSD.

0:No effect

1: Send internally-generated STOP command (CMD12) to the CE-ATA device

(R/W)

Continued on the next page...

Register 57.1. CTRL_REG (0x0000)

Continued from the previous page...

SDHOST_CEATA_DEVICE_INTERRUPT_STATUS Write 1 to enable interrupts in CE-ATA device (nIEN = 0 in ATE control register).

Software should appropriately write to this bit after the power-on reset or any other reset to the CE-ATA device. After reset, the CE-ATA device's interrupt is usually disabled (nIEN = 1). If the host enables the CE-ATA device's interrupt, then software should set this bit. (R/W)

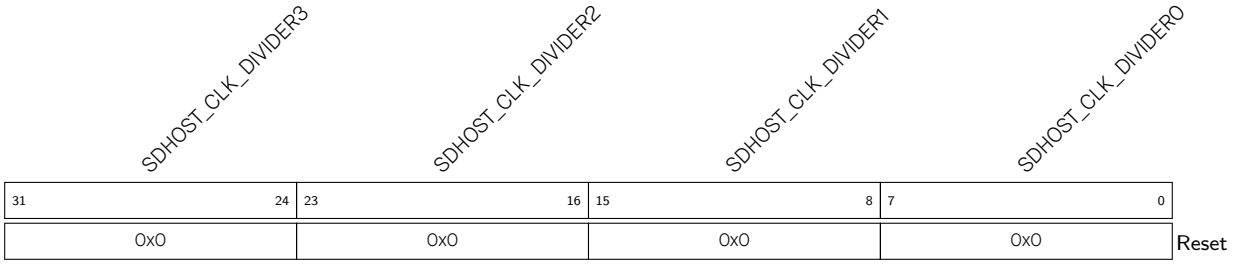
SDHOST_USE_INTERNAL_DMAC Configures whether to use DMA for data transfer.

0: Not use

1: Use

(R/W)

Register 57.2. SDHOST_CLKDIV_REG (0x0008)



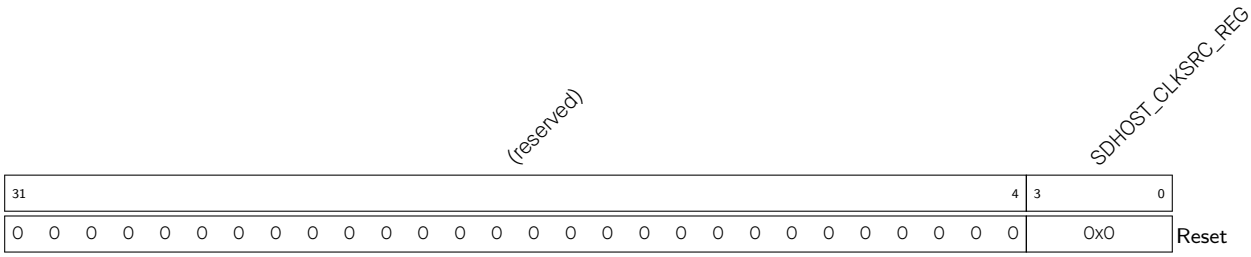
SDHOST_CLK_DIVIDER0 Configures clock divider0 value. Clock divisor is 2*n, where n = 0 bypasses the divider (divisor of 1). For example, a value of 1 means divided by 2*1 = 2, a value of 0xFF means divided by 2*255 = 510, and so on. (R/W)

SDHOST_CLK_DIVIDER1 Configures clock divider1 value. Clock divisor is 2*n, where n = 0 bypasses the divider (divisor of 1). For example, a value of 1 means divided by 2*1 = 2, a value of 0xFF means divided by 2*255 = 510, and so on. (R/W)

SDHOST_CLK_DIVIDER2 Configures clock divider2 value. Clock divisor is 2*n, where n = 0 bypasses the divider (divisor of 1). For example, a value of 1 means divided by 2*1 = 2, a value of 0xFF means divided by 2*255 = 510, and so on. (R/W)

SDHOST_CLK_DIVIDER3 Configures clock divider3 value. Clock divisor is 2*n, where n = 0 bypasses the divider (divisor of 1). For example, a value of 1 means divided by 2*1 = 2, a value of 0xFF means divided by 2*255 = 510, and so on. (R/W)

Register 57.3. SDHOST_CLKSRC_REG (0x000C)



SDHOST_CLKSRC_REG Configures clock divider source for two SD cards. Each card has two bits assigned to it. For example, bit[1:0] are assigned to card 0, and bit[3:2] are assigned to card 1. Card 1/0 maps and internally routes clock divider[0:3] outputs to cclk_out[1:0] pins, depending on bit value. For every card's clock divider source:

0x0: Clock divider 0

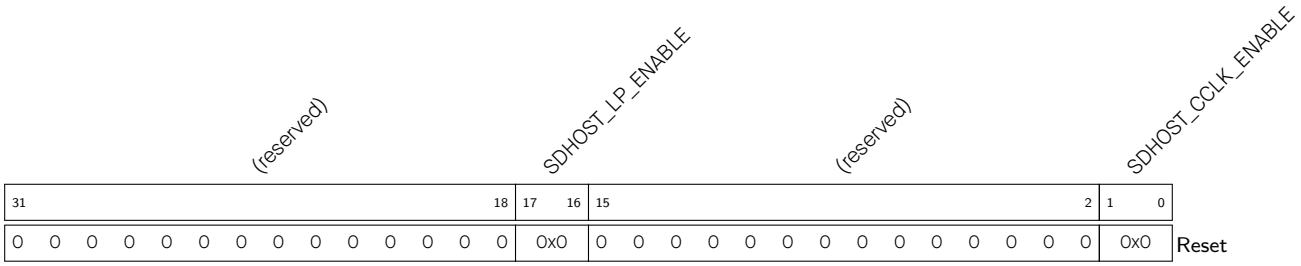
0x1: Clock divider 1

0x2: Clock divider 2

0x3: Clock divider 3

(R/W)

Register 57.4. SDHOST_CLKENA_REG (0x0010)



SDHOST_CCLK_ENABLE Configures whether to enable two SD card clocks or one MMC card clock. One bit per card. For every card's bit:

0: Clock disabled

1: Clock enabled

(R/W)

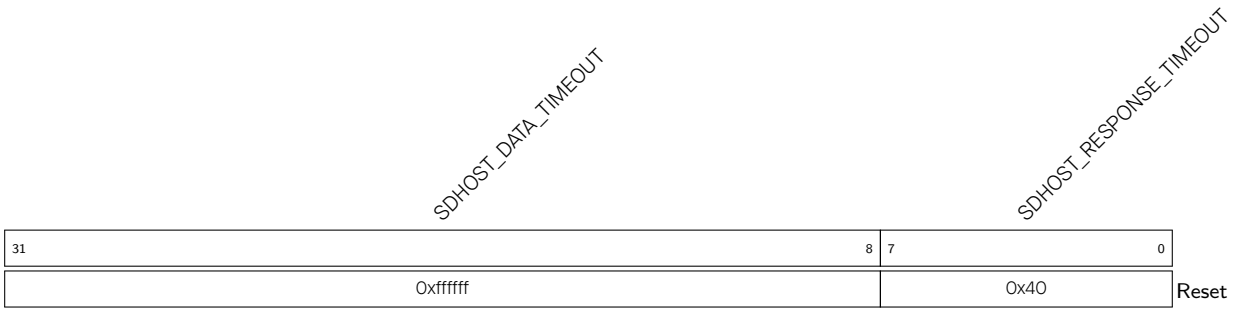
SDHOST_LP_ENABLE Configures whether to disable the clock when the card is in an IDLE state. One bit per card. For every card's bit:

0: Clock disabled

1: Clock enabled

(R/W)

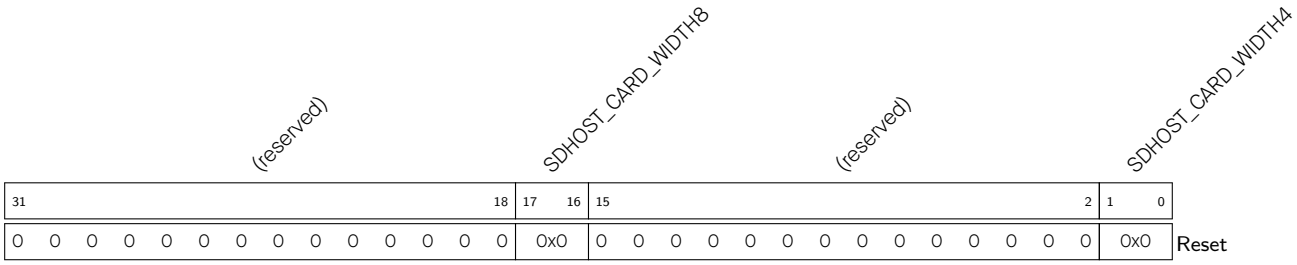
Register 57.5. SDHOST_TMOUT_REG (0x0014)



SDHOST_RESPONSE_TIMEOUT Configures response timeout value. Measurement unit: Card output clock period. (R/W)

SDHOST_DATA_TIMEOUT Configures card data read timeout value. This value is also used for data starvation by host timeout. The timeout counter is started only after the card clock is stopped. Measurement unit: Card output clock period.
NOTE: The software timer should be used if the timeout value is in the order of 100 ms. In this case, read data timeout interrupt (DRTO) needs to be disabled.
(R/W)

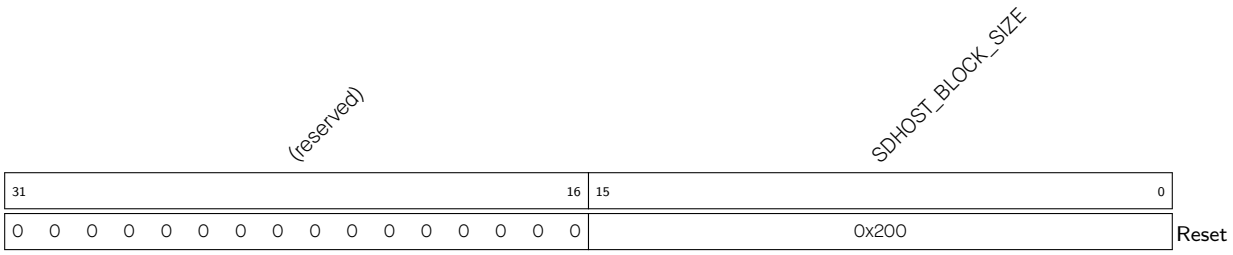
Register 57.6. SDHOST_CTYPE_REG (0x0018)



SDHOST_CARD_WIDTH4 Configures whether the card is 1-bit or 4-bit mode. One bit per card. Bit[0] corresponds to card[0]. For every card's bit:
0: 1-bit mode
1: 4-bit mode
(R/W)

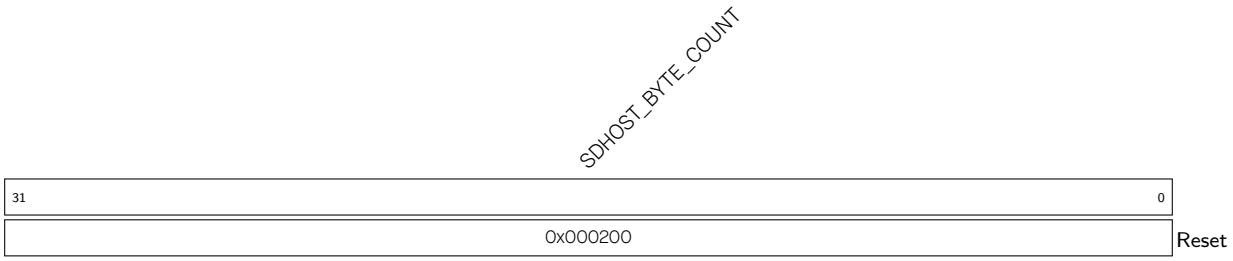
SDHOST_CARD_WIDTH8 Configures whether the card is 8-bit mode. One bit per card. Bit[0] corresponds to card[0]. For every card's bit:
0: Non 8-bit mode
1: 8-bit mode
(R/W)

Register 57.7. SDHOST_BLKSIZ_REG (0x001C)



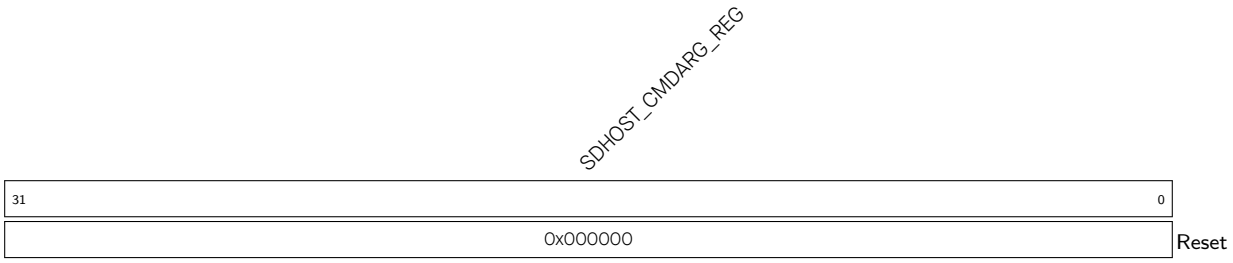
SDHOST_BLOCK_SIZE Configures the Block Size. (R/W)

Register 57.8. SDHOST_BYTCNT_REG (0x0020)



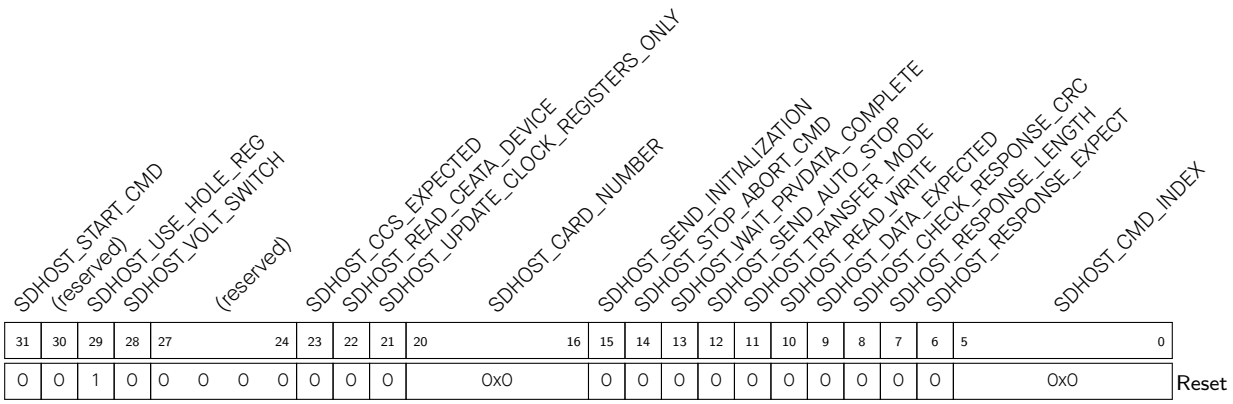
SDHOST_BYTE_COUNT Configures the number of bytes to be transferred, should be an integral multiple of Block Size for block transfers. For data transfers of undefined byte lengths, the byte count should be set to 0. When the byte count is set to 0, it is the responsibility of the host to explicitly send a stop/abort command to terminate data transfer. (R/W)

Register 57.9. SDHOST_CMDARG_REG (0x0028)



SDHOST_CMDARG_REG Configures the value indicates the command argument to be passed to the card. (R/W)

Register 57.10. SDHOST_CMD_REG (0x002C)



SDHOST_CMD_INDEX Configures the Command index send to card. (R/W)

SDHOST_RESPONSE_EXPECT Configures whether to expect response from card.

0: No response expected from card

1: Response expected from card

(R/W)

SDHOST_RESPONSE_LENGTH Configures response length from card.

0: Short response expected from card

1: Long response expected from card

(R/W)

SDHOST_CHECK_RESPONSE_CRC Configures whether to check response CRC.

0: Do not check

1: Check response CRC

(R/W)

SDHOST_DATA_EXPECTED Configures whether to expect data transfer.

0: No data transfer expected

1: Data transfer expected

(R/W)

SDHOST_READ_WRITE Configures data transfer direction.

0: Read from card

1: Write to card

(R/W)

SDHOST_TRANSFER_MODE Configures data transfer mode.

0: Block data transfer command

1: Stream data transfer command

(R/W)

SDHOST_SEND_AUTO_STOP Configures whether to automatically send stop command.

0: No stop command is sent at the end of data transfer

1: Send stop command at the end of data transfer

(R/W)

Continued on the next page...

Register 57.10. SDHOST_CMD_REG (0x002C)

Continued from the previous page...

SDHOST_WAIT_PRVDATA_COMPLETE Configures whether to send command at once.

0: Send command at once, even if previous data transfer has not completed

1: Wait for previous data transfer to complete before sending Command

(R/W)

SDHOST_STOP_ABORT_CMD Configures whether the stop/abort command stops data transfer.

0: Neither stop nor abort command can stop the current data transfer. If abort is sent to the card device currently selected, or the card device is not in data-transfer mode, then bit should be set to 0

1: Stop or abort command intended to stop current data transfer in progress

(R/W)

SDHOST_SEND_INITIALIZATION Configures whether to send initialization sequence (80 clocks of 1) before sending this command. After powered on, 80 clocks must be sent to card for initialization before sending any commands to card. Bit should be set while sending the first command to card so that the controller will initialize clocks before sending command to card.

0: Do not send initialization sequence before sending this command

1: Send initialization sequence before sending this command

(R/W)

SDHOST_CARD_NUMBER Configures card number in use, the physical slot number of the card being accessed. In SD-only mode, up to two cards are supported. (R/W)

SDHOST_UPDATE_CLOCK_REGISTERS_ONLY Configures whether to only update clock register value into card clock domain.

Following register values are transferred into card clock domain: SDHOST_CLKDIV_REG, SDHOST_CLRSRC_REG, and SDHOST_CLKENA_REG.

When this bit is set, there will be no Command Done interrupts because no command is sent to SD_MMC_CEATA cards.

0: Normal command sequence

1: Do not send commands, just update clock register value into card clock domain

(R/W)

SDHOST_READ_CEATA_DEVICE Configures whether to read access CE-ATA device.

This bit is used to disable read data timeout indication while performing CE-ATA read transfers.

0: Host is not performing read access (RW_REG or RW_BLK)towards CE-ATA device

1: Host is performing read access (RW_REG or RW_BLK) towards CE-ATA device

(R/W)

Continued on the next page...

Register 57.10. SDHOST_CMD_REG (0x002C)

Continued from the previous page...

SDHOST_CCS_EXPECTED Configures whether to expect Command Completion Signal (CCS).

0: Interrupts are not enabled in CE-ATA device (nIEN = 1 in ATA control register), or command does not expect CCS from CE-ATA device

1: Interrupts are enabled in CE-ATA device (nIEN = 0), and RW_BLK command expects CCS from CE-ATA device

(R/W)

SDHOST_VOLT_SWITCH Configures whether to enable voltage switching.

0: No voltage switching

1: Voltage switching enabled, must be set for CMD11 only

(R/W)

SDHOST_USE_HOLE_REG Configures whether to use Hold Register.

0: CMD and DATA sent to card bypassing HOLD Register

1: CMD and DATA sent to card through the HOLD Register

(R/W)

SDHOST_START_CMD Configures whether to to start the command.

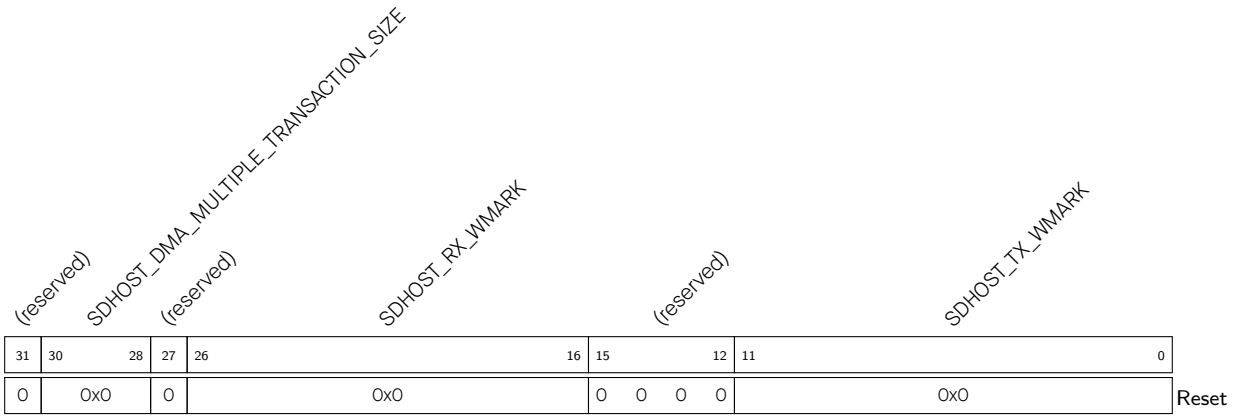
0: Not start

1: Start

Once command is served by the CIU, this bit is automatically cleared.

When this bit is set, host should not attempt to write to any command registers. If a write is attempted, hardware lock error is set in raw interrupt register. (R/W/SC)

Register 57.11. SDHOST_FIFOTH_REG (0x004C)

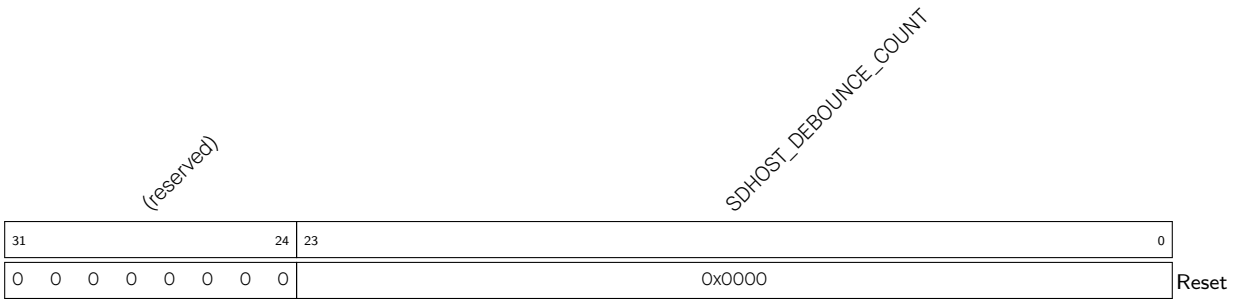


SDHOST_TX_WMARK Configures FIFO threshold watermark level when transmitting data to card. When FIFO data count is less than or equal to this number, DMA/FIFO request is raised. During end of packet, request or interrupt is generated, regardless of threshold programming.
In non-DMA mode, when transmitting FIFO threshold (TXDR) interrupt is enabled, then interrupt is generated instead of DMA request. During end of packet, on last interrupt, host is responsible for filling FIFO with only the required remaining bytes (not before FIFO is full or after CIU completes data transfers, because FIFO may not be empty).
In DMA mode, at end of packet, if last transfer is less than burst size, DMA controller does single cycles until required bytes are transferred. (R/W)

SDHOST_RX_WMARK Configures FIFO threshold watermark level when receiving data from card. When FIFO data count reaches greater than this number, DMA/FIFO request is raised. During end of packet, request is generated regardless of threshold programming in order to complete any remaining data.
In non-DMA mode, when receiver FIFO threshold (RXDR) interrupt is enabled, then interrupt is generated instead of DMA request. During end of packet, interrupt is not generated if threshold programming is larger than any remaining data. It is responsibility of host to read remaining bytes on seeing Data Transfer Done interrupt.
In DMA mode, at end of packet, even if remaining bytes are less than threshold, DMA request does single transfers to flush out any remaining bytes before Data Transfer Done interrupt is set. (R/W)

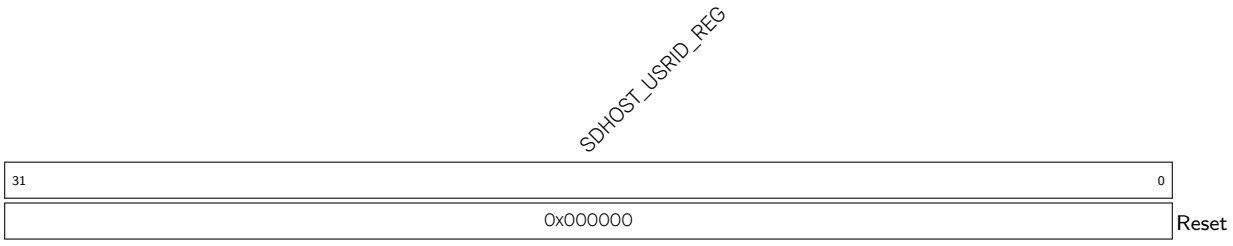
SDHOST_DMA_MULTIPLE_TRANSACTION_SIZE Configures burst size of multiple transaction, should be programmed same as DMA controller multiple-transaction-size SDHOST_SRC/DEST_MSIZE.
0x0: 1-byte transfer
0x1: 4-byte transfer
0x2: 8-byte transfer
0x3: 16-byte transfer
0x4: 32-byte transfer
0x5: 64-byte transfer
0x6: 128-byte transfer
0x7: 256-byte transfer
(R/W)

Register 57.12. SDHOST_DEBNCE_REG (0x0064)



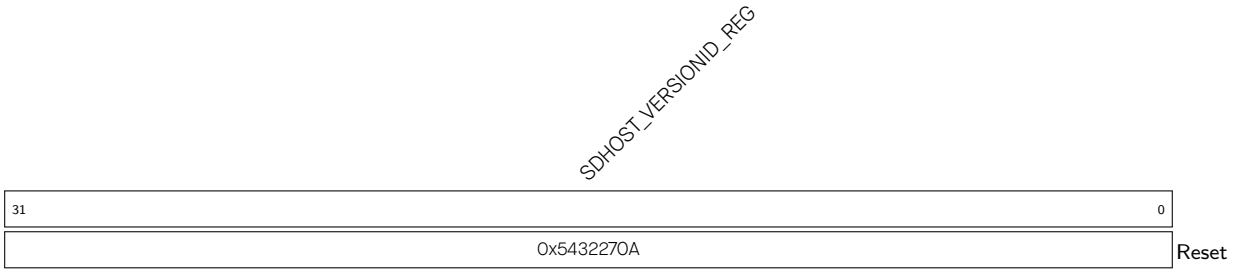
SDHOST_DEBOUNCE_COUNT Configures the number of host clocks (clk) used by debounce filter logic. The typical debounce time is 5 ~ 25 ms to prevent the card instability when the card is inserted or removed. (R/W)

Register 57.13. SDHOST_USRID_REG (0x0068)



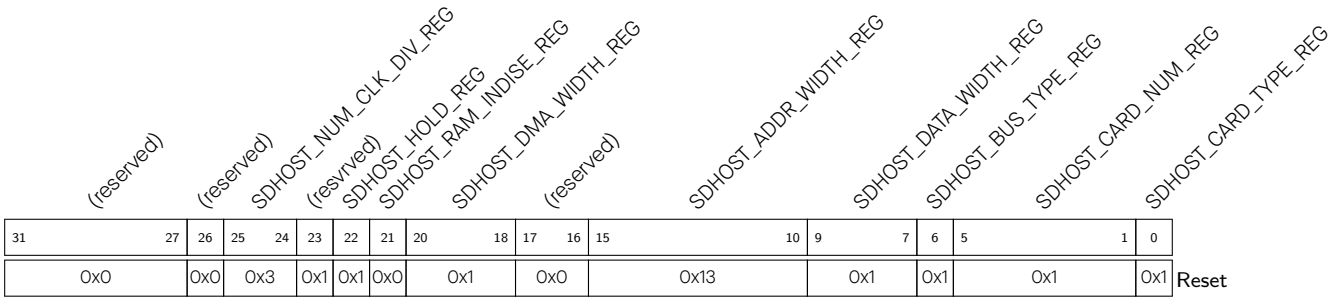
SDHOST_USRID_REG Configures user identification, value set by users. Can also be used as a scratchpad register by users. (R/W)

Register 57.14. SDHOST_VERID_REG (0x006C)



SDHOST_VERSIONID_REG Represents hardware version. Can also be read by firmware. (RO)

Register 57.15. SDHOST_HCON_REG (0x0070)



SDHOST_NUM_CLK_DIV_REG Represents 4 clk divider in design. (RO)

SDHOST_HOLD_REG Represents a hold register in data path. (RO)

SDHOST_RAM_INDISE_REG Represents that the SDMMC module contains RAM. (RO)

SDHOST_DMA_WIDTH_REG Represents that the DMA data width is 32. (RO)

SDHOST_ADDR_WIDTH_REG Represents that the register address width is 32. (RO)

SDHOST_DATA_WIDTH_REG Represents that the register data width is 32. (RO)

SDHOST_BUS_TYPE_REG Represents that the register is configured for the APB bus. (RO)

SDHOST_CARD_NUM_REG Represents that the number of supported cards is 2. (RO)

SDHOST_CARD_TYPE_REG Represents that the hardware supports SDIO and MMC. (RO)

Register 57.16. SDHOST_UHS_REG (0x0074)



SDHOST_VOLT_REG Configures High Voltage mode per card. Bit[0] corresponds to card[0]. For every card bit:

0: 3.3 V VDD

1: 1.8 V VDD

(R/W)

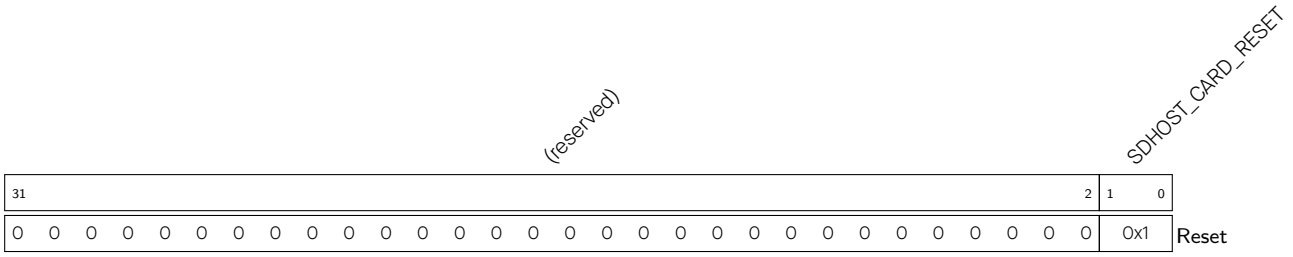
SDHOST_DDR_REG Configures DDR (double data rate) mode selection per card. Bit[0] corresponds to card[0]. For every card bit:

0: Non-DDR mode

1: DDR mode

(R/W)

Register 57.17. SDHOST_RST_N_REG (0x0078)



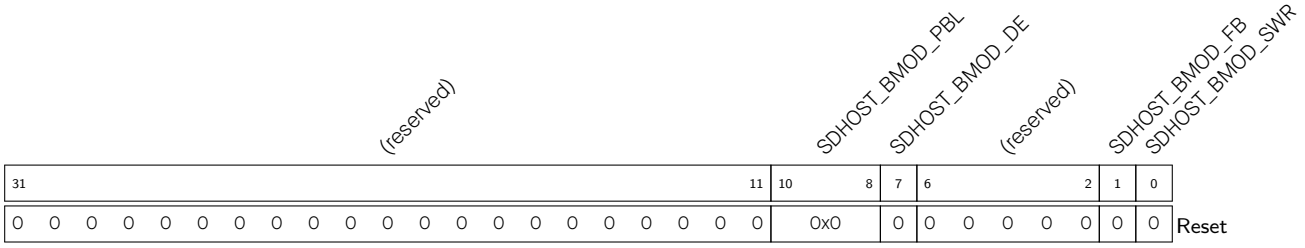
SDHOST_CARD_RESET Configures hardware reset per card. Bit[0] corresponds to card[0]. These bits cause the cards to enter the pre-idle state, which requires them to be re-initialized. For every card bit:

0: Reset

1: Active mode

(R/W)

Register 57.18. SDHOST_BMOD_REG (0x0080)



SDHOST_BMOD_SWR Configures whether to reset all DMA internal registers. It is automatically cleared after one clock cycle.

0: No effect

1: Reset

(R/W)

SDHOST_BMOD_FB Configures whether the AHB Master interface performs fixed burst transfers.

0: Use SINGLE and INCR burst transfer operations

1: Only use SINGLE, INCR4, INCR8 or INCR16 burst transfers

(R/W)

SDHOST_BMOD_DE Configures whether to enable the DMA.

0: Not enable

1: Enable

(R/W)

SDHOST_BMOD_PBL Configures the maximum number of beats to be performed in one DMA transaction.

0x0: 1-beat transfer

0x1: 4-beat transfer

0x2: 8-beat transfer

0x3: 16-beat transfer

0x4: 32-beat transfer

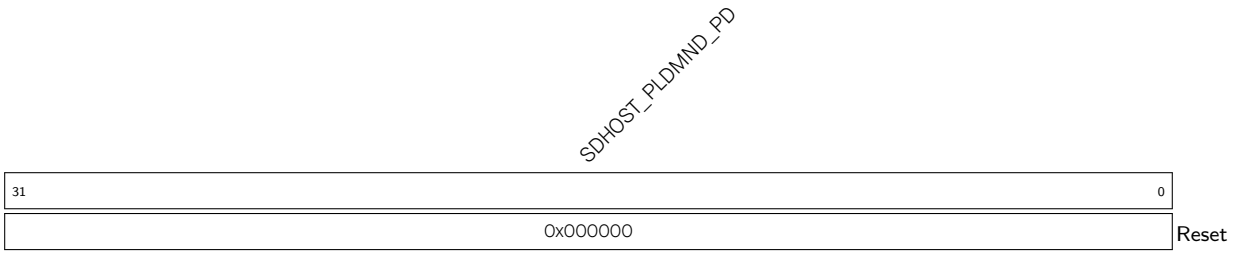
0x5: 64-beat transfer

0x6: 128-beat transfer

0x7: 256-beat transfer

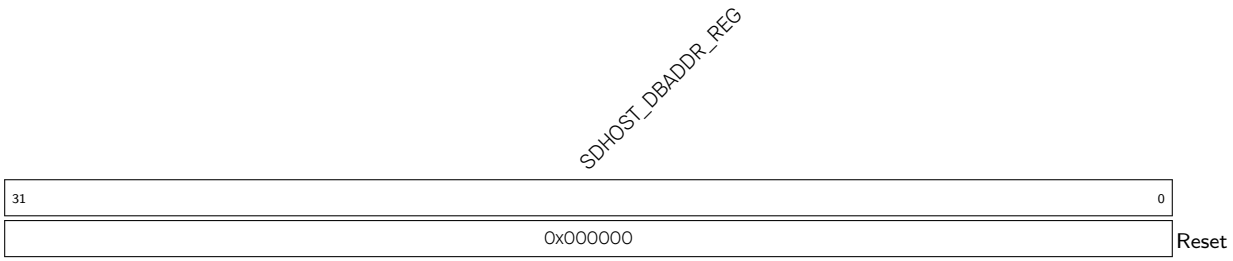
(R/W)

Register 57.19. SDHOST_PLDMND_REG (0x0084)



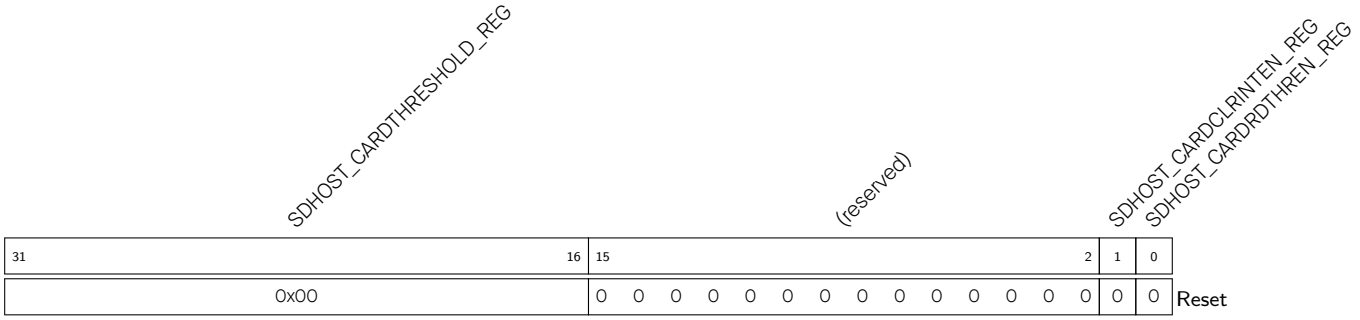
SDHOST_PLDMND_PD Configures the DMA FSM to resume normal descriptor fetch operation. If the OWNER bit of a descriptor is not set, the FSM goes to the Suspend state, software needs to write any value into this field for the FSM to resume normal descriptor fetch operation. (WO)

Register 57.20. SDHOST_DBADDR_REG (0x0088)



SDHOST_DBADDR_REG Configures the base address of the linked list (First Descriptor). Bit[1:0] are ignored and taken as all-zero by the DMA internally. (R/W)

Register 57.21. SDHOST_CARDTHRCTL_REG (0x0100)



SDHOST_CARDRDTHREN_REG Configures whether to enable card read threshold.

0: Not enable

1: Enable

(R/W)

SDHOST_CARDCLRINTEN_REG Configures whether to enable busy clear interrupt generation.

0: Not enable

1: Enable

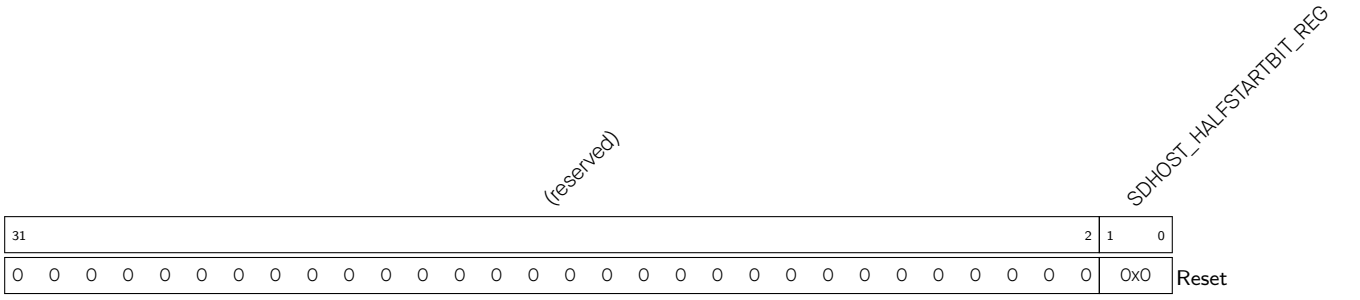
(R/W)

SDHOST_CARDTHRESHOLD_REG Configures the card read threshold. Measurement unit: byte.

This field is valid only when SDHOST_CARDRDTHREN_REG is set to 1.

The value should be less than the FIFO size 512.(R/W)

Register 57.22. SDHOST_EMMCDDR_REG (0x010C)



SDHOST_HALFSTARTBIT_REG Configures the start bit detection mechanism duration of start bit. Each bit refers to one card. Set this bit to 1 for eMMC4.5 and above, set to 0 for SD applications.

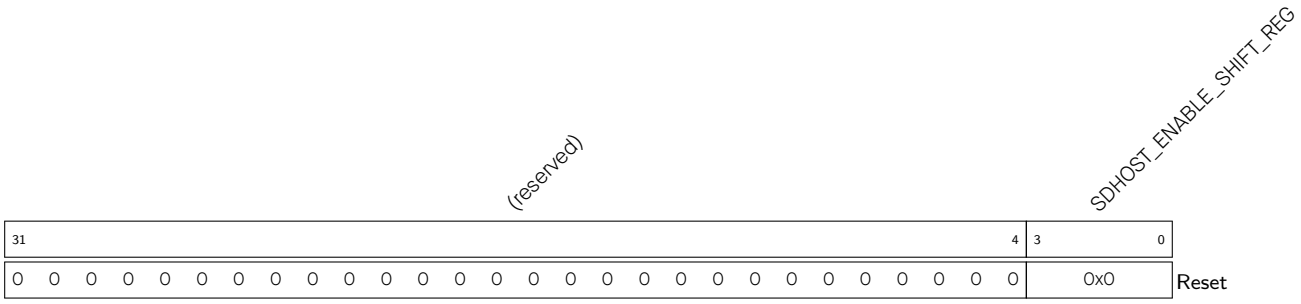
For eMMC4.5, start bit can be:

0: Full cycle

1: Less than one full cycle

(R/W)

Register 57.23. SDHOST_ENSHIFT_REG (0x0110)



SDHOST_ENABLE_SHIFT_REG Configures the amount of phase shift provided on the default enables in the design. Two bits are assigned for each card, e.g., bit[1:0] are assigned to card 0. For every card:

0x0: Default phase shift

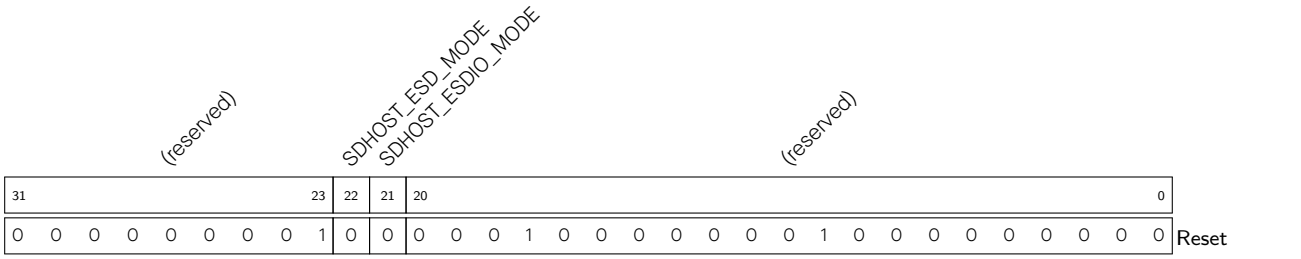
0x1: Enables shifted to next immediate positive edge

0x2: Enables shifted to next immediate negative edge

0x3: Reserved

(R/W)

Register 57.24. SDHOST_CLK_EDGE_SEL_REG (0x0800)



SDHOST_ESDIO_MODE Configures whether to enable eSDIO mode.

0: Not enable

1: Enable

(R/W)

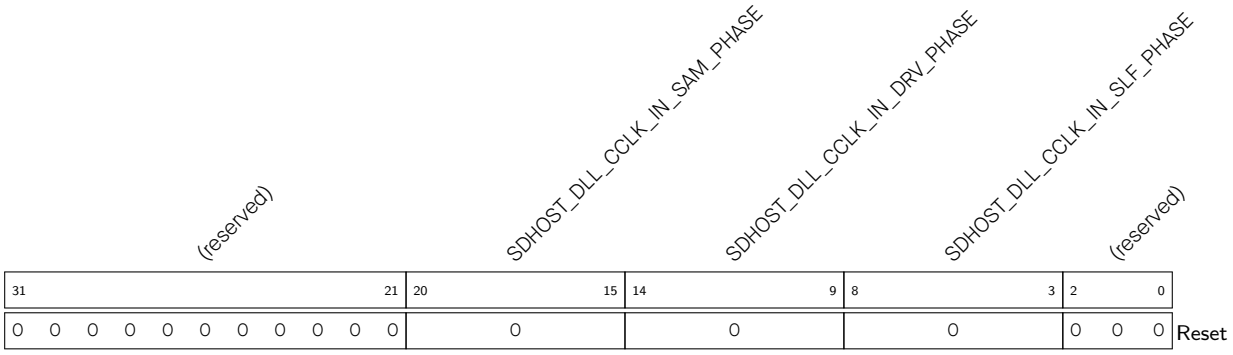
SDHOST_ESD_MODE Configures whether to enable eSD mode.

0: Not enable

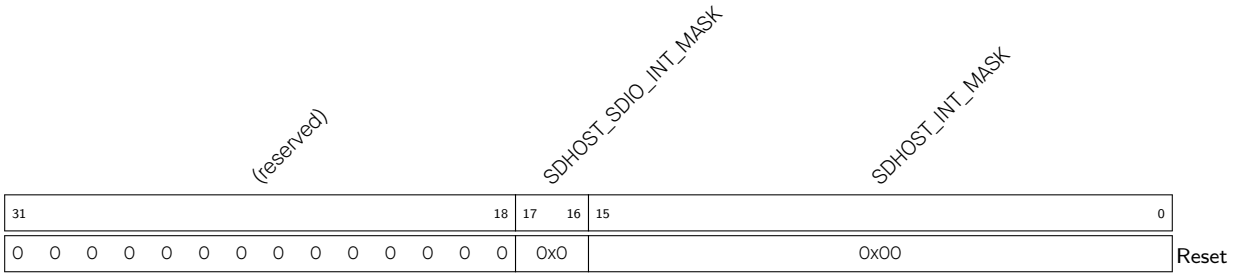
1: Enable

(R/W)

Register 57.25. SDHOST_DLL_CLK_CONF_REG (0x0808)


(reserved)
SDHOST_DLL_CCLK_IN_SAM_PHASE
SDHOST_DLL_CCLK_IN_DRV_PHASE
SDHOST_DLL_CCLK_IN_SLF_PHASE
(reserved)
31 21 20 15 14 9 8 3 2 0
0 0 0

Register 57.26. SDHOST_INTMASK_REG (0x0024)



SDHOST_INT_MASK Write 1 to correspond bit to enable correspond interrupt. For every bit:

Bit 15 (EBE): End-bit error/no CRC error

Bit 14 (ACD): Auto command done

Bit 13 (SBE/BCI): Rx Start Bit Error

Bit 12 (HLE): Hardware locked write error

Bit 11 (FRUN): FIFO underrun/overrun error

Bit 10 (HTO): Data starvation-by-host timeout

Bit 9 (DRTO): Data read timeout

Bit 8 (RTO): Response timeout

Bit 7 (DCRC): Data CRC error

Bit 6 (RCRC): Response CRC error

Bit 5 (RXDR): Receive FIFO data request

Bit 4 (TXDR): Transmit FIFO data request

Bit 3 (DTO): Data transfer over

Bit 2 (CMDD): Command done

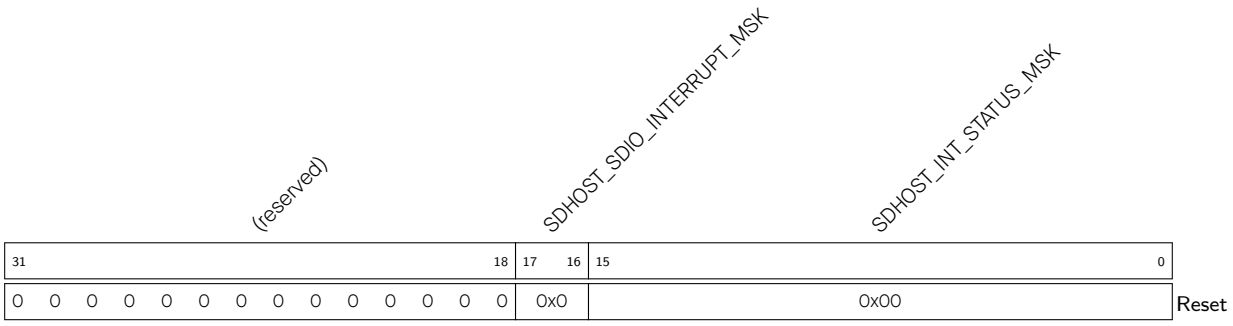
Bit 1 (RE): Response error

Bit 0 (CD): Card detect

(R/W)

SDHOST_SDIO_INT_MASK Write 1 to correspond bit to enable interrupt from correspond card. One bit per card. Bit[0] corresponds to card[0]. (R/W)

Register 57.27. SDHOST_MINTSTS_REG (0x0040)

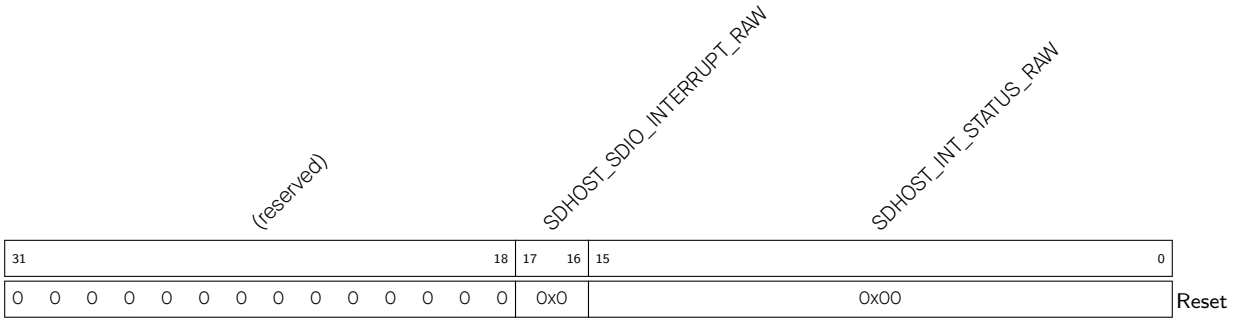


SDHOST_INT_STATUS_MSK The masked interrupt status. For every bit:

Bit 15 (EBE): End-bit error/no CRC error
Bit 14 (ACD): Auto command done
Bit 13 (SBE/BCI): RX Start Bit Error
Bit 12 (HLE): Hardware locked write error
Bit 11 (FRUN): FIFO underrun/overrun error
Bit 10 (HTO): Data starvation by host timeout
Bit 9 (DTRO): Data read timeout
Bit 8 (RTO): Response timeout
Bit 7 (DCRC): Data CRC error
Bit 6 (RCRC): Response CRC error
Bit 5 (RXDR): Receive FIFO data request
Bit 4 (TXDR): Transmit FIFO data request
Bit 3 (DTO): Data transfer over
Bit 2 (CMDD): Command done
Bit 1 (RE): Response error
Bit 0 (CD): Card detect
(RO)

SDHOST_SDIO_INTERRUPT_MSK The masked interrupt status from SDIO card. One bit for each card. Bit[0] corresponds to card[0]. (RO)

Register 57.28. SDHOST_RINTSTS_REG (0x0044)

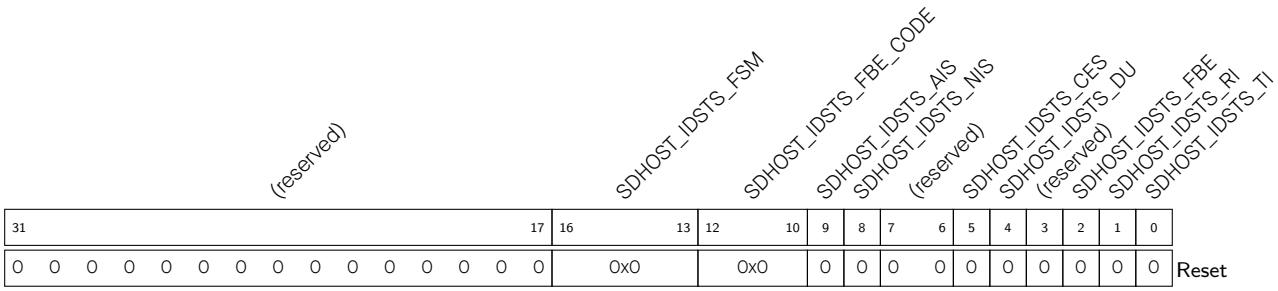


SDHOST_INT_STATUS_RAW The raw interrupt status. For every bit:

Bit 15 (EBE): End-bit error/no CRC error
Bit 14 (ACD): Auto command done
Bit 13 (SBE/BCI): RX Start Bit Error
Bit 12 (HLE): Hardware locked write error
Bit 11 (FRUN): FIFO underrun/overrun error
Bit 10 (HTO): Data starvation by host timeout
Bit 9 (DTRO): Data read timeout
Bit 8 (RTO): Response timeout
Bit 7 (DCRC): Data CRC error
Bit 6 (RCRC): Response CRC error
Bit 5 (RXDR): Receive FIFO data request
Bit 4 (TXDR): Transmit FIFO data request
Bit 3 (DTO): Data transfer over
Bit 2 (CMDD): Command done
Bit 1 (RE): Response error
Bit 0 (CD): Card detect
(R/W1C)

SDHOST_SDIO_INTERRUPT_RAW The raw interrupt status from SDIO card. One bit per card. Bit[0] corresponds to card[0]. (R/W1C)

Register 57.29. SDHOST_IDSTS_REG (0x008C)



SDHOST_IDSTS_TI The raw interrupt status of Transmit Interrupt. Indicates that data transmission is finished for a descriptor. (R/W1C)

SDHOST_IDSTS_RI The raw interrupt status of Receive Interrupt. Indicates the completion of data reception for a descriptor. (R/W1C)

SDHOST_IDSTS_FBE The raw interrupt status of Fatal Bus Error Interrupt. Indicates that a Bus Error occurred (SDHOST_IDSTS_REG[12:10]). (R/W1C)

SDHOST_IDSTS_DU The raw interrupt status of Descriptor Unavailable Interrupt. This bit is set when the descriptor is unavailable due to OWNER bit = 0 (DES0[31] = 0). (R/W1C)

SDHOST_IDSTS_CES The raw interrupt status of Card Error Summary. Indicates the status of the transaction to/from the card, also present in SDHOST_RINTSTS_REG. Indicates the logical OR of the following bits:
EBE: End-bit error/no CRC error
SBE/BCI: RX Start Bit Error
DRTO: Data read timeout
RTO: Response timeout
DCRC: Data CRC error
RCRC: Response CRC error
RE: Response error
The abort condition of the DMA depends on the setting of this field. If this field is enabled, then the DMA aborts on a response error. (R/W1C)

SDHOST_IDSTS_NIS The raw interrupt status of Normal Interrupt Summary. Logical OR of SDHOST_IDSTS_REG[1:0]. Only unmasked bits affect this bit. This is a sticky bit and must be cleared each time a corresponding bit that causes this field to be set is cleared by software. (R/W1C)

SDHOST_IDSTS_AIS The raw interrupt status of Abnormal Interrupt Summary. Logical OR of SDHOST_IDSTS_REG[2], SDHOST_IDSTS_REG[4]. Only unmasked bits affect this bit. This is a sticky bit and must be cleared each time a corresponding bit that causes this field to be set is cleared by software. (R/W1C)

Continued on the next page...

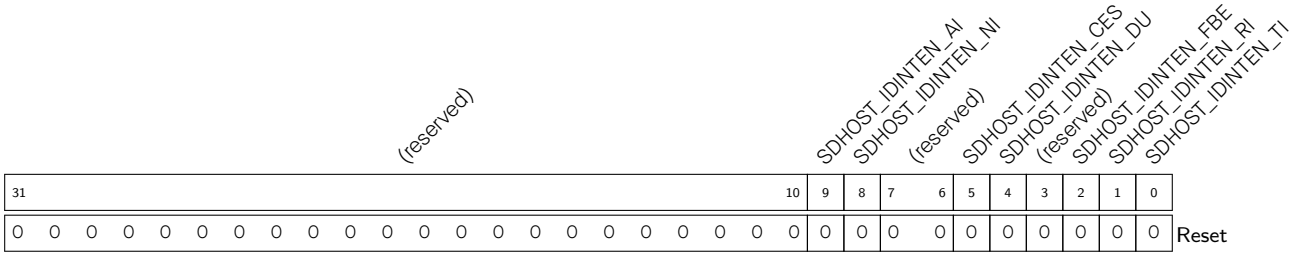
Register 57.29. SDHOST_IDSTS_REG (0x008C)

Continued from the previous page...

SDHOST_IDSTS_FBE_CODE Represents the type of error that caused a Bus Error, valid only when SDHOST_IDSTS_REG[2] is set.
0x1: Host Abort received during transmission
0x2: Host Abort received during reception
Others: Reserved
(RO)

SDHOST_IDSTS_FSM Represents DMA FSM present state.
0x0: DMA_IDLE (idle state)
0x1: DMA_SUSPEND (suspend state)
0x2: DESC_RD (descriptor reading state)
0x3: DESC_CHK (descriptor checking state)
0x4: DMA_RD_REQ_WAIT (read-data request waiting state)
0x5: DMA_WR_REQ_WAIT (write-data request waiting state)
0x6: DMA_RD (data-read state)
0x7: DMA_WR (data-write state)
0x8: DESC_CLOSE (descriptor close state)
(RO)

Register 57.30. SDHOST_IDINTEN_REG (0x0090)



SDHOST_IDINTEN_TI Write 1 to enable interrupt of Transmit.(R/W)

SDHOST_IDINTEN_RI Write 1 to enable interrupt of Receive. (R/W)

SDHOST_IDINTEN_FBE Write 1 to enable interrupt of Fatal Bus Error. (R/W)

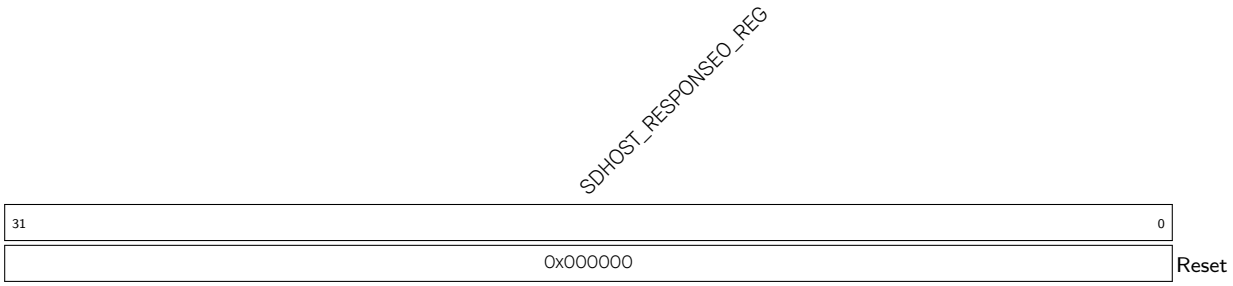
SDHOST_IDINTEN_DU Write 1 to enable interrupt of Descriptor Unavailable. (R/W)

SDHOST_IDINTEN_CES Write 1 to enable interrupt of Card Error summary. (R/W)

SDHOST_IDINTEN_NI Write 1 to enable interrupt of Normal Interrupt Summary. (R/W)

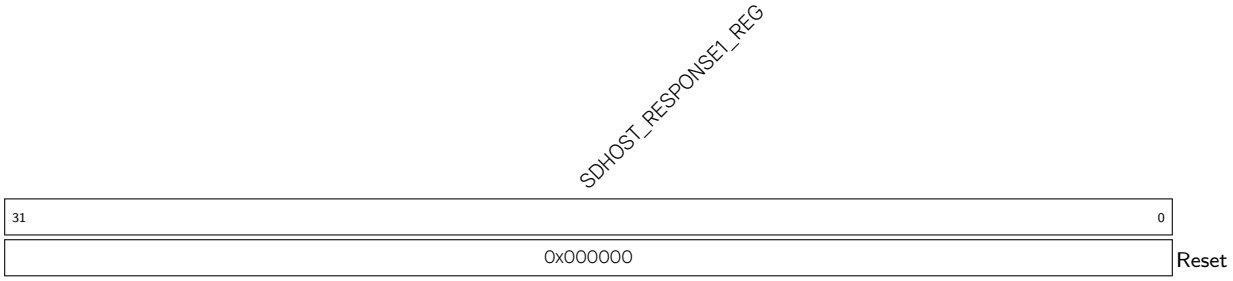
SDHOST_IDINTEN_AI Write 1 to enable interrupt of Abnormal Interrupt Summary. (R/W)

Register 57.31. SDHOST_RESP0_REG (0x0030)



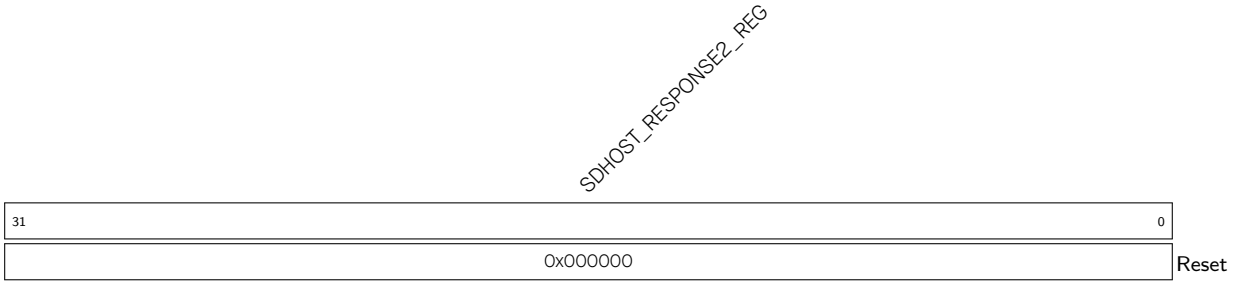
SDHOST_RESPONSE0_REG Represents bit[31:0] of response. (RO)

Register 57.32. SDHOST_RESP1_REG (0x0034)



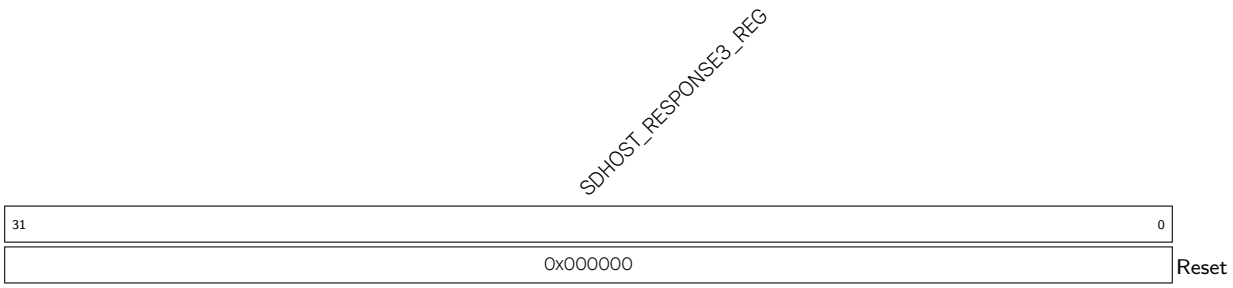
SDHOST_RESPONSE1_REG Represents bit[63:32] of long response. (RO)

Register 57.33. SDHOST_RESP2_REG (0x0038)



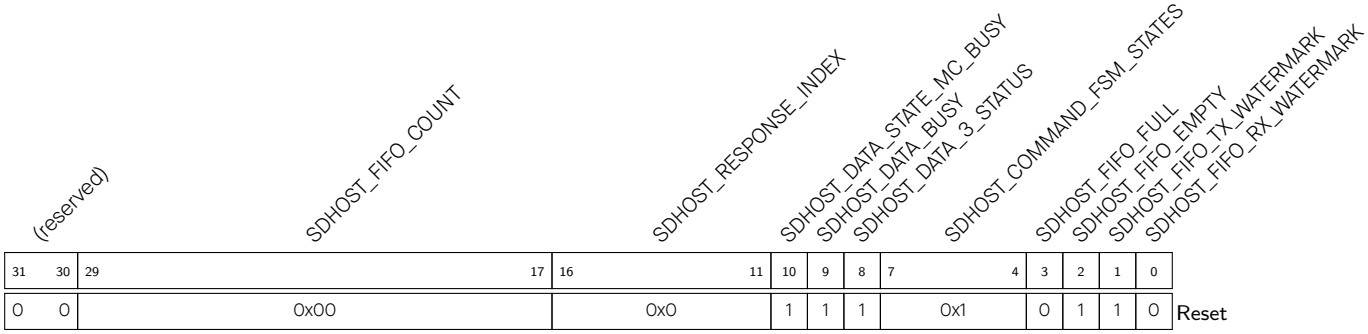
SDHOST_RESPONSE2_REG Represents bit[95:64] of long response. (RO)

Register 57.34. SDHOST_RESP3_REG (0x003C)



SDHOST_RESPONSE3_REG Represents bit[127:96] of long response. (RO)

Register 57.35. SDHOST_STATUS_REG (0x0048)



SDHOST_FIFO_RX_WATERMARK Represents whether the FIFO reached Receive watermark level, not qualified with data transfer.

0: Not reach

1: Reach

(RO)

SDHOST_FIFO_TX_WATERMARK Represents whether the FIFO reached Transmit watermark level, not qualified with data transfer.

0: Not reach

1: Reach

(RO)

SDHOST_FIFO_EMPTY Represents whether the FIFO is empty.

0: Not empty

1: Empty

(RO)

SDHOST_FIFO_FULL Represents whether the FIFO is full.

0: Not full

1: Full

(RO)

Continued on the next page...

Register 57.35. SDHOST_STATUS_REG (0x0048)

Continued from the previous page...

SDHOST_COMMAND_FSM_STATES Represents the command FSM states.
0x0: Idle
0x1: Send init sequence
0x2: Send cmd start bit
0x3: Send cmd tx bit
0x4: Send cmd index + arg
0x5: Send cmd crc7
0x6: Send cmd end bit
0x7: Receive resp start bit
0x8: Receive resp IRQ response
0x9: Receive resp tx bit
0xA: Receive resp cmd idx
0xB: Receive resp data
0xC: Receive resp crc7
0xD: Receive resp end bit
0xE: Cmd path wait NCC
0xF: Wait, cmd-to-response turnaround
(RO)

SDHOST_DATA_3_STATUS Represents the value of the sdhost_card_data[3] signal, which indicates whether card is present.
0: Card not present
1: Card present
(RO)

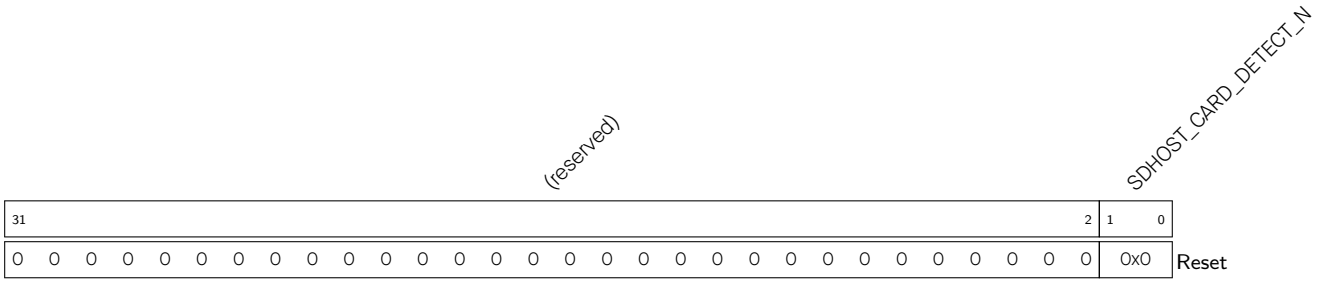
SDHOST_DATA_BUSY Represents inverted value of the sdhost_card_data[0] signal, which indicates card data is busy.
0: Card data not busy
1: Card data busy
(RO)

SDHOST_DATA_STATE_MC_BUSY Represents whether the data transmit or receive state-machine is busy.
0: Not busy
1: Busy
(RO)

SDHOST_RESPONSE_INDEX Represents index of previous response, including any auto-stop sent by core. (RO)

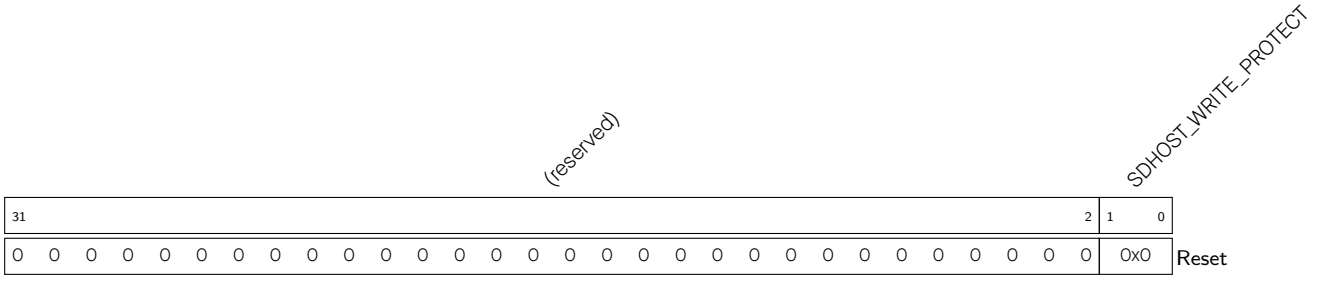
SDHOST_FIFO_COUNT Represents FIFO count, number of filled locations in FIFO. (RO)

Register 57.36. SDHOST_CDETECT_REG (0x0050)



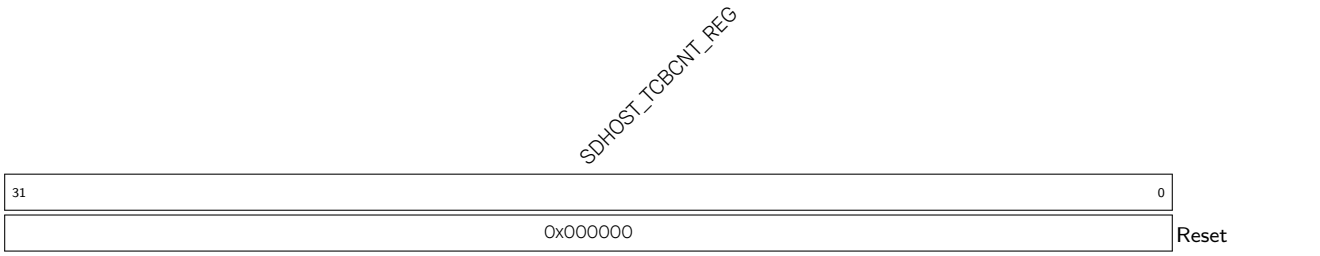
SDHOST_CARD_DETECT_N Represents value on sdhost_card_detect_n input ports. One bit per card. Bit[0] corresponds to card[0]. (RO)

Register 57.37. SDHOST_WRTPRT_REG (0x0054)



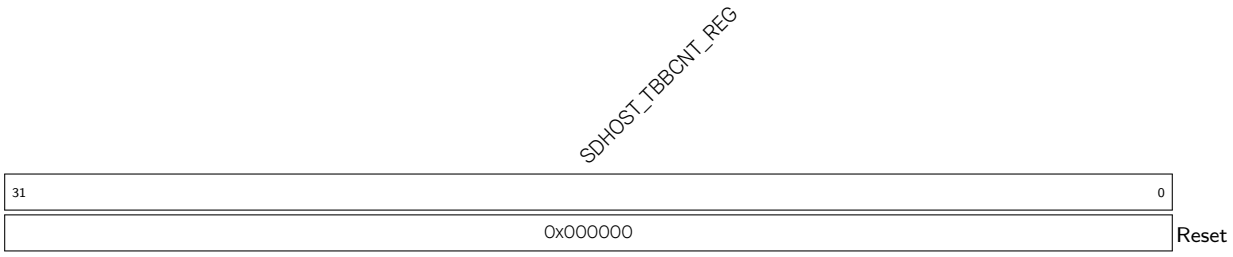
SDHOST_WRITE_PROTECT Represents value on sdhost_card_write_prt input ports, one bit per card. Bit[0] corresponds to card[0]. (RO)

Register 57.38. SDHOST_TCBCNT_REG (0x005C)



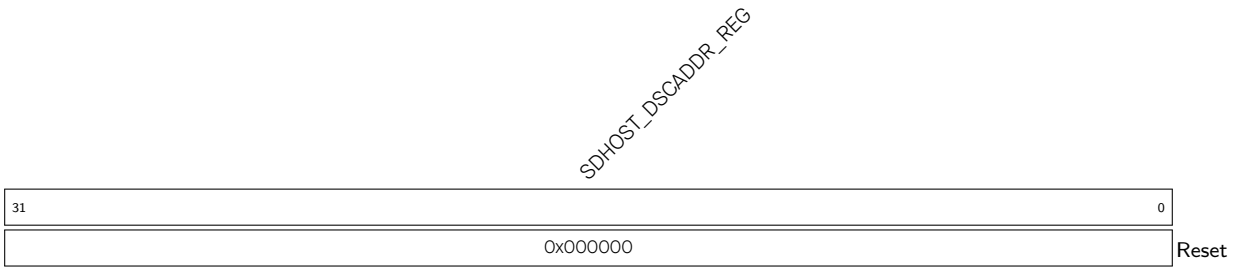
SDHOST_TCBCNT_REG Represents the number of bytes transferred by the CIU unit to the card. (RO)

Register 57.39. SDHOST_TBBCNT_REG (0x0060)



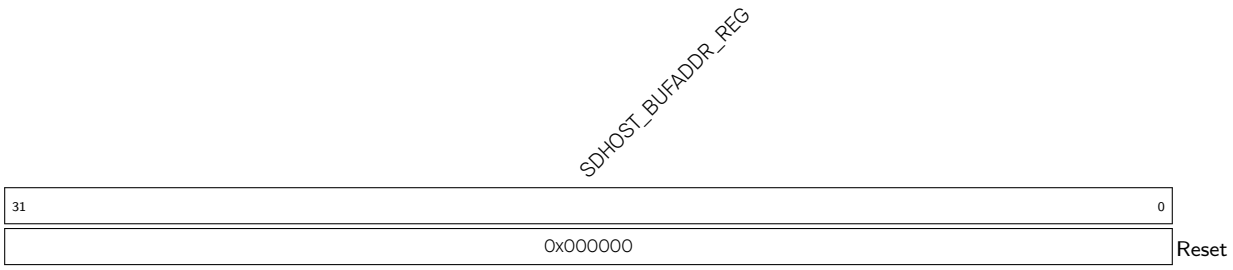
SDHOST_TBBCNT_REG Represents the number of bytes transferred between Host/DMA memory and BIU FIFO. (RO)

Register 57.40. SDHOST_DSCADDR_REG (0x0094)



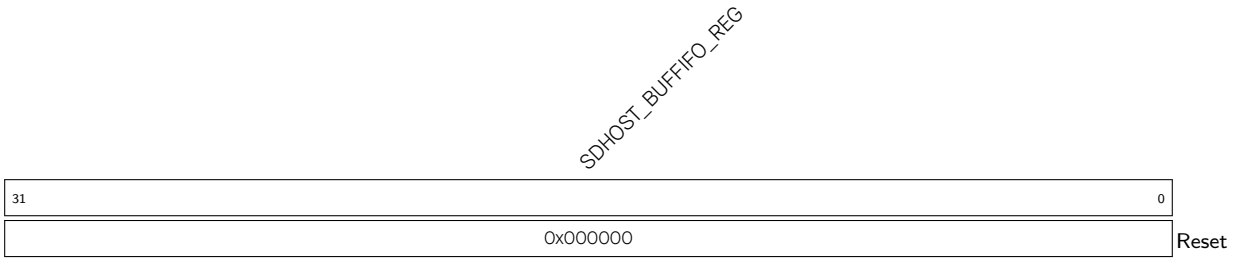
SDHOST_DSCADDR_REG Represents the Descriptor Address Pointer, points to the start address of current descriptor read by DMA. (RO)

Register 57.41. SDHOST_BUFADDR_REG (0x0098)



SDHOST_BUFADDR_REG Represents the Buffer Address Pointer, points to the current Data Buffer Address being accessed by the DMA. (RO)

Register 57.42. SDHOST_BUFFIFO_REG (0x0200)



SDHOST_BUFFIFO_REG Date received or to be transmitted between FIFO and software directly. This register points to the current Data FIFO. (R/W)

Chapter 58

LED PWM Controller (LEDC)

58.1 Overview

The LED PWM Controller is a peripheral designed to generate PWM signals for LED control. It has specialized features such as automatic duty cycle fading. However, the LED PWM Controller can also be used to generate PWM signals for other purposes.

58.2 Features

The LED PWM Controller has the following features:

- Eight independent PWM generators (i.e., eight channels)
- Maximum PWM duty cycle resolution: 20 bits
- Four independent timers that support fractional division
- Adjustable phase of PWM signal output
- PWM duty cycle dithering
- Automatic duty cycle fading — gradual increase/decrease of a PWM's duty cycle without interference from the processor. An interrupt will be generated upon fade completion
- Up to 16 duty cycle ranges for each PWM generator to generate gamma curve signals - each range can be independently configured in terms of fading direction (increase or decrease), fading amount (the amount by which the duty cycle increases or decreases each time), the number of fades (how many times the duty cycle fades in one range), and fading frequency
- PWM signal output in low-power mode (Light-sleep mode)
- Event generation and task response related to the Event Task Matrix (ETM) peripheral

Note that the four timers are identical regarding their features and operation. The following sections refer to the timers collectively as Timer*x* (where *x* ranges from 0 to 3). Likewise, the eight PWM generators are also identical in features and operation, and thus are collectively referred to as PWM*n* (where *n* ranges from 0 to 7).

58.3 Architectural Overview

Figure 58.3-1 shows the architecture of the LED PWM Controller.

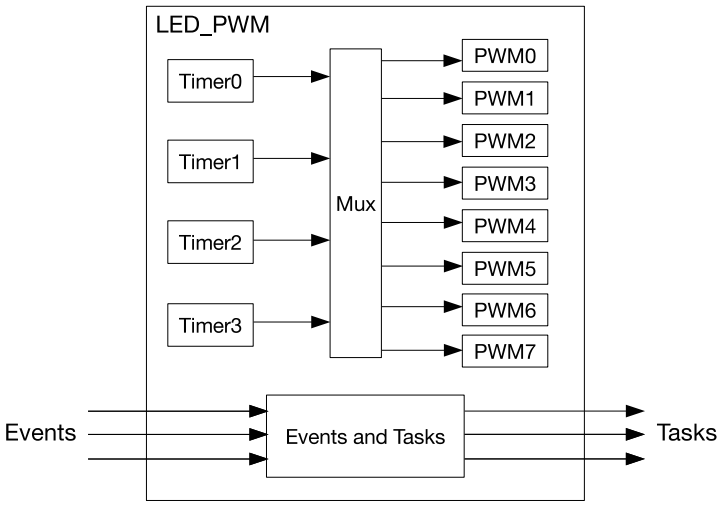


Figure 58.3-1. LED PWM Architecture

Each of the four timers has an internal timebase counter (i.e., a counter that counts on cycles of a reference clock) and thus can be independently configured (i.e., configurable clock divider, and counter overflow value). Each PWM generator selects one of the timers by configuring the LEDC_TIMER_SEL_CH*n*, and uses the timer's counter value timer*x*_cnt as a reference to generate its PWM signal.

Figure 58.3-2 illustrates the main functional blocks of the timer and the PWM generator.

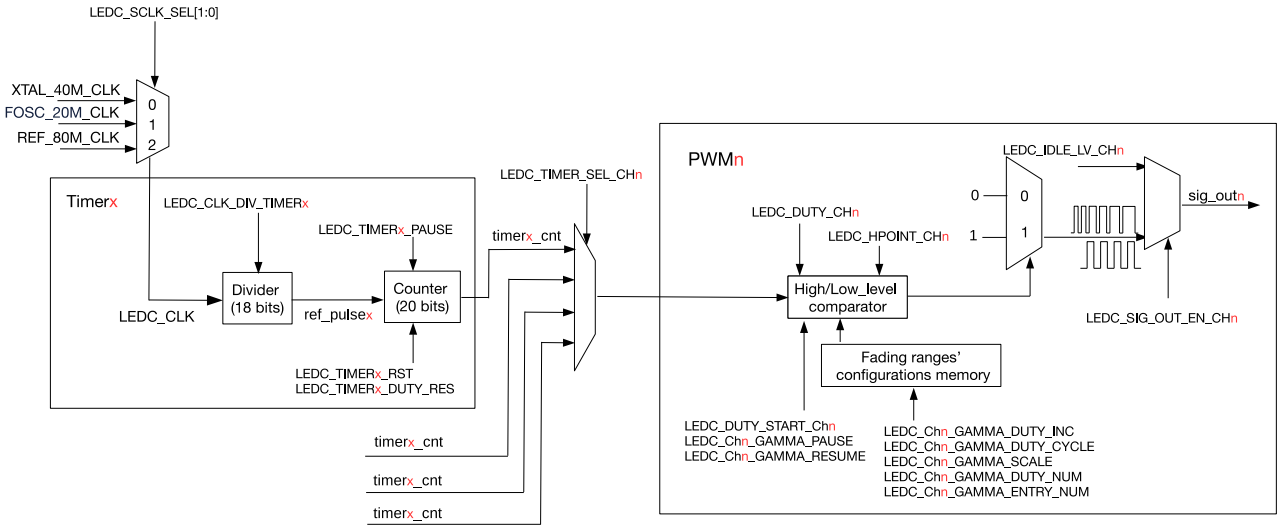


Figure 58.3-2. Timer and PWM Generator Block Diagram

58.4 Functional Description

58.4.1 Timers

Each timer in LED PWM Controller internally maintains a timebase counter. Referring to Figure 58.3-2, this clock signal used by the timebase counter is named ref_pulse*x*. All timers use the same clock source LEDC_CLK, which is then passed through a clock divider to generate ref_pulse*x* for the counter.

58.4.1.1 Clock Source

LED PWM registers configured by software are clocked by APB_CLK. To use the LED PWM peripheral, the APB_CLK signal going to the LED PWM has to be enabled. The APB_CLK signal to LED PWM can be enabled by setting the HP_SYS_CLKRST_LEDC_APB_CLK_EN field in the HP_SYS_CLKRST_SOC_CLK_CTRL3_REG register. The LEDC_CLK signal to LED PWM can be enabled by setting the HP_SYS_CLKRST_LEDC_CLK_EN field in the HP_SYS_CLKRST_PERI_CLK_CTRL22_REG register. The LED PWM peripheral can be reset via software by setting the HP_SYS_CLKRST_RST_EN_LEDC field in the HP_SYS_CLKRST_HP_RST_EN1_REG register.

Timers in the LED PWM Controller choose their common clock source from one of the following clock signals: XTAL_40M_CLK, FOSC_20M_CLK, and REF_80M_CLK. The procedure for selecting a clock source signal for LEDC_CLK is described below:

- XTAL_40M_CLK: Set PCR_LEDC_SCLK_SEL[1:0] to 0
- FOSC_20M_CLK: Set PCR_LEDC_SCLK_SEL[1:0] to 1
- REF_80M_CLK: Set PCR_LEDC_SCLK_SEL[1:0] to 2

The LEDC_CLK signal will then be passed through the clock divider.

For more information, please refer to Chapter 11 *Reset and Clock*.

58.4.1.2 Clock Divider Configuration

The LEDC_CLK signal is passed through a clock divider to generate the ref_pulse*x* signal for the counter. The frequency of ref_pulse*x* is equal to the frequency of LEDC_CLK divided by the divisor LEDC_CLK_DIV (see Figure 58.3-2).

The clock divider is a fractional divider. Thus, the divisor LEDC_CLK_DIV can be non-integer values. LEDC_CLK_DIV is configured according to the following equation.

$$\text{LEDC_CLK_DIV} = A + \frac{B}{256}$$

- The integer part *A* corresponds to the most significant 10 bits of LEDC_CLK_DIV_TIMER*x* (i.e., LEDC_TIMER*x*_CONF_REG[22:13])
- The fractional part *B* corresponds to the least significant 8 bits of LEDC_CLK_DIV_TIMER*x* (i.e., LEDC_TIMER*x*_CONF_REG[12:5])

When the fractional part *B* is 0, LEDC_CLK_DIV is equivalent to an integer divisor (i.e., an integer prescaler). In other words, a ref_pulse*x* clock pulse is generated after every *A* number of LEDC_CLK clock pulses.

However, when *B* is not 0, LEDC_CLK_DIV becomes a non-integer divisor. The clock divider implements non-integer frequency division by alternating between *A* and (*A*+1) LEDC_CLK clock pulses per ref_pulse*x* clock pulse. In this way, the average frequency of ref_pulse*x* clock pulse will be the desired frequency (i.e., the non-integer divided frequency). For every 256 ref_pulse*x* clock pulses:

- A number of *B* ref_pulse*x* clock pulses are generated every (*A*+1) LEDC_CLK clock pulses
- A number of (256-*B*) ref_pulse*x* clock pulses are generated every *A* LEDC_CLK clock pulses
- The ref_pulse*x* clock pulses generated every (*A*+1) pulses are evenly distributed amongst those generated every *A* pulses

Figure 58.4-1 illustrates the relation between LEDC_CLK clock pulses and ref_pulse*x* clock pulses when LEDC_CLK_DIV is a non-integer value.

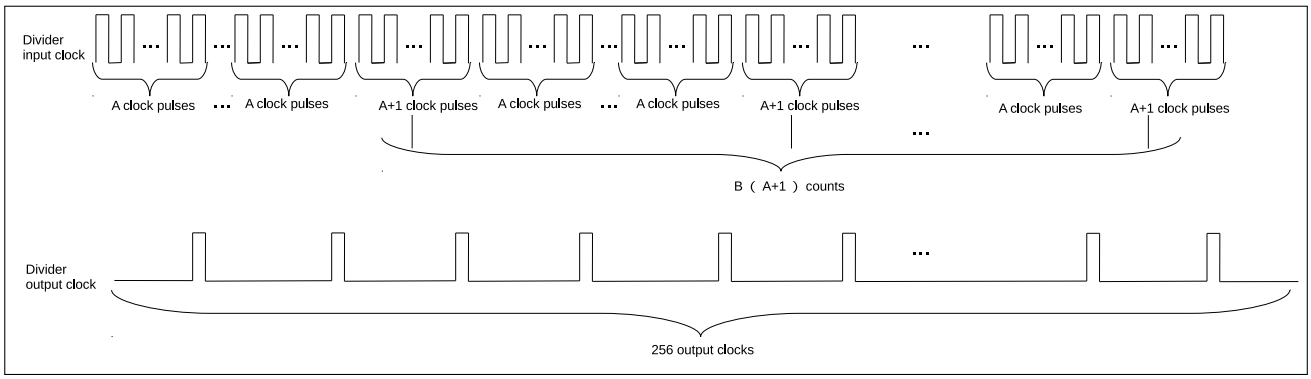


Figure 58.4-1. Frequency Division When LEDC_CLK_DIV is a Non-Integer Value

To change the timer's clock divisor at runtime, first configure the LEDC_CLK_DIV_TIMER*x* field, and then set the LEDC_TIMER*x*_PARA_UP field to apply the new configuration. This will cause the newly configured values to take effect upon the next overflow of the counter. The LEDC_TIMER*x*_PARA_UP field will be automatically cleared by hardware.

58.4.1.3 20-Bit Counter

Each timer contains a 20-bit timebase counter that uses ref_pulse*x* as its reference clock (see Figure 58.3-2). The LEDC_TIMER*x*_DUTY_RES field configures the actual used bit width of this 20-bit counter. Hence, the maximum resolution of the PWM signal is 20 bits. The counter counts up to $2^{\text{LEDC_TIMER}x\text{_DUTY_RES}} - 1$, overflows and begins counting from 0 again. The counter's value can be read, reset, and suspended by software. Figure 58.4-2 shows the relationship between the counter and PWM resolution.

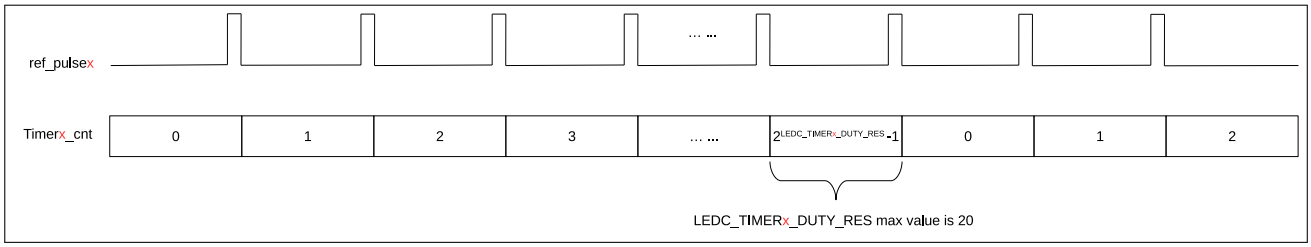


Figure 58.4-2. Relationship Between Counter And Resolution

Every time the counter overflows, it can trigger the LEDC_TIMER*x*_OVF_INT interrupt (generated automatically by hardware without configuration). It can also be configured to trigger LEDC_OVF_CNT_CH*n*_INT interrupt after overflowing LEDC_OVF_NUM_CH*n* + 1 times. To configure LEDC_OVF_CNT_CH*n*_INT interrupt, please:

1. Configure LEDC_TIMER_SEL_CH*n* to select the timer for the PWM generator
2. Enable the overflow counter by setting LEDC_OVF_CNT_EN_CH*n*
3. Configure LEDC_OVF_NUM_CH*n* with the number of counter overflows (that triggers an interrupt) minus 1
4. Enable the overflow interrupt by setting LEDC_OVF_CNT_CH*n*_INT_ENA
5. Configure LEDC_TIMER*x*_DUTY_RES to specify the counter bit width for the selected Timer*x* and wait for a LEDC_OVF_CNT_CH*n*_INT interrupt

To change the overflow value at runtime, first set the LEDC_TIMER*x*_DUTY_RES field, and then set the LEDC_TIMER*x*_PARA_UP field. This will cause the newly configured values to take effect upon the next overflow of the counter. If LEDC_OVF_CNT_EN_CH*n* field is reconfigured, LEDC_PARA_UP_CH*n* should be set to apply the new configuration. In summary, these configuration values need to be updated by setting LEDC_TIMER*x*_PARA_UP or LEDC_PARA_UP_CH*n*. LEDC_TIMER*x*_PARA_UP and LEDC_PARA_UP_CH*n* will be automatically cleared by hardware.

Referring to Figure 58.3-2, the frequency of a PWM generator output signal (sig_out*n*) is dependent on the frequency of the timer's clock source LEDC_CLK, the clock divisor LEDC_CLK_DIV, and the duty resolution (counter width) LEDC_TIMER*x*_DUTY_RES:

$$f_{\text{PWM}} = \frac{f_{\text{LEDC_CLK}}}{\text{LEDC_CLK_DIV} \cdot 2^{\text{LEDC_TIMER}x\text{_DUTY_RES}}}$$

Based on the formula above, the desired duty resolution can be calculated as follows:

$$\text{LEDC_TIMER}x\text{_DUTY_RES} = \log_2\left(\frac{f_{\text{LEDC_CLK}}}{f_{\text{PWM}} \cdot \text{LEDC_CLK_DIV}}\right)$$

Table 58.4-1 lists the commonly-used frequencies and their corresponding resolutions.

Table 58.4-1. Commonly-used Frequencies and Resolutions

LEDC_CLK	PWM Frequency	Highest Resolution (bit) [1]	Lowest Resolution (bit) [2]
REF_80M_CLK (80 MHz)	1 kHz	16	6
REF_80M_CLK (80 MHz)	5 kHz	13	3
REF_80M_CLK (80 MHz)	10 kHz	12	2
XTAL_40M_CLK (40 MHz)	1 kHz	15	5
XTAL_40M_CLK (40 MHz)	4 kHz	13	3
FOSC_20M_CLK (20 MHz)	1 kHz	14	4
FOSC_20M_CLK (20 MHz)	2 kHz	13	3

[1] The highest resolution is calculated when the clock divisor LEDC_CLK_DIV is 1. If the highest resolution calculated by the formula is higher than the counter's width 20 bits, then the highest resolution should be 20 bits.

[2] The lowest resolution is calculated when the clock divisor LEDC_CLK_DIV is $1023 + \frac{255}{256}$. If the lowest resolution calculated by the formula is lower than 0, then the lowest resolution should be 1.

58.4.2 PWM Generators

To generate a PWM signal, a PWM generator (PWM*n*) needs a timer (Timer*x*). Each PWM generator can be configured separately by setting LEDC_TIMER_SEL_CH*n* to use one of four timers to generate the PWM output.

As shown in Figure 58.3-2, each PWM generator has a comparator and two multiplexers. A PWM generator compares the timer's 20-bit counter value (Timer*x*_cnt) to two trigger values Hpoint*n* and Lpoint*n*. When the timer's counter value is equal to Hpoint*n* or Lpoint*n*, the PWM signal is high or low, respectively, as described below:

- If Timer*x*_cnt == Hpoint*n*, sig_out*n* is 1.
- If Timer*x*_cnt == Lpoint*n*, sig_out*n* is 0.

Figure 58.4-3 illustrates how Hpoint*n* and Lpoint*n* are used to generate a fixed duty cycle PWM output signal.

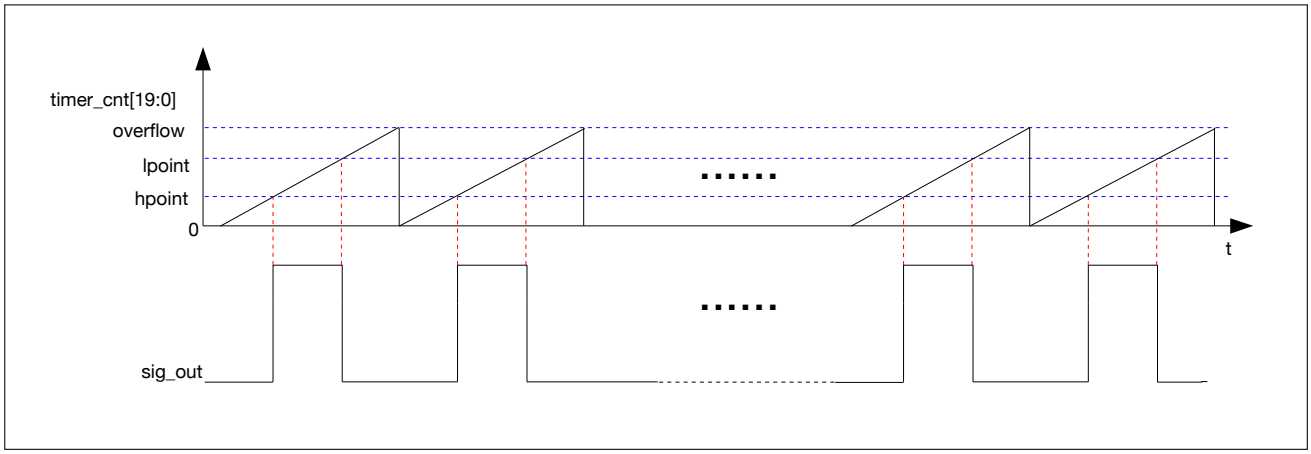


Figure 58.4-3. LED PWM Output Signal Diagram

For a particular PWM generator (PWM*n*), its Hpoint*n* is sampled from the LEDC_HPOINT_CH*n* field each time the selected timer's counter overflows. Likewise, Lpoint*n* is also sampled on every counter overflow and is calculated from the sum of the LEDC_DUTY_CH*n*[24:4] and LEDC_HPOINT_CH*n* fields. By setting Hpoint*n* and Lpoint*n* via the LEDC_HPOINT_CH*n* and LEDC_DUTY_CH*n*[24:4] fields, the relative phase and duty cycle of the PWM output can be set.

The PWM output signal (sig_out*n*) is enabled by setting LEDC_SIG_OUT_EN_CH*n*. When LEDC_SIG_OUT_EN_CH*n* is cleared, PWM signal output is disabled, and the output signal (sig_out*n*) will output a constant level specified by LEDC_IDLE_LV_CH*n*.

The bits LEDC_DUTY_CH*n*[3:0] are used to dither the duty cycles of the PWM output signal (sig_out*n*) by periodically altering the duty cycle of sig_out*n*. When LEDC_DUTY_CH*n*[3:0] is not 0, then for every 16 cycles of sig_out*n*, LEDC_DUTY_CH*n*[3:0] of those cycles will have PWM pulses that are one timer tick longer than the other (16- LEDC_DUTY_CH*n*[3:0]) cycles. For instance, if LEDC_DUTY_CH*n*[24:4] is set to 10 and LEDC_DUTY_CH*n*[3:0] is set to 5, then 5 of 16 cycles will have a PWM pulse with a duty value of 11 and the rest of the 16 cycles will have a PWM pulse with a duty value of 10. The average duty cycle after 16 cycles is 10.3125.

If fields LEDC_TIMER_SEL_CH*n*, LEDC_HPOINT_CH*n*, LEDC_DUTY_CH*n*[24:4], and LEDC_SIG_OUT_EN_CH*n* are reconfigured, LEDC_PARA_UP_CH*n* must be set to apply the new configuration. This will cause the newly configured values to take effect upon the next overflow of the counter. LEDC_PARA_UP_CH*n* field will be automatically cleared by hardware.

58.4.3 Duty Cycle Fading

The PWM generators can fade the duty cycle of a PWM output signal (i.e., gradually change the duty cycle from one value to another). Each PWM generator can have up to 16 duty cycle ranges, which can be independently configured in terms of fading direction (increase or decrease), fading amount, the number of fades, and fading frequency. If Duty Cycle Fading is enabled, every range's Lpoint*n* value will change

according to its fading configuration.

58.4.3.1 Linear Duty Cycle Fading

Linear fading PWM signals can be generated by configuring the direction, fading amount, the number of fades, and fading frequency of the first duty cycle range.

The programming procedures will be described in detail in the section 58.6.

After the procedures, the PWM generator can fade the duty cycle of a PWM signal once per LEDC_CH*n*_GAMMA_DUTY_CYCLE times of counter overflows. Every time when the PWM signal is faded, Lpoint*n* increases or decreases (configured by LEDC_CH*n*_GAMMA_DUTY_INC) by LEDC_CH*n*_GAMMA_SCALE, and the duty cycle increases or decreases (configured by LEDC_CH*n*_GAMMA_DUTY_INC) by

$$\frac{\text{LEDC_CH}n\text{_GAMMA_SCALE}}{\text{LEDC_TIMER}x\text{_DUTY_RES}}$$

The duty cycle is faded for LEDC_CH*n*_GAMMA_DUTY_NUM times. After that, the PWM generator stops fading and keeps outputting signals at this duty cycle. Upon each fading the duty cycle increases or decreases by the same amount, and therefore the PWM signal is a linear fading signal.

Figure 58.4-4 shows a linear fading PWM signal.

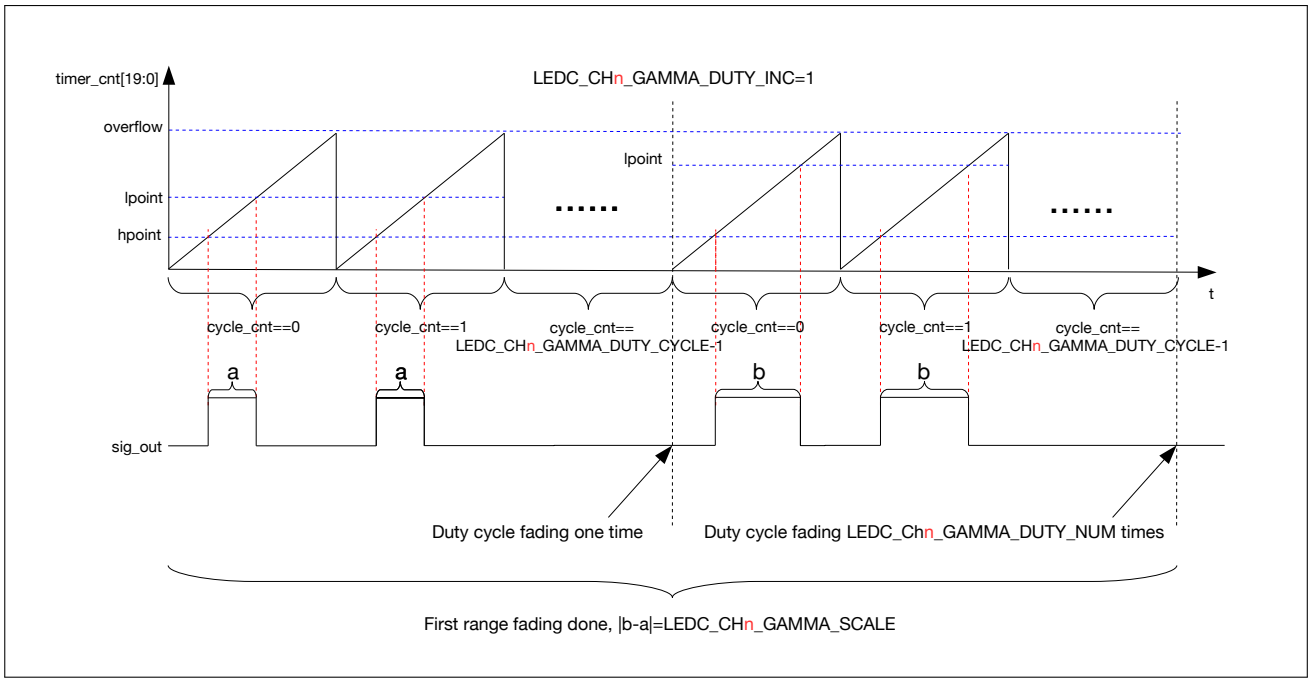


Figure 58.4-4. Output Signal of Linear Duty Cycle Fading

58.4.3.2 Gamma Curve Fading

Gamma curve fading PWM signals can be generated by configuring the fading direction, fading amount, the number of fades and fading frequency of multiple duty cycle fading ranges.

The programming procedures will be described in detail in the section 58.6.

After the procedures, the PWM generator can generate a PWM signal with LEDC_CH*n*_GAMMA_ENTRY_NUM ranges. The duty cycle of the PWM signal fades according to the configurations of range 0 first, and then range 1, till range (LEDC_CH*n*_GAMMA_ENTRY_NUM − 1) (the last range) where Duty Cycle Fading ends. The PWM signal fades independently in each range. According to the configuration of each range, every time when the counter overflows for LEDC_CH*n*_GAMMA_DUTY_CYCLE times, Lpoint*n* increases or decreases (configured by LEDC_CH*n*_GAMMA_DUTY_INC) by LEDC_CH*n*_GAMMA_SCALE, and accordingly the duty cycle increases or decreases (configured by LEDC_CH*n*_GAMMA_DUTY_INC) by

$$\frac{\text{LEDC_CH}n\text{_GAMMA_SCALE}}{\text{LEDC_TIMER}x\text{_DUTY_RES}}$$

After the duty cycle fades for LEDC_CH*n*_GAMMA_DUTY_NUM times in a range, Duty Cycle Fading in this range finishes.

When Duty Cycle Fading finishes in all ranges (the number of ranges is specified by LEDC_CH*n*_GAMMA_ENTRY_NUM), the PWM signal stops fading and keeps the duty cycle of the last fade. Given that the duty cycle fades differently and linearly in each range, several linear fading ranges would be fitted to a gamma curve.

Figure 58.4-5 illustrates a gamma curve fading PWM signal.

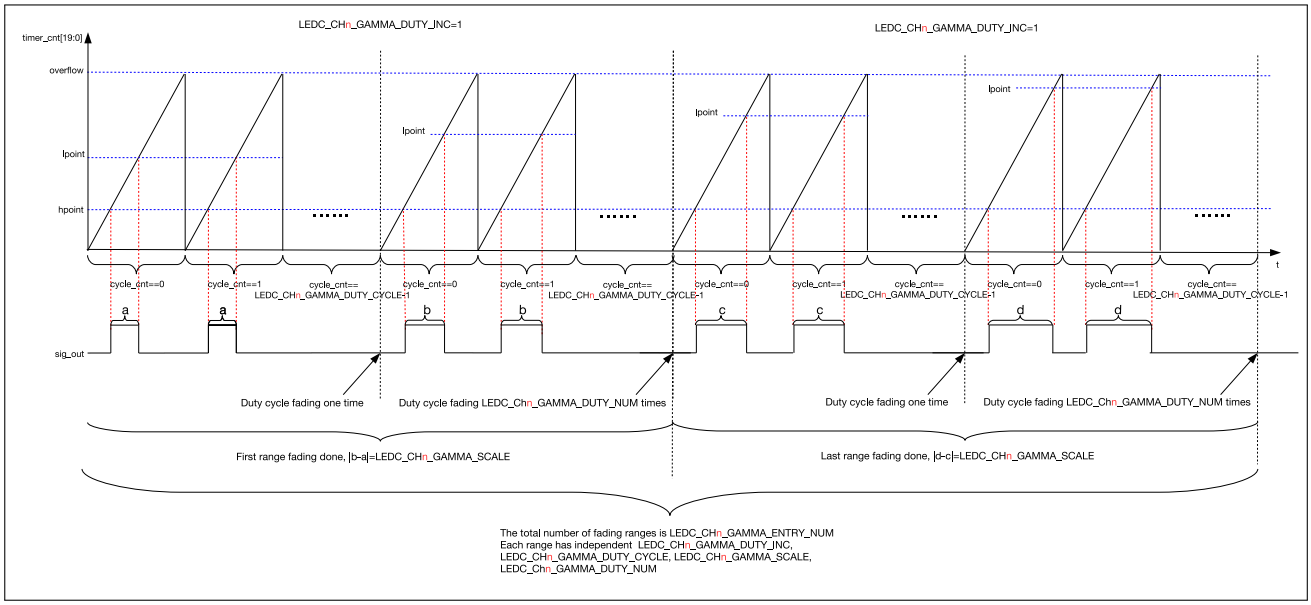


Figure 58.4-5. Output Signal of Gamma Curve Fading

58.4.3.3 Suspend and Resume Duty Cycle Fading

To suspend Duty Cycle Fading that has already been started, write 1 to the LEDC_CH*n*_GAMMA_PAUSE field of the LEDC_CH*n*_GAMMA_CONF_REG register. Once LEDC_CH*n*_GAMMA_PAUSE is set to 1, the PWM signal keeps the duty cycle of the most recent fade.

To resume Duty Cycle Fading, write 1 to the LEDC_CH*n*_GAMMA_RESUME field of the LEDC_CH*n*_GAMMA_CONF_REG register. Once LEDC_CH*n*_GAMMA_RESUME is set to 1, the PWM signal resumes fading from the range where the suspension occurs, until fading in the last range finishes. The fading will continue from the state when it was paused until all the ranges complete duty cycle fading (when

LEDC_CH*n*_GAMMA_RESUME is set to 1, LEDC_CH*n*_GAMMA_PAUSE is cleared automatically by hardware.

58.4.4 Event Task Matrix Feature

The LEDC on ESP32-P4 supports the Event Task Matrix (ETM) function, which allows LEDC's ETM tasks to be triggered by any peripherals' ETM events, or LEDC's ETM events to trigger any peripherals' ETM tasks. This section introduces the ETM tasks and events related to LEDC. For more information, please refer to Chapter 14 Event Task Matrix (ETM).

ETM-related events and tasks are enabled by configuring corresponding fields of LEDC_EVT_TASK_EN0_REG, LEDC_EVT_TASK_EN1_REG and LEDC_EVT_TASK_EN2_REG registers. For the correspondence between events, tasks, and fields, Please refer to Section 58.8).

LEDC can receive the following ETM tasks:

- LEDC_TASK_DUTY_SCALE_UPDATE_CH*n*: If the LEDC_TASK_DUTY_SCALE_UPDATE_CH*n*_EN field is enabled, upon receiving the LEDC_TASK_DUTY_SCALE_UPDATE_CH*n* task, PWM*n* generates fading PWM signals according to the newly configured LEDC_CH*n*_GAMMA_SCALE field.
- LEDC_TASK_TIMER*x*_RES_UPDATE: If the LEDC_TASK_TIMER*x*_RES_UPDATE_EN field is enabled, upon receiving the LEDC_TASK_TIMER*x*_RES_UPDATE task, Timer*x* updates its counter's overflow value to the value configured in the LEDC_TIMER*x*_DUTY_RES field at the next overflow of the counter.
- LEDC_TASK_TIMER*x*_CAP: If the LEDC_TASK_TIMER*x*_CAP_EN field is enabled, upon receiving the LEDC_TASK_TIMER*x*_CAP task, Timer*x* captures its counter's value, and stores the value into the LEDC_TIMER*x*_CNT_CAP field of register LEDC_TIMER*x*_CNT_CAP_REG.
- LEDC_TASK_SIG_OUT_DIS_CH*n*: If the LEDC_TASK_SIG_OUT_DIS_CH*n*_EN field is enabled, upon receiving the LEDC_TASK_SIG_OUT_DIS_CH*n* task, PWM*n*'s signal output is disabled, and the output signal (sig_out*n*) outputs a constant level as specified by field LEDC_IDLE_LV_CH*n*, as shown in Figure 58.3-2.
- LEDC_TASK_OVF_CNT_RST_CH*n*: If the LEDC_TASK_OVF_CNT_RST_CH*n*_EN field is enabled, upon receiving the LEDC_TASK_OVF_CNT_RST_CH*n* task, PWM*n* timer's overflow counter is reset to 0.
- LEDC_TASK_TIMER*x*_RST: If the LEDC_TASK_TIMER*x*_RST_EN field is enabled, upon receiving the LEDC_TASK_TIMER*x*_RST task, Timer*x*'s counter is reset to 0.
- LEDC_TASK_TIMER*x*_RESUME and LEDC_TASK_TIMER*x*_PAUSE: If the LEDC_TASK_TIMER*x*_PAUSE_RESUME_EN field is enabled, upon receiving the LEDC_TASK_TIMER*x*_RESUME and LEDC_TASK_TIMER*x*_PAUSE task, Timer*x* is suspended and resumed alternately. That is, when the task is received, Timer*x* is paused; and when the task is received again, Timer*x* is resumed.
- LEDC_TASK_GAMMA_RESTART_CH*n*: If the LEDC_TASK_GAMMA_RESTART_CH*n*_EN field is enabled, upon receiving the LEDC_TASK_GAMMA_RESTART_CH*n* task, the PWM*n* restarts to generate the fading PWM signal.
- LEDC_TASK_GAMMA_PAUSE_CH*n*: If the LEDC_TASK_GAMMA_PAUSE_CH*n*_EN field is enabled, upon receiving the LEDC_TASK_GAMMA_PAUSE_CH*n* task, PWM*n* suspends Duty Cycle Fading at the next timer overflow. That is, after the task has been received, PWM*n* keeps the duty cycle of the last fade.

- LEDC_TASK_GAMMA_RESUME_CH*n*: If the LEDC_TASK_GAMMA_RESUME_CH*n*_EN field is enabled, upon receiving the LEDC_TASK_GAMMA_RESUME_CH*n* task, PWM*n* resumes Duty Cycle Fading at the next counter overflow. That is, after the task has been received, PWM*n* resumes fading from the range where the suspension occurs.

LEDC can generate the following ETM events:

- LEDC_EVT_DUTY_CHNG_END_CH*n*: Generated when the LEDC_EVT_DUTY_CHNG_END_CH*n*_EN field is enabled, and PWM*n* has finished Duty Cycle Fading.
- LEDC_EVT_OVF_CNT_PLS_CH*n*: Generated when the LEDC_EVT_OVF_CNT_PLS_CH*n*_EN field is enabled and when PWM*n* timer's counter overflows for LEDC_OVF_NUM_CH*n* + 1 times.
- LEDC_EVT_TIME_OVF_TIMER*x*: Generated when the LEDC_EVT_TIME_OVF_TIMER*x*_EN field is enabled and Timer*x*'s counter overflows.
- LEDC_EVT_TIMER*x*_CMP: Generated when the LEDC_EVT_TIMER*x*_CMP_EN field is enabled and the value of Timer*x*)'s counter reaches that of the LEDC_TIMER*x*_CMP field of register LEDC_TIMER*x*_CMP_REG.

In practical applications, LEDC's ETM events can trigger its own ETM tasks. For example, LEDC_EVT_DUTY_CHNG_END_CH*n* event can trigger the LEDC_TASK_GAMMA_RESTART_CH*n* task, thus starting the next fading directly after the current fading is completed.

58.5 Interrupts

ESP32-P4's LEDC can generate the LEDC_INT interrupt signal, which will be sent to the Interrupt Matrix.

Interrupt signal LEDC_INT can be generated by the following internal interrupt sources:

- LEDC_OVF_CNT_CH*n*_INT: Triggered when the timer counter overflows for LEDC_OVF_NUM_CH*n* + 1 times and the register LEDC_OVF_CNT_EN_CH*n* is set to 1.
- LEDC_DUTY_CHNG_END_CH*n*_INT: Triggered when a fade on PWM*n* has finished.
- LEDC_TIMER*x*_OVF_INT: Triggered when Timer*x* has reached its maximum counter value.

The above interrupt sources can only be triggered when the interrupt enable register *XXX*_ENA is set to 1. Write 1 to *XXX*_CLR will clear the interrupt (*XXX* is the interrupt source name).

58.6 Programming Procedures

The configuration process of the LED PWM controller to generate PWM signals is as follows:

1. Configure the Timer*x* according to section 58.4.1. After that, the clock source, divider, and counter are ready.
2. Configure the PWM*n* generator according to section 58.4.2. After that, PWM*n* selects one of the four timers and the phase and duty cycle of the PWM*n* output is set.
3. Choose the corresponding configuration process based on the different duty cycle fading types:
   (a) Linear duty cycle fading:
      i. Configure the LEDC_DUTY_CH*n* field with the initial value of Lpoint*n*.

ii. Set the LEDC_DUTY_START_CH*n* field to enable Duty Cycle Fading. When this field is cleared, Duty Cycle Fading will be disabled.

iii. Write the linear duty cycle fading configuration to (LEDC_CH*n*_MEM starting address + duty cycle range number * 4), the LEDC_CH*n*_MEM starting address is provided in Table 58.7-1. Please note that the following bit names are just for easier description, and there are no such register fields.

- Bit 0 of the configuration data is used to configure the direction (hereafter referred to as LEDC_CH*n*_GAMMA_DUTY_INC). When it is set or cleared, the Lpoint*n* will increase or decrease in the current configured range.
- Bit 1 to 10 of the configuration data is used to configure the number of times the counter overflows per an increment or decrement of Lpoint*n* (hereafter referred to as LEDC_CH*n*_GAMMA_DUTY_CYCLE). In other words, Lpoint*n* will increase or decrease after the counter overflows for the configured number of times.
- Bit 11 to 20 of the configuration data is used to configure the amount by which Lpoint*n* increase or decrease in the current configured range (hereafter referred to as LEDC_CH*n*_GAMMA_SCALE).
- Bit 21 to 30 of the configuration data is used to configure the number of fades in the current configured range (hereafter referred to as LEDC_CH*n*_GAMMA_DUTY_NUM).
- The duty cycle range number (from 0 to 15) specifies to which range the above configuration data apply. For linear duty cycle fading only the first range needs to be configured, so the duty cycle range number should be configured as 0.

iv. Configure the number of ranges per each fading (1 in this case) via the LEDC_CH*n*_GAMMA_ENTRY_NUM field of the LEDC_CH*n*_GAMMA_CONF_REG. Once the specified number of ranges have been faded, Duty Cycle Fading stops and the PWM generator triggers the LEDC_DUTY_CHNG_END_CH*n*_INT interrupt. For linear duty cycle fading there is only one duty cycle range (i.e., the first one), so configure LEDC_CH*n*_GAMMA_ENTRY_NUM as 1.

v. Set the LEDC_PARA_UP_CH*n* field to apply the above configurations. After this field is set, the configurations for Duty Cycle Fading will take effect upon the next overflow of the counter, and the PWM generator will output a linear fading PWM signal following configurations. LEDC_PARA_UP_CH*n* field will be automatically cleared by hardware.

(b) Gamma curve fading:

i. The same as Step 1 in Section 58.4.3.1.

ii. The same as Step 2 in Section 58.4.3.1.

iii. Configure multiple duty cycle ranges with the following steps. Write the gamma curve fading configuration to (LEDC_CH*n*_MEM starting address + duty cycle range number * 4), the LEDC_CH*n*_MEM starting address is provided in Table 58.7-1.

- Bit 0 of the configuration data is used to configure the direction (hereafter referred to as LEDC_CH*n*_GAMMA_DUTY_INC). When it is set or cleared, the Lpoint*n* will increment or decrement in the current configured range.

- Bit 1 to 10 of the configuration data is used to configure the number of times the counter overflows per an increase or decrease of Lpoint*n* (hereafter referred to as LEDC_CH*n*_GAMMA_DUTY_CYCLE). In other words, Lpoint*n* will increase or decrease after the counter overflows for the configured number of times.
- Bit 11 to 20 of the configuration data is used to configure the amount by which Lpoint*n* increase or decrease in the current configured range (hereafter referred to as LEDC_CH*n*_GAMMA_SCALE).
- Bit 21 to 30 of the configuration data is used to configure the number of fades in the current configured range (hereafter referred to as LEDC_CH*n*_GAMMA_DUTY_NUM).
- The duty cycle range number (from 0 to 15) specifies to which range the above configuration data apply. For gamma curve fading, it must start from 0 and increase by 1 for the next range to be configured.
- Once the above procedures are finished, the configuration for one range is complete. Other ranges are configured by repeating the same set of procedures. You can configure any number of ranges from 0 to 15, and each can be configured independently.

iv. After all required ranges are configured, write the total number of ranges configured in Step 3 to the LEDC_CH*n*_GAMMA_ENTRY_NUM field of the LEDC_CH*n*_GAMMA_CONF_REG register.

v. Set the LEDC_PARA_UP_CH*n* field to apply the above configuration. After this field is set, the configurations for duty cycle fading will take effect upon the next overflow of the counter, and the PWM generator will output a gamma curve fading PWM signal following the configurations. LEDC_PARA_UP_CH*n* field will be automatically cleared by hardware.

After the above procedures, the LED PWM controller will generate the desired PWM signals.

At any time, duty cycle fading can be suspended or resumed, more details can be found in setction 58.4.3.3. If the duty cycle fading configurations need to be changed before the PWM signals stop fading (that is, the LEDC_DUTY_CHNG_END_CH*n*_INT has not been triggered), the duty cycle fading needs to be suspended before changing the configurations. However, if the duty cycle fading configurations need to be changed after the PWM signals stop fading (that is, the LEDC_DUTY_CHNG_END_CH*n*_INT has been triggered), the configurations can be changed directly.

At any time, PWM signal output can be enabled or disabled by software, and the output signal level can be changed, more details can be found in section 58.4.2.

58.7 Memory Blocks

Each LEDC channel has a memory to store duty cycle fading configuration data. Each memory can store configuration data for up to 16 ranges. The addresses in this section are relative to LED PWM Controller base address provided in Table 9.3-2 in Chapter 9 *System and Memory*.

Table 58.7-1. LEDC Memory Blocks

Name	Description	Size	Starting Address	Ending Address	Access
LEDC_CH*n*_MEM	Memory of PWM*n*	64 B	0x400 + PWM*n* * 64	0x400 + (PWM*n* + 1) * 64	WR

58.8 Register Summary

The addresses in this section are relative to LED PWM Controller base address provided in Table 9.3-2 in Chapter 9 *System and Memory*.

The abbreviations given in Column **Access** are explained in Section *Access Types for Registers*.

Name	Description	Address	Access
Configuration Register			
LEDC_CH0_CONF0_REG	Configuration register 0 for channel 0	0x0000	varies
LEDC_CH0_HPOINT_REG	High point register for channel 0	0x0004	R/W
LEDC_CH0_DUTY_REG	Initial duty cycle register for channel 0	0x0008	R/W
LEDC_CH0_CONF1_REG	Configuration register 1 for channel 0	0x000C	R/W/SC
LEDC_CH1_CONF0_REG	Configuration register 0 for channel 1	0x0014	varies
LEDC_CH1_HPOINT_REG	High point register for channel 1	0x0018	R/W
LEDC_CH1_DUTY_REG	Initial duty cycle register for channel 1	0x001C	R/W
LEDC_CH1_CONF1_REG	Configuration register 1 for channel 1	0x0020	R/W/SC
LEDC_CH2_CONF0_REG	Configuration register 0 for channel 2	0x0028	varies
LEDC_CH2_HPOINT_REG	High point register for channel 2	0x002C	R/W
LEDC_CH2_DUTY_REG	Initial duty cycle register for channel 2	0x0030	R/W
LEDC_CH2_CONF1_REG	Configuration register 1 for channel 2	0x0034	R/W/SC
LEDC_CH3_CONF0_REG	Configuration register 0 for channel 3	0x003C	varies
LEDC_CH3_HPOINT_REG	High point register for channel 3	0x0040	R/W
LEDC_CH3_DUTY_REG	Initial duty cycle register for channel 3	0x0044	R/W
LEDC_CH3_CONF1_REG	Configuration register 1 for channel 3	0x0048	R/W/SC
LEDC_CH4_CONF0_REG	Configuration register 0 for channel 4	0x0050	varies
LEDC_CH4_HPOINT_REG	High point register for channel 4	0x0054	R/W
LEDC_CH4_DUTY_REG	Initial duty cycle register for channel 4	0x0058	R/W
LEDC_CH4_CONF1_REG	Configuration register 1 for channel 4	0x005C	R/W/SC
LEDC_CH5_CONF0_REG	Configuration register 0 for channel 5	0x0064	varies
LEDC_CH5_HPOINT_REG	High point register for channel 5	0x0068	R/W
LEDC_CH5_DUTY_REG	Initial duty cycle register for channel 5	0x006C	R/W
LEDC_CH5_CONF1_REG	Configuration register 1 for channel 5	0x0070	R/W/SC
LEDC_CH6_CONF0_REG	Configuration register 0 for channel 6	0x0078	varies
LEDC_CH6_HPOINT_REG	High point register for channel 6	0x007C	R/W
LEDC_CH6_DUTY_REG	Initial duty cycle register for channel 6	0x0080	R/W
LEDC_CH6_CONF1_REG	Configuration register 1 for channel 6	0x0084	R/W/SC
LEDC_CH7_CONF0_REG	Configuration register 0 for channel 7	0x008C	varies
LEDC_CH7_HPOINT_REG	High point register for channel 7	0x0090	R/W
LEDC_CH7_DUTY_REG	Initial duty cycle register for channel 7	0x0094	R/W
LEDC_CH7_CONF1_REG	Configuration register 1 for channel 7	0x0098	R/W/SC
LEDC_TIMER0_CONF_REG	Timer 0 configuration register	0x00A0	varies
LEDC_TIMER1_CONF_REG	Timer 1 configuration register	0x00A8	varies
LEDC_TIMER2_CONF_REG	Timer 2 configuration register	0x00B0	varies
LEDC_TIMER3_CONF_REG	Timer 3 configuration register	0x00B8	varies

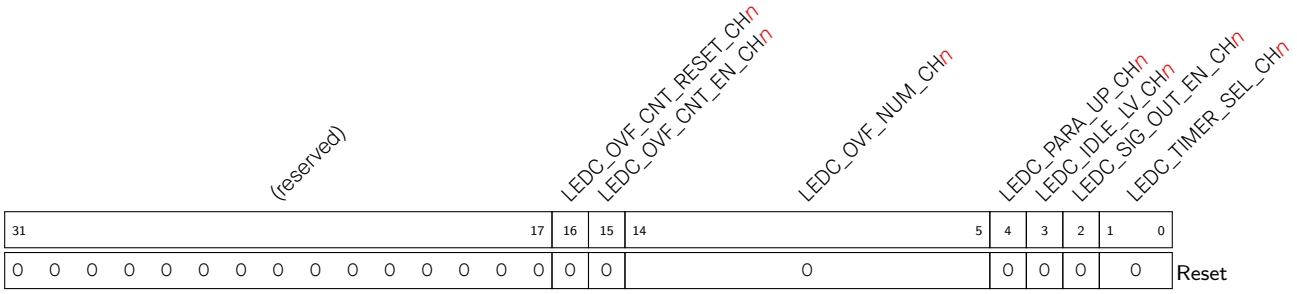
Name	Description	Address	Access
LEDC_CH0_GAMMA_CONF_REG	LEDC channel 0 gamma config register	0x0100	varies
LEDC_CH1_GAMMA_CONF_REG	LEDC channel 1 gamma config register	0x0104	varies
LEDC_CH2_GAMMA_CONF_REG	LEDC channel 2 gamma config register	0x0108	varies
LEDC_CH3_GAMMA_CONF_REG	LEDC channel 3 gamma config register	0x010C	varies
LEDC_CH4_GAMMA_CONF_REG	LEDC channel 4 gamma config register	0x0110	varies
LEDC_CH5_GAMMA_CONF_REG	LEDC channel 5 gamma config register	0x0114	varies
LEDC_CH6_GAMMA_CONF_REG	LEDC channel 6 gamma config register	0x0118	varies
LEDC_CH7_GAMMA_CONF_REG	LEDC channel 7 gamma config register	0x011C	varies
LEDC_EVT_TASK_EN0_REG	LEDC event task enable bit register 0	0x0120	R/W
LEDC_EVT_TASK_EN1_REG	LEDC event task enable bit register 1	0x0124	R/W
LEDC_EVT_TASK_EN2_REG	LEDC event task enable bit register 2	0x0128	R/W
LEDC_TIMER0_CMP_REG	LEDC timer 0 compare value register	0x0140	R/W
LEDC_TIMER1_CMP_REG	LEDC timer 1 compare value register	0x0144	R/W
LEDC_TIMER2_CMP_REG	LEDC timer 2 compare value register	0x0148	R/W
LEDC_TIMER3_CMP_REG	LEDC timer 3 compare value register	0x014C	R/W
LEDC_CONF_REG	LEDC global configuration register	0x0170	R/W
Status Register			
LEDC_CH0_DUTY_R_REG	Current duty cycle register for channel 0	0x0010	RO
LEDC_CH1_DUTY_R_REG	Current duty cycle register for channel 1	0x0024	RO
LEDC_CH2_DUTY_R_REG	Current duty cycle register for channel 2	0x0038	RO
LEDC_CH3_DUTY_R_REG	Current duty cycle register for channel 3	0x004C	RO
LEDC_CH4_DUTY_R_REG	Current duty cycle register for channel 4	0x0060	RO
LEDC_CH5_DUTY_R_REG	Current duty cycle register for channel 5	0x0074	RO
LEDC_CH6_DUTY_R_REG	Current duty cycle register for channel 6	0x0088	RO
LEDC_CH7_DUTY_R_REG	Current duty cycle register for channel 7	0x009C	RO
LEDC_TIMER0_VALUE_REG	Timer 0 current counter value register	0x00A4	RO
LEDC_TIMER1_VALUE_REG	Timer 1 current counter value register	0x00AC	RO
LEDC_TIMER2_VALUE_REG	Timer 2 current counter value register	0x00B4	RO
LEDC_TIMER3_VALUE_REG	Timer 3 current counter value register	0x00BC	RO
LEDC_TIMER0_CNT_CAP_REG	LEDC timer 0 captured count value register	0x0150	RO
LEDC_TIMER1_CNT_CAP_REG	LEDC timer 1 captured count value register	0x0154	RO
LEDC_TIMER2_CNT_CAP_REG	LEDC timer 2 captured count value register	0x0158	RO
LEDC_TIMER3_CNT_CAP_REG	LEDC timer 3 captured count value register	0x015C	RO
Interrupt Register			
LEDC_INT_RAW_REG	Interrupt raw status register	0x00C0	R/WTC/SS
LEDC_INT_ST_REG	Interrupt masked status register	0x00C4	RO
LEDC_INT_ENA_REG	Interrupt enable register	0x00C8	R/W
LEDC_INT_CLR_REG	Interrupt clear register	0x00CC	WT
Version Register			
LEDC_DATE_REG	Version control register	0x0174	R/W

58.9 Registers

The addresses in this section are relative to LED PWM Controller base address provided in Table 9.3-2 in Chapter 9 *System and Memory*.

For how to program reserved fields, please refer to Section Programming Reserved Register Field.

Register 58.1. LEDC_CH*n*_CONF0_REG (*n*: 0-7) (0x0000+0x14n*)**



LEDC_TIMER_SEL_CH*n* Configures which timer is channel *n* selected.

0: Select Timer 0

1: Select Timer 1

2: Select Timer 2

3: Select Timer 3

(R/W)

LEDC_SIG_OUT_EN_CH*n* Configures whether to enable signal output on channel *n*.

0: Signal output disable

1: Signal output enable

(R/W)

LEDC_IDLE_LV_CH*n* Configures the output value when channel *n* is inactive. Valid only when LEDC_SIG_OUT_EN_CH*n* is 0.

0: Output level is low

1: Output level is high

(R/W)

LEDC_PARA_UP_CH*n* Configures whether to update LEDC_HPOINT_CH*n*, LEDC_DUTY_START_CH*n*, LEDC_SIG_OUT_EN_CH*n*, LEDC_TIMER_SEL_CH*n*, LEDC_OVF_CNT_EN_CH*n* fields and duty cycle range configurations for channel *n*, and will be automatically cleared by hardware.

0: Invalid. No effect

1: Update

(WT)

LEDC_OVF_NUM_CH*n* Configures the maximum times of overflow minus 1. The LEDC_OVF_CNT_CH*n*_INT interrupt will be triggered when channel *n* overflows for (LEDC_OVF_NUM_CH*n* + 1) times. (R/W)

LEDC_OVF_CNT_EN_CH*n* Configures whether to enable the ovf_cnt of channel *n*.

0: Disable

1: Enable

(R/W)

Continued on the next page...

Register 58.1. LEDC_CH*n*_CONF0_REG (*n*: 0-7) (0x0000+0x14**n*)

Continued from the previous page...

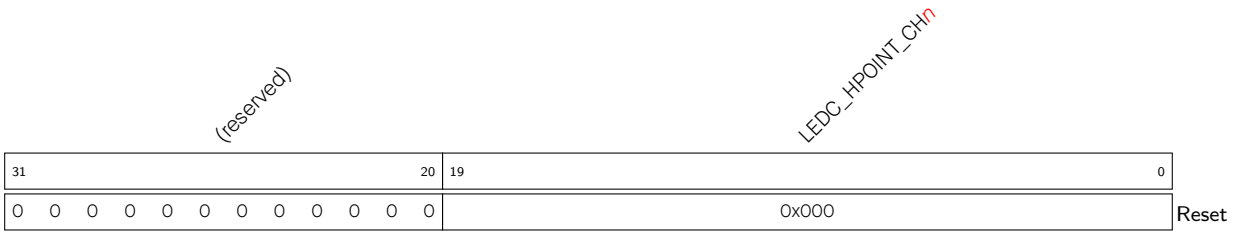
LEDC_OVF_CNT_RESET_CH*n* Configures whether to reset the ovf_cnt of channel *n*.

0: Invalid. No effect

1: Reset the ovf_cnt

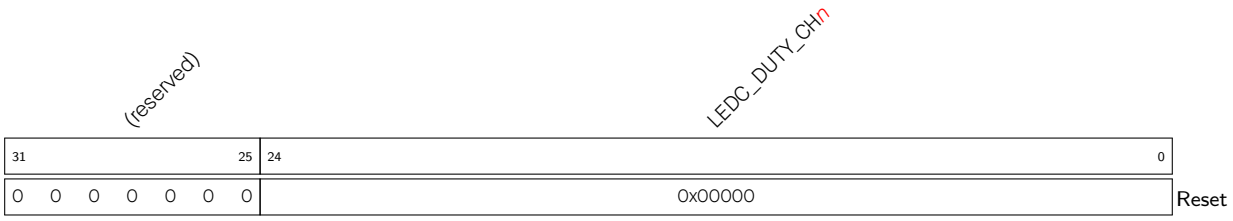
(WT)

Register 58.2. LEDC_CH*n*_HPOINT_REG (*n*: 0-7) (0x0004+0x14**n*)



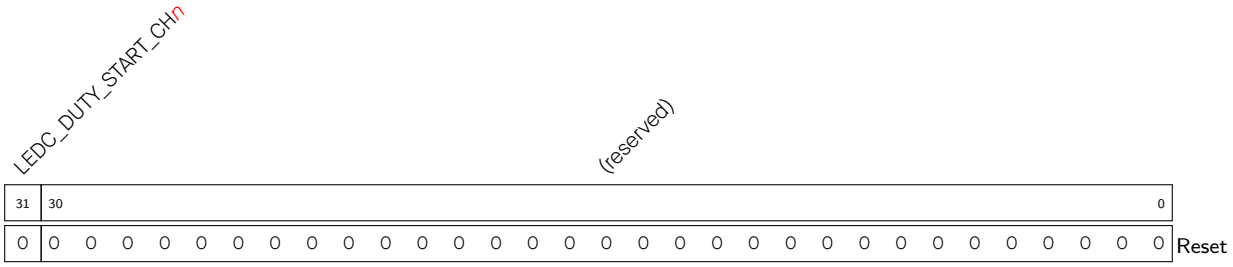
LEDC_HPOINT_CH*n* Configures high point of signal output on channel *n*. The output value changes to high when the selected timers have reached the value specified by this register. (R/W)

Register 58.3. LEDC_CH*n*_DUTY_REG (*n*: 0-7) (0x0008+0x14**n*)



LEDC_DUTY_CH*n* Configures the duty cycle of signal output on channel *n*. (R/W)

Register 58.4. LEDC_CH*n*_CONF1_REG (*n*: 0-7) (0x000C+0x14**n*)



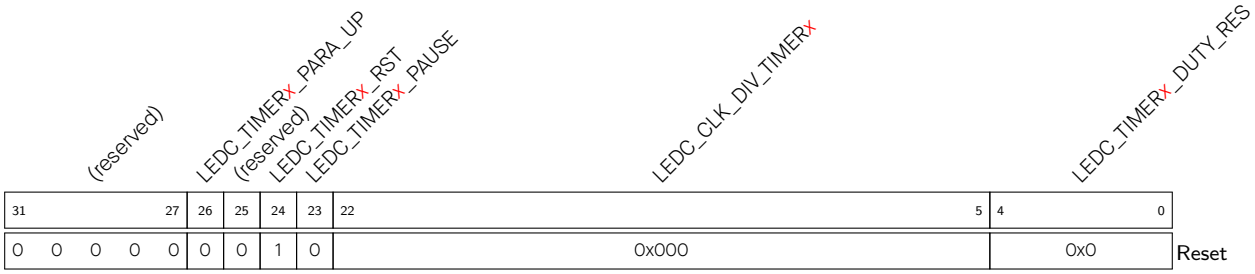
LEDC_DUTY_START_CH*n* Configures whether the duty cycle fading configurations take effect.

0: Not take effect

1: Take effect

(R/W/SC)

Register 58.5. LEDC_TIMER*x*_CONF_REG (*x*: 0-3) (0x00A0+0x8**x*)



LEDC_TIMER*x*_DUTY_RES Configures the bit width of the counter in timer *x*. Valid values are 1 to 20. (R/W)

LEDC_CLK_DIV_TIMER*x* Configures the divisor for the divider in timer *x*.

The least significant eight bits represent the fractional part. The most significant ten bits represent the integer part. (R/W)

LEDC_TIMER*x*_PAUSE Configures whether to pause the counter in timer *x*.

0: Normal

1: Pause

(R/W)

LEDC_TIMER*x*_RST Configures whether to reset timer *x*. The counter will show 0 after reset.

0: Not reset

1: Reset

(R/W)

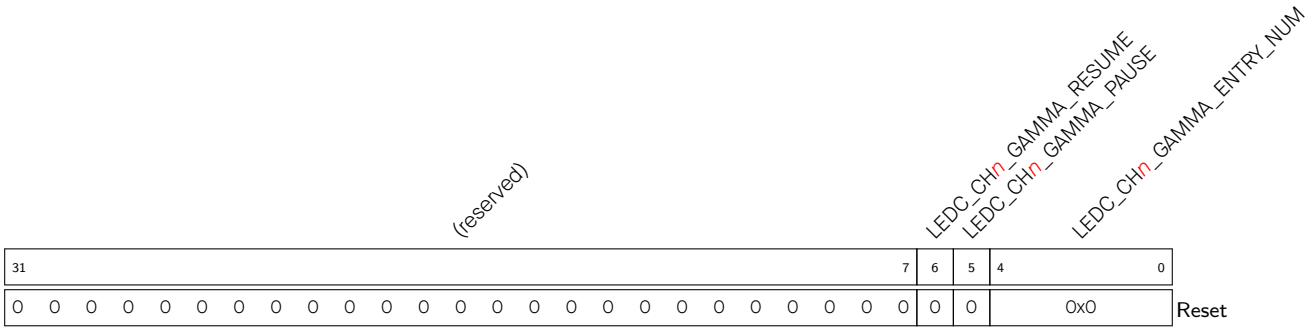
LEDC_TIMER*x*_PARA_UP Configures whether to update LEDC_CLK_DIV_TIMER*x* and LEDC_TIMER*x*_DUTY_RES.

0: Invalid. No effect

1: Update

(WT)

Register 58.6. LEDC_CH*n*_GAMMA_CONF_REG (*n*: 0-7) (0x0100+0x4n*)**



LEDC_CH*n*_GAMMA_ENTRY_NUM Configures the number of duty cycle fading ranges for LEDC channel *n*. (R/W)

LEDC_CH*n*_GAMMA_PAUSE Configures whether to pause duty cycle fading of LEDC channel *n*.

0: Invalid. No effect

1: Pause

(WT)

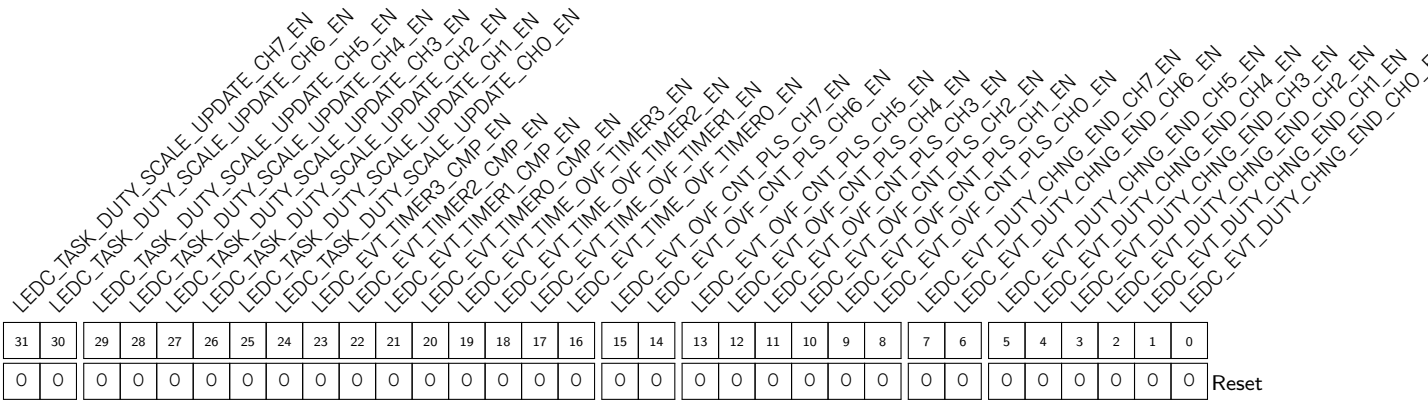
LEDC_CH*n*_GAMMA_RESUME Configures whether to resume duty cycle fading of LEDC channel *n*.

0: Invalid. No effect

1: Resume

(WT)

Register 58.7. LEDC_EVT_TASK_EN0_REG (0x0120)



LEDC_EVT_DUTY_CHNG_END_CH*n*_EN (*n*: 0-7) Configures whether to enable the LEDC_EVT_DUTY_CHNG_END_CH*n* event.

0: Disable

1: Enable

(R/W)

LEDC_EVT_OVF_CNT_PLS_CH*n*_EN (*n*: 0-7) Configures whether to enable the LEDC_EVT_OVF_CNT_PLS_CH*n* event.

0: Disable

1: Enable

(R/W)

LEDC_EVT_TIME_OVF_TIMER*x*_EN (*x*: 0-3) Configures whether to enable the LEDC_EVT_TIME_OVF_TIMER*x* event.

0: Disable

1: Enable

(R/W)

LEDC_EVT_TIMER*x*_CMP_EN (*x*: 0-3) Configures whether to enable the LEDC_EVT_TIMER*x*_CMP event.

0: Disable

1: Enable

(R/W)

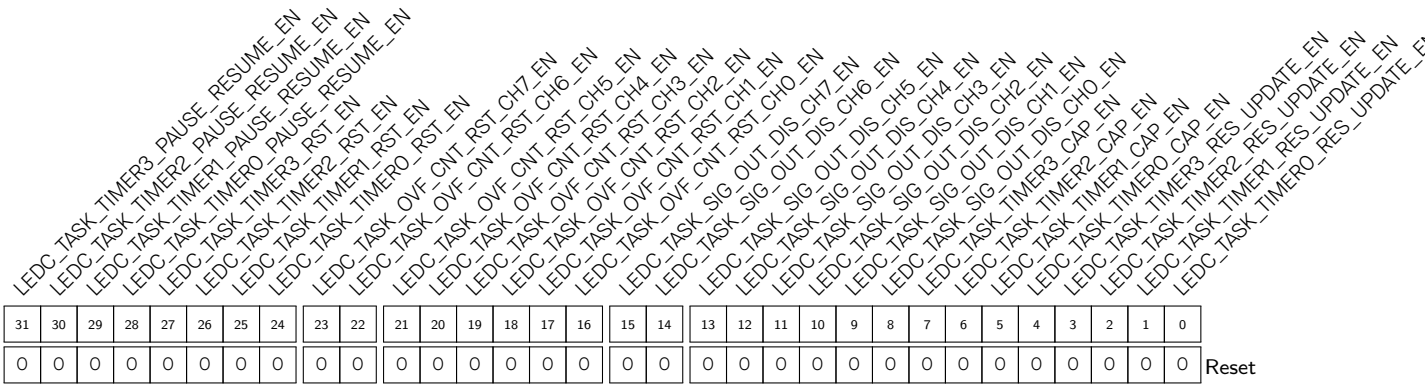
LEDC_TASK_DUTY_SCALE_UPDATE_CH*n*_EN (*n*: 0-7) Configures whether to enable the LEDC_TASK_DUTY_SCALE_UPDATE_CH*n* task.

0: Disable

1: Enable

(R/W)

Register 58.8. LEDC_EVT_TASK_EN1_REG (0x0124)



LEDC_TASK_TIMER*x*_RES_UPDATE_EN (*x*: 0-3) Configures whether to enable LEDC_TASK_TIMER*x*_RES_UPDATE task.

0: Disable

1: Enable

(R/W)

LEDC_TASK_TIMER*x*_CAP_EN (*x*: 0-3) Configures whether to enable LEDC_TASK_TIMER*x*_CAP task.

0: Disable

1: Enable

(R/W)

LEDC_TASK_SIG_OUT_DIS_CH*n*_EN (*n*: 0-7) Configures whether to enable LEDC_TASK_SIG_OUT_DIS_CH*n* task.

0: Disable

1: Enable

(R/W)

LEDC_TASK_OVF_CNT_RST_CH*n*_EN (*n*: 0-7) Configures whether to enable LEDC_TASK_OVF_CNT_RST_CH*n* task.

0: Disable

1: Enable

(R/W)

LEDC_TASK_TIMER*x*_RST_EN (*x*: 0-3) Configures whether to enable LEDC_TASK_TIMER*x*_RST task.

0: Disable

1: Enable

(R/W)

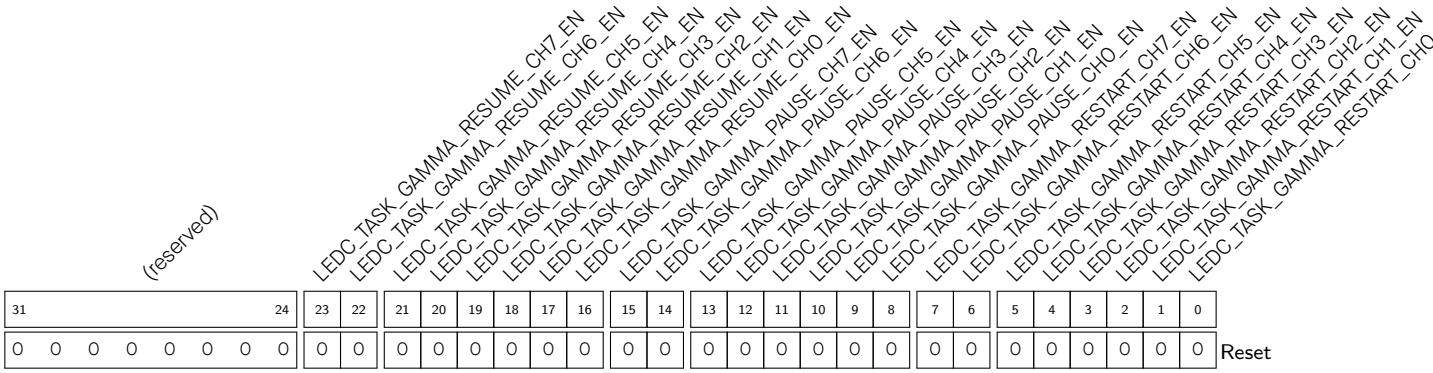
LEDC_TASK_TIMER*x*_PAUSE_RESUME_EN (*x*: 0-3) Configures whether to enable LEDC_TASK_TIMER*x*_PAUSE and LEDC_TASK_TIMER*x*_RESUME task.

0: Disable

1: Enable

(R/W)

Register 58.9. LEDC_EVT_TASK_EN2_REG (0x0128)



LEDC_TASK_GAMMA_RESTART_CH*n*_EN (*n*: 0-7) Configures whether to enable LEDC_TASK_GAMMA_RESTART_CH*n* task.

0: Disable

1: Enable

(R/W)

LEDC_TASK_GAMMA_PAUSE_CH*n*_EN (*n*: 0-7) Configures whether to enable LEDC_TASK_GAMMA_PAUSE_CH*n* task.

0: Disable

1: Enable

(R/W)

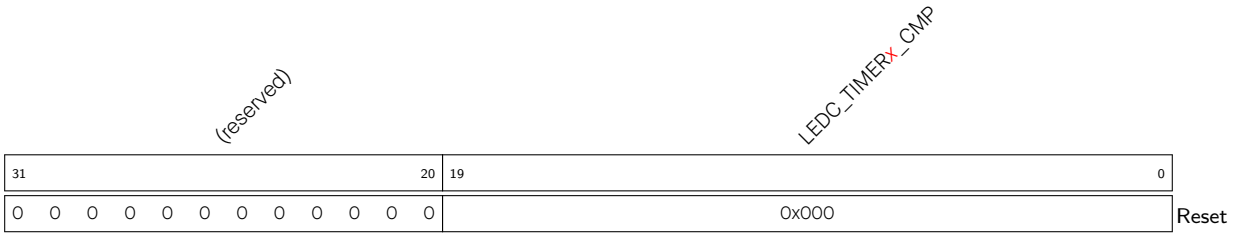
LEDC_TASK_GAMMA_RESUME_CH*n*_EN (*n*: 0-7) Configures whether to enable LEDC_TASK_GAMMA_RESUME_CH*n* task.

0: Disable

1: Enable

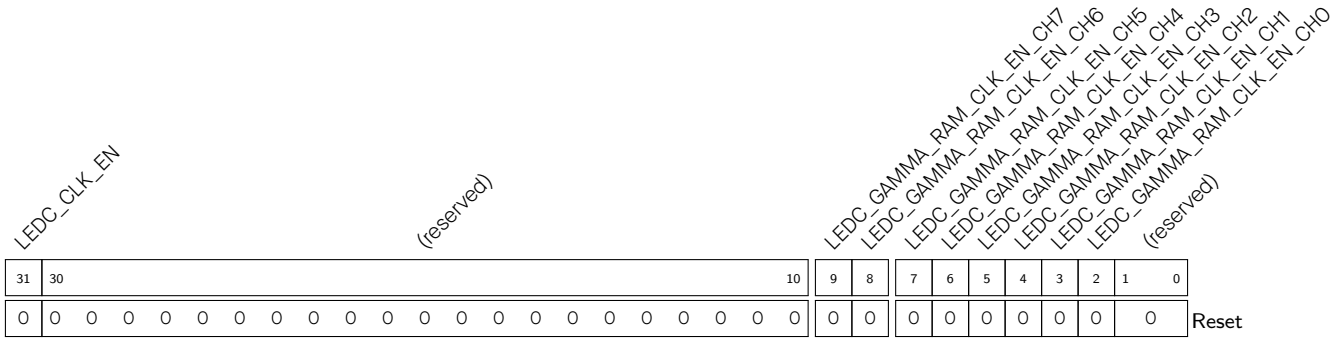
(R/W)

Register 58.10. LEDC_TIMER*x*_CMP_REG (*x*: 0-3) (0x0140+0x4**x*)



LEDC_TIMER*x*_CMP Configures the comparison value for LEDC timer *x*. (R/W)

Register 58.11. LEDC_CONF_REG (0x0170)



LEDC_GAMMA_RAM_CLK_EN_CH*n* (*n*: 0-7) Configures whether to open LEDC channel *n* gamma RAM clock gate.

0: Open the clock gate only when an application writes or reads LEDC channel *n* gamma RAM

1: Force open the clock gate for LEDC channel *n* gamma RAM

(R/W)

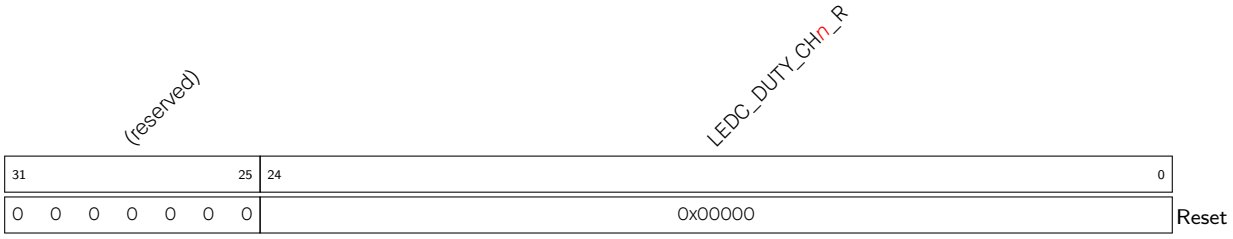
LEDC_CLK_EN Configures whether to open the register clock gate.

0: Open the clock gate only when an application writes registers

1: Force open the clock gate for register

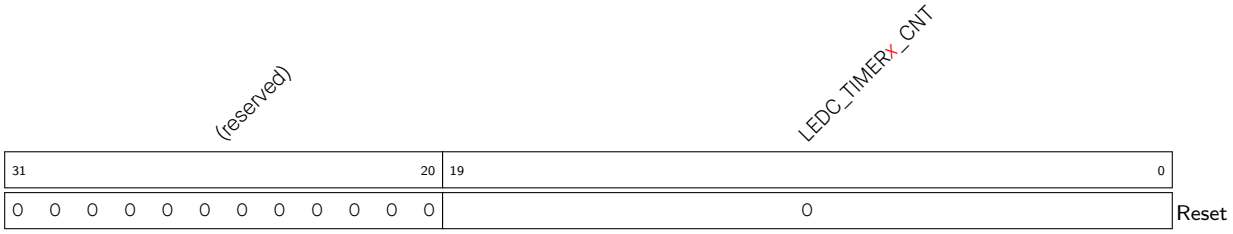
(R/W)

Register 58.12. LEDC_CH*n*_DUTY_R_REG (*n*: 0-7) (0x0010+0x14**n*)



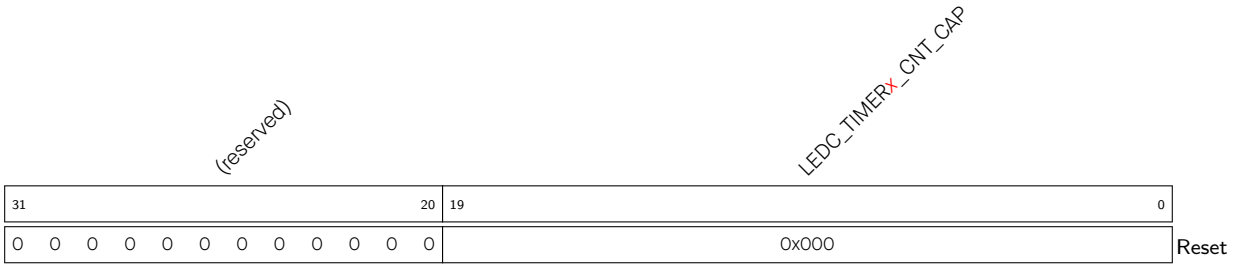
LEDC_DUTY_CH*n*_R Represents the current duty cycle of output signal on channel *n*. (RO)

Register 58.13. LEDC_TIMER*x*_VALUE_REG (*x*: 0-3) (0x00A4+0x8**x*)



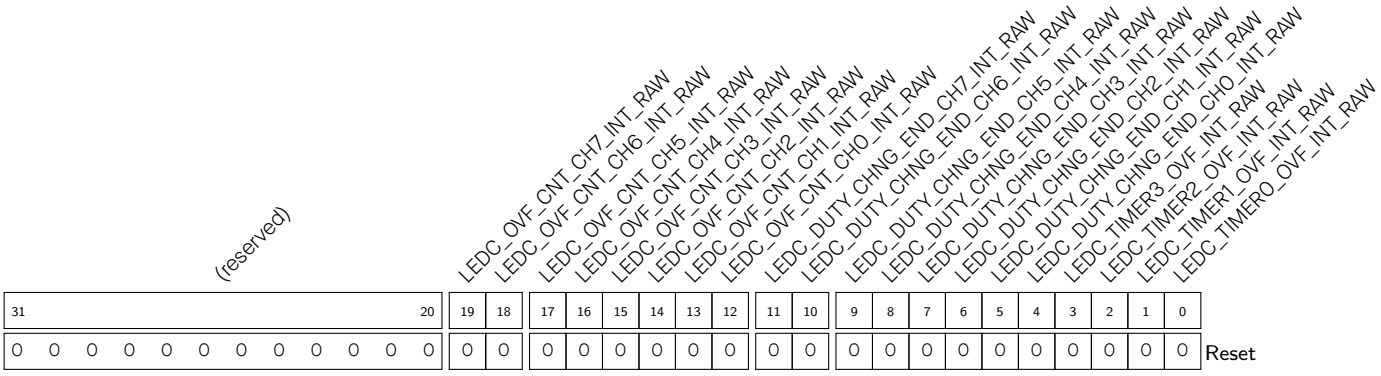
LEDC_TIMER*x*_CNT Represents the current counter value of timer *x*. (RO)

Register 58.14. LEDC_TIMERx_CNT_CAP_REG (x: 0-3) (0x0150+0x4*x)



LEDC_TIMERx_CNT_CAP Represents the captured LEDC timer *x* count value. (RO)

Register 58.15. LEDC_INT_RAW_REG (0x00C0)

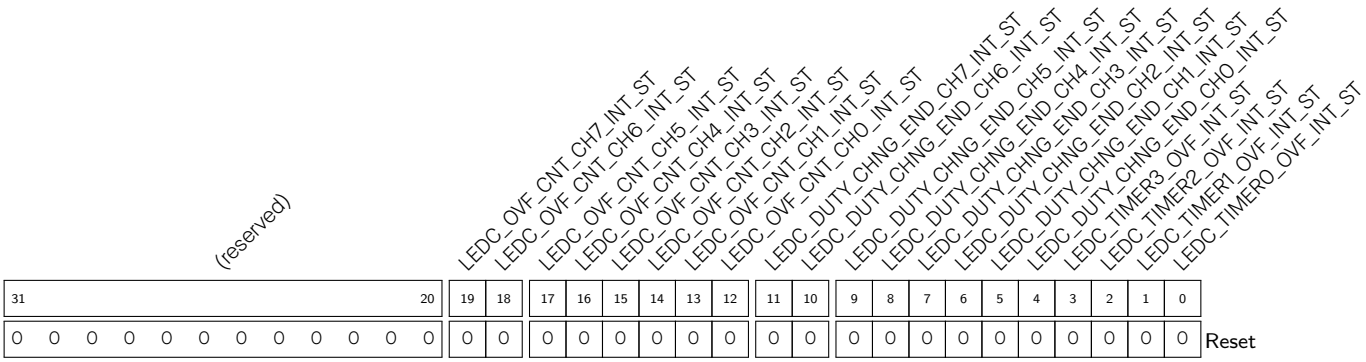


LEDC_TIMERx_OVF_INT_RAW (x: 0-3) Raw status bit: The raw interrupt status of LEDC_TIMERx_OVF_INT. Triggered when the timer *x* has reached its maximum counter value. (R/WTC/SS)

LEDC_DUTY_CHNG_END_CHn_INT_RAW (n: 0-7) Raw status bit: The raw interrupt status of LEDC_DUTY_CHNG_END_CHn_INT. Triggered when the fading of duty has finished. (R/WTC/SS)

LEDC_OVF_CNT_CHn_INT_RAW (n: 0-7) Raw status bit: The raw interrupt status of LEDC_OVF_CNT_CHn_INT. Triggered when the ovf_cnt has reached the value specified by LEDC_OVF_NUM_CHn. (R/WTC/SS)

Register 58.16. LEDC_INT_ST_REG (0x00C4)

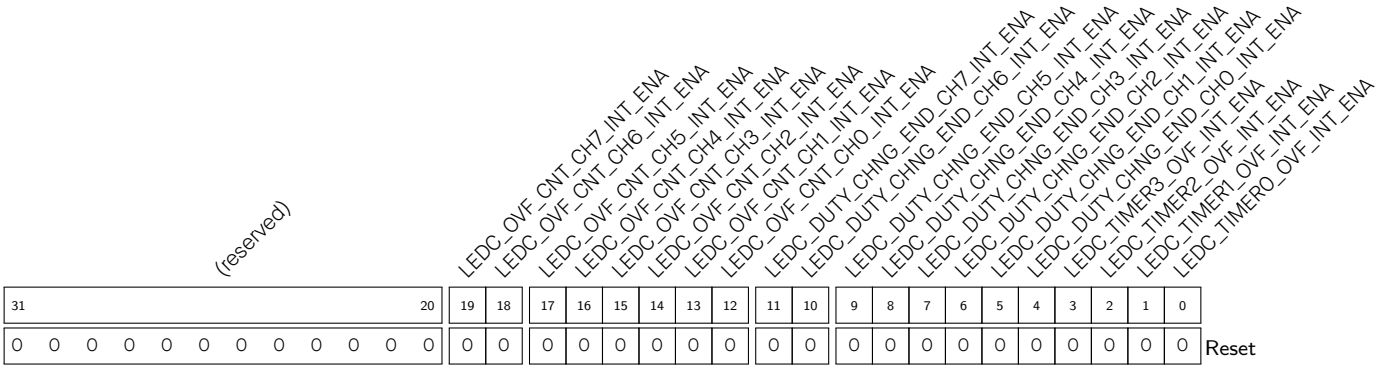


LEDC_TIMER*x*_OVF_INT_ST (*x*: 0-3) Masked status bit: The masked interrupt status of LEDC_TIMER*x*_OVF_INT. Valid only when LEDC_TIMER*x*_OVF_INT_ENA is set to 1. (RO)

LEDC_DUTY_CHNG_END_CH*n*_INT_ST (*n*: 0-7) Masked status bit: The masked interrupt status of LEDC_DUTY_CHNG_END_CH*n*_INT. Valid only when LEDC_DUTY_CHNG_END_CH*n*_INT_ENA is set to 1. (RO)

LEDC_OVF_CNT_CH*n*_INT_ST (*n*: 0-7) Masked status bit: The masked interrupt status of LEDC_OVF_CNT_CH*n*_INT. Valid only when LEDC_OVF_CNT_CH*n*_INT_ENA is set to 1. (RO)

Register 58.17. LEDC_INT_ENA_REG (0x00C8)

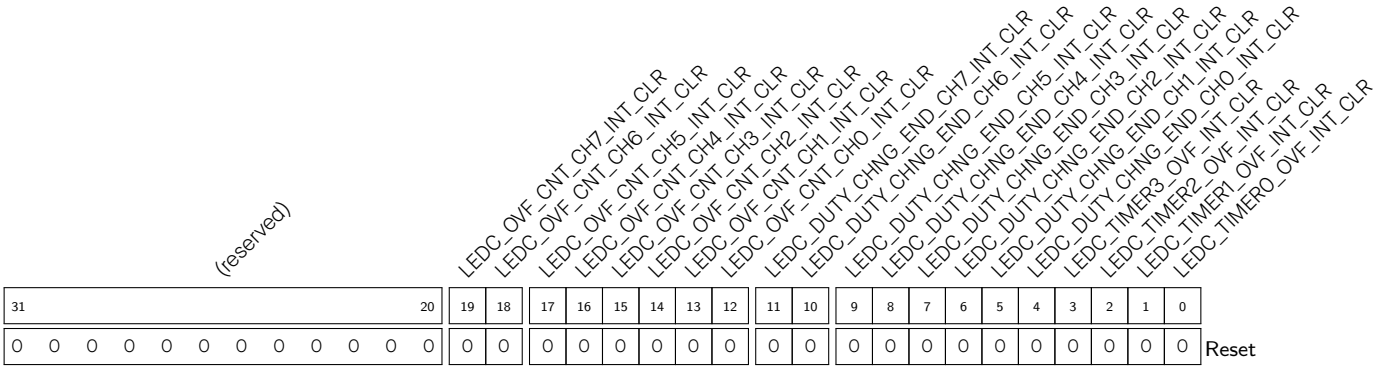


LEDC_TIMER*x*_OVF_INT_ENA (*x*: 0-3) Enable bit: Write 1 to enable LEDC_TIMER*x*_OVF_INT. (R/W)

LEDC_DUTY_CHNG_END_CH*n*_INT_ENA (*n*: 0-7) Enable bit: Write 1 to enable LEDC_DUTY_CHNG_END_CH*n*_INT. (R/W)

LEDC_OVF_CNT_CH*n*_INT_ENA (*n*: 0-7) Enable bit: Write 1 to enable LEDC_OVF_CNT_CH*n*_INT. (R/W)

Register 58.18. LEDC_INT_CLR_REG (0x00CC)

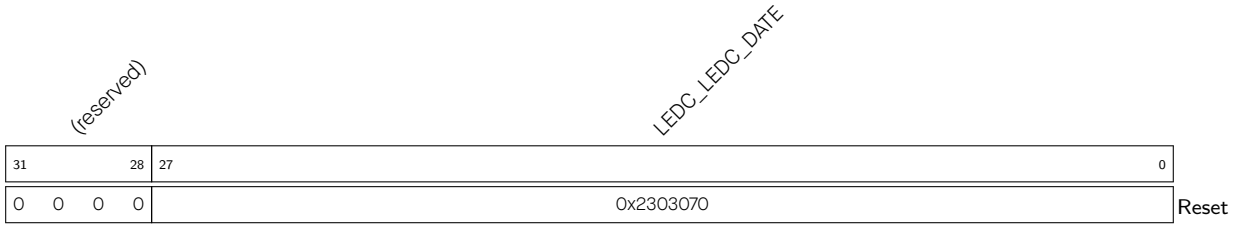


LEDC_TIMER*x*_OVF_INT_CLR (*x*: 0-3) Clear bit: Write 1 to clear LEDC_TIMER*x*_OVF_INT. (WT)

LEDC_DUTY_CHNG_END_CH*n*_INT_CLR (*n*: 0-7) Clear bit: Write 1 to clear LEDC_DUTY_CHNG_END_CH*n*_INT. (WT)

LEDC_OVF_CNT_CH*n*_INT_CLR (*n*: 0-7) Clear bit: Write 1 to clear LEDC_OVF_CNT_CH*n*_INT. (WT)

Register 58.19. LEDC_DATE_REG (0x0174)



LEDC_LEDC_DATE Configures the version. (R/W)

Chapter 59

Motor Control PWM (MCPWM)

59.1 Overview

The **M**otor **C**ontrol **P**ulse **W**idth **M**odulator (MCPWM) peripheral is intended for motor and power control. It provides six PWM outputs that can be set up to operate in several topologies. One common topology uses a pair of PWM outputs driving an H-bridge to control motor rotation speed and rotation direction.

The MCPWM can be divided into five main modules: PWM timers, PWM operators, Capture module, Event Task Matrix (ETM) module, and Fault Detection module. Each PWM timer provides timing references that can either run freely or be synced to other timers or external sources. Each PWM operator has all the necessary control resources to generate waveform pairs for one PWM channel. The Capture module is used for systems that need to accurately time external events. The ETM module responds to tasks received by the MCPWM, generating corresponding events depending on the state of motion. The Fault Detection module is used to capture external faults, allowing the system to respond by choice.

ESP32-P4 has two MCPWM peripherals, which are MCPWM0 and MCPWM1.

59.2 Features

An MCPWM peripheral has one clock divider (prescaler), three PWM timers, three PWM operators, a Capture module, an ETM module, and a Fault Detection module. MCPWM's core clock can be selected from three clock sources: PLL_F160M_CLK, XTAL_CLK, and RC_FAST_CLK (configured by the HP_SYS_CLKRST_MCPWM*n*_CLK_SRC_SEL field of the HP_SYS_CLKRST_PERI_CLK_CTRL20_REG register). Figure 59.2-1 shows the submodules inside MCPWM and the signals on the interface. PWM timers are used for generating timing references. The PWM operators generate the desired waveform based on the timing references. Any PWM operator can be configured to use the timing references of any PWM timers. Different PWM operators can use the same PWM timer's timing reference to generate PWM signals, or different PWM timers' values to generate separate PWM signals. Different PWM timers can also be synchronized together.

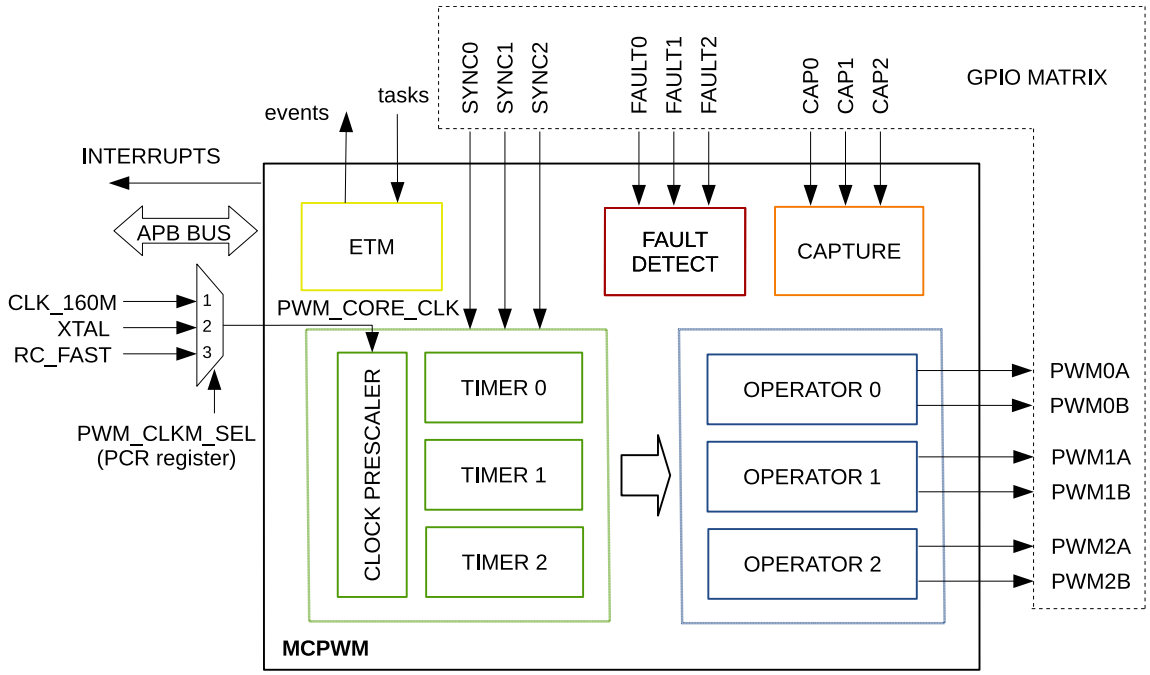


Figure 59.2-1. MCPWM Module Overview

Below is an overview of the submodules' functionality in Figure 59.2-1:

- PWM Timers 0, 1, and 2:
  - Every PWM timer has a dedicated 8-bit clock prescaler.
  - The 16-bit counter in the PWM timer can work in Count-Up Mode, Count-Down Mode, or Count-Up-Down Mode.
  - A hardware sync or software sync can trigger a reload on the PWM timer with a phase register. It will also trigger the prescaler's restart, so that the timer's clock can also be synced. The source of the hard sync can come from any GPIO or any other PWM timer's sync_out. The source of the soft sync comes from writing toggle value to the MCPWM_TIMER*n*_SYNC_SW bit.
- PWM Operators 0, 1, and 2:
  - Every PWM operator has two PWM outputs: PWM*x*A and PWM*x*B. They can work independently, in symmetric or asymmetric configurations.
  - The control of the PWM signal can be updated asynchronously.
  - Configurable dead time on rising and falling edges; each set up independently.
  - All events can trigger CPU interrupts.
  - Modulating of PWM output by high-frequency carrier signals, useful when gate drivers are insulated with a transformer.
  - Period, time stamps, and important control registers have shadow registers with flexible updating methods.
- Fault Detection Module:
  - Programmable fault handling in both cycle-by-cycle mode and one-shot mode.

  - A fault condition can force the PWM output to either high or low logic levels.

- Capture Module:
  - Clock of the capture module is the same as MCPWM's core clock.
  - Speed measurement of rotating machinery.
  - Measurement of elapsed time between position sensor pulses
  - Period and duty cycle measurement of pulse train signals
  - Decoding current or voltage amplitude derived from duty-cycle-encoded signals of current/voltage sensors
  - Three individual capture channels, each of which with a time-stamp register (32-bit)
  - Selection of edge polarity and prescaling of input capture signals
  - The capture timer can sync with a PWM timer or external signals.
  - Interrupt on each of the three capture channels
- ETM Module:
  - Generation of different events depending on the different running states of each timer and operator.
  - Each timer and operator responds to its corresponding task and automatically performs the corresponding operation.
  - Each event and task can be enabled independently. When an event is not enabled, the corresponding event will not be generated. When a task is not enabled, the corresponding task will not be responded to.

59.3 Modules

59.3.1 Overview

The key modules in each MCPWM are timer module, operator module, fault detection module, capture module, and ETM module. See their functions in the following sections.

59.3.1.1 Timer Module

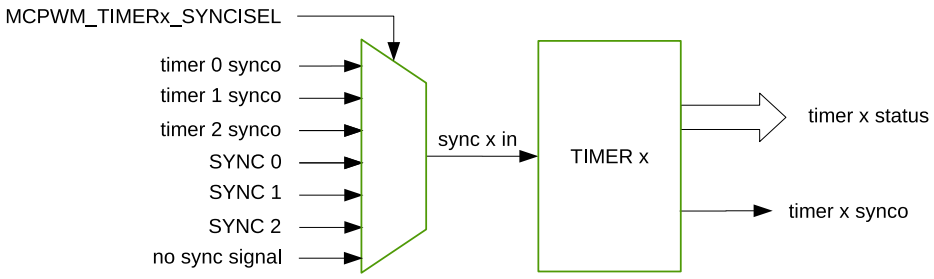


Figure 59.3-1. Timer Module

This module generates a timer, which can count at a specified period. It can work in Count-Up Mode, Count-Down Mode, or Count-Up-Down Mode. It supports different synchronization input sources (a total of seven optional input sources) for reloading the count value and direction, allowing synchronization among multiple timers. Furthermore, it provides synchronization output options (a total of four optional output sources) for use by other timers.

The timer*x* status in the figure represents the timer status output by the timer module, such as the counting mode, and count value equal to zero or period.

59.3.1.2 Operator Module

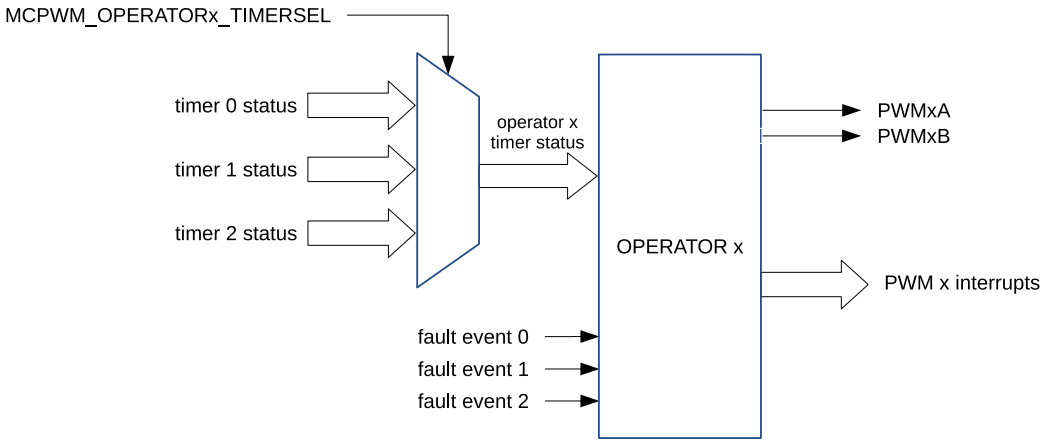


Figure 59.3-2. Operator Module

A PWM operator contains a PWM generator, a dead time generator, and a PWM carrier module. Their functions are shown in Table 59.3-1.

Table 59.3-1. Functions of Each Submodule in the Operator Module

Submodule	Functions
PWM Generator	Generate PWM signals according to the configured duty cycle and waveform.
Dead Time Generator	Choose to add dead time to the rising and falling edges of the PWM signal generated by the PWM generator (add delays to the rising and falling edges), and perform operations such as inversion.
PWM Carrier	Choose to add a carrier to the PWM signal generated by the dead time generator. The carrier frequency and the first pulse duration of the PWM waveform after adding the carrier can be configured.
Fault Detector	Detect faults and generate corresponding fault events, respond to fault events based on the specified interval mode (one-shot mode or cycle-by-cycle mode), and when a fault event occurs, take action on the PWM signal generated by the PWM carrier in the specified way.

59.3.1.3 Capture Module

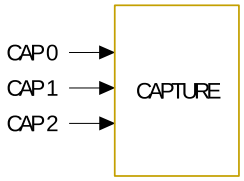


Figure 59.3-3. Capture Module

The module contains a timer that can be synchronized. When the input capture signal is valid, the count value of the timer is captured.

59.3.1.4 ETM Module

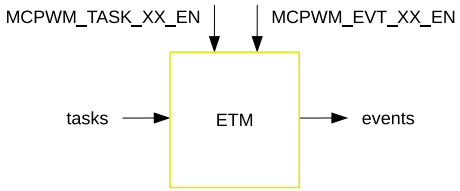


Figure 59.3-4. ETM Module

The module responds to received tasks, or generate and output events. Each event and task can be enabled independently.

59.3.2 PWM Timer Module

MCPWM has three PWM timer modules. Any of them can determine the necessary event timing for any of the three PWM operator modules. By using the synchronization signals from the GPIO matrix, built-in synchronization logic allows multiple PWM timer modules in one or more MCPWM peripherals to work together as a system.

59.3.2.1 Configurations of the PWM Timer Module

Users can configure the following functions of the PWM timer module:

- Control how often events occur by specifying the PWM timer frequency or period.
- Configure a particular PWM timer to synchronize phases with other PWM timers or modules.
- Configure the following timer counting modes: count-up, count-down, count-up-down.
- Change the rate of the PWM timer clock (PT_CLK) with a prescaler. Each timer has its own prescaler configured with MCPWM_TIMER*n*_PRESCALE of the register MCPWM_TIMER0_CFG0_REG. The PWM timer increments or decrements at a slower pace, depending on the setting of this field. The new MCPWM_TIMER*n*_PRESCALE configuration value will take effect when the timer stops and starts counting again.

59.3.2.2 PWM Timer's Working Modes and Timing Event Generation

The PWM timer has three working modes, selected by the PWM*x* timer mode field:

- Count-Up Mode:
  The PWM timer increments from zero until reaching the value configured in the period field. Once done, the PWM timer returns to zero and starts increasing again. PWM period = the value of the period field + 1.
  Note: The period field is MCPWM_TIMER*n*_PERIOD (*x* = 0, 1, 2), i.e., MCPWM_TIMER0_PERIOD, MCPWM_TIMER1_PERIOD, MCPWM_TIMER2_PERIOD.
- Count-Down Mode:
  The PWM timer decrements to zero, starting from the value configured in the period field. Once done, the PWM timer returns to the period value and starts decrementing again. In this case, the PWM period = the value of period field + 1.
- Count-Up-Down Mode:
  This is a combination of the two modes mentioned above. The PWM timer starts increasing from zero until the period value is reached. Then, the timer decreases back to zero. The PWM timer cycles incrementally and decrementally in this mode. The PWM period = the value of the period field × 2.

Figures 59.3-5 to 59.3-8 show PWM timer waveforms in different modes, including timer behavior during synchronization events. In Count-Up mode, the counting direction after synchronization is always counting up. In Count-Down mode, the counting direction after synchronization is always counting down. In Count-Up-Down Mode, the counting direction after synchronization can be chosen by setting the MCPWM_TIMER*n*_PHASE_DIRECTION.

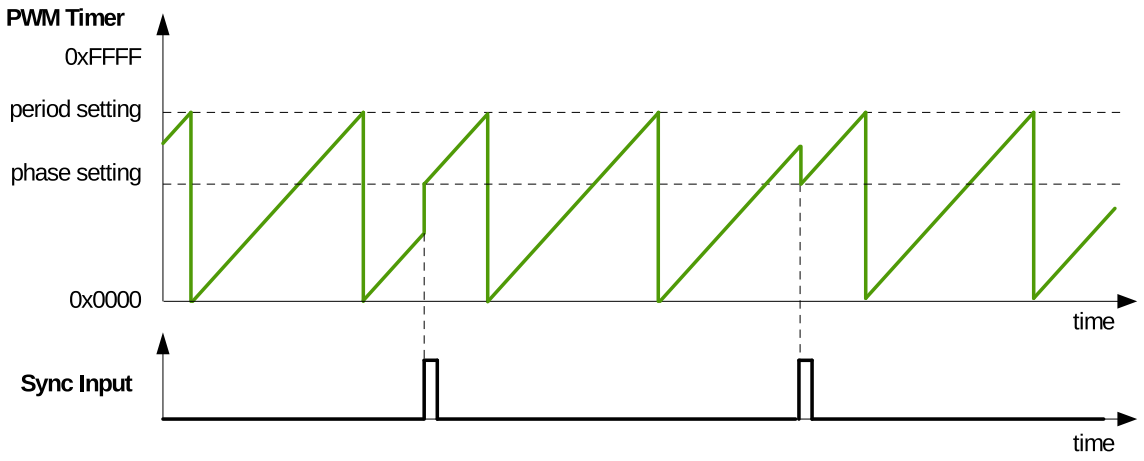


Figure 59.3-5. Count-Up Mode Waveform

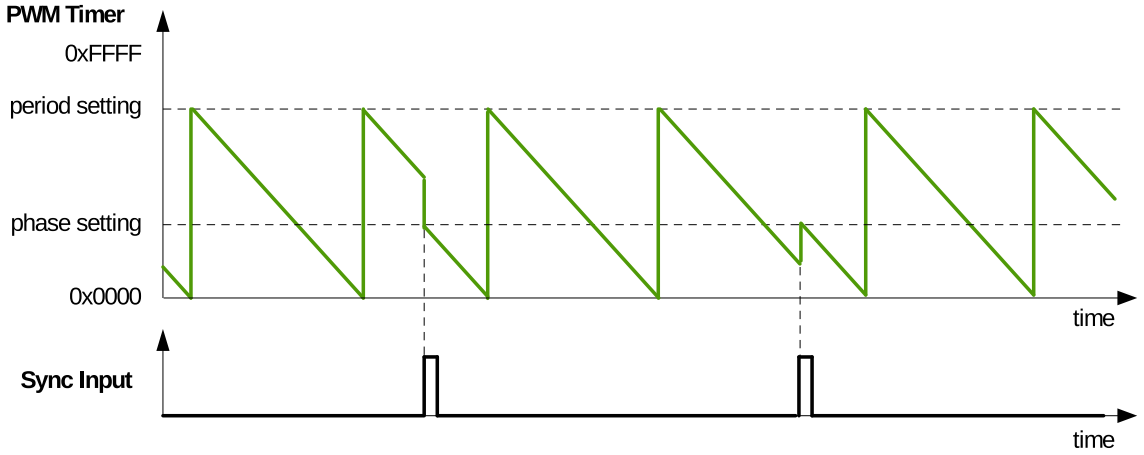


Figure 59.3-6. Count-Down Mode Waveforms

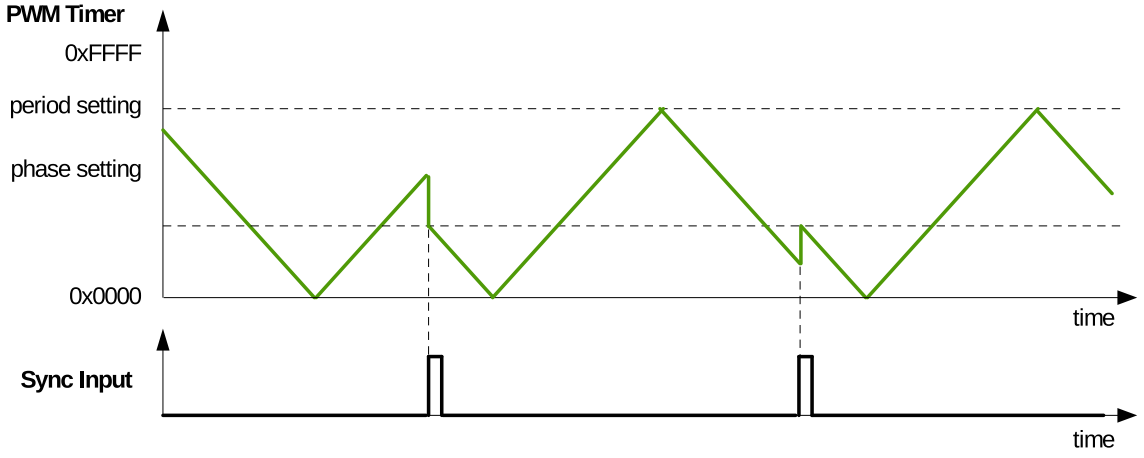


Figure 59.3-7. Count-Up-Down Mode Waveforms, Count-Down at Synchronization Event

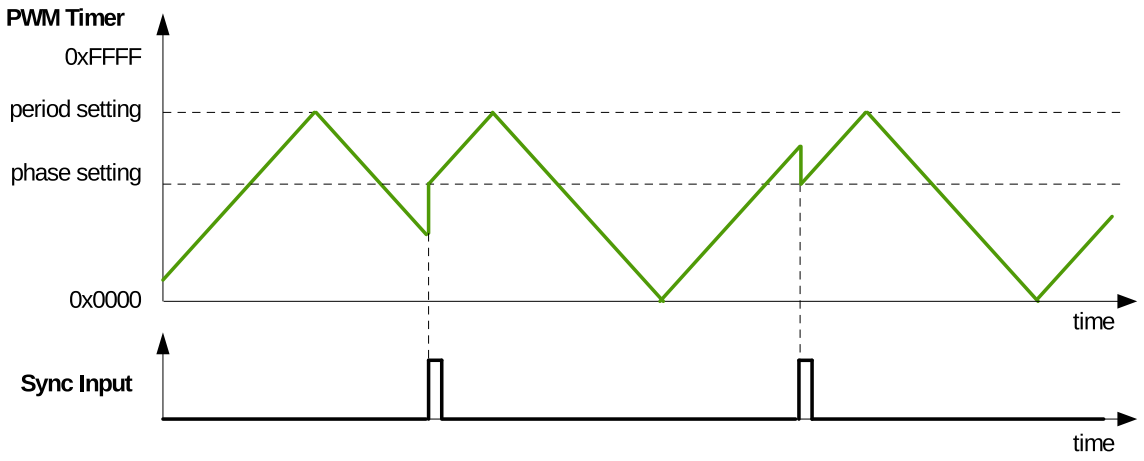


Figure 59.3-8. Count-Up-Down Mode Waveforms, Count-Up at Synchronization Event

When the PWM timer is running, it generates the following timing events periodically and automatically:

- UTEP: The timing event generated when the PWM timer's value is equal to the value of the period field (MCPWM_TIMER*n*_PERIOD) and when the PWM timer is increasing.
- UTEZ: The timing event generated when the PWM timer's value equals zero and when the PWM timer is increasing.
- DTEP: The timing event generated when the PWM timer's value equals the value of the period field (MCPWM_TIMER*n*_PERIOD) and when the PWM timer is decreasing.
- DTEZ: The timing event generated when the PWM timer's value equals zero and when the PWM timer is decreasing.

Figures 59.3-9 to 59.3-11 show the timing waveforms of U/DTEP and U/DTEZ.

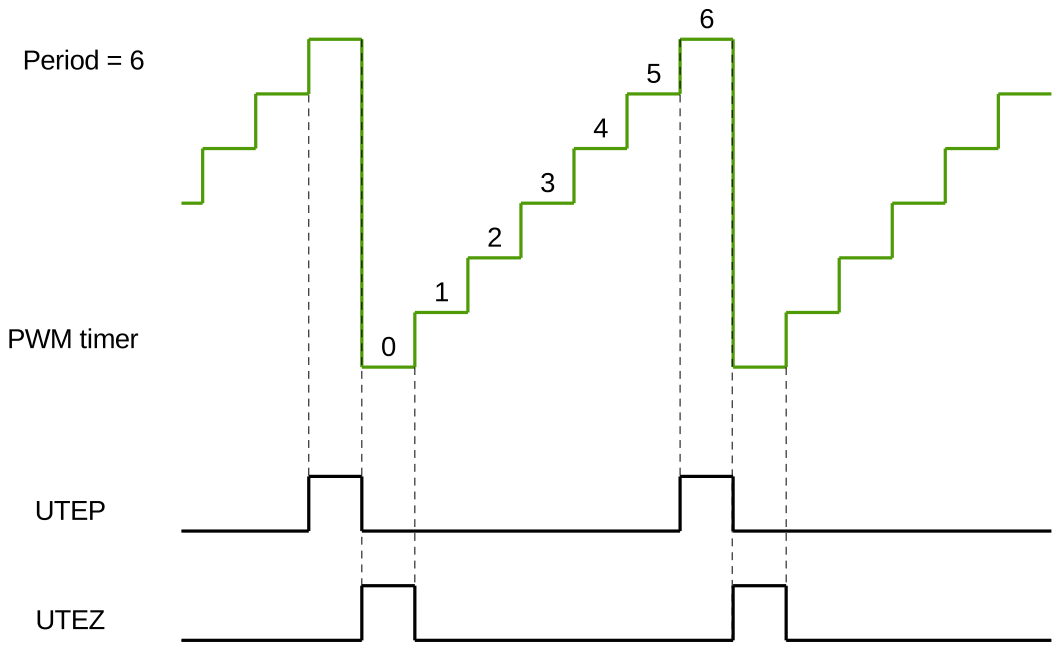


Figure 59.3-9. UTEP and UTEZ Generation in Count-Up Mode

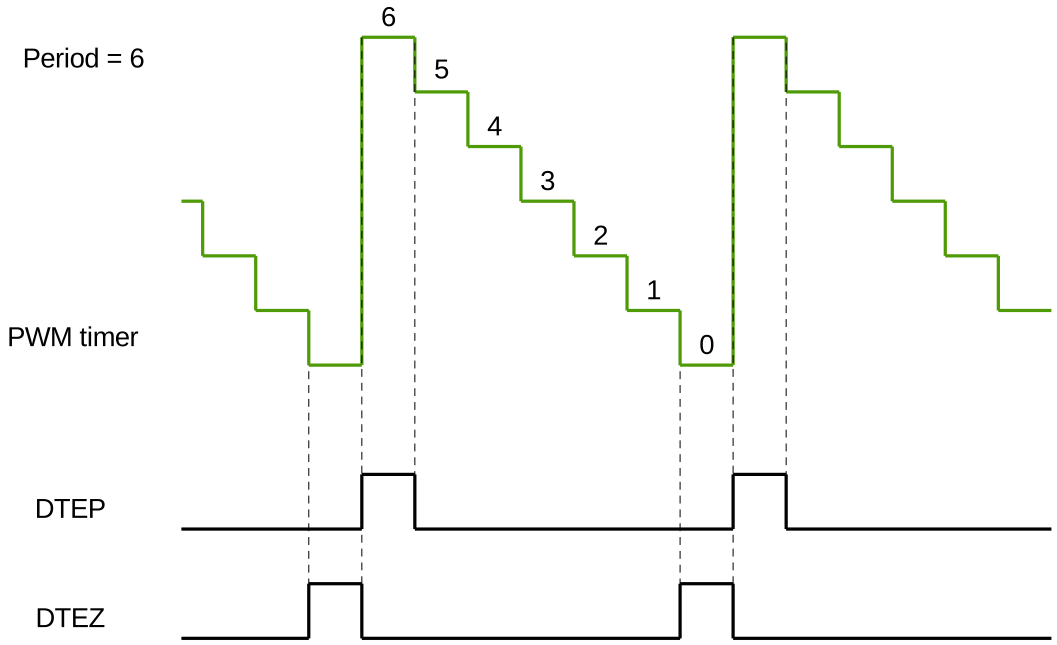


Figure 59.3-10. DTEP and DTEZ Generation in Count-Down Mode

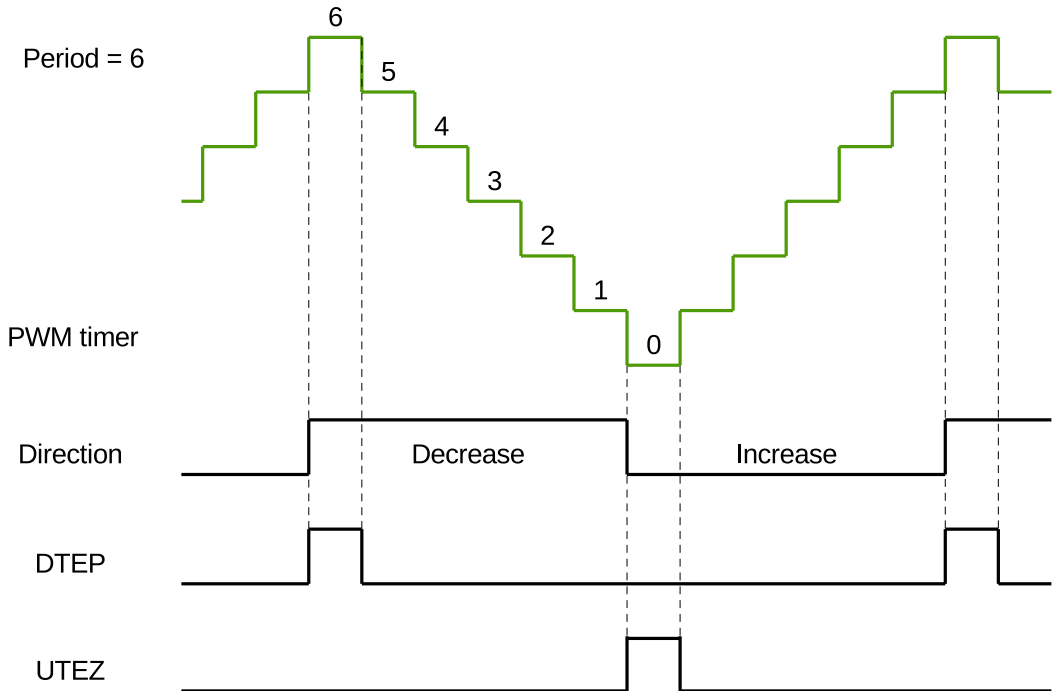


Figure 59.3-11. DTEP and UTEZ Generation in Count-Up-Down Mode

Please note that in the Count-Up-Down Mode, when the counting direction is increasing, the timer range is [0, period value - 1], and when the counting direction is decreasing, the timer range is [period value, 1]. That is, in this mode, when synchronizing the timer to 0, decreasing counting direction will be illegal, namely, MCPWM_TIMER*n*_PHASE_DIRECTION cannot be set to 1. Similarly, when synchronizing the timer to period value, increasing counting direction will be illegal, namely, MCPWM_TIMER*n*_PHASE_DIRECTION cannot be set to 0. Therefore, when the timer is synchronized to 0, the counting direction can only be increasing, and MCPWM_TIMER*n*_PHASE_DIRECTION will be 0. When the timer is synchronized to the period value, the counting direction can only be decreasing, and MCPWM_TIMER*n*_PHASE_DIRECTION will be 1.

59.3.2.3 Shadow Register of PWM Timer

The PWM timer's period register and the PWM timer's clock prescaler register have shadow registers. The shadow registers can back up the values that are about to be written to the valid registers. It also supports writing the values saved into the active register at a specific moment of hardware synchronization. The functionality of both register types is as follows:

- Active Register: Directly responsible for controlling all actions performed by hardware.
- Shadow Register: Acts as a temporary buffer for a value to be written to the active register. At a specific, user-configured point in time, the value saved in the shadow register is copied to the active register. Before this happens, the content of the shadow register has no direct effect on the controlled hardware. This helps to prevent erroneous operation of the hardware, which may happen when a register is asynchronously modified by software. Both the shadow register and the active register have the same memory address. The software always writes into, or reads from the shadow register.

  The moment of updating the clock prescaler's active register is at the time when the timer starts operating. When MCPWM_GLOBAL_UP_EN is set to 1, the moment of updating the period active register can be selected by the following ways:

  - By configuring the update method register MCPWM_TIMER*n*_PERIOD_UPMETHOD to 0, the update will start immediately.
  - By configuring the update method register MCPWM_TIMER*n*_PERIOD_UPMETHOD to 1, the update can start when the PWM timer is equal to zero.
  - By configuring the update method register MCPWM_TIMER*n*_PERIOD_UPMETHOD to 2, the update can start when the PWM timer is synchronized.
  - By configuring the update method register MCPWM_TIMER*n*_PERIOD_UPMETHOD to 3, the update can start when the PWM timer is equal to zero or is synchronized.
  - Software can also trigger a globally forced update bit MCPWM_GLOBAL_FORCE_UP which will prompt all registers in the module to be updated according to shadow registers.

59.3.2.4 PWM Timer Synchronization and Phase Locking

The PWM modules adopt a flexible synchronization method. Each PWM timer has a synchronization input and a synchronization output. The synchronization input can be selected from three synchronization outputs of the three PWM timers and three synchronization signals from the GPIO matrix. The synchronization output can be generated when the PWM timer's value equals to period or zero, the synchronization input signal is active, or software synchronizes. Thus, each PWM timer can flexibly select the synchronization input for different phase synchronization methods. During synchronization, the PWM timer clock prescaler will reset its counter and restart dividing the clock in order to ensure the correctness of the timer.

59.3.3 PWM Operator Module

The PWM Operator module has the following functions:

- Generates a PWM signal pair, based on timing references obtained from the corresponding PWM timer.

- Each signal out of the PWM signal pair includes a specific pattern of dead time (i.e., adding delays to the rising and falling edges of the PWM signal).
- Superimposes a carrier on the PWM signal, if configured to do so.
- Handles response under fault conditions.

Figure 59.3-12 shows the block diagram of a PWM operator.

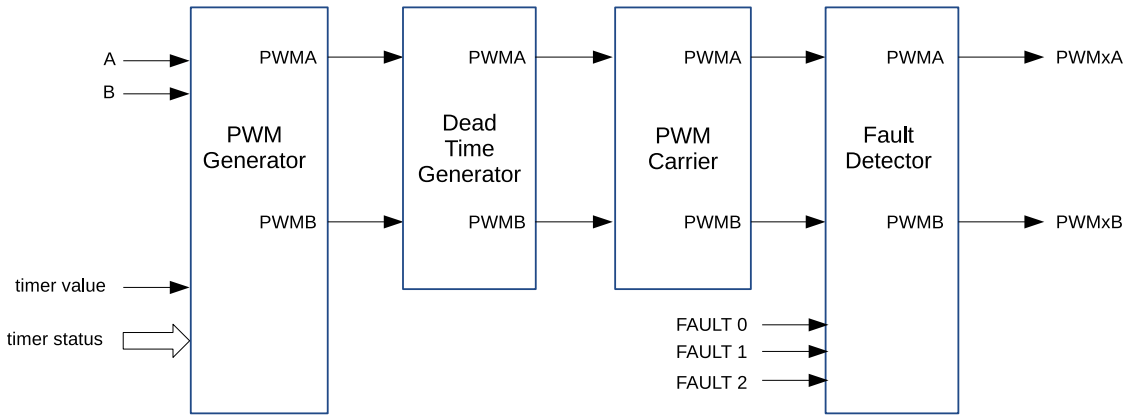


Figure 59.3-12. Block Diagram of A PWM Operator

59.3.3.1 PWM Generator Module

Purpose of the PWM Generator Module

In this module, important timing events are generated or imported. The events are then converted into specific actions to generate the desired waveforms at the PWM*x*A and PWM*x*B outputs.

The PWM generator module performs the following actions:

- Generation of timing events based on time stamps configured using the A and B registers. Events happen when the following conditions are met (for the configuration of registers A and B, see section 59.3.3.1):
  - UTEA: the PWM timer is counting up and its value is equal to register A.
  - UTEB: the PWM timer is counting up and its value is equal to register B.
  - DTEA: the PWM timer is counting down and its value is equal to register A.
  - DTEB: the PWM timer is counting down and its value is equal to register B.
- Generation of U/DT0, U/DT1 timing events based on fault or synchronization events.
  - UT0: the PWM timer is counting up and FAULT0 detected (field MCPWM_GEN*n*_T0_SEL is set to 0) or FAULT1 detected (field MCPWM_GEN*n*_T0_SEL is set to 1) or FAULT2 detected (field MCPWM_GEN*n*_T0_SEL is set to 2) or synchronized (field MCPWM_GEN*n*_T0_SEL is set to 3).
  - UT1: the PWM timer is counting up and FAULT0 detected (field MCPWM_GEN*n*_T1_SEL is set to 0) or FAULT1 detected (field MCPWM_GEN*n*_T1_SEL is set to 1) or FAULT2 detected (field MCPWM_GEN*n*_T1_SEL is set to 2) or synchronized (field MCPWM_GEN*n*_T1_SEL is set to 3).
  - DT0: the PWM timer is counting down and FAULT0 detected (field MCPWM_GEN*n*_T0_SEL is set to 0) or FAULT1 detected (field MCPWM_GEN*n*_T0_SEL is set to 1) or FAULT2 detected (field MCPWM_GEN*n*_T0_SEL is set to 2) or synchronized (field MCPWM_GEN*n*_T0_SEL is set to 3).
  - DT1: the PWM timer is counting down and FAULT0 detected(field MCPWM_GEN*n*_T1_SEL is set to 0) or FAULT1 detected (field MCPWM_GEN*n*_T1_SEL is set to 1) or FAULT2 detected (field MCPWM_GEN*n*_T1_SEL is set to 2) or synchronized (field MCPWM_GEN*n*_T1_SEL is set to 3).
- Management of priority when these timing events occur concurrently.
- Generation of set, clear, and toggle actions, based on the timing events.
- Controlling of the PWM duty cycle, depending on the configuration of the PWM generator module.
- Handling of new time stamp values, using shadow registers to prevent glitches in the PWM cycle.

Shadow Register of PWM Generator

The time stamp registers A and B used by the hardware have shadow registers, which are registers MCPWM_GEN*n*_A_REG and MCPWM_GEN*n*_B_REG. Shadowing provides a way of updating registers in sync with the hardware.

When MCPWM_GLOBAL_UP_EN is set to 1, the shadow registers can be written to the active register at a specified time. The update method field for MCPWM_GEN*n*_A_REG and MCPWM_GEN*n*_B_REG is MCPWM_GEN*n*_CFG_UPMETHOD. Software can also trigger a globally forced update bit

MCPWM_GLOBAL_FORCE_UP which will prompt all registers in the module to be updated according to shadow registers. For a description of the shadow registers, please see Section 59.3.2.3.

Timing Events

For convenience, all timing signals and events are summarized in Table 59.3-2.

Table 59.3-2. Timing Events Used in PWM Generator

Signal	Event Description	PWM Timer Operation
DTEP	PWM timer value is equal to the period register value	PWM timer counts down
DTEZ	PWM timer value is equal to zero	
DTEA	PWM timer value is equal to register A	
DTEB	PWM timer value is equal to register B	
DT0 event	Based on fault or synchronization events	
DT1 event	Based on fault or synchronization events	
UTEP	PWM timer value is equal to the period register value	PWM timer counts up
UTEZ	PWM timer value is equal to zero	
UTEA	PWM timer value is equal to register A	
UTEB	PWM timer value is equal to register B	
UT0 event	Based on fault or synchronization events	
UT1 event	Based on fault or synchronization events	
Software-force event	Software-initiated asynchronous event	N/A

The purpose of a software-force event is to impose non-continuous or continuous changes on the PWM*x*A and PWM*x*B outputs. The change is done asynchronously. Software-force control is handled by the MCPWM_GEN*n*_FORCE_REG registers.

The selection and configuration of T0/T1 in the PWM generator module is independent of the configuration of fault events in the fault handler module. A particular trip event may or may not be configured to cause trip action in the fault handler submodule, but the same event can be used by the PWM generator to trigger T0/T1 for controlling PWM waveforms.

It is important to know that when the PWM timer is in Count-Up-Down Mode. It will always decrement after a TEP event, and increment after a TEZ event. So, when the PWM timer is in Count-Up-Down Mode, DTEP and UTEZ events will occur, while UTEP and DTEZ events never occurs.

The PWM generator can handle multiple events at the same time. Events are prioritized by the hardware and relevant details are provided in Table 59.3-3 and Table 59.3-4. Priority levels range from 1 (the highest) to 7 (the lowest). Please note that the priority of TEP and TEZ events depends on the PWM timer's counting mode.

If the value of A or B is set to be greater than the period, then U/DTEA and U/DTEB will never occur.

Table 59.3-3. Timing Events Priority When PWM Timer Increments

Priority Level	Event
1 (highest)	Software-forced event
2	UTEP

Priority Level	Event
3	UT0
4	UT1
5	UTEB
6	UTEA
7 (lowest)	UTEZ

Table 59.3-4. Timing Events Priority when PWM Timer Decrements

Priority level	Event
1 (highest)	Software-forced event
2	DTEZ
3	DT0
4	DT1
5	DTEB
6	DTEA
7 (lowest)	DTEP

Notes:

1. UTEP and UTEZ do not happen simultaneously. When the PWM timer is in Count-Up Mode, UTEP will always happen one cycle earlier than UTEZ, as demonstrated in Figure 59.3-9, so their action on PWM signals will not interrupt each other. When the PWM timer is in Count-Up-Down Mode, UTEP will not occur.
2. DTEP and DTEZ do not happen simultaneously. When the PWM timer is in Count-Down Mode, DTEZ will always happen one cycle earlier than DTEP, as demonstrated in Figure 59.3-10, so their action on PWM signals will not interrupt each other. When the PWM timer is in Count-Up-Down Mode, DTEZ will not occur.

PWM Signal Generation

The PWM generator module controls the behavior of outputting PWM*x*A and PWM*x*B when a particular timing event occurs. The timing events are further qualified by the PWM timer's counting mode (increment or decrement). Knowing the counting mode, the module may then perform an independent action at each stage of the PWM timer counting up or down.

The following actions may be configured on PWM*x*A and PWM*x*B outputs:

- Set High: Set the output of PWM*x*A or PWM*x*B to a high level.
- Clear Low: Clear the output of PWM*x*A or PWM*x*B by setting it to a low level.
- Toggle: Change the current output level of PWM*x*A or PWM*x*B to the opposite value. If it is currently pulled up, then pull it down, or vice versa.
- Do Nothing: Keep both outputs PWM*x*A and PWM*x*B unchanged. In this state, interrupts can still be triggered.

Actions on outputs is configured by using registers MCPWM_GEN*n*_A_REG and

MCPWM_GEN*n*_B_REG. So, the action to be taken on each output is set independently. Also, there is great flexibility in selecting actions to be taken on a given output based on events. More specifically, any event listed in Table 59.3-2 can operate on either output of PWM*x*A or PWM*x*B. To check out registers for particular generator 0, 1, or 2, please refer to the register description in Section 59.5.

Waveforms for Common Configurations

In the configuration of various waveforms below, period refers to the configuration value stored in the period register of the PWM timer selected by the PWM operator (i.e., MCPWM_TIMER*n* _PERIOD (*x* = 0, 1, 2)). For configuration methods, please refer to the section 59.3.2. A and B denote the time stamp registers A and B used by the hardware. For register configuration, please refer to the section 59.3.3.1. UTEA/B, UTEZ/P, DTEA/B, DTEZ/P, DTO/1, and UTO/1 represent different timing events of the PWM generator. For event definition, please refer to the section 59.3.3.1. The upward arrow, downward arrow, and T mark signify setting the PWM waveform high, setting it low, toggling it when the timing event occurs. See section 59.3.3.1 for configuration. PWM*x*A and PWM*x*B are the output signals of PWM generator*x*.

Figure 59.3-13 presents the symmetric PWM waveform generated when the PWM timer is in Count-Up-Down mode. DC 0%–100% modulation can be calculated via the formula below:

$$Duty = (Period - A) \div Period$$

If A matches the PWM timer value and the PWM timer is incrementing, then the PWM output is pulled up. If A matches the PWM timer value while the PWM timer is decrementing, then the PWM output is pulled low.

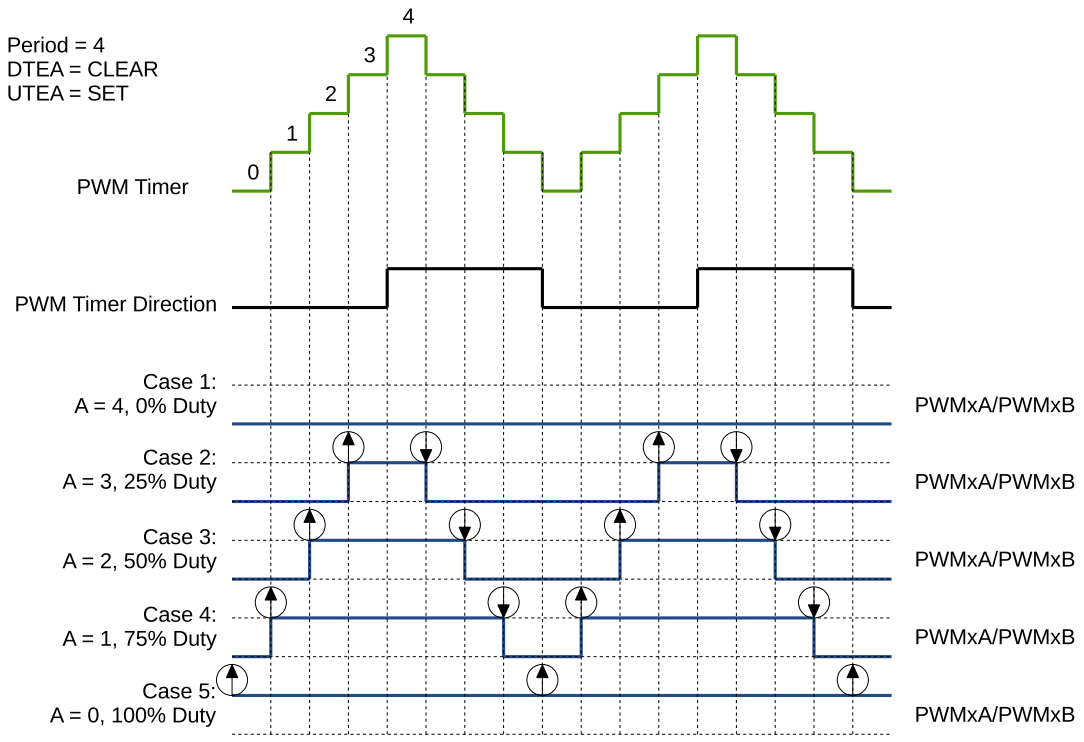


Figure 59.3-13. Symmetrical Waveform in Count-Up-Down Mode

The PWM waveforms in Figures 59.3-14 to 59.3-17 show some common PWM generator configurations.

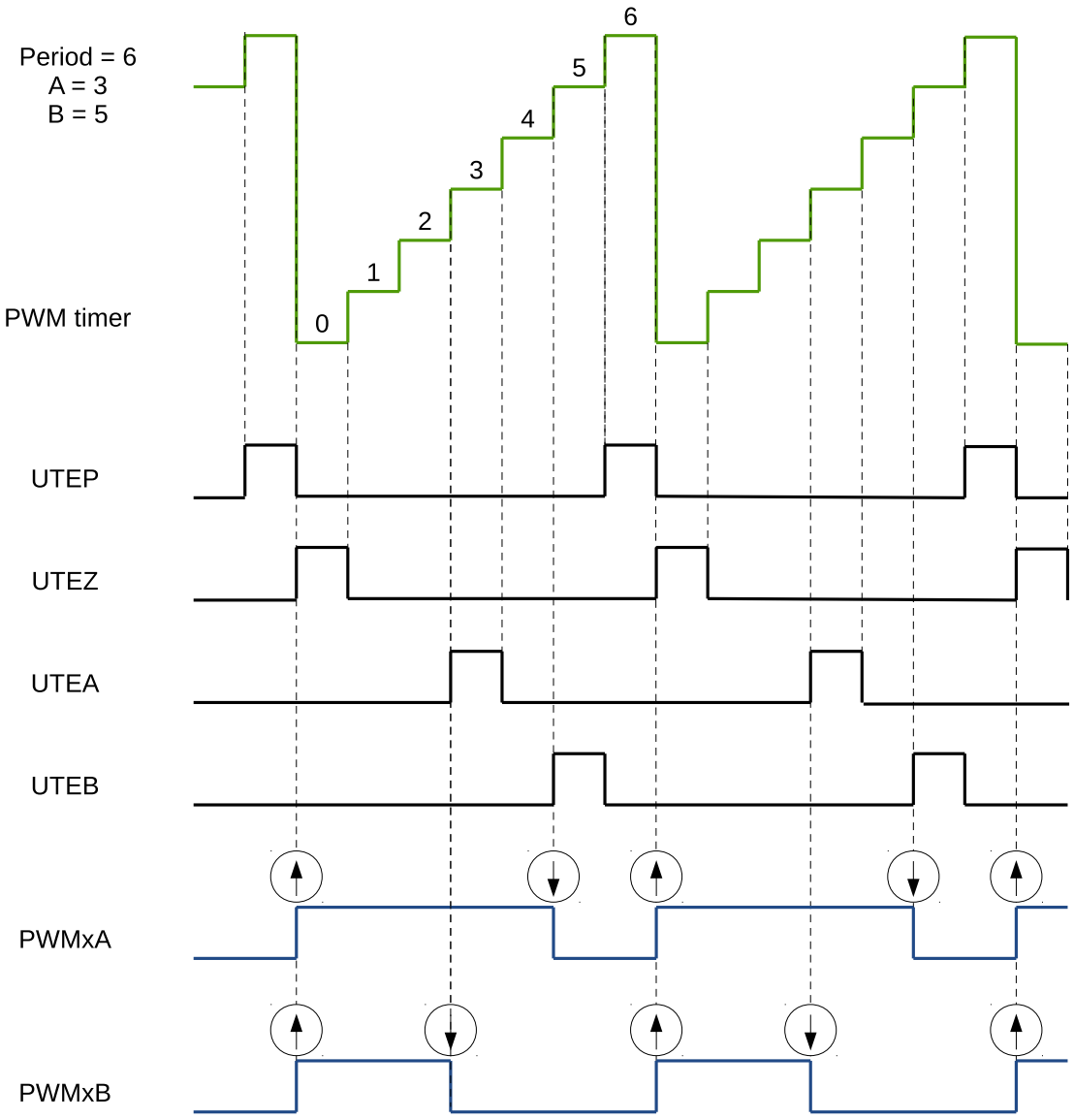


Figure 59.3-14. Count-Up, Single Edge Asymmetric Waveform, with Independent Modulation on PWM*x*A and PWM*x*B — Active High

The duty modulation for PWM*x*A is set by B, active high and proportional to B.
The duty modulation for PWM*x*B is set by A, active high and proportional to A.

$$Period = (MCPWM_TIMERn_PERIOD + 1) \times T_{PT_clk}$$

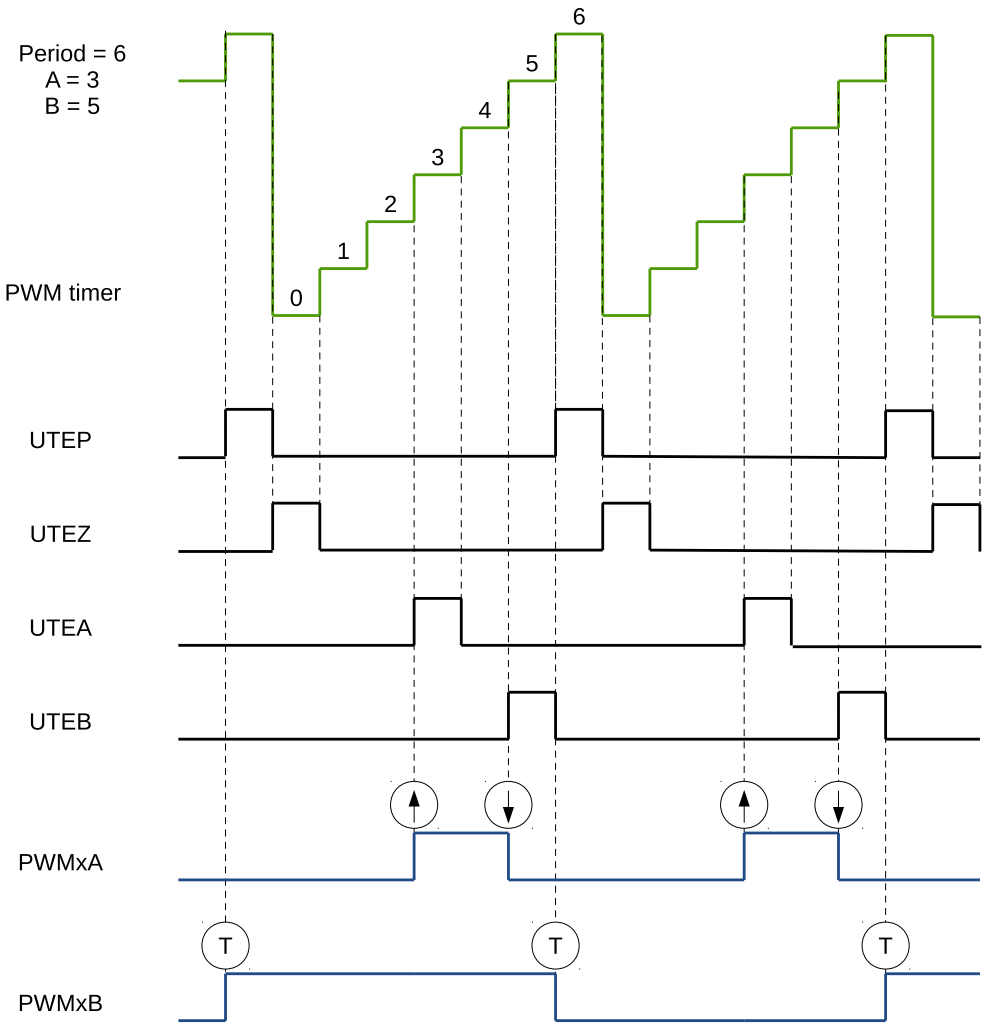


Figure 59.3-15. Count-Up, Pulse Placement Asymmetric Waveform with Independent Modulation on PWM*x*A

Pulses may be generated anywhere within the PWM cycle (zero to period).

PWM*x*A's high time duty is proportional to (B – A).

$$Period = (MCPWM_TIMERn_PERIOD + 1) \times T_{PT_clk}$$

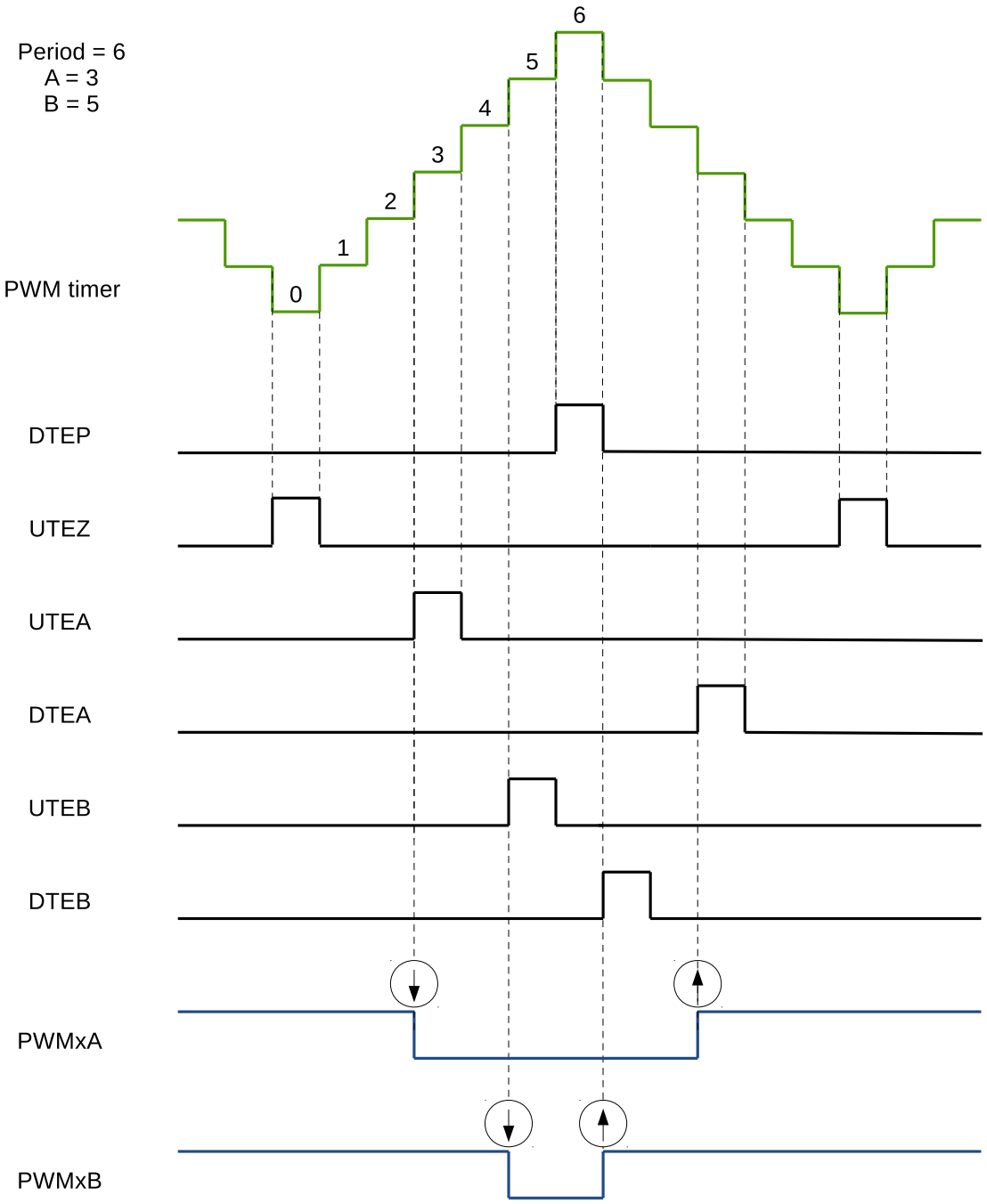


Figure 59.3-16. Count-Up-Down, Dual Edge Symmetric Waveform, with Independent Modulation on PWM*x*A and PWM*x*B — Active High

The duty modulation for PWM*x*A is set by A, active high and proportional to A.
The duty modulation for PWM*x*B is set by B, active high and proportional to B.
Outputting PWM*x*A and PWM*x*B can drive separate switches.

$$Period = (2 \times MCPWM_TIMERn_PERIOD) \times T_{PT_clk}$$

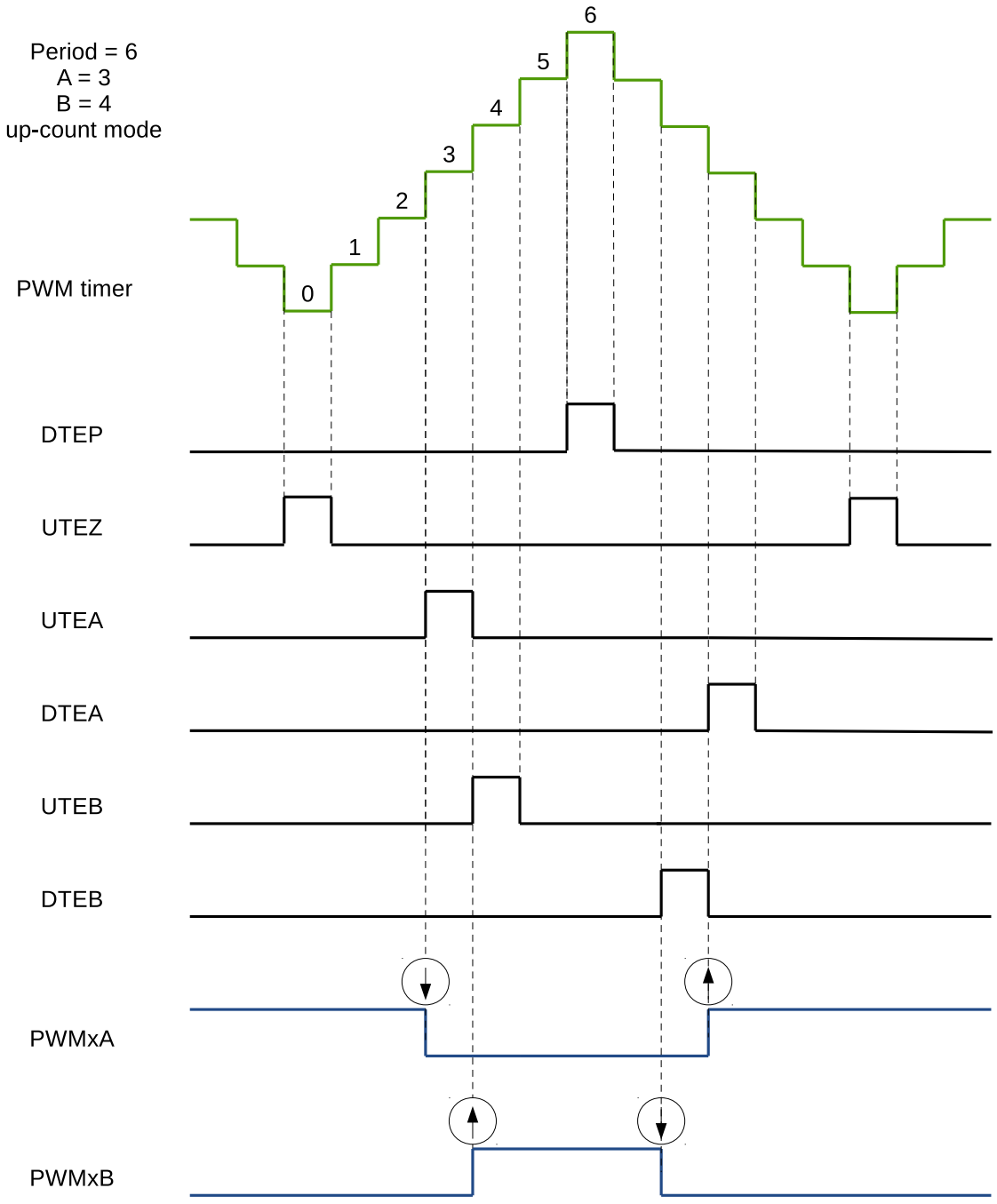


Figure 59.3-17. Count-Up-Down, Dual Edge Symmetric Waveform, with Independent Modulation on PWM*x*A and PWM*x*B — Complementary

The duty modulation of PWM*x*A is set by A, is active high and proportional to A.
The duty modulation of PWM*x*B is set by B, is active low and proportional to B.
Outputs PWM*x* can drive upper/lower (complementary) switches.
Dead time = B − A. Edge placement is configurable by software. Dead time generator module supports configuring edge delay methods when required.

$$Period = (2 \times MCPWM_TIMERn_PERIOD) \times T_{PT_clk}$$

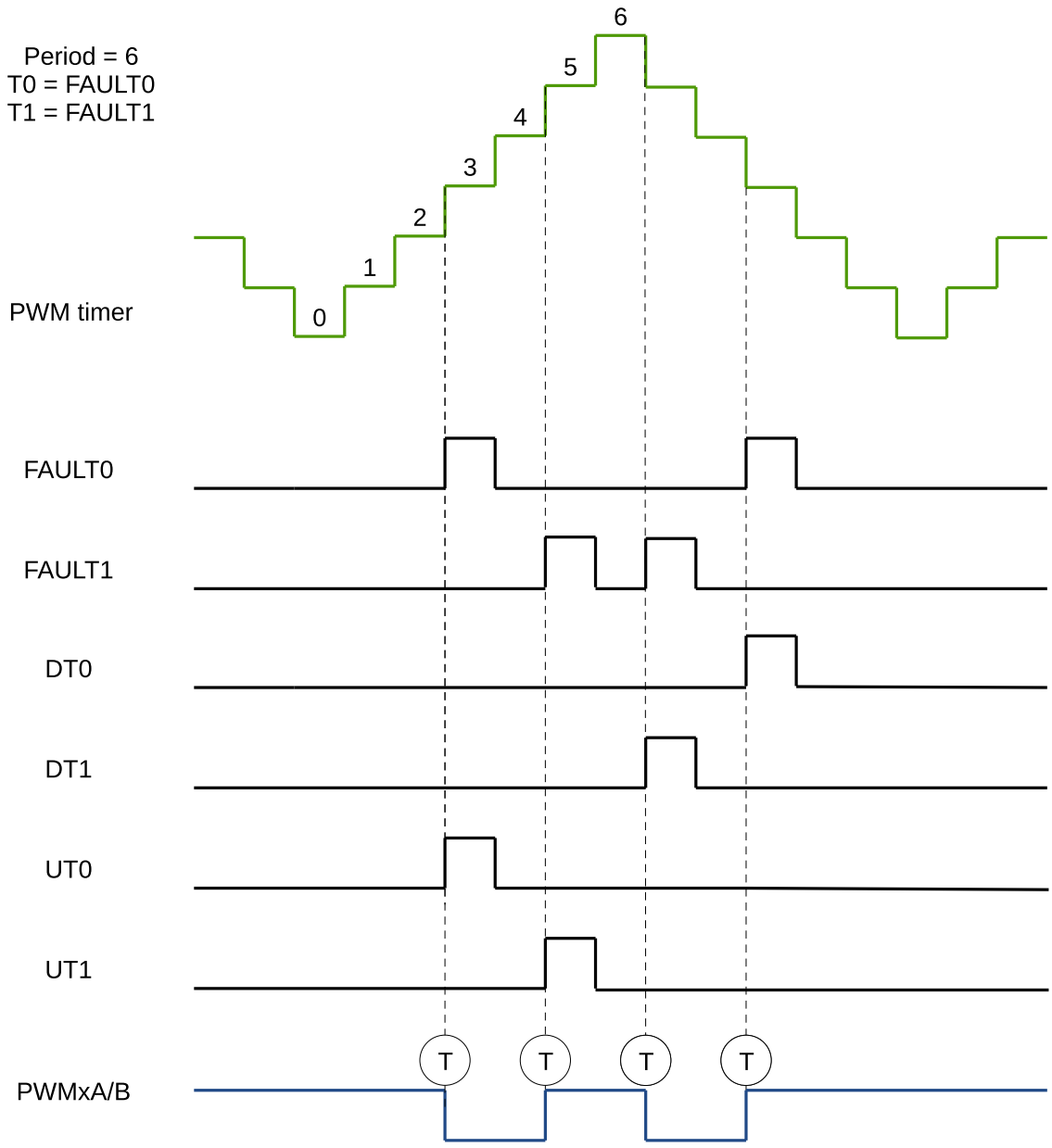


Figure 59.3-18. Count-Up-Down, Fault or Synchronization Events, with Same Modulation on PWM*x*A and PWM*x*B

Figure 59.3-18 shows a waveform when UT0/1 and DT0/1 events are generated. In this example, T0 selects FAULT0 and T1 selects FAULT1. The events selected by T0 and T1 can be configured independently, these events can be FAULT0, FAULT1, FAULT2 or synchronous. For detailed configuration, see section 59.3.3.1.

Software-Force Events

There are two types of software-force events inside the PWM generator:

- Non-continuous-immediate (NCI) software-force events: Such types of events are immediately effective on PWM outputs when triggered by software. The forcing is non-continuous, which means the next active timing events will be able to alter the PWM outputs.
- Continuous (CNTU) software-force events: Such types of events are continuous. The forced PWM

outputs will continue until they are released by software. The events' triggers are configurable. They can be configured to be timing events or immediate events.

For NCI software-force events, set register MCPWM_GEN*n*_A_NCIFORCE or MCPWM_GEN*n*_B_NCIFORCE to 1 and then to 0 to trigger the NCI software-forced events of PWM*x*A or PWM*x*B. Configure register MCPWM_GEN*n*_A_NCIFORCE_MODE or MCPWM_GEN*n*_B_NCIFORCE_MODE to specify the operation on the PWM*x*A or PWM*x*B waveform when the NCI software-force event occurs. Configuring the registers to 0 or 3 means no operation, 1 means low level, and 2 means high level.

For CNTU software-force events, set register MCPWM_GEN*n*_CNTUFORCE_UPMETHOD to specify the trigger method of CNTU software-force events. Setting all bits to 0 means trigger CNTU software-force events immediately, setting bit 0 to 1 means the TEZ event triggers CNTU software-force events, setting bit 1 to 1 means the TEP event triggers CNTU software-force events, setting bit 2 to 1 means the TEA event triggers CNTU software-force events, setting bit 3 to 1 means the TEB event triggers CNTU software-force events, setting bit 4 to 1 means CNTU software-force events triggered when the PWM timer selected by the PWM generator is synchronized, and setting bit 5 to 1 means the CNTU software-forced events will not be triggered. Configure register MCPWM_GEN*n*_A_CNTUFORCE_MODE or MCPWM_GEN*n*_B_CNTUFORCE_MODE to specify the operation on the PWM*x*A or PWM*x*B waveform when the CNTU software-force events occur. Configuring the registers to 0 or 3 means no operation, 1 means low level, and 2 means high level.

Figure 59.3-19 shows a waveform of NCI software-force events. NCI events are used to force PWM*x*A output low. Forcing on PWM*x*B is disabled in this case.

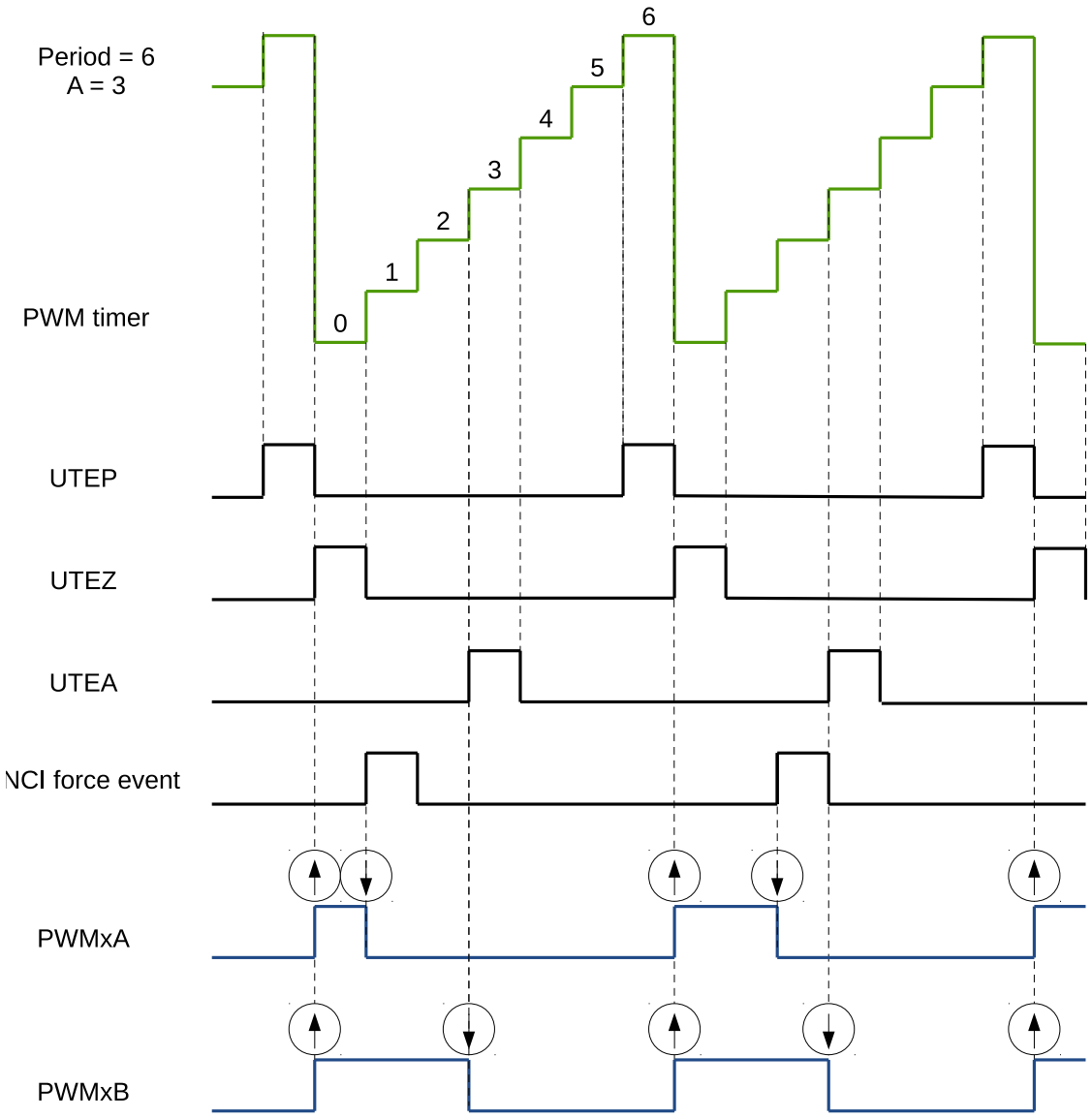


Figure 59.3-19. Example of an NCI Software-Force Event on PWM*x*A

Figure 59.3-20 shows a waveform of CNTU software-force events. TEZ events are selected as triggers for CNTU software-force events. CNTU is used to force the PWM*x*B output low. Forcing on PWM*x*A is disabled.

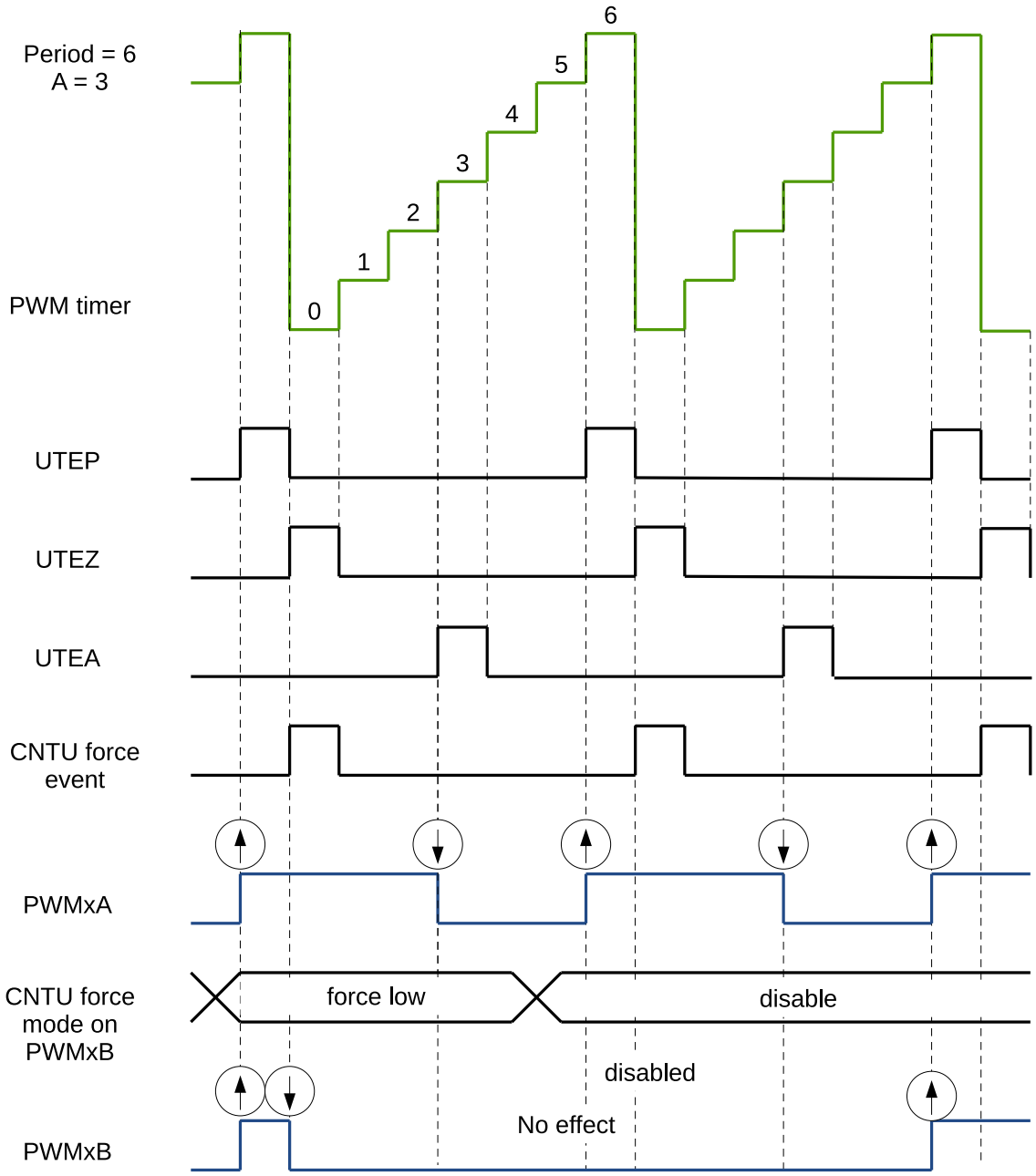


Figure 59.3-20. Example of a CNTU Software-Force Event on PWM*x*B

59.3.3.2 Dead Time Generator Module

Purpose of the Dead Time Generator Module

Section 59.3.3.1 introduced several options to generate signals on PWM*x*A and PWM*x*B outputs, with a specific placement of signal edges. The required dead time is obtained by altering the edge placement between signals and by setting the signal's duty cycle. Another option to control the dead time is to use a specialized module – Dead Time Generator.

The key functions of the Dead Time Generator module are as follows:

- Generating output signal pairs (PWM*x*A and PWM*x*B) with a dead time from a common source, which can be either the input signal PWM*x*A or PWM*x*B.
- Creating a dead time by adding delay to signal edges:
  - Rising edge delay (RED)
  - Falling edge delay (FED)
- Can generate PWM signal in various format. The typical dead time configurations are:
  - Active high complementary (AHC)
  - Active low complementary (ALC)
  - Active high (AH)
  - Active low (AL)
- This module may also be bypassed, if the dead time is configured directly in the generator module.

Shadow Register of Dead Time Generator

Delay registers RED and FED are shadowed with registers MCPWM_DT*n*_RED_CFG_REG and MCPWM_DT*n*_FED_CFG_REG. When MCPWM_GLOBAL_UP_EN is set to 1, the values saved in the shadow registers can be written to the active register at the specified time. The update method register for MCPWM_DT*n*_RED_CFG_REG is MCPWM_DB*n*_RED_UPMETHOD. The update method register for MCPWM_DT*n*_FED_CFG_REG is MCPWM_DB*n*_FED_UPMETHOD. The Software can also trigger a globally forced update bit MCPWM_GLOBAL_FORCE_UP which will prompt all registers in the module to be updated according to shadow registers. For the description of shadow registers, please see section 59.3.2.3.

Highlights for Operation of the Dead Time Generator

Options for setting up the dead time module are shown in Figure 59.3-21.

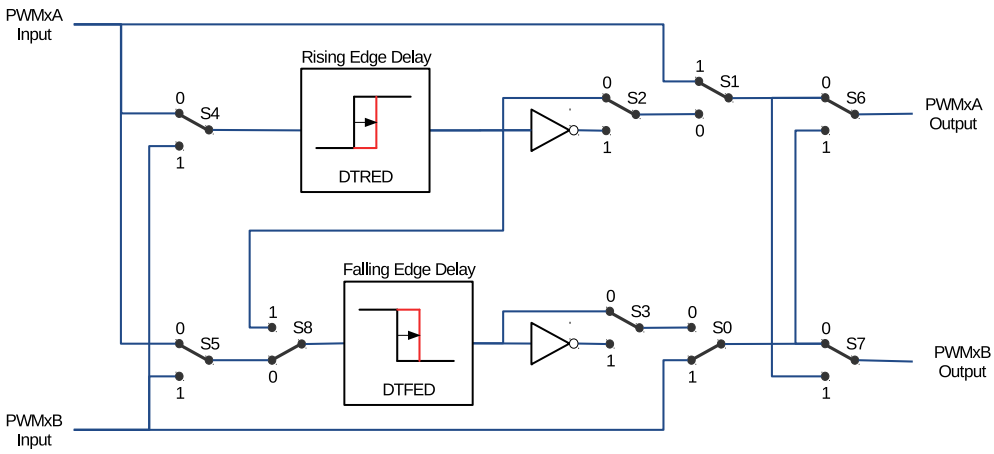


Figure 59.3-21. Options for Setting up the Dead Time Generator Module

S0-S8 in the figure above are switches controlled by fields in register MCPWM_DT*n*_CFG_REG shown in Table 59.3-5.

Table 59.3-5. Dead Time Generator Switches Control Fields

Switch	Field
S0	MCPWM_DB*n*_B_OUTBYPASS
S1	MCPWM_DB*n*_A_OUTBYPASS
S2	MCPWM_DB*n*_RED_OUTINVERT
S3	MCPWM_DB*n*_FED_OUTINVERT
S4	MCPWM_DB*n*_RED_INSEL
S5	MCPWM_DB*n*_FED_INSEL
S6	MCPWM_DB*n*_A_OUTSWAP
S7	MCPWM_DB*n*_B_OUTSWAP
S8	MCPWM_DB*n*_DEB_MODE

All switch combinations are supported, but not all of them represent the typical modes of use. Table 59.3-6 documents some typical dead time configurations. In these configurations, the position of S4 and S5 sets PWM*x*A as the common source of both falling edge delay (FED) and rising edge delay (RED). The modes presented in table 59.3-6 may be categorized as follows:

Table 59.3-6. Typical Dead Time Generator Operating Modes

Mode	Mode Description	S0	S1	S2	S3
1	PWM*x*A and PWM*x*B Pass Through/No Delay	1	1	X	X
2	Active High Complementary (AHC), see Figure 59.3-22	0	0	0	1
3	Active Low Complementary (ALC), see Figure 59.3-23	0	0	1	0
4	Active High (AH), see Figure 59.3-24	0	0	0	0

Mode	Mode Description	S0	S1	S2	S3
5	Active Low (AL), see Figure 59.3-25	0	0	1	1
6	PWM*x*A Output = PWM*x*A In (No Delay) PWM*x*B Output = PWM*x*A Input with Falling Edge Delay	0	1	0 or 1	0 or 1
7	PWM*x*A Output = PWM*x*A Input with Rising Edge Delay PWM*x*B Output = PWM*x*B Input with No Delay	1	0	0 or 1	0 or 1

Note:
For all the modes above, the position of the binary switches S4 to S8 is set to 0.

- **Mode 1: Bypass delays on both FED and RED**
  In this mode, the dead time module is disabled. Signals of PWM*x*A and PWM*x*B pass through without any modifications.
- **Mode 2-5: Classical Dead Time Polarity Settings**
  These four modes represent typical configurations of polarity and should cover the active-high/low modes in available industry power switch gate drivers. The typical waveforms are shown in Figures 59.3-22 to 59.3-25.
- **Modes 6 and 7: Bypass delay on the falling edge (FED) or rising edge (RED)**
  In these two modes, either RED or FED is bypassed. As a result, the corresponding delay is not applied.

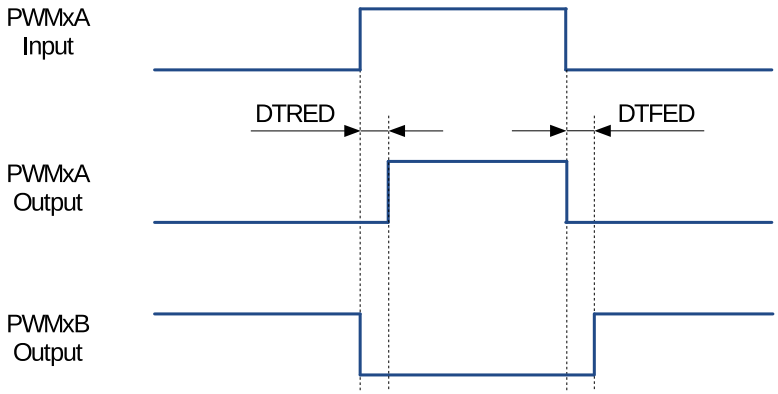


Figure 59.3-22. Active High Complementary (AHC) Dead Time Waveforms

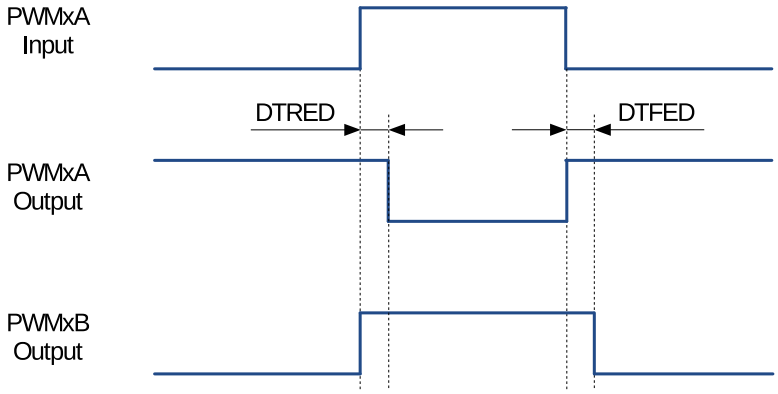


Figure 59.3-23. Active Low Complementary (ALC) Dead Time Waveforms

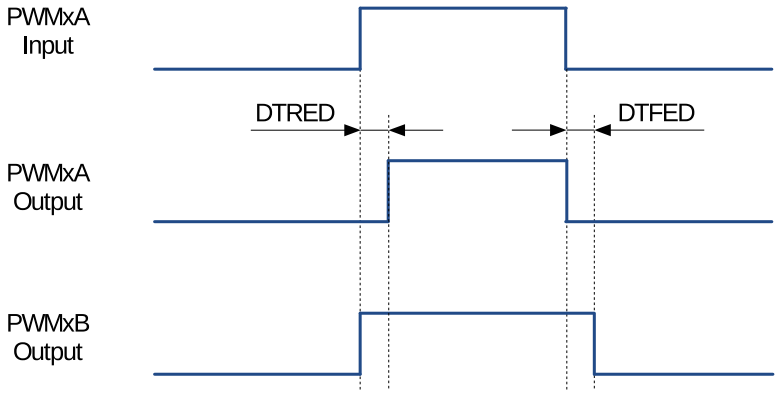


Figure 59.3-24. Active High (AH) Dead Time Waveforms

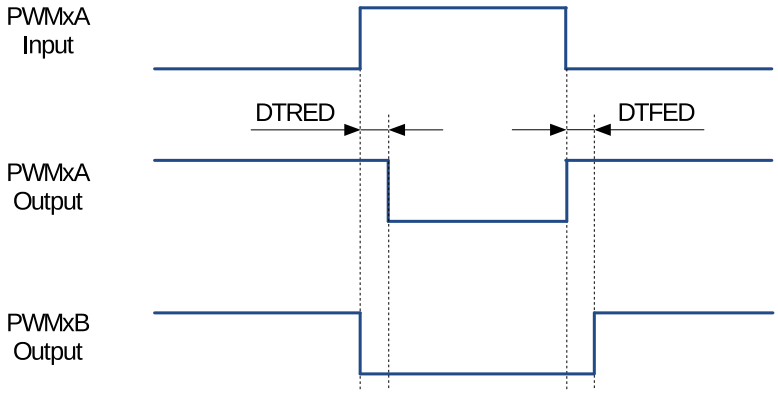


Figure 59.3-25. Active Low (AL) Dead Time Waveforms

RED and FED delays may be set up independently. The delay value is programmed using the 16-bit field MCPWM_DB*n*_RED and MCPWM_DB*n*_FED. The register value represents the number of clock (DT_CLK) periods by which a signal edge is delayed. DT_CLK can be selected from PWM_clk or PT_clk through register MCPWM_DB*n*_CLK_SEL.

To calculate the delay on the falling edge (FED) and rising edge (RED), use the following formulas:

$$FED = MCPWM_DTn_FED \times T_{DT_clk}$$

$$RED = MCPWM_DTn_RED \times T_{DT_clk}$$

59.3.3.3 PWM Carrier Module

The coupling of PWM output to a motor driver may need isolation with a transformer. Transformers deliver only AC signals, while the duty cycle of a PWM signal may range anywhere from 0% to 100%. The PWM carrier module passes such a PWM signal through a transformer by using a high frequency carrier to modulate the signal.

Function Overview

The following key characteristics of this module are configurable:

- Carrier frequency
- Pulse width of the first pulse
- Duty cycle of the second and the subsequent pulses
- Enabling/disabling the carrier function

Operational Highlights

The PWM carrier clock (PC_clk) is derived from PWM_CLK. The frequency and duty cycle are configured by the MCPWM_CHOPPER*n*_PRESCALE and MCPWM_CHOPPER*n*_DUTY bits in the MCPWM_CARRIER*n*_CFG_REG register. The purpose of one-shot pulses is to provide the high-energy impulse to reliably turn on the power switch. Subsequent pulses sustain the power-on status. The width of a one-shot pulse is configurable with the MCPWM_CHOPPER*n*_OSHTWTH field. Enabling/disabling of the carrier module is done with the MCPWM_CHOPPER*n*_EN bit.

Waveform Examples

Figure 59.3-26 shows an example of waveforms, where a carrier is superimposed on original PWM pulses. This figure does not show the first one-shot pulse and the duty-cycle control. Related details are covered in the following two sections.

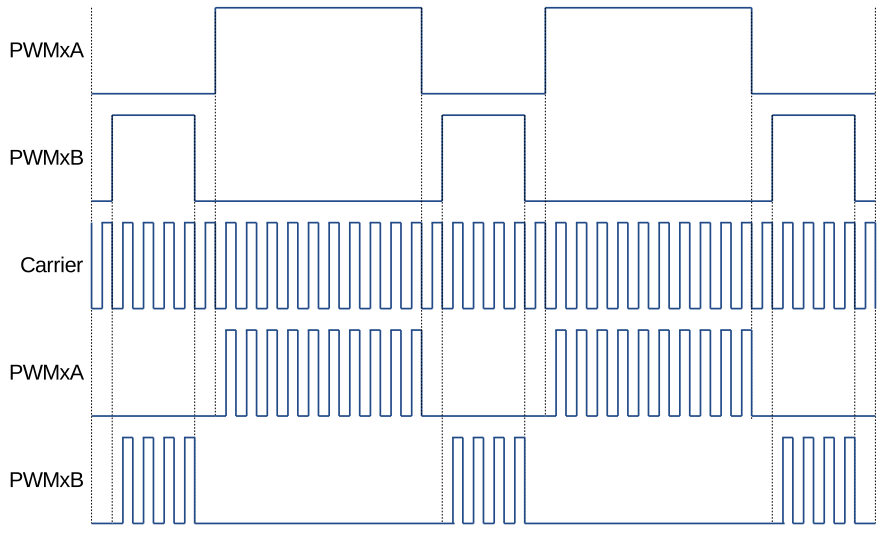


Figure 59.3-26. Example of Waveforms Showing PWM Carrier Action

One-Shot Pulse

The width of the first pulse can be configured to 16 different values, which can be calculated by the following equation:

$$T_{1stpulse} = T_{PWM_clk} \times 8 \times (MCPWM_CARRIERn_PRESCALE + 1) \times (MCPWM_CARRIERn_OSHTWTH + 1)$$

Where:

- T_{PMW_clk} is the period of the PWM clock (PWM_clk).
- $(MCPWM_CARRIERn_OSHTWTH + 1)$ is the width of the first pulse (whose value ranges from 1 to 16).
- $(MCPWM_CARRIERn_PRESCALE + 1)$ is the PWM carrier clock's (PC_clk) prescaler value.

The first one-shot pulse and subsequent sustaining pulses are shown in Figure 59.3-27.

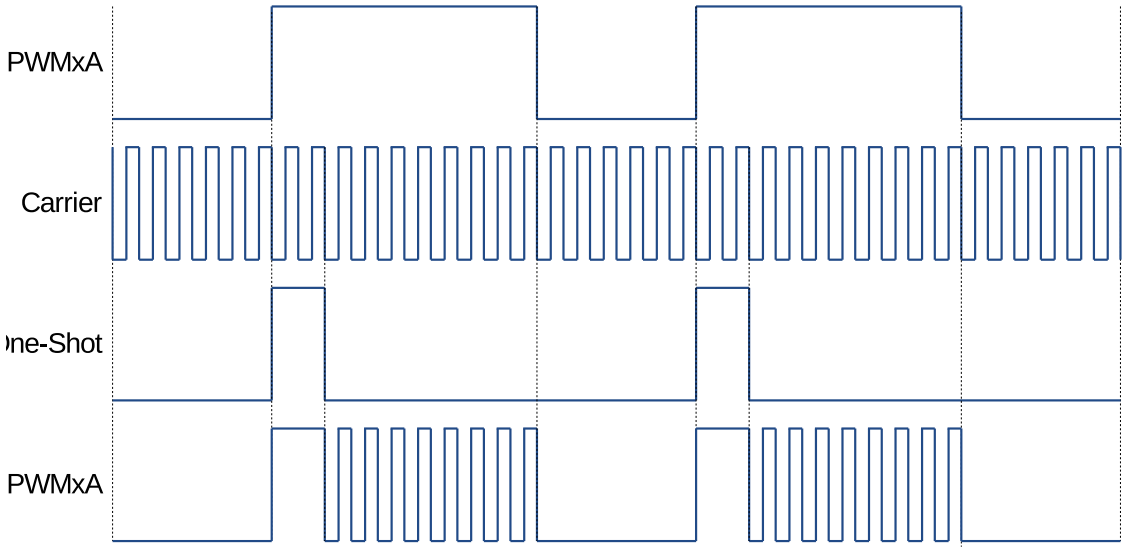


Figure 59.3-27. Example of the First Pulse and the Subsequent Sustaining Pulses of the PWM Carrier Submodule

Duty Cycle Control

After issuing the first one-shot pulse, the remaining PWM signal is modulated according to the carrier frequency. Users can configure the duty cycle of this signal. Tuning of duty may be required, so that the signal passes through the isolating transformer and can still operate (turn on/off) the motor drive, changing rotation speed and direction.

The duty cycle may be set to one of seven values, using MCPWM_CHOPPERn_DUTY, or bits [7:5] of register MCPWM_CARRIERn_CFG_REG.

Below is the formula for calculating the duty cycle:

$$Duty = MCPWM_CARRIERn_DUTY \div 8$$

All seven settings of the duty cycle are shown in Figure 59.3-28.

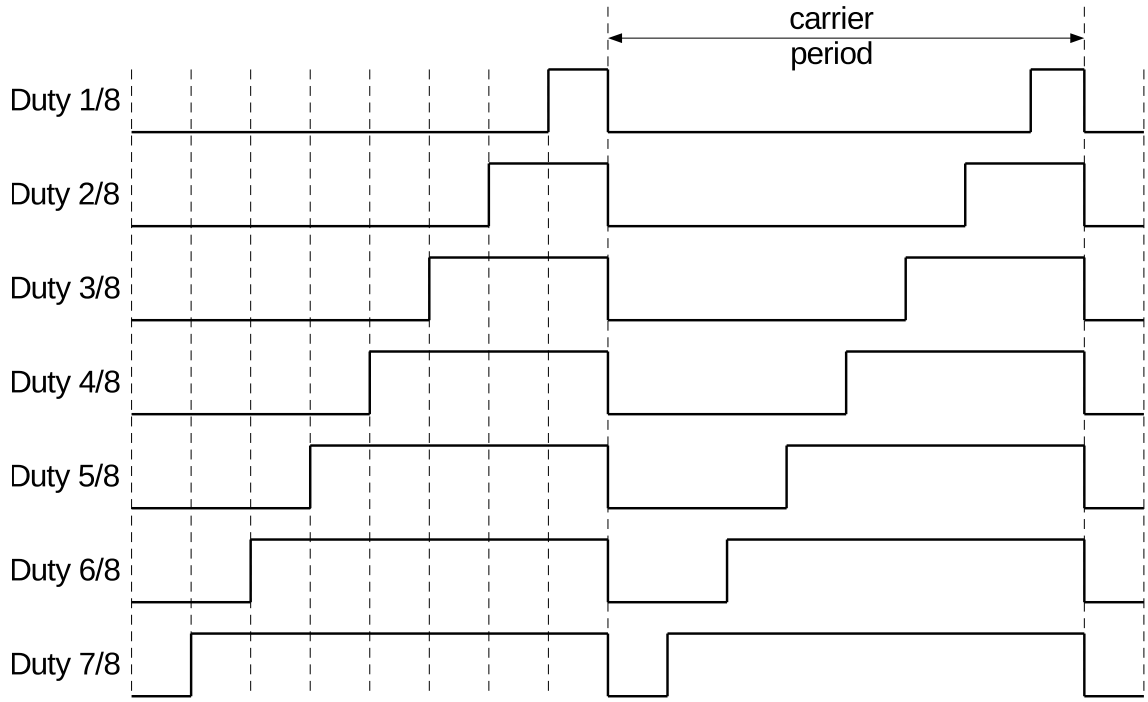


Figure 59.3-28. Possible Duty Cycle Settings for Sustaining Pulses in the PWM Carrier Submodule

59.3.3.4 Fault Detection Module

Each MCPWM peripheral is connected to three fault signals (FAULT0, FAULT1, and FAULT2) which are sourced from the GPIO matrix. These signals are intended to indicate external fault conditions, and may be preprocessed by the Fault Detection module to generate fault events. Fault events can then execute the user code to control MCPWM outputs in response to specific faults.

Function of Fault Detection Module

The key actions performed by the fault detection module are:

- Forcing outputs PWM*x*A and PWM*x*B, upon detected fault, to one of the following states:
  - High
  - Low
  - Toggle
  - No action taken
- Execution of one-shot trip (OST) upon detection of over-current conditions/short circuits.
- Cycle-by-cycle trip (CBC) to provide current-limiting operation.
- Allocation of either one-shot or cycle-by-cycle operation for each fault signal.
- Generation of interrupts for each fault input.
- Support for software-force tripping.
- Enabling or disabling of module function as required.

Operation and Configuration Tips

This section provides the operational tips and set-up options for the Fault Detection module.

Fault signals coming from pins are sampled and synced in the GPIO matrix. In order to guarantee the successful sampling of fault pulses, each pulse duration must be at least two APB clock cycles. The Fault Detection module will then sample fault signals by using PWM_clk. So, the duration of fault pulses coming from GPIO matrix must be at least one PWM_clk cycle. Differently put, regardless of the period relation between APB clock and PWM_clk, the width of fault signal pulses on pins must be at least equal to the sum of two APB clock cycles and one PWM_clk cycle.

Each level of fault signals, FAULT0 to FAULT2, can be used by the Fault Detection module to generate fault events (fault_event0 to fault_event2). Every fault event can be configured individually to provide CBC action, OST action, or none.

- **Cycle-by-Cycle (CBC) action:**
  When CBC action is triggered, the state of PWM*x*A and PWM*x*B will be changed immediately according to the configuration of fields MCPWM_TZ*n*_A_CBC_U/D and MCPWM_TZ*n*_B_CBC_U/D. Different actions can be indicated when the PWM timer is incrementing or decrementing. Different CBC action interrupts can be triggered for different fault events. Status field MCPWM_TZ*n*_CBC_ON indicates whether a CBC action is on or off. When the fault event is no longer present, CBC actions on PWM*x*A/B will be cleared at a specified point, which is either a D/UTEP or D/UTEZ event. Field MCPWM_TZ*n*_CBCPULSE determines at which event PWM*x*A and PWM*x*B will be able to resume normal actions. Therefore, in this mode, the CBC action is cleared or refreshed upon every PWM cycle.
- **One-Shot (OST) action:**
  When OST action is triggered, the state of PWM*x*A and PWM*x*B will be changed immediately, depending on the setting of fields MCPWM_TZ*n*_A_OST_U/D and MCPWM_TZ*n*_B_OST_U/D. Different actions can be configured when PWM timer is incrementing or decrementing. Different OST action interrupts can be triggered from different fault events. Status field MCPWM_TZ*n*_OST_ON indicates whether an OST action is on or off. The OST actions on PWM*x*A/B are not automatically cleared when the fault event is no longer present. One-shot actions must be cleared manually by setting the MCPWM_TZ*n*_CLR_OST bit.

59.3.4 Capture Module

59.3.4.1 Introduction

The capture module contains three complete capture channels. Channel inputs CAP0, CAP1, and CAP2 are sourced from the GPIO matrix. Thanks to the flexibility of the GPIO matrix, CAP0, CAP1, and CAP2 can be configured from any pin input. Multiple capture channels can be sourced from the same pin input, while prescaling for each channel can be set differently. Also, capture channels are sourced from different pins. This provides several options for handling capture signals by hardware in the background, instead of having them processed directly by the CPU. A capture module has the following independent key resources:

- One 32-bit timer (counter) which can be synchronized with the PWM timer, another module, or software.
- Three capture channels, each equipped with a 32-bit time-stamp and a capture prescaler.
- Independent edge polarity (rising/falling edge) selection for any capture channel.
- Input capture signal prescaling (from 1 to 256).

- Interrupt capabilities on any of the three capture events.

59.3.4.2 Capture Timer

The capture timer is a 32-bit counter incrementing continuously. It is enabled by setting MCPWM_CAP_TIMER_EN to 1. Its operating clock source is MCPWM core clock. When MCPWM_CAP_SYNCI_EN is configured, the counter will be loaded with phase stored in register MCPWM_CAP_TIMER_PHASE_REG at the time of a sync event. Sync events can select from PWM timers sync-out, or PWM module sync-in by configuring MCPWM_CAP_SYNCI_SEL. Sync events can also be generated by setting MCPWM_CAP_SYNC_SW. The capture timer provides timing references for all three capture channels.

59.3.4.3 Capture Channel

The capture signal coming to a capture channel will be inverted first, if needed, and then prescaled. Each capture channel has a prescaler register of MCPWM_CAP*n*_PRESCALE. Finally, specified edges of preprocessed capture signal will trigger capture events. Setting MCPWM_CAP*n*_EN to enable a capture channel. The capture event occurs at the time selected by the MCPWM_CAP*n*_MODE. When a capture event occurs, the capture timer's value is stored in time-stamp register MCPWM_CAP_CH*n*_REG. Different interrupts can be generated for different capture channels at capture events. The edge that triggers a capture event is recorded in register MCPWM_CAP*n*_EDGE. The capture event can be also forced by software setting MCPWM_CAP*n*_SW.

59.3.5 ETM Module

59.3.5.1 Overview

The MCPWM peripheral on ESP32-P4 supports the Event Task Matrix (ETM) function, which allows MCPWM's ETM tasks to be triggered by any peripherals' ETM events, or MCPWM's ETM events to trigger any peripherals' ETM tasks. The capture module, the fault detection module, three timers, and three operators can generate events and respond to tasks independently. This section introduces the ETM tasks and events related to MCPWM. For more information, please refer to Chapter 14 Event Task Matrix (ETM).

59.3.5.2 MCPWM-Related ETM Events

When setting enable field to 1, after the generation condition is met, the corresponding event would be generated. For details, please refer to Table 59.3-7 below:

Table 59.3-7. MCPWM-Related ETM Events

Enable Field	Generation Condition	Event Generated
MCPWM_EVT_CAP*n*_EN	CAP*n* capture event occurs	MCPWM*m*_EVT_CAP*n*
MCPWM_EVT_TZ*n*_OST_EN	PWM operator *n* performs a One-Shot trip (OST) operation	MCPWM*m*_EVT_TZ*n*_OST
MCPWM_EVT_TZ*n*_CBC_EN	PWM operator *n* performs a cycle-by-cycle trip (CBC) operation	MCPWM*m*_EVT_TZ*n*_CBC
MCPWM_EVT_F*n*_CLR_EN	Fault event fault_event*n* is cleared	MCPWM*m*_EVT_F*n*_CLR

[1] See Section 59.3.3.1 for a detailed description of timer stamp A and B.

Enable Field	Generation Condition	Event Generated
MCPWM_EVT_F*n*_EN	Fault event fault_event*n* is generated	MCPWM*m*_EVT_F*n*
MCPWM_EVT_OP*n*_TEB_EN	The count value of the timer that PWM operator *n* selects is equal to the value of timer stamp B[1]	MCPWM*m*_EVT_OP*n*_TEB
MCPWM_EVT_OP*n*_TEA_EN	The count value of the timer that PWM operator *n* selects is equal to the value of timer stamp A[1]	MCPWM*m*_EVT_OP*n*_TEA
MCPWM_EVT_OP*n*_TEE1_EN	The count value of the timer that PWM operator *n* selects is equal to the value of register MCPWM_OP*n*_TSTMP_E1_REG	MCPWM*m*_EVT_OP*n*_TEE1
MCPWM_EVT_OP*n*_TEE2_EN	The count value of the timer that PWM operator *n* selects is equal to the value of register MCPWM_OP*n*_TSTMP_E2_REG	MCPWM*m*_EVT_OP*n*_TEE2
MCPWM_EVT_TIMER*n*_TEP_EN	The count value of timer *n* is equal to the period value MCPWM_TIMER*n*_PERIOD	MCPWM*m*_EVT_TIMER*n*_TEP
MCPWM_EVT_TIMER*n*_TEZ_EN	The count value of timer *n* is equal to 0	MCPWM*m*_EVT_TIMER*n*_TEZ
MCPWM_EVT_TIMER*n*_STOP_EN	Timer *n*'s count stops	MCPWM*m*_EVT_TIMER*n*_STOP

[1] See Section 59.3.3.1 for a detailed description of timer stamp A and B.

59.3.5.3 MCPWM-Related ETM Tasks

When setting the enable field to 1, after inputting valid tasks, the corresponding response operation would be generated. For details, please refer to Table 59.3-8 below:

Table 59.3-8. ETM Related Tasks

Enable Field	Valid Task Input	Response Operation
MCPWM_TASK_CAP*n*_EN	MCPWM*m*_TASK_CAP*n*	CAP*n* channel performs a capture operation
MCPWM_TASK_CLR*n*_OST_EN	MCPWM*m*_TASK_CLR*n*_OST	PWM operator *n* clears the One-Shot Trip operation
MCPWM_TASK_TZ*n*_OST_EN	MCPWM*m*_TASK_TZ*n*_OST	PWM operator *n* performs a One-Shot Trip (OST) operation
MCPWM_TASK_TIMER*n*_PERIOD_UP_EN	MCPWM*m*_TASK_TIMER*n*_PERIOD_UP	The period of timer *n* is updated to the value configured in the period register MCPWM_TIMER*n*_PERIOD
MCPWM_TASK_TIMER*n*_SYNC_EN	MCPWM*m*_TASK_TIMER*n*_SYN	Timer *n* performs a sync operation
MCPWM_TASK_GEN_STOP_EN	MCPWM*m*_TASK_GEN_STOP	All the timers stop counting and the PWM signals output by all PWM operators remain unchanged

Enable Field	Valid Task Input	Response Operation
MCPWM_TASK_CMPR*n*_B_UP_EN	MCPWM*m*_TASK_CMPR*n*_B_UP	Timer stamp B of the PWM operator *n* is updated to the value of the shadow register MCPWM_GEN*n*_B
MCPWM_TASK_CMPR*n*_A_UP_EN	MCPWM*m*_TASK_CMPR*n*_A_UP	Timer stamp A of the PWM operator *n* is updated to the value of the shadow register MCPWM_GEN*n*_A

59.4 Interrupts

ESP32-P4's MCPWM0 can generate the PWM0_INTR interrupt signal and MCPWM1 can generate the PWM1_INTR interrupt signal. The PWM*n*_INTR will be sent to the Interrupt Matrix.

Interrupt signal PWM*n*_INTR can be generated by the following internal interrupt sources:

- MCPWM_TIMER*x*_STOP_INT: Triggered when timer*x* stops.
- MCPWM_TIMER*x*_TEZ_INT: Triggered by the TEZ event of PWM timer*x*.
- MCPWM_TIMER*x*_TEP_INT: Triggered by the TEP event of PWM timer*x*.
- MCPWM_FAULT*x*_INT: Triggered when fault_event*x* starts.
- MCPWM_FAULT*x*_CLR_INT: Triggered after fault_event*x* ends.
- MCPWM_CMPR*x*_TEA_INT: Triggered by the TEA event of PWM operator*x*.
- MCPWM_CMPR*x*_TEB_INT: Triggered by the TEB event of PWM operator*x*.
- MCPWM_TZ*x*_CBC_INT: Triggered by the CBC action of PWM*x*.
- MCPWM_TZ*x*_OST_INT: Triggered by the OST action of PWM*x*.
- MCPWM_CAP*x*_INT: Triggered by the capture event on channel*x*.

The above interrupt sources can only be triggered when the interrupt enable register *interrupt_source_name*_ENA is set to 1. Write 1 to *interrupt_source_name*_CLR will clear the interrupt.

Note:
For definitions of *interrupt*, *interrupt signal*, *interrupt source*, and their correlations, please refer to Chapter 13 *Interrupt Matrix* > Section 13.2 *Interrupt Terminology in ESP32-P4*.

Each interrupt source can be configured by a common set of registers that are described in Section *Interrupt Configuration Registers*. The specific registers can be found in Section 59.5 *Register Summary*.

59.5 Register Summary

The addresses in this section are relative to Motor Control PWM base address provided in Table 9.3-2 in Chapter 9 *System and Memory*.

The abbreviations given in Column **Access** are explained in Section *Access Types for Registers*.

Name	Description	Address	Access
Prescaler Configuration			
MCPWM_CLK_CFG_REG	PWM clock prescaler register	0x0000	R/W
MCPWM Timer 0 Configuration and Status			
MCPWM_TIMER0_CFG0_REG	PWM timer0 period and update method configuration register	0x0004	R/W
MCPWM_TIMER0_CFG1_REG	PWM timer0 working mode and start/stop control configuration register	0x0008	varies
MCPWM_TIMER0_SYNC_REG	PWM timer0 sync function configuration register	0x000C	R/W
MCPWM_TIMER0_STATUS_REG	PWM timer0 status register	0x0010	RO
MCPWM Timer 1 Configuration and Status			
MCPWM_TIMER1_CFG0_REG	PWM timer1 period and update method configuration register	0x0014	R/W
MCPWM_TIMER1_CFG1_REG	PWM timer1 working mode and start/stop control configuration register	0x0018	varies
MCPWM_TIMER1_SYNC_REG	PWM timer1 sync function configuration register	0x001C	R/W
MCPWM_TIMER1_STATUS_REG	PWM timer1 status register	0x0020	RO
MCPWM Timer 2 Configuration and status			
MCPWM_TIMER2_CFG0_REG	PWM timer2 period and update method configuration register	0x0024	R/W
MCPWM_TIMER2_CFG1_REG	PWM timer2 working mode and start/stop control configuration register	0x0028	varies
MCPWM_TIMER2_SYNC_REG	PWM timer2 sync function configuration register	0x002C	R/W
MCPWM_TIMER2_STATUS_REG	PWM timer2 status register	0x0030	RO
Common Configuration for MCPWM Timers			
MCPWM_TIMER_SYNCI_CFG_REG	Synchronization input selection for three PWM timers	0x0034	R/W
MCPWM_OPERATOR_TIMERSEL_REG	Select specific timer for PWM operators	0x0038	R/W
MCPWM Operator 0 Configuration and Status			
MCPWM_GEN0_STMP_CFG_REG	Transfer status and update method for time stamp registers A and B	0x003C	varies
MCPWM_GEN0_TSTMP_A_REG	Shadow register for register A	0x0040	R/W
MCPWM_GEN0_TSTMP_B_REG	Shadow register for register B	0x0044	R/W
MCPWM_GEN0_CFG0_REG	Fault event T0 and T1 handling	0x0048	R/W
MCPWM_GEN0_FORCE_REG	Permissives to force PWM0A and PWM0B outputs by software	0x004C	R/W

Name	Description	Address	Access
MCPWM_GEN0_A_REG	Actions triggered by events on PWM0A	0x0050	R/W
MCPWM_GEN0_B_REG	Actions triggered by events on PWM0B	0x0054	R/W
MCPWM_DT0_CFG_REG	Dead time type selection and configuration	0x0058	R/W
MCPWM_DT0_FED_CFG_REG	Shadow register for falling edge delay (FED)	0x005C	R/W
MCPWM_DT0_RED_CFG_REG	Shadow register for rising edge delay (RED)	0x0060	R/W
MCPWM_CARRIER0_CFG_REG	Carrier enable and configuration	0x0064	R/W
MCPWM_FH0_CFG0_REG	Actions on PWM0A and PWM0B trip events	0x0068	R/W
MCPWM_FH0_CFG1_REG	Software triggers for fault handler actions	0x006C	R/W
MCPWM_FH0_STATUS_REG	Status of fault events	0x0070	RO
MCPWM Operator 1 Configuration and Status			
MCPWM_GEN1_STMP_CFG_REG	Transfer status and update method for time stamp registers A and B	0x0074	varies
MCPWM_GEN1_TSTMP_A_REG	Shadow register for register A	0x0078	R/W
MCPWM_GEN1_TSTMP_B_REG	Shadow register for register B	0x007C	R/W
MCPWM_GEN1_CFG0_REG	Fault event T0 and T1 handling	0x0080	R/W
MCPWM_GEN1_FORCE_REG	Permissives to force PWM1A and PWM1B outputs by software	0x0084	R/W
MCPWM_GEN1_A_REG	Actions triggered by events on PWM1A	0x0088	R/W
MCPWM_GEN1_B_REG	Actions triggered by events on PWM1B	0x008C	R/W
MCPWM_DT1_CFG_REG	Dead time type selection and configuration	0x0090	R/W
MCPWM_DT1_FED_CFG_REG	Shadow register for falling edge delay (FED)	0x0094	R/W
MCPWM_DT1_RED_CFG_REG	Shadow register for rising edge delay (RED)	0x0098	R/W
MCPWM_CARRIER1_CFG_REG	Carrier enable and configuration	0x009C	R/W
MCPWM_FH1_CFG0_REG	Actions on PWM1A and PWM1B trip events	0x00A0	R/W
MCPWM_FH1_CFG1_REG	Software triggers for fault handler actions	0x00A4	R/W
MCPWM_FH1_STATUS_REG	Status of fault events	0x00A8	RO
MCPWM Operator 2 Configuration and Status			
MCPWM_GEN2_STMP_CFG_REG	Transfer status and update method for time stamp registers A and B	0x00AC	varies
MCPWM_GEN2_TSTMP_A_REG	Shadow register for register A	0x00B0	R/W
MCPWM_GEN2_TSTMP_B_REG	Shadow register for register B	0x00B4	R/W
MCPWM_GEN2_CFG0_REG	Fault event T0 and T1 handling	0x00B8	R/W
MCPWM_GEN2_FORCE_REG	Permissives to force PWM2A and PWM2B outputs by software	0x00BC	R/W
MCPWM_GEN2_A_REG	Actions triggered by events on PWM2A	0x00C0	R/W
MCPWM_GEN2_B_REG	Actions triggered by events on PWM2B	0x00C4	R/W
MCPWM_DT2_CFG_REG	Dead time type selection and configuration	0x00C8	R/W
MCPWM_DT2_FED_CFG_REG	Shadow register for falling edge delay (FED)	0x00CC	R/W
MCPWM_DT2_RED_CFG_REG	Shadow register for rising edge delay (RED)	0x00D0	R/W
MCPWM_CARRIER2_CFG_REG	Carrier enable and configuration	0x00D4	R/W
MCPWM_FH2_CFG0_REG	Actions on PWM2A and PWM2B trip events	0x00D8	R/W
MCPWM_FH2_CFG1_REG	Software triggers for fault handler actions	0x00DC	R/W
MCPWM_FH2_STATUS_REG	Status of fault events	0x00E0	RO

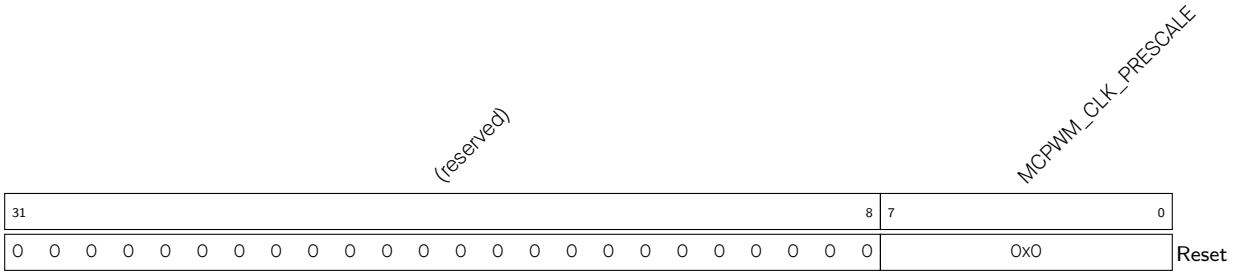
Name	Description	Address	Access
Fault Detection Configuration and Status			
MCPWM_FAULT_DETECT_REG	Fault detection configuration and status	0x00E4	varies
Capture Configuration and Status			
MCPWM_CAP_TIMER_CFG_REG	Configure capture timer	0x00E8	varies
MCPWM_CAP_TIMER_PHASE_REG	Phase for capture timer sync	0x00EC	R/W
MCPWM_CAP_CH0_CFG_REG	Capture channel 0 configuration and enable	0x00F0	varies
MCPWM_CAP_CH1_CFG_REG	Capture channel 1 configuration and enable	0x00F4	varies
MCPWM_CAP_CH2_CFG_REG	Capture channel 2 configuration and enable	0x00F8	varies
MCPWM_CAP_CH0_REG	ch0 capture value status register	0x00FC	RO
MCPWM_CAP_CH1_REG	ch1 capture value status register	0x0100	RO
MCPWM_CAP_CH2_REG	ch2 capture value status register	0x0104	RO
MCPWM_CAP_STATUS_REG	Edge of last capture trigger	0x0108	RO
Enable Update of Active Registers			
MCPWM_UPDATE_CFG_REG	Enable update	0x010C	R/W
Manage Interrupts			
MCPWM_INT_ENA_REG	Interrupt enable bits	0x0110	R/W
MCPWM_INT_RAW_REG	Raw interrupt status	0x0114	R/WTC/SS
MCPWM_INT_ST_REG	Masked interrupt status	0x0118	RO
MCPWM_INT_CLR_REG	Interrupt clear bits	0x011C	WT
MCPWM Event Enable Registers			
MCPWM_EVT_EN_REG	MCPWM event enable register	0x0120	R/W
MCPWM_EVT_EN2_REG	MCWM event enable register2	0x0128	R/W
MCPWM Task Enable Register			
MCPWM_TASK_EN_REG	MCPWM task enable register	0x0124	R/W
MCPWM Generator Configuration Registers			
MCPWM_OP0_TSTMP_E1_REG	Generator0 time stamp E1 value configuration register	0x012C	R/W
MCPWM_OP0_TSTMP_E2_REG	Generator0 time stamp E2 value configuration register	0x0130	R/W
MCPWM_OP1_TSTMP_E1_REG	Generator1 time stamp E1 value configuration register	0x0134	R/W
MCPWM_OP1_TSTMP_E2_REG	Generator1 time stamp E2 value configuration register	0x0138	R/W
MCPWM_OP2_TSTMP_E1_REG	Generator2 time stamp E1 value configuration register	0x013C	R/W
MCPWM_OP2_TSTMP_E2_REG	Generator2 time stamp E2 value configuration register	0x0140	R/W
MCPWM APB Configuration Register			
MCPWM_CLK_REG	MCPWM APB configuration register	0x0144	R/W
Version Register			
MCPWM_VERSION_REG	Version control register	0x0148	R/W

59.6 Registers

The addresses in this section are relative to Motor Control PWM base address provided in Table 9.3-2 in Chapter 9 *System and Memory*.

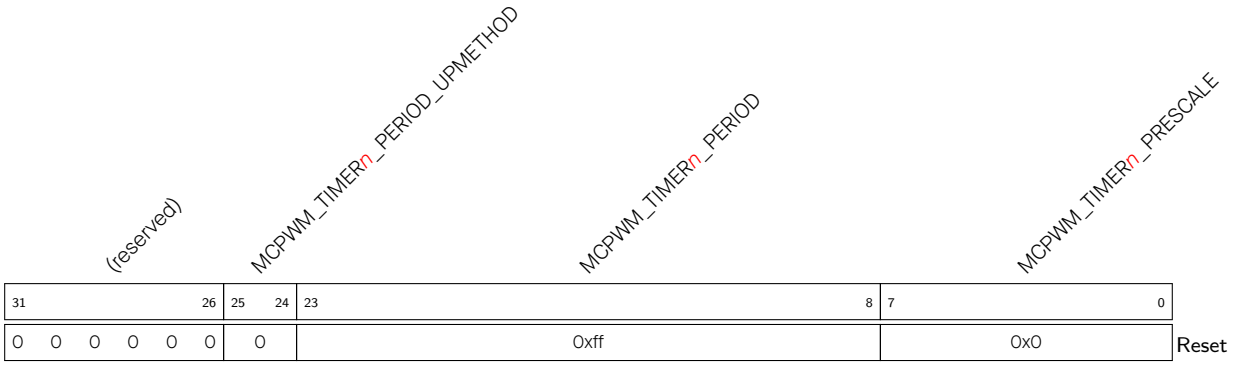
For how to program reserved fields, please refer to Section Programming Reserved Register Field.

Register 59.1. MCPWM_CLK_CFG_REG (0x0000)



MCPWM_CLK_PRESCALE Configures the prescaler value of the clock, so that the period of PWM_clk = 6.25ns * (PWM_CLK_PRESCALE + 1). (R/W)

Register 59.2. MCPWM_TIMER*n*_CFG0_REG (*n*: 0-2) (0x0004+0x10n*)**



MCPWM_TIMER*n*_PRESCALE Configures the prescaler value of timer*n*, so that the period of PTO_clk = Period of PWM_clk * (PWM_TIMER*n*_PRESCALE + 1). (R/W)

MCPWM_TIMER*n*_PERIOD Configures the period shadow of PWM timer*n*. (R/W)

MCPWM_TIMER*n*_PERIOD_UPMETHOD Configures the update method for active register of PWM timer*n* period.

0: Immediate

1: TEZ

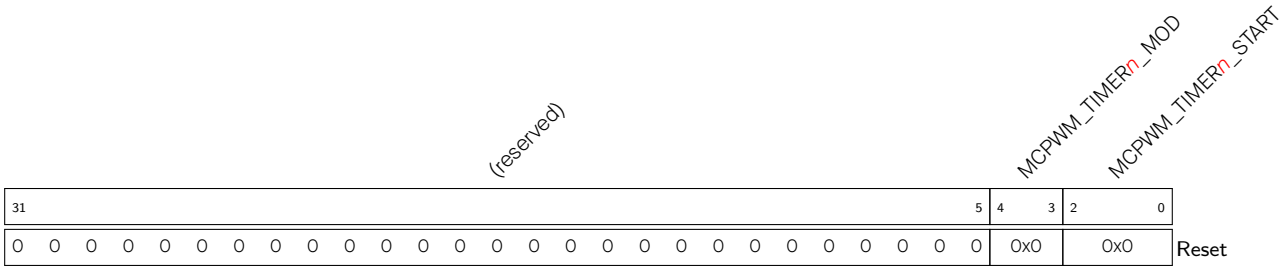
2: Sync

3: TEZ or sync

TEZ here and below means timer equals zero event.

(R/W)

Register 59.3. MCPWM_TIMER*n*_CFG1_REG(*n*: 0-2) (0x0008+0x10n*)**



MCPWM_TIMER*n*_START Configures conditions to start/stop PWM timer*n*.

0: If PWM timer0 starts, then stops at TEZ

1: If timer0 starts, then stops at TEP

2: PWM timer0 starts and runs on

3: Timer0 starts and stops at the next TEZ

4: Timer0 starts and stops at the next TEP

5: Invalid. No effect

6: Invalid. No effect

7: Invalid. No effect

TEP here and below means the event that happens when the timer equals to period.

(R/W/SC)

MCPWM_TIMER*n*_MOD Configures the working mode of PWM timer*n*.

0: Freeze

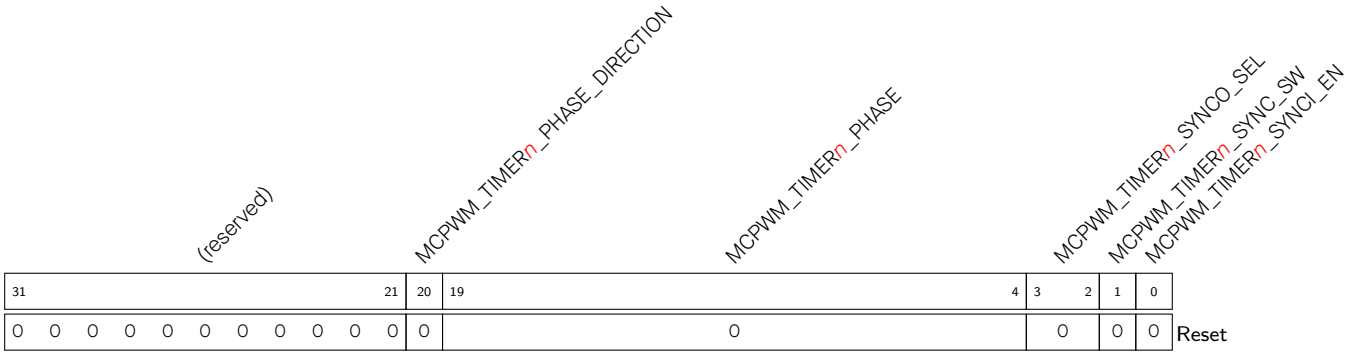
1: Increase mode

2: Decrease mode

3: Up-down mode

(R/W)

Register 59.4. MCPWM_TIMER*n*_SYNC_REG (*n*: 0-2) (0x000C+0x10**n*)



MCPWM_TIMER*n*_SYNCI_EN Configures whether to enable timer*n* reloading with phase on sync input event.

0: Disable

1: Enable

(R/W)

MCPWM_TIMER*n*_SYNC_SW Configures whether to trigger a software sync.

0: No effect

1: Trigger a software sync

(R/W)

MCPWM_TIMER*n*_SYNCO_SEL Configures PWM timer*n* sync out selection.

0: sync_in. The sync out will always generate when toggling the MCPWM_TIMER0_SYNC_SW bit.

1: TEZ

2: TEP

3: No effect

(R/W)

MCPWM_TIMER*n*_PHASE Configures the phase for timer*n* reload on sync event.

(R/W)

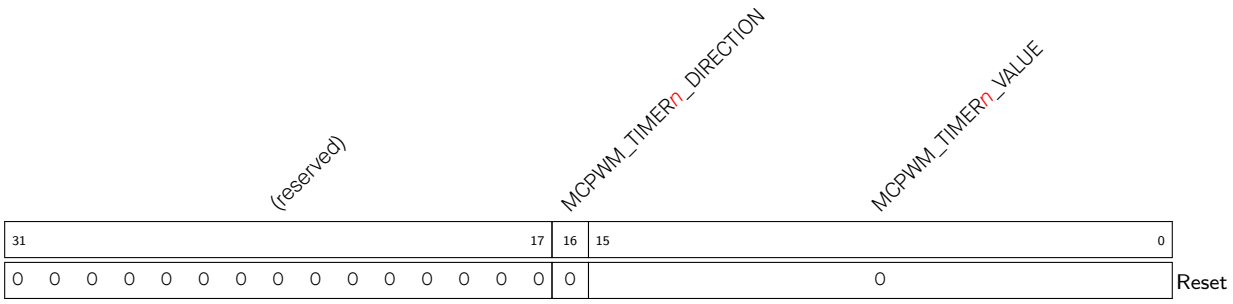
MCPWM_TIMER*n*_PHASE_DIRECTION Configures the PWM timer*n*'s direction when timer*n* mode is up-down mode.

0: Increase

1: Decrease

(R/W)

Register 59.5. MCPWM_TIMER*n*_STATUS_REG(*n*: 0-2) (0x0010+0x10**n*)



MCPWM_TIMER*n*_VALUE Represents current PWM timer*n* counter value. (RO)

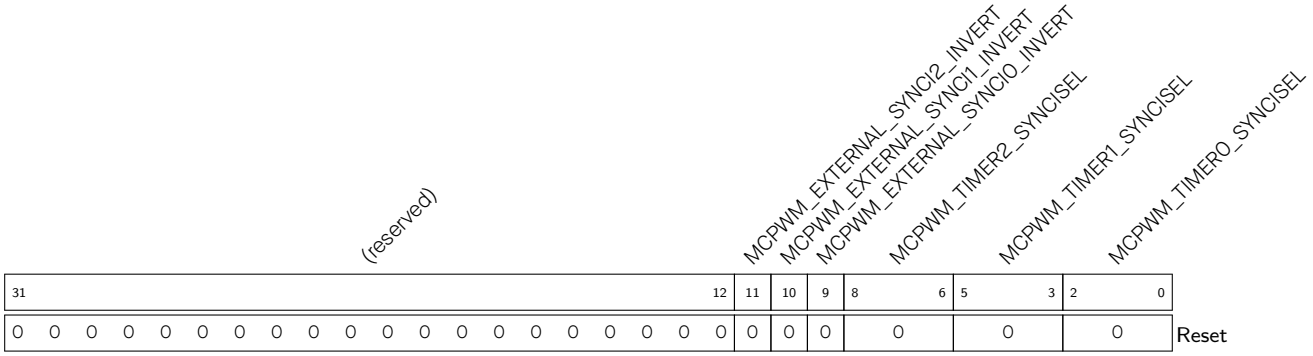
MCPWM_TIMER*n*_DIRECTION Represents current PWM timer*n* counter direction.

0: Increment

1: Decrement

(RO)

Register 59.6. MCPWM_TIMER_SYNCI_CFG_REG (0x0034)



MCPWM_TIMER*n*_SYNCISEL Configures sync input for PWM timer*n*.

1: PWM timer0 sync out

2: PWM timer1 sync out

3: PWM timer2 sync out

4: SYNC0 from GPIO matrix

5: SYNC1 from GPIO matrix

6: SYNC2 from GPIO matrix

Other values: no sync input selected

(R/W)

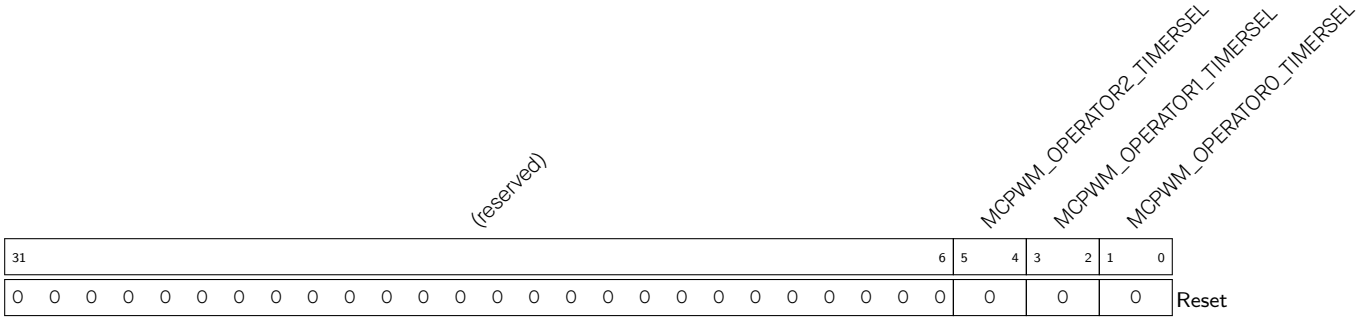
MCPWM_EXTERNAL_SYNCI*n*_INVERT Configures whether to invert SYNC*n* from GPIO matrix.

0: Not invert

1: Invert

(R/W)

Register 59.7. MCPWM_OPERATOR_TIMERSEL_REG (0x0038)



MCPWM_OPERATOR*n*_TIMERSEL Configures which PWM timer will be the timing reference for PWM operator*n*.

0: timer0

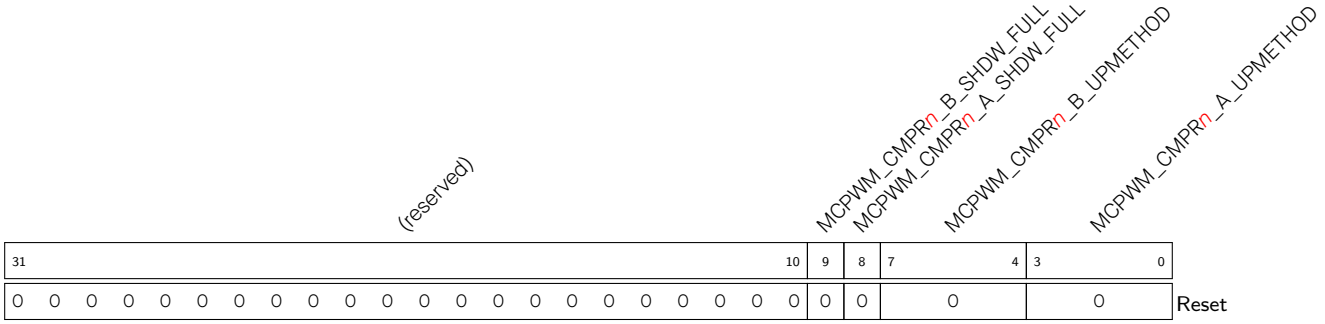
1: timer1

2: timer2

3: Invalid, will select timer2

(R/W)

Register 59.8. MCPWM_GEN*n*_STMP_CFG_REG (*n*: 0-2) (0x003C+0x38n*)**



MCPWM_CMPR*n*_A_UPMETHOD Configures the update method for PWM generator*n* time stamp A's active register.

When all bits are set to 0: Immediately

When bit0 is set to 1: TEZ

When bit1 is set to 1: TEP

When bit2 is set to 1: Sync

When bit3 is set to 1: Disable the update

(R/W)

MCPWM_CMPR*n*_B_UPMETHOD Configures the update method for PWM generator*n* time stamp B's active register.

When all bits are set to 0: Immediately

When bit0 is set to 1: TEZ

When bit1 is set to 1: TEP

When bit2 is set to 1: Sync

When bit3 is set to 1: Disable the update

(R/W)

MCPWM_CMPR*n*_A_SHDW_FULL Configures whether the value in the shadow register is written to the corresponding active register at a specific time. This field is set and reset by hardware.

0: Write the latest value in the shadow register to A's active register

1: Write the value to the PWM generator*n* time stamp A's shadow register, and wait to be transferred to A's active register

(R/SC/WTC)

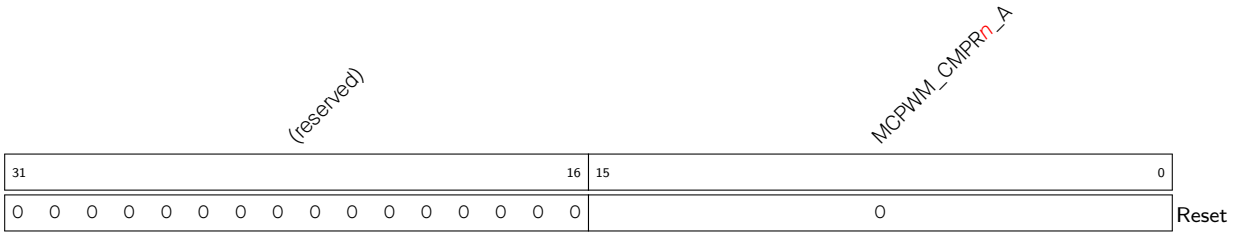
MCPWM_CMPR*n*_B_SHDW_FULL Configures whether the value in the shadow register is written to the corresponding active register at a specific time. This field is set and reset by hardware.

0: Write the latest value in the shadow register to B's active register

1: Write the value to the PWM generator*n* time stamp B's shadow register, and wait to be transferred to B's active register

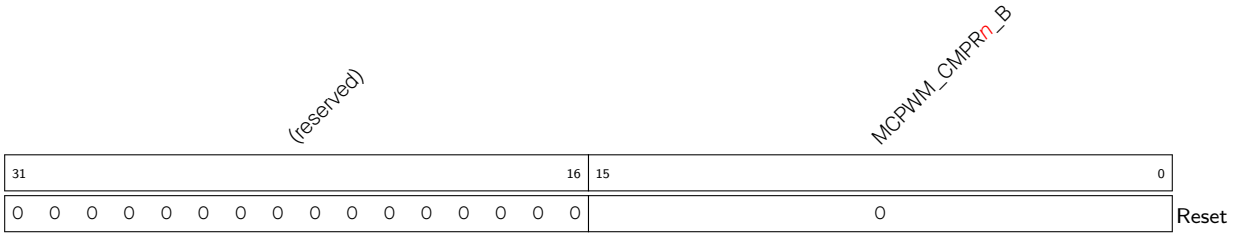
(R/SC/WTC)

Register 59.9. MCPWM_GEN*n*_TSTMP_A_REG(*n*: 0-2) (0x0040+0x38**n*)



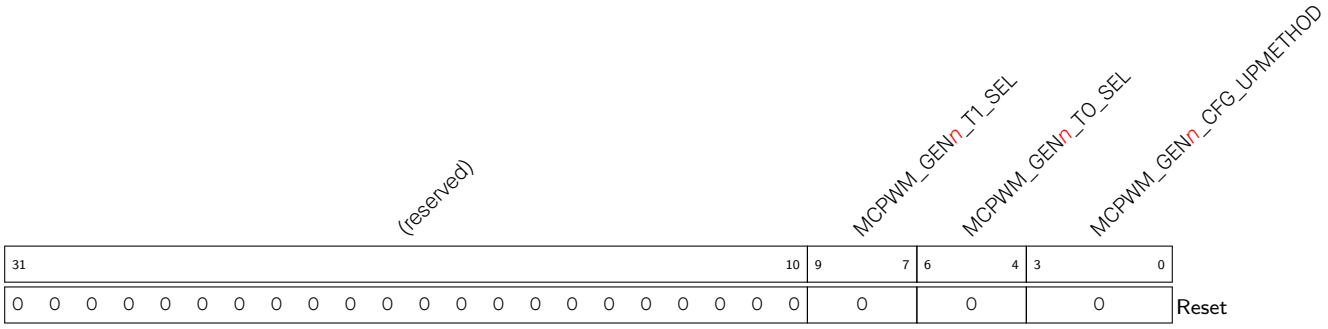
MCPWM_CMPR*n*_A Configures the value of PWM generator *n* time stamp A's shadow register. (R/W)

Register 59.10. MCPWM_GEN*n*_TSTMP_B_REG(*n*: 0-2) (0x0044+0x38**n*)



MCPWM_CMPR*n*_B Configures the value of PWM generator *n* time stamp B's shadow register. (R/W)

Register 59.11. MCPWM_GEN*n*_CFG0_REG(*n*: 0-2) (0x0048+0x38**n*)



MCPWM_GEN*n*_CFG_UPMETHOD Configures the update method for PWM generator*n*'s active register.

When all bits are set to 0: Immediately

When bit0 is set to 1: TEZ

When bit1 is set to 1: TEP

When bit2 is set to 1: Sync

When bit3 is set to 1: Disable the update

(R/W)

MCPWM_GEN*n*_T0_SEL Configures source selection for PWM generator*n* event_t0, take effect immediately.

0: fault_event0

1: fault_event1

2: fault_event2

3: sync_taken

4: Invalid, selects nothing

(R/W)

MCPWM_GEN*n*_T1_SEL Configures source selection for PWM generator*n* event_t1, take effect immediately.

0: fault_event0

1: fault_event1

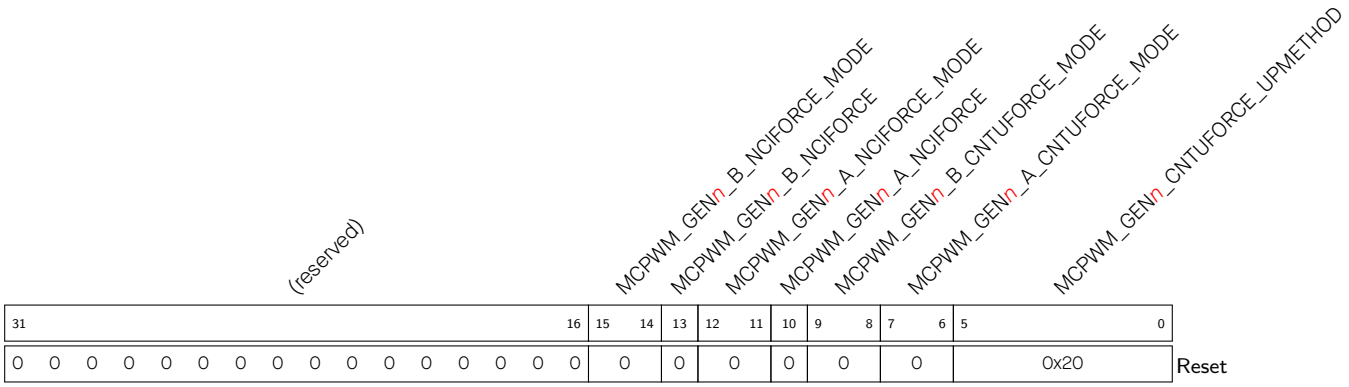
2: fault_event2

3: sync_taken

4: Invalid, selects nothing

(R/W)

Register 59.12. MCPWM_GEN*n*_FORCE_REG(*n*: 0-2) (0x004C+0x38**n*)



MCPWM_GEN*n*_CNTUFORCE_UPMETHOD Configures the update method for continuous software force of PWM generator*n*.

When all bits are set to 0: Immediately

When bit0 is set to 1: TEZ

When bit1 is set to 1: TEP

When bit2 is set to 1: TEA

When bit3 is set to 1: TEB

When bit4 is set to 1: Sync

When bit5 is set to 1: Disable update

TEA/B means an event generated when the timer's value equals to that of register A/B. (R/W)

MCPWM_GEN*n*_A_CNTUFORCE_MODE Configures the continuous software force mode for PWM*n* A.

0: Disabled

1: Low

2: High

3: Disabled

(R/W)

MCPWM_GEN*n*_B_CNTUFORCE_MODE Configures the continuous software force mode for PWM*n* B.

0: Disabled

1: Low

2: High

3: Disabled

(R/W)

MCPWM_GEN*n*_A_NCIFORCE Configures whether to trigger the non-continuous immediate software-force event for PWM*n* A.

0: Invalid

1: Trigger a force event

(R/W)

Continued on the next page...

Register 59.12. MCPWM_GEN*n*_FORCE_REG(*n*: 0-2) (0x004C+0x38**n*)

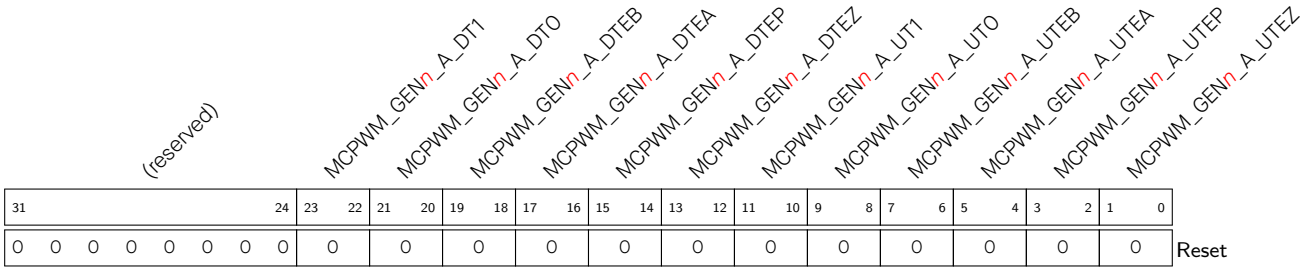
Continued from the previous page...

MCPWM_GEN*n*_A_NCIFORCE_MODE Configures non-continuous immediate software force mode for PWM*n* A.
0: Disabled
1: Low
2: High
3: Disabled
(R/W)

MCPWM_GEN*n*_B_NCIFORCE Configures whether to trigger the non-continuous immediate software-force event for PWM*n* B.
0: Invalid
1: Trigger a force event
(R/W)

MCPWM_GEN*n*_B_NCIFORCE_MODE Configures non-continuous immediate software force mode for PWM*n* B.
0: Disabled
1: Low
2: High
3: Disabled
(R/W)

Register 59.13. MCPWM_GEN*n*_A_REG(*n*: 0-2) (0x0050+0x38**n*)



MCPWM_GEN*n*_A_UTEZ Configures action on PWM*n* A triggered by event TEZ when timer increasing.

0: No change

1: Low

2: High

3: Toggle

(R/W)

MCPWM_GEN*n*_A_UTEP Configures action on PWM*n* A triggered by event TEP when timer increasing.

0: No change

1: Low

2: High

3: Toggle

(R/W)

MCPWM_GEN*n*_A_UTEA Configures action on PWM*n* A triggered by event TEA when timer increasing.

0: No change

1: Low

2: High

3: Toggle

(R/W)

MCPWM_GEN*n*_A_UTEB Configures action on PWM*n* A triggered by event TEB when timer increasing.

0: No change

1: Low

2: High

3: Toggle

(R/W)

Continued on the next page...

Register 59.13. MCPWM_GEN*n*_A_REG(*n*: 0-2) (0x0050+0x38**n*)

Continued from the previous page...

MCPWM_GEN*n*_A_UT0 Configures action on PWM*n* A triggered by event_t0 when timer increasing.
0: No change
1: Low
2: High
3: Toggle
(R/W)

MCPWM_GEN*n*_A_UT1 Configures action on PWM*n* A triggered by event_t1 when timer increasing.
0: No change
1: Low
2: High
3: Toggle
(R/W)

MCPWM_GEN*n*_A_DTEZ Configures action on PWM*n* A triggered by event TEZ when timer decreasing.
0: No change
1: Low
2: High
3: Toggle
(R/W)

MCPWM_GEN*n*_A_DTEP Configures action on PWM*n* A triggered by event TEP when timer decreasing.
0: No change
1: Low
2: High
3: Toggle
(R/W)

MCPWM_GEN*n*_A_DTEA Configures action on PWM*n* A triggered by event TEA when timer decreasing.
0: No change
1: Low
2: High
3: Toggle
(R/W)

Continued on the next page...

Register 59.13. MCPWM_GEN*n*_A_REG(*n*: 0-2) (0x0050+0x38**n*)

Continued from the previous page...

MCPWM_GEN*n*_A_DTEB Configures action on PWM*n* A triggered by event TEB when timer decreasing.

0: No change

1: Low

2: High

3: Toggle

(R/W)

MCPWM_GEN*n*_A_DT0 Configures action on PWM*n* A triggered by event_t0 when timer decreasing.

0: No change

1: Low

2: High

3: Toggle

(R/W)

MCPWM_GEN*n*_A_DT1 Configures action on PWM*n* A triggered by event_t1 when timer decreasing.

0: No change

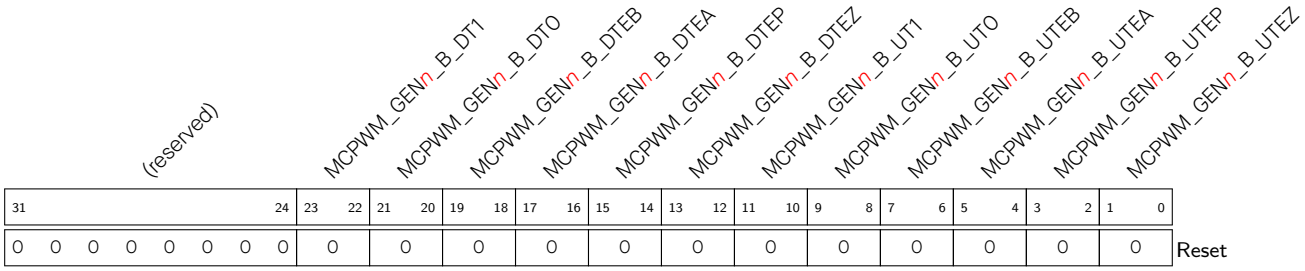
1: Low

2: High

3: Toggle

(R/W)

Register 59.14. MCPWM_GEN*n*_B_REG(*n*: 0-2) (0x0054+0x38**n*)



MCPWM_GEN*n*_B_UTEZ Configures action on PWM*n* B triggered by event TEZ when timer increasing.

0: No change

1: Low

2: High

3: Toggle

(R/W)

MCPWM_GEN*n*_B_UTEP Configures action on PWM*n* B triggered by event TEP when timer increasing.

0: No change

1: Low

2: High

3: Toggle

(R/W)

MCPWM_GEN*n*_B_UTEA Configures action on PWM*n* B triggered by event TEA when timer increasing.

0: No change

1: Low

2: High

3: Toggle

(R/W)

MCPWM_GEN*n*_B_UTEB Configures action on PWM*n* B triggered by event TEB when timer increasing.

0: No change

1: Low

2: High

3: Toggle

(R/W)

MCPWM_GEN*n*_B_UT0 Configures action on PWM*n* B triggered by event_t0 when timer increasing.

0: No change

1: Low

2: High

3: Toggle

(R/W)

Continued on the next page...

Register 59.14. MCPWM_GEN*n*_B_REG(*n*: 0-2) (0x0054+0x38**n*)

Continued from the previous page...

MCPWM_GEN*n*_B_UT1 Configures action on PWM*n* B triggered by event_t1 when timer increasing.
0: No change
1: Low
2: High
3: Toggle
(R/W)

MCPWM_GEN*n*_B_DTEZ Configures action on PWM*n* B triggered by event TEZ when timer decreasing.
0: No change
1: Low
2: High
3: Toggle
(R/W)

MCPWM_GEN*n*_B_DTEP Configures action on PWM*n* B triggered by event TEP when timer decreasing.
0: No change
1: Low
2: High
3: Toggle
(R/W)

MCPWM_GEN*n*_B_DTEA Configures action on PWM*n* B triggered by event TEA when timer decreasing.
0: No change
1: Low
2: High
3: Toggle
(R/W)

MCPWM_GEN*n*_B_DTEB Configures action on PWM*n* B triggered by event TEB when timer decreasing.
0: No change
1: Low
2: High
3: Toggle
(R/W)

Continued on the next page...

Register 59.14. MCPWM_GEN*n*_B_REG(*n*: 0-2) (0x0054+0x38**n*)

Continued from the previous page...

MCPWM_GEN*n*_B_DT0 Configures action on PWM*n* B triggered by event_t0 when timer decreasing.

0: No change

1: Low

2: High

3: Toggle

(R/W)

MCPWM_GEN*n*_B_DT1 Configures action on PWM*n* B triggered by event_t1 when timer decreasing.

0: No change

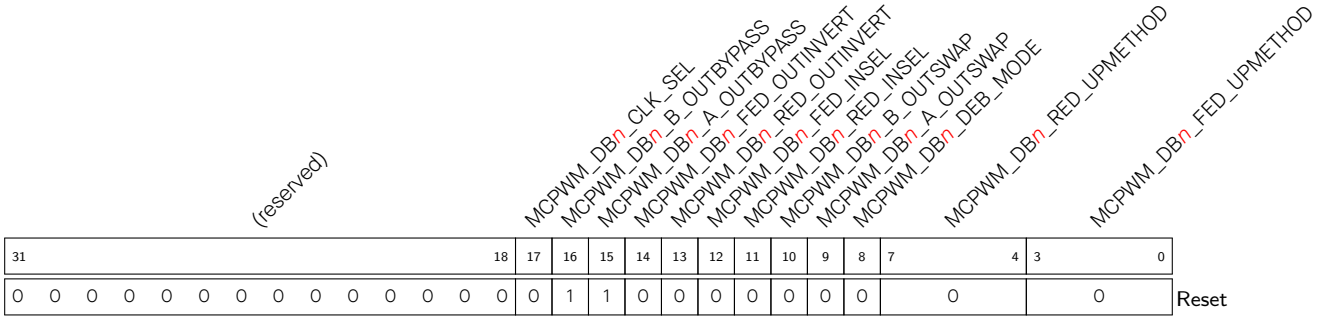
1: Low

2: High

3: Toggle

(R/W)

Register 59.15. MCPWM_DT*n*_CFG_REG(*n*: 0-2) (0x0058+0x38**n*)



MCPWM_DB*n*_FED_UPMETHOD Configures the update method for FED (Falling edge delay) active register.
0: Immediate
Bit0 is set to 1: TEZ
Bit1 is set to 1: TEP
Bit2 is set to 1: Sync
Bit3 is set to 1: Disable the update
(R/W)

MCPWM_DB*n*_RED_UPMETHOD Configures the update method for RED (Rising edge delay) active register.
0: Immediate
Bit0 is set to 1: TEZ
Bit1 is set to 1: TEP
Bit2 is set to 1: Sync
Bit3 is set to 1: Disable the update
(R/W)

MCPWM_DB*n*_DEB_MODE Configures the S8 switch in Table 59.3-5.
0: FED/RED take effect on different paths separately
1: FED/RED take effect on B path, A out is in bypass or dulpB mode
(R/W)

MCPWM_DB*n*_A_OUTSWAP Configures the S6 switch in Table 59.3-5. For typical configurations, please refer to Table 59.3-6. (R/W)

MCPWM_DB*n*_B_OUTSWAP Configures the S7 switch in Table 59.3-5. For typical configurations, please refer to Table 59.3-6. (R/W)

MCPWM_DB*n*_RED_INSEL Configures the S4 switch in Table 59.3-5. For typical configurations, please refer to Table 59.3-6. (R/W)

MCPWM_DB*n*_FED_INSEL Configures the S5 switch in Table 59.3-5. For typical configurations, please refer to Table 59.3-6. (R/W)

MCPWM_DB*n*_RED_OUTINVERT Configures the S2 switch in Table 59.3-5. For typical configurations, please refer to Table 59.3-6. (R/W)

Continued on the next page...

Register 59.15. MCPWM_DT*n*_CFG_REG(*n*: 0-2) (0x0058+0x38**n*)

Continued from the previous page...

MCPWM_DB*n*_FED_OUTINVERT Configures the S3 switch in Table 59.3-5. For typical configurations, please refer to Table 59.3-6. (R/W)

MCPWM_DB*n*_A_OUTBYPASS Configures the S1 switch in Table 59.3-5. For typical configurations, please refer to Table 59.3-6. (R/W)

MCPWM_DB*n*_B_OUTBYPASS Configures the S0 switch in Table 59.3-5. For typical configurations, please refer to Table 59.3-6. (R/W)

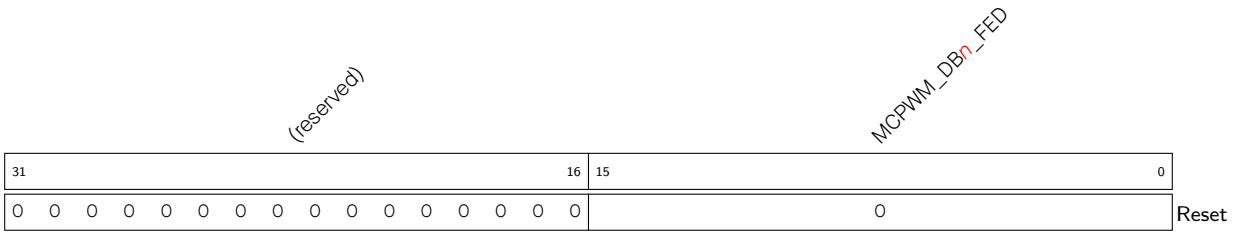
MCPWM_DB*n*_CLK_SEL Configures dead time generator *n* clock selection.

0: PWM_CLK

1: PT_CLK

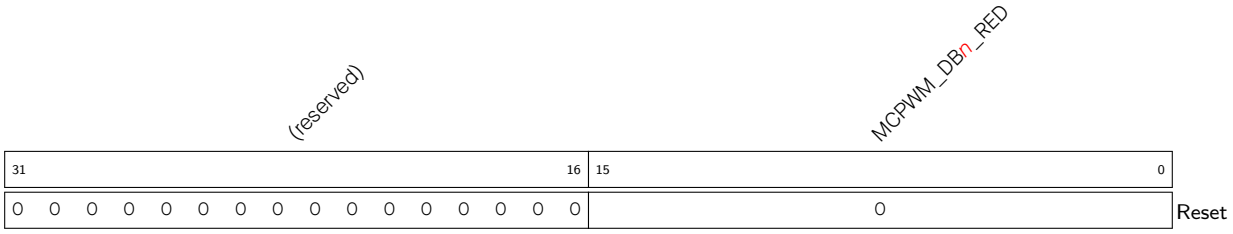
(R/W)

Register 59.16. MCPWM_DT*n*_FED_CFG_REG(*n*: 0-2) (0x005C+0x38**n*)



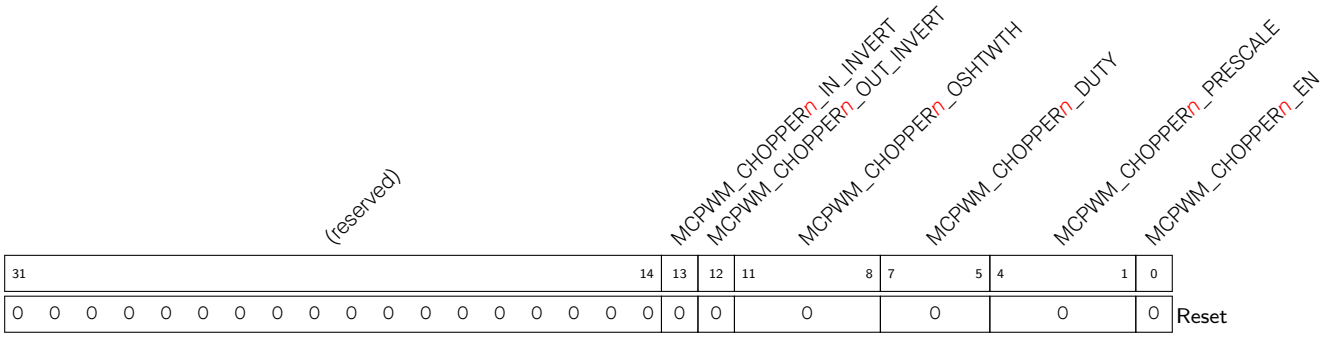
MCPWM_DB*n*_FED Configures the shadow register for FED. (R/W)

Register 59.17. MCPWM_DT*n*_RED_CFG_REG(*n*: 0-2) (0x0060+0x38**n*)



MCPWM_DB*n*_RED Configures the shadow register for RED. (R/W)

Register 59.18. MCPWM_CARRIER*n*_CFG_REG(*n*: 0-2) (0x0064+0x38n*)**



MCPWM_CHOPPER*n*_EN Configures whether to enable carrier*n*.

0: Bypassed

1: Enabled

(R/W)

MCPWM_CHOPPER*n*_PRESCALE Configures the prescale value of PWM carrier*n* clock (PC_clk), so that period of PC_clk = period of PWM_clk * (PWM_CARRIER*n*_PRESCALE + 1). (R/W)

MCPWM_CHOPPER*n*_DUTY Configures carrier duty. Duty = PWM_CARRIER*n*_DUTY / 8. (R/W)

MCPWM_CHOPPER*n*_OSHTWTH Configures width of the first pulse. Measurement unit: Periods of the carrier. (R/W)

MCPWM_CHOPPER*n*_OUT_INVERT Configures whether to invert the output of PWM*n* A and PWM*n* B for this submodule.

0: Normal

1: Invert

(R/W)

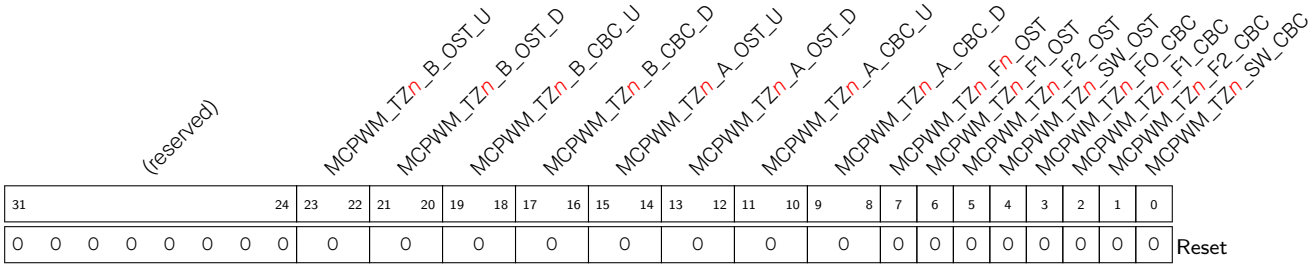
MCPWM_CHOPPER*n*_IN_INVERT Configures whether to invert the input of PWM*n* A and PWM*n* B for this submodule.

0: Normal

1: Invert

(R/W)

Register 59.19. MCPWM_FH*n*_CFG0_REG(*n*: 0-2) (0x0068+0x38**n*)



MCPWM_TZ*n*_SW_CBC Configures whether to enable software force cycle-by-cycle mode action.

0: Disable

1: Enable

(R/W)

MCPWM_TZ*n*_F2_CBC Configures whether event_f2 will trigger cycle-by-cycle mode action.

0: Disable

1: Enable

(R/W)

MCPWM_TZ*n*_F1_CBC Configures whether event_f1 will trigger cycle-by-cycle mode action.

0: Disable

1: Enable

(R/W)

MCPWM_TZ*n*_F0_CBC Configures whether event_f0 will trigger cycle-by-cycle mode action.

0: Disable

1: Enable

(R/W)

MCPWM_TZ*n*_SW_OST Configures whether to enable software force one-shot mode action.

0: Disable

1: Enable

(R/W)

MCPWM_TZ*n*_F2_OST Configures whether event_f2 will trigger one-shot mode action.

0: Disable

1: Enable

(R/W)

MCPWM_TZ*n*_F1_OST Configures whether event_f1 will trigger one-shot mode action.

0: Disable

1: Enable

(R/W)

MCPWM_TZ*n*_F*n*_OST Configures whether event_f0 will trigger one-shot mode action.

0: Disable

1: Enable

(R/W)

Continued on the next page...

Register 59.19. MCPWM_FH*n*_CFG0_REG(*n*: 0-2) (0x0068+0x38**n*)

Continued from the previous page...

MCPWM_TZ*n*_A_CBC_D Configures cycle-by-cycle mode action on PWM*n* A when fault event occurs and timer is decreasing.
0: Do nothing
1: Force low
2: Force high
3: Toggle
(R/W)

MCPWM_TZ*n*_A_CBC_U Configures cycle-by-cycle mode action on PWM*n* A when fault event occurs and timer is increasing.
0: Do nothing
1: Force low
2: Force high
3: Toggle
(R/W)

MCPWM_TZ*n*_A_OST_D Configures one-shot mode action on PWM*n* A when fault event occurs and timer is decreasing.
0: Do nothing
1: Force low
2: Force high
3: Toggle
(R/W)

MCPWM_TZ*n*_A_OST_U Configures one-shot mode action on PWM*n* A when fault event occurs and timer is increasing.
0: Do nothing
1: Force low
2: Force high
3: Toggle
(R/W)

MCPWM_TZ*n*_B_CBC_D Configures cycle-by-cycle mode action on PWM*n* B when fault event occurs and timer is decreasing.
0: Do nothing
1: Force low
2: Force high
3: Toggle
(R/W)

Continued on the next page...

Register 59.19. MCPWM_FH*n*_CFG0_REG(*n*: 0-2) (0x0068+0x38**n*)

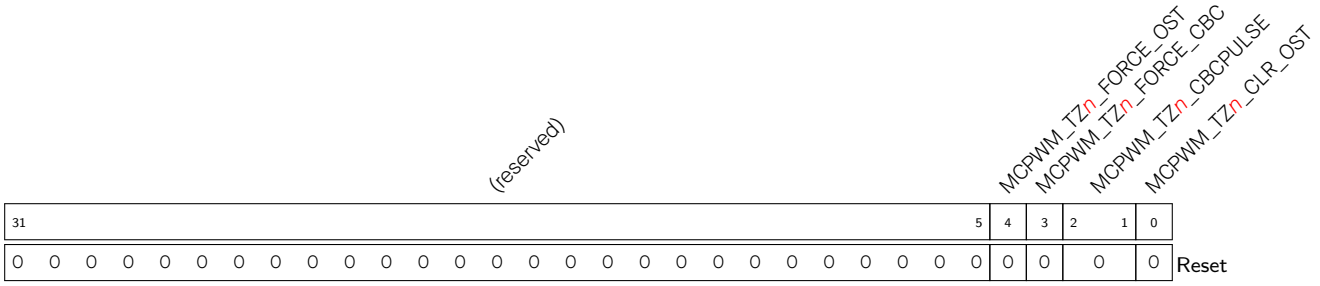
Continued from the previous page...

MCPWM_TZ*n*_B_CBC_U Configures cycle-by-cycle mode action on PWM*n* B when fault event occurs and timer is increasing.
0: Do nothing
1: Force low
2: Force high
3: Toggle
(R/W)

MCPWM_TZ*n*_B_OST_D Configures one-shot mode action on PWM*n* B when fault event occurs and timer is decreasing.
0: Do nothing
1: Force low
2: Force high
3: Toggle
(R/W)

MCPWM_TZ*n*_B_OST_U Configures one-shot mode action on PWM*n* B when fault event occurs and timer is increasing.
0: Do nothing
1: Force low
2: Force high
3: Toggle
(R/W)

Register 59.20. MCPWM_FH*n*_CFG1_REG(*n*: 0-2) (0x006C+0x38**n*)



MCPWM_TZ*n*_CLR_OST Configures whether to generate a one-shot mode action clear by software.

0: No effect

1: Triggers a clear for ongoing one-shot mode action by software

(R/W)

MCPWM_TZ*n*_CBCPULSE Configures the refresh moment selection of cycle-by-cycle mode action.

0: Select nothing, will not refresh

Bit0 is set to 1: TEZ

Bit1 is set to 1: TEP

(R/W)

MCPWM_TZ*n*_FORCE_CBC Configures whether to generate a software cycle-by-cycle mode action.

0: No effect

1: Triggers a cycle-by-cycle mode action by software

(R/W)

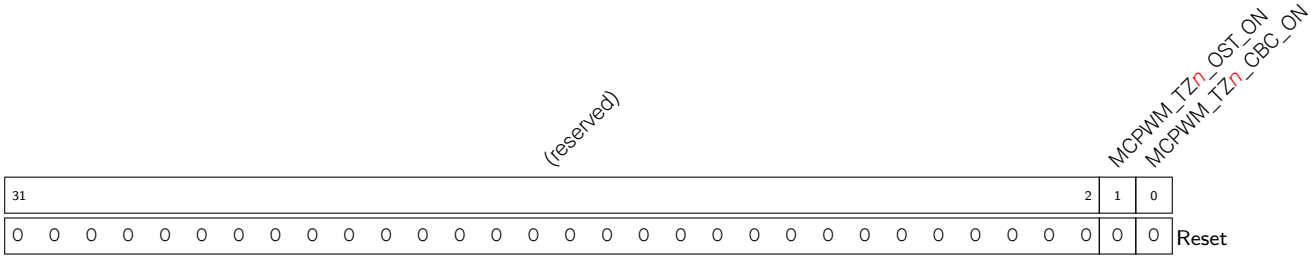
MCPWM_TZ*n*_FORCE_OST Configures whether to generate a software one-shot mode action.

0: No effect

1: Triggers a one-shot mode action by software

(R/W)

Register 59.21. MCPWM_FH*n*_STATUS_REG(*n*: 0-2) (0x0070+0x38**n*)



MCPWM_TZ*n*_CBC_ON Represents whether an cycle-by-cycle mode action is on going.

0: No action

1: Ongoing

(RO)

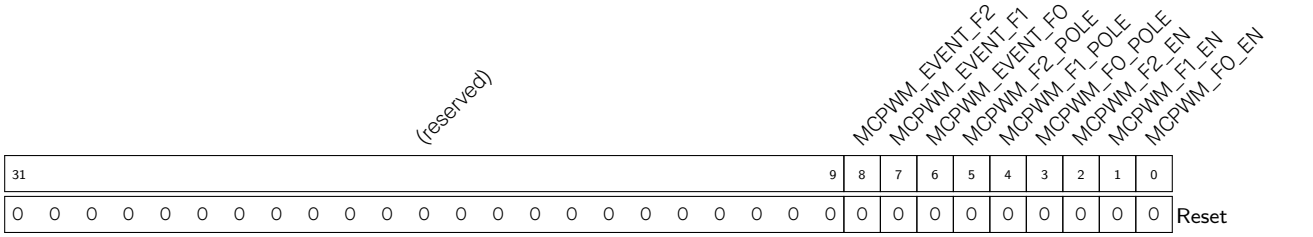
MCPWM_TZ*n*_OST_ON Represents whether a one-shot mode action is ongoing.

0: No action

1: Ongoing

(RO)

Register 59.22. MCPWM_FAULT_DETECT_REG (0x00E4)



MCPWM_F*n*_EN Configures whether to enable event_f*n* generation.

0: Disable

1: Enable

(R/W)

MCPWM_F*n*_POLE Configures event_f*n* trigger polarity on FAULT*n* source from GPIO matrix.

0: Level low

1: Level high

(R/W)

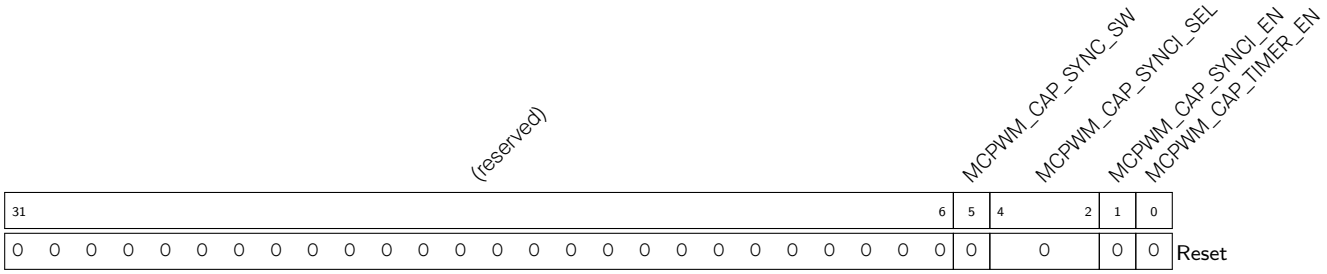
MCPWM_EVENT_F*n* Represents whether an event_f*n* is ongoing. This field is set and reset by hardware.

0: No action

1: Ongoing

(RO)

Register 59.23. MCPWM_CAP_TIMER_CFG_REG (0x00E8)



MCPWM_CAP_TIMER_EN Configures whether to enable capture timer increment incrementing under APB_CLK.

0: Disable

1: Enable

(R/W)

MCPWM_CAP_SYNCI_EN Configures whether to enable capture timer sync.

0: Disable

1: Enable

(R/W)

MCPWM_CAP_SYNCI_SEL Configures the selection of capture module sync input.

0: None

1: Timer0 sync_out

2: Timer1 sync_out

3: Timer2 sync_out

4: SYNC0 from GPIO matrix

5: SYNC1 from GPIO matrix

6: SYNC2 from GPIO matrix

7: None

(R/W)

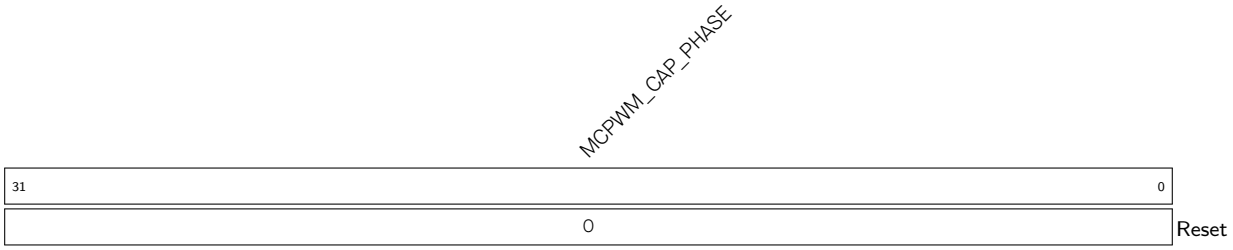
MCPWM_CAP_SYNC_SW When MCPWM_CAP_SYNCI_EN is set to 1, configures whether to trigger a capture timer sync so that capture timer is loaded with value in phase register.

0: No effect

1: Trigger a capture timer sync

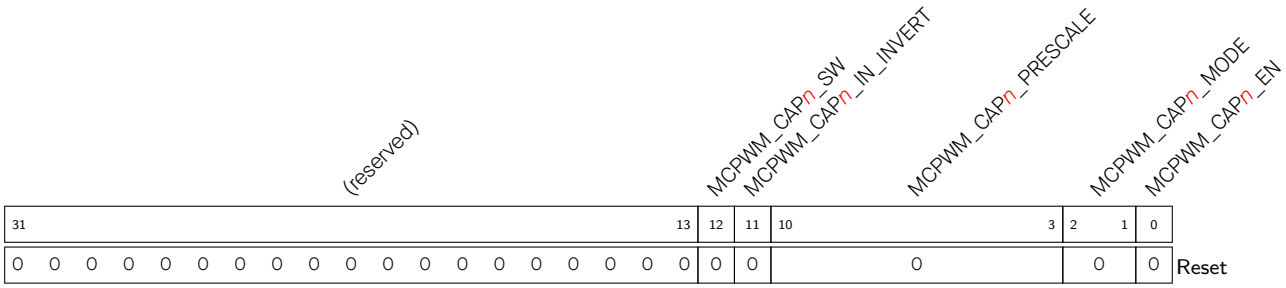
(WT)

Register 59.24. MCPWM_CAP_TIMER_PHASE_REG (0x00EC)



MCPWM_CAP_PHASE Configures phase value for capture timer sync operation. (R/W)

Register 59.25. MCPWM_CAP_CH*n*_CFG_REG (*n*: 0-2) (0x00F0+0x4**n*)



MCPWM_CAP*n*_EN Configures whether to enable capture on channel *n*.

0: Disable

1: Enable

(R/W)

MCPWM_CAP*n*_MODE Configures the edge of capture on channel 0 after prescaling.

When bit0 is set to 1: enable capture on the falling edge.

When bit1 is set to 1: enable capture on the rising edge.

(R/W)

MCPWM_CAP*n*_PRESCALE Configures the prescale value on the rising edge of CAP0. Prescale value = PWM_CAP0_PRESCALE + 1. (R/W)

MCPWM_CAP*n*_IN_INVERT Configures whether to invert CAP*n* from GPIO matrix before prescale.

0: Normal

1: Invert

(R/W)

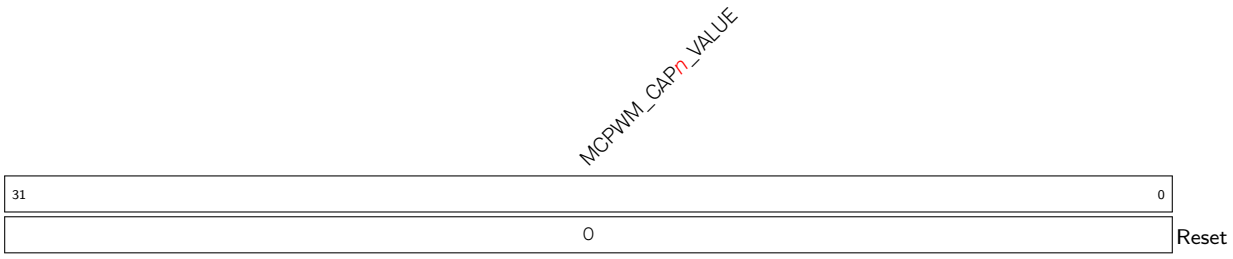
MCPWM_CAP*n*_SW Configures whether to trigger a software-forced capture on channel 0.

0: Not trigger

1: Trigger

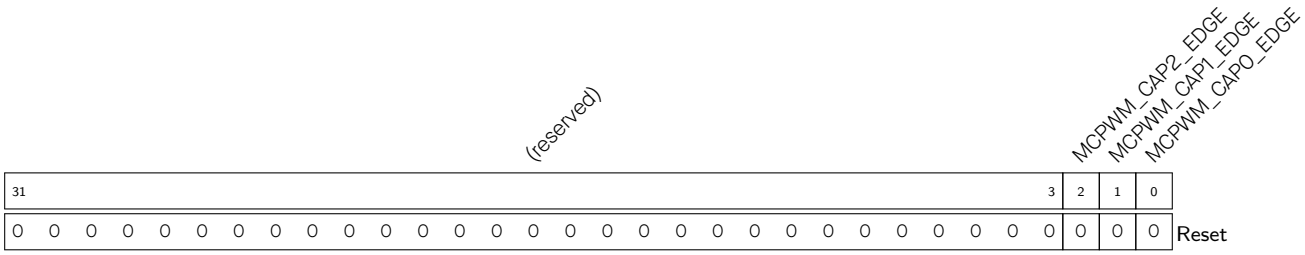
(WT)

Register 59.26. MCPWM_CAP_CH*n*_REG (*n*: 0-2) (0x00FC+0x4**n*)



MCPWM_CAP*n*_VALUE Represents value of last capture on CAP*n*. (RO)

Register 59.27. MCPWM_CAP_STATUS_REG (0x0108)



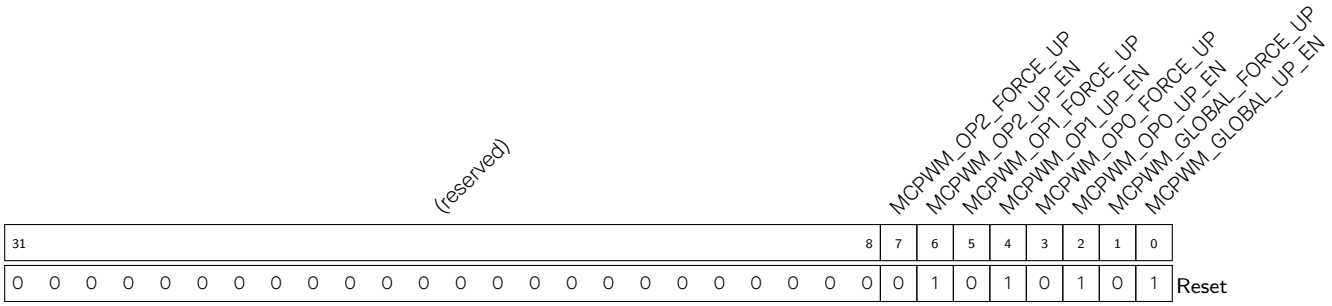
MCPWM_CAP*n*_EDGE Represents the edge of the last capture trigger on channel *n*.

0: Rising edge

1: Falling edge

(RO)

Register 59.28. MCPWM_UPDATE_CFG_REG (0x010C)



MCPWM_GLOBAL_UP_EN Configures whether to globally update all active registers.

0: No effect

1: Update all active registers globally

(R/W)

MCPWM_GLOBAL_FORCE_UP Configures whether to trigger a forced update of all active registers globally.

0: No effect

1: Trigger a forced update

(R/W)

MCPWM_OP0_UP_EN Configures whether to update active registers in PWM operator 0 when MCPWM_GLOBAL_UP_EN is set to 1.

0: No effect

1: Update active registers in PWM operator 0

(R/W)

MCPWM_OP0_FORCE_UP Configures whether to trigger a forced update of active registers in PWM operator 0.

0: No effect

1: Trigger a forced update

(R/W)

MCPWM_OP1_UP_EN Configures whether to update active registers in PWM operator 1 when MCPWM_GLOBAL_UP_EN is set to 1.

0: No effect

1: Update active registers in PWM operator 1

(R/W)

Continued on the next page...

Register 59.28. MCPWM_UPDATE_CFG_REG (0x010C)

Continued from the previous page...

MCPWM_OP1_FORCE_UP Configures whether to trigger a forced update of active registers in PWM operator 1.

0: No effect

1: Trigger a forced update

(R/W)

MCPWM_OP2_UP_EN Configures whether to update active registers in PWM operator 2 when MCPWM_GLOBAL_UP_EN is set to 1.

0: No effect

1: Update active registers in PWM operator 2

(R/W)

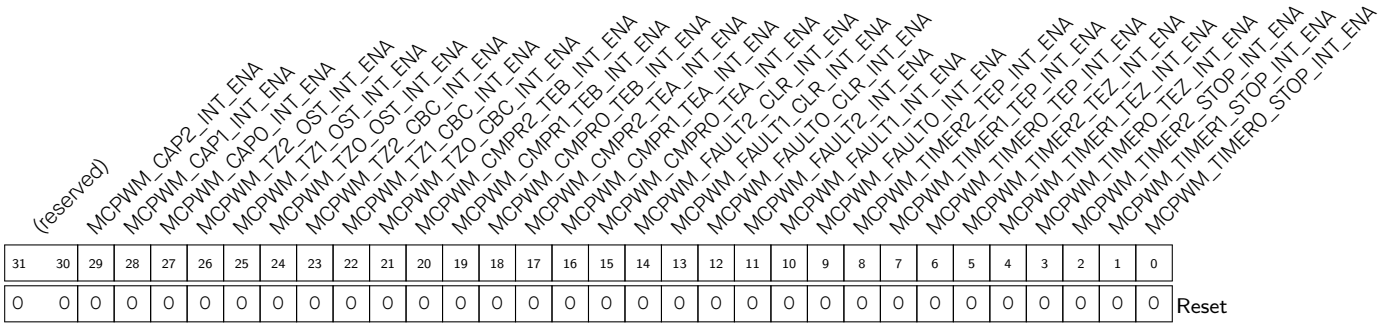
MCPWM_OP2_FORCE_UP Configures whether to trigger a forced update of active registers in PWM operator 2.

0: No effect

1: Trigger a forced update

(R/W)

Register 59.29. MCPWM_INT_ENA_REG (0x0110)



MCPWM_TIMER0_STOP_INT_ENA Write 1 to enable MCPWM_TIMER0_STOP_INT. (R/W)

MCPWM_TIMER1_STOP_INT_ENA Write 1 to enable MCPWM_TIMER1_STOP_INT. (R/W)

MCPWM_TIMER2_STOP_INT_ENA Write 1 to enable MCPWM_TIMER2_STOP_INT. (R/W)

MCPWM_TIMER0_TEZ_INT_ENA Write 1 to enable MCPWM_TIMER0_TEZ_INT. (R/W)

MCPWM_TIMER1_TEZ_INT_ENA Write 1 to enable MCPWM_TIMER1_TEZ_INT. (R/W)

MCPWM_TIMER2_TEZ_INT_ENA Write 1 to enable MCPWM_TIMER2_TEZ_INT. (R/W)

MCPWM_TIMER0_TEP_INT_ENA Write 1 to enable MCPWM_TIMER0_TEP_INT. (R/W)

MCPWM_TIMER1_TEP_INT_ENA Write 1 to enable MCPWM_TIMER1_TEP_INT. (R/W)

MCPWM_TIMER2_TEP_INT_ENA Write 1 to enable MCPWM_TIMER2_TEP_INT. (R/W)

MCPWM_FAULT0_INT_ENA Write 1 to enable MCPWM_FAULT0_INT. (R/W)

MCPWM_FAULT1_INT_ENA Write 1 to enable MCPWM_FAULT1_INT. (R/W)

MCPWM_FAULT2_INT_ENA Write 1 to enable MCPWM_FAULT2_INT. (R/W)

MCPWM_FAULT0_CLR_INT_ENA Write 1 to enable MCPWM_FAULT0_CLR_INT. (R/W)

MCPWM_FAULT1_CLR_INT_ENA Write 1 to enable MCPWM_FAULT1_CLR_INT. (R/W)

MCPWM_FAULT2_CLR_INT_ENA Write 1 to enable MCPWM_FAULT2_CLR_INT. (R/W)

MCPWM_CMPR0_TEA_INT_ENA Write 1 to enable MCPWM_CMPR0_TEA_INT. (R/W)

MCPWM_CMPR1_TEA_INT_ENA Write 1 to enable MCPWM_CMPR1_TEA_INT. (R/W)

MCPWM_CMPR2_TEA_INT_ENA Write 1 to enable MCPWM_CMPR2_TEA_INT. (R/W)

MCPWM_CMPR0_TEB_INT_ENA Write 1 to enable MCPWM_CMPR0_TEB_INT. (R/W)

MCPWM_CMPR1_TEB_INT_ENA Write 1 to enable MCPWM_CMPR1_TEB_INT. (R/W)

MCPWM_CMPR2_TEB_INT_ENA Write 1 to enable MCPWM_CMPR2_TEB_INT. (R/W)

MCPWM_TZ0_CBC_INT_ENA Write 1 to enable MCPWM_TZ0_CBC_INT. (R/W)

Continued on the next page...

Register 59.29. MCPWM_INT_ENA_REG (0x0110)

Continued from the previous page...

MCPWM_TZ1_CBC_INT_ENA Write 1 to enable MCPWM_TZ1_CBC_INT. (R/W)

MCPWM_TZ2_CBC_INT_ENA Write 1 to enable MCPWM_TZ2_CBC_INT. (R/W)

MCPWM_TZ0_OST_INT_ENA Write 1 to enable MCPWM_TZ0_OST_INT. (R/W)

MCPWM_TZ1_OST_INT_ENA Write 1 to enable MCPWM_TZ1_OST_INT. (R/W)

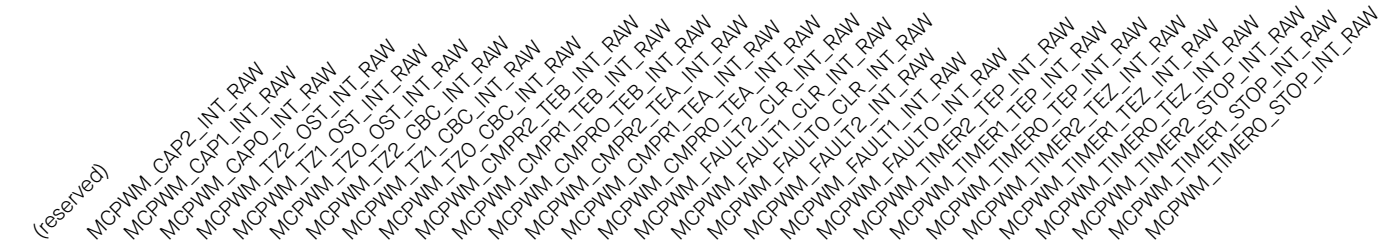
MCPWM_TZ2_OST_INT_ENA Write 1 to enable MCPWM_TZ2_OST_INT. (R/W)

MCPWM_CAP0_INT_ENA Write 1 to enable MCPWM_CAP0_INT. (R/W)

MCPWM_CAP1_INT_ENA Write 1 to enable MCPWM_CAP1_INT. (R/W)

MCPWM_CAP2_INT_ENA Write 1 to enable MCPWM_CAP2_INT. (R/W)

Register 59.30. MCPWM_INT_RAW_REG (0x0114)



31 30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
0 0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Reset

MCPWM_TIMER0_STOP_INT_RAW Represents the raw status of MCPWM_TIMER0_STOP_INT. (R/WTC/SS)

MCPWM_TIMER1_STOP_INT_RAW Represents the raw status of MCPWM_TIMER1_STOP_INT. (R/WTC/SS)

MCPWM_TIMER2_STOP_INT_RAW Represents the raw status of MCPWM_TIMER2_STOP_INT. (R/WTC/SS)

MCPWM_TIMER0_TEZ_INT_RAW Represents the raw status of MCPWM_TIMER0_TEZ_INT. (R/WTC/SS)

MCPWM_TIMER1_TEZ_INT_RAW Represents the raw status of MCPWM_TIMER1_TEZ_INT. (R/WTC/SS)

MCPWM_TIMER2_TEZ_INT_RAW Represents the raw status of MCPWM_TIMER2_TEZ_INT. (R/WTC/SS)

MCPWM_TIMER0_TEP_INT_RAW Represents the raw status of MCPWM_TIMER0_TEP_INT. (R/WTC/SS)

MCPWM_TIMER1_TEP_INT_RAW Represents the raw status of MCPWM_TIMER1_TEP_INT. (R/WTC/SS)

MCPWM_TIMER2_TEP_INT_RAW Represents the raw status of MCPWM_TIMER2_TEP_INT. (R/WTC/SS)

MCPWM_FAULT0_INT_RAW Represents the raw status of MCPWM_FAULT0_INT. (R/WTC/SS)

MCPWM_FAULT1_INT_RAW Represents the raw status of MCPWM_FAULT1_INT. (R/WTC/SS)

MCPWM_FAULT2_INT_RAW Represents the raw status of MCPWM_FAULT2_INT. (R/WTC/SS)

MCPWM_FAULT0_CLR_INT_RAW Represents the raw status of MCPWM_FAULT0_CLR_INT. (R/WTC/SS)

MCPWM_FAULT1_CLR_INT_RAW Represents the raw status of MCPWM_FAULT1_CLR_INT. (R/WTC/SS)

Continued on the next page...

Register 59.30. MCPWM_INT_RAW_REG (0x0114)

Continued from the previous page...

MCPWM_FAULT2_CLR_INT_RAW Represents the raw status of MCPWM_FAULT2_CLR_INT. (R/WTC/SS)

MCPWM_CMPR0_TEA_INT_RAW Represents the raw status of MCPWM_CMPR0_TEA_INT. (R/WTC/SS)

MCPWM_CMPR1_TEA_INT_RAW Represents the raw status of MCPWM_CMPR1_TEA_INT. (R/WTC/SS)

MCPWM_CMPR2_TEA_INT_RAW Represents the raw status of MCPWM_CMPR2_TEA_INT. (R/WTC/SS)

MCPWM_CMPR0_TEB_INT_RAW Represents the raw status of MCPWM_CMPR0_TEB_INT. (R/WTC/SS)

MCPWM_CMPR1_TEB_INT_RAW Represents the raw status of MCPWM_CMPR1_TEB_INT. (R/WTC/SS)

MCPWM_CMPR2_TEB_INT_RAW Represents the raw status of MCPWM_CMPR2_TEB_INT. (R/WTC/SS)

MCPWM_TZ0_CBC_INT_RAW Represents the raw status of MCPWM_TZ0_CBC_INT. (R/WTC/SS)

MCPWM_TZ1_CBC_INT_RAW Represents the raw status of MCPWM_TZ1_CBC_INT. (R/WTC/SS)

MCPWM_TZ2_CBC_INT_RAW Represents the raw status of MCPWM_TZ2_CBC_INT. (R/WTC/SS)

MCPWM_TZ0_OST_INT_RAW Represents the raw status of MCPWM_TZ0_OST_INT. (R/WTC/SS)

MCPWM_TZ1_OST_INT_RAW Represents the raw status of MCPWM_TZ1_OST_INT. (R/WTC/SS)

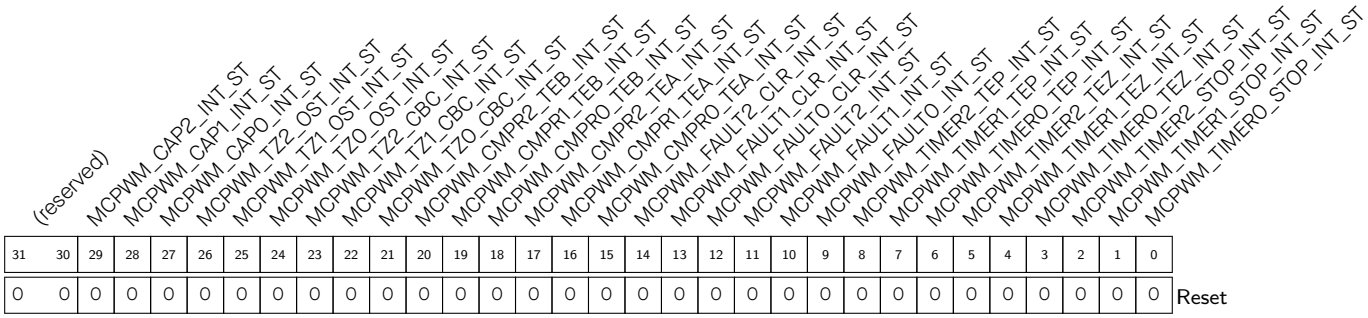
MCPWM_TZ2_OST_INT_RAW Represents the raw status of MCPWM_TZ2_OST_INT. (R/WTC/SS)

MCPWM_CAP0_INT_RAW Represents the raw status of MCPWM_CAP0_INT. (R/WTC/SS)

MCPWM_CAP1_INT_RAW Represents the raw status of MCPWM_CAP1_INT. (R/WTC/SS)

MCPWM_CAP2_INT_RAW Represents the raw status of MCPWM_CAP2_INT. (R/WTC/SS)

Register 59.31. MCPWM_INT_ST_REG (0x0118)



MCPWM_TIMER0_STOP_INT_ST Represents the masked status of MCPWM_TIMER0_STOP_INT. (RO)

MCPWM_TIMER1_STOP_INT_ST Represents the masked status of MCPWM_TIMER1_STOP_INT. (RO)

MCPWM_TIMER2_STOP_INT_ST Represents the masked status of MCPWM_TIMER2_STOP_INT. (RO)

MCPWM_TIMER0_TEZ_INT_ST Represents the masked status of MCPWM_TIMER0_TEZ_INT. (RO)

MCPWM_TIMER1_TEZ_INT_ST Represents the masked status of MCPWM_TIMER1_TEZ_INT. (RO)

MCPWM_TIMER2_TEZ_INT_ST Represents the masked status of MCPWM_TIMER2_TEZ_INT. (RO)

MCPWM_TIMER0_TEP_INT_ST Represents the masked status of MCPWM_TIMER0_TEP_INT. (RO)

MCPWM_TIMER1_TEP_INT_ST Represents the masked status of MCPWM_TIMER1_TEP_INT. (RO)

MCPWM_TIMER2_TEP_INT_ST Represents the masked status of MCPWM_TIMER2_TEP_INT. (RO)

MCPWM_FAULT0_INT_ST Represents the masked status of MCPWM_FAULT0_INT. (RO)

MCPWM_FAULT1_INT_ST Represents the masked status of MCPWM_FAULT1_INT. (RO)

MCPWM_FAULT2_INT_ST Represents the masked status of MCPWM_FAULT2_INT. (RO)

MCPWM_FAULT0_CLR_INT_ST Represents the masked status of MCPWM_FAULT0_CLR_INT. (RO)

MCPWM_FAULT1_CLR_INT_ST Represents the masked status of MCPWM_FAULT1_CLR_INT. (RO)

Continued on the next page...

Register 59.31. MCPWM_INT_ST_REG (0x0118)

Continued from the previous page...

MCPWM_FAULT2_CLR_INT_ST Represents the masked status of MCPWM_FAULT2_CLR_INT. (RO)

MCPWM_CMPR0_TEA_INT_ST Represents the masked status of MCPWM_CMPR0_TEA_INT. (RO)

MCPWM_CMPR1_TEA_INT_ST Represents the masked status of MCPWM_CMPR1_TEA_INT. (RO)

MCPWM_CMPR2_TEA_INT_ST Represents the masked status of MCPWM_CMPR2_TEA_INT. (RO)

MCPWM_CMPR0_TEB_INT_ST Represents the masked status of MCPWM_CMPR0_TEB_INT. (RO)

MCPWM_CMPR1_TEB_INT_ST Represents the masked status of MCPWM_CMPR1_TEB_INT. (RO)

MCPWM_CMPR2_TEB_INT_ST Represents the masked status of MCPWM_CMPR2_TEB_INT. (RO)

MCPWM_TZ0_CBC_INT_ST Represents the masked status of MCPWM_TZ0_CBC_INT_ST. (RO)

MCPWM_TZ1_CBC_INT_ST Represents the masked status of MCPWM_TZ1_CBC_INT_ST. (RO)

MCPWM_TZ2_CBC_INT_ST Represents the masked status of MCPWM_TZ2_CBC_INT_ST. (RO)

MCPWM_TZ0_OST_INT_ST Represents the masked status of MCPWM_TZ0_OST_INT. (RO)

MCPWM_TZ1_OST_INT_ST Represents the masked status of MCPWM_TZ1_OST_INT. (RO)

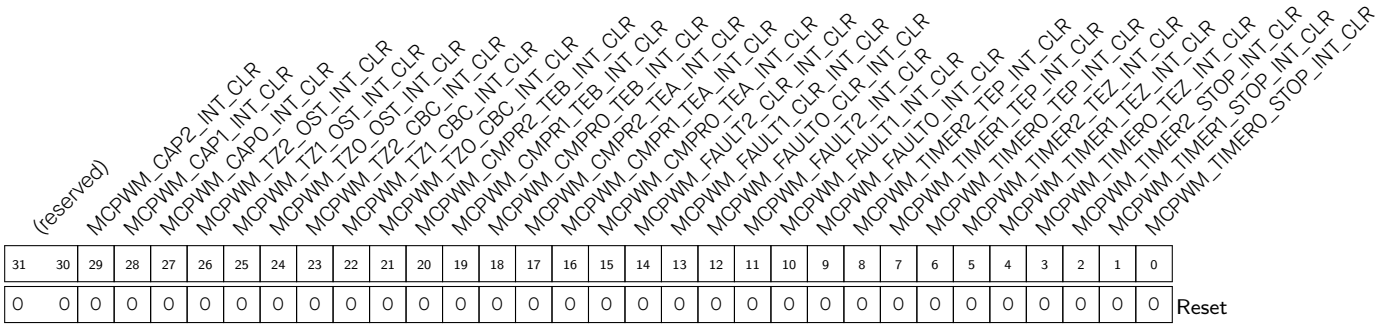
MCPWM_TZ2_OST_INT_ST Represents the masked status of MCPWM_TZ2_OST_INT. (RO)

MCPWM_CAP0_INT_ST Represents the masked status of MCPWM_CAP0_INT. (RO)

MCPWM_CAP1_INT_ST Represents the masked status of MCPWM_CAP1_INT. (RO)

MCPWM_CAP2_INT_ST Represents the masked status of MCPWM_CAP2_INT. (RO)

Register 59.32. MCPWM_INT_CLR_REG (0x011C)



MCPWM_TIMER0_STOP_INT_CLR Write 1 to clear MCPWM_TIMER0_STOP_INT. (WT)

MCPWM_TIMER1_STOP_INT_CLR Write 1 to clear MCPWM_TIMER1_STOP_INT. (WT)

MCPWM_TIMER2_STOP_INT_CLR Write 1 to clear MCPWM_TIMER2_STOP_INT. (WT)

MCPWM_TIMER0_TEZ_INT_CLR Write 1 to clear MCPWM_TIMER0_TEZ_INT. (WT)

MCPWM_TIMER1_TEZ_INT_CLR Write 1 to clear MCPWM_TIMER1_TEZ_INT. (WT)

MCPWM_TIMER2_TEZ_INT_CLR Write 1 to clear MCPWM_TIMER2_TEZ_INT. (WT)

MCPWM_TIMER0_TEP_INT_CLR Write 1 to clear MCPWM_TIMER0_TEP_INT. (WT)

MCPWM_TIMER1_TEP_INT_CLR Write 1 to clear MCPWM_TIMER1_TEP_INT. (WT)

MCPWM_TIMER2_TEP_INT_CLR Write 1 to clear MCPWM_TIMER2_TEP_INT. (WT)

MCPWM_FAULT0_INT_CLR Write 1 to clear MCPWM_FAULT0_INT. (WT)

MCPWM_FAULT1_INT_CLR Write 1 to clear MCPWM_FAULT1_INT. (WT)

MCPWM_FAULT2_INT_CLR Write 1 to clear MCPWM_FAULT2_INT. (WT)

MCPWM_FAULT0_CLR_INT_CLR Write 1 to clear MCPWM_FAULT0_CLR_INT. (WT)

MCPWM_FAULT1_CLR_INT_CLR Write 1 to clear MCPWM_FAULT1_CLR_INT. (WT)

MCPWM_FAULT2_CLR_INT_CLR Write 1 to clear MCPWM_FAULT2_CLR_INT. (WT)

MCPWM_CMPR0_TEA_INT_CLR Write 1 to clear MCPWM_CMPR0_TEA_INT. (WT)

MCPWM_CMPR1_TEA_INT_CLR Write 1 to clear MCPWM_CMPR1_TEA_INT. (WT)

MCPWM_CMPR2_TEA_INT_CLR Write 1 to clear MCPWM_CMPR2_TEA_INT. (WT)

MCPWM_CMPR0_TEB_INT_CLR Write 1 to clear MCPWM_CMPR0_TEB_INT. (WT)

MCPWM_CMPR1_TEB_INT_CLR Write 1 to clear MCPWM_CMPR1_TEB_INT. (WT)

MCPWM_CMPR2_TEB_INT_CLR Write 1 to clear MCPWM_CMPR2_TEB_INT. (WT)

MCPWM_TZ0_CBC_INT_CLR Write 1 to clear MCPWM_TZ0_CBC_INT. (WT)

Continued on the next page...

Register 59.32. MCPWM_INT_CLR_REG (0x011C)

Continued from the previous page...

MCPWM_TZ1_CBC_INT_CLR Write 1 to clear MCPWM_TZ1_CBC_INT. (WT)

MCPWM_TZ2_CBC_INT_CLR Write 1 to clear MCPWM_TZ2_CBC_INT. (WT)

MCPWM_TZ0_OST_INT_CLR Write 1 to clear MCPWM_TZ0_OST_INT. (WT)

MCPWM_TZ1_OST_INT_CLR Write 1 to clear MCPWM_TZ1_OST_INT. (WT)

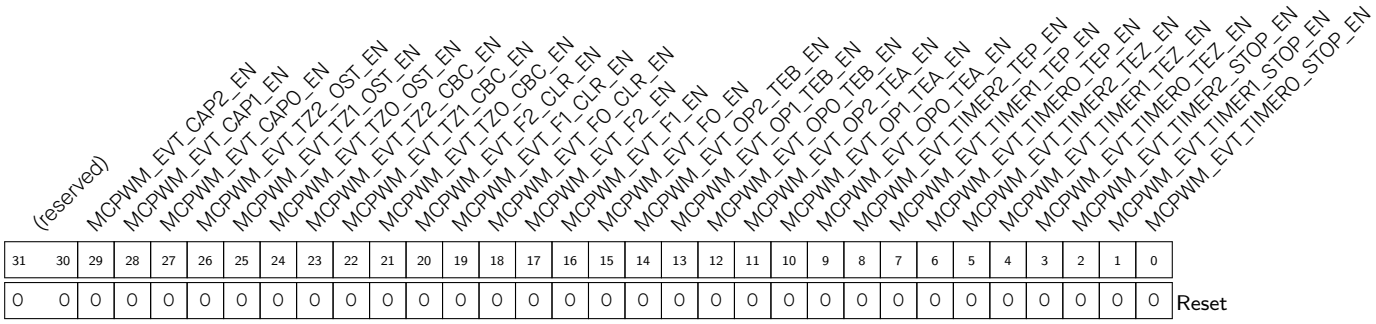
MCPWM_TZ2_OST_INT_CLR Write 1 to clear MCPWM_TZ2_OST_INT. (WT)

MCPWM_CAP0_INT_CLR Write 1 to clear MCPWM_CAP0_INT. (WT)

MCPWM_CAP1_INT_CLR Write 1 to clear MCPWM_CAP1_INT. (WT)

MCPWM_CAP2_INT_CLR Write 1 to clear MCPWM_CAP2_INT. (WT)

Register 59.33. MCPWM_EVT_EN_REG (0x0120)



MCPWM_EVT_TIMER0_STOP_EN Configures whether to enable timer0 stop event generation.

0: Disable

1: Enable

(R/W)

MCPWM_EVT_TIMER1_STOP_EN Configures whether to enable timer1 stop event generation.

0: Disable

1: Enable

(R/W)

MCPWM_EVT_TIMER2_STOP_EN Configures whether to enable timer2 stop event generation.

0: Disable

1: Enable

(R/W)

MCPWM_EVT_TIMER0_TEZ_EN Configures whether to enable timer0 equal zero event generation.

0: Disable

1: Enable

(R/W)

MCPWM_EVT_TIMER1_TEZ_EN Configures whether to enable timer1 equal zero event generation.

0: Disable

1: Enable

(R/W)

MCPWM_EVT_TIMER2_TEZ_EN Configures whether to enable timer2 equal zero event generation.

0: Disable

1: Enable

(R/W)

Continued on the next page...

Register 59.33. MCPWM_EVT_EN_REG (0x0120)

Continued from the previous page...

MCPWM_EVT_TIMER0_TEP_EN Configures whether to enable timer0 equal period event generation.
0: Disable
1: Enable
(R/W)

MCPWM_EVT_TIMER1_TEP_EN Configures whether to enable timer1 equal period event generation.
0: Disable
1: Enable
(R/W)

MCPWM_EVT_TIMER2_TEP_EN Configures whether to enable timer2 equal period event generation.
0: Disable
1: Enable
(R/W)

MCPWM_EVT_OP0_TEA_EN Configures whether to enable PWM generator0 timer equal A event generation.
0: Disable
1: Enable
(R/W)

MCPWM_EVT_OP1_TEA_EN Configures whether to enable PWM generator1 timer equal A event generation.
0: Disable
1: Enable
(R/W)

MCPWM_EVT_OP2_TEA_EN Configures whether to enable PWM generator2 timer equal A event generation.
0: Disable
1: Enable
(R/W)

MCPWM_EVT_OP0_TEB_EN Configures whether to enable PWM generator0 timer equal B event generation.
0: Disable
1: Enable
(R/W)

Continued on the next page...

Register 59.33. MCPWM_EVT_EN_REG (0x0120)

Continued from the previous page...

MCPWM_EVT_OP1_TEB_EN Configures whether to enable PWM generator1 timer equal B event generation.
0: Disable
1: Enable
(R/W)

MCPWM_EVT_OP2_TEB_EN Configures whether to enable PWM generator2 timer equal B event generation.
0: Disable
1: Enable
(R/W)

MCPWM_EVT_F0_EN Configures whether to enable FAULT0 event generation.
0: Disable
1: Enable
(R/W)

MCPWM_EVT_F1_EN Configures whether to enable FAULT1 event generation.
0: Disable
1: Enable
(R/W)

MCPWM_EVT_F2_EN Configures whether to enable FAULT2 event generation.
0: Disable
1: Enable
(R/W)

MCPWM_EVT_F0_CLR_EN Configures whether to enable FAULT0 clear event generation.
0: Disable
1: Enable
(R/W)

MCPWM_EVT_F1_CLR_EN Configures whether to enable FAULT1 clear event generation.
0: Disable
1: Enable
(R/W)

MCPWM_EVT_F2_CLR_EN Configures whether to enable FAULT2 clear event generation.
0: Disable
1: Enable
(R/W)

Continued on the next page...

Register 59.33. MCPWM_EVT_EN_REG (0x0120)

Continued from the previous page...

MCPWM_EVT_TZ0_CBC_EN Configures whether to enable cycle by cycle trip0 event generation.
0: Disable
1: Enable
(R/W)

MCPWM_EVT_TZ1_CBC_EN Configures whether to enable cycle by cycle trip1 event generation.
0: Disable
1: Enable
(R/W)

MCPWM_EVT_TZ2_CBC_EN Configures whether to enable cycle by cycle trip2 event generation.
0: Disable
1: Enable
(R/W)

MCPWM_EVT_TZ0_OST_EN Configures whether to enable one shot trip0 event generation.
0: Disable
1: Enable
(R/W)

MCPWM_EVT_TZ1_OST_EN Configures whether to enable one shot trip1 event generation.
0: Disable
1: Enable
(R/W)

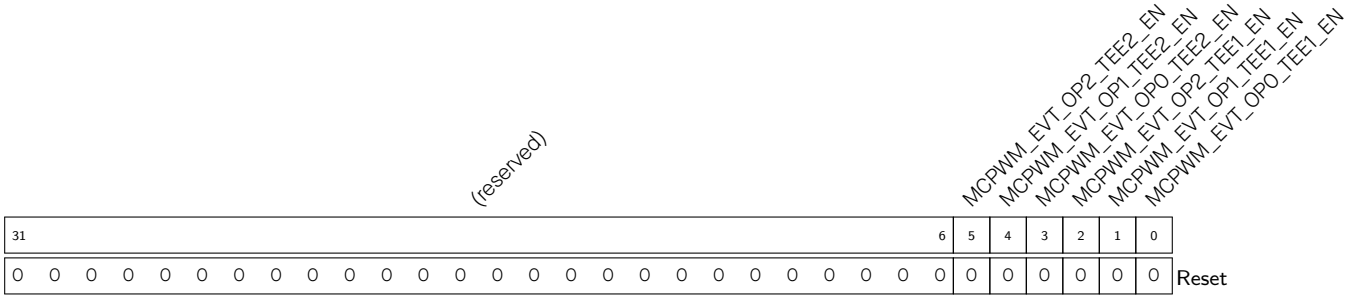
MCPWM_EVT_TZ2_OST_EN Configures whether to enable one shot trip2 event generation.
0: Disable
1: Enable
(R/W)

MCPWM_EVT_CAP0_EN Configures whether to enable capture0 event generation.
0: Disable
1: Enable
(R/W)

MCPWM_EVT_CAP1_EN Configures whether to enable capture1 event generation.
0: Disable
1: Enable
(R/W)

MCPWM_EVT_CAP2_EN Configures whether to enable capture2 event generation.
0: Disable
1: Enable
(R/W)

Register 59.34. MCPWM_EVT_EN2_REG (0x0128)



MCPWM_EVT_OP*n*_TEE1_EN Configures whether to generate the MCPWM_EVT_OP*n*_TEE1 event when the PWM generator*n* timer equals OP*n*_TSTMP_E1_REG.

0: Not generate

1: Generate

(R/W)

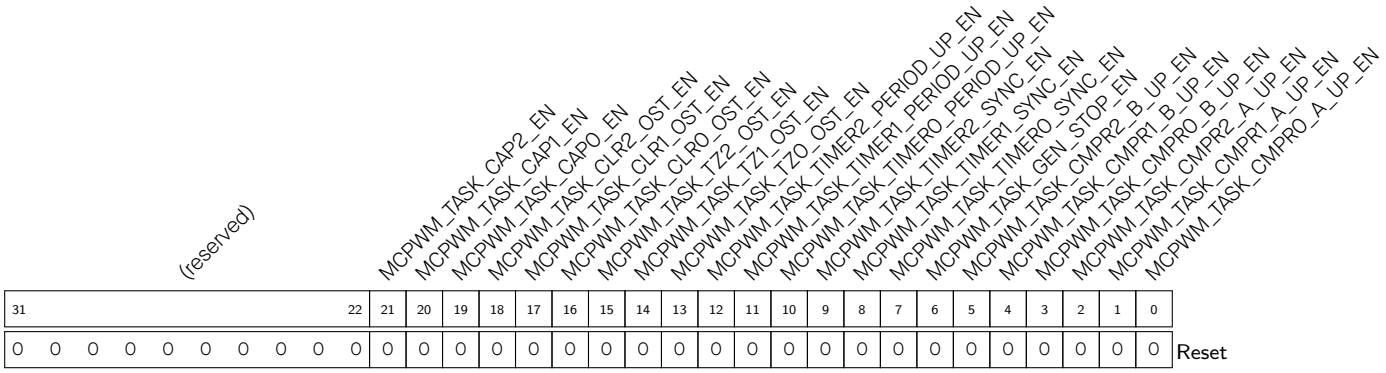
MCPWM_EVT_OP*n*_TEE2_EN Configures whether to generate the MCPWM_EVT_OP*n*_TEE2 event when the PWM generator*n* timer equals OP*n*_TSTMP_E2_REG.

0: Not generate

1: Generate

(R/W)

Register 59.35. MCPWM_TASK_EN_REG (0x0124)



MCPWM_TASK_CMPR0_A_UP_EN Configures whether to receive update task of PWM generator0 timer stamp A's shadow register.

0: No effect

1: Receive

(R/W)

MCPWM_TASK_CMPR1_A_UP_EN Configures whether to receive update task of PWM generator1 timer stamp A's shadow register.

0: No effect

1: Receive

(R/W)

MCPWM_TASK_CMPR2_A_UP_EN Configures whether to receive update task of PWM generator2 timer stamp A's shadow register.

0: No effect

1: Receive

(R/W)

MCPWM_TASK_CMPR0_B_UP_EN Configures whether to receive update task of PWM generator0 timer stamp B's shadow register.

0: No effect

1: Receive

(R/W)

MCPWM_TASK_CMPR1_B_UP_EN Configures whether to receive update task of PWM generator1 timer stamp B's shadow register.

0: No effect

1: Receive

(R/W)

MCPWM_TASK_CMPR2_B_UP_EN Configures whether to receive update task of PWM generator2 timer stamp B's shadow register.

0: No effect

1: Receive

(R/W)

Continued on the next page...

Register 59.35. MCPWM_TASK_EN_REG (0x0124)

Continued from the previous page...

MCPWM_TASK_GEN_STOP_EN Configures whether to receive all PWM generate stop task.

0: No effect

1: Receive

(R/W)

MCPWM_TASK_TIMER0_SYNC_EN Configures whether to receive timer0 sync task.

0: No effect

1: Receive

(R/W)

MCPWM_TASK_TIMER1_SYNC_EN Configures whether to receive timer1 sync task.

0: No effect

1: Receive

(R/W)

MCPWM_TASK_TIMER2_SYNC_EN Configures whether to receive timer2 sync task.

0: No effect

1: Receive

(R/W)

MCPWM_TASK_TIMER0_PERIOD_UP_EN Configures whether to receive timer0 period update task.

0: No effect

1: Receive

(R/W)

MCPWM_TASK_TIMER1_PERIOD_UP_EN Configures whether to receive timer1 period update task.

0: No effect

1: Receive

(R/W)

MCPWM_TASK_TIMER2_PERIOD_UP_EN Configures whether to receive timer2 period update task.

0: No effect

1: Receive

(R/W)

Continued on the next page...

Register 59.35. MCPWM_TASK_EN_REG (0x0124)

Continued from the previous page...

MCPWM_TASK_TZ0_OST_EN Configures whether to receive one shot trip0 task.

0: No effect

1: Receive

(R/W)

MCPWM_TASK_TZ1_OST_EN Configures whether to receive one shot trip1 task.

0: No effect

1: Receive

(R/W)

MCPWM_TASK_TZ2_OST_EN Configures whether to receive one shot trip2 task.

0: No effect

1: Receive

(R/W)

MCPWM_TASK_CLR0_OST_EN Configures whether to receive one shot trip0 clear task.

0: No effect

1: Receive

(R/W)

MCPWM_TASK_CLR1_OST_EN Configures whether to receive one shot trip1 clear task.

0: No effect

1: Receive

(R/W)

MCPWM_TASK_CLR2_OST_EN Configures whether to receive one shot trip2 clear task.

0: No effect

1: Receive

(R/W)

MCPWM_TASK_CAP0_EN Configures whether to receive capture0 task.

0: No effect

1: Receive

(R/W)

MCPWM_TASK_CAP1_EN Configures whether to receive capture1 task.

0: No effect

1: Receive

(R/W)

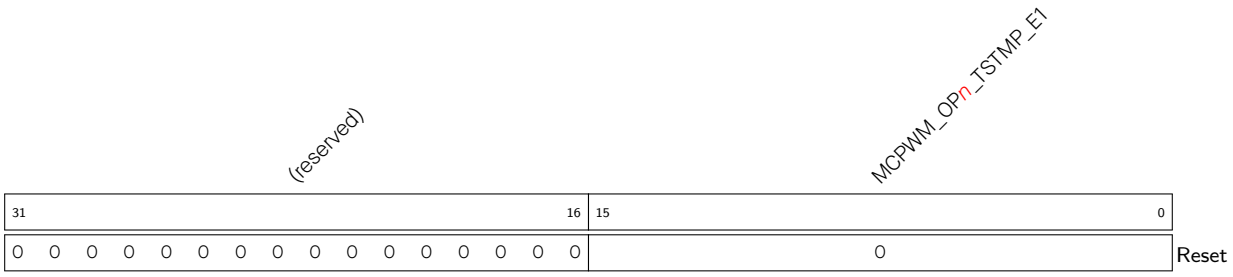
MCPWM_TASK_CAP2_EN Configures whether to receive capture2 task.

0: No effect

1: Receive

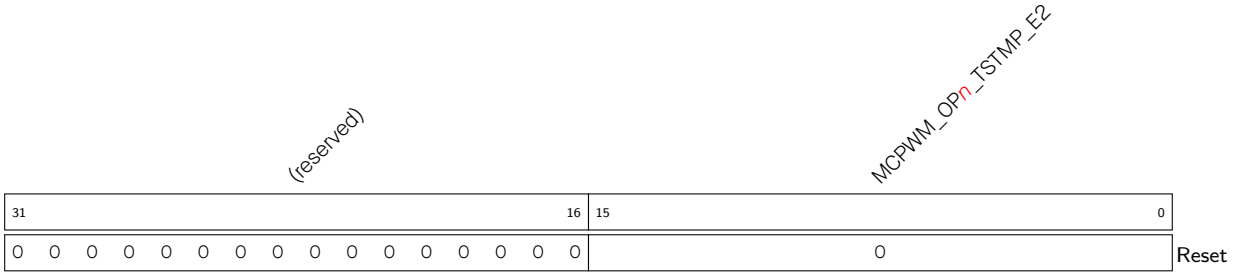
(R/W)

Register 59.36. MCPWM_OP*n*_TSTMP_E1_REG (*n*: 0-2) (0x012C+0x8**n*)



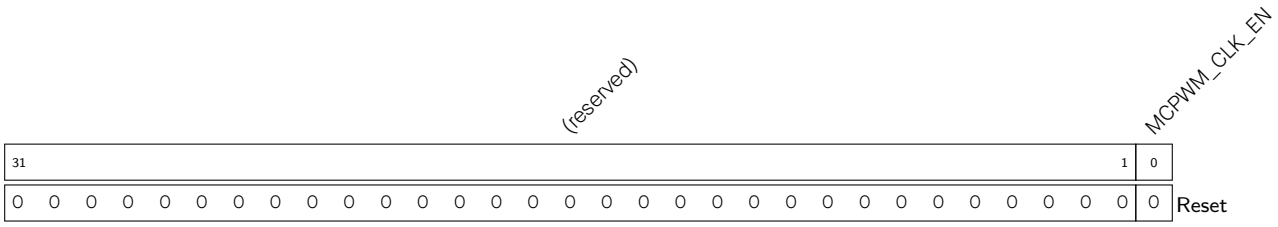
MCPWM_OP*n*_TSTMP_E1 Configures the time stamp E1 value of the generator*n*. (R/W)

Register 59.37. MCPWM_OP*n*_TSTMP_E2_REG(*n*: 0-2) (0x0130+0x8**n*)



MCPWM_OP*n*_TSTMP_E2 Configures the time stamp E2 value of the generator*n*. (R/W)

Register 59.38. MCPWM_CLK_REG (0x0144)



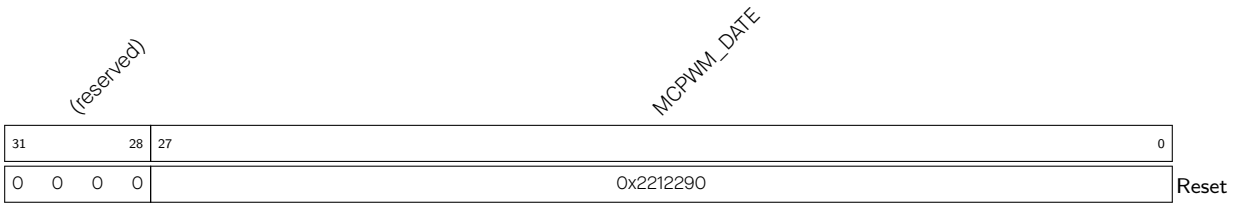
MCPWM_CLK_EN Configures whether to force open register clock gate.

0: Open the clock gate only when application writes registers

1: Force open the clock gate for register

(R/W)

Register 59.39. MCPWM_VERSION_REG (0x0148)



MCPWM_DATE Version control register. (R/W)

Chapter 60

Remote Control Peripheral (RMT)

60.1 Overview

The Remote Control (RMT) module is designed to transmit and receive infrared remote control signals. A variety of remote control protocols can be encoded/decoded via software based on the RMT module. The RMT module converts pulse codes stored in the module's built-in RAM into output signals, or converts input signals into pulse codes and stores them in RAM. In addition, the RMT module optionally modulates its output signals with a carrier wave, or optionally demodulates and filters its input signals.

The RMT module has eight channels, numbered from zero to seven. Each channel is able to independently transmit or receive signals.

- Channel 0 ~ 3 (TX channel) are dedicated to transmitting signals;
- Channel 4 ~ 7 (RX channel) are dedicated to receiving signals.

Each TX/RX channel has the same functionality controlled by a dedicated set of registers and is able to independently transmit or receive data. Channel 3 and channel 7 support GDMA access, so the two channels also have a set of GDMA-related control and status registers. TX channels are indicated by *n* which is used as a placeholder for the channel number, and by *m* for RX channels.

60.2 Features

The RMT module has the following features:

- Eight channels:
  - TX channels 0 ~ 3
  - RX channels 4 ~ 7
  - Eight channels share a 384 x 32-bit RAM
- The transmitter supports:
  - Normal TX mode
  - Wrap TX mode
  - Continuous TX mode
  - Modulation on TX pulses
  - Multiple channels transmitting data simultaneously (programmable)
  - GDMA access supported by TX channel 3

- The receiver supports:
  - Normal RX mode
  - Wrap RX mode
  - RX filtering
  - Demodulation on RX pulses
  - GDMA access supported by RX channel 7

60.3 Functional Description

60.3.1 Architecture

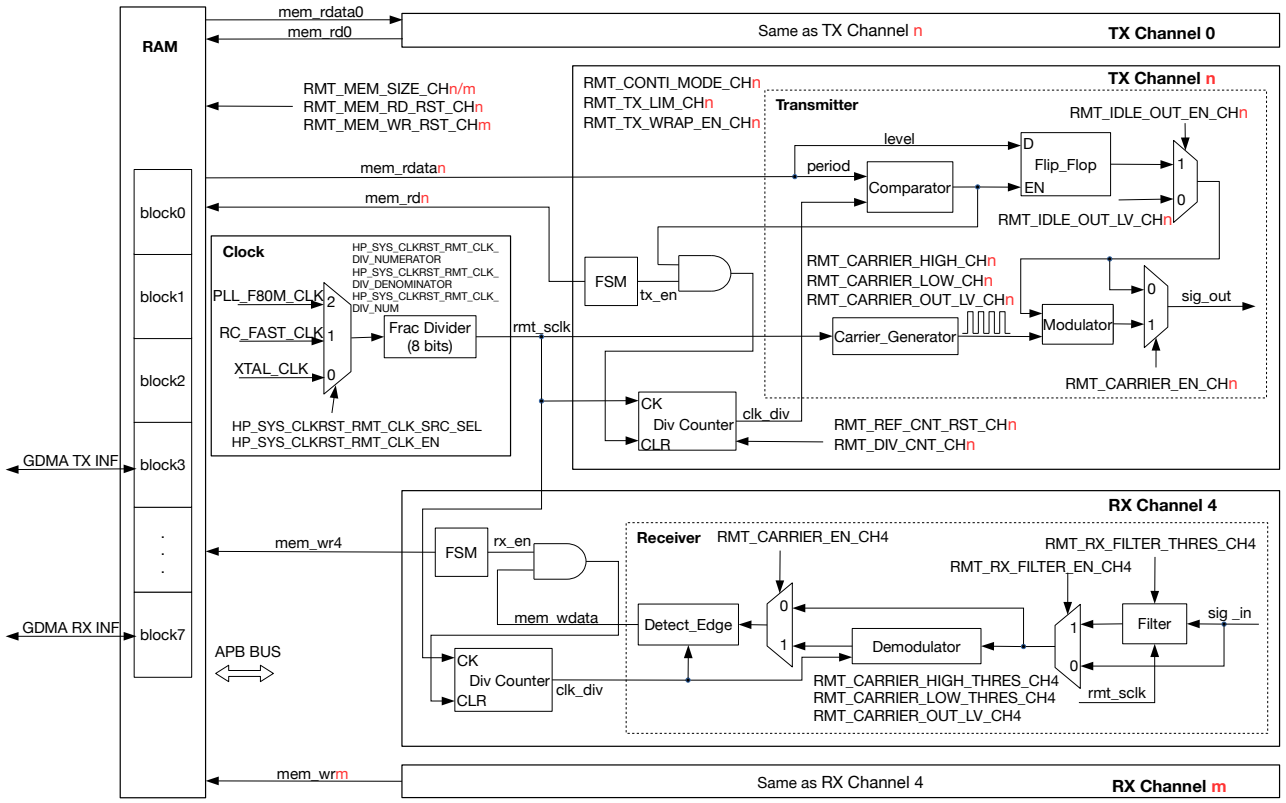


Figure 60.3-1. RMT Architecture

As shown in Figure 60.3-1, each TX channel has:

- 1 x clock divider counter (Div Counter)
- 1 x state machine (FSM)
- 1 x transmitter

Each RX channel has:

- 1 x clock divider counter (Div Counter)

- 1 x state machine (FSM)
- 1 x receiver

The eight channels share a 384 x 32-bit RAM.

60.3.2 RAM

60.3.2.1 Structure of RAM

Figure 60.3-2 shows the format of pulse code in RAM. Each pulse code contains a 16-bit entry with two fields: "level" and "period". "level" (0 or 1) indicates a low-/high-level value that has been received or is going to be sent, while "period" points out the number of clock cycles (see clk_div in Figure 60.3-1) that the level lasts for.

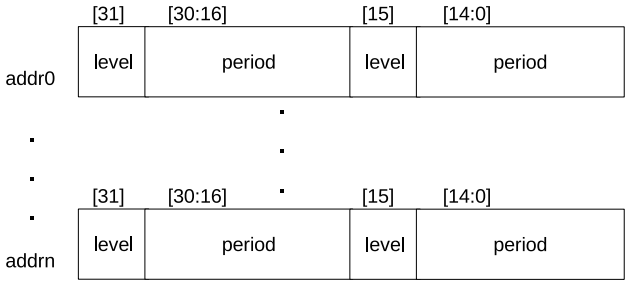


Figure 60.3-2. Format of Pulse Code in RAM

The minimum value for the period is zero (0) and is interpreted as a transmission end-marker. For a non-zero period (i.e., not an end-marker), its value is limited by APB clock and RMT clock according to the formula below:

$$5 \times T_{apb_clk} + 6 \times T_{rmt_sclk} < period \times T_{clk_div} \tag{60.1}$$

Note:
RMT clock and APB clock must satisfy the following relation:

$$1.5 \times T_{apb_clk} < 9 \times T_{rmt_sclk} \tag{60.2}$$

According to the formula above and the frequency of rmt_sclk, the pulse width (i.e., $period \times T_{clk_div}$) able to be captured by RMT is limited as follows:

- The minimum value of pulse width should be larger than $(5 \times T_{apb_clk} + 6 \times T_{rmt_sclk})$.
- The maximum value of pulse width should be smaller than or equal to (the maximum period $\times$ the maximum T_{clk_div}), i.e., $((2^{15} - 1) \times$ the maximum $T_{rmt_sclk} \times 256)$.

For more information about the rmt_sclk frequency, or APB_CLK frequency, see Section 60.3.3, or Chapter 11 *Reset and Clock*.

60.3.2.2 Use of RAM

The RAM is divided into eight 48 x 32-bit blocks. By default, each channel uses one block (block 0 for channel 0, block 1 for channel 1, and so on).

If the data size of one single transfer is larger than one block size of TX channel *n* or RX channel *m*, users can configure the channel:

- to enable wrap mode by setting RMT_MEM_TX/RX_WRAP_EN_CH*n*/*m*;
- or to use more blocks by configuring RMT_MEM_SIZE_CH*n*/*m*.

Setting RMT_MEM_SIZE_CH*n*/*m* > 1 allows channel *n*/*m* to use the memory of the subsequent channels, i.e., block (*n*/*m*) ~ block (*n*/*m* + RMT_MEM_SIZE_CH*n*/*m* - 1). In such case, the subsequent channels (*n*/*m* + 1) ~ (*n*/*m* + RMT_MEM_SIZE_CH*n*/*m* - 1) can not be used since their RAM blocks are occupied. For example, if channel 0 is configured to use block 0 and block 1, then channel 1 will be unavailable since its block is occupied, while channel 2 and channel 3 are not affected and can be used normally.

Note that the RAM used by each channel is mapped from low address to high address. Under such mapping, channel 0 is able to use the RAM blocks for channels 1, 2 ... and 7 by setting RMT_MEM_SIZE_CH0, but channel 7 can not use the blocks for channels 0, 1, ... or 6. Therefore, the value of RMT_MEM_SIZE_CH*n* should not exceed (8 - *n*) and the value of RMT_MEM_SIZE_CH*m* should not exceed (8 - *m*).

The RMT RAM can be accessed via APB bus, or read by the transmitter and written by the receiver. To avoid any possible access conflict between the receiver writing RAM and the APB bus reading RAM, RMT can be configured to designate the RAM block's owner, be it the receiver or the APB bus, by configuring RMT_MEM_OWNER_CH*m*. If this ownership is violated, a flag signal RMT_MEM_OWNER_ERR_CH*m* will be generated.

60.3.2.3 RAM Access

APB bus is able to access RAM in FIFO mode and in NONFIFO (Direct Address) mode, depending on the configuration of RMT_APB_FIFO_MASK:

- 0: Use FIFO mode;
- 1: Use NONFIFO mode.

Channels 3 and 7 also support GDMA access.

FIFO Mode

In FIFO mode, the APB reads data from or writes data to RAM via a fixed address stored in RMT_CH*n*/*m*DATA_REG.

NONFIFO Mode

In NONFIFO mode, the APB writes data to or reads data from a continuous address range.

- The write-starting address of TX channel *n* is: RMT base address + 0x800 + *n*x48. The access address for the second data and the following data are RMT base address + 0x800 + *n*x48 + 0x4, and so on, incremented by 0x4.
- The read-starting address of RX channel *m* is: RMT base address + 0x8C0 + *m*x48. The access address for the second data and the following data are RMT base address + 0x8C0 + *m*x48 + 0x4, and so on, incremented by 0x4.

GDMA Mode

Channel 3 also supports GDMA access. If RMT_DMA_ACCESS_EN_CH3 is set, RAM of channel 3 only allows GDMA access. FIFO access or NOFIFO access to channel 3 by the APB bus are forbidden, otherwise **unpredictable consequences may occur.**

To ensure correct data transmission,

1. GDMA should be started first.
2. RMT can only start transmitting data after GDMA channel gets data ready, otherwise, unexpected data may be sent.

In normal TX mode, when the RAM of channel 3 is fully written by GDMA, an RMT_APB_MEM_WR_ERR_CH3 interrupt is triggered. Setting RMT_MEM_TX_WRAP_EN_CH3 allows channel 3 to transmit more data than one block can fit, with no software operation needed.

Channel 7 also supports GDMA access. If RMT_DMA_ACCESS_EN_CH7 is set, the RAM of channel 7 is allowed to transmit data to GDMA. Note in this mode, channel 7's RAM can also be accessed by APB via NONFIFO mode.

In normal RX mode, when the size of data read by GDMA from channel 7 is equal to its RAM size, an RMT_APB_MEM_RD_ERR_CH7 is triggered and the subsequent data is discarded. If RMT_MEM_RX_WRAP_EN_CH7 is set, data of more than one block size can be received with no software wrap operation needed. If channel 7's RAM is full but the GDMA still does not start receiving data from the channel, the newly received data by this channel will replace the previous data.

Note:
When channel 7 receives an end-maker, a GDMA in_suc_eof interrupt is generated. Two bytes are written to GDMA if the period[14:0] is 0, and four bytes to GDMA if the period[30:16] is 0.

60.3.3 Clock

The clock source of RMT can be PLL_F80M_CLK, RC_FAST_CLK, or XTAL_CLK, depending on the configuration of HP_SYS_CLKRST_RMT_CLK_SRC_SEL. RMT clock can be enabled by setting HP_SYS_CLKRST_RMT_CLK_EN. RMT working clock (see rmt_sclk in Figure 60.3-1) is obtained by dividing the selected clock source with a fractional divider. The divider is HP_SYS_CLKRST_RMT_CLK_DIV_NUM + 1 + HP_SYS_CLKRST_RMT_CLK_DIV_NUMERATOR or HP_SYS_CLKRST_RMT_CLK_DIV_DENOMINATOR.

For more information, see Chapter 11 *Reset and Clock*. RMT_DIV_CNT_CH*n*/*m* is used to configure the divider coefficient of internal clock divider for RMT channels. The coefficient is normally equal to the value of RMT_DIV_CNT_CH*n*/*m*, except value 0 that represents divider 256. The clock divider can be reset by setting RMT_REF_CNT_RST_CH*n*/*m*. The clock generated from the divider can be used by the counter (see Figure 60.3-1).

60.3.4 Transmitter

Note:
Updating the configuration described in this and subsequent sections requires setting RMT_CONF_UPDATE_CH*n*/*m* first. See Section 60.3.6.

60.3.4.1 Normal TX Mode

When RMT_TX_START_CH*n* is set, the transmitter of channel *n* starts reading and transmitting pulse codes from the starting address of its RAM block. The codes are sent starting from low-address entry. When an end-marker (a zero period) is encountered, the transmitter stops the transmission, returns to idle state and generates an RMT_CH*n*_TX_END_INT interrupt. Setting RMT_TX_STOP_CH*n* to 1 also stops the transmission and immediately sets the transmitter back to idle. The output level of a transmitter in idle state is determined by the "level" field of the end-marker or by the content of RMT_IDLE_OUT_LV_CH*n*, depending on the configuration of RMT_IDLE_OUT_EN_CH*n*:

- 0: The level in idle state is determined by the "level" field of the end-marker;
- 1: The level is determined by RMT_IDLE_OUT_LV_CH*n*.

60.3.4.2 Wrap TX Mode

To transmit more pulse codes than that can be fitted in the channel's RAM, users can enable wrap TX mode for channel *n* by setting RMT_MEM_TX_WRAP_EN_CH*n*. In this mode, the transmitter transmits the data from RAM in loops till an end-marker is encountered. For example, if RMT_MEM_SIZE_CH*n* = 1, the transmitter starts transmitting data from the address 48 x *n*, and then the data from higher RAM address. Once the transmitter finishes transmitting data from (48 x (*n* + 1) - 1), it continues transmitting data from 48 x *n* again till an end-marker is encountered. Wrap mode is also applicable for RMT_MEM_SIZE_CH*n* > 1.

When the size of transmitted pulse codes is larger than or equal to the value set by RMT_TX_LIM_CH*n*, an RMT_CH*n*_TX_THR_EVENT_INT interrupt is generated. In wrap mode, RMT_TX_LIM_CH*n* can be set to a half or a fraction of the size of the channel's RAM block. When an RMT_CH*n*_TX_THR_EVENT_INT interrupt is detected by software, the already used RAM region can be updated by new pulse codes. In such way, the transmitter can seamlessly transmit unlimited pulse codes in wrap mode.

Note:
If RAM is accessed by GDMA, more pulse codes than one block size can be transmitted with no additional operation needed. If accessed by APB bus, wrap mode has to be enabled via software to transmit more data than one block size.

60.3.4.3 TX Modulation

Transmitter output can be modulated with a carrier wave by setting RMT_CARRIER_EN_CH*n*. The carrier waveform is configurable. In a carrier cycle, the high level lasts for (RMT_CARRIER_HIGH_CH*n* + 1) rmt_sclk cycles, while the low level lasts for (RMT_CARRIER_LOW_CH*n* + 1) rmt_sclk cycles. When RMT_CARRIER_OUT_LV_CH*n* is set, carrier wave is added on the high-level of output signals; while RMT_CARRIER_OUT_LV_CH*n* is cleared, carrier wave is added on the low-level of output signals. Carrier wave can be added on all output signals during modulation, or just added on valid pulse codes (the data stored in RAM), which can be set by configuring RMT_CARRIER_EFF_EN_CH*n*:

- 0: Add carrier wave on all output signals;

- 1: Add carrier wave only on valid signals.

60.3.4.4 Continuous TX Mode

The continuous TX mode can be enabled by setting RMT_TX_CONTI_MODE_CH*n*. In this mode, the transmitter transmits the pulse codes from RAM in loops:

- If an end-marker is encountered, the transmitter starts transmitting from the first data of the channel's RAM again.
- If no end-marker is encountered, the transmitter starts transmitting from the first data again after the last data is transmitted.

If RMT_TX_LOOP_CNT_EN_CH*n* is set, the loop counting is incremented by 1 each time an end-marker is encountered. If the counting reaches the value set by RMT_TX_LOOP_NUM_CH*n*, an RMT_CH*n*_TX_LOOP_INT interrupt is generated. If RMT_LOOP_STOP_EN_CH*n* is set, the transmission stops immediately once an RMT_CH*n*_TX_LOOP_INT interrupt is generated, otherwise, the transmission will continue. In an end-maker, if its period[14:0] is 0, then the period of the previous data must satisfy:

$$10 \times T_{apb_clk} + 19 \times T_{rmt_sclk} < period \times T_{clk_div} \tag{60.3}$$

The period of the other data only need to satisfy non-zero period relation.

60.3.4.5 Simultaneous TX Mode

RMT module supports multiple channels transmitting data simultaneously. To use this function, follow the steps below:

1. Configure RMT_TX_SIM_CH*n* to choose which multiple channels are used to transmit data simultaneously.
2. Set RMT_TX_SIM_EN to enable this transmission mode.
3. Set RMT_TX_START_CH*n* for each selected channel, to start data transmitting.

The transmission starts once the final channel is configured. RMT module also supports simultaneous transmission of channels 0 ~ 2's RAM accessed by APB bus and channel 3's RAM accessed by GDMA.

60.3.5 Receiver

60.3.5.1 Normal RX Mode

The receiver of channel *m* is controlled by RMT_RX_EN_CH*m*:

- 0: The receiver stops receiving data;
- 1: The receiver starts working.

When the receiver becomes active, it starts counting from the first edge of the signal, detecting signal levels and counting clock cycles the level lasts for. Each cycle count (period) is then written back to RAM together with the level information (level). When the receiver detects no change in a signal level for a number of clock cycles more than the value set by RMT_IDLE_THRES_CH*m*, the receiver will stop receiving data, return to idle state, and generate an RMT_CH*m*_RX_END_INT interrupt. Please note that RMT_IDLE_THRES_CH*m* should be

configured to a maximum value according to your application, otherwise a valid received level may be mistaken as a level in idle state. If the RAM space of this RX channel is used up by the received data, the receiver stops receiving data, and an RMT_CH*m*_ERR_INT interrupt is triggered by RAM FULL event.

60.3.5.2 Wrap RX Mode

To receive more pulse codes than can be fitted in the channel's RAM, users can enable wrap mode for channel *m* by configuring RMT_MEM_RX_WRAP_EN_CH*m*. But if RAM is accessed by GDMA, more pulse codes than one block size can be received with no additional operation needed. If accessed by APB bus, wrap mode has to be enabled to transmit more data than one block size. In wrap mode, the receiver stores the received data to RAM space of this channel in loops. Receiving ends, when the receiver detects no change in a signal level for a number of clock cycles more than the value set by RMT_IDLE_THRES_CH*m*. The receiver returns to idle state and generates an RMT_CH*m*_RX_END_INT interrupt. For example, if RMT_MEM_SIZE_CH*m* is set to 1, the receiver starts receiving data and stores the data to address 48 * *m*, and then to higher RAM address. When the receiver finishes storing the received data to (48 * (*m* + 1) - 1), the receiver continues receiving data and storing data to the address 48 * *m* again, till no change is detected on a signal level for more than RMT_IDLE_THRES_CH*m* clock cycles. Wrap mode is also applicable for RMT_MEM_SIZE_CH*m* > 1.

An RMT_CH*m*_RX_THR_EVENT_INT interrupt is generated when the size of received pulse codes is larger than or equal to the value set by RMT_CH*m*_RX_LIM_REG. In wrap mode, RMT_CH*m*_RX_LIM_REG can be set to a half or a fraction of the size of the channel's RAM block. When an RMT_CH*m*_RX_THR_EVENT_INT interrupt is detected, the already used RAM region can be updated by subsequent data.

60.3.5.3 RX Filtering

Users can enable the receiver to filter input signals by setting RMT_RX_FILTER_EN_CH*m* for channel *m*. The filter samples input signals continuously, and detects the signals which remain unchanged for a continuous RMT_RX_FILTER_THRES_CH*m* rmt_sclk cycles as valid, otherwise, the signals will be detected as invalid. Only the valid signals can pass through this filter. The filter removes pulses with a length of less than RMT_RX_FILTER_THRES_CH*m* rmt_sclk cycles.

60.3.5.4 RX Demodulation

Users can enable RX demodulation on input signals or on filtered signals by setting RMT_CARRIER_EN_CH*m*. RX demodulation can be applied to high-level carrier wave or low-level carrier wave, depending on the configuration of RMT_CARRIER_OUT_LV_CH*m*:

- 0: Demodulate low-level carrier wave;
- 1: Demodulate high-level carrier wave.

Users can configure RMT_CARRIER_HIGH_THRES_CH*m* and RMT_CARRIER_LOW_THRES_CH*m* to set the thresholds to demodulate high-level carrier or low-level carrier. If the high-level of a signal lasts for less than RMT_CARRIER_HIGH_THRES_CH*m* clk_div cycles, or the low-level lasts for less than RMT_CARRIER_LOW_THRES_CH*m* clk_div cycles, such level is detected as a carrier and then is filtered out.

60.3.6 Configuration Update

To update RMT registers configuration, please set RMT_CONF_UPDATE_CH*n*/*m* for each channel first. All the bits/fields listed in the second column of Table 60.3-1 should follow this rule.

Table 60.3-1. Configuration Update

Register	Bit/Field Configuration Update
TX Channel	
RMT_CH*n*CONFO_REG	RMT_CARRIER_OUT_LV_CH*n*
	RMT_CARRIER_EN_CH*n*
	RMT_CARRIER_EFF_EN_CH*n*
	RMT_DIV_CNT_CH*n*
	RMT_TX_STOP_CH*n*
	RMT_IDLE_OUT_EN_CH*n*
	RMT_IDLE_OUT_LV_CH*n*
	RMT_TX_CONTI_MODE_CH*n*
RMT_CH*n*CARRIER_DUTY_REG	RMT_CARRIER_HIGH_CH*n*
	RMT_CARRIER_LOW_CH*n*
RMT_CH*n*_TX_LIM_REG	RMT_TX_LOOP_CNT_EN_CH*n*
	RMT_TX_LOOP_NUM_CH*n*
	RMT_TX_LIM_CH*n*
RMT_TX_SIM_REG	RMT_TX_SIM_EN
RX Channel	
RMT_CH*m*CONFO_REG	RMT_CARRIER_OUT_LV_CH*m*
	RMT_CARRIER_EN_CH*m*
	RMT_IDLE_THRES_CH*m*
	RMT_DIV_CNT_CH*m*
RMT_CH*m*CONF1_REG	RMT_RX_FILTER_THRES_CH*m*
	RMT_RX_EN_CH*m*
RMT_CH*m*_RX_CARRIER_RM_REG	RMT_CARRIER_HIGH_THRES_CH*m*
	RMT_CARRIER_LOW_THRES_CH*m*
RMT_CH*m*_RX_LIM_REG	RMT_RX_LIM_CH*m*
RMT_REF_CNT_RST_REG	RMT_REF_CNT_RST_CH*m*

60.4 Interrupts

ESP32-P4's RMT can generate the RMT_INTR interrupt signal that will be sent to the Interrupt Matrix.

The following interrupt sources can generate the RMT_INTR interrupt signal:

- RMT_CH*n*_ERR_INT: Triggered when channel *n* does not read or write data correctly. For example, the transmitter still tries to read data from RAM when the RAM is empty.
- RMT_CH*m*_ERR_INT: Triggered when channel *m* does not read or write data correctly. For example, the receiver still tries to write data into RAM when the RAM is full.
- RMT_CH*n*_TX_THR_EVENT_INT: Triggered when the amount of data the transmitter has sent matches the value of RMT_CH*n*_TX_LIM_REG.
- RMT_CH*m*_RX_THR_EVENT_INT: Triggered each time when the amount of data received by the receiver reaches the value set in RMT_CH*m*_RX_LIM_REG.
- RMT_CH*n*_TX_END_INT: Triggered when the transmitter has finished transmitting signals.

- RMT_CH*m*_RX_END_INT: Triggered when the receiver has finished receiving signals.
- RMT_CH*n*_TX_LOOP_INT: Triggered when the loop counting reaches the value set by RMT_TX_LOOP_NUM_CH*n* in continuous TX mode.
- RMT_CH3_DMA_ACCESS_FAIL_INT: Triggered when the result of (the entries written to channel 3's RAM - the entries transmitted by channel 3) is larger than channel 3's RAM size, but DAM keeps writing data to this channel.
- RMT_CH7_DMA_ACCESS_FAIL_INT: Triggered when the result of (the entries received by channel 7's RAM - the entries read by GDMA) is larger than channel 7's RAM size, but channel 7 keeps receiving data.

Note:
For definitions of *interrupt*, *interrupt signal*, *interrupt source*, and their correlations, please refer to Chapter 13 *Interrupt Matrix* > Section 13.2 *Interrupt Terminology in ESP32-P4*.

Each interrupt source can be configured by a common set of registers that are described in Section *Interrupt Configuration Registers*. The specific registers can be found in Section 60.5 *Register Summary*.

60.5 Register Summary

The addresses in this section are relative to Remote Control Peripheral base address provided in Table 9.3-2 in Chapter 9 *System and Memory*.

The abbreviations given in Column **Access** are explained in Section *Access Types for Registers*.

Name	Description	Address	Access
FIFO R/W registers			
RMT_CH0DATA_REG	The read and write data register for channel 0 by APB FIFO access	0x0000	HRO
RMT_CH1DATA_REG	The read and write data register for channel 1 by APB FIFO access	0x0004	HRO
RMT_CH2DATA_REG	The read and write data register for channel 2 by APB FIFO access	0x0008	HRO
RMT_CH3DATA_REG	The read and write data register for channel 3 by APB FIFO access	0x000C	HRO
RMT_CH4DATA_REG	The read and write data register for channel 4 by APB FIFO access	0x0010	HRO
RMT_CH5DATA_REG	The read and write data register for channel 5 by APB FIFO access	0x0014	HRO
RMT_CH6DATA_REG	The read and write data register for channel 6 by APB FIFO access	0x0018	HRO
RMT_CH7DATA_REG	The read and write data register for channel 7 by APB FIFO access	0x001C	HRO
Configuration registers			
RMT_CH0CONF0_REG	Configuration register 0 for channel 0	0x0020	varies
RMT_CH1CONF0_REG	Configuration register 0 for channel 1	0x0024	varies
RMT_CH2CONF0_REG	Configuration register 0 for channel 2	0x0028	varies
RMT_CH3CONF0_REG	Configuration register 0 for channel 3	0x002C	varies
RMT_CH4CONF0_REG	Configuration register 0 for channel 4	0x0030	R/W
RMT_CH4CONF1_REG	Configuration register 1 for channel 4	0x0034	varies
RMT_CH5CONF0_REG	Configuration register 0 for channel 5	0x0038	R/W
RMT_CH5CONF1_REG	Configuration register 1 for channel 5	0x003C	varies
RMT_CH6CONF0_REG	Configuration register 0 for channel 6	0x0040	R/W
RMT_CH6CONF1_REG	Configuration register 1 for channel 6	0x0044	varies
RMT_CH7CONF0_REG	Configuration register 0 for channel 7	0x0048	R/W
RMT_CH7CONF1_REG	Configuration register 1 for channel 7	0x004C	varies
RMT_CH4_RX_CARRIER_RM_REG	Carrier remove register for channel 4	0x0090	R/W
RMT_CH5_RX_CARRIER_RM_REG	Carrier remove register for channel 5	0x0094	R/W
RMT_CH6_RX_CARRIER_RM_REG	Carrier remove register for channel 6	0x0098	R/W
RMT_CH7_RX_CARRIER_RM_REG	Carrier remove register for channel 7	0x009C	R/W
RMT_SYS_CONF_REG	RMT APB configuration register	0x00C0	R/W
RMT_REF_CNT_RST_REG	RMT clock divider reset register	0x00C8	WT
Status registers			

Name	Description	Address	Access
RMT_CH0STATUS_REG	Status register for channel 0	0x0050	RO
RMT_CH1STATUS_REG	Status register for channel 1	0x0054	RO
RMT_CH2STATUS_REG	Status register for channel 2	0x0058	RO
RMT_CH3STATUS_REG	Status register for channel 3	0x005C	RO
RMT_CH4STATUS_REG	Status register for channel 4	0x0060	RO
RMT_CH5STATUS_REG	Status register for channel 5	0x0064	RO
RMT_CH6STATUS_REG	Status register for channel 6	0x0068	RO
RMT_CH7STATUS_REG	Status register for channel 7	0x006C	RO
Interrupt registers			
RMT_INT_RAW_REG	Interrupt raw signal status register	0x0070	R/WTC/SS
RMT_INT_ST_REG	Interrupt signal status register	0x0074	RO
RMT_INT_ENA_REG	Interrupt enable signal configuration register	0x0078	R/W
RMT_INT_CLR_REG	Interrupt clear signal configuration register	0x007C	WT
Carrier wave duty cycle registers			
RMT_CH0CARRIER_DUTY_REG	Duty cycle configuration register for channel 0	0x0080	R/W
RMT_CH1CARRIER_DUTY_REG	Duty cycle configuration register for channel 1	0x0084	R/W
RMT_CH2CARRIER_DUTY_REG	Duty cycle configuration register for channel 2	0x0088	R/W
RMT_CH3CARRIER_DUTY_REG	Duty cycle configuration register for channel 3	0x008C	R/W
TX event configuration registers			
RMT_CH0_TX_LIM_REG	Configuration register for channel 0 TX event	0x00A0	varies
RMT_CH1_TX_LIM_REG	Configuration register for channel 1 TX event	0x00A4	varies
RMT_CH2_TX_LIM_REG	Configuration register for channel 2 TX event	0x00A8	varies
RMT_CH3_TX_LIM_REG	Configuration register for channel 3 TX event	0x00AC	varies
RMT_TX_SIM_REG	RMT TX synchronous register	0x00C4	R/W
RX event configuration registers			
RMT_CH4_RX_LIM_REG	Configuration register for channel 4 RX event	0x00B0	R/W
RMT_CH5_RX_LIM_REG	Configuration register for channel 5 RX event	0x00B4	R/W
RMT_CH6_RX_LIM_REG	Configuration register for channel 6 RX event	0x00B8	R/W
RMT_CH7_RX_LIM_REG	Configuration register for channel 7 RX event	0x00BC	R/W
Version register			
RMT_DATE_REG	Version control register	0x00CC	R/W

60.6 Registers

The addresses in this section are relative to Remote Control Peripheral base address provided in Table 9.3-2 in Chapter 9 *System and Memory*.

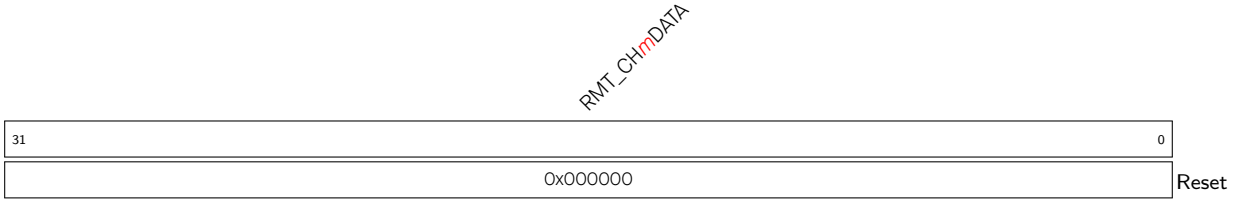
For how to program reserved fields, please refer to Section *Programming Reserved Register Field*.

Register 60.1. RMT_CH*n*DATA_REG (*n*: 0-3) (0x0000+0x4n*)**



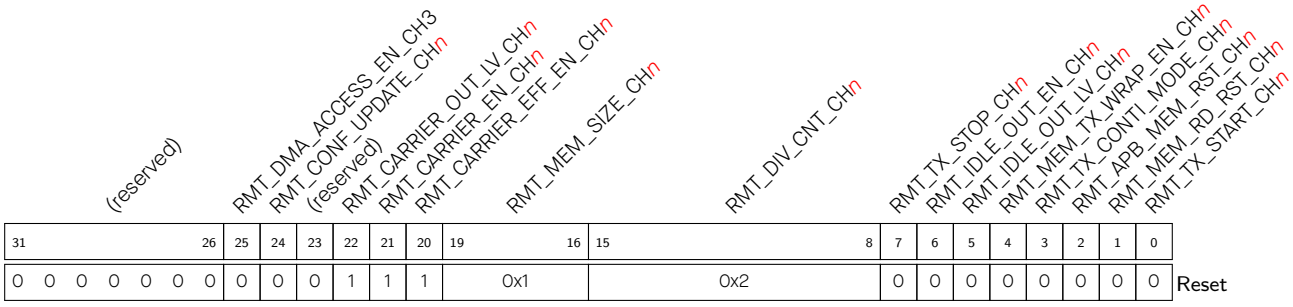
RMT_CH*n*DATA Read and write data for channel *n* via APB FIFO. (HRO)

Register 60.2. RMT_CH*m*DATA_REG (*m*: 4-7) (0x0000+0x4m*)**



RMT_CH*m*DATA Read and write data for channel *n* via APB FIFO. (HRO)

Register 60.3. RMT_CH*n*CONF0_REG (*n*: 0-3) (0x0020+0x4**n*)



RMT_TX_START_CH*n* Configures whether to enable sending data in channel *n*.

0: No effect

1: Enable

(WT)

RMT_MEM_RD_RST_CH*n* Configures whether to reset RAM read address accessed by the transmitter for channel *n*.

0: No effect

1: Reset

(WT)

RMT_APB_MEM_RST_CH*n* Configures whether to reset RAM W/R address accessed by APB FIFO for channel *n*.

0: No effect

1: Reset

(WT)

RMT_TX_CONTI_MODE_CH*n* Configures whether to enable continuous TX mode for channel *n*.

0: No Effect

1: Enable

In this mode, the transmitter starts transmission from the first data. If an end-marker is encountered, the transmitter starts transmitting data from the first data again; if no end-marker is encountered, the transmitter starts transmitting the first data again when the last data is transmitted.

(R/W)

RMT_MEM_TX_WRAP_EN_CH*n* Configures whether to enable wrap TX mode for channel *n*.

0: No effect

1: Enable

In this mode, if the TX data size is larger than the channel's RAM block size, the transmitter continues transmitting the first data to the last data in loops.

(R/W)

RMT_IDLE_OUT_LV_CH*n* Configures the level of output signal for channel *n* when the transmitter is in idle state. (R/W)

Continued on the next page...

Register 60.3. RMT_CH*n*CONF0_REG (*n*: 0-3) (0x0020+0x4**n*)

Continued from the previous page...

RMT_IDLE_OUT_EN_CH*n* Configures whether to enable the output for channel *n* in idle state.
0: No effect
1: Enable
(R/W)

RMT_TX_STOP_CH*n* Configures whether to stop the transmitter of channel *n* sending data out.
0: No effect
1: Stop
(R/W/SC)

RMT_DIV_CNT_CH*n* Configures the divider for clock of channel *n*.
Measurement unit: rmt_sclk
(R/W)

RMT_MEM_SIZE_CH*n* Configures the maximum number of memory blocks allocated to channel *n*.
(R/W)

RMT_CARRIER_EFF_EN_CH*n* Configures whether to add carrier modulation on the output signal only at data-sending state for channel *n*.
0: Add carrier modulation on the output signal at data-sending state and idle state for channel *n*
1: Add carrier modulation on the output signal only at data-sending state for channel *n*
Only valid when RMT_CARRIER_EN_CH*n* is 1.
(R/W)

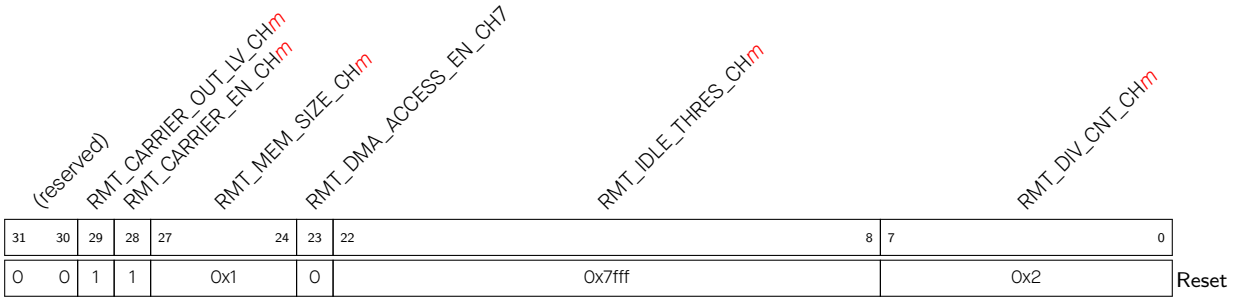
RMT_CARRIER_EN_CH*n* Configures whether to enable the carrier modulation on output signal for channel *n*.
0: Disable
1: Enable
(R/W)

RMT_CARRIER_OUT_LV_CH*n* Configures the position of carrier wave for channel *n*.
0: Add carrier wave on low level
1: Add carrier wave on high level
(R/W)

RMT_CONF_UPDATE_CH*n* Synchronization bit for channel *n*. (WT)

RMT_DMA_ACCESS_EN_CH3 Configures whether to enable the DMA access function for channel 3. This field is reserved for channel 0 ~ 2.
0: Disable
1: Enable
(R/W)

Register 60.4. RMT_CH*m*CONF0_REG (*m*: 4-7) (0x0030, 0x0038, 0x0040, 0x0048)



RMT_DIV_CNT_CH*m* Configures the clock divider of channel *m*.

Measurement unit: rmt_sclk

(R/W)

RMT_IDLE_THRES_CH*m* Configures RX threshold. When no edge is detected on the input signal for continuous clock cycles longer than this field value, the receiver stops receiving data.

Measurement unit: clk_div

(R/W)

RMT_DMA_ACCESS_EN_CH7 Configures whether to enable the DMA access function for channel 7. This field is reserved for channel 4 ~ 6.

0: Disable

1: Enable

(R/W)

RMT_MEM_SIZE_CH*m* Configures the maximum number of memory blocks allocated to channel *m*.

(R/W)

RMT_CARRIER_EN_CH*m* Configures whether to enable carrier demodulation on output signal for channel *m*.

0: Disable

1: Enable

(R/W)

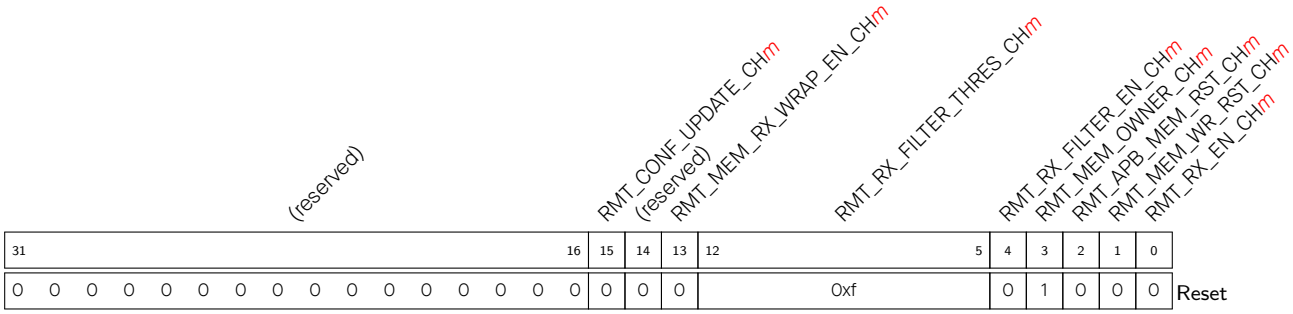
RMT_CARRIER_OUT_LV_CH*m* Configures the position of carrier wave for channel *m*.

0: Demodulate carrier wave on low level

1: Demodulate carrier wave on high level

(R/W)

Register 60.5. RMT_CH*m*CONF1_REG (*m*: 4-7) (0x0034, 0x003C, 0x0044, 0x004C)



RMT_RX_EN_CH*m* Configures whether to enable the receiver to start receiving data in channel *m*.

0: Disable

1: Enable

(R/W)

RMT_MEM_WR_RST_CH*m* Configures whether to reset RAM write address accessed by the receiver for channel *m*.

0: No effect

1: Reset

(WT)

RMT_APB_MEM_RST_CH*m* Configures whether to reset RAM W/R address accessed by APB FIFO for channel *m*.

0: No effect

1: Reset

(WT)

RMT_MEM_OWNER_CH*m* Configures the ownership of channel *m*'s RAM block.

0: APB bus is using the RAM

1: Receiver is using the RAM

(R/W/SC)

RMT_RX_FILTER_EN_CH*m* Configures whether to enable the receiver's filter for channel *m*.

0: Disable

1: Enable

(R/W)

RMT_RX_FILTER_THRES_CH*m* Configures whether the receiver, when receiving data, ignores the input pulse when its width is shorter than this register value in units of rmt_sclk cycles.

0: No effect

1: Reset

(R/W)

RMT_MEM_RX_WRAP_EN_CH*m* Configures whether to enable wrap RX mode for channel *m*.

0: Disable

1: Enable

In this mode, if the RX data size is larger than channel *m*'s RAM block size, the receiver stores the RX data from the first address to the last address in loops.

(R/W)

RMT_CONF_UPDATE_CH*m* Synchronization bit for channel *m*. (WT)

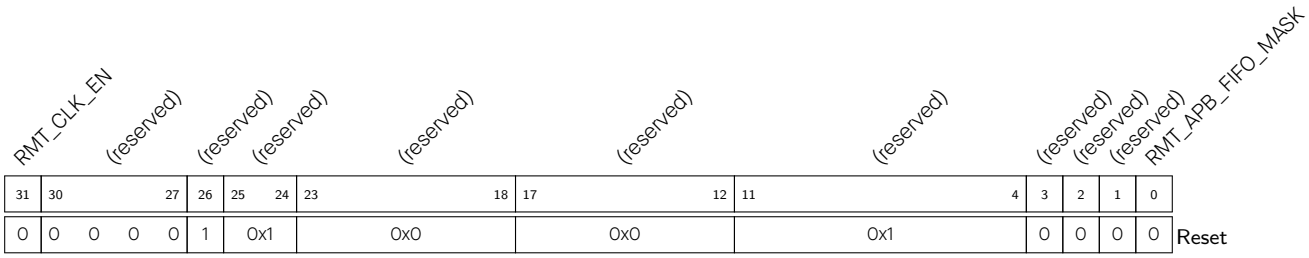
Register 60.6. RMT_CH*m*_RX_CARRIER_RM_REG (*m*: 4-7) (0x0090, 0x0094, 0x0098, 0x009C)



RMT_CARRIER_LOW_THRES_CH*m* Configures the low level period in a carrier modulation mode for channel *m*, which is equal to (RMT_CARRIER_LOW_THRES_CH*m* + 1). (R/W)

RMT_CARRIER_HIGH_THRES_CH*m* Configures the high level period in a carrier modulation mode for channel *m*, which is equal to (RMT_CARRIER_HIGH_THRES_CH*m* + 1). (R/W)

Register 60.7. RMT_SYS_CONF_REG (0x00C0)



RMT_APB_FIFO_MASK Configures the memory access mode.

0: Access memory by FIFO

1: Access memory directly

(R/W)

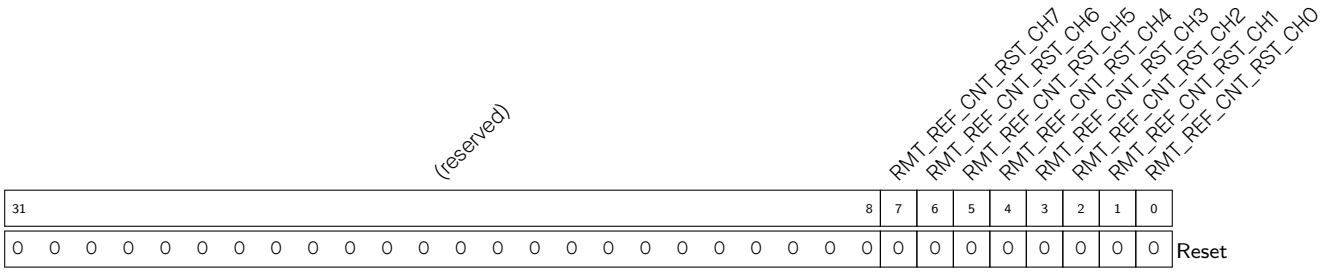
RMT_CLK_EN Configures whether to enable signal of RMT register clock gate.

0: Power down the drive clock of registers

1: Power up the drive clock of registers

(R/W)

Register 60.8. RMT_REF_CNT_RST_REG (0x00C8)



RMT_REF_CNT_RST_CH*n* (*n*: 0-3) Configures whether to reset the clock divider of channel *n*.

0: No effect

1: Reset

(WT)

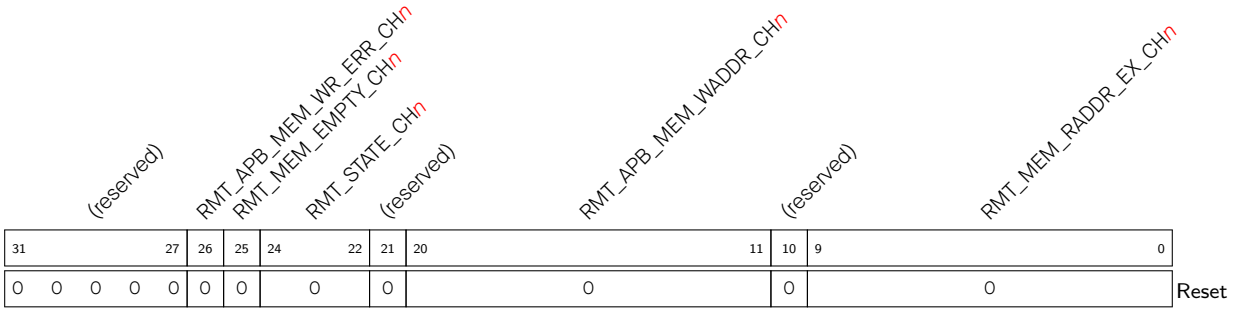
RMT_REF_CNT_RST_CH*m* (*m*: 4-7) Configures whether to reset the clock divider of channel *m*.

0: No effect

1: Reset

(WT)

Register 60.9. RMT_CH*n*STATUS_REG (*n*: 0-3) (0x0050+0x4n*)**



RMT_MEM_RADDR_EX_CH*n* Represents the memory address offset when transmitter of channel *n* is using the RAM. (RO)

RMT_APB_MEM_WADDR_CH*n* Represents the memory address offset when writes RAM over APB bus. (RO)

RMT_STATE_CH*n* Represents the FSM status of channel *n*. (RO)

RMT_MEM_EMPTY_CH*n* Represents whether the TX data size exceeds the memory size and the wrap TX mode is disabled.

0: Not exceed

1: Exceed

(RO)

RMT_APB_MEM_WR_ERR_CH*n* Represents whether the offset address exceeds memory size (overflows) when writes via APB bus.

0: Not exceed

1: Exceed

(RO)

Register 60.10. RMT_CH*m*STATUS_REG (*m*: 4-7) (0x0050+0x4m*)**



RMT_MEM_WADDR_EX_CH*m* Represents the memory address offset when receiver of channel *m* is using the RAM. (RO)

RMT_APB_MEM_RADDR_CH*m* Represents the memory address offset when reads RAM over APB bus. (RO)

RMT_STATE_CH*m* Represents the FSM status of channel *m*. (RO)

RMT_MEM_OWNER_ERR_CH*m* Represents whether the ownership of memory block is wrong.

0: The ownership of memory block is correct

1: The ownership of memory block is wrong

(RO)

RMT_MEM_FULL_CH*m* Represents whether the receiver receives more data than the memory can fit.

0: The receiver does not receive more data than the memory can fit

1: The receiver receives more data than the memory can fit

(RO)

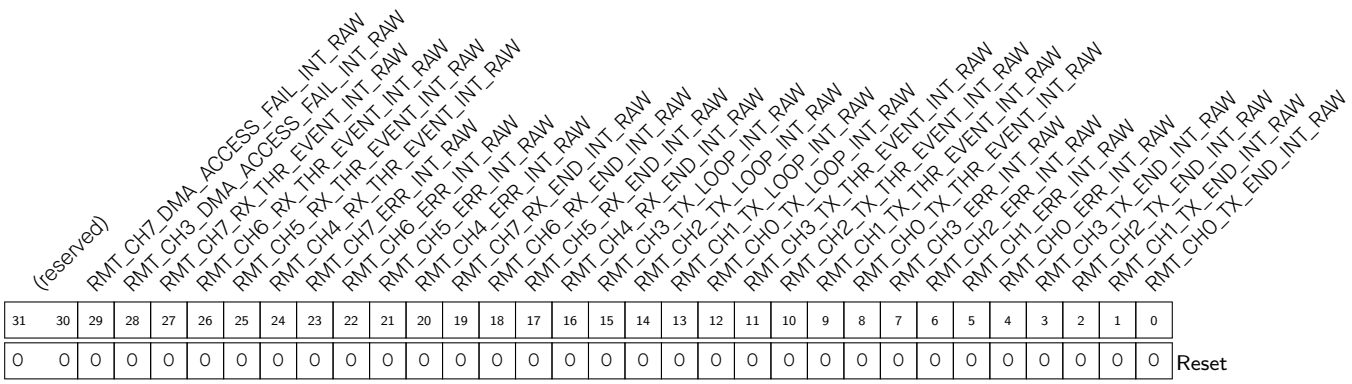
RMT_APB_MEM_RD_ERR_CH*m* Represents whether the offset address exceeds memory size (overflows) when reads RAM via APB bus.

0: Not exceed

1: Exceed

(RO)

Register 60.11. RMT_INT_RAW_REG (0x0070)



RMT_CH*n*_TX_END_INT_RAW (*n*: 0-3) The raw interrupt status of RMT_CH*n*_TX_END_INT. (R/WTC/SS)

RMT_CH*n*_ERR_INT_RAW (*n*: 0-3) The raw interrupt status of RMT_CH*n*_ERR_INT. (R/WTC/SS)

RMT_CH*n*_TX_THR_EVENT_INT_RAW (*n*: 0-3) The raw interrupt status of RMT_CH*n*_TX_THR_EVENT_INT. (R/WTC/SS)

RMT_CH*n*_TX_LOOP_INT_RAW (*n*: 0-3) The raw interrupt status of RMT_CH*n*_TX_LOOP_INT. (R/WTC/SS)

RMT_CH*m*_RX_END_INT_RAW (*m*: 4-7) The raw interrupt status of RMT_CH*m*_RX_END_INT. (R/WTC/SS)

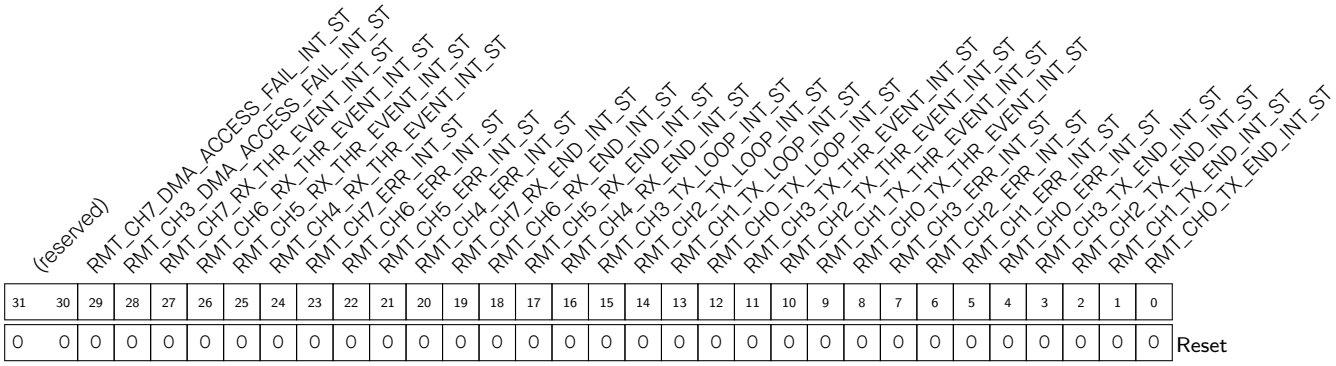
RMT_CH*m*_ERR_INT_RAW (*m*: 4-7) The raw interrupt status of RMT_CH*m*_ERR_INT. (R/WTC/SS)

RMT_CH*m*_RX_THR_EVENT_INT_RAW (*m*: 4-7) The raw interrupt status of RMT_CH*m*_RX_THR_EVENT_INT. (R/WTC/SS)

RMT_CH*n*_DMA_ACCESS_FAIL_INT_RAW (*n*: 3) The raw interrupt status of RMT_CH*n*_DMA_ACCESS_FAIL_INT. (R/WTC/SS)

RMT_CH*m*_DMA_ACCESS_FAIL_INT_RAW (*m*: 7) The raw interrupt status of RMT_CH*m*_DMA_ACCESS_FAIL_INT. (R/WTC/SS)

Register 60.12. RMT_INT_ST_REG (0x0074)



RMT_CH*n*_TX_END_INT_ST (*n*: 0-3) The masked interrupt status of RMT_CH*n*_TX_END_INT. (RO)

RMT_CH*n*_ERR_INT_ST (*n*: 0-3) The masked interrupt status of RMT_CH*n*_ERR_INT. (RO)

RMT_CH*n*_TX_THR_EVENT_INT_ST (*n*: 0-3) The masked interrupt status of RMT_CH*n*_TX_THR_EVENT_INT. (RO)

RMT_CH*n*_TX_LOOP_INT_ST (*n*: 0-3) The masked interrupt status of RMT_CH*n*_TX_LOOP_INT. (RO)

RMT_CH*m*_RX_END_INT_ST (*m*: 4-7) The masked interrupt status of RMT_CH*m*_RX_END_INT. (RO)

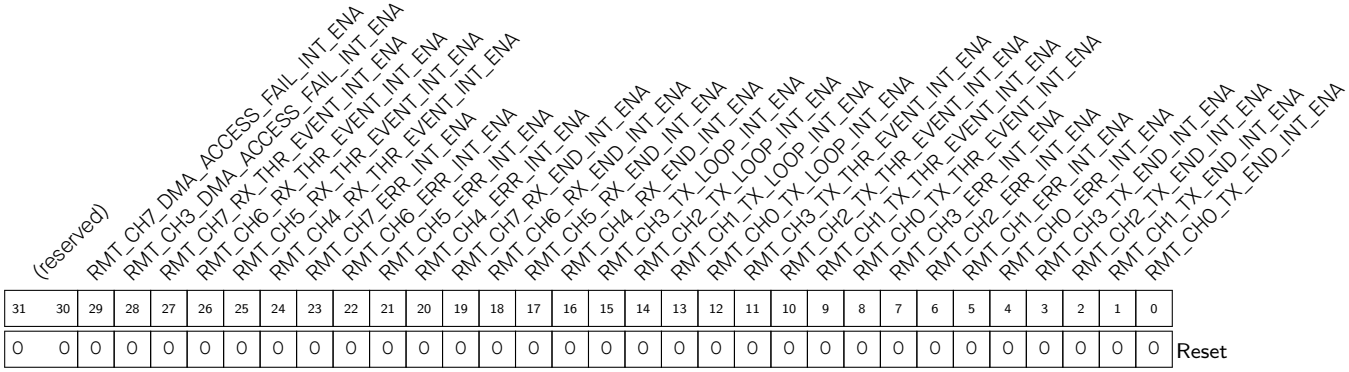
RMT_CH*m*_ERR_INT_ST (*m*: 4-7) The masked interrupt status of RMT_CH*m*_ERR_INT. (RO)

RMT_CH*m*_RX_THR_EVENT_INT_ST (*m*: 4-7) The masked interrupt status of RMT_CH*m*_RX_THR_EVENT_INT. (RO)

RMT_CH*n*_DMA_ACCESS_FAIL_INT_ST (*n*: 3) The masked interrupt status of RMT_CH*n*_DMA_ACCESS_FAIL_INT. (RO)

RMT_CH*m*_DMA_ACCESS_FAIL_INT_ST (*m*: 7) The masked interrupt status of RMT_CH*m*_DMA_ACCESS_FAIL_INT. (RO)

Register 60.13. RMT_INT_ENA_REG (0x0078)



RMT_CH*n*_TX_END_INT_ENA (*n*: 0-3) Write 1 to enable RMT_CH*n*_TX_END_INT. (R/W)

RMT_CH*n*_ERR_INT_ENA (*n*: 0-3) Write 1 to enable RMT_CH*n*_ERR_INT. (R/W)

RMT_CH*n*_TX_THR_EVENT_INT_ENA (*n*: 0-3) Write 1 to enable RMT_CH*n*_TX_THR_EVENT_INT. (R/W)

RMT_CH*n*_TX_LOOP_INT_ENA (*n*: 0-3) Write 1 to enable RMT_CH*n*_TX_LOOP_INT. (R/W)

RMT_CH*m*_RX_END_INT_ENA (*m*: 4-7) Write 1 to enable RMT_CH*m*_RX_END_INT. (R/W)

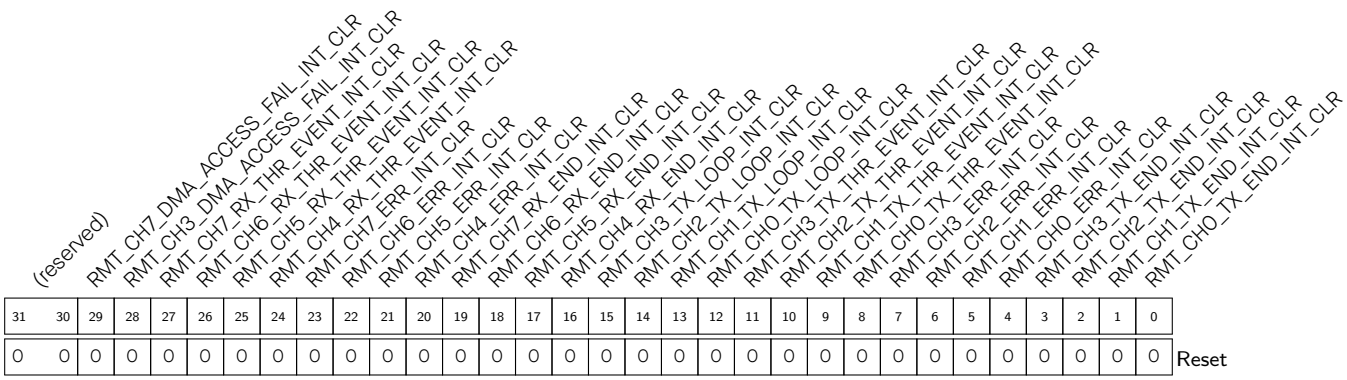
RMT_CH*m*_ERR_INT_ENA (*m*: 4-7) Write 1 to enable RMT_CH*m*_ERR_INT. (R/W)

RMT_CH*m*_RX_THR_EVENT_INT_ENA (*m*: 4-7) Write 1 to enable RMT_CH*m*_RX_THR_EVENT_INT. (R/W)

RMT_CH*n*_DMA_ACCESS_FAIL_INT_ENA (*n*: 3) Write 1 to enable RMT_CH*n*_DMA_ACCESS_FAIL_INT. (R/W)

RMT_CH*m*_DMA_ACCESS_FAIL_INT_ENA (*m*: 7) Write 1 to enable RMT_CH*m*_DMA_ACCESS_FAIL_INT. (R/W)

Register 60.14. RMT_INT_CLR_REG (0x007C)



RMT_CH*n*_TX_END_INT_CLR (*n*: 0-3) Write 1 to clear RMT_CH*n*_TX_END_INT. (WT)

RMT_CH*n*_ERR_INT_CLR (*n*: 0-3) Write 1 to clear RMT_CH*n*_ERR_INT. (WT)

RMT_CH*n*_TX_THR_EVENT_INT_CLR (*n*: 0-3) Write 1 to clear RMT_CH*n*_TX_THR_EVENT_INT. (WT)

RMT_CH*n*_TX_LOOP_INT_CLR (*n*: 0-3) Write 1 to clear RMT_CH*n*_TX_LOOP_INT. (WT)

RMT_CH*m*_RX_END_INT_CLR (*m*: 4-7) Write 1 to clear RMT_CH*m*_RX_END_INT. (WT)

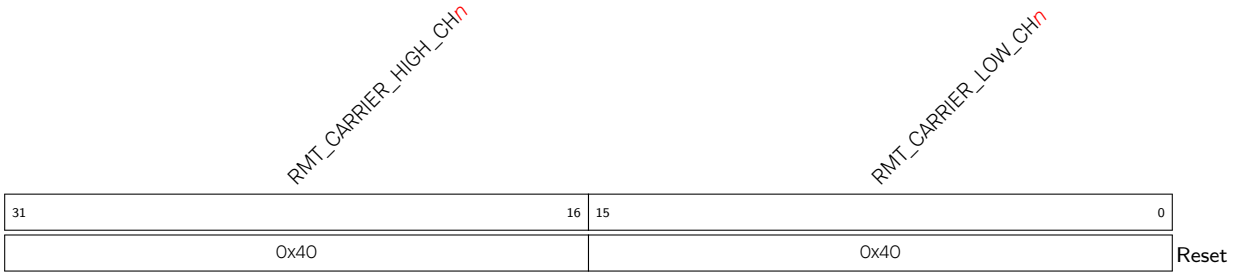
RMT_CH*m*_ERR_INT_CLR (*m*: 4-7) Write 1 to clear RMT_CH*m*_ERR_INT. (WT)

RMT_CH*m*_RX_THR_EVENT_INT_CLR (*m*: 4-7) Write 1 to clear RMT_CH*m*_RX_THR_EVENT_INT. (WT)

RMT_CH*n*_DMA_ACCESS_FAIL_INT_CLR (*n*: 3) Write 1 to clear RMT_CH*n*_DMA_ACCESS_FAIL_INT. (WT)

RMT_CH*m*_DMA_ACCESS_FAIL_INT_CLR (*m*: 7) Write 1 to clear RMT_CH*m*_DMA_ACCESS_FAIL_INT. (WT)

Register 60.15. RMT_CH*n*CARRIER_DUTY_REG (*n*: 0-3) (0x0080+0x4n*)**



RMT_CARRIER_LOW_CH*n* Configures carrier wave's low level clock period for channel *n*.

Measurement unit: rmt_sclk

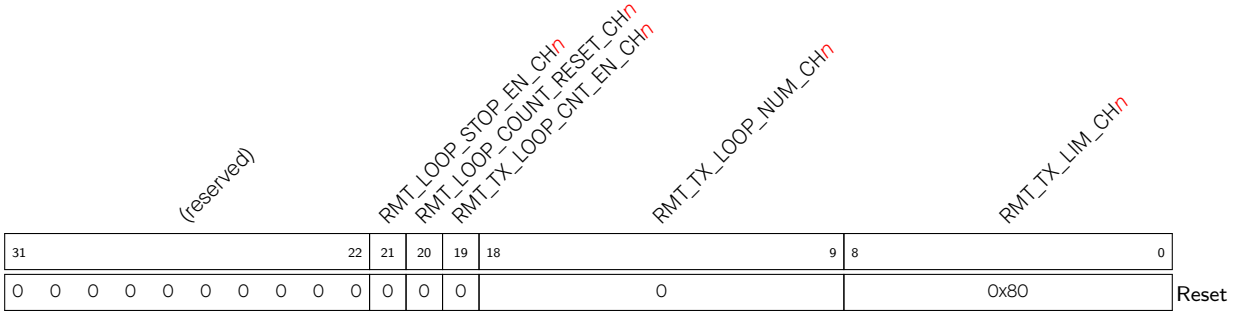
(R/W)

RMT_CARRIER_HIGH_CH*n* Configures carrier wave's high level clock period for channel *n*.

Measurement unit: rmt_sclk

(R/W)

Register 60.16. RMT_CH*n*_TX_LIM_REG (*n*: 0-3) (0x00A0+0x4n*)**



RMT_TX_LIM_CH*n* Configures the maximum entries that channel *n* can send out. (R/W)

RMT_TX_LOOP_NUM_CH*n* Configures the maximum loop count when Continuous TX mode is valid. (R/W)

RMT_TX_LOOP_CNT_EN_CH*n* Configures whether to enable loop count.

0: No effect

1: Enable

(R/W)

RMT_LOOP_COUNT_RESET_CH*n* Configures whether to reset the loop count when continuous TX mode is enabled.

0: No effect

1: Reset

(WT)

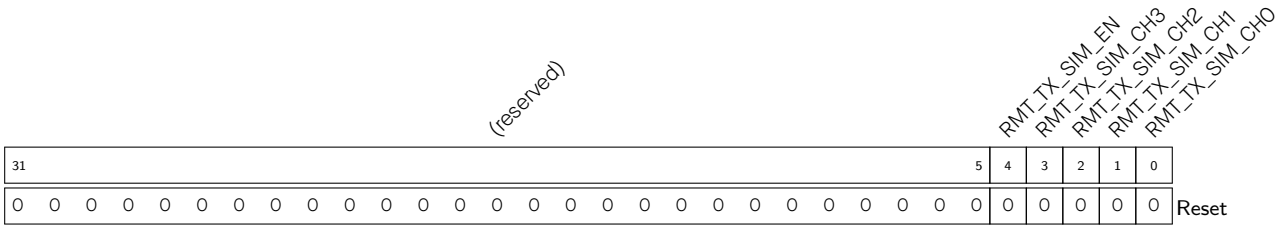
RMT_LOOP_STOP_EN_CH*n* Configures whether to enable the loop send stop function after the loop counter counts to loop number for channel *n*.

0: No effect

1: Enable

(R/W)

Register 60.17. RMT_TX_SIM_REG (0x00C4)



RMT_TX_SIM_CH*n* (*n*: 0-3) Configures whether to enable channel *n* to start sending data synchronously with other enabled channels.

0: No effect

1: Enable

(R/W)

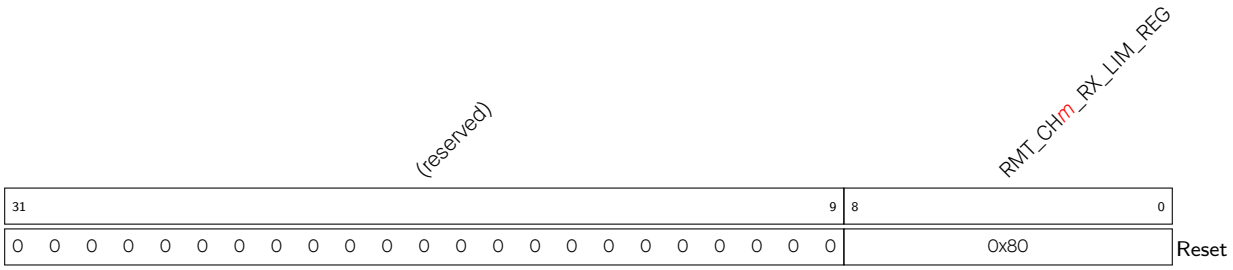
RMT_TX_SIM_EN Configures whether to enable multiple of channels to start sending data synchronously.

0: No effect

1: Enable

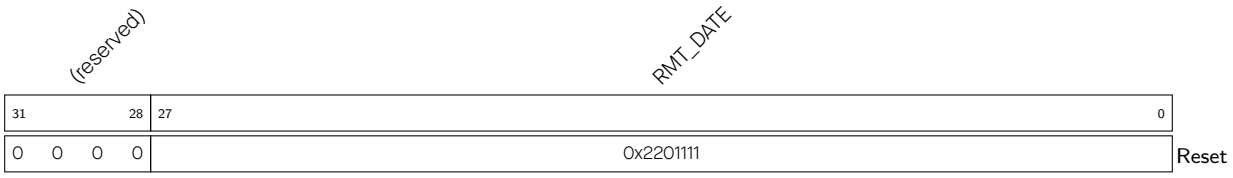
(R/W)

Register 60.18. RMT_CH*m*_RX_LIM_REG (*m*: 4-7) (0x00B0, 0x00B4, 0x00B8, 0x00BC)



RMT_CH*m*_RX_LIM_REG Configures the maximum entries that channel *m* can receive. (R/W)

Register 60.19. RMT_DATE_REG (0x00CC)



RMT_DATE Version control register. (R/W)

Chapter 61

Parallel IO Controller (PARLIO)

61.1 Introduction

ESP32-P4 contains a Parallel IO controller (PARLIO) capable of transferring data between external devices and internal memory on a parallel bus through General Direct Memory Access (GDMA). The PARLIO consists of a TX unit and an RX unit, serving as a transmitter and a receiver respectively. With the two units combined, PARLIO achieves full-duplex communication.

Due to its flexibility, PARLIO can function as a general interface to connect various peripherals. For example, with SPI as the master device and PARLIO as the slave device, a peer-to-peer transfer can be achieved. For detailed application examples, refer to Section 61.8.

61.2 Glossary

This section covers terminology used to describe the functionality of PARLIO.

RX unit	Module in PARLIO responsible for receiving data from the external parallel bus and storing them into internal memory.
TX unit	Module in PARLIO responsible for transmitting data from internal memory to external parallel bus.
RXD	Parallel data received from the IO interface of the RX unit.
TXD	Parallel data sent from the IO interface of the TX unit.
Frame	Transferred data unit from the moment the START signal is set to the moment the End of Frame (EOF) signal is received.
Free-running clock	Clock that toggles continuously as opposed to clock that only toggles when valid data is incoming, and remains constant for the rest of the time.
GDMA SUC EOF	Signal that indicates GDMA successful end of frame. When GDMA receives this signal, a GDMA interrupt will be triggered, indicating that the current frame is correct and the receive is finished.
GDMA ERR EOF	Signal that indicates GDMA error end of frame. When GDMA receives this signal, a GDMA interrupt will be triggered, indicating that the current frame has error and the receive is finished.
CDC	Clock domain crossing.

61.3 Features

The PARLIO module has the following main features:

- Various clock sources:
  - Including external IO clock PAD_CLK_TX/RX and internal system clock XTAL_CLK, PLL_F160M_CLK, and RC_FAST_CLK
  - Maximum IO clock frequency of 40 MHz
  - Integer and fractional clock frequency division
- 1/2/4/8/16-bit configurable data bus width
- Full-duplex communication with 16-bit data bus width
- Bit reversal when data bus width is 1/2/4-bit
- RX unit for receiving IO parallel data, which supports:
  - Output clock gating
  - RX unit input and output clock inverse
  - Various receive modes
  - Configurable GDMA SUC EOF generation
  - Configurable IO pin of external enable signal
- TX unit for sending IO parallel data, which supports:
  -
  - Chip select function
  - Output clock gating
  - TX unit input and output clock inverse
  - Configurable TX EOF generation
  - Configurable bus idle value

61.4 Architectural Overview

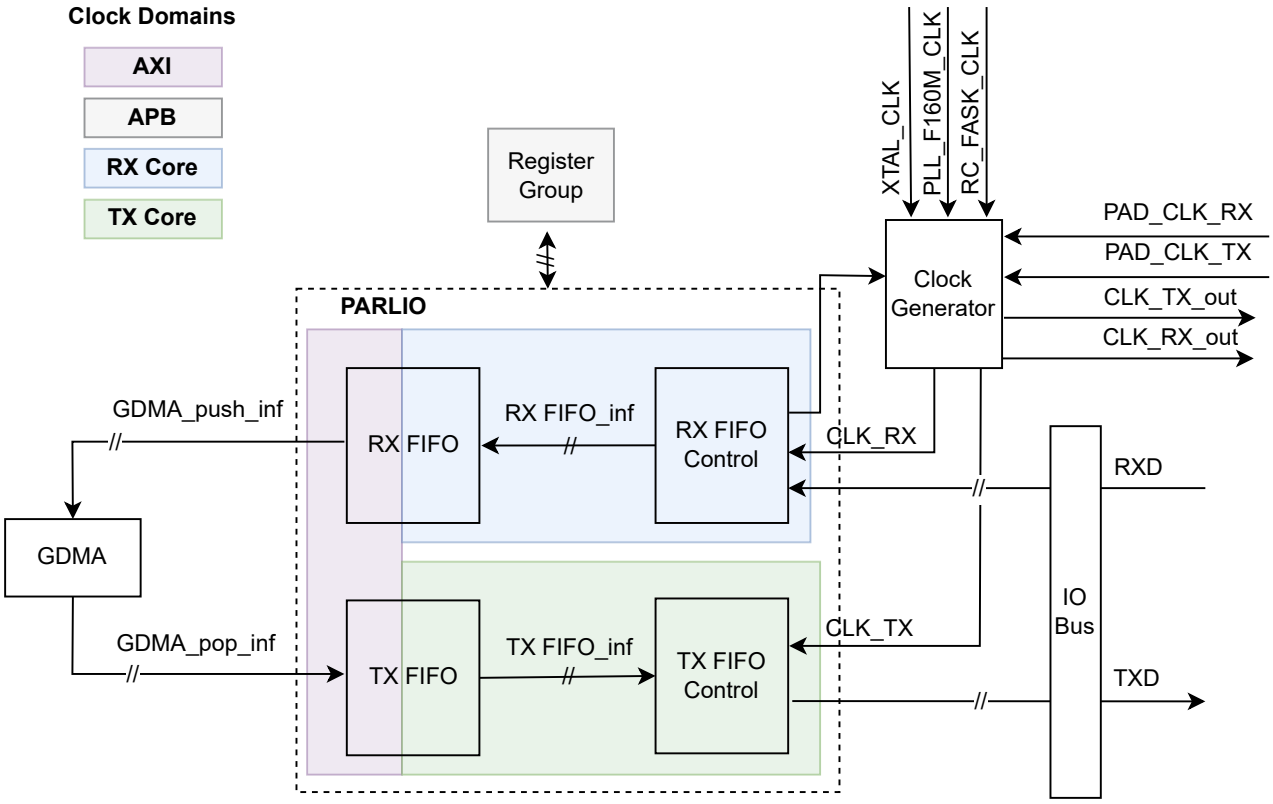


Figure 61.4-1. PARLIO Architecture

Figure 61.4-1 shows the architecture of PARLIO. In addition to the RX unit and the TX unit, a group of status configuration registers is also included.

The RX unit receives the RXD, stores it in an asynchronous FIFO after the serial-to-parallel conversion, and then stores the data to internal memory via GDMA.

The TX unit fetches data from the internal memory via GDMA, stores it in an asynchronous FIFO, and outputs the data from the IO bus via the TXD signal after the parallel-to-serial conversion.

61.5 Functional Description

61.5.1 Clock Generator

There are four input clock domains in PARLIO, namely, RX Core, TX Core, AXI, and APB, as shown in Figure 61.4-1.

The status configuration register group works in the APB clock domain.

The GDMA interface logic works in the AXI clock domain.

The RX Core clock domain has four clock sources for selection, i.e., the internal system clock source XTAL_CLK, RC_FAST_CLK, PLL_F160M_CLK, and the external clock source (PAD_CLK_RX), as shown in Figure 61.5-1. Clock sources can be selected by configuring HP_SYS_CLKRST_PARLIO_RX_CLK_SRC_SEL.

The clock can be divided by configuring HP_SYS_CLKRST_PARLIO_RX_CLK_DIV_NUM, HP_SYS_CLKRST_PARLIO_RX_CLK_DIV_DENOMINATOR, and HP_SYS_CLKRST_PARLIO_RX_CLK_DIV_NUMERATOR. The clock division factor can be configured up to $(2^8 - 1)$.

The TX Core clock domain has four clock sources for selection, i.e., the internal system clock sources XTAL_CLK, RC_FAST_CLK, PLL_F160M_CLK and the external clock source (PAD_CLK_TX), as shown in Figure 61.5-1. Clock sources can be selected by configuring HP_SYS_CLKRST_PARLIO_TX_CLK_SRC_SEL. The clock can be divided by configuring HP_SYS_CLKRST_PARLIO_TX_CLK_DIV_NUM, HP_SYS_CLKRST_PARLIO_TX_CLK_DIV_DENOMINATOR, and HP_SYS_CLKRST_PARLIO_TX_CLK_DIV_NUMERATOR. The clock division factor can be configured up to $(2^8 - 1)$.

The input clock of the TX and RX units can be inverted. The operating clock of the TX and RX units can also be inverted before being output to IO. The TX and RX units also support clock gating of the output clock.

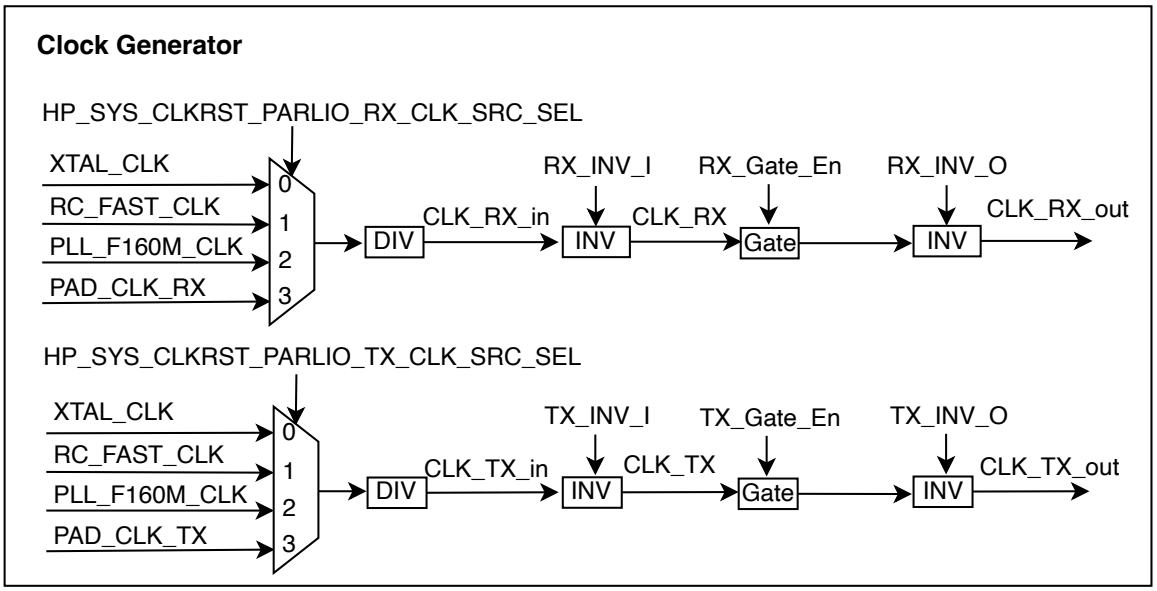


Figure 61.5-1. PARLIO Clock Generation

61.5.2 Clock & Reset Restriction

Due to the versatility of PARLIO, the PAD clocks (PAD_CLK_TX/RX) of PARLIO may come from different masters (external devices or internal clock sources). These clocks might be either free-running clock or not. If the clock is not free-running, some internal control signals of PARLIO cannot process CDC, so there are certain restrictions during the operation.

1. During the reset of the asynchronous FIFO, it takes two clock cycles to synchronize within AXI clock domain and Core clock domain. Therefore, if the reset of AXI clock domain is performed with a clock that is not free-running, the reset synchronization must be performed two clock cycles in advance. The specific operation is shown in the table below.

Table 61.5-1. Operations to Reset AXI Clock Domain with Clock Restrictions

Clock Restriction		Specific Operation
The Current Frame	The Next Frame	
Free-running clock	Not free-running clock	Users can reset the next frame transfer before switching to the clock that is not free-running. After the reset is completed, users can switch the clock.
Not free-running clock	Free-running clock	The next frame can be reset freely. Users only need to ensure that there is an interval of two clock cycles between the reset and the start of the transfer.
Not free-running clock	Not free-running clock	If the next frame transfer needs to be reset, users need to first switch to the internal free-running clock, and then switch to the actual clock after the reset is completed.

2. Due to the restrictions caused by a clock that is not free-running, PARL_IO_RX_START and PARL_IO_TX_START cannot perform CDC processing. Therefore, it is necessary to wait until PARL_IO_RX_START and PARL_IO_TX_START are stable before starting the data transfer. Otherwise, the transfer might enter a metastable state.

   Here are the specific operation steps for the RX unit:

   - Clear HP_SYS_CLKRST_PARLIO_RX_CLK_EN to turn off RX Core clock domain;
   - Write 1 to PARL_IO_RX_START;
   - Set HP_SYS_CLKRST_PARLIO_RX_CLK_EN to turn on RX Core clock domain;
   - Operate the external device to start sending data;
   - Clear HP_SYS_CLKRST_PARLIO_RX_CLK_EN to turn off RX Core clock domain;
   - Write 0 to PARL_IO_RX_START.

   Here are the specific operation steps for the TX unit:

   - Clear HP_SYS_CLKRST_PARLIO_TX_CLK_EN to turn off TX Core clock domain;
   - Write 1 to PARL_IO_TX_START;
   - Set HP_SYS_CLKRST_PARLIO_TX_CLK_EN to turn on TX Core clock domain;
   - Operate the external device to start receiving data;
   - Clear HP_SYS_CLKRST_PARLIO_TX_CLK_EN to turn off TX Core clock domain;
   - Write 0 to PARL_IO_TX_START.

3. Reset should follow the requirements below:
   - The clock reset during the chip start-up should follow the sequence below:
     (a) Reset APB clock domain;
     (b) Reset GDMA clock domain;
     (c) Reset Core clock domain.
   - Inter-frame transfer requires Core clock domain reset and async FIFO reset.

61.5.3 Master-Slave Mode

The TX and RX units can function as both master and slave.

When the TX unit serves as master, it is necessary to set the internal free-running clock as the clock source. The TX unit drives TXD on the rising edge of the clock.

When the TX unit functions as a slave device, there are three scenarios, as shown in the table below.

Table 61.5-2. Requirements for TX Unit Operating as Slave with Clock Restrictions

Clock Restriction		Requirement
Clock Sent by Master	Clock Waveform	
Free-running clock	Any waveform	There is no requirement for the sampling edge of the master clock.
Not free-running clock	Positive waveform (Figure 61.5-2)	The master clock should sample TXD at the falling edge.
Not free-running clock	Negative waveform (Figure 61.5-3)	The master device should invert the original clock and convert it to the waveform as Figure 61.5-2 shows before output.

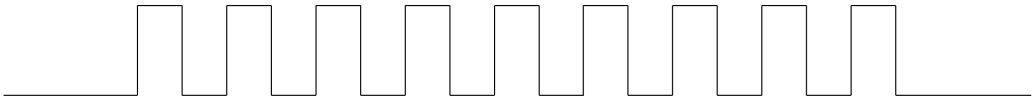


Figure 61.5-2. Positive Waveform

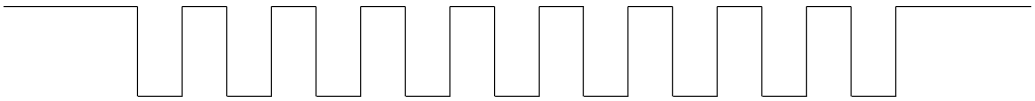


Figure 61.5-3. Negative Waveform

When the RX unit serves as the master, it is required to set the clock source as the internal free-running clock. The RX unit drives RXD on the rising edge of the clock.

When the RX unit functions as the slave, there are three scenarios, as shown in the table below.

Table 61.5-3. Requirements for RX Unit Operating as Slave with Clock Restrictions

Clock Restriction		Requirement
Clock Sent by Master	Clock Waveform	
Free-running clock	Any waveform	There is no requirement for the sampling edge of the master clock, and the valid data is subject to the external enable signal.
Not free-running clock	Positive waveform (Figure 61.5-2)	It is required for the master device to drive the data at the rising edge and the RX unit to sample the data at the falling edge (i.e., to inverse the master clock).
Not free-running clock	Negative waveform (Figure 61.5-3)	It is required for the master device to drive the data at the falling edge and the RX unit to sample the data at the rising edge (i.e., to use the original master clock).

61.5.4 Receive Modes of the RX Unit

PARLIO supports eight receive modes, which can be divided into three major categories according to the enable signal:

- Level Enable mode: data received is enabled by the external signal level;
- Pulse Enable mode: data received is enabled by the external signal pulse;
- Software Enable mode: the enable signal of data received can be configured by users directly.

The RX unit also supports inverse of the external enable signal. If the external enable signal is active-low, users can enable the function by setting PARL_IO_RX_EXT_EN_INV to switch to the corresponding receive mode introduced as follows.

61.5.4.1 Level Enable Mode

Figure 61.5-4 shows the Level Enable mode. In this mode, an active level on the external enable signal must be aligned with valid data. Since the external level enable signal occupies one IO pin, there are at most 15 IO pins left usable for RXD.

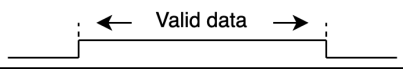
Mode	Sub-mode	Description
LEVEL_ENABLE	\	***signal level high*** 

Figure 61.5-4. Level Enable Mode for RX Unit

61.5.4.2 Pulse Enable Mode

Pulse Enable mode can be divided into six sub-modes depending on the pulse active level and its alignment with valid data. For detailed classification, see Figure 61.5-5.

Sub-modes 1 ~ 4 all contain start pulse and end pulse. The difference lies in whether start pulse and end pulse are aligned with valid data.

Sub-modes 5 ~ 6 only contain start pulse and the end of valid data is signaled by configuring PARL_IO_RX_BITLEN.

Since the external pulse enable signal occupies one IO pin, there are at most 15 IO pins left usable for RXD. However, in sub-mode 5 and sub-mode 6, as the data is considered valid after the pulse's first edge and before the pulse's last edge, the enable signal IO pin can serve as a data IO pin at the same time. Therefore, there are 16 IO pins usable for RXD in these two sub-modes.

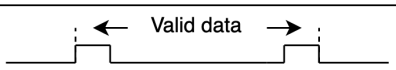
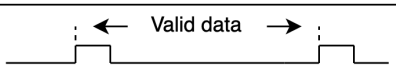
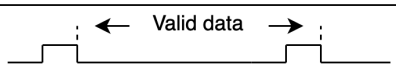
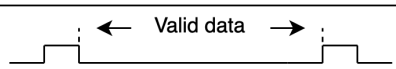
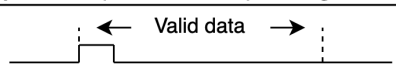
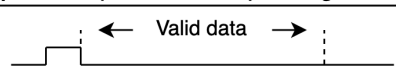
Mode	Sub-mode	Description
PULSE_ENABLE	sub-mode1	***pulse start(data bit include) && pulse end(data bit include)*** 
	sub-mode2	***pulse start(data bit include) && pulse end(data bit exclude)*** 
	sub-mode3	***pulse start(data bit exclude) && pulse end(data bit include)*** 
	sub-mode4	***pulse start(data bit exclude) && pulse end(data bit exclude)*** 
	sub-mode5	***pulse start(data bit include) && length end*** 
	sub-mode6	***pulse start(data bit exclude) && length end*** 

Figure 61.5-5. Sub-Modes of Pulse Enable Mode for RX Unit

61.5.4.3 Software Enable Mode

The enable signal in Software Enable mode is determined by the internal configuration register. If users switch to this mode, the receive will only be activated when both PARL_IO_RX_SW_EN and PARL_IO_RX_START are set to 1.

Since the enable signal does not occupy IO pins on the interface, there are at most 16 IO pins usable by the RXD. Due to the differences of clock domains, the enable signal cannot be aligned with valid data. Thus, the validity of data needs to be identified by the valid clock edge. In this case, the RX Core clock needs to be aligned with valid data.

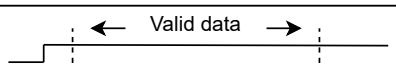
Mode	Sub-mode	Description
SW_ENABLE	/	/ 

Figure 61.5-6. Software Enable Mode for RX Unit

61.5.5 RX Unit GDMA SUC EOF Generation

The RX unit generates a GDMA SUC EOF signal to indicate the end of current frame transfer and sends it to the GDMA interface. GDMA SUC EOF can be generated by an external enable signal or triggered by the internally configured bit length.

- When GDMA SUC EOF is generated by the internally configured bit length, there is no restriction on the receive mode selection. However, PARL_IO_RX_BITLEN must be configured. If the configured value of PARL_IO_RX_BITLEN is less than the actual received data, the GDMA SUC EOF will be triggered in advance. In this case, the RX unit stops reading data from the FIFO, but the FIFO continues to receive external data until an RX_FIFO_WOVF_INT interrupt is triggered.
- When GDMA SUC EOF is generated by the external enable signal, only sub-modes 1 and 3 of Pulse Enable mode can be selected. In this mode, the transfer is not affected by the value of PARL_IO_RX_BITLEN, and the transferred data length of the frame is not limited.

61.5.6 TX Unit EOF Generation

The TX unit generates an EOF signal and triggers the TX_EOF_INT interrupt to indicate the end of current frame transfer. The TX EOF signal can be generated by a GDMA-configured EOF signal or triggered by the internally configured bit length.

- When TX EOF is generated by the internally configured bit length, PARL_IO_TX_BITLEN must be configured. If the configured value of PARL_IO_TX_BITLEN is less than the actual transmitted data, the TX EOF signal will be triggered in advance.
- When TX EOF is generated by a GDMA-configured EOF signal, the transfer is not affected by the value of PARL_IO_TX_BITLEN, and the transferred data length of the frame is not limited. The transfer does not end until the GDMA-configured EOF signal is received.

61.5.7 RX Unit Timeout

The RX unit supports the receive timeout. If the RX FIFO has not been receiving valid data for a long time, the timeout will be triggered. In such case, a GDMA ERR EOF signal will be generated and sent to the GDMA interface to indicate the end of the receiving. Configure PARL_IO_RX_TIMEOUT_THRES to set the timeout threshold.

The timeout function is enabled by default and can be disabled by users. The upper threshold of the configurable timeout is $(2^{16} - 1)$ cycles of GDMA clock domain, and the lower threshold depends on the relative frequency relationship between GDMA clock domain and RX Core clock domain. It is recommended to set a relatively large value for PARL_IO_RX_TIMEOUT_THRES to avoid undesired GDMA ERR EOF signals.

61.5.8 Unlimited Length Data Transmission of TX Unit

By default, the end of transmission for the TX unit is determined by the configured value of PARL_IO_TX_BITLEN. When the transmitted data reaches the configured value, the system generates a GDMA SUC EOF interrupt signal, indicating the completion of the current data frame transmission. This approach limits the maximum length of a single frame transmission to the bit width defined by PARL_IO_TX_BITLEN, preventing it from achieving unlimited length data transmission.

To enable larger single-frame transmissions, set PARL_IO_TX_EOF_GEN_SEL to 1. In this mode, the hardware uses the EOF configured in the GDMA inlink list as the end-of-transmission signal for a single frame. As a result, data transmission is no longer constrained by the bit width of PARL_IO_TX_BITLEN, allowing for significantly larger data frames.

61.5.9 TX Chip Select Function

To emulate the LCD I8080 protocol, the TX unit is equipped with an output chip select (CS) signal, which is active low. Configure PARL_IO_TX_CS_START_DELAY and PARL_IO_TX_CS_STOP_DELAY to control the phase relationship between the GDMA output data and the CS signal. Note that when clock gating is enabled and the CS signal is used as the control source for clock gating, configuring these two registers not only implements the basic function, but also allows control over the phase relationship between the output clock and the CS signal.

61.5.10 Output Clock Gating of TX Unit

The TX unit supports output clock gating. The clock gating function is disabled by default, and can be enabled by software via setting PARL_IO_TX_GATING_EN.

Currently, the most significant bit (MSB) of TXD can be configured to be controlled by one of two sources: GDMA output data or the chip select signal (CS). Set PARL_IO_TX_VALID_OUTPUT_EN to 0 to select the GDMA output data as the control source. In this case, the TX unit must be configured with the maximum bit width, because when the control source is the GDMA output data, the gating signal is fixed to the MSB of TXD. The clock signal can be toggled when this bit is at a high level. Note that when selecting GDMA output data as the control source, if the peripheral is in an idle state, the clock gating will be controlled by the MSB of the user-configured bus idle value. For configuration guidelines, refer to Section 61.5.11 Bus Idle Value of TX Unit. When the gating function is enabled while using GDMA output data as the control source, there are at most 15 IO pins left usable for TXD as the gating signal occupies one IO.

Set PARL_IO_TX_VALID_OUTPUT_EN to 1 to select the chip select signal (CS) as the control source. In this case, there is no limit to the bit width.

61.5.11 Bus Idle Value of TX Unit

The TX unit is regarded as in idle state when it is not transmitting data. It supports a configurable bus idle value.

The bus idle value is 0x0 by default, and its maximum configurable value is 0xFF. Note that the configured idle value should not conflict with other enabled functions. For example, when the MSB of TXD is used as the valid signal, users should avoid configuring the MSB of the idle value as 1.

61.5.12 Data Transfer in a Single Frame

The RX unit and the TX unit transfer data in the unit of bits, i.e., a single frame transfers 1 bit of data at least.

When the RX unit generates GDMA EOF signals through bit length, the maximum length of the single-frame transmission is $(2^{19} - 1)$ bits. When the RX unit generates GDMA EOF signals through the enable signal from an external device, there is no limit to the amount of bits of the single-frame transmission.

When the TX unit generates EOF signals through bit length, the maximum length of a single frame transmission is $(2^{19} - 1)$ bits. When the TX unit generates EOF signals through GDMA-configured EOF signals, there is no limit to the amount of bits of the single-frame transmission.

Since PARLIO transfers data in the unit of bytes on the GDMA side, it will process the IO data when it is not aligned with the bytes.

- When receiving data, PARLIO will automatically pad 0 to the high bits of the data stored in memory via GDMA to make it aligned with bytes.
- When sending data, PARLIO will truncate the data retrieved from memory according to the configured value of PARL_IO_TX_BITLEN. Data exceeding the value will not be sent.

When the configured data bus width is 2/4/8/16-bit, the bit length must be configured as a multiple of the corresponding bus width.

61.5.13 Bit Reversal in One Byte

The sequence of data within one byte can be reversed when data bus width is 1/2/4-bit. Taking the RX unit as an example, when the configured bus width is 2 bits, the data needs to be packed into one byte before being written into the RX FIFO.

Presume that the original bit sequence is:

$$\{\{b_0, b_1\}, \{b_2, b_3\}, \{b_4, b_5\}, \{b_6, b_7\}\}$$

If the bit reversal is enabled, the sequence will be reordered to:

$$\{\{b_6, b_7\}, \{b_4, b_5\}, \{b_2, b_3\}, \{b_0, b_1\}\}$$

61.6 Interrupts

ESP32-P4's PARLIO module can generate the following interrupt signals that will be sent to the Interrupt Matrix.

- PARL_IO_TX_INTR
- PARL_IO_RX_INTR

There are several internal interrupt sources from PARLIO that can generate the above interrupt signals. The interrupt sources from PARLIO are listed with their trigger conditions and the resulted interrupt signals in Table 61.6-1.

Table 61.6-1. PARLIO's Internal Interrupt Sources

Internal Interrupt Source	Trigger Condition	Interrupt Signal
TX_FIFO_REMPTY_INT	TX FIFO is empty, indicating possible error in the data sent by TX	PARL_IO_TX_INTR
RX_FIFO_WOVF_INT	RX FIFO is full, indicating possible error in the data received by RX	PARL_IO_RX_INTR
TX_EOF_INT	TX finishes sending a complete frame of data	PARL_IO_TX_INTR

Note:
For definitions of *interrupt*, *interrupt signal*, *interrupt source*, and their correlations, please refer to Chapter 13 *Interrupt Matrix* > Section 13.2 *Interrupt Terminology in ESP32-P4*.

Each interrupt source can be configured by a common set of registers that are described in Section *Interrupt Configuration Registers*. The specific registers can be found in Section 61.9 *Register Summary*.

61.7 Programming Procedures

61.7.1 Data Receiving Operation Process

This section introduces the programming procedure for receiving data through RX unit. To receive parallel data from IO pins that are connected to external devices and store the data in the internal memory, perform the following procedure. For a detailed description of the clock and reset operation restrictions in the RX unit, refer to Section 61.5.2.

1. Reset the RX unit. For specific reset scenarios and sequences, refer to Section 61.5.2.
2. Set PARL_IO_RX_FIFO_WOVF_INT_CLR and PARL_IO_RX_FIFO_WOVF_INT_ENA.
3. Select the RXD IO pins. If a PAD clock is used, the clock IO pin also needs to be configured.
4. Select the clock source and divide the clock by configuring clock registers.
5. Turn off the clock of RX Core clock domain.
6. Select the receive mode and enable functions required as described in Sections 61.3 and 61.5.
7. Configure GDMA inlink list.
8. Set PARL_IO_RX_REG_UPDATE to synchronize the register signals.
9. Set PARL_IO_RX_START.
10. Turn on the clock of RX Core clock domain.
11. Operate the external device to start sending data.
12. Poll the GDMA SUC EOF interrupt.
13. Clear the GDMA SUC EOF interrupt.
14. Turn off the clock of RX Core clock domain.
15. Clear PARL_IO_RX_START.

61.7.2 Data Transmitting Operation Process

This section introduces the programming procedure for transmitting data through TX unit. To transmit parallel data from internal memory to the IO pins that are connected to external devices, perform the following procedure. For detailed description of the clock and reset operation restrictions for the TX unit, refer to Section 61.5.2.

1. Reset the TX unit. For specific reset scenarios and sequences, refer to Section 61.5.2.

2. Set PARL_IO_TX_FIFO_REMPTY_INT_CLR, PARL_IO_TX_EOF_INT_CLR, PARL_IO_TX_FIFO_REMPTY_INT_ENA, and PARL_IO_TX_EOF_INT_ENA in sequence.
3. Select the TXD IO pins. If a PAD clock is used, the clock IO PAD also needs to be configured.
4. Select the clock source and divide the clock by configuring clock registers.
5. Turn off the clock of TX Core clock domain.
6. Select the functions required as described in Section 61.5.
7. Configure GDMA outlink list.
8. Poll the PARL_IO_TX_READY.
9. Set PARL_IO_TX_START.
10. Turn on the clock of TX Core clock domain.
11. Operate the external device to start receiving data.
12. Poll the PARL_IO_TX_EOF_INT_ST.
13. Set PARL_IO_TX_EOF_INT_CLR.
14. Turn off the clock of TX Core clock domain.
15. Clear PARL_IO_TX_START.

61.8 Application Examples

This section introduces some PARLIO application examples and their detailed operation process. All peripherals used in the examples are from ESP series chips and can work with PARLIO to form a complete data path.

Note:
The data paths constructed in the examples may not be the optimal. For example, users can use the SPI peripherals on two identical ESP chips to complete the peer-to-peer transfer in real case instead of using PARLIO to work with SPI. However, these examples demonstrate the flexibility of the PARLIO interface to a certain extent.

61.8.1 Co-working with SPI

In this example, external SPI sends data as a master device and PARLIO RX unit receives data as a slave device, and at the same time, PARLIO TX sends data as a master device and SPI receives data as a slave device, thus achieving a peer-to-peer serial data transfer.

- Follow the operation process below to achieve SPI transmit and PARLIO receive:
  1. Configure SPI clock.
  2. Configure SPI as the master device.
  3. Configure signal pins. Connect FSPICLK to PAD_CLK_RX, FSPICS0 to RXD[15], and FSPID to RXD[0].
  4. Write the data sent into the SPI buffer and configure the bit length of the data sent.

5. Set SPI_UPDATE to update the configured register value.
6. Reset PARLIO RX unit.
7. Configure PARLIO RX unit clock.
8. Turn off the PARLIO RX Core clock domain.
9. Configure PARLIO receive mode as sub-mode 1 of LEVEL Enable mode. Configure RX unit data bus width as 1 bit. Configure PARL_IO_RX_BITLEN according to the sending length of SPI. Set PARL_IO_RX_REG_UPDATE.
10. Configure PARLIO GDMA inlink list.
11. Set PARL_IO_RX_START.
12. Turn on the PARLIO RX Core clock domain.
13. Set SPI_USR to start transmitting data of SPI.
14. Poll GDMA SUC EOF interrupt.
15. Clear PARL_IO_RX_START.

- Follow the operation process below to achieve PARLIO transmit and SPI receive:

1. Configure SPI clock.
2. Configure SPI as the slave device.
3. Configure signal pins. Connect FSPICLK to PAD_CLK_TX, FSPICS0 to TXD[15], and FSPID to TXD[0].
4. Set SPI_RD_BIT_ORDER to invert the bit order.
5. Set SPI_UPDATE to update the configured register value.
6. Reset PARLIO TX unit.
7. Set PARL_IO_TX_EOF_INT_CLR and PARL_IO_TX_EOF_INT_ENA.
8. Configure PARLIO TX unit clock.
9. Turn off the clock of TX Core clock domain.
10. Configure data bus width as 1 bit. Write 1 to PARL_IO_TX_VALID_OUTPUT_EN. Configure PARL_IO_TX_BITLEN.
11. Configure GDMA outlink list.
12. Poll PARL_IO_TX_READY.
13. Write 1 to PARL_IO_TX_START.
14. Turn on the clock of TX Core clock domain.
15. Start data transfer.
16. Poll PARL_IO_TX_EOF_INT_ST.
17. Set PARL_IO_TX_EOF_INT_CLR.
18. Turn off the clock of TX Core clock domain.
19. Clear PARL_IO_TX_START.

61.8.2 Co-working with I2S

In this example, external I2S sends data as a master device and PARLIO RX unit receives data as a slave device. PARLIO supports the transmission of the I2S TDM MSB alignment standard and the TDM PCM standard. When the I2S transfer protocol is the TDM MSB alignment standard, it is required to configure the receive mode of PARLIO as Level Enable mode. When the I2S transfer protocol is the TDM PCM standard, it is required to configure the receive mode of PARLIO as the sub-mode 4 of Pulse Enable mode and set PARL_IO_RX_EXT_EN_INV.

This section takes the TDM PCM alignment standard as an example. The specific operation process is as follows:

1. Configure I2S clock.
2. Configure signal pins. Connect I2SO_BCK_out to PAD_CLK_RX, I2SO_WS_out to RXD[15], and I2SO_Data_out to RXD[0].
3. Configure I2S as the master device.
4. Configure the I2S TX data mode and channel mode required. Set I2S_TX_UPDATE.
5. Reset I2S TX unit and TX FIFO.
6. Enable I2S_TX_DONE_INT.
7. Configure I2S GDMA outlink list.
8. Set I2S_TX_STOP_EN.
9. Reset PARLIO RX unit.
10. Configure PARLIO RX unit clock.
11. Turn off PARLIO RX Core clock domain.
12. Configure PARLIO receive mode as sub-mode 4 of Pulse Enable mode and set PARL_IO_RX_EXT_EN_INV to configure the RX unit data bus width as 1 bit. Configure PARL_IO_TX_BITLEN according to the length of the data sent by I2S. Set PARL_IO_RX_REG_UPDATE.
13. Configure PARLIO GDMA inlink list.
14. Set PARL_IO_RX_START.
15. Turn on PARLIO RX Core clock domain.
16. Set I2S_TX_START to start transmitting data.
17. Poll I2S_TX_DONE_INT.
18. Poll GDMA SUC EOF interrupt.
19. Clear I2S_TX_START.
20. Clear PARL_IO_RX_START.

61.8.3 Co-working with LCD

In this example, PARLIO TX unit sends data as a master device and external LCD controller receives data as a slave device. The I8080/MOTO6800 format is used. 8-bit parallel data is transferred between devices. The

specific operation process is as follows:

1. Configure signal pins. Connect CLK_TX_out to LCD pixel clock PCLK, TXD[7:0] to LCD data input pin, TXD[8] to LCD CS pin, TXD[9] to LCD CD pin.
2. Reset PARLIO TX unit.
3. Set PARL_IO_TX_FIFO_REMPTY_INT_CLR, PARL_IO_TX_EOF_INT_CLR, PARL_IO_TX_FIFO_REMPTY_INT_ENA, and PARL_IO_TX_EOF_INT_ENA in sequence.
4. Configure PARLIO TX unit clock.
5. Turn off the clock of TX Core clock domain.
6. Configure data bus width as 16 bit. Configure PARL_IO_TX_BITLEN.
7. Configure GDMA outlink list. Note that the data sent in the linked list should conform to I8080/MOTO6800 format. The lower eight bits are valid parallel data. The 9th and 10th bits are respectively CS, CD. The MSB is the constant 1. The remaining bits are arbitrary values.
8. Poll PARL_IO_TX_READY.
9. Write 1 to PARL_IO_TX_START.
10. Turn on the clock of TX Core clock domain.
11. Start data transfer.
12. Poll PARL_IO_TX_EOF_INT_ST.
13. Set PARL_IO_TX_EOF_INT_CLR.
14. Turn off the clock of TX Core clock domain.
15. Clear PARL_IO_TX_START.

61.9 Register Summary

The addresses in this section are relative to Parallel IO Controller base address provided in Table 9.3-2 in Chapter 9 *System and Memory*.

The abbreviations given in Column **Access** are explained in Section *Access Types for Registers*.

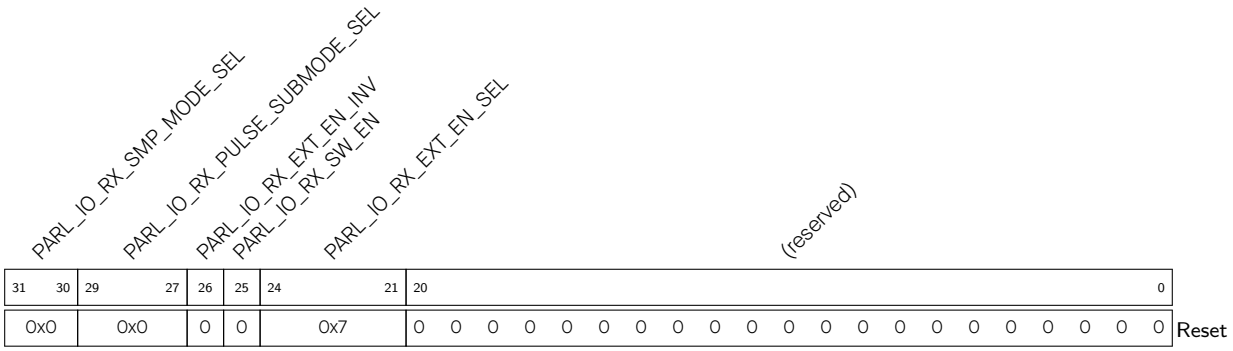
Name	Description	Address	Access
PARLIO RX Configuration Registers			
PARL_IO_RX_MODE_CFG_REG	PARLIO RX sampling mode configuration register	0x0000	R/W
PARL_IO_RX_DATA_CFG_REG	PARLIO RX data configuration register	0x0004	R/W
PARL_IO_RX_GENRL_CFG_REG	PARLIO RX general configuration register	0x0008	R/W
PARL_IO_RX_START_CFG_REG	PARLIO RX start configuration register	0x000C	R/W
PARLIO TX Configuration Registers			
PARL_IO_TX_DATA_CFG_REG	PARLIO TX data configuration register	0x0010	R/W
PARL_IO_TX_START_CFG_REG	PARLIO TX start configuration register	0x0014	R/W
PARL_IO_TX_GENRL_CFG_REG	PARLIO TX general configuration register	0x0018	R/W
PARLIO Configuration and Status Registers			
PARL_IO_FIFO_CFG_REG	PARLIO FIFO configuration register	0x001C	R/W
PARL_IO_REG_UPDATE_REG	PARLIO register update configuration register	0x0020	WT
PARL_IO_ST_REG	PARLIO module status register	0x0024	RO
PARLIO Interrupt Configuration and Status Registers			
PARL_IO_INT_ENA_REG	PARLIO interrupt enable signal configuration register	0x0028	R/W
PARL_IO_INT_RAW_REG	PARLIO interrupt raw signal status register	0x002C	R/SS/WTC
PARL_IO_INT_ST_REG	PARLIO interrupt signal status register	0x0030	RO
PARL_IO_INT_CLR_REG	PARLIO interrupt clear signal configuration register	0x0034	WT
PARLIO RX/TX Status Registers			
PARL_IO_RX_ST0_REG	PARLIO RX status register 0	0x0038	RO
PARL_IO_RX_ST1_REG	PARLIO RX status register 1	0x003C	RO
PARL_IO_TX_ST0_REG	PARLIO TX status register 0	0x0040	RO
PARLIO Clock Configuration Registers			
PARL_IO_RX_CLK_CFG_REG	PARLIO RX clock configuration register	0x0044	R/W
PARL_IO_TX_CLK_CFG_REG	PARLIO TX clock configuration register	0x0048	R/W
PARL_IO_CLK_REG	PARLIO clock configuration register	0x0120	R/W
PARLIO TX CS Configuration Register			
PARL_IO_TX_CS_CFG_REG	PARLIO TX CS configuration register	0x004C	R/W
PARLIO Version Register			
PARL_IO_VERSION_REG	Version control register	0x03FC	R/W

61.10 Registers

The addresses in this section are relative to Parallel IO Controller base address provided in Table 9.3-2 in Chapter 9 *System and Memory*.

For how to program reserved fields, please refer to Section *Programming Reserved Register Field*.

Register 61.1. PARL_IO_RX_MODE_CFG_REG (0x0000)



PARL_IO_RX_EXT_EN_SEL Configures the RX external enable signal from IO PAD. (R/W)

PARL_IO_RX_SW_EN Configures whether to enable data sampling by software.

0: Disable

1: Enable

(R/W)

PARL_IO_RX_EXT_EN_INV Configures whether to invert the external enable signal.

0: No effect

1: Invert

(R/W)

PARL_IO_RX_PULSE_SUBMODE_SEL Configures the RXD pulse sampling sub-mode.

0: Positive pulse start (data bit included) & Positive pulse end (data bit included)

1: Positive pulse start (data bit included) & Positive pulse end (data bit excluded)

2: Positive pulse start (data bit excluded) & Positive pulse end (data bit included)

3: Positive pulse start (data bit excluded) & Positive pulse end (data bit excluded)

4: Positive pulse start (data bit included) & Length end

5: Positive pulse start (data bit excluded) & Length end

(R/W)

PARL_IO_RX_SMP_MODE_SEL Configures the RXD sampling mode.

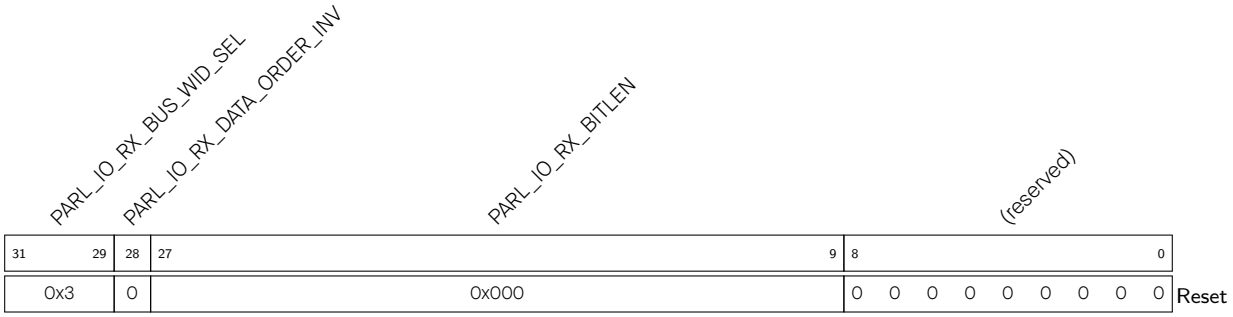
0: External Level Enable mode

1: External Pulse Enable mode

2: Internal Software Enable mode

(R/W)

Register 61.2. PARL_IO_RX_DATA_CFG_REG (0x0004)



PARL_IO_RX_BITLEN Configures the expected bit number of RXD. (R/W)

PARL_IO_RX_DATA_ORDER_INV Configures whether to invert the bit order of one byte sent from RX FIFO to GDMA. (R/W)

PARL_IO_RX_BUS_WID_SEL Configures the RXD bus width.

0: 1 bit

1: 2 bits

2: 4 bits

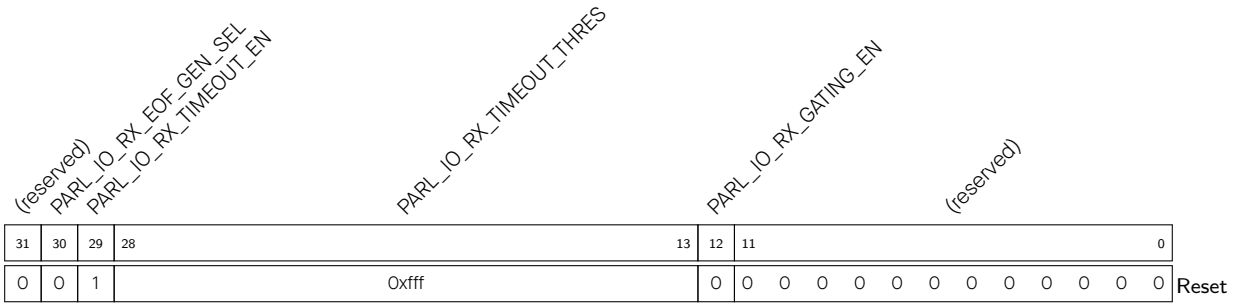
3: 8 bits

4: 16 bits

5 ~ 15: Invalid and will incur error

(R/W)

Register 61.3. PARL_IO_RX_GENRL_CFG_REG (0x0008)



PARL_IO_RX_GATING_EN Configures whether to enable the clock gating of the RX output clock.

0: Disable

1: Enable

(R/W)

PARL_IO_RX_TIMEOUT_THRES Configures the threshold of the RX timeout counter. (R/W)

PARL_IO_RX_TIMEOUT_EN Configures whether to enable the timeout function to generate GDMA ERR EOF.

0: Disable

1: Enable

(R/W)

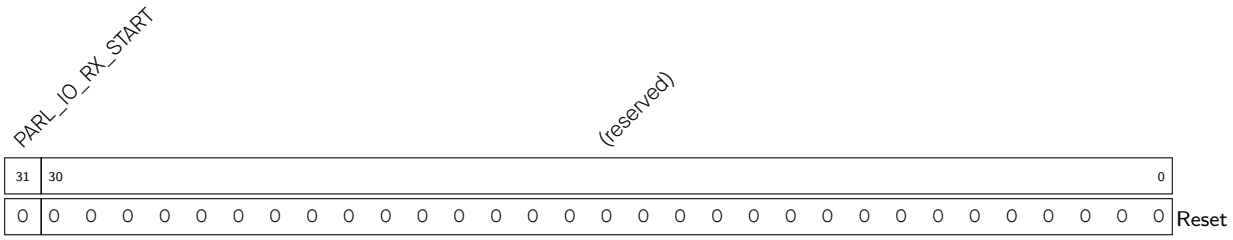
PARL_IO_RX_EOF_GEN_SEL Configures the generation mechanism of GDMA SUC EOF.

0: Generate GDMA SUC EOF by the configured data bit length

1: Generate GDMA SUC EOF by the external enable signal

(R/W)

Register 61.4. PARL_IO_RX_START_CFG_REG (0x000C)



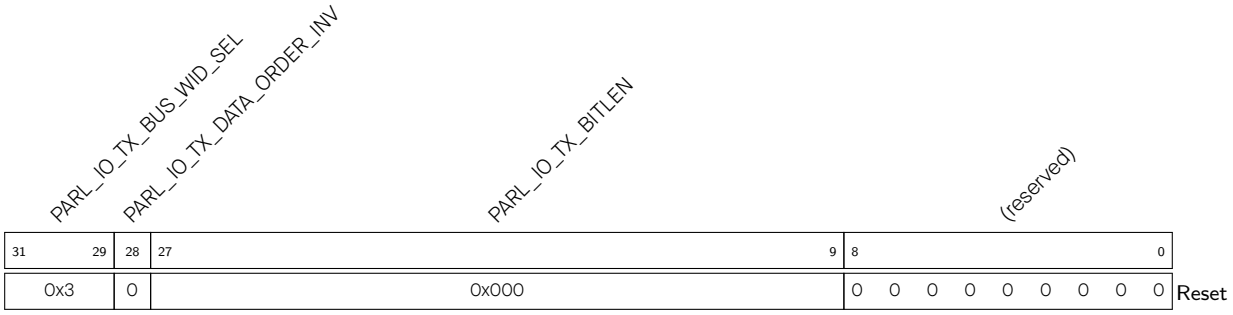
PARL_IO_RX_START Configures whether to start RX data sampling.

0: No effect

1: Start

(R/W)

Register 61.5. PARL_IO_TX_DATA_CFG_REG (0x0010)



PARL_IO_TX_BITLEN Configures the expected bit number of TXD. (R/W)

PARL_IO_TX_DATA_ORDER_INV Configures whether to invert the bit order of one byte sent from TX FIFO to IO data.

0: No effect

1: Invert

(R/W)

PARL_IO_TX_BUS_WID_SEL Configures the TXD bus width.

0: 1 bit

1: 2 bits

2: 4 bits

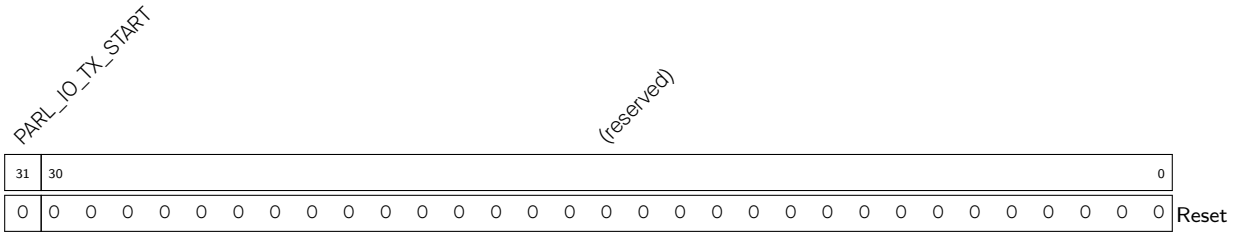
3: 8 bits

4: 16 bits

5 ~ 15: Invalid and will incur error

(R/W)

Register 61.6. PARL_IO_TX_START_CFG_REG (0x0014)



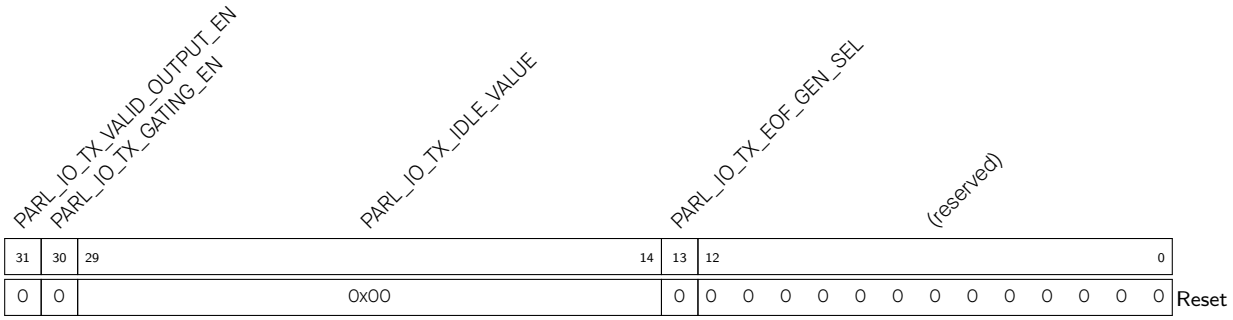
PARL_IO_TX_START Configures whether to start TX data transmission.

0: No effect

1: Start

(R/W)

Register 61.7. PARL_IO_TX_GENRL_CFG_REG (0x0018)



PARL_IO_TX_EOF_GEN_SEL Configures the generation mechanism of TX EOF.

0: Generate TX EOF by the configured data bit length

1: Generate TX EOF by the GDMA EOF signal.

(R/W)

PARL_IO_TX_IDLE_VALUE Configures the data value on TX bus in idle state. (R/W)

PARL_IO_TX_GATING_EN Configures whether to enable the clock gating of the TX output clock.

0: Disable

1: Enable

(R/W)

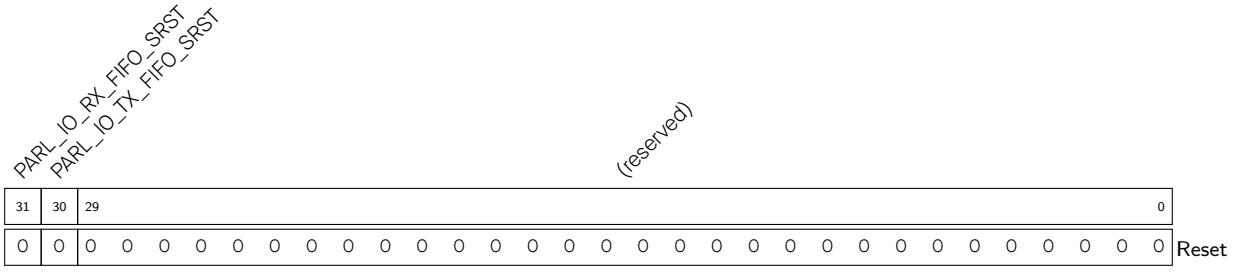
PARL_IO_TX_VALID_OUTPUT_EN Configures whether to select DMA output data or the chip select signal (CS) as the control source of TXD MSB.

0: Select the DMA output data as the control source

1: Select the chip select signal (CS) as the control source

(R/W)

Register 61.8. PARL_IO_FIFO_CFG_REG (0x001C)



PARL_IO_TX_FIFO_SRST Configures whether to reset async FIFO in the TX unit.

0: No effect

1: Reset

(R/W)

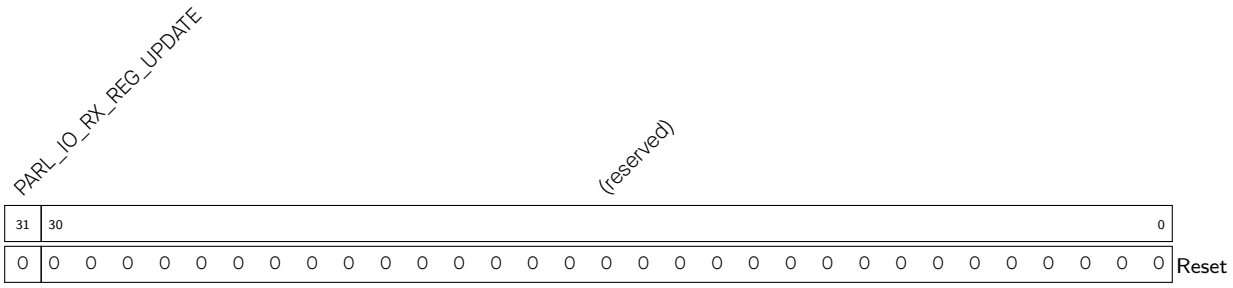
PARL_IO_RX_FIFO_SRST Configures whether to reset async FIFO in the RX unit.

0: No effect

1: Reset

(R/W)

Register 61.9. PARL_IO_REG_UPDATE_REG (0x0020)



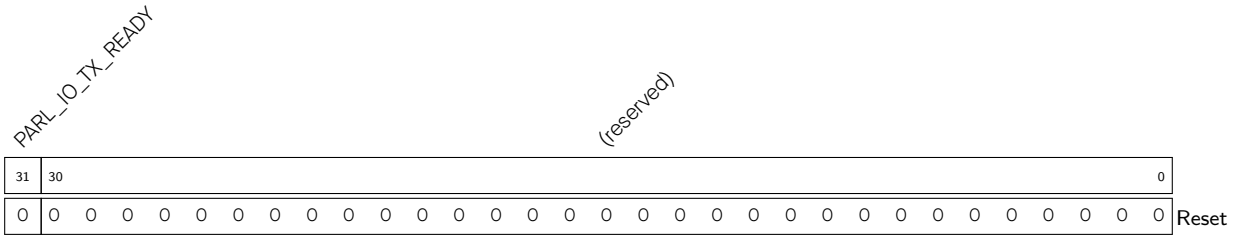
PARL_IO_RX_REG_UPDATE Configures whether to update RX register configuration.

0: No effect

1: Update

(WT)

Register 61.10. PARL_IO_ST_REG (0x0024)



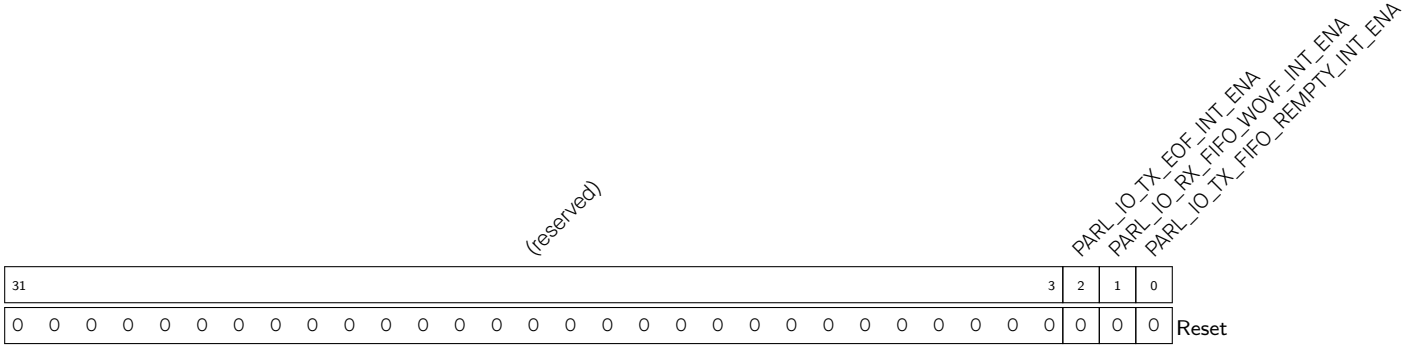
PARL_IO_TX_READY Represents whether TX is ready to transmit data.

0: Not ready

1: Ready

(RO)

Register 61.11. PARL_IO_INT_ENA_REG (0x0028)

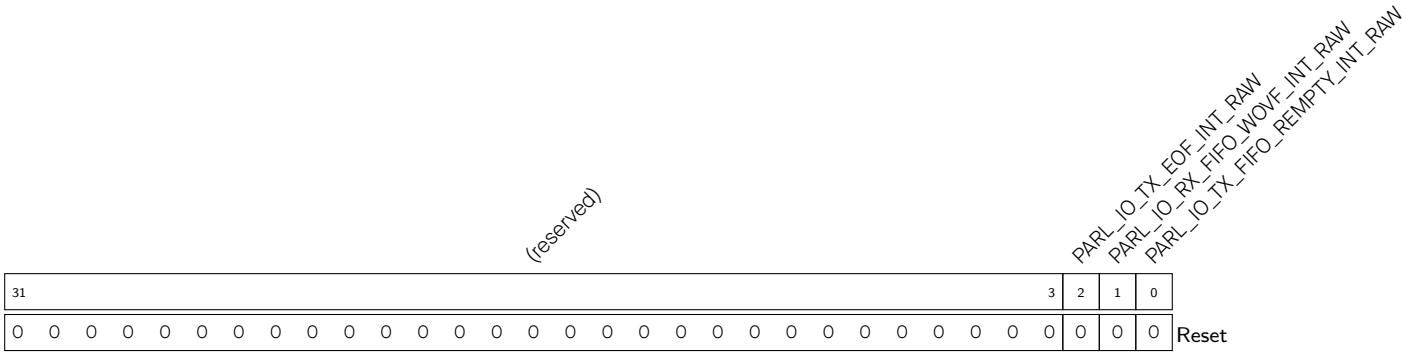


PARL_IO_TX_FIFO_REMPTY_INT_ENA Write 1 to enable TX_FIFO_REMPTY_INT. (R/W)

PARL_IO_RX_FIFO_WOVF_INT_ENA Write 1 to enable RX_FIFO_WOVF_INT. (R/W)

PARL_IO_TX_EOF_INT_ENA Write 1 to enable TX_EOF_INT. (R/W)

Register 61.12. PARL_IO_INT_RAW_REG (0x002C)

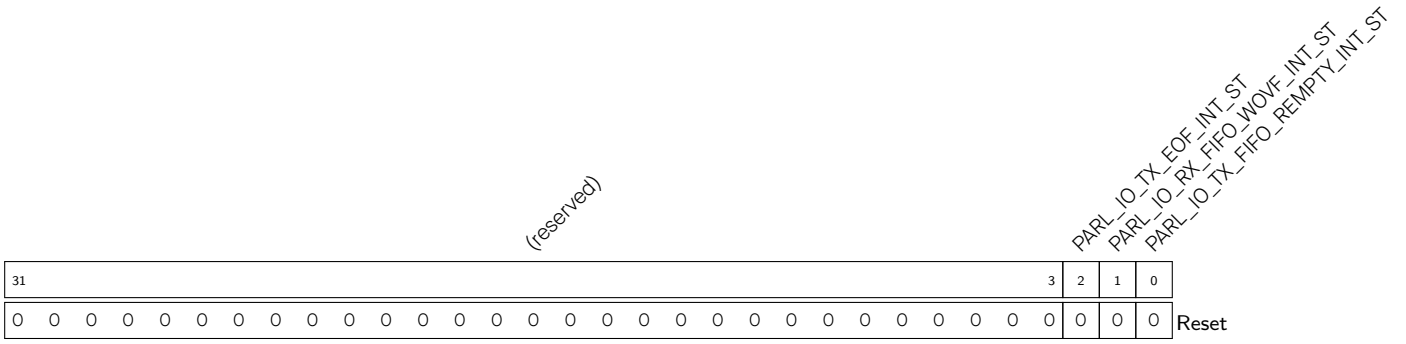


PARL_IO_TX_FIFO_REMPTY_INT_RAW The raw interrupt status of TX_FIFO_REMPTY_INT. (R/WTC/SS)

PARL_IO_RX_FIFO_WOVF_INT_RAW The raw interrupt status of RX_FIFO_WOVF_INT. (R/WTC/SS)

PARL_IO_TX_EOF_INT_RAW The raw interrupt status of TX_EOF_INT. (R/WTC/SS)

Register 61.13. PARL_IO_INT_ST_REG (0x0030)

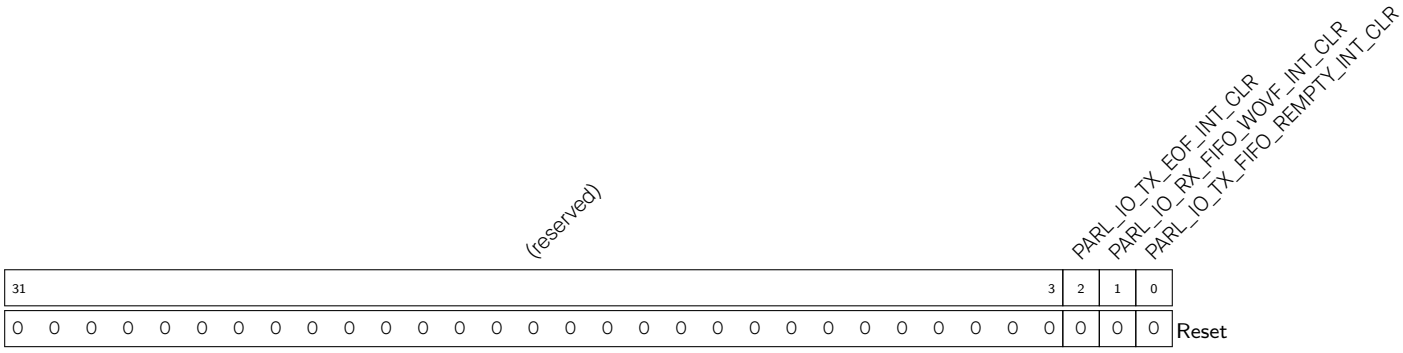


PARL_IO_TX_FIFO_REMPTY_INT_ST The masked interrupt status of TX_FIFO_REMPTY_INT. (RO)

PARL_IO_RX_FIFO_WOVF_INT_ST The masked interrupt status of RX_FIFO_WOVF_INT. (RO)

PARL_IO_TX_EOF_INT_ST The masked interrupt status of TX_EOF_INT. (RO)

Register 61.14. PARL_IO_INT_CLR_REG (0x0034)

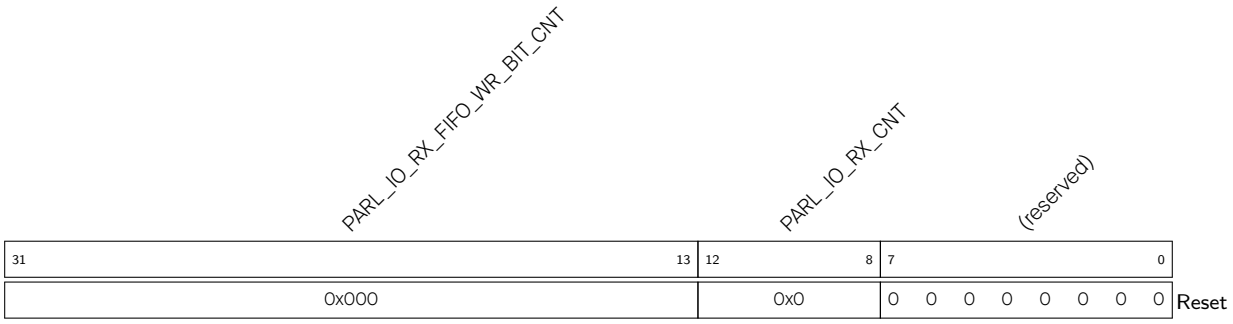


PARL_IO_TX_FIFO_REMPTY_INT_CLR Write 1 to clear TX_FIFO_REMPTY_INT. (WT)

PARL_IO_RX_FIFO_WOVF_INT_CLR Write 1 to clear RX_FIFO_WOVF_INT. (WT)

PARL_IO_TX_EOF_INT_CLR Write 1 to clear TX_EOF_INT. (WT)

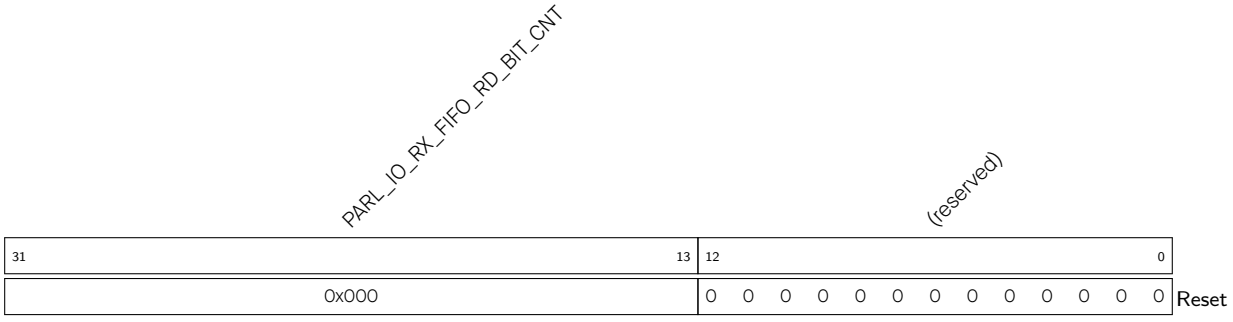
Register 61.15. PARL_IO_RX_ST0_REG (0x0038)



PARL_IO_RX_CNT Represents the clock cycle number of reading the RX FIFO. (RO)

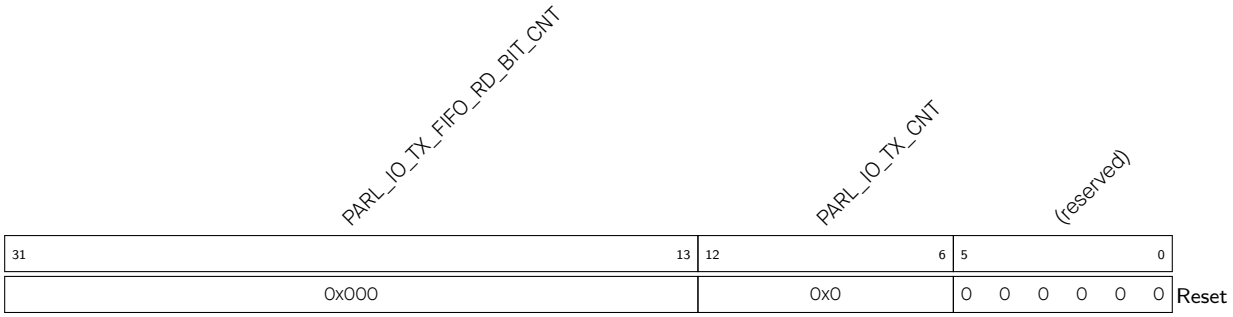
PARL_IO_RX_FIFO_WR_BIT_CNT Represents the bit number currently written into the RX FIFO. (RO)

Register 61.16. PARL_IO_RX_ST1_REG (0x003C)



PARL_IO_RX_FIFO_RD_BIT_CNT Represents the bit number currently read from the RX FIFO. (RO)

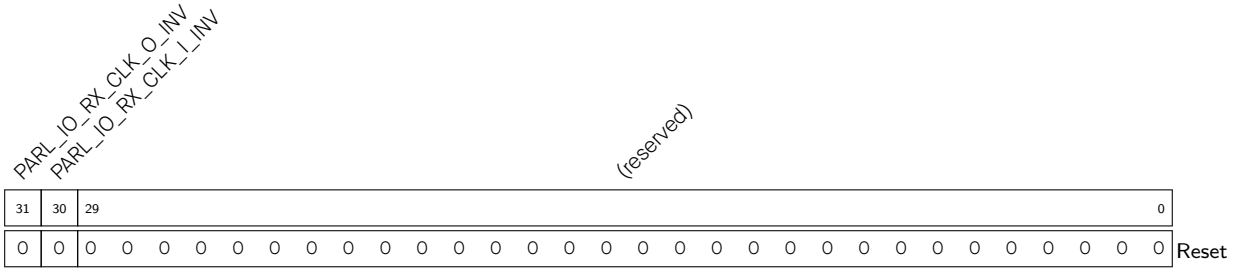
Register 61.17. PARL_IO_TX_ST0_REG (0x0040)



PARL_IO_TX_CNT Represents the cycle number of reading the TX FIFO. (RO)

PARL_IO_TX_FIFO_RD_BIT_CNT Represents the bit number currently read from the TX FIFO. (RO)

Register 61.18. PARL_IO_RX_CLK_CFG_REG (0x0044)



PARL_IO_RX_CLK_I_INV Configures whether to invert the RX input Core clock.

0: No effect

1: Invert

(R/W)

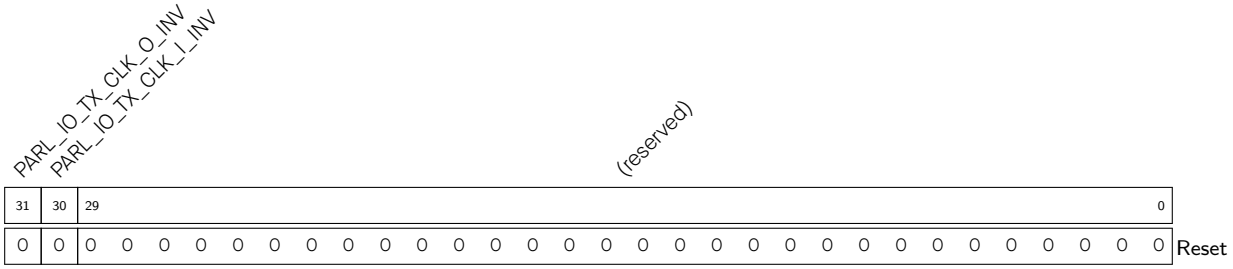
PARL_IO_RX_CLK_O_INV Configures whether to invert the RX output Core clock.

0: No effect

1: Invert

(R/W)

Register 61.19. PARL_IO_TX_CLK_CFG_REG (0x0048)



PARL_IO_TX_CLK_I_INV Configures whether to invert the TX input Core clock.

0: No effect

1: Invert

(R/W)

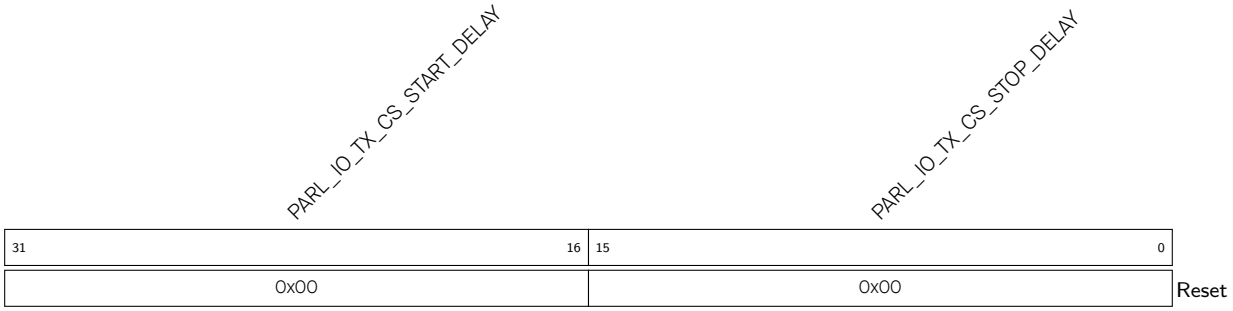
PARL_IO_TX_CLK_O_INV Configures whether to invert the TX output Core clock.

0: No effect

1: Invert

(R/W)

Register 61.20. PARL_IO_TX_CS_CFG_REG (0x004C)



PARL_IO_TX_CS_STOP_DELAY Configures the delay between the end of TX data transmission and the rising edge of TX_CS_O.

Unit: TX Core clock cycle

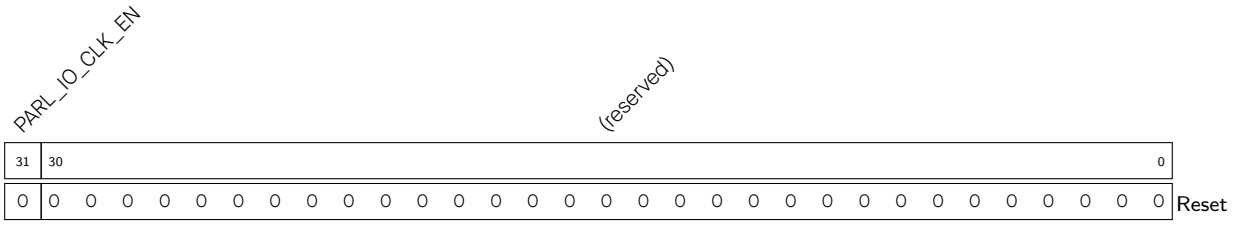
(R/W)

PARL_IO_TX_CS_START_DELAY Configures the delay between the falling edge of TX_CS_O and the start of TX data transmission.

Unit: TX Core clock cycle

(R/W)

Register 61.21. PARL_IO_CLK_REG (0x0120)



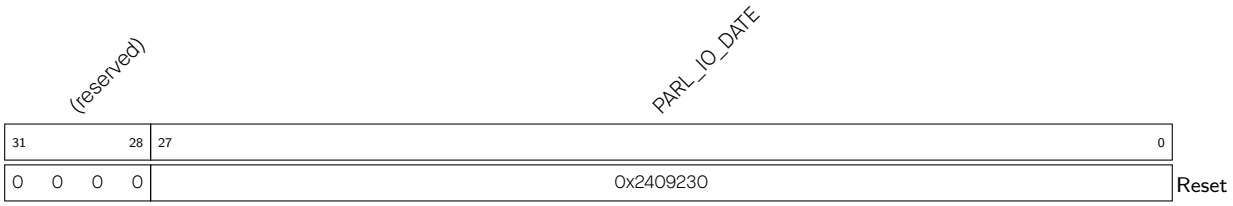
PARL_IO_CLK_EN Register clock gating.

0: Enable the register clock only when reading or writing registers

1: Always enable the register clock

(R/W)

Register 61.22. PARL_IO_VERSION_REG (0x03FC)



PARL_IO_DATE Version control register. (R/W)

Chapter 62

BitScrambler

The ESP32-P4 has an extensive amount of DMA-capable peripherals. These can move data from memory to an external device, and vice versa, without any interference from the CPU. This only works if the external device needs or emits the data in question in the same format as the software expects it: if not, the CPU needs to rewrite the format of the data. Examples include a need to swap bytes, reverse bytes, and shift the data left or right.

As bitwise operations tend to be fairly CPU-expensive and the purpose of DMA is to not use the CPU in the transfer, ESP32-P4 includes two BitScramblers, which are dedicated peripherals to change the format of data in between memory and the peripheral. A BitScrambler is capable of performing the aforementioned operations, but as a flexible programmable state machine, it is capable of more advanced things as well. One of the two BitScramblers is dedicated to memory-to-peripheral (or memory-to-memory) transfers, the other one is dedicated to peripheral-to-memory transfers.

62.1 Introduction

In ESP32-P4, a DMA-capable peripheral is one that can read data directly from memory and/or write data directly to memory using a DMA channel. The format this data is read/written in is dependent on the peripheral and generally has only a small amount of flexibility. For most DMA-capable peripherals in ESP32-P4, it is possible to attach a BitScrambler to any of these paths for additional flexibility.

When a BitScrambler is attached to a peripheral, it gets the chance to modify any data that flows down a DMA channel associated with that peripheral. Data written to the DMA channel will appear on the BitScrambler input, and what the BitScrambler writes to its output will flow down the DMA channel. The BitScrambler contains instruction memory where instructions on how to route the data between input and output can be stored. As such, by writing an appropriate BitScrambler program, the way the data is rewritten by the BitScrambler can be controlled.

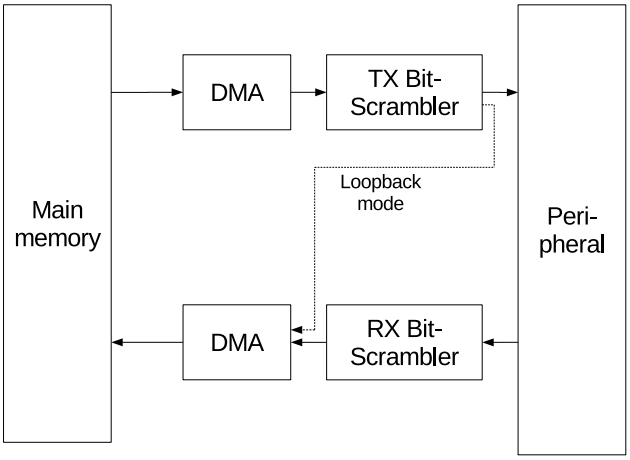


Figure 62.1-1. BitScrambler System Context Diagram

62.2 Feature List

The BitScrambler has the following features:

- Two BitScramblers, one for RX (peripheral-to-memory), one for TX (memory-to-peripheral)
- Support for memory-to-memory transfers
- Processing up to 32 bits per DMA clock period
- Data path controlled by a BitScrambler program stored in instruction memory
- Input registers able to read 0, 8, 16, or 32 bits per clock cycle
- Output registers:
  - Able to write 0, 8, 16, or 32 bits per clock cycle
  - Data sources for output register bits: 64 bits of input data, two counters, LUT RAM data, data output of last cycle, comparators
  - With some restrictions, each of the 32 output register bits can come from any bit on the data sources
- 8 x 257-bit instruction memory, for storing eight instructions, controlling control flow and the data path
- 2048 bytes of lookup table (LUT) memory, configurable as various word widths

62.3 Architectural Overview

The BitScrambler can be separated into a data path, a control path, and the registers that the CPU can use to program them. The data path moves bits between the incoming DMA stream, internal registers, and the outgoing DMA stream. The control path parses the instructions in the instruction memory to control the data path, as well as handle program flow.

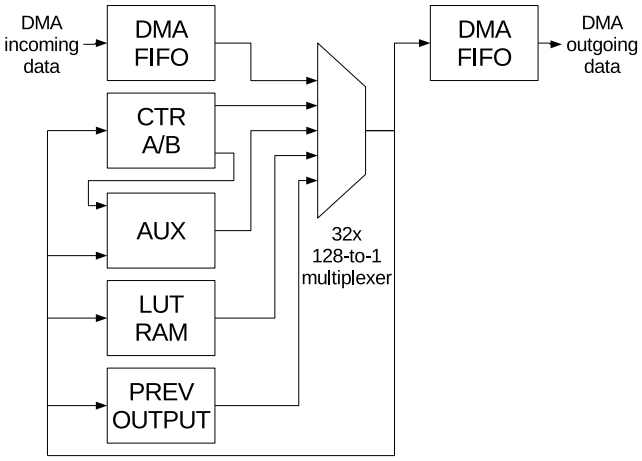


Figure 62.3-1. BitScrambler Data Path Diagram

62.3.1 Data Path

The data path of the BitScrambler is intended to take data from the incoming DMA stream, process it according to the program code stored in instruction memory, and write it to the outgoing DMA stream. Data is read from the incoming DMA stream into a 64-bit register. At the end of an instruction cycle, as specified in the instruction, N bits (with N being 0, 8, 16, or 32) are read from the DMA stream. The data in the register is then shifted toward the LSB of the register with the least significant N bits disappearing. Finally, the read data appears as the N most significant bits. Optionally, on startup, the BitScrambler will automatically read the first 64 bits into this register.

On the other side of the BitScrambler, data is deposited into a 32-bit output register. At the end of the instruction cycle, depending on what the instruction specifies, the least significant 0, 8, 16, or 32 bits will be written to the outgoing DMA stream.

The BitScrambler derives its name from the fact that it can put input bits in any random position in the output, and this is achieved by 32 x 128-to-1 multiplexers. For each of the 32 output register bits, there is one multiplexer that selects the input signals from one of the data sources, dependent on the output of the BitScrambler control path. These data sources are:

- 64 bits, shifted in from the input FIFO (i.e., the left DMA FIFO block in Figure 62.3-1). As described before, data from the incoming DMA stream is deposited here. In order to facilitate iterating over input bits in a loop, an instruction can enable relative addressing. In this addressing mode, any mux sourcing a bit from the input FIFO register will actually get the bit offset by the counter A register (i.e., CTR A in Figure 62.3-1).
- 32 bits that were sent to the output in the last cycle (i.e., PREV OUTPUT block in Figure 62.3-1). Data sent to the output register will appear on this register one clock cycle later. This allows the BitScrambler to 'remember' data over multiple clock cycles: by outputting a bit to the output register, even if that bit subsequently is not sent to the output FIFO, the next clock cycle can still access it by selecting from this register. Bits can be remembered longer-term by selecting bits from this register for output again, which makes them appear here in the subsequent cycle.
- 8, 16, or 32 bits that are the output of LUT RAM. The address used to select the data is bit 16 to 24, 25 or 26 of the output data of the last cycle, assuming the LUT width is 32, 16 or 8 bits, respectively. The LUT RAM is a part of the BitScrambler that takes an address, looks it up in its memory, and outputs the data

located at that specific address. The LUT RAM can be filled when the BitScrambler is initialized, but the BitScrambler itself does not have the ability to modify it. Depending on requirements, the LUT RAM can be initialized in the following modes:

- 512 x 32-bit words
- 1024 x 16-bit words
- 2048 x 8-bit words

The LUT RAM can be used to do translations of data, e.g. using a calibration curve for a given sensor. It can also be used to implement mathematical functions, with the address input split into two or more input variables and the data output used as the output of the function; the memory should be loaded with the output of the function for each of the input values.

- 32 bits from 2 x 16-bit counters (i.e., CTR A/B block in Figure 62.3-1). This register contains two 16-bit counters, named 'A' and 'B'. These can be loaded, incremented and decremented via control path instructions.
- 30 bits from comparators between counter B and the previous data on the output (PREV OUT in Figure 62.3-1; the 30 bits described here are part of the AUX block there). These allow the result of comparisons being used for output or for program flow. This can be useful in e.g. decoding run-length encoded streams, or generating PWM signals.
- 2 bits, one fixed-high, one fixed-low (i.e., part of AUX in Figure 62.3-1). These are useful if a protocol needs an always-high or always-low bit in the output that does not depend on the input data.

Note that some sources cannot be accessed at the same time. Specifically, a given instruction cannot source data from both the high 32 bits of the input FIFO source register as well as the counter registers. When data is read from the LUT, for a LUT that is set to a width of N, the top N bits of both the input FIFO register as well as the top N bits of the counter register become unavailable.

62.3.2 Control Path

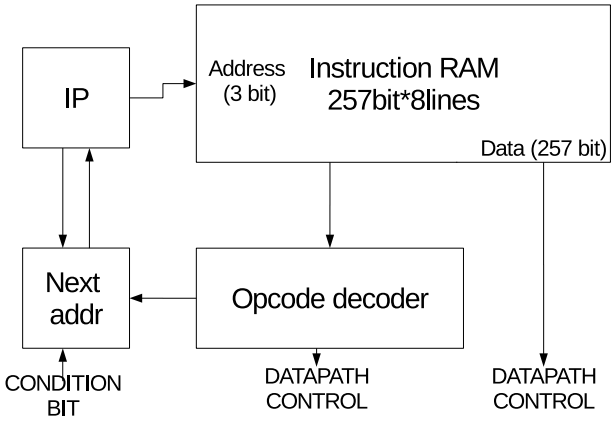


Figure 62.3-2. BitScrambler Control Path Diagram

The BitScrambler behavior is governed by a program stored in program RAM. The program RAM has space for up to eight instructions, each being 257 bits in size. An instruction configures the data path for the clock cycle the instruction runs in, as well as contains opcodes that control program flow.

A BitScrambler program is similar to a microprocessor program, in that each cycle the BitScrambler will execute the next instruction in instruction memory, unless the opcode tells it to specifically jump to another location. Concretely, this behaviour is implemented using an instruction pointer (IP, see Figure 62.3-2), which points to the currently executing instruction. Opcodes can also affect the counter registers.

The bits of the instruction that configure the data path consist of settings for the 32 muxes, as well as some bits to select the availability of the counter bits or LUT output. It also can put the muxes into relative mode, where if they get a bit from the DMA FIFO, the position of that bit is offset by counter B.

62.4 Functional Description

The BitScrambler is a flexible device, with its behavior controlled by the program loaded into it as well as the configuration registers that set several global BitScrambler behaviors as well. As such, a BitScrambler program consists of the contents of the 257-bit x 8 instruction RAM as well as the settings for various configuration registers. Both are described below.

62.4.1 Instructions

The 257 bits that make up an instruction are structured as shown in Table 62.4-1.

Table 62.4-1. Instruction Structure

Bit	Field	Bit	Field
0 - 6	CTL_MUX_SEL_0	133 - 139	CTL_MUX_SEL_19
7 - 13	CTL_MUX_SEL_1	140 - 146	CTL_MUX_SEL_20
14 - 20	CTL_MUX_SEL_2	147 - 153	CTL_MUX_SEL_21
21 - 27	CTL_MUX_SEL_3	154 - 160	CTL_MUX_SEL_22
28 - 34	CTL_MUX_SEL_4	161 - 167	CTL_MUX_SEL_23
35 - 41	CTL_MUX_SEL_5	168 - 174	CTL_MUX_SEL_24
42 - 48	CTL_MUX_SEL_6	175 - 181	CTL_MUX_SEL_25
49 - 55	CTL_MUX_SEL_7	182 - 188	CTL_MUX_SEL_26
56 - 62	CTL_MUX_SEL_8	189 - 195	CTL_MUX_SEL_27
63 - 69	CTL_MUX_SEL_9	196 - 202	CTL_MUX_SEL_28
70 - 76	CTL_MUX_SEL_10	203 - 209	CTL_MUX_SEL_29
77 - 83	CTL_MUX_SEL_11	210 - 216	CTL_MUX_SEL_30
84 - 90	CTL_MUX_SEL_12	217 - 223	CTL_MUX_SEL_31
91 - 97	CTL_MUX_SEL_13	224 - 249	OPCODE
98 - 104	CTL_MUX_SEL_14	250 - 251	CTL_READ_IN
105 - 111	CTL_MUX_SEL_15	252 - 253	CTL_WR_OUT
112 - 118	CTL_MUX_SEL_16	254	CTL_MUX_REL
119 - 125	CTL_MUX_SEL_17	255	CTL_CTR_SRC_SEL
126 - 132	CTL_MUX_SEL_18	256	CTL_LUT_SRC_SEL

The meanings of these fields are as follows:

- CTL_MUX_SEL_n: This 7-bit field selects the source of bit n in the output register. See Table 62.4-3 for details.

- OPCODE: These bits encode an opcode. For the bit pattern of this field, please refer to Table 62.4-4.
- CTL_READ_IN: This selects the number of bits that are read from the input FIFO at the end of the instruction:
  - 0: Do not read any data
  - 1: Read 8 bits
  - 2: Read 16 bits
  - 3: Read 32 bits
- CTL_WR_OUT: This selects the number of bits that are written from the output register to the output FIFO at the end of the instruction:
  - 0: Do not write any data
  - 1: Write 8 bits
  - 2: Write 16 bits
  - 3: Write 32 bits
- CTL_MUX_REL, CTL_CTR_SRC_SEL and CTR_LUT_SRC_SEL: Select which bits are available to select using the CTL_MUX_SEL bits.
  - CTRL_CTR_SRC_SEL (Control Counter Source Select) makes the bits from counter A and B available
  - CTL_LUT_SRC_SEL (Control LUT Source Select) makes the output value of the LUT RAM available. See Table 62.4-3 for details.
  - When CTL_MUX_REL (Control Mux Relative) is 1, the BitScrambler applies the value of counter A as an offset to any CTL_MUX_SEL_n value that is 63 or below. The exact behaviour depends on CTR_SRC_SEL, as detailed in Table 62.4-2.

Table 62.4-2. Effective CTL_MUX_SEL_n values when CTL_MUX_REL is 1

Condition	Effective value of CTL_MUX_SEL
CTL_MUX_SEL_n >= 64	CTL_MUX_SEL (unchanged)
CTL_MUX_SEL_n < 64 and CTL_CTR_SRC_SEL = 0	(CTL_MUX_SEL_n + CTRA) & 63
CTL_MUX_SEL_n < 31 and CTL_CTR_SRC_SEL = 1	(CTL_MUX_SEL_n + CTRA) & 31
32 <= CTL_MUX_SEL_n < 64 and CTL_CTR_SRC_SEL = 1	((CTL_MUX_SEL_n + CTRA) & 31) + 32

Of these fields, most are data path configuration items. The field that affects the control path is OPCODE, although CTL_MUX_REL, CTL_CTR_SRC_SEL, and CTR_LUT_SRC_SEL may influence how some of the opcodes work by adjusting the source selection for the IF/IFN opcodes.

CTL_MUX_SEL_n: These fields, one for each bit in the output register, select the source for that particular bit. The specific bit chosen is partially dependent on the settings of the CTL_MUX_REL, CTL_CTR_SRC_SEL, and CTR_LUT_SRC_SEL. Given CTL_MUX_SRC_n has value N, the bit is selected as described in Table 62.4-3. Note that the source selection for IF/IFN opcodes is affected in the same way.

Table 62.4-3. CTL_MUX_SEL Values

Bit	Description
0-31	The bit is sourced from bit N in the incoming FIFO register.
32-63	The source of the bit depends on the setting of CTL_CTR_SRC_SEL and CTR_LUT_SRC_SEL: If CTL_CTR_SRC_SEL = 0: the bit is sourced from bit N in the incoming FIFO register. If CTL_CTR_SRC_SEL = 1: the bit is sourced from bit (N-32) of the counter registers. Specifically, if N is 32 to 47, the bit is sourced from bit (N-32) of counter A and if N is 48 to 63, the bit is sourced from bit (N-48) of counter B. If CTL_LUT_SRC_SEL = 1: Depending on the width of the LUT (8, 16, or 32 bits), a read of the top 8, 16, or 32 bits will return data from the LUT rather than what CTL_CTR_SRC_SEL selects. Specifically, given a LUT width of N, 63 will return LUT output bit (N-1), 62 will return LUT output bit (N-2) and so on.
64	CounterB[7:0] =< last[7:0]
65	CounterB[7:0] > last[7:0]
66	CounterB[7:0] = last[7:0]
67	CounterB[7:0] =< last[15:8]
68	CounterB[7:0] > last[15:8]
69	CounterB[7:0] = last[15:8]
70	CounterB[7:0] =< last[23:16]
71	CounterB[7:0] > last[23:16]
72	CounterB[7:0] = last[23:16]
73	CounterB[7:0] =< last[31:24]
74	CounterB[7:0] > last[31:24]
75	CounterB[7:0] = last[31:24]
76	CounterB[15:8] =< last[7:0]
77	CounterB[15:8] > last[7:0]
78	CounterB[15:8] = last[7:0]
79	CounterB[15:8] =< last[15:8]
80	CounterB[15:8] > last[15:8]
81	CounterB[15:8] = last[15:8]
82	CounterB[15:8] =< last[23:16]
83	CounterB[15:8] > last[23:16]
84	CounterB[15:8] = last[23:16]
85	CounterB[15:8] =< last[31:24]
86	CounterB[15:8] > last[31:24]
87	CounterB[15:8] = last[31:24]
88	CounterB[15:0] =< last[15:0]
89	CounterB[15:0] > last[15:0]
90	CounterB[15:0] = last[15:0]
91	CounterB[15:0] =< last[31:16]
92	CounterB[15:0] > last[31:16]
93	CounterB[15:0] = last[31:16]
94	Always 0
95	Always 1
96-127	The bits selected here are the same as the bits on the output register at the end of the previous cycle.

The BitScrambler recognizes six opcodes, which are encoded in the opcode fields as shown in Table 62.4-4.

Table 62.4-4. Instruction Opcode

Bit	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
LOOP	1	c	tgt			end_val																ctr_add				
ADD	0	c	h	l	0	0	0	0	0	0	ctr_add															
IF	0	0	tgt			0	0	0	0	1	0	0	0	0	0	0	0	0	0	src						
IFN	0	0	tgt			0	0	0	1	0	0	0	0	0	0	0	0	0	0	src						
LDCTD	0	c	h	l	0	0	0	0	1	1	ctr_set															
LDCTI	0	c	h	l	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

The fields are defined as follows:

- c: Configures which counter to use.
  - 0: counter A
  - 1: counter B
- end_val, ctr_add, ctr_set: 16-bit (or 5-bit) value. Please refer to the instruction descriptions below for the meanings of these fields.
- tgt: The location of an instruction. Range: 0 ~ 7.
- h: 1 if the upper 8 bits need to be written back to a counter.
- l: 1 if the lower 8 bits need to be written back to a counter.

Note that if an opcode does not specify H or L, it is assumed that the full 16 bits need to be written back and the h and l bits will both be 1.

These six opcodes affect the state of the BitScrambler as follows:

Table 62.4-5. LOOP Opcode

`LOOPc end_val, ctr_add, tgt`	
Loop on counter indicated by c, add ctr_add to the counter, and loop back to tgt until the counter reaches end_val. When that happens, reset the counter and fall through.	
Argument	c: Either A or B, indicates counter to use tgt: 3-bit target instruction addresses end_val: 16-bit value to compare the counter to ctr_add: 5-bit signed value to add to the counter

Pseudo-code	`if ((unsigned)ctrC < end_val) begin`
	`  IP <= tgt`
	`  ctr_c <= ctr_c + sign_extend(ctr_add)`
	`end else begin`
	`  IP <= IP + 1`
	`  ctr_c <= 0`
	`end`

Table 62.4-6. ADD Opcode

ADDc[H|L] ctr_add	
Add immediate value to counter indicated by c, and optionally write back only upper or lower 8 bits.	
Argument	c: Either A or B, indicate counter to use ctr_add: 16-bit value to add to counter If opcode is ADDcH, only most significant 8 bits are written back If opcode is ADDcL, only least-significant 8 bits are written back If opcode is ADDc, all 16 bits will be written back (h=1, l=1).

Pseudo-code	`tmp <= ctr_c + ctr_add`
	`if (h) begin`
	`  ctr_c[15:8] <= tmp[15:8]`
	`end`
	`if (l) begin`
	`  ctr_c[7:0] <= tmp[7:0]`
	`end`
	`IP <= IP + 1`

Note	The NOP (no operation) pseudo-op is encoded as 'ADDA 0'.

Table 62.4-7. IF Opcode

IF ctl_cond_src, tgt	
Jump to target if the indicated mux source bit is set.	
Argument	ctl_cond_src: Mux source (see CTR_MUX_SRC sources) tgt: 3-bit target instruction addresses

Pseudo-code	`if (ctl_cond_src) begin`
	`  IP <= tgt`
	`end else begin`
	`  IP <= IP + 1`
	`end`

Note	The JMP (always jump) pseudo-op is implemented as an 'IF 95,tgt' opcode, using the always-1 bit from the AUX bits.

Table 62.4-8. IFN Opcode

IFN ctl_cond_src, tgt	
Jump to target if the indicated mux source bit is NOT set (note: same as IF but condition is negated).	
Argument	ctl_cond_src: Mux source (see CTR_MUX_SRC sources) tgt: 3-bit target instruction addresses

Pseudo-code	`if (ctl_cond_src) begin`
	`  IP <= IP + 1`
	`end else begin`
	`  IP <= tgt`
	`end`

Table 62.4-9. LDCTD Opcode

LDCTDc[H|L] ctr_set	
Load counter direct (from immediate), and optionally only load high or low 8 bits of counter.	
Argument	c: Either A or B, indicate counter to use ctr_set: 16-bit value to set the counter to h: if opcode is LDCTDcH, only most significant 8 bits are set l: if opcode is LDCTDcL, only least-significant 8 bits are set If opcode is LDCTDc, all 16 bits are set (h=1, l=1)

Pseudo-code	`if (h) begin`
	`  ctr_c[15:8] <= ctr_set[15:8]`
	`end`
	`if (l) begin`
	`  ctr_c[7:0] <= ctr_set[7:0]`
	`end`
	`IP <= IP + 1`

Table 62.4-10. LDCTI Opcode

LDCTIc[H|L]	
Load counter indirect (from most significant 16 bits of mux output), and optionally only load high or low 8 bits of counter.	
Argument	c: Either A or B, indicate counter to use h: If opcode is LDCTIcH, only the most significant 8 bits are set l: If opcode is LDCTIcL, only the least significant 8 bits are set If opcode is LDCTIc, all 16 bits are set (h=1, l=1)

Pseudo-code	`if (h) begin`
	`  ctr_c[15:8] <= mux_out[31:24]`
	`end`
	`if (l) begin`
	`  ctr_c[7:0] <= mux_out[23:16]`
	`end`
	`IP <= IP + 1`

62.4.2 Configuration Registers

The BitScrambler is configured using a set of registers. Specifically, these registers are used to write a program into BitScrambler, write its LUT RAM, configure its behaviour, and start and stop BitScrambler operation. They are also used to attach a BitScrambler to a peripheral DMA path.

The ESP32-P4 has two BitScramblers: one can be placed in the TX path (where data is sent from the memory to a peripheral) while the other can be placed in a RX path (where data is sent from the peripheral to memory). The configuration registers for the two BitScramblers are the same, with the only difference being the TX path BitScrambler registers are prefixed by BITSCRAMBLER_TX_ while the RX path BitScrambler uses BITSCRAMBLER_RX_ prefixes.

Note:
For convenience, in this section we will refer to the TX path BitScrambler only; to get the RX path BitScrambler descriptions, simply swap the prefixes.

When modifying the input or output of a peripheral, the BitScrambler needs to be attached to the DMA path of that peripheral. To do this, write the appropriate value for the peripheral into the HP_SYSTEM_BITSCRAMBLER_PERI_TX_SEL or HP_SYSTEM_BITSCRAMBLER_PERI_RX_SEL field of the HP_SYSTEM_BITSCRAMBLER_PERI_SEL_REG register.

The BitScrambler can also run in memory-to-memory mode. For this, the BITSCRAMBLER_LOOP_MODE bit in BITSCRAMBLER_SYS_REG needs to be set to one. In this mode, the TX BitScrambler is put into loopback mode while the RX BitScrambler is unused and unavailable. The BitScrambler still needs to be attached to a peripheral (although the peripheral does not need to be configured); the DMA path for this peripheral will be unusable.

Neither the program memory nor the LUT RAM are accessible to the main processor directly. Rather, they need to be written indirectly, by setting an address in one register and then writing the data in a second register.

Specifically, for the LUT RAM, the address is written in the BITSCRAMBLER_TX_LUT_IDX field of BITSCRAMBLER_TX_LUT_CFG0_REG and the data is written to or read from BITSCRAMBLER_TX_LUT_CFG1_REG. The width of the LUT, as visible to the BitScrambler, can be configured using the BITSCRAMBLER_TX_LUT_MODE field. Note that when the host processor writes to the LUT in the aforementioned fashion, the addressing and data size needs to adhere to the configured LUT width.

Similarly, the instruction memory is written by setting the address up in BITSCRAMBLER_TX_INST_CFG0_REG and writing the data to or reading it from BITSCRAMBLER_TX_INST_CFG1_REG. As the BitScrambler instructions are 257 bits, one instruction needs to be written as 9 32-bit words (with the 257th bit being in the LSB of the 9th word). To achieve that purpose, the BITSCRAMBLER_TX_INST_CFG0_REG register is divided into two fields. The position of the instruction is written into the BITSCRAMBLER_TX_INST_IDX field while the word offset within the instruction is written to BITSCRAMBLER_TX_INST_POS.

As the BitScrambler has the ability to generate more or less data than it gets on the input, a BitScrambler-controlled DMA stream can end in one or two ways. The first is that the receiver (either memory in case of the RX BitScrambler, or the peripheral in case of the TX BitScrambler) stops accepting data because it is configured to only receive a limited number of bytes. This case is pretty simple: the BitScrambler will simply halt as it cannot write any more data. The other way is that the transmitter (the peripheral for the RX BitScrambler and the memory for the TX BitScrambler) stops sending data. This is a more complicated situation, as the BitScrambler may or may not be processing some data that needs to be written to the output.

In order to allow the BitScrambler to send out the data it is still processing, two methods have been devised to handle a certain amount of data after the input data stream has ended. Both depend on setting BITSCRAMBLER_RX_TAILING_BITS_REG to a certain value N:

- EOF-on-N-reads: After the input data stream ends, the BitScrambler reads N dummy bytes from the input. The value of these bytes is zero. After the Nth byte is read, the BitScrambler halts and the DMA stream ends.
- EOF-on-N-writes: After the input data stream ends, the BitScrambler is allowed to write N more bytes to

the output. During this time, any reads on the input stream return zero-value dummy bytes. After the N'th byte is written, the BitScrambler halts and the DMA stream ends.

To configure either mode, set the option bits as follows:

Table 62.4-11. Settings for EOF modes

Register	EOF-on-N-reads	EOF-on-N-writes
BITSCRAMBLER_TX_RD_DUMMY	1	1
BITSCRAMBLER_TX_FETCH_MODE	0	0
BITSCRAMBLER_TX_EOF_MODE	1	0
BITSCRAMBLER_TX_HALT_MODE	1	1

62.5 Programming Procedures

Note that these instructions are written for a TX BitScrambler, which is in the memory-to-peripheral path, or memory-to-memory path if in loopback mode. For the peripheral-to-memory path, the RX BitScrambler is used. Unless otherwise indicated, you can exchange "TX" for "RX" in these instructions to configure the RX BitScrambler.

1. Attach the BitScrambler to a peripheral by configuring HP_SYSTEM_BITSCRAMBLER_PERI_TX_SEL field in HP_SYSTEM_BITSCRAMBLER_PERI_SEL_REG register. Note that this step is required even in loopback mode.
2. Enable the BitScrambler by setting the BITSCRAMBLER_TX_ENA field in the BITSCRAMBLER_TX_CTRL_REG register.
3. Configure loopback mode if required. If loopback mode is needed, set BITSCRAMBLER_LOOP_MODE in the BITSCRAMBLER_SYS_REG register. Otherwise, clear it.
4. Load program into BitScrambler. For each instruction and each 32-bit word within the instruction, set the BITSCRAMBLER_TX_INST_IDX and BITSCRAMBLER_TX_INST_POS fields in the BITSCRAMBLER_TX_INST_CFG0_REG register, then write the word in the BITSCRAMBLER_TX_INST_CFG1_REG register. Note that the 257th bit is written as bit 0 into BITSCRAMBLER_TX_INST_CFG1_REG.
5. Load LUT into BitScrambler, if needed:
   (a) Configure BITSCRAMBLER_TX_LUT_MODE in BITSCRAMBLER_TX_LUT_CFG0_REG to set the LUT width
   (b) Write the address to the BITSCRAMBLER_TX_LUT_CFG0_REG
   (c) Write the data word or the byte itself to BITSCRAMBLER_TX_LUT_CFG1_REG

   Note that it is acceptable to change the LUT width after writing the data and before starting the BitScrambler.
6. Configure BitScrambler. Dependent on what the BitScrambler program expects, you may need to set or clear the various bits in the BITSCRAMBLER_TX_CTRL_REG register. You may also need to set BITSCRAMBLER_TX_TAILING_BITS_REG to an applicable value.

7. Start the BitScrambler by clearing the BITSCRAMBLER_TX_HALT bit in the BITSCRAMBLER_TX_CTRL_REG register.
8. Start DMA transaction. Please refer to Chapter 6 *GDMA Controller (GDMA-AHB, GDMA-AXI)* for more information about DMA subsystem and refer to the peripheral chapter if loopback mode is not used.
9. Wait until DMA transaction is done.
10. Halt the BitScrambler by setting the BITSCRAMBLER_TX_HALT in the BITSCRAMBLER_TX_CTRL_REG register.
11. Wait for halt state. The BitScrambler is halted when the BITSCRAMBLER_TX_IN_IDLE bit in the BITSCRAMBLER_TX_STATE_REG register is set.
12. Reset the FIFO by writing a 1 and then a 0 to the BITSCRAMBLER_TX_FIFO_RST bit in the BITSCRAMBLER_TX_CTRL_REG register.
13. The BitScrambler is ready for a new transaction with the current program and LUT configuration. Simply start from step 7 to do this.

62.6 Register Summary

The addresses in this section are relative to the Bit-scrambler base address provided in Table 9.3-2 in Chapter 9 *System and Memory*.

The abbreviations given in Column **Access** are explained in Section *Access Types for Registers*.

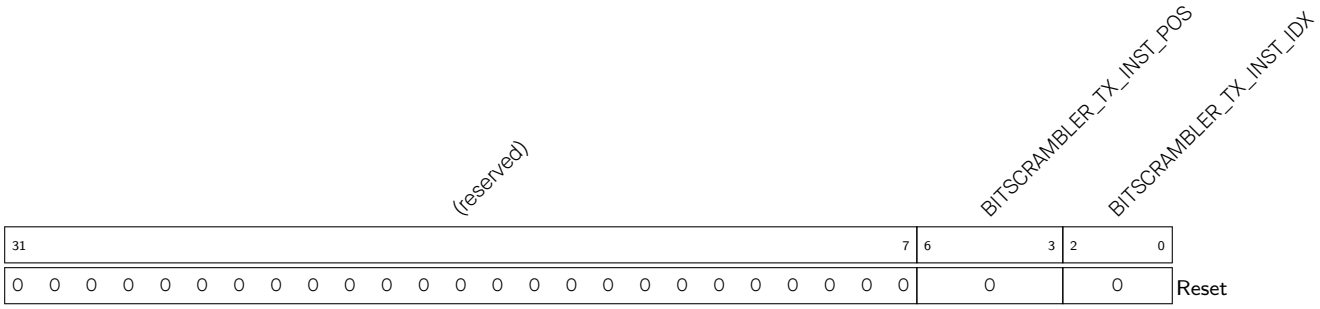
Name	Description	Address	Access
Control and configuration registers			
BITSCRAMBLER_TX_INST_CFG0_REG	BitScrambler TX core instruction memory address selection register	0x0000	R/W
BITSCRAMBLER_TX_INST_CFG1_REG	BitScrambler TX core instruction memory access register	0x0004	R/W
BITSCRAMBLER_RX_INST_CFG0_REG	BitScrambler RX core instruction memory address selection register	0x0008	R/W
BITSCRAMBLER_RX_INST_CFG1_REG	BitScrambler RX core instruction memory access register	0x000C	R/W
BITSCRAMBLER_TX_LUT_CFG0_REG	BitScrambler TX core LUT memory address selection register	0x0010	R/W
BITSCRAMBLER_TX_LUT_CFG1_REG	BitScrambler TX core LUT memory access register	0x0014	R/W
BITSCRAMBLER_RX_LUT_CFG0_REG	BitScrambler RX core LUT memory address selection register	0x0018	R/W
BITSCRAMBLER_RX_LUT_CFG1_REG	BitScrambler RX core LUT memory access register	0x001C	R/W
Configuration registers			
BITSCRAMBLER_TX_TAILING_BITS_REG	BitScrambler TX core extra data length register	0x0020	R/W
BITSCRAMBLER_RX_TAILING_BITS_REG	BitScrambler RX core extra data length register	0x0024	R/W
BITSCRAMBLER_TX_CTRL_REG	BitScrambler TX core control register	0x0028	varies
BITSCRAMBLER_RX_CTRL_REG	BitScrambler RX core control register	0x002C	varies
BITSCRAMBLER_SYS_REG	Loopback control register	0x00F8	R/W
Status registers			
BITSCRAMBLER_TX_STATE_REG	BitScrambler TX core status register	0x0030	varies
BITSCRAMBLER_RX_STATE_REG	BitScrambler RX core status register	0x0034	varies
Version register			
BITSCRAMBLER_VERSION_REG	Version control register	0x00FC	R/W

62.7 Registers

The addresses in this section are relative to the Bit-scrambler base address provided in Table 9.3-2 in Chapter 9 *System and Memory*.

For how to program reserved fields, please refer to Section Programming Reserved Register Field.

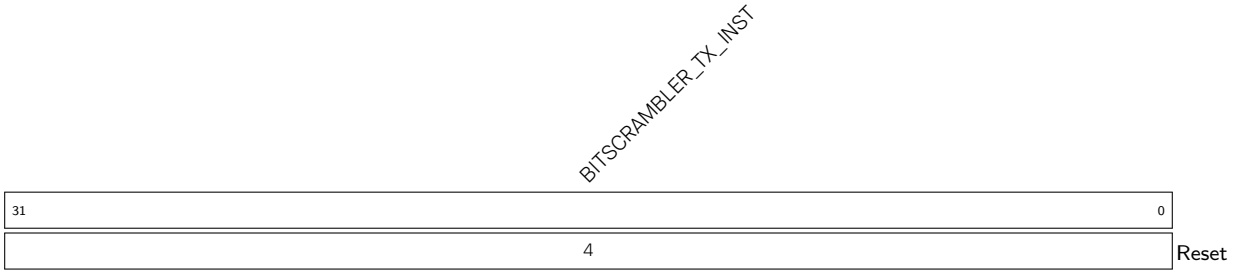
Register 62.1. BITSCRAMBLER_TX_INST_CFG0_REG (0x0000)



BITSCRAMBLER_TX_INST_IDX Configures the index of the BitScrambler instruction to be accessed via BITSCRAMBLER_TX_INST_CFG1_REG. (R/W)

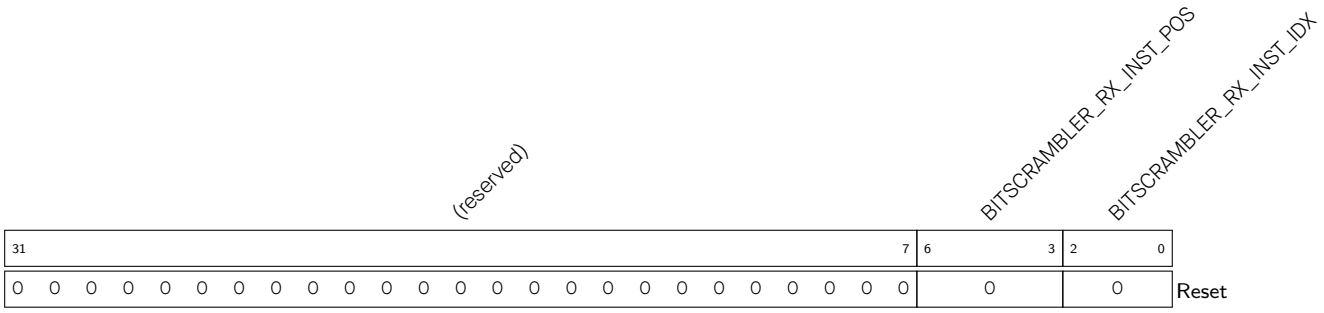
BITSCRAMBLER_TX_INST_POS Configures the offset into the 257-bit BitScrambler instruction (in 32-bit increments) to be accessed via BITSCRAMBLER_TX_INST_CFG1_REG. (R/W)

Register 62.2. BITSCRAMBLER_TX_INST_CFG1_REG (0x0004)



BITSCRAMBLER_TX_INST Configures the instruction to be accessed at the address specified by BITSCRAMBLER_TX_INST_CFG0_REG. (R/W)

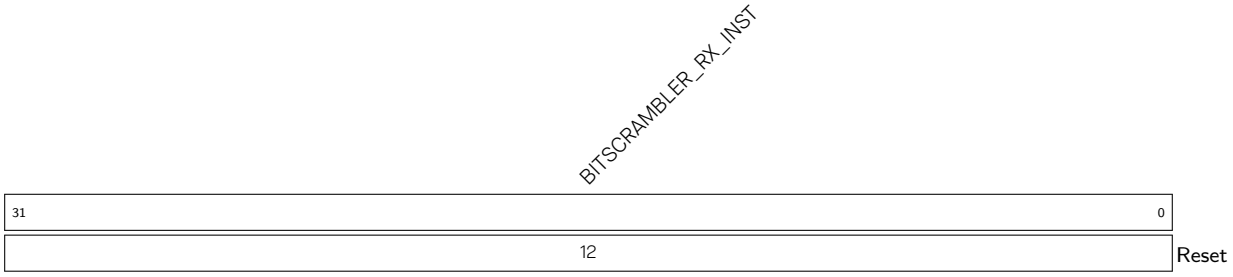
Register 62.3. BITSCRAMBLER_RX_INST_CFG0_REG (0x0008)



BITSCRAMBLER_RX_INST_IDX Configures the index of the BitScrambler instruction to be accessed via BITSCRAMBLER_RX_INST_CFG1_REG. (R/W)

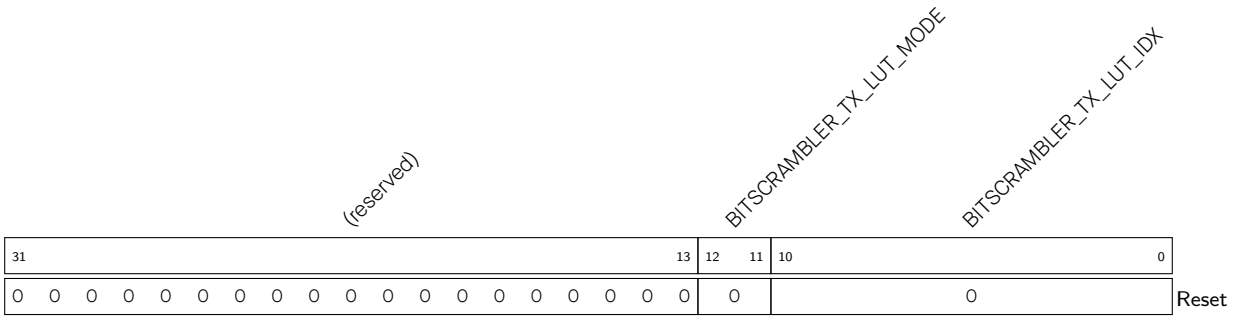
BITSCRAMBLER_RX_INST_POS Configures the offset into the 257-bit BitScrambler instruction (in 32-bit increments) to be accessed via BITSCRAMBLER_RX_INST_CFG1_REG. (R/W)

Register 62.4. BITSCRAMBLER_RX_INST_CFG1_REG (0x000C)



BITSCRAMBLER_RX_INST Configures the instruction to be accessed at the address specified by BITSCRAMBLER_RX_INST_CFG0_REG. (R/W)

Register 62.5. BITSCRAMBLER_TX_LUT_CFG0_REG (0x0010)



BITSCRAMBLER_TX_LUT_IDX Configures where in LUT RAM to access. Measurement unit: word size configured by BITSCRAMBLER_TX_LUT_MODE. (R/W)

BITSCRAMBLER_TX_LUT_MODE Configures the word size of LUT RAM for Bitscrambler programs.

0: 1 Byte

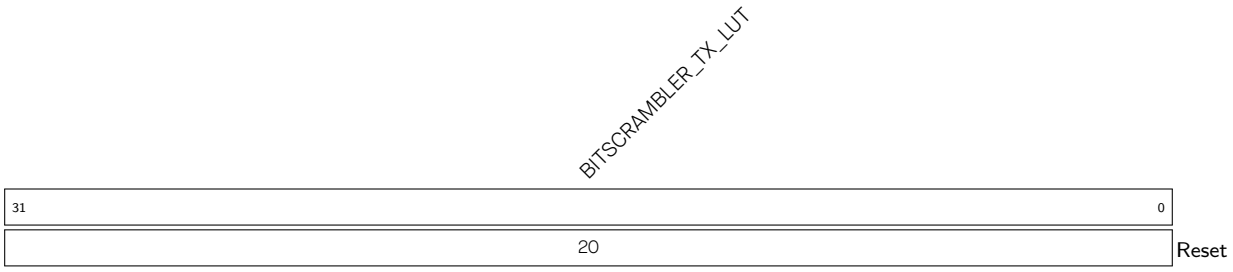
1: 2 Bytes

2: 4 Bytes

3: Reserved

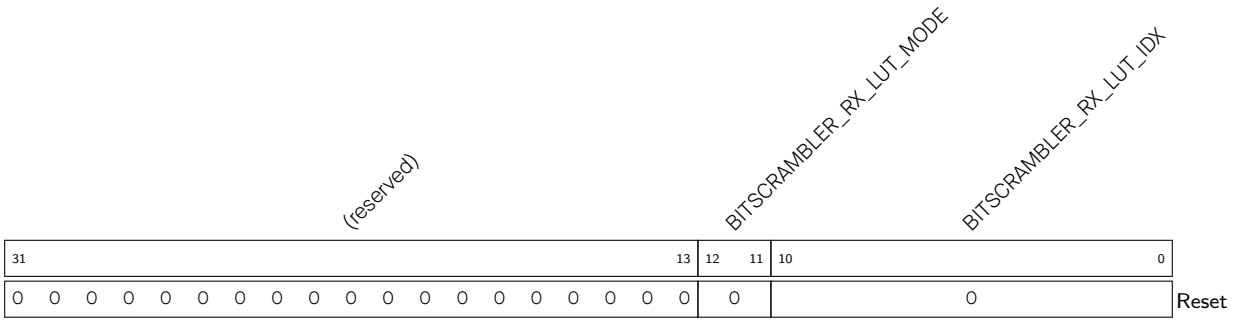
(R/W)

Register 62.6. BITSCRAMBLER_TX_LUT_CFG1_REG (0x0014)



BITSCRAMBLER_TX_LUT Configures the LUT entry to be accessed at the position specified by BITSCRAMBLER_TX_LUT_CFG0_REG. (R/W)

Register 62.7. BITSCRAMBLER_RX_LUT_CFG0_REG (0x0018)



BITSCRAMBLER_RX_LUT_IDX Configures where in LUT RAM to access. Measurement unit: word size configured by BITSCRAMBLER_RX_LUT_MODE. (R/W)

BITSCRAMBLER_RX_LUT_MODE Configures the word size of LUT RAM for BitScrambler programs.

0: 1 byte

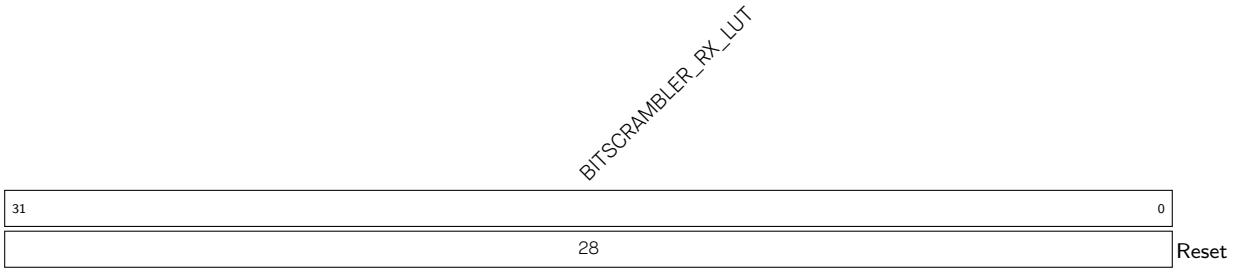
1: 2 bytes

2: 4 bytes

3: Reserved

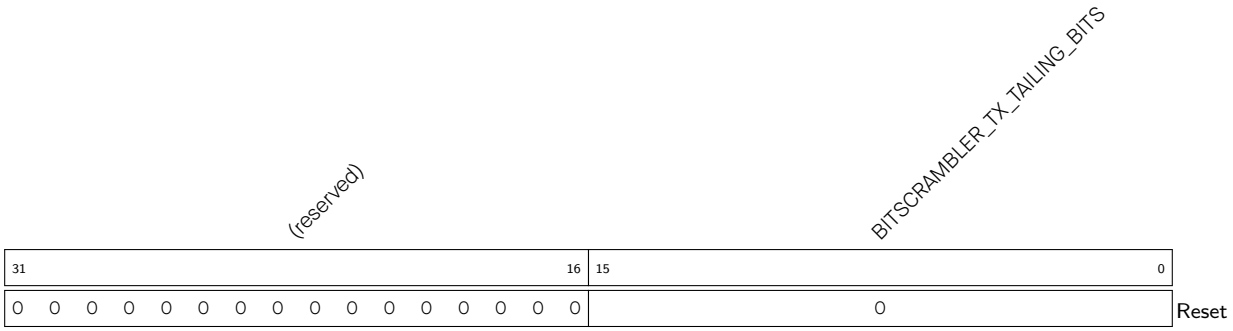
(R/W)

Register 62.8. BITSCRAMBLER_RX_LUT_CFG1_REG (0x001C)



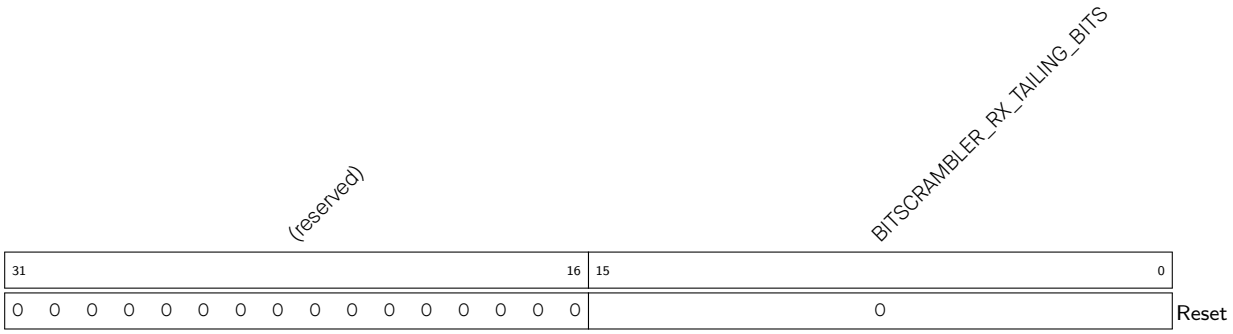
BITSCRAMBLER_RX_LUT Configures the LUT entry to be accessed at the position specified by BITSCRAMBLER_RX_LUT_CFG0_REG. (R/W)

Register 62.9. BITSCRAMBLER_TX_TAILING_BITS_REG (0x0020)



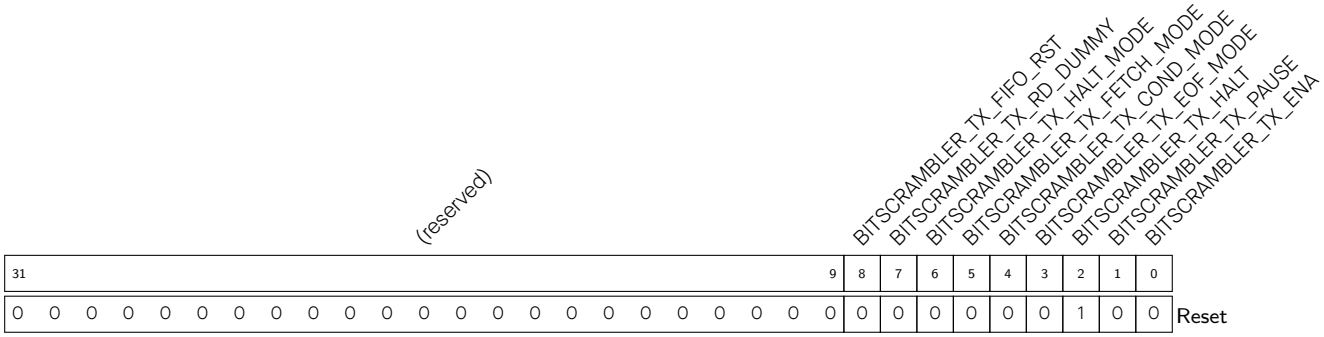
BITSCRAMBLER_TX_TAILING_BITS Configures the length of extra data after getting EOF for the TX BitScrambler core. Measurement unit: bit. (R/W)

Register 62.10. BITSCRAMBLER_RX_TAILING_BITS_REG (0x0024)



BITSCRAMBLER_RX_TAILING_BITS Configures the length of extra data after getting EOF for the BitScrambler RX core. Measurement unit: bit. (R/W)

Register 62.11. BITSCRAMBLER_TX_CTRL_REG (0x0028)



BITSCRAMBLER_TX_ENA Configures whether to enable BitScrambler TX core.

0: Disable

1: Enable

(R/W)

BITSCRAMBLER_TX_PAUSE Configures whether to pause BitScrambler TX core. A paused core can be un-paused to resume execution.

0: Not pause

1: Pause

(R/W)

BITSCRAMBLER_TX_HALT Configures whether to halt BitScrambler TX core. Halting the BitScrambler TX core will flush the FIFOs and cannot be resumed; a FIFO reset and a restart is needed.

0: Not halt

1: Halt (R/W)

BITSCRAMBLER_TX_EOF_MODE Configures BitScrambler TX core EOF signal generating mode. This is combined with BITSCRAMBLER_TX_TAILING_BITS for use.

0: Data written to the output FIFO are counted to generate delayed EOF

1: Data read from the input FIFO are counted to generate delayed EOF

(R/W)

BITSCRAMBLER_TX_COND_MODE Configures BitScrambler TX LOOP instruction condition mode.

0: Use the less than operator to get the condition

1: Use not equal operator to get the condition

(R/W)

BITSCRAMBLER_TX_FETCH_MODE Configures BitScrambler TX core fetch instruction mode.

0: Prefetch by reset

1: Fetch by instructions

(R/W)

BITSCRAMBLER_TX_HALT_MODE Configures BitScrambler TX core halt mode when BITSCRAMBLER_TX_HALT is set

0: Wait for write data back done

1: Ignore write data back

(R/W)

Continued on the next page...

Register 62.11. BITSCRAMBLER_TX_CTRL_REG (0x0028)

Continued from the previous page...

BITSCRAMBLER_TX_RD_DUMMY Configures BitScrambler TX core read data mode when EOF received.

0: Wait read data

1: Ignore read data

(R/W)

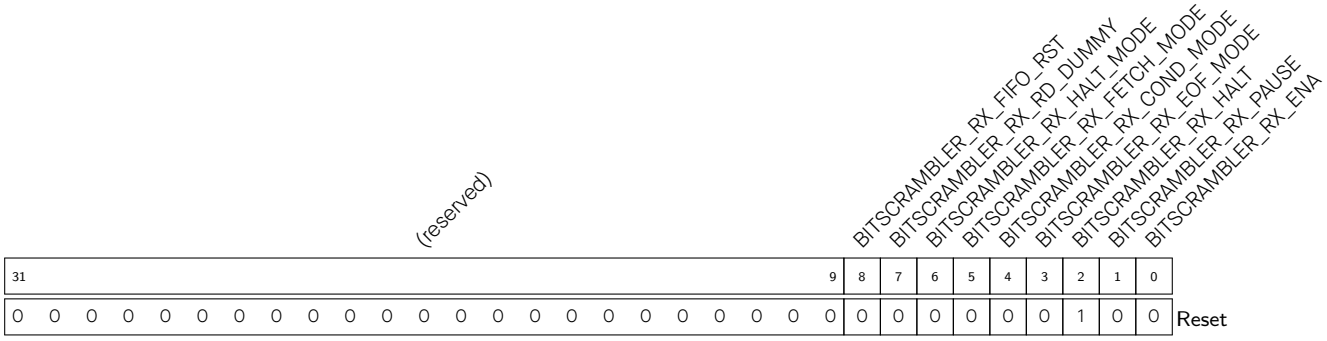
BITSCRAMBLER_TX_FIFO_RST Configures whether to reset BitScrambler TX FIFO.

0: Not reset

1: Reset

(WT)

Register 62.12. BITSCRAMBLER_RX_CTRL_REG (0x002C)



BITSCRAMBLER_RX_ENA Configures whether to enable BitScrambler RX core.

0: Disable

1: Enable

(R/W)

BITSCRAMBLER_RX_PAUSE Configures whether to pause BitScrambler RX core. A paused core can be un-paused to resume execution.

0: Not pause

1: Pause

(R/W)

BITSCRAMBLER_RX_HALT Configures whether to halt BitScrambler RX core. Halting the BitScrambler RX core will flush the FIFOs and cannot be resumed; a FIFO reset and a restart is needed.

0: Not halt

1: Halt

(R/W)

BITSCRAMBLER_RX_EOF_MODE Configures BitScrambler RX core EOF signal generating mode. This is combined with BITSCRAMBLER_RX_TAILING_BITS for use.

0: Data written to the output FIFO are counted to generate delayed EOF

1: Data read from the input FIFO are counted to generate delayed EOF

(R/W)

BITSCRAMBLER_RX_COND_MODE Configures BitScrambler RX LOOP instruction condition mode.

0: Use the less than operator to get the condition

1: Use not equal operator to get the condition

(R/W)

BITSCRAMBLER_RX_FETCH_MODE Configures BitScrambler RX core fetch instruction mode

0: Prefetch by reset

1: Fetch by instructions

(R/W)

Continued on the next page...

Register 62.12. BITSCRAMBLER_RX_CTRL_REG (0x002C)

Continued from the previous page...

BITSCRAMBLER_RX_HALT_MODE Configures BitScrambler RX core halt mode when BITSCRAMBLER_RX_HALT is set

0: Wait for write data back done

1: Ignore write data back

(R/W)

BITSCRAMBLER_RX_RD_DUMMY Configures BitScrambler RX core read data mode when EOF received.

0: Wait read data

1: Ignore read data

(R/W)

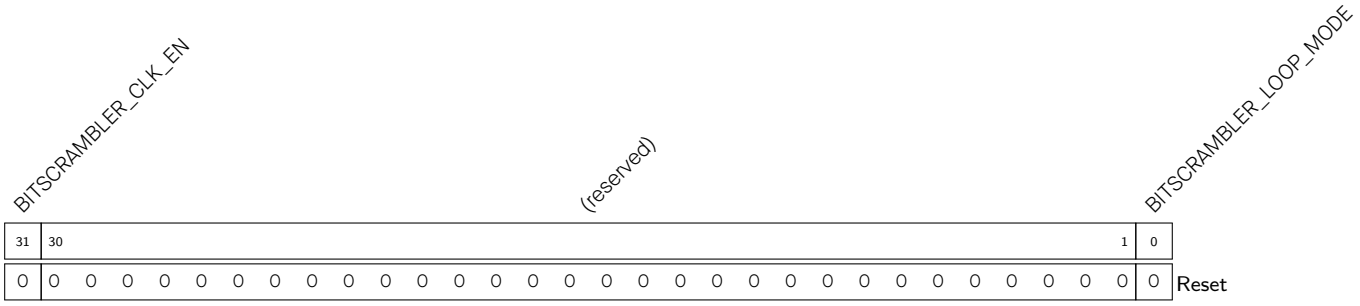
BITSCRAMBLER_RX_FIFO_RST Configures whether to reset BitScrambler RX FIFO.

0: Not reset

1: Reset

(WT)

Register 62.13. BITSCRAMBLER_SYS_REG (0x00F8)



BITSCRAMBLER_LOOP_MODE Configures whether to enable BitScrambler TX to DMA RX loopback mode

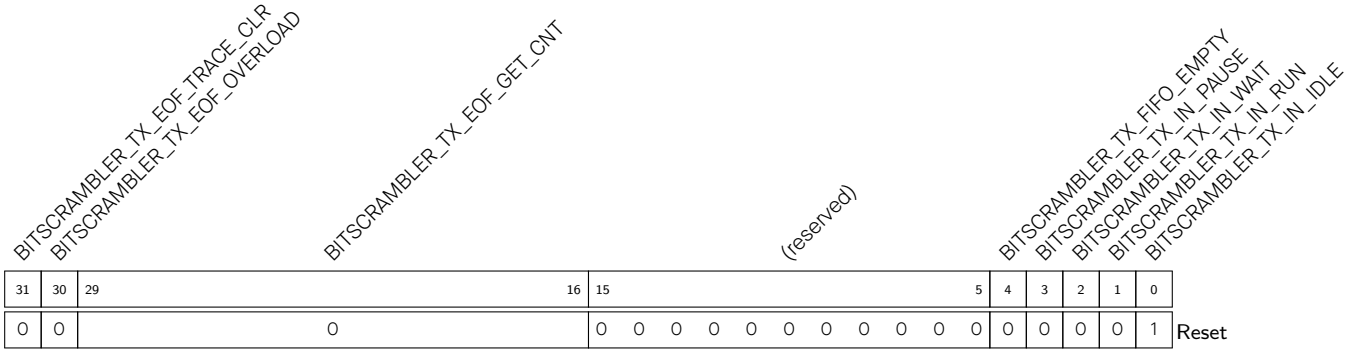
0: Disable

1: Enable

(R/W)

BITSCRAMBLER_CLK_EN Reserved. (R/W)

Register 62.14. BITSCRAMBLER_TX_STATE_REG (0x0030)



BITSCRAMBLER_TX_IN_IDLE Represents whether BitScrambler TX core is halted.

0: Not halted

1: Halted

(RO)

BITSCRAMBLER_TX_IN_RUN Represents whether BitScrambler TX core is running.

0: Not running

1: Running

(RO)

BITSCRAMBLER_TX_IN_WAIT Represents whether BitScrambler TX core is waiting for write back done.

0: Not waiting

1: Waiting

(RO)

BITSCRAMBLER_TX_IN_PAUSE Represents whether BitScrambler TX core is paused.

0: Not paused

1: Paused

(RO)

BITSCRAMBLER_TX_FIFO_EMPTY Represents whether BitScrambler TX FIFO is empty

0: Not empty

1: Empty

(RO)

BITSCRAMBLER_TX_EOF_GET_CNT Represents byte count of BitScrambler TX core after EOF is received. (RO)

BITSCRAMBLER_TX_EOF_OVERLOAD Represents whether BitScrambler TX core tries to process more than one EOF.

0: Not try to process more than one EOF

1: Try to process more than one EOF

(RO)

BITSCRAMBLER_TX_EOF_TRACE_CLR Configures whether to clear BITSCRAMBLER_TX_EOF_OVERLOAD and BITSCRAMBLER_TX_EOF_GET_CNT.

0: Not clear

1: Clear

(WT)

Register 62.15. BITSCRAMBLER_RX_STATE_REG (0x0034)



BITSCRAMBLER_RX_IN_IDLE Represents whether BitScrambler TX core is halted.

0: Not halted

1: Halted

(RO)

BITSCRAMBLER_RX_IN_RUN Represents whether BitScrambler TX core is running.

0: Not running

1: Running

(RO)

BITSCRAMBLER_RX_IN_WAIT Represents whether BitScrambler TX core is waiting for write back done.

0: Not waiting

1: Waiting

(RO)

BITSCRAMBLER_RX_IN_PAUSE Represents whether BitScrambler TX core is paused.

0: Not paused

1: Paused

(RO)

BITSCRAMBLER_RX_FIFO_FULL Represents whether BitScrambler RX FIFO is full.

0: Not full

1: Full

(RO)

BITSCRAMBLER_RX_EOF_GET_CNT Represents byte count of BitScrambler TX core after EOF is received. (RO)

BITSCRAMBLER_RX_EOF_OVERLOAD Represents whether BitScrambler RX core tries to process more than one EOF.

0: Not try to process more than one EOF

1: Try to process more than one EOF

(RO)

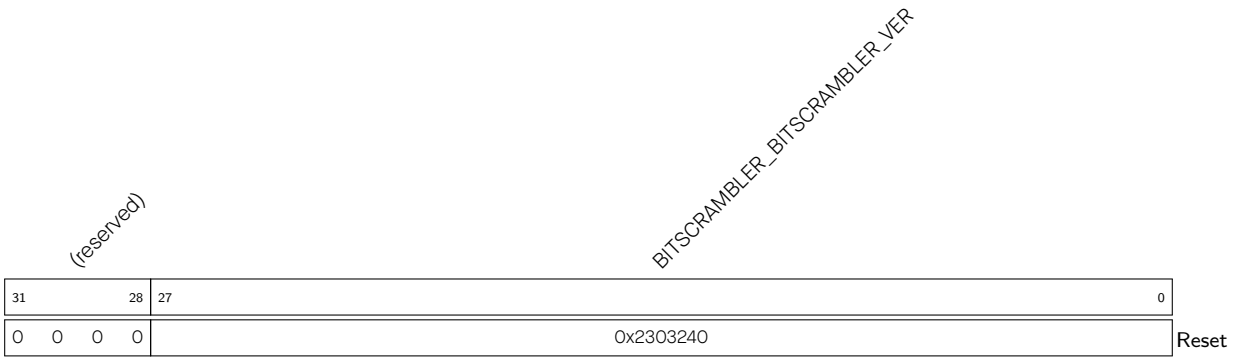
BITSCRAMBLER_RX_EOF_TRACE_CLR Configures whether to clear BITSCRAMBLER_RX_EOF_OVERLOAD and BITSCRAMBLER_RX_EOF_GET_CNT.

0: Not clear

1: Clear

(WT)

Register 62.16. BITSCRAMBLER_VERSION_REG (0x00FC)



BITSCRAMBLER_BITSCRAMBLER_VER Version control register. (R/W)

Part VIII

Analog Signal Processing

This part describes components related to analog-to-digital conversion, voltage comparison, on-chip sensors, and features such as temperature and touch sensing, demonstrating the system's capabilities in handling analog signals.

Chapter 63

Touch Sensor (TOUCH)

ESP32-P4 has 14 capacitive touch sensor units, each of which can be connected to an external touch panel via a touch pin (GPIO pin) to constitute a touch sensor system. The ESP32-P4 touch sensor system is mainly used in human-computer interactions to detect finger touch or proximity. It also features moisture resistance and waterproof design.

63.1 Terminology

To better illustrate the functions of capacitive touch sensors, the following terms are used in this section.

Touch Sensor	Touch-related internal sensing circuitry.
Touch Pin	Pins with touch sensing feature.
Touch Panel	The external device connected to touch sensor via channel (touch pin) to detect finger touch.
Touch Sensor System	Capacitive touch sensing system, consisting of touch sensor, touch pin, traces, and touch panel.

Note:
In subsequent description, "touch panel is sampled/scanned/measured" indicates the same action to touch pin, i.e. "touch pin is sampled/scanned/measured".

63.2 Feature List

The ESP32-P4 touch sensor has the following features:

- Detection of 14 capacitive touch pins
- Sampling triggered by software or dedicated hardware timer
- Two sampling methods:
  - Pulses from the touch pins used as clock signals to count the sampling period
  - Pulses from the touch pins used as digital signals; sample the rising edge of the digital signal with the system clock to count the sampling period
- Scan mode, supporting sequential sampling of multiple touch pins by configuring the Touch FSM.
- Timeout mechanism to monitor channel abnormality
- Frequency hopping to increase the anti-interference of detection
- Proximity sensing mode with up to three configurable channels

- Configuration of individual touch sensors to operate normally in sleep mode
- Wake-up by touch sensor
- Moisture resistance
- Waterproof design

63.3 Architectural Overview

63.3.1 Touch Panel

A touch panel consists of the following components:

- Electrode: Changes in capacitance when touched by a finger.
- Substrate: The base material on which the protective cover, electrode, and the electrode's connector to the channel are built.
- Protective cover: To physically separate the other components from the external environment.

When developing applications using the touch sensing feature, users should decide on the placement, the material, or the arrangement of the touch panels during mechanical design. For more information about the design guidelines, please refer to Touch Sensor Application Note. Touch panel can be connected to ESP32-P4 touch sensor via touch pin (channel). See Figure 63.3-1.

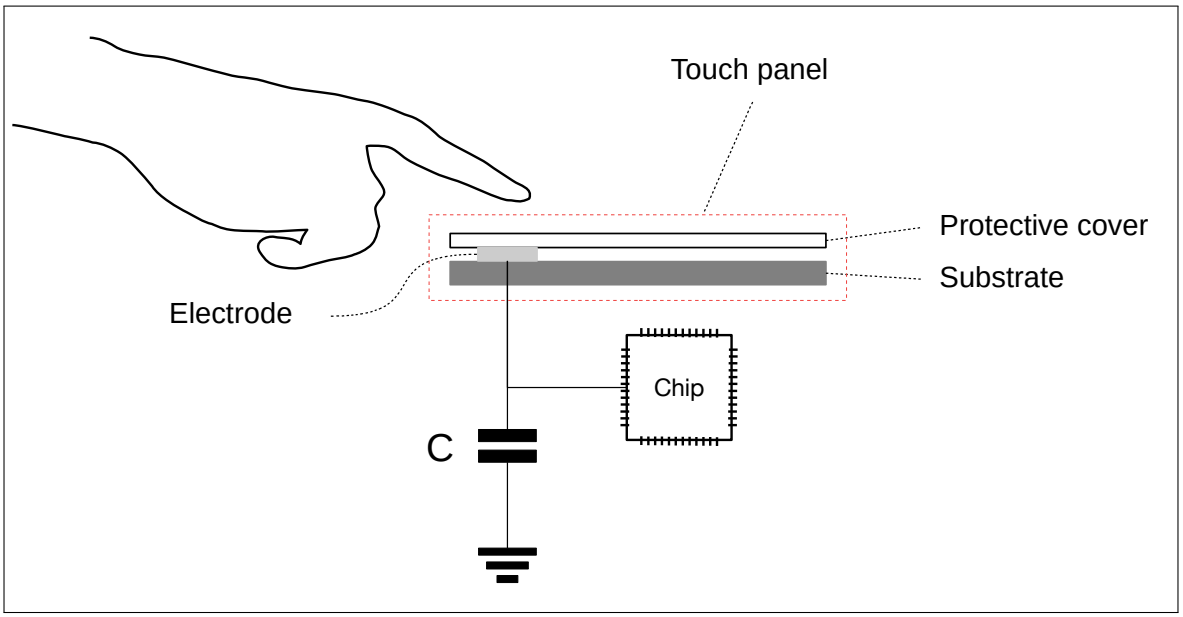


Figure 63.3-1. Touch Sensor System

When users touch the protective cover, the capacitance of the electrode increases. If the electrode is connected to one of the ESP32-P4 touch sensors via a channel, the touch sensor will be able to detect the electrode's change in capacitance. If the change of capacitance exceeds the configurable threshold, the touch sensor will trigger an interrupt.

63.3.2 Capacitive Touch Pin

ESP32-P4 provides 14 capacitive touch pins labeled as T1 ~ T14, each connected to the chip's pin to monitor finger touch from the external environment. The mapping between the touch sensors and the chip pins is shown in Table 63.3-1.

Table 63.3-1. ESP32-P4 Capacitive Touch Pins

Capacitive Touch Pin	Connected GPIO Pin
T1	GPIO2
T2	GPIO3
T3	GPIO4
T4	GPIO5
T5	GPIO6
T6	GPIO7
T7	GPIO8
T8	GPIO9
T9	GPIO10
T10	GPIO11
T11	GPIO12
T12	GPIO13
T13	GPIO14
T14	GPIO15

63.3.3 Touch Sensor

The touch sensor charges and discharges the touch pin with a fixed current source, and each charge and discharge generates a pulse signal, the TOUCH_OUT signal. The touch sensor detects the capacitance change of the touch pin. If the touch pin is touched or approached by a finger, the touch pin's capacitance increases and the charging and discharging time is extended. By measuring the time required for charging and discharging the touch pin a fixed number of times, it can be inferred whether the touch pin has been touched or not.

Figure 63.3-2 illustrates the internal structure of the touch sensors. Each of the touch sensors has a set of input and output signals, some of which are connected to the Touch FSM (see Section 63.4.1). Table 63.3-2 shows the output signals of the touch sensor and their functions.

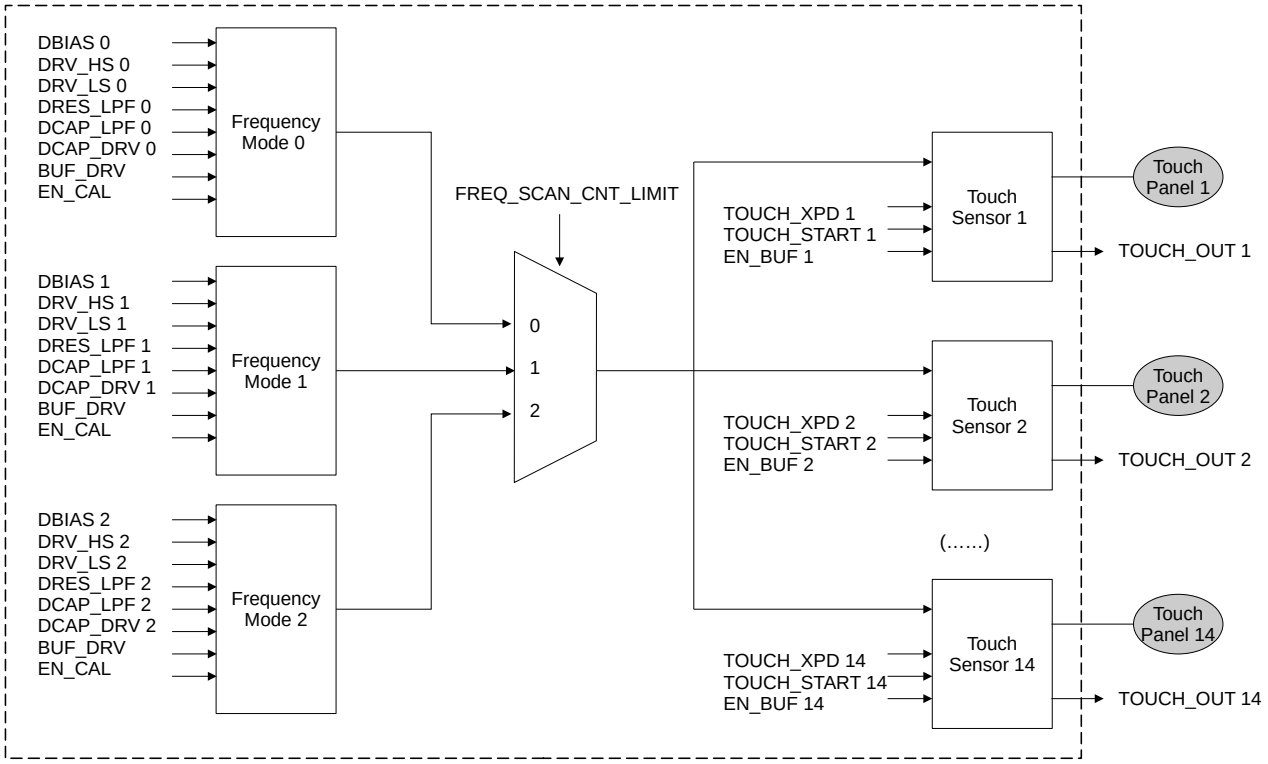


Figure 63.3-2. Touch Sensor Internal Structure

Table 63.3-2. Capacitive Touch Sensor Output Signal

Signal	Source	Function
TOUCH_XPD	Touch FSM	Controls the power-up and power-down of the touch sensor
TOUCH_START	Touch FSM	Controls the touch sensor to initiate a measurement. The measurement ends when the Touch FSM pulls down the TOUCH_START signal
DBIAS	Touch FSM	The FSM selects different DBIAS based on the frequency mode to adjust the voltage of the touch sensor's internal LDO, which affects the charging and discharging speed of the touch sensor
DRV_HS	Touch FSM	The FSM selects different DRV_HS based on the frequency mode to adjust the high-speed driving capability of the touch sensor
DRV_LS	Touch FSM	The FSM selects different DRV_LS based on the frequency mode to adjust the low-speed driving capability of the touch sensor
DRES_LPF	Touch FSM	The FSM selects different DRES_LPF based on the frequency mode to adjust the RC low-pass filter, which can be configured as 0 kΩ, 3 kΩ, 4.5 kΩ, or 7.5 kΩ
DCAP_LPF	Touch FSM	The FSM selects different DCAP_LPF based on the frequency mode to adjust the RC low-pass filter with an adjustment range of 0-2.54 pF and a step of 20 fF
DCAP_DRV	Touch FSM	Used to adjust the internal capacitance connected to touch channels during measurement, usually set to 0
EN_CAL	Touch FSM	Used to disconnect from the touch pins and to test the capacitance in the touch sensor (the touch panel needs to be disabled)

Signal	Source	Function
BUF_DRV	Touch FSM	Adjusts the driving capability of TOUCH BUFF
EN_BUF	Touch FSM	Enables TOUCH BUFF, a software-configurable touch panel, to mitigate the effects of water droplets on the sensor for moisture tolerance
TOUCH_OUT	Touch sensor	The Touch FSM counts the pulses in the TOUCH_OUT signal and measures the time it takes to generate N pulses

63.4 Functional Description

63.4.1 Touch FSM

The Touch FSM performs a measurement by selecting a touch sensor and controlling the necessary signals to and from a touch sensor. Figure 63.4-1 shows the internal structure and operating mechanism of the Touch FSM. The Touch FSM is clocked by LP_DYN_FAST_CLK. For more information on LP_DYN_FAST_CLK, see Chapter 11 *Reset and Clock*.

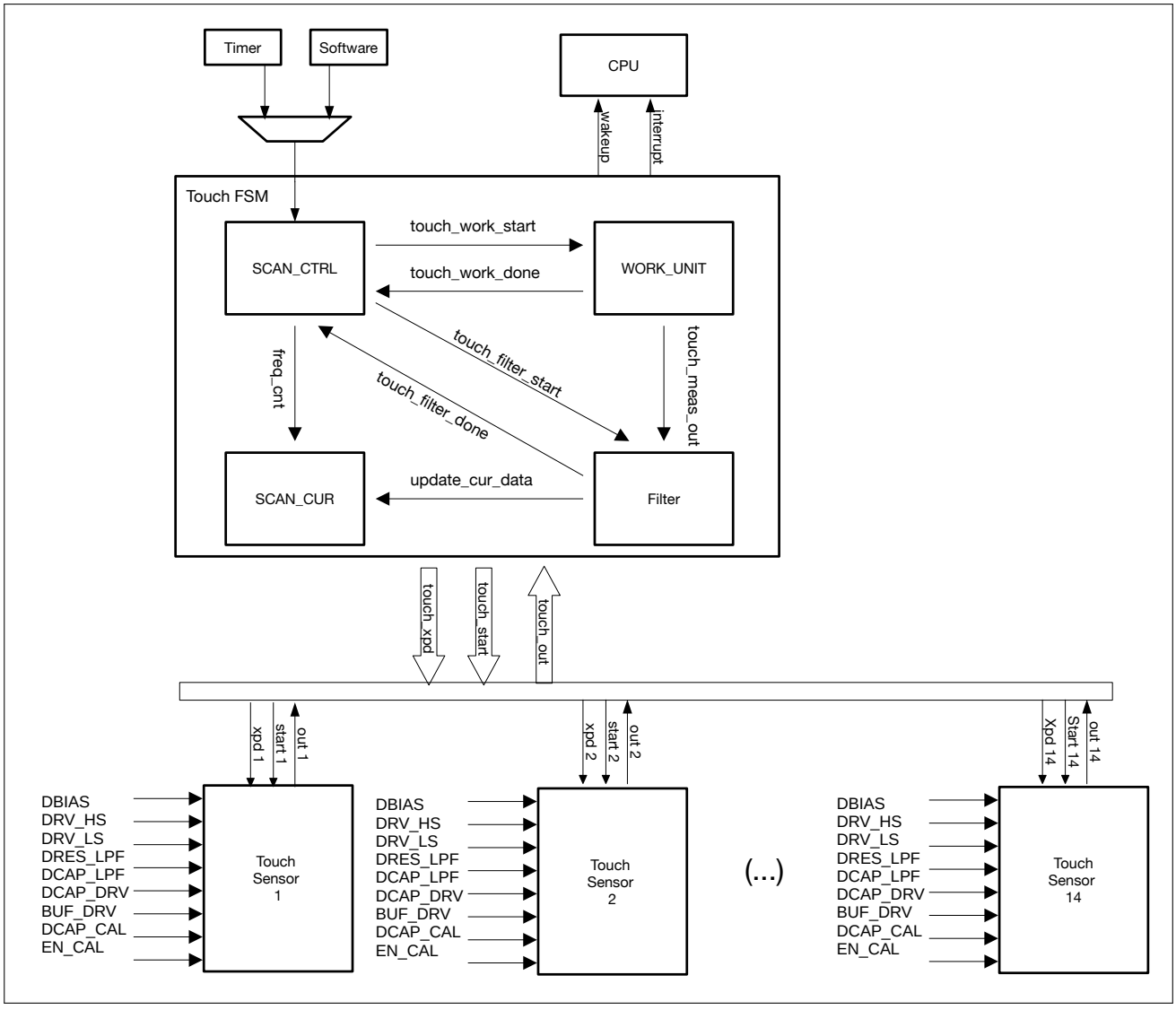


Figure 63.4-1. Touch FSM Structure and Mechanism

The following introduces the modules in the Touch FSM.

- SCAN_CTRL: Select the touch pin to measure, set the parameters such as hopping frequency, and controls the start and end of measurement for each touch sensor in scan mode.
- WORK_UNIT: Sample the selected touch sensors during a measurement.
- Filter: If enabled, each touch sensor can filter a series of measurements via an infinite impulse response (IIR) filter. The filtered value will be returned as the sampled value. See Section 63.4.3.1 for details.
- SCAN_CUR: Cache the parameters of different touch sensors at different frequency modes during the measurement to be used for the next measurement.

63.4.2 Sampled Signal Preprocessing

The core mechanism of the touch sensor is to reflect whether the touch panel is touched or not based on the change of the output pulse wave, i.e., the TOUCH_OUT signal, through detecting the change of capacitance on the touch panel. Therefore, the sampling of the TOUCH_OUT signal is critical. Figure 63.4-2 shows the preprocessing of the TOUCH_OUT signal.

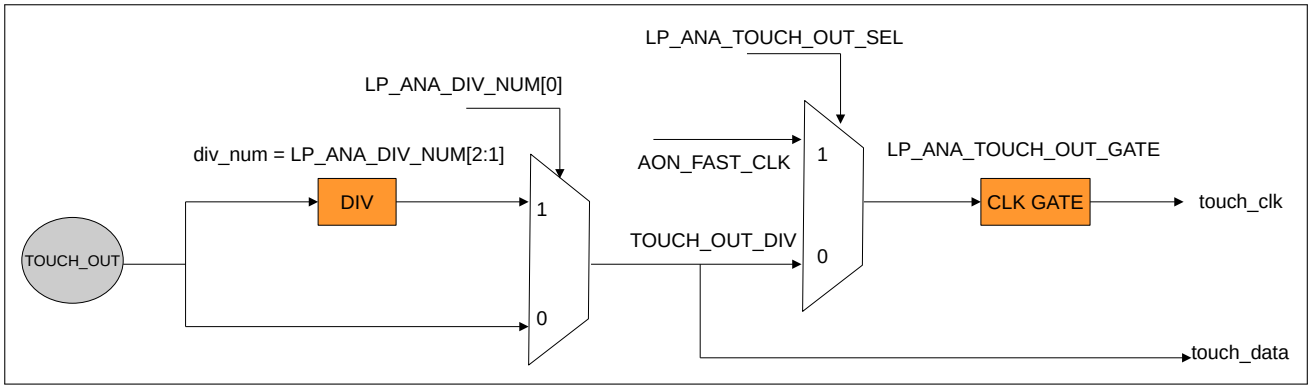


Figure 63.4-2. TOUCH_OUT Signal Preprocessing

The frequency of TOUCH_OUT might be much higher than the sampling frequency. Therefore, in order to improve the accuracy of the signal sampling, users can divide the TOUCH_OUT signal by configuring LP_ANA_DIV_NUM*n*. *n* in this field represents different frequency modes in frequency hopping (refer to Section 63.4.2.4) with the value 0-2. The lowest bit configures whether to enable or disable frequency division. Set the lowest bit to 0 to disable frequency division, and set it to 1 to enable division. The highest two bits indicate the frequency division coefficient with a value of 0/1/2/3, corresponding to the 0/2/4/6 divisions.

If the frequency is still higher than twice the sampling frequency after frequency division, set LP_ANA_TOUCH_OUT_SEL to 1 to directly use the divided TOUCH_OUT signal as a clock. Set LP_ANA_TOUCH_OUT_GATE to configure whether to enable this clock.

63.4.2.1 Measurement Process

Depending on whether frequency hopping is enabled, a measurement of a touch pin can be divided into the following two processes:

- Frequency hopping not enabled:

1. The Touch FSM selects the touch sensor to be measured. The relevant signals are routed to that touch sensor.
2. The Touch FSM drives the START signal to the touch sensor to initiate the measurement. Internally, the Touch FSM starts a touch counter to time the duration of the measurement.
3. A pulse counter in the Touch FSM is incremented on each pulse received from the touch sensor's TOUCH_OUT signal.
4. When the pulse counter reaches the count threshold set in LP_ANA_TOUCH_MEAS_NUMO, the measurement is complete. The START signal is de-asserted, and the touch counter is stopped. The value of the stopped touch counter x LP_DYN_SLOW_CLK cycles indicates the time taken to charge and discharge the touch pin for LP_ANA_TOUCH_MEAS_NUMO cycles.

- Frequency hopping enabled:

1. The Touch FSM selects the touch sensor to be measured. The relevant signals are routed to that touch sensor.
2. Set LP_ANA_FREQ_SCAN_EN to 1 to enable frequency hopping, configure LP_ANA_FREQ_SCAN_CNT_LIMIT to set the number of frequency modes supported for frequency hopping (up to three), and configure the parameters required for each frequency mode and the number of pulses for sampling the TOUCH_OUT signal.
3. The Touch FSM drives the START signal to the touch sensor to initiate the measurement. Internally, the Touch FSM starts a touch counter to time the duration of the measurement.
4. A pulse counter in the Touch FSM is incremented on each pulse received from the touch sensor's TOUCH_OUT signal.
5. When the pulse counter reaches the count threshold set in LP_ANA_TOUCH_MEAS_NUMO of the corresponding frequency mode, the measurement is complete. The value of the stopped touch counter indicates the time taken to charge and discharge the touch pin LP_ANA_TOUCH_MEAS_NUMO cycles with the current frequency.
6. The sampling frequency counter is incremented by 1 and the touch counter is cleared. The next frequency mode will be sampled after a small delay. When the samplings are finished for all frequency modes, the START signal is de-asserted, and the touch counter is cleared.

The sampled value (measured value) is the value of the touch counter indicated as touch_raw_data. The value of touch_raw_data can be read from RTC_TOUCH_PAD*n*_DATA. Note that the value returned by RTC_TOUCH_PAD*n*_DATA may also be the filtered touch_raw_data, i.e., touch_smooth_data and benchmark. Configure LP_ANA_TOUCH_DATA_SEL and LP_ANA_TOUCH_FREQ _SEL to select the type of the return value. For more information on the sampled value types and how to judge touch actions based on the data, refer to Section 63.4.3.1.

Note:
Note that if the pulse does not reach the set threshold for a long time while the touch counter has reached the timeout threshold set in LP_ANA_TOUCH_TIMEOUT_NUM with LP_ANA_TOUCH_TIMEOUT_EN set as 1, the RTC_TOUCH_TIMEOUT_INT_RAW timeout interrupt will be triggered, indicating a circuit abnormality.

63.4.2.2 Trigger Source of Measurement

The Touch FSM initiates a measurement by sending a START signal. The START signal can either be triggered by software, or by a dedicated hardware timer known as the "touch timer". The use of a touch timer allows periodic measurements to be taken without software intervention.

The touch timer is clocked by LP_DYN_SLOW_CLK and should be configured with a period in number of LP_DYN_SLOW_CLK cycles. The START signal is generated when the touch timer expires. The touch timer is reset when the measurement completes and restarts counting until the next expiration.

- To trigger the START signal by software:
  - Set LP_ANA_TOUCH_START_FORCE to 1 to trigger the START signal by software.
  - Software sets LP_ANA_TOUCH_START_EN to 1 and generates the START signal to initiate a measurement.
- To trigger the START signal by the touch timer:
  - Clear LP_ANA_TOUCH_START_FORCE.
  - Configure the wait time before the timer sends the START signal (unit: LP_DYN_SLOW_CLK cycle) via PMU_TOUCH_WAIT_CYCLES.
  - Configure the wait time after the timer receives the DONE signal upon the completion of the touch sensor sampling (unit: LP_DYN_SLOW_CLK cycle) via PMU_TOUCH_SLEEP_CYCLES.
  - Set PMU_TOUCH_SLEEP_TIMER_EN to 1 to enable the touch timer.
- To force exit the current measurement cycle of the touch timer by software, there are two approaches:
  1. Generate the end signal of the touch sensor to exit the current measurement cycle of the touch timer.
     - Set LP_ANA_TOUCH_DONE_FORCE to 1 to select the DONE signal to be triggered by software.
     - Software sets LP_ANA_TOUCH_DONE_EN to 1 to generate the DONE signal to end a measurement.
  2. Use the software to directly force the touch timer to exit the current measurement cycle.
     - Set PMU_TOUCH_FORCE_DONE to generate the DONE signal and force the touch timer to exit the current measurement cycle.

Note:
The touch timer waits for a period of time before generating the START signal and after receiving the DONE signal, so the interval between two consecutive measurements is (PMU_TOUCH_WAIT_CYCLES + PMU_TOUCH_SLEEP_CYCLES)□and the timing clock is LP_DYN_SLOW_CLK.

63.4.2.3 Scan Mode

Scan mode involves the Touch FSM taking measurements of multiple touch sensors in sequential order. On every START signal, a new touch sensor is selected for measurement, thus allowing multiple touch pins to be monitored. The scan process is illustrated in Figure 63.4-3.

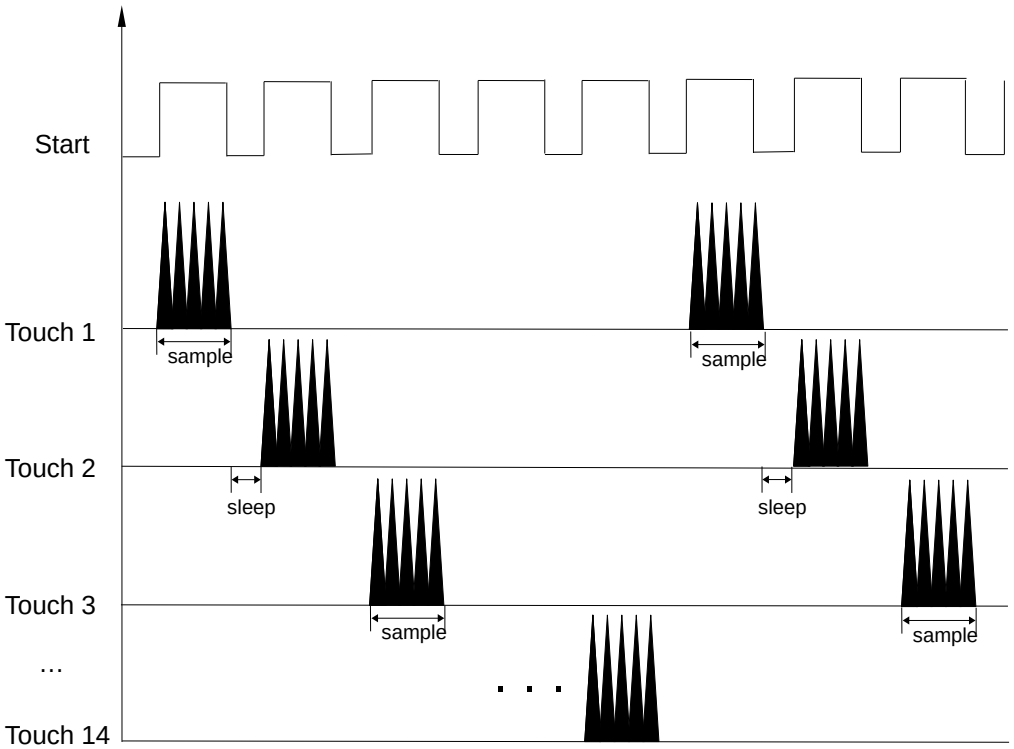


Figure 63.4-3. Timing Diagram of Scan Mode

To enable the scan mode, specify the bit map of the enabled touch pins in LP_ANA_TOUCH_SCAN_PAD_MAP. The Touch FSM will select one touch pin from the bit map of the enabled touch pins in turn to measure per START signal, i.e., only one measurement is conducted per START signal. As multiple START signals are generated, the Touch FSM will cycle through the enabled touch pins in sequential order.

When measuring for different touch sensors, the sleep interval between two measurements is PMU_TOUCH_WAIT_CYCLES + PMU_TOUCH_SLEEP_CYCLES with the clock cycle of LP_DYN_SLOW_CLK.

Note:

- After the touch sensor receives the START signal from the touch timer, it waits a short period before starting the measurement to ensure the stability of the sampling signal. The wait time is configured by LP_ANA_TOUCH_XPD_WAIT. The timing clock is AON_FAST_CLK.
- If the touch timer is used to generate the START signal, the scanning can be conducted without software intervention.

63.4.2.4 Frequency Hopping

When frequency hopping is enabled, the Touch FSM can measure the touch sensor at different frequency modes. Up to three frequency modes are supported, each corresponding to a set of configuration parameters. When a START signal is generated, the measurement will be carried out for each frequency mode sequentially, and the DONE signal will be generated when the measurements are completed for all frequency modes. Frequency hopping can be enabled together with the scan mode. In such case, users can monitor the touch condition of multiple touch sensors under different frequency modes by generating multiple START

signals. The scanning process is shown in Figure 63.4-4.

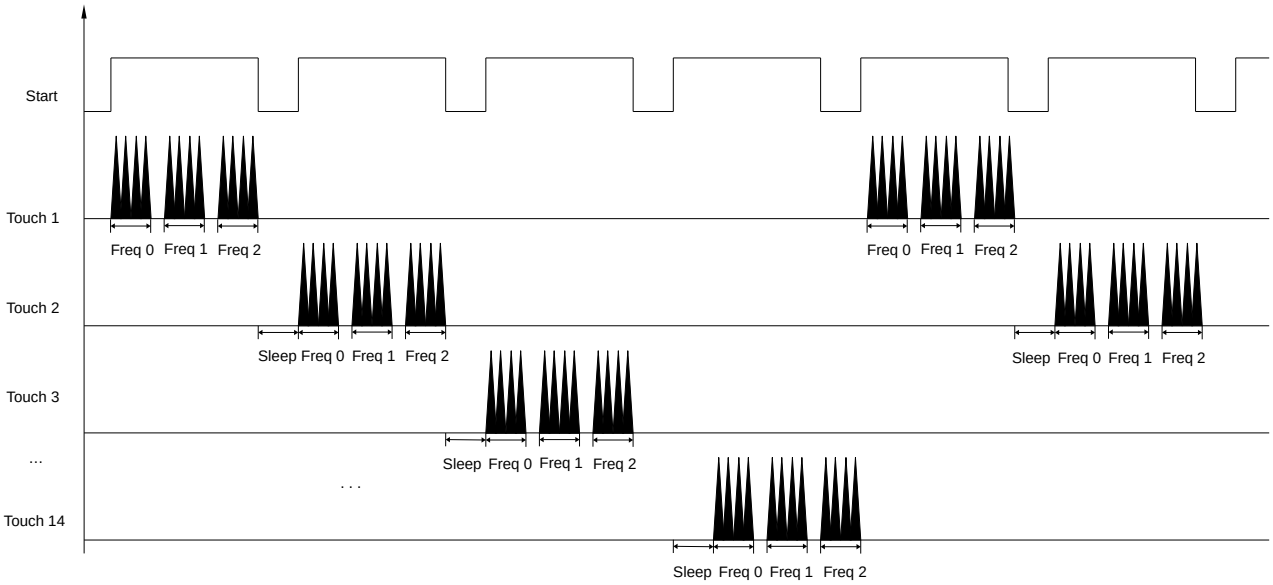


Figure 63.4-4. Touch Sensor Frequency Hopping Sequence Timing

Set LP_ANA_FREQ_SCAN_EN to 1 to enable frequency hopping, configure LP_ANA_FREQ_SCAN_CNT_LIMIT to select the maximum number of frequency mode supported, and configure the following analog parameter registers to adjust the sampling performance of the touch sensor at different frequency modes:

- LP_ANA_TOUCH_FREQ*n*_DBIAS (*n* = 0-2)
- LP_ANA_TOUCH_FREQ*n*_DRV_HS (*n* = 0-2)
- LP_ANA_TOUCH_FREQ*n*_DRV_LS (*n* = 0-2)
- LP_ANA_TOUCH_FREQ*n*_DRES_LPF (*n* = 0-2)
- LP_ANA_TOUCH_FREQ*n*_DCAP_LPF (*n* = 0-2)

Upon each START signal, the Touch FSM measures consecutively for frequency mode 0, 1, and 2. A START signal triggers a maximum of three measurement operations. When the measurements are completed for all frequency modes, the START signal is de-asserted until the next measurement. Therefore, the Touch FSM performs measurements on the enabled touch sensors in sequence upon multiple START signals. A touch on the same sensor is recognized only when the number of hit frequency points is greater than or equal to LP_ANA_FREQ_SCAN_CNT_RISE.

Note:

- When the same touch sensor is measured at different frequency modes, the wait time between measurements can be configured by LP_ANA_TOUCH_XPD_WAIT (count clock: AON_FAST_CLK).
- After receiving the START signal from the touch timer, the touch sensor waits for a short period of time before starting measuring to ensure the stability of the sampling signal. The wait time can be configured by LP_ANA_TOUCH_XPD_
  WAIT (count clock: AON_FAST_CLK).
- If a touch timer is used to generate the START signal, the scanning can be conducted without software intervention.

63.4.3 Touch Detection

63.4.3.1 Sampled Value

As introduced in Section 63.4.2.1, the sampled value of a measurement can be read from RTC_TOUCH_PAD*n*_DATA. However, the specific type of sampled value stored in RTC_TOUCH_PAD*n*_DATA is determined together by LP_ANA_TOUCH_DATA_SEL and LP_ANA_TOUCH_FREQ_SEL. The following three sampled value types are supported:

1. **touch_raw_data**: When the measurement is completed, the touch counter's value indicates the time taken to charge and discharge the touch pin a fixed number of times in LP_DYN_SLOW_CLK cycles. After the IIR filter is turned on, the data is the filtered data.
2. **touch_smooth_data**: A series of moving averages generated by the touch_raw_data values sampled from a single touch sensor filtred by an IIR filter.

   touch_smooth_data is less prone to noise spikes or outlier samples, thus often used for hardware touch detection. If frequency hopping is enabled, each touch pin will have a separate set of smooth_touch_data at different frequency modes and do not affect each other. The IIR filter that generates touch_smooth_data is configured via LP_ANA_TOUCH_SMOOTH_LVL. See Table 63.4-1 for more information.

Table 63.4-1. touch_smooth_data Calculation

LP_ANA_TOUCH_SMOOTH_LVL	Type	Formula
0	—	touch_raw_data
1	IIR 1/2	1/2 touch_raw_data + 1/2 touch_smooth_data
2	IIR 1/4	1/4 touch_raw_data + 3/4 touch_smooth_data
3	IIR 1/8	1/8 touch_raw_data + 7/8 touch_smooth_data

3. **benchmark**: Also generated from the touch_raw_data values by IIR filtering. However, the moving average window for benchmark is much wider.

   The benchmark value is usually used to represent the stable reading of a touch pin without the effect from a touch action. The touch detection is based on the benchmark value. Each touch sensor has a separate set of benchmark values at different frequency modes with a default value of 0xFFFF. The IIR filter that generates benchmark is configured via LP_ ANA_TOUCH_FILTER_MODE. Refer to Table 63.4-2 for more information.

   In addition to the IIR filtering, users can also update the benchmark value directly by software. The software update procedure is as follows:

   - Configure the target frequency point to be updated through LP_ANA_TOUCH_UPDATE_BENCHMARK_FREQ_SEL.
   - Configure the target channel to be updated through LP_ANA_TOUCH_UPDATE_BENCHMARK_PAD_SEL.
   - Configure the benchmark data value through LP_ANA_TOUCH_BENCHMARK_SW.
   - Configure LP_ANA_TOUCH_UPDATE_BENCHMARK_SW to trigger the benchmark data update by software.

This method is generally used for software debugging. When the benchmark data of the target frequency point and channel has been updated, the interrupt TOUCH_BENCHMARK_UPDATE_INT will be generated.

Table 63.4-2. benchmark Algorithm

LP_ANA_TOUCH_FILTER_MODE	Type	Formula
0	IIR 1/4	1/4 touch_raw_data + 3/4 benchmark
1	IIR 1/8	1/8 touch_raw_data + 7/8 benchmark
2	IIR 1/16	1/16 touch_raw_data + 15/16 benchmark
3	IIR 1/32	1/32 touch_raw_data + 31/32 benchmark
4	IIR 1/64	1/64 touch_raw_data + 63/64 benchmark
5	IIR 1/128	1/128 touch_raw_data + 127/128 benchmark
6	IIR 1/256	1/256 touch_raw_data + 255/256 benchmark
7	JITTER	touch_raw_data +/– LP_ANA_TOUCH_JITTER_STEP

63.4.3.2 Hardware Touch Detection

In hardware touch detection, the touch sensor can detect finger touch or release and trigger an interrupt. To use hardware touch detection, the following key parameters need to be defined:

- **finger_threshold**: The threshold value used to determine a touch or a touch interruption. finger_threshold is configured via LP_ANA_TOUCH_PAD*x*_TH*y*, where *x* (1-14) represents the touch pin index and *y* (0-2) represents the frequency mode configured in hopping frequency.
- **noise_threshold**: Indicates the maximum threshold of the upward and downward fluctuation of touch_smooth_data. noise_threshold is configured via LP_ANA_TOUCH_NOISE_THRES.
- **hysteresis**: Used to avoid touch misjudgments caused by fluctuations of touch_smooth_data around touch thresholds. The hysteresis value is configured via LP_ANA_TOUCH_HHYSTERESIS.

The finger_threshold and noise_threshold values are not absolute thresholds, but offsets from the benchmark value, while the hysteresis value is an offset from finger_threshold. This prevents from misdiagnosing the gradual or slow changes of touch_raw_data caused by environmental factors such as temperature, power, or noise as a touch action.

After a series of calculations and comparisons based on the above parameters, the touch sensor will output the touch results detected on the touch pins via interrupts. Each touch sensor corresponds to a control register LP_ANA_TOUCH_OUTEN to configure whether to output the detection result.

63.4.4 Proximity Mode

When an object such as a finger is close to, but not touching, a touch pin, a small change in capacitance occurs on the touch pin, which is much smaller than the change caused by a physical touch. Proximity mode allows for the detection of these small changes.

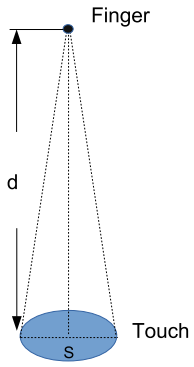


Figure 63.4-5. Sensing Area

- The maximum detection distance "d" is 16 cm and the sensing area "S" is 20 cm^2, where "d" is positively correlated with "S" as shown in Figure 63.4-5.
- Up to three touch pins can be configured to operate simultaneously in proximity mode.
- Frequency hopping is also supported in proximity mode.

In proximity mode, a touch sensor samples for a fixed number of times and accumulates the sample values. If the final cumulative value exceeds the configured threshold, a proximity object is considered to be detected and an interrupt will be triggered. Note that since the sample values are accumulated, the values introduced in Section 63.4.3.1, i.e., touch_raw_data, touch_smooth_data, and benchmark, will not be generated in proximity mode.

To operate in proximity mode:

- Configure a touch sensor to operate in proximity mode by setting LP_ANA_TOUCH_APPROACH_PAD0, LP_ANA_TOUCH_APPROACH_PAD1, or LP_ANA_TOUCH_APPROACH_PAD2.
- Configure LP_ANA_TOUCH_APPROACH_LIMIT to adjust the number of samplings to generate the cumulative value. The touch sensor contains an internal sample counter to record the number of samplings.
- Set the threshold value via LP_ANA_TOUCH_PAD*x*_TH*y*, where *x* (1-14) represents the touch pin index and *y* (0-2) represents the frequency mode configured in hopping frequency.
- When the sample counter reaches the number of samplings configured in LP_ANA_TOUCH_APPROACH_MEAS_NUM*n*:
  - If the cumulative value exceeds the threshold value, an interrupt will be triggered.
  - The sample counter and the cumulative value are reset to 0. The touch sensor restarts accumulating the sample values.

63.4.5 Sleep Mode

To reduce power consumption, the touch sensor also supports sleep mode. In sleep mode, users can designate one touch sensor to operate actively (hereinafter referred to as the sleeping touch sensor) to monitor whether a touch occurs in real time, while the remaining 13 touch sensors are all powered down in sleep mode. Once the sleeping touch sensor detects a touch or a touch release, it will generate an interrupt to wake up the rest of the touch sensors. The sleeping touch sensor also supports frequency hopping and

proximity mode. There is a separate set of threshold parameters for the sleeping touch sensor, which can be configured as follows:

- Configure LP_ANA_TOUCH_SLP_PAD to select a particular touch sensor as the sleeping touch sensor.
- Configure LP_ANA_TOUCH_SLP_TH0, LP_ANA_TOUCH_SLP_TH1, and LP_ANA_TOUCH_SLP_TH2 to set the touch threshold for the sleeping touch sensor at different frequency modes.
- Configure LP_ANA_TOUCH_SLP_APPROACH_EN to enable the proximity mode for the sleeping touch sensor.
- Configure LP_ANA_TOUCH_SLP_CHANNEL_CLR to clear the intermediate data for the sleeping touch sensor, such as benchmark, touch_raw_data, and the number of sampling.

Note:
The configuration for the noise_threshold, hysteresis, touch_smooth_data, and benchmark values are the same for the sleeping touch sensor in sleep mode as in the normal mode.

63.4.6 Moisture Tolerance

If there are water droplets on the touch pads and the droplets are large enough to physically bridge two more adjacent touch pads, the adjacent touch pads may be electrically coupled. Coupled touch pads can cause false touch detection due to capacitance changes caused by the coupling. If a water droplet is connected across the touch pad and ground, a large parasitic capacitance will be introduced, also leading to false detection. The moisture tolerance feature can mitigate the impact of water droplets.

To configure moisture tolerance:

- Configure LP_ANA_TOUCH_SHIELD_PAD_EN to enable moisture tolerance.
- Configure LP_ANA_TOUCH_BUFSEL to select a touch pin to be used for moisture tolerance.

Note:
The moisture tolerance feature only works at frequencies below 10 MHz. Therefore, when the frequency is higher than 10 MHz, please disable the moisture tolerance feature.

63.4.7 Water Rejection

If the sensor array becomes wet, i.e., the majority of the sensor pads are soaked by water, most (if not all) of the touch pads will become unusable due to the possible false touch detections. ESP32-P4 supports the water rejection feature to shut down the sensor array if it is detected to be wet.

Configure LP_ANA_TOUCH_OUT_RING to select one of the touch pins to be used for the water rejection feature.

63.5 Interrupts

ESP32-P4's touch sensor can generate the LP_TOUCH_INTR signal that will be sent to the Interrupt Matrix.

There are several internal interrupt sources from touch sensor that can generate the LP_TOUCH_INTR signal. The interrupt sources from touch sensor are listed with their trigger conditions and the resulted interrupt signal in Table 63.5-1.

Table 63.5-1. Touch Sensor's Internal Interrupt Sources

Internal Interrupt Source	Trigger Condition	Interrupt Signal
TOUCH_APPROACH_LOOP_DONE_INT	Completion of proximity mode cumulative sampling	LP_TOUCH_INTR
TOUCH_TIMEOUT_INT	Timeout for the TOUCH_OUT signal sampling	
TOUCH_INACTIVE_INT	Touch release detected	
TOUCH_ACTIVE_INT	Touch detected	
TOUCH_DONE_INT	Completion of sampling an individual touch pin at a certain frequency mode for comparison	
TOUCH_SCAN_DONE_INT	Completion of sampling selected touch pins for comparison at all frequency modes	
TOUCH_BENCHMARK_UPDATE_INT	Completion of software benchmark data update	

Note:
For definitions of *interrupt*, *interrupt signal*, *interrupt source*, and their correlations, please refer to Chapter 13 *Interrupt Matrix* > Section 13.2 *Interrupt Terminology in ESP32-P4*.

Each interrupt source can be configured by a common set of registers that are described in Section *Interrupt Configuration Registers*. The specific registers can be found in Section 63.6 *Register Summary*.

63.6 Register Summary

63.6.1 Interrupt and Status Register Summary

The addresses in this section are relative to Touch Sensor base address provided in Table 9.3-2 in Chapter 9 *System and Memory*.

The abbreviations given in Column **Access** are explained in Section *Access Types for Registers*.

Name	Description	Address	Access
Interrupt Register			
RTC_TOUCH_INT_RAW_REG	Touch sensor interrupt raw register	0x0000	R/SS/WTC
RTC_TOUCH_INT_ST_REG	Touch sensor interrupt status register	0x0004	RO
RTC_TOUCH_INT_ENA_REG	Touch sensor interrupt enable register	0x0008	R/W
RTC_TOUCH_INT_CLR_REG	Touch sensor interrupt clear register	0x000C	WT
Status Register			
RTC_TOUCH_CHN_STATUS_REG	Scan mode status register	0x0010	RO
RTC_TOUCH_STATUS_1_REG	Touch pin 1 status register	0x0018	RO
RTC_TOUCH_STATUS_2_REG	Touch pin 2 status register	0x001C	RO
RTC_TOUCH_STATUS_3_REG	Touch pin 3 status register	0x0020	RO
RTC_TOUCH_STATUS_4_REG	Touch pin 4 status register	0x0024	RO
RTC_TOUCH_STATUS_5_REG	Touch pin 5 status register	0x0028	RO
RTC_TOUCH_STATUS_6_REG	Touch pin 6 status register	0x002C	RO
RTC_TOUCH_STATUS_7_REG	Touch pin 7 status register	0x0030	RO
RTC_TOUCH_STATUS_8_REG	Touch pin 8 status register	0x0034	RO
RTC_TOUCH_STATUS_9_REG	Touch pin 9 status register	0x0038	RO
RTC_TOUCH_STATUS_10_REG	Touch pin 10 status register	0x003C	RO
RTC_TOUCH_STATUS_11_REG	Touch pin 11 status register	0x0040	RO
RTC_TOUCH_STATUS_12_REG	Touch pin 12 status register	0x0044	RO
RTC_TOUCH_STATUS_13_REG	Touch pin 13 status register	0x0048	RO
RTC_TOUCH_STATUS_14_REG	Touch pin 14 status register	0x004C	RO
RTC_TOUCH_STATUS_15_REG	Sleep mode touch pin status register	0x0050	RO

Name	Description	Address	Access
RTC_TOUCH_STATUS_16_REG	Proximity mode status register	0x0054	RO
RTC_TOUCH_STATUS_17_REG	Frequency hopping status register	0x0058	RO
RTC_TOUCH_CHN_TMP_STATUS_REG	Touch status register	0x005C	RO
Version Control Register			
RTC_TOUCH_DATE_REG	Version control register	0x0100	R/W

63.6.2 Configuration Register Summary

The addresses in this section are relative to LP Analog Peripheral base address provided in Table 9.3-2 in Chapter 9 *System and Memory*.

The abbreviations given in Column **Access** are explained in Section *Access Types for Registers*.

Name	Description	Address	Access
Configuration Register			
LP_ANA_TOUCH_APPROACH_WORK_MEAS_NUM_REG	Configuration register for the number of measurements in proximity mode	0x00FC	R/W
LP_ANA_TOUCH_SCAN_CTRL1_REG	Scan mode configuration register 1	0x0100	R/W
LP_ANA_TOUCH_SCAN_CTRL2_REG	Scan mode configuration register 2	0x0104	R/W
LP_ANA_TOUCH_WORK_REG	Sample preprocessing register	0x0108	varies
LP_ANA_TOUCH_WORK_MEAS_NUM_REG	Configuration register for number of measurements at different frequency modes	0x010C	R/W
LP_ANA_TOUCH_FILTER1_REG	Touch detection configuration register 1	0x0110	R/W
LP_ANA_TOUCH_FILTER2_REG	Touch detection configuration register 2	0x0114	R/W
LP_ANA_TOUCH_FILTER3_REG	Touch detection configuration register 3	0x0118	varies
LP_ANA_TOUCH_SLP0_REG	Sleep mode configuration register 1	0x011C	varies
LP_ANA_TOUCH_SLP1_REG	Sleep mode configuration register 2	0x0120	R/W
LP_ANA_TOUCH_CLR_REG	Status clear register	0x0124	WT
LP_ANA_TOUCH_APPROACH_REG	Proximity mode configuration register	0x0128	R/W
LP_ANA_TOUCH_FREQ0_SCAN_PARA_REG	Analog parameter configuration register for touch sensor for frequency mode 0	0x012C	R/W
LP_ANA_TOUCH_FREQ1_SCAN_PARA_REG	Analog parameter configuration register for touch sensor for frequency mode 1	0x0130	R/W

GoBack

Name	Description	Address	Access
LP_ANA_TOUCH_FREQ2_SCAN_PARA_REG	Analog parameter configuration register for touch sensor for frequency mode 2	0x0134	R/W
LP_ANA_TOUCH_ANA_PARA_REG	Touch sensor analog parameter configuration register	0x0138	R/W
LP_ANA_TOUCH_MUX0_REG	DEBUG mode selection configuration register	0x013C	R/W
LP_ANA_TOUCH_MUX1_REG	DEBUG mode function configuration register	0x0140	R/W
LP_ANA_TOUCH_PAD1_TH0_REG	Touch threshold configuration register for touch pin 1 for frequency mode 0	0x0150	R/W
LP_ANA_TOUCH_PAD1_TH1_REG	Touch threshold configuration register for touch pin 1 for frequency mode 1	0x0154	R/W
LP_ANA_TOUCH_PAD1_TH2_REG	Touch threshold configuration register for touch pin 1 for frequency mode 2	0x0158	R/W
LP_ANA_TOUCH_PAD2_TH0_REG	Touch threshold configuration register for touch pin 2 for frequency mode 0	0x015C	R/W
LP_ANA_TOUCH_PAD2_TH1_REG	Touch threshold configuration register for touch pin 2 for frequency mode 1	0x0160	R/W
LP_ANA_TOUCH_PAD2_TH2_REG	Touch threshold configuration register for touch pin 2 for frequency mode 2	0x0164	R/W
LP_ANA_TOUCH_PAD3_TH0_REG	Touch threshold configuration register for touch pin 3 for frequency mode 0	0x0168	R/W
LP_ANA_TOUCH_PAD3_TH1_REG	Touch threshold configuration register for touch pin 3 for frequency mode 1	0x016C	R/W
LP_ANA_TOUCH_PAD3_TH2_REG	Touch threshold configuration register for touch pin 3 for frequency mode 2	0x0170	R/W
LP_ANA_TOUCH_PAD4_TH0_REG	Touch threshold configuration register for touch pin 4 for frequency mode 0	0x0174	R/W
LP_ANA_TOUCH_PAD4_TH1_REG	Touch threshold configuration register for touch pin 4 for frequency mode 1	0x0178	R/W
LP_ANA_TOUCH_PAD4_TH2_REG	Touch threshold configuration register for touch pin 4 for frequency mode 2	0x017C	R/W
LP_ANA_TOUCH_PAD5_TH0_REG	Touch threshold configuration register for touch pin 5 for frequency mode 0	0x0180	R/W
LP_ANA_TOUCH_PAD5_TH1_REG	Touch threshold configuration register for touch pin 5 for frequency mode 1	0x0184	R/W
LP_ANA_TOUCH_PAD5_TH2_REG	Touch threshold configuration register for touch pin 5 for frequency mode 2	0x0188	R/W
LP_ANA_TOUCH_PAD6_TH0_REG	Touch threshold configuration register for touch pin 6 for frequency mode 0	0x018C	R/W
LP_ANA_TOUCH_PAD6_TH1_REG	Touch threshold configuration register for touch pin 6 for frequency mode 1	0x0190	R/W
LP_ANA_TOUCH_PAD6_TH2_REG	Touch threshold configuration register for touch pin 6 for frequency mode 2	0x0194	R/W
LP_ANA_TOUCH_PAD7_TH0_REG	Touch threshold configuration register for touch pin 7 for frequency mode 0	0x0198	R/W
LP_ANA_TOUCH_PAD7_TH1_REG	Touch threshold configuration register for touch pin 7 for frequency mode 1	0x019C	R/W
LP_ANA_TOUCH_PAD7_TH2_REG	Touch threshold configuration register for touch pin 7 for frequency mode 2	0x01A0	R/W
LP_ANA_TOUCH_PAD8_TH0_REG	Touch threshold configuration register for touch pin 8 for frequency mode 0	0x01A4	R/W
LP_ANA_TOUCH_PAD8_TH1_REG	Touch threshold configuration register for touch pin 8 for frequency mode 1	0x01A8	R/W
LP_ANA_TOUCH_PAD8_TH2_REG	Touch threshold configuration register for touch pin 8 for frequency mode 2	0x01AC	R/W

Name	Description	Address	Access
LP_ANA_TOUCH_PAD9_TH0_REG	Touch threshold configuration register for touch pin 9 for frequency mode 0	0x01B0	R/W
LP_ANA_TOUCH_PAD9_TH1_REG	Touch threshold configuration register for touch pin 9 for frequency mode 1	0x01B4	R/W
LP_ANA_TOUCH_PAD9_TH2_REG	Touch threshold configuration register for touch pin 9 for frequency mode 2	0x01B8	R/W
LP_ANA_TOUCH_PAD10_TH0_REG	Touch threshold configuration register for touch pin 10 for frequency mode 0	0x01BC	R/W
LP_ANA_TOUCH_PAD10_TH1_REG	Touch threshold configuration register for touch pin 10 for frequency mode 1	0x01C0	R/W
LP_ANA_TOUCH_PAD10_TH2_REG	Touch threshold configuration register for touch pin 10 for frequency mode 2	0x01C4	R/W
LP_ANA_TOUCH_PAD11_TH0_REG	Touch threshold configuration register for touch pin 11 for frequency mode 0	0x01C8	R/W
LP_ANA_TOUCH_PAD11_TH1_REG	Touch threshold configuration register for touch pin 11 for frequency mode 1	0x01CC	R/W
LP_ANA_TOUCH_PAD11_TH2_REG	Touch threshold configuration register for touch pin 11 for frequency mode 2	0x01D0	R/W
LP_ANA_TOUCH_PAD12_TH0_REG	Touch threshold configuration register for touch pin 12 for frequency mode 0	0x01D4	R/W
LP_ANA_TOUCH_PAD12_TH1_REG	Touch threshold configuration register for touch pin 12 for frequency mode 1	0x01D8	R/W
LP_ANA_TOUCH_PAD12_TH2_REG	Touch threshold configuration register for touch pin 12 for frequency mode 2	0x01DC	R/W
LP_ANA_TOUCH_PAD13_TH0_REG	Touch threshold configuration register for touch pin 13 for frequency mode 0	0x01E0	R/W
LP_ANA_TOUCH_PAD13_TH1_REG	Touch threshold configuration register for touch pin 13 for frequency mode 1	0x01E4	R/W
LP_ANA_TOUCH_PAD13_TH2_REG	Touch threshold configuration register for touch pin 13 for frequency mode 2	0x01E8	R/W
LP_ANA_TOUCH_PAD14_TH0_REG	Touch threshold configuration register for touch pin 14 for frequency mode 0	0x01EC	R/W
LP_ANA_TOUCH_PAD14_TH1_REG	Touch threshold configuration register for touch pin 14 for frequency mode 1	0x01F0	R/W
LP_ANA_TOUCH_PAD14_TH2_REG	Touch threshold configuration register for touch pin 14 for frequency mode 2	0x01F4	R/W
LP_ANA_DATE_REG	Version control register	0x03FC	R/W

GoBack

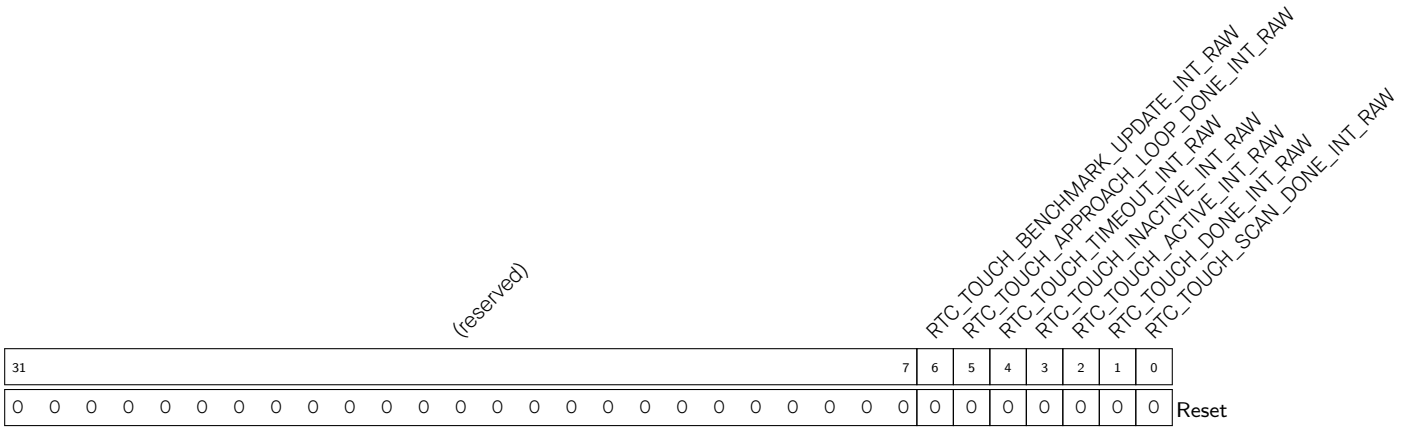
63.7 Registers

63.7.1 Interrupt and Status Registers

The addresses in this section are relative to Touch Sensor base address provided in Table 9.3-2 in Chapter 9 *System and Memory*.

For how to program reserved fields, please refer to Section *Programming Reserved Register Field*.

Register 63.1. RTC_TOUCH_INT_RAW_REG (0x0000)



RTC_TOUCH_SCAN_DONE_INT_RAW The raw interrupt status of TOUCH_SCAN_DONE_INT. (R/SS/WTC)

RTC_TOUCH_DONE_INT_RAW The raw interrupt status of TOUCH_DONE_INT. (R/SS/WTC)

RTC_TOUCH_ACTIVE_INT_RAW The raw interrupt status of TOUCH_ACTIVE_INT. (R/SS/WTC)

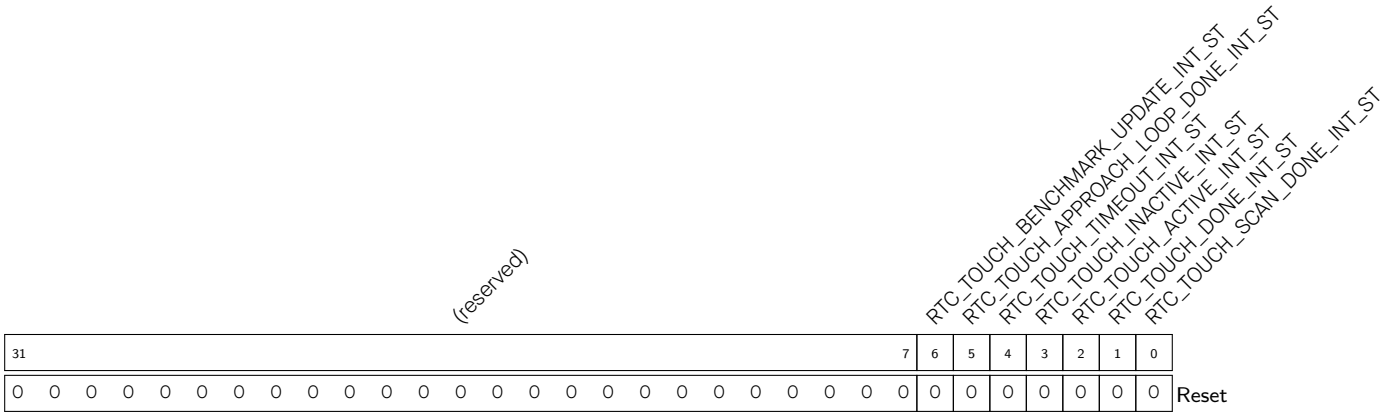
RTC_TOUCH_INACTIVE_INT_RAW The raw interrupt status of TOUCH_INACTIVE_INT. (R/SS/WTC)

RTC_TOUCH_TIMEOUT_INT_RAW The raw interrupt status of TOUCH_TIMEOUT_INT. (R/SS/WTC)

RTC_TOUCH_APPROACH_LOOP_DONE_INT_RAW The raw interrupt status of TOUCH_APPROACH_LOOP_DONE_INT. (R/SS/WTC)

RTC_TOUCH_BENCHMARK_UPDATE_INT_RAW The raw interrupt status of TOUCH_BENCHMARK_UPDATE_INT. (R/SS/WTC)

Register 63.2. RTC_TOUCH_INT_ST_REG (0x0004)



RTC_TOUCH_SCAN_DONE_INT_ST The masked interrupt status of TOUCH_DONE_INT. (RO)

RTC_TOUCH_DONE_INT_ST The masked interrupt status of TOUCH_DONE_INT. (RO)

RTC_TOUCH_ACTIVE_INT_ST The masked interrupt status of TOUCH_ACTIVE_INT. (RO)

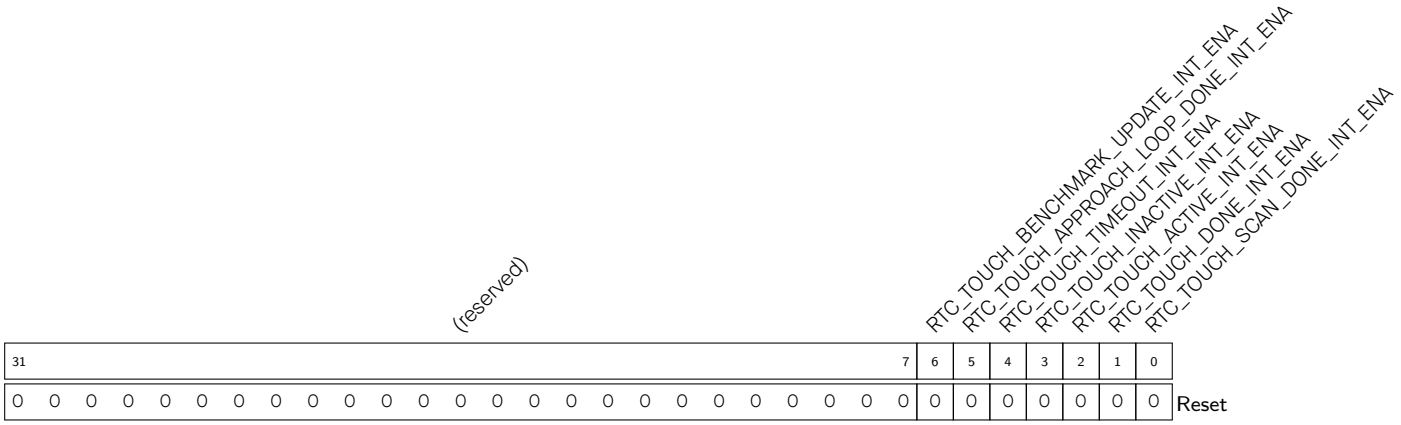
RTC_TOUCH_INACTIVE_INT_ST The masked interrupt status of TOUCH_INACTIVE_INT. (RO)

RTC_TOUCH_TIMEOUT_INT_ST The masked interrupt status of TOUCH_TIMEOUT_INT. (RO)

RTC_TOUCH_APPROACH_LOOP_DONE_INT_ST The masked interrupt status of TOUCH_APPROACH_LOOP_DONE_INT. (RO)

RTC_TOUCH_BENCHMARK_UPDATE_INT_ST The masked interrupt status of TOUCH_BENCHMARK_UPDATE_INT. (RO)

Register 63.3. RTC_TOUCH_INT_ENA_REG (0x0008)



RTC_TOUCH_SCAN_DONE_INT_ENA Write 1 to enable TOUCH_DONE_INT. (R/W)

RTC_TOUCH_DONE_INT_ENA Write 1 to enable TOUCH_DONE_INT. (R/W)

RTC_TOUCH_ACTIVE_INT_ENA Write 1 to enable TOUCH_ACTIVE_INT. (R/W)

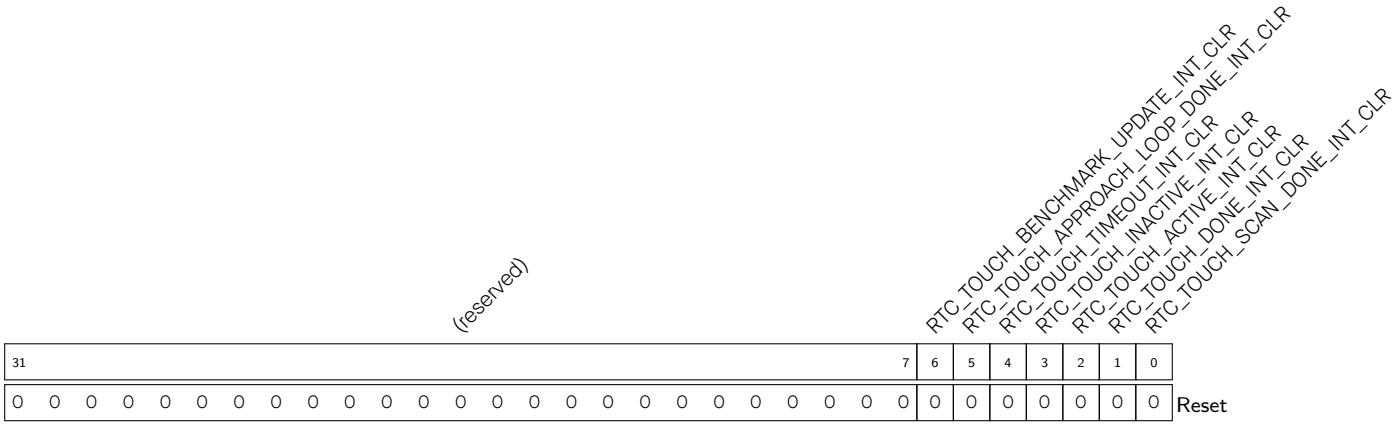
RTC_TOUCH_INACTIVE_INT_ENA Write 1 to enable TOUCH_INACTIVE_INT. (R/W)

RTC_TOUCH_TIMEOUT_INT_ENA Write 1 to enable TOUCH_TIMEOUT_INT. (R/W)

RTC_TOUCH_APPROACH_LOOP_DONE_INT_ENA Write 1 to enable TOUCH_APPROACH_LOOP_DONE_INT. (R/W)

RTC_TOUCH_BENCHMARK_UPDATE_INT_ENA Write 1 to enable TOUCH_BENCHMARK_UPDATE_INT. (R/W)

Register 63.4. RTC_TOUCH_INT_CLR_REG (0x000C)



RTC_TOUCH_SCAN_DONE_INT_CLR Write 1 to clear TOUCH_DONE_INT. (WT)

RTC_TOUCH_DONE_INT_CLR Write 1 to clear TOUCH_DONE_INT. (WT)

RTC_TOUCH_ACTIVE_INT_CLR Write 1 to clear TOUCH_ACTIVE_INT. (WT)

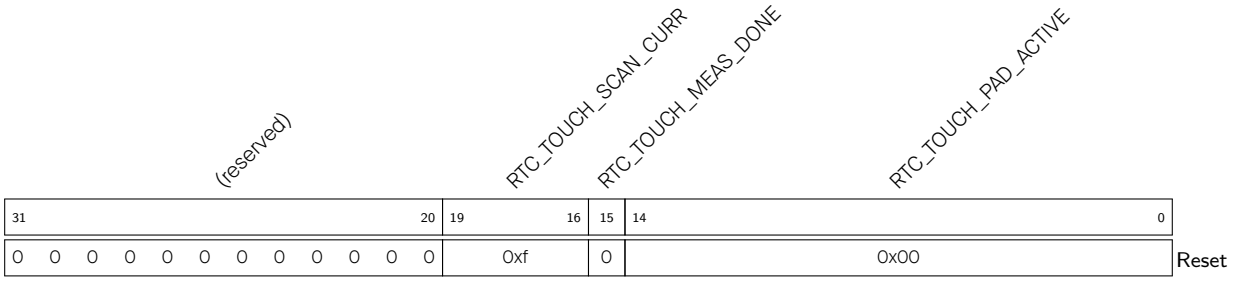
RTC_TOUCH_INACTIVE_INT_CLR Write 1 to clear TOUCH_INACTIVE_INT. (WT)

RTC_TOUCH_TIMEOUT_INT_CLR Write 1 to clear TOUCH_TIMEOUT_INT. (WT)

RTC_TOUCH_APPROACH_LOOP_DONE_INT_CLR Write 1 to clear TOUCH_APPROACH_LOOP_DONE_INT. (WT)

RTC_TOUCH_BENCHMARK_UPDATE_INT_CLR Write 1 to clear TOUCH_BENCHMARK_UPDATE_INT. (WT)

Register 63.5. RTC_TOUCH_CHN_STATUS_REG (0x0010)



RTC_TOUCH_PAD_ACTIVE Indicates whether the touch pin detects a touch. Bit 1-14 corresponds to touch pin 1-14 respectively. Other bits are invalid.

0: No touch detected

1: Touch detected

(RO)

RTC_TOUCH_MEAS_DONE Indicates whether the touch pins have been measured at all frequency modes.

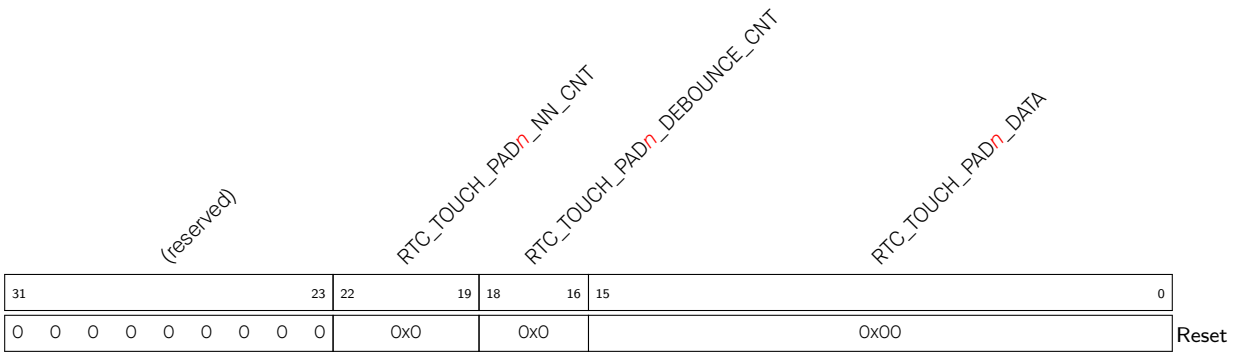
0: Not completed

1: Completed

(RO)

RTC_TOUCH_SCAN_CURR Indicates the index number of the currently scanned touch pin. (RO)

Register 63.6. RTC_TOUCH_STATUS_*n*_REG (*n*: 1-14) (0x0018+0x4*(*n*-1))

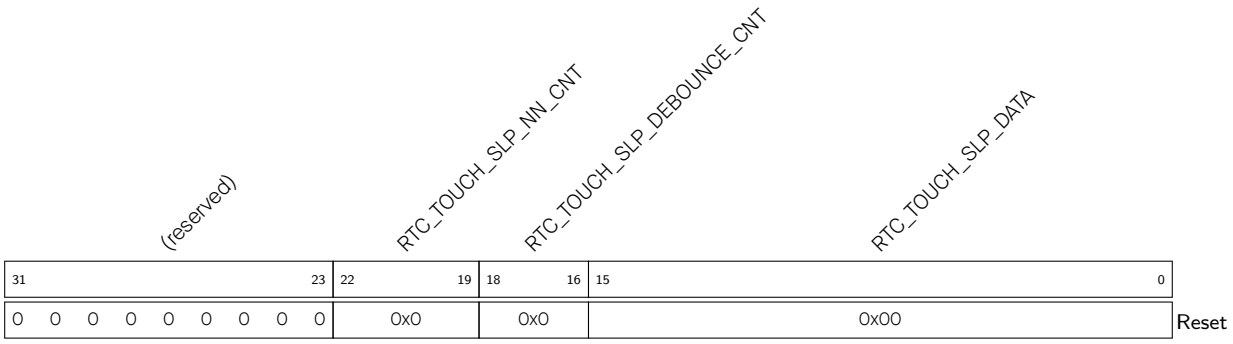


RTC_TOUCH_PAD*n*_DATA Indicates the touch_raw_data, touch_smooth_data, or benchmark sampled by touch pin *n*. (RO)

RTC_TOUCH_PAD*n*_DEBOUNCE_CNT Indicates the cumulative number of consecutive touch or touch release detection by touch pin *n*. (RO)

RTC_TOUCH_PAD*n*_NN_CNT Indicates the cumulative number of consecutive instances where $touch_smooth_data < benchmark - passive_noise_threshold$ has been detected by touch pin *n*. (RO)

Register 63.7. RTC_TOUCH_STATUS_15_REG (0x0050)

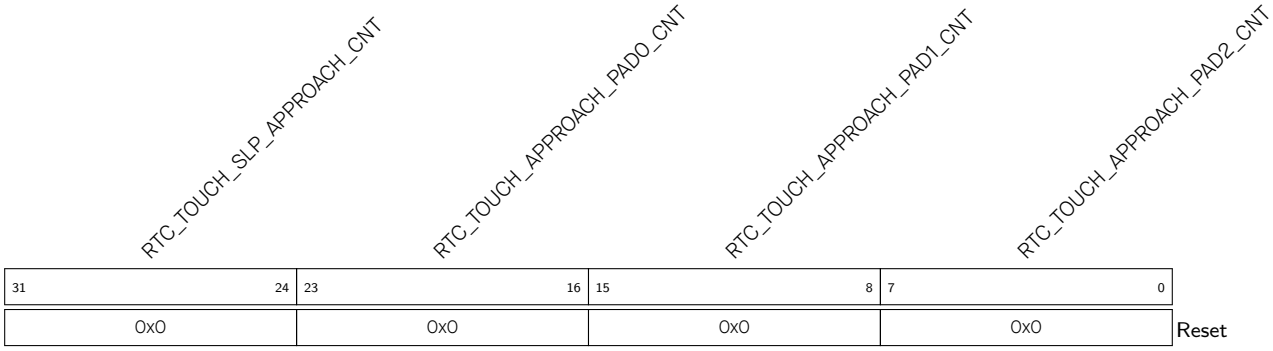


RTC_TOUCH_SLP_DATA Indicates the touch_smooth_data or benchmark sampled by touch pin in sleep mode. (RO)

RTC_TOUCH_SLP_DEBOUNCE_CNT Indicates the cumulative number of consecutive touch or touch release detection by touch pin in sleep mode. (RO)

RTC_TOUCH_SLP_NN_CNT Indicates the cumulative number of consecutive instances where $touch_smooth_data < benchmark - passive_noise_threshold$ has been detected by touch pin in sleep mode. (RO)

Register 63.8. RTC_TOUCH_STATUS_16_REG (0x0054)



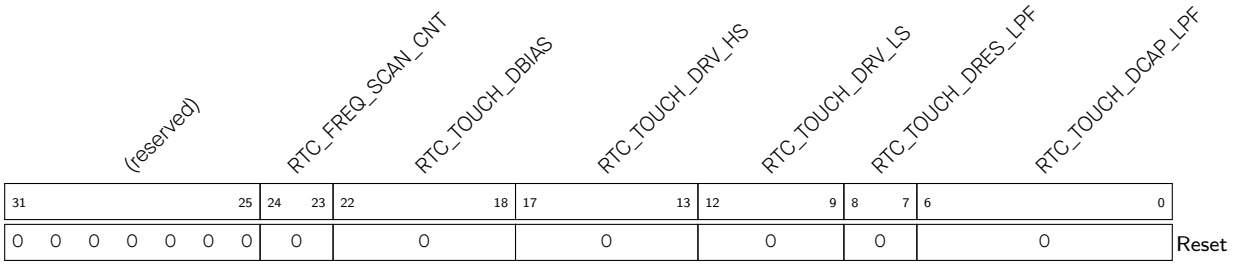
RTC_TOUCH_APPROACH_PAD2_CNT Indicates the cumulative number of samplings of touch pin 2 in proximity mode. (RO)

RTC_TOUCH_APPROACH_PAD1_CNT Indicates the cumulative number of samplings of touch pin 1 in proximity mode. (RO)

RTC_TOUCH_APPROACH_PAD0_CNT Indicates the cumulative number of samplings of touch pin 0 in proximity mode. (RO)

RTC_TOUCH_SLP_APPROACH_CNT Indicates the cumulative number of samplings of the sleeping touch pin in proximity mode. (RO)

Register 63.9. RTC_TOUCH_STATUS_17_REG (0x0058)



RTC_TOUCH_DCAP_LPF Indicates the DCAP_LPF value of the current touch pin. (RO)

RTC_TOUCH_DRES_LPF Indicates the DRES_LPF value of the current touch pin. (RO)

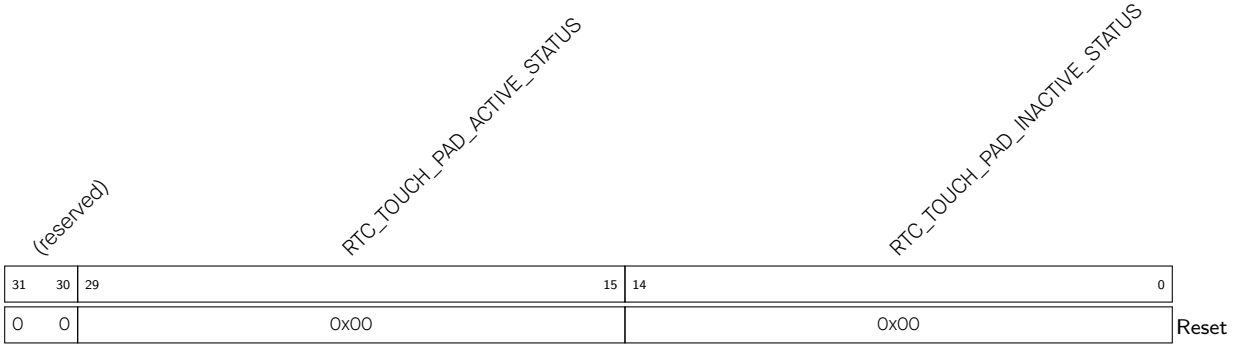
RTC_TOUCH_DRV_LS Indicates the DRV_LS value of the current touch pin. (RO)

RTC_TOUCH_DRV_HS Indicates the DRV_HS value of the current touch pin. (RO)

RTC_TOUCH_DBIAS Indicates the DBIAS value of the current touch pin. (RO)

RTC_FREQ_SCAN_CNT Indicates the SCAN_CNT value of the current touch pin. (RO)

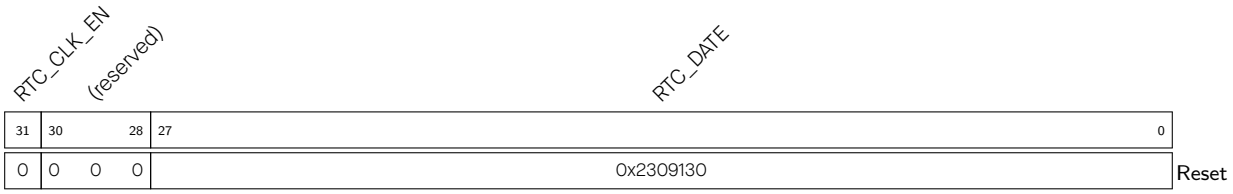
Register 63.10. RTC_TOUCH_CHN_TMP_STATUS_REG (0x005C)



RTC_TOUCH_PAD_INACTIVE_STATUS Indicates whether the touch pin detects the touch pad from being touched to being released. Bit 1-14 corresponds to touch pin 1-14 respectively. Other bits are invalid.
0: Touch release detected
1: Touch release not detected
(RO)

RTC_TOUCH_PAD_ACTIVE_STATUS Indicates whether the touch pin detects the process of being touched. Bit 1-14 corresponds to touch pin 1-14 respectively. Other bits are invalid.
0: Touch detected
1: Touch not detected
(RO)

Register 63.11. RTC_TOUCH_DATE_REG (0x0100)



RTC_CLK_EN Configures whether to enable the register read/write clock.

0: Disable

1: Enable

(R/W)

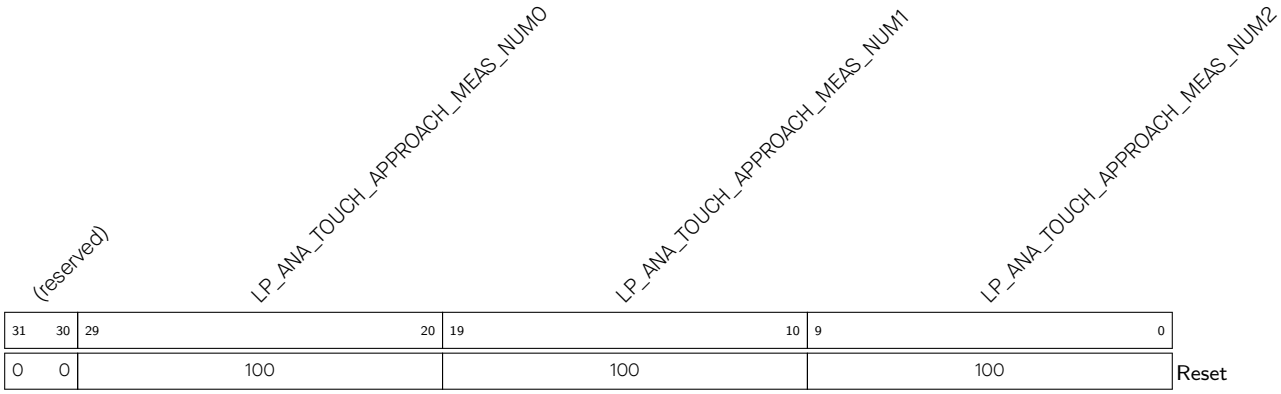
RTC_DATE Version control register. (R/W)

63.7.2 Configuration Registers

The addresses in this section are relative to LP Analog Peripheral base address provided in Table 9.3-2 in Chapter 9 *System and Memory*.

For how to program reserved fields, please refer to Section *Programming Reserved Register Field*.

Register 63.12. LP_ANA_TOUCH_APPROACH_WORK_MEAS_NUM_REG (0x00FC)

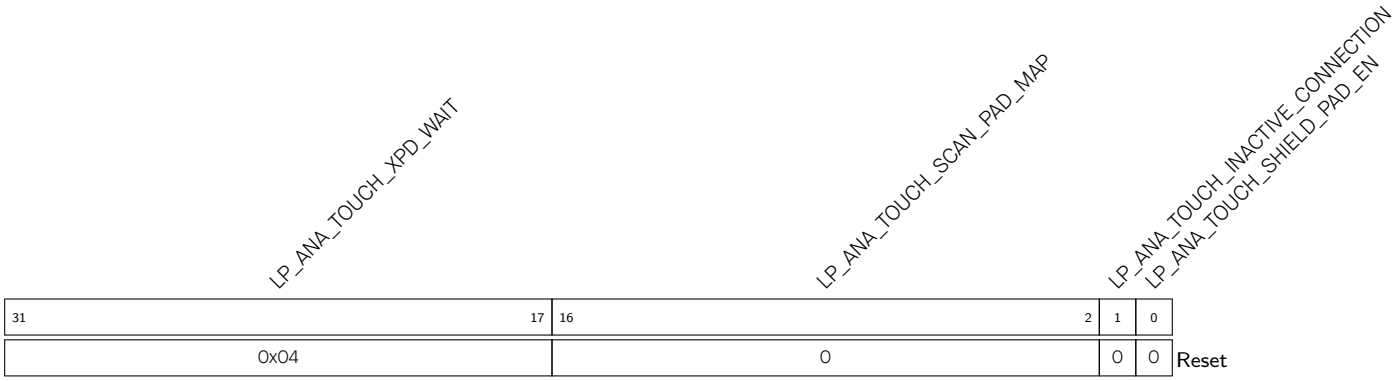


LP_ANA_TOUCH_APPROACH_MEAS_NUM2 Configures the number of measurements for sampling frequency mode 2 in proximity mode. (R/W)

LP_ANA_TOUCH_APPROACH_MEAS_NUM1 Configures the number of measurements for sampling frequency mode 1 in proximity mode. (R/W)

LP_ANA_TOUCH_APPROACH_MEAS_NUM0 Configures the number of measurements for sampling frequency mode 0 in proximity mode. (R/W)

Register 63.13. LP_ANA_TOUCH_SCAN_CTRL1_REG (0x0100)



LP_ANA_TOUCH_SHIELD_PAD_EN Configures whether to enable the moisture tolerance feature.

0: Disable

1: Enable

(R/W)

LP_ANA_TOUCH_INACTIVE_CONNECTION Configures whether to power up the touch pins involved in the measurement in scan mode.

0: Not power up

1: Power up

(R/W)

LP_ANA_TOUCH_SCAN_PAD_MAP Configures which touch pins to use in scan mode for measurement. Bit 1-14 corresponds to touch pin 1-14 respectively. Other bits are invalid.

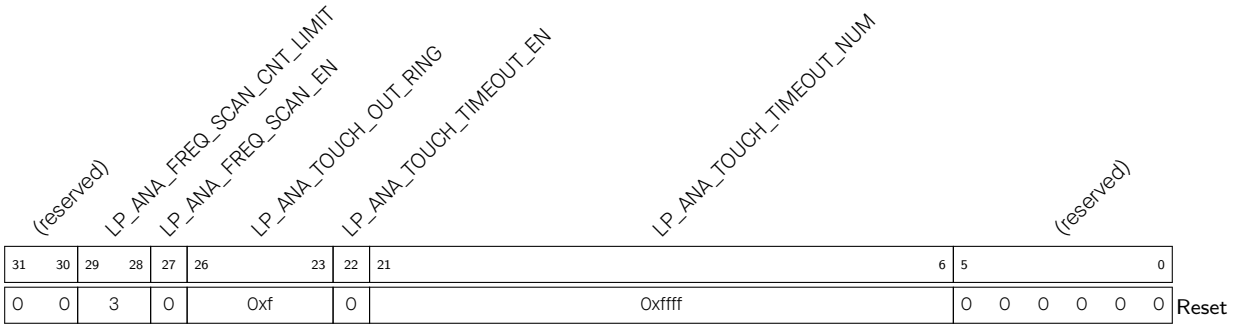
0: No effect

1: Enable the touch pin

(R/W)

LP_ANA_TOUCH_XPD_WAIT Configures the power-up wait time after initiating a measurement. (R/W)

Register 63.14. LP_ANA_TOUCH_SCAN_CTRL2_REG (0x0104)



LP_ANA_TOUCH_TIMEOUT_NUM Configures the number of measurement timeouts. (R/W)

LP_ANA_TOUCH_TIMEOUT_EN Configures whether to enable the measurement timeout.

0: Disable

1: Enable

(R/W)

LP_ANA_TOUCH_OUT_RING Configures which touch pins to be used for water rejection. Configurable values are 1-14. Other values are invalid. (R/W)

LP_ANA_FREQ_SCAN_EN Configures whether to enable frequency hopping.

0: Disable

1: Enable

(R/W)

LP_ANA_FREQ_SCAN_CNT_LIMIT Configures the number of frequency hops.

0: 1

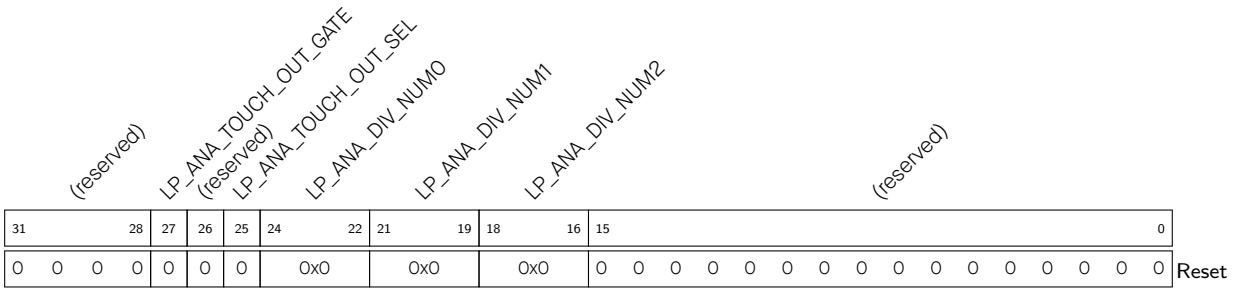
1: 2

2: 3

3: Invalid

(R/W)

Register 63.15. LP_ANA_TOUCH_WORK_REG (0x0108)



LP_ANA_DIV_NUM2 Configures the enabling of the TOUCH_OUT signal division and the coefficient for sampling frequency mode 0.

bit[0]: Configures whether to enable division.

0: Disable

1: Enable

bit[1-2]: Configures the TOUCH_OUT signal division coefficient.

0: 0

1: 2

2: 4

3: 6

(R/W)

LP_ANA_DIV_NUM1 Configures the enabling of the TOUCH_OUT signal division and the coefficient for sampling frequency mode 1. For detailed configuration, refer to LP_ANA_DIV_NUM2. (R/W)

LP_ANA_DIV_NUM0 Configures the enabling of the TOUCH_OUT signal division and the coefficient for sampling frequency mode 2. For detailed configuration, refer to LP_ANA_DIV_NUM2. (R/W)

LP_ANA_TOUCH_OUT_SEL Configures whether to use the TOUCH_OUT signal for clock or data.

0: TOUCH_OUT signal is used as data

1: TOUCH_OUT signal is used as clock

(R/W)

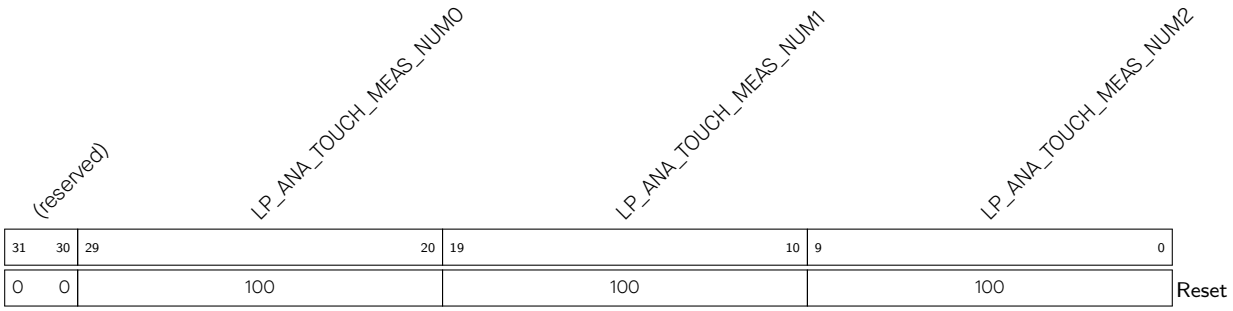
LP_ANA_TOUCH_OUT_GATE Configures whether to enable gating after enabling the TOUCH_OUT signal division.

0: Disable

1: Enable

(R/W)

Register 63.16. LP_ANA_TOUCH_WORK_MEAS_NUM_REG (0x010C)

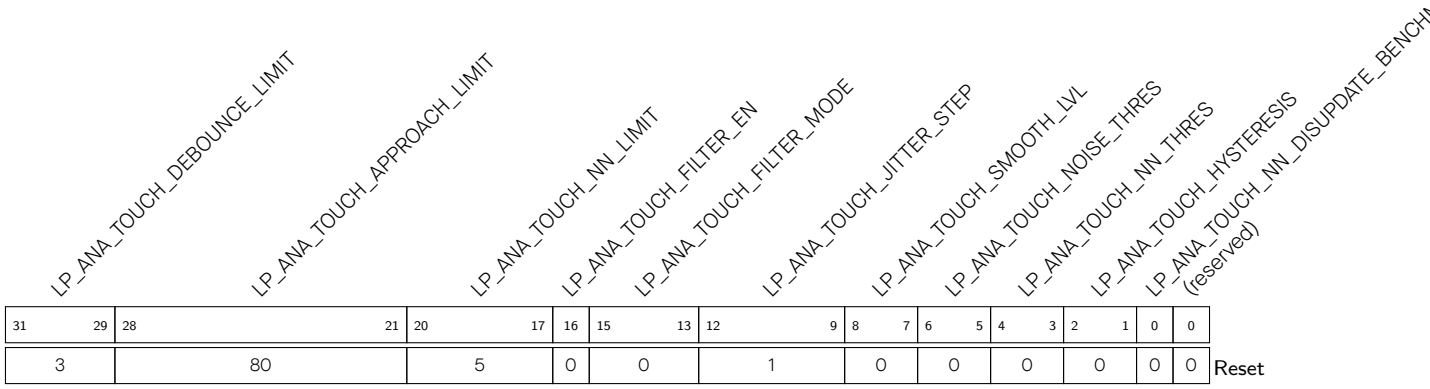


LP_ANA_TOUCH_MEAS_NUM2 Configures the number of measurements for sampling frequency mode 2 in normal operation. (R/W)

LP_ANA_TOUCH_MEAS_NUM1 Configures the number of measurements for sampling frequency mode 1 in normal operation. (R/W)

LP_ANA_TOUCH_MEAS_NUM0 Configures the number of measurements for sampling frequency mode 0 in normal operation.(R/W)

Register 63.17. LP_ANA_TOUCH_FILTER1_REG (0x0110)



LP_ANA_TOUCH_NN_DISUPDATE_BENCHMARK_EN Configures whether to turn off the benchmark data correction.
0: Do not turn off
1: Turn off
(R/W)

LP_ANA_TOUCH_HYSTERESIS Configures the sampling hysteresis. (R/W)

LP_ANA_TOUCH_NN_THRES Configures the out-of-phase noise_threshold. (R/W)

LP_ANA_TOUCH_NOISE_THRES Configures the noise_threshold. (R/W)

LP_ANA_TOUCH_SMOOTH_LVL Configures the IIR filter module for touch_smooth_data.(R/W)

LP_ANA_TOUCH_JITTER_STEP Configures the jitter step in filter module 7 for generating benchmark data. (R/W)

LP_ANA_TOUCH_FILTER_MODE Configures the filter module for generating benchmark data. (R/W)

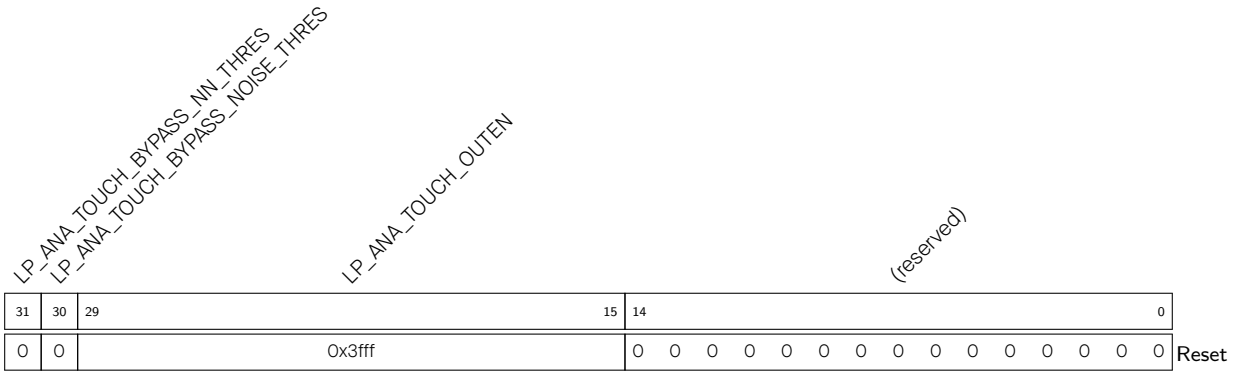
LP_ANA_TOUCH_FILTER_EN Configures whether to enable the filter module for generating benchmark data.
0: Disable
1: Enable
(R/W)

LP_ANA_TOUCH_NN_LIMIT Configures the number of consecutive sampling instances where the sampling value exceeds the out-of-phase noise_threshold. (R/W)

LP_ANA_TOUCH_APPROACH_LIMIT Configures the number of sampling in proximity mode. (R/W)

LP_ANA_TOUCH_DEBOUNCE_LIMIT Configures the number of consecutive sampling for detecting touch state changes. (R/W)

Register 63.18. LP_ANA_TOUCH_FILTER2_REG (0x0114)



LP_ANA_TOUCH_OUTEN Configures whether to enable output of the 14 touch pin measurement results. Bit 1-14 corresponds to touch pin 1-14 respectively. Other bits are invalid.

0: Disable

1: Enable

(R/W)

LP_ANA_TOUCH_BYPASS_NOISE_THRES Configures whether to enable the bypass mode for touch_smooth_data.

0: Disable

1: Enable

(R/W)

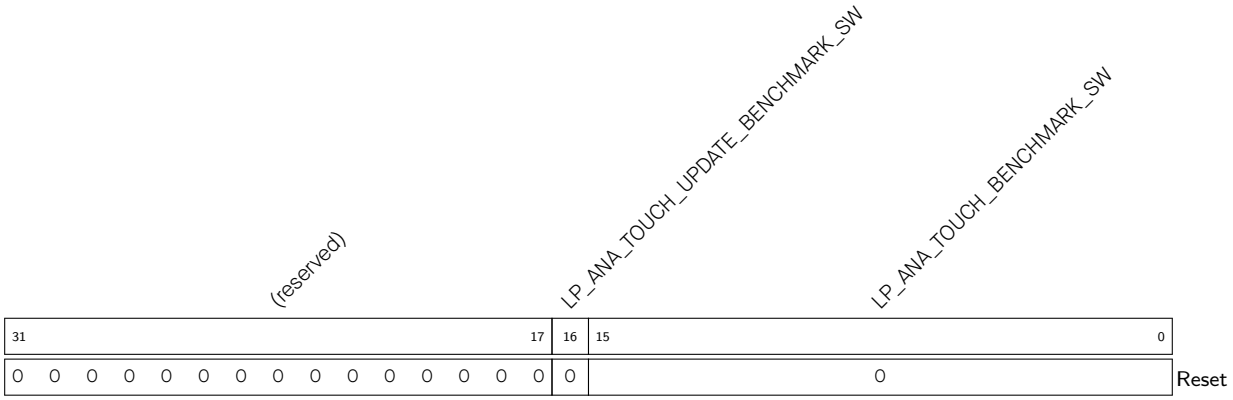
LP_ANA_TOUCH_BYPASS_NN_THRES Configures whether to enable the bypass mode for out-of-phase noise_threshold.

0: Disable

1: Enable

(R/W)

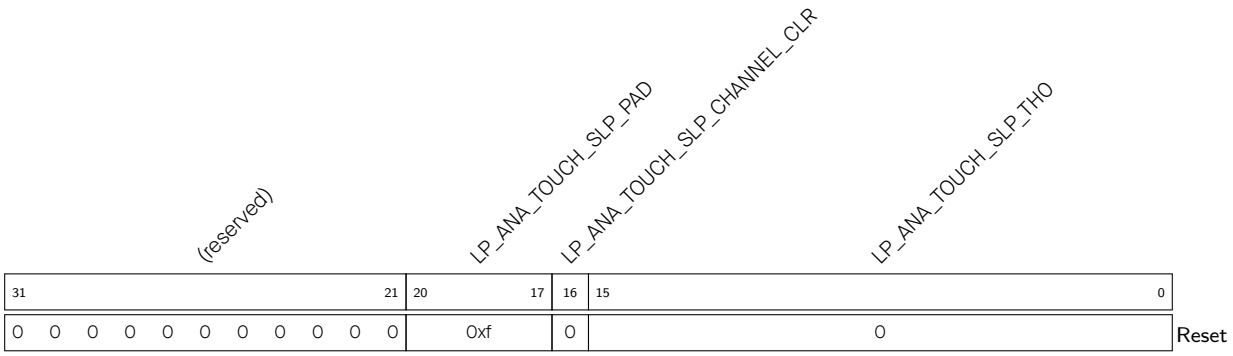
Register 63.19. LP_ANA_TOUCH_FILTER3_REG (0x0118)



LP_ANA_TOUCH_BENCHMARK_SW Configures software to write the benchmark data. (R/W)

LP_ANA_TOUCH_UPDATE_BENCHMARK_SW Write 1 to enable the software to write the benchmark data. (WT)

Register 63.20. LP_ANA_TOUCH_SLP0_REG (0x011C)



LP_ANA_TOUCH_SLP_TH0 Configures the touch threshold for sampling frequency mode 0 in sleep mode. (R/W)

LP_ANA_TOUCH_SLP_CHANNEL_CLR Write 1 to clear the benchmark data in sleep mode. (WT)

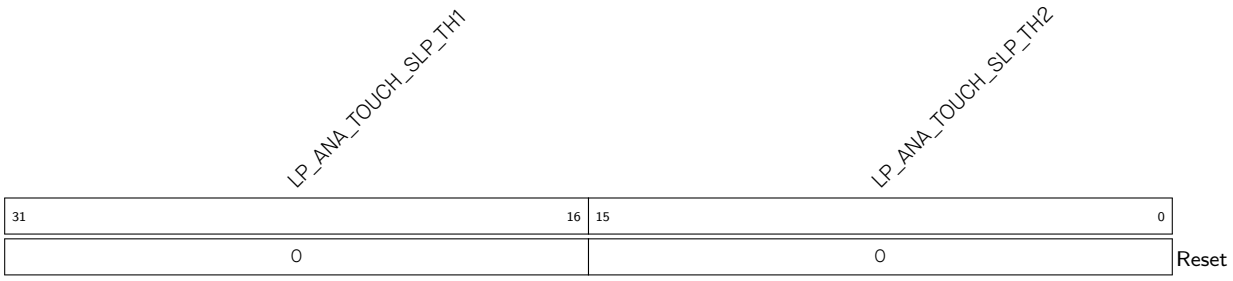
LP_ANA_TOUCH_SLP_PAD Configures which touch sensor to enter sleep mode. Bit 1-14 corresponds to touch pin 1-14 respectively. Other bits are invalid.

0: No effect

1: Selects the touch sensor to enter sleep mode

(R/W)

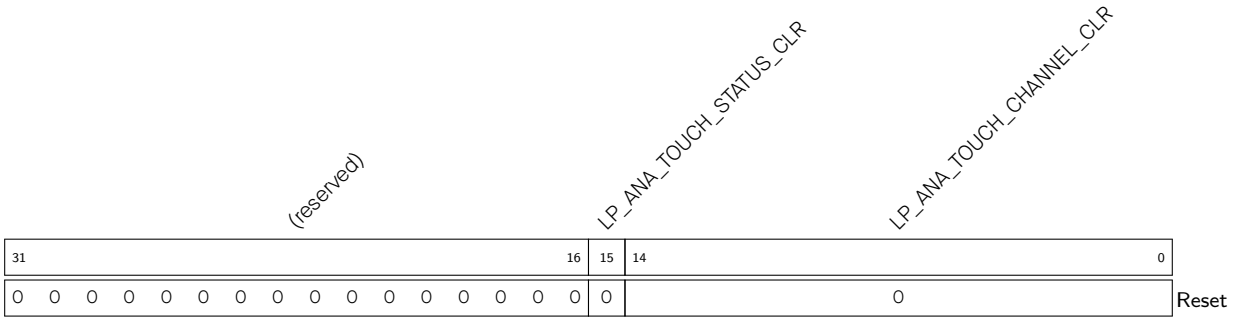
Register 63.21. LP_ANA_TOUCH_SLP1_REG (0x0120)



LP_ANA_TOUCH_SLP_TH2 Configures the touch threshold for sampling frequency mode 2 in sleep mode. (R/W)

LP_ANA_TOUCH_SLP_TH1 Configures the touch threshold for sampling frequency mode 1 in sleep mode. (R/W)

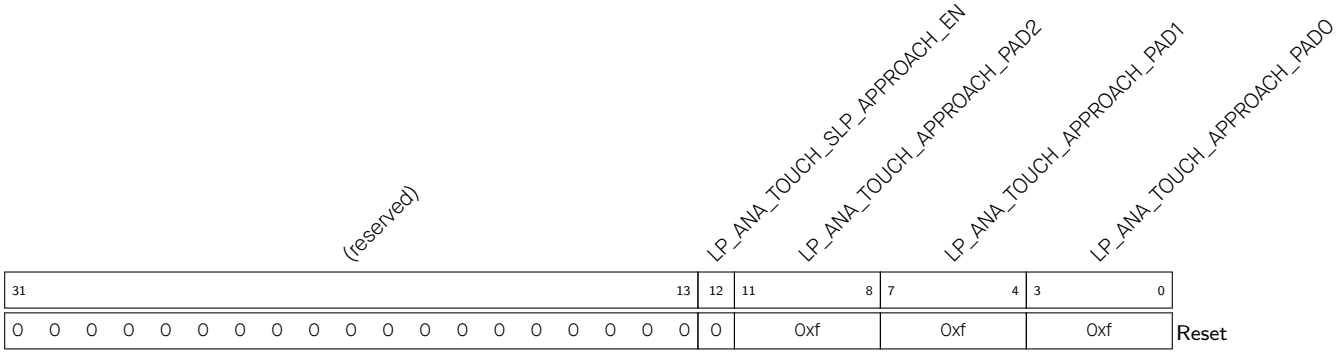
Register 63.22. LP_ANA_TOUCH_CLR_REG (0x0124)



LP_ANA_TOUCH_CHANNEL_CLR Write 1 to independently clear the touch state of the 14 touch sensors. Bit 1-14 corresponds to touch pin 1-14 respectively. Other bits are invalid. (WT)

LP_ANA_TOUCH_STATUS_CLR Write 1 to clear the touch state. (WT)

Register 63.23. LP_ANA_TOUCH_APPROACH_REG (0x0128)



LP_ANA_TOUCH_APPROACH_PAD0 Configure the first touch pin used in proximity mode. Configurable values are 1-14. Other values are invalid. (R/W)

LP_ANA_TOUCH_APPROACH_PAD1 Configure the second touch pin used in proximity mode. Configurable values are 1-14. Other values are invalid. (R/W)

LP_ANA_TOUCH_APPROACH_PAD2 Configure the third touch pin used in proximity mode. Configurable values are 1-14. Other values are invalid. (R/W)

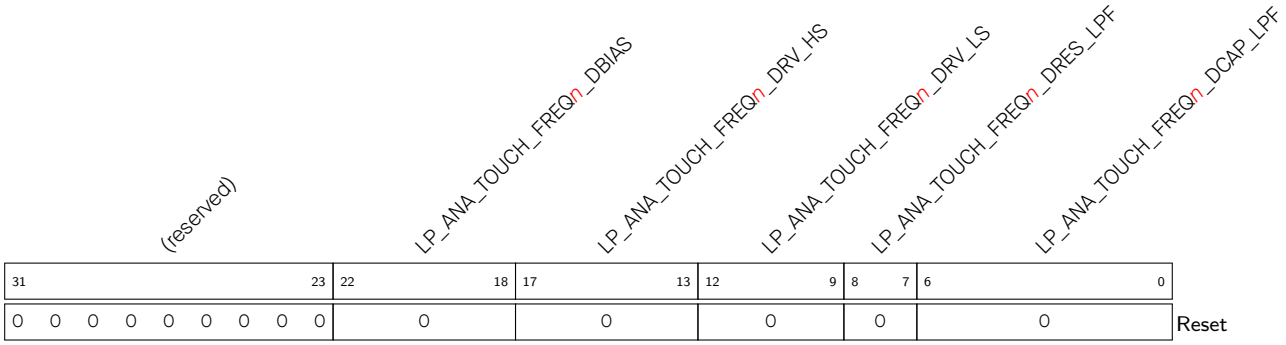
LP_ANA_TOUCH_SLP_APPROACH_EN Configures whether to enable proximity mode.

0: Disable

1: Enable

(R/W)

Register 63.24. LP_ANA_TOUCH_FREQ*n*_SCAN_PARA_REG (*n*: 0-2) (0x012C+0x4n*)**



LP_ANA_TOUCH_FREQ*n*_DCAP_LPF Configures the touch sensor low-pass filter for sampling frequency mode *n*. The adjustment range is 0-2.54 pF, and the step is 20 fF. (R/W)(R/W)

LP_ANA_TOUCH_FREQ*n*_DRES_LPF Configures the touch sensor low-pass filter for sampling frequency mode *n*.

0: 0 kΩ

1: 3 kΩ

2: 4.5 kΩ

3: 7.5 kΩ

(R/W)

LP_ANA_TOUCH_FREQ*n*_DRV_LS Configures the value of DRV_LS of the touch sensor for sampling frequency mode *n*. (R/W)

LP_ANA_TOUCH_FREQ*n*_DRV_HS Configures the value of DRV_HS of the touch sensor for sampling frequency mode *n*. (R/W)

LP_ANA_TOUCH_FREQ*n*_DBIAS Configures whether to enable the moisture tolerance function and configures the internal voltage of the touch sensor for sampling frequency mode *n*.

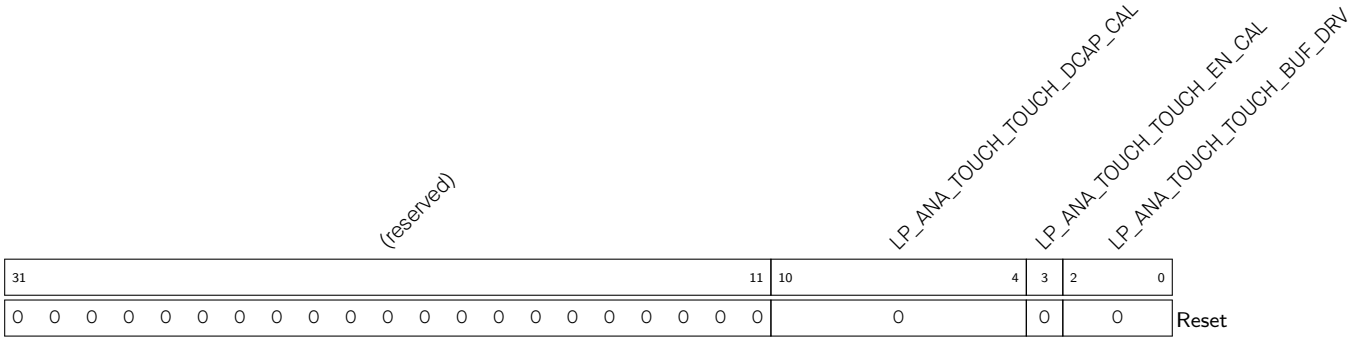
bit[0]: Configures whether to enable the moisture tolerance function for sampling frequency mode *n*.

0: Disable

1: Enable

bit[1-4]: Configures the internal voltage of the touch sensor for sampling frequency mode *n*. (R/W)

Register 63.25. LP_ANA_TOUCH_ANA_PARA_REG (0x0138)



LP_ANA_TOUCH_TOUCH_BUF_DRV Configures the value of BUF_DRV. (R/W)

LP_ANA_TOUCH_TOUCH_EN_CAL Configures whether to enable the touch sensor internal capacitance test mode.

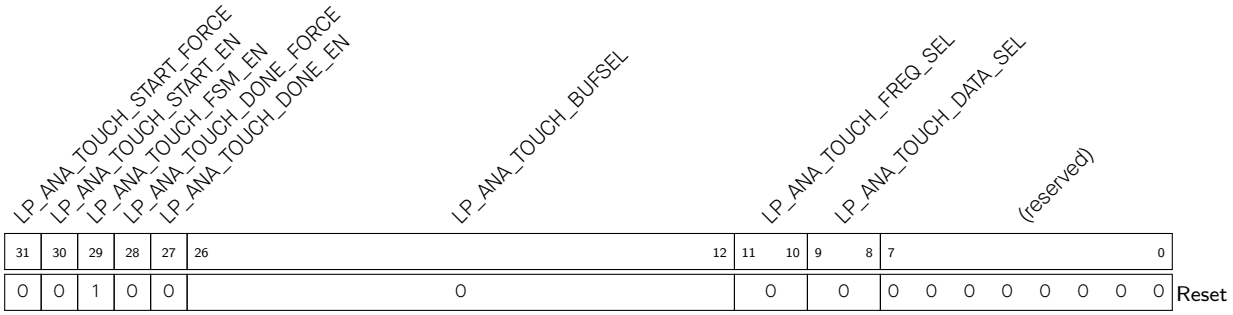
0: Disable

1: Enable

(R/W)

LP_ANA_TOUCH_TOUCH_DCAP_CAL Configures the internal capacitance connected to the touch pins. The default value is 0, and the adjustment range is 0-127 pF with a step of 1. (R/W)

Register 63.26. LP_ANA_TOUCH_MUX0_REG (0x013C)



LP_ANA_TOUCH_DATA_SEL Configures the type of the return value.

0/1: touch_raw_data

2: benchmark

3: touch_smooth_data

(R/W)

LP_ANA_TOUCH_FREQ_SEL Configures the corresponding sampling frequency mode of the return value.

0: 0

1: 1

2: 2

3: Invalid

(R/W)

LP_ANA_TOUCH_BUFSEL Configures whether to enable a touch pin to be used for moisture tolerance. Bit 1-14 corresponds to touch pin 1-14 respectively. Other bits are invalid.

0: Disable

1: Enable

(R/W)

LP_ANA_TOUCH_DONE_EN Configures whether to enable software to trigger the DONE signal that enables the measurement.

0: Disable

1: Enable

(R/W)

LP_ANA_TOUCH_DONE_FORCE Configures whether to generate the DONE signal to end the current measurement operation.

0: Not generate

1: Generate

(R/W)

LP_ANA_TOUCH_FSM_EN Configures the control source to power up and activate touch pins.

0: Controlled by software

1: Controlled by hardware

(R/W)

Continued on the next page...

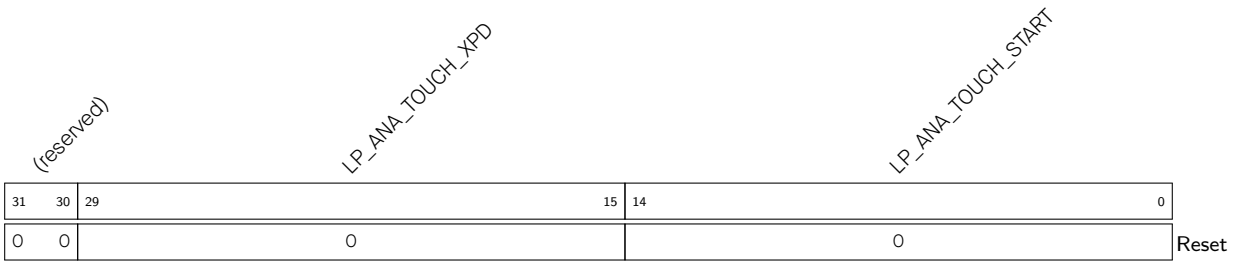
Register 63.26. LP_ANA_TOUCH_MUX0_REG (0x013C)

Continued from the previous page...

LP_ANA_TOUCH_START_EN Configures whether to enable software to trigger the START signal to start the measurement.
0: Disable
1: Enable
(R/W)

LP_ANA_TOUCH_START_FORCE Configures whether to generate the DONE signal to start a measurement.
0: Not generate
1: Generate
(R/W)

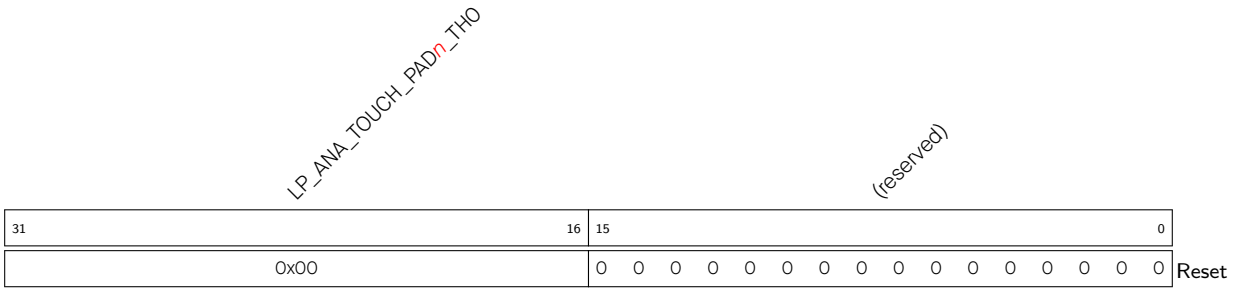
Register 63.27. LP_ANA_TOUCH_MUX1_REG (0x0140)



LP_ANA_TOUCH_START Configures whether to activate the 14 touch pins. Bit 1-14 corresponds to touch pin 1-14 respectively. Other bits are invalid.
0: No effect
1: Activate
(R/W)

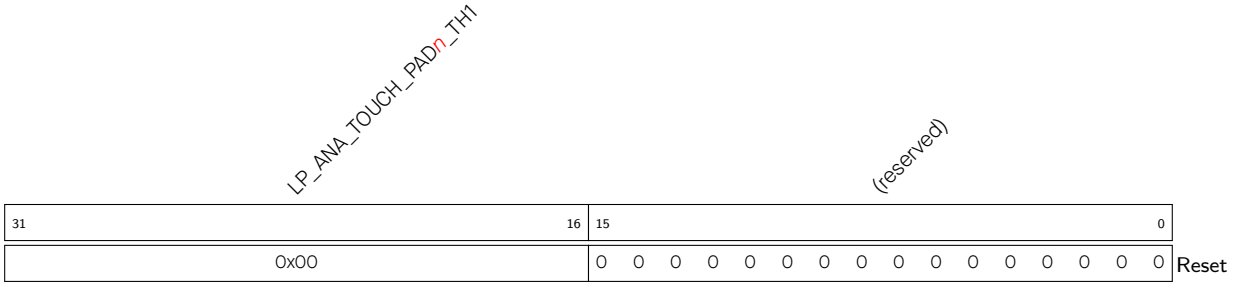
LP_ANA_TOUCH_XPD Configures whether to power up the 14 touch pins. Bit 1-14 corresponds to touch pin 1-14 respectively. Other bits are invalid.
0: Not power up
1: Power up
(R/W)

Register 63.28. LP_ANA_TOUCH_PADn_TH0_REG (n: 1-14) (0x0150+0xC*(n-1))



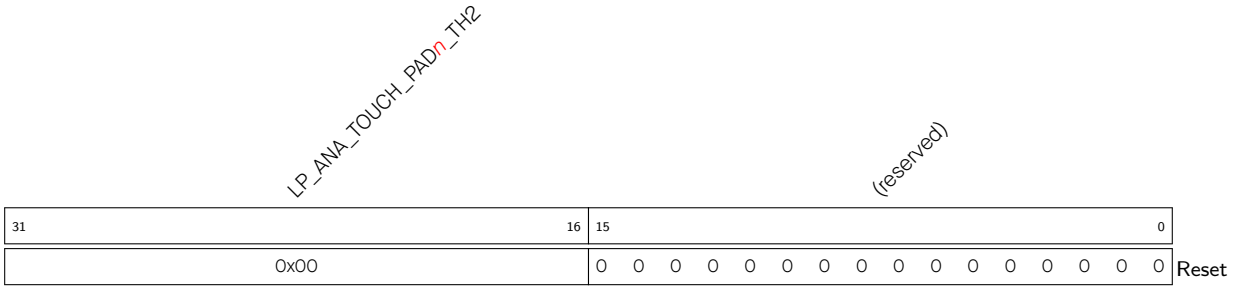
LP_ANA_TOUCH_PADn_TH0 Configures the touch threshold for touch sensor *n* for sampling frequency mode 0. (R/W)

Register 63.29. LP_ANA_TOUCH_PADn_TH1_REG (n: 1-14) (0x0154+0xC*(n-1))



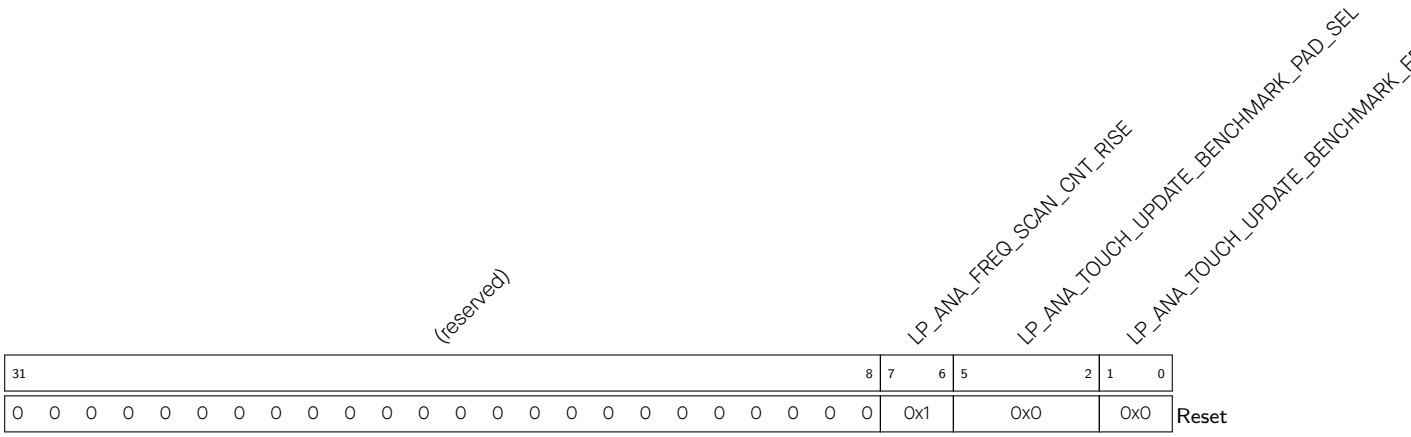
LP_ANA_TOUCH_PADn_TH1 Configures the touch threshold of touch sensor *n* for sampling frequency mode 1. (R/W)

Register 63.30. LP_ANA_TOUCH_PADn_TH2_REG (n: 1-14) (0x0158+0xC*(n-1))



LP_ANA_TOUCH_PADn_TH2 Configures the touch threshold of touch sensor *n* for sampling frequency mode 2. (R/W)

Register 63.31. LP_ANA_TOUCH_CTRL_REG (0x01FC)

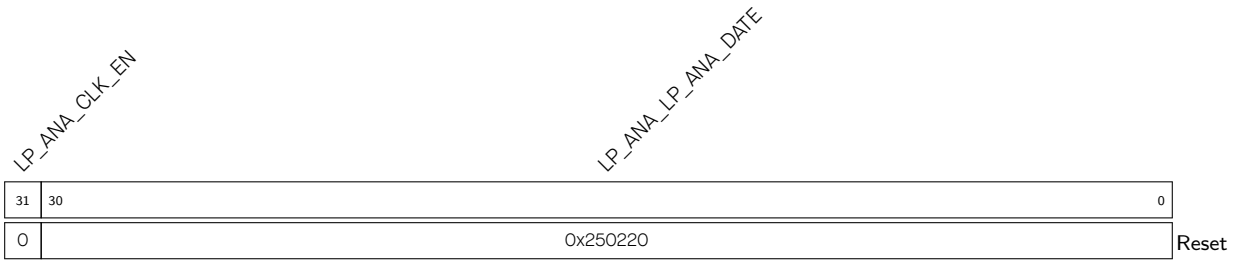


LP_ANA_TOUCH_UPDATE_BENCHMARK_FREQ_SEL Configures the frequency point used by software to update the benchmark. (R/W)

LP_ANA_TOUCH_UPDATE_BENCHMARK_PAD_SEL Configures the channel used by software to update the benchmark. (R/W)

LP_ANA_FREQ_SCAN_CNT_RISE Configures the number of hit frequency points to be evaluated for touch detection in frequency hopping mode. (R/W)

Register 63.32. LP_ANA_DATE_REG (0x03FC)



LP_ANA_LP_ANA_DATE Version control register. (R/W)

LP_ANA_CLK_EN Configures whether to enable the clock for accessing the registers.

0: Disable

1: Enable

(R/W)

Chapter 64

Temperature Sensor (TSENS)

64.1 Overview

ESP32-P4 provides a temperature sensor for real-time monitoring of temperature changes within the chip. The sensor converts analog voltage to digital values and provides compensation for temperature offsets.

64.2 Features

The temperature sensor has the following features:

- Software-triggered temperature measurement, which once triggered, the sensor continuously measures temperature. Software can read the data at any time.
- Hardware-triggered automatic temperature monitoring, supporting two wake-up modes
- Configurable temperature offset based on the application scenario for improved accuracy
- Configurable temperature measurement range
- Support for Event Task Matrix (ETM)-related events and tasks

64.3 Architecture

Figure 64.3-1 shows the internal structure of the temperature sensor.

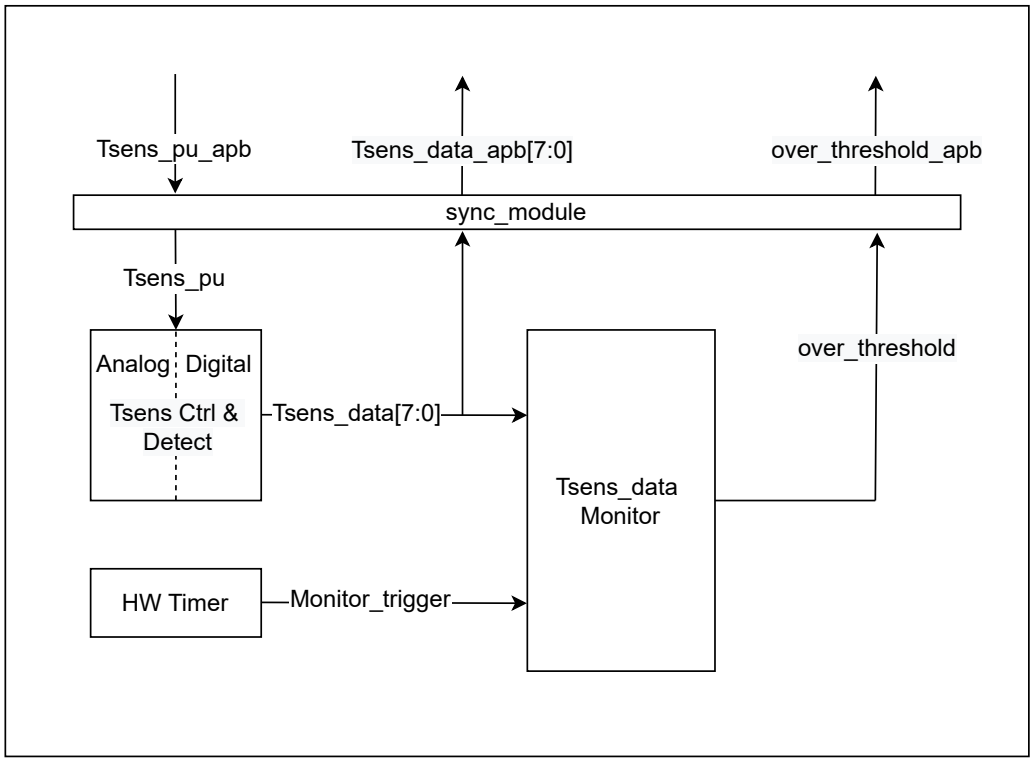


Figure 64.3-1. Temperature Sensor Architecture

As Figure 64.3-1 shows, the temperature sensor module contains the following major blocks:

- Tsens Ctrl & Detect: temperature sensor
- HW Timer: triggers automatic temperature monitoring
- Tsens_data Monitor: monitors whether the temperature is outside the threshold range
- sync_module: Synchronization block between APB clock domain and temperature sensor clock domain

64.4 Functional Description

64.4.1 Temperature Sensor Power Up

The temperature sensor can be powered up by setting the TSENS_POWER_UP register.

64.4.2 Temperature Sensor Clock

The temperature sensor has only one clock source: LP_PERI_CLK, and there is no frequency divider inside the module.

64.4.3 Wake-Up Modes for Automatic Temperature Monitoring

There are two wake-up modes for temperature monitoring, selected by TSENS_WAKEUP_MODE:

- Absolute value mode:

  - Monitors the absolute value of the current temperature. Configure TSENS_WAKEUP_TH_LOW and TSENS_WAKEUP_TH_HIGH to set the temperature thresholds. Wake-up will be triggered if the sampled value is above the high threshold or below the low threshold.

- Change value mode:
  - Monitors the temperature changes inside the chip. If the temperature increment of two consecutive samplings exceeds the high threshold configured in TSENS_WAKEUP_TH_HIGH, or the temperature decrement of two consecutive samplings exceeds the low threshold configured in TSENS_WAKEUP_TH_LOW, a wake-up will be triggered. For example, when TSENS_WAKEUP_TH_LOW is set to 8, and the two consecutive sampling values are 28 and 19, resulting in a temperature decrement of 9, a wake-up will be triggered.

64.4.4 Temperature Measurement Range and Offset

The temperature sensor is designed with five measurement ranges, as displayed in Table 64.4-1, to enhance accuracy. Each range has its own measurement errors, but users can select particular offsets to obtain calibrated data, so there is no need to be concerned about the errors.

Table 64.4-1. Temperature Measurement Range and Offset

Temperature Measurement Range (°C)	Temperature Offset
50 ~ 125	−2
20 ~ 100	−1
−10 ~ 80	0
−30 ~ 50	1
−40 ~ 20	2

64.4.5 Data Conversion

The sensor output value ($VALUE$) is stored in TSENS_OUT. To calculate the actual temperature T (°C) based on $VALUE$, use the following equation:

$$T = 0.4386 * VALUE - 27.88 * offset - 20.52$$

where $offset$ is the temperature offset displayed in Table 64.4-1.

64.5 Event Task Matrix Feature

The temperature sensor on ESP32-P4 supports the Event Task Matrix (ETM) function, which allows the temperature sensor's ETM tasks to be triggered by any peripherals' ETM events, or temperature sensor's ETM events to trigger any peripherals' ETM tasks. This section introduces the ETM tasks and events related to the temperature sensor. For more information, please refer to Chapter 14 Event Task Matrix (ETM).

The temperature sensor can receive the following ETM tasks:

- TMPSNSR_TASK_START_SAMPLE: The temperature sensor starts sampling when this task is triggered.
- TMPSNSR_TASK_STOP_SAMPLE: The temperature sensor stops sampling when this task is triggered.

The temperature sensor can generate the following ETM events:

- TMPSNSR_EVT_OVER_LIMIT: Generated when the temperature is beyond the threshold.

In practical applications, the temperature sensor's ETM events can trigger its own ETM tasks. For example, the TMPSNSR_EVT_OVER_LIMIT event can trigger the TMPSNSR_TASK_STOP_SAMPLE task.

64.6 Interrupts

ESP32-P4's Temperature Sensor can generate the following interrupt signal that will be sent to the Interrupt Matrix.

- LP_TSENS_INTR

There is one internal interrupt source from the Temperature Sensor that can generate the above interrupt signal:

- TSENS_COCPU_TSENS_WAKE_INT: Triggered when the temperature sample value exceeds the threshold.

Note:
For definitions of *interrupt*, *interrupt signal*, *interrupt source*, and their correlations, please refer to Chapter 13 *Interrupt Matrix* > Section 13.2 *Interrupt Terminology in ESP32-P4*.

Each interrupt source can be configured by a common set of registers that are described in Section *Interrupt Configuration Registers*. The specific registers can be found in Section 64.8 *Register Summary*.

64.7 Programming Procedure

The temperature sensor can be started by software as follows:

1. Set TSENS_POWER_UP to power up the temperature sensor.
2. Set LPPERI_CK_EN_LP_TSENS to enable the temperature sensor clock.
3. Wait for TSENS_XPD_WAIT clock cycles till the temperature sensor releases from reset and starts measuring the temperature.
4. Wait for 100 μs (so that the output value gradually approaches the actual temperature) and then read the data from TSENS_OUT.

To enable hardware-triggered automatic temperature monitoring, add the following programming steps before powering up the sensor:

1. Configure TSENS_SAMPLE_RATE to set sampling rate.
2. Configure TSENS_WAKEUP_MODE to select wake-up mode for temperature monitoring.
3. Configure TSENS_WAKEUP_TH_HIGH/LOW to set the high and low temperature monitoring thresholds.
4. Set TSENS_WAKEUP_EN to start temperature monitoring.
5. Set TSENS_SAMPLE_EN to enable automatic temperature monitoring.

In automatic temperature monitoring mode, the output value will not be saved. Nevertheless, users can get the value at any moment by accessing TSENS_OUT.

64.8 Register Summary

The addresses in this section are relative to Temperature Sensor base address provided in Table 9.3-2 in Chapter 9 *System and Memory*.

The abbreviations given in Column **Access** are explained in Section *Access Types for Registers*.

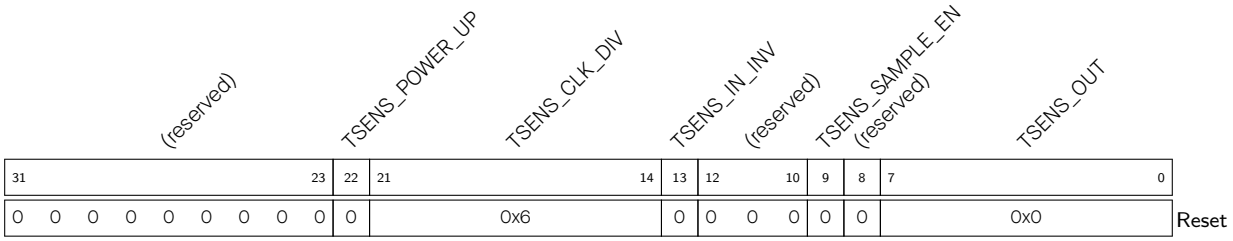
Name	Description	Address	Access
Control Registers			
TSENS_CTRL_REG	Temperature sensor control register 1.	0x0000	varies
TSENS_CTRL2_REG	Temperature sensor control register 2.	0x0004	R/W
Interrupt Registers			
TSENS_INT_RAW_REG	Raw register of temperature sensor interrupt.	0x0008	R/WTC/SS
TSENS_INT_ST_REG	State register of temperature sensor interrupt.	0x000C	RO
TSENS_INT_ENA_REG	Enable register of temperature sensor interrupt.	0x0010	R/WTC
TSENS_INT_CLR_REG	Clear register of temperature sensor interrupt.	0x0014	WT
TSENS_INT_ENA_W1TS_REG	TSENS_INT_ENA_REG configuration register	0x001C	WT
TSENS_INT_ENA_W1TC_REG	TSENS_INT_ENA_REG configuration register	0x0020	WT
Wake-up Control Registers			
TSENS_WAKEUP_CTRL_REG	Temperature sensor wake-up control registers.	0x0024	varies
TSENS_SAMPLE_RATE_REG	Hardware automatic sampling control registers.	0x0028	R/W

64.9 Registers

The addresses in this section are relative to Temperature Sensor base address provided in Table 9.3-2 in Chapter 9 *System and Memory*.

For how to program reserved fields, please refer to Section Programming Reserved Register Field.

Register 64.1. TSENS_CTRL_REG (0x0000)



TSENS_OUT Stores the temperature sensor output value. (RO)

TSENS_SAMPLE_EN Configures whether to enable automatic temperature monitoring.

0: Disable

1: Enable

(R/W)

TSENS_IN_INV Configures whether to invert temperature sensor data.

0: No effect

1: Invert

(R/W)

TSENS_CLK_DIV Configures the clock division of the temperature sensor, which affects the frequency of temperature data generation by the analog circuits. (R/W)

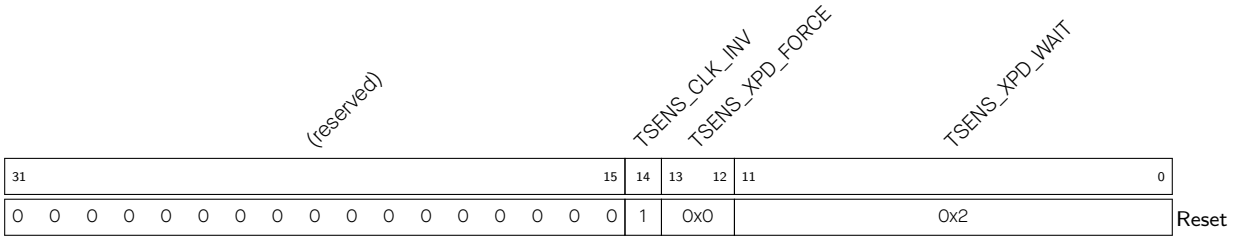
TSENS_POWER_UP Configures whether to power up temperature sensor.

0: No effect

1: Power up

(R/W)

Register 64.2. TSENS_CTRL2_REG (0x0004)



TSENS_XPD_WAIT Configures the wait time (in clock cycles) for the sensor to release from reset. (R/W)

TSENS_XPD_FORCE Configures whether to enable force power up/down the temperature sensor.

0: Disable force power up function

1: Disable force power down function

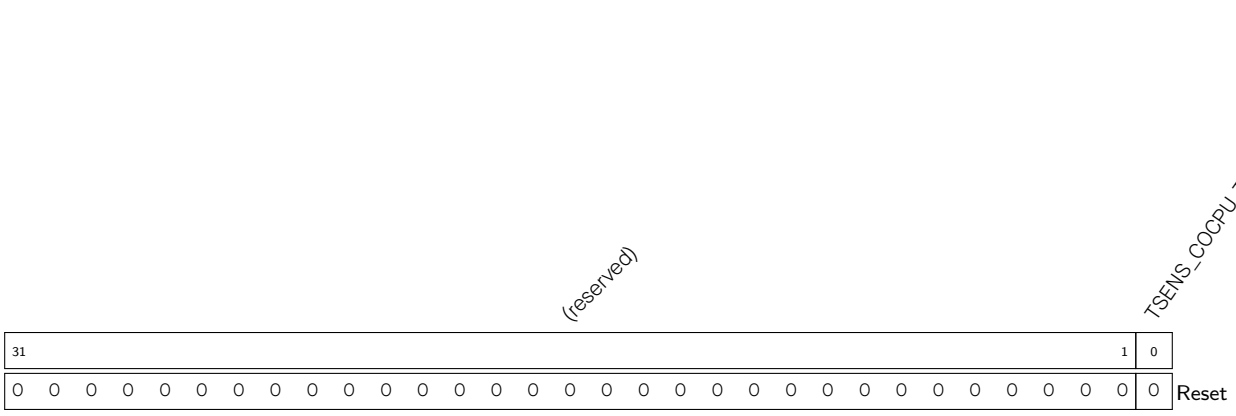
2: Enable force power up temperature sensor

3: Enable force power down temperature sensor

(R/W)

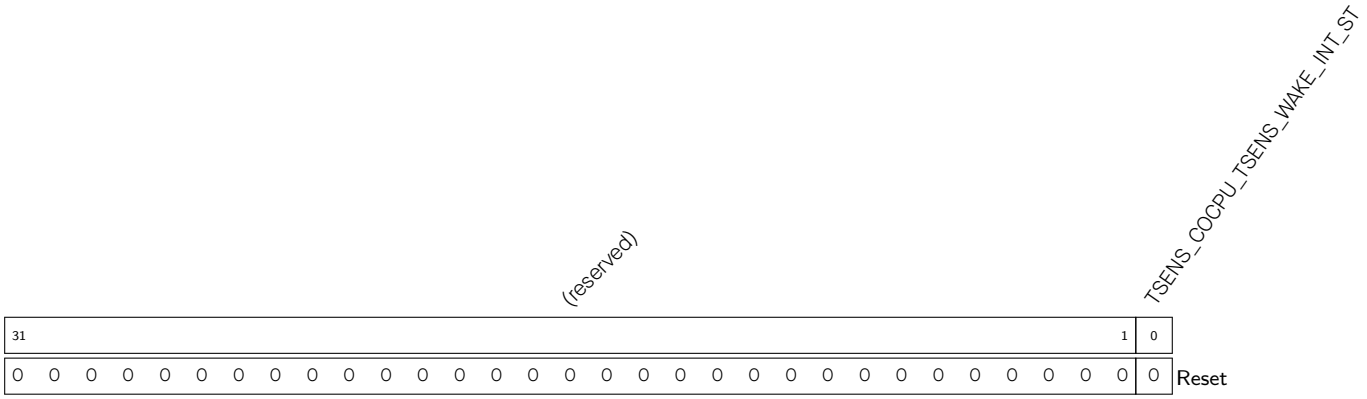
TSENS_CLK_INV Configures the phase of the temperature sensor clock. (R/W)

Register 64.3. TSENS_INT_RAW_REG (0x0008)



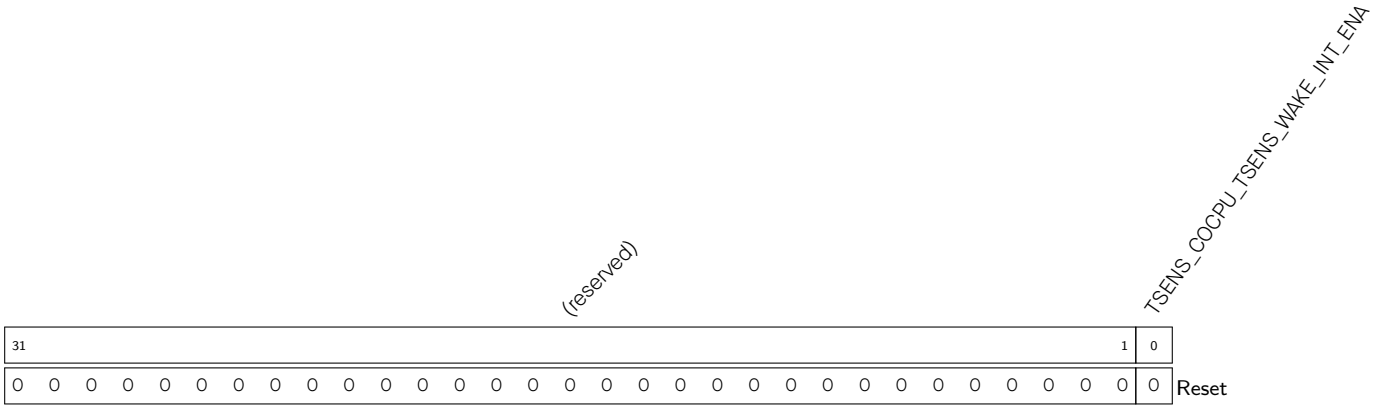
TSENS_COCPU_TSENS_WAKE_INT_RAW The raw interrupt status of the TSENS_COCPU_TSENS_WAKE_INT. (R/WTC/SS)

Register 64.4. TSENS_INT_ST_REG (0x000C)



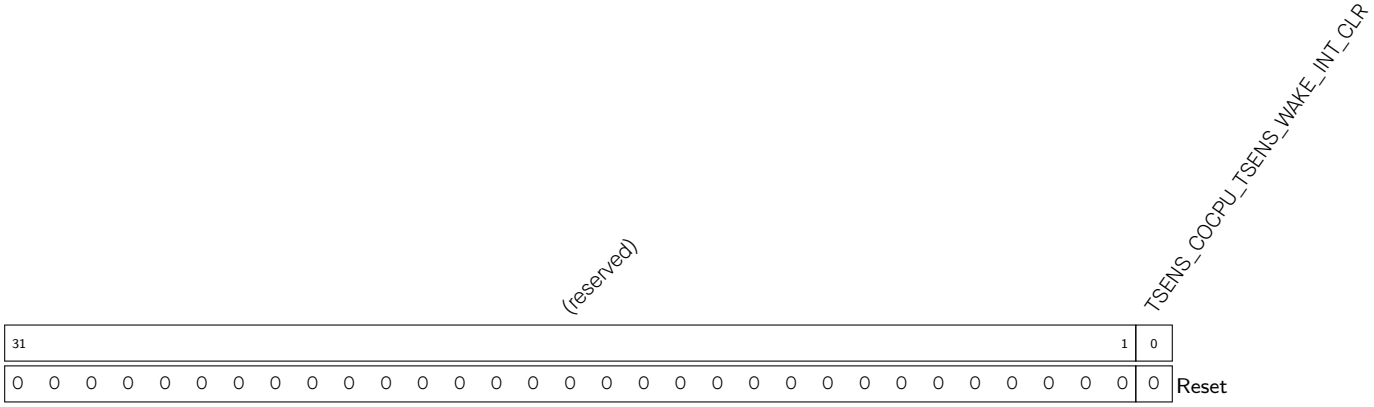
TSENS_COCPU_TSENS_WAKE_INT_ST The masked interrupt status of the TSENS_COCPU_TSENS_WAKE_INT. (RO)

Register 64.5. TSENS_INT_ENA_REG (0x0010)



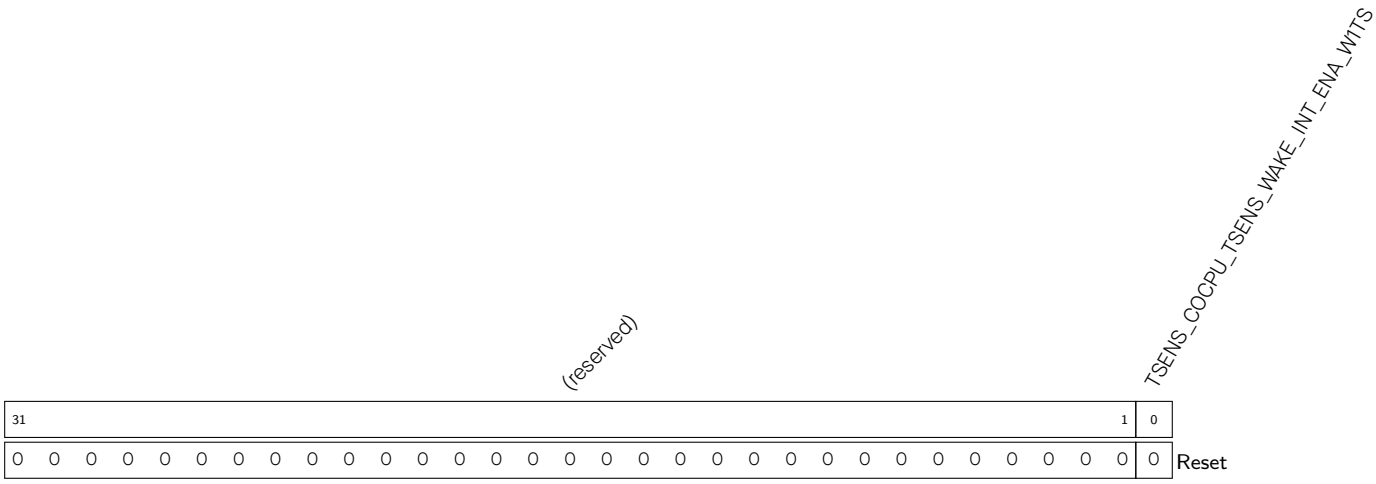
TSENS_COCPU_TSENS_WAKE_INT_ENA Write 1 to enable TSENS_COCPU_TSENS_WAKE_INT. (R/WTC)

Register 64.6. TSENS_INT_CLR_REG (0x0014)



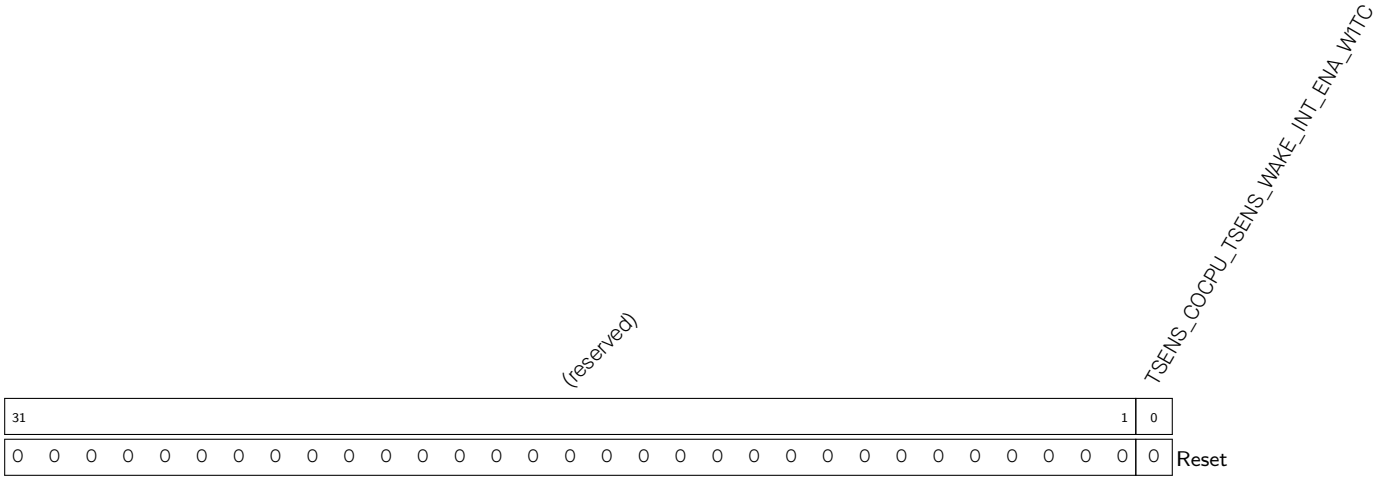
TSENS_COCPU_TSENS_WAKE_INT_CLR Write 1 to clear TSENS_COCPU_TSENS_WAKE_INT. (WT)

Register 64.7. TSENS_INT_ENA_W1TS_REG (0x001C)



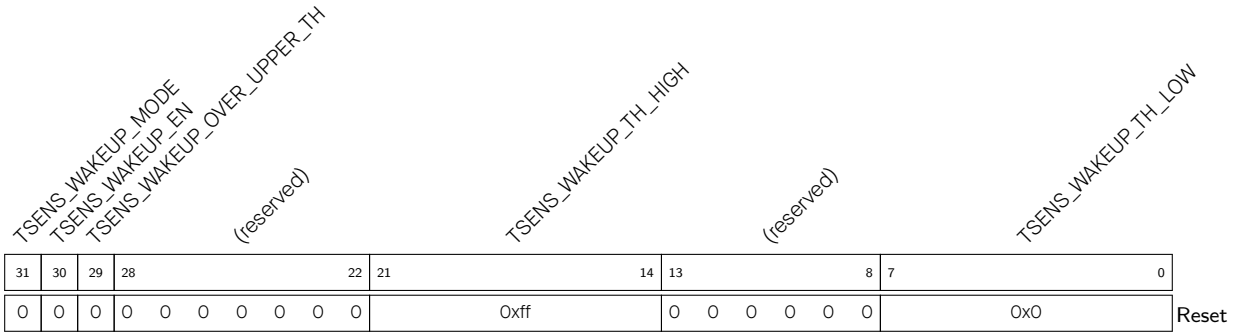
TSENS_COCPU_TSENS_WAKE_INT_ENA_W1TS Write 1 to set TSENS_INT_ENA_REG. (WT)

Register 64.8. TSENS_INT_ENA_W1TC_REG (0x0020)



TSENS_COCPU_TSENS_WAKE_INT_ENA_W1TC Write 1 to clear TSENS_INT_ENA_REG. (WT)

Register 64.9. TSENS_WAKEUP_CTRL_REG (0x0024)



TSENS_WAKEUP_TH_LOW Configures the low threshold for temperature monitoring wake-up function. (R/W)

TSENS_WAKEUP_TH_HIGH Configures the high threshold for temperature monitoring wake-up function. (R/W)

TSENS_WAKEUP_OVER_UPPER_TH Represents whether the temperature output value exceeds the threshold.

0: The temperature output value is below the low threshold.

1: The temperature output value is above the high threshold.

(RO)

TSENS_WAKEUP_EN Configures whether to enable temperature monitoring wake-up function.

0: Disable

1: Enable

(R/W)

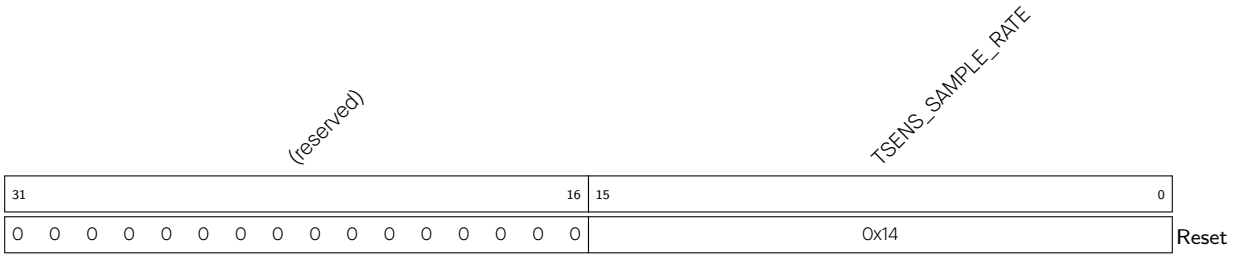
TSENS_WAKEUP_MODE Selects the wake-up mode for temperature monitoring. Valid only when TSENS_WAKEUP_EN = 1.

0: Absolute value mode

1: Change value mode

(R/W)

Register 64.10. TSENS_SAMPLE_RATE_REG (0x0028)



TSENS_SAMPLE_RATE Configures the sampling rate for hardware-triggered temperature monitoring. The sampling period = configured value × sensor's working clock cycle. (R/W)

Chapter 65

ADC Controller (ADC)

65.1 Overview

ESP32-P4 integrates two 12-bit successive approximation ADCs (SAR ADCs) for measuring analog signals from up to 14 pins. Figure 65.1-1 shows the basic architecture of the SAR ADC module on ESP32-P4.

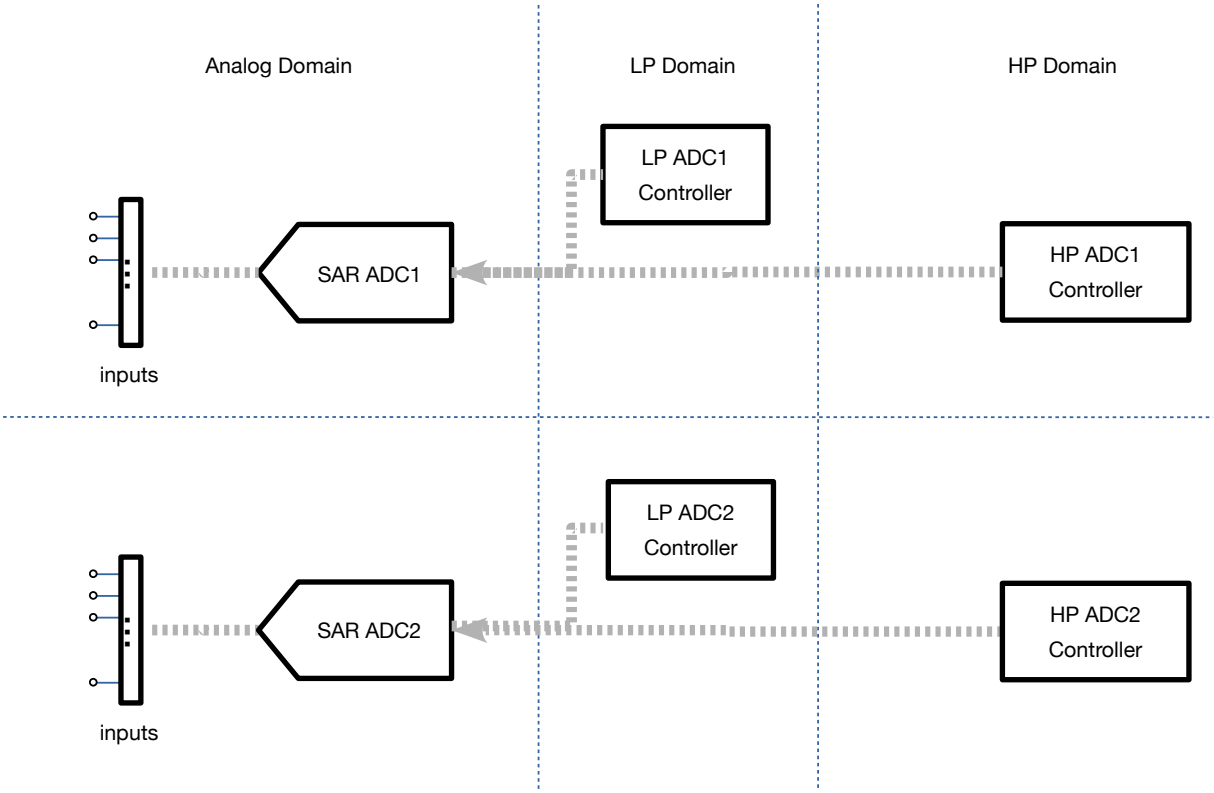


Figure 65.1-1. SAR ADC Simplified Architecture

As the figure shows, the SAR ADCs are managed by four dedicated controllers:

- Two HP (high-performance) controllers: HP ADC1 Controller and HP ADC2 Controller, designed for multi-channel sampling mode, supporting continuous transfer of conversion results to memory via the GDMA interface.
- Two LP (low-power) controllers: LP ADC1 Controller and LP ADC2 Controller, supporting operation in sleep mode and one-shot sampling mode.

65.2 Terminology

The following terms related to SAR ADC are defined in the context of the *ESP32-P4 Technical Reference Manual* to help readers better understand this document:

SAR ADC: Including analog ADC circuit and digital ADC controller. The HP ADC and LP ADC in the following text share the analog ADC circuit, but have their own controllers, namely the HP ADC controllers and LP ADC controllers. The SAR ADC mentioned in the text refers to both HP ADC and LP ADC.

HP ADC: High-performance ADC, supporting multi-channel sampling, GDMA data transfer, etc., not available in sleep mode.

LP ADC: Low-power ADC, supporting one-shot sampling mode, can be used in sleep mode.

One-shot sampling mode: In this mode, LP ADC samples one channel at a time.

Multi-channel sampling mode: In this mode, HP ADC sequentially samples a group of channels or continuously samples a single channel.

Dual HP ADC sampling: It means two HP ADCs sample either simultaneously or alternatively.

Conversion result: The conversion result is the binary digital value obtained after the analog-to-digital conversion process. It is sometimes written as conversion data.

Filtered data: Filtered data is the result of processing conversion data through a filter.

Sampling: The term typically refers to the entire process of sampling analog inputs, converting the sampled data, and transferring the conversion results to memory. In certain contexts and stages of the process, sampling may also refer to the SAR ADC capturing data points from an analog signal.

65.3 Features

The SAR ADCs have the following features:

- Support HP ADC*x* controllers and LP ADC*x* controllers to get control of SAR ADC*x* via software, *x* = 1, 2
- 12-bit resolution
- Analog inputs sampling from up to 14 pins
- HP ADC*x* controllers:
  - Provide multi-channel sampling control module, supporting multi-channel sampling mode, with configurable channel sampling sequence
  - Provide mode control module, supporting dual HP ADC sampling
  - Provide two filters with configurable filter coefficients

  - Provide two threshold monitors that can trigger an interrupt when the filtered data is above a high threshold or below a low threshold
  - Support continuous transfer of conversion results to memory via the GDMA interface
- LP ADC*x* controllers:
  - Support one-shot sampling mode
  - Support sampling in sleep mode (e.g., Deep-sleep)
- Support for several Event Task Matrix (ETM) related events and tasks

65.4 Architecture

The major components of SAR ADCs and their interconnections are shown in Figure 65.4-1.

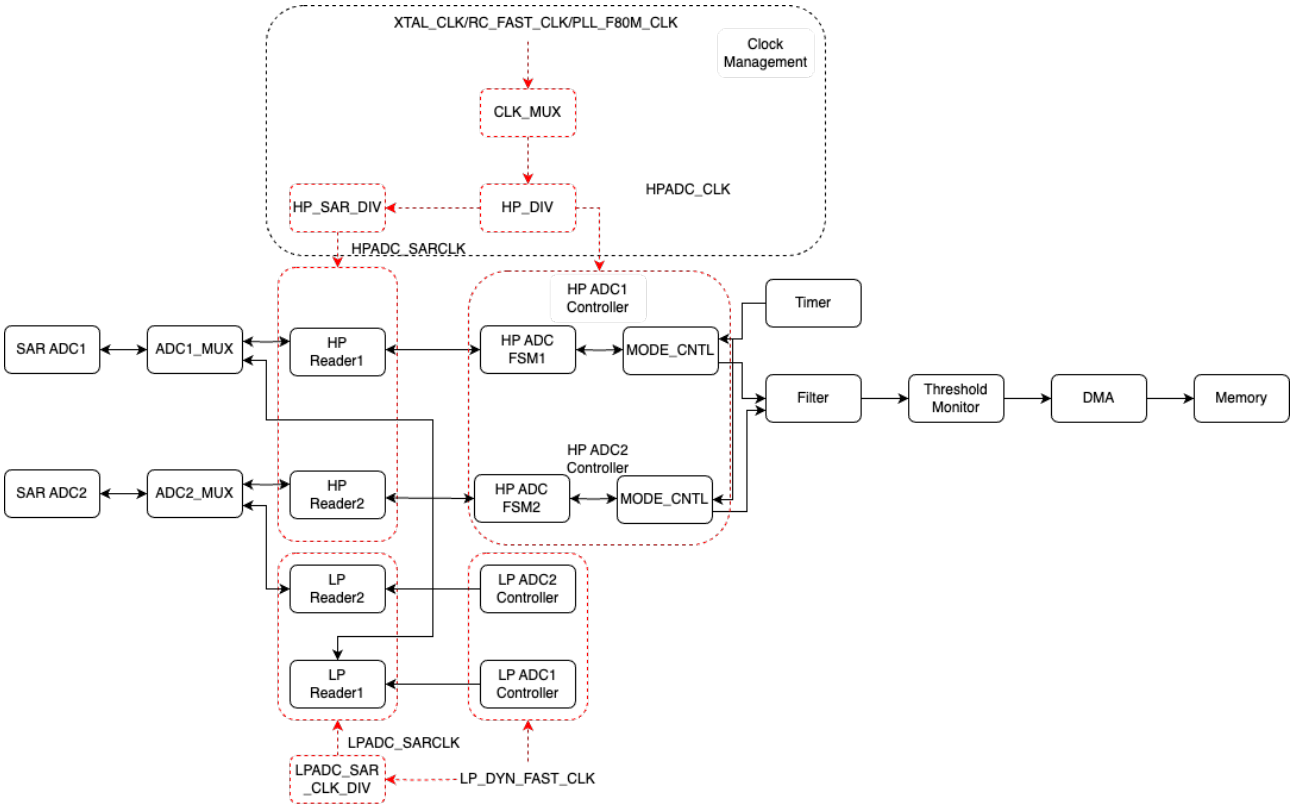


Figure 65.4-1. SAR ADC Architecture

- —>: clock signals
- : clock divider, clock mux, and the blocks the clock works for

As Figure 65.4-1 shows, the SAR ADC module contains the following major functional blocks:

- SAR ADC1: Samples analog inputs from up to 8 pins
- SAR ADC2: Samples analog inputs from up to 6 pins
- Clock Management: Selects clock source and division
- Timer: Dedicated timer for the HP ADC*x* controllers to generate sampling enable signal.
- HP ADC FSM*x*: FSM1 and FSM2 that can:

  - receive the sampling enable signal from the timer.
  - determine the conversion rules defined in the patterns for HP ADC*x*.
  - drive the HP Reader*x* module to read conversion data.
  - transfer conversion data to the filter.
- Filter: Filter0 and filter1, used for filtering conversion results in multi-channel sampling mode.
- Threshold Monitor: Threshold monitor 0 and threshold monitor 1 that trigger an interrupt when the filtered data is above a high threshold or below a low threshold.
- MODE_CNTL: Used for filtering the sampling signal triggered by the timer, supporting dual HP ADC sampling mode.
- HP Reader*x*: HP reader 1 and HP reader 2, driven by HP ADC FSM*x* to read data from HP ADC*x*.
- LP Reader*x*: LP reader 1 and LP reader 2, driven by LP ADC*x* Controller to read data from LP ADC*x*.
- LP ADC*x* Controller: LP ADC1 Controller and LP ADC2 Controller. The controllers provide sampling enable signal, drive the LP Reader*x* module to read the conversion results of LP ADC*x*, and store the conversion results.

65.5 Functional Description

65.5.1 SAR ADC Power Up

The SAR ADCs can be powered up by setting PMU_XPD_PERIF_I2C and PMU_PERIF_I2C_RST. The ADC sampling can be conducted immediately after power-up. Users need not be concerned about wait time, as it has been accounted for in the hardware design.

65.5.2 SAR ADC Channels

The SAR ADCs have 14 channels that are connected to 14 pins on the chip. To sample an analog signal, the SAR ADCs must first select the analog pin to measure via an internal multiplexer.

Table 65.5-1 shows the pins used as SAR ADC channels.

Table 65.5-1. SAR ADC Channels and Chip Pins

Chip Pins	SAR ADC Channel	SAR ADC Selection
GPIO16	0	SAR ADC1
GPIO17	1	
GPIO18	2	
GPIO19	3	
GPIO20	4	
GPIO21	5	
GPIO22	6	
GPIO23	7	
GPIO49	0	SAR ADC2
GPIO50	1	

Chip Pins	SAR ADC Channel	SAR ADC Selection
GPIO51	2	
GPIO52	3	
GPIO53	4	
GPIO54	5	

65.5.3 SAR ADC Clock

The clock for the HP ADC*x* Controller has three possible sources, selected by HP_SYS_CLKRST_ADC_CLK_SRC_SEL:

- XTAL_CLK
- RC_FAST_CLK
- PLL_F80M_CLK

The following clocks can be divided from the HP ADC*x* Controller's clock:

- HPADC_SARCLK: It is the operating clock of HP ADC*x* and HP Reader*x*.
- HPADC_CLK: It is the operating clock of HP ADC FSM*x*.

The frequency of HPADC_SARCLK can affect the sampling accuracy. If the HPADC_SARCLK frequency is higher than 5 MHz, the sampling accuracy will decrease. HPADC_SARCLK is divided from HPADC_CLK via a dedicated divider. The division is set in ADC_SAR_CLK_DIV.

It takes 25 HPADC_SARCLK clock cycles to complete one sampling, so the maximum sampling rate is limited by the frequency of HPADC_SARCLK.

The LP ADC*x* Controller is clocked from LP_DYN_FAST_CLK.

The following clocks can be divided from the LP ADC*x* Controller's clock:

- LPADC_SARCLK: It is the operating clock of LP ADC*x* and LP Reader*x*. The division set in LPADC_SAR1_CLK_DIV and LPADC_SAR2_CLK_DIV should be at least 2. The frequency of LPADC_SARCLK should not exceed 5 MHz.
- LPADC_CLK: It is the operating clock of LP ADC FSM*x*.

For more information about clocks, please refer to Chapter 11 *Reset and Clock*.

65.5.4 SAR ADC Conversion and Attenuation

The SAR ADCs can measure analog voltages from 0 mV to V_{ref}. V_{ref} is the SAR ADCs' internal reference voltage. The conversion result ($data$) is a 12-bit digital value, which is the raw data. To calculate the voltage V_{data} based on the raw data, the following formula can be used:

$$V_{data} = \frac{V_{ref}}{4095} \times data$$

k is the coefficient corresponding to the configured attenuation.

To convert voltages larger than V_{ref}, apply attenuation to the input signals. The supported attenuation levels are:

- 0 dB (k≈100%)
- 2.5 dB (k≈75%)
- 6 dB (k≈50%)
- 12 dB (k≈25%)

65.5.5 HP ADC FSM

In multi-channel sampling mode HP ADC FSM (hereinafter referred to as FSM) generates all types of signals used in the sampling process. Figure 65.5-1 illustrates how the FSM works.

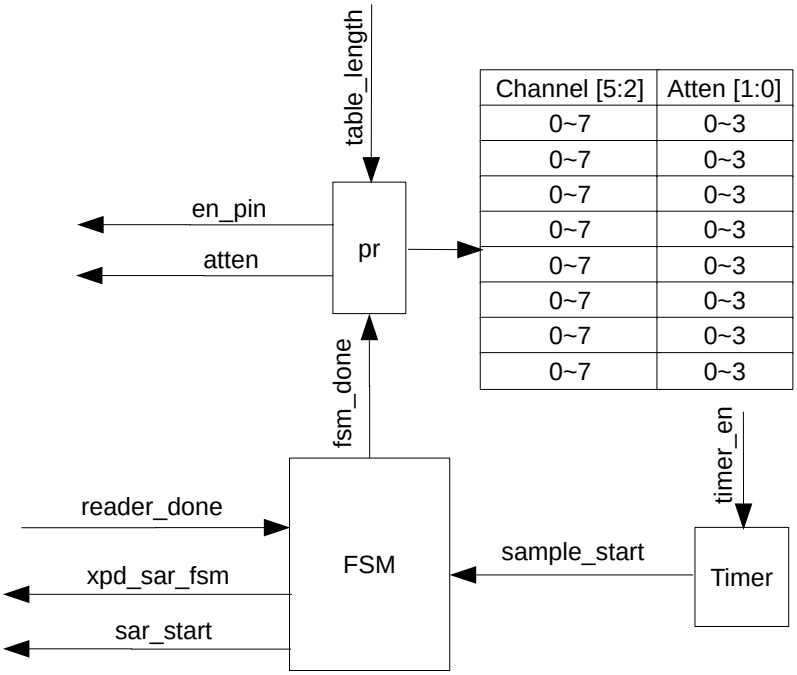


Figure 65.5-1. HP ADC FSM Block Diagram

Wherein:

- Timer: a dedicated timer for the HP ADC controller to generate `sample_start` signal.
- pr: a pointer to the pattern table that defines the conversion rules for HP ADC. FSM sends out corresponding signals based on the conversion rules.

Set ADC_TIMER_EN to enable the timer. The timeout event of this timer triggers a `sample_start` signal. This signal drives the FSM module to start sampling. When the FSM module receives the `sample_start` signal, it starts the following operations:

- Powers up HP ADC.
- Determines the conversion rules (selected sample channels and attenuations) defined in the patterns that the current pr points to.
- Outputs the `en_pin` and `atten` signals corresponding to the conversion rules to the analog side.
- Initiates the `sar_start` signal and starts sampling.

When FSM receives the `reader_done` signal from HP Reader, it starts the following operations:

- Stops sampling.
- Transfers the conversion result to the filter. Then the threshold monitor transfers the filtered result to memory via GDMA (see Figure 65.4-1).
- Updates pr and waits for the next sampling. The pointer pr counts cyclically between 0 and ADC_SAR1/2_PATT_LEN (table_length).

65.5.6 HP ADC Pattern Table

HP ADC FSM1/2 each contain a pattern table configured by ADC_SAR*x*_PATT_TAB*m*_REG, with *m* indicating registers 1 ~ 4. Each register contains four patterns and each pattern is 6 bits wide, as below shows.

(reserved)								cmd0						cmd1						cmd2						cmd3					
31							24	23					18	17					12	11					6	5					0
0	0	0	0	0	0	0	0	0x0000						0x0000						0x0000						0x0000					

cmd *n* (*n* = 0 - 3) represents patterns 0 – 3.

Figure 65.5-2. ADC_SAR*x*_PATT_TAB1_REG Contains Patterns 0 - 3

(reserved)								cmd4						cmd5						cmd6						cmd7					
31							24	23					18	17					12	11					6	5					0
0	0	0	0	0	0	0	0	0x0000						0x0000						0x0000						0x0000					

cmd *n* (*n* = 4 - 7) represents patterns 4 ~ 7.

Figure 65.5-3. ADC_SAR*x*_PATT_TAB2_REG Contains Patterns 4 - 7

(reserved)								cmd8						cmd9						cmd10						cmd11					
31							24	23					18	17					12	11					6	5					0
0	0	0	0	0	0	0	0	0x0000						0x0000						0x0000						0x0000					

cmd *n* (*n* = 8 - 11) represents patterns 8 ~ 11.

Figure 65.5-4. ADC_SAR*x*_PATT_TAB3_REG Contains Patterns 8 - 11

(reserved)								cmd12						cmd13						cmd14						cmd15					
31							24	23					18	17					12	11					6	5					0
0	0	0	0	0	0	0	0	0x0000						0x0000						0x0000						0x0000					

cmd *n* (*n* = 12 - 15) represents patterns 12 ~ 15.

Figure 65.5-5. ADC_SAR*x*_PATT_TAB4_REG Contains Patterns 12 - 15

Each pattern is 6 bits wide, consisting of two fields where conversion rules are stored. Figure 65.5-6 shows the pattern format and is followed by the descriptions of each field.

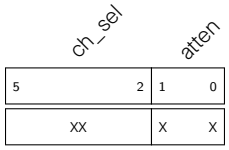


Figure 65.5-6. Pattern Structure

atten Configures attenuation:
0: 0 dB
1: 2.5 dB
2: 6 dB
3: 12 dB

ch_sel Configures channel. For HP ADC1, values 0-7 correspond to channels 0-7 individually, for HP ADC2, values 0-5 correspond to channels 0-5 individually.

65.5.7 HP ADC Pattern Configuration Example for Multi-channel Sampling

Assuming you want to implement the following sequence of HP ADC1 multi-channel sampling:

- Sample channel 2 of HP ADC1 with attenuation of 12 dB;
- Sample channel 0 of HP ADC1 with attenuation of 2.5 dB.

Then, the detailed configuration is as follows:

- Configure the first pattern (cmd0) of ADC_SAR1_PATT_TAB1_REG[5:0]:

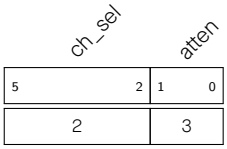


Figure 65.5-7. cmd0 configuration

atten write 3 to this field, to set the attenuation to 12 dB.

ch_sel write 2 to this field, to select channel 2.

- Configure the second pattern (cmd1) of ADC_SAR1_PATT_TAB1_REG[11:6]:

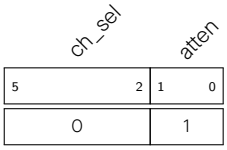


Figure 65.5-8. cmd1 Configuration

atten write 1 to this field, to set the attenuation to 2.5 dB.

ch_sel write 0 to this field, to select channel 0.

- Configure ADC_SAR1_PATT_LEN to 1, i.e., set pattern table size to (this value + 1) = 2. Then patterns cmd0 and cmd1 will be used.

- Enable the timer so that the HP ADC1 Controller starts sampling the two channels periodically.

65.5.8 Dual HP ADC Sampling Control

ADC_WORK_MODE controls how two HP ADCs work together:

- 0: Single HP ADC sampling, i.e., only one HP ADC samples. When a sampling request is triggered by the timer, ADC_SAR_SEL controls which FSM receives the sampling request:
  - 0: HP ADC FSM1, i.e., using HP ADC1 for sampling.
  - 1: HP ADC FSM2, i.e., using HP ADC2 for sampling.
- 1: Dual HP ADC simultaneous sampling. When a sampling request is triggered by the timer, both HP ADC FSM1 and HP ADC FSM2 receive the sampling request.
- 2: Dual HP ADC alternative sampling. When an odd number of sampling requests are triggered by the timer, HP ADC FSM1 receives the sampling request. When an even number of sampling requests are triggered by the timer, HP ADC FSM2 receives the sampling request.

65.5.9 HP ADC Filters

The HP ADC*x* controllers provide two filters for filtering conversion results in multi-channel sampling mode. Both filters can be configured to any HP ADC channel, but cannot be configured to the same channel. If the two filters are configured to the same channel, the first one takes effect.

The filtered data is determined by the following equation:

$$data_{cur} = \frac{(k-1)data_{prev}}{k} + \frac{data_{in}}{k} + 0.5$$

- $data_{cur}$: the filtered data
- $data_{in}$: the conversion result
- $data_{prev}$: the last filtered data
- k: the filter coefficient

The filters are configured as follows:

- Configure ADC_FILTER_CHANNEL*x* to select the HP ADC channel for filter *x*, *x*=0, 1.
- Configure ADC_FILTER_FACTOR*x* to set the coefficient k for filter *x*, *x*=0, 1.

65.5.10 HP ADC Threshold Monitors

The HP ADC*x* controllers provide two threshold monitors to monitor the filtered data in multi-channel sampling mode. When the data is above the high threshold, a high threshold interrupt is triggered; when the data is below the low threshold, a low threshold interrupt is triggered. Both monitors can be configured to any HP ADC channel, but cannot be configured to the same channel.

Threshold monitors are configured as follows:

- Set ADC_THRES*x*_EN to enable threshold monitor *x* (*x*=0, 1).
- Configure ADC_THRES*x*_LOW to set a low threshold.

- Configure ADC_THRES*x*_HIGH to set a high threshold.
- Configure ADC_THRES*x*_CHANNEL to select the channel to monitor.

65.5.11 HP ADC GDMA Support

When HP ADC converts data on multiple channels, it is necessary to enable GDMA so that the data can be transferred to memory continuously.

GDMA support is triggered by the timer in the HP ADC*x* controllers. Users can switch the GDMA data path to the HP ADC controller by configuring ADC_APB_ADC_TRANS. For specific GDMA configuration, please refer to Chapter 6 *GDMA Controller (GDMA-AHB, GDMA-AXI)*.

The HP ADC eventually passes 32-bit data to GDMA. The data format is shown in Figure 65.5-9.

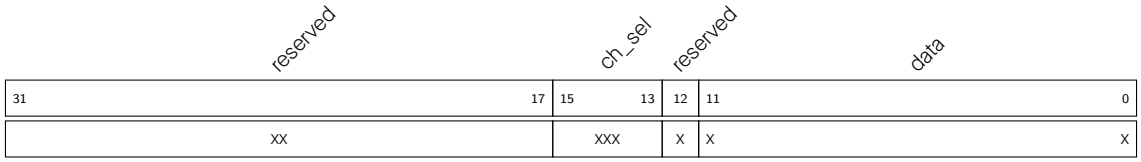


Figure 65.5-9. GDMA Data Format

data 12-bit HP ADC conversion result

ch_sel 3-bit channel information

65.5.12 LP ADC Wake-Up Modes for Automatic Monitoring

The LP ADC*x* controllers support automatic monitoring of the conversion data. A dedicated timer generates LP ADC sampling signals periodically based on the value of LPADC_ADC*x*_HW_READ_RATE_I. Then the conversion data is fed into a threshold monitor (not the HP ADC threshold monitors). If the data exceeds the threshold, an interrupt will be generated.

There are two wake-up modes for automatic monitoring, selected by LPADC_SAR*x*_WAKEUP_MODE:

- 0: Absolute value mode
  Monitors the absolute value of the conversion data. Configure LPADC_SAR*x*_WAKEUP_TH_LOW and LPADC_SAR*x*_WAKEUP_TH_HIGH to set the thresholds. Wake-up will be triggered if the data is above the high threshold or below the low threshold.
- 1: Change value mode
  Monitors the changes of the conversion data. If the increment of two consecutive samplings exceeds the high threshold configured in LPADC_SAR*x*_WAKEUP_TH_HIGH, or the decrement of two consecutive samplings exceeds the low threshold configured in LPADC_SAR*x*_WAKEUP_TH_LOW, a wake-up will be triggered. For example, when LPADC_SAR*x*_WAKEUP_TH_LOW is set to 8, and the two consecutive conversion results are 28 and 19, resulting in a decrement of 9, a wake-up will be triggered.

65.6 Event Task Matrix Feature

The SAR ADC on ESP32-P4 supports the Event Task Matrix (ETM) function, which allows SAR ADC's ETM tasks to be triggered by any peripherals' ETM events, or SAR ADC's ETM events to trigger any peripherals' ETM

tasks. This section introduces the ETM tasks and events related to SAR ADC. For more information, please refer to Chapter 14 Event Task Matrix (ETM).

The SAR ADC can receive the following ETM tasks:

- ADC_TASK_SAMPLE0: LP ADC starts one-shot sampling when this task is triggered.
- ADC_TASK_START0: HP ADC starts multi-channel sampling when this task is triggered.
- ADC_TASK_STOP0: SAR ADC stops sampling when this task is triggered.

The SAR ADC can generate the following ETM events:

- ADC_EVT_CONV_CMPLT0: Generated each time SAR ADC completes a sampling in either one-shot sampling mode or multi-channel sampling mode.
- ADC_EVT_EQ_ABOVE_THRESH*x*: Generated when the HP ADC's filtered data is above the threshold. *x* = 0, 1, representing threshold monitor 0, 1.
- ADC_EVT_EQ_BELOW_THRESH*x*: Generated when the HP ADC's filtered data is below the threshold. *x* = 0, 1, representing threshold monitor 0, 1.
- ADC_EVT_STARTED0: Generated when SAR ADC begins sampling; one-shot sampling will not trigger this event.
- ADC_EVT_STOPPED0: Generated when SAR ADC stops sampling, one-shot sampling will not trigger this event.

In practical applications, SAR ADC's ETM events can trigger its own ETM tasks. For example, the ADC_EVT_EQ_ABOVE_THRESH*x* event can trigger the ADC_TASK_STOP0 task.

65.7 Interrupts

ESP32-P4's SAR ADC can generate the following interrupt signals that will be sent to the Interrupt Matrix.

- ADC_INTR
- LP_ADC_INTR

The following internal interrupt sources from SAR ADC can generate the interrupt signal ADC_INTR:

- ADC_SAR*x*_DONE_INT: Triggered when HP ADC*x* completes one sampling.
- ADC_THRES*x*_HIGH_INT: Triggered when the filtered data is above the high threshold.
- ADC_THRES*x*_LOW_INT: Triggered when the filtered data is below the low threshold.

The following internal interrupt sources from SAR ADC can generate the interrupt signal LP_ADC_INTR:

- LPADC_COCPU_SARADC*x*_DONE_INT: Triggered when LP ADC*x* completes one sampling.
- LPADC_COCPU_SARADC*x*_ERROR_INT: Triggered when sampling error occurs.
- LPADC_COCPU_SARADC*x*_WAKE_INT: Triggered when LP ADC*x* generates a wake-up event (such as conversion data exceeding the threshold).

Each interrupt source can be configured by a common set of registers that are described in Section *Interrupt Configuration Registers*. The specific registers can be found in Section 65.9 *Register Summary*.

65.8 Programming Procedure

65.8.1 HP ADC Multi-Channel Sampling Mode

The multi-channel sampling mode can be configured with the following procedure:

1. Set PMU_XPD_PERIF_I2C and PMU_PERIF_I2C_RST to power up HP ADC.
2. Configure HP_SYS_CLKRST_ADC_CLK_SRC_SEL to select HP ADC clock source.
3. Configure HP_SYS_CLKRST_ADC_CLK_DIV_NUM and HP_SYS_CLKRST_ADC_SAR*x*_CLK_DIV_NUM to set clock division.
4. Set HP_SYS_CLKRST_ADC_CLK_EN to enable HP ADC clock.
5. Set LPADC_SAR*x*_DIG_FORCE to enable the sampling channel.
6. Configure the pattern table as described in Section 65.5.6 *HP ADC Pattern Table*.
7. Configure channels and filtering coefficients as described in Section 65.5.9 *HP ADC Filters*.
8. Configure threshold monitoring as described in Section 65.5.10 *HP ADC Threshold Monitors*, as needed.
9. Set ADC_APB_ADC_TRANS to use GDMA.
10. Configure ADC_TIMER_TARGET to set trigger target for HP ADC timer.
11. Set ADC_TIMER_EN to enable the timer.

The timer timeout will trigger HP ADC FSM to start sampling according to the pattern table. The conversion result will be automatically stored in memory. When the sampling reaches the specified limit in ADC_APB_ADC_EOF_NUM, it will terminate.

Once sampling is complete, an ADC_SAR*x*_DONE_INT_RAW interrupt will be generated.

65.8.2 LP ADC One-shot Sampling Mode

The one-shot sampling mode can be configured with the following procedure:

1. Set PMU_XPD_PERIF_I2C and PMU_PERIF_I2C_RSTB to power up LP ADC.
2. Configure LPPERI_LPADC_FUNC_DIV_NUM and LPPERI_LPADC_SAR*x*_DIV_NUM to set the frequency of LPADC_CLK and LPADC*x*_SARCLK.
3. Configure LPADC_SAR*x*_EN_PAD and LPADC_SAR*x*_ATTEN to select sampling channels and attenuation.
4. Set LPADC_SAR_MEAS*x*_START_SAR to enable the one-shot sampling mode of LP ADC*x*.

Once sampling is complete, an LPADC_COCPU_SARADC*x*_INT_RAW interrupt is generated. Software can read the conversion result from LPADC_SAR_MEAS*n*_DATA_SAR.

65.8.3 LP ADC Automatic Monitoring

The configuration process for the automatic monitoring of the LP ADC controller is as follows:

1. Configure LPADC_ADC*x*_HW_READ_RATE_I to set the sampling rate.
2. Configure LPADC_SAR*x*_WAKEUP_MODE to select the wake-up mode.

3. Configure LPADC_SAR*x*_WAKEUP_TH_HIGH and LPADC_SAR*x*_WAKEUP_TH_LOW to set the high and low thresholds.
4. Set LPADC_SAR*x*_WAKEUP_EN to enable automatic monitoring.
5. Set LPADC_ADC*x*_HW_READ_EN_I to enable periodic monitoring.

During the automatic monitoring process, the conversion data will not be stored, but software can always read the last conversion data from LPADC_SAR_MEAS*n*_DATA_SAR.

65.9 Register Summary

65.9.1 HP ADC Register Summary

The addresses in the table below are relative to the base address of HP ADC (written as ADC Controller) in Table 9.3-2 in Chapter 9 *System and Memory*.

The abbreviations given in Column **Access** are explained in Section *Access Types for Registers*.

Name	Description	Address	Access
HP ADC Configuration Registers			
ADC_CTRL_REG	Configuration register for HP ADC Controller	0x0000	R/W
ADC_CTRL2_REG	Configuration register for HP ADC Controller.	0x0004	R/W
ADC_FILTER_CTRL1_REG	Configuration register 1 for HP ADC filter.	0x0008	R/W
ADC_SAR1_PATT_TAB1_REG	Pattern table register 1 for HP ADC1.	0x0018	R/W
ADC_SAR1_PATT_TAB2_REG	Pattern table register 2 for HP ADC1.	0x001C	R/W
ADC_SAR1_PATT_TAB3_REG	Pattern table register 3 for HP ADC1.	0x0020	R/W
ADC_SAR1_PATT_TAB4_REG	Pattern table register 4 for HP ADC1.	0x0024	R/W
ADC_SAR2_PATT_TAB1_REG	Pattern table register 1 for HP ADC2.	0x0028	R/W
ADC_SAR2_PATT_TAB2_REG	Pattern table register 2 for HP ADC2.	0x002C	R/W
ADC_SAR2_PATT_TAB3_REG	Pattern table register 3 for HP ADC2.	0x0030	R/W
ADC_SAR2_PATT_TAB4_REG	Pattern table register 4 for HP ADC2.	0x0034	R/W
ADC_FILTER_CTRL0_REG	Configuration register 0 for HP ADC filter.	0x003C	R/W
ADC_THRES0_CTRL_REG	Filtered data threshold control register 0.	0x0044	R/W
ADC_THRES1_CTRL_REG	Filtered data threshold control register 1.	0x0048	R/W
ADC_THRES_CTRL_REG	Filter configuration register.	0x004C	R/W
ADC_INT_ENA_REG	Enable register of HP ADC interrupts.	0x0050	R/W
ADC_INT_RAW_REG	Raw register of HP ADC interrupts.	0x0054	R/WTC/SS
ADC_INT_ST_REG	State register of HP ADC interrupts.	0x0058	RO
ADC_INT_CLR_REG	Clear register of HP ADC interrupts.	0x005C	WT
ADC_DMA_CONF_REG	GDMA configuration register for HP ADC.	0x0060	R/W
HP ADC Version Control Registers			
ADC_CTRL_DATE_REG	HP ADC version control register	0x03FC	RO

65.9.2 LP ADC Register Summary

The addresses in the table below are relative to LP ADC's base address provided in Table 9.3-2 in Chapter 9 *System and Memory*.

The abbreviations given in Column **Access** are explained in Section *Access Types for Registers*.

Name	Description	Address	Access
LP ADC Configuration Registers			
LPADC_READER1_CTRL_REG	LP ADC1 Reader control register.	0x0000	R/W

Name	Description	Address	Access
LPADC_MEAS1_CTRL2_REG	LP ADC1 configuration registers.	0x000C	varies
LPADC_MEAS1_MUX_REG	LP or HP ADC controller selection register for SAR ADC1.	0x0010	R/W
LPADC_ATTEN1_REG	LP ADC1 attenuation registers.	0x0014	R/W
LPADC_READER2_CTRL_REG	LP ADC2 Reader control register.	0x0024	R/W
LPADC_MEAS2_CTRL2_REG	LP ADC2 configuration registers.	0x0030	varies
LPADC_MEAS2_MUX_REG	LP or HP controller selection register for SAR ADC2.	0x0034	R/W
LPADC_ATTEN2_REG	LP ADC2 attenuation registers.	0x0038	R/W
LP ADC Power Control Registers			
LPADC_FORCE_WPD_SAR_REG	LP ADC Controllers power control registers.	0x003C	R/W
LP ADC Interrupt Registers			
LPADC_COCPU_INT_RAW_REG	Raw register of LP ADC interrupts.	0x0048	R/WTC/SS
LPADC_INT_ENA_REG	Enable register of LP ADC interrupts.	0x004C	R/WTC
LPADC_INT_ST_REG	Status register of LP ADC interrupts.	0x0050	RO
LPADC_INT_CLR_REG	Clear register of LP ADC interrupts.	0x0054	WT
LPADC_INT_ENA_W1TS_REG	LPADC_INT_ENA_REG configuration register.	0x0058	WT
LPADC_INT_ENA_W1TC_REG	LPADC_INT_ENA_REG configuration register.	0x005C	WT
LP ADC Wake-up Control Registers			
LPADC_WAKEUP1_REG	LP ADC1 wake-up configuration registers.	0x0060	varies
LPADC_WAKEUP2_REG	LP ADC2 wake-up configuration registers.	0x0064	varies
LPADC_WAKEUP_SEL_REG	Wake-up source selection register.	0x0068	R/W
LPADC_SAR1_HW_WAKEUP_REG	LP ADC1 automatic monitoring configuration registers.	0x006C	R/W
LPADC_SAR2_HW_WAKEUP_REG	LP ADC2 automatic monitoring configuration registers.	0x0070	R/W

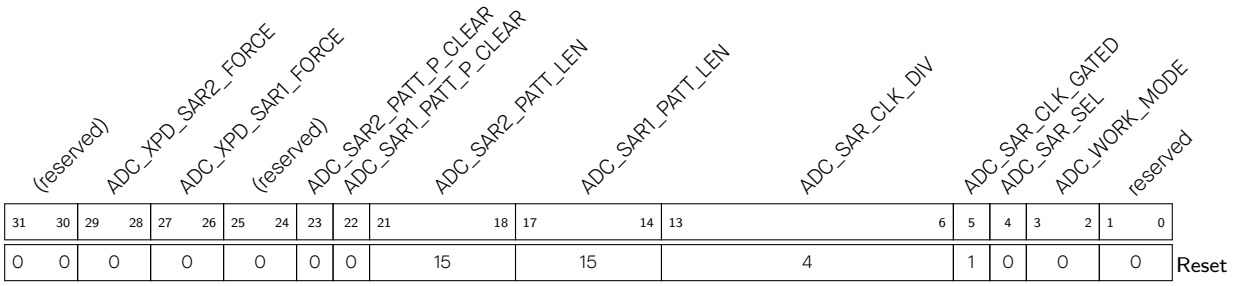
65.10 Registers

65.10.1 HP ADC Registers

The addresses in the following section are relative to the base address of HP ADC (written as ADC Controller) provided in Table 9.3-2 in Chapter 9 *System and Memory*.

For how to program reserved fields, please refer to Section Programming Reserved Register Field.

Register 65.1. ADC_CTRL_REG (0x0000)



ADC_WORK_MODE Configures dual HP ADC sampling.

0: Single HP ADC sampling

1: Dual HP ADC simultaneous sampling

2: Dual HP ADC alternative sampling.

(R/W)

ADC_SAR_SEL Configures which HP ADC to use for single HP ADC sampling.

0: Use HP ADC1

1: Use HP ADC2

(R/W)

ADC_SAR_CLK_GATED Enable HP ADC clock gate when HP ADC is in idle.

0: Disable

1: Enable

(R/W)

ADC_SAR_CLK_DIV Configures HP ADC clock division. Division = configured value + 1 . (R/W)

ADC_SAR1_PATT_LEN Configures how many pattern table entries will be used for HP ADC1.

0: Use only cmd0

1: Use cmd0 and cmd1

n: Use cmd0 to cmdn, the maximum n is 7

(R/W)

ADC_SAR2_PATT_LEN Configures how many pattern table entries will be used for HP ADC2.

0: Use only cmd0

1: Use cmd0 and cmd1

n: Use cmd0 to cmdn, the maximum n is 7

(R/W)

ADC_SAR1_PATT_P_CLEAR Clears the pointer of pattern table for HP ADC1 Controller.

0: No effect

1: Clear

(R/W)

ADC_SAR2_PATT_P_CLEAR Clears the pointer of pattern table for HP ADC2 Controller.

0: No effect

1: Clear

(R/W)

Continued on the next page...

Register 65.1. ADC_CTRL_REG (0x0000)

Continued from the previous page...

ADC_XPD_SAR1_FORCE Keep HP ADC1 powered on forcefully.

0: No effect

1: Power on forcefully

(R/W)

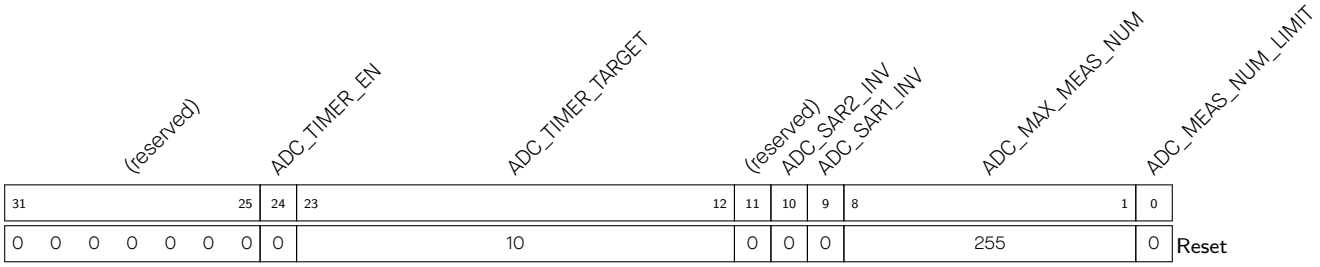
ADC_XPD_SAR2_FORCE Keep HP ADC2 powered on forcefully.

0: No effect

1: Power on forcefully

(R/W)

Register 65.2. ADC_CTRL2_REG (0x0004)



ADC_MEAS_NUM_LIMIT Configures whether to enable the limitation of HP ADC's maximum conversion times.

0: Disable

1: Enable

(R/W)

ADC_MAX_MEAS_NUM Configures the HP ADC's maximum conversion times. (R/W)

ADC_SAR1_INV Configures whether to invert the data of HP ADC1.

0: Do not invert.

1: Invert the date.

(R/W)

ADC_SAR2_INV Configures whether to to invert the data of HP ADC2.

0: Do not invert.

1: Invert the date.

(R/W)

ADC_TIMER_TARGET Configures HP ADC timer target. (R/W)

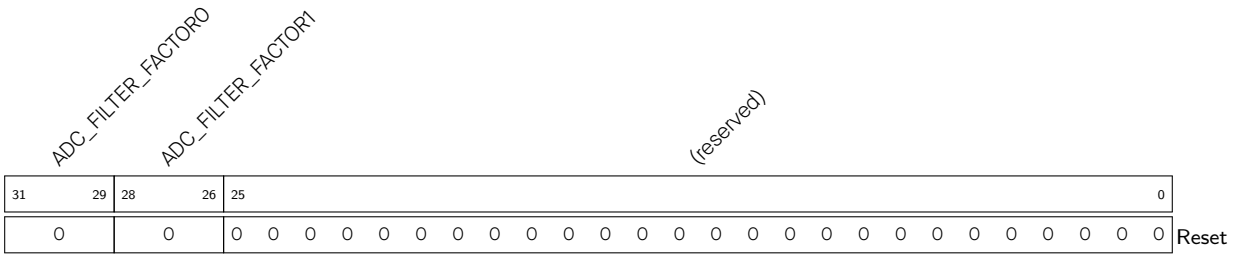
ADC_TIMER_EN Configures whether to enable HP ADC timer trigger.

0: Disable

1: Enable

(R/W)

Register 65.3. ADC_FILTER_CTRL1_REG (0x0008)



ADC_FILTER_FACTOR1 Configures the filter coefficient k for HP ADC filter 1.

0: k=0 (i.e., the filter is disabled)

1: k=2

2: k=4

3: k=8

4: k=16

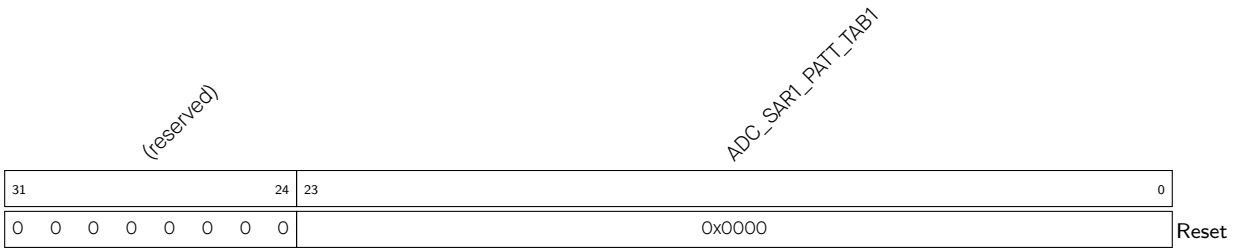
5: k=32

6: k=64

(R/W)

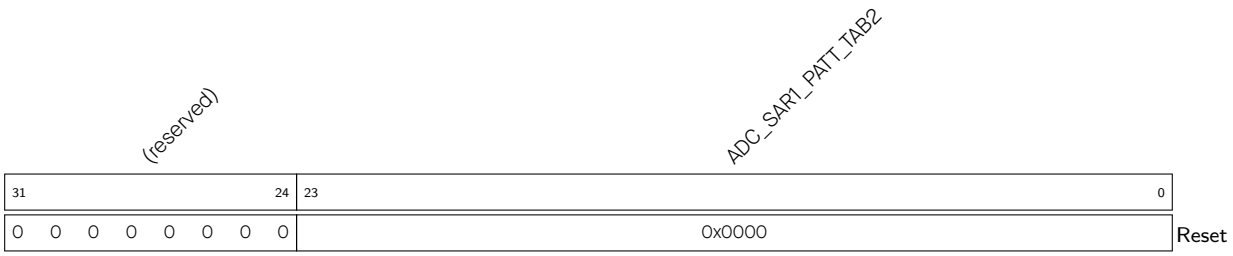
ADC_FILTER_FACTOR0 Configures the filter coefficient k for HP ADC filter 0 (same as above). (R/W)

Register 65.4. ADC_SAR1_PATT_TAB1_REG (0x0018)



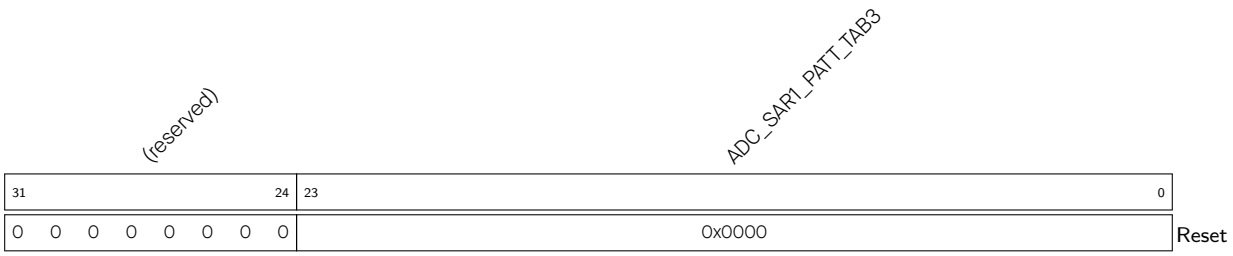
ADC_SAR1_PATT_TAB1 Configures entries 0 ~ 3 for HP ADC FSM1. Each entry is 6-bit wide. See Section 65.5.6 *HP ADC Pattern Table*. (R/W)

Register 65.5. ADC_SAR1_PATT_TAB2_REG (0x001C)



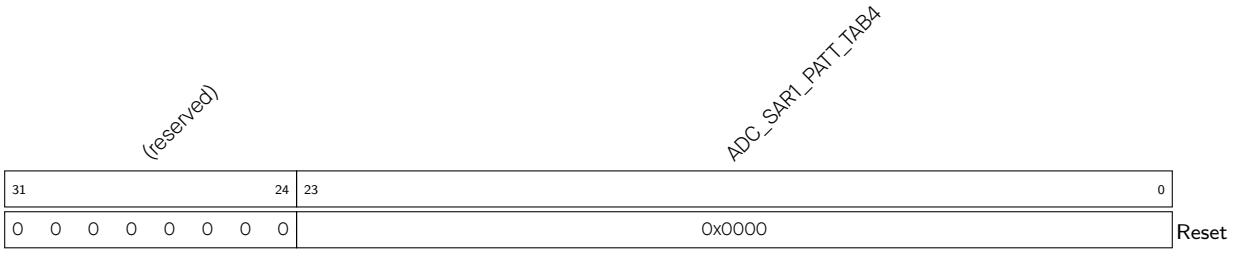
ADC_SAR1_PATT_TAB2 Configures entries 4 ~ 7 for HP ADC FSM1. Each entry is 6-bit wide. See Section 65.5.6 *HP ADC Pattern Table*. (R/W)

Register 65.6. ADC_SAR1_PATT_TAB3_REG (0x0020)



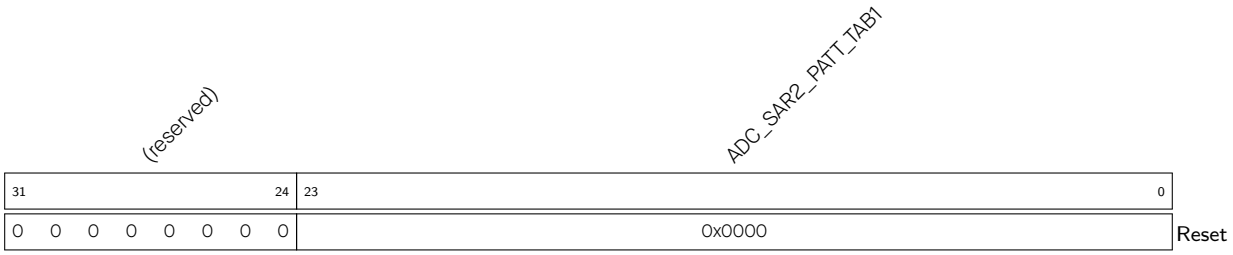
ADC_SAR1_PATT_TAB3 Configures entries 8 ~ 11 for HP ADC FSM1. Each entry is 6-bit wide. See Section 65.5.6 *HP ADC Pattern Table*. (R/W)

Register 65.7. ADC_SAR1_PATT_TAB4_REG (0x0024)



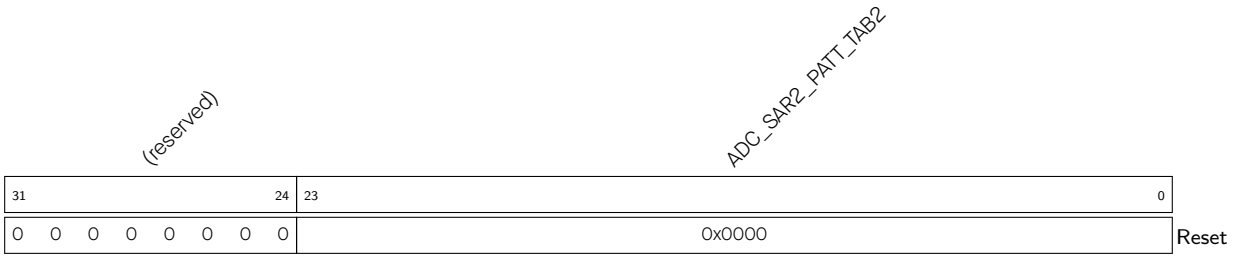
ADC_SAR1_PATT_TAB4 Configures entries 12 ~ 15 for HP ADC FSM1. Each entry is 6-bit wide. See Section 65.5.6 *HP ADC Pattern Table*. (R/W)

Register 65.8. ADC_SAR2_PATT_TAB1_REG (0x0028)



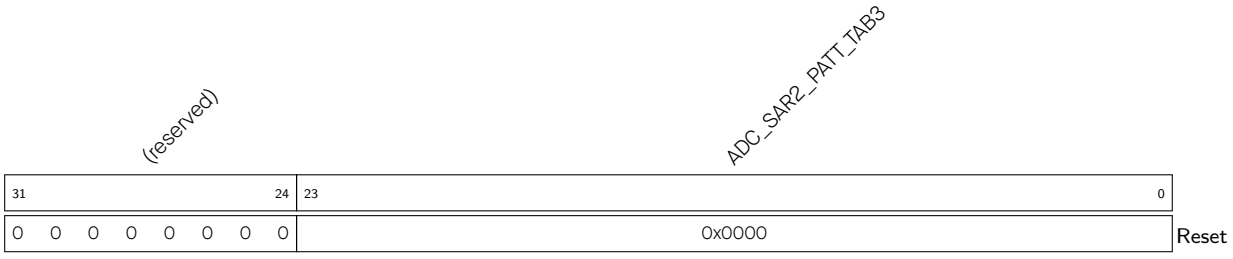
ADC_SAR2_PATT_TAB1 Configures entries 0 ~ 3 for HP ADC FSM2. Each entry is 6-bit wide. See Section 65.5.6 *HP ADC Pattern Table*. (R/W)

Register 65.9. ADC_SAR2_PATT_TAB2_REG (0x002C)



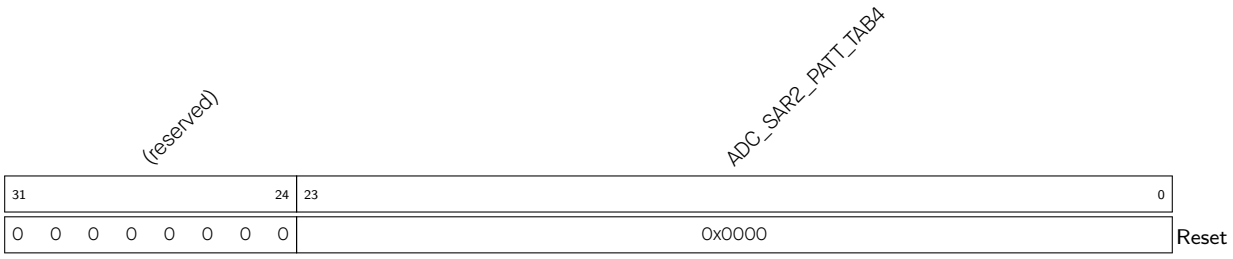
ADC_SAR2_PATT_TAB2 Configures entries 4 ~ 7 for HP ADC FSM2. Each entry is 6-bit wide. See Section 65.5.6 *HP ADC Pattern Table*. (R/W)

Register 65.10. ADC_SAR2_PATT_TAB3_REG (0x0030)



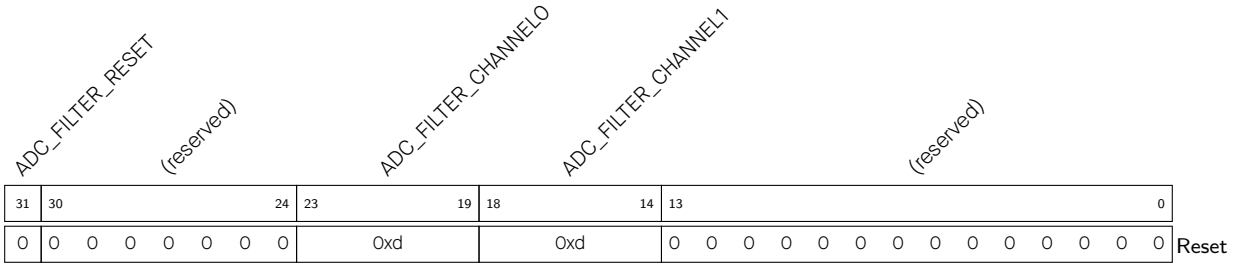
ADC_SAR2_PATT_TAB3 Configures entries 8 ~ 11 for HP ADC FSM2. Each entry is 6-bit wide. See Section 65.5.6 *HP ADC Pattern Table*. (R/W)

Register 65.11. ADC_SAR2_PATT_TAB4_REG (0x0034)



ADC_SAR2_PATT_TAB4 Configures entries 12 ~ 15 for HP ADC FSM2. Each entry is 6-bit wide. See Section 65.5.6 *HP ADC Pattern Table*. (R/W)

Register 65.12. ADC_FILTER_CTRL0_REG (0x003C)



ADC_FILTER_CHANNEL0 Configures the ADC channel for filter 0. Value 0 ~ 7 corresponds to channel 0 ~ 7 in HP ADC1, value 10 ~ 15 corresponds to channel 0 ~ 5 in HP ADC2. (R/W)

ADC_FILTER_CHANNEL1 Configure the ADC channel for filter 1. Same as above. (R/W)

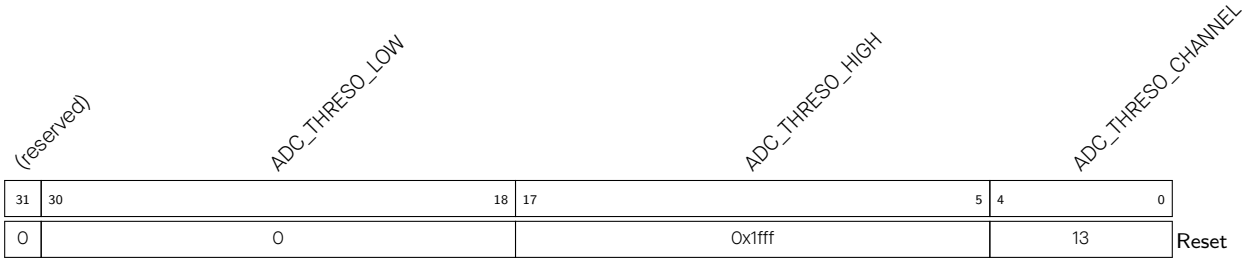
ADC_FILTER_RESET Configures whether to reset the filter.

0: No effect

1: Reset

(R/W)

Register 65.13. ADC_THRES0_CTRL_REG (0x0044)

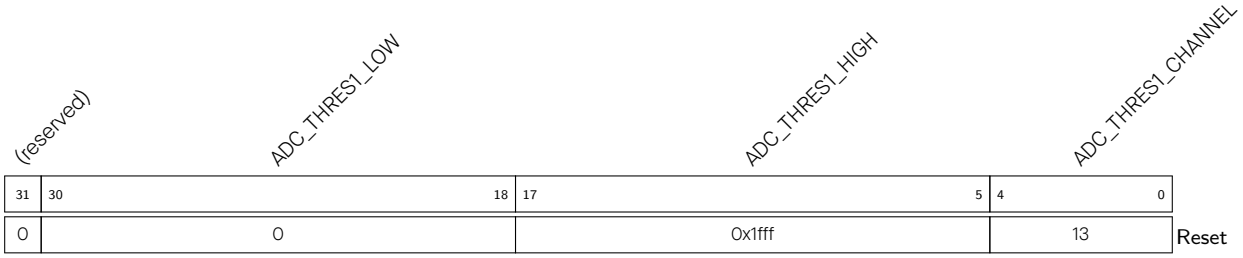


ADC_THRES0_CHANNEL Configures the ADC channel for threshold monitor 0. Value 0 ~ 7 corresponds to channel 0 ~ 7 in HP ADC1, value 10 ~ 15 corresponds to channel 0 ~ 5 in HP ADC2. (R/W)

ADC_THRES0_HIGH Configures the high threshold for HP ADC threshold monitor 0. (R/W)

ADC_THRES0_LOW Configures the low threshold for HP ADC threshold monitor 0. (R/W)

Register 65.14. ADC_THRES1_CTRL_REG (0x0048)

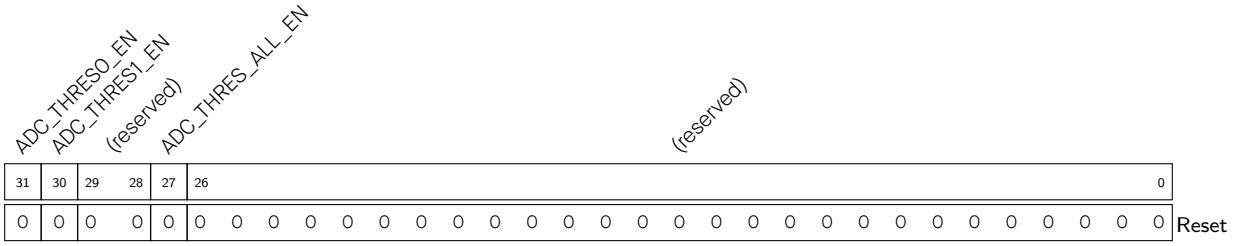


ADC_THRES1_CHANNEL Configures the ADC channel for threshold monitor 1. Value 0 ~ 7 corresponds to channel 0 ~ 7 in HP ADC1, value 10 ~ 15 corresponds to channel 0 ~ 5 in HP ADC2. (R/W)

ADC_THRES1_HIGH Configures the high threshold for HP ADC threshold monitor 1. (R/W)

ADC_THRES1_LOW Configures the low threshold for HP ADC threshold monitor 1. (R/W)

Register 65.15. ADC_THRES_CTRL_REG (0x004C)



ADC_THRES_ALL_EN Configures whether to enable the threshold monitoring for all configured channels.

0: Disable

1: Enable

(R/W)

ADC_THRES1_EN Configures whether to enable threshold monitor 1.

0: Disable

1: Enable

(R/W)

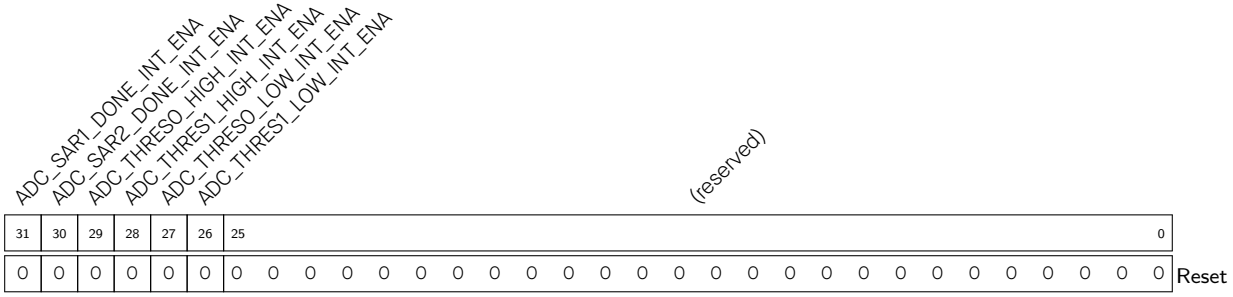
ADC_THRES0_EN Configures whether to enable threshold monitor 0.

0: Disable

1: Enable

(R/W)

Register 65.16. ADC_INT_ENA_REG (0x0050)



ADC_THRES1_LOW_INT_ENA Enable bit of ADC_THRES1_LOW_INT. (R/W)

ADC_THRES0_LOW_INT_ENA Enable bit of ADC_THRES0_LOW_INT. (R/W)

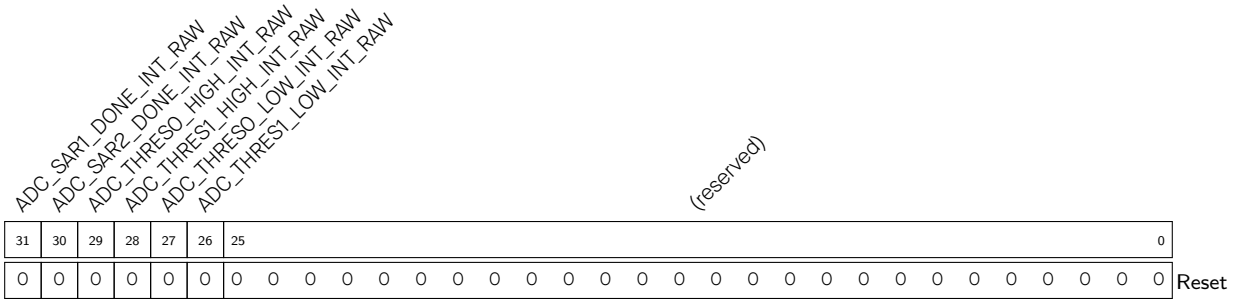
ADC_THRES1_HIGH_INT_ENA Enable bit of ADC_THRES1_HIGH_INT. (R/W)

ADC_THRES0_HIGH_INT_ENA Enable bit of ADC_THRES0_HIGH_INT. (R/W)

ADC_SAR2_DONE_INT_ENA Enable bit of ADC_SAR2_DONE_INT. (R/W)

ADC_SAR1_DONE_INT_ENA Enable bit of ADC_SAR1_DONE_INT. (R/W)

Register 65.17. ADC_INT_RAW_REG (0x0054)



ADC_THRES1_LOW_INT_RAW Raw bit of ADC_THRES1_LOW_INT. (R/WTC/SS)

ADC_THRES0_LOW_INT_RAW Raw bit of ADC_THRES0_LOW_INT. (R/WTC/SS)

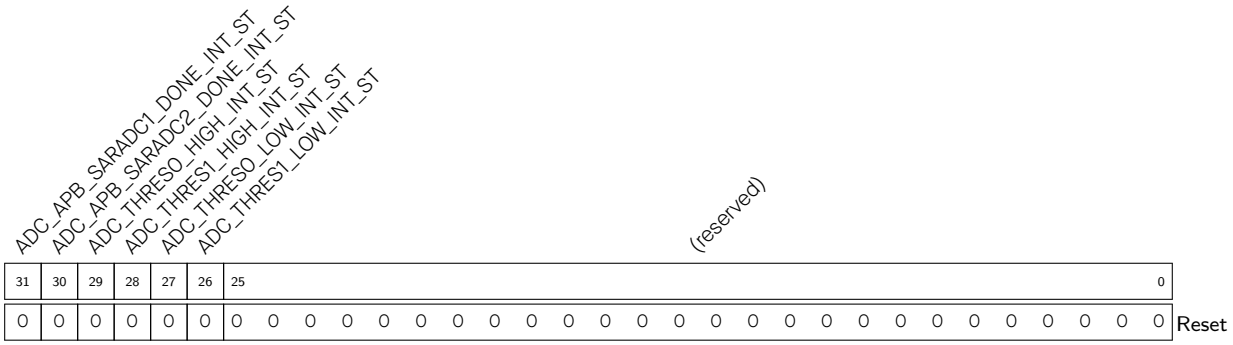
ADC_THRES1_HIGH_INT_RAW Raw bit of ADC_THRES1_HIGH_INT. (R/WTC/SS)

ADC_THRES0_HIGH_INT_RAW Raw bit of ADC_THRES0_HIGH_INT. (R/WTC/SS)

ADC_SAR2_DONE_INT_RAW Raw bit of ADC_SAR2_DONE_INT. (R/WTC/SS)

ADC_SAR1_DONE_INT_RAW Raw bit of ADC_SAR1_DONE_INT. (R/WTC/SS)

Register 65.18. ADC_INT_ST_REG (0x0058)



ADC_THRES1_LOW_INT_ST Status bit of ADC_THRES1_LOW_INT. (RO)

ADC_THRES0_LOW_INT_ST Status bit of ADC_THRES0_LOW_INT. (RO)

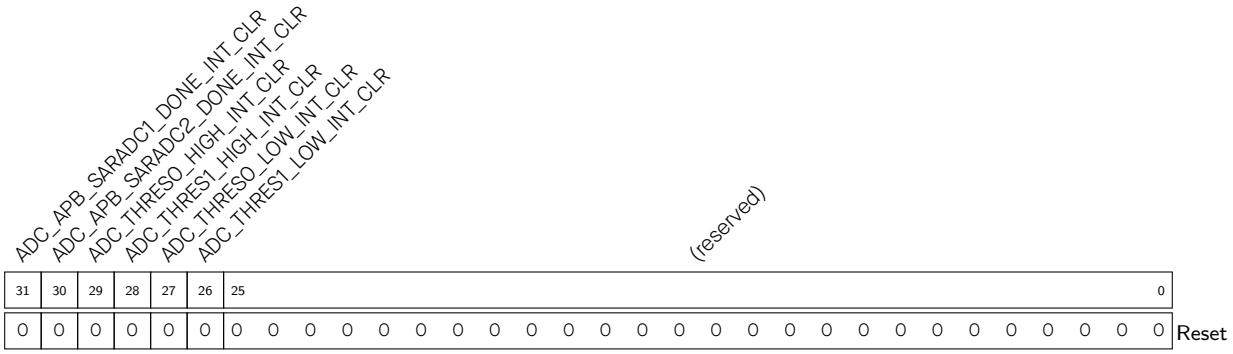
ADC_THRES1_HIGH_INT_ST Status bit of ADC_THRES1_HIGH_INT. (RO)

ADC_THRES0_HIGH_INT_ST Status bit of ADC_THRES0_HIGH_INT. (RO)

ADC_APB_SARADC2_DONE_INT_ST Status bit of ADC_SAR2_DONE_INT. (RO)

ADC_APB_SARADC1_DONE_INT_ST Status bit of ADC_SAR1_DONE_INT. (RO)

Register 65.19. ADC_INT_CLR_REG (0x005C)



ADC_THRES1_LOW_INT_CLR Clear bit of ADC_THRES1_LOW_INT. (WT)

ADC_THRES0_LOW_INT_CLR Clear bit of ADC_THRES0_LOW_INT. (WT)

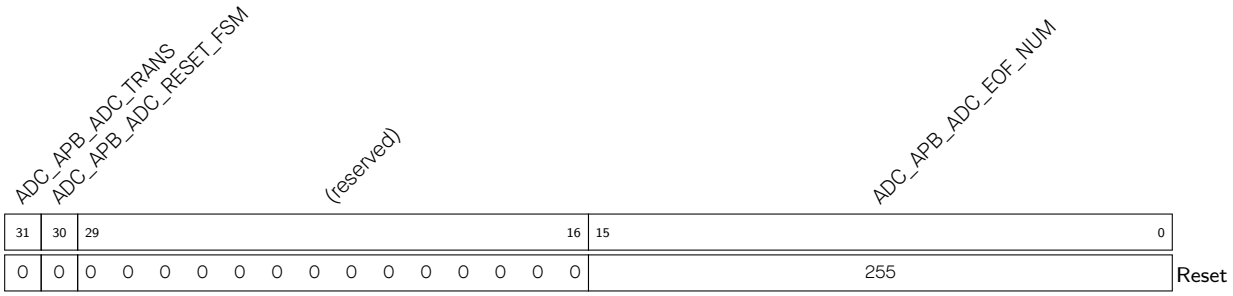
ADC_THRES1_HIGH_INT_CLR Clear bit of ADC_THRES1_HIGH_INT. (WT)

ADC_THRES0_HIGH_INT_CLR Clear bit of ADC_THRES0_HIGH_INT. (WT)

ADC_APB_SARADC2_DONE_INT_CLR Clear bit of ADC_SAR2_DONE_INT. (WT)

ADC_APB_SARADC1_DONE_INT_CLR Clear bit of ADC_SAR1_DONE_INT. (WT)

Register 65.20. ADC_DMA_CONF_REG (0x0060)



ADC_APB_ADC_EOF_NUM Configures the number of samples. When the sampling reaches the configured number, the EOF flag bit sent to GDMA will be pulled high. (R/W)

ADC_APB_ADC_RESET_FSM Configures whether to reset the status of HP ADC controller.

0: No effect

1: Reset

(R/W)

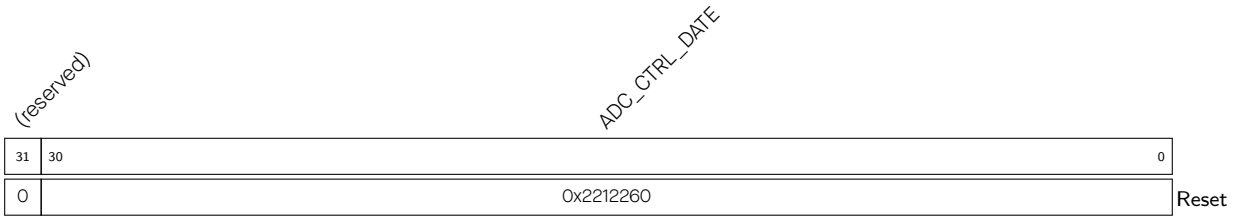
ADC_APB_ADC_TRANS Configures whether to let HP ADC controller use GDMA.

0: No effect

1: HP ADC controller uses DMA

(R/W)

Register 65.21. ADC_CTRL_DATE_REG (0x03FC)



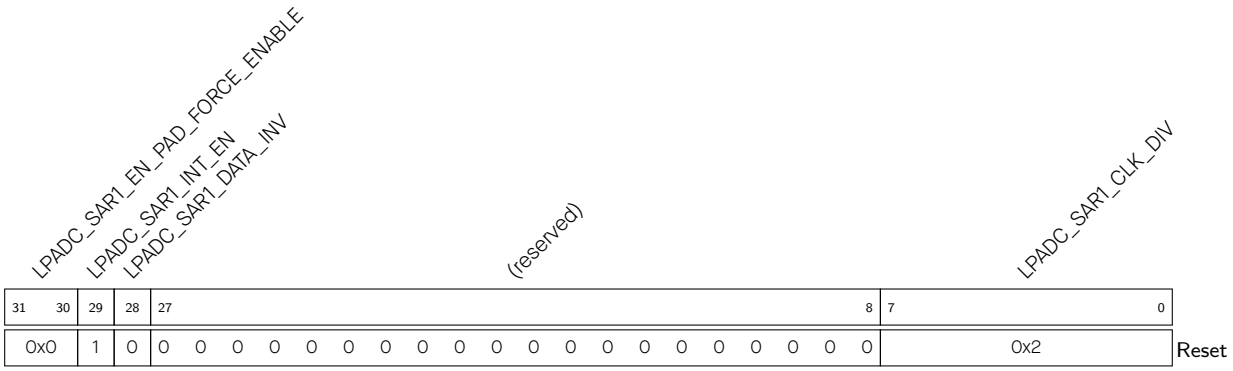
ADC_CTRL_DATE HP ADC version control register. (RO)

65.10.2 LP ADC Registers

The addresses in the following section are relative to LP ADC's base address provided in Table 9.3-2 in Chapter 9 *System and Memory*.

For how to program reserved fields, please refer to Section Programming Reserved Register Field.

Register 65.22. LPADC_READER1_CTRL_REG (0x0000)



LPADC_SAR1_CLK_DIV Configures the division of LP ADC1's clock LPADC_SARCLK. (R/W)

LPADC_SAR1_DATA_INV Configures whether to to invert the data of LP ADC1.

0: Do not invert the data.

1: Invert the data.

(R/W)

LPADC_SAR1_INT_EN Enables LP ADC1 to send out interrupt. (R/W)

LPADC_SAR1_EN_PAD_FORCE_ENABLE Configures whether to use software to force enable LPADC_SAR1_EN_PAD.

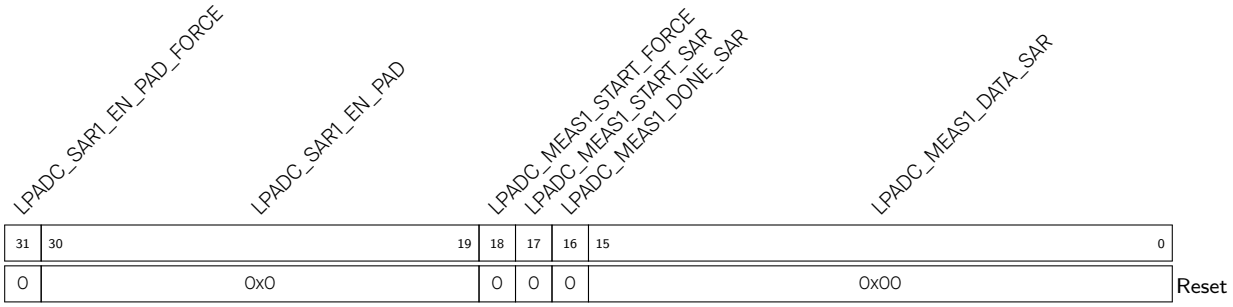
2: Force off

3: Force enable

Other values: Hardware control (enabled during sampling)

(R/W)

Register 65.23. LPADC_MEAS1_CTRL2_REG (0x000C)



LPADC_MEAS1_DATA_SAR Stores LP ADC1's conversion data in one-shot sampling mode. (RO)

LPADC_MEAS1_DONE_SAR Indicates whether LP ADC1 conversion is done. (RO)

LPADC_MEAS1_START_SAR Configures whether to start LP ADC1 by software.

0: No effect

1: Start LP ADC1 by software

Valid only when LPADC_MEAS1_START_FORCE = 1.

(R/W)

LPADC_MEAS1_START_FORCE Configures whether to use software to enable LP ADC1 sampling.

0: Select FSM to start LP ADC1 sampling

1: Select software to start LP ADC1 sampling

(R/W)

LPADC_SAR1_EN_PAD Configures the sampling channel for LP ADC1. Value 0 ~ 7 corresponds to channel 0 ~ 7. (R/W)

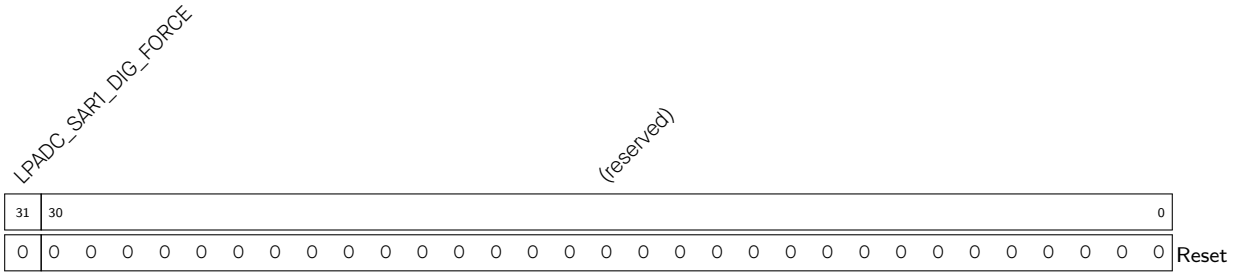
LPADC_SAR1_EN_PAD_FORCE Configures whether to use software to select the sampling channel for LP ADC1.

0: No effect

1: Use software to control

(R/W)

Register 65.24. LPADC_MEAS1_MUX_REG (0x0010)



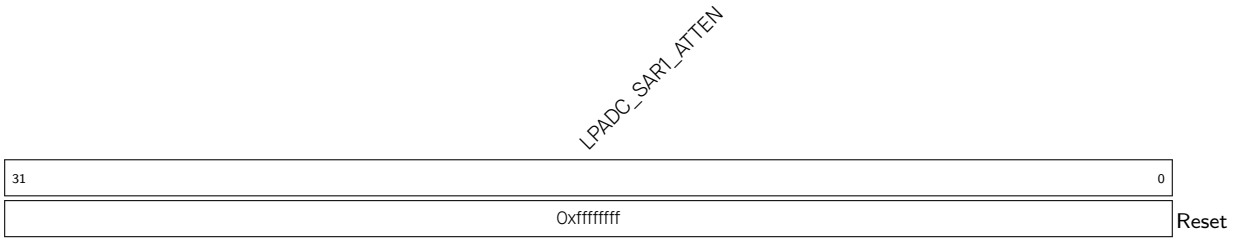
LPADC_SAR1_DIG_FORCE Configures whether to use HP ADC1 to control SAR ADC1.

0: SAR ADC1 controlled by LP ADC1 CTRL

1: SAR ADC1 controlled by HP ADC1 CTRL

(R/W)

Register 65.25. LPADC_ATTEN1_REG (0x0014)



LPADC_SAR1_ATTEN Configures the attenuation value for one-shot sampling of LP ADC1.

0: 0 dB

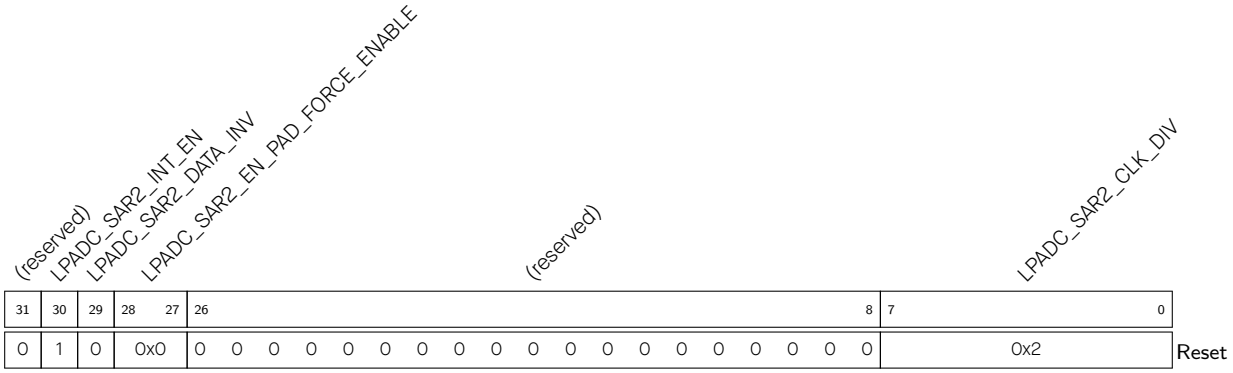
1: 2.5 dB

2: 6 dB

3: 12 dB

(R/W)

Register 65.26. LPADC_READER2_CTRL_REG (0x0024)



LPADC_SAR2_CLK_DIV Configures the division of LP ADC2's clock LPADC_SARCLK. (R/W)

LPADC_SAR2_EN_PAD_FORCE_ENABLE Configures whether to use software to force enable LPADC_SAR2_EN_PAD.

2: Force off

3: Force enable

Other values: Hardware control (enabled during sampling)

(R/W)

LPADC_SAR2_DATA_INV Configures whether to invert the data of LP ADC2.

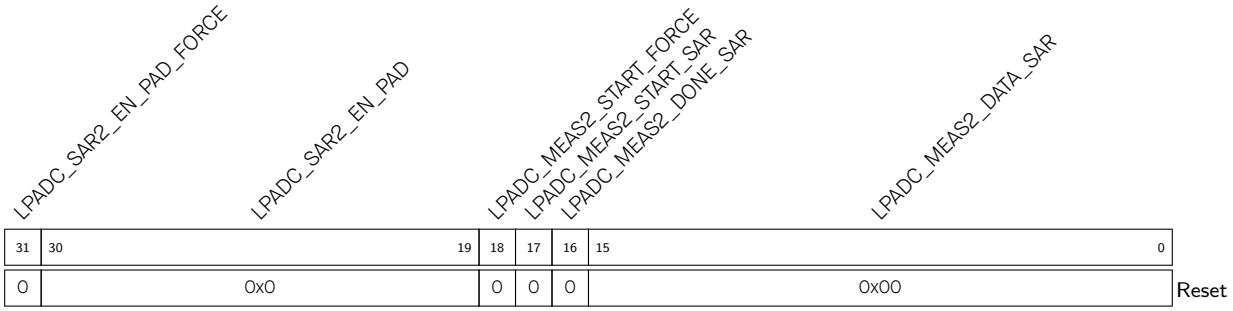
0: Do not invert the data.

1: Invert the data.

(R/W)

LPADC_SAR2_INT_EN Enable LP ADC2 to send out interrupt. (R/W)

Register 65.27. LPADC_MEAS2_CTRL2_REG (0x0030)



LPADC_MEAS2_DATA_SAR Stores LP ADC2's conversion data in one-shot sampling mode. (RO)

LPADC_MEAS2_DONE_SAR Indicates whether LP ADC2 conversion is done. (RO)

LPADC_MEAS2_START_SAR Configures whether to start LP ADC2 by software.

0: No effect

1: Start LP ADC2 by software

Valid only when LPADC_MEAS2_START_FORCE = 1.

(R/W)

LPADC_MEAS2_START_FORCE Configures whether to use software to enable LP ADC2 sampling.

0: Select FSM to start LP ADC2 sampling

1: Select software to start LP ADC2 sampling

(R/W)

LPADC_SAR2_EN_PAD Configures the sampling channel for LP ADC2. Value 2 ~ 7 corresponds to channel 0 ~ 5. (R/W)

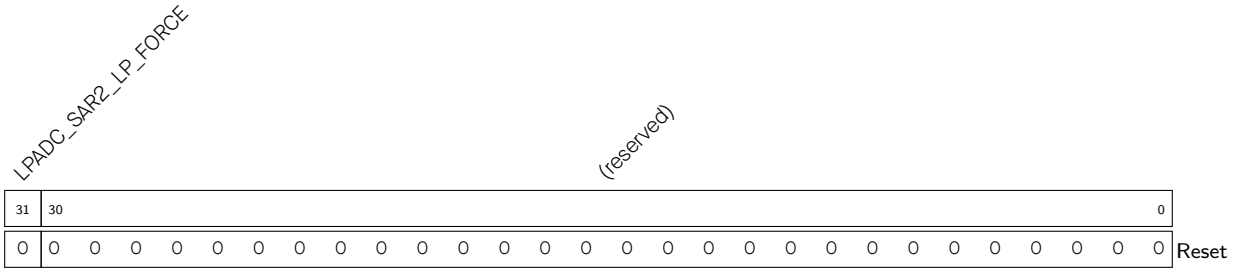
LPADC_SAR2_EN_PAD_FORCE Configures whether to use software to select the sampling channel for LP ADC1.

0: No effect

1: Use software to control

(R/W)

Register 65.28. LPADC_MEAS2_MUX_REG (0x0034)



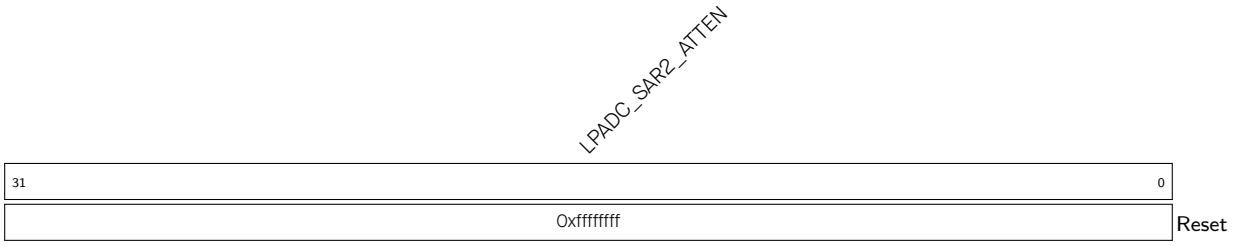
LPADC_SAR2_LP_FORCE Configures whether to disable HP ADC2 Controller to control SAR ADC2.

0: Do not disable HP ADC2 Controller.

1: Disable HP ADC2 Controller.

(R/W)

Register 65.29. LPADC_ATTEN2_REG (0x0038)



LPADC_SAR2_ATTEN Configures the attenuation value for one-shot sampling of LP ADC2.

0: 0 dB

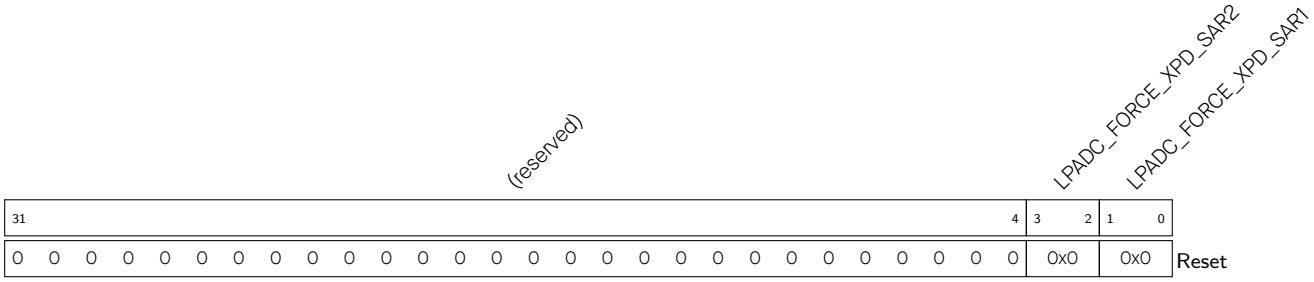
1: 2.5 dB

2: 6 dB

3: 12 dB

(R/W)

Register 65.30. LPADC_FORCE_WPD_SAR_REG (0x003C)



LPADC_FORCE_XPD_SAR1 Configures the power control for LP ADC1.

3: Software control, force on.

2: Software control, force off.

1/0: Hardware control.

(R/W)

LPADC_FORCE_XPD_SAR2 Configures the power control for LP ADC2.

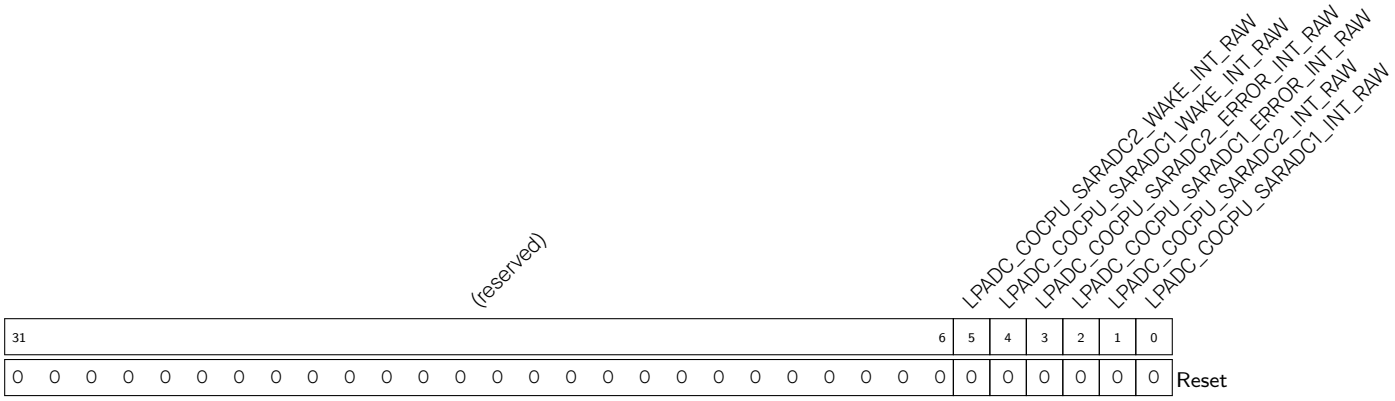
3: Software control, force on.

2: Software control, force off.

1/0: Hardware control.

(R/W)

Register 65.31. LPADC_COCPU_INT_RAW_REG (0x0048)



LPADC_COCPU_SARADC1_INT_RAW Raw bit of LPADC_COCPU_SARADC1_DONE_INT. (R/WTC/SS)

LPADC_COCPU_SARADC2_INT_RAW Raw bit of LPADC_COCPU_SARADC2_DONE_INT. (R/WTC/SS)

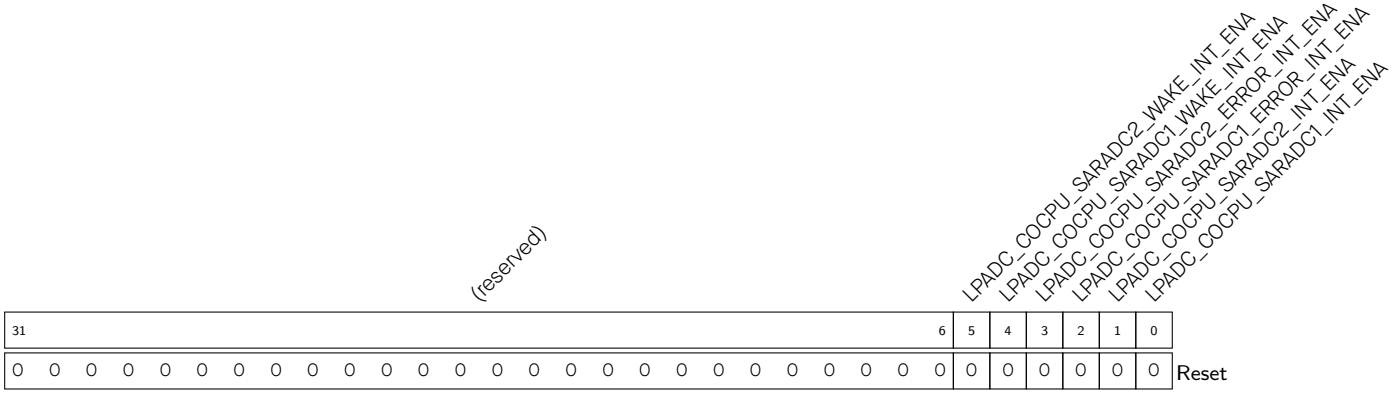
LPADC_COCPU_SARADC1_ERROR_INT_RAW Raw bit of LPADC_COCPU_SARADC1_ERROR_INT. (R/WTC/SS)

LPADC_COCPU_SARADC2_ERROR_INT_RAW Raw bit of LPADC_COCPU_SARADC2_ERROR_INT. (R/WTC/SS)

LPADC_COCPU_SARADC1_WAKE_INT_RAW Raw bit of LPADC_COCPU_SARADC1_WAKE_INT. (R/WTC/SS)

LPADC_COCPU_SARADC2_WAKE_INT_RAW Raw bit of LPADC_COCPU_SARADC2_WAKE_INT. (R/WTC/SS)

Register 65.32. LPADC_INT_ENA_REG (0x004C)



LPADC_COCPU_SARADC1_INT_ENA Enable bit of LPADC_COCPU_SARADC1_DONE_INT. (R/WTC)

LPADC_COCPU_SARADC2_INT_ENA Enable bit of LPADC_COCPU_SARADC2_DONE_INT. (R/WTC)

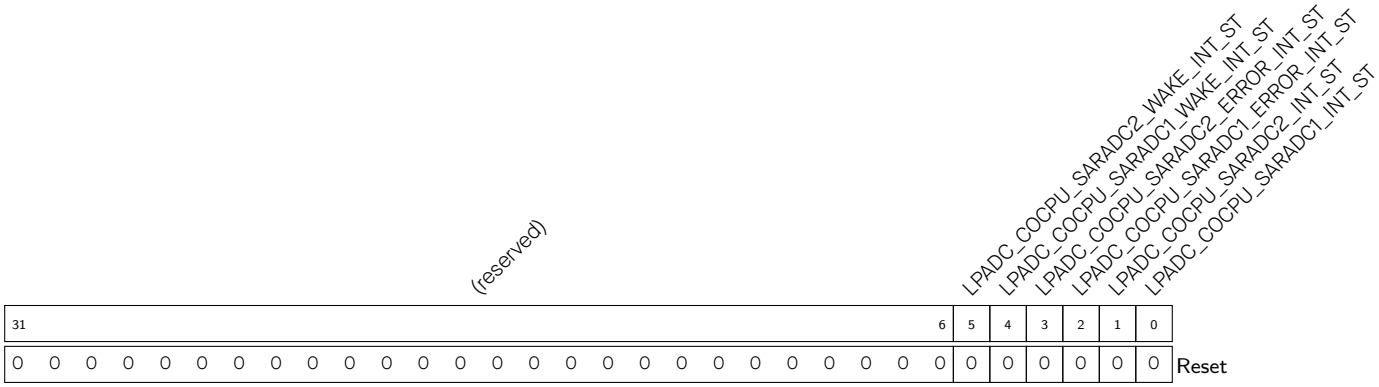
LPADC_COCPU_SARADC1_ERROR_INT_ENA Enable bit of LPADC_COCPU_SARADC1_ERROR_INT. (R/WTC)

LPADC_COCPU_SARADC2_ERROR_INT_ENA Enable bit of LPADC_COCPU_SARADC2_ERROR_INT. (R/WTC)

LPADC_COCPU_SARADC1_WAKE_INT_ENA Enable bit of LPADC_COCPU_SARADC1_WAKE_INT. (R/WTC)

LPADC_COCPU_SARADC2_WAKE_INT_ENA Enable bit of LPADC_COCPU_SARADC2_WAKE_INT. (R/WTC)

Register 65.33. LPADC_INT_ST_REG (0x0050)



LPADC_COCPU_SARADC1_INT_ST Status bit of LPADC_COCPU_SARADC1_DONE_INT. (RO)

LPADC_COCPU_SARADC2_INT_ST Status bit of LPADC_COCPU_SARADC2_DONE_INT. (RO)

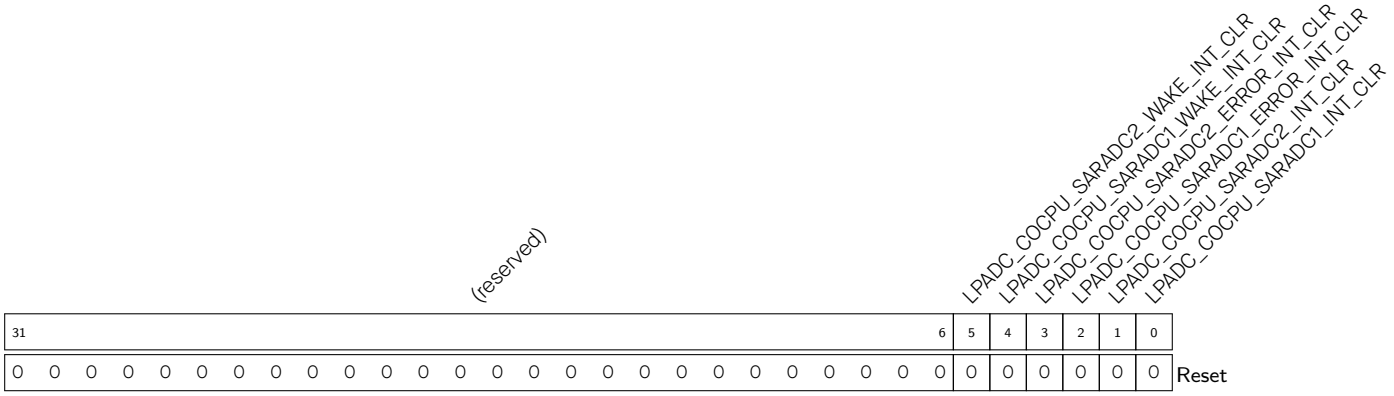
LPADC_COCPU_SARADC1_ERROR_INT_ST Status bit of LPADC_COCPU_SARADC1_ERROR_INT. (RO)

LPADC_COCPU_SARADC2_ERROR_INT_ST Status bit of LPADC_COCPU_SARADC2_ERROR_INT. (RO)

LPADC_COCPU_SARADC1_WAKE_INT_ST Status bit of LPADC_COCPU_SARADC1_WAKE_INT. (RO)

LPADC_COCPU_SARADC2_WAKE_INT_ST Status bit of LPADC_COCPU_SARADC2_WAKE_INT. (RO)

Register 65.34. LPADC_INT_CLR_REG (0x0054)



LPADC_COCPU_SARADC1_INT_CLR Clear bit of LPADC_COCPU_SARADC1_DONE_INT. (WT)

LPADC_COCPU_SARADC2_INT_CLR Clear bit of LPADC_COCPU_SARADC2_DONE_INT. (WT)

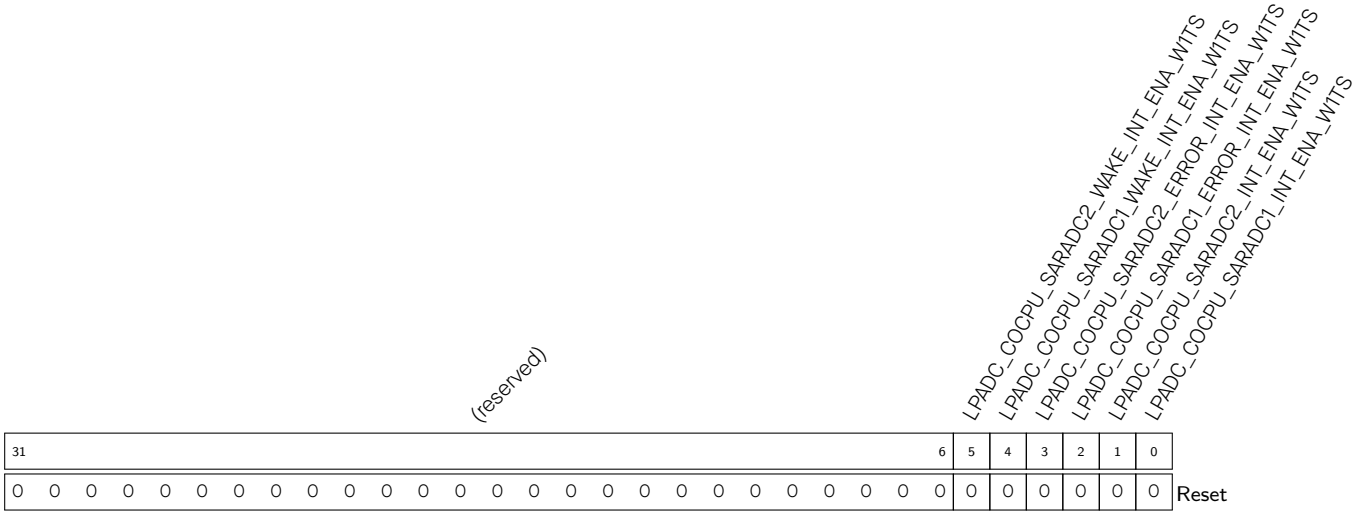
LPADC_COCPU_SARADC1_ERROR_INT_CLR Clear bit of LPADC_COCPU_SARADC1_ERROR_INT. (WT)

LPADC_COCPU_SARADC2_ERROR_INT_CLR Clear bit of LPADC_COCPU_SARADC2_ERROR_INT. (WT)

LPADC_COCPU_SARADC1_WAKE_INT_CLR Clear bit of LPADC_COCPU_SARADC1_WAKE_INT. (WT)

LPADC_COCPU_SARADC2_WAKE_INT_CLR Clear bit of LPADC_COCPU_SARADC2_WAKE_INT. (WT)

Register 65.35. LPADC_INT_ENA_W1TS_REG (0x0058)



LPADC_COCPU_SARADC1_INT_ENA_W1TS Write 1 to enable LPADC_COCPU_SARADC1_INT_ENA. (WT)

LPADC_COCPU_SARADC2_INT_ENA_W1TS Write 1 to enable LPADC_COCPU_SARADC2_INT_ENA. (WT)

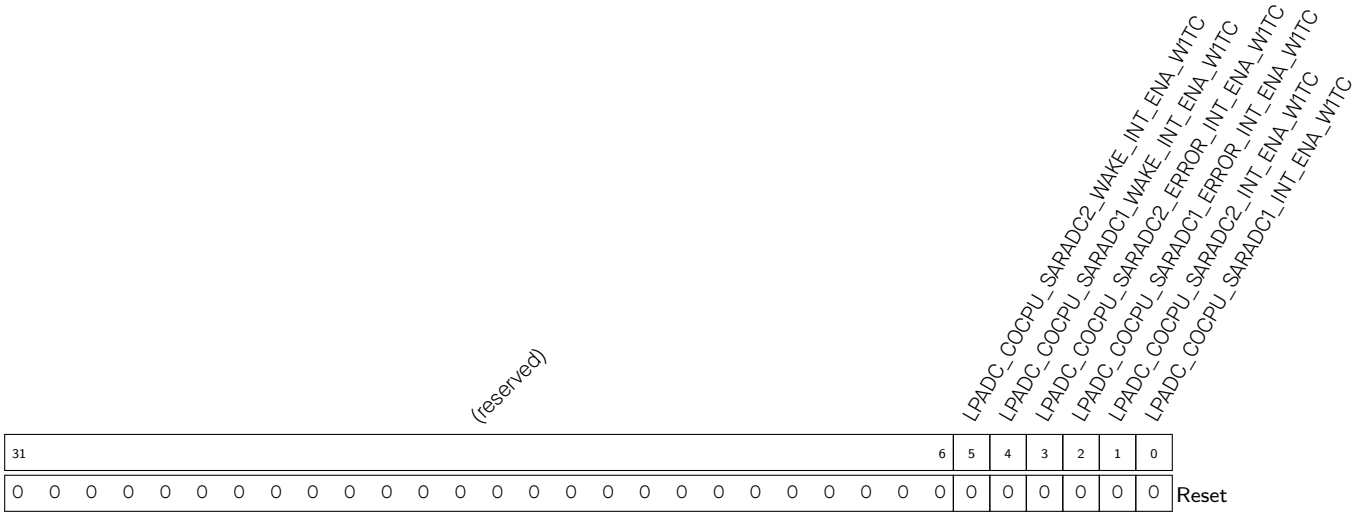
LPADC_COCPU_SARADC1_ERROR_INT_ENA_W1TS Write 1 to enable LPADC_COCPU_SARADC1_ERROR_INT_ENA. (WT)

LPADC_COCPU_SARADC2_ERROR_INT_ENA_W1TS Write 1 to enable LPADC_COCPU_SARADC2_ERROR_INT_ENA. (WT)

LPADC_COCPU_SARADC1_WAKE_INT_ENA_W1TS Write 1 to enable LPADC_COCPU_SARADC1_WAKE_INT_ENA. (WT)

LPADC_COCPU_SARADC2_WAKE_INT_ENA_W1TS Write 1 to enable LPADC_COCPU_SARADC2_WAKE_INT_ENA. (WT)

Register 65.36. LPADC_INT_ENA_W1TC_REG (0x005C)



LPADC_COCPU_SARADC1_INT_ENA_W1TC Write 1 to clear LPADC_COCPU_SARADC1_INT_ENA. (WT)

LPADC_COCPU_SARADC2_INT_ENA_W1TC Write 1 to clear LPADC_COCPU_SARADC2_INT_ENA. (WT)

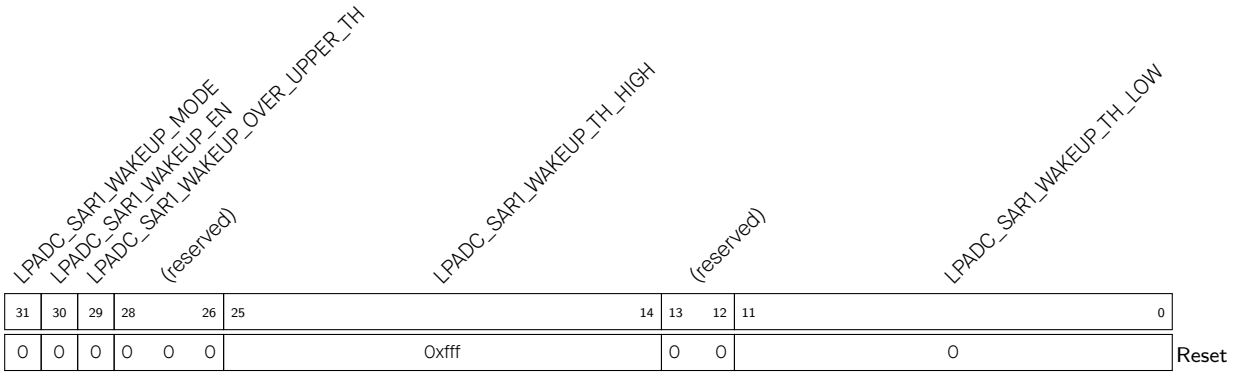
LPADC_COCPU_SARADC1_ERROR_INT_ENA_W1TC Write 1 to clear LPADC_COCPU_SARADC1_ERROR_INT_ENA. (WT)

LPADC_COCPU_SARADC2_ERROR_INT_ENA_W1TC Write 1 to clear LPADC_COCPU_SARADC2_ERROR_INT_ENA. (WT)

LPADC_COCPU_SARADC1_WAKE_INT_ENA_W1TC Write 1 to clear LPADC_COCPU_SARADC1_WAKE_INT_ENA. (WT)

LPADC_COCPU_SARADC2_WAKE_INT_ENA_W1TC Write 1 to clear LPADC_COCPU_SARADC2_WAKE_INT_ENA. (WT)

Register 65.37. LPADC_WAKEUP1_REG (0x0060)



LPADC_SAR1_WAKEUP_TH_LOW Configures the low threshold for LP ADC1's wake-up function. (R/W)

LPADC_SAR1_WAKEUP_TH_HIGH Configures the high threshold for LP ADC1's wake-up function. (R/W)

LPADC_SAR1_WAKEUP_OVER_UPPER_TH Indicates whether the wake-up event is triggered by the conversion data exceeding high threshold. (RO)

LPADC_SAR1_WAKEUP_EN Configures whether to enable wake-up function for LP ADC1.

0: Disable

1: Enable

(R/W)

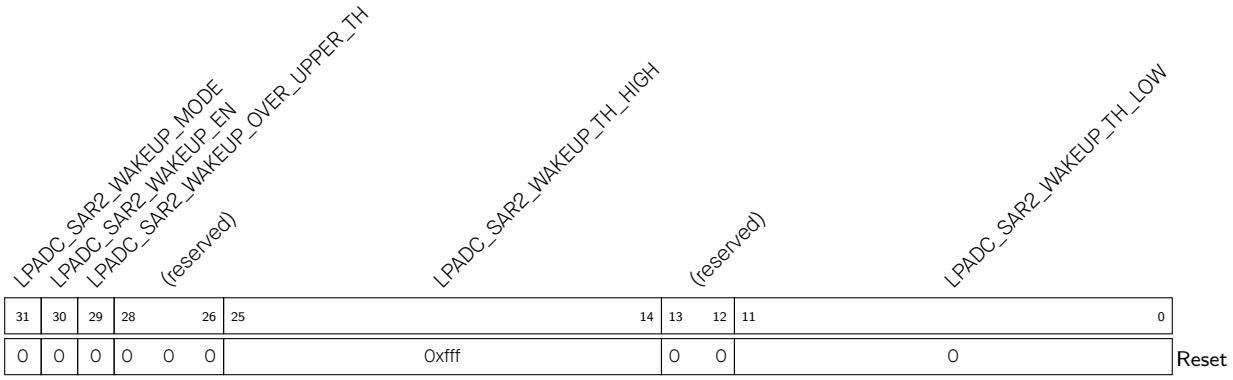
LPADC_SAR1_WAKEUP_MODE Selects the wake-up mode for automatic monitoring. Valid only when LPADC_SAR1_WAKEUP_EN=1.

0: Absolute value mode

1: Change value mode

(R/W)

Register 65.38. LPADC_WAKEUP2_REG (0x0064)



LPADC_SAR2_WAKEUP_TH_LOW Configures the low threshold for LP ADC2's wake-up function. (R/W)

LPADC_SAR2_WAKEUP_TH_HIGH Configures the high threshold for LP ADC2's wake-up function. (R/W)

LPADC_SAR2_WAKEUP_OVER_UPPER_TH Indicates whether the wake-up event is triggered by the conversion data exceeding high threshold. (RO)

LPADC_SAR2_WAKEUP_EN Configures whether to enable wake-up function for LP ADC2.

0: Disable

1: Enable

(R/W)

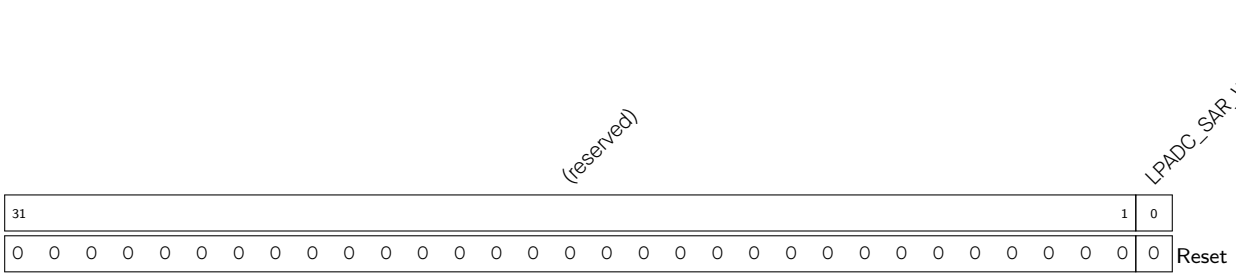
LPADC_SAR2_WAKEUP_MODE Selects the wake-up mode for automatic monitoring. Valid only when LPADC_SAR2_WAKEUP_EN=1.

0: Absolute value mode

1: Change value mode

(R/W)

Register 65.39. LPADC_WAKEUP_SEL_REG (0x0068)

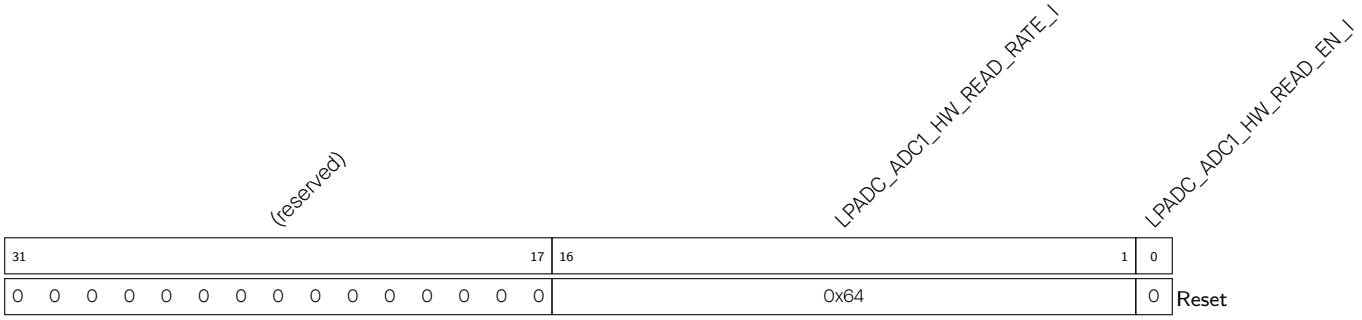


LPADC_SAR_WAKEUP_SEL Enables the wake-up function for LP ADC1 or LP ADC2. 0: LP ADC1.

1: LP ADC2.

(R/W)

Register 65.40. LPADC_SAR1_HW_WAKEUP_REG (0x006C)



LPADC_ADC1_HW_READ_EN_I Configures whether to enable automatic monitoring for LP ADC1.

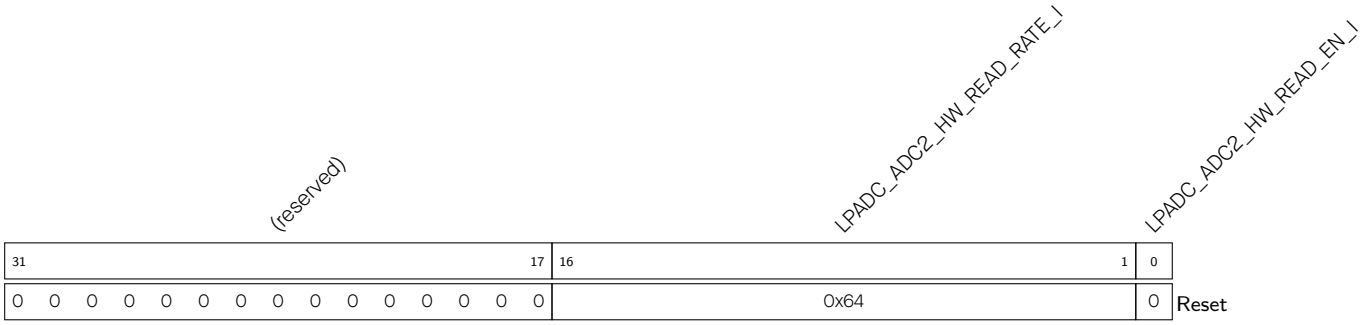
0: Disable

1: Enable

(R/W)

LPADC_ADC1_HW_READ_RATE_I Configures the sampling rate for hardware-triggered automatic monitoring for LP ADC1. The sampling period = configured value × LP ADC1's working clock cycle. (R/W)

Register 65.41. LPADC_SAR2_HW_WAKEUP_REG (0x0070)



LPADC_ADC2_HW_READ_EN_I Configures whether to enable automatic monitoring for LP ADC2.

0: Disable

1: Enable

(R/W)

LPADC_ADC2_HW_READ_RATE_I Configures the sampling rate for hardware-triggered automatic monitoring for LP ADC2. The sampling period = configured value × LP ADC2's working clock cycle. (R/W)

Chapter 66

Analog Voltage Comparator

66.1 Introduction

ESP32-P4 integrates two analog voltage comparators, i.e., analog voltage comparator *0* and analog voltage comparator *1*. These comparators rely on special pads that support voltage comparison functionality to monitor voltage changes on these pads. Each analog voltage comparator has two pads associated with it, for the main voltage and the reference voltage respectively. The voltage comparison result generated by the analog voltage comparator can be used as Event Task Matrix (ETM) events to drive ETM tasks of other peripherals or trigger interrupts.

Note:
The information provided in this chapter applies to both analog voltage comparator *0* and analog voltage comparator *1*. Unless otherwise indicated, the analog voltage comparator in this chapter refer to both analog voltage comparator *0* and analog voltage comparator *1*.

66.2 Feature List

The analog voltage comparator has the following features:

- Voltage comparison
  - Configurable voltage comparison mode
  - Configurable reference voltage
- Interrupt upon changes of voltage comparison result
- ETM event generation

66.3 Architectural Overview

Figure 66.3-1 shows the structure of the analog voltage comparator.

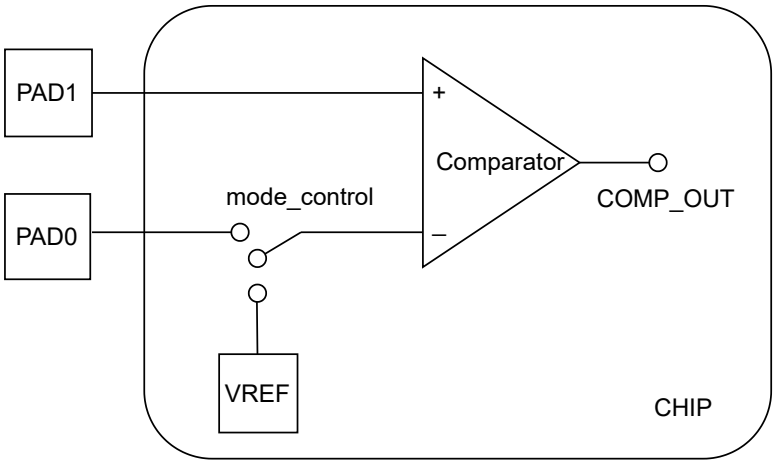


Figure 66.3-1. Analog Voltage Comparator Architecture

As shown in the figure above, the analog voltage comparator consists of:

- Two pad interfaces:
  - PAD1: Main voltage interface
  - PAD0: External reference voltage interface
- VREF: Internal reference voltage interface
- mode_control: Register-controlled switch for selecting the reference voltage source
- COMP_OUT: Comparator voltage comparison result, which is an on-chip internal signal that cannot be directly accessed externally
  - If the main voltage is higher than the reference voltage, the COMP_OUT signal is high.
  - If the main voltage is lower than the reference voltage, the COMP_OUT signal is low.

66.4 Functional Description

The analog voltage comparator has the following functions:

- **Voltage comparison**

  The analog voltage comparator relies on a specialized pads that support voltage comparison. For more information about general-purpose pads, please refer to Chapter 10 GPIO Matrix and IO MUX. Table below shows the mapping between the PAD of analog voltage comparator *0* and GPIO pins.

Table 66.4-1. Mapping Between PAD and GPIO

Analog Voltage Comparator	PAD0	PAD1
Analog voltage comparator *0*	GPIO51	GPIO52
Analog voltage comparator *1*	GPIO53	GPIO54

  The comparison mode and reference voltage are configurable. For detailed configuration procedures, see Section 66.7 Programming Procedures.

  The voltage comparison results are output as the COMP_OUT signal.

- **Voltage comparison mode control**

  The analog voltage comparator compares the main voltage with the reference voltage, which can either be internal or external. There are two voltage comparison modes depending on the reference voltage selection:

  - Comparing the main voltage with the external reference voltage.
  - Comparing the main voltage with the internal reference voltage.

  The voltage comparison mode can be configured through LPSYSTEM_MODE_COMP*n* (*n* = 0 ~ 1).

- **Reference voltage configuration**

  The internal reference voltage range is 0 ~ (0.7 × VDD_IO_6) V with a step of 0.1 × VDD_IO_6 V, where VDD_IO_6 is the supply voltage of the analog voltage comparator which is equal to the power supply voltage of the chip (usually 3.3 V). The internal reference voltage value can be configured through LPSYSTEM_DREF_COMP*n* (n*n* = 0 ~ 1).

  The external reference voltage is accessed directly from the pad with an input range of 0 ~ 0.7 x VDD_IO_6 V, same as the internal reference voltage, requiring no additional configuration.

- **Voltage comparison interrupt processing**

  The voltage comparison result, i.e., the COMP_OUT signal, can be either high or low. Whenever the output value changes, a corresponding interrupt signal is generated.

66.5 Event Task Matrix Feature

The analog voltage comparator on ESP32-P4 supports the Event Task Matrix (ETM) function, which allows analog voltage comparator ETM events to trigger any peripherals' ETM tasks. This section introduces the ETM events related to analog voltage comparator. For more information, please refer to Chapter 14 Event Task Matrix (ETM).

The analog voltage comparator can generate the following ETM events:

- GPIO_EVT_ZERO_DET_POS*n*: Indicates that the COMP_OUT signal changes from low level to high level, i.e., the main voltage changes from below to above the reference voltage.
- GPIO_EVT_ZERO_DET_NEG*n*: Indicates that the COMP_OUT signal changes from high level to low level, i.e., the main voltage changes from higher to lower than the reference voltage.

The analog voltage comparator does not support any ETM tasks.

66.6 Interrupts

ESP32-P4's analog voltage comparator can generate the GPIO_PAD_COMP_INT interrupt signal that will be sent to the Interrupt Matrix.

There are several internal interrupt sources from analog voltage comparator that can generate the above interrupt signal. The interrupt sources from analog voltage comparator are listed with their trigger conditions and the resulted interrupt signal(s) in Table 66.6-1.

Table 66.6-1. Analog Voltage Comparator's Internal Interrupt Sources

Internal Interrupt Source	Trigger Condition	Interrupt Signal
GPIO_COMP*n*_ALL_INT (*n* = 0 ~ 1)	Any COMP_OUT signal transition occurs	GPIO_PAD_COMP_INT
GPIO_COMP*n*_NEG_INT (*n* = 0 ~ 1)	The COMP_OUT signal changes from high level to low level	GPIO_PAD_COMP_INT
GPIO_COMP*n*_POS_INT (*n* = 0 ~ 1)	The COMP_OUT signal changes from low level to high level	GPIO_PAD_COMP_INT

Note:
For definitions of *interrupt*, *interrupt signal*, *interrupt source*, and their correlations, please refer to Chapter 13 *Interrupt Matrix* > Section 13.2 *Interrupt Terminology in ESP32-P4*.

Each interrupt source can be configured by a common set of registers that are described in Section *Interrupt Configuration Registers*. The specific registers can be found in Section 10.19.1 *HP GPIO Matrix Register Summary*.

66.7 Programming Procedures

The programming procedure for the analog voltage comparator is as follows:

Note:
For more information about the register fields mentioned in this section, refer to Chapter 21 System Registers (SYSREG).

1. Enable the comparator by setting LPSYSTEM_XPD_COMP*n* (*n* = 0 ~ 1) to 1.
2. Disable normal digital pad functions (including IE, OE, WPU, and WPD) for PAD1, and if using the external reference voltage, also for PAD0. For detailed configuration, see Chapter 10 GPIO Matrix and IO MUX.
3. Configure the comparison mode by setting LPSYSTEM_MODE_COMP*n* (*n* = 0 ~ 1):
   - 0: Configures to compare the main voltage with the internal reference voltage;
   - 1: Configures to compare the main voltage with the external reference voltage.
4. Configure the internal reference voltage through LPSYSTEM_DREF_COMP*n* (*n*n = 0 ~ 1), which offers a voltage range of 0 ~ (0.7 × VDD_IO_6) V with a step of 0.1 × VDD_IO_6 V.

Part IX

Appendix

This part contains the following information starting from the next page:

- *Related Documentation and Resources*
- *Glossary*
- *Programming Reserved Register Field*
- *Interrupt Configuration Registers*
- *Revision History*

Related Documentation and Resources

Related Documentation

- *ESP32-P4 Series Datasheet*- Specifications of the ESP32-P4 hardware.
- *ESP32-P4 Hardware Design Guidelines* – Guidelines on how to integrate the ESP32-P4 into your hardware product.
- *ESP32-P4 Series SoC Errata* – Descriptions of known errors in ESP32-P4 series of SoCs.
- *Technical Reference Manual LaTeX sources* – Public repository containing the LaTeX sources for Espressif chip TRMs.
  https://github.com/espressif/esp-technical-reference-manual-latex
- *Certificates*
  https://documentation.espressif.com/en/documentList?t=Certificate&eol=false
- *ESP32-P4 Product/Process Change Notifications (PCN)*
  https://documentation.espressif.com/en/documentList?t=PCN&s=ESP32-P4
- *ESP32-P4 Advisories* – Information on security, bugs, compatibility, component reliability.
  https://documentation.espressif.com/en/documentList?t=Advisory&s=ESP32-P4
- *Documentation Updates and Update Notification Subscription*
  https://documentation.espressif.com/en/documentList?eol=false

Developer Zone

- ESP-IDF Programming Guide for ESP32-P4 – Extensive documentation for the ESP-IDF development framework.
- *ESP-IDF* and other development frameworks on GitHub.
  https://github.com/espressif
- *ESP32 BBS Forum* – Engineer-to-Engineer (E2E) Community for Espressif products where you can post questions, share knowledge, explore ideas, and help solve problems with fellow engineers.
  https://esp32.com/
- *ESP-FAQ* – A summary document of frequently asked questions released by Espressif.
  https://espressif.com/projects/esp-faq/en/latest/index.html
- *The ESP Journal* – Best Practices, Articles, and Notes from Espressif folks.
  https://blog.espressif.com/
- See the tabs *SDKs and Demos*, *Apps*, *Tools*, *AT Firmware*.
  https://espressif.com/en/support/download/sdks-demos

Products

- *ESP32-P4 Series SoCs* – Browse through all ESP32-P4 SoCs.
  https://espressif.com/en/products/socs?id=ESP32-P4
- *ESP32-P4 Series DevKits* – Browse through all ESP32-P4-based devkits.
  https://espressif.com/en/products/devkits?id=ESP32-P4
- *ESP Product Selector* – Find an Espressif hardware product suitable for your needs by comparing or applying filters.
  https://products.espressif.com/#/product-selector?language=en

Contact Us

- See the tabs *Sales Questions*, *Technical Enquiries*, *Circuit Schematic & PCB Design Review*, *Get Samples* (Online stores), *Become Our Supplier*, *Comments & Suggestions*.
  https://espressif.com/en/contact-us/sales-questions

Glossary

Abbreviations for Peripherals

2D-DMA	2D-DMA Controller
ADC	ADC Controller
AES	AES (Advanced Encryption Standard) Accelerator
DMA	DMA (Direct Memory Access) Controller
ECC	ECC (Elliptic Curve Cryptography) Accelerator
ECDSA_DS	ECDSA Digital Signature Peripheral
eFuse	eFuse Controller
EMAC	Ethernet Media Access Controller
ETM	Event Task Matrix
HMAC	HMAC (Hash-based Message Authentication Code) Accelerator
I2C	I2C (Inter-Integrated Circuit) Controller
I2S	I2S (Inter-IC Sound) Controller
I3C	I3C (Improved Inter-Integrated Circuit) Controller
ISP	Image Signal Processor
LCD_CAM	LCD and Camera Controller
LEDC	LED PWM (Pulse Width Modulation) Controller
MCPWM	Motor Control PWM (Pulse Width Modulation)
PARLIO	Parallel IO Controller
PCNT	Pulse Count Controller
PIE	Processor Instruction Extensions
PMS	Permission Control
PPA	Pixel-Processing Accelerator
RMT	Remote Control Peripheral
RNG	Random Number Generator
RSA	RSA (Rivest Shamir Adleman) Accelerator
RSA_DS	RSA Digital Signature Peripheral
SDHOST	SD/MMC Host Controller
SHA	SHA (Secure Hash Algorithm) Accelerator
SPI	SPI (Serial Peripheral Interface) Controller
TIMG	Timer Group
TOUCH	Touch Sensor
TRACE	RISC-V Trace Encoder
TSENS	Temperature Sensor
TWAI	Two-Wire Automotive Interface
UART	UART (Universal Asynchronous Receiver-Transmitter) Controller
ULP Coprocessor	Ultra-low-power Coprocessor
USB OTG	USB On-The-Go
USB_SERIAL_JTAG	USB Serial/JTAG Controller
VAD	Voice Activity Detection
VDMA	VDMA Controller
WDT	Watchdog Timers

XTS_AES External Memory Encryption and Decryption

Abbreviations Related to Registers

REG **Register**.

SYSREG **System registers** are a group of registers that control system reset, memory, clocks, software interrupts, power management, clock gating, etc.

ISO **Isolation**. If a peripheral or other chip component is powered down, the pins, if any, to which its output signals are routed will go into a floating state. ISO registers isolate such pins and keep them at a certain determined value, so that the other non-powered-down peripherals/devices attached to these pins are not affected.

NMI **Non-maskable interrupt** is a hardware interrupt that cannot be disabled or ignored by the CPU instructions. Such interrupts exist to signal the occurrence of a critical error.

W1TS Abbreviation added to names of registers/fields to indicate that such register/-field should be used to set a field in a corresponding register with a similar name. For example, the register `GPIO_ENABLE_W1TS_REG` should be used to set the corresponding fields in the register `GPIO_ENABLE_REG`.

W1TC Same as *W1TS*, but used to clear a field in a corresponding register.

Access Types for Registers

Sections *Register Summary* and *Register Description* in TRM chapters specify access types for registers and their fields.

Most frequently used access types and their combinations are as follows:

- RO
- WO
- WT
- R/W
- R/W1
- WL
- R/W/SC
- R/W/SS
- R/W/SS/SC
- R/WC/SS
- R/WC/SC
- R/WC/SS/SC
- R/WS/SC
- R/WS/SS
- R/WS/SS/SC
- R/SS/WTC
- R/SC/WTC
- R/SS/SC/WTC
- RF/WF
- R/SS/RC
- varies

Descriptions of all access types are provided below.

R **Read.** User application can read from this register/field; usually combined with other access types.

RO **Read only.** User application can only read from this register/field.

HRO **Hardware Read Only.** Only hardware can read from this register/field; used for storing default settings for variable parameters.

W **Write.** User application can write to this register/field; usually combined with other access types.

WO **Write only.** User application can only write to this register/field.

W1 **Write Once.** User application can write to this register/field only once; only allowed to write 1; writing 0 is invalid.

SS **Self set.** On a specified event, hardware automatically writes 1 to this register/field; used with 1-bit fields.

SC **Self clear.** On a specified event, hardware automatically writes 0 to this register/field; used with 1-bit and multi-bit fields.

SM **Self modify.** On a specified event, hardware automatically writes a specified value to this register/field; used with multi-bit fields.

SU **Self update.** On a specified event, hardware automatically updates this register/field; used with multi-bit fields.

RS **Read to set.** If user application reads from this register/field, hardware automatically writes 1 to it.

RC **Read to clear.** If user application reads from this register/field, hardware automatically writes 0 to it.

RF **Read from FIFO.** If user application writes new data to FIFO, the register/field automatically reads it.

WF **Write to FIFO.** If user application writes new data to this register/field, it automatically passes the data to FIFO via APB bus.

WS **Write any value to set.** If user application writes to this register/field, hardware automatically sets this register/field.

W1S **Write 1 to set.** If user application writes 1 to this register/field, hardware automatically sets this register/field.

W0S **Write 0 to set.** If user application writes 0 to this register/field, hardware automatically sets this register/field.

WC **Write any value to clear.** If user application writes to this register/field, hardware automatically clears this register/field.

W1C **Write 1 to clear.** If user application writes 1 to this register/field, hardware automatically clears this register/field.

W0C **Write 0 to clear.** If user application writes 0 to this register/field, hardware automatically clears this register/field.

WT **Write 1 to trigger an event.** If user application writes 1 to this field, this action triggers an event (pulse in the APB bus) or clears a corresponding WTC field (see WTC).

WTC **Write to clear.** Hardware automatically clears this field if user application writes 1 to the corresponding WT field (see WT).

W1T **Write 1 to toggle.** If user application writes 1 to this field, hardware automatically inverts the corresponding field; otherwise - no effect.

W0T **Write 0 to toggle.** If user application writes 0 to this field, hardware automatically inverts the corresponding field; otherwise - no effect.

WL **Write if a lock is deactivated.** If the lock is deactivated, user application can write to this register/field.

varies **The access type varies.** Different fields of this register might have different access types.

Programming Reserved Register Field

Introduction

A field in a register is reserved if the field is not open to users, or produces unpredictable results if configured to values other than defaults.

Programming Reserved Register Field

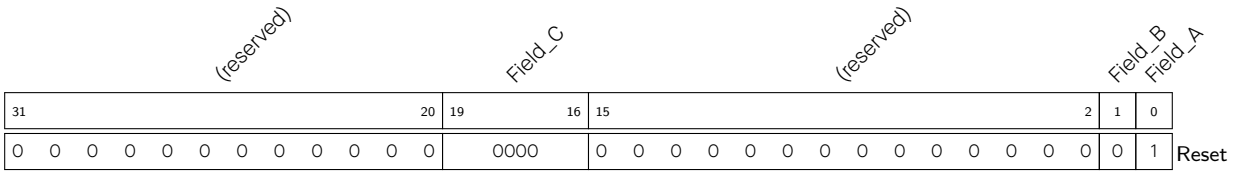
The reserved fields should not be modified. It is not possible to write only part of a register since registers must always be written as a whole. As a result, to write an entire register that contains reserved fields, you can choose one of the following two options:

1. Read the value of the register, modify only the fields you want to configure and then write back the value so that reserved fields are untouched.

   OR

2. Modify only the fields you want to configure and write back the default value of the reserved fields. The default value of a field is provided in the "Reset" line of a register diagram. For example, the default value of Field_A in Register X is 1.

Register 66.1. Register X (Address)



Suppose you want to set Field_A, Field_B, and Field_C of Register X to 0x0, 0x1, and 0x2, you can:

- Use option 1 and fill in the reserved fields with the value you have just read. Suppose the register reads as 0x0000_0003. Then, you can modify the fields you want to configure, thus writing 0x0002_0002 to the register.
- Use option 2 and fill in the reserved fields with their defaults, thus writing 0x0002_0002 to the register.

Interrupt Configuration Registers

Generally, the peripherals' internal interrupt sources can be configured by the following common set of registers:

- **RAW** (Raw Interrupt Status) register: This register indicates the raw interrupt status. Each bit in the register represents a specific internal interrupt source. When an interrupt source triggers, its RAW bit is set to 1.
- **ENA** (Enable) register: This register is used to enable or disable the internal interrupt sources. Each bit in the ENA register corresponds to an internal interrupt source.

  By manipulating the ENA register, you can mask or unmask individual internal interrupt source as needed. When an internal interrupt source is masked (disabled), it will not generate an interrupt signal, but its value can still be read from the RAW register.
- **ST** (Status) register: This register reflects the status of enabled interrupt sources. Each bit in the ST register corresponds to a specific internal interrupt source. The ST bit being 1 means that both the corresponding RAW bit and ENA bit are 1, indicating that the interrupt source is triggered and not masked. The other combinations of the RAW bit and ENA bit will result in the ST bit being 0.

  The configuration of ENA/RAW/ST registers is shown in Table 66.7-4.
- **CLR** (Clear) register: The CLR register is responsible for clearing the internal interrupt sources. Writing 1 to the corresponding bit in the CLR register clears the interrupt source.

Table 66.7-4. Configuration of ENA/RAW/ST Registers

ENA Bit Value	RAW Bit Value	ST Bit Value
0	Ignored	0
1	0	0
	1	1

Revision History

Date	Version	Release notes
2026-08-20	v0.7	Added the following chapter: • Chapter 4 *Processor Instruction Extensions (PIE)* Updated the following chapters: • Chapter 21 *System Registers (SYSREG)*: Updated throughout for the latest chip revision • Chapter 41 *H264 Encoder*: Updated throughout for the latest chip revision
2026-07-02	v0.6	Added the following chapters: • Chapter 1 *Bus Architecture* • Chapter 23 *AXI Performance Monitor* • Chapter 43 *MIPI DSI* Updated the following chapters: • Chapter 6 *GDMA Controller (GDMA-AHB, GDMA-AXI)*: Updated the descriptions of GDMA-AHB memory-to-memory data transfer • Chapter 38 *Image Signal Processor (ISP)*: Updated the functional descriptions of the lens shading correction, gamma correction, and automatic white balance statistics modules • Chapter 49 *I2S Controller (I2S)*: Updated the maximum frequency of the APLL clock source
2025-03-11	v0.5	Preliminary release



Disclaimer and Copyright Notice

Information in this document, including URL references, is subject to change without notice.

ALL THIRD PARTY'S INFORMATION IN THIS DOCUMENT IS PROVIDED AS IS WITH NO WARRANTIES TO ITS AUTHENTICITY AND ACCURACY.

NO WARRANTY IS PROVIDED TO THIS DOCUMENT FOR ITS MERCHANTABILITY, NON-INFRINGEMENT, FITNESS FOR ANY PARTICULAR PURPOSE, NOR DOES ANY WARRANTY OTHERWISE ARISING OUT OF ANY PROPOSAL, SPECIFICATION OR SAMPLE.

All liability, including liability for infringement of any proprietary rights, relating to use of information in this document is disclaimed. No licenses express or implied, by estoppel or otherwise, to any intellectual property rights are granted herein.

The Wi-Fi Alliance Member logo is a trademark of the Wi-Fi Alliance. The Bluetooth logo is a registered trademark of Bluetooth SIG.

All trade names, trademarks and registered trademarks mentioned in this document are property of their respective owners, and are hereby acknowledged.

Copyright © 2026 Espressif Systems (Shanghai) Co., Ltd. All rights reserved.

www.espressif.com