# ESP32-C61

## Technical Reference Manual Version 1.0

ESPRESSIF

www.espressif.com

## About This Document

The **ESP32-C61 Technical Reference Manual** is targeted at developers working on low level software projects that use the ESP32-C61 SoC. It describes the hardware modules listed below for the ESP32-C61 SoC and other products in ESP32-C61 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

| No. | Chapter | Progress |
|---|---|---|
| **Part I. Microprocessor and Master** | | |
| 1 | Bus Architecture | Published |
| 2 | ESP-RISC-V CPU | Published |
| 3 | RISC-V Trace Encoder (TRACE) | Published |
| 4 | GDMA Controller (GDMA) | Published |
| **Part II. Memory Organization** | | |
| 5 | System and Memory | Published |
| 6 | eFuse Controller (EFUSE) | Published |
| **Part III. System Component** | | |
| 7 | GPIO Matrix and IO MUX | Published |
| 8 | Reset and Clock | Published |
| 9 | Chip Boot Control | Published |
| 10 | Interrupt Matrix | Published |
| 11 | Event Task Matrix (ETM) | Published |
| 12 | Low-Power Management | Published |
| 13 | System Timer | Published |
| 14 | Timer Group (TIMG) | Published |
| 15 | Watchdog Timers (WDT) | Published |
| 16 | RTC Timer | Published |
| 17 | Permission Control (PMS) | Published |
| 18 | System Registers | Published |
| 19 | Debug Assistant | Published |
| 20 | Power Supply Detector | Published |
| **Part IV. Cryptography/Security Component** | | |
| 21 | ECC Accelerator (ECC) | Published |
| 22 | SHA Accelerator (SHA) | Published |
| 23 | ECDSA Digital Signature Peripheral (ECDSA_DS) | Published |
| 24 | External Memory Encryption and Decryption (XTS_AES) | Published |
| 25 | Random Number Generator (RNG) | Published |
| **Part V. Connectivity Interface** | | |
| 26 | UART Controller (UART) | Published |
| 27 | SPI Controller (SPI) | Published |
| 28 | I2C Controller (I2C) | Published |
| 29 | I2S Controller (I2S) | Published |
| 30 | USB Serial/JTAG Controller | Published |
| 31 | SDIO Slave Controller (SDIO) | Published |
| 32 | LED PWM Controller (LEDC) | Published |
| **Part VI. Analog Signal Processing** | | |
| 33 | Temperature Sensor | Published |
| 34 | ADC Controller | Published |
| 35 | 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-c61_technical_reference_manual_en.pdf

# Contents

I Microprocessor and Master 34

1 Bus Architecture 35

2 ESP-RISC-V CPU 37
2.1 Overview 37
2.2 Features 37
2.3 Terminology 38
2.4 Address Map 39
2.5 Configuration and Status Registers (CSRs) 39
2.5.1 Register Summary 40
2.5.2 Register Description 43
2.6 RISC-V Standard ISA Extensions Support 74
2.6.1 Multiplication/Division (M) Extension 74
2.6.1.1 Overview 74
2.6.1.2 Functional Description 74
2.6.2 Compressed (C) Extension 74
2.6.2.1 Overview 74
2.6.2.2 Functional Description 74
2.6.3 Code Size Reduction (Zc) Extension 75
2.6.3.1 Overview 75
2.6.3.2 Functional Description 75
2.6.3.3 Zc Instructions 75
2.6.3.4 Limitations 76
2.6.4 Bit Manipulation (Zb) Extension 77
2.6.4.1 Overview 77
2.6.4.2 Functional Description 77
2.6.5 Atomic (A) Extension 78
2.6.5.1 Overview 78
2.6.5.2 Functional Description 78
2.6.5.3 Load-Reserve (LR.W) Instruction 78
2.6.5.4 Store-Conditional (SC.W) Instruction 78
2.6.5.5 AMO Instructions 79
2.7 Memory-Mapped Registers 80
2.7.1 Overview 80
2.7.2 Features 80
2.7.3 Functional Description 80
2.8 Interrupt Controller 81
2.8.1 Overview 81
2.8.2 CLIC 81
2.8.2.1 Overview 81
2.8.2.2 CLIC CSRs 81

2.8.2.3 Interrupt Mapping 81
2.8.2.4 CLIC Parameters 82
2.8.2.5 Interrupt Level and Priority Encoding 82
2.8.2.6 CLIC Memory-Mapped Register Summary 83
2.8.2.7 CLIC Memory-Mapped Register Description 84
2.8.3 Core-Local Interrupts (CLINT) 87
2.8.3.1 Overview 87
2.8.3.2 Features 87
2.8.3.3 Software Interrupt 87
2.8.3.4 Timer Counter and Interrupt 87
2.8.3.5 Register Summary 87
2.8.3.6 Register Description 88
2.9 Memory Protection Unit 92
2.9.1 Standard Physical Memory Protection 92
2.9.1.1 Overview 92
2.9.1.2 Features 92
2.9.1.3 Functional Description 92
2.9.1.4 Register Summary 93
2.9.1.5 Register Description 93
2.9.2 Custom Physical Memory Attribute (PMA) Checker 96
2.9.2.1 Overview 96
2.9.2.2 Features 96
2.9.2.3 Functional Description 96
2.9.2.4 Register Summary 98
2.9.2.5 Register Description 99
2.10 Debug and Trace Support 101
2.10.1 Debug 101
2.10.1.1 Overview 101
2.10.1.2 Features 103
2.10.1.3 Functional Description 103
2.10.1.4 JTAG Control 103
2.10.1.5 Register Summary 104
2.10.1.6 Register Description 104
2.10.2 Hardware Trigger 108
2.10.2.1 Overview 108
2.10.2.2 Features 108
2.10.2.3 Functional Description 108
2.10.2.4 Trigger Execution Flow 109
2.10.2.5 Register Summary 109
2.10.2.6 Register Description 110
2.10.3 Trace 115
2.10.3.1 Overview 115
2.10.3.2 Features 115
2.10.3.3 Functional Description 115
2.11 Performance 116
2.11.1 Branch Prediction 116

2.11.1.1 Overview 116
2.11.1.2 Features 116
2.11.1.3 Functional Description 116
2.11.2 RAS 117
2.11.2.1 Overview 117
2.11.2.2 Features 117
2.11.2.3 Functional Description 118
2.11.3 Control Status Register for Performance Configuration 118
2.12 Custom Features 119
2.12.1 Bus Error Response 119
2.12.1.1 Overview 119
2.12.1.2 Functional Description 119
2.12.1.3 Control Status Registers 120
2.12.2 Dedicated IO 121
2.12.2.1 Overview 121
2.12.2.2 Features 121
2.12.2.3 Functional Description 121
2.12.2.4 Register Summary 122
2.12.2.5 Register Description 122
2.12.3 RunStall Support 124
2.12.3.1 Overview 124
2.12.3.2 Functional Description 124
2.12.3.3 Register Summary 124
2.12.4 Debug Assist Information 124
2.12.5 Core Lock-up 124
2.12.5.1 Overview 125
2.12.5.2 Functional Description 125
2.12.5.3 Register Summary 125
2.12.5.4 Register Description 125

**3 RISC-V Trace Encoder (TRACE) 126**
3.1 Terminology 126
3.2 Introduction 127
3.3 Features 128
3.4 Architectural Overview 129
3.5 Functional Description 129
3.5.1 Synchronization 129
3.5.2 Address Mode 130
3.5.3 Optional Sideband Signals 130
3.5.4 Filtering 131
3.5.5 Anchor Tag 131
3.5.6 Memory Writing Mode 132
3.5.7 Automatic Restart 132
3.6 Encoder Output Packets 132
3.6.1 Header 132
3.6.2 Index 133

3.6.3 Payload 133
3.6.3.1 Format 3 Packets 133
3.6.3.2 Format 2 Packets 135
3.6.3.3 Format 1 Packets 136
3.7 Interrupt 137
3.8 Programming Procedures 138
3.8.1 Encoder Option Configuration 138
3.8.2 Filter Configuration 139
3.8.3 Enable Encoder 140
3.8.4 Disable Encoder 141
3.8.5 Notify 141
3.8.6 Decode Data Packets 142
3.8.7 AHB Configuration 142
3.8.8 Software Retention 143
3.9 Register Summary 144
3.10 Registers 145

# 4 GDMA Controller (GDMA) 156
4.1 Overview 156
4.2 Features 156
4.3 Architecture 157
4.4 Functional Description 158
4.4.1 Linked List 158
4.4.2 Peripheral-to-Memory and Memory-to-Peripheral Data Transfer 159
4.4.3 Memory-to-Memory Data Transfer 160
4.4.4 Enabling GDMA 160
4.4.5 Linked List Reading Process 161
4.4.6 EOF 161
4.4.7 Accessing Memory 162
4.4.8 Arbitration 162
4.5 Event Task Matrix Feature 163
4.6 Interrupts 164
4.7 Programming Procedures 165
4.7.1 Programming Procedures for GDMA's Transmit Channel 165
4.7.2 Programming Procedures for GDMA's Receive Channel 165
4.7.3 Programming Procedures for Memory-to-Memory Transfer 165
4.7.4 Programming Procedures for Channel Priority and Weight 166
4.8 Register Summary 167
4.9 Registers 172

# II Memory Organization 200

# 5 System and Memory 201
5.1 Overview 201
5.2 Features 201
5.3 Functional Description 201

5.3.1 Address Mapping 202
5.3.2 Internal Memory 204
5.3.2.1 ROM 204
5.3.2.2 HP SRAM 204
5.3.3 External Memory 204
5.3.3.1 External Memory Address Mapping 204
5.3.3.2 Cache 205
5.3.3.3 Cache Operations 205
5.3.4 GDMA Address Space 206
5.3.5 Modules/Peripherals Address Mapping 207

6 eFuse Controller (EFUSE) 210
6.1 Overview 210
6.2 Feature List 210
6.3 Architectural Overview 210
6.3.1 EFUSE_WR_DIS 217
6.3.2 EFUSE_RD_DIS 217
6.3.3 Data Storage 217
6.4 Functional Description 218
6.4.1 Programming of Parameters 219
6.4.2 Reading of Parameters 220
6.4.3 eFuse VDDQ Timing 222
6.4.4 Parameters Used by Hardware Modules 222
6.5 Interrupts 222
6.6 Register Summary 223
6.7 Registers 227

III System Component 254

7 GPIO Matrix and IO MUX 255
7.1 Overview 255
7.2 Features 255
7.2.1 HP GPIO Matrix and HP IO MUX 255
7.2.2 LP GPIO Matrix and LP IO MUX 256
7.3 Architectural Overview 256
7.4 Peripheral Input via GPIO Matrix 258
7.4.1 Overview 258
7.4.2 Signal Synchronization 258
7.4.3 GPIO Filter 259
7.4.4 Simple GPIO Input 259
7.4.5 GPIO Wakeup 260
7.4.5.1 HP GPIO Wakeup 260
7.4.5.2 LP GPIO Wakeup 260
7.4.6 Programming Procedure 261
7.4.6.1 HP GPIO Matrix 261
7.4.6.2 LP GPIO Matrix 262

7.5 Peripheral Output via GPIO Matrix 262
7.5.1 Overview 262
7.5.2 Simple GPIO Output 262
7.5.3 Programming Procedure 263
7.5.3.1 HP GPIO Matrix 263
7.5.3.2 LP GPIO Matrix 264
7.6 Direct Input and Output via IO MUX 264
7.6.1 Overview 264
7.6.2 Functional Description 264
7.6.2.1 HP IO MUX 264
7.6.2.2 LP IO MUX 265
7.7 Analog Functions 265
7.7.1 Overview 265
7.7.2 Analog Functions 265
7.8 Pin Functions in Light-sleep 265
7.9 Pin Hold Feature 266
7.10 Hysteresis Characteristics 266
7.11 Power Supplies and Management of GPIO Pins 267
7.11.1 Power Supplies of GPIO Pins 268
7.11.2 Power Supply Management 268
7.12 HP Peripheral Signal List 268
7.13 HP IO MUX Function List 271
7.14 LP IO MUX Function List 273
7.15 GPIO Pin Analog Function List 273
7.16 Event Task Matrix Function 274
7.17 Interrupts 275
7.18 Register Summary 276
7.18.1 HP GPIO Matrix Register Summary 276
7.18.2 HP IO MUX Register Summary 279
7.18.3 GPIO EXT Register Summary 279
7.18.4 LP GPIO Matrix Register Summary 280
7.18.5 LP IO MUX Register Summary 281
7.19 Registers 281
7.19.1 HP GPIO Matrix Registers 282
7.19.2 HP IO MUX Registers 297
7.19.3 GPIO EXT Registers 300
7.19.4 LP GPIO Matrix Registers 313
7.19.5 LP IO MUX Registers 321

8 Reset and Clock 325
8.1 Reset 325
8.1.1 Overview 325
8.1.2 Architectural Overview 325
8.1.3 Features 325
8.1.4 Functional Description 326
8.1.5 Peripheral Reset 327

8.2 Clock 327
8.2.1 Overview 327
8.2.2 Architectural Overview 328
8.2.3 Features 328
8.2.4 Functional Description 329
8.2.4.1 HP System Clock 329
8.2.4.2 LP System Clock 330
8.2.4.3 Peripheral Clocks 330
8.2.4.4 PMU Control of High-Performance System Clock Gating 333
8.3 Programming Procedures 335
8.3.1 HP System Clock Configuration 335
8.3.2 LP System Clock Configuration 335
8.3.3 Peripheral Clock Reset and Configuration 335
8.4 Register Summary 338
8.4.1 PCR Register Summary 338
8.4.2 LP System Clock Register Summary 340
8.5 Registers 341
8.5.1 PCR Registers 341
8.5.2 LP System Clock Registers 380

9 Chip Boot Control 388
9.1 Overview 388
9.2 Functional Description 388
9.2.1 Default Configuration 389
9.2.2 Boot Mode Control 389
9.2.3 SDIO Sampling and Driving Clock Edge Control 391
9.2.4 ROM Messages Printing Control 391
9.2.5 JTAG Signal Source Control 392

10 Interrupt Matrix 393
10.1 Overview 393
10.2 Interrupt Terminology in ESP32-C61 393
10.2.1 Interrupt 393
10.2.2 Interrupt Signal/Interrupt Source 393
10.2.3 Interrupt Flow in ESP32-C61 394
10.3 Features 394
10.4 Architecture 394
10.5 Functional Description 395
10.5.1 Peripheral Interrupt Sources 395
10.5.2 HP CPU Interrupts 398
10.5.3 Assign Peripheral Interrupt Source to HP CPU Peripheral Interrupt 398
10.5.3.1 Assign One Peripheral Interrupt Source to HP CPU Peripheral Interrupt 398
10.5.3.2 Assign Multiple Peripheral Interrupt Sources to HP CPU Peripheral Interrupt 398
10.5.3.3 Unassign SOURCE 398
10.5.4 Delegated Interrupts 399
10.5.5 Query Current Interrupt Status of SOURCE 399

10.6 Register Summary 400
10.6.1 Interrupt Matrix Register Summary 400
10.6.2 Software Interrupt Register Summary 402
10.7 Registers 403
10.7.1 Interrupt Matrix Registers 403
10.7.2 Software Interrupt Registers 409

11 Event Task Matrix (ETM) 410
11.1 Overview 410
11.2 Features 410
11.3 Functional Description 410
11.3.1 Architecture 410
11.3.2 Events 411
11.3.3 Tasks 413
11.3.4 Event and Task Status 416
11.3.5 Timing Considerations 417
11.3.6 Channel Control 418
11.4 Register Summary 419
11.5 Registers 423

12 Low-Power Management 496
12.1 Overview 496
12.2 Terminology 496
12.3 Feature List 496
12.4 Functional Description 497
12.4.1 Power Scheme 497
12.4.1.1 Regulators 498
12.4.1.2 Digital Power Domains 499
12.4.1.3 Analog Power Domains 499
12.4.2 PMU 499
12.4.2.1 PMU Main State Machine 501
12.4.2.2 Sleep/Wake-up Controller 502
12.4.2.3 Analog Power Controller 503
12.4.2.4 Digital Power Controller 504
12.4.2.5 Clock Controller 506
12.4.2.6 Backup Controller 507
12.4.2.7 System Controller 508
12.4.3 RTC Timer 508
12.5 Power Modes 509
12.6 Event Task Matrix Feature 509
12.7 Interrupts 510
12.8 Register Summary 512
12.8.1 PMU Register Summary 512
12.8.2 PAU Register Summary 516
12.9 Registers 518
12.9.1 PMU Registers 518

12.9.2 PAU Registers 591

13 System Timer 598
13.1 Overview 598
13.2 Features 598
13.3 System Timer Structure 599
13.4 Clock Source Selection 599
13.5 Functional Description 600
13.5.1 Counter 600
13.5.2 Comparator and Alarm 601
13.5.3 Event Task Matrix 602
13.5.4 Synchronization Operation 602
13.6 Interrupts 603
13.7 Programming Procedure 603
13.7.1 Read Current Count Value 603
13.7.2 Configure a One-Time Alarm in Target Mode 604
13.7.3 Configure Periodic Alarms in Period Mode 604
13.7.4 Update After Light-sleep 604
13.8 Register Summary 606
13.9 Registers 608

14 Timer Group (TIMG) 623
14.1 Overview 623
14.2 Feature List 623
14.3 Architectural Overview 624
14.4 Functional Description 624
14.4.1 16-bit Prescaler and Clock Selection 624
14.4.2 54-bit Time-base Counter 625
14.4.3 Alarm Generation 625
14.4.4 Timer Reload 626
14.4.5 Frequency Calculation of Slow Clock for Timer Group 0 627
14.5 Event Task Matrix Feature 627
14.6 Interrupts 628
14.7 Programming Procedures 629
14.7.1 Timer as a Simple Clock 629
14.7.2 Timer as One-shot Alarm 629
14.7.3 Timer as Periodic Alarm by APB 629
14.7.4 Timer as Periodic Alarm by ETM 630
14.7.5 Frequency Calculation of Slow Clock 631
14.8 Register Summary 632
14.9 Registers 633

15 Watchdog Timers (WDT) 646
15.1 Overview 646
15.2 Digital Watchdog Timers 647
15.2.1 Features 647

15.2.2 Functional Description 648
15.2.2.1 Clock Source and 32-Bit Counter 648
15.2.2.2 Stages and Timeout Actions 649
15.2.2.3 Write Protection 650
15.2.2.4 Flash Boot Protection 650
15.3 Super Watchdog 650
15.3.1 Features 650
15.3.2 Super Watchdog Controller 651
15.3.2.1 Structure 651
15.3.2.2 Workflow 651
15.4 Interrupts 652
15.5 Register Summary 652
15.6 Registers 652

16 RTC Timer 660
16.1 Introduction 660
16.2 Feature List 660
16.3 Functional Description 660
16.4 Event Task Matrix Feature 662
16.5 Interrupts 662
16.6 Register Summary 663
16.7 Registers 664

17 Permission Control (PMS) 670
17.1 Overview 670
17.2 Introduction to APM 671
17.3 Features 671
17.4 TEE and REE Terminology 672
17.5 Functional Description 672
17.5.1 TEE Controller Functional Description 672
17.5.2 APM Controller Functional Description 673
17.5.2.1 Architecture 673
17.5.2.2 Address Ranges 674
17.5.2.3 Access Permissions of Address Ranges 674
17.6 Programming Procedure 675
17.7 Illegal Access and Interrupts 676
17.8 Register Summary 677
17.8.1 HP_APM_REG 677
17.8.2 LP_APM_REG 680
17.8.3 CPU_APM_REG 681
17.8.4 HP_TEE_REG 682
17.9 Registers 684
17.9.1 HP_APM_REG 684
17.9.2 LP_APM_REG 695
17.9.3 CPU_APM_REG 699
17.9.4 HP_TEE_REG 707

18 System Registers 709
18.1 Overview 709
18.2 Features 709
18.3 Function Description 709
18.3.1 External Memory Encryption/Decryption Configuration 709
18.3.2 Anti-DPA Attack Security Control 709
18.3.3 Bus Timeout Protection 710
18.3.3.1 CPU Peripheral Timeout Protection Register 710
18.3.3.2 HP Peripheral Timeout Protection Register 710
18.3.3.3 LP Peripheral Timeout Protection Register 711
18.4 Register Summary 712
18.5 Registers 713

19 Debug Assistant 720
19.1 Overview 720
19.2 Features 720
19.3 Functional Description 720
19.3.1 Region Read/Write Monitoring 720
19.3.2 SP Monitoring 721
19.3.3 PC Logging 721
19.3.4 CPU/DMA Bus Access Logging 721
19.4 Interrupts 721
19.5 Programming Procedures 722
19.5.1 Region Monitoring and SP Monitoring Configuration 722
19.5.2 PC Logging Configuration 723
19.5.3 CPU/DMA Bus Access Logging Configuration 724
19.5.3.1 Bus Access Logging 1 Configuration 724
19.5.3.2 Bus Access Logging 2 Configuration 727
19.6 Register Summary 730
19.6.1 Bus Logging Configuration Registers 730
19.6.2 SOther Registers 731
19.7 Registers 733
19.7.1 Bus Logging Configuration Registers 733
19.7.2 Other Registers 742

20 Power Supply Detector 757
20.1 Overview 757
20.2 Features 757
20.3 Functional Description 757
20.3.1 Architecture 757
20.3.2 Brown-out Detector 758
20.3.3 Voltage Glitch Detectors 759
20.4 Interrupts 760
20.5 Register Summary 762
20.6 Registers 763

# IV Cryptography/Security Component 770

## 21 ECC Accelerator (ECC) 771
21.1 Overview 771
21.2 Feature List 771
21.3 ECC Basics 771
21.3.1 Elliptic Curve and Points on the Curves 771
21.3.2 Affine Coordinates and Jacobian Coordinates 772
21.3.3 Memory Blocks 772
21.3.4 Data and Data Block 772
21.3.5 Writing Data 773
21.3.6 Reading Data 773
21.3.7 Standard Calculation and Jacobian Calculation 773
21.4 Function Description 773
21.4.1 Key Size 773
21.4.2 Working Modes 774
21.4.2.1 Affine Point Multiplication (Affine Point Multi) 774
21.4.2.2 Affine Point Verification (Affine Point Verif) 775
21.4.2.3 Affine Point Verification + Affine Point Multiplication (Affine Point Verif + Multi) 775
21.4.2.4 Jacobian Point Multiplication (Jacobian Point Multi) 775
21.4.2.5 Point Addition (Point Add) 775
21.4.2.6 Jacobian Point Verification (Jacobian Point Verif) 776
21.4.2.7 Affine Point Verification + Jacobian Point Multiplication (Affine Point Verif + Jacobian Point Multi) 776
21.4.2.8 Mod Addition (Mod Add) 776
21.4.2.9 Mod Subtraction (Mod Sub) 777
21.4.2.10 Mod Multiplication (Mod Multi) 777
21.4.2.11 Mod Division (Mod Div) 777
21.4.3 Enhancing Anti-Attack Performance 778
21.5 Clock and Reset 778
21.6 Interrupts 778
21.7 Programming Procedures 779
21.8 Register Summary 780
21.9 Registers 781

## 22 SHA Accelerator (SHA) 785
22.1 Introduction 785
22.2 Features 785
22.3 Working Modes 785
22.4 Function Description 786
22.4.1 Preprocessing 786
22.4.1.1 Padding the Message 786
22.4.1.2 Parsing the Message 786
22.4.1.3 Setting the Initial Hash Value 787
22.4.2 Hash Operation 787
22.4.2.1 Typical SHA Mode Process 787

22.4.2.2 DMA-SHA Mode Process 788
22.4.3 Message Digest 789
22.4.4 Interrupt 790
22.5 Register Summary 791
22.6 Registers 792

23 ECDSA Digital Signature Peripheral (ECDSA_DS) 796
23.1 Introduction 796
23.2 Features 796
23.3 ECDSA Basics 796
23.3.1 Domain Parameters 796
23.3.2 Key Generation 797
23.3.3 Signature Generation 797
23.3.4 Signature Verification 798
23.4 Functional Description 798
23.4.1 ECDSA_DS Working Modes 798
23.4.2 Data and Data Block 799
23.4.2.1 Writing Data 800
23.4.2.2 Reading Data 800
23.4.2.3 Padding the Message 800
23.4.2.4 Parsing the Message 801
23.4.3 Security Features 801
23.4.3.1 High Anti-Attack Performance 801
23.4.3.2 State-Dependent Register Access Control 801
23.4.3.3 Hardware Occupation 801
23.5 Programming Procedures 802
23.5.1 ECDSA_DS Process 802
23.5.1.1 IDLE Stage 802
23.5.1.2 PREP Stage 803
23.5.1.3 LOAD Stage 803
23.5.1.4 PROC Stage 804
23.5.1.5 GAIN Stage 804
23.5.1.6 POST Stage 804
23.5.1.7 ECDSA_DS SHA Interface 804
23.5.2 Clocks and Resets 805
23.5.3 Interrupts 805
23.6 Memory Blocks 807
23.7 Register Summary 808
23.8 Registers 809

24 External Memory Encryption and Decryption (XTS_AES) 816
24.1 Overview 816
24.2 Features 816
24.3 Module Structure 816
24.4 Functional Description 817
24.4.1 XTS Algorithm 818

24.4.2 Key 818
24.4.3 Target Memory Space 818
24.4.4 Data Writing 819
24.4.5 Manual Encryption Block 820
24.4.6 Auto Encryption Block 820
24.4.7 Auto Decryption Block 821
24.5 Software Process 821
24.6 Anti-DPA 822
24.6.1 Clock Anti-DPA Function 822
24.6.2 Pseudo-round Anti-DPA Function 823
24.7 Register Summary 825
24.8 Registers 826

25 Random Number Generator (RNG) 832
25.1 Introduction 832
25.2 Features 832
25.3 Functional Description 832
25.4 Programming Procedure 833
25.5 Register Summary 835
25.6 Registers 836

V Connectivity Interface 837

26 UART Controller (UART) 838
26.1 Overview 838
26.2 Features 838
26.3 UART Structure 839
26.4 Functional Description 840
26.4.1 Clock and Reset 840
26.4.2 UART FIFO 840
26.4.3 Baud Rate Generation and Detection 841
26.4.3.1 Baud Rate Generation 841
26.4.3.2 Baud Rate Detection 842
26.4.4 UART Data Frame 843
26.4.5 AT_CMD Character Structure 843
26.4.6 RS485 844
26.4.6.1 Driver Control 844
26.4.6.2 Turnaround Delay 845
26.4.6.3 Bus Snooping 845
26.4.7 IrDA 845
26.4.8 Wakeup 846
26.4.9 Flow Control 847
26.4.9.1 Hardware Flow Control 847
26.4.9.2 Software Flow Control 849
26.5 Interrupts 849
26.6 Programming Procedures 850

26.6.1 Register Type 851
26.6.2 Detailed Steps 852
26.6.2.1 Initializing UART*n* 852
26.6.2.2 Configuring UART*n* Communication 852
26.6.2.3 Enabling UART*n* 853
26.7 Register Summary 854
26.7.1 UART Register Summary 854
26.8 Registers 856
26.8.1 UART Registers 856

**27 SPI Controller (SPI) 879**
27.1 Overview 879
27.2 Glossary 879
27.3 Features 880
27.4 Architectural Overview 881
27.5 Functional Description 882
27.5.1 Data Modes 882
27.5.2 FSPI Bus Signals 882
27.5.3 Bit Read/Write Order Control 884
27.5.4 Unaligned Byte Transfer 886
27.5.5 Transfer Types 886
27.5.6 CPU-Controlled Data Transfer 886
27.5.6.1 CPU-Controlled Master Transfer 887
27.5.6.2 CPU-Controlled Slave Transfer 889
27.5.7 DMA-Controlled Data Transfer 890
27.5.7.1 DMA Configuration 890
27.5.7.2 GDMA TX/RX Buffer Length Control 891
27.5.8 Data Flow Control 891
27.5.8.1 GP-SPI2 Functional Blocks 892
27.5.8.2 Data Flow Control When GP-SPI2 Works as Master 893
27.5.8.3 Data Flow Control When GP-SPI2 Works as Slave 893
27.5.9 GP-SPI2 Works as Master 894
27.5.9.1 State Machine 894
27.5.9.2 Register Configuration for State and Bit Mode Control 897
27.5.9.3 Full-Duplex Communication (1-bit Mode Only) 901
27.5.9.4 Half-Duplex Communication (1/2/4-bit Mode) 902
27.5.9.5 DMA-Controlled Configurable Segmented Transfer 904
27.5.10 GP-SPI2 Works as Slave 907
27.5.10.1 Configurable Communication Formats 908
27.5.10.2 CMD Values Supported in Half-Duplex Communication 909
27.5.10.3 Slave Single Transfer and Slave Segmented Transfer 911
27.5.10.4 Configuration of Slave Single Transfer 912
27.5.10.5 Configuration of Slave Segmented Transfer in Half-Duplex 912
27.5.10.6 Configuration of Slave Segmented Transfer in Full-Duplex 913
27.6 CS Setup Time and Hold Time Control 913
27.7 GP-SPI2 Clock Control 914

27.7.1 Clock Phase and Polarity 915
27.7.2 Clock Control When GP-SPI2 Works as Master 917
27.7.3 Clock Control When GP-SPI2 Works as Slave 917
27.8 GP-SPI2 Timing Compensation 917
27.9 Interrupt 920
27.9.1 Interrupt Description 920
27.9.2 Interrupts Used in Master and in Slave 921
27.10 Register Summary 923
27.11 Register 925

28 I2C Controller (I2C) 965
28.1 Overview 965
28.2 Feature List 965
28.3 Architecture Overview 966
28.4 Functional Description 968
28.4.1 Clock Configuration 968
28.4.2 SCL and SDA Noise Filtering 969
28.4.3 SCL Clock Stretching 969
28.4.4 Generating SCL Pulses in Idle State 970
28.4.5 Synchronization 970
28.4.6 Open-Drain Output 971
28.4.7 Timing Parameters Configuration 972
28.4.8 Timeout Control 974
28.4.9 Command Configuration 974
28.4.10 TX/RX RAM Data Storage 975
28.4.11 Data Conversion 976
28.4.12 Addressing Mode 976
28.4.13 $R/\overline{W}$ Bit Check in 10-bit Addressing Mode 977
28.4.14 Start the I2C Controller 977
28.5 Interrupts 978
28.6 Programming Procedures 979
28.6.1 $I2C_{master}$ Writes to $I2C_{slave}$ with a 7-bit Address in One Command Sequence 979
28.6.1.1 Introduction 979
28.6.1.2 Configuration Example 980
28.6.2 $I2C_{master}$ Writes to $I2C_{slave}$ with a 10-bit Address in One Command Sequence 981
28.6.2.1 Introduction 982
28.6.2.2 Configuration Example 982
28.6.3 $I2C_{master}$ Writes to $I2C_{slave}$ with Two 7-bit Addresses in One Command Sequence 983
28.6.3.1 Introduction 983
28.6.3.2 Configuration Example 984
28.6.4 $I2C_{master}$ Writes to $I2C_{slave}$ with a 7-bit Address in Multiple Command Sequences 985
28.6.4.1 Introduction 985
28.6.4.2 Configuration Example 986
28.6.5 $I2C_{master}$ Reads $I2C_{slave}$ with a 7-bit Address in One Command Sequence 987
28.6.5.1 Introduction 987
28.6.5.2 Configuration Example 988

28.6.6 $I2C_{master}$ Reads $I2C_{slave}$ with a 10-bit Address in One Command Sequence 989
28.6.6.1 Introduction 989
28.6.6.2 Configuration Example 990
28.6.7 $I2C_{master}$ Reads $I2C_{slave}$ with Two 7-bit Addresses in One Command Sequence 991
28.6.7.1 Introduction 992
28.6.7.2 Configuration Example 992
28.6.8 $I2C_{master}$ Reads $I2C_{slave}$ with a 7-bit Address in Multiple Command Sequences 994
28.6.8.1 Introduction 994
28.6.8.2 Configuration Example 995
28.7 Register Summary 997
28.7.1 I2C Register Summary 997
28.8 Registers 999
28.8.1 I2C Registers 999

**29 I2S Controller (I2S) 1024**
29.1 Overview 1024
29.2 Terminology 1024
29.3 Features 1025
29.4 System Architecture 1026
29.5 Supported Audio Standards 1028
29.5.1 TDM Philips Standard 1028
29.5.2 TDM MSB Alignment Standard 1029
29.5.3 TDM PCM Standard 1029
29.5.4 PDM Standard 1030
29.6 I2S TX/RX Clock 1030
29.7 I2S Reset 1032
29.8 I2S Master/Slave Mode 1032
29.8.1 Master/Slave TX Mode 1033
29.8.2 Master/Slave RX Mode 1033
29.9 Transmitting Data 1034
29.9.1 Data Format Control 1034
29.9.1.1 Bit Width Control of Channel Valid Data 1034
29.9.1.2 Endian Control of Channel Valid Data 1035
29.9.1.3 A-law/$\mu$-law Compression and Decompression 1035
29.9.1.4 Bit Width Control of Channel TX Data 1036
29.9.1.5 Bit Order Control of Channel Data 1036
29.9.2 Channel Mode Control 1036
29.9.2.1 TDM TX Mode 1037
29.9.2.2 PDM TX Mode 1038
29.10 Receiving Data 1040
29.10.1 Channel Mode Control 1040
29.10.1.1 TDM RX Mode 1040
29.10.1.2 PDM RX Mode 1041
29.10.2 Data Format Control 1041
29.10.2.1 Bit Order Control of Channel Data 1041
29.10.2.2 Bit Width Control of Channel Storage (Valid) Data 1041

29.10.2.3 Bit Width Control of Channel RX Data 1042
29.10.2.4 Endian Control of Channel Storage Data 1042
29.10.2.5 A-law/$\mu$-law Compression and Decompression 1042
29.11 Event Task Matrix Feature 1043
29.12 I2S Interrupts 1043
29.13 Software Configuration Process 1044
29.13.1 Configure I2S as TX Mode 1044
29.13.2 Configure I2S as RX Mode 1045
29.14 Register Summary 1046
29.15 Registers 1047

30 USB Serial/JTAG Controller 1064
30.1 Overview 1064
30.2 Feature List 1064
30.3 Functional Description 1066
30.3.1 CDC-ACM USB Interface Functional Description 1066
30.3.2 CDC-ACM Firmware Interface Functional Description 1067
30.3.3 USB-to-JTAG Interface: JTAG Command Processor 1068
30.3.4 USB-to-JTAG Interface: CMD_REP Usage Example 1069
30.3.5 USB-to-JTAG Interface: Response Capture Unit 1070
30.3.6 USB-to-JTAG Interface: Control Transfer Requests 1070
30.4 Interrupts 1071
30.5 Programming Procedures 1072
30.6 Register Summary 1074
30.7 Registers 1076

31 SDIO Slave Controller (SDIO) 1102
31.1 Overview 1102
31.2 Features 1102
31.3 Architecture Overview 1103
31.4 Standards Compliance 1103
31.5 Functional Description 1103
31.5.1 Physical Bus 1103
31.5.2 Supported Commands 1104
31.5.3 I/O Function 0 Address Space 1104
31.5.4 I/O Function 1/2 Address Space Map 1107
31.5.4.1 Accessing SLC HOST Register Space 1107
31.5.4.2 Transferring Incremental-Address Packets 1108
31.5.4.3 Transferring Fixed-Address Packets 1108
31.5.5 DMA 1108
31.5.5.1 Linked List 1109
31.5.5.2 Write-Back of Linked List 1111
31.5.5.3 Data Padding and Discarding 1111
31.5.6 SDIO Bus Timing 1112
31.6 Interrupt 1113
31.6.1 Host Interrupt 1113

31.6.2 Slave Interrupt 1114
31.7 Packet Sending and Receiving Procedure 1114
31.7.1 Sending Packets to SDIO Host 1114
31.7.2 Receiving Packets from SDIO Host 1116
31.8 Register Summary 1119
31.8.1 HINF Register Summary 1119
31.8.2 SLC Register Summary 1119
31.8.3 SLC Host Register Summary 1120
31.9 Registers 1122
31.9.1 HINF Registers 1122
31.9.2 SLC Registers 1127
31.9.3 SLC Host Registers 1149

32 LED PWM Controller (LEDC) 1159
32.1 Overview 1159
32.2 Features 1159
32.3 Architectural Overview 1159
32.4 Functional Description 1160
32.4.1 Timers 1160
32.4.1.1 Clock Source 1161
32.4.1.2 Clock Divider Configuration 1161
32.4.1.3 20-Bit Counter 1162
32.4.2 PWM Generators 1163
32.4.3 Duty Cycle Fading 1164
32.4.3.1 Linear Duty Cycle Fading 1165
32.4.3.2 Gamma Curve Fading 1165
32.4.3.3 Suspend and Resume Duty Cycle Fading 1166
32.4.4 Event Task Matrix Feature 1167
32.5 Interrupts 1168
32.6 Programming Procedures 1168
32.7 Memory Blocks 1170
32.8 Register Summary 1172
32.9 Registers 1174

VI Analog Signal Processing 1186

33 Temperature Sensor 1187
33.1 Overview 1187
33.2 Features 1187
33.3 Architecture 1187
33.4 Functional Description 1188
33.4.1 Temperature Sensor Power Up 1188
33.4.2 Temperature Sensor Clock 1188
33.4.3 Automatic Temperature Monitoring Modes 1188
33.4.4 Temperature Measurement Range and Offset 1189
33.4.5 Data Conversion 1189

33.5 Programming Procedure 1189
33.6 Interrupts 1190
33.7 Event Task Matrix Feature 1190
33.8 Register Summary 1192
33.9 Registers 1193

34 ADC Controller 1197
34.1 Overview 1197
34.2 Terminology 1197
34.3 Feature List 1197
34.4 Architectural Overview 1198
34.5 Functional Description 1199
34.5.1 ADC Power Up 1199
34.5.2 ADC Channels 1199
34.5.3 ADC Clock 1199
34.5.4 One-Shot Sampling Mode 1200
34.5.5 Multi-Channel Sampling Mode 1200
34.5.6 ADC Conversion and Attenuation 1200
34.5.7 DIG ADC FSM 1201
34.5.8 Pattern Table 1202
34.5.9 ADC Filters 1203
34.5.10 Threshold Monitors 1204
34.5.11 GDMA Support 1204
34.6 Event Task Matrix Feature 1205
34.7 Interrupts 1205
34.8 Programming Procedure 1206
34.8.1 Configuring One-shot Sampling Mode 1206
34.8.2 Configuring Multi-Channel Sampling Mode 1206
34.9 Register Summary 1207
34.10 Registers 1208

35 Analog Voltage Comparator 1220
35.1 Introduction 1220
35.2 Feature List 1220
35.3 Architectural Overview 1220
35.4 Functional Description 1221
35.5 Event Task Matrix Feature 1222
35.6 Interrupts 1222
35.7 Programming Procedures 1223

VII Appendix 1224

Related Documentation and Resources 1225

Glossary 1226
Abbreviations for Peripherals 1226

Abbreviations Related to Registers 1226
Access Types for Registers 1227

Programming Reserved Register Field 1229
Introduction 1229
Programming Reserved Register Field 1229

Interrupt Configuration Registers 1230

Revision History 1231

# List of Tables


2.4-1 CPU Address Map 39
2.7-1 Address Decoding for Memory-Mapped Register Access 80
2.7-2 Internal Memory Address Map 80
2.8-1 CLIC Interrupt Mapping 82
2.8-2 CLIC Level and Priority Encoding with CLICINTCTLBITS=3 83
2.8-4 Core-Local Interrupt Sources 87
2.10-3 Virtual address in DPC upon Debug Mode Entry 106
2.10-4 NAPOT encoding for maddress 109

3.3-1 Trace Encoder Parameters 128
3.5-1 Optional sideband signals 130
3.6-1 Header Format 133
3.6-2 Index Format 133
3.6-3 Packet format 3 subformat 0 133
3.6-4 Packet format 3 subformat 1 134
3.6-5 Packet format 3 subformat 3 134
3.6-6 Packet format 2 135
3.6-7 Packet format 1 with address 136
3.6-8 Packet format 1 without address 137
3.7-1 TRACE's Internal Interrupt Sources 138
3.8-1 Debug module trigger support (the action field of the mcontrol register) 139
3.8-2 Trace Status 141

4.4-1 GDMA Selecting Peripherals via Register Configuration 160

5.3-1 Memory Address Mapping 203
5.3-2 Module/Peripheral Address Mapping 208

6.3-1 Parameters in eFuse BLOCK0 212
6.3-2 Secure Key Purpose Values 215
6.3-3 Parameters in BLOCK1 to BLOCK10 216
6.4-1 Registers Information 221
6.4-2 Configuration of Default VDDQ Timing Parameters 222

7.4-1 HP GPIO Wakeup Signal Trigger and Clear Conditions 260
7.4-2 LP GPIO Wakeup Signal Trigger and Clear Conditions 260
7.8-1 Bit Used to Control IO MUX Functions in Light-sleep Mode 265
7.12-1 Peripheral Signals via HP GPIO Matrix 268
7.13-1 HP IO MUX Pin Functions 272
7.14-1 LP IO MUX Pin Functions 273
7.15-1 GPIO Pin Analog Functions 274
7.17-1 HP IO MUX and HP GPIO Matrix's Internal Interrupt Sources 275
7.17-2 LP IO MUX and LP GPIO Matrix's Internal Interrupt Sources 276

8.1-1 Reset Source 326

8.2-1 CPU_CLK Clock Source 329
8.2-2 Frequency of CPU_CLK, AHB_CLK and HP_ROOT_CLK 329
8.2-3 Derived HP Clock Source 331
8.2-4 HP Clocks Used by Each Peripheral 332
8.2-5 Derived LP Clock Source 332
8.2-6 LP Clocks Used by Each Peripheral 332
8.2-7 Mapping between controlling the clock gating of read/write registers and related PMU registers 334
8.2-8 Mapping between controlling the functional clock gating of high-performance system peripherals and related PMU registers 335

9.2-1 Default Configuration of Strapping Pins 389
9.2-2 Boot Mode Control 389
9.2-3 SDIO Input Sampling Edge/Output Driving Edge Control 391
9.2-4 ROM Message Printing Control 392
9.2-5 JTAG Signal Source Control 392

10.5-1 CPU Peripheral Interrupt Source Mapping/Status Registers and Peripheral Interrupt Sources 396

11.3-1 Selectable Events for ETM Channel*n* 411
11.3-2 Mappable Tasks for ETM Channel*n* 414

12.4-1 Wake-up Sources 503
12.4-2 Trigger Conditions for the RTC Timer 508
12.5-1 Preset Power Modes 509

13.5-1 UNIT*n* Configuration Bits 600
13.5-2 Trigger Point 602
13.5-3 Synchronization Operation for Configuration Registers 602
13.6-1 System Timer's Internal Interrupt Sources 603

14.4-1 Alarm Generation When Up-Down Counter Increments 626
14.4-2 Alarm Generation When Up-Down Counter Decrements 626
14.6-1 TIMG*n*'s Internal Interrupt Sources 628

15.2-1 Timeout Actions 649

16.3-1 Configure RTC Timer to Log the Occurrence Time of Specific Events 661
16.3-2 Target Time Configuration 661
16.5-1 RTC Timer's Internal Interrupt Sources 662

17.1-1 Management Areas by PMP and APM 670
17.4-2 Comparison Between TEE and REE 672
17.5-1 Master Access Source from HP System 672
17.5-2 APM Controllers Configuration Registers 674

18.3-1 Security Level 710

19.5-1 HP CPU Packet Format 725
19.5-2 DMA Packet Format 725
19.5-3 LOST Packet Format 725

19.5-4 DMA Access Source 726
19.5-5 DMA_0 Packet Format 728
19.5-6 DMA LOST Packet Format 729

20.3-1 Configuration for Detecting Power Pins 760
20.4-1 Power Supply Detector's Internal Interrupt Source 760

21.3-1 ECC Accelerator Memory Blocks 772
21.4-1 ECC Accelerator Key Size Selection 774
21.4-2 Working Modes of ECC Accelerator 774

22.3-1 SHA Accelerator Working Mode 786
22.3-2 SHA Hash Algorithm Selection 786
22.4-1 The Storage and Length of Message Digest from Different Algorithms 790
22.4-2 SHA's Internal Interrupt Sources 790

23.4-1 ECDSA_DS Working Mode 799
23.4-2 ECDSA_DS Elliptic Curve Selection 799
23.4-3 ECDSA_DS SHA Algorithm 799
23.4-4 ECDSA_DS Working State 799
23.6-1 ECDSA_DS Memory Blocks 807

24.4-1 $Key$ Generated Based on $Key_A$ 818
24.4-2 Mapping Between Offsets and Registers 819
24.6-1 Configuration of XTS_AES Pseudo-round Anti-DPA 824

26.4-1 UART_CHAR_WAKEUP Mode Configuration 847

27.5-1 Data Modes Supported by GP-SPI2 882
27.5-2 Functional Description of FSPI Bus Signals 882
27.5-3 FSPI Bus Signals Used in Various SPI Modes 883
27.5-4 Bit Order Control in GP-SPI2 885
27.5-5 Supported Transfer Types as Master or Slave 886
27.5-6 Interrupt Trigger Condition on GP-SPI2 Data Transfer as Slave 891
27.5-7 Registers Used for State Control in 1/2/4-bit Modes 898
27.5-8 Sending Sequence of Command Value 900
27.5-9 Sending Sequence of Address Value 900
27.5-10 BM Table for CONF State 906
27.5-11 An Example of CONF buffer*i* in Segment *i* 906
27.5-12 BM Bit Value and Register to Be Updated in This Example 907
27.5-13 CMD Values Supported in SPI Mode 910
27.5-13 CMD Values Supported in SPI Mode 911
27.5-14 CMD Values Supported in QPI Mode 911
27.7-1 Clock Phase and Polarity Configuration as Master 917
27.7-2 Clock Phase and Polarity Configuration as Slave 917
27.9-1 Internal Interrupt Sources of GP-SPI2 920
27.9-2 GP-SPI2 Master Mode Interrupts 921
27.9-3 GP-SPI2 Slave Mode Interrupts 923

28.3-1 I2C Timing Parameters (Cited from Table 5 in The I2C-bus specification Version 2.1) 968
28.4-1 I2C Synchronous Registers 971

29.4-1 I2S Signal Description 1027
29.9-1 Bit Width of Channel Valid Data 1034
29.9-2 Endian of Channel Valid Data 1035
29.9-3 Data-Fetching Control in PDM Mode 1038
29.9-4 I2S Channel Control for Raw PDM Data 1038
29.9-5 I2S PCM-to-PDM Data Output 1039
29.10-1 Channel Storage Data Width 1041
29.10-2 Channel Storage Data Endian 1042

30.3-1 Standard CDC-ACM Control Requests 1066
30.3-2 CDC-ACM Settings with RTS and DTR 1067
30.3-3 Commands of a Nibble 1069
30.3-4 USB-to-JTAG Control Requests 1071
30.3-5 JTAG Capability Descriptors 1071
30.5-1 Reset SoC into Download Mode 1073
30.5-2 Reset SoC into Booting from flash 1073

31.5-1 SDIO Slave CCCR Configuration 1105
31.5-2 SDIO Slave FBR Configuration 1106

32.4-1 Commonly-used Frequencies and Resolutions 1163
32.7-1 LEDC Memory Blocks 1170

33.4-1 Temperature Measurement Range and Offset 1189

34.5-1 SAR ADC Channels 1199

35.4-1 Mapping Between PAD and GPIO 1221
35.6-1 Analog Voltage Comparator's Internal Interrupt Sources 1222

35.7-4 Configuration of ENA/RAW/ST Registers 1230

# List of Figures


1.0-1 ESP32-C61 Bus Architecture 36

2.1-1 CPU Core Complex Block Diagram 37
2.10-1 Debug System Overview 102
2.11-1 Two-bit Saturation Counter 117
2.12-1 CPU Core State Transition Using RunStall 124

3.0-1 Trace Encoder Overview 126
3.2-1 Trace Flow Overview 127
3.6-1 Trace packet Format 132

4.1-1 Modules that Share GDMA Channels 156
4.3-1 GDMA controller Architecture 157
4.4-1 Structure of a Linked List 158
4.4-2 Relationship among Linked Lists 161

5.3-1 System Structure and Address Mapping 203
5.3-2 Cache Structure 205
5.3-3 Modules/peripherals that can work with GDMA 207

6.3-1 Data Flow in eFuse 211
6.3-2 Shift Register Circuit (first 32 output) 218
6.3-3 Shift Register Circuit (last 12 output) 218

7.3-1 Architecture of HP GPIO Matrix, HP IO MUX, LP GPIO Matrix, and LP IO MUX 257
7.3-2 Internal Structure of a Pad 258
7.4-1 GPIO Input Synchronized on Rising Edge or on Falling Edge of HP IO MUX Operating Clock 259
7.4-2 GPIO Filter Timing of GPIO Input Signals 259
7.10-1 Example of Level Flip on the Chip Pad — Hysteresis Function Not Enabled 267
7.10-2 Example of Level Flip on the Chip Pad — Hysteresis Function Enabled 267

8.1-1 Reset Types 325
8.2-1 System Clock 328
8.3-1 I2C Clock Configuration Example 336

9.2-1 Chip Boot Flow 390

10.2-1 Interrupt Flow in ESP32-C61 394
10.4-1 Interrupt Matrix Structure 394

11.3-1 ETM Channel*n* Architecture 411
11.3-2 Event Task Matrix Clock Architecture 417

12.4-1 ESP32-C61 Power Scheme 498
12.4-2 PMU Workflow 500

13.3-1 System Timer Structure 599
13.5-1 System Timer Alarm Generation 600

14.1-1 Timer Group Overview 623
14.3-1 Timer Group Architecture 624

15.1-1 Watchdog Timers Overview 646
15.2-1 Digital Watchdog Timers in ESP32-C61 648
15.3-1 Super Watchdog Controller Structure 651

17.5-1 APM Controller Architecture 673

20.3-1 Architecture of Power Supply Detector 758
20.3-2 Brown-out Reset Workflow 759
20.3-3 Structure of Voltage Glitch Detectors 760

23.5-1 ECDSA_DS Process 802

24.3-1 Architecture of the External Memory Encryption and Decryption 817

25.3-1 Noise Source 833

26.3-1 UART Structure 839
26.4-1 UART Controllers Division 841
26.4-2 The Timing Diagram of Weak UART Signals Along Negative Edges 842
26.4-3 Structure of UART Data Frame 843
26.4-4 AT_CMD Character Structure 843
26.4-5 Driver Control Diagram in RS485 Mode 844
26.4-6 The Timing Diagram of Encoding and Decoding in SIR mode 846
26.4-7 IrDA Encoding and Decoding Diagram 846
26.4-8 Hardware Flow Control Diagram 848
26.4-9 Connection between Hardware Flow Control Signals 848
26.6-1 UART Programming Procedures 852

27.4-1 SPI Module Overview 881
27.5-1 Data Buffer Used in CPU-Controlled Transfer 887
27.5-2 GP-SPI2 Functional Blocks 892
27.5-3 Data Flow Control When GP-SPI2 Works as Master 893
27.5-4 Data Flow Control When GP-SPI2 Works as Slave 893
27.5-5 GP-SPI2 State Machine 896
27.5-6 Full-Duplex Communication Between GP-SPI2 Master and a Slave 901
27.5-7 Connection of GP-SPI2 to Flash and External RAM in 4-bit Mode 903
27.5-8 SPI Quad I/O Read Command Sequence Sent by GP-SPI2 to Flash 904
27.5-9 Configurable Segmented Transfer 904
27.6-1 Recommended CS Timing and Settings When Accessing External RAM 914
27.6-2 Recommended CS Timing and Settings When Accessing Flash 914
27.7-1 SPI Clock Mode 0 or 2 916
27.7-2 SPI Clock Mode 1 or 3 916
27.8-1 Timing Compensation Control Diagram in GP-SPI as Master 918
27.8-2 Timing Compensation Example in GP-SPI2 919

28.3-1 I2C Master Architecture 966

28.3-2 I2C Slave Architecture 967
28.3-3 I2C Protocol Timing (Cited from Fig.31 in The I2C-bus specification Version 2.1) 968
28.4-1 I2C Timing Diagram 972
28.4-2 Structure of I2C Command Registers 974
28.6-1 $I2C_{master}$ Writing to $I2C_{slave}$ with a 7-bit Address 979
28.6-2 $I2C_{master}$ Writing to a Slave with a 10-bit Address 982
28.6-3 $I2C_{master}$ Writing to $I2C_{slave}$ with Two 7-bit Addresses 983
28.6-4 $I2C_{master}$ Writing to $I2C_{slave}$ with a 7-bit Address in Multiple Sequences 985
28.6-5 $I2C_{master}$ Reading $I2C_{slave}$ with a 7-bit Address 987
28.6-6 $I2C_{master}$ Reading $I2C_{slave}$ with a 10-bit Address 989
28.6-7 $I2C_{master}$ Reading N Bytes of Data from addrM of $I2C_{slave}$ with Two 7-bit Addresses 992
28.6-8 $I2C_{master}$ Reading $I2C_{slave}$ with a 7-bit Address in Segments 994

29.4-1 ESP32-C61 I2S System Diagram 1026
29.5-1 TDM Philips Standard Timing Diagram 1029
29.5-2 TDM MSB Alignment Standard Timing Diagram 1029
29.5-3 TDM PCM Standard Timing Diagram 1030
29.5-4 PDM Standard Timing Diagram 1030
29.6-1 I2S Clock Generator 1031
29.9-1 TX Data Format Control 1036
29.9-2 TDM Channel Control 1037
29.9-3 PDM Channel Control Example 1040

30.2-1 USB Serial/JTAG High Level Diagram 1065
30.2-2 USB Serial/JTAG Block Diagram 1066

31.3-1 SDIO Slave Block Diagram 1103
31.5-1 CMD52 Content 1104
31.5-2 CMD53 Content 1104
31.5-3 Function 0 Address Space 1105
31.5-4 Function 1/2 Address Space Map 1107
31.5-5 DMA Linked List Descriptor Structure of the SDIO Slave 1109
31.5-6 DMA Linked List of the SDIO Slave 1110
31.5-7 Data Flow of Sending Incremental-address Packets From Host to Slave 1111
31.5-8 Sampling Timing Diagram 1112
31.5-9 Output Timing Diagram 1113
31.7-1 Procedure of Slave Sending Packets to Host 1115
31.7-2 Procedure of Slave Receiving Packets from Host 1117
31.7-3 Loading Receiving Buffer 1118

32.3-1 LED PWM Architecture 1160
32.3-2 Timer and PWM Generator Block Diagram 1160
32.4-1 Frequency Division When LEDC_CLK_DIV is a Non-Integer Value 1162
32.4-2 Relationship Between Counter And Resolution 1162
32.4-3 LED PWM Output Signal Diagram 1164
32.4-4 Output Signal of Linear Duty Cycle Fading 1165
32.4-5 Output Signal of Gamma Curve Fading 1166

33.3-1 Temperature Sensor Architecture 1188

34.4-1 SAR ADC Architecture 1198
34.5-1 SAR ADC Clock Structure 1199
34.5-2 DIG ADC FSM Block Diagram 1201
34.5-3 APB_SARADC_SAR_PATT_TAB1_REG Contains Patterns 0 - 3 1202
34.5-4 APB_SARADC_SAR_PATT_TAB2_REG Contains Patterns 4 - 7 1202
34.5-5 Pattern Structure 1202
34.5-6 cmd0 configuration 1203
34.5-7 cmd1 Configuration 1203
34.5-8 DMA Data Format 1204

35.3-1 Analog Voltage Comparator Architecture 1220

# Part I

# Microprocessor and Master

This part covers the essential processing elements of the system, diving into the microprocessor architecture with high-performance CPU. Details include trace encoding for debugging purposes and controllers for Direct Memory Access (DMA).

# Chapter 1

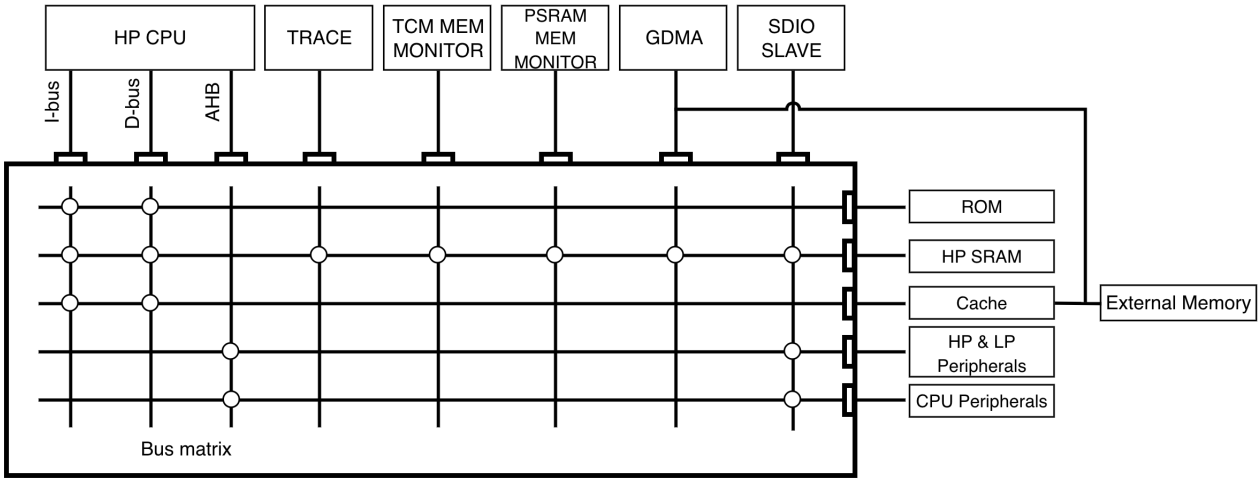
# Bus Architecture

The ESP32-C61 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 RISC-V core, 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-C61 SoC consists of:

- Masters:
  - High-performance (HP) 32-bit RISC-V processor I-bus, D-bus, and AHB bus
  - TRACE
  - TCM MEM MONITOR
  - PSRAM MEM MONITOR
  - GDMA
  - SDIO SLAVE
- Slaves:
  - ROM
  - HP SRAM
  - Cache
  - HP & LP peripherals
  - CPU peripherals

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. The GDMA has a data path that bypasses the bus matrix and directly accesses the external memory.

**Figure 1.0-1. ESP32-C61 Bus Architecture**

Below is the description of each bus shown in Figure 1.0-1 *ESP32-C61 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. Accessible targets include memory that stores code (such as ROM, HP SRAM, or external memory accessed via MSPI).
- **HP CPU D-bus**: Connects the data bus of the high-performance (HP) 32-bit RISC-V processor to the bus matrix for literal loads and data read/write. Accessible targets include memory that stores code or data (such as ROM, HP SRAM, or external memory accessed via MSPI).
- **HP CPU AHB bus**: Connects the AHB bus of the high-performance (HP) 32-bit RISC-V processor to the bus matrix for accessing peripherals' data. Accessible targets include HP & LP peripherals and CPU peripherals.
- **TRACE bus**: Connects the TRACE module to the bus matrix for performing debug operations. Accessible targets include HP SRAM.
- **TCM MEM MONITOR bus**: Connects the TCM MEM MONITOR to the bus matrix for writing monitored data into memory. Accessible targets include HP SRAM.
- **PSRAM MEM MONITOR bus**: Connects the PSRAM MEM MONITOR to the bus matrix for writing monitored data into memory. Accessible targets include HP SRAM.
- **GDMA bus**: Provides one route connected to the bus matrix and another route connected to external memory. GDMA uses this bus for high-speed data transfers, including transfers between peripherals and memory, as well as memory-to-memory transfers. Accessible targets include HP SRAM and external memory. For scenarios where GDMA accesses external memory, it is recommended to use a longer burst length whenever possible to ensure high data throughput.
- **SDIO SLAVE bus**: Connects the SDIO SLAVE to the bus matrix for accessing data in memory or peripherals. Accessible targets include HP SRAM, HP & LP peripherals, and CPU peripherals.

# Chapter 2

# ESP-RISC-V CPU

## 2.1 Overview

ESP32-C61 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 a RISC-V CPU core 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.

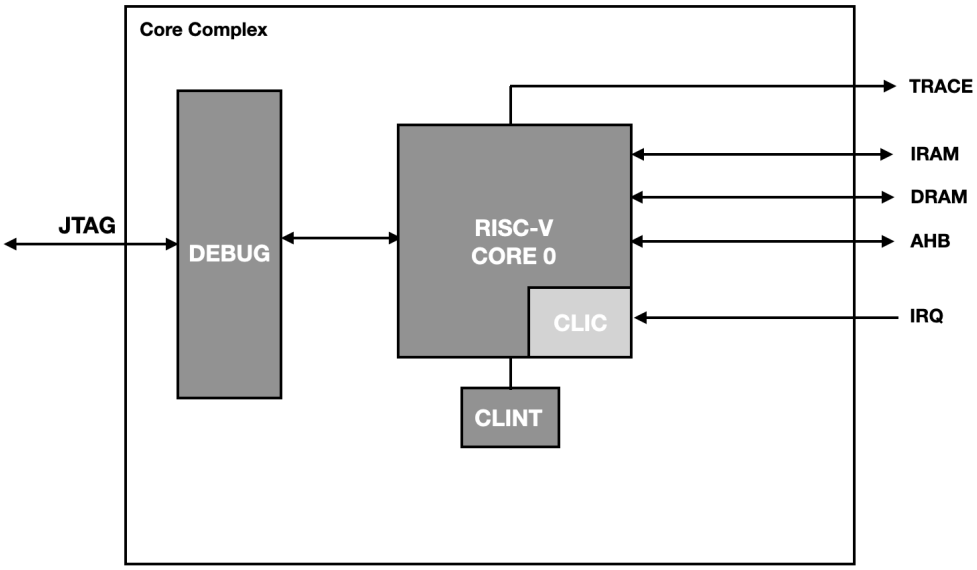



Figure 2.1-1. CPU Core Complex Block Diagram

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

## 2.2 Features

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

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

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

- Five-stage pipeline that supports an operating clock frequency up to 160 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 RISC-V Core-Local Interrupt Controller (CLIC) Version 0.9
- 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) support
- User (U) privilege mode execution
- User (U) mode interrupt delegation via CLIC
- Bus error response
- Standard physical memory protection (PMP) and custom attributes (PMA) for up to 16 configurable 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 the core 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
- Support for up to 8 dedicated IOs
- Configurable events for core performance metrics

## 2.3 Terminology

To better illustrate the functionality of ESP-RISC-V 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 | 0x4000_0000 | 0x4FFF_FFFF | R/W |
| CPU peripherals | AHB (Strong order access) | 0x6000_0000 | 0x6FFF_FFFF | R/W |
| AHB | AHB (Weak Order Access) | *Default | *Default | R/W |

***Default:** Address 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 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 |
| **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 |
| mintthresh | Interrupt threshold (Refer to CLIC specifications) | 0x347 | R/W |
| mintstatus | Current interrupt levels (Refer to CLIC specifications) | 0xFB1 | RO |
| 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** | | | |

[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

[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 |
|---|---|---|---|
| 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 register (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 |
| pmpaddr*n* (*n*: 0-15) | 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 |
| **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 |

[4] utvec Holds trap vector configuration consisting of vector base address and a vector mode

[5] External interrupt IDs reflected in mcause include even those IDs which have been reserved by RISC-V standard for core internal sources.

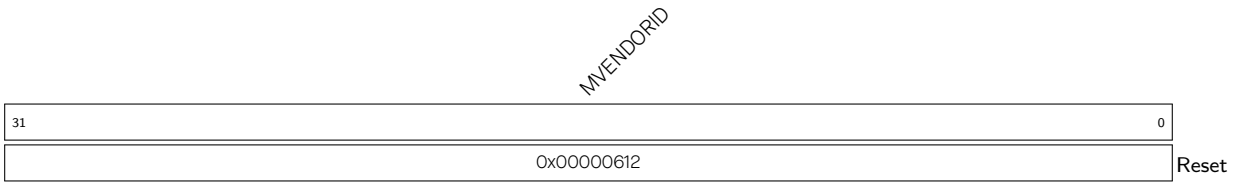
| Name | Description | Address | Access |
|---|---|---|---|
| mhpmcounter9 | Machine performance-monitoring counter | 0xB09 | R/W |
| mhpmcounter13 | Machine performance-monitoring counter | 0xB0D | R/W |
| mcycleh | Upper 32 bits of mcycle | 0xB80 | R/W |
| minstreth | Upper 32 bits of minstret | 0xB82 | 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 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 |
| 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 |
| | | | |
| **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 |
| | | | |
| **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 |
| **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 |
| 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 |

| Name | Description | Address | Access |
|---|---|---|---|
| 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.

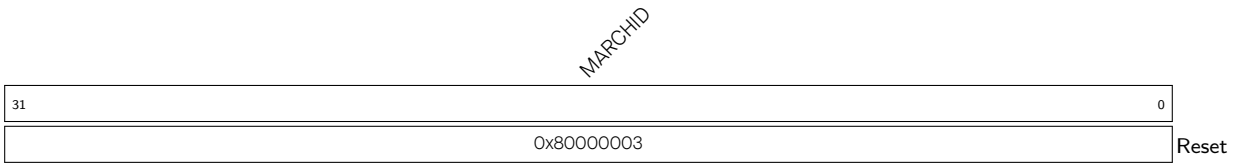
## 2.5.2 Register Description

Register 2.1. mvendorid (0xF11)



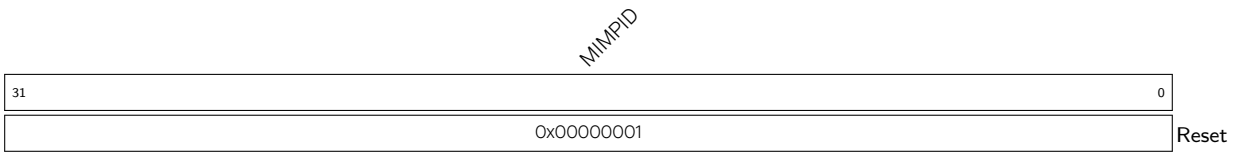

**MVENDORID** Represents Vendor ID. (RO)

Register 2.2. marchid (0xF12)



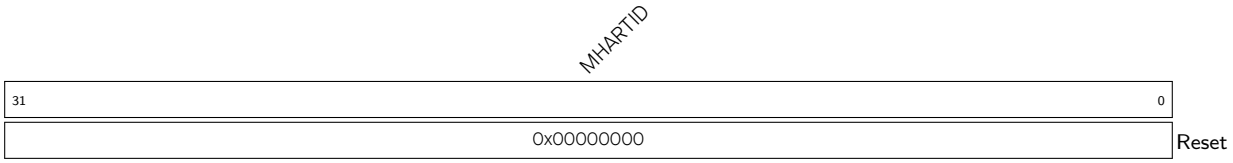

**MARCHID** Represents Architecture ID. (RO)

Register 2.3. mimpid (0xF13)




**MIMPID** Represents Implementation ID. (RO)

**Register 2.4. mhartid (0xF14)**



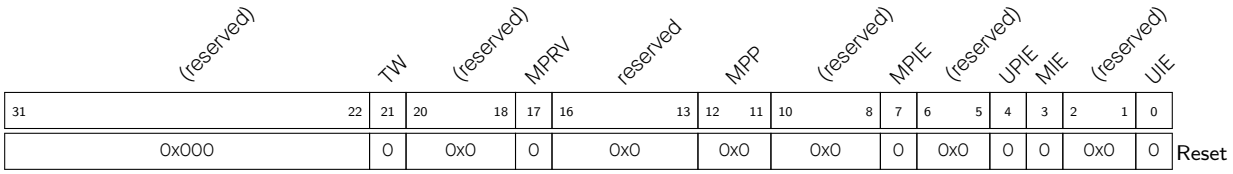

**MHARTID** Represents Hart ID.

0: Hart 0

1: Hart 1

(RO)

Register 2.5. mstatus (0x300)




**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)

**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)

**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)

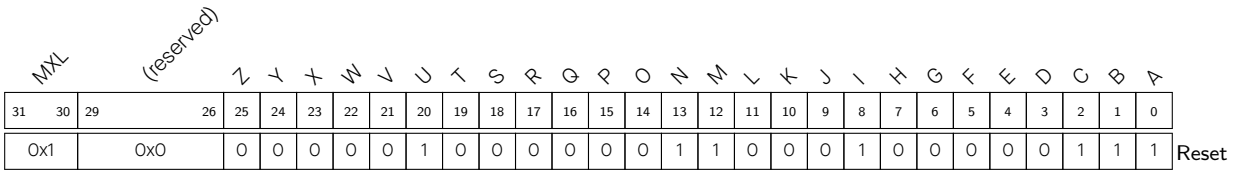
**UIE** Configures whether to enable the global user interrupt.

0: Disable

1: Enable

(R/W)

Register 2.6. misa (0x301)



| MXL | (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 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 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 = 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 = 1. (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)

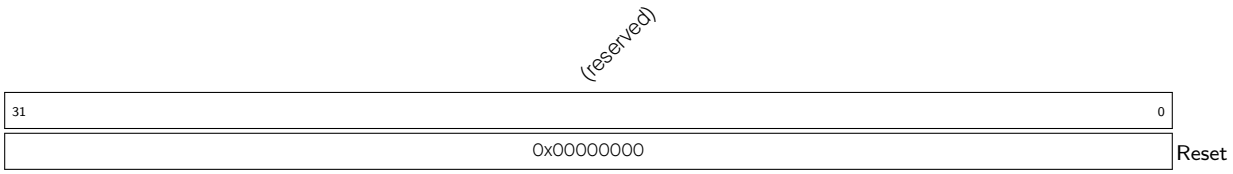
**D** Double-precision floating-point extension = 0. (RO)

**C** Compressed Extension = 1. (RO)

**B** Bit Manip = 1. (RO)

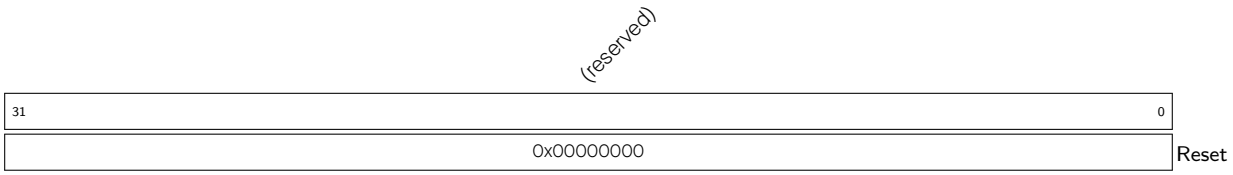
**A** Atomic Extension = 1. (RO)

Register 2.7. mideleg (0x303)



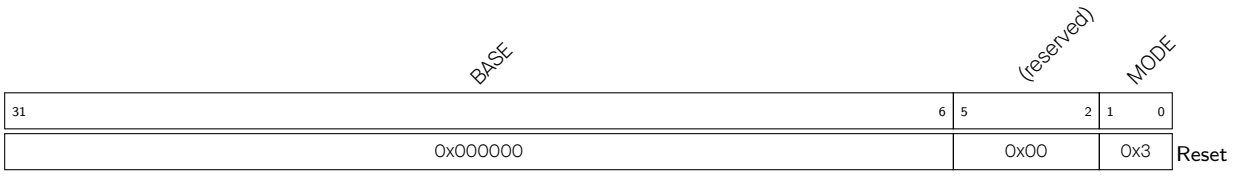

**mideleg** All bits are hardwired to 0 since only CLIC mode is available. (RO)

Register 2.8. mie (0x304)




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

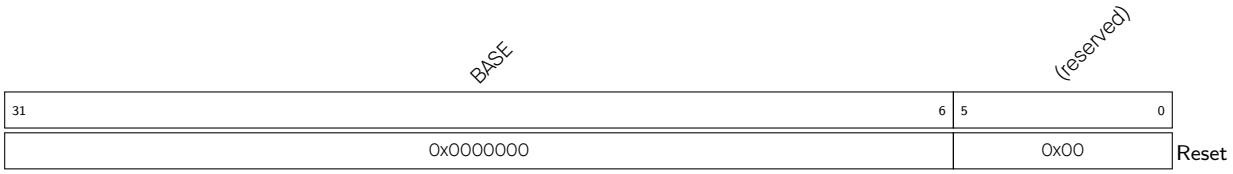
Register 2.9. mtvec (0x305)




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

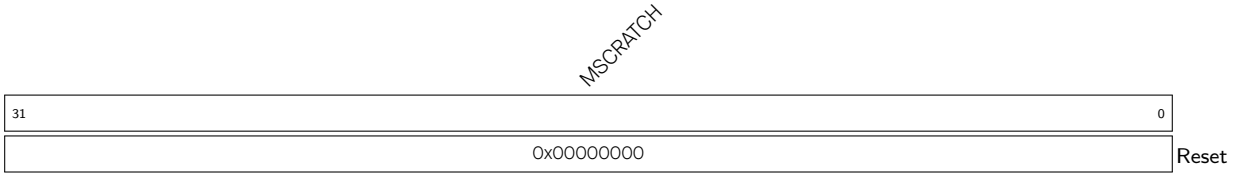
**MODE** Represents whether machine mode interrupts are operating in CLIC mode or vectored/non-vectored CLINT mode. Only CLIC mode **0x3** is available. (RO)

Register 2.10. mtvt (0x307)



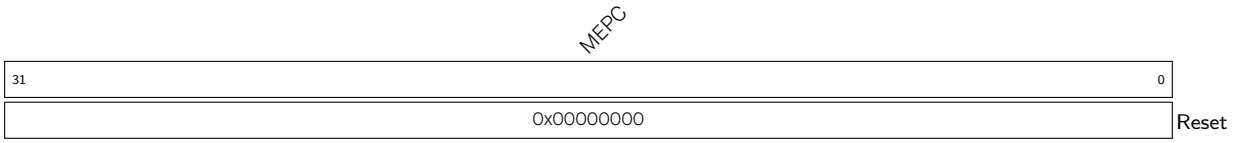

**BASE** Configures the higher 26 bits of CLIC machine mode interrupt vector base address aligned to 64 bytes. (R/W)

Register 2.11. mscratch (0x340)



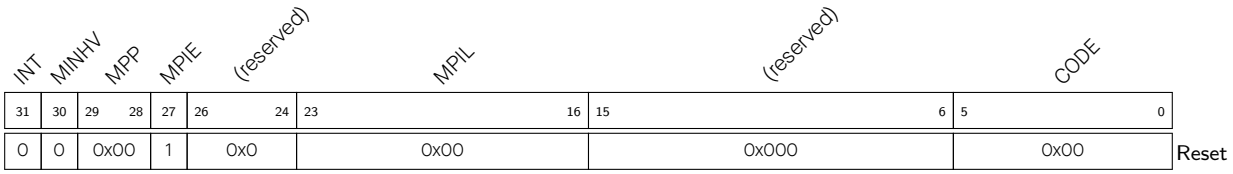

**MSCRATCH** Configures machine scratch information for custom use. (R/W)

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)

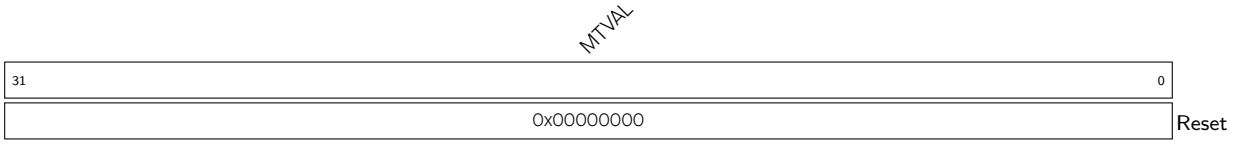
**MPP** Represents the previous privilege mode, same as mstatus.MPP. (R/W)

**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

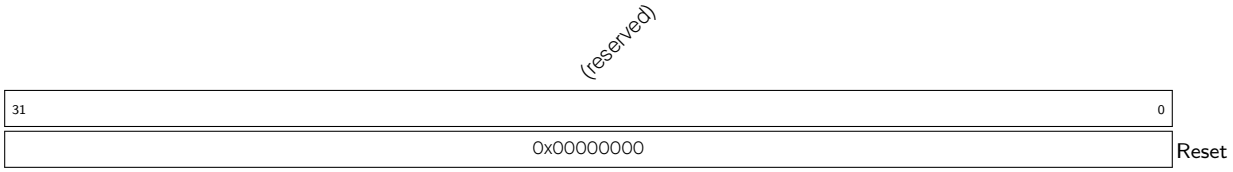
0x07: Faulting data address of store operation

0x1F: Faulting instruction opcode related to illegal PIE instruction

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

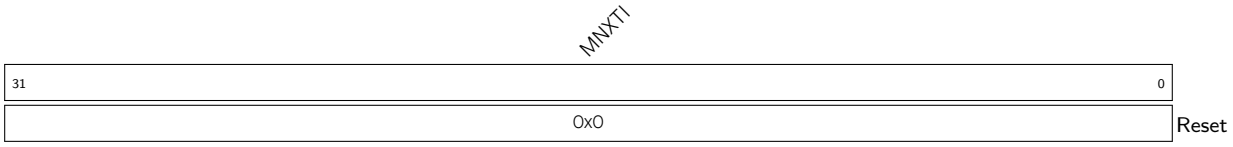
(R/W)

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.

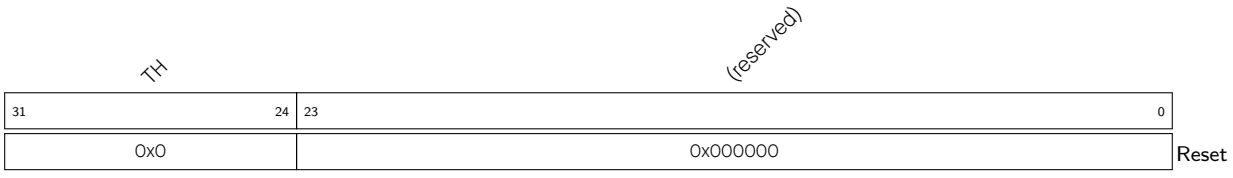
If the CSR instruction that accesses mnxti includes a write, the mstatus CSR is the one used for the read-modify-write portion of the operation, while the mcause register's exccode field and the mintstatus.MIL field can also be updated with the new interrupt id and level. If the interrupt is edge-triggered, 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. mintthresh (0x347)

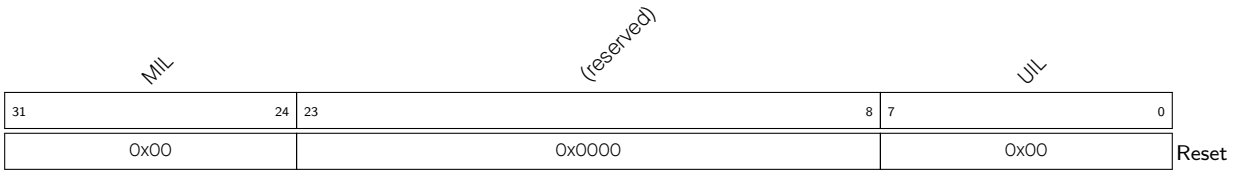



**TH** Configures the 8-bit level threshold of machine mode interrupts.

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)

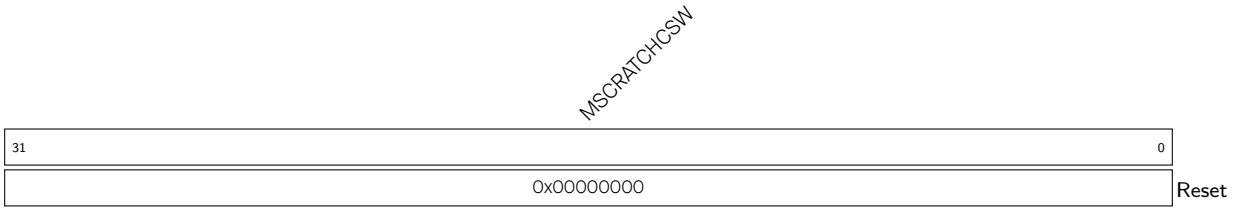
Register 2.18. mintstatus (0xFB1)




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

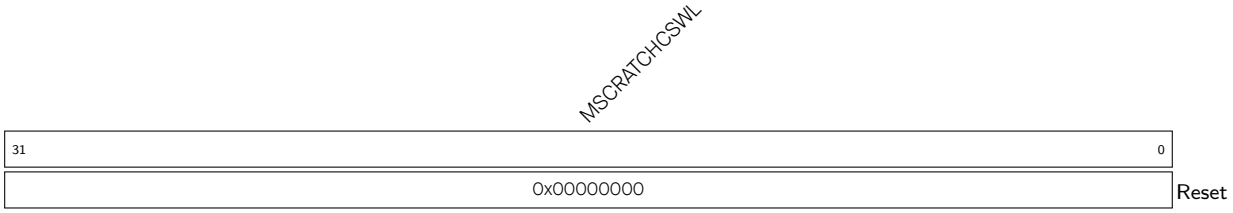
**UIL** Represents the active 8-bit interrupt level for current user mode interrupt. (RO)

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.

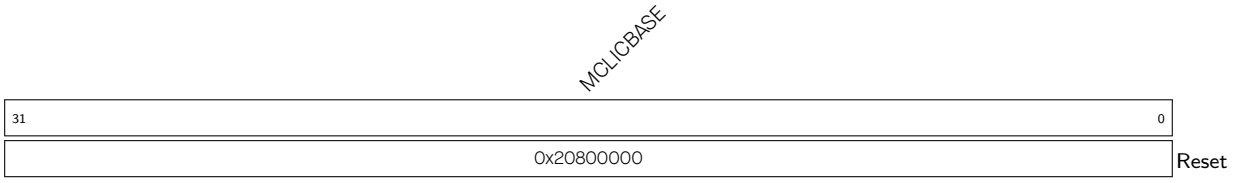
This is the conditional scratch swap CSR, which allows performing a scratch value swap conditionally, based on interrupt level change, in a single instruction.

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.

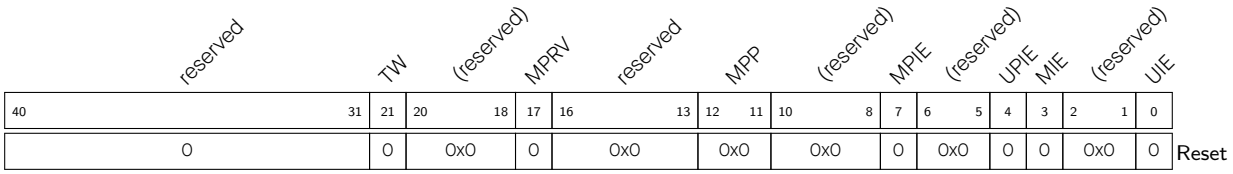
(R/W)

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)




**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)

**MPP** Represents machine previous privilege mode (before trap).

0x0: User mode

0x3: Machine mode

(RO)

Continued on the next page...

Register 2.22. ustatus (0x000)

Continued from the previous page...

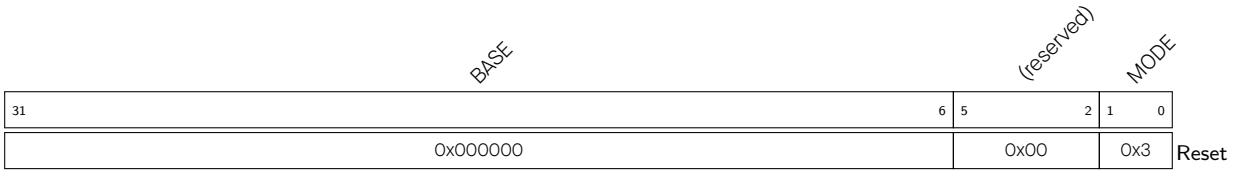
**MPIE** Represents the machine previous interrupt enable status (before trap).
0: Disabled
1: Enabled
(RO)

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

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

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

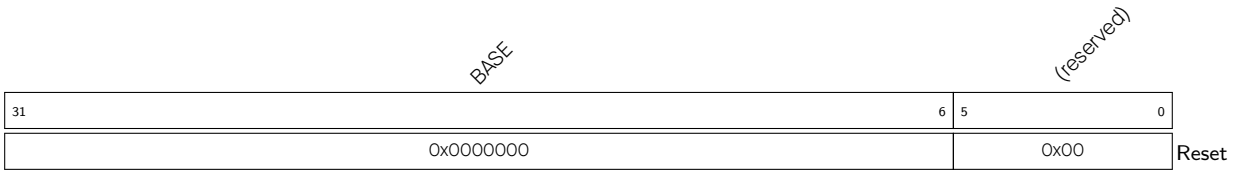
Register 2.23. utvec (0x005)




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

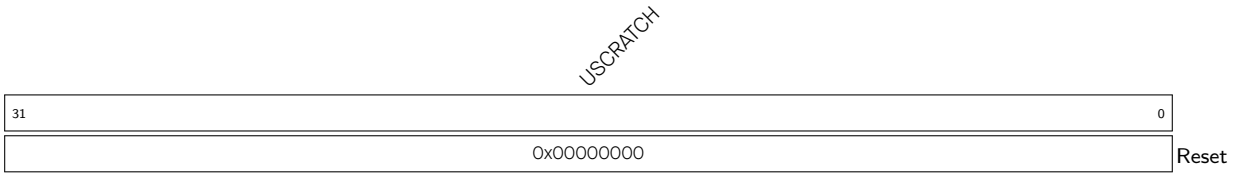
**MODE** Represents whether user mode interrupts are operating in CLIC mode or vectored/non-vectored CLINT mode. Only CLIC mode **0x3** is available. (RO)

Register 2.24. utvt (0x007)



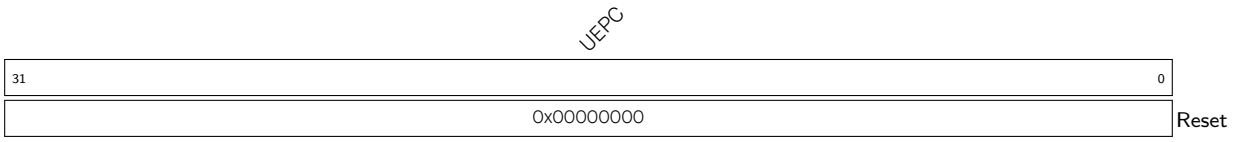

**BASE** Configures the higher 26 bits of CLIC user mode interrupt vector base address aligned to 64 bytes. (R/W)

Register 2.25. uscratch (0x040)



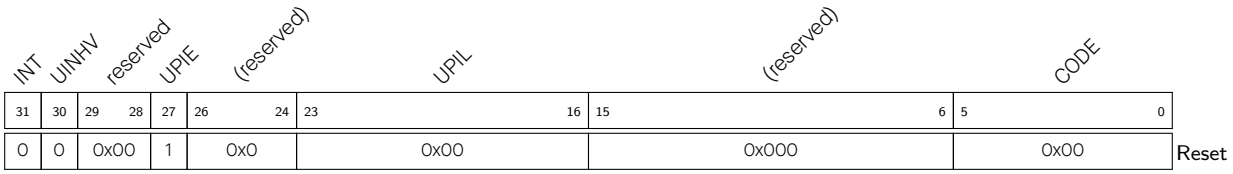

**USCRATCH** Configures user scratch information for custom use. (R/W)

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

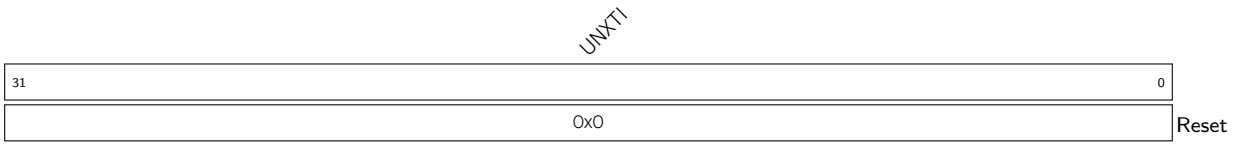
0x1f: Illegal PIE instruction

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.

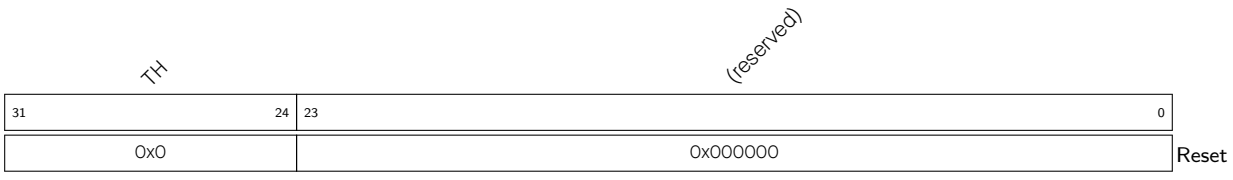
If the CSR instruction that accesses unxti includes a write, the ustatus CSR is the one used for the read-modify-write portion of the operation, while the ucause register's exccode field and the uintstatus.UIL field can also be updated with the new interrupt id and level. If the interrupt is edge-triggered,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

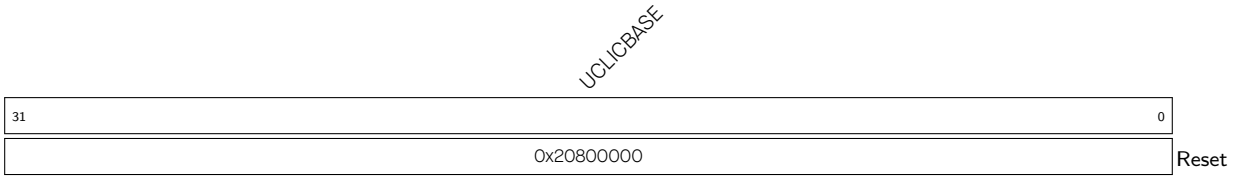



**TH** Configures the 8-bit level threshold of user mode interrupts.

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.

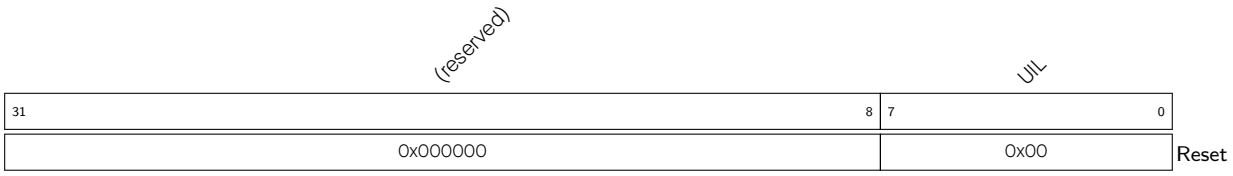
(R/W)

Register 2.30. uclicbase (0x050)



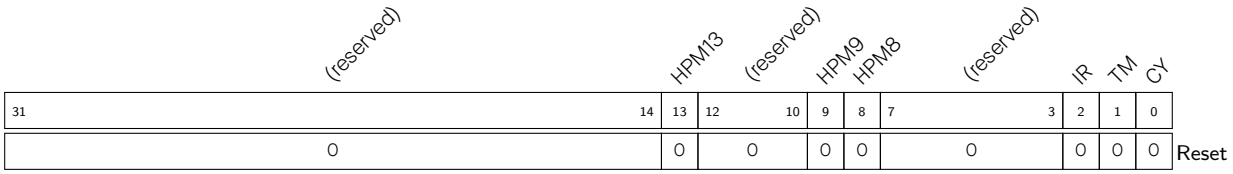

**UCLICBASE** Represents the base address of the CLIC memory-mapped registers for user mode. (RO)

Register 2.31. uintstatus (0xCB1)




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

Register 2.32. mcounteren (0x306)




**HPM13** Configures 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)

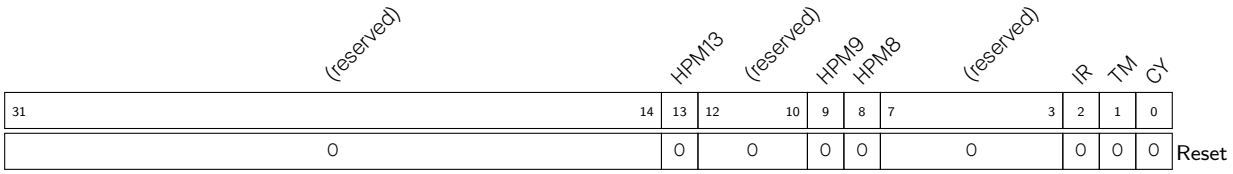
**IR** Configures access of instret counter from user mode.

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)

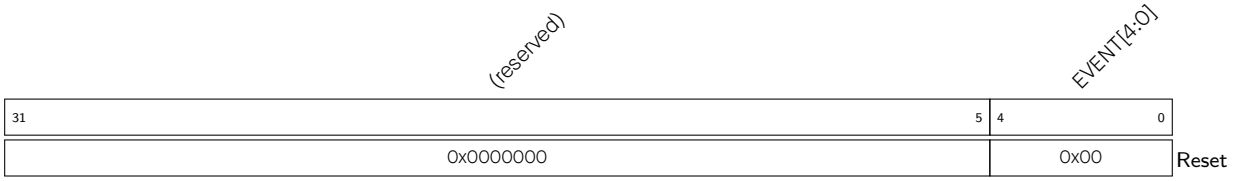
**CY** Configures the incrementing of the cycle counter.

0: Continue incrementing the cycle counter.

1: Stop incrementing the cycle counter.

(R/W)

Register 2.34. mhpmevent8 (0x328)

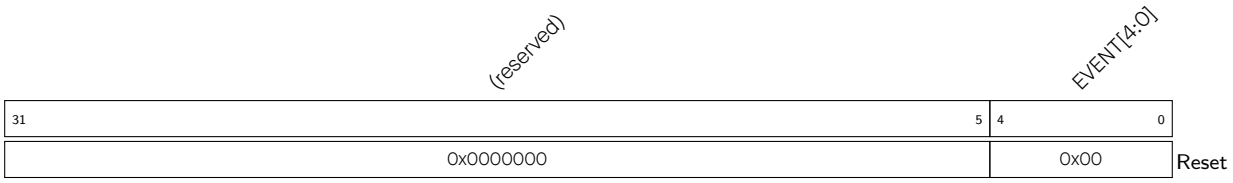



**EVENT[4:0]** Event selector for mhpmcounter8.

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

(R/W)

Register 2.35. mhpmevent9 (0x329)

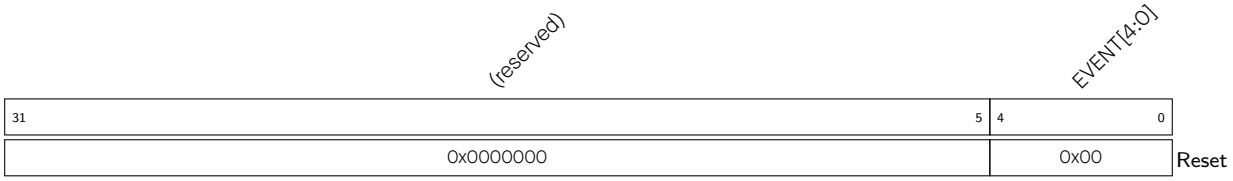



**EVENT[4:0]** Event selector for mhpmcounter9.

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

(R/W)

Register 2.36. mhpmevent13 (0x32D)

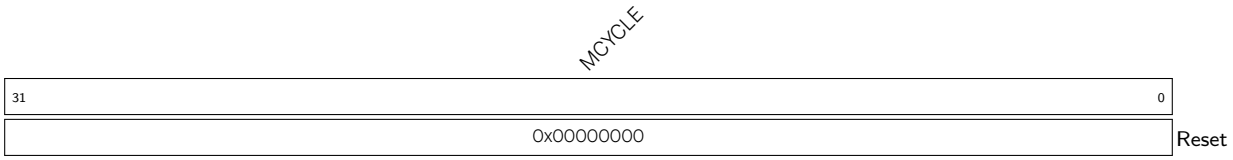



**EVENT[4:0]** Event selector for mhpmcounter13.

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

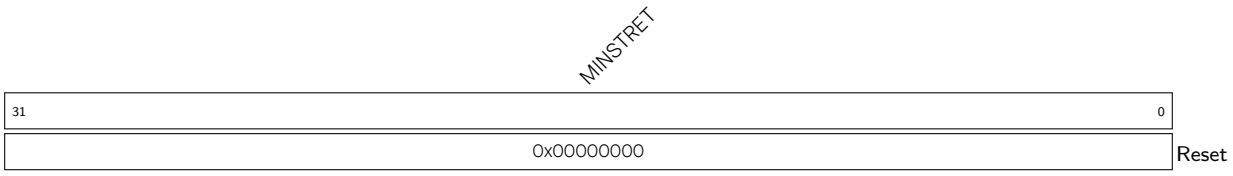
(R/W)

Register 2.37. mcycle (0xB00)



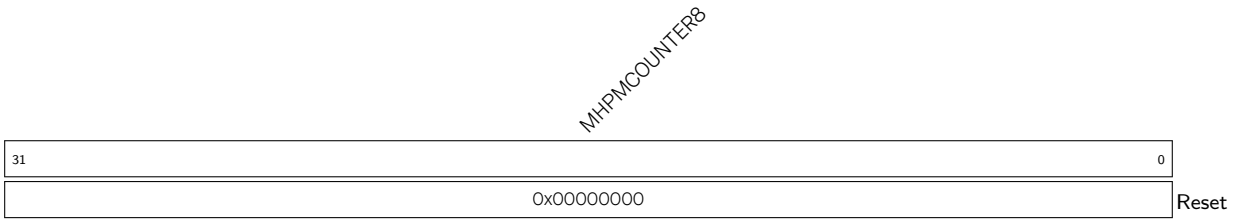

**MCYCLE** Represents the lower 32 bits of the machine mode cycle counter. (R/W)

Register 2.38. minstret (0xB02)



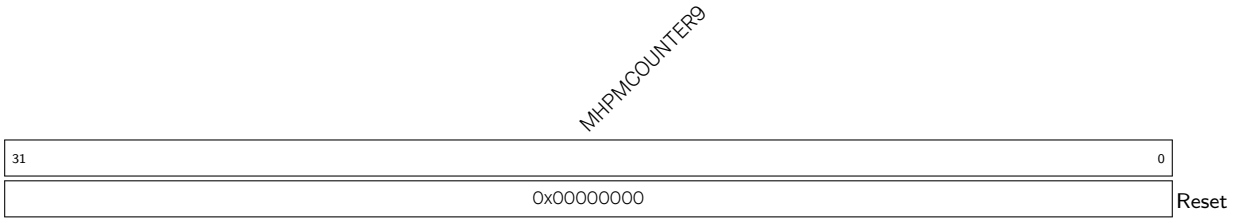

**MINSTRET** Represents the lower 32 bits of the machine instructions retired counter. (R/W)

Register 2.39. mhpmcounter8 (0xB08)



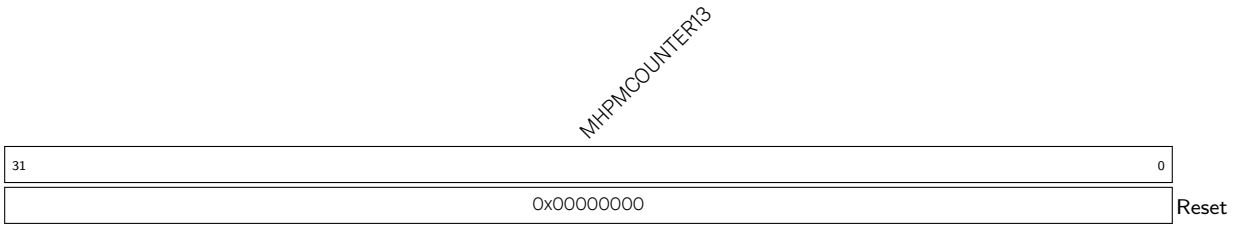

**MHPMCOUNTER8** Represents the lower 32 bits of the machine performance-monitoring counter8. (R/W)

Register 2.40. mhpmcounter9 (0xB09)



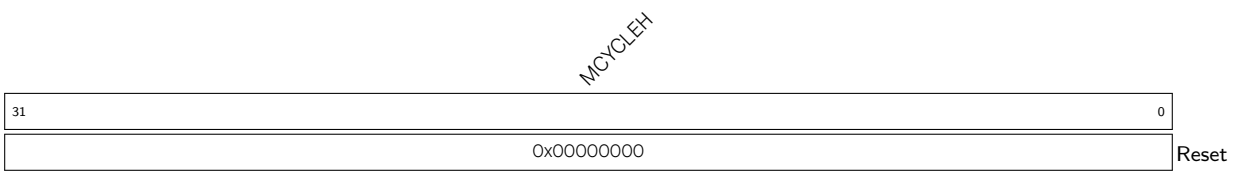

**MHPMCOUNTER9** Represents the lower 32 bits of the machine performance-monitoring counter9. (R/W)

Register 2.41. mhpmcounter13 (0xB0D)



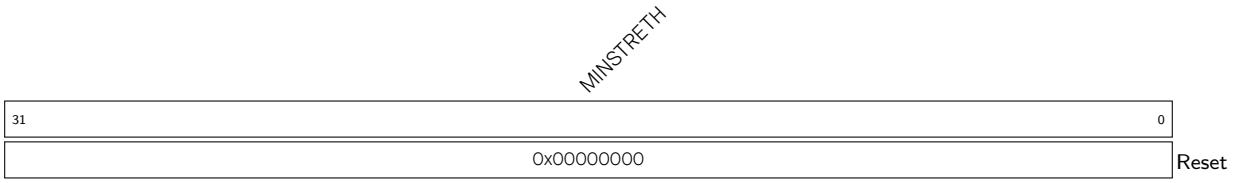

**MHPMCOUNTER13** Represents the lower 32 bits of the machine performance-monitoring counter13. (R/W)

Register 2.42. mcycleh (0xB80)



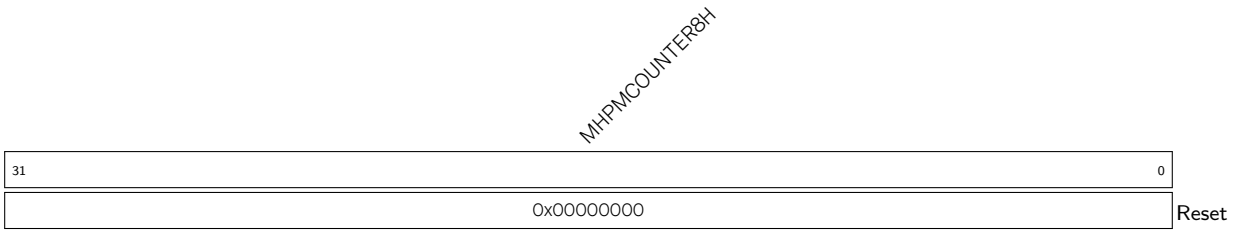

**MCYCLEH** Represents the upper 32 bits of mcycle counter. (R/W)

Register 2.43. minstreth (0xB82)



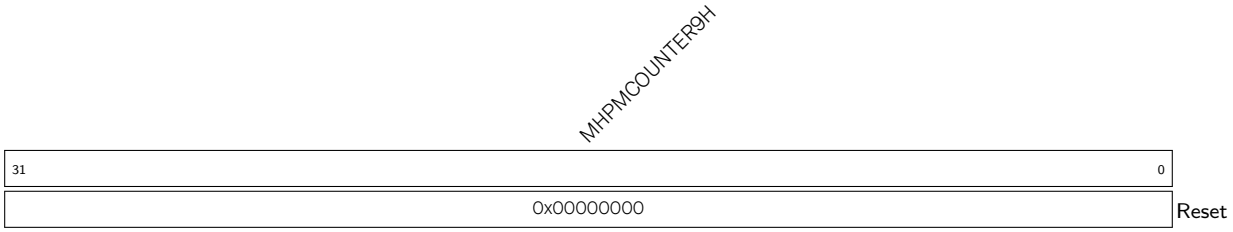

**MINSTRETH** Represents the upper 32 bits of minstret counter. (R/W)

Register 2.44. mhpmcounter8h (0xB88)



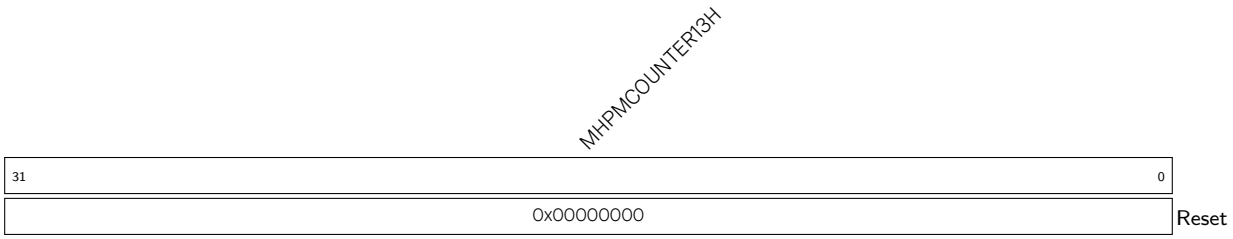

**MHPMCOUNTER8H** Represents the upper 32 bits of mhpmcounter8 counter. (R/W)

Register 2.45. mhpmcounter9h (0xB89)



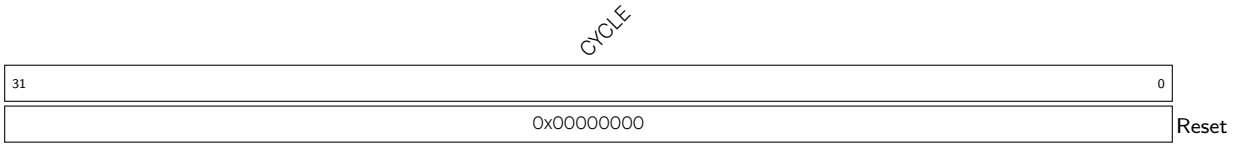

**MHPMCOUNTER9H** Represents the upper 32 bits of mhpmcounter9 counter. (R/W)

Register 2.46. mhpmcounter13h (0xB8D)



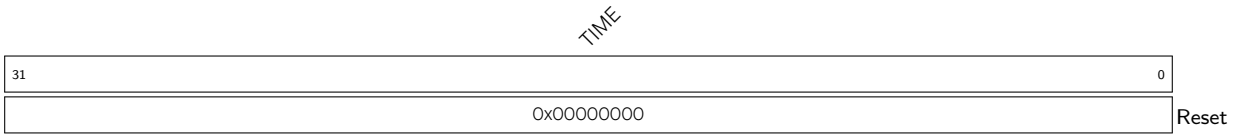

**MHPMCOUNTER13H** Represents the upper 32 bits of mhpmcounter13 counter. (R/W)

Register 2.47. cycle (0xC00)



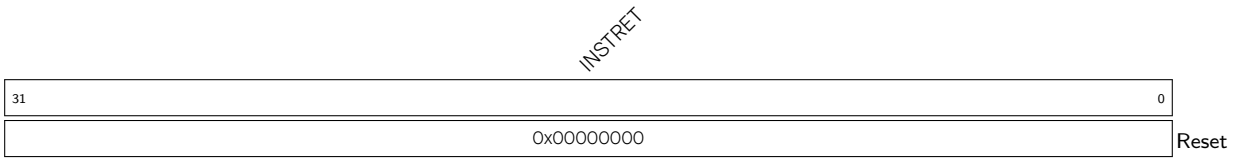

**CYCLE** Read-only mirror of mcycle. (RO)

Register 2.48. time (0xC01)



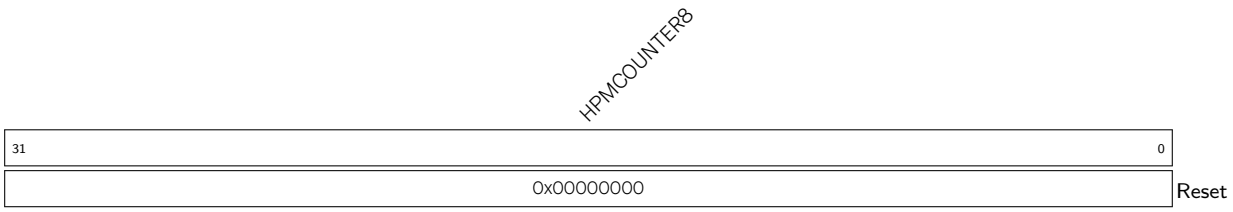

**TIME** Represents the timer value for RDTIME instruction. (RO)

Register 2.49. instret (0xC02)



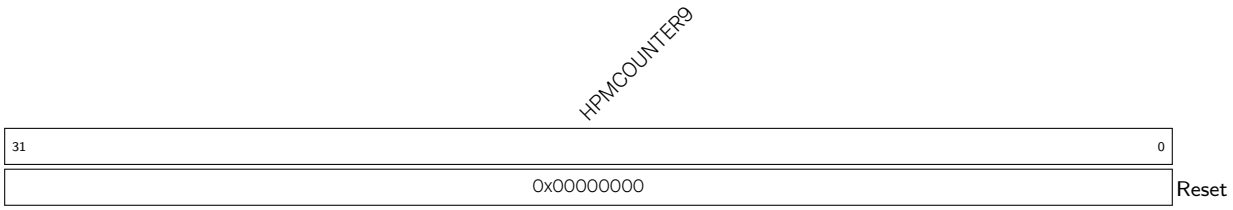

**INSTRET** Read-only mirror of minstret. (RO)

Register 2.50. hpmcounter8 (0xC08)



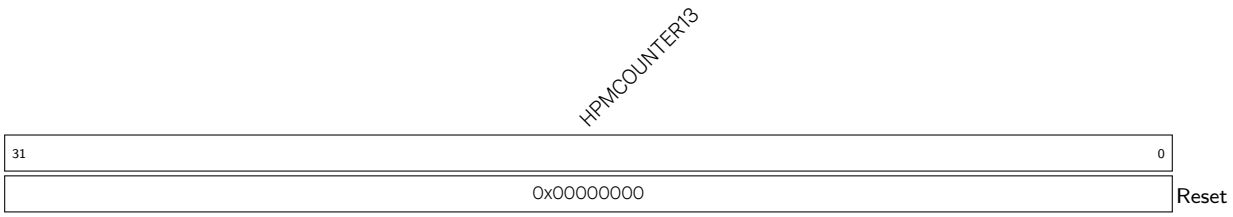

**HPMCOUNTER8** Read-only mirror of mhpmcounter8. (RO)

Register 2.51. hpmcounter9 (0xC09)



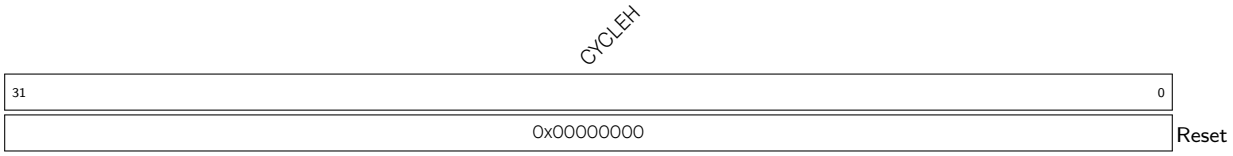

**HPMCOUNTER9** Read-only mirror of mhpmcounter9. (RO)

Register 2.52. hpmcounter13 (0xC0D)



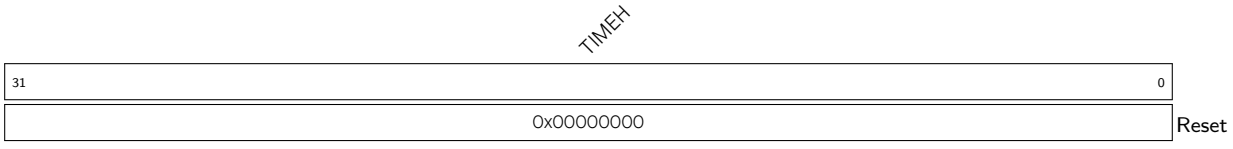

**HPMCOUNTER13** Read-only mirror of mhpmcounter13. (RO)

Register 2.53. cycleh (0xC80)



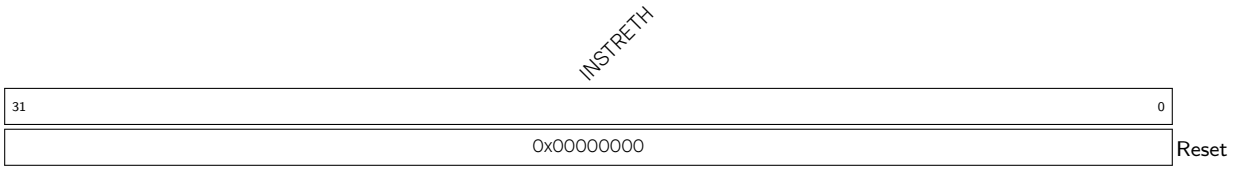

**CYCLEH** Read-only mirror of mcycleh. (RO)

Register 2.54. timeh (0xC81)



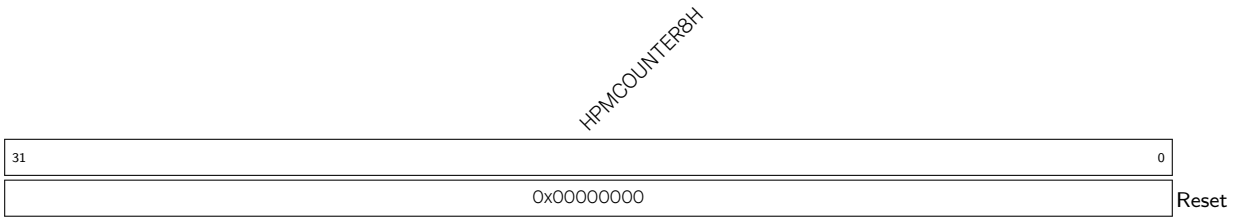

**TIMEH** Represents the upper 32 bits of the timer for RDTIME instruction. (RO)

Register 2.55. instreth (0xC82)



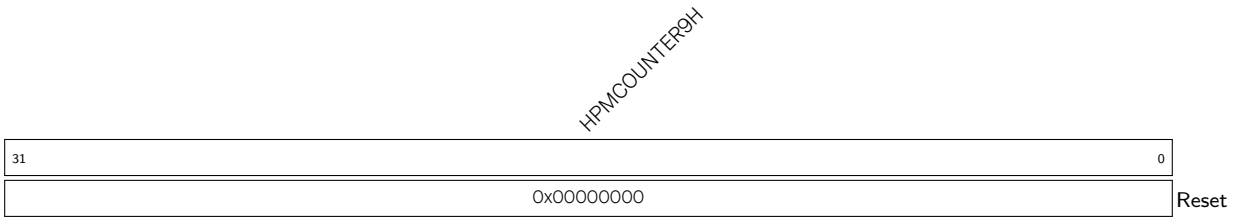

**INSTRETH** Read-only mirror of minstreth. (RO)

Register 2.56. hpmcounter8h (0xC88)



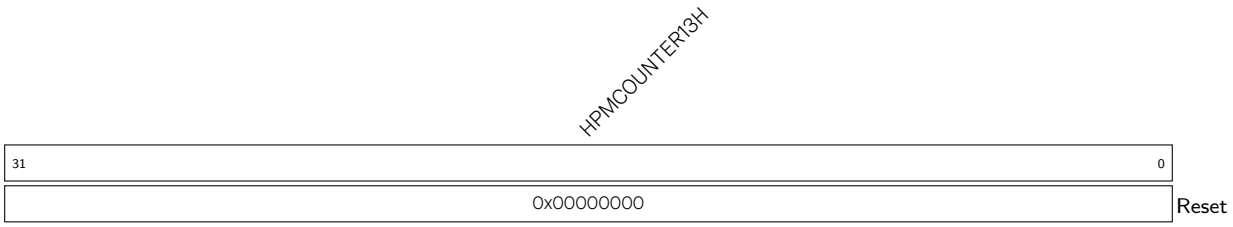

**HPMCOUNTER8H** Read-only mirror of mhpmcounter8h. (RO)

Register 2.57. hpmcounter9h (0xC89)



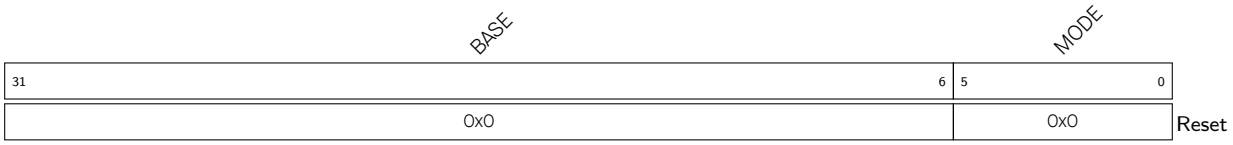

**HPMCOUNTER9H** Read-only mirror of mhpmcounter9h. (RO)

Register 2.58. hpmcounter13h (0xC8D)




**HPMCOUNTER13H** Read-only mirror of mhpmcounter13h. (RO)

Register 2.59. 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 2.60. 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

(R/W) 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 2.60. 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)

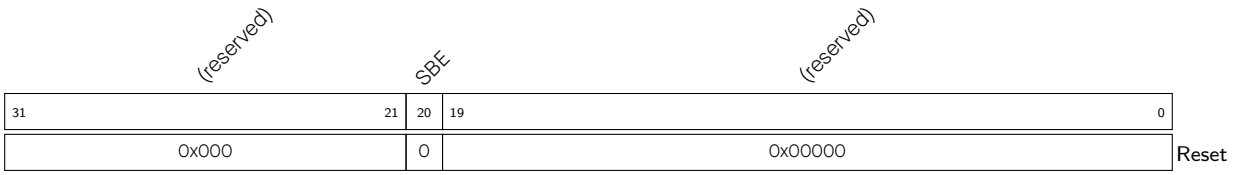
**CLIC_INHV** Configures whether to enable resuming faulting vector table fetch.

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 2.61. mhint (0x7C5)



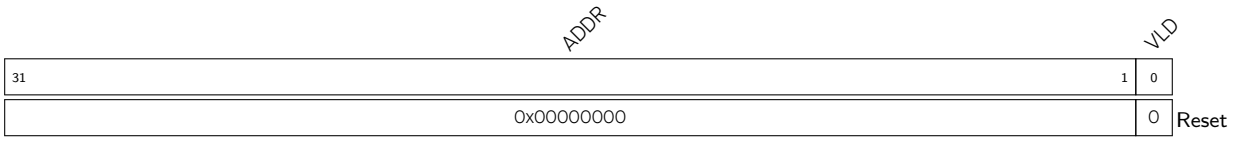

**SBE** Configures whether to enable the synchronous load/store bus-error exception.

0: Disable

1: Enable

(R/W)

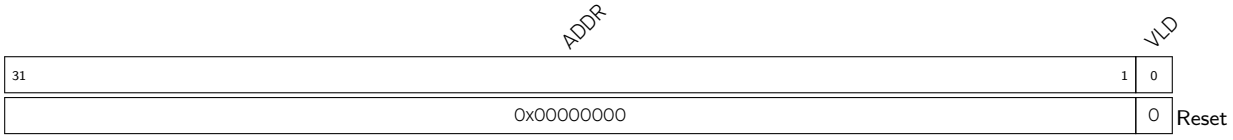
Register 2.62. ldpc0 (0xBE0)




**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)

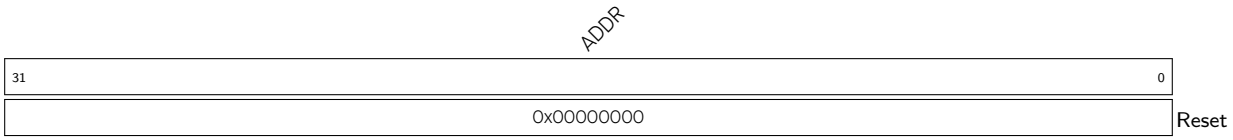
Register 2.63. ldpc1 (0xBE1)




**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)

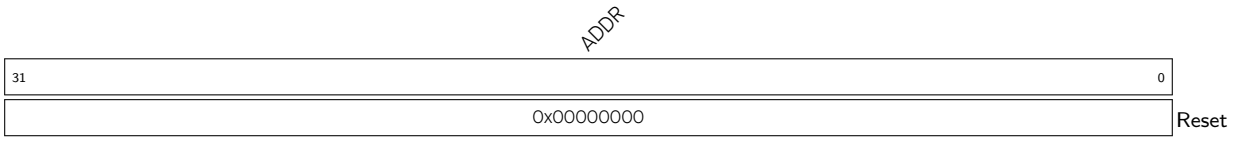
**ADDR** Configures the higher 31 bits of the half-word-aligned program counter corresponding to the same load bus-error exception. (R/W)

Register 2.64. ldtval0 (0xBE8)



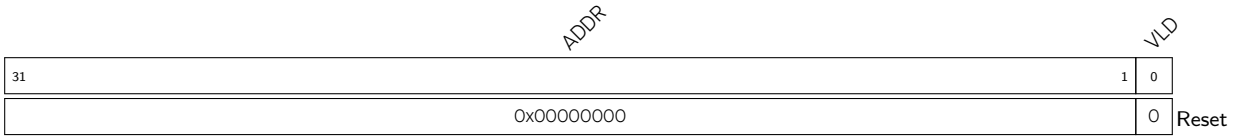

**ADDR** Configures the access address corresponding to the load bus-error exception recorded in ldpc0. (R/W)

Register 2.65. ldtval1 (0xBE9)




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

Register 2.66. stpc0 (0xBF0)




**VLD** Configures whether the program-counter address value in this CSR is valid for a recent store bus-error exception.

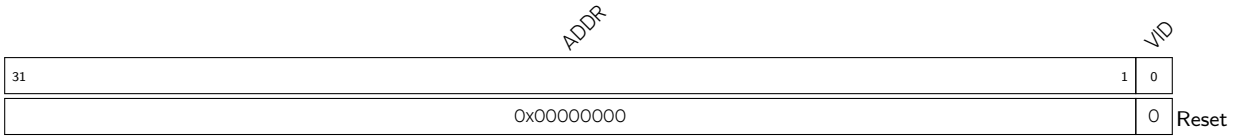
0: Not valid

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 2.67. stpc1 (0xBF1)




**VLD** Represents that the program-counter address value in this CSR is valid for a recent store bus-error exception.

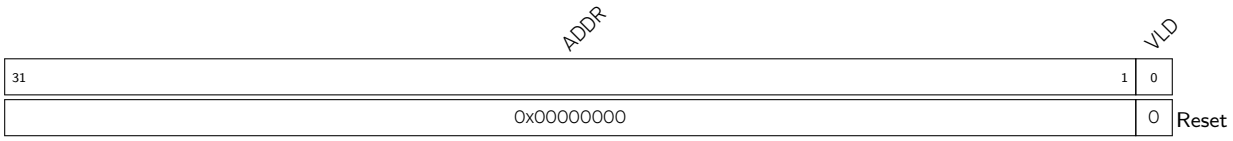
0: Not valid

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)

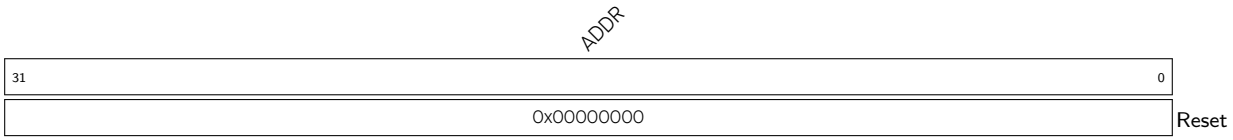
Register 2.68. stpc2 (0xBF2)




**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)

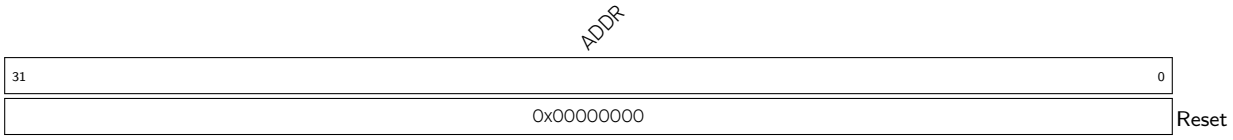
**ADDR** Configures the higher 31 bits of the half-word-aligned program counter corresponding to the same store bus-error exception. (R/W)

Register 2.69. sttval0 (0xBF8)



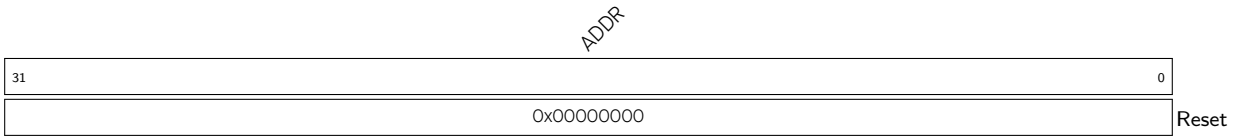

**ADDR** Configures the access address corresponding to the store bus-error exception recorded in stpc0. (R/W)

Register 2.70. sttval1 (0xBF9)




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

Register 2.71. 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

HP CPU 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 Multiplication/Division (M) Extension

#### 2.6.1.1 Overview

In addition to base RV32I instruction set, 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 Compressed (C) Extension

#### 2.6.2.1 Overview

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 link 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 Code Size Reduction (Zc) Extension

### 2.6.3.1 Overview

Zc extension is useful for code size reduction. HP core implements this extension compatible with Zc-specification v1.0.4-3. The Zc-specification has multiple subset extensions, and the ESP32-C61HP CPU has implemented Zcb, Zcmp, and Zcmt extensions.

### 2.6.3.2 Functional Description

**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

### 2.6.3.3 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 |

| Instruction | Mnemonic | Description |
|---|---|---|
| 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 |
| 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.

#### 2.6.3.4 Limitations

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

### 2.6.4 Bit Manipulation (Zb) Extension

#### 2.6.4.1 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.

#### 2.6.4.2 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.

### 2.6.5 Atomic (A) Extension

#### 2.6.5.1 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.

#### 2.6.5.2 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.

#### 2.6.5.3 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 2.6.5.4.

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

#### 2.6.5.4 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 2.6.5.3), 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.

#### 2.6.5.5 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 Memory-Mapped Registers

## 2.7.1 Overview

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

- CLIC registers
- CLINT registers

## 2.7.2 Features

HP CPU core supports:

- Access to CLIC registers
- Bus-error exception generation on invalid address access
- Access to CLINT registers

## 2.7.3 Functional Description

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

**Table 2.7-1. Address Decoding for Memory-Mapped Register Access**

| Bits | Description |
|---|---|
| 31:28 | Fixed as 0x2, i.e. base address |
| 27:20 | Identifies the sub module:<br>0x0: CLINT<br>0x8: CLIC<br>0xB: CLIC user<br>Others: Reserved |
| 19:16 | Controls access type<br>0x0: Access own registers<br>0x1: Reserved |
| 15:0 | Register offset |

**Table 2.7-2. Internal Memory Address Map**

| Block | Start Address | End Address |
|---|---|---|
| CLINT | 0x20000000 | 0x2000FFFF |
| CLIC | 0x20800000 | 0x2080FFFF |

# 2.8 Interrupt Controller

## 2.8.1 Overview

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 HP core.

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

## 2.8.2 CLIC

### 2.8.2.1 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.

### 2.8.2.2 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

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

Each 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.

The base address of the vector table is configured in mtvt. 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.

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.

### 2.8.2.3 Interrupt Mapping

CLIC in the 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 2.8-1. 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 2.8.3.

The mapping of the 32 external interrupts to the interrupt sources in SoC is managed using the Interrupt Matrix. Please refer to Chapter 10 *Interrupt Matrix* for details on how to configure and map to SoC interrupts to CLIC interrupts for the HP core.

#### 2.8.2.4 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 2.8.2.6) to the mclicbase value.

#### 2.8.2.5 Interrupt Level and Priority Encoding

Table 2.8-2 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 2.8-2. 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.

#### 2.8.2.6 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.

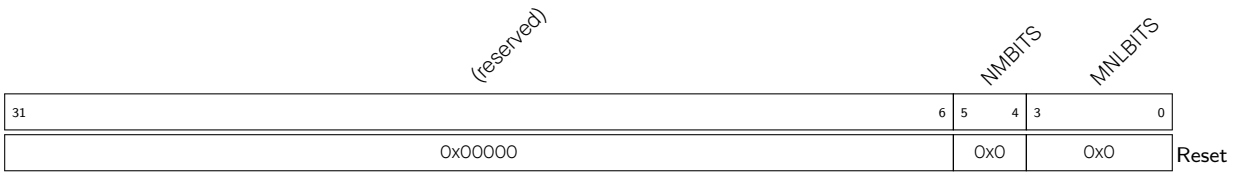
Note that the clicintip[i], clicintie[i], clicintattr[i] and clicintctl[i] registers are all byte sized. Still word and half-word aligned accesses to these registers will be performed atomically. Please note mintthresh is a machine mode CSR instead of memory mapped register.

| 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 |
| 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 |
| 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 |

| Name | Description | MCLICBASE Offset | Absolute Address | Access |
|---|---|---|---|---|
| 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 |

### 2.8.2.7 CLIC Memory-Mapped Register Description

**Register 2.72. mcliccfg (0x20800000)**




**MNLBITS** Configures the number of upper bits that are used globally (for 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 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)

[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.

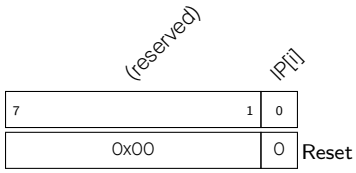
Register 2.73. clicinfo (0x20800004)




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




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

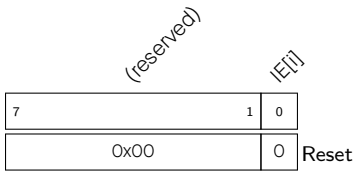
0: ith interrupt is not pending

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




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

0: Disable

1: Enabled

(R/W)

Register 2.76. 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)

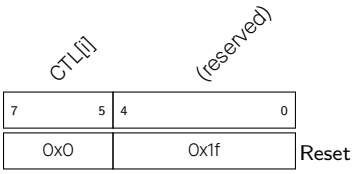
**MODE** Configures the privilege mode of the ith CLIC interrupt.

0x3: machine mode

0x0: user mode

(R/W)

Register 2.77. 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)

## 2.8.3 Core-Local Interrupts (CLINT)

### 2.8.3.1 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 2.8-4. Core-Local Interrupt Sources**

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

### 2.8.3.2 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

### 2.8.3.3 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.

### 2.8.3.4 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.

### 2.8.3.5 Register Summary

The addresses in this section are relative to CPU sub-system base address i.e. 0x20000000.

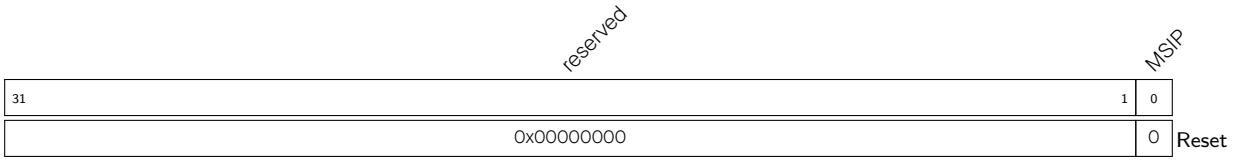
| Name | Description | Address | Access |
|---|---|---|---|
| msip | Core-local machine software interrupt pending register | 0x0000 | R/W |

| Name | Description | Address | Access |
|---|---|---|---|
| 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 |

### 2.8.3.6 Register Description

The addresses in this section are relative to CPU sub-system base address i.e. 0x20000000.

**Register 2.78. msip (0x0000)**




**MSIP** Configures the pending status of the machine software interrupt.

0: Not pending

1: Pending

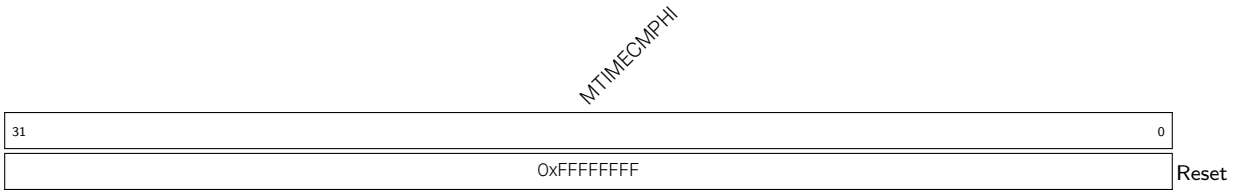
(R/W)

**Register 2.79. mtimecmplo (0x4000)**



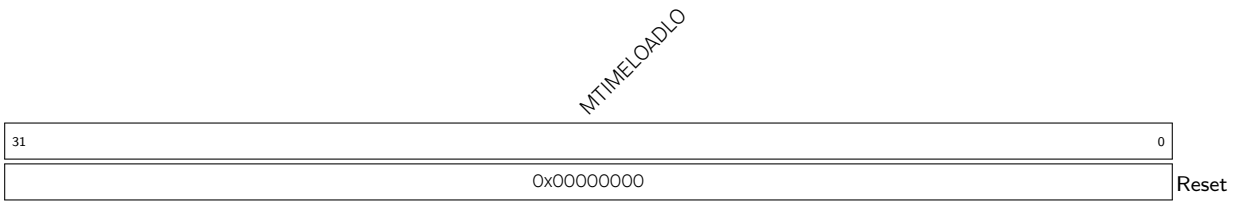

**MTIMECMPLO** Represents the value to compare with the lower 32 bits of the system counter. (R/W)

Register 2.80. mtimecmphi (0x4004)



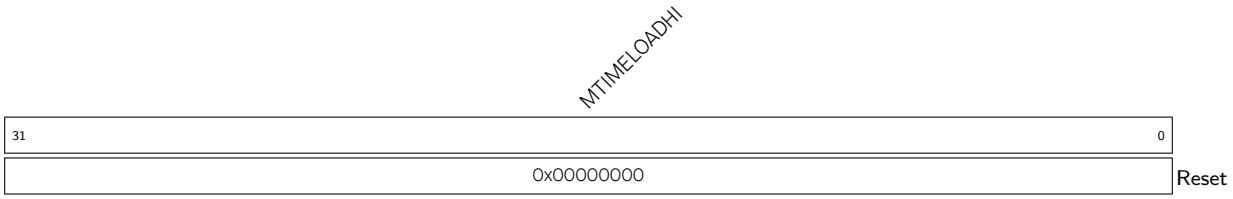

**MTIMECMPHI** Represents the value to compare with the higher 32 bits of the system counter. (R/W)

Register 2.81. mtimeloadlo (0x4008)



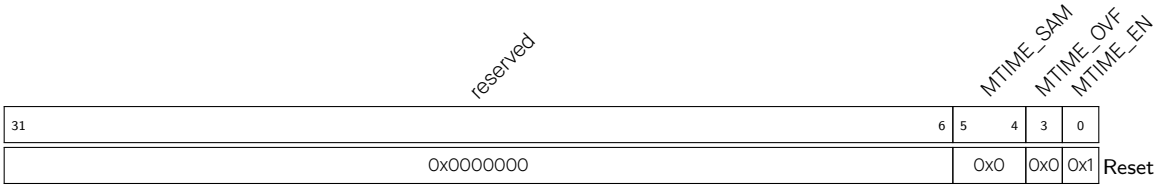

**MTIME_LOADLO** Represents the lower 32 bits to be loaded to the system counter. (R/W)

Register 2.82. mtimeloadhi (0x400C)




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

Register 2.83. mtimectl (0x4010)




**MTIME_EN** Configures whether to enable the system counter.

1: Enable

0: Disable

(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)

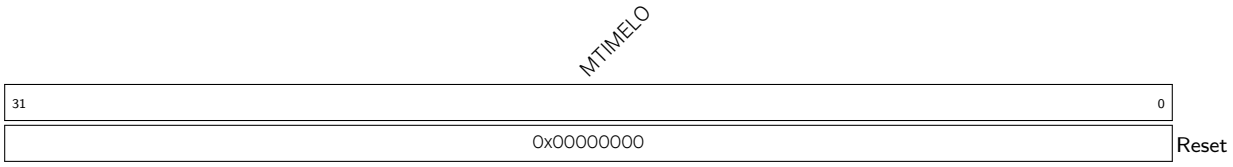
0x1: Sample upper half of the system count on reading MTIMELO

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

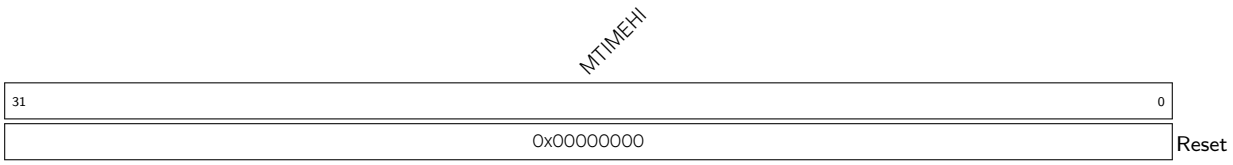
(R/W)

Register 2.84. mtimelo (0xBFF8)




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

Register 2.85. mtimehi (0xBFFC)




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

# 2.9 Memory Protection Unit

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.

## 2.9.1 Standard Physical Memory Protection

### 2.9.1.1 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.

### 2.9.1.2 Features

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

- Up to 16 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

### 2.9.1.3 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 pmpcfg*n* (*n*: 0-3) and pmpaddr*n* (*n*: 0-15) registers (refer to Section 2.9.1.4).

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 2.9.1.4) 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.

#### 2.9.1.4 Register Summary

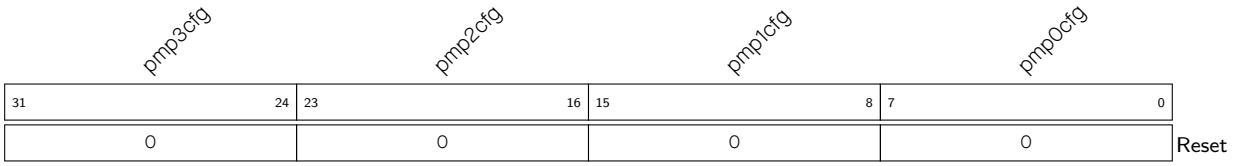
Below is a list of PMP CSRs supported by the CPU. These are only accessible from machine mode.

| 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 |
| pmpaddr*n* (*n*: 0-15) | Physical memory protection address register | 0x3B0+0x1**n* | R/W |

#### 2.9.1.5 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 2.86. pmpcfg0 (0x3A0)**



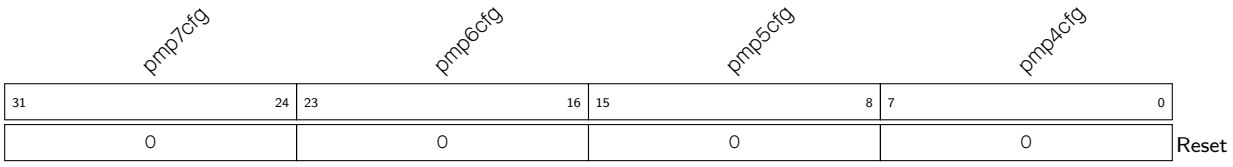

**pmp3cfg** Configuration register for PMP entry 3. (R/W)

**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 2.87. pmpcfg1 (0x3A1)**



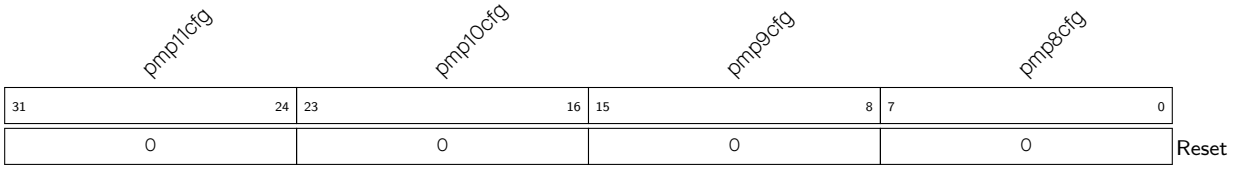

**pmp7cfg** Configuration register for PMP entry 7. (R/W)

**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 2.88. pmpcfg2 (0x3A1)**



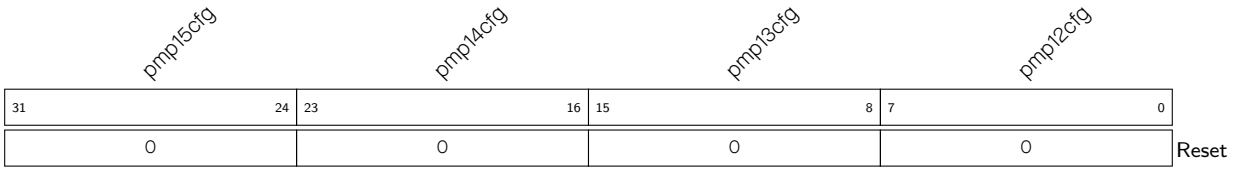

**pmp11cfg** Configuration register for PMP entry 11. (R/W)

**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 2.89. pmpcfg3 (0x3A2)**




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

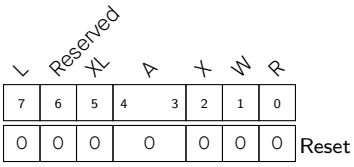
**pmp14cfg** Configuration register for PMP entry 14. (R/W)

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

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

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

Register 2.90. pmpXcfg




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

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

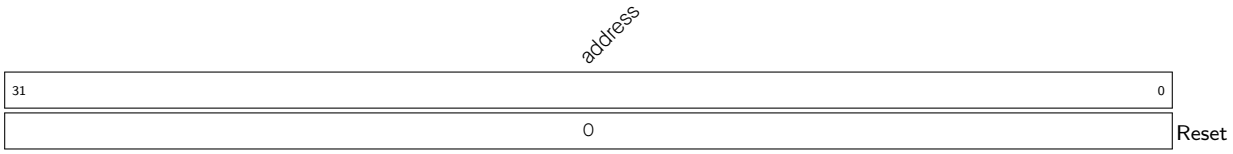
**A** Configures address matching mode.
0: OFF
1: TOR (Top of range)
2: Not Supported
3: NAPOT (Natullary aligned power-of-two region >= 128 Bytes)
(R/W)

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

Register 2.91. pmpaddr*n* (*n*: 0-15) (0x3B0+0x1**n*)




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

## 2.9.2 Custom Physical Memory Attribute (PMA) Checker

### 2.9.2.1 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.

### 2.9.2.2 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

### 2.9.2.3 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 2.9.2.4). 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.

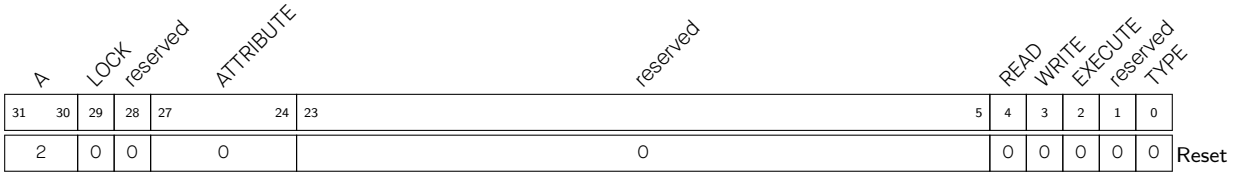
#### 2.9.2.4 Register Summary

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 |

#### 2.9.2.5 Register Description

Register 2.92. 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 2.92. 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)

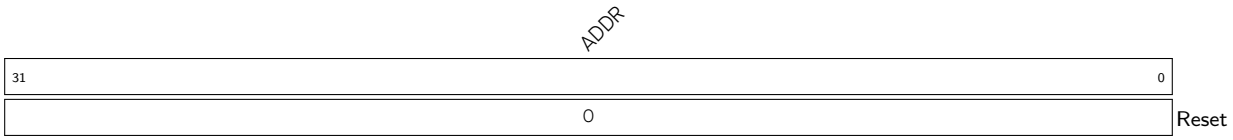
**TYPE** Configures region type.

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




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

# 2.10 Debug and Trace Support

## 2.10.1 Debug

### 2.10.1.1 Overview

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

HP core has its own JTAG DTM (debug transport module) in order to connect with its Debug Module.

Figure 2.10-1 below shows the main components of External Debug Support.

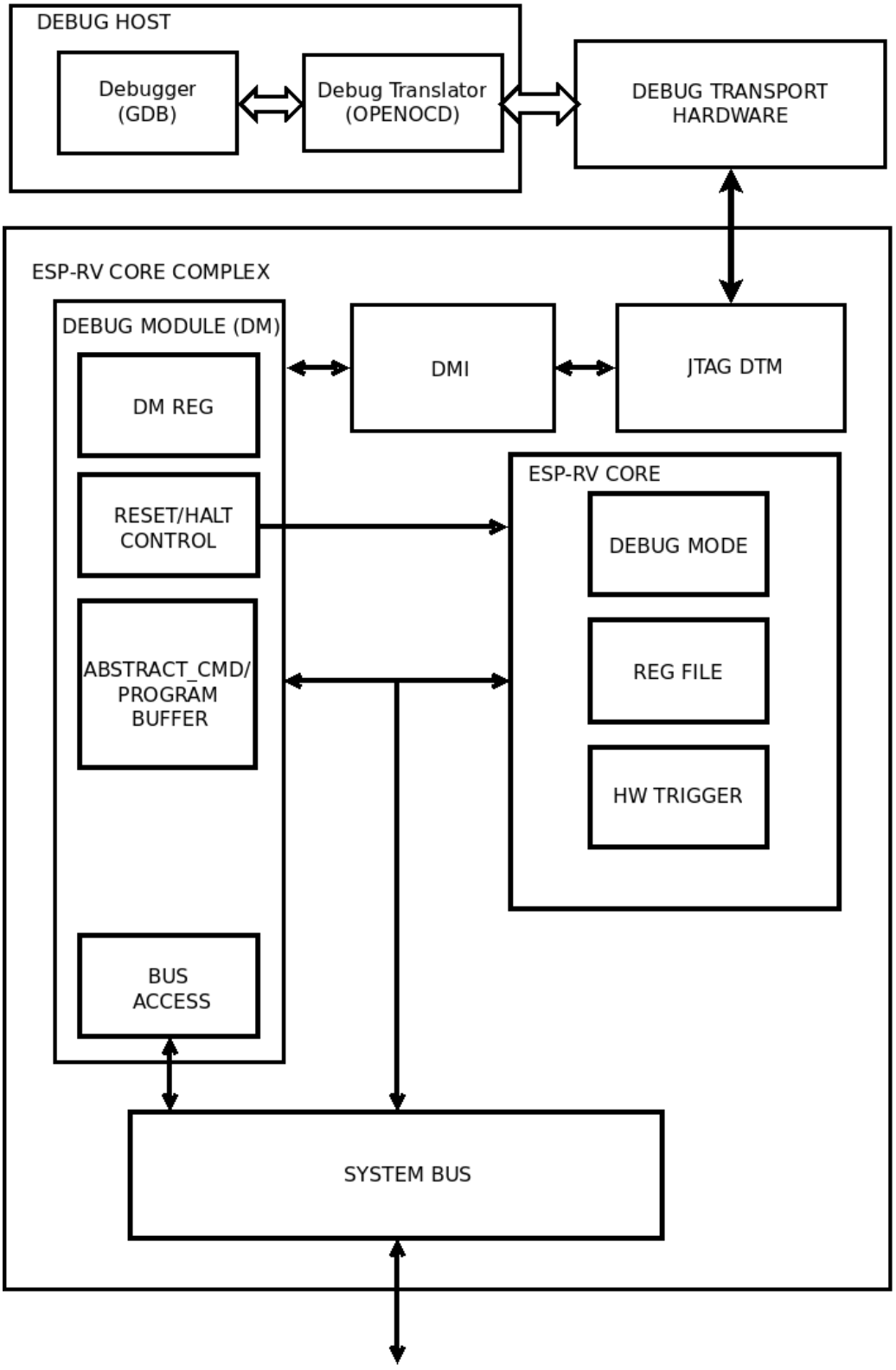



Figure 2.10-1. Debug System Overview

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 HP core's Debug Transport Module (DTM) through standard JTAG interface. The DTM provides access to the Debug Module (DM) using the Debug Module Interface (DMI).

The DM allows the debugger to halt the HP core. 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.

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.

#### 2.10.1.2 Features

HP CPU supports the following basic debugging features:

- 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) as described in Section 2.10.2

#### 2.10.1.3 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.

#### 2.10.1.4 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.

| EFUSE_DIS_USB_JTAG 3 | EFUSE_DIS_USB_SERIAL_JTAG 3 | EFUSE_DIS_PAD_JTAG 3 | EFUSE_JTAG_SEL_ENABLE 3 | Strapping Pin GPIO7 4 | USB JTAG Status | PAD JTAG Status |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | x 2 | Available 1 | Unavailable 1 |
| 0 | 0 | 0 | 1 | 1 | Available | Unavailable |
| 0 | 0 | 0 | 1 | 0 | Unavailable | Available |
| 0 | 1 | 0 | x | x | Unavailable | Available |
| 1 | 0 | 0 | x | x | Unavailable | Available |
| 1 | 1 | 0 | x | x | Unavailable | Available |
| 0 | 0 | 1 | x | x | Available | Unavailable |
| 0 | 1 | 1 | x | x | Unavailable | Unavailable |
| 1 | 0 | 1 | x | x | Unavailable | Unavailable |
| 1 | 1 | 1 | 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. Please refer to Chapter eFuse Controller to get more information about eFuse.
4. Please refer to Chip Boot Control to get more information about the strapping pin GPIO7.

#### 2.10.1.5 Register Summary

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

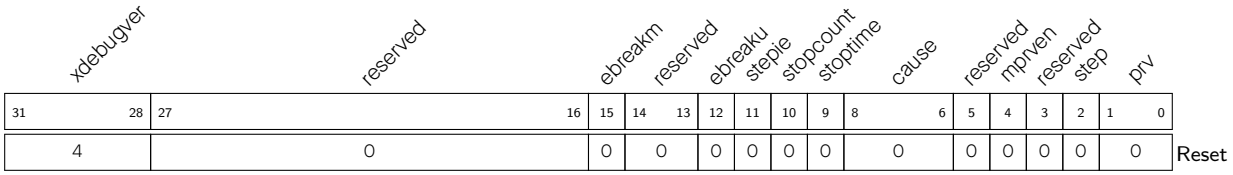
| Name | Description | Address | Access |
|---|---|---|---|
| dcsr | Debug Control and Status Register | 0x7B0 | R/W |
| dpc | Debug PC Register | 0x7B1 | R/W |
| dscratch0 | Debug Scratch Register 0 | 0x7B2 | R/W |
| dscratch1 | Debug Scratch Register 1 | 0x7B3 | R/W |

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.

#### 2.10.1.6 Register Description

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

Register 2.94. 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...**

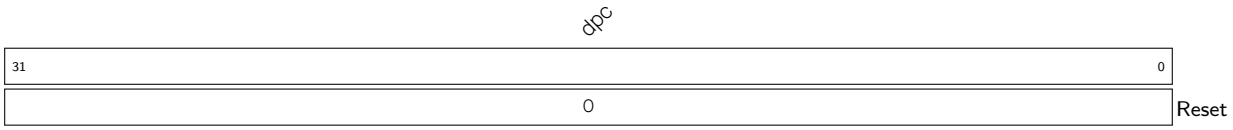
Register 2.94. dcsr (0x7B0)

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 2.95. dpc (0x7B1)

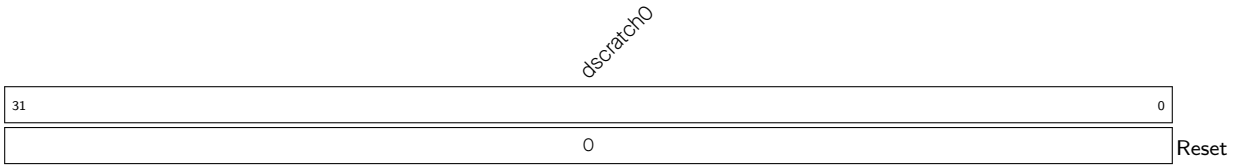



**dpc** Upon entry to debug mode, dpc is written with the virtual address of the next instruction to be executed. When resuming, the CPU core's PC is updated to the virtual address stored in dpc. A debugger may write dpc to change where the CPU resumes. (R/W)

Table 2.10-3. 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 2.96. dscratch0 (0x7B2)




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

Register 2.97. dscratch1 (0x7B3)




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

## 2.10.2 Hardware Trigger

### 2.10.2.1 Overview

HP core implements a hardware trigger block to provide breakpoint and watchpoint capability for debugging. It conforms to the RISC-V External Debug Support Version 0.13.2 specification. Please refer to it for functional operation details.

### 2.10.2.2 Features

HP core trigger block support 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

### 2.10.2.3 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 three 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 2.10-4) 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 2.10-4. 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.

#### 2.10.2.4 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.*

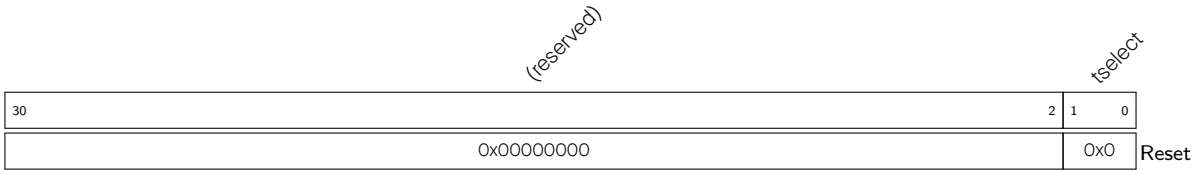
#### 2.10.2.5 Register Summary

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 |

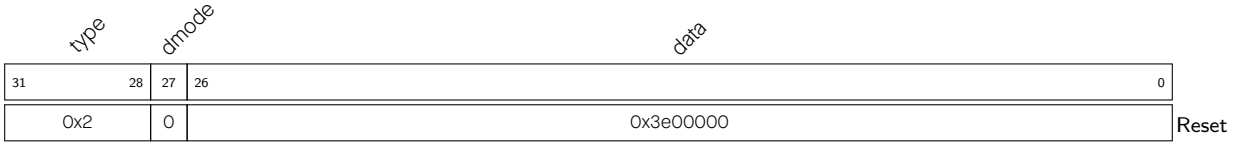
### 2.10.2.6 Register Description

**Register 2.98. tselect (0x7A0)**




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

**Register 2.99. tdata1 (0x7A1)**

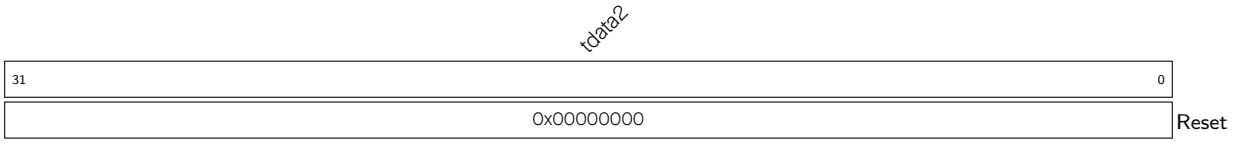



**type** Represents the trigger type. This field is reserved since only match type (0x2) triggers are supported. (RO)

**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)

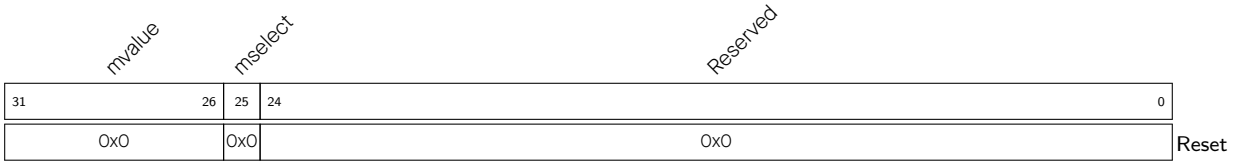
**data** Configures the abstract tdata1 content. This will always be interpreted as fields of mcontrol since only match type (0x2) triggers are supported. (R/W)

Register 2.100. 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 2.101. tdata3 (0x7A3)

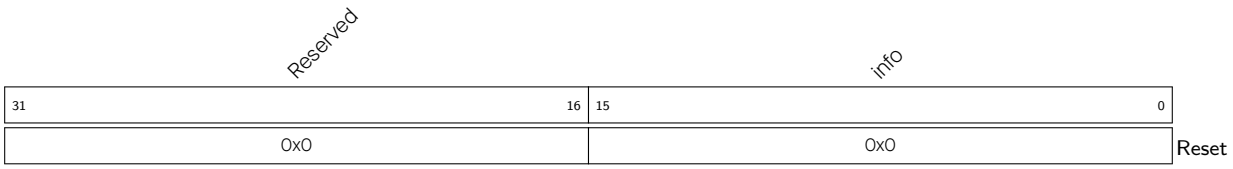



**mvalue** Data used together with mselect. (R/W)

**mselect** Configures whether the mvalue data is used for matching mcontext value.
0: Ignore mvalue.
1: This trigger will only match if the low bits of mcontext equal mvalue.
(R/W)

**Reserved** Read as 0. (RO)

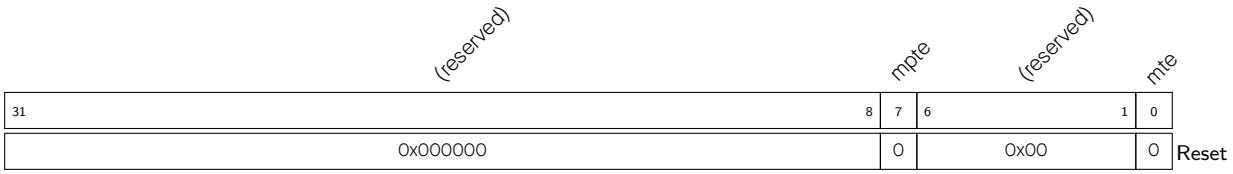
Register 2.102. tinfo (0x7A4)




**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)

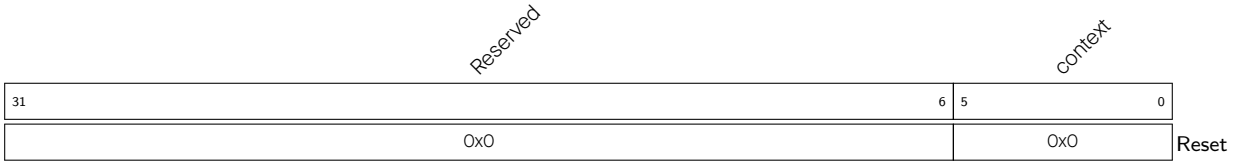
Register 2.103. tcontrol (0x7A5)




**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)

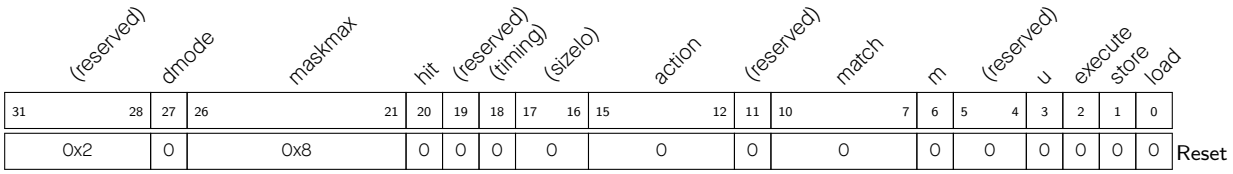
Register 2.104. MCONTEXT (0x7A8)




**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 2.105. 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 2.105. mcontrol (0x7A1)

Continued from the previous page...

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

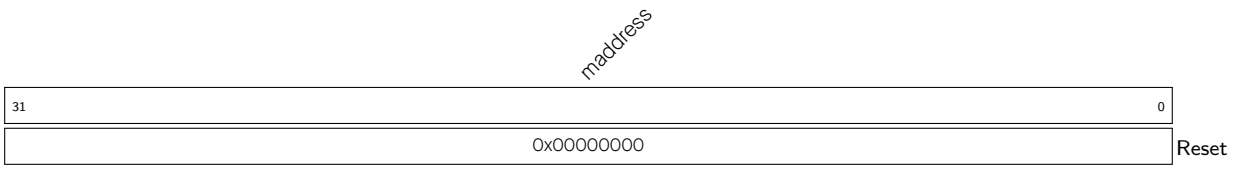
**u** Set this field to enable the selected trigger to operate in user mode. (R/W)

**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 2.106. 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)

## 2.10.3 Trace

### 2.10.3.1 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.

### 2.10.3.2 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.

### 2.10.3.3 Functional Description

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.

# 2.11 Performance

## 2.11.1 Branch Prediction

### 2.11.1.1 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 corresponding PC. Then, it validates the prediction on the execution of the instruction. If the predication turns out to be incorrect, the core flushes the pipeline and start fetching from correct PC. Branch prediction improves the throughput by allowing the core to keep fetching instructions without waiting for the completion of earlier branch instruction. The details about the prediction technique used by the core are described in the following section.

### 2.11.1.2 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

### 2.11.1.3 Functional Description

The core uses Branch History Table (BHT) along with Branch Target Buffer (BTB) for prediction of branch and jump type instructions. A 512x16-bit SRAM memory used for maintaining Branch History Table for HP core. During prediction, it is indexed by Virtual Branch History Register (VBHR). During update, after branch resolves, the BHT is indexed using Global Branch History Register (GBHR), which records the outcomes of the last 13 executed branch instructions. 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 2.11-1, 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 prediction result counter is in any not-taken state, the branch is predicted as not taken. Similarly, if the prediction result counter is in any taken state, the branch is predicted as taken. When the actual branch

outcome is known, the BHT entry is updated based on the following 2-bit saturating counter transition logic.

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 2.11-1 shows 2-bit saturation scheme details used in HP core.

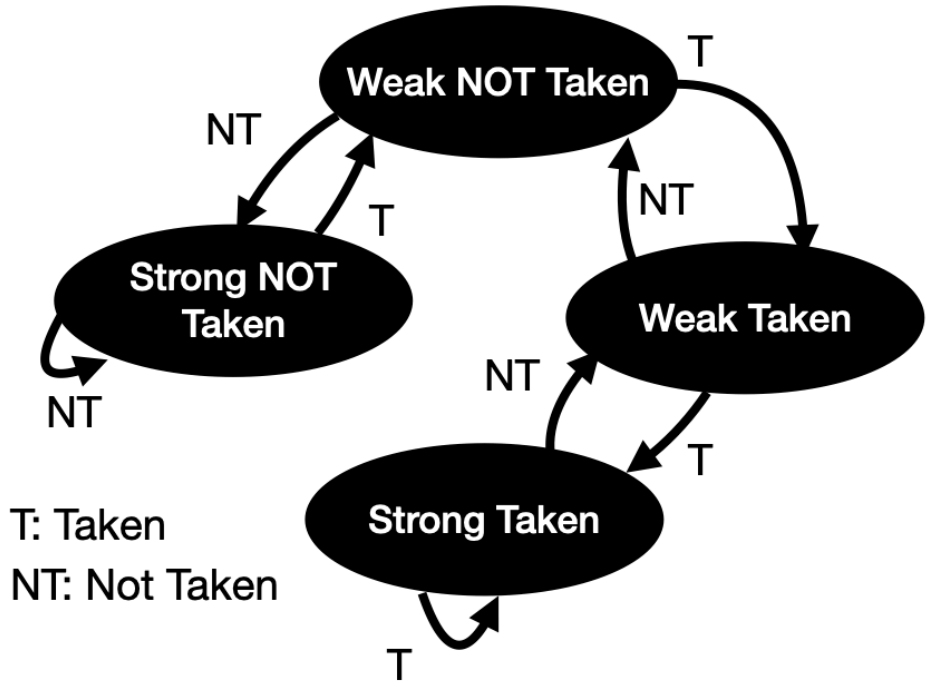



Figure 2.11-1. Two-bit Saturation Counter

## 2.11.2 RAS

### 2.11.2.1 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.

### 2.11.2.2 Features

- Increased core performance by avoiding pipeline stall on return of function call
- Able to store up to four levels of function call nesting

#### 2.11.2.3 Functional Description

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

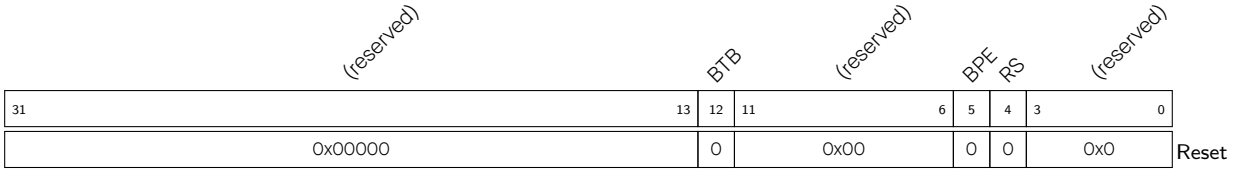
The address is popped out from RAS when the return type instruction (JALR, C.JR, C.JALR) is pre-decoded by the core. The core then starts fetching from the popped address. On execution of the return instruction, if the popped address does not match the address stored in the link register, then the core 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 |

### 2.11.3 Control Status Register for Performance Configuration

**Register 2.107. 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)

# 2.12 Custom Features

## 2.12.1 Bus Error Response

### 2.12.1.1 Overview

In the HP core, load/store bus errors are handled as asynchronous exceptions. As a result, the CPU may have executed some instructions following the load/store instruction that caused the exception before it is detected.

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.

### 2.12.1.2 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.

#### 2.12.1.3 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 |

## 2.12.2 Dedicated IO

### 2.12.2.1 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. These registers are single-instruction accessible, allowing rapid control of the IO pads.

### 2.12.2.2 Features

- 8 dedicated IOs directly mapped on GPIOs
- No latency for driving output ports
- Two CPU cycle latency for sensing input values

### 2.12.2.3 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 *7 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.

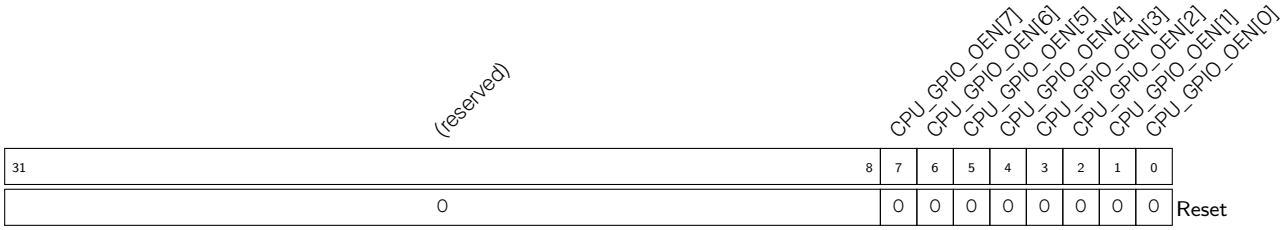
#### 2.12.2.4 Register Summary

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 |

#### 2.12.2.5 Register Description

Register 2.108. cpu_gpio_oen (0x803)



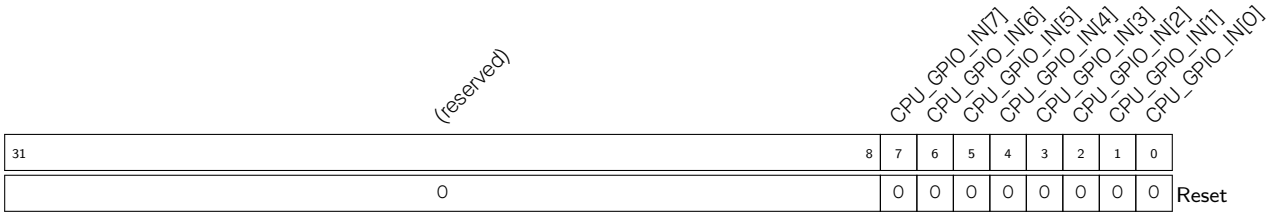

**CPU_GPIO_OEN** Configures whether to enable GPIOn (n=0 ~ 7) output. CPU_GPIO_OEN[7:0] correspond to output enable signals cpu_gpio_out_oen[7:0] in Table 7.12-1 *Peripheral Signals via GPIO Matrix*. CPU_GPIO_OEN value matches that of cpu_gpio_out_oen. CPU_GPIO_OEN is the enable signal of CPU_GPIO_OUT.

0: Disable GPIO output

1: Enable GPIO output

(R/W)

Register 2.109. cpu_gpio_in (0x804)



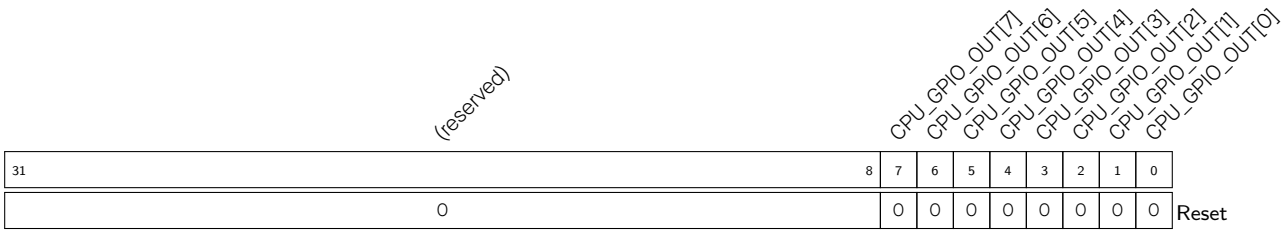

**CPU_GPIO_IN** Represents GPIOn (n=0 ~ 7) input value. It is a CPU CSR to read input value (1=high, 0=low) from SoC GPIO pin.

CPU_GPIO_IN[7:0] correspond to input signals cpu_gpio_in[7:0] in Table 7.12-1 *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 7.4 in Chapter *GPIO Matrix and IO MUX*.

(RO)

Register 2.110. cpu_gpio_out (0x805)




**CPU_GPIO_OUT** Configures GPIOn (n=0 ~ 7) 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 7.12-1 *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 7.5 in Chapter *GPIO Matrix and IO MUX*.

(R/W)

## 2.12.3 RunStall Support

### 2.12.3.1 Overview

The RunStall signal allows an external agent to stall the processor and shut off most 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.

### 2.12.3.2 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 2.12-1 shows the timing diagram for RunStall and CoreStalled signals and CPU core state transition.

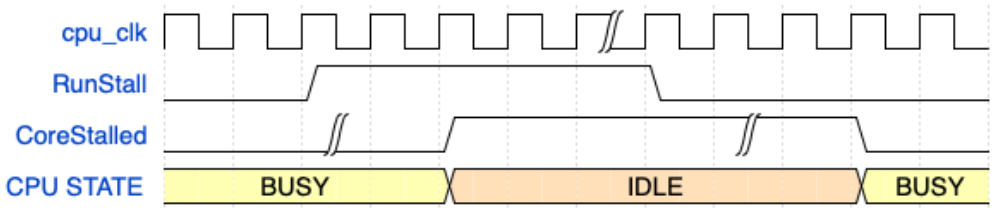



Figure 2.12-1. CPU Core State Transition Using RunStall

### 2.12.3.3 Register Summary

The TRACE_STALL_ENA register field in Chapter 3 *RISC-V Trace Encoder (TRACE)* can be used to assert RunStall to HP core.

## 2.12.4 Debug Assist Information

The HP CPU core complex provides additional debug information to the Debug Assistant module in ESP32-C61. For details, please refer to Chapter 19 *Debug Assistant*.

## 2.12.5 Core Lock-up

#### 2.12.5.1 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.

#### 2.12.5.2 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.

#### 2.12.5.3 Register Summary

The mexstatus.LOCKUP bit provides lock-up status of individual core.

#### 2.12.5.4 Register Description

Please refer to mexstatus CSR.

# Chapter 3

# RISC-V Trace Encoder (TRACE)

The high-performance CPU (HP CPU) of ESP32-C61 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.

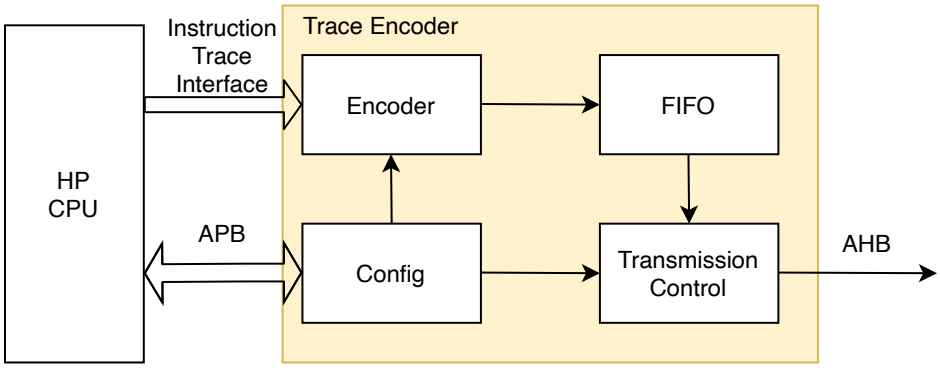



Figure 3.0-1. Trace Encoder Overview

## 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 a trace of the program execution. It works by tracking execution from a known start address and sending messages about the address deltas taken by the program. These deltas are typically introduced by jump, call, return, and branch type instructions, although interrupts and exceptions are also types of deltas.

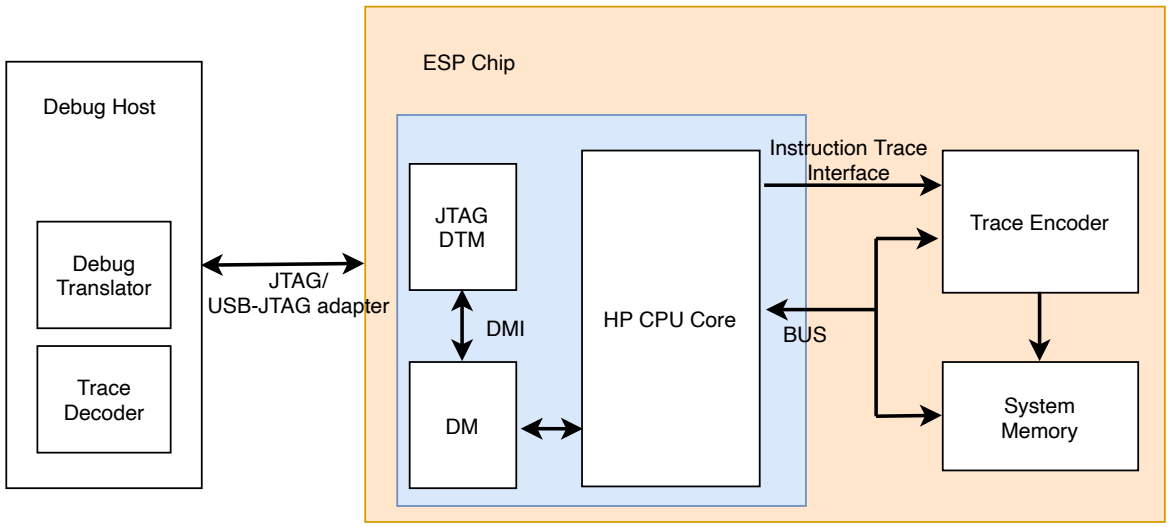



Figure 3.2-1. Trace Flow Overview

Figure 3.2-1 shows the instruction delta trace flow.

- 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-C61 HP CPU's instruction trace interface, please refer to Chapter *2 ESP-RISC-V 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 stop 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 trace 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-C61'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, resulting in packet loss. When a packet is lost, the encoder will send a packet to indicate this event. Thereafter, the encoder will stop working till FIFO gets empty. An interrupt will also be generated to indicate this event if it has been 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-C61 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-C61 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.<br>• A pulse on bit 1 will cause the encoder to stop tracing until further notice.<br>• A pulse on bit 2 will cause the encoder to report a specific address.<br>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 the 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 signal unit if required, and offers:

- Input selected from instruction address (iaddress) and trap value (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:

- 1 run-time selectable comparator units to match
- Selection for specific privilege level (priv) and exception cause (ecause) as 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-C61 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-C61 trace encoder output packet format.

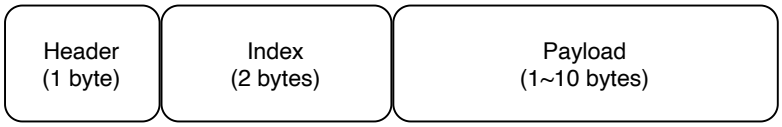



**Figure 3.6-1. Trace packet Format**

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 | Bit Width | 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 | Bit Width | Description | Value |
|---|---|---|---|
| index | 16 | The index of each packet | 0~65536 |

### 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-C61 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 the expiry of the synchronization timer. The payload length is 5 bytes.

Table 3.6-3. Packet format 3 subformat 0

| Field name | Bit Width | 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 | Bit Width | 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 | Bit Width | 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:<br>• 00 (no_change): No change to filter qualification<br>• 01 (ended_rep): Qualification ended, preceding instruction sent explicitly to indicate last qualification instruction<br>• 10 (trace lost): One or more packets lost<br>• 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 | Bit Width | Description |
|---|---|---|
| ioptions | 6 | Indicates optional modes:<br>• 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)<br>• 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)<br>• 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)<br>• full address (Bit 2):<br>– 0: delta address mode<br>– 1: full address mode<br>• 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)<br>• 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 | Bit Width | 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 | Bit Width | 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 | Bit Width | 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:<br>• 0: (cannot occur for this format)<br>• 1: 1 bit<br>• 2 ~ 3: 3 bits<br>• 4 ~ 7: 7 bits<br>• 8 ~ 15: 15 bits<br>• 16 ~ 31: 31 bits<br>For example if branches = 12, branch_map is 15-bit long, and the 12 LSBs are valid. |

| Field name | Bit Width | 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:<br>• 0: branch taken<br>• 1: branch not taken<br>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:<br>• 1: 6 bits<br>• 2 ~ 3: 4 bits<br>• 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 | Bit Width | 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:<br>• 0: branch taken<br>• 1: branch not taken |
| sign_extend | 2 | Reserved |

## 3.7 Interrupt

ESP32-C61's TRACE (i.e., the trace encoder of HP CPU) can generate the TRACE_INTR interrupt signal that will be sent to the Interrupt Matrix.

There are several internal interrupt sources from TRACE that can generate the TRACE_INTR interrupt signal. The interrupt sources from TRACE are listed in Table 3.7-1 with their trigger conditions and the resulted interrupt signal in Table 3.7-1.

Table 3.7-1. 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 | TRACE_INTR |
| TRACE_FIFO_OVERFLOW_INTR | Triggered when the internal FIFO overflows and one or more packets have been lost. | TRACE_INTR |

**Note:**
For definitions of *interrupt*, *interrupt signal*, *interrupt source*, and their correlations, please refer to Chapter 10 *Interrupt Matrix* > Section 10.2 *Interrupt Terminology in ESP32-C61*.

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

By default, trace packets are sent to memory with deltas address until the trace FIFO overflows or the trace memory becomes full. There are some optional configuration options that can change the trace behavior.

- 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 will automatically stop if TRACE_RESTART_ENA is set. When the FIFO is empty, the encoder will restart to report packets.
- 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 HP CPU 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-C61 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 *ESP-RISC-V 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. For details about notify, please refer to Section 3.8.5.
  - 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 0x40800000 ~ 0x4085FFFF
- 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 by configuring Trace-on trigger action 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 stop producing trace packets:

1. Set TRACE_TRIGGER_OFF of TRACE_TRIGGER_REG to end the encoder
2. Through the Debug module by configuring Trace-off trigger action to stop the encoder. When the trigger is matched and the action is 3, it will disable 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:<br>• 0: idle state, the encoder is not started<br>• 1: the encoder is normally working that output packets<br>• 2: the encoder is not outputting packets, the CPU is halted or in reset<br>• 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-C61 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 5.3-2 in Chapter 5 *System and Memory*.

The abbreviations given in Column **Access** are explained in Section *Access Types for Registers*.

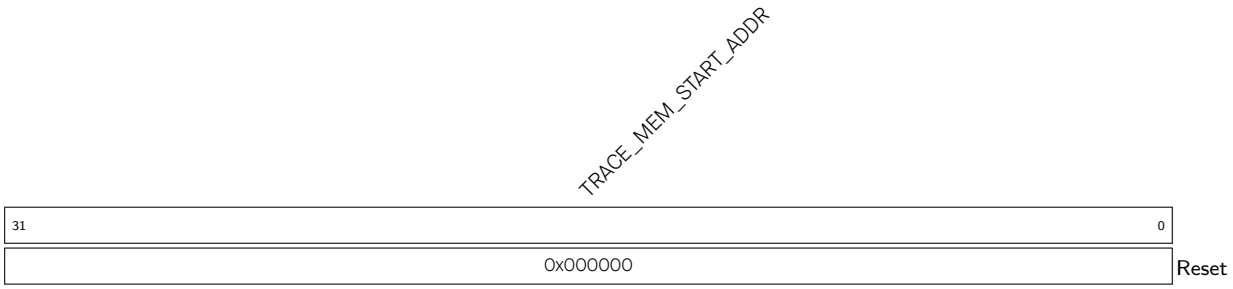
| Name | Description | Address | Access |
|---|---|---|---|
| **Memory configuration registers** | | | |
| TRACE_MEM_START_ADDR_REG | Memory start address | 0x0000 | R/W |
| TRACE_MEM_END_ADDR_REG | Memory end address | 0x0004 | R/W |
| TRACE_MEM_CURRENT_ADDR_REG | Memory current address | 0x0008 | RO |
| TRACE_MEM_ADDR_UPDATE_REG | Memory address update | 0x000C | WT |
| **Trace 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 gate control and configuration register** | | | |
| TRACE_CLOCK_GATE_REG | Clock gating control register | 0x0044 | R/W |
| **Version register** | | | |
| TRACE_DATE_REG | Version control register | 0x03FC | R/W |

## 3.10 Registers

The addresses in this section are relative to RISC-V Trace Encoder base address provided in Table 5.3-2 in Chapter 5 *System and Memory*.

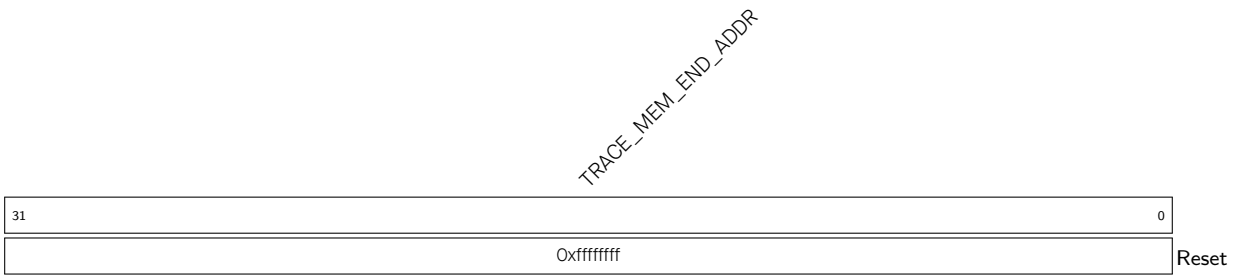
For how to program reserved fields, please refer to Section Programming Reserved Register Field.

**Register 3.1. TRACE_MEM_START_ADDR_REG (0x0000)**



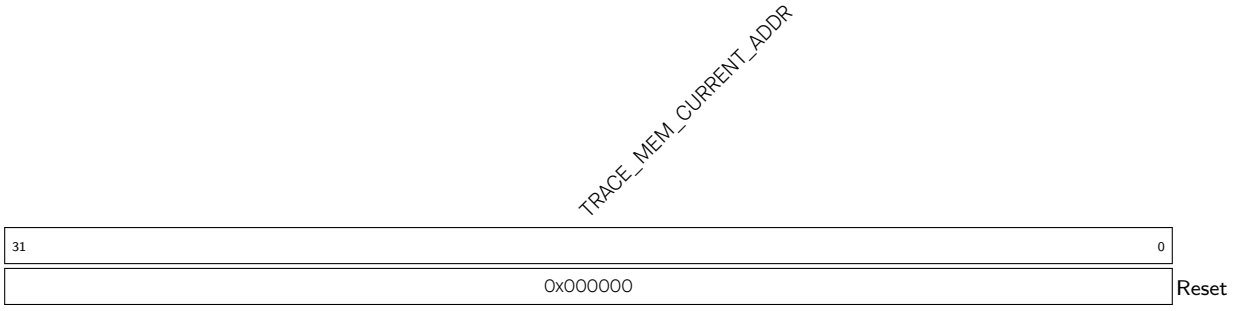

**TRACE_MEM_START_ADDR** Configures the start address of the trace memory. (R/W)

**Register 3.2. TRACE_MEM_END_ADDR_REG (0x0004)**



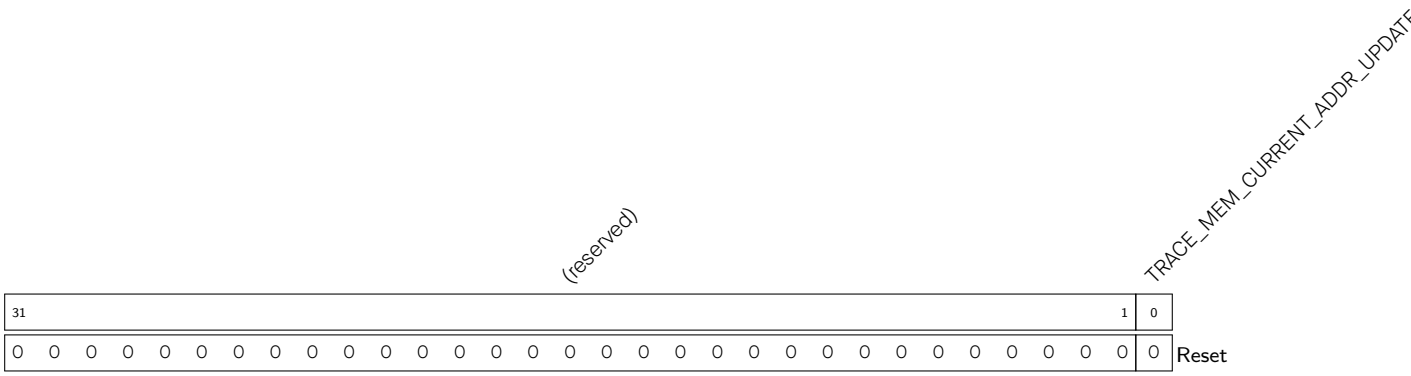

**TRACE_MEM_END_ADDR** Configures the end address of the trace memory. (R/W)

**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)



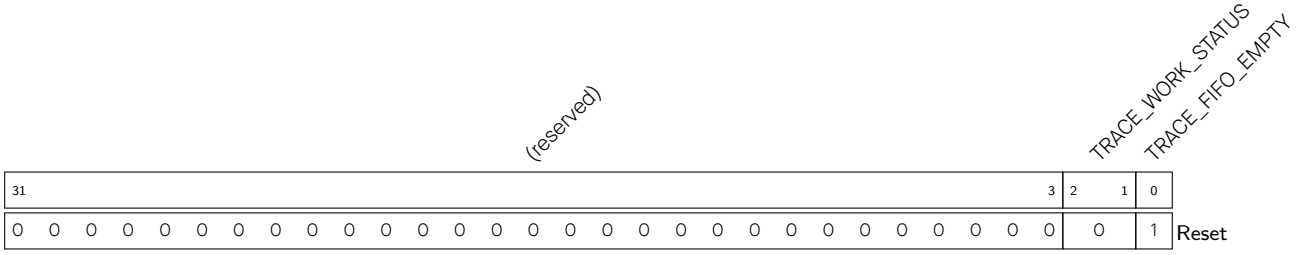

**TRACE_MEM_CURRENT_ADDR_UPDATE** Configures whether to update the value of TRACE_MEM_CURRENT_ADDR to TRACE_MEM_START_ADDR.

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

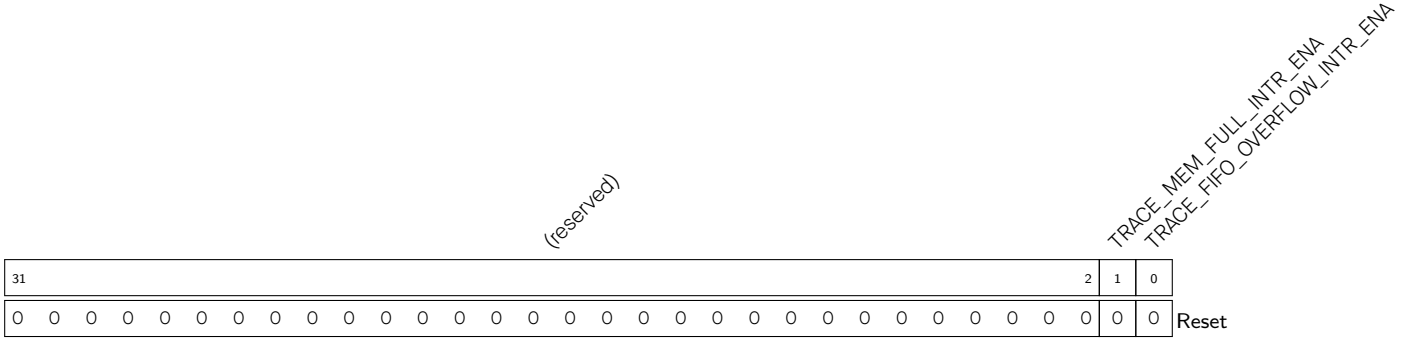
1: Working state

2: Wait state because the hart is halted or in reset

3: Lost state

(RO)

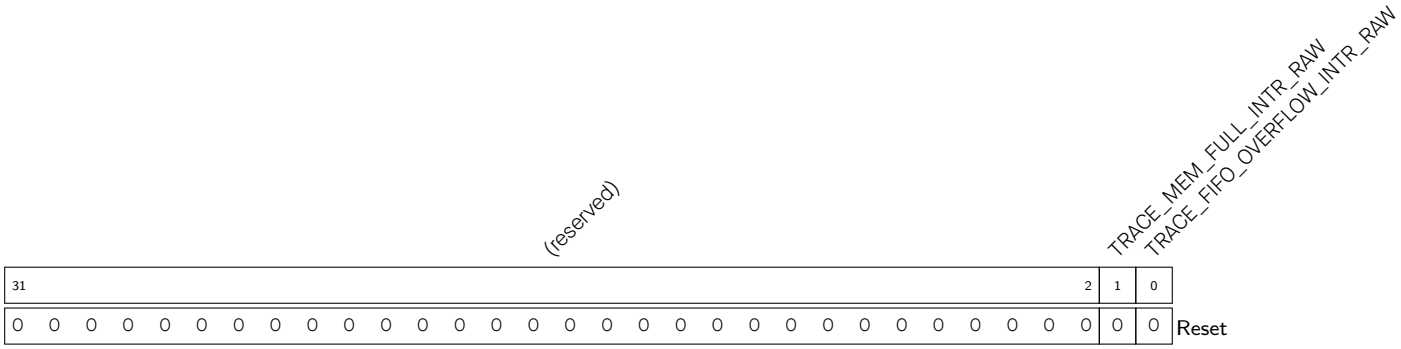
Register 3.6. TRACE_INTR_ENA_REG (0x0014)




**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)

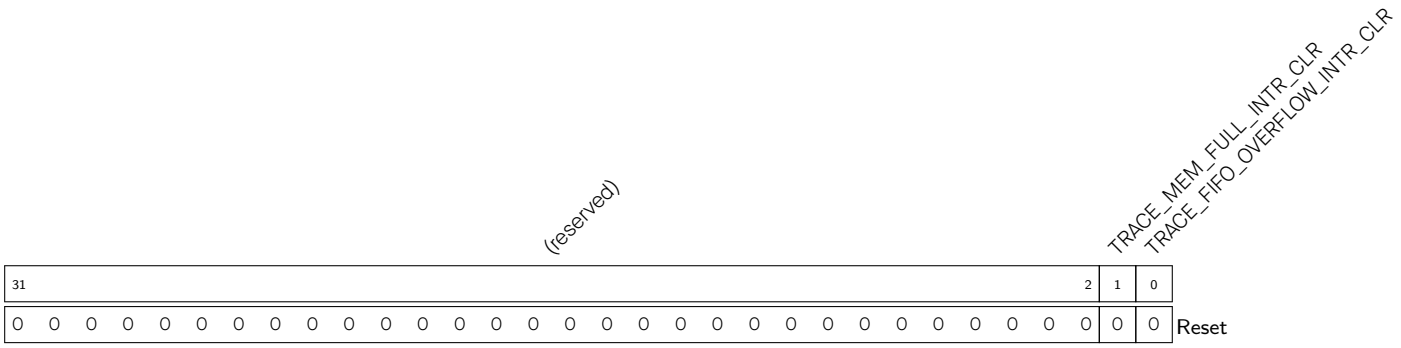
Register 3.7. TRACE_INTR_RAW_REG (0x0018)




**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)

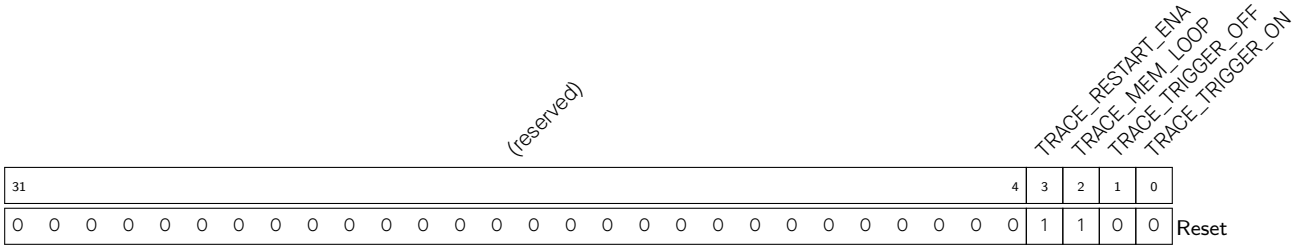
Register 3.8. TRACE_INTR_CLR_REG (0x001C)




**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)

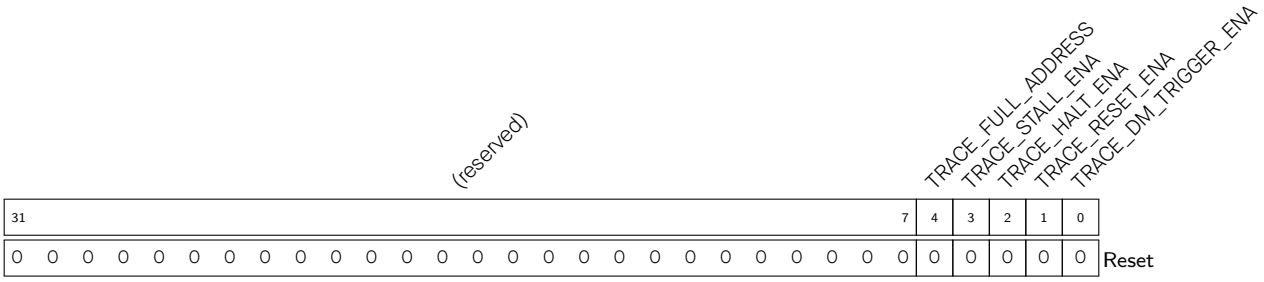
**TRACE_RESTART_ENA** Configures whether to enable automatic restart.

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

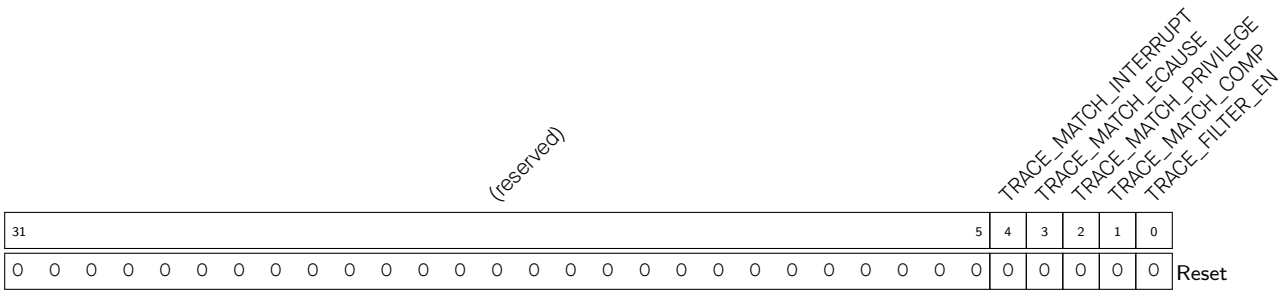
(R/W)

**TRACE_FULL_ADDRESS** Configure the address mode.

0: Delta address mode

1: Full address mode

**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)

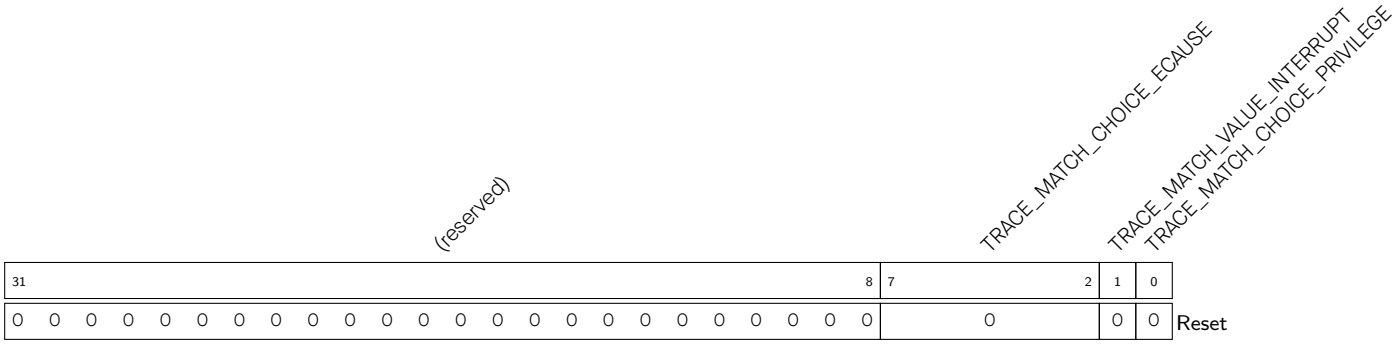
**TRACE_MATCH_INTERRUPT** Configures whether to enable the interrupt match mode.

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.

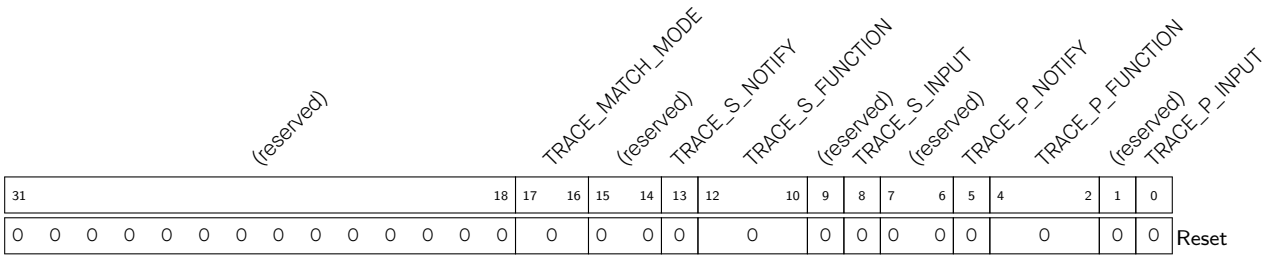
0: itype=1

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)




**TRACE_P_INPUT** Configures the input of the primary comparator for matching.

0: iaddr

1: tval

(R/W)

**TRACE_P_FUNCTION** Configures the function for the primary comparator.

0: Equal

1: Not equal

2: Less than

3: Less than or equal

4: Greater than

5: Greater than or equal

Others: Always match

(R/W)

**TRACE_P_NOTIFY** Configures whether to explicitly report an instruction address matched against the primary comparator.

0: Not report

1: Report

(R/W)

**TRACE_S_INPUT** Configures the input of the secondary comparator for matching.

0: iaddr

1: tval

(R/W)

**TRACE_S_FUNCTION** Configures the function of for secondary comparator.

0: Equal

1: Not equal

2: Less than

3: Less than or equal

4: Greater than

5: Greater than or equal

6: The second comparator value is the masked value of the primary comparator

Others: Always match

(R/W)

**Continued on the next page...**

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

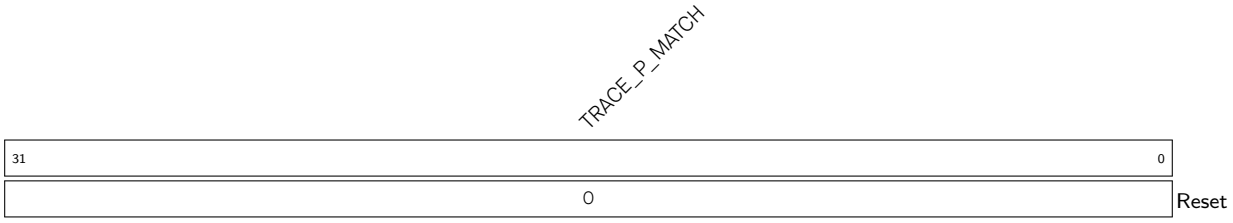
1: Both primary and secondary comparators match (P&&S)

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

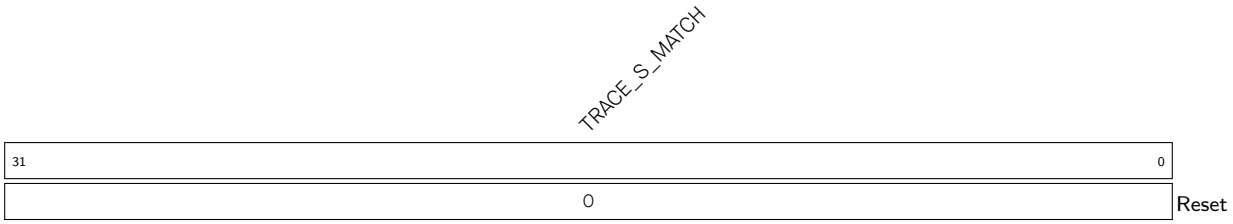
(R/W)

Register 3.14. TRACE_FILTER_P_COMPARATOR_MATCH_REG (0x0034)



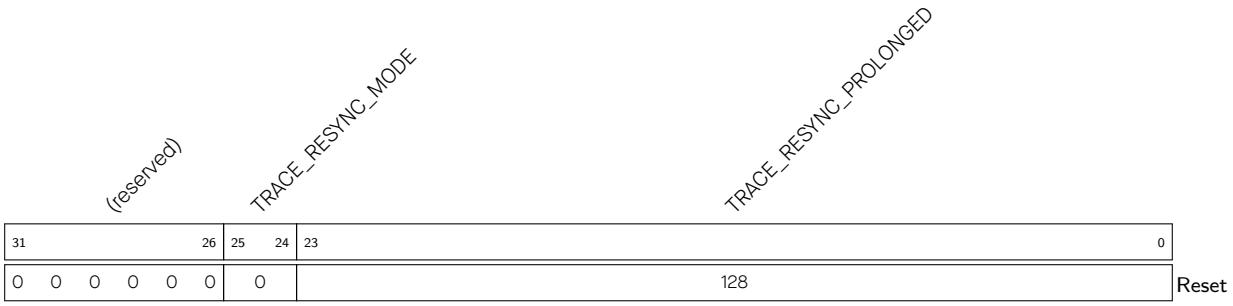

**TRACE_P_MATCH** Configures the match value for the primary comparator. (R/W)

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

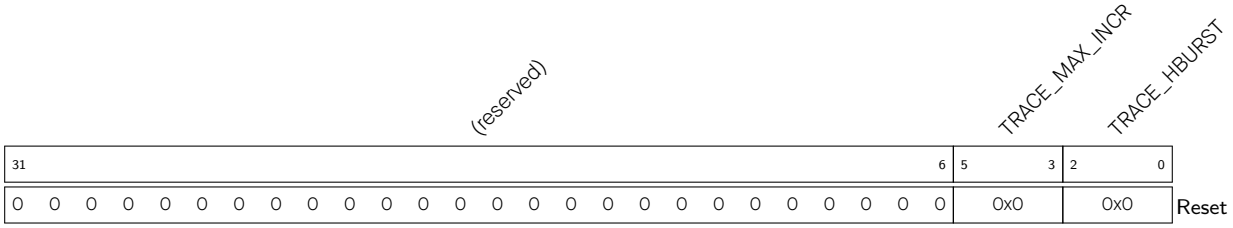
1: Invalid

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

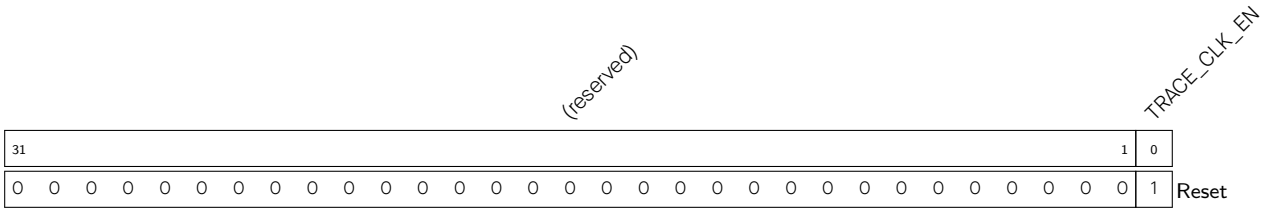
4: INCR8

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)



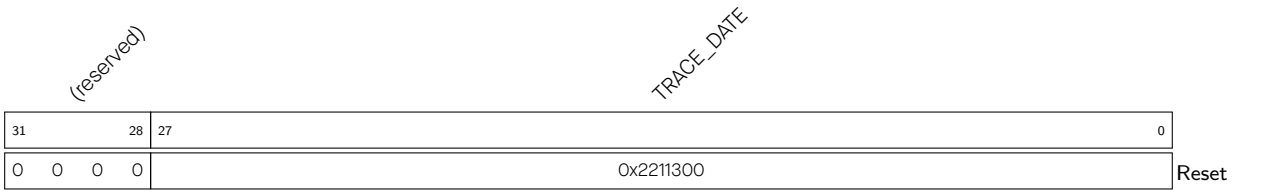

**TRACE_CLK_EN** Configures register clock gating. 0: Support clock only when the application writes registers to save power

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)

# Chapter 4

# GDMA Controller (GDMA)

## 4.1 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.

GDMA has four independent channels: two transmit channels and two receive channels. The GDMA channels are shared and can be assigned to ADC, SHA, I2S or general-purpose SPI (GP-SPI) to access internal or external memory.

GDMA uses configurable priority and weight arbitration schemes to manage peripherals' needs for bandwidth.

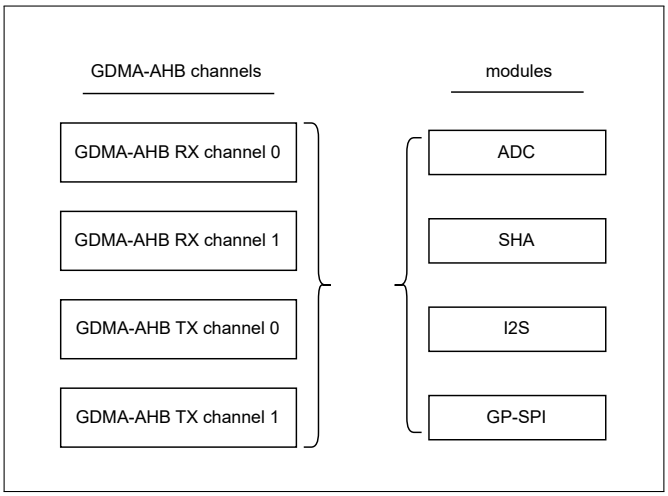



Figure 4.1-1. Modules that Share GDMA Channels

## 4.2 Features

GDMA has the following features:

- AHB bus architecture
- Programmable length of data to be transferred in bytes
- Access via any address and size
- Alignment:
  - Descriptor address: 1-word aligned

  - Data address and length: no requirements
- Linked list of descriptors
- INCR burst transfer when accessing memory
- Two transmit channels and two receive channels
- Software-configurable selection of peripheral requesting its service
- Configurable channel priority and weight arbitration
- Support for memory transfer

## 4.3 Architecture

In ESP32-C61, 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 4.3-1 shows the basic architecture of the GDMA controller.

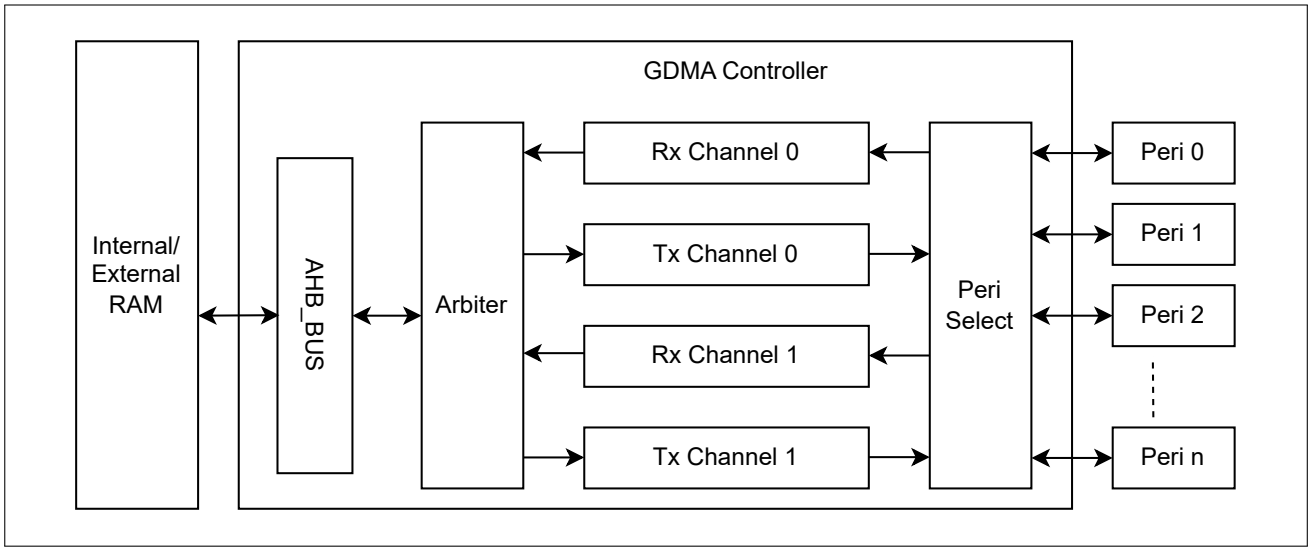



**Figure 4.3-1. GDMA controller Architecture**

GDMA has four independent channels: two transmit channels and two receive channels. Every channel can be connected to different peripherals. In other words, channels are general-purpose, and shared by peripherals.

GDMA reads data from or writes data to memory via AHB_BUS. 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 5 *System and Memory*.

Software can use the GDMA through linked lists, which are stored in the internal memory. These linked lists consist of outlink*n* and inlink*n*, where *n* indicates the channel number (ranging from 0 to 1). The GDMA reads an outlink*n* (i.e., a linked list of transmit descriptors) from memory and transmits data in the corresponding memory according to the outlink*n*, or reads an inlink*n* (i.e., a linked list of receive descriptors) and stores received data into specific address space in the memory according to the inlink*n*.

# 4.4 Functional Description

## 4.4.1 Linked List

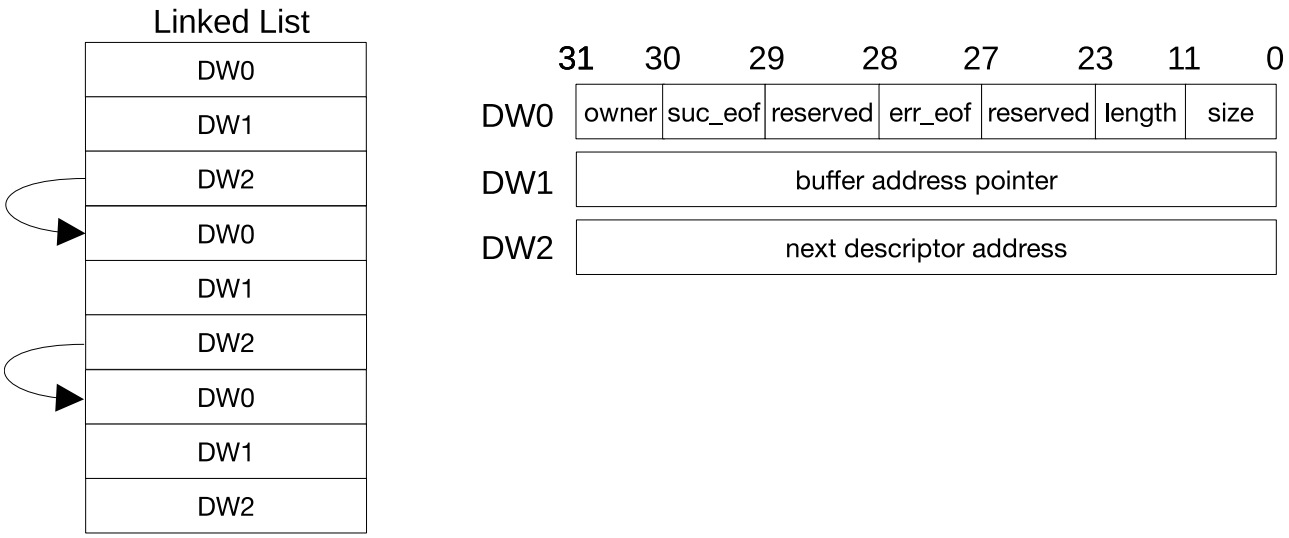



**Figure 4.4-1. Structure of a Linked List**

Figure 4.4-1 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_DMA_OUT_AUTO_WRBACK_CH*n* is set to 1. Software can disable automatic clearing by hardware by setting the AHB_DMA_OUT_LOOP_TEST_CH*n* or AHB_DMA_IN_LOOP_TEST_CH*n* bit. When software loads a linked list, this bit should be set to 1.
  **Note:** AHB_DMA_OUT is the prefix of transmit channel registers, and AHB_DMA_IN is the prefix of receive channel registers.
- suc_eof (DW0) [30]: Specifies whether the AHB_DMA_IN_SUC_EOF_CH*n*_INT or AHB_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.
  GDMA configures this bit in a receive descriptor based on the err_eof provided by the peripheral when writing back the descriptor (Currently, no peripheral on ESP32-C61 supports the err_eof flag).
- 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. This field can only point to the memory space.

> **Note:**
> Please note that addresses of GDMA descriptors should be 4-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. If the length of data received is larger than the size of the buffer, the GDMA controller will use new descriptors to complete the data transfer after the current transactions.

### 4.4.2 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 4.4-1 illustrates 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 4.4-1. GDMA Selecting Peripherals via Register Configuration

| AHB_DMA_PERI_IN_SEL_CH*n*<br>AHB_DMA_PERI_OUT_SEL_CH*n* | Peripheral |
|---|---|
| 0 | Dummy-0 |
| 1 | GP-SPI |
| 2 | Dummy-2 |
| 3 | I2S |
| 4 ~ 6 | Dummy-4 ~ 6 |
| 7 | SHA |
| 8 | ADC |
| 9 ~ 15 | Dummy-9 ~ 15 |
| 16 ~ 63 | Invalid |

### 4.4.3 Memory-to-Memory Data Transfer

The GDMA controller also allows memory-to-memory data transfer. Such data transfer can be enabled by setting AHB_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_DMA_PERI_IN_SEL_CH*n* and AHB_DMA_PERI_OUT_SEL_CH*n* should be configured to the same value corresponding to "Dummy".

### 4.4.4 Enabling GDMA

The software uses the GDMA controller through linked lists. When the GDMA controller receives data, software loads an inlink, configures the AHB_DMA_INLINK_ADDR_CH*n* field with the address of the first receive descriptor, and sets the AHB_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_DMA_OUTLINK_ADDR_CH*n* field with the address of the first transmit descriptor, and sets the AHB_DMA_OUTLINK_START_CH*n* bit to enable GDMA. The AHB_DMA_INLINK_START_CH*n* bit and AHB_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_DMA_INLINK_RESTART_CH*n* bit or AHB_DMA_OUTLINK_RESTART_CH*n* bit (these two bits are cleared automatically by hardware). As shown in Figure 4.4-2, 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.

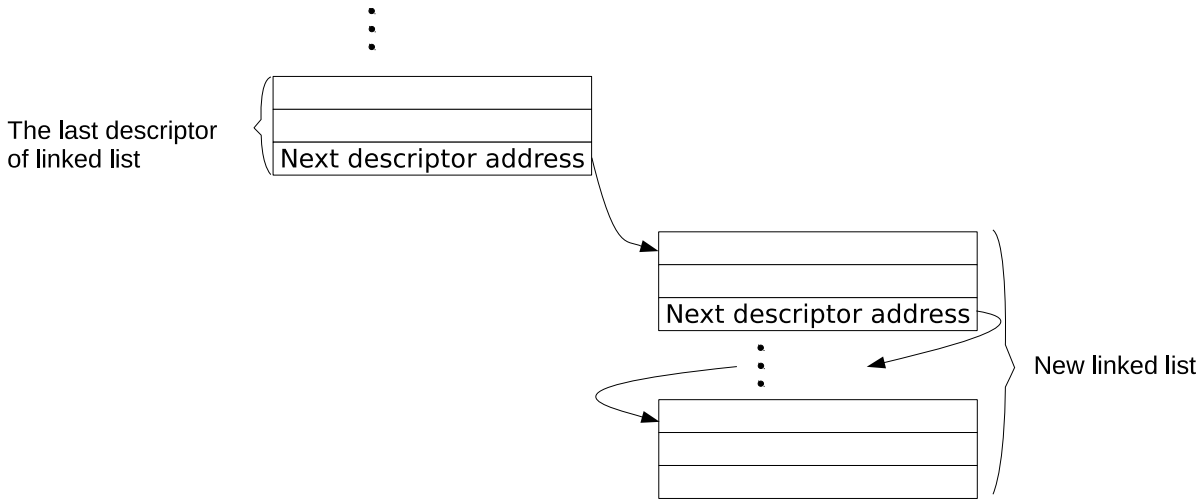



Figure 4.4-2. Relationship among Linked Lists

### 4.4.5 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_DMA_IN_DSCR_ERR_CH*n*_INT or AHB_DMA_OUT_DSCR_ERR_CH*n*_INT), and the channel will halt.

The checks performed on descriptors are:

- Owner bit check when AHB_DMA_IN_CHECK_OWNER_CH*n* or AHB_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_DMA_IN_CHECK_OWNER_CH*n* or AHB_DMA_OUT_CHECK_OWNER_CH*n* is 0.
- Descriptor address check, which checks if the descriptor address is located in configured memory space for GDMA. If a GDMA descriptor points to AHB_DMA_ACCESS_INTR_MEM_START_ADDR ~ AHB_DMA_ACCESS_INTR_MEM_END_ADDR, it passes the check. For details, please refer to Section 4.4.7.

After the software detects a descriptor error interrupt, it must load new descriptors, and enable GDMA by setting AHB_DMA_OUTLINK_START_CH*n* or AHB_DMA_INLINK_START_CH*n* bit.

### 4.4.6 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_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_DMA_OUT_EOF_CH*n*_INT_ENA bit.

- AHB_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_DMA_OUT_TOTAL_EOF_CH*n*_INT_ENA bit.

For data reception, the GDMA generates one type of EOF interrupts:

- AHB_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_DMA_IN_SUC_EOF_CH*n*_INT_ENA bit.

When detecting an AHB_DMA_OUT_TOTAL_EOF_CH*n*_INT or AHB_DMA_IN_SUC_EOF_CH*n*_INT interrupt, software can read the value of the AHB_DMA_OUT_EOF_DES_ADDR_CH*n* or AHB_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.

### 4.4.7 Accessing Memory

Any transmit and receive channels of GDMA can access the memory address space configured by AHB_DMA_ACCESS_INTR_MEM_START_ADDR and AHB_DMA_ACCESS_INTR_MEM_END_ADDR.

To improve data transfer efficiency, GDMA can send data in burst mode. Burst mode is disabled by default, and the burst length can be configured to SINGLE, INCR4, or INCR8 respectively by setting AHB_DMA_IN_DATA_BURST_MODE_SEL_CH*n* and AHB_DMA_OUT_DATA_BURST_MODE_SEL_CH*n* to 0, 1, or 2.

When GDMA accesses the configured memory space, there are no requirements for descriptor field alignment.

### 4.4.8 Arbitration

To ensure timely response to peripherals running at a high speed with low latency (such as general-purpose 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 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.

## 4.5 Event Task Matrix Feature

The GDMA controller on ESP32-C61 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 11 Event Task Matrix (ETM).

GDMA can receive the following ETM tasks:

- GDMA_TASK_IN_START_CH*n*: Enables the corresponding RX channel *n* for data transfer.
- GDMA_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_DMA_INLINK_START_CH*n* and AHB_DMA_OUTLINK_START_CH*n*. When AHB_DMA_IN_ETM_EN_CH*n* or AHB_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_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_EVT_IN_DONE_CH*n*: Indicates that the data has been received according to the receive descriptor via channel *n*.
- GDMA_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_EVT_IN_FIFO_EMPTY_CH*n*: Indicates that the RX FIFO has become empty.
- GDMA_EVT_IN_FIFO_FULL_CH*n*: Indicates that the RX FIFO has become full.
- GDMA_EVT_OUT_DONE_CH*n*: Indicates that the data has been transmitted according to the transmit descriptor via channel *n*.
- GDMA_EVT_OUT_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_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_EVT_OUT_FIFO_EMPTY_CH*n*: Indicates that the TX FIFO has become empty.

- GDMA_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_EVT_OUT_TOTAL_EOF_CH0 event can trigger the GDMA_TASK_IN_START_CH1 task, and in this way trigger a new round of GDMA operations.

## 4.6 Interrupts

ESP32-C61's GDMA module can generate the following interrupt signals that will be sent to the Interrupt Matrix.

- GDMA_IN_CH0_INTR
- GDMA_IN_CH1_INTR
- GDMA_OUT_CH0_INTR
- GDMA_OUT_CH1_INTR

There are several internal interrupt sources from GDMA that can generate the above interrupt signals.

The GDMA_IN_CH*n*_INTR (*n* is 0 ~ 1) signals are generated by the following interrupt sources:

- AHB_DMA_IN_DONE_CH*n*_INT: Triggered when all data corresponding to a receive descriptor has been received via receive channel *n*.
- AHB_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_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_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: Triggered when the RX FIFO of GDMA overflows.
- AHB_DMA_INFIFO_UDF_CH*n*_INT: Triggered when the RX FIFO of GDMA underflows.

The GDMA_OUT_CH*n*_INTR (*n* is 0 ~ 1) signals are generated by the following interrupt sources:

- AHB_DMA_OUT_DONE_CH*n*_INT: Triggered when all data corresponding to a transmit descriptor has been sent via transmit channel *n*.
- AHB_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_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_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_DMA_OUT_DSCR_ERR_CH*n*_INT: Triggered when a transmit descriptor on transmit channel *n* fails any of the two descriptor checks.
- AHB_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: Triggered when the TX FIFO of GDMA overflows.

- AHB_DMA_OUTFIFO_UDF_CH*n*_INT: Triggered when the TX FIFO of GDMA underflows.

## 4.7 Programming Procedures

The clock gating for GDMA can be configured via PCR_GDMA_CLK_EN, and is enabled by default. GDMA can be reset by configuring PCR_GDMA_RST_EN.

### 4.7.1 Programming Procedures for GDMA's Transmit Channel

To transmit data, GDMA's transmit channel should be configured by software as follows:

1. Set AHB_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_DMA_OUTLINK_ADDR_CH*n* with address of the first transmit descriptor.
3. Configure AHB_DMA_PERI_OUT_SEL_CH*n* with the value corresponding to the peripheral to be connected, as shown in Table 4.4-1.
4. Set AHB_DMA_OUTLINK_START_CH*n* to enable GDMA's transmit channel for data transfer.
5. Configure and enable the corresponding peripheral. See details in the individual chapter for the corresponding peripheral.
6. Wait for the AHB_DMA_OUT_TOTAL_EOF_CH*n*_INT interrupt, which indicates the completion of data transfer.

### 4.7.2 Programming Procedures for GDMA's Receive Channel

To receive data, GDMA's receive channel should be configured by software as follows:

1. Set AHB_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_DMA_INLINK_ADDR_CH*n* with address of the first receive descriptor.
3. Configure AHB_DMA_PERI_IN_SEL_CH*n* with the value corresponding to the peripheral to be connected, as shown in Table 4.4-1.
4. Set AHB_DMA_INLINK_START_CH*n* to enable GDMA's receive channel for data transfer.
5. Configure and enable the corresponding peripheral. See details in the individual chapter for the corresponding peripheral.

### 4.7.3 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_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_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_DMA_OUTLINK_ADDR_CH*n* with address of the first transmit descriptor.
4. Load an inlink, and configure AHB_DMA_INLINK_ADDR_CH*n* with address of the first receive descriptor.
5. Set AHB_DMA_MEM_TRANS_EN_CH*n* to enable memory-to-memory transfer.
6. Configure AHB_DMA_PERI_IN_SEL_CH*n* and AHB_DMA_PERI_OUT_SEL_CH*n* to the same "Dummy" value.
7. Set AHB_DMA_OUTLINK_START_CH*n* to enable GDMA's transmit channel for data transfer.
8. Set AHB_DMA_INLINK_START_CH*n* to enable GDMA's receive channel for data transfer.
9. If the suc_eof bit is set in a transmit descriptor, an AHB_DMA_IN_SUC_EOF_CH*n*_INT interrupt will be triggered when the data segment corresponding to this descriptor has been transmitted.

### 4.7.4 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_DMA_TX_PRI_CH*n* and AHB_DMA_RX_PRI_CH*n*.

The weight arbitration can be configured as follows:

1. Configure the time slot for TX and RX respectively via AHB_DMA_ARB_TIMEOUT_TX and AHB_DMA_ARB_TIMEOUT_RX.
2. Configure the number of tokens for TX and RX respectively via AHB_DMA_TX_CH_ARB_WEIGH_CH*n* and AHB_DMA_RX_CH_ARB_WEIGH_CH*n*.
3. Enable weight arbitration optimization for TX and RX respectively by clearing AHB_DMA_TX_ARB_WEIGH_OPT_DIR_CH*n* and AHB_DMA_RX_ARB_WEIGH_OPT_DIR_CH*n*.
4. Enable the arbitration for TX and RX respectively by setting AHB_DMA_WEIGHT_EN_TX and AHB_DMA_WEIGHT_EN_RX.

## 4.8 Register Summary

The addresses in this section are relative to GDMA base address provided in Table 5.3-2 in Chapter 5 *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_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 |
| **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 |

| Name | Description | Address | Access |
|---|---|---|---|
| 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_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_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_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_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 |
| AHB_DMA_IN_LINK_ADDR_CH0_REG | Linked list descriptor configuration register of RX channel 0 | 0x03AC | R/W |

| Name | Description | Address | Access |
|---|---|---|---|
| AHB_DMA_IN_LINK_ADDR_CH1_REG | Linked list descriptor configuration register of RX channel 1 | 0x03B0 | 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_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_TX_REG | TX arbitration timeout configuration register | 0x03CC | R/W |
| AHB_DMA_ARB_TIMEOUT_RX_REG | RX arbitration timeout configuration register | 0x03D0 | R/W |
| AHB_DMA_WEIGHT_EN_TX_REG | TX weight arbitration enable register | 0x03D4 | R/W |
| AHB_DMA_WEIGHT_EN_RX_REG | RX weight arbitration enable register | 0x03D8 | R/W |
| **Version Registers** | | | |
| 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_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 |

GoBack

| Name | Description | Address | Access |
|---|---|---|---|
| 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_IN_DONE_DES_ADDR_CH0_REG | Address of the completed inlink descriptor on RX channel 0 | 0x00B0 | RO |
| AHB_DMA_OUT_DONE_DES_ADDR_CH0_REG | Address of the completed outlink descriptor on TX channel 0 | 0x0110 | 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_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 |
| AHB_DMA_OUT_EOF_DES_ADDR_CH1_REG | Transmit descriptor address when EOF occurs on TX channel 1 | 0x01A8 | RO |

GoBack

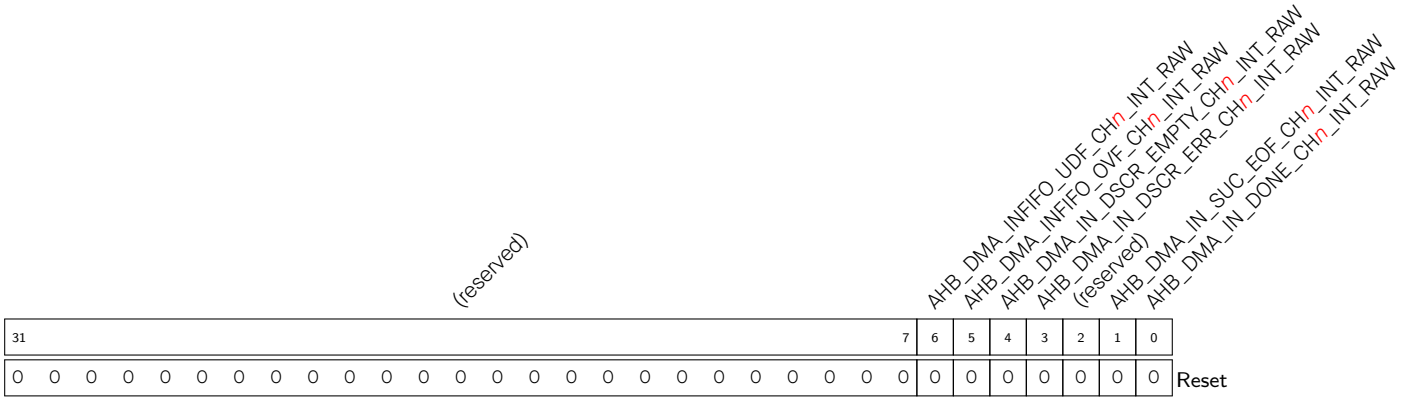
| Name | Description | Address | Access |
|---|---|---|---|
| AHB_DMA_OUT_EOF_BFR_DES_ADDR_CH1_REG | The last transmit descriptor address when EOF occurs on TX channel 1 | 0x01AC | RO |
| AHB_DMA_OUT_DSCR_CH1_REG | Address of the next transmit descriptor pointed by the current pre-read transmit descriptor 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_IN_DONE_DES_ADDR_CH1_REG | Address of the completed inlink descriptor on RX channel 1 | 0x0170 | RO |
| AHB_DMA_OUT_DONE_DES_ADDR_CH1_REG | Address of the completed outlink descriptor on TX channel 1 | 0x01D0 | 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 |
| **Peripheral Selection 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 |

## 4.9 Registers

The addresses in this section are relative to GDMA base address provided in Table 5.3-2 in Chapter 5 *System and Memory*.

For how to program reserved fields, please refer to Section Programming Reserved Register Field.

**Register 4.1. AHB_DMA_IN_INT_RAW_CH*n*_REG (*n*: 0-1) (0x0000+0x10**n*)**




**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. (R/WTC/SS)

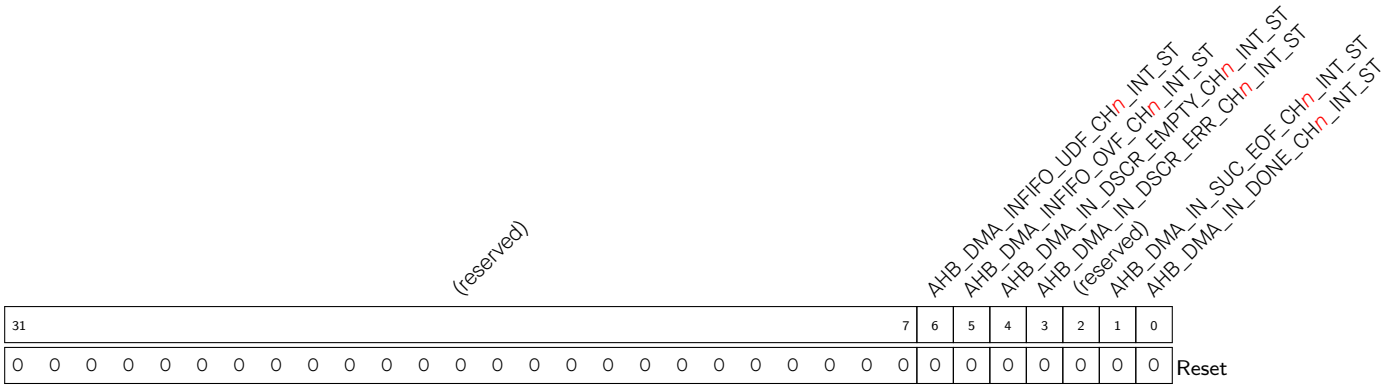
**AHB_DMA_IN_DSCR_ERR_CH*n*_INT_RAW** The raw interrupt status of AHB_DMA_IN_DSCR_ERR_CH*n*_INT. (R/WTC/SS)

**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)

**Register 4.2. AHB_DMA_IN_INT_ST_CH*n*_REG (*n*: 0-1) (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)

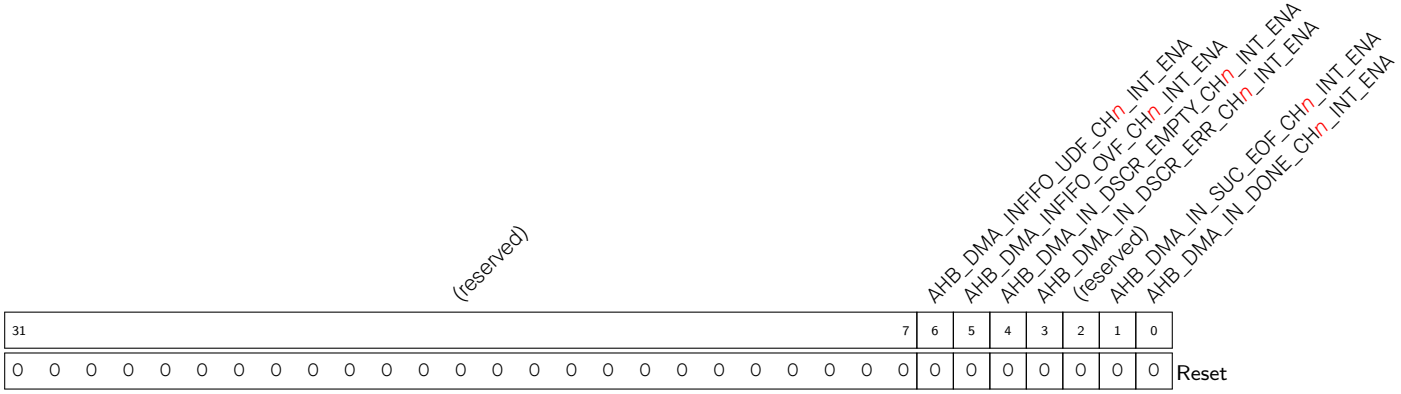
**AHB_DMA_IN_DSCR_ERR_CH*n*_INT_ST** The masked interrupt status of AHB_DMA_IN_DSCR_ERR_CH*n*_INT. (RO)

**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)

**Register 4.3. AHB_DMA_IN_INT_ENA_CH*n*_REG (*n*: 0-1) (0x0008+0x10**n*)**




**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)

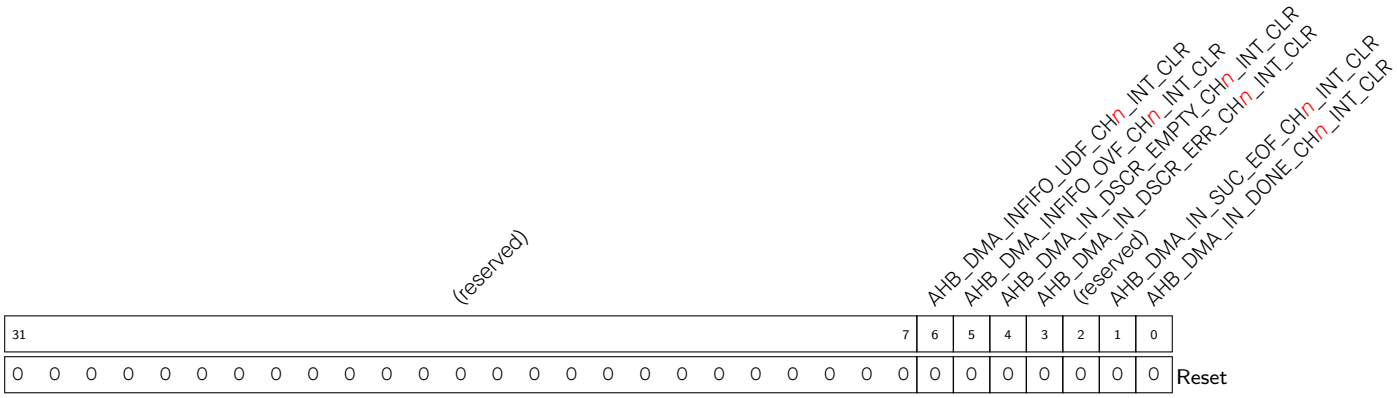
**AHB_DMA_IN_DSCR_ERR_CH*n*_INT_ENA** Write 1 to enable AHB_DMA_IN_DSCR_ERR_CH*n*_INT. (R/W)

**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)

**Register 4.4. AHB_DMA_IN_INT_CLR_CH*n*_REG (*n*: 0-1) (0x000C+0x10**n*)**




**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)

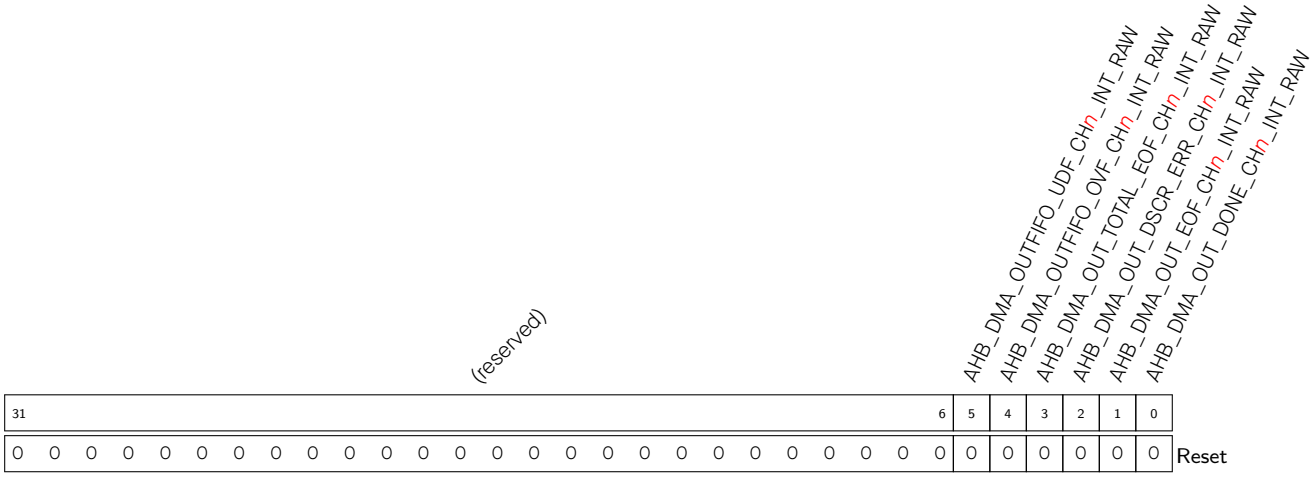
**AHB_DMA_IN_DSCR_ERR_CH*n*_INT_CLR** Write 1 to clear AHB_DMA_IN_DSCR_ERR_CH*n*_INT. (WT)

**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)

**Register 4.5. AHB_DMA_OUT_INT_RAW_CH*n*_REG (*n*: 0-1) (0x0030+0x10**n*)**




**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)

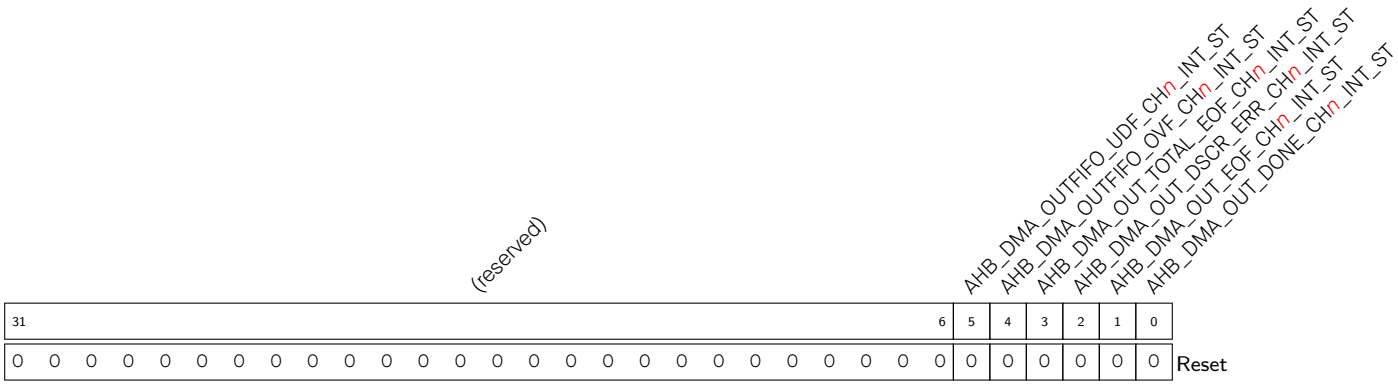
**AHB_DMA_OUT_DSCR_ERR_CH*n*_INT_RAW** The raw interrupt status of AHB_DMA_OUT_DSCR_ERR_CH*n*_INT. (R/WTC/SS)

**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)

**Register 4.6. AHB_DMA_OUT_INT_ST_CH*n*_REG (*n*: 0-1) (0x0034+0x10**n*)**




**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)

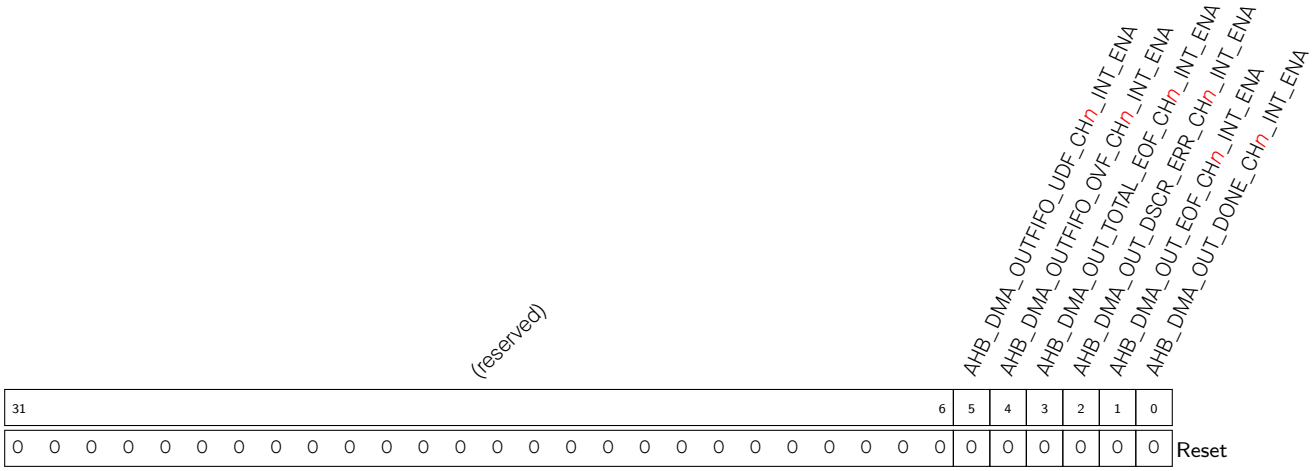
**AHB_DMA_OUT_DSCR_ERR_CH*n*_INT_ST** The masked interrupt status of AHB_DMA_OUT_DSCR_ERR_CH*n*_INT. (RO)

**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)

Register 4.7. AHB_DMA_OUT_INT_ENA_CH*n*_REG (*n*: 0-1) (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)

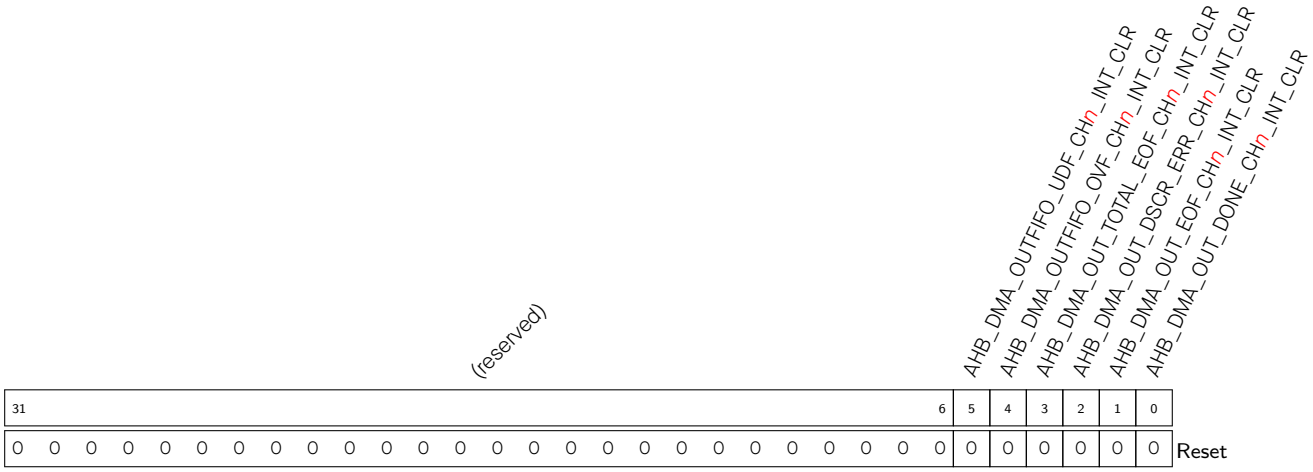
**AHB_DMA_OUT_DSCR_ERR_CH*n*_INT_ENA** Write 1 to enable AHB_DMA_OUT_DSCR_ERR_CH*n*_INT. (R/W)

**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)

Register 4.8. AHB_DMA_OUT_INT_CLR_CH*n*_REG (*n*: 0-1) (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)

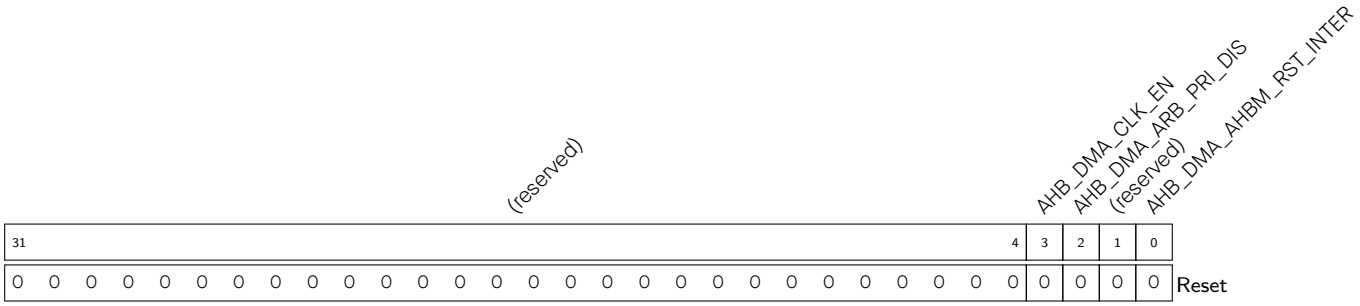
**AHB_DMA_OUT_DSCR_ERR_CH*n*_INT_CLR** Write 1 to clear AHB_DMA_OUT_DSCR_ERR_CH*n*_INT. (WT)

**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)

Register 4.9. 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)

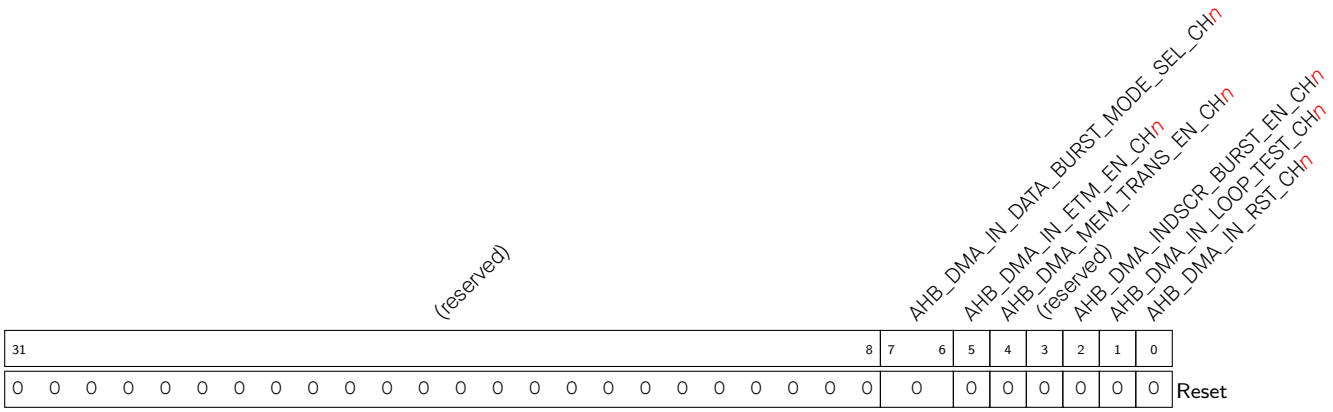
**AHB_DMA_CLK_EN** Configures AHB DMA clock gating.

0: Support clock only when the application writes registers

1: Always force the clock on for registers

(R/W)

Register 4.10. AHB_DMA_IN_CONF0_CH*n*_REG (*n*: 0-1) (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*** Configures the owner bit value for inlink write-back. (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_MEM_TRANS_EN_CH*n*** Configures whether to enable memory-to-memory data transfer.

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)

**AHB_DMA_IN_DATA_BURST_MODE_SEL_CH*n*** Configures maximum burst length for RX channel*n*.

0: SINGLE

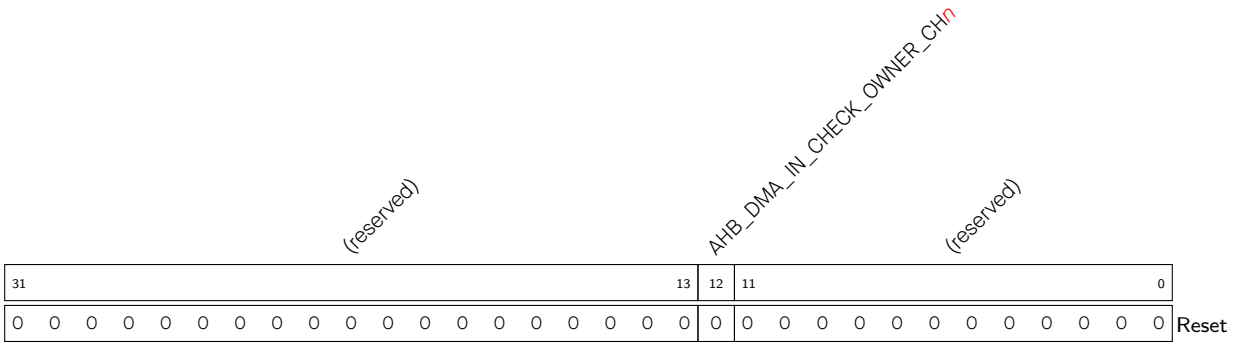
1: INCR4

2: INCR8

3: Reserved

(R/W)

Register 4.11. AHB_DMA_IN_CONF1_CH*n*_REG (*n*: 0-1) (0x0074+0xC0*n*)



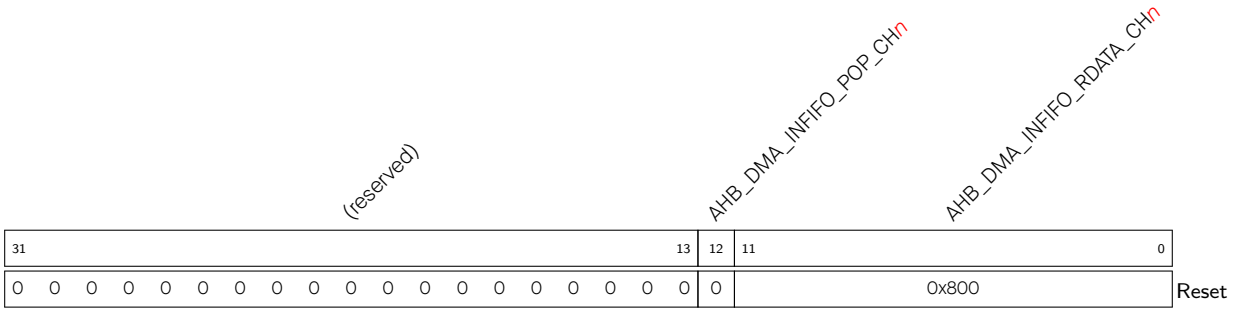

**AHB_DMA_IN_CHECK_OWNER_CH*n*** Configures whether to enable owner bit check for RX channel *n*.

0: Disable

1: Enable

(R/W)

Register 4.12. AHB_DMA_IN_POP_CH*n*_REG (*n*: 0-1) (0x007C+0xC0*n*)




**AHB_DMA_INFIFO_RDATA_CH*n*** Represents the data popped from AHB DMA RX FIFO. (RO)

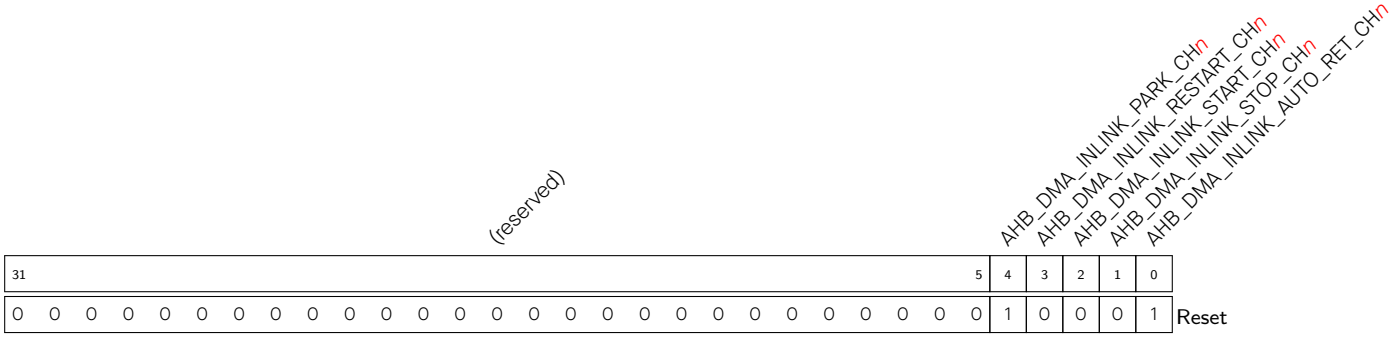
**AHB_DMA_INFIFO_POP_CH*n*** Configures whether to pop data from AHB DMA RX FIFO.

0: Invalid. No effect

1: Pop

(WT)

Register 4.13. AHB_DMA_IN_LINK_CH*n*_REG (*n*: 0-1) (0x0080+0xC0*n*)




**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)

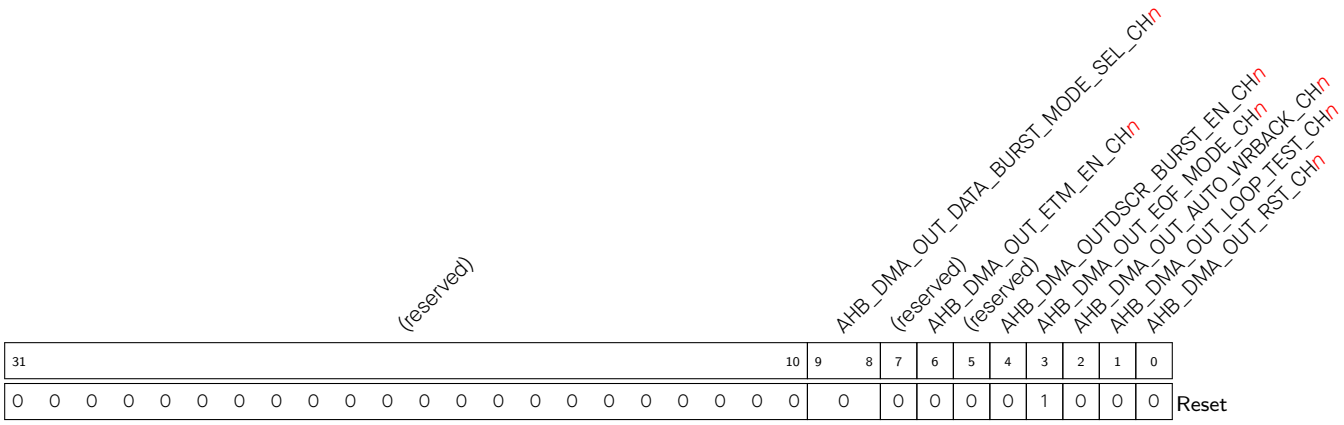
**AHB_DMA_INLINK_PARK_CH*n*** Represents the status of the receive descriptor's FSM.

0: Running

1: Idle

(RO)

**Register 4.14. AHB_DMA_OUT_CONFO_CH*n*_REG (*n*: 0-1) (0x00D0+0xC0*n)**




**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*** Configures the owner bit value for outlink write-back. (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_ETM_EN_CH*n*** Configures whether to enable ETM control for TX channel *n*.

0: Disable

1: Enable

(R/W)

**AHB_DMA_OUT_DATA_BURST_MODE_SEL_CH*n*** Configures maximum burst length for TX channel*n*.

0: SINGLE

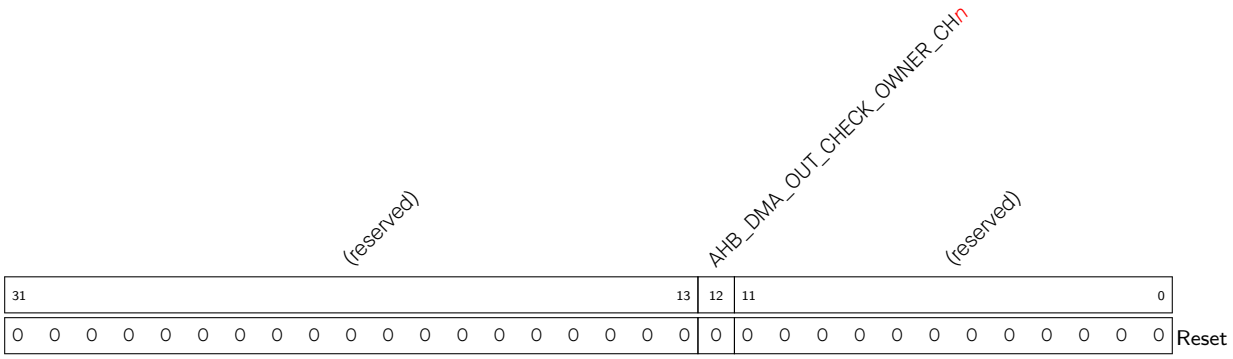
1: INCR4

2: INCR8

3: Reserved

(R/W)

Register 4.15. AHB_DMA_OUT_CONF1_CH*n*_REG (*n*: 0-1) (0x00D4+0xC0**n*)



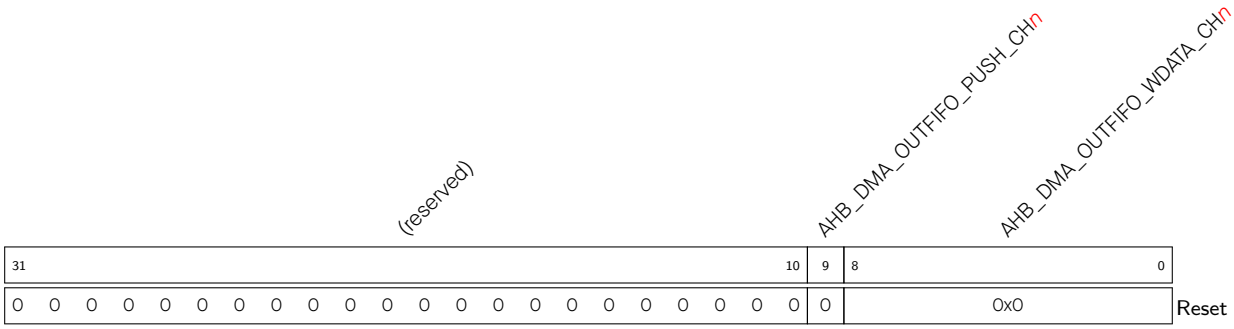

**AHB_DMA_OUT_CHECK_OWNER_CH*n*** Configures whether to enable owner bit check for TX channel *n*.

0: Disable

1: Enable

(R/W)

Register 4.16. AHB_DMA_OUT_PUSH_CH*n*_REG (*n*: 0-1) (0x00DC+0xC0**n*)




**AHB_DMA_OUTFIFO_WDATA_CH*n*** Represents the data that need to be pushed into AHB DMA TX FIFO. (R/W)

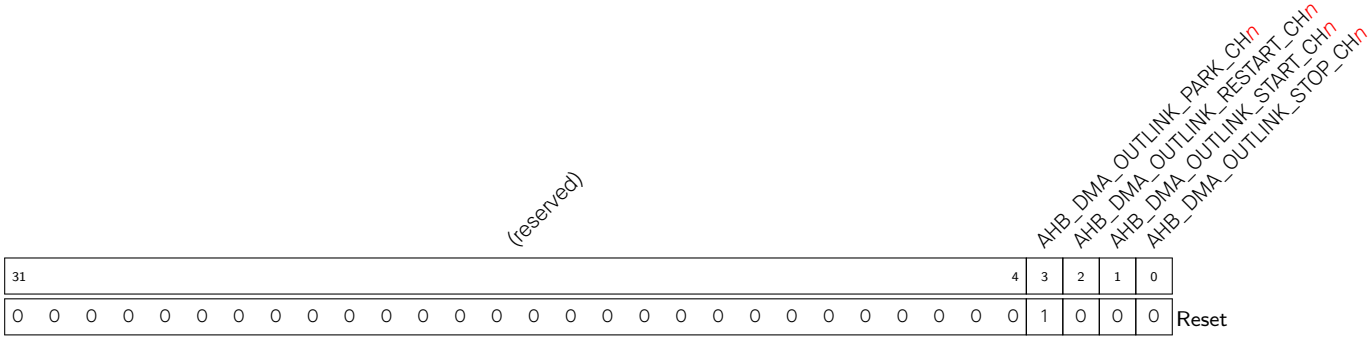
**AHB_DMA_OUTFIFO_PUSH_CH*n*** Configures whether to push data into AHB DMA TX FIFO.

0: Invalid. No effect

1: Push

(WT)

Register 4.17. AHB_DMA_OUT_LINK_CH*n*_REG (*n*: 0-1) (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)

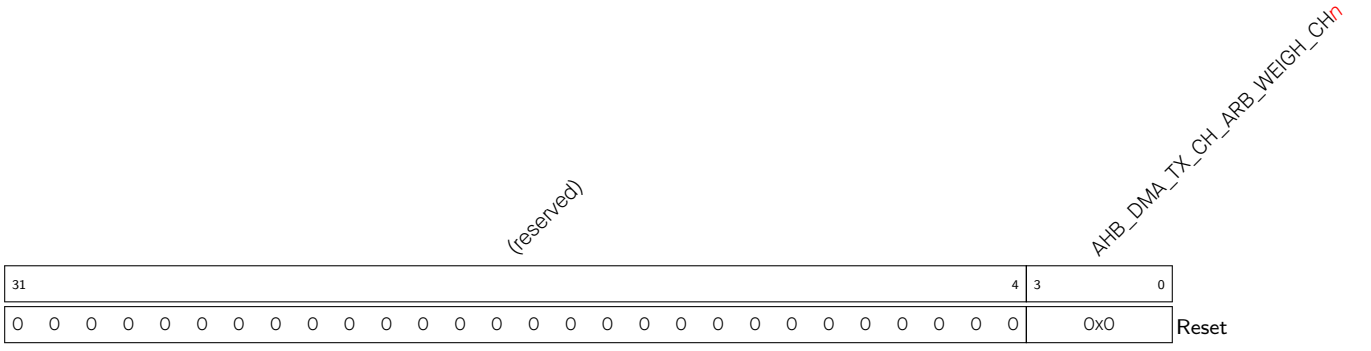
**AHB_DMA_OUTLINK_PARK_CH*n*** Represents the status of the transmit descriptor's FSM.

0: Running

1: Idle

(RO)

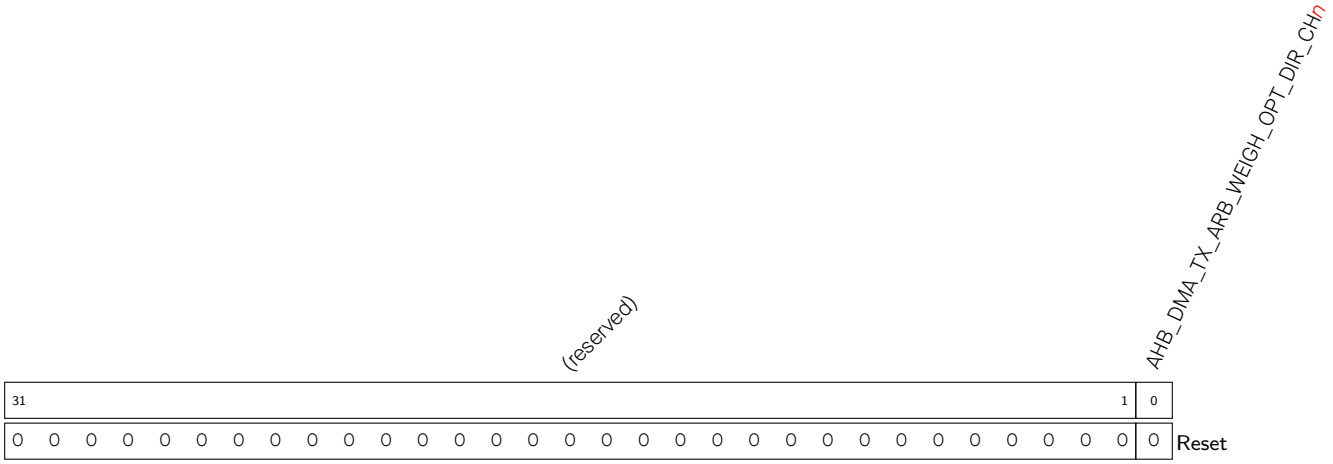
Register 4.18. AHB_DMA_TX_CH_ARB_WEIGH_CH*n*_REG (*n*: 0-1) (0x02DC+0x28**n*)




**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 4.19. AHB_DMA_TX_ARB_WEIGH_OPT_DIR_CH*n*_REG (*n*: 0-1) (0x02E0+0x28**n*)**



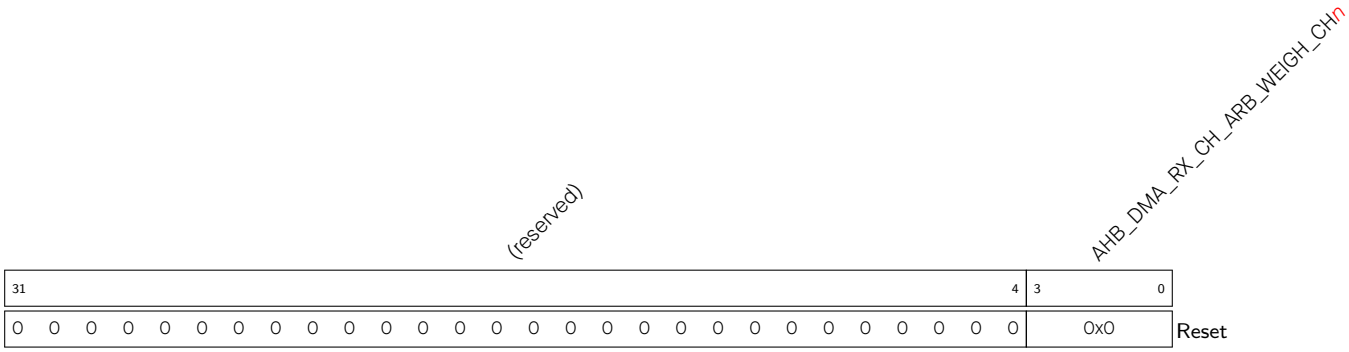

**AHB_DMA_TX_ARB_WEIGH_OPT_DIR_CH*n*** Configures whether to enable weight optimization for TX channel *n*.

0: Disable

1: Enable

(R/W)

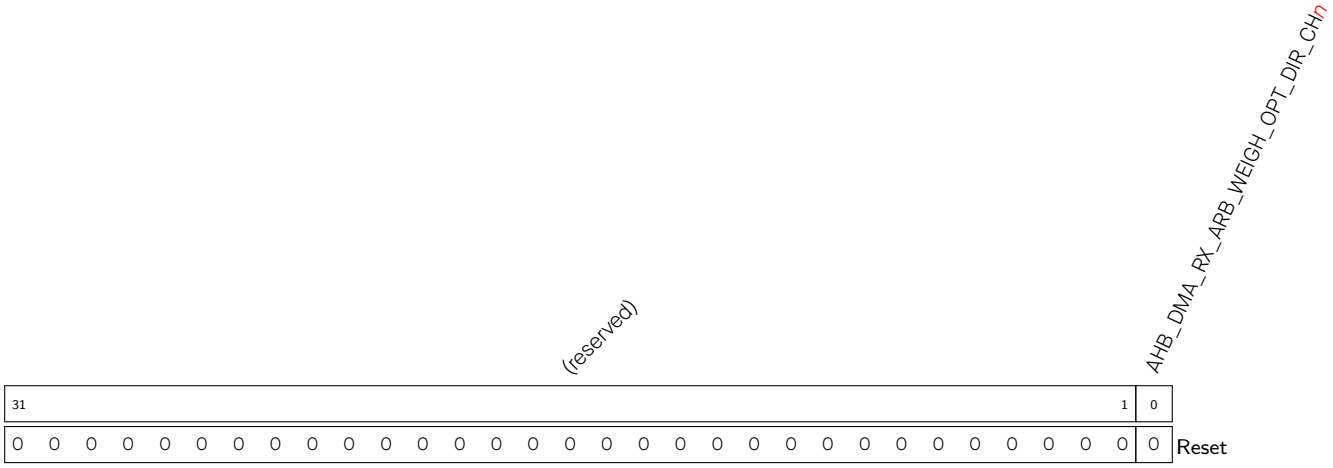
**Register 4.20. AHB_DMA_RX_CH_ARB_WEIGH_CH*n*_REG (*n*: 0-1) (0x0354+0x28**n*)**




**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 4.21. AHB_DMA_RX_ARB_WEIGH_OPT_DIR_CH*n*_REG (*n*: 0-1) (0x0358+0x28**n*)



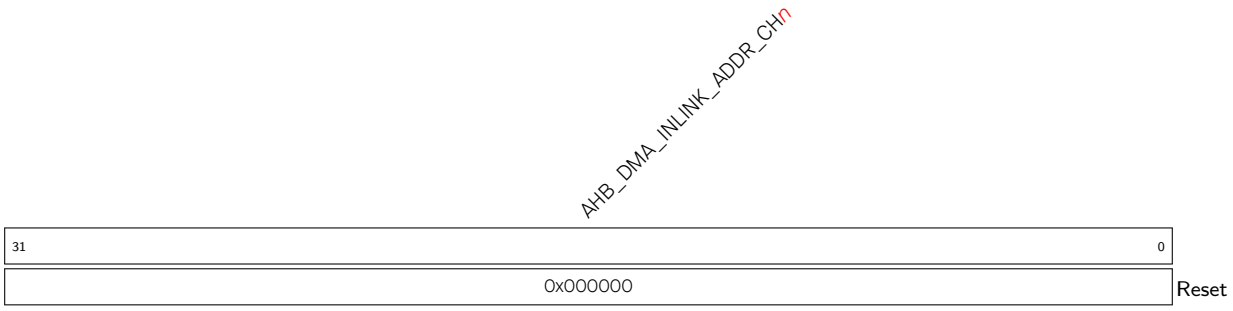

**AHB_DMA_RX_ARB_WEIGH_OPT_DIR_CH*n*** Configures whether to enable weight optimization for RX channel *n*.

0: Disable

1: Enable

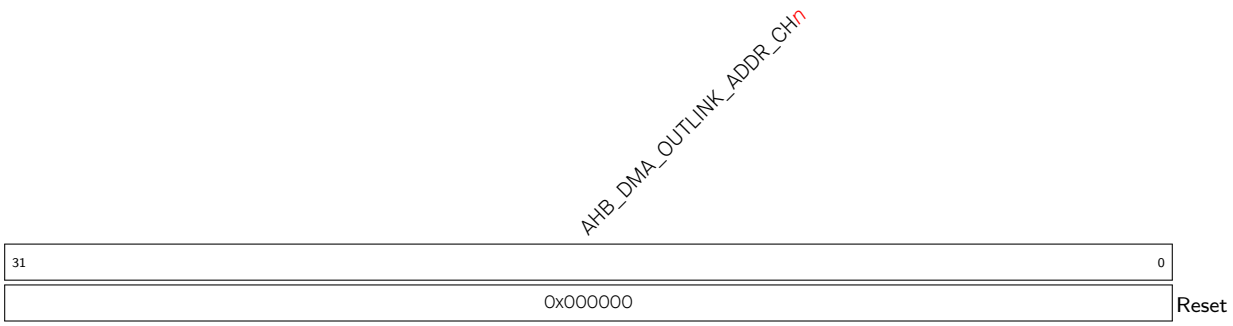
(R/W)

Register 4.22. AHB_DMA_IN_LINK_ADDR_CH*n*_REG (*n*: 0-1) (0x03AC+0x4**n*)



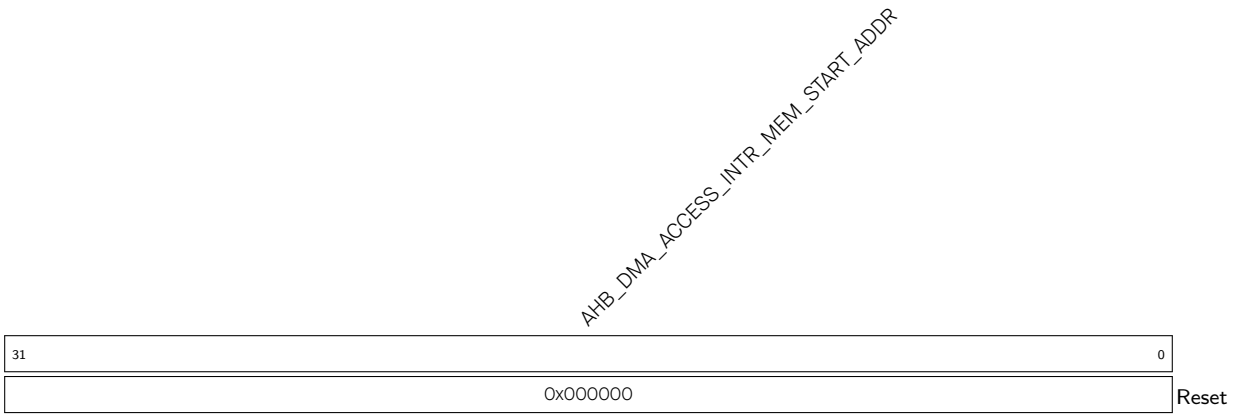

**AHB_DMA_INLINK_ADDR_CH*n*** Represents the first receive descriptor's address. (R/W)

Register 4.23. AHB_DMA_OUT_LINK_ADDR_CH*n*_REG (*n*: 0-1) (0x03B8+0x4**n*)



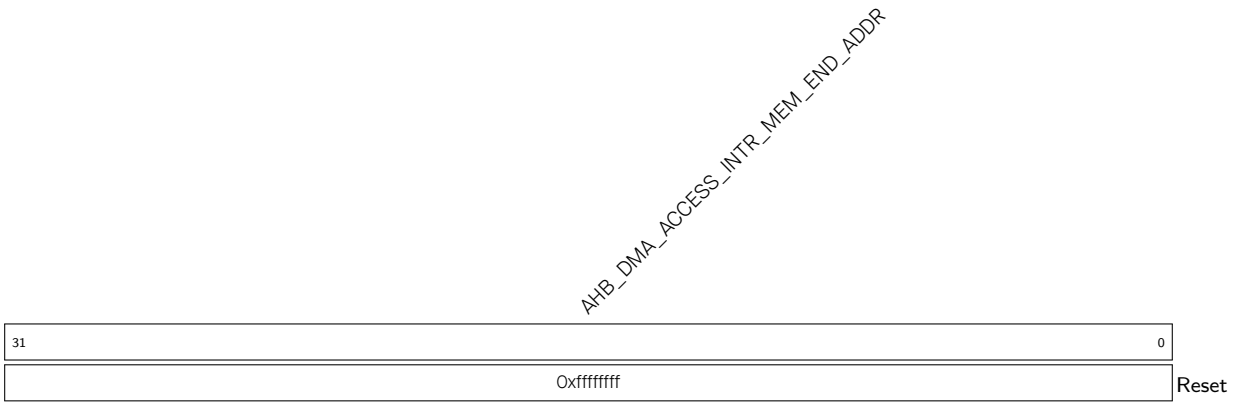

**AHB_DMA_OUTLINK_ADDR_CH*n*** Represents the first transmit descriptor's address. (R/W)

Register 4.24. AHB_DMA_INTR_MEM_START_ADDR_REG (0x03C4)



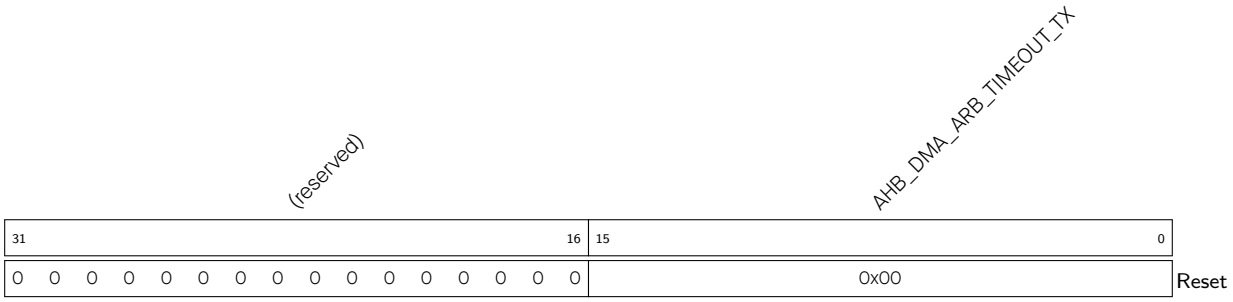

**AHB_DMA_ACCESS_INTR_MEM_START_ADDR** Configures the start address of accessible address space. (R/W)

Register 4.25. AHB_DMA_INTR_MEM_END_ADDR_REG (0x03C8)



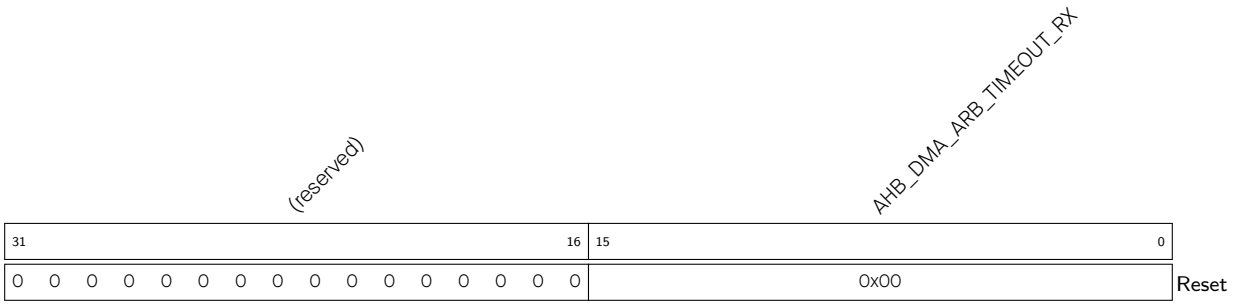

**AHB_DMA_ACCESS_INTR_MEM_END_ADDR** Configures the end address of accessible address space. (R/W)

Register 4.26. AHB_DMA_ARB_TIMEOUT_TX_REG (0x03CC)



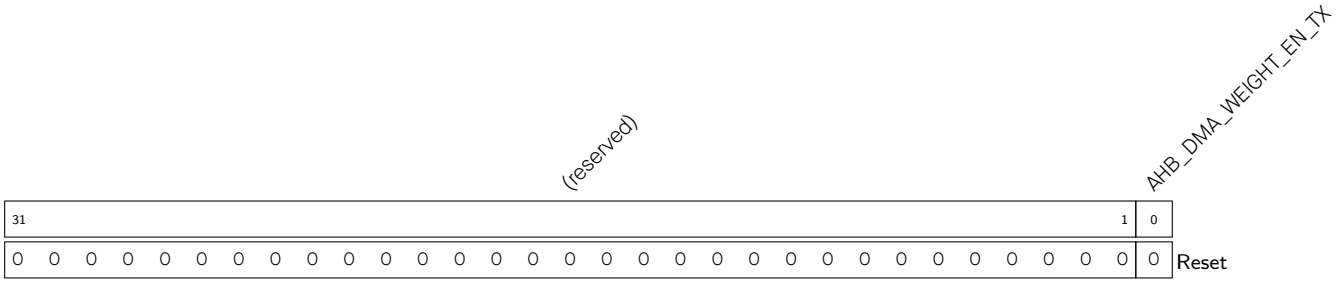

**AHB_DMA_ARB_TIMEOUT_TX** Configures the time slot for TX. Measurement unit: AHB bus clock cycle. (R/W)

Register 4.27. AHB_DMA_ARB_TIMEOUT_RX_REG (0x03D0)




**AHB_DMA_ARB_TIMEOUT_RX** Configures the time slot for RX. Measurement unit: AHB bus clock cycle. (R/W)

Register 4.28. AHB_DMA_WEIGHT_EN_TX_REG (0x03D4)



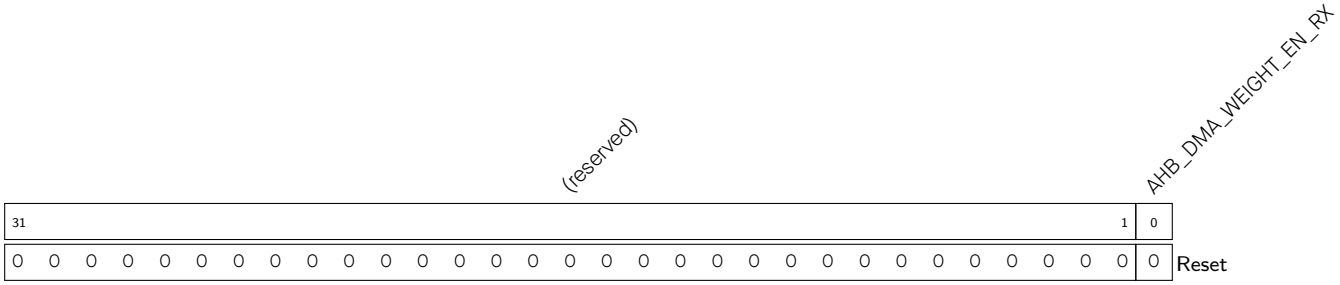

**AHB_DMA_WEIGHT_EN_TX** Configures whether to enable weight arbitration for TX.

0: Disable

1: Enable

(R/W)

Register 4.29. AHB_DMA_WEIGHT_EN_RX_REG (0x03D8)



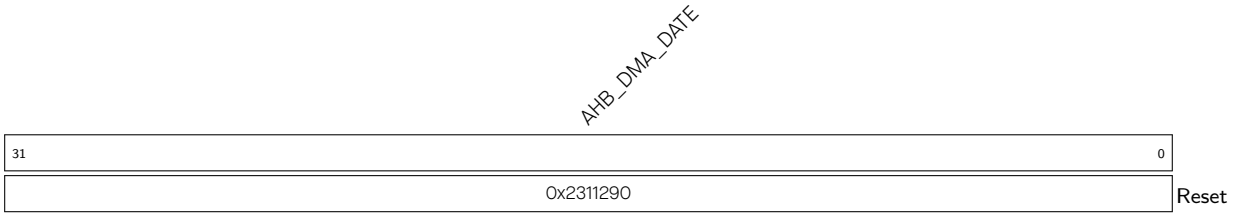

**AHB_DMA_WEIGHT_EN_RX** Configures whether to enable weight arbitration for RX.

0: Disable

1: Enable

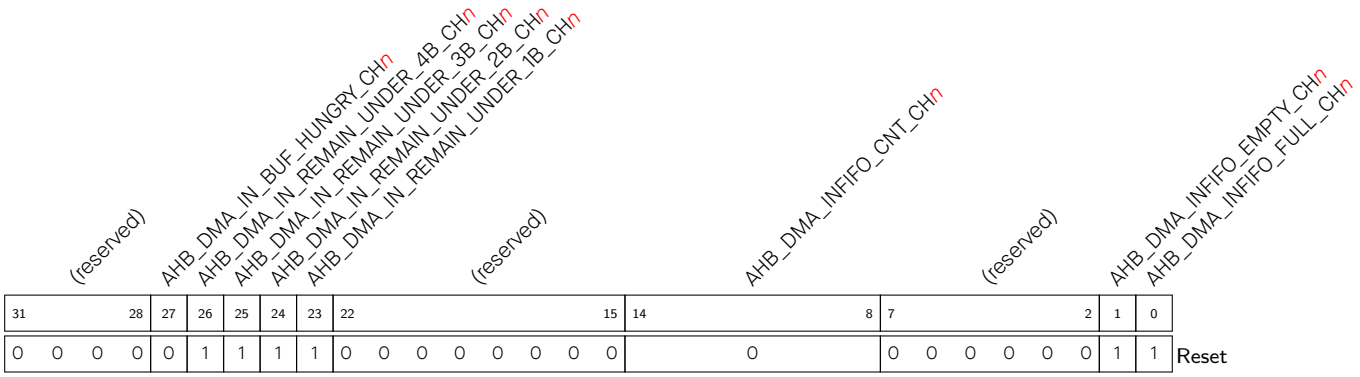
(R/W)

Register 4.30. AHB_DMA_DATE_REG (0x0068)




**AHB_DMA_DATE** Version control register. (R/W)

Register 4.31. AHB_DMA_INFIFO_STATUS_CH*n*_REG (*n*: 0-1) (0x0078+0xC0*n*)




**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)

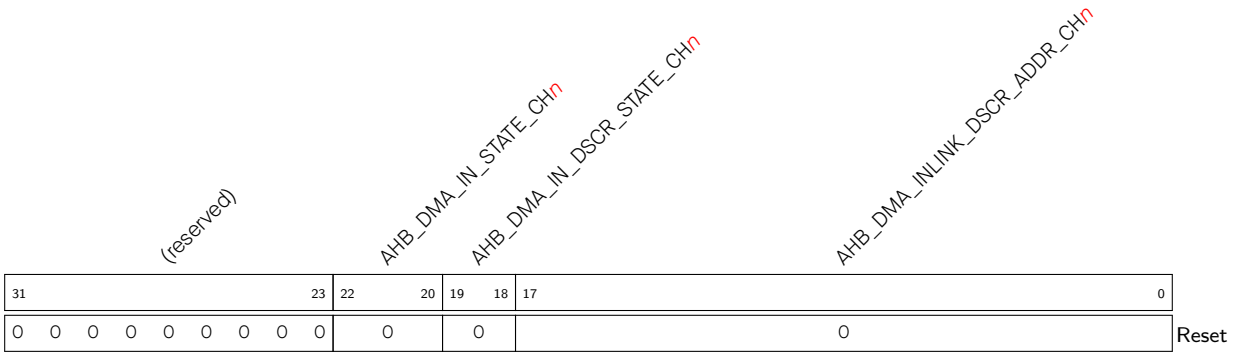
**AHB_DMA_IN_REMAIN_UNDER_2B_CH*n*** Reserved. (RO)

**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 4.32. AHB_DMA_IN_STATE_CH*n*_REG (*n*: 0-1) (0x0084+0xC0*n*)**

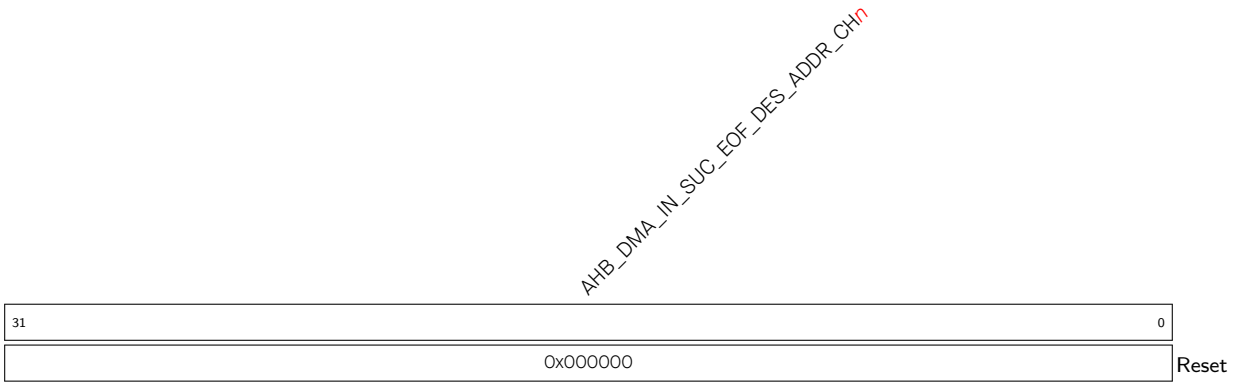



**AHB_DMA_INLINK_DSCR_ADDR_CH*n*** Represents the lower 18 bits of the next receive descriptor address that is pre-read (but not processed yet). If the current receive descriptor is the last descriptor, then this field represents the address of the current receive descriptor. (RO)

**AHB_DMA_IN_DSCR_STATE_CH*n*** Reserved. (RO)

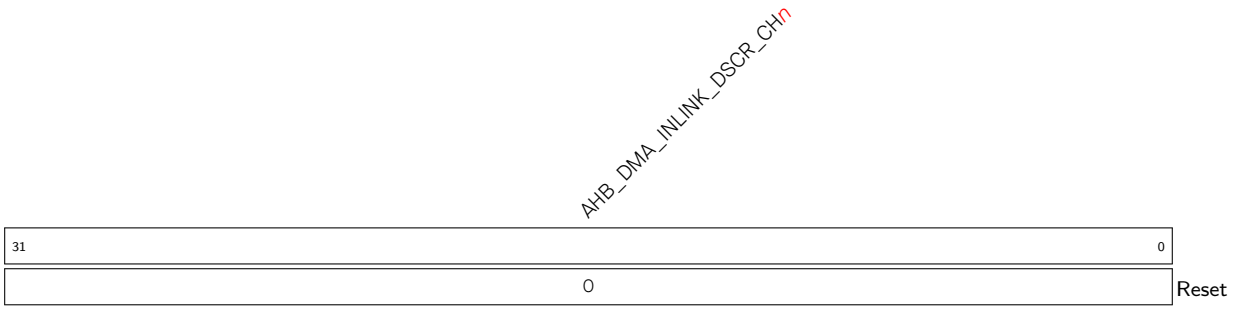
**AHB_DMA_IN_STATE_CH*n*** Reserved. (RO)

**Register 4.33. AHB_DMA_IN_SUC_EOF_DES_ADDR_CH*n*_REG (*n*: 0-1) (0x0088+0xC0*n*)**



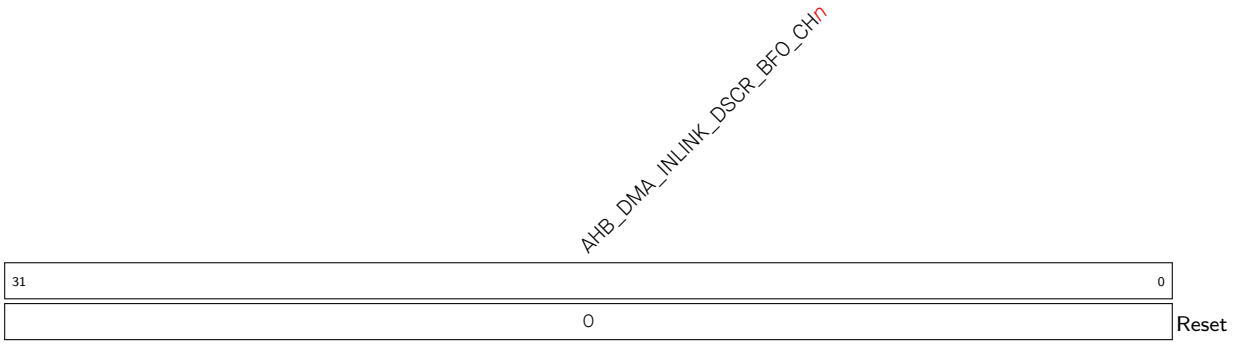

**AHB_DMA_IN_SUC_EOF_DES_ADDR_CH*n*** Represents the address of the receive descriptor when the EOF bit in this descriptor is 1. (RO)

Register 4.34. AHB_DMA_IN_DSCR_CH*n*_REG (*n*: 0-1) (0x0090+0xC0**n*)



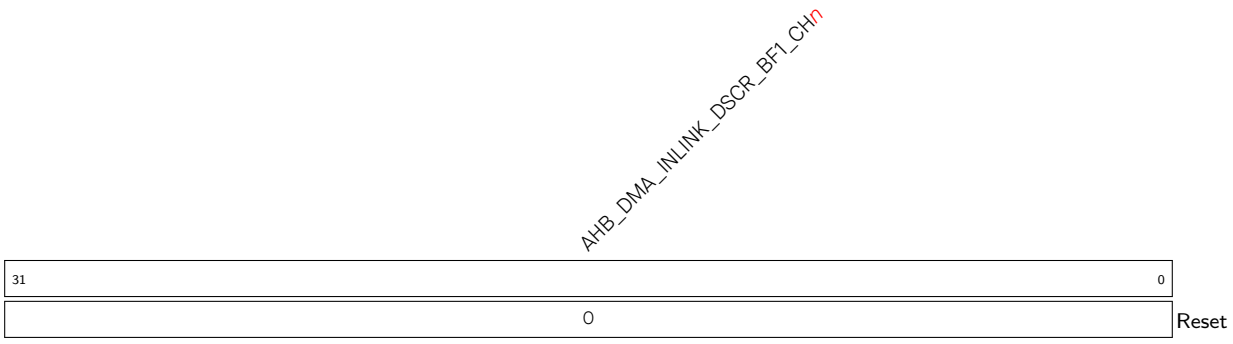

**AHB_DMA_INLINK_DSCR_CH*n*** Represents the address of the next receive descriptor x+1 pointed by the current receive descriptor that is pre-read. (RO)

Register 4.35. AHB_DMA_IN_DSCR_BF0_CH*n*_REG (*n*: 0-1) (0x0094+0xC0**n*)



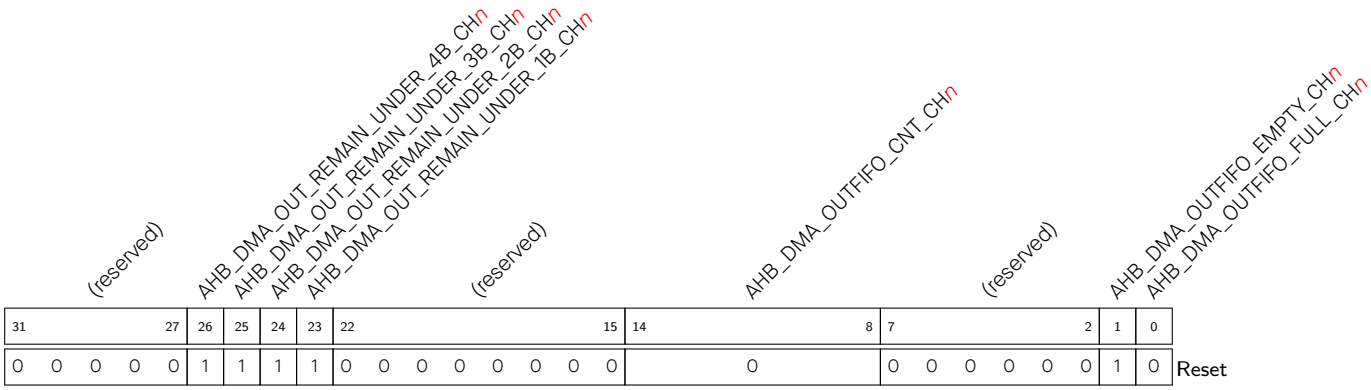

**AHB_DMA_INLINK_DSCR_BF0_CH*n*** Represents the address of the current receive descriptor x that is pre-read. (RO)

Register 4.36. AHB_DMA_IN_DSCR_BF1_CH*n*_REG (*n*: 0-1) (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 4.37. AHB_DMA_OUTFIFO_STATUS_CH*n*_REG (*n*: 0-1) (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)

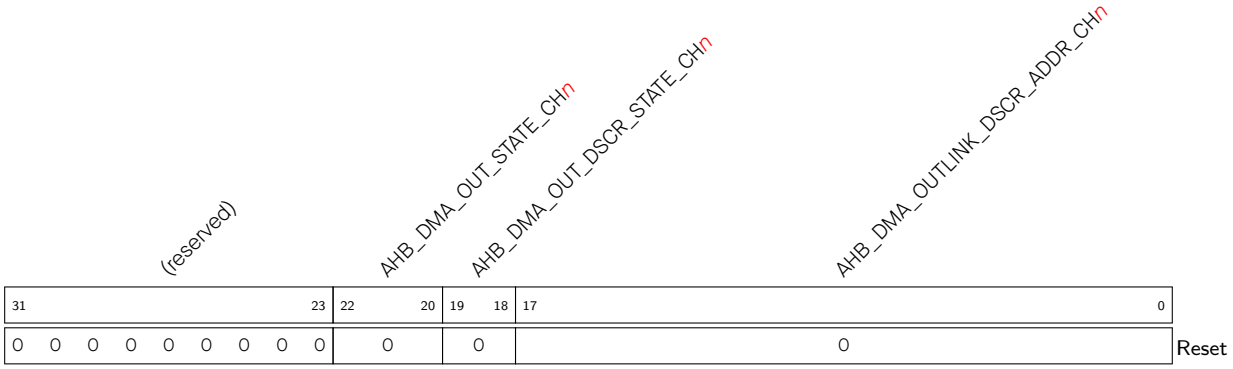
**AHB_DMA_OUT_REMAIN_UNDER_1B_CH*n*** Reserved. (RO)

**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 4.38. AHB_DMA_OUT_STATE_CH*n*_REG (*n*: 0-1) (0x00E4+0xC0*n*)

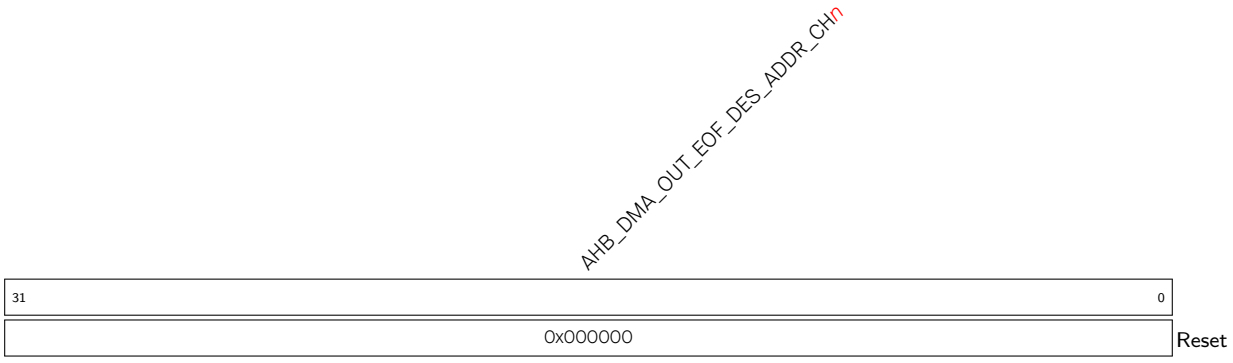



**AHB_DMA_OUTLINK_DSCR_ADDR_CH*n*** Represents the lower 18 bits of the next transmit descriptor address that is pre-read (but not processed yet). If the current transmit descriptor is the last descriptor, then this field represents the address of the current transmit descriptor. (RO)

**AHB_DMA_OUT_DSCR_STATE_CH*n*** Reserved. (RO)

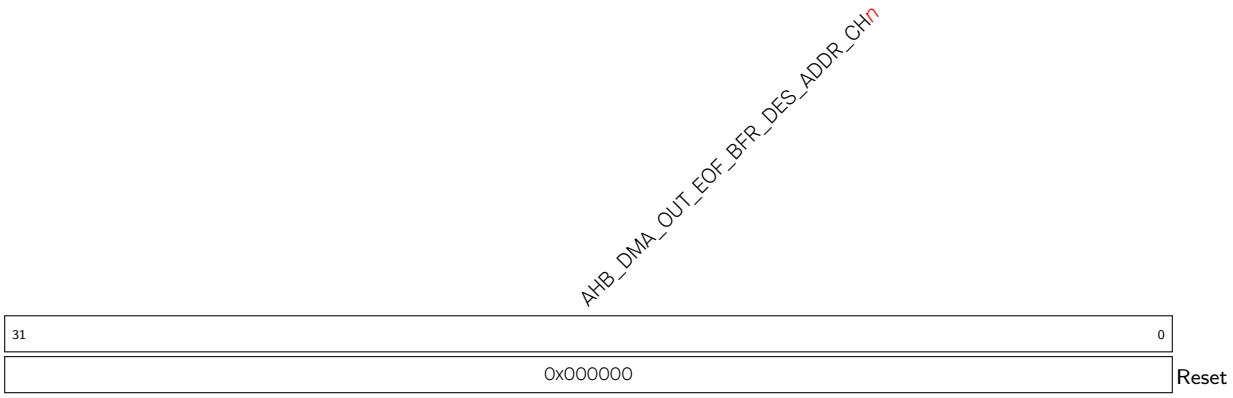
**AHB_DMA_OUT_STATE_CH*n*** Reserved. (RO)

Register 4.39. AHB_DMA_OUT_EOF_DES_ADDR_CH*n*_REG (*n*: 0-1) (0x00E8+0xC0*n*)



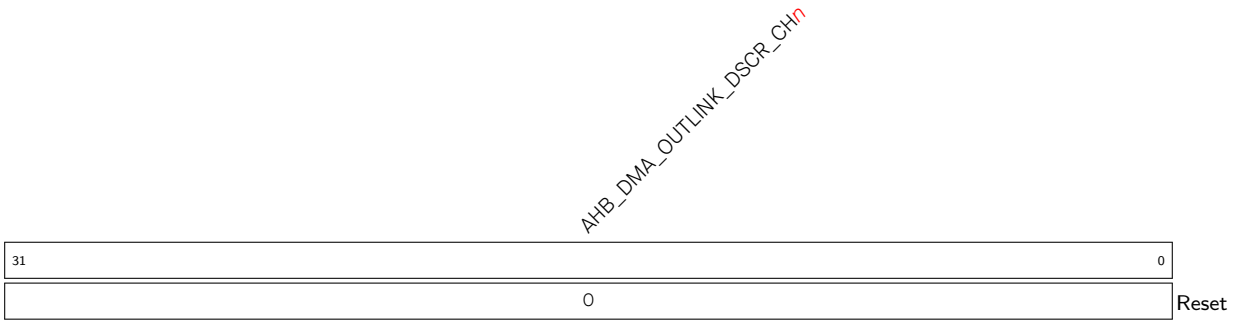

**AHB_DMA_OUT_EOF_DES_ADDR_CH*n*** Represents the address of the transmit descriptor when the EOF bit in this descriptor is 1. (RO)

Register 4.40. AHB_DMA_OUT_EOF_BFR_DES_ADDR_CH*n*_REG (*n*: 0-1) (0x00EC+0xC0*n*)



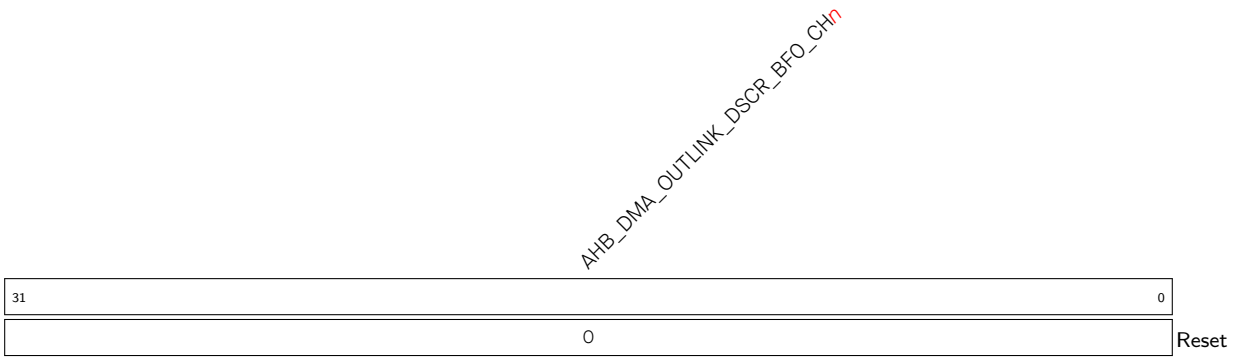

**AHB_DMA_OUT_EOF_BFR_DES_ADDR_CH*n*** Represents the address of the transmit descriptor before the last transmit descriptor. (RO)

Register 4.41. AHB_DMA_OUT_DSCR_CH*n*_REG (*n*: 0-1) (0x00F0+0xC0*n*)



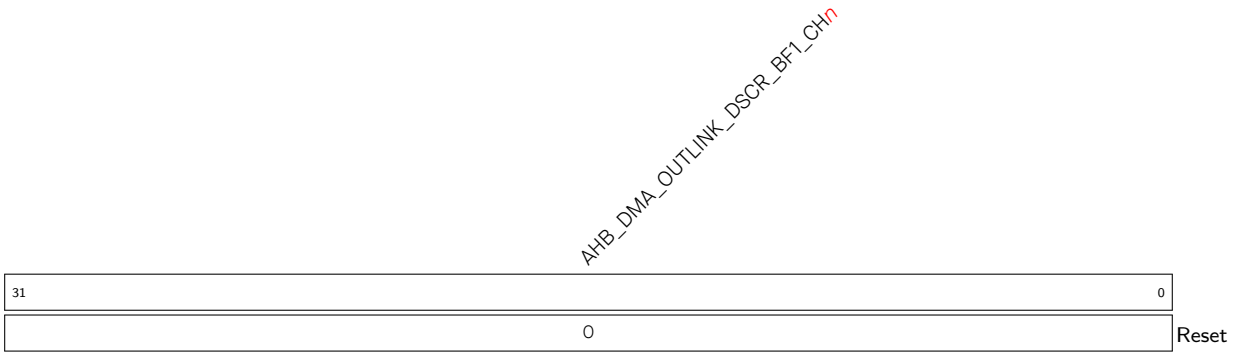

**AHB_DMA_OUTLINK_DSCR_CH*n*** Represents the address of the next transmit descriptor y+1 pointed by the current transmit descriptor that is pre-read. (RO)

Register 4.42. AHB_DMA_OUT_DSCR_BF0_CH*n*_REG (*n*: 0-1) (0x00F4+0xC0*n*)



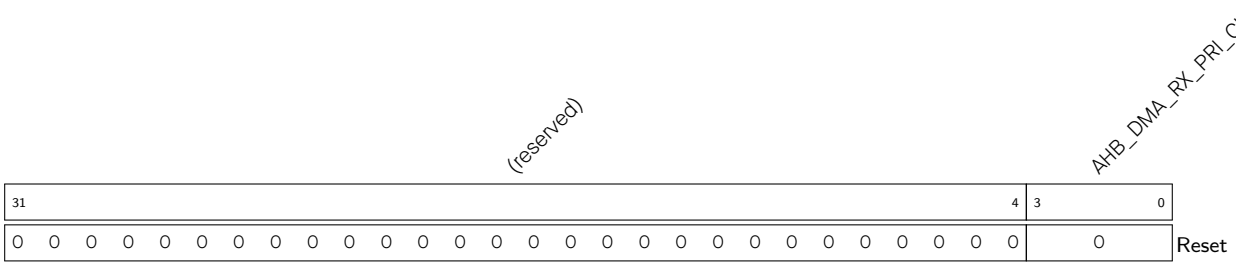

**AHB_DMA_OUTLINK_DSCR_BF0_CH*n*** Represents the address of the current transmit descriptor y that is pre-read. (RO)

Register 4.43. AHB_DMA_OUT_DSCR_BF1_CH*n*_REG (*n*: 0-1) (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)

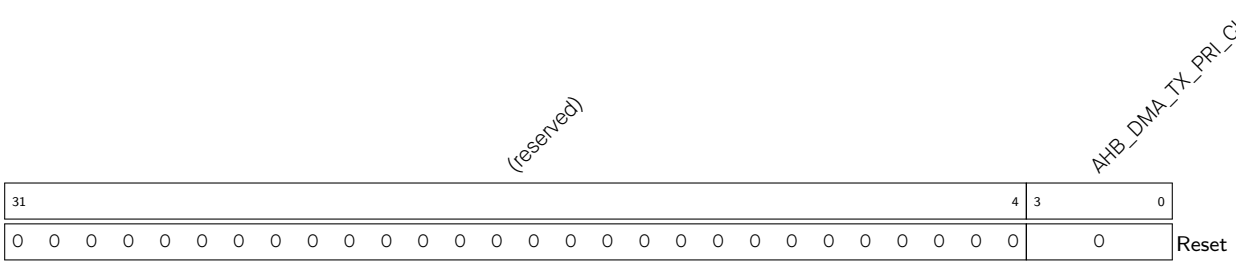
Register 4.44. AHB_DMA_IN_PRI_CH*n*_REG (*n*: 0-1) (0x009C+0xC0*n*)




**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)

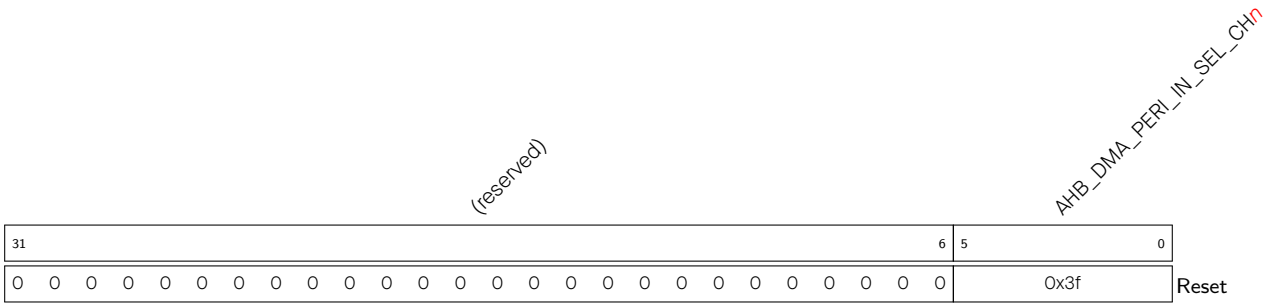
Register 4.45. AHB_DMA_OUT_PRI_CH*n*_REG (*n*: 0-1) (0x00FC+0xC0*n*)




**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)

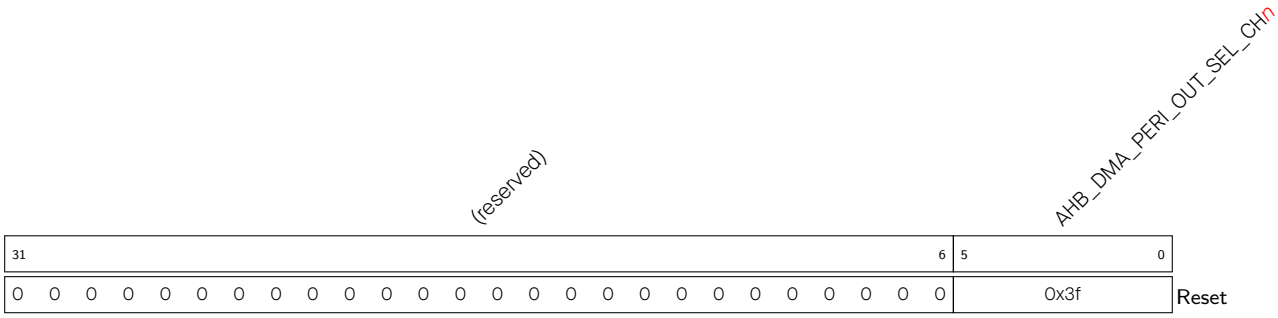
Register 4.46. AHB_DMA_IN_PERI_SEL_CH*n*_REG (*n*: 0-1) (0x00A0+0xC0**n*)




**AHB_DMA_PERI_IN_SEL_CH*n*** Configures the peripheral connected to RX channel *n*.

0: Dummy
1: GP-SPI
2: Dummy
3: I2S
4 ~ 6: Dummy
7: SHA
8: ADC
9 ~ 15: Dummy
16 ~ 63: Invalid
(R/W)

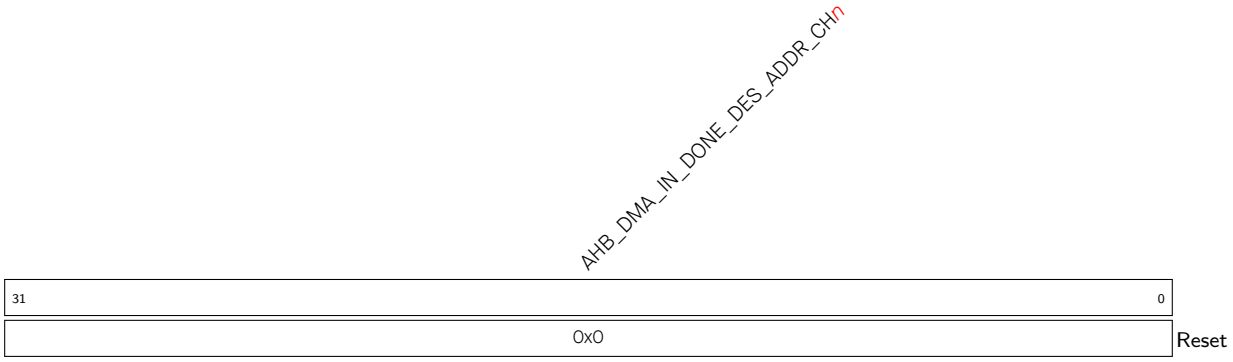
Register 4.47. AHB_DMA_OUT_PERI_SEL_CH*n*_REG (*n*: 0-1) (0x0100+0xC0**n*)




**AHB_DMA_PERI_OUT_SEL_CH*n*** Configures the peripheral connected to TX channel *n*.

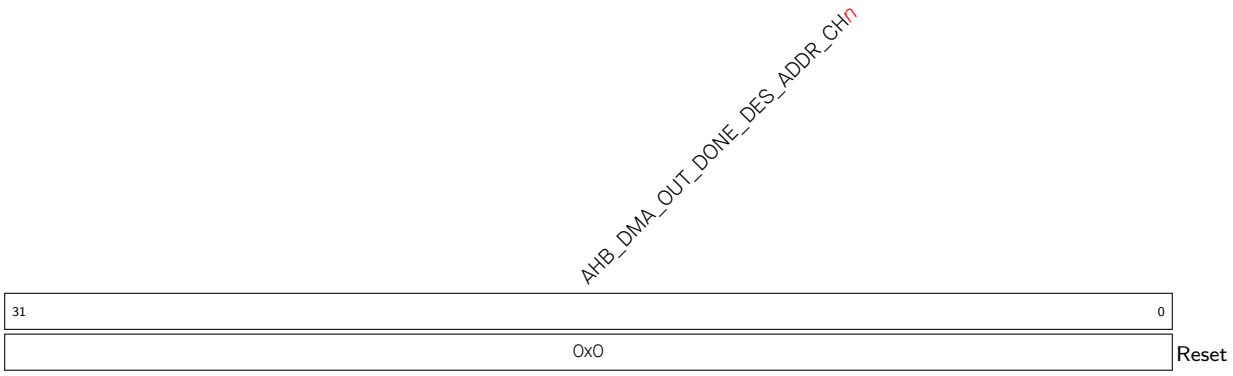
0: Dummy
1: GP-SPI
2: Dummy
3: I2S
4 ~ 6: Dummy
7: SHA
8: ADC
9 ~ 15: Dummy
16 ~ 63: Invalid
(R/W)

Register 4.48. AHB_DMA_IN_DONE_DES_ADDR_CH*n*_REG (*n*: 0-1) (0x00B0+0xC0*n*)




**AHB_DMA_IN_DONE_DES_ADDR_CH*n*** Represents the address of the inlink descriptor when this descriptor is completed. (RO)

Register 4.49. AHB_DMA_OUT_DONE_DES_ADDR_CH*n*_REG (*n*: 0-1) (0x0110+0xC0*n*)




**AHB_DMA_OUT_DONE_DES_ADDR_CH*n*** Represents the address of the outlink descriptor when this descriptor is completed. (RO)

# Part II

# 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 5

# System and Memory

## 5.1 Overview

ESP32-C61 has an ultra-low power and highly-integrated system with a high-performance 32-bit RISC-V processor (CPU), five-stage pipeline, clock frequency up to 160 MHz.

All internal memory, external memory, and peripherals are located on the CPU bus.

## 5.2 Features

- **Address Space**
  - 576 KB of internal memory address space, accessible by the instruction bus or data bus
  - 832 KB of peripheral address space
  - 32 MB of virtual address space for external memory, accessible by the instruction bus or data bus
  - 320 KB of internal DMA address space
  - 32 MB of external DMA address space
- **Internal Memory**
  - 256 KB of ROM
  - 320 KB of HP SRAM
- **External Memory**
  - Up to 32 MB of external flash
  - Up to 32 MB of external RAM
- **Peripheral Space**
  - 44 modules/peripherals in total
- **GDMA**
  - 4 GDMA-supported modules/peripherals

## 5.3 Functional Description

### 5.3.1 Address Mapping

Figure 5.3-1 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 CPU are little-endian. All buses have a 32-bit data width. The CPU can access data via the data bus using single-byte, double-byte, and 4-byte alignment.

The CPU has the following access capabilities:

- Direct access to internal memory via both the data bus and instruction bus
  - 320 KB HP SRAM (0x4080_0000~0x4084_FFFF)
  - 256 KB ROM (0x4000_0000~0x4003_FFFF)
- Access to external memory through the cache
  The virtual address is mapped to the physical address space of the external memory through the MMU.
  - Up to 32 MB external flash (0×4200_0000~0×43FF_FFFF)
  - Up to 32 MB external RAM (0×4200_0000~0×43FF_FFFF)
- Direct access to modules/peripherals via the data bus, including:
  - CPU Peripherals
  - HP Peripherals
  - LP Peripherals

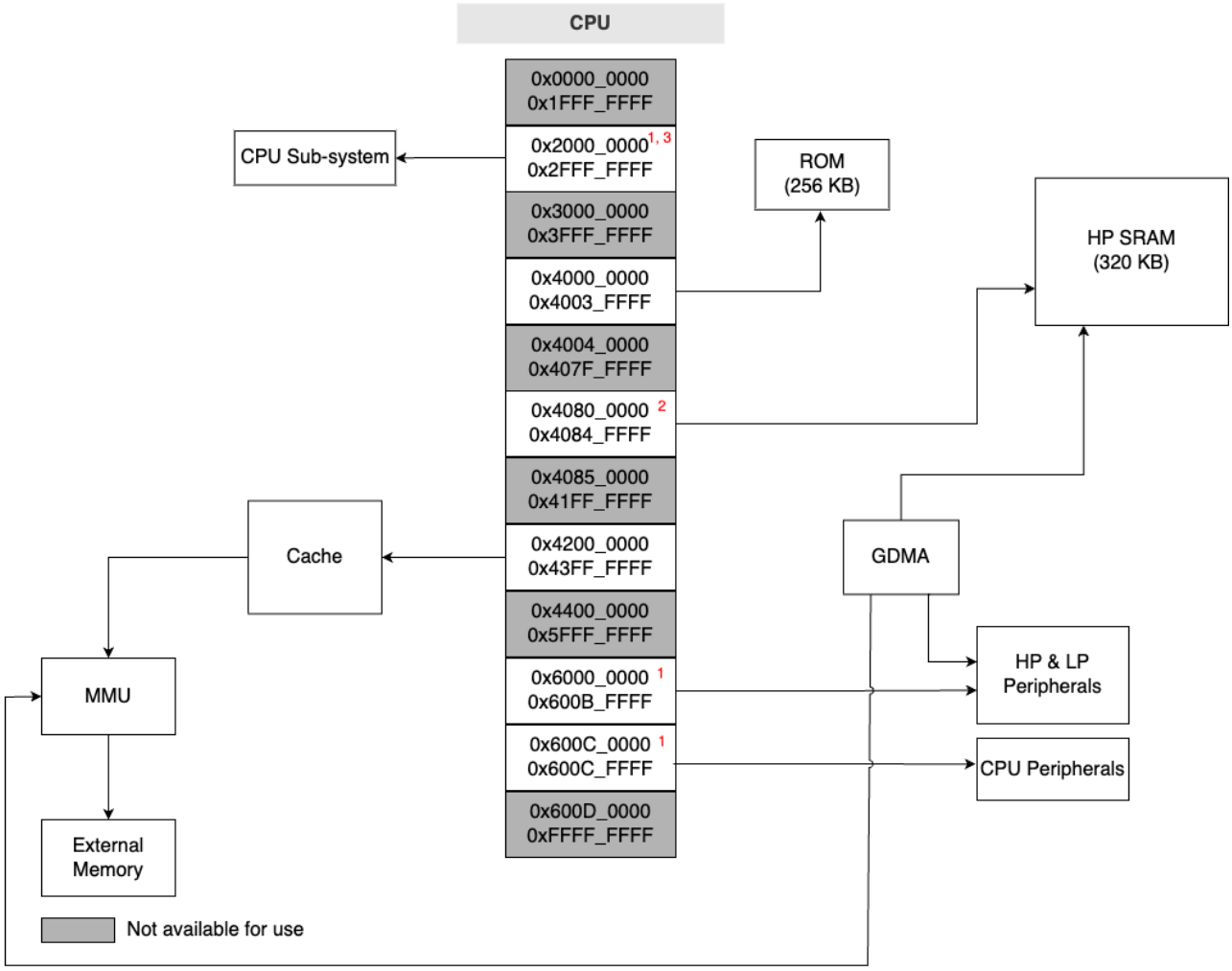



Figure 5.3-1. System Structure and Address Mapping

**Note:**

1. The range of addresses available in the address space may be larger than the actual available memory of a particular type.
2. Part of the internal memory is reserved for system instructions and data, therefore, the address range available in the address space may be larger than the actual available internal memory.
3. For CPU Sub-system, please refer to Chapter *2 ESP-RISC-V CPU*.

Table 5.3-1 lists the address ranges on the data bus and instruction bus and their corresponding target memories.

Table 5.3-1. Memory Address Mapping

| Bus Type | Boundary Address | | Size | Target |
|---|---|---|---|---|
| | Low Address | High Address | | |
| | 0x0000_0000 | 0x3FFF_FFFF | | Reserved |
| Data/Instruction bus | 0x4000_0000 | 0x4003_FFFF | 256 KB | ROM* |

Cont'd on next page

Table 5.3-1 – cont'd from previous page

| Bus Type | Boundary Address | | Size | Target |
|---|---|---|---|---|
| | Low Address | High Address | | |
| | 0x4004_0000 | 0x407F_FFFF | | Reserved |
| Data/Instruction bus | 0x4080_0000 | 0x4084_FFFF | 320 KB | HP SRAM* |
| | 0x4085_0000 | 0x41FF_FFFF | | Reserved |
| Data/Instruction bus | 0x4200_0000 | 0x43FF_FFFF | 32 MB | External memory |
| | 0x4400_0000 | 0x5FFF_FFFF | | Reserved |
| Data bus | 0x6000_0000 | 0x600C_FFFF | 832 KB | Peripherals |
| | 0x600D_0000 | 0xFFFF_FFFF | | Reserved |

* All of the internal memories are managed by Permission Control module. An internal memory can only be accessed when it is allowed by Permission Control, then the internal memory can be available to the CPU. For more information about Permission Control, please refer to Chapter 17 *Permission Control (PMS)*.

## 5.3.2 Internal Memory

ESP32-C61 has various types of internal memory:

- ROM (256 KB): The ROM is a read-only memory and can not be programmable. It contains the ROM code of some low-level system software and read-only data. Note that this memory can only be accessed by CPU.
- HP SRAM (320 KB): The HP SRAM is a volatile memory that can be quickly accessed by the CPU (generally within a single CPU clock cycle).

### 5.3.2.1 ROM

This 256 KB ROM is a read-only memory, accessed by the CPU directly through the instruction bus or through the data bus via the address range (0x4000_0000~0x4003_FFFF), see Table 5.3-1.

### 5.3.2.2 HP SRAM

This 320 KB HP SRAM is a read-and-write memory, accessed by the CPU through the instruction bus or data bus via the address range (0×4080_0000~0×4084_FFFF), see Table 5.3-1.

## 5.3.3 External Memory

ESP32-C61 supports SPI, Dual SPI, Quad SPI, and QPI interfaces that allow connection to external flash and RAM. ESP32-C61 also supports hardware manual encryption and automatic encryption/decryption based on XTS-AES algorithm to protect users' programs and data in the external flash and RAM.

### 5.3.3.1 External Memory Address Mapping

The external memory can be accessed by CPU via the cache or accessed by the GDMA. According to the information inside the MMU (Memory Management Unit), the cache maps the CPU and GDMA's address (0x4200_0000~0x43FF_FFFF) into a physical address of the external memory. With this address mapping, ESP32-C61 can address up to 32 MB external flash and 32 MB external RAM.

Note that the instruction bus shares the same address space (32 MB) with the data bus to access the external memory.

#### 5.3.3.2 Cache

As shown in Figure 5.3-2, ESP32-C61 has a shared four-way set-associative cache. Its size is 32 KB and its block size is 32 bytes. The instruction bus and data bus can access the cache simultaneously, but the cache can only respond to one of them at a time through arbitration. If a data missing happens, cache controller initiates a request to the external memory.

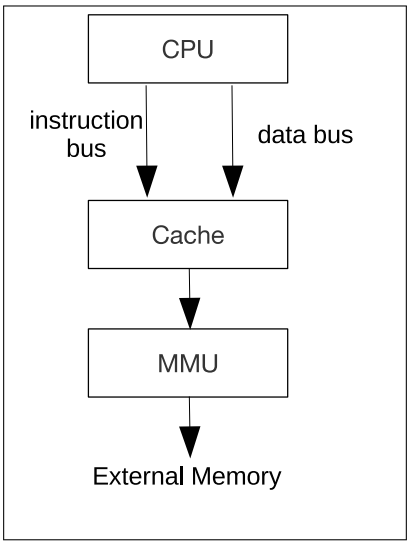



**Figure 5.3-2. Cache Structure**

#### 5.3.3.3 Cache Operations

ESP32-C61 cache supports the following operations:

1. **Write-back**
   - Cleans 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.
   - This write-back operation does not invalidate the data in the cache, therefore, CPU still can read/write data directly from/to the cache, where data missing will not occur.
2. **Write-back & Invalidate**
   - Cleans 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.
   - This operation invalidates the data in the cache, therefore, CPU cannot read/write data directly from/to the cache, where data missing will occur.
3. **Clean**
   - Cleans 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.

4. **Invalidate**
   - Cleans the valid bits in the tag memory. This is to remove valid data from the cache.
   - Once this operation is done, the deleted data is stored only in external memory.
   - 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.
5. **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).
6. **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.
   - Manual-Invalidate operation only works on the unlocked data. If you plan to perform this operation on the locked data, please unlock them first.

### 5.3.4 GDMA Address Space

The General Direct Memory Access (GDMA) peripheral in ESP32-C61 consisting of two TX channels and two RX channels provides Direct Memory Access (DMA) service, including:

- data transfers between different locations of internal memory

- data transfers between modules/peripherals and internal memory
- data transfers between different locations of external memory
- data transfers between modules/peripherals and external memory
- data transfers between internal memory and external memory

GDMA uses the same addresses as the data bus to access HP SRAM and external RAM, i.e., GDMA uses address range (0x4080_0000~0x4084_FFFF) to access HP SRAM, and (0x4200_0000~0x43FF_FFFF) to external RAM. Four modules/peripherals in ESP32-C61 work together with GDMA. As shown in Figure 5.3-3, four vertical lines correspond to these four modules/peripherals with GDMA function. The horizontal line represents a certain channel of GDMA (can be any channel), and the intersection of the vertical line and the horizontal line indicates that a module/peripheral has the ability to access the corresponding channel of GDMA.

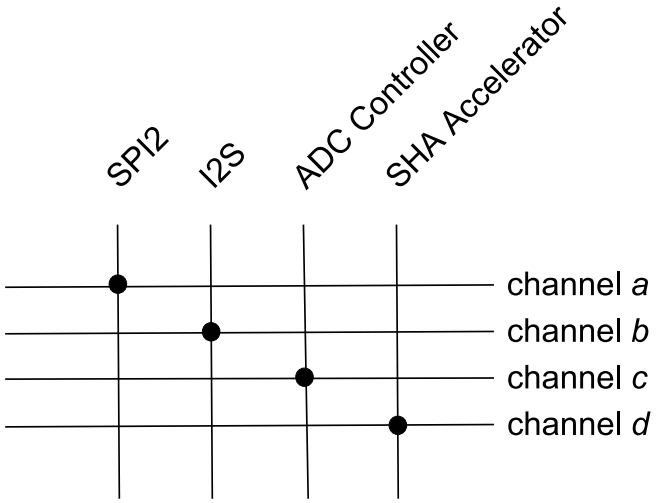



Figure 5.3-3. Modules/peripherals that can work with GDMA

These modules/peripherals can access any memory available to GDMA. For more information, please refer to Chapter 4 *GDMA Controller (GDMA)*.

**Note:**
When accessing a memory via GDMA, a corresponding access permission is needed, otherwise this access may fail. For more information about permission control, please refer to Chapter 17 *Permission Control (PMS)*.

### 5.3.5 Modules/Peripherals Address Mapping

Table 5.3-2 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 5.3-2. Module/Peripheral Address Mapping

| Target | Boundary Address | | Size (KB) |
|---|---|---|---|
| | Low Address | High Address | |
| UART Controller 0 (UART0) | 0x6000_0000 | 0x6000_0FFF | 4 |
| UART Controller 1 (UART1) | 0x6000_1000 | 0x6000_1FFF | 4 |
| External Memory Encryption and Decryption (XTS_AES) [1] | 0x6000_2000 | 0x6000_2FFF | 4 |
| SPI Controller 0 (SPI0) [1] | 0x6000_2000 | 0x6000_2FFF | 4 |
| SPI Controller 1 (SPI1) | 0x6000_3000 | 0x6000_3FFF | 4 |
| I2C Controller (I2C) | 0x6000_4000 | 0x6000_4FFF | 4 |
| Reserved | 0x6000_5000 | 0x6000_5FFF | |
| UART Controller 2 (UART2) | 0x6000_6000 | 0x6000_6FFF | 4 |
| LED PWM Controller (LEDC) | 0x6000_7000 | 0x6000_7FFF | 4 |
| Timer Group 0 (TIMG0) | 0x6000_8000 | 0x6000_8FFF | 4 |
| Timer Group 1 (TIMG1) | 0x6000_9000 | 0x6000_9FFF | 4 |
| System Timer (SYSTIMER) | 0x6000_A000 | 0x6000_AFFF | 4 |
| Reserved | 0x6000_B000 | 0x6000_BFFF | |
| I2S Controller (I2S) | 0x6000_C000 | 0x6000_CFFF | 4 |
| Reserved | 0x6000_D000 | 0x6000_DFFF | |
| SAR ADC | 0x6000_E000 | 0x6000_EFFF | 4 |
| USB Serial/JTAG Controller (USB_SERIAL_JTAG) | 0x6000_F000 | 0x6000_FFFF | 4 |
| Interrupt Matrix | 0x6001_0000 | 0x6001_0FFF | 4 |
| Reserved | 0x6001_1000 | 0x6001_2FFF | |
| Event Task Matrix (SOC_ETM) | 0x6001_3000 | 0x6001_3FFF | 4 |
| Reserved | 0x6001_4000 | 0x6001_9FFF | |
| Memory Access Monitor 2 (PSARM_MEM_MONITOR) [2] | 0x6001_A000 | 0x6001_AFFF | 4 |
| Reserved | 0x6001_B000 | 0x6007_FFFF | |
| General DMA Controller (GDMA) | 0x6008_0000 | 0x6008_0FFF | 4 |
| General Purpose SPI2 Controller (GP-SPI2) | 0x6008_1000 | 0x6008_1FFF | 4 |
| Reserved | 0x6008_2000 | 0x6008_8FFF | |
| SHA Accelerator (SHA) | 0x6008_9000 | 0x6008_9FFF | 4 |
| Reserved | 0x6008_A000 | 0x6008_AFFF | |
| ECC Accelerator (ECC) | 0x6008_B000 | 0x6008_BFFF | 4 |
| Reserved | 0x6008_C000 | 0x6008_DFFF | |
| ECDSA Digital Signature Peripheral (ECDSA_DS) | 0x6008_E000 | 0x6008_EFFF | 4 |
| Reserved | 0x6008_F000 | 0x6008_FFFF | |
| IO MUX | 0x6009_0000 | 0x6009_0FFF | 4 |
| GPIO Matrix | 0x6009_1000 | 0x6009_1FFF | 4 |
| Memory Access Monitor 1 (TCM_MEM_MONITOR) [2] | 0x6009_2000 | 0x6009_2FFF | 4 |

Cont'd on next page

Table 5.3-2 – cont'd from previous page

| Target | Boundary Address | | Size (KB) |
|---|---|---|---|
| | Low Address | High Address | |
| Power Always-on Unit (PAU) | 0x6009_3000 | 0x6009_3FFF | 4 |
| Reserved | 0x6009_4000 | 0x6009_4FFF | |
| HP System Registers (HP_SYSREG) | 0x6009_5000 | 0x6009_5FFF | 4 |
| Power/Clock/Reset Registers (PCR) | 0x6009_6000 | 0x6009_6FFF | 4 |
| Reserved | 0x6009_7000 | 0x6009_7FFF | |
| Trusted Execution Environment (TEE) [2] | 0x6009_8000 | 0x6009_8FFF | 4 |
| High-Performance Access Permission Management (HP_APM) | 0x6009_9000 | 0x6009_9FFF | 4 |
| CPU Access Permission Management (CPU_APM) | 0x6009_A000 | 0x6009_AFFF | 4 |
| Reserved | 0x6009_B000 | 0x600A_FFFF | |
| Power Management Unit (PMU) | 0x600B_0000 | 0x600B_03FF | 1 |
| Low-Power Clock/Reset Registers (LP_CLKRST) | 0x600B_0400 | 0x600B_07FF | 1 |
| Reserved | 0x600B_0800 | 0x600B_0BFF | |
| Low-Power Timer (LP_TIMER) | 0x600B_0C00 | 0x600B_0FFF | 1 |
| Low-Power Always-on Registers (LP_AON) | 0x600B_1000 | 0x600B_13FF | 1 |
| Reserved | 0x600B_1400 | 0x600B_1BFF | |
| Low-Power Watchdog Timer (LP_WDT) | 0x600B_1C00 | 0x600B_1FFF | 1 |
| Reserved | 0x600B_2000 | 0x600B_23FF | |
| Low-Power Peripherals (LPPERI) | 0x600B_2800 | 0x600B_2BFF | 1 |
| Low-Power Analog Peripherals (LP_ANA_PERI) | 0x600B_2C00 | 0x600B_2FFF | 1 |
| Reserved | 0x600B_3000 | 0x600B_37FF | |
| Low-Power Access Permission Management (LP_APM) [2] | 0x600B_3800 | 0x600B_3BFF | 1 |
| Reserved | 0x600B_3C00 | 0x600B_3FFF | |
| Low-Power IO MUX (LP_IO_MUX) | 0x600B_4000 | 0x600B_43FF | 1 |
| Low-Power GPIO Matrix (LP_GPIO) | 0x600B_4400 | 0x600B_47FF | 1 |
| eFuse Controller | 0x600B_4800 | 0x600B_4FFF | 2 |
| Reserved | 0x600B_5000 | 0x600B_FFFF | |
| RISC-V Trace Encoder (TRACE) | 0x600C_0000 | 0x600C_0FFF | 4 |
| Reserved | 0x600C_1000 | 0x600C_1FFF | |
| Bus Access Monitor (BUS_MONITOR) [2] | 0x600C_2000 | 0x600C_2FFF | 4 |
| Reserved | 0x600C_3000 | 0x600C_4FFF | |
| Interrupt Priority Registers (INTPRI) | 0x600C_5000 | 0x600C_5FFF | 4 |
| Reserved | 0x600C_6000 | 0x600C_7FFF | |
| Cache | 0x600C_8000 | 0x600C_8FFF | 4 |
| Reserved | 0x600C_9000 | 0x600C_FFFF | |

[1] The registers of External Memory Encryption and Decryption (XTS_AES) module share the same address range of 0x6000_2000 ~ 0x6000_2FFF with SPI0 controller.

[2] The address space of this module/peripheral is not continuous.

# Chapter 6

# eFuse Controller (EFUSE)

## 6.1 Overview

ESP32-C61 contains a 4096-bit eFuse memory to store parameters and user data. The parameters include control settings for some hardware modules, system data parameters, and keys used for the 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.

## 6.2 Feature List

- 4096-bit one-time programmable storage with 1792 reserved bits for users
- Configurable write protection
- Configurable read protection
- Various hardware encoding schemes against data corruption

## 6.3 Architectural Overview

The eFuse system consists of the eFuse controller and eFuse memory. Data flow in this system is shown in Figure 6.3-1.

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 steps, please refer to Section 6.4.1.

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 6.3.3 for details), and send the error message to the error report register. For detailed steps to read parameters, please refer to the Section 6.4.2.

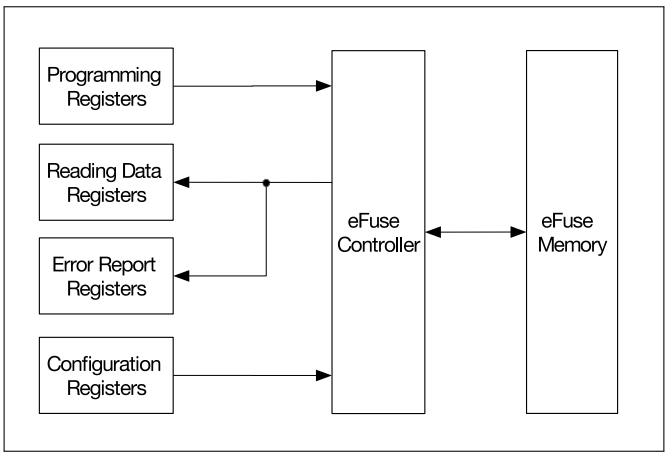



Figure 6.3-1. Data Flow in eFuse

The eFuse memory has 11 data blocks. (BLOCK0 ~ BLOCK10).

BLOCK0 holds most parameters for software and hardware uses.

Table 6.3-1 lists all the parameters accessible (readable and usable) to users in BLOCK0 and their offsets, bit widths, accessibility by hardware, write protection, and brief function description. For more description on the parameters, please click the link of the corresponding parameter in the table.

The **EFUSE_WR_DIS** parameter is used to enable write protection of other parameters. **EFUSE_RD_DIS** is used to enable read protection of BLOCK4 ~ BLOCK10. For more information on these two parameters, please see Section 6.3.1 and Section 6.3.2.

Table 6.3-1. 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 programming of individual eFuse memory bit is disabled or enabled. |
| **EFUSE_RD_DIS** | 7 | Y | 0 | Represents whether reading of individual eFuse block (BLOCK4 ~ BLOCK10) is disabled or enabled. |
| EFUSE_DIS_ICACHE | 1 | Y | 2 | Represents whether iCache is disabled or enabled. |
| EFUSE_DIS_USB_JTAG | 1 | Y | 2 | Represents whether the function of USB-to-JTAG is disabled or enabled. |
| EFUSE_DIS_USB_SERIAL_JTAG | 1 | Y | 2 | Represents whether USB-Serial-JTAG is disabled or enabled. |
| EFUSE_DIS_FORCE_DOWNLOAD | 1 | Y | 2 | Represents whether the function that forces chip into download mode is disabled or enabled. |
| EFUSE_SPI_DOWNLOAD_MSPI_DIS | 1 | Y | 17 | Represents whether SPI0 controller is disabled during boot_mode_download. |
| EFUSE_JTAG_SEL_ENABLE | 1 | Y | 2 | Represents whether the selection between usb_to_jtag and pad_to_jtag through strapping GPIO15 is disabled when both EFUSE_DIS_PAD_JTAG and EFUSE_DIS_USB_JTAG are configured to 0. |
| EFUSE_DIS_PAD_JTAG | 1 | Y | 2 | Represents whether JTAG is disabled in the hard way (permanently). |
| EFUSE_DIS_DOWNLOAD_MANUAL_ENCRYPT | 1 | Y | 2 | Represents whether flash encrypt function is disabled or enabled (except in SPI boot mode). |
| EFUSE_USB_EXCHG_PINS | 1 | Y | 30 | Represents whether the D+ and D- pins is exchanged. |
| EFUSE_VDD_SPI_AS_GPIO | 1 | Y | 30 | Represents whether VDD_SPI pin is used as a regular GPIO. |
| EFUSE_WDT_DELAY_SEL | 2 | Y | 3 | Represents the threshold level of the RTC watchdog STG0 timeout. |

Cont'd on next page



GoBack

Table 6.3-1 – cont'd from previous page

| Parameters | Bit Width | Accessible by Hardware | Write Protection by EFUSE_WR_DIS Bit Number | Description |
|---|---|---|---|---|
| EFUSE_SPI_BOOT_CRYPT_CNT | 3 | Y | 4 | Represents whether SPI boot encryption/decryption is enabled. |
| EFUSE_SECURE_BOOT_KEY_REVOKE0 | 1 | N | 5 | Represents whether revoking first secure boot key is enabled. |
| EFUSE_SECURE_BOOT_KEY_REVOKE1 | 1 | N | 6 | Represents whether revoking second secure boot key is enabled. |
| EFUSE_SECURE_BOOT_KEY_REVOKE2 | 1 | N | 7 | Represents whether revoking third secure boot key is enabled. |
| EFUSE_KEY_PURPOSE_0 | 4 | Y | 8 | Represents the purpose of Key0. |
| EFUSE_KEY_PURPOSE_1 | 4 | Y | 9 | Represents the purpose of Key1. |
| EFUSE_KEY_PURPOSE_2 | 4 | Y | 10 | Represents the purpose of Key2. |
| EFUSE_KEY_PURPOSE_3 | 4 | Y | 11 | Represents the purpose of Key3. |
| EFUSE_KEY_PURPOSE_4 | 4 | Y | 12 | Represents the purpose of Key4. |
| EFUSE_KEY_PURPOSE_5 | 4 | Y | 13 | Represents the purpose of Key5. |
| EFUSE_SEC_DPA_LEVEL | 2 | Y | 14 | Represents whether to determine DPA secure level by configuring the clock random frequency dividing mode. |
| EFUSE_SECURE_BOOT_EN | 1 | N | 16 | Represents whether Secure Boot is enabled. |
| EFUSE_SECURE_BOOT_AGGRESSIVE_REVOKE | 1 | N | 16 | Represents whether aggressive revocation of Secure Boot is enabled. |
| EFUSE_FLASH_TPUW | 4 | N | 18 | Represents the flash waiting time after power-up, in unit of ms. When the value less than 15, the waiting time is the programmed value. Otherwise, the waiting time is 2 times the programmed value. |
| EFUSE_DIS_DOWNLOAD_MODE | 1 | N | 18 | Represents whether download mode is disabled. |
| EFUSE_DIS_DIRECT_BOOT | 1 | N | 18 | Represents whether direct boot mode is disabled. |
| EFUSE_DIS_USB_SERIAL_JTAG_ROM_PRINT | 1 | N | 18 | Represents whether print from USB-Serial-JTAG is disabled. |

Cont'd on next page



GoBack

Table 6.3-1 – cont'd from previous page

| Parameters | Bit Width | Accessible by Hardware | Write Protection by EFUSE_WR_DIS Bit Number | Description |
|---|---|---|---|---|
| EFUSE_DIS_USB_SERIAL_JTAG_DOWNLOAD_MODE | 1 | N | 18 | Represents whether the USB-Serial-JTAG download function is disabled. |
| EFUSE_ENABLE_SECURITY_DOWNLOAD | 1 | N | 18 | Represents whether security download is enabled. |
| EFUSE_UART_PRINT_CONTROL | 2 | N | 18 | Represents the types of UART printing. |
| EFUSE_FORCE_SEND_RESUME | 1 | N | 18 | Represents whether ROM code is forced to send a resume command during SPI boot. |
| EFUSE_SECURE_VERSION | 16 | N | 18 | Represents the security version used by ESP-IDF anti-rollback feature. |
| EFUSE_SECURE_BOOT_DISABLE_FAST_WAKE | 1 | N | 19 | Represents whether FAST_VERIFY_ON_WAKE is disabled when Secure Boot is enabled. |

Table 6.3-2 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 6.3-2. Secure Key Purpose Values**

| Key Purpose Values | Purposes |
|---|---|
| 0 | User purposes |
| 1 | Reserved |
| 2 | Reserved |
| 3 | Reserved |
| 4 | XTS_AES_128_KEY (flash/SRAM encryption and decryption) |
| 5 | HMAC Downstream mode (both JTAG and DS) |
| 6 | JTAG in HMAC Downstream mode |
| 7 | Digital Signature peripheral in HMAC Downstream mode |
| 8 | HMAC Upstream mode |
| 9 | SECURE_BOOT_DIGEST0 (secure boot key digest) |
| 10 | SECURE_BOOT_DIGEST1 (secure boot key digest) |
| 11 | SECURE_BOOT_DIGEST2 (secure boot key digest) |

Table 6.3-3 provides the details of parameters in BLOCK1 ~ BLOCK10.

Table 6.3-3. 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 | N | 20 | N/A | MAC address |
| | EFUSE_MAC_EXT | 16 | N | 20 | N/A | Extended MAC address |
| | EFUSE_SYS_DATA_PART0 | 69 | N | 20 | N/A | 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 | 256 | N | 29 | 6 | System data |

Among these blocks, BLOCK4 ~ 9 can 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 6.3-2). 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.

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 6.3.3 and Section 6.4.1.

### 6.3.1 EFUSE_WR_DIS

Parameter EFUSE_WR_DIS determines whether individual eFuse parameters are write-protected. After EFUSE_WR_DIS programming, execute an eFuse read operation for the new values to take effect.

Column "Write Protection by EFUSE_WR_DIS Bit Number" in Table 6.3-1 and Table 6.3-3 list the specific bits in EFUSE_WR_DIS that disable writing.

When the write protection bit of a parameter is set to 0, 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, the 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.

### 6.3.2 EFUSE_RD_DIS

Only the parameters in BLOCK4 ~ BLOCK10 can be set to be read-protected from users, as shown in column "Read Protection by EFUSE_RD_DIS Bit Number" of Table 6.3-3. After EFUSE_RD_DIS programming, 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 outside 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.

### 6.3.3 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 other parameters takes 152 bits each. So, the eFuse memory space occupied by BLOCK0 is 32 + 152 * 4 = 640 bits.

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 $p(x) = x^8 + x^4 + x^3 + x^2 + 1$.

The shift register circuit shown in Figure 6.3-2 and 6.3-3 processes 32 data bytes using RS (44, 32). This coding scheme encodes 32 bytes of data into 44 bytes:

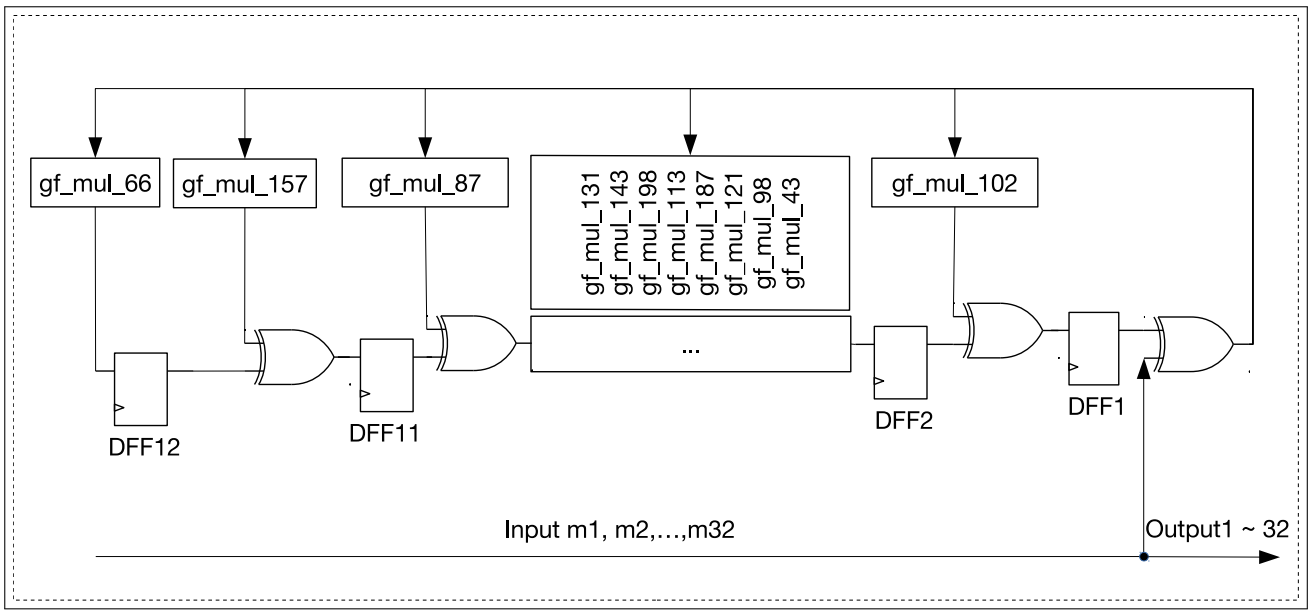



Figure 6.3-2. Shift Register Circuit (first 32 output)

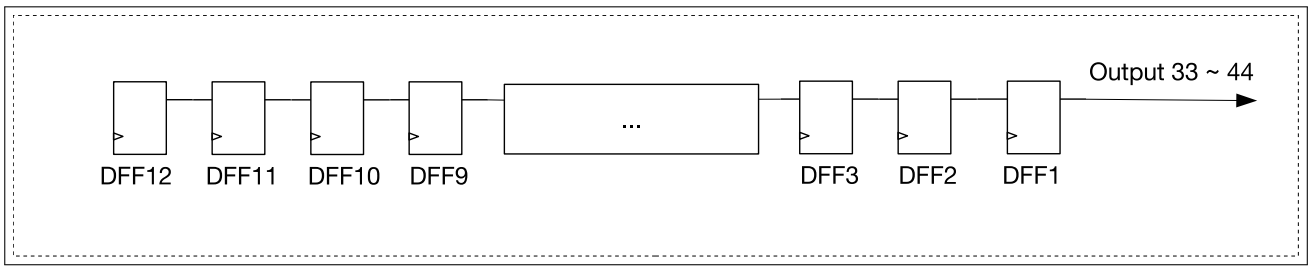



Figure 6.3-3. Shift Register Circuit (last 12 output)

- Bytes [0:31] are the data bytes itself
- 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 $\alpha^n$, 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.

The parameter in other blocks (Block2 ~ 10) is 32 bytes respectively, and the RS check code is 12 bytes, so they occupy (32 + 12) * 9 = 396 bytes in eFuse memory.

## 6.4 Functional Description

### 6.4.1 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.

There is a one-to-one correspondence between the reading data registers and the programming data registers (see table 6.4-1 for details). Users can determine the location of the data to be programmed in programming registers by checking the parameter description and the corresponding read registers.

For example, if the user wants to program the parameter EFUSE_DIS_ICACHE 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 completing these steps, 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.
- **Programming BLOCK1**
  1. Set EFUSE_BLK_NUM to 1.
  2. Write into EFUSE_PGM_DATA0_REG ~ EFUSE_PGM_DATA5_REG the data to be programmed to BLOCK1. Write into EFUSE_PGM_CHECK_VALUE0_REG ~ EFUSE_PGM_CHECK_VALUE2_REG the corresponding RS check code.
     The data in EFUSE_PGM_DATA6_REG ~ EFUSE_PGM_DATA7_REG does not affect the programming of BLOCK1. When calculating RS check of BLOCK1 using software, please treat the 8 bytes as 0.
- **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 6.4.3.

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 6.4.2.
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 6.4.2) to update eFuse registers with the new values.
9. Check error record registers. If the values read in error record registers are not 0, the programming process should be performed again following above steps 1 ~ 7. Please check the following error record registers for different eFuse blocks:
    - 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_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, each bit can be programmed separately. However, we recommend minimizing programming cycles by programming all bits of a parameter in one action. In addition, after all parameters controlled by a certain bit of EFUSE_WR_DIS are programmed, that bit should be immediately programmed. The programming of parameters controlled by a certain bit of EFUSE_WR_DIS, and the programming of the bit itself can even be completed at the same time in one programming 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.

### 6.4.2 Reading of Parameters

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 6.4-1.

Table 6.4-1. Registers Information

| BLOCK | Read Registers | Registers When Programming This Block |
|---|---|---|
| 0 | EFUSE_RD_WR_DIS0_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_0 ~ 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_0 ~ 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.

#### Error detection

Error record registers allow users to detect if there is any inconsistency between the parameter read by eFuse controller and that in eFuse memory.

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 program/read operation

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 10 *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.

### 6.4.3 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 $\mu$s.
- EFUSE_PWR_ON_NUM (the power-up 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 times EFUSE_DAC_NUM.
- EFUSE_PWR_OFF_NUM (the power-out time for VDDQ): The value of this parameter should be larger than 10 $\mu$s.

**Table 6.4-2. 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 |

### 6.4.4 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 6.3-1 and Table 6.3-3. Users cannot intervene in this process.

## 6.5 Interrupts

- PGM_DONE interrupt: Triggered when eFuse programming has finished. To enable this interrupt, set the EFUSE_PGM_DONE_INT_ENA field of register EFUSE_INT_ENA_REG to 1.
- READ_DONE interrupt: Triggered when eFuse reading has finished. To enable this interrupt, set the EFUSE_READ_DONE_INT_ENA field of register EFUSE_INT_ENA_REG to 1.

## 6.6 Register Summary

The addresses in this section are relative to eFuse controller base address provided in Table 5.3-2 in Chapter 5 *System and Memory*.

| Name | Description | Address | Access |
|---|---|---|---|
| **Programming Data Register** | | | |
| EFUSE_PGM_DATA0_REG | Register 0 that stores data to be programmed | 0x0000 | R/W |
| EFUSE_PGM_DATA1_REG | Register 1 that stores data to be programmed | 0x0004 | R/W |
| EFUSE_PGM_DATA2_REG | Register 2 that stores data to be programmed | 0x0008 | R/W |
| EFUSE_PGM_DATA3_REG | Register 3 that stores data to be programmed | 0x000C | R/W |
| EFUSE_PGM_DATA4_REG | Register 4 that stores data to be programmed | 0x0010 | R/W |
| EFUSE_PGM_DATA5_REG | Register 5 that stores data to be programmed | 0x0014 | R/W |
| EFUSE_PGM_DATA6_REG | Register 6 that stores data to be programmed | 0x0018 | R/W |
| EFUSE_PGM_DATA7_REG | Register 7 that stores data to be programmed | 0x001C | R/W |
| EFUSE_PGM_CHECK_VALUE0_REG | Register 0 that stores the RS code to be programmed | 0x0020 | R/W |
| EFUSE_PGM_CHECK_VALUE1_REG | Register 1 that stores the RS code to be programmed | 0x0024 | R/W |
| EFUSE_PGM_CHECK_VALUE2_REG | Register 2 that stores the RS code to be programmed | 0x0028 | R/W |
| **Reading Data Register** | | | |
| EFUSE_RD_WR_DIS0_REG | Register 0 of BLOCK0 | 0x002C | RO |
| EFUSE_RD_REPEAT_DATA0_REG | Register 1 of BLOCK0 | 0x0030 | RO |
| EFUSE_RD_REPEAT_DATA1_REG | Register 2 of BLOCK0 | 0x0034 | RO |
| EFUSE_RD_REPEAT_DATA2_REG | Register 3 of BLOCK0 | 0x0038 | RO |
| EFUSE_RD_REPEAT_DATA3_REG | Register 4 of BLOCK0 | 0x003C | RO |
| EFUSE_RD_REPEAT_DATA4_REG | Register 5 of BLOCK0 | 0x0040 | RO |
| EFUSE_RD_MAC_SYS0_REG | Register 0 of BLOCK1 | 0x0044 | RO |
| EFUSE_RD_MAC_SYS1_REG | Register 1 of BLOCK1 | 0x0048 | RO |
| EFUSE_RD_MAC_SYS2_REG | Register 2 of BLOCK1 | 0x004C | RO |
| EFUSE_RD_MAC_SYS3_REG | Register 3 of BLOCK1 | 0x0050 | RO |
| EFUSE_RD_MAC_SYS4_REG | Register 4 of BLOCK1 | 0x0054 | RO |
| EFUSE_RD_MAC_SYS5_REG | Register 5 of BLOCK1 | 0x0058 | RO |
| EFUSE_RD_SYS_PART1_DATA0_REG | Register 0 of BLOCK2 (system) | 0x005C | RO |
| EFUSE_RD_SYS_PART1_DATA1_REG | Register 1 of BLOCK2 (system) | 0x0060 | RO |
| EFUSE_RD_SYS_PART1_DATA2_REG | Register 2 of BLOCK2 (system) | 0x0064 | RO |
| EFUSE_RD_SYS_PART1_DATA3_REG | Register 3 of BLOCK2 (system) | 0x0068 | RO |
| EFUSE_RD_SYS_PART1_DATA4_REG | Register 4 of BLOCK2 (system) | 0x006C | RO |
| EFUSE_RD_SYS_PART1_DATA5_REG | Register 5 of BLOCK2 (system) | 0x0070 | RO |
| EFUSE_RD_SYS_PART1_DATA6_REG | Register 6 of BLOCK2 (system) | 0x0074 | RO |
| EFUSE_RD_SYS_PART1_DATA7_REG | Register 7 of BLOCK2 (system) | 0x0078 | RO |
| EFUSE_RD_USR_DATA0_REG | Register 0 of BLOCK3 (user) | 0x007C | RO |
| EFUSE_RD_USR_DATA1_REG | Register 1 of BLOCK3 (user) | 0x0080 | RO |
| EFUSE_RD_USR_DATA2_REG | Register 2 of BLOCK3 (user) | 0x0084 | RO |

| Name | Description | Address | Access |
|---|---|---|---|
| EFUSE_RD_USR_DATA3_REG | Register 3 of BLOCK3 (user) | 0x0088 | RO |
| EFUSE_RD_USR_DATA4_REG | Register 4 of BLOCK3 (user) | 0x008C | RO |
| EFUSE_RD_USR_DATA5_REG | Register 5 of BLOCK3 (user) | 0x0090 | RO |
| EFUSE_RD_USR_DATA6_REG | Register 6 of BLOCK3 (user) | 0x0094 | RO |
| EFUSE_RD_USR_DATA7_REG | Register 7 of BLOCK3 (user) | 0x0098 | RO |
| EFUSE_RD_KEY0_DATA0_REG | Register 0 of BLOCK4 (KEY0) | 0x009C | RO |
| EFUSE_RD_KEY0_DATA1_REG | Register 1 of BLOCK4 (KEY0) | 0x00A0 | RO |
| EFUSE_RD_KEY0_DATA2_REG | Register 2 of BLOCK4 (KEY0) | 0x00A4 | RO |
| EFUSE_RD_KEY0_DATA3_REG | Register 3 of BLOCK4 (KEY0) | 0x00A8 | RO |
| EFUSE_RD_KEY0_DATA4_REG | Register 4 of BLOCK4 (KEY0) | 0x00AC | RO |
| EFUSE_RD_KEY0_DATA5_REG | Register 5 of BLOCK4 (KEY0) | 0x00B0 | RO |
| EFUSE_RD_KEY0_DATA6_REG | Register 6 of BLOCK4 (KEY0) | 0x00B4 | RO |
| EFUSE_RD_KEY0_DATA7_REG | Register 7 of BLOCK4 (KEY0) | 0x00B8 | RO |
| EFUSE_RD_KEY1_DATA0_REG | Register 0 of BLOCK5 (KEY1) | 0x00BC | RO |
| EFUSE_RD_KEY1_DATA1_REG | Register 1 of BLOCK5 (KEY1) | 0x00C0 | RO |
| EFUSE_RD_KEY1_DATA2_REG | Register 2 of BLOCK5 (KEY1) | 0x00C4 | RO |
| EFUSE_RD_KEY1_DATA3_REG | Register 3 of BLOCK5 (KEY1) | 0x00C8 | RO |
| EFUSE_RD_KEY1_DATA4_REG | Register 4 of BLOCK5 (KEY1) | 0x00CC | RO |
| EFUSE_RD_KEY1_DATA5_REG | Register 5 of BLOCK5 (KEY1) | 0x00D0 | RO |
| EFUSE_RD_KEY1_DATA6_REG | Register 6 of BLOCK5 (KEY1) | 0x00D4 | RO |
| EFUSE_RD_KEY1_DATA7_REG | Register 7 of BLOCK5 (KEY1) | 0x00D8 | RO |
| EFUSE_RD_KEY2_DATA0_REG | Register 0 of BLOCK6 (KEY2) | 0x00DC | RO |
| EFUSE_RD_KEY2_DATA1_REG | Register 1 of BLOCK6 (KEY2) | 0x00E0 | RO |
| EFUSE_RD_KEY2_DATA2_REG | Register 2 of BLOCK6 (KEY2) | 0x00E4 | RO |
| EFUSE_RD_KEY2_DATA3_REG | Register 3 of BLOCK6 (KEY2) | 0x00E8 | RO |
| EFUSE_RD_KEY2_DATA4_REG | Register 4 of BLOCK6 (KEY2) | 0x00EC | RO |
| EFUSE_RD_KEY2_DATA5_REG | Register 5 of BLOCK6 (KEY2) | 0x00F0 | RO |
| EFUSE_RD_KEY2_DATA6_REG | Register 6 of BLOCK6 (KEY2) | 0x00F4 | RO |
| EFUSE_RD_KEY2_DATA7_REG | Register 7 of BLOCK6 (KEY2) | 0x00F8 | RO |
| EFUSE_RD_KEY3_DATA0_REG | Register 0 of BLOCK7 (KEY3) | 0x00FC | RO |
| EFUSE_RD_KEY3_DATA1_REG | Register 1 of BLOCK7 (KEY3) | 0x0100 | RO |
| EFUSE_RD_KEY3_DATA2_REG | Register 2 of BLOCK7 (KEY3) | 0x0104 | RO |
| EFUSE_RD_KEY3_DATA3_REG | Register 3 of BLOCK7 (KEY3) | 0x0108 | RO |
| EFUSE_RD_KEY3_DATA4_REG | Register 4 of BLOCK7 (KEY3) | 0x010C | RO |
| EFUSE_RD_KEY3_DATA5_REG | Register 5 of BLOCK7 (KEY3) | 0x0110 | RO |
| EFUSE_RD_KEY3_DATA6_REG | Register 6 of BLOCK7 (KEY3) | 0x0114 | RO |
| EFUSE_RD_KEY3_DATA7_REG | Register 7 of BLOCK7 (KEY3) | 0x0118 | RO |
| EFUSE_RD_KEY4_DATA0_REG | Register 0 of BLOCK8 (KEY4) | 0x011C | RO |
| EFUSE_RD_KEY4_DATA1_REG | Register 1 of BLOCK8 (KEY4) | 0x0120 | RO |
| EFUSE_RD_KEY4_DATA2_REG | Register 2 of BLOCK8 (KEY4) | 0x0124 | RO |
| EFUSE_RD_KEY4_DATA3_REG | Register 3 of BLOCK8 (KEY4) | 0x0128 | RO |
| EFUSE_RD_KEY4_DATA4_REG | Register 4 of BLOCK8 (KEY4) | 0x012C | RO |
| EFUSE_RD_KEY4_DATA5_REG | Register 5 of BLOCK8 (KEY4) | 0x0130 | RO |

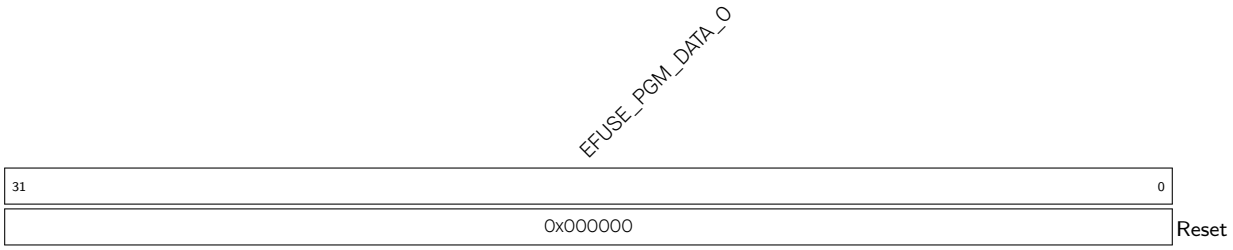
| Name | Description | Address | Access |
|---|---|---|---|
| EFUSE_RD_KEY4_DATA6_REG | Register 6 of BLOCK8 (KEY4) | 0x0134 | RO |
| EFUSE_RD_KEY4_DATA7_REG | Register 7 of BLOCK8 (KEY4) | 0x0138 | RO |
| EFUSE_RD_KEY5_DATA0_REG | Register 0 of BLOCK9 (KEY5) | 0x013C | RO |
| EFUSE_RD_KEY5_DATA1_REG | Register 1 of BLOCK9 (KEY5) | 0x0140 | RO |
| EFUSE_RD_KEY5_DATA2_REG | Register 2 of BLOCK9 (KEY5) | 0x0144 | RO |
| EFUSE_RD_KEY5_DATA3_REG | Register 3 of BLOCK9 (KEY5) | 0x0148 | RO |
| EFUSE_RD_KEY5_DATA4_REG | Register 4 of BLOCK9 (KEY5) | 0x014C | RO |
| EFUSE_RD_KEY5_DATA5_REG | Register 5 of BLOCK9 (KEY5) | 0x0150 | RO |
| EFUSE_RD_KEY5_DATA6_REG | Register 6 of BLOCK9 (KEY5) | 0x0154 | RO |
| EFUSE_RD_KEY5_DATA7_REG | Register 7 of BLOCK9 (KEY5) | 0x0158 | RO |
| EFUSE_RD_SYS_PART2_DATA0_REG | Register 0 of BLOCK10 (system) | 0x015C | RO |
| EFUSE_RD_SYS_PART2_DATA1_REG | Register 1 of BLOCK10 (system) | 0x0160 | RO |
| EFUSE_RD_SYS_PART2_DATA2_REG | Register 2 of BLOCK10 (system) | 0x0164 | RO |
| EFUSE_RD_SYS_PART2_DATA3_REG | Register 3 of BLOCK10 (system) | 0x0168 | RO |
| EFUSE_RD_SYS_PART2_DATA4_REG | Register 4 of BLOCK10 (system) | 0x016C | RO |
| EFUSE_RD_SYS_PART2_DATA5_REG | Register 5 of BLOCK10 (system) | 0x0170 | RO |
| EFUSE_RD_SYS_PART2_DATA6_REG | Register 6 of BLOCK10 (system) | 0x0174 | RO |
| EFUSE_RD_SYS_PART2_DATA7_REG | Register 7 of BLOCK10 (system) | 0x0178 | RO |
| **Report Register** | | | |
| EFUSE_RD_REPEAT_DATA_ERR0_REG | Programming error record register 0 of BLOCK0 | 0x017C | RO |
| EFUSE_RD_REPEAT_DATA_ERR1_REG | Programming error record register 1 of BLOCK0 | 0x0180 | RO |
| EFUSE_RD_REPEAT_DATA_ERR2_REG | Programming error record register 2 of BLOCK0 | 0x0184 | RO |
| EFUSE_RD_REPEAT_DATA_ERR3_REG | Programming error record register 3 of BLOCK0 | 0x0188 | RO |
| EFUSE_RD_REPEAT_DATA_ERR4_REG | Programming error record register 4 of BLOCK0 | 0x018C | RO |
| **RS block error report registers** | | | |
| EFUSE_RD_RS_DATA_ERR0_REG | Programming error record register 0 of BLOCK1-10 | 0x0190 | RO |
| EFUSE_RD_RS_DATA_ERR1_REG | Programming error record register 1 of BLOCK1-10 | 0x0194 | RO |
| **Version Control Register** | | | |
| EFUSE_DATE_REG | Version control register | 0x0198 | R/W |
| **Configuration Register** | | | |
| EFUSE_CLK_REG | eFuse clock configuration register | 0x01C8 | R/W |
| EFUSE_CONF_REG | eFuse operation mode configuration register | 0x01CC | R/W |
| 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 | Configures register 1 of eFuse programming timing parameters | 0x01F0 | R/W |

| Name | Description | Address | Access |
|---|---|---|---|
| EFUSE_WR_TIM_CONF2_REG | Configures register 2 of eFuse programming timing parameters | 0x01F4 | R/W |
| EFUSE_WR_TIM_CONF0_RS_BYPASS_REG | Configures register 0 of eFuse programming time parameters and rs bypass operation | 0x01F8 | varies |
| **Status Register** | | | |
| EFUSE_STATUS_REG | eFuse status register | 0x01D0 | RO |
| **Command Register** | | | |
| EFUSE_CMD_REG | eFuse command register | 0x01D4 | varies |
| **Interrupt Register** | | | |
| 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 |
| EFUSE_INT_CLR_REG | eFuse interrupt clear register | 0x01E4 | WT |

## 6.7 Registers

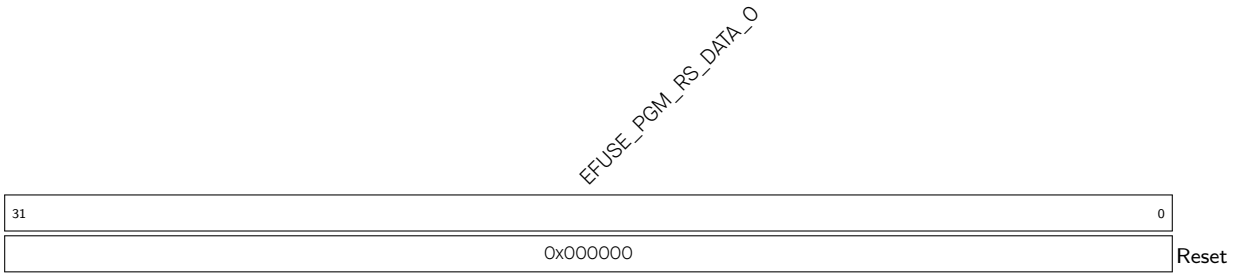
The addresses in this section are relative to eFuse controller base address provided in Table 5.3-2 in Chapter 5 *System and Memory*.

**Register 6.1. EFUSE_PGM_DATA*n*_REG (*n*: 0-7) (0x0000+0x4**n*)**



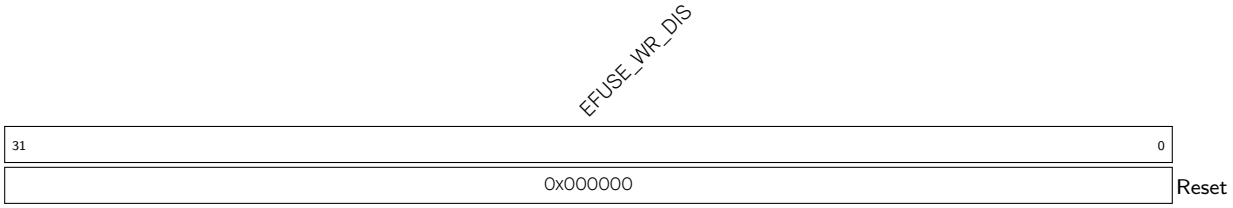

**EFUSE_PGM_DATA_*n*** Configures the *n*th 32-bit data to be programmed. (R/W)

**Register 6.2. EFUSE_PGM_CHECK_VALUE*n*_REG (*n*: 0-2) (0x0020+0x4**n*)**



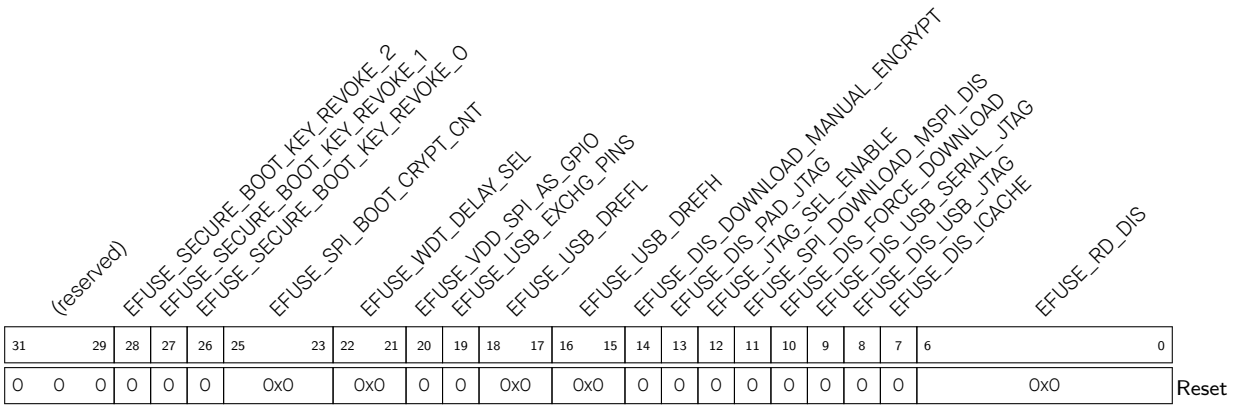

**EFUSE_PGM_RS_DATA_*n*** Configures the *n*th RS code to be programmed. (R/W)

**Register 6.3. EFUSE_RD_WR_DIS0_REG (0x002C)**




**EFUSE_WR_DIS** Represents whether programming of individual eFuse memory bit is disabled or enabled.
1: Disabled
0: Enabled
(RO)

Register 6.4. EFUSE_RD_REPEAT_DATA0_REG (0x0030)




**EFUSE_RD_DIS** Represents whether reading of individual eFuse block (BLOCK4 ~ BLOCK10) is disabled or enabled.

1: Disabled

0: Enabled

(RO)

**EFUSE_DIS_ICACHE** Represents whether iCache is disabled or enabled.

1: Disabled

0: Enabled

(RO)

**EFUSE_DIS_USB_JTAG** Represents whether the function of USB-to-JTAG is disabled or enabled.

1: Disabled

0: Enabled

(RO)

**EFUSE_DIS_USB_SERIAL_JTAG** Represents whether USB-Serial-JTAG is disabled or enabled.

1: Disabled

0: Enabled

(RO)

**EFUSE_DIS_FORCE_DOWNLOAD** Represents whether the function that forces chip into download mode is disabled or enabled.

1: Disabled

0: Enabled

(RO)

**EFUSE_SPI_DOWNLOAD_MSPI_DIS** Represents whether SPI0 controller is disabled during boot_mode_download.

1: Disabled

0: Enabled

(RO)

Continued on the next page...

Register 6.4. EFUSE_RD_REPEAT_DATA0_REG (0x0030)

Continued from the previous page...

**EFUSE_JTAG_SEL_ENABLE** Represents whether the selection between usb_to_jtag and pad_to_jtag through strapping GPIO15 is disabled when both EFUSE_DIS_PAD_JTAG and EFUSE_DIS_USB_JTAG are configured to 0.
1: Enabled
0: Disabled
(RO)

**EFUSE_DIS_PAD_JTAG** Represents whether JTAG is disabled in the hard way (permanently).
1: Disabled
0: Enabled
(RO)

**EFUSE_DIS_DOWNLOAD_MANUAL_ENCRYPT** Represents whether flash encrypt function is disabled or enabled (except in SPI boot mode).
1: Disabled
0: Enabled
(RO)

**EFUSE_USB_DREFH** Represents the single-end input threshold Vrefh, ranging from 1.76 V to 2 V, with a step size of 80 mV. (RO)

**EFUSE_USB_DREFL** Represents the single-end input threshold Vrefl, ranging from 1.76 V to 2 V, with a step size of 80 mV. (RO)

**EFUSE_USB_EXCHG_PINS** Represents whether the D+ and D- pins is exchanged.
1: Exchanged
0: Not exchanged
(RO)

**EFUSE_VDD_SPI_AS_GPIO** Represents whether VDD_SPI pin is used as a regular GPIO.
1: Used as a regular GPIO
0: Not used as a regular GPIO
(RO)

**EFUSE_WDT_DELAY_SEL** Represents the threshold level of the RTC watchdog STG0 timeout.
0: Original threshold configuration value of STG0 *2
1: Original threshold configuration value of STG0 *4
2: Original threshold configuration value of STG0 *8
3: Original threshold configuration value of STG0 *16
(RO)

**EFUSE_SPI_BOOT_CRYPT_CNT** Represents whether SPI boot encryption/decryption is enabled.
Odd count of bits with a value of 1: Enabled
Even count of bits with a value of 1: Disabled
(RO)

Continued on the next page...

Register 6.4. EFUSE_RD_REPEAT_DATA0_REG (0x0030)

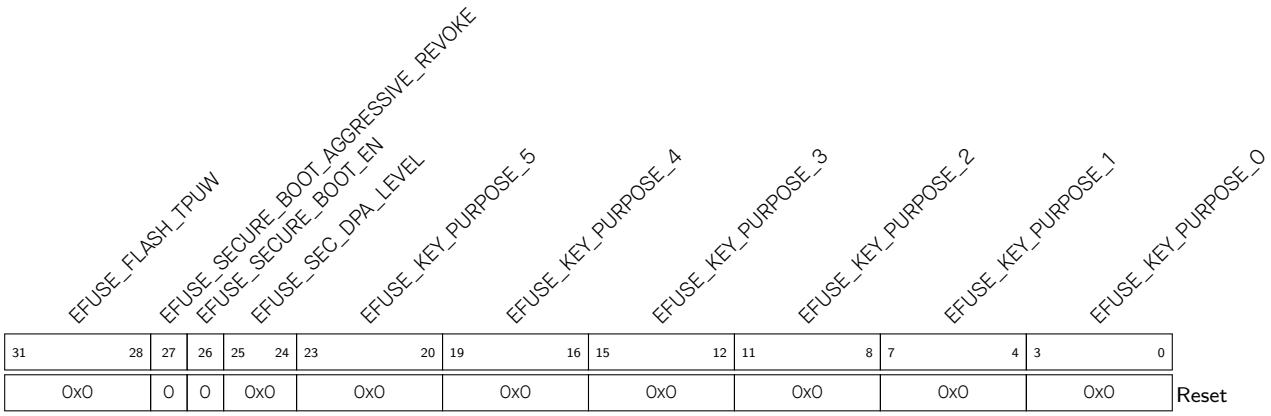
Continued from the previous page...

**EFUSE_SECURE_BOOT_KEY_REVOKE_0** Represents whether revoking first secure boot key is enabled.
1: Enabled
0: Disabled
(RO)

**EFUSE_SECURE_BOOT_KEY_REVOKE_1** Represents whether revoking second secure boot key is enabled.
1: Enabled
0: Disabled
(RO)

**EFUSE_SECURE_BOOT_KEY_REVOKE_2** Represents whether revoking third secure boot key is enabled.
1: Enabled
0: Disabled
(RO)

Register 6.5. EFUSE_RD_REPEAT_DATA1_REG (0x0034)




**EFUSE_KEY_PURPOSE_0** Represents the purpose of Key0. (RO)

**EFUSE_KEY_PURPOSE_1** Represents the purpose of Key1. (RO)

**EFUSE_KEY_PURPOSE_2** Represents the purpose of Key2. (RO)

**EFUSE_KEY_PURPOSE_3** Represents the purpose of Key3. (RO)

**EFUSE_KEY_PURPOSE_4** Represents the purpose of Key4. (RO)

**EFUSE_KEY_PURPOSE_5** Represents the purpose of Key5. (RO)

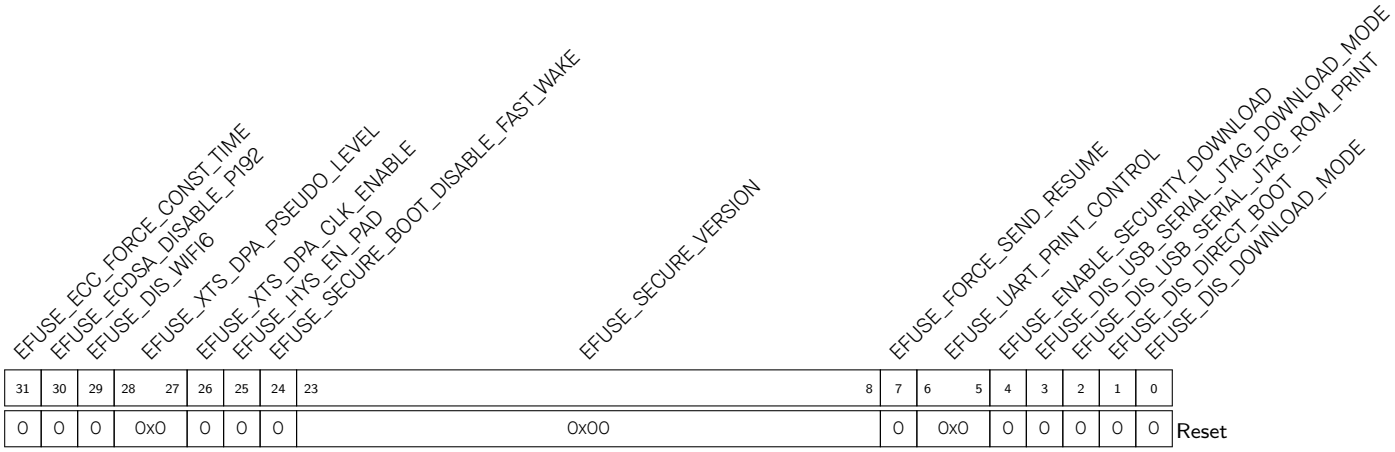
**EFUSE_SEC_DPA_LEVEL** Represents whether to determine DPA secure level by configuring the clock random frequency dividing mode.
0: No effect
1: Determine
(RO)

**EFUSE_SECURE_BOOT_EN** Represents whether Secure Boot is enabled.
1: Enabled
0: Disabled
(RO)

**EFUSE_SECURE_BOOT_AGGRESSIVE_REVOKE** Represents whether aggressive revocation of Secure Boot is enabled.
1: Enabled.
0: Disabled
(RO)

**EFUSE_FLASH_TPUW** Represents the flash waiting time after power-up, in unit of ms. When the value less than 15, the waiting time is the programmed value. Otherwise, the waiting time is 2 times the programmed value. (RO)

Register 6.6. EFUSE_RD_REPEAT_DATA2_REG (0x0038)




**EFUSE_DIS_DOWNLOAD_MODE** Represents whether download mode is disabled.

1. Disabled

0: Enabled

(RO)

**EFUSE_DIS_DIRECT_BOOT** Represents whether direct boot mode is disabled.

1. Disabled

0: Enabled

(RO)

**EFUSE_DIS_USB_SERIAL_JTAG_ROM_PRINT** Represents whether print from USB-Serial-JTAG is disabled.

1. Disabled

0: Enabled

(RO)

**EFUSE_DIS_USB_SERIAL_JTAG_DOWNLOAD_MODE** Represents whether the USB-Serial-JTAG download function is disabled.

1: Disabled

0: Enabled

(RO)

**EFUSE_ENABLE_SECURITY_DOWNLOAD** Represents whether security download is enabled.

1: Enabled

0: Disabled

(RO)

**EFUSE_UART_PRINT_CONTROL** Represents the types of UART printing.

0: Force enable printing.

1: Enable printing when GPIO8 is reset at low level.

2: Enable printing when GPIO8 is reset at high level.

3: Force disable printing.

(RO)

**Continued on the next page...**

Register 6.6. EFUSE_RD_REPEAT_DATA2_REG (0x0038)

Continued from the previous page...

**EFUSE_FORCE_SEND_RESUME** Represents whether ROM code is forced to send a resume command during SPI boot.
1: Forced.
0: Not forced.
(RO)

**EFUSE_SECURE_VERSION** Represents the security version used by ESP-IDF anti-rollback feature. (RO)

**EFUSE_SECURE_BOOT_DISABLE_FAST_WAKE** Represents whether FAST_VERIFY_ON_WAKE is disabled when Secure Boot is enabled.
1: Disabled
0: Enabled
(RO)

**EFUSE_HYS_EN_PAD** Represents whether the hysteresis function of corresponding PAD is enabled.
1: Enabled
0: Disabled
(RO)

**EFUSE_XTS_DPA_CLK_ENABLE** Represents whether anti-DPA attack clock function is enabled.
1: Enabled
0: Disabled
(RO)

**EFUSE_XTS_DPA_PSEUDO_LEVEL** Represents the anti-SCA attack pseudo function level.
3: High
2: Moderate
1: Low
0: Decided by register configuration
(RO)

**EFUSE_DIS_WIFI6** Represents whether the Wi-Fi 6 feature is enabled.
1: Disabled
0: Enabled
(RO)

**EFUSE_ECDSA_DISABLE_P192** Represents whether to disable P192 curve in ECDSA_DS.
1: Disabled
0: Enabled
(RO)

Continued on the next page...

Register 6.6. EFUSE_RD_REPEAT_DATA2_REG (0x0038)

Continued from the previous page...

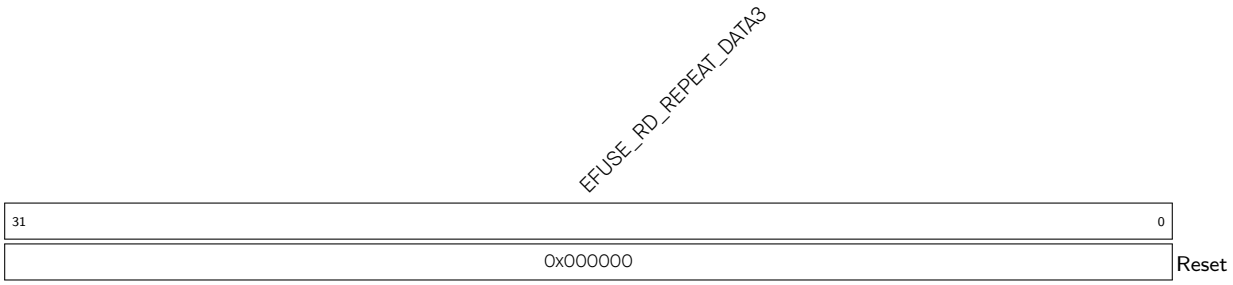
**EFUSE_ECC_FORCE_CONST_TIME** Represents whether to force ECC to use const-time calculation mode.

1: Enabled

0: Disabled

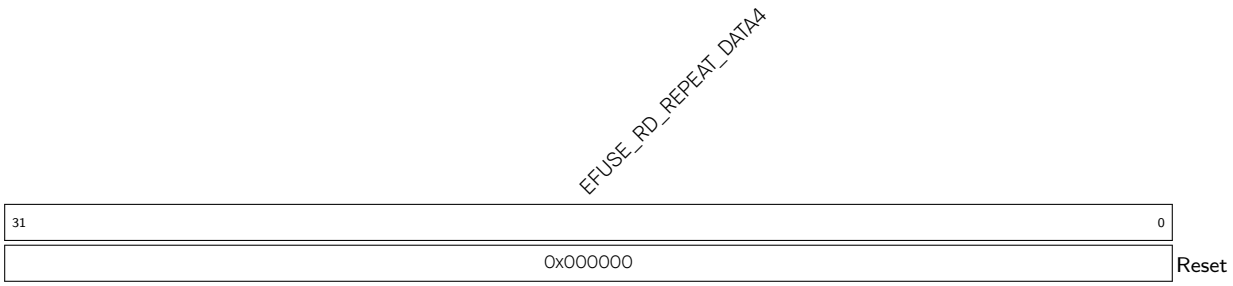
(RO)

Register 6.7. EFUSE_RD_REPEAT_DATA3_REG (0x003C)



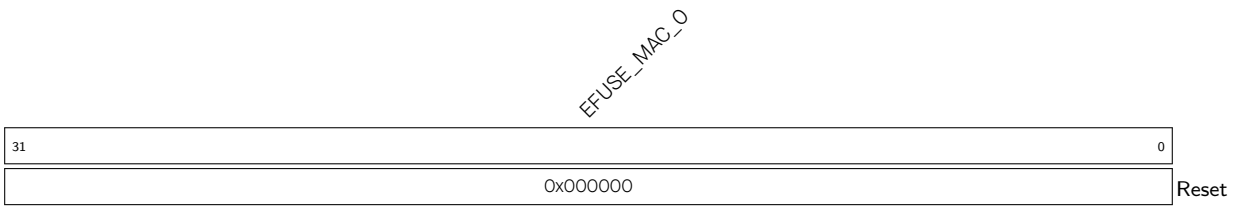

**EFUSE_RD_REPEAT_DATA3** Reserved (RO)

Register 6.8. EFUSE_RD_REPEAT_DATA4_REG (0x0040)



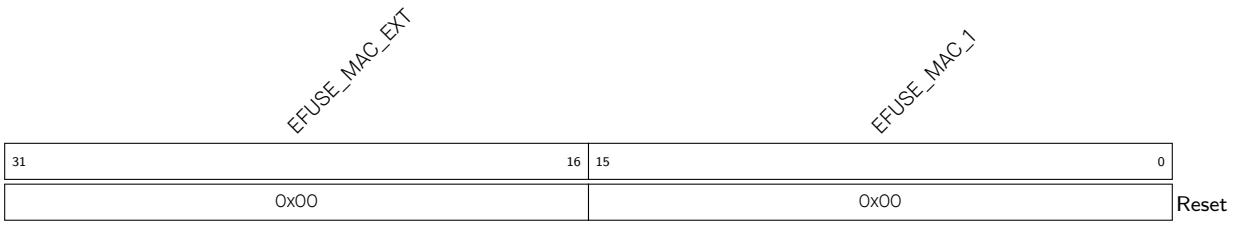

**EFUSE_RD_REPEAT_DATA4** Reserved (RO)

Register 6.9. EFUSE_RD_MAC_SYS0_REG (0x0044)




**EFUSE_MAC_0** Represents low 32-bit of MAC address. (RO)

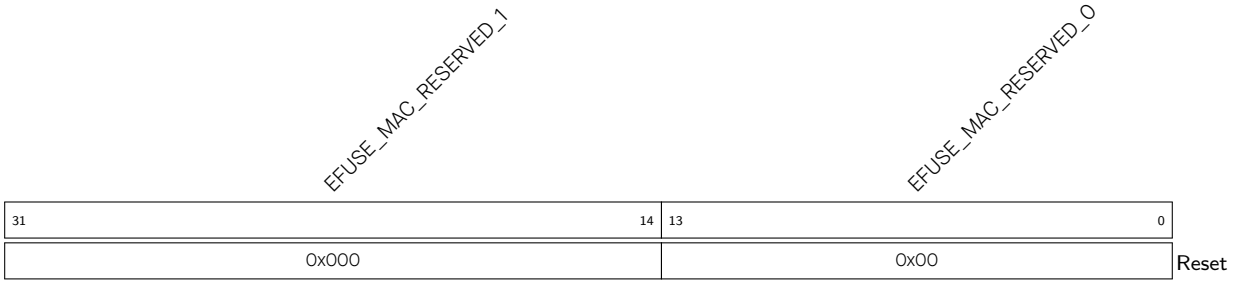
Register 6.10. EFUSE_RD_MAC_SYS1_REG (0x0048)




**EFUSE_MAC_1** Represents high 16-bit of MAC address. (RO)

**EFUSE_MAC_EXT** Represents the extended bits of MAC address. (RO)

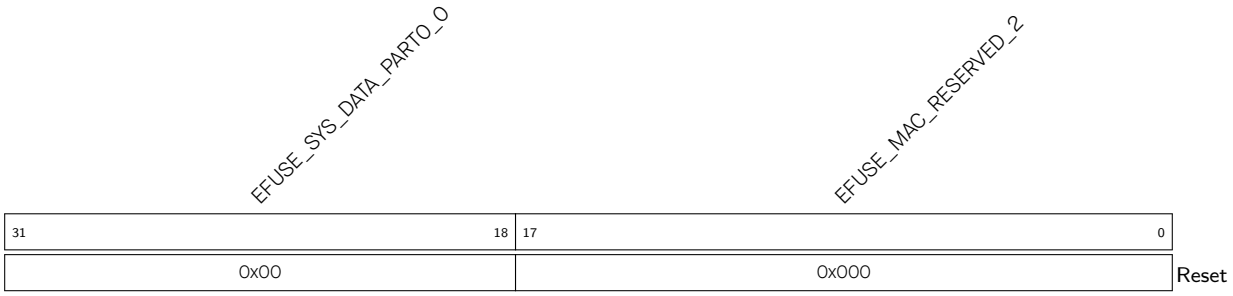
Register 6.11. EFUSE_RD_MAC_SYS2_REG (0x004C)




**EFUSE_MAC_RESERVED_0** Reserved (RO)

**EFUSE_MAC_RESERVED_1** Reserved (RO)

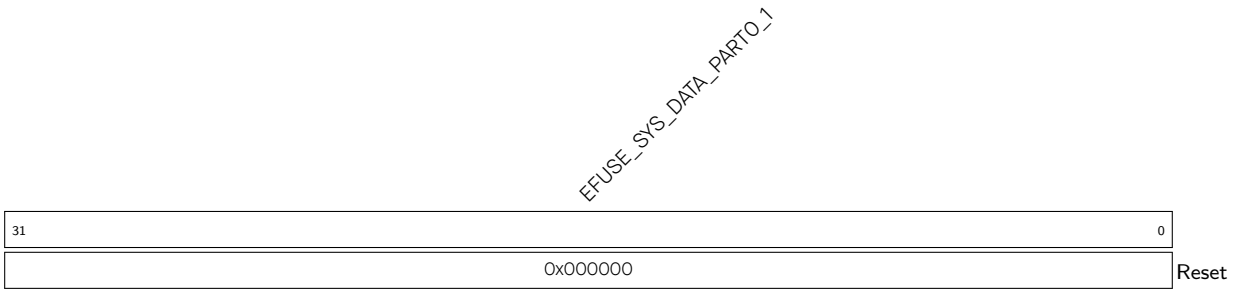
Register 6.12. EFUSE_RD_MAC_SYS3_REG (0x0050)




**EFUSE_MAC_RESERVED_2** Reserved (RO)

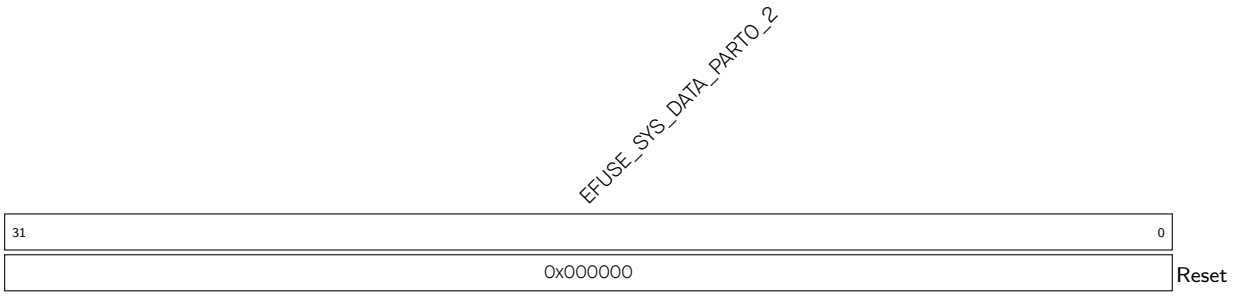
**EFUSE_SYS_DATA_PART0_0** Represents the first 14 bits of zeroth part of system data. (RO)

Register 6.13. EFUSE_RD_MAC_SYS4_REG (0x0054)



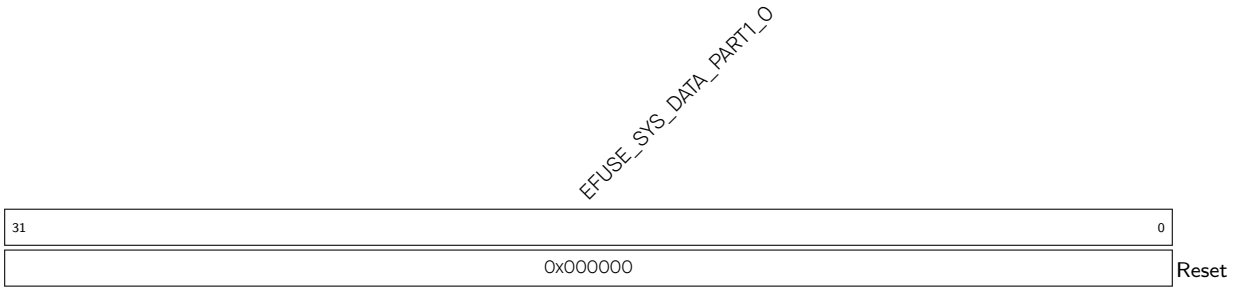

**EFUSE_SYS_DATA_PART0_1** Represents the second 32 bits of zeroth part of system data. (RO)

Register 6.14. EFUSE_RD_MAC_SYS5_REG (0x0058)



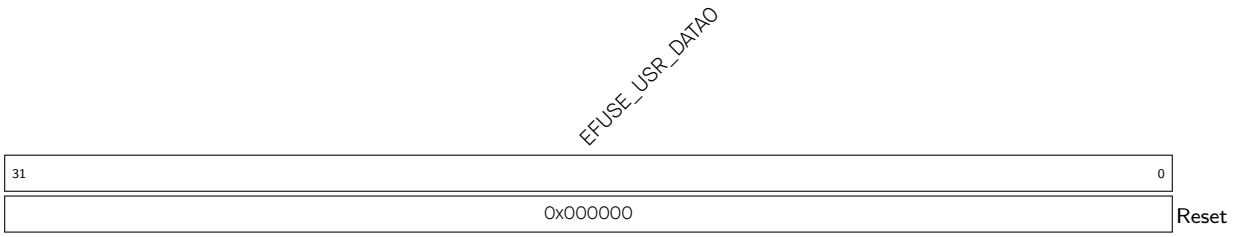

**EFUSE_SYS_DATA_PART0_2** Represents the third 32 bits of zeroth part of system data. (RO)

Register 6.15. EFUSE_RD_SYS_PART1_DATA*n*_REG (*n*: 0-7) (0x005C+0x4**n*)



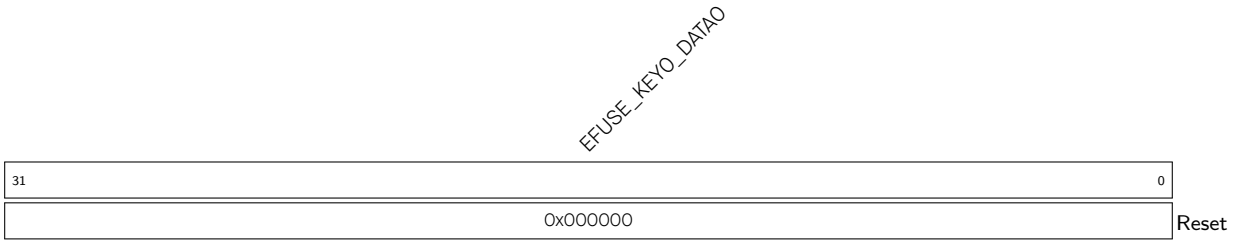

**EFUSE_SYS_DATA_PART1_*n*** Represents the *n*th 32 bits of first part of system data. (RO)

Register 6.16. EFUSE_RD_USR_DATA*n*_REG (*n*: 0-7) (0x007C+0x4*n)



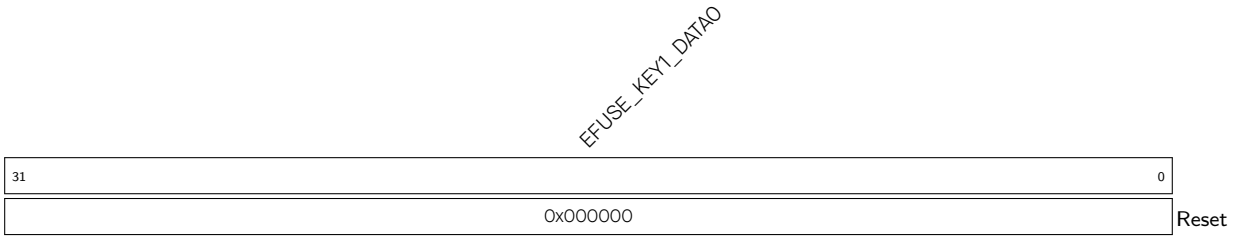

**EFUSE_USR_DATA*n*** Represents the *n*th 32 bits of Block 3 (user). (RO)

Register 6.17. EFUSE_RD_KEY0_DATA*n*_REG (*n*: 0-7) (0x009C+0x4*n)



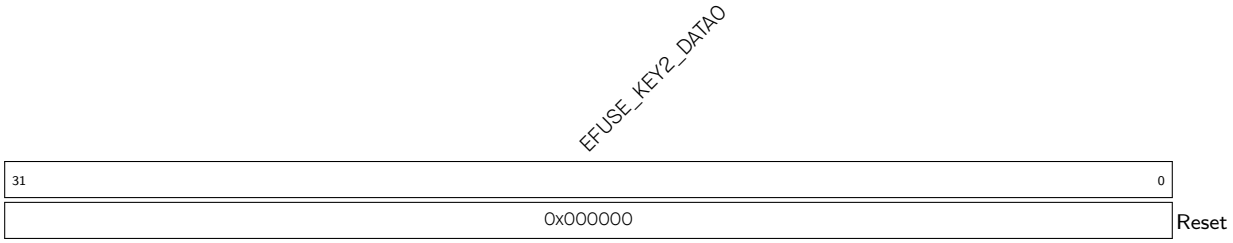

**EFUSE_KEY0_DATA*n*** Represents the *n*th 32 bits of Key0. (RO)

Register 6.18. EFUSE_RD_KEY1_DATA*n*_REG (*n*: 0-7) (0x00BC+0x4*n)



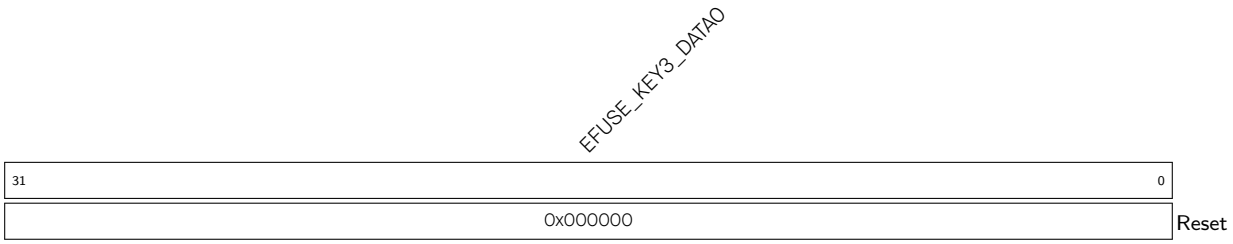

**EFUSE_KEY1_DATA*n*** Represents the *n*th 32 bits of Key1. (RO)

Register 6.19. EFUSE_RD_KEY2_DATA*n*_REG (*n*: 0-7) (0x00DC+0x4*n)



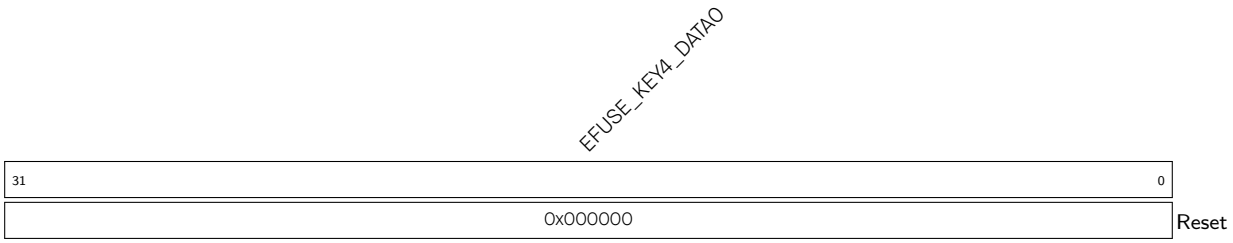

**EFUSE_KEY2_DATA*n*** Represents the *n*th 32 bits of Key2. (RO)

Register 6.20. EFUSE_RD_KEY3_DATA*n*_REG (*n*: 0-7) (0x00FC+0x4**n*)



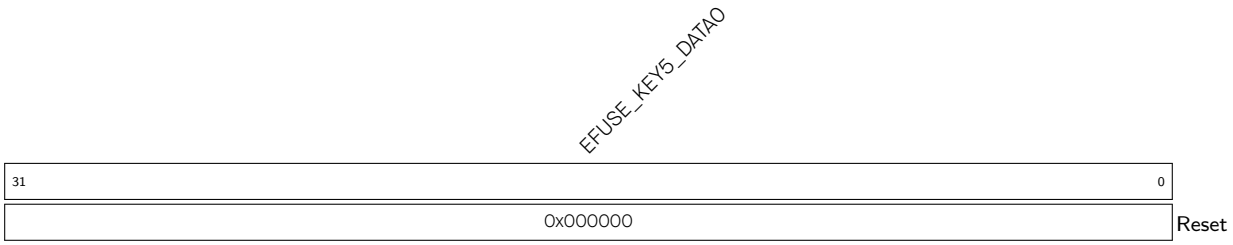

**EFUSE_KEY3_DATA*n*** Represents the *n*th 32 bits of Key3. (RO)

Register 6.21. EFUSE_RD_KEY4_DATA*n*_REG (*n*: 0-7) (0x011C+0x4**n*)



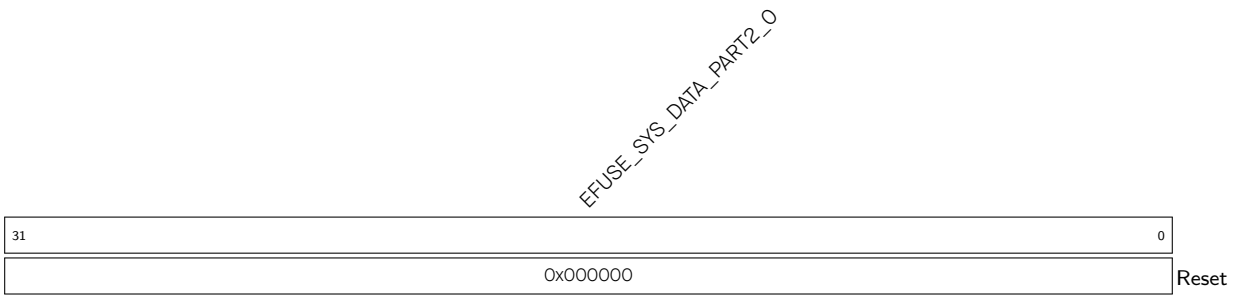

**EFUSE_KEY4_DATA*n*** Represents the *n*th 32 bits of Key4. (RO)

Register 6.22. EFUSE_RD_KEY5_DATA*n*_REG (*n*: 0-7) (0x013C+0x4**n*)



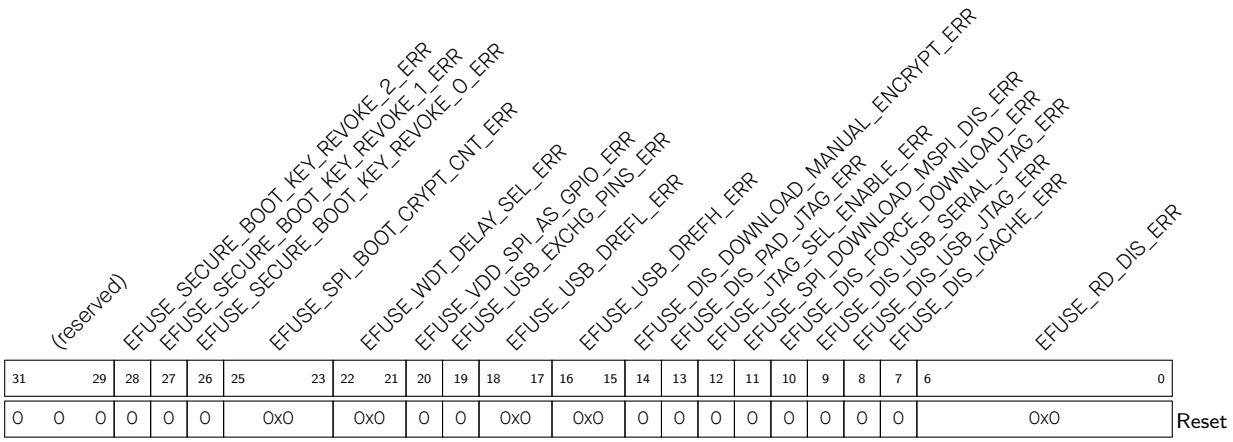

**EFUSE_KEY5_DATA*n*** Represents the *n*th 32 bits of Key5. (RO)

Register 6.23. EFUSE_RD_SYS_PART2_DATA*n*_REG (*n*: 0-7) (0x015C+0x4**n*)




**EFUSE_SYS_DATA_PART2_*n*** Represents the *n*th 32 bits of second part of system data. (RO)

Register 6.24. EFUSE_RD_REPEAT_DATA_ERR0_REG (0x017C)




**EFUSE_RD_DIS_ERR** This bit being 1 represents the programming error of EFUSE_RD_DIS. (RO)

**EFUSE_DIS_ICACHE_ERR** This bit being 1 represents the programming error of EFUSE_DIS_ICACHE. (RO)

**EFUSE_DIS_USB_JTAG_ERR** This bit being 1 represents the programming error of EFUSE_DIS_USB_JTAG. (RO)

**EFUSE_DIS_USB_SERIAL_JTAG_ERR** This bit being 1 represents the programming error of EFUSE_DIS_USB_SERIAL_JTAG. (RO)

**EFUSE_DIS_FORCE_DOWNLOAD_ERR** This bit being 1 represents the programming error of EFUSE_DIS_FORCE_DOWNLOAD. (RO)

**EFUSE_SPI_DOWNLOAD_MSPI_DIS_ERR** This bit being 1 represents the programming error of EFUSE_SPI_DOWNLOAD_MSPI_DIS. (RO)

**EFUSE_JTAG_SEL_ENABLE_ERR** This bit being 1 represents the programming error of EFUSE_JTAG_SEL_ENABLE. (RO)

**EFUSE_DIS_PAD_JTAG_ERR** This bit being 1 represents the programming error of EFUSE_DIS_PAD_JTAG. (RO)

**EFUSE_DIS_DOWNLOAD_MANUAL_ENCRYPT_ERR** This bit being 1 represents the programming error of EFUSE_DIS_DOWNLOAD_MANUAL_ENCRYPT. (RO)

**EFUSE_USB_DREFH_ERR** This bit being 1 represents the programming error of EFUSE_USB_DREFH. (RO)

**EFUSE_USB_DREFL_ERR** This bit being 1 represents the programming error of EFUSE_USB_DREFL. (RO)

**EFUSE_USB_EXCHG_PINS_ERR** This bit being 1 represents the programming error of EFUSE_USB_EXCHG_PINS. (RO)

**EFUSE_VDD_SPI_AS_GPIO_ERR** This bit being 1 represents the programming error of EFUSE_VDD_SPI_AS_GPIO. (RO)

**EFUSE_WDT_DELAY_SEL_ERR** This bit being 1 represents the programming error of EFUSE_WDT_DELAY_SEL. (RO)

Continued on the next page...

Register 6.24. EFUSE_RD_REPEAT_DATA_ERR0_REG (0x017C)

Continued from the previous page...

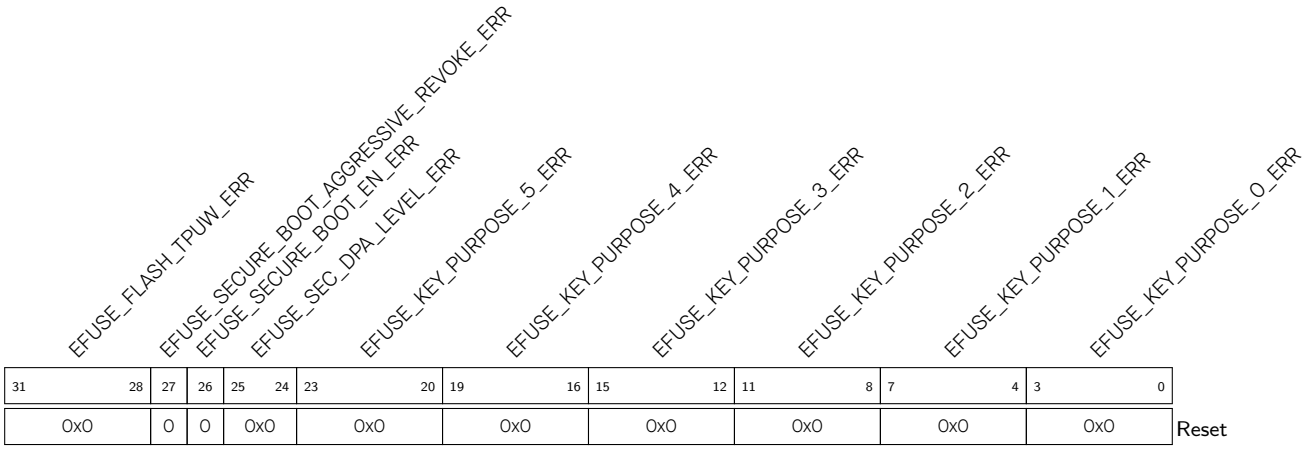
**EFUSE_SPI_BOOT_CRYPT_CNT_ERR** This bit being 1 represents the programming error of EFUSE_SPI_BOOT_CRYPT_CNT. (RO)

**EFUSE_SECURE_BOOT_KEY_REVOKE_0_ERR** This bit being 1 represents the programming error of EFUSE_SECURE_BOOT_KEY_REVOKE_0. (RO)

**EFUSE_SECURE_BOOT_KEY_REVOKE_1_ERR** This bit being 1 represents the programming error of EFUSE_SECURE_BOOT_KEY_REVOKE_1. (RO)

**EFUSE_SECURE_BOOT_KEY_REVOKE_2_ERR** This bit being 1 represents the programming error of EFUSE_SECURE_BOOT_KEY_REVOKE_2. (RO)

Register 6.25. EFUSE_RD_REPEAT_DATA_ERR1_REG (0x0180)




**EFUSE_KEY_PURPOSE_0_ERR** This bit being 1 represents the programming error of EFUSE_KEY_PURPOSE_0. (RO)

**EFUSE_KEY_PURPOSE_1_ERR** This bit being 1 represents the programming error of EFUSE_KEY_PURPOSE_1. (RO)

**EFUSE_KEY_PURPOSE_2_ERR** This bit being 1 represents the programming error of EFUSE_KEY_PURPOSE_2. (RO)

**EFUSE_KEY_PURPOSE_3_ERR** This bit being 1 represents the programming error of EFUSE_KEY_PURPOSE_3. (RO)

**EFUSE_KEY_PURPOSE_4_ERR** This bit being 1 represents the programming error of EFUSE_KEY_PURPOSE_4. (RO)

**EFUSE_KEY_PURPOSE_5_ERR** This bit being 1 represents the programming error of EFUSE_KEY_PURPOSE_5. (RO)

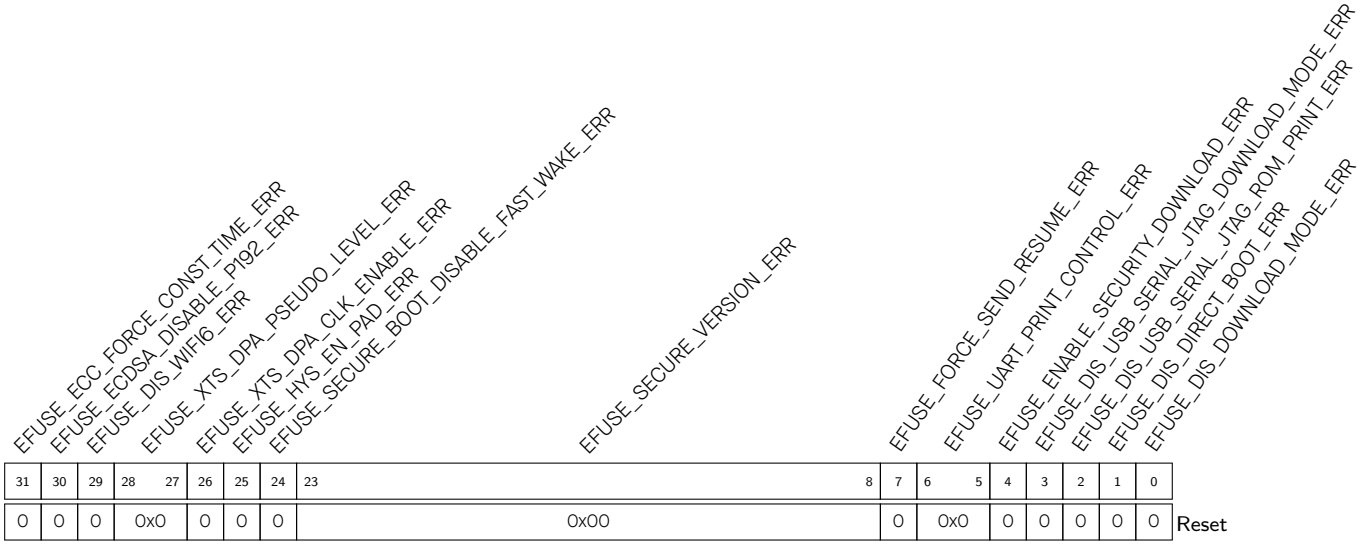
**EFUSE_SEC_DPA_LEVEL_ERR** This bit being 1 represents the programming error of EFUSE_SEC_DPA_LEVEL. (RO)

**EFUSE_SECURE_BOOT_EN_ERR** This bit being 1 represents the programming error of EFUSE_SECURE_BOOT_EN. (RO)

**EFUSE_SECURE_BOOT_AGGRESSIVE_REVOKE_ERR** This bit being 1 represents the programming error of EFUSE_SECURE_BOOT_AGGRESSIVE_REVOKE. (RO)

**EFUSE_FLASH_TPUW_ERR** This bit being 1 represents the programming error of EFUSE_FLASH_TPUW. (RO)

Register 6.26. EFUSE_RD_REPEAT_DATA_ERR2_REG (0x0184)




**EFUSE_DIS_DOWNLOAD_MODE_ERR** This bit being 1 represents the programming error of EFUSE_DIS_DOWNLOAD_MODE. (RO)

**EFUSE_DIS_DIRECT_BOOT_ERR** This bit being 1 represents the programming error of EFUSE_DIS_DIRECT_BOOT. (RO)

**EFUSE_DIS_USB_SERIAL_JTAG_ROM_PRINT_ERR** This bit being 1 represents the programming error of EFUSE_DIS_USB_SERIAL_JTAG_ROM_PRINT. (RO)

**EFUSE_DIS_USB_SERIAL_JTAG_DOWNLOAD_MODE_ERR** This bit being 1 represents the programming error of EFUSE_DIS_USB_SERIAL_JTAG_DOWNLOAD_MODE. (RO)

**EFUSE_ENABLE_SECURITY_DOWNLOAD_ERR** This bit being 1 represents the programming error of EFUSE_ENABLE_SECURITY_DOWNLOAD. (RO)

**EFUSE_UART_PRINT_CONTROL_ERR** This bit being 1 represents the programming error of EFUSE_UART_PRINT_CONTROL. (RO)

**EFUSE_FORCE_SEND_RESUME_ERR** This bit being 1 represents the programming error of EFUSE_FORCE_SEND_RESUME. (RO)

**EFUSE_SECURE_VERSION_ERR** This bit being 1 represents the programming error of EFUSE_SECURE_VERSION. (RO)

**EFUSE_SECURE_BOOT_DISABLE_FAST_WAKE_ERR** This bit being 1 represents the programming error of EFUSE_SECURE_BOOT_DISABLE_FAST_WAKE. (RO)

**EFUSE_HYS_EN_PAD_ERR** This bit being 1 represents the programming error of EFUSE_HYS_EN_PAD. (RO)

**EFUSE_XTS_DPA_CLK_ENABLE_ERR** This bit being 1 represents the programming error of EFUSE_XTS_DPA_CLK_ENABLE. (RO)

**EFUSE_XTS_DPA_PSEUDO_LEVEL_ERR** This bit being 1 represents the programming error of EFUSE_XTS_DPA_PSEUDO_LEVEL. (RO)

Continued on the next page...

Register 6.26. EFUSE_RD_REPEAT_DATA_ERR2_REG (0x0184)

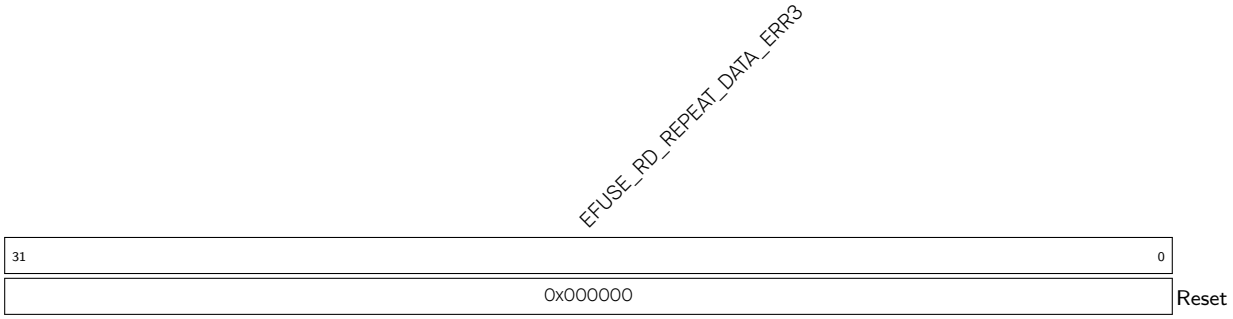
Continued from the previous page...

**EFUSE_DIS_WIFI6_ERR** This bit being 1 represents the programming error of EFUSE_DIS_WIFI6. (RO)

**EFUSE_ECDSA_DISABLE_P192_ERR** This bit being 1 represents the programming error of EFUSE_ECDSA_DISABLE_P192. (RO)

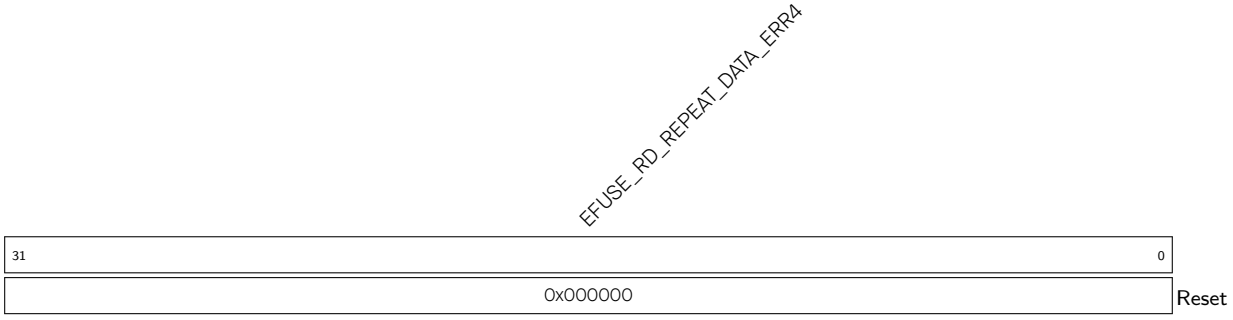
**EFUSE_ECC_FORCE_CONST_TIME_ERR** This bit being 1 represents the programming error of EFUSE_ECC_FORCE_CONST_TIME. (RO)

Register 6.27. EFUSE_RD_REPEAT_DATA_ERR3_REG (0x0188)



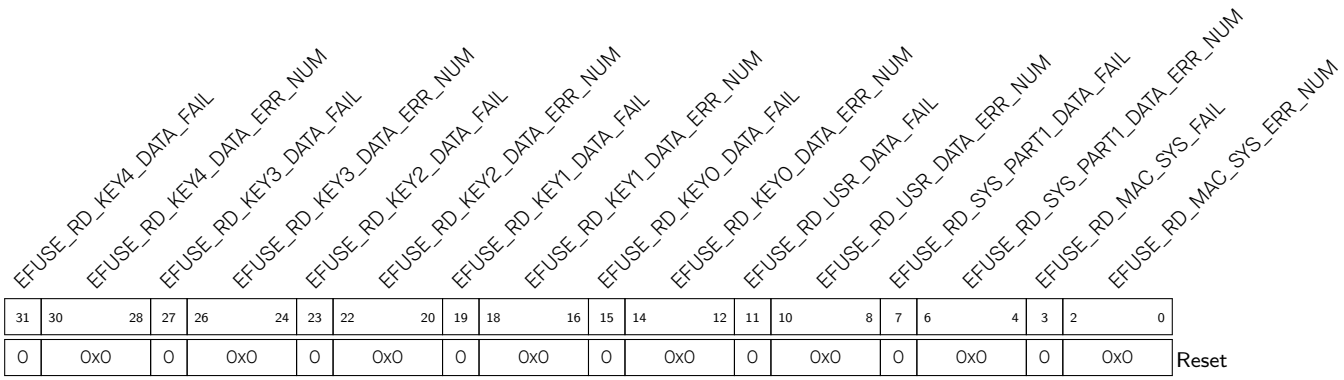

**EFUSE_RD_REPEAT_DATA_ERR3** Reserved (RO)

Register 6.28. EFUSE_RD_REPEAT_DATA_ERR4_REG (0x018C)




**EFUSE_RD_REPEAT_DATA_ERR4** Reserved (RO)

Register 6.29. EFUSE_RD_RS_DATA_ERR0_REG (0x0190)




**EFUSE_RD_MAC_SYS_ERR_NUM** Represents the number of error bytes in RD_MAC_SYS. (RO)

**EFUSE_RD_MAC_SYS_FAIL** Represents error status of register.

0: No failure and the data of RD_MAC_SYS is reliable

1: Programming RD_MAC_SYS failed and the number of error bytes is over 6.

(RO)

**EFUSE_RD_SYS_PART1_DATA_ERR_NUM** Represents the number of error bytes in RD_SYS_PART1_DATA. (RO)

**EFUSE_RD_SYS_PART1_DATA_FAIL** Represents error status of register.

0: No failure and the data of RD_SYS_PART1_DATA is reliable

1: Programming RD_SYS_PART1_DATA failed and the number of error bytes is over 6.

(RO)

**EFUSE_RD_USR_DATA_ERR_NUM** Represents the number of error bytes in RD_USR_DATA. (RO)

**EFUSE_RD_USR_DATA_FAIL** Represents error status of register.

0: No failure and that the data of RD_USR_DATA is reliable.

1: Programming RD_USR_DATA failed and the number of error bytes is over 6.

(RO)

**EFUSE_RD_KEY0_DATA_ERR_NUM** Represents the number of error bytes in RD_KEY0_DATA. (RO)

**EFUSE_RD_KEY0_DATA_FAIL** Represents error status of register.

0: No failure and the data of RD_KEY0_DATA is reliable.

1: Programming RD_KEY0_DATA failed and the number of error bytes is over 6.

(RO)

**EFUSE_RD_KEY1_DATA_ERR_NUM** Represents the number of error bytes in RD_KEY1_DATA. (RO)

**EFUSE_RD_KEY1_DATA_FAIL** Represents error status of register.

0: No failure and the data of RD_KEY1_DATA is reliable.

1: Programming RD_KEY1_DATA failed and the number of error bytes is over 6.

(RO)

**EFUSE_RD_KEY2_DATA_ERR_NUM** Represents the number of error bytes in RD_KEY2_DATA. (RO)

**Continued on the next page...**

Register 6.29. EFUSE_RD_RS_DATA_ERR0_REG (0x0190)

Continued from the previous page...

**EFUSE_RD_KEY2_DATA_FAIL** Represents error status of register.

0: No failure and the data of RD_KEY2_DATA is reliable.

1: Programming RD_KEY2_DATA failed and the number of error bytes is over 6.

(RO)

**EFUSE_RD_KEY3_DATA_ERR_NUM** Represents the number of error bytes in RD_KEY3_DATA. (RO)

**EFUSE_RD_KEY3_DATA_FAIL** Represents error status of register.

0: No failure and the data of RD_KEY3_DATA is reliable.

1: Programming RD_KEY3_DATA failed and the number of error bytes is over 6.

(RO)

**EFUSE_RD_KEY4_DATA_ERR_NUM** Represents the number of error bytes in RD_KEY4_DATA. (RO)

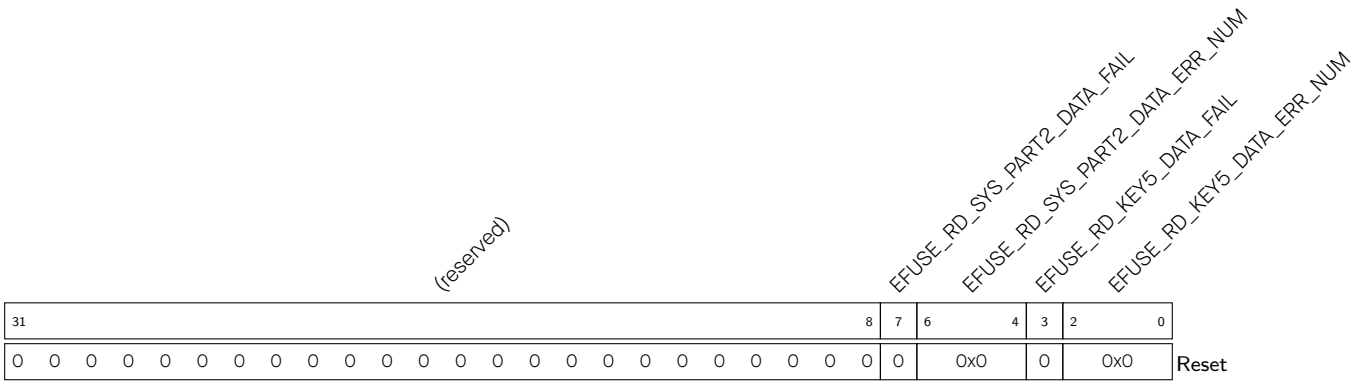
**EFUSE_RD_KEY4_DATA_FAIL** Represents error status of register.

0: No failure and the data of RD_KEY4_DATA is reliable.

1: Programming RD_KEY4_DATA failed and the number of error bytes is over 6.

(RO)

Register 6.30. EFUSE_RD_RS_DATA_ERR1_REG (0x0194)




**EFUSE_RD_KEY5_DATA_ERR_NUM** Represents the number of error bytes in RD_KEY5_DATA. (RO)

**EFUSE_RD_KEY5_DATA_FAIL** Represents error status of register.

0: No failure and that the data of RD_KEY5_DATA is reliable.

1: Programming RD_KEY5_DATA failed and the number of error bytes is over 6.

(RO)

**EFUSE_RD_SYS_PART2_DATA_ERR_NUM** Represents the number of error bytes in RD_SYS_PART2_DATA. (RO)

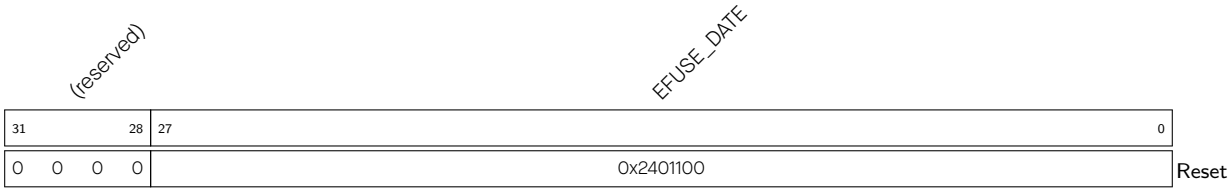
**EFUSE_RD_SYS_PART2_DATA_FAIL** Represents error status of register.

0: No failure and that the data of RD_SYS_PART2_DATA is reliable.

1: Programming RD_SYS_PART2_DATA failed and the number of error bytes is over 6.

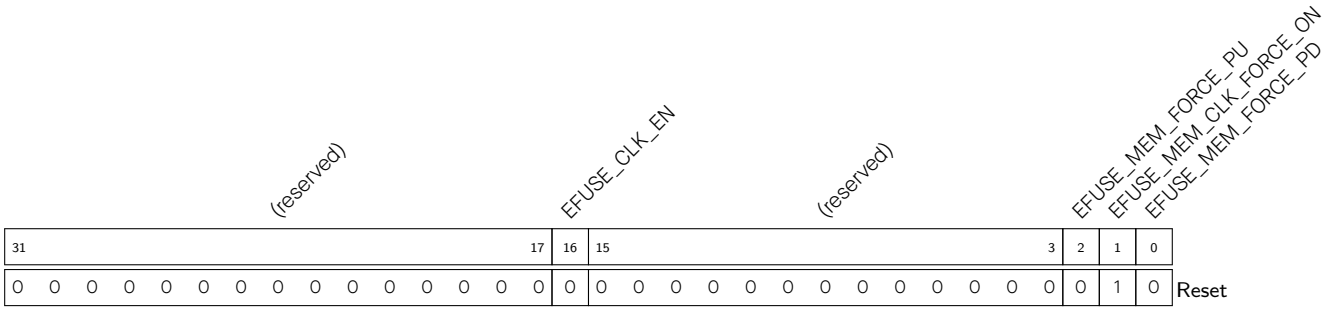
(RO)

Register 6.31. EFUSE_DATE_REG (0x0198)




**EFUSE_DATE** Stores eFuse version. (R/W)

Register 6.32. EFUSE_CLK_REG (0x01C8)




**EFUSE_MEM_FORCE_PD** Configures whether to force eFuse SRAM into power-saving mode.

1: Force

0: No effect

(R/W)

**EFUSE_MEM_CLK_FORCE_ON** Configures whether to force activate clock signal of eFuse SRAM.

1: Force activate

0: No effect

(R/W)

**EFUSE_MEM_FORCE_PU** Configures whether to force eFuse SRAM into working mode.

1: Force

0: No effect

(R/W)

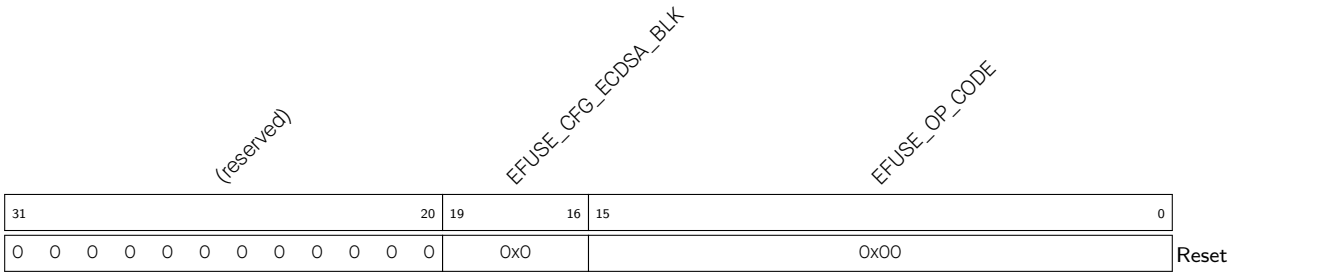
**EFUSE_CLK_EN** Configures whether to force enable eFuse register configuration clock signal.

1: Force

0: The clock is enabled only during the reading and writing of registers

(R/W)

Register 6.33. EFUSE_CONF_REG (0x01CC)




**EFUSE_OP_CODE** Configures operation command type.

0x5A5A: Programming operation command

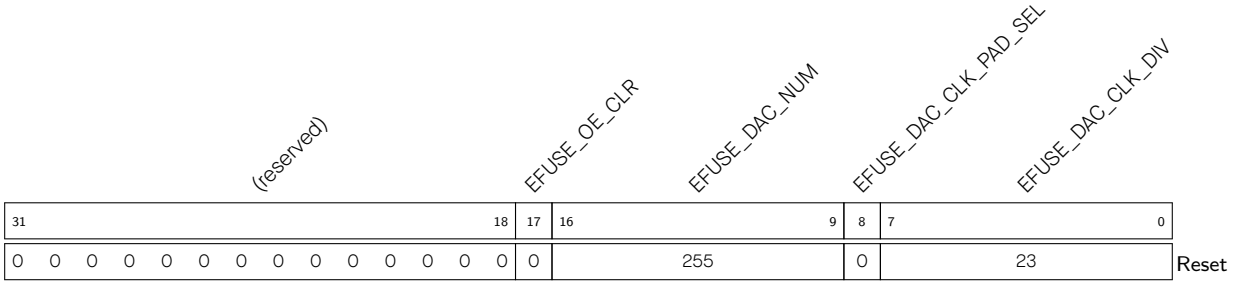
0x5AA5: Read operation command

Other values: No effect

(R/W)

**EFUSE_CFG_ECDSA_BLK** Configures which block to use for ECDSA_DS key output. (R/W)

Register 6.34. EFUSE_DAC_CONF_REG (0x01E8)



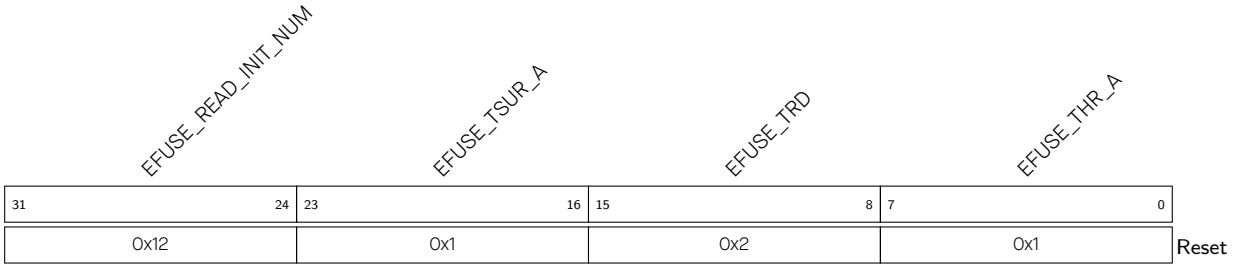

**EFUSE_DAC_CLK_DIV** Controls the division factor of the rising clock of the programming voltage. (R/W)

**EFUSE_DAC_CLK_PAD_SEL** Reserved (R/W)

**EFUSE_DAC_NUM** Controls the rising period of the programming voltage.
Measurement unit: One cycle of the eFuse core clock. (R/W)

**EFUSE_OE_CLR** Reduces the power supply of the programming voltage. (R/W)

Register 6.35. EFUSE_RD_TIM_CONF_REG (0x01EC)



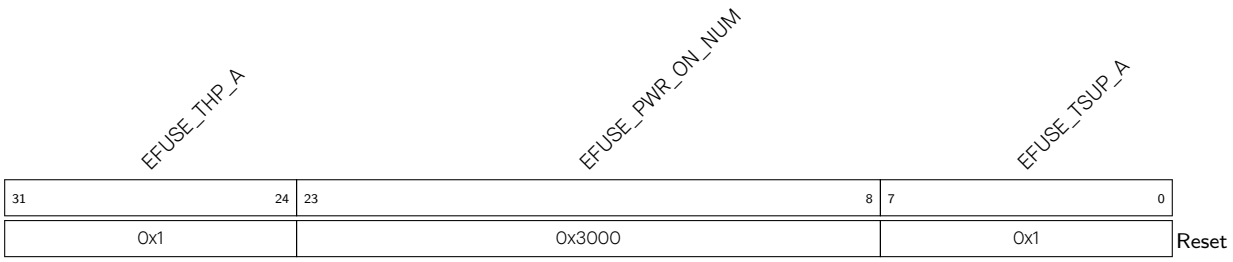

**EFUSE_THR_A** Configures the read hold time.
Measurement unit: One cycle of the eFuse core clock.(R/W)

**EFUSE_TRD** Configures the read time.
Measurement unit: One cycle of the eFuse core clock.(R/W)

**EFUSE_TSUR_A** Configures the read setup time.
Measurement unit: One cycle of the eFuse core clock.(R/W)

**EFUSE_READ_INIT_NUM** Configures the waiting time of reading eFuse memory.
Measurement unit: One cycle of the eFuse core clock. (R/W)

Register 6.36. EFUSE_WR_TIM_CONF1_REG (0x01F0)




**EFUSE_TSUP_A** Configures the programming setup time.

Measurement unit: One cycle of the eFuse core clock.(R/W)

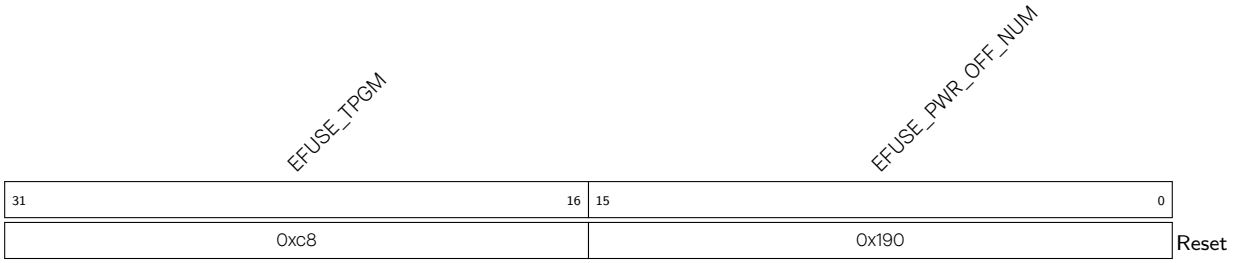
**EFUSE_PWR_ON_NUM** Configures the power up time for VDDQ.

Measurement unit: One cycle of the eFuse core clock.(R/W)

**EFUSE_THP_A** Configures the programming hold time.

Measurement unit: One cycle of the eFuse core clock.(R/W)

Register 6.37. EFUSE_WR_TIM_CONF2_REG (0x01F4)



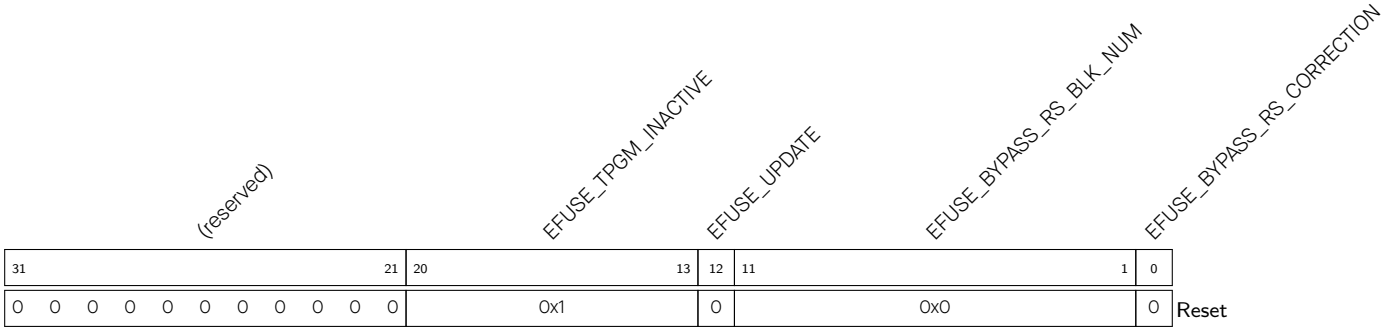

**EFUSE_PWR_OFF_NUM** Configures the power outage time for VDDQ.

Measurement unit: One cycle of the eFuse core clock.(R/W)

**EFUSE_TPGM** Configures the active programming time.

Measurement unit: One cycle of the eFuse core clock.(R/W)

Register 6.38. EFUSE_WR_TIM_CONF0_RS_BYPASS_REG (0x01F8)




**EFUSE_BYPASS_RS_CORRECTION** Configures whether to bypass Reed-Solomon correction step.

0: Not bypass

1: Bypass

(R/W)

**EFUSE_BYPASS_RS_BLK_NUM** Configures block number for the double programming operation.

(R/W)

**EFUSE_UPDATE** Configures whether to update multi-bit register signals.

1: Update

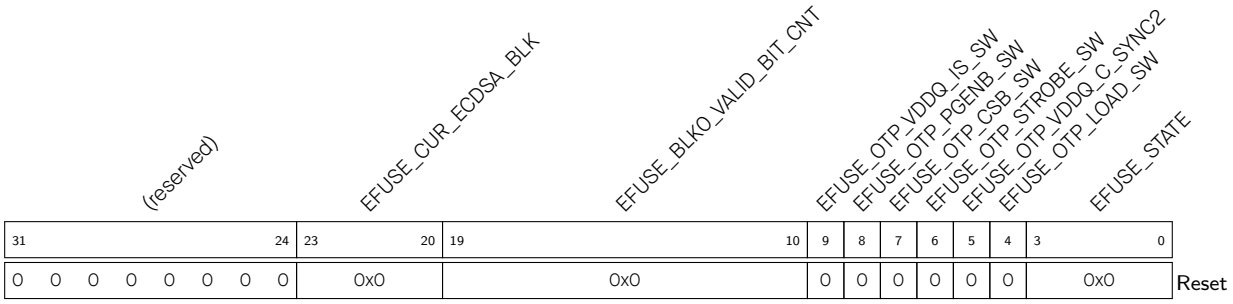
0: No effect

(WT)

**EFUSE_TPGM_INACTIVE** Configures the inactive programming time.

Measurement unit: One cycle of the eFuse core clock. (R/W)

Register 6.39. EFUSE_STATUS_REG (0x01D0)




**EFUSE_STATE** Represents the state of the eFuse state machine. (RO)

**EFUSE_OTP_LOAD_SW** Represents the value of OTP_LOAD_SW. (RO)

**EFUSE_OTP_VDDQ_C_SYNC2** Represents the value of OTP_VDDQ_C_SYNC2. (RO)

**EFUSE_OTP_STROBE_SW** Represents the value of OTP_STROBE_SW. (RO)

**EFUSE_OTP_CSB_SW** Represents the value of OTP_CSB_SW. (RO)

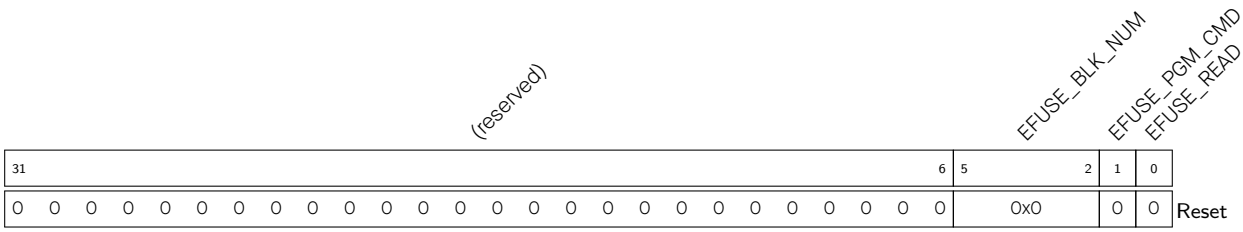
**EFUSE_OTP_PGENB_SW** Represents the value of OTP_PGENB_SW. (RO)

**EFUSE_OTP_VDDQ_IS_SW** Represents the value of OTP_VDDQ_IS_SW. (RO)

**EFUSE_BLK0_VALID_BIT_CNT** Represents the number of valid block bits. (RO)

**EFUSE_CUR_ECDSA_BLK** Represents the block used for ECDSA_DS key output. (RO)

Register 6.40. EFUSE_CMD_REG (0x01D4)




**EFUSE_READ_CMD** Configures whether to send read command.

1: Send

0: No effect

(R/W/SC)

**EFUSE_PGM_CMD** Configures whether to send programming command.

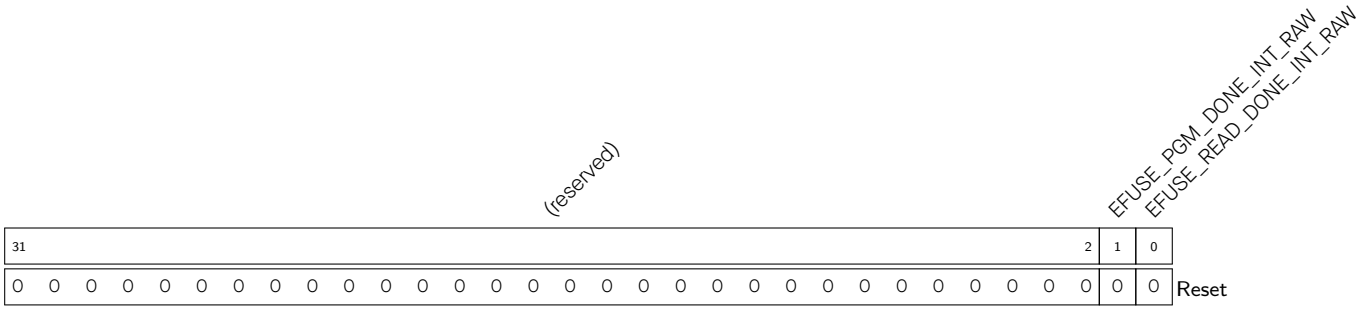
1: Send

0: No effect

(R/W/SC)

**EFUSE_BLK_NUM** Represents the serial number of the block to be programmed. Value 0-10 corresponds to block number 0-10, respectively. (R/W)

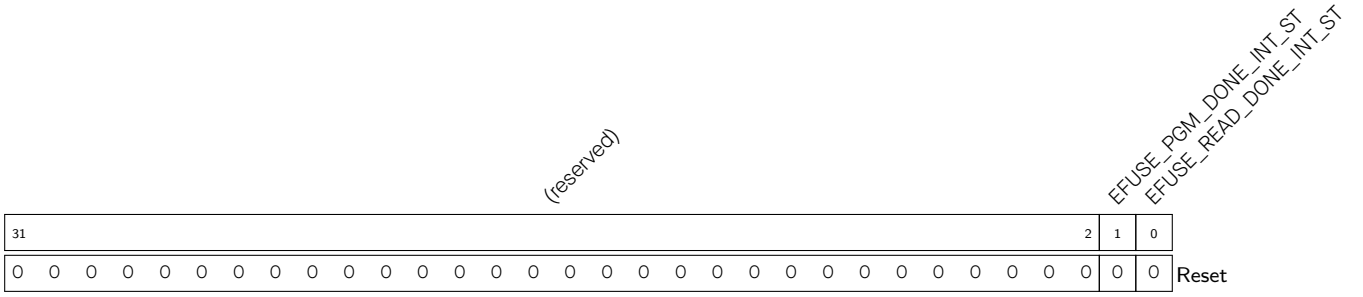
Register 6.41. EFUSE_INT_RAW_REG (0x01D8)




**EFUSE_READ_DONE_INT_RAW** Represents the raw interrupt status of read_done. (R/SS/WTC)

**EFUSE_PGM_DONE_INT_RAW** Represents the raw interrupt status of pgm_done. (R/SS/WTC)

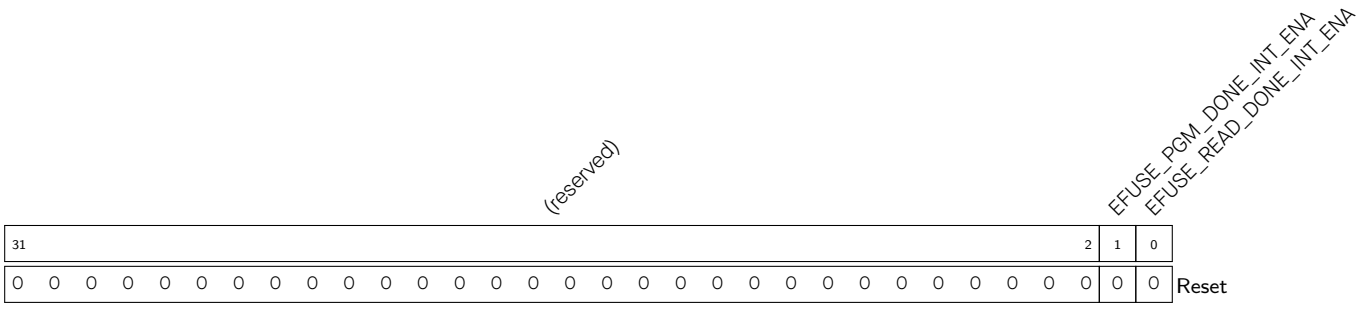
Register 6.42. EFUSE_INT_ST_REG (0x01DC)




**EFUSE_READ_DONE_INT_ST** Represents the masked interrupt status of read_done. (RO)

**EFUSE_PGM_DONE_INT_ST** Represents the masked interrupt status of pgm_done. (RO)

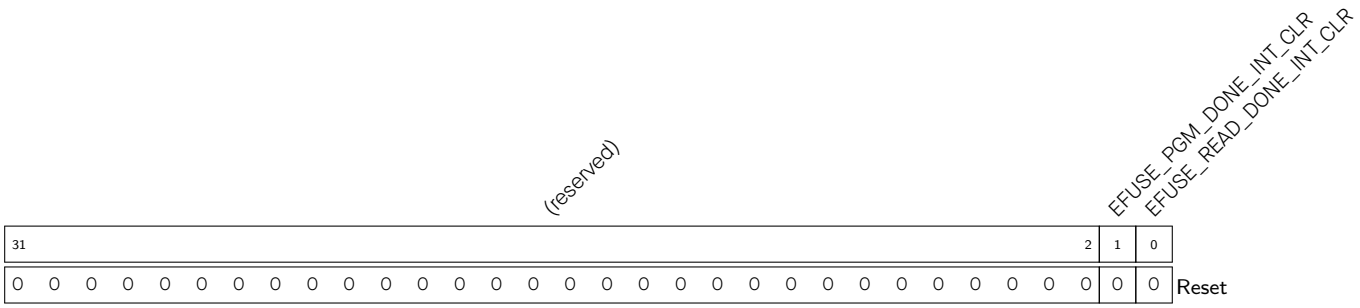
Register 6.43. EFUSE_INT_ENA_REG (0x01E0)




**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 6.44. 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)

# Part III

# 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 7

# GPIO Matrix and IO MUX

## 7.1 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-C61 provides two groups of GPIO matrix and IO MUX:

- HP GPIO matrix and HP IO MUX
- LP GPIO matrix and LP IO MUX

The ESP32-C61 chip features 30 GPIO pins, numbered from GPIO0 to GPIO29, including

- 7 LP GPIO pins (GPIO0~GPIO6) for either HP or LP peripherals.
- 15 HP GPIO pins (GPIO7~GPIO13, GPIO22~GPIO29) for HP peripherals only.
- 8 dedicated GPIO pins (GPIO14~GPIO21) for external flash or PSRAM. **These pins can not be used for other purpose, and therefore are not described in subsequent sections**. For more information regarding these pins, please refer to *ESP32-C61 Datasheet* > Section *Pin Mapping Between Chip and Flash/PSRAM*.

Each pin can be used as a general-purpose I/O, or be connected to an internal peripheral signal. Together these modules provide highly configurable I/O.

- 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 the LP IO MUX, the LP system can be configured to simple GPIO input/output and GPIO wake-up functions.

**Unless otherwise specified, GPIO matrix refers to both LP GPIO matrix and HP GPIO matrix, so as to IO MUX.**

## 7.2 Features

### 7.2.1 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
- 36 HP peripheral input signals sourced from the input of any GPIO pins

- 48 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
- GPIO simple input and output
- HP GPIO Wakeup

**HP IO MUX has the following features:**

- Control of 22 GPIOs (GPIO0~GPIO13, GPIO22~GPIO29) 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.
- Enhanced high-frequency digital performance achieved by routing some digital signals (such as SPI) directly from HP IO MUX to peripherals.

### 7.2.2 LP GPIO Matrix and LP IO MUX

**LP GPIO matrix has the following features:**

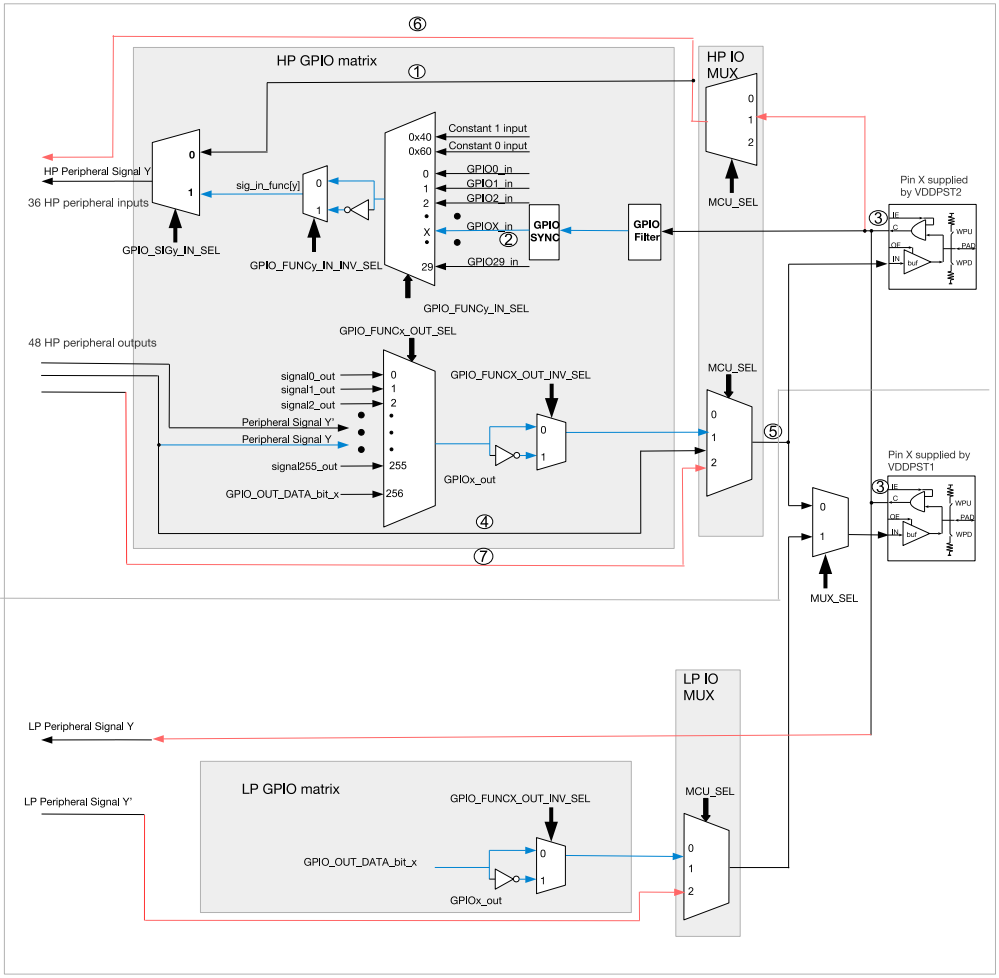
- GPIO simple input and output
- LP GPIO Wakeup

**LP IO MUX has the following feature:**

- Control of 7 LP GPIO pins (GPIO0~GPIO6) for LP peripherals.
- A configuration register LP_IO_MUX_GPIO*n*_REG provided for each LP GPIO pin, to control the pin's input/output, pull-up/pull-down, drive strength, and function selection.
- A field LP_AON_GPIO_MUX_SEL provided to select the source of the pin control signals, from HP IO MUX or LP IO MUX.

## 7.3 Architectural Overview

Figure 7.3-1 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.




—>: Signals bypassing the GPIO matrix. —>: Signals passing through the GPIO matrix.

**Figure 7.3-1. 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 Peripheral input signals marked "yes" in column "Direct input through IO MUX" in Table 7.12-1 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 22 inputs from GPIO SYNC to HP GPIO matrix, since ESP32-C61 provides 22 GPIO pins available for users in total.

③ The pins are controlled by the signals: IE, OE, WPU, and WPD.

④ Peripheral outputs marked "yes" in column "Direct output through IO MUX" in Table 7.12-1 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 22 outputs (corresponding to GPIO pin *X*: 0~14, 23~28) 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 7.3-2 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 GPIO pins and can be controlled using IE, OE, WPU, and WPD

signals. For the configuration of these signals, see IO_MUX_GPIOx_REG or LP_IO_MUX_GPIOx_REG.

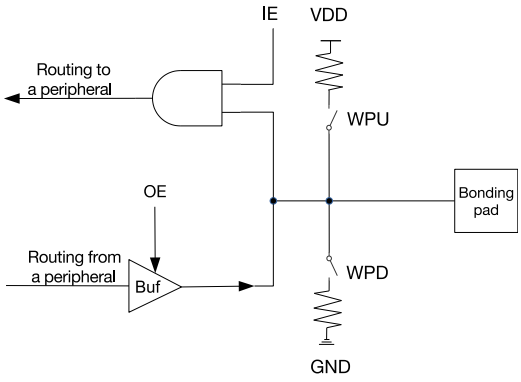



Figure 7.3-2. Internal Structure of a Pad

- IE: input enable
- OE: output enable
- WPU: internal weak pull-up resistor enable
- WPD: internal weak pull-down resistor enable
- 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

## 7.4 Peripheral Input via GPIO Matrix

### 7.4.1 Overview

To receive a peripheral input signal via HP GPIO matrix,

- Configure the matrix to source the peripheral input signal from one of the 22 GPIOs (0~13, 22~29), see Table 7.12-1.
- 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 7.3-1 and Section 7.4.6.

As shown in Figure 7.3-1, 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 7.4.6.

All signals filtered by the GPIO Filter hardware are synchronized by the GPIO SYNC hardware to IO MUX operating clock and then enter the GPIO matrix, see Section 7.4.2. Such signal filtering and synchronization features apply to all GPIO matrix signals but do not apply when using the IO MUX.

### 7.4.2 Signal Synchronization

Only HP GPIO matrix supports this signal synchronization function. Figure 7.4-1 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.

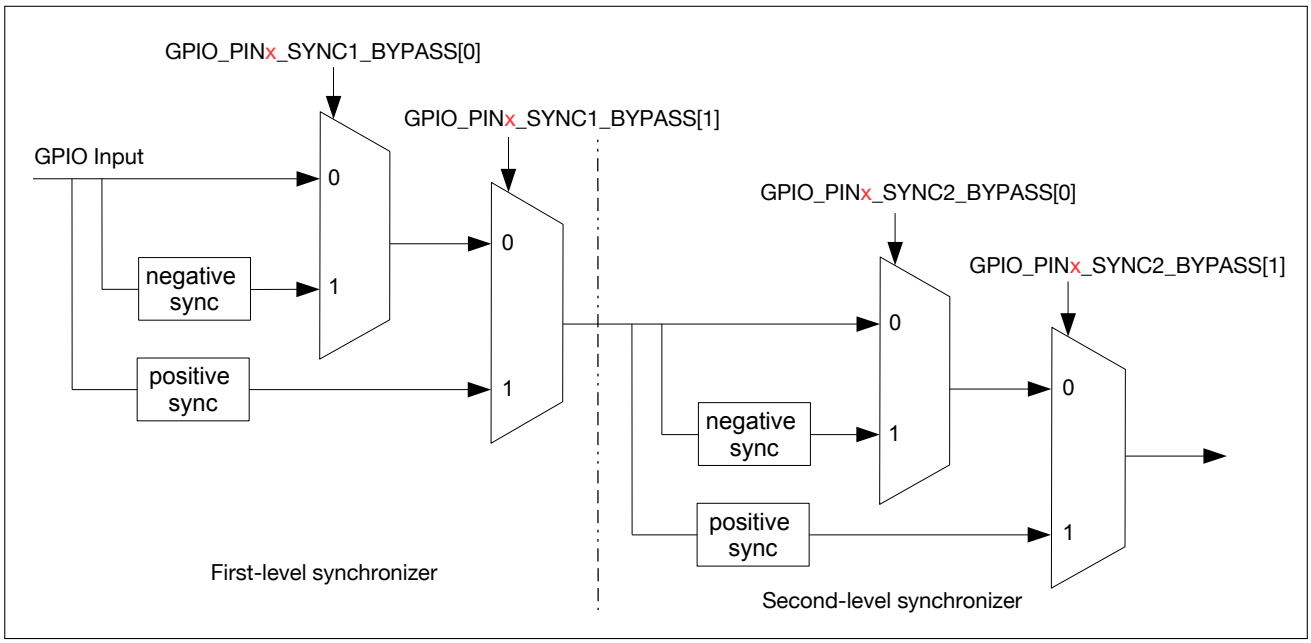



Figure 7.4-1. GPIO Input Synchronized on Rising Edge or on Falling Edge of HP IO MUX Operating Clock

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 7.4-1) to lower the probability of causing metastability.

### 7.4.3 GPIO Filter

Only HP GPIO matrix supports this 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 7.4-2.

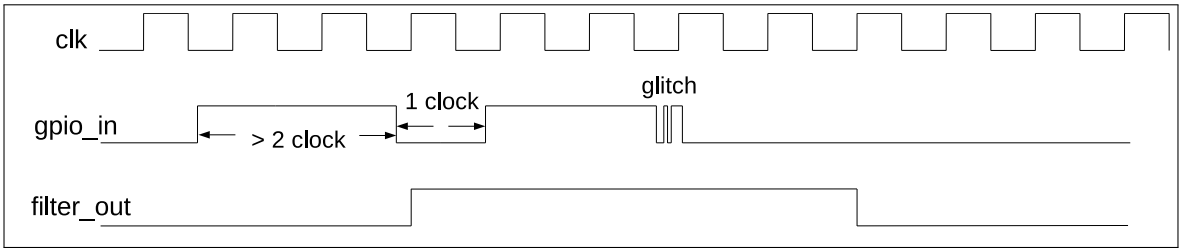



Figure 7.4-2. GPIO Filter Timing of GPIO Input Signals

### 7.4.4 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_GPIO*x*_REG, to enable pin input.
- Read the GPIO input from LP_GPIO_IN_REG[*x*].

## 7.4.5 GPIO Wakeup

### 7.4.5.1 HP GPIO Wakeup

The HP GPIO wakeup feature is designed to awaken the system from Light-sleep. HP GPIO wakeup is not available when the HP system is powered down.

GPIO0~GPIO13 and GPIO22~GPIO29 can be used to generate HP GPIO wakeup by enabling GPIO_PIN*x*_WAKEUP_ENABLE (*x*: 0~13, 22~29).

HP GPIO wakeup supports both high-voltage and low-voltage triggers, depending on the configuration of GPIO_PIN*x*_INT_TYPE.

**Table 7.4-1. HP GPIO Wakeup Signal Trigger and Clear Conditions**

| GPIO_PIN*x*_INT_TYPE[1,2] | Wakeup is generated when | Wakeup is cleared when |
|---|---|---|
| 5 | GPIO input is high-level | GPIO input is low-level |
| 4 | GPIO input is low-level | GPIO input is high-level |

[1] *x* ranges from 0 to 13, 22 to 29.
[2] If the field is configured to any other value, HP GPIO wakeup feature is disabled.

### 7.4.5.2 LP GPIO Wakeup

The LP GPIO wakeup feature is designed to awaken the system from Deep-sleep. LP GPIO wakeup is not available when the LP peripherals are powered down.

GPIO0~GPIO6 can be used to generate LP GPIO wakeup by enabling LP_GPIO_PIN*x*_WAKEUP_ENABLE (*x*: 0~6).

LP GPIO wakeup supports the following types, depending on the configuration of LP_GPIO_PIN*x*_INT_TYPE:

- posedge sensitive
- negedge sensitive
- both posedge and negedge sensitive
- high-voltage sensitive and low-voltage sensitive

**Table 7.4-2. 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 | GPIO input toggles from low-level to high-level | LP_GPIO_PIN*x*_EDGE_WAKEUP_CLR is set |

| | | |
|---|---|---|
| 2 | GPIO input toggles from high-level to low-level | LP_GPIO_PINx_EDGE_WAKEUP_CLR is set |
| 3 | GPIO input toggles from high-level to low-level, or vice versa | LP_GPIO_PINx_EDGE_WAKEUP_CLR is set |
| 4 | GPIO input is low-level | GPIO input is high-level |
| 5 | GPIO input is high-level | GPIO input is low-level |

[1] *x* ranges from 0 to 6.
[2] If the field is configured to any other value, LP GPIO wakeup feature is disabled.

## 7.4.6 Programming Procedure

### 7.4.6.1 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 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.
2. Optionally enable the GPIO Filter for pin input signals by setting IO_MUX_GPIO*x*_FILTER_EN.
3. Synchronize GPIO input signals. To do so, 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 7.4-1.
   - 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 7.4-1.
4. Configure HP IO MUX register to enable pin input[2]. For this end, set IO_MUX_GPIO*x*_REG corresponding to GPIO pin *X* as follows:
   - Set IO_MUX_GPIO*x*_FUN_IE to enable input[3].
   - 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.

**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 an 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 0x40, input signal is always 0.
   - When GPIO_FUNC*y*_IN_SEL is set to 0x60, input signal is always 1.

**Programming Example**

To connect UART0 RXD input signal (U0RXD_in, signal index 6) to GPIO7, please follow the steps below.

1. Set GPIO_SIG6_IN_SEL in GPIO_FUNC6_IN_SEL_CFG_REG to enable peripheral signal input via HP GPIO matrix.
2. Set GPIO_FUNC6_IN_SEL in GPIO_FUNC6_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.

#### 7.4.6.2 LP GPIO Matrix

For inputs, LP GPIO matrix only supports the GPIO wakeup function. For LP GPIO input wakeup programming, please refer to Section 7.4.5.2.

## 7.5 Peripheral Output via GPIO Matrix

### 7.5.1 Overview

To output a signal from a peripheral via HP 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 7.12-1) to one of the 22 GPIOs (0∼13, 22∼29).
- Then the output signal is routed from the peripheral into HP GPIO matrix and then into HP IO MUX. HP 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.

For the detailed programming procedure, see Section 7.5.3.1.

**Note:**
There is a range of peripheral output signals (97∼100 in Table 7.12-1) which are not connected to any peripheral, but to the input signals (97∼100) directly.

### 7.5.2 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 to the desired GPIO output value.

**Note:**

- The bits GPIO_OUT_REG[0∼13, 22∼29] correspond to GPIO0∼GPIO13, GPIO22∼GPIO29.
- Recommended operation: use GPIO_OUT_W1TS and GPIO_OUT_W1TC to set or clear GPIO_OUT_REG.

LP GPIO matrix supports only the Simple GPIO Output function. There is no need to configure function selection registers. Just 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.

## 7.5.3 Programming Procedure

### 7.5.3.1 HP GPIO Matrix

The output signals with a name assigned in the column "Output signal" in Table 7.12-1 can be set to go through the HP GPIO matrix into HP IO MUX and then to a pin. Figure 7.3-1 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 bit GPIO_FUNC*x*_OE_SEL and the corresponding bit in GPIO_ENABLE_W1TS_REG. To have the output enable signal decided by internal logic (for example, the FSPIQ_oe in column "Output enable signal when GPIO_FUNC*n*_OE_SEL = 0" in Table 7.12-1), 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 field GPIO_PIN*x*_PAD_DRIVER.
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 or clear the IO_MUX_GPIO*x*_FUN_WPU and IO_MUX_GPIO*x*_FUN_WPD bits to enable or disable the internal pull-up and pull-down resistors.
4. Enable hysteresis function:
   - See Section 7.10.

**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.

#### 7.5.3.2 LP GPIO Matrix

For outputs, LP GPIO matrix only supports the Simple GPIO Output function. For the detailed programming, please refer to Section 7.5.2.

## 7.6 Direct Input and Output via IO MUX

### 7.6.1 Overview

To achieve better high-frequency performance, digital signals such as SPI and JTAG can bypass the GPIO matrix and be routed directly to peripherals through the IO MUX. While this method is less flexible than using the GPIO matrix as each GPIO pin's IO MUX register offers only a limited set of functions, it significantly improves signal integrity for high-speed interfaces.

ESP32-C61 provides 7 GPIO pins (GPIO0~GPIO6) 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, these pins remain operational and can be controlled by LP system peripherals even during Deep-sleep mode. They can also be used to wake the chip from Deep-sleep.

### 7.6.2 Functional Description

The pins with LP functions (GPIO0~GPIO6) are controlled by bit[*n*] of LP_AON_GPIO_MUX_SEL in LP_AON_GPIO_MUX_REG (*n* = 0~6). By default, all these bits are set to 0, routing all input/output signals via HP IO MUX.

If bit[*n*] is set to 1 in LP_AON_GPIO_MUX_SEL, then input/output signals are controlled by LP IO MUX. In this mode, LP_IO_MUX_GPIO*n*_REG is used to control the LP GPIO pins. See 7.14-1 for the LP functions of each LP GPIO pin. Note that LP_IO_MUX_GPIO*n*_REG applies the LP GPIO pin numbering. In ESP32-C61, the numbering of LP GPIO pins is the same as that of the HP GPIO pins.

#### 7.6.2.1 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 7.13-1.
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.

**Note:**
Not all signals can be directly connected to peripheral via HP IO MUX. Some input/output signals can only be connected to peripheral via HP GPIO matrix. See Table 7.12-1.

#### 7.6.2.2 LP IO MUX

To bypass LP GPIO matrix for LP peripheral input/output signals, LP_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 7.14-1.

## 7.7 Analog Functions

### 7.7.1 Overview

ESP32-C61 provides 9 GPIO pins (GPIO0~GPIO1, GPIO3~GPIO5, GPIO8~GPIO9, GPIO12~GPIO13) with analog functions, including 5 LP GPIO pins and 4 HP GPIO pins. See Table 7.15-1.

### 7.7.2 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, to clear output enable.

To use these functions listed in Table 7.15-1, please refer to the following related chapters:

- For XTAL_32K related functions, see Chapter 12 *Low-Power Management*.
- For ADC related functions, see Chapter 34 *ADC Controller*.
- For voltage comparator related functions, see Chapter 35 *Analog Voltage Comparator*.

## 7.8 Pin Functions in Light-sleep

Pins may provide different functions when ESP32-C61 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 7.8-1. Bit Used to Control IO MUX Functions in Light-sleep Mode**

| IO MUX Function | Normal Execution<br>OR IO_MUX_GPIO*n*_SLP_SEL = 0 | Light-sleep Mode<br>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 |

* 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 7.5.3.1 for how to enable output in normal execution.

## 7.9 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.

The Hold state of each GPIO pin is controlled by the result of OR operation of the pin's Hold enable signal and the global Hold enable signal.

- Hold signal of each individual pin:
  - LP_AON_GPIO_HOLDO_REG[*n*] (*n* = 0∼13, 22∼29) controls the Hold signal of each pin.
- Global Hold signal:
  - Global Hold signal of HP GPIO pins:
    - PMU_TIE_HIGH_HP_PAD_HOLD_ALL: enables the global Hold signal of all HP GPIO pins.
    - PMU_TIE_LOW_HP_PAD_HOLD_ALL: disables the global Hold signal of all HP GPIO pins.
  - Global Hold signal of LP GPIO pins:
    - PMU_TIE_HIGH_LP_PAD_HOLD_ALL: enables the global Hold signal of all LP GPIO pins.
    - PMU_TIE_LOW_LP_PAD_HOLD_ALL: disables the global Hold signal of all LP GPIO pins.

Enable or disable the Hold feature of the pins by one of the following ways:

- **Method 1: enable or disable the Hold feature of individual pins:**
  To maintain the pin's input/output status in Deep-sleep, set LP_AON_GPIO_HOLDO_REG[*n*] (where *n* = 0∼13, 22∼29 corresponds to GPIO0∼GPIO13, GPIO22∼GPIO29). To disable the hold function after waking up, clear the bit[*n*].
- **Method 2: enable or disable the Hold feature of all HP/LP pins:**
  - HP Pins
    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.
  - LP Pins
    Set PMU_TIE_HIGH_LP_PAD_HOLD_ALL to maintain the input/output status of all LP pins and set PMU_TIE_LOW_LP_PAD_HOLD_ALL to disable the hold function for all LP pins.
- **Method 3: Use PMU Task**
  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.

## 7.10 Hysteresis Characteristics

Each GPIO pin has hysteresis functionality. When hysteresis is not enabled, as shown in Figure 7.10-1, 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.

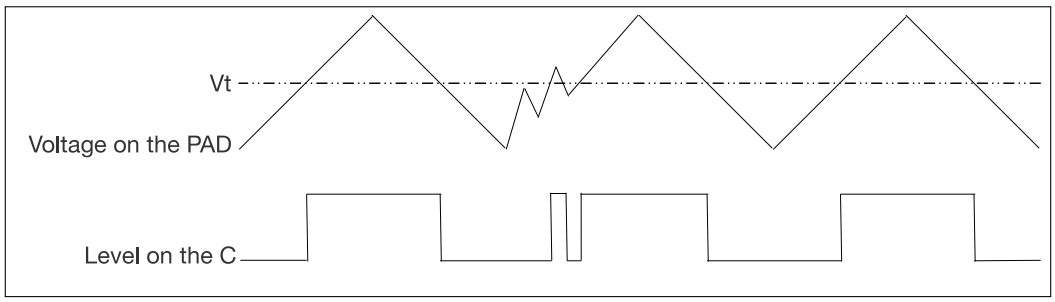



**Figure 7.10-1. Example of Level Flip on the Chip Pad — Hysteresis Function Not Enabled**

When hysteresis is enabled, as shown in Figure 7.10-2, 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.

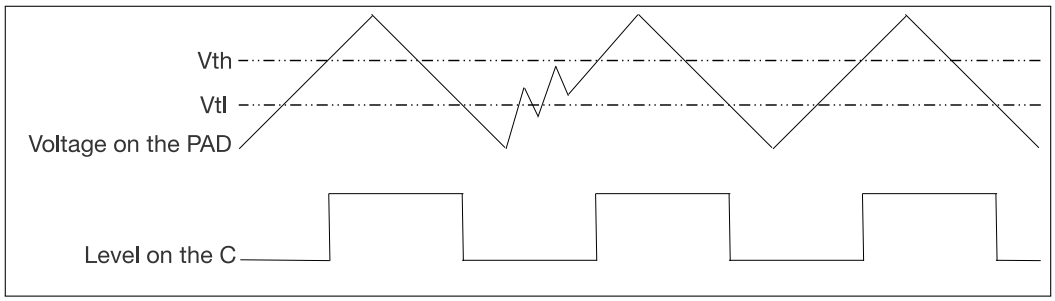



**Figure 7.10-2. Example of Level Flip on the Chip Pad — Hysteresis Function Enabled**

To enable the hysteresis function, follow the steps below:

- IO_MUX_GPIO*n*_HYS_SEL = 0 (*n* = 0∼13, 22∼29, corresponding to GPIO0∼GPIO13, GPIO22∼GPIO29)
  - Set EFUSE_HYS_EN_PAD to enable the hysteresis function for GPIO*n*.
  - Or clear EFUSE_HYS_EN_PAD to disable the hysteresis function for GPIO*n*.
- IO_MUX_GPIO*n*_HYS_SEL = 1 (*n* = 0∼13, 22∼29, corresponding to GPIO0∼GPIO13, GPIO22∼GPIO29)
  - Set IO_MUX_GPIO*n*_HYS_EN to enable the hysteresis function for GPIO*n*.
  - Clear IO_MUX_GPIO*n*_HYS_EN to disable the hysteresis function for GPIO*n*.

**Recommended Operation**:

- Set IO_MUX_GPIO*n*_HYS_SEL.
- Then enable or disable the the hysteresis function for GPIO*n* using IO_MUX_GPIO*n*_HYS_EN.

## 7.11 Power Supplies and Management of GPIO Pins

### 7.11.1 Power Supplies of GPIO Pins

For more information on the power supply for GPIO pins, please refer to Pin Definition in *ESP32-C61 Datasheet*. All the pins can be used to wake up the chip from Light-sleep, but only the pins (GPIO0~GPIO6) in VDDPST1 domain can be used to wake up the chip from Deep-sleep.

### 7.11.2 Power Supply Management

Each ESP32-C61 pin is connected to one of the following power domains.

- VDDPST1: the input power supply for LP GPIO pins
- VDDPST2: the input power supply for HP GPIO pins

## 7.12 HP Peripheral Signal List

Table 7.12-1 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_REG:
  - GPIO_ENABLE_REG = 0: output is disabled.
  - GPIO_ENABLE_REG = 1: output is enabled.
- GPIO_FUNC*n*_OE_SEL = 0: use the output enable signal from peripheral, for example FSPIQ_oe in the column "Output enable signal when GPIO_FUNC*n*_OE_SEL = 0" of Table 7.12-1. Note that the signals such as FSPIQ_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" or in the column "Output signal" in Table 7.12-1 are valid input or output signals.

**Table 7.12-1. 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 | — | - | - | ledc_ls_sig_out0 | 1'd1 | no |
| 1 | — | — | — | ledc_ls_sig_out1 | 1'd1 | no |
| 2 | — | — | — | ledc_ls_sig_out2 | 1'd1 | no |
| 3 | — | — | — | ledc_ls_sig_out3 | 1'd1 | no |
| 4 | — | — | — | ledc_ls_sig_out4 | 1'd1 | no |
| 5 | — | — | — | ledc_ls_sig_out5 | 1'd1 | no |
| 6 | U0RXD_in | 0 | yes | U0TXD_out | 1'd1 | yes |

| 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 |
|---|---|---|---|---|---|---|
| 7 | U0CTS_in | 0 | no | U0RTS_out | 1'd1 | no |
| 8 | U0DSR_in | 0 | no | U0DTR_out | 1'd1 | no |
| 9 | U1RXD_in | 1 | no | U1TXD_out | 1'd1 | no |
| 10 | U1CTS_in | 0 | no | U1RTS_out | 1'd1 | no |
| 11 | U1DSR_in | 0 | no | U1DTR_out | 1'd1 | no |
| 12 | I2S_MCLK_in | 0 | no | I2S_MCLK_out | 1'd1 | no |
| 13 | I2SO_BCK_in | 0 | no | I2SO_BCK_out | 1'd1 | no |
| 14 | I2SO_WS_in | 0 | no | I2SO_WS_out | 1'd1 | no |
| 15 | I2SI_SD_in | 0 | no | I2SO_SD_out | 1'd1 | no |
| 16 | I2SI_BCK_in | 0 | no | I2SI_BCK_out | 1'd1 | no |
| 17 | I2SI_WS_in | 0 | no | I2SI_WS_out | 1'd1 | no |
| 18 | — | — | — | I2SO_SD1_out | 1'd1 | no |
| 19 | — | — | — | — | — | — |
| 20 | — | — | — | — | — | — |
| 21 | — | — | — | — | — | — |
| 22 | — | — | — | — | — | — |
| 23 | — | — | — | — | — | — |
| 24 | — | — | — | — | — | — |
| 25 | — | — | — | — | — | — |
| 26 | — | — | — | — | — | — |
| 27 | cpu_gpio_in0 | 0 | no | cpu_gpio_out0 | cpu_gpio_out_oen0 | no |
| 28 | cpu_gpio_in1 | 0 | no | cpu_gpio_out1 | cpu_gpio_out_oen1 | no |
| 29 | cpu_gpio_in2 | 0 | no | cpu_gpio_out2 | cpu_gpio_out_oen2 | no |
| 30 | cpu_gpio_in3 | 0 | no | cpu_gpio_out3 | cpu_gpio_out_oen3 | no |
| 31 | cpu_gpio_in4 | 0 | no | cpu_gpio_out4 | cpu_gpio_out_oen4 | no |
| 32 | cpu_gpio_in5 | 0 | no | cpu_gpio_out5 | cpu_gpio_out_oen5 | no |
| 33 | cpu_gpio_in6 | 0 | no | cpu_gpio_out6 | cpu_gpio_out_oen6 | no |
| 34 | cpu_gpio_in7 | 0 | no | cpu_gpio_out7 | cpu_gpio_out_oen7 | no |
| 35 | usb_jtag_tdo | 0 | no | — | — | — |
| 36 | — | — | — | — | — | — |
| 37 | — | — | — | — | — | — |
| 38 | — | — | — | — | — | — |
| 39 | — | — | — | — | — | — |
| 40 | — | — | — | — | — | — |
| 41 | — | — | — | — | — | — |
| 42 | — | — | — | — | — | — |
| 43 | — | — | — | — | — | — |
| 44 | — | — | — | — | — | — |
| 45 | — | — | — | — | — | — |
| 46 | I2CEXT0_SCL_in | 1 | no | I2CEXT0_SCL_out | I2CEXT0_SCL_oe | no |
| 47 | I2CEXT0_SDA_in | 1 | no | I2CEXT0_SDA_out | I2CEXT0_SDA_oe | no |
| 48 | — | — | — | — | — | — |
| 49 | — | — | — | — | — | — |
| 50 | — | — | — | — | — | — |

| 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 |
|---|---|---|---|---|---|---|
| 51 | — | — | — | — | — | — |
| 52 | — | — | — | — | — | — |
| 43 | — | — | — | — | — | — |
| 54 | — | — | — | — | — | — |
| 55 | — | — | — | — | — | — |
| 56 | — | — | — | — | — | — |
| 57 | — | — | — | — | — | — |
| 58 | — | — | — | — | — | — |
| 59 | — | — | — | — | — | — |
| 60 | — | — | — | — | — | — |
| 61 | — | — | — | — | — | — |
| 62 | — | — | — | — | — | — |
| 63 | — | — | — | — | — | — |
| 64 | FSPICLK_in | 0 | yes | FSPICLK_out | FSPICLK_oe | yes |
| 65 | FSPIQ_in | 0 | yes | FSPIQ_out | FSPIQ_oe | yes |
| 66 | FSPID_in | 0 | yes | FSPID_out | FSPID_oe | yes |
| 67 | FSPIHD_in | 0 | yes | FSPIHD_out | FSPIHD_oe | yes |
| 68 | FSPIWP_in | 0 | yes | FSPIWP_out | FSPIWP_oe | yes |
| 69 | FSPICS0_in | 0 | yes | FSPICS0_out | FSPICS0_oe | yes |
| 70 | — | — | — | — | — | — |
| 71 | — | — | — | — | — | — |
| 72 | U2RXD_in | 1 | no | U2TXD_out | 1'd1 | no |
| 73 | U2CTS_in | 0 | no | U2RTS_out | 1'd1 | no |
| 74 | U2DSR_in | 0 | no | U2DTR_out | 1'd1 | no |
| 75 | — | — | — | — | — | — |
| 76 | — | — | — | — | — | — |
| 77 | — | — | — | — | — | — |
| 78 | — | — | — | — | — | — |
| 79 | — | — | — | — | — | — |
| 80 | — | — | — | — | — | — |
| 81 | — | — | — | — | — | — |
| 82 | — | — | — | — | — | — |
| 83 | — | — | — | — | — | — |
| 84 | — | — | — | — | — | — |
| 85 | — | — | — | — | — | — |
| 86 | — | — | — | — | — | — |
| 87 | — | — | — | — | — | — |
| 88 | — | — | — | — | — | — |
| 89 | — | — | — | — | — | — |
| 90 | — | — | — | — | — | — |
| 91 | — | — | — | — | — | — |
| 92 | — | — | — | — | — | — |
| 93 | — | — | — | — | — | — |
| 94 | — | — | — | — | — | — |

| 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 |
|---|---|---|---|---|---|---|
| 95 | — | — | — | — | — | — |
| 96 | — | — | — | — | — | — |
| 97 | sig_in_func_97 | 0 | no | sig_in_func97 | 1'd1 | no |
| 98 | sig_in_func_98 | 0 | no | sig_in_func98 | 1'd1 | no |
| 99 | sig_in_func_99 | 0 | no | sig_in_func99 | 1'd1 | no |
| 100 | sig_in_func_100 | 0 | no | sig_in_func100 | 1'd1 | no |
| 101 | — | — | — | — | — | — |
| 102 | — | — | — | FSPICS1_out | FSPICS1_oe | yes |
| 103 | — | — | — | FSPICS2_out | FSPICS2_oe | yes |
| 104 | — | — | — | FSPICS3_out | FSPICS3_oe | yes |
| 105 | — | — | — | FSPICS4_out | FSPICS4_oe | yes |
| 106 | — | — | — | FSPICS5_out | FSPICS5_oe | yes |
| 107 | — | — | — | — | — | — |
| 108 | — | — | — | — | — | — |
| 109 | — | — | — | — | — | — |
| 110 | — | — | — | — | — | — |
| 111 | — | — | — | — | — | — |
| 112 | — | — | — | — | — | — |
| 113 | — | — | — | — | — | — |
| 114 | — | — | — | — | — | — |
| 115 | — | — | — | — | — | — |
| 116 | — | — | — | — | — | — |
| 117 | — | — | — | — | — | — |
| 118 | — | — | — | — | — | — |
| 119 | — | — | — | — | — | — |
| 120 | — | — | — | — | — | — |
| 121 | — | — | — | — | — | — |
| 122 | — | — | — | — | — | — |
| 123 | — | — | — | — | — | — |
| 124 | — | — | — | sdio_tohost_int_out | 1'd1 | no |
| 125 ~ 255 | — | — | — | — | — | — |

## 7.13 HP IO MUX Function List

Table 7.13-1 shows the HP IO MUX functions and default states of each HP GPIO pin.

Table 7.13-1. HP IO MUX Pin Functions

| GPIO No. | Pin Name | Function 0 | Function 1 | Function 2 | Function 3 | DRV | Reset | Notes |
|---|---|---|---|---|---|---|---|---|
| 0 | XTAL_32K_P | GPIO0 | GPIO0 | — | — | 2 | 0 | R |
| 1 | XTAL_32K_N | GPIO1 | GPIO1 | — | — | 2 | 0 | R |
| 2 | GPIO2 | GPIO2 | GPIO2 | FSPIQ | — | 2 | 0 | R |
| 3 | MTMS | MTMS | GPIO3 | FSPIHD | — | 2 | 1 | R |
| 4 | MTDI | MTDI | GPIO4 | FSPIWP | — | 2 | 1 | R |
| 5 | MTCK | MTCK | GPIO5 | — | — | 2 | 1* | R |
| 6 | MTDO | MTDO | GPIO6 | FSPICLK | — | 2 | 1 | R |
| 7 | GPIO7 | GPIO7 | GPIO7 | FSPID | — | 2 | 1 | R |
| 8 | GPIO8 | GPIO8 | GPIO8 | FSPICS0 | — | 2 | 1 | |
| 9 | GPIO9 | GPIO9 | GPIO9 | — | — | 2 | 3 | — |
| 10 | U0RXD | U0RXD | GPIO10 | — | — | 2 | 1 | — |
| 11 | U0TXD | U0TXD | GPIO11 | — | — | 2 | 4 | — |
| 12 | GPIO12 | GPIO12 | GPIO12 | — | — | 3 | 1 | USB |
| 13 | GPIO13 | GPIO13 | GPIO13 | — | — | 3 | 3* | USB |
| 22 | SDIO_DATA2 | SDIO_DATA2 | GPIO22 | — | — | 2 | 1 | — |
| 23 | SDIO_DATA3 | SDIO_DATA3 | GPIO23 | — | — | 2 | 1 | — |
| 24 | GPIO24 | GPIO24 | GPIO24 | — | — | 2 | 0 | — |
| 25 | SDIO_CMD | SDIO_CMD | GPIO25 | — | — | 2 | 1 | — |
| 26 | SDIO_CLK | SDIO_CLK | GPIO26 | — | — | 2 | 1 | — |
| 27 | SDIO_DATA0 | SDIO_DATA0 | GPIO27 | — | — | 2 | 1 | — |
| 28 | SDIO_DATA1 | SDIO_DATA1 | GPIO28 | — | — | 2 | 1 | — |
| 29 | GPIO29 | GPIO29 | GPIO29 | — | — | 2 | 0 | — |

**Notice:**
In HP IO MUX, unused pins must be configured to GPIO function.

### 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** - IE = 0 (input disabled)
- **1** - IE = 1 (input enabled)

- **3** - IE = 1, WPU = 1 (input enabled, pull-up resistor enabled)
- **4** - IE = 1, OE = 1 (input enabled, output enabled)
- **1*** - If EFUSE_DIS_PAD_JTAG = 1, the pin MTCK is left floating after reset, i.e., IE = 1. If EFUSE_DIS_PAD_JTAG = 0, the pin MTCK is connected to internal pull-up resistor, i.e., IE = 1, WPU = 1.
- **3*** - IE = 1, WPU = 0. The default value of GPIO14's USB pull-up is 1, which means the pull-up resistor is enabled. For details, please refer to the **Notes** below.

**Notes**

- **R** - LP pins. Some LP pins have analog functions. For details, see 7.15-1.
- **USB** - USB pull-up resistor enabled
  - By default, the USB function is enabled for USB pins (i.e., GPIO13 and GPIO14), and the pin pull-up is decided by the USB pull-up. The USB pull-up is controlled by USB_SERIAL_JTAG_DP/DM_PULLUP and the pull-up resistor value is controlled by USB_SERIAL_JTAG_PULLUP_VALUE. For details, see Chapter USB Serial/JTAG Controller.
  - When the USB function is disabled, USB pins are used as regular GPIOs and the pin's internal weak pull-up and pull-down resistors are disabled by default (configurable by IO_MUX_GPIO*n*_MCU_WPU/WPD).

## 7.14 LP IO MUX Function List

Table 7.14-1 shows the LP IO MUX functions of each GPIO pin. GPIO pins are default controlled by HP IO MUX. Table 7.14-1 only show the LP IO MUX functions of LP GPIO pins.

**Table 7.14-1. LP IO MUX Pin Functions**

| GPIO No. | Name | Function 0 | Function 1 | Function 2 | Function 3 |
|---|---|---|---|---|---|
| 0 | XTAL_32K_P | — | LP_GPIO0 | — | — |
| 1 | XTAL_32K_N | — | LP_GPIO1 | — | — |
| 2 | GPIO2 | — | LP_GPIO2 | — | — |
| 3 | MTMS | — | LP_GPIO3 | — | — |
| 4 | MTDI | — | LP_GPIO4 | — | — |
| 5 | MTCK | — | LP_GPIO5 | — | — |
| 6 | MTDO | — | LP_GPIO6 | — | — |

**Notice:**
In LP IO MUX, unused pins must be configured to LP_GPIO function.

## 7.15 GPIO Pin Analog Function List

Table 7.15-1 shows the GPIO pins and their corresponding analog functions.

Table 7.15-1. GPIO Pin Analog Functions

| GPIO No. | Pin Name | Analog Function 0 | Analog Function 1 |
|---|---|---|---|
| 0 | XTAL_32K_P | XTAL_32K_P | |
| 1 | XTAL_32K_N | XTAL_32K_N | ADC1_CH0 |
| 3 | MTMS | — | ADC1_CH1 |
| 4 | MTDI | — | ADC1_CH2 |
| 5 | MTCK | — | ADC1_CH3 |
| 8 | GPIO8 | ZCD0 | — |
| 9 | GPIO9 | ZCD1 | — |
| 12 | GPIO12 | USB_D- | — |
| 13 | GPIO13 | USB_D+ | — |

**Note:**

- GPIO1 has two different analog functions, which can be used simultaneously.
- ZCD0 and ZCD1 are Analog Voltage Comparator functions. See Chapter 35 *Analog Voltage Comparator* for details.

## 7.16 Event Task Matrix Function

In ESP32-C61, 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. This section describes only the GPIO-related ETM tasks and events. For a comprehensive description of the ETM system, please refer to 11 *Event Task Matrix (ETM)*.

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 7.13-1.
- 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 more of the following signals for task channel *x*: GPIO_TASK_CH*x*_SET, GPIO_TASK_CH*x*_CLEAR, and GPIO_TASK_CH*x*_TOGGLE, are asserted at the same time by GPIO*y*, the priority is as follows: SET has the highest priority, CLEAR is second, and TOGGLE has the lowest priority.

- When GPIO*y* is controlled by ETM task channel, the values of GPIO_OUT_REG, GPIO_FUNC*y*_OUT_INV_SEL, and GPIO_FUNC*y*_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: The output signal of the corresponding GPIO Filter (see Figure 7.3-1) has a rising edge.
- GPIO_EVT_CH*x*_FALL_EDGE: The output signal of the corresponding GPIO Filter (see Figure 7.3-1) has a falling edge.
- GPIO_EVT_CH*x*_ANY_EDGE: The output signal of the corresponding GPIO Filter (see Figure 7.3-1) 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∼13, 22∼29), i.e., select one from the 22 GPIOs.

**Note:**
One GPIO can be selected by one or more event channels.

In some 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.

## 7.17 Interrupts

ESP32-C61's HP IO MUX and HP GPIO matrix can generate the following interrupt signals that will be sent to the Interrupt Matrix.

- GPIO_EXT_REG_INT
- GPIO_PROCPU_INT

Several internal interrupt sources from HP IO MUX and HP GPIO matrix can generate the above interrupt signals. Table 7.17-1 lists the interrupt sources from HP IO MUX and HP GPIO matrix with their trigger conditions and the resulted interrupt signals.

**Table 7.17-1. HP IO MUX and HP GPIO Matrix's Internal Interrupt Sources**

| Internal Interrupt Source | Trigger Condition | Interrupt Signal |
|---|---|---|
| GPIO_EXT_COMP_ALL_0_INT | See Chapter 35 *Analog Voltage Comparator* | GPIO_EXT_REG_INT |
| GPIO_EXT_COMP_NEG_0_INT | See Chapter 35 *Analog Voltage Comparator* | GPIO_EXT_REG_INT |
| GPIO_EXT_COMP_POS_0_INT | See Chapter 35 *Analog Voltage Comparator* | GPIO_EXT_REG_INT |

| | | |
|---|---|---|
| GPIO_REG_INTR | GPIO_STATUS_INTERRUPT[*n*] & GPIO_PIN*n*_INT_ENA[0] (*n*: 0∼13, 22∼29) | GPIO_PROCPU_INT |

ESP32-C61's LP IO MUX and LP GPIO matrix can generate the following interrupt signal that will be sent to the LP_CORE.

- LP_GPIO_INTR

The interrupt source from LP IO MUX and LP GPIO matrix is listed with its trigger condition and the resulted interrupt signal in Table 7.17-2.

**Table 7.17-2. LP IO MUX and LP GPIO Matrix's Internal Interrupt Sources**

| Internal Interrupt Source | Trigger Condition | Interrupt Signal |
|---|---|---|
| LP_GPIO_REG_INTR | LP_GPIO_STATUS_INTERRUPT[*n*] (*n*: 0∼6) | LP_GPIO_INTR |

# 7.18 Register Summary

## 7.18.1 HP GPIO Matrix Register Summary

The addresses in this section are relative to HP GPIO matrix base address provided in Table 5.3-2 in Chapter 5 *System and Memory*.

The abbreviations given in Column **Access** are explained in Section *Access Types for Registers*.

| Name | Description | Address | Access |
|---|---|---|---|
| **Configuration Registers** | | | |
| GPIO_STRAP_REG | Strapping pin register | 0x0000 | RO |
| GPIO_OUT_REG | GPIO output register | 0x0004 | R/W/SC/WTC |
| GPIO_OUT_W1TS_REG | GPIO output set register | 0x0008 | WT |
| GPIO_OUT_W1TC_REG | GPIO output clear register | 0x000C | WT |
| GPIO_ENABLE_REG | GPIO output enable register | 0x0034 | R/W/WTC |
| GPIO_ENABLE_W1TS_REG | GPIO output enable set register | 0x0038 | WT |
| GPIO_ENABLE_W1TC_REG | GPIO output enable clear register | 0x003C | WT |
| GPIO_IN_REG | GPIO input register | 0x0064 | RO |
| **Interrupt Status Registers** | | | |
| GPIO_STATUS_REG | GPIO interrupt status register | 0x0074 | R/W/WTC |
| GPIO_STATUS_W1TS_REG | GPIO interrupt status set register | 0x0078 | WT |
| GPIO_STATUS_W1TC_REG | GPIO interrupt status clear register | 0x007C | WT |
| GPIO_PROCPU_INT_REG | GPIO_PROCPU_INT interrupt status register | 0x00A4 | RO |
| GPIO_SDIO_INT_REG | GPIO_SDIO_INT interrupt status register | 0x00A8 | RO |
| GPIO_STATUS_NEXT_REG | GPIO interrupt source register | 0x00C4 | RO |
| **Pin Configuration Registers** | | | |
| GPIO_PIN0_REG | GPIO0 configuration register | 0x00D4 | R/W |
| GPIO_PIN1_REG | GPIO1 configuration register | 0x00D8 | R/W |

| Name | Description | Address | Access |
|---|---|---|---|
| GPIO_PIN2_REG | GPIO2 configuration register | 0x00DC | R/W |
| GPIO_PIN3_REG | GPIO3 configuration register | 0x00E0 | R/W |
| GPIO_PIN4_REG | GPIO4 configuration register | 0x00E4 | R/W |
| GPIO_PIN5_REG | GPIO5 configuration register | 0x00E8 | R/W |
| GPIO_PIN6_REG | GPIO6 configuration register | 0x00EC | R/W |
| GPIO_PIN7_REG | GPIO7 configuration register | 0x00F0 | R/W |
| GPIO_PIN8_REG | GPIO8 configuration register | 0x00F4 | R/W |
| GPIO_PIN9_REG | GPIO9 configuration register | 0x00F8 | R/W |
| GPIO_PIN10_REG | GPIO10 configuration register | 0x00FC | R/W |
| GPIO_PIN11_REG | GPIO11 configuration register | 0x0100 | R/W |
| GPIO_PIN12_REG | GPIO12 configuration register | 0x0104 | R/W |
| GPIO_PIN13_REG | GPIO13 configuration register | 0x0108 | R/W |
| GPIO_PIN22_REG | GPIO22 configuration register | 0x012C | R/W |
| GPIO_PIN23_REG | GPIO23 configuration register | 0x0130 | R/W |
| GPIO_PIN24_REG | GPIO24 configuration register | 0x0134 | R/W |
| GPIO_PIN25_REG | GPIO25 configuration register | 0x0138 | R/W |
| GPIO_PIN26_REG | GPIO26 configuration register | 0x013C | R/W |
| GPIO_PIN27_REG | GPIO27 configuration register | 0x0140 | R/W |
| GPIO_PIN28_REG | GPIO28 configuration register | 0x0144 | R/W |
| GPIO_PIN29_REG | GPIO29 configuration register | 0x0148 | R/W |
| **Input Configuration Registers** | | | |
| GPIO_FUNC6_IN_SEL_CFG_REG | Configuration register for input signal 6 | 0x02EC | R/W |
| GPIO_FUNC7_IN_SEL_CFG_REG | Configuration register for input signal 7 | 0x02F0 | R/W |
| GPIO_FUNC8_IN_SEL_CFG_REG | Configuration register for input signal 8 | 0x02F4 | R/W |
| GPIO_FUNC9_IN_SEL_CFG_REG | Configuration register for input signal 9 | 0x02F8 | R/W |
| GPIO_FUNC10_IN_SEL_CFG_REG | Configuration register for input signal 10 | 0x02FC | R/W |
| GPIO_FUNC11_IN_SEL_CFG_REG | Configuration register for input signal 11 | 0x0300 | R/W |
| GPIO_FUNC12_IN_SEL_CFG_REG | Configuration register for input signal 12 | 0x0304 | R/W |
| GPIO_FUNC13_IN_SEL_CFG_REG | Configuration register for input signal 13 | 0x0308 | R/W |
| GPIO_FUNC14_IN_SEL_CFG_REG | Configuration register for input signal 14 | 0x030C | R/W |
| GPIO_FUNC15_IN_SEL_CFG_REG | Configuration register for input signal 15 | 0x0310 | R/W |
| GPIO_FUNC16_IN_SEL_CFG_REG | Configuration register for input signal 16 | 0x0314 | R/W |
| GPIO_FUNC17_IN_SEL_CFG_REG | Configuration register for input signal 17 | 0x0318 | R/W |
| GPIO_FUNC27_IN_SEL_CFG_REG | Configuration register for input signal 27 | 0x0340 | R/W |
| GPIO_FUNC28_IN_SEL_CFG_REG | Configuration register for input signal 28 | 0x0344 | R/W |
| GPIO_FUNC29_IN_SEL_CFG_REG | Configuration register for input signal 29 | 0x0348 | R/W |
| GPIO_FUNC30_IN_SEL_CFG_REG | Configuration register for input signal 30 | 0x034C | R/W |
| GPIO_FUNC31_IN_SEL_CFG_REG | Configuration register for input signal 31 | 0x0350 | R/W |
| GPIO_FUNC32_IN_SEL_CFG_REG | Configuration register for input signal 32 | 0x0354 | R/W |
| GPIO_FUNC33_IN_SEL_CFG_REG | Configuration register for input signal 33 | 0x0358 | R/W |
| GPIO_FUNC34_IN_SEL_CFG_REG | Configuration register for input signal 34 | 0x035C | R/W |
| GPIO_FUNC35_IN_SEL_CFG_REG | Configuration register for input signal 35 | 0x0360 | R/W |
| GPIO_FUNC46_IN_SEL_CFG_REG | Configuration register for input signal 46 | 0x038C | R/W |

| Name | Description | Address | Access |
|---|---|---|---|
| GPIO_FUNC47_IN_SEL_CFG_REG | Configuration register for input signal 47 | 0x0390 | R/W |
| GPIO_FUNC64_IN_SEL_CFG_REG | Configuration register for input signal 64 | 0x03D4 | R/W |
| GPIO_FUNC65_IN_SEL_CFG_REG | Configuration register for input signal 65 | 0x03D8 | R/W |
| GPIO_FUNC66_IN_SEL_CFG_REG | Configuration register for input signal 66 | 0x03DC | R/W |
| GPIO_FUNC67_IN_SEL_CFG_REG | Configuration register for input signal 67 | 0x03E0 | R/W |
| GPIO_FUNC68_IN_SEL_CFG_REG | Configuration register for input signal 68 | 0x03E4 | R/W |
| GPIO_FUNC69_IN_SEL_CFG_REG | Configuration register for input signal 69 | 0x03E8 | R/W |
| GPIO_FUNC72_IN_SEL_CFG_REG | Configuration register for input signal 72 | 0x03F4 | R/W |
| GPIO_FUNC73_IN_SEL_CFG_REG | Configuration register for input signal 73 | 0x03F8 | R/W |
| GPIO_FUNC74_IN_SEL_CFG_REG | Configuration register for input signal 74 | 0x03FC | R/W |
| GPIO_FUNC97_IN_SEL_CFG_REG | Configuration register for input signal 97 | 0x0458 | R/W |
| GPIO_FUNC98_IN_SEL_CFG_REG | Configuration register for input signal 98 | 0x045C | R/W |
| GPIO_FUNC99_IN_SEL_CFG_REG | Configuration register for input signal 99 | 0x0460 | R/W |
| GPIO_FUNC100_IN_SEL_CFG_REG | Configuration register for input signal 100 | 0x0464 | R/W |
| **Output Configuration Registers** | | | |
| GPIO_FUNC0_OUT_SEL_CFG_REG | Configuration register for GPIO0 output | 0x0AD4 | varies |
| GPIO_FUNC1_OUT_SEL_CFG_REG | Configuration register for GPIO1 output | 0x0AD8 | varies |
| GPIO_FUNC2_OUT_SEL_CFG_REG | Configuration register for GPIO2 output | 0x0ADC | varies |
| GPIO_FUNC3_OUT_SEL_CFG_REG | Configuration register for GPIO3 output | 0x0AE0 | varies |
| GPIO_FUNC4_OUT_SEL_CFG_REG | Configuration register for GPIO4 output | 0x0AE4 | varies |
| GPIO_FUNC5_OUT_SEL_CFG_REG | Configuration register for GPIO5 output | 0x0AE8 | varies |
| GPIO_FUNC6_OUT_SEL_CFG_REG | Configuration register for GPIO6 output | 0x0AEC | varies |
| GPIO_FUNC7_OUT_SEL_CFG_REG | Configuration register for GPIO7 output | 0x0AF0 | varies |
| GPIO_FUNC8_OUT_SEL_CFG_REG | Configuration register for GPIO8 output | 0x0AF4 | varies |
| GPIO_FUNC9_OUT_SEL_CFG_REG | Configuration register for GPIO9 output | 0x0AF8 | varies |
| GPIO_FUNC10_OUT_SEL_CFG_REG | Configuration register for GPIO10 output | 0x0AFC | varies |
| GPIO_FUNC11_OUT_SEL_CFG_REG | Configuration register for GPIO11 output | 0x0B00 | varies |
| GPIO_FUNC12_OUT_SEL_CFG_REG | Configuration register for GPIO12 output | 0x0B04 | varies |
| GPIO_FUNC13_OUT_SEL_CFG_REG | Configuration register for GPIO13 output | 0x0B08 | varies |
| GPIO_FUNC22_OUT_SEL_CFG_REG | Configuration register for GPIO22 output | 0x0B2C | varies |
| GPIO_FUNC23_OUT_SEL_CFG_REG | Configuration register for GPIO23 output | 0x0B30 | varies |
| GPIO_FUNC24_OUT_SEL_CFG_REG | Configuration register for GPIO24 output | 0x0B34 | varies |
| GPIO_FUNC25_OUT_SEL_CFG_REG | Configuration register for GPIO25 output | 0x0B38 | varies |
| GPIO_FUNC26_OUT_SEL_CFG_REG | Configuration register for GPIO26 output | 0x0B3C | varies |
| GPIO_FUNC27_OUT_SEL_CFG_REG | Configuration register for GPIO27 output | 0x0B40 | varies |
| GPIO_FUNC28_OUT_SEL_CFG_REG | Configuration register for GPIO28 output | 0x0B44 | varies |
| GPIO_FUNC29_OUT_SEL_CFG_REG | Configuration register for GPIO29 output | 0x0B48 | varies |
| **Clock Gate Register** | | | |
| GPIO_CLOCK_GATE_REG | GPIO clock gate register | 0x0DF8 | R/W |
| **Version Register** | | | |
| GPIO_DATE_REG | Version control register | 0x0DFC | R/W |

## 7.18.2 HP IO MUX Register Summary

The addresses in this section are relative to the HP IO MUX base address provided in Table 5.3-2 in Chapter 5 *System and Memory*.

The abbreviations given in Column **Access** are explained in Section *Access Types for Registers*.

| Name | Description | Address | Access |
|---|---|---|---|
| **Configuration Registers** | | | |
| IO_MUX_GPIO0_REG | HP IO MUX configuration register for GPIO0 | 0x0000 | R/W |
| IO_MUX_GPIO1_REG | HP IO MUX configuration register for GPIO1 | 0x0004 | R/W |
| IO_MUX_GPIO2_REG | HP IO MUX configuration register for GPIO2 | 0x0008 | R/W |
| IO_MUX_GPIO3_REG | HP IO MUX configuration register for GPIO3 | 0x000C | R/W |
| IO_MUX_GPIO4_REG | HP IO MUX configuration register for GPIO4 | 0x0010 | R/W |
| IO_MUX_GPIO5_REG | HP IO MUX configuration register for GPIO5 | 0x0014 | R/W |
| IO_MUX_GPIO6_REG | HP IO MUX configuration register for GPIO6 | 0x0018 | R/W |
| IO_MUX_GPIO7_REG | HP IO MUX configuration register for GPIO7 | 0x001C | R/W |
| IO_MUX_GPIO8_REG | HP IO MUX configuration register for GPIO8 | 0x0020 | R/W |
| IO_MUX_GPIO9_REG | HP IO MUX configuration register for GPIO9 | 0x0024 | R/W |
| IO_MUX_GPIO10_REG | HP IO MUX configuration register for GPIO10 | 0x0028 | R/W |
| IO_MUX_GPIO11_REG | HP IO MUX configuration register for GPIO11 | 0x002C | R/W |
| IO_MUX_GPIO12_REG | HP IO MUX configuration register for GPIO12 | 0x0030 | R/W |
| IO_MUX_GPIO13_REG | HP IO MUX configuration register for GPIO13 | 0x0034 | R/W |
| IO_MUX_GPIO22_REG | HP IO MUX configuration register for GPIO22 | 0x0058 | R/W |
| IO_MUX_GPIO23_REG | HP IO MUX configuration register for GPIO23 | 0x005C | R/W |
| IO_MUX_GPIO24_REG | HP IO MUX configuration register for GPIO24 | 0x0060 | R/W |
| IO_MUX_GPIO25_REG | HP IO MUX configuration register for GPIO25 | 0x0064 | R/W |
| IO_MUX_GPIO26_REG | HP IO MUX configuration register for GPIO26 | 0x0068 | R/W |
| IO_MUX_GPIO27_REG | HP IO MUX configuration register for GPIO27 | 0x006C | R/W |
| IO_MUX_GPIO28_REG | HP IO MUX configuration register for GPIO28 | 0x0070 | R/W |
| IO_MUX_GPIO29_REG | HP IO MUX configuration register for GPIO29 | 0x0074 | R/W |
| **Version Register** | | | |
| IO_MUX_DATE_REG | Version control register | 0x01FC | R/W |

## 7.18.3 GPIO EXT Register Summary

GPIO EXT registers consist of Analog Voltage Comparator and ETM registers.

The addresses in this section are relative to GPIO EXT base address provided in Table 5.3-2 in Chapter 5 *System and Memory*.

The abbreviations given in Column **Access** are explained in Section *Access Types for Registers*.

| Name | Description | Address | Access |
|---|---|---|---|
| **Analog Voltage Comparator Configuration Registers** | | | |
| GPIO_EXT_PAD_COMP_CONFIG_0_REG | Analog Voltage Comparator configuration register | 0x0058 | R/W |

| Name | Description | Address | Access |
|---|---|---|---|
| GPIO_EXT_PAD_COMP_FILTER_0_REG | Zero-crossing detection register | 0x005C | R/W |
| **ETM Configuration Registers** | | | |
| GPIO_EXT_ETM_EVENT_CH0_CFG_REG | Configuration register of ETM channel 0 | 0x0118 | R/W |
| GPIO_EXT_ETM_EVENT_CH1_CFG_REG | Configuration register of ETM channel 1 | 0x011C | R/W |
| GPIO_EXT_ETM_EVENT_CH2_CFG_REG | Configuration register of ETM channel 2 | 0x0120 | R/W |
| GPIO_EXT_ETM_EVENT_CH3_CFG_REG | Configuration register of ETM channel 3 | 0x0124 | R/W |
| GPIO_EXT_ETM_EVENT_CH4_CFG_REG | Configuration register of ETM channel 4 | 0x0128 | R/W |
| GPIO_EXT_ETM_EVENT_CH5_CFG_REG | Configuration register of ETM channel 5 | 0x012C | R/W |
| GPIO_EXT_ETM_EVENT_CH6_CFG_REG | Configuration register of ETM channel 6 | 0x0130 | R/W |
| GPIO_EXT_ETM_EVENT_CH7_CFG_REG | Configuration register of ETM channel 7 | 0x0134 | R/W |
| GPIO_EXT_ETM_TASK_P0_CFG_REG | GPIO selection register 0 | 0x0158 | R/W |
| GPIO_EXT_ETM_TASK_P1_CFG_REG | GPIO selection register 1 | 0x015C | R/W |
| GPIO_EXT_ETM_TASK_P2_CFG_REG | GPIO selection register 2 | 0x0160 | R/W |
| GPIO_EXT_ETM_TASK_P4_CFG_REG | GPIO selection register 4 | 0x0168 | R/W |
| GPIO_EXT_ETM_TASK_P5_CFG_REG | GPIO selection register 5 | 0x016C | R/W |
| **Interrupt Registers** | | | |
| GPIO_EXT_INT_RAW_REG | GPIO_EXT interrupt raw register | 0x01D0 | RO/WTC/SS |
| GPIO_EXT_INT_ST_REG | GPIO_EXT interrupt masked register | 0x01D4 | RO |
| GPIO_EXT_INT_ENA_REG | GPIO_EXT interrupt enable register | 0x01D8 | R/W |
| GPIO_EXT_INT_CLR_REG | GPIO_EXT interrupt clear register | 0x01DC | WT |
| **Version Register** | | | |
| GPIO_EXT_VERSION_REG | Version control register | 0x01FC | R/W |

### 7.18.4 LP GPIO Matrix Register Summary

The addresses in this section are relative to LP GPIO matrix base address provided in Table 5.3-2 in Chapter 5 *System and Memory*.

The abbreviations given in Column **Access** are explained in Section *Access Types for Registers*.

| Name | Description | Address | Access |
|---|---|---|---|
| **Configuration Registers** | | | |
| LP_GPIO_OUT_REG | LP_GPIO output register | 0x0004 | R/W/WTC |
| LP_GPIO_OUT_W1TS_REG | LP_GPIO output set register | 0x0008 | WT |
| LP_GPIO_OUT_W1TC_REG | LP_GPIO output clear register | 0x000C | WT |
| LP_GPIO_ENABLE_REG | LP_GPIO output enable register | 0x0010 | R/W/WTC |
| LP_GPIO_ENABLE_W1TS_REG | LP_GPIO output enable set register | 0x0014 | WT |
| LP_GPIO_ENABLE_W1TC_REG | LP_GPIO output enable clear register | 0x0018 | WT |
| LP_GPIO_IN_REG | LP_GPIO input register | 0x001C | RO |
| **Interrupt Status Registers** | | | |
| 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 |
| LP_GPIO_STATUS_NEXT_REG | LP_GPIO interrupt source register | 0x002C | RO |

| Name | Description | Address | Access |
|---|---|---|---|
| **Pin Configuration Registers** | | | |
| LP_GPIO_PIN0_REG | LP_GPIO0 configuration register | 0x0030 | varies |
| LP_GPIO_PIN1_REG | LP_GPIO1 configuration register | 0x0034 | varies |
| LP_GPIO_PIN2_REG | LP_GPIO2 configuration register | 0x0038 | varies |
| LP_GPIO_PIN3_REG | LP_GPIO3 configuration register | 0x003C | varies |
| LP_GPIO_PIN4_REG | LP_GPIO4 configuration register | 0x0040 | varies |
| LP_GPIO_PIN5_REG | LP_GPIO5 configuration register | 0x0044 | varies |
| LP_GPIO_PIN6_REG | LP_GPIO6 configuration register | 0x0048 | varies |
| **Output Configuration Registers** | | | |
| LP_GPIO_FUNC0_OUT_SEL_CFG_REG | Configuration register for LP_GPIO0 output | 0x02B0 | R/W |
| LP_GPIO_FUNC1_OUT_SEL_CFG_REG | Configuration register for LP_GPIO1 output | 0x02B4 | R/W |
| LP_GPIO_FUNC2_OUT_SEL_CFG_REG | Configuration register for LP_GPIO2 output | 0x02B8 | R/W |
| LP_GPIO_FUNC3_OUT_SEL_CFG_REG | Configuration register for LP_GPIO3 output | 0x02BC | R/W |
| LP_GPIO_FUNC4_OUT_SEL_CFG_REG | Configuration register for LP_GPIO4 output | 0x02C0 | R/W |
| LP_GPIO_FUNC5_OUT_SEL_CFG_REG | Configuration register for LP_GPIO5 output | 0x02C4 | R/W |
| LP_GPIO_FUNC6_OUT_SEL_CFG_REG | Configuration register for LP_GPIO6 output | 0x02C8 | R/W |
| **Clock Gate Register** | | | |
| LP_GPIO_CLOCK_GATE_REG | LP_GPIO clock gate register | 0x03F8 | R/W |
| **Version Register** | | | |
| LP_GPIO_DATE_REG | Version control register | 0x03FC | R/W |

### 7.18.5 LP IO MUX Register Summary

The addresses in this section are relative to LP IO MUX base address provided in Table 5.3-2 in Chapter 5 *System and Memory*.

The abbreviations given in Column **Access** are explained in Section *Access Types for Registers*.

| Name | Description | Address | Access |
|---|---|---|---|
| **Configuration Registers** | | | |
| LP_IO_MUX_GPIO0_REG | LP IO MUX configuration register for LP_GPIO0 | 0x0000 | R/W |
| LP_IO_MUX_GPIO1_REG | LP IO MUX configuration register for LP_GPIO1 | 0x0004 | R/W |
| LP_IO_MUX_GPIO2_REG | LP IO MUX configuration register for LP_GPIO2 | 0x0008 | R/W |
| LP_IO_MUX_GPIO3_REG | LP IO MUX configuration register for LP_GPIO3 | 0x000C | R/W |
| LP_IO_MUX_GPIO4_REG | LP IO MUX configuration register for LP_GPIO4 | 0x0010 | R/W |
| LP_IO_MUX_GPIO5_REG | LP IO MUX configuration register for LP_GPIO5 | 0x0014 | R/W |
| LP_IO_MUX_GPIO6_REG | LP IO MUX configuration register for LP_GPIO6 | 0x0018 | R/W |
| **Version Register** | | | |
| LP_IO_MUX_DATE_REG | Version control register | 0x01FC | R/W |

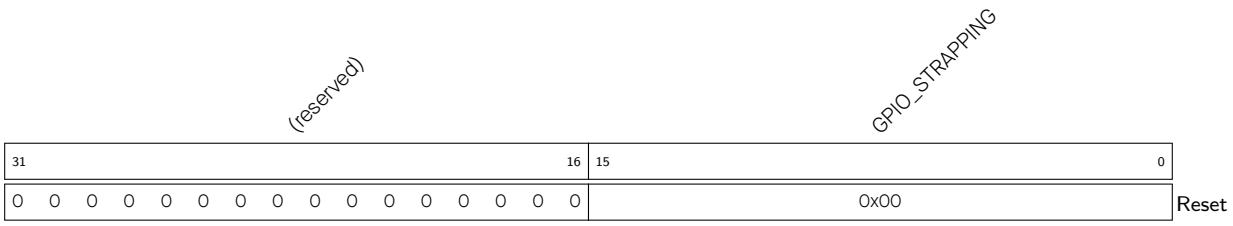
## 7.19 Registers

### 7.19.1 HP GPIO Matrix Registers

The addresses in this section are relative to HP GPIO matrix base address provided in Table 5.3-2 in Chapter 5 *System and Memory*.

For how to program reserved fields, please refer to Section VII .

**Register 7.1. GPIO_STRAP_REG (0x0000)**



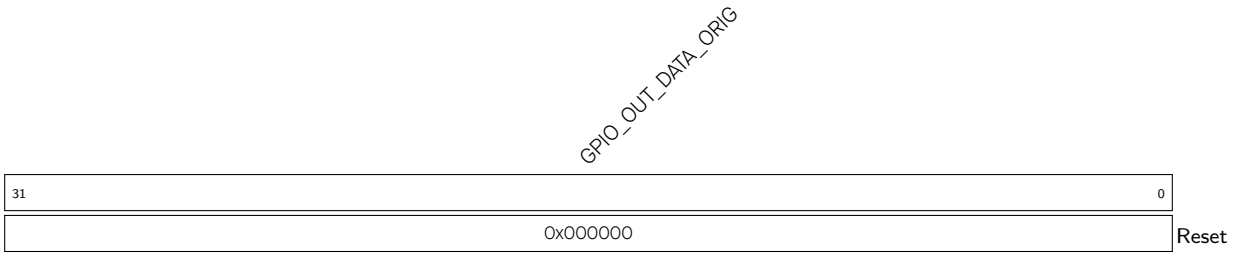

**GPIO_STRAPPING** Represents the values of GPIO strapping pins.

- Bit[0]: invalid
- Bit[1]: invalid
- Bit[2]: GPIO8
- Bit[3]: GPIO9
- Bit[4]: GPIO7
- Bit[5]~Bit[15]: Invalid

For more information about the functions controlled by strapping pins, see Chapter 9 *Chip Boot Control*.

(RO)

**Register 7.2. GPIO_OUT_REG (0x0004)**




**GPIO_OUT_DATA_ORIG** Configures the output value of GPIO0~GPIO13 and GPIO22~GPIO29 in simple GPIO output mode.

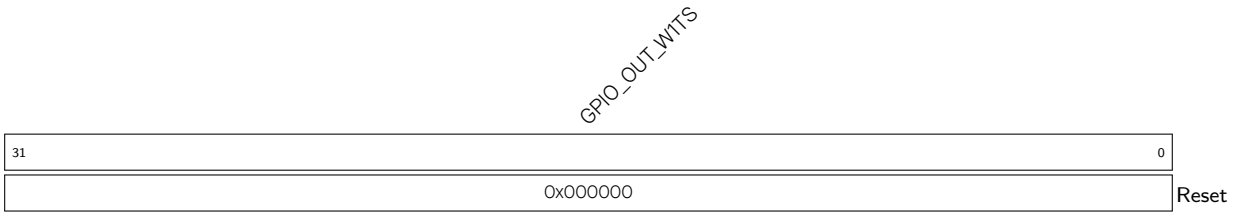
0: Low level

1: High level

The value of bit[0]~bit[13] and bit[22]~bit[29] corresponds to the output value of GPIO0~GPIO13 and GPIO22~GPIO29 respectively.

(R/W/SC/WTC)

Register 7.3. GPIO_OUT_W1TS_REG (0x0008)




**GPIO_OUT_W1TS** Configures whether or not to set the output register GPIO_OUT_REG of GPIO0~GPIO13 and GPIO22~GPIO29.

0: Not set

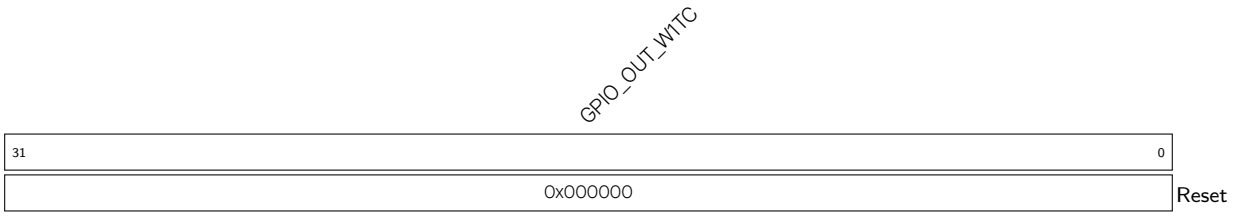
1: The corresponding bit in GPIO_OUT_REG will be set to 1

Bit[0]~bit[13] and bit[22]~bit[29] are corresponding to GPIO0~GPIO13 and GPIO22~GPIO29.

Recommended operation: use this register to set GPIO_OUT_REG.

(WT)

Register 7.4. GPIO_OUT_W1TC_REG (0x000C)




**GPIO_OUT_W1TC** Configures whether or not to clear the output register GPIO_OUT_REG of GPIO0~GPIO13 and GPIO22~GPIO29.

0: Not clear

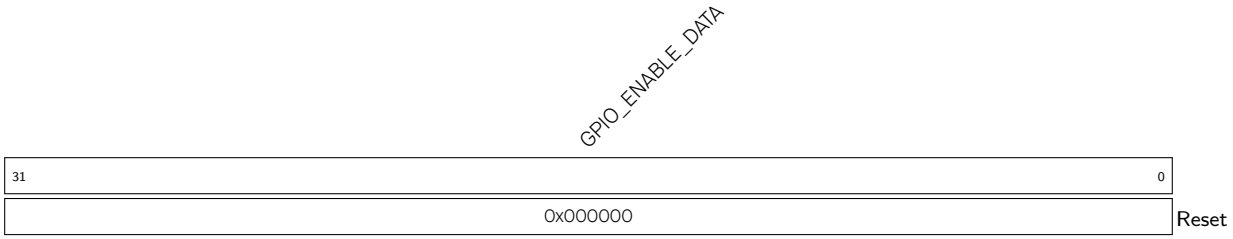
1: The corresponding bit in GPIO_OUT_REG will be cleared.

Bit[0]~bit[13] and bit[22]~bit[29] are corresponding to GPIO0~GPIO13 and GPIO22~GPIO29.

Recommended operation: use this register to clear GPIO_OUT_REG.

(WT)

Register 7.5. GPIO_ENABLE_REG (0x0034)



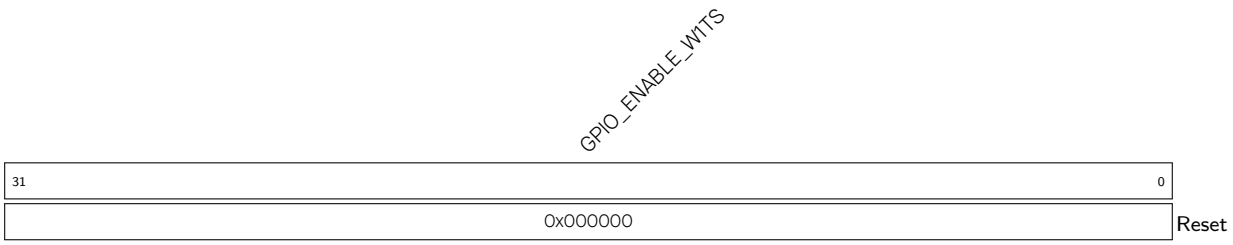

**GPIO_ENABLE_DATA** Configures whether or not to enable the output of GPIO0~GPIO13 and GPIO22~GPIO29.

0: Not enable

1: Enable

Bit[0]~bit[13] and bit[22]~bit[29] are corresponding to GPIO0~GPIO13 and GPIO22~GPIO29. (R/W/WTC)

Register 7.6. GPIO_ENABLE_W1TS_REG (0x0038)




**GPIO_ENABLE_W1TS** Configures whether or not to set the output enable register GPIO_ENABLE_REG of GPIO0~GPIO13 and GPIO22~GPIO29.

0: Not set

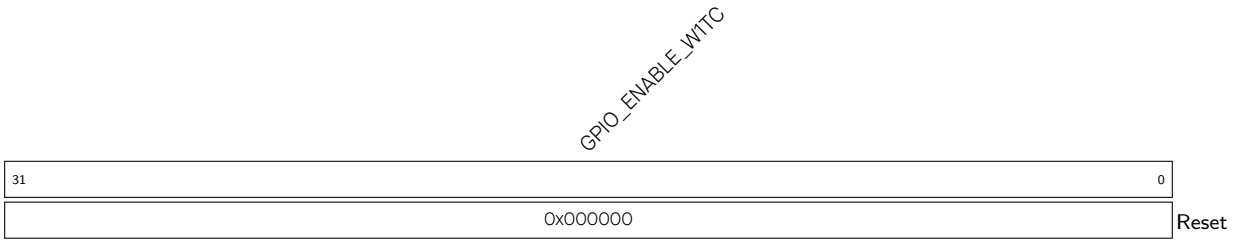
1: The corresponding bit in GPIO_ENABLE_REG will be set to 1

Bit[0]~bit[13] and bit[22]~bit[29] are corresponding to GPIO0~GPIO13 and GPIO22~GPIO29.

Recommended operation: use this register to set GPIO_ENABLE_REG.

(WT)

Register 7.7. GPIO_ENABLE_W1TC_REG (0x003C)




**GPIO_ENABLE_W1TC** Configures whether or not to clear the output enable register GPIO_ENABLE_REG of GPIO0~GPIO13 and GPIO22~GPIO29.

0: Not clear

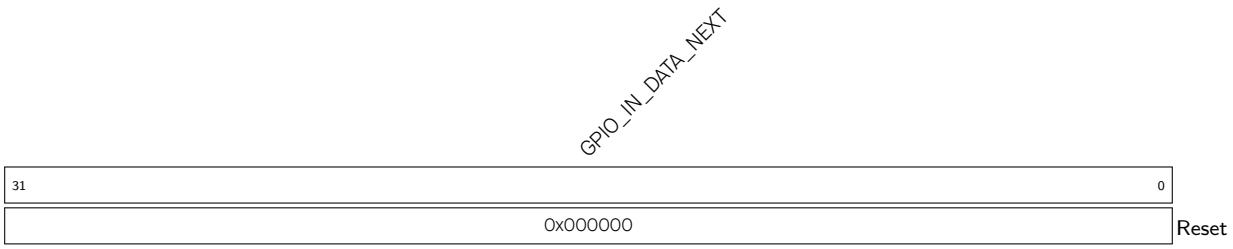
1: The corresponding bit in GPIO_ENABLE_REG will be cleared

Bit[0]~bit[13] and bit[22]~bit[29] are corresponding to GPIO0~GPIO13 and GPIO22~GPIO29.

Recommended operation: use this register to clear GPIO_ENABLE_REG.

(WT)

Register 7.8. GPIO_IN_REG (0x0064)




**GPIO_IN_DATA_NEXT** Represents the input value of GPIO0~GPIO13 and GPIO22~GPIO29.

Each bit represents a pin input value:

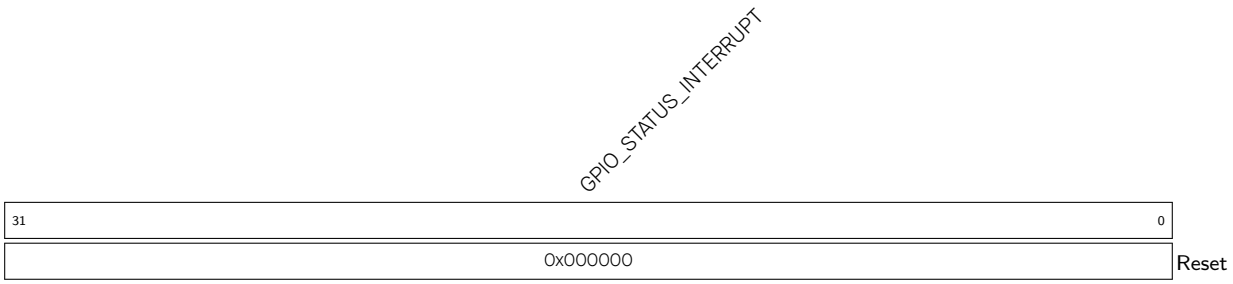
0: Low level

1: High level

Bit[0]~bit[13] and bit[22]~bit[29] are corresponding to GPIO0~GPIO13 and GPIO22~GPIO29.

(RO)

Register 7.9. GPIO_STATUS_REG (0x0074)



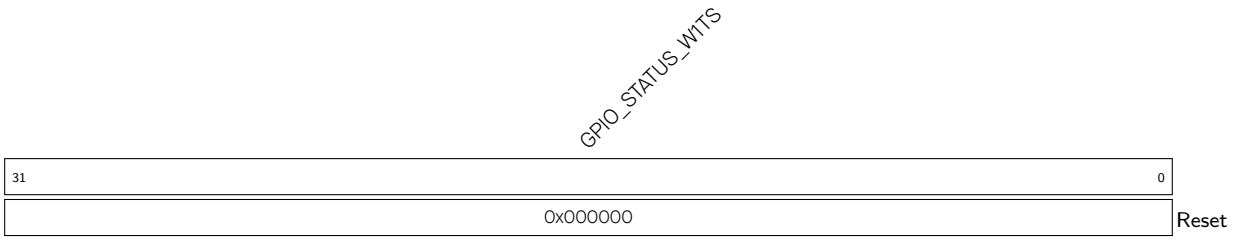

**GPIO_STATUS_INTERRUPT** The interrupt status of GPIO0~GPIO13 and GPIO22~GPIO29, can be configured by the software.

Bit[0]~bit[13] and bit[22]~bit[29] are corresponding to GPIO0~GPIO13 and GPIO22~GPIO29. 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 7.10. GPIO_STATUS_W1TS_REG (0x0078)




**GPIO_STATUS_W1TS** Configures whether or not to set the interrupt status register GPIO_STATUS_INTERRUPT of GPIO0~GPIO13 and GPIO22~GPIO29.

Bit[0]~bit[13] and bit[22]~bit[29] are corresponding to GPIO0~GPIO13 and GPIO22~GPIO29. The value of each bit can be:

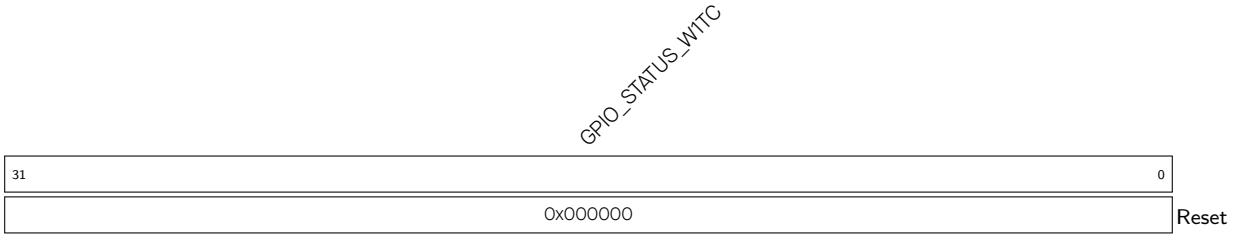
0: Not set

1: The corresponding bit in GPIO_STATUS_INTERRUPT will be set to 1.

Recommended operation: use this register to set GPIO_STATUS_INTERRUPT.

(WT)

Register 7.11. GPIO_STATUS_W1TC_REG (0x007C)




**GPIO_STATUS_W1TC** Configures whether or not to clear the interrupt status register GPIO_STATUS_INTERRUPT of GPIO0~GPIO13 and GPIO22~GPIO29.

Bit[0]~bit[13] and bit[22]~bit[29] are corresponding to GPIO0~GPIO13 and GPIO22~GPIO29.

The value of each bit can be:

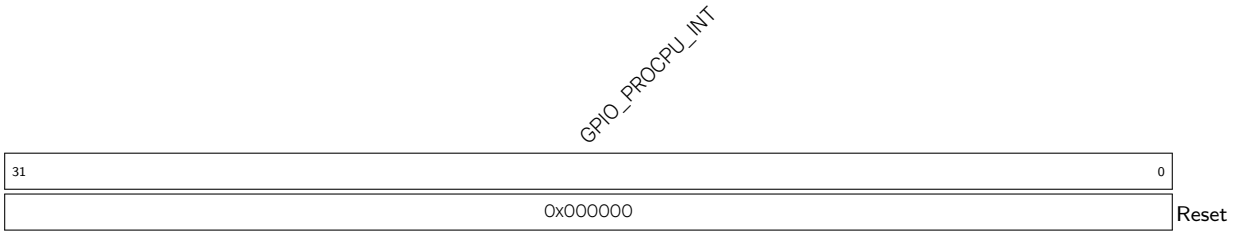
0: Not clear

1: The corresponding bit in GPIO_STATUS_INTERRUPT will be cleared.

Recommended operation: use this register to clear GPIO_STATUS_INTERRUPT.

(WT)

Register 7.12. GPIO_PROCPU_INT_REG (0x00A4)




**GPIO_PROCPU_INT** Represents the CPU interrupt status of GPIO0~GPIO13 and GPIO22~GPIO29.

Bit[0]~bit[13] and bit[22]~bit[29] are corresponding to GPIO0~GPIO13 and GPIO22~GPIO29.

Each bit represents:

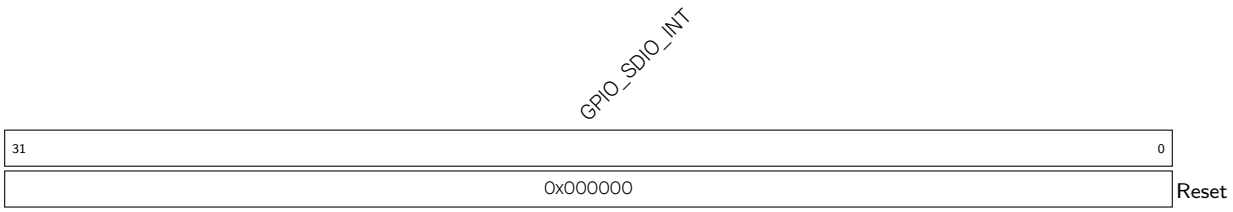
0: Represents CPU interrupt is not enabled, or the GPIO does not generate the interrupt configured by GPIO_PIN*n*_INT_TYPE.

1: Represents the GPIO generates an interrupt configured by GPIO_PIN*n*_INT_TYPE after the CPU interrupt is enabled.

This interrupt status is corresponding to the bit in GPIO_STATUS_REG when assert (high) enable signal (bit[13] of GPIO_PIN*n*_REG).

(RO)

Register 7.13. GPIO_SDIO_INT_REG (0x00A8)




**GPIO_SDIO_INT** Represents the GPIO_SDIO_INT interrupt status of GPIO0~GPIO13 and GPIO22~GPIO29.

Bit[0]~bit[13] and bit[22]~bit[29] are corresponding to GPIO0~GPIO13 and GPIO22~GPIO29.

Each bit represents:

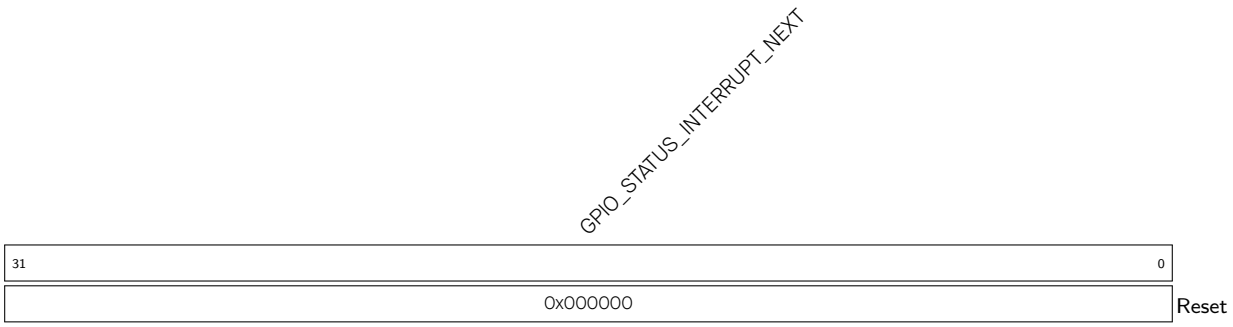
0: Represents GPIO_SDIO_INT interrupt is not enabled, or the GPIO does not generate the interrupt configured by GPIO_PIN*n*_INT_TYPE.

1: Represents the GPIO generates an interrupt configured by GPIO_PIN*n*_INT_TYPE after the GPIO_SDIO_INT interrupt is enabled.

This interrupt status is corresponding to the bit in GPIO_STATUS_REG when assert (high) enable signal (bit[14] of GPIO_PIN*n*_REG).

(RO)

Register 7.14. GPIO_STATUS_NEXT_REG (0x00C4)




**GPIO_STATUS_INTERRUPT_NEXT** Represents the interrupt source signal of GPIO0~GPIO13 and GPIO22~GPIO29.

Bit[0]~bit[13] and bit[22]~bit[29] are corresponding to GPIO0~GPIO13 and GPIO22~GPIO29.

Each bit represents:

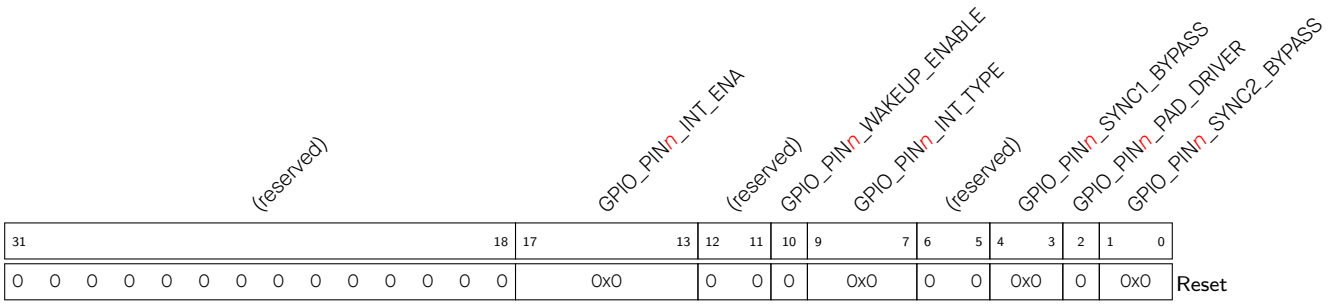
0: The GPIO does not generate the interrupt configured by GPIO_PIN*n*_INT_TYPE.

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 or any edge interrupt.

(RO)

Register 7.15. GPIO_PIN*n*_REG (*n*: 0-13, 22-29) (0x00D4+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 wakeup function.

0: Disable

1: Enable

This function only wakes up the CPU from Light-sleep.

(R/W)

**Continued on the next page...**

Register 7.15. GPIO_PIN*n*_REG (*n*: 0-29) (0x00D4+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.

- Bit[13]: Configures whether or not to enable the GPIO_EXT_REG_INT interrupt:

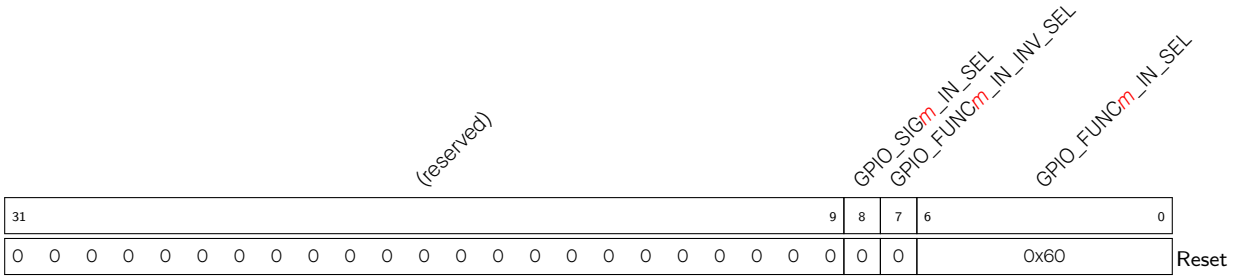
0: Disable

1: Enable

- Bit[14]: Invalid
- Bit[15]: Configures whether or not to enable the GPIO_SDIO_INT interrupt
- Bit[16]~bit[17]: Invalid

(R/W)

Register 7.16. GPIO_FUNC*m*_IN_SEL_CFG_REG (*m*: 6-17) (0x02D4+0x4*(*m*-6))




**GPIO_FUNC*m*_IN_SEL** Configures to select a pin from the 22 GPIO pins (GPIO0~GPIO13, GPIO22~GPIO29) to connect the input signal *m*.

0x0: Select GPIO0

0x1: Select GPIO1

.....

0x1C: Select GPIO28

0x1D: Select GPIO29

Or

0x40: A constantly high input

0x60: A constantly low input

(R/W)

**GPIO_FUNC*m*_IN_INV_SEL** Configures whether or not to invert the input value.

0: Not invert

1: Invert

(R/W)

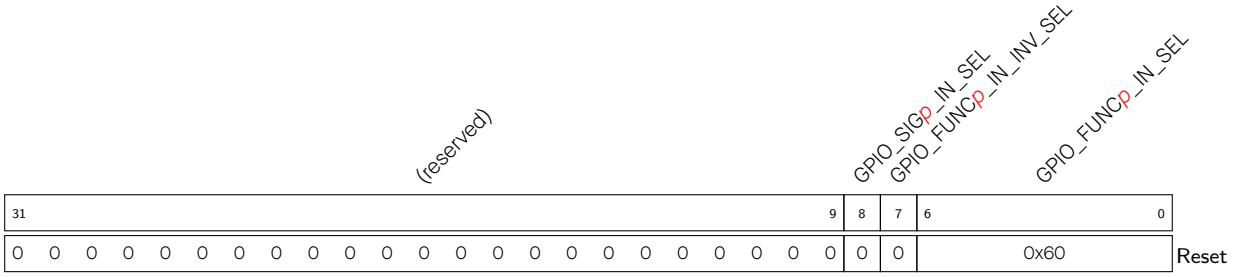
**GPIO_SIG*m*_IN_SEL** Configures whether or not to route signals via HP GPIO matrix.

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 7.17. GPIO_FUNC*p*_IN_SEL_CFG_REG (*p*: 27-35) (0x0340+0x4*(*p*-27))**




**GPIO_FUNC*p*_IN_SEL** Configures to select a pin from the 22 GPIO pins (GPIO0~GPIO13, GPIO22~GPIO29) to connect the input signal *p*.

0x0: Select GPIO0

0x1: Select GPIO1

.....

0x1C: Select GPIO28

0x1D: Select GPIO29

Or

0x40: A constantly high input

0x60: A constantly low input

(R/W)

**GPIO_FUNC*p*_IN_INV_SEL** Configures whether or not to invert the input value.

0: Not invert

1: Invert

(R/W)

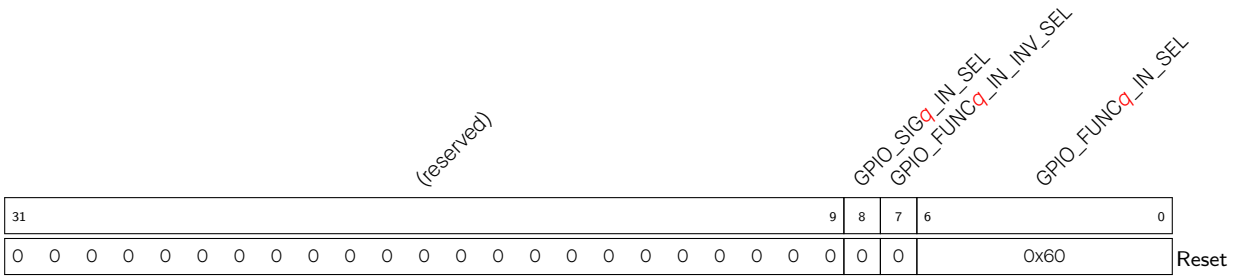
**GPIO_SIG*p*_IN_SEL** Configures whether or not to route signals via HP GPIO matrix.

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 7.18. GPIO_FUNC*q*_IN_SEL_CFG_REG (*q*: 46-47) (0x038C+0x4*(*q*-46))**




**GPIO_FUNC*q*_IN_SEL** Configures to select a pin from the 22 GPIO pins (GPIO0∼GPIO13, GPIO22∼GPIO29) to connect the input signal *q*.

0x0: Select GPIO0

0x1: Select GPIO1

.....

0x1C: Select GPIO28

0x1D: Select GPIO29

Or

0x40: A constantly high input

0x60: A constantly low input

(R/W)

**GPIO_FUNC*q*_IN_INV_SEL** Configures whether or not to invert the input value.

0: Not invert

1: Invert

(R/W)

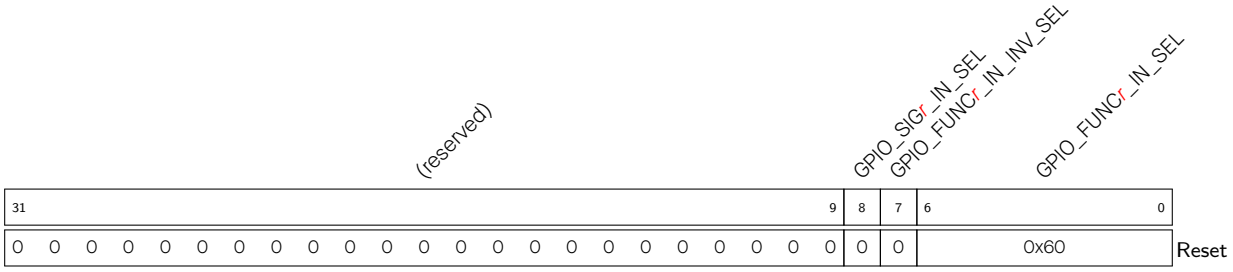
**GPIO_SIG*q*_IN_SEL** Configures whether or not to route signals via HP GPIO matrix.

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 7.19. GPIO_FUNC*r*_IN_SEL_CFG_REG (*r*: 64-69) (0x03D4+0x4*(*r*-64))**




**GPIO_FUNC*r*_IN_SEL** Configures to select a pin from the 22 GPIO pins (GPIO0∼GPIO13, GPIO22∼GPIO29) to connect the input signal *r*.

0x0: Select GPIO0

0x1: Select GPIO1

.....

0x1C: Select GPIO28

0x1D: Select GPIO29

Or

0x40: A constantly high input

0x60: A constantly low input

(R/W)

**GPIO_FUNC*r*_IN_INV_SEL** Configures whether or not to invert the input value.

0: Not invert

1: Invert

(R/W)

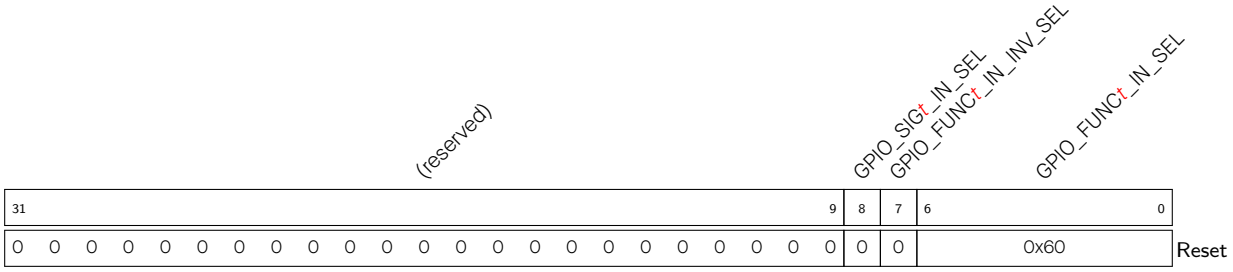
**GPIO_SIG*r*_IN_SEL** Configures whether or not to route signals via HP GPIO matrix.

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 7.20. GPIO_FUNC*t*_IN_SEL_CFG_REG (*t*: 72-74) (0x03F4+0x4*(*t*-72))**




**GPIO_FUNC*t*_IN_SEL** Configures to select a pin from the 22 GPIO pins (GPIO0~GPIO13, GPIO22~GPIO29) to connect the input signal *t*.

0x0: Select GPIO0

0x1: Select GPIO1

.....

0x1C: Select GPIO28

0x1D: Select GPIO29

Or

0x40: A constantly high input

0x60: A constantly low input

(R/W)

**GPIO_FUNC*t*_IN_INV_SEL** Configures whether or not to invert the input value.

0: Not invert

1: Invert

(R/W)

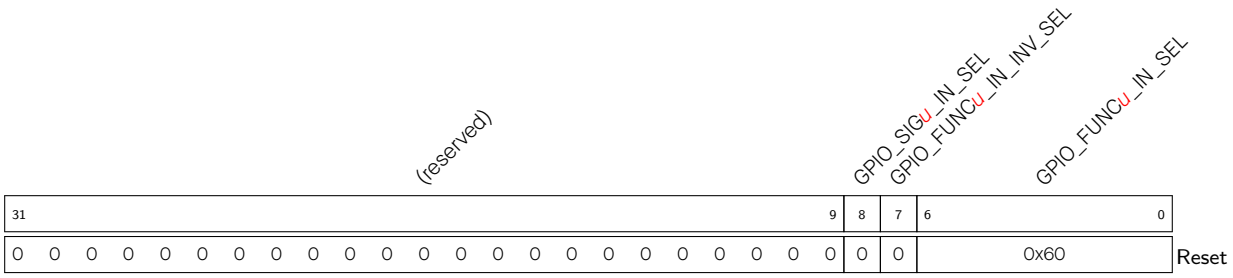
**GPIO_SIG*t*_IN_SEL** Configures whether or not to route signals via HP GPIO matrix.

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 7.21. GPIO_FUNC*u*_IN_SEL_CFG_REG (*u*: 97-100) (0x0458+0x4*(*u*-97))**




**GPIO_FUNC*u*_IN_SEL** Configures to select a pin from the 22 GPIO pins (GPIO0~GPIO13, GPIO22~GPIO29) to connect the input signal *u*.

0x0: Select GPIO0

0x1: Select GPIO1

.....

0x1C: Select GPIO28

0x1D: Select GPIO29

Or

0x40: A constantly high input

0x60: A constantly low input

(R/W)

**GPIO_FUNC*u*_IN_INV_SEL** Configures whether or not to invert the input value.

0: Not invert

1: Invert

(R/W)

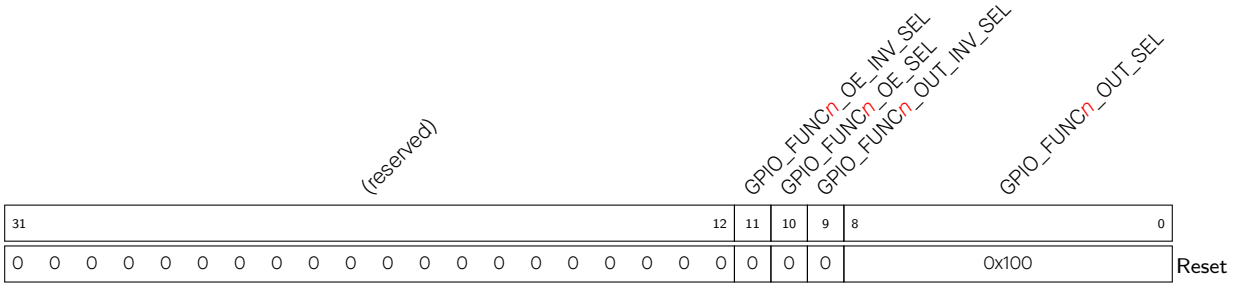
**GPIO_SIG*u*_IN_SEL** Configures whether or not to route signals via HP GPIO matrix.

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 7.22. GPIO_FUNC*n*_OUT_SEL_CFG_REG (*n*: 0-13, 22-29) (0x0AD4+0x4**n*)**




**GPIO_FUNC*n*_OUT_SEL** Configures to select a signal $Y$ (0 <= $Y$ < 256) from peripheral signals to be output from 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_ENABLE_REG are selected as the output value and output enable.

For the detailed signal list, see Table 7.12-1.

(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*] of GPIO_ENABLE_REG.

(R/W)

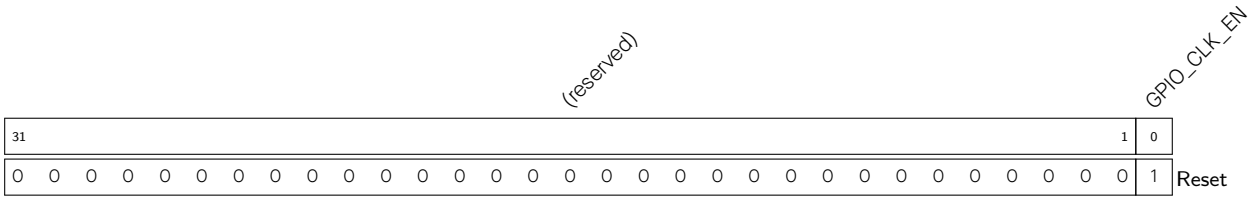
**GPIO_FUNC*n*_OE_INV_SEL** Configures whether or not to invert the output enable signal.

0: Not invert

1: Invert

(R/W)

Register 7.23. GPIO_CLOCK_GATE_REG (0x0DF8)



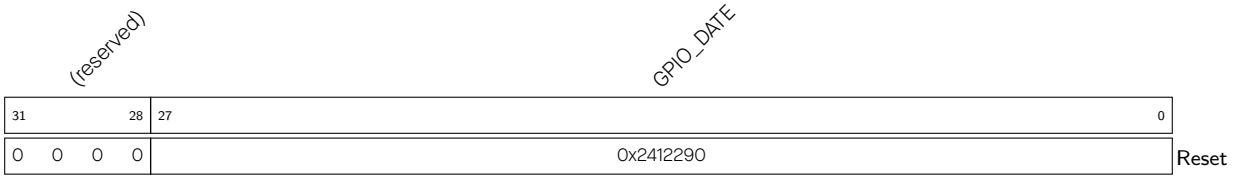

**GPIO_CLK_EN** Configures whether or not to enable clock gate.

0: Not enable

1: Enable, the clock is free running.

(R/W)

Register 7.24. GPIO_DATE_REG (0x0DFC)




**GPIO_DATE** Version control register.

(R/W)

## 7.19.2 HP IO MUX Registers

The addresses in this section are relative to the HP IO MUX base address provided in Table 5.3-2 in Chapter 5 *System and Memory*.

For how to program reserved fields, please refer to Section VII .

Register 7.25. IO_MUX_GPIO*n*_REG (*n*: 0-13, 22-29) (0x0000+0x4**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* in 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)

**IO_MUX_GPIO*n*_FUN_WPD** Configures whether or not to enable pull-down resistor of GPIO*n*.

0: Disable

1: Enable

(R/W)

**Continued on the next page...**

Register 7.25. IO_MUX_GPIO*n*_REG (*n*: 0-13, 22-29) (0x0000+0x4**n*)

Continued from the previous page...

**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 HP IO MUX function for this signal.

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)

**IO_MUX_GPIO*n*_HYS_EN** Configures whether or not to enable the hysteresis function of the pin when IO_MUX_GPIO*n*_HYS_SEL is set to 1.

0: Disable

1: Enable

(R/W)

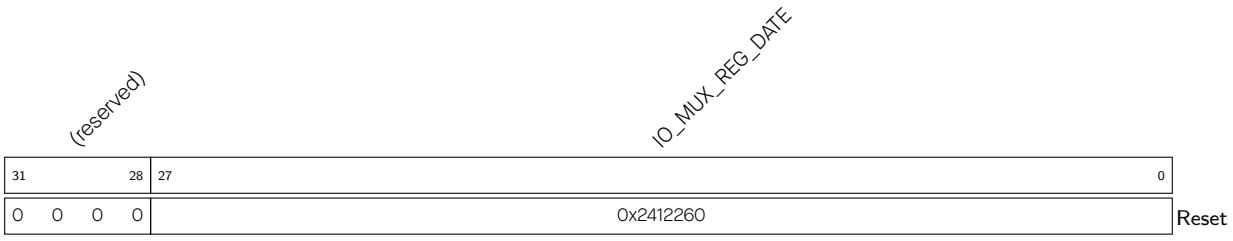
**IO_MUX_GPIO*n*_HYS_SEL** Configures to choose the signal for enabling the hysteresis function for GPIO*n*.

0: Choose the output enable signal of eFuse

1: Choose the output enable signal of IO_MUX_GPIO*n*_HYS_EN

(R/W)

Register 7.26. IO_MUX_DATE_REG (0x01FC)




**IO_MUX_REG_DATE** Version control register.

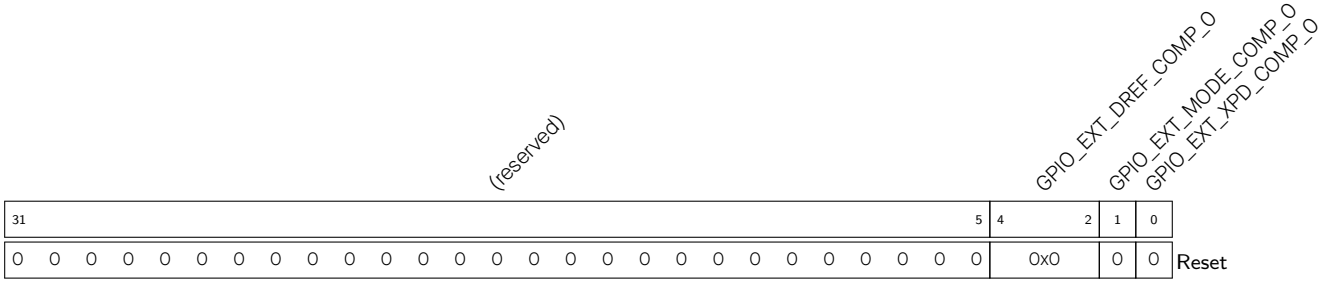
(R/W)

## 7.19.3 GPIO EXT Registers

The addresses in this section are relative to GPIO EXT base address provided in Table 5.3-2 in Chapter 5 *System and Memory*.

For how to program reserved fields, please refer to Section VII .

Register 7.27. GPIO_EXT_PAD_COMP_CONFIG_0_REG (0x0058)




**GPIO_EXT_XPD_COMP_0** Configures whether to enable the function of Analog Voltage Comparator.

0: Disable

1: Enable

(R/W)

**GPIO_EXT_MODE_COMP_0** Configures the reference voltage for Analog Voltage Comparator.

0: Reference voltage is the internal reference voltage, meanwhile GPIO8 PAD can be used as a regular GPIO

1: Reference voltage is the voltage on the GPIO8 PAD

(R/W)

**GPIO_EXT_DREF_COMP_0** Configures the internal reference voltage for Analog Voltage Comparator.

0: Internal reference voltage is 0 * VDDPST1

1: Internal reference voltage is 0.1 * VDDPST1

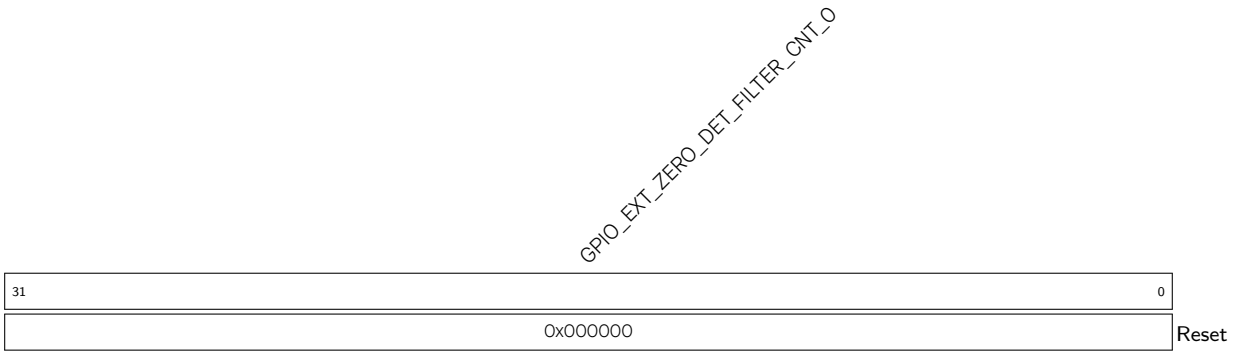
.....

6: Internal reference voltage is 0.6 * VDDPST1

7: Internal reference voltage is 0.7 * VDDPST1

(R/W)

Register 7.28. GPIO_EXT_PAD_COMP_FILTER_0_REG (0x005C)

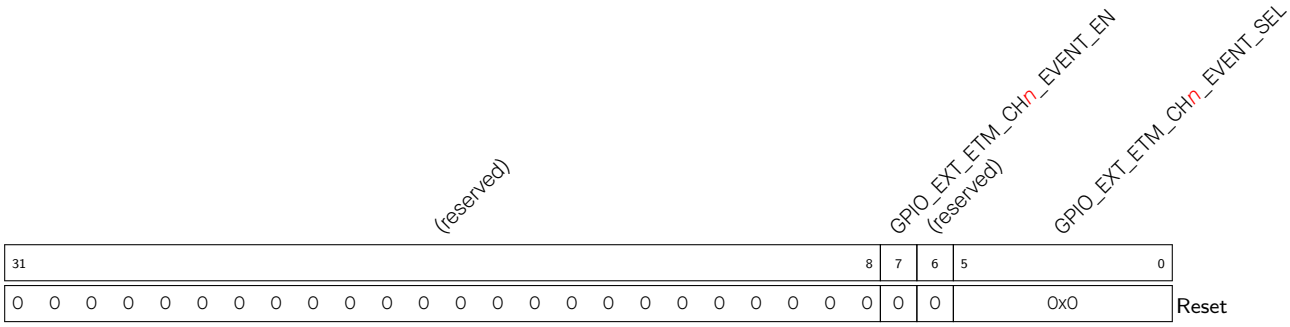



**GPIO_EXT_ZERO_DET_FILTER_CNT_0** Configures the period of masking new interrupt source for Analog Voltage Comparator.

Measurement unit: HP IO MUX operating clock cycle

(R/W)

Register 7.29. GPIO_EXT_ETM_EVENT_CH*n*_CFG_REG (*n*: 0-7) (0x0118+0x4**n*)




**GPIO_EXT_ETM_CH*n*_EVENT_SEL** Configures to select GPIO for ETM event channel.

0: Select GPIO0

1: Select GPIO1

.....

28: Select GPIO28

29: Select GPIO29

30~31: Reserved

(R/W)

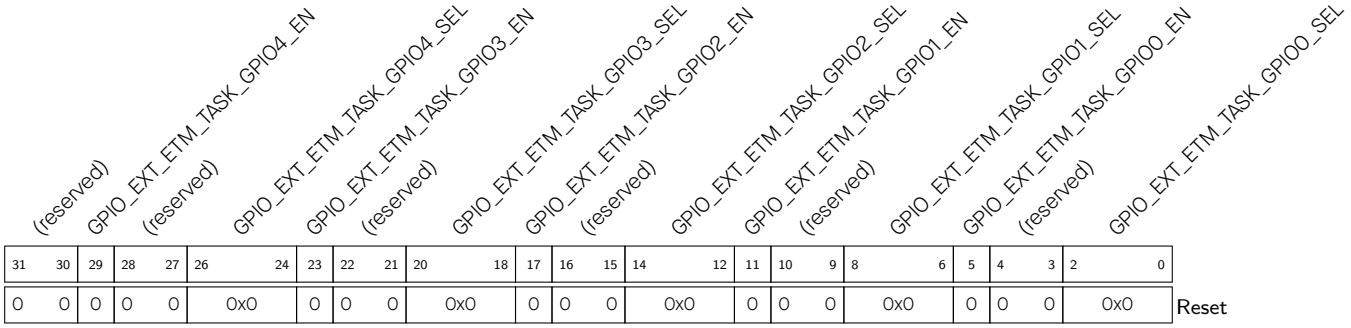
**GPIO_EXT_ETM_CH*n*_EVENT_EN** Configures whether or not to enable ETM event send.

0: Not enable

1: Enable

(R/W)

Register 7.30. GPIO_EXT_ETM_TASK_P0_CFG_REG (0x0158)




**GPIO_EXT_ETM_TASK_GPIO0_SEL** Configures to select an ETM task channel for GPIO0.

0: Select channel 0

1: Select channel 1

.....

7: Select channel 7

(R/W)

**GPIO_EXT_ETM_TASK_GPIO0_EN** Configures whether or not to enable GPIO0 to response ETM task.

0: Not enable

1: Enable

(R/W)

**GPIO_EXT_ETM_TASK_GPIO1_SEL** Configures to select an ETM task channel for GPIO1.

0: Select channel 0

1: Select channel 1

.....

7: Select channel 7

(R/W)

**GPIO_EXT_ETM_TASK_GPIO1_EN** Configures whether or not to enable GPIO1 to response ETM task.

0: Not enable

1: Enable

(R/W)

**GPIO_EXT_ETM_TASK_GPIO2_SEL** Configures to select an ETM task channel for GPIO2.

0: Select channel 0

1: Select channel 1

.....

7: Select channel 7

(R/W)

**GPIO_EXT_ETM_TASK_GPIO2_EN** Configures whether or not to enable GPIO2 to response ETM task.

0: Not enable

1: Enable

(R/W)

**Continued on the next page...**

Register 7.30. GPIO_EXT_ETM_TASK_P0_CFG_REG (0x0158)

Continued from the previous page...

**GPIO_EXT_ETM_TASK_GPIO3_SEL** Configures to select an ETM task channel for GPIO3.

0: Select channel 0

1: Select channel 1

.....

7: Select channel 7

(R/W)

**GPIO_EXT_ETM_TASK_GPIO3_EN** Configures whether or not to enable GPIO3 to response ETM task.

0: Not enable

1: Enable

(R/W)

**GPIO_EXT_ETM_TASK_GPIO4_SEL** Configures to select an ETM task channel for GPIO4.

0: Select channel 0

1: Select channel 1

.....

7: Select channel 7

(R/W)

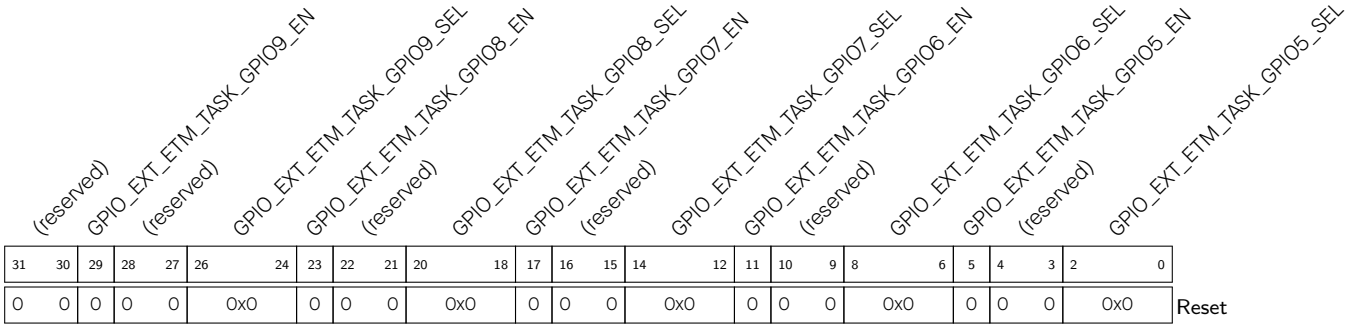
**GPIO_EXT_ETM_TASK_GPIO4_EN** Configures whether or not to enable GPIO4 to response ETM task.

0: Not enable

1: Enable

(R/W)

Register 7.31. GPIO_EXT_ETM_TASK_P1_CFG_REG (0x015C)




**GPIO_EXT_ETM_TASK_GPIO5_SEL** Configures to select an ETM task channel for GPIO5.

0: Select channel 0

1: Select channel 1

.....

7: Select channel 7

(R/W)

**GPIO_EXT_ETM_TASK_GPIO5_EN** Configures whether or not to enable GPIO5 to response ETM task.

0: Not enable

1: Enable

(R/W)

**GPIO_EXT_ETM_TASK_GPIO6_SEL** Configures to select an ETM task channel for GPIO6.

0: Select channel 0

1: Select channel 1

.....

7: Select channel 7

(R/W)

**GPIO_EXT_ETM_TASK_GPIO6_EN** Configures whether or not to enable GPIO6 to response ETM task.

0: Not enable

1: Enable

(R/W)

**GPIO_EXT_ETM_TASK_GPIO7_SEL** Configures to select an ETM task channel for GPIO7.

0: Select channel 0

1: Select channel 1

.....

7: Select channel 7

(R/W)

**GPIO_EXT_ETM_TASK_GPIO7_EN** Configures whether or not to enable GPIO7 to response ETM task.

0: Not enable

1: Enable

(R/W)

**Continued on the next page...**

Register 7.31. GPIO_EXT_ETM_TASK_P1_CFG_REG (0x015C)

Continued from the previous page...

**GPIO_EXT_ETM_TASK_GPIO8_SEL** Configures to select an ETM task channel for GPIO8.

0: Select channel 0

1: Select channel 1

.....

7: Select channel 7

**GPIO_EXT_ETM_TASK_GPIO8_EN** Configures whether or not to enable GPIO8 to response ETM task.

0: Not enable

1: Enable

(R/W)

**GPIO_EXT_ETM_TASK_GPIO9_SEL** Configures to select an ETM task channel for GPIO9.

0: Select channel 0

1: Select channel 1

.....

7: Select channel 7

(R/W)

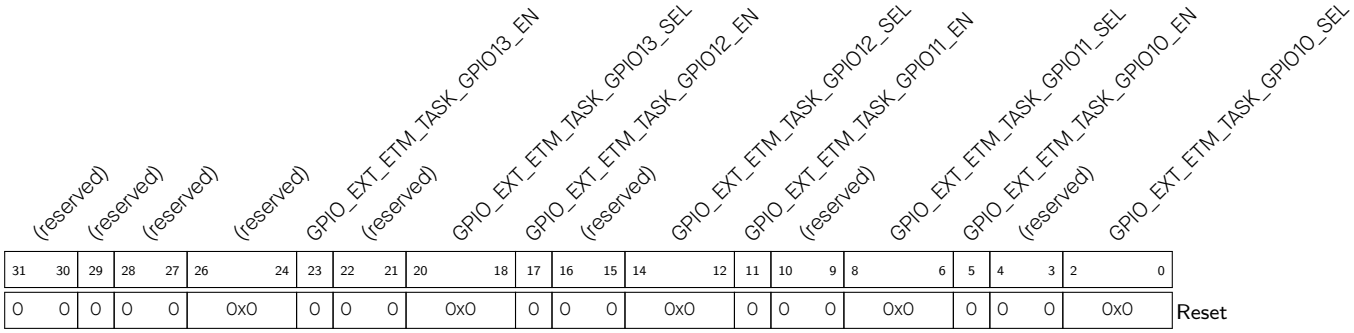
**GPIO_EXT_ETM_TASK_GPIO9_EN** Configures whether or not to enable GPIO9 to response ETM task.

0: Not enable

1: Enable

(R/W)

Register 7.32. GPIO_EXT_ETM_TASK_P2_CFG_REG (0x0160)




**GPIO_EXT_ETM_TASK_GPIO10_SEL** Configures to select an ETM task channel for GPIO10.

0: Select channel 0

1: Select channel 1

.....

7: Select channel 7

(R/W)

**GPIO_EXT_ETM_TASK_GPIO10_EN** Configures whether or not to enable GPIO10 to response ETM task.

0: Not enable

1: Enable

(R/W)

**GPIO_EXT_ETM_TASK_GPIO11_SEL** Configures to select an ETM task channel for GPIO11.

0: Select channel 0

1: Select channel 1

.....

7: Select channel 7

(R/W)

**GPIO_EXT_ETM_TASK_GPIO11_EN** Configures whether or not to enable GPIO11 to response ETM task.

0: Not enable

1: Enable

(R/W)

**GPIO_EXT_ETM_TASK_GPIO12_SEL** Configures to select an ETM task channel for GPIO12.

0: Select channel 0

1: Select channel 1

.....

7: Select channel 7

(R/W)

**GPIO_EXT_ETM_TASK_GPIO12_EN** Configures whether or not to enable GPIO12 to response ETM task.

0: Not enable

1: Enable

(R/W)

Continued on the next page...

Register 7.32. GPIO_EXT_ETM_TASK_P2_CFG_REG (0x0160)

Continued from the previous page...

**GPIO_EXT_ETM_TASK_GPIO13_SEL** Configures to select an ETM task channel for GPIO13.

0: Select channel 0

1: Select channel 1

.....

7: Select channel 7

(R/W)

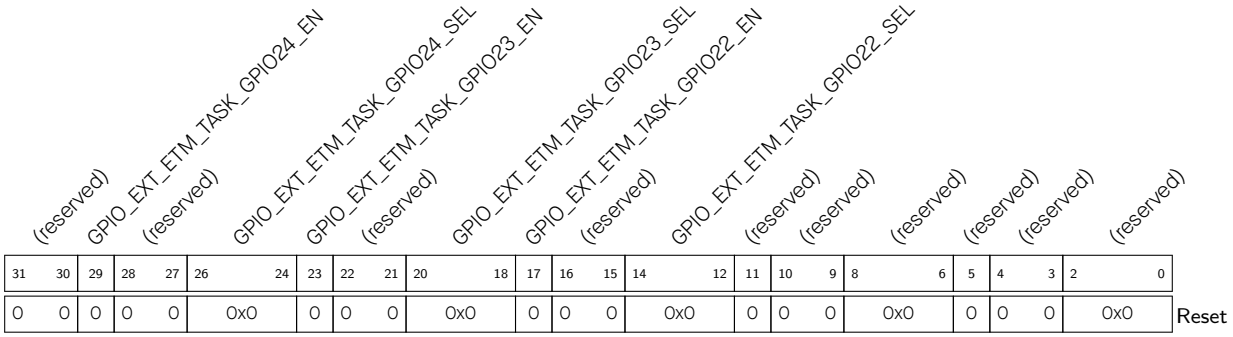
**GPIO_EXT_ETM_TASK_GPIO13_EN** Configures whether or not to enable GPIO13 to response ETM task.

0: Not enable

1: Enable

(R/W)

Register 7.33. GPIO_EXT_ETM_TASK_P4_CFG_REG (0x0168)




**GPIO_EXT_ETM_TASK_GPIO22_SEL** Configures to select an ETM task channel for GPIO22.

0: Select channel 0

1: Select channel 1

.....

7: Select channel 7

(R/W)

**GPIO_EXT_ETM_TASK_GPIO22_EN** Configures whether or not to enable GPIO22 to response ETM task.

0: Not enable

1: Enable

(R/W)

Continued on the next page...

Register 7.33. GPIO_EXT_ETM_TASK_P4_CFG_REG (0x0168)

Continued from the previous page...

**GPIO_EXT_ETM_TASK_GPIO23_SEL** Configures to select an ETM task channel for GPIO23.

0: Select channel 0

1: Select channel 1

.....

7: Select channel 7

(R/W)

**GPIO_EXT_ETM_TASK_GPIO23_EN** Configures whether or not to enable GPIO23 to response ETM task.

0: Not enable

1: Enable

(R/W)

**GPIO_EXT_ETM_TASK_GPIO24_SEL** Configures to select an ETM task channel for GPIO24.

0: Select channel 0

1: Select channel 1

.....

7: Select channel 7

(R/W)

**GPIO_EXT_ETM_TASK_GPIO24_EN** Configures whether or not to enable GPIO24 to response ETM task.

0: Not enable

1: Enable

(R/W)

Register 7.34. GPIO_EXT_ETM_TASK_P5_CFG_REG (0x016C)




**GPIO_EXT_ETM_TASK_GPIO25_SEL** Configures to select an ETM task channel for GPIO25.

0: Select channel 0

1: Select channel 1

.....

7: Select channel 7

(R/W)

**GPIO_EXT_ETM_TASK_GPIO25_EN** Configures whether or not to enable GPIO25 to response ETM task.

0: Not enable

1: Enable

(R/W)

**GPIO_EXT_ETM_TASK_GPIO26_SEL** Configures to select an ETM task channel for GPIO26.

0: Select channel 0

1: Select channel 1

.....

7: Select channel 7

(R/W)

**GPIO_EXT_ETM_TASK_GPIO26_EN** Configures whether or not to enable GPIO26 to response ETM task.

0: Not enable

1: Enable

(R/W)

**GPIO_EXT_ETM_TASK_GPIO27_SEL** Configures to select an ETM task channel for GPIO27.

0: Select channel 0

1: Select channel 1

.....

7: Select channel 7

(R/W)

**GPIO_EXT_ETM_TASK_GPIO27_EN** Configures whether or not to enable GPIO27 to response ETM task.

0: Not enable

1: Enable

(R/W)

Continued on the next page...

Register 7.34. GPIO_EXT_ETM_TASK_P5_CFG_REG (0x016C)

Continued from the previous page...

**GPIO_EXT_ETM_TASK_GPIO28_SEL** Configures to select an ETM task channel for GPIO28.

0: Select channel 0

1: Select channel 1

.....

7: Select channel 7

(R/W)

**GPIO_EXT_ETM_TASK_GPIO28_EN** Configures whether or not to enable GPIO28 to response ETM task.

0: Not enable

1: Enable

(R/W)

**GPIO_EXT_ETM_TASK_GPIO29_SEL** Configures to select an ETM task channel for GPIO29.

0: Select channel 0

1: Select channel 1

.....

7: Select channel 7

(R/W)

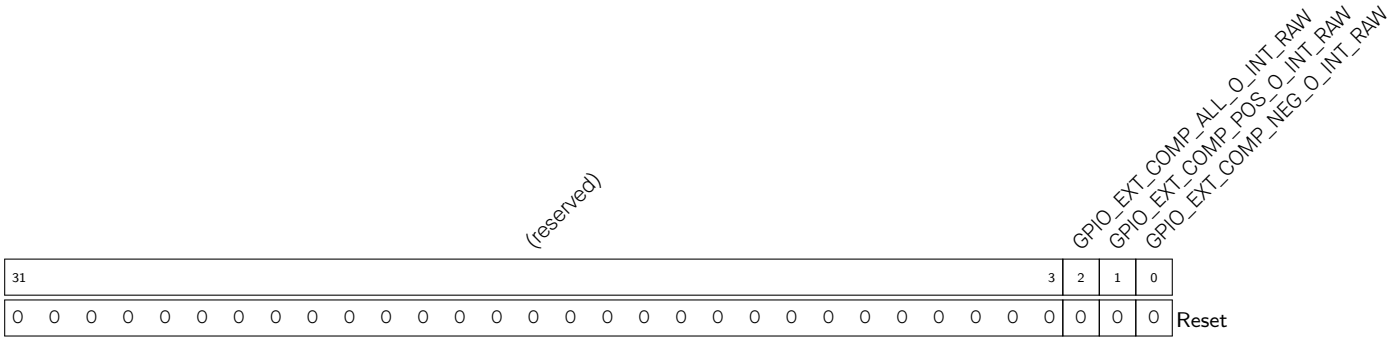
**GPIO_EXT_ETM_TASK_GPIO29_EN** Configures whether or not to enable GPIO28 to response ETM task.

0: Not enable

1: Enable

(R/W)

Register 7.35. GPIO_EXT_INT_RAW_REG (0x01D0)

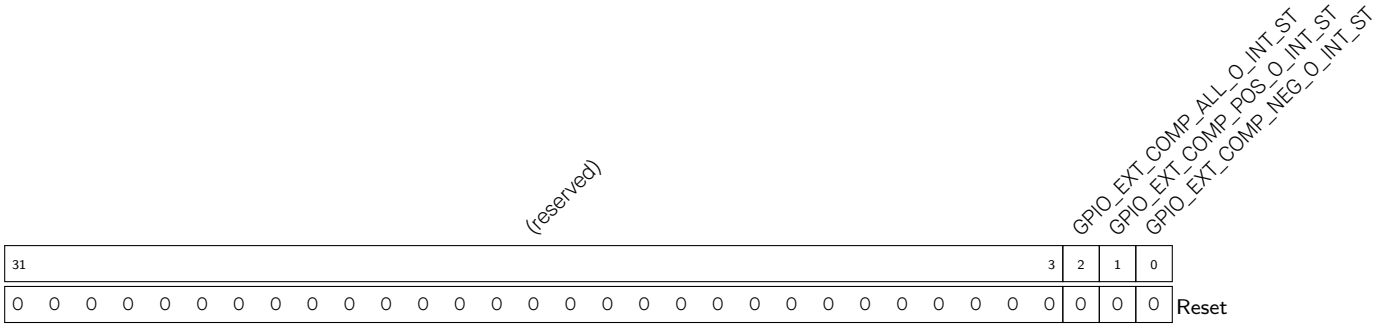



**GPIO_EXT_COMP_NEG_0_INT_RAW** The raw interrupt status of GPIO_EXT_COMP_NEG_0_INT. (RO/WTC/SS)

**GPIO_EXT_COMP_POS_0_INT_RAW** The raw interrupt status of GPIO_EXT_COMP_POS_0_INT. (RO/WTC/SS)

**GPIO_EXT_COMP_ALL_0_INT_RAW** The raw interrupt status of GPIO_EXT_COMP_ALL_0_INT. (RO/WTC/SS)

Register 7.36. GPIO_EXT_INT_ST_REG (0x01D4)

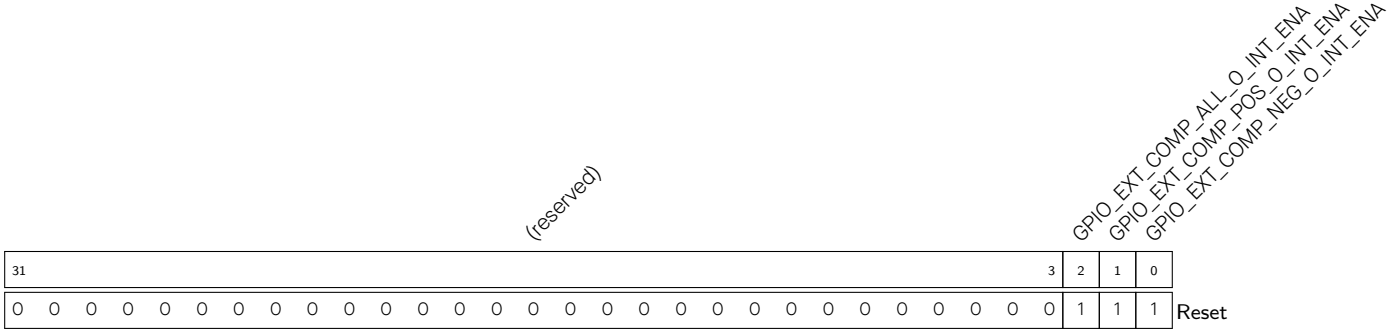



**GPIO_EXT_COMP_NEG_0_INT_ST** The interrupt status of GPIO_EXT_COMP_NEG_0_INT. (RO)

**GPIO_EXT_COMP_POS_0_INT_ST** The interrupt status of GPIO_EXT_COMP_POS_0_INT. (RO)

**GPIO_EXT_COMP_ALL_0_INT_ST** The interrupt status of GPIO_EXT_COMP_ALL_0_INT. (RO)

Register 7.37. GPIO_EXT_INT_ENA_REG (0x01D8)

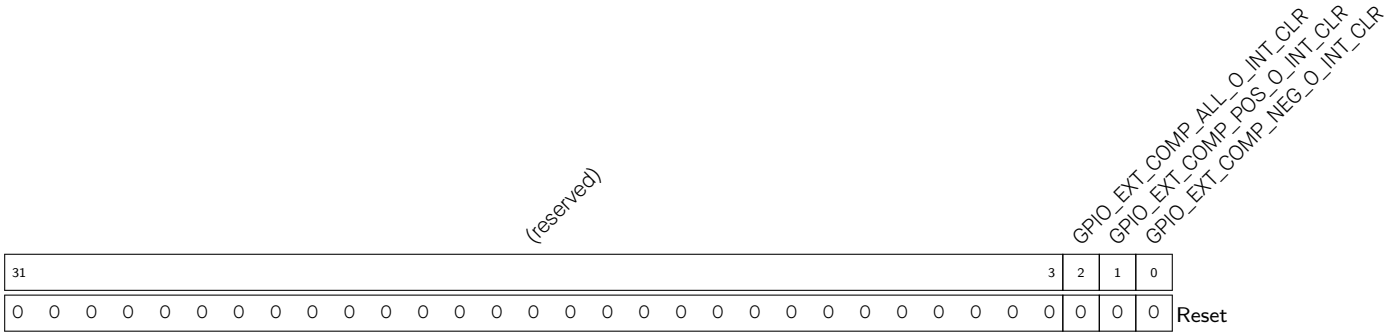



**GPIO_EXT_COMP_NEG_0_INT_ENA** Write 1 to enable GPIO_EXT_COMP_NEG_0_INT. (R/W)

**GPIO_EXT_COMP_POS_0_INT_ENA** Write 1 to enable GPIO_EXT_COMP_POS_0_INT. (R/W)

**GPIO_EXT_COMP_ALL_0_INT_ENA** Write 1 to enable GPIO_EXT_COMP_ALL_0_INT. (R/W)

Register 7.38. GPIO_EXT_INT_CLR_REG (0x01DC)

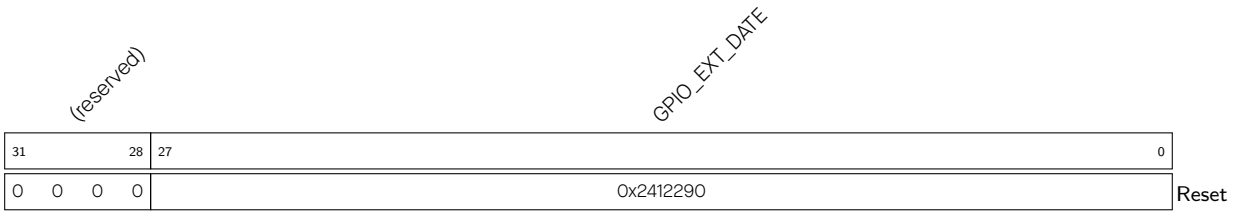



**GPIO_EXT_COMP_NEG_0_INT_CLR** Write 1 to clear GPIO_EXT_COMP_NEG_0_INT. (WT)

**GPIO_EXT_COMP_POS_0_INT_CLR** Write 1 to clear GPIO_EXT_COMP_POS_0_INT. (WT)

**GPIO_EXT_COMP_ALL_0_INT_CLR** Write 1 to clear GPIO_EXT_COMP_ALL_0_INT. (WT)

Register 7.39. GPIO_EXT_VERSION_REG (0x01FC)




**GPIO_EXT_DATE** Version control register.

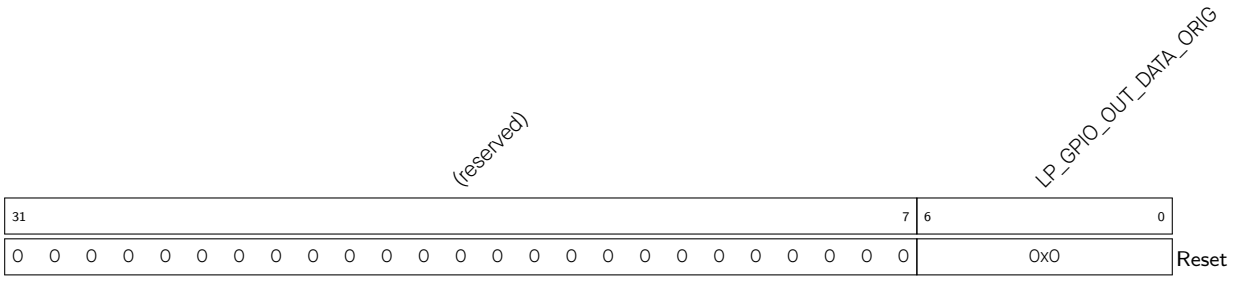
(R/W)

## 7.19.4 LP GPIO Matrix Registers

The addresses in this section are relative to LP GPIO matrix base address provided in Table 5.3-2 in Chapter 5 *System and Memory*.

For how to program reserved fields, please refer to Section VII .

Register 7.40. LP_GPIO_OUT_REG (0x0004)




**LP_GPIO_OUT_DATA_ORIG** Configures the output of GPIO0∼GPIO6.

The value of each bit can be:

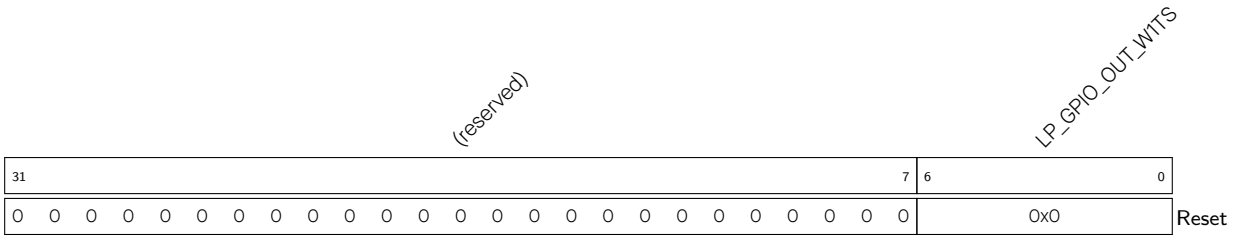
0: Low level

1: High level

Bit[0]∼bit[6] are corresponding to GPIO0∼GPIO6.

(R/W/WTC)

Register 7.41. LP_GPIO_OUT_W1TS_REG (0x0008)




**LP_GPIO_OUT_W1TS** Configures whether or not to enable the output register LP_GPIO_OUT_REG of GPIO0∼GPIO6.

The value of each bit can be:

0: Not set

1: The corresponding bit in LP_GPIO_OUT_REG will be set to 1

Bit[0]∼bit[6] are corresponding to GPIO0∼GPIO6.

Recommended operation: use this register to set LP_GPIO_OUT_REG.

(WT)

Register 7.42. LP_GPIO_OUT_W1TC_REG (0x000C)




**LP_GPIO_OUT_W1TC** Configures whether or not to clear the output register LP_GPIO_OUT_REG of GPIO0∼GPIO6.

The value of each bit can be:

0: Not clear

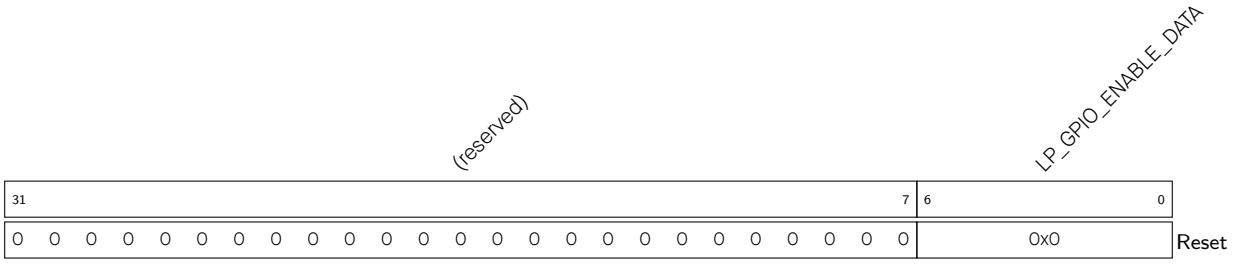
1: The corresponding bit in LP_GPIO_OUT_REG will be cleared.

Bit[0]∼bit[6] are corresponding to GPIO0∼GPIO6.

Recommended operation: use this register to clear LP_GPIO_OUT_REG.

(WT)

Register 7.43. LP_GPIO_ENABLE_REG (0x0010)




**LP_GPIO_ENABLE_DATA** Configures whether or not to enable the output of GPIO0~GPIO6.

The value of each bit can be:

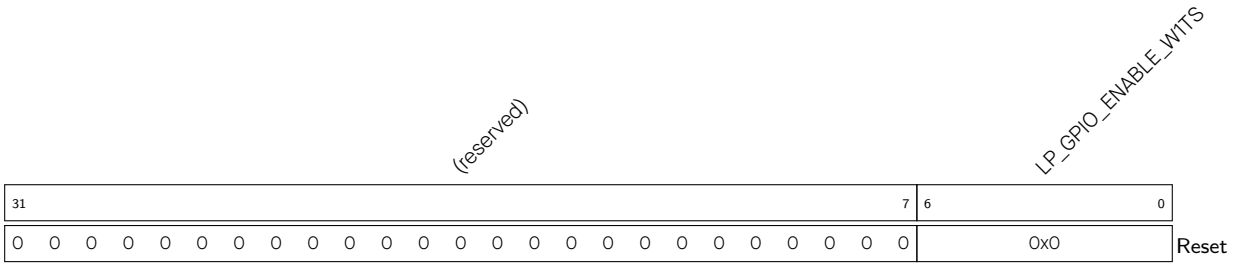
0: Not enable

1: Enable

Bit[0]~bit[6] are corresponding to GPIO0~GPIO6.

(R/W/WTC)

Register 7.44. LP_GPIO_ENABLE_W1TS_REG (0x0014)




**LP_GPIO_ENABLE_W1TS** Configures whether or not to set the output enable register LP_GPIO_ENABLE_REG of GPIO0~GPIO6.

The value of each bit can be:

0: Not set

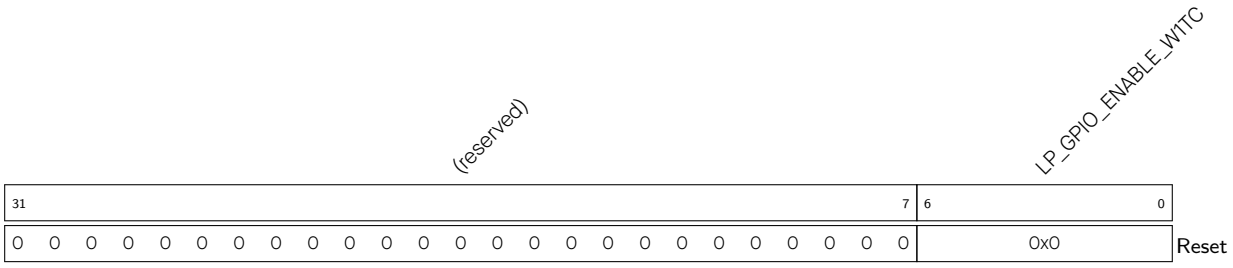
1: The corresponding bit in LP_GPIO_ENABLE_REG will be set to 1

Bit[0]~bit[6] are corresponding to GPIO0~GPIO6.

Recommended operation: use this register to set LP_GPIO_ENABLE_REG.

(WT)

Register 7.45. LP_GPIO_ENABLE_W1TC_REG (0x0018)




**LP_GPIO_ENABLE_W1TC** Configures whether or not to clear the output enable register LP_GPIO_ENABLE_REG of GPIO0~GPIO6.

The value of each bit can be:

0: Not clear

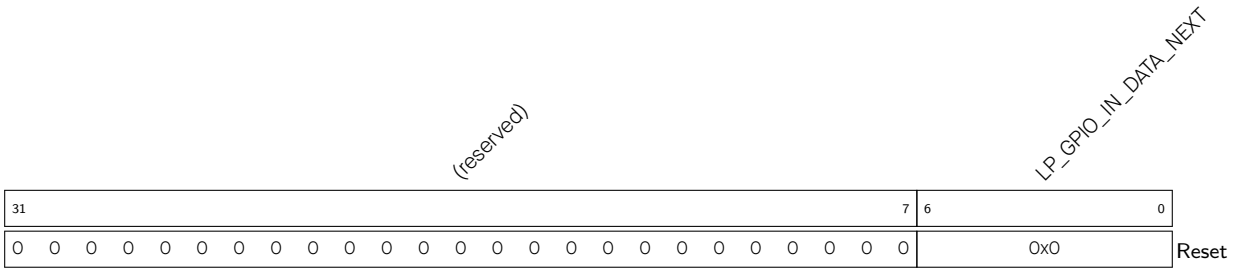
1: The corresponding bit in LP_GPIO_ENABLE_REG will be cleared

Bit[0]~bit[6] are corresponding to GPIO0~GPIO6.

Recommended operation: use this register to clear LP_GPIO_ENABLE_REG.

(WT)

Register 7.46. LP_GPIO_IN_REG (0x001C)




**LP_GPIO_IN_DATA_NEXT** Represents the input value of GPIO0~GPIO6.

Each bit represents a pin input value:

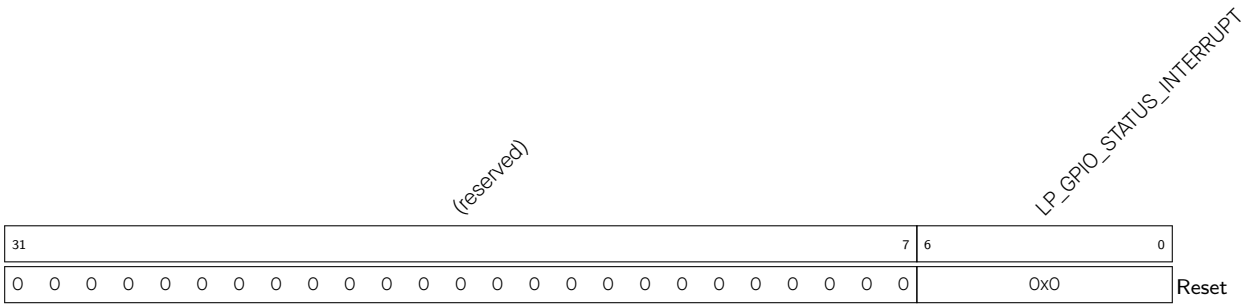
0: Low level input

1: High level input

Bit[0]~bit[6] are corresponding to GPIO0~GPIO6.

(RO)

Register 7.47. LP_GPIO_STATUS_REG (0x0020)




**LP_GPIO_STATUS_INTERRUPT** Configures the interrupt status of GPIO0~GPIO6.

The value of each bit can be:

0: No interrupt

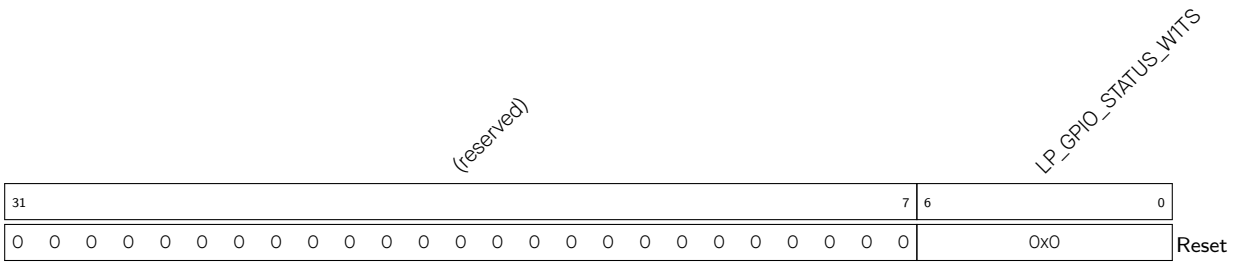
1: Interrupt is triggered

Bit[0]~bit[6] are corresponding to GPIO0~GPIO6.

This field is used together with LP_GPIO_PIN*n*_INT_TYPE in register LP_GPIO_PIN*n*_REG.

(R/W/WTC)

Register 7.48. LP_GPIO_STATUS_W1TS_REG (0x0024)




**LP_GPIO_STATUS_W1TS** Configures whether or not to set the interrupt status register LP_GPIO_STATUS_INT of GPIO0~GPIO6.

The value of each bit can be:

0: Not set

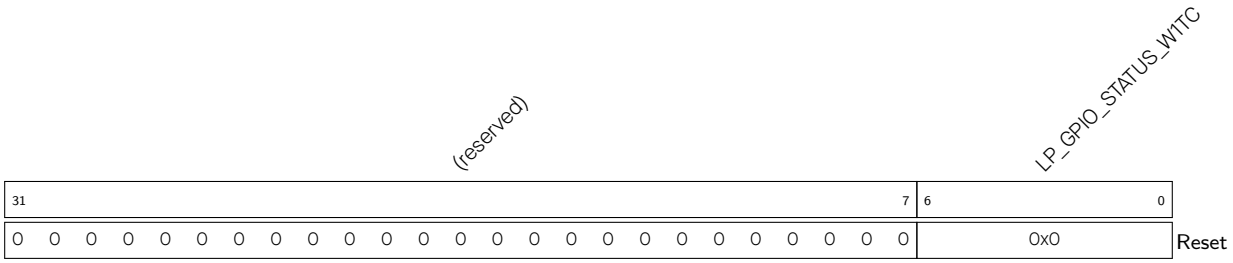
1: The corresponding bit in LP_GPIO_STATUS_INT will be set

Bit[0]~bit[6] are corresponding to GPIO0~GPIO6.

Recommended operation: use this register to set LP_GPIO_STATUS_INT.

(WT)

Register 7.49. LP_GPIO_STATUS_W1TC_REG (0x0028)




**LP_GPIO_STATUS_W1TC** Configures whether or not to clear the interrupt status register LP_GPIO_STATUS_INT of GPIO0~GPIO6.

The value of each bit can be:

0: Not clear

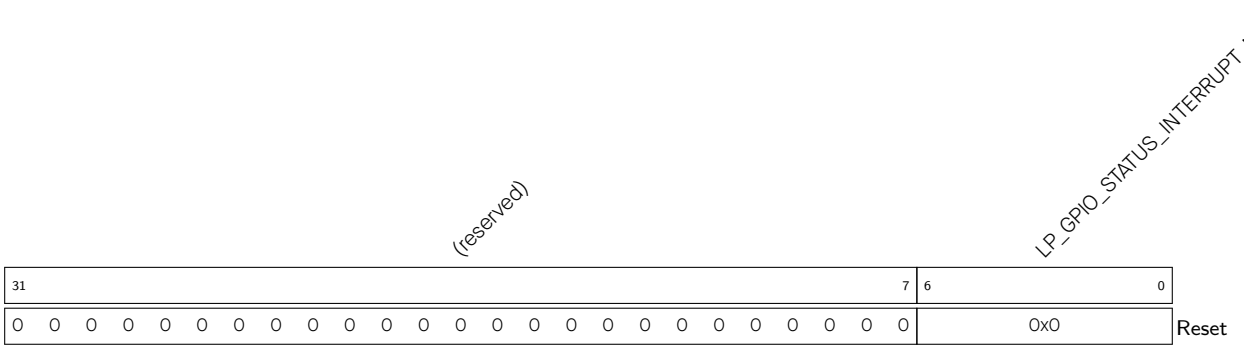
1: The corresponding bit in LP_GPIO_STATUS_INT will be cleared

Bit[0]~bit[6] are corresponding to GPIO0~GPIO6.

Recommended operation: use this register to clear LP_GPIO_STATUS_INT.

(WT)

Register 7.50. LP_GPIO_STATUS_NEXT_REG (0x002C)




**LP_GPIO_STATUS_INTERRUPT_NEXT** Represents the interrupt source status of GPIO0~GPIO6.

Bit[0]~bit[6] are corresponding to GPIO0~GPIO6.

Each bit represents:

0: Interrupt source status is invalid.

1: Interrupt source status is valid.

The interrupt here can be rising-edge triggered, falling-edge triggered, any edge triggered, or level triggered.

(RO)

**Register 7.51. LP_GPIO_PIN*n*_REG (*n*: 0-6) (0x0030+0x4**n*)**

| Field | (reserved) | LP_GPIO_PIN*n*_WAKEUP_ENABLE | LP_GPIO_PIN*n*_INT_TYPE | (reserved) | LP_GPIO_PIN*n*_EDGE_WAKEUP_CLR | LP_GPIO_PIN*n*_SYNC1_BYPASS | LP_GPIO_PIN*n*_PAD_DRIVER | LP_GPIO_PIN*n*_SYNC2_BYPASS |
|---|---|---|---|---|---|---|---|---|
| Bits | 31 ... 11 | 10 | 9 ... 7 | 6 | 5 | 4 ... 3 | 2 | 1 ... 0 |
| Reset | 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | 0 | 0x0 | 0 | 0 | 0x0 | 0 | 0x0 |

**LP_GPIO_PIN*n*_SYNC2_BYPASS** Configures whether or not to synchronize GPIO input data on either edge of LP 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)

**LP_GPIO_PIN*n*_PAD_DRIVER** Configures to select the pin dirve mode of GPIO*n*.
0: Normal output
1: Open drain output
(R/W)

**LP_GPIO_PIN*n*_SYNC1_BYPASS** Configures whether or not to synchronize GPIO input data on either edge of LP 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)

**LP_GPIO_PIN*n*_EDGE_WAKEUP_CLR** Configures whether or not to clear the edge wake-up status of GPIO0~GPIO6.
0: No effect
1: Clear
(WT)

**LP_GPIO_PIN*n*_INT_TYPE** Configures GPIO*n* 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)

**Continued on the next page...**

Register 7.51. LP_GPIO_PIN*n*_REG (*n*: 0-6) (0x0030+0x4**n*)

Continued from the previous page...

**LP_GPIO_PIN*n*_WAKEUP_ENABLE** Configures whether or not to enable GPIO*n* wake-up function.

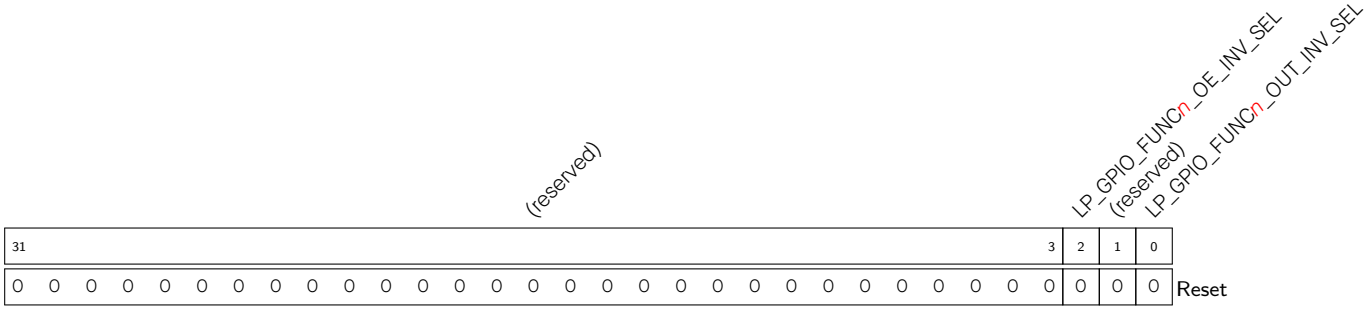
0: Not enable

1: Enable

This function is disabled when PD_LP_PERI is powered off. For more information, see Chapter Low-Power Management.

(R/W)

Register 7.52. LP_GPIO_FUNC*n*_OUT_SEL_CFG_REG (*n*: 0-6) (0x02B0+0x4**n*)




**LP_GPIO_FUNC*n*_OUT_INV_SEL** Configures whether or not to invert the output value.

0: Not invert

1: Invert

(R/W)

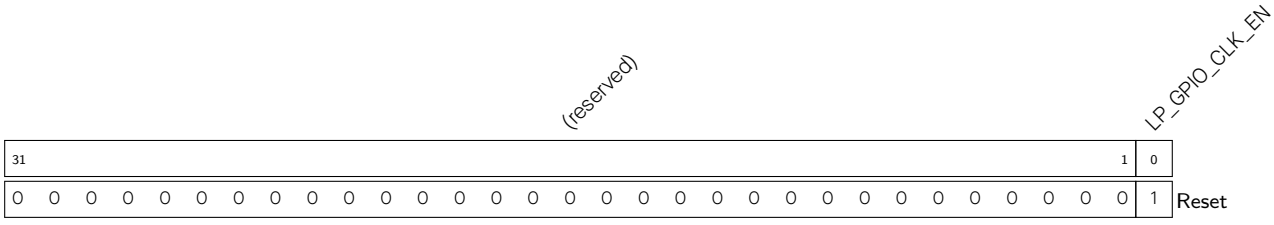
**LP_GPIO_FUNC*n*_OE_INV_SEL** Configures whether or not to invert the output enable signal.

0: Not invert

1: Invert

(R/W)

Register 7.53. LP_GPIO_CLOCK_GATE_REG (0x03F8)



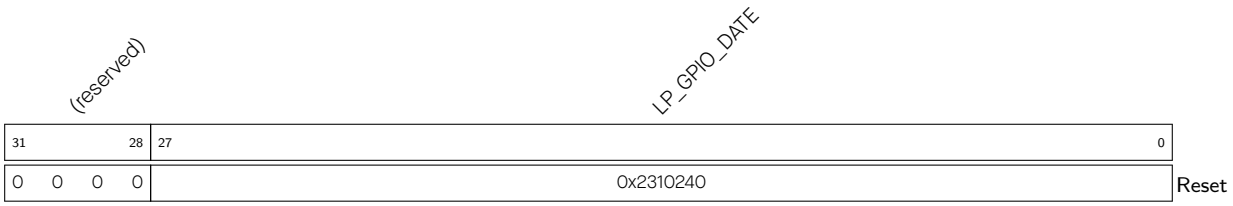

**LP_GPIO_CLK_EN** Configure to enable the GPIO clock gate or not.

0: Not enable

1: Enable

(R/W)

Register 7.54. LP_GPIO_DATE_REG (0x03FC)




**LP_GPIO_DATE** Version control register.

(R/W)

## 7.19.5 LP IO MUX Registers

The addresses in this section are relative to LP IO MUX base address provided in Table 5.3-2 in Chapter 5 *System and Memory*.

For how to program reserved fields, please refer to Section VII .

**Register 7.55. LP_IO_MUX_GPIO*n*_REG (*n*: 0-6) (0x0000+0x4**n*)**




**LP_IO_MUX_GPIO*n*_MCU_OE** Configures whether or not to enable the output of GPIO*n* during sleep mode.

0: Not enable

1: Enable

(R/W)

**LP_IO_MUX_GPIO*n*_SLP_SEL** Configures whether or not to enable the sleep mode for GPIO*n*.

0: Not enable

1: Enable

(R/W)

**LP_IO_MUX_GPIO*n*_MCU_WPD** Configures whether or not to enable the pull-down resistor of GPIO*n* during sleep mode.

0: Not enable

1: Enable

(R/W)

**LP_IO_MUX_GPIO*n*_MCU_WPU** Configures whether or not to enable the pull-up resistor of GPIO*n* during sleep mode.

0: Not enable

1: Enable

(R/W)

**LP_IO_MUX_GPIO*n*_MCU_IE** Configures whether or not to enable the input of GPIO*n* during sleep mode.

0: Not enable

1: Enable

(R/W)

**LP_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 7.55. LP_IO_MUX_GPIO*n*_REG (*n*: 0-6) (0x0000+0x4**n*)

Continued from the previous page...

**LP_IO_MUX_GPIO*n*_FUN_WPD** Configures whether or not to enable the pull-down resistor of GPIO*n* in normal execution mode.
0: Not enable
1: Enable
(R/W)

**LP_IO_MUX_GPIO*n*_FUN_WPU** Configures whether or not to enable the pull-up resistor of GPIO*n* in normal execution mode.
0: Not enable
1: Enable
(R/W)

**LP_IO_MUX_GPIO*n*_FUN_IE** Configures whether or not to enable the input of GPIO*n* in normal execution mode.
0: Not enable
1: Enable
(R/W)

**LP_IO_MUX_GPIO*n*_FUN_DRV** Configures the drive strength of GPIO*n* in normal execution mode.
0: ~5 mA
1: ~10 mA
2: ~20 mA
3: ~40 mA
(R/W)

**LP_IO_MUX_GPIO*n*_MCU_SEL** Configures to select the LP IO MUX function for GPIO*n* in normal execution mode.
0: Select Function 0
1: Select Function 1
.....
(R/W)

**LP_IO_MUX_GPIO*n*_FILTER_EN** Configures whether or not to enable filter for pin input signals.
0: Disable
1: Enable
(R/W)

**LP_IO_MUX_GPIO*n*_HYS_EN** Configures whether or not to enable the hysteresis function of the pin when LP_IO_MUX_GPIO*n*_HYS_SEL is set to 1.
0: Disable
1: Enable
(R/W)

Continued on the next page...

Register 7.55. LP_IO_MUX_GPIO*n*_REG (*n*: 0-6) (0x0000+0x4**n*)

Continued from the previous page...

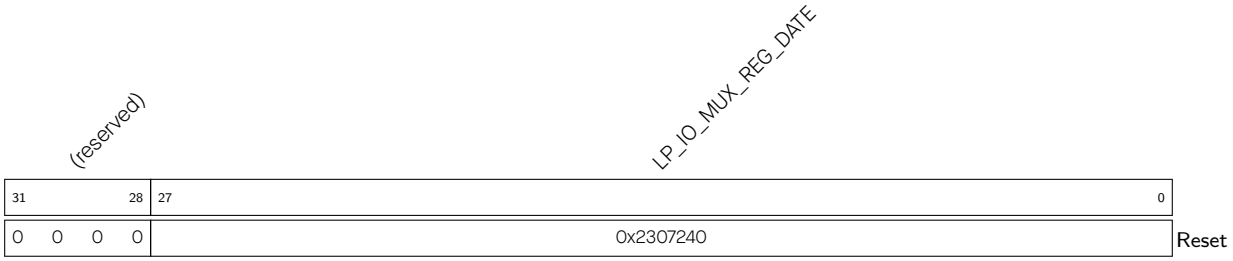
**LP_IO_MUX_GPIO*n*_HYS_SEL** Configures to choose the signal for enabling the hysteresis function for GPIO*n*.

0: Choose the output enable signal of eFuse

1: Choose the output enable signal of LP_IO_MUX_GPIO*n*_HYS_EN

(R/W)

Register 7.56. LP_IO_MUX_DATE_REG (0x01FC)




**LP_IO_MUX_REG_DATE** Version control register.

(R/W)

# Chapter 8

# Reset and Clock

## 8.1 Reset

### 8.1.1 Overview

ESP32-C61 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 8.1-1 shows the scopes of affected subsystems by each type of reset.

### 8.1.2 Architectural Overview

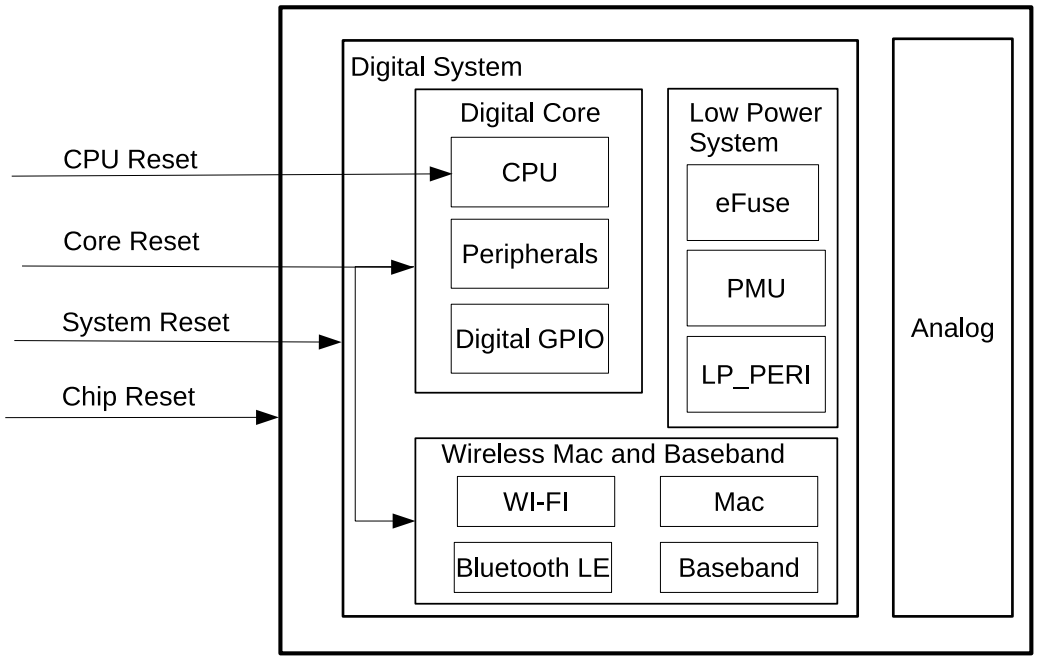



Figure 8.1-1. Reset Types

ESP32-C61's Digital System can be divided into two parts: High Performance System (HP system) that includes Digital Core, Wireless Mac and Baseband, and HP SRAM, and Low Power System (LP system) that only includes some low-power peripherals and LP SRAM. See Figure 8.1-1 for details (note that HP SRAM and LP SRAM would not be reset, so they are not shown in the figure).

### 8.1.3 Features

- Four reset types:

  - CPU Reset: resets CPU core. Once such a reset is released, the instructions from the CPU reset vector (0x40000000) will be executed.
  - Core Reset: resets the whole digital system except LP system, including CPU, peripherals, digital GPIOs, Wireless MAC and Baseband.
  - System Reset: resets the whole digital system, including LP system.
  - Chip Reset: resets the whole chip, including the digital system and analog system.
- Software reset and hardware reset:
  - Software Reset: triggered via software by configuring the corresponding registers of CPU, see Chapter 12 *Low-Power Management*.
  - Hardware Reset: triggered directly by the hardware.

### 8.1.4 Functional Description

CPU will be reset immediately when any type of reset above occurs. Users can retrieve reset source codes by reading LP_CLKRST_RESET_CAUSE after the reset is released. The current reset source recorded in the LP_CLKRST_RESET_CAUSE register can be cleared by configuring LP_CLKRST_CORE0_RESET_CAUSE_CLR.

Table 8.1-1 lists possible reset sources and the types of reset they trigger. When multiple reset sources are active simultaneously, the reset source recorded in LP_CLKRST_RESET_CAUSE is the most top one among the active reset sources listed in Table 8.1-1. When multiple reset sources are active but not at the same time, LP_CLKRST_RESET_CAUSE records the most recent reset source.

**Table 8.1-1. Reset Source**

| Code | Source | Reset Type | Note |
|---|---|---|---|
| 0x01 | Chip reset[1] | Chip Reset | — |
| 0x12 | Super watchdog reset | System Reset | See Chapter 15 *Watchdog Timers (WDT)* |
| 0x19 | Power glitch reset | System Reset | See Chapter 20 *Power Supply Detector* |
| 0x0F | Brown-out system reset | Chip Reset or System Reset | Triggered by brown-out detector[2] |
| 0x10 | RWDT system reset | System Reset | See Chapter 15 *Watchdog Timers (WDT)* |
| 0x09 | RWDT core reset | Core Reset | See Chapter 15 *Watchdog Timers (WDT)* |
| 0x14 | eFuse reset | Core Reset | Triggered by eFuse CRC error check |
| 0x07 | MWDT0 core reset | Core Reset | See Chapter 15 *Watchdog Timers (WDT)* |
| 0x08 | MWDT1 core reset | Core Reset | See Chapter 15 *Watchdog Timers (WDT)* |
| 0x15 | USB (UART) reset | Core Reset | Triggered when external USB host sends a specific command to the serial interface of USB Serial/JTAG Controller. See Chapter 30 *USB Serial/JTAG Controller* |

**Cont'd on next page**

Table 8.1-1 – cont'd from previous page

| Code | Source | Reset Type | Note |
|---|---|---|---|
| 0x16 | USB (JTAG) reset | Core Reset | Triggered when external USB host sends a specific command to the JTAG interface of USB Serial/JTAG Controller. See Chapter 30 *USB Serial/JTAG Controller* |
| 0x03 | Software system reset | Core Reset | Triggered by configuring LP_AON_HPSYS_SW_RESET |
| 0x0D | RWDT CPU reset | CPU Reset | See Chapter 15 *Watchdog Timers (WDT)* |
| 0x0C | Software CPU reset | CPU Reset | Triggered by configuring LP_AON_CPU_COREO_SW_RESET |
| 0x0B | MWDTO CPU reset | CPU Reset | See Chapter 15 *Watchdog Timers (WDT)* |
| 0x11 | MWDT1 CPU reset | CPU Reset | See Chapter 15 *Watchdog Timers (WDT)* |
| 0x18 | JTAG CPU reset | CPU Reset | Triggered when a "JDB Resetting CPU" instruction is received |
| 0x1A | CPU lockup reset | CPU Reset | Triggered when CPU lockup |
| 0x05 | Deep-sleep reset | Core Reset | See Chapter 12 *Low-Power Management* |

[1] Chip Reset can be triggered by the following sources:

- Triggered by chip power-up.
- Triggered by brown-out detector.

[2] Once brown-out status is detected, the detector will trigger System Reset or Chip Reset, depending on the register configuration. See Chapter 12 *Low-Power Management*.

### 8.1.5 Peripheral Reset

Peripherals can be reset individually by configuring corresponding registers, or globally by core reset, system reset, or chip reset. After releasing peripherals from reset, the reset release status registers can be read to determine the reset status. These registers turning to 1 indicates that peripherals have been released from reset and can operate properly.

The reset registers of ESP32-C61's HP system peripherals are controlled by the Power/Clock/Reset (PCR) module. Please see 8.4.1 *PCR Register Summary* for the reset registers of HP system peripherals and 8.4.2 *LP System Clock Register Summary* for the clock registers of low-power system peripherals.

## 8.2 Clock

### 8.2.1 Overview

ESP32-C61 clocks are mainly sourced from oscillator (OSC), RC, and PLL circuits, 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 8.2-1 shows the system clock structure.

## 8.2.2 Architectural Overview

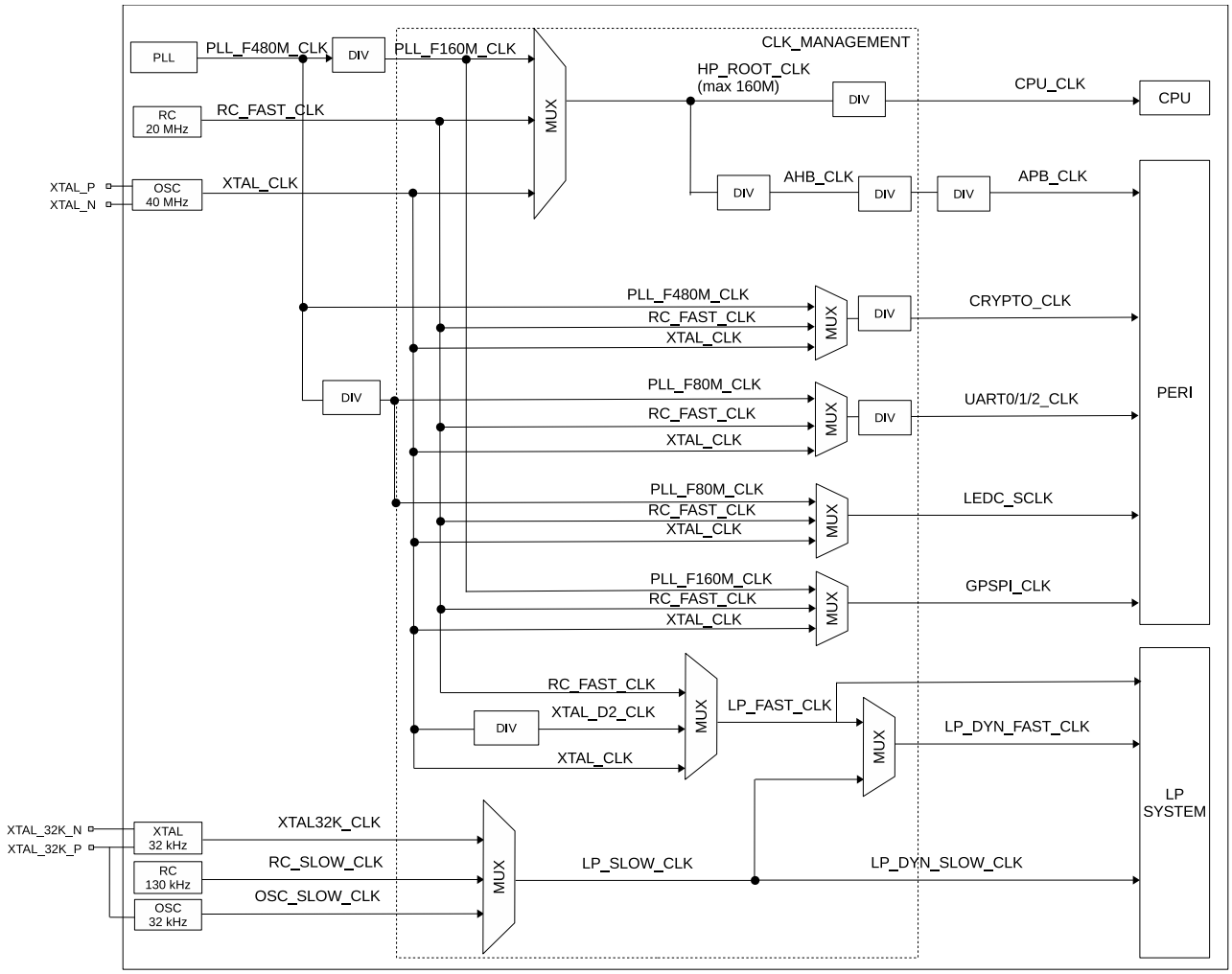



Figure 8.2-1. System Clock

## 8.2.3 Features

ESP32-C61 clocks can be classified into two types depending on their frequencies:

- High-performance (HP) clocks for devices working at a higher frequency, such as CPU and digital peripherals
  - PLL_F480M_CLK (480 MHz): internal PLL clock. Its reference clock is XTAL_CLK
  - XTAL_CLK (40 MHz): external crystal clock
- Low-power (LP) clocks for low-power system and some peripherals working in low-power mode
  - XTAL32K_CLK (32 kHz): external crystal clock
  - RC_FAST_CLK (20 MHz by default): internal fast RC oscillator with adjustable frequency
  - RC_SLOW_CLK (130 kHz by default): internal slow RC oscillator with adjustable frequency
  - OSC_SLOW_CLK (32 kHz by default): external slow clock input through XTAL_32K_P. After configuring this GPIO, also configure the Hold function (see Chapter 7 *GPIO Matrix and IO MUX* > 7.9

*Pin Hold Feature*)

## 8.2.4 Functional Description

### 8.2.4.1 HP System Clock

As Figure 8.2-1 shows, CPU_CLK is the master clock for CPU and its frequency can be as high as 160 MHz. Alternatively, CPU can run at lower frequencies, such as at 2 MHz, to achieve lower power consumption. CPU_CLK shares the same clock sources with AHB_CLK and APB_CLK. Users can select from XTAL_CLK, PLL_160M_CLK, or RC_FAST_CLK as the clock source of CPU_CLK by configuring PCR_SOC_CLK_SEL. For details, see Table 8.2-1 and Table 8.2-2. By default, the CPU clock is sourced from XTAL_CLK with a division factor of 1.

**Table 8.2-1. CPU_CLK Clock Source**

| PCR_SOC_CLK_SEL | CPU Clock Source |
|---|---|
| 0 | XTAL_CLK |
| 1 | RC_FAST_CLK |
| 2 | PLL_F160M_CLK |

**Table 8.2-2. Frequency of CPU_CLK, AHB_CLK and HP_ROOT_CLK**

| Clock | Source | Frequency |
|---|---|---|
| HP_ROOT_CLK | PLL_F160M_CLK | 160 MHz |
| | XTAL_CLK | 40 MHz |
| | RC_FAST_CLK | 20 MHz |
| CPU_CLK[1] | HP_ROOT_CLK | $f_{HP\_ROOT\_CLK}$ / ( PCR_CPU_DIV_NUM + 1 ) |
| AHB_CLK[2] | HP_ROOT_CLK | $f_{HP\_ROOT\_CLK}$ / ( PCR_AHB_DIV_NUM + 1 ) |

[1] CPU_CLK frequency must be larger than or equal to AHB_CLK frequency, and must be an integer multiple of AHB_CLK frequency.

[2] AHB_CLK frequency can not exceed XTAL_CLK frequency.

**Note:**
When selecting the clock source of HP_ROOT_CLK, or configuring the clock divisor for CPU_CLK and AHB_CLK, please also set PCR_BUS_CLOCK_UPDATE to apply the new configuration, and read PCR_BUS_CLOCK_UPDATE to see if new configuration takes effect.

As shown in 8.2-1, to generate APB_CLK, AHB_CLK might be divided twice. The first division is compulsory. That is, AHB_CLK is always divided by the divisor (PCR_APB_DIV_NUM + 1). The second division (also called automatic frequency reduction) is optional. When there is no request from the host in the chip to access peripheral registers, AHB_CLK will be further divided by (APB_DECREASE_DIV_NUM + 1) to lower power consumption. If the host initiates a request to access peripheral registers, APB_CLK will be restored to the frequency after the first division.

Note that the function of automatic frequency reduction can be disabled (already disabled by default) by configuring APB_DECREASE_DIV_NUM to 0.

#### 8.2.4.2 LP System Clock

The LP system can operate when most other clocks are disabled. LP system clocks include LP_SLOW_CLK and LP_FAST_CLK.

The clock sources for LP_SLOW_CLK and LP_FAST_CLK are low-frequency clocks:

- LP_SLOW_CLK can be derived from:
  - RC_SLOW_CLK
  - XTAL32K_CLK
  - OSC_SLOW_CLK
- LP_FAST_CLK can be derived from:
  - 20 MHz XTAL_D2_CLK, which is XTAL_CLK divided by 2
  - RC_FAST_CLK
  - XTAL_CLK

The clock source of LP_DYN_SLOW_CLK is LP_SLOW_CLK. There is no frequency change from the clock source.

The clock source of LP_DYN_FAST_CLK depends on the chip's power mode (see Chapter 12 *Low-Power Management*).

- Select LP_FAST_CLK as its clock source in Active and Modem-sleep mode
- Select LP_SLOW_CLK as its clock source in Light-sleep and Deep-sleep mode

#### 8.2.4.3 Peripheral Clocks

Table 8.2-3, Table 8.2-4, Table 8.2-5, and Table 8.2-6 list the derived HP clock sources, HP clocks for each peripheral, derived LP clock sources and LP clocks for each peripheral.

Table 8.2-3. Derived HP Clock Source

| Derived Clock | Source Clock | | Derived Clock | Source Clock | | | | Derived Clock | | | | | | | | | Source Clock |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | XTAL_CLK 40 MHz | PLL_CLK PLL_F480M_CLK 480 MHz | PLL_F160M_CLK 160 MHz | RC_FAST_CLK 20 MHz | RC_SLOW_CLK 130 kHz | OSC_SLOW_CLK 32 kHz | XTAL32K_CLK 32 kHz | HP_ROOT_CLK 160 MHz/40 MHz/20 MHz | CRYPTO_CLK | APB_CLK | AHB_CLK | CPU_CLK | LP_DYN_ FAST_CLK | LP_DYN_ SLOW_CLK | XTAL_D2_CLK 20 MHz | LP_FAST_CLK | Clock from IO |
| PLL_F160M_CLK | | Y | | | | | | | | | | | | | | | |
| PLL_F120M_CLK | | Y | | | | | | | | | | | | | | | |
| PLL_F80M_CLK | | Y | | | | | | | | | | | | | | | |
| PLL_F48M_CLK | | Y | | | | | | | | | | | | | | | |
| PLL_F40M_CLK | | Y | | | | | | | | | | | | | | | |
| HP_ROOT_CLK | Y | | Y | Y | | | | | | | | | | | | | |
| CRYPTO_CLK | Y | Y | | Y | | | | | | | | | | | | | |
| APB_CLK | | | | | | | | Y | | | | | | | | | |
| AHB_CLK | | | | | | | | Y | | | | | | | | | |
| CPU_CLK | | | | | | | | Y | | | | | | | | | |

## Table 8.2-4. HP Clocks Used by Each Peripheral

| Peripherals | Source Clock | Derived Clock | | | | Source Clock | Derived Clock | | | | | | | | | Source Clock |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | XTAL_CLK 40 MHz | PLL_CLK PLL_F160M_CLK 160 MHz | PLL_F120M_CLK 120 MHz | PLL_F80M_CLK 80 MHz | PLL_F48M_CLK 48 MHz | RC_FAST_CLK 20 MHz | HP_ROOT_CLK 160 MHz/40 MHz/20 MHz | CRYPTO_CLK | APB_CLK | AHB_CLK | CPU_CLK | LP_DYN_ FAST_CLK | LP_DYN_ SLOW_CLK | XTAL_D2_CLK 20 MHz | LP_FAST_CLK | Clock from IO |
| Timer Group (TIMG) | Y | | | Y | | Y | | | | | | | | | | |
| Main System Watchdog Timers (MWDT) | Y | | | Y | | Y | | | | | | | | | | |
| I2S Controller (I2S) | Y | Y | Y | | | | | | | | | | | | | I2S_MCLK_in |
| UART Controller (UART) | Y | | | Y | | Y | | | | | | | | | | |
| I2C Controller (I2C) | Y | | | | | Y | | | | | | | | | | |
| SPI2 | Y | Y | | | | Y | | | | | | | | | | |
| SAR ADC | Y | | | Y | | Y | | | | | | | | | | |
| USB Serial/JTAG Controller | | | | | Y | | | | | | | | | | | |
| LED PWM Controller (LEDC) | Y | | | Y | | Y | | | | | | | | | | |
| System Timer | Y | | | | | Y | | | | | | | | | | |
| IO MUX | Y | | | Y | | Y | | | | | | | | | | |
| SHA Accelerator (SHA)<br>ECC Accelerator (ECC)<br>ECDSA Digital Signature Peripheral (ECDSA_DS) | | | | | | | | Y | | | | | | | | |
| Interrupt Matrix | | | | | | | | | | Y | | | | | | |
| Event Task Matrix (ETM) | | | | | | | | | | Y | | | | | | |
| GDMA Controller (GDMA) | | | | | | | | | | Y | | | | | | |
| Debug Assistant | | | | | | | | | | | Y | | | | | |
| RISC-V Trace Encoder (TRACE) | | | | | | | | | | Y | Y | | | | | |
| Interrupt Priority Registers (INTPRI) | | | | | | | | | | | Y | | | | | |

## Table 8.2-5. Derived LP Clock Source

| Derived Clock | Source Clock | | Derived Clock | Source Clock | | | | Derived Clock | | | | | | | | | Source Clock |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | XTAL_CLK 40 MHz | PLL_CLK PLL_F480M_CLK 480 MHz | PLL_F160M_CLK 160 MHz | RC_FAST_CLK 20 MHz | RC_SLOW_CLK 130 kHz | OSC_SLOW_CLK 32 kHz | XTAL32K_CLK 32 kHz | HP_ROOT_CLK 160 MHz/40 MHz/20 MHz | CRYPTO_CLK | APB_CLK | AHB_CLK | CPU_CLK | LP_DYN_ FAST_CLK | LP_DYN_ SLOW_CLK | XTAL_D2_CLK 20 MHz | LP_FAST_CLK | Clock from IO |
| LP_DYN_FAST_CLK | | | | | Y | Y | Y | | | | | | | | | Y | |
| LP_DYN_SLOW_CLK | | | | | Y | Y | Y | | | | | | | | | | |
| XTAL_D2_CLK | Y | | | | | | | | | | | | | | | | |
| LP_FAST_CLK | Y | | | Y | | | | | | | | | | | Y | | |

## Table 8.2-6. LP Clocks Used by Each Peripheral

| Derived Clock | Source Clock | | Derived Clock | Source Clock | | | | Derived Clock | | | | | | | | | Source Clock |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | XTAL_CLK 40 MHz | PLL_CLK PLL_F480M_CLK 480 MHz | PLL_F160M_CLK 160 MHz | RC_FAST_CLK 20 MHz | RC_SLOW_CLK 130 kHz | OSC_SLOW_CLK 32 kHz | XTAL32K_CLK 32 kHz | HP_ROOT_CLK 160 MHz/40 MHz/20 MHz | CRYPTO_CLK | APB_CLK | AHB_CLK | CPU_CLK | LP_DYN_ FAST_CLK | LP_DYN_ SLOW_CLK | XTAL_D2_CLK 20 MHz | LP_FAST_CLK | Clock from IO |
| eFuse Controller (eFuse) | | | | | | | | | | | | | | | Y | | |
| RTC Watchdog Timer (RWDT) | | | | | | | | | | | | | Y | Y | | | |
| RTC Timer (RTC Timer) | | | | | | | | | | | | | Y | Y | | | |
| Brownout Detector | | | | | | | | | | | | | Y | | | | |
| Power Management Unit (PMU) | | | | | | | | | | | | | Y | Y | | Y | |



GoBack

#### PLL_F480M_CLK

PLL_F480M_CLK is a 480 MHz clock. PLL_F160M_CLK, PLL_F120M_CLK, PLL_F80M_CLK, PLL_F48M_CLK, PLL_F40M_CLK, and CRYPTO_CLK are divided from PLL_F480M_CLK.

#### CRYPTO_CLK

As shown in Figure 8.2-3, CRYPTO_CLK can be derived from XTAL_CLK, PLL_F480M_CLK, or RC_FAST_CLK, and its frequency is up to 160 MHz.

To protect encryption and decryption peripherals from DPA (Differential Power Analysis) attacks, a random divider strategy is implemented for the functional clock of encryption and decryption peripherals. Four security levels are available, depending on the range of random divider. Users can select the security level by configuring HP_SYSTEM_SEC_DPA_CONF_REG. If HP_SYSTEM_SEC_DPA_CFG_SEL is set to 1, the security level is determined by the configuration of EFUSE_SEC_DPA_LEVEL, otherwise, by the value of HP_SYSTEM_SEC_DPA_LEVEL.

#### LED_PWM Clock

LEDC module uses PLL_F80M_CLK, RC_FAST_CLK or XTAL_CLK as its clock source. When the system is in low-power mode (APB_CLK is disabled), most peripherals are halted, but LEDC can still work via RC_FAST_CLK.

#### Slow Clock for Timer Group 0

Using XTAL_CLK as a reference, TIMG0 can calculate the frequency of slow clock sources provided by ESP32-C61. The slow clock source can be selected by configuring PCR_32K_SEL:

0: Select RC_FAST_CLK clock
1: Select XTAL32K_CLK clock
2: Select OSC_SLOW_CLK clock
3: Select RC_SLOW_CLK clock

#### Low-Power System Clock Registers

See 8.4.2 *LP System Clock Register Summary* for the clock registers of low-power system peripherals.

### 8.2.4.4 PMU Control of High-Performance System Clock Gating

In various operating modes of ESP32-C61, the following register fields can be pre-configured to enable the Power Management Unit (PMU) to control the clock gating of high-performance system peripherals:

- PMU_HP_*x*_DIG_ICG_APB_EN (x = ACTIVE/MODEM/SLEEP): Controls the clock gating for reading and writing registers of high-performance system peripherals.
- PMU_HP_*x*_DIG_ICG_FUNC_EN (x = ACTIVE/MODEM/SLEEP): Controls the functional clock gating of high-performance system peripherals.

For the specific configuration process, please refer to the Chapter 12 *Low-Power Management*.

Tables 8.2-7 and 8.2-8 list the correspondence between PMU pre-configured register bits and high-performance system clock gating.

Table 8.2-7. Mapping between controlling the clock gating of read/write registers and related PMU registers

| PMU_HP_*x*_DIG_ICG_APB_EN bit | Control the clock gating of read/write registers |
|---|---|
| 0 | SHA Accelerator (SHA)<br>ECC Accelerator (ECC)<br>ECDSA Digital Signature Peripheral (ECDSA_DS) |
| 1 | GDMA Controller (GDMA) |
| 2 | SPI2 |
| 3 | Interrupt Matrix |
| 4 | I2S Controller (I2S) |
| 6 | UART0 |
| 7 | UART1 |
| 11 | Timer Group 0 |
| 12 | Timer Group 1 |
| 13 | I2C Controller (I2C) |
| 14 | LED PWM Controller (LEDC) |
| 16 | System Timer |
| 17 | USB Serial/JTAG Controller |
| 22 | Event Task Matrix (ETM) |
| 25 | Debug Assistant |
| 26 | GPIO Matrix and IO MUX |
| 29 | UART2 |

Table 8.2-8. Mapping between controlling the functional clock gating of high-performance system peripherals and related PMU registers

| PMU_HP_x_DIG_ICG_FUNC_EN bit | control the functional clock gating of high-performance system peripherals |
|---|---|
| 0 | GDMA Controller (GDMA) |
| 1 | SPI2 |
| 2 | I2S Receive Side |
| 3 | UART0 |
| 4 | UART1 |
| 6 | USB Serial/JTAG Controller |
| 7 | I2S Transmit Side |
| 10 | MEM_MONITOR |
| 12 | Temperature Sensor |
| 13 | Timer Group 0 |
| 14 | Timer Group 1 |
| 16 | Event Task Matrix (ETM) |
| 17 | ESP-RISC-V CPU<br>ASSIST_DEBUG |
| 18 | System Timer |
| 19 | SHA Accelerator (SHA)<br>ECC Accelerator (ECC)<br>ECDSA Digital Signature Peripheral (ECDSA_DS) |
| 22 | UART2 |
| 27 | LED PWM Controller (LEDC) |
| 28 | GPIO Matrix and IO MUX |
| 29 | I2C Controller (I2C) |

## 8.3 Programming Procedures

### 8.3.1 HP System Clock Configuration

When configuring PCR_SOC_CLK_SEL to select the clock source of HP_ROOT_CLK, or configuring the clock divisor for CPU_CLK via PCR_CPU_DIV_NUM and AHB_CLK via PCR_AHB_DIV_NUM, please also set the enable register PCR_BUS_CLOCK_UPDATE to apply the new configuration. To check whether the new configuration takes effect, read PCR_BUS_CLOCK_UPDATE and see if it is 0.

### 8.3.2 LP System Clock Configuration

The clock source of LP_SLOW_CLK can be configured via LP_CLKRST_SLOW_CLK_SEL.

The clock source of LP_FAST_CLK can be configured via LP_CLKRST_FAST_CLK_SEL.

### 8.3.3 Peripheral Clock Reset and Configuration

**Notice:**

ESP32-C61 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 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.
- Functional clock: such as UART's reference clock, used by peripherals to operate.

The functional clock of most peripherals can be selected from multiple clock sources. For clock gating registers, whether they are used to gate the bus clock (AHB_CLK, APB_CLK) or the functional clock will be stated in the corresponding register descriptions.

Bus clock switches, functional clock switches, and configuration registers for clock source selection and clock frequency division are grouped into the PCR module. For more information, see Section 8.4 *Register Summary*.

When a peripheral is not working, users can turn off its functional clock by configuring related PCR registers. Turning off the peripheral's functional clock does not affect the rest of the system.

Take the I2C clock configuration as an example.

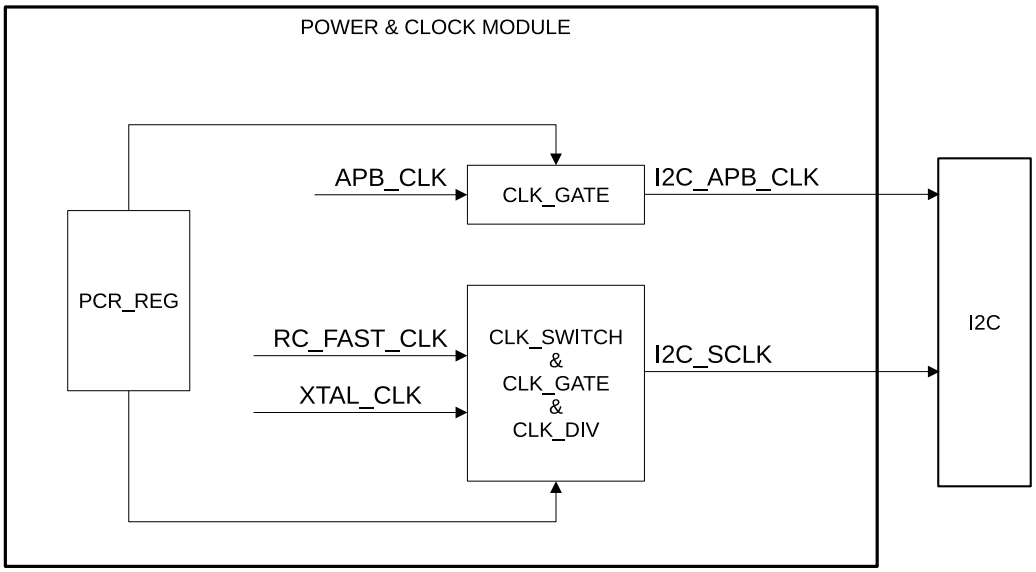



**Figure 8.3-1. I2C Clock Configuration Example**

Figure 8.3-1 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 functional clock, the bus clock of the peripheral can also be turned off to further lower the 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.

# 8.4 Register Summary

## 8.4.1 PCR Register Summary

The addresses in this section are relative to Power/Clock/Reset (PCR) Register base address provided in Table 5.3-2 in Chapter 5 *System and Memory*.

The abbreviations given in Column **Access** are explained in Section Access Types for Registers.

| Name | Description | Address | Access |
|---|---|---|---|
| **Configuration Register** | | | |
| PCR_UART0_CONF_REG | UART0 configuration register | 0x0000 | varies |
| PCR_UART0_SCLK_CONF_REG | UART0_SCLK configuration register | 0x0004 | R/W |
| PCR_UART0_PD_CTRL_REG | UART0 power control register | 0x0008 | R/W |
| PCR_UART1_CONF_REG | UART1 configuration register | 0x000C | varies |
| PCR_UART1_SCLK_CONF_REG | UART1_SCLK configuration register | 0x0010 | R/W |
| PCR_UART1_PD_CTRL_REG | UART1 power control register | 0x0014 | R/W |
| PCR_UART2_CONF_REG | UART2 configuration register | 0x0018 | varies |
| PCR_UART2_SCLK_CONF_REG | UART2_SCLK configuration register | 0x001C | R/W |
| PCR_UART2_PD_CTRL_REG | UART2 power control register | 0x0020 | R/W |
| PCR_I2C_CONF_REG | I2C configuration register | 0x002C | R/W |
| PCR_I2C_SCLK_CONF_REG | I2C_SCLK configuration register | 0x0030 | R/W |
| PCR_LEDC_CONF_REG | LEDC configuration register | 0x0034 | varies |
| PCR_LEDC_SCLK_CONF_REG | LEDC_SCLK configuration register | 0x0038 | R/W |
| PCR_LEDC_PD_CTRL_REG | LEDC power control register | 0x003C | R/W |
| PCR_TIMERGROUP0_CONF_REG | TIMERGROUP0 configuration register | 0x0040 | varies |
| PCR_TIMERGROUP0_TIMER_CLK_CONF_REG | TIMERGROUP0_TIMER_CLK configuration register | 0x0044 | R/W |
| PCR_TIMERGROUP0_WDT_CLK_CONF_REG | TIMERGROUP0_WDT_CLK configuration register | 0x0048 | R/W |
| PCR_TIMERGROUP1_CONF_REG | TIMERGROUP1 configuration register | 0x004C | varies |
| PCR_TIMERGROUP1_TIMER_CLK_CONF_REG | TIMERGROUP1_TIMER_CLK configuration register | 0x0050 | R/W |
| PCR_TIMERGROUP1_WDT_CLK_CONF_REG | TIMERGROUP1_WDT_CLK configuration register | 0x0054 | R/W |
| PCR_SYSTIMER_CONF_REG | SYSTIMER configuration register | 0x0058 | varies |
| PCR_SYSTIMER_FUNC_CLK_CONF_REG | SYSTIMER_FUNC_CLK configuration register | 0x005C | R/W |
| PCR_I2S_CONF_REG | I2S configuration register | 0x0060 | varies |
| PCR_I2S_TX_CLKM_CONF_REG | I2S_TX_CLKM configuration register | 0x0064 | R/W |
| PCR_I2S_TX_CLKM_DIV_CONF_REG | I2S_TX_CLKM_DIV configuration register | 0x0068 | R/W |
| PCR_I2S_RX_CLKM_CONF_REG | I2S_RX_CLKM configuration register | 0x006C | R/W |
| PCR_I2S_RX_CLKM_DIV_CONF_REG | I2S_RX_CLKM_DIV configuration register | 0x0070 | R/W |
| PCR_SARADC_CONF_REG | SARADC configuration register | 0x0074 | R/W |

| Name | Description | Address | Access |
|---|---|---|---|
| PCR_SARADC_CLKM_CONF_REG | SARADC_CLKM configuration register | 0x0078 | R/W |
| PCR_TSENS_CLK_CONF_REG | TSENS_CLK configuration register | 0x007C | R/W |
| PCR_USB_DEVICE_CONF_REG | USB_DEVICE configuration register | 0x0080 | varies |
| PCR_INTMTX_CONF_REG | INTMTX configuration register | 0x0084 | varies |
| PCR_GDMA_CONF_REG | GDMA configuration register | 0x0090 | R/W |
| PCR_SPI2_CONF_REG | SPI2 configuration register | 0x0094 | varies |
| PCR_SPI2_CLKM_CONF_REG | SPI2_CLKM configuration register | 0x0098 | R/W |
| PCR_SHA_CONF_REG | SHA configuration register | 0x00A0 | varies |
| PCR_ECC_CONF_REG | ECC configuration register | 0x00AC | varies |
| PCR_ECC_PD_CTRL_REG | ECC power control register | 0x00B0 | R/W |
| PCR_ECDSA_CONF_REG | ECDSA_DS configuration register | 0x00BC | varies |
| PCR_IOMUX_CONF_REG | IOMUX configuration register | 0x00C0 | R/W |
| PCR_IOMUX_CLK_CONF_REG | IOMUX_CLK configuration register | 0x00C4 | R/W |
| PCR_TCM_MEM_MONITOR_CONF_REG | TCM_MEM_MONITOR configuration register | 0x00C8 | varies |
| PCR_PSRAM_MEM_MONITOR_CONF_REG | PSRAM_MEM_MONITOR configuration register | 0x00CC | varies |
| PCR_TRACE_CONF_REG | TRACE configuration register | 0x00D0 | R/W |
| PCR_ASSIST_CONF_REG | ASSIST configuration register | 0x00D4 | R/W |
| PCR_CACHE_CONF_REG | CACHE configuration register | 0x00D8 | R/W |
| PCR_CACHE_PD_CTRL_REG | CACHE power control register | 0x00DC | R/W |
| PCR_TIMEOUT_CONF_REG | TIMEOUT configuration register | 0x00E4 | R/W |
| PCR_SYSCLK_CONF_REG | SYSCLK configuration register | 0x00E8 | varies |
| PCR_CPU_WAITI_CONF_REG | CPU_WAITI configuration register | 0x00EC | R/W |
| PCR_CPU_FREQ_CONF_REG | CPU_FREQ configuration register | 0x00F0 | R/W |
| PCR_AHB_FREQ_CONF_REG | AHB_FREQ configuration register | 0x00F4 | R/W |
| PCR_APB_FREQ_CONF_REG | APB_FREQ configuration register | 0x00F8 | R/W |
| PCR_PLL_DIV_CLK_EN_REG | PLL DIV clock-gating configuration register | 0x0100 | R/W |
| PCR_CTRL_32K_CONF_REG | 32 kHz clock configuration register | 0x010C | R/W |
| PCR_SEC_CONF_REG | Clock source configuration register for External Memory Encryption and Decryption | 0x0118 | R/W |
| PCR_BUS_CLK_UPDATE_REG | Configuration register for applying updated high-performance system clock sources | 0x0120 | R/W/WTC |
| PCR_SAR_CLK_DIV_REG | SAR ADC clock divisor configuration register | 0x0124 | R/W |
| PCR_TIMERGROUP_WDT_CONF_REG | TIMERGROUP_WDT configuration register | 0x012C | R/W |
| PCR_TIMERGROUP_XTAL_CONF_REG | TIMERGROUP1 configuration register | 0x0130 | R/W |
| PCR_ETM_CONF_REG | ETM configuration register | 0x013C | varies |
| **Version Register** | | | |
| PCR_DATE_REG | Version control register. | 0x0FFC | R/W |

### 8.4.2 LP System Clock Register Summary

The addresses of the last two registers with the LPPERI prefix in this section are relative to the Low-power Peripheral Register (LPPERI) base address. The addresses of the third and fourth to last registers with the LP_AON prefix in this section are relative to the Low-power Always-on Register (LP_AON) base address. The other addresses in this section are relative to the Low-power Clock/Reset Register (LP_CLKRST) base address. For base address, please refer to Table 5.3-2 in Chapter 5 *System and Memory*.

The abbreviations given in Column **Access** are explained in Section Access Types for Registers.

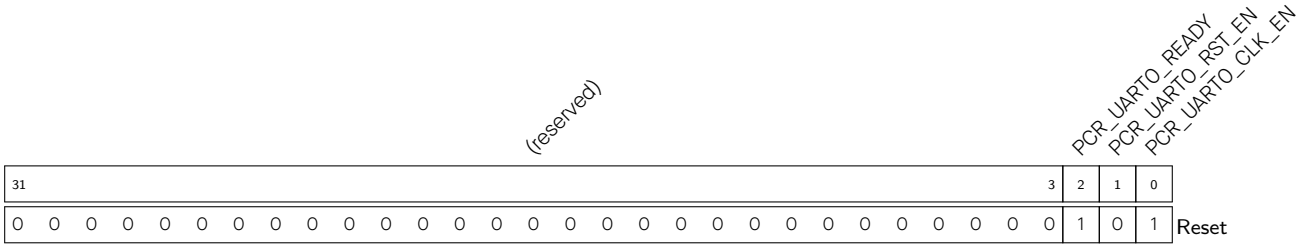
| Name | Description | Address | Access |
|---|---|---|---|
| **configure_register** | | | |
| LP_CLKRST_LP_CLK_CONF_REG | LP system clock source configuration register | 0x0000 | R/W |
| LP_CLKRST_LP_CLK_PO_EN_REG | Configuration register for gating clock signals to pins | 0x0004 | R/W |
| LP_CLKRST_LP_CLK_EN_REG | Configuration register for gating LP clock source | 0x0008 | R/W |
| LP_CLKRST_LP_RST_EN_REG | Configuration register for LP peripheral software reset | 0x000C | R/W |
| LP_CLKRST_RESET_CAUSE_REG | Reset cause status register | 0x0010 | varies |
| LP_CLKRST_CPU_RESET_REG | CPU reset configuration register | 0x0014 | R/W |
| LP_CLKRST_FOSC_CNTL_REG | RC_FAST_CLK frequency configuration register | 0x0018 | R/W |
| LP_CLKRST_CLK_TO_HP_REG | Configuration register for gating clock signals to HP system | 0x0020 | R/W |
| LP_CLKRST_DATE_REG | Version control register | 0x03FC | R/W |

# 8.5 Registers

## 8.5.1 PCR Registers

The addresses in this section are relative to the Power/Clock/Reset (PCR) Register base address provided in Table 5.3-2 in Chapter 5 *System and Memory*.

**Register 8.1. PCR_UART0_CONF_REG (0x0000)**




**PCR_UART0_CLK_EN** Configures whether or not to enable APB_CLK for UART0.

0: Not enable

1: Enable

(R/W)

**PCR_UART0_RST_EN** Configures whether or not to reset UART0.

0: Not reset

1: Reset

(R/W)

**PCR_UART0_READY** Represents whether or not UART0 is released from reset.

0: Not released

1: Released

(RO)

Register 8.2. PCR_UART0_SCLK_CONF_REG (0x0004)




**PCR_UART0_SCLK_DIV_A** Configures the denominator of the frequency divider factor of the UART0 functional clock. (R/W)

**PCR_UART0_SCLK_DIV_B** Configures the numerator of the frequency divider factor of the UART0 functional clock. (R/W)

**PCR_UART0_SCLK_DIV_NUM** Configures the integral part of the frequency divider factor of the UART0 functional clock. (R/W)

**PCR_UART0_SCLK_SEL** Configures the clock source of UART0.

0 (default): XTAL_CLK

1: RC_FAST_CLK

2: PLL_F80M_CLK

(R/W)

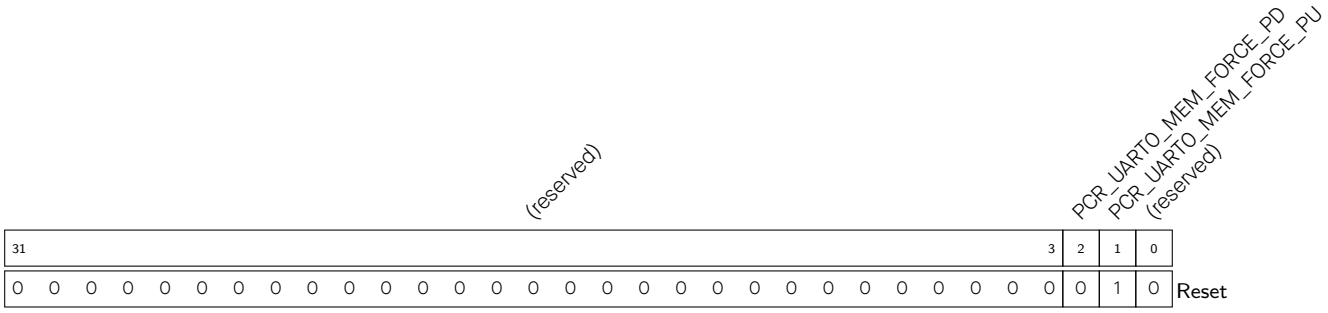
**PCR_UART0_SCLK_EN** Configures whether or not to enable UART0 functional clock.

0: Not enable

1: Enable

(R/W)

Register 8.3. PCR_UART0_PD_CTRL_REG (0x0008)




**PCR_UART0_MEM_FORCE_PU** Configures whether or not to force power up UART0 memory.

0: Not force power up UART0 memory

1: Force power up UART0 memory

(R/W)

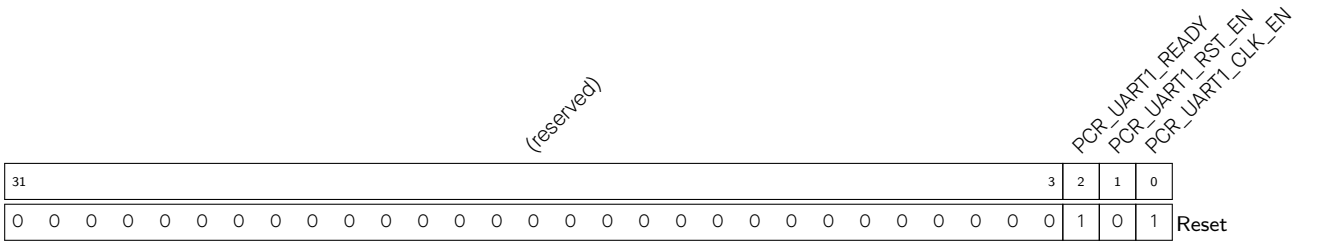
**PCR_UART0_MEM_FORCE_PD** Configures whether or not to force power down UART0 memory.

0: Not force power down UART0 memory

1: Force power down UART0 memory

(R/W)

Register 8.4. PCR_UART1_CONF_REG (0x000C)




**PCR_UART1_CLK_EN** Configures whether or not to enable APB_CLK for UART1.

0: Not enable

1: Enable

(R/W)

**PCR_UART1_RST_EN** Configures whether or not to reset UART1.

0: Not reset

1: Reset

(R/W)

**PCR_UART1_READY** Represents whether or not UART1 is released from reset.

0: Not released

1: Released

(RO)

Register 8.5. PCR_UART1_SCLK_CONF_REG (0x0010)




**PCR_UART1_SCLK_DIV_A** Configures the denominator of the frequency divider factor of the UART1 functional clock. (R/W)

**PCR_UART1_SCLK_DIV_B** Configures the numerator of the frequency divider factor of the UART1 functional clock. (R/W)

**PCR_UART1_SCLK_DIV_NUM** Configures the integral part of the frequency divider factor of the UART1 functional clock. (R/W)

**PCR_UART1_SCLK_SEL** Configures the clock source of UART1.

0 (default): XTAL_CLK

1: RC_FAST_CLK

2: PLL_F80M_CLK

(R/W)

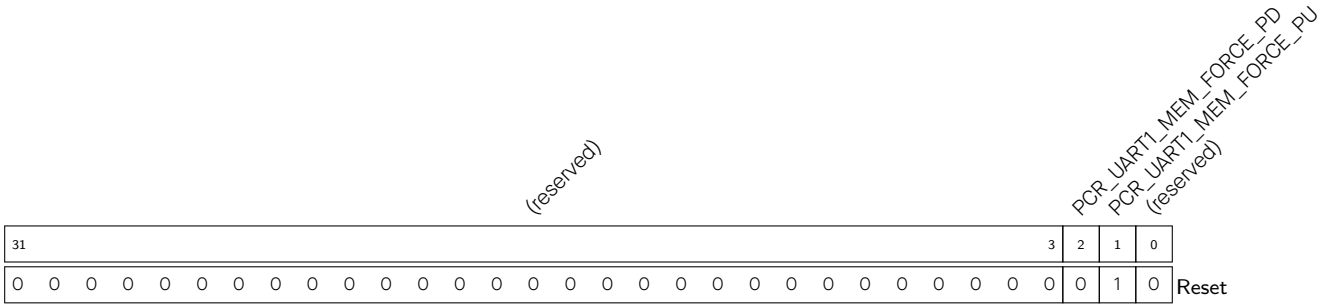
**PCR_UART1_SCLK_EN** Configures whether or not to enable UART1 functional clock.

0: Not enable

1: Enable

(R/W)

Register 8.6. PCR_UART1_PD_CTRL_REG (0x0014)




**PCR_UART1_MEM_FORCE_PU** Configures whether or not to force power up UART1 memory.

0: Not force power up UART1 memory

1: Force power up UART1 memory

(R/W)

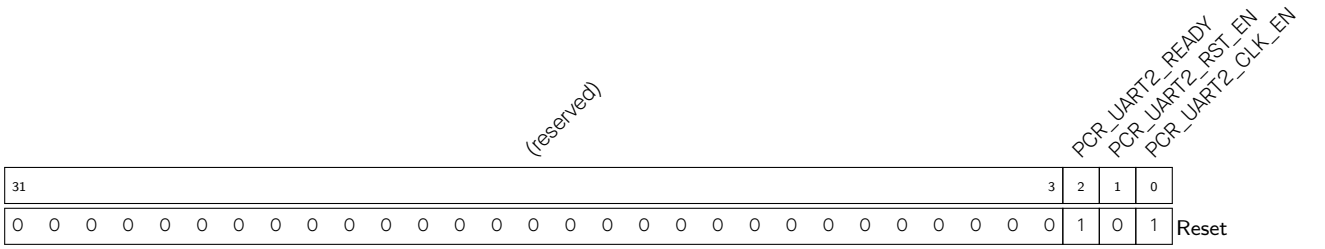
**PCR_UART1_MEM_FORCE_PD** Configures whether or not to force power down UART1 memory.

0: Not force power down UART1 memory

1: Force power down UART1 memory

(R/W)

Register 8.7. PCR_UART2_CONF_REG (0x0018)




**PCR_UART2_CLK_EN** Configures whether or not to enable APB_CLK for UART2.

0: Not enable

1: Enable

(R/W)

**PCR_UART2_RST_EN** Configures whether or not to reset UART2.

0: Not reset

1: Reset

(R/W)

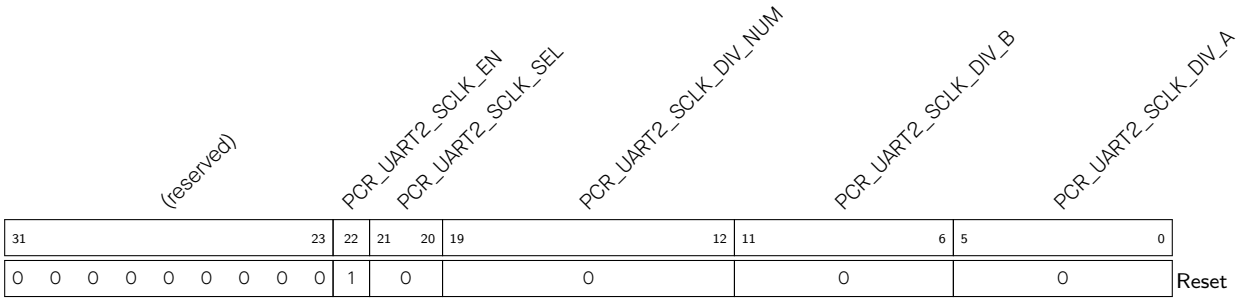
**PCR_UART2_READY** Represents whether or not UART2 is released from reset.

0: Not released

1: Released

(RO)

Register 8.8. PCR_UART2_SCLK_CONF_REG (0x001C)




**PCR_UART2_SCLK_DIV_A** Configures the denominator of the frequency divider factor of the UART2 functional clock. (R/W)

**PCR_UART2_SCLK_DIV_B** Configures the numerator of the frequency divider factor of the UART2 functional clock. (R/W)

**PCR_UART2_SCLK_DIV_NUM** Configures the integral part of the frequency divider factor of the UART2 functional clock. (R/W)

**PCR_UART2_SCLK_SEL** Configures the clock source of UART2.

0 (default): XTAL_CLK

1: RC_FAST_CLK

2: PLL_F80M_CLK

(R/W)

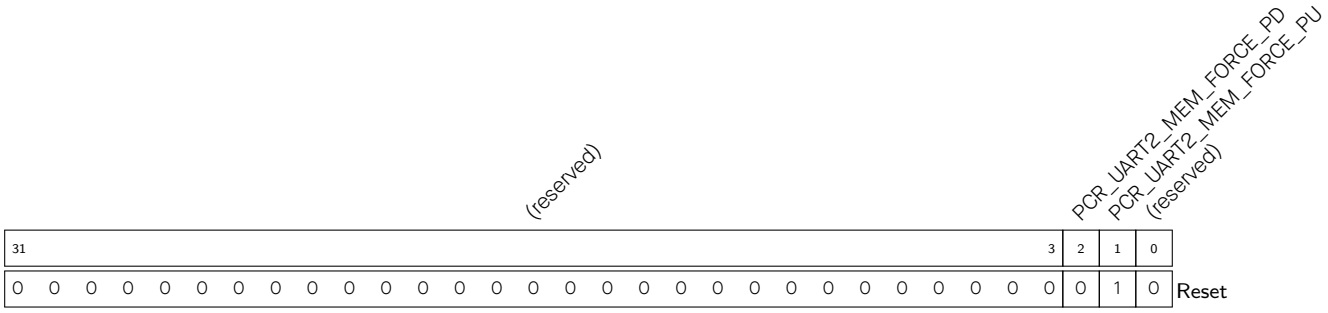
**PCR_UART2_SCLK_EN** Configures whether or not to enable UART2 functional clock.

0: Not enable

1: Enable

(R/W)

Register 8.9. PCR_UART2_PD_CTRL_REG (0x0020)




**PCR_UART2_MEM_FORCE_PU** Configures whether or not to force power up UART2 memory.

0: Not force power up UART2 memory

1: Force power up UART2 memory

(R/W)

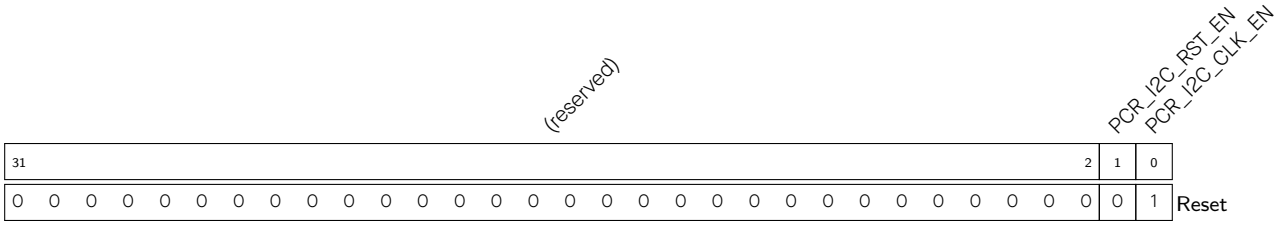
**PCR_UART2_MEM_FORCE_PD** Configures whether or not to force power down UART2 memory.

0: Not force power down UART2 memory

1: Force power down UART2 memory

(R/W)

Register 8.10. PCR_I2C_CONF_REG (0x002C)




**PCR_I2C_CLK_EN** Configures whether or not to enable APB_CLK for I2C.

0: Not enable

1: Enable

(R/W)

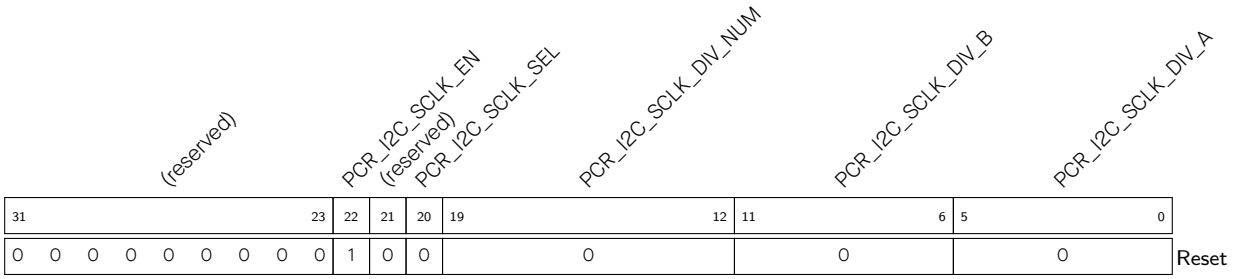
**PCR_I2C_RST_EN** Configures whether or not to reset I2C.

0: Not reset

1: Reset

(R/W)

Register 8.11. PCR_I2C_SCLK_CONF_REG (0x0030)




**PCR_I2C_SCLK_DIV_A** Configures the denominator of the divisor's fractional part for I2C functional clock. (R/W)

**PCR_I2C_SCLK_DIV_B** Configures the numerator of the divisor's fractional part for I2C functional clock. (R/W)

**PCR_I2C_SCLK_DIV_NUM** Configures the integral part of the divisor for I2C functional clock. (R/W)

**PCR_I2C_SCLK_SEL** Configures the clock source of I2C.

0 (default): XTAL_CLK

1: RC_FAST_CLK

(R/W)

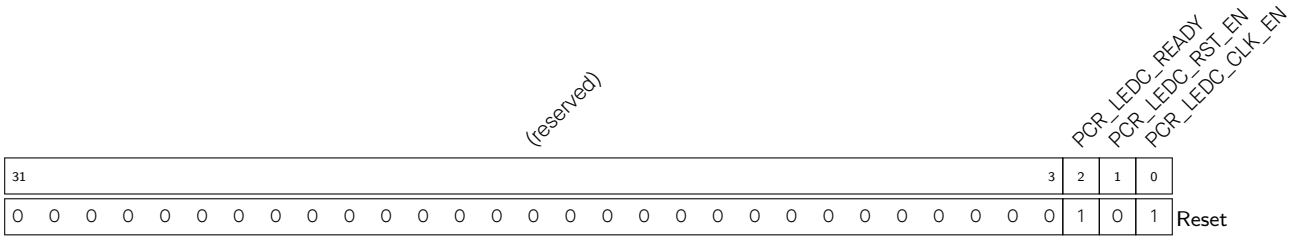
**PCR_I2C_SCLK_EN** Configures whether or not to enable I2C functional clock.

0: Not enable

1: Enable

(R/W)

Register 8.12. PCR_LEDC_CONF_REG (0x0034)




**PCR_LEDC_CLK_EN** Configures whether or not to enable APB_CLK for LEDC.

0: Not enable

1: Enable

(R/W)

**PCR_LEDC_RST_EN** Configures whether or not to reset LEDC.

0: Not reset

1: Reset

(R/W)

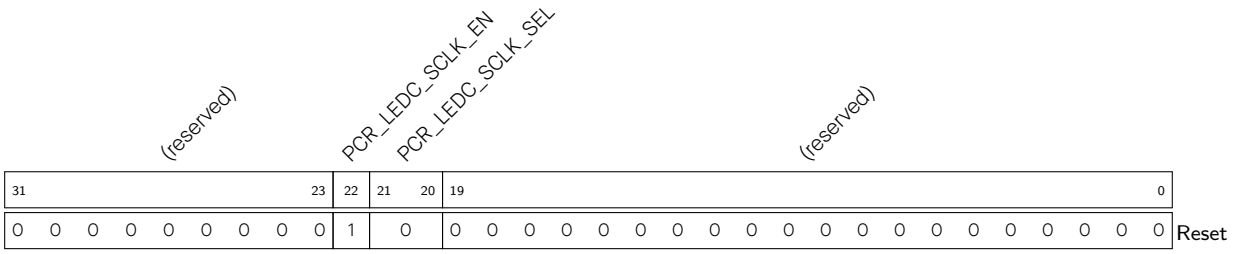
**PCR_LEDC_READY** Represents whether or not LEDC is released from reset.

0: Not released

1: Released

(RO)

Register 8.13. PCR_LEDC_SCLK_CONF_REG (0x0038)




**PCR_LEDC_SCLK_SEL** Configures the clock source of LEDC.

0 (default): XTAL_CLK

1: RC_FAST_CLK

2: PLL_F80M_CLK

(R/W)

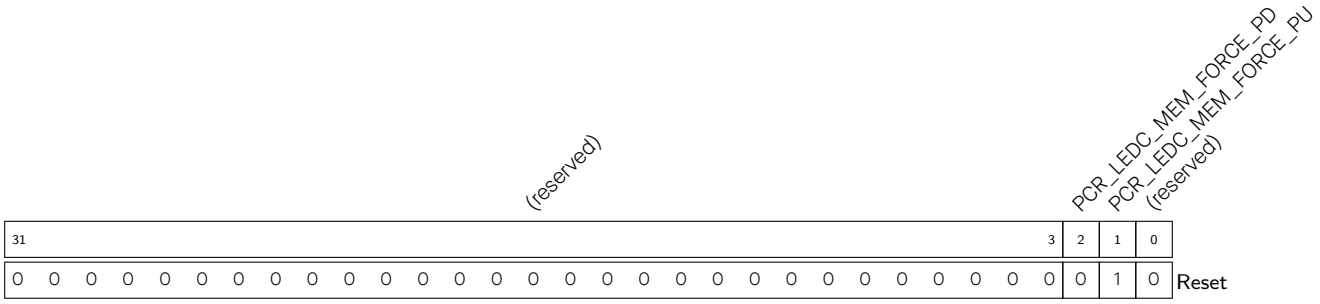
**PCR_LEDC_SCLK_EN** Configures whether or not to enable FUNC_CLK for LEDC.

0: Not enable

1: Enable

(R/W)

Register 8.14. PCR_LEDC_PD_CTRL_REG (0x003C)




**PCR_LEDC_MEM_FORCE_PU** Configures whether or not to force power up LEDC memory.

0: Not force power up LEDC memory

1: Force power up LEDC memory

(R/W)

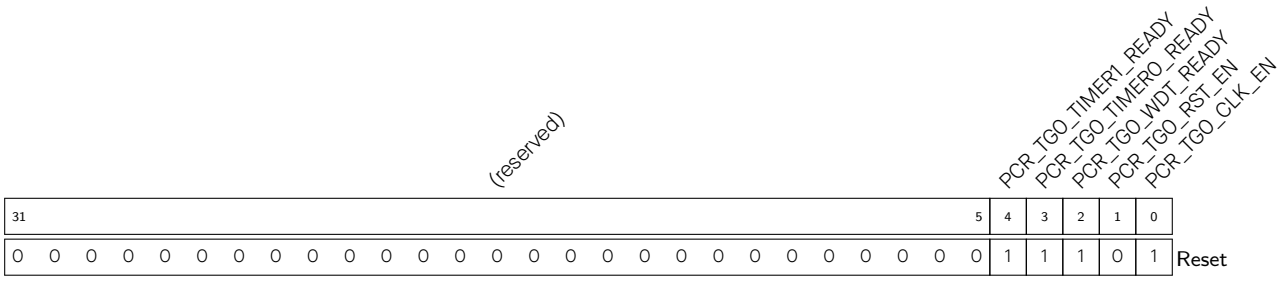
**PCR_LEDC_MEM_FORCE_PD** Configures whether or not to force power down LEDC memory.

0: Not force power down LEDC memory

1: Force power down LEDC memory

(R/W)

Register 8.15. PCR_TIMERGROUP0_CONF_REG (0x0040)




**PCR_TG0_CLK_EN** Configures whether or not to enable APB_CLK for Timer Group 0.

0: Not enable

1: Enable

(R/W)

**PCR_TG0_RST_EN** Configures whether or not to reset Timer Group 0.

0: Not reset

1: Reset

(R/W)

**PCR_TG0_WDT_READY** Represents whether or not the WDT in Timer Group 0 is released from reset.

0: Not released

1: Released

(RO)

**PCR_TG0_TIMER0_READY** Represents whether or not Timer 0 in Timer Group 0 is released from reset.

0: Not released

1: Released

(RO)

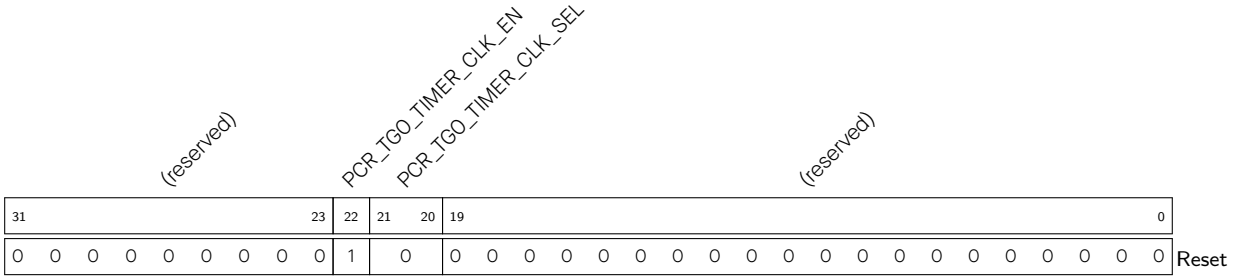
**PCR_TG0_TIMER1_READY** Represents whether or not Timer 1 in Timer Group 0 is released from reset.

0: Not released

1: Released

(RO)

Register 8.16. PCR_TIMERGROUP0_TIMER_CLK_CONF_REG (0x0044)




**PCR_TG0_TIMER_CLK_SEL** Configures the clock source of general-purpose timers in Timer Group 0.

0 (default): XTAL_CLK

1: RC_FAST_CLK

2: PLL_F80M_CLK

(R/W)

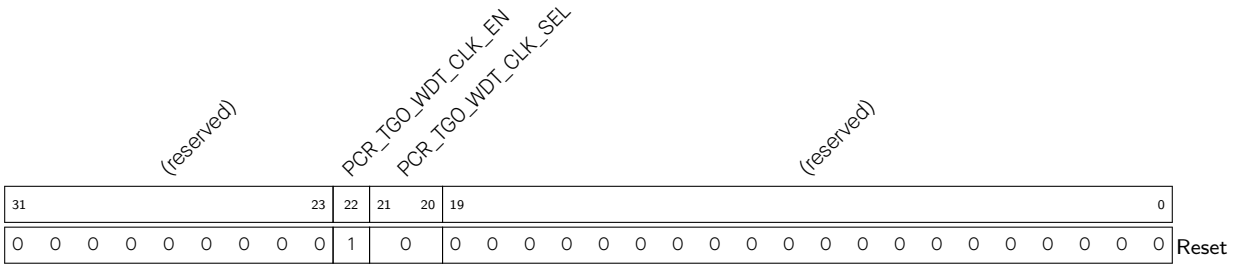
**PCR_TG0_TIMER_CLK_EN** Configures whether or not to enable the clock of general-purpose timers in Timer Group 0.

0: Not enable

1: Enable

(R/W)

Register 8.17. PCR_TIMERGROUP0_WDT_CLK_CONF_REG (0x0048)




**PCR_TG0_WDT_CLK_SEL** Configures the clock source of WDT in Timer Group 0.

0 (default): XTAL_CLK

1: RC_FAST_CLK

2: PLL_F80M_CLK

(R/W)

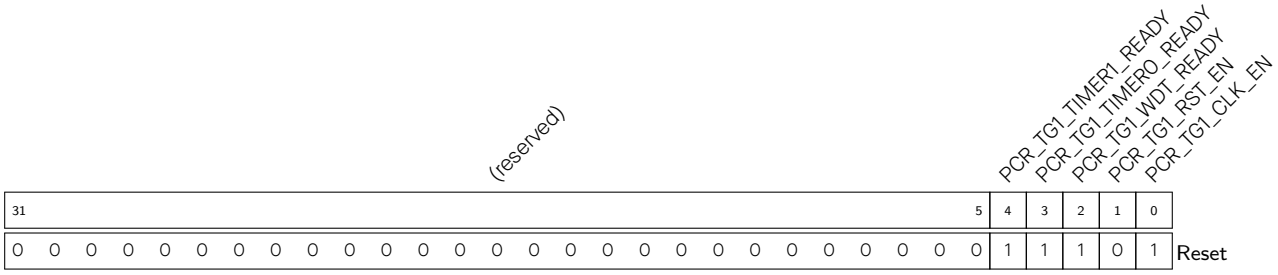
**PCR_TG0_WDT_CLK_EN** Configures whether or not to enable the clock of WDT in Timer Group 0.

0: Not enable

1: Enable

(R/W)

Register 8.18. PCR_TIMERGROUP1_CONF_REG (0x004C)




**PCR_TG1_CLK_EN** Configures whether or not to enable APB_CLK for Timer Group 1.

0: Not enable

1: Enable

(R/W)

**PCR_TG1_RST_EN** Configures whether or not to reset Timer Group 1.

0: Not reset

1: Reset

(R/W)

**PCR_TG1_WDT_READY** Represents whether or not the WDT in Timer Group 1 is released from reset.

0: Not released

1: Released

(RO)

**PCR_TG1_TIMER0_READY** Represents whether or not Timer 0 in Timer Group 1 is released from reset.

0: Not released

1: Released

(RO)timer_group0 Timer 0module (RO)

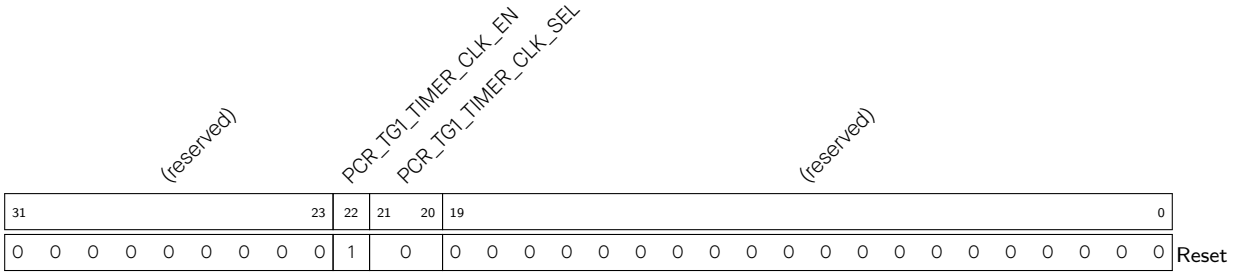
**PCR_TG1_TIMER1_READY** Represents whether or not Timer 1 in Timer Group 1 is released from reset.

0: Not released

1: Released

(RO)

Register 8.19. PCR_TIMERGROUP1_TIMER_CLK_CONF_REG (0x0050)




**PCR_TG1_TIMER_CLK_SEL** Configures the clock source of general-purpose timers in Timer Group 1.

0 (default): XTAL_CLK

1: RC_FAST_CLK

2: PLL_F48M_CLK

3: No clock source

(R/W)

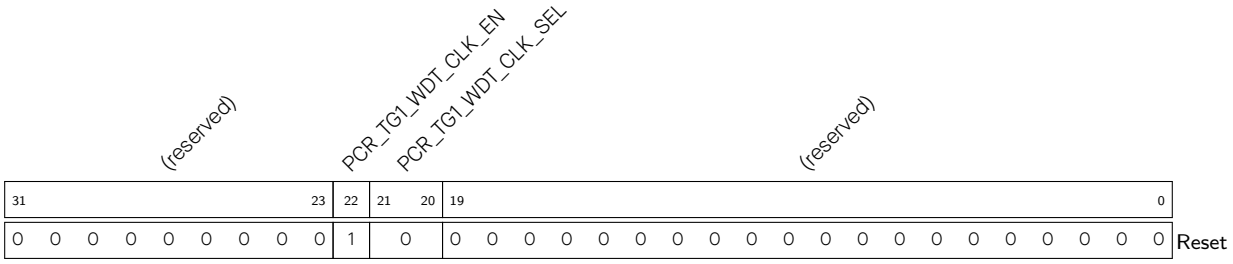
**PCR_TG1_TIMER_CLK_EN** Configures whether or not to enable the clock of general-purpose timers in Timer Group 1.

0: Not enable

1: Enable

(R/W)

Register 8.20. PCR_TIMERGROUP1_WDT_CLK_CONF_REG (0x0054)




**PCR_TG1_WDT_CLK_SEL** Configures the clock source of WDT in Timer Group 1.

0 (default): XTAL_CLK

1: RC_FAST_CLK

2: PLL_F80M_CLK

(R/W)

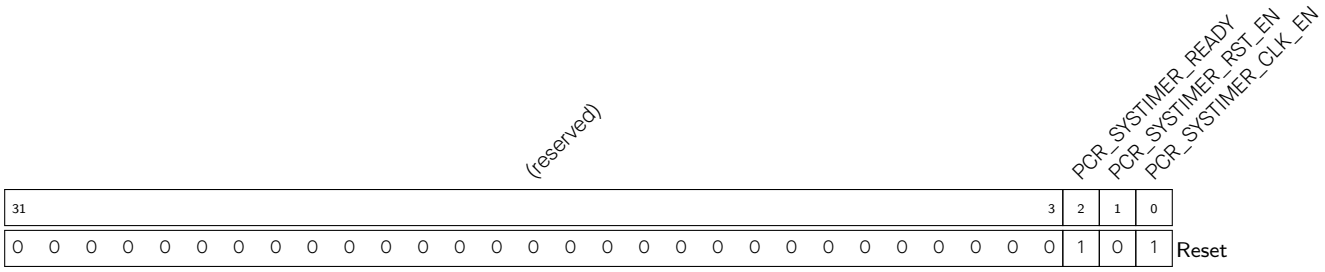
**PCR_TG1_WDT_CLK_EN** Configures whether or not to enable the clock for WDT in Timer Group 1.

0: Not enable

1: Enable

(R/W)

Register 8.21. PCR_SYSTIMER_CONF_REG (0x0058)




**PCR_SYSTIMER_CLK_EN** Configures whether or not to enable APB_CLK for System Timer.

0: Not enable

1: Enable

(R/W)

**PCR_SYSTIMER_RST_EN** Configures whether or not to reset System Timer.

0: Not reset

1: Reset

(R/W)

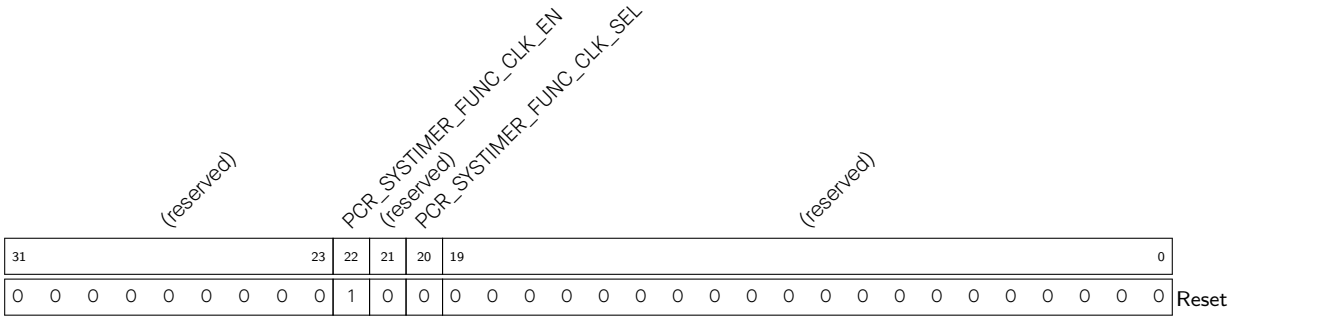
**PCR_SYSTIMER_READY** Represents whether or not the System Timer is released from reset.

0: Not released

1: Released

(RO)

Register 8.22. PCR_SYSTIMER_FUNC_CLK_CONF_REG (0x005C)




**PCR_SYSTIMER_FUNC_CLK_SEL** Configures the clock source of System Timer.

0 (default): XTAL_CLK

1: RC_FAST_CLK

(R/W)

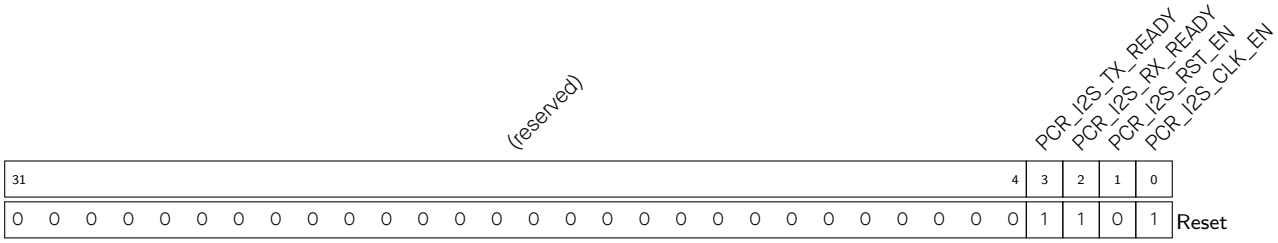
**PCR_SYSTIMER_FUNC_CLK_EN** Configures whether or not to enable System Timer functional clock.

0: Not enable

1: Enable

(R/W)

Register 8.23. PCR_I2S_CONF_REG (0x0060)




**PCR_I2S_CLK_EN** Configures whether or not to enable APB_CLK for I2S.

0: Not enable

1: Enable

(R/W)

**PCR_I2S_RST_EN** Configures whether or not to reset I2S.

0: Not reset

1: Reset

(R/W)

**PCR_I2S_RX_READY** Represents whether or not I2S RX is released from reset.

0: Not released

1: Released

(RO)

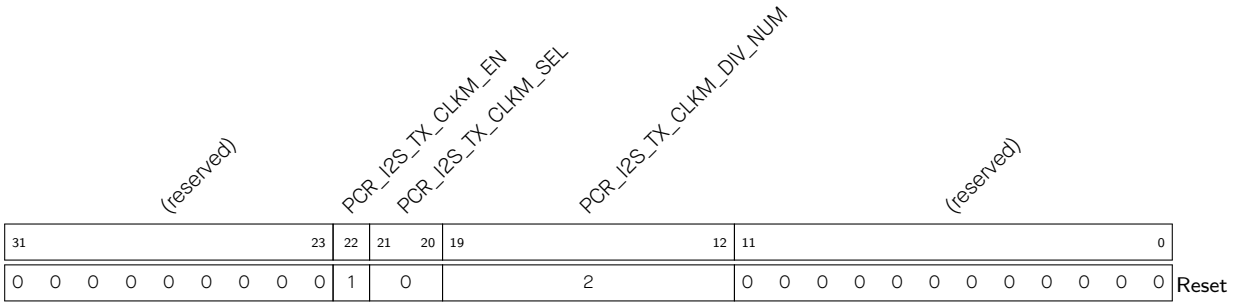
**PCR_I2S_TX_READY** Represents whether or not I2S TX is released from reset.

0: Not released

1: Released

(RO)

Register 8.24. PCR_I2S_TX_CLKM_CONF_REG (0x0064)




**PCR_I2S_TX_CLKM_DIV_NUM** Configures the integral I2S TX clock divider value. (R/W)

**PCR_I2S_TX_CLKM_SEL** Configures the clock source of I2S TX.

0 (default): XTAL_CLK

1: PLL_F120M_CLK

2: PLL_F160M_CLK

3: I2S_MCLK_in

(R/W)

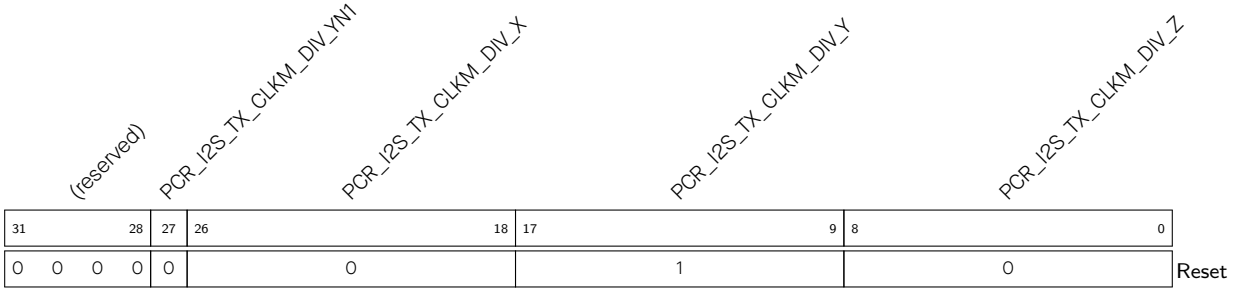
**PCR_I2S_TX_CLKM_EN** Configures whether or not to enable I2S TX functional clock.

0: Not enable

1: Enable

(R/W)

Register 8.25. PCR_I2S_TX_CLKM_DIV_CONF_REG (0x0068)



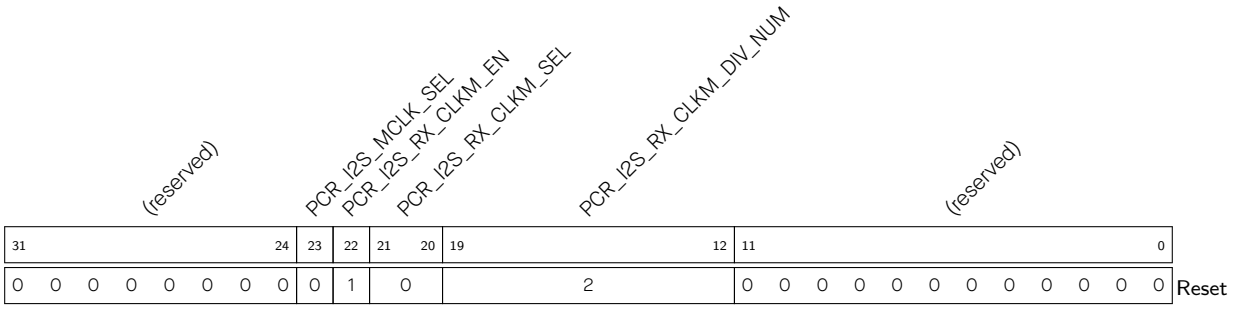

**PCR_I2S_TX_CLKM_DIV_Z** For b <= a/2, the value of I2S_TX_CLKM_DIV_Z is b. For b > a/2, the value of I2S_TX_CLKM_DIV_Z is (a-b). (R/W)

**PCR_I2S_TX_CLKM_DIV_Y** For b <= a/2, the value of I2S_TX_CLKM_DIV_Y is (a%b) . For b > a/2, the value of I2S_TX_CLKM_DIV_Y is (a%(a-b)). (R/W)

**PCR_I2S_TX_CLKM_DIV_X** For b <= a/2, the value of I2S_TX_CLKM_DIV_X is (a/b) - 1. For b > a/2, the value of I2S_TX_CLKM_DIV_X is (a/(a-b)) - 1. (R/W)

**PCR_I2S_TX_CLKM_DIV_YN1** For b <= a/2, the value of I2S_TX_CLKM_DIV_YN1 is 0 . For b > a/2, the value of I2S_TX_CLKM_DIV_YN1 is 1. (R/W)

Register 8.26. PCR_I2S_RX_CLKM_CONF_REG (0x006C)




**PCR_I2S_RX_CLKM_DIV_NUM** Configures the integral I2S clock divider value (R/W)

**PCR_I2S_RX_CLKM_SEL** Configures the clock source of I2S RX.

0 (default): XTAL_CLK

1: PLL_F120M_CLK

2: PLL_F160M_CLK

3: I2S_MCLK_in

(R/W)

**PCR_I2S_RX_CLKM_EN** Configures whether or not to enable I2S RX functional clock.

0: Not enable

1: Enable

(R/W)

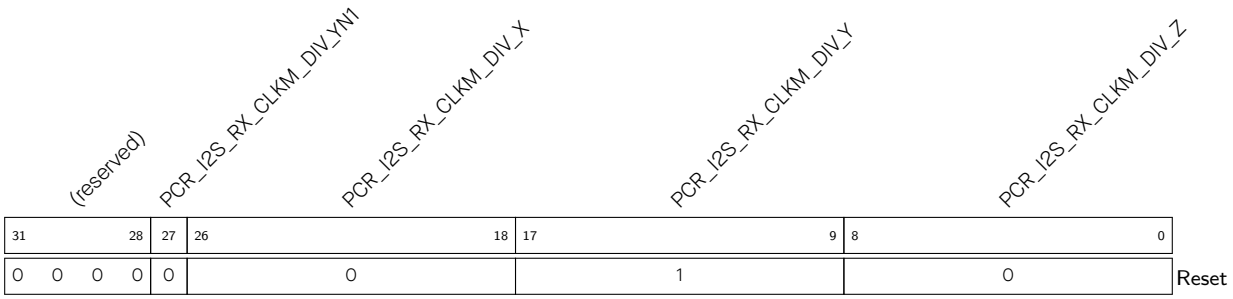
**PCR_I2S_MCLK_SEL** Configures to select master clock.

0 (default): I2S_TX_CLK

1: I2S_RX_CLK

(R/W)

Register 8.27. PCR_I2S_RX_CLKM_DIV_CONF_REG (0x0070)



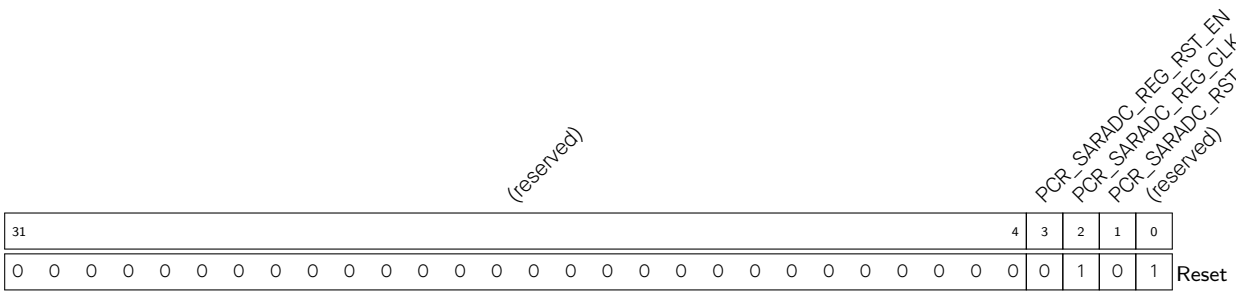

**PCR_I2S_RX_CLKM_DIV_Z** For b <= a/2, the value of I2S_RX_CLKM_DIV_Z is b. For b > a/2, the value of I2S_RX_CLKM_DIV_Z is (a-b). (R/W)

**PCR_I2S_RX_CLKM_DIV_Y** For b <= a/2, the value of I2S_RX_CLKM_DIV_Y is (a%b) . For b > a/2, the value of I2S_RX_CLKM_DIV_Y is (a%(a-b)). (R/W)

**PCR_I2S_RX_CLKM_DIV_X** For b <= a/2, the value of I2S_RX_CLKM_DIV_X is (a/b) - 1. For b > a/2, the value of I2S_RX_CLKM_DIV_X is (a/(a-b)) - 1. (R/W)

**PCR_I2S_RX_CLKM_DIV_YN1** For b <= a/2, the value of I2S_RX_CLKM_DIV_YN1 is 0 . For b > a/2, the value of I2S_RX_CLKM_DIV_YN1 is 1. (R/W)

Register 8.28. PCR_SARADC_CONF_REG (0x0074)




**PCR_SARADC_RST_EN** Configures whether or not to reset functional registers of SAR ADC.

0: Not reset

1: Reset

(R/W)

**PCR_SARADC_REG_CLK_EN** Configures whether or not to enable APB_CLK for SAR ADC.

0: Not enable

1: Enable

(R/W)

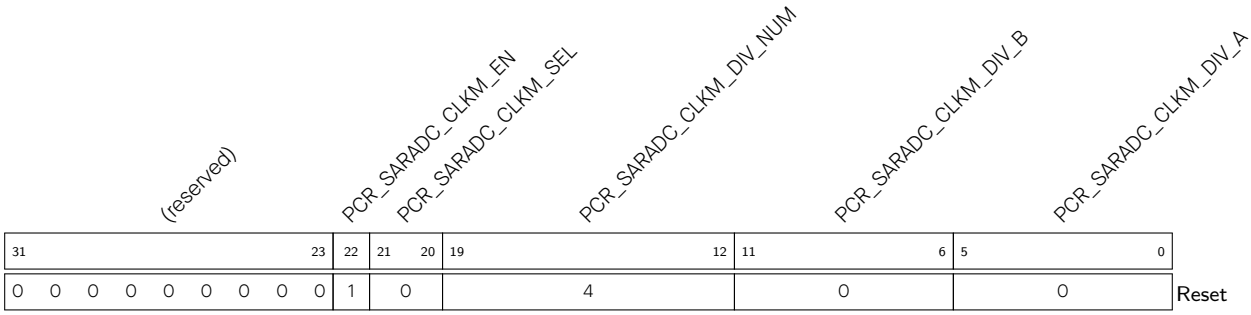
**PCR_SARADC_REG_RST_EN** Configures whether or not to reset APB registers of SAR ADC.

0: Not reset

1: Reset

(R/W)

Register 8.29. PCR_SARADC_CLKM_CONF_REG (0x0078)




**PCR_SARADC_CLKM_DIV_A** Configures the denominator of the frequency divider factor of the SAR ADC functional clock. (R/W)

**PCR_SARADC_CLKM_DIV_B** Configures the numerator of the frequency divider factor of the SAR ADC functional clock. (R/W)

**PCR_SARADC_CLKM_DIV_NUM** Configures the integral part of the frequency divider factor of the SAR ADC functional clock. (R/W)

**PCR_SARADC_CLKM_SEL** Configures the clock source of SAR ADC.

0 (default): XTAL_CLK

1: RC_FAST_CLK

2: PLL_F80M_CLK

(R/W)

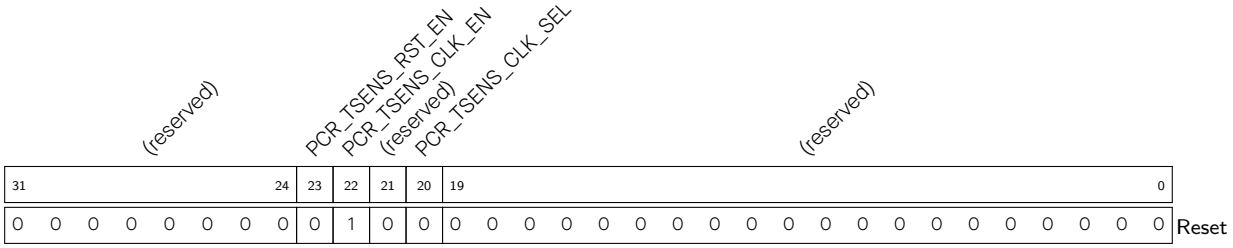
**PCR_SARADC_CLKM_EN** Configures whether or not to enable SAR ADC functional clock.

0: Not enable

1: Enable

(R/W)

Register 8.30. PCR_TSENS_CLK_CONF_REG (0x007C)




**PCR_TSENS_CLK_SEL** Configures the clock source of the temperature sensor.

0 (default): XTAL_CLK

1: RC_FAST_CLK

(R/W)

**PCR_TSENS_CLK_EN** Configures whether or not to enable the clock of the temperature sensor.

0: Not enable

1: Enable

(R/W)

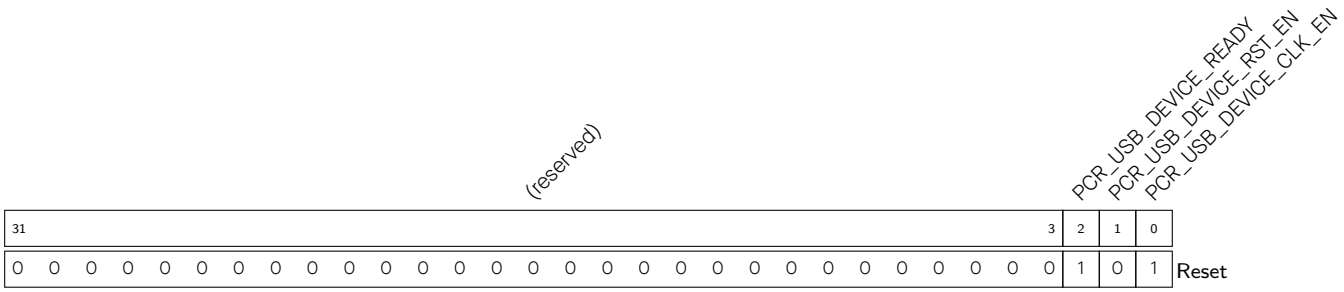
**PCR_TSENS_RST_EN** Configures whether or not to reset the temperature sensor.

0: Not reset

1: Reset

(R/W)

Register 8.31. PCR_USB_DEVICE_CONF_REG (0x0080)




**PCR_USB_DEVICE_CLK_EN** Configures whether or not to enable USB Serial/JTAG clock.

0: Not enable

1: Enable

(R/W)

**PCR_USB_DEVICE_RST_EN** Configures whether or not to reset USB Serial/JTAG.

0: Not reset

1: Reset

(R/W)

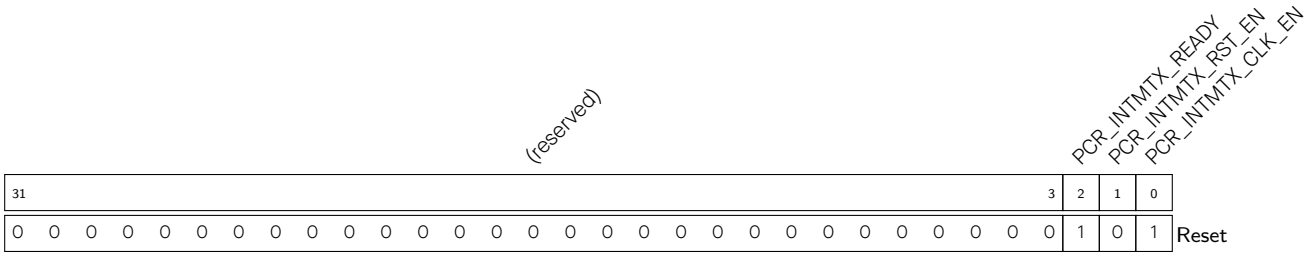
**PCR_USB_DEVICE_READY** Represents whether or not USB Serial/JTAG is released from reset.

0: Not released

1: Released

(RO)

Register 8.32. PCR_INTMTX_CONF_REG (0x0084)




**PCR_INTMTX_CLK_EN** Configures whether or not to enable Interrupt Matrix clock.

0: Not enable

1: Enable

(R/W)

**PCR_INTMTX_RST_EN** Configures whether or not to reset Interrupt Matrix.

0: Not reset

1: Reset

(R/W)

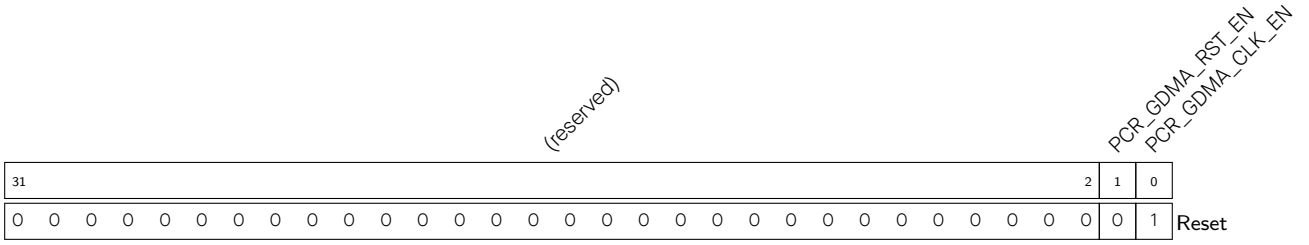
**PCR_INTMTX_READY** Represents whether or not Interrupt Matrix is released from reset.

0: Not released

1: Released

(RO)

Register 8.33. PCR_GDMA_CONF_REG (0x0090)




**PCR_GDMA_CLK_EN** Configures whether or not to enable GDMA clock.

0: Not enable

1: Enable

(R/W)

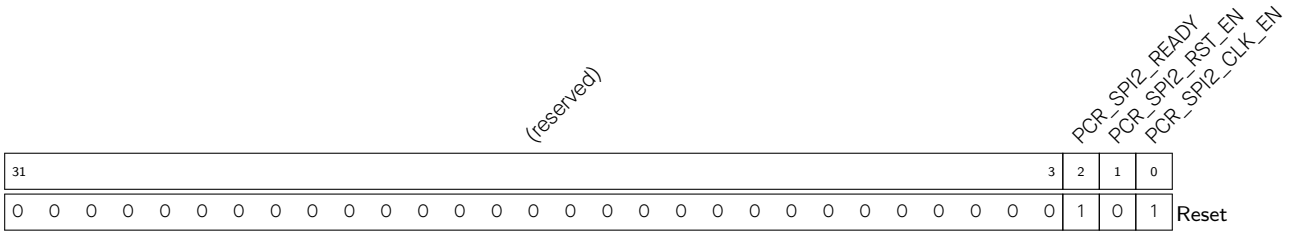
**PCR_GDMA_RST_EN** Configures whether or not to reset GDMA.

0: Not reset

1: Reset

(R/W)

Register 8.34. PCR_SPI2_CONF_REG (0x0094)




**PCR_SPI2_CLK_EN** Configures whether or not to enable APB_CLK for SPI2.

0: Not enable

1: Enable

(R/W)

**PCR_SPI2_RST_EN** Configures whether or not to reset SPI2.

0: Not reset

1: Reset

(R/W)

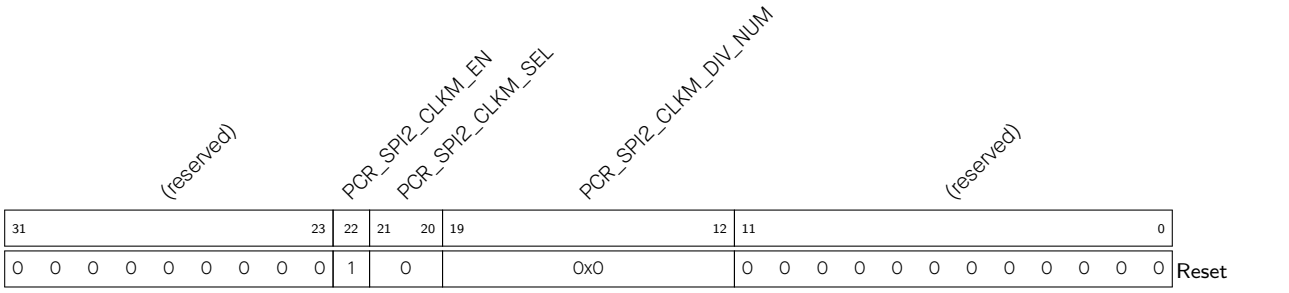
**PCR_SPI2_READY** Represents whether or not SPI2 is released from reset.

0: Not released

1: Released

(RO)

Register 8.35. PCR_SPI2_CLKM_CONF_REG (0x0098)




**PCR_SPI2_CLKM_DIV_NUM** Configures the integral part of the frequency divider factor of the spi2_mst clock. (R/W)

**PCR_SPI2_CLKM_SEL** Configures the clock source of SPI2.

0 (default): XTAL_CLK

1: PLL_F160M_CLK

2: RC_FAST_CLK

(R/W)

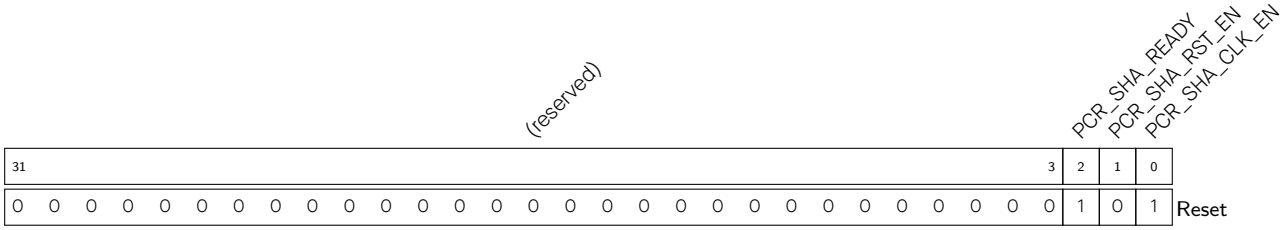
**PCR_SPI2_CLKM_EN** Configures whether or not to enable SPI2 functional clock.

0: Not enable

1: Enable

(R/W)

Register 8.36. PCR_SHA_CONF_REG (0x00A0)




**PCR_SHA_CLK_EN** Configures whether or not to enable SHA clock.

0: Not enable

1: Enable

(R/W)

**PCR_SHA_RST_EN** Configures whether or not to reset SHA.

0: Not reset

1: Reset

(R/W)

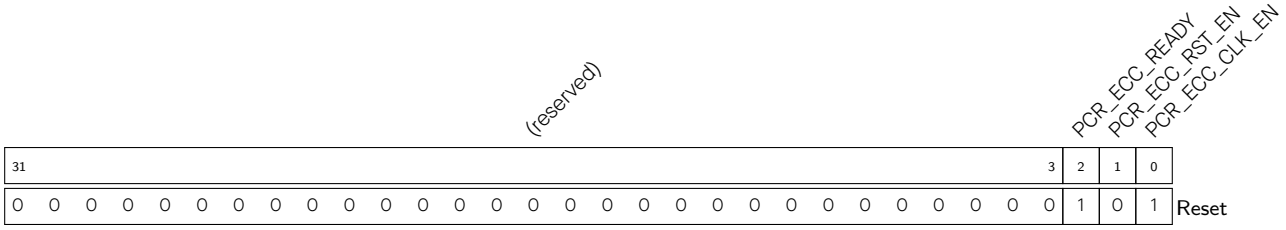
**PCR_SHA_READY** Represents whether or not SHA is released from reset.

0: Not released

1: Released

(RO)

Register 8.37. PCR_ECC_CONF_REG (0x00AC)




**PCR_ECC_CLK_EN** Configures whether or not to enable ECC clock.

0: Not enable

1: Enable

(R/W)

**PCR_ECC_RST_EN** Configures whether or not to reset ECC.

0: Not reset

1: Reset

(R/W)

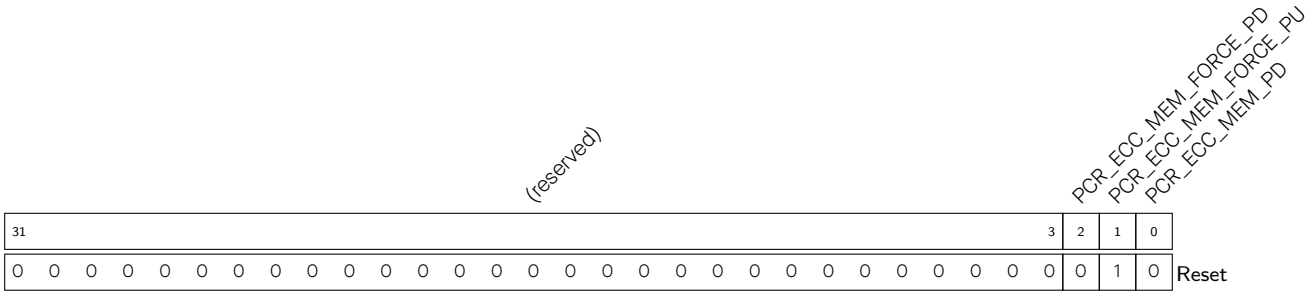
**PCR_ECC_READY** Represents whether or not ECC is released from reset.

0: Not released

1: Released

(RO)

Register 8.38. PCR_ECC_PD_CTRL_REG (0x00B0)




**PCR_ECC_MEM_PD** Configures whether or not to power down ECC internal memory.

0: Not power down

1: Power down

(R/W)

**PCR_ECC_MEM_FORCE_PU** Configures whether or not to force power up ECC internal memory.

0: Not force power up

1: Force power up

(R/W)

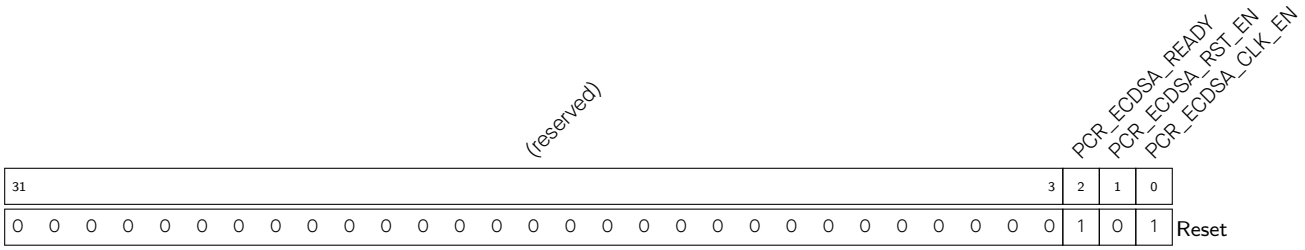
**PCR_ECC_MEM_FORCE_PD** Configures whether or not to force power down ECC internal memory.

0: Not force power down

1: Force power down

(R/W)

Register 8.39. PCR_ECDSA_CONF_REG (0x00BC)




**PCR_ECDSA_CLK_EN** Configures whether or not to enable ECDSA_DS clock.

0: Not enable

1: Enable

(R/W)

**PCR_ECDSA_RST_EN** Configures whether or not to reset ECDSA_DS.

0: Not reset

1: Reset

(R/W)

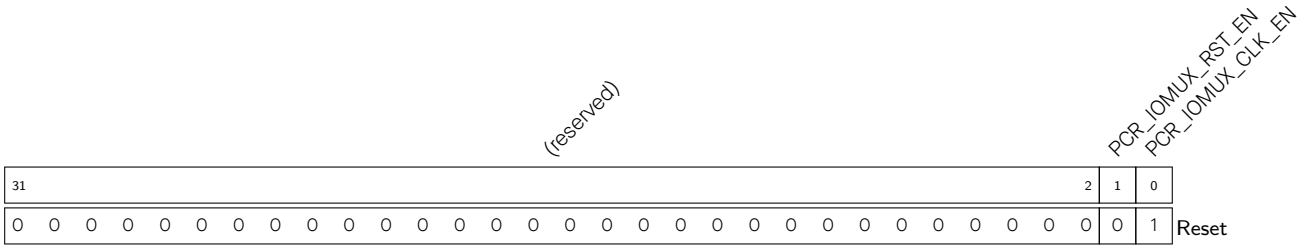
**PCR_ECDSA_READY** Represents whether or not ECDSA_DS is released from reset.

0: Not released

1: Released

(RO)

Register 8.40. PCR_IOMUX_CONF_REG (0x00C0)




**PCR_IOMUX_CLK_EN** Configures whether or not to enable APB_CLK for IO MUX.

0: Not enable

1: Enable

(R/W)

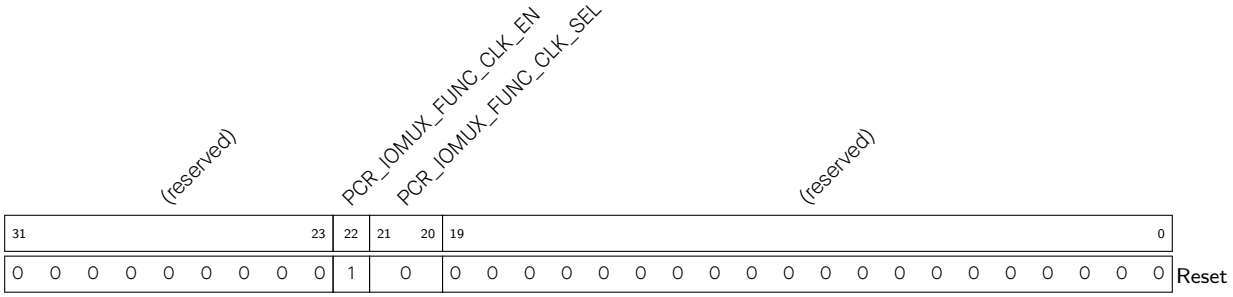
**PCR_IOMUX_RST_EN** Configures whether or not to reset IO MUX.

0: Not reset

1: Reset

(R/W)

Register 8.41. PCR_IOMUX_CLK_CONF_REG (0x00C4)




**PCR_IOMUX_FUNC_CLK_SEL** Configures the clock source of IO MUX.

0 (default): XTAL_CLK

1: RC_FAST_CLK

2: PLL_F80M_CLK

(R/W)

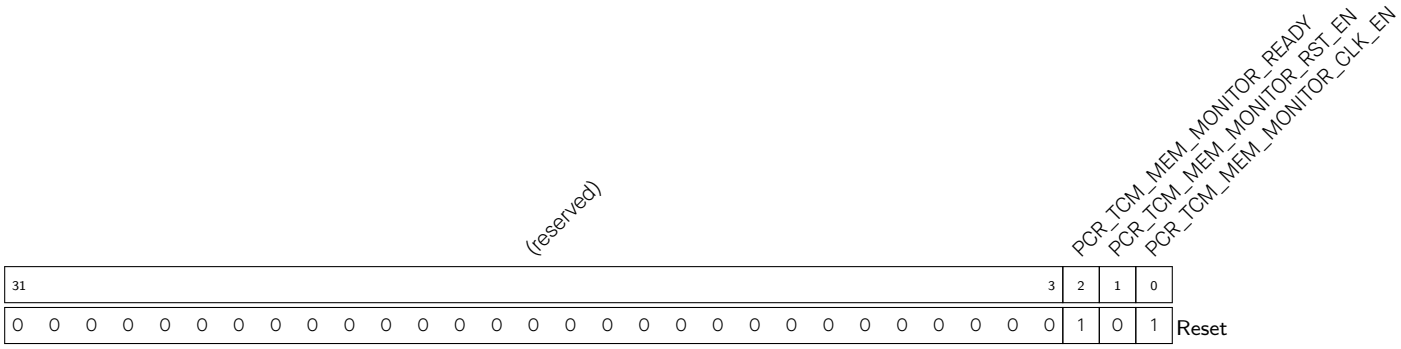
**PCR_IOMUX_FUNC_CLK_EN** Configures whether or not to enable IO MUX functional clock.

0: Not enable

1: Enable

(R/W)

Register 8.42. PCR_TCM_MEM_MONITOR_CONF_REG (0x00C8)




**PCR_TCM_MEM_MONITOR_CLK_EN** Configures whether or not to enable TCM_MEM_MONITOR clock.

0: Not enable

1: Enable

(R/W)

**PCR_TCM_MEM_MONITOR_RST_EN** Configures whether or not to reset TCM_MEM_MONITOR.

0: Not reset

1: Reset

(R/W)

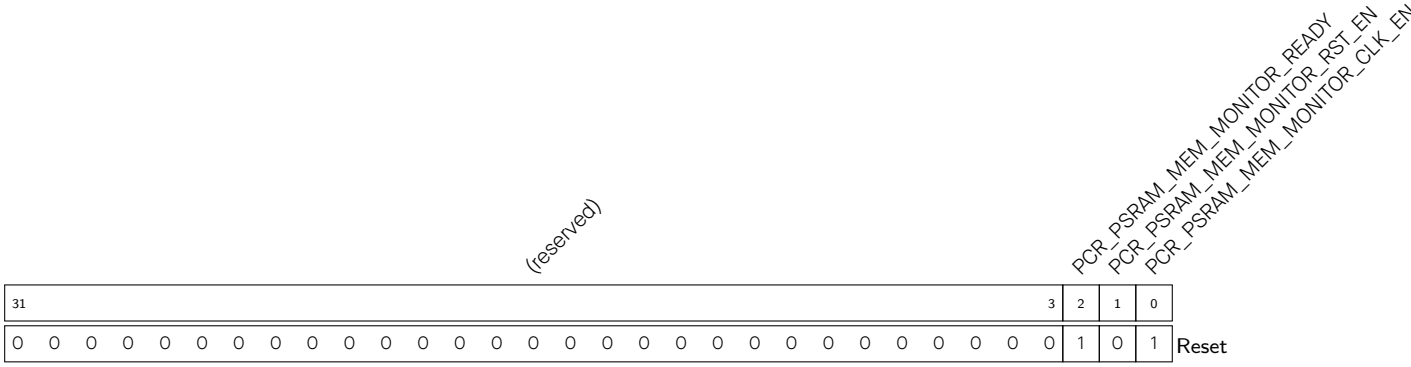
**PCR_TCM_MEM_MONITOR_READY** Represents whether or not TCM_MEM_MONITOR is released from reset.

0: Not released

1: Released

(RO)

Register 8.43. PCR_PSRAM_MEM_MONITOR_CONF_REG (0x00CC)




**PCR_PSRAM_MEM_MONITOR_CLK_EN** Configures whether or not to enable PSRAM_MEM_MONITOR clock.

0: Not enable

1: Enable

(R/W)

**PCR_PSRAM_MEM_MONITOR_RST_EN** Configures whether or not to reset PSRAM_MEM_MONITOR.

0: Not reset

1: Reset

(R/W)

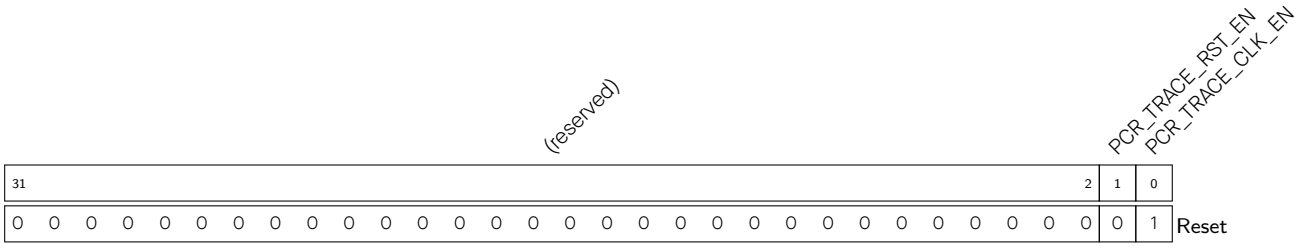
**PCR_PSRAM_MEM_MONITOR_READY** Represents whether or not PSRAM_MEM_MONITOR is released from reset.

0: Not released

1: Released

(RO)

Register 8.44. PCR_TRACE_CONF_REG (0x00D0)




**PCR_TRACE_CLK_EN** Configures whether or not to enable the clock of RISC-V Trace Encoder.

0: Not enable

1: Enable

(R/W)

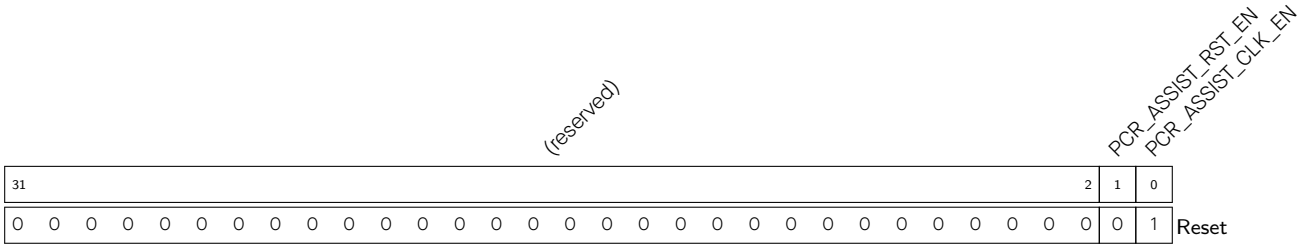
**PCR_TRACE_RST_EN** Configures whether or not to reset RISC-V Trace Encoder.

0: Not reset

1: Reset

(R/W)

Register 8.45. PCR_ASSIST_CONF_REG (0x00D4)




**PCR_ASSIST_CLK_EN** Configures whether or not to enable Debug Assistant clock.

0: Not enable

1: Enable

(R/W)

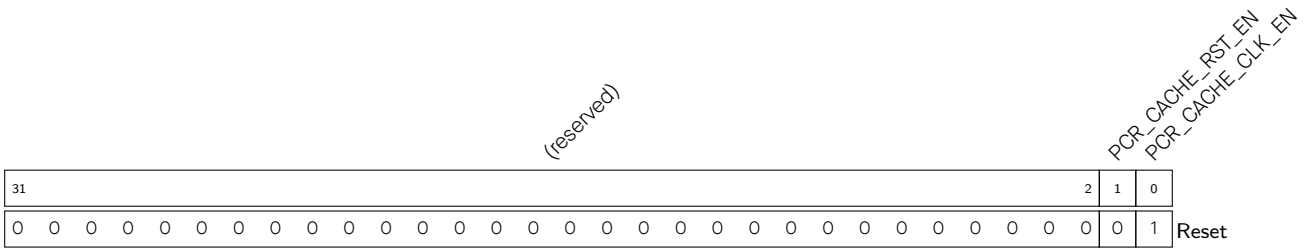
**PCR_ASSIST_RST_EN** Configures whether or not to reset Debug Assistant.

0: Not reset

1: Reset

(R/W)

Register 8.46. PCR_CACHE_CONF_REG (0x00D8)




**PCR_CACHE_CLK_EN** Configures whether or not to enable Cache clock.

0: Not enable

1: Enable

(R/W)

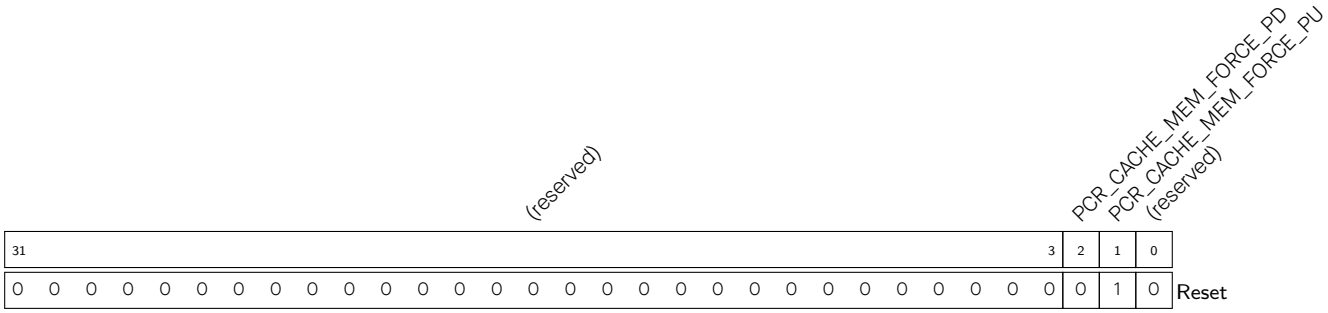
**PCR_CACHE_RST_EN** Configures whether or not to reset Cache.

0: Not reset

1: Reset

(R/W)

Register 8.47. PCR_CACHE_PD_CTRL_REG (0x00DC)




**PCR_CACHE_MEM_FORCE_PU** Configures whether or not to force power up Cache internal memory.

0: Not force power up

1: Force power up

(R/W)

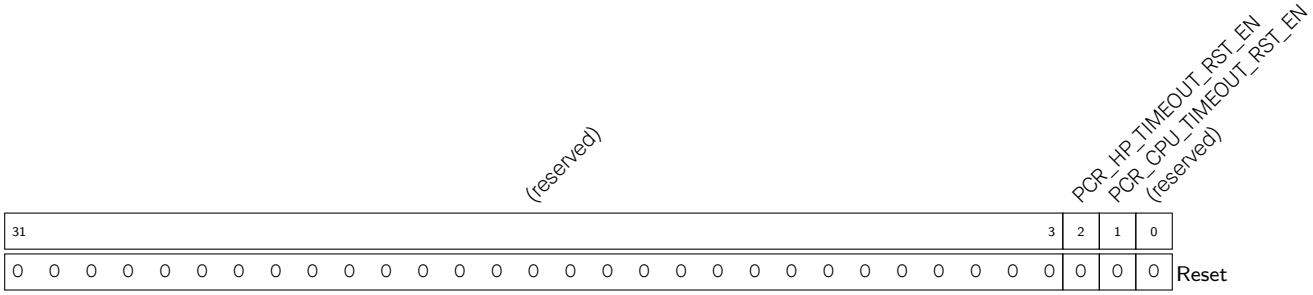
**PCR_CACHE_MEM_FORCE_PD** Configures whether or not to force power down Cache internal memory.

0: Not force power down

1: Force power down

(R/W)

Register 8.48. PCR_TIMEOUT_CONF_REG (0x00E4)




**PCR_CPU_TIMEOUT_RST_EN** Configures whether or not to reset CPU Peripheral Timeout Protection.

0: Not reset

1: Reset

(R/W)

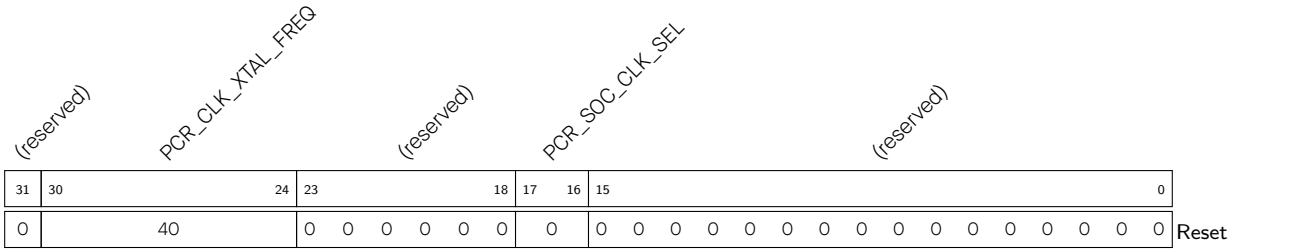
**PCR_HP_TIMEOUT_RST_EN** Configures whether or not to reset HP Peripheral Timeout Protection.

0: Not reset

1: Reset

(R/W)

Register 8.49. PCR_SYSCLK_CONF_REG (0x00E8)




**PCR_SOC_CLK_SEL** Configures to select the clock source of HP_ROOT_CLK.

0 (default): XTAL_CLK

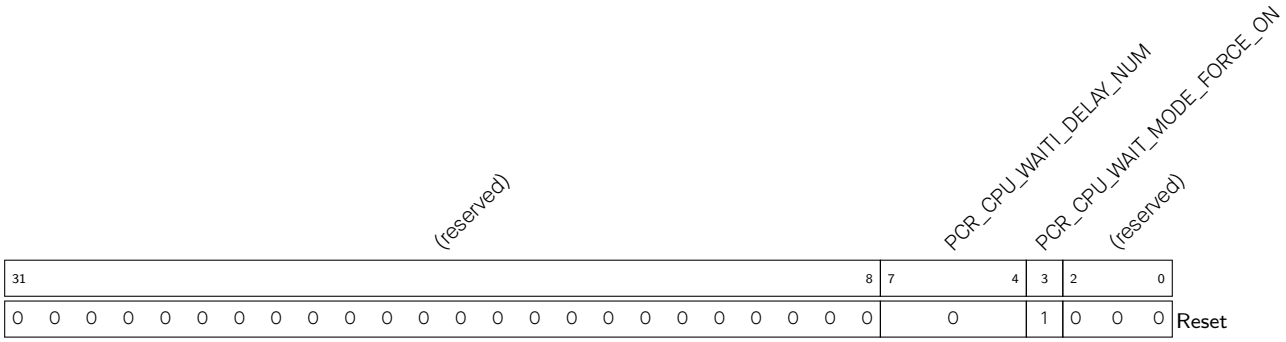
1: RC_FAST_CLK

2: PLL_F160M_CLK

(R/W)

**PCR_CLK_XTAL_FREQ** This field indicates the frequency (MHz) of XTAL_CLK. (RO)

Register 8.50. PCR_CPU_WAITI_CONF_REG (0x00EC)




**PCR_CPU_WAIT_MODE_FORCE_ON** Configures whether or not to force on the gated CPU clock when CPU is in WFI (wait for interrupt) mode.

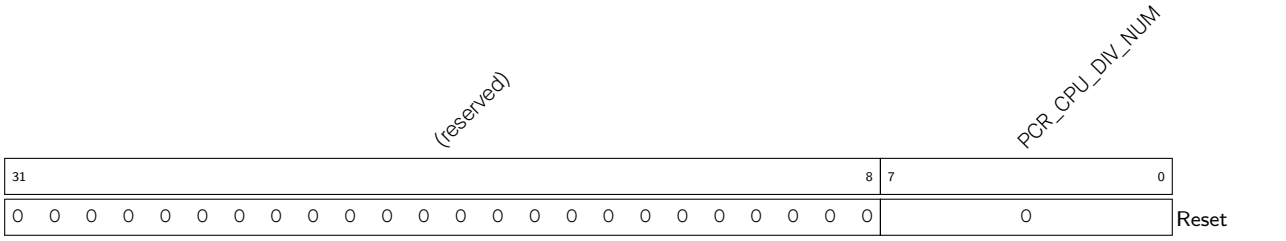
0: Not force enable

1: Force enable

Usually, after executing the WFI instruction, CPU enters the WFI mode, during which the gated CPU clock is turned off until any interrupts occur. In this way, power consumption is saved. If this bit is set, the gated CPU clock is always on and will not be turned off by the WFI instruction. (R/W)

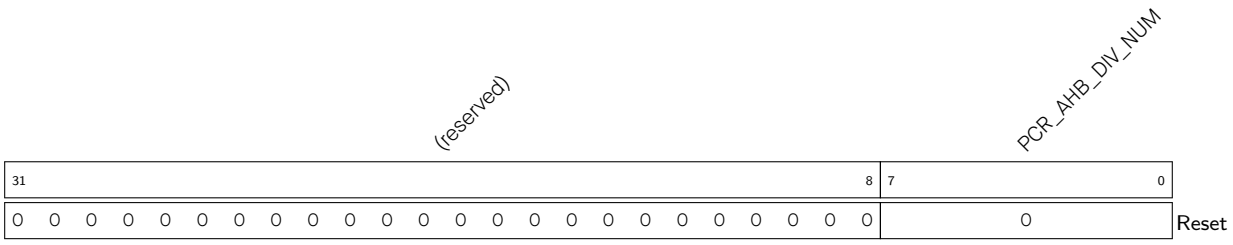
**PCR_CPU_WAITI_DELAY_NUM** Configures the number of delay cycles to turn off the CPU clock after the CPU enters the WFI mode because of WFI

Register 8.51. PCR_CPU_FREQ_CONF_REG (0x00F0)



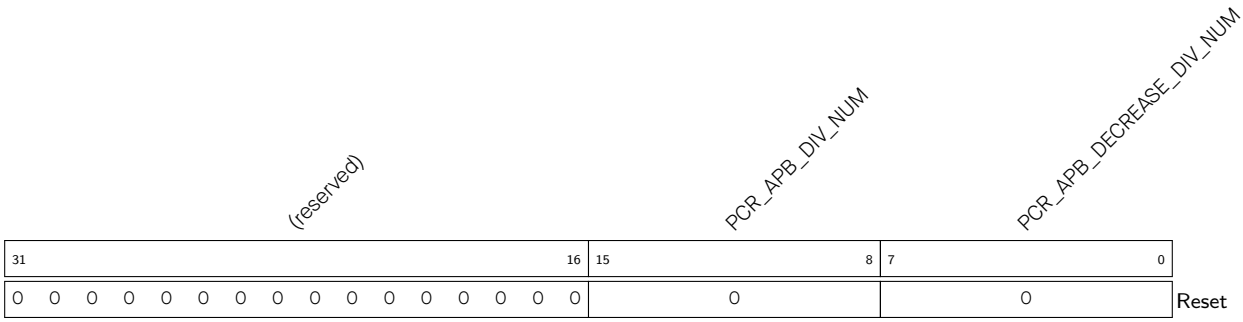

**PCR_CPU_DIV_NUM** Configures the divisor of HP_ROOT_CLK to generate CPU_CLK. This field should be used together with PCR_AHB_DIV_NUM. (R/W)

Register 8.52. PCR_AHB_FREQ_CONF_REG (0x00F4)




**PCR_AHB_DIV_NUM** Configures the divisor of HP_ROOT_CLK to generate AHB_CLK. This field should be used together with PCR_CPU_DIV_NUM.
(R/W)

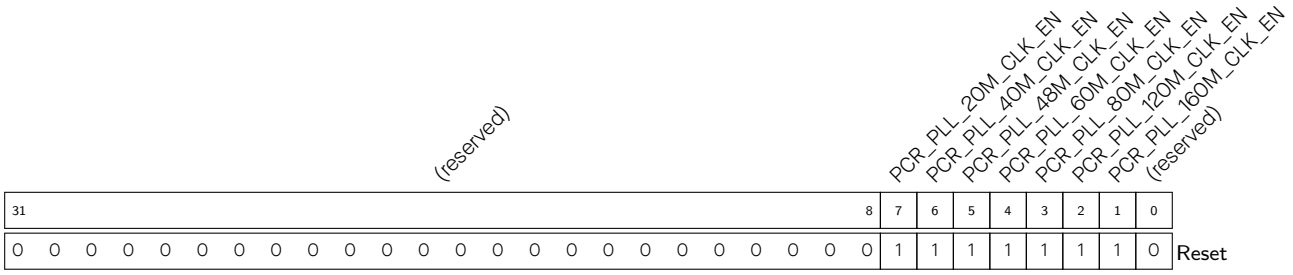
Register 8.53. PCR_APB_FREQ_CONF_REG (0x00F8)




**PCR_APB_DECREASE_DIV_NUM** Configures the divisor AHB_CLK to generate APB_CLK during the first division.
(R/W)

**PCR_APB_DIV_NUM** Configures the divisor of AHB_CLK to generate APB_CLK during the second division.
(R/W)

Register 8.54. PCR_PLL_DIV_CLK_EN_REG (0x0100)




**PCR_PLL_160M_CLK_EN** Configures whether or not to enable PLL_F160M_CLK.

0: Not enable

1 (default): Enable

(R/W)

**PCR_PLL_120M_CLK_EN** Configures whether or not to enable PLL_F120M_CLK.

0: Not enable

1 (default): Enable

(R/W)

**PCR_PLL_80M_CLK_EN** Configures whether or not to enable PLL_F80M_CLK.

0: Not enable

1 (default): Enable

(R/W)

**PCR_PLL_60M_CLK_EN** Configures whether or not to enable PLL_F60M_CLK.

0: Not enable

1 (default): Enable

(R/W)

**PCR_PLL_48M_CLK_EN** Configures whether or not to enable PLL_F48M_CLK.

0: Not enable

1 (default): Enable

(R/W)

**PCR_PLL_40M_CLK_EN** Configures whether or not to enable PLL_F40M_CLK.

0: Not enable

1 (default): Enable

(R/W)

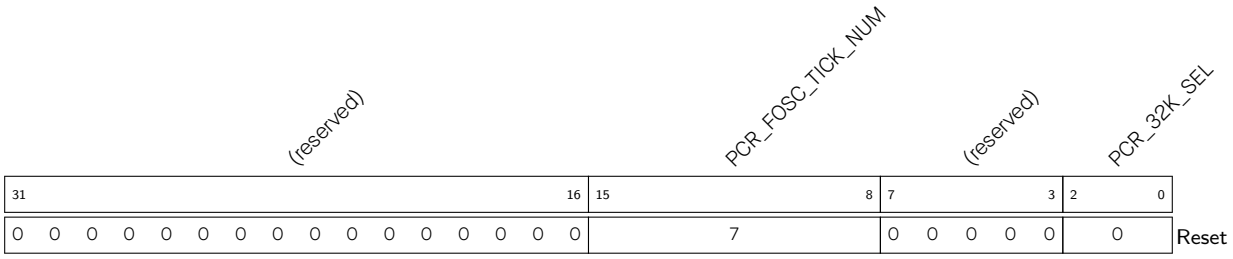
**PCR_PLL_20M_CLK_EN** Configures whether or not to enable PLL_F20M_CLK.

0: Not enable

1 (default): Enable

(R/W)

Register 8.55. PCR_CTRL_32K_CONF_REG (0x010C)




**PCR_32K_SEL** Configures the 32 kHz clock for TIMER_GROUP.

0: reserved

1: XTAL32K_CLK

2: OSC_SLOW_CLK

3: RC_SLOW_CLK

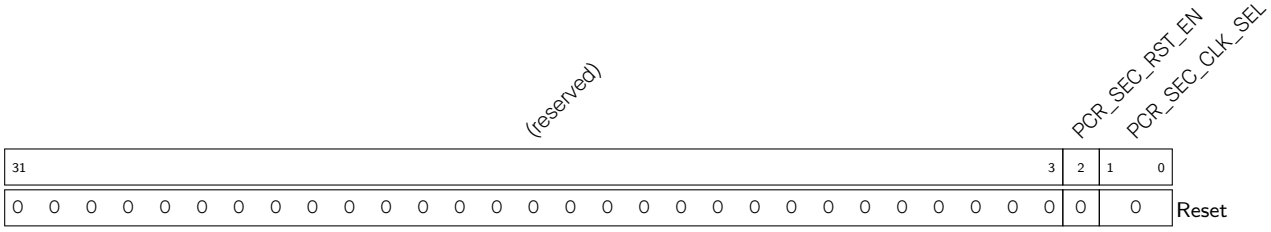
4: Clock from RC_FAST_CLK after division by PCR_FOSC_TICK_NUM

(R/W)

**PCR_FOSC_TICK_NUM** Configure the division factor for the RC_FAST_CLK to enter the calibration module. (Effective when PCR_32K_SEL is set to 4).

(R/W)

Register 8.56. PCR_SEC_CONF_REG (0x0118)




**PCR_SEC_CLK_SEL** Configures the clock source for the External Memory Encryption and Decryption module.

0 (default): XTAL_CLK

1: RC_FAST_CLK

2: PLL_F480M_CLK

(R/W)

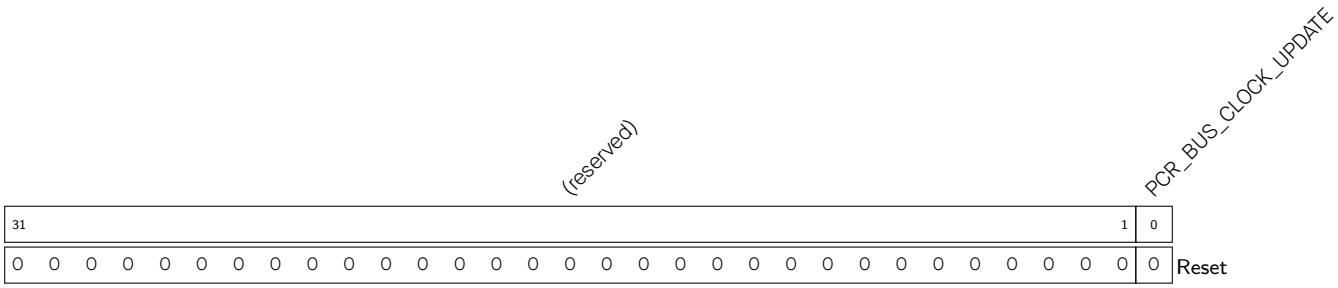
**PCR_SEC_RST_EN** Configures whether or not to reset the SEC.

0: Not reset

1: Reset

(R/W)

Register 8.57. PCR_BUS_CLK_UPDATE_REG (0x0120)




**PCR_BUS_CLOCK_UPDATE** Configures whether or not to update configurations for CPU_CLK division, AHB_CLK division and HP_ROOT_CLK clock source selection.

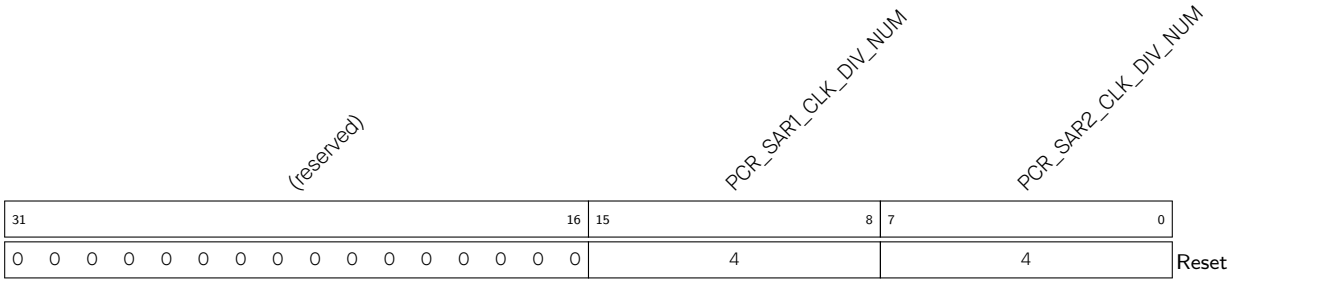
0: Not update configurations

1: Update configurations

This bit is automatically cleared when configurations have been updated.

(R/W/WTC)

Register 8.58. PCR_SAR_CLK_DIV_REG (0x0124)



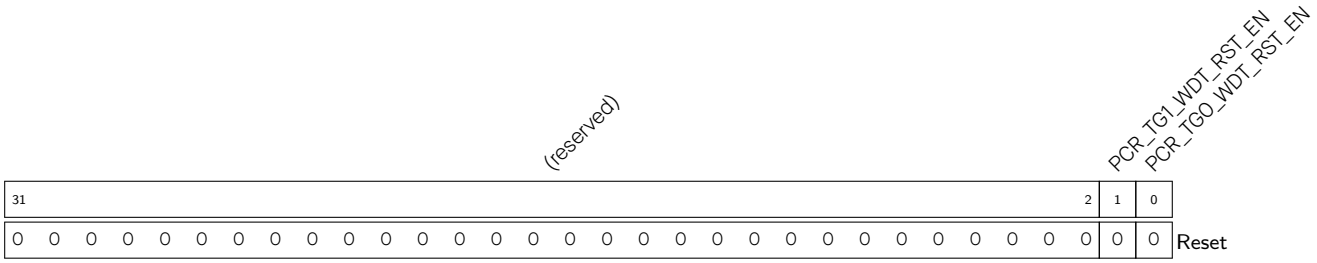

**PCR_SAR2_CLK_DIV_NUM** Configures the divisor for SAR ADC2 clock to generate ADC analog control signals.

(R/W)

**PCR_SAR1_CLK_DIV_NUM** Configures the divisor for SAR ADC1 clock to generate ADC analog control signals.

(R/W)

Register 8.59. PCR_TIMERGROUP_WDT_CONF_REG (0x012C)




**PCR_TG0_WDT_RST_EN** Configures whether or not to reset the watchdog of Timer Group 0.

0: Not reset

1: Reset

(R/W)

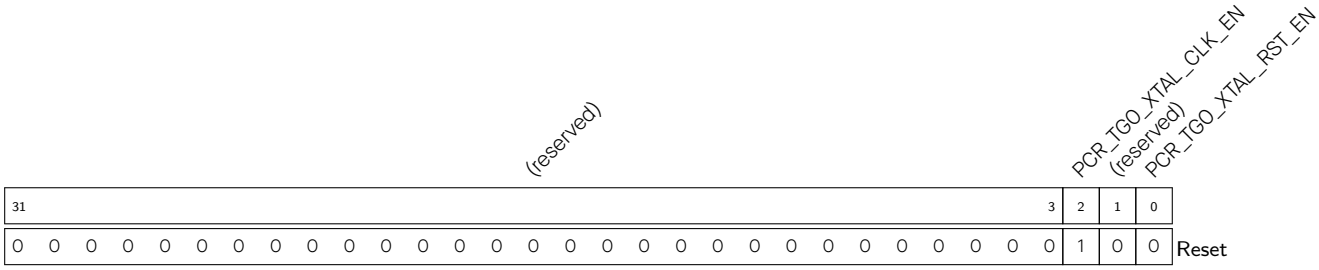
**PCR_TG1_WDT_RST_EN** Configures whether or not to reset the watchdog of Timer Group 1.

0: Not reset

1: Reset

(R/W)

Register 8.60. PCR_TIMERGROUP_XTAL_CONF_REG (0x0130)




**PCR_TG0_XTAL_RST_EN** Configures whether to reset the clock calibration function of the Timer Group 0.

0: Not reset

1: Reset

(R/W)

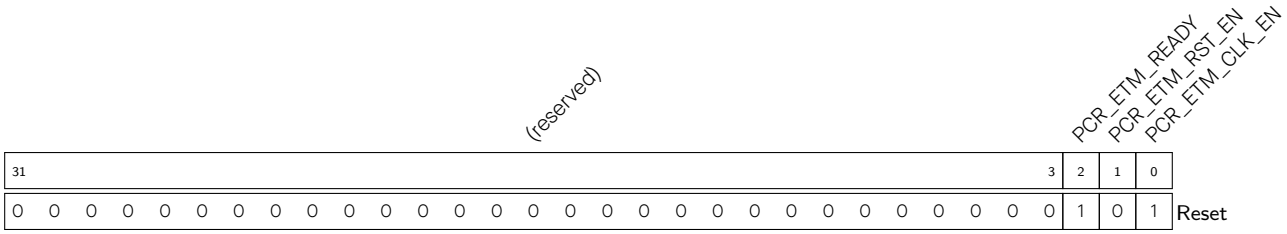
**PCR_TG0_XTAL_CLK_EN** Configures whether to enable the clock calibration function of the Timer Group 0.

0: Not enable

1: Enable

(R/W)

Register 8.61. PCR_ETM_CONF_REG (0x013C)




**PCR_ETM_CLK_EN** onfigures whether or not to enable ETM clock.

0: Not enable

1: Enable

(R/W)

**PCR_ETM_RST_EN** Configures whether or not to reset ETM.

0: Reset

1: Not reset

(R/W)

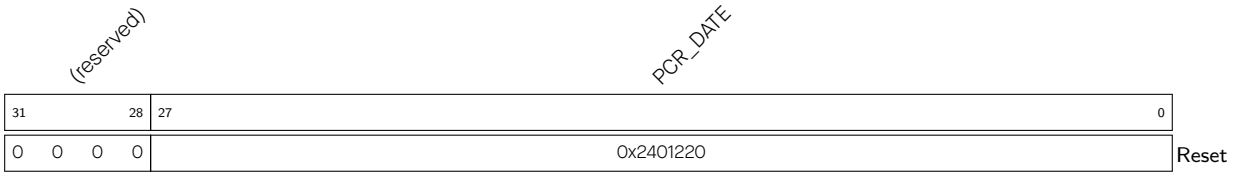
**PCR_ETM_READY** Represents whether or not ETM is released from reset.

0: Not released

1: Released

(RO)

Register 8.62. PCR_DATE_REG (0x0FFC)

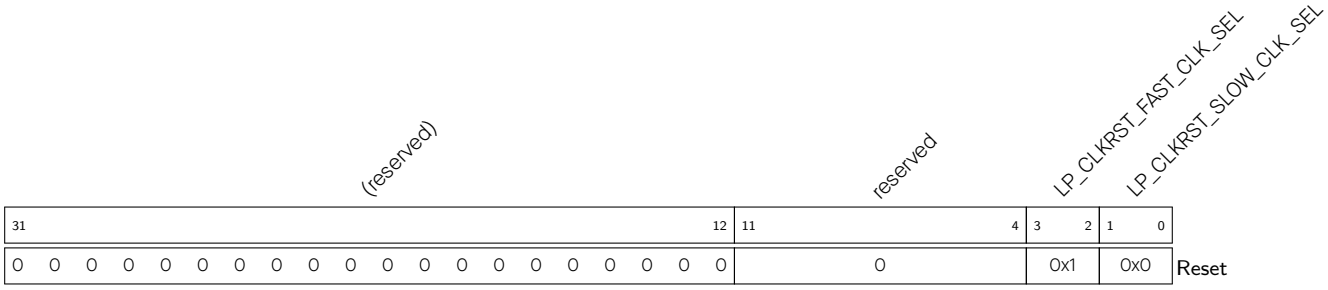



**PCR_DATE** Version control register. (R/W)

## 8.5.2 LP System Clock Registers

The addresses of the last two registers with the LPPERI prefix in this section are relative to the Low-power Peripheral Register (LPPERI) base address. The addresses of the third and fourth to last registers with the LP_AON prefix in this section are relative to the Low-power Always-on Register (LP_AON) base address.The other addresses in this section are relative to the Low-power Clock/Reset Register (LP_CLKRST) base address. For base address, please refer to Table 5.3-2 in Chapter 5 *System and Memory*.

Register 8.63. LP_CLKRST_LP_CLK_CONF_REG (0x0000)




**LP_CLKRST_SLOW_CLK_SEL** Configures the source of LP_SLOW_CLK.

0: RC_SLOW_CLK

1: XTAL32K_CLK

2: reserved

3: OSC_SLOW_CLK

(R/W)

**LP_CLKRST_FAST_CLK_SEL** Configures the source of LP_FAST_CLK.

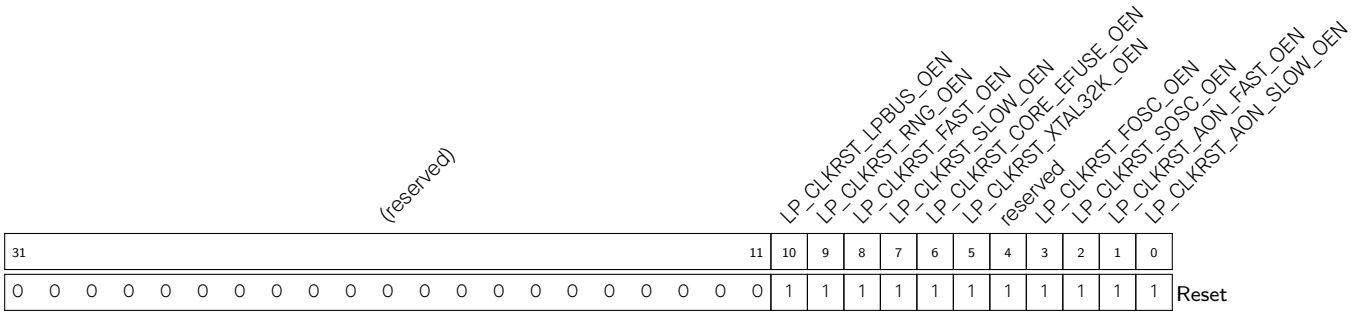
0: RC_FAST_CLK

1: XTAL_D2_CLK

2: XTAL_CLK

(R/W)

Register 8.64. LP_CLKRST_LP_CLK_PO_EN_REG (0x0004)




**LP_CLKRST_AON_SLOW_OEN** Configures whether to gate the LP_DYN_SLOW_CLK signals to pad.

0: Disable the clock gate

1: Enable the clock gate

(R/W)

**LP_CLKRST_AON_FAST_OEN** Configures whether to gate the LP_DYN_FAST_CLK signals to pad.

0: Disable the clock gate

1: Enable the clock gate

(R/W)

**LP_CLKRST_SOSC_OEN** Configures whether to gate the OSC_SLOW_CLK signals to pad.

0: Disable the clock gate

1: Enable the clock gate

(R/W)

**LP_CLKRST_FOSC_OEN** Configures whether to gate the RC_FAST_CLK signals to pad.

0: Disable the clock gate

1: Enable the clock gate

(R/W)

**LP_CLKRST_XTAL32K_OEN** Configures whether to gate the XTAL32K_CLK signals to pad.

0: Disable the clock gate

1: Enable the clock gate

(R/W)

**LP_CLKRST_CORE_EFUSE_OEN** Configures whether to gate the EFUSE_CTRL clock.

0: Disable the clock gate

1: Enable the clock gate

(R/W)

**LP_CLKRST_SLOW_OEN** Configures whether to gate the LP_SLOW_CLK signals to pad.

0: Disable the clock gate

1: Enable the clock gate

(R/W)

**LP_CLKRST_FAST_OEN** Configures whether to gate the LP_FAST_CLK signals to pad.

0: Disable the clock gate

1: Enable the clock gate

(R/W)

**Continued on the next page...**

Register 8.64. LP_CLKRST_LP_CLK_PO_EN_REG (0x0004)

Continued from the previous page...

**LP_CLKRST_RNG_OEN** Configures whether to gate the RNG clock signals to pad.

0: Disable the clock gate

1: Enable the clock gate

(R/W)

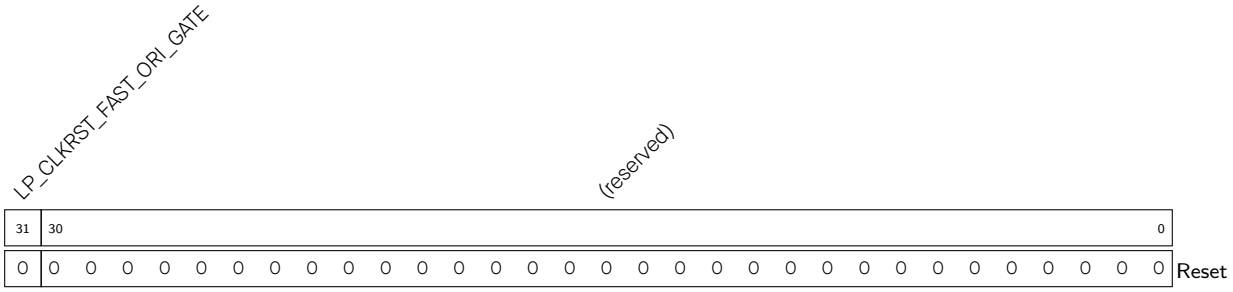
**LP_CLKRST_LPBUS_OEN** Configures whether to gate the LP bus clock signals to pad.

0: Disable the clock gate

1: Enable the clock gate

(R/W)

Register 8.65. LP_CLKRST_LP_CLK_EN_REG (0x0008)



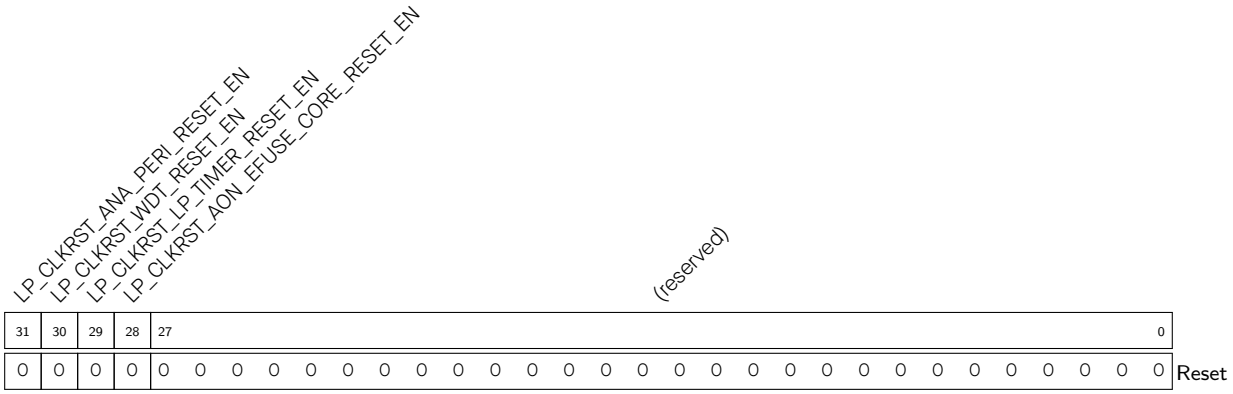

**LP_CLKRST_FAST_ORI_GATE** Configures the clock gate to LP_FAST_CLK.

0: Invalid. The clock gate controlled by hardware FSM

1: Force the clock to pass the clock gate

(R/W)

Register 8.66. LP_CLKRST_LP_RST_EN_REG (0x000C)




**LP_CLKRST_AON_EFUSE_CORE_RESET_EN** Configures whether to reset the always-on part of EFUSE_CTRL.

0: Invalid. No effect

1: Reset

(R/W)

**LP_CLKRST_LP_TIMER_RESET_EN** Configures whether to reset RTC timer.

0: Invalid. No effect

1: Reset

(R/W)

**LP_CLKRST_WDT_RESET_EN** Configures whether to reset RWDT and Super Watchdog Timer.

0: Invalid. No effect

1: Reset

(R/W)

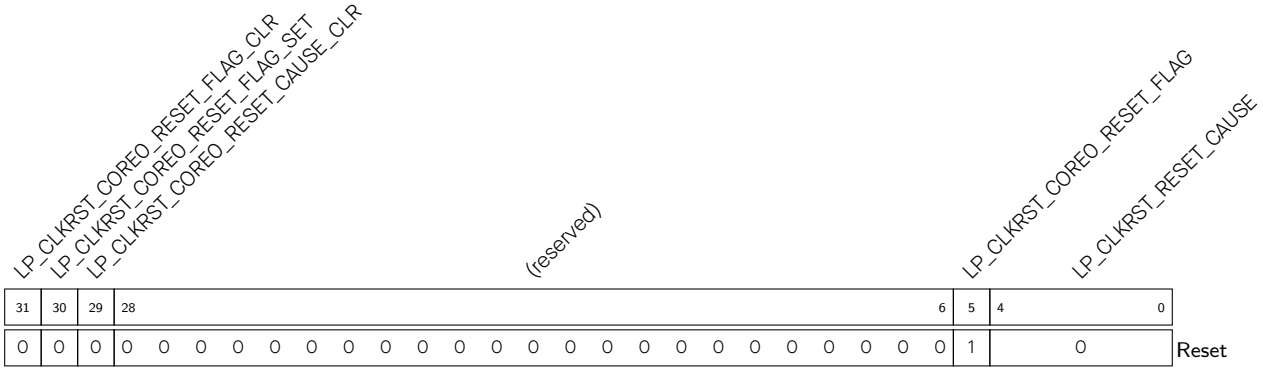
**LP_CLKRST_ANA_PERI_RESET_EN** Configures whether to reset analog peripherals, including the Brownout Detector.

0: Invalid. No effect

1: Reset

(R/W)

Register 8.67. LP_CLKRST_RESET_CAUSE_REG (0x0010)




**LP_CLKRST_RESET_CAUSE** Represents the reset cause. (RO)

**LP_CLKRST_COREO_RESET_FLAG** Represents whether the reset occurred is recorded in LP_CLKRST_RESET_CAUSE.

0: Illegal reset

1: Recorded reset

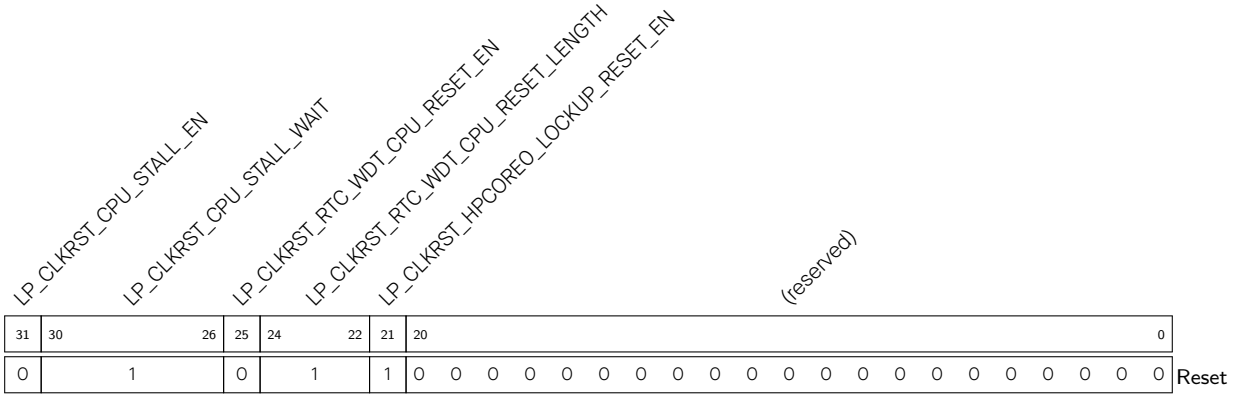
(RO)

**LP_CLKRST_COREO_RESET_CAUSE_CLR** Write 1 to clear LP_CLKRST_RESET_CAUSE. (WT)

**LP_CLKRST_COREO_RESET_FLAG_SET** Write 1 to set LP_CLKRST_COREO_RESET_FLAG. (WT)

**LP_CLKRST_COREO_RESET_FLAG_CLR** Write 1 to clear LP_CLKRST_COREO_RESET_FLAG. (WT)

Register 8.68. LP_CLKRST_CPU_RESET_REG (0x0014)




**LP_CLKRST_HPCORE0_LOCKUP_RESET_EN** Configures the lockup reset enable setting for HP core0.

0: Disable

1: Enable (R/W)

**LP_CLKRST_RTC_WDT_CPU_RESET_LENGTH** Configures the length of RWDT CPU reset. Measurement unit: LP_DYN_FAST_CLK clock cycles.

(R/W)

**LP_CLKRST_RTC_WDT_CPU_RESET_EN** Configures whether or not to enable RWDT CPU reset.

0: Disable RWDT CPU reset.

1: Enable RWDT CPU reset.

(R/W)

**LP_CLKRST_CPU_STALL_WAIT** Configure the time interval between CPU stall and reset. Measurement unit: LP_DYN_FAST_CLK clock cycles.

(R/W)

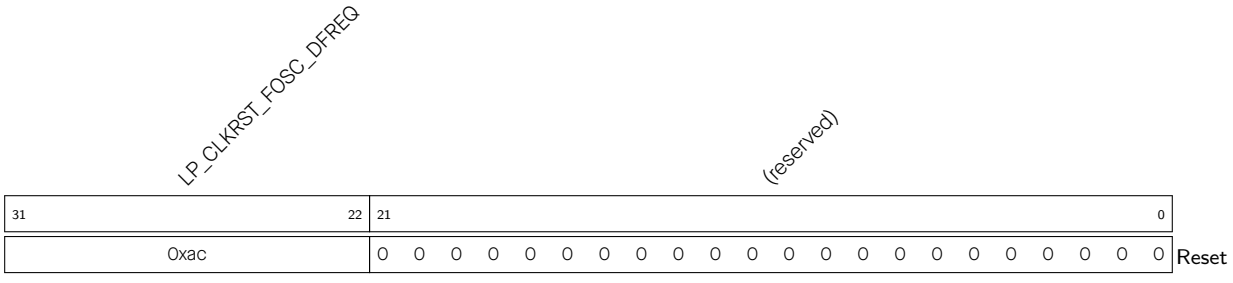
**LP_CLKRST_CPU_STALL_EN** Configures whether or not CPU will stall before RWDT and software reset CPU.

0: CPU will not stall.

1: CPU will stall.

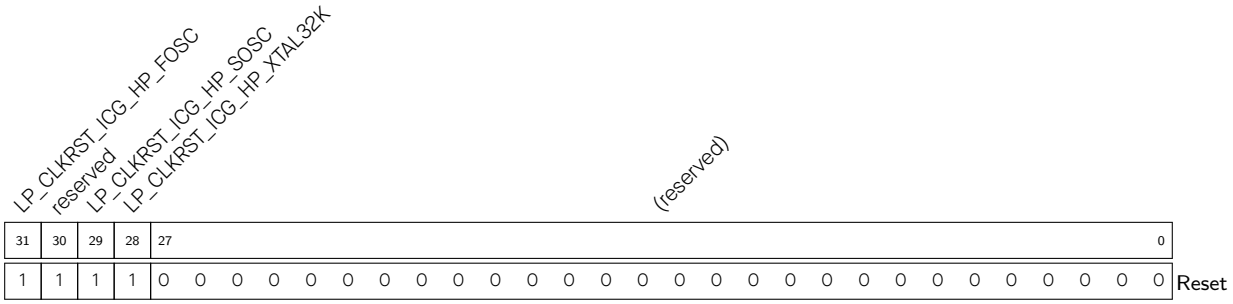
(R/W)

Register 8.69. LP_CLKRST_FOSC_CNTL_REG (0x0018)




**LP_CLKRST_FOSC_DFREQ** Configures the frequency of RC_FAST_CLK frequency. (R/W)

Register 8.70. LP_CLKRST_CLK_TO_HP_REG (0x0020)




**LP_CLKRST_ICG_HP_XTAL32K** Configures whether to gate the XTAL32K_CLK signals to HP system.

0: Disable the clock gate

1: Enable the clock gate

(R/W)

**LP_CLKRST_ICG_HP_SOSC** Configures whether to gate the RC_SLOW_CLK signals to HP system.

0: Disable the clock gate

1: Enable the clock gate

(R/W)

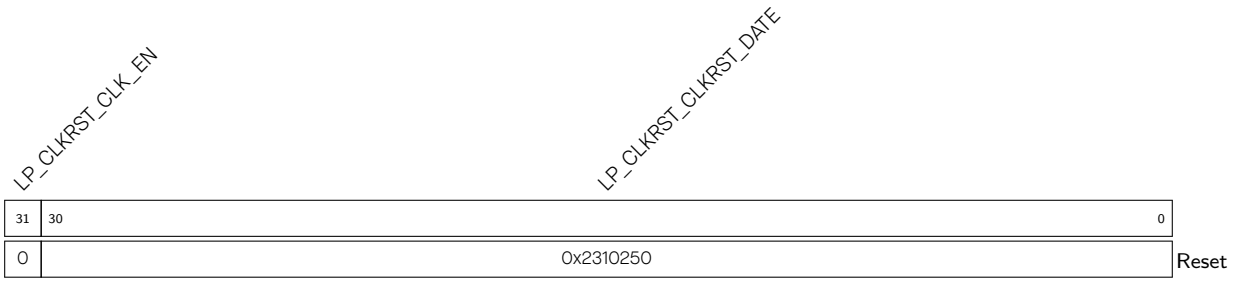
**LP_CLKRST_ICG_HP_FOSC** Configures whether to gate the RC_FAST_CLK signals to HP system.

0: Disable the clock gate

1: Enable the clock gate

(R/W)

Register 8.71. LP_CLKRST_DATE_REG (0x03FC)




**LP_CLKRST_CLKRST_DATE** Version control register. (R/W)

**LP_CLKRST_CLK_EN** Configures whether to force enable the clock gate for header registers.

0: Invalid

1: Force enable

(R/W)

# Chapter 9

# Chip Boot Control

## 9.1 Overview

Chip boot process and some chip functions are determined on power-on or hardware reset by strapping pins and eFuse parameters. The following functionality can be determined:

- **Chip boot mode**
  - Strapping pins: GPIO8 and GPIO9
- **SDIO sampling and driving clock edge**
  - Strapping pins: MTDI and MTMS
- **UART0/USB ROM message printing**
  - Strapping pin: GPIO8
  - eFuse parameters: EFUSE_UART_PRINT_CONTROL and EFUSE_DIS_USB_SERIAL_JTAG_ROM_PRINT
- **JTAG signal source**
  - Strapping pin: GPIO7
  - eFuse parameters: EFUSE_DIS_PAD_JTAG, EFUSE_DIS_USB_JTAG, and EFUSE_JTAG_SEL_ENABLE

The default values of all the above eFuse parameters are 0, which means that they are not burnt. Given that eFuse is one-time programmable, once programmed to 1, it can never be reverted to 0. For how to program eFuse parameters, please refer to Chapter 6 *eFuse Controller (EFUSE)*.

During power-on reset, and brownout reset (see Chapter 8 *Reset and Clock*), hardware captures 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 or next chip reset. Software can read the latch status (strapping value) from GPIO_STRAPPING.

## 9.2 Functional Description

This section introduces chip reset functions and the patterns of the strapping pins and eFuse values used to invoke each function.

**Notice:** Only documented patterns should be used. Undocumented patterns may trigger unexpected behaviors.

### 9.2.1 Default Configuration

By default, GPIO9 is connected to the chip's internal pull-up resistor. If GPIO9 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 9.2-1).

Table 9.2-1. Default Configuration of Strapping Pins

| Strapping Pin | Default Configuration |
|---|---|
| MTMS | Floating |
| MTDI | Floating |
| GPIO7 | Floating |
| GPIO8 | Floating |
| GPIO9 | Pull-up |

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-C61. After the reset is released, the strapping pins work as normal-function pins.

### 9.2.2 Boot Mode Control

The values of GPIO9 and GPIO8 at reset determine the boot mode after the reset is released. Table 9.2-2 shows the strapping pin values of GPIO9 and GPIO8, and the associated boot modes.

Table 9.2-2. Boot Mode Control

| Boot Mode | GPIO9 | GPIO8 |
|---|---|---|
| SPI Boot mode | 1 | x[1] |
| Joint Download Boot mode[2] | 0 | 1 |

[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
- SDIO Slave 2.0 Download Boot

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 SRAM 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 9.2-1.

In **Joint Download Boot mode**, users can download binary files into flash using UART0, USB or SDIO Slave interface. It is also possible to download binary files into SRAM and execute it from SRAM.

Figure 9.2-1 shows the detailed boot flow of the chip.




Figure 9.2-1. Chip Boot Flow

**Notice:** The strapping values "1xxx" and "01xx" in the above figure are the combination of GPIO9 and GPIO8. See Table 9.2-2.

The following eFuses control 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 LP_AON_FORCE_DOWNLOAD_BOOT and triggering a CPU reset. In this case, hardware overwrites GPIO_STRAPPING[3:2] from "1x" to "01".

  - If this eFuse is 1, LP_AON_FORCE_DOWNLOAD_BOOT is disabled, and GPIO_STRAPPING can not be overwritten.

- **EFUSE_DIS_DOWNLOAD_MODE**

  If this eFuse is 1, Joint Download Boot mode is disabled, and GPIO_STRAPPING will not be overwritten by LP_AON_FORCE_DOWNLOAD_BOOT.

- **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 SRAM or register operations. Ignore this eFuse if Joint Download Boot mode is disabled.

- **EFUSE_DIS_DIRECT_BOOT**

  If this eFuse is 1, Direct Boot mode is disabled.

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 30 *USB Serial/JTAG Controller*.

### 9.2.3 SDIO Sampling and Driving Clock Edge Control

The strapping pin MTMS and MTDI can be used to decide on which clock edge to sample signals and drive output lines. See Table 9.2-3 SDIO Input Sampling Edge/Output Driving Edge Control.

**Table 9.2-3. SDIO Input Sampling Edge/Output Driving Edge Control**

| Edge behavior | MTMS[1] | MTDI[1] |
|---|---|---|
| Falling edge sampling, falling edge output | 0 | 0 |
| Falling edge sampling, rising edge output | 0 | 1 |
| Rising edge sampling, falling edge output | 1 | 0 |
| Rising edge sampling, rising edge output | 1 | 1 |

[1] MTMS and MTDI are floating by default, so above are not default configurations.

### 9.2.4 ROM Messages Printing Control

During the ROM boot stage of SPI Boot mode, GPIO8, LP_AON_STORE4_REG[0], and EFUSE_UART_PRINT_CONTROL jointly control the printing of ROM messages.

Table 9.2-4. ROM Message Printing Control

| LP_AON _STORE4_REG[0] | EFUSE_UART _PRINT_CONTROL | GPIO8 | ROM Message Printing |
|---|---|---|---|
| 0 | 0 (0b00) | x[1] | Always printed to UART0 |
| | 1 (0b01) | 0 | Enabled |
| | | 1 | Disabled |
| | 2 (0b10) | 0 | Disabled |
| | | 1 | Enabled |
| | 3 (0b11) | x | Disabled |
| 1 | x | x | Disabled |

[1] x: values that have no effect on the result and can therefore be ignored.

ROM message is printed to UART0 and USB Serial/JTAG Controller by default during power-on. Users can disable the printing to USB Serial/JTAG Controller by setting the eFuse bit EFUSE_DIS_USB_SERIAL_JTAG_ROM_PRINT.

Note that if EFUSE_DIS_USB_SERIAL_JTAG_ROM_PRINT is set to 0 to print to USB, but the USB Serial/JTAG Controller has been disabled, then ROM messages will not be printed to USB Serial/JTAG Controller.

### 9.2.5 JTAG Signal Source Control

The strapping pin GPIO7 can be used to control the source of JTAG signals during the early boot process. This pin does not have any internal pull resistors and the strapping value must be controlled by the external circuit that cannot be in a high impedance state.

As Table 9.2-5 shows, GPIO7 is used in combination with EFUSE_DIS_PAD_JTAG, EFUSE_DIS_USB_JTAG, and EFUSE_JTAG_SEL_ENABLE.

Table 9.2-5. JTAG Signal Source Control

| EFUSE_DIS _PAD_JTAG | EFUSE_DIS _USB_JTAG | EFUSE_JTAG _SEL_ENABLE | GPIO7 | JTAG Signal Source |
|---|---|---|---|---|
| 0 | 0 | 0 | x[1] | USB Serial/JTAG Controller |
| | | 1 | 1 | |
| | | | 0 | JTAG pins MTDI, MTCK, MTMS, and MTDO |
| 0 | x | x | x | |
| 0 | 1 | x | x | |
| 1 | 0 | x | x | USB Serial/JTAG Controller |
| 1 | 1 | x | x | JTAG is disabled |
| 1 | x | x | x | |

[1] x: x indicates that the value has no effect on the result and can be ignored.

# Chapter 10

# Interrupt Matrix

## 10.1 Overview

The interrupt matrix embedded in ESP32-C61 routes one or multiple peripheral interrupt sources to any of the ESP-RISC-V CPU's peripheral interrupts.

The ESP32-C61 has 66 peripheral interrupt sources that can be routed to any of the 32 CPU peripheral interrupts using the interrupt matrix.

**Note:**
This chapter focuses on how to map peripheral interrupt sources to the CPU interrupts. For information about interrupt configuration, vector, and interrupt handling operations recommended by the ISA, please refer to Chapter 2 *ESP-RISC-V CPU* > Section 2.8 *Interrupt Controller*.

## 10.2 Interrupt Terminology in ESP32-C61

The following terms related to interrupts are defined in the context of the *ESP32-C61 Technical Reference Manual* to help readers better understand this document:

### 10.2.1 Interrupt

An interrupt refers to the event or condition that occurs, causing the CPU to temporarily suspend its current execution and handle a higher-priority task. It is a mechanism that allows the CPU to respond to specific events promptly.

The *ESP32-C61 Technical Reference Manual* may use the term "interrupt" in a broader sense to refer to both interrupt signal and interrupt source.

### 10.2.2 Interrupt Signal/Interrupt Source

Interrupt signal and interrupt source are only defined from different perspectives and mean the same thing.

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 the peripheral 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 sources and are sent to the CPU core together with the core local interrupt sources.

### 10.2.3 Interrupt Flow in ESP32-C61

Figure 10.2-1 shows the interrupt flow in ESP32-C61.

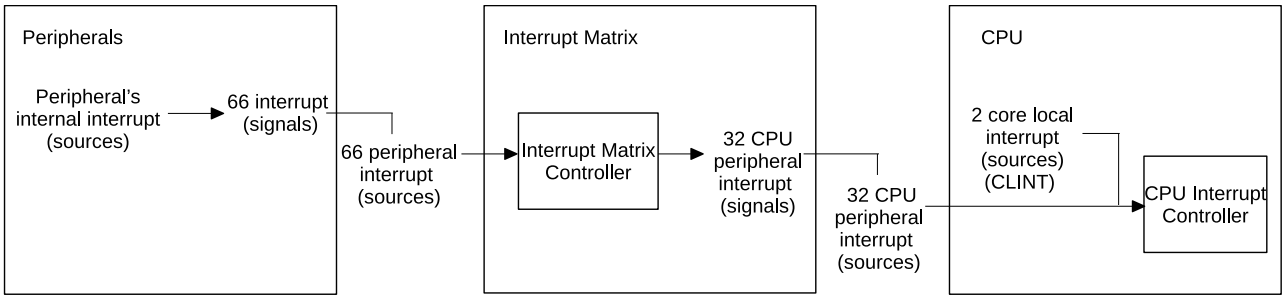



Figure 10.2-1. Interrupt Flow in ESP32-C61

## 10.3 Features

The interrupt matrix embedded in the ESP32-C61 has the following features:

- 66 peripheral interrupt sources accepted as input
- 32 HP CPU peripheral interrupts generated to the HP CPU as output
- Current interrupt status query of peripheral interrupt sources
- Multiple interrupt sources mapping to a single HP CPU interrupt (i.e., shared interrupts)

## 10.4 Architecture

Figure 10.4-1 shows the structure of the interrupt matrix.

You need to configure the interrupt matrix registers to map the peripheral interrupt sources to the HP CPU interrupts. The Interrupt Matrix Controller in Figure 10.4-1 manages the mapping and sends the interrupt status of each interrupt source to the interrupt status registers which belong to the interrupt matrix registers.

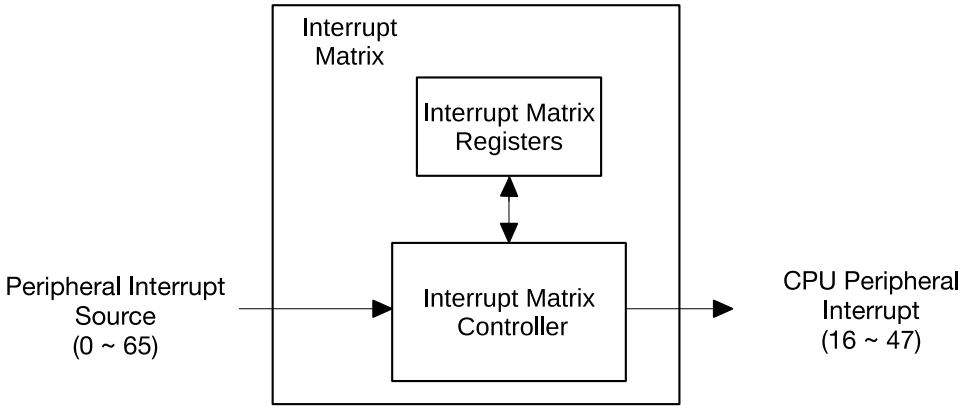



Figure 10.4-1. Interrupt Matrix Structure

## 10.5 Functional Description

### 10.5.1 Peripheral Interrupt Sources

The ESP32-C61 has 66 peripheral interrupt sources in total. Table 10.5-1 lists all these sources and their mapping registers, as well as the interrupt status registers.

- Column "No.": Peripheral interrupt source number, can be 0 ~ 61.
- Column "Chapter": in which chapter the interrupt source is described in detail.
- Column "Interrupt Source": Name of the peripheral interrupt source.
- Column "Interrupt Source Mapping Register": Registers used to configure the routing of the peripheral interrupt sources to the HP CPU peripheral interrupts.
- Column "Interrupt Status Register": Registers used to reflect the interrupt source status.
  - Column "Interrupt Status Register - Bit": Bit position in status registers, indicating the interrupt status.
  - Column "Interrupt Status Register - Name": Name of status registers.

Table 10.5-1. 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 | n/a | reserved | reserved | 0 | INTMTX_COREO_INT_STATUS_0_REG |
| 1 | n/a | reserved | reserved | 1 | |
| 2 | n/a | reserved | reserved | 2 | |
| 3 | n/a | reserved | reserved | 3 | |
| 4 | n/a | reserved | reserved | 4 | |
| 5 | n/a | reserved | reserved | 5 | |
| 6 | n/a | reserved | reserved | 6 | |
| 7 | n/a | reserved | reserved | 7 | |
| 8 | n/a | reserved | reserved | 9 | |
| 9 | n/a | reserved | reserved | 9 | |
| 10 | n/a | reserved | reserved | 10 | |
| 11 | n/a | reserved | reserved | 11 | |
| 12 | n/a | reserved | reserved | 12 | |
| 13 | *Low-Power Management* | PMU_INTR | INTMTX_COREO_PMU_INTR_MAP_REG | 13 | |
| 14 | *eFuse Controller (EFUSE)* | EFUSE_INTR | INTMTX_COREO_EFUSE_INTR_MAP_REG | 14 | |
| 15 | *Low-Power Management* | LP_RTC_TIMER_INTR | INTMTX_COREO_LP_RTC_TIMER_INTR_MAP_REG | 15 | |
| 16 | *Low-Power Management* | LP_WDT_INTR | INTMTX_COREO_LP_WDT_INTR_MAP_REG | 16 | |
| 17 | *System Registers* | LP_PERI_TIMEOUT_INTR | INTMTX_COREO_LP_PERI_TIMEOUT_INTR_MAP_REG | 17 | |
| 18 | *Permission Control (PMS)* | LP_APM_M0_INTR | INTMTX_COREO_LP_APM_M0_INTR_MAP_REG | 18 | |
| 19 | *Software Interrupt Registers* | CPU_INTR_FROM_CPU_0 | INTMTX_COREO_CPU_INTR_FROM_CPU_0_MAP_REG | 19 | |
| 20 | *Software Interrupt Registers* | CPU_INTR_FROM_CPU_1 | INTMTX_COREO_CPU_INTR_FROM_CPU_1_MAP_REG | 20 | |
| 21 | *Software Interrupt Registers* | CPU_INTR_FROM_CPU_2 | INTMTX_COREO_CPU_INTR_FROM_CPU_2_MAP_REG | 21 | |
| 22 | *ESP-RISC-V CPU* | CPU_INTR_FROM_CPU_3 | INTMTX_COREO_CPU_INTR_FROM_CPU_3_MAP_REG | 22 | |
| 23 | *Debug Assistant* | ASSIST_DEBUG_INTR | INTMTX_COREO_ASSIST_DEBUG_INTR_MAP_REG | 23 | |
| 24 | *ESP-RISC-V CPU* | TRACE_INTR | INTMTX_COREO_TRACE_INTR_MAP_REG | 24 | |
| 25 | n/a | reserved | reserved | 25 | |
| 26 | *System Registers* | CPU_PERI_TIMEOUT_INTR | INTMTX_COREO_CPU_PERI_TIMEOUT_INTR_MAP_REG | 26 | |
| 27 | *GPIO Matrix and IO MUX* | GPIO_INTR_PRO | INTMTX_COREO_GPIO_INTR_PRO_MAP_REG | 27 | |
| 28 | *GPIO Matrix and IO MUX* | GPIO_INTR_EXT | INTMTX_COREO_GPIO_INTR_EXT_MAP_REG | 28 | |
| 29 | *Low-Power Management* | PAU_INTR | INTMTX_COREO_PAU_INTR_MAP_REG | 29 | |
| 30 | *System Registers* | HP_PERI_TIMEOUT_INTR | INTMTX_COREO_HP_PERI_TIMEOUT_INTR_MAP_REG | 30 | |
| 31 | n/a | reserved | reserved | 31 | |
| 32 | *Permission Control (PMS)* | HP_APM_M0_INTR | INTMTX_COREO_HP_APM_M0_INTR_MAP_REG | 0 | INTMTX_COREO_INT_STATUS_1_REG |
| 33 | *Permission Control (PMS)* | HP_APM_M1_INTR | INTMTX_COREO_HP_APM_M1_INTR_MAP_REG | 1 | |
| 34 | *Permission Control (PMS)* | HP_APM_M2_INTR | INTMTX_COREO_HP_APM_M2_INTR_MAP_REG | 2 | |
| 35 | *Permission Control (PMS)* | HP_APM_M3_INTR | INTMTX_COREO_HP_APM_M3_INTR_MAP_REG | 3 | |
| 36 | *Permission Control (PMS)* | CPU_APM_M0_INTR | INTMTX_COREO_CPU_APM_M0_INTR_MAP_REG | 4 | |

| No. | Chapter | Interrupt Source | Interrupt Source Mapping Register | Interrupt Status Register | |
|---|---|---|---|---|---|
| | | | | Bit | Name |
| 37 | *Permission Control (PMS)* | CPU_APM_M1_INTR | INTMTX_COREO_CPU_APM_M1_INTR_MAP_REG | 5 | |
| 38 | *Permission Control (PMS)* | MSPI_INTR | INTMTX_COREO_MSPI_INTR_MAP_REG | 6 | |
| 39 | *I2S Controller (I2S)* | I2S_INTR | INTMTX_COREO_I2S_INTR_MAP_REG | 7 | |
| 40 | *UART Controller (UART)* | UART0_INTR | INTMTX_COREO_UART0_INTR_MAP_REG | 8 | |
| 41 | *UART Controller (UART)* | UART1_INTR | INTMTX_COREO_UART1_INTR_MAP_REG | 9 | |
| 42 | *UART Controller (UART)* | UART2_INTR | INTMTX_COREO_UART2_INTR_MAP_REG | 10 | |
| 43 | *LED PWM Controller (LEDC)* | LEDC_INTR | INTMTX_COREO_LEDC_INTR_MAP_REG | 11 | |
| 44 | *USB Serial/JTAG Controller* | USB_INTR | INTMTX_COREO_USB_INTR_MAP_REG | 12 | |
| 45 | *I2C Controller (I2C)* | I2C_EXT0_INTR | INTMTX_COREO_I2C_EXT0_INTR_MAP_REG | 13 | |
| 46 | *Timer Group (TIMG)* | TG0_T0_INTR | INTMTX_COREO_TG0_T0_INTR_MAP_REG | 14 | |
| 47 | *Timer Group (TIMG)* | TG0_T1_INTR | INTMTX_COREO_TG0_T1_INTR_MAP_REG | 15 | |
| 48 | *Timer Group (TIMG)* | TG0_WDT_INTR | INTMTX_COREO_TG0_WDT_INTR_MAP_REG | 16 | |
| 49 | *Timer Group (TIMG)* | TG1_T0_INTR | INTMTX_COREO_TG1_T0_INTR_MAP_REG | 17 | |
| 50 | *Timer Group (TIMG)* | TG1_T1_INTR | INTMTX_COREO_TG1_T1_INTR_MAP_REG | 18 | |
| 51 | *Timer Group (TIMG)* | TG1_WDT_INTR | INTMTX_COREO_TG1_WDT_INTR_MAP_REG | 19 | |
| 52 | *System Timer* | SYSTIMER_TARGET0_INTR | INTMTX_COREO_SYSTIMER_TARGET0_INTR_MAP_REG | 20 | |
| 53 | *System Timer* | SYSTIMER_TARGET1_INTR | INTMTX_COREO_SYSTIMER_TARGET1_INTR_MAP_REG | 21 | |
| 54 | *System Timer* | SYSTIMER_TARGET2_INTR | INTMTX_COREO_SYSTIMER_TARGET2_INTR_MAP_REG | 22 | |
| 55 | *ADC Controller* | APB_ADC_INTR | INTMTX_COREO_APB_ADC_INTR_MAP_REG | 23 | |
| 56 | *SDIO Slave Controller (SDIO)* | SLC0_INTR | INTMTX_COREO_SLC0_INTR_MAP_REG | 24 | |
| 57 | *SDIO Slave Controller (SDIO)* | SLC1_INTR | INTMTX_COREO_SLC1_INTR_MAP_REG | 25 | |
| 58 | *GDMA Controller (GDMA)* | GDMA_IN_CH0_INTR | INTMTX_COREO_DMA_IN_CH0_INTR_MAP_REG | 26 | |
| 59 | *GDMA Controller (GDMA)* | GDMA_IN_CH1_INTR | INTMTX_COREO_DMA_IN_CH1_INTR_MAP_REG | 27 | |
| 60 | *GDMA Controller (GDMA)* | GDMA_OUT_CH0_INTR | INTMTX_COREO_DMA_OUT_CH0_INTR_MAP_REG | 28 | |
| 61 | *GDMA Controller (GDMA)* | GDMA_OUT_CH1_INTR | INTMTX_COREO_DMA_OUT_CH1_INTR_MAP_REG | 29 | |
| 62 | *SPI Controller (SPI)* | GPSPI2_INTR | INTMTX_COREO_GPSPI2_INTR_MAP_REG | 30 | |
| 63 | *SHA Accelerator (SHA)* | SHA_INTR | INTMTX_COREO_SHA_INTR_MAP_REG | 31 | |
| 64 | *ECC Accelerator (ECC)* | ECC_INTR | INTMTX_COREO_ECC_INTR_MAP_REG | 0 | INTMTX_COREO_INT_STATUS_2_REG |
| 65 | *ECDSA Digital Signature Peripheral (ECDSA_DS)* | ECDSA_INTR | INTMTX_COREO_ECDSA_INTR_MAP_REG | 1 | |



GoBack

### 10.5.2 HP CPU Interrupts

The HP CPU of the ESP32-C61 implements its interrupt mechanism using standard RISC-V (v0.9) Core-Local Interrupt Controller (CLIC) interrupt. The HP CPU supports 2 core local interrupts (CLINT) and 32 peripheral interrupts, which are numbered from 16 to 47. The peripheral interrupt sources are mapped to the 32 HP CPU peripheral interrupts through the interrupt matrix. Each HP CPU peripheral interrupt has the following properties:

- Priority levels from 1 (lowest) to 15 (highest).
- Configurable as level-triggered or edge-triggered.
- Lower-priority interrupts maskable by setting interrupt threshold.

**Note:**
For detailed information about the functions and configuration procedures of CPU interrupts, see Chapter 2 *ESP-RISC-V CPU* > Section 2.8*Interrupt Controller*. The configuration registers of CPU interrupts are listed in Section 10.6.2 *Software Interrupt Register Summary*.

### 10.5.3 Assign Peripheral Interrupt Source to HP CPU Peripheral 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 10.5-1.
- INTMTX_COREO_*SOURCE*_INTR_MAP_REG: stands for the interrupt source mapping register for a *SOURCE*.
- Num_P: stands for the index of HP CPU interrupts which can be 16 ~ 47.
- Interrupt_P: stands for the HP CPU interrupt numbered as Num_P.

#### 10.5.3.1 Assign One Peripheral Interrupt Source to HP CPU Peripheral Interrupt

Setting the corresponding source mapping register INTMTX_COREO_*SOURCE*_INTR_MAP_REG of *SOURCE* to Num_P assigns this interrupt source to Interrupt_P.

#### 10.5.3.2 Assign Multiple Peripheral Interrupt Sources to HP CPU Peripheral Interrupt

Setting the corresponding source mapping register INTMTX_COREO_*SOURCE*_INTR_MAP_REG of each *SOURCE* to the same Num_P assigns multiple sources to the same Interrupt_P. Any of these sources can trigger CPU Interrupt_P. When an interrupt signal is generated, the CPU should check the interrupt status registers to figure out which peripheral generated the interrupt. For more information, see Chapter 2 *ESP-RISC-V CPU* > Section 2.8*Interrupt Controller*.

#### 10.5.3.3 Unassign SOURCE

Writing 0 to the INTMTX_COREO_*SOURCE*_INTR_MAP_REG register disables the corresponding interrupt source.

### 10.5.4 Delegated Interrupts

The ESP32-C61 RISC-V CPU supports Machine Mode and User Mode. The 32 external HP CPU interrupts can be configured as either Machine Mode interrupts or User Mode interrupts. When the HP CPU is in Machine Mode, it only responds to Machine Mode interrupts and does not respond to User Mode interrupts. When the HP CPU is in User Mode, it can respond to all interrupts.

Under normal conditions, User Mode interrupts are only handled when the CPU is operating in User Mode. However, with the interrupt delegation feature, Machine Mode can be configured to handle certain User Mode interrupts, while the CPU itself remains in Machine Mode. This is achieved by configuring registers to delegate User Mode interrupts to Machine Mode interrupts.

- Each interrupt source of the interrupt matrix can independently enable or disable the interrupt delegation feature.
- Multiple interrupt sources from the interrupt matrix can enable the interrupt delegation feature at the same time.
- Delegation is allowed to only one interrupt signal from the interrupt matrix.

Software can configure the INTMTX_COREO_INT_SIG_IDX_ASSERT_IN_SEC_REG register to n to delegate a User Mode interrupt to a Machine Mode interrupt with interrupt number n.

To enable the delegated interrupt function for interrupt source *SOURCE* in the interrupt matrix, software can set the corresponding bit in the INTMTX_COREO_*SOURCE*_INTR_PASS_IN_SEC_REG register to 1.

### 10.5.5 Query Current Interrupt Status of SOURCE

After enabling a *SOURCE*, you can query its current interrupt status by reading the corresponding bit value in INTMTX_COREO_INT_STATUS_*n*_REG (read only). For the mapping between INTMTX_COREO_INT_STATUS_*n*_REG and the *SOURCE*, please refer to Table 10.5-1.

## 10.6 Register Summary

### 10.6.1 Interrupt Matrix Register Summary

The addresses in this section are relative to the interrupt matrix base address provided in Table 5.3-2 in Chapter 5 *System and Memory*.

The abbreviations given in column **Access** are explained in Section *Access Types for Registers*.

| Name | Description | Address | Access |
|---|---|---|---|
| **Configuration Registers** | | | |
| INTMTX_COREO_PMU_INTR_MAP_REG | PMU_INTR mapping register | 0x034 | R/W |
| INTMTX_COREO_EFUSE_INTR_MAP_REG | EFUSE_INTR mapping register | 0x038 | R/W |
| INTMTX_COREO_LP_RTC_TIMER_INTR_MAP_REG | LP_RTC_TIMER_INTR mapping register | 0x03C | R/W |
| INTMTX_COREO_LP_WDT_INTR_MAP_REG | LP_WDT_INTR mapping register | 0x040 | R/W |
| INTMTX_COREO_LP_PERI_TIMEOUT_INTR_MAP_REG | LP_PERI_TIMEOUT_INTR mapping register | 0x044 | R/W |
| INTMTX_COREO_LP_APM_M0_INTR_MAP_REG | LP_APM_M0_INTR mapping register | 0x048 | R/W |
| INTMTX_COREO_CPU_INTR_FROM_CPU_0_MAP_REG | CPU_INTR_FROM_CPU_0 mapping register | 0x04C | R/W |
| INTMTX_COREO_CPU_INTR_FROM_CPU_1_MAP_REG | CPU_INTR_FROM_CPU_1 mapping register | 0x050 | R/W |
| INTMTX_COREO_CPU_INTR_FROM_CPU_2_MAP_REG | CPU_INTR_FROM_CPU_2 mapping register | 0x054 | R/W |
| INTMTX_COREO_CPU_INTR_FROM_CPU_3_MAP_REG | CPU_INTR_FROM_CPU_3 mapping register | 0x058 | R/W |
| INTMTX_COREO_ASSIST_DEBUG_INTR_MAP_REG | ASSIST_DEBUG_INTR mapping register | 0x05C | R/W |
| INTMTX_COREO_TRACE_INTR_MAP_REG | TRACE_INTR mapping register | 0x060 | R/W |
| INTMTX_COREO_CPU_PERI_TIMEOUT_INTR_MAP_REG | CPU_PERI_TIMEOUT_INTR mapping register | 0x068 | R/W |
| INTMTX_COREO_GPIO_INTR_PRO_MAP_REG | GPIO_INTMTX_PRO mapping register | 0x06C | R/W |
| INTMTX_COREO_GPIO_INTR_EXT_MAP_REG | GPIO_INTMTX_EXT mapping register | 0x070 | R/W |
| INTMTX_COREO_PAU_INTR_MAP_REG | PAU_INTR mapping register | 0x074 | R/W |
| INTMTX_COREO_HP_PERI_TIMEOUT_INTR_MAP_REG | HP_PERI_TIMEOUT_INTR mapping register | 0x078 | R/W |
| INTMTX_COREO_HP_APM_M0_INTR_MAP_REG | HP_APM_M0_INTR mapping register | 0x080 | R/W |
| INTMTX_COREO_HP_APM_M1_INTR_MAP_REG | HP_APM_M1_INTR mapping register | 0x084 | R/W |
| INTMTX_COREO_HP_APM_M2_INTR_MAP_REG | HP_APM_M2_INTR mapping register | 0x088 | R/W |
| INTMTX_COREO_HP_APM_M3_INTR_MAP_REG | HP_APM_M3_INTR mapping register | 0x08C | R/W |

| Name | Description | Address | Access |
|---|---|---|---|
| INTMTX_COREO_CPU_APM_MO_INTR_MAP_REG | CPU_APM_MO_INTR mapping register | 0x090 | R/W |
| INTMTX_COREO_CPU_APM_M1_INTR_MAP_REG | CPU_APM_M1_INTR mapping register | 0x094 | R/W |
| INTMTX_COREO_MSPI_INTR_MAP_REG | MSPI_INTR mapping register | 0x098 | R/W |
| INTMTX_COREO_I2S_INTR_MAP_REG | I2S_INTR mapping register | 0x09C | R/W |
| INTMTX_COREO_UARTO_INTR_MAP_REG | UARTO_INTR mapping register | 0x0A0 | R/W |
| INTMTX_COREO_UART1_INTR_MAP_REG | UART1_INTR mapping register | 0x0A4 | R/W |
| INTMTX_COREO_UART2_INTR_MAP_REG | UART2_INTR mapping register | 0x0A8 | R/W |
| INTMTX_COREO_LEDC_INTR_MAP_REG | LEDC_INTR mapping register | 0x0AC | R/W |
| INTMTX_COREO_USB_INTR_MAP_REG | USB_INTR mapping register | 0x0B0 | R/W |
| INTMTX_COREO_I2C_EXTO_INTR_MAP_REG | I2C_EXTO_INTR mapping register | 0x0B4 | R/W |
| INTMTX_COREO_TGO_TO_INTR_MAP_REG | TGO_TO_INTR mapping register | 0x0B8 | R/W |
| INTMTX_COREO_TGO_T1_INTR_MAP_REG | TGO_T1_INTR mapping register | 0x0BC | R/W |
| INTMTX_COREO_TGO_WDT_INTR_MAP_REG | TGO_WDT_INTR mapping register | 0x0C0 | R/W |
| INTMTX_COREO_TG1_TO_INTR_MAP_REG | TG1_TO_INTR mapping register | 0x0C4 | R/W |
| INTMTX_COREO_TG1_T1_INTR_MAP_REG | TG1_T1_INTR mapping register | 0x0C8 | R/W |
| INTMTX_COREO_TG1_WDT_INTR_MAP_REG | TG1_WDT_INTR mapping register | 0x0CC | R/W |
| INTMTX_COREO_SYSTIMER_TARGETO_INTR_MAP_REG | SYSTIMER_TARGETO_INTR mapping register | 0x0D0 | R/W |
| INTMTX_COREO_SYSTIMER_TARGET1_INTR_MAP_REG | SYSTIMER_TARGET1_INTR mapping register | 0x0D4 | R/W |
| INTMTX_COREO_SYSTIMER_TARGET2_INTR_MAP_REG | SYSTIMER_TARGET2_INTR mapping register | 0x0D8 | R/W |
| INTMTX_COREO_APB_ADC_INTR_MAP_REG | APB_ADC_INTR mapping register | 0x0DC | R/W |
| INTMTX_COREO_SLCO_INTR_MAP_REG | SLCO_INTR mapping register | 0x0E0 | R/W |
| INTMTX_COREO_SLC1_INTR_MAP_REG | SLC1_INTR mapping register | 0x0E4 | R/W |
| INTMTX_COREO_DMA_IN_CHO_INTR_MAP_REG | DMA_IN_CHO_INTR mapping register | 0x0E8 | R/W |
| INTMTX_COREO_DMA_IN_CH1_INTR_MAP_REG | DMA_IN_CH1_INTR mapping register | 0x0EC | R/W |
| INTMTX_COREO_DMA_OUT_CHO_INTR_MAP_REG | DMA_OUT_CHO_INTR mapping register | 0x0F0 | R/W |
| INTMTX_COREO_DMA_OUT_CH1_INTR_MAP_REG | DMA_OUT_CH1_INTR mapping register | 0x0F4 | R/W |
| INTMTX_COREO_GPSPI2_INTR_MAP_REG | GPSPI2_INTR mapping register | 0x0F8 | R/W |
| INTMTX_COREO_SHA_INTR_MAP_REG | SHA_INTR mapping register | 0x0FC | R/W |
| INTMTX_COREO_ECC_INTR_MAP_REG | ECC_INTR mapping register | 0x100 | R/W |



GoBack

| Name | Description | Address | Access |
|---|---|---|---|
| INTMTX_COREO_ECDSA_INTR_MAP_REG | ECDSA_INTR mapping register | 0x104 | R/W |
| INTMTX_COREO_INT_STATUS_0_REG | Status register for INTMTX sources 0 ~ 31 | 0x0108 | RO |
| INTMTX_COREO_INT_STATUS_1_REG | Status register for INTMTX sources 32 ~ 63 | 0x010C | RO |
| INTMTX_COREO_INT_STATUS_1_REG | Status register for INTMTX sources 64 ~ 65 | 0x0110 | RO |
| INTMTX_COREO_INT_SRC_PASS_IN_SEC_STATUS_0_REG | Delegation Status Register for INTMTX sources 0 ~ 31 | 0x0114 | RO |
| INTMTX_COREO_INT_SRC_PASS_IN_SEC_STATUS_1_REG | Delegation Status Register for INTMTX sources 32 ~ 63 | 0x0118 | RO |
| INTMTX_COREO_INT_SRC_PASS_IN_SEC_STATUS_2_REG | Delegation Status Register for INTMTX sources 64 ~ 65 | 0x011C | RO |
| INTMTX_COREO_INT_SIG_IDX_ASSERT_IN_SEC_REG | Configuration register for interrupt delegation | 0x0120 | R/W |
| INTMTX_COREO_SECURE_STATUS_REG | Status register for interrupt delegation | 0x0124 | RO |
| INTMTX_COREO_CLOCK_GATE_REG | Clock gating register | 0x0128 | R/W |
| **Version Register** | | | |
| INTMTX_COREO_INTMTX_DATE_REG | Version control register | 0x07FC | R/W |

## 10.6.2 Software Interrupt Register Summary

The addresses in this section are relative to the software interrupt base address provided in Table 5.3-2 in Chapter 5 *System and Memory*.

The abbreviations given in column **Access** are explained in Section *Access Types for Registers*.

| Name | Description | Address | Access |
|---|---|---|---|
| **Interrupt Registers** | | | |
| INTPRI_CPU_INTR_FROM_CPU_0_REG | CPU_INTR_FROM_CPU_0 mapping register | 0x0090 | R/W |
| INTPRI_CPU_INTR_FROM_CPU_1_REG | CPU_INTR_FROM_CPU_1 mapping register | 0x0094 | R/W |
| INTPRI_CPU_INTR_FROM_CPU_2_REG | CPU_INTR_FROM_CPU_2 mapping register | 0x0098 | R/W |
| INTPRI_CPU_INTR_FROM_CPU_3_REG | CPU_INTR_FROM_CPU_3 mapping register | 0x009C | R/W |
| **Version Registers** | | | |
| INTPRI_DATE_REG | Version control register | 0x00A0 | R/W |

# 10.7 Registers

## 10.7.1 Interrupt Matrix Registers

The addresses in this section are relative to the interrupt matrix base address provided in Table 5.3-2 in Chapter 5 *System and Memory*.

Register 10.1. INTMTX_COREO_PMU_INTR_MAP_REG (0x034)

Register 10.2. INTMTX_COREO_EFUSE_INTR_MAP_REG (0x038)

Register 10.3. INTMTX_COREO_LP_RTC_TIMER_INTR_MAP_REG (0x03C)

Register 10.4. INTMTX_COREO_LP_WDT_INTR_MAP_REG (0x040)

Register 10.5. INTMTX_COREO_LP_PERI_TIMEOUT_INTR_MAP_REG (0x044)

Register 10.6. INTMTX_COREO_LP_APM_M0_INTR_MAP_REG (0x048)

Register 10.7. INTMTX_COREO_CPU_INTR_FROM_CPU_0_MAP_REG (0x04C)

Register 10.8. INTMTX_COREO_CPU_INTR_FROM_CPU_1_MAP_REG (0x050)

Register 10.9. INTMTX_COREO_CPU_INTR_FROM_CPU_2_MAP_REG (0x054)

Register 10.10. INTMTX_COREO_CPU_INTR_FROM_CPU_3_MAP_REG (0x058)

Register 10.11. INTMTX_COREO_ASSIST_DEBUG_INTR_MAP_REG (0x05C)

Register 10.12. INTMTX_COREO_TRACE_INTR_MAP_REG (0x060)

Register 10.13. INTMTX_COREO_CPU_PERI_TIMEOUT_INTR_MAP_REG (0x068)

Register 10.14. INTMTX_COREO_GPIO_INTR_PRO_MAP_REG (0x06C)

Register 10.15. INTMTX_COREO_GPIO_INTR_EXT_MAP_REG (0x070)

Register 10.16. INTMTX_COREO_PAU_INTR_MAP_REG (0x074)

Register 10.17. INTMTX_COREO_HP_PERI_TIMEOUT_INTR_MAP_REG (0x078)

Register 10.18. INTMTX_COREO_HP_APM_M0_INTR_MAP_REG (0x080)

Register 10.19. INTMTX_COREO_HP_APM_M1_INTR_MAP_REG (0x084)

Register 10.20. INTMTX_COREO_HP_APM_M2_INTR_MAP_REG (0x088)

Register 10.21. INTMTX_COREO_HP_APM_M3_INTR_MAP_REG (0x08C)

Register 10.22. INTMTX_COREO_CPU_APM_M0_INTR_MAP_REG (0x090)

Register 10.23. INTMTX_COREO_CPU_APM_M1_INTR_MAP_REG (0x094)

Register 10.24. INTMTX_COREO_MSPI_INTR_MAP_REG (0x098)

Register 10.25. INTMTX_COREO_I2S_INTR_MAP_REG (0x09C)

Register 10.26. INTMTX_COREO_UART0_INTR_MAP_REG (0x0A0)

Register 10.27. INTMTX_COREO_UART1_INTR_MAP_REG (0x0A4)

Register 10.28. INTMTX_COREO_UART2_INTR_MAP_REG (0x0A8)

Register 10.29. INTMTX_COREO_LEDC_INTR_MAP_REG (0x0AC)

Register 10.30. INTMTX_COREO_USB_INTR_MAP_REG (0x0B0)

Register 10.31. INTMTX_COREO_I2C_EXTO_INTR_MAP_REG (0x0B4)

Register 10.32. INTMTX_COREO_TGO_TO_INTR_MAP_REG (0x0B8)

Register 10.33. INTMTX_COREO_TGO_T1_INTR_MAP_REG (0x0BC)

Register 10.34. INTMTX_COREO_TGO_WDT_INTR_MAP_REG (0x0C0)

Register 10.35. INTMTX_COREO_TG1_TO_INTR_MAP_REG (0x0C4)

Register 10.36. INTMTX_COREO_TG1_T1_INTR_MAP_REG (0x0C8)

Register 10.37. INTMTX_COREO_TG1_WDT_INTR_MAP_REG (0x0CC)

Register 10.38. INTMTX_COREO_SYSTIMER_TARGETO_INTR_MAP_REG (0x0D0)

Register 10.39. INTMTX_COREO_SYSTIMER_TARGET1_INTR_MAP_REG (0x0D4)

Register 10.40. INTMTX_COREO_SYSTIMER_TARGET2_INTR_MAP_REG (0x0D8)

Register 10.41. INTMTX_COREO_APB_ADC_INTR_MAP_REG (0x0DC)

Register 10.42. INTMTX_COREO_SLCO_INTR_MAP_REG (0x0E0)

Register 10.43. INTMTX_COREO_SLC1_INTR_MAP_REG (0x0E4)

Register 10.44. INTMTX_COREO_DMA_IN_CHO_INTR_MAP_REG (0x0E8)

Register 10.45. INTMTX_COREO_DMA_IN_CH1_INTR_MAP_REG (0x0EC)

Register 10.46. INTMTX_COREO_DMA_OUT_CHO_INTR_MAP_REG (0x0F0)

Register 10.47. INTMTX_COREO_DMA_OUT_CH1_INTR_MAP_REG (0x0F4)

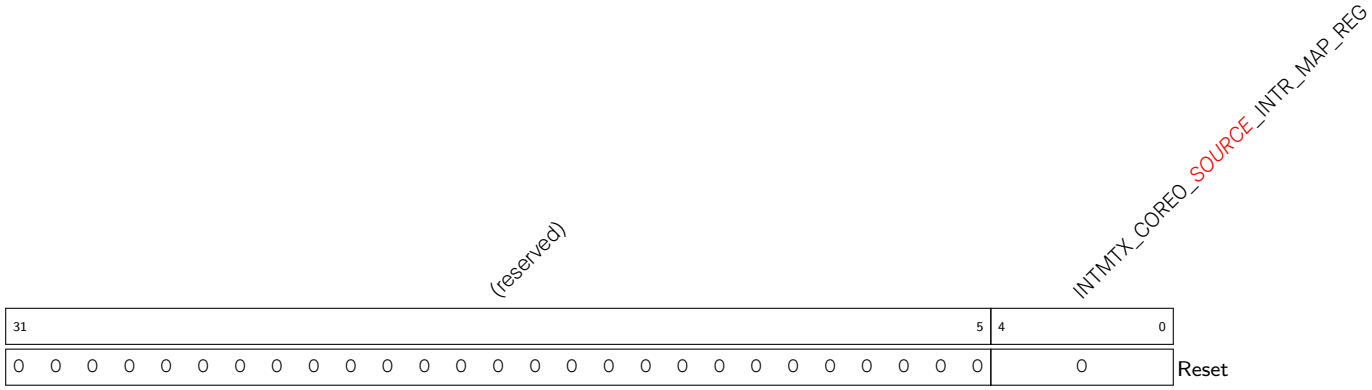
Register 10.48. INTMTX_COREO_GPSPI2_INTR_MAP_REG (0x0F8)

Register 10.49. INTMTX_COREO_SHA_INTR_MAP_REG (0x0FC)

Register 10.50. INTMTX_COREO_ECC_INTR_MAP_REG (0x100)

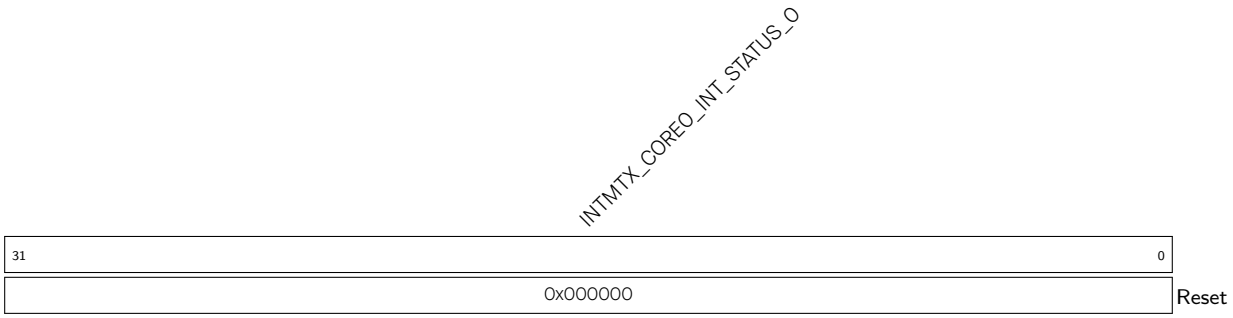
Register 10.51. INTMTX_COREO_ECDSA_INTR_MAP_REG (0x104)

Register 10.52. INTMTX_COREO_*SOURCE*_INTR_MAP_REG (0x0000 - 0x0104)




**INTMTX_COREO_*SOURCE*_INTR_MAP** Map the INTMTX source (*SOURCE*) into one CPU INTMTX. For the information of *SOURCE*, see Table 10.5-1. (R/W)

Register 10.53. INTMTX_COREO_INT_STATUS_0_REG (0x0108)



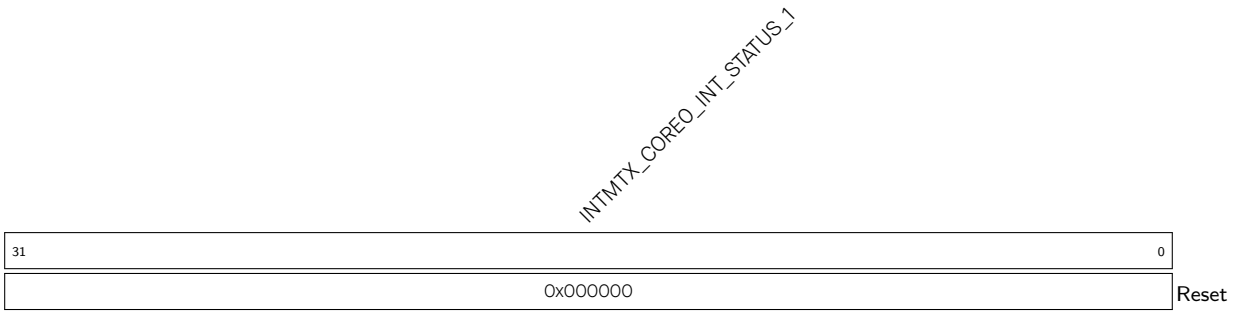

**INTMTX_COREO_INT_STATUS_0** Represents the status of the INTMTX sources numbered from 0 ~ 31. Each bit corresponds to one INTMTX source.

0: The corresponding INTMTX source triggered an interrupt

1: No interrupt triggered

(RO)

Register 10.54. INTMTX_COREO_INT_STATUS_1_REG (0x0144)



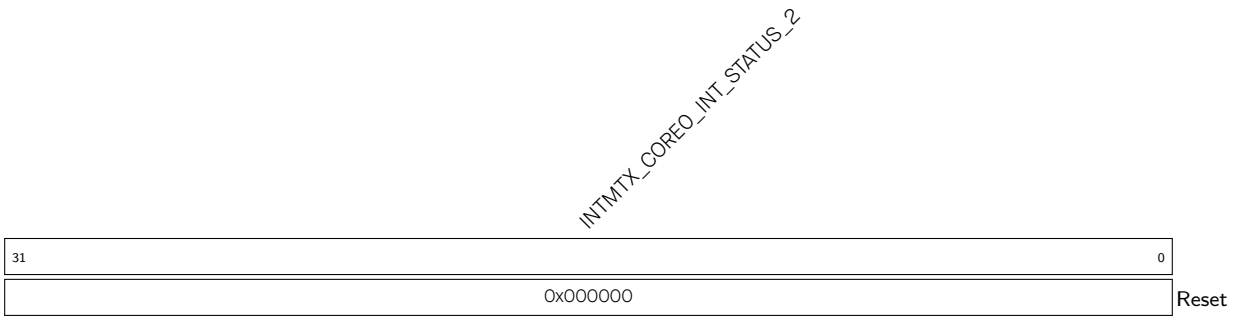

**INTMTX_COREO_INT_STATUS_1** Represents the status of the INTMTX sources numbered from 32 ~ 63. Each bit corresponds to one INTMTX source.

0: The corresponding INTMTX source triggered an interrupt

1: No INTMTX triggered

(RO)

Register 10.55. INTMTX_CORE0_INT_STATUS_2_REG (0x0110)



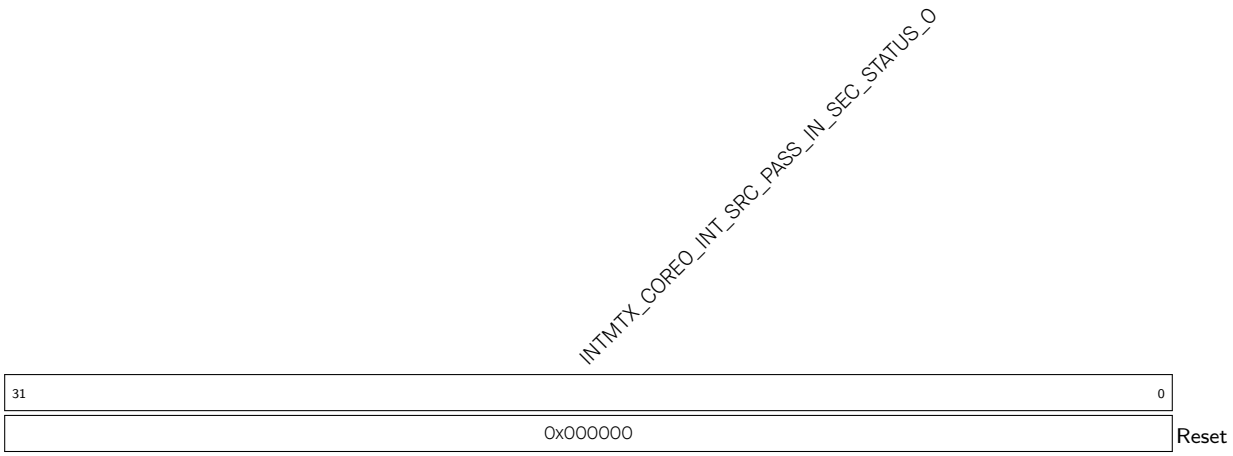

**INTMTX_CORE0_INT_STATUS_2** Represents the status of the INTMTX sources numbered from 64 ~ 65. Each bit corresponds to one INTMTX source.

0: The corresponding INTMTX source triggered an interrupt

1: No INTMTX triggered

(RO)

Register 10.56. INTMTX_CORE0_SRC_PASS_IN_SEC_STATUS_0_REG (0x0114)



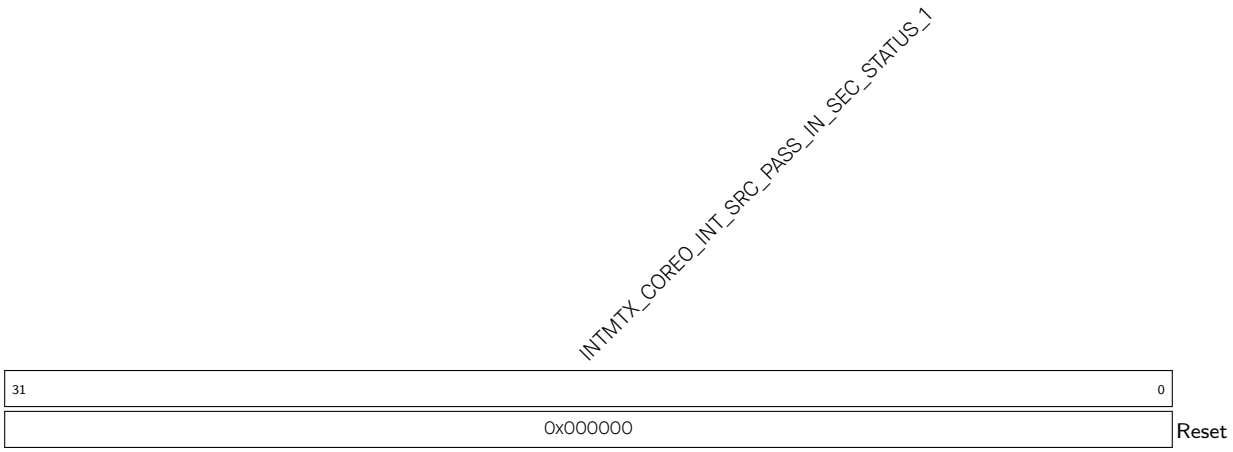

**INTMTX_CORE0_INT_SRC_PASS_IN_SEC_STATUS_0** Represents whether the INTMTX sources numbered from 0 ~ 31 are configured for delegated interrupts. Each bit represents the configuration status of one INTMTX source.

0: The corresponding INTMTX source is not configured for delegation.

1: The corresponding INTMTX source is configured for delegation.

(RO)

Register 10.57. INTMTX_COREO_SRC_PASS_IN_SEC_STATUS_1_REG (0x0118)



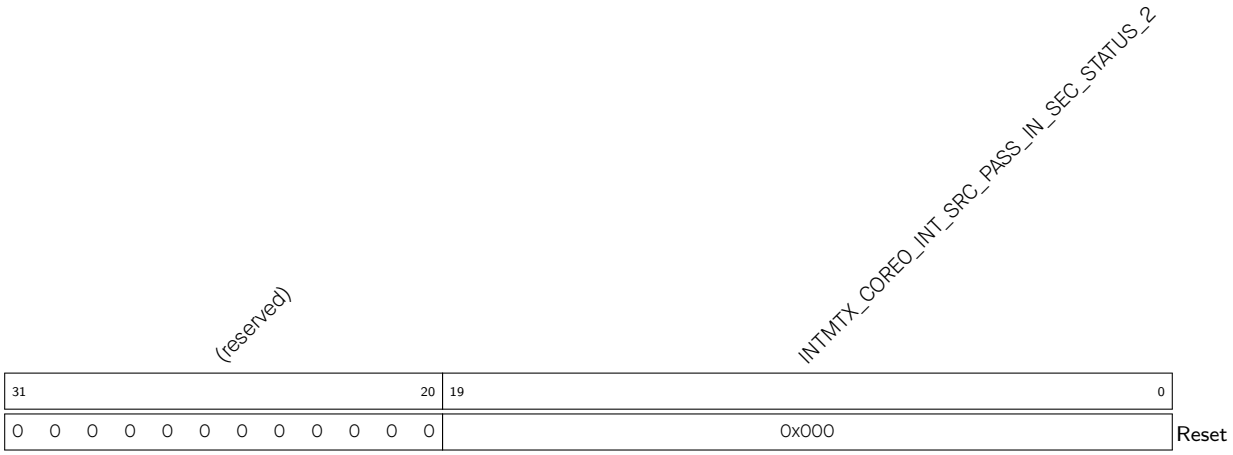

**INTMTX_COREO_INT_SRC_PASS_IN_SEC_STATUS_1** Represents whether the INTMTX sources numbered from 31 ~ 63 are configured for delegated interrupts. Each bit represents the configuration status of one INTMTX source.

0: The corresponding INTMTX source is not configured for delegation.

1: The corresponding INTMTX source is configured for delegation.

(RO)

Register 10.58. INTMTX_COREO_SRC_PASS_IN_SEC_STATUS_2_REG (0x011C)



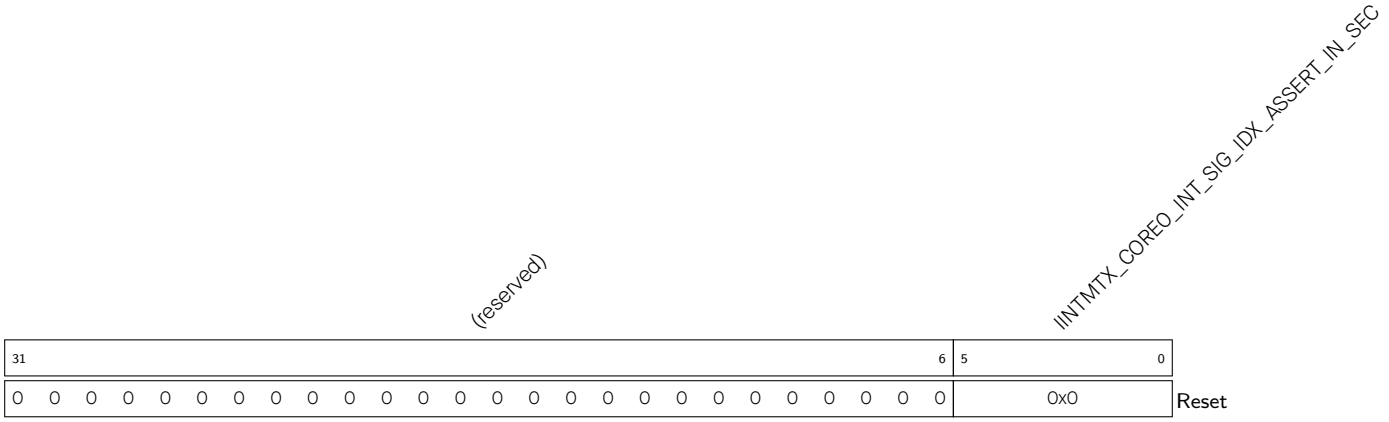

**INTMTX_COREO_INT_SRC_PASS_IN_SEC_STATUS_2** Represents whether the INTMTX sources numbered from 64 ~ 65 are configured for delegated interrupts. Each bit represents the configuration status of one INTMTX source.

0: The corresponding INTMTX source is not configured for delegation.

1: The corresponding INTMTX source is configured for delegation.

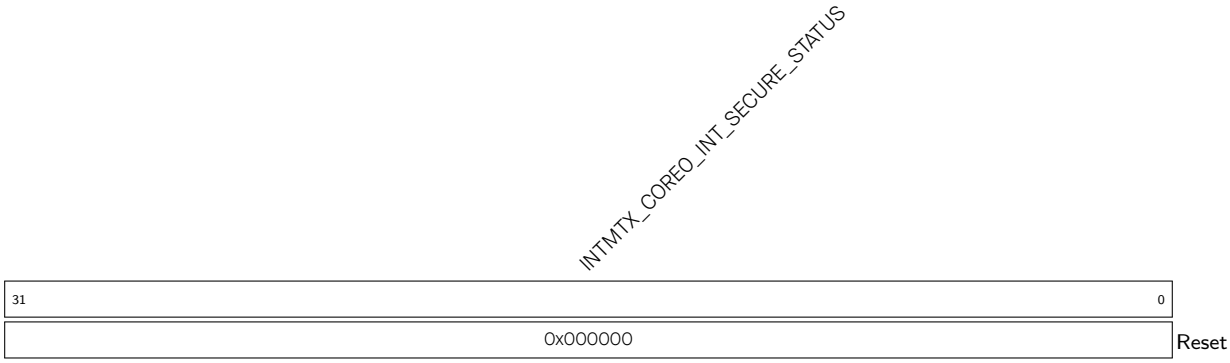
(RO)

Register 10.59. INTMTX_COREO_INT_SIG_IDX_ASSERT_IN_SEC_REG (0x0120)



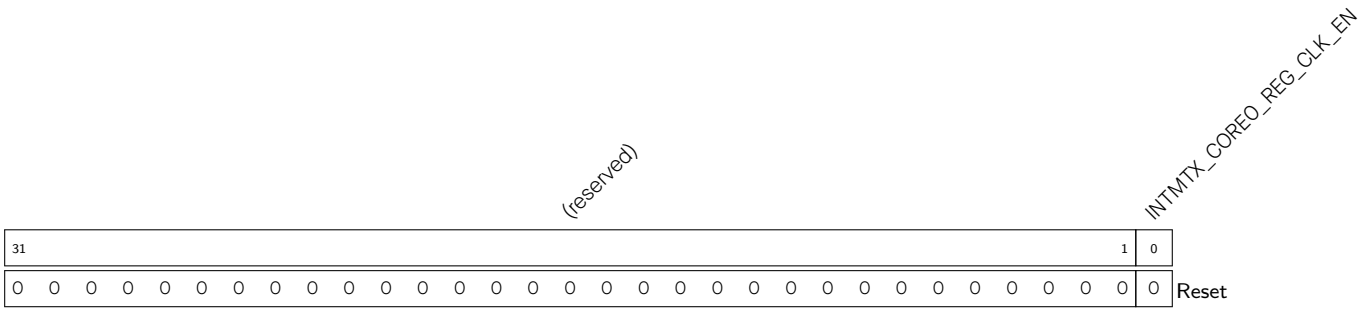

**INTMTX_COREO_INT_SIG_IDX_ASSERT_IN_SEC** Configure which CPU Machine Mode INTMTX source the interrupt should be delegated to. (R/W)

Register 10.60. INTMTX_COREO_SECURE_STATUS_REG (0x0124)



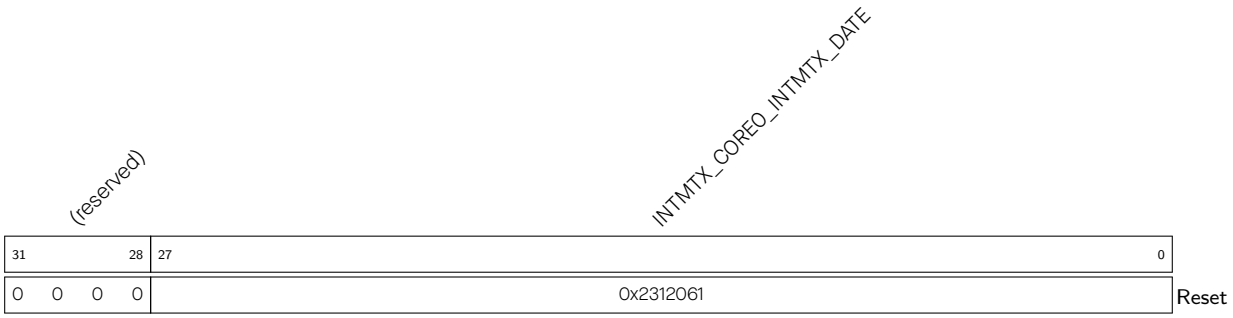

**INTMTX_COREO_INT_SECURE_STATUS** Represents which CPU User Mode interrupt the delegated Machine Mode interrupt was originally mapped to. (RO)

Register 10.61. INTMTX_COREO_CLOCK_GATE_REG (0x0128)




**INTMTX_COREO_REG_CLK_EN** INTMTX clock gating configure register. (R/W)

Register 10.62. INTMTX_COREO_INTMTX_DATE_REG (0x07FC)

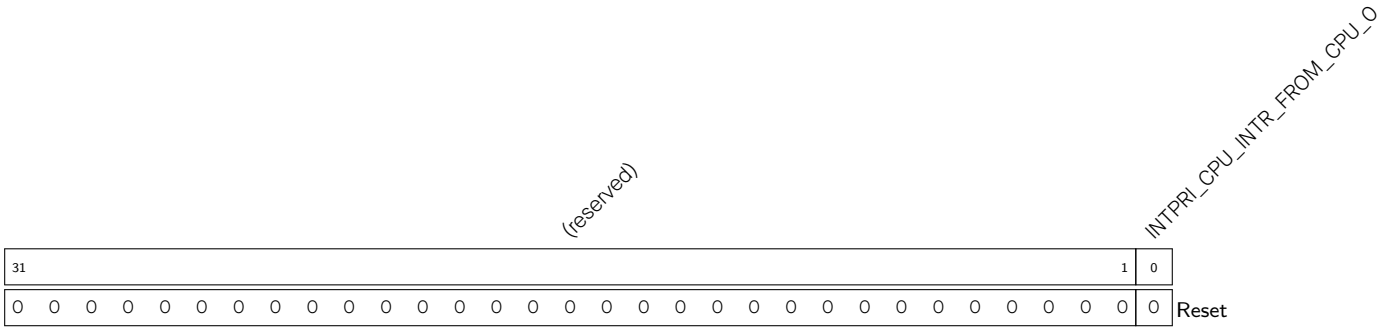



**INTMTX_COREO_INTMTX_DATE** Version control register. (R/W)

## 10.7.2 Software Interrupt Registers

The addresses in this section are relative to the software interrupt base address provided in Table 5.3-2 in Chapter 5 *System and Memory*.

Register 10.63. INTPRI_CPU_INTR_FROM_CPU_*n*_REG (*n*: 0-3) (0x0090+0x4**n*)



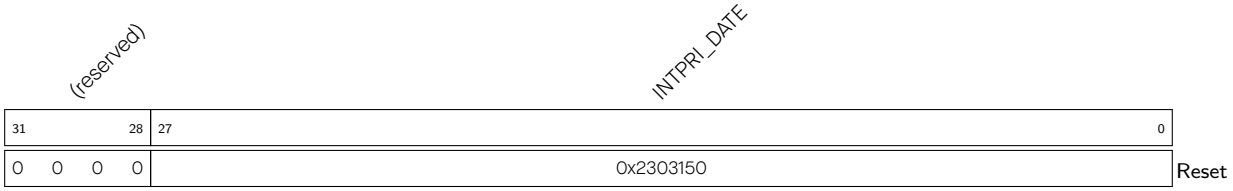

**INTPRI_CPU_INTR_FROM_CPU_*n*** CPU_INTR_FROM_CPU_*n* mapping register. Configures whether to generate interrupts by configuring the register through software.

0: Stop generating interrupts

1: Generate interrupts

(R/W)

Register 10.64. INTPRI_DATE_REG (0x00A0)




**INTPRI_DATE** Version control register. (R/W)

# Chapter 11

# Event Task Matrix (ETM)

## 11.1 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.

## 11.2 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 or disabled independently. If disabled, 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, temperature sensor, ADC, I2S, GDMA, 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).

## 11.3 Functional Description

### 11.3.1 Architecture

The Event Task Matrix has 50 independent channels. Each channel can select any event as input, and map the event to any task as output (see Section 11.3.2 and Section 11.3.3 respectively). Each channel has an individual enable bit (see Section 11.3.6).

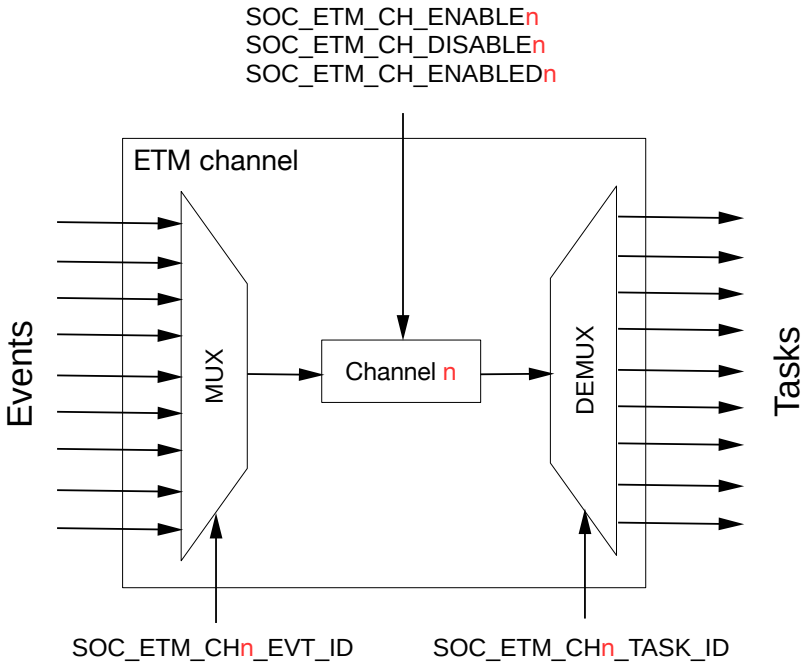



Figure 11.3-1. ETM Channel*n* Architecture

Figure 11.3-1 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*.

## 11.3.2 Events

An ETM channel can be set up to select one event to receive by configuring the SOC_ETM_CH*n*_EVT_ID field. Table 11.3-1 shows the configuration values of SOC_ETM_CH*n*_EVT_ID and their corresponding events.

Table 11.3-1. 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 | |
| 11 | GPIO_EVT_CH2_FALL_EDGE | |

| SOC_ETM_CH*n*_EVT_ID | Selected Event | Peripheral Generating This Event |
|---|---|---|
| 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_POS0 | |
| 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_OVF_CNT_PLS_CH0 | |
| 36 | LEDC_EVT_OVF_CNT_PLS_CH1 | |
| 37 | LEDC_EVT_OVF_CNT_PLS_CH2 | |
| 38 | LEDC_EVT_OVF_CNT_PLS_CH3 | |
| 39 | LEDC_EVT_OVF_CNT_PLS_CH4 | |
| 40 | LEDC_EVT_OVF_CNT_PLS_CH5 | |
| 41 | LEDC_EVT_TIME_OVF_TIMER0 | |
| 42 | LEDC_EVT_TIME_OVF_TIMER1 | |
| 43 | LEDC_EVT_TIME_OVF_TIMER2 | |
| 44 | LEDC_EVT_TIME_OVF_TIMER3 | |
| 45 | LEDC_EVT_TIMER0_CMP | |
| 46 | LEDC_EVT_TIMER1_CMP | |
| 47 | LEDC_EVT_TIMER2_CMP | |
| 48 | LEDC_EVT_TIMER3_CMP | |
| 49 | TG0_EVT_CNT_CMP_TIMER0 | General-purpose timer group 0 |
| 50 | TG0_EVT_CNT_CMP_TIMER1 | |
| 51 | TG1_EVT_CNT_CMP_TIMER0 | General-purpose timer group 1 |
| 52 | TG1_EVT_CNT_CMP_TIMER1 | |
| 53 | SYSTIMER_EVT_CNT_CMP0 | System Timer |
| 54 | SYSTIMER_EVT_CNT_CMP1 | |
| 55 | SYSTIMER_EVT_CNT_CMP2 | |
| 56 | ADC_EVT_CONV_CMPLT0 | ADC Controller |
| 57 | ADC_EVT_EQ_ABOVE_THRESH0 | |
| 58 | ADC_EVT_EQ_ABOVE_THRESH1 | |

| SOC_ETM_CH*n*_EVT_ID | Selected Event | Peripheral Generating This Event |
|---|---|---|
| 59 | ADC_EVT_EQ_BELOW_THRESH0 | |
| 60 | ADC_EVT_EQ_BELOW_THRESH1 | |
| 61 | ADC_EVT_RESULT_DONE0 | |
| 62 | ADC_EVT_STOPPED0 | |
| 63 | ADC_EVT_STARTED0 | |
| 72 | TMPSNSR_EVT_OVER_LIMIT | Temperature Sensor |
| 73 | I2S0_EVT_RX_DONE | I2S |
| 74 | I2S0_EVT_TX_DONE | |
| 75 | I2S0_EVT_X_WORDS_RECEIVED | |
| 76 | I2S0_EVT_X_WORDS_SENT | |
| 84 | RTC_EVT_TICK | RTC Timer |
| 85 | RTC_EVT_OVF | |
| 86 | RTC_EVT_CMP | |
| 87 | GDMA_EVT_IN_DONE_CH0 | GDMA Controller (GDMA) |
| 88 | GDMA_EVT_IN_DONE_CH1 | |
| 90 | GDMA_EVT_IN_SUC_EOF_CH0 | |
| 91 | GDMA_EVT_IN_SUC_EOF_CH1 | |
| 93 | GDMA_EVT_IN_FIFO_EMPTY_CH0 | |
| 94 | GDMA_EVT_IN_FIFO_EMPTY_CH1 | |
| 96 | GDMA_EVT_IN_FIFO_FULL_CH0 | |
| 97 | GDMA_EVT_IN_FIFO_FULL_CH1 | |
| 99 | GDMA_EVT_OUT_DONE_CH0 | |
| 100 | GDMA_EVT_OUT_DONE_CH1 | |
| 102 | GDMA_EVT_OUT_EOF_CH0 | |
| 103 | GDMA_EVT_OUT_EOF_CH1 | |
| 105 | GDMA_EVT_OUT_TOTAL_EOF_CH0 | |
| 106 | GDMA_EVT_OUT_TOTAL_EOF_CH1 | |
| 108 | GDMA_EVT_OUT_FIFO_EMPTY_CH0 | |
| 109 | GDMA_EVT_OUT_FIFO_EMPTY_CH1 | |
| 111 | GDMA_EVT_OUT_FIFO_FULL_CH0 | |
| 112 | GDMA_EVT_OUT_FIFO_FULL_CH1 | |
| 114 | PMU_EVT_SLEEP_WEEKUP | PMU |

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 corresponding to the peripheral that generates it.

### 11.3.3 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. Table 11.3-2 shows the configuration values of SOC_ETM_CH*n*_TASK_ID and their corresponding tasks.

Table 11.3-2. 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_TIMER0_CAP | |
| 36 | LEDC_TASK_TIMER1_CAP | |
| 37 | LEDC_TASK_TIMER2_CAP | |
| 38 | LEDC_TASK_TIMER3_CAP | |
| 39 | LEDC_TASK_SIG_OUT_DIS_CH0 | |
| 40 | LEDC_TASK_SIG_OUT_DIS_CH1 | |
| 41 | LEDC_TASK_SIG_OUT_DIS_CH2 | |
| 42 | LEDC_TASK_SIG_OUT_DIS_CH3 | |
| 43 | LEDC_TASK_SIG_OUT_DIS_CH4 | |

| SOC_ETM_CH*n*_TASK_ID | Mapped Task | Peripheral Receiving This Task |
|---|---|---|
| 44 | LEDC_TASK_SIG_OUT_DIS_CH5 | |
| 45 | LEDC_TASK_OVF_CNT_RST_CH0 | |
| 46 | LEDC_TASK_OVF_CNT_RST_CH1 | |
| 47 | LEDC_TASK_OVF_CNT_RST_CH2 | |
| 48 | LEDC_TASK_OVF_CNT_RST_CH3 | |
| 49 | LEDC_TASK_OVF_CNT_RST_CH4 | |
| 50 | LEDC_TASK_OVF_CNT_RST_CH5 | |
| 51 | LEDC_TASK_TIMER0_RST | |
| 52 | LEDC_TASK_TIMER1_RST | |
| 53 | LEDC_TASK_TIMER2_RST | |
| 54 | LEDC_TASK_TIMER3_RST | |
| 55 | LEDC_TASK_TIMER0_RESUME | |
| 56 | LEDC_TASK_TIMER1_RESUME | |
| 57 | LEDC_TASK_TIMER2_RESUME | |
| 58 | LEDC_TASK_TIMER3_RESUME | |
| 59 | LEDC_TASK_TIMER0_PAUSE | |
| 60 | LEDC_TASK_TIMER1_PAUSE | |
| 61 | LEDC_TASK_TIMER2_PAUSE | |
| 62 | LEDC_TASK_TIMER3_PAUSE | |
| 63 | LEDC_TASK_GAMMA_RESTART_CH0 | |
| 64 | LEDC_TASK_GAMMA_RESTART_CH1 | |
| 65 | LEDC_TASK_GAMMA_RESTART_CH2 | |
| 66 | LEDC_TASK_GAMMA_RESTART_CH3 | |
| 67 | LEDC_TASK_GAMMA_RESTART_CH4 | |
| 68 | LEDC_TASK_GAMMA_RESTART_CH5 | |
| 69 | LEDC_TASK_GAMMA_PAUSE_CH0 | |
| 70 | LEDC_TASK_GAMMA_PAUSE_CH1 | |
| 71 | LEDC_TASK_GAMMA_PAUSE_CH2 | |
| 72 | LEDC_TASK_GAMMA_PAUSE_CH3 | |
| 73 | LEDC_TASK_GAMMA_PAUSE_CH4 | |
| 74 | LEDC_TASK_GAMMA_PAUSE_CH5 | |
| 75 | LEDC_TASK_GAMMA_RESUME_CH0 | |
| 76 | LEDC_TASK_GAMMA_RESUME_CH1 | |
| 77 | LEDC_TASK_GAMMA_RESUME_CH2 | |
| 78 | LEDC_TASK_GAMMA_RESUME_CH3 | |
| 79 | LEDC_TASK_GAMMA_RESUME_CH4 | |
| 80 | LEDC_TASK_GAMMA_RESUME_CH5 | |
| 81 | TG0_TASK_CNT_START_TIMER0 | General-purpose timer group 0 |
| 82 | TG0_TASK_ALARM_START_TIMER0 | |
| 83 | TG0_TASK_CNT_STOP_TIMER0 | |
| 84 | TG0_TASK_CNT_RELOAD_TIMER0 | |
| 85 | TG0_TASK_CNT_CAP_TIMER0 | |
| 86 | TG0_TASK_CNT_START_TIMER1 | |
| 87 | TG0_TASK_ALARM_START_TIMER1 | |
| 88 | TG0_TASK_CNT_STOP_TIMER1 | |

| SOC_ETM_CH*n*_TASK_ID | Mapped Task | Peripheral Receiving This Task |
|---|---|---|
| 89 | TG0_TASK_CNT_RELOAD_TIMER1 | |
| 90 | TG0_TASK_CNT_CAP_TIMER1 | |
| 91 | TG1_TASK_CNT_START_TIMER0 | General-purpose timer group 1 |
| 92 | TG1_TASK_ALARM_START_TIMER0 | |
| 93 | TG1_TASK_CNT_STOP_TIMER0 | |
| 94 | TG1_TASK_CNT_RELOAD_TIMER0 | |
| 95 | TG1_TASK_CNT_CAP_TIMER0 | |
| 96 | TG1_TASK_CNT_START_TIMER1 | |
| 97 | TG1_TASK_ALARM_START_TIMER1 | |
| 98 | TG1_TASK_CNT_STOP_TIMER1 | |
| 99 | TG1_TASK_CNT_RELOAD_TIMER1 | |
| 100 | TG1_TASK_CNT_CAP_TIMER1 | |
| 101 | ADC_TASK_SAMPLE0 | ADC Controller |
| 103 | ADC_TASK_START0 | |
| 104 | ADC_TASK_STOP0 | |
| 109 | TMPSNSR_TASK_START_SAMPLE | Temperature Sensor |
| 110 | TMPSNSR_TASK_STOP_SAMPLE | |
| 111 | I2S0_TASK_START_RX | I2S |
| 112 | I2S0_TASK_START_TX | |
| 113 | I2S0_TASK_STOP_RX | |
| 114 | I2S0_TASK_STOP_TX | |
| 125 | GDMA_TASK_IN_START_CH0 | GDMA Controller (GDMA) |
| 126 | GDMA_TASK_IN_START_CH1 | |
| 128 | GDMA_TASK_OUT_START_CH0 | |
| 129 | GDMA_TASK_OUT_START_CH1 | |
| 131 | PMU_TASK_SLEEP_REQ | PMU |

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 will be generated. If events received by multiple channels are valid at the same time, the task will be generated only once.

### 11.3.4 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.

### 11.3.5 Timing Considerations

Figure 11.3-2 shows the structure of clocks that drive received events, sent tasks, and ETM channels.

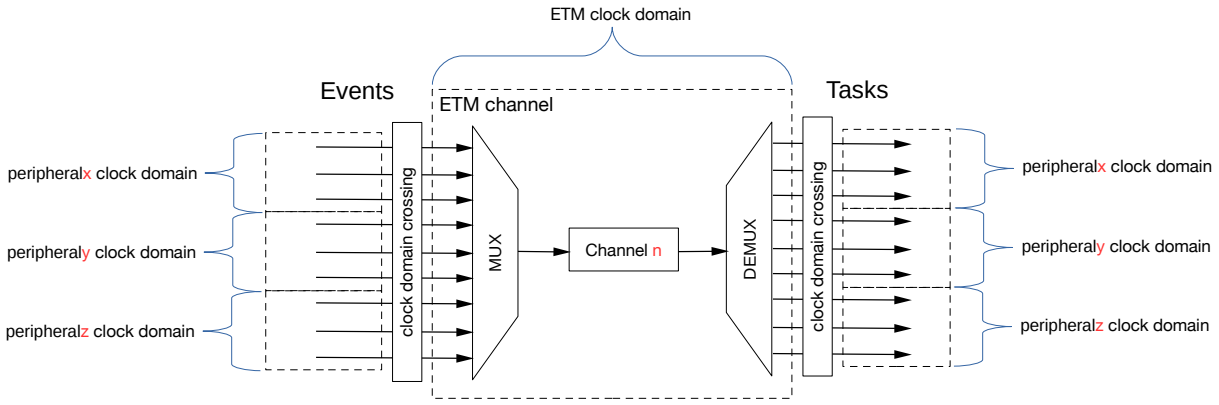



Figure 11.3-2. Event Task Matrix Clock Architecture

ETM is running at the AHB_CLK domain (see Chapter 8 *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 the peripherals receiving tasks do not necessarily operate on the same clock and therefore require synchronization. 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 (PLL_F80M_CLK), and the ETM runs in the 40 MHz clock domain (AHB_CLK). Peripheral A generates event 1, which is received by the ETM. To ensure successful reception of each event 1, the interval between two consecutive occurrences of event 1 must exceed two clock cycles of Peripheral A (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 (RC_FAST_CLK), and the ETM runs in the 40 MHz clock domain (AHB_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 (PLL_F80M_CLK), peripheral B is in the 20 MHz clock domain (RC_FAST_CLK), and the ETM runs in the 40 MHz clock domain (AHB_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.

### 11.3.6 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 PCR_ETM_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 further 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 PCR_ETM_RST_EN field

## 11.4 Register Summary

The addresses in this section are relative to Event Task Matrix base address provided in Table 5.3-2 in Chapter *5 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 enable status register | 0x0000 | R/WTC/WS |
| SOC_ETM_CH_ENA_AD1_REG | Channel enable status register | 0x000C | R/WTC/WS |
| SOC_ETM_EVT_ST0_REG | Event trigger status register | 0x01A8 | R/WTC/SS |
| SOC_ETM_EVT_ST1_REG | Event trigger status register | 0x01B0 | R/WTC/SS |
| SOC_ETM_EVT_ST2_REG | Event trigger status register | 0x01B8 | R/WTC/SS |
| SOC_ETM_EVT_ST3_REG | Event trigger status register | 0x01C0 | R/WTC/SS |
| SOC_ETM_TASK_ST0_REG | Task trigger status register | 0x01C8 | R/WTC/SS |
| SOC_ETM_TASK_ST1_REG | Task trigger status register | 0x01D0 | R/WTC/SS |
| SOC_ETM_TASK_ST2_REG | Task trigger status register | 0x01D8 | R/WTC/SS |
| SOC_ETM_TASK_ST3_REG | Task trigger status register | 0x01E0 | R/WTC/SS |
| SOC_ETM_TASK_ST4_REG | Task trigger status register | 0x01E8 | 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 |
| 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 |

| Name | Description | Address | Access |
|---|---|---|---|
| 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 |
| 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 |

| Name | Description | Address | Access |
|---|---|---|---|
| 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 |
| SOC_ETM_EVT_ST0_CLR_REG | Event trigger status clear register | 0x01AC | WT |
| SOC_ETM_EVT_ST1_CLR_REG | Event trigger status clear register | 0x01B4 | WT |
| SOC_ETM_EVT_ST2_CLR_REG | Event trigger status clear register | 0x01BC | WT |
| SOC_ETM_EVT_ST3_CLR_REG | Event trigger status clear register | 0x01C4 | WT |
| SOC_ETM_TASK_ST0_CLR_REG | Task trigger status clear register | 0x01CC | WT |
| SOC_ETM_TASK_ST1_CLR_REG | Task trigger status clear register | 0x01D4 | WT |

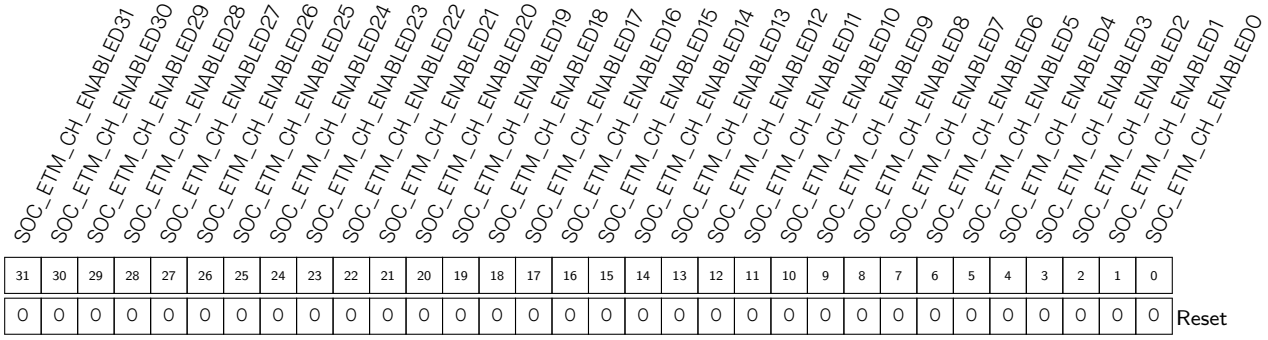
| Name | Description | Address | Access |
|---|---|---|---|
| SOC_ETM_TASK_ST2_CLR_REG | Task trigger status clear register | 0x01DC | WT |
| SOC_ETM_TASK_ST3_CLR_REG | Task trigger status clear register | 0x01E4 | WT |
| SOC_ETM_TASK_ST4_CLR_REG | Task trigger status clear register | 0x01EC | WT |
| SOC_ETM_CLK_EN_REG | ETM clock enable register | 0x01F0 | R/W |
| **Version Register** | | | |
| SOC_ETM_DATE_REG | Version control register | 0x01F4 | R/W |

## 11.5 Registers

The addresses in this section are relative to Event Task Matrix base address provided in Table 5.3-2 in Chapter *5 System and Memory*.

For how to program reserved fields, please refer to Section Programming Reserved Register Field.

**Register 11.1. SOC_ETM_CH_ENA_AD0_REG (0x0000)**



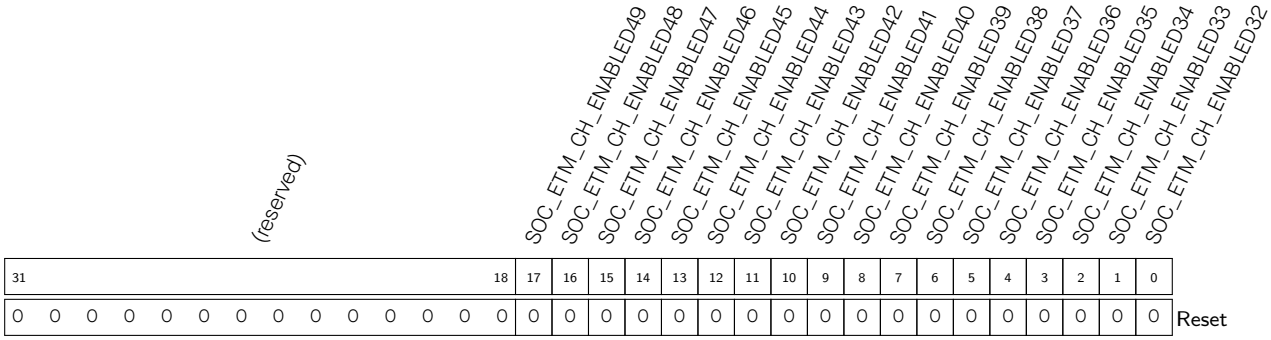

**SOC_ETM_CH_ENABLED*n* (*n*: 0-31)** Represents channel*n* enable status.

0: Disable

1: Enable

(R/WTC/WS)

**Register 11.2. SOC_ETM_CH_ENA_AD1_REG (0x000C)**



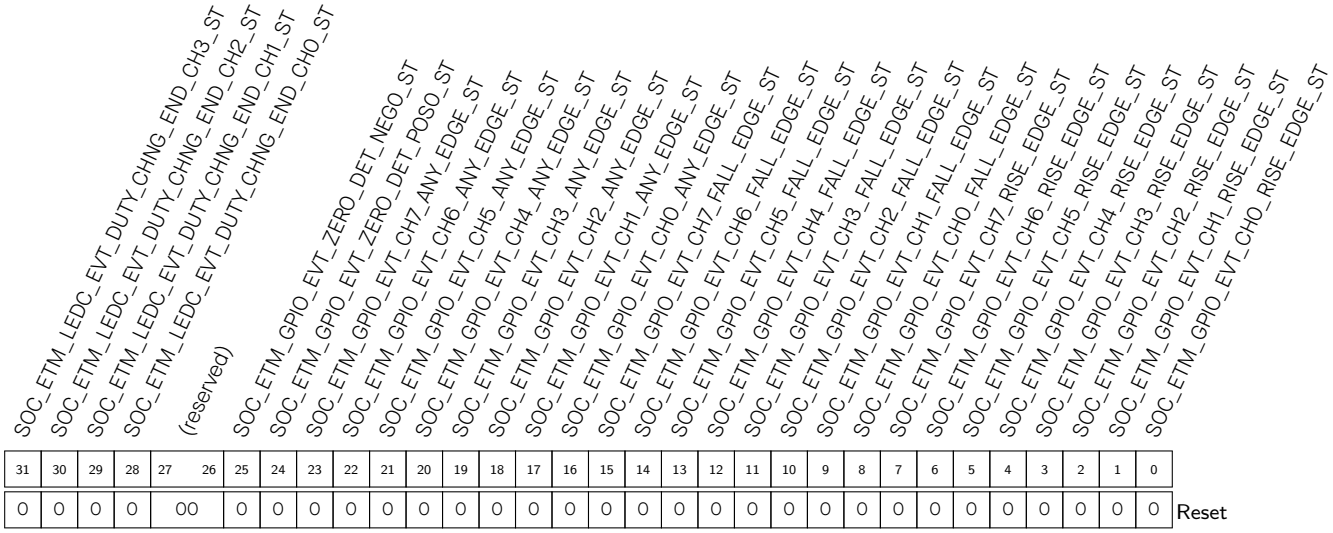

**SOC_ETM_CH_ENABLED*n* (*n*: 32-49)** Represents channel*n* enable status.

0: Disable

1: Enable

(R/WTC/WS)

**Register 11.3. SOC_ETM_EVT_ST0_REG (0x01A8)**




**SOC_ETM_GPIO_EVT_CH0_RISE_EDGE_ST** Represents GPIO_EVT_CH0_RISE_EDGE trigger status.
0: Not triggered
1: Triggered
(R/WTC/SS)

**SOC_ETM_GPIO_EVT_CH1_RISE_EDGE_ST** Represents GPIO_EVT_CH1_RISE_EDGE trigger status.
0: Not triggered
1: Triggered
(R/WTC/SS)

**SOC_ETM_GPIO_EVT_CH2_RISE_EDGE_ST** Represents GPIO_EVT_CH2_RISE_EDGE trigger status.
0: Not triggered
1: Triggered
(R/WTC/SS)

**SOC_ETM_GPIO_EVT_CH3_RISE_EDGE_ST** Represents GPIO_EVT_CH3_RISE_EDGE trigger status.
0: Not triggered
1: Triggered
(R/WTC/SS)

**SOC_ETM_GPIO_EVT_CH4_RISE_EDGE_ST** Represents GPIO_EVT_CH4_RISE_EDGE trigger status.
0: Not triggered
1: Triggered
(R/WTC/SS)

Continued on the next page...

Register 11.3. SOC_ETM_EVT_ST0_REG (0x01A8)

Continued from the previous page...

**SOC_ETM_GPIO_EVT_CH5_RISE_EDGE_ST** Represents GPIO_EVT_CH5_RISE_EDGE trigger status.
0: Not triggered
1: Triggered
(R/WTC/SS)

**SOC_ETM_GPIO_EVT_CH6_RISE_EDGE_ST** Represents GPIO_EVT_CH6_RISE_EDGE trigger status.
0: Not triggered
1: Triggered
(R/WTC/SS)

**SOC_ETM_GPIO_EVT_CH7_RISE_EDGE_ST** Represents GPIO_EVT_CH7_RISE_EDGE trigger status.
0: Not triggered
1: Triggered
(R/WTC/SS)

**SOC_ETM_GPIO_EVT_CH0_FALL_EDGE_ST** Represents GPIO_EVT_CH0_FALL_EDGE trigger status.
0: Not triggered
1: Triggered
(R/WTC/SS)

**SOC_ETM_GPIO_EVT_CH1_FALL_EDGE_ST** Represents GPIO_EVT_CH1_FALL_EDGE trigger status.
0: Not triggered
1: Triggered
(R/WTC/SS)

**SOC_ETM_GPIO_EVT_CH2_FALL_EDGE_ST** Represents GPIO_EVT_CH2_FALL_EDGE trigger status.
0: Not triggered
1: Triggered
(R/WTC/SS)

**SOC_ETM_GPIO_EVT_CH3_FALL_EDGE_ST** Represents GPIO_EVT_CH3_FALL_EDGE trigger status.
0: Not triggered
1: Triggered
(R/WTC/SS)

Continued on the next page...

Register 11.3. SOC_ETM_EVT_ST0_REG (0x01A8)

Continued from the previous page...

**SOC_ETM_GPIO_EVT_CH4_FALL_EDGE_ST** Represents GPIO_EVT_CH4_FALL_EDGE trigger status.
0: Not triggered
1: Triggered
(R/WTC/SS)

**SOC_ETM_GPIO_EVT_CH5_FALL_EDGE_ST** Represents GPIO_EVT_CH5_FALL_EDGE trigger status.
0: Not triggered
1: Triggered
(R/WTC/SS)

**SOC_ETM_GPIO_EVT_CH6_FALL_EDGE_ST** Represents GPIO_EVT_CH6_FALL_EDGE trigger status.
0: Not triggered
1: Triggered
(R/WTC/SS)

**SOC_ETM_GPIO_EVT_CH7_FALL_EDGE_ST** Represents GPIO_EVT_CH7_FALL_EDGE trigger status.
0: Not triggered
1: Triggered
(R/WTC/SS)

**SOC_ETM_GPIO_EVT_CH0_ANY_EDGE_ST** Represents GPIO_EVT_CH0_ANY_EDGE trigger status.
0: Not triggered
1: Triggered
(R/WTC/SS)

**SOC_ETM_GPIO_EVT_CH1_ANY_EDGE_ST** Represents GPIO_EVT_CH1_ANY_EDGE trigger status.
0: Not triggered
1: Triggered
(R/WTC/SS)

**SOC_ETM_GPIO_EVT_CH2_ANY_EDGE_ST** Represents GPIO_EVT_CH2_ANY_EDGE trigger status.
0: Not triggered
1: Triggered
(R/WTC/SS)

**SOC_ETM_GPIO_EVT_CH3_ANY_EDGE_ST** Represents GPIO_EVT_CH3_ANY_EDGE trigger status.
0: Not triggered
1: Triggered
(R/WTC/SS)

Continued on the next page...

Register 11.3. SOC_ETM_EVT_ST0_REG (0x01A8)

Continued from the previous page...

**SOC_ETM_GPIO_EVT_CH4_ANY_EDGE_ST** Represents GPIO_EVT_CH4_ANY_EDGE trigger status.
0: Not triggered
1: Triggered
(R/WTC/SS)

**SOC_ETM_GPIO_EVT_CH5_ANY_EDGE_ST** Represents GPIO_EVT_CH5_ANY_EDGE trigger status.
0: Not triggered
1: Triggered
(R/WTC/SS)

**SOC_ETM_GPIO_EVT_CH6_ANY_EDGE_ST** Represents GPIO_EVT_CH6_ANY_EDGE trigger status.
0: Not triggered
1: Triggered
(R/WTC/SS)

**SOC_ETM_GPIO_EVT_CH7_ANY_EDGE_ST** Represents GPIO_EVT_CH7_ANY_EDGE trigger status.
0: Not triggered
1: Triggered
(R/WTC/SS)

**SOC_ETM_GPIO_EVT_ZERO_DET_POS0_ST** Represents GPIO_EVT_ZERO_DET_POS0 trigger status.
0: Not triggered
1: Triggered
(R/WTC/SS)

**SOC_ETM_GPIO_EVT_ZERO_DET_NEG0_ST** Represents GPIO_EVT_ZERO_DET_NEG0 trigger status.
0: Not triggered
1: Triggered
(R/WTC/SS)

**SOC_ETM_LEDC_EVT_DUTY_CHNG_END_CH0_ST** Represents LEDC_EVT_DUTY_CHNG_END_CH0 trigger status.
0: Not triggered
1: Triggered
(R/WTC/SS)

Continued on the next page...

Register 11.3. SOC_ETM_EVT_ST0_REG (0x01A8)

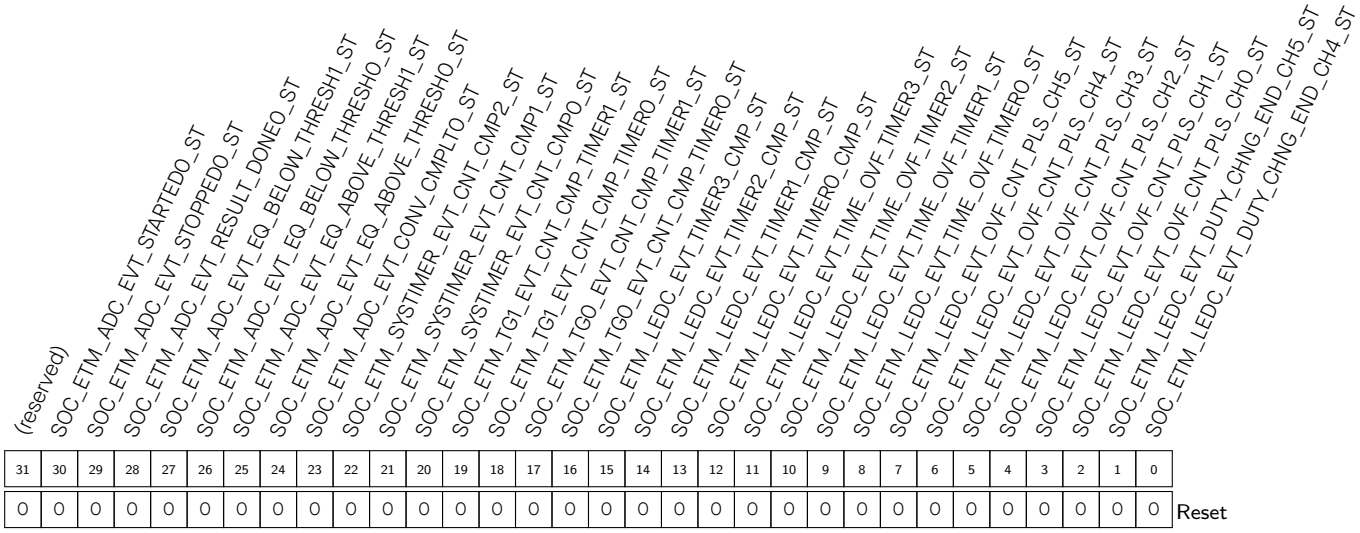
Continued from the previous page...

**SOC_ETM_LEDC_EVT_DUTY_CHNG_END_CH1_ST** Represents LEDC_EVT_DUTY_CHNG_END_CH1 trigger status.
0: Not triggered
1: Triggered
(R/WTC/SS)

**SOC_ETM_LEDC_EVT_DUTY_CHNG_END_CH2_ST** Represents LEDC_EVT_DUTY_CHNG_END_CH2 trigger status.
0: Not triggered
1: Triggered
(R/WTC/SS)

**SOC_ETM_LEDC_EVT_DUTY_CHNG_END_CH3_ST** Represents LEDC_EVT_DUTY_CHNG_END_CH3 trigger status.
0: Not triggered
1: Triggered
(R/WTC/SS)

Register 11.4. SOC_ETM_EVT_ST1_REG (0x01B0)




**SOC_ETM_LEDC_EVT_DUTY_CHNG_END_CH4_ST** Represents LEDC_EVT_DUTY_CHNG_END_CH4 trigger status.

0: Not triggered

1: Triggered

(R/WTC/SS)

**SOC_ETM_LEDC_EVT_DUTY_CHNG_END_CH5_ST** Represents LEDC_EVT_DUTY_CHNG_END_CH5 trigger status.

0: Not triggered

1: Triggered

(R/WTC/SS)

**SOC_ETM_LEDC_EVT_OVF_CNT_PLS_CHO_ST** Represents LEDC_EVT_OVF_CNT_PLS_CHO trigger status.

0: Not triggered

1: Triggered

(R/WTC/SS)

**SOC_ETM_LEDC_EVT_OVF_CNT_PLS_CH1_ST** Represents LEDC_EVT_OVF_CNT_PLS_CH1 trigger status.

0: Not triggered

1: Triggered

(R/WTC/SS)

**SOC_ETM_LEDC_EVT_OVF_CNT_PLS_CH2_ST** Represents LEDC_EVT_OVF_CNT_PLS_CH2 trigger status.

0: Not triggered

1: Triggered

(R/WTC/SS)

Continued on the next page...

Register 11.4. SOC_ETM_EVT_ST1_REG (0x01B0)

Continued from the previous page...

**SOC_ETM_LEDC_EVT_OVF_CNT_PLS_CH3_ST** Represents LEDC_EVT_OVF_CNT_PLS_CH3 trigger status.
0: Not triggered
1: Triggered
(R/WTC/SS)

**SOC_ETM_LEDC_EVT_OVF_CNT_PLS_CH4_ST** Represents LEDC_EVT_OVF_CNT_PLS_CH4 trigger status.
0: Not triggered
1: Triggered
(R/WTC/SS)

**SOC_ETM_LEDC_EVT_OVF_CNT_PLS_CH5_ST** Represents LEDC_EVT_OVF_CNT_PLS_CH5 trigger status.
0: Not triggered
1: Triggered
(R/WTC/SS)

**SOC_ETM_LEDC_EVT_TIME_OVF_TIMER0_ST** Represents LEDC_EVT_TIME_OVF_TIMER0 trigger status.
0: Not triggered
1: Triggered
(R/WTC/SS)

**SOC_ETM_LEDC_EVT_TIME_OVF_TIMER1_ST** Represents LEDC_EVT_TIME_OVF_TIMER1 trigger status.
0: Not triggered
1: Triggered
(R/WTC/SS)

**SOC_ETM_LEDC_EVT_TIME_OVF_TIMER2_ST** Represents LEDC_EVT_TIME_OVF_TIMER2 trigger status.
0: Not triggered
1: Triggered
(R/WTC/SS)

**SOC_ETM_LEDC_EVT_TIME_OVF_TIMER3_ST** Represents LEDC_EVT_TIME_OVF_TIMER3 trigger status.
0: Not triggered
1: Triggered
(R/WTC/SS)

Continued on the next page...

Register 11.4. SOC_ETM_EVT_ST1_REG (0x01B0)

Continued from the previous page...

**SOC_ETM_LEDC_EVT_TIMER0_CMP_ST** Represents LEDC_EVT_TIMER0_CMP trigger status.
0: Not triggered
1: Triggered
(R/WTC/SS)

**SOC_ETM_LEDC_EVT_TIMER1_CMP_ST** Represents LEDC_EVT_TIMER1_CMP trigger status.
0: Not triggered
1: Triggered
(R/WTC/SS)

**SOC_ETM_LEDC_EVT_TIMER2_CMP_ST** Represents LEDC_EVT_TIMER2_CMP trigger status.
0: Not triggered
1: Triggered
(R/WTC/SS)

**SOC_ETM_LEDC_EVT_TIMER3_CMP_ST** Represents LEDC_EVT_TIMER3_CMP trigger status.
0: Not triggered
1: Triggered
(R/WTC/SS)

**SOC_ETM_TG0_EVT_CNT_CMP_TIMER0_ST** Represents TG0_EVT_CNT_CMP_TIMER0 trigger status.
0: Not triggered
1: Triggered
(R/WTC/SS)

**SOC_ETM_TG0_EVT_CNT_CMP_TIMER1_ST** Represents TG0_EVT_CNT_CMP_TIMER1 trigger status.
0: Not triggered
1: Triggered
(R/WTC/SS)

**SOC_ETM_TG1_EVT_CNT_CMP_TIMER0_ST** Represents TG1_EVT_CNT_CMP_TIMER0 trigger status.
0: Not triggered
1: Triggered
(R/WTC/SS)

**SOC_ETM_TG1_EVT_CNT_CMP_TIMER1_ST** Represents TG1_EVT_CNT_CMP_TIMER1 trigger status.
0: Not triggered
1: Triggered
(R/WTC/SS)

Continued on the next page...

Register 11.4. SOC_ETM_EVT_ST1_REG (0x01B0)

Continued from the previous page...

**SOC_ETM_SYSTIMER_EVT_CNT_CMP0_ST** Represents SYSTIMER_EVT_CNT_CMP0 trigger status.
0: Not triggered
1: Triggered
(R/WTC/SS)

**SOC_ETM_SYSTIMER_EVT_CNT_CMP1_ST** Represents SYSTIMER_EVT_CNT_CMP1 trigger status.
0: Not triggered
1: Triggered
(R/WTC/SS)

**SOC_ETM_SYSTIMER_EVT_CNT_CMP2_ST** Represents SYSTIMER_EVT_CNT_CMP2 trigger status.
0: Not triggered
1: Triggered
(R/WTC/SS)

**SOC_ETM_ADC_EVT_CONV_CMPLT0_ST** Represents ADC_EVT_CONV_CMPLT0 trigger status.
0: Not triggered
1: Triggered
(R/WTC/SS)

**SOC_ETM_ADC_EVT_EQ_ABOVE_THRESH0_ST** Represents ADC_EVT_EQ_ABOVE_THRESH0 trigger status.
0: Not triggered
1: Triggered
(R/WTC/SS)

**SOC_ETM_ADC_EVT_EQ_ABOVE_THRESH1_ST** Represents ADC_EVT_EQ_ABOVE_THRESH1 trigger status.
0: Not triggered
1: Triggered
(R/WTC/SS)

**SOC_ETM_ADC_EVT_EQ_BELOW_THRESH0_ST** Represents ADC_EVT_EQ_BELOW_THRESH0 trigger status.
0: Not triggered
1: Triggered
(R/WTC/SS)

Continued on the next page...

Register 11.4. SOC_ETM_EVT_ST1_REG (0x01B0)

Continued from the previous page...

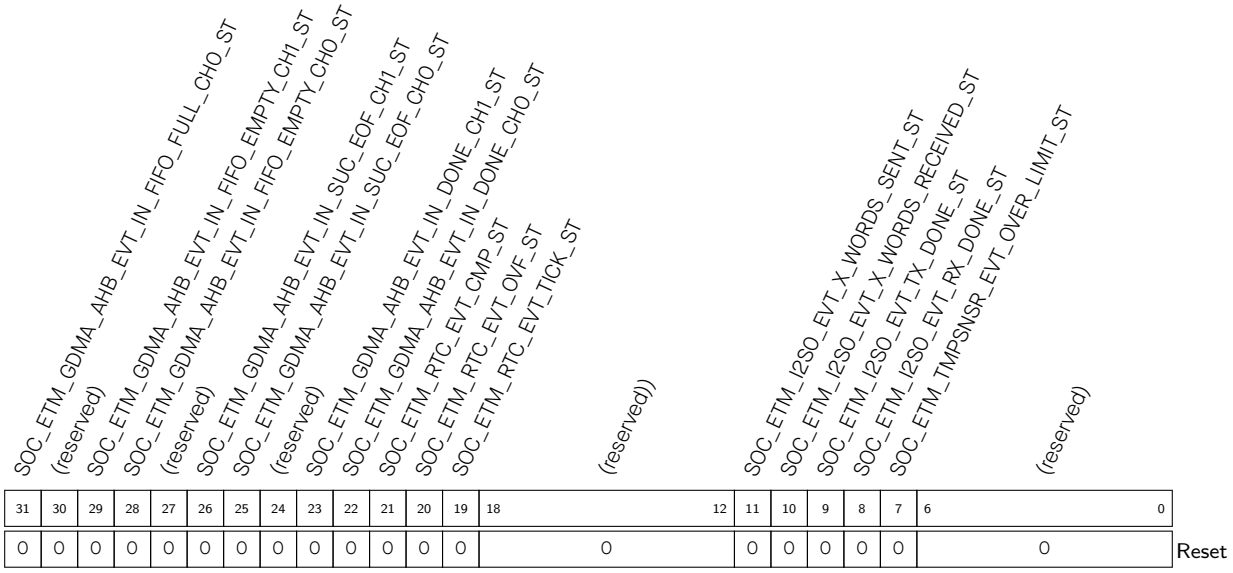
**SOC_ETM_ADC_EVT_EQ_BELOW_THRESH1_ST** Represents ADC_EVT_EQ_BELOW_THRESH1 trigger status.
0: Not triggered
1: Triggered
(R/WTC/SS)

**SOC_ETM_ADC_EVT_RESULT_DONE0_ST** Represents ADC_EVT_RESULT_DONE0 trigger status.
0: Not triggered
1: Triggered
(R/WTC/SS)

**SOC_ETM_ADC_EVT_STOPPED0_ST** Represents ADC_EVT_STOPPED0 trigger status.
0: Not triggered
1: Triggered
(R/WTC/SS)

**SOC_ETM_ADC_EVT_STARTED0_ST** Represents ADC_EVT_STARTED0 trigger status.
0: Not triggered
1: Triggered
(R/WTC/SS)

Register 11.5. SOC_ETM_EVT_ST2_REG (0x01B8)




**SOC_ETM_TMPSNSR_EVT_OVER_LIMIT_ST** Represents TMPSNSR_EVT_OVER_LIMIT trigger status.

0: Not triggered

1: Triggered

(R/WTC/SS)

**SOC_ETM_I2S0_EVT_RX_DONE_ST** Represents I2S0_EVT_RX_DONE trigger status.

0: Not triggered

1: Triggered

(R/WTC/SS)

**SOC_ETM_I2S0_EVT_TX_DONE_ST** Represents I2S0_EVT_TX_DONE trigger status.

0: Not triggered

1: Triggered

(R/WTC/SS)

**SOC_ETM_I2S0_EVT_X_WORDS_RECEIVED_ST** Represents I2S0_EVT_X_WORDS_RECEIVED trigger status.

0: Not triggered

1: Triggered

(R/WTC/SS)

**SOC_ETM_I2S0_EVT_X_WORDS_SENT_ST** Represents I2S0_EVT_X_WORDS_SENT trigger status.

0: Not triggered

1: Triggered

(R/WTC/SS)

**SOC_ETM_RTC_EVT_TICK_ST** Represents RTC_EVT_TICK trigger status.

0: Not triggered

1: Triggered

(R/WTC/SS)

Continued on the next page...

Register 11.5. SOC_ETM_EVT_ST2_REG (0x01B8)

Continued from the previous page...

**SOC_ETM_RTC_EVT_OVF_ST** Represents RTC_EVT_OVF trigger status.
0: Not triggered
1: Triggered
(R/WTC/SS)

**SOC_ETM_RTC_EVT_CMP_ST** Represents RTC_EVT_CMP trigger status.
0: Not triggered
1: Triggered
(R/WTC/SS)

**SOC_ETM_GDMA_AHB_EVT_IN_DONE_CH0_ST** Represents GDMA_AHB_EVT_IN_DONE_CH0 trigger status.
0: Not triggered
1: Triggered
(R/WTC/SS)

**SOC_ETM_GDMA_AHB_EVT_IN_DONE_CH1_ST** Represents GDMA_AHB_EVT_IN_DONE_CH1 trigger status.
0: Not triggered
1: Triggered
(R/WTC/SS)

**SOC_ETM_GDMA_AHB_EVT_IN_SUC_EOF_CH0_ST** Represents GDMA_AHB_EVT_IN_SUC_EOF_CH0 trigger status.
0: Not triggered
1: Triggered
(R/WTC/SS)

**SOC_ETM_GDMA_AHB_EVT_IN_SUC_EOF_CH1_ST** Represents GDMA_AHB_EVT_IN_SUC_EOF_CH1 trigger status.
0: Not triggered
1: Triggered
(R/WTC/SS)

**SOC_ETM_GDMA_AHB_EVT_IN_FIFO_EMPTY_CH0_ST** Represents GDMA_AHB_EVT_IN_FIFO_EMPTY_CH0 trigger status.
0: Not triggered
1: Triggered
(R/WTC/SS)

Continued on the next page...

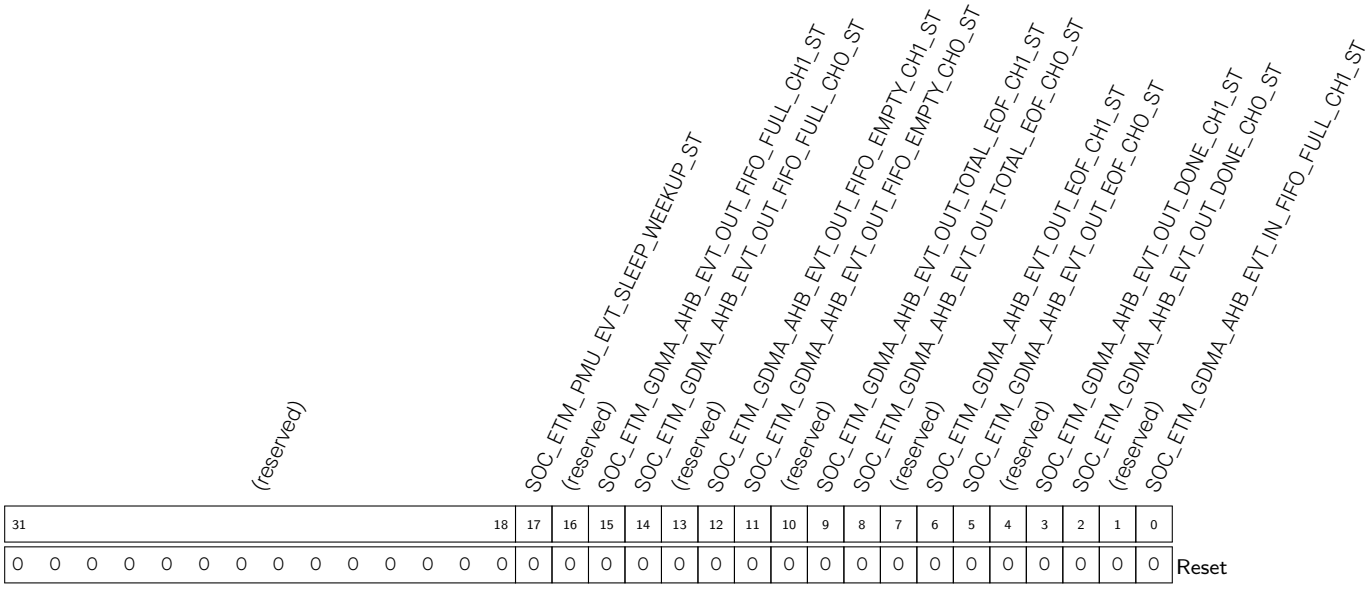
Register 11.5. SOC_ETM_EVT_ST2_REG (0x01B8)

Continued from the previous page...

**SOC_ETM_GDMA_AHB_EVT_IN_FIFO_EMPTY_CH1_ST** Represents GDMA_AHB_EVT_IN_FIFO_EMPTY_CH1 trigger status.
0: Not triggered
1: Triggered
(R/WTC/SS)

**SOC_ETM_GDMA_AHB_EVT_IN_FIFO_FULL_CH0_ST** Represents GDMA_AHB_EVT_IN_FIFO_FULL_CH0 trigger status.
0: Not triggered
1: Triggered
(R/WTC/SS)

Register 11.6. SOC_ETM_EVT_ST3_REG (0x01C0)




**SOC_ETM_GDMA_AHB_EVT_IN_FIFO_FULL_CH1_ST** Represents GDMA_AHB_EVT_IN_FIFO_FULL_CH1 trigger status.
0: Not triggered
1: Triggered
(R/WTC/SS)

**SOC_ETM_GDMA_AHB_EVT_OUT_DONE_CH0_ST** Represents GDMA_AHB_EVT_OUT_DONE_CH0 trigger status.
0: Not triggered
1: Triggered
(R/WTC/SS)

**SOC_ETM_GDMA_AHB_EVT_OUT_DONE_CH1_ST** Represents GDMA_AHB_EVT_OUT_DONE_CH1 trigger status.
0: Not triggered
1: Triggered
(R/WTC/SS)

**SOC_ETM_GDMA_AHB_EVT_OUT_EOF_CH0_ST** Represents GDMA_AHB_EVT_OUT_EOF_CH0 trigger status.
0: Not triggered
1: Triggered
(R/WTC/SS)

Continued on the next page...

Register 11.6. SOC_ETM_EVT_ST3_REG (0x01C0)

Continued from the previous page...

**SOC_ETM_GDMA_AHB_EVT_OUT_EOF_CH1_ST** Represents GDMA_AHB_EVT_OUT_EOF_CH1 trigger status.
0: Not triggered
1: Triggered
(R/WTC/SS)

**SOC_ETM_GDMA_AHB_EVT_OUT_TOTAL_EOF_CH0_ST** Represents GDMA_AHB_EVT_OUT_TOTAL_EOF_CH0 trigger status.
0: Not triggered
1: Triggered
(R/WTC/SS)

**SOC_ETM_GDMA_AHB_EVT_OUT_TOTAL_EOF_CH1_ST** Represents GDMA_AHB_EVT_OUT_TOTAL_EOF_CH1 trigger status.
0: Not triggered
1: Triggered
(R/WTC/SS)

**SOC_ETM_GDMA_AHB_EVT_OUT_FIFO_EMPTY_CH0_ST** Represents GDMA_AHB_EVT_OUT_FIFO_EMPTY_CH0 trigger status.
0: Not triggered
1: Triggered
(R/WTC/SS)

**SOC_ETM_GDMA_AHB_EVT_OUT_FIFO_EMPTY_CH1_ST** Represents GDMA_AHB_EVT_OUT_FIFO_EMPTY_CH1 trigger status.
0: Not triggered
1: Triggered
(R/WTC/SS)

**SOC_ETM_GDMA_AHB_EVT_OUT_FIFO_FULL_CH0_ST** Represents GDMA_AHB_EVT_OUT_FIFO_FULL_CH0 trigger status.
0: Not triggered
1: Triggered
(R/WTC/SS)

**SOC_ETM_GDMA_AHB_EVT_OUT_FIFO_FULL_CH1_ST** Represents GDMA_AHB_EVT_OUT_FIFO_FULL_CH1 trigger status.
0: Not triggered
1: Triggered
(R/WTC/SS)

Continued on the next page...

Register 11.6. SOC_ETM_EVT_ST3_REG (0x01C0)

Continued from the previous page...

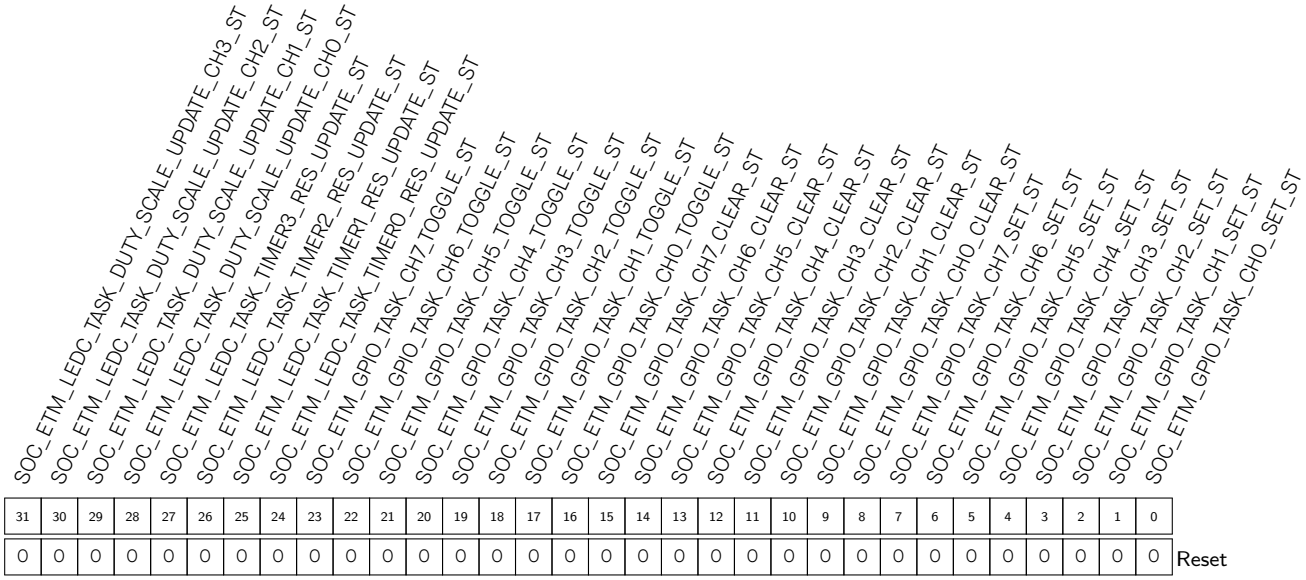
**SOC_ETM_PMU_EVT_SLEEP_WEEKUP_ST** Represents PMU_EVT_SLEEP_WEEKUP trigger status.

0: Not triggered

1: Triggered

(R/WTC/SS)

Register 11.7. SOC_ETM_TASK_ST0_REG (0x01C8)




**SOC_ETM_GPIO_TASK_CH0_SET_ST** Represents GPIO_TASK_CH0_SET trigger status.

0: Not triggered

1: Triggered

(R/WTC/SS)

**SOC_ETM_GPIO_TASK_CH1_SET_ST** Represents GPIO_TASK_CH1_SET trigger status.

0: Not triggered

1: Triggered

(R/WTC/SS)

**SOC_ETM_GPIO_TASK_CH2_SET_ST** Represents GPIO_TASK_CH2_SET trigger status.

0: Not triggered

1: Triggered

(R/WTC/SS)

**SOC_ETM_GPIO_TASK_CH3_SET_ST** Represents GPIO_TASK_CH3_SET trigger status.

0: Not triggered

1: Triggered

(R/WTC/SS)

**SOC_ETM_GPIO_TASK_CH4_SET_ST** Represents GPIO_TASK_CH4_SET trigger status.

0: Not triggered

1: Triggered

(R/WTC/SS)

**SOC_ETM_GPIO_TASK_CH5_SET_ST** Represents GPIO_TASK_CH5_SET trigger status.

0: Not triggered

1: Triggered

(R/WTC/SS)

Continued on the next page...

Register 11.7. SOC_ETM_TASK_ST0_REG (0x01C8)

Continued from the previous page...

**SOC_ETM_GPIO_TASK_CH6_SET_ST** Represents GPIO_TASK_CH6_SET trigger status.
0: Not triggered
1: Triggered
(R/WTC/SS)

**SOC_ETM_GPIO_TASK_CH7_SET_ST** Represents GPIO_TASK_CH7_SET trigger status.
0: Not triggered
1: Triggered
(R/WTC/SS)

**SOC_ETM_GPIO_TASK_CH0_CLEAR_ST** Represents GPIO_TASK_CH0_CLEAR trigger status.
0: Not triggered
1: Triggered
(R/WTC/SS)

**SOC_ETM_GPIO_TASK_CH1_CLEAR_ST** Represents GPIO_TASK_CH1_CLEAR trigger status.
0: Not triggered
1: Triggered
(R/WTC/SS)

**SOC_ETM_GPIO_TASK_CH2_CLEAR_ST** Represents GPIO_TASK_CH2_CLEAR trigger status.
0: Not triggered
1: Triggered
(R/WTC/SS)

**SOC_ETM_GPIO_TASK_CH3_CLEAR_ST** Represents GPIO_TASK_CH3_CLEAR trigger status.
0: Not triggered
1: Triggered
(R/WTC/SS)

**SOC_ETM_GPIO_TASK_CH4_CLEAR_ST** Represents GPIO_TASK_CH4_CLEAR trigger status.
0: Not triggered
1: Triggered
(R/WTC/SS)

**SOC_ETM_GPIO_TASK_CH5_CLEAR_ST** Represents GPIO_TASK_CH5_CLEAR trigger status.
0: Not triggered
1: Triggered
(R/WTC/SS)

Continued on the next page...

Register 11.7. SOC_ETM_TASK_ST0_REG (0x01C8)

Continued from the previous page...

**SOC_ETM_GPIO_TASK_CH6_CLEAR_ST** Represents GPIO_TASK_CH6_CLEAR trigger status.
0: Not triggered
1: Triggered
(R/WTC/SS)

**SOC_ETM_GPIO_TASK_CH7_CLEAR_ST** Represents GPIO_TASK_CH7_CLEAR trigger status.
0: Not triggered
1: Triggered
(R/WTC/SS)

**SOC_ETM_GPIO_TASK_CH0_TOGGLE_ST** Represents GPIO_TASK_CH0_TOGGLE trigger status.
0: Not triggered
1: Triggered
(R/WTC/SS)

**SOC_ETM_GPIO_TASK_CH1_TOGGLE_ST** Represents GPIO_TASK_CH1_TOGGLE trigger status.
0: Not triggered
1: Triggered
(R/WTC/SS)

**SOC_ETM_GPIO_TASK_CH2_TOGGLE_ST** Represents GPIO_TASK_CH2_TOGGLE trigger status.
0: Not triggered
1: Triggered
(R/WTC/SS)

**SOC_ETM_GPIO_TASK_CH3_TOGGLE_ST** Represents GPIO_TASK_CH3_TOGGLE trigger status.
0: Not triggered
1: Triggered
(R/WTC/SS)

**SOC_ETM_GPIO_TASK_CH4_TOGGLE_ST** Represents GPIO_TASK_CH4_TOGGLE trigger status.
0: Not triggered
1: Triggered
(R/WTC/SS)

Continued on the next page...

Register 11.7. SOC_ETM_TASK_ST0_REG (0x01C8)

Continued from the previous page...

**SOC_ETM_GPIO_TASK_CH5_TOGGLE_ST** Represents GPIO_TASK_CH5_TOGGLE trigger status.
0: Not triggered
1: Triggered
(R/WTC/SS)

**SOC_ETM_GPIO_TASK_CH6_TOGGLE_ST** Represents GPIO_TASK_CH6_TOGGLE trigger status.
0: Not triggered
1: Triggered
(R/WTC/SS)

**SOC_ETM_GPIO_TASK_CH7_TOGGLE_ST** Represents GPIO_TASK_CH7_TOGGLE trigger status.
0: Not triggered
1: Triggered
(R/WTC/SS)

**SOC_ETM_LEDC_TASK_TIMER0_RES_UPDATE_ST** Represents LEDC_TASK_TIMER0_RES_UPDATE trigger status.
0: Not triggered
1: Triggered
(R/WTC/SS)

**SOC_ETM_LEDC_TASK_TIMER1_RES_UPDATE_ST** Represents LEDC_TASK_TIMER1_RES_UPDATE trigger status.
0: Not triggered
1: Triggered
(R/WTC/SS)

**SOC_ETM_LEDC_TASK_TIMER2_RES_UPDATE_ST** Represents LEDC_TASK_TIMER2_RES_UPDATE trigger status.
0: Not triggered
1: Triggered
(R/WTC/SS)

**SOC_ETM_LEDC_TASK_TIMER3_RES_UPDATE_ST** Represents LEDC_TASK_TIMER3_RES_UPDATE trigger status.
0: Not triggered
1: Triggered
(R/WTC/SS)

Continued on the next page...

Register 11.7. SOC_ETM_TASK_ST0_REG (0x01C8)

Continued from the previous page...

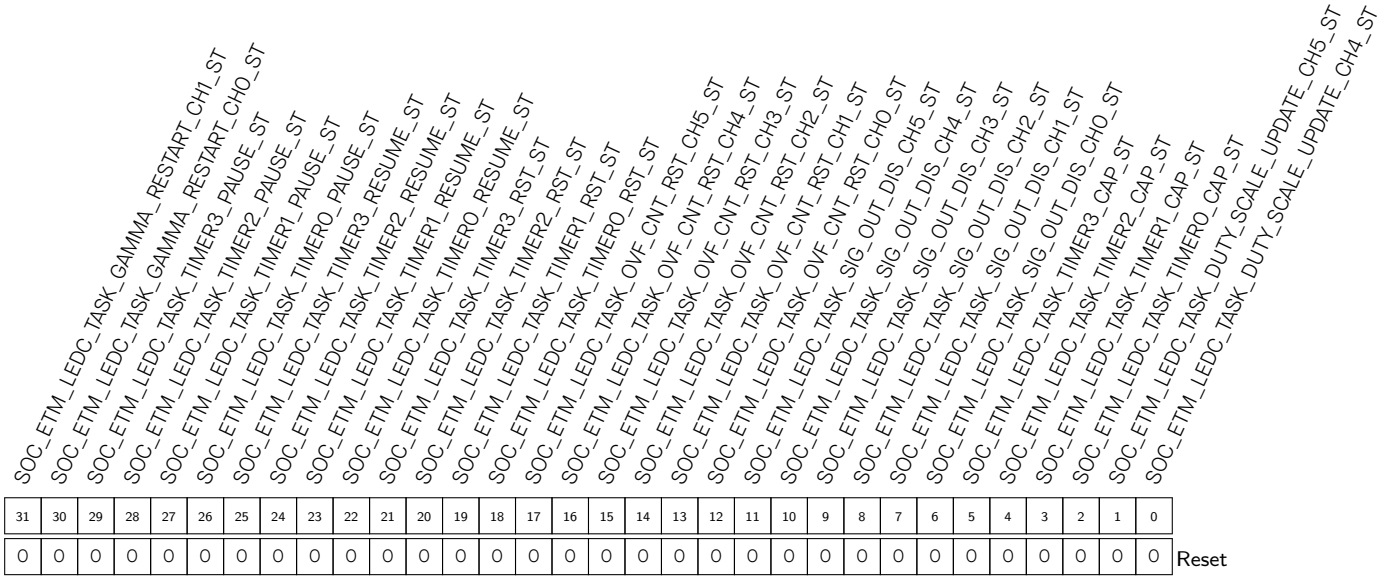
**SOC_ETM_LEDC_TASK_DUTY_SCALE_UPDATE_CH0_ST** Represents LEDC_TASK_DUTY_SCALE_UPDATE_CH0 trigger status.
0: Not triggered
1: Triggered
(R/WTC/SS)

**SOC_ETM_LEDC_TASK_DUTY_SCALE_UPDATE_CH1_ST** Represents LEDC_TASK_DUTY_SCALE_UPDATE_CH1 trigger status.
0: Not triggered
1: Triggered
(R/WTC/SS)

**SOC_ETM_LEDC_TASK_DUTY_SCALE_UPDATE_CH2_ST** Represents LEDC_TASK_DUTY_SCALE_UPDATE_CH2 trigger status.
0: Not triggered
1: Triggered
(R/WTC/SS)

**SOC_ETM_LEDC_TASK_DUTY_SCALE_UPDATE_CH3_ST** Represents LEDC_TASK_DUTY_SCALE_UPDATE_CH3 trigger status.
0: Not triggered
1: Triggered
(R/WTC/SS)

Register 11.8. SOC_ETM_TASK_ST1_REG (0x01D0)




**SOC_ETM_LEDC_TASK_DUTY_SCALE_UPDATE_CH4_ST** Represents LEDC_TASK_DUTY_SCALE_UPDATE_CH4 trigger status.

0: Not triggered

1: Triggered

(R/WTC/SS)

**SOC_ETM_LEDC_TASK_DUTY_SCALE_UPDATE_CH5_ST** Represents LEDC_TASK_DUTY_SCALE_UPDATE_CH5 trigger status.

0: Not triggered

1: Triggered

(R/WTC/SS)

**SOC_ETM_LEDC_TASK_TIMER0_CAP_ST** Represents LEDC_TASK_TIMER0_CAP trigger status.

0: Not triggered

1: Triggered

(R/WTC/SS)

**SOC_ETM_LEDC_TASK_TIMER1_CAP_ST** Represents LEDC_TASK_TIMER1_CAP trigger status.

0: Not triggered

1: Triggered

(R/WTC/SS)

**SOC_ETM_LEDC_TASK_TIMER2_CAP_ST** Represents LEDC_TASK_TIMER2_CAP trigger status.

0: Not triggered

1: Triggered

(R/WTC/SS)

**SOC_ETM_LEDC_TASK_TIMER3_CAP_ST** Represents LEDC_TASK_TIMER3_CAP trigger status.

0: Not triggered

1: Triggered

(R/WTC/SS)

Continued on the next page...

Register 11.8. SOC_ETM_TASK_ST1_REG (0x01D0)

Continued from the previous page...

**SOC_ETM_LEDC_TASK_SIG_OUT_DIS_CH0_ST** Represents LEDC_TASK_SIG_OUT_DIS_CH0 trigger status.
0: Not triggered
1: Triggered
(R/WTC/SS)

**SOC_ETM_LEDC_TASK_SIG_OUT_DIS_CH1_ST** Represents LEDC_TASK_SIG_OUT_DIS_CH1 trigger status.
0: Not triggered
1: Triggered
(R/WTC/SS)

**SOC_ETM_LEDC_TASK_SIG_OUT_DIS_CH2_ST** Represents LEDC_TASK_SIG_OUT_DIS_CH2 trigger status.
0: Not triggered
1: Triggered
(R/WTC/SS)

**SOC_ETM_LEDC_TASK_SIG_OUT_DIS_CH3_ST** Represents LEDC_TASK_SIG_OUT_DIS_CH3 trigger status.
0: Not triggered
1: Triggered
(R/WTC/SS)

**SOC_ETM_LEDC_TASK_SIG_OUT_DIS_CH4_ST** Represents LEDC_TASK_SIG_OUT_DIS_CH4 trigger status.
0: Not triggered
1: Triggered
(R/WTC/SS)

**SOC_ETM_LEDC_TASK_SIG_OUT_DIS_CH5_ST** Represents LEDC_TASK_SIG_OUT_DIS_CH5 trigger status.
0: Not triggered
1: Triggered
(R/WTC/SS)

**SOC_ETM_LEDC_TASK_OVF_CNT_RST_CH0_ST** Represents LEDC_TASK_OVF_CNT_RST_CH0 trigger status.
0: Not triggered
1: Triggered
(R/WTC/SS)

Continued on the next page...

Register 11.8. SOC_ETM_TASK_ST1_REG (0x01D0)

Continued from the previous page...

**SOC_ETM_LEDC_TASK_OVF_CNT_RST_CH1_ST** Represents LEDC_TASK_OVF_CNT_RST_CH1 trigger status.
0: Not triggered
1: Triggered
(R/WTC/SS)

**SOC_ETM_LEDC_TASK_OVF_CNT_RST_CH2_ST** Represents LEDC_TASK_OVF_CNT_RST_CH2 trigger status.
0: Not triggered
1: Triggered
(R/WTC/SS)

**SOC_ETM_LEDC_TASK_OVF_CNT_RST_CH3_ST** Represents LEDC_TASK_OVF_CNT_RST_CH3 trigger status.
0: Not triggered
1: Triggered
(R/WTC/SS)

**SOC_ETM_LEDC_TASK_OVF_CNT_RST_CH4_ST** Represents LEDC_TASK_OVF_CNT_RST_CH4 trigger status.
0: Not triggered
1: Triggered
(R/WTC/SS)

**SOC_ETM_LEDC_TASK_OVF_CNT_RST_CH5_ST** Represents LEDC_TASK_OVF_CNT_RST_CH5 trigger status.
0: Not triggered
1: Triggered
(R/WTC/SS)

**SOC_ETM_LEDC_TASK_TIMER0_RST_ST** Represents LEDC_TASK_TIMER0_RST trigger status.
0: Not triggered
1: Triggered
(R/WTC/SS)

**SOC_ETM_LEDC_TASK_TIMER1_RST_ST** Represents LEDC_TASK_TIMER1_RST trigger status.
0: Not triggered
1: Triggered
(R/WTC/SS)

Continued on the next page...

Register 11.8. SOC_ETM_TASK_ST1_REG (0x01D0)

Continued from the previous page...

**SOC_ETM_LEDC_TASK_TIMER2_RST_ST** Represents LEDC_TASK_TIMER2_RST trigger status.
0: Not triggered
1: Triggered
(R/WTC/SS)

**SOC_ETM_LEDC_TASK_TIMER3_RST_ST** Represents LEDC_TASK_TIMER3_RST trigger status.
0: Not triggered
1: Triggered
(R/WTC/SS)

**SOC_ETM_LEDC_TASK_TIMER0_RESUME_ST** Represents LEDC_TASK_TIMER0_RESUME trigger status.
0: Not triggered
1: Triggered
(R/WTC/SS)

**SOC_ETM_LEDC_TASK_TIMER1_RESUME_ST** Represents LEDC_TASK_TIMER1_RESUME trigger status.
0: Not triggered
1: Triggered
(R/WTC/SS)

**SOC_ETM_LEDC_TASK_TIMER2_RESUME_ST** Represents LEDC_TASK_TIMER2_RESUME trigger status.
0: Not triggered
1: Triggered
(R/WTC/SS)

**SOC_ETM_LEDC_TASK_TIMER3_RESUME_ST** Represents LEDC_TASK_TIMER3_RESUME trigger status.
0: Not triggered
1: Triggered
(R/WTC/SS)

**SOC_ETM_LEDC_TASK_TIMER0_PAUSE_ST** Represents LEDC_TASK_TIMER0_PAUSE trigger status.
0: Not triggered
1: Triggered
(R/WTC/SS)

Continued on the next page...

Register 11.8. SOC_ETM_TASK_ST1_REG (0x01D0)

Continued from the previous page...

**SOC_ETM_LEDC_TASK_TIMER1_PAUSE_ST** Represents LEDC_TASK_TIMER1_PAUSE trigger status.
0: Not triggered
1: Triggered
(R/WTC/SS)

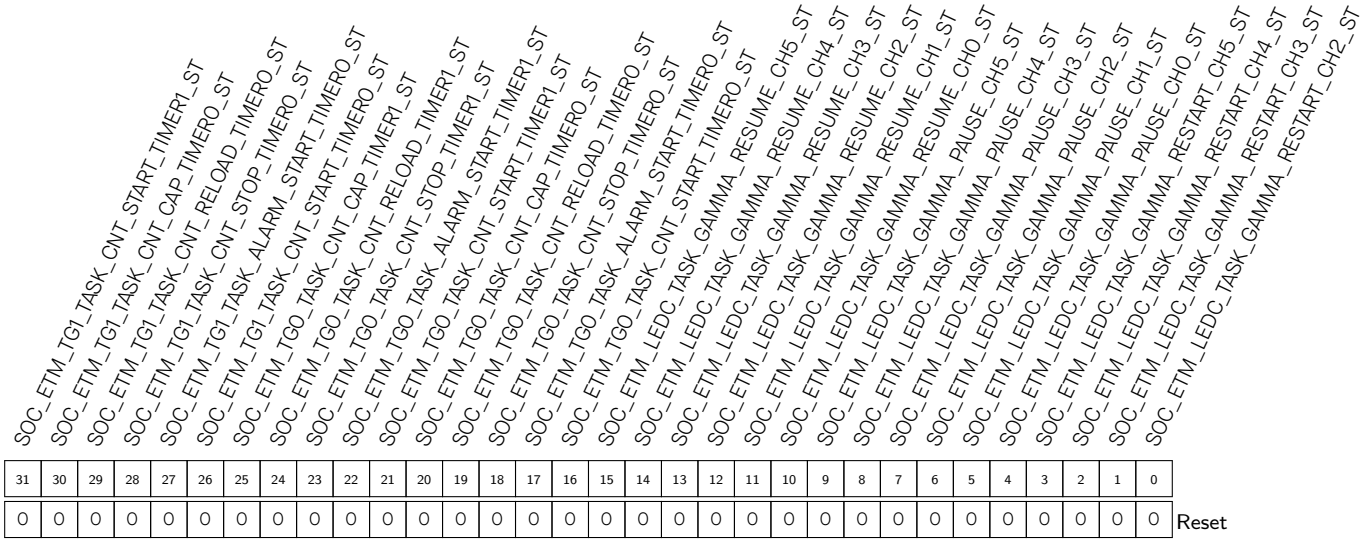
**SOC_ETM_LEDC_TASK_TIMER2_PAUSE_ST** Represents LEDC_TASK_TIMER2_PAUSE trigger status.
0: Not triggered
1: Triggered
(R/WTC/SS)

**SOC_ETM_LEDC_TASK_TIMER3_PAUSE_ST** Represents LEDC_TASK_TIMER3_PAUSE trigger status.
0: Not triggered
1: Triggered
(R/WTC/SS)

**SOC_ETM_LEDC_TASK_GAMMA_RESTART_CH0_ST** Represents LEDC_TASK_GAMMA_RESTART_CH0 trigger status.
0: Not triggered
1: Triggered
(R/WTC/SS)

**SOC_ETM_LEDC_TASK_GAMMA_RESTART_CH1_ST** Represents LEDC_TASK_GAMMA_RESTART_CH1 trigger status.
0: Not triggered
1: Triggered
(R/WTC/SS)

Register 11.9. SOC_ETM_TASK_ST2_REG (0x01D8)




**SOC_ETM_LEDC_TASK_GAMMA_RESTART_CH2_ST** Represents LEDC_TASK_GAMMA_RESTART_CH2 trigger status.

0: Not triggered

1: Triggered

(R/WTC/SS)

**SOC_ETM_LEDC_TASK_GAMMA_RESTART_CH3_ST** Represents LEDC_TASK_GAMMA_RESTART_CH3 trigger status.

0: Not triggered

1: Triggered

(R/WTC/SS)

**SOC_ETM_LEDC_TASK_GAMMA_RESTART_CH4_ST** Represents LEDC_TASK_GAMMA_RESTART_CH4 trigger status.

0: Not triggered

1: Triggered

(R/WTC/SS)

**SOC_ETM_LEDC_TASK_GAMMA_RESTART_CH5_ST** Represents LEDC_TASK_GAMMA_RESTART_CH5 trigger status.

0: Not triggered

1: Triggered

(R/WTC/SS)

**SOC_ETM_LEDC_TASK_GAMMA_PAUSE_CH0_ST** Represents LEDC_TASK_GAMMA_PAUSE_CH0 trigger status.

0: Not triggered

1: Triggered

(R/WTC/SS)

**Continued on the next page...**

Register 11.9. SOC_ETM_TASK_ST2_REG (0x01D8)

Continued from the previous page...

**SOC_ETM_LEDC_TASK_GAMMA_PAUSE_CH1_ST** Represents LEDC_TASK_GAMMA_PAUSE_CH1 trigger status.
0: Not triggered
1: Triggered
(R/WTC/SS)

**SOC_ETM_LEDC_TASK_GAMMA_PAUSE_CH2_ST** Represents LEDC_TASK_GAMMA_PAUSE_CH2 trigger status.
0: Not triggered
1: Triggered
(R/WTC/SS)

**SOC_ETM_LEDC_TASK_GAMMA_PAUSE_CH3_ST** Represents LEDC_TASK_GAMMA_PAUSE_CH3 trigger status.
0: Not triggered
1: Triggered
(R/WTC/SS)

**SOC_ETM_LEDC_TASK_GAMMA_PAUSE_CH4_ST** Represents LEDC_TASK_GAMMA_PAUSE_CH4 trigger status.
0: Not triggered
1: Triggered
(R/WTC/SS)

**SOC_ETM_LEDC_TASK_GAMMA_PAUSE_CH5_ST** Represents LEDC_TASK_GAMMA_PAUSE_CH5 trigger status.
0: Not triggered
1: Triggered
(R/WTC/SS)

**SOC_ETM_LEDC_TASK_GAMMA_RESUME_CH0_ST** Represents LEDC_TASK_GAMMA_RESUME_CH0 trigger status.
0: Not triggered
1: Triggered
(R/WTC/SS)

**SOC_ETM_LEDC_TASK_GAMMA_RESUME_CH1_ST** Represents LEDC_TASK_GAMMA_RESUME_CH1 trigger status.
0: Not triggered
1: Triggered
(R/WTC/SS)

Continued on the next page...

Register 11.9. SOC_ETM_TASK_ST2_REG (0x01D8)

Continued from the previous page...

**SOC_ETM_LEDC_TASK_GAMMA_RESUME_CH2_ST** Represents LEDC_TASK_GAMMA_RESUME_CH2 trigger status.
0: Not triggered
1: Triggered
(R/WTC/SS)

**SOC_ETM_LEDC_TASK_GAMMA_RESUME_CH3_ST** Represents LEDC_TASK_GAMMA_RESUME_CH3 trigger status.
0: Not triggered
1: Triggered
(R/WTC/SS)

**SOC_ETM_LEDC_TASK_GAMMA_RESUME_CH4_ST** Represents LEDC_TASK_GAMMA_RESUME_CH4 trigger status.
0: Not triggered
1: Triggered
(R/WTC/SS)

**SOC_ETM_LEDC_TASK_GAMMA_RESUME_CH5_ST** Represents LEDC_TASK_GAMMA_RESUME_CH5 trigger status.
0: Not triggered
1: Triggered
(R/WTC/SS)

**SOC_ETM_TGO_TASK_CNT_START_TIMER0_ST** Represents TGO_TASK_CNT_START_TIMER0 trigger status.
0: Not triggered
1: Triggered
(R/WTC/SS)

**SOC_ETM_TGO_TASK_ALARM_START_TIMER0_ST** Represents TGO_TASK_ALARM_START_TIMER0 trigger status.
0: Not triggered
1: Triggered
(R/WTC/SS)

**SOC_ETM_TGO_TASK_CNT_STOP_TIMER0_ST** Represents TGO_TASK_CNT_STOP_TIMER0 trigger status.
0: Not triggered
1: Triggered
(R/WTC/SS)

Continued on the next page...

Register 11.9. SOC_ETM_TASK_ST2_REG (0x01D8)

Continued from the previous page...

**SOC_ETM_TG0_TASK_CNT_RELOAD_TIMER0_ST** Represents TG0_TASK_CNT_RELOAD_TIMER0 trigger status.
0: Not triggered
1: Triggered
(R/WTC/SS)

**SOC_ETM_TG0_TASK_CNT_CAP_TIMER0_ST** Represents TG0_TASK_CNT_CAP_TIMER0 trigger status.
0: Not triggered
1: Triggered
(R/WTC/SS)

**SOC_ETM_TG0_TASK_CNT_START_TIMER1_ST** Represents TG0_TASK_CNT_START_TIMER1 trigger status.
0: Not triggered
1: Triggered
(R/WTC/SS)

**SOC_ETM_TG0_TASK_ALARM_START_TIMER1_ST** Represents TG0_TASK_ALARM_START_TIMER1 trigger status.
0: Not triggered
1: Triggered
(R/WTC/SS)

**SOC_ETM_TG0_TASK_CNT_STOP_TIMER1_ST** Represents TG0_TASK_CNT_STOP_TIMER1 trigger status.
0: Not triggered
1: Triggered
(R/WTC/SS)

**SOC_ETM_TG0_TASK_CNT_RELOAD_TIMER1_ST** Represents TG0_TASK_CNT_RELOAD_TIMER1 trigger status.
0: Not triggered
1: Triggered
(R/WTC/SS)

**SOC_ETM_TG0_TASK_CNT_CAP_TIMER1_ST** Represents TG0_TASK_CNT_CAP_TIMER1 trigger status.
0: Not triggered
1: Triggered
(R/WTC/SS)

Continued on the next page...

Register 11.9. SOC_ETM_TASK_ST2_REG (0x01D8)

Continued from the previous page...

**SOC_ETM_TG1_TASK_CNT_START_TIMER0_ST** Represents TG1_TASK_CNT_START_TIMER0 trigger status.
0: Not triggered
1: Triggered
(R/WTC/SS)

**SOC_ETM_TG1_TASK_ALARM_START_TIMER0_ST** Represents TG1_TASK_ALARM_START_TIMER0 trigger status.
0: Not triggered
1: Triggered
(R/WTC/SS)

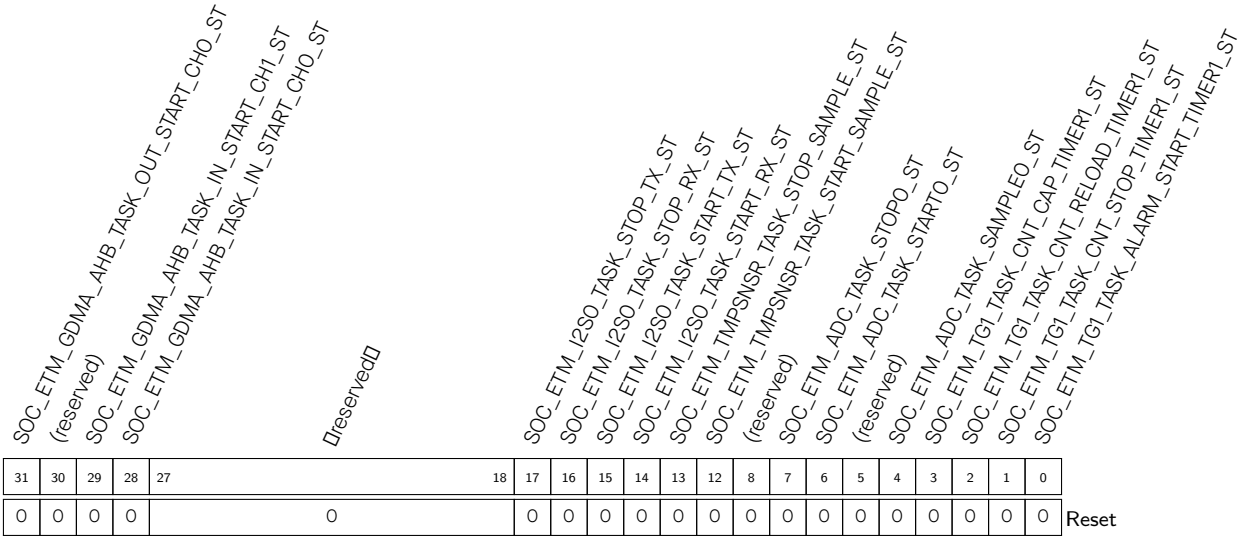
**SOC_ETM_TG1_TASK_CNT_STOP_TIMER0_ST** Represents TG1_TASK_CNT_STOP_TIMER0 trigger status.
0: Not triggered
1: Triggered
(R/WTC/SS)

**SOC_ETM_TG1_TASK_CNT_RELOAD_TIMER0_ST** Represents TG1_TASK_CNT_RELOAD_TIMER0 trigger status.
0: Not triggered
1: Triggered
(R/WTC/SS)

**SOC_ETM_TG1_TASK_CNT_CAP_TIMER0_ST** Represents TG1_TASK_CNT_CAP_TIMER0 trigger status.
0: Not triggered
1: Triggered
(R/WTC/SS)

**SOC_ETM_TG1_TASK_CNT_START_TIMER1_ST** Represents TG1_TASK_CNT_START_TIMER1 trigger status.
0: Not triggered
1: Triggered
(R/WTC/SS)

Register 11.10. SOC_ETM_TASK_ST3_REG (0x01E0)




**SOC_ETM_TG1_TASK_ALARM_START_TIMER1_ST** Represents TG1_TASK_ALARM_START_TIMER1 trigger status.

0: Not triggered

1: Triggered

(R/WTC/SS)

**SOC_ETM_TG1_TASK_CNT_STOP_TIMER1_ST** Represents TG1_TASK_CNT_STOP_TIMER1 trigger status.

0: Not triggered

1: Triggered

(R/WTC/SS)

**SOC_ETM_TG1_TASK_CNT_RELOAD_TIMER1_ST** Represents TG1_TASK_CNT_RELOAD_TIMER1 trigger status.

0: Not triggered

1: Triggered

(R/WTC/SS)

**SOC_ETM_TG1_TASK_CNT_CAP_TIMER1_ST** Represents TG1_TASK_CNT_CAP_TIMER1 trigger status.

0: Not triggered

1: Triggered

(R/WTC/SS)

**SOC_ETM_ADC_TASK_SAMPLE0_ST** Represents ADC_TASK_SAMPLE0 trigger status.

0: Not triggered

1: Triggered

(R/WTC/SS)

**Continued on the next page...**

Register 11.10. SOC_ETM_TASK_ST3_REG (0x01E0)

Continued from the previous page...

**SOC_ETM_ADC_TASK_START0_ST** Represents ADC_TASK_START0 trigger status.
0: Not triggered
1: Triggered
(R/WTC/SS)

**SOC_ETM_ADC_TASK_STOP0_ST** Represents ADC_TASK_STOP0 trigger status.
0: Not triggered
1: Triggered
(R/WTC/SS)

**SOC_ETM_TMPSNSR_TASK_START_SAMPLE_ST** Represents TMPSNSR_TASK_START_SAMPLE trigger status.
0: Not triggered
1: Triggered
(R/WTC/SS)

**SOC_ETM_TMPSNSR_TASK_STOP_SAMPLE_ST** Represents TMPSNSR_TASK_STOP_SAMPLE trigger status.
0: Not triggered
1: Triggered
(R/WTC/SS)

**SOC_ETM_I2S0_TASK_START_RX_ST** Represents I2S0_TASK_START_RX trigger status.
0: Not triggered
1: Triggered
(R/WTC/SS)

**SOC_ETM_I2S0_TASK_START_TX_ST** Represents I2S0_TASK_START_TX trigger status.
0: Not triggered
1: Triggered
(R/WTC/SS)

**SOC_ETM_I2S0_TASK_STOP_RX_ST** Represents I2S0_TASK_STOP_RX trigger status.
0: Not triggered
1: Triggered
(R/WTC/SS)

Continued on the next page...

Register 11.10. SOC_ETM_TASK_ST3_REG (0x01E0)

Continued from the previous page...

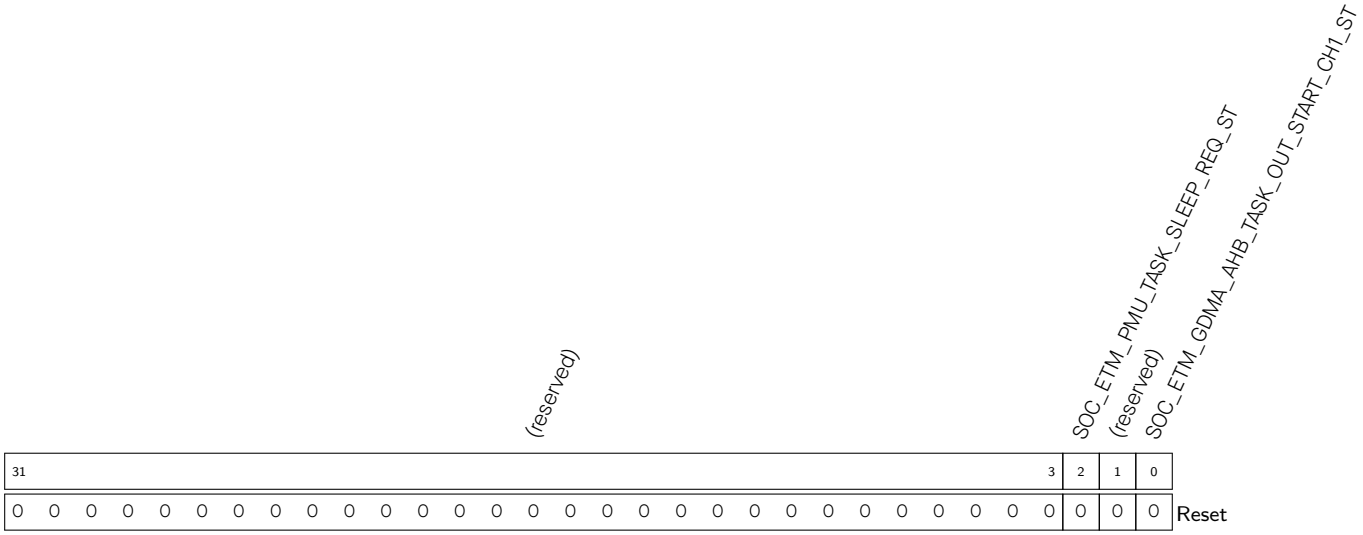
**SOC_ETM_I2S0_TASK_STOP_TX_ST** Represents I2S0_TASK_STOP_TX trigger status.
0: Not triggered
1: Triggered
(R/WTC/SS)

**SOC_ETM_GDMA_AHB_TASK_IN_START_CH0_ST** Represents GDMA_AHB_TASK_IN_START_CH0 trigger status.
0: Not triggered
1: Triggered
(R/WTC/SS)

**SOC_ETM_GDMA_AHB_TASK_IN_START_CH1_ST** Represents GDMA_AHB_TASK_IN_START_CH1 trigger status.
0: Not triggered
1: Triggered
(R/WTC/SS)

**SOC_ETM_GDMA_AHB_TASK_OUT_START_CH0_ST** Represents GDMA_AHB_TASK_OUT_START_CH0 trigger status.
0: Not triggered
1: Triggered
(R/WTC/SS)

Register 11.11. SOC_ETM_TASK_ST4_REG (0x01E8)




**SOC_ETM_GDMA_AHB_TASK_OUT_START_CH1_ST** Represents GDMA_AHB_TASK_OUT_START_CH1 trigger status.

0: Not triggered

1: Triggered

(R/WTC/SS)

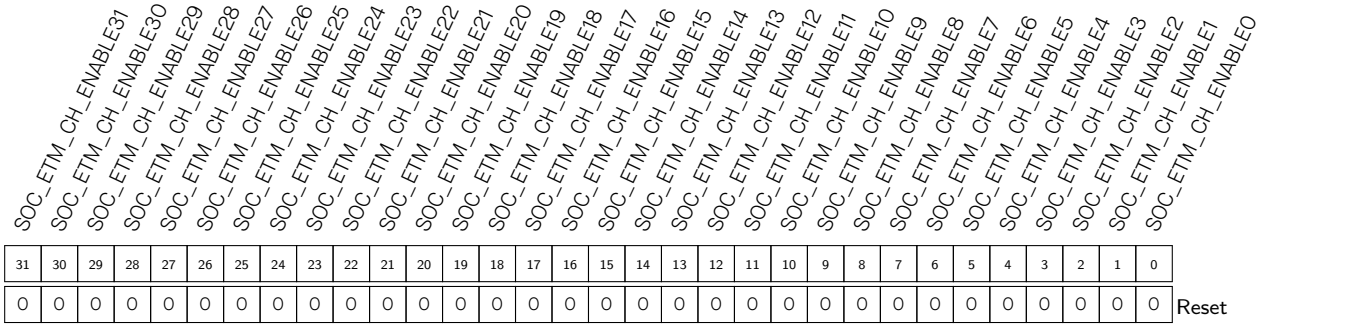
**SOC_ETM_PMU_TASK_SLEEP_REQ_ST** Represents PMU_TASK_SLEEP_REQ trigger status.

0: Not triggered

1: Triggered

(R/WTC/SS)

Register 11.12. SOC_ETM_CH_ENA_AD0_SET_REG (0x0004)



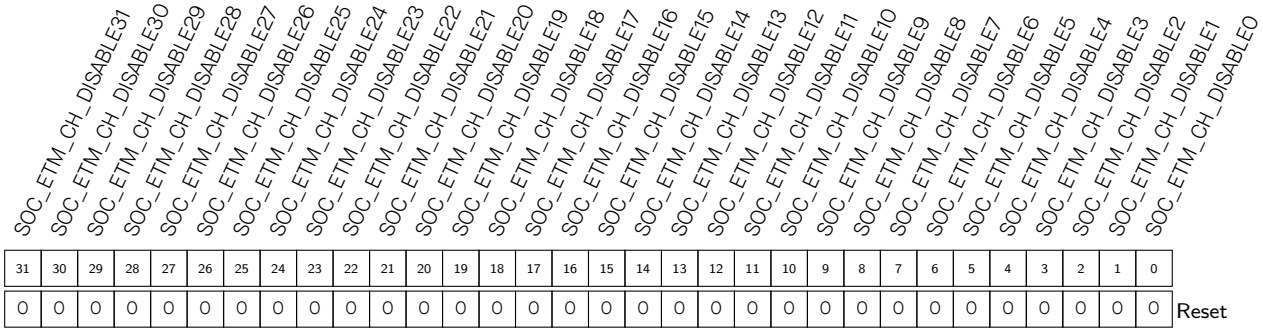

**SOC_ETM_CH_ENABLE*n* (*n*: 0-31)** Configures whether or not to enable channel*n*.

0: Invalid. No effect

1: Enable

(WT)

Register 11.13. SOC_ETM_CH_ENA_AD0_CLR_REG (0x0008)



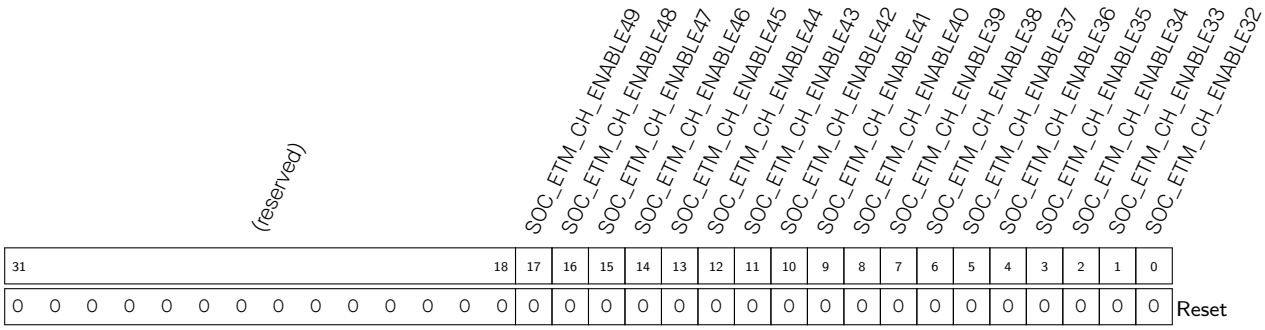

**SOC_ETM_CH_DISABLE*n* (*n*: 0-31)** Configures whether or not to disable channel*n*.

0: Invalid. No effect

1: Clear

(WT)

Register 11.14. SOC_ETM_CH_ENA_AD1_SET_REG (0x0010)



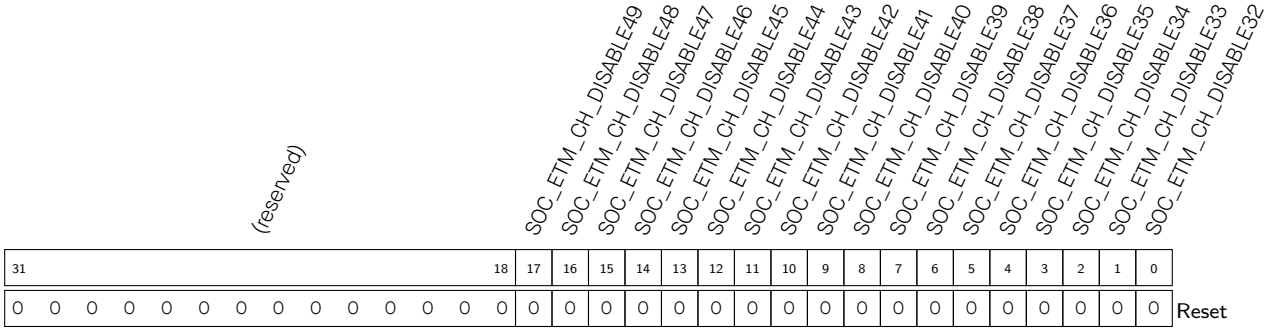

**SOC_ETM_CH_ENABLE*n* (*n*: 32-49)** Configures whether or not to enable channel*n*.

0: Invalid. No effect

1: Enable

(WT)

Register 11.15. SOC_ETM_CH_ENA_AD1_CLR_REG (0x0014)



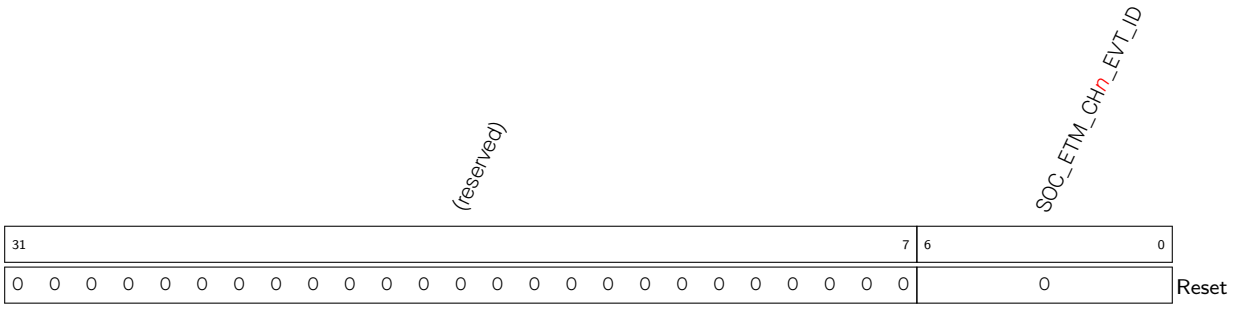

**SOC_ETM_CH_DISABLE*n* (*n*: 32-49)** Configures whether or not to disable channel*n*.

0: Invalid. No effect

1: Clear

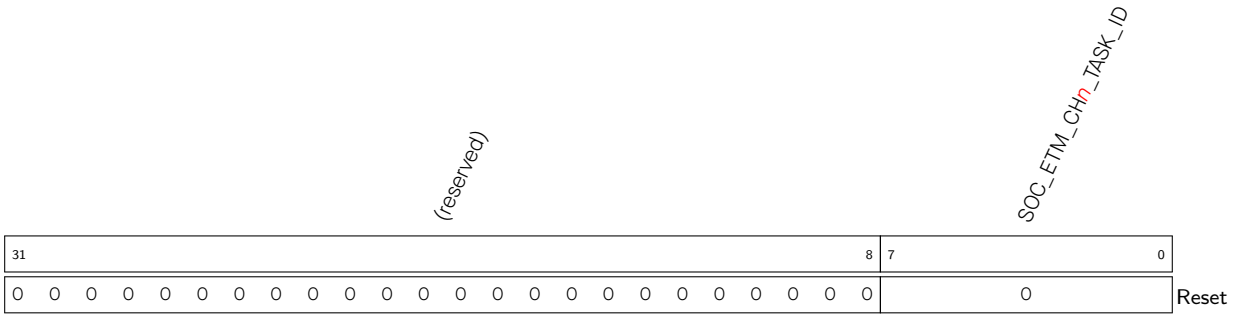
(WT)

Register 11.16. SOC_ETM_CH*n*_EVT_ID_REG (*n*: 0-49) (0x0018+0x8**n*)



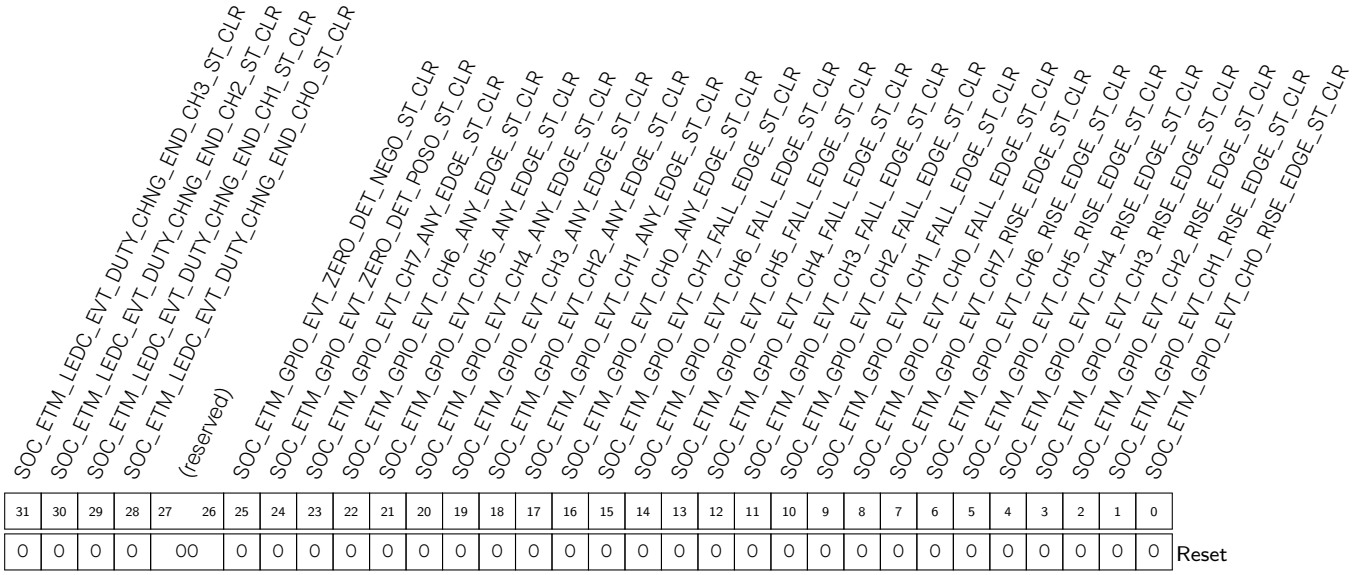

**SOC_ETM_CH*n*_EVT_ID** Configures channel*n* event ID. (R/W)

Register 11.17. SOC_ETM_CH*n*_TASK_ID_REG (*n*: 0-49) (0x001C+0x8**n*)




**SOC_ETM_CH*n*_TASK_ID** Configures channel*n* task ID. (R/W)

Register 11.18. SOC_ETM_EVT_ST0_CLR_REG (0x01AC)




**SOC_ETM_GPIO_EVT_CH0_RISE_EDGE_ST_CLR** Configures whether or not to clear GPIO_EVT_CH0_RISE_EDGE trigger status.

0: Invalid. No effect

1: Clear

(WT)

**SOC_ETM_GPIO_EVT_CH1_RISE_EDGE_ST_CLR** Configures whether or not to clear GPIO_EVT_CH1_RISE_EDGE trigger status.

0: Invalid. No effect

1: Clear

(WT)

**SOC_ETM_GPIO_EVT_CH2_RISE_EDGE_ST_CLR** Configures whether or not to clear GPIO_EVT_CH2_RISE_EDGE trigger status.

0: Invalid. No effect

1: Clear

(WT)

**SOC_ETM_GPIO_EVT_CH3_RISE_EDGE_ST_CLR** Configures whether or not to clear GPIO_EVT_CH3_RISE_EDGE trigger status.

0: Invalid. No effect

1: Clear

(WT)

**SOC_ETM_GPIO_EVT_CH4_RISE_EDGE_ST_CLR** Configures whether or not to clear GPIO_EVT_CH4_RISE_EDGE trigger status.

0: Invalid. No effect

1: Clear

(WT)

**Continued on the next page...**

Register 11.18. SOC_ETM_EVT_ST0_CLR_REG (0x01AC)

Continued from the previous page...

**SOC_ETM_GPIO_EVT_CH5_RISE_EDGE_ST_CLR** Configures whether or not to clear GPIO_EVT_CH5_RISE_EDGE trigger status.
0: Invalid. No effect
1: Clear
(WT)

**SOC_ETM_GPIO_EVT_CH6_RISE_EDGE_ST_CLR** Configures whether or not to clear GPIO_EVT_CH6_RISE_EDGE trigger status.
0: Invalid. No effect
1: Clear
(WT)

**SOC_ETM_GPIO_EVT_CH7_RISE_EDGE_ST_CLR** Configures whether or not to clear GPIO_EVT_CH7_RISE_EDGE trigger status.
0: Invalid. No effect
1: Clear
(WT)

**SOC_ETM_GPIO_EVT_CH0_FALL_EDGE_ST_CLR** Configures whether or not to clear GPIO_EVT_CH0_FALL_EDGE trigger status.
0: Invalid. No effect
1: Clear
(WT)

**SOC_ETM_GPIO_EVT_CH1_FALL_EDGE_ST_CLR** Configures whether or not to clear GPIO_EVT_CH1_FALL_EDGE trigger status.
0: Invalid. No effect
1: Clear
(WT)

**SOC_ETM_GPIO_EVT_CH2_FALL_EDGE_ST_CLR** Configures whether or not to clear GPIO_EVT_CH2_FALL_EDGE trigger status.
0: Invalid. No effect
1: Clear
(WT)

**SOC_ETM_GPIO_EVT_CH3_FALL_EDGE_ST_CLR** Configures whether or not to clear GPIO_EVT_CH3_FALL_EDGE trigger status.
0: Invalid. No effect
1: Clear
(WT)

Continued on the next page...

Register 11.18. SOC_ETM_EVT_ST0_CLR_REG (0x01AC)

Continued from the previous page...

**SOC_ETM_GPIO_EVT_CH4_FALL_EDGE_ST_CLR** Configures whether or not to clear GPIO_EVT_CH4_FALL_EDGE trigger status.
0: Invalid. No effect
1: Clear
(WT)

**SOC_ETM_GPIO_EVT_CH5_FALL_EDGE_ST_CLR** Configures whether or not to clear GPIO_EVT_CH5_FALL_EDGE trigger status.
0: Invalid. No effect
1: Clear
(WT)

**SOC_ETM_GPIO_EVT_CH6_FALL_EDGE_ST_CLR** Configures whether or not to clear GPIO_EVT_CH6_FALL_EDGE trigger status.
0: Invalid. No effect
1: Clear
(WT)

**SOC_ETM_GPIO_EVT_CH7_FALL_EDGE_ST_CLR** Configures whether or not to clear GPIO_EVT_CH7_FALL_EDGE trigger status.
0: Invalid. No effect
1: Clear
(WT)

**SOC_ETM_GPIO_EVT_CH0_ANY_EDGE_ST_CLR** Configures whether or not to clear GPIO_EVT_CH0_ANY_EDGE trigger status.
0: Invalid. No effect
1: Clear
(WT)

**SOC_ETM_GPIO_EVT_CH1_ANY_EDGE_ST_CLR** Configures whether or not to clear GPIO_EVT_CH1_ANY_EDGE trigger status.
0: Invalid. No effect
1: Clear
(WT)

**SOC_ETM_GPIO_EVT_CH2_ANY_EDGE_ST_CLR** Configures whether or not to clear GPIO_EVT_CH2_ANY_EDGE trigger status.
0: Invalid. No effect
1: Clear
(WT)

Continued on the next page...

Register 11.18. SOC_ETM_EVT_ST0_CLR_REG (0x01AC)

Continued from the previous page...

**SOC_ETM_GPIO_EVT_CH3_ANY_EDGE_ST_CLR** Configures whether or not to clear GPIO_EVT_CH3_ANY_EDGE trigger status.
0: Invalid. No effect
1: Clear
(WT)

**SOC_ETM_GPIO_EVT_CH4_ANY_EDGE_ST_CLR** Configures whether or not to clear GPIO_EVT_CH4_ANY_EDGE trigger status.
0: Invalid. No effect
1: Clear
(WT)

**SOC_ETM_GPIO_EVT_CH5_ANY_EDGE_ST_CLR** Configures whether or not to clear GPIO_EVT_CH5_ANY_EDGE trigger status.
0: Invalid. No effect
1: Clear
(WT)

**SOC_ETM_GPIO_EVT_CH6_ANY_EDGE_ST_CLR** Configures whether or not to clear GPIO_EVT_CH6_ANY_EDGE trigger status.
0: Invalid. No effect
1: Clear
(WT)

**SOC_ETM_GPIO_EVT_CH7_ANY_EDGE_ST_CLR** Configures whether or not to clear GPIO_EVT_CH7_ANY_EDGE trigger status.
0: Invalid. No effect
1: Clear
(WT)

**SOC_ETM_GPIO_EVT_ZERO_DET_POS0_ST_CLR** Configures whether or not to clear GPIO_EVT_ZERO_DET_POS0 trigger status.
0: Invalid. No effect
1: Clear
(WT)

**SOC_ETM_GPIO_EVT_ZERO_DET_NEG0_ST_CLR** Configures whether or not to clear GPIO_EVT_ZERO_DET_NEG0 trigger status.
0: Invalid. No effect
1: Clear
(WT)

Continued on the next page...

Register 11.18. SOC_ETM_EVT_ST0_CLR_REG (0x01AC)

Continued from the previous page...

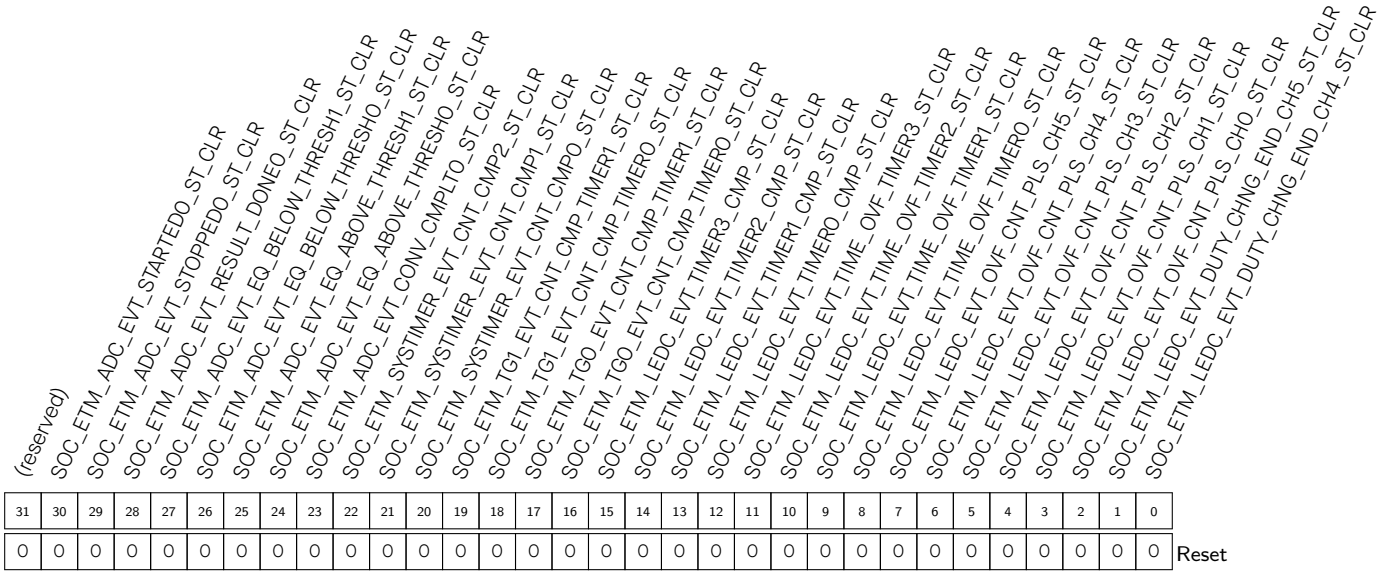
**SOC_ETM_LEDC_EVT_DUTY_CHNG_END_CH0_ST_CLR** Configures whether or not to clear LEDC_EVT_DUTY_CHNG_END_CH0 trigger status.
0: Invalid. No effect
1: Clear
(WT)

**SOC_ETM_LEDC_EVT_DUTY_CHNG_END_CH1_ST_CLR** Configures whether or not to clear LEDC_EVT_DUTY_CHNG_END_CH1 trigger status.
0: Invalid. No effect
1: Clear
(WT)

**SOC_ETM_LEDC_EVT_DUTY_CHNG_END_CH2_ST_CLR** Configures whether or not to clear LEDC_EVT_DUTY_CHNG_END_CH2 trigger status.
0: Invalid. No effect
1: Clear
(WT)

**SOC_ETM_LEDC_EVT_DUTY_CHNG_END_CH3_ST_CLR** Configures whether or not to clear LEDC_EVT_DUTY_CHNG_END_CH3 trigger status.
0: Invalid. No effect
1: Clear
(WT)

Register 11.19. SOC_ETM_EVT_ST1_CLR_REG (0x01B4)




**SOC_ETM_LEDC_EVT_DUTY_CHNG_END_CH4_ST_CLR** Configures whether or not to clear LEDC_EVT_DUTY_CHNG_END_CH4 trigger status.

0: Invalid. No effect

1: Clear

(WT)

**SOC_ETM_LEDC_EVT_DUTY_CHNG_END_CH5_ST_CLR** Configures whether or not to clear LEDC_EVT_DUTY_CHNG_END_CH5 trigger status.

0: Invalid. No effect

1: Clear

(WT)

**SOC_ETM_LEDC_EVT_OVF_CNT_PLS_CHO_ST_CLR** Configures whether or not to clear LEDC_EVT_OVF_CNT_PLS_CH0 trigger status.

0: Invalid. No effect

1: Clear

(WT)

**SOC_ETM_LEDC_EVT_OVF_CNT_PLS_CH1_ST_CLR** Configures whether or not to clear LEDC_EVT_OVF_CNT_PLS_CH1 trigger status.

0: Invalid. No effect

1: Clear

(WT)

**SOC_ETM_LEDC_EVT_OVF_CNT_PLS_CH2_ST_CLR** Configures whether or not to clear LEDC_EVT_OVF_CNT_PLS_CH2 trigger status.

0: Invalid. No effect

1: Clear

(WT)

Continued on the next page...

Register 11.19. SOC_ETM_EVT_ST1_CLR_REG (0x01B4)

Continued from the previous page...

**SOC_ETM_LEDC_EVT_OVF_CNT_PLS_CH3_ST_CLR** Configures whether or not to clear LEDC_EVT_OVF_CNT_PLS_CH3 trigger status.
0: Invalid. No effect
1: Clear
(WT)

**SOC_ETM_LEDC_EVT_OVF_CNT_PLS_CH4_ST_CLR** Configures whether or not to clear LEDC_EVT_OVF_CNT_PLS_CH4 trigger status.
0: Invalid. No effect
1: Clear
(WT)

**SOC_ETM_LEDC_EVT_OVF_CNT_PLS_CH5_ST_CLR** Configures whether or not to clear LEDC_EVT_OVF_CNT_PLS_CH5 trigger status.
0: Invalid. No effect
1: Clear
(WT)

**SOC_ETM_LEDC_EVT_TIME_OVF_TIMER0_ST_CLR** Configures whether or not to clear LEDC_EVT_TIME_OVF_TIMER0 trigger status.
0: Invalid. No effect
1: Clear
(WT)

**SOC_ETM_LEDC_EVT_TIME_OVF_TIMER1_ST_CLR** Configures whether or not to clear LEDC_EVT_TIME_OVF_TIMER1 trigger status.
0: Invalid. No effect
1: Clear
(WT)

**SOC_ETM_LEDC_EVT_TIME_OVF_TIMER2_ST_CLR** Configures whether or not to clear LEDC_EVT_TIME_OVF_TIMER2 trigger status.
0: Invalid. No effect
1: Clear
(WT)

**SOC_ETM_LEDC_EVT_TIME_OVF_TIMER3_ST_CLR** Configures whether or not to clear LEDC_EVT_TIME_OVF_TIMER3 trigger status.
0: Invalid. No effect
1: Clear
(WT)

Continued on the next page...

Register 11.19. SOC_ETM_EVT_ST1_CLR_REG (0x01B4)

Continued from the previous page...

**SOC_ETM_LEDC_EVT_TIMER0_CMP_ST_CLR** Configures whether or not to clear LEDC_EVT_TIMER0_CMP trigger status.
0: Invalid. No effect
1: Clear
(WT)

**SOC_ETM_LEDC_EVT_TIMER1_CMP_ST_CLR** Configures whether or not to clear LEDC_EVT_TIMER1_CMP trigger status.
0: Invalid. No effect
1: Clear
(WT)

**SOC_ETM_LEDC_EVT_TIMER2_CMP_ST_CLR** Configures whether or not to clear LEDC_EVT_TIMER2_CMP trigger status.
0: Invalid. No effect
1: Clear
(WT)

**SOC_ETM_LEDC_EVT_TIMER3_CMP_ST_CLR** Configures whether or not to clear LEDC_EVT_TIMER3_CMP trigger status.
0: Invalid. No effect
1: Clear
(WT)

**SOC_ETM_TG0_EVT_CNT_CMP_TIMER0_ST_CLR** Configures whether or not to clear TG0_EVT_CNT_CMP_TIMER0 trigger status.
0: Invalid. No effect
1: Clear
(WT)

**SOC_ETM_TG0_EVT_CNT_CMP_TIMER1_ST_CLR** Configures whether or not to clear TG0_EVT_CNT_CMP_TIMER1 trigger status.
0: Invalid. No effect
1: Clear
(WT)

**SOC_ETM_TG1_EVT_CNT_CMP_TIMER0_ST_CLR** Configures whether or not to clear TG1_EVT_CNT_CMP_TIMER0 trigger status.
0: Invalid. No effect
1: Clear
(WT)

Continued on the next page...

Register 11.19. SOC_ETM_EVT_ST1_CLR_REG (0x01B4)

Continued from the previous page...

**SOC_ETM_TG1_EVT_CNT_CMP_TIMER1_ST_CLR** Configures whether or not to clear TG1_EVT_CNT_CMP_TIMER1 trigger status.
0: Invalid. No effect
1: Clear
(WT)

**SOC_ETM_SYSTIMER_EVT_CNT_CMP0_ST_CLR** Configures whether or not to clear SYSTIMER_EVT_CNT_CMP0 trigger status.
0: Invalid. No effect
1: Clear
(WT)

**SOC_ETM_SYSTIMER_EVT_CNT_CMP1_ST_CLR** Configures whether or not to clear SYSTIMER_EVT_CNT_CMP1 trigger status.
0: Invalid. No effect
1: Clear
(WT)

**SOC_ETM_SYSTIMER_EVT_CNT_CMP2_ST_CLR** Configures whether or not to clear SYSTIMER_EVT_CNT_CMP2 trigger status.
0: Invalid. No effect
1: Clear
(WT)

**SOC_ETM_ADC_EVT_CONV_CMPLTO_ST_CLR** Configures whether or not to clear ADC_EVT_CONV_CMPLTO trigger status.
0: Invalid. No effect
1: Clear
(WT)

**SOC_ETM_ADC_EVT_EQ_ABOVE_THRESH0_ST_CLR** Configures whether or not to clear ADC_EVT_EQ_ABOVE_THRESH0 trigger status.
0: Invalid. No effect
1: Clear
(WT)

**SOC_ETM_ADC_EVT_EQ_ABOVE_THRESH1_ST_CLR** Configures whether or not to clear ADC_EVT_EQ_ABOVE_THRESH1 trigger status.
0: Invalid. No effect
1: Clear
(WT)

Continued on the next page...

Register 11.19. SOC_ETM_EVT_ST1_CLR_REG (0x01B4)

Continued from the previous page...

**SOC_ETM_ADC_EVT_EQ_BELOW_THRESH0_ST_CLR** Configures whether or not to clear ADC_EVT_EQ_BELOW_THRESH0 trigger status.
0: Invalid. No effect
1: Clear
(WT)

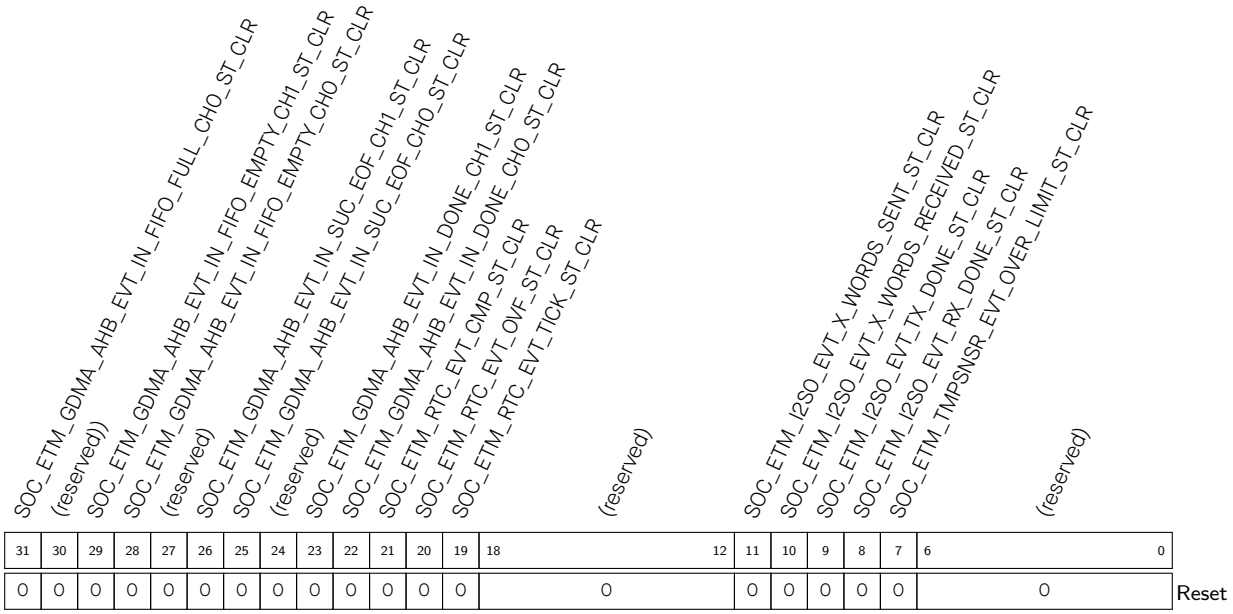
**SOC_ETM_ADC_EVT_EQ_BELOW_THRESH1_ST_CLR** Configures whether or not to clear ADC_EVT_EQ_BELOW_THRESH1 trigger status.
0: Invalid. No effect
1: Clear
(WT)

**SOC_ETM_ADC_EVT_RESULT_DONE0_ST_CLR** Configures whether or not to clear ADC_EVT_RESULT_DONE0 trigger status.
0: Invalid. No effect
1: Clear
(WT)

**SOC_ETM_ADC_EVT_STOPPED0_ST_CLR** Configures whether or not to clear ADC_EVT_STOPPED0 trigger status.
0: Invalid. No effect
1: Clear
(WT)

**SOC_ETM_ADC_EVT_STARTED0_ST_CLR** Configures whether or not to clear ADC_EVT_STARTED0 trigger status.
0: Invalid. No effect
1: Clear
(WT)

Register 11.20. SOC_ETM_EVT_ST2_CLR_REG (0x01BC)




**SOC_ETM_TMPSNSR_EVT_OVER_LIMIT_ST_CLR** Configures whether or not to clear TMPSNSR_EVT_OVER_LIMIT trigger status.

0: Invalid. No effect

1: Clear

(WT)

**SOC_ETM_I2S0_EVT_RX_DONE_ST_CLR** Configures whether or not to clear I2S0_EVT_RX_DONE trigger status.

0: Invalid. No effect

1: Clear

(WT)

**SOC_ETM_I2S0_EVT_TX_DONE_ST_CLR** Configures whether or not to clear I2S0_EVT_TX_DONE trigger status.

0: Invalid. No effect

1: Clear

(WT)

**SOC_ETM_I2S0_EVT_X_WORDS_RECEIVED_ST_CLR** Configures whether or not to clear I2S0_EVT_X_WORDS_RECEIVED trigger status.

0: Invalid. No effect

1: Clear

(WT)

**SOC_ETM_I2S0_EVT_X_WORDS_SENT_ST_CLR** Configures whether or not to clear I2S0_EVT_X_WORDS_SENT trigger status.

0: Invalid. No effect

1: Clear

(WT)

Continued on the next page...

Register 11.20. SOC_ETM_EVT_ST2_CLR_REG (0x01BC)

Continued from the previous page...

**SOC_ETM_RTC_EVT_TICK_ST_CLR** Configures whether or not to clear RTC_EVT_TICK trigger status.
0: Invalid. No effect
1: Clear
(WT)

**SOC_ETM_RTC_EVT_OVF_ST_CLR** Configures whether or not to clear RTC_EVT_OVF trigger status.
0: Invalid. No effect
1: Clear
(WT)

**SOC_ETM_RTC_EVT_CMP_ST_CLR** Configures whether or not to clear RTC_EVT_CMP trigger status.
0: Invalid. No effect
1: Clear
(WT)

**SOC_ETM_GDMA_AHB_EVT_IN_DONE_CH0_ST_CLR** Configures whether or not to clear GDMA_AHB_EVT_IN_DONE_CH0 trigger status.
0: Invalid. No effect
1: Clear
(WT)

**SOC_ETM_GDMA_AHB_EVT_IN_DONE_CH1_ST_CLR** Configures whether or not to clear GDMA_AHB_EVT_IN_DONE_CH1 trigger status.
0: Invalid. No effect
1: Clear
(WT)

**SOC_ETM_GDMA_AHB_EVT_IN_SUC_EOF_CH0_ST_CLR** Configures whether or not to clear GDMA_AHB_EVT_IN_SUC_EOF_CH0 trigger status.
0: Invalid. No effect
1: Clear
(WT)

**SOC_ETM_GDMA_AHB_EVT_IN_SUC_EOF_CH1_ST_CLR** Configures whether or not to clear GDMA_AHB_EVT_IN_SUC_EOF_CH1 trigger status.
0: Invalid. No effect
1: Clear
(WT)

Continued on the next page...

Register 11.20. SOC_ETM_EVT_ST2_CLR_REG (0x01BC)

Continued from the previous page...

**SOC_ETM_GDMA_AHB_EVT_IN_FIFO_EMPTY_CH0_ST_CLR** Configures whether or not to clear GDMA_AHB_EVT_IN_FIFO_EMPTY_CH0 trigger status.

0: Invalid. No effect

1: Clear

(WT)

**SOC_ETM_GDMA_AHB_EVT_IN_FIFO_EMPTY_CH1_ST_CLR** Configures whether or not to clear GDMA_AHB_EVT_IN_FIFO_EMPTY_CH1 trigger status.

0: Invalid. No effect

1: Clear

(WT)

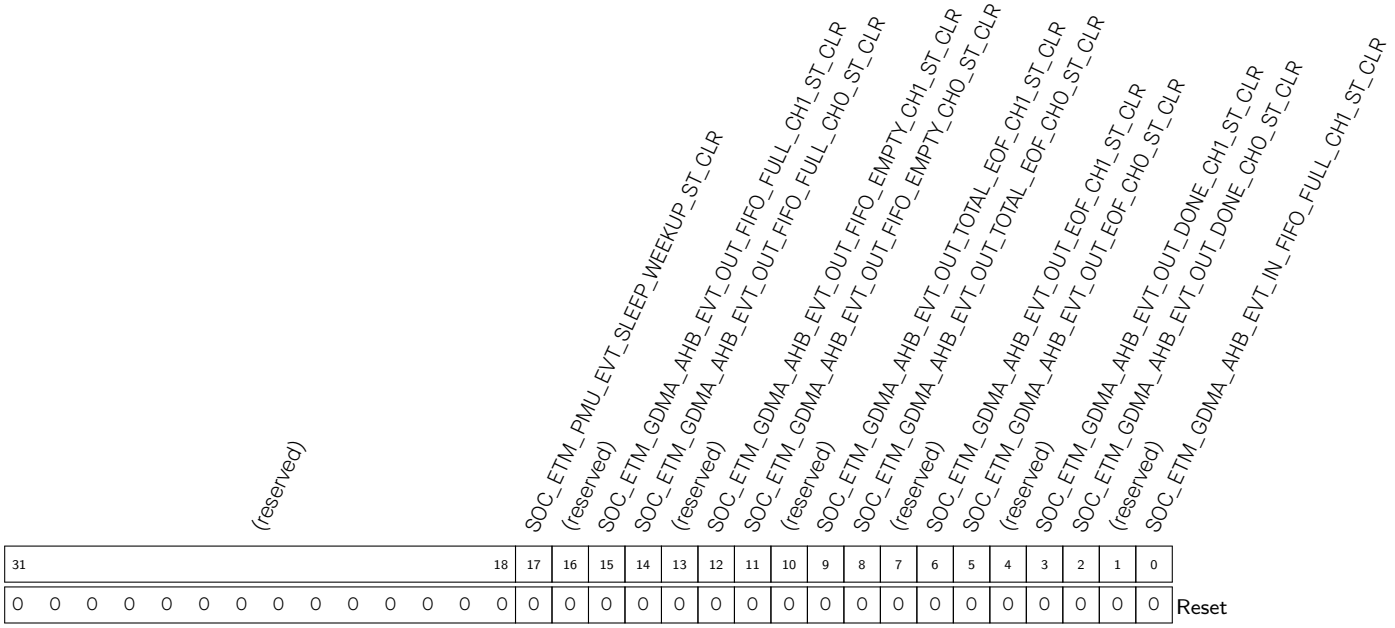
**SOC_ETM_GDMA_AHB_EVT_IN_FIFO_FULL_CH0_ST_CLR** Configures whether or not to clear GDMA_AHB_EVT_IN_FIFO_FULL_CH0 trigger status.

0: Invalid. No effect

1: Clear

(WT)

Register 11.21. SOC_ETM_EVT_ST3_CLR_REG (0x01C4)




**SOC_ETM_GDMA_AHB_EVT_IN_FIFO_FULL_CH1_ST_CLR** Configures whether or not to clear GDMA_AHB_EVT_IN_FIFO_FULL_CH1 trigger status.

0: Invalid. No effect

1: Clear

(WT)

**SOC_ETM_GDMA_AHB_EVT_OUT_DONE_CH0_ST_CLR** Configures whether or not to clear GDMA_AHB_EVT_OUT_DONE_CH0 trigger status.

0: Invalid. No effect

1: Clear

(WT)

**SOC_ETM_GDMA_AHB_EVT_OUT_DONE_CH1_ST_CLR** Configures whether or not to clear GDMA_AHB_EVT_OUT_DONE_CH1 trigger status.

0: Invalid. No effect

1: Clear

(WT)

**SOC_ETM_GDMA_AHB_EVT_OUT_EOF_CH0_ST_CLR** Configures whether or not to clear GDMA_AHB_EVT_OUT_EOF_CH0 trigger status.

0: Invalid. No effect

1: Clear

(WT)

**SOC_ETM_GDMA_AHB_EVT_OUT_EOF_CH1_ST_CLR** Configures whether or not to clear GDMA_AHB_EVT_OUT_EOF_CH1 trigger status.

0: Invalid. No effect

1: Clear

(WT)

Continued on the next page...

Register 11.21. SOC_ETM_EVT_ST3_CLR_REG (0x01C4)

Continued from the previous page...

**SOC_ETM_GDMA_AHB_EVT_OUT_TOTAL_EOF_CH0_ST_CLR** Configures whether or not to clear GDMA_AHB_EVT_OUT_TOTAL_EOF_CH0 trigger status.
0: Invalid. No effect
1: Clear
(WT)

**SOC_ETM_GDMA_AHB_EVT_OUT_TOTAL_EOF_CH1_ST_CLR** Configures whether or not to clear GDMA_AHB_EVT_OUT_TOTAL_EOF_CH1 trigger status.
0: Invalid. No effect
1: Clear
(WT)

**SOC_ETM_GDMA_AHB_EVT_OUT_FIFO_EMPTY_CH0_ST_CLR** Configures whether or not to clear GDMA_AHB_EVT_OUT_FIFO_EMPTY_CH0 trigger status.
0: Invalid. No effect
1: Clear
(WT)

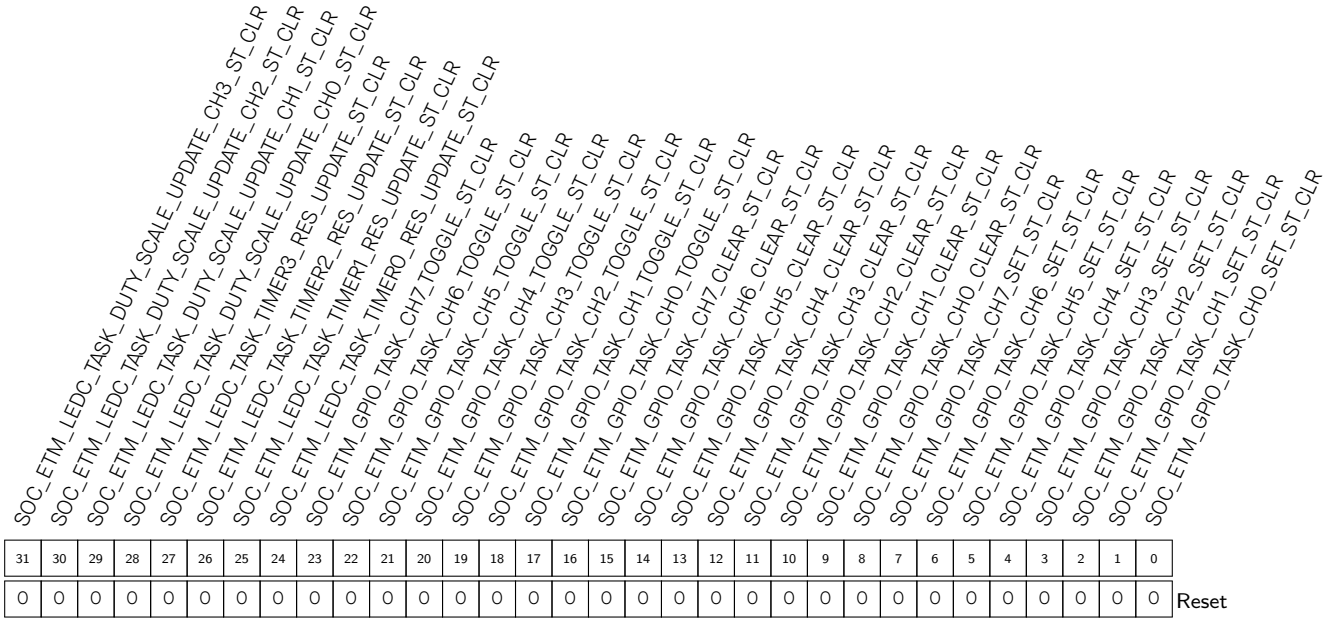
**SOC_ETM_GDMA_AHB_EVT_OUT_FIFO_EMPTY_CH1_ST_CLR** Configures whether or not to clear GDMA_AHB_EVT_OUT_FIFO_EMPTY_CH1 trigger status.
0: Invalid. No effect
1: Clear
(WT)

**SOC_ETM_GDMA_AHB_EVT_OUT_FIFO_FULL_CH0_ST_CLR** Configures whether or not to clear GDMA_AHB_EVT_OUT_FIFO_FULL_CH0 trigger status.
0: Invalid. No effect
1: Clear
(WT)

**SOC_ETM_GDMA_AHB_EVT_OUT_FIFO_FULL_CH1_ST_CLR** Configures whether or not to clear GDMA_AHB_EVT_OUT_FIFO_FULL_CH1 trigger status.
0: Invalid. No effect
1: Clear
(WT)

**SOC_ETM_PMU_EVT_SLEEP_WEEKUP_ST_CLR** Configures whether or not to clear PMU_EVT_SLEEP_WEEKUP trigger status.
0: Invalid. No effect
1: Clear
(WT)

Register 11.22. SOC_ETM_TASK_ST0_CLR_REG (0x01CC)




**SOC_ETM_GPIO_TASK_CH0_SET_ST_CLR** Configures whether or not to clear GPIO_TASK_CH0_SET trigger status.

0: Invalid. No effect

1: Clear

(WT)

**SOC_ETM_GPIO_TASK_CH1_SET_ST_CLR** Configures whether or not to clear GPIO_TASK_CH1_SET trigger status.

0: Invalid. No effect

1: Clear

(WT)

**SOC_ETM_GPIO_TASK_CH2_SET_ST_CLR** Configures whether or not to clear GPIO_TASK_CH2_SET trigger status.

0: Invalid. No effect

1: Clear

(WT)

**SOC_ETM_GPIO_TASK_CH3_SET_ST_CLR** Configures whether or not to clear GPIO_TASK_CH3_SET trigger status.

0: Invalid. No effect

1: Clear

(WT)

**SOC_ETM_GPIO_TASK_CH4_SET_ST_CLR** Configures whether or not to clear GPIO_TASK_CH4_SET trigger status.

0: Invalid. No effect

1: Clear

(WT)

**Continued on the next page...**

Register 11.22. SOC_ETM_TASK_ST0_CLR_REG (0x01CC)

Continued from the previous page...

**SOC_ETM_GPIO_TASK_CH5_SET_ST_CLR** Configures whether or not to clear GPIO_TASK_CH5_SET trigger status.
0: Invalid. No effect
1: Clear
(WT)

**SOC_ETM_GPIO_TASK_CH6_SET_ST_CLR** Configures whether or not to clear GPIO_TASK_CH6_SET trigger status.
0: Invalid. No effect
1: Clear
(WT)

**SOC_ETM_GPIO_TASK_CH7_SET_ST_CLR** Configures whether or not to clear GPIO_TASK_CH7_SET trigger status.
0: Invalid. No effect
1: Clear
(WT)

**SOC_ETM_GPIO_TASK_CH0_CLEAR_ST_CLR** Configures whether or not to clear GPIO_TASK_CH0_CLEAR trigger status.
0: Invalid. No effect
1: Clear
(WT)

**SOC_ETM_GPIO_TASK_CH1_CLEAR_ST_CLR** Configures whether or not to clear GPIO_TASK_CH1_CLEAR trigger status.
0: Invalid. No effect
1: Clear
(WT)

**SOC_ETM_GPIO_TASK_CH2_CLEAR_ST_CLR** Configures whether or not to clear GPIO_TASK_CH2_CLEAR trigger status.
0: Invalid. No effect
1: Clear
(WT)

**SOC_ETM_GPIO_TASK_CH3_CLEAR_ST_CLR** Configures whether or not to clear GPIO_TASK_CH3_CLEAR trigger status.
0: Invalid. No effect
1: Clear
(WT)

Continued on the next page...

Register 11.22. SOC_ETM_TASK_ST0_CLR_REG (0x01CC)

Continued from the previous page...

**SOC_ETM_GPIO_TASK_CH4_CLEAR_ST_CLR** Configures whether or not to clear GPIO_TASK_CH4_CLEAR trigger status.
0: Invalid. No effect
1: Clear
(WT)

**SOC_ETM_GPIO_TASK_CH5_CLEAR_ST_CLR** Configures whether or not to clear GPIO_TASK_CH5_CLEAR trigger status.
0: Invalid. No effect
1: Clear
(WT)

**SOC_ETM_GPIO_TASK_CH6_CLEAR_ST_CLR** Configures whether or not to clear GPIO_TASK_CH6_CLEAR trigger status.
0: Invalid. No effect
1: Clear
(WT)

**SOC_ETM_GPIO_TASK_CH7_CLEAR_ST_CLR** Configures whether or not to clear GPIO_TASK_CH7_CLEAR trigger status.
0: Invalid. No effect
1: Clear
(WT)

**SOC_ETM_GPIO_TASK_CH0_TOGGLE_ST_CLR** Configures whether or not to clear GPIO_TASK_CH0_TOGGLE trigger status.
0: Invalid. No effect
1: Clear
(WT)

**SOC_ETM_GPIO_TASK_CH1_TOGGLE_ST_CLR** Configures whether or not to clear GPIO_TASK_CH1_TOGGLE trigger status.
0: Invalid. No effect
1: Clear
(WT)

**SOC_ETM_GPIO_TASK_CH2_TOGGLE_ST_CLR** Configures whether or not to clear GPIO_TASK_CH2_TOGGLE trigger status.
0: Invalid. No effect
1: Clear
(WT)

Continued on the next page...

Register 11.22. SOC_ETM_TASK_ST0_CLR_REG (0x01CC)

Continued from the previous page...

**SOC_ETM_GPIO_TASK_CH3_TOGGLE_ST_CLR** Configures whether or not to clear GPIO_TASK_CH3_TOGGLE trigger status.
0: Invalid. No effect
1: Clear
(WT)

**SOC_ETM_GPIO_TASK_CH4_TOGGLE_ST_CLR** Configures whether or not to clear GPIO_TASK_CH4_TOGGLE trigger status.
0: Invalid. No effect
1: Clear
(WT)

**SOC_ETM_GPIO_TASK_CH5_TOGGLE_ST_CLR** Configures whether or not to clear GPIO_TASK_CH5_TOGGLE trigger status.
0: Invalid. No effect
1: Clear
(WT)

**SOC_ETM_GPIO_TASK_CH6_TOGGLE_ST_CLR** Configures whether or not to clear GPIO_TASK_CH6_TOGGLE trigger status.
0: Invalid. No effect
1: Clear
(WT)

**SOC_ETM_GPIO_TASK_CH7_TOGGLE_ST_CLR** Configures whether or not to clear GPIO_TASK_CH7_TOGGLE trigger status.
0: Invalid. No effect
1: Clear
(WT)

**SOC_ETM_LEDC_TASK_TIMER0_RES_UPDATE_ST_CLR** Configures whether or not to clear LEDC_TASK_TIMER0_RES_UPDATE trigger status.
0: Invalid. No effect
1: Clear
(WT)

**SOC_ETM_LEDC_TASK_TIMER1_RES_UPDATE_ST_CLR** Configures whether or not to clear LEDC_TASK_TIMER1_RES_UPDATE trigger status.
0: Invalid. No effect
1: Clear
(WT)

Continued on the next page...

Register 11.22. SOC_ETM_TASK_ST0_CLR_REG (0x01CC)

Continued from the previous page...

**SOC_ETM_LEDC_TASK_TIMER2_RES_UPDATE_ST_CLR** Configures whether or not to clear LEDC_TASK_TIMER2_RES_UPDATE trigger status.
0: Invalid. No effect
1: Clear
(WT)

**SOC_ETM_LEDC_TASK_TIMER3_RES_UPDATE_ST_CLR** Configures whether or not to clear LEDC_TASK_TIMER3_RES_UPDATE trigger status.
0: Invalid. No effect
1: Clear
(WT)

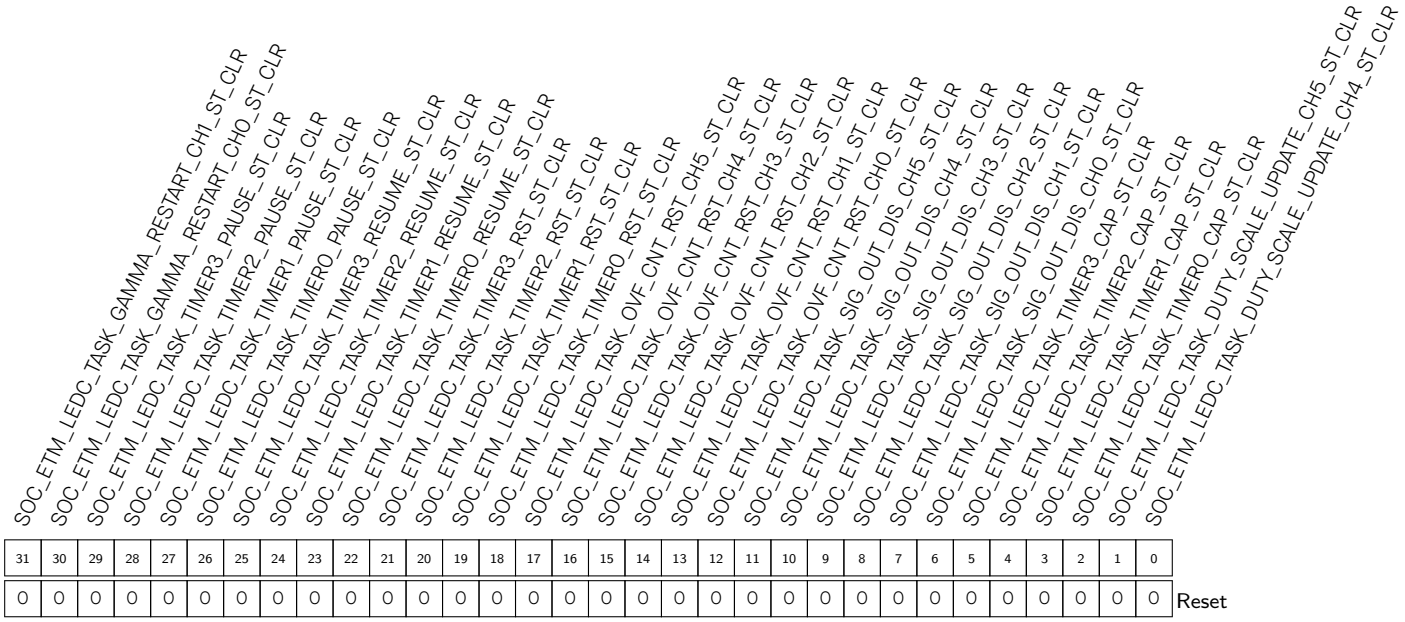
**SOC_ETM_LEDC_TASK_DUTY_SCALE_UPDATE_CH0_ST_CLR** Configures whether or not to clear LEDC_TASK_DUTY_SCALE_UPDATE_CH0 trigger status.
0: Invalid. No effect
1: Clear
(WT)

**SOC_ETM_LEDC_TASK_DUTY_SCALE_UPDATE_CH1_ST_CLR** Configures whether or not to clear LEDC_TASK_DUTY_SCALE_UPDATE_CH1 trigger status.
0: Invalid. No effect
1: Clear
(WT)

**SOC_ETM_LEDC_TASK_DUTY_SCALE_UPDATE_CH2_ST_CLR** Configures whether or not to clear LEDC_TASK_DUTY_SCALE_UPDATE_CH2 trigger status.
0: Invalid. No effect
1: Clear
(WT)

**SOC_ETM_LEDC_TASK_DUTY_SCALE_UPDATE_CH3_ST_CLR** Configures whether or not to clear LEDC_TASK_DUTY_SCALE_UPDATE_CH3 trigger status.
0: Invalid. No effect
1: Clear
(WT)

Register 11.23. SOC_ETM_TASK_ST1_CLR_REG (0x01D4)




**SOC_ETM_LEDC_TASK_DUTY_SCALE_UPDATE_CH4_ST_CLR** Configures whether or not to clear LEDC_TASK_DUTY_SCALE_UPDATE_CH4 trigger status.

0: Invalid. No effect

1: Clear

(WT)

**SOC_ETM_LEDC_TASK_DUTY_SCALE_UPDATE_CH5_ST_CLR** Configures whether or not to clear LEDC_TASK_DUTY_SCALE_UPDATE_CH5 trigger status.

0: Invalid. No effect

1: Clear

(WT)

**SOC_ETM_LEDC_TASK_TIMER0_CAP_ST_CLR** Configures whether or not to clear LEDC_TASK_TIMER0_CAP trigger status.

0: Invalid. No effect

1: Clear

(WT)

**SOC_ETM_LEDC_TASK_TIMER1_CAP_ST_CLR** Configures whether or not to clear LEDC_TASK_TIMER1_CAP trigger status.

0: Invalid. No effect

1: Clear

(WT)

**SOC_ETM_LEDC_TASK_TIMER2_CAP_ST_CLR** Configures whether or not to clear LEDC_TASK_TIMER2_CAP trigger status.

0: Invalid. No effect

1: Clear

(WT)

Continued on the next page...

Register 11.23. SOC_ETM_TASK_ST1_CLR_REG (0x01D4)

Continued from the previous page...

**SOC_ETM_LEDC_TASK_TIMER3_CAP_ST_CLR** Configures whether or not to clear LEDC_TASK_TIMER3_CAP trigger status.
0: Invalid. No effect
1: Clear
(WT)

**SOC_ETM_LEDC_TASK_SIG_OUT_DIS_CH0_ST_CLR** Configures whether or not to clear LEDC_TASK_SIG_OUT_DIS_CH0 trigger status.
0: Invalid. No effect
1: Clear
(WT)

**SOC_ETM_LEDC_TASK_SIG_OUT_DIS_CH1_ST_CLR** Configures whether or not to clear LEDC_TASK_SIG_OUT_DIS_CH1 trigger status.
0: Invalid. No effect
1: Clear
(WT)

**SOC_ETM_LEDC_TASK_SIG_OUT_DIS_CH2_ST_CLR** Configures whether or not to clear LEDC_TASK_SIG_OUT_DIS_CH2 trigger status.
0: Invalid. No effect
1: Clear
(WT)

**SOC_ETM_LEDC_TASK_SIG_OUT_DIS_CH3_ST_CLR** Configures whether or not to clear LEDC_TASK_SIG_OUT_DIS_CH3 trigger status.
0: Invalid. No effect
1: Clear
(WT)

**SOC_ETM_LEDC_TASK_SIG_OUT_DIS_CH4_ST_CLR** Configures whether or not to clear LEDC_TASK_SIG_OUT_DIS_CH4 trigger status.
0: Invalid. No effect
1: Clear
(WT)

**SOC_ETM_LEDC_TASK_SIG_OUT_DIS_CH5_ST_CLR** Configures whether or not to clear LEDC_TASK_SIG_OUT_DIS_CH5 trigger status.
0: Invalid. No effect
1: Clear
(WT)

Continued on the next page...

Register 11.23. SOC_ETM_TASK_ST1_CLR_REG (0x01D4)

Continued from the previous page...

**SOC_ETM_LEDC_TASK_OVF_CNT_RST_CH0_ST_CLR** Configures whether or not to clear LEDC_TASK_OVF_CNT_RST_CH0 trigger status.
0: Invalid. No effect
1: Clear
(WT)

**SOC_ETM_LEDC_TASK_OVF_CNT_RST_CH1_ST_CLR** Configures whether or not to clear LEDC_TASK_OVF_CNT_RST_CH1 trigger status.
0: Invalid. No effect
1: Clear
(WT)

**SOC_ETM_LEDC_TASK_OVF_CNT_RST_CH2_ST_CLR** Configures whether or not to clear LEDC_TASK_OVF_CNT_RST_CH2 trigger status.
0: Invalid. No effect
1: Clear
(WT)

**SOC_ETM_LEDC_TASK_OVF_CNT_RST_CH3_ST_CLR** Configures whether or not to clear LEDC_TASK_OVF_CNT_RST_CH3 trigger status.
0: Invalid. No effect
1: Clear
(WT)

**SOC_ETM_LEDC_TASK_OVF_CNT_RST_CH4_ST_CLR** Configures whether or not to clear LEDC_TASK_OVF_CNT_RST_CH4 trigger status.
0: Invalid. No effect
1: Clear
(WT)

**SOC_ETM_LEDC_TASK_OVF_CNT_RST_CH5_ST_CLR** Configures whether or not to clear LEDC_TASK_OVF_CNT_RST_CH5 trigger status.
0: Invalid. No effect
1: Clear
(WT)

**SOC_ETM_LEDC_TASK_TIMER0_RST_ST_CLR** Configures whether or not to clear LEDC_TASK_TIMER0_RST trigger status.
0: Invalid. No effect
1: Clear
(WT)

Continued on the next page...

Register 11.23. SOC_ETM_TASK_ST1_CLR_REG (0x01D4)

Continued from the previous page...

**SOC_ETM_LEDC_TASK_TIMER1_RST_ST_CLR** Configures whether or not to clear LEDC_TASK_TIMER1_RST trigger status.
0: Invalid. No effect
1: Clear
(WT)

**SOC_ETM_LEDC_TASK_TIMER2_RST_ST_CLR** Configures whether or not to clear LEDC_TASK_TIMER2_RST trigger status.
0: Invalid. No effect
1: Clear
(WT)

**SOC_ETM_LEDC_TASK_TIMER3_RST_ST_CLR** Configures whether or not to clear LEDC_TASK_TIMER3_RST trigger status.
0: Invalid. No effect
1: Clear
(WT)

**SOC_ETM_LEDC_TASK_TIMER0_RESUME_ST_CLR** Configures whether or not to clear LEDC_TASK_TIMER0_RESUME trigger status.
0: Invalid. No effect
1: Clear
(WT)

**SOC_ETM_LEDC_TASK_TIMER1_RESUME_ST_CLR** Configures whether or not to clear LEDC_TASK_TIMER1_RESUME trigger status.
0: Invalid. No effect
1: Clear
(WT)

**SOC_ETM_LEDC_TASK_TIMER2_RESUME_ST_CLR** Configures whether or not to clear LEDC_TASK_TIMER2_RESUME trigger status.
0: Invalid. No effect
1: Clear
(WT)

**SOC_ETM_LEDC_TASK_TIMER3_RESUME_ST_CLR** Configures whether or not to clear LEDC_TASK_TIMER3_RESUME trigger status.
0: Invalid. No effect
1: Clear
(WT)

Continued on the next page...

Register 11.23. SOC_ETM_TASK_ST1_CLR_REG (0x01D4)

Continued from the previous page...

**SOC_ETM_LEDC_TASK_TIMER0_PAUSE_ST_CLR** Configures whether or not to clear LEDC_TASK_TIMER0_PAUSE trigger status.
0: Invalid. No effect
1: Clear
(WT)

**SOC_ETM_LEDC_TASK_TIMER1_PAUSE_ST_CLR** Configures whether or not to clear LEDC_TASK_TIMER1_PAUSE trigger status.
0: Invalid. No effect
1: Clear
(WT)

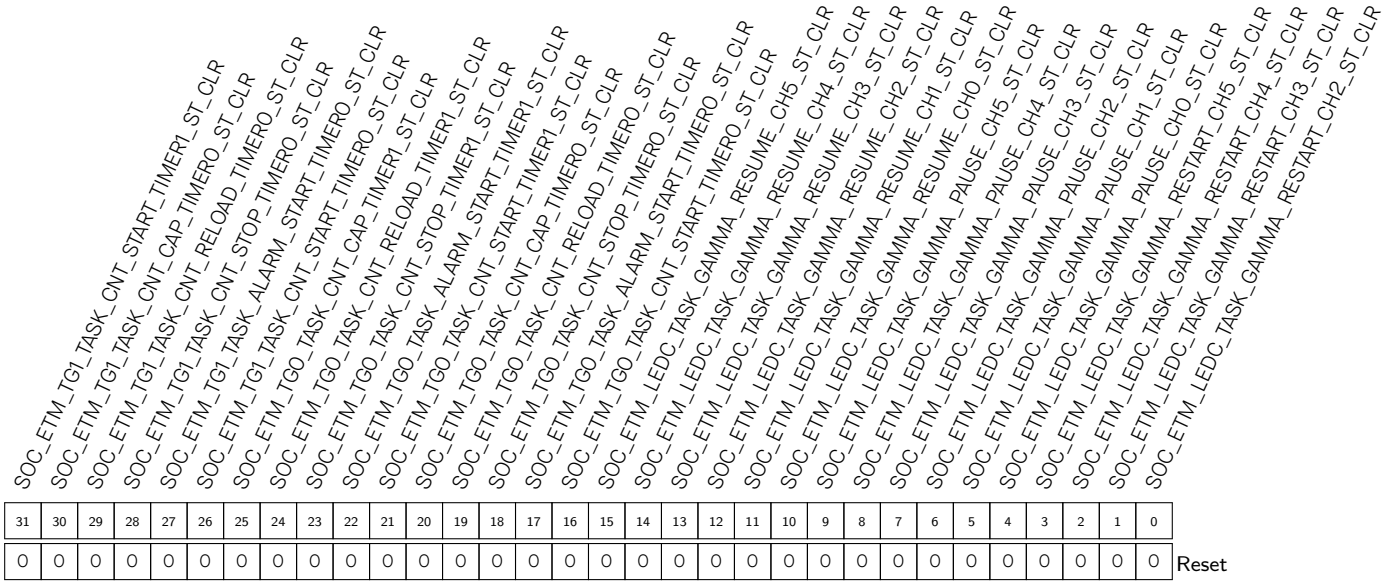
**SOC_ETM_LEDC_TASK_TIMER2_PAUSE_ST_CLR** Configures whether or not to clear LEDC_TASK_TIMER2_PAUSE trigger status.
0: Invalid. No effect
1: Clear
(WT)

**SOC_ETM_LEDC_TASK_TIMER3_PAUSE_ST_CLR** Configures whether or not to clear LEDC_TASK_TIMER3_PAUSE trigger status.
0: Invalid. No effect
1: Clear
(WT)

**SOC_ETM_LEDC_TASK_GAMMA_RESTART_CH0_ST_CLR** Configures whether or not to clear LEDC_TASK_GAMMA_RESTART_CH0 trigger status.
0: Invalid. No effect
1: Clear
(WT)

**SOC_ETM_LEDC_TASK_GAMMA_RESTART_CH1_ST_CLR** Configures whether or not to clear LEDC_TASK_GAMMA_RESTART_CH1 trigger status.
0: Invalid. No effect
1: Clear
(WT)

Register 11.24. SOC_ETM_TASK_ST2_CLR_REG (0x01DC)




**SOC_ETM_LEDC_TASK_GAMMA_RESTART_CH2_ST_CLR** Configures whether or not to clear LEDC_TASK_GAMMA_RESTART_CH2 trigger status.

0: Invalid. No effect

1: Clear

(WT)

**SOC_ETM_LEDC_TASK_GAMMA_RESTART_CH3_ST_CLR** Configures whether or not to clear LEDC_TASK_GAMMA_RESTART_CH3 trigger status.

0: Invalid. No effect

1: Clear

(WT)

**SOC_ETM_LEDC_TASK_GAMMA_RESTART_CH4_ST_CLR** Configures whether or not to clear LEDC_TASK_GAMMA_RESTART_CH4 trigger status.

0: Invalid. No effect

1: Clear

(WT)

**SOC_ETM_LEDC_TASK_GAMMA_RESTART_CH5_ST_CLR** Configures whether or not to clear LEDC_TASK_GAMMA_RESTART_CH5 trigger status.

0: Invalid. No effect

1: Clear

(WT)

**SOC_ETM_LEDC_TASK_GAMMA_PAUSE_CH0_ST_CLR** Configures whether or not to clear LEDC_TASK_GAMMA_PAUSE_CH0 trigger status.

0: Invalid. No effect

1: Clear

(WT)

Continued on the next page...

Register 11.24. SOC_ETM_TASK_ST2_CLR_REG (0x01DC)

Continued from the previous page...

**SOC_ETM_LEDC_TASK_GAMMA_PAUSE_CH1_ST_CLR** Configures whether or not to clear LEDC_TASK_GAMMA_PAUSE_CH1 trigger status.
0: Invalid. No effect
1: Clear
(WT)

**SOC_ETM_LEDC_TASK_GAMMA_PAUSE_CH2_ST_CLR** Configures whether or not to clear LEDC_TASK_GAMMA_PAUSE_CH2 trigger status.
0: Invalid. No effect
1: Clear
(WT)

**SOC_ETM_LEDC_TASK_GAMMA_PAUSE_CH3_ST_CLR** Configures whether or not to clear LEDC_TASK_GAMMA_PAUSE_CH3 trigger status.
0: Invalid. No effect
1: Clear
(WT)

**SOC_ETM_LEDC_TASK_GAMMA_PAUSE_CH4_ST_CLR** Configures whether or not to clear LEDC_TASK_GAMMA_PAUSE_CH4 trigger status.
0: Invalid. No effect
1: Clear
(WT)

**SOC_ETM_LEDC_TASK_GAMMA_PAUSE_CH5_ST_CLR** Configures whether or not to clear LEDC_TASK_GAMMA_PAUSE_CH5 trigger status.
0: Invalid. No effect
1: Clear
(WT)

**SOC_ETM_LEDC_TASK_GAMMA_RESUME_CH0_ST_CLR** Configures whether or not to clear LEDC_TASK_GAMMA_RESUME_CH0 trigger status.
0: Invalid. No effect
1: Clear
(WT)

**SOC_ETM_LEDC_TASK_GAMMA_RESUME_CH1_ST_CLR** Configures whether or not to clear LEDC_TASK_GAMMA_RESUME_CH1 trigger status.
0: Invalid. No effect
1: Clear
(WT)

Continued on the next page...

Register 11.24. SOC_ETM_TASK_ST2_CLR_REG (0x01DC)

Continued from the previous page...

**SOC_ETM_LEDC_TASK_GAMMA_RESUME_CH2_ST_CLR** Configures whether or not to clear LEDC_TASK_GAMMA_RESUME_CH2 trigger status.
0: Invalid. No effect
1: Clear
(WT)

**SOC_ETM_LEDC_TASK_GAMMA_RESUME_CH3_ST_CLR** Configures whether or not to clear LEDC_TASK_GAMMA_RESUME_CH3 trigger status.
0: Invalid. No effect
1: Clear
(WT)

**SOC_ETM_LEDC_TASK_GAMMA_RESUME_CH4_ST_CLR** Configures whether or not to clear LEDC_TASK_GAMMA_RESUME_CH4 trigger status.
0: Invalid. No effect
1: Clear
(WT)

**SOC_ETM_LEDC_TASK_GAMMA_RESUME_CH5_ST_CLR** Configures whether or not to clear LEDC_TASK_GAMMA_RESUME_CH5 trigger status.
0: Invalid. No effect
1: Clear
(WT)

**SOC_ETM_TGO_TASK_CNT_START_TIMER0_ST_CLR** Configures whether or not to clear TGO_TASK_CNT_START_TIMER0 trigger status.
0: Invalid. No effect
1: Clear
(WT)

**SOC_ETM_TGO_TASK_ALARM_START_TIMER0_ST_CLR** Configures whether or not to clear TGO_TASK_ALARM_START_TIMER0 trigger status.
0: Invalid. No effect
1: Clear
(WT)

**SOC_ETM_TGO_TASK_CNT_STOP_TIMER0_ST_CLR** Configures whether or not to clear TGO_TASK_CNT_STOP_TIMER0 trigger status.
0: Invalid. No effect
1: Clear
(WT)

Continued on the next page...

Register 11.24. SOC_ETM_TASK_ST2_CLR_REG (0x01DC)

Continued from the previous page...

**SOC_ETM_TG0_TASK_CNT_RELOAD_TIMER0_ST_CLR** Configures whether or not to clear TG0_TASK_CNT_RELOAD_TIMER0 trigger status.
0: Invalid. No effect
1: Clear
(WT)

**SOC_ETM_TG0_TASK_CNT_CAP_TIMER0_ST_CLR** Configures whether or not to clear TG0_TASK_CNT_CAP_TIMER0 trigger status.
0: Invalid. No effect
1: Clear
(WT)

**SOC_ETM_TG0_TASK_CNT_START_TIMER1_ST_CLR** Configures whether or not to clear TG0_TASK_CNT_START_TIMER1 trigger status.
0: Invalid. No effect
1: Clear
(WT)

**SOC_ETM_TG0_TASK_ALARM_START_TIMER1_ST_CLR** Configures whether or not to clear TG0_TASK_ALARM_START_TIMER1 trigger status.
0: Invalid. No effect
1: Clear
(WT)

**SOC_ETM_TG0_TASK_CNT_STOP_TIMER1_ST_CLR** Configures whether or not to clear TG0_TASK_CNT_STOP_TIMER1 trigger status.
0: Invalid. No effect
1: Clear
(WT)

**SOC_ETM_TG0_TASK_CNT_RELOAD_TIMER1_ST_CLR** Configures whether or not to clear TG0_TASK_CNT_RELOAD_TIMER1 trigger status.
0: Invalid. No effect
1: Clear
(WT)

**SOC_ETM_TG0_TASK_CNT_CAP_TIMER1_ST_CLR** Configures whether or not to clear TG0_TASK_CNT_CAP_TIMER1 trigger status.
0: Invalid. No effect
1: Clear
(WT)

Continued on the next page...

Register 11.24. SOC_ETM_TASK_ST2_CLR_REG (0x01DC)

Continued from the previous page...

**SOC_ETM_TG1_TASK_CNT_START_TIMER0_ST_CLR** Configures whether or not to clear TG1_TASK_CNT_START_TIMER0 trigger status.
0: Invalid. No effect
1: Clear
(WT)

**SOC_ETM_TG1_TASK_ALARM_START_TIMER0_ST_CLR** Configures whether or not to clear TG1_TASK_ALARM_START_TIMER0 trigger status.
0: Invalid. No effect
1: Clear
(WT)

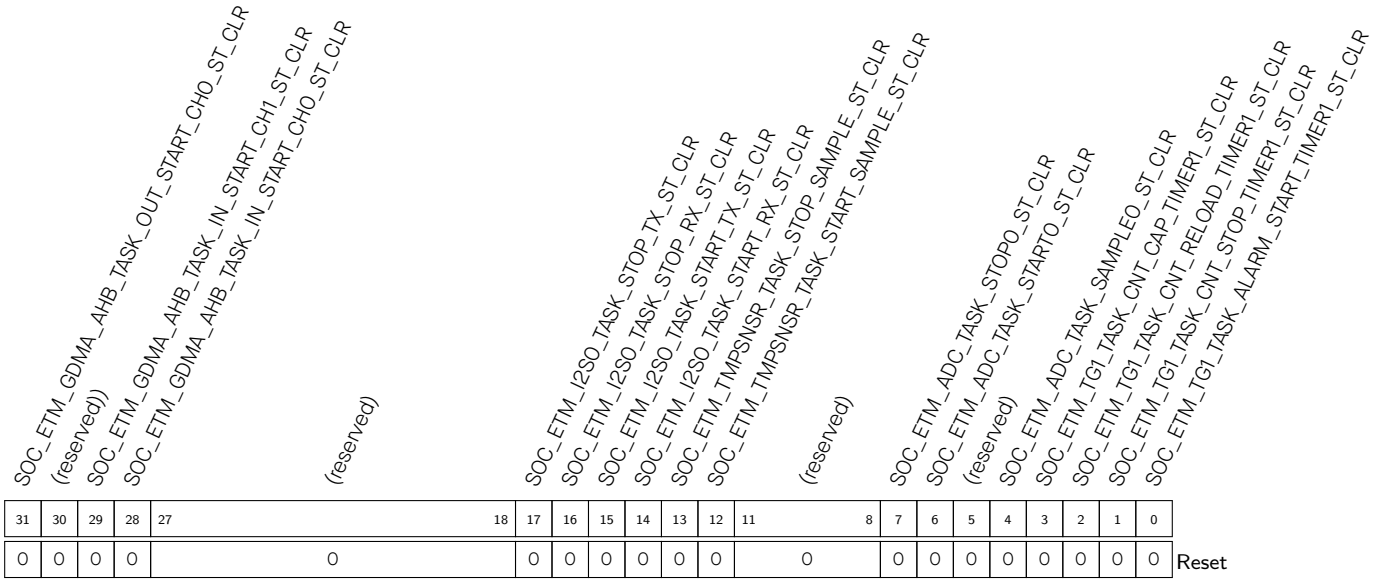
**SOC_ETM_TG1_TASK_CNT_STOP_TIMER0_ST_CLR** Configures whether or not to clear TG1_TASK_CNT_STOP_TIMER0 trigger status.
0: Invalid. No effect
1: Clear
(WT)

**SOC_ETM_TG1_TASK_CNT_RELOAD_TIMER0_ST_CLR** Configures whether or not to clear TG1_TASK_CNT_RELOAD_TIMER0 trigger status.
0: Invalid. No effect
1: Clear
(WT)

**SOC_ETM_TG1_TASK_CNT_CAP_TIMER0_ST_CLR** Configures whether or not to clear TG1_TASK_CNT_CAP_TIMER0 trigger status.
0: Invalid. No effect
1: Clear
(WT)

**SOC_ETM_TG1_TASK_CNT_START_TIMER1_ST_CLR** Configures whether or not to clear TG1_TASK_CNT_START_TIMER1 trigger status.
0: Invalid. No effect
1: Clear
(WT)

Register 11.25. SOC_ETM_TASK_ST3_CLR_REG (0x01E4)




**SOC_ETM_TG1_TASK_ALARM_START_TIMER1_ST_CLR** Configures whether or not to clear TG1_TASK_ALARM_START_TIMER1 trigger status.

0: Invalid. No effect

1: Clear

(WT)

**SOC_ETM_TG1_TASK_CNT_STOP_TIMER1_ST_CLR** Configures whether or not to clear TG1_TASK_CNT_STOP_TIMER1 trigger status.

0: Invalid. No effect

1: Clear

(WT)

**SOC_ETM_TG1_TASK_CNT_RELOAD_TIMER1_ST_CLR** Configures whether or not to clear TG1_TASK_CNT_RELOAD_TIMER1 trigger status.

0: Invalid. No effect

1: Clear

(WT)

**SOC_ETM_TG1_TASK_CNT_CAP_TIMER1_ST_CLR** Configures whether or not to clear TG1_TASK_CNT_CAP_TIMER1 trigger status.

0: Invalid. No effect

1: Clear

(WT)

**SOC_ETM_ADC_TASK_SAMPLE0_ST_CLR** Configures whether or not to clear ADC_TASK_SAMPLE0 trigger status.

0: Invalid. No effect

1: Clear

(WT)

**Continued on the next page...**

Register 11.25. SOC_ETM_TASK_ST3_CLR_REG (0x01E4)

Continued from the previous page...

**SOC_ETM_ADC_TASK_START0_ST_CLR** Configures whether or not to clear ADC_TASK_START0 trigger status.
0: Invalid. No effect
1: Clear
(WT)

**SOC_ETM_ADC_TASK_STOP0_ST_CLR** Configures whether or not to clear ADC_TASK_STOP0 trigger status.
0: Invalid. No effect
1: Clear
(WT)

**SOC_ETM_TMPSNSR_TASK_START_SAMPLE_ST_CLR** Configures whether or not to clear TMPSNSR_TASK_START_SAMPLE trigger status.
0: Invalid. No effect
1: Clear
(WT)

**SOC_ETM_TMPSNSR_TASK_STOP_SAMPLE_ST_CLR** Configures whether or not to clear TMPSNSR_TASK_STOP_SAMPLE trigger status.
0: Invalid. No effect
1: Clear
(WT)

**SOC_ETM_I2S0_TASK_START_RX_ST_CLR** Configures whether or not to clear I2S0_TASK_START_RX trigger status.
0: Invalid. No effect
1: Clear
(WT)

**SOC_ETM_I2S0_TASK_START_TX_ST_CLR** Configures whether or not to clear I2S0_TASK_START_TX trigger status.
0: Invalid. No effect
1: Clear
(WT)

Continued on the next page...

Register 11.25. SOC_ETM_TASK_ST3_CLR_REG (0x01E4)

Continued from the previous page...

**SOC_ETM_I2S0_TASK_STOP_RX_ST_CLR** Configures whether or not to clear I2S0_TASK_STOP_RX trigger status.
0: Invalid. No effect
1: Clear
(WT)

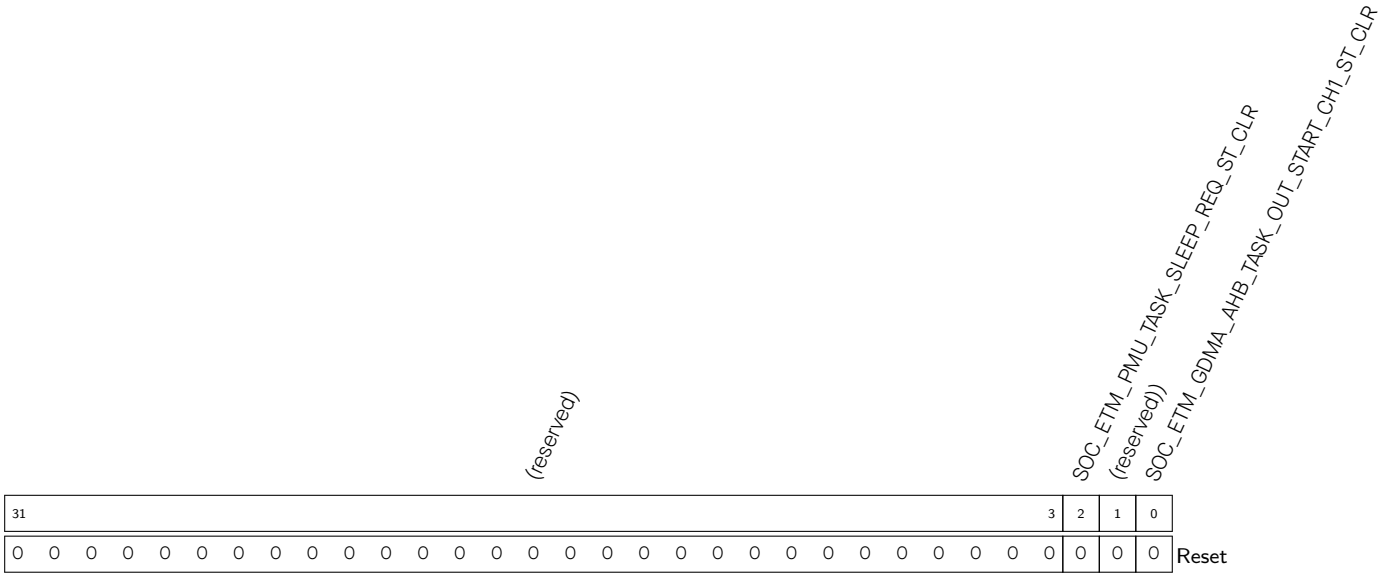
**SOC_ETM_I2S0_TASK_STOP_TX_ST_CLR** Configures whether or not to clear I2S0_TASK_STOP_TX trigger status.
0: Invalid. No effect
1: Clear
(WT)

**SOC_ETM_GDMA_AHB_TASK_IN_START_CH0_ST_CLR** Configures whether or not to clear GDMA_AHB_TASK_IN_START_CH0 trigger status.
0: Invalid. No effect
1: Clear
(WT)

**SOC_ETM_GDMA_AHB_TASK_IN_START_CH1_ST_CLR** Configures whether or not to clear GDMA_AHB_TASK_IN_START_CH1 trigger status.
0: Invalid. No effect
1: Clear
(WT)

**SOC_ETM_GDMA_AHB_TASK_OUT_START_CH0_ST_CLR** Configures whether or not to clear GDMA_AHB_TASK_OUT_START_CH0 trigger status.
0: Invalid. No effect
1: Clear
(WT)

Register 11.26. SOC_ETM_TASK_ST4_CLR_REG (0x01EC)




**SOC_ETM_GDMA_AHB_TASK_OUT_START_CH1_ST_CLR** Configures whether or not to clear GDMA_AHB_TASK_OUT_START_CH1 trigger status.

0: Invalid. No effect

1: Clear

(WT)

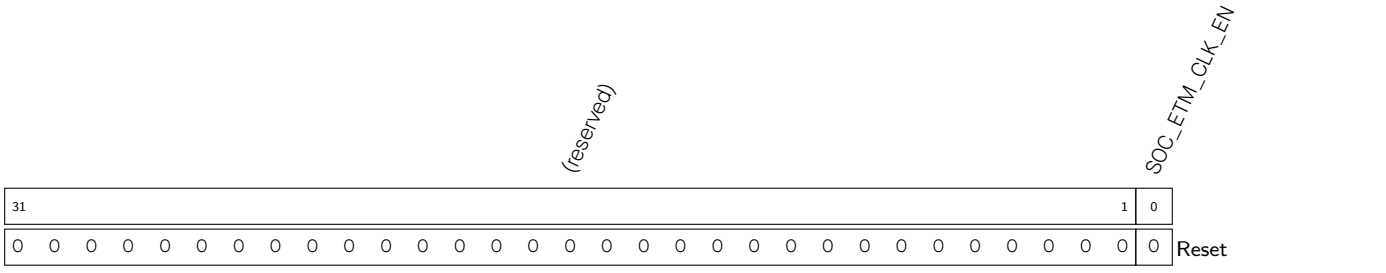
**SOC_ETM_PMU_TASK_SLEEP_REQ_ST_CLR** Configures whether or not to clear PMU_TASK_SLEEP_REQ trigger status.

0: Invalid. No effect

1: Clear

(WT)

Register 11.27. SOC_ETM_CLK_EN_REG (0x01F0)



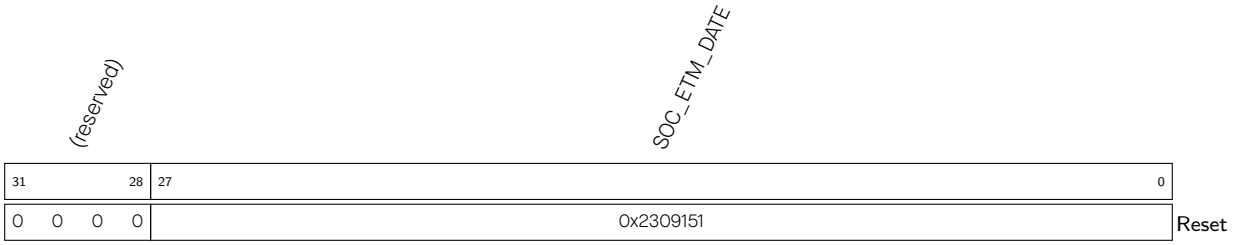

**SOC_ETM_CLK_EN** Configures whether or not to open register clock gate.

0: Open the clock gate only when application writes registers

1: Force open the clock gate for register

(R/W)

**Register 11.28. SOC_ETM_DATE_REG (0x01F4)**




**SOC_ETM_DATE** Version control register. (R/W)

# Chapter 12

# Low-Power Management

## 12.1 Overview

ESP32-C61 features an advanced low-power management system that can optimize 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.

The power management unit (PMU) is a hardware component that is the core part of the low-power management system and is responsible for powering up and down different power domains of the chip, to achieve the best balance among chip performance, power consumption, and wake-up latency.

## 12.2 Terminology

The following terms related to low-power management are defined in the context of the *ESP32-C61 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 the 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 four states of the PMU's state machine. Users can configure the clock gating and power gating of a power domain in each of the four states. |
| **Power modes** | Refers to the five preset power modes that power up different domains for typical application scenarios. |

## 12.3 Feature List

The PMU has the following features:

- Supports four configurable PMU states. Software can flexibly configure them according to specific needs.
  - HP_ACTIVE

  - HP_MODEM
  - HP_SLEEP
  - LP_SLEEP
- Supports five preset power modes that suit various typical usage scenarios:
  - Active
  - Modem-sleep
  - Light-sleep0
  - Light-sleep1
  - Deep-sleep
- 10 always-on (AON) registers
- Programmable retention DMA to backup and restore the states of the CPU and peripherals when the chip switches between PMU states
- Supports a power controller that controls the power and clocks depending on the power modes

## 12.4 Functional Description

ESP32-C61's low-power management involves the following components:

- **Power scheme**: includes power regulators, digital power domains, and analog power domains, etc.
- **PMU controller**: controls the power up and down of the power domains, clocks, etc. This is the core of PMU.
- **One RTC timer**
- **10 always-on registers (LP_AON_STORE0_REG ~ LP_AON_STORE9_REG)**: These registers are always powered up and are not affected by any low-power modes, thus can be used for storing data that cannot be lost.
- **Eight LP GPIO pins (GPIO0 ~ GPIO7)**: 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 about the LP GPIOs, please refer to Chapter *7 GPIO Matrix and IO MUX*.
- **320 KB HP SRAM**: The 320 KB SRAM is accessible to CPU. It works under the CPU clock.
- **Brownout detector**: It monitors the power of the supply voltage pins, ensuring stable chip operation and preventing the SoC from potential malfunction if subject to glitches or under voltage.

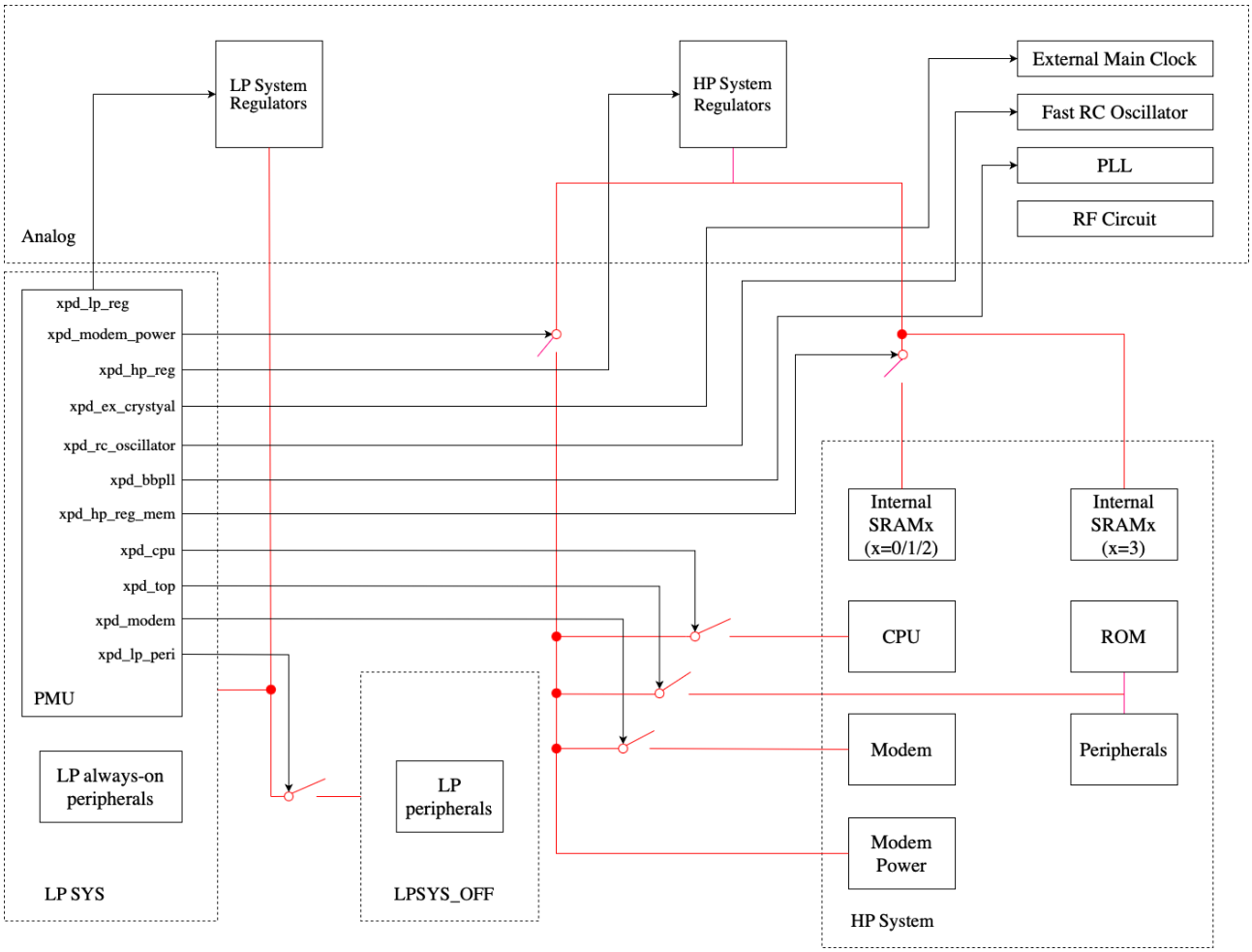
The following sections provide a detailed description of the components mentioned above.

### 12.4.1 Power Scheme

Figure 12.4-1 shows the power scheme of ESP32-C61 that mainly includes:

- Two sets of regulators

- Analog power domains
- Digital power domains




→: Control signals --: Power lines

**Figure 12.4-1. ESP32-C61 Power Scheme**

### 12.4.1.1 Regulators

As shown in Figure 12.4-1, the analog part of ESP32-C61 contains two sets of regulators that regulate the power supply to different power domains. The two regulators are:

- One set of HP system regulators:
  - one main HP regulator
  - two secondary HP regulators powering the HP memory and HP logic circuits respectively.

  They are used for regulating the power supply to high-performance modules and feature high drive strength, high power consumption, and regulated output power.
- One set of LP system regulators:
  - one main LP regulator
  - one secondary LP regulator

They are used for regulating the power supply to low-power modules, and also feature regulated output power.

#### 12.4.1.2 Digital Power Domains

ESP32-C61 has digital power domains in the HP system and LP system as listed below.

- The HP system contains the following digital power domains:
  - **CPU**: includes the CPU and its supporting peripherals (such as TRACE) in the HP system.
  - **Modem**: consists of wireless MAC, baseband and respective peripheral controllers.
  - **Internal SRAMx**: As shown in Figure 12.4-1, each of SRAM0/1/2 has an independent power switch, while SRAM3 is directly connected to the regulator without a power switch.
  - **Modem Power**: includes modules that control the operating of the wireless section.
  - **Peripherals+ROM**: includes ROM, HP peripherals, and all other modules in the HP system that are not mentioned above.
- The LP system contains the following digital power domains:
  - **LPSYS_OFF**: includes LP peripherals APB bus, and lpperi clk.
  - **LPSYS**: includes all modules in the LP system except for LPSYS_OFF, such as LP always-on peripherals (e.g., RTC timer) and PMU controller. This power domain has the highest priority among all power domains and remains powered on unless the chip is completely powered down.

Power domains in the HP system and LP system are powered by HP system regulators and LP system regulators respectively. There is an independent power switch between the regulators and each power domain, enabling up/down control of the digital power domain. With the exception of LPSYS — which remains always-on and thus does not require power gating — power control for the digital power domains can be achieved.

#### 12.4.1.3 Analog Power Domains

As Figure 12.4-1 shows, ESP32-C61 contains the following analog power domains, among which PLL belongs to the HP system while the other domains belong to the LP system:

- External Main Clock
- Fast RC Oscillator
- PLL
- RF circuit

### 12.4.2 PMU

The PMU of ESP32-C61 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**: records and switches the PMU states.
- **Sleep/wake-up controller**: sends sleep or wake-up requests to the PMU main state machine.
- **Power controllers**: control the power and clock signals depending on the power modes of the chip. The power controllers include:

- **Digital power controller**: powers up or down the digital power domains.
- **Analog power controller**: enables the analog modules, such as the regulators, analog clocks, etc.
- **Clock controller**: manages the clock gating of peripherals and selects analog clock sources for digital clocks.
- **Data backup controller**: controls the data backup and restore process when the chip switches between PMU states.
- **System controller**: controls some system-level modules, such as suspending watchdog functionality or handling backups in sleep mode (when the CPU in the HP system is unavailable).

The PMU workflow involves the sleep/wake-up controller sending sleep or wake-up requests to the PMU main state machine, which then generates power gating, clock gating, and reset signals. The power controllers and clock controller will then 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 12.4-2.

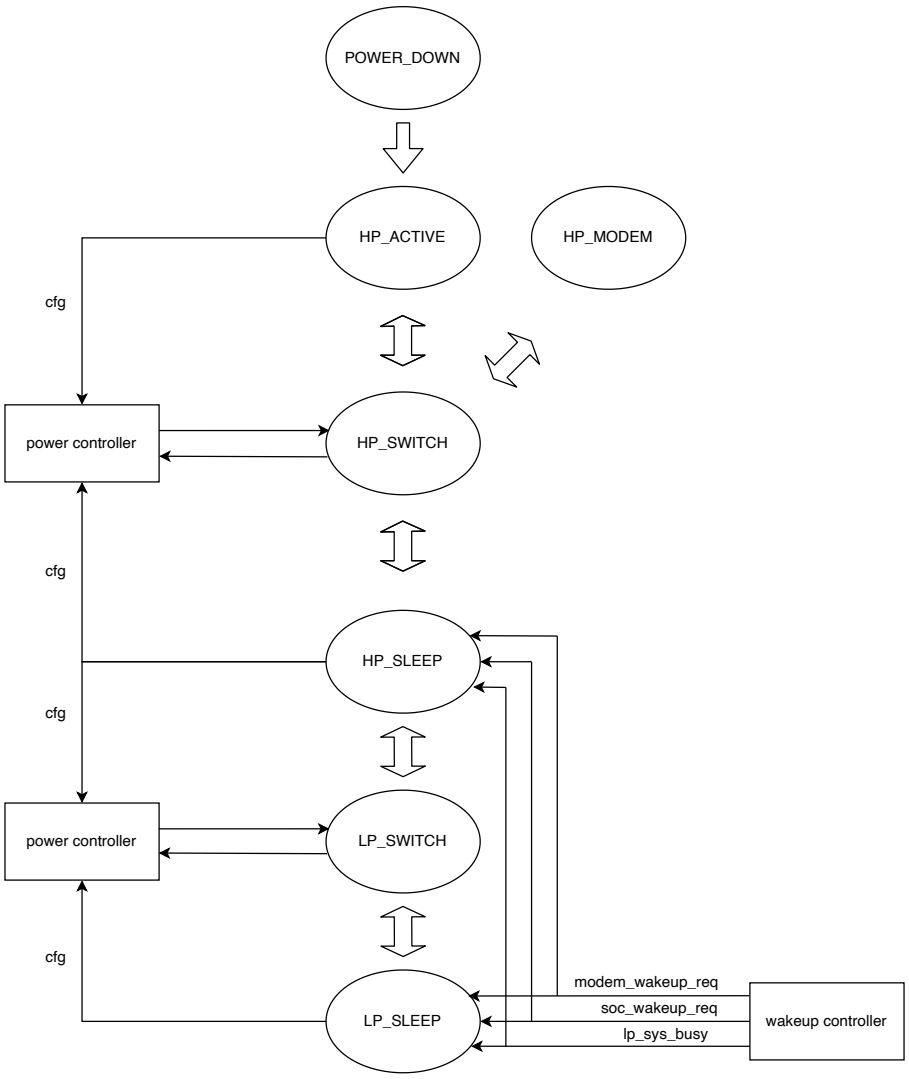



Figure 12.4-2. PMU Workflow

The following sections describe the main parts of PMU.

#### 12.4.2.1 PMU Main State Machine

The PMU main state machine can receive sleep and wake-up signals, change 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 four PMU states and two transition states.

Each PMU state is controlled by different sleep and wake-up signals, supporting software customization of power and clocks. These four PMU states allow the software to expand power modes for various application scenarios. The four PMU states are:

- HP_ACTIVE: PMU state where the circuits on the chip are powered up to a maximum, supporting the HP system and LP system operation.
- HP_MODEM: PMU state where Modem (wireless MAC and baseband) can operate independently of the CPU.
- HP_SLEEP: PMU state where the HP system is in sleep, supporting the operation of LP system.
- LP_SLEEP: PMU state where HP system and LP peripherals are in sleep, while the always-on circuits remain operational.

Two transition states indicate that the PMU is waiting for the system to switch to a different power mode. For example, during the period of "waiting for the configured clock gating to take effect and the output clock to stabilize."

- HP_SWITCH: An intermediate state when the HP system switches between HP_ACTIVE, HP_MODEM, and HP_SLEEP. The hardware completes the control transition of various controllers in this state.
- LP_SWITCH: An intermediate state when the LP system switches between HP_SLEEP and LP_SLEEP. The hardware completes the control transition of various controllers in this state.

**Note:**
Referring to *ESP32-C61 Datasheet* > *Functional Block Diagram* The division of HP and LP system is as follows:

- LP system (peripherals): modules that are marked as "optional in Deep-sleep" in the Functional Block Diagram, such as LP IO.
- LP system (always-on circuits): modules that are marked as "All modes" and "optional in Deep-sleep" in the Functional Block Diagram, such as PMU, RTC Watchdog Timer, Super Watchdog.
- HP system: modules that are marked as "Active and Modem-sleep" in the Functional Block Diagram.

HP_ACTIVE, HP_MODEM, HP_SLEEP are the states of the HP system, while LP_SLEEP is the state of the LP system.

- If a module belongs to the HP system, its power up/down can be configured in the HP_ACTIVE, HP_MODEM and HP_SLEEP states, but not in the LP_SLEEP state. In the LP_SLEEP state, it will reuse the HP_SLEEP state.
- If a module belongs to the LP system, its power up/down can be configured in the HP_SLEEP and LP_SLEEP states. In the HP_ACTIVE and HP_MODEM states, it will reuse the HP_SLEEP configuration.

Take the CPU as an example: CPU belongs to the HP system, so it can be configured to power up/down in the

HP_ACTIVE, HP_MODEM and HP_SLEEP states through the following registers and reuses the HP_SLEEP configuration in LP_SLEEP.

- HP_ACTIVE: PMU_HP_ACTIVE_PD_HP_CPU_PD_EN
- HP_MODEM: PMU_HP_MODEM_PD_HP_CPU_PD_EN
- HP_SLEEP: PMU_HP_SLEEP_PD_HP_CPU_PD_EN

Similarly, users can define other power domains' power up and down in different PMU states. For specific registers, please refer to Section 12.8.

**Note:**
In the following text, all such registers will be collectively referred to as PMU_*PMUSTATE*_PD_*POWERDOMAIN*_PD_EN, where, unless specified otherwise:

- *PMUSTATE* represents HP_ACTIVE, HP_MODEM, HP_SLEEP and LP_SLEEP.
- *POWERDOMAIN* represents TOP, HP_AON, HP_CPU, HP_WIFI, HP_MEM and LP_PERI.

All such registers are linked to a representative of the same type for reference. For example, PMU_*PMUSTATE*_PD_*POWERDOMAIN*_PD_EN will be linked to PMU_*HP_ACTIVE*_PD_*TOP*_PD_EN.

Once the configuration is done, PMU will use various controllers to make these configurations effective, as described in the sections below.

#### 12.4.2.2 Sleep/Wake-up Controller

The sleep/wake-up controller is responsible for initiating sleep and wake-up requests to the PMU main state machine. ESP32-C61 supports multiple wake-up sources to wake the CPU from different power modes that can be enabled through PMU_WAKEUP_ENA.

Table 12.4-1. Wake-up Sources

| PMU_WAKEUP_ENA | Wake-up Sources | Light-sleep | Deep-sleep |
|---|---|---|---|
| 0x4 | GPIO[1] | Y | Y |
| 0x8 | Wi-Fi beacon | Y | N |
| 0x10 | RTC Timer | Y | Y |
| 0x20 | Wi-Fi[2] | Y | – |
| 0x40 | UART0[3] | Y | – |
| 0x80 | UART1[3] | Y | – |
| 0x100 | SDIO | Y | – |
| 0x400 | Bluetooth | Y | – |

[1] In Deep-sleep mode, only the LP GPIOs can work as wake-up sources.

[2] To wake up the chip with Wi-Fi, the chip switches between Active, Modem-sleep, and Light-sleep modes. The RF module is woken up at predetermined intervals by timers to keep Wi-Fi connectivity and communication.

[3] A wake-up is triggered when the number of RX pulses received exceeds the threshold set in UART_ACTIVE_THRESHOLD. For details, please refer to Chapter 26 *UART Controller (UART)*.

ESP32-C61 provides a hardware mechanism that can reject sleep, meaning if some peripherals are in an uninterruptible working state and the CPU tries to sleep, the peripherals will send a wake-up signal to prevent the CPU from sleeping, thus ensuring the peripherals work normally.

The wake-up sources in Table 12.4-1 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_REJECT_CAUSE and the corresponding wake-up sources are the same as shown in Table 12.4-1.

- Enable sleep rejection feature:
  - Set PMU_SLP_REJECT_EN to 1 to enable the sleep rejection feature.
  - Configure PMU_SLEEP_REJECT_ENA to enable the sleep rejection signal source.
- Read PMU_REJECT_CAUSE for the source of sleep rejection event.

#### 12.4.2.3 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 PMU states.

The configuration of the regulators is as follows:

- Configure (PMU_*PMUSTATE*_HP_REGULATOR_XPD) or (PMU_*PMUSTATE*_LP_REGULATOR_XPD) to enable or disable the output voltage of the HP/LP regulators in the target PMU state. Turning off the LP regulators 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_CLK) is as follows:

- XTAL_CLK: Configure PMU_HP_SLEEP_XPD_XTAL to 1 to enable XTAL_CLK when the chip switches 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 afterPMU_WAIT_XTL_STABLE cycles in CLK_DYN_FAST_CLK following the power-up of XTL_CLK.

- PLL_CLK: PMU can enable PLL_CLK in different PMU states by configuring PMU_*PMUSTATE*_XPD_BBPLL to 1. For example, configuring PMU_HP_ACTIVE_XPD_BBPLL to 1 will enable the PLL_CLK clock when the chip is in HP_ACTIVE state.

  Note:

  - Before enabling PLL_CLK, please ensure that XTAL_CLK is stable.
  - To avoid the instability in PLL_CLK during startup, users have the option to configure PMU_WAIT_PLL_STABLE to delay the gate opening for PLL_CLK. This delay ensures that the gate opening is enabled after PMU_WAIT_PLL_STABLE cycles in CLK_DYN_FAST_CLK following the power-up of PLL_CLK.

Slow-speed clocks include:

- RC_FAST_CLK
- XTAL32K_CLK
- RC32K_CLK
- SOSC_CLK

These slow-speed clocks operate with low power. The power up and down of these clocks in different PMU states is configured via different registers. Take RC_FAST_CLK as an example:

- HP_ACTIVE, HP_MODEM and HP_SLEEP: PMU_HP_SLEEP_XPD_FOSC_CLK
- LP_SLEEP: PMU_LP_SLEEP_XPD_FOSC_CLK

#### 12.4.2.4 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.

Among the digital power domains, the LPSYS_OFF domain can only be powered up and down during the PMU state switch between HP_SLEEP and LP_SLEEP PMU, while the other power domains can be powered up and down during the PMU states switch among HP_SLEEP, HP_ACTIVE, and HP_MODEM.

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_MODEM_PD_HP_CPU_PD_EN:

- 0: indicates that the CPU power domain is powered up in the HP_MODEM state.
- 1: indicates that the CPU power domain is powered down in the HP_MODEM state.

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 will explain the power up and down of each digital power domain:

- Internal SRAMx

  The Internal SRAMx power domain and "ROM + Peripherals" power domain share the same control register PMU_*PMUSTATE*_PD_TOP_PD_EN. The PMU_PD_HP_MEM*x*_PD_MASK (*x*=0,1,2) configures whether Internal SRAMx will power on/off simultaneously with the HPTOP power domain:

  - 0: powers on/off simultaneously with the "ROM + Peripherals" power domain.
  - 1: does not power on/off simultaneously with the "ROM + Peripherals" power domain. In this case, users can configure PMU_PD_HP_MEM*x*_PD_MASK (*x*=0,1,2) to keep Internal SRAMx powered on even when the "ROM + Peripherals" power domain is powered off.

- Modem Power

  Modem Power is connected to the HP system regulator through a power switch. The power up and down of Modem Power in different PMU state is determined by PMU_*PMUSTATE*_PD_HP_AON_PD_EN (*PMUSTATE*=HP_ACTIVE/HP_MODEM/HP_SLEEP). From Figure 12.4-1, it can be seen that if any of the CPU, Modem, or "ROM + Peripherals" power domains needs to be powered up, the Modem Power domain must also be powered up. Such a design meets functional requirements.

- "ROM + Peripherals", Modem and CPU

  Each of the three power domains can be powered up and down in different PMU states using register PMU_*POWERDOMAIN*_PD_*PMUSTATE*_PD_EN(*POWERDOMAIN*=TOP/HP_WIFI/HP_CPU) respectively.

  When the "ROM + Peripherals" domain is powered down, the following features are configurable:

  - Powering down the "ROM + Peripherals" domain may cause instability in GPIOs. This can be addressed by maintaining the state of the GPIOs (excluding eight LP GPIOs) through PMU. For example, configuring PMU_HP_SLEEP_HP_PAD_HOLD_ALL can keep GPIOs in the same state as before "ROM + Peripherals" was powered down in HP_SLEEP.
  - In HP_ACTIVE, HP_SLEEP and HP_MODEM states, the Internal SRAMx domain can be powered down or put into Deep-sleep mode. In Deep-sleep, the memory cannot be read or written to, but data can be retained. Configure PMU_*PMUSTATE*_HP_MEM_DSLP to enable Memory Deep-sleep mode.

- LPSYS_OFF

  The LPSYS_OFF power domain remains powered up when the chip is in HP_ACTIVE and HP_MODEM states. The power state of LPSYS_OFF is configurable only in HP_SLEEP and LP_SLEEP states via PMU_LP_SLEEP_LP_DIG_POWER_REG and PMU_HP_SLEEP_LP_DIG_POWER_REG respectively. The LPSYS_OFF power domain has an independent digital power switch, and its power control is not dependent on the power state of other digital power domains.

- LP always-on (LPSYS)

  LPSYS includes modules such as the PMU that will not power down unless the chip is completely powered off. Therefore, the power on/off state of this power domain cannot be configured.

#### 12.4.2.5 Clock Controller

The clock controller is mainly used to select, enable and configure high-performance system clocks and low-power system clocks when the chip switches between PMU states.

High-performance system clocks include HP_ROOT_CLK and high-performance system peripherals clocks. When the chip switches among HP_ACTIVE, HP_MODEM, and HP_SLEEP states, PMU can switch, power up/down, and divide the frequency of HP_ROOT_CLK, as well as power up/down high-performance system peripherals clocks.

- HP_ROOT_CLK can be controlled as follows:
  - Configure PMU_*PMUSTATE*_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_*PMUSTATE*_ICG_SYS_CLOCK_EN to 0 to disable HP_ROOT_CLK in the corresponding PMU state.
  - Configure PMU_*PMUSTATE*_DIG_SYS_CLK_SEL to select the clock source after the chip enters the corresponding PMU state. For details, please refer to Chapter 8 *Reset and Clock* > Table 8.2-1.
- High-performance system peripheral clocks can be controlled as follows:
  - Configure PMU_*PMUSTATE*_ICG_SLP_SEL to 1 so that the clock gating in the target state will be controlled by PMU. Configure this register to 0 so that the clock gating is controlled by PCR registers.
  - Configure PMU_*PMUSTATE*_DIG_ICG_FUNC_EN to power up/down the function clock in the target PMU state. For detailed configuration please see Chapter 8 *Reset and Clock* > Table 8.2-8.
  - Configure PMU_*PMUSTATE*_DIG_ICG_APB_EN to power up/down the APB clock in the target PMU state. For detailed configuration please see Chapter 8 *Reset and Clock* > Table 8.2-7.

LP system clocks are mainly used in the low-power system and include the following four clocks:

- LP_SLOW_CLK
- LP_FAST_CLK
- LP_DYN_SLOW_CLK
- LP_DYN_FAST_CLK

The clock frequencies of LP_DYN_FAST_CLK and LP_DYN_SLOW_CLK are controlled by hardware as follows, depending on the PMU state (and cannot be changed by the user):

- LP_SLEEP: The frequency of LP_DYN_FAST_CLK and LP_DYN_SLOW_CLK is the same as LP_SLOW_CLK.
- HP_ACTIVE, HP_MODEM, HP_SLEEP:
  - The LP_DYN_FAST_CLK frequency is the same as LP_FAST_CLK.

- LP_DYN_SLOW_CLK is clk_root_slow_gated, which is generated from LP_DYN_FAST_CLK through a clock gate. The enable signal of the clock gate is controlled by the rising edge of LP_DYN_SLOW_CLK after being synchronized through a double flip-flop.

#### 12.4.2.6 Backup Controller

ESP32-C61 has a Retention DMA module, which is in the "ROM + Peripherals" power domain, that can transfer data between memory and peripherals when the chip switches between PMU states, so that 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 via REGDMA. 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.

Take the HP_SLEEP target PMU state as an example, the control registers for transitioning from HP_ACTIVE to HP_SLEEP and from HP_MODEM to HP_SLEEP are different. The possible PMU state switches are listed below, collectively represented by *PMUSTATE_TRANS* in the register names:

- HP_SLEEP2ACTIVE
- HP_SLEEP2MODEM
- HP_MODEM2ACTIVE
- HP_MODEM2SLEEP
- HP_ACTIVE2SLEEP

All such registers are linked to a representative of the same type for reference. For example, PMU_*PMUSTATE_TRANS*_BACKUP_EN will be linked to PMU_HP_SLEEP2ACTIVE_BACKUP_EN.

The following will introduce how PMU controls the Retention DMA:

- Enable data transfer: Configure PMU_*PMUSTATE_TRANS*_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_*PMUSTATE_TRANS*_BACKUP_CLK_SEL to select the clock source of the Retention DMA, and configure PMU_*PMUSTATE*_BACKUP_ICG_FUNC_EN to enable the clock.

  After the data transfer is completed, the value of PMU_*PMUSTATE*_BACKUP_ICG_FUNC_EN is determined by the configuration of PMU_*PMUSTATE*_DIG_ICG_FUNC_EN in the target PMU state.
- Configure data transfer direction: Configure the highest bit of PMU_*PMUSTATE_TRANS*_BACKUP_MODE:
  - 1: From peripheral to memory
  - 0: From memory to peripheral
- Select linked list pointer: Configure the lower two bits of PMU_*PMUSTATE_TRANS*_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

#### 12.4.2.7 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_*PMUSTATE*_DIG_PAUSE_WDT to 1 can disable 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_*PMUSTATE*_HP_PAD_HOLD_ALL to 1 can latch the current configuration of the GPIO pin as the sleep configuration when the chip transitions to the corresponding PMU state. For more information about GPIO's hold function, please refer to Chapter 7 *GPIO Matrix and IO MUX* > Section 7.9.
- Disabling UART wake-up function: Configuring PMU_*PMUSTATE*_UART_WAKEUP_EN to 0 can disable the four UART wake-up modes in the corresponding PMU state. For more information on UART wake-up modes, please refer to Chapter 26 *UART Controller (UART)*.
- Pausing CPU: Setting PMU_*PMUSTATE*_DIG_CPU_STALL to 1 can suspend the CPU in the corresponding PMU state.

### 12.4.3 RTC Timer

ESP32-C61's low-power management system features an RTC timer. The 48-bit RTC timer is a real-time counter that logs time when certain events occur, working at LP_DYN_FAST_CLK. For the trigger conditions for the RTC timer, see Table 12.4-2.

**Table 12.4-2. Trigger Conditions for the RTC Timer**

| Triggering Conditions | Description |
|---|---|
| RTC_TIMER_MAIN_TIMER_XTAL_OFF | Triggered when PMU powers up or down the 40 MHz crystal. |
| RTC_TIMER_MAIN_TIMER_SYS_STALL | Triggered when the CPU enters or exits the stall state. This is to ensure the system timer is continuous in time. |
| RTC_TIMER_MAIN_TIMER_SYS_RST | Triggered upon system reset. |
| RTC_TIMER_MAIN_TIMER_UPDATE | Triggered when RTC_TIMER_MAIN_TIMER_UPDATE is configured by the CPU (e.g., users). |

The RTC timer updates two groups of registers upon any new trigger.

- Register group 0 records the count value of the RTC timer under the current trigger, with the counting unit being LP_DYN_SLOW_CLK.
  - RTC_TIMER_MAIN_TIMER_TAR_HIGH0
  - RTC_TIMER_MAIN_TIMER_TAR_LOW0

- Register group 1 records the count value of the RTC timer under the previous trigger, with the counting unit being LP_DYN_SLOW_CLK.
  - RTC_TIMER_MAIN_TIMER_TAR_HIGH1
  - RTC_TIMER_MAIN_TIMER_TAR_LOW1

Each time there is a new trigger, the record from the previous trigger will be moved from register group 0 to register group 1 (the record in register group 1 will be overwritten), and the record of the current trigger will be stored in register group 0. Therefore, the RTC timer can record up to two trigger values simultaneously.

It is worth noting that any reset or sleep state other than the chip's power-up reset will not stop or reset the RTC timer, which is in the LPSYS power domain. Additionally, the RTC timer can also be used as a wake-up source (see Table 12.4-1).

## 12.5 Power Modes

ESP32-C61 has four configurable PMU states. Based on the four PMU states, five power modes have been defined for the most commonly seen application scenarios. For the details, please see Table 12.5-1.

**Table 12.5-1. Preset Power Modes**

| Power Modes | Power Domain | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | LPSYS | LPSYS_OFF | ROM + Peripherals | Modem | CPU | RC_FAST_CLK | XTAL_CLK | PLL | RF circuit |
| Active | ON | ON | ON | ON | ON | ON | ON | ON | ON |
| Modem-sleep | ON | ON | ON | ON | OFF | ON | ON | ON/OFF | OFF |
| Light-sleep0 | ON | ON | ON | ON/OFF | OFF | ON | ON/OFF | ON/OFF | ON/OFF |
| Light-sleep1 | ON | ON/OFF | ON/OFF | ON/OFF | OFF | ON/OFF | ON/OFF | ON/OFF | ON/OFF |
| Deep-sleep | ON | OFF | OFF | OFF | OFF | OFF | OFF | OFF | OFF |

**Note:**

1. *ESP32-C61 Datasheet* > Section *Current Consumption*.
2. For supported wake-up sources, please refer to Table 12.4-1.

## 12.6 Event Task Matrix Feature

The low-power management system on ESP32-C61 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 11 Event Task Matrix (ETM).

The low-power management system can receive the following ETM tasks:

- PMU_ETM_SLEEP_REQ: Triggers PMU's sleep process.

The low-power management system can generate the following ETM events:

- PMU_ETM_SLEEP_WAKEUP: Indicates that PMU is woken up to the HP_ACTIVE state.

## 12.7 Interrupts

ESP32-C61's low-power management system can generate the following interrupt signals:

- PMU_HP_INTR
- PMU_LP_INTR

Among these interrupt signals, PMU_HP_INTR is sent to the Interrupt Matrix.

The interrupt signals are generated by the internal interrupt sources of each module, specifically:

PMU_HP_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_SDIO_IDLE_INT: Triggered when SDIO is in idle state.

PMU_LP_INT:

- PMU_ACTIVE_SWITCH_SLEEP_START_INT: Triggered when the PMU state starts to switch from HP_ACTIVE to HP_SLEEP.
- PMU_MODEM_SWITCH_SLEEP_START_INT: Triggered when the PMU state starts to switch from HP_MODEM to HP_SLEEP.
- PMU_SLEEP_SWITCH_MODEM_START_INT: Triggered when the PMU state starts to switch from HP_SLEEP to HP_MODEM.
- PMU_SLEEP_SWITCH_ACTIVE_START_INT: Triggered when the PMU state starts to switch from HP_SLEEP to HP_ACTIVE.
- PMU_MODEM_SWITCH_ACTIVE_START_INT: Triggered when the PMU state starts to switch from HP_MODEM to HP_ACTIVE.
- PMU_ACTIVE_SWITCH_SLEEP_END_INT: Triggered when the PMU state has switched from HP_ACTIVE to HP_SLEEP.
- PMU_MODEM_SWITCH_SLEEP_END_INT: Triggered when the PMU state has switched from HP_MODEM to HP_SLEEP.
- PMU_SLEEP_SWITCH_MODEM_END_INT: Triggered when the PMU state has switched from HP_SLEEP to HP_MODEM.
- PMU_SLEEP_SWITCH_ACTIVE_END_INT: Triggered when the PMU state has switched from HP_SLEEP to HP_ACTIVE.
- PMU_MODEM_SWITCH_ACTIVE_END_INT: Triggered when the PMU state has switched from HP_MODEM to HP_ACTIVE.

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 12.8 *Register Summary*.

## 12.8 Register Summary

The abbreviations given in Column **Access** are explained in Section *Access Types for Registers*.

### 12.8.1 PMU Register Summary

The addresses in this section are relative to the PMU base address provided in Table 5.3-2 in Chapter 5 *System and Memory*.

| Name | Description | Address | Access |
|---|---|---|---|
| PMU_HP_ACTIVE_DIG_POWER_REG | Configuration register of digital power domains in HP_ACTIVE state | 0x0000 | R/W |
| PMU_HP_ACTIVE_ICG_HP_FUNC_REG | Control register of HP system peripherals' function clocks in HP_ACTIVE state | 0x0004 | R/W |
| PMU_HP_ACTIVE_ICG_HP_APB_REG | Control register of HP system peripherals' APB clocks in HP_ACTIVE state | 0x0008 | R/W |
| PMU_HP_ACTIVE_ICG_MODEM_REG | Control register of HP system Modem clock gating in HP_ACTIVE state | 0x000C | 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 | Control register of clock source's power in HP_ACTIVE state | 0x0014 | R/W |
| PMU_HP_ACTIVE_BIAS_REG | Control register for the operating state of analog circuits (BIAS, DBG, CUR circuits) in HP_ACTIVE state | 0x0018 | R/W |
| PMU_HP_ACTIVE_BACKUP_REG | Data backup control register in HP_ACTIVE state | 0x001C | R/W |
| PMU_HP_ACTIVE_BACKUP_CLK_REG | Control register for the operating clocks of the backup module in HP_ACTIVE state | 0x0020 | R/W |
| PMU_HP_ACTIVE_SYSCLK_REG | Control register of the system clocks in HP_ACTIVE state | 0x0024 | R/W |
| PMU_HP_ACTIVE_HP_REGULATOR0_REG | Control register of HP regulators' power in HP_ACTIVE state, controlling all settings except drive | 0x0028 | varies |
| PMU_HP_ACTIVE_HP_REGULATOR1_REG | Control register of HP regulators' power in HP_ACTIVE state, controlling drive settings | 0x002C | R/W |
| PMU_HP_ACTIVE_XTAL_REG | XTAL_CLK power control register in HP_ACTIVE state | 0x0030 | R/W |
| PMU_HP_MODEM_DIG_POWER_REG | Configuration register of digital power domains in HP_MODEM state | 0x0034 | R/W |
| PMU_HP_MODEM_ICG_HP_FUNC_REG | Control register of HP system peripherals' function clocks in HP_MODEM state | 0x0038 | R/W |

| Name | Description | Address | Access |
|---|---|---|---|
| PMU_HP_MODEM_ICG_HP_APB_REG | Control register of HP system peripherals' APB clocks in HP_MODEM state | 0x003C | R/W |
| PMU_HP_MODEM_ICG_MODEM_REG | Control register of HP system Modem clock gating in HP_MODEM state | 0x0040 | R/W |
| PMU_HP_MODEM_HP_SYS_CNTL_REG | System control register in HP_MODEM state | 0x0044 | R/W |
| PMU_HP_MODEM_HP_CK_POWER_REG | Control register of clock source's power in HP_MODEM state | 0x0048 | R/W |
| PMU_HP_MODEM_BIAS_REG | Control register for the operating state of analog circuits (BIAS, DBG, CUR circuits) in HP_MODEM state | 0x004C | R/W |
| PMU_HP_MODEM_BACKUP_REG | Data backup control register in HP_MODEM state | 0x0050 | R/W |
| PMU_HP_MODEM_BACKUP_CLK_REG | Control register for the operating clocks of the backup module in HP_MODEM state | 0x0054 | R/W |
| PMU_HP_MODEM_SYSCLK_REG | Control register of the system clocks in HP_MODEM state | 0x0058 | R/W |
| PMU_HP_MODEM_HP_REGULATOR0_REG | Control register of HP regulators' power in HP_MODEM state, controlling all settings except drive | 0x005C | R/W |
| PMU_HP_MODEM_HP_REGULATOR1_REG | Control register of HP regulators' power in HP_MODEM state, controlling drive settings | 0x0060 | R/W |
| PMU_HP_MODEM_XTAL_REG | XTAL_CLK power control register in HP_MODEM state | 0x0064 | R/W |
| PMU_HP_SLEEP_DIG_POWER_REG | Configuration register of digital power domains in HP_SLEEP state | 0x0068 | R/W |
| PMU_HP_SLEEP_ICG_HP_FUNC_REG | Control register of HP system peripherals' function clocks in HP_SLEEP state | 0x006C | R/W |
| PMU_HP_SLEEP_ICG_HP_APB_REG | Control register of HP system peripherals' APB clocks in HP_SLEEP state | 0x0070 | R/W |
| PMU_HP_SLEEP_ICG_MODEM_REG | Control register of HP system Modem clock gating in HP_SLEEP state | 0x0074 | 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 | Control register of clock source's power in HP_SLEEP state | 0x007C | R/W |
| PMU_HP_SLEEP_BIAS_REG | Control register for operation state of analog circuits (BIAS, DBG, CUR) in HP_SLEEP | 0x0080 | R/W |
| PMU_HP_SLEEP_BACKUP_REG | Data backup control register in HP_SLEEP state | 0x0084 | R/W |

| Name | Description | Address | Access |
|---|---|---|---|
| PMU_HP_SLEEP_BACKUP_CLK_REG | Control register for the operating clocks of the backup module in HP_SLEEP state | 0x0088 | R/W |
| PMU_HP_SLEEP_SYSCLK_REG | Control register of the system clocks in HP_SLEEP state | 0x008C | R/W |
| PMU_HP_SLEEP_HP_REGULATOR0_REG | Control register of HP regulators' power in HP_SLEEP state, controlling all settings except drive | 0x0090 | R/W |
| PMU_HP_SLEEP_HP_REGULATOR1_REG | Control register of HP regulators' power in HP_SLEEP state, controlling drive settings | 0x0094 | R/W |
| PMU_HP_SLEEP_XTAL_REG | XTAL_CLK power control register in HP_SLEEP state | 0x0098 | R/W |
| PMU_HP_SLEEP_LP_REGULATOR0_REG | LP regulator power control register in HP_SLEEP state, controlling all settings except drive | 0x009C | R/W |
| PMU_HP_SLEEP_LP_REGULATOR1_REG | LP regulator power control register in HP_SLEEP state, controlling drive settings | 0x00A0 | R/W |
| PMU_HP_SLEEP_LP_DIG_POWER_REG | Configuration register of digital power domains in LP system in HP_SLEEP state | 0x00A8 | R/W |
| PMU_HP_SLEEP_LP_CK_POWER_REG | Low-speed clock power control register in HP_ACTIVE, HP_MODEM and HP_SLEEP states | 0x00AC | R/W |
| PMU_LP_SLEEP_LP_REGULATOR0_REG | LP regulator power control register in LP_SLEEP state, controlling all settings except drive | 0x00B4 | R/W |
| PMU_LP_SLEEP_LP_REGULATOR1_REG | LP regulator power control register in LP_SLEEP state, controlling drive settings | 0x00B8 | 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 | Configuration register of digital power domains in LP system 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_LP_SLEEP_BIAS_REG | Control register for operation state of analog circuits (BIAS, DBG, CUR) in LP_SLEEP | 0x00C8 | R/W |
| PMU_IMM_HP_CK_POWER_REG | Software-controlled enable signal for HP clock power on/off | 0x00CC | WT |

| Name | Description | Address | Access |
|---|---|---|---|
| PMU_IMM_SLEEP_SYSCLK_REG | Software-controlled enable signal for system root clk switching during sleep | 0x00D0 | WT |
| PMU_IMM_HP_FUNC_ICG_REG | Software-controlled configuration for functional clock gating of PMU's forced update output | 0x00D4 | WT |
| PMU_IMM_HP_APB_ICG_REG | Software-controlled configuration for APB clock gating of PMU's forced update output | 0x00D8 | WT |
| PMU_IMM_MODEM_ICG_REG | Software-controlled configuration for HP system Modem clock gating of PMU's forced update output | 0x00DC | WT |
| PMU_IMM_PAD_HOLD_ALL_REG | Software-controlled signal to tie PMU's HP/LP GPIOs HOLD high or low | 0x00E4 | WT |
| PMU_IMM_I2C_ISO_REG | Software-controlled signal to tie PMU's i2c iso enable high or low | 0x00E8 | WT |
| PMU_POWER_WAIT_TIMER0_REG | Software-controlled power-on/off waiting timer for PMU | 0x00EC | R/W |
| PMU_POWER_WAIT_TIMER1_REG | Software-controlled power-on/off waiting timer for LP PERI controlled by PMU | 0x00F0 | R/W |
| PMU_POWER_WAIT_TIMER2_REG | Software-controlled waiting timer for HP, LP ISO and reset in PMU | 0x00F4 | R/W |
| PMU_POWER_PD_TOP_CNTL_REG | Configuration register for "Peripherals+ROM" power domain controlled by PMU | 0x00F8 | R/W |
| PMU_POWER_PD_HPAON_CNTL_REG | Configuration register for Modem Power power domain controlled by PMU | 0x00FC | R/W |
| PMU_POWER_PD_HPCPU_CNTL_REG | Configuration register for CPU power domain controlled by PMU | 0x0100 | R/W |
| PMU_POWER_PD_HPWIFI_CNTL_REG | Configuration register for MODEM power domain controlled by PMU | 0x0108 | R/W |
| PMU_POWER_PD_LPPERI_CNTL_REG | Configuration register for LPSYS_OFF power domain controlled by PMU | 0x010C | R/W |
| PMU_POWER_PD_MEM_CNTL_REG | Configuration register for Internal SRAMx power domain's ISO and overall switch controlled by PMU | 0x0110 | R/W |
| PMU_POWER_PD_MEM_MASK_REG | Configuration register for LDO switch of individual Internal SRAMx power supply controlled by PMU | 0x0114 | R/W |
| PMU_POWER_HP_PAD_REG | Configuration register for HP GPIOs power supply controlled by PMU | 0x0118 | R/W |
| PMU_POWER_VDD_SPI_CNTL_REG | Configuration register for VDD_SPI power supply controlled by PMU | 0x011C | R/W |

| Name | Description | Address | Access |
|---|---|---|---|
| PMU_POWER_CK_WAIT_CNTL_REG | Configuration register for system waiting time after clock change controlled by PMU | 0x0120 | R/W |
| PMU_SLP_WAKEUP_CNTL0_REG | Sleep request register | 0x0124 | WT |
| PMU_SLP_WAKEUP_CNTL1_REG | Sleep reject register | 0x0128 | R/W |
| PMU_SLP_WAKEUP_CNTL2_REG | Wakeup source enable register | 0x012C | R/W |
| PMU_SLP_WAKEUP_CNTL3_REG | PMU state selection signal and waiting time register for sleep-Wakeup | 0x0130 | R/W |
| PMU_SLP_WAKEUP_CNTL4_REG | Sleep reject event clear register | 0x0134 | WT |
| PMU_SLP_WAKEUP_CNTL5_REG | Waiting time register after Wakeup in LP_SLEEP and HP_SLEEP | 0x0138 | R/W |
| PMU_SLP_WAKEUP_CNTL6_REG | Waiting time register after Wakeup in HP_MODEM phase | 0x013C | R/W |
| PMU_SLP_WAKEUP_CNTL7_REG | Waiting time register after Wakeup in HP_SWITCH phase | 0x0140 | R/W |
| PMU_SLP_WAKEUP_STATUS0_REG | Wakeup event register | 0x0144 | RO |
| PMU_SLP_WAKEUP_STATUS1_REG | Reset reject event register | 0x0148 | RO |
| PMU_HP_CK_CNTL_REG | PMU-set waiting time after ICG configuration adjustment | 0x0150 | R/W |
| PMU_RF_PWC_REG | Configuration for i2c and rf circuits controlled by PMU | 0x0158 | R/W |
| PMU_BACKUP_CFG_REG | Configuration for backup clock frequency division controlled by PMU | 0x015C | R/W |
| PMU_INT_RAW_REG | PMU sleep/Wakeup raw interrupt | 0x0160 | R/WTC/SS |
| PMU_HP_INT_ST_REG | PMU sleep/Wakeup status interrupt | 0x0164 | RO |
| PMU_HP_INT_ENA_REG | PMU sleep/Wakeup interrupt enable | 0x0168 | R/W |
| PMU_HP_INT_CLR_REG | PMU sleep/Wakeup interrupt clear | 0x016C | WT |
| PMU_LP_INT_RAW_REG | Low-power system raw interrupt | 0x0170 | R/WTC/SS |
| PMU_LP_INT_ST_REG | PMU state transition interrupt status | 0x0174 | RO |
| PMU_LP_INT_ENA_REG | PMU state transition interrupt enable | 0x0178 | R/W |
| PMU_LP_INT_CLR_REG | PMU state transition interrupt clear | 0x017C | WT |
| PMU_HP_REGULATOR_CFG_REG | PMU-controlled update of digital regulator configuration | 0x018C | R/W |
| PMU_DATE_REG | Version control | 0x01A8 | R/W |

### 12.8.2 PAU Register Summary

The addresses in this section below are relative to the PAU base address provided in Table 5.3-2 in Chapter 5 *System and Memory*.

| Name | Description | Address | Access |
|---|---|---|---|
| **Control & Configuration Register** | | | |
| PAU_REGDMA_CONF_REG | Peripherals backup configuration/control register | 0x0000 | varies |

| Name | Description | Address | Access |
|---|---|---|---|
| PAU_REGDMA_CLK_CONF_REG | Clock control register | 0x0004 | R/W |
| PAU_REGDMA_ETM_CTRL_REG | ETM start configuration/control register | 0x0008 | varies |
| PAU_REGDMA_LINK_0_ADDR_REG | Link address configuration register 0 | 0x000C | varies |
| PAU_REGDMA_LINK_1_ADDR_REG | Link address configuration register 1 | 0x0010 | varies |
| PAU_REGDMA_LINK_2_ADDR_REG | Link address configuration register 2 | 0x001C | varies |
| PAU_REGDMA_LINK_3_ADDR_REG | Link address configuration register 3 | 0x0018 | varies |
| PAU_REGDMA_LINK_MAC_ADDR_REG | Link MAC address register | 0x000C | varies |
| PAU_REGDMA_CURRENT_LINK_ADDR_REG | Current link address register | 0x0020 | varies |
| PAU_REGDMA_BACKUP_ADDR_REG | Backup address register | 0x0024 | varies |
| PAU_REGDMA_MEM_ADDR_REG | Memory address configuration register | 0x0028 | RO |
| PAU_REGDMA_BKP_CONF_REG | Link configuration register | 0x002C | R/W |
| PAU_INT_ENA_REG | Interrupt register | 0x0030 | R/W |
| PAU_INT_RAW_REG | Interrupt register | 0x0034 | R/WTC/SS |
| PAU_INT_CLR_REG | Interrupt register | 0x0038 | WT |
| PAU_INT_ST_REG | Interrupt register | 0x003C | RO |
| **Version Register** | | | |
| PAU_DATE_REG | Version control register | 0x03FC | R/W |

## 12.9 Registers

For how to program reserved fields, please refer to Section *Programming Reserved Register Field*.

### 12.9.1 PMU Registers

The addresses in this section are relative to the PMU base address provided in Table 5.3-2 in Chapter 5 *System and Memory*.

Register 12.1. PMU_HP_ACTIVE_DIG_POWER_REG (0x0000)

| PMU_HP_ACTIVE_PD_TOP_PD_EN | PMU_HP_ACTIVE_PD_HP_AON_PD_EN | PMU_HP_ACTIVE_PD_HP_CPU_PD_EN | (reserved) | PMU_HP_ACTIVE_PD_HP_WIFI_PD_EN | (reserved) | PMU_HP_ACTIVE_HP_MEM_DSLP | PMU_HP_ACTIVE_VDD_SPI_PD_EN | (reserved) | |
|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 23 | 22 | 21 | 20 0 | |
| 0 | 0 | 0 | 0 | 0 | 0000 | 0 | 0 | 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | Reset |

**PMU_HP_ACTIVE_VDD_SPI_PD_EN** Configures whether to power down external flash in HP_ACTIVE state.
0: Power up
1: Power down
(R/W)

**PMU_HP_ACTIVE_HP_MEM_DSLP** Configures whether to put Internal SRAM*x* (*x*= 0, 1, 2) into Deep-sleep mode in HP_ACTIVE state.
0: Do not put Internal SRAM into Deep-sleep
1: Put Internal SRAM into Deep-sleep
(R/W)

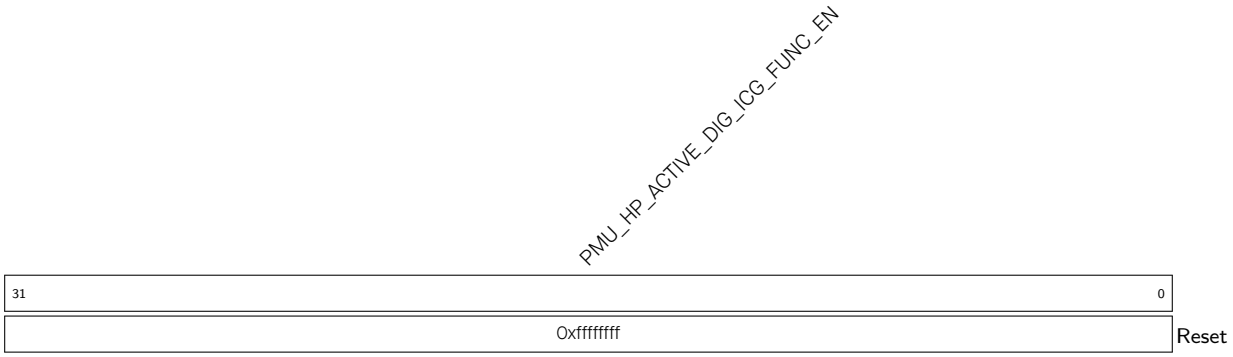
**PMU_HP_ACTIVE_PD_HP_WIFI_PD_EN** Configures whether to power down MODEM domain in HP_ACTIVE state.
0: Power up
1: Power down
(R/W)

**PMU_HP_ACTIVE_PD_HP_CPU_PD_EN** Configures whether to power down CPU domain in HP_ACTIVE state.
0: Power up
1: Power down
(R/W)

**PMU_HP_ACTIVE_PD_HP_AON_PD_EN** Configures whether to power down Modem Power domain in HP_ACTIVE state.
0: Power up
1: Power down
(R/W)

**PMU_HP_ACTIVE_PD_TOP_PD_EN** Configures whether to power down "Peripherals+ROM" domain in HP_ACTIVE state.
0: Power up
1: Power down
(R/W)

Register 12.2. PMU_HP_ACTIVE_ICG_HP_FUNC_REG (0x0004)



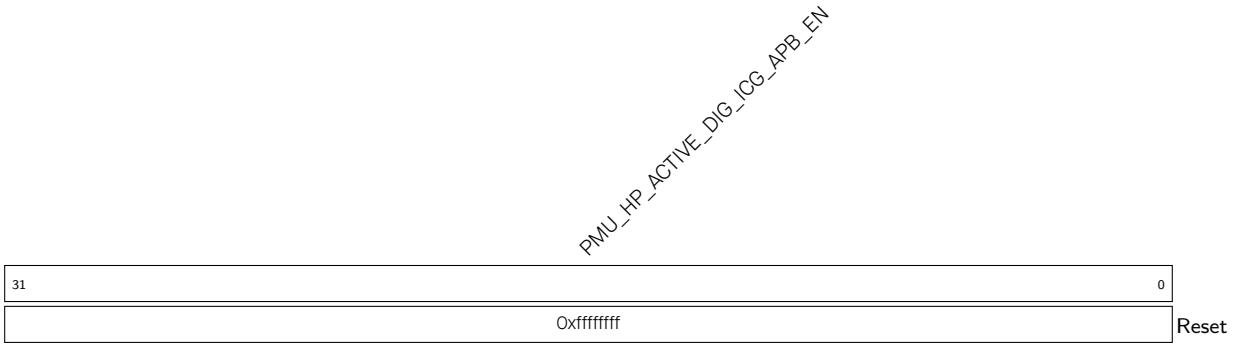

**PMU_HP_ACTIVE_DIG_ICG_FUNC_EN** Configures whether to enable HP system peripherals' function clock in HP_ACTIVE state. For detailed configuration please see Chapter 8 *Reset and Clock* > Table 8.2-8.

0: Disable

1: Enable

(R/W)

Register 12.3. PMU_HP_ACTIVE_ICG_HP_APB_REG (0x0008)



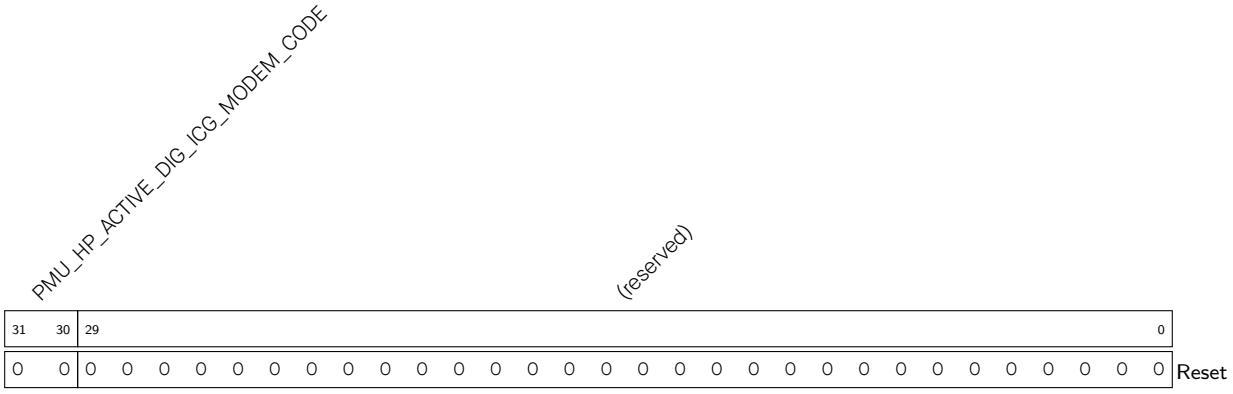

**PMU_HP_ACTIVE_DIG_ICG_APB_EN** Configures whether to enable HP system peripherals' APB clock in HP_ACTIVE state. For detailed configuration please see Chapter 8 *Reset and Clock* > Table 8.2-7.

0: Disable

1: Enable

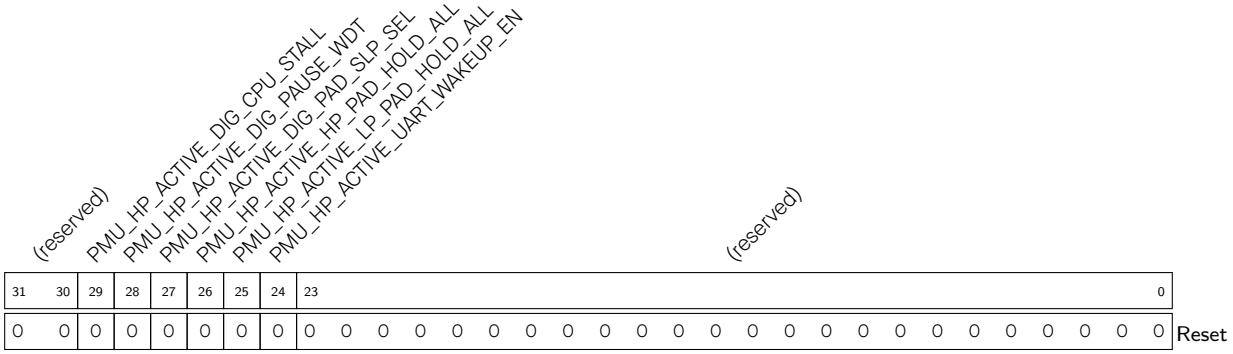
(R/W)

Register 12.4. PMU_HP_ACTIVE_ICG_MODEM_REG (0x000C)




**PMU_HP_ACTIVE_DIG_ICG_MODEM_CODE** Configures the Modem clock gate in HP_ACTIVE state.(R/W)

Register 12.5. 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 GPIOs' configuration in HP_ACTIVE state.

0: Do not hold

1: Hold

(R/W)

**PMU_HP_ACTIVE_HP_PAD_HOLD_ALL** Configures whether to hold HP GPIOs' 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 *7 GPIO Matrix and IO MUX* > Section *7.8 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)

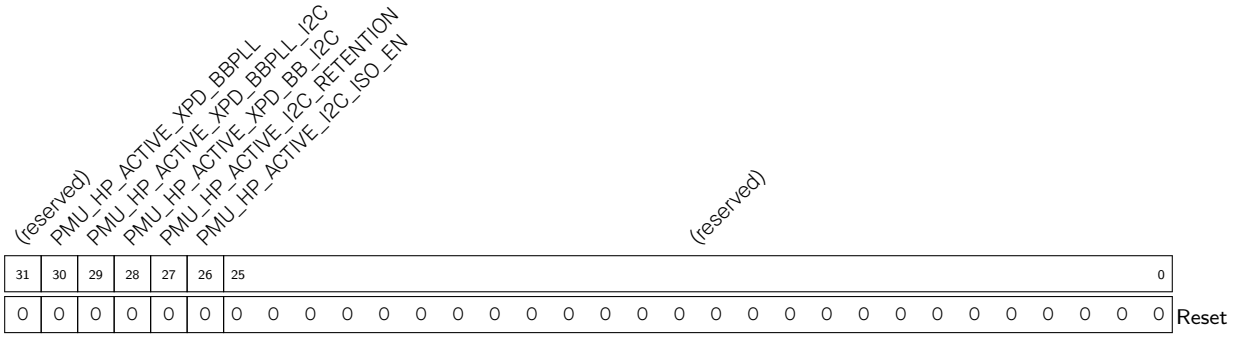
**PMU_HP_ACTIVE_DIG_CPU_STALL** Configures whether to stall CPU in HP_ACTIVE state.

0: Do not stall

1: Stall

(R/W)

Register 12.6. PMU_HP_ACTIVE_HP_CK_POWER_REG (0x0014)




**PMU_HP_ACTIVE_I2C_ISO_EN** Configures whether to enable I2C_ISO in HP_ACTIVE state.

0: Disable

1: Enable

(R/W)

**PMU_HP_ACTIVE_XPD_BB_I2C** Configures whether to enable BB_I2C in HP_ACTIVE state.

0: Disable

1: Enable

(R/W)

**PMU_HP_ACTIVE_XPD_BBPLL_I2C** Configures whether to enable BBPLL_I2C in HP_ACTIVE state.

0: Disable

1: Enable

(R/W)

**PMU_HP_ACTIVE_XPD_BBPLL** Configures whether to enable BBPLL in HP_ACTIVE state.

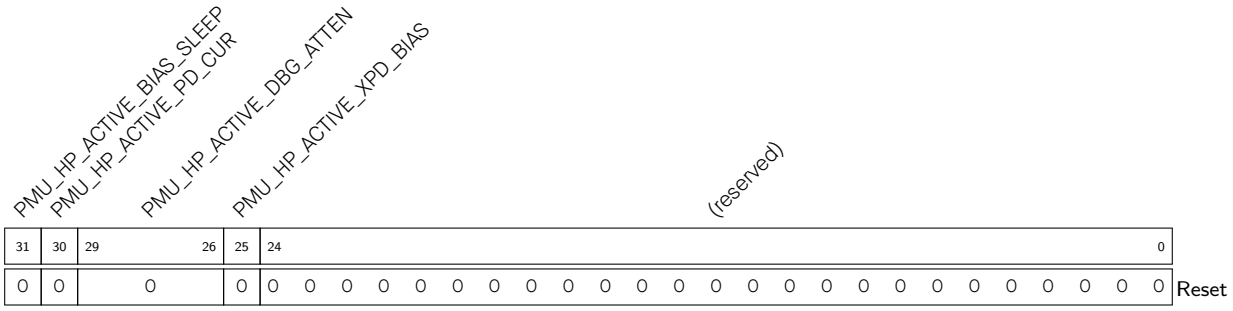
0: Disable

1: Enable

(R/W)

> **Notice:**
>
> BB_I2C, BBPLL_I2C and XTAL must be turned on first before enabling BBPLL.

**Register 12.7. PMU_HP_ACTIVE_BIAS_REG (0x0018)**



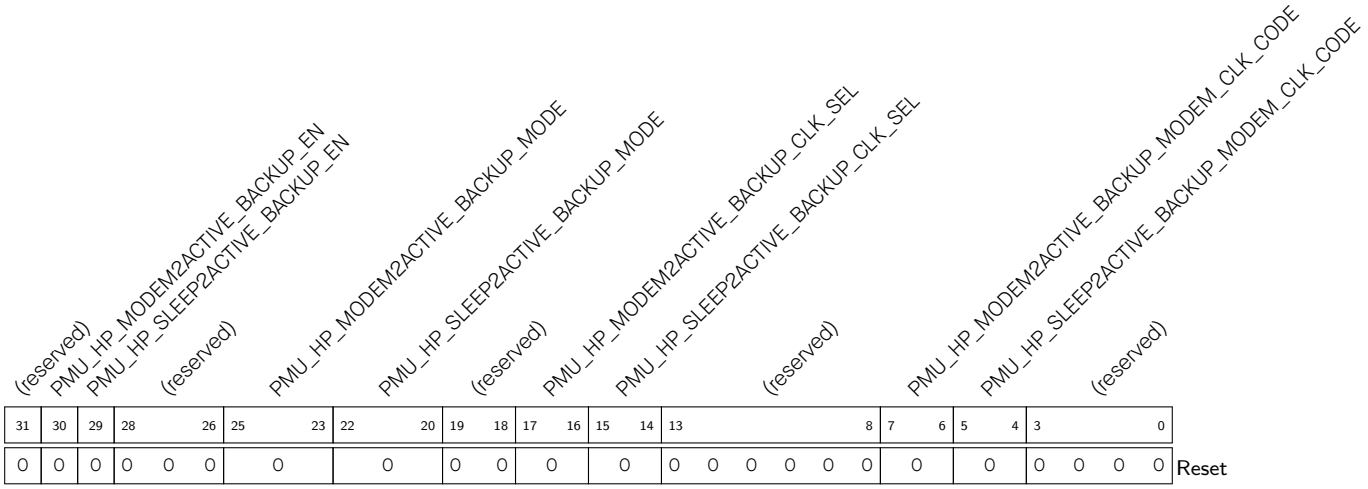

**PMU_HP_ACTIVE_XPD_BIAS** Configures the power supply of BIAS in HP_ACTIVE state. (R/W)

**PMU_HP_ACTIVE_DBG_ATTEN** Configures the degree of attenuation of the analog band gap in HP_ACTIVE state. (R/W)

**PMU_HP_ACTIVE_PD_CUR** Configures the power-down current in HP_ACTIVE state. (R/W)

**PMU_HP_ACTIVE_BIAS_SLEEP** Configures the sleep or wakeup status of the BIAS circuit in HP_ACTIVE state. (R/W)

Register 12.8. PMU_HP_ACTIVE_BACKUP_REG (0x001C)




**PMU_HP_SLEEP2ACTIVE_BACKUP_MODEM_CLK_CODE** Configures the ICG during backup process when PMU state switches from HP_SLEEP to HP_ACTIVE. (R/W)

**PMU_HP_MODEM2ACTIVE_BACKUP_MODEM_CLK_CODE** Configures the ICG during backup process when PMU state switches from HP_MODEM to HP_ACTIVE. (R/W)

**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_MODEM2ACTIVE_BACKUP_CLK_SEL** Configures the backup module's function clock source when PMU state switches from HP_MODEM to HP_ACTIVE. The configuration is the same as the register above. (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.

Higher two bits:

0: From peripheral to memory

1: From memory to peripheral

Lower four bits:

0: PAU_LINK_ADDR_0

1: PAU_LINK_ADDR_1

2: PAU_LINK_ADDR_2

3: PAU_LINK_ADDR_3

(R/W)

Continued on the next page...

Register 12.8. PMU_HP_ACTIVE_BACKUP_REG (0x001C)

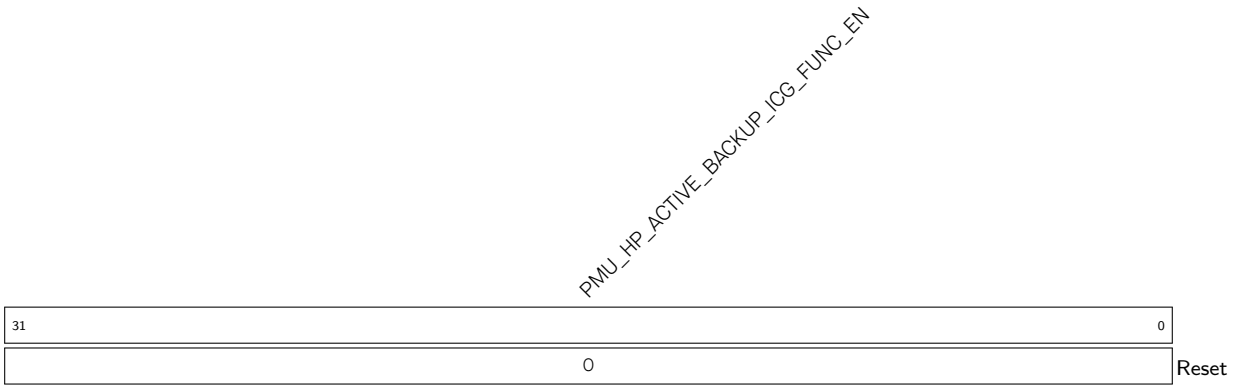
Continued from the previous page...

**PMU_HP_MODEM2ACTIVE_BACKUP_MODE** Configures the backup direction and link list when PMU state switches from HP_MODEM to HP_ACTIVE. The configuration is the same as the register above. (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)

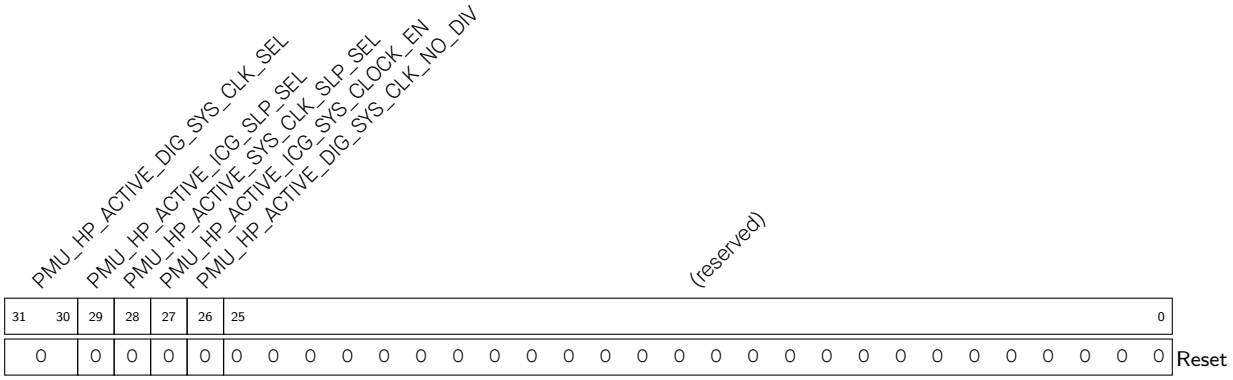
**PMU_HP_MODEM2ACTIVE_BACKUP_EN** Configures whether to enable the backup flow when PMU state switches from HP_MODEM to HP_ACTIVE.
0: Disable backup
1: Enable backup
(R/W)

Register 12.9. PMU_HP_ACTIVE_BACKUP_CLK_REG (0x0020)




**PMU_HP_ACTIVE_BACKUP_ICG_FUNC_EN** Configures whether to enable each peripherals' function clock when the target state is HP_ACTIVE. For details, please refer to Chapter 8 *Reset and Clock*.
0: Disable
1: Enable
(R/W)

Register 12.10. PMU_HP_ACTIVE_SYSCLK_REG (0x0024)




**PMU_HP_ACTIVE_DIG_SYS_CLK_NO_DIV** Configures whether to enable the clock division for HP_ROOT_CLK in HP_ACTIVE state. (R/W)

**PMU_HP_ACTIVE_ICG_SYS_CLOCK_EN** Configures whether to enable HP_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 HP_ROOT_CLK in HP_ACTIVE state.

0: XTAL

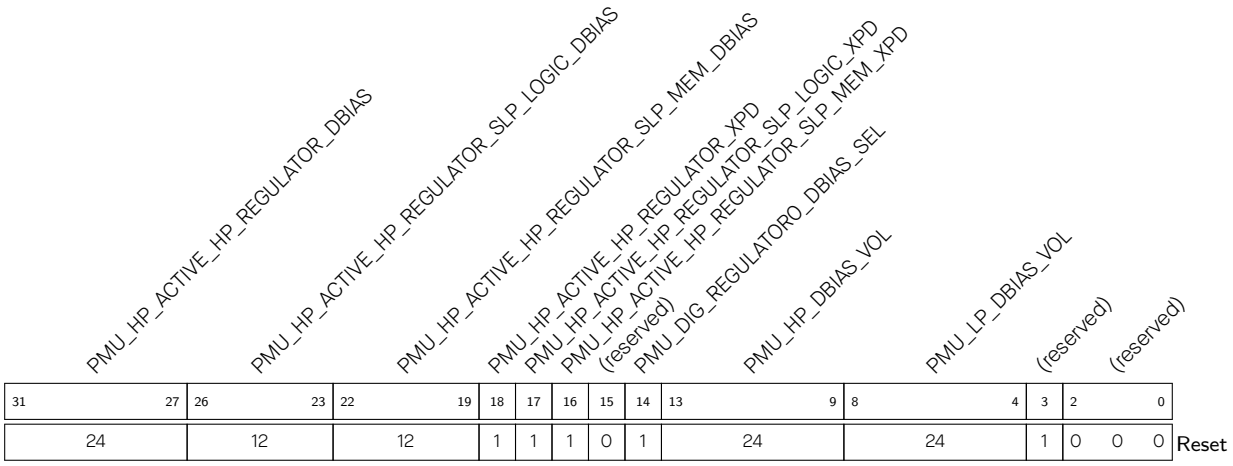
1: PLL_CLK

2: RC_FAST_CLK

3: Invalid value

(R/W)

Register 12.11. PMU_HP_ACTIVE_HP_REGULATOR0_REG (0x0028)




**PMU_LP_DBIAS_VOL** Indicates the voltage level based on PVT_RTC data, which is the feedback from the PVT module in LP_SLEEP state. (RO)

**PMU_HP_DBIAS_VOL** Indicates the voltage level based on PVT data, which is the feedback from PVT module in HP_SLEEP state. (RO)

**PMU_DIG_REGULATOR0_DBIAS_SEL** Configures to select the DREG configuration source for main HP and LP regulators in different PMU states.

0: PMU_HP_DBIAS_VOL

1: DREG value of respective main regulators in respective PMU states configured via the following register fields:

- PMU_HP_ACTIVE_HP_REGULATOR_DBIAS
- PMU_HP_SLEEP_HP_REGULATOR_DBIAS
- PMU_HP_ACTIVE_LP_REGULATOR_DBIAS
- PMU_HP_SLEEP_LP_REGULATOR_DBIAS

(R/W)

**PMU_HP_ACTIVE_HP_REGULATOR_SLP_MEM_XPD** Configures whether to enable the secondary regulator powering the HP memory in HP_ACTIVE state.

0: Disable

1: Enable

(R/W)

**PMU_HP_ACTIVE_HP_REGULATOR_SLP_LOGIC_XPD** Configures whether to enable the secondary regulator powering the logic circuit in all HP domains in HP_ACTIVE state.

0: Disable

1: Enable

(R/W)

Continued on the next page...

Register 12.11. PMU_HP_ACTIVE_HP_REGULATOR0_REG (0x0028)

Continued from the previous page...

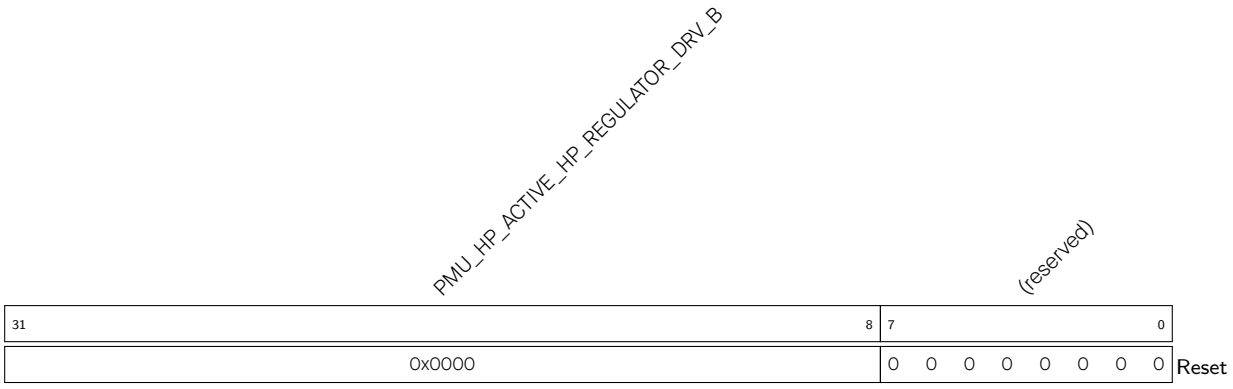
**PMU_HP_ACTIVE_HP_REGULATOR_XPD** Configures whether to enable the main HP regulator in HP_ACTIVE state.
0: Disable
1: Enable
(R/W)

**PMU_HP_ACTIVE_HP_REGULATOR_SLP_MEM_DBIAS** Configures the DREG value of the secondary HP regulator powering the HP memory in HP_ACTIVE state, adjusting its output voltage. (R/W)

**PMU_HP_ACTIVE_HP_REGULATOR_SLP_LOGIC_DBIAS** Configures the DREG value of the secondary HP regulator powering the logic circuits in HP domain in HP_ACTIVE state, adjusting its output voltage. (R/W)

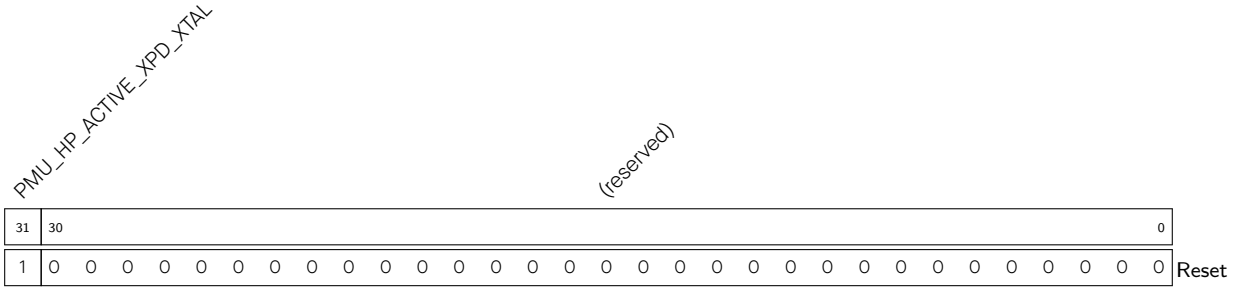
**PMU_HP_ACTIVE_HP_REGULATOR_DBIAS** Configures the DREG value of the main HP regulator in HP_ACTIVE state, adjusting its output voltage. (R/W)

Register 12.12. PMU_HP_ACTIVE_HP_REGULATOR1_REG (0x002C)




**PMU_HP_ACTIVE_HP_REGULATOR_DRV_B** Configures the drive strength of the main HP regulator in HP_ACTIVE state. (R/W)

**Register 12.13. PMU_HP_ACTIVE_XTAL_REG (0x0030)**



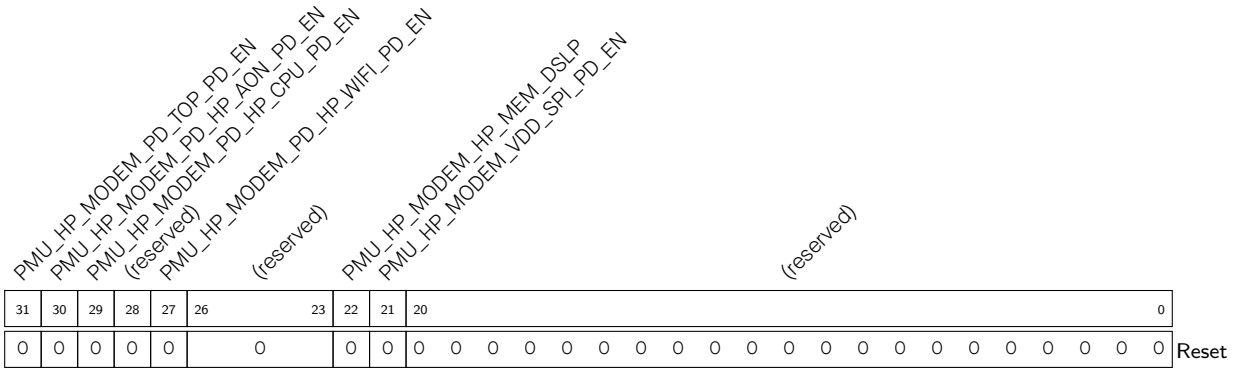

**PMU_HP_ACTIVE_XPD_XTAL** Configures whether to enable XTAL_CLK in HP_ACTIVE state.

0: Disable

1: Enable

(R/W)

Register 12.14. PMU_HP_MODEM_DIG_POWER_REG (0x0034)




**PMU_HP_MODEM_VDD_SPI_PD_EN** Configures whether to power down external flash in HP_MODEM state.

0: Power up

1: Power down

(R/W)

**PMU_HP_MODEM_HP_MEM_DSLP** Configures whether to put Internal SRAM*x* (*x*= 0, 1, 2) into Deep-sleep mode in HP_MODEM state.

0: Do not put Internal SRAM into Deep-sleep

1: Put Internal SRAM into Deep-sleep

(R/W)

**PMU_HP_MODEM_PD_HP_WIFI_PD_EN** Configures whether to power down MODEM domain in HP_MODEM state.

0: Power up

1: Power down

(R/W)

**PMU_HP_MODEM_PD_HP_CPU_PD_EN** Configures whether to power down CPU domain in HP_MODEM state.

0: Power up

1: Power down

(R/W)

**PMU_HP_MODEM_PD_HP_AON_PD_EN** Configures whether to power down Modem Power domain in HP_MODEM state.

0: Power up

1: Power down

(R/W)

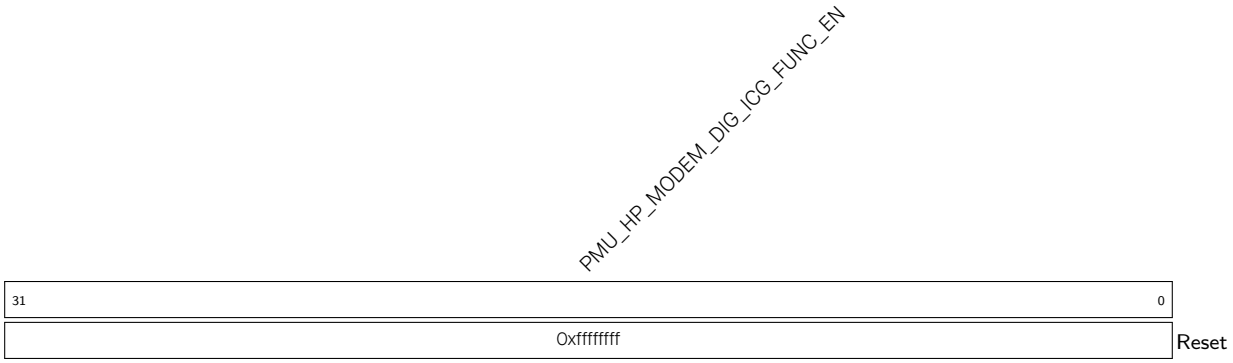
**PMU_HP_MODEM_PD_TOP_PD_EN** Configures whether to power down "Peripherals+ROM" domain in HP_MODEM state.

0: Power up

1: Power down

(R/W)

Register 12.15. PMU_HP_MODEM_ICG_HP_FUNC_REG (0x0038)



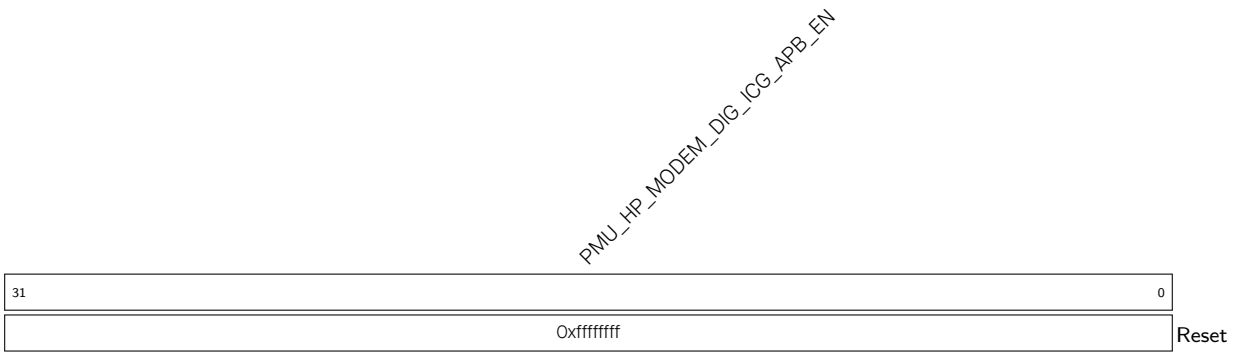

**PMU_HP_MODEM_DIG_ICG_FUNC_EN** Configures whether to enable HP system peripherals' function clock in HP_MODEM state. For detailed configuration please see Chapter 8 *Reset and Clock* > Table 8.2-8.

0: Disable

1: Enable

(R/W)

Register 12.16. PMU_HP_MODEM_ICG_HP_APB_REG (0x003C)



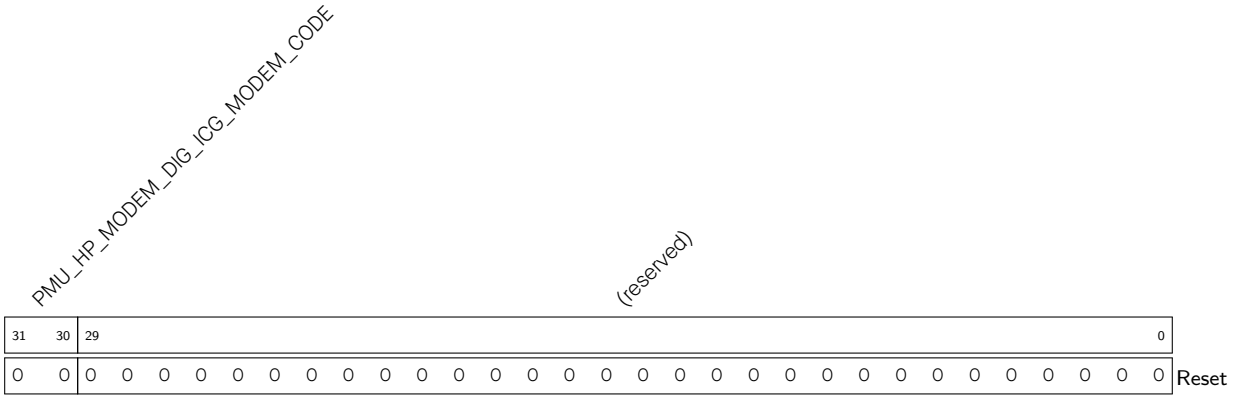

**PMU_HP_MODEM_DIG_ICG_APB_EN** Configures whether to enable HP system peripherals' APB clock in HP_MODEM state. For detailed configuration please see Chapter 8 *Reset and Clock* > Table 8.2-7.

0: Disable

1: Enable

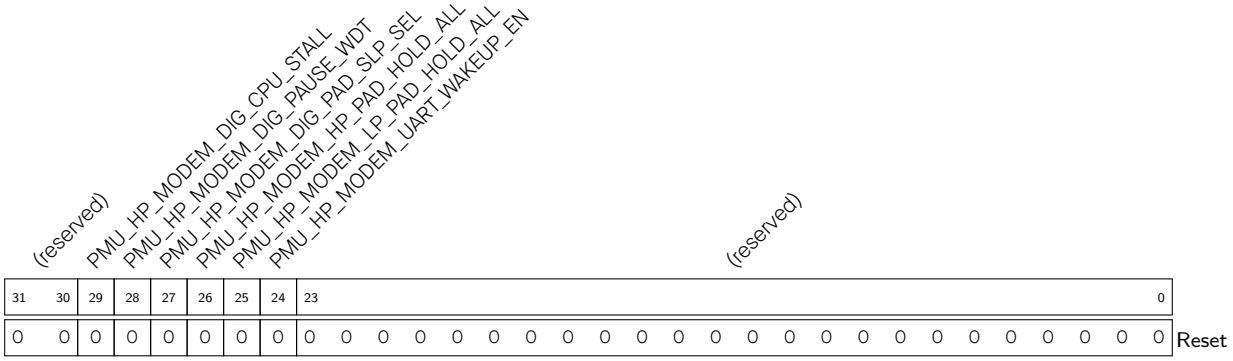
(R/W)

Register 12.17. PMU_HP_MODEM_ICG_MODEM_REG (0x0040)




**PMU_HP_MODEM_DIG_ICG_MODEM_CODE** Configures the Modem clock gate in HP_MODEM state. (R/W)

Register 12.18. PMU_HP_MODEM_HP_SYS_CNTL_REG (0x0044)




**PMU_HP_MODEM_UART_WAKEUP_EN** Configures whether to enable UART wake-up function in HP_MODEM state.

0: Disable wake-up function

1: Enable wake-up function

(R/W)

**PMU_HP_MODEM_LP_PAD_HOLD_ALL** Configures whether to hold LP GPIOs' configuration in HP_MODEM state.

0: Do not hold

1: Hold

(R/W)

**PMU_HP_MODEM_HP_PAD_HOLD_ALL** Configures whether to hold HP GPIOs' configuration in HP_MODEM state.

0: Do not hold

1: Hold

(R/W)

**PMU_HP_MODEM_DIG_PAD_SLP_SEL** Configures whether to use Light-sleep mode configuration for GPIO in HP_MODEM state.

0: Use normal configuration

1: Use Light-sleep mode configuration. For details please refer to Chapter 7 *GPIO Matrix and IO MUX* > Section 7.8 *Pin Functions in Light-sleep*.

(R/W)

**PMU_HP_MODEM_DIG_PAUSE_WDT** Configures whether to pause watchdog in HP_MODEM state.

0: Do not pause

1: Pause

(R/W)

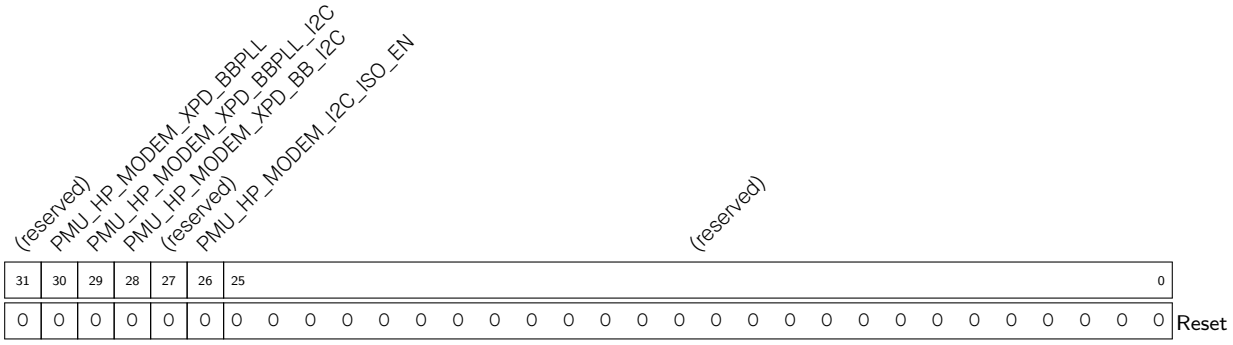
**PMU_HP_MODEM_DIG_CPU_STALL** Configures whether to stall CPU in HP_MODEM state.

0: Do not stall

1: Stall

(R/W)

Register 12.19. PMU_HP_MODEM_HP_CK_POWER_REG (0x0048)




**PMU_HP_MODEM_I2C_ISO_EN** Configures whether to enable I2C_ISO in HP_MODEM state.

0: Disable

1: Enable

(R/W)

**PMU_HP_MODEM_XPD_BB_I2C** Configures whether to enable BB_I2C in HP_MODEM state.

0: Disable

1: Enable

(R/W)

**PMU_HP_MODEM_XPD_BBPLL_I2C** Configures whether to enable BBPLL_I2C in HP_MODEM state.

0: Disable

1: Enable

(R/W)

**PMU_HP_MODEM_XPD_BBPLL** Configures whether to enable BBPLL in HP_MODEM state.

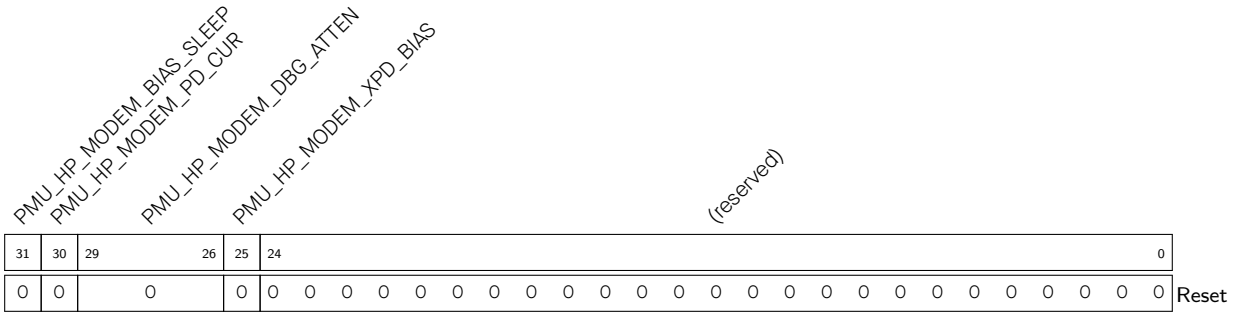
0: Disable

1: Enable

(R/W)

> **Notice:**
>
> BB_I2C, BBPLL_I2C and XTAL must be turned on first before enabling BBPLL.

Register 12.20. PMU_HP_MODEM_BIAS_REG (0x004C)



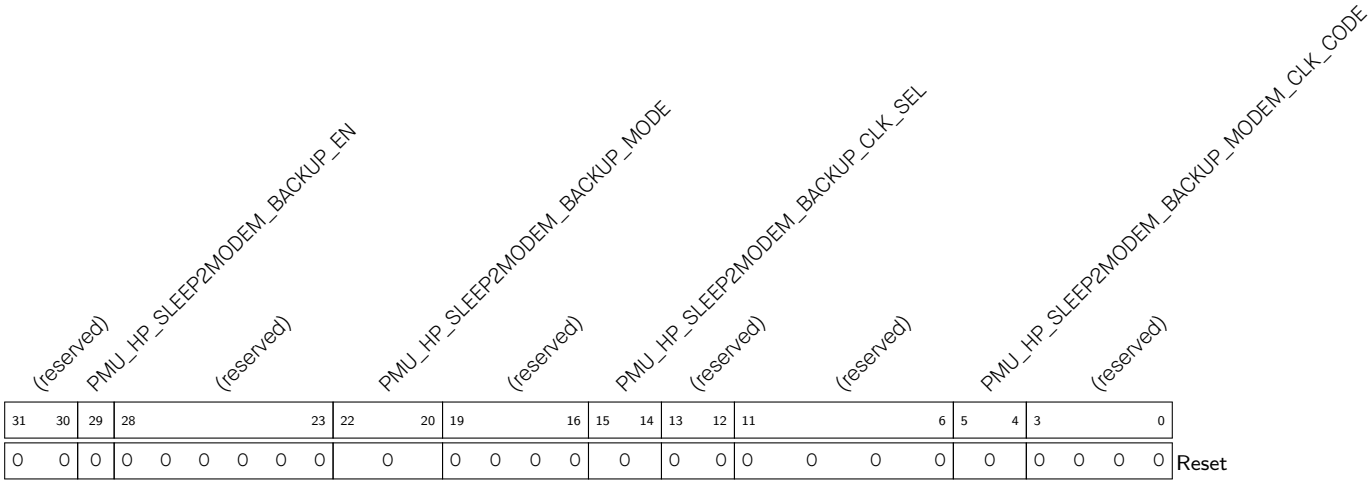

**PMU_HP_MODEM_XPD_BIAS** Configures the power supply of BIAS in HP_MODEM state. (R/W)

**PMU_HP_MODEM_DBG_ATTEN** Configures the degree of attenuation of the analog band gap in HP_MODEM state. (R/W)

**PMU_HP_MODEM_PD_CUR** Configures the power-down current in HP_MODEM state. (R/W)

**PMU_HP_MODEM_BIAS_SLEEP** Configures the sleep or wakeup status of the BIAS circuit in HP_MODEM state. (R/W)

Register 12.21. PMU_HP_MODEM_BACKUP_REG (0x0050)




**PMU_HP_SLEEP2MODEM_BACKUP_MODEM_CLK_CODE** Configures the ICG during backup process when PMU state switches from HP_SLEEP to HP_MODEM. (R/W)

**PMU_HP_SLEEP2MODEM_BACKUP_CLK_SEL** Configures the backup module's function clock source when PMU state switches from HP_SLEEP to HP_MODEM.

0: Select XTAL

1: Select PLL_CLK

2: Select RC_FAST_CLK

3: Invalid value

(R/W)

**PMU_HP_SLEEP2MODEM_BACKUP_MODE** Configures the backup direction and link list when PMU state switches switch from HP_SLEEP to HP_MODEM.

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)

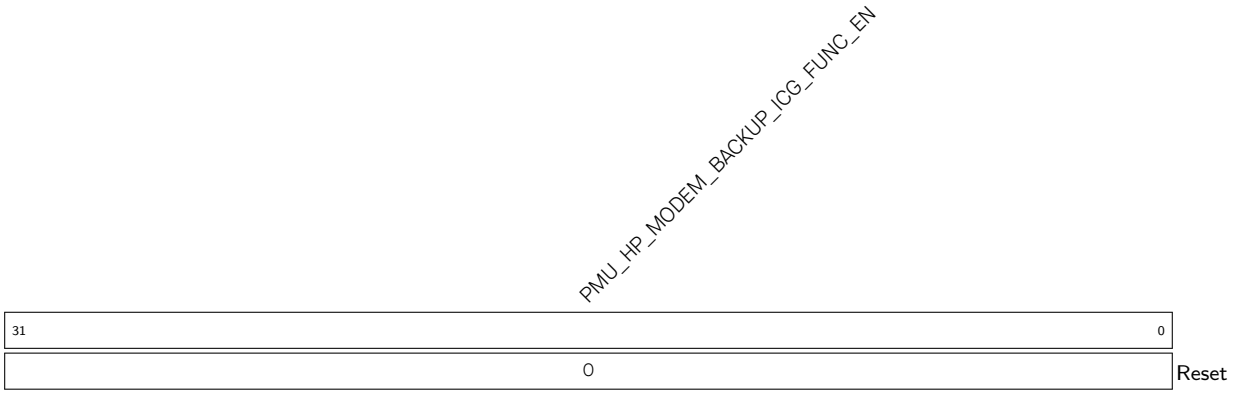
**PMU_HP_SLEEP2MODEM_BACKUP_EN** Configures whether to enable the backup flow when PMU state switches from HP_SLEEP to HP_MODEM.

0: Disable backup

1: Enable backup

(R/W)

Register 12.22. PMU_HP_MODEM_BACKUP_CLK_REG (0x0054)



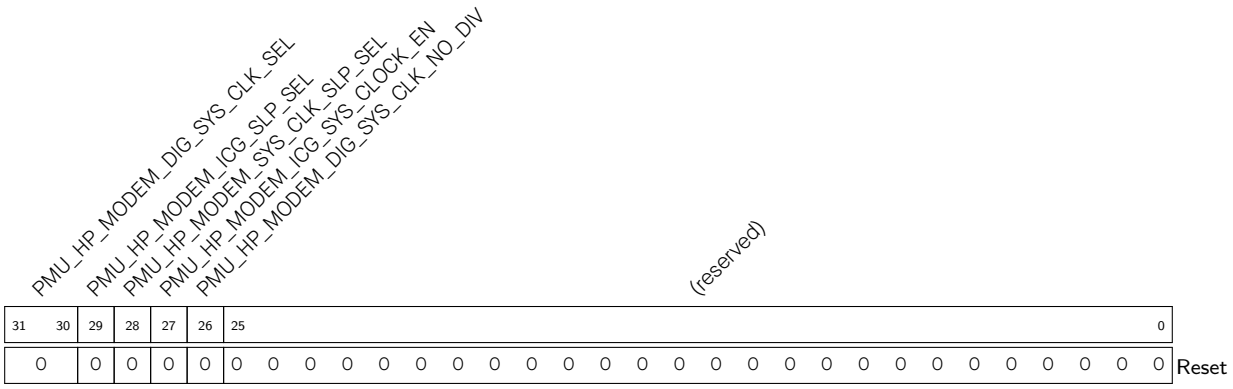

**PMU_HP_MODEM_BACKUP_ICG_FUNC_EN** Configures whether to enable each peripherals' function clock when the target state is HP_MODEM. For details, please refer to Chapter 8 *Reset and Clock*.

0: Disable

1: Enable

(R/W)

Register 12.23. PMU_HP_MODEM_SYSCLK_REG (0x0058)




**PMU_HP_MODEM_DIG_SYS_CLK_NO_DIV** Configures whether to enable the clock division for HP_ROOT_CLK in HP_MODEM state. (R/W)

**PMU_HP_MODEM_ICG_SYS_CLOCK_EN** Configures whether to enable HP_ROOT_CLK in HP_MODEM state.

0: Disable

1: Enable

(R/W)

**PMU_HP_MODEM_SYS_CLK_SLP_SEL** Configures whether to allow PMU to control the clock source in HP_MODEM state.

0: Controlled by PCR registers

1: Controlled by PMU

(R/W)

**PMU_HP_MODEM_ICG_SLP_SEL** Configures whether to allow PMU to control the clock gating in HP_MODEM state.

0: Controlled by PCR registers

1: Controlled by PMU

(R/W)

**PMU_HP_MODEM_DIG_SYS_CLK_SEL** Configures the source of HP_ROOT_CLK in HP_MODEM state.

0: XTAL

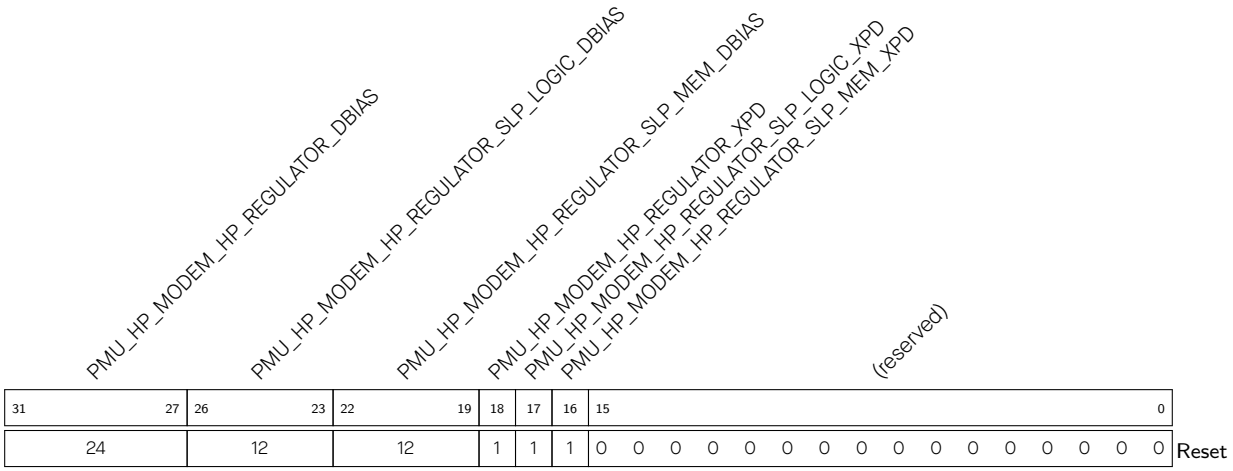
1: PLL_CLK

2: RC_FAST_CLK

3: Invalid value

(R/W)

Register 12.24. PMU_HP_MODEM_HP_REGULATOR0_REG (0x005C)




**PMU_HP_MODEM_HP_REGULATOR_SLP_MEM_XPD** Configures whether to enable the secondary regulator powering the HP memory in HP domain in HP_MODEM state.

0: Disable the regulator

1: Enable the regulator

(R/W)

**PMU_HP_MODEM_HP_REGULATOR_SLP_LOGIC_XPD** Configures whether to enable the secondary regulator powering the logic circuit in HP domain in HP_MODEM state.

0: Disable the regulator

1: Enable the regulator

(R/W)

**PMU_HP_MODEM_HP_REGULATOR_XPD** Configures whether to enable the main HP regulator in HP_MODEM state.

0: Disable

1: Enable

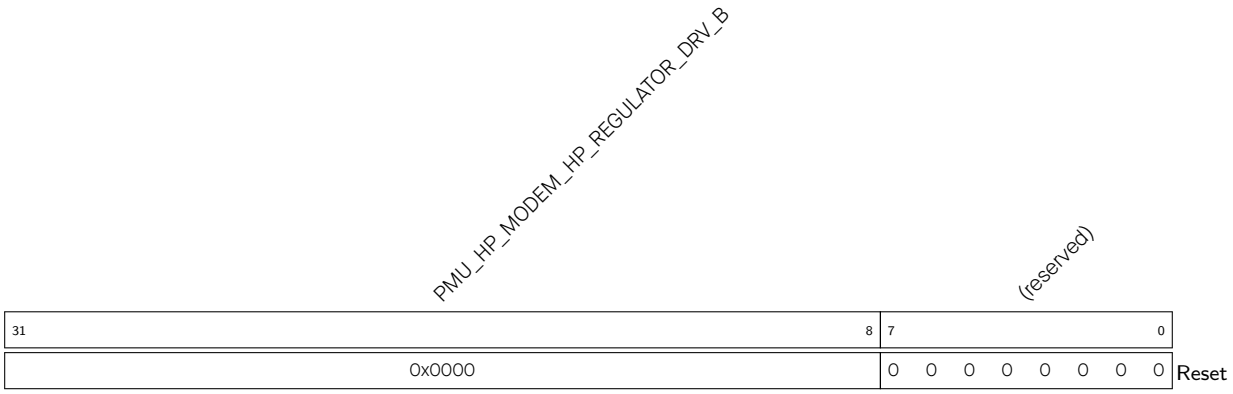
(R/W)

**PMU_HP_MODEM_HP_REGULATOR_SLP_MEM_DBIAS** Configures the DREG value of the secondary HP regulator powering the HP memory in HP_MODEM state, adjusting its output voltage. (R/W)

**PMU_HP_MODEM_HP_REGULATOR_SLP_LOGIC_DBIAS** Configures the DREG value of the secondary HP regulator powering the logic circuits in HP domain in HP_MODEM state, adjusting its output voltage. (R/W)

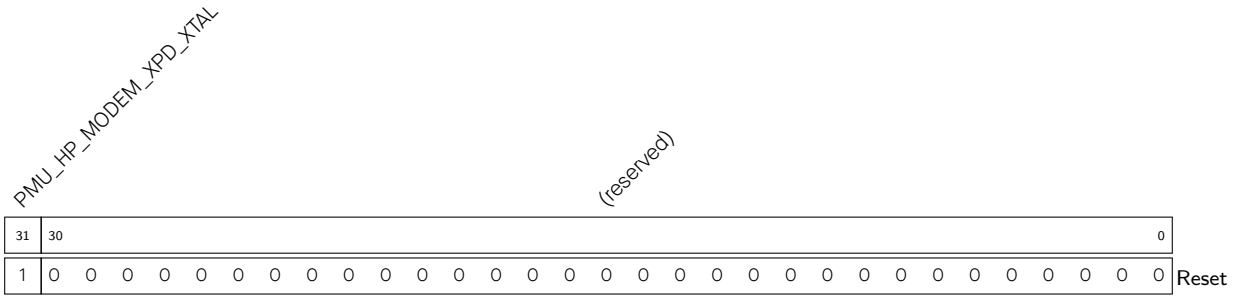
**PMU_HP_MODEM_HP_REGULATOR_DBIAS** Configures the DREG value of the main HP regulator in HP_MODEM state, adjusting its output voltage. (R/W)

Register 12.25. PMU_HP_MODEM_HP_REGULATOR1_REG (0x0060)




**PMU_HP_MODEM_HP_REGULATOR_DRV_B** Configures the drive strength of the main HP regulator in HP_MODEM state. (R/W)

Register 12.26. PMU_HP_MODEM_XTAL_REG (0x0064)



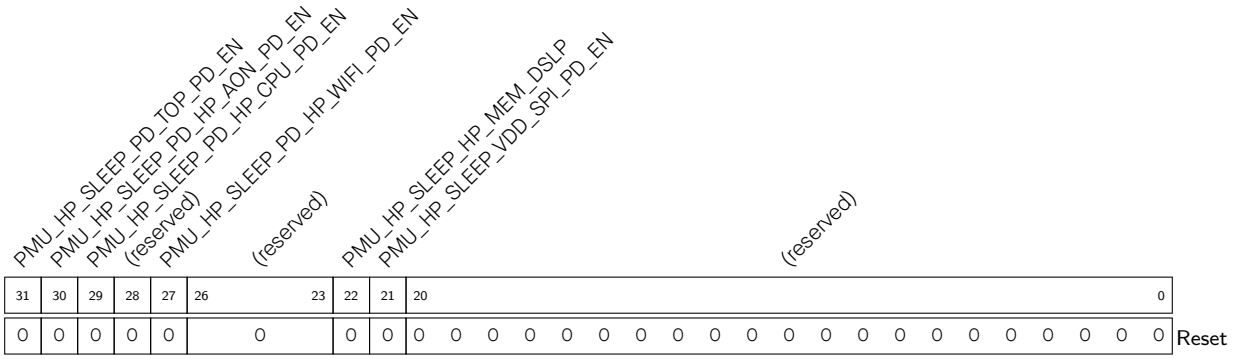

**PMU_HP_MODEM_XPD_XTAL** Configures whether to enable XTAL_CLK in HP_MODEM state.

0: Disable

1: Enable

(R/W)

**Register 12.27. PMU_HP_SLEEP_DIG_POWER_REG (0x0068)**




**PMU_HP_SLEEP_VDD_SPI_PD_EN** Configures whether to power down external flash in HP_SLEEP state.

0: Power up

1: Power down

(R/W)

**PMU_HP_SLEEP_HP_MEM_DSLP** Configures whether to put Internal SRAM*x* (*x*= 0, 1, 2) into Deep-sleep mode in HP_SLEEP state.

0: Do not put Internal SRAM into Deep-sleep

1: Put Internal SRAM into Deep-sleep

(R/W)

**PMU_HP_SLEEP_PD_HP_WIFI_PD_EN** Configures whether to power down MODEM domain 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 CPU domain in HP_SLEEP state.

0: Power up

1: Power down

(R/W)

**PMU_HP_SLEEP_PD_HP_AON_PD_EN** Configures whether to power down Modem Power domain in HP_SLEEP state.

0: Power up

1: Power down

(R/W)

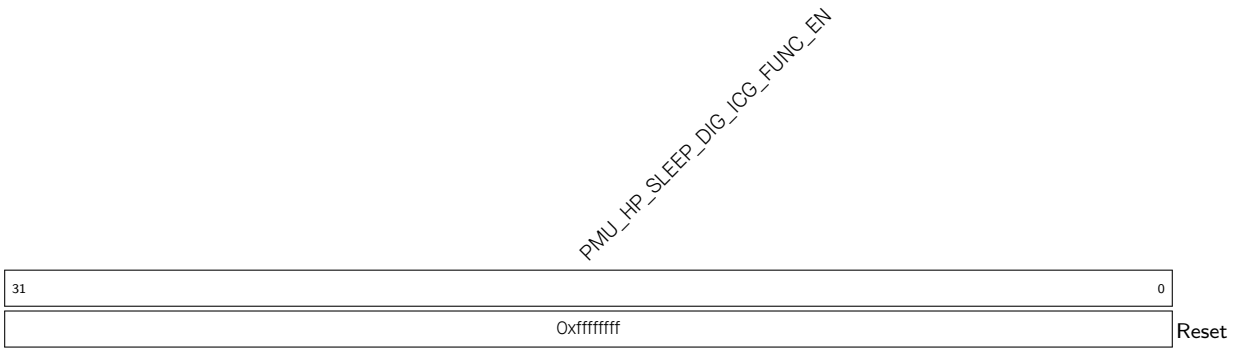
**PMU_HP_SLEEP_PD_TOP_PD_EN** Configures whether to power down “Peripherals+ROM” domain in HP_SLEEP state.

0: Power up

1: Power down

(R/W)

Register 12.28. PMU_HP_SLEEP_ICG_HP_FUNC_REG (0x006C)



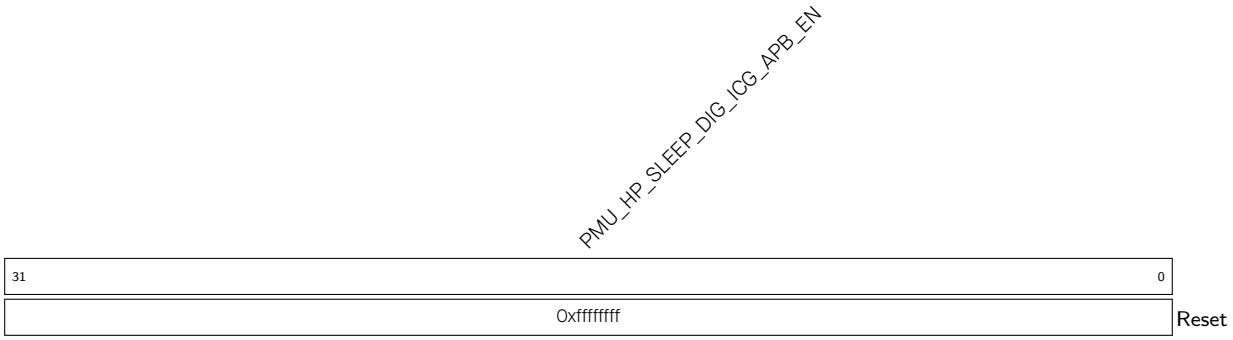

**PMU_HP_SLEEP_DIG_ICG_FUNC_EN** Configures whether to enable HP system peripherals' function clock in HP_SLEEP state. For detailed configuration please see Chapter 8 *Reset and Clock* > Table 8.2-8.

0: Disable

1: Enable

(R/W)

Register 12.29. PMU_HP_SLEEP_ICG_HP_APB_REG (0x0070)



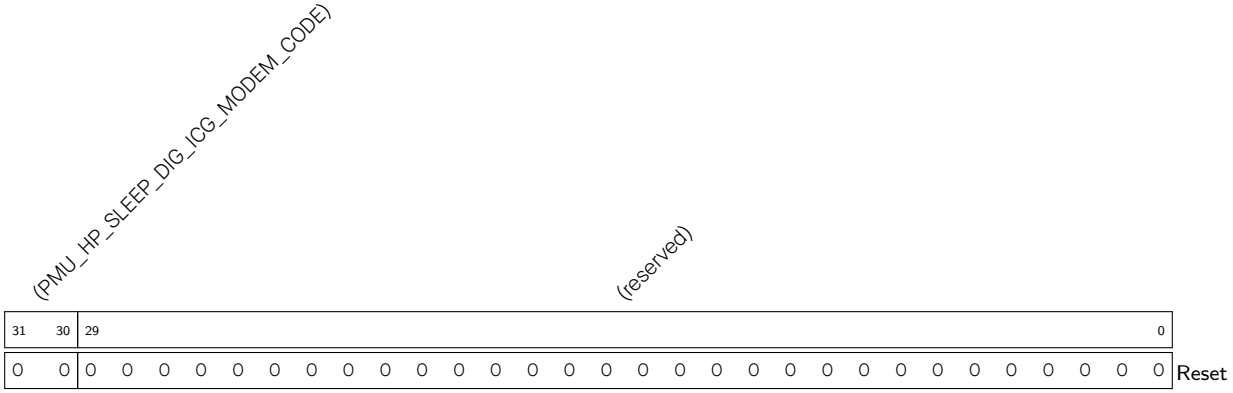

**PMU_HP_SLEEP_DIG_ICG_APB_EN** Configures whether to enable HP system peripherals' APB clock in HP_SLEEP state. For detailed configuration please see Chapter 8 *Reset and Clock* > Table 8.2-7.

0: Disable

1: Enable

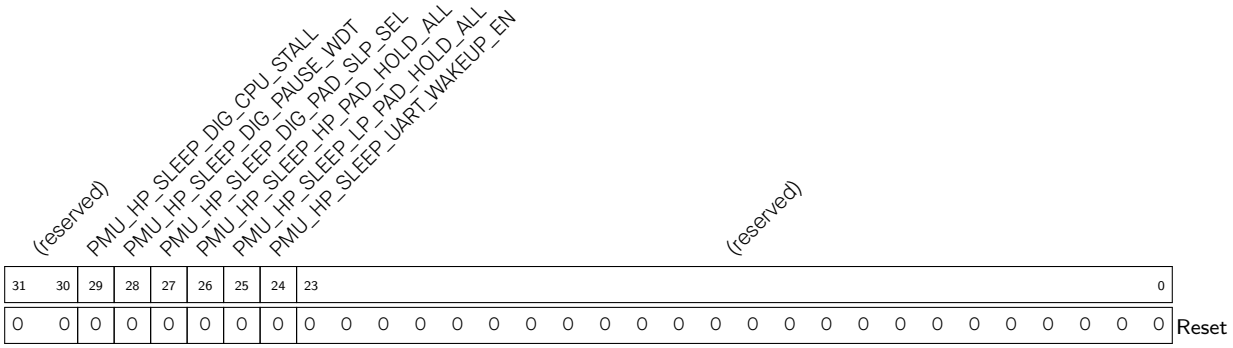
(R/W)

Register 12.30. PMU_HP_SLEEP_ICG_MODEM_REG (0x0074)




**PMU_HP_SLEEP_DIG_ICG_MODEM_CODE** Configures the Modem clock gate in HP_SLEEP state. (R/W)

Register 12.31. 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 GPIOs' configuration in HP_SLEEP state.

0: Do not hold

1: Hold

(R/W)

**PMU_HP_SLEEP_HP_PAD_HOLD_ALL** Configures whether to hold HP GPIOs' 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 7 *GPIO Matrix and IO MUX* > Section 7.8 *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)

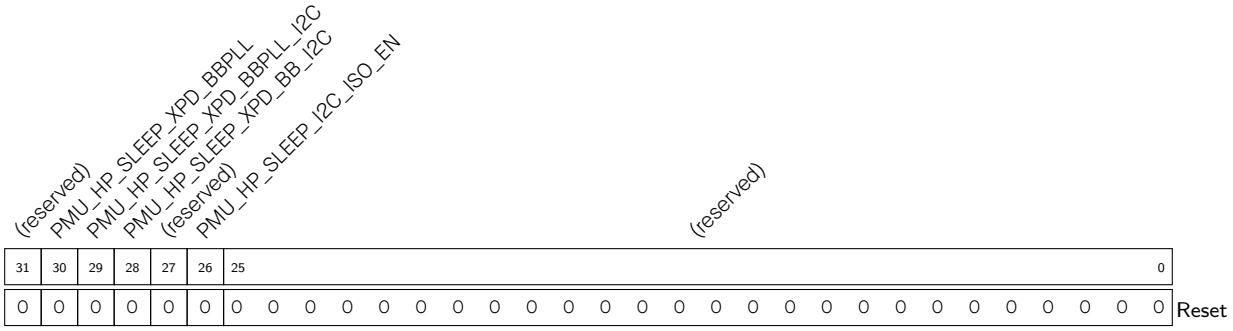
**PMU_HP_SLEEP_DIG_CPU_STALL** Configures whether to stall CPU in HP_SLEEP state.

0: Do not stall

1: Stall

(R/W)

Register 12.32. PMU_HP_SLEEP_HP_CK_POWER_REG (0x007C)




**PMU_HP_SLEEP_I2C_ISO_EN** Configures whether to enable I2C_ISO in HP_SLEEP state.

0: Disable

1: Enable

(R/W)

**PMU_HP_SLEEP_XPD_BB_I2C** Configures whether to enable BB_I2C in HP_SLEEP state.

0: Disable

1: Enable

(R/W)

**PMU_HP_SLEEP_XPD_BBPLL_I2C** Configures whether to enable BBPLL_I2C in HP_SLEEP state.

0: Disable

1: Enable

(R/W)

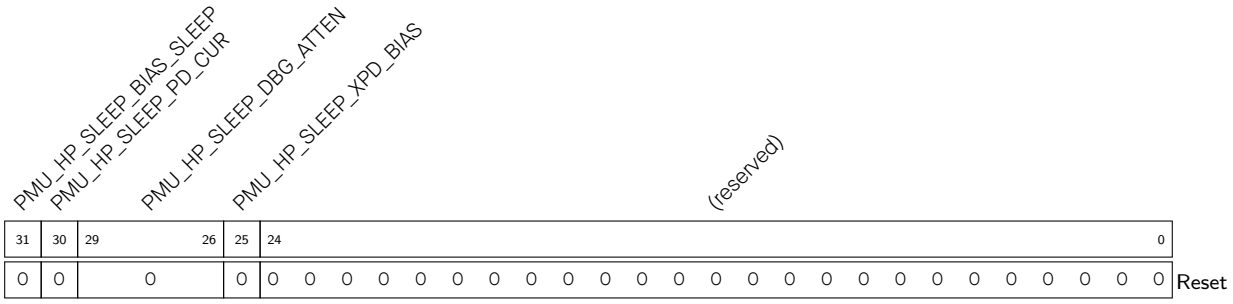
**PMU_HP_SLEEP_XPD_BBPLL** Configures whether to enable BBPLL in HP_SLEEP state.

0: Disable

1: Enable

(R/W)

**Register 12.33. PMU_HP_SLEEP_BIAS_REG (0x0080)**



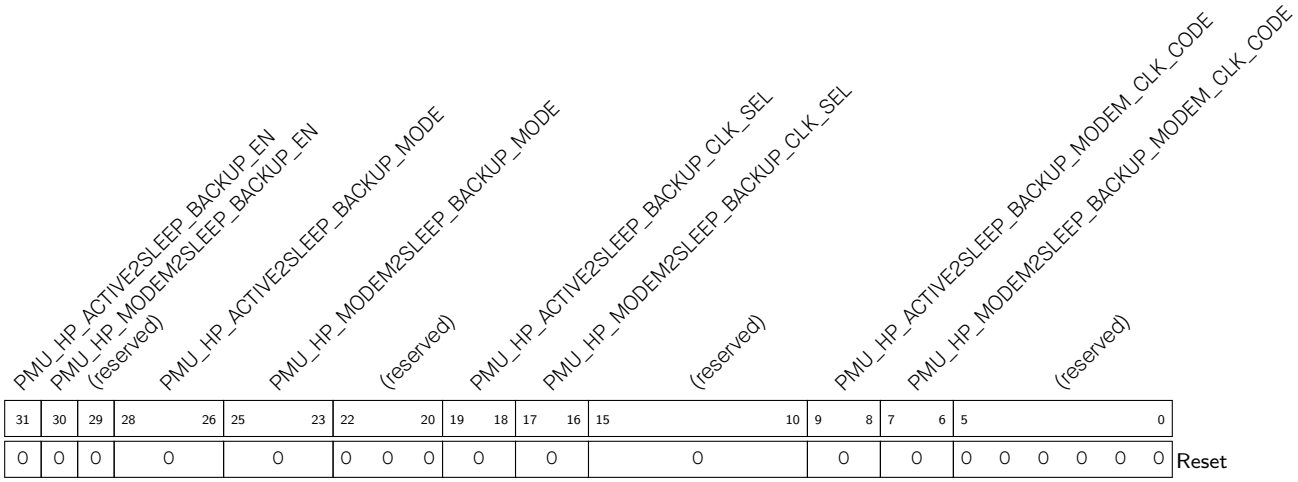

**PMU_HP_SLEEP_XPD_BIAS** Configures the power supply of BIAS in HP_SLEEP state. (R/W)

**PMU_HP_SLEEP_DBG_ATTEN** Configures the degree of attenuation of the analog band gap in HP_SLEEP state. (R/W)

**PMU_HP_SLEEP_PD_CUR** Configures the power-down current in HP_SLEEP state. (R/W)

**PMU_HP_SLEEP_BIAS_SLEEP** Configures the sleep or wakeup status of the BIAS circuit in HP_SLEEP state. (R/W)

Register 12.34. PMU_HP_SLEEP_BACKUP_REG (0x0084)




**PMU_HP_MODEM2SLEEP_BACKUP_MODEM_CLK_CODE** Configures the ICG during backup process when PMU state switches from HP_MODEM to HP_SLEEP. (R/W)

**PMU_HP_ACTIVE2SLEEP_BACKUP_MODEM_CLK_CODE** Configures the ICG during backup process when PMU state switches from HP_ACTIVE to HP_SLEEP. (R/W)

**PMU_HP_MODEM2SLEEP_BACKUP_CLK_SEL** Configures the backup module's function clock source when PMU state switches from HP_MODEM to HP_SLEEP.
0: Select XTAL
1: Select PLL_CLK
2: Select RC_FAST_CLK
3: Invalid value
(R/W)

**PMU_HP_ACTIVE2SLEEP_BACKUP_CLK_SEL** Configures the backup module function clock source when PMU state switches from HP_ACTIVE to HP_SLEEP. The configuration is the same as the register above. (R/W)

**PMU_HP_MODEM2SLEEP_BACKUP_MODE** Configures the backup direction and link list when PMU state switches switch from HP_MODEM to HP_SLEEP.
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)

Continued on the next page...

Register 12.34. PMU_HP_SLEEP_BACKUP_REG (0x0084)

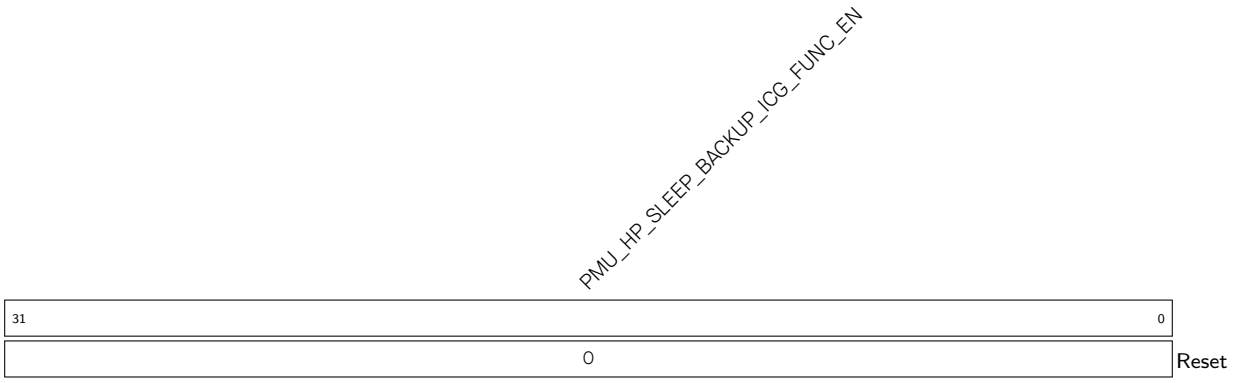
Continued from the previous page...

**PMU_HP_ACTIVE2SLEEP_BACKUP_MODE** Configures the backup direction and link list when PMU state switches from HP_ACTIVE to HP_MODEM. The configuration is the same as the register above. (R/W)

**PMU_HP_MODEM2SLEEP_BACKUP_EN** Configures whether to enable the backup flow when PMU state switches from HP_MODEM to HP_SLEEP.
0: Disable backup
1: Enable backup
(R/W)

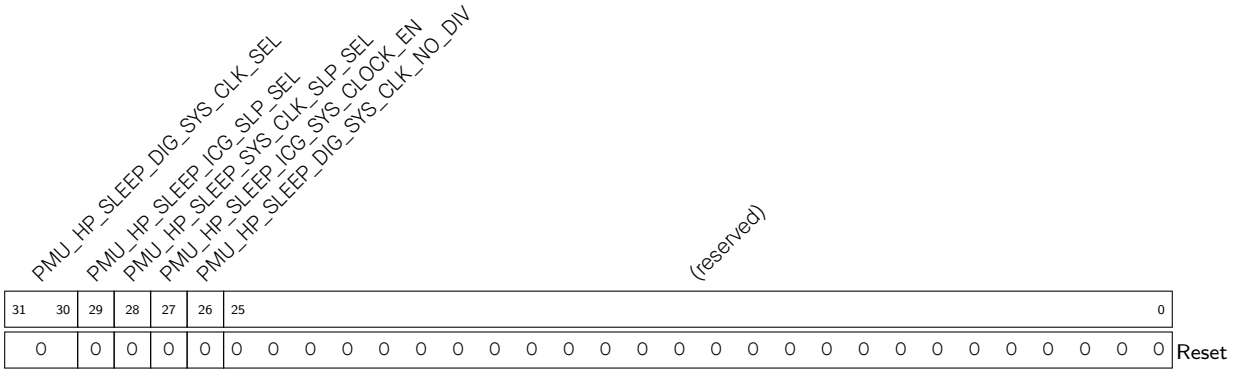
**PMU_HP_ACTIVE2SLEEP_BACKUP_EN** Configures whether to enable the backup flow when PMU state switches from HP_ACTIVE to HP_SLEEP.
0: Disable backup
1: Enable backup
(R/W)

Register 12.35. PMU_HP_SLEEP_BACKUP_CLK_REG (0x0088)




**PMU_HP_SLEEP_BACKUP_ICG_FUNC_EN** Configures whether to enable each peripherals' function clock when the target state is HP_SLEEP. For details, please refer to Chapter 8 *Reset and Clock*.
0: Disable
1: Enable
(R/W)

Register 12.36. PMU_HP_SLEEP_SYSCLK_REG (0x008C)




**PMU_HP_SLEEP_DIG_SYS_CLK_NO_DIV** Configures whether to enable the clock division for HP_ROOT_CLK in HP_SLEEP state. (R/W)

**PMU_HP_SLEEP_ICG_SYS_CLOCK_EN** Configures whether to enable HP_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 HP_ROOT_CLK in HP_SLEEP state.

0: XTAL

1: PLL_CLK

2: RC_FAST_CLK

3: Invalid value

(R/W)

Register 12.37. PMU_HP_SLEEP_HP_REGULATOR0_REG (0x0090)




**PMU_HP_SLEEP_HP_REGULATOR_SLP_MEM_XPD** Configures whether to enable the secondary HP regulator powering the HP memory in HP_SLEEP state.

0: Disable

1: Enable

(R/W)

**PMU_HP_SLEEP_HP_REGULATOR_SLP_LOGIC_XPD** Configures whether to enable the secondary HP regulator powering the logic circuits in HP domain in HP_SLEEP state.

0: Disable

1: Enable

(R/W)

**PMU_HP_SLEEP_HP_REGULATOR_XPD** Configures whether to enable the main HP regulator in HP_SLEEP state.

0: Disable

1: Enable

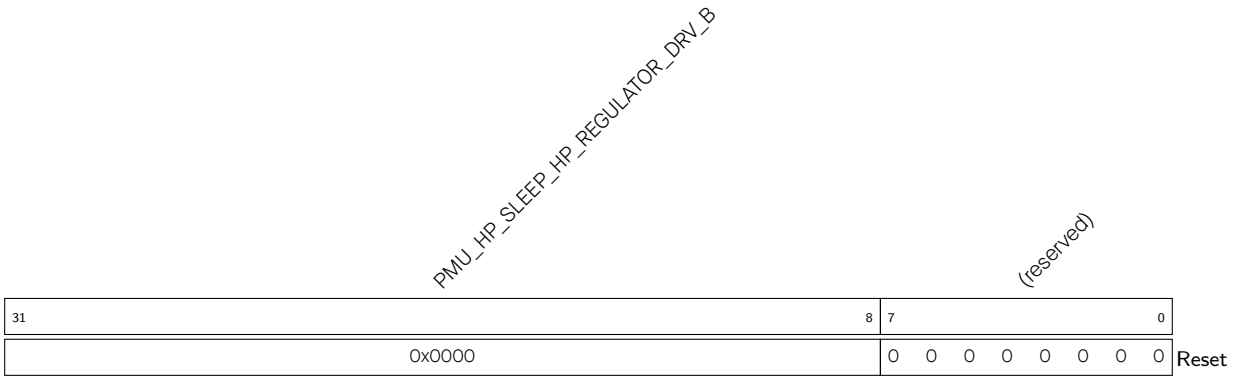
(R/W)

**PMU_HP_SLEEP_HP_REGULATOR_SLP_MEM_DBIAS** Configures the DREG value of the secondary HP regulator powering the HP memory in HP_SLEEP state, adjusting its output voltage. (R/W)

**PMU_HP_SLEEP_HP_REGULATOR_SLP_LOGIC_DBIAS** Configures the DREG value of the secondary HP regulator powering the logic circuits in HP domain in HP_SLEEP state, adjusting its output voltage. (R/W)

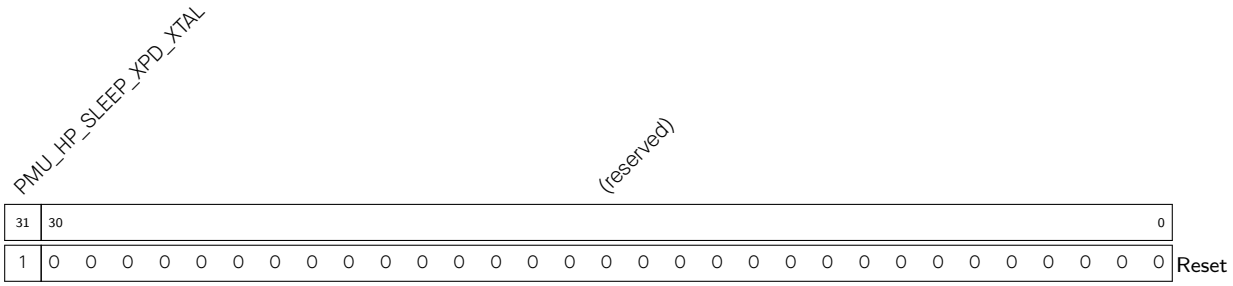
**PMU_HP_SLEEP_HP_REGULATOR_DBIAS** Configures the DREG value of the main HP regulator in HP_SLEEP state, adjusting its output voltage. (R/W)

Register 12.38. PMU_HP_SLEEP_HP_REGULATOR1_REG (0x0094)




**PMU_HP_SLEEP_HP_REGULATOR_DRV_B** Configures the drive strength of the main HP regulator in HP_SLEEP state. (R/W)

Register 12.39. PMU_HP_SLEEP_XTAL_REG (0x0098)



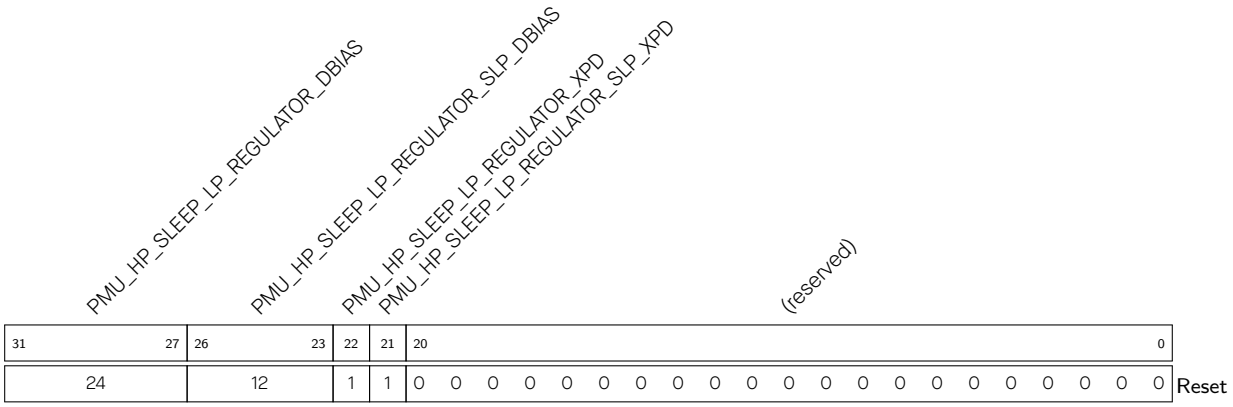

**PMU_HP_SLEEP_XPD_XTAL** Configures whether to enable XTAL_CLK in HP_SLEEP state.

0: Disable

1: Enable

(R/W)

Register 12.40. PMU_HP_SLEEP_LP_REGULATOR0_REG (0x009C)




**PMU_HP_SLEEP_LP_REGULATOR_SLP_XPD** Configures whether to enable the secondary LP regulator in HP_SLEEP state.

0: Disable

1: Enable

(R/W)

**PMU_HP_SLEEP_LP_REGULATOR_XPD** Configures whether to enable the main LP regulator in HP_SLEEP state.

0: Disable

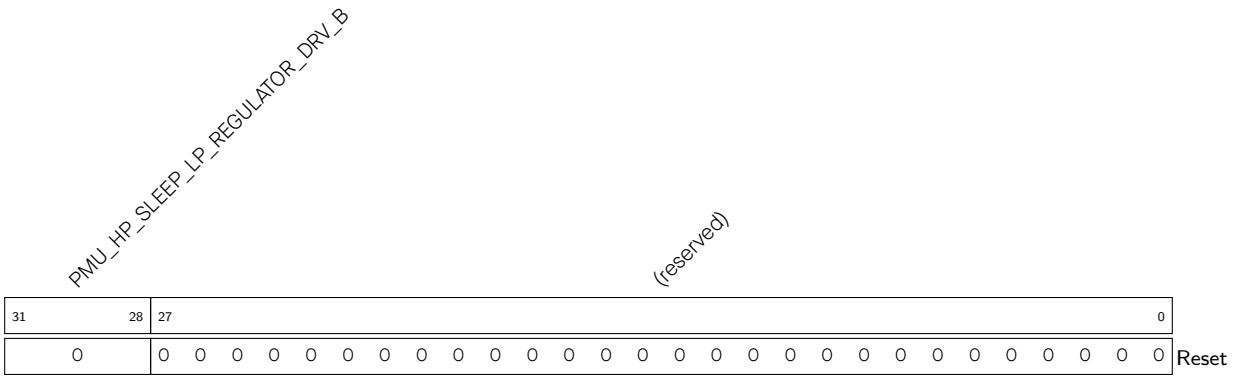
1: Enable

(R/W)

**PMU_HP_SLEEP_LP_REGULATOR_SLP_DBIAS** Configures the DREG value of the secondary LP regulator in HP_SLEEP state, adjusting its output voltage. (R/W)

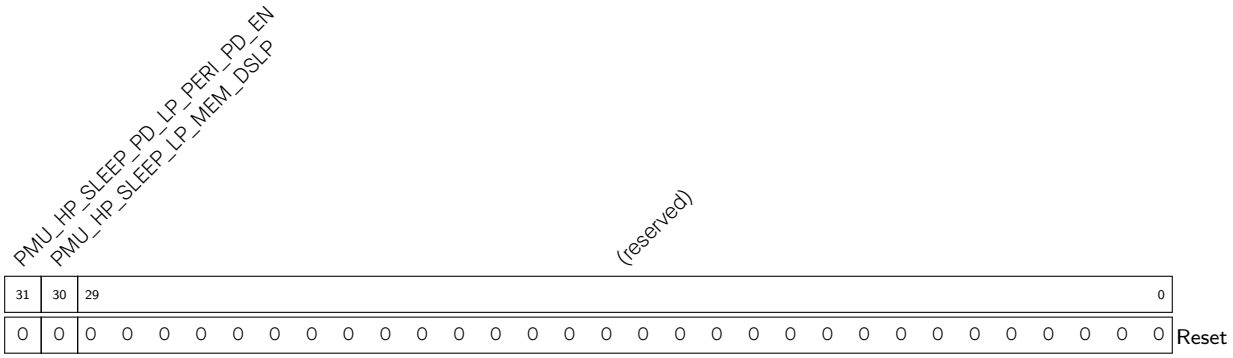
**PMU_HP_SLEEP_LP_REGULATOR_DBIAS** Configures the DREG value of the main LP regulator in HP_SLEEP state, adjusting its output voltage. (R/W)

Register 12.41. PMU_HP_SLEEP_LP_REGULATOR1_REG (0x00A0)




**PMU_HP_SLEEP_LP_REGULATOR_DRV_B** Configures the drive strength of the main LP regulator in HP_SLEEP state. (R/W)

Register 12.42. PMU_HP_SLEEP_LP_DIG_POWER_REG (0x00A8)




**PMU_HP_SLEEP_LP_MEM_DSLP** Configures whether to enable Memory Deep-sleep mode for the 320 KB SRAM in HP_ACTIVE, HP_MODEM and HP_SLEEP states.

0: Disable

1: Enable

(R/W)

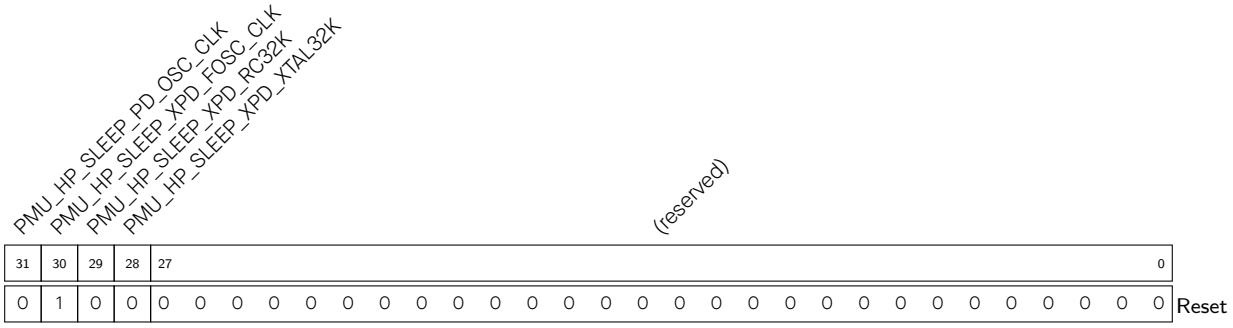
**PMU_HP_SLEEP_PD_LP_PERI_PD_EN** Configures whether to power down "Peripherals+ROM" domain in HP_ACTIVE, HP_MODEM and HP_SLEEP states.

0: Power up

1: Power down

(R/W)

Register 12.43. PMU_HP_SLEEP_LP_CK_POWER_REG (0x00AC)




**PMU_HP_SLEEP_XPD_XTAL32K** Configures whether to power up XTAL32K_CLK in HP_ACTIVE, HP_MODEM and HP_SLEEP states.

0: Power down

1: Power up

(R/W)

**PMU_HP_SLEEP_XPD_RC32K** Configures whether to power up RC32K_CLK in HP_ACTIVE, HP_MODEM and HP_SLEEP states.

0: Power down

1: Power up

(R/W)

**PMU_HP_SLEEP_XPD_FOSC_CLK** Configures whether to power up RC_FAST_CLK in HP_ACTIVE, HP_MODEM and HP_SLEEP states.

0: Power down

1: Power up

(R/W)

**PMU_HP_SLEEP_PD_OSC_CLK** Configures whether to power up SOSC_CLK in HP_ACTIVE, HP_MODEM and HP_SLEEP states.

0: Power down

1: Power up

(R/W)

**Register 12.44. PMU_LP_SLEEP_LP_REGULATOR0_REG (0x00B4)**

| Field | PMU_LP_SLEEP_LP_REGULATOR_DBIAS | PMU_LP_SLEEP_LP_REGULATOR_SLP_DBIAS | PMU_LP_SLEEP_LP_REGULATOR_XPD | PMU_LP_SLEEP_LP_REGULATOR_SLP_XPD | (reserved) | |
|---|---|---|---|---|---|---|
| Bits | 31 27 | 26 23 | 22 | 21 | 20 0 | |
| | 24 | 12 | 1 | 1 | 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | Reset |

**PMU_LP_SLEEP_LP_REGULATOR_SLP_XPD** Configures whether to enable the secondary LP regulator in LP_SLEEP state.
0: Disable
1: Enable
(R/W)

**PMU_LP_SLEEP_LP_REGULATOR_XPD** Configures whether to enable the main LP regulator in LP_SLEEP state.
0: Disable
1: Enable
(R/W)

**PMU_LP_SLEEP_LP_REGULATOR_SLP_DBIAS** Configures the DREG value of the secondary LP regulator in LP_SLEEP state, adjusting its output voltage. (R/W)

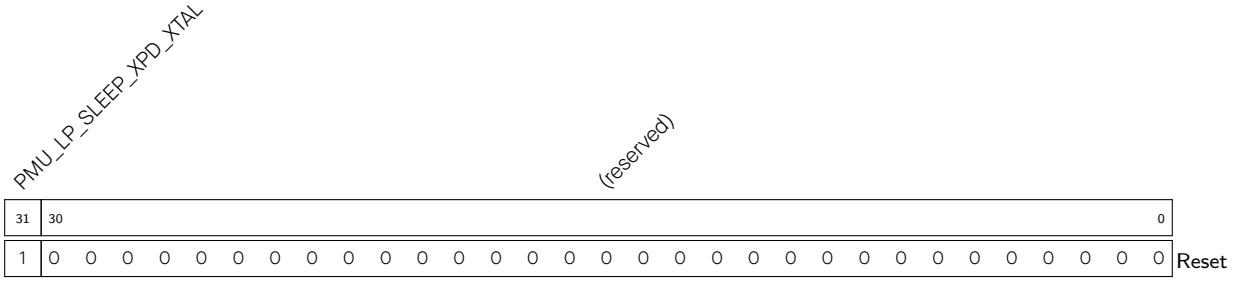
**PMU_LP_SLEEP_LP_REGULATOR_DBIAS** Configures the DREG value of the main LP regulator in LP_SLEEP state, adjusting its output voltage. (R/W)

**Register 12.45. PMU_LP_SLEEP_LP_REGULATOR1_REG (0x00B8)**

| Field | PMU_LP_SLEEP_LP_REGULATOR_DRV_B | (reserved) | |
|---|---|---|---|
| Bits | 31 28 | 27 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 |

**PMU_LP_SLEEP_LP_REGULATOR_DRV_B** Configures the drive strength of the main LP regulator in LP_SLEEP state. (R/W)

Register 12.46. PMU_LP_SLEEP_XTAL_REG (0x00BC)



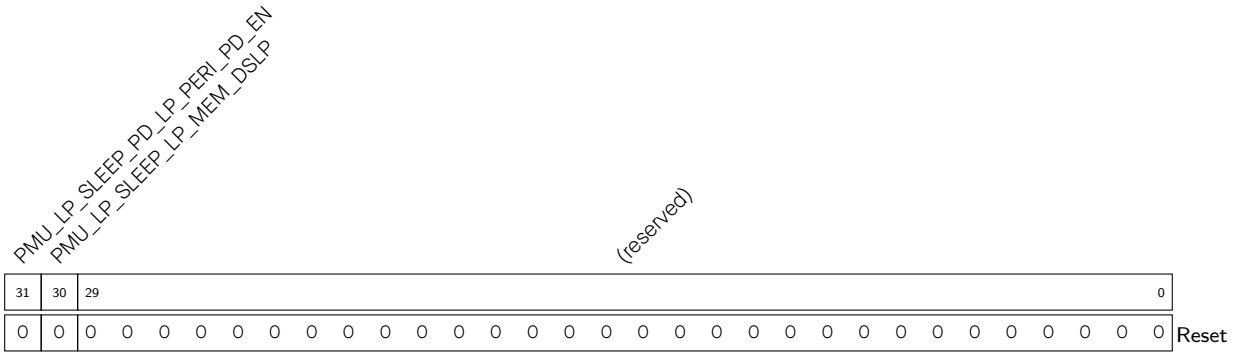

**PMU_LP_SLEEP_XPD_XTAL** Configures whether to enable XTAL_CLK in LP_SLEEP state.

0: Disable

1: Enable

(R/W)

Register 12.47. PMU_LP_SLEEP_LP_DIG_POWER_REG (0x00C0)




**PMU_LP_SLEEP_LP_MEM_DSLP** Configures whether to enable Memory Deep-sleep mode for the 320 KB SRAM in LP_SLEEP state.

0: Disable

1: Enable

(R/W)

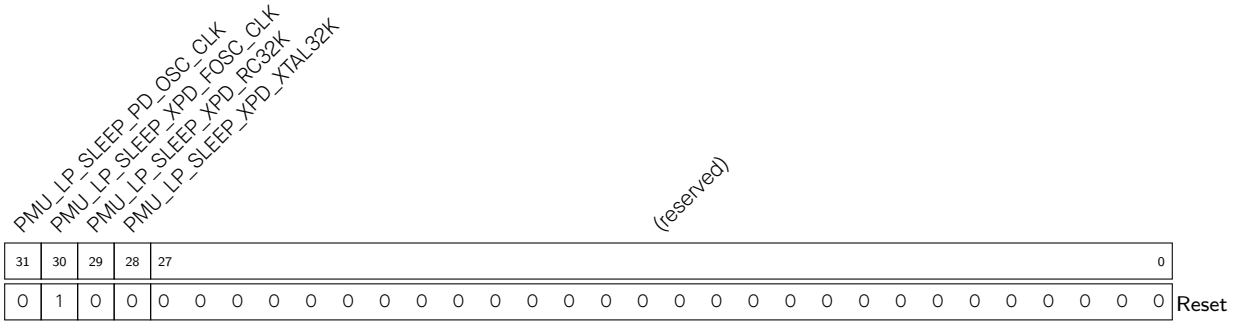
**PMU_LP_SLEEP_PD_LP_PERI_PD_EN** Configures whether to power down the "Peripherals+ROM" domain in LP_SLEEP state.

0: Power up

1: Power down

(R/W)

**Register 12.48. PMU_LP_SLEEP_LP_CK_POWER_REG (0x00C4)**




**PMU_LP_SLEEP_XPD_XTAL32K** Configures whether to power up XTAL32K_CLK in LP_SLEEP state.

0: Power down

1: Power up

(R/W)

**PMU_LP_SLEEP_XPD_RC32K** Configures whether to power up RC32K_CLK in LP_SLEEP state.

0: Power down

1: Power up

(R/W)

**PMU_LP_SLEEP_XPD_FOSC_CLK** Configures whether to power up RC_FAST_CLK in LP_SLEEP state.

0: Power down

1: Power up

(R/W)

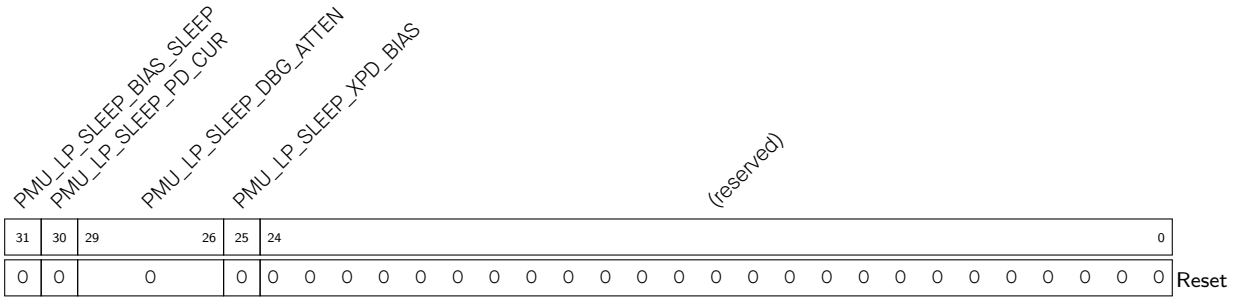
**PMU_LP_SLEEP_PD_OSC_CLK** Configures whether to power up SOSC_CLK in LP_SLEEP state.

0: Power down

1: Power up

(R/W)

**Register 12.49. PMU_LP_SLEEP_BIAS_REG (0x00C8)**



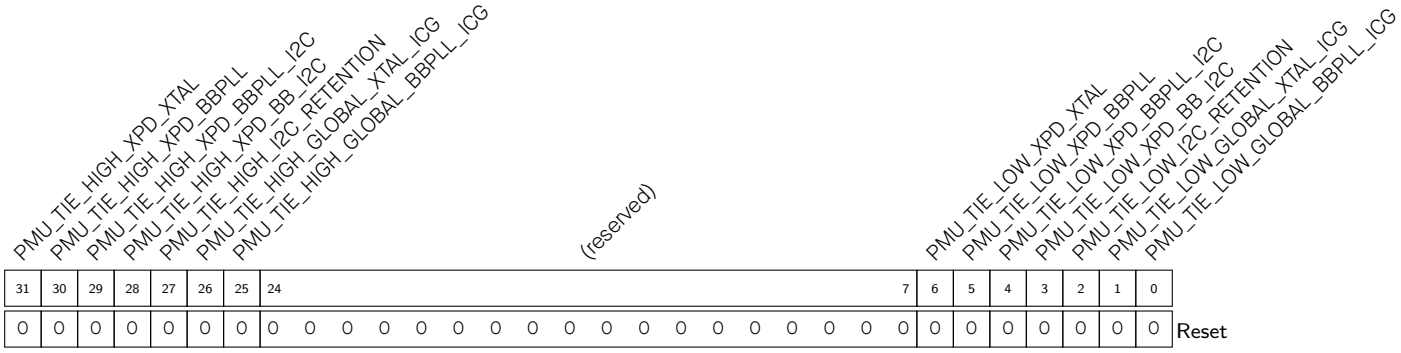

**PMU_LP_SLEEP_XPD_BIAS** Configures the power supply of BIAS in LP_SLEEP state. (R/W)

**PMU_LP_SLEEP_DBG_ATTEN** Configures the degree of attenuation of the analog band gap in LP_SLEEP state. (R/W)

**PMU_LP_SLEEP_PD_CUR** Configures the power-down current in LP_SLEEP state. (R/W)

**PMU_LP_SLEEP_BIAS_SLEEP** Configures the sleep or wakeup status of the BIAS circuit in LP_SLEEP state. (R/W)

Register 12.50. PMU_IMM_HP_CK_POWER_REG (0x00CC)




**PMU_TIE_LOW_GLOBAL_BBPLL_ICG** Configures whether to force disable the integrated clock gating (ICG) of PLL clock

0: No effect

1: Force disable

(WT)

**PMU_TIE_LOW_GLOBAL_XTAL_ICG** Configures whether to force disable the integrated clock gating (ICG) of XTAL clock.

0: No effect

1: Force disable

(WT)

**PMU_TIE_LOW_I2C_RETENTION** Configures whether to force the power-on reset (POR) signal for the analog I2C to low, releasing the reset.

0: No effect

1: Force disable

(WT)

**PMU_TIE_LOW_XPD_BB_I2C** Configures whether to force the power-on reset (POR) signal for BB_I2C to low, releasing the reset.

0: No effect

1: Force disable

(WT)

**PMU_TIE_LOW_XPD_BBPLL_I2C** Configures whether to force the power-on reset (POR) signal for BBPLL_I2C to low, releasing the reset.

0: No effect

1: Force disable

(WT)

**PMU_TIE_LOW_XPD_BBPLL** Configures whether to force the power-on reset (POR) signal for BBPLL to low, releasing the reset.

0: No effect

1: Force disable

(WT)

**Continued on the next page...**

Register 12.50. PMU_IMM_HP_CK_POWER_REG (0x00CC)

Continued from the previous page...

**PMU_TIE_LOW_XPD_XTAL** Configures whether to force the power-on reset (POR) signal for XTAL_CLK to low, releasing the reset.
0: No effect
1: Force disable
(WT)

**PMU_TIE_HIGH_GLOBAL_BBPLL_ICG** Configures whether to force enable the integrated clock gating (ICG) of PLL clock.
0: No effect
1: Force enable
(WT)

**PMU_TIE_HIGH_GLOBAL_XTAL_ICG** Configures whether to force enable the integrated clock gating (ICG) of XTAL clock.
0: No effect
1: Force enable
(WT)

**PMU_TIE_HIGH_I2C_RETENTION** Configures whether to force the power-on reset (POR) signal for the analog I2C to high, putting it into reset.
0: No effect
1: Force enable
(WT)

**PMU_TIE_HIGH_XPD_BB_I2C** Configures whether to force the power-on reset (POR) signal for BB_I2C to high, putting it into reset.
0: No effect
1: Force enable
(WT)

**PMU_TIE_HIGH_XPD_BBPLL_I2C** Configures whether to force the power-on reset (POR) signal for BBPLL_I2C to high, putting it into reset.
0: No effect
1: Force enable
(WT)

**PMU_TIE_HIGH_XPD_BBPLL** Configures whether to force the power-on reset (POR) signal for BBPLL to high, putting it into reset.
0: No effect
1: Force enable
(WT)

Continued on the next page...

Register 12.50. PMU_IMM_HP_CK_POWER_REG (0x00CC)

Continued from the previous page...

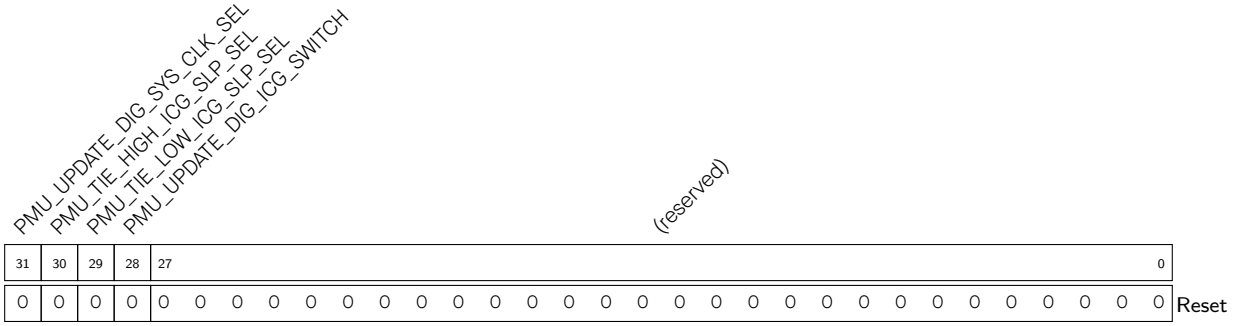
**PMU_TIE_HIGH_XPD_XTAL** Configures whether to force the power-on reset (POR) signal for XTAL_CLK to high, putting it into reset.

0: No effect

1: Force enable

(WT)

Register 12.51. PMU_IMM_SLEEP_SYSCLK_REG (0x00D0)




**PMU_UPDATE_DIG_ICG_SWITCH** Configures whether to force power up XTAL.

0: No effect

1: Force enable

(WT)

**PMU_TIE_LOW_ICG_SLP_SEL** Configures whether to force disable the integrated clock gating (ICG) of PMU clock controller.

0: No effect

1: Force disable

(WT)

**PMU_TIE_HIGH_ICG_SLP_SEL** Configures whether to force enable the integrated clock gating (ICG) of PMU clock controller.

0: No effect

1: Force enable

(WT)

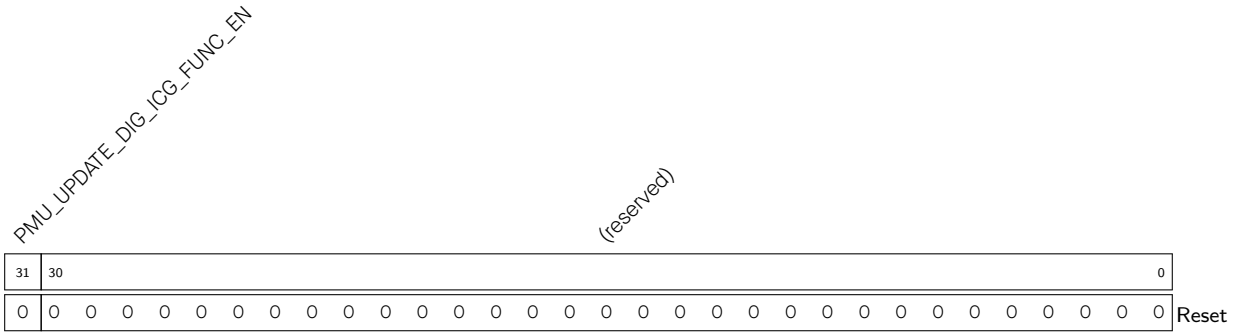
**PMU_UPDATE_DIG_SYS_CLK_SEL** Configures whether to force update the HP_ROOT_CLK configuration stored in PMU registers.

0: No effect

1: Force update

(WT)

Register 12.52. PMU_IMM_HP_FUNC_ICG_REG (0x00D4)



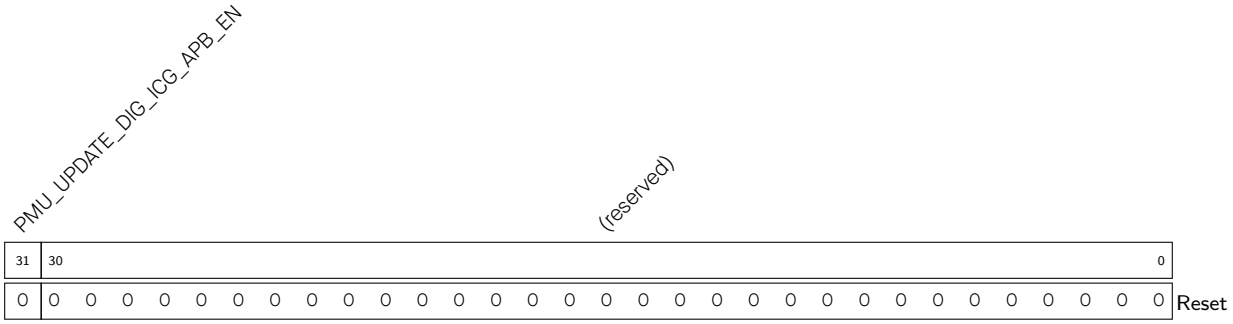

**PMU_UPDATE_DIG_ICG_FUNC_EN** Configure whether to force the PMU to update its ICG enable signals of HP system peripherals' function clocks based on the current PMU state.

0: No effect

1: Force update

(WT)

Register 12.53. PMU_IMM_HP_APB_ICG_REG (0x00D8)



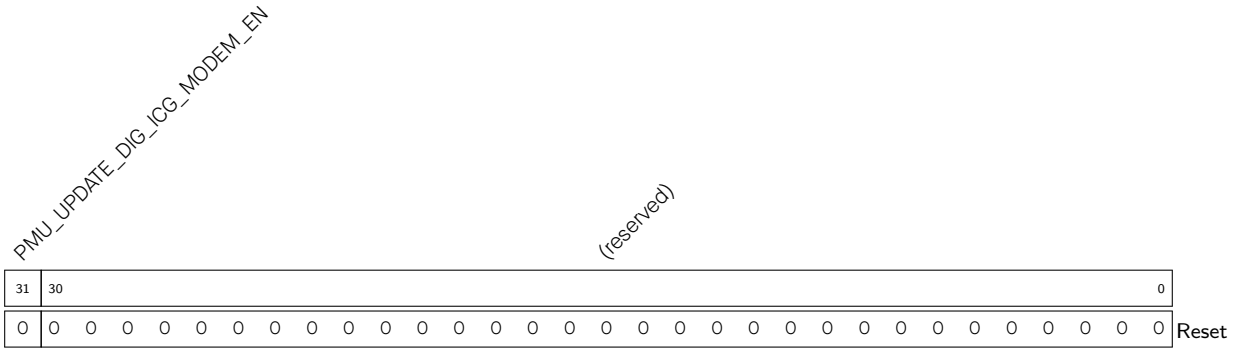

**PMU_UPDATE_DIG_ICG_APB_EN** Configure whether to force the PMU to update its ICG enable signals of HP system peripherals' APB clock based on the current PMU state.

0: No effect

1: Force update

(WT)

Register 12.54. PMU_IMM_MODEM_ICG_REG (0x00DC)



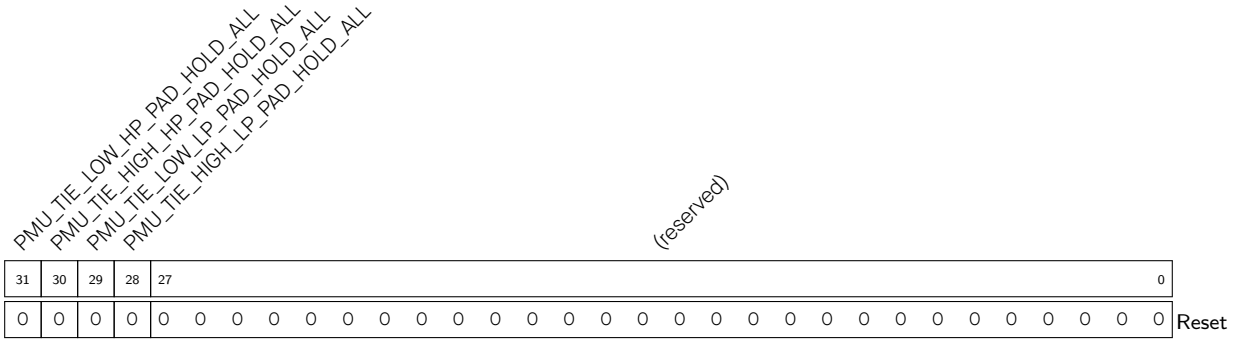

**PMU_UPDATE_DIG_ICG_MODEM_EN** Configure whether to force the PMU to update its ICG enable signals its Modem clock based on the current PMU state.

0: No effect

1: Force update

(WT)

Register 12.55. PMU_IMM_PAD_HOLD_ALL_REG (0x00E4)




**PMU_TIE_HIGH_LP_PAD_HOLD_ALL** Configures whether to force the PMU LP GPIO HOLD signal to high.

0: No effect

1: Force to high

(WT)

**PMU_TIE_LOW_LP_PAD_HOLD_ALL** Configures whether to force the PMU LP GPIO HOLD signal to low.

0: No effect

1: Force to low

(WT)

**PMU_TIE_HIGH_HP_PAD_HOLD_ALL** Configures whether to force the PMU HP GPIO HOLD signal to high.

0: No effect

1: Force to high

(WT)

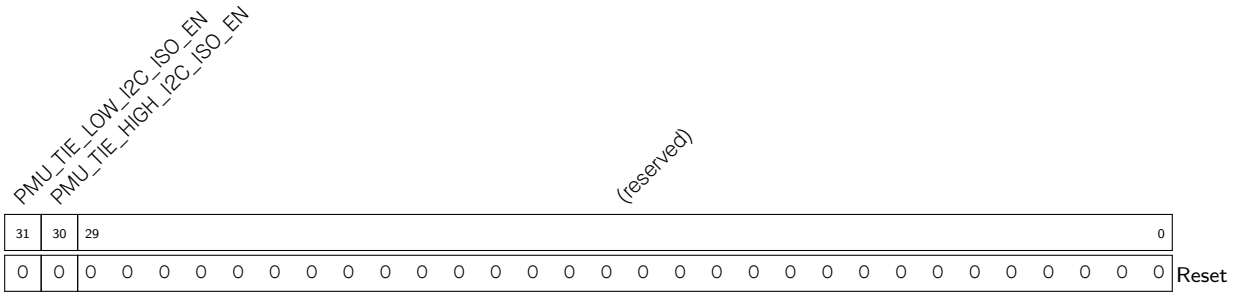
**PMU_TIE_LOW_HP_PAD_HOLD_ALL** Configures whether to force the PMU HP GPIO HOLD signal to low.

0: No effect

1: Force to low

(WT)

Register 12.56. PMU_IMM_I2C_ISO_REG (0x00E8)




**PMU_TIE_HIGH_I2C_ISO_EN** Configures whether to force the PMU I2C_ISO enabling signal to high.

0: No effect

1: Force to high

(WT)

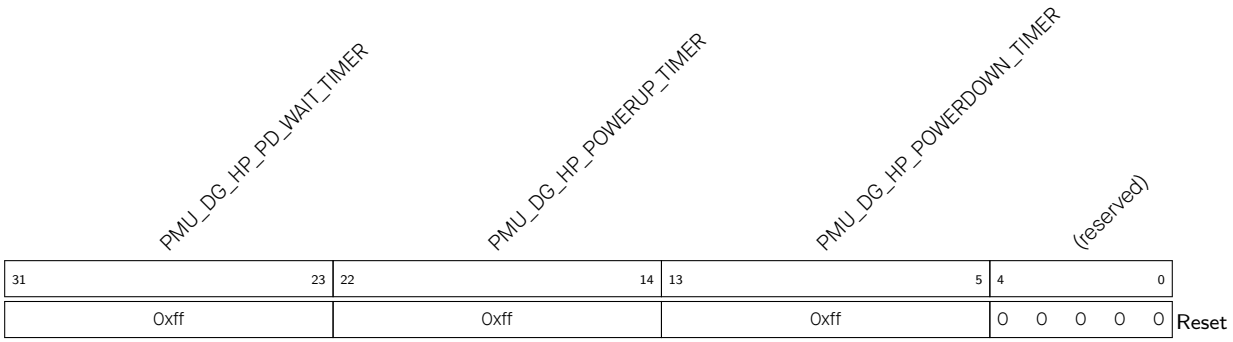
**PMU_TIE_LOW_I2C_ISO_EN** Configures whether to force the PMU I2C_ISO enabling signal to low.

0: No effect

1: Force to low

(WT)

Register 12.57. PMU_POWER_WAIT_TIMER0_REG (0x00EC)

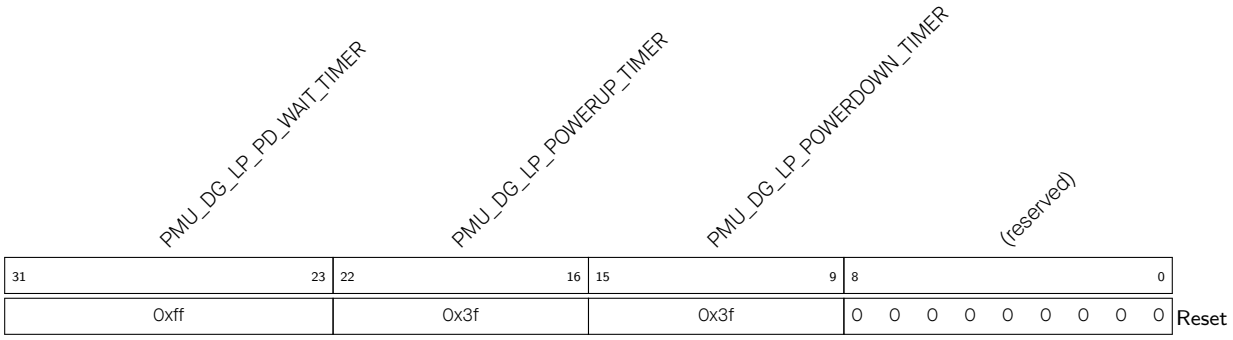



**PMU_DG_HP_POWERDOWN_TIMER** Configures the wait cycles before powering down HP power domains. (R/W)

**PMU_DG_HP_POWERUP_TIMER** Configures the wait cycles before waking up HP power domains. (R/W)

**PMU_DG_HP_PD_WAIT_TIMER** Configures the wait cycles during sleep of HP power domains. (R/W)

Register 12.58. PMU_POWER_WAIT_TIMER1_REG (0x00F0)

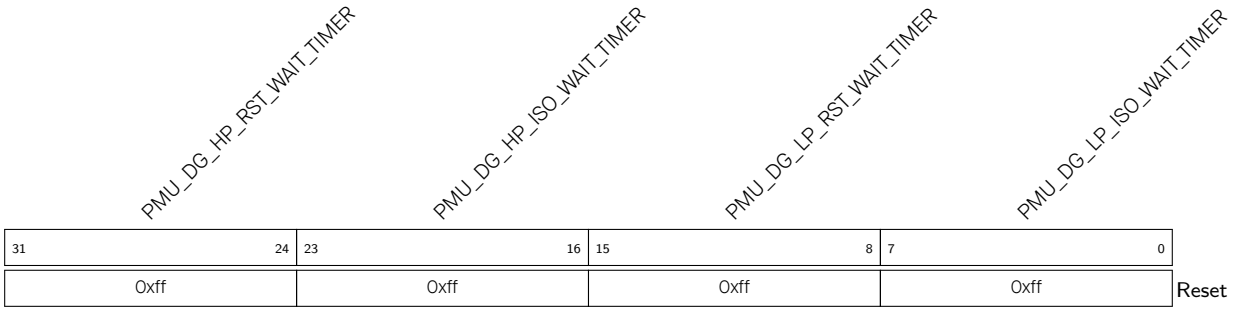



**PMU_DG_LP_POWERDOWN_TIMER** Configures the wait cycles before powering down LP power domains. (R/W)

**PMU_DG_LP_POWERUP_TIMER** Configures the wait cycles before waking up LP power domains. (R/W)

**PMU_DG_LP_PD_WAIT_TIMER** Configures the wait cycles during sleep of LP power domains. (R/W)

Register 12.59. PMU_POWER_WAIT_TIMER2_REG (0x00F4)



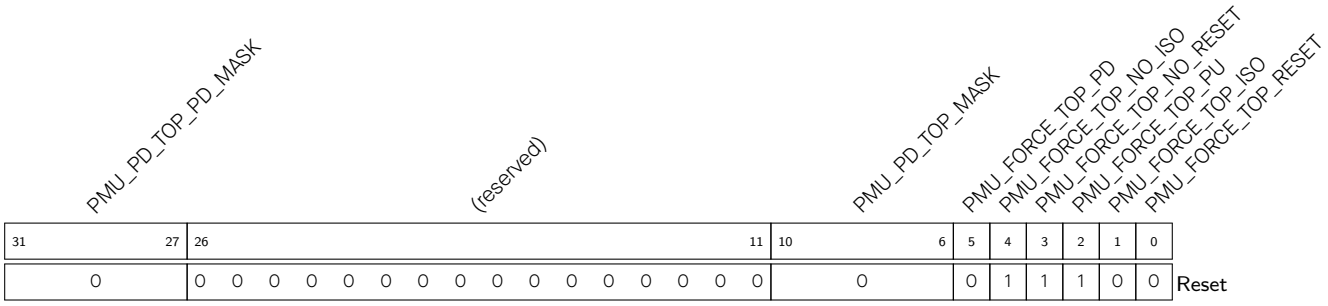

**PMU_DG_LP_ISO_WAIT_TIMER** Configures the wait cycles of LP ISO control. (R/W)

**PMU_DG_LP_RST_WAIT_TIMER** Configures the wait cycles of LP reset control. (R/W)

**PMU_DG_HP_ISO_WAIT_TIMER** Configures the wait cycles of HP ISO control. (R/W)

**PMU_DG_HP_RST_WAIT_TIMER** Configures the wait cycles of HP reset control. (R/W)

Register 12.60. PMU_POWER_PD_TOP_CNTL_REG (0x00F8)




**PMU_FORCE_TOP_RESET** Configures whether or not to force reset "Peripherals+ROM" domain.

0: No effect

1: Force reset

(R/W)

**PMU_FORCE_TOP_ISO** Configures whether or not to enable the force isolation of "Peripherals+ROM" domain. 0: No effect

1: Enable

(R/W)

**PMU_FORCE_TOP_PU** Configures whether or not to force power up "Peripherals+ROM" domain. This setting has a lower priority than PMU_PD_TOP_MASK.

0: No effect

1: Force power up

(R/W)

**PMU_FORCE_TOP_NO_RESET** Configures whether or not to forcefully prevent the reset of "Peripherals+ROM" domain. This setting has a lower priority than PMU_FORCE_TOP_RESET.

0: No effect

1: Force not reset

(R/W)

**PMU_FORCE_TOP_NO_ISO** Configures whether or not to disable the force isolation of "Peripherals+ROM" domain. This setting has a lower priority than PMU_FORCE_TOP_ISO.

0: No effect

1: Disable

(R/W)

**PMU_FORCE_TOP_PD** Configures whether or not to force power down "Peripherals+ROM" domain. This setting has a lower priority than PMU_FORCE_TOP_PU.

0: No effect

1: Force power down

(R/W)

Continued on the next page...

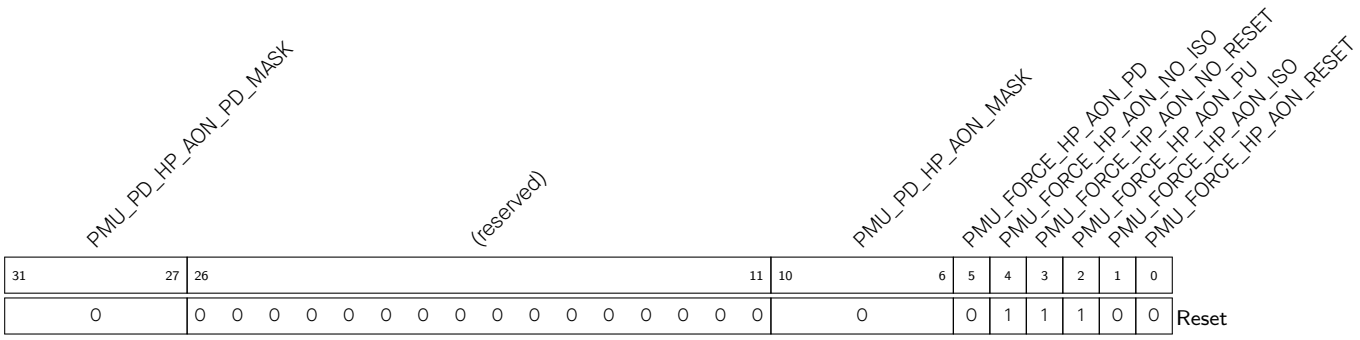
Register 12.60. PMU_POWER_PD_TOP_CNTL_REG (0x00F8)

Continued from the previous page...

**PMU_PD_TOP_MASK** Configures whether or not to force power up "Peripherals+ROM" domain, regardless the signals from PMU.
0: No effect
1: Force power up
(R/W)

**PMU_PD_TOP_PD_MASK** Configures whether or not to force power down "Peripherals+ROM" domain, regardless the signals from PMU. This setting has a lower priority than PMU_PD_TOP_MASK.
0: No effect
1: Force power up
(R/W)

Register 12.61. PMU_POWER_PD_HPAON_CNTL_REG (0x00FC)




**PMU_FORCE_HP_AON_RESET** Configures whether or not to force reset Modem Power domain.

0: No effect

1: Force reset

(R/W)

**PMU_FORCE_HP_AON_ISO** Configures whether or not to enable the force isolation of Modem Power domain.

0: No effect

1: Enable

(R/W)

**PMU_FORCE_HP_AON_PU** Configures whether or not to force power up Modem Power domain. This setting has a lower priority than PMU_PD_HP_AON_MASK.

0: No effect

1: Force power up

(R/W)

**PMU_FORCE_HP_AON_NO_RESET** Configures whether or not to forcefully prevent the reset of Modem Power domain. This setting has a lower priority than PMU_FORCE_HP_AON_RESET.

0: No effect

1: Force not reset

(R/W)

**PMU_FORCE_HP_AON_NO_ISO** Configures whether or not to disable the force isolation of Modem Power domain. This setting has a lower priority than PMU_FORCE_HP_AON_ISO.

0: No effect

1: Disable

(R/W)

**PMU_FORCE_HP_AON_PD** Configures whether or not to force power down Modem Power domain. This setting has a lower priority than PMU_FORCE_HP_AON_PU.

0: No effect

1: Force power down

(R/W)

**Continued on the next page...**

Register 12.61. PMU_POWER_PD_HPAON_CNTL_REG (0x00FC)

Continued from the previous page...

**PMU_PD_HP_AON_MASK** Configures whether or not to force power up Modem Power domain, regardless the signals from PMU.

0: No effect

1: Force power up

(R/W)

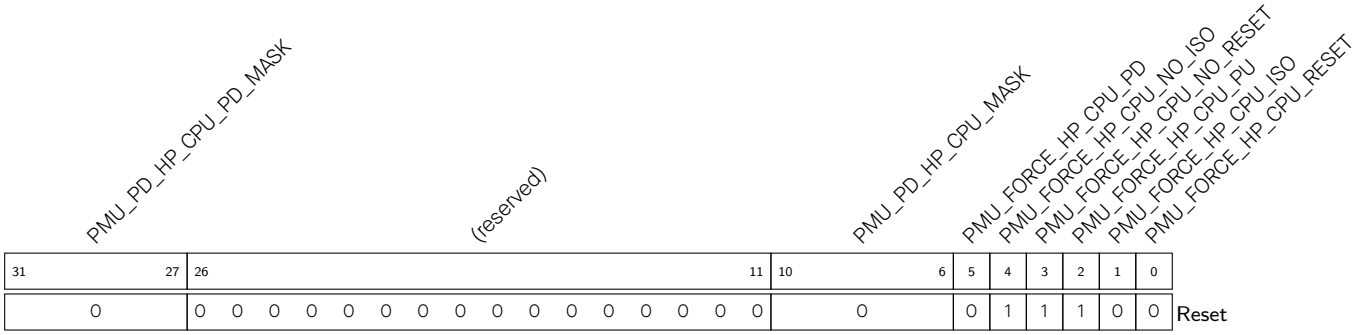
**PMU_PD_HP_AON_PD_MASK** Configures whether or not to force power down Modem Power domain, regardless the signals from PMU. This setting has a lower priority than PMU_PD_HP_AON_MASK.

0: No effect

1: Force power up

(R/W)

Register 12.62. PMU_POWER_PD_HPCPU_CNTL_REG (0x0100)




**PMU_FORCE_HP_CPU_RESET** Configures whether or not to force reset CPU domain.

0: No effect

1: Force reset

(R/W)

**PMU_FORCE_HP_CPU_ISO** Configures whether or not to enable the force isolation of CPU domain.

0: No effect

1: Enable

(R/W)

**PMU_FORCE_HP_CPU_PU** Configures whether or not to force power up CPU domain. This setting has a lower priority than PMU_PD_HP_CPU_MASK.

0: No effect

1: Force power up

(R/W)

Continued on the next page...

Register 12.62. PMU_POWER_PD_HPCPU_CNTL_REG (0x0100)

Continued from the previous page...

**PMU_FORCE_HP_CPU_NO_RESET** Configures whether or not to forcefully prevent the reset of CPU domain. This setting has a lower priority than PMU_FORCE_HP_CPU_RESET.
0: No effect
1: Force not reset
(R/W)

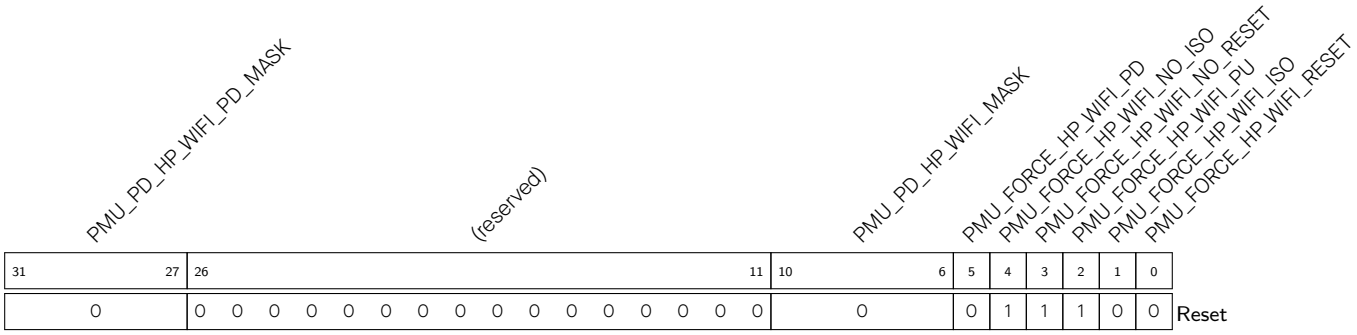
**PMU_FORCE_HP_CPU_NO_ISO** Configures whether or not to disable the force isolation of CPU domain. This setting has a lower priority than PMU_FORCE_HP_CPU_ISO.
0: No effect
1: Disable
(R/W)

**PMU_FORCE_HP_CPU_PD** Configures whether or not to force power down CPU domain. This setting has a lower priority than PMU_FORCE_HP_CPU_PU.
0: No effect
1: Force power down
(R/W)

**PMU_PD_HP_CPU_MASK** Configures whether or not to force power up CPU domain, regardless the signals from PMU.
0: No effect
1: Force power up
(R/W)

**PMU_PD_HP_CPU_PD_MASK** Configures whether or not to force power down CPU domain, regardless the signals from PMU. This setting has a lower priority than PMU_PD_HP_CPU_MASK.
0: No effect
1: Force power up
(R/W)

Register 12.63. PMU_POWER_PD_HPWIFI_CNTL_REG (0x0108)




**PMU_FORCE_HP_WIFI_RESET** Configures whether or not to force reset MODEM domain.

0: No effect

1: Force reset

(R/W)

**PMU_FORCE_HP_WIFI_ISO** Configures whether or not to enable the force isolation of MODEM domain.

0: No effect

1: Enable

(R/W)

**PMU_FORCE_HP_WIFI_PU** Configures whether or not to force power up MODEM domain. This setting has a lower priority than PMU_PD_HP_WIFI_MASK.

0: No effect

1: Force power up

(R/W)

**PMU_FORCE_HP_WIFI_NO_RESET** Configures whether or not to forcefully prevent the reset of MODEM domain. This setting has a lower priority than PMU_FORCE_HP_WIFI_RESET.

0: No effect

1: Force not reset

(R/W)

**PMU_FORCE_HP_WIFI_NO_ISO** Configures whether or not to disable the force isolation of MODEM domain. This setting has a lower priority than PMU_FORCE_HP_WIFI_ISO.

0: No effect

1: Disable

(R/W)

**PMU_FORCE_HP_WIFI_PD** Configures whether or not to force power down MODEM domain. This setting has a lower priority than PMU_FORCE_HP_WIFI_PU.

0: No effect

1: Force power down

(R/W)

**Continued on the next page...**

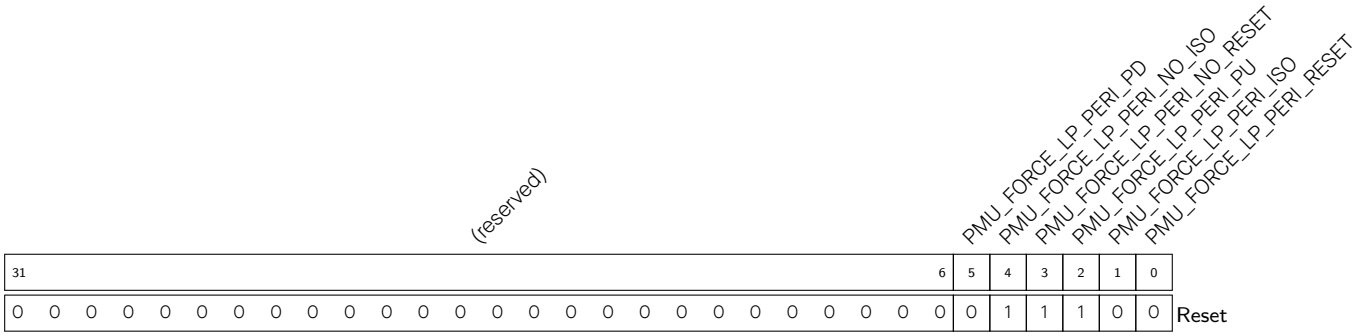
Register 12.63. PMU_POWER_PD_HPWIFI_CNTL_REG (0x0108)

Continued from the previous page...

**PMU_PD_HP_WIFI_MASK** Configures whether or not to force power up MODEM domain, regardless the signals from PMU.
0: No effect
1: Force power up
(R/W)

**PMU_PD_HP_WIFI_PD_MASK** Configures whether or not to force power down MODEM domain, regardless the signals from PMU. This setting has a lower priority than PMU_PD_HP_WIFI_MASK.
0: No effect
1: Force power up
(R/W)

**Register 12.64. PMU_POWER_PD_LPPERI_CNTL_REG (0x010C)**




**PMU_FORCE_LP_PERI_RESET** Configures whether or not to force reset LPSYS_OFF domain.

0: No effect

1: Force reset

(R/W)

**PMU_FORCE_LP_PERI_ISO** Configures whether or not to enable the force isolation of LPSYS_OFF domain.

0: No effect

1: Enable

(R/W)

**PMU_FORCE_LP_PERI_PU** Configures whether or not to force power up LPSYS_OFF domain.

0: No effect

1: Force power up

(R/W)

**PMU_FORCE_LP_PERI_NO_RESET** Configures whether or not to forcefully prevent the reset of LPSYS_OFF domain. This setting has a lower priority than PMU_FORCE_LP_PERI_RESET.

0: No effect

1: Force not reset

(R/W)

**PMU_FORCE_LP_PERI_NO_ISO** Configures whether or not to disable the force isolation of LPSYS_OFF domain. This setting has a lower priority than PMU_FORCE_LP_PERI_ISO.

0: No effect

1: Disable

(R/W)

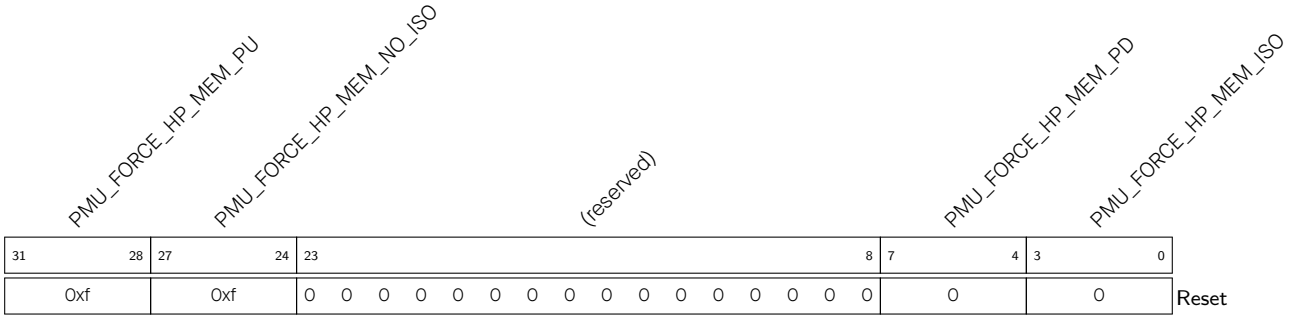
**PMU_FORCE_LP_PERI_PD** Configures whether or not to force power down LPSYS_OFF domain. This setting has a lower priority than PMU_FORCE_LP_PERI_PU.

0: No effect

1: Force power down

(R/W)

Register 12.65. PMU_POWER_PD_MEM_CNTL_REG (0x0110)




**PMU_FORCE_HP_MEM_ISO** Configures whether or not to enable the force isolation of Internal SRAMx domain.

0: No effect

1: Enable

(R/W)

**PMU_FORCE_HP_MEM_PD** Configures whether or not to force power down Internal SRAMx domain. This setting has a lower priority than PMU_FORCE_HP_MEM_PU.

0: No effect

1: Force power down

(R/W)

**PMU_FORCE_HP_MEM_NO_ISO** Configures whether or not to disable the force isolation of Internal SRAMx domain. This setting has a lower priority than PMU_FORCE_HP_MEM_ISO.

0: No effect

1: Disable

(R/W)

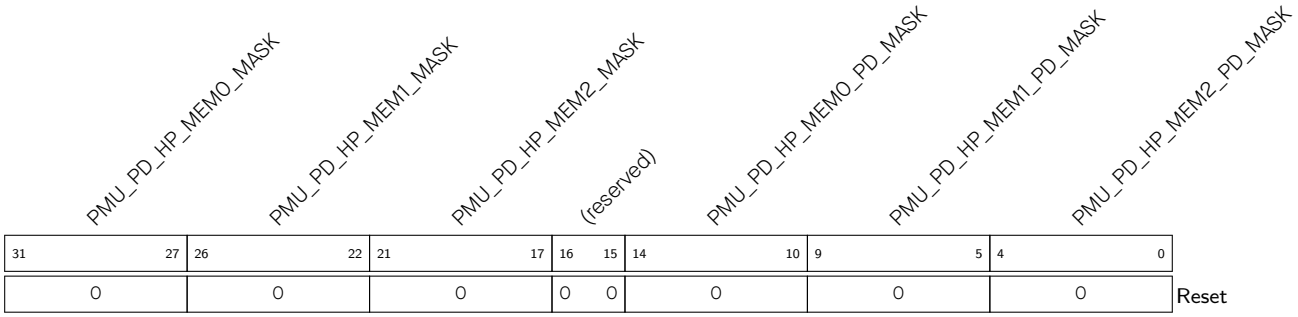
**PMU_FORCE_HP_MEM_PU** Configures whether or not to force power up Internal SRAMx domain. This setting has a lower priority than PMU_PD_HP_AON_MASK.

0: No effect

1: Force power up

(R/W)

Register 12.66. PMU_POWER_PD_MEM_MASK_REG (0x0114)




**PMU_PD_HP_MEM2_PD_MASK** Configures whether the Internal SRAM2 follows the power up/down state of "Peripherals+ROM" domain.

0: Follows "Peripherals+ROM" domain

1: Does not follow "Peripherals+ROM" domain.

(R/W)

**PMU_PD_HP_MEM1_PD_MASK** Configures whether the Internal SRAM1 follows the power up/down state of "Peripherals+ROM" domain.

0: Follows "Peripherals+ROM" domain

1: Does not follow "Peripherals+ROM" domain.

(R/W)

**PMU_PD_HP_MEM0_PD_MASK** Configures whether the Internal SRAM0 follows the power up/down state of "Peripherals+ROM" domain.

0: Follows "Peripherals+ROM" domain

1: Does not follow "Peripherals+ROM" domain.

(R/W)

**PMU_PD_HP_MEM2_MASK** Configures whether to force power up Internal SRAM2.

0: Do not force power up

1: Force power up

(R/W)

**PMU_PD_HP_MEM1_MASK** Configures whether to to force power up Internal SRAM1.

0: Do not force power up

1: Force power up

(R/W)

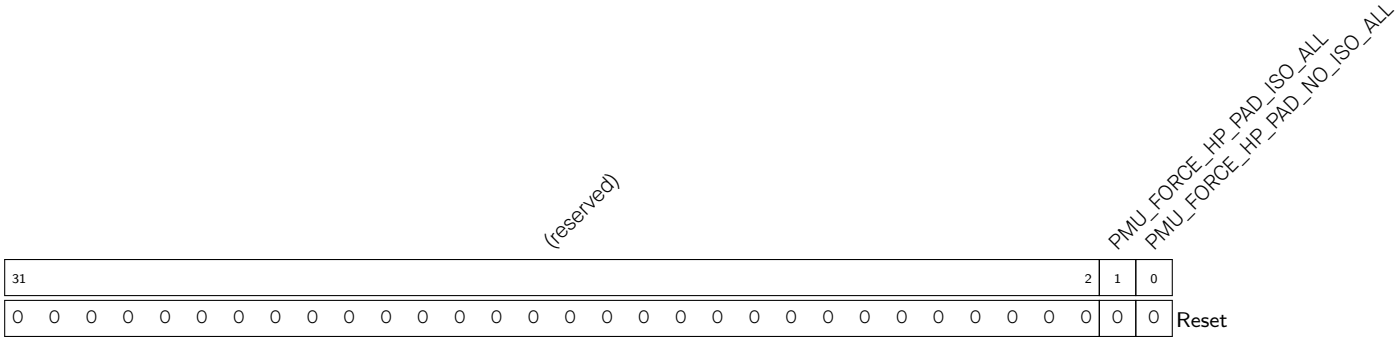
**PMU_PD_HP_MEM0_MASK** Configures whether to force power up Internal SRAM0.

0: Do not force power up

1: Force power up

(R/W)

Register 12.67. PMU_POWER_HP_PAD_REG (0x0118)




**PMU_FORCE_HP_PAD_NO_ISO_ALL** Configures whether or not to disable HP GPIOs to enable ISO signal. This setting has a lower priority than PMU_FORCE_HP_PAD_ISO_ALL.

0: No effect

1: Disable

(R/W)

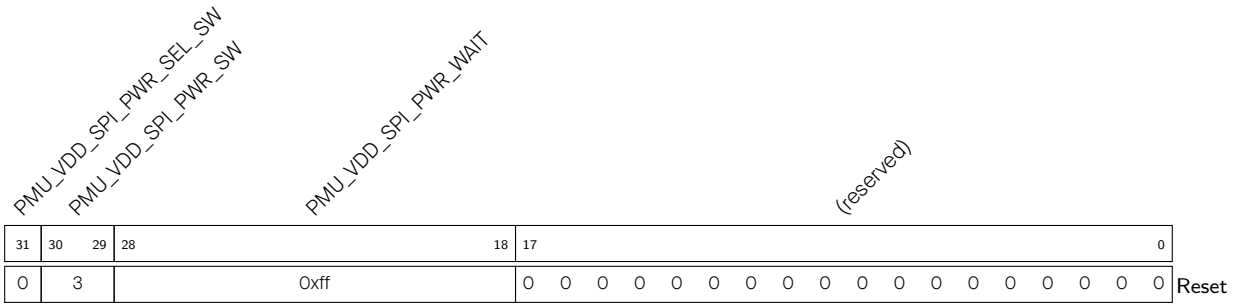
**PMU_FORCE_HP_PAD_ISO_ALL** Configures whether or not to enable HP GPIOs to enable ISO signal. This setting has a higher priority than PMU_FORCE_HP_PAD_NO_ISO_ALL.

0: No effect

1: Enable

(R/W)

Register 12.68. PMU_POWER_VDD_SPI_CNTL_REG (0x011C)




**PMU_VDD_SPI_PWR_WAIT** Configures the number of cycles to delay before powering up or down the VDD_SPI cell. (R/W)

**PMU_VDD_SPI_PWR_SW** Configure the power switch signal of VDD_SPI cell of HP GPIOs. (R/W)

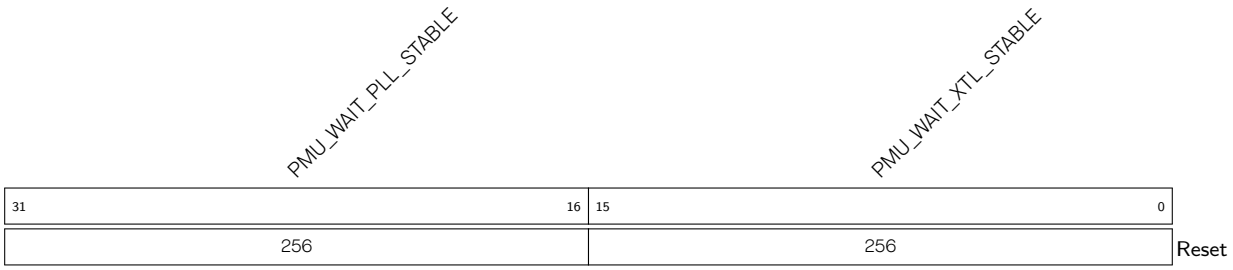
**PMU_VDD_SPI_PWR_SEL_SW** Configure to select the power switch control signals of VDD_SPI cell of HP GPIOs.

0: controlled by PMU

1: controlled by PMU_VDD_SPI_PWR_SW

(R/W)

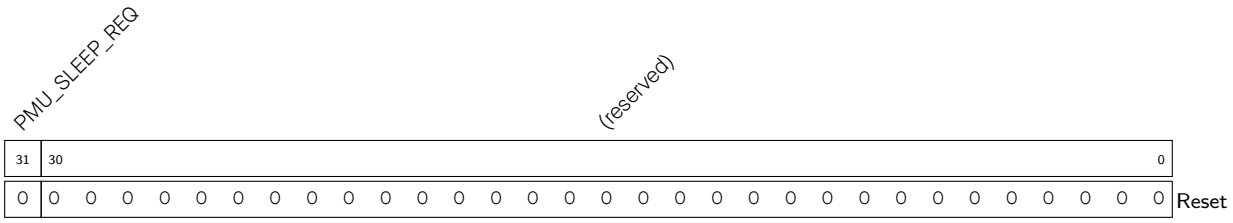
Register 12.69. PMU_POWER_CK_WAIT_CNTL_REG (0x0120)




**PMU_WAIT_XTL_STABLE** Configures the number of CLK_DYN_FAST_CLK cycles to delay the XTAL_CLK gate opening after XTAL_CLK powers up, ensuring clock stability during startup. (R/W)

**PMU_WAIT_PLL_STABLE** Configures the number of CLK_DYN_FAST_CLK cycles to delay the PLL_CLK gate opening after PLL_CLK powers up, ensuring clock stability during startup. (R/W)

Register 12.70. PMU_SLP_WAKEUP_CNTL0_REG (0x0124)



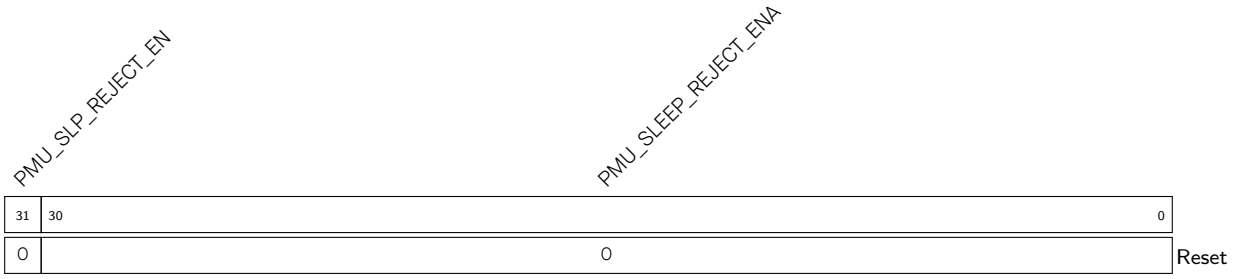

**PMU_SLEEP_REQ** Configures whether to switch the chip's PMU state to HP_SLEEP or LP_SLEEP.

0: Do not switch

1: Switch depending on the status of LP CPU.

(WT)

Register 12.71. PMU_SLP_WAKEUP_CNTL1_REG (0x0128)




**PMU_SLEEP_REJECT_ENA** Configures the sleep rejection source. For the mapping between values and sources please refer to Table 12.4-1. (R/W)

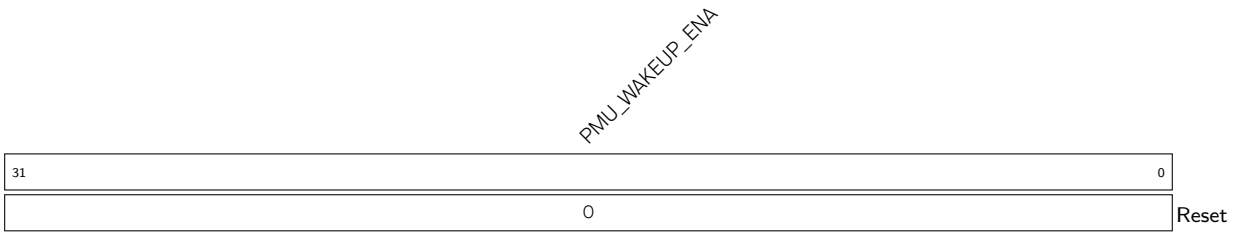
**PMU_SLP_REJECT_EN** Configures whether to enable sleep rejection function.

0: Disable

1: Enable

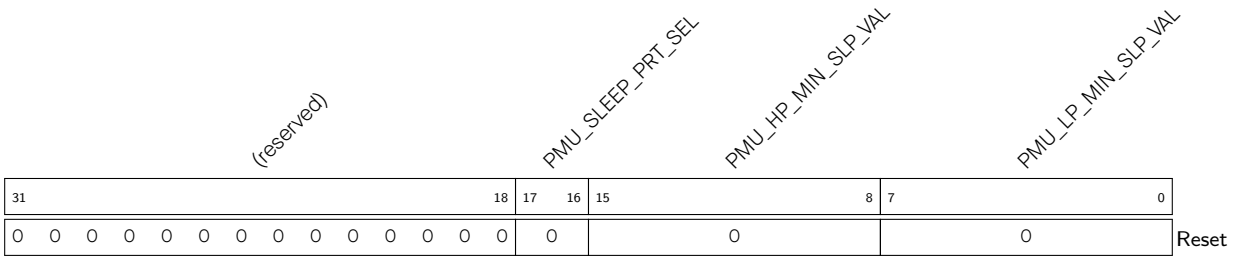
(R/W)

Register 12.72. PMU_SLP_WAKEUP_CNTL2_REG (0x012C)




**PMU_WAKEUP_ENA** Configures wake-up source. For the mapping between values and sources please refer to Table 12.4-1. (R/W)

Register 12.73. PMU_SLP_WAKEUP_CNTL3_REG (0x0130)

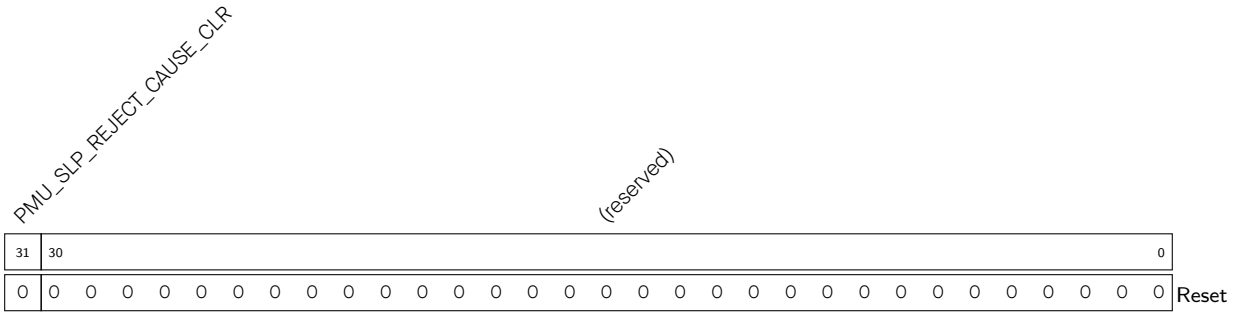



**PMU_LP_MIN_SLP_VAL** Configures the LP_SLEEP sleep/wakeup wait time. (R/W)

**PMU_HP_MIN_SLP_VAL** Configures the HP_SLEEP sleep/wakeup wait time. (R/W)

**PMU_SLEEP_PRT_SEL** Configures to select the configuration data for sleep start and sleep end events. (R/W)

Register 12.74. PMU_SLP_WAKEUP_CNTL4_REG (0x0134)




**PMU_SLP_REJECT_CAUSE_CLR** Write 1 to clear PMU_REJECT_CAUSE. (WT)

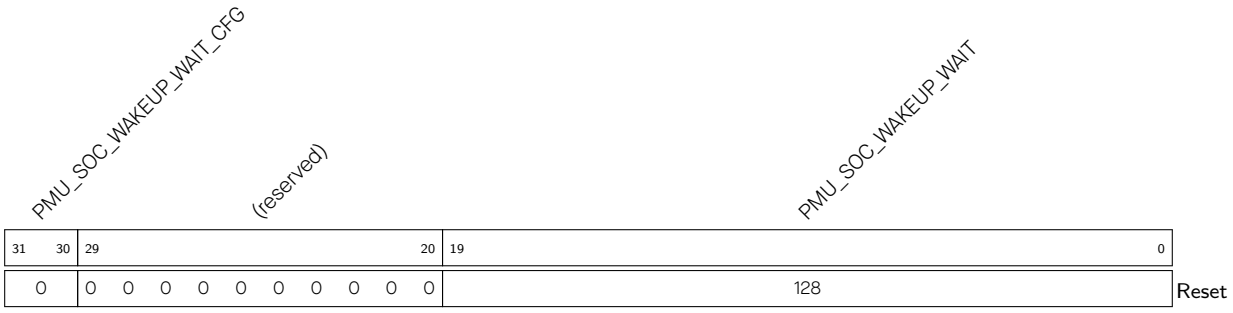
Register 12.75. PMU_SLP_WAKEUP_CNTL5_REG (0x0138)




**PMU_MODEM_WAIT_TARGET** Configures the wait timer after the Modem wakeup. (R/W)

**PMU_LP_ANA_WAIT_TARGET** Configures the wait timer after turning on the analog power. (R/W)

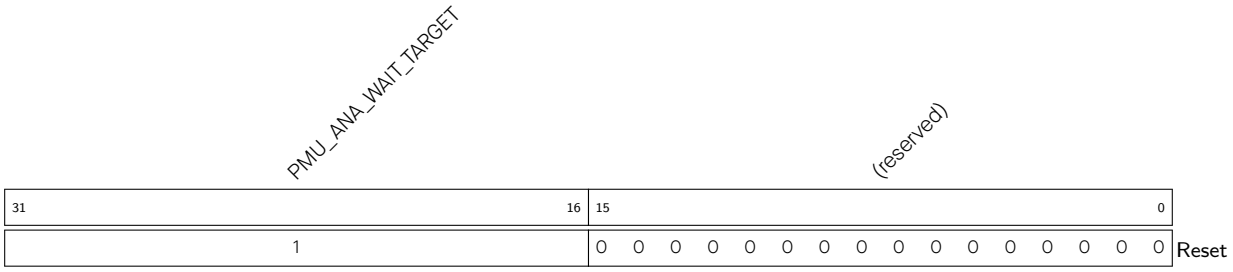
Register 12.76. PMU_SLP_WAKEUP_CNTL6_REG (0x013C)




**PMU_SOC_WAKEUP_WAIT** Configures the HP_MODEM wakeup wait timer. (R/W)

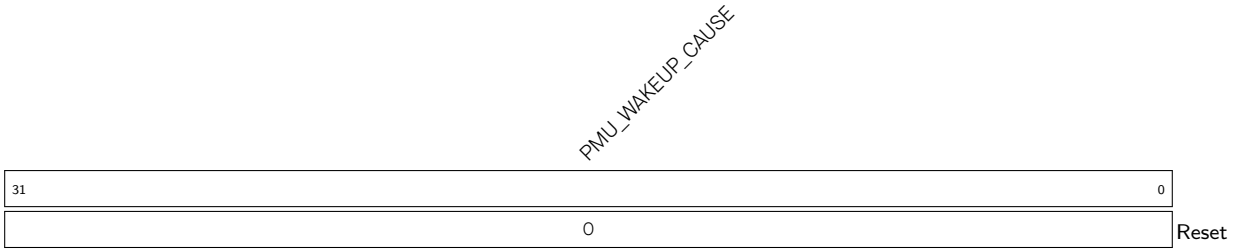
**PMU_SOC_WAKEUP_WAIT_CFG** Configures to select the wait timer for HP_MODEM wakeup. (R/W)

Register 12.77. PMU_SLP_WAKEUP_CNTL7_REG (0x0140)



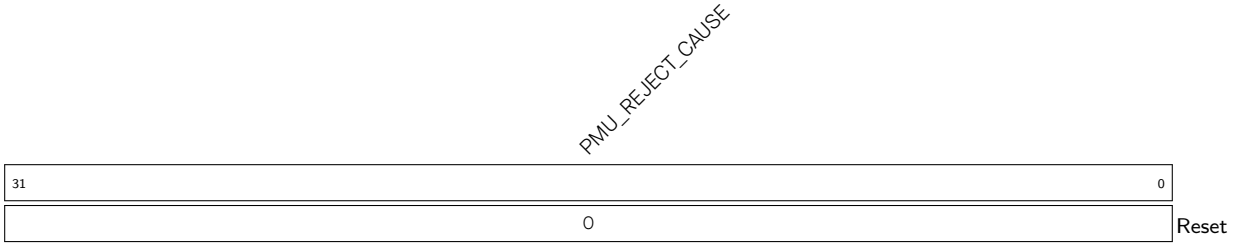

**PMU_ANA_WAIT_TARGET** Configures the HP_SWITCH wakeup wait timer. (R/W)

Register 12.78. PMU_SLP_WAKEUP_STATUS0_REG (0x0144)



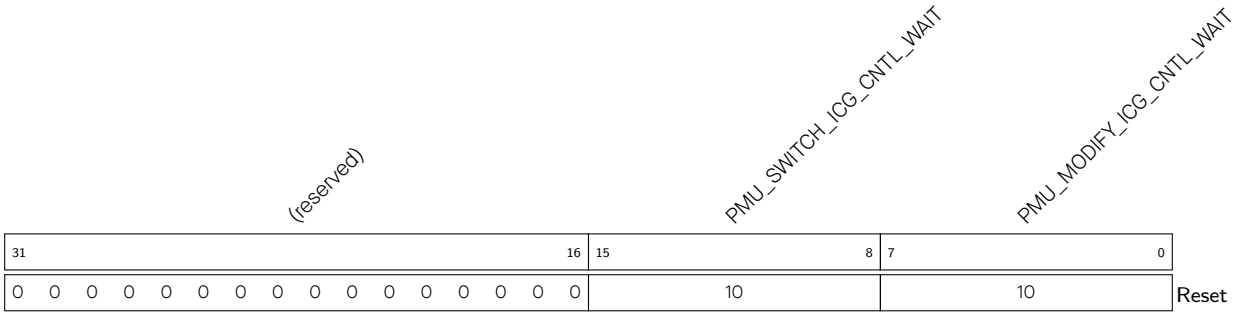

**PMU_WAKEUP_CAUSE** Indicates the wake-up source. For the mapping between values and sources please refer to Table 12.4-1. (RO)

Register 12.79. 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 12.4-1. (RO)

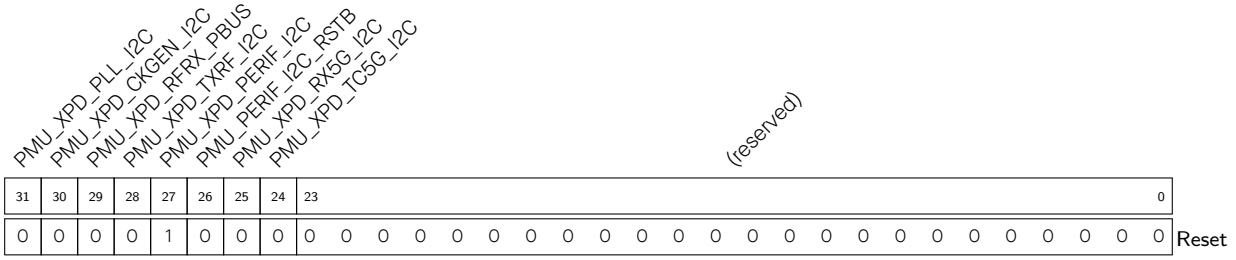
Register 12.80. PMU_HP_CK_CNTL_REG (0x0150)




**PMU_MODIFY_ICG_CNTL_WAIT** Configures the number of wait cycles required after modifying the input configurations of the Integrated Clock Gating (ICG) module, ensuring the changes take effect and the clock signals stabilize. (R/W)

**PMU_SWITCH_ICG_CNTL_WAIT** Configures the number of wait cycles required after switching the Integrated Clock Gating (ICG) module to its modified configuration, ensuring the transition is properly completed. (R/W)

Register 12.81. PMU_RF_PWC_REG (0x0158)




**PMU_XPD_TC5G_I2C** Configures the 5G transmitting output. (R/W)

**PMU_XPD_RX5G_I2C** Configures the 5G receiving output. (R/W)

**PMU_PERIF_I2C_RSTB** Configures the I2C reset. (R/W)

**PMU_XPD_PERIF_I2C** Configures the I2C SDA output.

0: outputs 0

1: outputs analog peripheral output

(R/W)

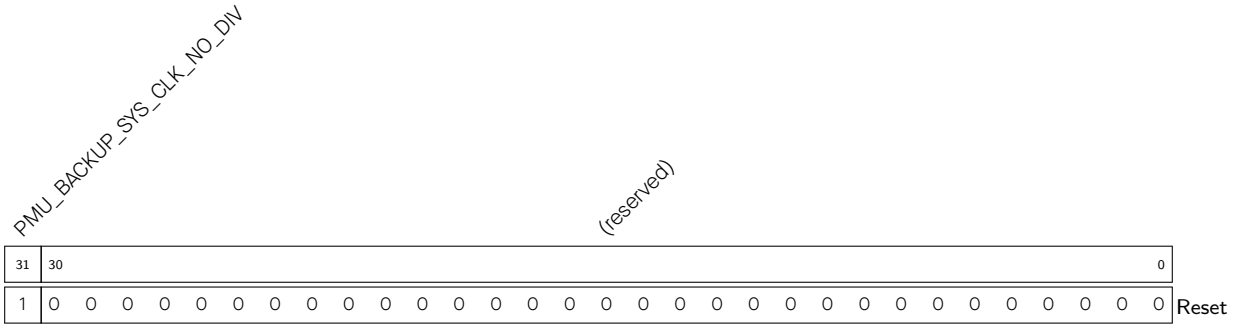
**PMU_XPD_TXRF_I2C** Configures the RF transmitting output. (R/W)

**PMU_XPD_RFRX_PBUS** Configures the RF receiving output. (R/W)

**PMU_XPD_CKGEN_I2C** Configures the 5G clock generation. (R/W)

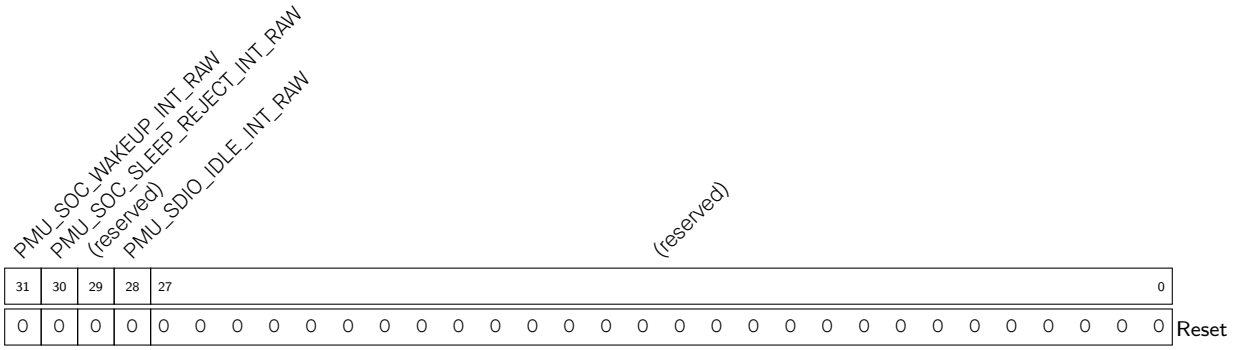
**PMU_XPD_PLL_I2C** Configures the PLL clock generation. (R/W)

Register 12.82. PMU_BACKUP_CFG_REG (0x015C)




**PMU_BACKUP_SYS_CLK_NO_DIV** Configures the division number of the backup clock. (R/W)

Register 12.83. PMU_INT_RAW_REG (0x0160)

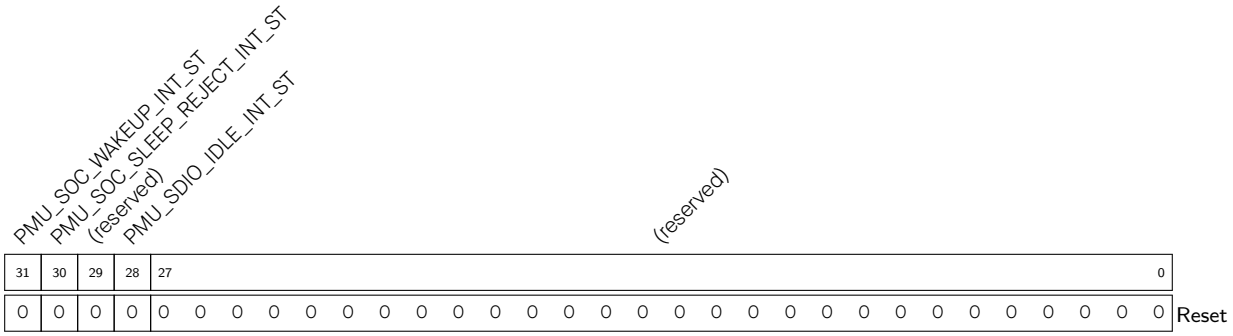



**PMU_SDIO_IDLE_INT_RAW** The raw interrupt status of PMU_SDIO_IDLE_INT. (R/WTC/SS)

**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 12.84. PMU_HP_INT_ST_REG (0x0160)

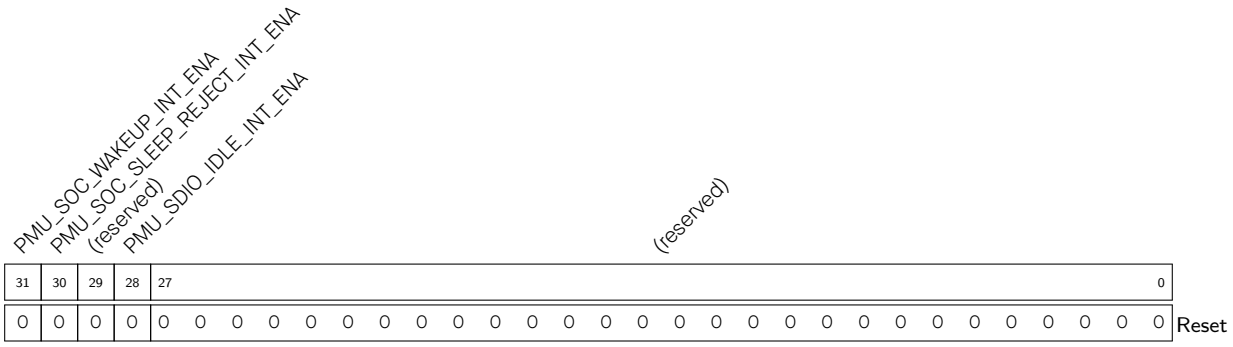



**PMU_SDIO_IDLE_INT_ST** The masked interrupt status of PMU_SDIO_IDLE_INT. (RO)

**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 12.85. PMU_HP_INT_ENA_REG (0x0168)

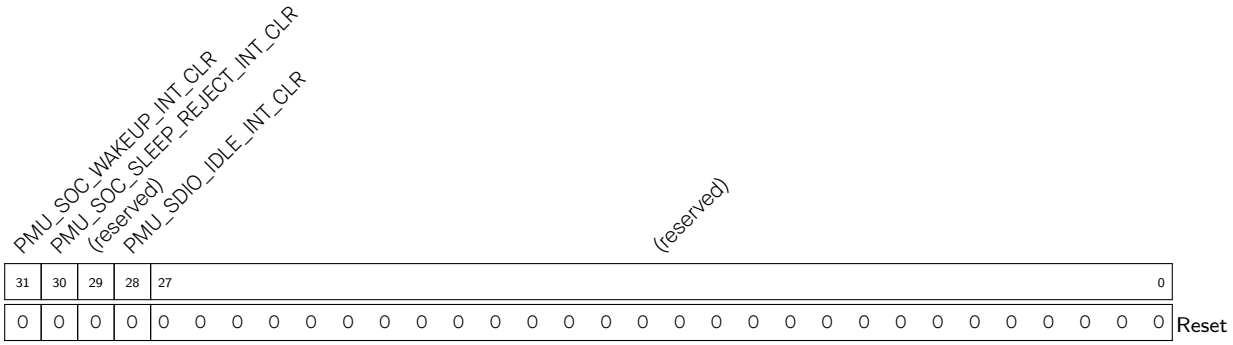



**PMU_SDIO_IDLE_INT_ENA** Write 1 to enable PMU_SDIO_IDLE_INT. (R/W)

**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 12.86. PMU_HP_INT_CLR_REG (0x016C)

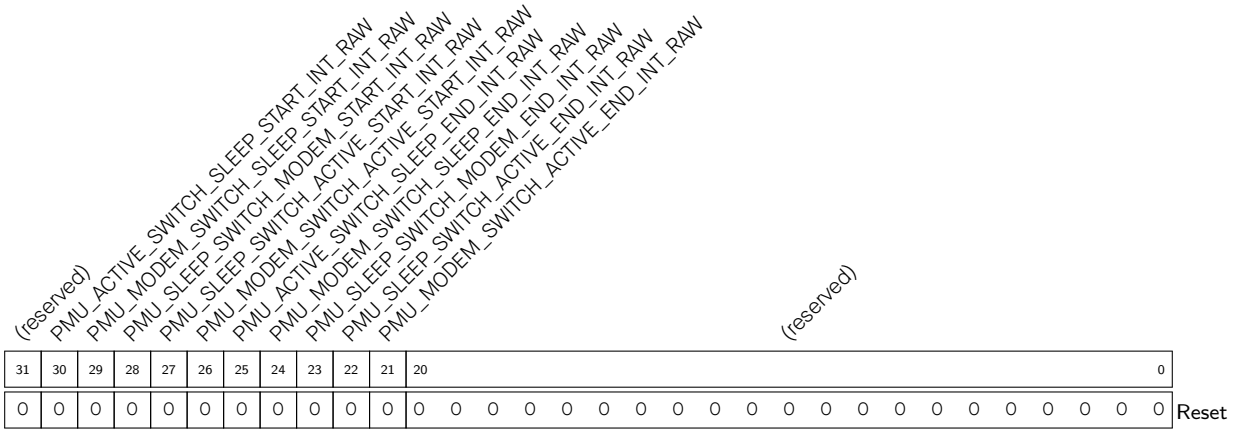



**PMU_SDIO_IDLE_INT_CLR** Write 1 to clear PMU_SDIO_IDLE_INT. (WT)

**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 12.87. PMU_LP_INT_RAW_REG (0x0170)




**PMU_MODEM_SWITCH_ACTIVE_END_INT_RAW** The raw interrupt status of PMU_MODEM_SWITCH_ACTIVE_END_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_SLEEP_SWITCH_MODEM_END_INT_RAW** The raw interrupt status of PMU_SLEEP_SWITCH_MODEM_END_INT. (R/WTC/SS)

**PMU_MODEM_SWITCH_SLEEP_END_INT_RAW** The raw interrupt status of PMU_MODEM_SWITCH_SLEEP_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_MODEM_SWITCH_ACTIVE_START_INT_RAW** The raw interrupt status of PMU_MODEM_SWITCH_ACTIVE_START_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)

**PMU_SLEEP_SWITCH_MODEM_START_INT_RAW** The raw interrupt status of PMU_SLEEP_SWITCH_MODEM_START_INT. (R/WTC/SS)

**PMU_MODEM_SWITCH_SLEEP_START_INT_RAW** The raw interrupt status of PMU_MODEM_SWITCH_SLEEP_START_INT. (R/WTC/SS)

**PMU_ACTIVE_SWITCH_SLEEP_START_INT_RAW** The raw interrupt status of PMU_ACTIVE_SWITCH_SLEEP_START_INT. (R/WTC/SS)

Register 12.88. PMU_LP_INT_ST_REG (0x0170)

| Bit | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 … 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Field | (reserved) | PMU_ACTIVE_SWITCH_SLEEP_START_INT_ST | PMU_MODEM_SWITCH_SLEEP_START_INT_ST | PMU_SLEEP_SWITCH_MODEM_START_INT_ST | PMU_SLEEP_SWITCH_ACTIVE_START_INT_ST | PMU_MODEM_SWITCH_ACTIVE_START_INT_ST | PMU_ACTIVE_SWITCH_SLEEP_END_INT_ST | PMU_MODEM_SWITCH_SLEEP_END_INT_ST | PMU_SLEEP_SWITCH_MODEM_END_INT_ST | PMU_SLEEP_SWITCH_ACTIVE_END_INT_ST | PMU_MODEM_SWITCH_ACTIVE_END_INT_ST | (reserved) |
| Reset | 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 |

**PMU_MODEM_SWITCH_ACTIVE_END_INT_ST** The masked interrupt status of PMU_MODEM_SWITCH_ACTIVE_END_INT. (RO)

**PMU_SLEEP_SWITCH_ACTIVE_END_INT_ST** The masked interrupt status of PMU_SLEEP_SWITCH_ACTIVE_END_INT. (RO)

**PMU_SLEEP_SWITCH_MODEM_END_INT_ST** The masked interrupt status of PMU_SLEEP_SWITCH_MODEM_END_INT. (RO)

**PMU_MODEM_SWITCH_SLEEP_END_INT_ST** The masked interrupt status of PMU_MODEM_SWITCH_SLEEP_END_INT. (RO)

**PMU_ACTIVE_SWITCH_SLEEP_END_INT_ST** The masked interrupt status of PMU_ACTIVE_SWITCH_SLEEP_END_INT. (RO)

**PMU_MODEM_SWITCH_ACTIVE_START_INT_ST** The masked interrupt status of PMU_MODEM_SWITCH_ACTIVE_START_INT. (RO)

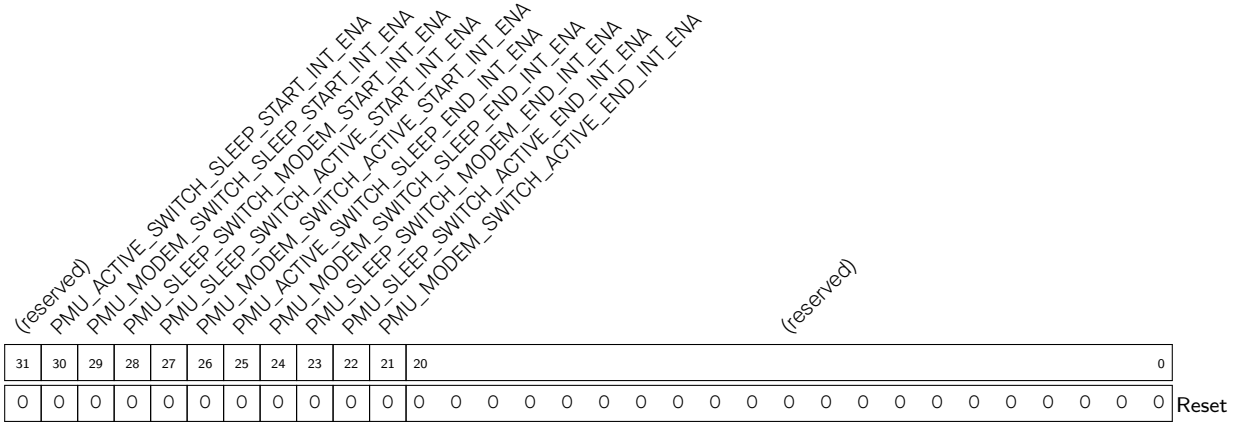
**PMU_SLEEP_SWITCH_ACTIVE_START_INT_ST** The masked interrupt status of PMU_SLEEP_SWITCH_ACTIVE_START_INT. (RO)

**PMU_SLEEP_SWITCH_MODEM_START_INT_ST** The masked interrupt status of PMU_SLEEP_SWITCH_MODEM_START_INT. (RO)

**PMU_MODEM_SWITCH_SLEEP_START_INT_ST** The masked interrupt status of PMU_MODEM_SWITCH_SLEEP_START_INT. (RO)

**PMU_ACTIVE_SWITCH_SLEEP_START_INT_ST** The masked interrupt status of PMU_ACTIVE_SWITCH_SLEEP_START_INT. (RO)

Register 12.89. PMU_LP_INT_ENA_REG (0x0178)




**PMU_MODEM_SWITCH_ACTIVE_END_INT_ENA** Write 1 to enable PMU_MODEM_SWITCH_ACTIVE_END_INT. (R/W)

**PMU_SLEEP_SWITCH_ACTIVE_END_INT_ENA** Write 1 to enable PMU_SLEEP_SWITCH_ACTIVE_END_INT. (R/W)

**PMU_SLEEP_SWITCH_MODEM_END_INT_ENA** Write 1 to enable PMU_SLEEP_SWITCH_MODEM_END_INT. (R/W)

**PMU_MODEM_SWITCH_SLEEP_END_INT_ENA** Write 1 to enable PMU_MODEM_SWITCH_SLEEP_END_INT. (R/W)

**PMU_ACTIVE_SWITCH_SLEEP_END_INT_ENA** Write 1 to enable PMU_ACTIVE_SWITCH_SLEEP_END_INT. (R/W)

**PMU_MODEM_SWITCH_ACTIVE_START_INT_ENA** Write 1 to enable PMU_MODEM_SWITCH_ACTIVE_START_INT. (R/W)

**PMU_SLEEP_SWITCH_ACTIVE_START_INT_ENA** Write 1 to enable PMU_SLEEP_SWITCH_ACTIVE_START_INT. (R/W)

**PMU_SLEEP_SWITCH_MODEM_START_INT_ENA** Write 1 to enable PMU_SLEEP_SWITCH_MODEM_START_INT. (R/W)

**PMU_MODEM_SWITCH_SLEEP_START_INT_ENA** Write 1 to enable PMU_MODEM_SWITCH_SLEEP_START_INT. (R/W)

**PMU_ACTIVE_SWITCH_SLEEP_START_INT_ENA** Write 1 to enable PMU_ACTIVE_SWITCH_SLEEP_START_INT. (R/W)

### Register 12.90. PMU_LP_INT_CLR_REG (0x017C)

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 … 0 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| (reserved) | PMU_ACTIVE_SWITCH_SLEEP_START_INT_CLR | PMU_MODEM_SWITCH_SLEEP_START_INT_CLR | PMU_SLEEP_SWITCH_MODEM_START_INT_CLR | PMU_SLEEP_SWITCH_ACTIVE_START_INT_CLR | PMU_MODEM_SWITCH_ACTIVE_START_INT_CLR | PMU_ACTIVE_SWITCH_SLEEP_END_INT_CLR | PMU_MODEM_SWITCH_SLEEP_END_INT_CLR | PMU_SLEEP_SWITCH_MODEM_END_INT_CLR | PMU_SLEEP_SWITCH_ACTIVE_END_INT_CLR | PMU_MODEM_SWITCH_ACTIVE_END_INT_CLR | (reserved) | |
| 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 |

**PMU_MODEM_SWITCH_ACTIVE_END_INT_CLR** Write 1 to clear PMU_MODEM_SWITCH_ACTIVE_END_INT. (WT)

**PMU_SLEEP_SWITCH_ACTIVE_END_INT_CLR** Write 1 to clear PMU_SLEEP_SWITCH_ACTIVE_END_INT. (WT)

**PMU_SLEEP_SWITCH_MODEM_END_INT_CLR** Write 1 to clear PMU_SLEEP_SWITCH_MODEM_END_INT. (WT)

**PMU_MODEM_SWITCH_SLEEP_END_INT_CLR** Write 1 to clear PMU_MODEM_SWITCH_SLEEP_END_INT. (WT)

**PMU_ACTIVE_SWITCH_SLEEP_END_INT_CLR** Write 1 to clear PMU_ACTIVE_SWITCH_SLEEP_END_INT. (WT)

**PMU_MODEM_SWITCH_ACTIVE_START_INT_CLR** Write 1 to clear PMU_MODEM_SWITCH_ACTIVE_START_INT. (WT)

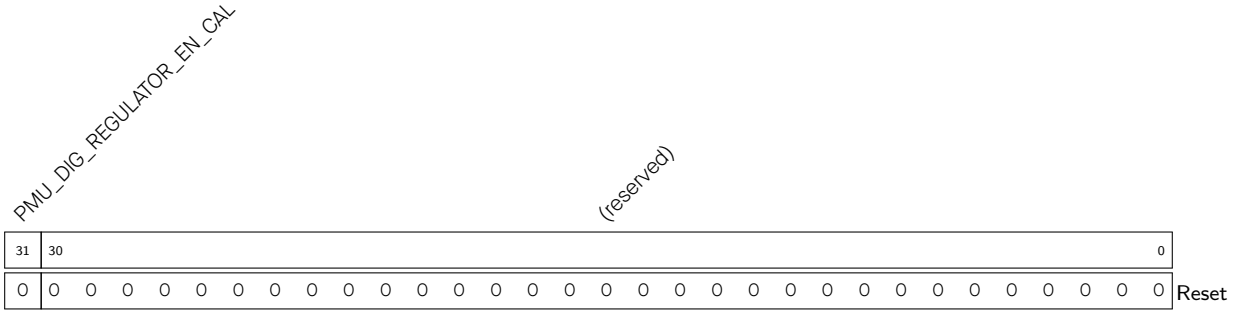
**PMU_SLEEP_SWITCH_ACTIVE_START_INT_CLR** Write 1 to clear PMU_SLEEP_SWITCH_ACTIVE_START_INT. (WT)

**PMU_SLEEP_SWITCH_MODEM_START_INT_CLR** Write 1 to clear PMU_SLEEP_SWITCH_MODEM_START_INT. (WT)

**PMU_MODEM_SWITCH_SLEEP_START_INT_CLR** Write 1 to clear PMU_MODEM_SWITCH_SLEEP_START_INT. (WT)

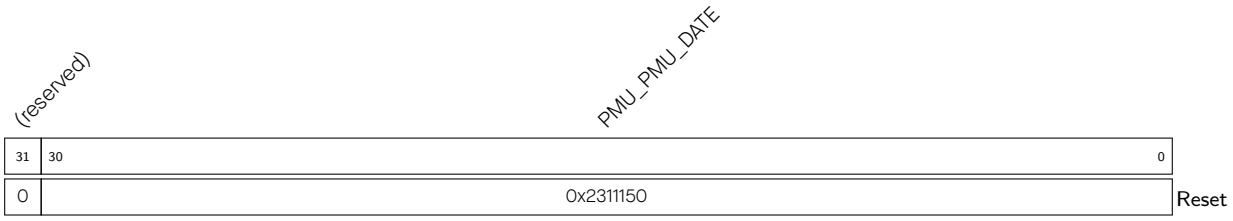
**PMU_ACTIVE_SWITCH_SLEEP_START_INT_CLR** Write 1 to clear PMU_ACTIVE_SWITCH_SLEEP_START_INT. (WT)

Register 12.91. PMU_HP_REGULATOR_CFG_REG (0x018C)




**PMU_DIG_REGULATOR_EN_CAL** Configures to enable the digital regulator. (R/W)

Register 12.92. PMU_DATE_REG (0x01A8)

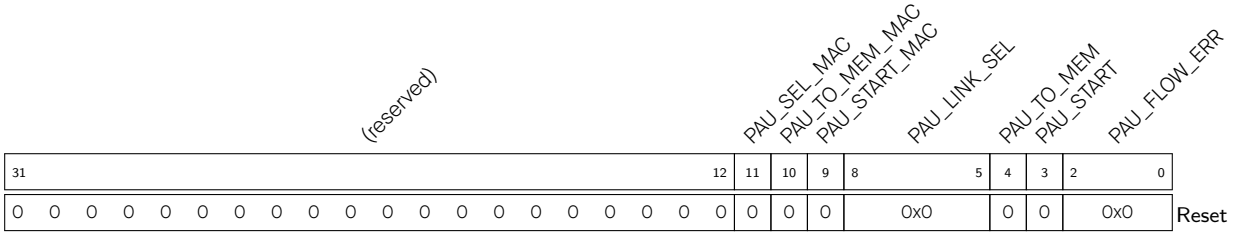



**PMU_PMU_DATE** Version control register. (R/W)

## 12.9.2 PAU Registers

The addresses in this section below are relative to the PAU base address provided in Table 5.3-2 in Chapter 5 *System and Memory*.

Register 12.93. PAU_REGDMA_CONF_REG (0x0000)




**PAU_FLOW_ERR** Represents the backup error type.

0: Received software triggered interrupt clear signal

1: Peripheral AHB bus timeout

2: Memory AHB bus timeout

3: Address link list wait timer timeout

4: Address mapping error

(RO)

**PAU_START** Write 1 to initiate the backup start signal. (WT)

**PAU_TO_MEM** Configures the backup direction.

0: register to memory

1: memory to register

(R/W)

**PAU_LINK_SEL** Configures to select the link.

0: PAU_REGDMA_LINK_0_ADDR_REG

1: PAU_REGDMA_LINK_1_ADDR_REG

2: PAU_REGDMA_LINK_2_ADDR_REG

3: PAU_REGDMA_LINK_3_ADDR_REG

(R/W)

**PAU_START_MAC** Write 1 to initiate the MAC software backup start signal. (WT)

**PAU_TO_MEM_MAC** Configures the MAC software backup direction.

0: register to memory

1: memory to register

(R/W)

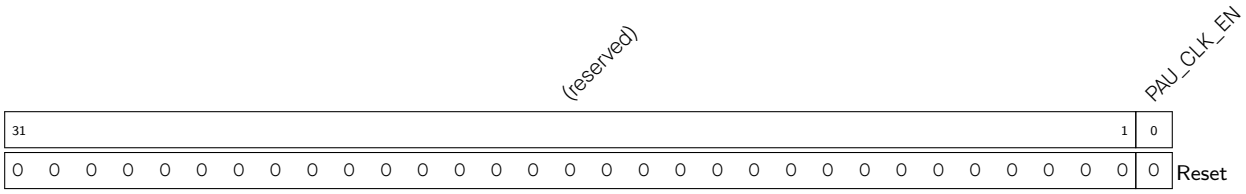
**PAU_SEL_MAC** Configures the MAC hardware/software selection.

0: select the hardware path

1: select the software path

(R/W)

Register 12.94. PAU_REGDMA_CLK_CONF_REG (0x0004)



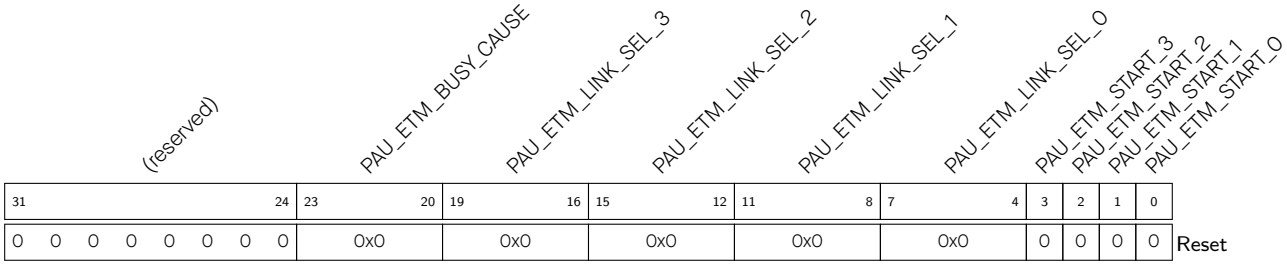

**PAU_CLK_EN** Configures whether or not to enable clock for the PAU module.

0: Disable

1: Enable

(R/W)

Register 12.95. PAU_REGDMA_ETM_CTRL_REG (0x0008)




**PAU_ETM_BUSY_CAUSE** Records the ETM busy cause for debug purpose. (RO)

**PAU_ETM_LINK_SEL_0** Configures to select a link for ETM operation 0. (R/W)

**PAU_ETM_LINK_SEL_1** Configures to select a link for ETM operation 1. (R/W)

**PAU_ETM_LINK_SEL_2** Configures to select a link for ETM operation 2. (R/W)

**PAU_ETM_LINK_SEL_3** Configures to select a link for ETM operation 3. (R/W)

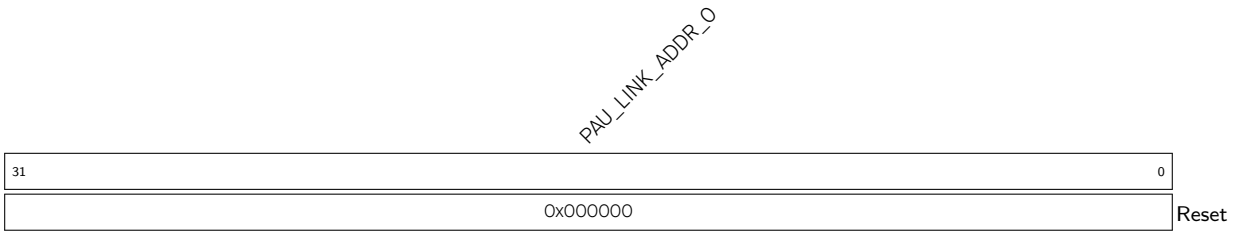
**PAU_ETM_START_0** Write 1 to start ETM operation 0. (WT)

**PAU_ETM_START_1** Write 1 to start ETM operation 1. (WT)

**PAU_ETM_START_2** Write 1 to start ETM operation 2. (WT)

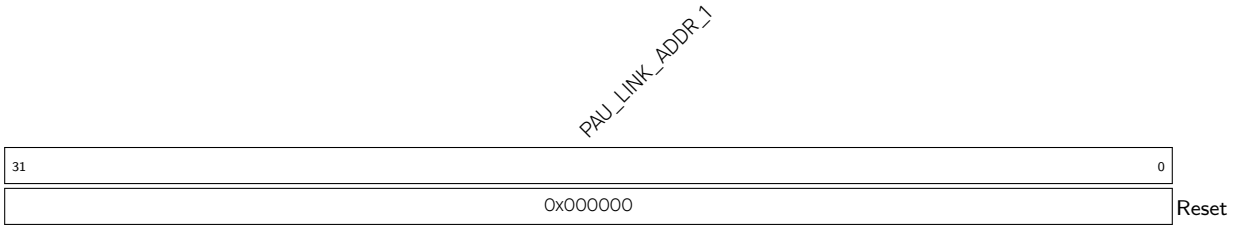
**PAU_ETM_START_3** Write 1 to start ETM operation 3. (WT)

Register 12.96. PAU_REGDMA_LINK_0_ADDR_REG (0x000C)



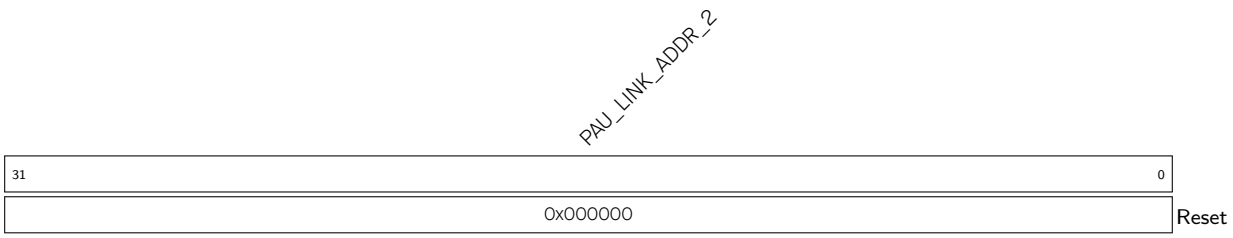

**PAU_LINK_ADDR_0** Represents the address for link 0. (R/W)

Register 12.97. PAU_REGDMA_LINK_1_ADDR_REG (0x0010)



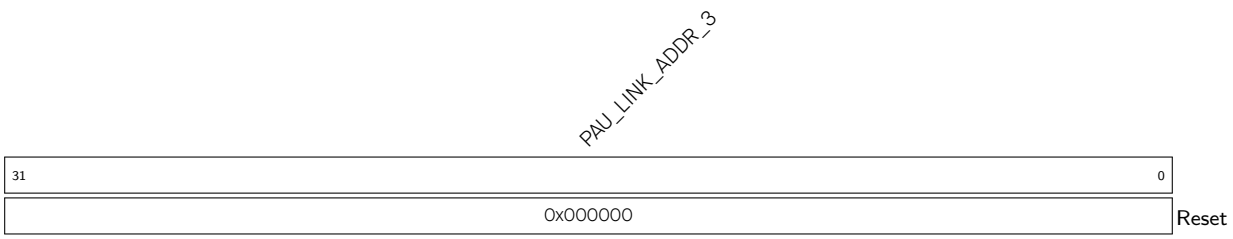

**PAU_LINK_ADDR_1** Represents the address for link 1. (R/W)

Register 12.98. PAU_REGDMA_LINK_2_ADDR_REG (0x0014)



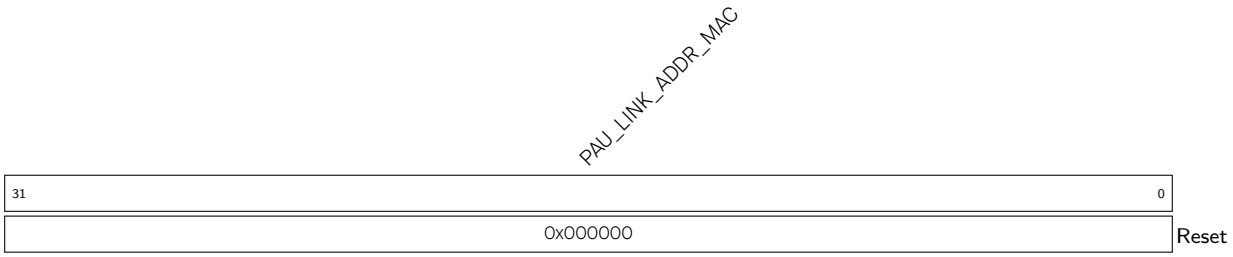

**PAU_LINK_ADDR_2** Represents the address for link 2. (R/W)

Register 12.99. PAU_REGDMA_LINK_3_ADDR_REG (0x0018)



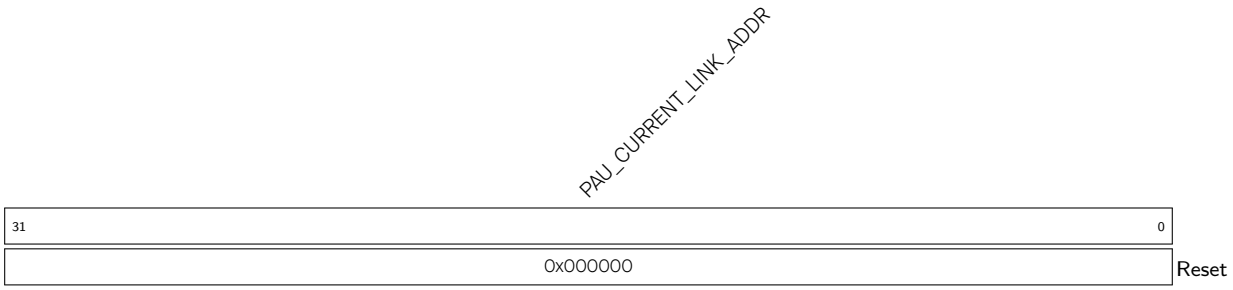

**PAU_LINK_ADDR_3** Represents the address for link 3. (R/W)

Register 12.100. PAU_REGDMA_LINK_MAC_ADDR_REG (0x001C)



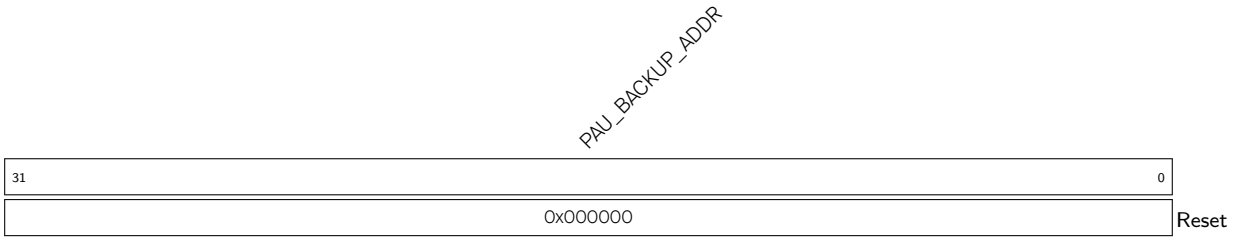

**PAU_LINK_ADDR_MAC** Represents the MAC address for the link. (R/W)

Register 12.101. PAU_REGDMA_CURRENT_LINK_ADDR_REG (0x0020)



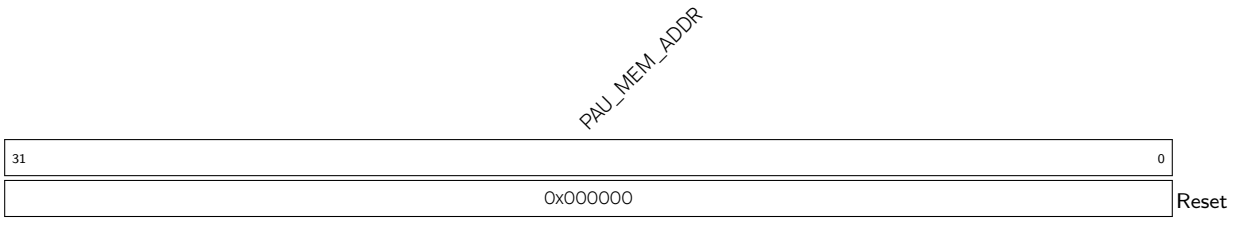

**PAU_CURRENT_LINK_ADDR** Represents the current link address. (RO)

Register 12.102. PAU_REGDMA_BACKUP_ADDR_REG (0x0024)



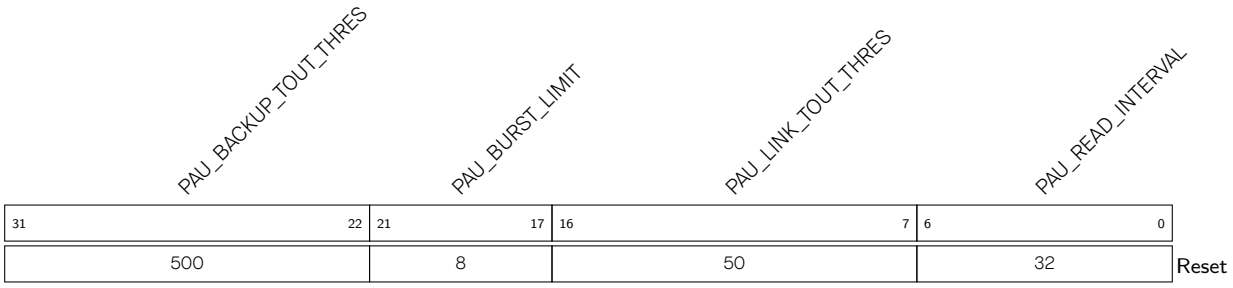

**PAU_BACKUP_ADDR** Represents the backup address. (RO)

Register 12.103. PAU_REGDMA_MEM_ADDR_REG (0x0028)




**PAU_MEM_ADDR** Represents the memory address. (RO)

Register 12.104. PAU_REGDMA_BKP_CONF_REG (0x002C)



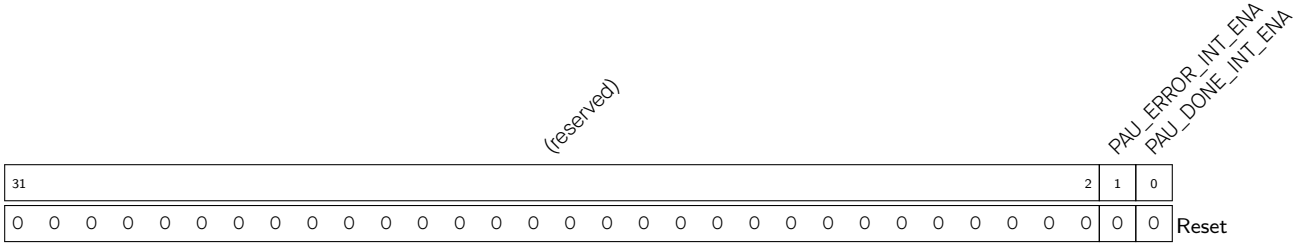

**PAU_READ_INTERVAL** Configures the read interval for the link. (R/W)

**PAU_LINK_TOUT_THRES** Configures the link wait timeout threshold. (R/W)

**PAU_BURST_LIMIT** Configures the burst limit. (R/W)

**PAU_BACKUP_TOUT_THRES** Configures the backup timeout threshold. (R/W)

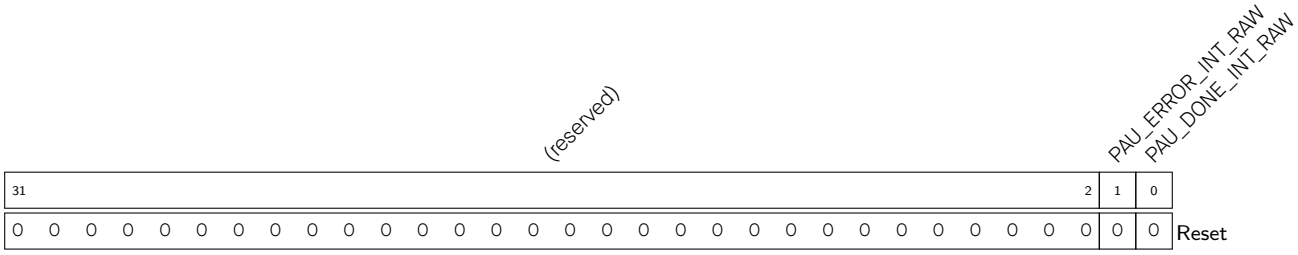
Register 12.105. PAU_INT_ENA_REG (0x0030)




**PAU_DONE_INT_ENA** Write 1 to enable backup done interrupt. (R/W)

**PAU_ERROR_INT_ENA** Write 1 to enable error interrupt. (R/W)

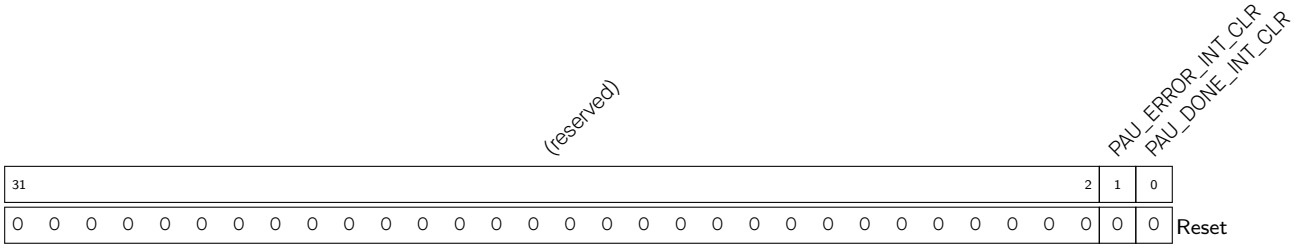
Register 12.106. PAU_INT_RAW_REG (0x0034)




**PAU_DONE_INT_RAW** The raw interrupt status for backup completion. (R/WTC/SS)

**PAU_ERROR_INT_RAW** The raw interrupt status for errors. (R/WTC/SS)

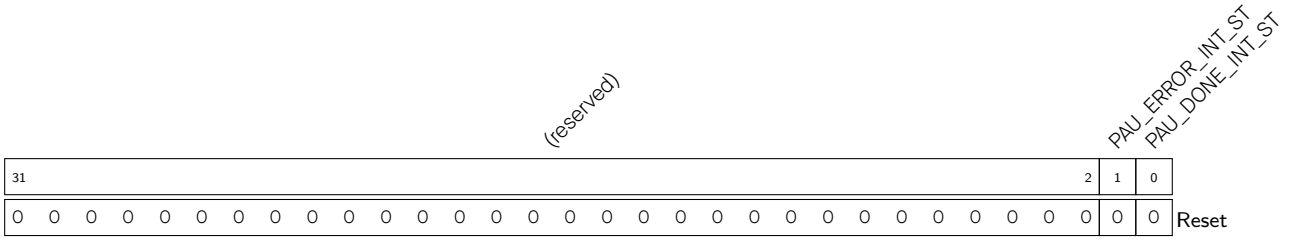
Register 12.107. PAU_INT_CLR_REG (0x0038)




**PAU_DONE_INT_CLR** Write 1 to clear the backup done interrupt. (WT)

**PAU_ERROR_INT_CLR** Write 1 to clear the error interrupt. (WT)

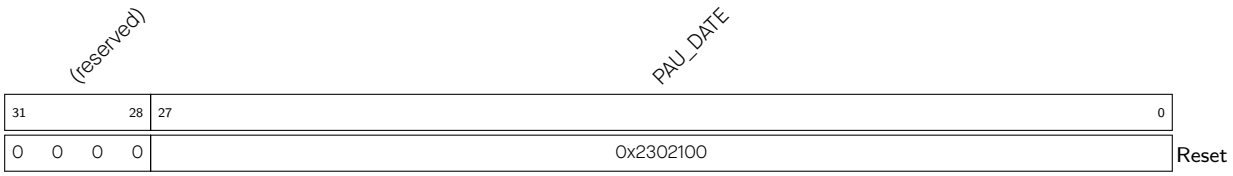
Register 12.108. PAU_INT_ST_REG (0x003C)




**PAU_DONE_INT_ST** Represents the backup completion status. (RO)

**PAU_ERROR_INT_ST** Represents the error status. (RO)

Register 12.109. PAU_DATE_REG (0x03FC)




**PAU_DATE** Version control register. (R/W)

# Chapter 13

# System Timer

## 13.1 Overview

ESP32-C61 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.

## 13.2 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/48 MHz XTAL_CLK as the clock source of CNT_CLK
- 52-bit alarm values (t) and 26-bit alarm periods ($\delta$t)
- 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 ($\delta$t)
- Three comparators generating three independent interrupts based on configured alarm value (t) or alarm period ($\delta$t)
- 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

## 13.3 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 13.3-1.

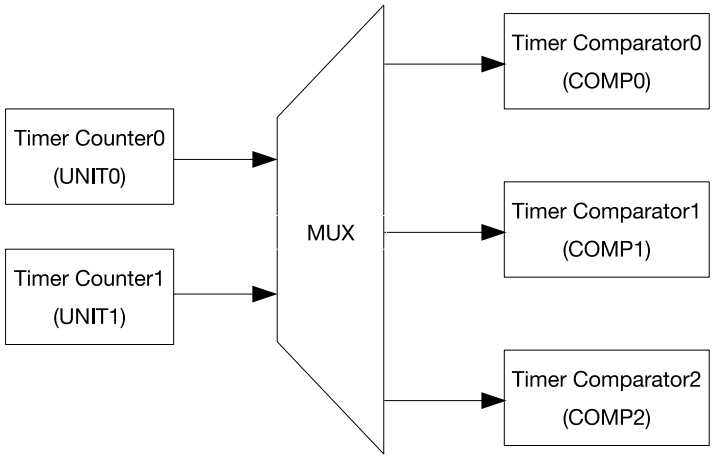



Figure 13.3-1. System Timer Structure

## 13.4 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 PCR_SYSTIMER_FUNC_CLK_SEL in register PCR_SYSTIMER_FUNC_CLK_CONF_REG. After XTAL_CLK is scaled, a counter clock signal CNT_CLK clock is generated. The average clock frequency of CNT_CLK is 16 MHz, as shown in Figure 13.5-1. The timer counter is incremented by 1/16 $\mu s$ on each CNT_CLK cycle.

Software operation such as configuring registers is clocked by APB_CLK. For more information about APB_CLK, see Chapter 8 *Reset and Clock*.

The following two bits of system registers are also used to control the system timer:

- Set PCR_SYSTIMER_CLK_EN in register PCR_SYSTIMER_CONF_REG to enable APB_CLK signal to the system timer.
- Set PCR_SYSTIMER_RST_EN in register PCR_SYSTIMER_CONF_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 8 *Reset and Clock*.

## 13.5 Functional Description

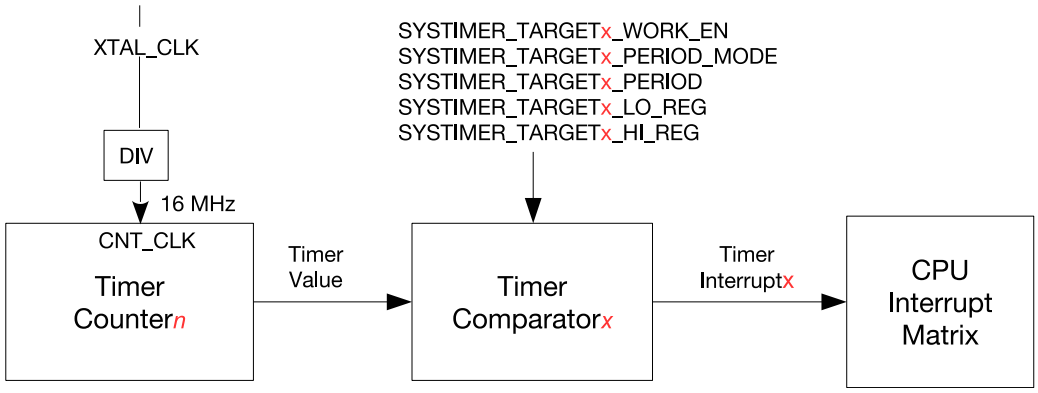



Figure 13.5-1. System Timer Alarm Generation

Figure 13.5-1 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.

### 13.5.1 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 13.5-1. 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 | x | 1 | Stop counting, but will continue its counting after the CPU1 resumes |
| 1 | 1 | x | Stop counting, but will continue its counting after the CPU0 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.

### 13.5.2 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 ($\delta$t).

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 ($\delta$t) is provided by the register SYSTIMER_TARGET*x*_PERIOD. Assuming that current count value is t1, when it reaches (t1 + $\delta$t), an alarm interrupt will be generated. When the count value reaches (t1 + 2×$\delta$t), 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×$\delta$t).

An alarm is generated when the count value equals to the alarm value (t) in target mode or to the start value + n×$\delta$t (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 13.5-2. 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<br>( $t_c$ < $2^{51}$ and $t_t$ < $2^{51}$,<br>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. |

### 13.5.3 Event Task Matrix

The system timer on ESP32-C61 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 11 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.

### 13.5.4 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 13.5-3.
2. Software writes 1 to corresponding bits to start synchronization. See the second column in Table 13.5-3.

Table 13.5-3. Synchronization Operation for Configuration Registers

| Configuration Fields | Synchronization Enable Bit |
|---|---|
| SYSTIMER_TIMER_UNIT*n*_LOAD_LO<br>SYSTIMER_TIMER_UNIT*n*_LOAD_HI | SYSTIMER_TIMER_UNIT*n*_LOAD |
| SYSTIMER_TARGET*x*_PERIOD<br>SYSTIMER_TIMER_TARGET*x*_HI<br>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.

## 13.6 Interrupts

ESP32-C61'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 13.6-1.

**Table 13.6-1. 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 10 *Interrupt Matrix* > Section 10.2 *Interrupt Terminology in ESP32-C61*.

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 13.8 *Register Summary*.

## 13.7 Programming Procedure

When configuring COMP*x* and UNIT*n*, please ensure the corresponding COMP and UNIT are at work.

### 13.7.1 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.

### 13.7.2 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 13.7.1. 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.

### 13.7.3 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 ($\delta$t), and fill it into SYSTIMER_TARGET*x*_PERIOD.
3. Set SYSTIMER_TIMER_COMP*x*_LOAD to synchronize the alarm period ($\delta$t) to COMP*x*, i.e., load the alarm period ($\delta$t) 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×$\delta$t) (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×$\delta$t (n = 1, 2, 3...) set in Step 2.

### 13.7.4 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 12 *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 13.7.1.

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.

## 13.8 Register Summary

The addresses in this section are relative to system timer base address provided in Table 5.3-2 in Chapter 5 *System and Memory*.

The abbreviations given in Column **Access** are explained in Section *Access Types for Registers*.

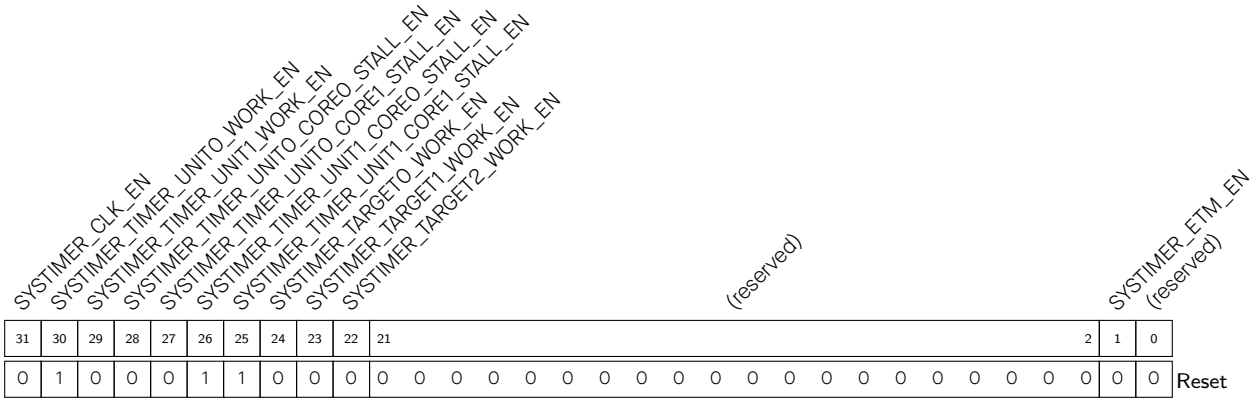
| Name | Description | Address | Access |
|---|---|---|---|
| **Clock Control Register** | | | |
| SYSTIMER_CONF_REG | Configure system timer clock | 0x0000 | R/W |
| **UNIT*0* Control and Configuration Registers** | | | |
| SYSTIMER_UNIT0_OP_REG | Read UNIT0 value to registers | 0x0004 | varies |
| 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 |

## 13.9 Registers

The addresses in this section are relative to system timer base address provided in Table 5.3-2 in Chapter 5 *System and Memory*.

**Register 13.1. SYSTIMER_CONF_REG (0x0000)**




**SYSTIMER_ETM_EN** Configures whether to enable generation of ETM events.

0: Disable

1: Enable

(R/W)

**SYSTIMER_TARGET2_WORK_EN** Configures whether to enable COMP2.

0: Disable

1: Enable

(R/W)

**SYSTIMER_TARGET1_WORK_EN** Configures whether to enable COMP1. See details in SYSTIMER_TARGET2_WORK_EN. (R/W)

**SYSTIMER_TARGET0_WORK_EN** Configures whether to enable COMP0. See details in SYSTIMER_TARGET2_WORK_EN. (R/W)

**SYSTIMER_TIMER_UNIT1_CORE1_STALL_EN** Configures whether UNIT1 is stalled when CORE1 is stalled.

0: UNIT1 is not stalled.

1: UNIT1 is stalled.

(R/W)

**SYSTIMER_TIMER_UNIT1_CORE0_STALL_EN** Configures whether UNIT1 is stalled when CORE0 is stalled. See details in SYSTIMER_TIMER_UNIT1_CORE1_STALL_EN. (R/W)

**SYSTIMER_TIMER_UNIT0_CORE1_STALL_EN** Configures whether UNIT0 is stalled when CORE1 is stalled. See details in SYSTIMER_TIMER_UNIT1_CORE1_STALL_EN. (R/W)

**SYSTIMER_TIMER_UNIT0_CORE0_STALL_EN** Configures whether UNIT0 is stalled when CORE0 is stalled. See details in SYSTIMER_TIMER_UNIT1_CORE1_STALL_EN. (R/W)

**Continued on the next page...**

Register 13.1. 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)

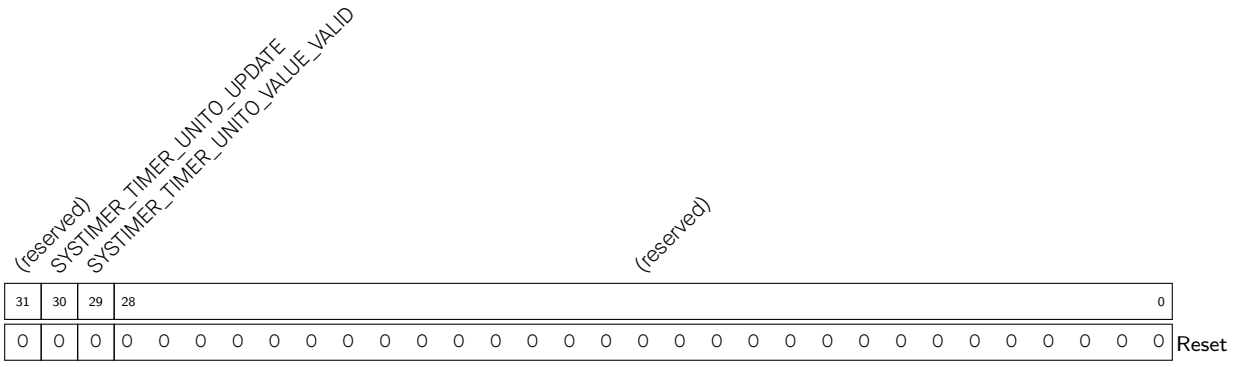
**SYSTIMER_CLK_EN** Configures register clock gating.

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 13.2. 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)

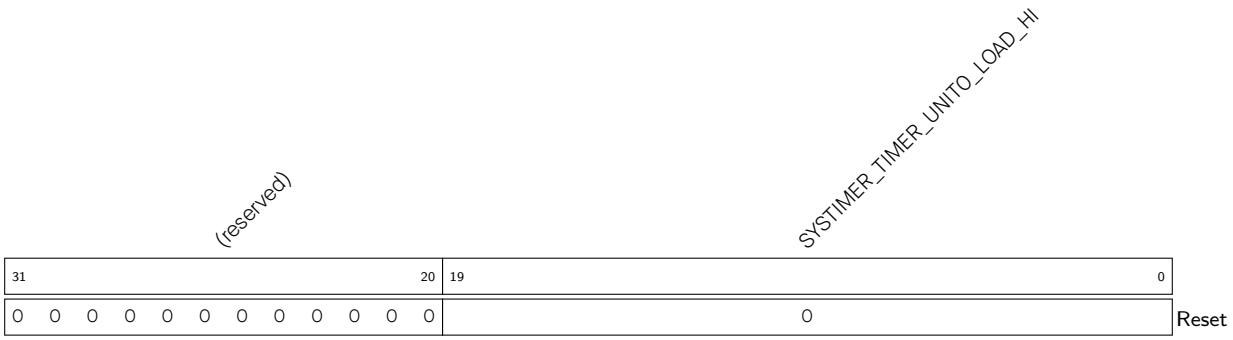
**SYSTIMER_TIMER_UNIT0_UPDATE** Configures whether to update timer UNIT0, i.e., reads the UNIT0 count value to SYSTIMER_TIMER_UNIT0_VALUE_HI and SYSTIMER_TIMER_UNIT0_VALUE_LO.

0: No effect

1: Update timer UNIT0

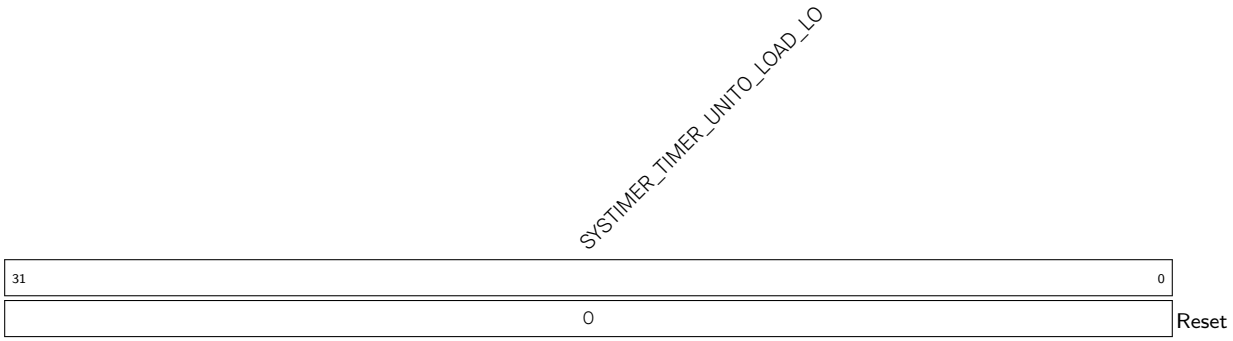
(WT)

Register 13.3. SYSTIMER_UNIT0_LOAD_HI_REG (0x000C)



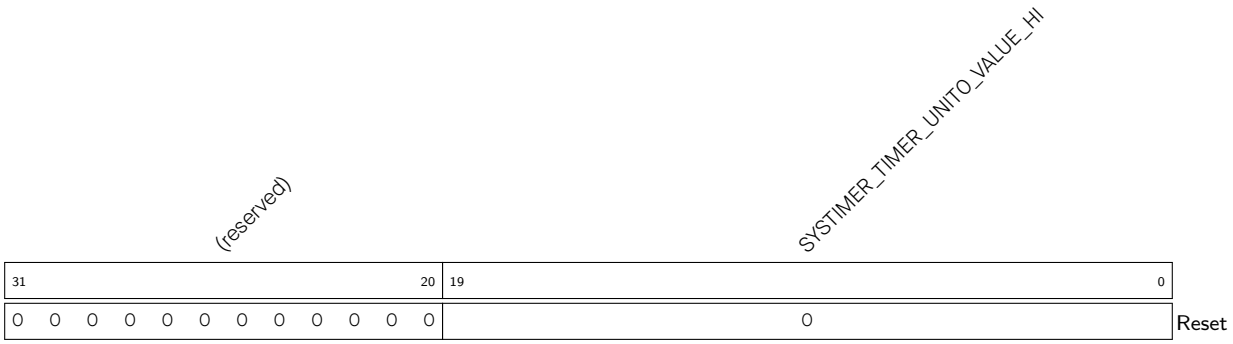

**SYSTIMER_TIMER_UNIT0_LOAD_HI** Configures the value to be loaded to UNIT0, high 20 bits. (R/W)

Register 13.4. SYSTIMER_UNIT0_LOAD_LO_REG (0x0010)



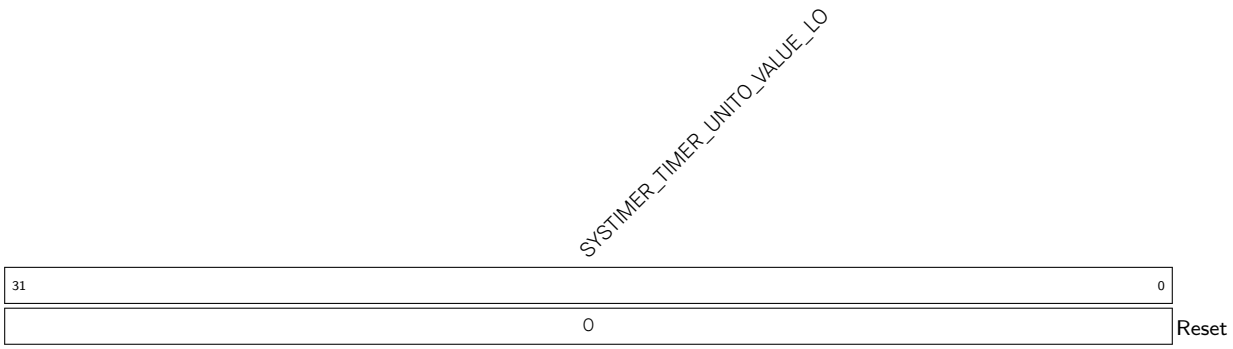

**SYSTIMER_TIMER_UNIT0_LOAD_LO** Configures the value to be loaded to UNIT0, low 32 bits. (R/W)

Register 13.5. SYSTIMER_UNIT0_VALUE_HI_REG (0x0040)



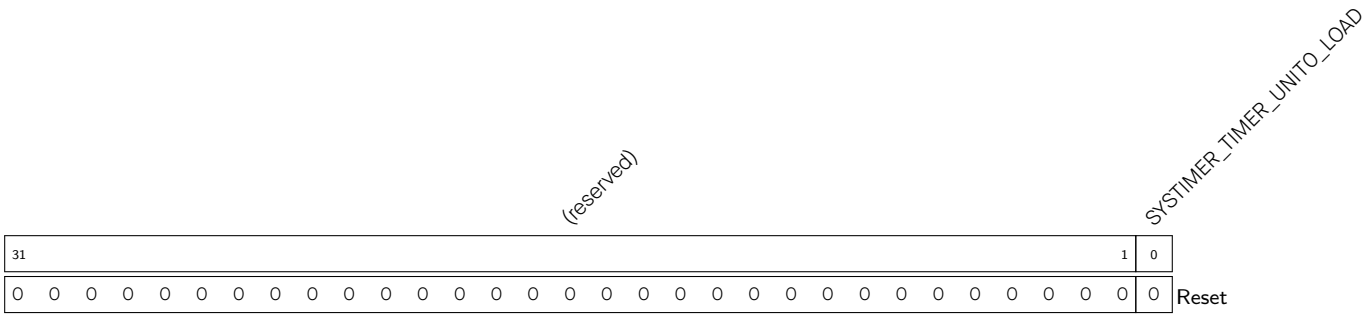

**SYSTIMER_TIMER_UNIT0_VALUE_HI** Represents UNIT0 read value, high 20 bits. (RO)

Register 13.6. SYSTIMER_UNIT0_VALUE_LO_REG (0x0044)




**SYSTIMER_TIMER_UNIT0_VALUE_LO** Represents UNIT0 read value, low 32 bits. (RO)

Register 13.7. SYSTIMER_UNIT0_LOAD_REG (0x005C)



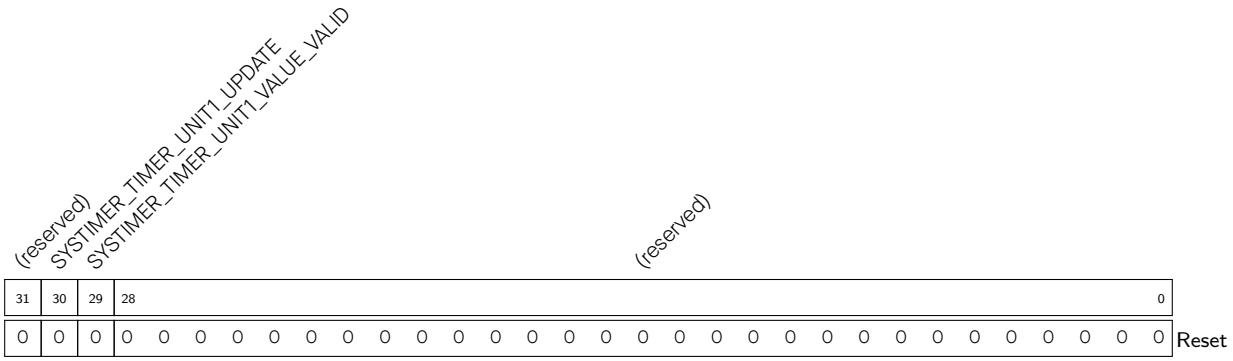

**SYSTIMER_TIMER_UNIT0_LOAD** Configures whether to reload the value of UNIT0, i.e., reloads the values of SYSTIMER_TIMER_UNIT0_VALUE_HI and SYSTIMER_TIMER_UNIT0_VALUE_LO to UNIT0.

0: No effect

1: Reload the value of UNIT0

(WT)

Register 13.8. SYSTIMER_UNIT1_OP_REG (0x0008)




**SYSTIMER_TIMER_UNIT1_VALUE_VALID** Represents UNIT1 value is synchronized and valid. (R/SS/WTC)

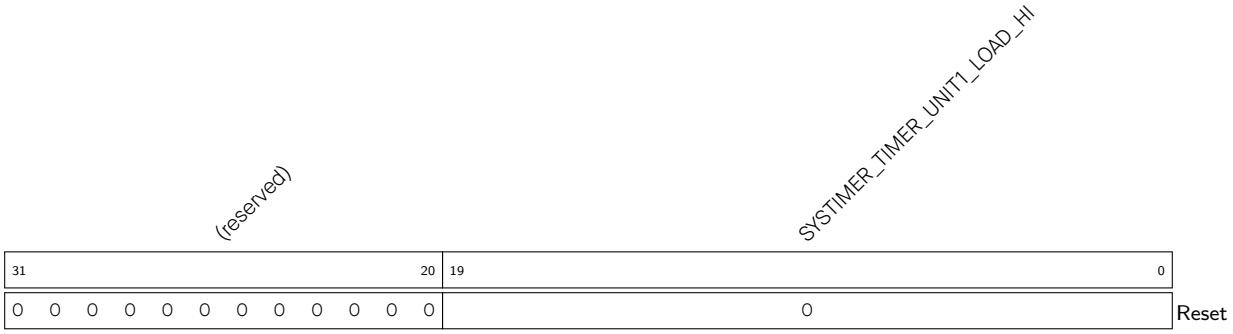
**SYSTIMER_TIMER_UNIT1_UPDATE** Configures whether to update timer UNIT1, i.e., reads the UNIT1 count value to SYSTIMER_TIMER_UNIT1_VALUE_HI and SYSTIMER_TIMER_UNIT1_VALUE_LO.

0: No effect

1: Update timer UNIT1

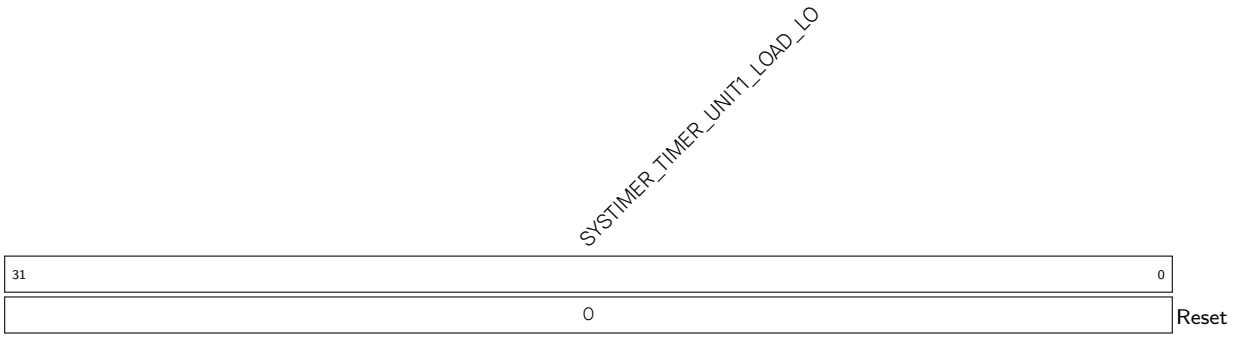
(WT)

Register 13.9. SYSTIMER_UNIT1_LOAD_HI_REG (0x0014)



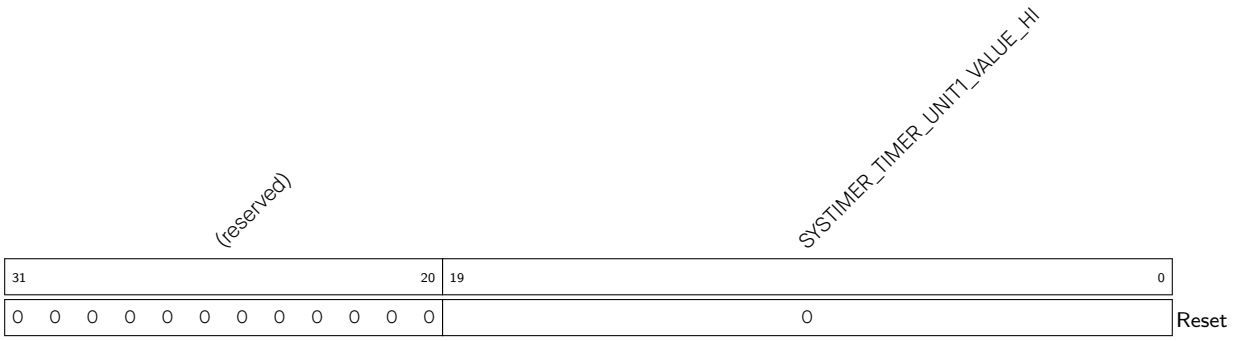

**SYSTIMER_TIMER_UNIT1_LOAD_HI** Configures the value to be loaded to UNIT1, high 20 bits. (R/W)

Register 13.10. SYSTIMER_UNIT1_LOAD_LO_REG (0x0018)



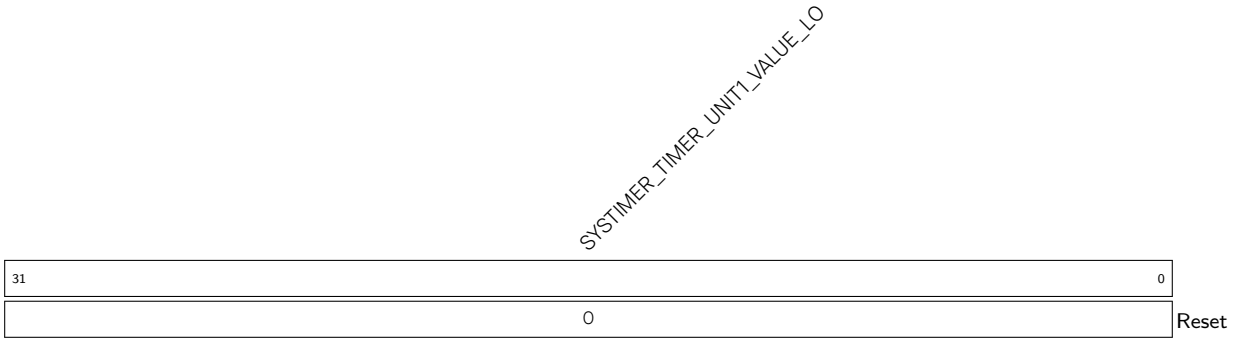

**SYSTIMER_TIMER_UNIT1_LOAD_LO** Configures the value to be loaded to UNIT1, low 32 bits. (R/W)

Register 13.11. SYSTIMER_UNIT1_VALUE_HI_REG (0x0048)



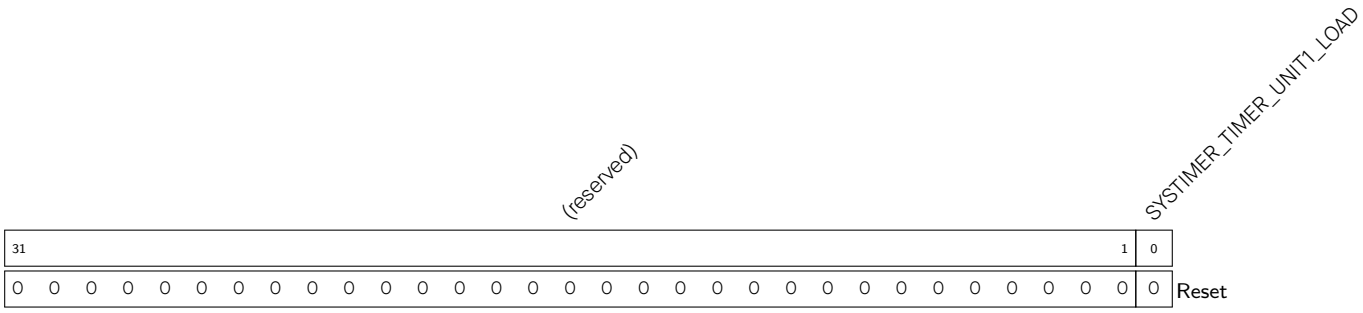

**SYSTIMER_TIMER_UNIT1_VALUE_HI** Represents UNIT1 read value, high 20 bits. (RO)

Register 13.12. SYSTIMER_UNIT1_VALUE_LO_REG (0x004C)



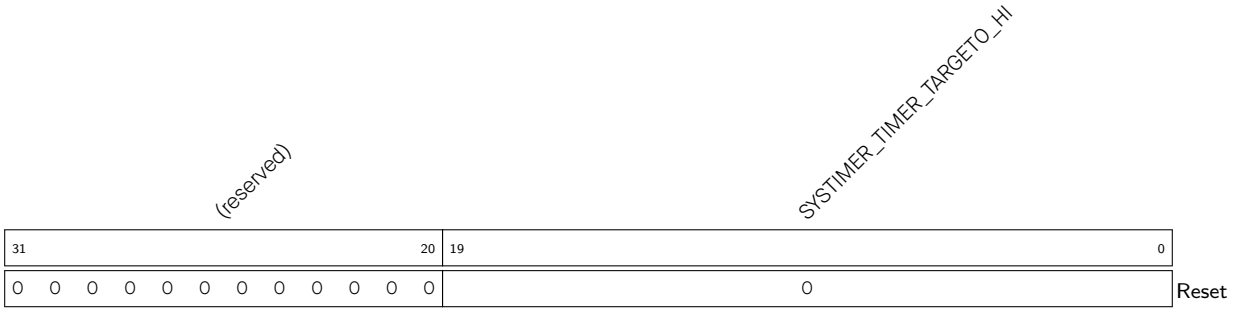

**SYSTIMER_TIMER_UNIT1_VALUE_LO** Represents UNIT1 read value, low 32 bits. (RO)

Register 13.13. SYSTIMER_UNIT1_LOAD_REG (0x0060)



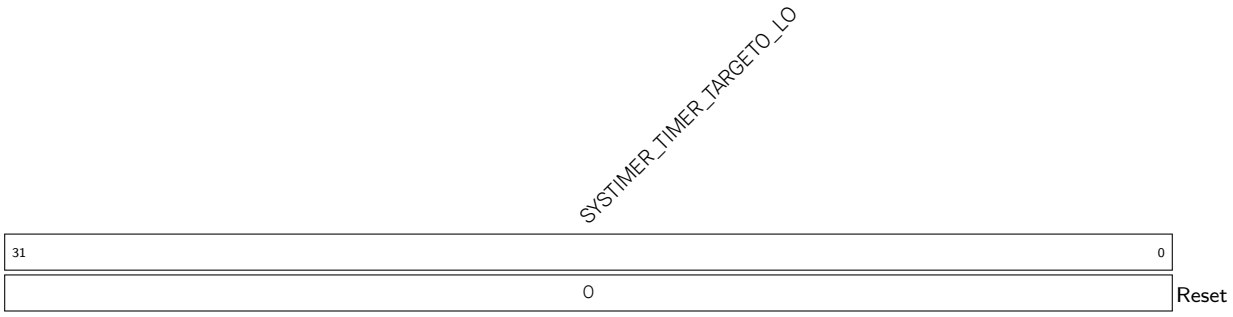

**SYSTIMER_TIMER_UNIT1_LOAD** Configures whether to reload the value of UNIT1, i.e., reload the values of SYSTIMER_TIMER_UNIT1_VALUE_HI and SYSTIMER_TIMER_UNIT1_VALUE_LO to UNIT1.
0: No effect
1: Reload the value of UNIT1
(WT)

Register 13.14. SYSTIMER_TARGET0_HI_REG (0x001C)



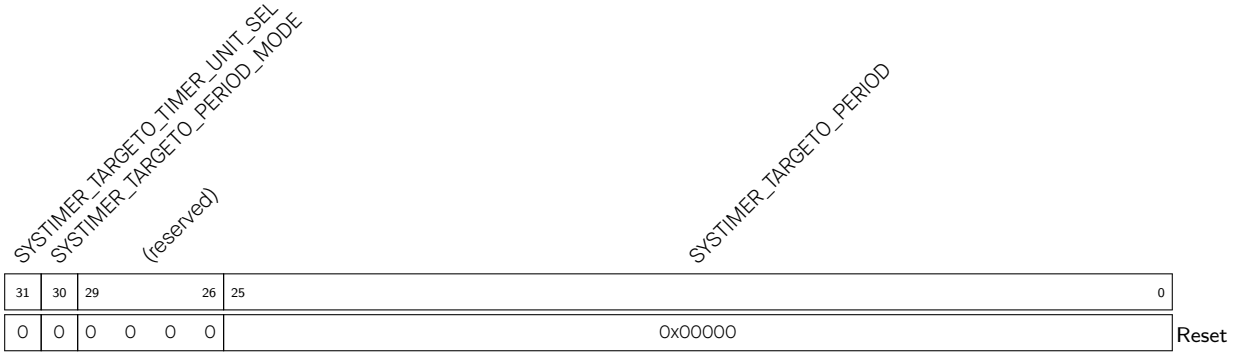

**SYSTIMER_TIMER_TARGET0_HI** Configures the alarm value to be loaded to COMP0, high 20 bits. (R/W)

Register 13.15. SYSTIMER_TARGET0_LO_REG (0x0020)




**SYSTIMER_TIMER_TARGET0_LO** Configures the alarm value to be loaded to COMP0, low 32 bits. (R/W)

Register 13.16. 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)

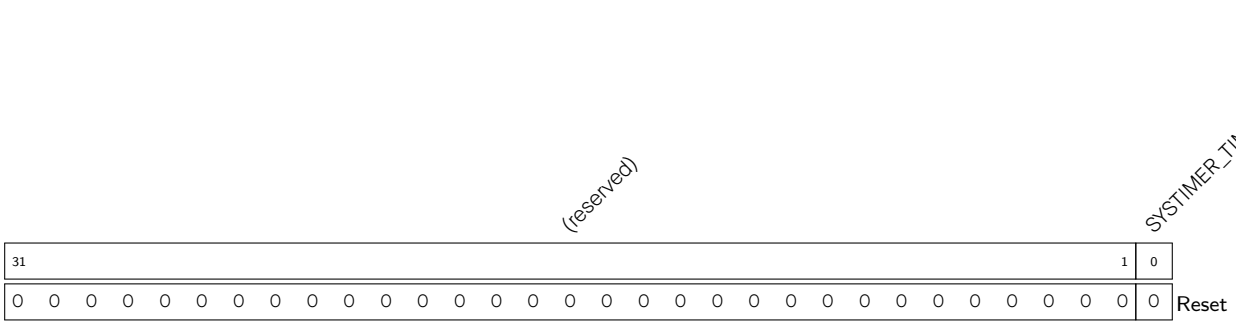
**SYSTIMER_TARGET0_TIMER_UNIT_SEL** Chooses the count value for comparison with COMP0.

0: Use the count value from UNIT*0*

1: Use the count value from UNIT*1*

(R/W)

Register 13.17. SYSTIMER_COMP0_LOAD_REG (0x0050)



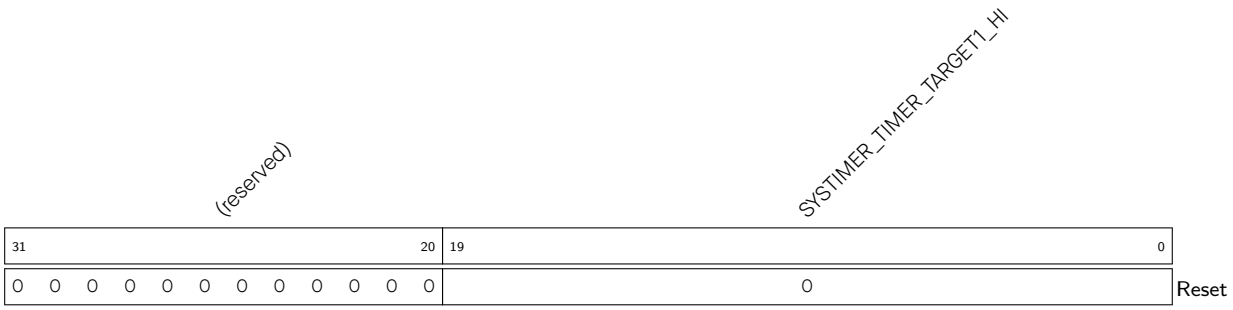

**SYSTIMER_TIMER_COMP0_LOAD** Configures whether to enable COMP0 synchronization, i.e., reload the alarm value/period to COMP0.

0: No effect

1: Enable COMP0 synchronization

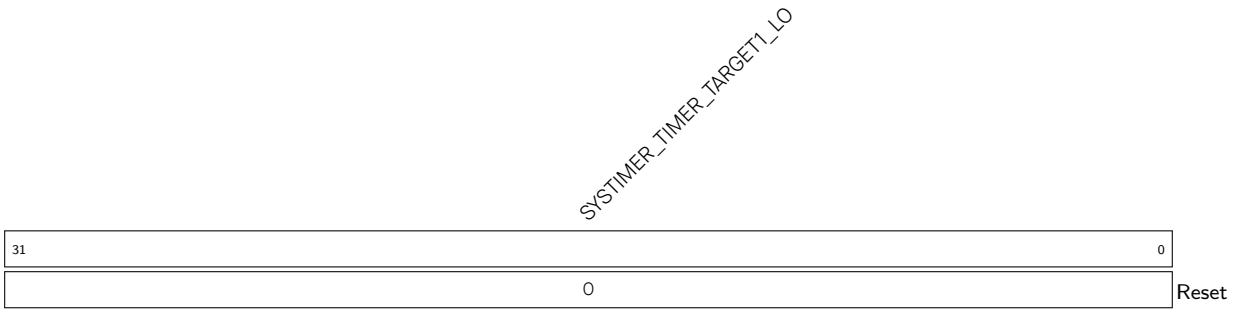
(WT)

Register 13.18. SYSTIMER_TARGET1_HI_REG (0x0024)



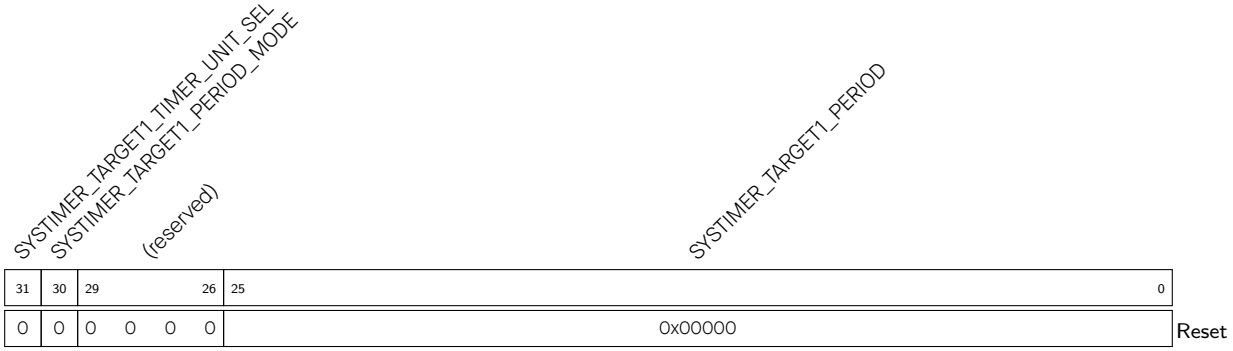

**SYSTIMER_TIMER_TARGET1_HI** Configures the alarm value to be loaded to COMP1, high 20 bits. (R/W)

Register 13.19. SYSTIMER_TARGET1_LO_REG (0x0028)




**SYSTIMER_TIMER_TARGET1_LO** Configures the alarm value to be loaded to COMP1, low 32 bits. (R/W)

Register 13.20. SYSTIMER_TARGET1_CONF_REG (0x0038)

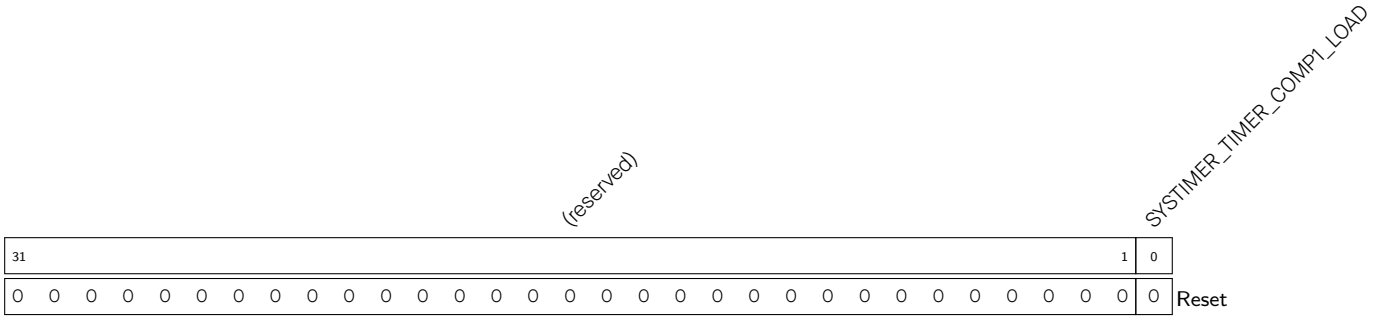



**SYSTIMER_TARGET1_PERIOD** Configures COMP1 alarm period. (R/W)

**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 13.21. SYSTIMER_COMP1_LOAD_REG (0x0054)



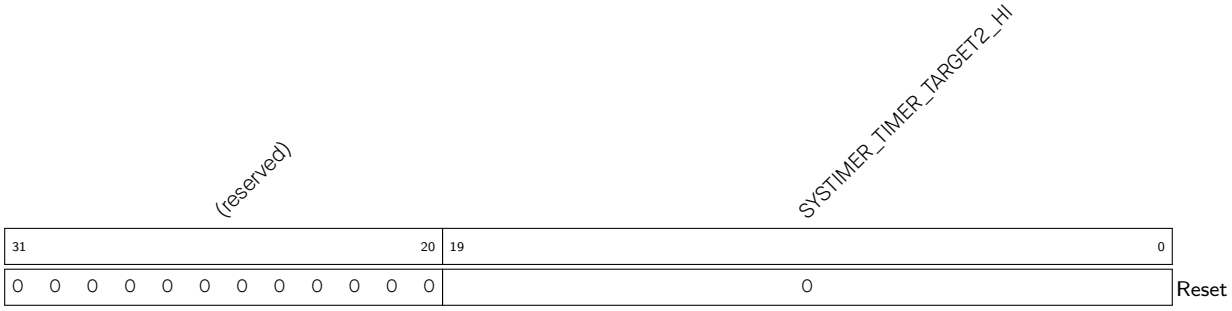

**SYSTIMER_TIMER_COMP1_LOAD** Configures whether to enable COMP1 synchronization, i.e., reload the alarm value/period to COMP1.

0: No effect

1: Enable COMP1 synchronization

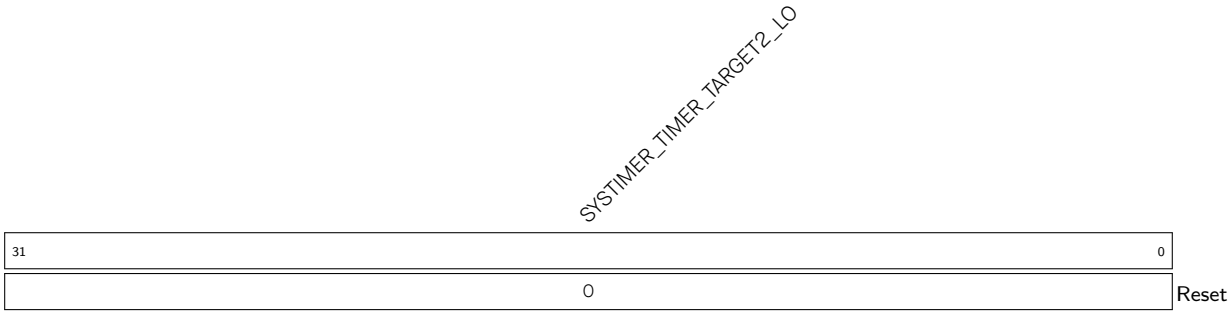
(WT)

Register 13.22. SYSTIMER_TARGET2_HI_REG (0x002C)



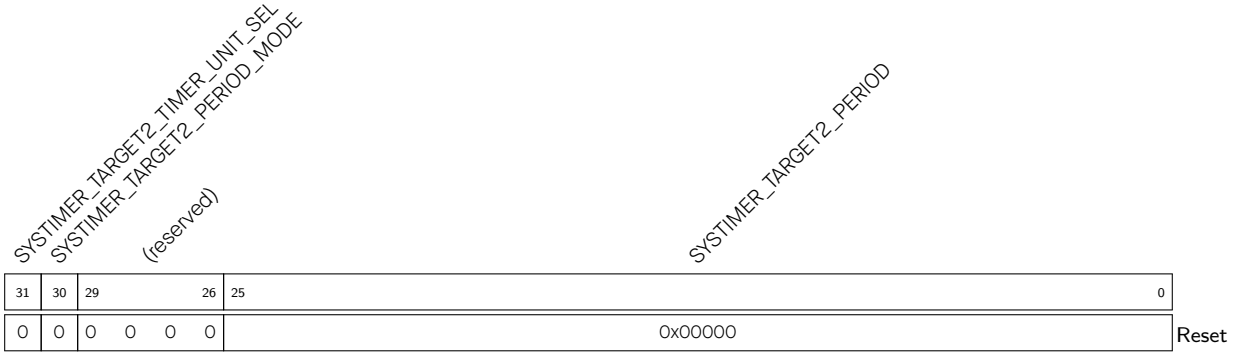

**SYSTIMER_TIMER_TARGET2_HI** Configures the alarm value to be loaded to COMP2, high 20 bits. (R/W)

Register 13.23. SYSTIMER_TARGET2_LO_REG (0x0030)




**SYSTIMER_TIMER_TARGET2_LO** Configures the alarm value to be loaded to COMP2, low 32 bits. (R/W)

Register 13.24. SYSTIMER_TARGET2_CONF_REG (0x003C)

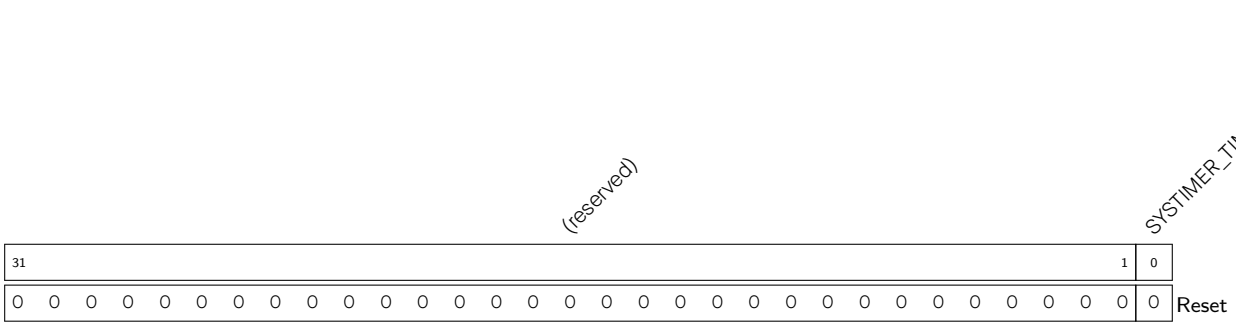



**SYSTIMER_TARGET2_PERIOD** Configures COMP2 alarm period. (R/W)

**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 13.25. SYSTIMER_COMP2_LOAD_REG (0x0058)



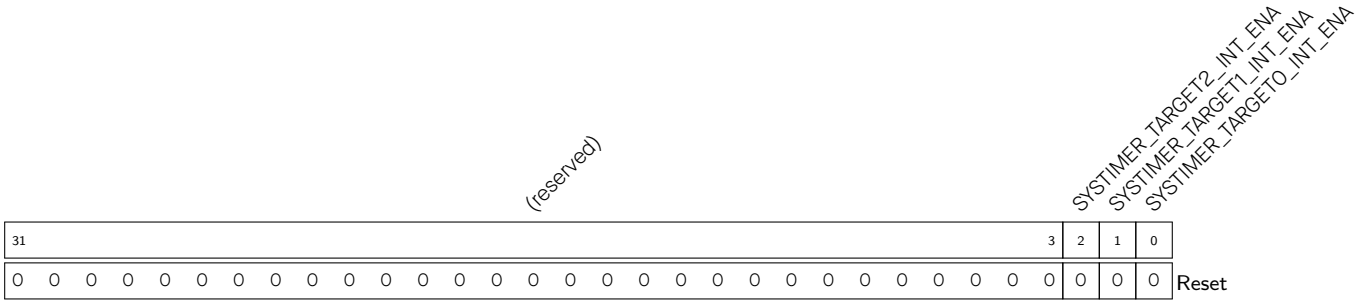

**SYSTIMER_TIMER_COMP2_LOAD** Configures whether to enable COMP2 synchronization, i.e., reload the alarm value/period to COMP2.

0: No effect

1: Enable COMP2 synchronization

(WT)

Register 13.26. SYSTIMER_INT_ENA_REG (0x0064)

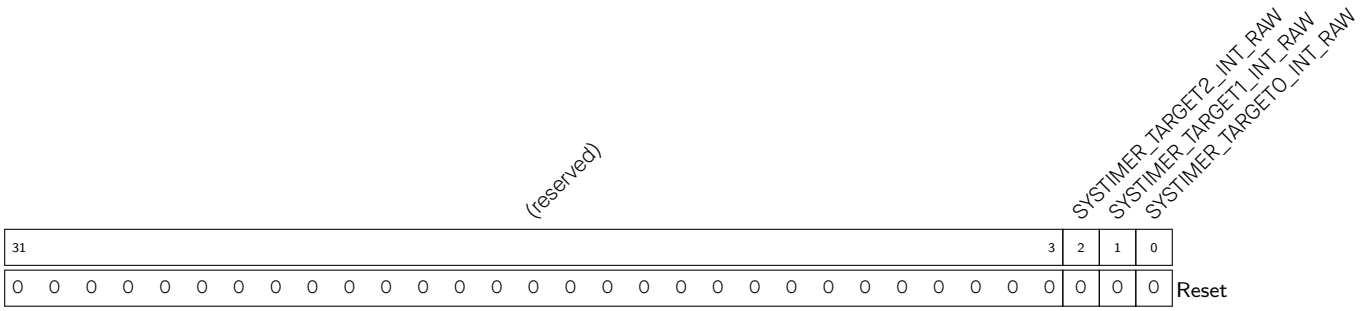



**SYSTIMER_TARGET0_INT_ENA** Write 1 to enable SYSTIMER_TARGET0_INT. (R/W)

**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 13.27. SYSTIMER_INT_RAW_REG (0x0068)

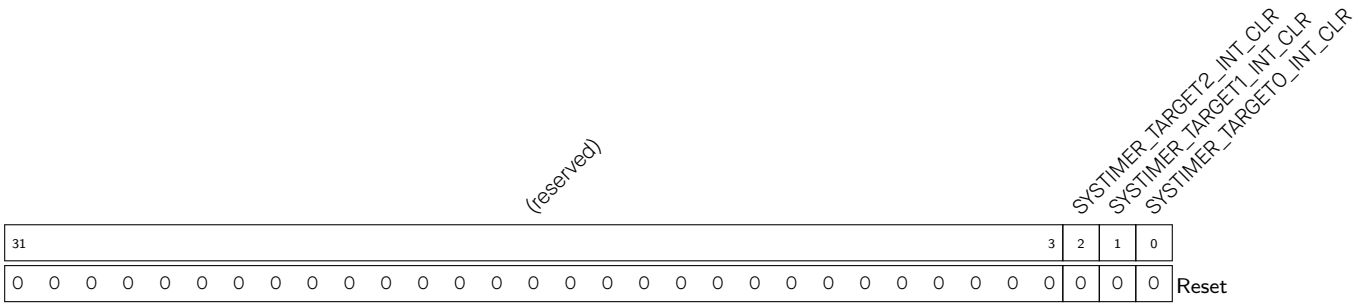



**SYSTIMER_TARGET0_INT_RAW** The raw interrupt status of SYSTIMER_TARGET0_INT. (R/WTC/SS)

**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 13.28. SYSTIMER_INT_CLR_REG (0x006C)

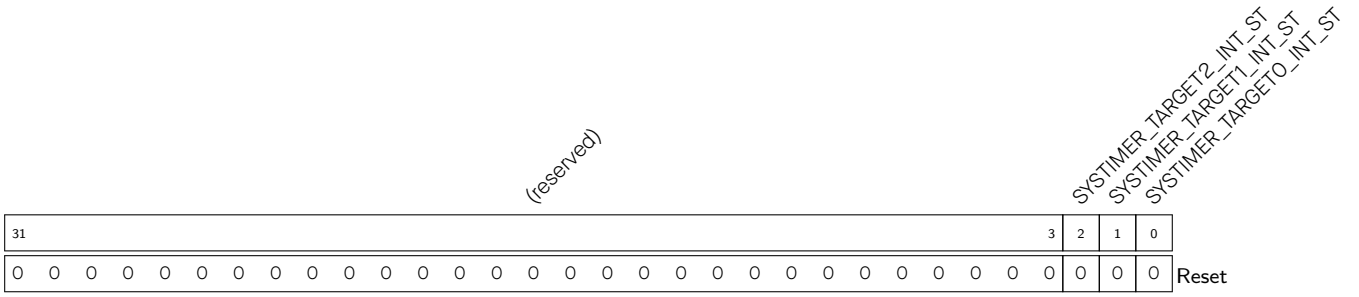



**SYSTIMER_TARGET0_INT_CLR** Write 1 to clear SYSTIMER_TARGET0_INT. (WT)

**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 13.29. SYSTIMER_INT_ST_REG (0x0070)

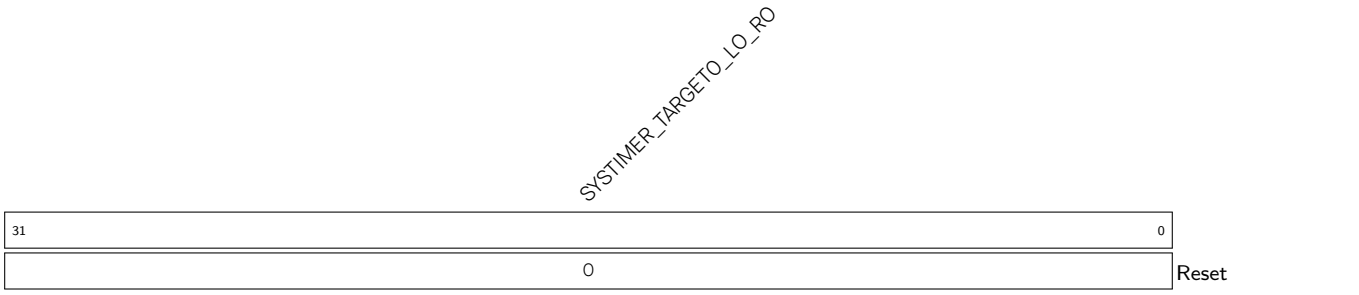



**SYSTIMER_TARGET0_INT_ST** The masked interrupt status of SYSTIMER_TARGET0_INT. (RO)

**SYSTIMER_TARGET1_INT_ST** The masked interrupt status of SYSTIMER_TARGET1_INT. (RO)

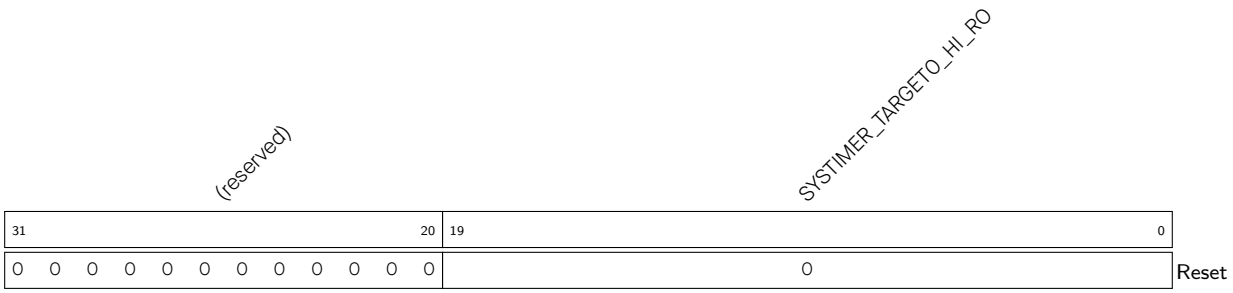
**SYSTIMER_TARGET2_INT_ST** The masked interrupt status of SYSTIMER_TARGET2_INT. (RO)

Register 13.30. SYSTIMER_REAL_TARGET0_LO_REG (0x0074)




**SYSTIMER_TARGET0_LO_RO** Represents the actual target value of COMP0, low 32 bits. (RO)

Register 13.31. SYSTIMER_REAL_TARGET0_HI_REG (0x0078)



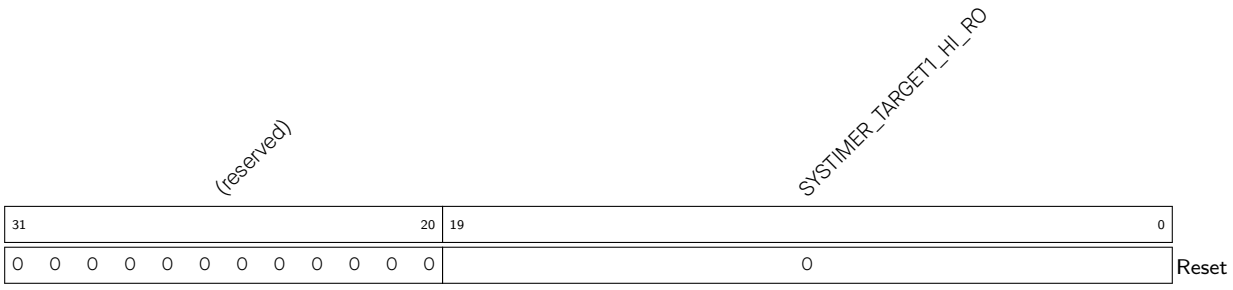

**SYSTIMER_TARGET0_HI_RO** Represents the actual target value of COMP0, high 20 bits. (RO)

Register 13.32. SYSTIMER_REAL_TARGET1_LO_REG (0x007C)




**SYSTIMER_TARGET1_LO_RO** Represents the actual target value of COMP1, low 32 bits. (RO)

Register 13.33. SYSTIMER_REAL_TARGET1_HI_REG (0x0080)




**SYSTIMER_TARGET1_HI_RO** Represents the actual target value of COMP1, high 20 bits. (RO)

Register 13.34. SYSTIMER_REAL_TARGET2_LO_REG (0x0084)



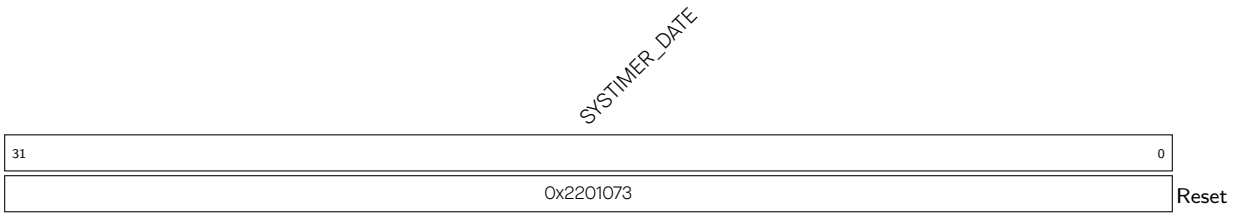

**SYSTIMER_TARGET2_LO_RO** Represents the actual target value of COMP2, low 32 bits. (RO)

Register 13.35. SYSTIMER_REAL_TARGET2_HI_REG (0x0088)




**SYSTIMER_TARGET2_HI_RO** Represents the actual target value of COMP2, high 20 bits. (RO)

Register 13.36. SYSTIMER_DATE_REG (0x00FC)




**SYSTIMER_DATE** Version control register. (R/W)

# Chapter 14

# Timer Group (TIMG)

## 14.1 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 14.1-1, the ESP32-C61 chip contains two timer groups, namely timer group 0 and timer group 1. Each timer group consists of one general-purpose timer referred to as T0 and one Main System Watchdog Timer. The general-purpose timer is based on a 16-bit prescaler and a 54-bit auto-reload-capable up-down counter.

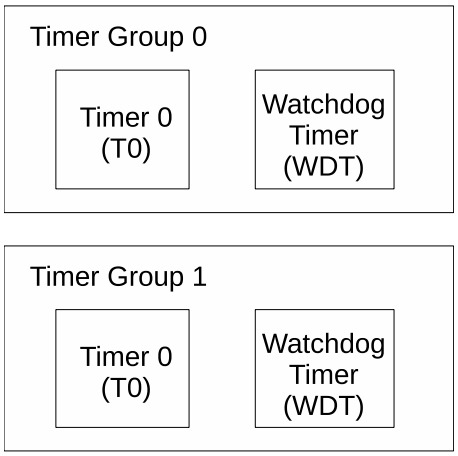



**Figure 14.1-1. Timer Group Overview**

Note that while the Main System Watchdog Timer registers are described in this chapter, their functional description is included in Chapter 15 *Watchdog Timers (WDT)*. Therefore, the term "timer" within this chapter refers to the general-purpose timer.

## 14.2 Feature List

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
- Frequency calculation of slow clock for timer group 0
- Level interrupt generation
- Support several ETM tasks and events

## 14.3 Architectural Overview

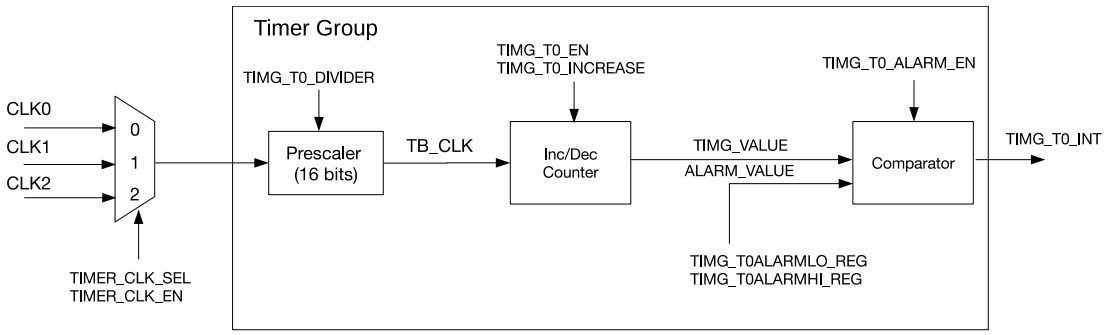



**Figure 14.3-1. Timer Group Architecture**

Figure 14.3-1 is a diagram of timer T0 in a timer group. T0 contains a 16-bit integer divider as a prescaler, a timer-based counter, and a comparator for alarm generation.

## 14.4 Functional Description

### 14.4.1 16-bit Prescaler and Clock Selection

Take the T0 in timer group 0 as an example:

- The timer can select its clock source by setting the PCR_TG0_TIMER_CLK_SEL field of the PCR_TIMERGROUP0_TIMER_CLK_CONF_REG register. When the field is 0, CLK0 is selected, which indicates XTAL_CLK; when the field is 1, CLK1 is selected, which indicates RC_FAST_CLK; when the field is 2, CLK2 is selected, which indicates PLL_F80M_CLK.
- The selected clock can be switched on by setting PCR_TG0_TIMER_CLK_EN field of the PCR_TIMERGROUP0_TIMER_CLK_CONF_REG register to 1 and switched off by setting it to 0. This field is indicated as TIMER_CLK_EN in Figure 14.3-1. 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_T0_DIVIDER field.

TIMG_T0_DIVIDER field can be configured as 0 ~ 65535 for a divisor range of 2 ~ 65536. To be more specific, when TIMG_T0_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_TO_DIVIDER field, and then set TIMG_TO_DIVCNT_RST to 1. Meanwhile, the timer must be disabled (i.e., TIMG_TO_EN should be cleared). Otherwise, the result can be unpredictable.

### 14.4.2 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_TO_INCREASE field. The time-base counter can be enabled or disabled by setting or clearing the TIMG_TO_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_TO_INCREASE field can be changed no matter whether TIMG_TO_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_TOUPDATE_REG, the current value of the 54-bit timer starts to be latched into the TIMG_TOLO_REG and TIMG_TOHI_REG registers containing the lower 32-bits and higher 22-bits, respectively. When TIMG_TOUPDATE_REG is cleared by hardware, it indicates the latch operation has been completed and current timer value can be read from the TIMG_TOLO_REG and TIMG_TOHI_REG registers. TIMG_TOLO_REG and TIMG_TOHI_REG registers will remain unchanged for the CPU to read in its own time until TIMG_TOUPDATE_REG is written to again.

### 14.4.3 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 14.4.4).

The 54-bit alarm value is configured using TIMG_TOALARMLO_REG and TIMG_TOALARMHI_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_TO_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 14.4-1 and Table 14.4-2 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_TOHI_REG, TIMG_TOLO_REG}
- ALARM_VALUE = {TIMG_TOALARMHI_REG, TIMG_TOALARMLO_REG}

Table 14.4-1. 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 14.4-2. 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_TO_ALARM_EN field is automatically cleared and no alarm will occur again until the TIMG_TO_ALARM_EN is set next time.

### 14.4.4 Timer Reload

A timer is reloaded when a timer's current value is overwritten with a reload value stored in the TIMG_TO_LOAD_LO and TIMG_TO_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_TO_LOAD_LO and TIMG_TO_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_TOLOAD_REG, which causes the timer's current value to be instantly reloaded. If TIMG_TO_EN is set, the timer will continue incrementing or decrementing from the new value. In this case, if TIMG_TO_ALARM_EN is set, the timer will still trigger alarms in scenarios listed in Table 14.4-1 and 14.4-2. If TIMG_TO_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_TO_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.

### 14.4.5 Frequency Calculation of Slow Clock for Timer Group 0

Using XTAL_CLK as a reference, it is possible to calculate the frequency of slow clock sources provided by ESP32-C61. For the slow clock inputted to the timer group, please refer to 10 *Interrupt Matrix* > 10.2 *Interrupt Terminology in ESP32-C61*. The calculation method is as follows:

1. Start periodic or one-shot frequency calculation (see Section 14.7.5 for details);
2. Once receiving the signal to start calculation, the counter of XTAL_CLK and the counter of slow clock begin to work at the same time. When the counter of slow clock counts to C0, the two counters stop counting simultaneously;
3. Assume the value of XTAL_CLK's counter is C1, and the frequency of slow clock would be calculated as: $f\_rtc = \frac{C0 \times f\_XTAL\_CLK}{C1}$

Please note that the input frequency should not be larger than $\frac{f\_XTAL\_CLK}{2}$ when calculating the frequency of slow clock.

## 14.5 Event Task Matrix Feature

The timer groups on ESP32-C61 support the Event Task Matrix (ETM) function, which allows timer groups' ETM tasks to be triggered by any peripherals' ETM events, or timer groups' ETM events to trigger any peripherals' ETM tasks. This section introduces the ETM tasks and events related to timer groups. For more information, please refer to Chapter 11 Event Task Matrix (ETM).

The timer groups 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: When triggered, it will 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 configured through APB method 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_ALARM_EN: When triggered, it will 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.

The timer groups 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.

All the ETM tasks and events will not take effect until the TIMG_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 14.7.4 Timer as Periodic Alarm by ETM.

## 14.6 Interrupts

ESP32-C61's TIMG*n* can generate the following interrupt signal(s) that will be sent to the Interrupt Matrix.

- TG*n*_T0_INTR
- TG*n*_WDT_INTR

There are several internal interrupt sources from TIMG*n* that can generate the above interrupt signals. The interrupt sources from TIMG*n* are listed with their trigger conditions and the resulted interrupt signals in Table 14.6-1.

**Table 14.6-1. TIMG*n*'s Internal Interrupt Sources**

| Internal Interrupt Source | Trigger Condition | Interrupt Signal |
|---|---|---|
| TIMG_T0_INT | T0 generates an alarm | TG*n*_T0_INTR |
| TIMG_WDT_INT | The watchdog timer generates an alarm | TG*n*_WDT_INTR |

**Note:**
For definitions of *interrupt*, *interrupt signal*, *interrupt source*, and their correlations, please refer to Chapter 10 *Interrupt Matrix* > Section 10.2 *Interrupt Terminology in ESP32-C61*.

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 14.8 *Register Summary*.

## 14.7 Programming Procedures

### 14.7.1 Timer as a Simple Clock

1. Configure the time-base counter
   - Select clock source by setting or clearing PCR_TGO_TIMER_CLK_SEL field.
   - Configure the 16-bit prescaler by setting TIMG_TO_DIVIDER.
   - Configure the timer direction by setting or clearing TIMG_TO_INCREASE.
   - Set the timer's starting value by writing the starting value to TIMG_TO_LOAD_LO and TIMG_TO_LOAD_HI, then reloading it into the timer by writing any value to TIMG_TOLOAD_REG.
2. Start the timer by setting TIMG_TO_EN.
3. Get the timer's current value.
   - Write any value to TIMG_TOUPDATE_REG to latch the timer's current value.
   - Wait until TIMG_TOUPDATE_REG is cleared by hardware.
   - Read the latched timer value from TIMG_TOLO_REG and TIMG_TOHI_REG.

### 14.7.2 Timer as One-shot Alarm

1. Configure the time-base counter following step 1 of Section 14.7.1.
2. Configure the alarm.
   - Configure the alarm value by setting TIMG_TOALARMLO_REG and TIMG_TOALARMHI_REG.
   - Enable interrupt by setting TIMG_TO_INT_ENA.
3. Disable auto reload by clearing TIMG_TO_AUTORELOAD.
4. Start the alarm by setting TIMG_TO_ALARM_EN.
5. Handle the alarm interrupt.
   - Clear the interrupt by setting the timer's corresponding bit in TIMG_TO_INT_CLR.
   - Disable the timer by clearing TIMG_TO_EN.

### 14.7.3 Timer as Periodic Alarm by APB

1. Configure the time-base counter following step 1 in Section 14.7.1.

2. Configure the alarm following step 2 in Section 14.7.2.
3. Enable auto reload by setting TIMG_T0_AUTORELOAD and configure the reload value via TIMG_T0_LOAD_LO and TIMG_T0_LOAD_HI.
4. Start the alarm by setting TIMG_T0_ALARM_EN.
5. Handle the alarm interrupt (repeat on each alarm iteration).
   - Clear the interrupt by setting the timer's corresponding bit in TIMG_T0_INT_CLR.
   - If the next alarm requires a new alarm value and reload value (i.e., different alarm interval per iteration), then TIMG_T0ALARMLO_REG, TIMG_T0ALARMHI_REG, TIMG_T0_LOAD_LO, and TIMG_T0_LOAD_HI should be reconfigured as needed. Otherwise, the aforementioned registers should remain unchanged.
   - Re-enable the alarm by setting TIMG_T0_ALARM_EN.
6. Stop the timer (on final alarm iteration).
   - Clear the interrupt by setting the timer's corresponding bit in TIMG_T0_INT_CLR.
   - Disable the timer by clearing TIMG_T0_EN.

### 14.7.4 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 14.7.1.
6. Configure the alarm following step 2 in Section 14.7.2.
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_T0_INT_CLR.
   - Disable the timer by clearing TIMG_T0_EN.

### 14.7.5 Frequency Calculation of Slow Clock

1. One-shot frequency calculation
   - Select the clock whose frequency is to be calculated via PCR_32_SEL. For the relationship between the value of this register and the clock source, please refer to Chapter 16 *RTC Timer*.
   - To meet the prerequisites for calculating the slow clock frequency, prescale RC_FAST_CLK via PCR_FOSC_TICK_NUM.
   - 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 slow clock according to the formula in Section 14.4.5.
2. Periodic frequency calculation
   - Select the clock whose frequency is to be calculated via PCR_32_SEL.
   - To meet the prerequisites for calculating the slow clock frequency, prescale RC_FAST_CLK via PCR_FOSC_TICK_NUM.
   - 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 slow clock cannot finish counting in TIMG_RTC_CALI_TIMEOUT_RST_CNT cycles, TIMG_RTC_CALI_TIMEOUT will be set to indicate a timeout.

## 14.8 Register Summary

The addresses in this section are relative to Timer Group base address provided in Table 5.3-2 in Chapter 5 *System and Memory*.

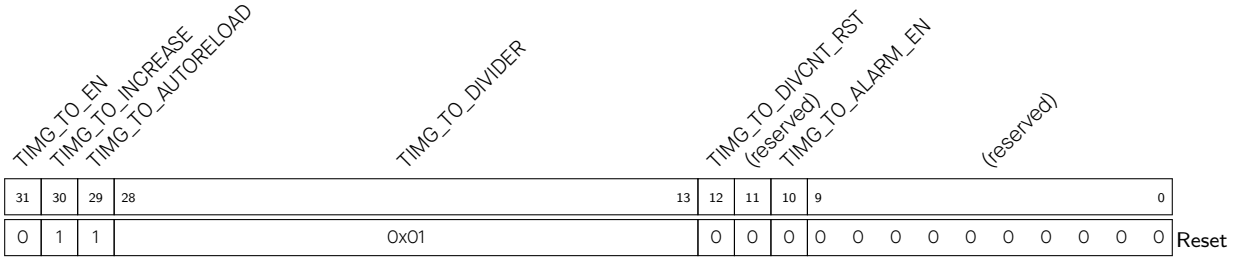
The abbreviations given in Column **Access** are explained in Section *Access Types for Registers*.

| 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 |
| **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 |
| **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 |

## 14.9 Registers

The addresses in this section are relative to Timer Group base address provided in Table 5.3-2 in Chapter 5 *System and Memory*.

**Register 14.1. TIMG_T0CONFIG_REG (0x0000)**




**TIMG_T0_ALARM_EN** Configures whether or not to enable the timer 0 alarm function. This bit will be automatically cleared once an alarm occurs.

0: Disable

1: Enable

(R/W/SC)

**TIMG_T0_DIVCNT_RST** Configures whether or not to reset the timer 0 's clock divider counter.

0: No effect

1: Reset

(WT)

**TIMG_T0_DIVIDER** Represents the timer 0 clock (T0_clk) prescaler value. (R/W)

**TIMG_T0_AUTORELOAD** Configures whether or not to enable the timer 0 auto-reload function at the time of alarm.

0: No effect

1: Enable

(R/W)

**TIMG_T0_INCREASE** Configures the counting direction of the timer 0 time-base counter.

0: Decrement

1: Increment

(R/W)

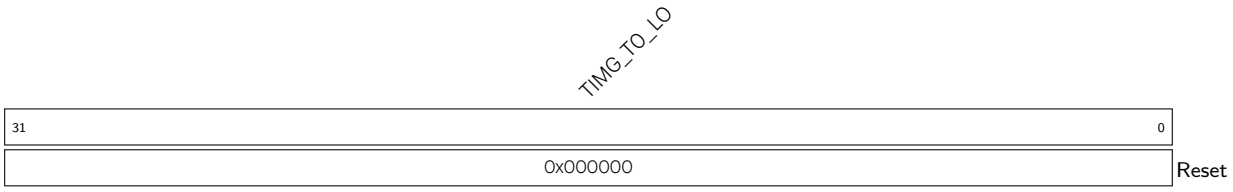
**TIMG_T0_EN** Configures whether or not to enable the timer 0 time-base counter.

0: Disable

1: Enable

(R/W/SS/SC)

Register 14.2. TIMG_T0LO_REG (0x0004)

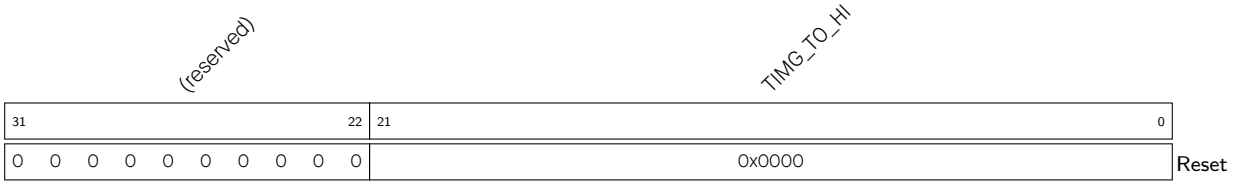



**TIMG_T0_LO** Represents the low 32 bits of the time-base counter of timer 0. Valid only after writing to TIMG_T0UPDATE_REG.

Measurement unit: T0_clk

(RO)

Register 14.3. TIMG_T0HI_REG (0x0008)

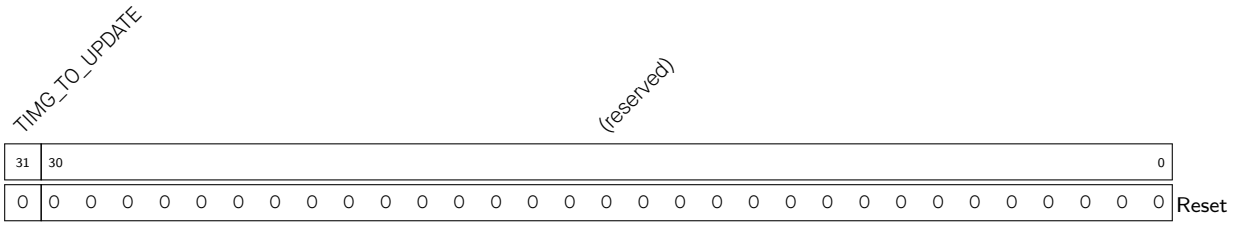



**TIMG_T0_HI** Represents the high 22 bits of the time-base counter of timer 0. Valid only after writing to TIMG_T0UPDATE_REG.

Measurement unit: T0_clk

(RO)

Register 14.4. TIMG_T0UPDATE_REG (0x000C)




**TIMG_T0_UPDATE** Configures to latch the counter value.

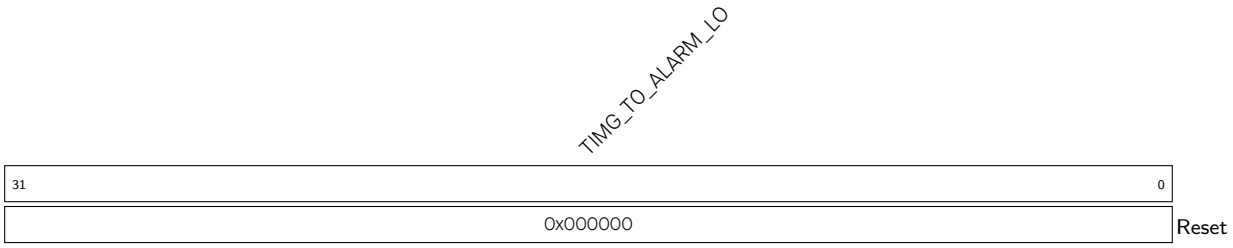
0: Latch

1: Latch

Note that writing either 0 or 1 to this bit will trigger latching of the counter value.

(R/W/SC)

Register 14.5. TIMG_T0ALARMLO_REG (0x0010)

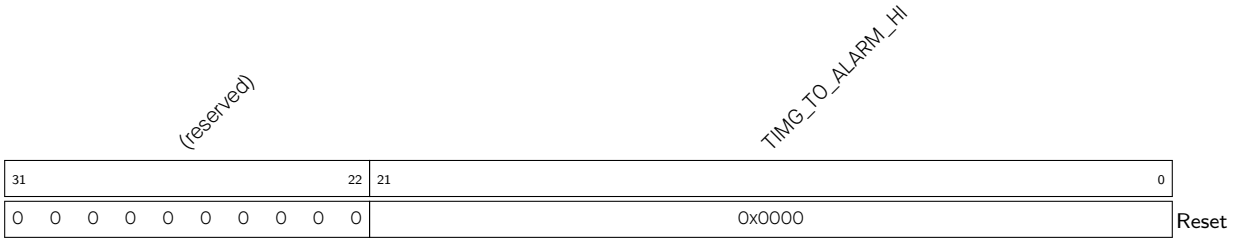



**TIMG_T0_ALARM_LO** Configures the low 32 bits of timer 0 alarm trigger time-base counter value. Valid only when TIMG_T0_ALARM_EN is 1.

Measurement unit: T0_clk

(R/W)

Register 14.6. TIMG_T0ALARMHI_REG (0x0014)

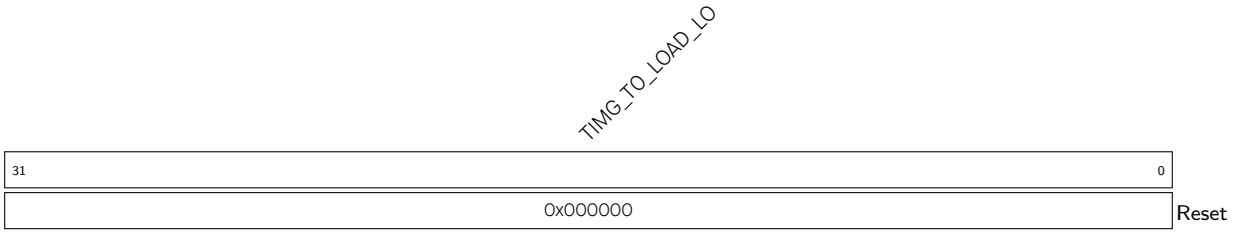



**TIMG_T0_ALARM_HI** Configures the high 22 bits of timer 0 alarm trigger time-base counter value. Valid only when TIMG_T0_ALARM_EN is 1.

Measurement unit: T0_clk

(R/W)

Register 14.7. TIMG_T0LOADLO_REG (0x0018)

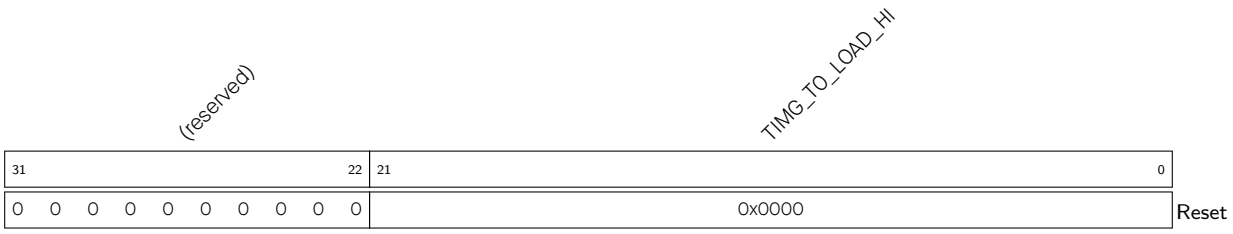



**TIMG_T0_LOAD_LO** Configures low 32 bits of the value that a reload will load onto timer 0 time-base counter.

Measurement unit: T0_clk

(R/W)

Register 14.8. TIMG_T0LOADHI_REG (0x001C)

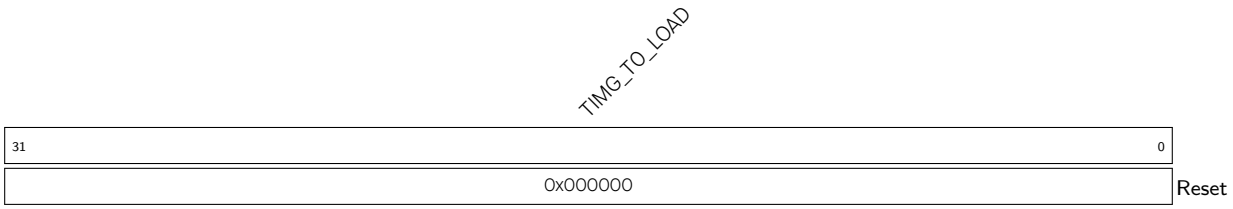



**TIMG_T0_LOAD_HI** Configures high 22 bits of the value that a reload will load onto timer 0 time-base counter.

Measurement unit: T0_clk

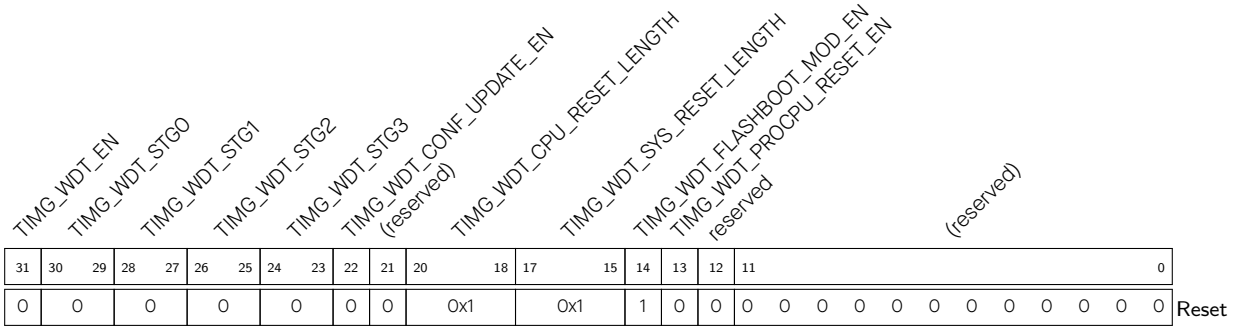
(R/W)

Register 14.9. TIMG_T0LOAD_REG (0x0020)




**TIMG_T0_LOAD** Write any value to trigger a timer 0 time-base counter reload. (WT)

**Register 14.10. TIMG_WDTCONFIG0_REG (0x0048)**




**TIMG_WDT_PROCPU_RESET_EN** Configures whether to mask the 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: 8

1: 16

2: 24

3: 32

4: 40

5: 64

6: 128

7: 256

(R/W)

**TIMG_WDT_CPU_RESET_LENGTH** Configures the CPU reset signal length. Valid only when write protection is disabled.

Measurement unit: mwdt_clk

0: 8

1: 16

2: 24

3: 32

4: 40

5: 64

6: 128

7: 256

(R/W)

Continued on the next page...

Register 14.10. USB_SERIAL_JTAG_CONF0_REG (0x0018)

Continued from the previous page...

**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. See details in TIMG_WDT_STG0. Valid only when write protection is disabled. (R/W)

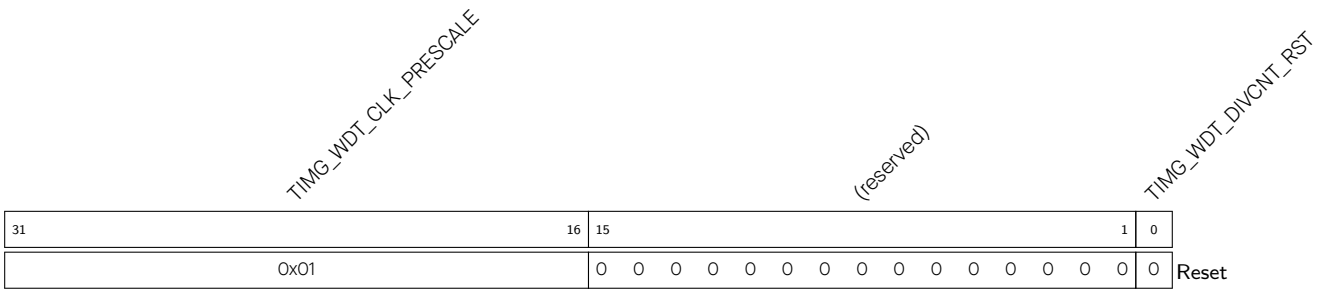
**TIMG_WDT_STG2** Configures the timeout action of stage 2. See details in TIMG_WDT_STG0. Valid only when write protection is disabled. (R/W)

**TIMG_WDT_STG1** Configures the timeout action of stage 1. See details in TIMG_WDT_STG0. Valid only when write protection is disabled. (R/W)

**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 14.11. TIMG_WDTCONFIG1_REG (0x004C)




**TIMG_WDT_DIVCNT_RST** Configures whether to reset WDT 's clock divider counter.

0: No effect

1: Reset

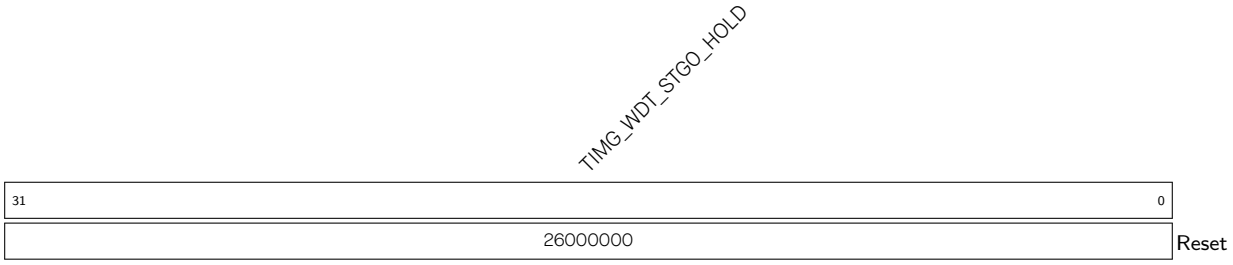
(WT)

**TIMG_WDT_CLK_PRESCALE** Configures MWDT clock prescaler value. Valid only when write protection is disabled.

MWDT clock period = MWDT's clock source period * TIMG_WDT_CLK_PRESCALE.

(R/W)

Register 14.12. TIMG_WDTCONFIG2_REG (0x0050)

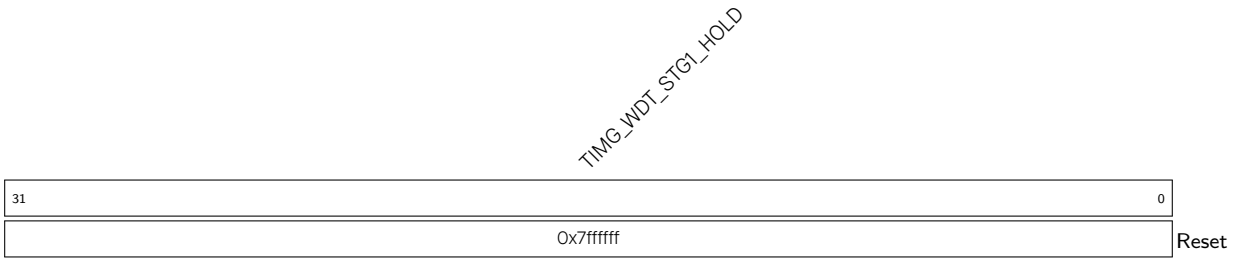



**TIMG_WDT_STG0_HOLD** Configures the stage 0 timeout value. Valid only when write protection is disabled.

Measurement unit: mwdt_clk

(R/W)

Register 14.13. TIMG_WDTCONFIG3_REG (0x0054)

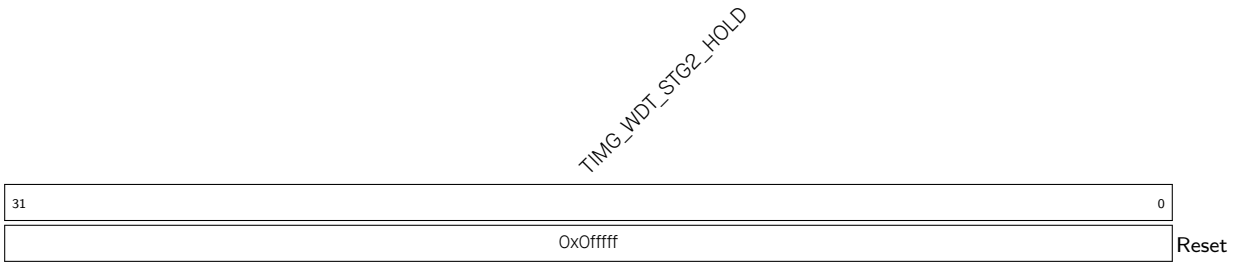



**TIMG_WDT_STG1_HOLD** Configures the stage 1 timeout value. Valid only when write protection is disabled.

Measurement unit: mwdt_clk

(R/W)

Register 14.14. TIMG_WDTCONFIG4_REG (0x0058)

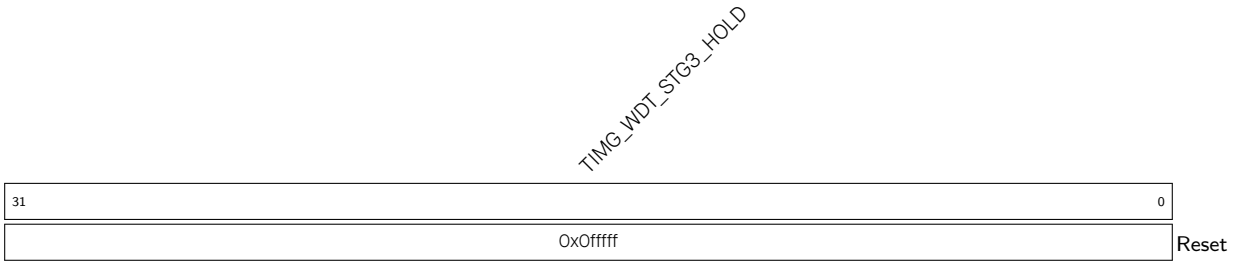



**TIMG_WDT_STG2_HOLD** Configures the stage 2 timeout value. Valid only when write protection is disabled.

Measurement unit: mwdt_clk

(R/W)

Register 14.15. TIMG_WDTCONFIG5_REG (0x005C)

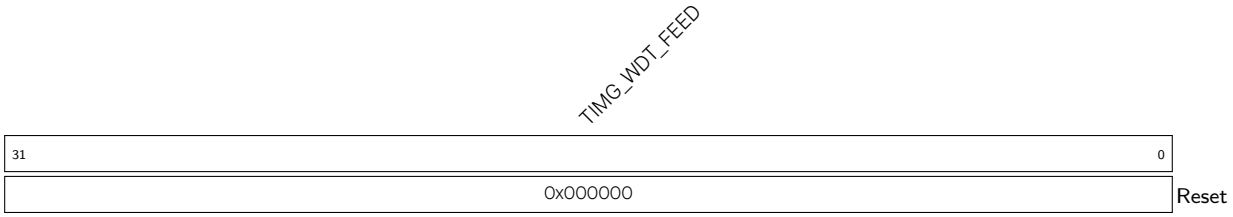



**TIMG_WDT_STG3_HOLD** Configures the stage 3 timeout value. Valid only when write protection is disabled.

Measurement unit: mwdt_clk

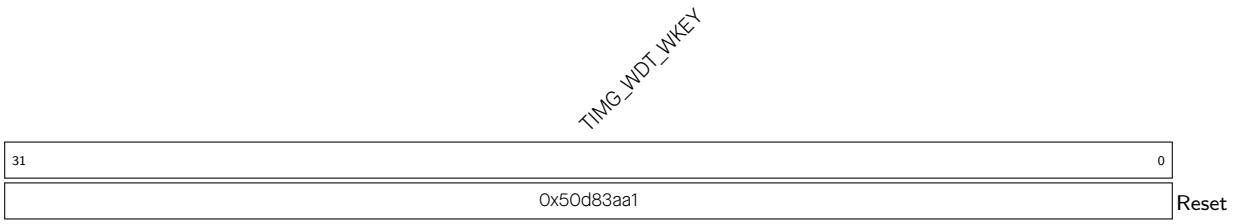
(R/W)

Register 14.16. TIMG_WDTFEED_REG (0x0060)



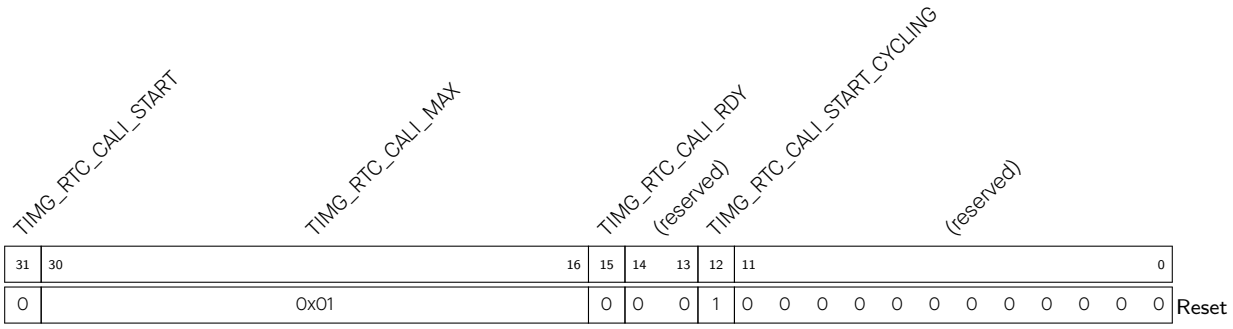

**TIMG_WDT_FEED** Write any value to feed the MWDT. Valid only when write protection is disabled. (WT)

Register 14.17. TIMG_WDTWPROTECT_REG (0x0064)




**TIMG_WDT_WKEY** Configures a different value than its reset value to enable write protection. (R/W)

**Register 14.18. 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)

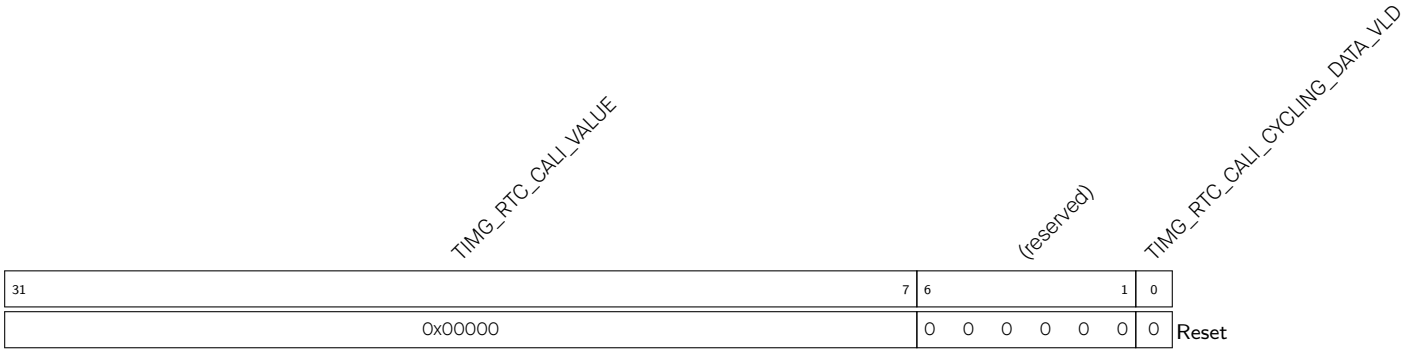
**TIMG_RTC_CALI_START** Configures whether to enable one-shot frequency calculation.

0: Disable

1: Enable

(R/W)

Register 14.19. TIMG_RTCCALICFG1_REG (0x006C)




**TIMG_RTC_CALI_CYCLING_DATA_VLD** Represents whether periodic frequency calculation is done.

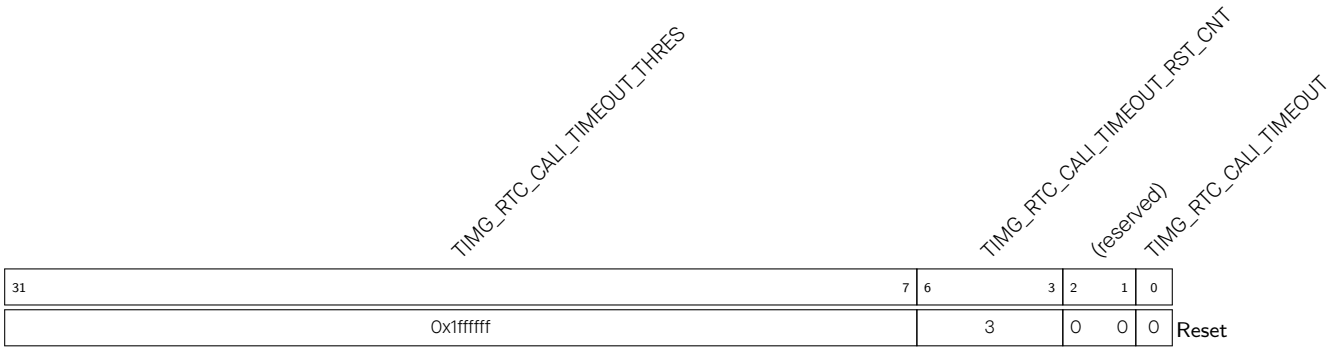
0: Not done

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 14.20. 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

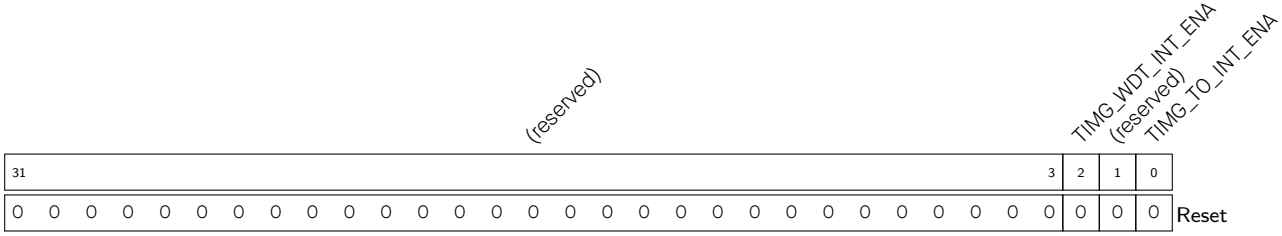
(R/W)

**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)

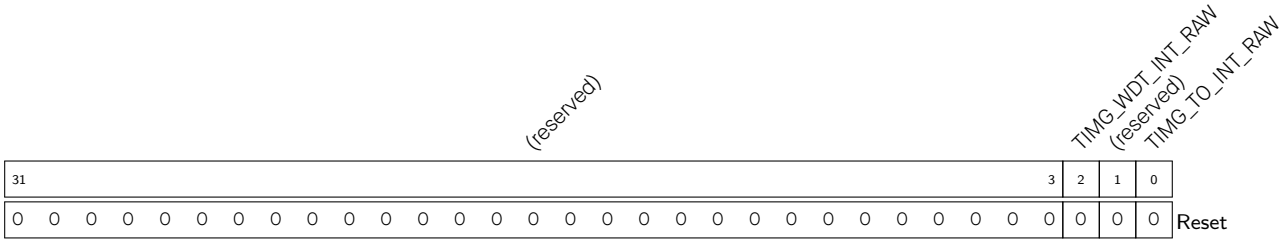
**Register 14.21. TIMG_INT_ENA_TIMERS_REG (0x0070)**




**TIMG_TO_INT_ENA** Write 1 to enable the TIMG_TO_INT interrupt. (R/W)

**TIMG_WDT_INT_ENA** Write 1 to enable the TIMG_WDT_INT interrupt. (R/W)

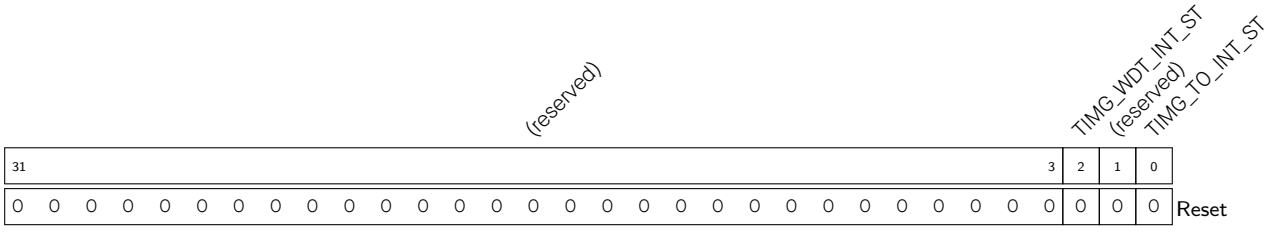
**Register 14.22. TIMG_INT_RAW_TIMERS_REG (0x0074)**




**TIMG_TO_INT_RAW** The raw interrupt status bit of the TIMG_TO_INT interrupt. (R/SS/WTC)

**TIMG_WDT_INT_RAW** The raw interrupt status bit of the TIMG_WDT_INT interrupt. (R/SS/WTC)

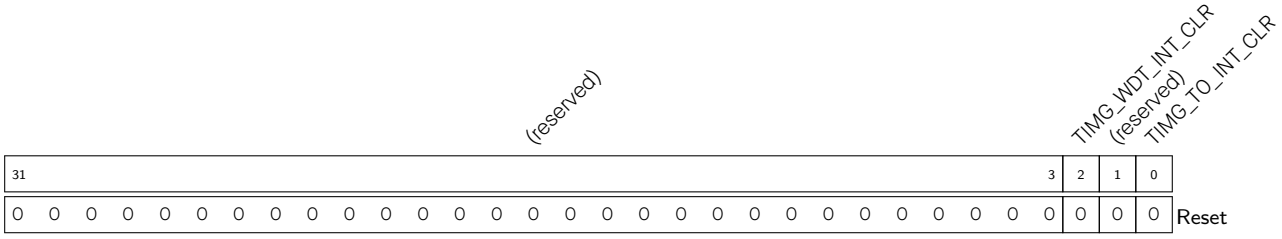
**Register 14.23. TIMG_INT_ST_TIMERS_REG (0x0078)**




**TIMG_TO_INT_ST** The masked interrupt status bit of the TIMG_TO_INT interrupt. (RO)

**TIMG_WDT_INT_ST** The masked interrupt status bit of the TIMG_WDT_INT interrupt. (RO)

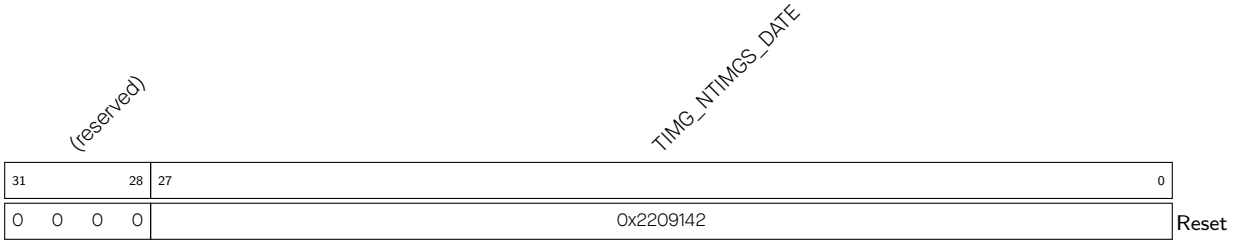
Register 14.24. TIMG_INT_CLR_TIMERS_REG (0x007C)




**TIMG_TO_INT_CLR** Write 1 to clear the TIMG_TO_INT interrupt. (WT)

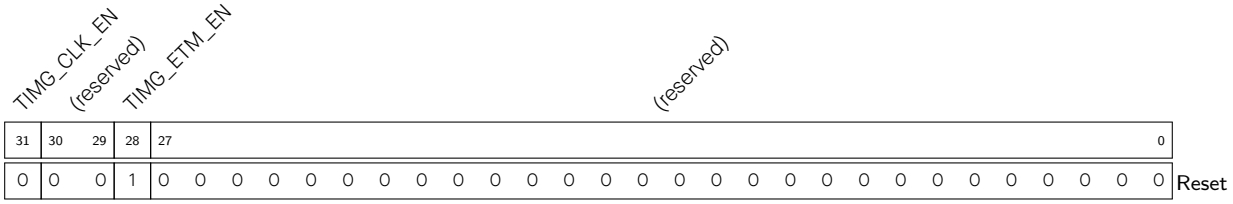
**TIMG_WDT_INT_CLR** Write 1 to clear the TIMG_WDT_INT interrupt. (WT)

Register 14.25. TIMG_NTIMERS_DATE_REG (0x00F8)




**TIMG_NTIMGS_DATE** Version control register (R/W)

Register 14.26. 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 15

# Watchdog Timers (WDT)

## 15.1 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 and thus trigger a watchdog timeout. Therefore, watchdog timers are useful for detecting and handling erroneous system/software behaviors.

As shown in Figure 15.1-1, ESP32-C61 contains three digital watchdog timers: one in each of the two timer groups in Chapter 14 *Timer Group (TIMG)* (called Main System Watchdog Timers, or MWDT) and one in the RTC Module (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, CPU reset, and core reset, while RWDT supports four timeout actions: interrupt, CPU reset, core reset, and system reset (see details in Section 15.2.2.2 *Stages and Timeout Actions*). A timeout value can be set for each stage individually.

During the flash boot process, RWDT and the MWDT in timer group 0 are enabled automatically in order to detect and recover from booting errors.

ESP32-C61 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.

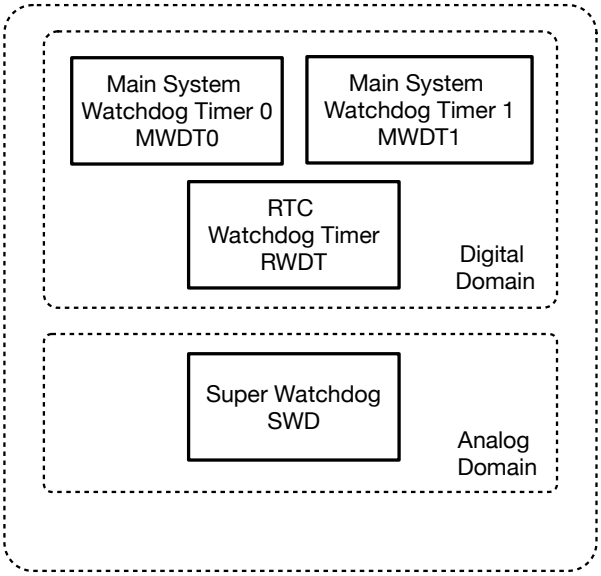



**Figure 15.1-1. Watchdog Timers Overview**

Note that while this chapter provides the functional descriptions of the watchdog timers, MWDT register descriptions are detailed in Chapter 14 *Timer Group (TIMG)*, and the RWDT and SWD register descriptions are detailed in Section 15.5 *Register Summary*.

## 15.2 Digital Watchdog Timers

### 15.2.1 Features

Watchdog timers have the following features:

- Four stages, each with a separately programmable timeout value and timeout action
- Timeout actions:
  - MWDT: interrupt, CPU reset, core reset
  - RWDT: interrupt, CPU reset, core reset, system reset
- Flash boot protection at stage 0:
  - MWDT0: 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: RTC_SLOW_CLK

## 15.2.2 Functional Description

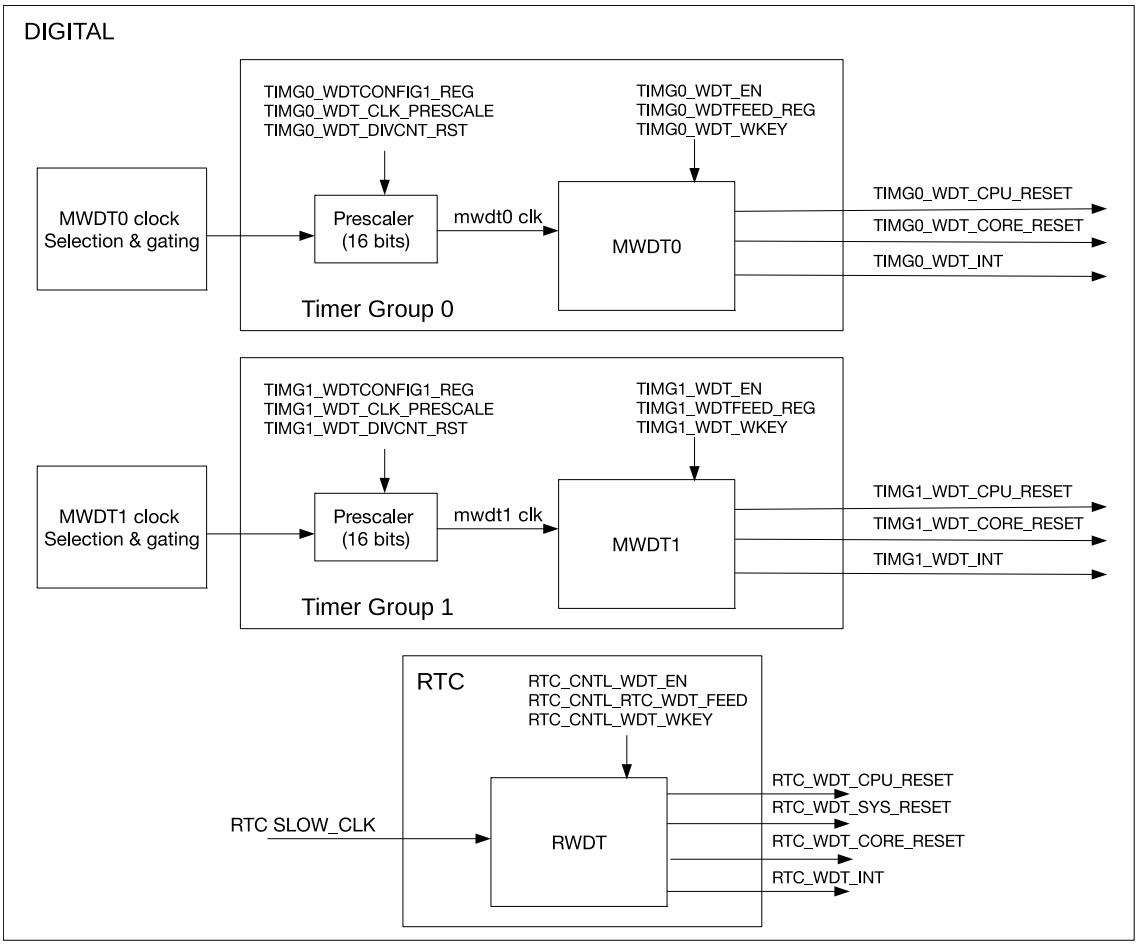



Figure 15.2-1. Digital Watchdog Timers in ESP32-C61

Figure 15.2-1 shows the three watchdog timers in ESP32-C61 digital systems.

### 15.2.2.1 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 PCR_TG0_WDT_CLK_SEL field of the PCR_TIMERGROUP0_WDT_CLK_CONF_REG register.
- The selected clock is switched on by setting PCR_TG0_WDT_CLK_EN field of the PCR_TIMERGROUP0_WDT_CLK_CONF_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 8.2-1 of Chapter 8 Reset and Clock.

The 16-bit prescaler value 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 RTC_SLOW_CLK (see details in Chapter 8 *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.

#### 15.2.2.2 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 provide four stages (called 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 the eFuse register EFUSE_RD_REPEAT_DATA0_REG and the
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 15.2-1. Timeout Actions**

| Timeout Action | Description |
|---|---|
| Interrupt | Trigger an interrupt |
| CPU reset | See Chapter 8 *Reset and Clock* > Section 8.1.3 *Features* |
| Core reset | |
| System reset | |
| 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.

#### 15.2.2.3 Write Protection

Watchdog timers are critical to detecting and handling erroneous system/software behavior, and thus should not be disabled easily (e.g. due to a misplaced register write). Therefore, MWDT and RWDT incorporate a write protection mechanism that prevents 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.

#### 15.2.2.4 Flash Boot Protection

During flash booting process, MWDT0 as well as RWDT, are automatically enabled. Stage 0 for the enabled MWDT0 is automatically configured as core reset action upon timeout, known as core reset. Likewise, stage 0 for RWDT is configured to system reset, which resets the main system and RTC when it times out. After 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.

## 15.3 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.

### 15.3.1 Features

SWD has the following features:

- An analog watchdog that operates independently of the digital circuit and can function in environments where the digital circuit's clock and voltage are abnormal
- 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

## 15.3.2 Super Watchdog Controller

### 15.3.2.1 Structure

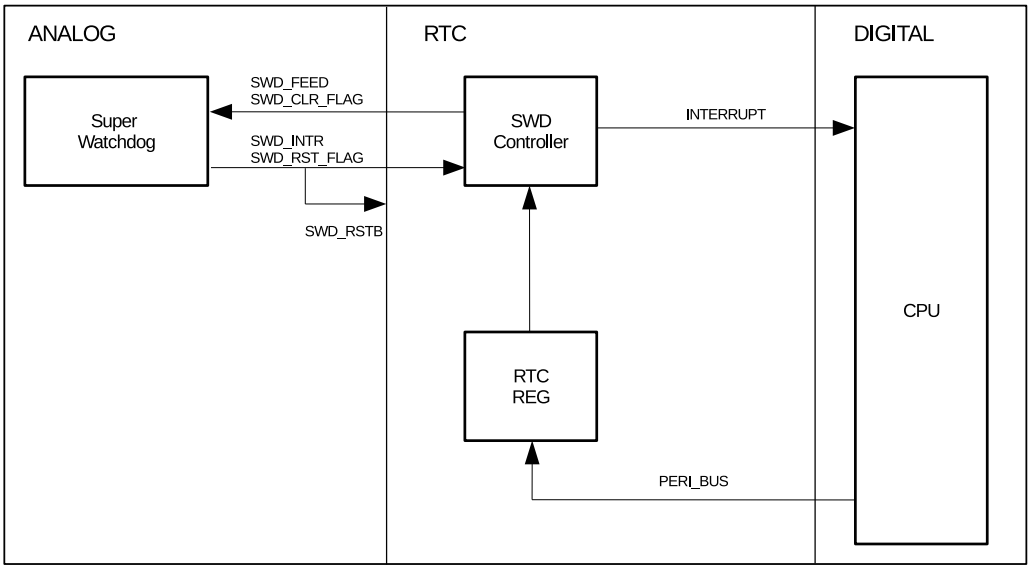



Figure 15.3-1. Super Watchdog Controller Structure

### 15.3.2.2 Workflow

In normal state:

- SWD controller receives feed request from SWD.
- SWD controller sends an interrupt to HP CPU.
- HP CPU feeds 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 receive interrupts from the SWD for feeding the watchdog itself without any interaction with CPU.

After reset:

- Check LP_CLKRST_RESET_CAUSE[4:0] for the cause of CPU reset.
  If LP_CLKRST_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.

## 15.4 Interrupts

For watchdog timer interrupts, please refer to Section 14.6 *Interrupts* in Chapter 14 *Timer Group (TIMG)*.

## 15.5 Register Summary

The addresses in this section are relative to RTC_WDT base address provided in Table 5.3-2 in Chapter 5 *System and Memory*.

The abbreviations given in Column **Access** are explained in Section *Access Types for Registers*.

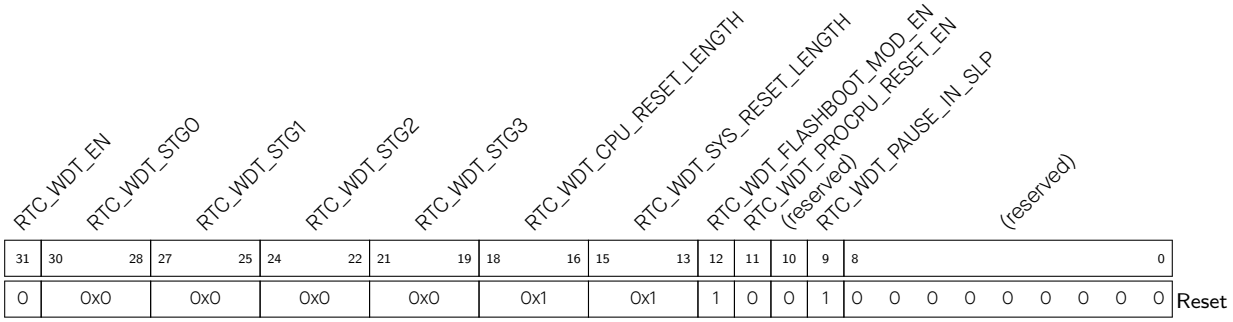
| Name | Description | Address | Access |
|---|---|---|---|
| **configuration register** | | | |
| RTC_WDT_CONFIG0_REG | Configure the RWDT operation | 0x0000 | R/W |
| RTC_WDT_CONFIG1_REG | Configure the RWDT timeout time of stage0 | 0x0004 | R/W |
| RTC_WDT_CONFIG2_REG | Configure the RWDT timeout 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 | The interrupt raw register of WDT | 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 |

## 15.6 Registers

MWDT registers are part of the timer submodule and are described in Section 14.8 *Register Summary* in Chapter 14 *Timer Group (TIMG)*.

The addresses of RWDT and SWD registers in this section are relative to RTC_WDT base address provided in Table 5.3-2 in Chapter 5 *System and Memory*.

Register 15.1. RTC_WDT_CONFIG0_REG (0x0000)




**RTC_WDT_PAUSE_IN_SLP** Configures whether or not to pause RWDT when chip is in Light-sleep or Deep-sleep mode.

0: Enable

1: Disable

(R/W)

**RTC_WDT_PROCPU_RESET_EN** Configures whether or not to enable RWDT to reset CPU.

0: Disable

1: Enable

(R/W)

**RTC_WDT_FLASHBOOT_MOD_EN** Configures whether or not to enable RWDT when chip is in SPI boot mode.

0: Disable

1: Enable

(R/W)

**RTC_WDT_SYS_RESET_LENGTH** Configures the core reset time.

Measurement unit: LP_DYN_FAST_CLK cycles

(R/W)

**RTC_WDT_CPU_RESET_LENGTH** Configures the CPU reset time.

Measurement unit: LP_DYN_FAST_CLK cycles

(R/W)

**RTC_WDT_STG3** Configures the timeout action of stage3.

0: No operation

1: Generate interrupt

2: Generate CPU reset

3: Generate core reset

4: Generate system reset

(R/W)

Continued on the next page...

Register 15.1. RTC_WDT_CONFIG0_REG (0x0000)

Continued from the previous page...

**RTC_WDT_STG2** Configures the timeout action of stage2.

0: No operation

1: Generate interrupt

2: Generate CPU reset

3: Generate core reset

4: Generate system reset

(R/W)

**RTC_WDT_STG1** Configures the timeout action of stage1.

0: No operation

1: Generate interrupt

2: Generate CPU reset

3: Generate core reset

4: Generate system reset

(R/W)

**RTC_WDT_STG0** Configures the timeout action of stage0.

0: No operation

1: Generate interrupt

2: Generate CPU reset

3: Generate core reset

4: Generate system reset

(R/W)

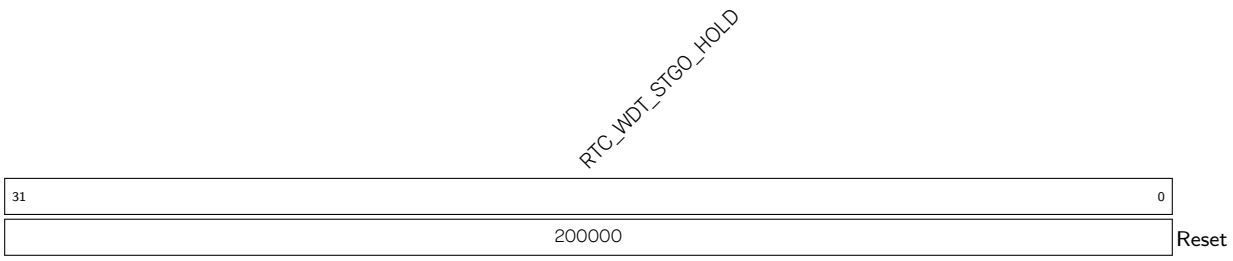
**RTC_WDT_EN** Configures whether or not enable RWDT.

0: Disable RWDT

1: Enable RWDT

(R/W)

Register 15.2. RTC_WDT_CONFIG1_REG (0x0004)

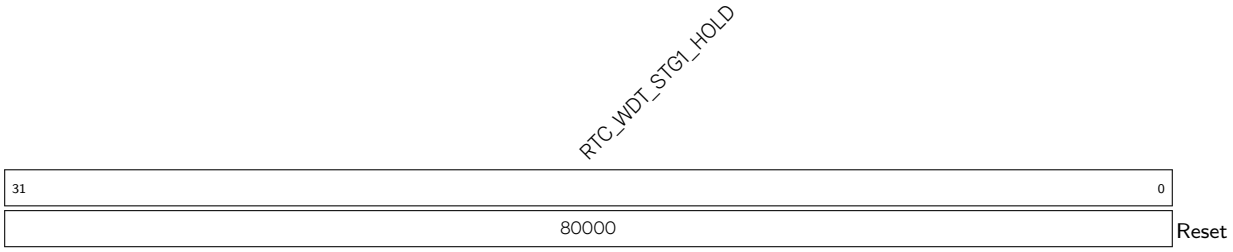



**RTC_WDT_STG0_HOLD** Configures the timeout time for stage0.

Measurement unit: LP_DYN_SLOW_CLK cycles

(R/W)

Register 15.3. RTC_WDT_CONFIG2_REG (0x0008)

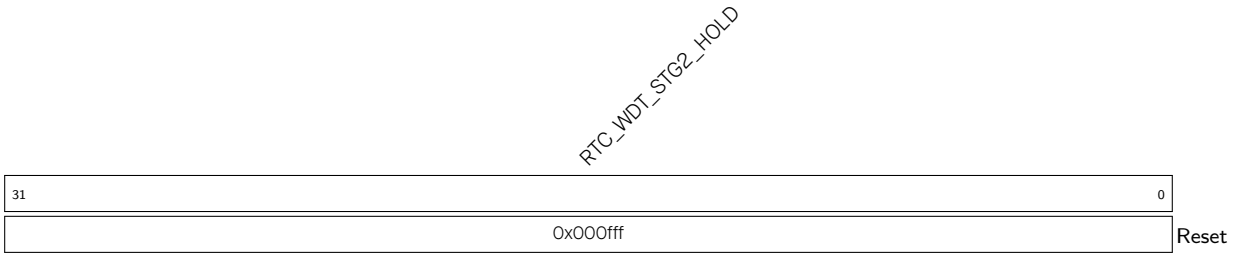



**RTC_WDT_STG1_HOLD** Configures the timeout time for stage1.

Measurement unit: LP_DYN_SLOW_CLK cycles

(R/W)

Register 15.4. RTC_WDT_CONFIG3_REG (0x000C)

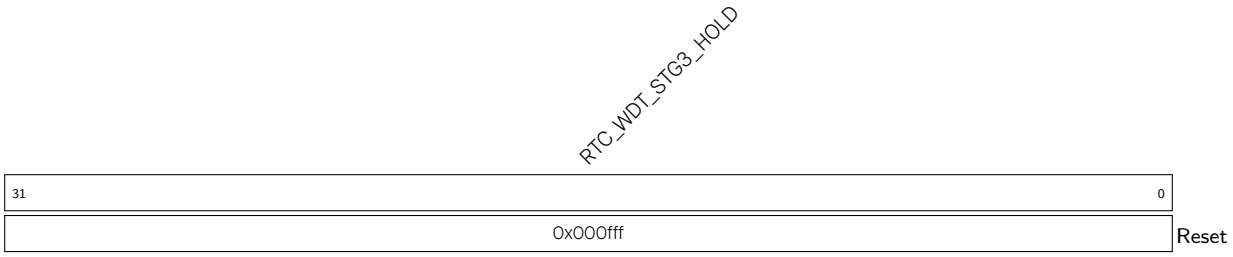



**RTC_WDT_STG2_HOLD** Configures the timeout time for stage2.

Measurement unit: LP_DYN_SLOW_CLK cycles

(R/W)

Register 15.5. RTC_WDT_CONFIG4_REG (0x0010)

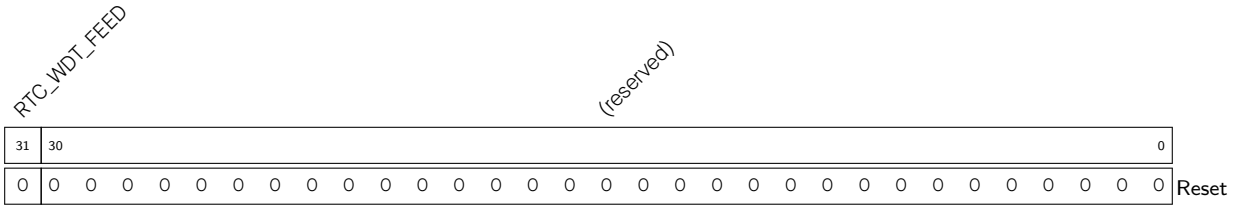



**RTC_WDT_STG3_HOLD** Configures the timeout time for stage3.

Measurement unit: LP_DYN_SLOW_CLK cycles

(R/W)

Register 15.6. RTC_WDT_FEED_REG (0x0014)



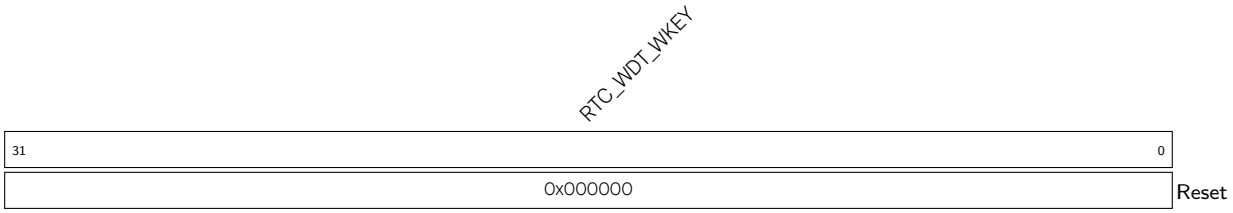

**RTC_WDT_FEED** Configures this bit to feed the RWDT.

0: Invalid

1: Feed RWDT

(WT)

Register 15.7. RTC_WDT_WPROTECT_REG (0x0018)



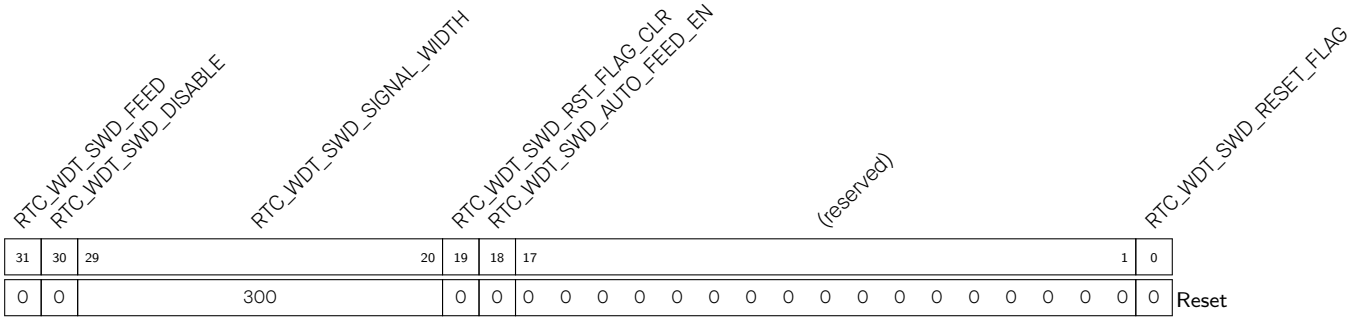

**RTC_WDT_WKEY** Configures this field to lock or unlock RWDT's configuration registers.

0x50D83AA1: unlock the RWDT configuration register

Other values: lock the RWDT configuration register which can't be modified by software.

(R/W)

**Register 15.8. RTC_WDT_SWD_CONFIG_REG (0x001C)**




**RTC_WDT_SWD_RESET_FLAG** Represents whether the SWD generates a reset signal or not.

0: No

1: Yes

(RO)

**RTC_WDT_SWD_AUTO_FEED_EN** Configures this bit to enable to feed SWD automatically by hardware.

0: Disable

1: Enable

(R/W)

**RTC_WDT_SWD_RST_FLAG_CLR** Configures 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** Configures this bit to disable the SWD.

0: Enable the SWD

1: Disable the SWD

(R/W)

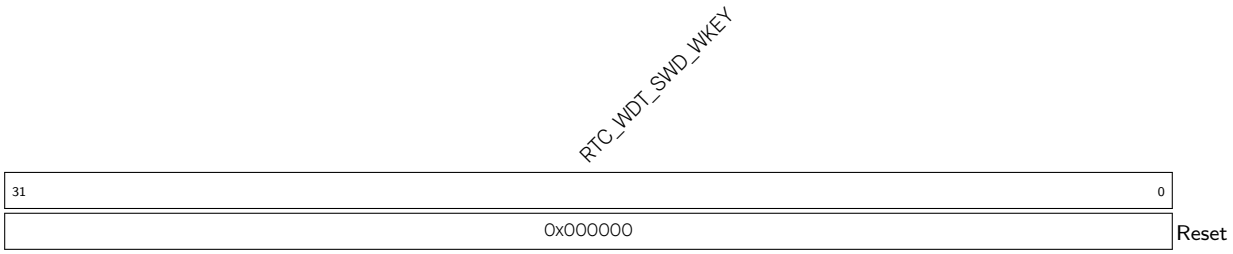
**RTC_WDT_SWD_FEED** Configures this bit to feed the SWD.

0: Invalid

1: Feed SWD

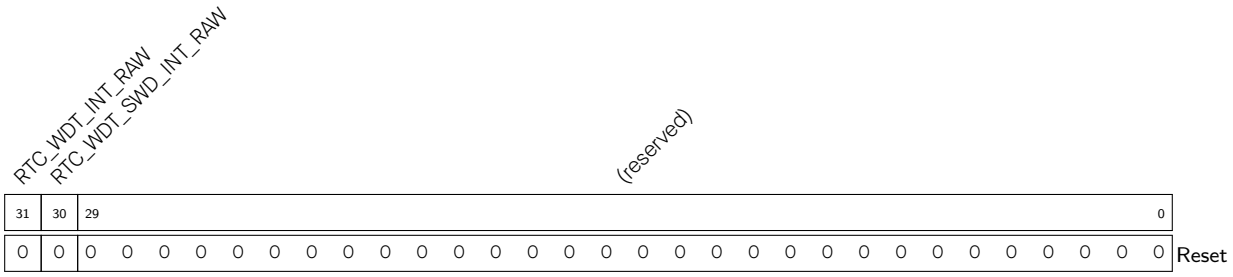
(WT)

Register 15.9. RTC_WDT_SWD_WPROTECT_REG (0x0020)




**RTC_WDT_SWD_WKEY** Configures this field to lock or unlock SWD's configuration registers.
0x50D83AA1: unlock the SWD configuration register.
Other values: lock the SWD configuration register which can't be modified by the software.
(R/W)

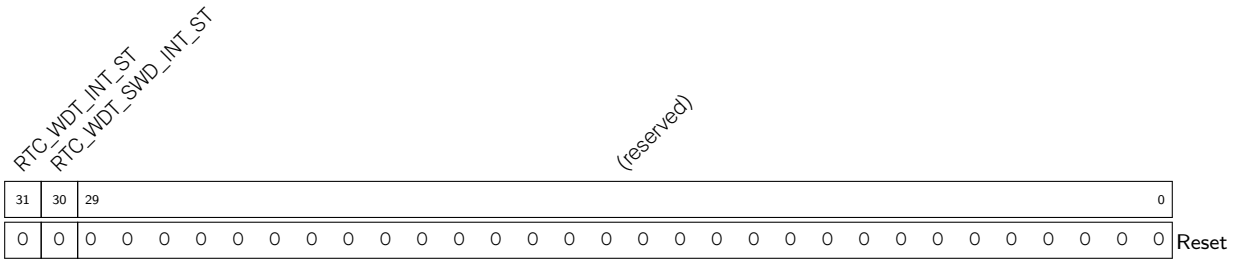
Register 15.10. RTC_WDT_INT_RAW_REG (0x0024)




**RTC_WDT_SWD_INT_RAW** The raw interrupt status of RTC_WDT_SWD_INT interrupt.(R/WTC/SS)

**RTC_WDT_INT_RAW** The raw interrupt status of RTC_WDT_INT interrupt. (R/WTC/SS)

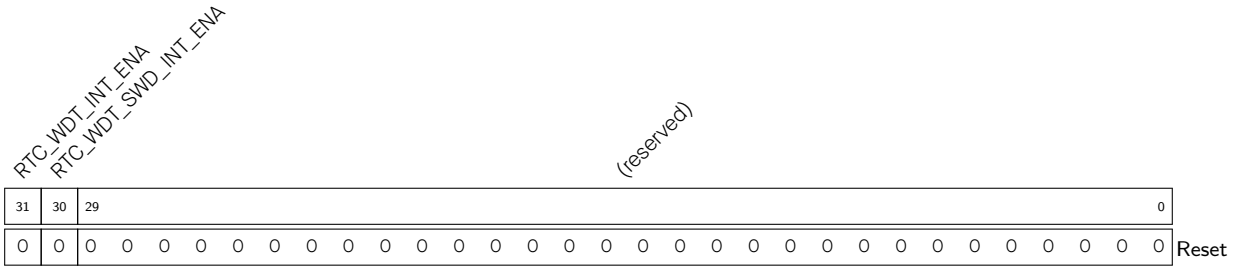
Register 15.11. RTC_WDT_INT_ST_REG (0x0028)




**RTC_WDT_SWD_INT_ST** The masked interrupt status of RTC_WDT_SWD_INT interrupt.(RO)

**RTC_WDT_INT_ST** The masked interrupt status of RTC_WDT_INT interrupt.(RO)

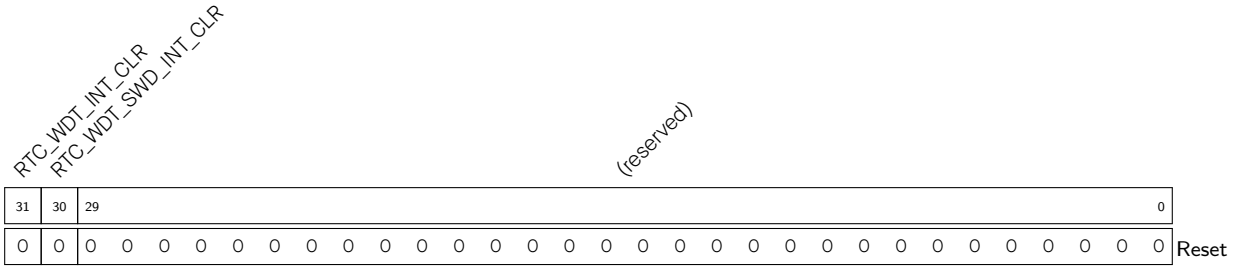
Register 15.12. RTC_WDT_INT_ENA_REG (0x002C)




**RTC_WDT_SWD_INT_ENA** Write 1 to enable RTC_WDT_SWD_INT. (R/W)

**RTC_WDT_INT_ENA** Write 1 to enable RTC_WDT_INT. (R/W)

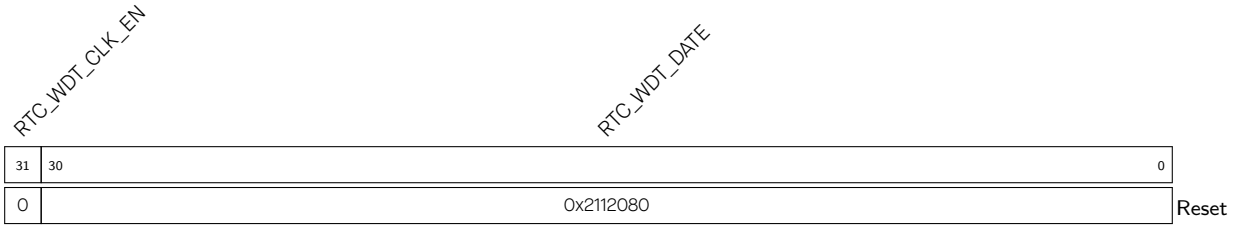
Register 15.13. RTC_WDT_INT_CLR_REG (0x0030)




**RTC_WDT_SWD_INT_CLR** Write 1 to clear RTC_WDT_SWD_INT. (WT)

**RTC_WDT_INT_CLR** Write 1 to clear RTC_WDT_INT. (WT)

Register 15.14. RTC_WDT_DATE_REG (0x03FC)




**RTC_WDT_DATE** Version control register. (R/W)

**RTC_WDT_CLK_EN** Reserved. (R/W)

# Chapter 16

# RTC Timer

## 16.1 Introduction

The RTC Timer is a 48-bit readable counter that can operate in any power mode. It is used as a system timer when the timers in the HP system is unavailable. It also allows for configuring timer interrupts and logging the time when specific events happen in the system.

## 16.2 Feature List

- 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 the target time, which is configurable
- Uninterrupted operation during any reset or sleep mode, except for power-on reset of LP system
- Can work as the wake-up source (see Chapter 12 *Low-Power Management*)

## 16.3 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 Always-On power domain (for details about power domains, see Chapter 12 *Low-Power Management*). 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
  - The 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 16.3-1. 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 log 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 log the current time.
- Generate counting interrupts at target time
  - The RTC Timer supports configuring the target time, referred to as target time 0. When the timer reaches target time 0, it will trigger the RTC_TIMER_CMP0_INT interrupt.
  - Configure the target time through the registers below:

**Table 16.3-2. Target Time Configuration**

| Configuration Register | Description |
|---|---|
| RTC_TIMER_MAIN_TIMER_TAR_EN0 | Enable target time configuration. |
| RTC_TIMER_MAIN_TIMER_TAR_LOW0 | Configure the lower 32 bits of the target time. |
| RTC_TIMER_MAIN_TIMER_TAR_HIGH0 | Configure the higher 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.

## 16.4 Event Task Matrix Feature

The RTC Timer on ESP32-C61 supports the Event Task Matrix (ETM) function, which allows RTC Timer's ETM events to trigger any peripherals' ETM tasks. The ETM tasks of RTC timer are not supported. This section introduces the ETM events related to RTC Timer. For more information, please refer to Chapter 11 Event Task Matrix (ETM).

The RTC Timer can generate the following ETM events:

- RTC_EVT_CMP: Indicates 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: Indicates that the 48-bit counter overflows.
- RTC_EVT_TICK: Indicates the event that occurs when the RTC timer increments by 1.

## 16.5 Interrupts

ESP32-C61's RTC Timer can generate the following interrupt signal(s) that will be sent to the Interrupt Matrix.

- LP_TIMER_REG_0_INTR (only for HP CPU)

There are several internal interrupt sources from the RTC Timer that can generate the above interrupt signal(s). The interrupt sources from the RTC Timer are listed with their trigger conditions and the resulted interrupt signal(s) in Table 16.5-1.

**Table 16.5-1. 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_OVERFLOW_INT | 48-bit counter overflows | LP_TIMER_REG_0_INTR |

**Note:**
For definitions of *interrupt*, *interrupt signal*, *interrupt source*, and their correlations, please refer to Chapter 10 *Interrupt Matrix* > Section 10.2 *Interrupt Terminology in ESP32-C61*.

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 16.6 *Register Summary*.

## 16.6 Register Summary

The addresses in this section are relative to RTC Timer base address provided in Table 5.3-2 in Chapter 5 *System and Memory*.

The abbreviations given in Column **Access** are explained in Section *Access Types for Registers*.

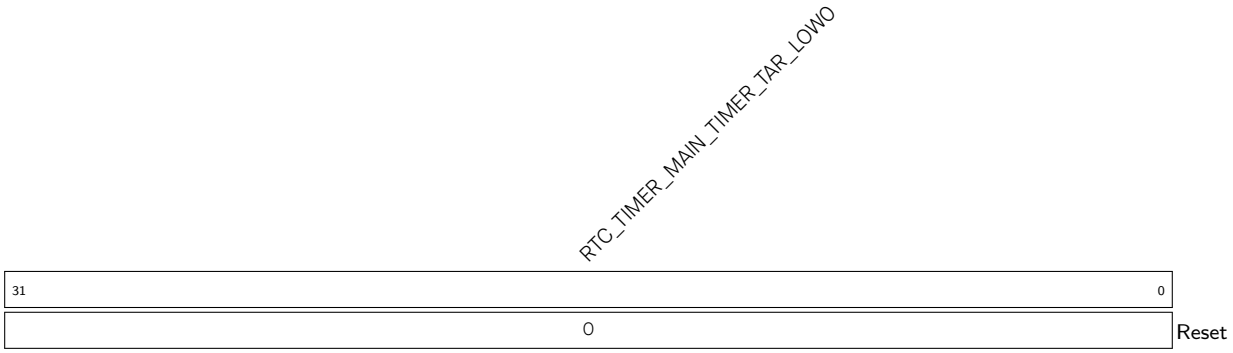
| Name | Description | Address | Access |
|---|---|---|---|
| **Configuration Registers** | | | |
| RTC_TIMER_TARO_LOW_REG | Configures the low 32 bits of the target time 0 | 0x0000 | R/W |
| RTC_TIMER_TARO_HIGH_REG | Configures the high 16 bits of the target time 0 | 0x0004 | varies |
| RTC_TIMER_UPDATE_REG | Configures 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 | R/WTC/SS |
| 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 | WT |
| **Version Register** | | | |
| RTC_TIMER_DATE_REG | Version control register | 0x03FC | R/W |

## 16.7 Registers

The addresses in this section are relative to RTC Timer base address provided in Table 5.3-2 in Chapter 5 *System and Memory*.

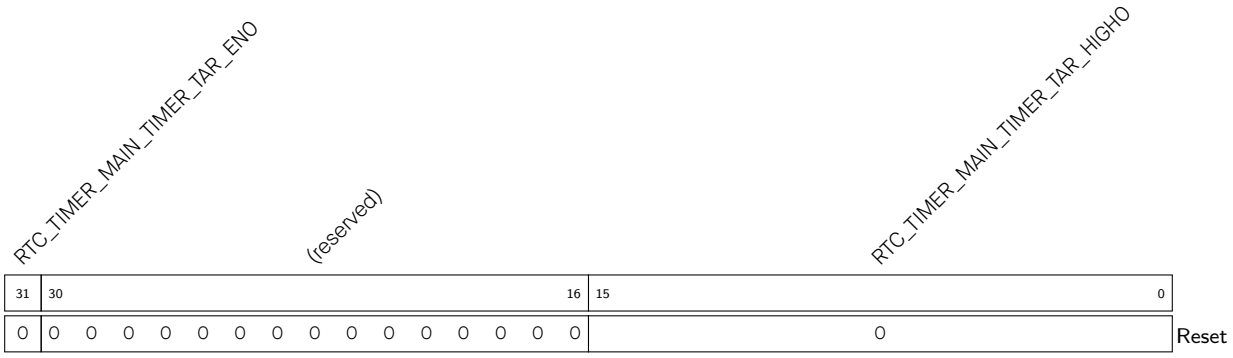
For how to program reserved fields, please refer to Section Programming Reserved Register Field.

**Register 16.1. RTC_TIMER_TAR0_LOW_REG (0x0000)**




**RTC_TIMER_MAIN_TIMER_TAR_LOW0** Configures the low 32 bits of the target time 0 of the RTC Timer. (R/W)

**Register 16.2. RTC_TIMER_TAR0_HIGH_REG (0x0004)**




**RTC_TIMER_MAIN_TIMER_TAR_HIGH0** Configures the high 16 bits of the target time 0. (R/W)

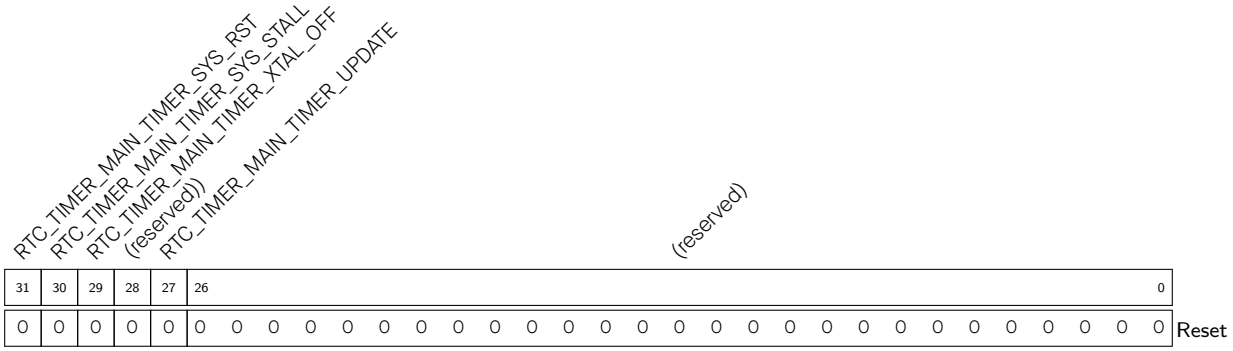
**RTC_TIMER_MAIN_TIMER_TAR_EN0** Configures to enable the target time 0 of the RTC Timer.

0: Disable

1: Enable

(WT)

Register 16.3. RTC_TIMER_UPDATE_REG (0x0010)




**RTC_TIMER_MAIN_TIMER_UPDATE** Configures whether to log the current time through software configuration.

0: Not log

1: Log

(WT)

**RTC_TIMER_MAIN_TIMER_XTAL_OFF** Configures whether to enable RTC Timer to log the time when the crystal powers up or down.

0: Disable

1: Enable

(R/W)

**RTC_TIMER_MAIN_TIMER_SYS_STALL** Configures whether to enable the RTC Timer to log the time when the CPU enters or exits the stall state.

0: Disable

1: Enable

(R/W)

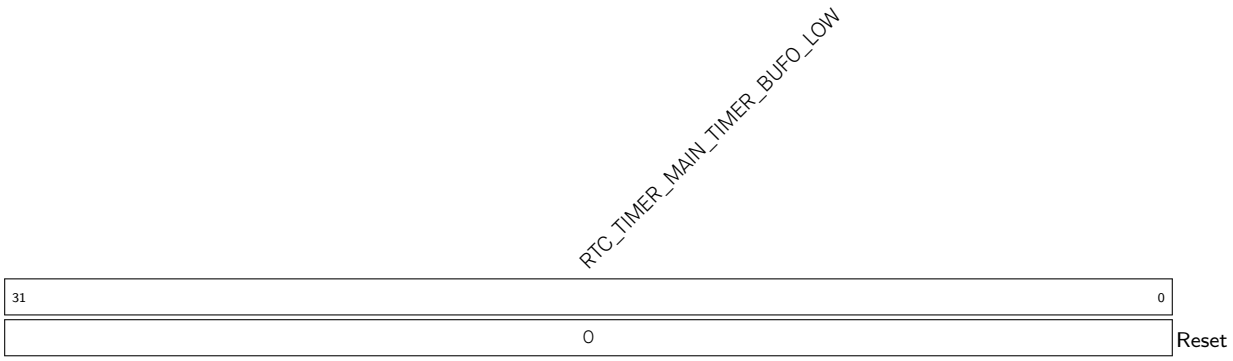
**RTC_TIMER_MAIN_TIMER_SYS_RST** Configures whether to enable RTC Timer to log the time of system reset.

0: Disable

1: Enable

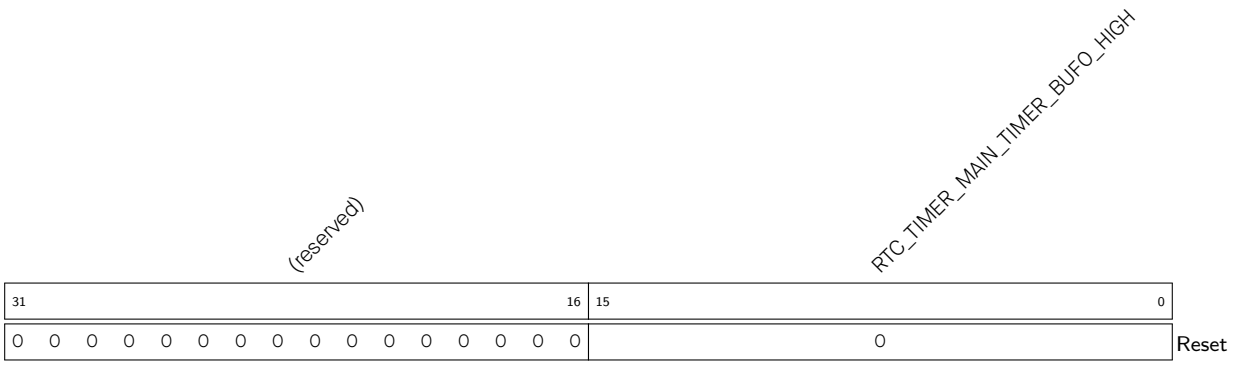
(R/W)

Register 16.4. RTC_TIMER_MAIN_BUF0_LOW_REG (0x0014)



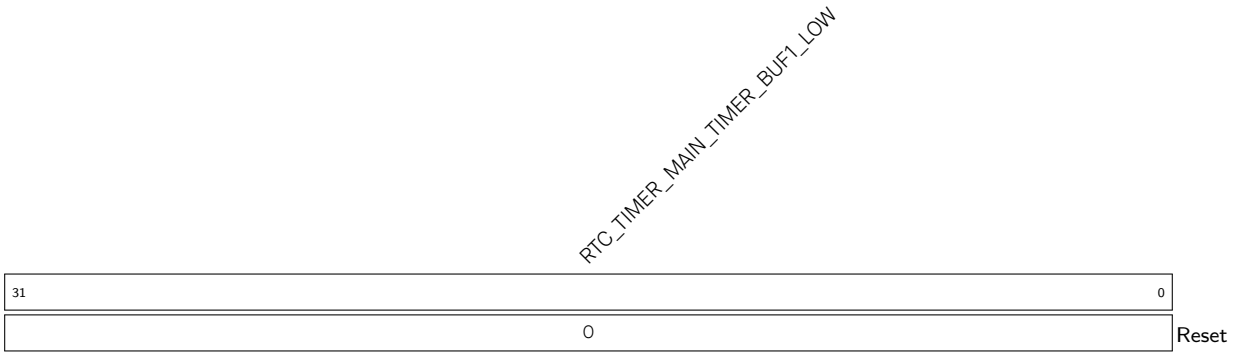

**RTC_TIMER_MAIN_TIMER_BUF0_LOW** Represents the low 32 bits of the cached value 0 in RTC Timer. (RO)

Register 16.5. RTC_TIMER_MAIN_BUF0_HIGH_REG (0x0018)



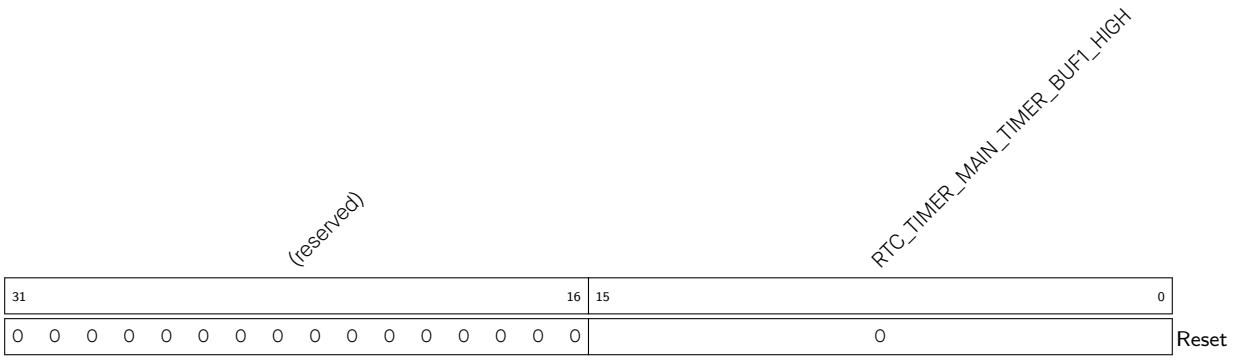

**RTC_TIMER_MAIN_TIMER_BUF0_HIGH** Represents the high 16 bits of the cached value 0 in RTC Timer. (RO)

Register 16.6. RTC_TIMER_MAIN_BUF1_LOW_REG (0x001C)



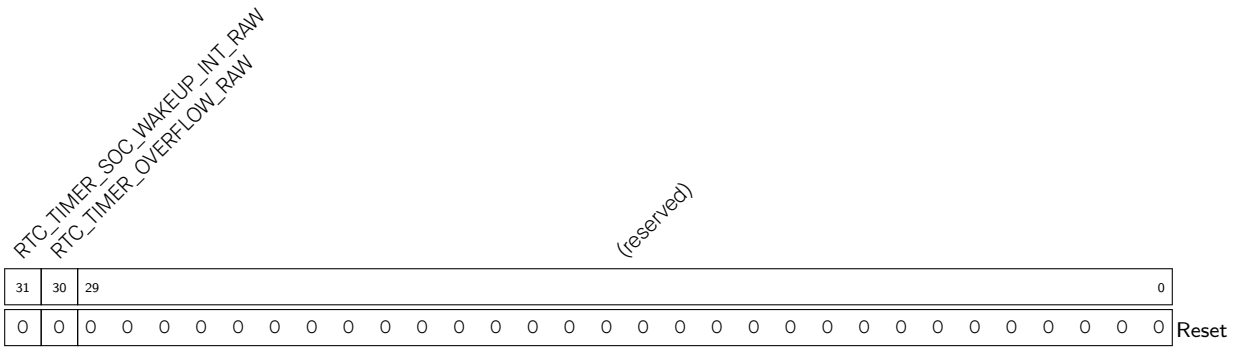

**RTC_TIMER_MAIN_TIMER_BUF1_LOW** Represents the low 32 bits of the cached value 1 in RTC Timer. (RO)

Register 16.7. RTC_TIMER_MAIN_BUF1_HIGH_REG (0x0020)




**RTC_TIMER_MAIN_TIMER_BUF1_HIGH** Represents the high 16 bits of the cached value 1 in RTC Timer. (RO)

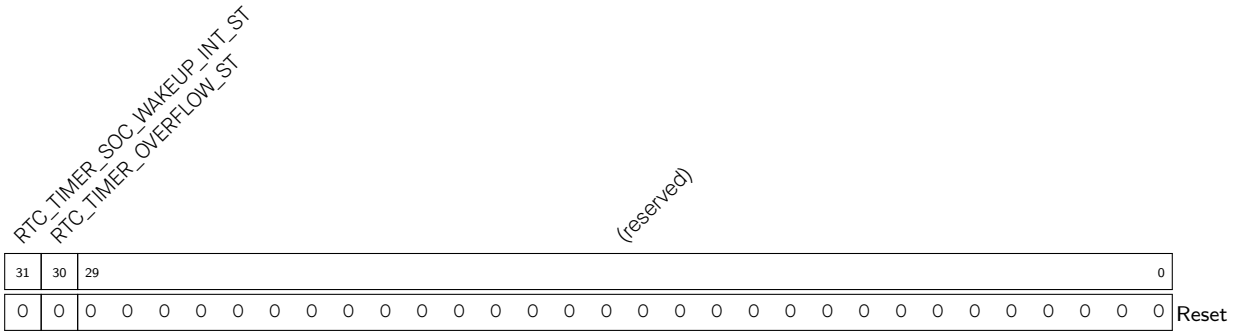
Register 16.8. RTC_TIMER_INT_RAW_REG (0x0028)




**RTC_TIMER_OVERFLOW_RAW** The raw interrupt status of RTC_TIMER_OVERFLOW_INT (HP CPU). (R/WTC/SS)

**RTC_TIMER_SOC_WAKEUP_INT_RAW** The raw interrupt status of RTC_TIMER_CMP0_INT. (R/WTC/SS)

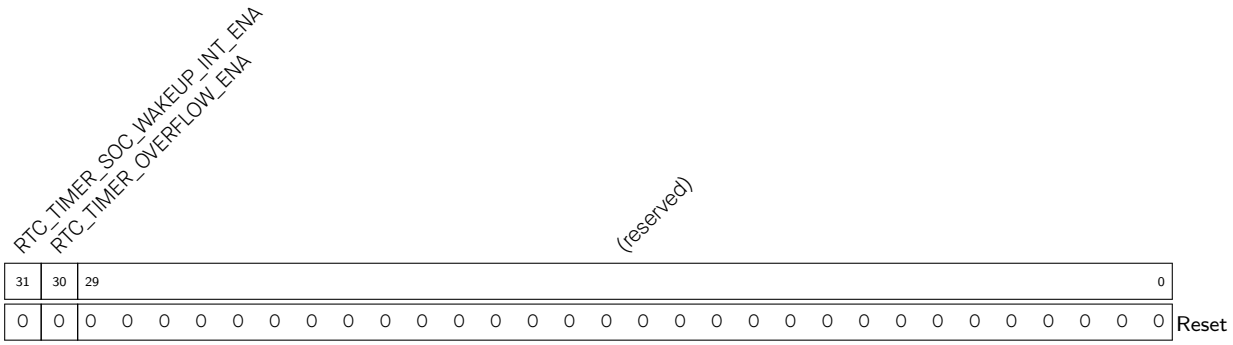
Register 16.9. RTC_TIMER_INT_ST_REG (0x002C)




**RTC_TIMER_OVERFLOW_ST** The masked interrupt status of RTC_TIMER_OVERFLOW_INT (HP CPU). (RO)

**RTC_TIMER_SOC_WAKEUP_INT_ST** The masked interrupt status of RTC_TIMER_CMP0_INT. (RO)

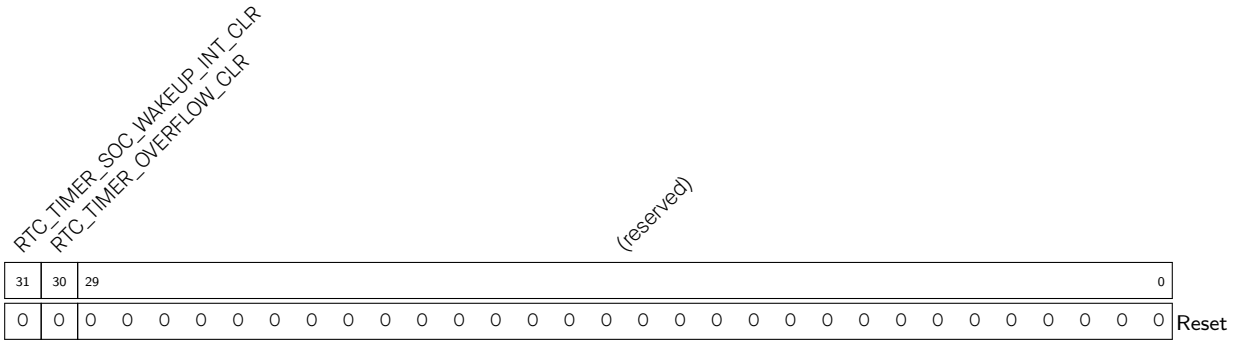
Register 16.10. RTC_TIMER_INT_ENA_REG (0x0030)




**RTC_TIMER_OVERFLOW_ENA** Write 1 to enable RTC_TIMER_OVERFLOW_INT (HP CPU). (R/W)

**RTC_TIMER_SOC_WAKEUP_INT_ENA** Write 1 to enable RTC_TIMER_CMP0_INT. (R/W)

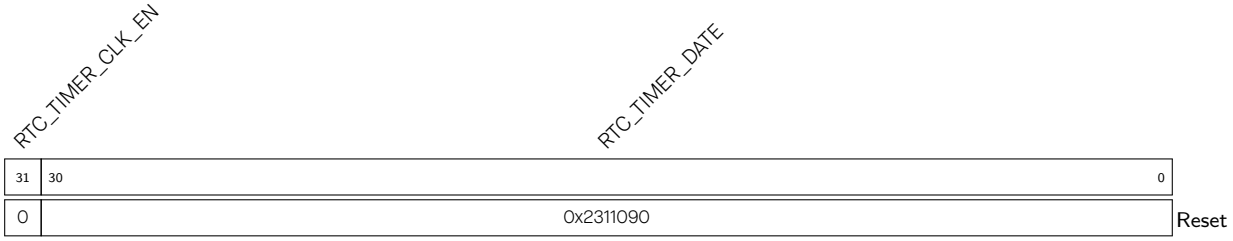
Register 16.11. RTC_TIMER_INT_CLR_REG (0x0034)




**RTC_TIMER_OVERFLOW_CLR** Write 1 to clear RTC_TIMER_OVERFLOW_INT (HP CPU). (WT)

**RTC_TIMER_SOC_WAKEUP_INT_CLR** Write 1 to clear RTC_TIMER_CMP0_INT. (WT)

Register 16.12. RTC_TIMER_DATE_REG (0x03FC)




**RTC_TIMER_DATE** Version control register. (R/W)

**RTC_TIMER_CLK_EN** Configures RTC timer clock gating.

0: Support clock only when the application writes registers.

1: Always force the clock on for registers.

(R/W)

# Chapter 17

# Permission Control (PMS)

## 17.1 Overview

The ESP32-C61 employs a two-fold permission control strategy: Physical Memory Protection (PMP) and Access Permission Management (APM).

The PMP is located inside the HP CPU and controls the HP CPU's access to all address spaces. APM is an access permission management module located at the bus port. APM, together with PMP, manages permissions for the HP CPU to access parts of the memory and the peripheral registers. Additionally, APM manages DMA masters such as GDMA and MEM_MONITOR to access parts of internal and external memory.

The distribution of management areas by PMP and APM is shown in Table 17.1-1.

**Table 17.1-1. Management Areas by PMP and APM**

| Slaves / Masters | ROM | HP SRAM | HP CPU_PERI[1] | HP_PERI[2] | LP_PERI[3] | EXT_MEM[4] |
|---|---|---|---|---|---|---|
| **HP CPU** | PMP | PMP + CPU_APM | PMP + HP_APM | PMP + HP_APM | PMP + LP_APM | PMP |
| **GDMA** | N/A | HP_APM | N/A | N/A | N/A | HP_APM |
| **Other masters[5]** | N/A | HP_APM | N/A | N/A | N/A | N/A |

[1] Peripheral registers in the HP CPU, address range: 0x600C_0000 – 0x600C_FFFF
[2] Peripheral registers in the high-performance system, address range: 0x6000_0000 – 0x600A_FFFF
[3] Peripheral registers in the low-power system, address range: 0x600B_0000 – 0x600B_FFFF
[4] External memory, e.g., flash, PSRAM.
[5] Masters that request access to the bus, such as TCM_MEM_MONITOR, PSRAM_MEM_MONITOR. For a complete list of masters, please refer to Table 17.5-1.
[6] For more information on HP_APM, LP_APM, and CPU_APM, please refer to Section 17.5.2.

As shown in Table 17.1-1, when the HP CPU accesses ROM and EXT MEM, the access paths are solely managed by PMP. On the other hand, when the HP CPU accesses HP SRAM, HP CPU_PERI, HP_PERI, and LP_PERI, 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 CPU and can be read or configured with special instructions. For how to configure PMP, please refer to Chapter *2 ESP-RISC-V CPU* > *Standard Physical Memory Protection*.

The following sections will provide a detailed introduction to the APM module, covering its features, functions, and configurations.

## 17.2 Introduction to APM

The APM module contains two parts: the TEE (Trusted Execution Environment) controller and the APM controller. Each module contains its own register: the TEE register and APM register.

- The TEE controller is responsible for configuring the security mode of a particular master in ESP32-C61 (such as GDMA) to access memory or peripheral registers. There are four security modes: TEE, REE0 (Rich Execution Environment), REE1, and REE2.
- The APM controller manages a master's access permissions when accessing memory and peripheral registers. By comparing the pre-configured address ranges and corresponding access permissions with the information carried on the bus, such as ID number (please refer to Table 17.5-1), security mode, access address, access permissions, etc, the APM controller determines whether the access is allowed.

TEE-related registers configure the security mode of each master. The APM-related registers specify the access permission and access address range of each master in the configured security mode. With the TEE controller and APM controller, ESP32-C61 can precisely control the access permission of all masters to the memory and peripheral registers.

## 17.3 Features

ESP32-C61 has one TEE controller: HP_TEE.

- HP_TEE has the following features:
  - Four security modes for the masters
  - Configurable security mode for 32 masters

ESP32-C61 has three APM controllers: HP_APM_CTRL, LP_APM_CTRL, and CPU_APM_CTRL.

- HP_APM_CTRL has the following features:
  - Access permission configuration for up to 16 address ranges
  - Access management to HP SRAM, HP CPU_PERI, HP_PERI, and EXT MEM
  - Interrupt function on illegal access
  - Exception information record
- LP_APM_CTRL has the following features:
  - Access permission configuration for up to four address ranges
  - Access permission management to LP_PERI
  - Interrupt function on illegal access
  - Exception information record
- CPU_APM_CTRL has the following features:
  - Access permission configuration for up to eight address ranges
  - Access permission management for HP CPU to access HP SRAM
  - Interrupt function on illegal access

- Exception information record

## 17.4 TEE and REE Terminology

**TEE** Stands for Trusted Execution Environment, which is a secure area that is isolated from the main operating system and provides a secure environment for executing sensitive operations.

**REE** Stands for Rich Execution Environment, which is the main operating system and environment in which most applications run.

**Table 17.4-2. Comparison Between TEE and REE**

| Aspect | TEE | REE |
|---|---|---|
| Security | Enhanced security | Normal security |
| Access permission | Masters in TEE mode always have read, write, and execute permissions in the address range. | Different levels of access permissions of REE0/1/2 are configurable by software. |

## 17.5 Functional Description

### 17.5.1 TEE Controller Functional Description

ESP32-C61 supports four security modes: TEE, REE0, REE1, and REE2.

For the HP CPU to access memory or peripheral registers, first select the machine mode or user mode for it, then configure its security mode with HP_TEE registers. For the configuration of machine mode and user mode, please refer to RISC-V Instruction Set Manual, Volume II: Privileged Architecture, Version 1.10.

- When the HP CPU is in machine mode, its security mode is TEE mode.
- When the HP CPU is in user mode, its security mode is REE mode. The specific REE0, REE1 or REE2 mode depends on TEE_M0_MODE as follows:
  - If TEE_M0_MODE is 0, which is TEE mode, the valid mode that actually takes effect in the HP CPU user mode is REE0.
  - if TEE_M0_MODE is set to 1, 2, or 3, the security mode is REE0, REE1, and REE2 respectively.

For masters other than the HP CPU to access memory or peripheral registers, configure their security mode with TEE_M*n*_MODE. *n* represents the ID number of master as listed in Table 17.5-1.

**Table 17.5-1. Master Access Source from HP System**

| Value of *n* | Source |
|---|---|
| 0 | HP CPU |
| 1 | Reserved |
| 2 | Reserved |
| 3 | Reserved |

| Value | Source |
|---|---|
| 4 | Reserved |
| 5 | TCM_MEM_MONITOR |
| 6 | TRACE |
| 7 | Reserved |
| 8 | PSRAM_MEM_MONITOR |
| 9 ~ 15 | Reserved |
| 16 ~ 31 | See the peripherals corresponding to the values 0 ~ 15 in Chapter 4 *GDMA Controller (GDMA)* > Table 4.4-1 *GDMA Selecting Peripherals via Register Configuration*. For example, 16 corresponds to the peripheral with value 0 in Table 4.4-1, and 17 corresponds to the peripheral with value 1 in the table, and so on. |

Configure TEE_M*n*_LOCK to lock the configuration of the master's secure mode. Only chip reset, system reset, and core reset can unlock the configurations and restore all configurations in TEE registers to their default values.

## 17.5.2 APM Controller Functional Description

### 17.5.2.1 Architecture

Figure 17.5-1 shows the architecture of the APM controller and the access paths managed by it.

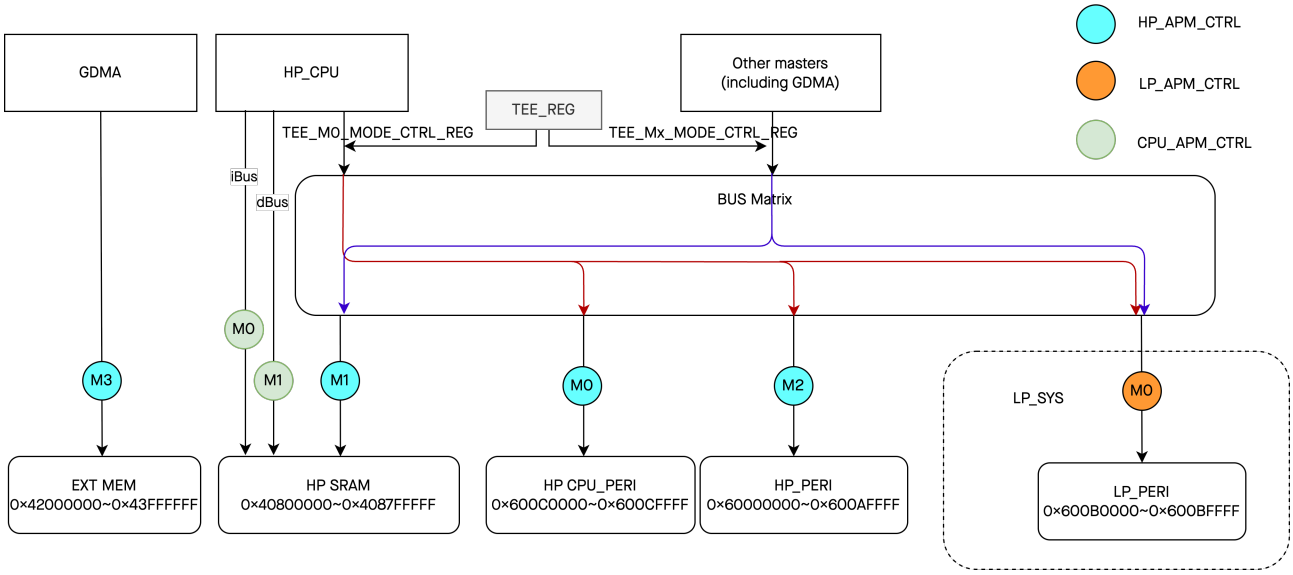



Figure 17.5-1. APM Controller Architecture

As shown in the figure, there are three APM controllers: HP_APM_CTRL, LP_APM_CTRL, and CPU_APM_CTRL, configured by their register modules HP_APM_REG, LP_APM_REG, and CPU_APM_REG, respectively.

- HP_APM_CTRL manages four access paths, namely M0–M3 highlighted in blue in Figure 17.5-1. Enable the permission management of each path by setting HP_APM_FUNC_CTRL_REG to 1 (enabled by default).
- LP_APM_CTRL manages one access path, namely M0 highlighted in orange in Figure 17.5-1. Enable the permission management of each path by setting LP_APM_FUNC_CTRL_REG to 1 (enabled by default).

- CPU_APM_CTRL manages two access paths, namely M0–M1 highlighted in green in Figure 17.5-1. Enable the permission management of each path by setting CPU_APM _FUNC_CTRL_REG to 1 (enabled by default).

Table 17.5-2 below shows the detailed information of the three APM controllers:

**Table 17.5-2. APM Controllers Configuration Registers**

| APM Controllers | Register Modules | Number of Access Paths | Enable Permission Management | Number of Configurable Address Ranges | Enable Address Ranges |
|---|---|---|---|---|---|
| HP_APM_CTRL | HP_APM_REG | 4 | HP_APM_ FUNC_CTRL_REG | 16 | HP_APM_REGION_FILTER_ EN_REG |
| LP_APM_CTRL | LP_APM_REG | 1 | LP_APM_ FUNC_CTRL_REG | 4 | LP_APM_REGION_FILTER_ EN_REG |
| CPU_APM_CTRL | CPU_APM_REG | 2 | CPU_APM _FUNC_CTRL_REG | 8 | CPU_APM_REGION_FILTER_ EN_REG |

#### 17.5.2.2 Address Ranges

- The HP_APM_REG register module configures up to 16 address ranges (i.e., region *n*, *n*=0-15) for HP_APM_CTRL. The start and end address of each region (address range) are configured by HP_APM_REGION*n*_ADDR_START and HP_APM_REGION*n*_ADDR_END, respectively. Configure the bit *n* of HP_APM_REGION_FILTER_EN_REG to enable region *n*. Region 0 (i.e., the first address range) is enabled by default.
- The LP_APM_REG register module configures up to four address ranges (i.e., region *n*, *n*=0-3) for LP_APM_CTRL. The start and end address of each region are configured by LP_APM_REGION*n*_ADDR_START and LP_APM_REGION*n*_ADDR_END. Configure the bit *n* of LP_APM_REGION_FILTER_EN_REG to enable region *n*. Region 0 (i.e., the first address range) is enabled by default.
- The CPU_APM_REG register module configures up to eight address ranges (i.e., region *n*, *n*=0-7) for CPU_APM_CTRL. The start and end address of each region are configured by CPU_APM_REGION*n*_ADDR_START and CPU_APM_REGION*n*_ADDR_END. Configure the bit *n* of CPU_APM_REGION_FILTER_EN_REG to enable region *n*. Region 0 (i.e., the first address range) is enabled by default.

When configuring the address ranges, the address requires a 4-byte alignment (the lower two bits of the address are 0). For example, the address range could be set to 0x4080000C ~ 0x40808774 or 0x600C0008 ~ 0×600CFF70.

#### 17.5.2.3 Access Permissions of Address Ranges

For each address range, the access permissions (read/write/execute) are configurable in different security modes:

- Masters in TEE mode always have read, write, and execute permissions in the address range.
- For masters in REE0, REE1 or REE2 mode, access permissions are configured with HP_APM_REGION*n*_ATTR_REG, LP_APM_REGION*n*_ATTR_REG, or CPU_APM_REGION*n*_ATTR_REG depending on the access path.

Different access paths managed by the same register module share the configuration of address ranges and access permissions. For example, the permission management of data paths HP_APM_CTRL M0-M3 follows the address ranges and access permissions of the 16 address ranges configured in the HP_APM_REG register module. Likewise, the permission management of data path LP_APM_CTRL M0 follows the address ranges and access permissions of the four address ranges configured in the LP_APM_REG register module. The permission management of data paths CPU_APM_CTRL M0-M1 follows the address ranges and access permissions of the eight address ranges configured in the CPU_APM_REG register module.

For HP SRAM, all masters access it through HP_APM_CTRL M1, except that the HP CPU accesses it through PMP and CPU_APM_CTRL. Suppose that HP_APM_M1_FUNC_EN is enabled and a master in REE1 mode needs to access HP SRAM. The whole process is as follows:

1. HP_APM_CTRL M1 first determines whether the address requested to access is within the 16 address ranges configured in the HP_APM_REG register module.
2. Assuming that the address requested to access is within the second address range, then HP_APM_CTRL M1 determines whether the address range is enabled, that is, whether bit 1 of HP_APM_REGION_FILTER_EN is 1.
3. If the address range is enabled, HP_APM_CTRL M1 checks whether the master has read permission for the second address range, that is, whether HP_APM_REGION1_R1_R in HP_APM_REGION1_ATTR_REG is 1. If valid, the read request will be allowed, otherwise, 0 will be returned.

**Note:**

- When the chip is powered up, only the HP CPU is in TEE mode by default, and other masters are in REE2 mode. By default, the APM controller blocks access requests from all masters in REE0, REE1, and REE2 modes.
- All registers listed in 17.8 *Register Summary* can only be configured by the masters that are in TEE security mode.
- The configured address ranges may overlap. For example, if one region is set to be unreadable and another region readable, then the overlapping area of the two regions is readable. The same rules apply to write and execute permissions.
- Configure HP_APM_REGION*n*_LOCK and LP_APM_REGION*n*_LOCK to lock the configuration of the start and end address of REGION*n* and its access permission. Only chip reset, system reset, and core reset can unlock the configurations and restore all configurations in APM registers to their default values.

## 17.6 Programming Procedure

For a master to access memory or peripheral registers, follow the programming procedures below:

1. Set the HP CPU to the machine mode (i.e., TEE mode).
2. Configure TEE_M*n*_MODE to choose the security mode of the master. The master ID *n* is defined in Table 17.5-1.
3. Configure HP_APM_REGION*n*_ADDR_START and HP_APM_REGION*n*_ADDR_END, LP_APM_REGION*n*_ADDR_START and LP_APM_REGION*n*_ADDR_END, or CPU_APM_REGION*n*_ADDR_START and CPU_APM_REGION*n*_ADDR_END to define the start and end address of the access address ranges.

4. Configure HP_APM_REGION*n*_ATTR_REG, LP_APM_REGION*n*_ATTR_REG, or CPU_APM_REGION*n*_ATTR_REG to set the access permissions for each region.
5. Configure bit *n* of HP_APM_REGION_FILTER_ EN_REG, LP_APM_REGION_FILTER_ EN_REG, or CPU_APM_REGION_FILTER_ EN_REG to enable permission checks for the corresponding address range.
6. Configure HP_APM_ FUNC_CTRL_REG, LP_APM _FUNC_CTRL_REG, or CPU_APM _FUNC_CTRL_REG to enable the permission management for different access paths (enabled by default).

Take I2S accessing HP SRAM via GDMA as an example, assuming that it is only allowed to read and write in the fourth address range 0x40805000 ~ 0x4080FFFF:

1. Configure the HP CPU to machine mode (i.e., TEE mode).
2. According to the master ID number in Table 17.5-1, set TEE_M19_MODE to 1, so that the security mode for I2S access via GDMA is REE0.
3. Configure HP_APM_REGION3_ADDR_START to 0x40805000 and HP_APM_REGION3_ADDR_END to 0x4080EFFC, respectively.
4. Set HP_APM_REGION3_RO_W and HP_APM_REGION3_RO_R to 1 to enable the access permissions to the address range.
5. Set bit 3 of HP_APM_REGION_FILTER_EN to 1 to enable region 3 (i.e., the fourth address range).
6. Set HP_APM_M1_FUNC_EN to 1 to enable the permission management for HP_APM_CTRL M1.

Through this configuration, I2S can read and write in the address range of 0x40805000 to 0x4080FFFF in HP SRAM using GDMA.

## 17.7 Illegal Access and Interrupts

If the information carried on the bus is inconsistent with the configuration, ESP32-C61 will regard it as an illegal access and proceed as follows:

- Denies the access request and returns the default value:
  - Returns 0 on instruction execution and read operations
  - Invalidate write operations
- Triggers interrupt

The APM controller will automatically record relevant information about the illegal access, including the master ID, security mode, access address, reasons for illegal access (address out of bounds or permission restrictions), and permission management result of each access path. All this information is obtained from relevant registers listed in Section 17.8 *Register Summary*.

Take the access path HP_APM_CTRL M0 as an example. When an illegal access occurs:

- HP_APM_M0_EXCEPTION_ID records the master ID.
- HP_APM_M0_EXCEPTION_MODE records the security mode.
- HP_APM_M0_EXCEPTION_ADDR records the access address.
- HP_APM_M0_EXCEPTION_STATUS records the reason for illegal access.

  - If the address requested to access is not within the enabled address ranges, bit 1 will be set to 1, indicating the address is out of bounds.
  - If the address requested to access is within the enabled address ranges, but the master does not have the read/write/execution permission within this region, then bit 0 will be set to 1, indicating permission restrictions.
- HP_APM_M0_EXCEPTION_REGION records the permission management result of each address range. This register has a total of 16 bits, corresponding to 16 address ranges, and bit 0 corresponds to the first address range. When the address to access is within a particular enabled address range, but the master does not have the corresponding read/write/execute permission within this address range, the corresponding bit of this register will be set to 1 .

ESP32-C61's APM controller can generate the following interrupt signals and send them to the Interrupt Matrix:

- HP_APM_M0_INTR
- HP_APM_M1_INTR
- HP_APM_M2_INTR
- HP_APM_M3_INTR
- LP_APM_M0_INTR
- CPU_APM_M0_INTR
- CPU_APM_M1_INTR

The interrupt signals correspond to the controlled access paths shown in Figure 17.5-1. If an illegal access occurs in a controlled access path, the corresponding interrupt will be generated.

# 17.8 Register Summary

## 17.8.1 HP_APM_REG

The addresses in this section are relative to the Access Permission Management Controller (HP_APM) base address provided in Table 5.3-2 in Chapter 5 *System and Memory*.

The abbreviations given in Column **Access** are explained in Section *Access Types for Registers*.

| Name | Description | Address | Access |
|---|---|---|---|
| **Configuration Registers** | | | |
| HP_APM_REGION_FILTER_EN_REG | Region enable register | 0x0000 | R/W |
| HP_APM_REGION0_ADDR_START_REG | Region address register | 0x0004 | R/W |
| HP_APM_REGION0_ADDR_END_REG | Region address register | 0x0008 | R/W |
| HP_APM_REGION1_ADDR_START_REG | Region address register | 0x0010 | R/W |
| HP_APM_REGION1_ADDR_END_REG | Region address register | 0x0014 | R/W |
| HP_APM_REGION2_ADDR_START_REG | Region address register | 0x001C | R/W |
| HP_APM_REGION2_ADDR_END_REG | Region address register | 0x0020 | R/W |
| HP_APM_REGION3_ADDR_START_REG | Region address register | 0x0028 | R/W |

| Name | Description | Address | Access |
|---|---|---|---|
| HP_APM_REGION3_ADDR_END_REG | Region address register | 0x002C | R/W |
| HP_APM_REGION4_ADDR_START_REG | Region address register | 0x0034 | R/W |
| HP_APM_REGION4_ADDR_END_REG | Region address register | 0x0038 | R/W |
| HP_APM_REGION5_ADDR_START_REG | Region address register | 0x0040 | R/W |
| HP_APM_REGION5_ADDR_END_REG | Region address register | 0x0044 | R/W |
| HP_APM_REGION6_ADDR_START_REG | Region address register | 0x004C | R/W |
| HP_APM_REGION6_ADDR_END_REG | Region address register | 0x0050 | R/W |
| HP_APM_REGION7_ADDR_START_REG | Region address register | 0x0058 | R/W |
| HP_APM_REGION7_ADDR_END_REG | Region address register | 0x005C | R/W |
| HP_APM_REGION8_ADDR_START_REG | Region address register | 0x0064 | R/W |
| HP_APM_REGION8_ADDR_END_REG | Region address register | 0x0068 | R/W |
| HP_APM_REGION9_ADDR_START_REG | Region address register | 0x0070 | R/W |
| HP_APM_REGION9_ADDR_END_REG | Region address register | 0x0074 | R/W |
| HP_APM_REGION10_ADDR_START_REG | Region address register | 0x007C | R/W |
| HP_APM_REGION10_ADDR_END_REG | Region address register | 0x0080 | R/W |
| HP_APM_REGION11_ADDR_START_REG | Region address register | 0x0088 | R/W |
| HP_APM_REGION11_ADDR_END_REG | Region address register | 0x008C | R/W |
| HP_APM_REGION12_ADDR_START_REG | Region address register | 0x0094 | R/W |
| HP_APM_REGION12_ADDR_END_REG | Region address register | 0x0098 | R/W |
| HP_APM_REGION13_ADDR_START_REG | Region address register | 0x00A0 | R/W |
| HP_APM_REGION13_ADDR_END_REG | Region address register | 0x00A4 | R/W |
| HP_APM_REGION14_ADDR_START_REG | Region address register | 0x00AC | R/W |
| HP_APM_REGION14_ADDR_END_REG | Region address register | 0x00B0 | R/W |
| HP_APM_REGION15_ADDR_START_REG | Region address register | 0x00B8 | R/W |
| HP_APM_REGION15_ADDR_END_REG | Region address register | 0x00BC | R/W |
| HP_APM_REGION0_ATTR_REG | Region access permissions configuration register | 0x000C | R/W |
| HP_APM_REGION1_ATTR_REG | Region access permissions configuration register | 0x0018 | R/W |
| HP_APM_REGION2_ATTR_REG | Region access permissions configuration register | 0x0024 | R/W |
| HP_APM_REGION3_ATTR_REG | Region access permissions configuration register | 0x0030 | R/W |
| HP_APM_REGION4_ATTR_REG | Region access permissions configuration register | 0x003C | R/W |
| HP_APM_REGION5_ATTR_REG | Region access permissions configuration register | 0x0048 | R/W |
| HP_APM_REGION6_ATTR_REG | Region access permissions configuration register | 0x0054 | R/W |
| HP_APM_REGION7_ATTR_REG | Region access permissions configuration register | 0x0060 | R/W |
| HP_APM_REGION8_ATTR_REG | Region access permissions configuration register | 0x006C | R/W |

| Name | Description | Address | Access |
|---|---|---|---|
| HP_APM_REGION9_ATTR_REG | Region access permissions configuration register | 0x0078 | R/W |
| HP_APM_REGION10_ATTR_REG | Region access permissions configuration register | 0x0084 | R/W |
| HP_APM_REGION11_ATTR_REG | Region access permissions configuration register | 0x0090 | R/W |
| HP_APM_REGION12_ATTR_REG | Region access permissions configuration register | 0x009C | R/W |
| HP_APM_REGION13_ATTR_REG | Region access permissions configuration register | 0x00A8 | R/W |
| HP_APM_REGION14_ATTR_REG | Region access permissions configuration register | 0x00B4 | R/W |
| HP_APM_REGION15_ATTR_REG | Region access permissions configuration register | 0x00C0 | R/W |
| HP_APM_FUNC_CTRL_REG | APM access path permission management register | 0x00C4 | R/W |
| **Status Registers** | | | |
| HP_APM_M0_STATUS_REG | HP_APM_CTRL M0 status register | 0x00C8 | RO |
| HP_APM_M0_STATUS_CLR_REG | HP_APM_CTRL M0 status clear register | 0x00CC | WT |
| HP_APM_M0_EXCEPTION_INFO0_REG | HP_APM_CTRL M0 exception information register | 0x00D0 | RO |
| HP_APM_M0_EXCEPTION_INFO1_REG | HP_APM_CTRL M0 exception information register | 0x00D4 | RO |
| HP_APM_M1_STATUS_REG | HP_APM_CTRL M1 status register | 0x00D8 | RO |
| HP_APM_M1_STATUS_CLR_REG | HP_APM_CTRL M1 status clear register | 0x00DC | WT |
| HP_APM_M1_EXCEPTION_INFO0_REG | HP_APM_CTRL M1 exception information register | 0x00E0 | RO |
| HP_APM_M1_EXCEPTION_INFO1_REG | HP_APM_CTRL M1 exception information register | 0x00E4 | RO |
| HP_APM_M2_STATUS_REG | HP_APM_CTRL M2 status register | 0x00E8 | RO |
| HP_APM_M2_STATUS_CLR_REG | HP_APM_CTRL M2 status clear register | 0x00EC | WT |
| HP_APM_M2_EXCEPTION_INFO0_REG | HP_APM_CTRL M2 exception information register | 0x00F0 | RO |
| HP_APM_M2_EXCEPTION_INFO1_REG | HP_APM_CTRL M2 exception information register | 0x00F4 | RO |
| HP_APM_M3_STATUS_REG | HP_APM_CTRL M3 status register | 0x00F8 | RO |
| HP_APM_M3_STATUS_CLR_REG | HP_APM_CTRL M3 status clear register | 0x00FC | WT |
| HP_APM_M3_EXCEPTION_INFO0_REG | HP_APM_CTRL M3 exception information register | 0x0100 | RO |
| HP_APM_M3_EXCEPTION_INFO1_REG | HP_APM_CTRL M3 exception information register | 0x0104 | RO |
| **Interrupt Registers** | | | |
| HP_APM_INT_EN_REG | HP_APM_CTRL M0/1/2/3 interrupt enable register | 0x0118 | R/W |

| Name | Description | Address | Access |
|---|---|---|---|
| **Clock Gating Registers** | | | |
| HP_APM_CLOCK_GATE_REG | Clock gating register | 0x07F8 | R/W |
| **Version Control Registers** | | | |
| HP_APM_DATE_REG | Version control register | 0x07FC | R/W |

### 17.8.2 LP_APM_REG

The addresses in this section are relative to the Low-Power Access Permission Management (LP_APM) base address provided in Table 5.3-2 in Chapter 5 *System and Memory*.

The abbreviations given in Column **Access** are explained in Section *Access Types for Registers*.

| Name | Description | Address | Access |
|---|---|---|---|
| **Configuration Registers** | | | |
| LP_APM_REGION_FILTER_EN_REG | Region enable register | 0x0000 | R/W |
| LP_APM_REGION0_ADDR_START_REG | Region address register | 0x0004 | R/W |
| LP_APM_REGION0_ADDR_END_REG | Region address register | 0x0008 | R/W |
| LP_APM_REGION1_ADDR_START_REG | Region address register | 0x0010 | R/W |
| LP_APM_REGION1_ADDR_END_REG | Region address register | 0x0014 | R/W |
| LP_APM_REGION2_ADDR_START_REG | Region address register | 0x001C | R/W |
| LP_APM_REGION2_ADDR_END_REG | Region address register | 0x0020 | R/W |
| LP_APM_REGION3_ADDR_START_REG | Region address register | 0x0028 | R/W |
| LP_APM_REGION3_ADDR_END_REG | Region address register | 0x002C | R/W |
| LP_APM_REGION0_ATTR_REG | Region access permissions configuration register | 0x000C | R/W |
| LP_APM_REGION1_ATTR_REG | Region access permissions configuration register | 0x0018 | R/W |
| LP_APM_REGION2_ATTR_REG | Region access permissions configuration register | 0x0024 | R/W |
| LP_APM_REGION3_ATTR_REG | Region access permissions configuration register | 0x0030 | R/W |
| LP_APM_FUNC_CTRL_REG | APM access path permission management register | 0x00C4 | R/W |
| **Status Registers** | | | |
| LP_APM_M0_STATUS_REG | LP_APM_CTRL M0 status register | 0x00C8 | RO |
| LP_APM_M0_STATUS_CLR_REG | LP_APM_CTRL M0 status clear register | 0x00CC | WT |
| LP_APM_M0_EXCEPTION_INFO0_REG | LP_APM_CTRL M0 exception information register | 0x00D0 | RO |
| LP_APM_M0_EXCEPTION_INFO1_REG | LP_APM_CTRL M0 exception information register | 0x00D4 | RO |
| **Interrupt Registers** | | | |
| LP_APM_INT_EN_REG | LP_APM_CTRL M0 interrupt enable register | 0x00E8 | R/W |
| **Clock Gating Registers** | | | |

| Name | Description | Address | Access |
|---|---|---|---|
| LP_APM_CLOCK_GATE_REG | Clock gating register | 0x00EC | R/W |
| **Version Control Registers** | | | |
| LP_APM_DATE_REG | Version control register | 0x00FC | R/W |

### 17.8.3 CPU_APM_REG

The addresses in this section are relative to the CPU_APM base address provided in Table 5.3-2 in Chapter 5 *System and Memory*.

The abbreviations given in Column **Access** are explained in Section *Access Types for Registers*.

| Name | Description | Address | Access |
|---|---|---|---|
| **Configuration Registers** | | | |
| CPU_APM_REGION_FILTER_EN_REG | Region enable register | 0x0000 | R/W |
| **Region Address Registers** | | | |
| CPU_APM_REGION0_ADDR_START_REG | Region address register | 0x0004 | varies |
| CPU_APM_REGION0_ADDR_END_REG | Region address register | 0x0008 | varies |
| CPU_APM_REGION1_ADDR_START_REG | Region address register | 0x0010 | varies |
| CPU_APM_REGION1_ADDR_END_REG | Region address register | 0x0014 | varies |
| CPU_APM_REGION2_ADDR_START_REG | Region address register | 0x001C | varies |
| CPU_APM_REGION2_ADDR_END_REG | Region address register | 0x0020 | varies |
| CPU_APM_REGION3_ADDR_START_REG | Region address register | 0x0028 | varies |
| CPU_APM_REGION3_ADDR_END_REG | Region address register | 0x002C | varies |
| CPU_APM_REGION4_ADDR_START_REG | Region address register | 0x0034 | varies |
| CPU_APM_REGION4_ADDR_END_REG | Region address register | 0x0038 | varies |
| CPU_APM_REGION5_ADDR_START_REG | Region address register | 0x0040 | varies |
| CPU_APM_REGION5_ADDR_END_REG | Region address register | 0x0044 | varies |
| CPU_APM_REGION6_ADDR_START_REG | Region address register | 0x004C | varies |
| CPU_APM_REGION6_ADDR_END_REG | Region address register | 0x0050 | varies |
| CPU_APM_REGION7_ADDR_START_REG | Region address register | 0x0058 | varies |
| CPU_APM_REGION7_ADDR_END_REG | Region address register | 0x005C | varies |
| CPU_APM_REGION0_ATTR_REG | Region access permissions configuration register | 0x000C | R/W |
| CPU_APM_REGION1_ATTR_REG | Region access permissions configuration register | 0x0018 | R/W |
| CPU_APM_REGION2_ATTR_REG | Region access permissions configuration register | 0x0024 | R/W |
| CPU_APM_REGION3_ATTR_REG | Region access permissions configuration register | 0x0030 | R/W |
| CPU_APM_REGION4_ATTR_REG | Region access permissions configuration register | 0x003C | R/W |
| CPU_APM_REGION5_ATTR_REG | Region access permissions configuration register | 0x0048 | R/W |

| Name | Description | Address | Access |
|---|---|---|---|
| CPU_APM_REGION6_ATTR_REG | Region access permissions configuration register | 0x0054 | R/W |
| CPU_APM_REGION7_ATTR_REG | Region access permissions configuration register | 0x0060 | R/W |
| CPU_APM_FUNC_CTRL_REG | APM access path permission management register | 0x00C4 | R/W |
| **Status Registers** | | | |
| CPU_APM_M0_STATUS_REG | CPU_APM_CTRL M0 status register | 0x00C8 | RO |
| CPU_APM_M0_STATUS_CLR_REG | CPU_APM_CTRL M0 status clear register | 0x00CC | WT |
| CPU_APM_M0_EXCEPTION_INFO0_REG | CPU_APM_CTRL M0 exception information register | 0x00D0 | RO |
| CPU_APM_M0_EXCEPTION_INFO1_REG | CPU_APM_CTRL M0 exception information register | 0x00D4 | RO |
| CPU_APM_M1_STATUS_REG | CPU_APM_CTRL M1 status register | 0x00D8 | RO |
| CPU_APM_M1_STATUS_CLR_REG | CPU_APM_CTRL M1 status clear register | 0x00DC | WT |
| CPU_APM_M1_EXCEPTION_INFO0_REG | CPU_APM_CTRL M1 exception information register | 0x00E0 | RO |
| CPU_APM_M1_EXCEPTION_INFO1_REG | CPU_APM_CTRL M1 exception information register | 0x00E4 | RO |
| **Interrupt Registers** | | | |
| CPU_APM_INT_EN_REG | CPU_APM_CTRL M0/1 interrupt enable register | 0x0118 | R/W |
| **Clock Gating Registers** | | | |
| CPU_APM_CLOCK_GATE_REG | Clock gating register | 0x07F8 | R/W |
| **Version Control Registers** | | | |
| CPU_APM_DATE_REG | Version control register | 0x07FC | R/W |

### 17.8.4 HP_TEE_REG

The addresses in this section are relative to the Trusted Execution Environment (TEE) Register provided in Table 5.3-2 in Chapter 5 *System and Memory*.

The abbreviations given in Column **Access** are explained in Section *Access Types for Registers*.

| Name | Description | Address | Access |
|---|---|---|---|
| **Configuration Registers** | | | |
| TEE_M0_MODE_CTRL_REG | Security mode configuration register | 0x0000 | R/W |
| TEE_M1_MODE_CTRL_REG | Security mode configuration register | 0x0004 | R/W |
| TEE_M2_MODE_CTRL_REG | Security mode configuration register | 0x0008 | R/W |
| TEE_M3_MODE_CTRL_REG | Security mode configuration register | 0x000C | R/W |
| TEE_M4_MODE_CTRL_REG | Security mode configuration register | 0x0010 | R/W |
| TEE_M5_MODE_CTRL_REG | Security mode configuration register | 0x0014 | R/W |
| TEE_M6_MODE_CTRL_REG | Security mode configuration register | 0x0018 | R/W |
| TEE_M7_MODE_CTRL_REG | Security mode configuration register | 0x001C | R/W |

| Name | Description | Address | Access |
|---|---|---|---|
| TEE_M8_MODE_CTRL_REG | Security mode configuration register | 0x0020 | R/W |
| TEE_M9_MODE_CTRL_REG | Security mode configuration register | 0x0024 | R/W |
| TEE_M10_MODE_CTRL_REG | Security mode configuration register | 0x0028 | R/W |
| TEE_M11_MODE_CTRL_REG | Security mode configuration register | 0x002C | R/W |
| TEE_M12_MODE_CTRL_REG | Security mode configuration register | 0x0030 | R/W |
| TEE_M13_MODE_CTRL_REG | Security mode configuration register | 0x0034 | R/W |
| TEE_M14_MODE_CTRL_REG | Security mode configuration register | 0x0038 | R/W |
| TEE_M15_MODE_CTRL_REG | Security mode configuration register | 0x003C | R/W |
| TEE_M16_MODE_CTRL_REG | Security mode configuration register | 0x0040 | R/W |
| TEE_M17_MODE_CTRL_REG | Security mode configuration register | 0x0044 | R/W |
| TEE_M18_MODE_CTRL_REG | Security mode configuration register | 0x0048 | R/W |
| TEE_M19_MODE_CTRL_REG | Security mode configuration register | 0x004C | R/W |
| TEE_M20_MODE_CTRL_REG | Security mode configuration register | 0x0050 | R/W |
| TEE_M21_MODE_CTRL_REG | Security mode configuration register | 0x0054 | R/W |
| TEE_M22_MODE_CTRL_REG | Security mode configuration register | 0x0058 | R/W |
| TEE_M23_MODE_CTRL_REG | Security mode configuration register | 0x005C | R/W |
| TEE_M24_MODE_CTRL_REG | Security mode configuration register | 0x0060 | R/W |
| TEE_M25_MODE_CTRL_REG | Security mode configuration register | 0x0064 | R/W |
| TEE_M26_MODE_CTRL_REG | Security mode configuration register | 0x0068 | R/W |
| TEE_M27_MODE_CTRL_REG | Security mode configuration register | 0x006C | R/W |
| TEE_M28_MODE_CTRL_REG | Security mode configuration register | 0x0070 | R/W |
| TEE_M29_MODE_CTRL_REG | Security mode configuration register | 0x0074 | R/W |
| TEE_M30_MODE_CTRL_REG | Security mode configuration register | 0x0078 | R/W |
| TEE_M31_MODE_CTRL_REG | Security mode configuration register | 0x007C | R/W |
| **Clock Gating Registers** | | | |
| TEE_CLOCK_GATE_REG | Clock gating register | 0x0080 | R/W |
| **Version Control Registers** | | | |
| TEE_DATE_REG | Version control register | 0x0FFC | R/W |

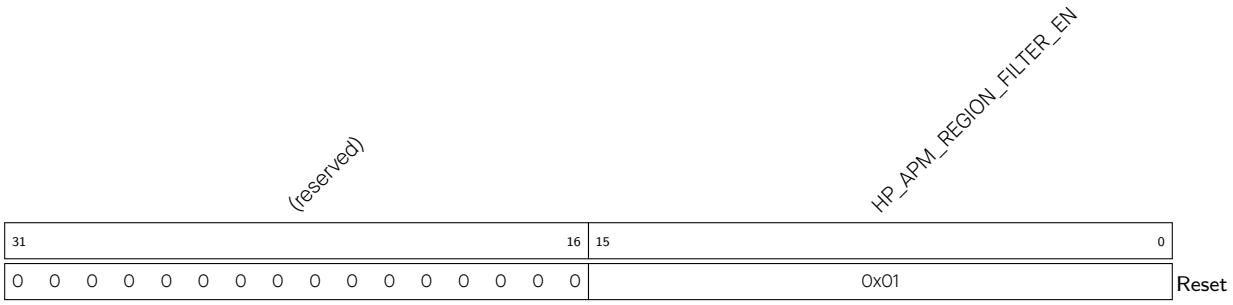
# 17.9 Registers

## 17.9.1 HP_APM_REG

The addresses in this section are relative to the HP_APM base address provided in Table 5.3-2 in Chapter 5 *System and Memory*.

For how to program reserved fields, please refer to Section *Programming Reserved Register Field*.

**Register 17.1. HP_APM_REGION_FILTER_EN_REG (0x0000)**



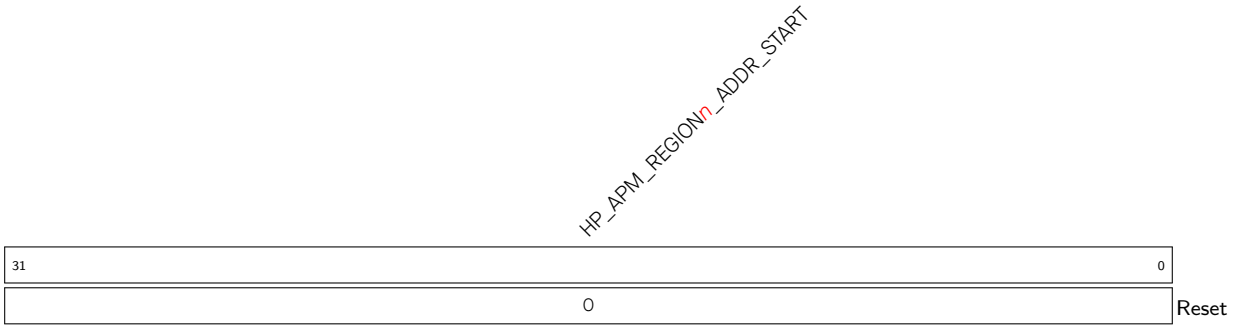

**HP_APM_REGION_FILTER_EN** Configure bit *n* (0-15) to enable permission checks for region *n* (0-15).

0: Disable

1: Enable

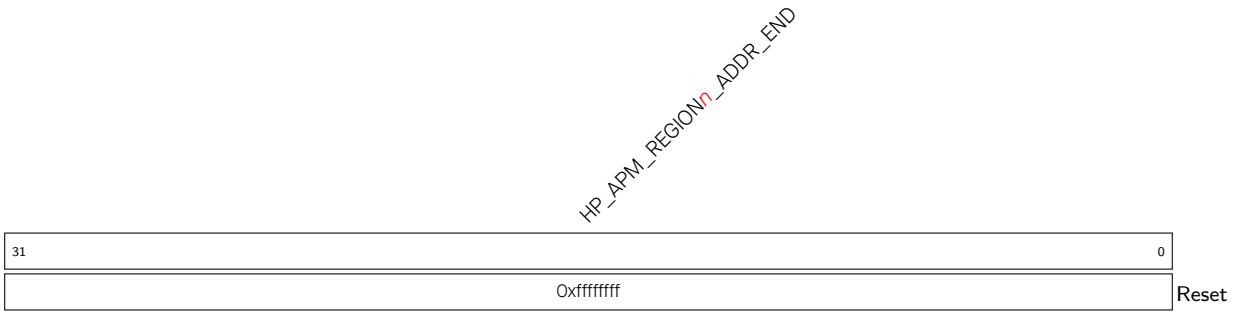
(R/W)

**Register 17.2. HP_APM_REGION*n*_ADDR_START_REG (*n*: 0-15) (0x0004+0xC**n*)**



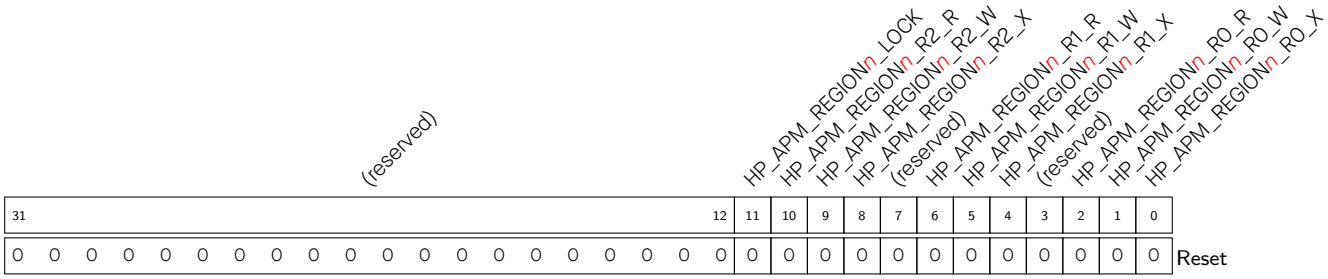

**HP_APM_REGION*n*_ADDR_START** Configures the start address of region *n*. (R/W)

**Register 17.3. HP_APM_REGION*n*_ADDR_END_REG (*n*: 0-15) (0x0008+0xC**n*)**




**HP_APM_REGION*n*_ADDR_END** Configures the end address of region *n*. (R/W)

**Register 17.4. HP_APM_REGION*n*_ATTR_REG (*n*: 0-15) (0x000C+0xC**n*)**




**HP_APM_REGION*n*_R0_X** Configures the execution permission in region *n* in REE0 mode. (R/W)

**HP_APM_REGION*n*_R0_W** Configures the write permission in region *n* in REE0 mode. (R/W)

**HP_APM_REGION*n*_R0_R** Configures the read permission in region *n* in REE0 mode. (R/W)

**HP_APM_REGION*n*_R1_X** Configures the execution permission in region *n* in REE1 mode. (R/W)

**HP_APM_REGION*n*_R1_W** Configures the write permission in region *n* in REE1 mode. (R/W)

**HP_APM_REGION*n*_R1_R** Configures the read permission in region *n* in REE1 mode. (R/W)

**HP_APM_REGION*n*_R2_X** Configures the execution permission in region *n* in REE2 mode. (R/W)

**HP_APM_REGION*n*_R2_W** Configures the write permission in region *n* in REE2 mode. (R/W)

**HP_APM_REGION*n*_R2_R** Configures the read permission in region *n* in REE2 mode. (R/W)

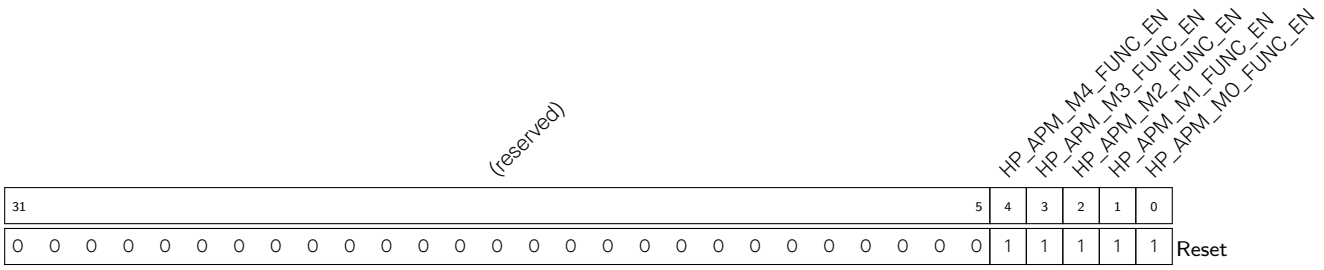
**HP_APM_REGION*n*_LOCK** Configures to lock the value of region *n* configuration registers (HP_APM_REGION*n*_ADDR_START_REG, HP_APM_REGION*n*_ADDR_END_REG and HP_APM_REGION*n*_ATTR_REG).

0: Do not lock

1: Lock

(R/W)

Register 17.5. HP_APM_FUNC_CTRL_REG (0x00C4)




**HP_APM_M0_FUNC_EN** Configures to enable permission management for HP_APM_CTRL M0. (R/W)

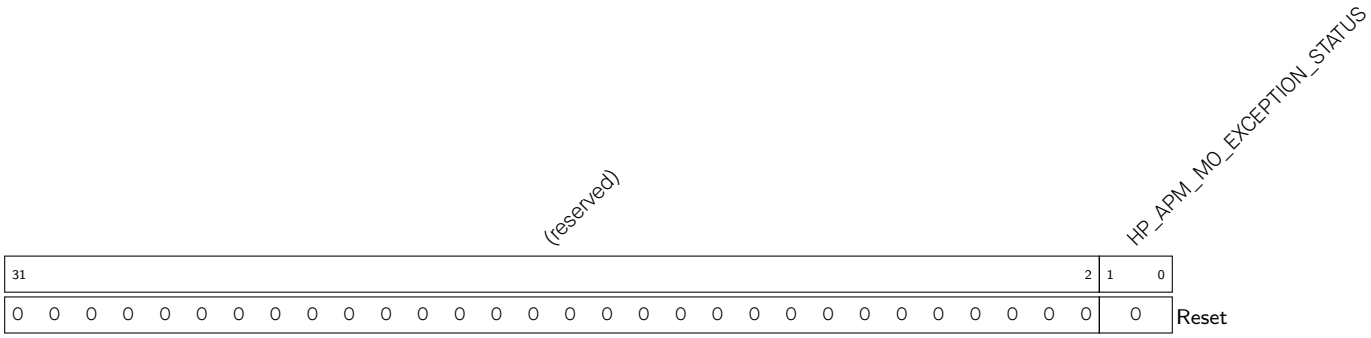
**HP_APM_M1_FUNC_EN** Configures to enable permission management for HP_APM_CTRL M1. (R/W)

**HP_APM_M2_FUNC_EN** Configures to enable permission management for HP_APM_CTRL M2. (R/W)

**HP_APM_M3_FUNC_EN** Configures to enable permission management for HP_APM_CTRL M3. (R/W)

**HP_APM_M4_FUNC_EN** Configures to enable permission management for HP_APM_CTRL M4. (R/W)

Register 17.6. HP_APM_M0_STATUS_REG (0x00C8)



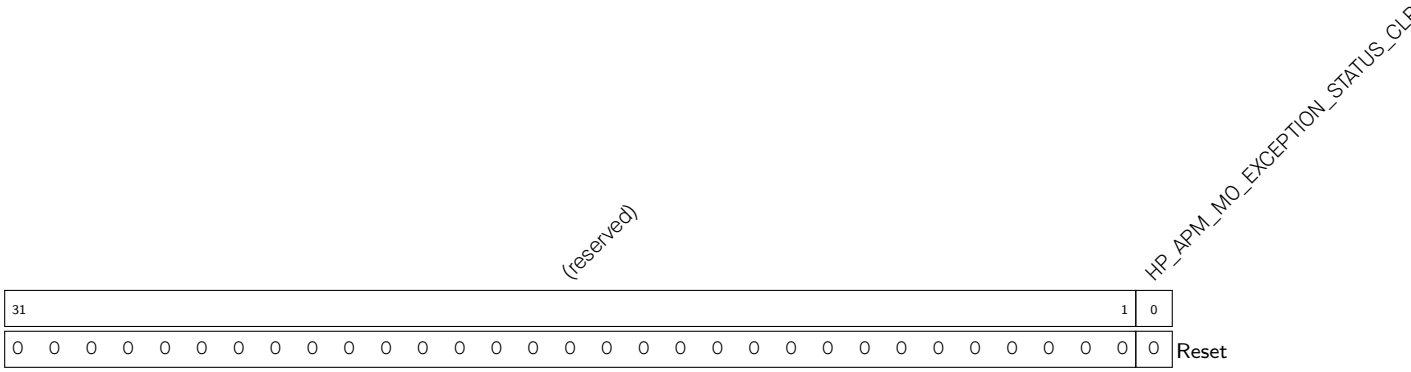

**HP_APM_M0_EXCEPTION_STATUS** Represents exception status.

bit0: 1 represents permission restrictions

bit1: 1 represents address out of bounds

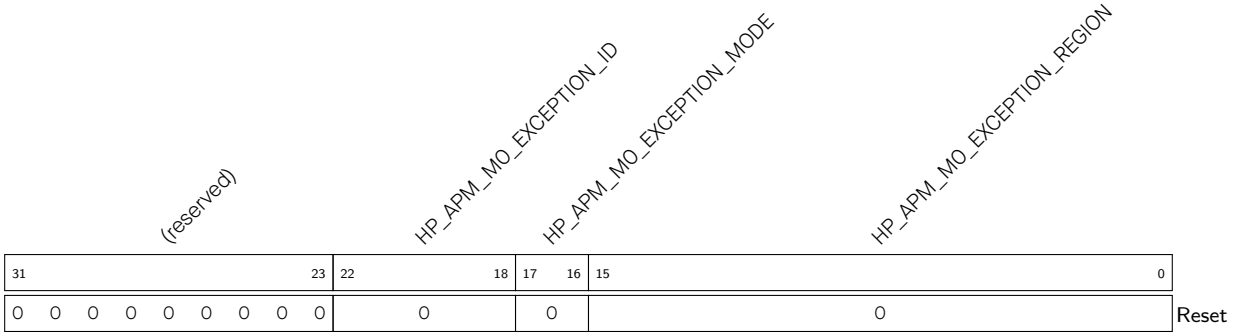
(RO)

Register 17.7. HP_APM_M0_STATUS_CLR_REG (0x00CC)




**HP_APM_M0_EXCEPTION_STATUS_CLR** Configures to clear exception status. (WT)

Register 17.8. HP_APM_M0_EXCEPTION_INFO0_REG (0x00D0)

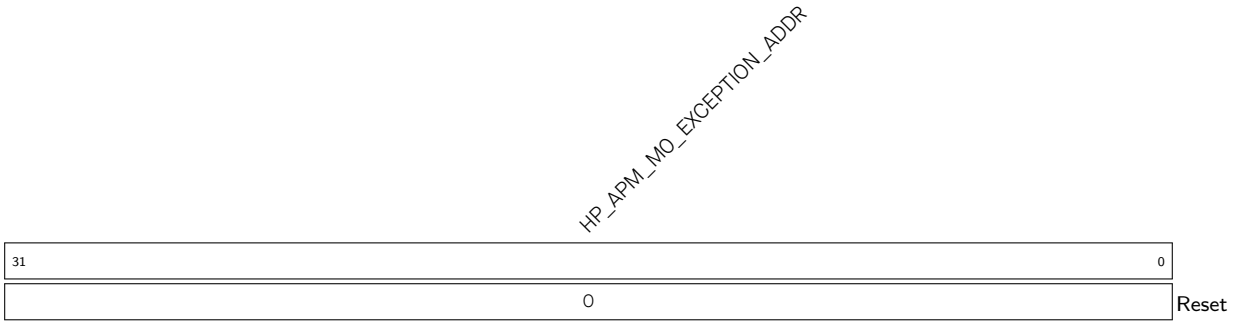



**HP_APM_M0_EXCEPTION_REGION** Represents the region where an exception occurs. (RO)

**HP_APM_M0_EXCEPTION_MODE** Represents the master's security mode when an exception occurs. (RO)

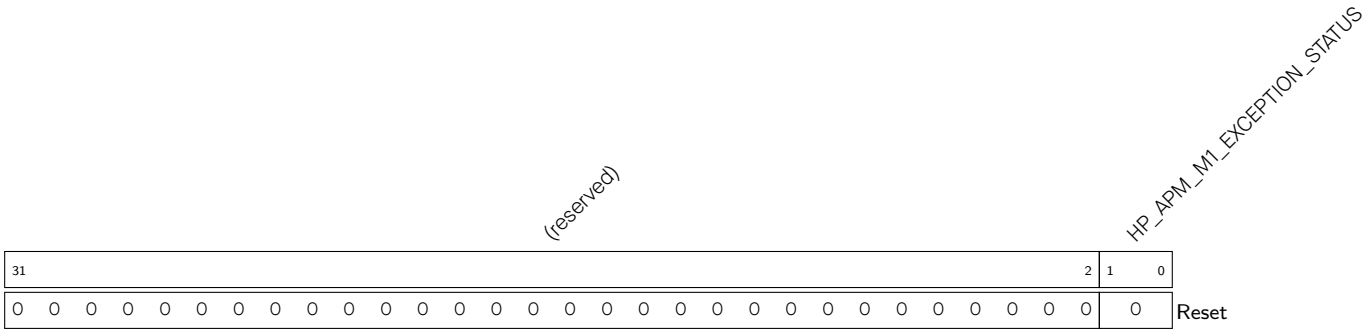
**HP_APM_M0_EXCEPTION_ID** Represents master ID when an exception occurs. (RO)

Register 17.9. HP_APM_M0_EXCEPTION_INFO1_REG (0x00D4)




**HP_APM_M0_EXCEPTION_ADDR** Represents the access address when an exception occurs. (RO)

Register 17.10. HP_APM_M1_STATUS_REG (0x00D8)



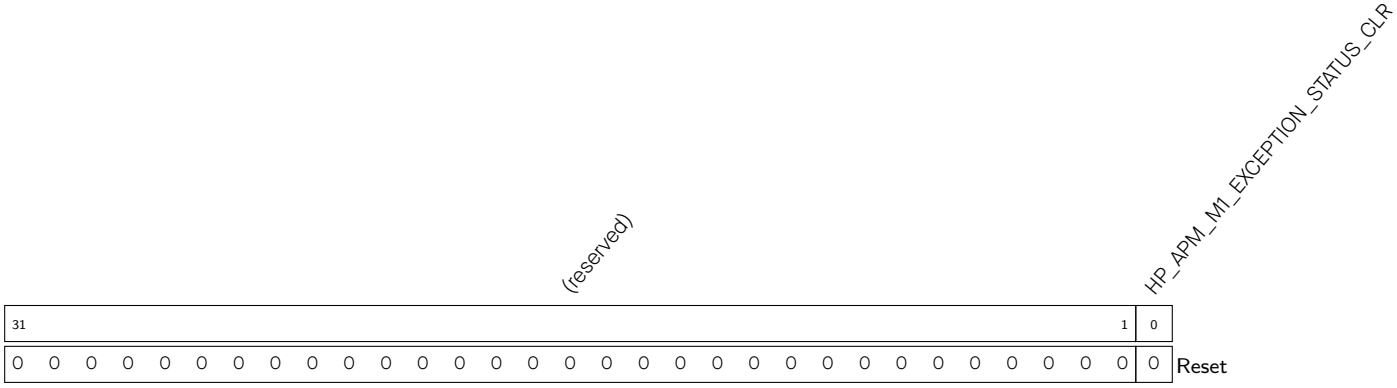

**HP_APM_M1_EXCEPTION_STATUS** Represents exception status.

bit0: 1 represents permission restrictions

bit1: 1 represents address out of bounds

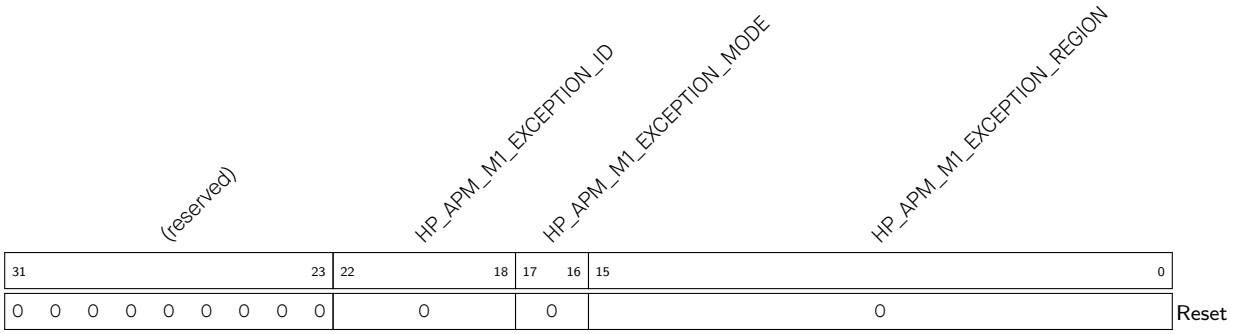
(RO)

Register 17.11. HP_APM_M1_STATUS_CLR_REG (0x00DC)




**HP_APM_M1_EXCEPTION_STATUS_CLR** Configures to clear exception status. (WT)

**Register 17.12. HP_APM_M1_EXCEPTION_INFO0_REG (0x00E0)**

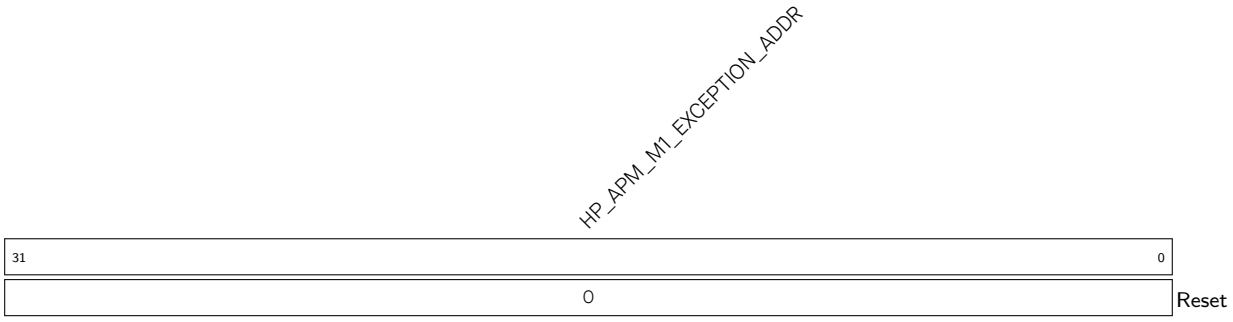



**HP_APM_M1_EXCEPTION_REGION** Represents the region where an exception occurs. (RO)

**HP_APM_M1_EXCEPTION_MODE** Represents the master's security mode when an exception occurs. (RO)

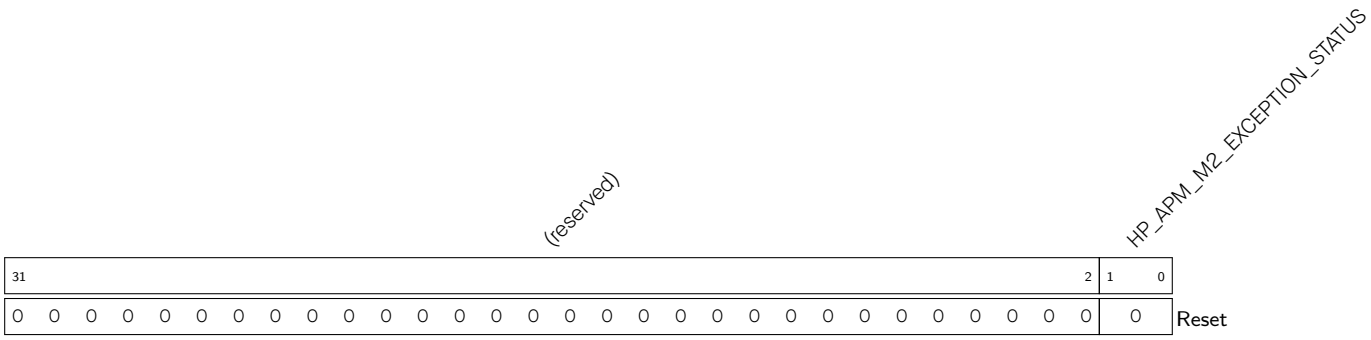
**HP_APM_M1_EXCEPTION_ID** Represents master ID when an exception occurs. (RO)

**Register 17.13. HP_APM_M1_EXCEPTION_INFO1_REG (0x00E4)**




**HP_APM_M1_EXCEPTION_ADDR** Represents the access address when an exception occurs. (RO)

Register 17.14. HP_APM_M2_STATUS_REG (0x00E8)



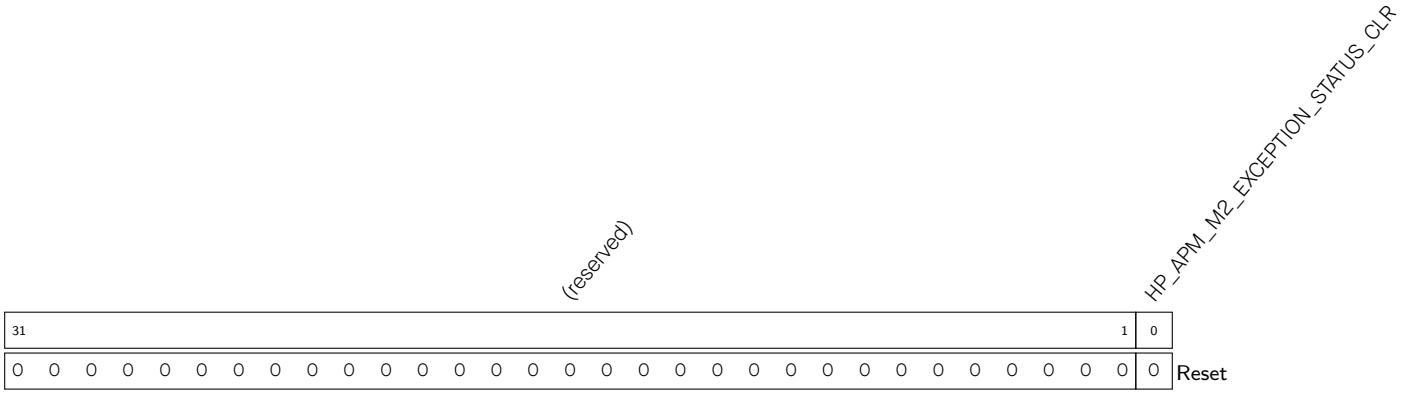

**HP_APM_M2_EXCEPTION_STATUS** Represents exception status.

bit0: 1 represents permission restrictions

bit1: 1 represents address out of bounds

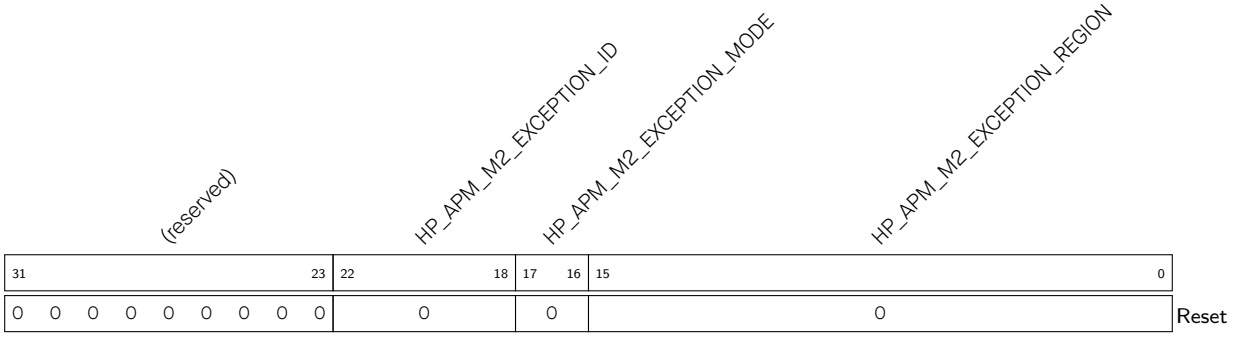
(RO)

Register 17.15. HP_APM_M2_STATUS_CLR_REG (0x00EC)




**HP_APM_M2_EXCEPTION_STATUS_CLR** Configures to clear exception status. (WT)

Register 17.16. HP_APM_M2_EXCEPTION_INFO0_REG (0x00F0)

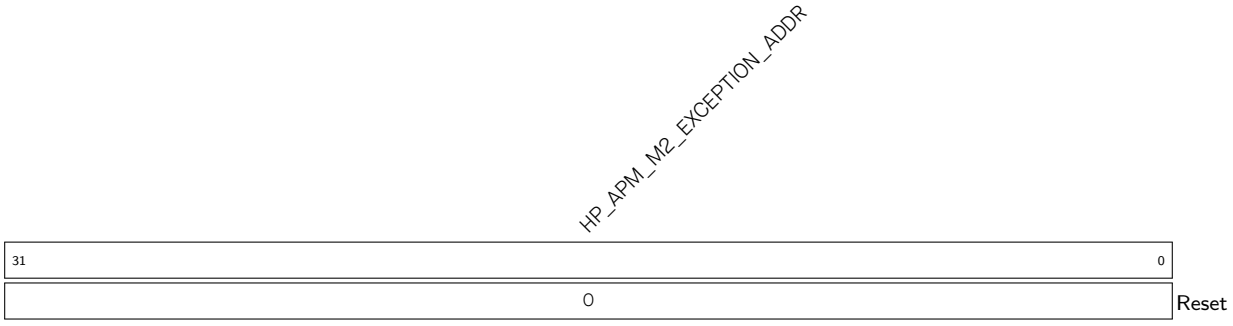



**HP_APM_M2_EXCEPTION_REGION** Represents the region where an exception occurs. (RO)

**HP_APM_M2_EXCEPTION_MODE** Represents the master's security mode when an exception occurs. (RO)

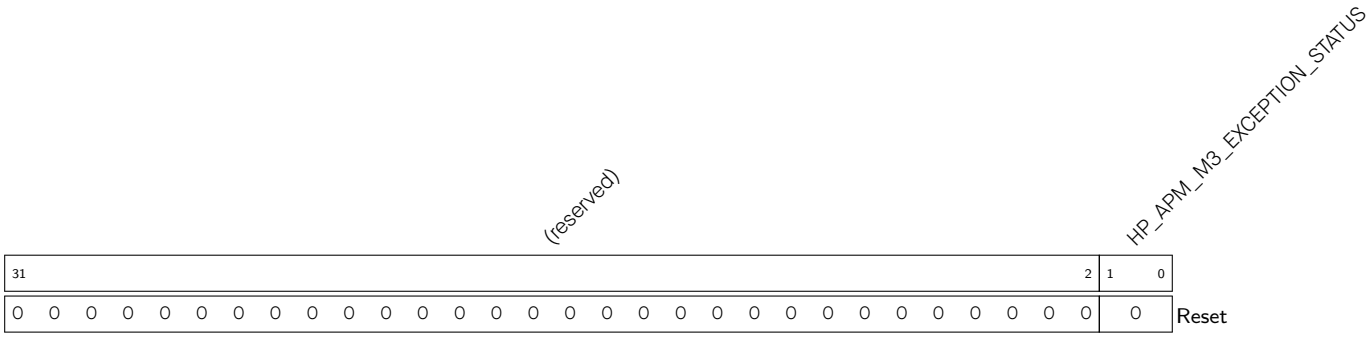
**HP_APM_M2_EXCEPTION_ID** Represents master ID when an exception occurs. (RO)

Register 17.17. HP_APM_M2_EXCEPTION_INFO1_REG (0x00F4)




**HP_APM_M2_EXCEPTION_ADDR** Represents the access address when an exception occurs. (RO)

Register 17.18. HP_APM_M3_STATUS_REG (0x00F8)



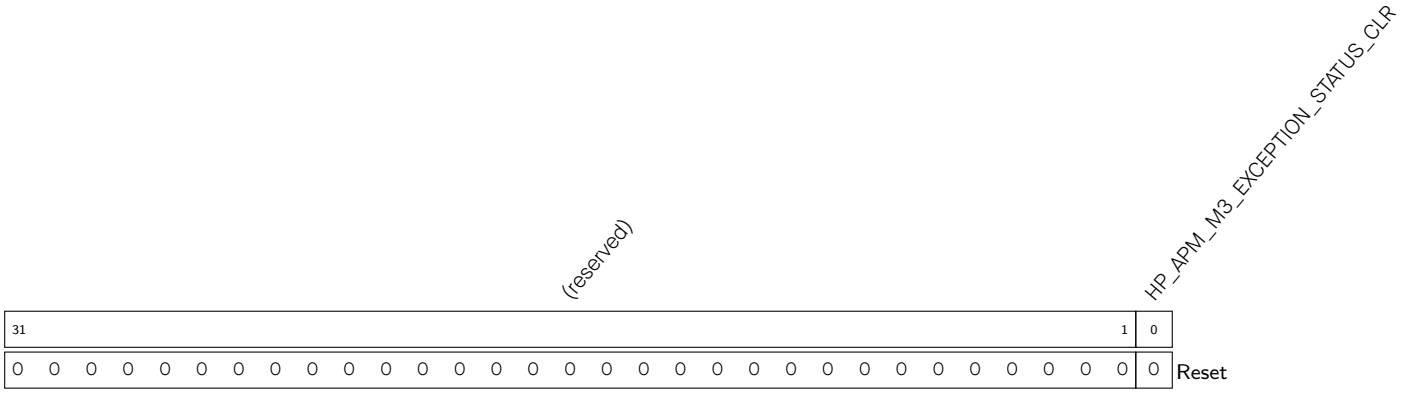

**HP_APM_M3_EXCEPTION_STATUS** Represents exception status.

bit0: 1 represents permission restrictions

bit1: 1 represents address out of bounds

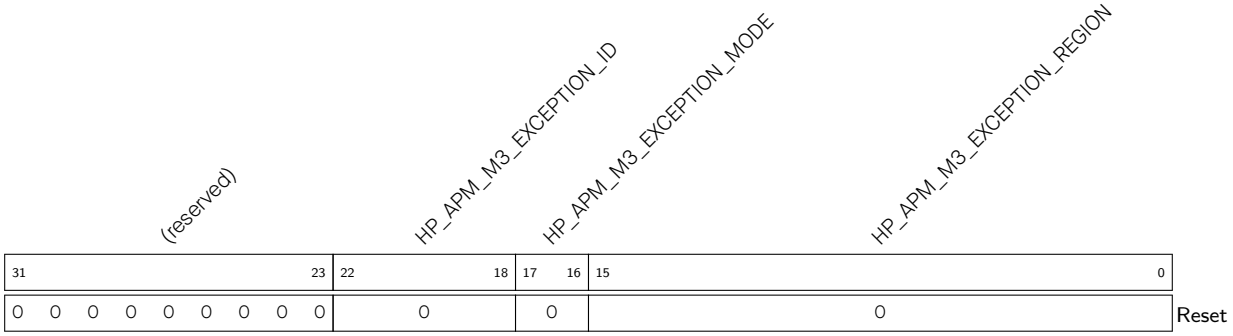
(RO)

Register 17.19. HP_APM_M3_STATUS_CLR_REG (0x00FC)




**HP_APM_M3_EXCEPTION_STATUS_CLR** Configures to clear exception status. (WT)

Register 17.20. HP_APM_M3_EXCEPTION_INFO0_REG (0x0100)

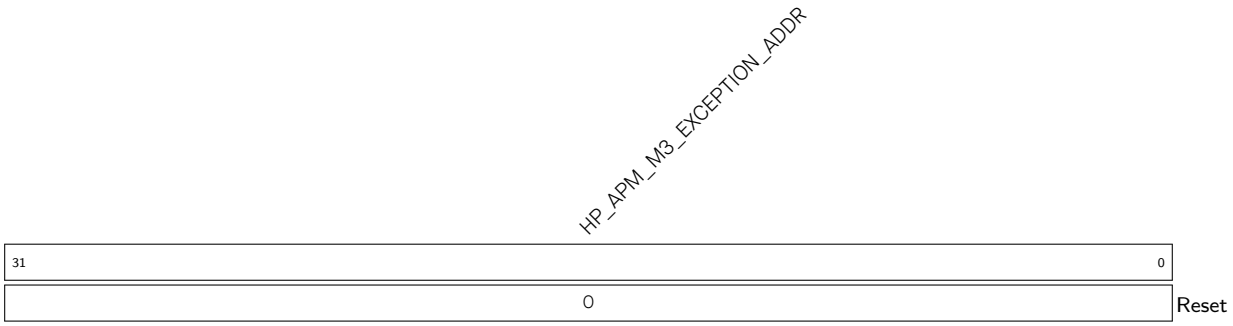



**HP_APM_M3_EXCEPTION_REGION** Represents the region where an exception occurs. (RO)

**HP_APM_M3_EXCEPTION_MODE** Represents the master's security mode when an exception occurs. (RO)

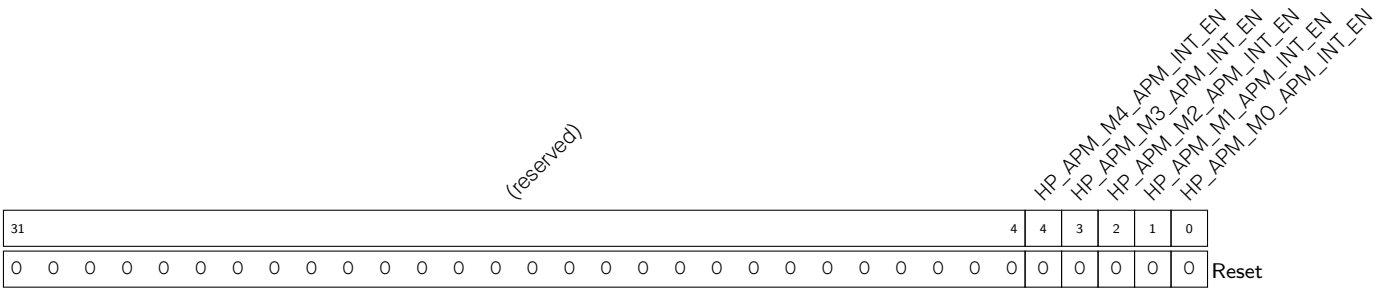
**HP_APM_M3_EXCEPTION_ID** Represents master ID when an exception occurs. (RO)

Register 17.21. HP_APM_M3_EXCEPTION_INFO1_REG (0x0104)




**HP_APM_M3_EXCEPTION_ADDR** Represents the access address when an exception occurs. (RO)

Register 17.22. HP_APM_INT_EN_REG (0x0118)




**HP_APM_M0_APM_INT_EN** Configures to enable HP_APM_CTRL M0 interrupt.

0: Disable

1: Enable

(R/W)

**HP_APM_M1_APM_INT_EN** Configures to enable HP_APM_CTRL M1 interrupt.

0: Disable

1: Enable

(R/W)

**HP_APM_M2_APM_INT_EN** Configures to enable HP_APM_CTRL M2 interrupt.

0: Disable

1: Enable

(R/W)

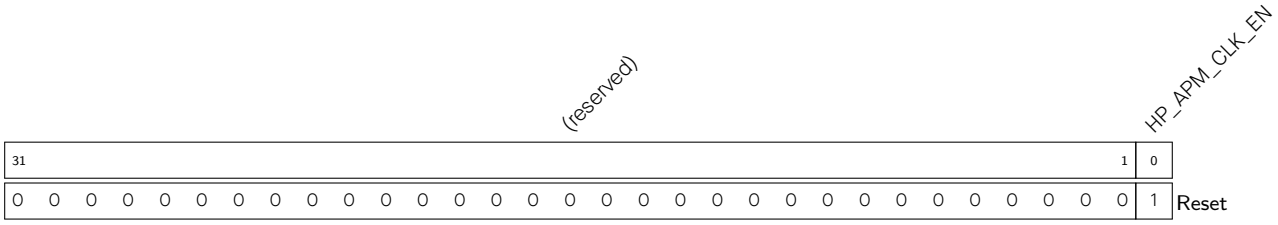
**HP_APM_M3_APM_INT_EN** Configures to enable HP_APM_CTRL M3 interrupt.

0: Disable

1: Enable

(R/W)

Register 17.23. HP_APM_CLOCK_GATE_REG (0x07F8)



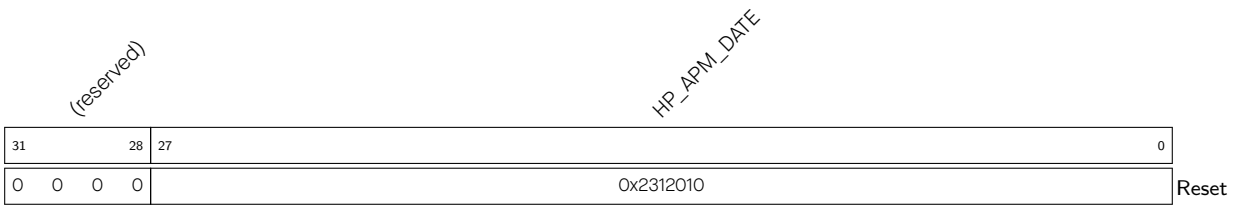

**HP_APM_CLK_EN** Configures whether to keep the clock always on.

0: Enable automatic clock gating

1: Keep the clock always on

(R/W)

Register 17.24. HP_APM_DATE_REG (0x07FC)



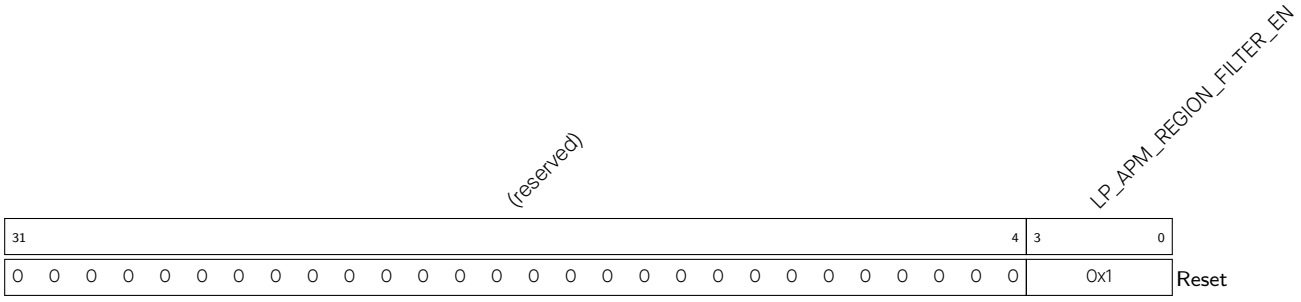

**HP_APM_DATE** Version control register. (R/W)

## 17.9.2 LP_APM_REG

The addresses in this section are relative to the LP_APM base address provided in Table 5.3-2 in Chapter 5 *System and Memory*.

For how to program reserved fields, please refer to Section *Programming Reserved Register Field*.

Register 17.25. LP_APM_REGION_FILTER_EN_REG (0x0000)



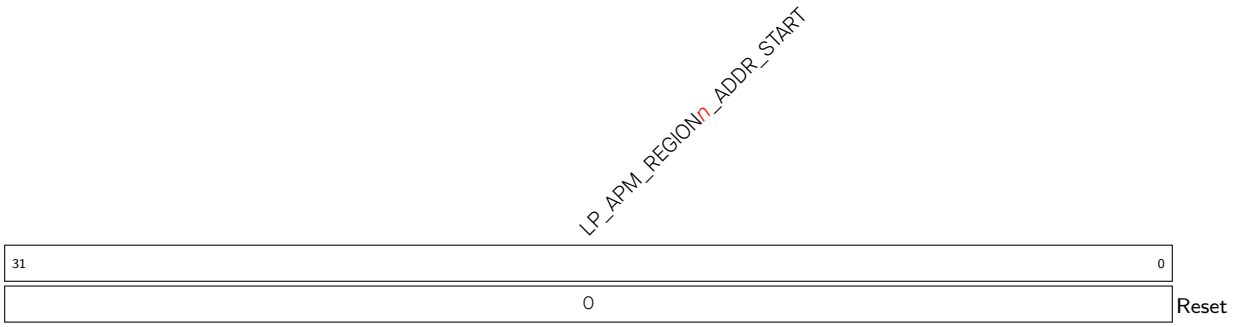

**LP_APM_REGION_FILTER_EN** Configure bit *n* (0-3) to enable permission checks for region *n* (0-3).

0: Disable

1: Enable

(R/W)

Register 17.26. LP_APM_REGION*n*_ADDR_START_REG (*n*: 0-3) (0x0004+0xC**n*)



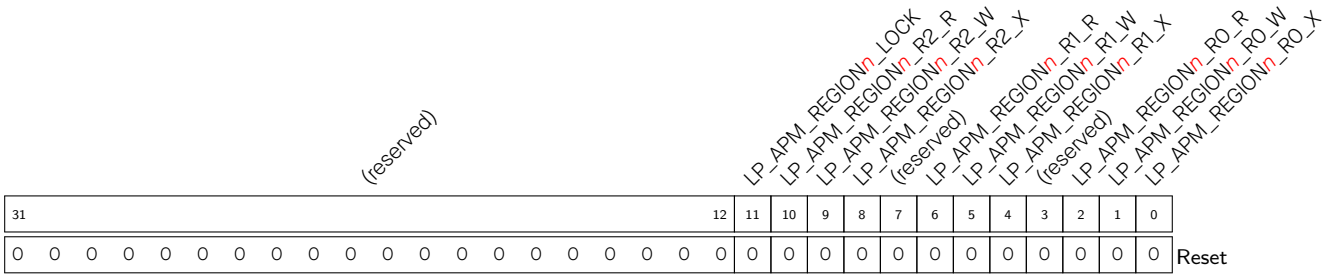

**LP_APM_REGION*n*_ADDR_START** Configures the start address of region *n*. (R/W)

**Register 17.27. LP_APM_REGION*n*_ADDR_END_REG (*n*: 0-3) (0x0008+0xC**n*)**




**LP_APM_REGION*n*_ADDR_END** Configures the end address of region *n*. (R/W)

**Register 17.28. LP_APM_REGION*n*_ATTR_REG (*n*: 0-3) (0x000C+0xC**n*)**




**LP_APM_REGION*n*_R0_X** Configures the execution permission in region *n* in REE0 mode. (R/W)

**LP_APM_REGION*n*_R0_W** Configures the write permission in region *n* in REE0 mode. (R/W)

**LP_APM_REGION*n*_R0_R** Configures the read permission in region *n* in REE0 mode. (R/W)

**LP_APM_REGION*n*_R1_X** Configures the execution permission in region *n* in REE1 mode. (R/W)

**LP_APM_REGION*n*_R1_W** Configures the write permission in region *n* in REE1 mode. (R/W)

**LP_APM_REGION*n*_R1_R** Configures the read permission in region *n* in REE1 mode. (R/W)

**LP_APM_REGION*n*_R2_X** Configures the execution permission in region *n* in REE1 mode. (R/W)

**LP_APM_REGION*n*_R2_W** Configures the write permission in region *n* in REE2 mode. (R/W)

**LP_APM_REGION*n*_R2_R** Configures the read permission in region *n* in REE2 mode. (R/W)

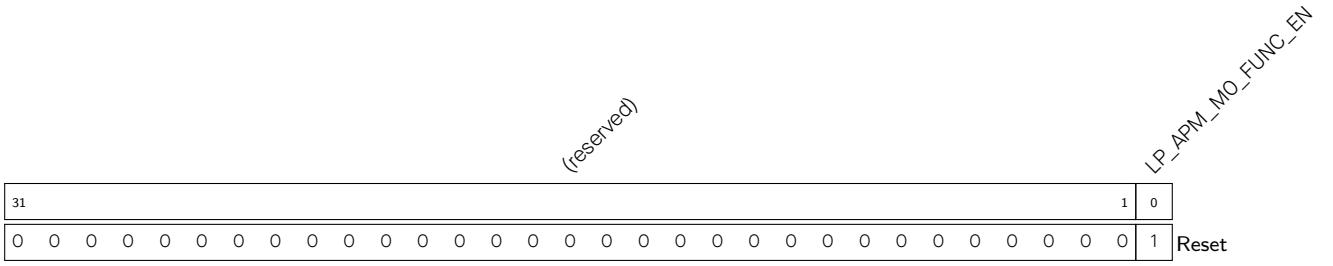
**LP_APM_REGION*n*_LOCK** Configures to lock the value of region *n* configuration registers (LP_APM_REGION*n*_ADDR_START_REG, LP_APM_REGION*n*_ADDR_END_REG and LP_APM_REGION*n*_ATTR_REG).

0: Do not lock

1: Lock

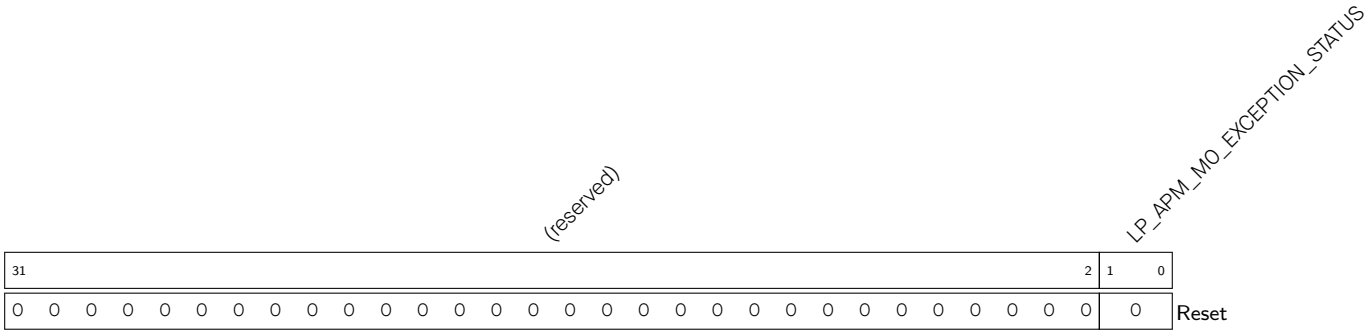
(R/W)

Register 17.29. LP_APM_FUNC_CTRL_REG (0x00C4)




**LP_APM_M0_FUNC_EN** Configures to enable permission management for LP_APM_CTRL M0. (R/W)

Register 17.30. LP_APM_M0_STATUS_REG (0x00C8)



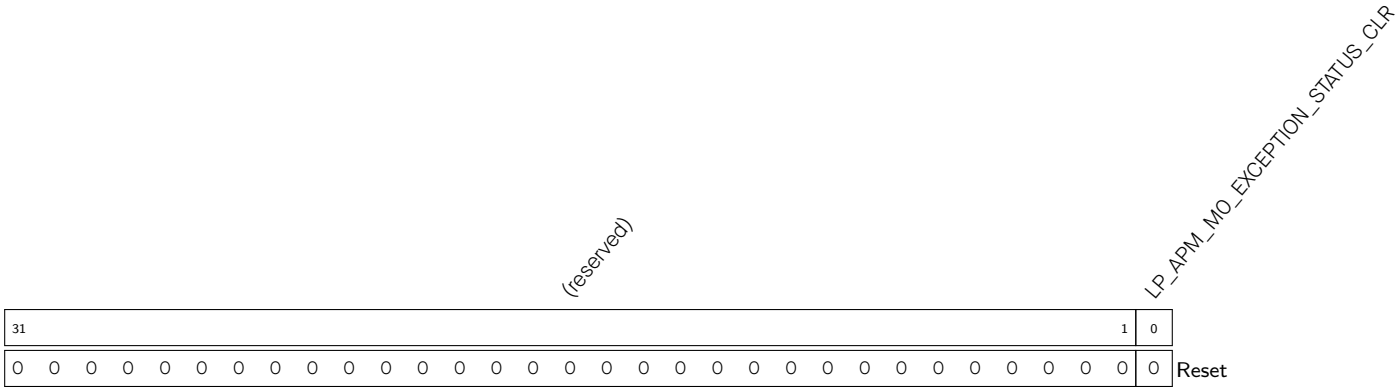

**LP_APM_M0_EXCEPTION_STATUS** Represents exception status.

bit0: 1 represents permission restrictions

bit1: 1 represents address out of bounds

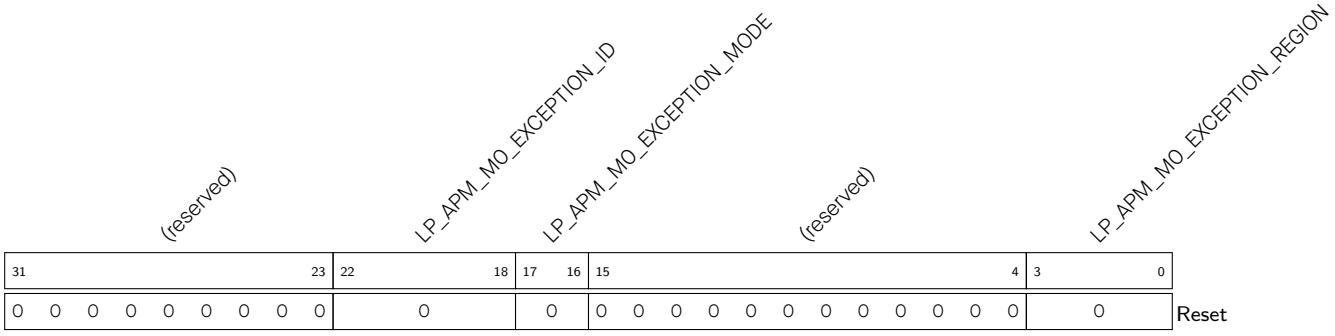
(RO)

Register 17.31. LP_APM_M0_STATUS_CLR_REG (0x00CC)




**LP_APM_M0_EXCEPTION_STATUS_CLR** Configures to clear exception status. (WT)

Register 17.32. LP_APM_M0_EXCEPTION_INFO0_REG (0x00D0)

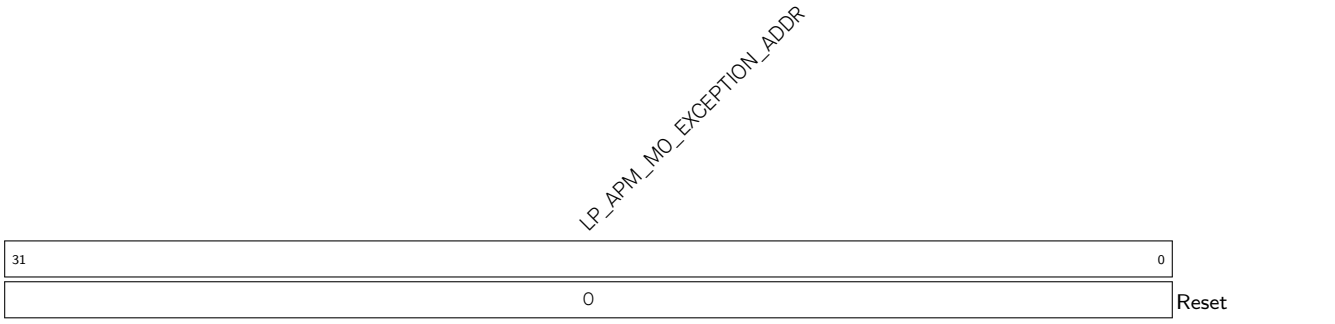



**LP_APM_M0_EXCEPTION_REGION** Represents the region where an exception occurs. (RO)

**LP_APM_M0_EXCEPTION_MODE** Represents the master's security mode when an exception occurs. (RO)

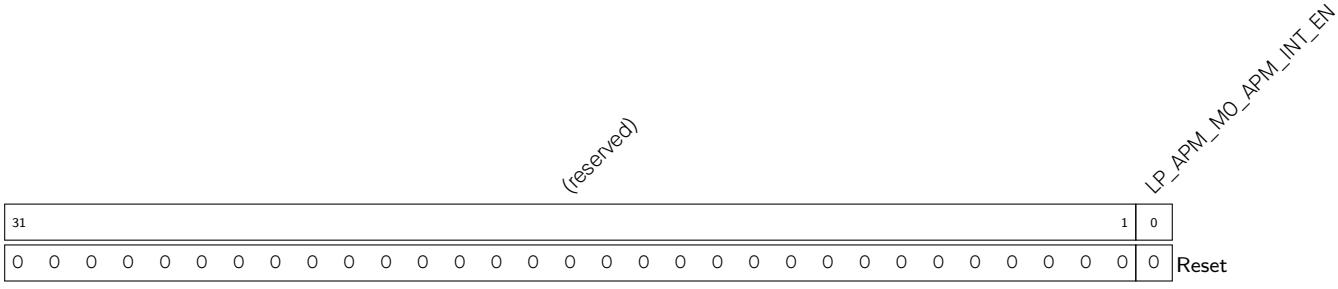
**LP_APM_M0_EXCEPTION_ID** Represents master ID when an exception occurs. (RO)

Register 17.33. LP_APM_M0_EXCEPTION_INFO1_REG (0x00D4)




**LP_APM_M0_EXCEPTION_ADDR** Represents the access address when an exception occurs. (RO)

Register 17.34. LP_APM_INT_EN_REG (0x00E8)



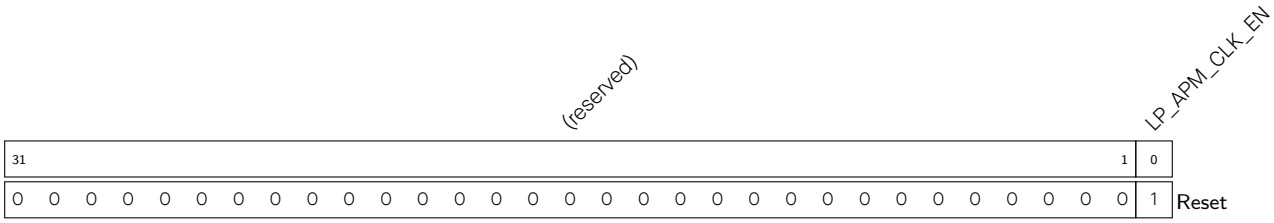

**LP_APM_M0_APM_INT_EN** Configures to enable LP_APM_CTRL M0 interrupt.

0: Disable

1: Enable

(R/W)

Register 17.35. LP_APM_CLOCK_GATE_REG (0x00EC)



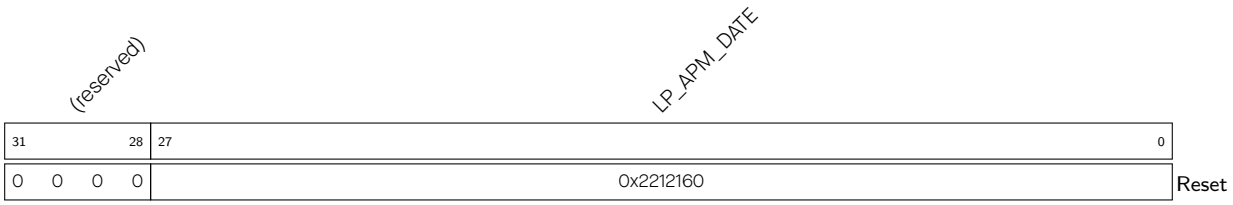

**LP_APM_CLK_EN** Configures whether to keep the clock always on.

0: Enable automatic clock gating

1: Keep the clock always on

(R/W)

Register 17.36. LP_APM_DATE_REG (0x00FC)



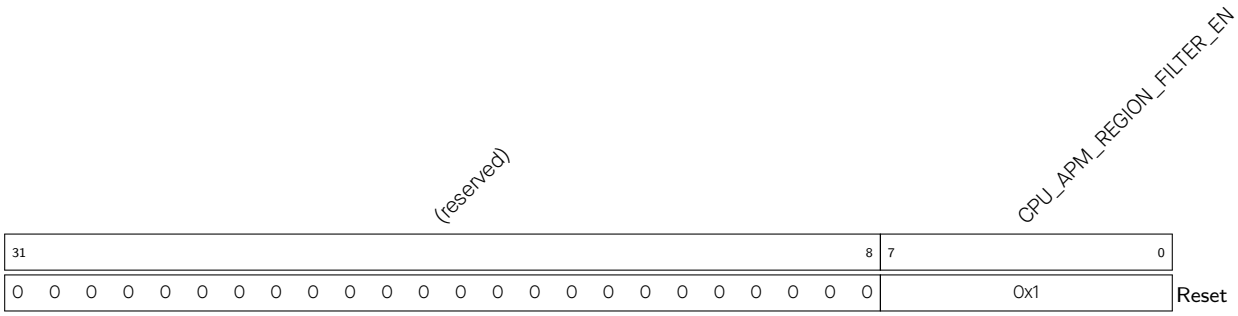

**LP_APM_DATE** Version control register. (R/W)

## 17.9.3 CPU_APM_REG

The addresses in this section are relative to the CPU_APM base address provided in Table 5.3-2 in Chapter 5 *System and Memory*.

For how to program reserved fields, please refer to Section *Programming Reserved Register Field*.

Register 17.37. CPU_APM_REGION_FILTER_EN_REG (0x0000)




**CPU_APM_REGION_FILTER_EN** Configures bit *n* (0-7) to enable permission checks for region *n* (0-7).

0: Disable

1: Enable

(R/W)

Register 17.38. CPU_APM_REGION*n*_ADDR_START_REG (*n*: 0-7) (0x0004+0xC**n*)

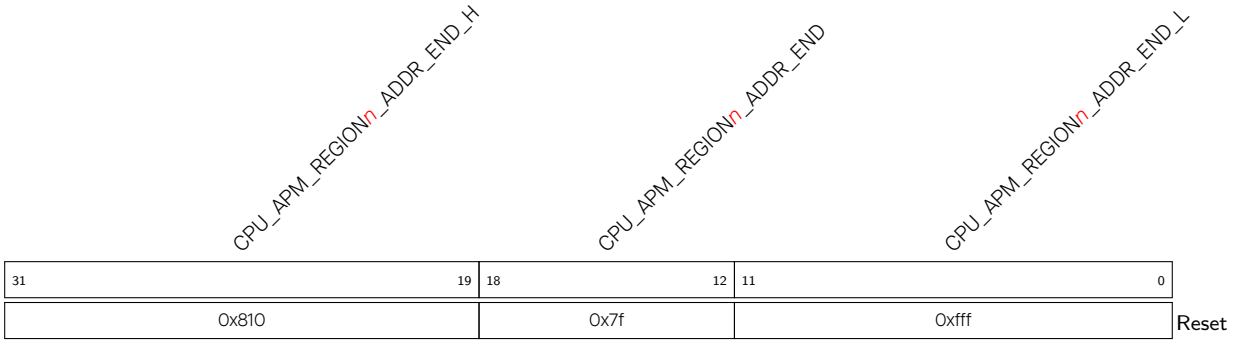



**CPU_APM_REGION*n*_ADDR_START_L** Indicates the lower 12 bits of the start address of region *n*. (HRO)

**CPU_APM_REGION*n*_ADDR_START** Configures the start address of region *n*. (R/W)

**CPU_APM_REGION*n*_ADDR_START_H** Indicates the higher 13 bits of the start address of region *n*. (HRO)

Register 17.39. CPU_APM_REGION*n*_ADDR_END_REG (*n*: 0-7) (0x0008+0xC**n*)

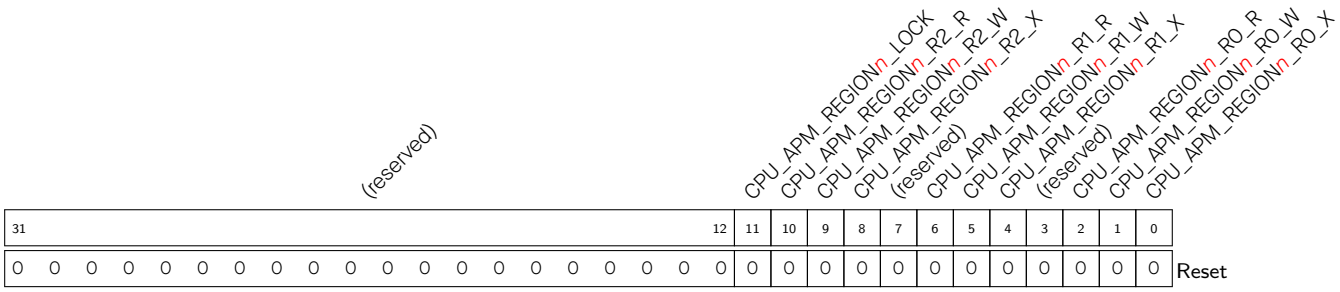



**CPU_APM_REGION*n*_ADDR_END_L** Indicates the lower 12 bits of the end address of region *n*. (HRO)

**CPU_APM_REGION*n*_ADDR_END** Configures the end address of region *n*. (R/W)

**CPU_APM_REGION*n*_ADDR_END_H** Indicates the higher 13 bits of the end address of region *n*. (HRO)

**Register 17.40. CPU_APM_REGIONn_ATTR_REG (n: 0-7) (0x000C+0xC*n)**




**CPU_APM_REGIONn_R0_X** Configures the execution permission in region *n* in REE0 mode. (R/W)

**CPU_APM_REGIONn_R0_W** Configures the write permission in region *n* in REE0 mode. (R/W)

**CPU_APM_REGIONn_R0_R** Configures the read permission in region *n* in REE0 mode. (R/W)

**CPU_APM_REGIONn_R1_X** Configures the execution permission in region *n* in REE1 mode. (R/W)

**CPU_APM_REGIONn_R1_W** Configures the write permission in region *n* in REE1 mode. (R/W)

**CPU_APM_REGIONn_R1_R** Configures the read permission in region *n* in REE1 mode. (R/W)

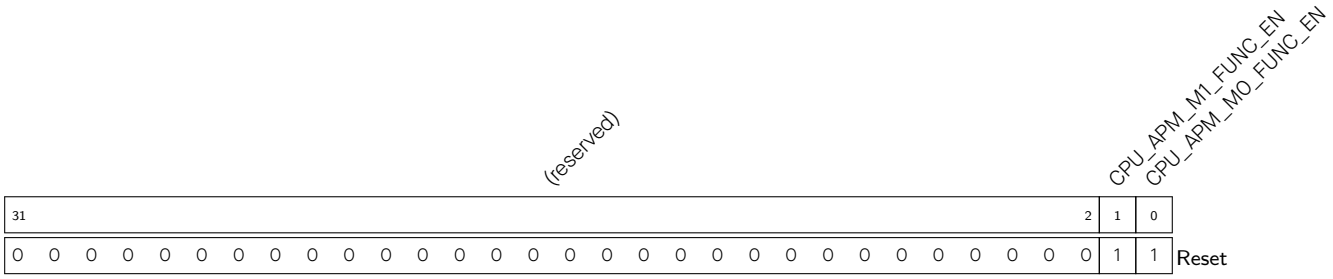
**CPU_APM_REGIONn_R2_X** Configures the execution permission in region *n* in REE2 mode. (R/W)

**CPU_APM_REGIONn_R2_W** Configures the write permission in region *n* in REE2 mode. (R/W)

**CPU_APM_REGIONn_R2_R** Configures the read permission in region *n* in REE2 mode. (R/W)

**CPU_APM_REGIONn_LOCK** Configures to lock the value of region *n*'s configuration registers (CPU_APM_REGIONn_ADDR_START_REG, CPU_APM_REGIONn_ADDR_END_REG, and CPU_APM_REGIONn_ATTR_REG).
0: Do not lock
1: Lock
(R/W)

Register 17.41. CPU_APM_FUNC_CTRL_REG (0x00C4)




**CPU_APM_M0_FUNC_EN** Configures whether to enable permission management for CPU_APM_CTRL M0.

0: Disable

1: Enable

(R/W)

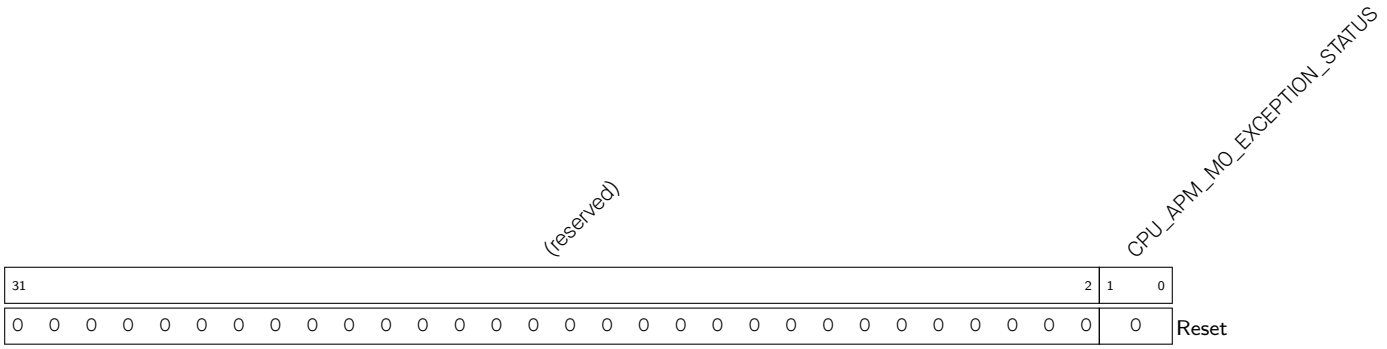
**CPU_APM_M1_FUNC_EN** Configures whether to enable permission management for CPU_APM_CTRL M1.

0: Disable

1: Enable

(R/W)

Register 17.42. CPU_APM_M0_STATUS_REG (0x00C8)



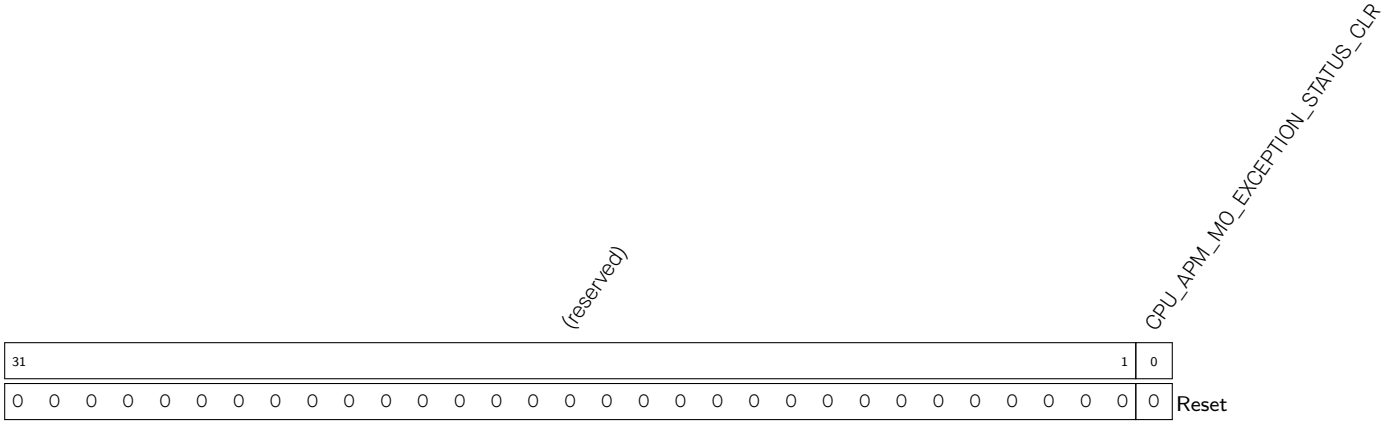

**CPU_APM_M0_EXCEPTION_STATUS** Represents exception status.

bit0: 1 represents permission restrictions

bit1: 1 represents address out of bounds

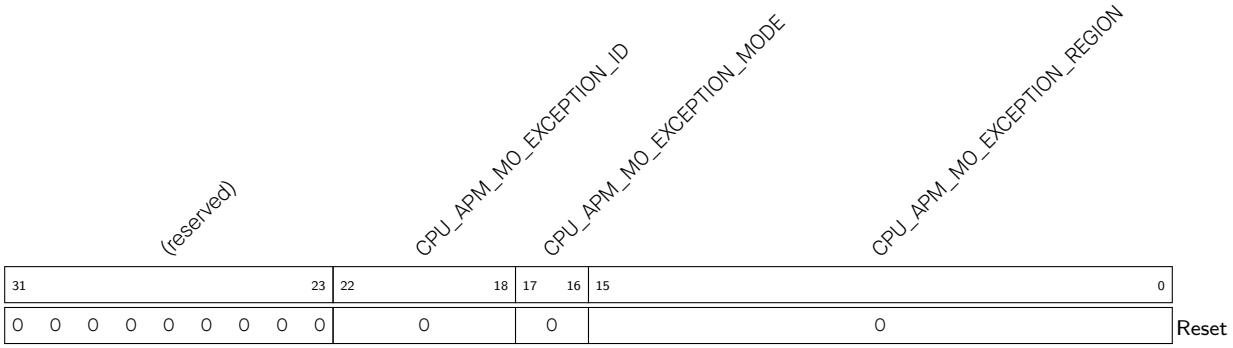
(RO)

Register 17.43. CPU_APM_M0_STATUS_CLR_REG (0x00CC)




**CPU_APM_M0_EXCEPTION_STATUS_CLR** Write 1 to clear exception status. (WT)

Register 17.44. CPU_APM_M0_EXCEPTION_INFO0_REG (0x00D0)

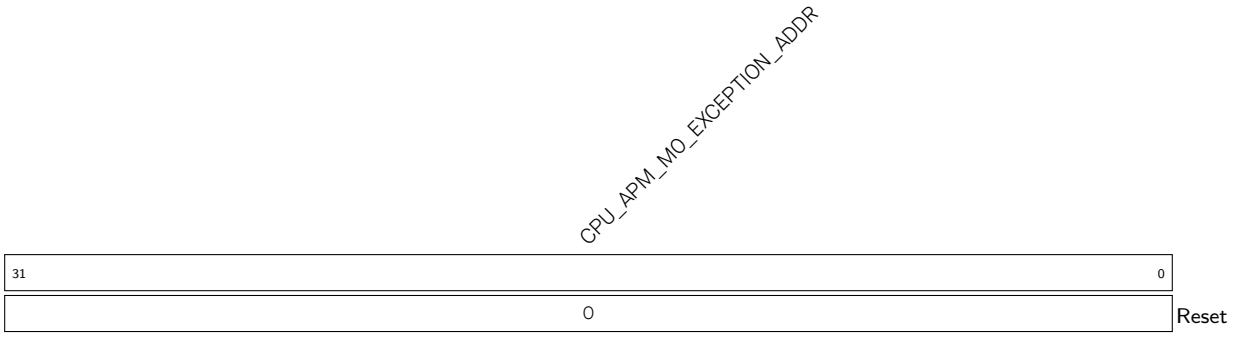



**CPU_APM_M0_EXCEPTION_REGION** Represents the region where an exception occurs. (RO)

**CPU_APM_M0_EXCEPTION_MODE** Represents the master's security mode when an exception occurs. (RO)

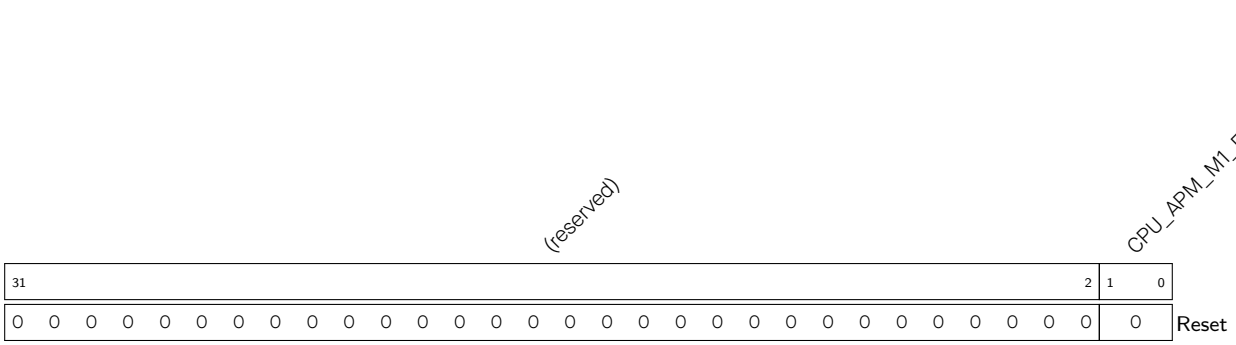
**CPU_APM_M0_EXCEPTION_ID** Represents master ID when an exception occurs. (RO)

Register 17.45. CPU_APM_M0_EXCEPTION_INFO1_REG (0x00D4)




**CPU_APM_M0_EXCEPTION_ADDR** Represents the access address when an exception occurs. (RO)

Register 17.46. CPU_APM_M1_STATUS_REG (0x00D8)



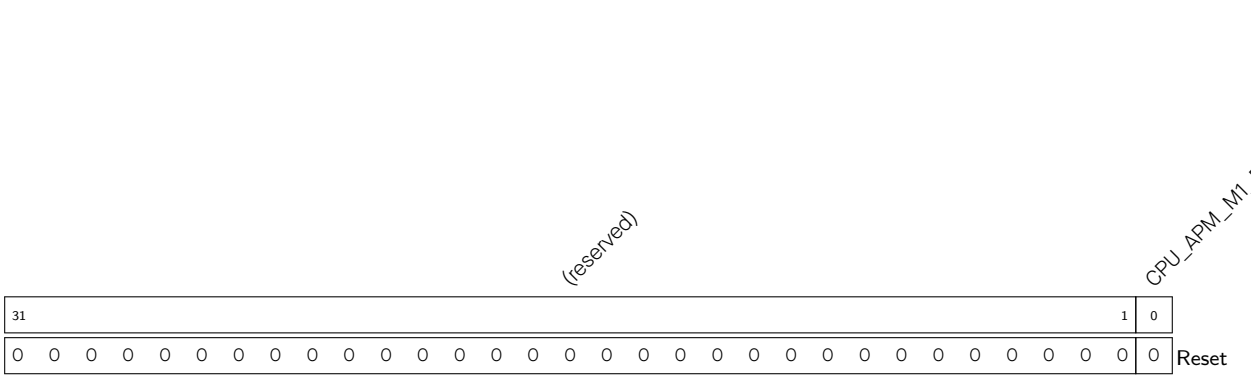

**CPU_APM_M1_EXCEPTION_STATUS** Represents exception status.

bit0: 1 represents permission restrictions

bit1: 1 represents address out of bounds

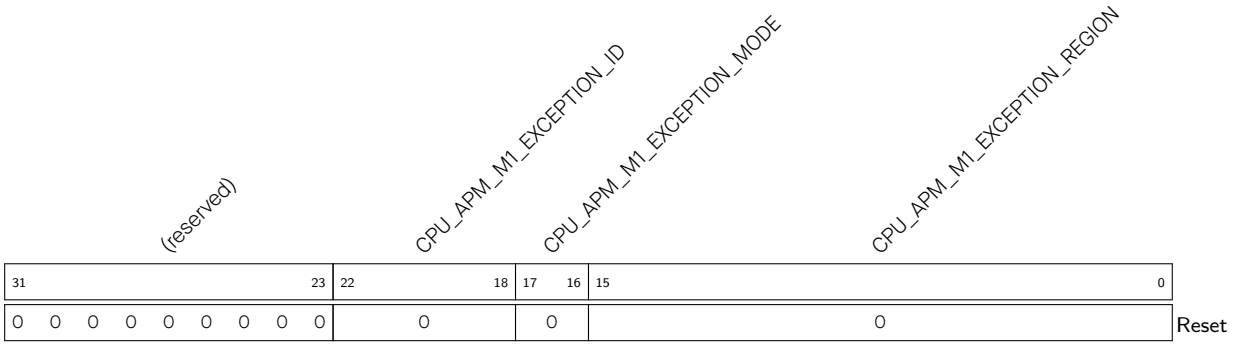
(RO)

Register 17.47. CPU_APM_M1_STATUS_CLR_REG (0x00DC)




**CPU_APM_M1_EXCEPTION_STATUS_CLR** Write 1 to clear exception status. (WT)

Register 17.48. CPU_APM_M1_EXCEPTION_INFO0_REG (0x00E0)

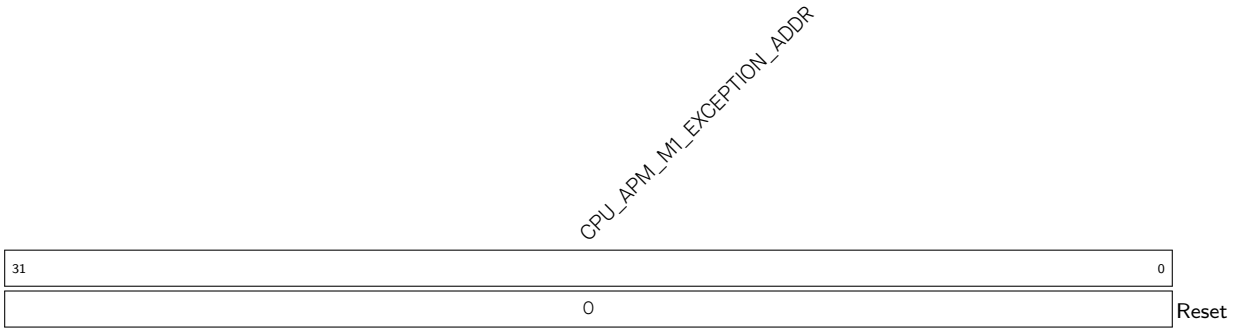



**CPU_APM_M1_EXCEPTION_REGION** Represents the region where an exception occurs. (RO)

**CPU_APM_M1_EXCEPTION_MODE** Represents the master's security mode when an exception occurs. (RO)

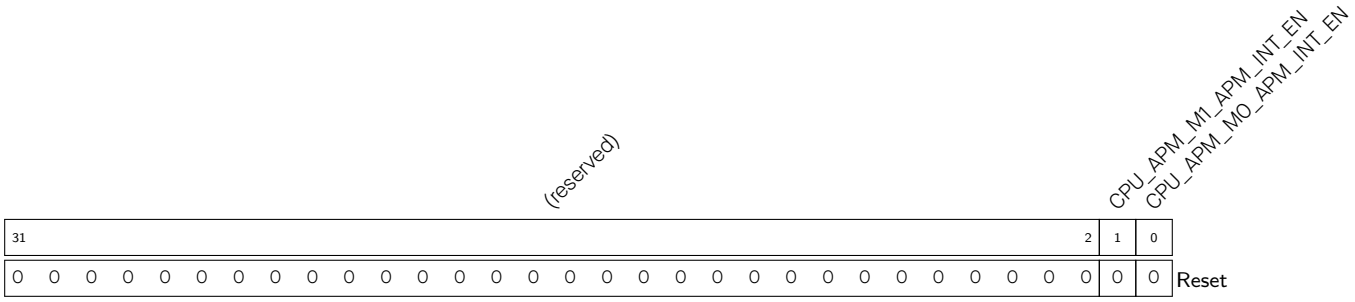
**CPU_APM_M1_EXCEPTION_ID** Represents master ID when an exception occurs. (RO)

Register 17.49. CPU_APM_M1_EXCEPTION_INFO1_REG (0x00E4)




**CPU_APM_M1_EXCEPTION_ADDR** Represents the access address when an exception occurs. (RO)

Register 17.50. CPU_APM_INT_EN_REG (0x0118)




**CPU_APM_M0_APM_INT_EN** Configures whether to enable CPU_APM_CTRL M0 interrupt.

0: Disable

1: Enable

(R/W)

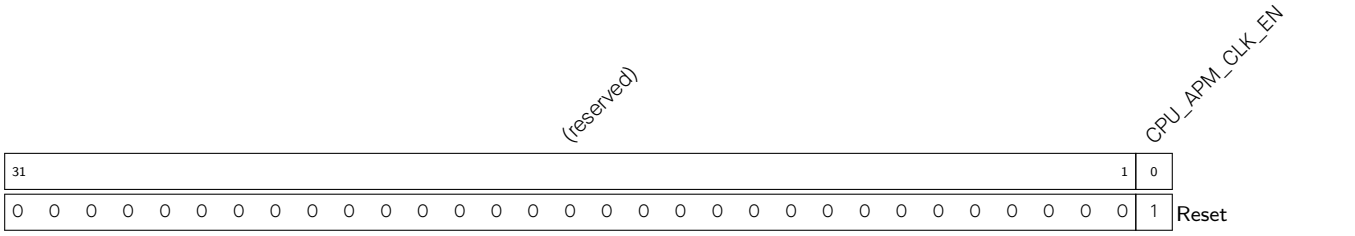
**CPU_APM_M1_APM_INT_EN** Configures whether to enable CPU_APM_CTRL M1 interrupt.

0: Disable

1: Enable

(R/W)

Register 17.51. CPU_APM_CLOCK_GATE_REG (0x07F8)



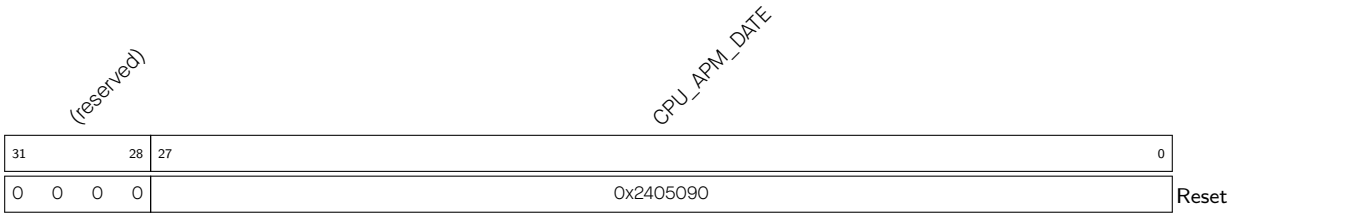

**CPU_APM_CLK_EN** Configures whether to keep the clock always on.

0: Enable automatic clock gating

1: Keep the clock always on

(R/W)

Register 17.52. CPU_APM_DATE_REG (0x07FC)



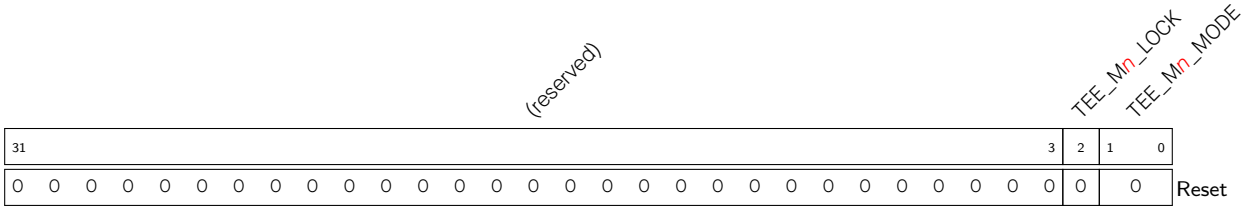

**CPU_APM_DATE** Version control register. (R/W)

### 17.9.4 HP_TEE_REG

The addresses in this section are relative to the TEE base address provided in Table 5.3-2 in Chapter 5 *System and Memory*.

For how to program reserved fields, please refer to Section *Programming Reserved Register Field*.

**Register 17.53. TEE_M*n*_MODE_CTRL_REG (*n*: 0-31) (0x0000+0x4**n*)**




**TEE_M*n*_MODE** Configures the security mode for master *n*.

0: TEE

1: REE0

2: REE1

3: REE2

(R/W)

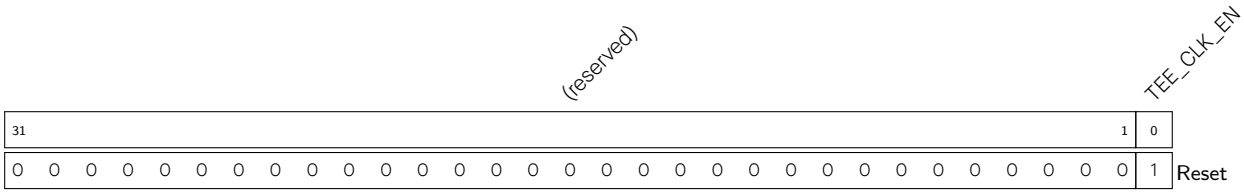
**TEE_M*n*_LOCK** Configures to lock the configuration of master *n*'s security mode.

0: Do not lock

1: Lock

(R/W)

**Register 17.54. TEE_CLOCK_GATE_REG (0x0080)**



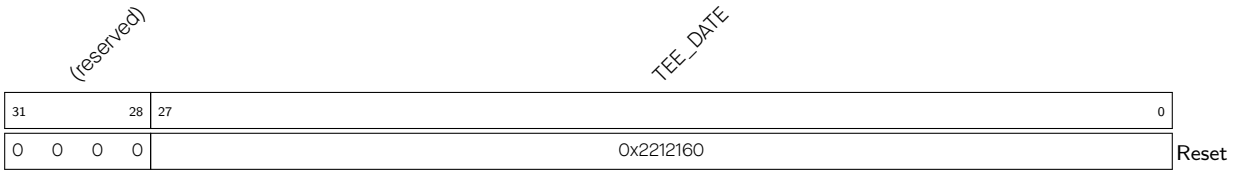

**TEE_CLK_EN** Configures whether to keep the clock always on.

0: Enable automatic clock gating

1: Keep the clock always on

(R/W)

**Register 17.55. TEE_DATE_REG (0x0FFC)**




**TEE_DATE** Version control register. (R/W)

# Chapter 18

# System Registers

## 18.1 Overview

ESP32-C61 supports a set of auxiliary chip features listed in subsection 18.2 *Features* below, which are configured via registers. This chapter describes the registers used to configure these features.

## 18.2 Features

ESP32-C61 system registers control the following peripheral blocks and core modules:

- External Memory Encryption/Decryption
- Anti-DPA attack security
- Bus timeout protection

## 18.3 Function Description

### 18.3.1 External Memory Encryption/Decryption Configuration

Register HP_SYSTEM_EXTERNAL_DEVICE_ENCRYPT_DECRYPT_CONTROL_REG configures encryption and decryption options of the external memory.

For details, please refer to Chapter 24 *External Memory Encryption and Decryption (XTS_AES)*.

### 18.3.2 Anti-DPA Attack Security Control

ESP32-C61 has a dual protection mechanism against Differential Power Analysis (DPA) attacks at the hardware level.

- First, a mask mechanism is introduced in the symmetric encryption operation, which interferes with the power consumption trajectory by masking the real data in the operation process. This security mechanism cannot be turned off.
- Second, the clock selected for the operation will change dynamically in real time, blurring the power consumption trajectory during the operation. For this security mechanism, ESP32-C61 provides 4 security levels for users to choose to adapt to different applications.

Table 18.3-1. Security Level

| Security-Level Name | Security-Level Value | PLL_CLK (MHz) | XTAL_CLK (MHz) |
|---|---|---|---|
| SEC_DPA_OFF | 0 | 160 | 40 |
| SEC_DPA_LOW | 1 | (120,160][A] | (20,40][A] |
| SEC_DPA_MIDDLE | 2 | (96,160][A] | (13,40][A] |
| SEC_DPA_HIGH | 3 | (80,160][A] | (8,40][A] |

[A] (x,y] means the operating frequency is greater than x MHz, and equal to or less than y MHz.

The field HP_SYSTEM_SEC_DPA_CFG_SEL in register HP_SYSTEM_SEC_DPA_CONF_REG can be:

- 0 (default): The security-level is decided by the eFuse field EFUSE_SEC_DPA_LEVEL.
- 1: The security-level is decided by HP_SYSTEM_SEC_DPA_CFG_LEVEL in register HP_SYSTEM_SEC_DPA_CONF_REG.

### 18.3.3 Bus Timeout Protection

ESP32-C61 supports APB bus timeout protection and allows configurable timeout threshold. When a transfer is initiated, the internal counter of the timeout protection module increments by 1 on each clock cycle.

- If the accumulated value remains below the timeout threshold and a response is received from the slave device, the counter is cleared.
- If the accumulated value exceeds the threshold and no response has been received, the module forces the bus return signal high to terminate the transfer. At the same time, an interrupt is triggered, and the module logs the exception address and the master ID associated with the access.

#### 18.3.3.1 CPU Peripheral Timeout Protection Register

Register HP_SYSTEM_CPU_PERI_TIMEOUT_CONF_REG configures the timeout protection for accessing CPU peripherals, which refer to the peripherals or modules whose addresses are in the range of 0x600C_0000–0x600C_FFFF. For details, please refer to Subsection 5.3.5 *Modules/Peripherals Address Mapping* in Chapter 5 *System and Memory*.

When a timeout occurs, the CPU_PERI_TIMEOUT_INTR interrupt will be triggered. The related registers are:

- HP_SYSTEM_CPU_PERI_TIMEOUT_CONF_REG: Enables timeout protection and configures the timeout threshold.
- HP_SYSTEM_CPU_PERI_TIMEOUT_ADDR_REG: When a timeout occurs, this register will record the address of the timeout.
- HP_SYSTEM_CPU_PERI_TIMEOUT_UID_REG: When a timeout occurs, this register will record the master ID of the timeout.

#### 18.3.3.2 HP Peripheral Timeout Protection Register

Register HP_SYSTEM_HP_PERI_TIMEOUT_CONF_REG configures the timeout protection for accessing HP peripherals, which refer to the peripherals or modules whose addresses are in the range of

0x6000_0000–0x6009_FFFF. For details, please refer to Subsection 5.3.5 *Modules/Peripherals Address Mapping* in Chapter 5 *System and Memory*.

When a timeout occurs, the HP_PERI_TIMEOUT_INTR interrupt will be triggered. The related registers are:

- HP_SYSTEM_HP_PERI_TIMEOUT_CONF_REG: Enables timeout protection and configures the timeout threshold.
- HP_SYSTEM_HP_PERI_TIMEOUT_ADDR_REG: When a timeout occurs, this register will record the address of the timeout.
- HP_SYSTEM_HP_PERI_TIMEOUT_UID_REG: When a timeout occurs, this register will record the master ID of the timeout.

#### 18.3.3.3 LP Peripheral Timeout Protection Register

Register LP_PERI_BUS_TIMEOUT_CONF_REG configures the timeout protection for accessing LP peripherals, which refer to the peripherals or modules whose addresses are in the range of 0x600B_0000–0x600B_FFFF. For details, please refer to Subsection 5.3.5 *Modules/Peripherals Address Mapping* in Chapter 5 *System and Memory*.

When a timeout occurs, the LP_PERI_TIMEOUT_INTR interrupt will be triggered. The related registers are:

- LP_PERI_BUS_TIMEOUT_CONF_REG: Enables timeout protection and configures the timeout threshold.
- LP_PERI_BUS_TIMEOUT_ADDR_REG: When a timeout occurs, this register will record the address of the timeout.
- LP_PERI_BUS_TIMEOUT_UID_REG: When a timeout occurs, this register will record the master ID of the timeout.

## 18.4 Register Summary

In this section, addresses starting with HP_SYSTEM are relative to HP System Registers base address and addresses starting with LP_PERI are relative to Low-Power Peripherals base address provided in Table 5.3-2 in Chapter 5 *System and Memory*.

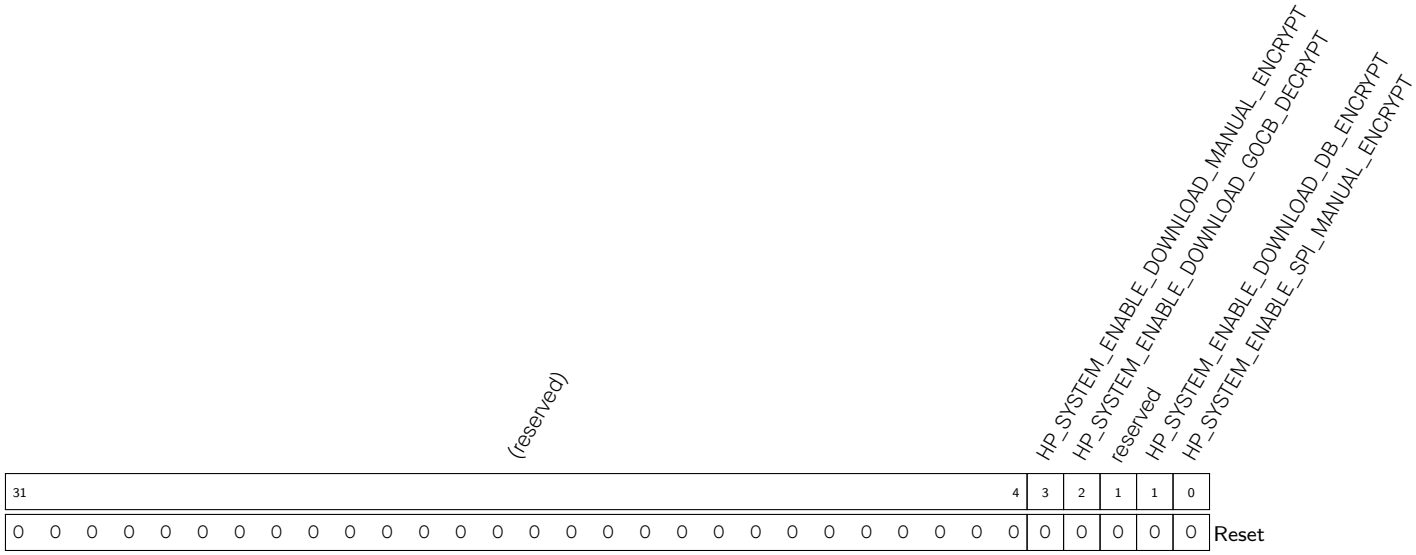
The abbreviations given in Column **Access** are explained in Section *Access Types for Registers*.

| Name | Description | Address | Access |
|---|---|---|---|
| **Configuration Register** | | | |
| HP_SYSTEM_EXTERNAL_DEVICE_ENCRYPT_-DECRYPT_CONTROL_REG | External device encryption/decryption configuration register | 0x0000 | R/W |
| HP_SYSTEM_SEC_DPA_CONF_REG | HP anti-DPA security configuration register | 0x0008 | R/W |
| **CPU Peripheral Timeout Register** | | | |
| HP_SYSTEM_CPU_PERI_TIMEOUT_CONF_REG | Timeout protection configuration register | 0x000C | varies |
| HP_SYSTEM_CPU_PERI_TIMEOUT_ADDR_REG | Abnormal access address register | 0x0010 | RO |
| HP_SYSTEM_CPU_PERI_TIMEOUT_UID_REG | Master ID and permission register | 0x0014 | WTC |
| **HP Peripheral Timeout Register** | | | |
| HP_SYSTEM_HP_PERI_TIMEOUT_CONF_REG | Timeout protection configuration register | 0x0018 | varies |
| HP_SYSTEM_HP_PERI_TIMEOUT_ADDR_REG | Abnormal access address register | 0x001C | RO |
| HP_SYSTEM_HP_PERI_TIMEOUT_UID_REG | Master ID and permission register | 0x0020 | WTC |
| **LP Peripheral Timeout Register** | | | |
| LP_PERI_BUS_TIMEOUT_CONF_REG | Timeout protection configuration register | 0x0010 | varies |
| LP_PERI_BUS_TIMEOUT_ADDR_REG | Abnormal access address register | 0x0014 | RO |
| LP_PERI_BUS_TIMEOUT_UID_REG | Master ID and permission register | 0x0018 | WTC |
| **Version Register** | | | |
| HP_SYSTEM_DATE_REG | Date control and version control register | 0x03FC | R/W |

## 18.5 Registers

In this section, addresses starting with HP_SYSTEM are relative to HP System Registers base address and addresses starting with LP_PERI are relative to Low-Power Peripherals base address provided in Table 5.3-2 in Chapter 5 *System and Memory*.

**Register 18.1. HP_SYSTEM_EXTERNAL_DEVICE_ENCRYPT_DECRYPT_CONTROL_REG (0x0000)**




**HP_SYSTEM_ENABLE_SPI_MANUAL_ENCRYPT** Configures whether to enable MSPI XTS manual encryption in SPI boot mode.

0: Disable

1: Enable

(R/W)

**HP_SYSTEM_ENABLE_DOWNLOAD_DB_ENCRYPT** Configures whether or not 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 MSPI XTS auto decryption in download boot mode.

0: Disable

1: Enable

(R/W)

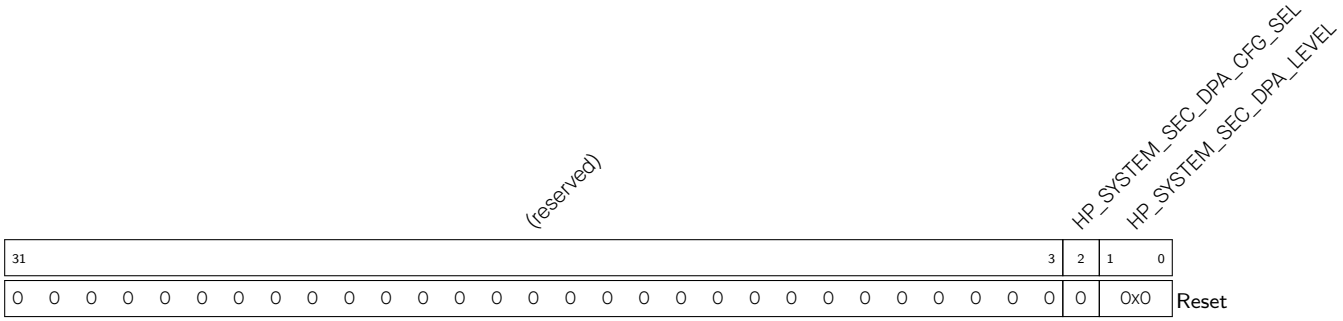
**HP_SYSTEM_ENABLE_DOWNLOAD_MANUAL_ENCRYPT** Configures whether to enable MSPI XTS manual encryption in download boot mode.

0: Disable

1: Enable

(R/W)

Register 18.2. HP_SYSTEM_SEC_DPA_CONF_REG (0x0008)




**HP_SYSTEM_SEC_DPA_LEVEL** Configures whether to enable anti-DPA attack. Valid only when HP_SYSTEM_SEC_DPA_CFG_SEL is 1.

0: Disable

1-3: Enable. The larger the number, the higher the security level, which represents the ability to resist DPA attacks, with increased computational overhead of the hardware crypto-accelerators at the same time.

(R/W)

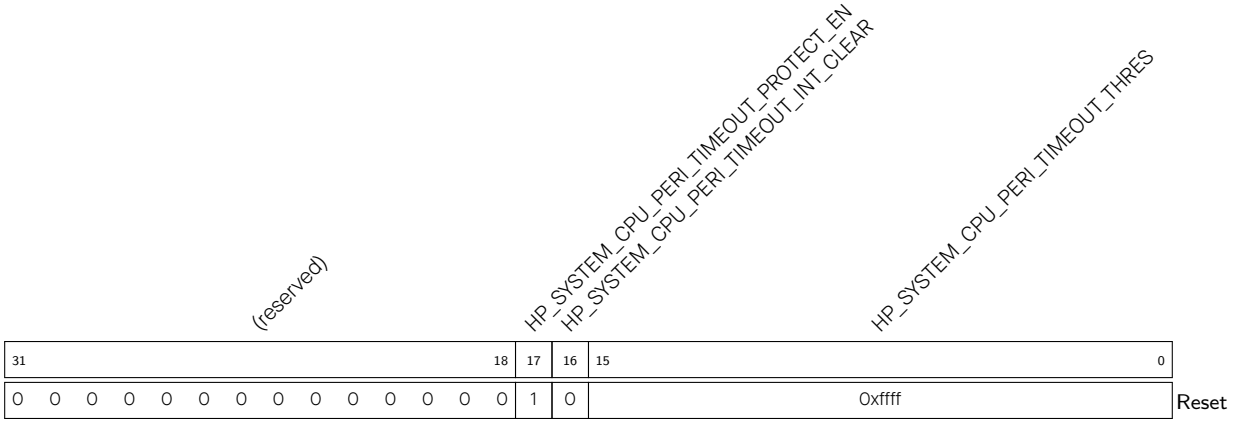
**HP_SYSTEM_SEC_DPA_CFG_SEL** Configures whether to select HP_SYSTEM_SEC_DPA_LEVEL or EFUSE_SEC_DPA_LEVEL (from eFuse) to control DPA level.

0: Select EFUSE_SEC_DPA_LEVEL

1: Select HP_SYSTEM_SEC_DPA_LEVEL

(R/W)

Register 18.3. HP_SYSTEM_CPU_PERI_TIMEOUT_CONF_REG (0x000C)




**HP_SYSTEM_CPU_PERI_TIMEOUT_THRES** Configures the timeout threshold for bus access for accessing CPU peripheral register in the number of clock cycles of the clock domain. (R/W)

**HP_SYSTEM_CPU_PERI_TIMEOUT_INT_CLEAR** Write 1 to clear the timeout interrupt. (WT)

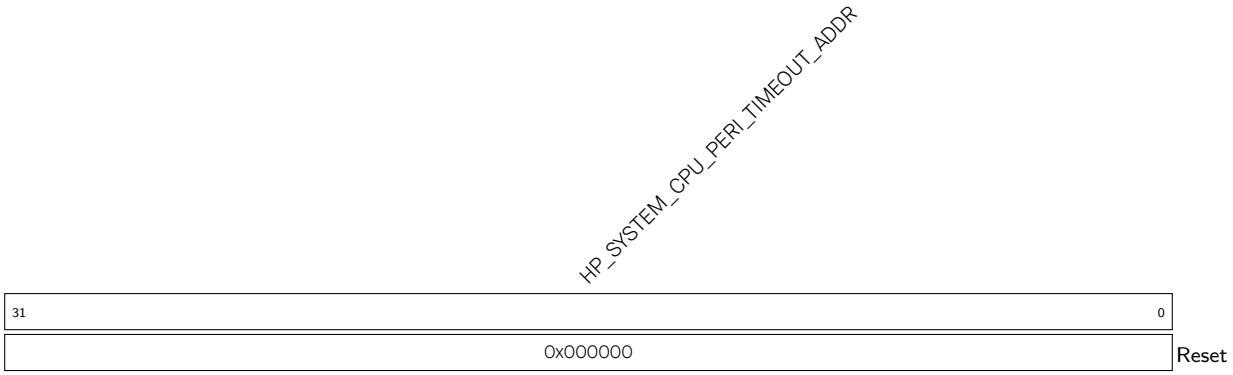
**HP_SYSTEM_CPU_PERI_TIMEOUT_PROTECT_EN** Configures whether to enable timeout protection for accessing CPU peripheral registers.

0: Disable

1: Enable

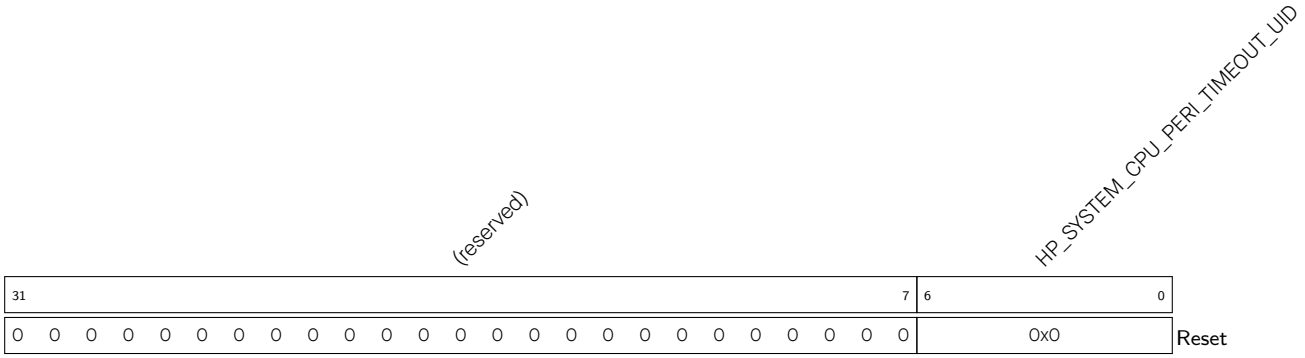
(R/W)

Register 18.4. HP_SYSTEM_CPU_PERI_TIMEOUT_ADDR_REG (0x0010)



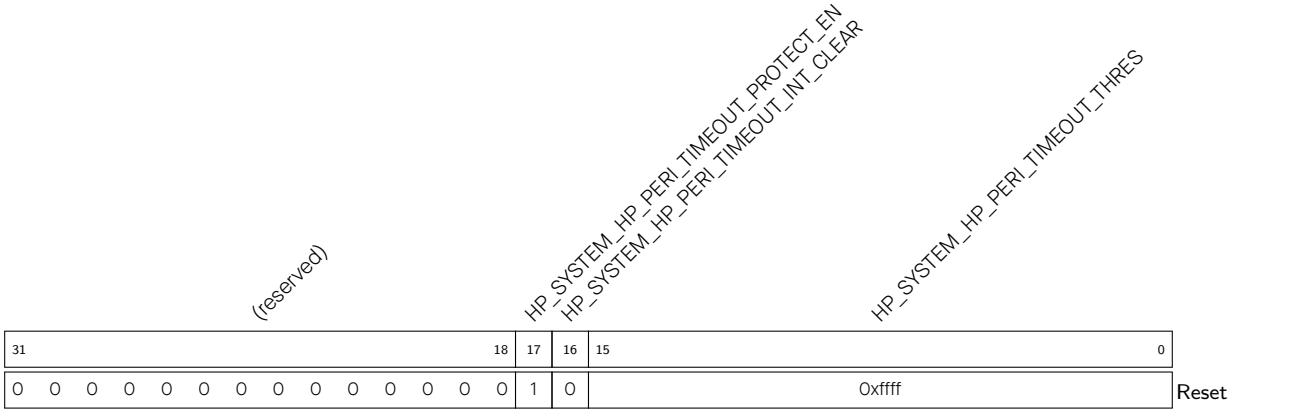

**HP_SYSTEM_CPU_PERI_TIMEOUT_ADDR** Represents the address information of abnormal access. (RO)

Register 18.5. HP_SYSTEM_CPU_PERI_TIMEOUT_UID_REG (0x0014)




**HP_SYSTEM_CPU_PERI_TIMEOUT_UID** Represents the master ID[4:0] and master permission[6:5] when trigger timeout. For details, see Chapter 17 *Permission Control (PMS)*. This register will be cleared after the interrupt is cleared. (WTC)

Register 18.6. HP_SYSTEM_HP_PERI_TIMEOUT_CONF_REG (0x0018)




**HP_SYSTEM_HP_PERI_TIMEOUT_THRES** Configures the timeout threshold for bus access for accessing HP peripheral registers, corresponding to the number of clock cycles of the clock domain. (R/W)

**HP_SYSTEM_HP_PERI_TIMEOUT_INT_CLEAR** Write 1 to clear timeout interrupt.(WT)

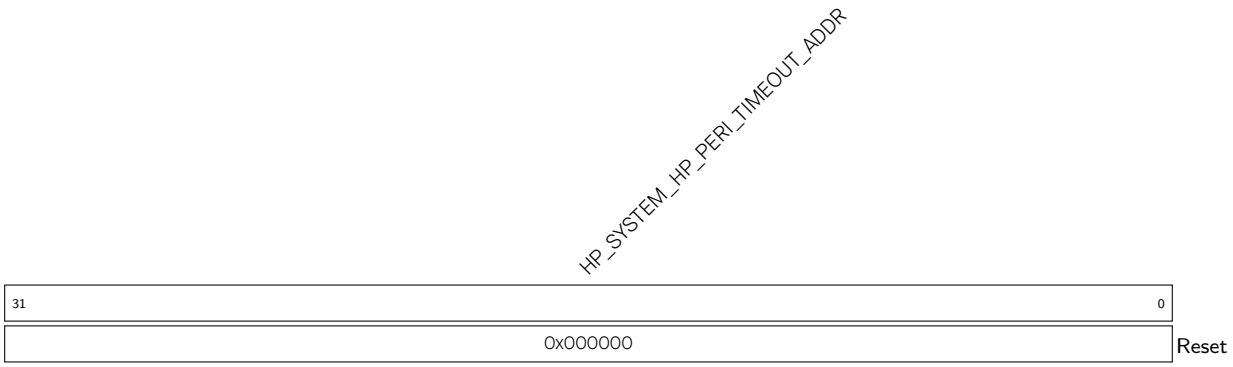
**HP_SYSTEM_HP_PERI_TIMEOUT_PROTECT_EN** Configures whether to enable timeout protection for accessing HP peripheral registers.

0: Disable

1: Enable

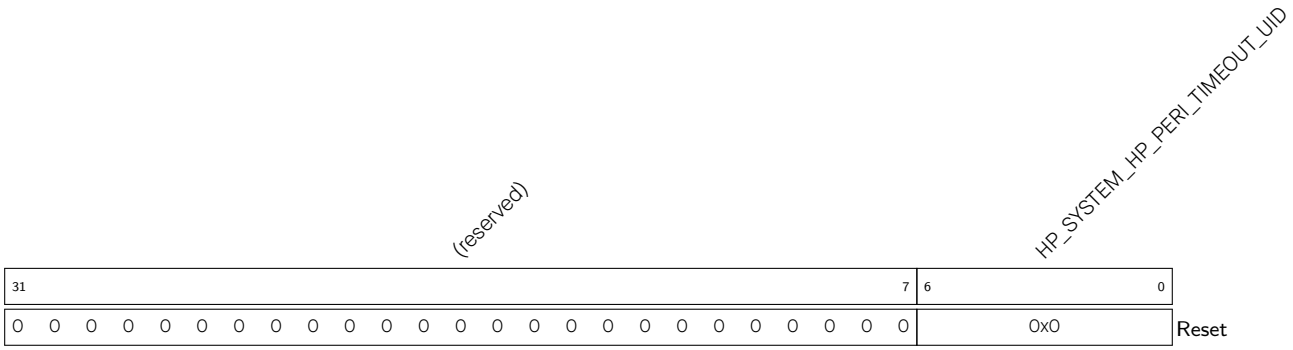
(R/W)

Register 18.7. HP_SYSTEM_HP_PERI_TIMEOUT_ADDR_REG (0x001C)



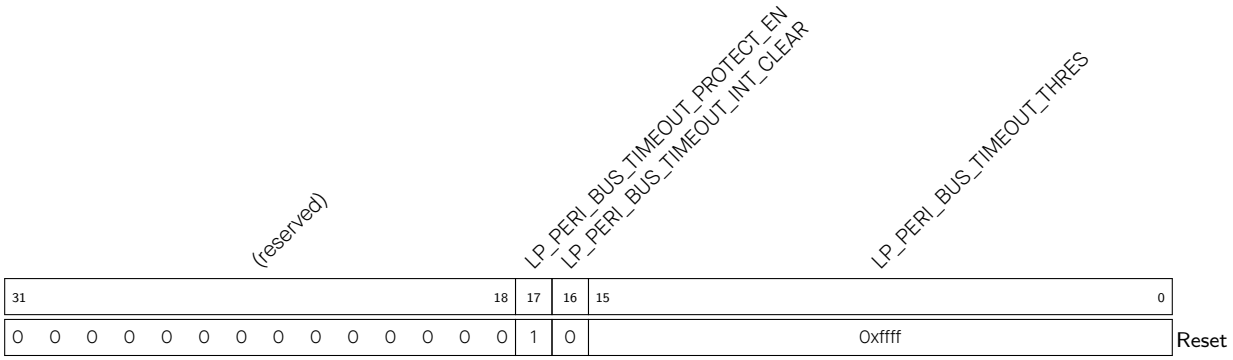

**HP_SYSTEM_HP_PERI_TIMEOUT_ADDR** Represents the address information of abnormal access. (RO)

Register 18.8. HP_SYSTEM_HP_PERI_TIMEOUT_UID_REG (0x0020)




**HP_SYSTEM_HP_PERI_TIMEOUT_UID** Represents the master ID[4:0] and master permission[6:5] when trigger timeout. For details, see Chapter 17 *Permission Control (PMS)*. This register will be cleared after the interrupt is cleared. (WTC)

Register 18.9. LP_PERI_BUS_TIMEOUT_CONF_REG (0x0010)




**LP_PERI_BUS_TIMEOUT_THRES** Configures the timeout threshold for bus access for accessing LP peripheral register, corresponding to the number of clock cycles of the clock domain. (R/W)

**LP_PERI_BUS_TIMEOUT_INT_CLEAR** Write 1 to clear timeout interrupt.(WT)

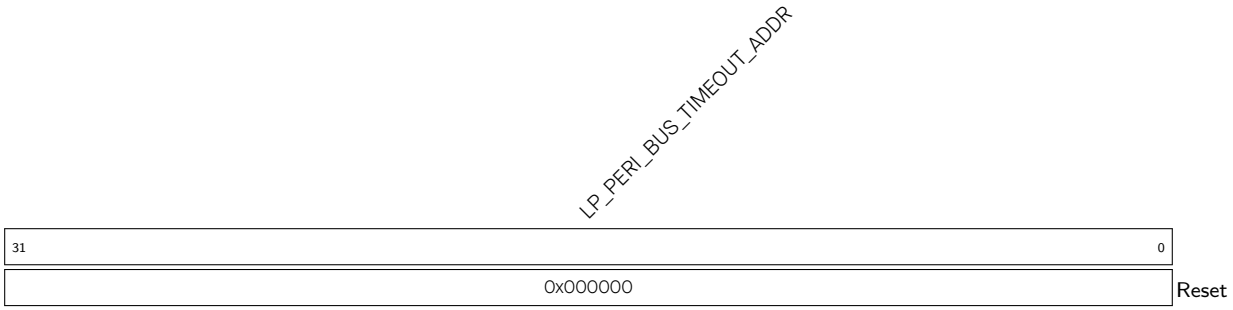
**LP_PERI_BUS_TIMEOUT_PROTECT_EN** Configures whether to enable timeout protection for accessing LP peripheral registers.

0: Disable

1: Enable

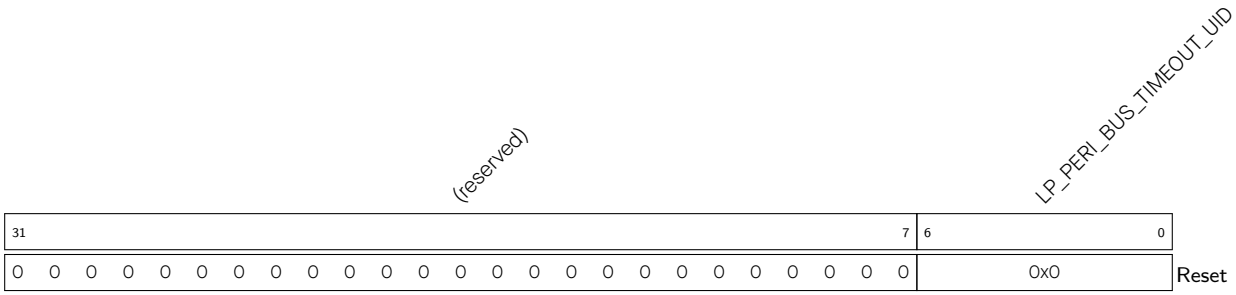
(R/W)

Register 18.10. LP_PERI_BUS_TIMEOUT_ADDR_REG (0x0014)



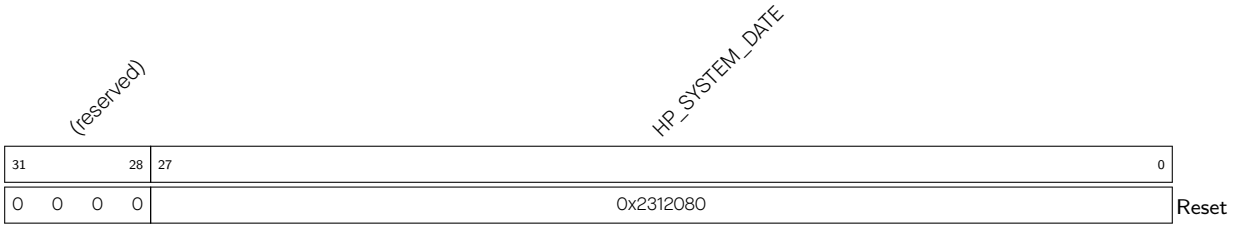

**LP_PERI_BUS_TIMEOUT_ADDR** Represents the address information of abnormal access. (RO)

Register 18.11. LP_PERI_BUS_TIMEOUT_UID_REG (0x0018)




**LP_PERI_BUS_TIMEOUT_UID** Represents the master ID[4:0] and master permission[6:5] when trigger timeout. For details, see Chapter 17 *Permission Control (PMS)*. This register will be cleared after the interrupt is cleared. (WTC)

Register 18.12. HP_SYSTEM_DATE_REG (0x03FC)




**HP_SYSTEM_DATE** Version control register. (R/W)

# Chapter 19

# Debug Assistant

## 19.1 Overview

Debug Assistant is an auxiliary module that features a set of functions to help locate bugs and issues during software debugging.

## 19.2 Features

The Debug Assistant module has the following features:

- **Region read/write monitoring**: Monitors whether the High-Performance CPU (HP CPU) bus reads from or writes to a specified memory address space. A detected read or write in the monitored address space triggers an interrupt.
- **Stack pointer (SP) monitoring**: Monitors whether the SP exceeds the specified address space. A bounds violation triggers an interrupt.
- **Program counter (PC) logging**: Records the PC value. The last PC value at the most recent reset of the HP CPU can be retrieved.
- **Bus access logging**: Records information about bus access. When the HP CPU or the Direct Memory Access controller (DMA) writes a specified value, the Debug Assistant module records the data type, address of the write operation, and the PC value when the write is performed by the HP CPU, then stores this information in the HP SRAM.

## 19.3 Functional Description

### 19.3.1 Region Read/Write Monitoring

The Debug Assistant module monitors reads and writes performed by the HP CPU bus in a specified address space, i.e., memory region. Whenever the bus reads from or writes to this address space, an interrupt is triggered. Refer to Table 2.4-1 in Chapter 2 *ESP-RISC-V CPU*.

In this chapter, when the HP CPU bus accesses the data space, it is referred to as the **Data bus**. When the HP CPU bus accesses the AHB peripheral space, it is referred to as the **Peripheral bus**. The Debug Assistant module can simultaneously monitor two memory regions (HP CPU region 0 and HP CPU region 1) on the Data bus and the Peripheral bus. The regions should be defined based on the developer's needs.

The supported region read/write monitoring modes are:

- Monitoring read/write operations on the Data bus:

  - HP CPU Data bus reads from HP CPU region 0.
  - HP CPU Data bus writes to HP CPU region 0.
  - HP CPU Data bus reads from HP CPU region 1.
  - HP CPU Data bus writes to HP CPU region 1.
- Monitoring read/write operations on the Peripheral bus:
  - HP CPU Peripheral bus reads from HP CPU region 0.
  - HP CPU Peripheral bus writes to HP CPU region 0.
  - HP CPU Peripheral bus reads from HP CPU region 1.
  - HP CPU Peripheral bus writes to HP CPU region 1.

### 19.3.2 SP Monitoring

The Debug Assistant module monitors the SP to prevent stack overflow or erroneous push/pop operations in the HP CPU. When the HP CPU SP exceeds the monitored region's lower or upper bounds, the module records the PC and generates an interrupt. The bounds are configured by software.

The supported SP monitoring modes are:

- The HP CPU SP exceeds the upper bound of the monitored region.
- The HP CPU SP falls below the lower bound of the monitored region.

### 19.3.3 PC Logging

Software developers may need to identify the PC value at the last reset of the HP CPU. For instance, when the program is stuck and requires a reset, the developer may want to know where the program got stuck to debug. The Debug Assistant module records the PC at the last reset of the HP CPU, which can then be read for debugging.

### 19.3.4 CPU/DMA Bus Access Logging

The Debug Assistant module records write operations of the HP CPU Data bus and the DMA bus in real time. When a write operation occurs or a specific value is written to a specified address space, the module records the bus type, address, PC (only when the write is performed by the HP CPU), and other information. It then stores the data in the HP SRAM in a specific format. The specified address range must fall within the permitted memory regions of HP SRAM or external RAM.

## 19.4 Interrupts

The Debug Assistant can generate the BUS_MONITOR_INT interrupt signal that will be sent to the Interrupt Matrix. The following interrupt sources can generate this interrupt signal.

- BUS_MONITOR_CORE_0_AREA_DRAM0_0_RD_INT: Triggered when the HP CPU Data bus reads from HP CPU region 0.

- BUS_MONITOR_CORE_0_AREA_DRAM0_0_WR_INT: Triggered when the HP CPU Data bus writes to HP CPU region 0.
- BUS_MONITOR_CORE_0_AREA_DRAM0_1_RD_INT: Triggered when the HP CPU Data bus reads from HP CPU region 1.
- BUS_MONITOR_CORE_0_AREA_DRAM0_1_WR_INT: Triggered when the HP CPU Data bus writes to HP CPU region 1.
- BUS_MONITOR_CORE_0_AREA_PIF_0_RD_INT: Triggered when the HP CPU Peripheral bus reads from HP CPU region 0.
- BUS_MONITOR_CORE_0_AREA_PIF_0_WR_INT: Triggered when the HP CPU Peripheral bus writes to HP CPU region 0.
- BUS_MONITOR_CORE_0_AREA_PIF_1_RD_INT: Triggered when the HP CPU Peripheral bus reads from HP CPU region 1.
- BUS_MONITOR_CORE_0_AREA_PIF_1_WR_INT: Triggered when the HP CPU Peripheral bus writes to HP CPU region 1.
- BUS_MONITOR_CORE_0_SP_SPILL_MIN_INT: Triggered when the HP CPU SP falls below the lower bound of the monitored region.
- BUS_MONITOR_CORE_0_SP_SPILL_MAX_INT: Triggered when the HP CPU SP exceeds the upper bound of the monitored region.

**Note:**
For definitions of *interrupt*, *interrupt signal*, *interrupt source*, and their correlations, please refer to Chapter 10 *Interrupt Matrix* > Section 10.2 *Interrupt Terminology in ESP32-C61*.

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*.

## 19.5 Programming Procedures

### 19.5.1 Region Monitoring and SP Monitoring Configuration

The configuration process for region monitoring and SP monitoring is as follows:

1. Configure the monitored region and SP bounds.
   - Set the HP CPU Data bus region 0 with BUS_MONITOR_CORE_0_AREA_DRAM0_0_MIN_REG and BUS_MONITOR_CORE_0_AREA_DRAM0_0_MAX_REG.
   - Set the HP CPU Data bus region 1 with BUS_MONITOR_CORE_0_AREA_DRAM0_1_MIN_REG and BUS_MONITOR_CORE_0_AREA_DRAM0_1_MAX_REG.
   - Set the HP CPU Peripheral bus region 0 with BUS_MONITOR_CORE_0_AREA_PIF_0_MIN_REG and BUS_MONITOR_CORE_0_AREA_PIF_0_MAX_REG.
   - Set the HP CPU Peripheral bus region 1 with BUS_MONITOR_CORE_0_AREA_PIF_1_MIN_REG and BUS_MONITOR_CORE_0_AREA_PIF_1_MAX_REG.

- Set the HP CPU SP bounds with BUS_MONITOR_CORE_0_SP_MIN_REG and BUS_MONITOR_CORE_0_SP_MAX_REG.

2. Set BUS_MONITOR_CORE_0_INTR_ENA_REG to enable interrupts for monitoring mode.
3. Set BUS_MONITOR_CORE_0_MONTR_ENA_REG to enable the monitoring mode(s). You can enable multiple monitoring modes simultaneously.
4. Read BUS_MONITOR_CORE_0_INTR_RAW_REG to obtain the raw interrupt status of a monitoring mode.
5. Set BUS_MONITOR_CORE_0_INTR_CLR_REG to clear the interrupt for a monitoring mode.

For example, if the Debug Assistant module needs to monitor whether the HP CPU Data bus has written to the [A ~ B] address space, the user can enable monitoring in either the HP CPU Data bus region 0 or region 1. The following configuration process is based on region 0:

1. Set BUS_MONITOR_CORE_0_RCD_PDEBUGEN to 1 to enable the HP CPU to update the PC signals to the Debug Assistant module.
2. Set BUS_MONITOR_CORE_0_AREA_DRAMO_0_MIN_REG to Address A.
3. Set BUS_MONITOR_CORE_0_AREA_DRAMO_0_MAX_REG to Address B.
4. Set BUS_MONITOR_CORE_0_INTR_ENA_REG bit[1] to enable the interrupt for write operations by the HP CPU Data bus in region 0.
5. Set BUS_MONITOR_CORE_0_MONTR_ENA_REG bit[1] to enable monitoring of write operations by the HP CPU Data bus in region 0.
6. Configure the interrupt matrix to map BUS_MONITOR_INTR into the HP CPU interrupt (refer to Chapter 10 *Interrupt Matrix*).
7. After the interrupt is triggered:
   - Read BUS_MONITOR_CORE_0_INTR_RAW_REG to identify the interrupt source.
   - If the interrupt is triggered by the HP CPU region monitoring, read BUS_MONITOR_CORE_0_AREA_PC for the HP CPU PC value and BUS_MONITOR_CORE_0_AREA_SP for the HP CPU SP.
   - If the interrupt is triggered by SP monitoring, read BUS_MONITOR_CORE_0_SP_PC for the HP CPU PC value.
   - Write 1 to the corresponding bit of BUS_MONITOR_CORE_0_INTR_CLR_REG to clear the interrupt.

### 19.5.2 PC Logging Configuration

Configure BUS_MONITOR_CORE_0_RCD_PDEBUGEN to 1 to enable the HP CPU to update the PC signals to the Debug Assistant module. If BUS_MONITOR_CORE_0_RCD_RECORDEN is also set to 1, BUS_MONITOR_CORE_0_RCD_PDEBUGPC_REG will record the HP CPU's PC value, and BUS_MONITOR_CORE_0_RCD_PDEBUGSP_REG will record the HP CPU's SP value. Otherwise, these two registers will retain their original values.

When the HP CPU resets, BUS_MONITOR_CORE_0_RCD_EN_REG will reset, while BUS_MONITOR_CORE_0_RCD_PDEBUGPC_REG and BUS_MONITOR_CORE_0_RCD_PDEBUGSP_REG will not. Therefore, these two registers will retain the PC and SP values during the HP CPU reset.

### 19.5.3 CPU/DMA Bus Access Logging Configuration

#### 19.5.3.1 Bus Access Logging 1 Configuration

Bus access logging 1 includes HP CPU bus access logging (HP SRAM and external RAM monitoring) and DMA bus access logging (HP SRAM monitoring). The configuration process is described below.

1. Configure the monitored address space: set MEM_MONITOR_LOG_MIN_REG and MEM_MONITOR_LOG_MAX_REG to specify the monitored address space. For HP CPU, the monitored address space should be in the range of HP SRAM (0x4080_0000 ~ 0x4084_FFFF) or external RAM (0x4200_0000 ~ 0x43FF_FFFF). For DMA, the monitored address space should be in the range of HP SRAM (0x4080_0000 ~ 0x4084_FFFF).
2. Configure the monitoring mode with MEM_MONITOR_LOG_MODE:
   - Write monitoring (detects bus write operations)
   - Word monitoring (detects writes of a specific word)
   - Halfword monitoring (detects writes of a specific halfword)
   - Byte monitoring (detects writes of a specific byte)
3. Configure the specific values to monitor:
   - In word monitoring mode, MEM_MONITOR_LOG_CHECK_DATA_REG specifies the monitored word.
   - In halfword monitoring mode, MEM_MONITOR_LOG_CHECK_DATA_REG[15:0] specifies the monitored halfword.
   - In byte monitoring mode, MEM_MONITOR_LOG_CHECK_DATA_REG[7:0] specifies the monitored byte.
   - MEM_MONITOR_LOG_DATA_MASK_REG masks the byte(s) specified in MEM_MONITOR_LOG_CHECK_DATA_REG. A masked byte can be any value. For example, in word monitoring, if MEM_MONITOR_LOG_CHECK_DATA_REG is set to 0x01020304 and MEM_MONITOR_LOG_DATA_MASK_REG is set to 0x1, then any writes matching the 0x010203XX pattern will be recorded.
4. Configure the storage space for recorded data:
   - MEM_MONITOR_LOG_MEM_START_REG and MEM_MONITOR_LOG_MEM_END_REG specify the storage space for recorded data. The storage space must be in the range of 0x4080_0000 ~ 0x4084_FFFF.
   - Set MEM_MONITOR_LOG_MEM_ADDR_UPDATE_REG to update the value in MEM_MONITOR_LOG_MEM_CURRENT_ADDR_REG to MEM_MONITOR_LOG_MEM_START_REG.
   - Configure the Debug Assistant module's permission to access HP SRAM. The Debug Assistant module can access HP SRAM only when the access permission is enabled. For more information, refer to Chapter 17 *Permission Control (PMS)*.
5. Configure the writing mode for recorded data for loop mode or non-loop mode:
   - In loop mode, writing to the specified address space occurs in loops. When writing reaches the end address, it returns to the starting address and continues, overwriting previously recorded data. Set MEM_MONITOR_LOG_MEM_LOOP_ENABLE to 1 to enable loop mode.

For example, if 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 data written by the 1st to 5th operations.

- In non-loop mode, when writing reaches the end address, it stops at the end address and discards the remaining data. The previously recorded data will not be overwritten. Clear MEM_MONITOR_LOG_MEM_LOOP_ENABLE to enable non-loop mode.
  For example, if 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. Therefore, address 0 ~ 4 will store the values written by the 1st to 5th operations, while the values of the 6th to 10th operations will be discarded.

6. Configure bus enable registers.
   - Enable HP CPU and DMA bus access logging respectively with MEM_MONITOR_LOG_CORE_ENA and MEM_MONITOR_LOG_DMA_0_ENA. They can be enabled at the same time.

The Debug Assistant module first writes the recorded data to internal buffer A, then fetches the data from the buffer and writes it to the configured memory space. When monitored behaviors are triggered continuously, the generated recording packets may fill the buffer, preventing it from accepting new packets. In such cases, the module discards incoming packets and buffers a LOST packet instead before the buffer reaches full capacity. The DMA LOST packet contains information about the bus type of the discarded packets but not the number of discarded packets.

When bus access logging is finished, the recorded data can be read from memory for decoding. The recorded data consists of three packet formats: HP CPU packet (for the HP CPU Data bus), DMA packet (for the DMA Data bus), and LOST packet. The packet formats are shown in Tables 19.5-1, 19.5-2, and 19.5-3.

**Table 19.5-1. HP CPU 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 19.5-2. DMA Packet Format**

| Bit[31:8] | Bit[7:3] | Bit[2:1] | Bit[0] |
|---|---|---|---|
| addr_offset | dma_source | format | anchored(0) |

**Table 19.5-3. LOST Packet Format**

| Bit[31:7] | Bit[6:3] | Bit[2:1] | Bit[0] |
|---|---|---|---|
| reserved | lost_source | format | anchored(0) |

The data packet formats show that the HP CPU packet is 64 bits, while the other packets are 32 bits each. These packets have the following fields:

- **format** – indicates the source of the packet type:
  - 0: HP CPU packet
  - 1: DMA packet

  - 2: Reserved
  - 3: LOST packet

- **pc_offset** - indicates the offset of the HP CPU 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 + {MEM_MONITOR_LOG_MIN_REG[31:2], 2'b0}.
- **dma_source** - the source of DMA access. Details can be found in Table 19.5-4, where sources corresponding to values 16 ~ 31 are detailed in *4 GDMA Controller (GDMA)*.
- **lost_source** - indicates which packets were discarded when the LOST packet was generated.
  - Bit[4:3]:
    * 0: HP CPU packets were not discarded
    * 3: HP CPU packets were discarded
    * Other values: reserved
  - Bit[5]:
    * 0: DMA packets were not discarded
    * 1: DMA packets were discarded
  - Bit[6]: Reserved
- **anchored** - indicates the location of the 32 bits in the data packet:
  - 0: Lower 32 bits
  - 1: Higher 32 bits

**Table 19.5-4. DMA Access Source**

| Value | Source |
|---|---|
| 0 | HP CPU |
| 1 | Reserved |
| 2 | Reserved |
| 3 | Reserved |
| 4 | Reserved |
| 5 | MEM_MONITOR |
| 6 | TRACE |
| 7 | Reserved |
| 8 | PSRAM_MEM_MONITOR |
| 9 ~ 15 | Reserved |
| 16 ~ 31 | Refer to the peripheral corresponding to the value 0 ~ 15 in Chapter *4 GDMA Controller (GDMA)* > Table 4.4-1. For example, a value of 16 matches the peripheral corresponding to 0 in this table, and a value of 17 matches the peripheral corresponding to 1. |

The module's internal buffer is 32 bits wide. If the HP CPU and DMA bus access logging are enabled simultaneously and data is recorded at the same time, the HP CPU packets are cached in the buffer first, followed by the DMA packets. The Debug Assistant module automatically fetches the buffered data and stores it in the specified memory space with a 32-bit data width.

In loop mode, data looping multiple times in the storage memory may cause residual data, which can interfere with packet parsing. For example, the lower 32 bits of an HP CPU packet may overwrite, leaving higher 32-bit residual data. Therefore, users must filter out possible residual data when determining the starting position of the first valid packet. Read MEM_MONITOR_LOG_MEM_CURRENT_ADDR_REG to identify the starting position of the packet, then check the anchored bit value of the packet. If it is 0, retain the data. If it is 1, discard it.

The packet parsing process is described below.

- Read MEM_MONITOR_LOG_MEM_FULL_FLAG to determine whether there is a data overflow.
  - If no, the address space to read is MEM_MONITOR_LOG_MEM_START_REG ~ MEM_MONITOR_LOG_MEM_CURRENT_ADDR_REG − 4.
  - If yes and loop mode is enabled, the address space is MEM_MONITOR_LOG_MEM_CURRENT_ADDR_REG ~ MEM_MONITOR_LOG_MEM_END_REG and MEM_MONITOR_LOG_MEM_START_REG ~ MEM_MONITOR_LOG_MEM_CURRENT_ADDR_REG − 4.
  - If yes and loop mode is not enabled, the address space is MEM_MONITOR_LOG_MEM_START_REG ~ MEM_MONITOR_LOG_MEM_END_REG.
- Read and parse data from the starting address. Read 32 bits at a time.

After completing packet parsing, clear the MEM_MONITOR_LOG_MEM_FULL_FLAG flag bit by setting MEM_MONITOR_CLR_LOG_MEM_FULL_FLAG to 1.

#### 19.5.3.2 Bus Access Logging 2 Configuration

Bus access logging 2 includes DMA bus access logging (external RAM monitoring). The configuration process is described below.

1. Configure monitored address space: Set MEM_MONITOR_LOG_MIN_REG and MEM_MONITOR_LOG_MAX_REG to specify the monitored address space, which should range from 0x4200_0000 to 0x43FF_FFFF.
2. Configure the monitoring mode with MEM_MONITOR_LOG_MODE:
   - Write monitoring (detects bus write operations)
   - Word monitoring (detects writes of a specific word)
   - Halfword monitoring (detects writes of a specific halfword)
   - Byte monitoring (detects writes of a specific byte)
3. Configure the specific values to monitor:
   - In word monitoring mode, MEM_MONITOR_LOG_CHECK_DATA_REG specifies the monitored word.

- In halfword monitoring mode, MEM_MONITOR_LOG_CHECK_DATA_REG[15:0] specifies the monitored halfword.
- In byte monitoring mode, MEM_MONITOR_LOG_CHECK_DATA_REG[7:0] specifies the monitored byte.
- Use MEM_MONITOR_LOG_DATA_MASK_REG to mask the byte specified in MEM_MONITOR_LOG_CHECK_DATA_REG. A masked byte can be any value. For example, if MEM_MONITOR_LOG_CHECK_DATA_REG is set to 0x01020304 and MEM_MONITOR_LOG_DATA_MASK_REG is set to 0x1, then any writes matching the 0x010203XX pattern will be recorded.

4. Configure the storage space for recorded data:
   - MEM_MONITOR_LOG_MEM_START_REG and MEM_MONITOR_LOG_MEM_END_REG specify the storage space for recorded data, which must be in the range of 0x4080_0000 to 0x4084_FFFF.
   - Set MEM_MONITOR_LOG_MEM_ADDR_UPDATE_REG to update the value in MEM_MONITOR_LOG_MEM_CURRENT_ADDR_REG to MEM_MONITOR_LOG_MEM_START_REG.
   - Configure the Debug Assistant module's permission to access the HP SRAM. The Debug Assistant module can access the HP SRAM only when access permission is enabled. For more information, refer to Chapter 17 *Permission Control (PMS)*.
5. Configure the writing mode for recorded data in loop mode or non-loop mode:
   - In loop mode, writing to the specified address space occurs in loops. When writing reaches the end address, it returns to the starting address and continues, overwriting previously recorded data. Set MEM_MONITOR_LOG_MEM_LOOP_ENABLE to enable loop mode.
   - In non-loop mode, when writing reaches the end address, it stops and discards the remaining data, without overwriting previously recorded data. Clear MEM_MONITOR_LOG_MEM_LOOP_ENABLE to enable non-loop mode.
   - See examples in Section 19.5.3.1 > step 5.
6. Set MEM_MONITOR_LOG_DMA_0_ENA to enable DMA_0 channel groups bus access logging.

The Debug Assistant module first writes the DMA recorded data to the internal buffer B, then fetches the data from the buffer and writes it to the configured memory space. When monitored behaviors are triggered continuously, the generated recording packets may fill the buffer, preventing it from accepting new packets. In such cases, the module discards incoming packets and buffers a DMA LOST packet to prevent the buffer from reaching full capacity. The DMA LOST packet contains information about the bus type of the discarded packets, but not the number of discarded packets.

When bus access logging is finished, read the recorded data from memory for decoding. The recorded data is in two packet formats: DMA_0 packet (corresponding to DMA_0 Data bus) and DMA LOST packet. The packet formats are shown in Table 19.5-5 and 19.5-6.

**Table 19.5-5. DMA_0 Packet Format**

| Bit[31:7] | Bit[6:2] | Bit[1] | Bit[0] |
|---|---|---|---|
| addr_offset | dma_source | format | anchored(0) |

Table 19.5-6. DMA LOST Packet Format

| Bit[31:3] | Bit[2] | Bit[1] | Bit[0] |
|---|---|---|---|
| reserved | lost_source | format | anchored(0) |

The data packet formats show that the DMA_0 and DMA LOST packets are both 32 bits. These packets contain the following fields:

- **format** – indicates the source of the packet type:
  - 0: DMA_0 packet
  - 1: DMA LOST packet
- **addr_offset** – indicates the address offset of a write operation. Actual address = addr_offset + {MEM_MONITOR_LOG_MIN_REG[31:2], 2'b0}.
- **dma_source** – the source of DMA access. Details are in Table 19.5-4.
- **lost_source** – indicates which packets were discarded when the DMA LOST packet was generated:
  - 0: DMA_0 packets were not discarded
  - 1: DMA_0 packets were discarded
- **anchored** – indicates 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 logging of the DMA_0 channel groups is enabled and the record data is generated, the DMA_0 data packets are buffered. The Debug Assistant module automatically fetches the buffered data and stores it in 32-bit width in the specified memory space.

The packet parsing process is described below.

- Read MEM_MONITOR_LOG_MEM_FULL_FLAG to determine whether there is a data overflow.
  - If no, the address space to read is MEM_MONITOR_LOG_MEM_START_REG ~ MEM_MONITOR_LOG_MEM_CURRENT_ADDR_REG − 4.
  - If yes and loop mode is enabled, the address space is MEM_MONITOR_LOG_MEM_CURRENT_ADDR_REG ~ MEM_MONITOR_LOG_MEM_END_REG and MEM_MONITOR_LOG_MEM_START_REG ~ MEM_MONITOR_LOG_MEM_CURRENT_ADDR_REG − 4.
  - If yes and loop mode is not enabled, the address space is MEM_MONITOR_LOG_MEM_START_REG ~ MEM_MONITOR_LOG_MEM_END_REG.
- Read and parse data from the starting address. Read 32 bits at a time.

After packet parsing is complete, clear the MEM_MONITOR_LOG_MEM_FULL_FLAG flag bit by setting MEM_MONITOR_CLR_LOG_MEM_FULL_FLAG to 1.

## 19.6 Register Summary

The addresses of **bus logging configuration registers** in Section 19.6.1 are relative to the **TCM_MEM_MONITOR** base address and the **PSRAM_MEM_MONITOR**. The addresses of other registers in Section 19.6.2 are relative to the **BUS_MONITOR** base address. All base addresses are provided in Table 5.3-2 in Chapter 5 *System and Memory*.

The abbreviations given in Column **Access** are explained in Section *Access Types for Registers*.

### 19.6.1 Bus Logging Configuration Registers

| Name | Description | Address | Access |
|---|---|---|---|
| **Bus logging configuration registers** | | | |
| MEM_MONITOR_LOG_SETTING_REG | Configures bus access logging | 0x0000 | R/W |
| MEM_MONITOR_LOG_SETTING1_REG | Configures bus access logging | 0x0004 | R/W |
| MEM_MONITOR_LOG_CHECK_DATA_REG | Configures data to be monitored for bus access logging | 0x0008 | R/W |
| MEM_MONITOR_LOG_DATA_MASK_REG | Configures data mask for bus access logging | 0x000C | R/W |
| MEM_MONITOR_LOG_MIN_REG | Configures the monitored lower address for bus access logging | 0x0010 | R/W |
| MEM_MONITOR_LOG_MAX_REG | Configures the monitored upper address for bus access logging | 0x0014 | R/W |
| MEM_MONITOR_LOG_MON_ADDR_UPDATE_0_REG | Configures whether to update the monitored address space for HP CPU bus access logging | 0x0018 | WT |
| MEM_MONITOR_LOG_MON_ADDR_UPDATE_1_REG | Configures whether to update the monitored address space for DMA_0 bus access logging | 0x001C | WT |
| MEM_MONITOR_LOG_MEM_START_REG | Configures the starting address of the storage memory for recorded data | 0x0020 | R/W |
| MEM_MONITOR_LOG_MEM_END_REG | Configures the end address of the storage memory for recorded data | 0x0024 | R/W |
| MEM_MONITOR_LOG_MEM_CURRENT_ADDR_REG | Represents the address for the next write | 0x0028 | RO |
| MEM_MONITOR_LOG_MEM_ADDR_UPDATE_REG | Updates the address for the next write with the starting address for the recorded data | 0x002C | WT |
| MEM_MONITOR_LOG_MEM_FULL_FLAG_REG | Represents logging buffer overflow status register | 0x0030 | varies |
| **Clock gating control register** | | | |
| MEM_MONITOR_CLOCK_GATE_REG | Clock gating control register | 0x0034 | R/W |
| **Version control register** | | | |







GoBack

| Name | Description | Address | Access |
|---|---|---|---|
| MEM_MONITOR_DATE_REG | Version control register | 0x03FC | R/W |

### 19.6.2 SOther Registers

| Name | Description | Address | Access |
|---|---|---|---|
| **Monitor configuration registers** | | | |
| BUS_MONITOR_CORE_0_MONTR_ENA_REG | Configures whether to enable HP CPU monitoring | 0x0000 | R/W |
| BUS_MONITOR_CORE_0_AREA_DRAMO_0_MIN_REG | Configures lower boundary address of region 0 monitored on Data bus | 0x0010 | R/W |
| BUS_MONITOR_CORE_0_AREA_DRAMO_0_MAX_REG | Configures upper boundary address of region 0 monitored on Data bus | 0x0014 | R/W |
| BUS_MONITOR_CORE_0_AREA_DRAMO_1_MIN_REG | Configures lower boundary address of region 1 monitored on Data bus | 0x0018 | R/W |
| BUS_MONITOR_CORE_0_AREA_DRAMO_1_MAX_REG | Configures upper boundary address of region 1 monitored on Data bus | 0x001C | R/W |
| BUS_MONITOR_CORE_0_AREA_PIF_0_MIN_REG | Configures lower boundary address of region 0 monitored on Peripheral bus | 0x0020 | R/W |
| BUS_MONITOR_CORE_0_AREA_PIF_0_MAX_REG | Configures upper boundary address of region 0 monitored on Peripheral bus | 0x0024 | R/W |
| BUS_MONITOR_CORE_0_AREA_PIF_1_MIN_REG | Configures lower boundary address of region 1 monitored on Peripheral bus | 0x0028 | R/W |
| BUS_MONITOR_CORE_0_AREA_PIF_1_MAX_REG | Configures upper boundary address of region 1 monitored on Peripheral bus | 0x002C | R/W |
| BUS_MONITOR_CORE_0_AREA_PC_REG | Represents the PC value when HP CPU region monitor is triggered | 0x0030 | RO |
| BUS_MONITOR_CORE_0_AREA_SP_REG | Represents the SP value when HP CPU region monitor is triggered | 0x0034 | RO |
| BUS_MONITOR_CORE_0_SP_MIN_REG | Configures SP monitoring lower boundary address | 0x0038 | R/W |
| BUS_MONITOR_CORE_0_SP_MAX_REG | Configures SP monitoring upper boundary address | 0x003C | R/W |
| BUS_MONITOR_CORE_0_SP_PC_REG | Represents the PC value when HP CPU SP monitor is triggered | 0x0040 | RO |
| **Interrupt configuration registers** | | | |
| BUS_MONITOR_CORE_0_INTR_RAW_REG | HP CPU monitor raw interrupt status register | 0x0004 | RO |
| BUS_MONITOR_CORE_0_INTR_ENA_REG | HP CPU monitor interrupt enable register | 0x0008 | R/W |
| BUS_MONITOR_CORE_0_INTR_CLR_REG | HP CPU monitor interrupt clear register | 0x000C | WT |
| **PC recording configuration register** | | | |
| BUS_MONITOR_CORE_0_RCD_EN_REG | HP CPU PC logging enable register | 0x0044 | R/W |
| **PC recording status registers** | | | |
| BUS_MONITOR_CORE_0_RCD_PDEBUGPC_REG | HP CPU PC logging register | 0x0048 | RO |

| Name | Description | Address | Access |
|---|---|---|---|
| BUS_MONITOR_CORE_0_RCD_PDEBUGSP_REG | HP CPU SP logging register | 0x004C | RO |
| **CPU status registers** | | | |
| BUS_MONITOR_CORE_0_LASTPC_BEFORE_EXCEPTION_REG | PC of the last command before HP CPU enters exception | 0x0070 | RO |
| BUS_MONITOR_CORE_0_DEBUG_MODE_REG | HP CPU debug mode status register | 0x0074 | RO |
| **Clock control register** | | | |
| BUS_MONITOR_CLOCK_GATE_REG | Clock control register | 0x0108 | R/W |
| **Version control register** | | | |
| BUS_MONITOR_DATE_REG | Version control register | 0x03FC | R/W |

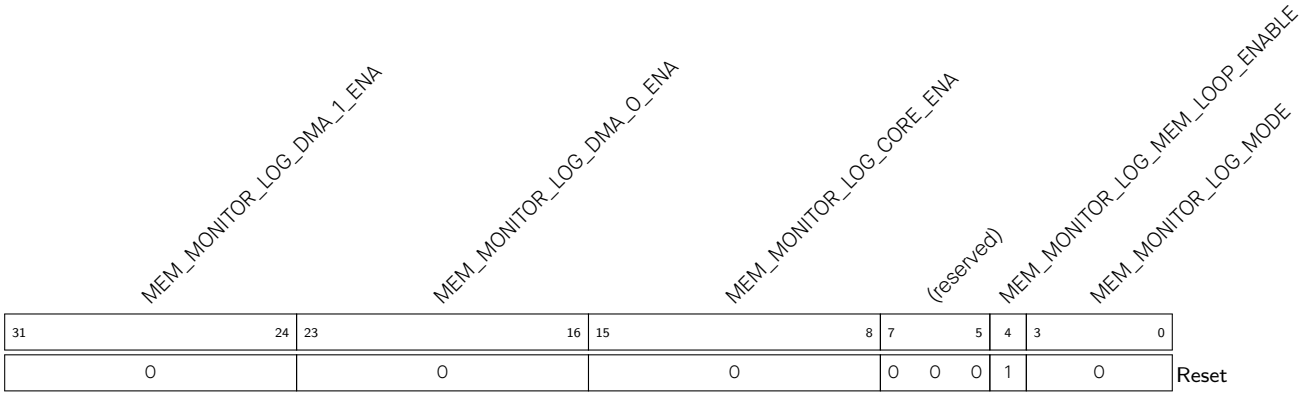
## 19.7 Registers

The addresses of **bus logging configuration registers** in Section 19.7.1 are relative to the **TCM_MEM_MONITOR** base address and the **PSRAM_MEM_MONITOR**. The addresses of other registers in Section 19.7.2 are relative to the **BUS_MONITOR** base address. All base addresses are provided in Table 5.3-2 in Chapter 5 *System and Memory*.

For how to program reserved fields, please refer to Section *Programming Reserved Register Field*.

### 19.7.1 Bus Logging Configuration Registers

**Register 19.1. MEM_MONITOR_LOG_SETTING_REG (0x0000)**




**MEM_MONITOR_LOG_MODE** Configures monitoring modes.

1: Enable write monitoring
2: Enable word monitoring
4: Enable halfword monitoring
8: Enable byte monitoring
Other values: Invalid
(R/W)

**MEM_MONITOR_LOG_MEM_LOOP_ENABLE** Configures the writing mode for recorded data.

0: Non-loop mode
1: Loop mode
(R/W)

**MEM_MONITOR_LOG_CORE_ENA** Configures whether to enable HP CPU bus access logging.

0: Disable
1: Enable
Other values: Invalid
(R/W)

**Continued on the next page...**

Register 19.1. MEM_MONITOR_LOG_SETTING_REG (0x0000)

Continued from the previous page...

**MEM_MONITOR_LOG_DMA_0_ENA** Configures whether to enable DMA_0 bus access logging.

0: Disable

1: Enable

Other values: Invalid

(R/W)

**MEM_MONITOR_LOG_DMA_1_ENA** Configures whether to enable DMA_1 bus access logging.

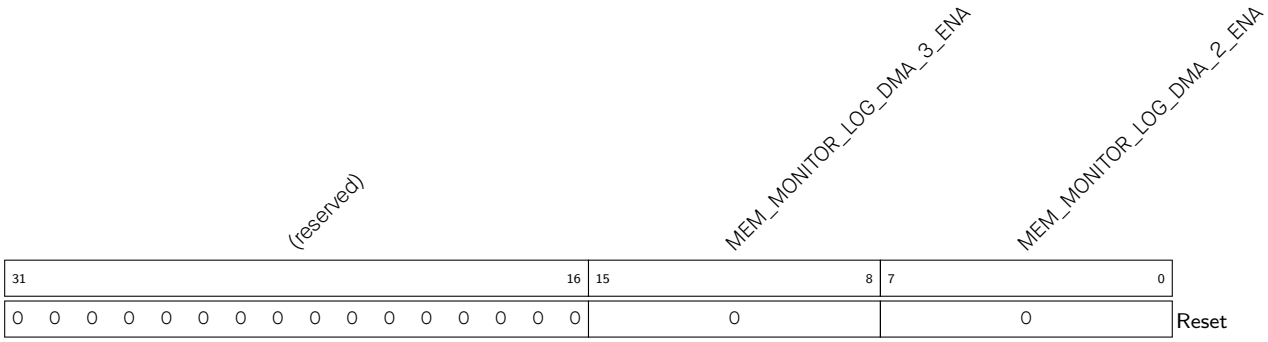
0: Disable

1: Enable

Other values: Invalid

(R/W)

Register 19.2. MEM_MONITOR_LOG_SETTING1_REG (0x0004)




**MEM_MONITOR_LOG_DMA_2_ENA** Configures whether to enable DMA_2 bus access logging.

0: Disable

1: Enable

Other values: Invalid

(R/W)

**MEM_MONITOR_LOG_DMA_3_ENA** Configures whether to enable DMA_3 bus access logging.

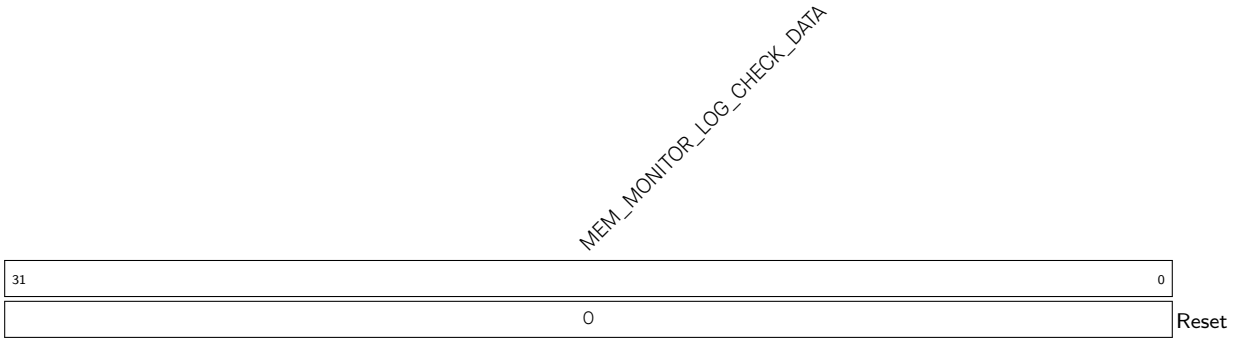
0: Disable

1: Enable

Other values: Invalid

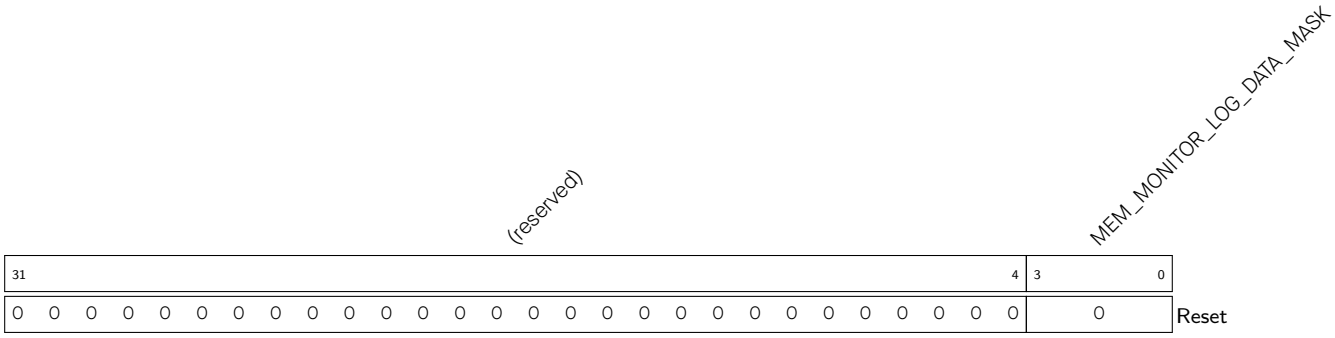
(R/W)

Register 19.3. MEM_MONITOR_LOG_CHECK_DATA_REG (0x0008)




**MEM_MONITOR_LOG_CHECK_DATA** Configures the data to be monitored during bus accessing. (R/W)

Register 19.4. MEM_MONITOR_LOG_DATA_MASK_REG (0x000C)




**MEM_MONITOR_LOG_DATA_MASK** Configures which byte(s) in MEM_MONITOR_LOG_CHECK_DATA_REG to mask. Multiple bytes can be masked at the same time.

bit[0]: Configures whether to mask the least significant byte of MEM_MONITOR_LOG_CHECK_DATA_REG.

0: Not mask

1: Mask

bit[1]: Configures whether to mask the second least significant byte of MEM_MONITOR_LOG_CHECK_DATA_REG.

0: Not mask

1: Mask

bit[2]: Configures whether to mask the second most significant byte of MEM_MONITOR_LOG_CHECK_DATA_REG.

0: Not mask

1: Mask

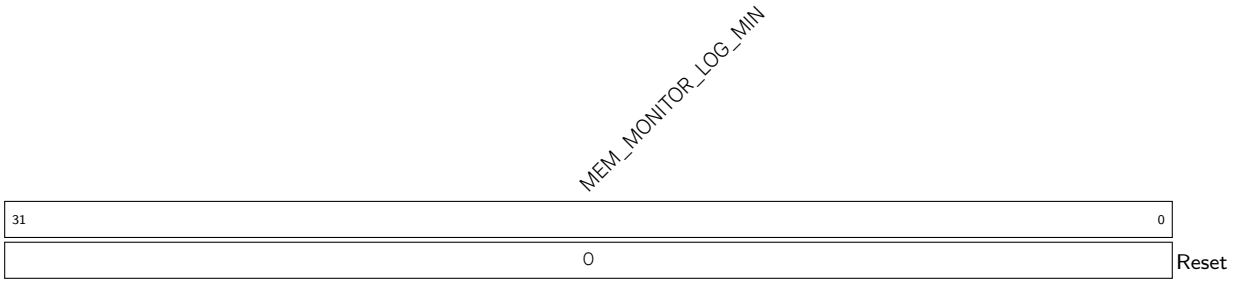
bit[3]: Configures whether to mask the most significant byte of MEM_MONITOR_LOG_CHECK_DATA_REG.

0: Not mask

1: Mask

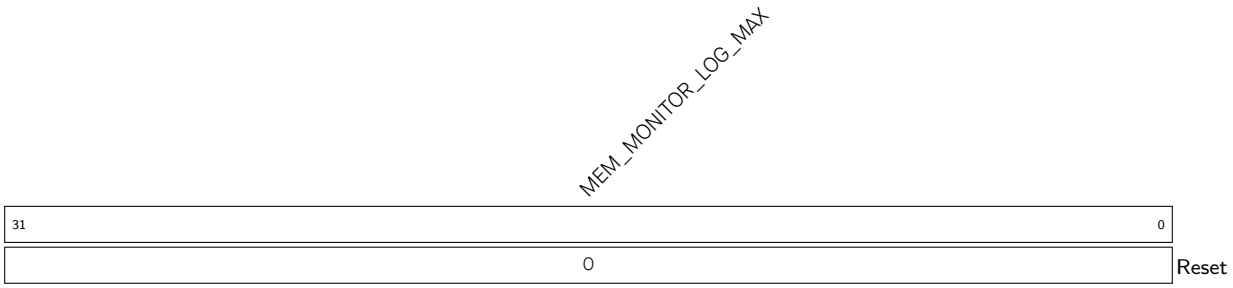
(R/W)

Register 19.5. MEM_MONITOR_LOG_MIN_REG (0x0010)



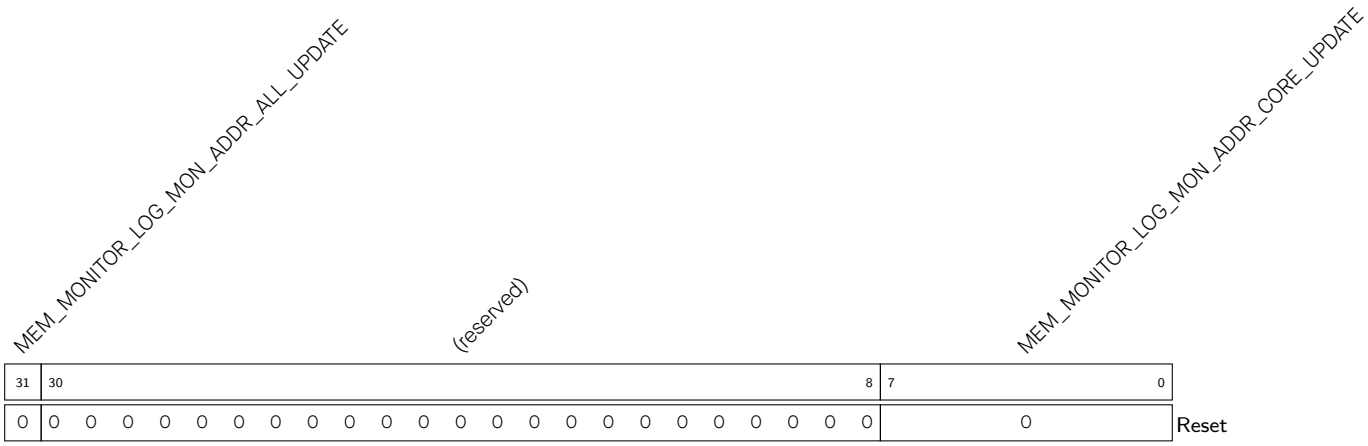

**MEM_MONITOR_LOG_MIN** Configures the lower bound address of the monitored address space. (R/W)

Register 19.6. MEM_MONITOR_LOG_MAX_REG (0x0014)




**MEM_MONITOR_LOG_MAX** Configures the upper bound address of the monitored address space. (R/W)

Register 19.7. MEM_MONITOR_LOG_MON_ADDR_UPDATE_0_REG (0x0018)




**MEM_MONITOR_LOG_MON_ADDR_CORE_UPDATE** Configures whether to update the monitored address space of the HP CPU bus as the address space from MEM_MONITOR_LOG_MIN_REG to MEM_MONITOR_LOG_MAX_REG.

0: Not update

1: Update

Other values: Invalid

(WT)

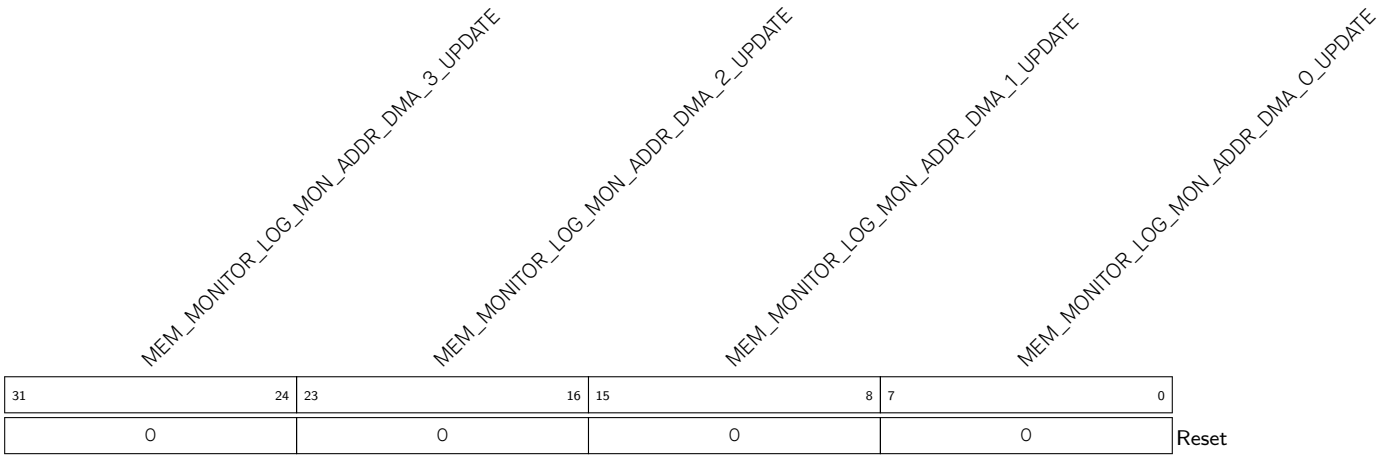
**MEM_MONITOR_LOG_MON_ADDR_ALL_UPDATE** Configures whether to update the monitored address space of all masters as the address space from MEM_MONITOR_LOG_MIN_REG to MEM_MONITOR_LOG_MAX_REG.

0: Not update

1: Update

(WT)

Register 19.8. MEM_MONITOR_LOG_MON_ADDR_UPDATE_1_REG (0x001C)




**MEM_MONITOR_LOG_MON_ADDR_DMA_0_UPDATE** Configures whether to update the monitored address space of the DMA_0 bus as the address space from MEM_MONITOR_LOG_MIN_REG to MEM_MONITOR_LOG_MAX_REG.

0: Not update

1: Update

Other values: Invalid

(WT)

**MEM_MONITOR_LOG_MON_ADDR_DMA_1_UPDATE** Configures whether to update the monitored address space of the DMA_1 bus as the address space from MEM_MONITOR_LOG_MIN_REG to MEM_MONITOR_LOG_MAX_REG.

0: Not update

1: Update

Other values: Invalid

(WT)

**MEM_MONITOR_LOG_MON_ADDR_DMA_2_UPDATE** Configures whether to update the monitored address space of the DMA_2 bus as the address space from MEM_MONITOR_LOG_MIN_REG to MEM_MONITOR_LOG_MAX_REG.

0: Not update

1: Update

Other values: Invalid

(WT)

**MEM_MONITOR_LOG_MON_ADDR_DMA_3_UPDATE** Configures whether to update the monitored address space of the DMA_3 bus as the address space from MEM_MONITOR_LOG_MIN_REG to MEM_MONITOR_LOG_MAX_REG.

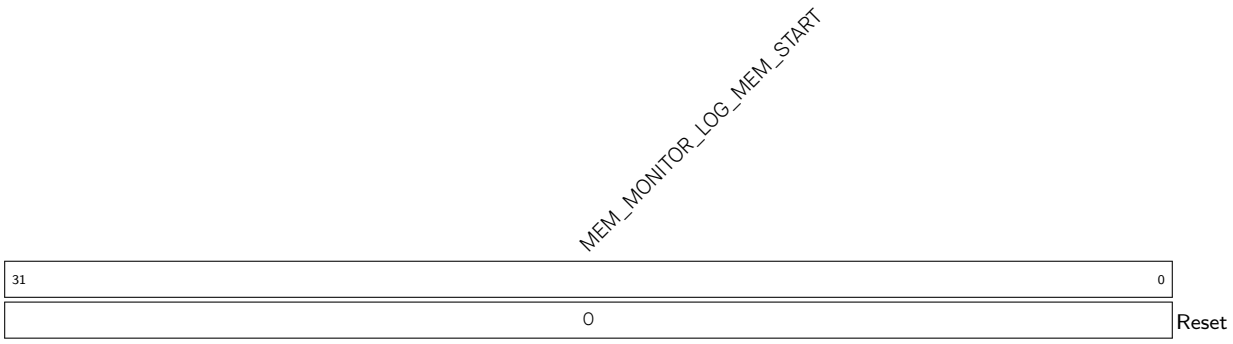
0: Not update

1: Update

Other values: Invalid

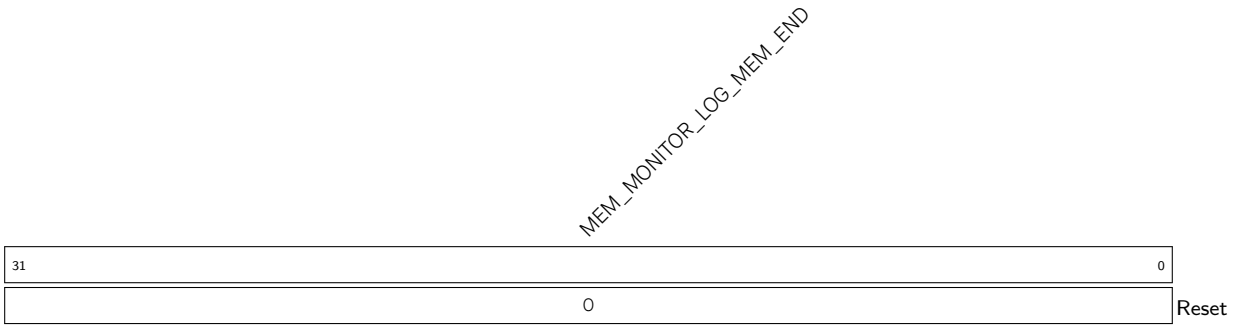
(WT)

Register 19.9. MEM_MONITOR_LOG_MEM_START_REG (0x0020)



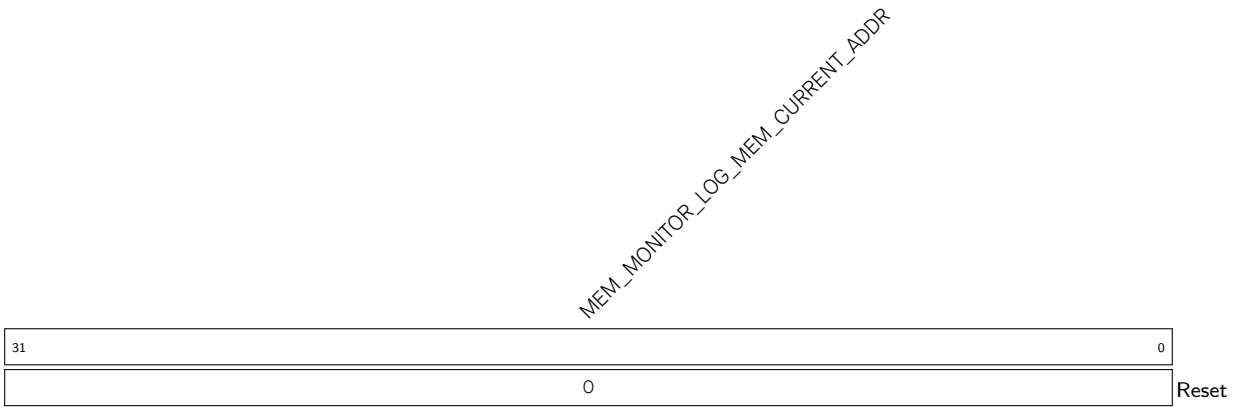

**MEM_MONITOR_LOG_MEM_START** Configures the starting address of the storage space for recorded data. (R/W)

Register 19.10. MEM_MONITOR_LOG_MEM_END_REG (0x0024)



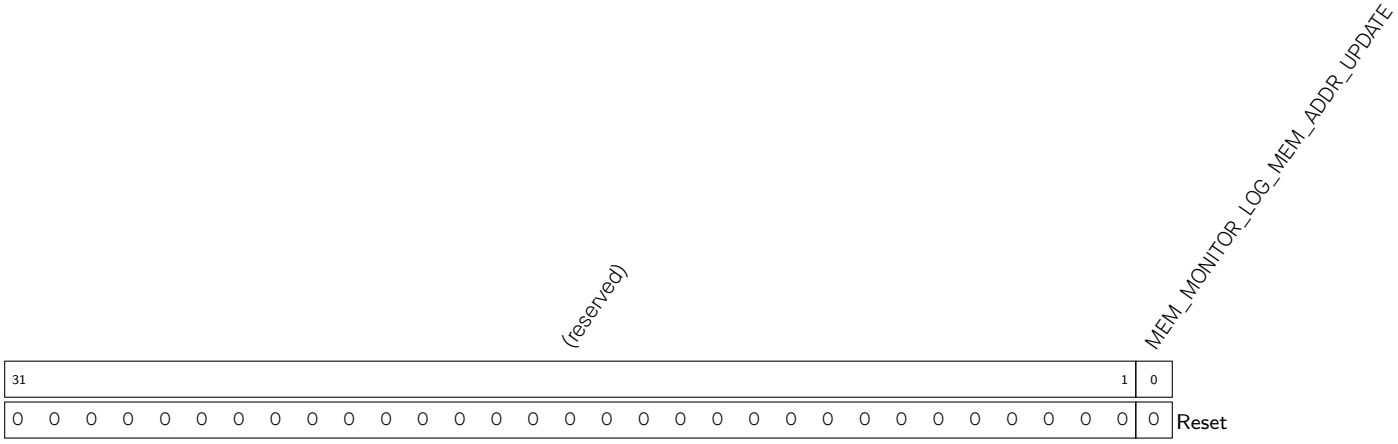

**MEM_MONITOR_LOG_MEM_END** Configures the ending address of the storage space for recorded data. (R/W)

Register 19.11. MEM_MONITOR_LOG_MEM_CURRENT_ADDR_REG (0x0028)




**MEM_MONITOR_LOG_MEM_CURRENT_ADDR** Represents the address of the next write. (RO)

Register 19.12. MEM_MONITOR_LOG_MEM_ADDR_UPDATE_REG (0x002C)



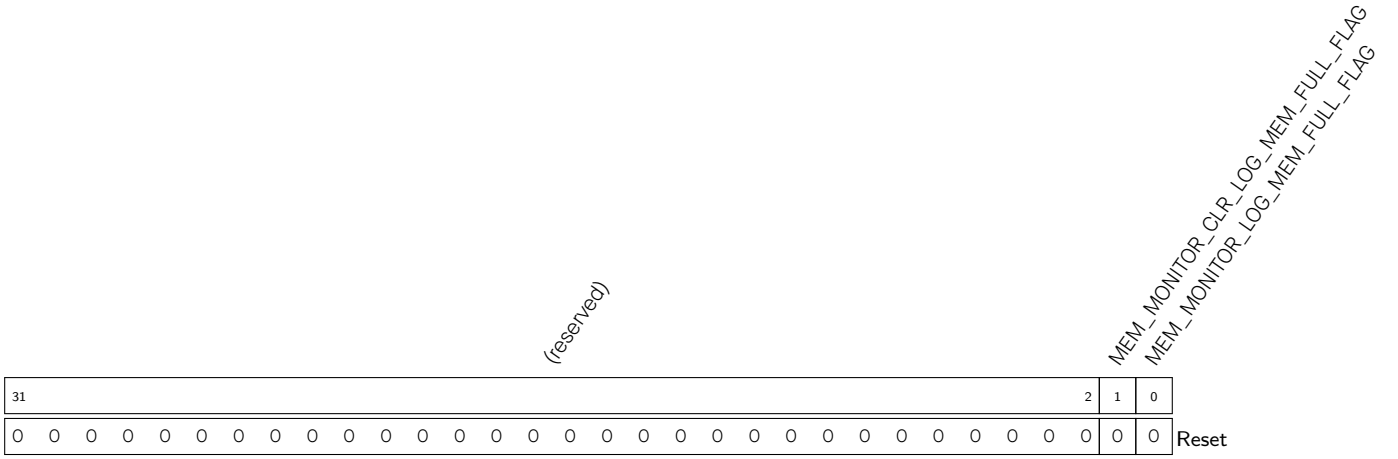

**MEM_MONITOR_LOG_MEM_ADDR_UPDATE** Configures whether to update the value in MEM_MONITOR_LOG_MEM_START_REG to the value of MEM_MONITOR_LOG_MEM_CURRENT_ADDR_REG.

0: Not update

1: Update

(WT)

Register 19.13. MEM_MONITOR_LOG_MEM_FULL_FLAG_REG (0x0030)




**MEM_MONITOR_LOG_MEM_FULL_FLAG** Represents whether data overflows the storage space.

0: Not Overflow

1: Overflow

(RO)

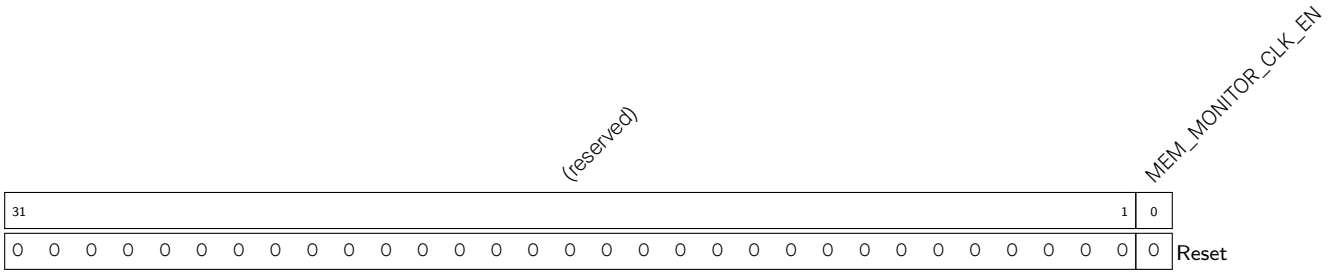
**MEM_MONITOR_CLR_LOG_MEM_FULL_FLAG** Configures whether to clear the MEM_MONITOR_LOG_MEM_FULL_FLAG flag bit.

0: Not clear (default)

1: Clear

(WT)

Register 19.14. MEM_MONITOR_CLOCK_GATE_REG (0x0034)



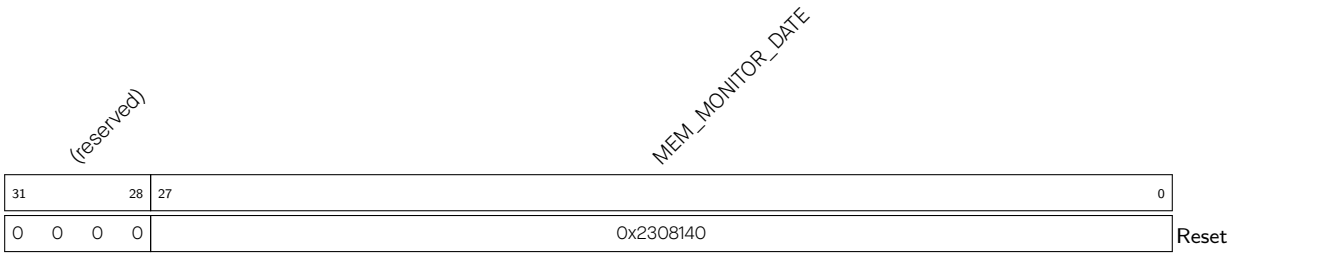

**MEM_MONITOR_CLK_EN** Configures whether to enable the register clock gating.

0: Disable

1: Enable

(R/W)

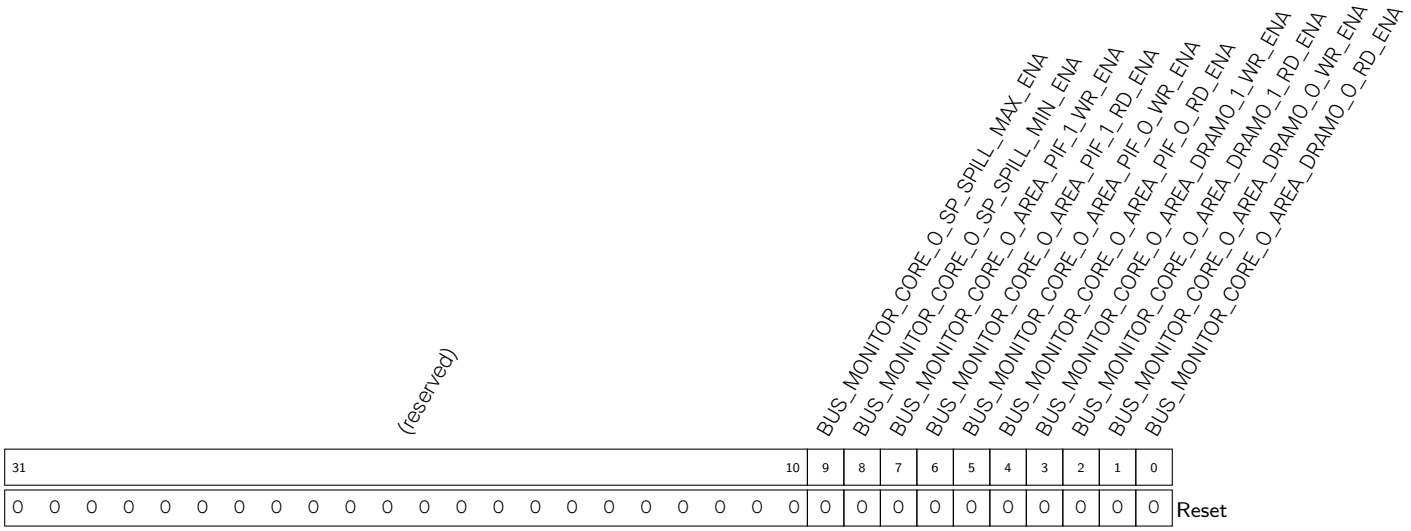
Register 19.15. MEM_MONITOR_DATE_REG (0x03FC)




**MEM_MONITOR_DATE** Version control register. (R/W)

### 19.7.2 Other Registers

Register 19.16. BUS_MONITOR_CORE_0_MONTR_ENA_REG (0x0000)




**BUS_MONITOR_CORE_0_AREA_DRAM0_0_RD_ENA** Configures whether to monitor read operations in region 0 by the Data bus.

0: Not monitor

1: Monitor

(R/W)

**BUS_MONITOR_CORE_0_AREA_DRAM0_0_WR_ENA** Configures whether to monitor write operations in region 0 by the Data bus.

0: Not monitor

1: Monitor

(R/W)

**BUS_MONITOR_CORE_0_AREA_DRAM0_1_RD_ENA** Configures whether to monitor read operations in region 1 by the Data bus.

0: Not Monitor

1: Monitor

(R/W)

**BUS_MONITOR_CORE_0_AREA_DRAM0_1_WR_ENA** Configures whether to monitor write operations in region 1 by the Data bus.

0: Not Monitor

1: Monitor

(R/W)

**BUS_MONITOR_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 19.16. BUS_MONITOR_CORE_0_MONTR_ENA_REG (0x0000)

Continued from the previous page...

**BUS_MONITOR_CORE_0_AREA_PIF_0_WR_ENA** Configures whether to monitor write operations in region 0 by the Peripheral bus.
0: Not Monitor
1: Monitor
(R/W)

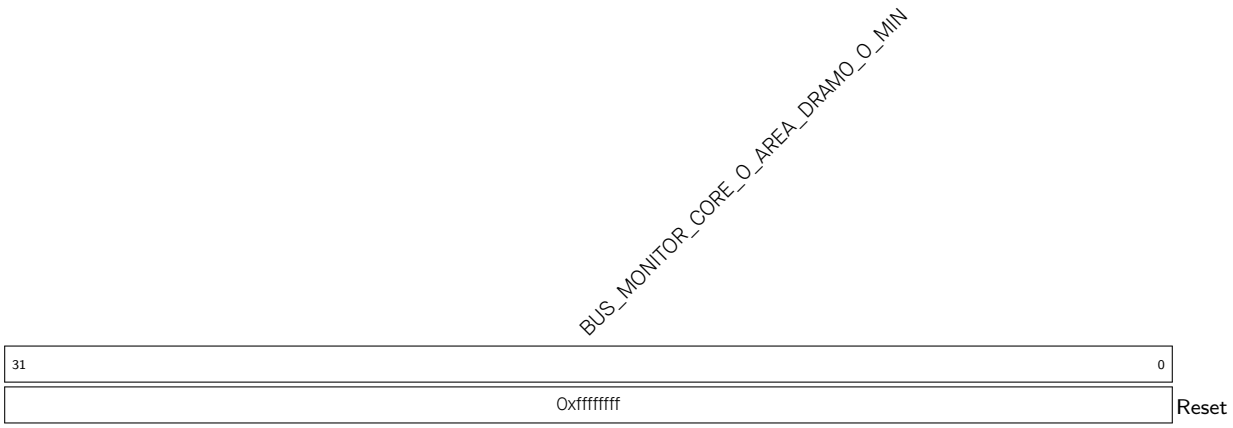
**BUS_MONITOR_CORE_0_AREA_PIF_1_RD_ENA** Configures whether to monitor read operations in region 1 by the Peripheral bus.
0: Not Monitor
1: Monitor
(R/W)

**BUS_MONITOR_CORE_0_AREA_PIF_1_WR_ENA** Configures whether to monitor write operations in region 1 by the Peripheral bus.
0: Not Monitor
1: Monitor
(R/W)

**BUS_MONITOR_CORE_0_SP_SPILL_MIN_ENA** Configures whether to monitor SP exceeding the lower bound address of SP monitored region.
0: Not Monitor
1: Monitor
(R/W)

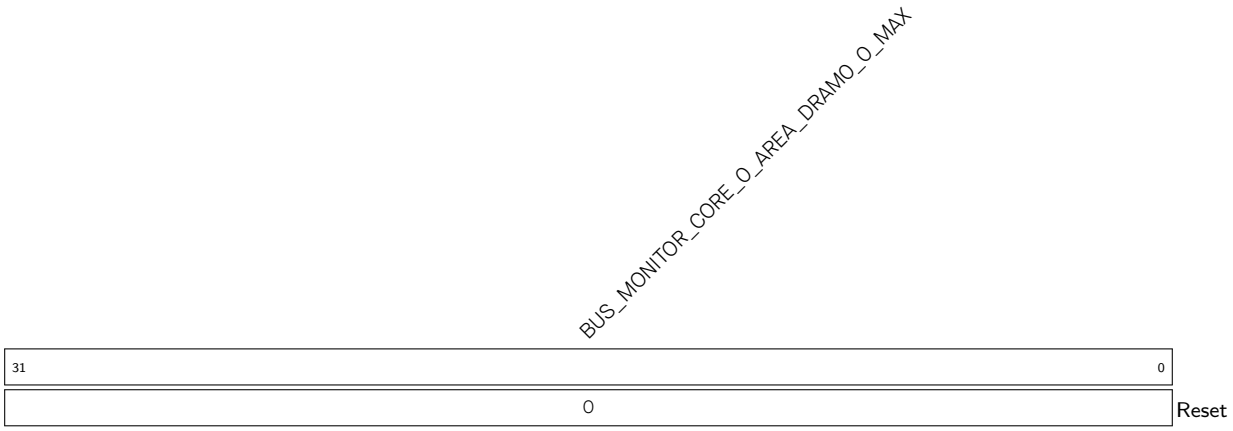
**BUS_MONITOR_CORE_0_SP_SPILL_MAX_ENA** Configures whether to monitor SP exceeding the upper bound address of SP monitored region.
0: Not Monitor
1: Monitor
(R/W)

Register 19.17. BUS_MONITOR_CORE_0_AREA_DRAM0_0_MIN_REG (0x0010)



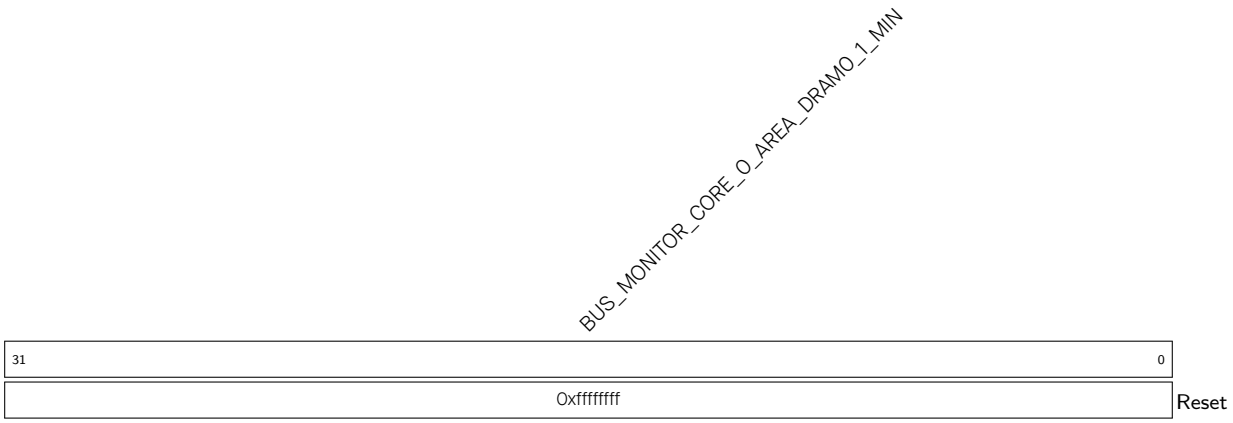

**BUS_MONITOR_CORE_0_AREA_DRAM0_0_MIN** Configures the lower bound address of Data bus region 0. (R/W)

Register 19.18. BUS_MONITOR_CORE_0_AREA_DRAM0_0_MAX_REG (0x0014)



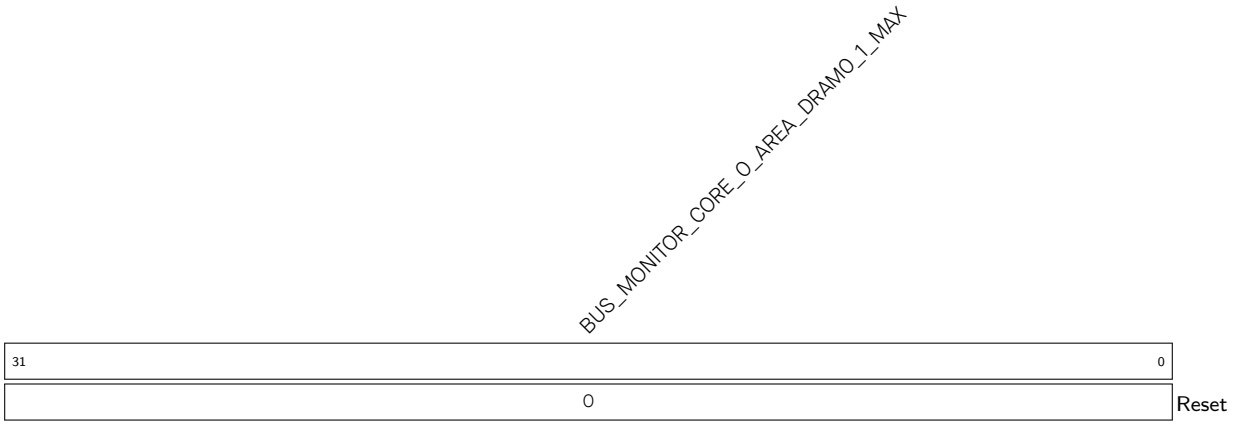

**BUS_MONITOR_CORE_0_AREA_DRAM0_0_MAX** Configures the upper bound address of Data bus region 0. (R/W)

Register 19.19. BUS_MONITOR_CORE_0_AREA_DRAM0_1_MIN_REG (0x0018)



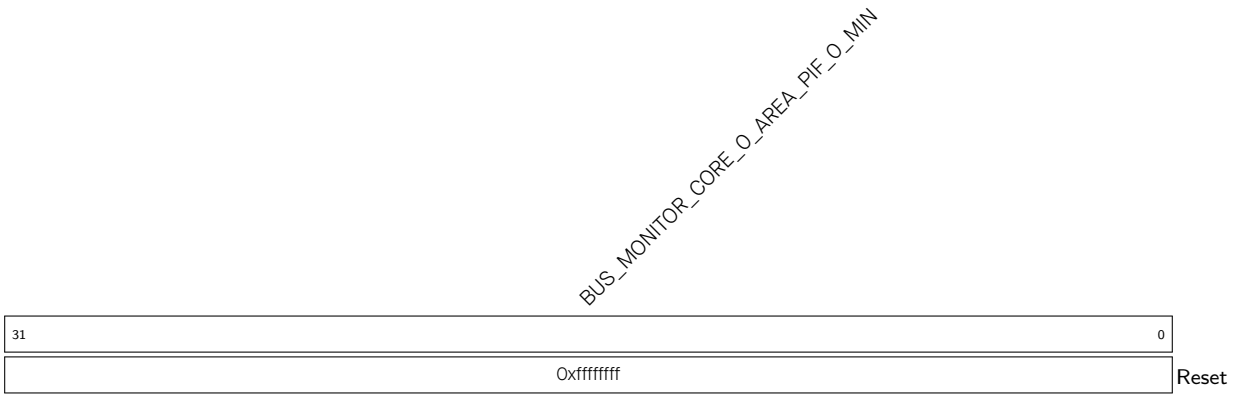

**BUS_MONITOR_CORE_0_AREA_DRAM0_1_MIN** Configures the lower bound address of Data bus region 1. (R/W)

Register 19.20. BUS_MONITOR_CORE_0_AREA_DRAM0_1_MAX_REG (0x001C)



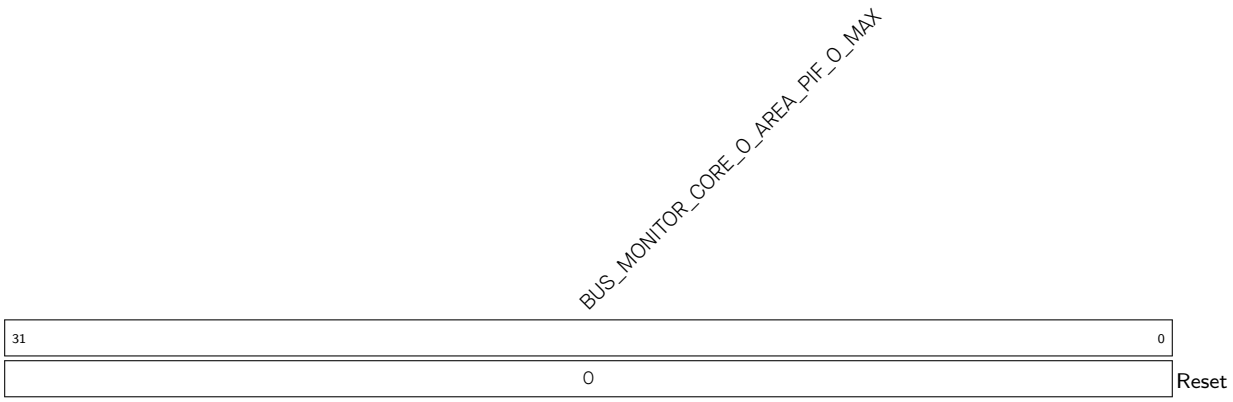

**BUS_MONITOR_CORE_0_AREA_DRAM0_1_MAX** Configures the upper bound address of Data bus region 1. (R/W)

Register 19.21. BUS_MONITOR_CORE_0_AREA_PIF_0_MIN_REG (0x0020)



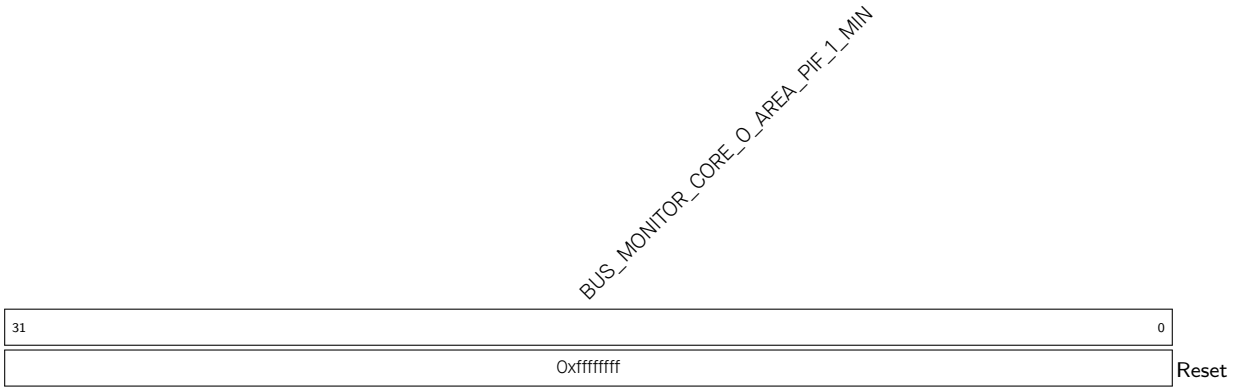

**BUS_MONITOR_CORE_0_AREA_PIF_0_MIN** Configures the lower bound address of Peripheral bus region 0. (R/W)

Register 19.22. BUS_MONITOR_CORE_0_AREA_PIF_0_MAX_REG (0x0024)



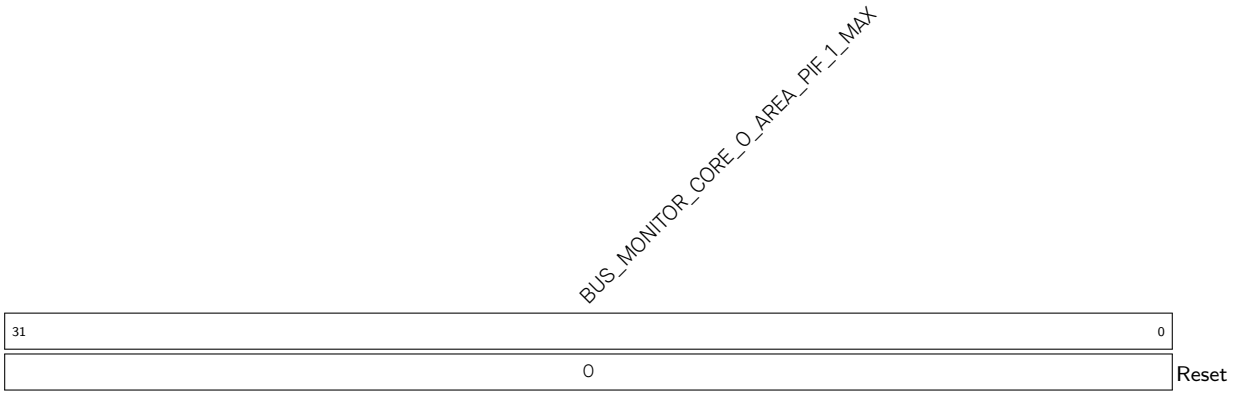

**BUS_MONITOR_CORE_0_AREA_PIF_0_MAX** Configures the upper bound address of Peripheral bus region 0. (R/W)

Register 19.23. BUS_MONITOR_CORE_0_AREA_PIF_1_MIN_REG (0x0028)



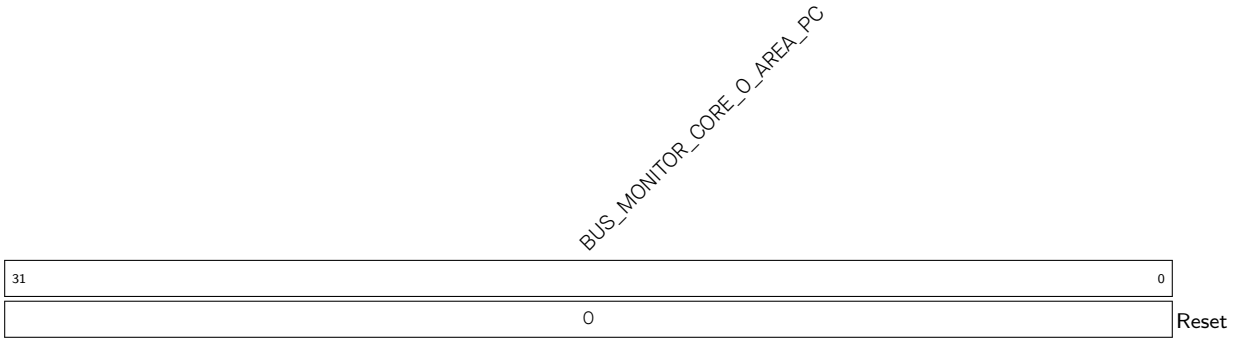

**BUS_MONITOR_CORE_0_AREA_PIF_1_MIN** Configures the lower bound address of Peripheral bus region 1. (R/W)

Register 19.24. BUS_MONITOR_CORE_0_AREA_PIF_1_MAX_REG (0x002C)



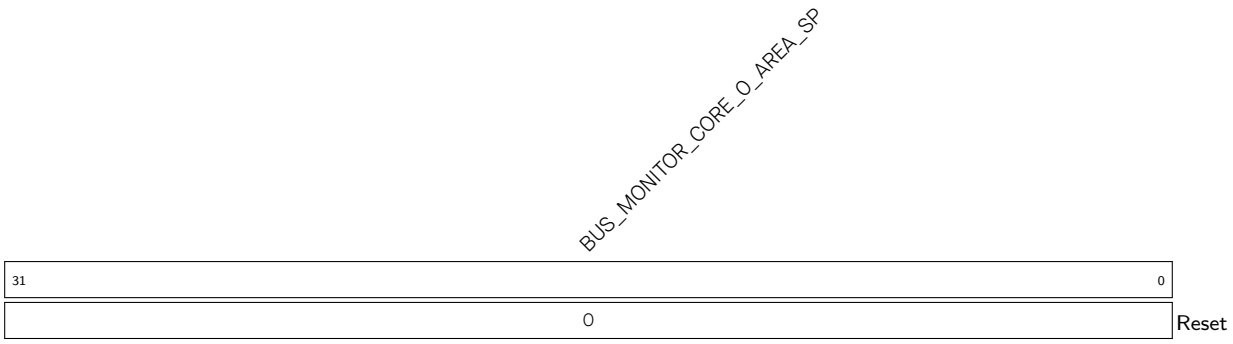

**BUS_MONITOR_CORE_0_AREA_PIF_1_MAX** Configures the upper bound address of Peripheral bus region 1. (R/W)

Register 19.25. BUS_MONITOR_CORE_0_AREA_PC_REG (0x0030)



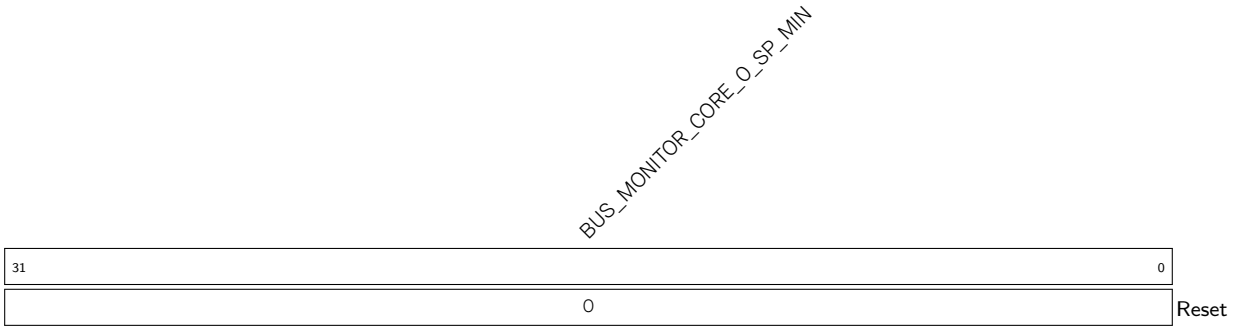

**BUS_MONITOR_CORE_0_AREA_PC** Represents the PC value when an interrupt is triggered during region monitoring. (RO)

Register 19.26. BUS_MONITOR_CORE_0_AREA_SP_REG (0x0034)



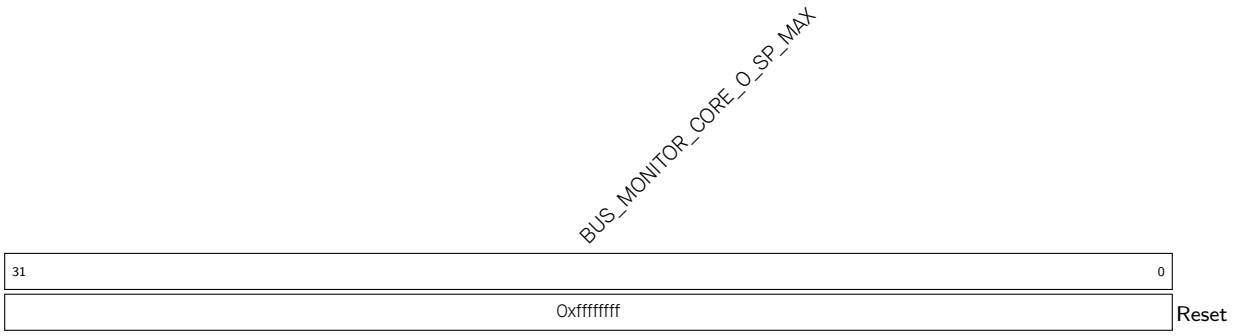

**BUS_MONITOR_CORE_0_AREA_SP** Represents the SP value when an interrupt is triggered during region monitoring. (RO)

Register 19.27. BUS_MONITOR_CORE_0_SP_MIN_REG (0x0038)



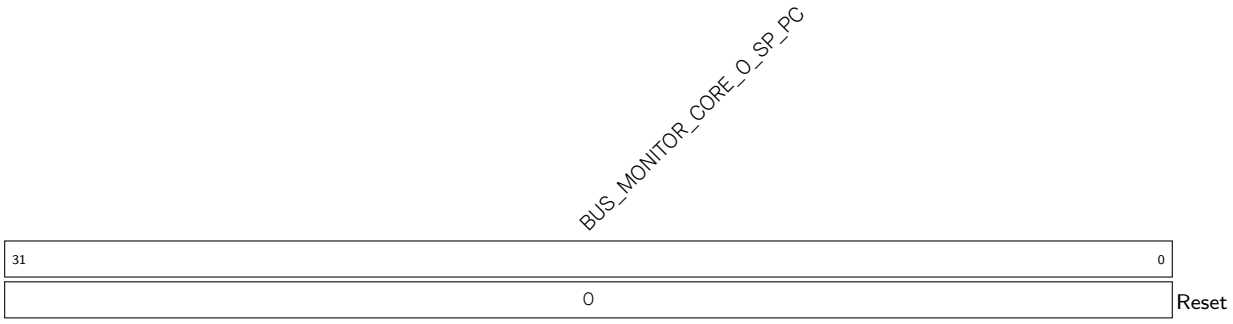

**BUS_MONITOR_CORE_0_SP_MIN** Configures the lower bound address of SP monitored region. (R/W)

Register 19.28. BUS_MONITOR_CORE_0_SP_MAX_REG (0x003C)



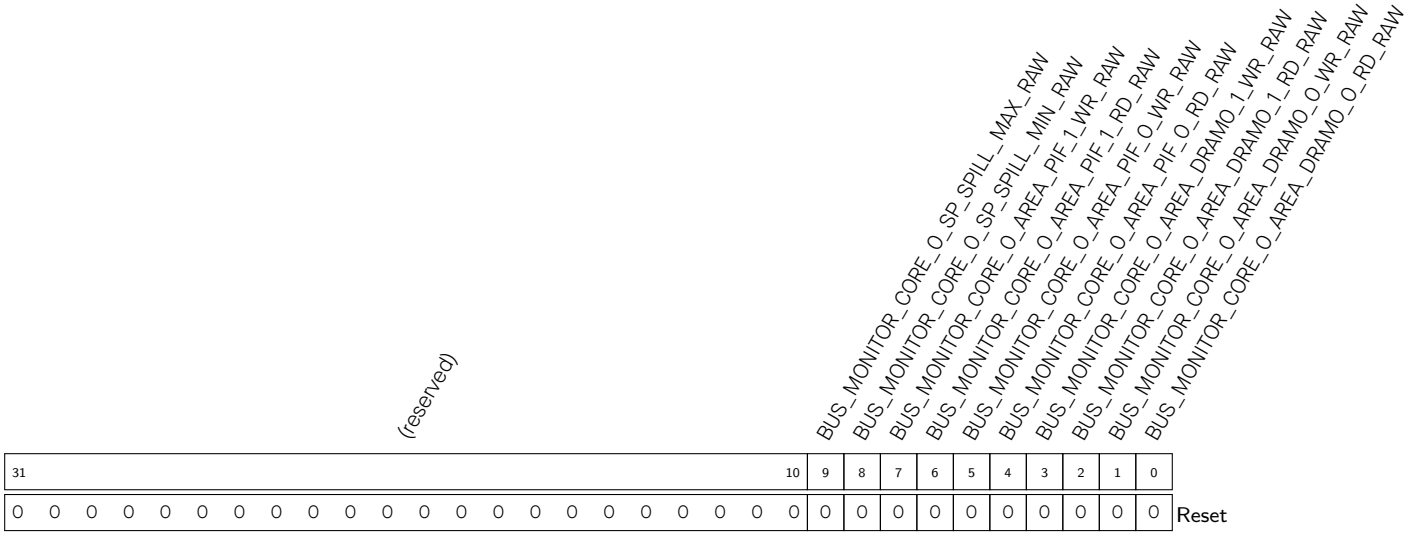

**BUS_MONITOR_CORE_0_SP_MAX** Configures the upper bound address of SP monitored region. (R/W)

Register 19.29. BUS_MONITOR_CORE_0_SP_PC_REG (0x0040)




**BUS_MONITOR_CORE_0_SP_PC** Represents the PC value when SP monitoring is triggered. (RO)

Register 19.30. BUS_MONITOR_CORE_0_INTR_RAW_REG (0x0004)




**BUS_MONITOR_CORE_0_AREA_DRAM0_0_RD_RAW** The raw interrupt status of BUS_MONITOR_CORE_0_AREA_DRAM0_0_RD_INT. (RO)

**BUS_MONITOR_CORE_0_AREA_DRAM0_0_WR_RAW** The raw interrupt status of BUS_MONITOR_CORE_0_AREA_DRAM0_0_WR_INT. (RO)

**BUS_MONITOR_CORE_0_AREA_DRAM0_1_RD_RAW** The raw interrupt status of BUS_MONITOR_CORE_0_AREA_DRAM0_1_RD_INT. (RO)

**BUS_MONITOR_CORE_0_AREA_DRAM0_1_WR_RAW** The raw interrupt status of BUS_MONITOR_CORE_0_AREA_DRAM0_1_WR_INT. (RO)

**BUS_MONITOR_CORE_0_AREA_PIF_0_RD_RAW** The raw interrupt status of BUS_MONITOR_CORE_0_AREA_PIF_0_RD_INT. (RO)

**BUS_MONITOR_CORE_0_AREA_PIF_0_WR_RAW** The raw interrupt status of BUS_MONITOR_CORE_0_AREA_PIF_0_WR_INT. (RO)

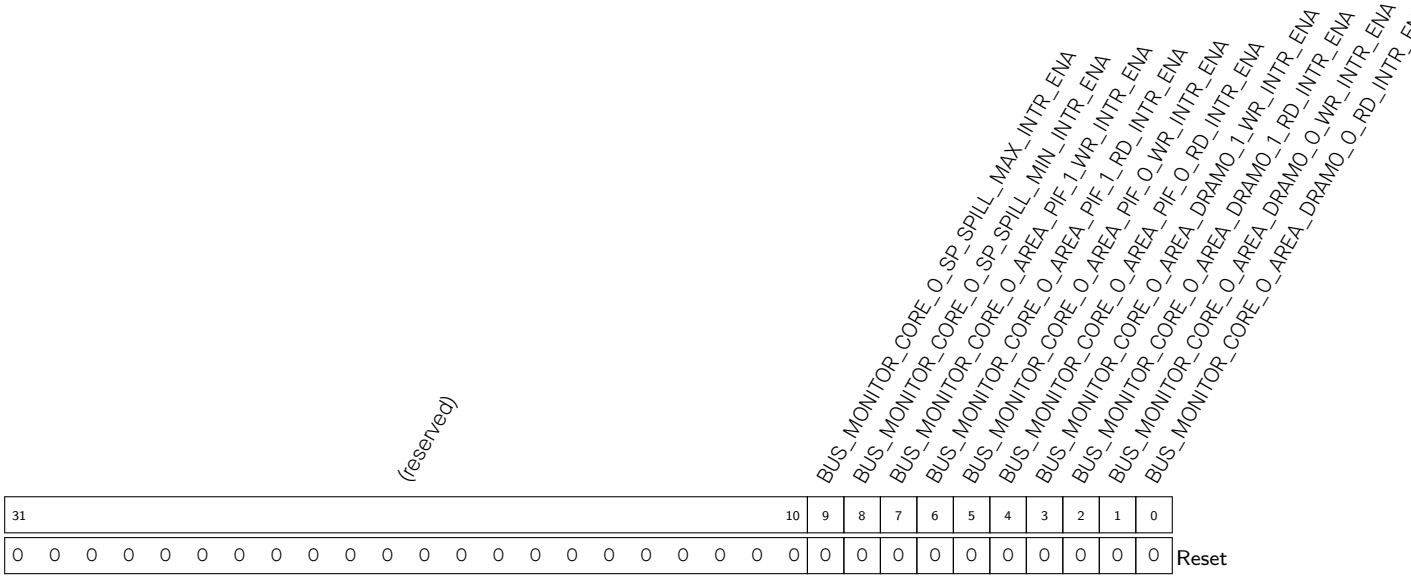
**BUS_MONITOR_CORE_0_AREA_PIF_1_RD_RAW** The raw interrupt status of BUS_MONITOR_CORE_0_AREA_PIF_1_RD_INT. (RO)

**BUS_MONITOR_CORE_0_AREA_PIF_1_WR_RAW** The raw interrupt status of BUS_MONITOR_CORE_0_AREA_PIF_1_WR_INT. (RO)

**BUS_MONITOR_CORE_0_SP_SPILL_MIN_RAW** The raw interrupt status of BUS_MONITOR_CORE_0_SP_SPILL_MIN_INT. (RO)

**BUS_MONITOR_CORE_0_SP_SPILL_MAX_RAW** The raw interrupt status of BUS_MONITOR_CORE_0_SP_SPILL_MAX_INT. (RO)

Register 19.31. BUS_MONITOR_CORE_0_INTR_ENA_REG (0x0008)




**BUS_MONITOR_CORE_0_AREA_DRAM0_0_RD_INTR_ENA** Write 1 to enable
BUS_MONITOR_CORE_0_AREA_DRAM0_0_RD_INT. (R/W)

**BUS_MONITOR_CORE_0_AREA_DRAM0_0_WR_INTR_ENA** Write 1 to enable
BUS_MONITOR_CORE_0_AREA_DRAM0_0_WR_INT. (R/W)

**BUS_MONITOR_CORE_0_AREA_DRAM0_1_RD_INTR_ENA** Write 1 to enable
BUS_MONITOR_CORE_0_AREA_DRAM0_1_RD_INT. (R/W)

**BUS_MONITOR_CORE_0_AREA_DRAM0_1_WR_INTR_ENA** Write 1 to enable
BUS_MONITOR_CORE_0_AREA_DRAM0_1_WR_INT. (R/W)

**BUS_MONITOR_CORE_0_AREA_PIF_0_RD_INTR_ENA** Write 1 to enable
BUS_MONITOR_CORE_0_AREA_PIF_0_RD_INT. (R/W)

**BUS_MONITOR_CORE_0_AREA_PIF_0_WR_INTR_ENA** Write 1 to enable
BUS_MONITOR_CORE_0_AREA_PIF_0_WR_INT. (R/W)

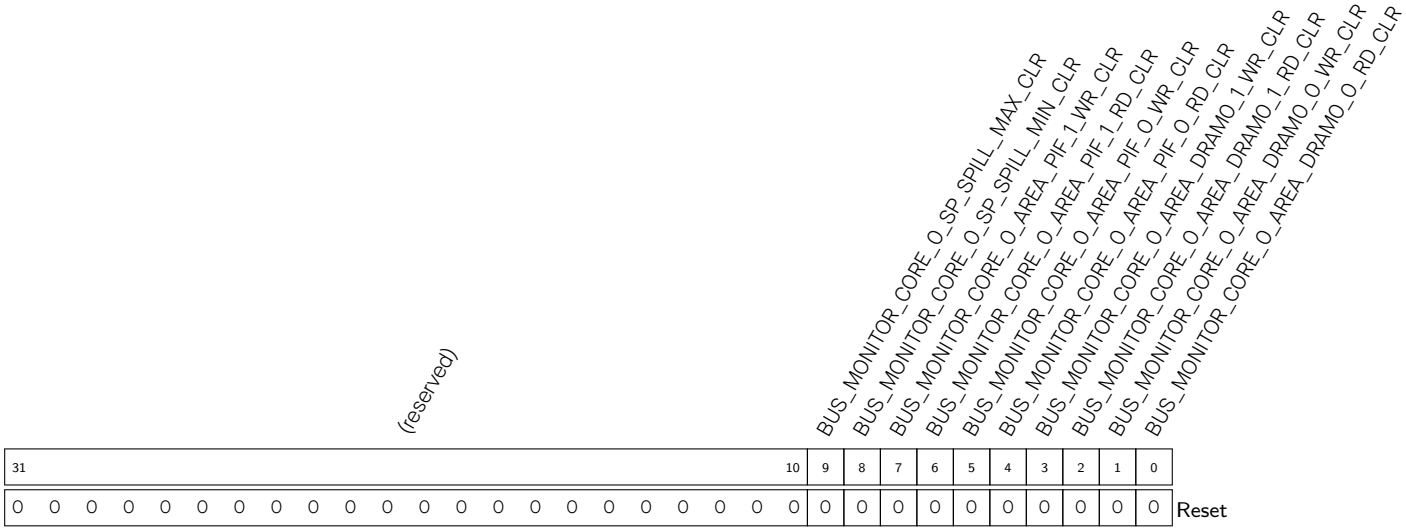
**BUS_MONITOR_CORE_0_AREA_PIF_1_RD_INTR_ENA** Write 1 to enable
BUS_MONITOR_CORE_0_AREA_PIF_1_RD_INT. (R/W)

**BUS_MONITOR_CORE_0_AREA_PIF_1_WR_INTR_ENA** Write 1 to enable
BUS_MONITOR_CORE_0_AREA_PIF_1_WR_INT. (R/W)

**BUS_MONITOR_CORE_0_SP_SPILL_MIN_INTR_ENA** Write 1 to enable
BUS_MONITOR_CORE_0_SP_SPILL_MIN_INT. (R/W)

**BUS_MONITOR_CORE_0_SP_SPILL_MAX_INTR_ENA** Write 1 to enable
BUS_MONITOR_CORE_0_SP_SPILL_MAX_INT. (R/W)

Register 19.32. BUS_MONITOR_CORE_0_INTR_CLR_REG (0x000C)




**BUS_MONITOR_CORE_0_AREA_DRAM0_0_RD_CLR** Write 1 to clear BUS_MONITOR_CORE_0_AREA_DRAM0_0_RD_INT. (WT)

**BUS_MONITOR_CORE_0_AREA_DRAM0_0_WR_CLR** Write 1 to clear BUS_MONITOR_CORE_0_AREA_DRAM0_0_WR_INT. (WT)

**BUS_MONITOR_CORE_0_AREA_DRAM0_1_RD_CLR** Write 1 to clear BUS_MONITOR_CORE_0_AREA_DRAM0_1_RD_INT. (WT)

**BUS_MONITOR_CORE_0_AREA_DRAM0_1_WR_CLR** Write 1 to clear BUS_MONITOR_CORE_0_AREA_DRAM0_1_WR_INT. (WT)

**BUS_MONITOR_CORE_0_AREA_PIF_0_RD_CLR** Write 1 to clear BUS_MONITOR_CORE_0_AREA_PIF_0_RD_INT. (WT)

**BUS_MONITOR_CORE_0_AREA_PIF_0_WR_CLR** Write 1 to clear BUS_MONITOR_CORE_0_AREA_PIF_0_WR_INT. (WT)

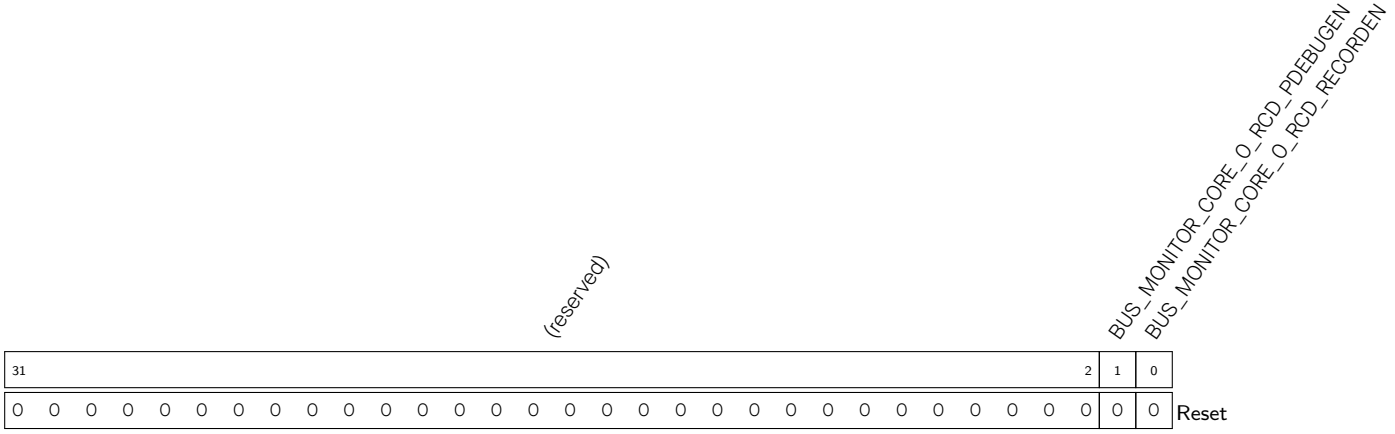
**BUS_MONITOR_CORE_0_AREA_PIF_1_RD_CLR** Write 1 to clear BUS_MONITOR_CORE_0_AREA_PIF_1_RD_INT. (WT)

**BUS_MONITOR_CORE_0_AREA_PIF_1_WR_CLR** Write 1 to clear BUS_MONITOR_CORE_0_AREA_PIF_1_WR_INT. (WT)

**BUS_MONITOR_CORE_0_SP_SPILL_MIN_CLR** Write 1 to clear BUS_MONITOR_CORE_0_SP_SPILL_MIN_INT. (WT)

**BUS_MONITOR_CORE_0_SP_SPILL_MAX_CLR** Write 1 to clear BUS_MONITOR_CORE_0_SP_SPILL_MAX_INT. (WT)

Register 19.33. BUS_MONITOR_CORE_0_RCD_EN_REG (0x0044)




**BUS_MONITOR_CORE_0_RCD_RECORDEN** Configures whether to enable PC and SP logging.

0: Disable

1: BUS_MONITOR_CORE_0_RCD_PDEBUGPC_REG starts to record PC in real time, BUS_MONITOR_CORE_0_RCD_PDEBUGSP_REG starts to record SP in real time

(R/W)

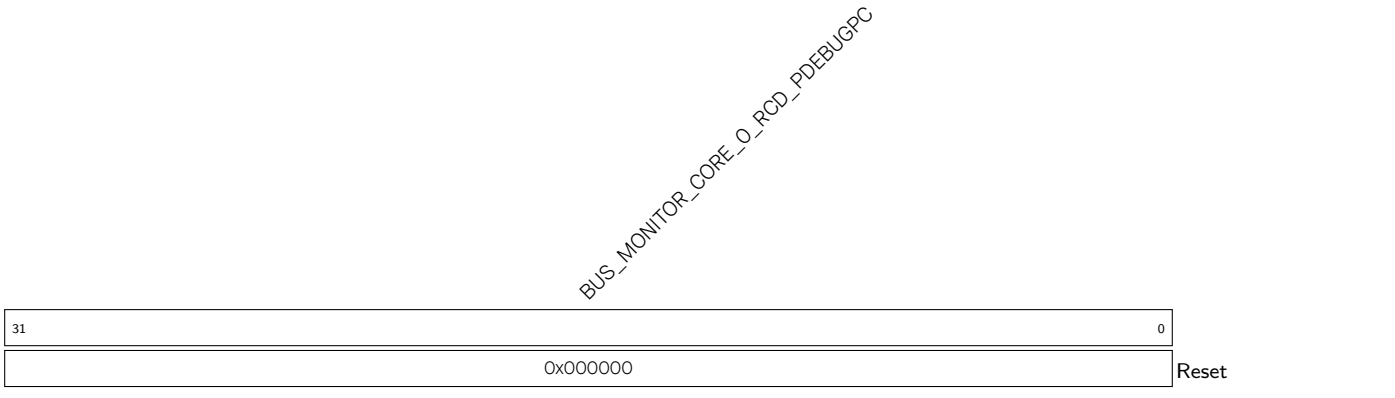
**BUS_MONITOR_CORE_0_RCD_PDEBUGEN** Configures whether to enable HP CPU debugging.

0: Disable

1: HP CPU outputs PC

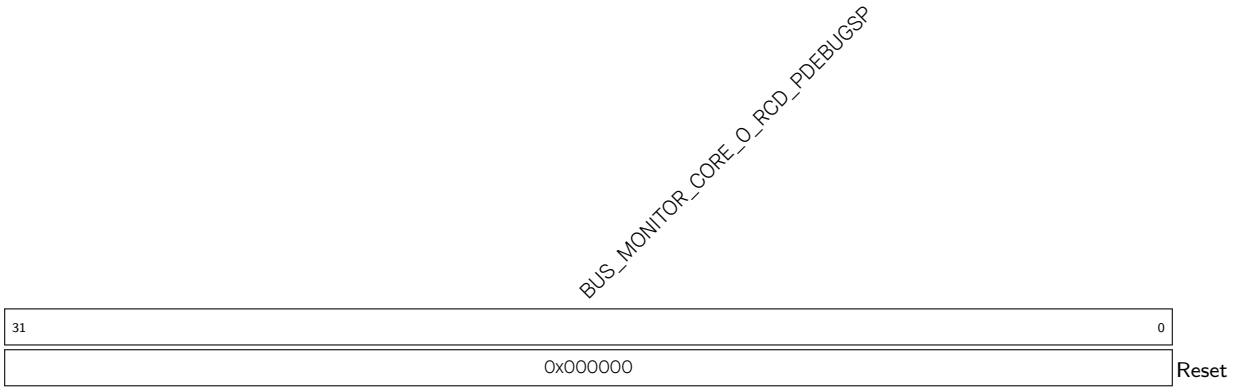
(R/W)

Register 19.34. BUS_MONITOR_CORE_0_RCD_PDEBUGPC_REG (0x0048)



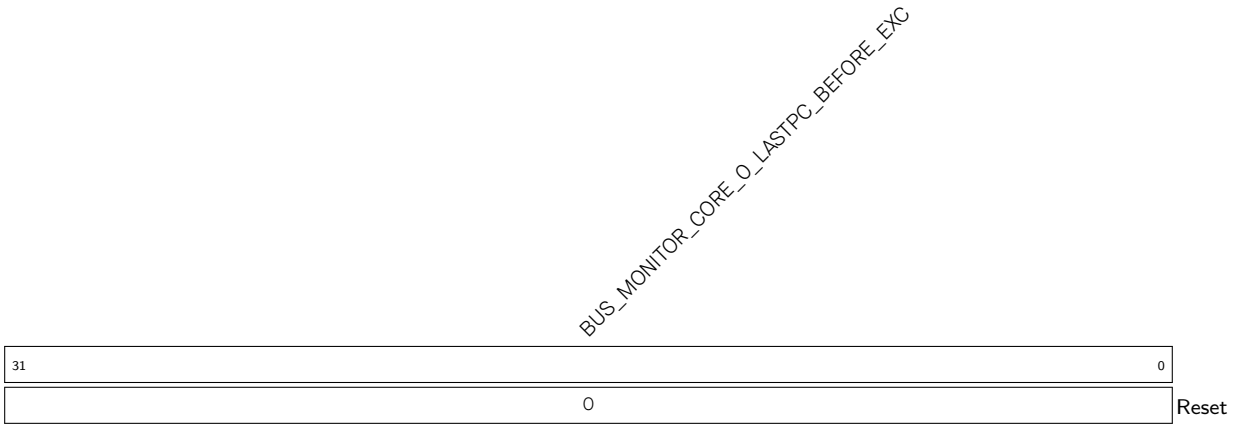

**BUS_MONITOR_CORE_0_RCD_PDEBUGPC** Represents the PC value at HP CPU reset. (RO)

Register 19.35. BUS_MONITOR_CORE_0_RCD_PDEBUGSP_REG (0x004C)



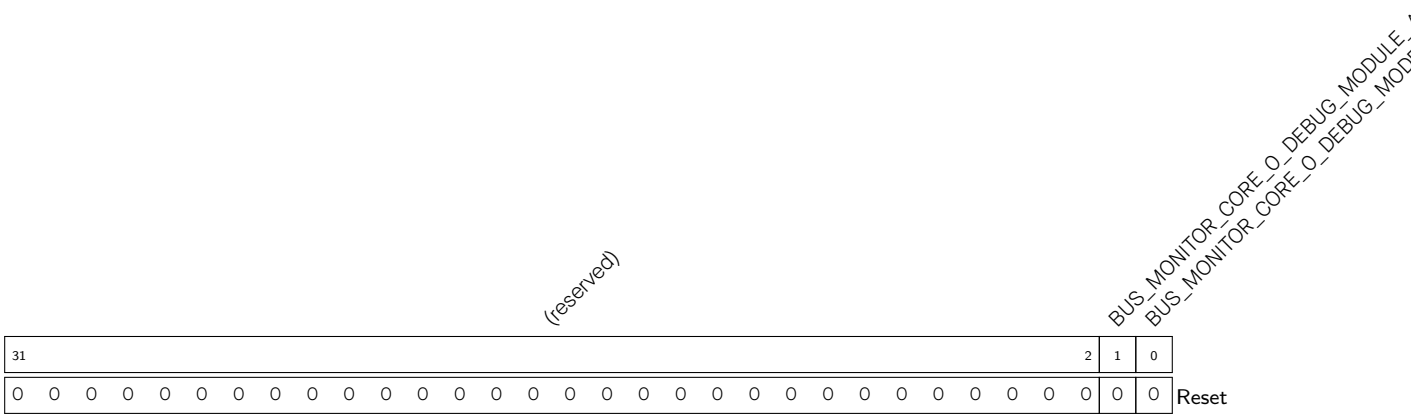

**BUS_MONITOR_CORE_0_RCD_PDEBUGSP** Represents SP. (RO)

Register 19.36. BUS_MONITOR_CORE_0_LASTPC_BEFORE_EXCEPTION_REG (0x0070)




**BUS_MONITOR_CORE_0_LASTPC_BEFORE_EXC** Represents the PC of the last command before the HP CPU enters exception. (RO)

Register 19.37. BUS_MONITOR_CORE_0_DEBUG_MODE_REG (0x0074)




**BUS_MONITOR_CORE_0_DEBUG_MODE** Represents whether RISC-V CPU (HP CPU) is in debugging mode.

1: In debugging mode

0: Not in debugging mode

(RO)

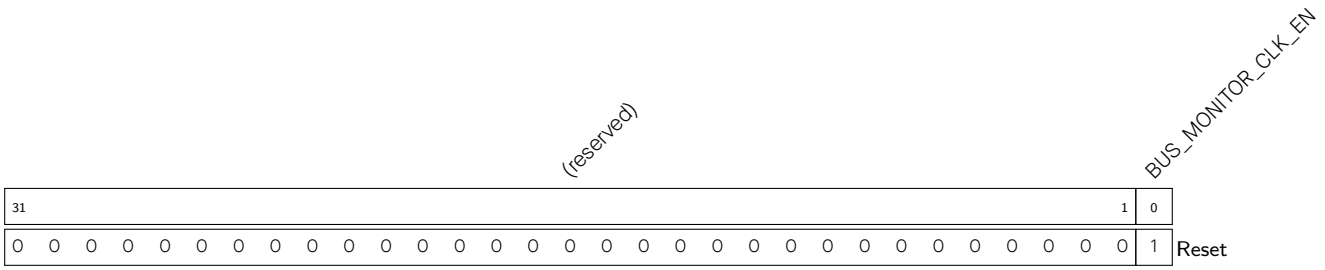
**BUS_MONITOR_CORE_0_DEBUG_MODULE_ACTIVE** Represents the status of the RISC-V CPU (HP CPU) debug module.

1: Active status

Other: Inactive status

(RO)

Register 19.38. BUS_MONITOR_CLOCK_GATE_REG (0x0108)



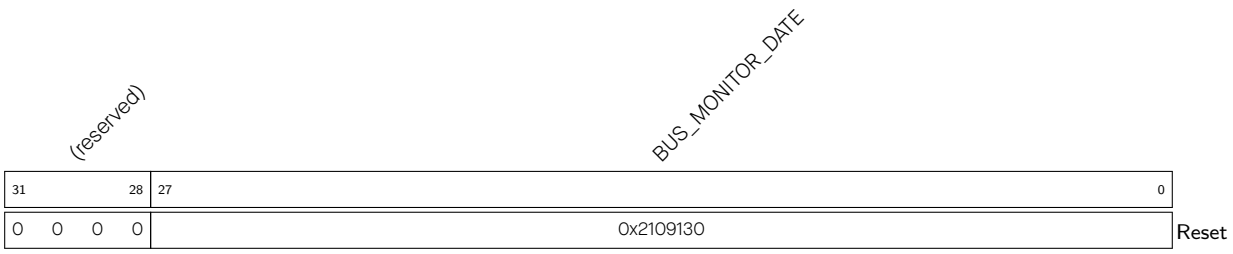

**BUS_MONITOR_CLK_EN** Configures whether to enable the register clock gating.

0: Disable

1: Enable

(R/W)

Register 19.39. BUS_MONITOR_DATE_REG (0x03FC)




**BUS_MONITOR_DATE** Version control register. (R/W)

# Chapter 20

# Power Supply Detector

## 20.1 Overview

ESP32-C61 has a power supply detector that can monitor the voltage on the power pins related to the on-chip clock and digital circuits. The power supply detector includes a brown-out detector and four voltage glitch detectors. The brown-out detector prevents the SoC from brown-out resets, while the voltage glitch detectors protect the SoC against voltage glitch attacks. The power detector operates in the always-on power domain, allowing it to monitor voltage at any time.

## 20.2 Features

- Brown-out detector supports two brown-out detection modes (Mode 0 and Mode 1)
  - Mode 0 supports:
    * interrupt generation
    * RF circuits power-down
    * flash suspend triggering
    * system reset
  - Mode 1 supports:
    * system reset until the voltage returns to normal
- Voltage glitch detector can detect voltage glitches lasting 50 ns or longer and trigger a system reset.

## 20.3 Functional Description

### 20.3.1 Architecture

Figure 20.3-1 shows the architecture of the power supply detector. As shown, the power supply detector consists of one brown-out detector and four voltage glitch detectors.

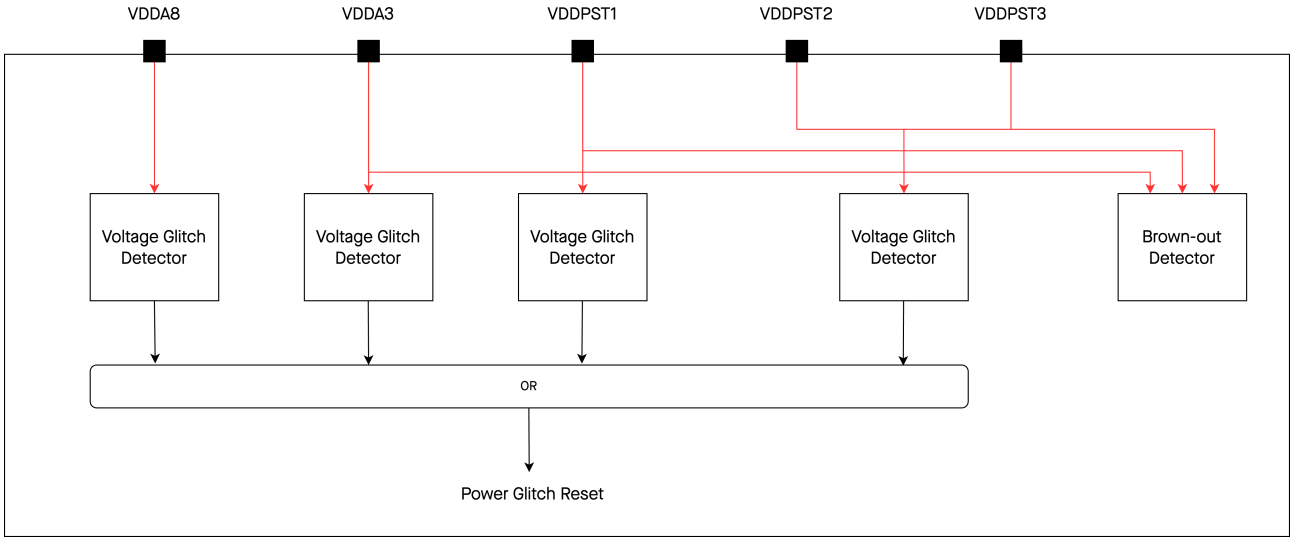



Figure 20.3-1. Architecture of Power Supply Detector

## 20.3.2 Brown-out Detector

The brown-out detector checks the voltage of pins VDDA3, VDDPST1, VDDPST2, and VDDPST3, about every 280 $\mu$s. When the voltage on these pins falls below a predefined threshold of 2.7 V (default setting), the detector activates a signal to power down the power-hungry RF circuits. This action provides additional time for the digital system to save and transfer important data. The brown-out detector consumes very little power and remains active as long as the chip is powered on.

LP_ANA_BOD_MODE0_INT_RAW indicates the detection result from the brown-out detector. By default, this register reads a value of 0. It changes to 1 when the voltage on the monitored pin falls below a predefined threshold.

When a brown-out signal is detected, the brown-out detector handles it in one of the following two modes (Mode 1 is the default):

- Mode 0:
  - Mode 0 is enabled by setting the `bod_mode0_en` (LP_ANA_BOD_MODE0_INTR_ENA) signal.
  - The brown-out detector triggers an interrupt when the counter counts to the thresholds predefined in Int Comparer (LP_ANA_BOD_MODE0_INTR_WAIT). If LP_ANA_BOD_MODE0_CLOSE_FLASH_ENA is set, flash suspend will be triggered; if LP_ANA_BOD_MODE0_PD_RF_ENA is set, the RF module will be powered down.
  - The brown-out detector resets the chip based on the configuration of `bod_mode0_rst_sel` (LP_ANA_BOD_MODE0_RESET_SEL) when the brown-out counter counts to the thresholds predefined in Rst Comparer (LP_ANA_BOD_MODE0_RESET_WAIT). The reset is enabled by `bod_mode0_rst_en` (LP_ANA_BOD_MODE0_RESET_ENA).
- Mode 1: Resets the system directly.

The brown-out reset workflow is illustrated in the diagram below.

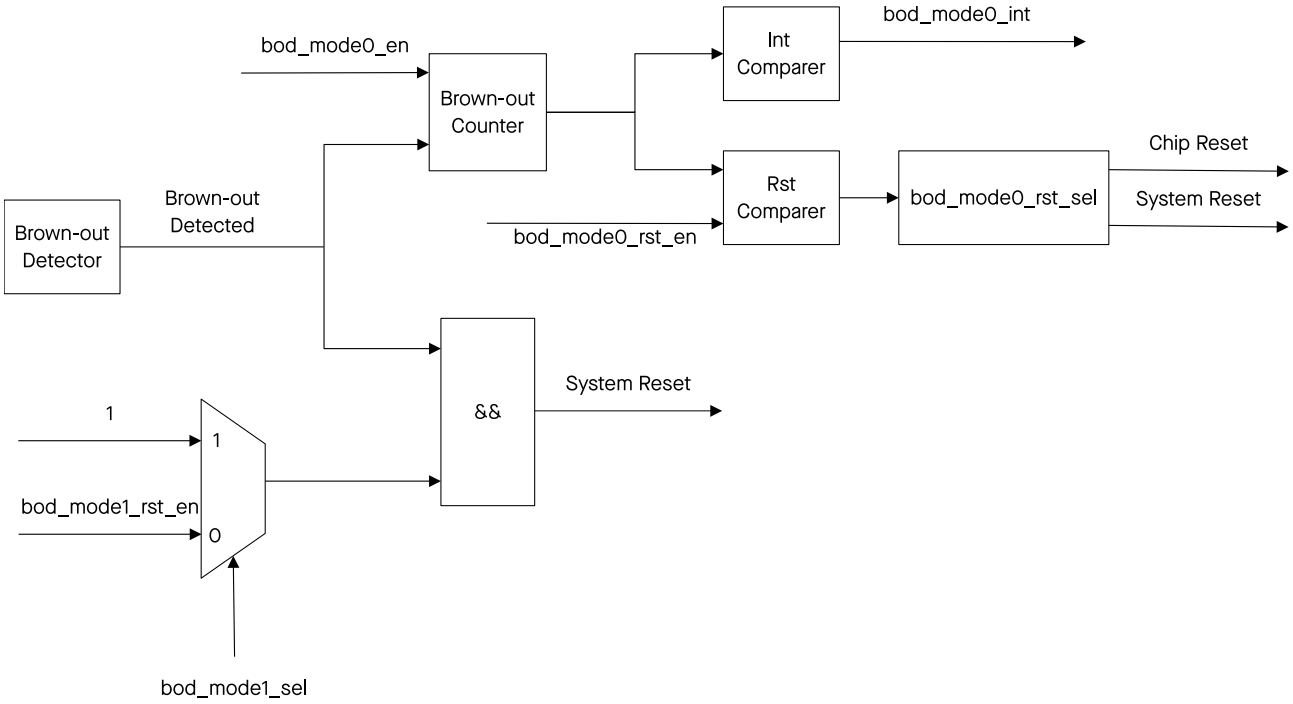



Figure 20.3-2. Brown-out Reset Workflow

Registers for controlling related signals are described below.

- `bod_mode0_en`: LP_ANA_BOD_MODE0_INTR_ENA
- `bod_mode0_rst_en`: LP_ANA_BOD_MODE0_RESET_ENA
- `bod_mode0_rst_sel`: LP_ANA_BOD_MODE0_RESET_SEL configures the reset type:
  - 0: chip reset
  - 1: system reset

  For more information regarding chip reset and system reset, please refer to Chapter 8 *Reset and Clock*.
- `bod_mode1_sel`: The first bit of LP_ANA_ANA_FIB_ENA.
- `bod_mode1_rst_en`: LP_ANA_BOD_MODE1_RESET_ENA

### 20.3.3 Voltage Glitch Detectors

Figure 20.3-3 shows the structure of the voltage glitch detectors.

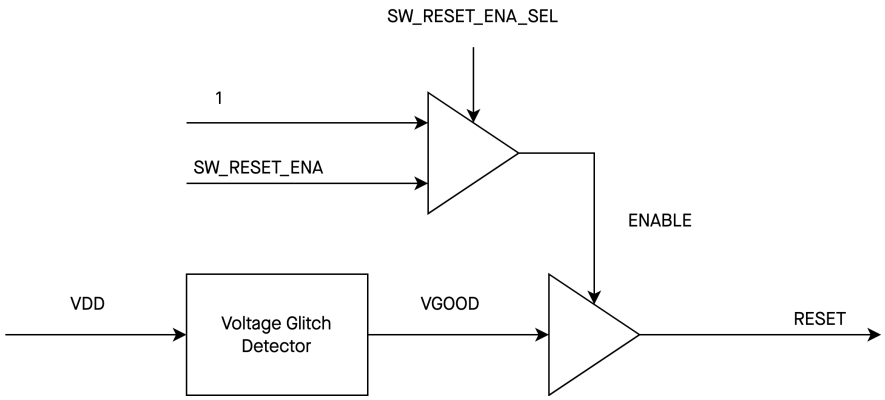



Figure 20.3-3. Structure of Voltage Glitch Detectors

The ESP32-C61 has four voltage glitch detectors, each of which can detect glitches longer than 50 ns on the corresponding power supply pins, including VDDPST1, VDDPST2, VDDPST3, VDDA8, and VDDA3. VDDPST2 and VDDPST3 are connected and thus share one voltage glitch detector. All resets generated by the detectors are combined into one voltage glitch reset.

The voltage glitch detectors' enable control is managed by SW_RESET_ENA_SEL. When set to 1, it is forcibly enabled by hardware, and when set to 0, it is controlled by the software register SW_RESET_ENA. The configuration for detecting different power pins is shown in the table below:

Table 20.3-1. Configuration for Detecting Power Pins

| VDD Pins | SW_RESET_ENA | SW_RESET_ENA_SEL |
|---|---|---|
| VDDPST2/3 | LP_ANA_PWR_GLITCH_RESET_ENA[0] | LP_ANA_ANA_FIB_ENA[2] |
| VDDPST1 | LP_ANA_PWR_GLITCH_RESET_ENA[1] | LP_ANA_ANA_FIB_ENA[3] |
| VDDA3 | LP_ANA_PWR_GLITCH_RESET_ENA[2] | LP_ANA_ANA_FIB_ENA[4] |
| VDDA8 | LP_ANA_PWR_GLITCH_RESET_ENA[3] | LP_ANA_ANA_FIB_ENA[5] |

## 20.4 Interrupts

ESP32-C61's power supply detector can generate the following interrupt signals that will be sent to the Interrupt Matrix.

- LP_ANA_INTR (only for HP CPU)
- LP_ANA_LP_INTR (only for LP CPU)

The interrupt signals are generated by the power supply detector's internal interrupt sources which are listed in Table 20.4-1 with their trigger conditions.

Table 20.4-1. Power Supply Detector's Internal Interrupt Source

| Internal Interrupt Source | Trigger Condition | Interrupt Signal |
|---|---|---|
| LP_ANA_BOD_MODE0_INT | Triggered when the brown-out detector detects that the voltage is below the threshold. | LP_ANA_INTR |
| LP_ANA_BOD_MODE0_LP_INT | Triggered when the brown-out detector detects that the voltage is below the threshold. | LP_ANA_LP_INTR |

**Note:**
For definitions of *interrupt*, *interrupt signal*, *interrupt source*, and their correlations, please refer to Chapter 10 *Interrupt Matrix* > Section 10.2 *Interrupt Terminology in ESP32-C61*.

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 20.5 *Register Summary*.

## 20.5 Register Summary

The addresses in this section are relative to the Power Supply Detector base address provided in Table 5.3-2 in Chapter 5 *System and Memory*.

The abbreviations given in Column **Access** are explained in Section *Access Types for Registers*.

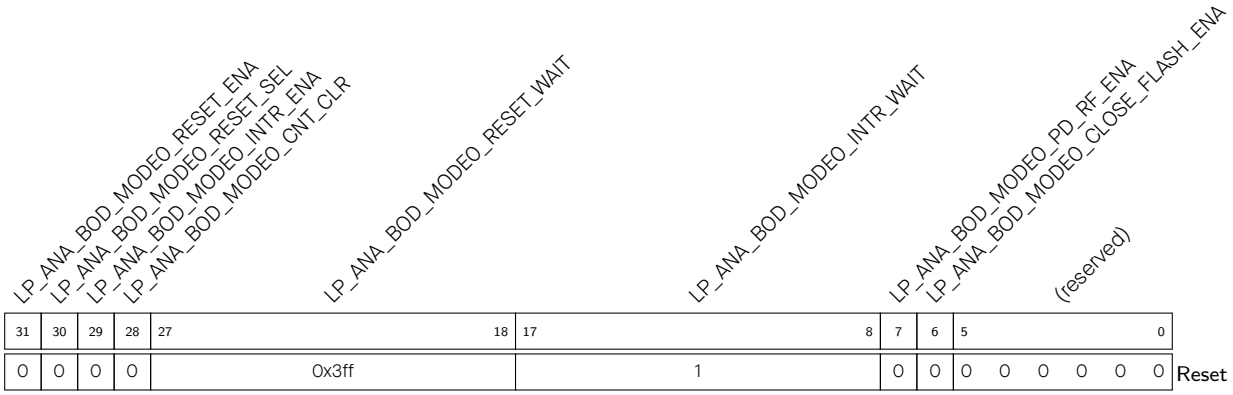
| Name | Description | Address | Access |
|---|---|---|---|
| **Configuration Registers** | | | |
| LP_ANA_BOD_MODE0_CNTL_REG | Brownout detector mode 0 configuration register | 0x0000 | R/W |
| LP_ANA_BOD_MODE1_CNTL_REG | Brownout detector mode 1 configuration register | 0x0004 | R/W |
| LP_ANA_POWER_GLITCH_CNTL_REG | Voltage glitch configuration register | 0x0008 | R/W |
| LP_ANA_FIB_ENABLE_REG | Voltage glitch detectors' enable control register | 0x000C | R/W |
| LP_ANA_INT_RAW_REG | LP_ANA_BOD_MODE0_INT raw interrupt | 0x0010 | R/WTC/SS |
| LP_ANA_INT_ST_REG | LP_ANA_BOD_MODE0_INT state interrupt | 0x0014 | RO |
| LP_ANA_INT_ENA_REG | LP_ANA_BOD_MODE0_INT enable register | 0x0018 | R/W |
| LP_ANA_INT_CLR_REG | LP_ANA_BOD_MODE0_INT clear register | 0x001C | WT |
| LP_ANA_LP_INT_RAW_REG | LP_ANA_BOD_MODE0_LP_INT raw interrupt | 0x0020 | R/WTC/SS |
| LP_ANA_LP_INT_ST_REG | LP_ANA_BOD_MODE0_LP_INT state interrupt | 0x0024 | RO |
| LP_ANA_LP_INT_ENA_REG | LP_ANA_BOD_MODE0_LP_INT enable register | 0x0028 | R/W |
| LP_ANA_LP_INT_CLR_REG | LP_ANA_BOD_MODE0_LP_INT clear register | 0x002C | WT |
| **Version Control Registers** | | | |
| LP_ANA_DATE_REG | Version control register | 0x03FC | R/W |

## 20.6 Registers

The addresses in this section are relative to the Power Supply Detector base address provided in Table 5.3-2 in Chapter 5 *System and Memory*.

For how to program reserved fields, please refer to Section *Programming Reserved Register Field*.

Register 20.1. LP_ANA_BOD_MODE0_CNTL_REG (0x0000)




**LP_ANA_BOD_MODE0_CLOSE_FLASH_ENA** Configures whether to enable the brown-out detector to trigger flash suspend.
0: Disable
1: Enable
(R/W)

**LP_ANA_BOD_MODE0_PD_RF_ENA** Configures whether to enable the brown-out detector to power down the RF module.
0: Disable
1: Enable
(R/W)

**LP_ANA_BOD_MODE0_INTR_WAIT** Configures the time to generate an interrupt after the brown-out signal is valid. The unit is LP_FAST_CLK cycles. (R/W)

**LP_ANA_BOD_MODE0_RESET_WAIT** Configures the time to generate a reset after the brown-out signal is valid. The unit is LP_FAST_CLK cycles. (R/W)

**LP_ANA_BOD_MODE0_CNT_CLR** Configures whether to clear the count value of the brown-out detector.
0: Do not clear
1: Clear
(R/W)

**LP_ANA_BOD_MODE0_INTR_ENA** Enables the interrupts for the brown-out detector mode 0. LP_ANA_BOD_MODE0_INT_RAW and LP_ANA_BOD_MODE0_LP_INT_RAW are valid only when this field is set to 1.
0: Disable
1: Enable
(R/W)

**LP_ANA_BOD_MODE0_RESET_SEL** Configures the reset type when the brown-out detector is triggered.
0: Chip reset
1: System reset
(R/W)

**Continued on the next page...**

Register 20.1. LP_ANA_BOD_MODE0_CNTL_REG (0x0000)

Continued from the previous page...

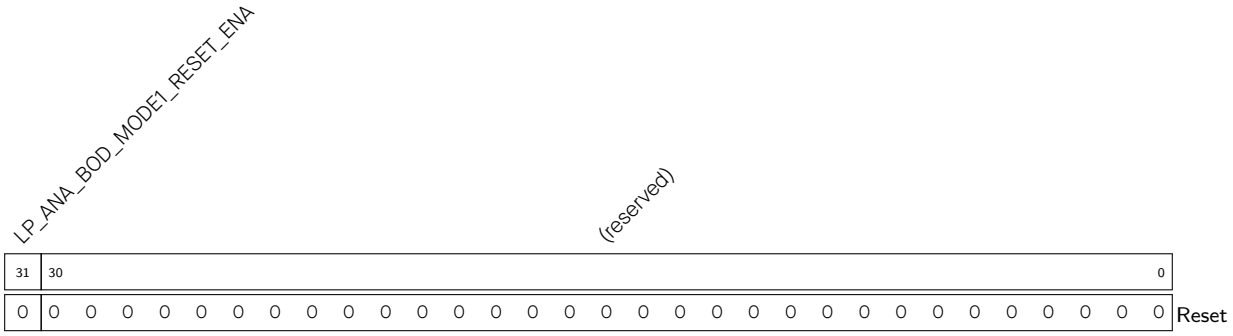
**LP_ANA_BOD_MODE0_RESET_ENA** Configures whether to enable reset for the brown-out detector.

0: Disable

1: Enable

(R/W)

Register 20.2. LP_ANA_BOD_MODE1_CNTL_REG (0x0004)



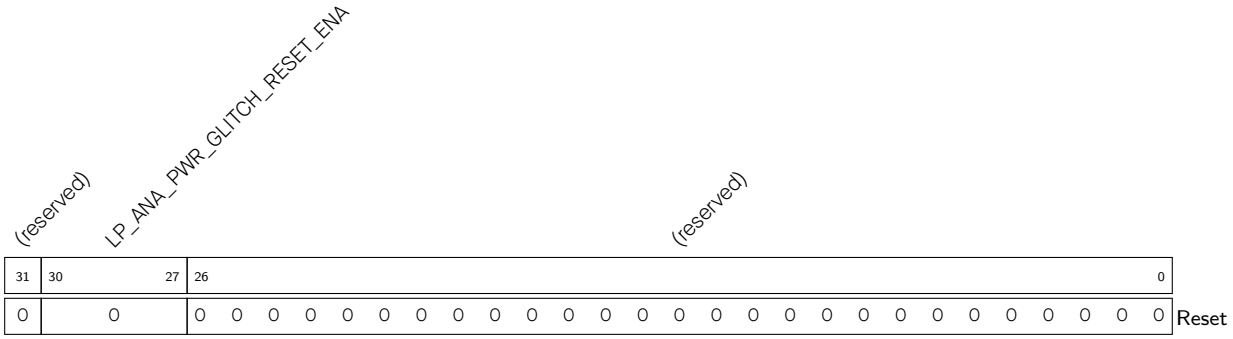

**LP_ANA_BOD_MODE1_RESET_ENA** Configures whether to enable brown-out detector mode 1.

0: Disable

1: Enable

(R/W)

Register 20.3. LP_ANA_POWER_GLITCH_CNTL_REG (0x0008)



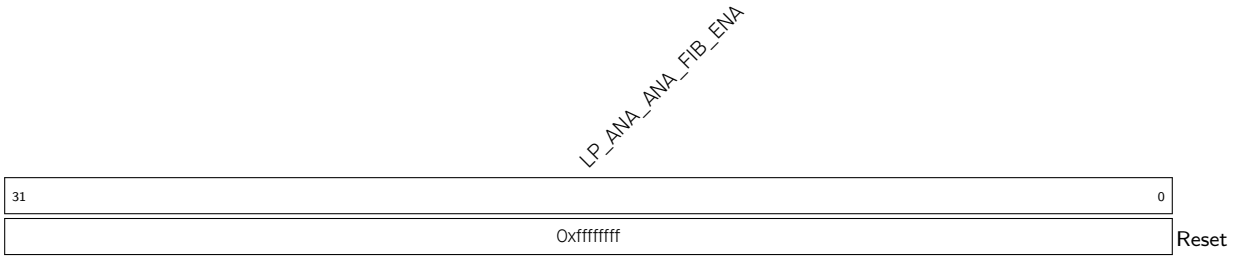

**LP_ANA_PWR_GLITCH_RESET_ENA** Configures whether to enable the voltage glitch detectors. Bit0, bit1, bit2, bit3 correspond to VDDPST2/3, VDDPST1, VDDA3, and VDDA8, respectively.

0: Disable

1: Enable

(R/W)

Register 20.4. LP_ANA_FIB_ENABLE_REG (0x000C)



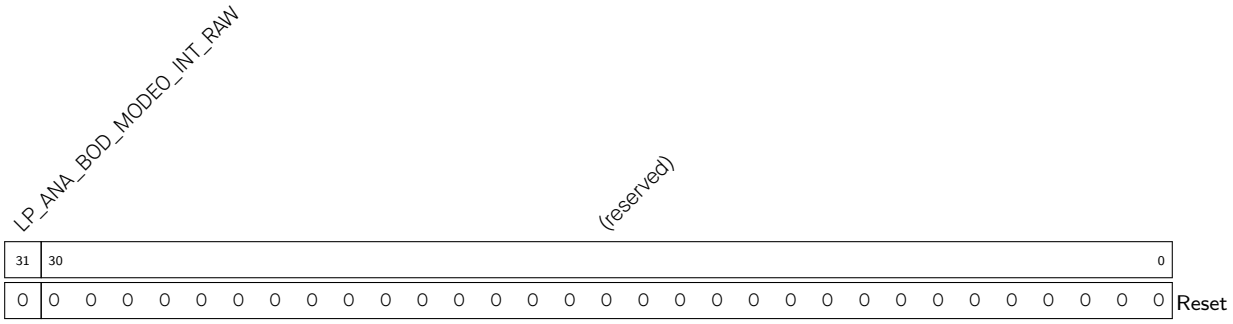

**LP_ANA_ANA_FIB_ENA** Controls the enable of the voltage glitch detectors. Bit2, bit3, bit4, bit5 correspond to VDDPST2/3, VDDPST1, VDDA3, and VDDA8, respectively.

0: Controlled by LP_ANA_PWR_GLITCH_RESET_ENA

1: Forcibly enabled by hardware

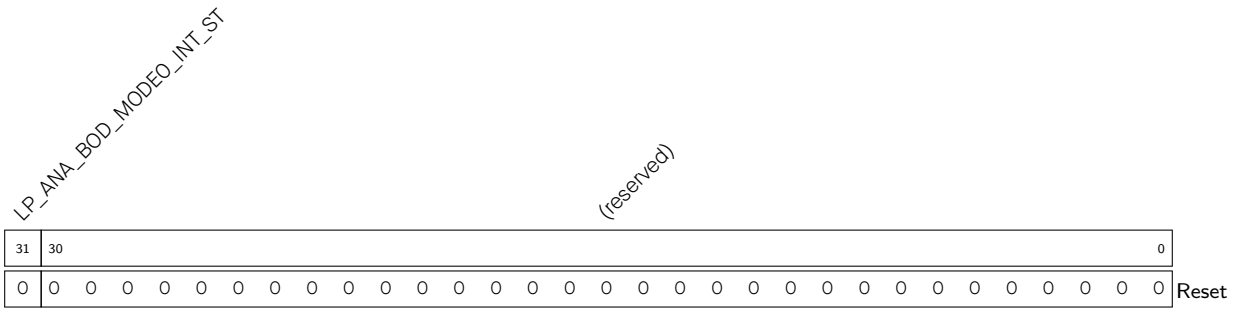
(R/W)

Register 20.5. LP_ANA_INT_RAW_REG (0x0010)



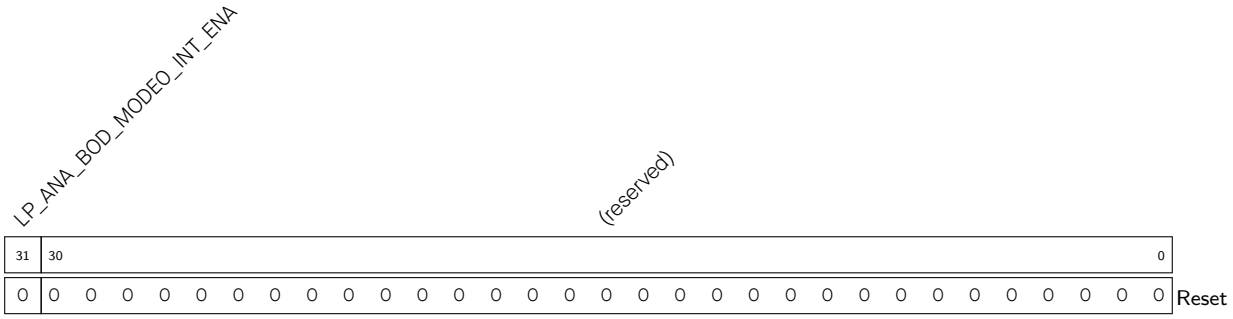

**LP_ANA_BOD_MODE0_INT_RAW** The raw interrupt status of LP_ANA_BOD_MODE0_INT. (R/WTC/SS)

Register 20.6. LP_ANA_INT_ST_REG (0x0014)



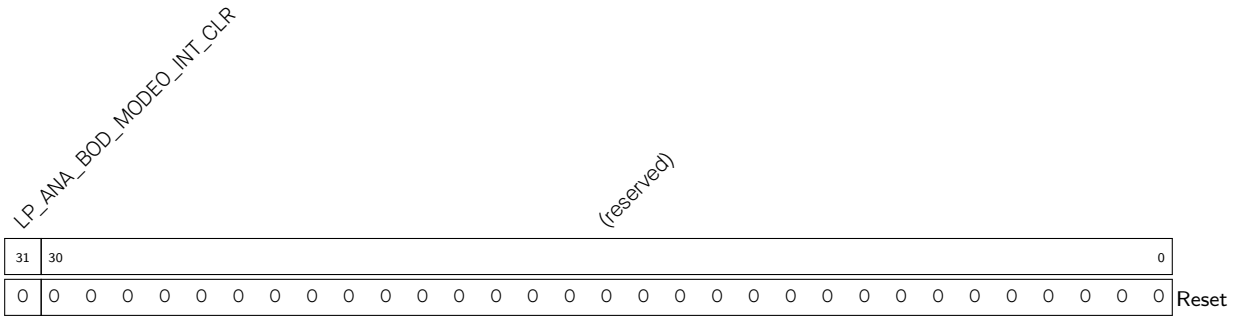

**LP_ANA_BOD_MODE0_INT_ST** The masked interrupt status of LP_ANA_BOD_MODE0_INT. (RO)

Register 20.7. LP_ANA_INT_ENA_REG (0x0018)



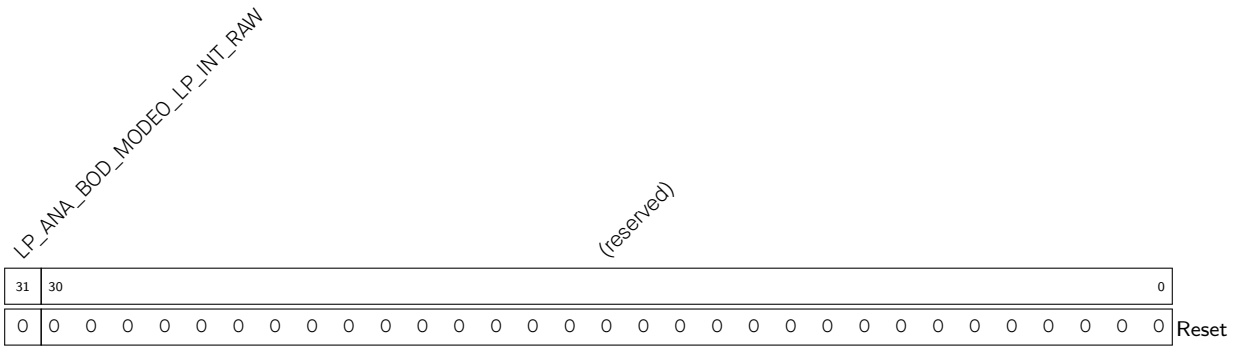

**LP_ANA_BOD_MODE0_INT_ENA** Write 1 to enable LP_ANA_BOD_MODE0_INT. (R/W)

Register 20.8. LP_ANA_INT_CLR_REG (0x001C)



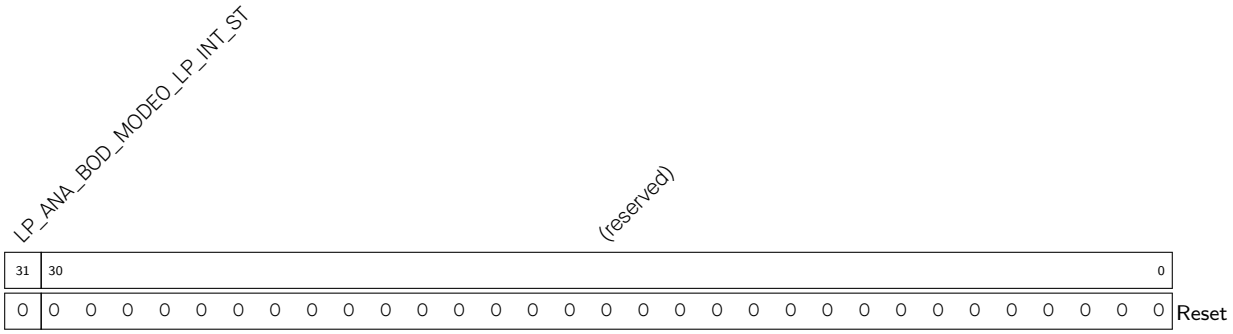

**LP_ANA_BOD_MODE0_INT_CLR** Write 1 to clear LP_ANA_BOD_MODE0_INT. (WT)

Register 20.9. LP_ANA_LP_INT_RAW_REG (0x0020)



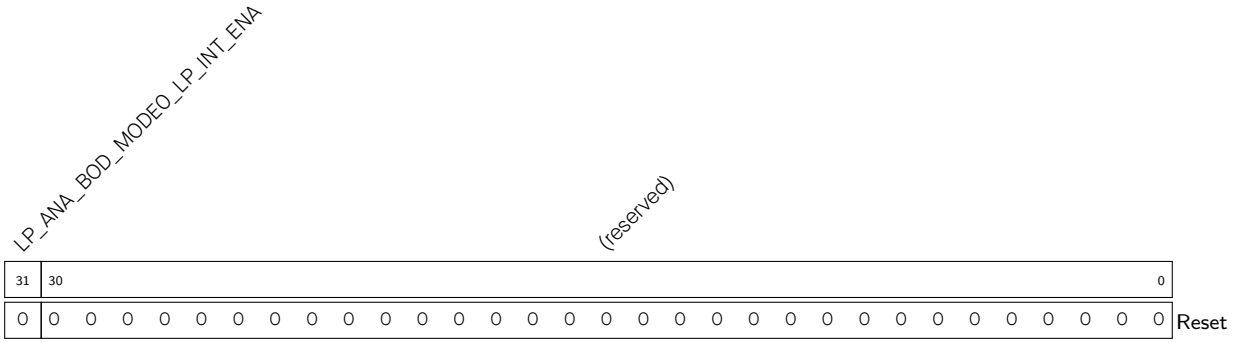

**LP_ANA_BOD_MODE0_LP_INT_RAW** The raw interrupt status of LP_ANA_BOD_MODE0_LP_INT. (R/WTC/SS)

Register 20.10. LP_ANA_LP_INT_ST_REG (0x0024)



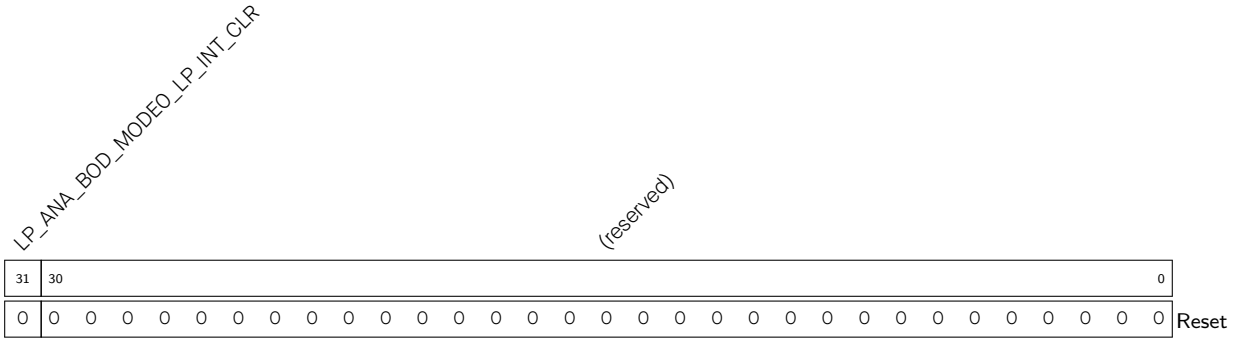

**LP_ANA_BOD_MODE0_LP_INT_ST** The masked interrupt status of LP_ANA_BOD_MODE0_LP_INT. (RO)

Register 20.11. LP_ANA_LP_INT_ENA_REG (0x0028)



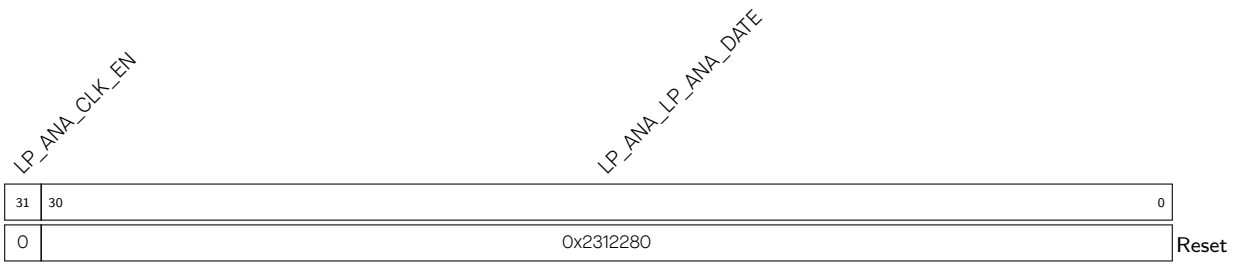

**LP_ANA_BOD_MODE0_LP_INT_ENA** Write 1 to enable LP_ANA_BOD_MODE0_LP_INT. (R/W)

Register 20.12. LP_ANA_LP_INT_CLR_REG (0x002C)




**LP_ANA_BOD_MODE0_LP_INT_CLR** Write 1 to clear LP_ANA_BOD_MODE0_LP_INT. (WT)

Register 20.13. LP_ANA_DATE_REG (0x03FC)




**LP_ANA_LP_ANA_DATE** Version control register. (R/W)

**LP_ANA_CLK_EN** Configures whether to force enable register clock.

0: Automatic clock gating

1: Force enable register clock

The configuration of this field does not effect the access of registers.

(R/W)

# Part IV

# Cryptography/Security Component

Dedicated to security features, this part explores cryptographic accelerators like ECC and SHA. It also covers random number generation and encryption/decryption algorithms, showcasing the SoC's capabilities in cryptography and secure data processing.

# Chapter 21

# ECC Accelerator (ECC)

## 21.1 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-C61's ECC accelerator can complete various calculations based on different elliptic curves, thus accelerating the ECC algorithm and ECC-derived algorithms such as ECDSA.

## 21.2 Feature List

ESP32-C61's ECC accelerator has the following features:

- 2 different elliptic curves, namely P-192 and P-256 defined in FIPS 186-5
- 11 working modes
- Enhanced anti-attack performance
- Interrupt upon completion of calculation

## 21.3 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.

### 21.3.1 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.

### 21.3.2 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:

$$X = x$$

$$Y = y$$

$$Z = 1$$

### 21.3.3 Memory Blocks

ECC's memory blocks store the input and output data of the ECC operation.

**Table 21.3-1. ECC Accelerator Memory Blocks**

| Memory Block[1] | Size (byte) | Starting Address[2] | Ending Address [2] | 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 |

[1] The memory blocks store different types of data in different working modes. Refer to Section 21.4.2 for more details.

[2] Address offset related to the ECC accelerator base address is provided in Table 5.3-2 in Chapter 5 *System and Memory*.

### 21.3.4 Data and Data Block

ESP32-C61's ECC operates on 256-bit data. The 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:

$$D[255:0] = D[7][31:0], D[6][31:0], D[5][31:0], D[4][31:0], D[3][31:0], D[2][31:0], D[1][31:0], D[0][31:0]$$

### 21.3.5 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 + $4 \times n$" successively:

- write $D[0]$ to "starting address"
- write $D[1]$ to "starting address + 4"
- $\cdots$
- write $D[7]$ to "starting address + 28"

**Note:**
When the key size of 192 bits is used, append 0 before 192 bits of data to ensure 256-bit data is written.

### 21.3.6 Reading Data

Reading 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, reading 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 the key size of 192 bits is used, only read the low 192 bits (6 blocks) of data.

### 21.3.7 Standard Calculation and Jacobian Calculation

ESP32-C61'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.

## 21.4 Function Description

### 21.4.1 Key Size

ESP32-C61's ECC supports acceleration based on two key sizes, each corresponding to an elliptic curve. By configuring the ECC_MULT_KEY_LENGTH field, users can select the desired key size. For details, see Table 21.4-1 below.

Table 21.4-1. ECC Accelerator Key Size Selection

| ECC_MULT_KEY_LENGTH | Elliptic Curves* |
|---|---|
| 0 | FIPS P-192 |
| 1 | FIPS P-256 |

* See definition of FIPS P-192 and P-256 in FIPS 186-5.

## 21.4.2 Working Modes

ESP32-C61'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 21.4-2.

Table 21.4-2. 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.

### 21.4.2.1 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.

#### 21.4.2.2 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.

#### 21.4.2.3 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.

#### 21.4.2.4 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.

#### 21.4.2.5 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.

#### 21.4.2.6 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.

#### 21.4.2.7 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.

#### 21.4.2.8 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_KEY_LENGTH 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.

#### 21.4.2.9 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_KEY_LENGTH 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.

#### 21.4.2.10 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_KEY_LENGTH 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.

#### 21.4.2.11 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_KEY_LENGTH 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.

### 21.4.3 Enhancing Anti-Attack Performance

When performing different types of point multiplication calculations, i.e., mode 0, 3, 4, and 7, ESP32-C61'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.

## 21.5 Clock and Reset

ESP32-C61'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 8 *Reset and Clock* for more information). Users should enable the ECC clock by setting PCR_ECC_CLK_EN and release the ECC reset by clearing PCR_ECC_RST_EN before starting the ECC accelerator. Besides, due to resource reuse between cryptography accelerator modules, users also need to additionally clear the PCR_ECDSA_RST_EN bit.

## 21.6 Interrupts

ESP32-C61'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 21.8 *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 10 *Interrupt Matrix* > Section 10.2 *Interrupt Terminology in ESP32-C61*.

## 21.7 Programming Procedures

The programming procedure for configuring ECC is described below:

1. Configure the ECC clock and reset. Refer to Section 21.5 for detailed information.
2. Select the key size and working mode as described in Section 21.4.
3. Enable the ECC_MULT_CALC_DONE_INT interrupt as described in Section 21.6.
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 21.4.

## 21.8 Register Summary

The addresses in this section are relative to ECC Accelerator base address provided in Table 5.3-2 in Chapter 5 *System and Memory*.

The abbreviations given in Column **Access** are explained in Section *Access Types for Registers*.

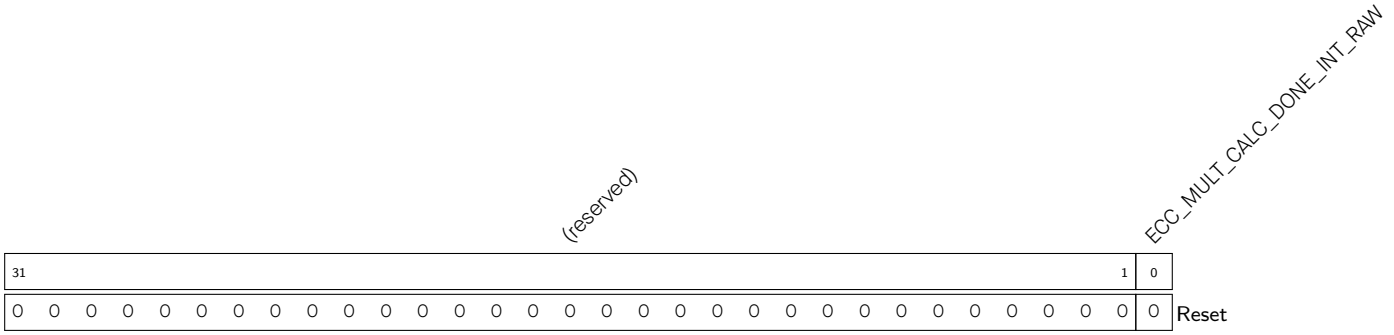
| Name | Description | Address | Access |
|---|---|---|---|
| **Interrupt Registers** | | | |
| ECC_MULT_INT_RAW_REG | ECC raw interrupt status register | 0x000C | R/SS/WTC |
| ECC_MULT_INT_ST_REG | ECC masked interrupt status register | 0x0010 | RO |
| ECC_MULT_INT_ENA_REG | ECC interrupt enable register | 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 |

## 21.9 Registers

The addresses in this section are relative to ECC Accelerator base address provided in Table 5.3-2 in Chapter 5 *System and Memory*.

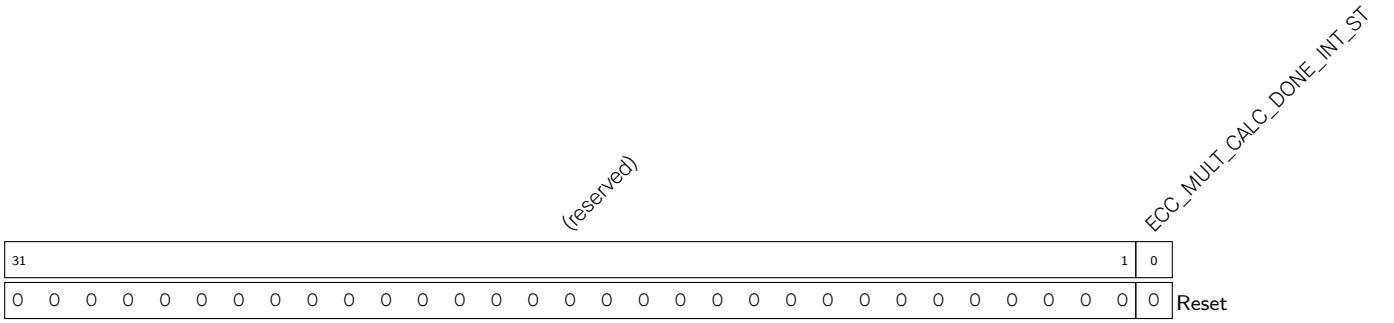
For how to program reserved fields, please refer to Section *Programming Reserved Register Field*.

**Register 21.1. ECC_MULT_INT_RAW_REG (0x000C)**



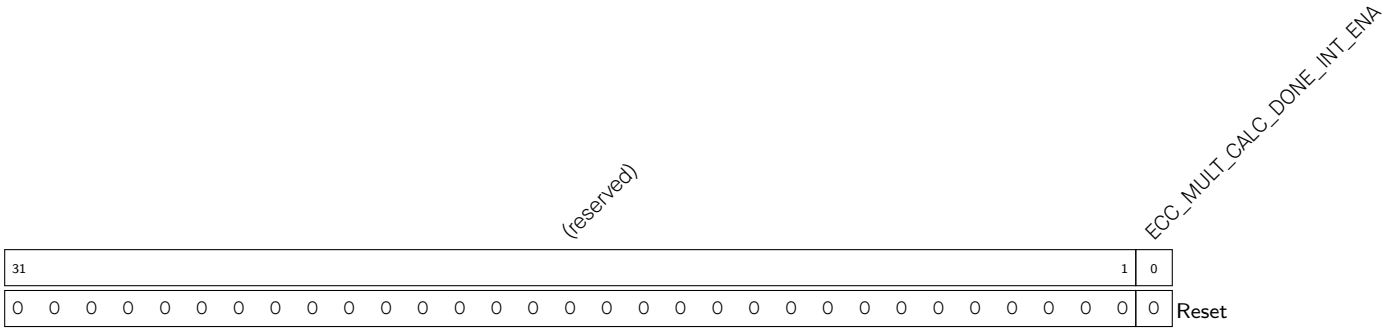

**ECC_MULT_CALC_DONE_INT_RAW** The raw interrupt status of the ECC_MULT_CALC_DONE_INT interrupt. (R/SS/WTC)

**Register 21.2. ECC_MULT_INT_ST_REG (0x0010)**



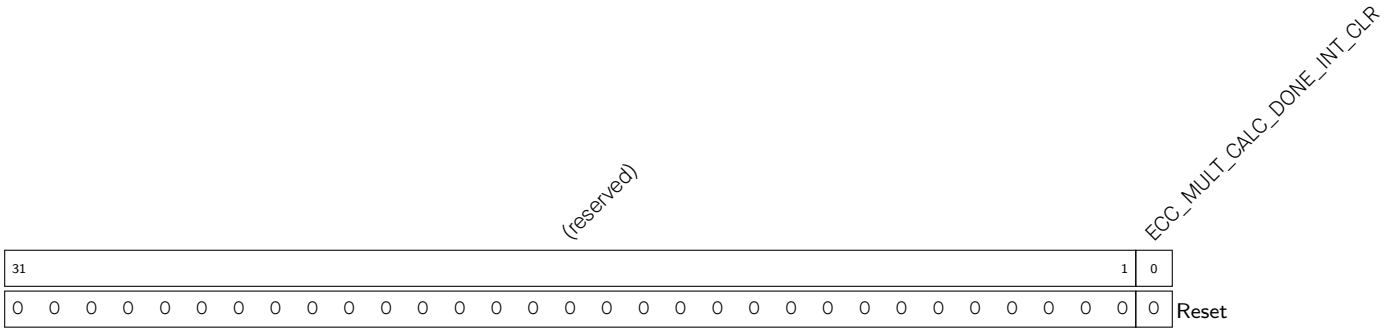

**ECC_MULT_CALC_DONE_INT_ST** The masked interrupt status of the ECC_MULT_CALC_DONE_INT interrupt. (RO)

Register 21.3. 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 21.4. 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 21.5. ECC_MULT_CONF_REG (0x001C)

| Bits | 31 | 30 | 29 | 28 … 9 | 8 | 7 … 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Field | ECC_MULT_MEM_CLOCK_GATE_FORCE_ON | ECC_MULT_CLK_EN | ECC_MULT_VERIFICATION_RESULT | (reserved) | ECC_MULT_SECURITY_MODE | ECC_MULT_WORK_MODE | ECC_MULT_MOD_BASE | ECC_MULT_KEY_LENGTH | ECC_MULT_RESET | ECC_MULT_START | |
| | 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 |

**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_KEY_LENGTH** Configures the key length of the ECC accelerator.
0: P-192
1: P-256
(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 21.5. ECC_MULT_CONF_REG (0x001C)

Continued from the previous page...

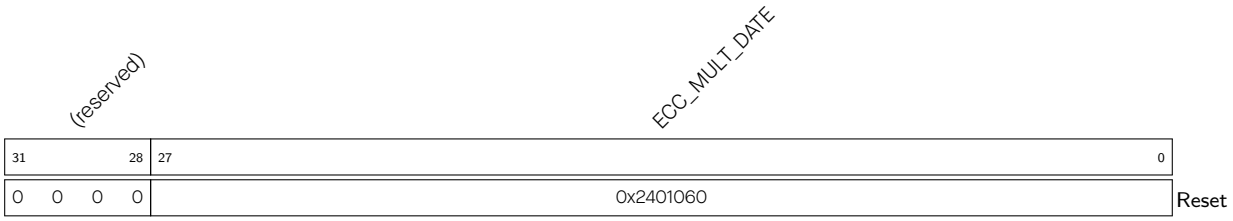
**ECC_MULT_SECURITY_MODE** Configures whether to enhance the anti-attack performance when calculating point multiplication. Only valid in working modes 0, 3, 4, and 7.
0: Disable
1: Enable
(R/W)

**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 the ECC memory clock gate on.
0: No effect
1: Force on
(R/W)

Register 21.6. ECC_MULT_DATE_REG (0x00FC)




**ECC_MULT_DATE** Version control register. (R/W)

# Chapter 22

# SHA Accelerator (SHA)

## 22.1 Introduction

ESP32-C61 integrates an SHA accelerator, which is a hardware device that speeds up the SHA algorithm significantly, compared to a SHA algorithm implemented solely in software. The SHA accelerator integrated in ESP32-C61 has two working modes, which are Typical SHA and DMA-SHA.

## 22.2 Features

The following functionality is supported:

- The following hash algorithms introduced in FIPS PUB 180-4 Spec
  - SHA-1
  - SHA-224
  - SHA-256
- 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

## 22.3 Working Modes

The SHA accelerator integrated in ESP32-C61 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 DMA 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 PCR_SHA_CLK_EN bit and clearing the PCR_SHA_RST_EN bit in the PCR_SHA_CONF_REG register. Additionally, users also need to clear PCR_ECDSA_RST_EN bit to reset ECDSA Digital Signature Peripheral (ECDSA_DS).

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 22.3-1.

Table 22.3-1. 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 22.3-2.

Table 22.3-2. SHA Hash Algorithm Selection

| Hash Algorithm | SHA_MODE_REG Configuration |
|---|---|
| SHA-1 | 0 |
| SHA-224 | 1 |
| SHA-256 | 2 |

## 22.4 Function Description

The SHA accelerator generates the message digest via two steps: Preprocessing and Hash operation.

### 22.4.1 Preprocessing

Preprocessing consists of three steps: padding the message, parsing the message into message blocks and setting the initial hash value.

#### 22.4.1.1 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 $m$ bits. Then $M$ shall be padded as introduced below:

1. First, append the bit "1" to the end of the message;
2. Second, append $k$ 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.

For more details, please refer to FIPS PUB 180-4 Spec > Section "Padding the Message".

#### 22.4.1.2 Parsing the Message

The message and its padding must be 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, all the message blocks are written into the SHA_M_*n*_REG: $M_0^{(i)}$ is 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.

**Note:**

For more information about "message block", please refer to FIPS PUB 180-4 Spec > Section "Glossary of Terms and Acronyms".

#### 22.4.1.3 Setting the Initial Hash Value

Before hash operation begins for any secure hash algorithms, the initial Hash value $H^{(0)}$ must be set based on different algorithms. However, the SHA accelerator uses the initial Hash values (constant C) stored in the hardware for hash tasks.

### 22.4.2 Hash Operation

After the preprocessing, the ESP32-C61 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-C61 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.

#### 22.4.2.1 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-C61 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**

1. Select a hash algorithm.
   - Configure the SHA_MODE_REG register based on Table 22.3-2.
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 the 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.

**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-C61 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.

#### 22.4.2.2 DMA-SHA Mode Process

ESP32-C61 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 data 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 DMA. Therefore, users are required to configure the DMA controller following the description in Chapter 4 *GDMA Controller (GDMA)*.

#### DMA-SHA process

1. Select a hash algorithm.
   - Select a hash algorithm by configuring the SHA_MODE_REG register. For details, please refer to Table 22.3-2.
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.

### 22.4.3 Message Digest

After the hash task completes, the SHA accelerator writes the message digest from the task to registers SHA_H_*n*_REG(*n*: 0~7). The lengths of the generated message digest are different depending on different hash algorithms. For details, see Table 22.4-1 below:

Table 22.4-1. 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 |

[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 22.4.1.2.

## 22.4.4 Interrupt

ESP32-C61's SHA accelerator can generate the following interrupt signal(s) that will be sent to the Interrupt Matrix.

- SHA_INTR

There is an internal interrupt source from SHA that can generate the above interrupt signal(s). The interrupt source from SHA is listed with their trigger conditions and the resulted interrupt signal(s) in Table 22.4-2.

Table 22.4-2. 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 10 *Interrupt Matrix* > Section 10.2 *Interrupt Terminology in ESP32-C61*.
- 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.

## 22.5 Register Summary

The addresses in this section are relative to the SHA accelerator base address provided in Table 5.3-2 in Chapter 5 *System and Memory*.

The abbreviations given in Column **Access** are explained in Section *Access Types for Registers*.

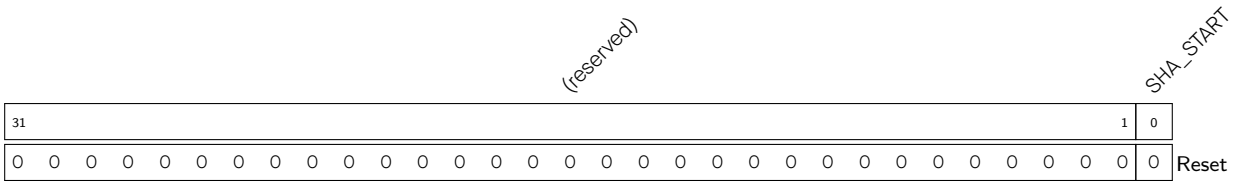
| Name | Description | Address | Access |
|---|---|---|---|
| **Control/Configuration Registers** | | | |
| SHA_MODE_REG | Configures SHA algorithm | 0x0000 | R/W |
| SHA_CONTINUE_REG | Continues SHA operation (only effective in Typical SHA mode) | 0x0014 | WO |
| SHA_DMA_START_REG | Starts the SHA accelerator 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_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 |
| 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 |

| Name | Description | Address | Access |
|---|---|---|---|
| **Version Register** | | | |
| SHA_DATE_REG | Version control register | 0x002C | R/W |

## 22.6 Registers

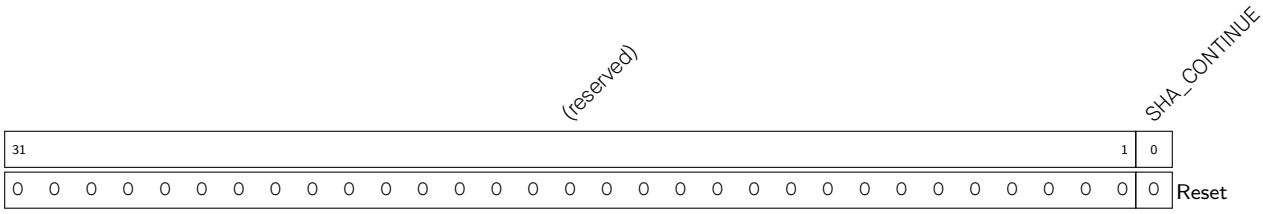
The addresses in this section are relative to the SHA accelerator base address provided in Table 5.3-2 in Chapter 5 *System and Memory*.

**Register 22.1. SHA_START_REG (0x0010)**



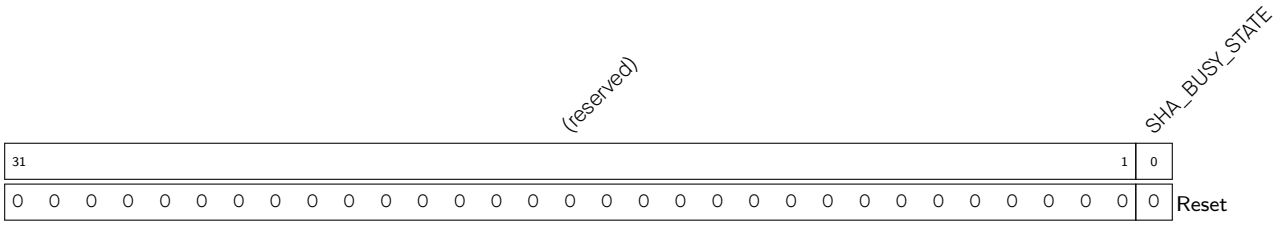

**SHA_START** Write 1 to start Typical SHA calculation. (WO)

**Register 22.2. SHA_CONTINUE_REG (0x0014)**




**SHA_CONTINUE** Write 1 to continue Typical SHA calculation. (WO)

**Register 22.3. SHA_BUSY_REG (0x0018)**



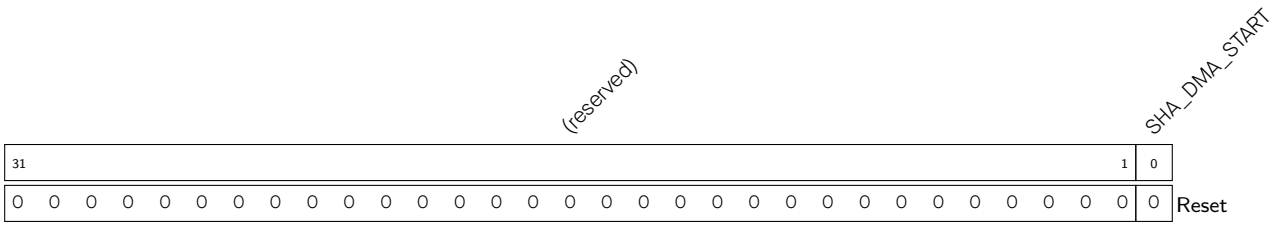

**SHA_BUSY_STATE** Represents the states of SHA accelerator.

0: idle

1: busy

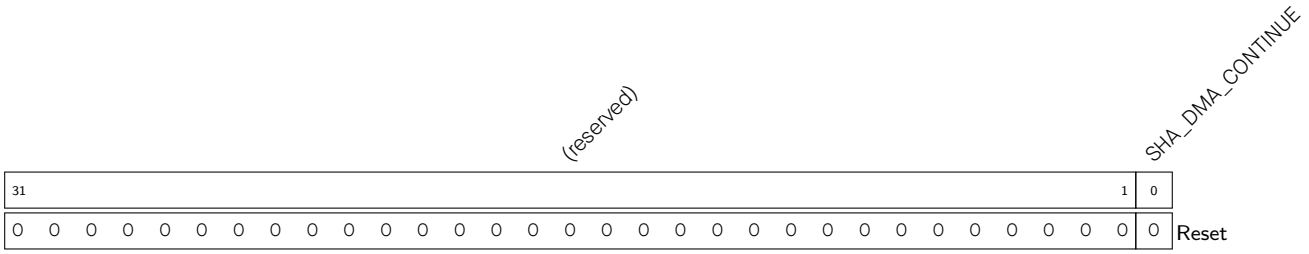
(RO)

Register 22.4. SHA_DMA_START_REG (0x001C)



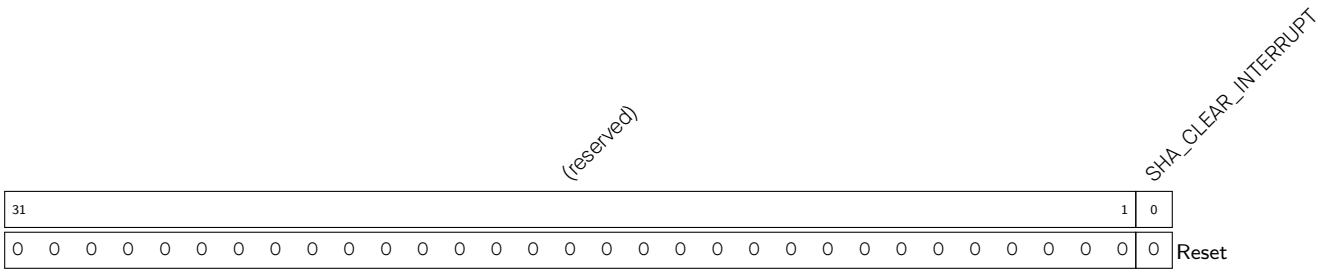

**SHA_DMA_START** Write 1 to start DMA-SHA calculation. (WO)

Register 22.5. SHA_DMA_CONTINUE_REG (0x0020)



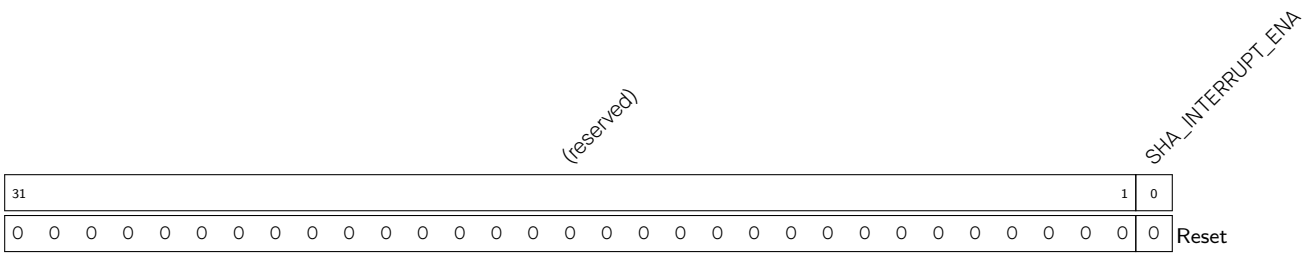

**SHA_DMA_CONTINUE** Write 1 to continue DMA-SHA calculation. (WO)

Register 22.6. SHA_INT_CLEAR_REG (0x0024)




**SHA_CLEAR_INTERRUPT** Write 1 to clear DMA-SHA interrupt. (WO)

Register 22.7. SHA_INT_ENA_REG (0x0028)




**SHA_INTERRUPT_ENA** Write 1 to enable DMA-SHA interrupt. (R/W)

### Register 22.8. SHA_DATE_REG (0x002C)

| Bits | 31:30 | 29:0 |
|---|---|---|
| Field | (reserved) | SHA_DATE |
| Reset | 0 0 | 0x20201229 |

**SHA_DATE** Version control register. (R/W)

### Register 22.9. SHA_MODE_REG (0x0000)

| Bits | 31:3 | 2:0 |
|---|---|---|
| Field | (reserved) | SHA_MODE |
| Reset | 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 | 0x0 |

**SHA_MODE** Configures the SHA algorithm.

0: SHA-1

1: SHA-224

2: SHA-256

3 ~ 7: invalid value

(R/W)

### Register 22.10. SHA_DMA_BLOCK_NUM_REG (0x000C)

| Bits | 31:6 | 5:0 |
|---|---|---|
| Field | (reserved) | SHA_DMA_BLOCK_NUM |
| Reset | 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 | 0x0 |

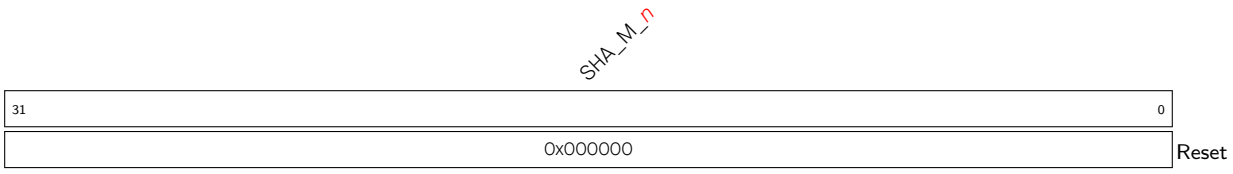
**SHA_DMA_BLOCK_NUM** Configures the DMA-SHA block number. (R/W)

### Register 22.11. SHA_H_*n*_REG (*n*: 0-7) (0x0040+4**n*)

| Bits | 31:0 |
|---|---|
| Field | SHA_H_*n* |
| Reset | 0x000000 |

**SHA_H_*n*** Represents the *n*th 32-bit piece of the Hash value. (R/W)

**Register 22.12. SHA_M_*n*_REG (*n*: 0-15) (0x0080+4**n*)**




**SHA_M_*n*** Represents the *n*th 32-bit piece of the message. (R/W)

# Chapter 23

# ECDSA Digital Signature Peripheral (ECDSA_DS)

## 23.1 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-C61'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.

## 23.2 Features

ESP32-C61's ECDSA_DS module supports:

- Digital signature generation and verification
- Two different elliptic curves, namely P-192 and P-256, defined in FIPS 186-5
- Two hash algorithms for message hash in the ECDSA_DS operation, namely SHA-224 and SHA-256, defined in FIPS PUB 180-4 Spec
- State-dependent register access control in different operation states to ensure information security

## 23.3 ECDSA Basics

### 23.3.1 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.

### 23.3.2 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.
   - Once a private key is chosen, it should be burned into the eFuse in ESP32-C61. (Please refer to Chapter 6 *eFuse Controller (EFUSE)*).
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.

### 23.3.3 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 $r = x \bmod n$. If $r$ is equal to zero, return to the previous step and select a new random value of $k$.

(c) Calculate $s = k^{-1} * (z + d * r) \bmod n$. 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.

### 23.3.4 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 $u_1 = z * s^{-1} \bmod n$ and $u_2 = r * s^{-1} \bmod n$.
   (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 $r = x_1 \bmod n$. If $r$ is not equal to $x_1 \bmod n$, 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.

## 23.4 Functional Description

This section describes the details of ESP32-C61's ECDSA_DS module.

### 23.4.1 ECDSA_DS Working Modes

The ECDSA_DS module integrated in the ESP32-C61 has three working modes, which are Signature Generation, Signature Verification and Public Key Export modes.

Users can select the working mode for ECDSA_DS by configuring ECDSA_WORK_MODE according to Table 23.4-1 below.

Table 23.4-1. ECDSA_DS Working Mode

| ECDSA_WORK_MODE | Working Mode |
|---|---|
| 0 | Signature Verification |
| 1 | Signature Generation |
| 2 | Public Key Export |

Users can select the elliptic curves used by the ECDSA_DS module by configuring ECDSA_ECC_CURVE according to Table 23.4-2.

Table 23.4-2. ECDSA_DS Elliptic Curve Selection

| ECDSA_ECC_CURVE | Elliptic Curve |
|---|---|
| 0 | P-192 |
| 1 | P-256 |

Users can select the SHA algorithms for message hash by configuring ECDSA_SHA_MODE according to Table 23.4-3.

Table 23.4-3. ECDSA_DS SHA Algorithm

| ECDSA_SHA_MODE | SHA Algorithm |
|---|---|
| 1 | SHA-224 |
| 2 | SHA-256 |
| Others | Invalid |

Additionally, users can check the working states of ECDSA_DS by inquiring the ECDSA_STATE_REG register and comparing the return value against the Table 23.4-4 below.

Table 23.4-4. 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 | GAIN | Waiting for users to gain information from ECDSA_DS. Corresponding to GAIN stage. |
| 3 | BUSY | In the middle of a hardware operation. Corresponding to PREP, PROC, and POST stage. |

## 23.4.2 Data and Data Block

ESP32-C61's ECDSA_DS module operates on data of 256 or 512 bits. This 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:

$$D[255:0] = D[7][31:0], D[6][31:0], D[5][31:0], D[4][31:0], D[3][31:0], D[2][31:0], D[1][31:0], D[0][31:0]$$

#### 23.4.2.1 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 example:

- write $D[0]$ to "starting address"
- write $D[1]$ to "starting address + 4"
- $\cdots$
- write $D[7]$ to "starting address + 28"

**Note:**
When the data size of 192 bits is used, you need to append 0 after 192 bits of data and write 256 bits of data.

#### 23.4.2.2 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 the 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 example:

- read $D[0]$ from "starting address"
- read $D[1]$ from "starting address + 4"
- $\cdots$
- read $D[7]$ from "starting address + 28"

**Note:**
When the data size of 192 bits is used, only use the low 192 bits (6 blocks) of data.

#### 23.4.2.3 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".

#### 23.4.2.4 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".

### 23.4.3 Security Features

To ensure the security of the operation process, the ECDSA_DS module implements a variety of security functions.

#### 23.4.3.1 High Anti-Attack Performance

ESP32-C61's ECDSA_DS module leverages ECC's enhanced anti-attack performance (refer to Section 21.4.3 *Enhancing Anti-Attack Performance*) every time it performs an operation. This means that each time a signature is generated and verified by ECDSA_DS:

- The execution latency is constant for a given operation.
- Power consumption variations are minimized.

This provides ESP32-C61's ECDSA_DS module strong anti-attack performance.

#### 23.4.3.2 State-Dependent Register Access Control

ESP32-C61'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, GAIN, and BUSY. For details about all ECDSA_DS working states, please refer to Table 23.4-4.

For detailed information on the state-dependent access control of each ECDSA_DS register, please refer to Section Register Summary.

#### 23.4.3.3 Hardware Occupation

During the operation, the following hardware modules will be occupied by ESP32-C61's ECDSA_DS:

- 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.

## 23.5 Programming Procedures

### 23.5.1 ECDSA_DS Process

The overall ECDSA_DS process consists of the following six stages.

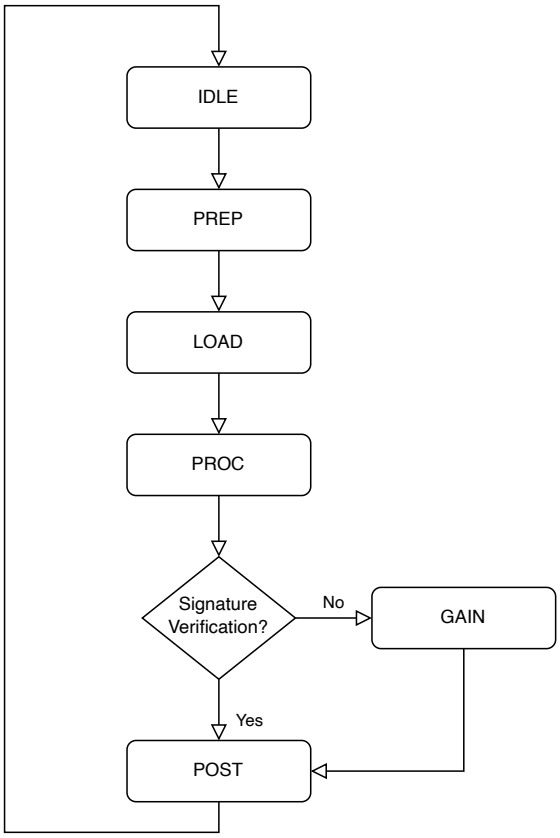



Figure 23.5-1. ECDSA_DS Process

The detailed programming procedures of each stage are described in the following sections.

#### 23.5.1.1 IDLE Stage

In the IDLE stage:

1. Configure the static parameters including eFuse bits:
   (a) ECDSA_KEY: The value of private key $d$ in ECDSA_DS. To correctly configure the key value in eFuse, users 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 6 *eFuse Controller (EFUSE)* for more detailed configuration steps.
2. Configure ECDSA_CONF_REG, including the following fields:
   (a) ECDSA_WORK_MODE: Select the working mode of ECDSA_DS.
   (b) ECDSA_ECC_CURVE: Select the elliptic curve of ECDSA_DS.
   (c) ECDSA_SOFTWARE_SET_Z: Configure whether to use direct input $z$.
   (d) ECDSA_DETERMINISTIC_K: Configure whether to use deterministic k.
   (e) ECDSA_DETERMINISTIC_LOOP: Configure the loop number of deterministic k.
3. Configure the register field ECDSA_START to enter the PREP stage.

**Note:**
For more details about Deterministic ECDSA, please refer to RFC6979.

#### 23.5.1.2 PREP Stage

In the PREP stage, ECDSA_STATE_REG is BUSY, and ECDSA_DS performs preparation.

Wait till the PREP stage to end by polling ECDSA_BUSY until it is not BUSY. Then ECDSA_DS will automatically enter the LOAD stage.

#### 23.5.1.3 LOAD Stage

In the LOAD stage:

1. Provide input $z$ into ECDSA_DS using one of the following options:
   - Direct input $z$: write $z$ to ECDSA_MEM_Z.
   - ECDSA_DS SHA interface: generate $z$ from the message. For details, please refer to Section 23.5.1.7.
2. According to the selected ECDSA_WORK_MODE, configure as follows:
   - Signature Verification:
     (a) write signature ($r$, $s$) to ECDSA_MEM_R and ECDSA_MEM_S.
     (b) write public key ($Qx$, $Qy$) to ECDSA_MEM_Qx and ECDSA_MEM_Qy.
   - Signature Generation:
     (a) no other configuration is required.
   - Public Key Export:
     (a) no other configuration is required.

3. Write 1 to ECDSA_LOAD_DONE, indicating that the configuration is done. Then ECDSA_DS will automatically enter the PROC stage.

#### 23.5.1.4 PROC Stage

In the PROC stage, ECDSA_STATE_REG is BUSY, and ECDSA_DS performs operation based on the selected working mode.

Wait till the PROC stage to end by polling ECDSA_BUSY until it is not BUSY. Then ECDSA_DS will automatically enter the either the GAIN stage or the POST stage depending on the selected working mode:

- Signature generation: GAIN stage
- Signature verification: POST stage

#### 23.5.1.5 GAIN Stage

When the Signature Generation mode or the Public Key Export mode is selected, ECDSA_DS enters the GAIN stage after PROC stage:

1. Read data from the ECDSA_DS memory:
   - read signature ($r$, $s$) from ECDSA_MEM_R and ECDSA_MEM_S only in the Signature Generation mode.
   - read public key ($Qx$, $Qy$) from ECDSA_MEM_Qx and ECDSA_MEM_Qy both in the Signature Generation mode and the Public Key Export mode.
2. Write 1 to ECDSA_GET_DONE, indicating that the GAIN stage is done. Then ECDSA_DS will automatically enter the POST stage.

#### 23.5.1.6 POST Stage

In the POST stage, ECDSA_STATE_REG is BUSY, and 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 ECDSA_DS will automatically return to the IDLE stage.

#### 23.5.1.7 ECDSA_DS SHA Interface

ESP32-C61's ECDSA_DS module can automatically execute hash operation and generates $z$ based on a direct input message.

For message hash, ECDSA_DS 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 23.4.2.3.
2. Parse the message and its padding into message blocks. See details in Section 23.4.2.4.
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.

### 23.5.2 Clocks and Resets

ESP32-C61's ECDSA_DS module has one clock module and one reset module. It is activated by setting the PCR_ECDSA_CLK_EN bit and clearing the PCR_ECDSA_RST_EN bit. For details on how to configure the ECDSA_DS clock and reset, please refer to Chapter 8 *Reset and Clock*.

### 23.5.3 Interrupts

ESP32-C61'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 10 *Interrupt Matrix* > Section 10.2 *Interrupt Terminology in ESP32-C61*.

## 23.6 Memory Blocks

ECDSA_DS's memory blocks store input data and output data of the operation.

Table 23.6-1. 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 ECDSA_DS base address is provided in Table 5.3-2 in Chapter 5 *System and Memory*.

## 23.7 Register Summary

The addresses in this section are relative to ECDSA_DS base address provided in Table 5.3-2 in Chapter 5 *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.
- PG: GAIN state, corresponding to GAIN 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.

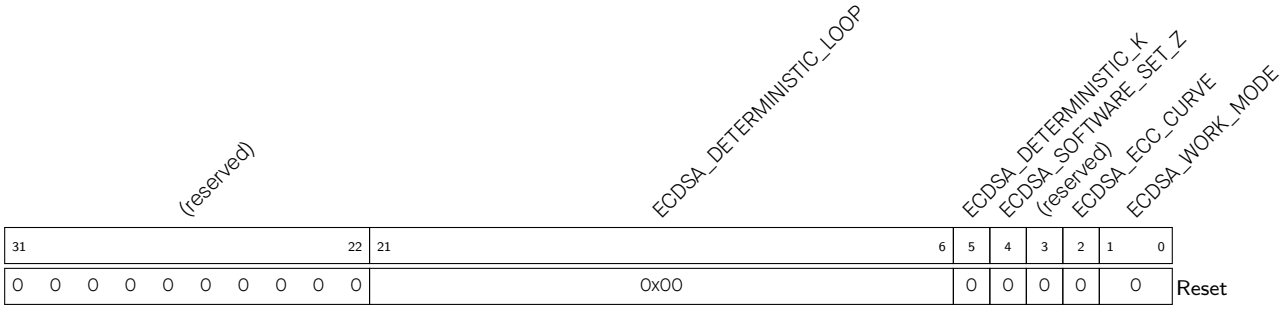
| Name | Description | Address | Access | | | |
|---|---|---|---|---|---|---|
| | | | PI | PL | PG | PB |
| **Data Memory** | See Table 23.6-1. | | | | | |
| **Configuration Registers** | | | | | | |
| ECDSA_CONF_REG | ECDSA_DS configuration register | 0x0004 | R/W | N/A | | |
| ECDSA_START_REG | ECDSA_DS start register | 0x001C | WT | | | N/A |
| **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/A |
| **SHA Registers** | | | | | | |
| ECDSA_SHA_MODE_REG | ECDSA_DS SHA-control register (Hash algorithm) | 0x0200 | N/A | R/W | N/A | |
| ECDSA_SHA_START_REG | ECDSA_DS SHA-control register (operation) | 0x0210 | N/A | WT | N/A | |
| ECDSA_SHA_CONTINUE_REG | ECDSA_DS SHA-control register (operation) | 0x0214 | N/A | WT | N/A | |
| ECDSA_SHA_BUSY_REG | ECDSA_DS SHA-control status register | 0x0218 | N/A | RO | N/A | |
| **Version Register** | | | | | | |
| ECDSA_DATE_REG | Version control register | 0x00FC | R/W | N/A | | |

## 23.8 Registers

The addresses in this section are relative to ECDSA_DS base address provided in Table 5.3-2 in Chapter 5 *System and Memory*.

For how to program reserved fields, please refer to Section *Programming Reserved Register Field*.

**Register 23.1. ECDSA_CONF_REG (0x0004)**




**ECDSA_WORK_MODE** Configures the working mode of ECDSA_DS.

0: Signature Verification Mode

1: Signature Generation Mode

2: Public Key Export Mode

3: Invalid

(R/W)

**ECDSA_ECC_CURVE** Configures the elliptic curve used.

0: P-192

1: P-256

(R/W)

**ECDSA_SOFTWARE_SET_Z** Configures how the parameter $z$ is set.

0: Generated from SHA result

1: Written by software

(R/W)

**ECDSA_DETERMINISTIC_K** Configures how the parameter $k$ is set.

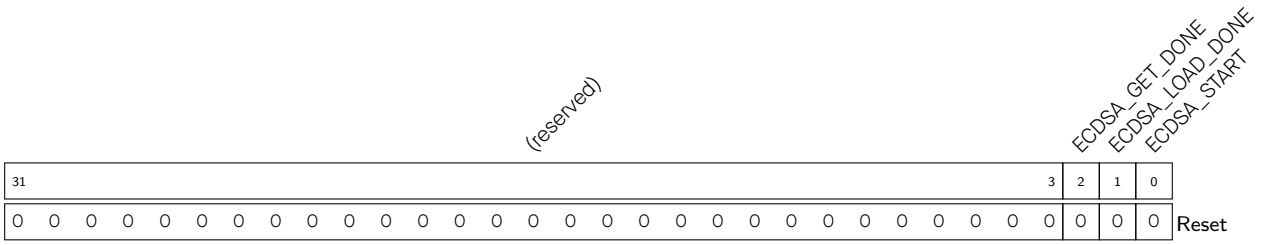
0: Automatically generated by TRNG

1: Generated by the deterministic derivation algorithm

(R/W)

**ECDSA_DETERMINISTIC_LOOP** Configures the (loop number - 1) value used in the deterministic derivation algorithm to derive $k$. (R/W)

Register 23.2. 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

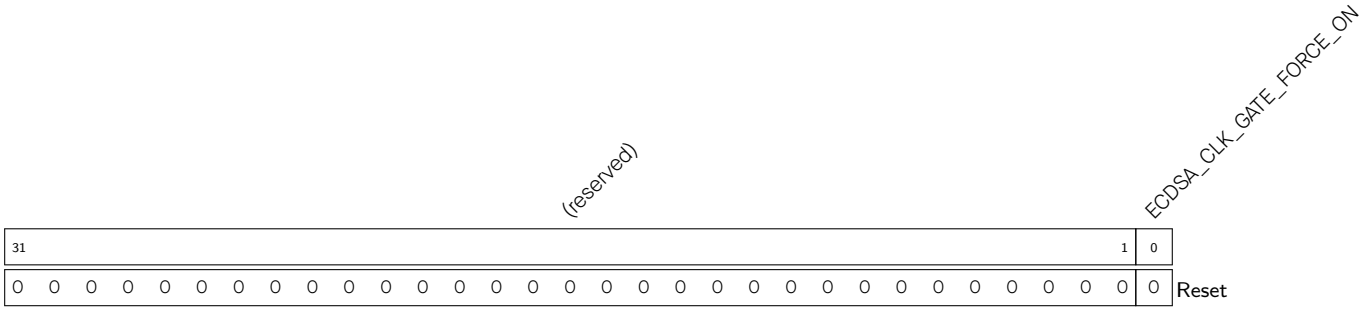
1: Start the ECDSA_DS operation

(WT)

**ECDSA_LOAD_DONE** Write 1 to generate a signal indicating ECDSA_DS's LOAD operation is done. This bit will be self-cleared after configuration. (WT)

**ECDSA_GET_DONE** Write 1 to generate a signal indicating ECDSA_DS's GAIN operation is done. This bit will be self-cleared after configuration. (WT)

Register 23.3. ECDSA_CLK_REG (0x0008)



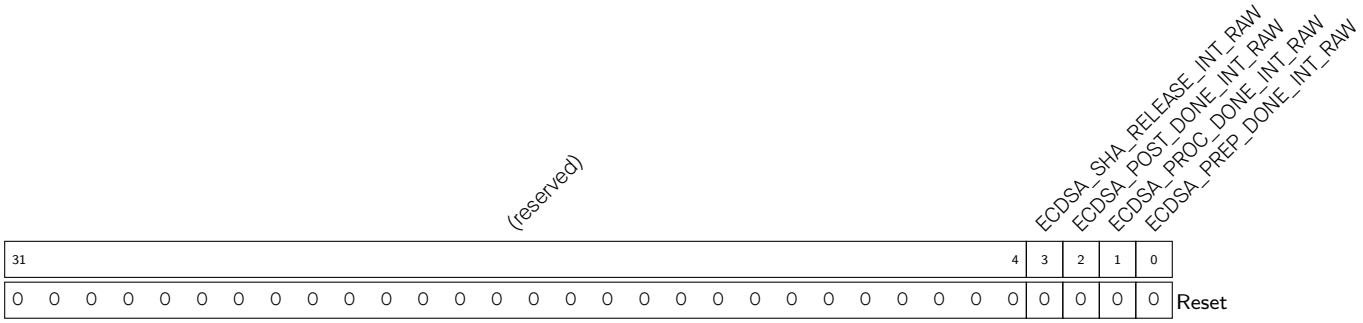

**ECDSA_CLK_GATE_FORCE_ON** Configures whether to force on ECDSA_DS memory clock gate.

0: No effect

1: Force on

(R/W)

Register 23.4. ECDSA_INT_RAW_REG (0x000C)



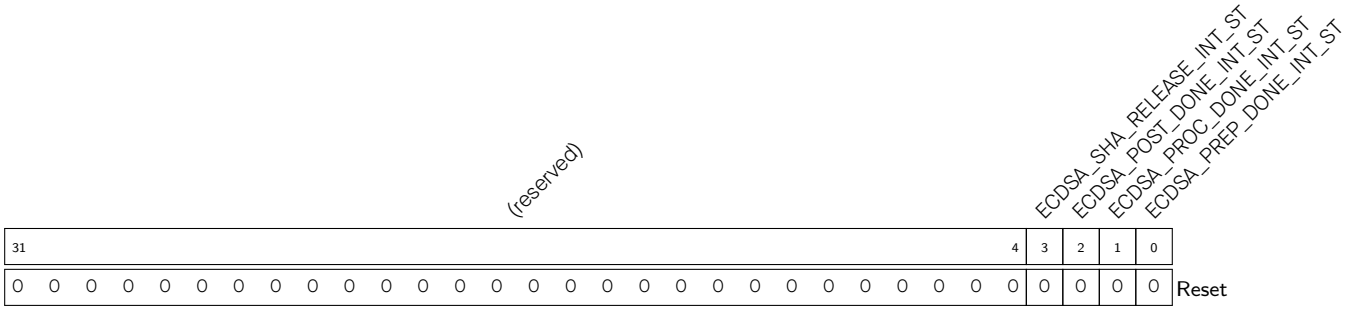

**ECDSA_PREP_DONE_INT_RAW** The raw interrupt status of the ECDSA_PREP_DONE_INT interrupt. (R/SS/WTC)

**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 23.5. ECDSA_INT_ST_REG (0x0010)



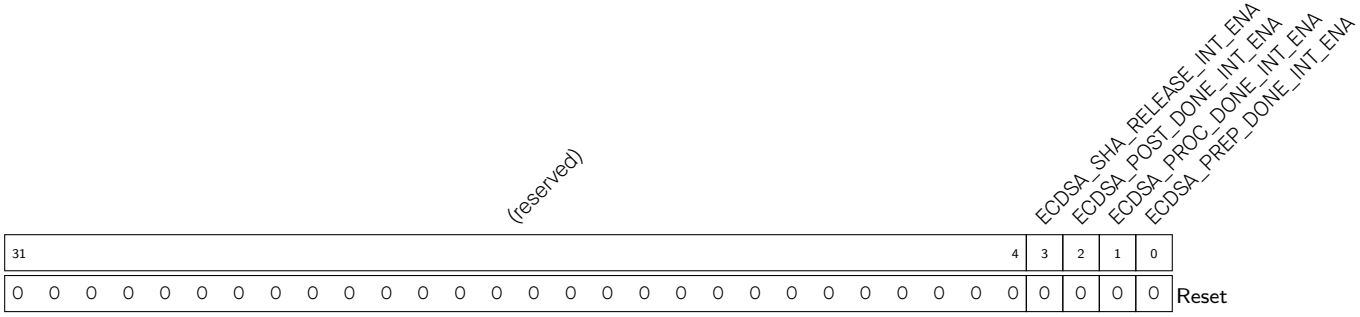

**ECDSA_PREP_DONE_INT_ST** The masked interrupt status of the ECDSA_PREP_DONE_INT interrupt. (RO)

**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 23.6. ECDSA_INT_ENA_REG (0x0014)



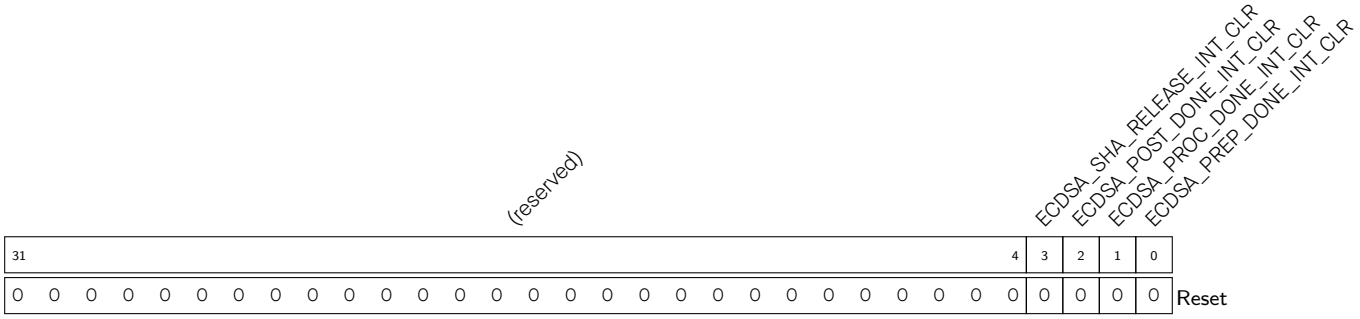

**ECDSA_PREP_DONE_INT_ENA** Write 1 to enable the ECDSA_PREP_DONE_INT interrupt. (R/W)

**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 23.7. ECDSA_INT_CLR_REG (0x0018)



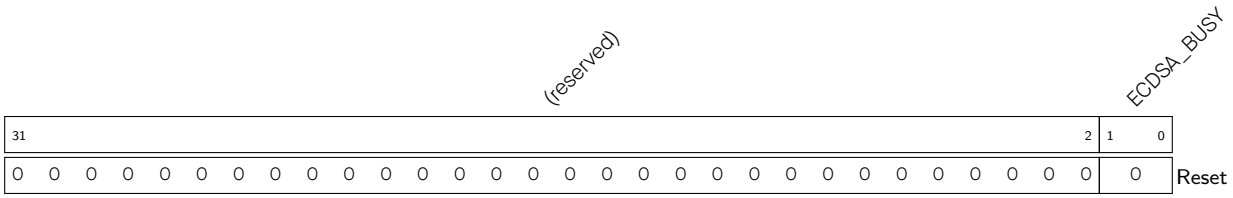

**ECDSA_PREP_DONE_INT_CLR** Write 1 to clear the ECDSA_PREP_DONE_INT interrupt. (WT)

**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 23.8. ECDSA_STATE_REG (0x0020)




**ECDSA_BUSY** Represents the working state of ECDSA_DS.

0: IDLE

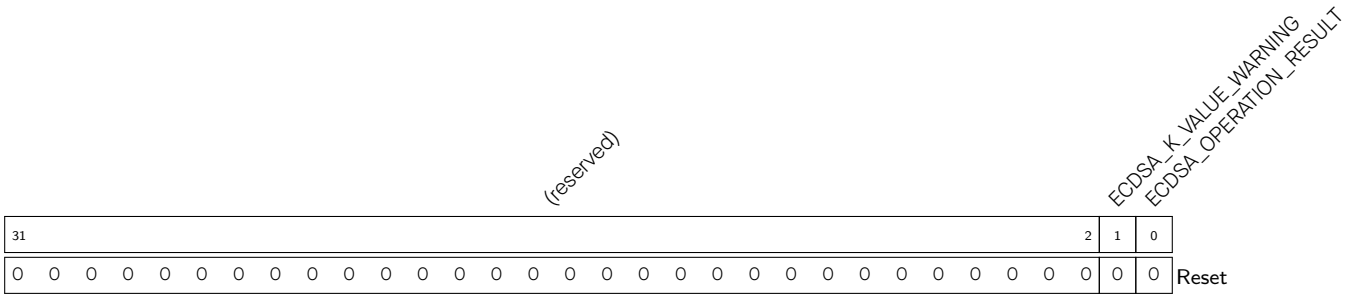
1: LOAD

2: GAIN

3: BUSY

(RO)

Register 23.9. 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)

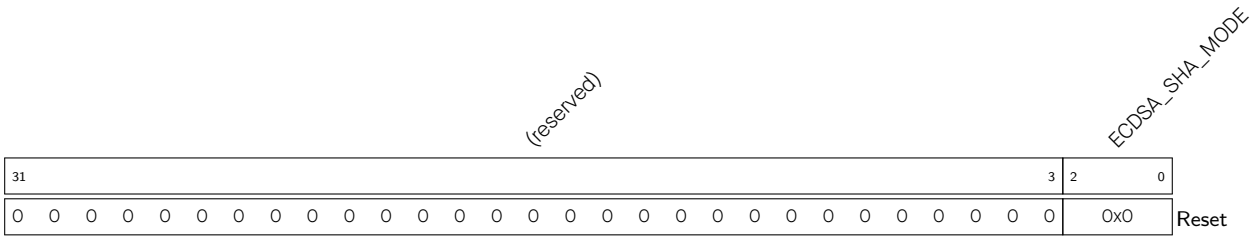
**ECDSA_K_VALUE_WARNING** Indicates if the $k$ value is greater than the base point order.

0: $k$ value is not greater than the base point order. In this case, the $k$ value is the set $k$ value.

1: $k$ value is greater than than the base point order. In this case, the $k$ value is the set $k$ mod n.

Only valid when the ECDSA_DS operation is done. (RO/SS)

Register 23.10. ECDSA_SHA_MODE_REG (0x0200)




**ECDSA_SHA_MODE** Configures SHA algorithms for message hash.

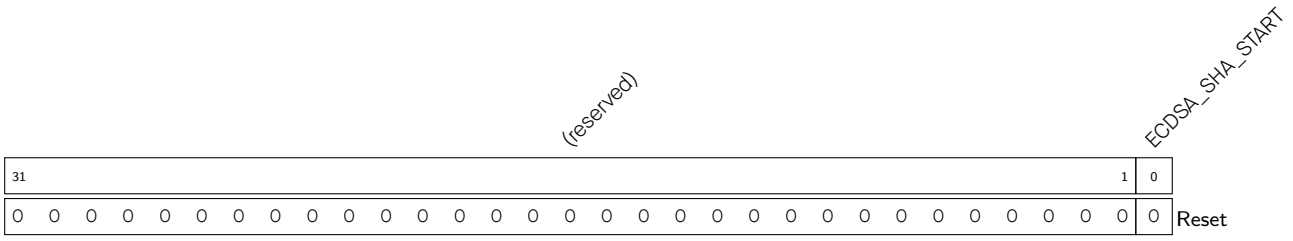
1: SHA-224

2: SHA-256

Others: invalid

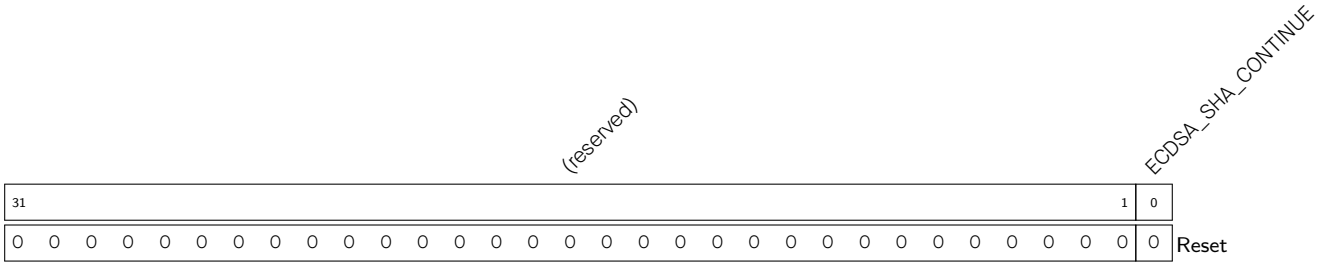
(R/W)

Register 23.11. ECDSA_SHA_START_REG (0x0210)



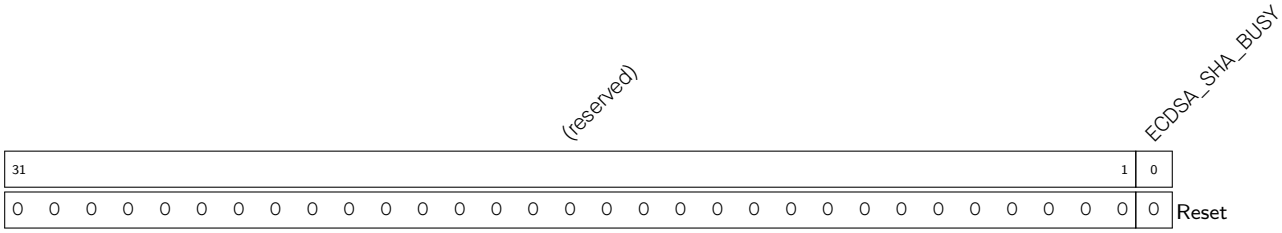

**ECDSA_SHA_START** Write 1 to start the first SHA operation in the ECDSA_DS process. This bit will be self-cleared after configuration. (WT)

Register 23.12. 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 23.13. ECDSA_SHA_BUSY_REG (0x0218)



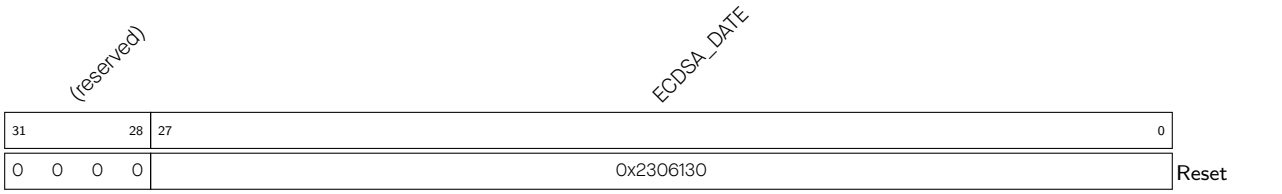

**ECDSA_SHA_BUSY** Represents the working state of the SHA accelerator in the ECDSA_DS process.

0: IDLE

1: BUSY

(RO)

Register 23.14. ECDSA_DATE_REG (0x00FC)




**ECDSA_DATE** The ECDSA_DS version control register. (R/W)

# Chapter 24

# External Memory Encryption and Decryption (XTS_AES)

## 24.1 Overview

The ESP32-C61 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.

## 24.2 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

## 24.3 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 24.3-1.

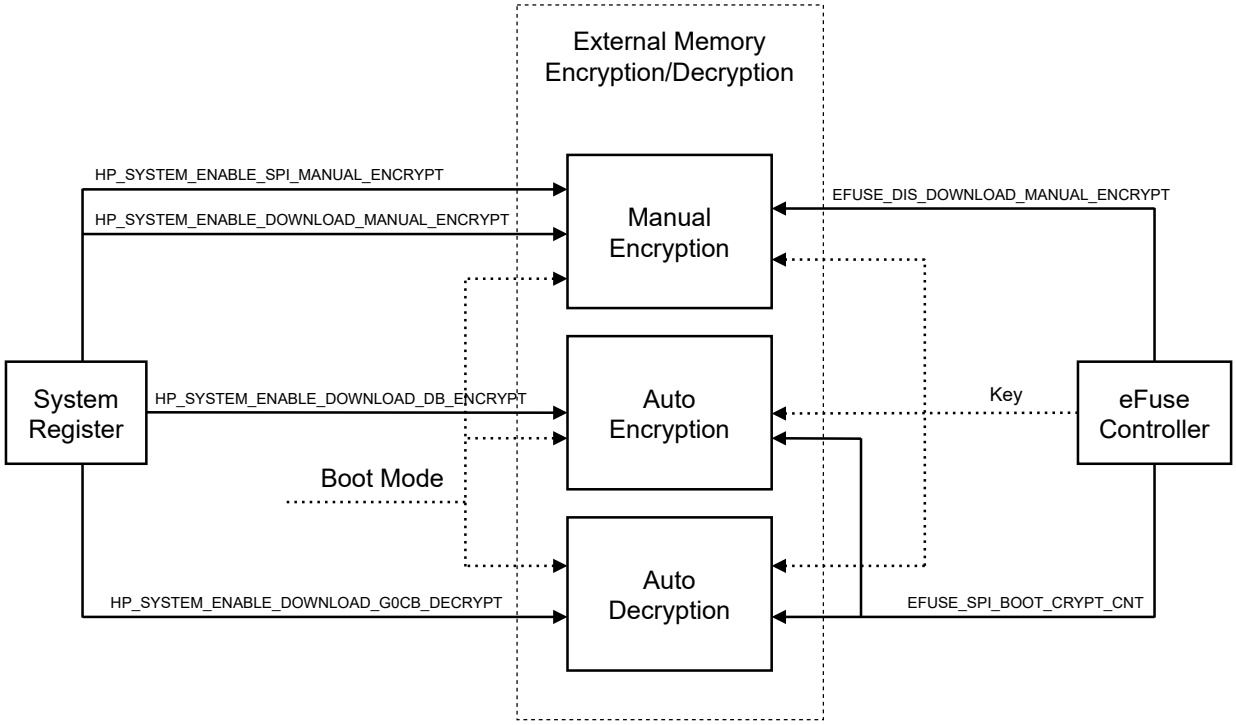



**Figure 24.3-1. Architecture of the External Memory Encryption and Decryption**

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 18 *System Registers*), the following three bits in register HP_SYSTEM_EXTERNAL_DEVICE_ENCRYPT_DECRYPT_CONTROL_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 6 *eFuse Controller (EFUSE)*.

## 24.4 Functional Description

### 24.4.1 XTS Algorithm

Both manual encryption and auto encryption/decryption use the XTS 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.

### 24.4.2 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 and cannot be accessed by software.

The $Key$ is 256-bit long. The value of the $Key$ is determined by the content in one eFuse block from BLOCK4 ~ BLOCK9. For easier description, we define:

- Block$_A$: the block whose key purpose is EFUSE_KEY_PURPOSE_XTS_AES_128_KEY (please refer to Table 6.3-2 *Secure Key Purpose Values*) in Chapter 6 *eFuse Controller (EFUSE)*. If Block$_A$ exists, a 256-bit $Key_A$ is stored in it.

There are two possibilities of how the $Key$ is generated depending on whether Block$_A$ exists or not, as shown in Table 24.4-1. In each case, the $Key$ can be uniquely determined.

**Table 24.4-1. $Key$ Generated Based on $Key_A$**

| Block$_A$ | $Key$ | $Key$ Length (bit) |
|---|---|---|
| Exists | $Key_A$ | 256 |
| Does not exist | $0^{256}$ | 256 |

**Notes:**

- "$0^{256}$" indicates a bit string that consists of 256-bit zeros.
- Using $0^{256}$ as $Key$ is not secure. It is recommended to configure a valid key.

For more information of key purposes, please refer to Table 6.3-2 *Secure Key Purpose Values* in Chapter 6 *eFuse Controller (EFUSE)*.

### 24.4.3 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 24-bit physical address, with range of 0x0000_0000 ~ 0x00FF_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 its individual target memory space and the relevant parameters.

For Auto Encryption/Decryption blocks, these parameters are automatically determined by hardware. For the Manual Encryption blocks, 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.

### 24.4.4 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 24.4-2.

**Table 24.4-2. 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 |

### 24.4.5 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.

**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_EXTERNAL_DEVICE_ENCRYPT_DECRYPT_CONTROL_REG is 1, the Manual Encryption block can be enabled. Otherwise, it is not operational.

- In Download Boot mode:

  If bit HP_SYSTEM_ENABLE_DOWNLOAD_MANUAL_ENCRYPT in register HP_SYSTEM_EXTERNAL_DEVICE_ENCRYPT_DECRYPT_CONTROL_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 $Key$.

### 24.4.6 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 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 Encryption block can be enabled. Otherwise, it is not operational.

- In Joint Download Boot mode:

  If bit SYSTEM_ENABLE_DOWNLOAD_DB_ENCRYPT in register SYSTEM_EXTERNAL_DEVICE_ENCRYPT_DECRYPT_CONTROL_REG is 1, the Auto Encryption block can be enabled. Otherwise, it is not operational.

**Note:**

- When the Auto Encryption block is enabled, it automatically encrypts data written by the CPU to external RAM. The encrypted data is stored in the external RAM, and this encryption process is fully hardware-based, requiring no software intervention. The operation is transparent to the cache, and the encryption key remains inaccessible to the user during the process.
- When the Auto Encryption block is disabled, it bypasses the CPU's access request to the cache and does not

process the data. As a result, data is written directly to external RAM in plaintext.

### 24.4.7 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 Download Boot mode

  when bit HP_SYS_ENABLE_DOWNLOAD_GOCB_DECRYPT in register HP_SYS_EXTERNAL_DEVICE_ENCRYPT_DECRYPT_CONTROL_REG is 1, the Auto Decryption block can be enabled. Otherwise, it is not operational.

**Note:**

- When the Auto Decryption block is enabled, it automatically decrypts encrypted data or instructions read by the CPU from external memory via the cache. This decryption process is hardware-based and requires no software intervention, remaining transparent to the cache. The decryption key is not accessible to the software during this process.
- When the Auto Decryption block is disabled, it does not affect the contents stored in external memory, regardless of whether they are encrypted. Consequently, the CPU reads the original data stored in external memory via the cache.

## 24.5 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:
   - Write 0 to register XTS_AES_DESTINATION_REG.
   - 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 24.4.3.
2. Write plaintext instructions/data to the registers block XTS_AES_PLAIN_*n*_REG (*n*: 0-15). For detailed information, please refer to Section 24.4.4.
   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.

## 24.6 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-C61 XTS_AES supports two Anti-DPA methods 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 Tweak value.
- Step 2: Calculating Cipher/Plain text. In this section, we define this step as calculating Data units.

ESP32-C61 allows users to enable anti-DPA function separately during the above-mentioned two steps to enhance security. See details in the following sections.

### 24.6.1 Clock Anti-DPA Function

Clock Anti-DPA function is the first method to defend against external DPA attacks. 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 Data units:

$$Anti\_DPA\_enabled\_in\_calc\_D = select\_reg\ ?\ reg\_d\_dpa\_en : efuse\_dpa\_en$$

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 Data units.
If $Anti\_DPA\_level$ is not 0, Anti-DPA is always enabled when the XTS-AES algorithm is calculating Tweak value.

**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 Data units, 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 Data units, the read and write bandwidth will be reduced by more than 50% when the Anti-DPA level >= 6.

### 24.6.2 Pseudo-round Anti-DPA Function

The pseudo-round function of ESP32-C61's XTS-AES is the second method to achieve higher anti-attack performance. When the pseudo-round function is enabled, XTS-AES will randomly add pseudo-rounds before and after the original operation rounds. In a pseudo-round, XTS-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 XTS-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 in different calculation steps of XTS-AES:

- 0: not enable pseudo-round function during calculation;
- 1: enable pseudo-round function when calculating Tweak value;
- 2: enable pseudo-round function when calculating Tweak value and some rounds of calculating Data units;
- 3: enable pseudo-round function when calculating Tweak value and all rounds of calculating Data units.

**Total number of pseudo-rounds**

The total number of pseudo-rounds that will be randomly inserted into each XTS-AES operation round is controlled by:

- pseudo_base: configures the basic round number of pseudo-round function.
- pseudo_inc: configures the random incremental round number of pseudo-round function.

The total number of pseudo-rounds will be randomly in the range

$$[pseudo\_base, pseudo\_base + (2^{pseudo\_inc} - 1)]$$

#### Random number update frequency

pseudo_rnd_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).

#### Configuration

The parameters mentioned above can be configured through eFuse bits or XTS_AES related registers, depending on the value of EFUSE_XTS_DPA_PSEUDO_LEVEL:

- 0: The user can configure using XTS_AES related registers.
- 1, 2, or 3: User configuration is not allowed.

See Table 24.6-1 for details.

**Table 24.6-1. Configuration of XTS_AES Pseudo-round Anti-DPA**

| EFUSE_XTS_DPA_PSEUDO_LEVEL | pseudo_mode | pseudo_base | pseudo_inc | pseudo_rnd_cnt |
|---|---|---|---|---|
| 0 | XTS_AES_PSEUDO_MODE | XTS_AES_PSEUDO_BASE | XTS_AES_PSEUDO_INC | XTS_AES_PSEUDO_RNG_CNT |
| 1 | 1 | 4 | 2 | 7 |
| 2 | 2 | 4 | 2 | 7 |
| 3 | 3 | 4 | 2 | 7 |

## 24.7 Register Summary

The addresses in this section are relative to the External Memory Encryption and Decryption base address provided in Table 5.3-2 in Chapter 5 *System and Memory*.

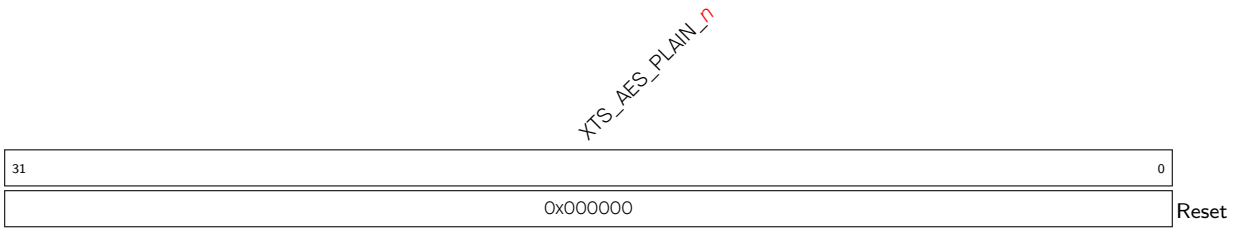
The abbreviations given in Column **Access** are explained in Section *Access Types for Registers*.

| 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 | Stores the physical address of the external memory | 0x0348 | R/W |
| XTS_AES_DPA_CTRL_REG | Configures the Anti-DPA function | 0x0388 | R/W |
| XTS_AES_PSEUDO_ROUND_CONF_REG | XTS-AES pseudo-round function configuration register | 0x038C | R/W |
| **Control/status Registers** | | | |
| XTS_AES_TRIGGER_REG | Activates AES algorithm | 0x034C | WT |
| XTS_AES_RELEASE_REG | Releases control | 0x0350 | WT |
| XTS_AES_DESTROY_REG | Destroys control | 0x0354 | WT |
| XTS_AES_STATE_REG | Status register | 0x0358 | RO |
| **Version control register** | | | |
| XTS_AES_DATE_REG | Version control register | 0x035C | R/W |

## 24.8 Registers

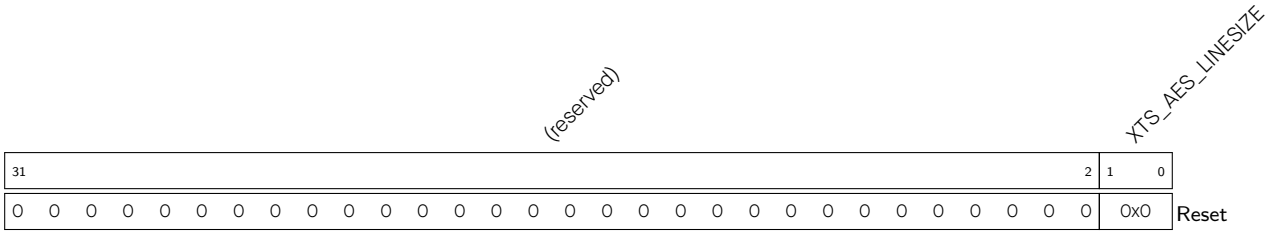
The addresses in this section are relative to the External Memory Encryption and Decryption base address provided in Table 5.3-2 in Chapter 5 *System and Memory*.

**Register 24.1. XTS_AES_PLAIN_*n*_REG (*n*: 0-15) (0x0300+4**n*)**




**XTS_AES_PLAIN_*n*** Configures the *n*th 32-bit piece of plain text. (R/W)

**Register 24.2. XTS_AES_LINESIZE_REG (0x0340)**




**XTS_AES_LINESIZE** Configures the data size of one encryption operation.

0: 16 bytes

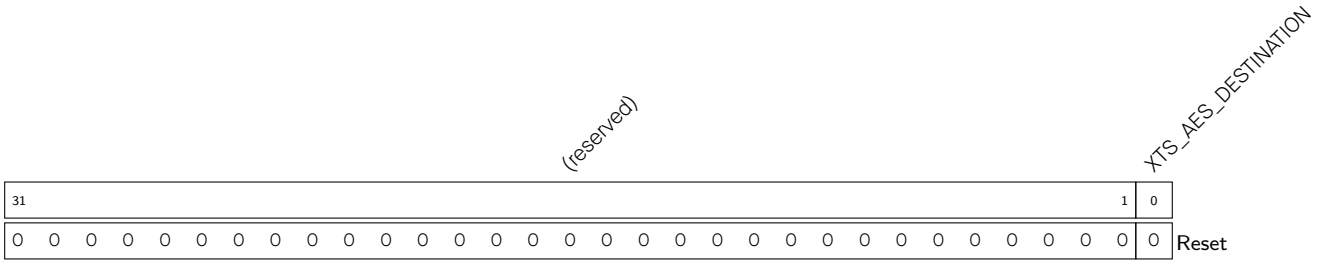
1: 32 bytes

2: 64 bytes

3: Invalid

(R/W)

Register 24.3. XTS_AES_DESTINATION_REG (0x0344)



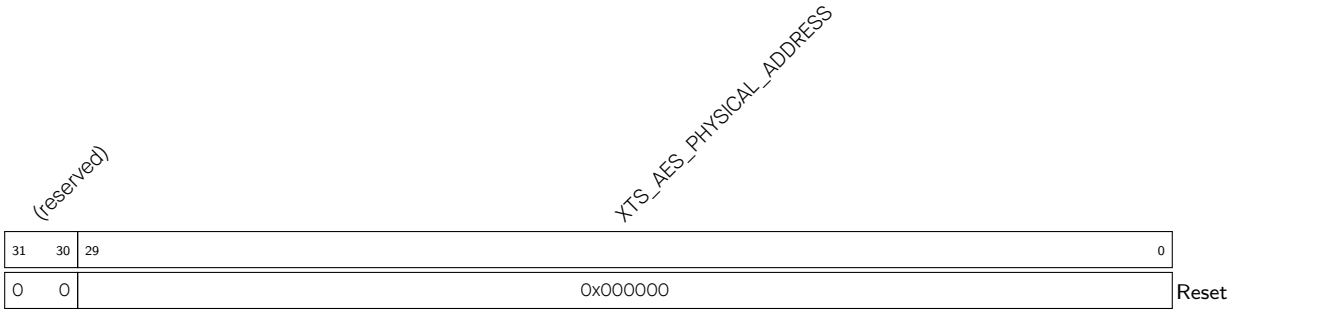

**XTS_AES_DESTINATION** Configures the type of external memory for Manual Encryption. Currently, it must be set to 0, as the Manual Encryption block only supports flash encryption. Set this bit to 1 may cause an error.

0: flash

1: external RAM

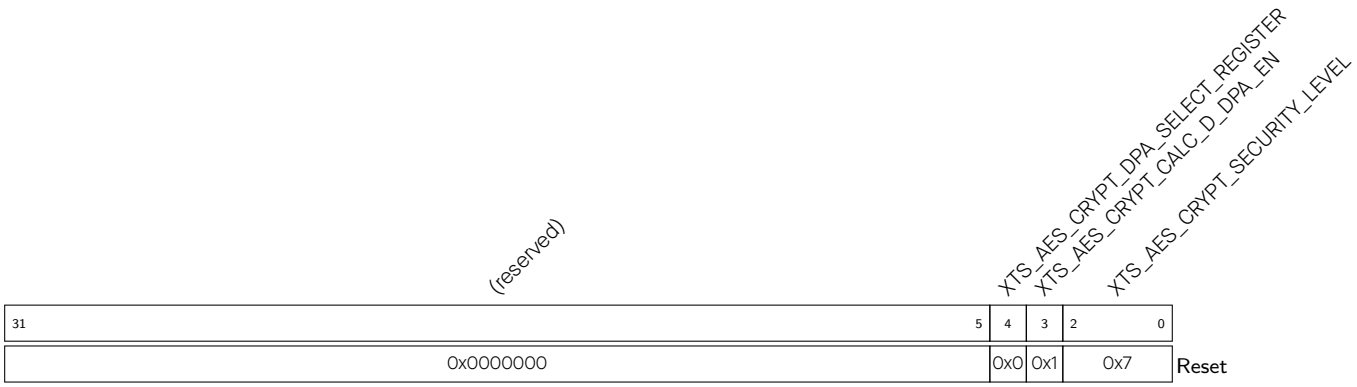
(R/W)

Register 24.4. XTS_AES_PHYSICAL_ADDRESS_REG (0x0348)




**XTS_AES_PHYSICAL_ADDRESS** Configures the physical address which will be used in Manual Encryption. This value should be aligned with the byte number configured via the XTS_AES_LINESIZE parameter. (R/W)

**Register 24.5. XTS_AES_DPA_CTRL_REG (0x0388)**




**XTS_AES_CRYPT_DPA_SELECT_REGISTER** Configures whether the clock Anti-DPA function is controlled by eFuse or register.

0: The clock Anti-DPA function is configured by register.

1: The clock Anti-DPA function is configured by eFuse.

(R/W)

**XTS_AES_CRYPT_CALC_D_DPA_EN** Configures whether to enable the clock Anti-DPA in the XTS_AES algorithm.

0: Enable the clock Anti-DPA only when calculating Tweak value

1: Enable the clock Anti-DPA both when calculating Tweak value and Date units

Note that this field is only effective when XTS_AES_CRYPT_SECURITY_LEVEL is not 0.

(R/W)

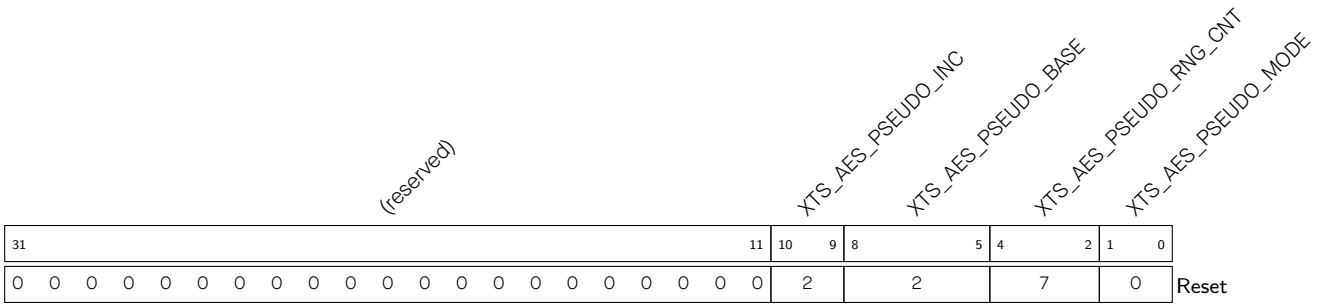
**XTS_AES_CRYPT_SECURITY_LEVEL** Configures the security level of external memory encryption and decryption.

0: Disable the clock Anti-DPA function

1-7: The bigger the number is, the more secure the encryption and decryption are

(R/W)

Register 24.6. XTS_AES_PSEUDO_ROUND_CONF_REG (0x038C)




**XTS_AES_MODE_PSEUDO** Set the calculation steps of XTS-AES to enable pseudo-round Anti-DPA function.

0: not enable the pseudo-round Anti-DPA function during calculation

1: enable the pseudo-round Anti-DPA function when calculating Tweak value

2: enable the pseudo-round Anti-DPA function when calculating Tweak value and some rounds of calculating Data units

3: enable the pseudo-round Anti-DPA function when calculating Tweak value and all rounds of calculating Data units

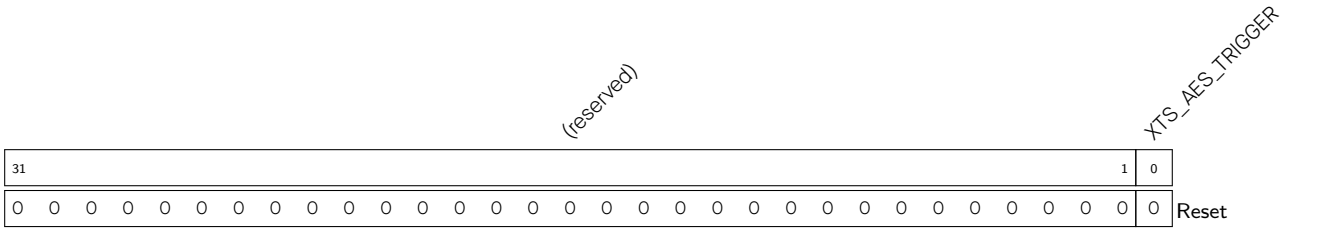
(R/W)

**XTS_AES_PSEUDO_RNG_CNT** Configures the update frequency of the pseudo-key in the pseudo-round Anti-DPA function. (R/W)

**XTS_AES_PSEUDO_BASE** Configures the basic round number of pseudo-round Anti-DPA function. (R/W)

**XTS_AES_PSEUDO_INC** Configures the incremental round number of pseudo-round Anti-DPA function. (R/W)

Register 24.7. XTS_AES_TRIGGER_REG (0x034C)



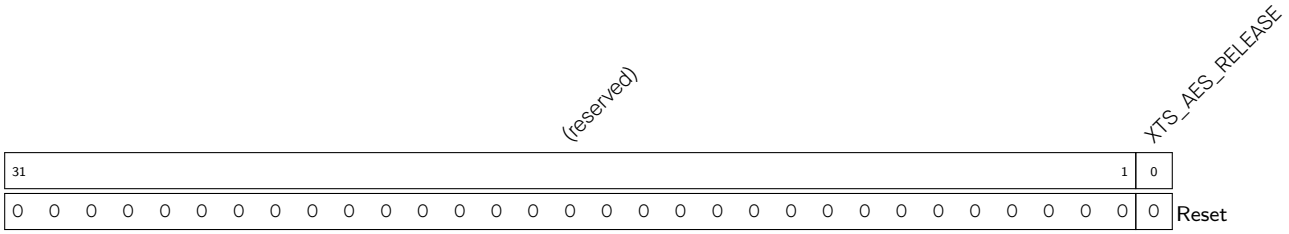

**XTS_AES_TRIGGER** Set this bit to trigger the process of manual encryption calculation.

0: Disable manual encryption

1: Enable manual encryption

This action should only be asserted when manual encryption status is 0. After this action, manual encryption status becomes 1. After calculation is done, manual encryption status becomes 2. (WT)

Register 24.8. XTS_AES_RELEASE_REG (0x0350)



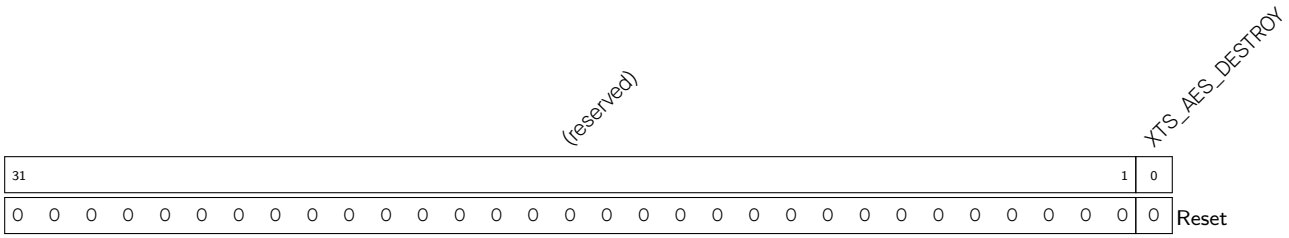

**XTS_AES_RELEASE** Set this bit to release encrypted result to MSPI.

0: do not release

1: release

This action should only be asserted when manual encryption status is 2. After this action, manual encryption status will become 3. (WT)

Register 24.9. XTS_AES_DESTROY_REG (0x0354)



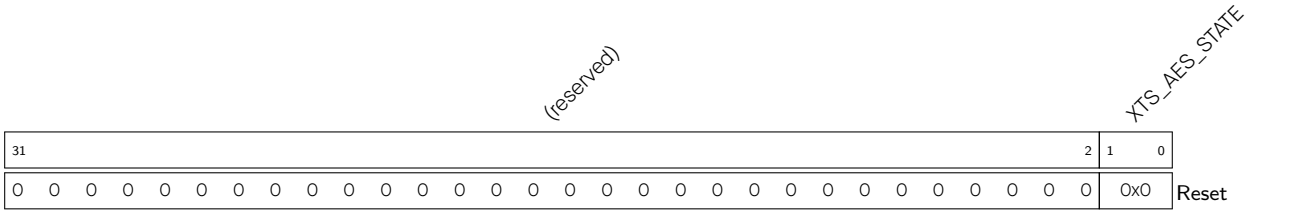

**XTS_AES_DESTROY** Set this bit to destroy encrypted result.

0: No effect

1: Destroy encrypted result

This action should be asserted only when manual encryption status is 3. After this action, manual encryption status will become 0. (WT)

Register 24.10. XTS_AES_STATE_REG (0x0358)




**XTS_AES_STATE** Represents the status of the Manual Encryption block. 0 (XTS_AES_IDLE): Idle

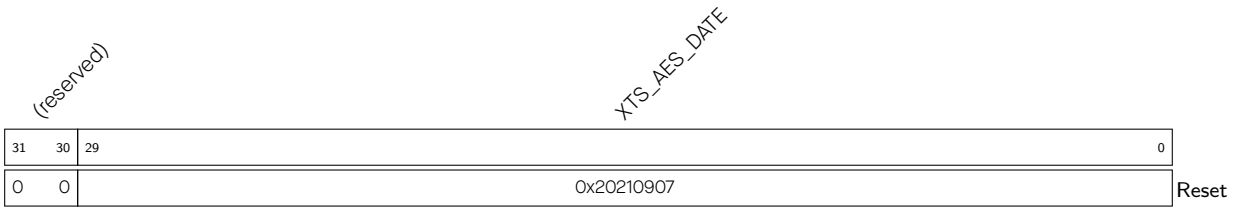
1 (XTS_AES_BUSY): Busy with encryption

2 (XTS_AES_DONE): Encryption completed, but the encrypted result is not accessible to MSPI

3 (XTS_AES_RELEASE): Encrypted result is accessible to MSPI

(RO)

**Register 24.11. XTS_AES_DATE_REG (0x035C)**




**XTS_AES_DATE** Version control register. (R/W)

# Chapter 25

# Random Number Generator (RNG)

## 25.1 Introduction

The ESP32-C61 contains a true random number generator, which generates 32-bit random numbers that can be used for cryptographical operations, among other things.

## 25.2 Features

The random number generator in ESP32-C61 generates true random numbers, meaning they are derived from a physical process rather than an algorithm. Each number within the specified range has an equal probability of occurrence.

## 25.3 Functional Description

Every 32-bit value that the system reads from the LPPERI_RNG_DATA_SYNC_REG register of the random number generator is a true random number. These true random numbers are generated based on the **thermal noise** in the system, the **asynchronous clock mismatch**, and the **asynchronous counter**.

- **Thermal noise** comes from the high-speed ADC, or SAR ADC, or both. Whenever the high-speed ADC or 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** source that can generate metastability in the circuit. This metastability can also be used as a random number seed [2] to feed into the random number generator.
- **Asynchronous counters** count using an asynchronous clock source. The asynchronous clock source generates circuit metastability, which can be used as a random number seed to feed into the random number generator. There are two types of asynchronous counters:
  - On-chip RTC Timer **(RTC_TIMER)**: The counter directly counts using the on-chip RTC timer. The count value is fed into the random number generator as a random number seed after applying XOR logic.
  - **Ring oscillator (BUF_CHAIN)**: The counter counts based on the ring oscillator, formed by connecting buffer cells end-to-end. The count value derived from this oscillator is fed into the random number generator as a random number seed after applying XOR logic.

  These two types of asynchronous counters can work independently or in combination.

When there is noise coming from the SAR ADC, the random number generator is fed with a 2-bit entropy in one clock cycle (configurable, normally 66 kHz). Thus, it is advisable to read the

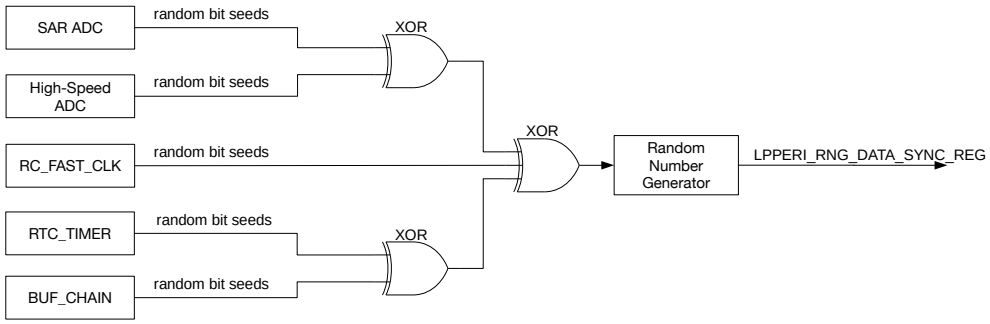



Figure 25.3-1. Noise Source

LPPERI_RNG_DATA_SYNC_REG register at a maximum rate of 66 kHz to obtain the maximum entropy.

When there is noise coming from the high-speed ADC, the random number generator is fed with a 2-bit entropy in one APB clock cycle, which is normally 80 MHz. Thus, it is advisable to read the LPPERI_RNG_DATA_SYNC_REG register at a maximum rate of 5 MHz to obtain the maximum entropy.

When BUF_CHAIN is enabled, 1 bit of entropy can be obtained in each RC_FAST_CLK cycle.

A data sample of 2 GB, which is read from the random number generator at a rate of 5 MHz with only the high-speed ADC being enabled, has been tested using the Dieharder Random Number Testsuite (version 3.31.1). The sample passed all tests.

## 25.4 Programming Procedure

To use the random number generator, set LPPERI_RNG_CK_EN to enable its clock. The clock is automatically enabled when ECDSA Digital Signature Peripheral (ECDSA_DS) is used, regardless of LPPERI_RNG_CK_EN.

When using the random number generator, make sure the entropy source is enabled. Otherwise, pseudo-random numbers will be returned.

- SAR ADC can be enabled by using the DIG ADC controller. For details, please refer to Chapter 34 *ADC Controller*.
- High-speed ADC is enabled automatically when the Wi-Fi or Bluetooth module is enabled.
- RC_FAST_CLK is always enabled
- Asynchronous counters
  - RTC timer (RTC_TIMER) is enabled by setting the LPPERI_RTC_TIMER_EN field.
  - Ring oscillator (BUF_CHAIN) is enabled by setting the LPPERI_RNG_SAMPLE_ENABLE bit.

**Note:**

1. When the Wi-Fi module is enabled, the high-speed ADC may produce saturated readings in extreme cases, which reduces entropy. To mitigate this, enable the SAR ADC as an additional noise source for the random number generator.
2. Enabling RC_FAST_CLK and asynchronous counters increases the RNG entropy. However, to ensure maximum entropy, always enable the SAR ADC or high-speed ADC as well.

When multiple random numbers are required, the value of the LPPERI_RNG_DATA_SYNC_REG register can be read continuously. However, when reading the register, ensure that the rate does not exceed the entropy generation rate described in Section 25.3. If multiple entropy sources are configured, it is recommended to not exceed the entropy generation rate of the lowest noise source.

## 25.5 Register Summary

The addresses in this section are relative to Random Number Generator base address provided in Table 5.3-2 in Chapter 5 *System and Memory*.

The abbreviations given in Column **Access** are explained in Section *Access Types for Registers*.

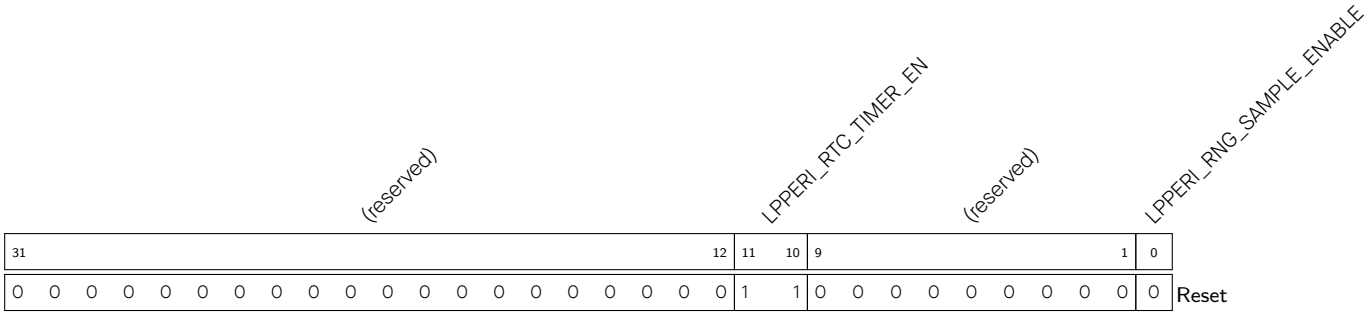
| Name | Description | Address | Access |
|---|---|---|---|
| LPPERI_RNG_CFG_REG | RNG configuration register | 0x0024 | varies |
| LPPERI_RNG_DATA_SYNC_REG | RNG result synchronization register | 0x0028 | RO |

## 25.6 Registers

The addresses in this section are relative to Random Number Generator base address provided in Table 5.3-2 in Chapter 5 *System and Memory*.

For how to program reserved fields, please refer to Section *Programming Reserved Register Field*.

**Register 25.1. LPPERI_RNG_CFG_REG (0x0024)**




**LPPERI_RNG_SAMPLE_ENABLE** Configures whether to enable BUF_CHAIN.

0: Disable

1: Enable

(R/W)

**LPPERI_RTC_TIMER_EN** Configures whether to enable the RTC timer before CRC.

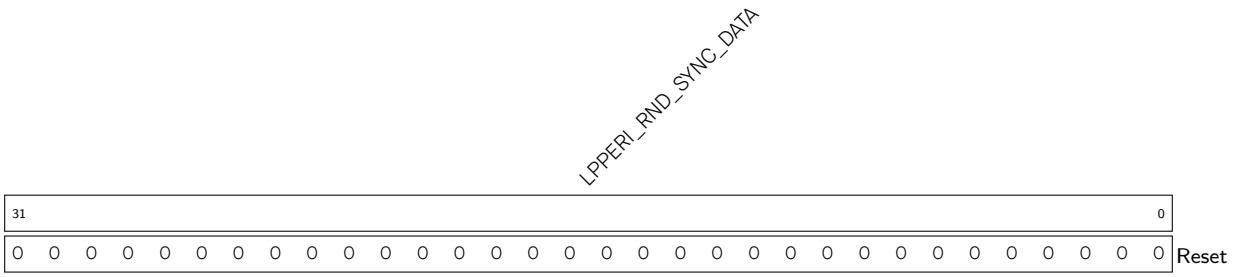
0: Disable

1: Enable

(R/W)

**LPPERI_RNG_SAMPLE_CNT** Represents the count value of BUF_CHAIN. (RO)

**Register 25.2. LPPERI_RNG_DATA_SYNC_REG (0x0028)**




**LPPERI_RND_SYNC_DATA** Represents the RNG synchronization result. (RO)

# Part V

# 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 LED control.

# Chapter 26

# UART Controller (UART)

## 26.1 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-C61 support various lengths of data bits and stop bits. This allows developers to use multiple UART ports at minimal software cost.

ESP32-C61 has two UART controllers. These UARTs are compatible with various UART devices, and support Infrared Data Association (IrDA) and RS485 communication. In this chapter, the two regular UART controllers are referred to as UART*n*, in which *n* denotes 0, 1.

## 26.2 Features

UART controllers 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
- 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
- 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

## 26.3 UART Structure

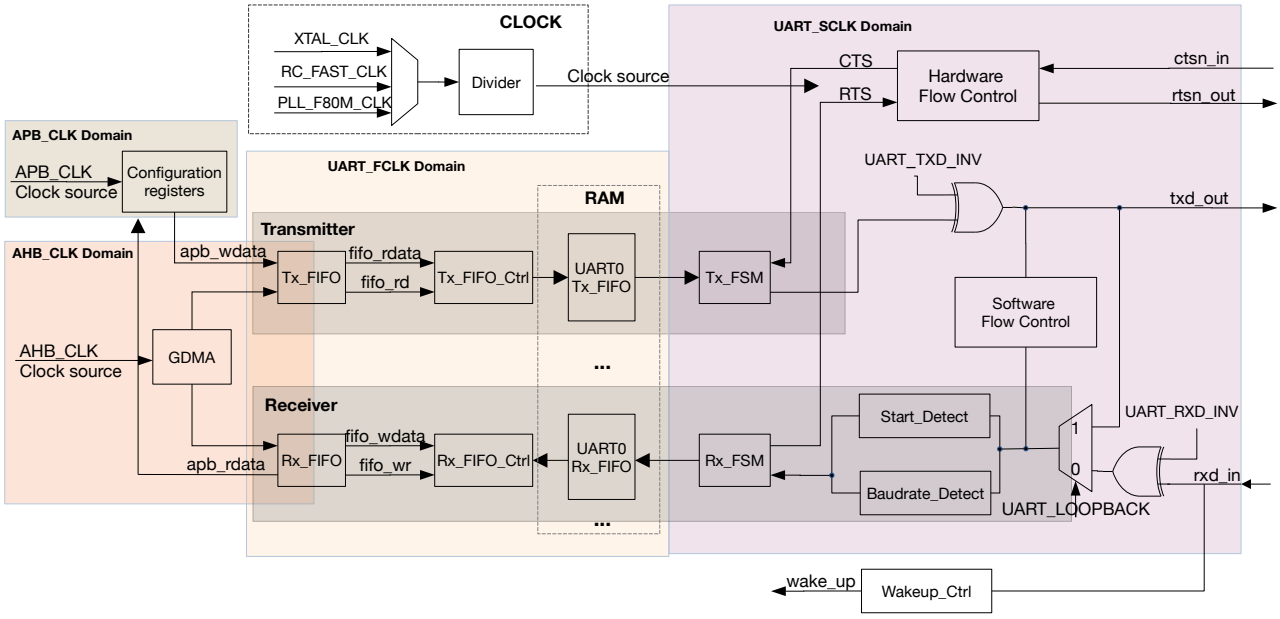



Figure 26.3-1. UART Structure

Figure 26.3-1 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 8 *Reset and Clock*), which are selected by configuring PCR_UART*n*_SCLK_SEL. The selected clock source is divided by a divider to generate UART_SCLK clock signals. The divisor is configured by PCR_UART*n*_SCLK_DIV_NUM for the integral part, PCR_UART*n*_SCLK_DIV_A for the denominator of the fractional part, and PCR_UART*n*_SCLK_DIV_B for the numerator of the fractional part. The divisor ranges from 1 ~ 256.

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 26.3-1), which buffers data to be sent. Software can write data to Tx_FIFO via the APB bus. 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 26.3-1), 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.

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 12 *Low-Power Management* for more details), a wake_up signal can be generated in four ways and sent to PMU, which then wakes up the ESP32-C61 chip. For more information about wakeup, please refer to Section 26.4.8.

# 26.4 Functional Description

## 26.4.1 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 Section26.6.

To reset the whole UART, please:

- Enable the UART Core's clock by setting PCR_UART*n*_CLK_EN to 1.
- Write 1 to PCR_UART*n*_RST_EN.
- Clear PCR_UART*n*_RST_EN to 0.

## 26.4.2 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, 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.

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.

## 26.4.3 Baud Rate Generation and Detection

### 26.4.3.1 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 26.4-1, 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 26.4-1 below, to make the output timing more uniform:

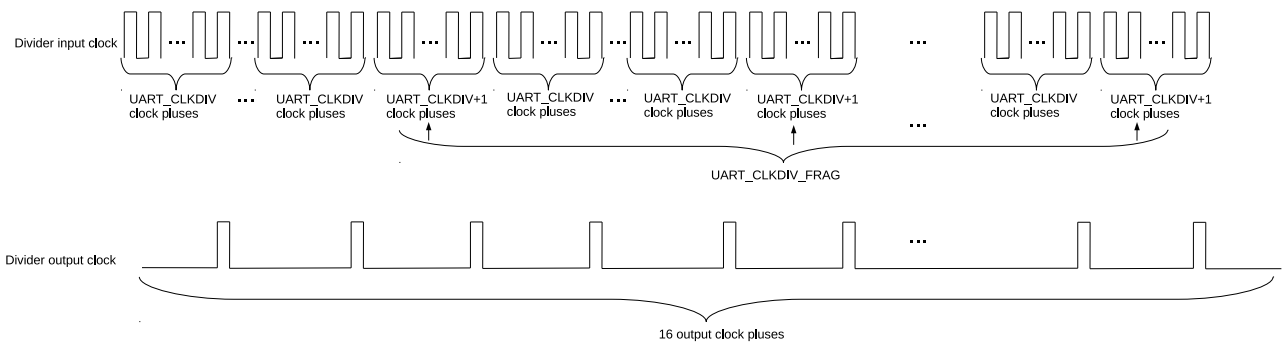



Figure 26.4-1. UART Controllers Division

To support IrDA (see Section 26.4.7 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.

#### 26.4.3.2 Baud Rate Detection

Automatic baud rate detection (Autobaud) on UARTs is enabled by setting UART_AUTOBAUD_EN. The Baudrate_Detect module shown in Figure 26.4-2 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.

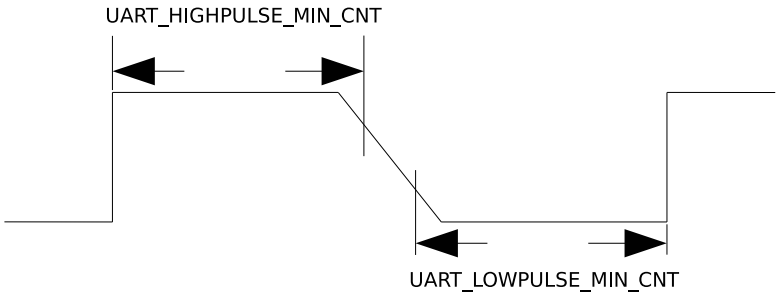



Figure 26.4-2. The Timing Diagram of Weak UART Signals Along Negative Edges

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:

$$B_{\text{uart}} = \frac{f_{\text{clk}}}{(\text{UART\_LOWPULSE\_MIN\_CNT} + \text{UART\_HIGHPULSE\_MIN\_CNT} + 2)/2}$$

2. If UART signals are weak along negative edges as shown in Figure 26.4-2, 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:

$$B_{\text{uart}} = \frac{f_{\text{clk}}}{(\text{UART\_POSEDGE\_MIN\_CNT} + 1)/2}$$

3. If UART signals are weak along positive edges, use UART_NEGEDGE_MIN_CNT to determine the transmitter's baud rate as follows:

$$B_{\text{uart}} = \frac{f_{\text{clk}}}{(\text{UART\_NEGEDGE\_MIN\_CNT} + 1)/2}$$

### 26.4.4 UART Data Frame

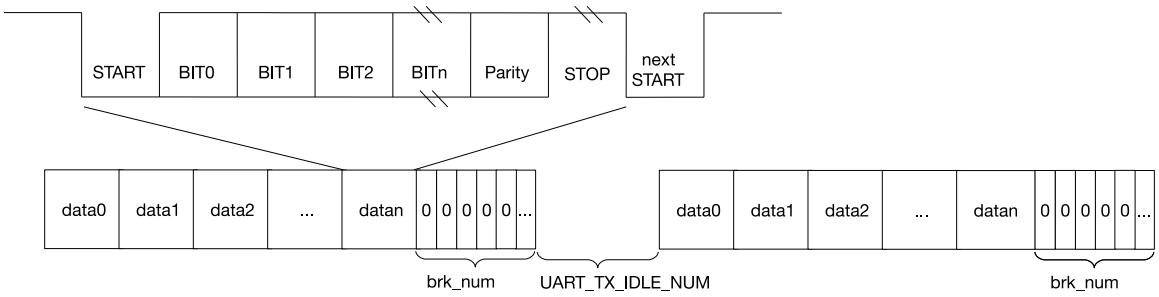



Figure 26.4-3. Structure of UART Data Frame

Figure 26.4-3 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 26.4.6.2). 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. After the NULL characters are sent, 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.

### 26.4.5 AT_CMD Character Structure

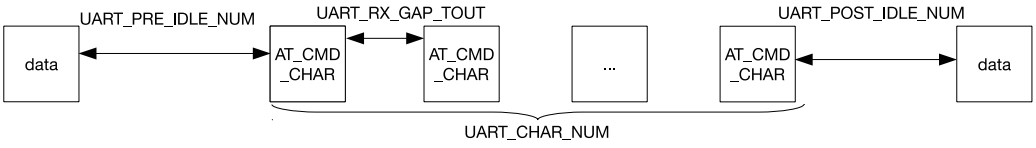



Figure 26.4-4. AT_CMD Character Structure

Figure 26.4-4 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.

## 26.4.6 RS485

The two 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.

### 26.4.6.1 Driver Control

As shown in Figure 26.4-5, 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 26.4-5, DE is connected to dtrn_out of UART (please refer to Section 26.4.9.1 for more details).

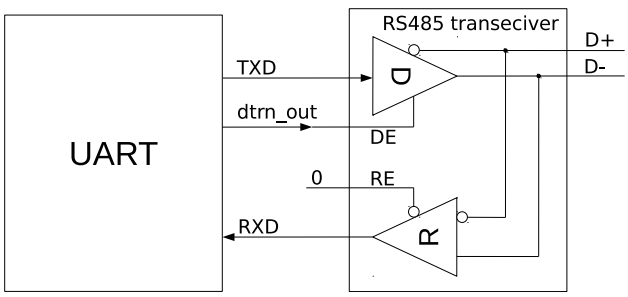



Figure 26.4-5. Driver Control Diagram in RS485 Mode

#### 26.4.6.2 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.

#### 26.4.6.3 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 26.4-5, 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.

### 26.4.7 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 26.4-6, 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.

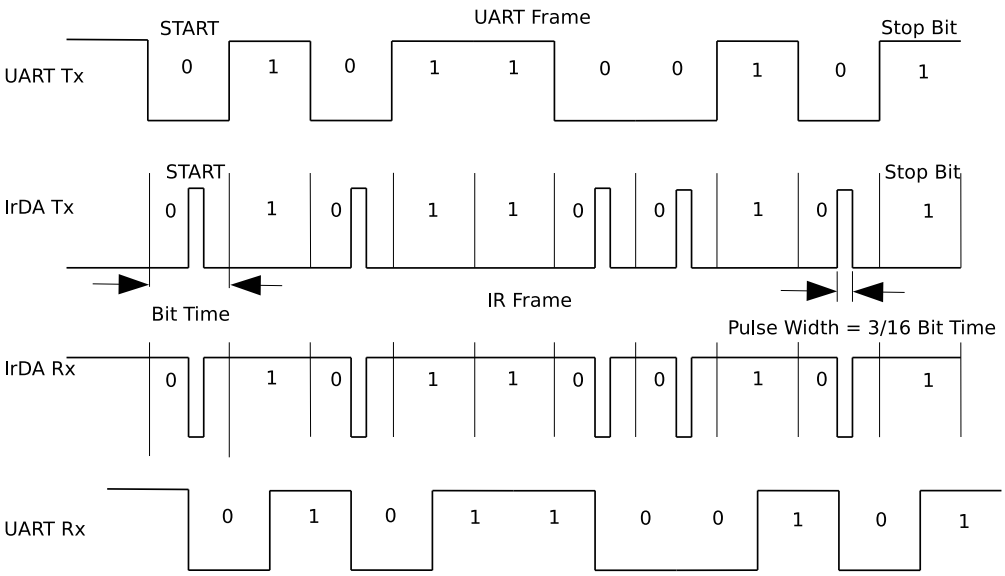



Figure 26.4-6. The Timing Diagram of Encoding and Decoding in SIR mode

The IrDA transceiver is half-duplex, meaning that it cannot send and receive data simultaneously. As shown in Figure 26.4-7, 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.

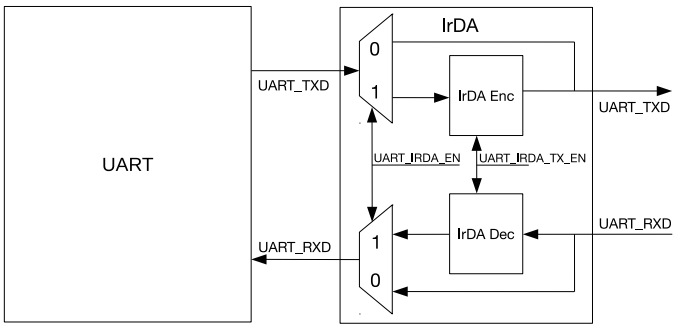



Figure 26.4-7. IrDA Encoding and Decoding Diagram

## 26.4.8 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 PMU module, which then wakes up ESP32-C61.

- 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 26.4-1. Once the sequence is detected, the chip will be woken up. For the last configuration in Table 26.4-1, UART will detect for CHAR0 ~ CHAR4 in order.

**Table 26.4-1. 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.

## 26.4.9 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.

### 26.4.9.1 Hardware Flow Control

Figure 26.4-8 shows the hardware flow control of a UART controller. Hardware flow control uses output signal rtsn_out and input signal dsrn_in. Figure 26.4-9 illustrates how these signals are connected between UART on ESP32-C61 (hereinafter referred to as IU0) and the external UART (hereinafter referred to as EU0).

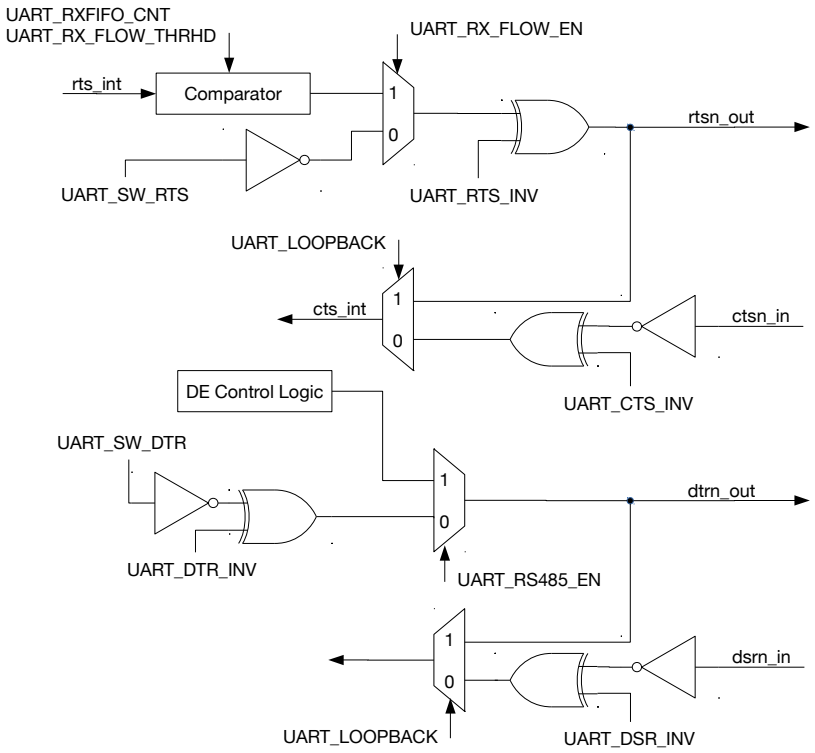



Figure 26.4-8. Hardware Flow Control Diagram

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.

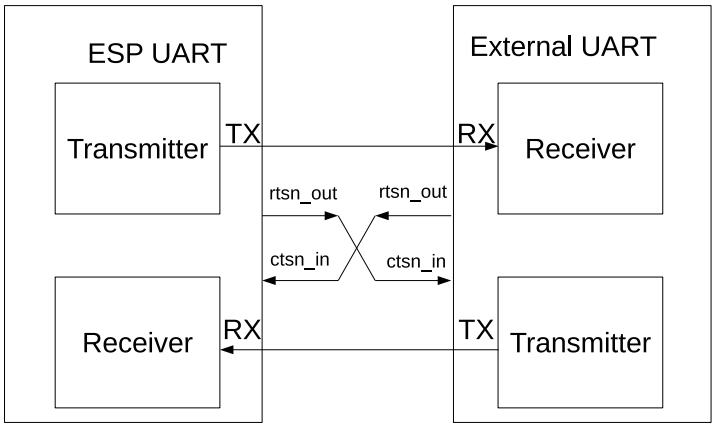



Figure 26.4-9. Connection between Hardware Flow Control Signals

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.

#### 26.4.9.2 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.

## 26.5 Interrupts

ESP32-C61's UART*n* can generate the following interrupt signals that will be sent to the Interrupt Matrix.

- UART*n*_INTR

There are several internal interrupt sources from UART*n* 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.

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 26.7 *Register Summary*.

## 26.6 Programming Procedures

### 26.6.1 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 26.7. Those without the _SYNC suffix in Section 26.7 are configuration registers that require no clock domain crossing.

## 26.6.2 Detailed Steps

Figure 26.6-1 illustrates the process to program UART controllers, namely initialize UART, configure registers, enable the UART transmitter or receiver, and finish data transmission.

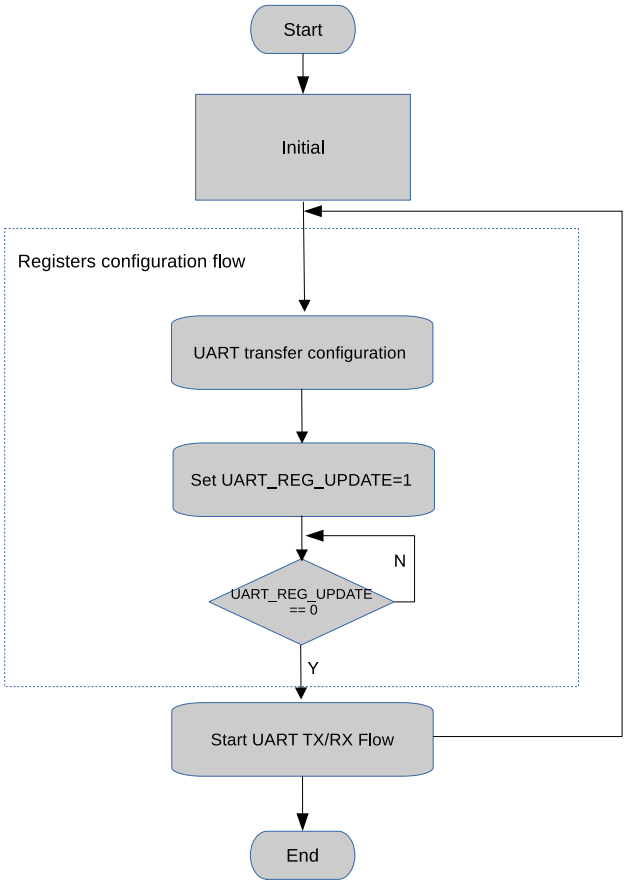



Figure 26.6-1. UART Programming Procedures

### 26.6.2.1 Initializing UART*n*

To initialize UART*n*:

- Enable APB_CLK for UART*n* by setting PCR_UART*n*_CLK_EN to 1.
- Write 1 to PCR_UART*n*_RST_EN.
- Clear PCR_UART*n*_RST_EN to 0.

### 26.6.2.2 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 PCR_UART*n*_SCLK_SEL.
- Configure divisor of the divider via PCR_UART*n*_SCLK_DIV_NUM, PCR_UART*n*_SCLK_DIV_A, and PCR_UART*n*_SCLK_DIV_B.
- 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.

#### 26.6.2.3 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.

# 26.7 Register Summary

## 26.7.1 UART Register Summary

The addresses in this section are relative to UART Controller base address provided in Table 5.3-2 in Chapter 5 *System and Memory*.

The abbreviations given in Column **Access** are explained in Section *Access Types for Registers*.

| Name | Description | Address | Access |
|---|---|---|---|
| **FIFO Configuration Register** | | | |
| 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 Register** | | | |
| 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 |

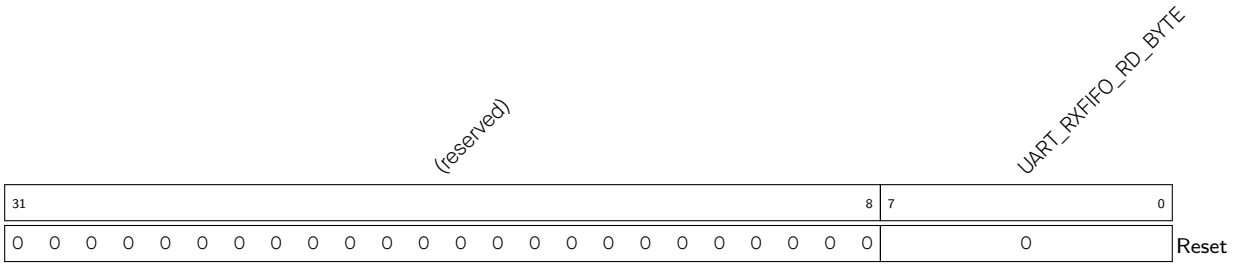
# 26.8 Registers

## 26.8.1 UART Registers

The addresses in this section are relative to UART Controller base address provided in Table 5.3-2 in Chapter 5 *System and Memory*.

For how to program reserved fields, please refer to Section *Programming Reserved Register Field*.

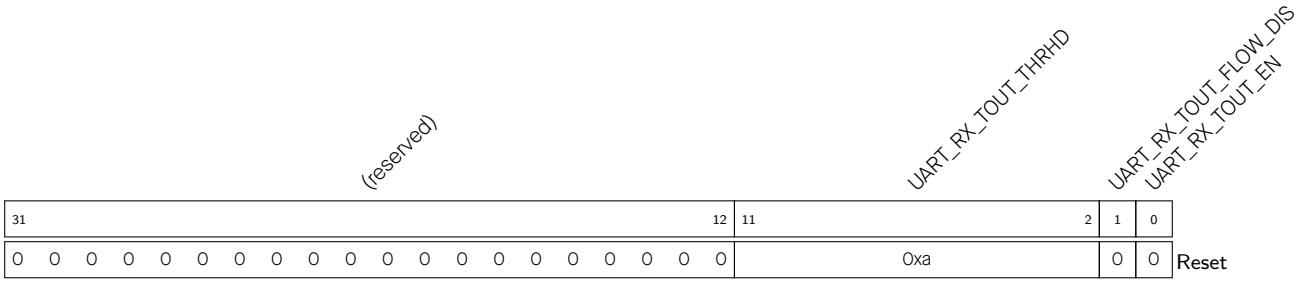
**Register 26.1. UART_FIFO_REG (0x0000)**




**UART_RXFIFO_RD_BYTE** Represents the data UART*n* reads from FIFO.

Measurement unit: byte. (RO)

**Register 26.2. 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

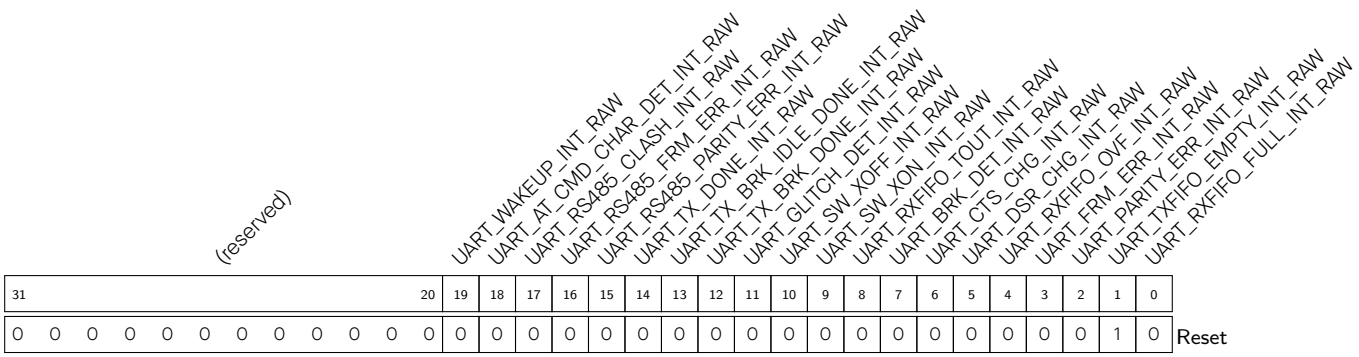
1: Disable

(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 26.3. 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 26.3. UART_INT_RAW_REG (0x0004)

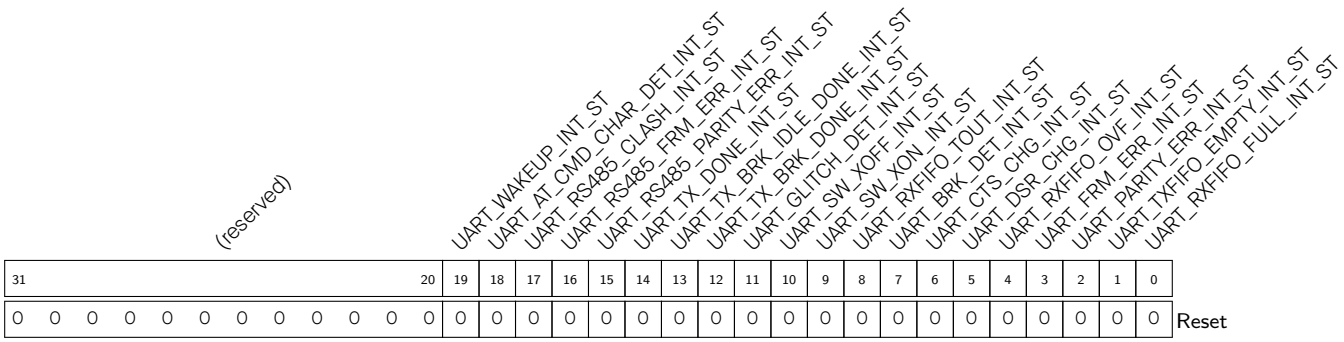
Continued from the previous page...

**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 26.4. 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 26.5. 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)

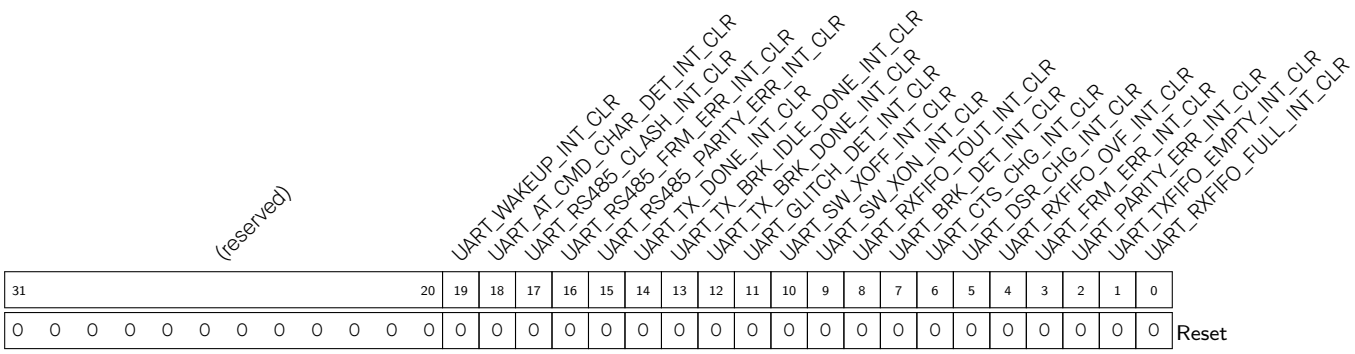
**UART_RS485_FRM_ERR_INT_ENA** Write 1 to enable UART_RS485_FRM_ERR_INT. (R/W)

**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 26.6. 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)

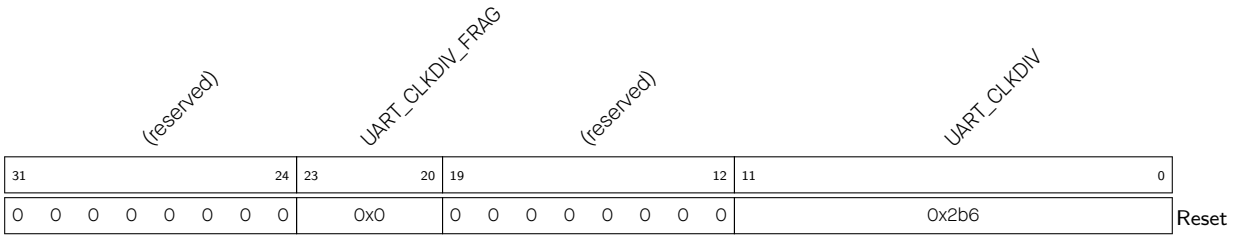
**UART_RS485_FRM_ERR_INT_CLR** Write 1 to clear UART_RS485_FRM_ERR_INT. (WT)

**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)

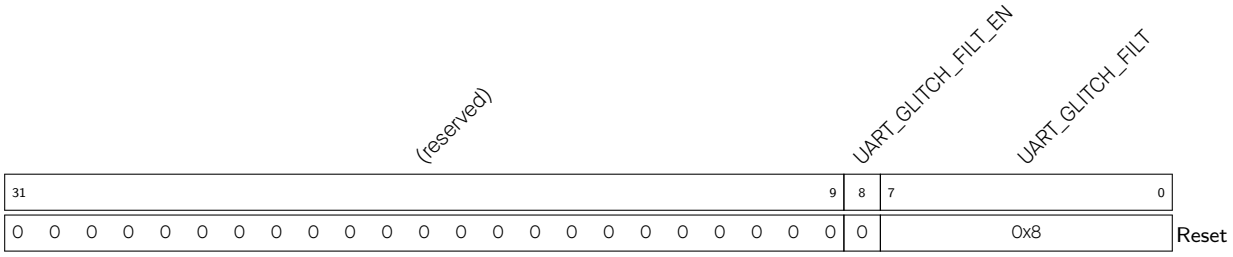
Register 26.7. UART_CLKDIV_SYNC_REG (0x0014)




**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 26.8. 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)

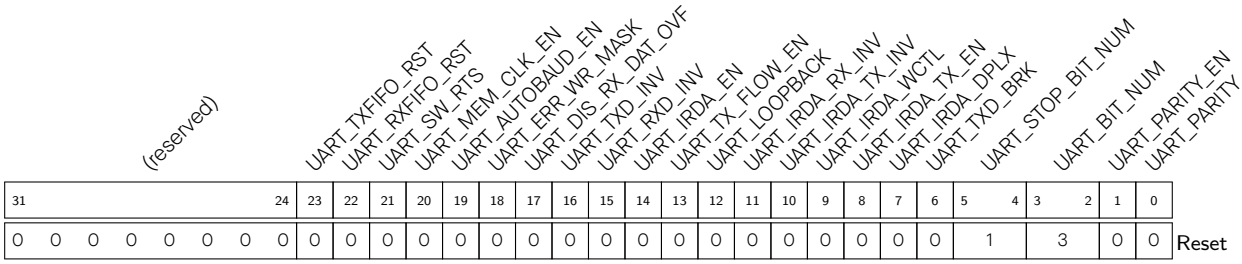
**UART_GLITCH_FILT_EN** Configures whether or not to enable RX signal filter.

0: Disable

1: Enable

(R/W)

Register 26.9. 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 26.9. 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 26.9. 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)

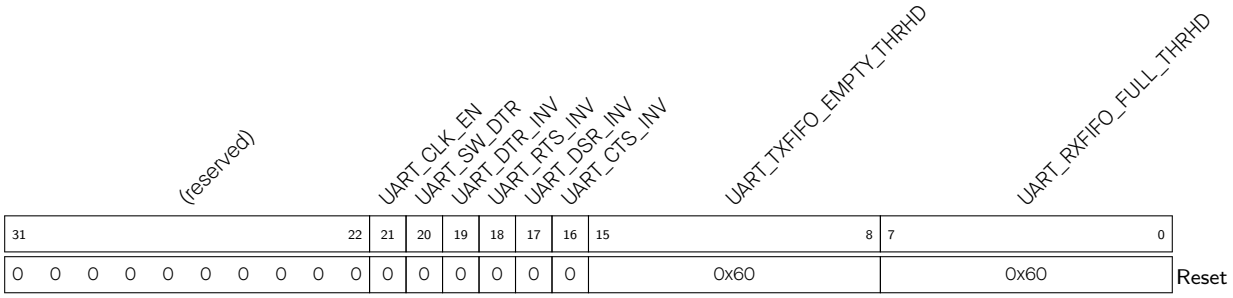
**UART_TXFIFO_RST** Configures whether or not to reset the UART TX FIFO.

0: Not reset

1: Reset

(R/W)

Register 26.10. 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)

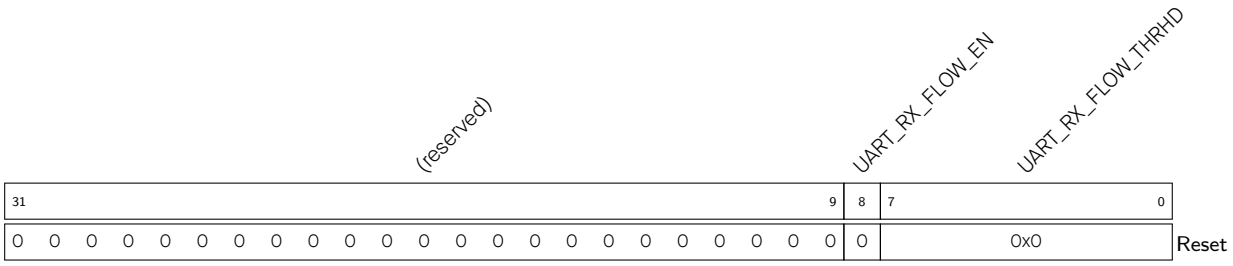
**UART_CLK_EN** Configures clock gating.

0: Support clock only when the application writes registers.

1: Always force the clock on for registers.

(R/W)

Register 26.11. 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)

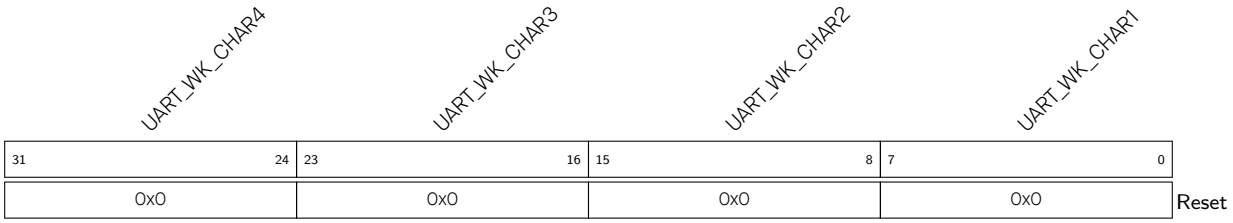
**UART_RX_FLOW_EN** Configures whether or not to enable the UART receiver.

0: Disable

1: Enable

(R/W)

Register 26.12. UART_SLEEP_CONF0_REG (0x0030)



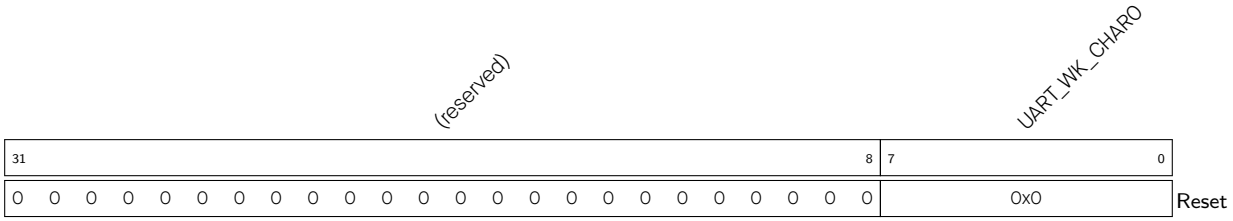

**UART_WK_CHAR1** Configures wakeup character 1. (R/W)

**UART_WK_CHAR2** Configures wakeup character 2. (R/W)

**UART_WK_CHAR3** Configures wakeup character 3. (R/W)

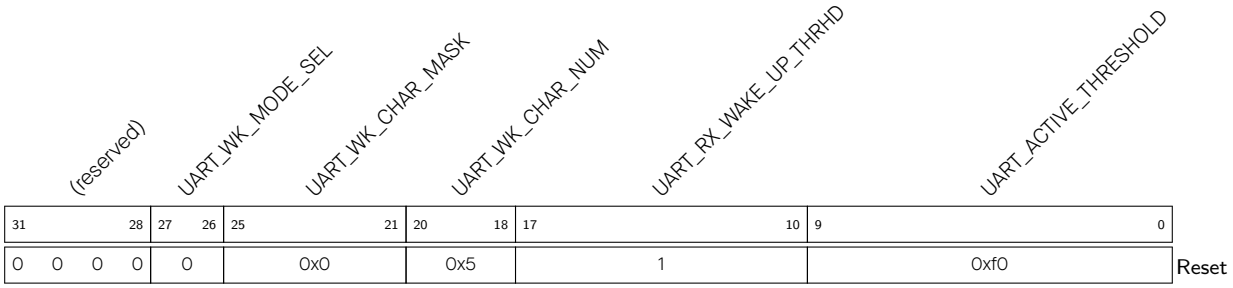
**UART_WK_CHAR4** Configures wakeup character 4. (R/W)

Register 26.13. UART_SLEEP_CONF1_REG (0x0034)




**UART_WK_CHAR0** Configures wakeup character 0. (R/W)

**Register 26.14. 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 26.4.8 *Wakeup* for the explanation of each mode.

0: Mode 0

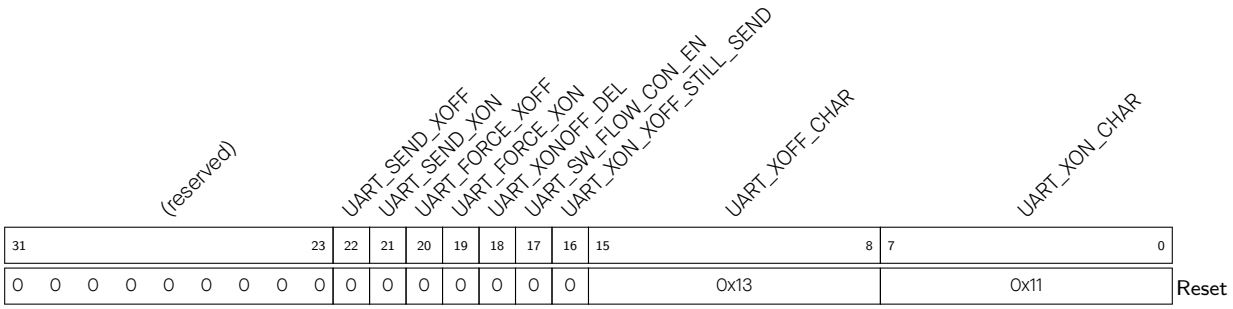
1: Mode 1

2: Mode 2

3: Mode 3

(R/W)

Register 26.15. 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)

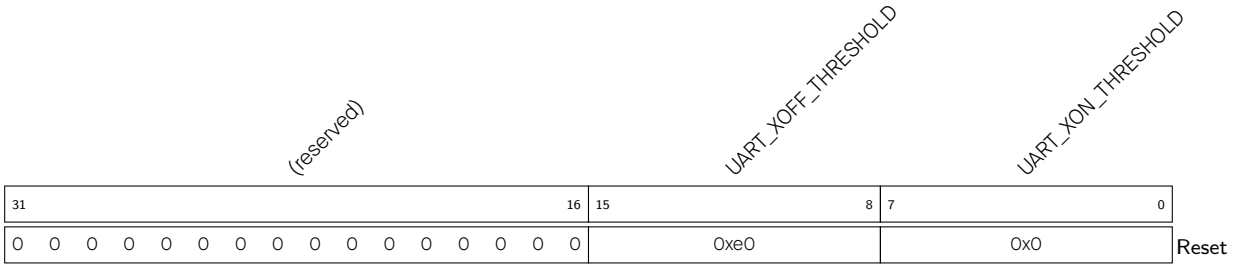
**UART_SEND_XOFF** Configures whether or not to send XOFF characters.

0: Not send

1: Send

(R/W/SS/SC)

Register 26.16. UART_SWFC_CONF1_REG (0x0040)



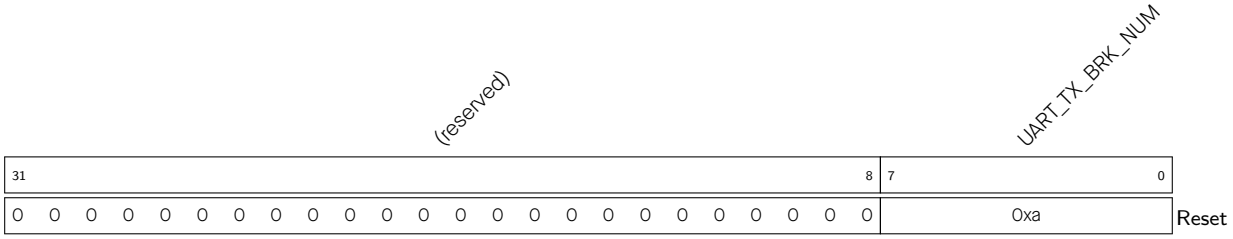

**UART_XON_THRESHOLD** Configures the threshold for data in RX FIFO to send XON characters in software flow control.

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)

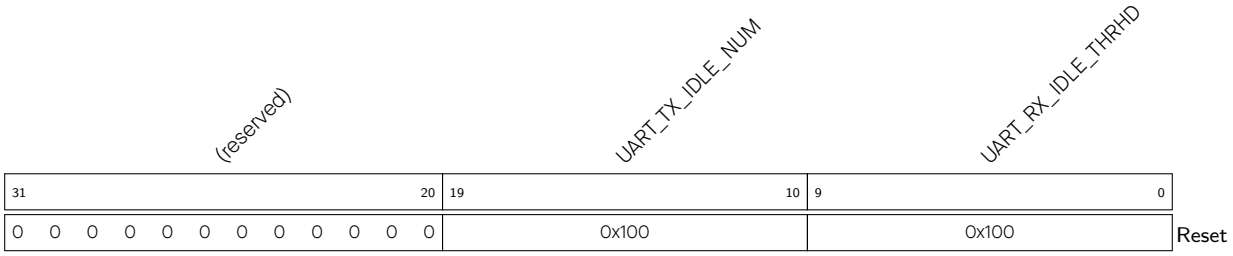
Register 26.17. UART_TXBRK_CONF_SYNC_REG (0x0044)




**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 26.18. UART_IDLE_CONF_SYNC_REG (0x0048)



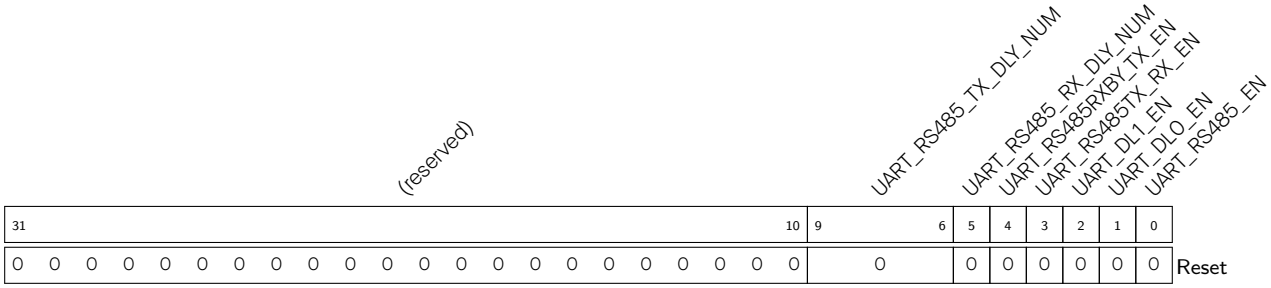

**UART_RX_IDLE_THRHD** Configures the threshold to generate a frame end signal when the receiver takes more time to receive one data byte data.

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 26.19. 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)

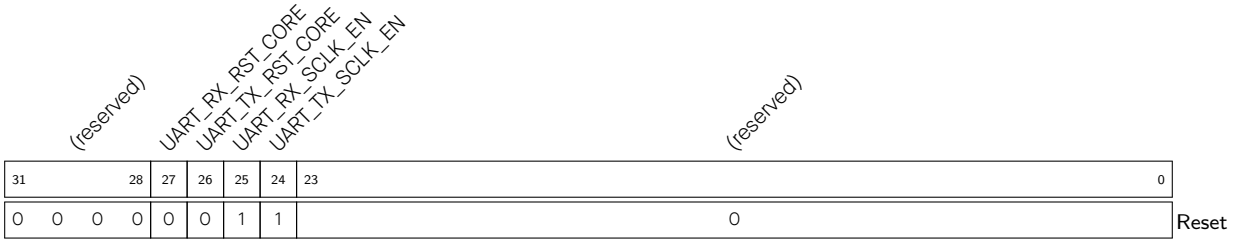
**UART_RS485_RX_DLY_NUM** Configures the delay of internal data signals in the receiver.

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 26.20. 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

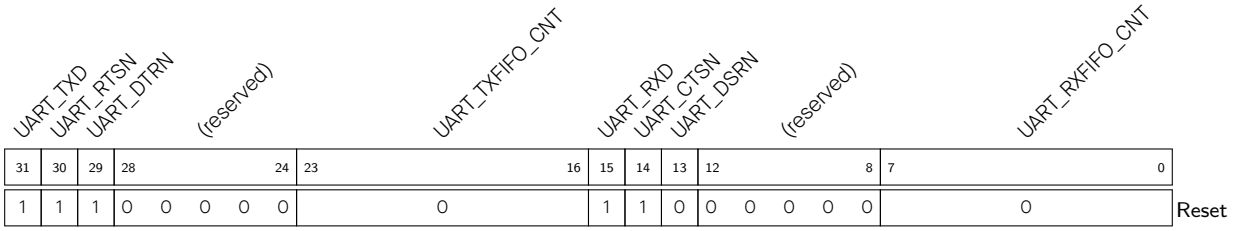
1: Enable

(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 26.21. 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)

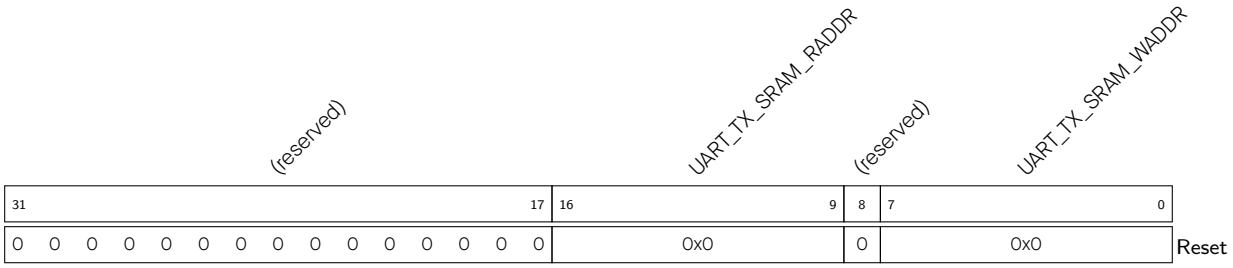
**UART_TXFIFO_CNT** Represents the number of valid data bytes in RX FIFO. (RO)

**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)

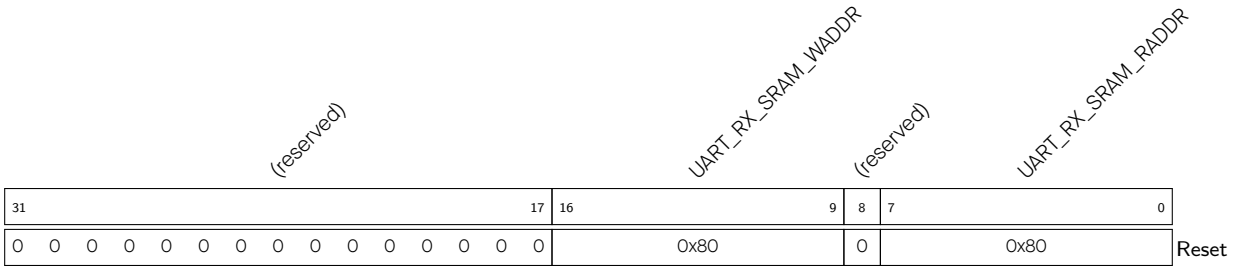
Register 26.22. UART_MEM_TX_STATUS_REG (0x0068)




**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)

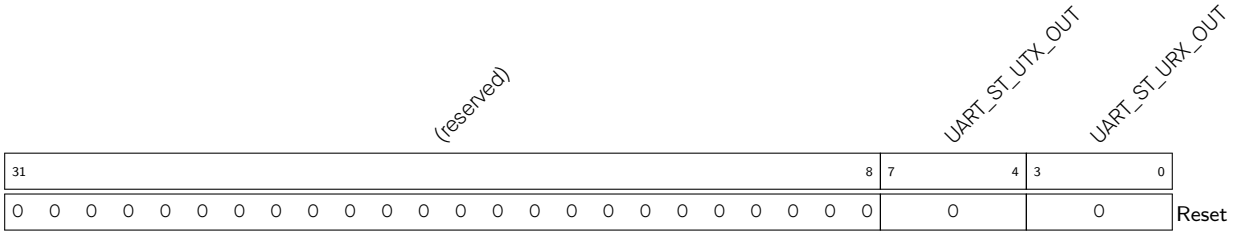
Register 26.23. UART_MEM_RX_STATUS_REG (0x006C)




**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)

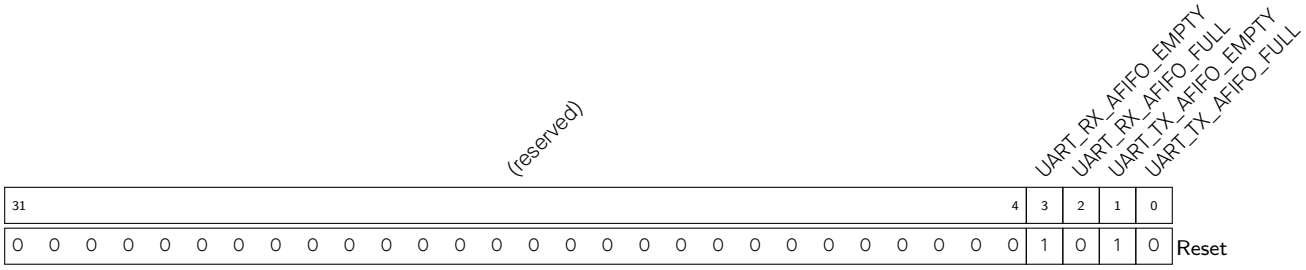
Register 26.24. UART_FSM_STATUS_REG (0x0070)




**UART_ST_URX_OUT** Represents the status of the receiver. (RO)

**UART_ST_UTX_OUT** Represents the status of the transmitter. (RO)

Register 26.25. 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)

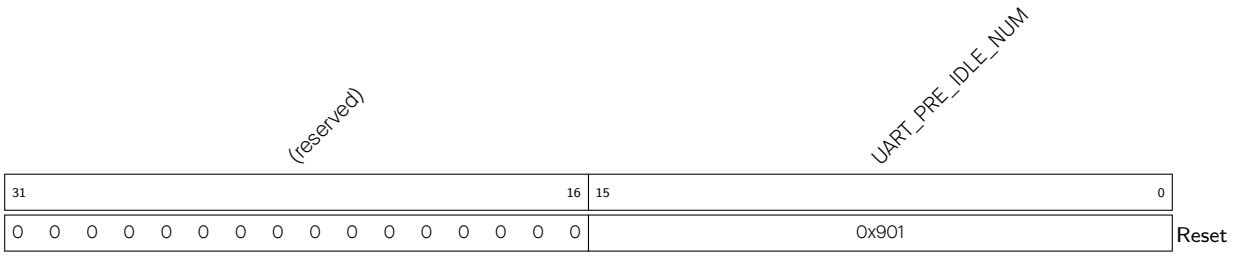
**UART_RX_AFIFO_EMPTY** Represents whether or not the APB RX asynchronous FIFO is empty.

0: Not empty

1: Empty

(RO)

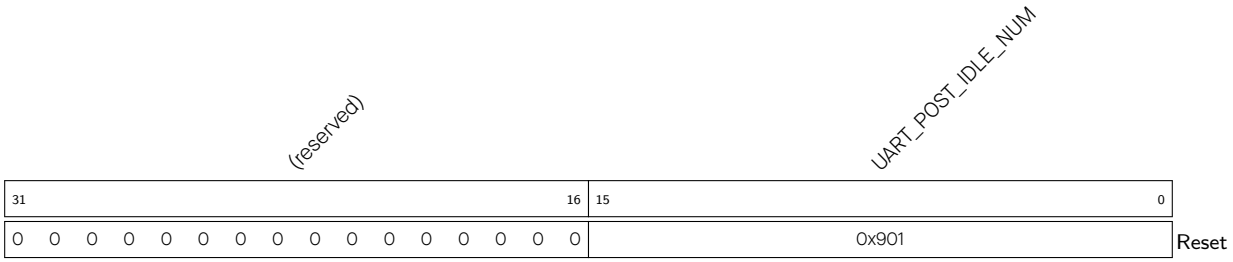
Register 26.26. UART_AT_CMD_PRECNT_SYNC_REG (0x0050)




**UART_PRE_IDLE_NUM** Configures the idle time before the receiver receives the first AT_CMD.

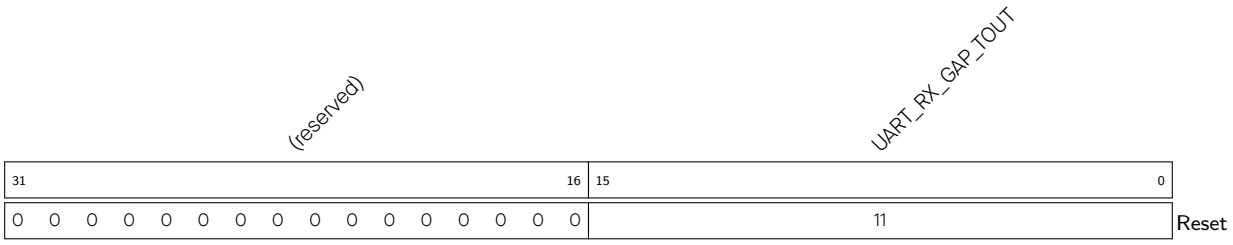
Measurement unit: bit time (the time to transmit 1 bit). (R/W)

Register 26.27. UART_AT_CMD_POSTCNT_SYNC_REG (0x0054)



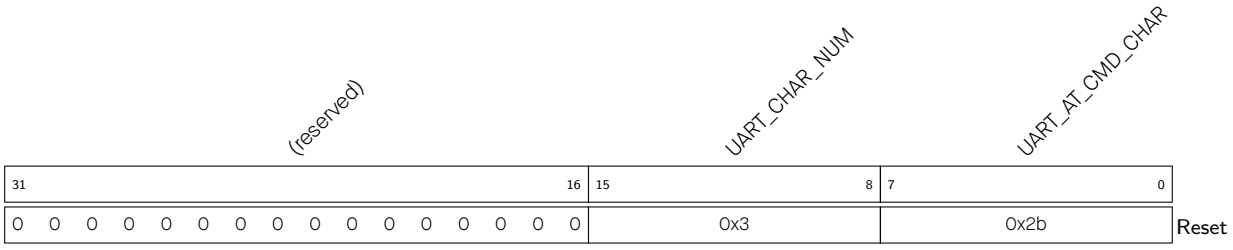

**UART_POST_IDLE_NUM** Configures the interval between the last AT_CMD and subsequent data. Measurement unit: bit time (the time to transmit 1 bit). (R/W)

Register 26.28. 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)

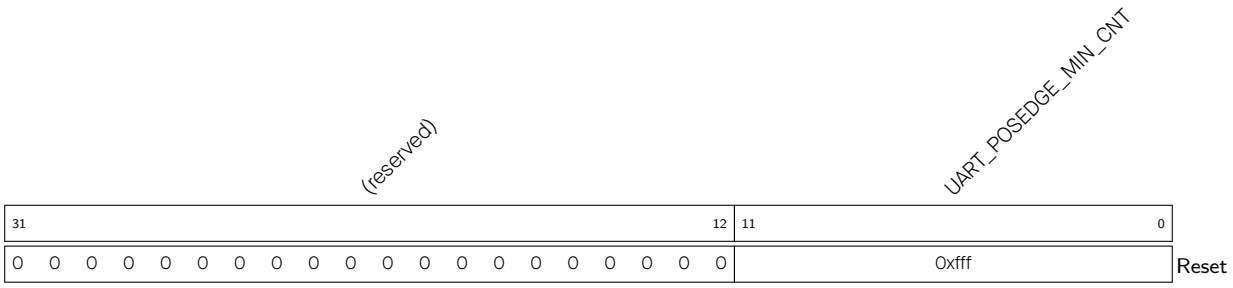
Register 26.29. UART_AT_CMD_CHAR_SYNC_REG (0x005C)




**UART_AT_CMD_CHAR** Configures the AT_CMD character. (R/W)

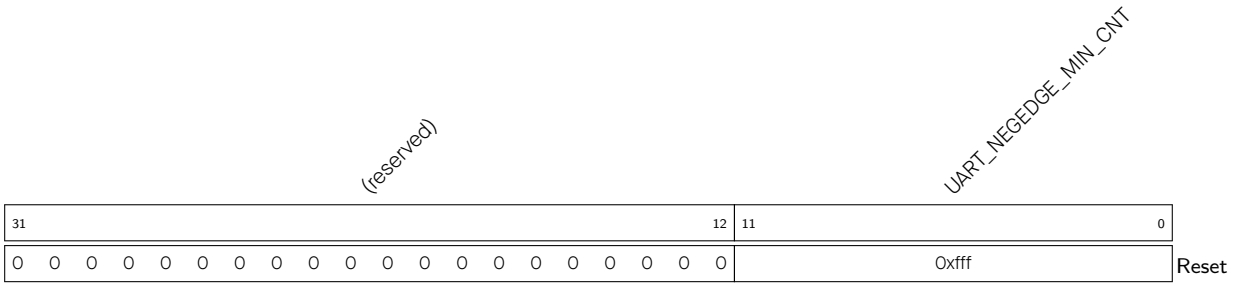
**UART_CHAR_NUM** Configures the number of continuous AT_CMD characters a receiver can receive. (R/W)

Register 26.30. UART_POSPULSE_REG (0x0074)



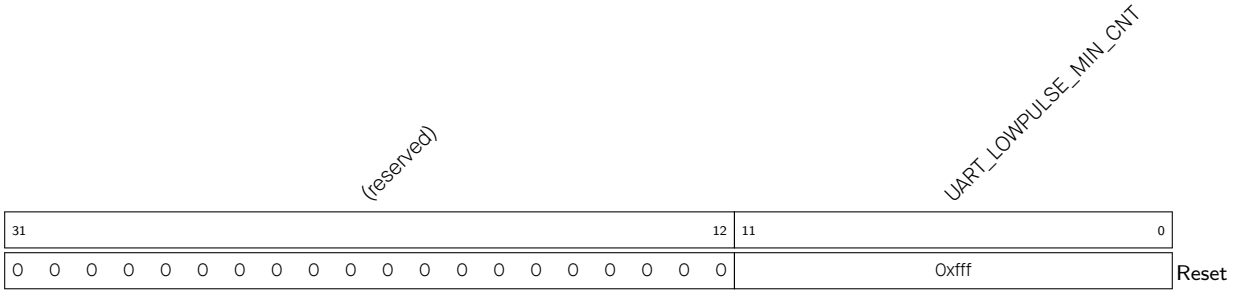

**UART_POSEDGE_MIN_CNT** Represents the minimal input clock counter value between two positive edges. It is used for baud rate detection. (RO)

Register 26.31. 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)

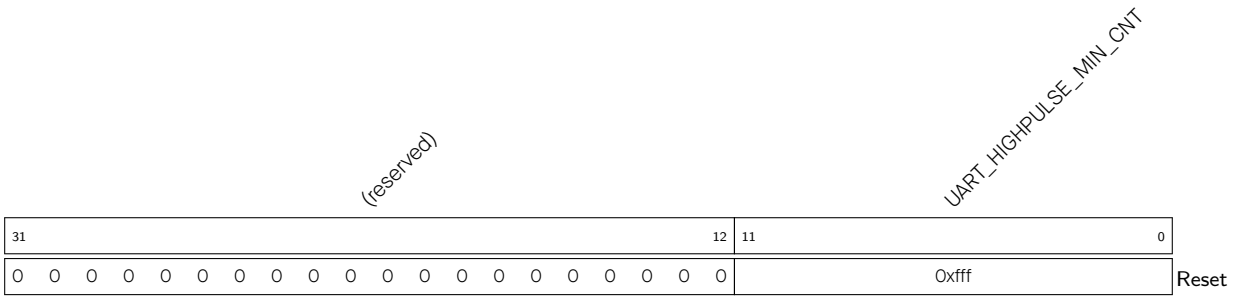
Register 26.32. UART_LOWPULSE_REG (0x007C)




**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)

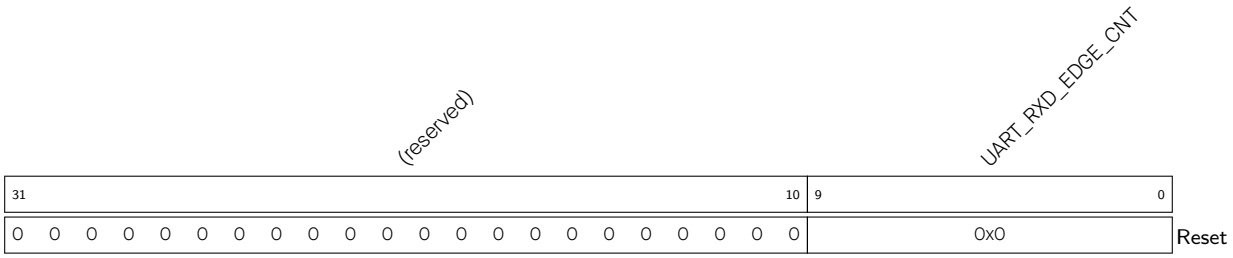
Register 26.33. UART_HIGHPULSE_REG (0x0080)




**UART_HIGHPULSE_MIN_CNT** Represents the maximum duration time for a high-level pulse. It is used for baud rate detection.

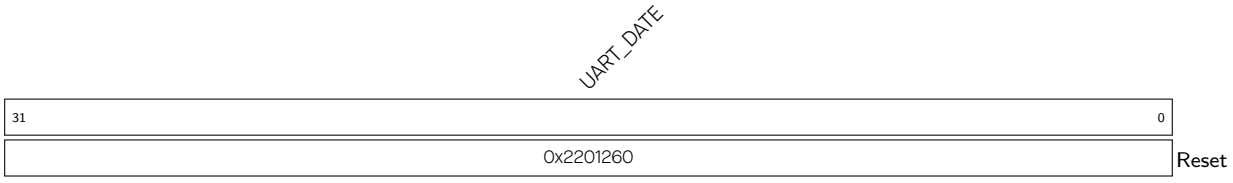
Measurement unit: APB_CLK clock cycle. (RO)

Register 26.34. UART_RXD_CNT_REG (0x0084)



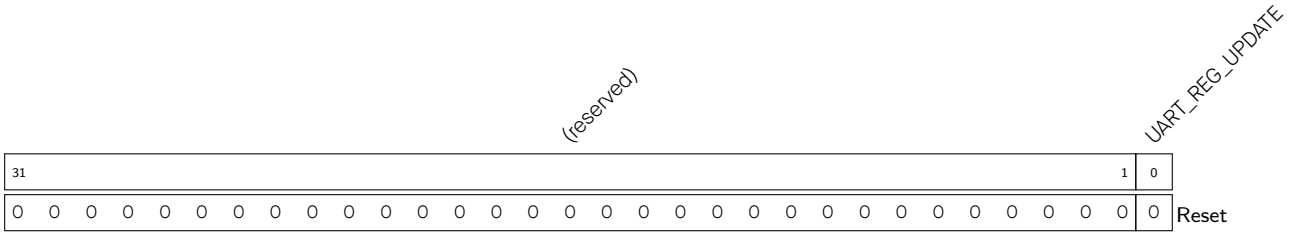

**UART_RXD_EDGE_CNT** Represents the number of RXD edge changes. It is used for baud rate detection. (RO)

Register 26.35. UART_DATE_REG (0x008C)




**UART_DATE** Version control register. (R/W)

Register 26.36. UART_REG_UPDATE_REG (0x0098)



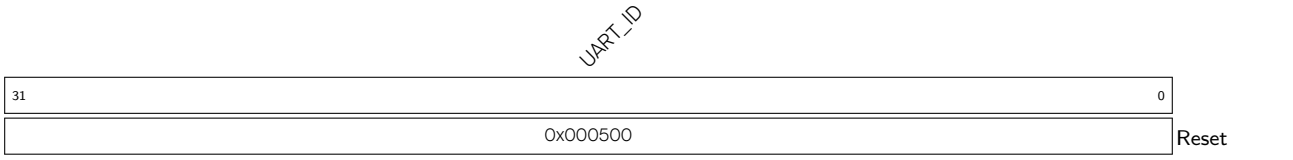

**UART_REG_UPDATE** Configures whether or not to synchronize registers.

0: Not synchronize

1: Synchronize

(R/W/SC)

Register 26.37. UART_ID_REG (0x009C)




**UART_ID** Configures the UART ID. (R/W)

# Chapter 27

# SPI Controller (SPI)

## 27.1 Overview

The Serial Peripheral Interface (SPI) is a synchronous serial interface useful for communication with external peripherals. The ESP32-C61 chip integrates three SPI controllers:

- SPI0
- SPI1
- General Purpose SPI2 (GP-SPI2)

SPI0 and SPI1 controllers (MSPI) are primarily reserved for internal use to communicate with external flash and PSRAM memory. This chapter mainly focuses on the GP-SPI2 controller.

## 27.2 Glossary

To better illustrate the functions of GP-SPI2, the following terms are used in this chapter.

| | |
|---|---|
| **Master Mode** | GP-SPI2 acts as an SPI master and initiates SPI transactions. |
| **Slave Mode** | GP-SPI2 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 buffer SPI_W0_REG ~ SPI_W15_REG and SPI peripheral. |
| **DMA-Controlled Transfer** | A data transfer that happens between DMA and SPI peripheral, controlled by the DMA engine. |
| **Configurable Segmented Transfer** | A data transfer controlled by DMA in SPI master mode. Such transfer consists of multiple transactions (segments), and each transaction can be configured independently. |
| **Slave Segmented Transfer** | A data transfer controlled by DMA in SPI slave mode. 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, and the other side receives data. Sending data and receiving data can not happen simultaneously on one side. |
| **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. |
| **FSPI** | Fast SPI. The prefix of the signals for GP-SPI2. FSPI bus signals are routed to GPIO pins via either GPIO matrix or IO MUX. |

## 27.3 Features

GP-SPI2 has the following features:

- Works as master or as slave
- Half- and full-duplex communications
- CPU- and DMA-controlled transfers
- Various data modes
  - 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 40 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: CS0~CS5
- Able to communicate with SPI devices, such as a sensor, a screen controller, as well as a flash or RAM chip

## 27.4 Architectural Overview

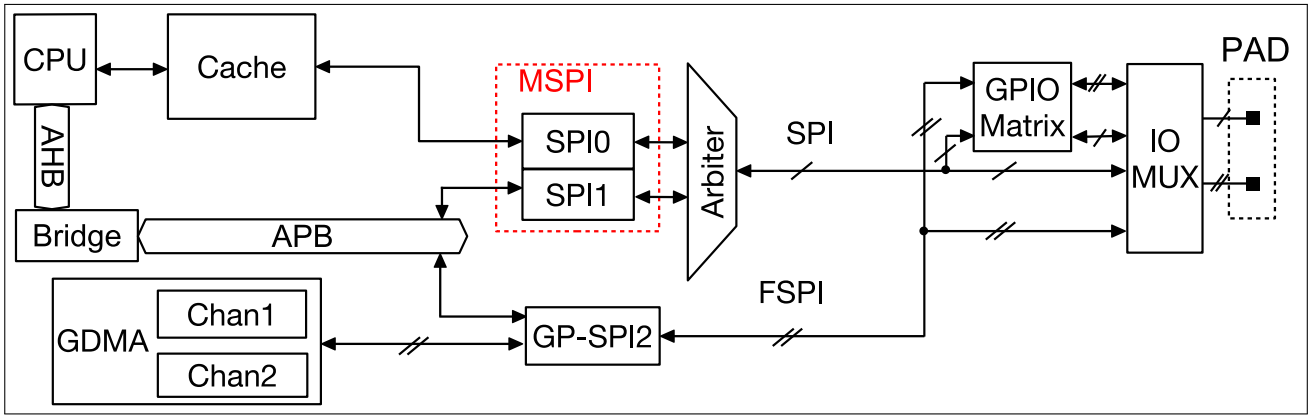



Figure 27.4-1. SPI Module Overview

Figure 27.4-1 shows an overview of this SPI module. GP-SPI2 exchanges data with SPI devices in the following ways:

- CPU-controlled transfer: CPU <-> GP-SPI2 <-> SPI devices
- DMA-controlled transfer: GDMA <-> GP-SPI2 <-> SPI devices

The signals for GP-SPI2 are prefixed with "FSPI" (Fast SPI). FSPI bus signals are routed to GPIO pins via either the GPIO matrix or the IO MUX. For more information, see Chapter 7 *GPIO Matrix and IO MUX*.

# 27.5 Functional Description

## 27.5.1 Data Modes

GP-SPI2 can be configured as either a master or a slave to communicate with other SPI devices in the following data modes. See Table 27.5-1.

**Table 27.5-1. Data Modes Supported by GP-SPI2**

| Data Mode | | CMD State | Address State | Data State |
|---|---|---|---|---|
| 1-bit SPI | | 1-bit | 1-bit | 1-bit |
| Dual SPI | Dual Output Read | 1-bit | 1-bit | 2-bit |
| | Dual I/O Read | 1-bit | 2-bit | 2-bit |
| Quad SPI | Quad Output Read | 1-bit | 1-bit | 4-bit |
| | Quad I/O Read | 1-bit | 4-bit | 4-bit |
| QPI | | 4-bit | 4-bit | 4-bit |

For more information about the states used when GP-SPI2 works as a master or a slave, see Section 27.5.9 and Section 27.5.10, respectively.

## 27.5.2 FSPI Bus Signals

Table 27.5-2 describes the functions of FSPI bus signals. Table 27.5-3 lists the signals used in various SPI modes.

**Table 27.5-2. Functional Description of FSPI Bus Signals**

| FSPI Bus Signal | Function |
|---|---|
| FSPID | MOSI/SIO0 (serial data input and output, bit0) |
| FSPIQ | MISO/SIO1 (serial data input and output, bit1) |
| FSPIWP | SIO2 (serial data input and output, bit2) |
| FSPIHD | SIO3 (serial data input and output, bit3) |
| FSPICLK | Input and output clock as master/slave |
| FSPICS0 | Input and output CS signal as master/slave |
| FSPICS1~5 | Output CS signal as master |

Table 27.5-3. FSPI Bus Signals Used in Various SPI Modes

| FSPI Bus Signal | Master | | | | | | Slave | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 1-bit SPI | | | 2-bit Dual SPI | 4-bit Quad SPI | QPI | 1-bit SPI | | | 2-bit Dual SPI | 4-bit Quad SPI | QPI |
| | FD[1] | 3-line HD[2] | 4-line HD | | | | FD | 3-line HD | 4-line HD | | | |
| FSPICLK | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y |
| FSPICS0 | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y |
| FSPICS1 | Y | Y | Y | Y | Y | Y | | | | | | |
| FSPICS2 | Y | Y | Y | Y | Y | Y | | | | | | |
| FSPICS3 | Y | Y | Y | Y | Y | Y | | | | | | |
| FSPICS4 | Y | Y | Y | Y | Y | Y | | | | | | |
| FSPICS5 | Y | Y | Y | Y | Y | Y | | | | | | |
| FSPID | Y | Y | (Y)[3] | Y[4] | Y[5] | Y | Y | Y | (Y)[6] | Y[7] | Y[8] | Y |
| FSPIQ | Y | | (Y)[3] | Y[4] | Y[5] | Y | Y | | (Y)[6] | Y[7] | Y[8] | Y |
| FSPIWP | | | | | Y[5] | Y | | | | | Y[8] | Y |
| FSPIHD | | | | | 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.

### 27.5.3 Bit Read/Write Order Control

- 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.

Table 27.5-4 shows the functions of SPI_RD/WR_BIT_ORDER.

GoBack

Table 27.5-4. Bit Order Control in GP-SPI2

| Bit Mode | FSPI 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 | FSPID or FSPIQ | 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 | FSPIQ | B7→B5→B3→B1 | B6→B4→B2→B0 | B1→B3→B5→B7 | B0→B2→B4→B6 |
| | FSPID | B6→B4→B2→B0 | B7→B5→B3→B1 | B0→B2→B4→B6 | B1→B3→B5→B7 |
| 4-bit mode | FSPIHD | B7→B3 | B4→B0 | B3→B7 | B0→B4 |
| | FSPIWP | B6→B2 | B5→B1 | B2→B6 | B1→B5 |
| | FSPIQ | B5→B1 | B6→B2 | B1→B5 | B2→B6 |
| | FSPID | B4→B0 | B7→B3 | B0→B4 | B3→B7 |

### 27.5.4 Unaligned Byte Transfer

- When operating as a master, GP-SPI2 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-SPI2 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-SPI2 sends and receives the data parts less than 1 byte following the same bit order as the other data bits as configured.

### 27.5.5 Transfer Types

The transfer types supported by GP-SPI2 working as a master or a slave are shown in Table 27.5-5.

**Table 27.5-5. Supported Transfer Types as Master or Slave**

| Mode | | CPU-Controlled Single Transfer | DMA-Controlled Single Transfer | DMA-Controlled Configurable Segmented Transfer | 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 |

The following sections provide detailed information about the transfer types listed in the table above.

### 27.5.6 CPU-Controlled Data Transfer

GP-SPI2 provides 16 x 32-bit data buffers: SPI_W0_REG~SPI_W15_REG. Figure 27.5-1 shows the buffer's structure. CPU-controlled transfer indicates the transfer in which the data to send is from GP-SPI2 data buffer and the received data is stored to GP-SPI2 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.

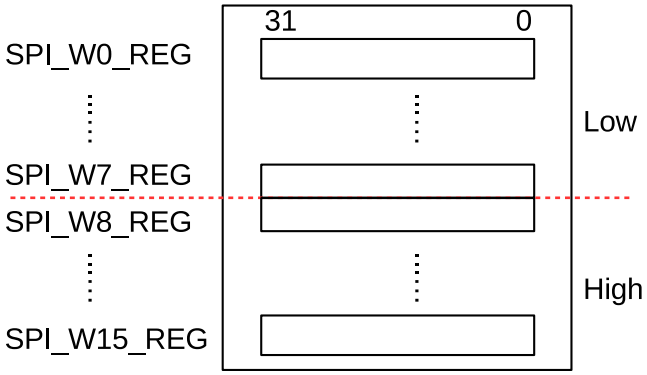



Figure 27.5-1. Data Buffer Used in CPU-Controlled Transfer

#### 27.5.6.1 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.

#### 27.5.6.2 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

### 27.5.7 DMA-Controlled Data Transfer

DMA-controlled transfer refers to the transfer in which the GDMA RX module receives data and the GDMA TX module sends data. This transfer is supported when the GP-SPI2 works as master or as slave.

A DMA-controlled transfer can be:

- a single transfer, consisting of only one transaction. GP-SPI2 supports this transfer when it works as master or as slave.
- a configurable segmented transfer, consisting of several transactions (segments). GP-SPI2 supports this transfer only when works as a master. For more information, see Section 27.5.9.5.
- a slave segmented transfer, consisting of several transactions (segments). GP-SPI2 supports this transfer only when works as a slave. For more information, see Section 27.5.10.3.

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 27.5.9 and Section 27.5.10. Meanwhile, the GDMA RX module is independent from the GDMA 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.

#### 27.5.7.1 DMA Configuration

- Select a GDMA channel *n* and configure GDMA TX/RX descriptor. See Chapter 4 *GDMA Controller (GDMA)*.
- Set GDMA_INLINK_START_CH*n* or GDMA_OUTLINK_START_CH*n* to start DMA RX or TX engine, respectively.
- Before all the GDMA TX buffer is used or the GDMA TX engine is reset, if GDMA_OUTLINK_RESTART_CH*n* is set, a new TX buffer will be added to the end of the last TX buffer in use.
- GDMA RX buffer is linked in the same way as the GDMA TX buffer, by setting GDMA_INLINK_START_CH*n* or GDMA_INLINK_RESTART_CH*n*.
- The TX and RX data lengths are determined by the configured GDMA TX and RX buffer respectively, both of which are 0∼32 KB.
- Initialize GDMA inlink and outlink before GDMA 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-SPI2 operates as master, if GDMA_IN_SUC_EOF_CH*n*_INT_ENA is set, then the interrupt GDMA_IN_SUC_EOF_CH*n*_INT will be triggered when one single transfer or one configurable segmented transfer is finished.

When GP-SPI2 operates as slave, if GDMA_IN_SUC_EOF_CH*n*_INT_ENA is set, then the interrupt GDMA_IN_SUC_EOF_CH*n*_INT will be triggered when one of the following conditions are met.

**Table 27.5-6. Interrupt Trigger Condition on GP-SPI2 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

#### 27.5.7.2 GDMA 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 GDMA 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 GDMA_OUT_EOF_CH*n*_INT are triggered.
- If the length of the configured GDMA TX buffer is longer than that of the 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 GDMA.
- If the length of the configured GDMA 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 GDMA_IN_SUC_EOF_CH*n*_INT is not triggered.
- If the length of the configured GDMA 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.

### 27.5.8 Data Flow Control

CPU-controlled and DMA-controlled transfers are supported in GP-SPI2 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 GDMA 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.

#### 27.5.8.1 GP-SPI2 Functional Blocks

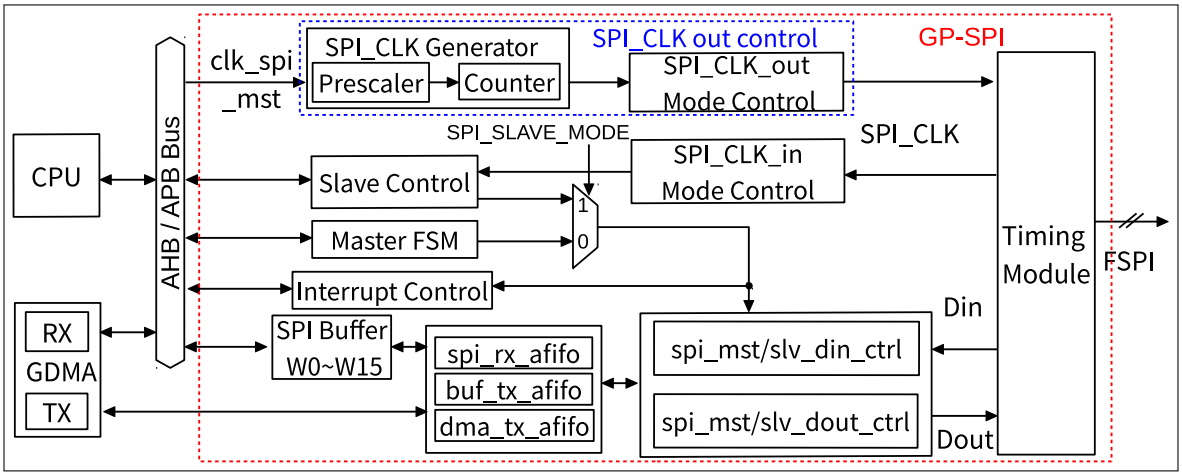



Figure 27.5-2. GP-SPI2 Functional Blocks

Figure 27.5-2 shows the main functional blocks in GP-SPI2, including:

- Master FSM: all the features supported in GP-SPI2 as master are controlled by this state machine together with register configuration.
- SPI Buffer: SPI_W0_REG~SPI_W15_REG. See Figure 27.5-1. The data in CPU-controlled transfer is prepared in this buffer.
- Timing Module: captures data on FSPI 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 GDMA.
- clk_spi_mst: this clock is divided from PLL_CLK and works as the module clock of GP-SPI2, to generate SPI_CLK signal for data transfer and for slaves, when GP-SPI2 works as master.
- 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-SPI2 works as slave.

#### 27.5.8.2 Data Flow Control When GP-SPI2 Works as Master

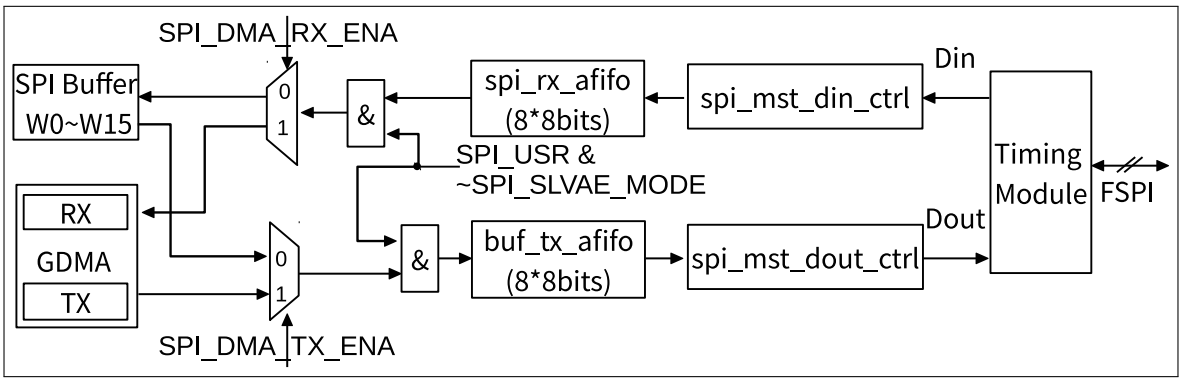



Figure 27.5-3. Data Flow Control When GP-SPI2 Works as Master

Figure 27.5-3 shows the data flow when GP-SPI2 works as master. Its control logic is as follows:

- RX data: data bits on FSPI bus are 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 GDMA 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 GDMA TX buffer.

The data in *buf_tx_afifo* is sent out to Timing Module in 1/2/4-bit modes, controlled by GP-SPI2 state machine. The Timing Module can be used for timing compensation. For more information, see Section 27.8.

#### 27.5.8.3 Data Flow Control When GP-SPI2 Works as Slave

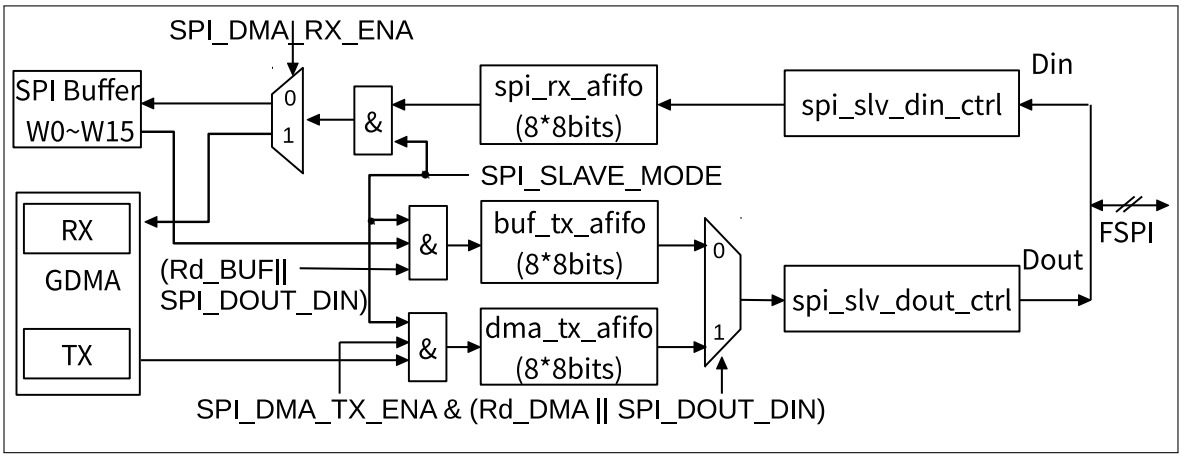



Figure 27.5-4. Data Flow Control When GP-SPI2 Works as Slave

Figure 27.5-4 shows the data flow when GP-SPI2 works 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 FSPI 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 GDMA 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 GDMA 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-bit modes.

### 27.5.9 GP-SPI2 Works as Master

GP-SPI2 can be configured as an SPI master by clearing the bit SPI_SLAVE_MODE in SPI_SLAVE_REG. In this operation mode, GP-SPI2 provides clock signal (the divided clock from GP-SPI module clock) and CS signal (CS0~CS5).

#### 27.5.9.1 State Machine

When GP-SPI2 works as master, the state machine controls GP-SPI2'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-SPI2 is mainly used to access 1/2/4-bit SPI devices, such as flash and external RAM, thus the naming of GP-SPI2 states keeps consistent with the sequence naming of flash and external RAM. The meaning of each state is described as follows and Figure 27.5-5 shows the workflow of GP-SPI2 state machine.

1. **IDLE:** GP-SPI2 is not active or is operating as a slave.

2. **CONF:** 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*.

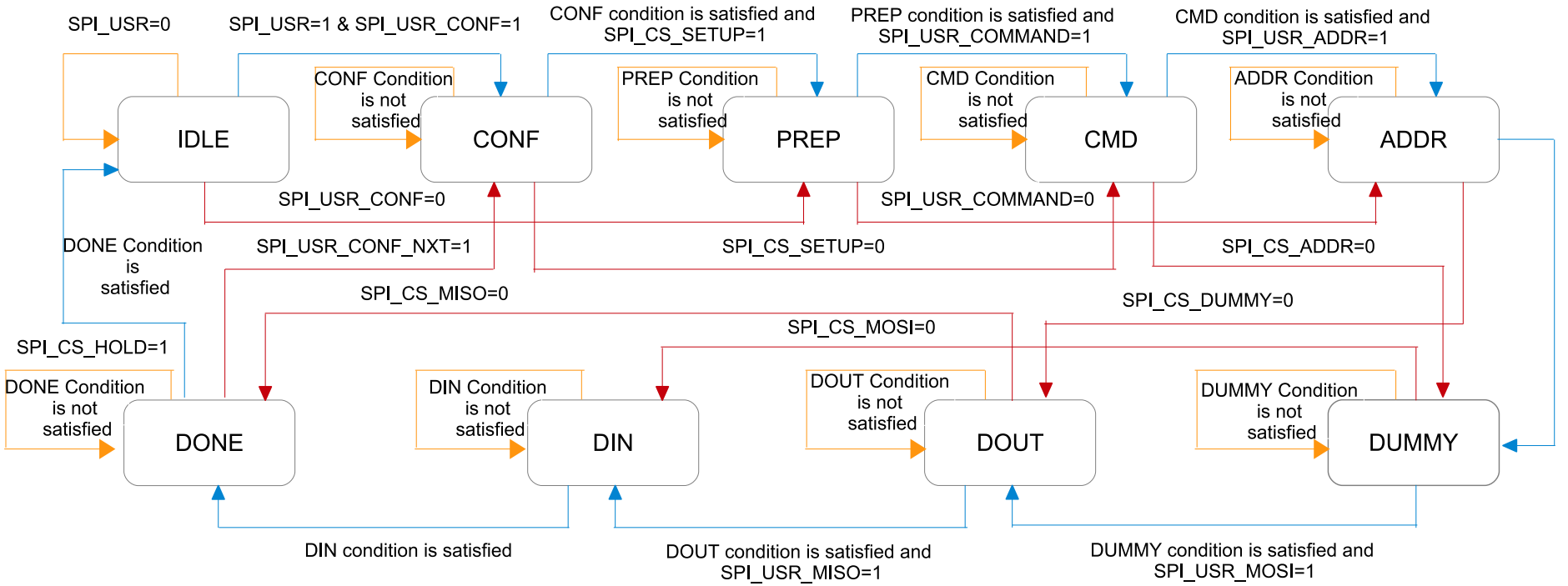



Figure 27.5-5. GP-SPI2 State Machine

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 whether the registers of the following states are set or not.

Explanation of 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.

#### 27.5.9.2 Register Configuration for State and Bit Mode Control

**Introduction**

The registers, related to GP-SPI2 state control, are listed in Table 27.5-7. Users can enable QPI mode for GP-SPI2 by setting the bit SPI_QPI_MODE in register SPI_USER_REG.

Table 27.5-7. Registers Used for State Control in 1/2/4-bit Modes

| State | Control Registers for 1-bit Mode FSPI Bus | Control Registers for 2-bit Mode FSPI Bus | Control Registers for 4-bit Mode FSPI Bus |
|---|---|---|---|
| CMD | SPI_USR_COMMAND_VALUE<br>SPI_USR_COMMAND_BITLEN<br>SPI_USR_COMMAND | SPI_USR_COMMAND_VALUE<br>SPI_USR_COMMAND_BITLEN<br>SPI_FCMD_DUAL<br>SPI_USR_COMMAND | SPI_USR_COMMAND_VALUE<br>SPI_USR_COMMAND_BITLEN<br>SPI_FCMD_QUAD<br>SPI_USR_COMMAND |
| ADDR | SPI_USR_ADDR_VALUE<br>SPI_USR_ADDR_BITLEN<br>SPI_USR_ADDR | SPI_USR_ADDR_VALUE<br>SPI_USR_ADDR_BITLEN<br>SPI_USR_ADDR<br>SPI_FADDR_DUAL | SPI_USR_ADDR_VALUE<br>SPI_USR_ADDR_BITLEN<br>SPI_USR_ADDR<br>SPI_FADDR_QUAD |
| DUMMY | SPI_USR_DUMMY_CYCLELEN<br>SPI_USR_DUMMY | SPI_USR_DUMMY_CYCLELEN<br>SPI_USR_DUMMY | SPI_USR_DUMMY_CYCLELEN<br>SPI_USR_DUMMY |
| DIN | SPI_USR_MISO<br>SPI_MS_DATA_BITLEN | SPI_USR_MISO<br>SPI_MS_DATA_BITLEN<br>SPI_FREAD_DUAL | SPI_USR_MISO<br>SPI_MS_DATA_BITLEN<br>SPI_FREAD_QUAD |
| DOUT | SPI_USR_MOSI<br>SPI_MS_DATA_BITLEN | SPI_USR_MOSI<br>SPI_MS_DATA_BITLEN<br>SPI_FWRITE_DUAL | SPI_USR_MOSI<br>SPI_MS_DATA_BITLEN<br>SPI_FWRITE_QUAD |

As shown in Table 27.5-7, the registers in each cell should be configured to set the FSPI 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 AND SPI_FCMD_QUAD.
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.
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.
   - Configure GDMA 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-SPI2 works as master described in this and subsequent sections requires setting 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-SPI2 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 GDMA TX buffer for DMA-controlled transfer.

Pay special attention to the command value in SPI_USR_COMMAND_VALUE and the address value in SPI_USR_ADDR_VALUE.

The configuration of command value is as follows:

**Table 27.5-8. 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: For the detailed configuration, see Table 27.5-4.

The configuration of address value is as follows:

**Table 27.5-9. 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: For the detailed configuration, see Table 27.5-4.

#### 27.5.9.3 Full-Duplex Communication (1-bit Mode Only)

**Introduction**

GP-SPI2 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 (FSPID, sending) and MISO (FSPIQ, receiving) at the same time. To enable this communication mode, set the bit SPI_DOUTDIN in register SPI_USER_REG. Figure 27.5-6 illustrates the connection of GP-SPI2 with its slave in full-duplex communication.

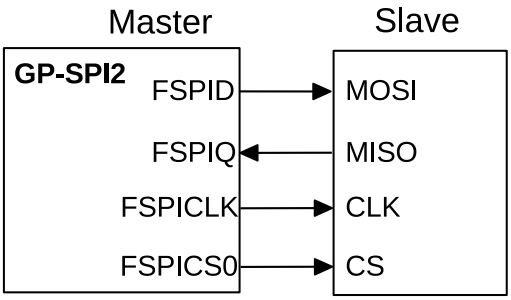



**Figure 27.5-6. Full-Duplex Communication Between GP-SPI2 Master and a Slave**

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**

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_CLK and APB_CLK (See Chapter 8 *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 27.5-7.
- Configure SPI CS setup time and hold time according to Section 27.6.
- Set the polarity of FSPICLK according to Section 27.7.

- 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 GDMA TX/RX link,
    * and start GDMA TX/RX engine, as described in Section 27.5.7 and Section 27.5.8.
- 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.

#### 27.5.9.4 Half-Duplex Communication (1/2/4-bit Mode)

**Introduction**

In this mode, GP-SPI2 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 27.5.9.2, the properties of GP-SPI2 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 27.5-7.

The detailed properties of half-duplex GP-SPI2 are as follows:

1. CMD: 0~16 bits, MOSI (master output, slave input).
2. ADDR: 0~32 bits, MOSI (master output, slave input).
3. DUMMY: 0~256 FSPICLK cycles, MOSI (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, MOSI (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, MISO (slave output, master input).

**Configuration**

The register configuration can be as follows:

1. Configure the IO path via IO MUX or GPIO matrix between GP-SPI2 and an external SPI device.
2. Configure AHB_CLK, APB_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 27.5-7.
5. Configure SPI CS setup time and hold time according to Section 27.6.

6. Set the polarity of FSPICLK according to Section 27.7.
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 GDMA TX/RX link,
     - and start GDMA TX/RX engine, as described in Section 27.5.7 and Section 27.5.8.
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**

The following example shows how GP-SPI2 accesses flash and external RAM in master half-duplex communication.

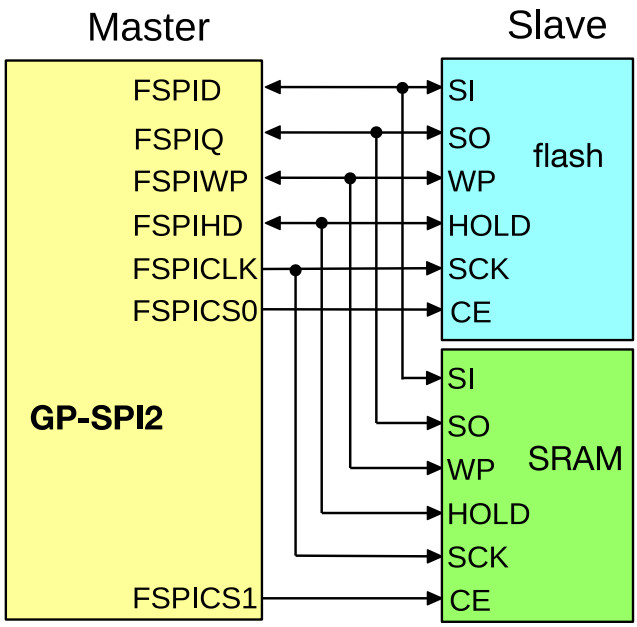



**Figure 27.5-7. Connection of GP-SPI2 to Flash and External RAM in 4-bit Mode**

Figure 27.5-8 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.

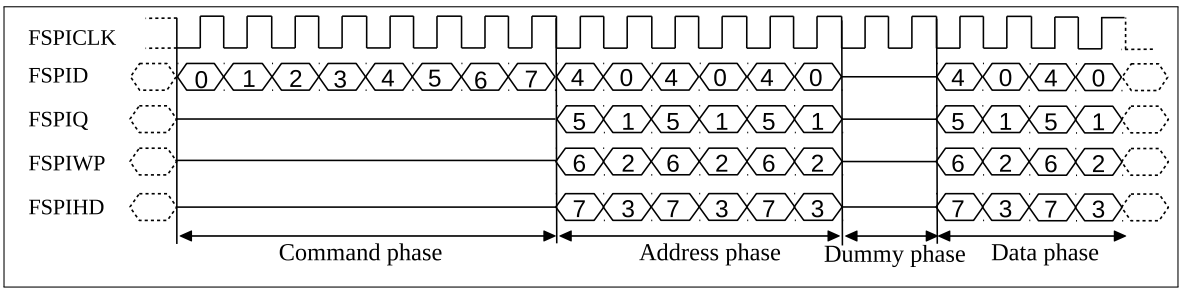



Figure 27.5-8. SPI Quad I/O Read Command Sequence Sent by GP-SPI2 to Flash

### 27.5.9.5 DMA-Controlled Configurable Segmented Transfer

**Note:**

Users can simply skip the CONF state of a configurable segmented transfer to implement a single transfer, so there is no separate section on how to configure a single transfer as master.

#### 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 27.5-9 shows how this feature works.

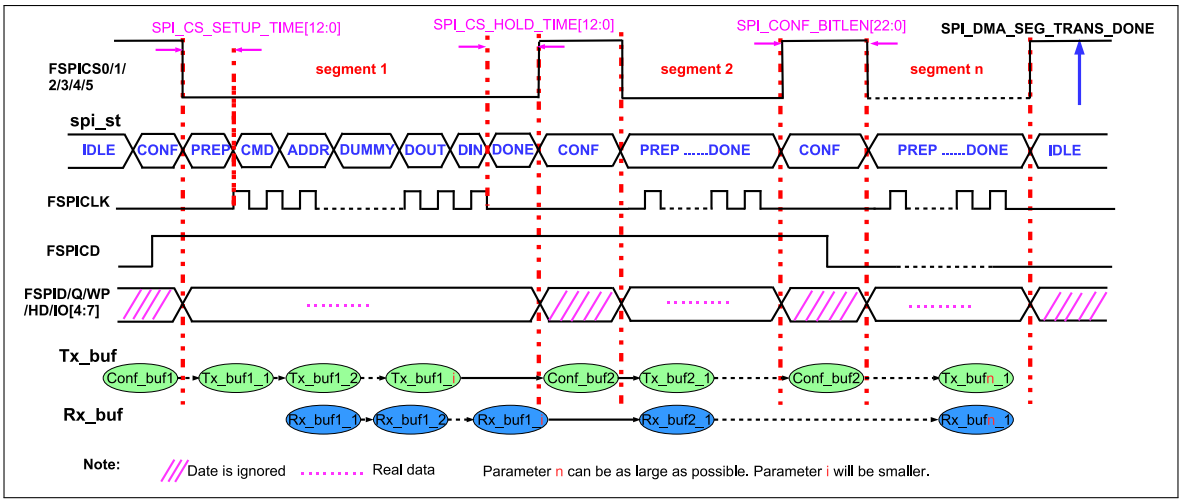



Figure 27.5-9. Configurable Segmented Transfer

As shown in Figure 27.5-9, 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 GDMA CONF links and CONF buffers (Conf_buf*i* in Figure 27.5-9) for each CONF state. A GDMA TX link is used to connect all the CONF buffers and TX data buffers (Tx_buf*i* in

Figure 27.5-9) into a chain. Hence, the behavior of the FSPI 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 FSPI bus, and the behavior of the GDMA, 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 IO MUX or GPIO matrix between GP-SPI2 and an external SPI device.
2. Configure AHB_CLK, APB_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 27.5-7.
5. Configure SPI CS setup time and hold time according to Section 27.6.
6. Set the polarity of FSPICLK according to Section 27.7.
7. Prepare descriptors for GDMA 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 GDMA.
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 GDMA.
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 27.9.
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 GDMA 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 27.5-10 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 27.5-10. 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 27.5-11 and Table 27.5-12 provide an example 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 27.5-11. 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 27.5-12, 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 27.5-12. 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 27.6 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.

## 27.5.10 GP-SPI2 Works as Slave

GP-SPI2 can be used as a slave to communicate with an SPI master. As a slave, GP-SPI2 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 and rising edges indicate the start and 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.

#### 27.5.10.1 Configurable Communication Formats

When GP-SPI2 works as slave, 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-SPI2.
- "DOUT" means that an SPI master writes data to GP-SPI2.

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 27.5-13 and Table 27.5-14 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 FSPI 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-SPI2 as slave. SPI_SLV_CMD_ERR_INT_RAW can only be cleared by software.

#### 27.5.10.2 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-SPI2 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-SPI2. 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-SPI2 receives data. The data is stored in GP-SPI2 GDMA RX buffer.
- 0x4 (Rd_DMA): DMA-controlled read operation. Master receives the data sent by GP-SPI2. The data comes from GP-SPI2 GDMA TX buffer.
- 0x7 (CMD7): used to generate an SPI_SLV_CMD7_INT interrupt. It can also generate a GDMA_IN_SUC_EOF_CH*n*_INT interrupt in a slave segmented transfer when GDMA RX link is used. But it will not end GP-SPI2's slave segmented transfer.
- 0x8 (CMD8): only used to generate an SPI_SLV_CMD8_INT interrupt, which will not end GP-SPI2's slave segmented transfer.
- 0x9 (CMD9): only used to generate an SPI_SLV_CMD9_INT interrupt, which will not end GP-SPI2's slave segmented transfer.
- 0xA (CMDA): only used to generate an SPI_SLV_CMDA_INT interrupt, which will not end GP-SPI2'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 are all 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-SPI2 enters QPI mode when receiving this command and the bit SPI_QPI_MODE in register SPI_USER_REG is set.
- 0xDD (Ex_QPI): GP-SPI2 exits QPI mode when receiving this command and the bit SPI_QPI_MODE is cleared.

All the CMD values supported by GP-SPI2 are listed in Table 27.5-13 and Table 27.5-14. Note that the DUMMY state is always in 1-bit mode and lasts for eight SPI_CLK cycles.

**Table 27.5-13. 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 |
| | 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 | - |

Table 27.5-13. CMD Values Supported in SPI Mode

| Transfer Type | CMD[7:0] | CMD State | ADDR State | DATA State |
|---|---|---|---|---|
| | 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 27.5-14. 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-SPI2 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-SPI2 slave will be back to SPI mode.

Other transfer types than these described in Table 27.5-13 and Table 27.5-14 are ignored. 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 27.9.

### 27.5.10.3 Slave Single Transfer and Slave Segmented Transfer

When GP-SPI2 works as 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 27.5-13 and Table 27.5-14 are supported in a single transaction (segment). It means that CPU-controlled transactions and DMA-controlled transactions 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.

#### 27.5.10.4 Configuration of Slave Single Transfer

When operating as slave, GP-SPI2 supports CPU/DMA-controlled full-duplex/half-duplex single transfers.

The register configuration procedure is as follows:

1. Configure the IO path via IO MUX or GPIO matrix between GP-SPI2 and an external SPI device.
2. Configure AHB_CLK and APB_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 GDMA TX/RX link,
     - and start GDMA TX/RX engine, as described in Section 27.5.7 and Section 27.5.8.
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 GDMA_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 27.9 are optional.

#### 27.5.10.5 Configuration of Slave Segmented Transfer in Half-Duplex

GDMA must be used in this mode. The register configuration procedure is as follows:

1. Configure the IO path via IO MUX or GPIO matrix between GP-SPI2 and an external SPI device.
2. Configure AHB_CLK and 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 GDMA TX/RX link and start GDMA TX/RX engine, as shown in Section 27.5.7 and Section 27.5.8.
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 27.9 are optional.

When End_SEG_TRANS (0x05 in SPI mode, 0xA5 in QPI mode) is received by GP-SPI2, this slave segmented transfer is ended and the interrupt SPI_DMA_SEG_TRANS_DONE_INT is triggered.

#### 27.5.10.6 Configuration of Slave Segmented Transfer in Full-Duplex

GDMA must be used in this mode. In such transfer, the data is transferred from and to the GDMA buffer. The interrupt GDMA_IN_SUC_EOF_CH*n*_INT is triggered when the transfer ends.

The register configuration procedure is as follows:

1. Configure the IO path via IO MUX or GPIO matrix between GP-SPI2 and an external SPI device.
2. Configure AHB_CLK and 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 and SPI_DMA_RX_ENA. Configure GDMA TX/RX link and start GDMA TX/RX engine, as shown in Section 27.5.7 and Section 27.5.8.
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 GDMA data in unit of byte.
7. Set SPI_DMA_SLV_SEG_TRANS_EN in register SPI_DMA_CONF_REG to enable slave segmented transfer.
8. Set GDMA_IN_SUC_EOF_CH*n*_INT_ENA and wait for the interrupt GDMA_IN_SUC_EOF_CH*n*_INT.

## 27.6 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 2 and mode 4.

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.

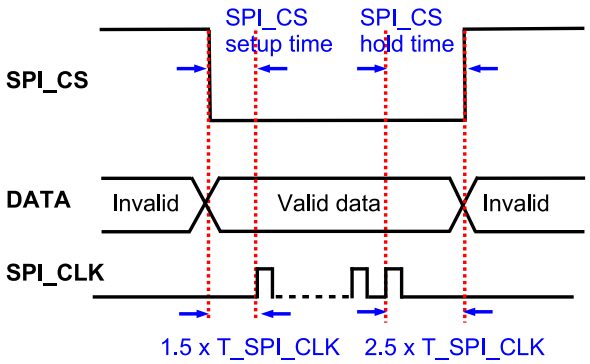
When operating as master, set the CS setup time by specifying SPI_CS_SETUP in register SPI_USER_REG and SPI_CS_SETUP_TIME in register SPI_USER1_REG.

- 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.

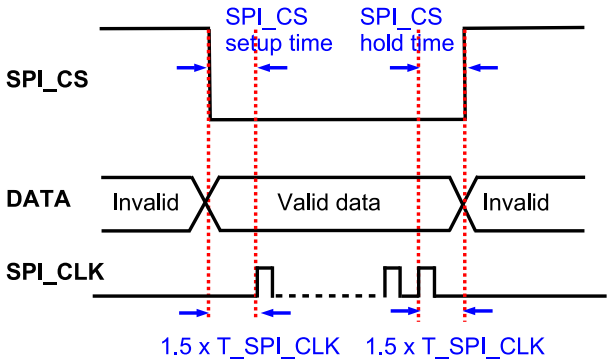
Figure 27.6-1 and Figure 27.6-2 show the recommended CS timing and register configuration to access external RAM and flash.




Register Configurations:

SPI_CS_SETUP = 1; SPI_CS_SETUP_TIME = 0;
SPI_CS_HOLD = 1; SPI_CS_HOLD_TIME = 1.

Figure 27.6-1. Recommended CS Timing and Settings When Accessing External RAM




Register Configurations:

SPI_CS_SETUP = 1; SPI_CS_SETUP_TIME = 0;
SPI_CS_HOLD = 1; SPI_CS_HOLD_TIME = 0.

Figure 27.6-2. Recommended CS Timing and Settings When Accessing Flash

## 27.7 GP-SPI2 Clock Control

GP-SPI2 has the following clocks:

- clk_spi_mst: module clock of GP-SPI2. When GP-SPI2 works as master, this clk_spi_mst is used to generate SPI_CLK signal for data transfer and for slaves.
- clk_hclk: module timing compensation clock of GP-SPI2, which is a frequency-doubled clock derived from the same source as clk_spi_mst.
- SPI_CLK: the output clock when the GP-SPI2 works as master.

- AHB_CLK: clock for register configuration.

clk_spi_mst is enabled by PCR_SPI2_CLK_EN and its clock source is controlled by PCR_SPI2_CLKMST_SEL:

- 0: XTAL_CLK
- 1: PLL_F160M_CLK
- 2: FOSC_CLK

When operating as master, the maximum output clock frequency of GP-SPI2 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-SPI2 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-SPI2 is $f_{SPI\_CLK}$ <= $f_{AHB\_CLK}$.

### 27.7.1 Clock Phase and Polarity

SPI protocol has four clock modes, i.e., modes 0∼3. See Figure 27.7-1 and Figure 27.7-2.

**Note:**
The images are sourced from the SPI protocol. In the two images, the black **SAMPLE** signal represents the standard sampling edge, while the red represents the default sampling edge of GP-SPI2.

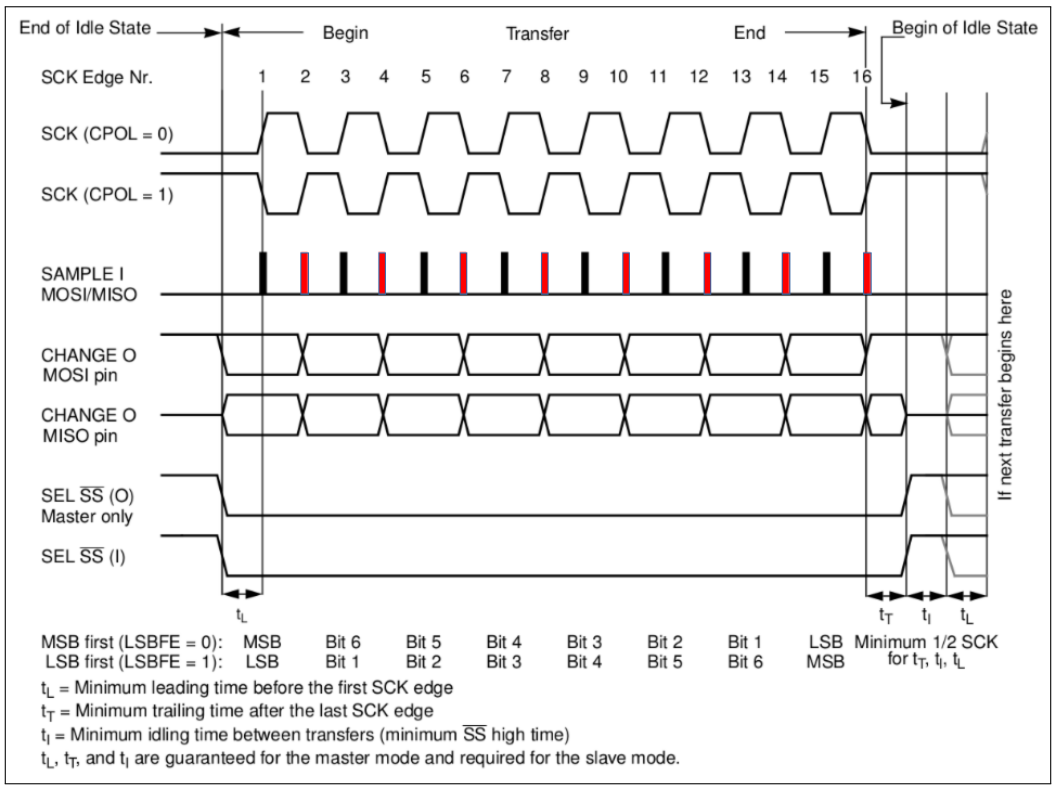



Figure 27.7-1. SPI Clock Mode 0 or 2

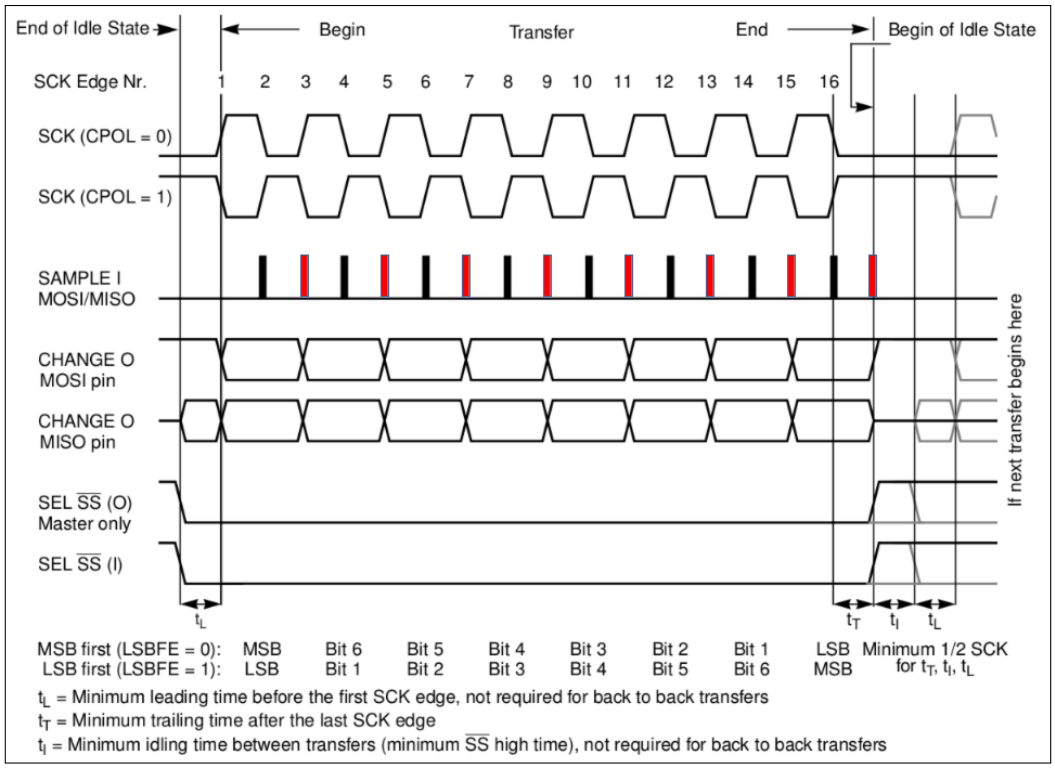



Figure 27.7-2. SPI Clock Mode 1 or 3

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.

### 27.7.2 Clock Control When GP-SPI2 Works as Master

The four clock modes 0~3 are supported in GP-SPI2 when it works as master. The polarity and phase of GP-SPI2 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 27.7-1, and can be changed according to the path delay in the application.

**Table 27.7-1. 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.

### 27.7.3 Clock Control When GP-SPI2 Works as Slave

GP-SPI2 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 27.7-2:

**Table 27.7-2. 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 |

## 27.8 GP-SPI2 Timing Compensation

**Introduction**

Based on the role of GP-SPI2 in a user-defined SPI transmission system (whether GP-SPI2 works as master or slave), GP-SPI2 provides the following timing compensation schemes:

- In master mode, the default compensation scheme delays SPI_CLK by half a cycle. Refer to Figure 27.7-1 and Figure 27.7-2. You can also switch to standard sampling via register configuration.
- If the series register compensation scheme is used, it further improves timing compensation for high-speed transmissions on top of the default half-cycle delay.
- When working as slave, GP-SPI2 follows the standard SPI protocol for data transmission by default. You can also advance data transmission by half a cycle via register configuration.

When GP-SPI2 works as master, it delays the sampling of data returned by the slave by half an SPI clock cycle. You can switch to the standard SPI protocol sampling scheme by setting SPI_CLK_EDGE_SEL high. Note that when SPI_CLK_EQU_SYSCLK is set to 1, the data sampling from slave is always delayed by half an SPI clock cycle, and at this point, you can use other methods to adjust the IO timing.

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 7 *GPIO Matrix and IO MUX*.

Figure 27.8-1 shows the timing compensation control for GP-SPI2 as master, including the following paths:

- "CLK": the output path of GP-SPI2 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-SPI2 (see line 3 path in color purple in Figure 27.8-1). The input data from an external SPI device passes through GPIO matrix or IO MUX, then is adjusted by the Timing Module (see Figure 27.5-2) and finally is stored into spi_rx_afifo.
- "OUT": data output path of GP-SPI2 (see line 2 path in color rose-red in Figure 27.8-1). 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.

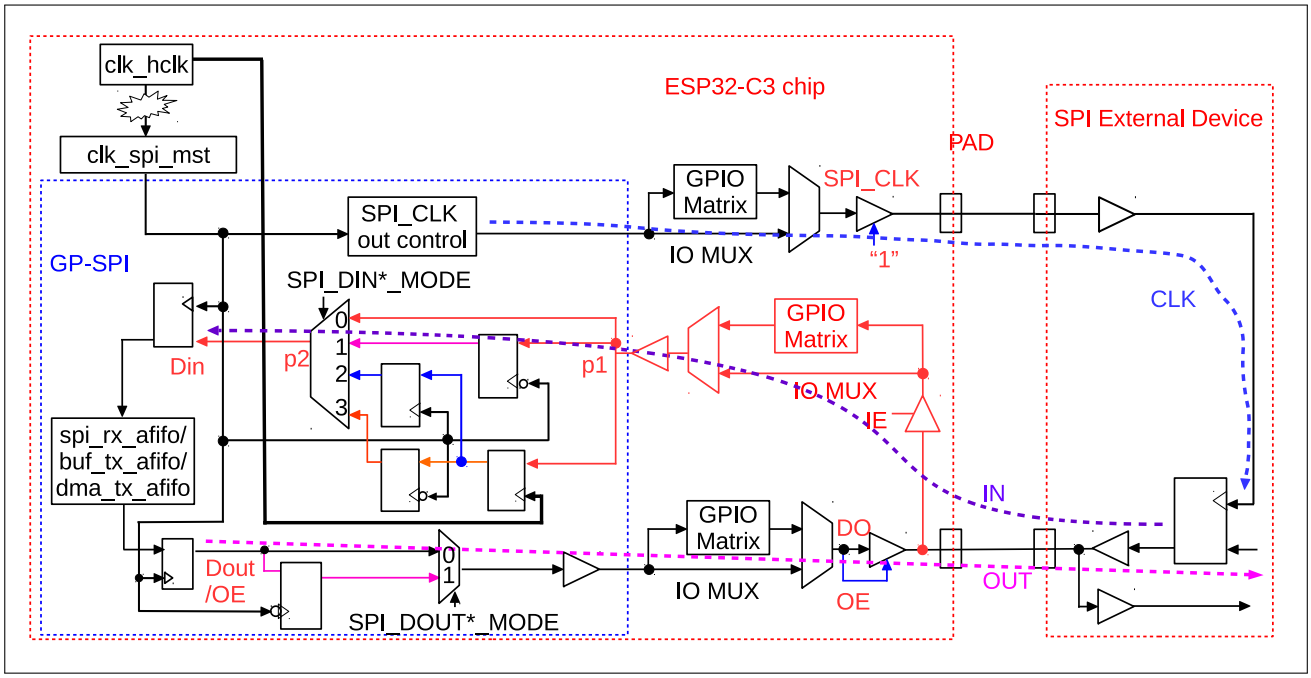



Figure 27.8-1. Timing Compensation Control Diagram in GP-SPI as Master

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 27.8-2 shows a timing compensation example when GP-SPI2 works 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.

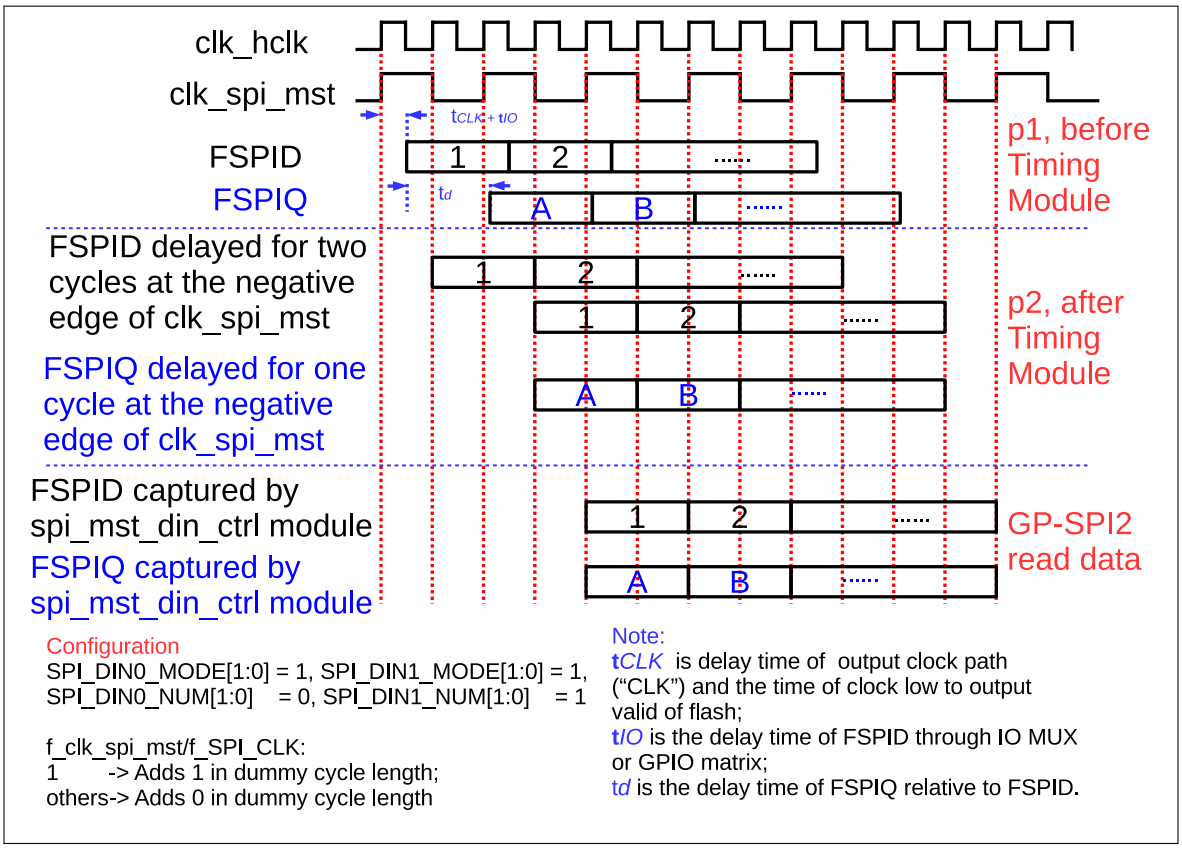



Figure 27.8-2. Timing Compensation Example in GP-SPI2

In Figure 27.8-2, "p1" is the point of input data of Timing Module, "p2" is the point of output data of Timing Module. Since the input data FSPIQ is unaligned to FSPID, 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 FSPID for two cycles at the falling edge of clk_spi_mst.
- Delay FSPIQ for one cycle at the falling edge of clk_spi_mst.
- Add one extra dummy cycle.

When GP-SPI2 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.

# 27.9 Interrupt

## 27.9.1 Interrupt Description

GP-SPI2 can generate the following external interrupt signal:

- SPI_INTR

SPI_INTR can be mapped to the CPU through the Interrupt Matrix. The interrupt signal is generated by internal interrupt sources. Table 27.9-1 lists the internal interrupt sources, interrupt trigger conditions, and generated interrupt signals.

**Table 27.9-1. Internal Interrupt Sources of GP-SPI2**

| Internal Interrupt Source | Trigger Condition | Interrupt Signal |
|---|---|---|
| SPI_DMA_INFIFO_FULL_ERR_INT | The length of GDMA RX FIFO is shorter than that of actual data transferred | SPI_INTR |
| SPI_DMA_OUTFIFO_EMPTY_ERR_INT | The length of GDMA TX FIFO is shorter than that of actual data transferred | SPI_INTR |
| SPI_SLV_EX_QPI_INT | Ex_QPI is received correctly when GP-SPI2 works as slave and the SPI transfer ends | SPI_INTR |
| SPI_SLV_EN_QPI_INT | En_QPI is received correctly when GP-SPI2 works as slave and the SPI transfer ends | SPI_INTR |
| SPI_SLV_CMD7_INT | CMD7 is received correctly when GP-SPI2 works as slave and the SPI transfer ends | SPI_INTR |
| SPI_SLV_CMD8_INT | CMD8 is received correctly when GP-SPI2 works as slave and the SPI transfer ends | SPI_INTR |
| SPI_SLV_CMD9_INT | CMD9 is received correctly when GP-SPI2 works as slave and the SPI transfer ends | SPI_INTR |
| SPI_SLV_CMDA_INT | CMDA is received correctly when GP-SPI2 works as slave and the SPI transfer ends | SPI_INTR |
| SPI_SLV_RD_DMA_DONE_INT | At the end of Rd_DMA transfer when GP-SPI2 works as slave | SPI_INTR |
| SPI_SLV_WR_DMA_DONE_INT | At the end of Wr_DMA transfer when GP-SPI2 works as slave | SPI_INTR |
| SPI_SLV_RD_BUF_DONE_INT | At the end of Rd_BUF transfer when GP-SPI2 works as slave | SPI_INTR |
| SPI_SLV_WR_BUF_DONE_INT | At the end of Wr_BUF transfer when GP-SPI2 works as slave | SPI_INTR |
| SPI_TRANS_DONE_INT | At the end of SPI bus transfer when GP-SPI2 works as master or as slave | SPI_INTR |

Cont'd on next page

Table 27.9-1 – cont'd from previous page

| Internal Interrupt Source | Trigger Condition | Interrupt Signal |
|---|---|---|
| SPI_DMA_SEG_TRANS_DONE_INT | At the end of End_SEG_TRANS transfer in GP-SPI2 slave segmented transfer mode or at the end of master configurable segmented transfer mode | SPI_INTR |
| SPI_SEG_MAGIC_ERR_INT | A Magic error occurs in CONF buffer during configurable segmented transfer when GP-SPI2 works as master | SPI_INTR |
| SPI_MST_RX_AFIFO_WFULL_ERR_INT | RX AFIFO write-full error when GP-SPI2 works as master | SPI_INTR |
| SPI_MST_TX_AFIFO_REMPTY_ERR_INT | TX AFIFO read-empty error when GP-SPI2 works as master | SPI_INTR |
| SPI_SLV_CMD_ERR_INT | A received command value is not supported when GP-SPI2 works as slave | SPI_INTR |
| SPI_APP2_INT | Set SPI_APP2_INT_SET (Only used for user defined function) | SPI_INTR |
| SPI_APP1_INT | Set SPI_APP1_INT_SET (Only used for user defined function) | SPI_INTR |

## 27.9.2 Interrupts Used in Master and in Slave

Table 27.9-2 and Table 27.9-3 show the interrupts used in GP-SPI2 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 27.9-2. GP-SPI2 Master Mode Interrupts

| Transfer Type | Communication Mode | Controlled by | Interrupt |
|---|---|---|---|
| Single Transfer | Full-duplex | DMA | GDMA_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 | GDMA_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 GDMA_IN_SUC_EOF_CH*n*_INT is triggered, it means all the RX data of GP-SPI2 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 27.9-3. GP-SPI2 Slave Mode Interrupts

| Transfer Type | Communication Mode | Controlled by | Interrupt |
|---|---|---|---|
| Single Transfer | Full-duplex | DMA | GDMA_IN_SUC_EOF_CH*n*_INT [1] |
| | | CPU | SPI_TRANS_DONE_INT [2] |
| | Half-duplex MOSI Mode | DMA (Wr_DMA) | GDMA_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 | GDMA_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 GDMA_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, GDMA_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 GDMA_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.

## 27.10 Register Summary

The addresses in this section are relative to GP-SPI2 base address provided in Table 5.3-2 in Chapter 5 *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 |

| Name | Description | Address | Access |
|---|---|---|---|
| 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 | varies |
| SPI_MS_DLEN_REG | SPI data bit length control register | 0x001C | R/W |
| SPI_MISC_REG | SPI misc register | 0x0020 | varies |
| 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 | varies |
| SPI_DIN_NUM_REG | SPI input delay number configuration | 0x0028 | varies |
| SPI_DOUT_MODE_REG | SPI output delay mode configuration | 0x002C | varies |
| **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** | | | |
| SPI_DATE_REG | Version control | 0x00F0 | R/W |

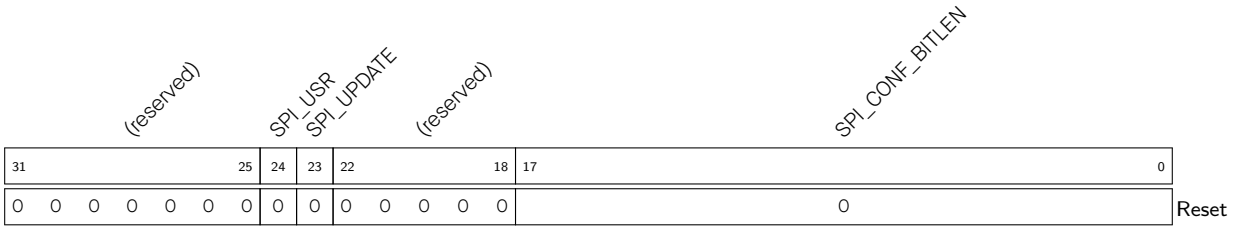
## 27.11 Register

**Notice:**
ESP32-C61 does not currently support the functions associated with the fields marked with HRO access in this section.

The addresses in this section are relative to GP-SPI2 base address provided in Table 5.3-2 in Chapter 5 *System and Memory*.

For how to program reserved fields, please refer to Section *Programming Reserved Register Field*.

**Register 27.1. SPI_CMD_REG (0x0000)**

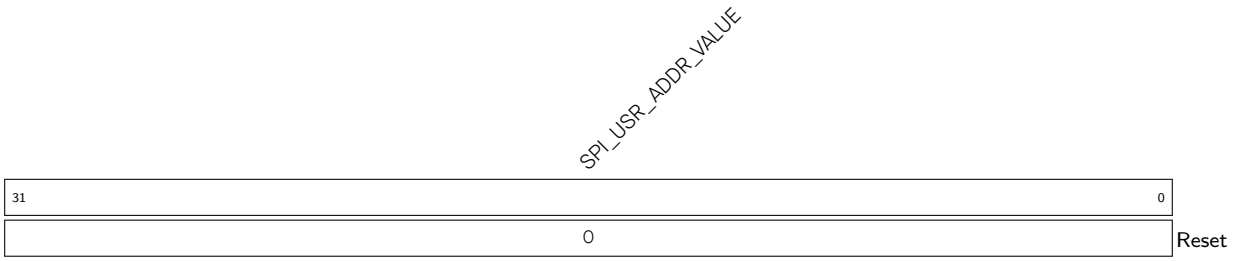



**SPI_CONF_BITLEN** Configures the SPI_CLK cycles of SPI CONF state.
Measurement unit: SPI_CLK clock cycle.
Can be configured in CONF state.
(R/W)

**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 27.2. SPI_ADDR_REG (0x0004)

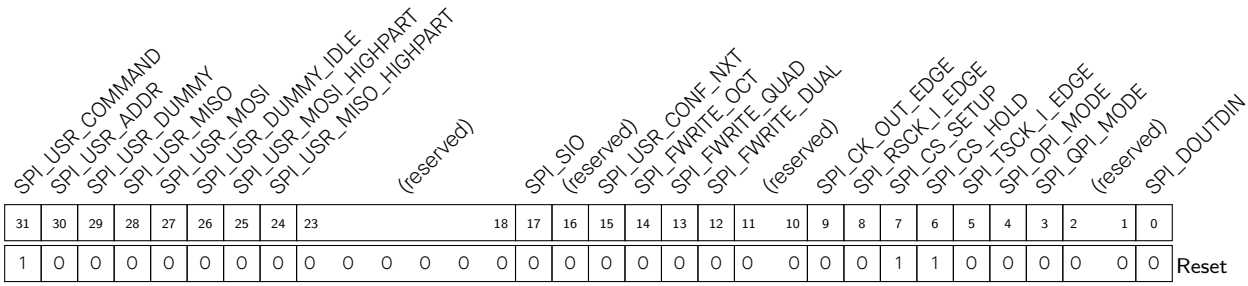



**SPI_USR_ADDR_VALUE** Configures the address to slave.

Can be configured in CONF state.

(R/W)

Register 27.3. 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 the SPI controller operates in OPI mode. Valid only in master mode.

1: Controller is in OPI mode (all in 8-bit mode).

0: Controller is in non-OPI mode. Can be configured in CONF state.

(HRO)

**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)

**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)

**Continued on the next page...**

Register 27.3. SPI_USER_REG (0x0010)

Continued from the previous page...

**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 27.7.2.
(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 to use 8 signals during the read-data phase of write operations.
0: Not enable
1: Enable
Can be configured in CONF state.
(HRO)

**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 27.3. SPI_USER_REG (0x0010)

Continued from the previous page...

**SPI_USR_MISO_HIGHPART** Configures whether or not to enable high part mode, i.e., only access to high part of the buffers: SPI_W8_REG~SPI_W15_REG in read-data phase.
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, i.e., only access to high part of the buffers: SPI_W8_REG~SPI_W15_REG in write-data phase.
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 27.3. 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.

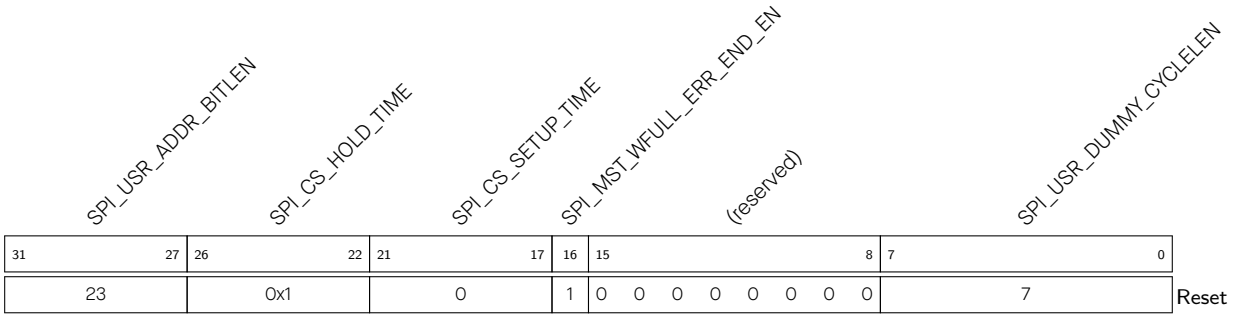
0: Disable

1: Enable

Can be configured in CONF state.

(R/W)

**Register 27.4. 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)

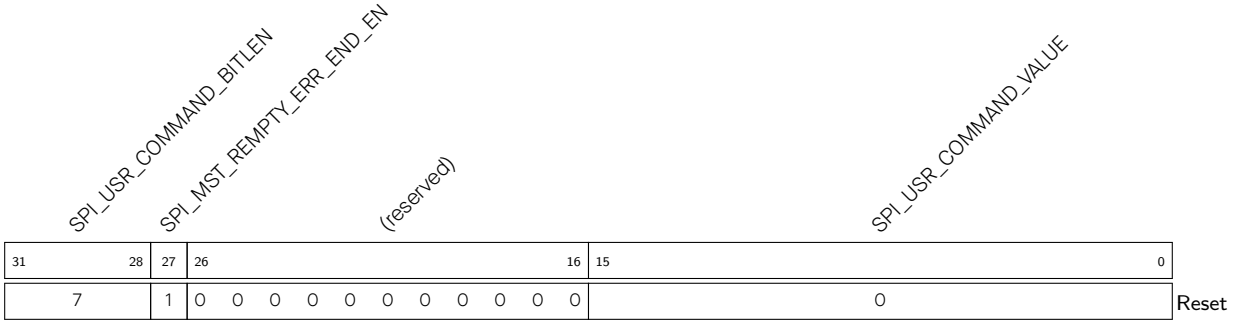
**SPI_USR_ADDR_BITLEN** Configures the bit length in address state.

This value is (expected bit number - 1).

Can be configured in CONF state.

(R/W)

Register 27.5. 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)

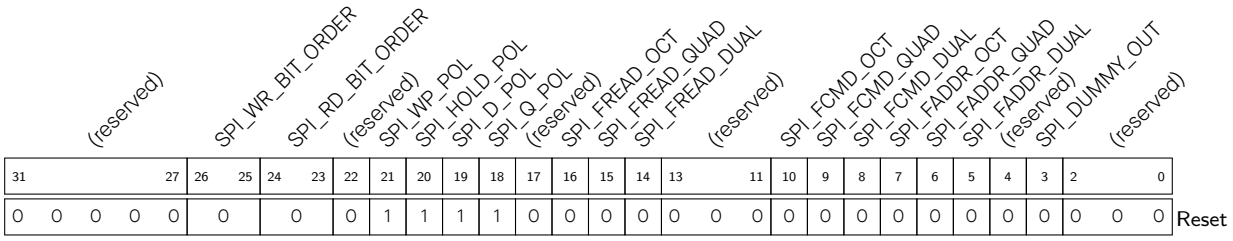
**SPI_USR_COMMAND_BITLEN** Configures the bit length of command state.

This value is (expected bit number - 1).

Can be configured in CONF state.

(R/W)

Register 27.6. SPI_CTRL_REG (0x0008)




**SPI_DUMMY_OUT** Configures whether or not to output the FSPI 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.

(HRO)

**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 27.6. 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.
(HRO)

**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.
(HRO)

**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)

Continued on the next page...

Register 27.6. SPI_CTRL_REG (0x0008)

Continued from the previous page...

**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)

**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

Can be configured in CONF state.

(R/W)

**SPI_WR_BIT_ORDER** Configures the bit order in command (CMD), address (ADDR), and write-data (MOSI) states.

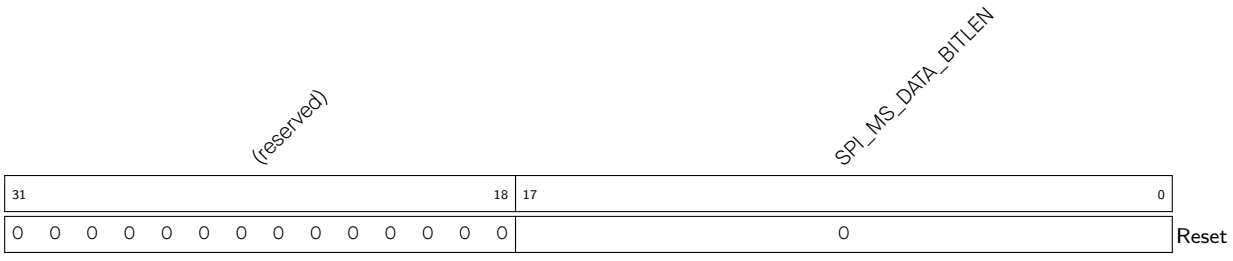
0: MSB first

1: LSB first

Can be configured in CONF state.

(R/W)

Register 27.7. SPI_MS_DLEN_REG (0x001C)



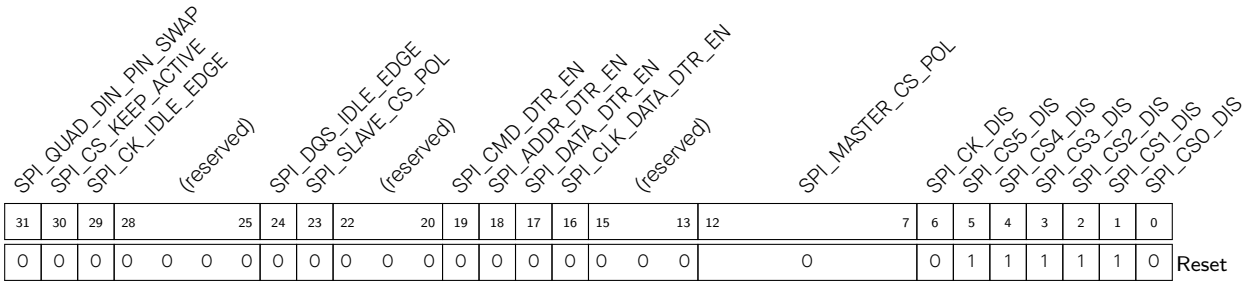

**SPI_MS_DATA_BITLEN** Configures the data bit length of SPI transfer in DMA-controlled master transfer or in CPU-controlled master transfer. Or configures the bit length of SPI RX transfer in DMA-controlled slave transfer.

This value shall be (expected bit_num - 1).

Can be configured in CONF state.

(R/W)

Register 27.8. SPI_MISC_REG (0x0020)




**SPI_CS*n*_DIS** Configures whether or not to disable SPI_CS*n* pin.(N=0∼5)

0: SPI_CS*n* signal is from/to SPI_CS*n* pin.

1: Disable SPI_CS*n* pin.

Can be configured in CONF state.

(R/W)

**SPI_CK_DIS** Configures whether 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 to enable DTR mode for SPI_CLK, DATA, and SPI_DQS when 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.

(HRO)

**SPI_DATA_DTR_EN** Configures whether 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.

(HRO)

**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.

(HRO)

**Continued on the next page...**

Register 27.8. SPI_MISC_REG (0x0020)

Continued from the previous page...

**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.
(HRO)

**SPI_SLAVE_CS_POL** Configures whether or not invert SPI slave input CS polarity.
0: Not change
1: Invert
Can be configured in CONF state.
(R/W)

**SPI_DQS_IDLE_EDGE** Configures the default value of SPI_DQS.
0: Low
1: High
Can be configured in CONF state.
(HRO)

**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)

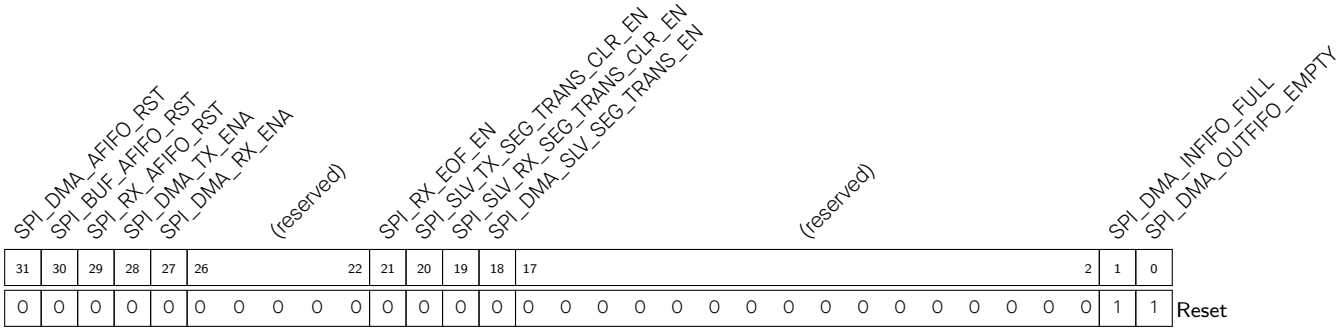
Continued on the next page...

Register 27.8. SPI_MISC_REG (0x0020)

Continued from the previous page...

**SPI_QUAD_DIN_PIN_SWAP** Configures whether to swap SPI QAUD input pins.
0: disable the swap
1: Swap FSPID with FSPIQ, and FSPIWP with FSPIHD
Can be configured in CONF state.
(R/W)

Register 27.9. 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 27.9. 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)

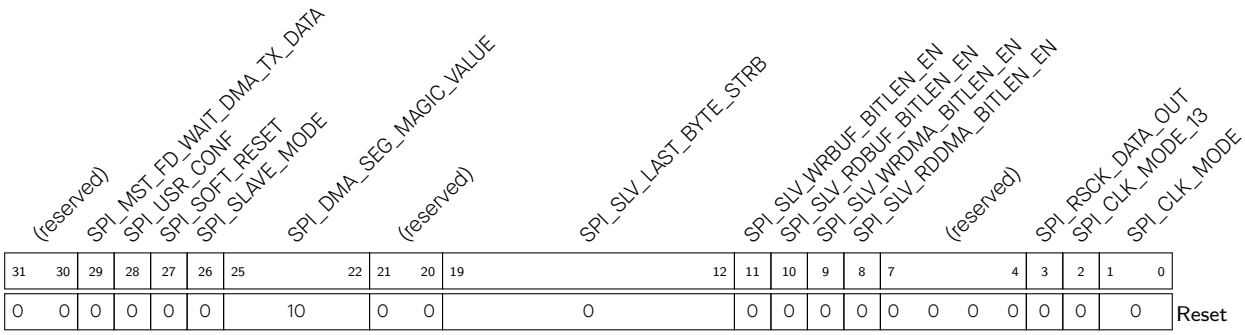
**SPI_DMA_TX_ENA** Configures whether or not to enable DMA-controlled send data transfer.
0: Disable
1: Enable
(R/W)

**SPI_RX_AFIFO_RST** Configures whether or not to reset spi_rx_afifo as shown in Figure 27.5-3 and in Figure 27.5-4.
0: Not reset
1: Reset
spi_rx_afifo is used to receive data in SPI master and slave transfer.
(WT)

**SPI_BUF_AFIFO_RST** Configures whether or not to reset buf_tx_afifo as shown in Figure 27.5-3 and in Figure 27.5-4.
0: Not reset
1: Reset
buf_tx_afifo is used to send data out in CPU-controlled master and slave transfer.
(WT)

**SPI_DMA_AFIFO_RST** Configures whether or not to reset dma_tx_afifo as shown in Figure 27.5-3 and in Figure 27.5-4.
0: Not reset
1: Reset
dma_tx_afifo is used to send data out in DMA-controlled slave transfer.
(WT)

Register 27.10. 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** Configure clock mode.

0: Support SPI clock mode 0 or 2. See Table 27.7-2.

1: Support SPI clock mode 1 or 3. See Table 27.7-2.

(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 27.10. 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 effective bit of the last received data byte in SPI slave FD and HD mode.

(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 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)

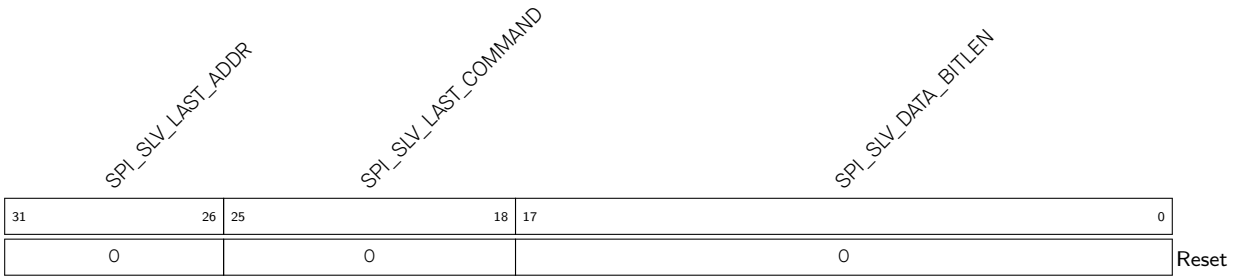
**SPI_MST_FD_WAIT_DMA_TX_DATA** Configures whether or not to wait DMA TX data gets ready before starting SPI transfer in master full-duplex transfer.

0: Not wait

1: Wait

(R/W)

Register 27.11. SPI_SLAVE1_REG (0x00E4)

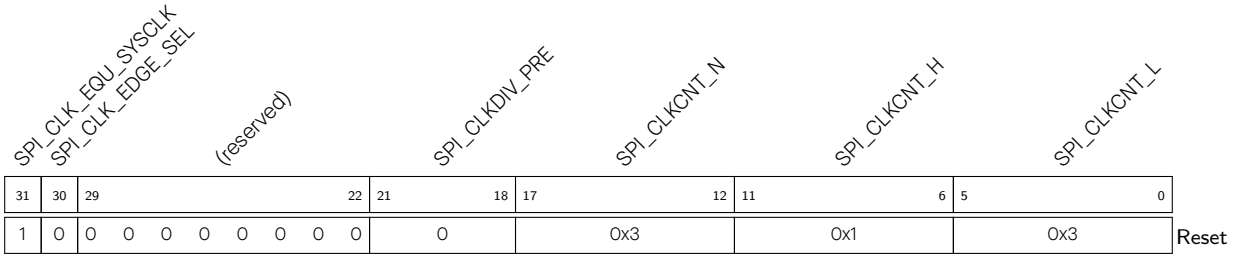



**SPI_SLV_DATA_BITLEN** Configures the transferred data bit length in SPI slave full-/half-duplex modes.
(R/W/SS)

**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 27.12. SPI_CLOCK_REG (0x000C)**




**SPI_CLKCNT_L** 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)

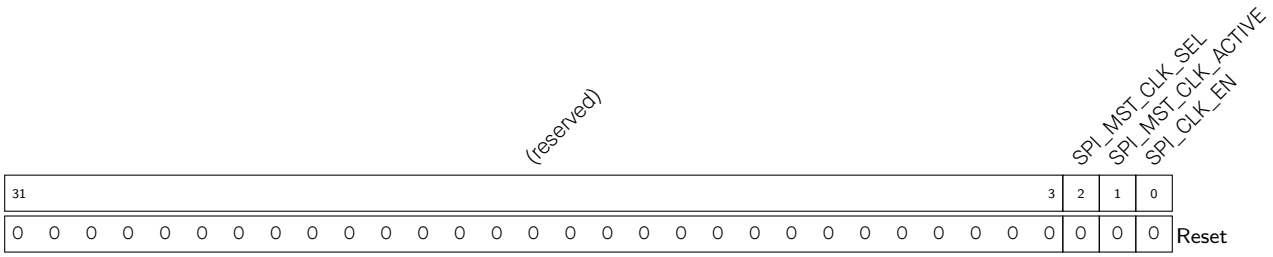
**SPI_CLKCNT_N** Configures the divider of SPI_CLK in master transfer.
SPI_CLK frequency is $f_{clk\_spi\_mst}$/(SPI_CLKDIV_PRE + 1)/(SPI_CLKCNT_N + 1)
Can be configured in CONF state.
(R/W)

**SPI_CLKDIV_PRE** Configures the pre-divider of SPI_CLK in master transfer.
Can be configured in CONF state.
(R/W)

**SPI_CLK_EDGE_SEL** Configures use standard clock sampling edge or delay the sampling edge by half a cycle in master transfer.
0: clock sampling edge is delayed by half a cycle.
1: clock sampling edge is standard.
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 27.13. SPI_CLK_GATE_REG (0x00E8)




**SPI_CLK_EN** Configures whether or not to enable clock gate.

0: Disable

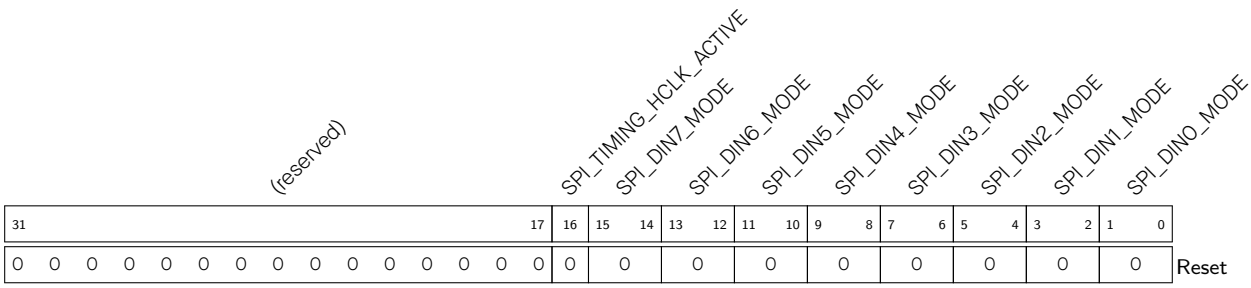
1: Enable

(R/W)

**SPI_MST_CLK_ACTIVE** Set this bit to power on the SPI module clock. (R/W)

**SPI_MST_CLK_SEL** This bit is used to select SPI module clock source in master mode. 1: PLL_CLK_80M. 0: XTAL CLK. (R/W)

Register 27.14. SPI_DIN_MODE_REG (0x0024)




**SPI_DIN0_MODE** Configures the input mode for FSPID 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 FSPIQ 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 FSPIWP 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)

**SPI_DIN3_MODE** Configures the input mode for FSPIHD 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)

**Continued on the next page...**

Register 27.14. SPI_DIN_MODE_REG (0x0024)

Continued from the previous page...

**SPI_DIN4_MODE** Configures the delay of input signals relative to the SPI module clock. 0: Input without delay 1: Input delayed with the rising edge of clk_apb 2: Input delayed with the falling edge of clk_apb 3: Input delayed with spi_clk Valid only in CONF state. (HRO)

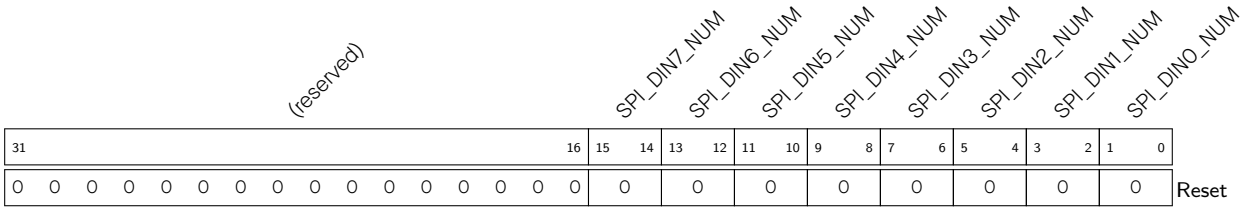
**SPI_DIN5_MODE** Configures the delay of input signals relative to the SPI module clock. 0: Input without delay 1: Input delayed with the rising edge of clk_apb 2: Input delayed with the falling edge of clk_apb 3: Input delayed with spi_clk Valid only in CONF state. (HRO)

**SPI_DIN6_MODE** Configures the delay of input signals relative to the SPI module clock. 0: Input without delay 1: Input delayed with the rising edge of clk_apb 2: Input delayed with the falling edge of clk_apb 3: Input delayed with spi_clk Valid only in CONF state. (HRO)

**SPI_DIN7_MODE** Configures the delay of input signals relative to the SPI module clock. 0: Input without delay 1: Input delayed with the rising edge of clk_apb 2: Input delayed with the falling edge of clk_apb 3: Input delayed with spi_clk Valid only in CONF state. (HRO)

**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 27.15. SPI_DIN_NUM_REG (0x0028)




**SPI_DIN0_NUM** Configures the delays to input signal FSPID 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 FSPIQ 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 FSPIWP 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 FSPIHD 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 27.15. SPI_DIN_NUM_REG (0x0028)

Continued from the previous page...

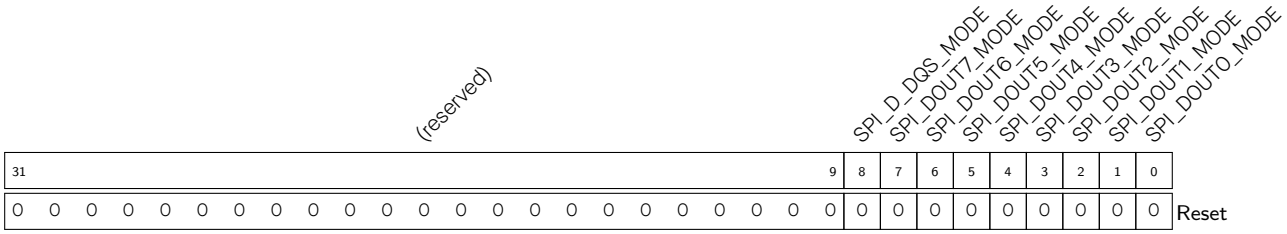
**SPI_DIN4_NUM** Configures the number of SPI module clock cycles by which the input signal DIN4 is delayed.
0: Delayed by 1 cycle.
1: Delayed by 2 cycles.
...
Can be configured in CONF state.
(HRO)

**SPI_DIN5_NUM** Configures the number of SPI module clock cycles by which the input signal DIN5 is delayed.
0: Delayed by 1 cycle.
1: Delayed by 2 cycles.
...
Can be configured in CONF state.
(HRO)

**SPI_DIN6_NUM** Configures the number of SPI module clock cycles by which the input signal DIN6 is delayed.
0: Delayed by 1 cycle.
1: Delayed by 2 cycles.
...
Can be configured in CONF state.
(HRO)

**SPI_DIN7_NUM** Configures the number of SPI module clock cycles by which the input signal DIN7 is delayed.
0: Delayed by 1 cycle.
1: Delayed by 2 cycles.
...
Can be configured in CONF state.
(HRO)

Register 27.16. SPI_DOUT_MODE_REG (0x002C)




**SPI_DOUT0_MODE** Configures the output mode for FSPID 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 FSPIQ 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 FSPIWP 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 FSPIHD 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 whether the output signal DOUT4 is delayed by one SPI module clock cycle at its falling edge.

0: No delay.

1: Delayed by one SPI module clock cycle (falling edge).

Can be configured in CONF state.

(HRO)

**SPI_DOUT5_MODE** Configures whether the output signal DOUT5 is delayed by one SPI module clock cycle at its falling edge.

0: No delay.

1: Delayed by one SPI module clock cycle (falling edge).

Can be configured in CONF state.

(HRO)

**Continued on the next page...**

Register 27.16. SPI_DOUT_MODE_REG (0x002C)

Continued from the previous page...

**SPI_DOUT6_MODE** Configures whether the output signal DOUT6 is delayed by one SPI module clock cycle at its falling edge.
0: No delay.
1: Delayed by one SPI module clock cycle (falling edge).
Can be configured in CONF state.
(HRO)

**SPI_DOUT7_MODE** Configures whether the output signal DOUT7 is delayed by one SPI module clock cycle at its falling edge.
0: No delay.
1: Delayed by one SPI module clock cycle (falling edge).
Can be configured in CONF state.
(HRO)

**SPI_D_DQS_MODE** Configures whether the SPI_DQS output signal is delayed by one SPI module clock cycle at its falling edge.
0: No delay.
1: Delayed by one SPI module clock cycle (falling edge).
Can be configured in CONF state.
(HRO)

Register 27.17. SPI_DMA_INT_ENA_REG (0x0034)




**SPI_DMA_INFIFO_FULL_ERR_INT_ENA** Write 1 to enable the SPI_DMA_INFIFO_FULL_ERR_INT interrupt.
(R/W)

**SPI_DMA_OUTFIFO_EMPTY_ERR_INT_ENA** Write 1 to enable the SPI_DMA_OUTFIFO_EMPTY_ERR_INT interrupt.
(R/W)

**SPI_SLV_EX_QPI_INT_ENA** Write 1 to enable the SPI_SLV_EX_QPI_INT interrupt.
(R/W)

**SPI_SLV_EN_QPI_INT_ENA** Write 1 to enable the SPI_SLV_EN_QPI_INT interrupt.
(R/W)

**SPI_SLV_CMD7_INT_ENA** Write 1 to enable the SPI_SLV_CMD7_INT interrupt.
(R/W)

**SPI_SLV_CMD8_INT_ENA** Write 1 to enable the SPI_SLV_CMD8_INT interrupt.
(R/W)

**SPI_SLV_CMD9_INT_ENA** Write 1 to enable the SPI_SLV_CMD9_INT interrupt.
(R/W)

**SPI_SLV_CMDA_INT_ENA** Write 1 to enable the SPI_SLV_CMDA_INT interrupt.
(R/W)

**SPI_SLV_RD_DMA_DONE_INT_ENA** Write 1 to enable the SPI_SLV_RD_DMA_DONE_INT interrupt.
(R/W)

**SPI_SLV_WR_DMA_DONE_INT_ENA** Write 1 to enable the SPI_SLV_WR_DMA_DONE_INT interrupt.
(R/W)

**SPI_SLV_RD_BUF_DONE_INT_ENA** Write 1 to enable the SPI_SLV_RD_BUF_DONE_INT interrupt.
(R/W)

**SPI_SLV_WR_BUF_DONE_INT_ENA** Write 1 to enable the SPI_SLV_WR_BUF_DONE_INT interrupt.
(R/W)

Continued on the next page...

Register 27.17. SPI_DMA_INT_ENA_REG (0x0034)

Continued from the previous page...

**SPI_TRANS_DONE_INT_ENA** Write 1 to enable the SPI_TRANS_DONE_INT interrupt. (R/W)

**SPI_DMA_SEG_TRANS_DONE_INT_ENA** Write 1 to enable the SPI_DMA_SEG_TRANS_DONE_INT interrupt. (R/W)

**SPI_SEG_MAGIC_ERR_INT_ENA** Write 1 to enable the SPI_SEG_MAGIC_ERR_INT interrupt. (R/W)

**SPI_SLV_BUF_ADDR_ERR_INT_ENA** Write 1 to enable the SPI_SLV_BUF_ADDR_ERR_INT interrupt. (R/W)

**SPI_SLV_CMD_ERR_INT_ENA** Write 1 to enable the SPI_SLV_CMD_ERR_INT interrupt. (R/W)

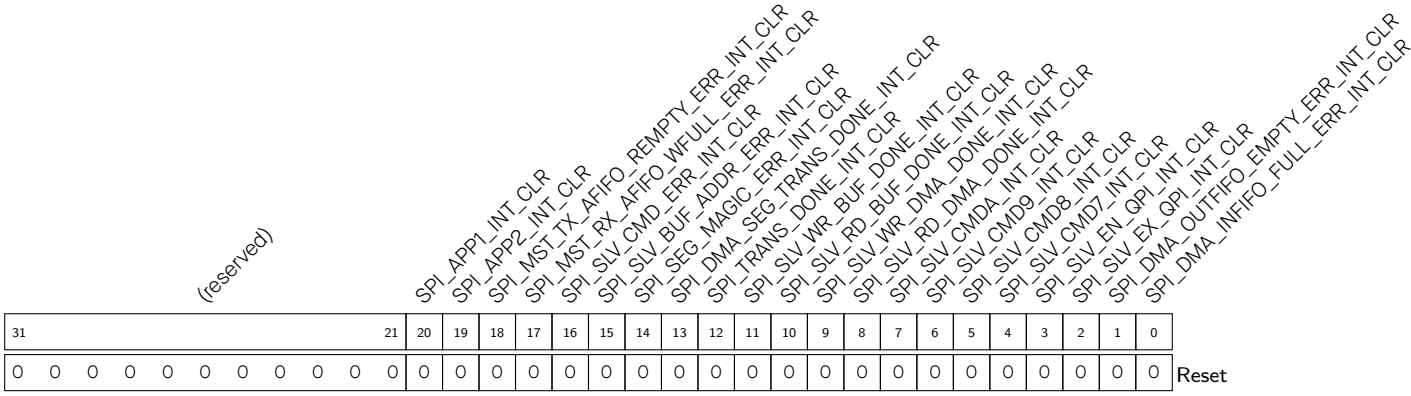
**SPI_MST_RX_AFIFO_WFULL_ERR_INT_ENA** Write 1 to enable the SPI_MST_RX_AFIFO_WFULL_ERR_INT interrupt. (R/W)

**SPI_MST_TX_AFIFO_REMPTY_ERR_INT_ENA** Write 1 to enable the SPI_MST_TX_AFIFO_REMPTY_ERR_INT interrupt. (R/W)

**SPI_APP2_INT_ENA** Write 1 to enable the SPI_APP2_INT interrupt. (R/W)

**SPI_APP1_INT_ENA** Write 1 to enable the SPI_APP1_INT interrupt. (R/W)

Register 27.18. SPI_DMA_INT_CLR_REG (0x0038)




**SPI_DMA_INFIFO_FULL_ERR_INT_CLR** Write 1 to clear the SPI_DMA_INFIFO_FULL_ERR_INT interrupt.
(WT)

**SPI_DMA_OUTFIFO_EMPTY_ERR_INT_CLR** Write 1 to clear the SPI_DMA_OUTFIFO_EMPTY_ERR_INT interrupt.
(WT)

**SPI_SLV_EX_QPI_INT_CLR** Write 1 to clear the SPI_SLV_EX_QPI_INT interrupt.
(WT)

**SPI_SLV_EN_QPI_INT_CLR** Write 1 to clear the SPI_SLV_EN_QPI_INT interrupt.
(WT)

**SPI_SLV_CMD7_INT_CLR** Write 1 to clear the SPI_SLV_CMD7_INT interrupt.
(WT)

**SPI_SLV_CMD8_INT_CLR** Write 1 to clear the SPI_SLV_CMD8_INT interrupt.
(WT)

**SPI_SLV_CMD9_INT_CLR** Write 1 to clear the SPI_SLV_CMD9_INT interrupt.
(WT)

**SPI_SLV_CMDA_INT_CLR** Write 1 to clear the SPI_SLV_CMDA_INT interrupt.
(WT)

**SPI_SLV_RD_DMA_DONE_INT_CLR** Write 1 to clear the SPI_SLV_RD_DMA_DONE_INT interrupt.
(WT)

**SPI_SLV_WR_DMA_DONE_INT_CLR** Write 1 to clear the SPI_SLV_WR_DMA_DONE_INT interrupt.
(WT)

**SPI_SLV_RD_BUF_DONE_INT_CLR** Write 1 to clear the SPI_SLV_RD_BUF_DONE_INT interrupt.
(WT)

**SPI_SLV_WR_BUF_DONE_INT_CLR** Write 1 to clear the SPI_SLV_WR_BUF_DONE_INT interrupt.
(WT)

Continued on the next page...

Register 27.18. SPI_DMA_INT_CLR_REG (0x0038)

Continued from the previous page...

**SPI_TRANS_DONE_INT_CLR** Write 1 to clear the SPI_TRANS_DONE_INT interrupt. (WT)

**SPI_DMA_SEG_TRANS_DONE_INT_CLR** Write 1 to clear the SPI_DMA_SEG_TRANS_DONE_INT interrupt. (WT)

**SPI_SEG_MAGIC_ERR_INT_CLR** Write 1 to clear the SPI_SEG_MAGIC_ERR_INT interrupt. (WT)

**SPI_SLV_BUF_ADDR_ERR_INT_CLR** Write 1 to clear the SPI_SLV_BUF_ADDR_ERR_INT interrupt. (WT)

**SPI_SLV_CMD_ERR_INT_CLR** Write 1 to clear the SPI_SLV_CMD_ERR_INT interrupt. (WT)

**SPI_MST_RX_AFIFO_WFULL_ERR_INT_CLR** Write 1 to clear the SPI_MST_RX_AFIFO_WFULL_ERR_INT interrupt. (WT)

**SPI_MST_TX_AFIFO_REMPTY_ERR_INT_CLR** Write 1 to clear the SPI_MST_TX_AFIFO_REMPTY_ERR_INT interrupt. (WT)

**SPI_APP2_INT_CLR** Write 1 to clear the SPI_APP2_INT interrupt. (WT)

**SPI_APP1_INT_CLR** Write 1 to clear the SPI_APP1_INT interrupt. (WT)

Register 27.19. SPI_DMA_INT_RAW_REG (0x003C)




**SPI_DMA_INFIFO_FULL_ERR_INT_RAW** The raw interrupt status of SPI_DMA_INFIFO_FULL_ERR_INT.
(R/WTC/SS)

**SPI_DMA_OUTFIFO_EMPTY_ERR_INT_RAW** The raw interrupt status of SPI_DMA_OUTFIFO_EMPTY_ERR_INT.
(R/WTC/SS)

**SPI_SLV_EX_QPI_INT_RAW** The raw interrupt status of SPI_SLV_EX_QPI_INT.
(R/WTC/SS)

**SPI_SLV_EN_QPI_INT_RAW** The raw interrupt status of SPI_SLV_EN_QPI_INT.
(R/WTC/SS)

**SPI_SLV_CMD7_INT_RAW** The raw interrupt status of SPI_SLV_CMD7_INT.
(R/WTC/SS)

**SPI_SLV_CMD8_INT_RAW** The raw interrupt status of SPI_SLV_CMD8_INT.
(R/WTC/SS)

**SPI_SLV_CMD9_INT_RAW** The raw interrupt status of SPI_SLV_CMD9_INT.
(R/WTC/SS)

**SPI_SLV_CMDA_INT_RAW** The raw interrupt status of SPI_SLV_CMDA_INT.
(R/WTC/SS)

**SPI_SLV_RD_DMA_DONE_INT_RAW** The raw interrupt status of SPI_SLV_RD_DMA_DONE_INT.
(R/WTC/SS)

**SPI_SLV_WR_DMA_DONE_INT_RAW** The raw interrupt status of SPI_SLV_WR_DMA_DONE_INT.
(R/WTC/SS)

**SPI_SLV_RD_BUF_DONE_INT_RAW** The raw interrupt status of SPI_SLV_RD_BUF_DONE_INT.
(R/WTC/SS)

**SPI_SLV_WR_BUF_DONE_INT_RAW** The raw interrupt status of SPI_SLV_WR_BUF_DONE_INT.
(R/WTC/SS)

**SPI_TRANS_DONE_INT_RAW** The raw interrupt status of SPI_TRANS_DONE_INT.
(R/WTC/SS)

Continued on the next page...

Register 27.19. SPI_DMA_INT_RAW_REG (0x003C)

Continued from the previous page...

**SPI_DMA_SEG_TRANS_DONE_INT_RAW** The raw interrupt status of SPI_DMA_SEG_TRANS_DONE_INT.
(R/WTC/SS)

**SPI_SEG_MAGIC_ERR_INT_RAW** The raw interrupt status of SPI_SEG_MAGIC_ERR_INT.
(R/WTC/SS)

**SPI_SLV_BUF_ADDR_ERR_INT_RAW** Raw status bit for the SPI_SLV_BUF_ADDR_ERR_INT interrupt.
1: The accessed data address during a CPU-controlled FD, Wr_BUF, or Rd_BUF transmission in SPI slave mode exceeds 63.
0: No address error.
(R/WTC/SS)

**SPI_SLV_CMD_ERR_INT_RAW** The raw interrupt status of SPI_SLV_CMD_ERR_INT.
(R/WTC/SS)

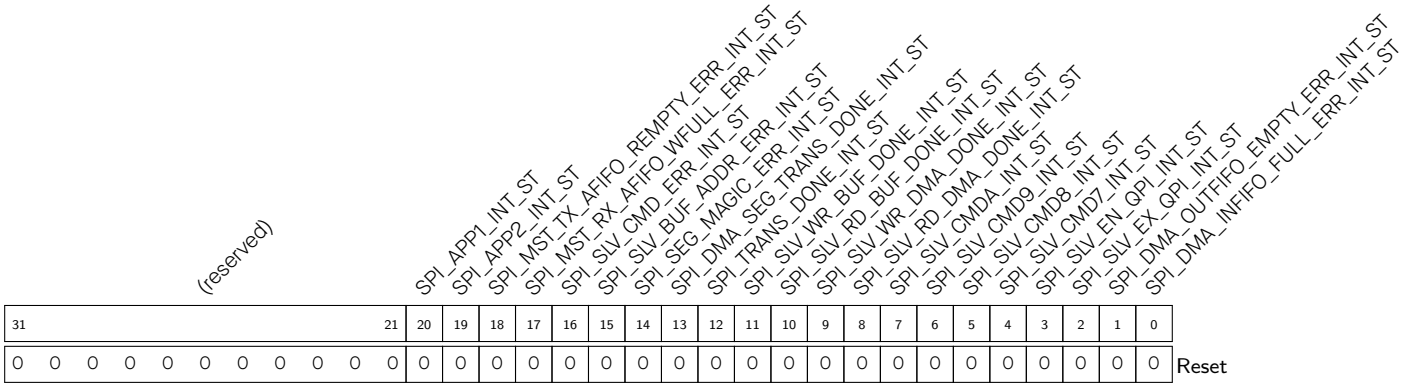
**SPI_MST_RX_AFIFO_WFULL_ERR_INT_RAW** The raw interrupt status of SPI_MST_RX_AFIFO_WFULL_ERR_INT.
(R/WTC/SS)

**SPI_MST_TX_AFIFO_REMPTY_ERR_INT_RAW** The raw interrupt status of SPI_MST_TX_AFIFO_REMPTY_ERR_INT.
(R/WTC/SS)

**SPI_APP2_INT_RAW** The raw interrupt status of SPI_APP2_INT.
The value is only controlled by the application.
(R/WTC/SS)

**SPI_APP1_INT_RAW** The raw interrupt status of SPI_APP1_INT.
The value is only controlled by the application.
(R/WTC/SS)

Register 27.20. SPI_DMA_INT_ST_REG (0x0040)




**SPI_DMA_INFIFO_FULL_ERR_INT_ST** The interrupt status of SPI_DMA_INFIFO_FULL_ERR_INT. (RO)

**SPI_DMA_OUTFIFO_EMPTY_ERR_INT_ST** The interrupt status of SPI_DMA_OUTFIFO_EMPTY_ERR_INT. (RO)

**SPI_SLV_EX_QPI_INT_ST** The interrupt status of SPI_SLV_EX_QPI_INT. (RO)

**SPI_SLV_EN_QPI_INT_ST** The interrupt status of SPI_SLV_EN_QPI_INT. (RO)

**SPI_SLV_CMD7_INT_ST** The interrupt status of SPI_SLV_CMD7_INT. (RO)

**SPI_SLV_CMD8_INT_ST** The interrupt status of SPI_SLV_CMD8_INT. (RO)

**SPI_SLV_CMD9_INT_ST** The interrupt status of SPI_SLV_CMD9_INT. (RO)

**SPI_SLV_CMDA_INT_ST** The interrupt status of SPI_SLV_CMDA_INT. (RO)

**SPI_SLV_RD_DMA_DONE_INT_ST** The interrupt status of SPI_SLV_RD_DMA_DONE_INT. (RO)

**SPI_SLV_WR_DMA_DONE_INT_ST** The interrupt status of SPI_SLV_WR_DMA_DONE_INT. (RO)

**SPI_SLV_RD_BUF_DONE_INT_ST** The interrupt status of SPI_SLV_RD_BUF_DONE_INT. (RO)

**SPI_SLV_WR_BUF_DONE_INT_ST** The interrupt status of SPI_SLV_WR_BUF_DONE_INT. (RO)

Continued on the next page...

Register 27.20. SPI_DMA_INT_ST_REG (0x0040)

Continued from the previous page...

**SPI_TRANS_DONE_INT_ST** The interrupt status of SPI_TRANS_DONE_INT.
(RO)

**SPI_DMA_SEG_TRANS_DONE_INT_ST** The interrupt status of SPI_DMA_SEG_TRANS_DONE_INT.
(RO)

**SPI_SEG_MAGIC_ERR_INT_ST** The interrupt status of SPI_SEG_MAGIC_ERR_INT.
(RO)

**SPI_SLV_BUF_ADDR_ERR_INT_ST** The interrupt status of SPI_SLV_BUF_ADDR_ERR_INT.
(RO)

**SPI_SLV_CMD_ERR_INT_ST** The interrupt status of SPI_SLV_CMD_ERR_INT.
(RO)

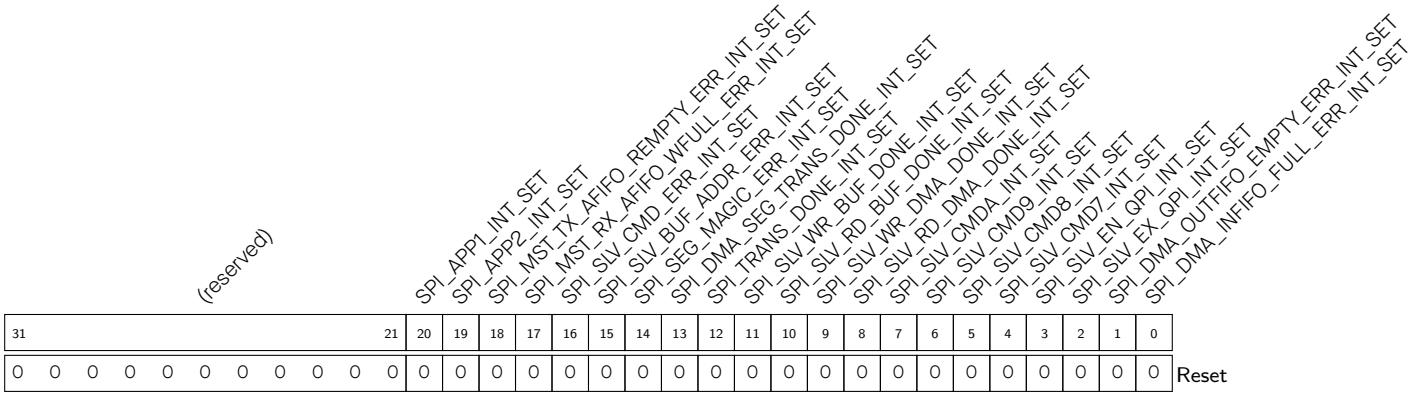
**SPI_MST_RX_AFIFO_WFULL_ERR_INT_ST** The interrupt status of SPI_MST_RX_AFIFO_WFULL_-ERR_INT.
(RO)

**SPI_MST_TX_AFIFO_REMPTY_ERR_INT_ST** The interrupt status of SPI_MST_TX_AFIFO_-REMPTY_ERR_INT.
(RO)

**SPI_APP2_INT_ST** The interrupt status of SPI_APP2_INT.
(RO)

**SPI_APP1_INT_ST** The interrupt status of SPI_APP1_INT.
(RO)

Register 27.21. SPI_DMA_INT_SET_REG (0x0044)




**SPI_DMA_INFIFO_FULL_ERR_INT_SET** Write 1 to set the SPI_DMA_INFIFO_FULL_ERR_INT interrupt.
(WT)

**SPI_DMA_OUTFIFO_EMPTY_ERR_INT_SET** Write 1 to set the SPI_DMA_OUTFIFO_EMPTY_ERR_INT interrupt.
(WT)

**SPI_SLV_EX_QPI_INT_SET** Write 1 to set the SPI_SLV_EX_QPI_INT interrupt.
(WT)

**SPI_SLV_EN_QPI_INT_SET** Write 1 to set the SPI_SLV_EN_QPI_INT interrupt.
(WT)

**SPI_SLV_CMD7_INT_SET** Write 1 to set the SPI_SLV_CMD7_INT interrupt.
(WT)

**SPI_SLV_CMD8_INT_SET** Write 1 to set the SPI_SLV_CMD8_INT interrupt.
(WT)

**SPI_SLV_CMD9_INT_SET** Write 1 to set the SPI_SLV_CMD9_INT interrupt.
(WT)

**SPI_SLV_CMDA_INT_SET** Write 1 to set the SPI_SLV_CMDA_INT interrupt.
(WT)

**SPI_SLV_RD_DMA_DONE_INT_SET** Write 1 to set the SPI_SLV_RD_DMA_DONE_INT interrupt.
(WT)

**SPI_SLV_WR_DMA_DONE_INT_SET** Write 1 to set the SPI_SLV_WR_DMA_DONE_INT interrupt.
(WT)

**SPI_SLV_RD_BUF_DONE_INT_SET** Write 1 to set the SPI_SLV_RD_BUF_DONE_INT interrupt.
(WT)

**SPI_SLV_WR_BUF_DONE_INT_SET** Write 1 to set the SPI_SLV_WR_BUF_DONE_INT interrupt.
(WT)

**SPI_TRANS_DONE_INT_SET** Write 1 to set the SPI_TRANS_DONE_INT interrupt.
(WT)

Continued on the next page...

Register 27.21. SPI_DMA_INT_SET_REG (0x0044)

Continued from the previous page...

**SPI_DMA_SEG_TRANS_DONE_INT_SET** Write 1 to set the SPI_DMA_SEG_TRANS_DONE_INT interrupt.
(WT)

**SPI_SEG_MAGIC_ERR_INT_SET** Write 1 to set the SPI_SEG_MAGIC_ERR_INT interrupt.
(WT)

**SPI_SLV_BUF_ADDR_ERR_INT_SET** Write 1 to set the SPI_SLV_BUF_ADDR_ERR_INT interrupt.
(WT)

**SPI_SLV_CMD_ERR_INT_SET** Write 1 to set the SPI_SLV_CMD_ERR_INT interrupt.
(WT)

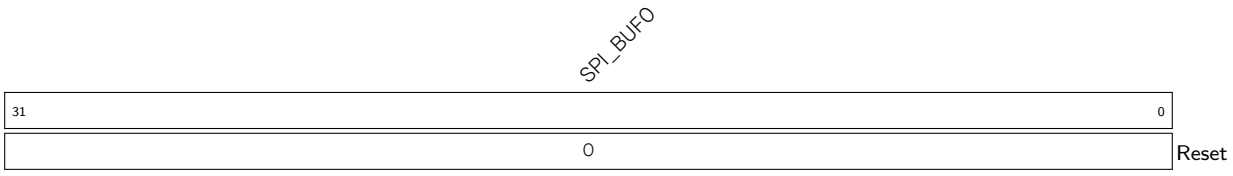
**SPI_MST_RX_AFIFO_WFULL_ERR_INT_SET** Write 1 to set the SPI_MST_RX_AFIFO_WFULL_ERR_INT interrupt.
(WT)

**SPI_MST_TX_AFIFO_REMPTY_ERR_INT_SET** Write 1 to set the SPI_MST_TX_AFIFO_REMPTY_ERR_INT interrupt.
(WT)

**SPI_APP2_INT_SET** Write 1 to set the SPI_APP2_INT interrupt.
(WT)

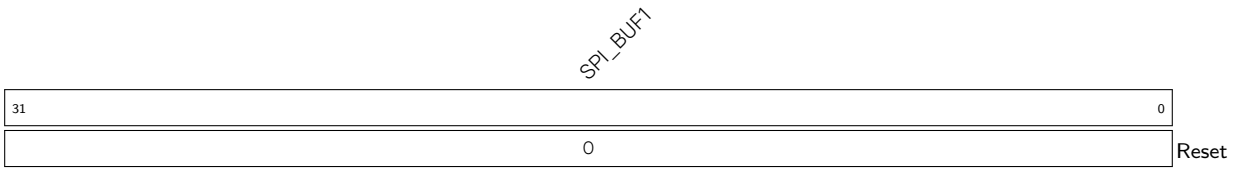
**SPI_APP1_INT_SET** Write 1 to set the SPI_APP1_INT interrupt.
(WT)

Register 27.22. SPI_W0_REG (0x0098)



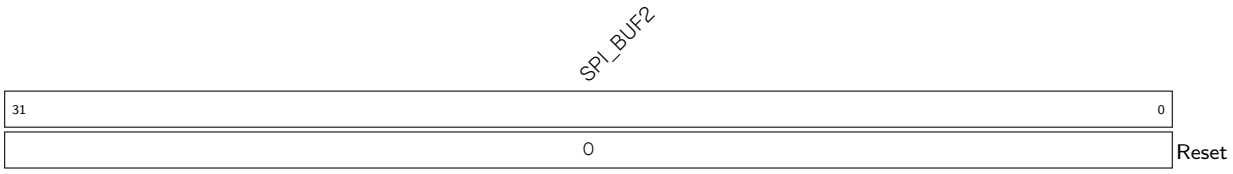

**SPI_BUF0** 32-bit data buffer 0.
(R/W/SS)

Register 27.23. SPI_W1_REG (0x009C)



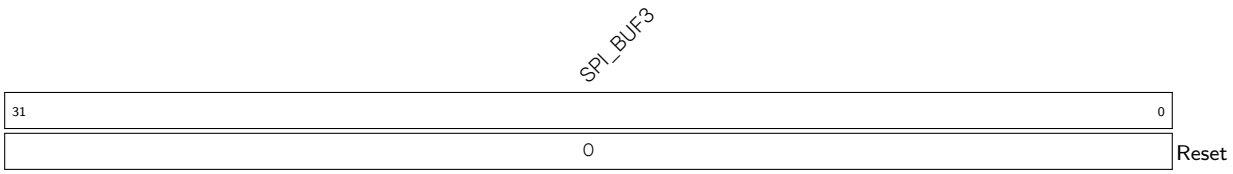

**SPI_BUF1** 32-bit data buffer 1.
(R/W/SS)

Register 27.24. SPI_W2_REG (0x00A0)



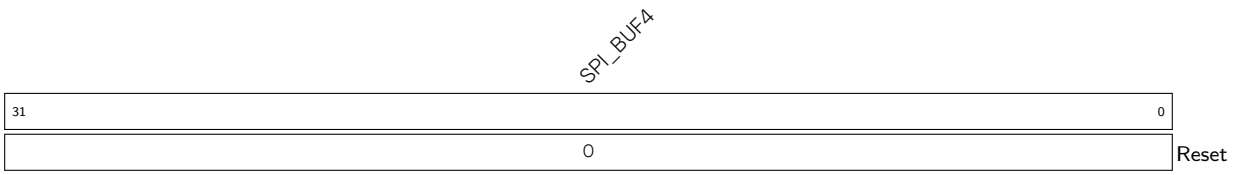

**SPI_BUF2** 32-bit data buffer 2.
(R/W/SS)

Register 27.25. SPI_W3_REG (0x00A4)



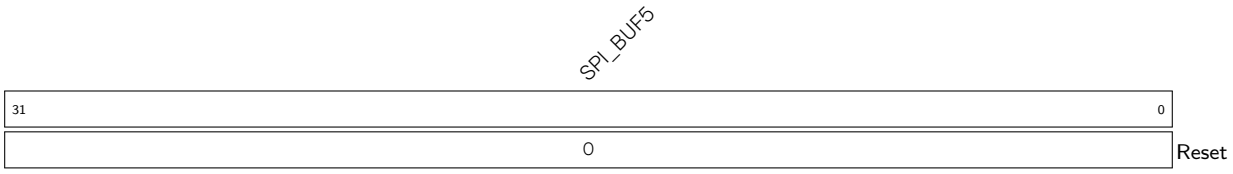

**SPI_BUF3** 32-bit data buffer 3.
(R/W/SS)

Register 27.26. SPI_W4_REG (0x00A8)



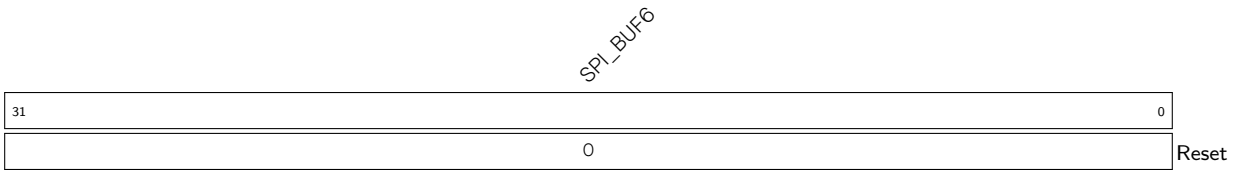

**SPI_BUF4** 32-bit data buffer 4. (R/W/SS)

Register 27.27. SPI_W5_REG (0x00AC)



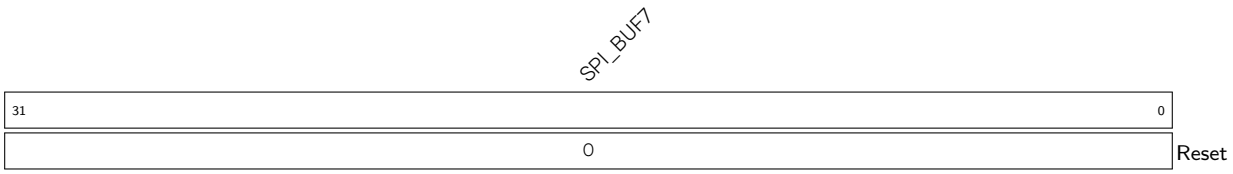

**SPI_BUF5** 32-bit data buffer 5.
(R/W/SS)

Register 27.28. SPI_W6_REG (0x00B0)



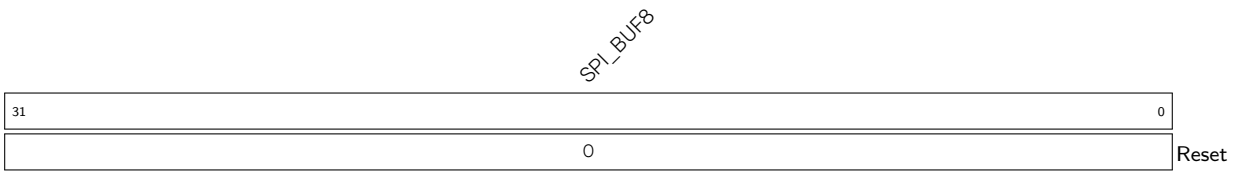

**SPI_BUF6** 32-bit data buffer 6.
(R/W/SS)

Register 27.29. SPI_W7_REG (0x00B4)



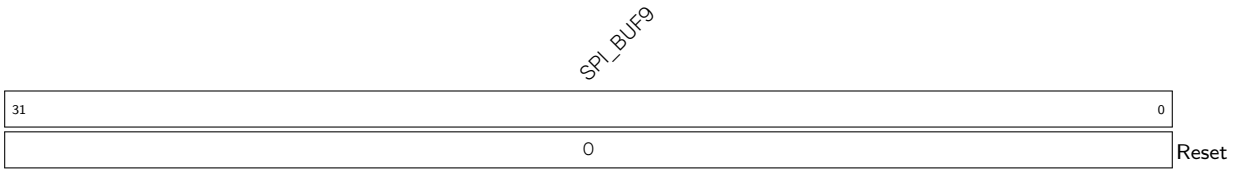

**SPI_BUF7** 32-bit data buffer 7.
(R/W/SS)

Register 27.30. SPI_W8_REG (0x00B8)



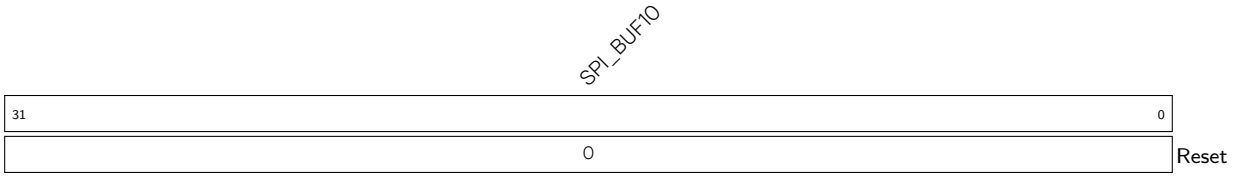

**SPI_BUF8** 32-bit data buffer 8.
(R/W/SS)

Register 27.31. SPI_W9_REG (0x00BC)



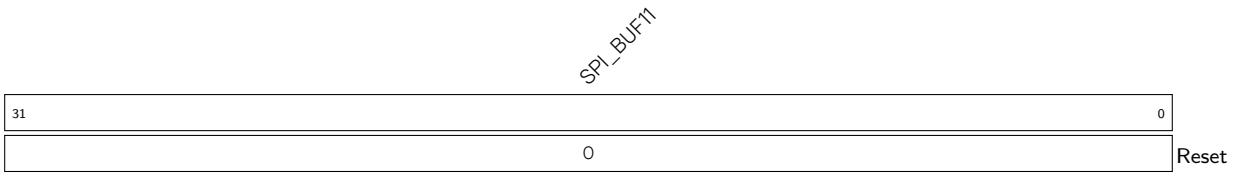

**SPI_BUF9** 32-bit data buffer 9.
(R/W/SS)

Register 27.32. SPI_W10_REG (0x00C0)



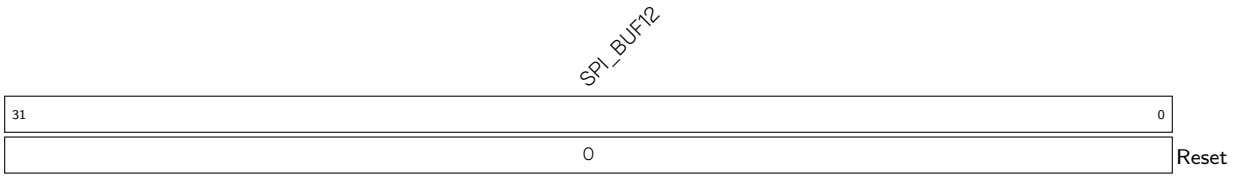

**SPI_BUF10** 32-bit data buffer 10.
(R/W/SS)

Register 27.33. SPI_W11_REG (0x00C4)



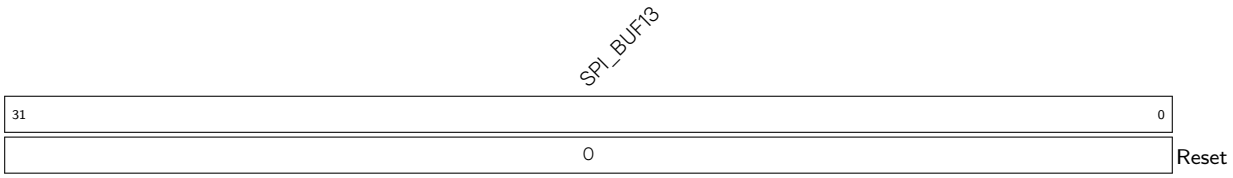

**SPI_BUF11** 32-bit data buffer 11.
(R/W/SS)

Register 27.34. SPI_W12_REG (0x00C8)




**SPI_BUF12** 32-bit data buffer 12.
(R/W/SS)

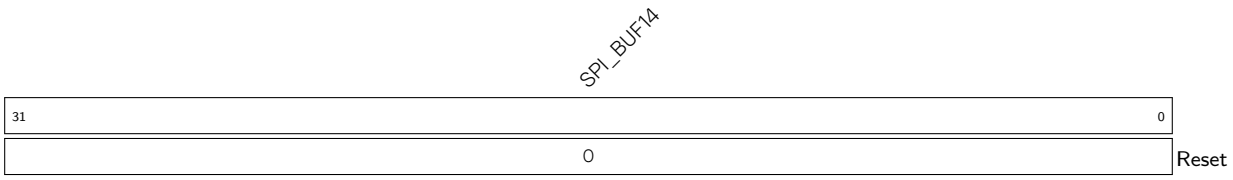
Register 27.35. SPI_W13_REG (0x00CC)




**SPI_BUF13** 32-bit data buffer 13.

(R/W/SS)

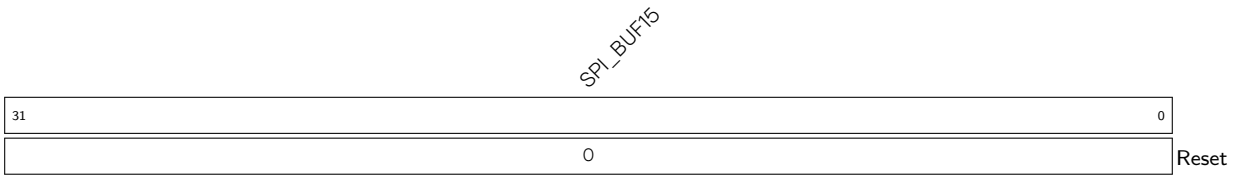
Register 27.36. SPI_W14_REG (0x00D0)




**SPI_BUF14** 32-bit data buffer 14.

(R/W/SS)

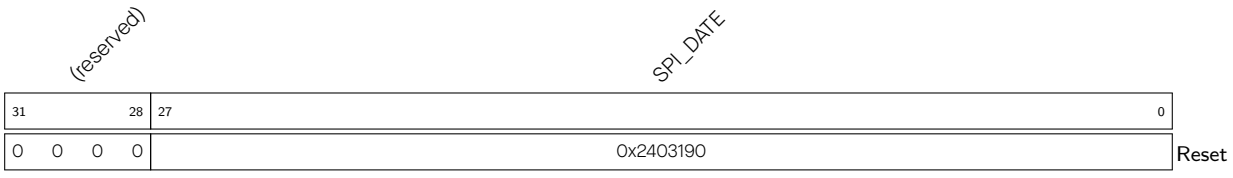
Register 27.37. SPI_W15_REG (0x00D4)




**SPI_BUF15** 32-bit data buffer 15.

(R/W/SS)

Register 27.38. SPI_DATE_REG (0x00F0)




**SPI_DATE** Version control register.

(R/W)

# Chapter 28

# I2C Controller (I2C)

The I2C (Inter-Integrated Circuit) bus allows ESP32-C61 to communicate with multiple external devices. These external devices can share one I2C bus.

ESP32-C61 has one I2C controller in the main system. It can act as a master or a slave (referred to as I2C below).

## 28.1 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.

In I2C communication, the master device controls data transmission by sending signals. Key steps include initiating communication with a start signal, transmitting the address and read/write bit, receiving an acknowledge signal, transferring data, and terminating or restarting the communication.

- **Start signal** The master initiates communication by generating a START condition: pulling the SDA line low while SCL is high
- **Address and read/write bits** 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.
- **Slave acknowledge** 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.
- **Data Transmission** 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.
- **Termination signal** Once the communication has finished, the master sends a STOP condition: pulling SDA up while SCL is high.
- **RSTART signal** 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).

## 28.2 Feature List

The I2C controller of ESP32-C61 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

## 28.3 Architecture Overview

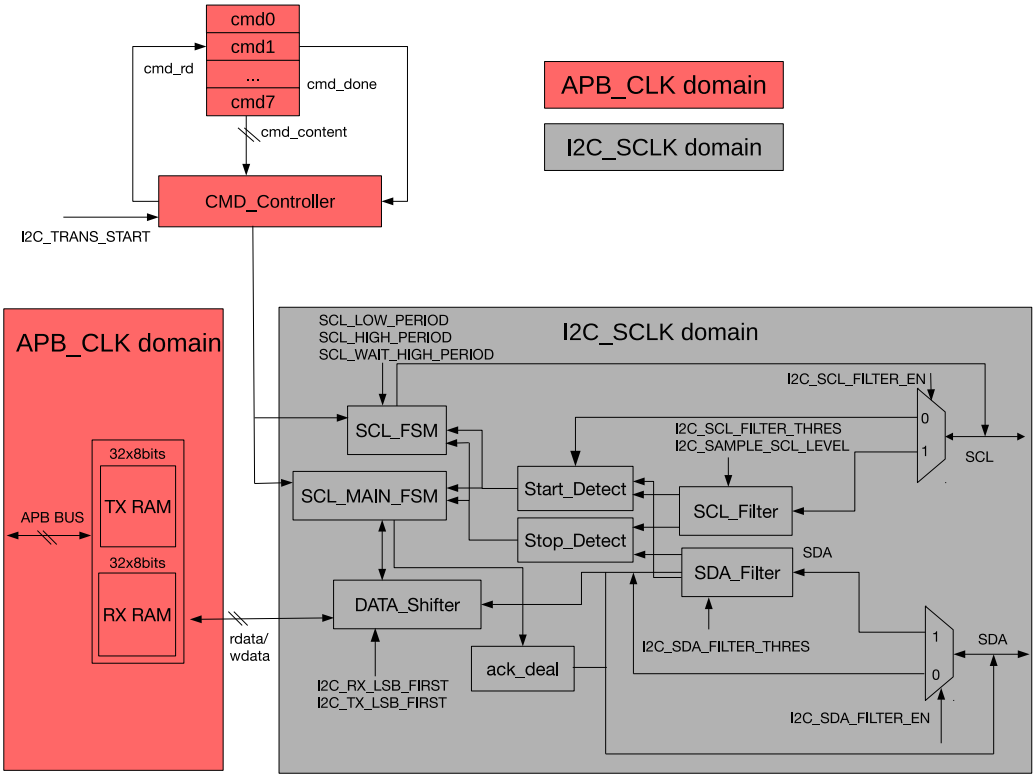



Figure 28.3-1. I2C Master Architecture

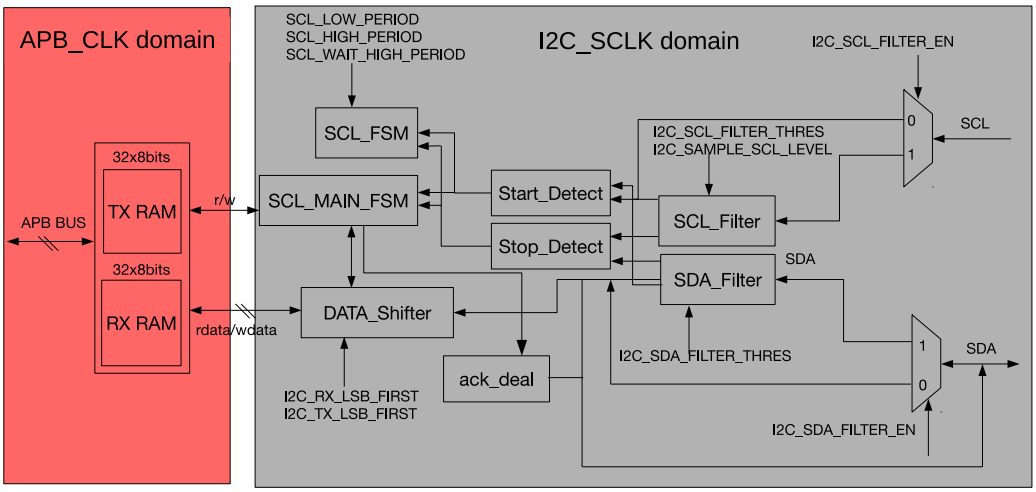



Figure 28.3-2. I2C Slave Architecture

The I2C controller runs either in master mode or slave mode, which is determined by I2C_MS_MODE. Figure 28.3-1 shows the architecture of a master, while Figure 28.3-2 shows that of a slave. The I2C controller has the following main parts:

- Transmit and receive memory (TX/RX RAM): stores data to be transmitted and data received respectively.
- Command controller (CMD_Controller): generates RSTART, STOP, WRITE, READ, and END commands
- SCL clock controller (SCL_FSM): generates the timing sequence conforming to the I2C protocol. Figure 28.3-3 and Table 28.3-1 are the timing diagram and corresponding parameters of the I2C protocol.
- SDA data controller (SCL_MAIN_FSM): controls 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.

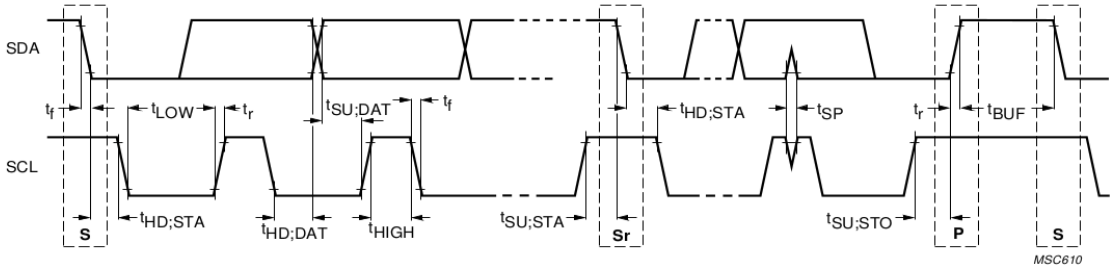



Fig.31 Definition of timing for F/S-mode devices on the I²C-bus.

Figure 28.3-3. 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:<br>for CBUS compatible masters (see NOTE, Section 10.1.3)<br>for I²C-bus devices | $t_{HD;DAT}$ | <br>5.0<br>0(2) | <br>–<br>3.45(3) | <br>–<br>0(2) | <br>–<br>0.9(3) | <br>µs<br>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 28.3-1. I2C Timing Parameters (Cited from Table 5 in The I2C-bus specification Version 2.1)

## 28.4 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-C61 I2C controller. Note that operations may differ between the I2C controller in ESP32-C61 and other masters or slaves on the bus. Please refer to datasheets of individual I2C devices for specific information.

### 28.4.1 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 I2C in the main system to XTAL_CLK or RC_FAST_CLK via PCR_I2C_SCLK_SEL.

The steps to configure the clock source for I2C are as follows:

- Enable the clock source for I2C_SCLK of I2C by configuring PCR_I2C_SCLK_EN to 1.
- When PCR_I2C_SCLK_SEL is 0, the clock source is XTAL_CLK.
- When PCR_I2C_SCLK_SEL is 1, the clock source is RC_FAST_CLK.

The clock source then passes through a fractional divider to generate I2C_SCLK of I2C according to the following formula:

$$I2C\_SCLK\_DIV\_NUM + 1 + \frac{I2C\_SCLK\_DIV\_A}{I2C\_SCLK\_DIV\_B}$$

In the formula, 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 I2C:

- Configure I2C_SCLK_DIV_NUM via PCR_I2C_SCLK_DIV_NUM.
- Configure I2C_SCLK_DIV_A via PCR_I2C_SCLK_DIV_A.
- Configure I2C_SCLK_DIV_B via PCR_I2C_SCLK_DIV_B.

### 28.4.2 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.

### 28.4.3 SCL Clock Stretching

The I2C controller operating in slave mode (i.e., as a slave) can perform clock stretching by holding the SCL line low. This action suspends data transmission, providing more time to process data. This function is enabled by setting the I2C_SLAVE_SCL_STRETCH_EN bit. The time period for releasing the SCL line from stretching is configured by setting the I2C_STRETCH_PROTECT_NUM field to avoid timing sequence errors.

The slave can activate clock stretching by pulling the SCL line low in response to one of four specific events.

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-C61 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 released by setting the I2C_SLAVE_SCL_STRETCH_CLR bit.

### 28.4.4 Generating SCL Pulses in Idle State

Usually, when the I2C bus is idle, the SCL line is held high. The I2C controller in ESP32-C61 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.

### 28.4.5 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 28.4-1.

Table 28.4-1. 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 | |

### 28.4.6 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.

## 28.4.7 Timing Parameters Configuration

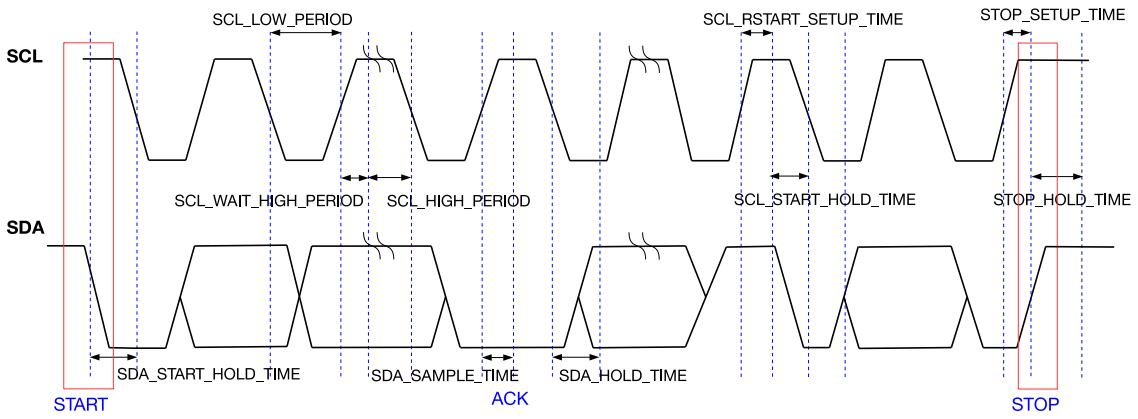



Figure 28.4-1. I2C Timing Diagram

Figure 28.4-1 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

### 28.4.8 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.

### 28.4.9 Command Configuration

When the I2C controller works in master mode, CMD_Controller reads commands from eight sequential command registers and controls SCL_FSM and SCL_MAIN_FSM accordingly.

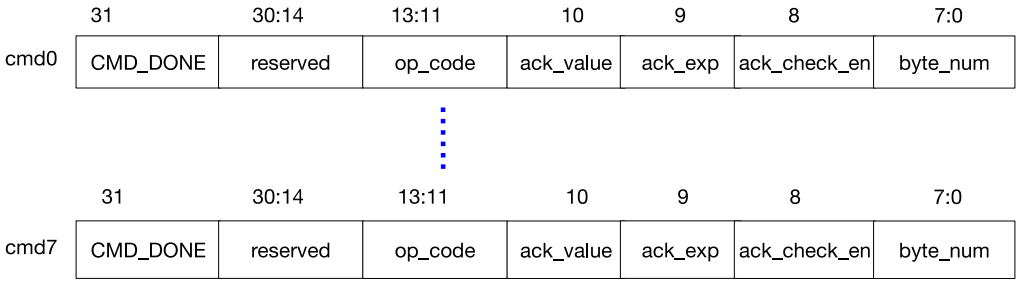



**Figure 28.4-2. Structure of I2C Command Registers**

Command registers, whose structure is illustrated in Figure 28.4-2, 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.

### 28.4.10 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 TX 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).

### 28.4.11 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.

### 28.4.12 Addressing Mode

The ESP32-C61 I2C controller supports 7-bit and 10-bit addressing. 10-bit addressing can be mixed with 7-bit addressing. Besides, the ESP32-C61 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.

### 28.4.13 $R/\overline{W}$ 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.

### 28.4.14 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.

## 28.5 Interrupts

ESP32-C61'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 $2^{I2C\_SCL\_MAIN\_ST\_TO\_I2C+1}$ clock cycles.
- I2C_SCL_ST_TO_INT: Triggered when the state machine SCL_FSM remains unchanged for over $2^{I2C\_SCL\_ST\_TO\_I2C+1}$ 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 $2^{I2C\_TIME\_OUT\_VALUE+1}$ 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_GENERAL_CALL_INT: Triggered when the general call address is received in slave mode.
- 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 10 *Interrupt Matrix* > Section 10.2 *Interrupt Terminology in ESP32-C61*.

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 28.7.1 *I2C Register Summary*.

## 28.6 Programming Procedures

This section provides programming examples for typical communication scenarios. For the convenience of description, I2C masters and slaves in all subsequent figures are ESP32-C61 I2C controller. I2C master is referred to as $I2C_{master}$, and I2C slave is referred to as $I2C_{slave}$.

### 28.6.1 $I2C_{master}$ Writes to $I2C_{slave}$ with a 7-bit Address in One Command Sequence

#### 28.6.1.1 Introduction

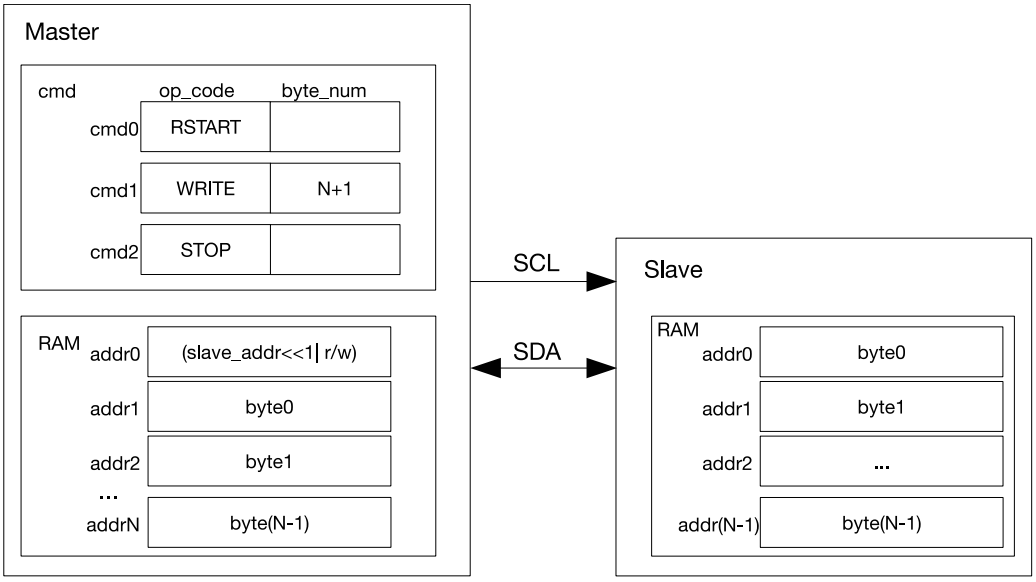



Figure 28.6-1. $I2C_{master}$ Writing to $I2C_{slave}$ with a 7-bit Address

Figure 28.6-1 shows how $I2C_{master}$ writes N bytes of data to $I2C_{slave}$ registers or RAM using 7-bit addressing. As shown in Figure 28.6-1, 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.

#### 28.6.1.2 Configuration Example

1. Configure the timing parameter registers of $I2C_{master}$ and $I2C_{slave}$ according to Section 28.4.7.
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_en | 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 28.4.10.
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) in the WRITE command, $I2C_{master}$ continues data transfer.
   - Not match: If the received ACK value does not match ack_exp in the WRITE command, $I2C_{master}$ 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 28.4.10.
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 28.4.10.

    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.

### 28.6.2 $I2C_{master}$ Writes to $I2C_{slave}$ with a 10-bit Address in One Command Sequence

### 28.6.2.1 Introduction

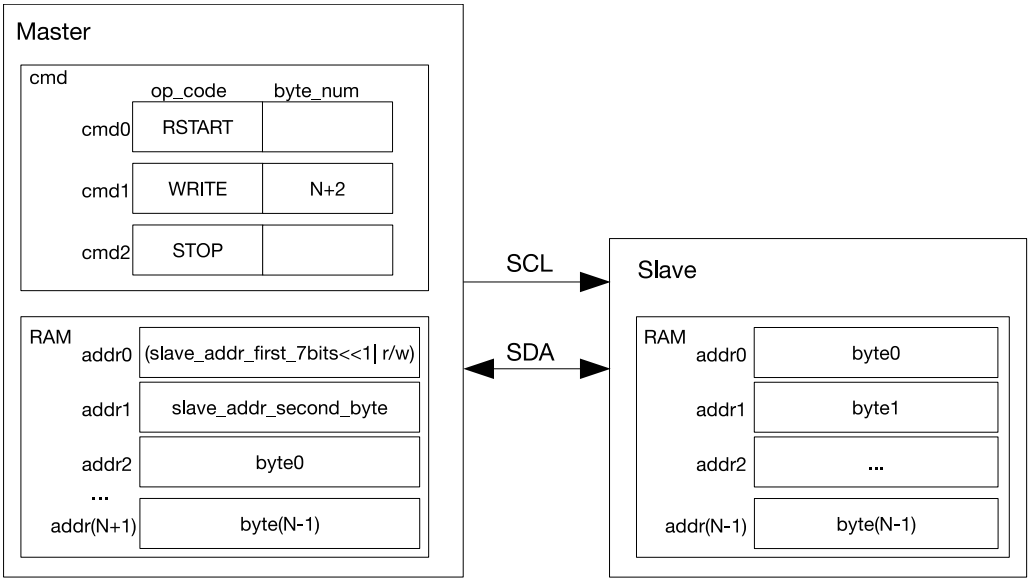



**Figure 28.6-2. $I2C_{master}$ Writing to a Slave with a 10-bit Address**

Figure 28.6-2 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 28.6.1, 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.

### 28.6.2.2 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_en | 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 28.4.10.
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 28.4.10.

    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.

## 28.6.3 $I2C_{master}$ Writes to $I2C_{slave}$ with Two 7-bit Addresses in One Command Sequence

### 28.6.3.1 Introduction

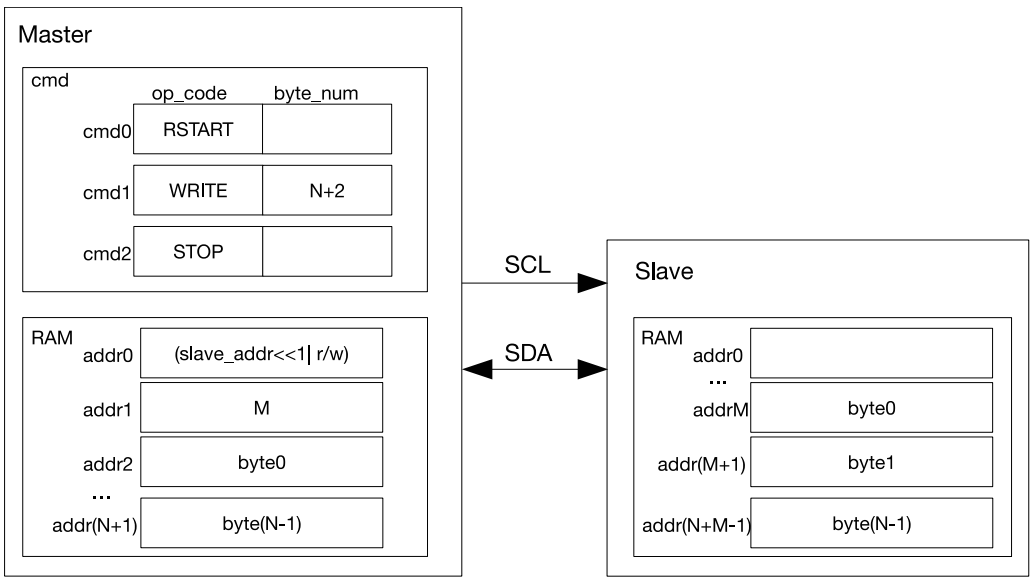



Figure 28.6-3. $I2C_{master}$ Writing to $I2C_{slave}$ with Two 7-bit Addresses

Figure 28.6-3 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 28.6.1, 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 28.6-3). When using double addressing, the slave 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 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.

#### 28.6.3.2 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_en | 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 28.4.10.
13. After data transfer completes, $I2C_{master}$ executes the STOP command, and generates an I2C_TRANS_COMPLETE_INT (master) interrupt.

## 28.6.4 $I2C_{master}$ Writes to $I2C_{slave}$ with a 7-bit Address in Multiple Command Sequences

### 28.6.4.1 Introduction

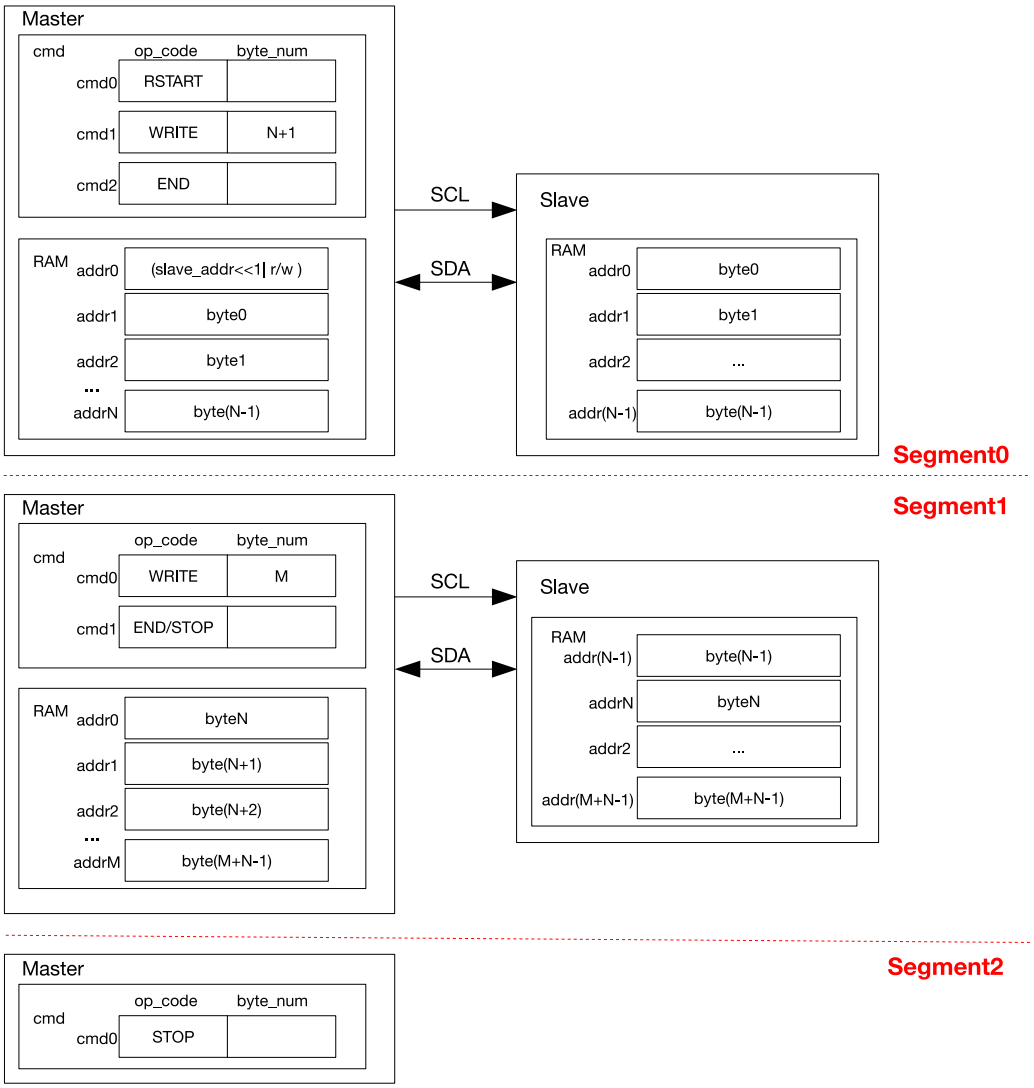



Figure 28.6-4. $I2C_{master}$ Writing to $I2C_{slave}$ with a 7-bit Address in Multiple Sequences

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. Each command sequence ends with 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 28.6-4 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.

#### 28.6.4.2 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_en | 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 28.4.10.
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 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 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_en | 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_en | 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.

## 28.6.5 $I2C_{master}$ Reads $I2C_{slave}$ with a 7-bit Address in One Command Sequence

### 28.6.5.1 Introduction

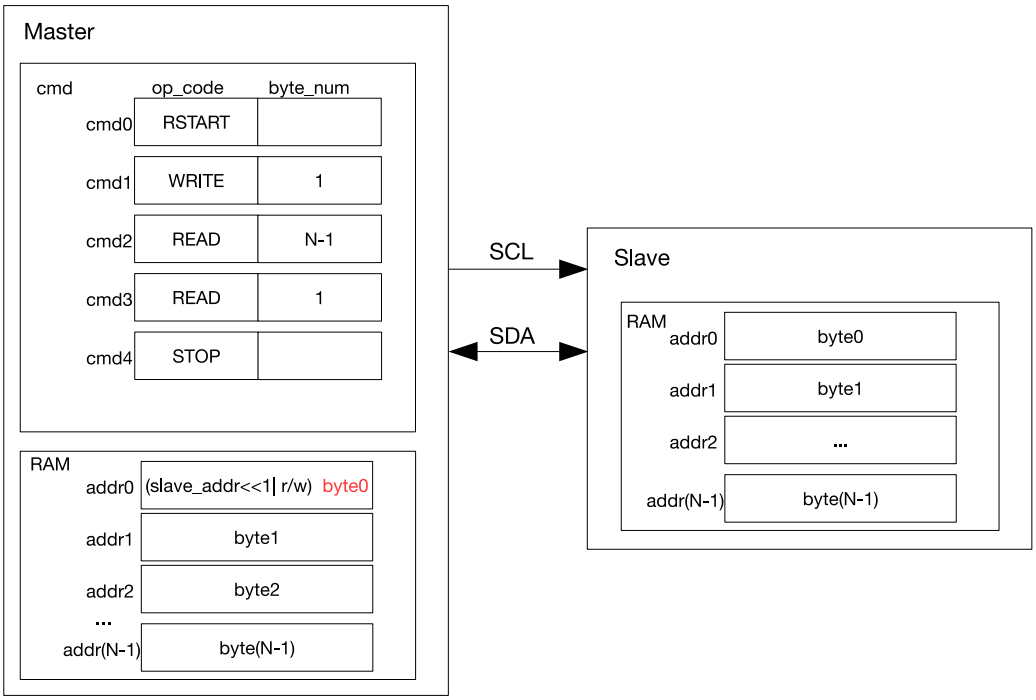



Figure 28.6-5. $I2C_{master}$ Reading $I2C_{slave}$ with a 7-bit Address

Figure 28.6-5 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 28.6-5, $I2C_{master}$ 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 28.6-5.

### 28.6.5.2 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_en | 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 28.4.10.
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 28.4.14.
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 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.
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 28.4.10.
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.

## 28.6.6 $I2C_{master}$ Reads $I2C_{slave}$ with a 10-bit Address in One Command Sequence

### 28.6.6.1 Introduction

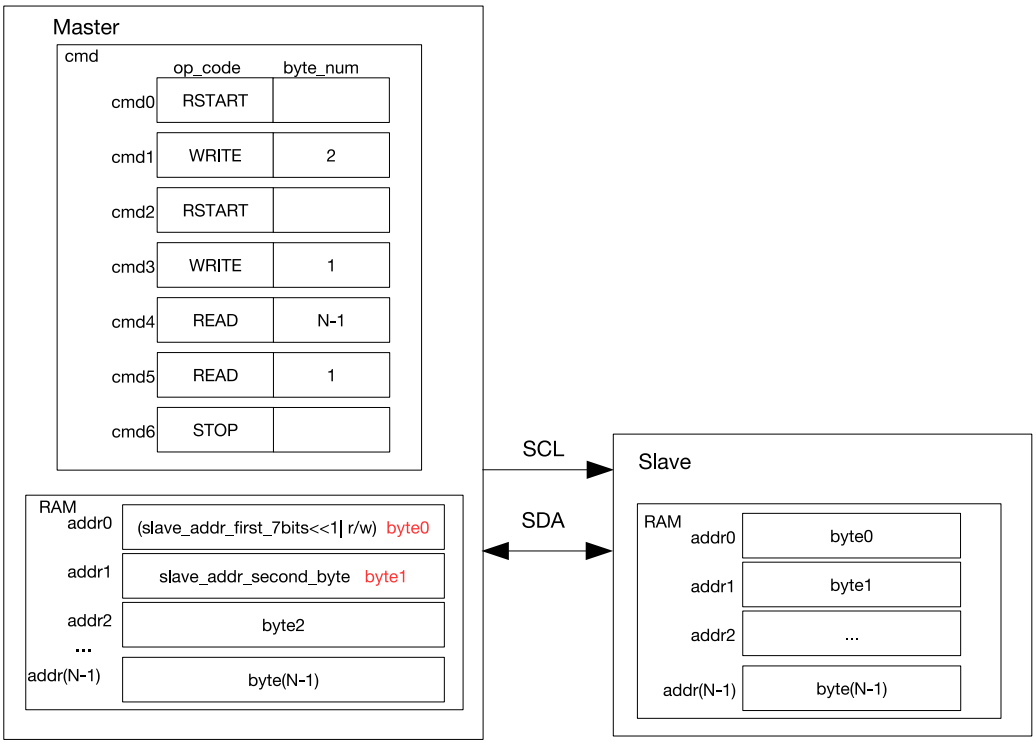



Figure 28.6-6. $I2C_{master}$ Reading $I2C_{slave}$ with a 10-bit Address

Figure 28.6-6 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 28.6.2.

#### 28.6.6.2 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_en | 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 28.4.14.
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 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.

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 28.4.10.
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.

### 28.6.7 $I2C_{master}$ Reads $I2C_{slave}$ with Two 7-bit Addresses in One Command Sequence

#### 28.6.7.1 Introduction

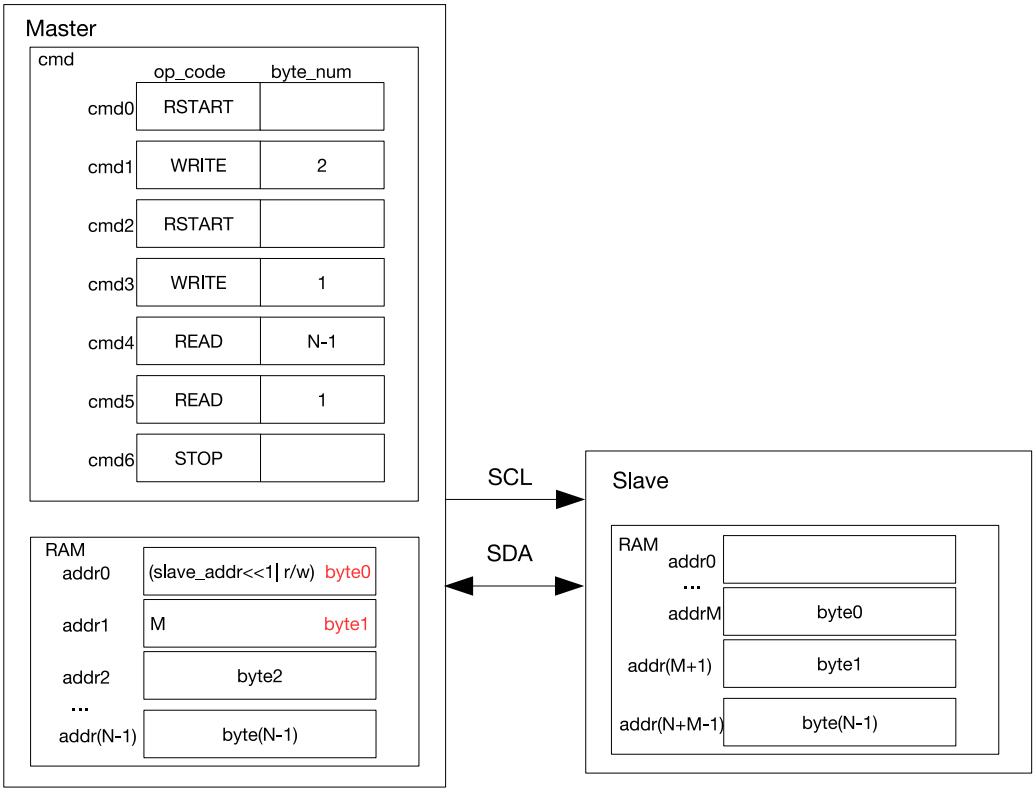



**Figure 28.6-7. $I2C_{master}$ Reading N Bytes of Data from addrM of $I2C_{slave}$ with Two 7-bit Addresses**

Figure 28.6-7 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 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.

#### 28.6.7.2 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_en | 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 28.4.10. 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 28.4.14.
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.

## 28.6.8 $I2C_{master}$ Reads $I2C_{slave}$ with a 7-bit Address in Multiple Command Sequences

### 28.6.8.1 Introduction

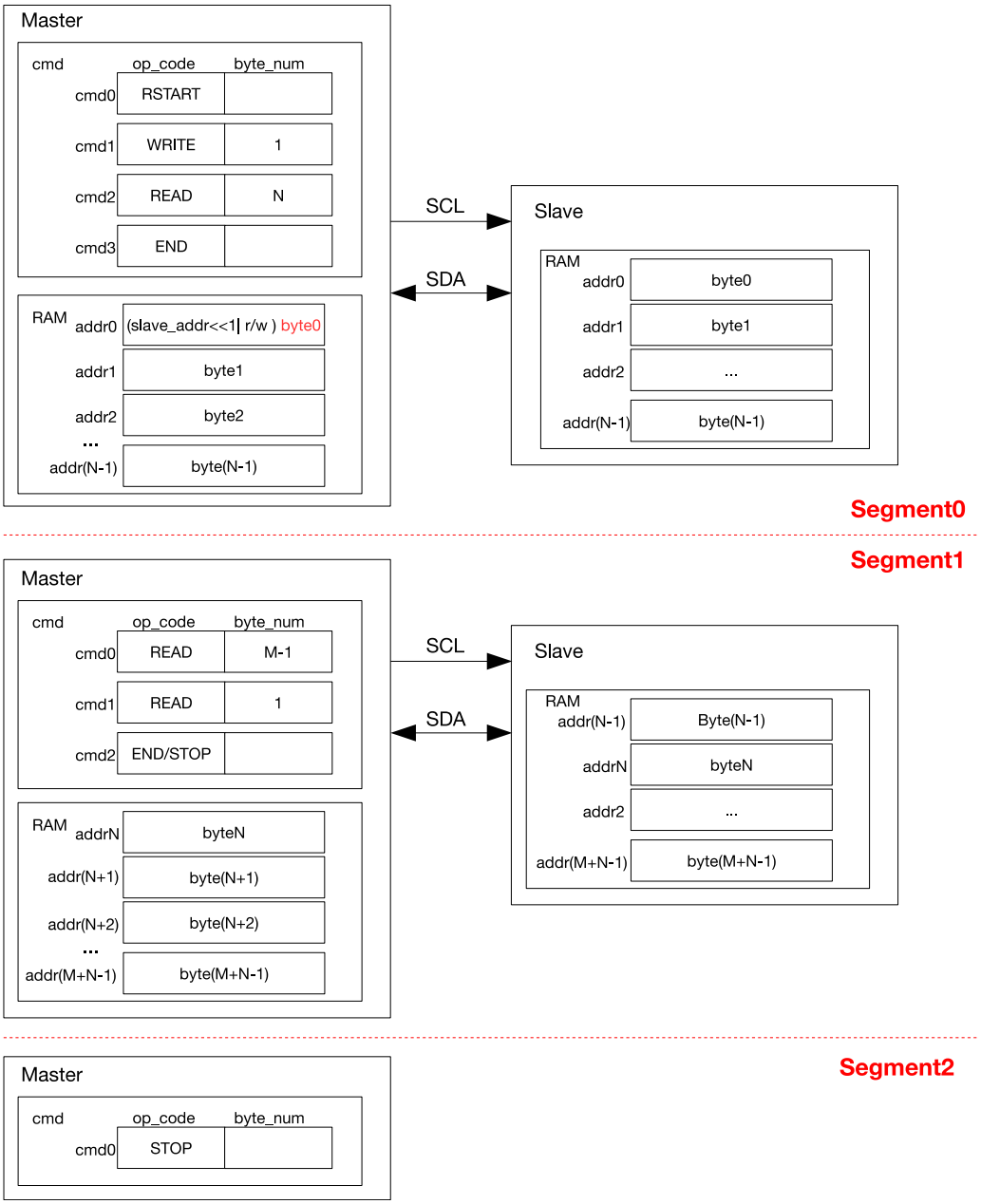



Figure 28.6-8. $I2C_{master}$ Reading $I2C_{slave}$ with a 7-bit Address in Segments

Figure 28.6-8 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 28.6-5, 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.

#### 28.6.8.2 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_en | 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 28.4.14.
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 28.4.10.
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_en | 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_en | 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_en | 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.

## 28.7 Register Summary

### 28.7.1 I2C Register Summary

The addresses in this section are relative to the I2C Controller base address provided in Table 5.3-2 in Chapter *5 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 | 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 |
| I2C_INT_STATUS_REG | Status of captured I2C communication events | 0x002C | RO |

| Name | Description | Address | Access |
|---|---|---|---|
| **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 | RO |
| I2C_RXFIFO_START_ADDR_REG | I2C RX FIFO base address register | 0x0180 | RO |

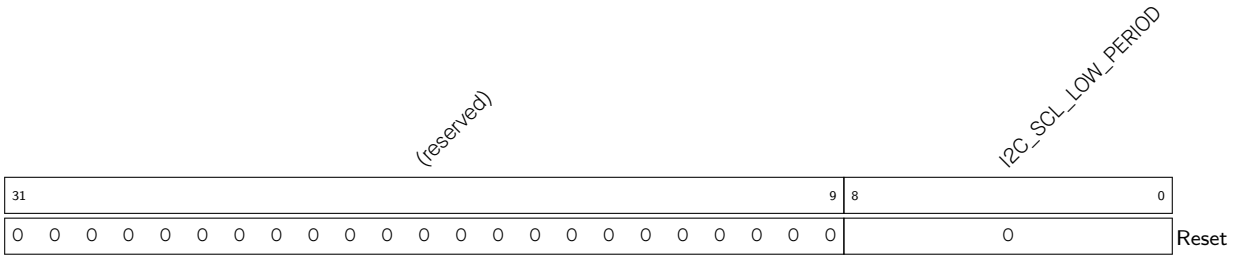
# 28.8 Registers

## 28.8.1 I2C Registers

The addresses in this section are relative to the I2C Controller base address provided in Table 5.3-2 in Chapter *5 System and Memory*.

For how to program reserved fields, please refer to Section Programming Reserved Register Field.

**Register 28.1. I2C_SCL_LOW_PERIOD_REG (0x0000)**

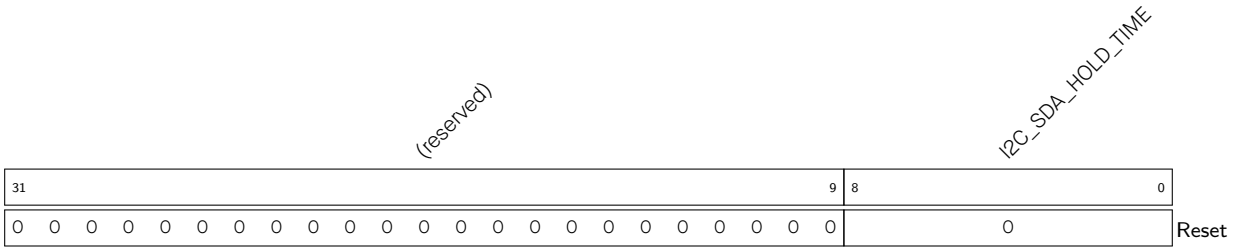



**I2C_SCL_LOW_PERIOD** Configures the low level width of the SCL clock in master mode.

Measurement unit: I2C_SCLK clock cycles

(R/W)

**Register 28.2. I2C_SDA_HOLD_REG (0x0030)**

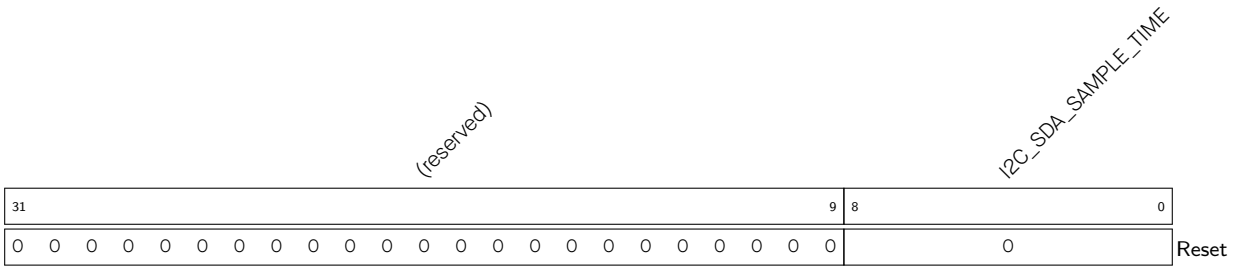



**I2C_SDA_HOLD_TIME** Configures the time to hold the data after the falling edge of SCL.

Measurement unit: I2C_SCLK clock cycles

(R/W)

Register 28.3. I2C_SDA_SAMPLE_REG (0x0034)




**I2C_SDA_SAMPLE_TIME** Configures the time for sampling SDA.

Measurement unit: I2C_SCLK clock cycles

(R/W)

Register 28.4. 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

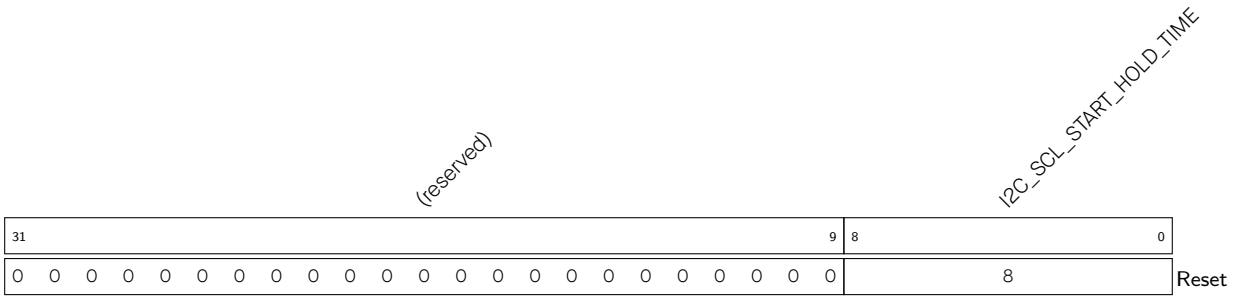
(R/W)

**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 28.5. I2C_SCL_START_HOLD_REG (0x0040)

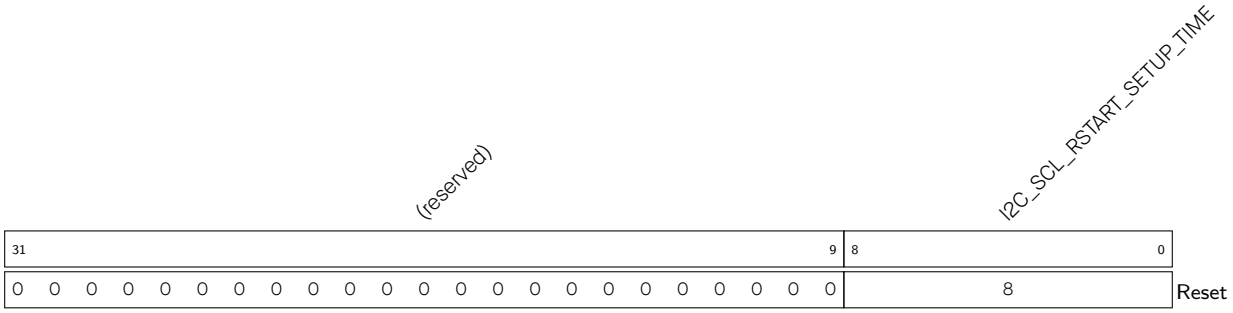



**I2C_SCL_START_HOLD_TIME** Configures the time between the falling edge of SDA and the falling edge of SCL for a START condition.

Measurement unit: I2C_SCLK clock cycles

(R/W)

Register 28.6. I2C_SCL_RSTART_SETUP_REG (0x0044)

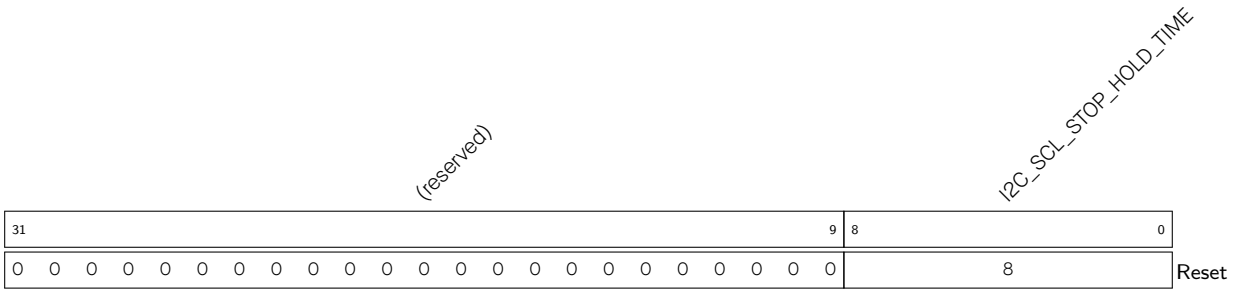



**I2C_SCL_RSTART_SETUP_TIME** Configures the time between the rising edge of SCL and the negative edge of SDA for a RESTART condition.

Measurement unit: I2C_SCLK clock cycles

(R/W)

Register 28.7. I2C_SCL_STOP_HOLD_REG (0x0048)

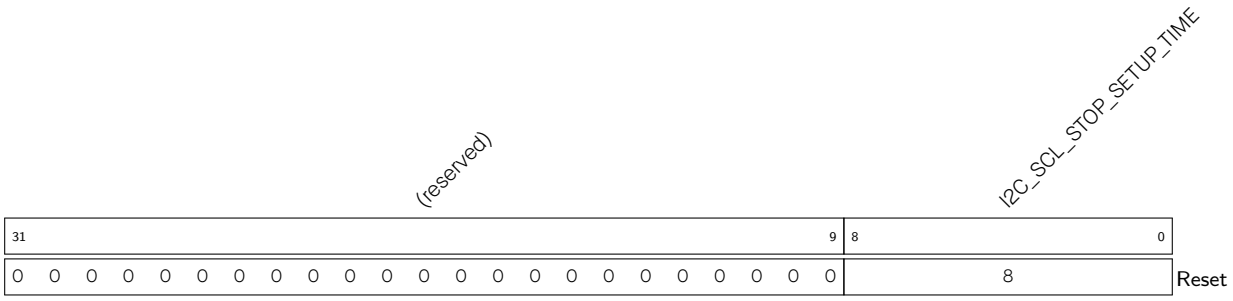



**I2C_SCL_STOP_HOLD_TIME** Configures the delay after the STOP condition.

Measurement unit: I2C_SCLK clock cycles

(R/W)

Register 28.8. I2C_SCL_STOP_SETUP_REG (0x004C)

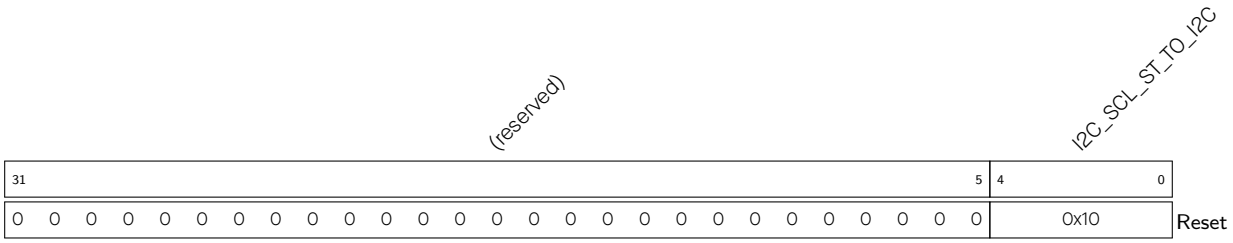



**I2C_SCL_STOP_SETUP_TIME** Configures the time between the rising edge of SCL and the rising edge of SDA.

Measurement unit: I2C_SCLK clock cycles

(R/W)

Register 28.9. I2C_SCL_ST_TIME_OUT_REG (0x0078)



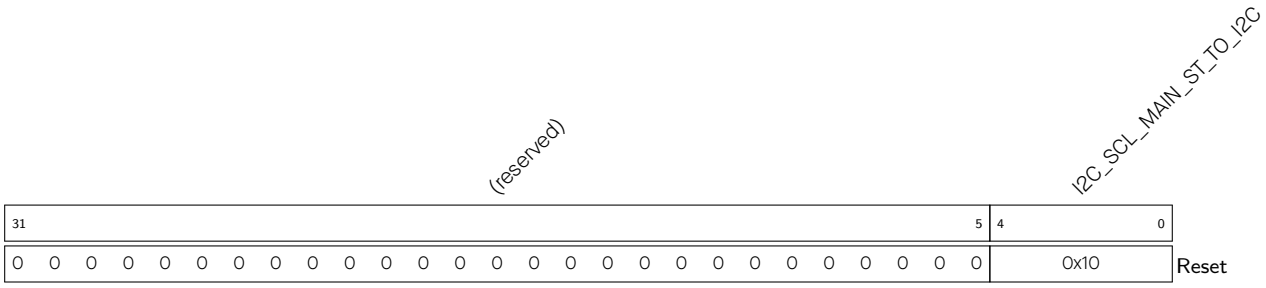

**I2C_SCL_ST_TO_I2C** Configures the threshold value of SCL_FSM state unchanged period. It should be no more than 23, more than 1.

$$\text{Maximum Clock Cycle Threshold} = 2^{\text{I2C\_SCL\_ST\_TO\_I2C}+1}-1$$

Measurement unit: I2C_SCLK clock cycles

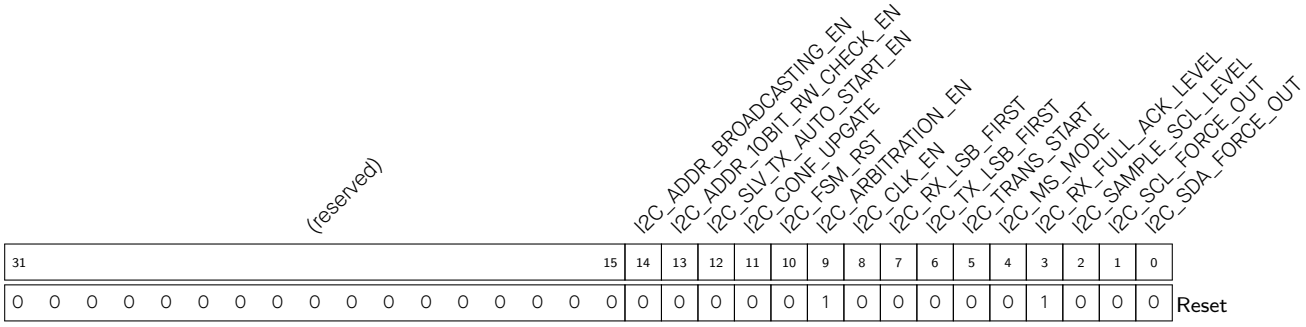
(R/W)

Register 28.10. 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. The actual period is 2^(I2C_SCL_MAIN_ST_TO_I2C+1)–1. Measurement unit: I2C_SCLK clock cycles
(R/W)

Register 28.11. 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 28.11. 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)

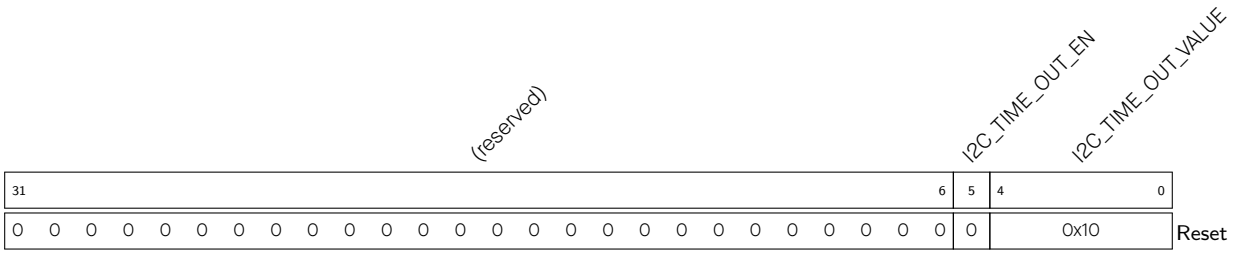
**I2C_ADDR_BROADCASTING_EN** Configures to support the 7-bit general call function.

0: Not support

1: Support

(R/W)

Register 28.12. 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)

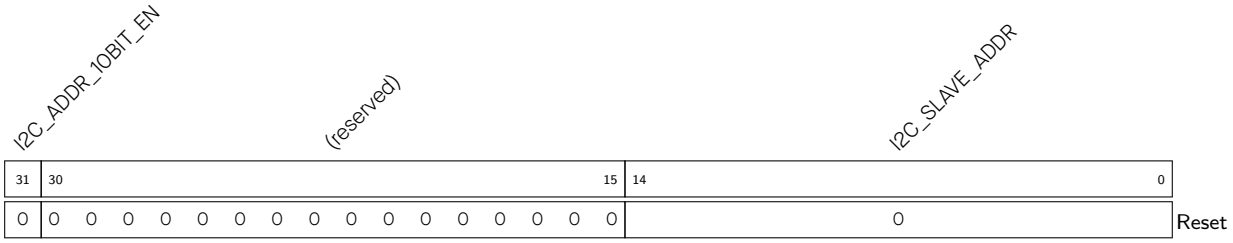
**I2C_TIME_OUT_EN** Configures to enable time out control.

0: No effect

1: Enable

(R/W)

Register 28.13. I2C_SLAVE_ADDR_REG (0x0010)




**I2C_SLAVE_ADDR** Configure the slave address of I2C Slave. (R/W)

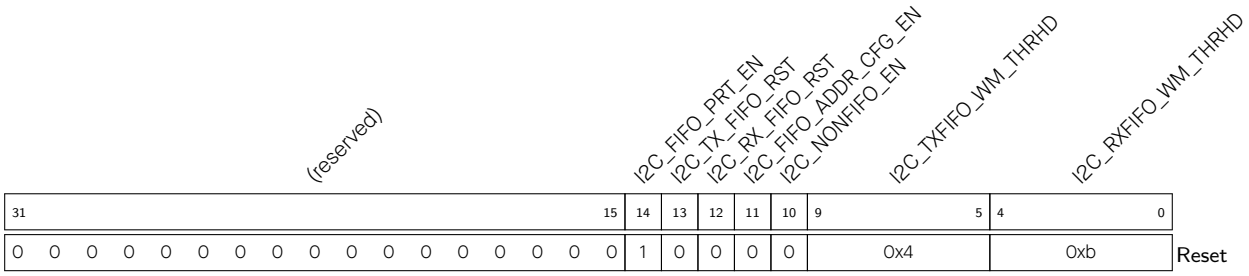
**I2C_ADDR_10BIT_EN** Configures to enable the slave 10-bit addressing mode in master mode.

0: No effect

1: Enable

(R/W)

Register 28.14. 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)

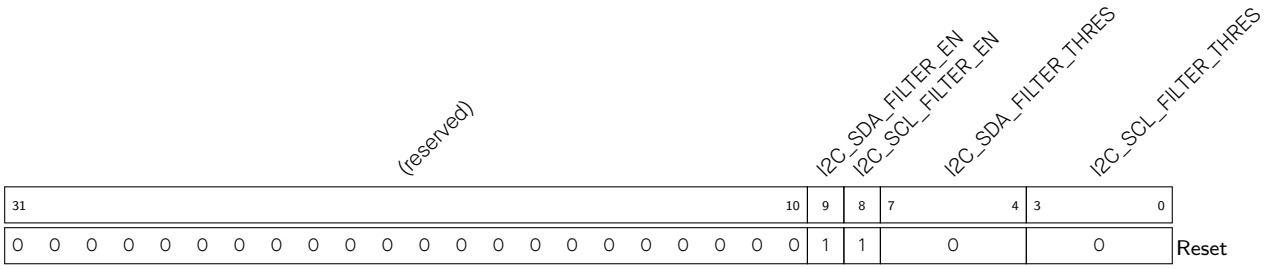
**I2C_FIFO_ADDR_CFG_EN** Configures the slave to enable dual address mode. When this mode is enabled, the byte received after the I2C address byte represents the offset address in the I2C Slave RAM.
0: Disable
1: Enable
(R/W)

**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 28.15. I2C_FILTER_CFG_REG (0x0050)



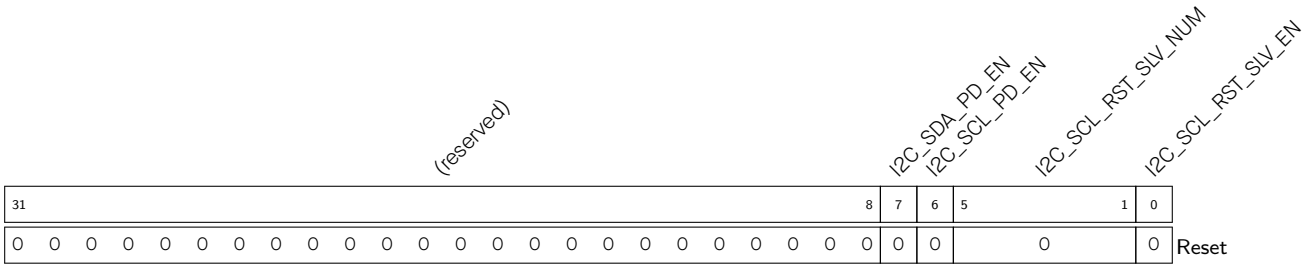

**I2C_SCL_FILTER_THRES** Configures the threshold pulse width to be filtered on SCL. When a pulse on the SCL input has smaller width than this register value, the I2C controller will ignore that pulse. Measurement unit: I2C_SCLK clock cycles
(R/W)

**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 28.16. 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.

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)

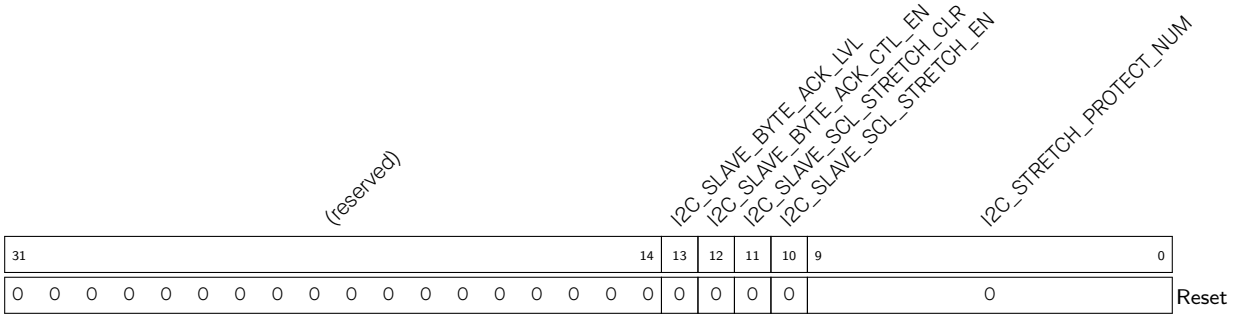
**I2C_SDA_PD_EN** Configures to power down the I2C output SDA line.

0: Not power down.

1: Not work and power down.

Valid only when I2C_SDA_FORCE_OUT is 1. (R/W)

Register 28.17. 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)

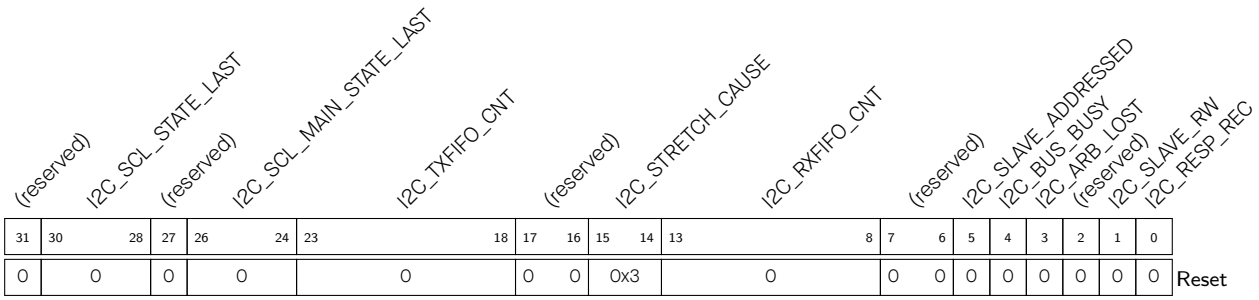
**I2C_SLAVE_BYTE_ACK_LVL** Configures the ACK level when slave controlling ACK level function is enabled.

0: Low level

1: High level

(R/W)

Register 28.18. 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 28.18. I2C_SR_REG (0x0008)

Continued from the previous page...

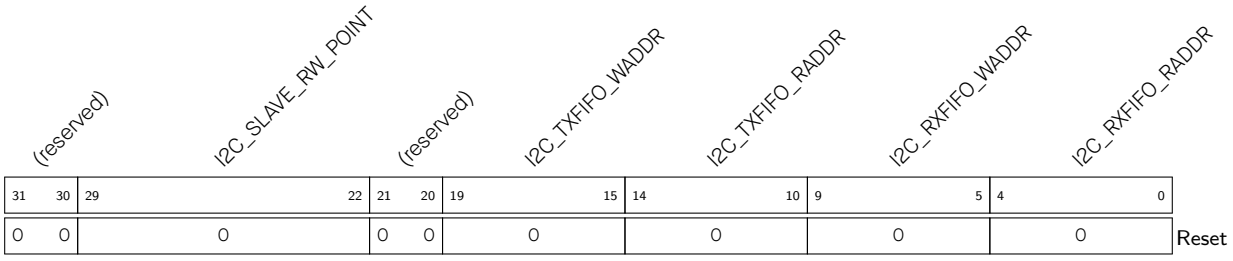
**I2C_SCL_MAIN_STATE_LAST** Represents the states of the I2C module state machine.

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: rising edge
5: High
6: Stop
7: Invalid
(RO)

Register 28.19. I2C_FIFO_ST_REG (0x0014)




**I2C_RXFIFO_RADDR** Represents the offset address of the APB reading from RX FIFO. (RO)

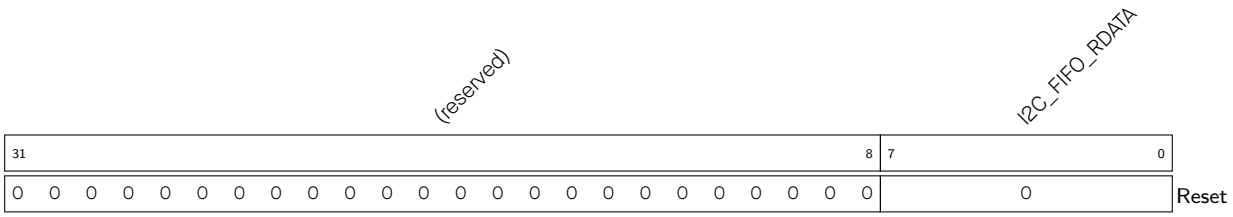
**I2C_RXFIFO_WADDR** Represents the offset address of the I2C module receiving data and writing to RX FIFO. (RO)

**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)

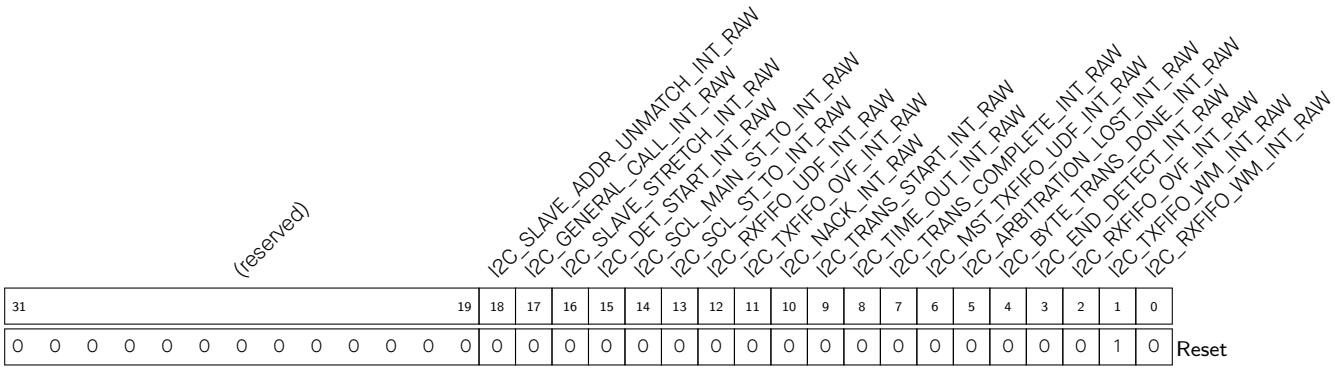
**I2C_SLAVE_RW_POINT** Represents the offset address in the I2C Slave RAM addressed by I2C Master when in I2C slave mode. (RO)

Register 28.20. I2C_DATA_REG (0x001C)




**I2C_FIFO_RDATA** Represents the value of RX FIFO read data. (RO)

Register 28.21. 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 28.21. 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)

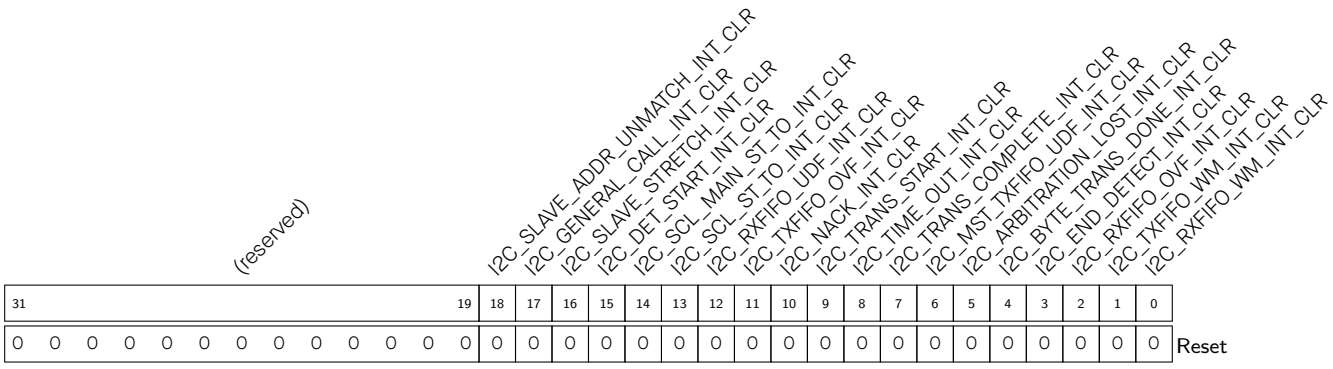
**I2C_DET_START_INT_RAW** The raw interrupt status of I2C_DET_START_INT interrupt. (R/SS/WTC)

**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 28.22. 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 28.23. I2C_INT_ENA_REG (0x0028)

| Bits | 31–19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Field | (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 | |
| | 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 |

**I2C_RXFIFO_WM_INT_ENA** Write 1 to enable I2C_RXFIFO_WM_INT interrupt. (R/W)

**I2C_TXFIFO_WM_INT_ENA** Write 1 to enable I2C_TXFIFO_WM_INT interrupt. (R/W)

**I2C_RXFIFO_OVF_INT_ENA** Write 1 to enable I2C_RXFIFO_OVF_INT interrupt. (R/W)

**I2C_END_DETECT_INT_ENA** Write 1 to enable the I2C_END_DETECT_INT interrupt. (R/W)

**I2C_BYTE_TRANS_DONE_INT_ENA** Write 1 to enable the I2C_END_DETECT_INT interrupt. (R/W)

**I2C_ARBITRATION_LOST_INT_ENA** Write 1 to enable the I2C_ARBITRATION_LOST_INT interrupt. (R/W)

**I2C_MST_TXFIFO_UDF_INT_ENA** Write 1 to enable I2C_TRANS_COMPLETE_INT interrupt. (R/W)

**I2C_TRANS_COMPLETE_INT_ENA** Write 1 to enable the I2C_TRANS_COMPLETE_INT interrupt. (R/W)

**I2C_TIME_OUT_INT_ENA** Write 1 to enable the I2C_TIME_OUT_INT interrupt. (R/W)

**I2C_TRANS_START_INT_ENA** Write 1 to enable the I2C_TRANS_START_INT interrupt. (R/W)

**I2C_NACK_INT_ENA** Write 1 to enable I2C_SLAVE_STRETCH_INT interrupt. (R/W)

**I2C_TXFIFO_OVF_INT_ENA** Write 1 to enable I2C_TXFIFO_OVF_INT interrupt. (R/W)

**I2C_RXFIFO_UDF_INT_ENA** Write 1 to enable I2C_RXFIFO_UDF_INT interrupt. (R/W)

**I2C_SCL_ST_TO_INT_ENA** Write 1 to enable I2C_SCL_ST_TO_INT interrupt. (R/W)

**I2C_SCL_MAIN_ST_TO_INT_ENA** Write 1 to enable I2C_SCL_MAIN_ST_TO_INT interrupt. (R/W)

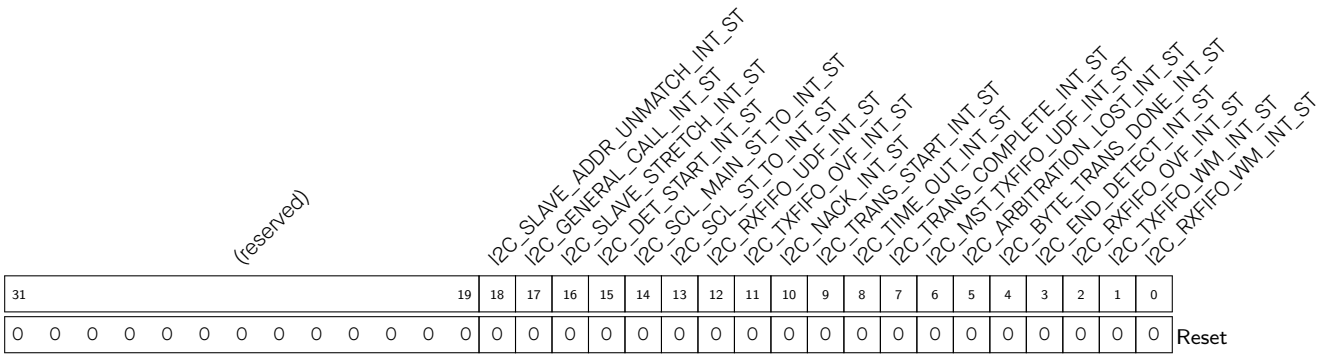
**I2C_DET_START_INT_ENA** Write 1 to enable I2C_DET_START_INT interrupt. (R/W)

**I2C_SLAVE_STRETCH_INT_ENA** Write 1 to enable I2C_SLAVE_STRETCH_INT interrupt. (R/W)

**I2C_GENERAL_CALL_INT_ENA** Write 1 to enable I2C_GENARAL_CALL_INT interrupt. (R/W)

**I2C_SLAVE_ADDR_UNMATCH_INT_ENA** Write 1 to enable I2C_SLAVE_ADDR_UNMATCH_INT interrupt. (R/W)

Register 28.24. I2C_INT_STATUS_REG (0x002C)




**I2C_RXFIFO_WM_INT_ST** The masked interrupt status status of I2C_RXFIFO_WM_INT interrupt. (RO)

**I2C_TXFIFO_WM_INT_ST** The masked interrupt status status of I2C_TXFIFO_WM_INT interrupt. (RO)

**I2C_RXFIFO_OVF_INT_ST** The masked interrupt status status of I2C_RXFIFO_OVF_INT interrupt. (RO)

**I2C_END_DETECT_INT_ST** The masked interrupt status status of the I2C_END_DETECT_INT interrupt. (RO)

**I2C_BYTE_TRANS_DONE_INT_ST** The masked interrupt status status of the I2C_END_DETECT_INT interrupt. (RO)

**I2C_ARBITRATION_LOST_INT_ST** The masked interrupt status status of the I2C_ARBITRATION_LOST_INT interrupt. (RO)

**I2C_MST_TXFIFO_UDF_INT_ST** The masked interrupt status status of I2C_TRANS_COMPLETE_INT interrupt. (RO)

**I2C_TRANS_COMPLETE_INT_ST** The masked interrupt status status of the I2C_TRANS_COMPLETE_INT interrupt. (RO)

**I2C_TIME_OUT_INT_ST** The masked interrupt status status of the I2C_TIME_OUT_INT interrupt. (RO)

**I2C_TRANS_START_INT_ST** The masked interrupt status status of the I2C_TRANS_START_INT interrupt. (RO)

**I2C_NACK_INT_ST** The masked interrupt status status of I2C_SLAVE_STRETCH_INT interrupt. (RO)

**I2C_TXFIFO_OVF_INT_ST** The masked interrupt status status of I2C_TXFIFO_OVF_INT interrupt. (RO)

Continued on the next page...

Register 28.24. I2C_INT_STATUS_REG (0x002C)

Continued from the previous page...

**I2C_RXFIFO_UDF_INT_ST** The masked interrupt status status of I2C_RXFIFO_UDF_INT interrupt. (RO)

**I2C_SCL_ST_TO_INT_ST** The masked interrupt status status of I2C_SCL_ST_TO_INT interrupt. (RO)

**I2C_SCL_MAIN_ST_TO_INT_ST** The masked interrupt status status of I2C_SCL_MAIN_ST_TO_INT interrupt. (RO)

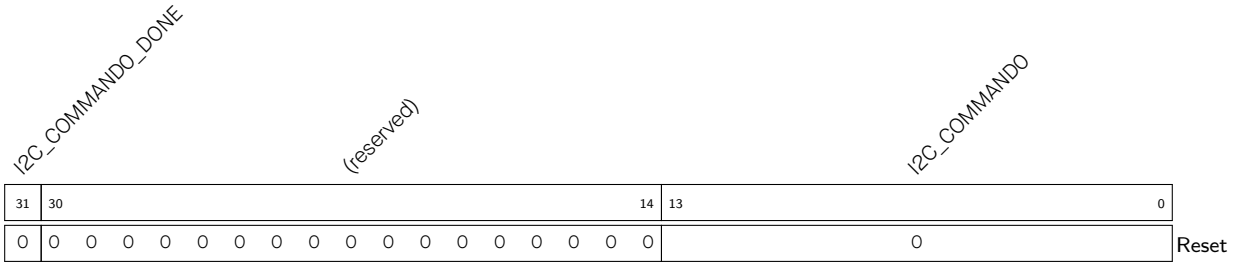
**I2C_DET_START_INT_ST** The masked interrupt status status of I2C_DET_START_INT interrupt. (RO)

**I2C_SLAVE_STRETCH_INT_ST** The masked interrupt status status of I2C_SLAVE_STRETCH_INT interrupt. (RO)

**I2C_GENERAL_CALL_INT_ST** The masked interrupt status status of I2C_GENARAL_CALL_INT interrupt. (RO)

**I2C_SLAVE_ADDR_UNMATCH_INT_ST** The masked interrupt status status of I2C_SLAVE_ADDR_UNMATCH_INT interrupt. (RO)

Register 28.25. 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 28.4-2 for more information.

(R/W)

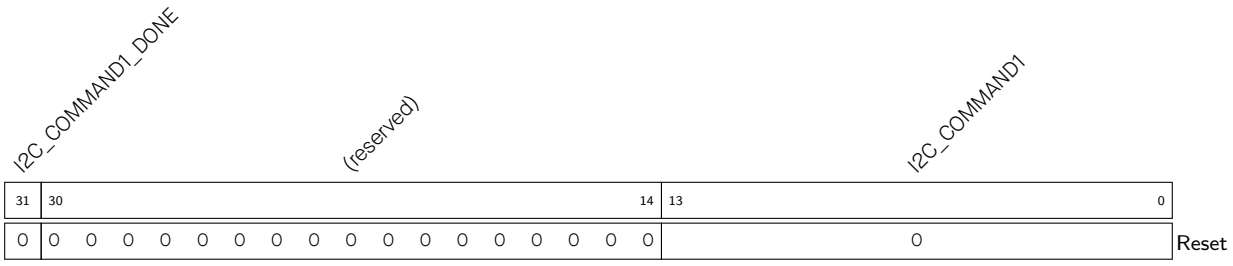
**I2C_COMMAND0_DONE** Represents whether command 0 is done in I2C Master mode.

0: Not done

1: Done

(R/W/SS)

Register 28.26. I2C_COMD1_REG (0x005C)




**I2C_COMMAND1** Configures command 1. See details in I2C_COMMAND0. (R/W)

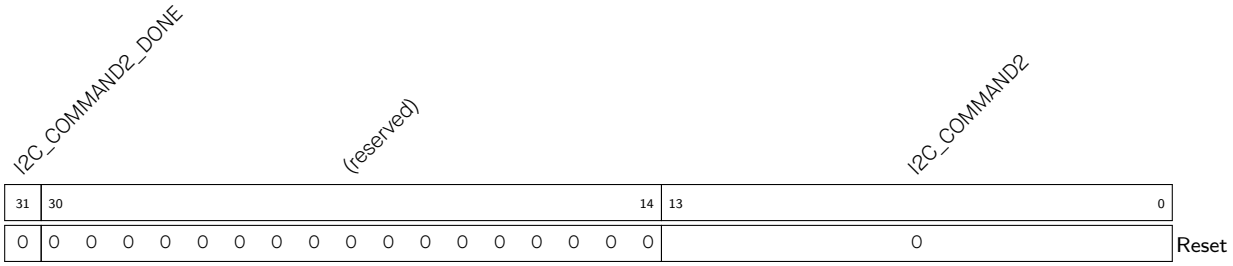
**I2C_COMMAND1_DONE** Represents whether command 1 is done in I2C Master mode.

0: Not done

1: Done

(R/W/SS)

Register 28.27. I2C_COMD2_REG (0x0060)




**I2C_COMMAND2** Configures command 2. See details in I2C_COMMAND0. (R/W)

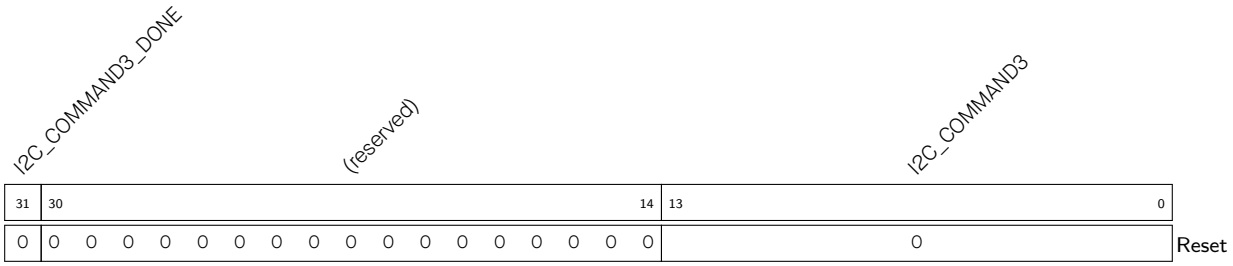
**I2C_COMMAND2_DONE** Represents whether command 2 is done in I2C Master mode.

0: Not done

1: Done

(R/W/SS)

Register 28.28. I2C_COMD3_REG (0x0064)




**I2C_COMMAND3** Configures command 3. See details in I2C_COMMAND0. (R/W)

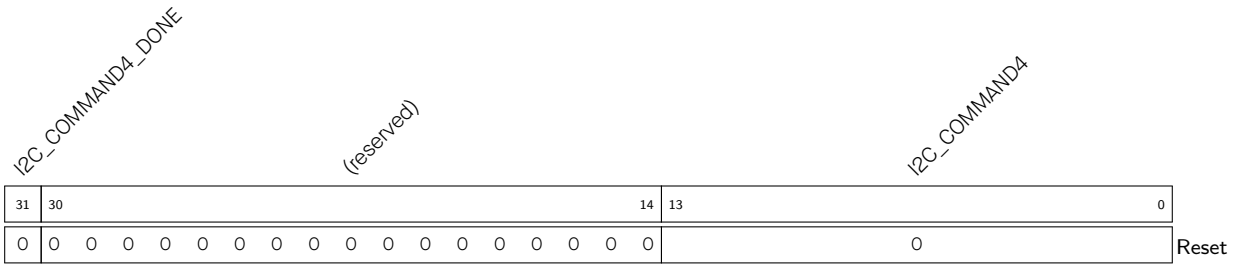
**I2C_COMMAND3_DONE** Represents whether command 3 is done in I2C Master mode.

0: Not done

1: Done

(R/W/SS)

Register 28.29. I2C_COMD4_REG (0x0068)




**I2C_COMMAND4** Configures command 4. See details in I2C_COMMAND0. (R/W)

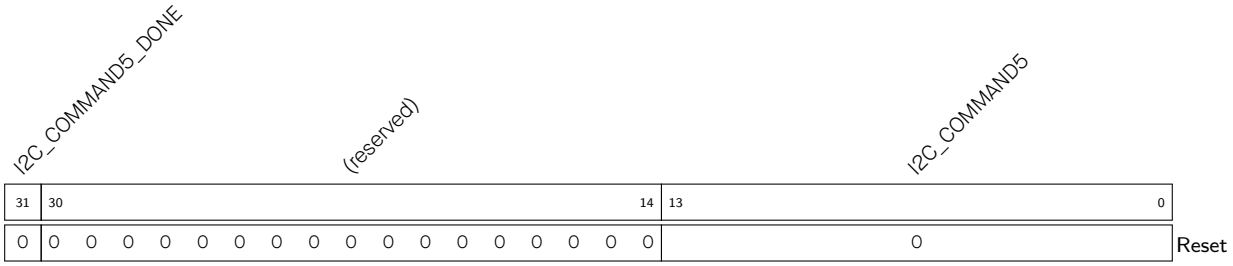
**I2C_COMMAND4_DONE** Represents whether command 4 is done in I2C Master mode.

0: Not done

1: Done

(R/W/SS)

Register 28.30. I2C_COMD5_REG (0x006C)




**I2C_COMMAND5** Configures command 5. See details in I2C_COMMAND0. (R/W)

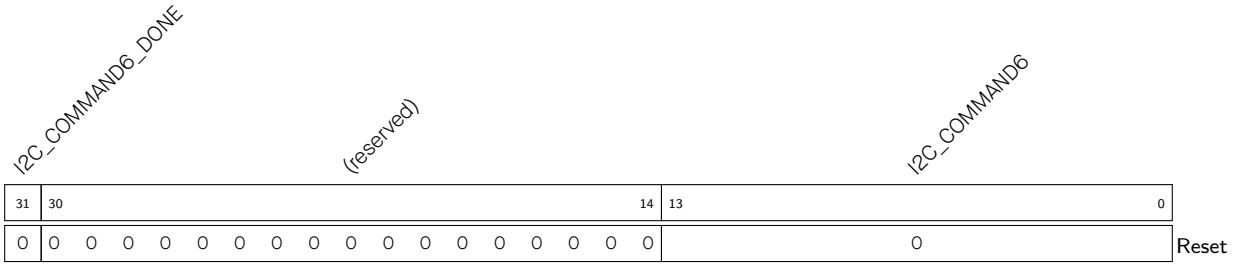
**I2C_COMMAND5_DONE** Represents whether command 5 is done in I2C Master mode.

0: Not done

1: Done

(R/W/SS)

Register 28.31. I2C_COMD6_REG (0x0070)




**I2C_COMMAND6** Configures command 6. See details in I2C_COMMAND0. (R/W)

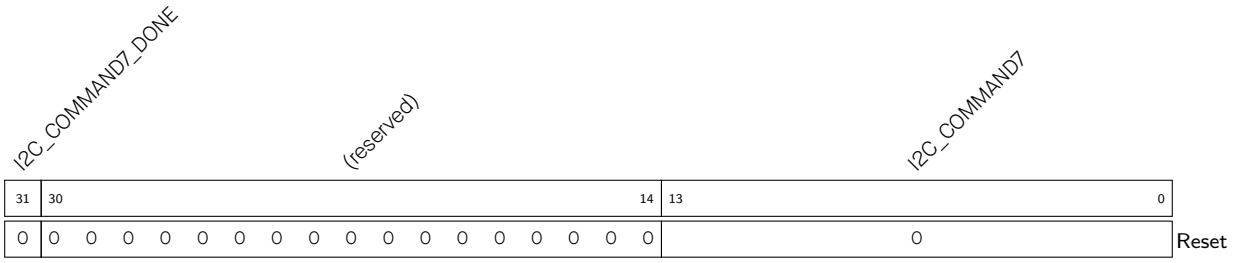
**I2C_COMMAND6_DONE** Represents whether command 6 is done in I2C Master mode.

0: Not done

1: Done

(R/W/SS)

Register 28.32. I2C_COMD7_REG (0x0074)




**I2C_COMMAND7** Configures command 7. See details in I2C_COMMAND0. (R/W)

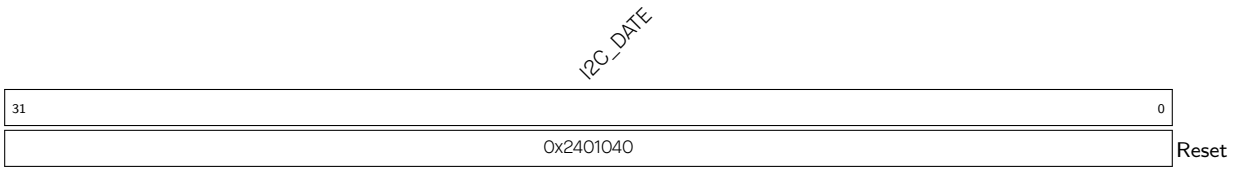
**I2C_COMMAND7_DONE** Represents whether command 7 is done in I2C Master mode.

0: Not done

1: Done

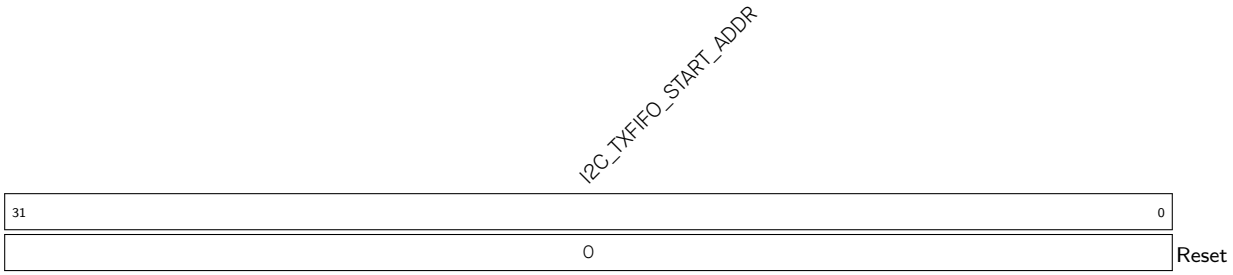
(R/W/SS)

Register 28.33. I2C_DATE_REG (0x00F8)



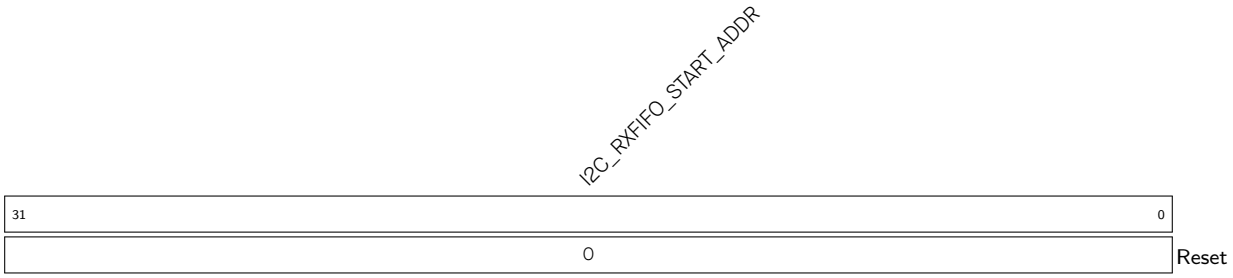

**I2C_DATE** Version control register. (R/W)

Register 28.34. I2C_TXFIFO_START_ADDR_REG (0x0100)




**I2C_TXFIFO_START_ADDR** Represents the I2C TX FIFO first address. (HRO)

Register 28.35. I2C_RXFIFO_START_ADDR_REG (0x0180)




**I2C_RXFIFO_START_ADDR** Represents the I2C RX FIFO first address. (HRO)

# Chapter 29

# I2S Controller (I2S)

## 29.1 Overview

ESP32-C61 has a built-in I2S interface, 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-C61 provides separate transmit (TX) and receive (RX) units for high performance.

## 29.2 Terminology

To better illustrate the functionality of I2S, the following terms are used in this chapter.

**Master mode** As a master, I2S drives BCK/WS signals and transmits data to or receives data from a slave.

**Slave mode** As a slave, I2S 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 $\mu$-law** A-law and $\mu$-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** 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.

## 29.3 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, up to 2 channels supported
  - PDM RX mode
    - Raw PDM data reception
- Configurable clock source with frequencies up to 160 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

- ETM feature
- Direct Memory Access
- Standard I2S interface interrupts

**Note:**
In slave mode, due to the frequency limitation of the clock source, the maximum sampling frequency of the ESP32-C61 I2S is limited by the data bit width and the number of channels. For example, sampling frequencies up to 312.5 kHz (such as 8 kHz, 16 kHz, 32 kHz, 44.1 kHz, 48 kHz, 88.2 kHz, 96 kHz, 128 kHz, and 256 kHz) are supported in 32-bit two-channel sampling. Please refer to Section 29.6 for detailed information.

## 29.4 System Architecture

Figure 29.4-1 shows the structure of the ESP32-C61 I2S 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

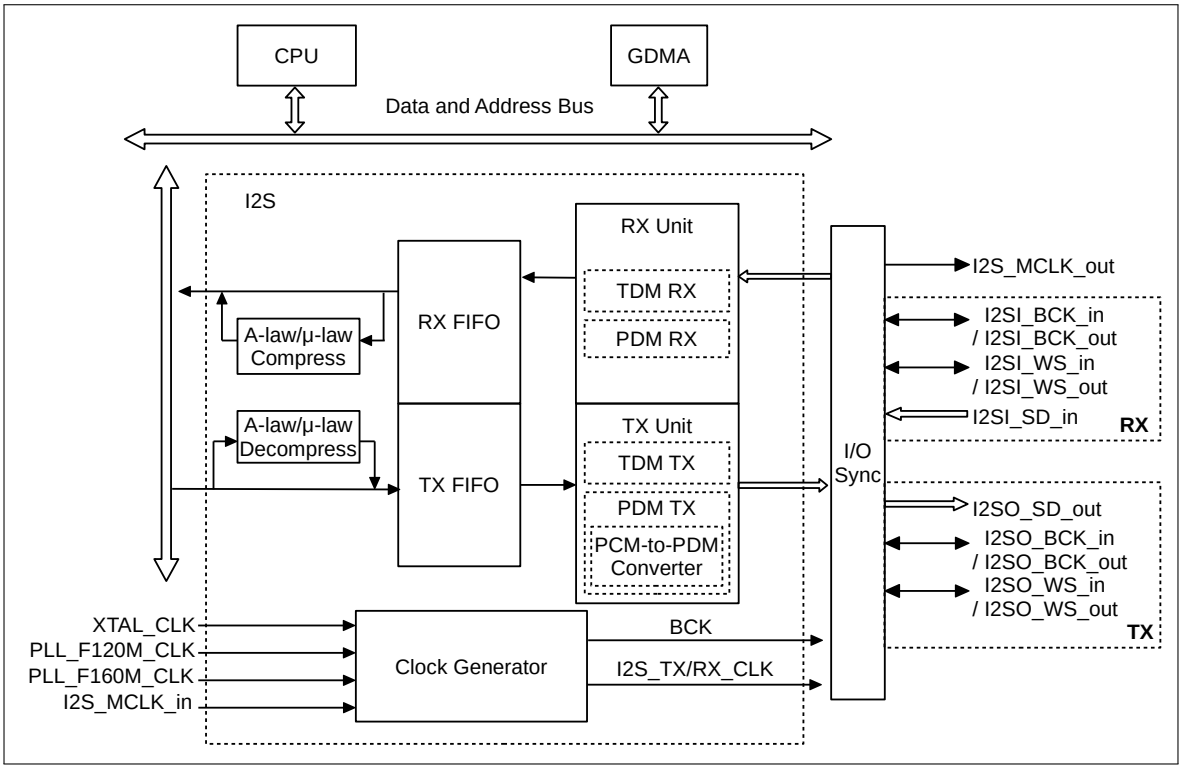



Figure 29.4-1. ESP32-C61 I2S System Diagram

The I2S module supports direct memory access (DMA) to internal memory. For more information, see Chapter *4 GDMA Controller (GDMA)*.

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 module is shown at the right part of Figure 29.4-1. The naming of these signals in RX and TX units follows the pattern of I2SA_B_C, such as I2SI_BCK_in.

- "A" indicates that the signal belongs to the I2S 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 module
  - "out": Output signal from the I2S module

Table 29.4-1 provides a detailed description of I2S signals.

**Table 29.4-1. I2S Signal Description**

| Signal | Direction | Function |
|---|---|---|
| I2SI_BCK_in | Input | In I2S slave mode, inputs BCK signal for RX unit. |
| I2SI_BCK_out | Output | In I2S master mode, outputs BCK signal for RX unit. |
| I2SI_WS_in | Input | In I2S slave mode, inputs WS signal for RX unit. |
| I2SI_WS_out | Output | In I2S master mode, outputs WS signal for RX unit. |
| I2SI_SD_in | Input | Works as the serial input data bus for I2S RX unit. |
| I2SO_SD_out | Output | Works as the serial output data bus for I2S TX unit. |
| I2SO_BCK_in | Input | In I2S slave mode, inputs BCK signal for TX unit. |
| I2SO_BCK_out | Output | In I2S master mode, outputs BCK signal for TX unit. |
| I2SO_WS_in | Input | In I2S slave mode, inputs WS signal for TX unit. |
| I2SO_WS_out | Output | In I2S master mode, outputs WS signal for TX unit. |
| I2S_MCLK_in | Input | In I2S slave mode, works as a clock source from the external master. |
| I2S_MCLK_out | Output | In I2S master mode, works as a clock source for the external slave. |
| I2SO_SD1_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 must be mapped to the chip's pins via GPIO matrix. See Chapter *7 GPIO Matrix and IO MUX*.

## 29.5 Supported Audio Standards

ESP32-C61 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:

- 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-C61 I2S will transmit data incorrectly in a mode that is neither TDM nor PDM.

### 29.5.1 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 29.5-1.

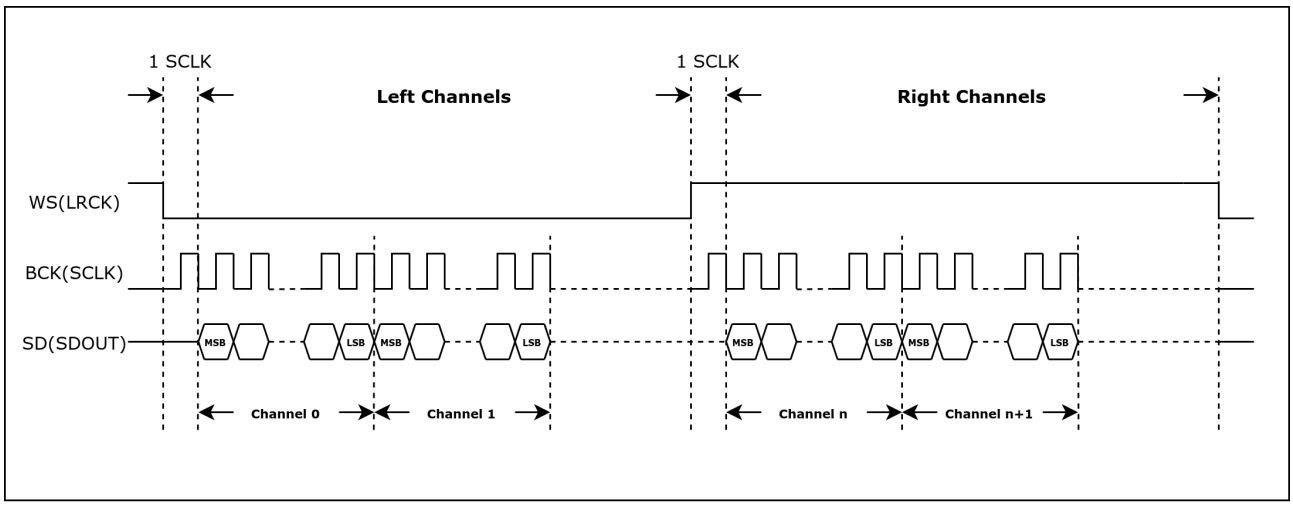



Figure 29.5-1. TDM Philips Standard Timing Diagram

## 29.5.2 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 29.5-2.

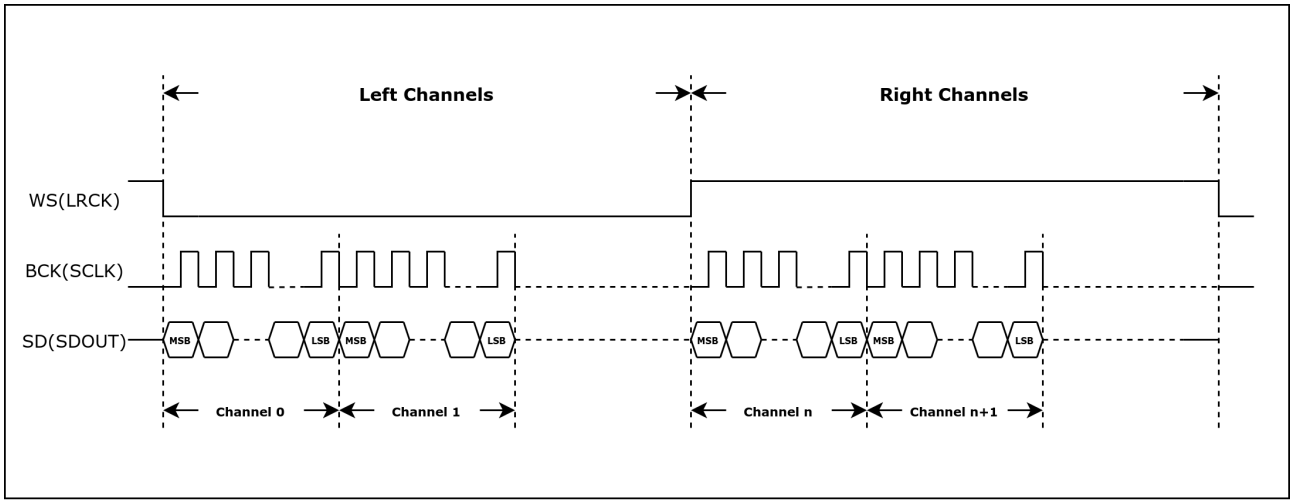



Figure 29.5-2. TDM MSB Alignment Standard Timing Diagram

## 29.5.3 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 29.5-3.

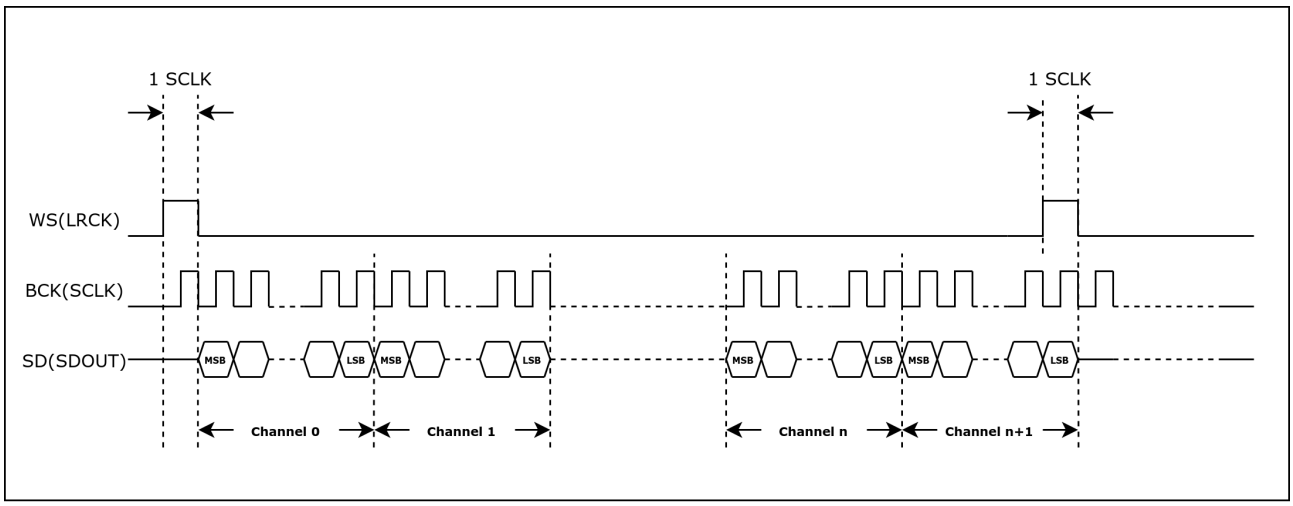



Figure 29.5-3. TDM PCM Standard Timing Diagram

### 29.5.4 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 29.5-4.

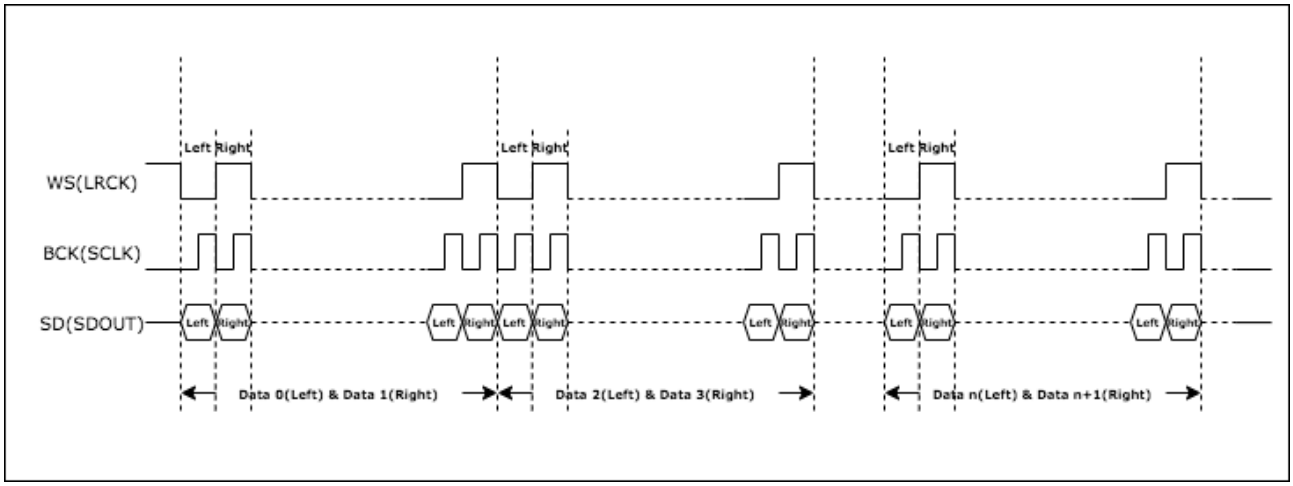



Figure 29.5-4. PDM Standard Timing Diagram

## 29.6 I2S TX/RX Clock

I2S_TX/RX_CLK is the master clock of I2S TX/RX unit, divided from:

- 40 MHz XTAL_CLK
- 120 MHz PLL_F120M_CLK
- 160 MHz PLL_F160M_CLK
- I2S_MCLK_in (external input clock)

The serial clock (BCK) of the I2S TX/RX unit is divided from I2S_TX/RX_CLK, as shown in Figure 29.6-1.

PCR_I2S_TX/RX_CLKM_SEL is used to select clock source for TX/RX unit, and PCR_I2S_TX/RX_CLKM_EN to enable or disable the clock source.

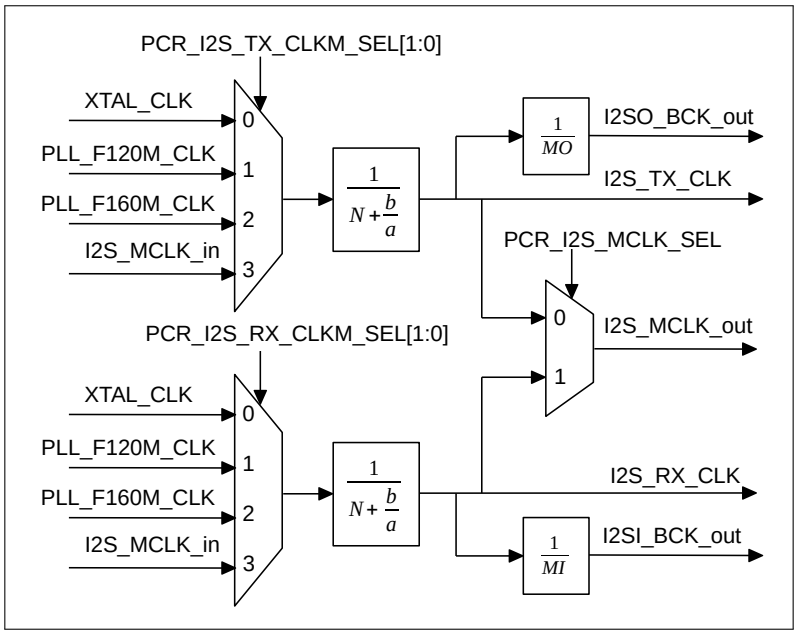



Figure 29.6-1. I2S Clock Generator

The following formula shows the relation between I2S_TX/RX_CLK frequency $f_{\text{I2S\_TX/RX\_CLK}}$ and the divider clock source frequency $f_{\text{I2S\_CLK\_S}}$:

$$f_{\text{I2S\_TX/RX\_CLK}} = \frac{f_{\text{I2S\_CLK\_S}}}{\text{N} + \frac{\text{b}}{\text{a}}}$$

N is an integer value between 2 and 256. The value of N is mapped to that of PCR_I2S_TX/RX_CLKM_DIV_NUM as follows:

- When PCR_I2S_TX/RX_CLKM_DIV_NUM = 0, N = 256;
- When PCR_I2S_TX/RX_CLKM_DIV_NUM = 1, N = 2;
- When PCR_I2S_TX/RX_CLKM_DIV_NUM has any other value, N = PCR_I2S_TX/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 in PCR_I2S_TX/RX_CLKM_DIV_X, PCR_I2S_TX/RX_CLKM_DIV_Y, PCR_I2S_TX/RX_CLKM_DIV_Z, and PCR_I2S_TX/RX_CLKM_DIV_YN1.

To configure the integer divider, clear PCR_I2S_TX/RX_CLKM_DIV_X and PCR_I2S_TX/RX_CLKM_DIV_Z, then set PCR_I2S_TX/RX_CLKM_DIV_Y to 1.

**Note:**
Using fractional divider may introduce some clock jitter.

In master TX mode, the serial clock BCK for I2S TX unit is I2SO_BCK_out divided from I2S_TX_CLK, which is:

$$f_{\text{I2SO\_BCK\_out}} = \frac{f_{\text{I2S\_TX\_CLK}}}{\text{MO}}$$

"MO" is an integer value:

$$\text{MO} = \text{I2S\_TX\_BCK\_DIV\_NUM} + 1$$

**Note:**
Note that I2S_TX_BCK_DIV_NUM must not be configured as 1.

In master RX mode, the serial clock BCK for I2S RX unit is I2SI_BCK_out divided from I2S_RX_CLK, which is:

$$f_{\text{I2SI\_BCK\_out}} = \frac{f_{\text{I2S\_RX\_CLK}}}{\text{MI}}$$

"MI" is an integer value:

$$\text{MI} = \text{I2S\_RX\_BCK\_DIV\_NUM} + 1$$

**Note:**

- I2S_RX_BCK_DIV_NUM must not be configured as 1.
- In I2S slave mode, make sure $f_{\text{I2S\_TX/RX\_CLK}}$ >= 8 * $f_{\text{BCK}}$. The I2S module can output I2S_MCLK_out as the master clock for peripherals.

## 29.7 I2S Reset

The units and FIFOs in the I2S module are reset by the following bits.

- I2S TX/RX units: Reset by I2S_TX_RESET and I2S_RX_RESET;
- I2S TX/RX FIFO: Reset by I2S_TX_FIFO_RESET and I2S_RX_FIFO_RESET.

**Note:**
The I2S module clock must be configured first before the module and FIFO are reset.

## 29.8 I2S Master/Slave Mode

The ESP32-C61 I2S 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

### 29.8.1 Master/Slave TX Mode

- I2S 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 I2S_TX_START is cleared.
- I2S 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.

### 29.8.2 Master/Slave RX Mode

- I2S 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 I2S_RX_START is cleared.
- I2S 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 I2S_RX_START is cleared.

## 29.9 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 29.13.1.

In TX mode, I2S first reads data through GDMA and transmits these data out via output signals according to the configured data mode and channel mode.

### 29.9.1 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.

#### 29.9.1.1 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 29.9-1. 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 (applies to all tables in this chapter).

#### 29.9.1.2 Endian Control of Channel Valid Data

When I2S reads data through GDMA, the data endian under various data width is controlled by I2S_TX_BIG_ENDIAN. Table 29.9-2 shows how I2S_TX_BIG_ENDIAN controls the reading of the data with different valid data widths of the channel.

Table 29.9-2. 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.

#### 29.9.1.3 A-law/$\mu$-law Compression and Decompression

ESP32-C61 I2S compresses/decompresses the valid data into 32-bit by A-law or by $\mu$-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 $\mu$-law
- 3: Compress the data using $\mu$-law

At this point, the first phase of data format control is completed.

#### 29.9.1.4 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.

#### 29.9.1.5 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 29.9-1 shows the complete process of TX data format control.

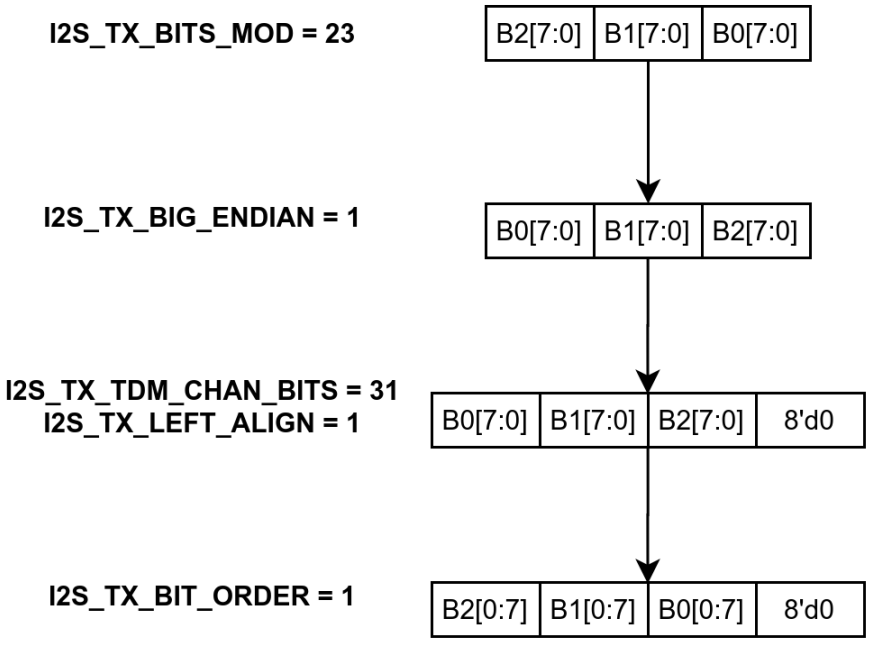



Figure 29.9-1. TX Data Format Control

### 29.9.2 Channel Mode Control

ESP32-C61 I2S 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.

#### 29.9.2.1 TDM TX Mode

In TDM TX mode, I2S 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 29.9-2.

**Note:**
Most stereo I2S codecs can be controlled by setting the I2S module into 2-channel mode under TDM standard.

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.

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.

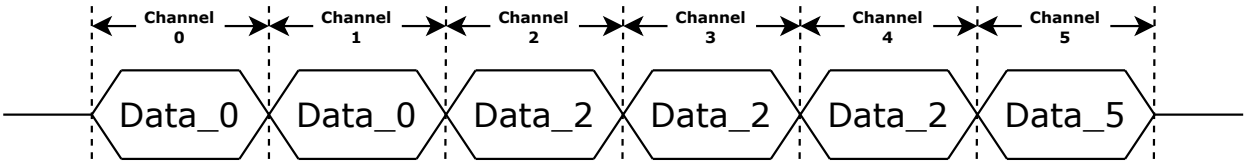
I2S_TX_HALF_SAMPLE_BITS x 2 is equal to the BCK cycles in one WS period.

**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 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 29.9-2. TDM Channel Control

#### 29.9.2.2 PDM TX Mode

In PDM TX mode, I2S supports both PDM raw data transmission and PCM-to-PDM data format conversion.

In PDM TX mode, fetching data through GDMA is controlled by I2S_TX_MONO and I2S_TX_MONO_FST_VLD. See Table 29.9-3. Configure the two bits according to the data stored in memory, be it the single-channel or dual-channel data.

Table 29.9-3. 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 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 29.6 and Figure 29.6.

When **transmitting raw PDM data**, the I2S channel mode is controlled by I2S_TX_CHAN_MOD and I2S_TX_WS_IDLE_POL. See the table below.

Table 29.9-4. I2S 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**, 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 29.9-5. I2S 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 I2SO_Data_out as the data output line.
4. 2-line DAC output format uses I2SO_Data_out and I2SO_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 I2S 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, 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 29.9.1.

Then the channel data is transmitted after channel mode control as follows.

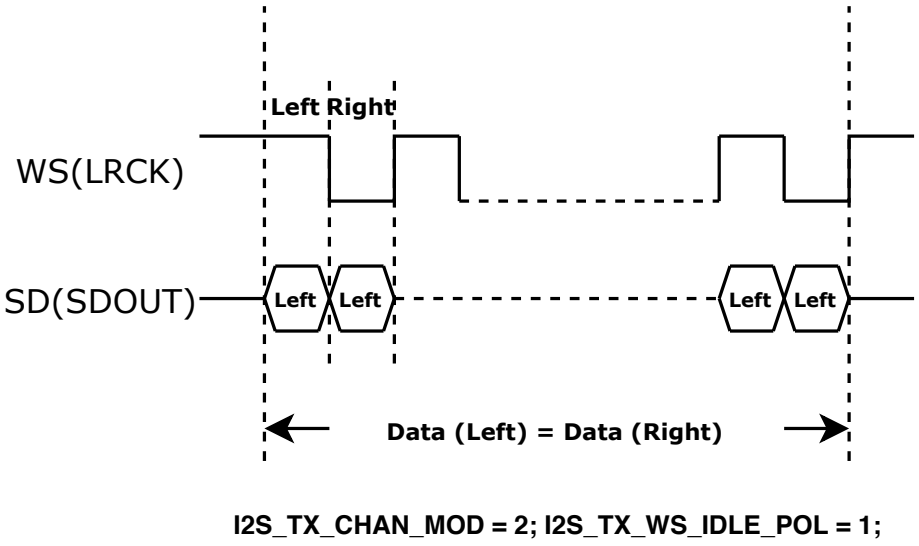



Figure 29.9-3. PDM Channel Control Example

## 29.10 Receiving Data

In RX mode, I2S 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.

### 29.10.1 Channel Mode Control

ESP32-C61 I2S 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.

#### 29.10.1.1 TDM RX Mode

In TDM RX mode, I2S 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_CHANn_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.

#### 29.10.1.2 PDM RX Mode

In PDM RX mode, I2S supports PDM raw data reception. It 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 29.6. Note, in PDM RX mode, the value of I2S_RX_HALF_SAMPLE_BITS must be same as that of I2S_RX_BITS_MOD.

### 29.10.2 Data Format Control

The data format 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.

#### 29.10.2.1 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.

#### 29.10.2.2 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 29.10-1. 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 |

#### 29.10.2.3 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.

#### 29.10.2.4 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 29.10-2. 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 |

#### 29.10.2.5 A-law/$\mu$-law Compression and Decompression

ESP32-C61 I2S compresses/decompresses the storage data in 32-bit by A-law or by $\mu$-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 $\mu$-law
- 3: Compress the data using $\mu$-law

At this point, the data format control is completed. Data then is stored into memory via GDMA.

## 29.11 Event Task Matrix Feature

ESP32-C61 I2S supports the Event Task Matrix (ETM) function, which allows I2S's ETM tasks to be triggered by any peripherals' ETM events, or I2S's ETM events to trigger any peripherals' ETM tasks. This section introduces the ETM tasks and events related to I2S. For more information, please refer to Chapter 11 Event Task Matrix (ETM).

I2S can receive the following ETM tasks:

- I2S0_TASK_START_TX: Enables I2S TX for data transfer.
- I2S0_TASK_START_RX: Enables I2S RX for data transfer.
- I2S0_TASK_STOP_TX: Stops I2S TX data transfer.
- I2S0_TASK_STOP_RX: Stops I2S RX data transfer.

I2S can generate the following ETM events:

- I2S0_EVT_TX_DONE: Indicates that I2S TX has completed data transmission. Triggered when all data in the TX FIFO has been sent.
- I2S0_EVT_RX_DONE: Can be triggered in different ways depending on the configured value of I2S_RX_STOP_MODE:
  - 0: Will not be triggered;
  - 1: When triggered, indicates that the number of bytes received by I2S RX is greater than the receive length value configured by I2S_RX_EOF_NUM_REG;
  - 2: When triggered, indicates that the GDMA RX FIFO is full.
- I2S0_EVT_X_WORDS_SENT: Indicates that the word number sent by I2S TX is equal to or larger than the value set by I2S_ETM_TX_SEND_WORD_NUM.
- I2S0_EVT_X_WORDS_RECEIVED: Indicates that the word number received by I2S RX is equal to or larger than the value set by I2S_ETM_RX_RECEIVE_WORD_NUM.

In practical applications, I2S's ETM events can trigger its own ETM tasks. For example, the I2S0_EVT_X_WORDS_SENT event can trigger the I2S0_TASK_STOP_TX task, and in this way stop the I2S operation through ETM.

## 29.12 I2S Interrupts

ESP32-C61's I2S can generate the I2S_INTR interrupt signal that will be sent to the Interrupt Matrix. There are several internal interrupt sources from I2S that can generate the above interrupt signal(s) as follows:

- I2S_TX_HUNG_INT: Triggered when data transmission is timed out. For example, if the I2S 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_RX_HUNG_INT: Triggered when the data reception is timed out. For example, if the I2S 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_TX_DONE_INT: Triggered when sending data is complete, i.e., all data in I2S TX FIFO has been sent.
- I2S_RX_DONE_INT: Triggered when the data receiving is completed.

**Note:**
For definitions of *interrupt*, *interrupt signal*, *interrupt source*, and their correlations, please refer to Chapter 10 *Interrupt Matrix* > Section 10.2 *Interrupt Terminology in ESP32-C61*.

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 29.14 *Register Summary*.

## 29.13 Software Configuration Process

### 29.13.1 Configure I2S as TX Mode

Follow the steps below to configure I2S as TX mode via software:

1. Configure the clock as described in Section 29.6.
2. Configure signal pins according to Table 29.4-1.
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 29.9, and then set I2S_TX_UPDATE.
5. Reset TX unit and TX FIFO as described in Section 29.7.
6. Enable corresponding interrupts. See Section 29.12.
7. Configure DMA outlink.
8. Set I2S_TX_STOP_EN if needed. For more information, please refer to Section 29.8.1.
9. Start transmitting data:
   - In master mode, wait till I2S slave gets ready, then set I2S_TX_START to start transmitting data;
   - In slave mode, set I2S_TX_START. When the I2S master supplies BCK and WS signals, I2S 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.

### 29.13.2 Configure I2S as RX Mode

Follow the steps below to configure I2S as RX mode via software:

1. Configure the clock as described in Section 29.6.
2. Configure signal pins according to Table 29.4-1.
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 29.10, and then set I2S_RX_UPDATE.
5. Reset the RX unit and its FIFO according to Section 29.7.
6. Enable the corresponding interrupts. See Section 29.12.
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.

## 29.14 Register Summary

The addresses in this section are relative to the I2S Controller base address provided in Table 5.3-2 in Chapter 5 *System and Memory*.

The abbreviations given in Column **Access** are explained in Section *Access Types for Registers*.

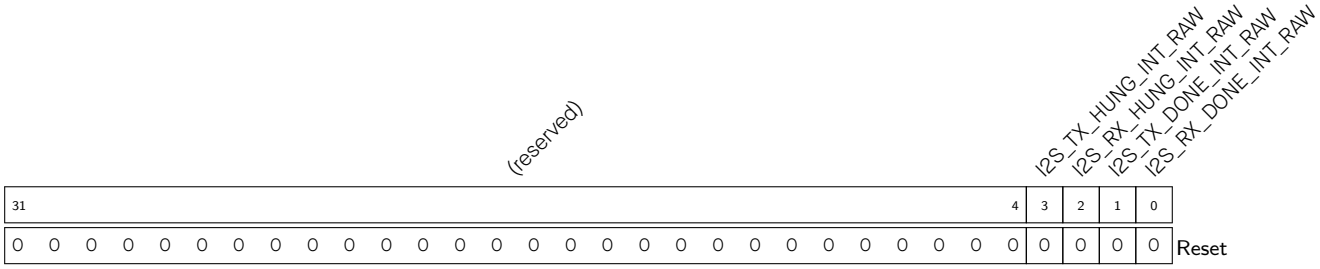
| Name | Description | Address | Access |
|---|---|---|---|
| **Interrupt registers** | | | |
| I2S_INT_RAW_REG | I2S interrupt raw register | 0x000C | R/SS/WTC |
| I2S_INT_ST_REG | I2S interrupt status register | 0x0010 | RO |
| I2S_INT_ENA_REG | I2S interrupt enable register | 0x0014 | R/W |
| I2S_INT_CLR_REG | 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_TX_PCM2PDM_CONF_REG | I2S TX PCM-to-PDM configuration register | 0x0040 | R/W |
| I2S_TX_PCM2PDM_CONF1_REG | I2S TX PCM-to-PDM configuration register 1 | 0x0044 | 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_TDM_CTRL_REG | I2S TX TDM mode control register | 0x0054 | R/W |
| **RX timing register** | | | |
| I2S_RX_TIMING_REG | I2S RX timing control register | 0x0058 | R/W |
| **TX timing register** | | | |
| I2S_TX_TIMING_REG | I2S TX timing control register | 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 configure register | 0x0070 | R/W |
| **Version register** | | | |
| I2S_DATE_REG | Version control register | 0x0080 | R/W |

## 29.15 Registers

The addresses in this section are relative to the I2S Controller base address provided in Table 5.3-2 in Chapter 5 *System and Memory*.

For how to program reserved fields, please refer to Section *Programming Reserved Register Field*.

**Register 29.1. I2S_INT_RAW_REG (0x000C)**



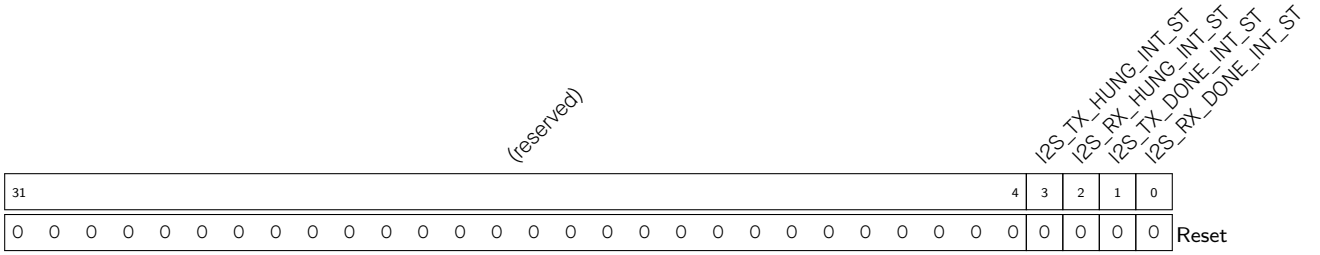

**I2S_RX_DONE_INT_RAW** The raw interrupt status of the I2S_RX_DONE_INT interrupt. (R/SS/WTC)

**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 29.2. I2S_INT_ST_REG (0x0010)**



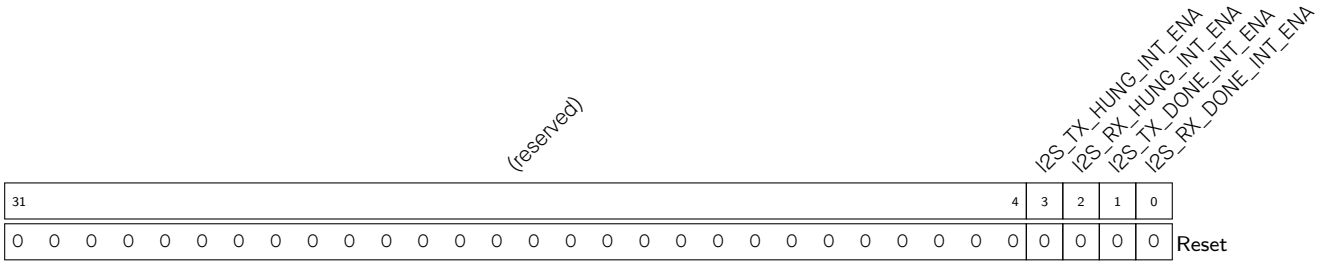

**I2S_RX_DONE_INT_ST** The masked interrupt status of the I2S_RX_DONE_INT interrupt. (RO)

**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 29.3. I2S_INT_ENA_REG (0x0014)



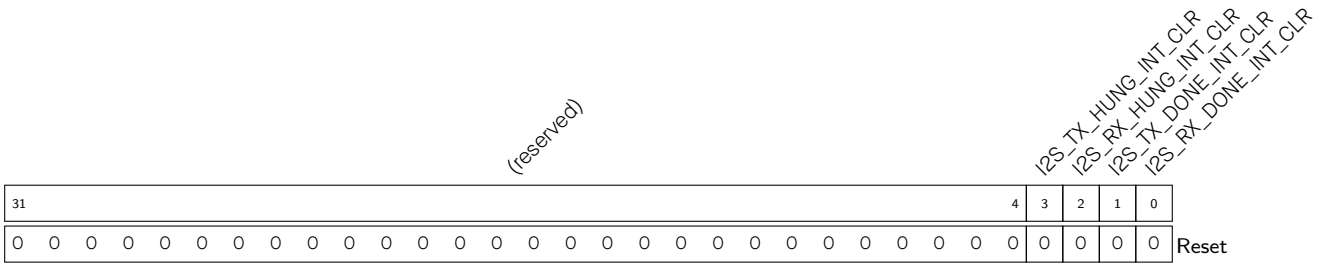

**I2S_RX_DONE_INT_ENA** Write 1 to enable the I2S_RX_DONE_INT interrupt. (R/W)

**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 29.4. I2S_INT_CLR_REG (0x0018)



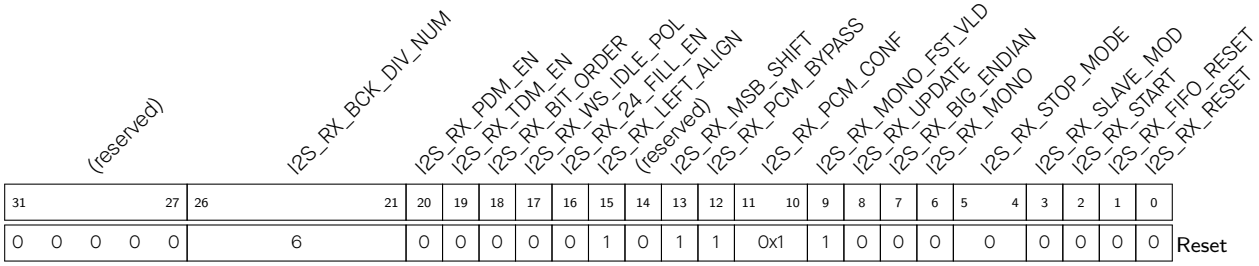

**I2S_RX_DONE_INT_CLR** Write 1 to clear the I2S_RX_DONE_INT interrupt. (WT)

**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 29.5. 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 29.5. 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): $\mu$-law decompress
3 (ltou): $\mu$-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 29.5. 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.

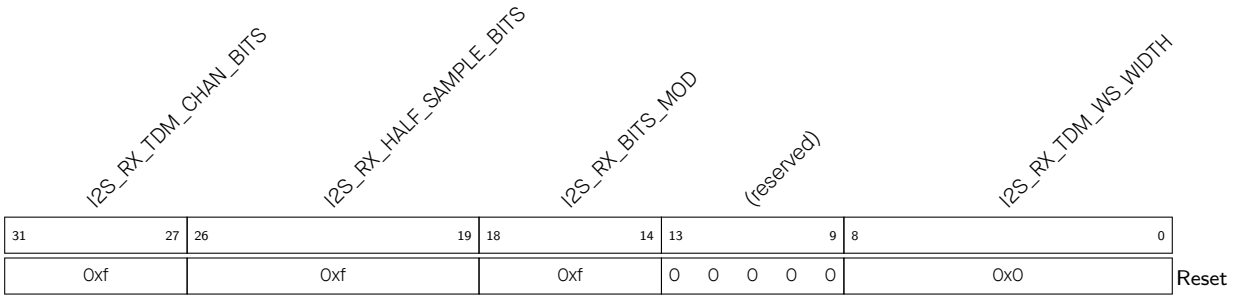
0: Disable

1: Enable

(R/W)

**I2S_RX_BCK_DIV_NUM** Configures the divider of BCK in RX mode. Note that this divider must not be configured to 1. (R/W)

Register 29.6. I2S_RX_CONF1_REG (0x0028)




**I2S_RX_TDM_WS_WIDTH** Configures the width of I2SI_WS_out (WS default level) at idle level in TDM mode. Width of I2SI_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

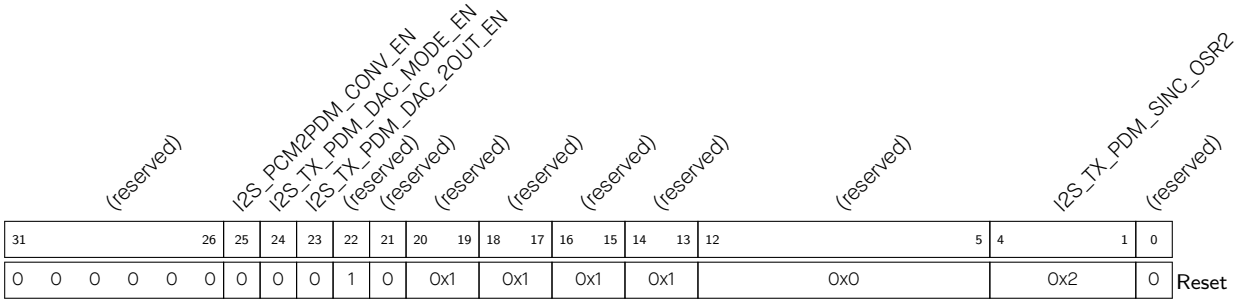
Other values are invalid.

(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 29.7. I2S_TX_PCM2PDM_CONF_REG (0x0040)**




**I2S_TX_PDM_SINC_OSR2** Configures I2S TX PDM OSR value. (R/W)

**I2S_TX_PDM_DAC_2OUT_EN** Configures DAC output 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 I2S TX PCM-to-PDM conversion.

0: Disable

1: Enable

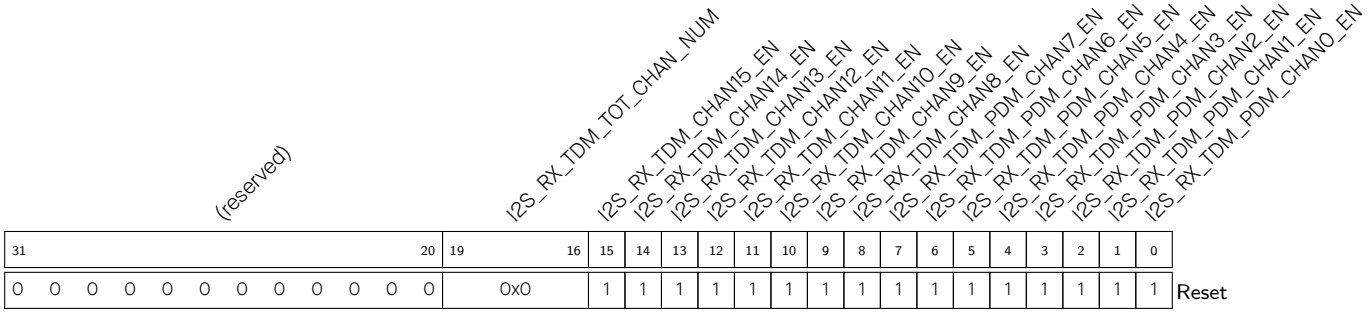
(R/W)

**Register 29.8. I2S_TX_PCM2PDM_CONF1_REG (0x0044)**




**I2S_TX_PDM_FS** Configures I2S PDM TX upsampling parameter. (R/W)

Register 29.9. 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*.

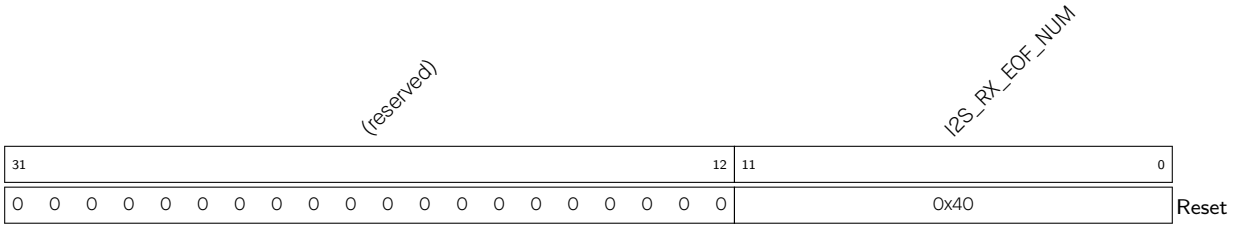
0: Disable. Channel *n* only inputs 0

1: Enable

(R/W)

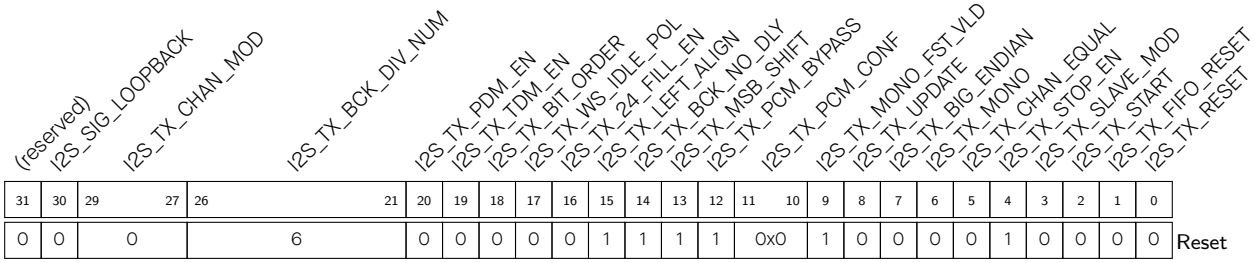
**I2S_RX_TDM_TOT_CHAN_NUM** Configures the total number of channels in use in I2S RX TDM mode. Total channel number in use = I2S_RX_TDM_TOT_CHAN_NUM + 1. (R/W)

Register 29.10. 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, a GDMA_IN_SUC_EOF_CH*n*_INT interrupt is triggered in the configured GDMA RX channel. (R/W)

Register 29.11. 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 29.11. 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): $\mu$-law decompress
3 (ltou): $\mu$-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 I2SO_BCK_in
1: Rising and falling edges are constructed by dividing the clock of I2S 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 29.11. 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)

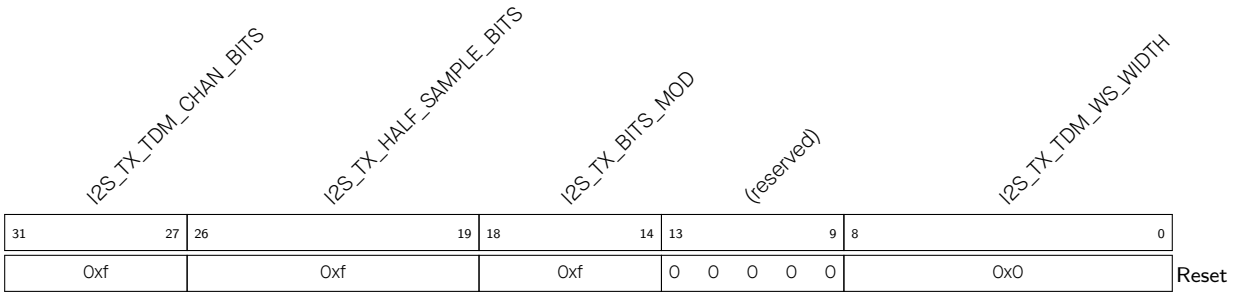
**I2S_TX_PDM_EN** Configures whether to enable I2S PDM TX mode.
0: Disable
1: Enable
(R/W)

**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 29.9-4. (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 29.12. I2S_TX_CONF1_REG (0x002C)**




**I2S_TX_TDM_WS_WIDTH** Configures the width of I2SO_WS_out (WS default level) at idle level in TDM mode. The width of I2SO_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

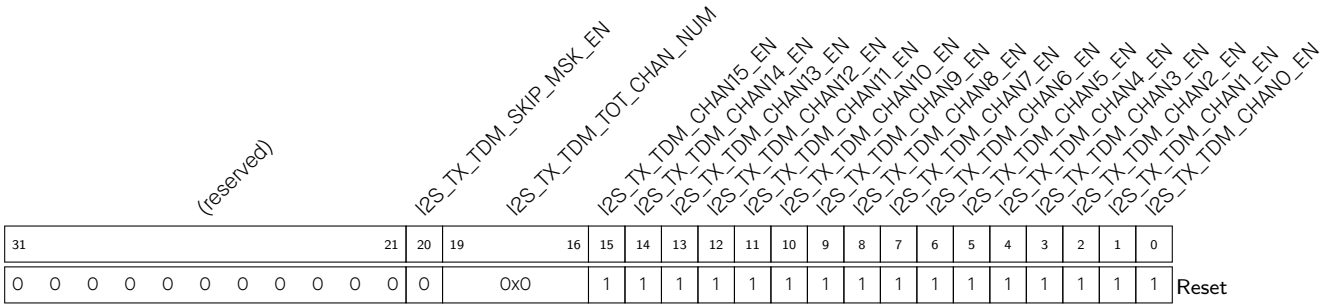
Other values are invalid.

(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 29.13. 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 29.9.2.1

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)

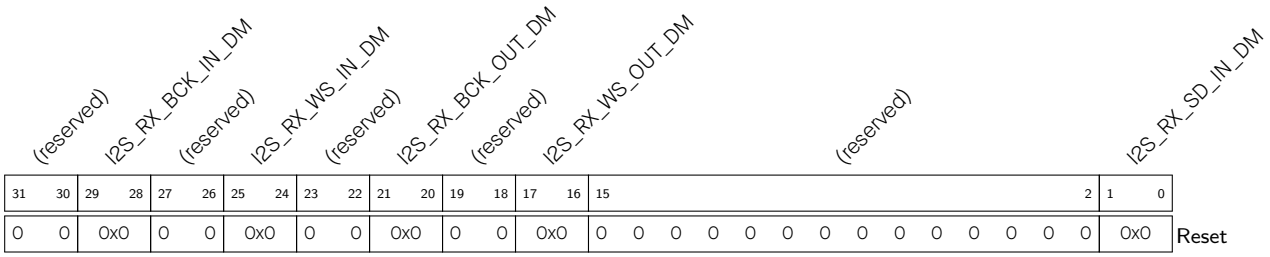
**I2S_TX_TDM_SKIP_MSK_EN** Configures the data to be sent in GDMA TX buffer.

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 29.14. 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)

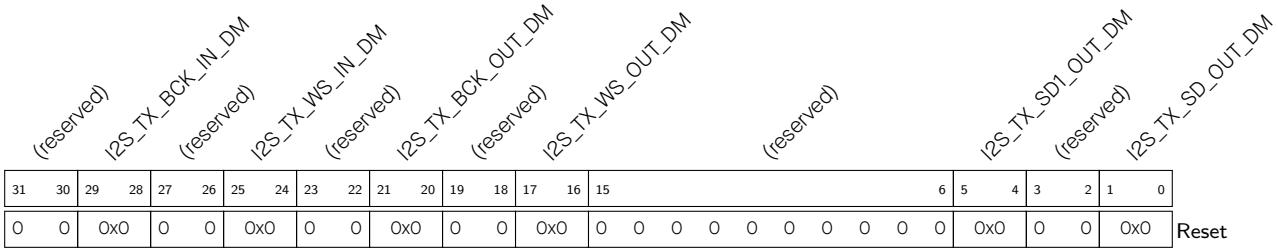
**I2S_RX_WS_OUT_DM** Configures the delay mode of I2S RX WS output signal. For detailed configuration values, please refer to I2S_RX_SD_IN_DM. (R/W)

**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 29.15. 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** Configures the delay mode of I2S TX SD1 output signal. For detailed configuration values, please refer to I2S_TX_SD_OUT_DM. (R/W)

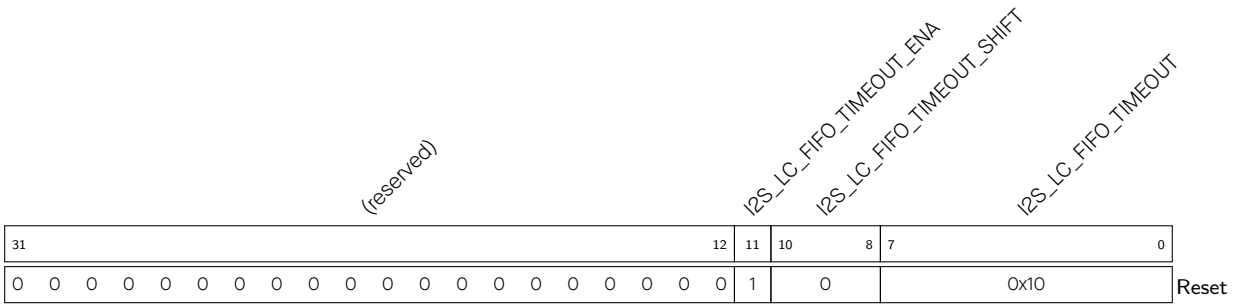
**I2S_TX_WS_OUT_DM** Configures the delay mode of I2S TX WS output signal. For detailed configuration values, please refer to I2S_TX_SD_OUT_DM. (R/W)

**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 29.16. 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 the 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)

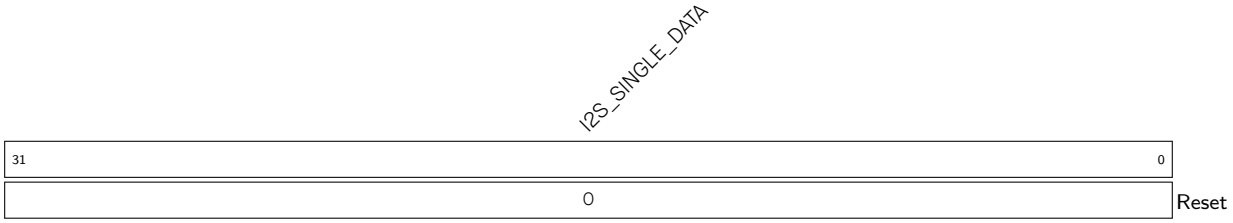
**I2S_LC_FIFO_TIMEOUT_ENA** Configures whether to enable FIFO timeout.

0: Disable

1: Enable

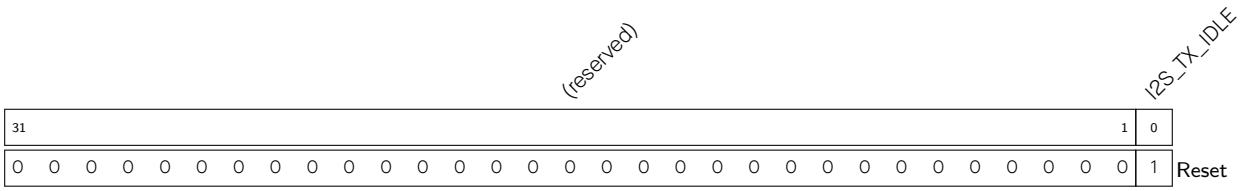
(R/W)

Register 29.17. I2S_CONF_SIGLE_DATA_REG (0x0068)




**I2S_SINGLE_DATA** Configures constant channel data to be sent out. (R/W)

Register 29.18. I2S_STATE_REG (0x006C)



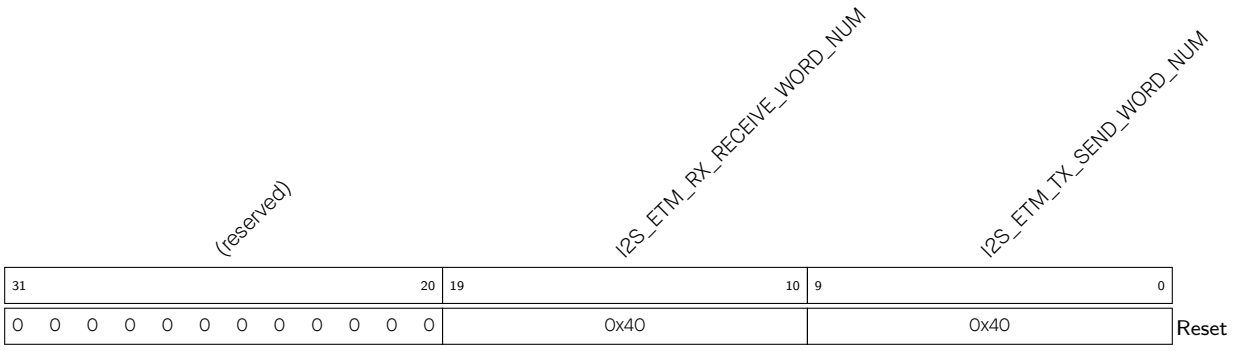

**I2S_TX_IDLE** Represents the TX unit state.

0: I2S TX unit is working

1: I2S TX unit is in idle state

(RO)

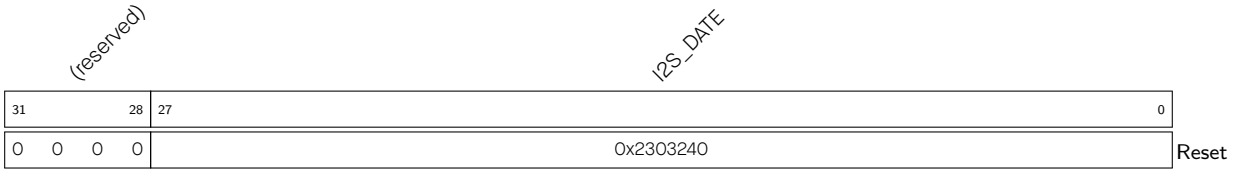
Register 29.19. I2S_ETM_CONF_REG (0x0070)




**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 29.20. I2S_DATE_REG (0x0080)




**I2S_DATE** Version control register. (R/W)

# Chapter 30

# USB Serial/JTAG Controller

ESP32-C61 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.

## 30.1 Overview

While programming and debugging an ESP32-C61 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-C61 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-C61.

## 30.2 Feature List

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

**Note:**
The SDIO Slave controller can be used together with the USB Serial/JTAG controller in single SPI mode, but not in quad

SPI mode.

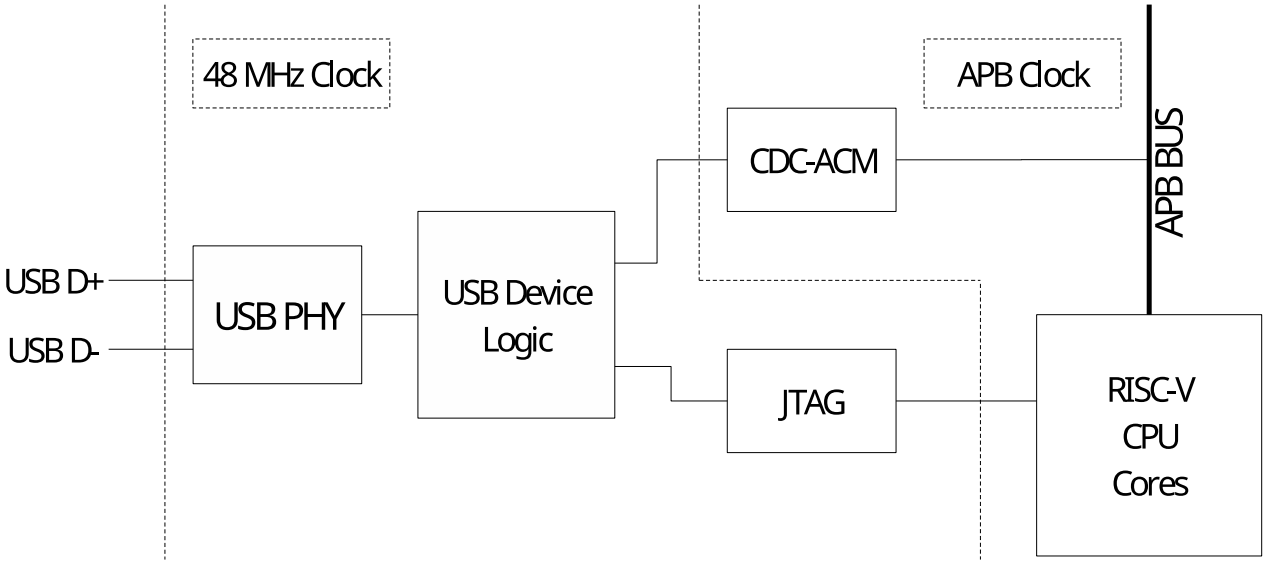



Figure 30.2-1. USB Serial/JTAG High Level Diagram

As shown in Figure 30.2-1, 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 30.2-2 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.

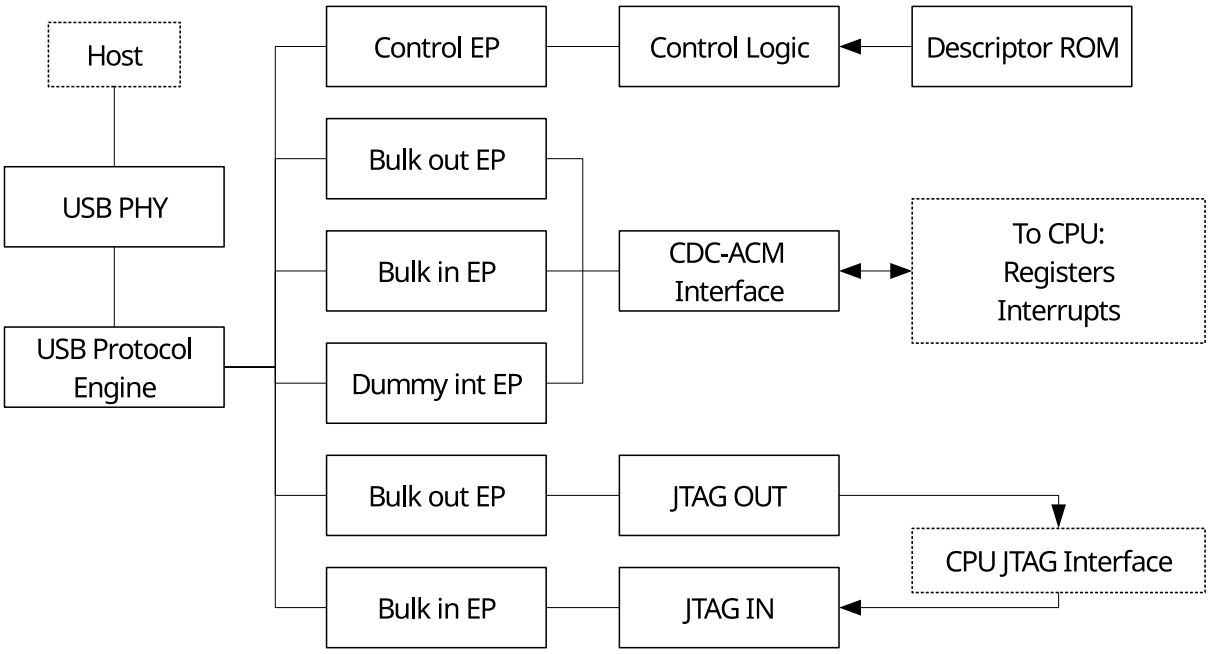



Figure 30.2-2. USB Serial/JTAG Block Diagram

## 30.3 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.

### 30.3.1 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 30.3-1. 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 30.3-2 |

Aside from general-purpose communication, the CDC-ACM interface can also be used to reset ESP32-C61 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 30.3-2. 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-C61 |
| 1 | 1 | No action |

Note that if the download mode flag is set when ESP32-C61 is reset, ESP32-C61 will reboot into download mode. When this flag is cleared and the chip is reset, ESP32-C61 will boot from flash. For specific sequences, please refer to Section 30.5. All these functions can also be disabled by programming various eFuses. Please refer to Chapter 6 *eFuse Controller (EFUSE)* for more details.

### 30.3.2 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-C61. 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.

Generally, after sending data, the attached host will block until the data is actually read by firmware in one of the described methods. However, it is possible that a given firmware program does not read data at all. This situation generally is not expected by host programs which are written to talk to a discrete USB-serial converter; the program e.g. stops executing until the data is read. The ESP32-C61 USB-serial-JTAG adapter can simulate the behavior of a discrete USB-serial-JTAG converter: by writing a timeout to USB_SERIAL_JTAG_SERIAL_TIMEOUT_MAX and writing an 1 to USB_SERIAL_JTAG_SERIAL_TIMEOUT_EN, any unread data will be automatically flushed after the given timeout.

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-C61. 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.

### 30.3.3 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-C61.

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-C61. 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-C61 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 30.3-3. 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-C61.
- 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-C61.
- 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.

### 30.3.4 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.

### 30.3.5 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.

### 30.3.6 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 30.3-4. 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 30.3-5). 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-C61 are as follows. Note that all 16-bit values are little-endian.

Table 30.3-5. 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 |

## 30.4 Interrupts

ESP32-C61'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.

## 30.5 Programming Procedures

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 30.5-1. 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 30.5-2. 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 |

## 30.6 Register Summary

The addresses in this section are relative to USB Serial/JTAG controller base address provided in Table 5.3-2 in Chapter 5 *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 |
| USB_SERIAL_JTAG_SERIAL_EP_TIMEOUT0_REG | USB UART out endpoint timeout configuration | 0x006C | varies |
| USB_SERIAL_JTAG_SERIAL_EP_TIMEOUT1_REG | USB UART out endpoint timeout configuration | 0x0070 | R/W |
| **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 |

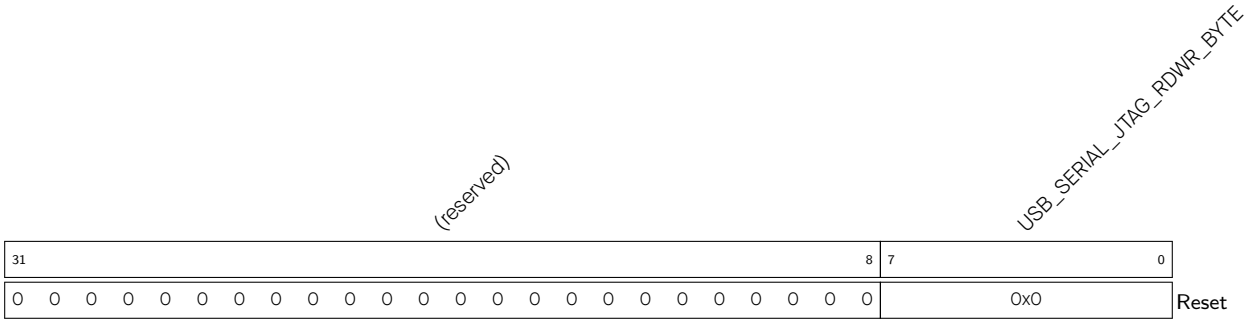
| Name | Description | Address | Access |
|---|---|---|---|
| USB_SERIAL_JTAG_OUT_EP1_ST_REG | CDC-ACM OUT endpoint status information | 0x003C | RO |
| USB_SERIAL_JTAG_OUT_EP2_ST_REG | JTAG OUT endpoint status information | 0x0040 | RO |
| USB_SERIAL_JTAG_SET_LINE_CODE_W0_REG | W0 of SET_LINE_CODING 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 | 0x0080 | R/W |

## 30.7 Registers

The addresses in this section are relative to USB Serial/JTAG controller base address provided in Table 5.3-2 in Chapter 5 *System and Memory*.

For how to program reserved fields, please refer to Section Programming Reserved Register Field.

**Register 30.1. USB_SERIAL_JTAG_EP1_REG (0x0000)**



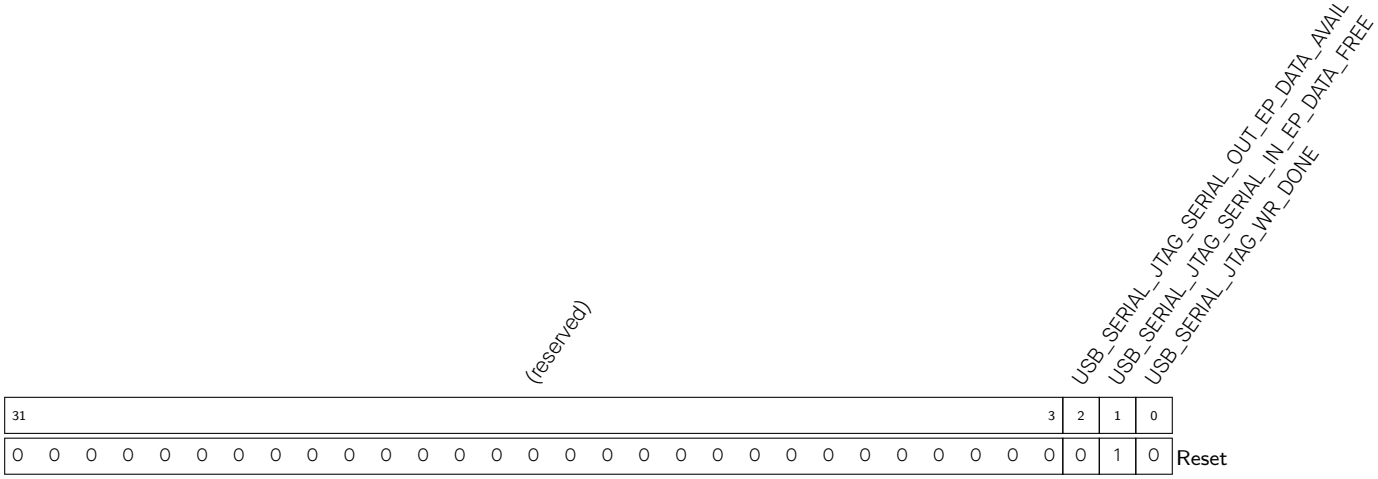

**USB_SERIAL_JTAG_RDWR_BYTE** A write to this register pushes the written data into the CDC TX FIFO; a read from this register pops a byte from the CDC RX FIFO and returns it.

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 30.2. 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)

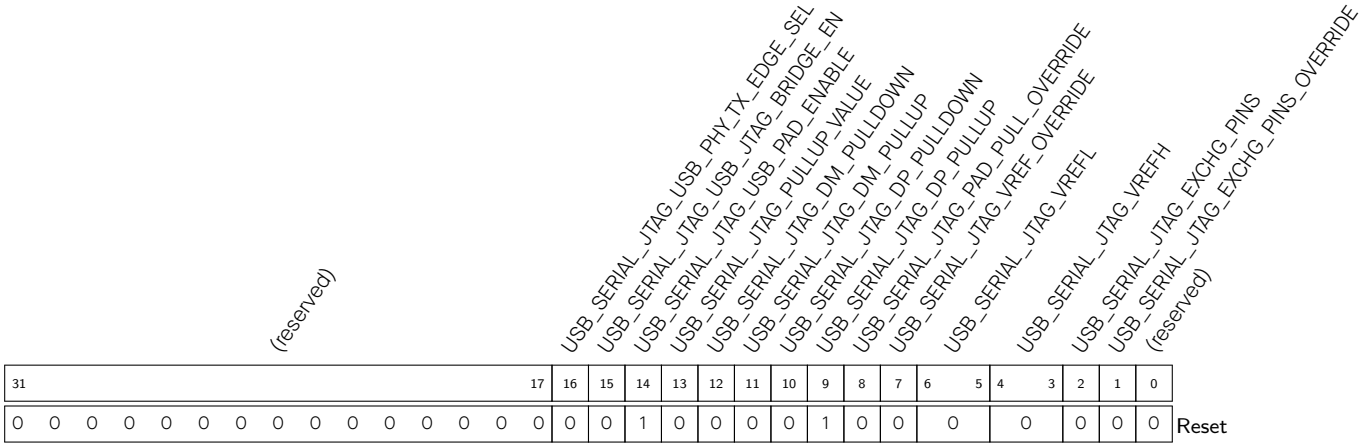
**USB_SERIAL_JTAG_SERIAL_OUT_EP_DATA_AVAIL** Represents whether there is data in CDC-ACM RX FIFO.

0: There is no data in CDC-ACM RX FIFO

1: There is data in CDC-ACM RX FIFO

(RO)

Register 30.3. USB_SERIAL_JTAG_CONF0_REG (0x0018)




**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 30.3. 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_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_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)

Continued on the next page...

Register 30.3. USB_SERIAL_JTAG_CONF0_REG (0x0018)

Continued from the previous page...

**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)

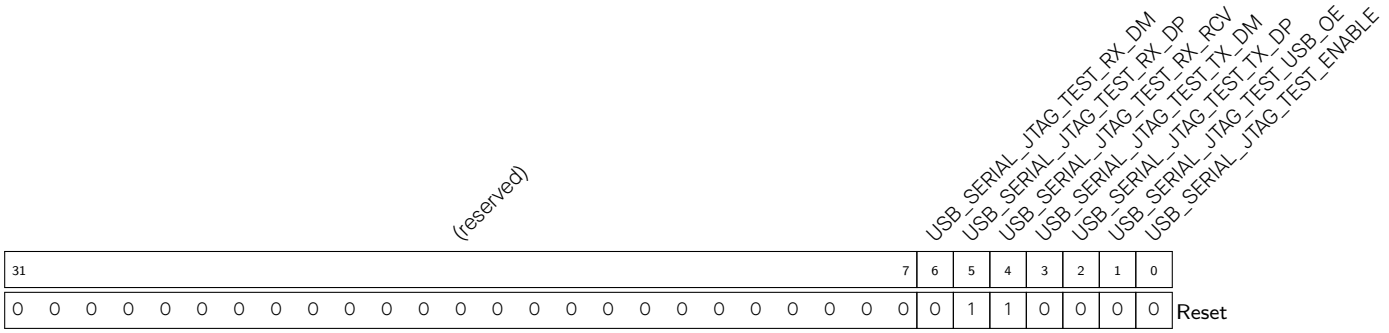
**USB_SERIAL_JTAG_USB_PHY_TX_EDGE_SEL** Configures the clock edge at which DP and DM signals are transmitted to the USB PHY.

0: Transmit on the falling edge of the clock

1: Transmit on the rising edge of the clock

(R/W)

Register 30.4. 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+

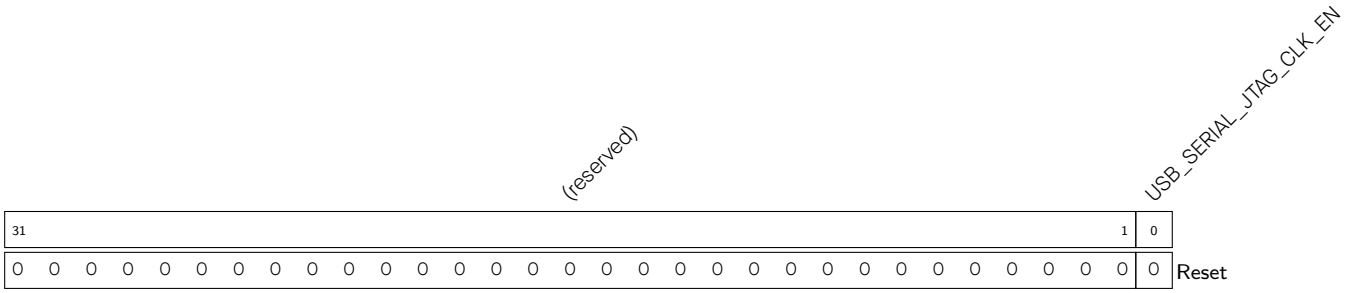
1: USB D+ voltage is higher than USB D-

(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 30.5. USB_SERIAL_JTAG_MISC_CONF_REG (0x0044)



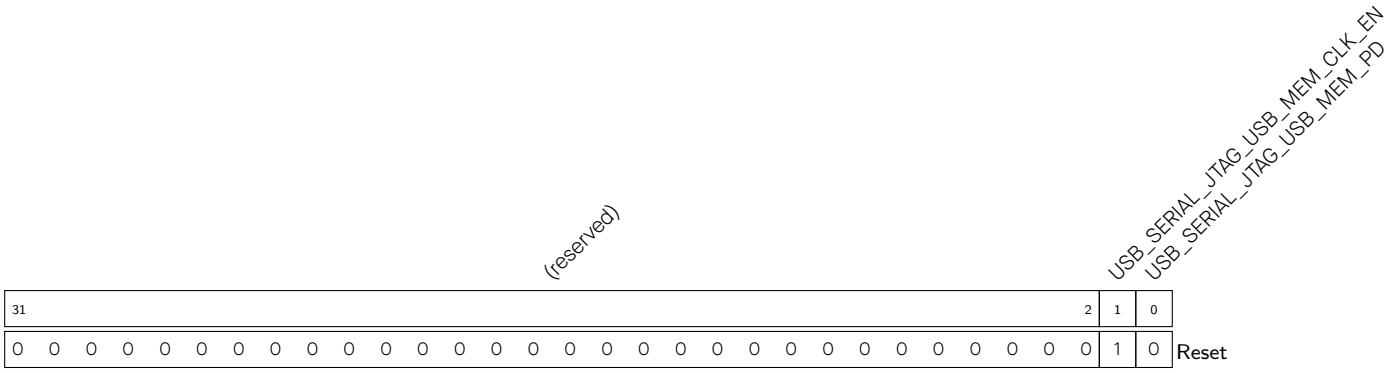

**USB_SERIAL_JTAG_CLK_EN** Configures whether to force clock on for register.

0: Support clock only when application writes registers

1: Force clock on for register

(R/W)

Register 30.6. 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)

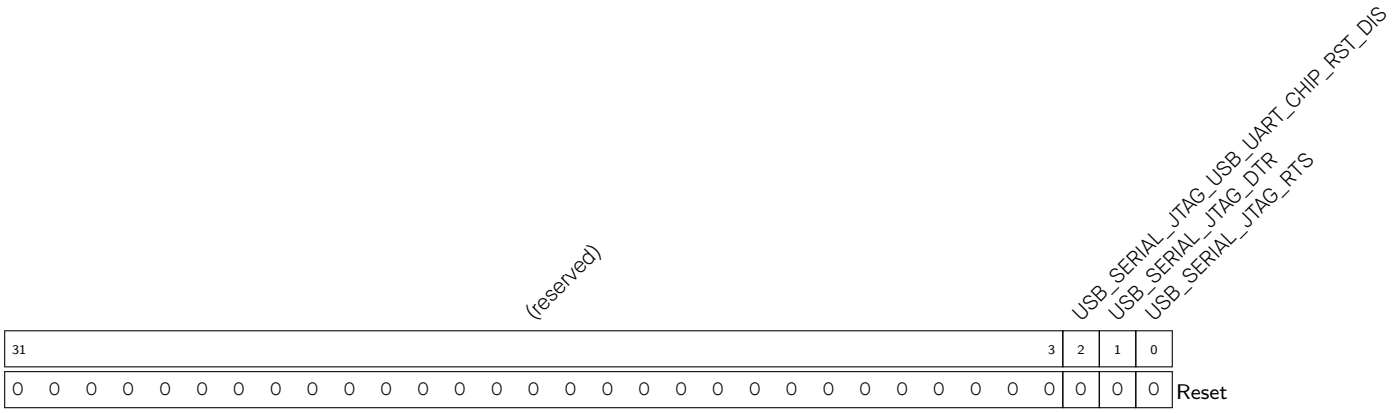
**USB_SERIAL_JTAG_USB_MEM_CLK_EN** Configures whether to force clock on for USB memory.

0: No effect

1: Force

(R/W)

Register 30.7. 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)

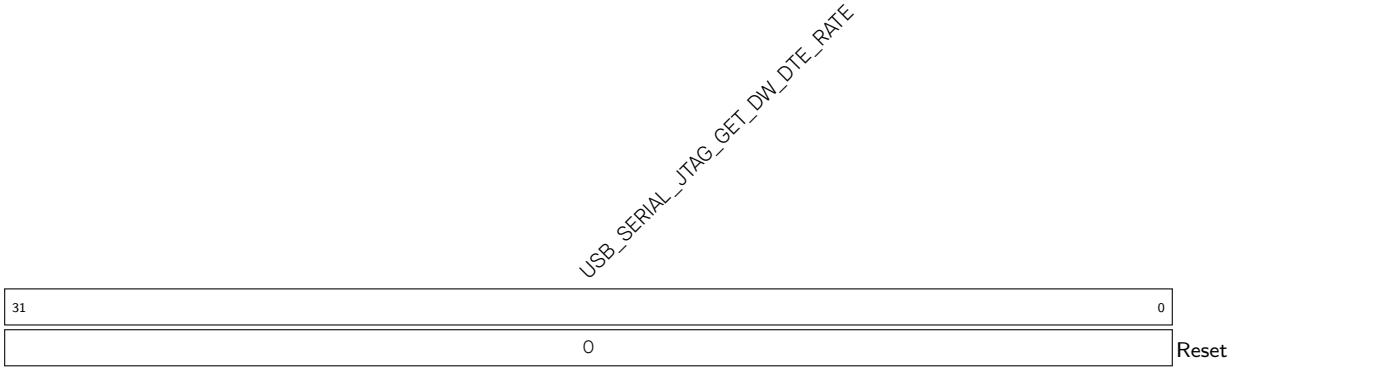
**USB_SERIAL_JTAG_USB_UART_CHIP_RST_DIS** Configures whether to disable chip reset from USB serial channel.

0: No effect

1: Disable

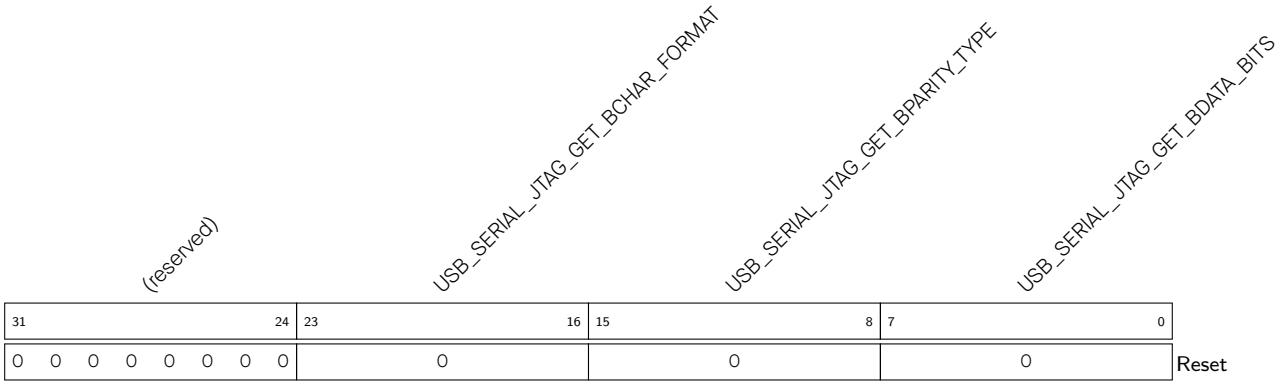
(R/W)

Register 30.8. 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 30.9. USB_SERIAL_JTAG_GET_LINE_CODE_W1_REG (0x005C)

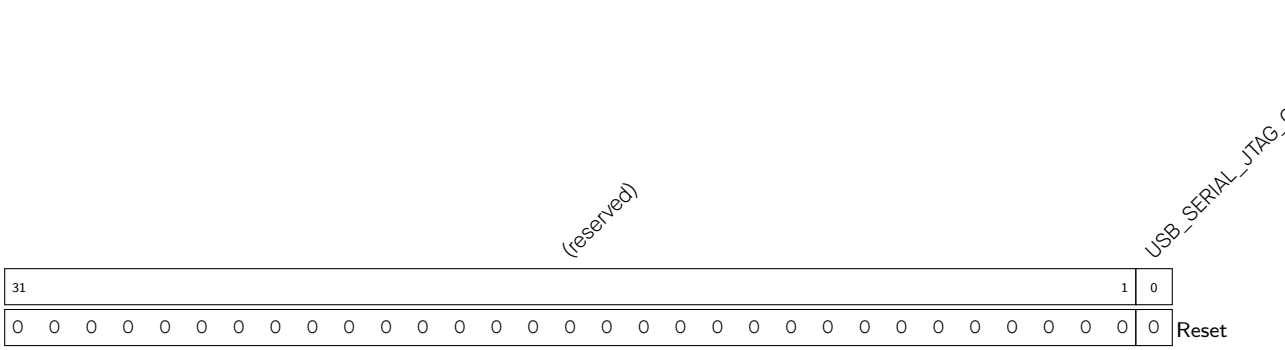



**USB_SERIAL_JTAG_GET_BDATA_BITS** Configures the value of bDataBits set by software, which is requested by GET_LINE_CODING command. (R/W)

**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 30.10. USB_SERIAL_JTAG_CONFIG_UPDATE_REG (0x0060)



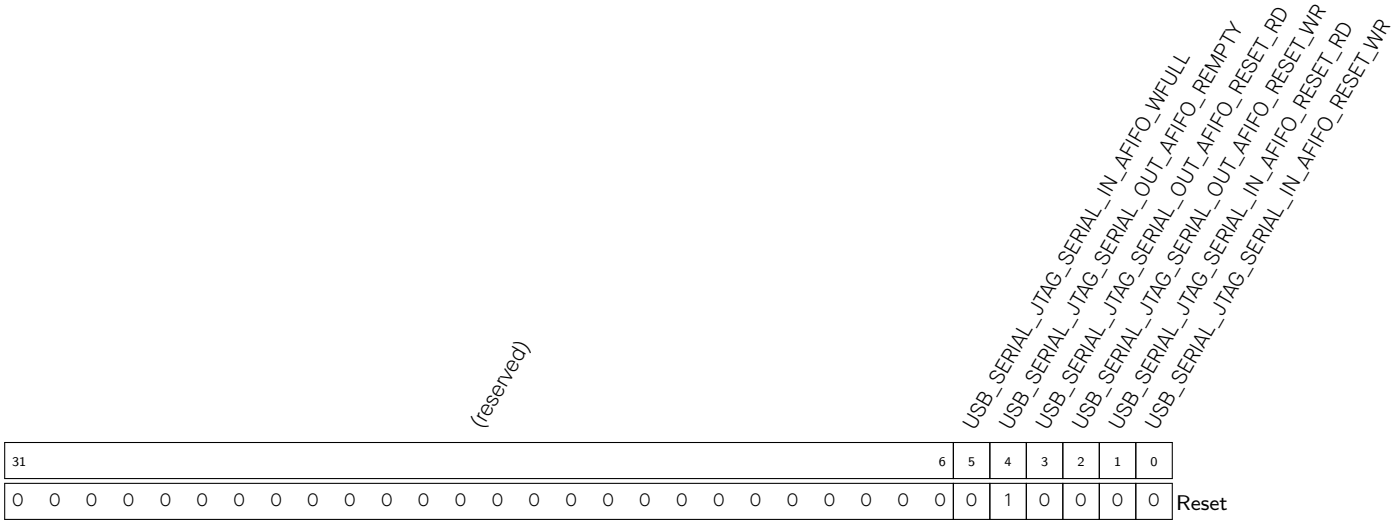

**USB_SERIAL_JTAG_CONFIG_UPDATE** Configures whether to update the value of configuration registers from APB clock domain to 48 MHz clock domain.

0: No effect

1: Update

(WT)

Register 30.11. 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

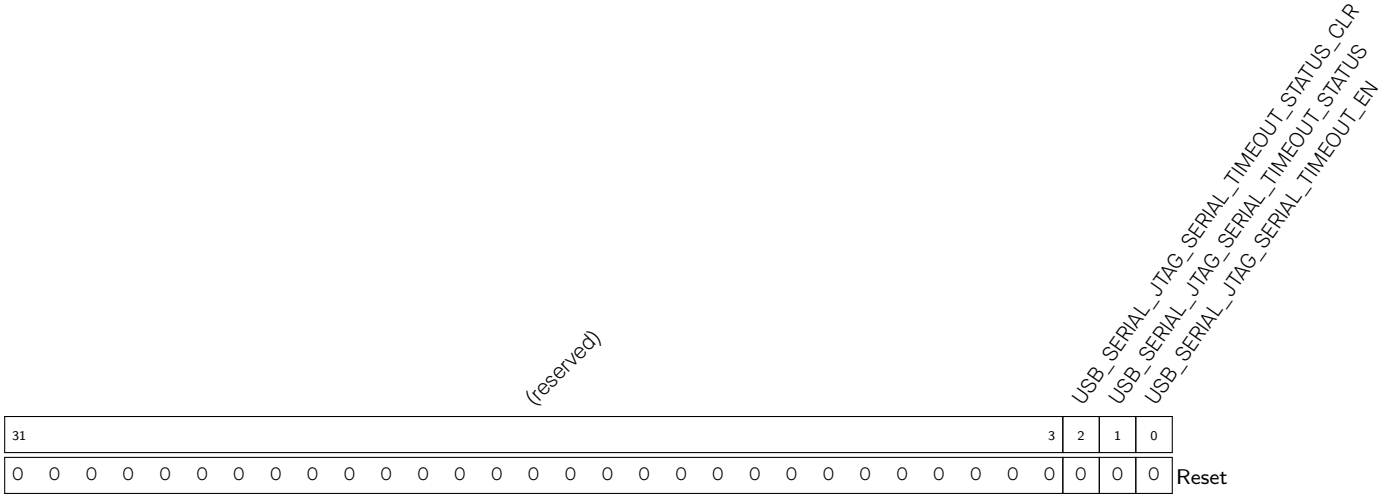
1: Reset

(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 30.12. USB_SERIAL_JTAG_SERIAL_EP_TIMEOUT0_REG (0x006C)




**USB_SERIAL_JTAG_SERIAL_TIMEOUT_EN** USB serial out endpoint timeout enable. When a timeout event occurs, serial out endpoint buffer is automatically cleared and USB_SERIAL_JTAG_SERIAL_TIMEOUT_STATUSis asserted.

0: timeout disabled, buffer will not be cleared automatically

1: timeout enabled

(R/W)

**USB_SERIAL_JTAG_SERIAL_TIMEOUT_STATUS** Timeout status bit

0: No timeout occurred.

1: Serial out endpoint has triggered a timeout event.

(R/WTC/SS)

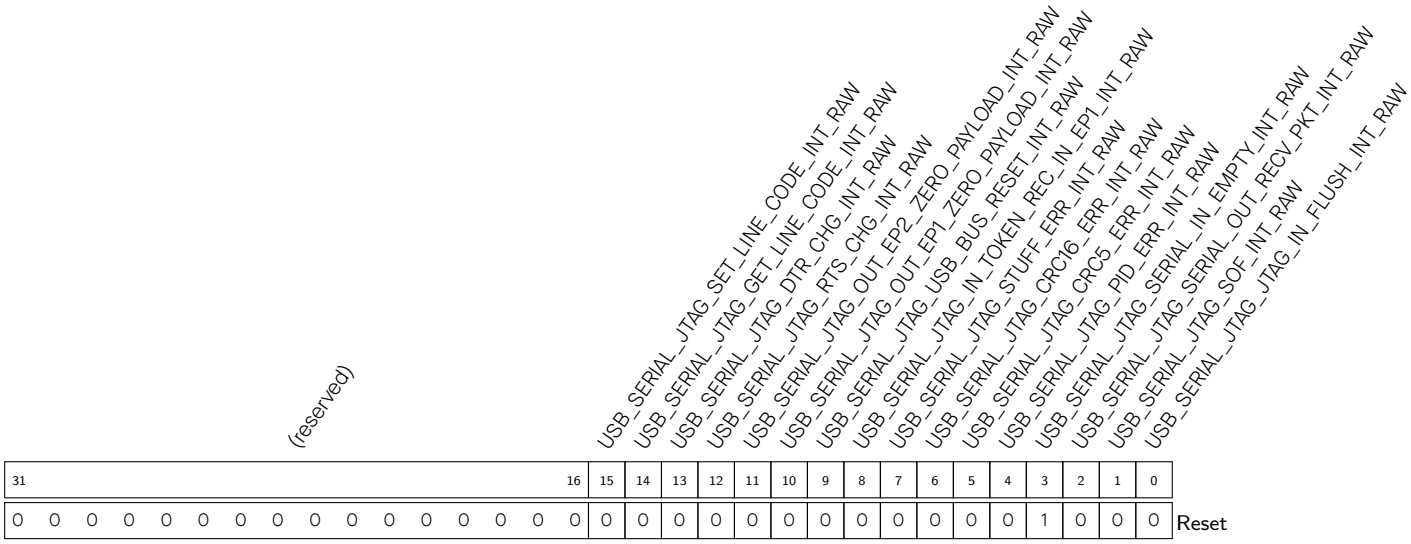
**USB_SERIAL_JTAG_SERIAL_TIMEOUT_STATUS_CLR** Write 1 to clear USB_SERIAL_JTAG_SERIAL_TIMEOUT_STATUS. (WT)

Register 30.13. USB_SERIAL_JTAG_SERIAL_EP_TIMEOUT1_REG (0x0070)




**USB_SERIAL_JTAG_SERIAL_TIMEOUT_MAX** USB serial out endpoint timeout max threshold value, indicates the maximum time that waiting for chips to take away data in memory. This value is in steps of 20.83 ns. (R/W)

Register 30.14. 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 30.14. 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)

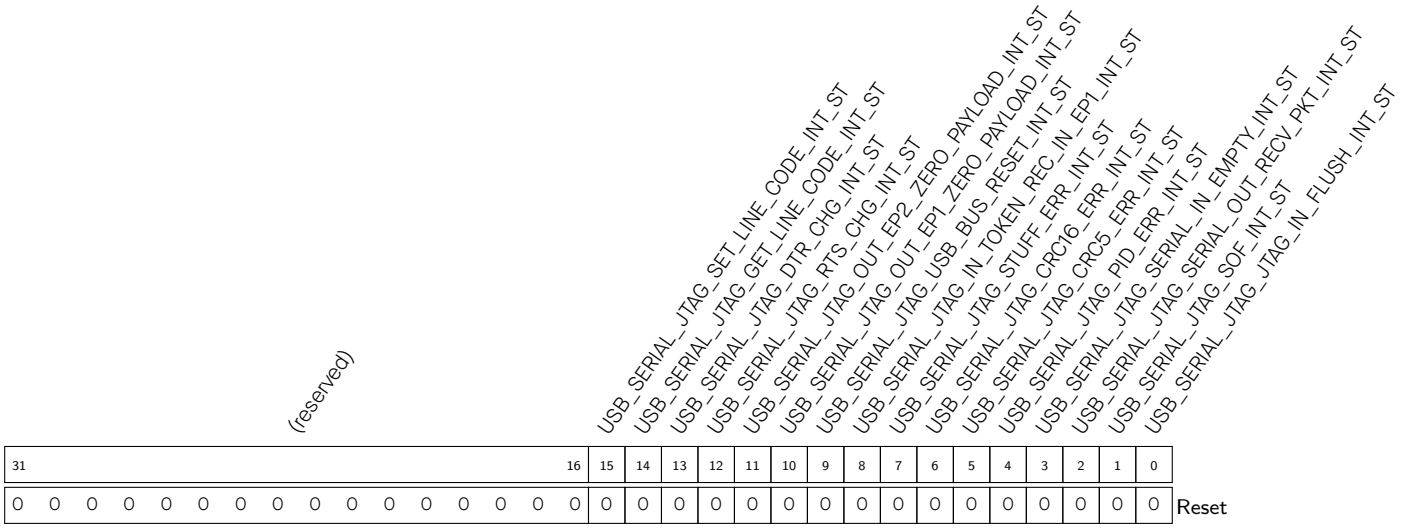
**USB_SERIAL_JTAG_RTS_CHG_INT_RAW** The raw interrupt status of USB_SERIAL_JTAG_RTS_CHG_INT. (R/WTC/SS)

**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 30.15. 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 30.15. USB_SERIAL_JTAG_INT_ST_REG (0x000C)

Continued from the previous page...

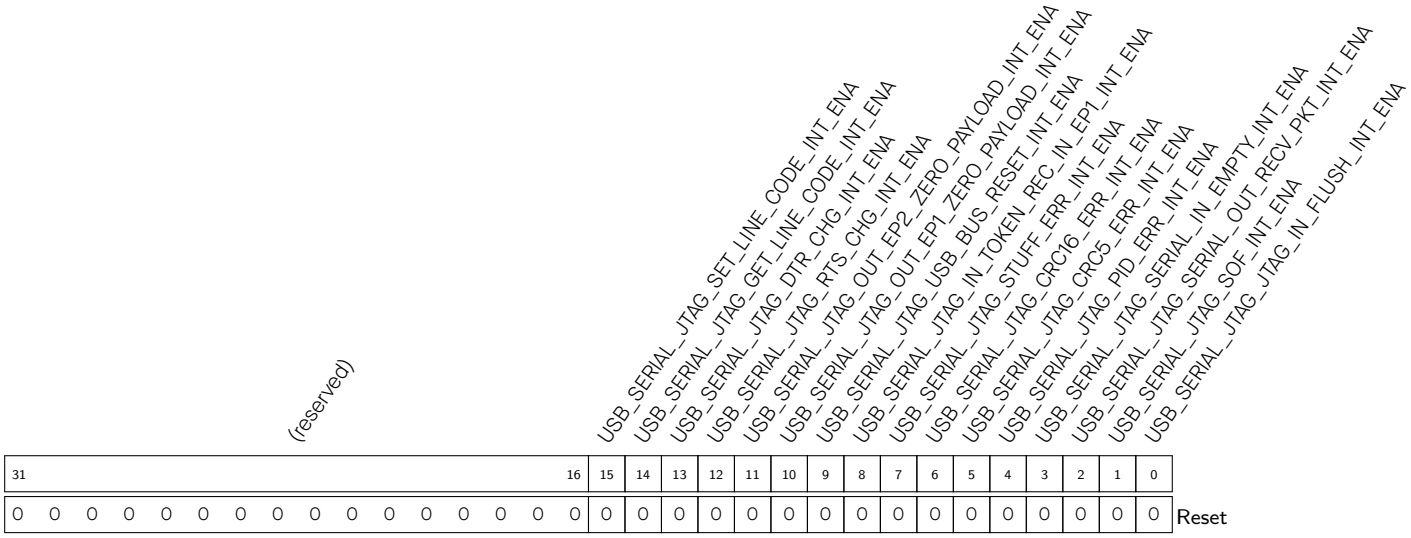
**USB_SERIAL_JTAG_RTS_CHG_INT_ST** The masked interrupt status of USB_SERIAL_JTAG_RTS_CHG_INT. (RO)

**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 30.16. 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 30.16. USB_SERIAL_JTAG_INT_ENA_REG (0x0010)

Continued from the previous page...

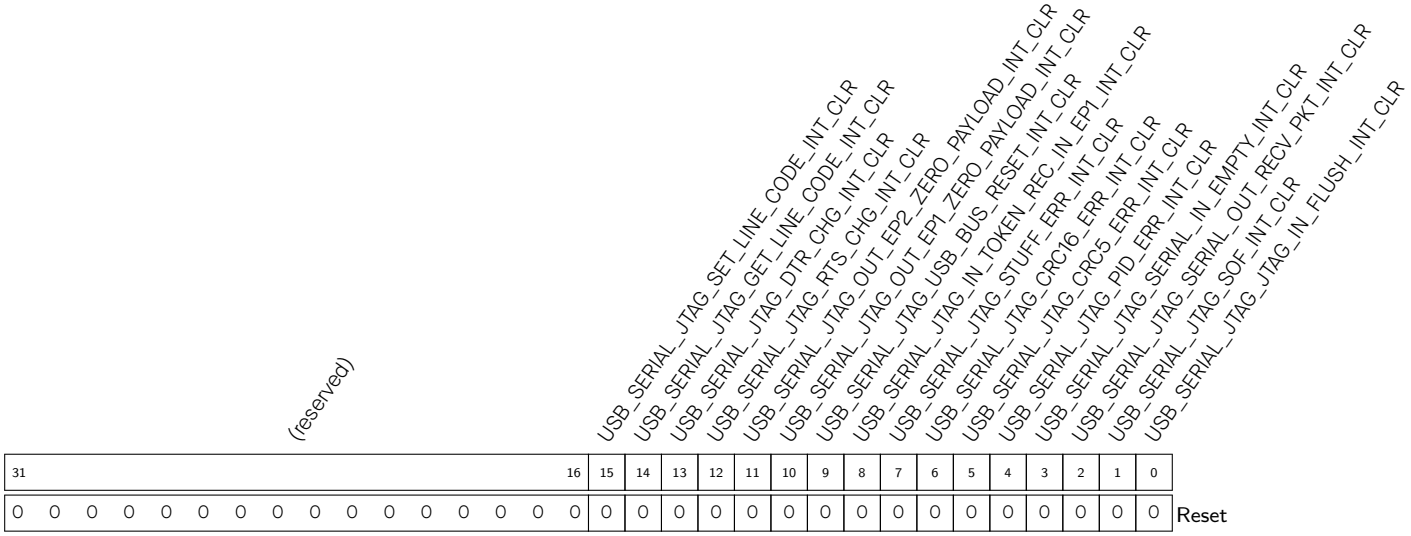
**USB_SERIAL_JTAG_RTS_CHG_INT_ENA** Write 1 to enable USB_SERIAL_JTAG_RTS_CHG_INT. (R/W)

**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 30.17. 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 30.17. USB_SERIAL_JTAG_INT_CLR_REG (0x0014)

Continued from the previous page...

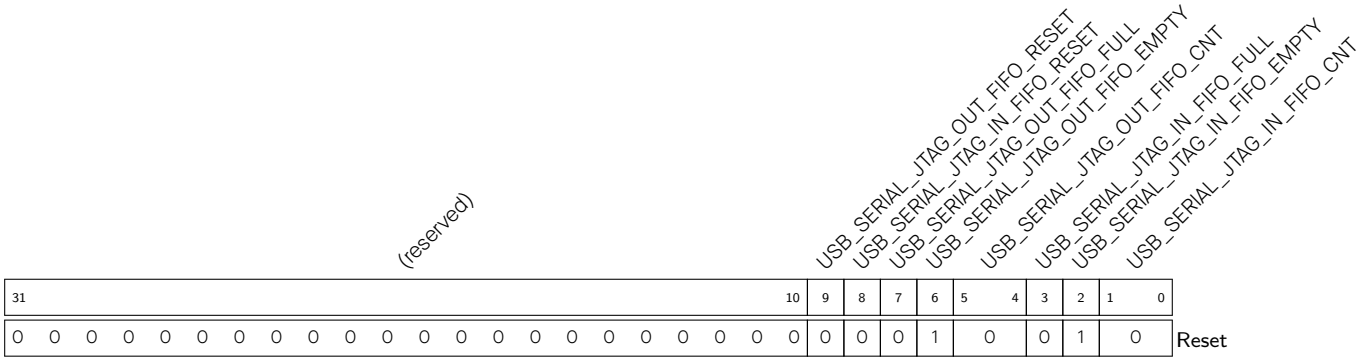
**USB_SERIAL_JTAG_RTS_CHG_INT_CLR** Write 1 to clear USB_SERIAL_JTAG_RTS_CHG_INT. (WT)

**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 30.18. 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)

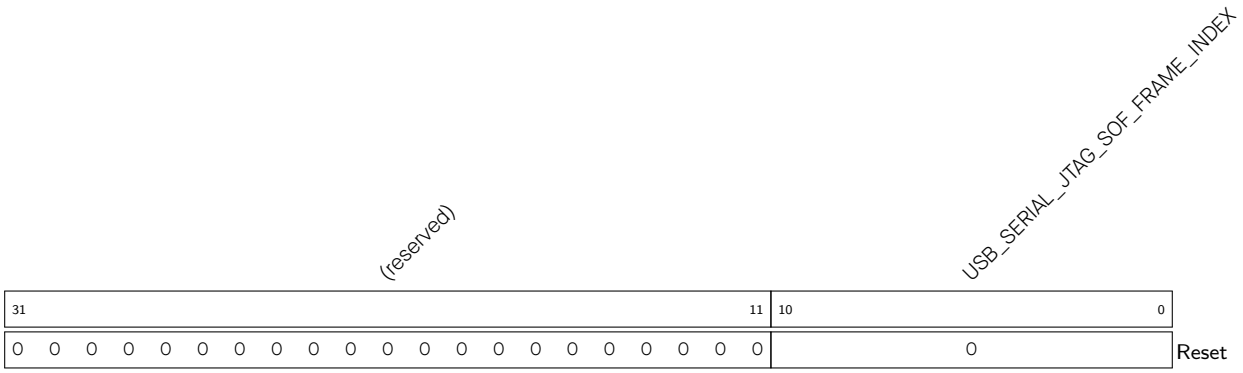
**USB_SERIAL_JTAG_OUT_FIFO_RESET** Configures whether to reset JTAG OUT FIFO.

0: No effect

1: Reset

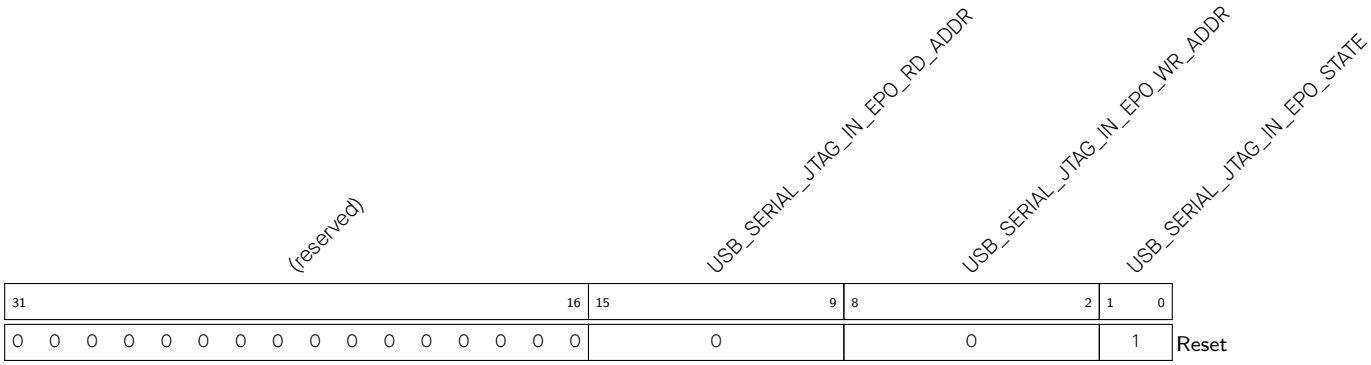
(R/W)

Register 30.19. USB_SERIAL_JTAG_FRAM_NUM_REG (0x0024)




**USB_SERIAL_JTAG_SOF_FRAME_INDEX** Represents frame index of received SOF frame. (RO)

Register 30.20. USB_SERIAL_JTAG_IN_EP0_ST_REG (0x0028)

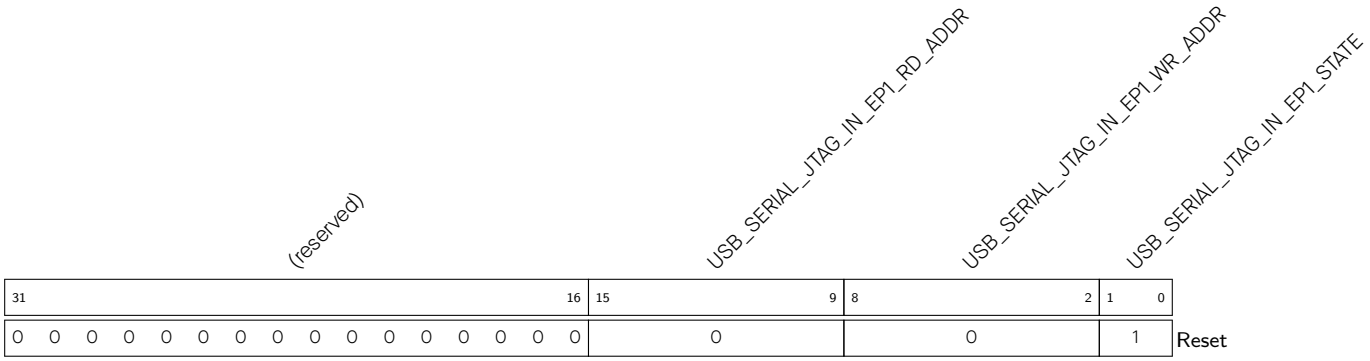



**USB_SERIAL_JTAG_IN_EP0_STATE** Represents state of IN Endpoint 0. (RO)

**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 30.21. USB_SERIAL_JTAG_IN_EP1_ST_REG (0x002C)

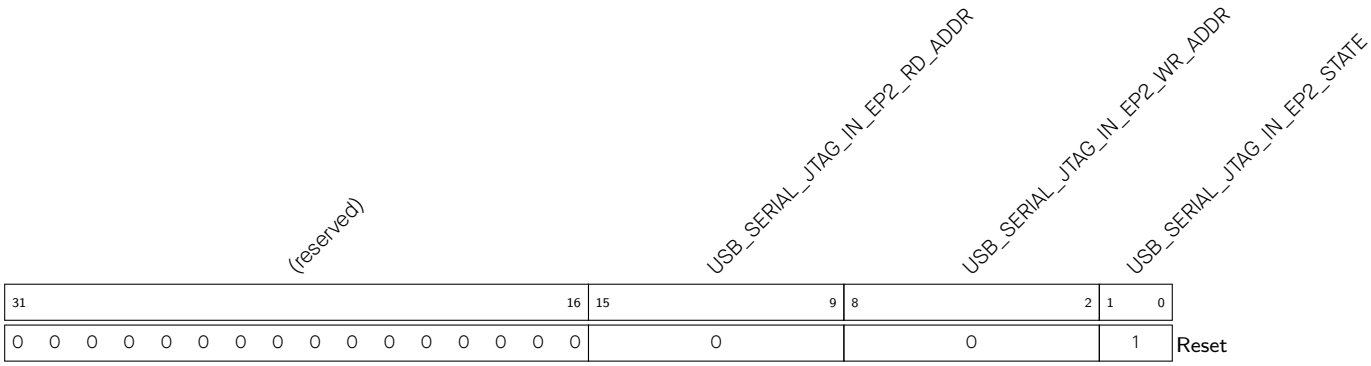



**USB_SERIAL_JTAG_IN_EP1_STATE** Represents state of IN Endpoint 1. (RO)

**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 30.22. USB_SERIAL_JTAG_IN_EP2_ST_REG (0x0030)

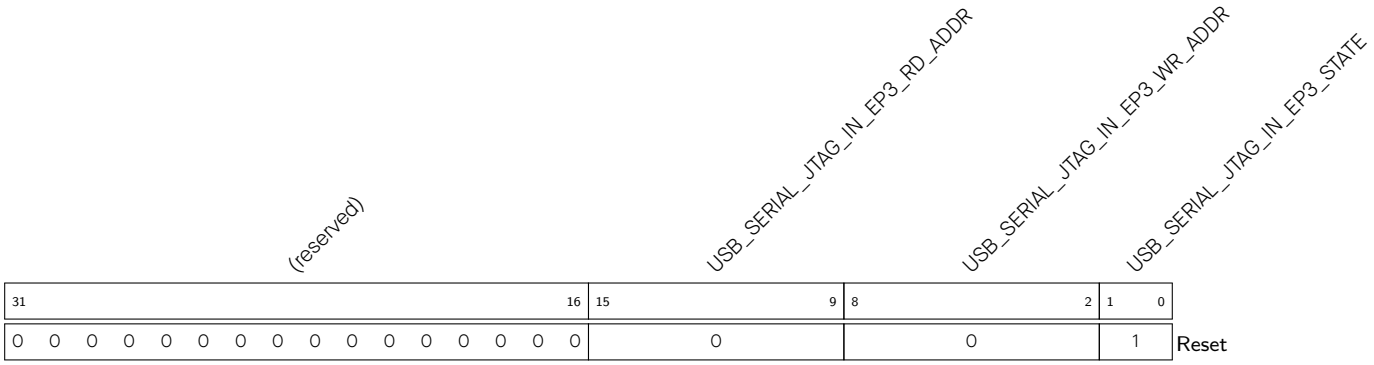



**USB_SERIAL_JTAG_IN_EP2_STATE** Represents state of IN Endpoint 2. (RO)

**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 30.23. USB_SERIAL_JTAG_IN_EP3_ST_REG (0x0034)

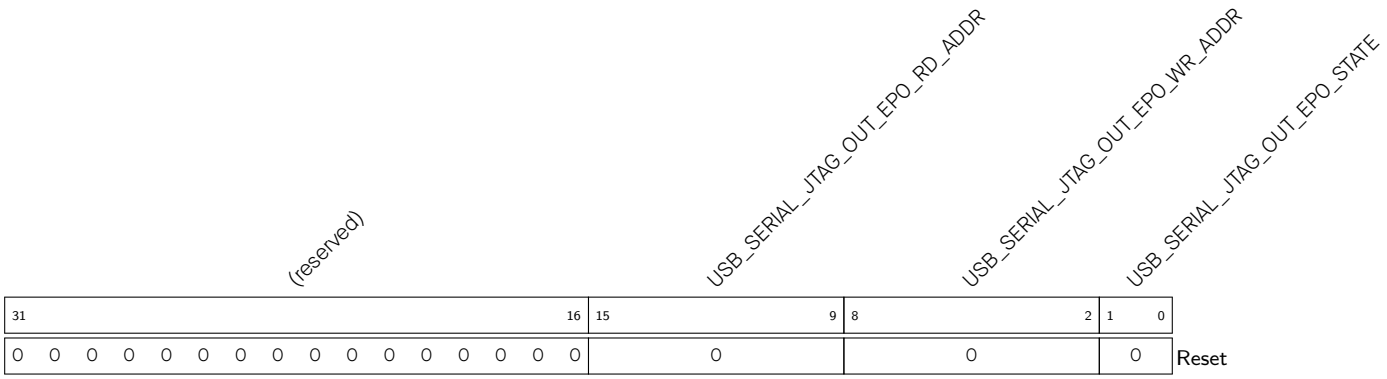



**USB_SERIAL_JTAG_IN_EP3_STATE** Represents state of IN Endpoint 3. (RO)

**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 30.24. 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 30.25. USB_SERIAL_JTAG_OUT_EP1_ST_REG (0x003C)



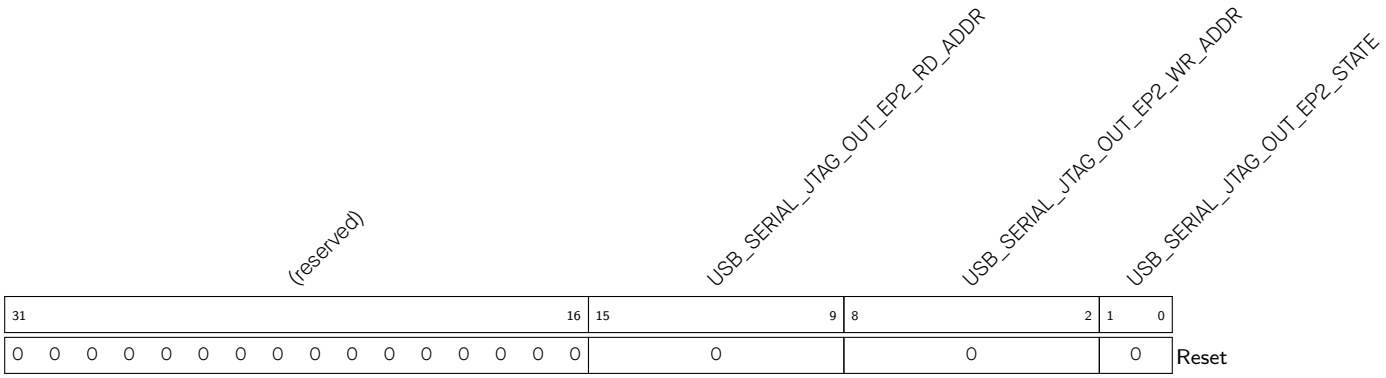

**USB_SERIAL_JTAG_OUT_EP1_STATE** Represents state of OUT Endpoint 1. (RO)

**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 30.26. USB_SERIAL_JTAG_OUT_EP2_ST_REG (0x0040)

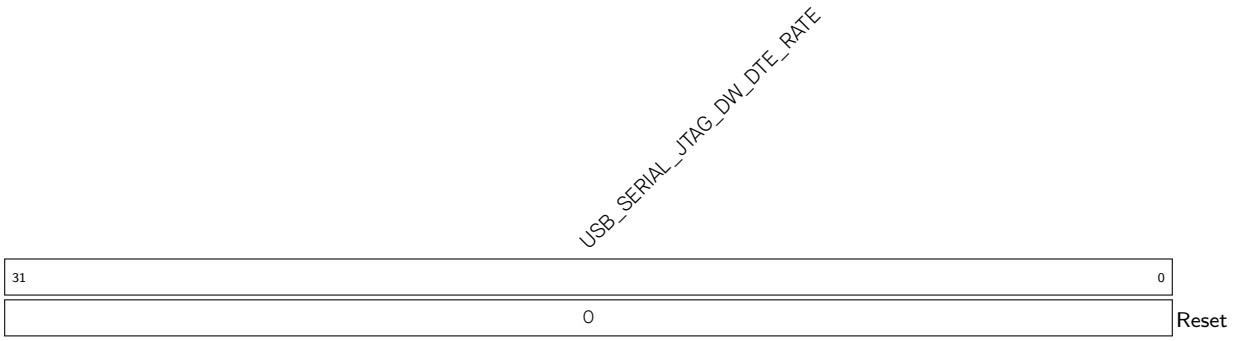



**USB_SERIAL_JTAG_OUT_EP2_STATE** Represents state of OUT Endpoint 2. (RO)

**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)

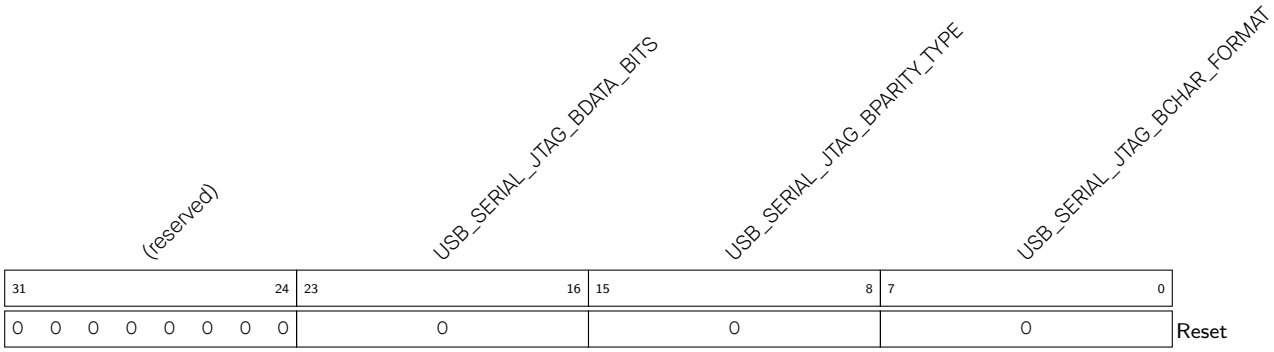
**USB_SERIAL_JTAG_OUT_EP2_RD_ADDR** Represents read data address of OUT endpoint 2. (RO)

Register 30.27. 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 30.28. USB_SERIAL_JTAG_SET_LINE_CODE_W1_REG (0x0054)

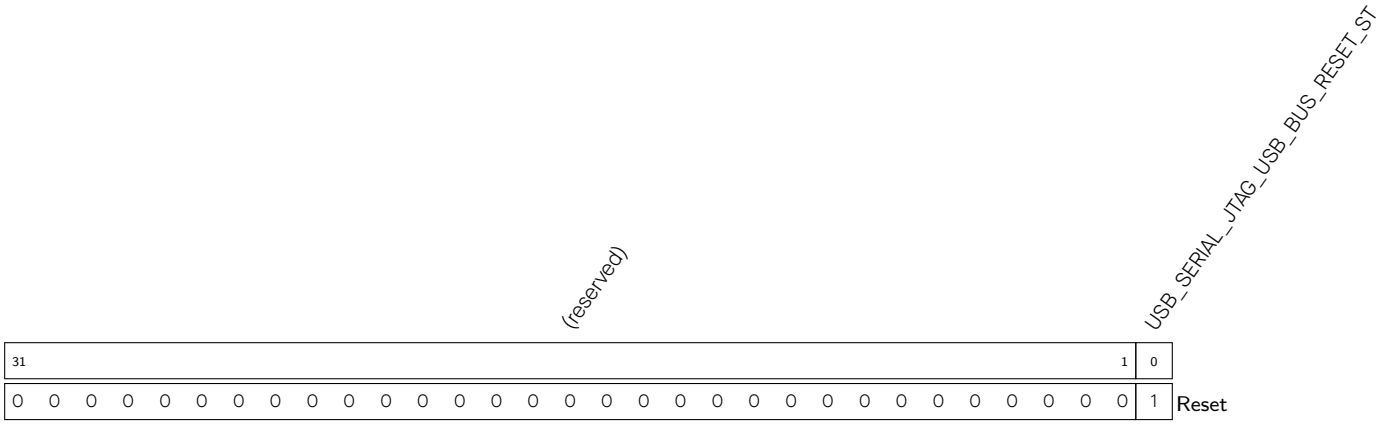



**USB_SERIAL_JTAG_BCHAR_FORMAT** Represents the value of bCharFormat set by host through SET_LINE_CODING command. (RO)

**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 30.29. USB_SERIAL_JTAG_BUS_RESET_ST_REG (0x0068)



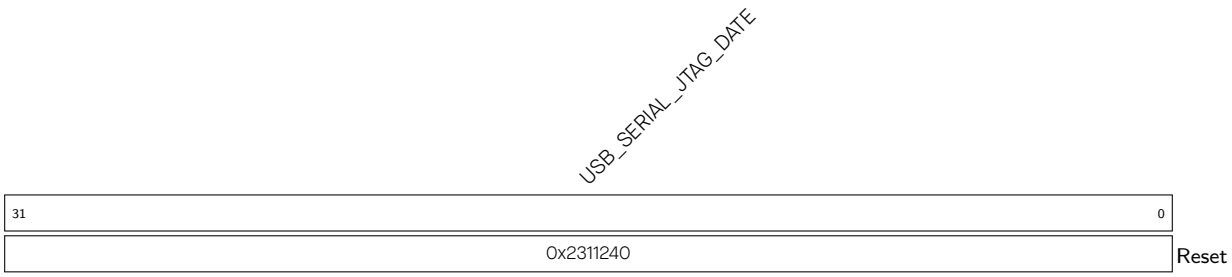

**USB_SERIAL_JTAG_USB_BUS_RESET_ST** Represents whether USB bus reset is released.

0: USB Serial/JTAG is in USB bus reset status

1: USB bus reset is released

(RO)

Register 30.30. USB_SERIAL_JTAG_DATE_REG (0x0080)




**USB_SERIAL_JTAG_DATE** Version control register. (R/W)

# Chapter 31

# SDIO Slave Controller (SDIO)

## 31.1 Overview

The ESP32-C61 features hardware support for the Secure Digital Input/Output (SDIO) device interface that conforms to the SDIO Specification V2.00. This interface allows an SDIO host to access the ESP32-C61 using the SDIO bus protocol.

The SDIO host can directly access the ESP32-C61 SDIO interface registers or use the Direct Memory Access (DMA) engine to access shared memory. This approach reduces processor overhead and maintains high performance.

## 31.2 Features

The SDIO Slave Controller has the following features:

- Compatible with SD Physical Layer Specification V2.00 and SDIO V2.00 specifications
- Support for two I/O functions (excluding function 0)
- Support for SPI, 1-bit SDIO, and 4-bit SDIO transfer modes
- Clock frequency range from 0 to 50 MHz
- Configurable sample and drive clock edge
- Integrated and SDIO-accessible registers for information exchange
- Support for SDIO interrupt mechanism
- Automatic padding data and discarding the padded data on the SDIO bus
- Block size up to 512 bytes
- Bidirectional interrupt vector between host and slave
- DMA support for data transfer
- Wake-up from sleep mode when connection is retained

**Note:**
The SDIO Slave controller can be used together with the USB Serial/JTAG controller in 1-bit SDIO transfer mode, but not in 4-bit SDIO transfer mode.

## 31.3 Architecture Overview

Figure 31.3-1 shows the functional block diagram of the SDIO slave module.

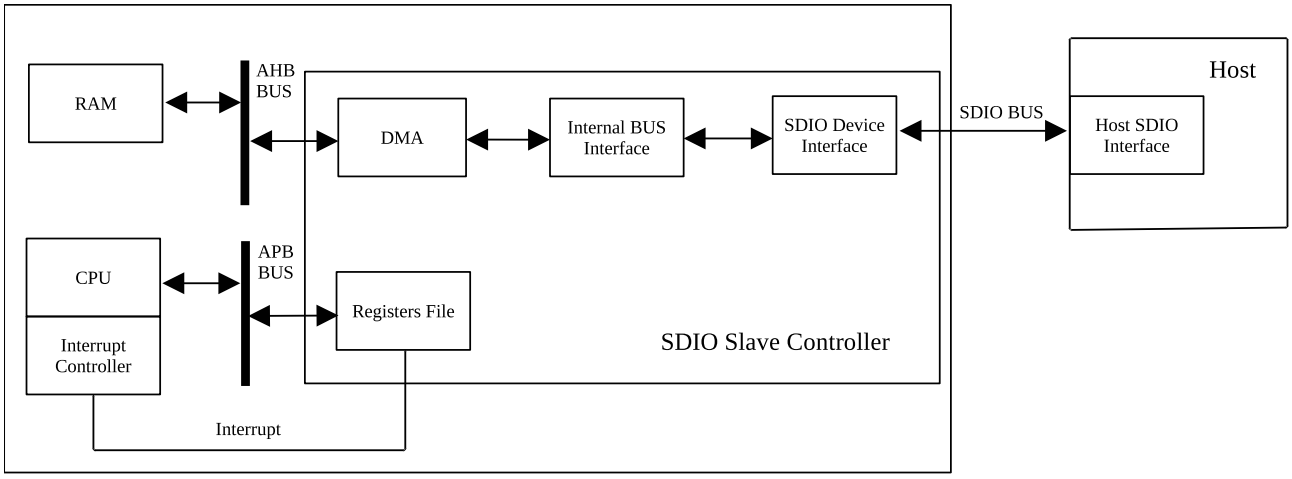



Figure 31.3-1. SDIO Slave Block Diagram

In this figure, the Host represents any device compatible with SDIO Specification V2.00. It communicates with the ESP32-C61 (configured as the SDIO slave) via the standard SDIO bus.

The SDIO Device Interface block enables efficient communication with the external host by providing direct access to SDIO interface registers. It also supports DMA operation for high-speed data transfer over the Advanced High-Performance Bus (AHB) without engaging the CPU.

## 31.4 Standards Compliance

The ESP32-C61 SDIO Slave Controller conforms to the following standards:

- SD Specifications Part 1 Physical Layer Specification Version 2.00 (referred to as Physical Layer Specification V2.00 in this chapter)
- SD Specifications Part E1 SDIO Specification Version 2.00, January 30, 2007 (referred to as SDIO Specification V2.00 in this chapter)

## 31.5 Functional Description

### 31.5.1 Physical Bus

- Bus modes: SPI, 1-bit, and 4-bit SDIO transfer modes.
- Bus signals: Standard SDIO Specification V2.00 signals, including CS/DI/SCLK/DO/IRQ for SPI mode, CMD/CLK/DATA/IRQ for SDIO 1-bit mode, and CMD/CLK/DAT[3:0] for SDIO 4-bit mode
- Bus speed modes: Full-speed mode (0–50 MHz) and low-speed mode (0–400 kHz)
- I/O functions: Two I/O functions in addition to function 0. Function 0 is used only for CCCR, FBR, and CIS operations. Functions 1 and 2 can be used simultaneously to transfer application data packets (such

as Wi-Fi and Bluetooth packets) and to access SLC Host registers.

For more information, please refer to the Physical Layer Specification V2.00 and SDIO Specification V2.00.

## 31.5.2 Supported Commands

The SDIO Slave Controller primarily supports the IO_RW_DIRECT (CMD52) and IO_RW_EXTENDED (CMD53) data transfer commands.

IO_RW_DIRECT (CMD52) is used to access registers and transfer data. Figure 31.5-1 shows its fields. For details on each field, please refer to the SDIO Specification V2.00.

| S | D | Command Index 110100b | R/W flag | Function Number | RAW flag | Stuff | Register Address | Stuff | Write Data of Stuff Bits | CRC7 | E |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 6 | 1 | 3 | 1 | 1 | 17 | 1 | 8 | 7 | 1 |

Figure 31.5-1. CMD52 Content

IO_RW_EXTENDED (CMD53) initiates the transfer of packets of an arbitrary length. Figure 31.5-2 shows its fields. For details on each field, please refer to the SDIO Specification V2.00.

| S | D | Command Index 110101b | R/W flag | Function Number | Block Mode 1b | OP Code | Register Address | Byte/Block Count Roundup (Packet_length/Block_Size) | CRC7 | E |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 6 | 1 | 3 | 1 | 1 | 17 | 9 | 7 | 1 |

Figure 31.5-2. CMD53 Content

## 31.5.3 I/O Function 0 Address Space

I/O function 0 is used only for Card Common Control Registers (CCCR), Function Basic Registers (FBR), and Card Information Structure (CIS) operations. Figure 31.5-3 shows its address space map, as specified by the SDIO Specification. For details on each section in this map, please refer to the SDIO Specification V2.00.

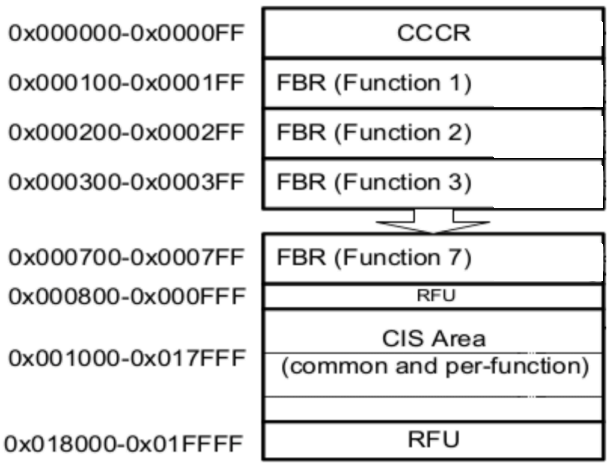



Figure 31.5-3. Function 0 Address Space

As defined in the SDIO Specification, CCCR are common control registers, FBR are control configuration registers for each function, and CIS are status registers for storing card information, such as version, power consumption, and manufacturer. The functions of these registers are optional, and are detailed in the SDIO Specification.

The CCCR configuration of ESP32-C61 SDIO slave is shown in Table 31.5-1, and the FBR configuration is shown in Table 31.5-2.

Table 31.5-1. SDIO Slave CCCR Configuration

| Address | Register Name | Bit 7 | Bit 6 | Bit 5 | Bit 4 | Bit 3 | Bit 2 | Bit 1 | Bit 0 |
|---|---|---|---|---|---|---|---|---|---|
| 0x00 | CCCR/SDIO Revision | Set SDIO bit[3:0] using HINF_SDIO_VER[7:4] in HINF_CFG_DATA1_REG | | | | Set CCCR bit[3:0] using HINF_SDIO_VER[3:0] in HINF_CFG_DATA1_REG | | | |
| 0x01 | SD Specification Revision | 0 (RFU) | | | | Set SD bit[3:0] using HINF_SDIO_VER[11:8] in HINF_CFG_DATA1_REG | | | |
| 0x02 | I/O Enable | 0 (IOE[7:3]) | | | | | R/W (IOE[2:1]) | | 0 (RFU) |
| 0x03 | I/O Ready | 0 (IOR[7:3]) | | | | | R (IOR[2:1]) | | 0 (RFU) |
| 0x04 | Int Enable | 0 (IEN[7:3]) | | | | | R/W (IEN[2:1]) | | R/W (IENM) |
| 0x05 | Int Pending | 0 (INT[7:3]) | | | | | R (INT[2:1]) | | 0 (RFU) |
| 0x06 | I/O Abort | 0 (RFU) | | | | W (RES) | W (AS[2:0]) | | |
| 0x07 | Bus Interface Control | R/W (CD Disable) | 1 (SCSI) | R/W (ECSI) | 0 (RFU) | | | | R/W (Bus Width[1:0]) |

Cont'd on next page

Table 31.5-1. SDIO Slave CCCR Configuration – cont'd from previous page

| Address | Register Name | Bit 7 | Bit 6 | Bit 5 | Bit 4 | Bit 3 | Bit 2 | Bit 1 | Bit 0 |
|---|---|---|---|---|---|---|---|---|---|
| 0x08 | Card Capability | 0 (4BLS) | 0 (LSC) | R/W (E4MI) | 1 (S4MI) | 0 (SBS) | 1 (SRW) | 1 (SMB) | 1 (SDC) |
| 0x09-0x0B | Common CIS Pointer | Address 0x09: 0x0; Address 0x0A: 0x10; Address 0x0B: 0x0 (Pointer to card's common CIS) | | | | | | | |
| 0x0C | Bus Suspend | 0 (RFU) | | | | | | 0 (BR) | 0 (BS) |
| 0x0D | Function Select | 0 (DF) | 0 (RFU) | | | 0 (FS[3:0]) | | | |
| 0x0E | Exec Flags | 0 (EX[7:1]) | | | | | | | 0 (EXM) |
| 0x0F | Ready Flags | 0 (RF[7:1]) | | | | | | | 0 (RFM) |
| 0x10-0x11 | FN0 Block Size | R/W (Supported range: 0-512) (I/O block size for Function 0) | | | | | | | |
| 0x12 | Power Control | 0 (RFU) | | | | | | R/W (EMPC) | 1 (SMPC) |
| 0x13 | High-Speed | 0 (RFU) | | | | | | R/W (EHS) | Note [a] (SHS) |

[a] Set SHS using HINF_HIGHSPEED_ENABLE in HINF_CFG_DATA1_REG.

Table 31.5-2. SDIO Slave FBR Configuration

| Address | Bit 7 | Bit 6 | Bit 5 | Bit 4 | Bit 3 | Bit 2 | Bit 1 | Bit 0 |
|---|---|---|---|---|---|---|---|---|
| 0x100 | 0 (Function 1 CSA enable) | 0 (Function 1 supports CSA) | 0 (RFU) | | 0 (Function 1 Standard SDIO Function interface code) | | | |
| 0x101 | 0 (Function 1 Extended standard SDIO Function interface code) | | | | | | | |
| 0x102 | 0 (RFU) | | | | | | R/W (EPS) | 0 (SPS) |
| 0x109-0x10B | Address 0x109: 0x0; Address 0x10A: 0x11; Address 0x10B: 0x0 (Pointer to Function 1 CIS) | | | | | | | |
| 0x110-0x111 | R/W (Supported range: 0-512) (I/O block size for Function 1) | | | | | | | |

Cont'd on next page

Table 31.5-2. SDIO Slave FBR Configuration – cont'd from previous page

| Address | Bit 7 | Bit 6 | Bit 5 | Bit 4 | Bit 3 | Bit 2 | Bit 1 | Bit 0 |
|---|---|---|---|---|---|---|---|---|
| 0x200 | 0 (Function 2 CSA enable) | 0 (Function 2 supports CSA) | 0 (RFU) | | 0x2 (Function 2 Standard SDIO Function interface code) | | | |
| 0x201 | 0 (Function 2 Extended standard SDIO Function interface code) | | | | | | | |
| 0x202 | 0 (RFU) | | | | | | R/W (EPS) | 0 (SPS) |
| 0x209-0x20B | Address 0x209: 0x0; Address 0x20A: 0x12; Address 0x20B: 0x0 (Pointer to Function 2 CIS) | | | | | | | |
| 0x210-0x211 | R/W (Supported range: 0-512) (I/O block size for Function 2) | | | | | | | |

## 31.5.4 I/O Function 1/2 Address Space Map

I/O function 1 and function 2 have identical functions and permissions. They can be used simultaneously or independently to transmit application data (such as Wi-Fi or Bluetooth data) in fixed-address packets or incremental-address packets. Both functions can access the same set of SLC Host registers. Figure 31.5-4 shows their address space map. All segments in this space can be accessed by the host.

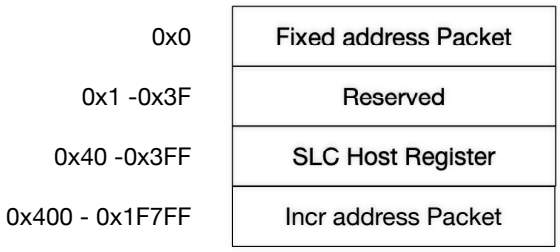



Figure 31.5-4. Function 1/2 Address Space Map

### 31.5.4.1 Accessing SLC HOST Register Space

The host can access registers in the contiguous address range from 0x40 to 0x3FF in the slave via I/O functions 1 or 2. To access these registers, the host sets the Register Address field of CMD52 or CMD53 to the low 10 bits of the address. CMD53 also allows the host to access multiple registers in a single operation for higher transfer rates.

From SLCHOST_CONF_W0_REG and SLCHOST_CONF_W15_REG, there are 52 bytes of fields accessible and modifiable by both the host and slave, facilitating information exchange.

Both host and slave software can access the SLC Host register space simultaneously. Therefore, an upper-layer mechanism should be implemented to prevent errors caused by concurrent access.

#### 31.5.4.2 Transferring Incremental-Address Packets

When the host uses addresses 0x400 to 0x1F7FF to transmit multiple application data packets (such as Wi-Fi packets), the address field in CMD53 should be set to increment mode, and the OP Code field should be set to 1.

For example, to transfer (send or receive) three data blocks starting from base address 0x500 using CMD53, the host should:

- Set the Block Mode field in CMD53 to 1 (block data unit)
- Set the OP Code field to 1 (incremental address mode)
- Set the Register Address field to 0x500 (base address)
- Set the Byte/Block Count field to 0x3 (three data blocks)
- Set other fields according to the SDIO Specification

When a packet is transmitted (slave to host or host to slave) through CMD53, the slave determines whether all valid data of the current packet has been transmitted and pads (when sending) or discards (when receiving) invalid data as needed. For more information, see Section 31.5.5.3.

#### 31.5.4.3 Transferring Fixed-Address Packets

When the host uses address 0x0 to transmit application data packets (such as Bluetooth packets), both the address field and OP Code field in CMD53 should be set to 0.

For example, to transfer (send or receive) three data blocks starting from fixed address 0x0 using CMD53, the host should:

- Set the Block Mode field in CMD53 to 1 (block data unit)
- Set the OP Code field to 0 (fixed address mode)
- Set the Register Address field to 0x0 (fixed address)
- Set the Byte/Block Count field to 0x3 (three data blocks)
- Set other fields according to the SDIO Specification

When a packet is transmitted between the host and slave through CMD53, the slave determines whether all valid data of the current packet has been transmitted and pads or discards invalid data as needed. For more information, see Section 31.5.5.3.

### 31.5.5 DMA

The SDIO Slave Controller uses a dedicated DMA to access transfer data from RAM or store it to RAM. As shown in Figure 31.3-1, RAM is accessed over the AHB. For the RAM space accessible by the Controller, please refer to Chapter 5 *System and Memory*. To set the RAM address range that can be accessed for one transfer, please configure the *SHAREMEM*_REG fields of SLC register described in Section 31.8.

The DMA engine provides two channels: SLC0 and SLC1. SLC0 is used for transferring incremental-address packets, while SLC1 handles fixed-address packets. The SDIO slave supports two I/O functions (Function 1 and Function 2) for data transmission. It is recommended to use Function 1 with SLC0 for incremental-address

packets and Function 2 with SLC1 for fixed-address packets. For details on the address ranges of these functions, see Section 31.5.4.

DMA accesses RAM over AHB. You can configure the burst operation mode and type by setting the relevant fields of SDIO_SLCCONF0_REG and SDIO_SLC_BURST_LEN_REG. For more information, see Section 31.8.

#### 31.5.5.1 Linked List

The slave software can use the DMA engine by mounting linked lists. The DMA engine transfers data from the RAM address space specified in the RX (slave-to-host) linked list and stores received data in the RAM address space specified in the TX (host-to-slave) linked list. A linked list is composed of multiple descriptors.

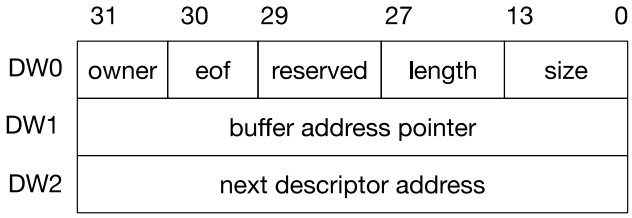



Figure 31.5-5. DMA Linked List Descriptor Structure of the SDIO Slave

The TX and RX linked list descriptors share the same structure, as shown in Figure 31.5-5. Each descriptor consists of three words, with the following fields:

- **owner (DW0) [31]**: Specifies the entity allowed to access the buffer.
  0: CPU
  1: DMA engine
  The slave software sets this field to 1 when creating the descriptor. After the DMA write-back permission is enabled and the corresponding buffer is used by the DMA, the field is cleared to 0.
- **eof (DW0) [30]**: Indicates the end of a data packet.
  0: The current descriptor is not the last descriptor of the packet
  1: The current descriptor is the last descriptor of the packet
  When the host sends a packet to the slave, the slave software should set the field to 0 while creating the descriptor. DMA sets the field of the last descriptor in the packet to 1; When the host receives packets from the slave, the slave software configures the field depending on whether this descriptor is the last descriptor of the packet.
- **reserved (DW0) [29:28]**: Reserved field.
  The slave software sets this field to 0x0.
- **length (DW0) [27:14]**: Indicates the number of valid bytes in the corresponding buffer. When the DMA engine is reading data from the buffer, it indicates the number of bytes that can be read; when DMA engine is storing data in the buffer, it indicates the number of bytes of the stored data.
  When the host sends a packet to the slave, the slave software should set this field to 0x0 while creating the descriptor. DMA writes back the field after the corresponding buffer is used up; when the host receives the packet from the slave and the slave creates the descriptor, the slave software should set this field to the number of bytes that can be read by the corresponding buffer.
- **size (DW0) [13:0]**: Specifies the size of the buffer in bytes.

This field must be word-aligned. The slave software configures it during descriptor creation.

- **buffer address pointer (DW1)**: Points to the buffer address.
  This field must be word-aligned. The slave software configures it during descriptor creation.
- **next descriptor address (DW2)**: Points to the next descriptor in the linked list.
  If no next descriptor exists, this field is set to 0. The slave software configures it during descriptor creation.

For more information on DMA write-back linked list descriptor fields, see Section 31.5.5.2.

The slave software can combine multiple descriptors into a linked list using the next descriptor address (DW2) field. The SDIO slave DMA linked list is shown in Figure 31.5-6.

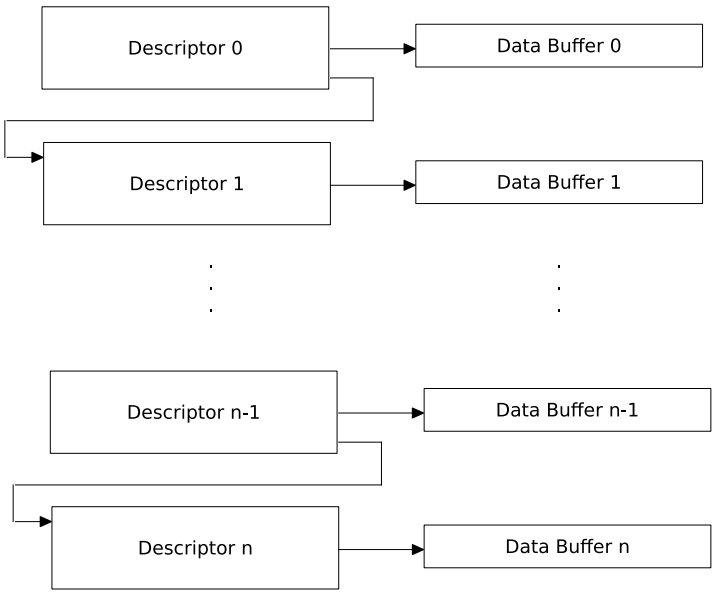



Figure 31.5-6. DMA Linked List of the SDIO Slave

The following example demonstrates the use of a linked list and the `eof` bit. Suppose the slave software creates a linked list with three descriptors:

- Descriptor 0 points to 500 bytes of data, with its `eof` bit set to 0.
- Descriptor 1 points to 200 bytes of data, with its `eof` bit set to 1.
- Descriptor 2 points to 200 bytes of data, with its `eof` bit set to 1.

1. If the first CMD53 command requests 400 bytes, the DMA sends the first 400 bytes from Descriptor 0 to the host.
2. If the second CMD53 command requests 400 bytes, the DMA first sends the remaining 100 bytes from Descriptor 0 to the host, followed by 200 bytes from Descriptor 1. Since the `eof` bit of Descriptor 1 is set to 1, the DMA considers the valid data for the current CMD53 command complete and pads the remaining 100 bytes with invalid data (0x0).
3. If the third CMD53 command requests 400 bytes, the DMA sends 200 bytes from Descriptor 2 to the host. As the `eof` bit of Descriptor 2 is set to 1, the DMA considers the valid data for the current CMD53 command complete and pads the remaining 200 bytes with invalid data (0x0).

#### 31.5.5.2 Write-Back of Linked List

When sending packets from the host to the slave, if the buffer specified by a linked list descriptor is full or a packet transmission ends, the DMA engine jumps to the next descriptor to store subsequent data. Before jumping, the DMA writes back the current descriptor. In DW0, the DMA updates the `eof` and `length` bits to their latest values. The value of the `owner` bit is determined by SDIO_SLC0/1_TX_LOOP_TEST in SDIO_SLCCONF0_REG.

When receiving packets from the slave, if the host reads all data in the buffer specified by a linked list descriptor, the DMA engine jumps to the next descriptor to read subsequent data. Before jumping, the slave software can set SDIO_SLC0/1_RX_AUTO_WRBACK in SDIO_SLCCONF0_REG to 1 so that the DMA writes back the current descriptor. The value to write to the `owner` bit is determined by SDIO_SLC0/1_RX_LOOP_TEST in SDIO_SLCCONF0_REG. Other bits in DW0 remain unchanged.

The relevant register fields are described in Section 31.8.

#### 31.5.5.3 Data Padding and Discarding

To transfer data in blocks, both the host and the slave pad the data sent on the SDIO bus into entire blocks. The slave automatically pads data when sending a packet and discards the padded data after receiving packets.

- When the host sends a data packet to the slave through CMD53 and the amount of data reaches the packet length, the SDIO slave considers the valid data of the current packet complete. At this time, the DMA writes back the current linked list descriptor, sets the `eof` bit of the current descriptor to 1, and generates the SLC0/1_TX_SUC_EOF_INT interrupt. After determining that the valid data is complete, the remaining data of the current packet is considered invalid and is not received into the buffer by the DMA. The slave does not restart receiving data into the next buffer until the next CMD53.
  - For incremental-address packets, the slave determines valid data based on the address. Data with an address greater than or equal to 0x1F800 is considered invalid and is discarded. Therefore, the host should set the CMD53 start address field to 0x1F800 – Packet_length (in bytes). The data flow of incremental-address packets on the SDIO bus is shown in Figure 31.5-7.

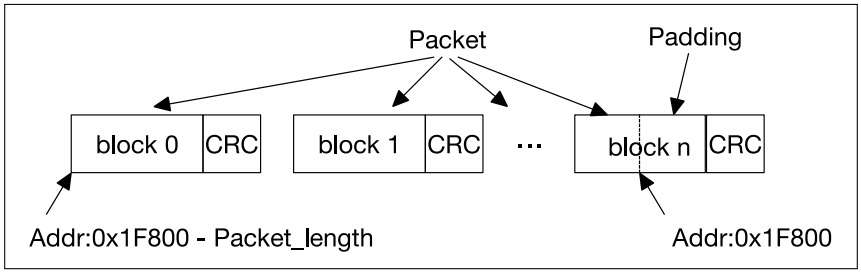



**Figure 31.5-7. Data Flow of Sending Incremental-address Packets From Host to Slave**

  - For fixed-address packets, the slave interprets the first three bytes of the data packet as the packet length. These three bytes are also stored in the buffer specified by the linked list. After the received data length matches the value indicated by these three bytes, any subsequent data is considered invalid and discarded.
- When the host receives data packets (including incremental-address and fixed-address packets) from the slave through a CMD53 command, and the DMA reads the last byte of a buffer where the `eof` bit of

the DMA linked list descriptor is set to 1, the SDIO slave considers the valid data of the current packet complete. At this point, the DMA writes back the current descriptor and generates the SLC0/1_RX_EOF_INT interrupt. After the SDIO slave determines that the valid data is complete, the remaining bits of the current data packet are padded with invalid data (0x0) and are not read from the buffer via DMA. The slave restarts reading data from the buffer via DMA when the next CMD53 command is sent by the host.

**Note:** When the host receives either incremental-address or fixed-address data packets from the slave, the `eof` bit of the DMA linked list descriptor is always used to determine the end of data, rather than the address 0x1F800. Therefore, when the host sends multiple CMD53 commands to obtain multiple data packets, as long as the DMA does not encounter the `eof` bit set to 1 in the descriptors, the slave obtains the data from buffers in sequence according to the linked list and transmits it to the host. When the DMA encounters the `eof` bit set to 1, the data is fetched from the corresponding buffer, and then invalid data is padded to complete the current CMD53 command. The next CMD53 command will fetch data from the buffer pointed to by the next descriptor.

### 31.5.6 SDIO Bus Timing

The SDIO bus operates at high speed, and PCB trace length can affect signal integrity by introducing latency. To ensure proper timing characteristics, the SDIO slave module supports configuration of the input sampling clock edge and output driving clock edge.

When incoming data changes near the rising edge of the clock, the slave samples on the falling edge, and vice versa, as shown in Figure 31.5-8.

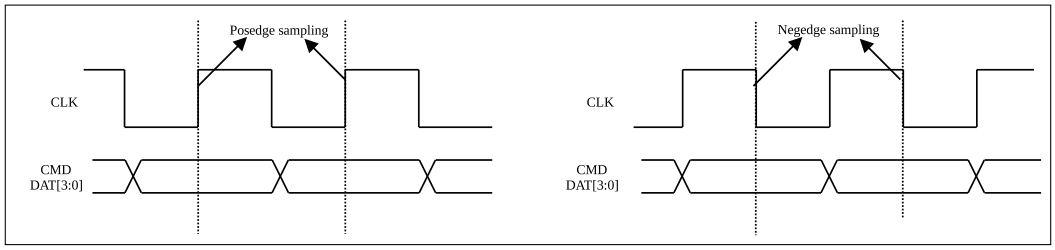



Figure 31.5-8. Sampling Timing Diagram

By default, the voltage level of the GPIO25 strapping pin determines the slave pin's sampling edge. You can also configure the sampling edge using the SLCHOST_CONF_REG register, with the following priority (from high to low): (1) Set SLCHOST_FRC_POS_SAMP to sample the corresponding signal at the rising edge; (2) Set SLCHOST_FRC_NEG_SAMP to sample at the falling edge.

The SLCHOST_FRC_POS_SAMP and SLCHOST_FRC_NEG_SAMP fields are five bits wide, corresponding to the CMD line and four DATA lines (0–3). Setting a bit causes the corresponding line to be sampled at the rising or falling clock edge.

The slave can also select which edge to drive the output lines, to compensate for any latency caused by the physical signal path. The output timing is shown in Figure 31.5-9.

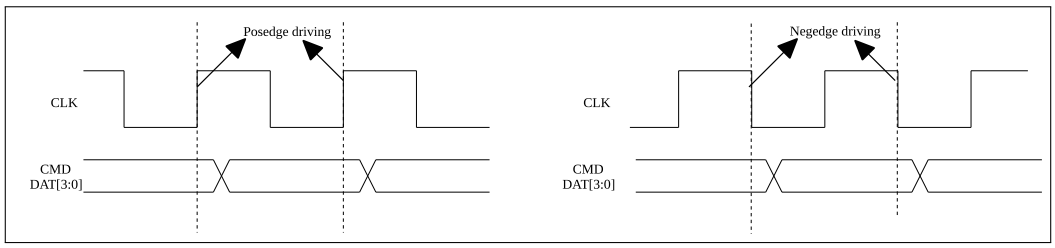



Figure 31.5-9. Output Timing Diagram

By default, the voltage level of the MTDI strapping pin determines the slave pin's output driving edge. You can also configure the output driving edge using the following registers, with priority from high to low: (1) Set SLCHOST_FRC_SDIO11 in SLCHOST_CONF_REG to output the corresponding signal at the falling clock edge; (2) Set SLCHOST_FRC_SDIO22 in SLCHOST_CONF_REG to output at the rising clock edge; (3) Set HINF_HIGHSPEED_ENABLE in HINF_CFG_DATA1_REG and SLCHOST_HSPEED_CON_EN in SLCHOST_CONF_REG, then set the EHS (Enable High-Speed) bit in CCCR at the host side to output at the rising clock edge.

The SLCHOST_FRC_SDIO11 and SLCHOST_FRC_SDIO22 fields are five bits wide, corresponding to the CMD line and four DATA lines (0–3). Setting a bit causes the corresponding line to output at the rising or falling clock edge.

**Note on priority**: The configuration of strapping pins has the lowest priority for controlling the sampling or driving edge. Lower-priority configurations take effect only when higher-priority configurations are not set. For example, the GPIO25 strapping value determines the sampling edge only when SCLHOST_FRC_POS_SAMP and SCLHOST_FRC_NEG_SAMP are not set.

## 31.6 Interrupt

The host and slave can interrupt each other via the interrupt vector. There are eight interrupt vectors between the host and each DMA SLC channel of the slave. To send an interrupt, set the enable bit of the interrupt vector register to 1.

### 31.6.1 Host Interrupt

- *SLCHOST_SLC0/1_RX_NEW_PACKET_INT* : The slave has a packet to send. This interrupt can be triggered when:
  - SDIO_SLC0_RXLINK_START or SDIO_SLC1_RXLINK_START is set to enable DMA.
  - A new RX linked list descriptor is available after DMA processes the RX linked list descriptor with the `eof` bit set to 1.
  - A packet needs to be retransmitted.
- *SLCHOST_SLC0/1_TX_OVF_INT* : Slave TX (transmit) buffer overflow interrupt.
- *SLCHOST_SLC0/1_RX_UDF_INT* : Slave RX (receive) buffer underflow interrupt.
- *SLCHOST_SLC0/1_TOHOST_BITn_INT* (*n*: 0–7): Slave interrupts the host.

### 31.6.2 Slave Interrupt

- *SLC0/1_TX_DSCR_ERR_INT* : Slave TX linked list descriptor error.
- *SLC0/1_RX_DSCR_ERR_INT* : Slave RX linked list descriptor error.
- *SLC0/1_RX_EOF_INT* : Slave RX operation is finished.
- *SLC0/1_RX_DONE_INT* : A single buffer is received by the slave.
- *SLC0/1_TX_SUC_EOF_INT* : Slave TX operation is finished.
- *SLC0/1_TX_DONE_INT* : A single buffer is finished during TX operation.
- *SLC0/1_TX_OVF_INT* : Slave TX buffer overflow interrupt.
- *SLC0/1_RX_UDF_INT* : Slave RX buffer underflow interrupt.
- *SLC0/1_TX_START_INT* : Slave TX start interrupt.
- *SLC0/1_RX_START_INT* : Slave RX start interrupt.
- *SLC_FRHOST_BITn_INT* (*n*: 0–15): The host interrupts the slave via the SLC0 channel if interrupt vector Bit[7:0] is set or via the SLC1 channel if interrupt vector Bit[15:8] is set.

## 31.7 Packet Sending and Receiving Procedure

The SDIO host and slave devices must follow specific data transfer procedures to successfully exchange data over the SDIO interface. In addition to SDIO Specifications, ESP32-C61 should also follow the procedures below to transmit data over higher abstraction layers, such as Wi-Fi and Bluetooth.

### 31.7.1 Sending Packets to SDIO Host

The transmission of packets from the slave to the host is initiated by the slave. The host is notified with an interrupt (for details, refer to the SDIO Specification). After the host reads the relevant information from the slave, it initiates an SDIO bus transmission accordingly. The procedure is illustrated in Figure 31.7-1.

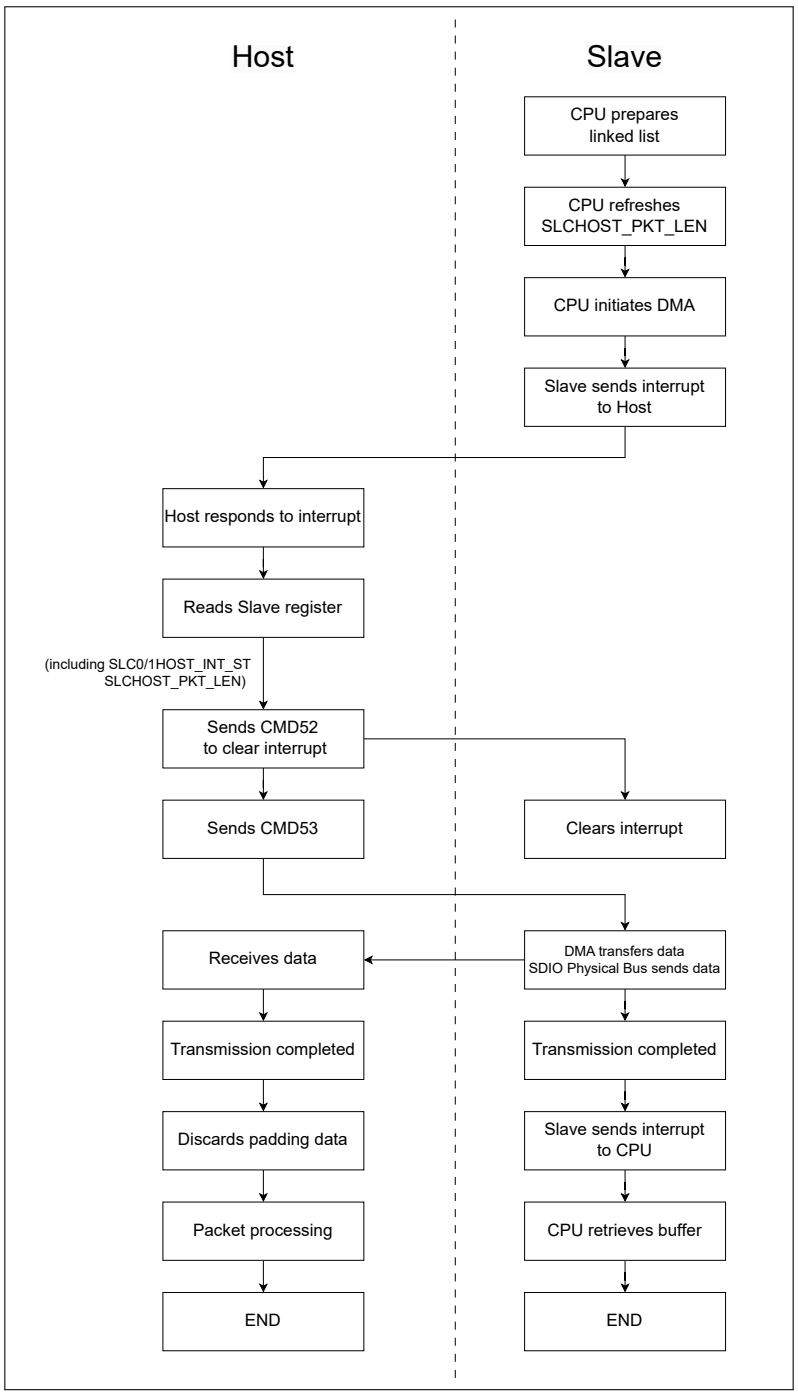



Figure 31.7-1. Procedure of Slave Sending Packets to Host

1. The slave CPU creates the linked list for the data packets to be sent to the host. For details, see Section 31.5.5.1.
2. The slave CPU updates the length of data to be sent using the register SDIO_SLC0_LEN_CONF_REG.
3. The slave CPU starts DMA by writing the 32-bit address of the first descriptor in linked list to SDIO_SLC0RX_LINK_ADDR_REG or SDIO_SLC1RX_LINK_ADDR_REG and then configuring SDIO_SLC0_RXLINK_START or SDIO_SLC1_RXLINK_START to start DMA. For more information on DMA, please refer to Section 31.5.5.
4. The slave DMA sends an interrupt to the host.

5. After the host received the interrupt, it reads from SLCHOST_SLC0HOST_INT_ST_REG, SLCHOST_SLC1HOST_INT_ST_REG, and SLCHOST_PKT_LEN_REG for the following information:
   - SLCHOST_SLC0/1HOST_INT_ST_REG: Interrupt status register. The SLCHOST_SLC0/1_RX_NEW_PACKET_INT_ST bit set to 1 indicates that cause of the interrupt is the slave sending packets.
   - SLCHOST_PKT_LEN_REG: Packet length accumulator register. The current value minus the value of last time equals the packet length sent this time.
6. The host clears the interrupt by writing to the register SLCHOST_SLC0/1HOST_INT_CLR_REG after the CMD52 command.
7. The host fetches packets from the slave after the CMD53 command. During transmission, when the slave determines that the valid data of the current packet is complete, the remaining bits are padded with invalid data (0x0). For details on determining the end of valid data, see Section 31.5.5.3.
8. After the packets are transmitted, the slave DMA sends an interrupt to the CPU, and the CPU can recycle the buffer.

**Notes:**

- Do not set all `eof` bits to 0 in the linked list. Otherwise, the DMA may mistakenly send the next packet's data as part of the current command, causing errors. If all `eof` bits are set to 0, the slave software should align the length of each packet to the data block size by padding data, to prevent errors. When the host sends CMD53 to read data, it should accurately control the number of data blocks in each packet and be able to identify the padded data.
- It is recommended that each CMD53 command transmit only one data packet and each data packet use only one linked list to avoid exceptions caused by complex transmission.
- Avoid sending multiple packets through one linked list, as it complicates the host and software's ability to distinguish between packets. If necessary, set the `eof` bit when creating the linked list to split the packets, so the DMA can pad data accordingly. The length of each packet should be aligned to the data block size to avoid data padding by DMA, and the host should be able to identify the padded data.

### 31.7.2 Receiving Packets from SDIO Host

Transmission of packets from the host to slave is initiated by the host. The slave receives data via DMA and stores it in RAM. After transmission is complete, the CPU is interrupted to process the data. The procedure is shown in Figure 31.7-2.

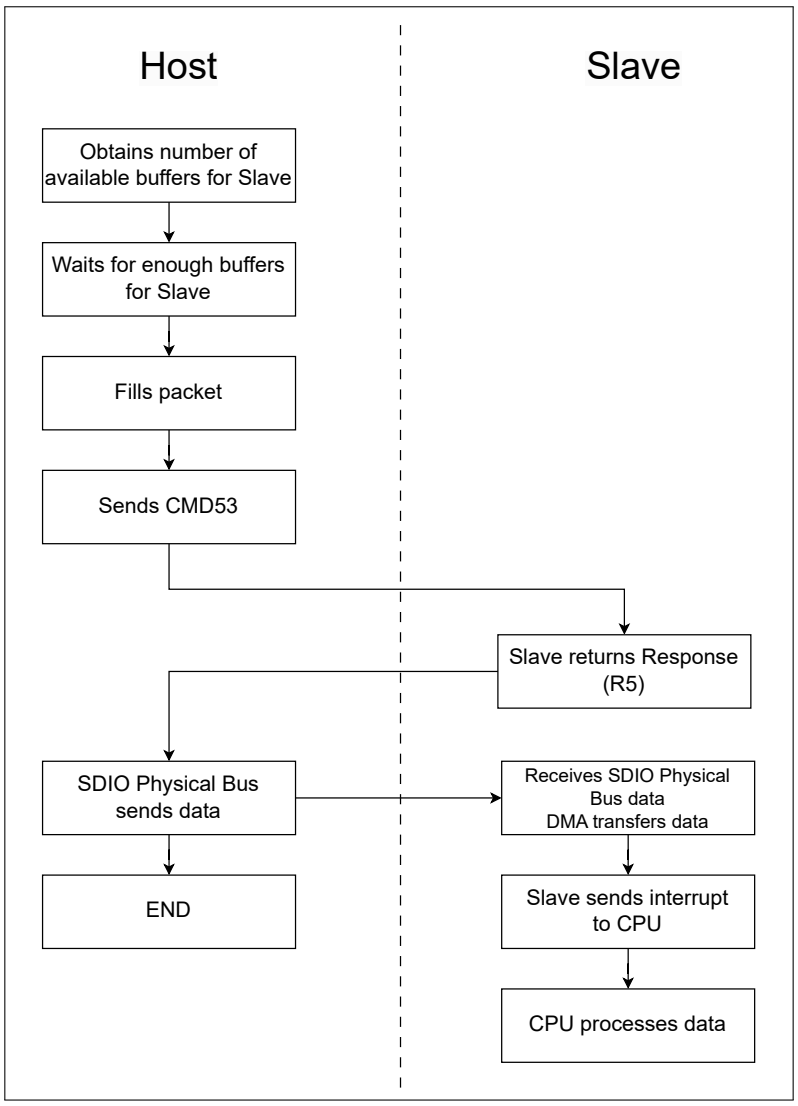



Figure 31.7-2. Procedure of Slave Receiving Packets from Host

The host obtains the number of available receiving buffers from the slave by accessing SLCHOST_SLC0HOST_TOKEN_RDATA_REG or SLCHOST_SLC1HOST_TOKEN_RDATA_REG. The slave CPU should update the value of the register after the receiving DMA linked list is prepared.

SLCHOST_HOSTSLCHOST_SLC0_TOKEN1 or SLCHOST_HOSTSLCHOST_SLC1_TOKEN1 stores the accumulated number of available buffers. The host can determine the available buffer space by subtracting the number of buffers already used from the register value. If buffers are insufficient, the host should poll the register until enough buffers are available.

During packet transmission to the slave through the CMD53 command, when a buffer specified by a linked list descriptor is full or a packet transmission ends, the DMA jumps to the next buffer to store subsequent data. When the slave determines that the valid data of the current packet is complete, the remaining data is considered invalid and discarded. The DMA writes back the current linked list descriptor, sets the `eof` bit of the current descriptor to 1, and generates the SLC0/1_TX_SUC_EOF_INT interrupt.

For more information about DMA functions, linked list, and data discarding, see Section 31.5.5.

To ensure sufficient receiving buffers, the slave CPU must continuously load buffers onto the receiving linked list. The process is shown in Figure 31.7-3.

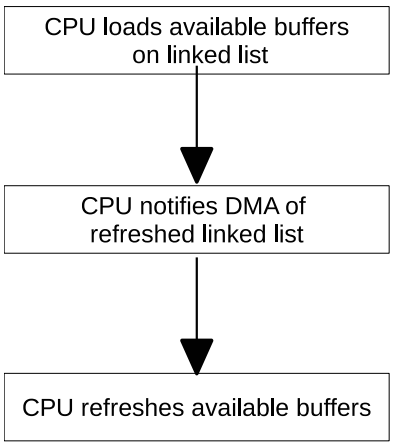



Figure 31.7-3. Loading Receiving Buffer

The CPU should append new buffer segments at the end of the linked list used by DMA and available for receiving data.

The CPU must then notify the DMA that the linked list has been updated. This can be done by setting SDIO_SLC0_TXLINK_RESTART or SDIO_SLC1_TXLINK_RESTART. When the CPU initiates DMA to receive packets for the first time, SDIO_SLC0_TXLINK_START or SDIO_SLC1_TXLINK_START should be set to 1.

**Notes:** Use the *_RESTART field to restart DMA only in the two scenarios:

- DMA is suspended by configuration of the *_STOP field. You can restart it after configuring the *_RESTART field.
- DMA is suspended due to insufficient linked list descriptors. You can restart it by adding descriptors and configuring the *_RESTART field.

Finally, the CPU refreshes available buffer information by writing to the SDIO_SLC0TOKEN1_REG or SDIO_SLC1TOKEN1_REG register.

# 31.8 Register Summary

The addresses in this section are relative to the SDIO Slave Controller base address provided in Table 5.3-2 in Chapter 5 *System and Memory*.

The abbreviations given in the **Access** column are explained in Section *Access Types for Registers*.

## 31.8.1 HINF Register Summary

| Name | Description | Address | Access |
|---|---|---|---|
| **Configuration registers** | | | |
| HINF_CFG_DATA0_REG | SDIO CIS configuration | 0x0000 | R/W |
| HINF_CFG_DATA1_REG | SDIO configuration | 0x0004 | R/W |
| HINF_CFG_DATA7_REG | SDIO configuration | 0x001C | varies |
| HINF_CIS_CONF_W*n*_REG(*n*: 0-7) | SDIO CIS configuration | 0x0020+0x4**n* | R/W |
| HINF_CFG_DATA16_REG | SDIO CIS configuration | 0x0040 | R/W |
| **Status registers** | | | |
| HINF_CONF_STATUS_REG | SDIO CIS function 0 config0 status | 0x0054 | RO |

## 31.8.2 SLC Register Summary

| Name | Description | Address | Access |
|---|---|---|---|
| **Configuration registers** | | | |
| SDIO_SLCCONF0_REG | DMA configuration | 0x0000 | R/W |
| SDIO_SLC0RX_LINK_REG | SCL0 RX linked list configuration | 0x003C | varies |
| SDIO_SLC0RX_LINK_ADDR_REG | SCL0 RX linked list address | 0x0040 | R/W |
| SDIO_SLC0TX_LINK_REG | SCL0 TX linked list configuration | 0x0044 | varies |
| SDIO_SLC0TX_LINK_ADDR_REG | SCL0 TX linked list address | 0x0048 | R/W |
| SDIO_SLC1RX_LINK_REG | SCL1 RX linked list configuration | 0x004C | varies |
| SDIO_SLC1RX_LINK_ADDR_REG | SCL1 RX linked list address | 0x0050 | R/W |
| SDIO_SLC1TX_LINK_REG | SCL1 TX linked list configuration | 0x0054 | varies |
| SDIO_SLC1TX_LINK_ADDR_REG | SCL1 TX linked list address | 0x0058 | R/W |
| SDIO_SLC0TOKEN1_REG | SLC0 receiving buffer configuration | 0x0064 | varies |
| SDIO_SLC1TOKEN1_REG | SLC1 receiving buffer configuration | 0x006C | varies |
| SDIO_SLCCONF1_REG | DMA configuration | 0x0070 | R/W |
| SDIO_SLC_RX_DSCR_CONF_REG | DMA slave to host configuration register | 0x00A8 | R/W |
| SDIO_SLC0_LEN_CONF_REG | Length control of transmitting packets | 0x00F4 | varies |
| SDIO_SLC0_TX_SHAREMEM_START_REG | SLC0 AHB TX start address range | 0x0154 | R/W |
| SDIO_SLC0_TX_SHAREMEM_END_REG | SLC0 AHB TX end address range | 0x0158 | R/W |
| SDIO_SLC0_RX_SHAREMEM_START_REG | SLC0 AHB RX start address range | 0x015C | R/W |
| SDIO_SLC0_RX_SHAREMEM_END_REG | SLC0 AHB RX end address range | 0x0160 | R/W |
| SDIO_SLC1_TX_SHAREMEM_START_REG | SLC1 AHB TX start address range | 0x0164 | R/W |
| SDIO_SLC1_TX_SHAREMEM_END_REG | SLC1 AHB TX end address range | 0x0168 | R/W |
| SDIO_SLC1_RX_SHAREMEM_START_REG | SLC1 AHB RX start address range | 0x016C | R/W |

| Name | Description | Address | Access |
|---|---|---|---|
| SDIO_SLC1_RX_SHAREMEM_END_REG | SLC1 AHB RX end address range | 0x0170 | R/W |
| SDIO_SLC_BURST_LEN_REG | DMA AHB burst type configuration | 0x017C | R/W |
| **Interrupt registers** | | | |
| SDIO_SLCOINT_RAW_REG | SLC0 to slave raw interrupt status | 0x0004 | varies |
| SDIO_SLCOINT_ST_REG | SLC0 to slave masked interrupt status | 0x0008 | RO |
| SDIO_SLCOINT_ENA_REG | SLC0 to slave interrupt enable | 0x000C | R/W |
| SDIO_SLCOINT_CLR_REG | SLC0 to slave interrupt clear | 0x0010 | WT |
| SDIO_SLC1INT_RAW_REG | SLC1 to slave raw interrupt status | 0x0014 | varies |
| SDIO_SLC1INT_CLR_REG | SLC1 to slave interrupt clear | 0x0020 | WT |
| SDIO_SLCINTVEC_TOHOST_REG | Slave to host interrupt vector set | 0x005C | WT |
| SDIO_SLC1INT_ST1_REG | SLC1 to slave masked interrupt status | 0x014C | RO |
| SDIO_SLC1INT_ENA1_REG | SLC1 to slave interrupt enable | 0x0150 | R/W |
| **Status registers** | | | |
| SDIO_SLC0_LENGTH_REG | Length of transmitting packets | 0x00F8 | RO |

### 31.8.3 SLC Host Register Summary

| Name | Description | Address | Access |
|---|---|---|---|
| **Configuration registers** | | | |
| SLCHOST_CONF_REG | Edge configuration | 0x01F0 | R/W |
| **Interrupt registers** | | | |
| SLCHOST_SLC0HOST_INT_RAW_REG | SLC0 to host raw interrupt status | 0x0050 | varies |
| SLCHOST_SLC1HOST_INT_RAW_REG | SLC1 to host raw interrupt status | 0x0054 | varies |
| SLCHOST_SLC0HOST_INT_ST_REG | SLC0 to host masked interrupt status | 0x0058 | RO |
| SLCHOST_SLC1HOST_INT_ST_REG | SLC1 to host masked interrupt status | 0x005C | RO |
| SLCHOST_CONF_W7_REG | Host to slave interrupt vector set | 0x008C | R/W |
| SLCHOST_SLC0HOST_INT_CLR_REG | SLC0 to host interrupt clear | 0x00D4 | WT |
| SLCHOST_SLC1HOST_INT_CLR_REG | SLC1 to host interrupt clear | 0x00D8 | WT |
| SLCHOST_SLC0HOST_FUNC1_INT_ENA_REG | SLC0 to host interrupt enable | 0x00DC | R/W |
| SLCHOST_SLC1HOST_FUNC1_INT_ENA_REG | SLC0 to host interrupt enable | 0x00E0 | R/W |
| **Status registers** | | | |
| SLCHOST_SLC0HOST_TOKEN_RDATA_REG | Accumulated number of SLC0 receiving buffers | 0x0044 | RO |
| SLCHOST_PKT_LEN_REG | Length of the transmitting packets | 0x0060 | RO |
| SLCHOST_SLC1HOST_TOKEN_RDATA_REG | Accumulated number of SLC1 receiving buffers | 0x00C4 | RO |
| **Communication Registers** | | | |
| SLCHOST_CONF_W*n*_REG(*n*: 0-2) | Host and slave communication | 0x006C+0x4**n* | R/W |
| SLCHOST_CONF_W3_REG | Host and slave communication | 0x0078 | R/W |
| SLCHOST_CONF_W4_REG | Host and slave communication | 0x007C | R/W |

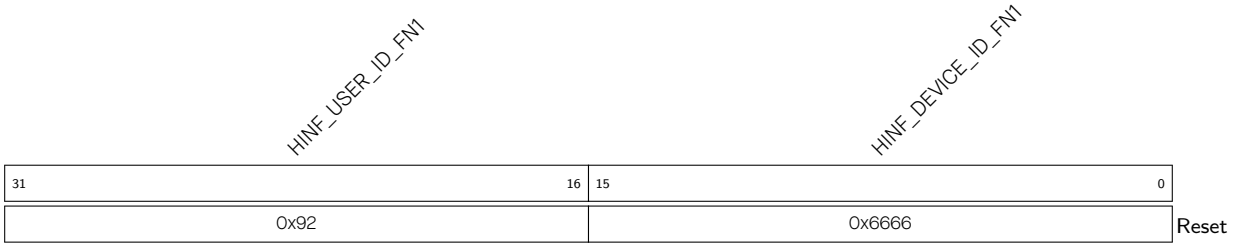
| Name | Description | Address | Access |
|---|---|---|---|
| SLCHOST_CONF_W6_REG | Host and slave communication | 0x0088 | R/W |
| SLCHOST_CONF_W*n*_REG(*n*: 8-15) | Host and slave communication | 0x009C+0x4*(*n*-8) | R/W |

# 31.9 Registers

The addresses in this section are relative to the SDIO Slave Controller base address provided in Table 5.3-2 in Chapter 5 *System and Memory*.

## 31.9.1 HINF Registers

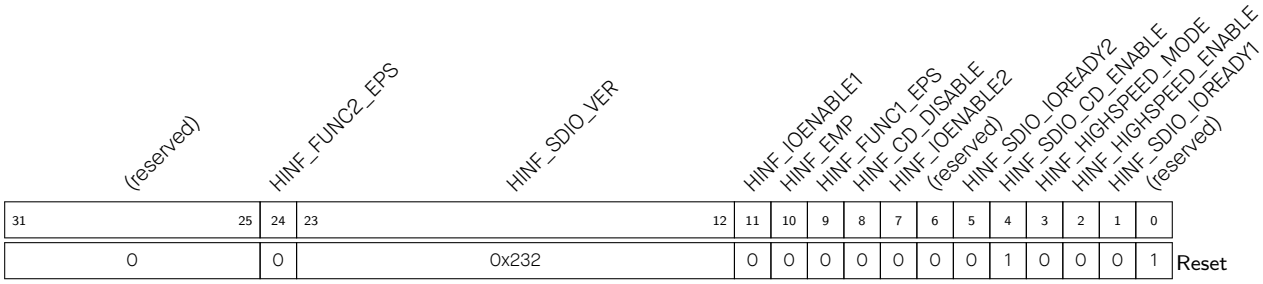
**Register 31.1. HINF_CFG_DATA0_REG (0x0000)**




**HINF_DEVICE_ID_FN1** Configures device ID of function 1 in SDIO CIS. (R/W)

**HINF_USER_ID_FN1** Configures user ID of function 1 in SDIO CIS. (R/W)

Register 31.2. HINF_CFG_DATA1_REG (0x0004)




**HINF_SDIO_IOREADY1** Configures the field IOR1 in SDIO CCCR and the field function 1 ready in SDIO CIS.

0: The function 1 is not ready

1: The function 1 is ready

Please refer to SDIO Specification for details.

(R/W)

**HINF_HIGHSPEED_ENABLE** Configures whether to support SHS in SDIO CCCR.

0: Not support High-Speed mode

1: Support High-Speed mode

Please refer to SDIO Specification for details.

(R/W)

**HINF_HIGHSPEED_MODE** Represents whether EHS status is enabled in SDIO CCCR.

0: Disabled

1: Enabled

Please refer to SDIO Specification for details.

(RO)

**HINF_SDIO_CD_ENABLE** Configures whether to enable SDIO card detection.

0: Disable

1: Enable

(R/W)

**HINF_SDIO_IOREADY2** Configures the field IOR2 in SDIO CCCR and the field function 2 ready in SDIO CIS.

0: The function 2 is not ready

1: The function 2 is ready

Please refer to SDIO Specification for details.

(R/W)

**HINF_IOENABLE2** Represents whether IOE2 is enabled in SDIO CCCR.

0: The function 2 is disabled

1: The function 2 is enabled

Please refer to SDIO Specification for details.

(RO)

**Continued on the next page...**

Register 31.2. HINF_CFG_DATA1_REG (0x0004)

Continued from the previous page...

**HINF_CD_DISABLE** Represents whether CD is disabled in SDIO CCCR.

0: Enabled

1: Disabled

Please refer to SDIO Specification for details.

(RO)

**HINF_FUNC1_EPS** Represents function 1 EPS status in SDIO FBR.

0: The function 1 operates in Higher Current Mode

1: The function 1 works in Lower Current Mode

Please refer to SDIO Specification for details.

(RO)

**HINF_EMP** Represents EMPC status in SDIO CCCR.

0: Master Power Control is disabled

1: Master Power Control is enabled

Please refer to SDIO Specification for details.

(RO)

**HINF_IOENABLE1** Represents IOE1 status in SDIO CCCR.

0: The function 1 is disabled

1: The function 1 is enabled

Please refer to SDIO Specification for details.

(RO)

**HINF_SDIO_VER** Configures SD bit[3:0], SDIO bit[3:0], CCCR bit[3:0] in SDIO CCCR.

HINF_SDIO_VER[11:8] mapping to SD bit[3:0]

HINF_SDIO_VER[7:4] mapping to SDIO bit[3:0]

HINF_SDIO_VER[3:0] mapping to CCCR bit[3:0]

Please refer to SDIO Specification for details.

(R/W)

**HINF_FUNC2_EPS** Represents function 2 EPS status in SDIO FBR.

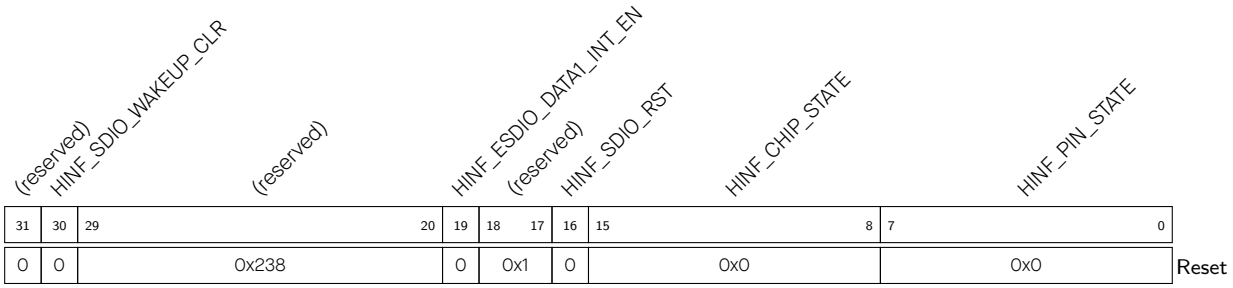
0: The function 2 operates in Higher Current Mode

1: The function 2 works in Lower Current Mode

Please refer to SDIO Specification for details.

(RO)

Register 31.3. HINF_CFG_DATA7_REG (0x001C)




**HINF_PIN_STATE** Configures SDIO CIS address 318 and 574. Please refer to SDIO Specification for details. (R/W)

**HINF_CHIP_STATE** Configures SDIO CIS address 312, 315, 568, and 571. Please refer to SDIO Specification for details. (R/W)

**HINF_SDIO_RST** Configures whether to reset the SDIO slave module.

0: No effect

1: Reset

(R/W)

**HINF_ESDIO_DATA1_INT_EN** Configures whether to enable SDIO interrupt on data1 line.

0: Disable

1: Enable

(R/W)

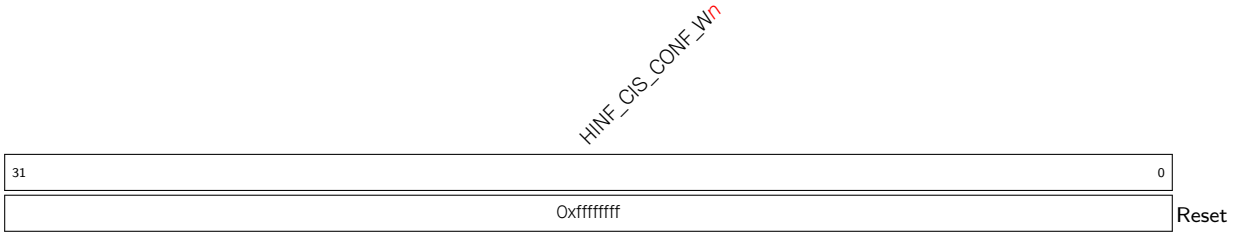
**HINF_SDIO_WAKEUP_CLR** Configures whether to clear wake up signal after the chip is waken up by the SDIO slave.

0: No effect

1: Clear

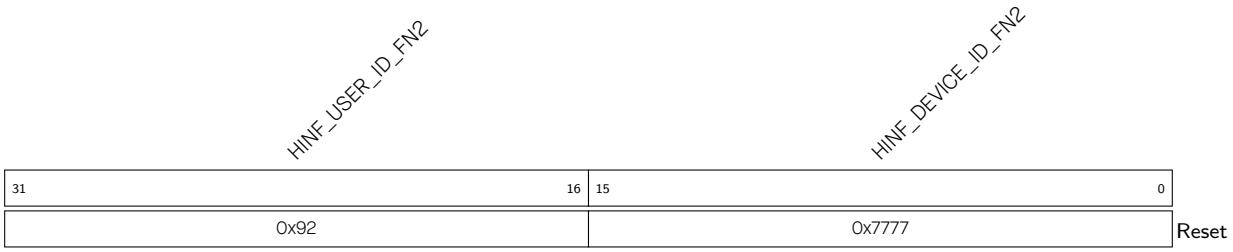
(WT)

Register 31.4. HINF_CIS_CONF_W*n*_REG(*n*: 0-7) (0x0020+0x4**n*)




**HINF_CIS_CONF_W*n*** Configures SDIO CIS address (39+4***n*) ~ (36+4***n*). Please refer to SDIO Specification for details. (R/W)

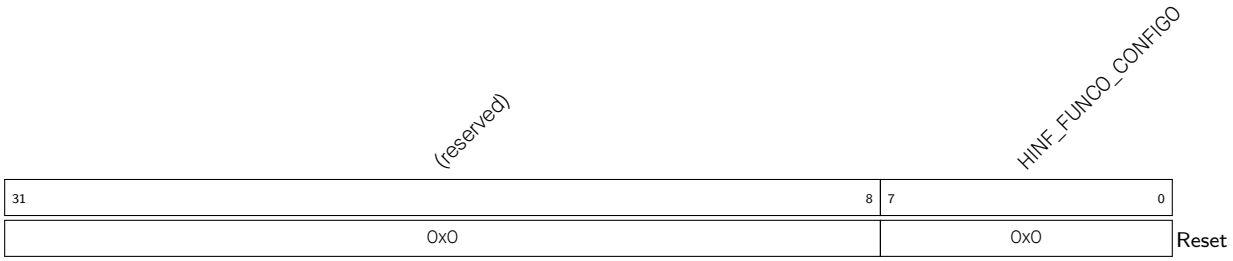
Register 31.5. HINF_CFG_DATA16_REG (0x0040)




**HINF_DEVICE_ID_FN2** Configures device ID of function 2 in SDIO CIS. (R/W)

**HINF_USER_ID_FN2** Configures user ID of function 2 in SDIO CIS. (R/W)

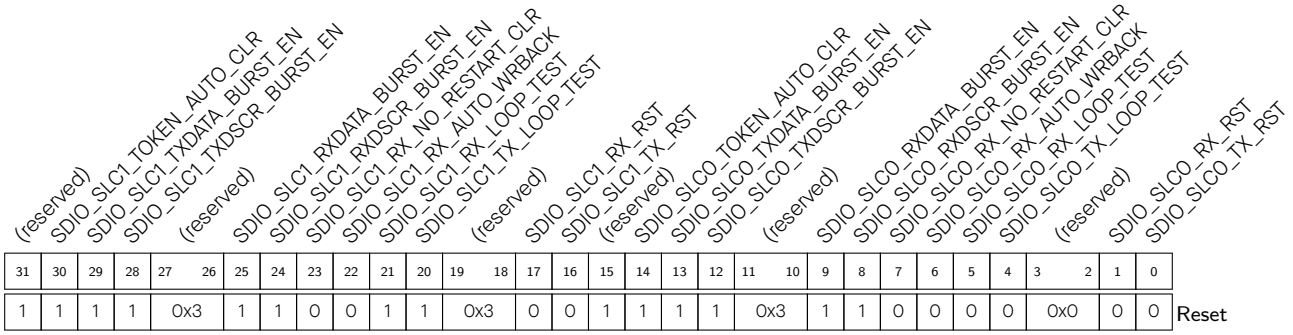
Register 31.6. HINF_CONF_STATUS_REG (0x0054)




**HINF_FUNC0_CONFIG0** Represents SDIO CIS function 0 config0 (addr: 0x20f0) status. Please refer to SDIO Specification for details. (RO)

## 31.9.2 SLC Registers

Register 31.7. SDIO_SLCCONF0_REG (0x0000)




**SDIO_SLC0_TX_RST** Configures whether to reset TX (host to slave) FSM (finite state machine) in SLC0.

0: No effect

1: Reset

(R/W)

**SDIO_SLC0_RX_RST** Configures whether to reset RX (slave to host) FSM in SCL0.

0: No effect

1: Reset

(R/W)

**SDIO_SLC0_TX_LOOP_TEST** Configures whether SCL0 loops around when the slave buffer finishes receiving packets from the host.

0: Not loop around

1: Loop around, and hardware will not change the owner bit in the linked list

(R/W)

**SDIO_SLC0_RX_LOOP_TEST** Configures whether SCL0 loops around when the slave buffer finishes sending packets to the host.

0: Not loop around

1: Loop around, and hardware will not change the owner bit in the linked list

(R/W)

**SDIO_SLC0_RX_AUTO_WRBACK** Configures whether SCL0 changes the owner bit of RX linked list.

0: Not change

1: Change

(R/W)

**SDIO_SLC0_RX_NO_RESTART_CLR** Please initialize to 1, and do not modify it. (R/W)

**SDIO_SLC0_RXDSCR_BURST_EN** Configures whether SCL0 can use AHB burst operation when reading the RX linked list from memory.

0: Only use single operation

1: Can use burst operation

(R/W)

**Continued on the next page...**

Register 31.7. SDIO_SLCCONF0_REG (0x0000)

Continued from the previous page...

**SDIO_SLC0_RXDATA_BURST_EN** Configures whether SCL0 can use AHB burst operation when read data from memory.
0: Only use single operation
1: Can use burst operation
(R/W)

**SDIO_SLC0_TXDSCR_BURST_EN** Configures whether SCL0 can use AHB burst operation when read the TX linked list from memory.
0: Only use single operation
1: Can use burst operation
(R/W)

**SDIO_SLC0_TXDATA_BURST_EN** Configures whether SCL0 can use AHB burst operation when send data to memory.
0: Only use single operation
1: Can use burst operation
(R/W)

**SDIO_SLC0_TOKEN_AUTO_CLR** Please initialize to 0, and do not modify it. (R/W)

**SDIO_SLC1_TX_RST** Configures whether to reset TX FSM in SLC1.
0: No effect
1: Reset
(R/W)

**SDIO_SLC1_RX_RST** Configures whether to reset RX FSM in SLC1.
0: No effect
1: Reset
(R/W)

**SDIO_SLC1_TX_LOOP_TEST** Configures whether SCL1 loops around when the slave buffer finishes receiving packets from the host.
0: Not loop around
1: Loop around, and hardware will not change the owner bit in the linked list
(R/W)

**SDIO_SLC1_RX_LOOP_TEST** Configures whether SCL1 loops around when the slave buffer finishes sending packets to the host.
0: Not loop around
1: Loop around, and hardware will not change the owner bit in the linked list
(R/W)

Continued on the next page...

Register 31.7. SDIO_SLCCONF0_REG (0x0000)

Continued from the previous page...

**SDIO_SLC1_RX_AUTO_WRBACK** Configures whether SCL1 changes the owner bit of the RX linked list.
0: Not change
1: Change
(R/W)

**SDIO_SLC1_RX_NO_RESTART_CLR** Please initialize to 1, and do not modify it. (R/W)

**SDIO_SLC1_RXDSCR_BURST_EN** Configures whether SCL1 can use AHB burst operation when read the RX linked list from memory.
0: Only use single operation
1: Can use burst operation
(R/W)

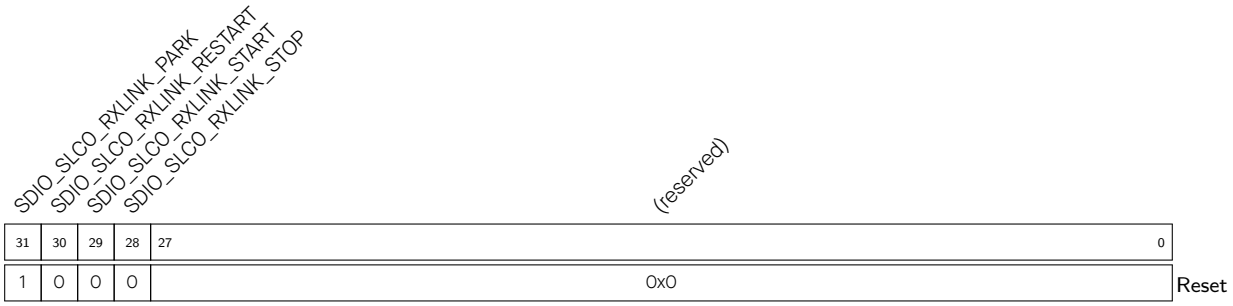
**SDIO_SLC1_RXDATA_BURST_EN** Configures whether SCL1 can use AHB burst operation when reading data from memory.
0: Only use single operation
1: Can use burst operation
(R/W)

**SDIO_SLC1_TXDSCR_BURST_EN** Configures whether SCL1 can use AHB burst operation when read the TX linked list from memory.
0: Only use single operation
1: Can use burst operation
(R/W)

**SDIO_SLC1_TXDATA_BURST_EN** Configures whether SCL1 can use AHB burst operation when send data to memory.
0: Only use single operation
1: Can use burst operation
(R/W)

**SDIO_SLC1_TOKEN_AUTO_CLR** Please initialize to 0, and do not modify it. (R/W)

Register 31.8. SDIO_SLC0RX_LINK_REG (0x003C)




**SDIO_SLC0_RXLINK_STOP** Configures whether to stop SLC0 RX linked list operation.

0: No effect

1: Stop the operation

(R/W/SC)

**SDIO_SLC0_RXLINK_START** Configures whether to start SLC0 RX linked list operation from the address indicated by SDIO_SLC0_RXLINK_ADDR.

0: No effect

1: Start the operation

(R/W/SC)

**SDIO_SLC0_RXLINK_RESTART** Configures whether to restart and continue SLC0 RX linked list operation.

0: No effect

1: Restart the operation

(R/W/SC)

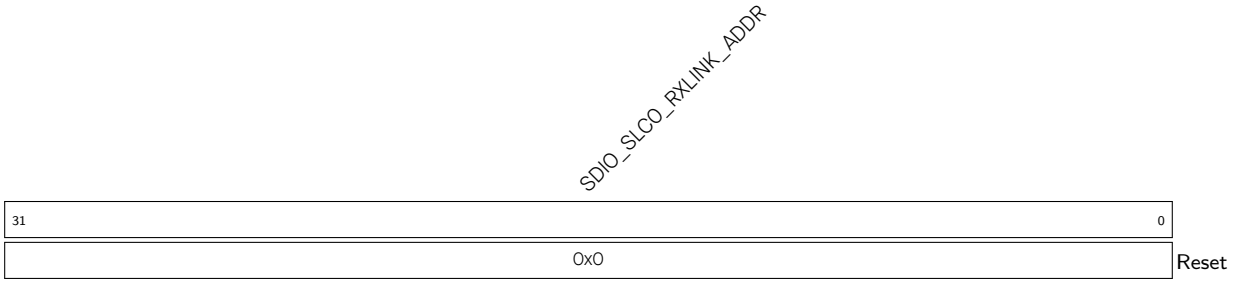
**SDIO_SLC0_RXLINK_PARK** Represents SLC0 RX linked list FSM state.

0: The FSM not in idle state

1: The FSM in idle state

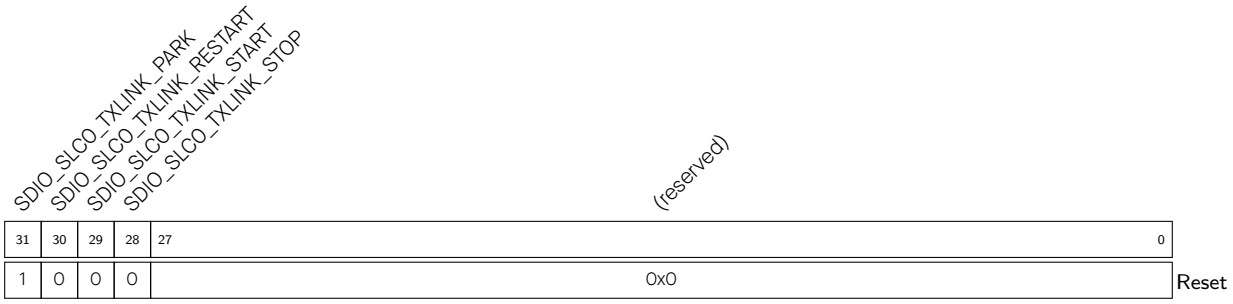
(RO)

Register 31.9. SDIO_SLC0RX_LINK_ADDR_REG (0x0040)




**SDIO_SLC0_RXLINK_ADDR** Configures SLC0 RX linked list initial address. (R/W)

Register 31.10. SDIO_SLC0TX_LINK_REG (0x0044)




**SDIO_SLC0_TXLINK_STOP** Configures whether to stop SLC0 TX linked list operation.

0: No effect

1: Stop the operation

(R/W/SC)

**SDIO_SLC0_TXLINK_START** Configures whether to start SLC0 TX linked list operation from the address indicated by SDIO_SLC0_TXLINK_ADDR.

0: No effect

1: Start the operation

(R/W/SC)

**SDIO_SLC0_TXLINK_RESTART** Configures whether to restart and continue SLC0 TX linked list operation.

0: No effect

1: Restart the operation

(R/W/SC)

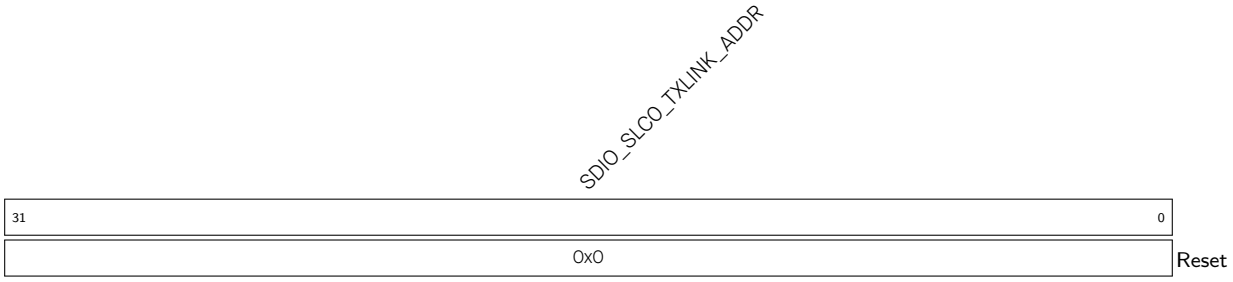
**SDIO_SLC0_TXLINK_PARK** Represents SLC0 TX linked list FSM state.

0: The FSM not in idle state

1: The FSM in idle state

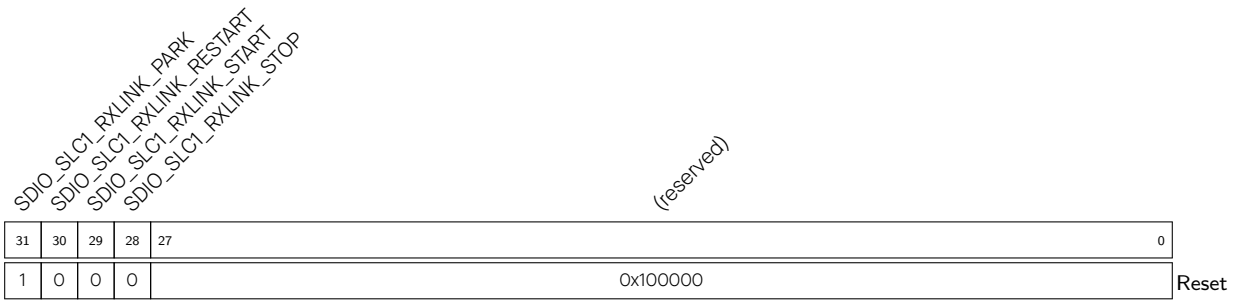
(RO)

Register 31.11. SDIO_SLC0TX_LINK_ADDR_REG (0x0048)




**SDIO_SLC0_TXLINK_ADDR** Configures SLC0 TX linked list initial address. (R/W)

Register 31.12. SDIO_SLC1RX_LINK_REG (0x004C)




**SDIO_SLC1_RXLINK_STOP** Configures whether to stop SLC1 RX linked list operation.

0: No effect

1: Stop the operation

(R/W/SC)

**SDIO_SLC1_RXLINK_START** Configures whether to start SLC1 RX linked list operation from the address indicated by SDIO_SLC1_RXLINK_ADDR.

0: No effect

1: Start the operation

(R/W/SC)

**SDIO_SLC1_RXLINK_RESTART** Configures whether to restart and continue SLC1 RX linked list operation.

0: No effect

1: Restart the operation

(R/W/SC)

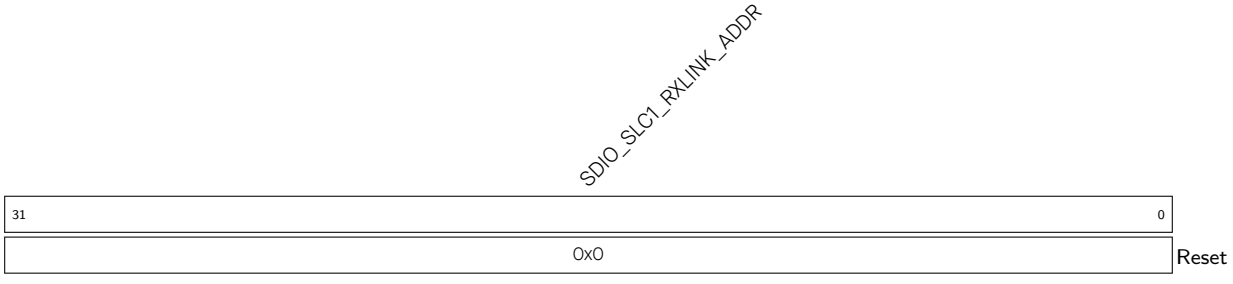
**SDIO_SLC1_RXLINK_PARK** Represents SLC1 RX linked list FSM state.

0: The FSM not in idle state

1: The FSM in idle state

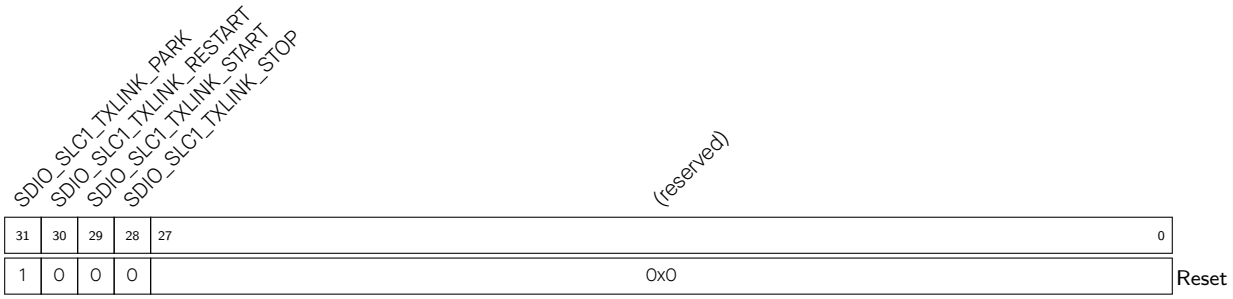
(RO)

Register 31.13. SDIO_SLC1RX_LINK_ADDR_REG (0x0050)




**SDIO_SLC1_RXLINK_ADDR** Configures SLC1 RX linked list initial address. (R/W)

Register 31.14. SDIO_SLC1TX_LINK_REG (0x0054)




**SDIO_SLC1_TXLINK_STOP** Configures whether to stop SLC1 TX linked list operation.

0: No effect

1: Stop the operation

(R/W/SC)

**SDIO_SLC1_TXLINK_START** Configures whether to start SLC1 TX linked list operation from the address indicated by SDIO_SLC1_TXLINK_ADDR.

0: No effect

1: Start the operation

(R/W/SC)

**SDIO_SLC1_TXLINK_RESTART** Configures whether to restart and continue SLC1 TX linked list operation.

0: No effect

1: Restart the operation

(R/W/SC)

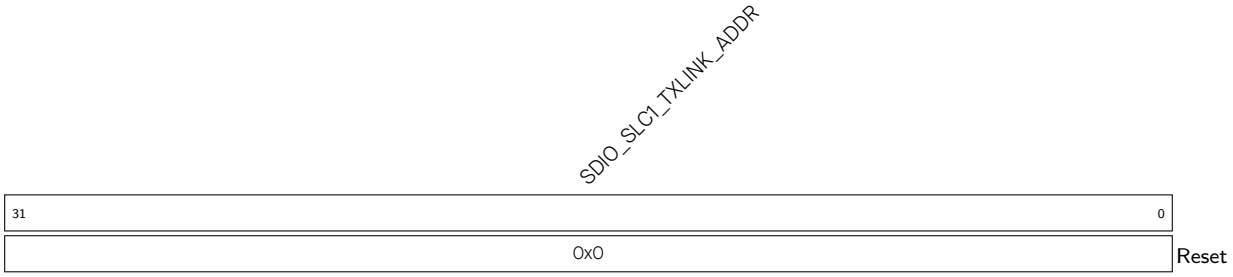
**SDIO_SLC1_TXLINK_PARK** Represents SLC1 TX linked list FSM state.

0: The FSM not in idle state

1: The FSM in idle state

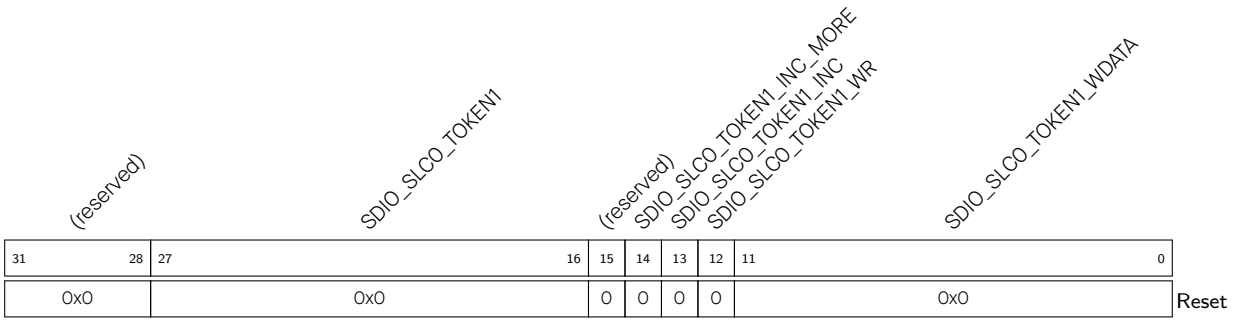
(RO)

Register 31.15. SDIO_SLC1TX_LINK_ADDR_REG (0x0058)




**SDIO_SLC1_TXLINK_ADDR** Configures SLC1 TX linked list initial address. (R/W)

Register 31.16. SDIO_SLC0TOKEN1_REG (0x0064)




**SDIO_SLC0_TOKEN1_WDATA** Configures SLC0 token 1 value. (WT)

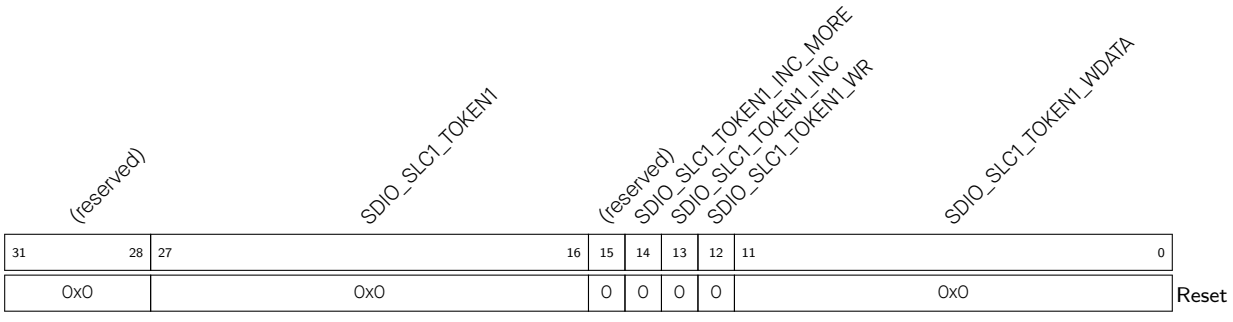
**SDIO_SLC0_TOKEN1_WR** Configures this bit to 1 to write SDIO_SLC0_TOKEN1_WDATA into SDIO_SLC0_TOKEN1. (WT)

**SDIO_SLC0_TOKEN1_INC** Configures this bit to 1 to add 1 to SDIO_SLC0_TOKEN1. (WT)

**SDIO_SLC0_TOKEN1_INC_MORE** Configures this bit to 1 to add the value of SDIO_SLC0_TOKEN1_WDATA to SDIO_SLC0_TOKEN1. (WT)

**SDIO_SLC0_TOKEN1** Represents the SLC0 accumulated number of buffers for receiving packets. (RO)

Register 31.17. SDIO_SLC1TOKEN1_REG (0x006C)




**SDIO_SLC1_TOKEN1_WDATA** Configures SLC1 token1 value. (WT)

**SDIO_SLC1_TOKEN1_WR** Configures this bit to 1 to write SDIO_SLC1_TOKEN1_WDATA into SDIO_SLC1_TOKEN1. (WT)

**SDIO_SLC1_TOKEN1_INC** Configures this bit to 1 to add 1 to SDIO_SLC1_TOKEN1. (WT)

**SDIO_SLC1_TOKEN1_INC_MORE** Configures this bit to 1 to add the value of SDIO_SLC1_TOKEN1_WDATA to SDIO_SLC1_TOKEN1. (WT)

**SDIO_SLC1_TOKEN1** Represents SLC1 accumulated number of buffers for receiving packets. (RO)

Register 31.18. SDIO_SLCCONF1_REG (0x0070)




**SDIO_SDIO_CMD_HOLD_EN** Please initialize to 0, and do not modify it. (R/W)

**SDIO_SLC0_LEN_AUTO_CLR** Please initialize to 0, and do not modify it. (R/W)

**SDIO_SLC0_TX_STITCH_EN** Please initialize to 0, and do not modify it. (R/W)

**SDIO_SLC0_RX_STITCH_EN** Please initialize to 0, and do not modify it. (R/W)

**SDIO_HOST_INT_LEVEL_SEL** Configures the polarity of interrupt to host.

0: Low active

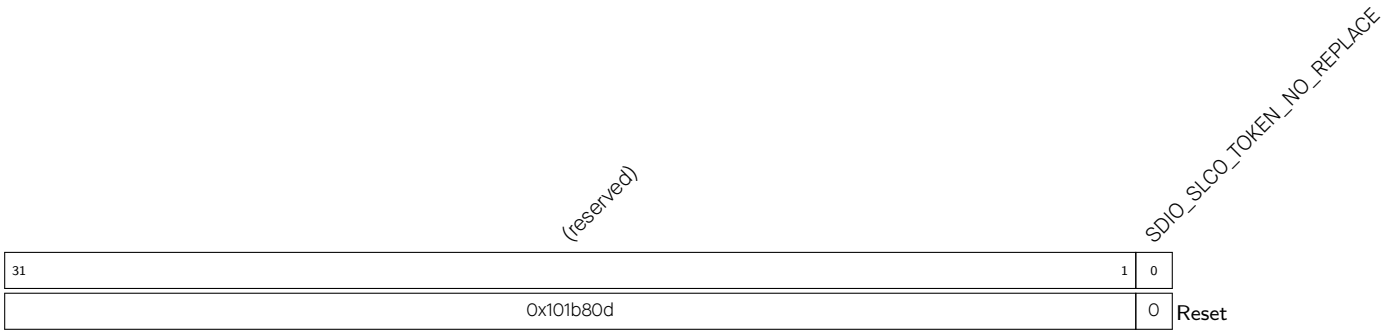
1: High active

(R/W)

**SDIO_SLC1_TX_STITCH_EN** Please initialize to 0, and do not modify it. (R/W)

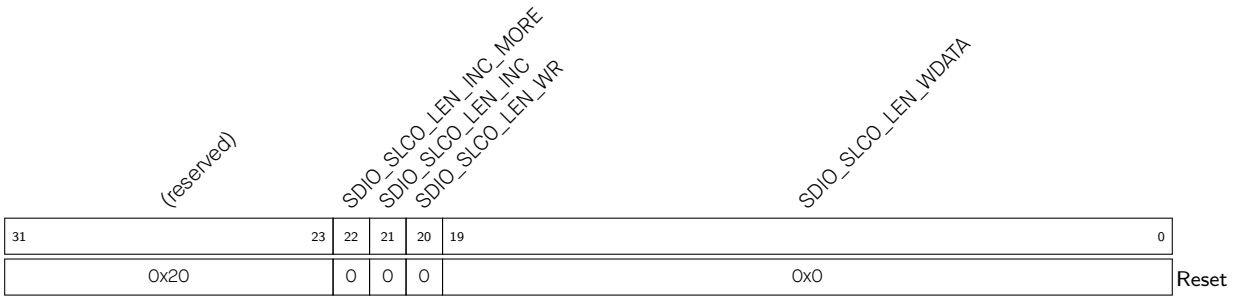
**SDIO_SLC1_RX_STITCH_EN** Please initialize to 0, and do not modify it. (R/W)

Register 31.19. SDIO_SLC_RX_DSCR_CONF_REG (0x00A8)




**SDIO_SLC0_TOKEN_NO_REPLACE** Please initialize to 1, and do not modify it. (R/W)

Register 31.20. SDIO_SLCO_LEN_CONF_REG (0x00F4)



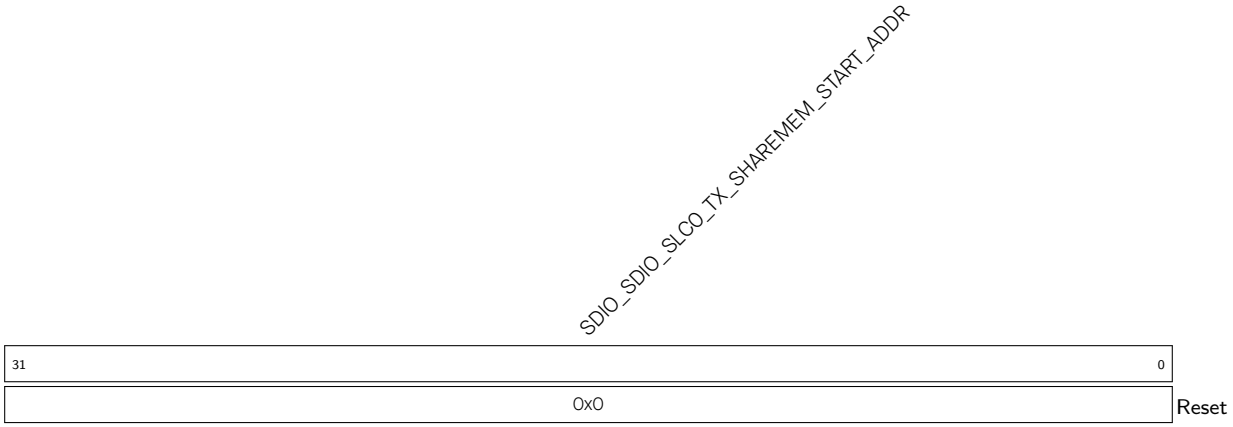

**SDIO_SLCO_LEN_WDATA** Configures the length of the data that the slave wants to send. (WT)

**SDIO_SLCO_LEN_WR** Configures this bit to 1 to write SDIO_SLCO_LEN_WDATA into SDIO_SLCO_LEN and SLCHOST_HOSTSLCHOST_SLCO_LEN. (WT)

**SDIO_SLCO_LEN_INC** Configures this bit to 1 to add 1 to SDIO_SLCO_LEN and SLCHOST_HOSTSLCHOST_SLCO_LEN. (WT)

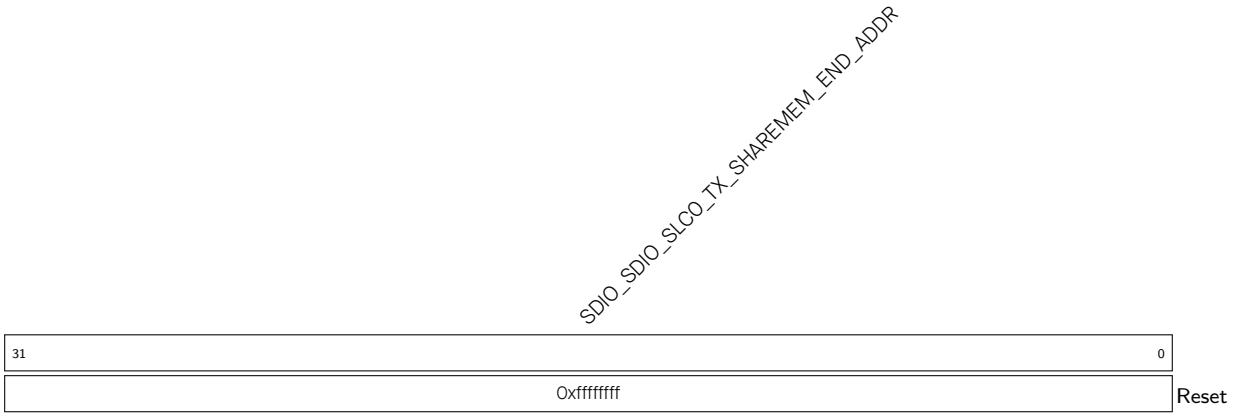
**SDIO_SLCO_LEN_INC_MORE** Configures this bit to 1 to add the value of SDIO_SLCO_LEN_WDATA to SDIO_SLCO_LEN and SLCHOST_HOSTSLCHOST_SLCO_LEN. (WT)

Register 31.21. SDIO_SLCO_TX_SHAREMEM_START_REG (0x0154)



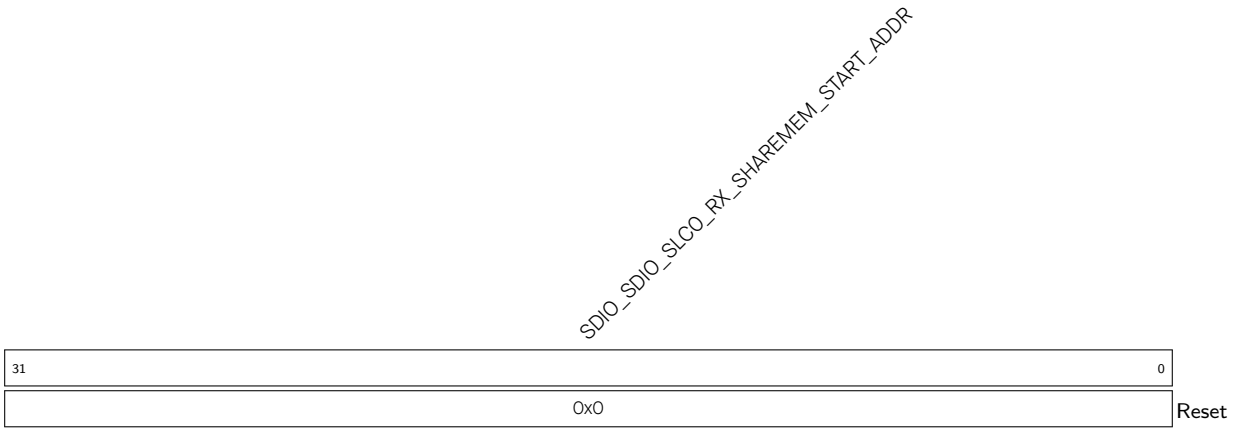

**SDIO_SDIO_SLCO_TX_SHAREMEM_START_ADDR** Configures SLC0 host to slave channel AHB start address boundary. (R/W)

Register 31.22. SDIO_SLC0_TX_SHAREMEM_END_REG (0x0158)



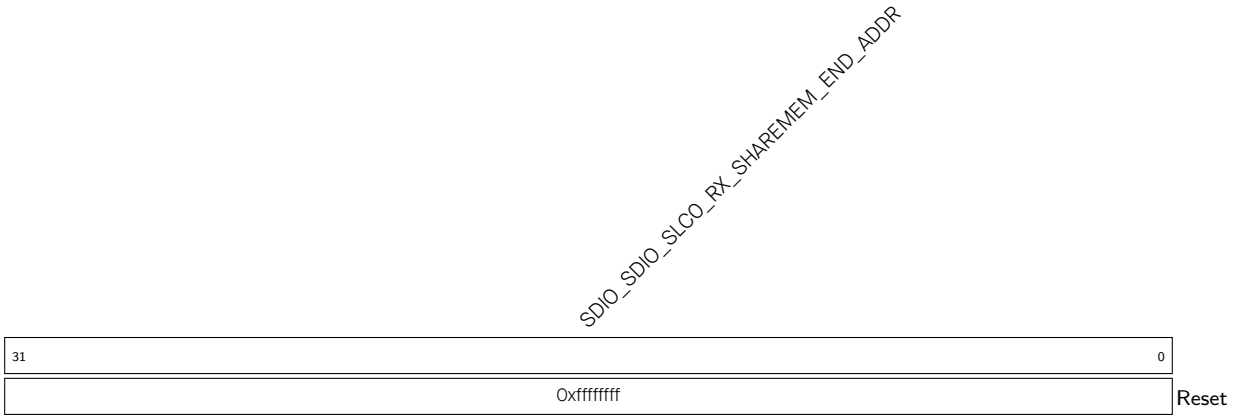

**SDIO_SDIO_SLC0_TX_SHAREMEM_END_ADDR** Configures SLC0 host to slave channel AHB end address boundary. (R/W)

Register 31.23. SDIO_SLC0_RX_SHAREMEM_START_REG (0x015C)



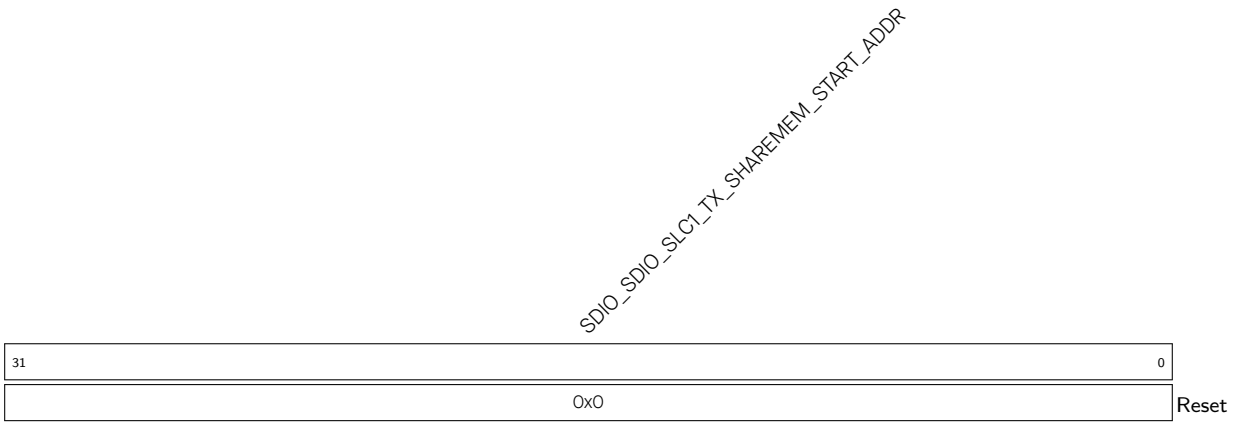

**SDIO_SDIO_SLC0_RX_SHAREMEM_START_ADDR** Configures SLC0 slave to host channel AHB start address boundary. (R/W)

Register 31.24. SDIO_SLC0_RX_SHAREMEM_END_REG (0x0160)



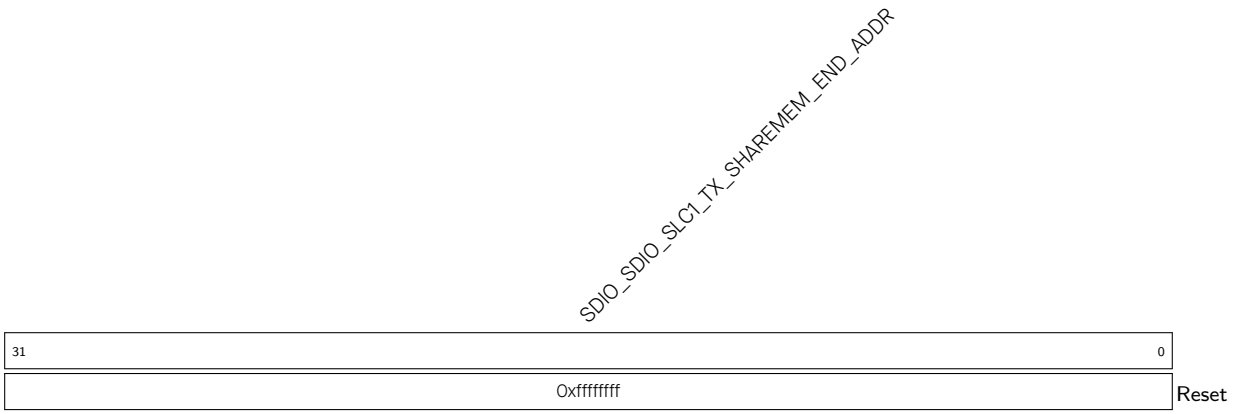

**SDIO_SDIO_SLC0_RX_SHAREMEM_END_ADDR** Configures SLC0 slave to host channel AHB end address boundary. (R/W)

Register 31.25. SDIO_SLC1_TX_SHAREMEM_START_REG (0x0164)



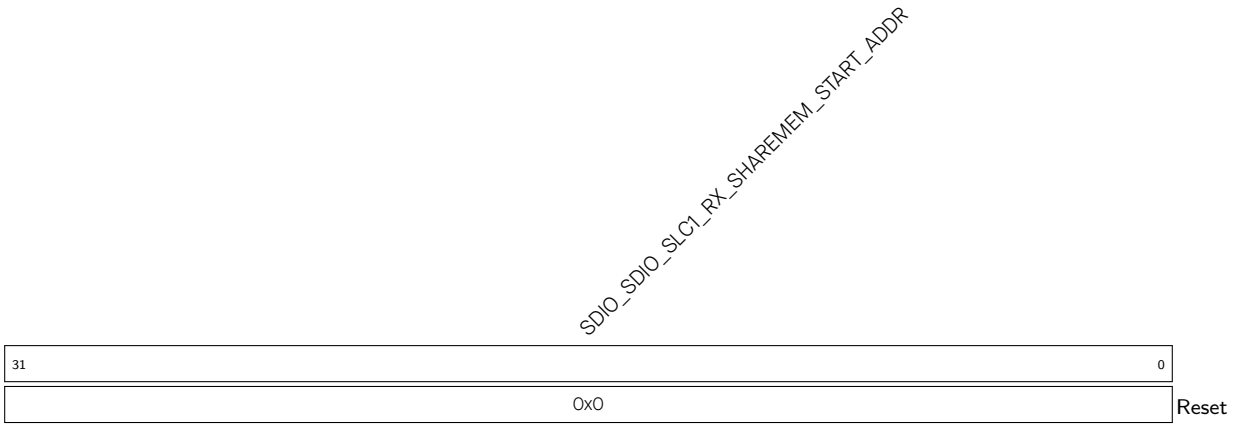

**SDIO_SDIO_SLC1_TX_SHAREMEM_START_ADDR** Configures SLC1 host to slave channel AHB start address boundary. (R/W)

Register 31.26. SDIO_SLC1_TX_SHAREMEM_END_REG (0x0168)



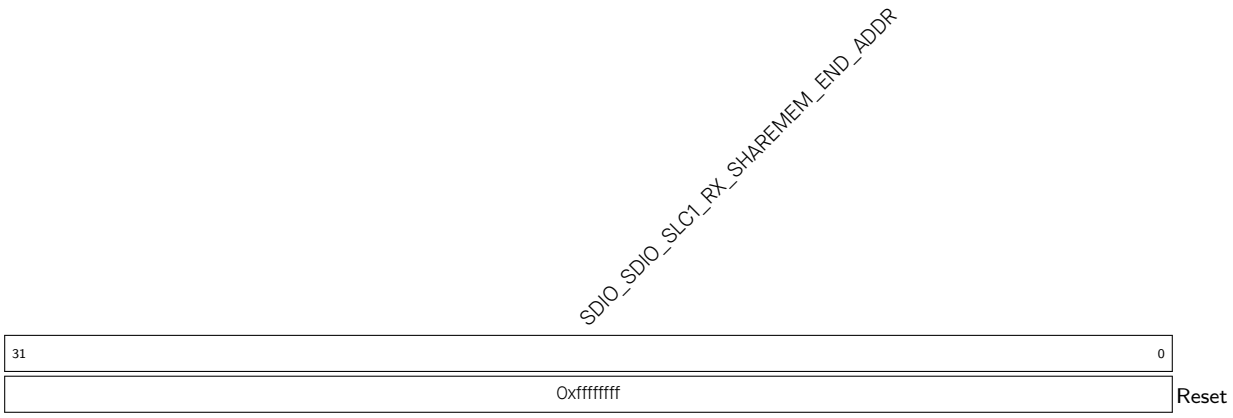

**SDIO_SDIO_SLC1_TX_SHAREMEM_END_ADDR** Configures SLC1 host to slave channel AHB end address boundary. (R/W)

Register 31.27. SDIO_SLC1_RX_SHAREMEM_START_REG (0x016C)




**SDIO_SDIO_SLC1_RX_SHAREMEM_START_ADDR** Configures SLC1 slave to host channel AHB start address boundary. (R/W)

Register 31.28. SDIO_SLC1_RX_SHAREMEM_END_REG (0x0170)




**SDIO_SDIO_SLC1_RX_SHAREMEM_END_ADDR** Configures SLC1 slave to host channel AHB end address boundary. (R/W)

Register 31.29. SDIO_SLC_BURST_LEN_REG (0x017C)




**SDIO_SLC0_TXDATA_BURST_LEN** Configures SLC0 host to slave channel AHB burst type.

0: Can use incr4

1: Can use incr8

(R/W)

**SDIO_SLC0_RXDATA_BURST_LEN** Configures SLC0 slave to host channel AHB burst type.

0: Can use incr and incr4

1: Can use incr and incr8

(R/W)

**SDIO_SLC1_TXDATA_BURST_LEN** Configures SLC1 host to slave channel AHB burst type.

0: Can use incr4

1: Can use incr8

(R/W)

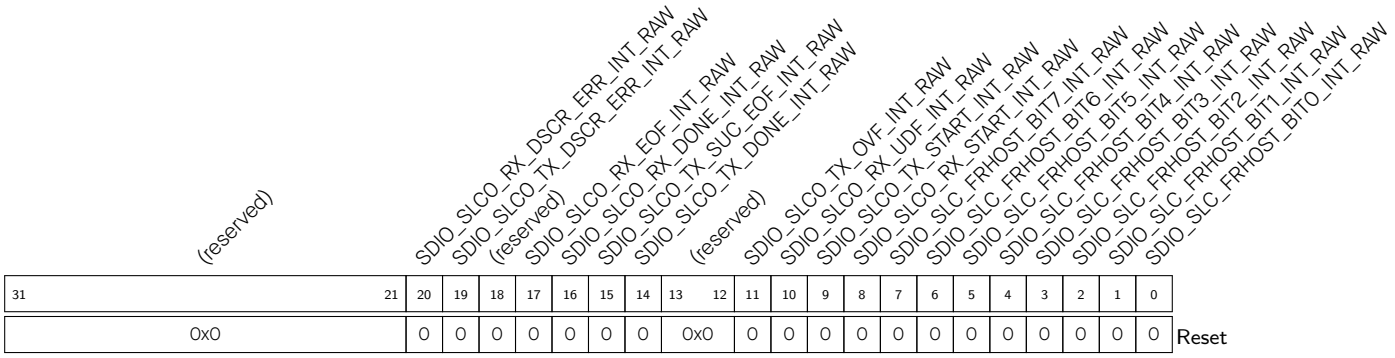
**SDIO_SLC1_RXDATA_BURST_LEN** Configures SLC1 slave to host channel AHB burst type.

0: Can use incr and incr4

1: Can use incr and incr8

(R/W)

Register 31.30. SDIO_SLCOINT_RAW_REG (0x0004)




**SDIO_SLC_FRHOST_BIT*n*_INT_RAW (*n*: 0-7)** The raw interrupt status of SLC_FRHOST_BIT*n*_INT (*n*: 0-7). (R/WTC/SS)

**SDIO_SLCO_RX_START_INT_RAW** The raw interrupt status of SLCO_RX_START_INT. (R/WTC/SS)

**SDIO_SLCO_TX_START_INT_RAW** The raw interrupt status of SLCO_TX_START_INT. (R/WTC/SS)

**SDIO_SLCO_RX_UDF_INT_RAW** The raw interrupt status of SLCO_RX_UDF_INT. (R/WTC/SS)

**SDIO_SLCO_TX_OVF_INT_RAW** The raw interrupt status of SLCO_TX_OVF_INT. (R/WTC/SS)

**SDIO_SLCO_TX_DONE_INT_RAW** The raw interrupt status of SLCO_TX_DONE_INT. (R/WTC/SS)

**SDIO_SLCO_TX_SUC_EOF_INT_RAW** The raw interrupt status of SLCO_TX_SUC_EOF_INT. (R/WTC/SS)

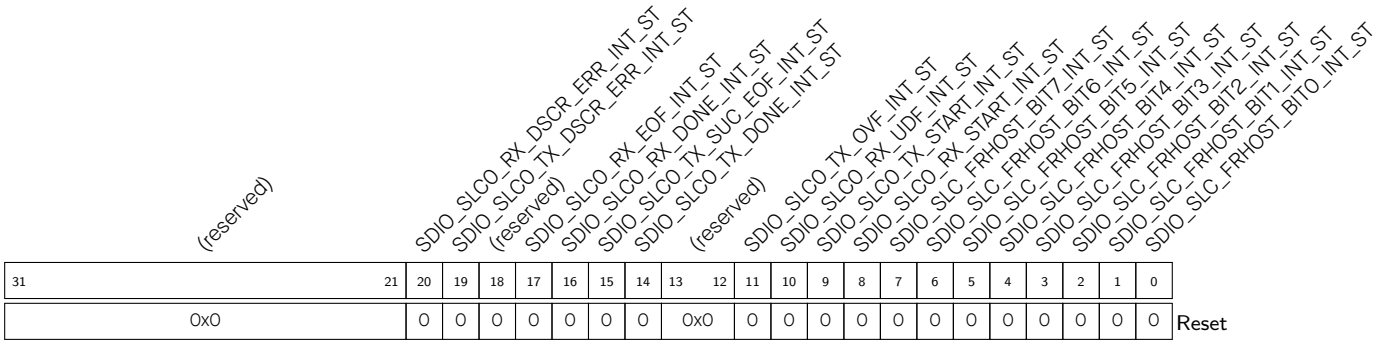
**SDIO_SLCO_RX_DONE_INT_RAW** The raw interrupt status of SLCO_RX_DONE_INT. (R/WTC/SS)

**SDIO_SLCO_RX_EOF_INT_RAW** The raw interrupt status of SLCO_RX_EOF_INT. (R/WTC/SS)

**SDIO_SLCO_TX_DSCR_ERR_INT_RAW** The raw interrupt status of SLCO_TX_DSCR_ERR_INT. (R/WTC/SS)

**SDIO_SLCO_RX_DSCR_ERR_INT_RAW** The raw interrupt status of SLCO_RX_DSCR_ERR_INT. (R/WTC/SS)

Register 31.31. SDIO_SLC0INT_ST_REG (0x0008)




**SDIO_SLC_FRHOST_BIT*n*_INT_ST (*n*: 0-7)** The masked interrupt status of SLC_FRHOST_BIT*n*_INT (*n*: 0-7). (RO)

**SDIO_SLC0_RX_START_INT_ST** The masked interrupt status of SLC0_RX_START_INT. (RO)

**SDIO_SLC0_TX_START_INT_ST** The masked interrupt status bit of SLC0_TX_START_INT. (RO)

**SDIO_SLC0_RX_UDF_INT_ST** The masked interrupt status of SLC0_RX_UDF_INT. (RO)

**SDIO_SLC0_TX_OVF_INT_ST** The masked interrupt status of SLC0_TX_OVF_INT. (RO)

**SDIO_SLC0_TX_DONE_INT_ST** The masked interrupt status of SLC0_TX_DONE_INT. (RO)

**SDIO_SLC0_TX_SUC_EOF_INT_ST** The masked interrupt status of SLC0_TX_SUC_EOF_INT. (RO)

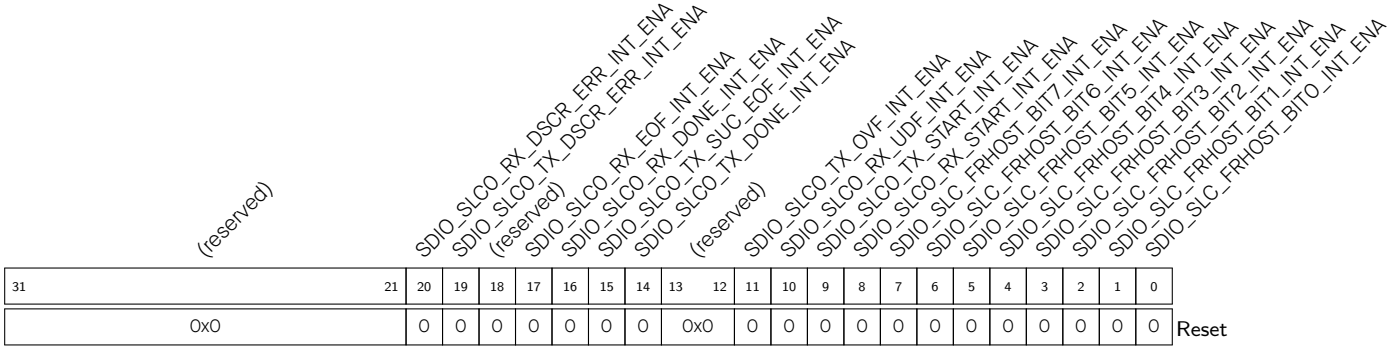
**SDIO_SLC0_RX_DONE_INT_ST** The masked interrupt status of SLC0_RX_DONE_INT. (RO)

**SDIO_SLC0_RX_EOF_INT_ST** The masked interrupt status bit of SLC0_RX_EOF_INT. (RO)

**SDIO_SLC0_TX_DSCR_ERR_INT_ST** The masked interrupt status of SLC0_TX_DSCR_ERR_INT. (RO)

**SDIO_SLC0_RX_DSCR_ERR_INT_ST** The masked interrupt status of SLC0_RX_DSCR_ERR_INT. (RO)

Register 31.32. SDIO_SLCOINT_ENA_REG (0x000C)




**SDIO_SLC_FRHOST_BIT*n*_INT_ENA (*n*: 0-7)** Write 1 to enable interrupt SLC_FRHOST_BIT*n*_INT (*n*: 0-7). (R/W)

**SDIO_SLCO_RX_START_INT_ENA** Write 1 to enable interrupt SLCO_RX_START_INT. (R/W)

**SDIO_SLCO_TX_START_INT_ENA** Write 1 to enable interrupt SLCO_TX_START_INT. (R/W)

**SDIO_SLCO_RX_UDF_INT_ENA** Write 1 to enable interrupt SLCO_RX_UDF_INT. (R/W)

**SDIO_SLCO_TX_OVF_INT_ENA** Write 1 to enable interrupt SLCO_TX_OVF_INT. (R/W)

**SDIO_SLCO_TX_DONE_INT_ENA** Write 1 to enable interrupt SLCO_TX_DONE_INT. (R/W)

**SDIO_SLCO_TX_SUC_EOF_INT_ENA** Write 1 to enable interrupt SLCO_TX_SUC_EOF_INT. (R/W)

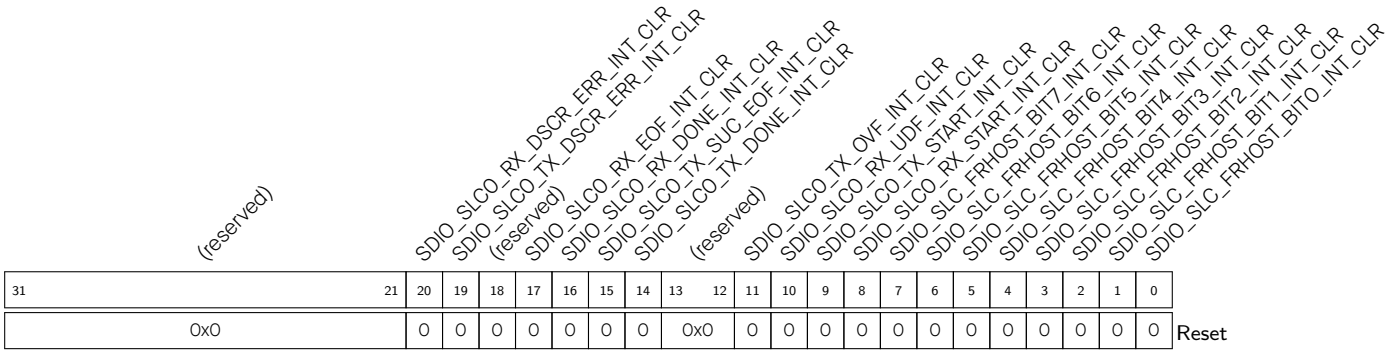
**SDIO_SLCO_RX_DONE_INT_ENA** Write 1 to enable interrupt SLCO_RX_DONE_INT. (R/W)

**SDIO_SLCO_RX_EOF_INT_ENA** Write 1 to enable interrupt SLCO_RX_EOF_INT. (R/W)

**SDIO_SLCO_TX_DSCR_ERR_INT_ENA** Write 1 to enable interrupt SLCO_TX_DSCR_ERR_INT. (R/W)

**SDIO_SLCO_RX_DSCR_ERR_INT_ENA** Write 1 to enable interrupt SLCO_RX_DSCR_ERR_INT. (R/W)

Register 31.33. SDIO_SLC0INT_CLR_REG (0x0010)




**SDIO_SLC_FRHOST_BIT*n*_INT_CLR (*n*: 0-7)** Write 1 to clear interrupt SLC_FRHOST_BIT*n*_INT (*n*: 0-7).(WT)

**SDIO_SLC0_RX_START_INT_CLR** Write 1 to clear interrupt SLC0_RX_START_INT. (WT)

**SDIO_SLC0_TX_START_INT_CLR** Write 1 to clear interrupt SLC0_TX_START_INT. (WT)

**SDIO_SLC0_RX_UDF_INT_CLR** Write 1 to clear interrupt SLC0_RX_UDF_INT. (WT)

**SDIO_SLC0_TX_OVF_INT_CLR** Write 1 to clear interrupt SLC0_TX_OVF_INT. (WT)

**SDIO_SLC0_TX_DONE_INT_CLR** Write 1 to clear interrupt SLC0_TX_DONE_INT. (WT)

**SDIO_SLC0_TX_SUC_EOF_INT_CLR** Write 1 to clear interrupt SLC0_TX_SUC_EOF_INT. (WT)

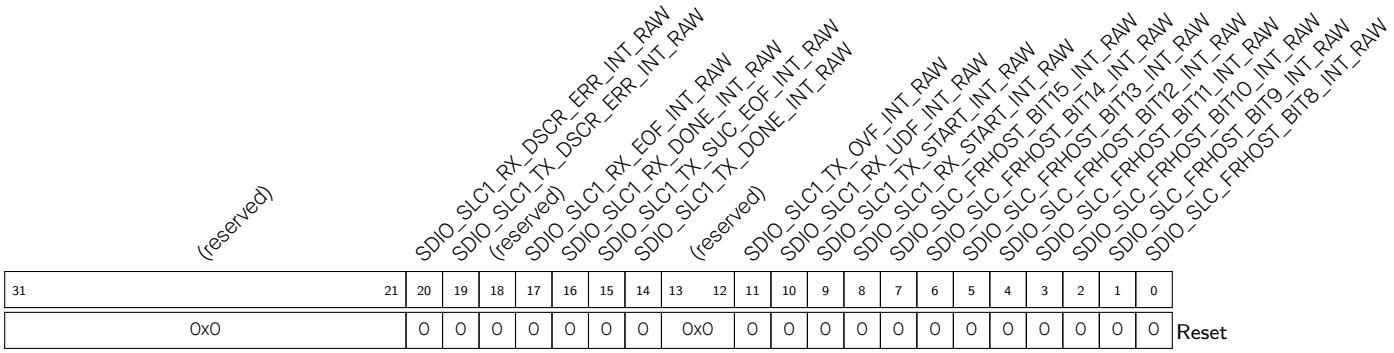
**SDIO_SLC0_RX_DONE_INT_CLR** Write 1 to clear interrupt SLC0_RX_DONE_INT. (WT)

**SDIO_SLC0_RX_EOF_INT_CLR** Write 1 to clear interrupt SLC0_RX_EOF_INT. (WT)

**SDIO_SLC0_TX_DSCR_ERR_INT_CLR** Write 1 to clear interrupt SLC0_TX_DSCR_ERR_INT. (WT)

**SDIO_SLC0_RX_DSCR_ERR_INT_CLR** Write 1 to clear interrupt SLC0_RX_DSCR_ERR_INT. (WT)

Register 31.34. SDIO_SLC1INT_RAW_REG (0x0014)




**SDIO_SLC_FRHOST_BIT*n*_INT_RAW (*n*: 8-15)** The raw interrupt status of SLC_FRHOST_BIT*n*_INT (*n*: 8-15). (R/WTC/SS)

**SDIO_SLC1_RX_START_INT_RAW** The raw interrupt status of SLC1_RX_START_INT. (R/WTC/SS)

**SDIO_SLC1_TX_START_INT_RAW** The raw interrupt status of SLC1_TX_START_INT. (R/WTC/SS)

**SDIO_SLC1_RX_UDF_INT_RAW** The raw interrupt status of SLC1_RX_UDF_INT. (R/WTC/SS)

**SDIO_SLC1_TX_OVF_INT_RAW** The raw interrupt status of SLC1_TX_OVF_INT. (R/WTC/SS)

**SDIO_SLC1_TX_DONE_INT_RAW** The raw interrupt status of SLC1_TX_DONE_INT. (R/WTC/SS)

**SDIO_SLC1_TX_SUC_EOF_INT_RAW** The raw interrupt status of SLC1_TX_SUC_EOF_INT. (R/WTC/SS)

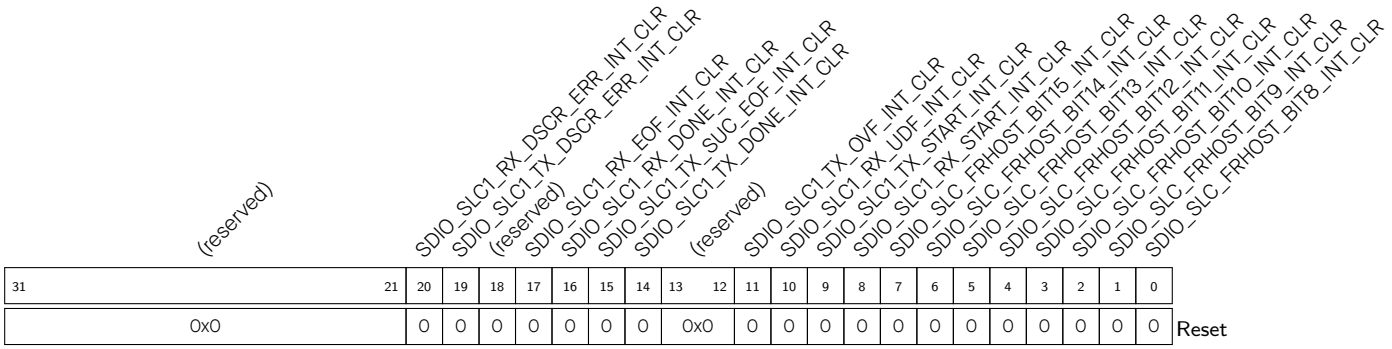
**SDIO_SLC1_RX_DONE_INT_RAW** The raw interrupt status of SLC1_RX_DONE_INT. (R/WTC/SS)

**SDIO_SLC1_RX_EOF_INT_RAW** The raw interrupt status of SLC1_RX_EOF_INT. (R/WTC/SS)

**SDIO_SLC1_TX_DSCR_ERR_INT_RAW** The raw interrupt status of SLC1_TX_DSCR_ERR_INT. (R/WTC/SS)

**SDIO_SLC1_RX_DSCR_ERR_INT_RAW** The raw interrupt status of SLC1_RX_DSCR_ERR_INT. (R/WTC/SS)

Register 31.35. SDIO_SLC1INT_CLR_REG (0x0020)




**SDIO_SLC_FRHOST_BIT*n*_INT_CLR (*n*: 8-15)** Write 1 to clear interrupt SLC_FRHOST_BIT*n*_INT (*n*: 8-15). (WT)

**SDIO_SLC1_RX_START_INT_CLR** Write 1 to clear interrupt SLC1_RX_START_INT. (WT)

**SDIO_SLC1_TX_START_INT_CLR** Write 1 to clear interrupt SLC1_TX_START_INT. (WT)

**SDIO_SLC1_RX_UDF_INT_CLR** Write 1 to clear interrupt SLC1_RX_UDF_INT. (WT)

**SDIO_SLC1_TX_OVF_INT_CLR** Write 1 to clear interrupt SLC1_TX_OVF_INT. (WT)

**SDIO_SLC1_TX_DONE_INT_CLR** Write 1 to clear interrupt SLC1_TX_DONE_INT. (WT)

**SDIO_SLC1_TX_SUC_EOF_INT_CLR** Write 1 to clear interrupt SLC1_TX_SUC_EOF_INT. (WT)

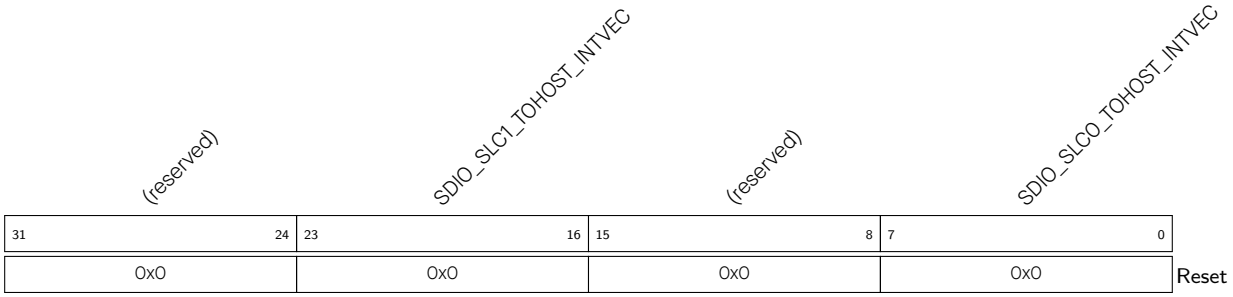
**SDIO_SLC1_RX_DONE_INT_CLR** Write 1 to clear interrupt SLC1_RX_DONE_INT. (WT)

**SDIO_SLC1_RX_EOF_INT_CLR** Write 1 to clear interrupt SLC1_RX_EOF_INT. (WT)

**SDIO_SLC1_TX_DSCR_ERR_INT_CLR** Write 1 to clear interrupt SLC1_TX_DSCR_ERR_INT. (WT)

**SDIO_SLC1_RX_DSCR_ERR_INT_CLR** Write 1 to clear interrupt SLC1_RX_DSCR_ERR_INT. (WT)

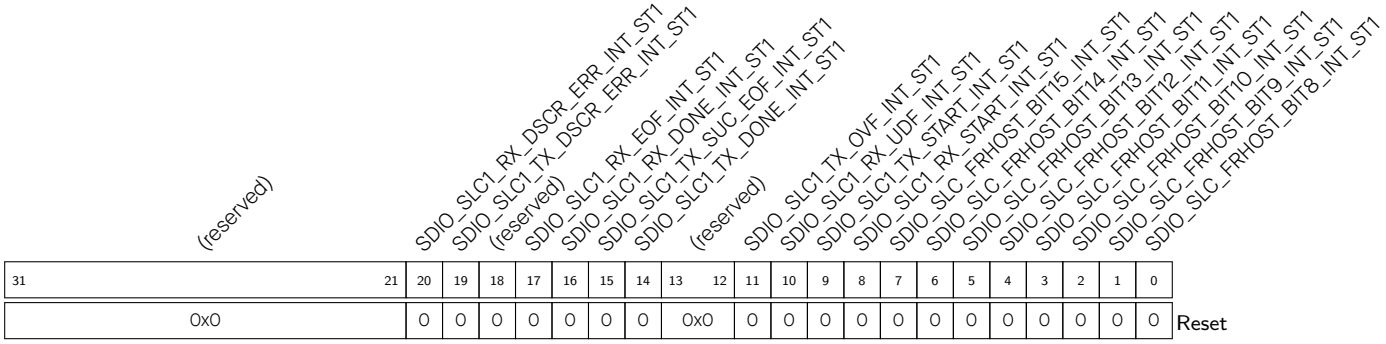
Register 31.36. SDIO_SLCINTVEC_TOHOST_REG (0x005C)




**SDIO_SLC0_TOHOST_INTVEC** The interrupt set bit of SLCHOST_SLC0_TOHOST_BIT*n*_INT (*n*: 0-7). These bits will be cleared automatically. (WT)

**SDIO_SLC1_TOHOST_INTVEC** The interrupt set bit of SLCHOST_SLC1_TOHOST_BIT*n*_INT (*n*: 0-7). These bits will be cleared automatically. (WT)

Register 31.37. SDIO_SLC1INT_ST1_REG (0x014C)




**SDIO_SLC_FRHOST_BIT*n*_INT_ST1 (*n*: 8-15)** The masked interrupt status of SLC_FRHOST_BIT*n*_INT (*n*: 8-15). (RO)

**SDIO_SLC1_RX_START_INT_ST1** The masked interrupt status of SLC1_RX_START_INT. (RO)

**SDIO_SLC1_TX_START_INT_ST1** The masked interrupt status of SLC1_TX_START_INT. (RO)

**SDIO_SLC1_RX_UDF_INT_ST1** The masked interrupt status of SLC1_RX_UDF_INT. (RO)

**SDIO_SLC1_TX_OVF_INT_ST1** The masked interrupt status of SLC1_TX_OVF_INT. (RO)

**SDIO_SLC1_TX_DONE_INT_ST1** The masked interrupt status of SLC1_TX_DONE_INT. (RO)

**SDIO_SLC1_TX_SUC_EOF_INT_ST1** The masked interrupt status of SLC1_TX_SUC_EOF_INT. (RO)

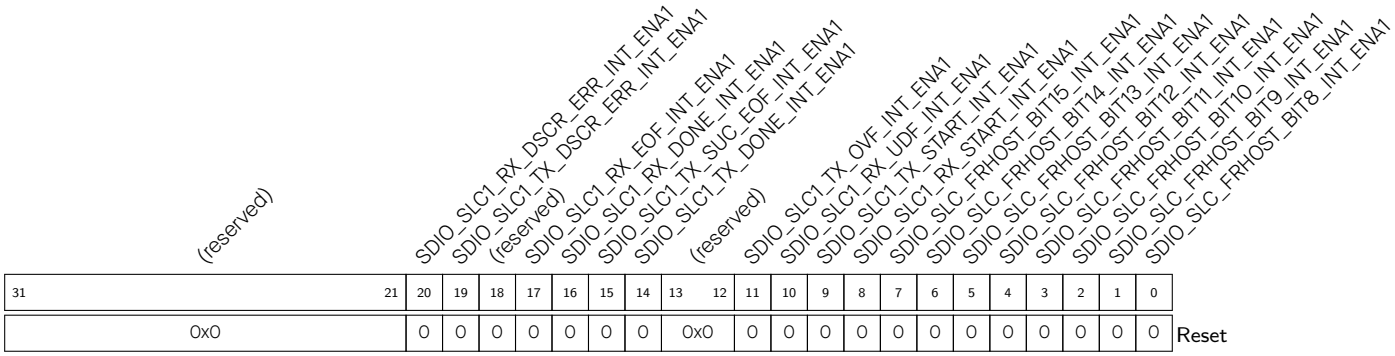
**SDIO_SLC1_RX_DONE_INT_ST1** The masked interrupt status of SLC1_RX_DONE_INT. (RO)

**SDIO_SLC1_RX_EOF_INT_ST1** The masked interrupt status of SLC1_RX_EOF_INT. (RO)

**SDIO_SLC1_TX_DSCR_ERR_INT_ST1** The masked interrupt status of SLC1_TX_DSCR_ERR_INT. (RO)

**SDIO_SLC1_RX_DSCR_ERR_INT_ST1** The masked interrupt status of SLC1_RX_DSCR_ERR_INT. (RO)

Register 31.38. SDIO_SLC1INT_ENA1_REG (0x0150)




**SDIO_SLC_FRHOST_BIT*n*_INT_ENA1 (*n*: 8-15)** Write 1 to enable interrupt SLC_FRHOST_BIT*n*_INT (*n*: 8-15). (R/W)

**SDIO_SLC1_RX_START_INT_ENA1** Write 1 to enable interrupt SLC1_RX_START_INT. (R/W)

**SDIO_SLC1_TX_START_INT_ENA1** Write 1 to enable interrupt SLC1_TX_START_INT. (R/W)

**SDIO_SLC1_RX_UDF_INT_ENA1** Write 1 to enable interrupt SLC1_RX_UDF_INT. (R/W)

**SDIO_SLC1_TX_OVF_INT_ENA1** Write 1 to enable interrupt SLC1_TX_OVF_INT. (R/W)

**SDIO_SLC1_TX_DONE_INT_ENA1** Write 1 to enable interrupt SLC1_TX_DONE_INT. (R/W)

**SDIO_SLC1_TX_SUC_EOF_INT_ENA1** Write 1 to enable interrupt SLC1_TX_SUC_EOF_INT. (R/W)

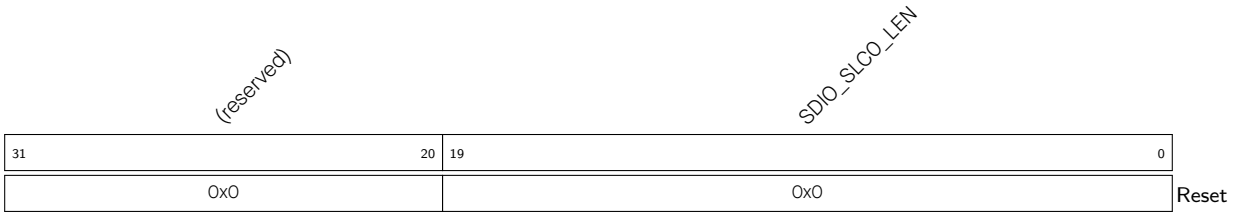
**SDIO_SLC1_RX_DONE_INT_ENA1** Write 1 to enable interrupt SLC1_RX_DONE_INT. (R/W)

**SDIO_SLC1_RX_EOF_INT_ENA1** Write 1 to enable interrupt SLC1_RX_EOF_INT. (R/W)

**SDIO_SLC1_TX_DSCR_ERR_INT_ENA1** Write 1 to enable interrupt SLC1_TX_DSCR_ERR_INT. (R/W)

**SDIO_SLC1_RX_DSCR_ERR_INT_ENA1** Write 1 to enable interrupt SLC1_RX_DSCR_ERR_INT. (R/W)

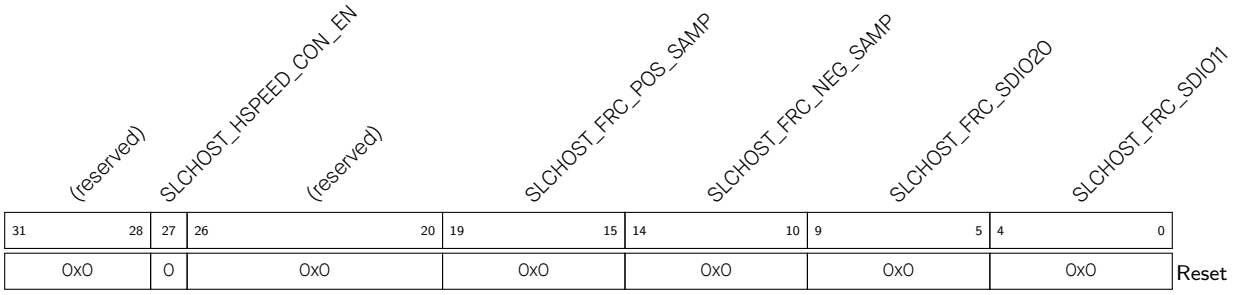
Register 31.39. SDIO_SLC0_LENGTH_REG (0x00F8)




**SDIO_SLC0_LEN** Represents the accumulated length of data that the slave wants to send. (RO)

### 31.9.3 SLC Host Registers

Register 31.40. SLCHOST_CONF_REG (0x01F0)




**SLCHOST_FRC_SDIO11** Configure 1 to bit[4] to force drive CMD signal at the falling clock edge. Configures 1 to bit[3:0] corresponding bit to force drive DAT[3:0] signal corresponding bit at the falling clock edge. (R/W)

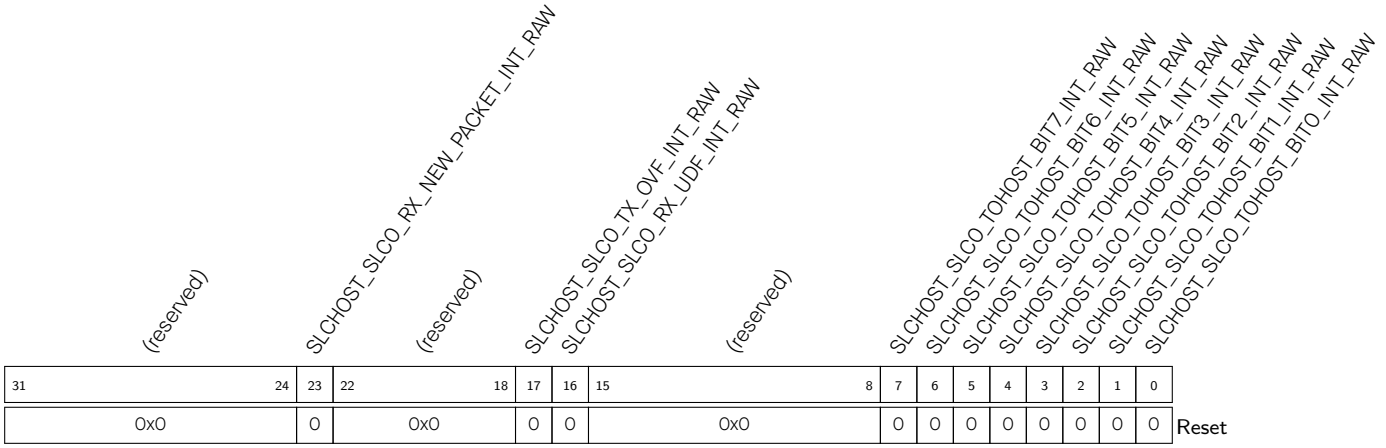
**SLCHOST_FRC_SDIO20** Configure 1 to bit[4] to force drive CMD signal at the rising clock edge. Configures 1 to bit[3:0] corresponding bit to force drive DAT[3:0] signal corresponding bit at the rising clock edge. (R/W)

**SLCHOST_FRC_NEG_SAMP** Configure 1 to bit[4] to force sample CMD signal at the falling clock edge. Configures 1 to bit[3:0] corresponding bit to force sample DAT[3:0] signal corresponding bit at the falling clock edge. (R/W)

**SLCHOST_FRC_POS_SAMP** Configure 1 to bit[4] to force sample CMD signal at the rising clock edge. Configures 1 to bit[3:0] corresponding bit to force sample DAT[3:0] signal corresponding bit at the rising clock edge. (R/W)

**SLCHOST_HSPEED_CON_EN** Configures 1 to this bit, configures 1 to HINF_HIGHSPEED_ENABLE, and then the host configures 1 to EHS in CCCR to force drive CMD and DAT signals at the rising clock edge. (R/W)

Register 31.41. SLCHOST_SLCOHOST_INT_RAW_REG (0x0050)



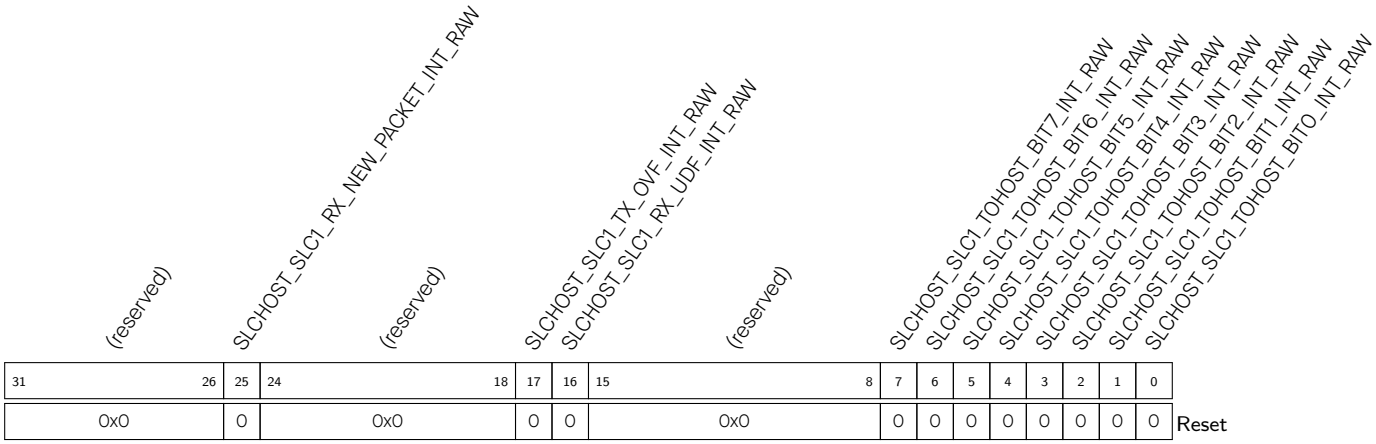

**SLCHOST_SLCO_TOHOST_BIT*n*_INT_RAW (*n*: 0-7)** The raw interrupt status of SLCHOST_SLCO_TOHOST_BIT*n*_INT (*n*: 0-7). (R/WTC/SS)

**SLCHOST_SLCO_RX_UDF_INT_RAW** The raw interrupt status of SLCHOST_SLCO_RX_UDF_INT. (R/WTC/SS)

**SLCHOST_SLCO_TX_OVF_INT_RAW** The raw interrupt status of SLCHOST_SLCO_TX_OVF_INT. (R/WTC/SS)

**SLCHOST_SLCO_RX_NEW_PACKET_INT_RAW** The raw interrupt status of SLCHOST_SLCO_RX_NEW_PACKET_INT. (R/WTC/SS)

Register 31.42. SLCHOST_SLC1HOST_INT_RAW_REG (0x0054)



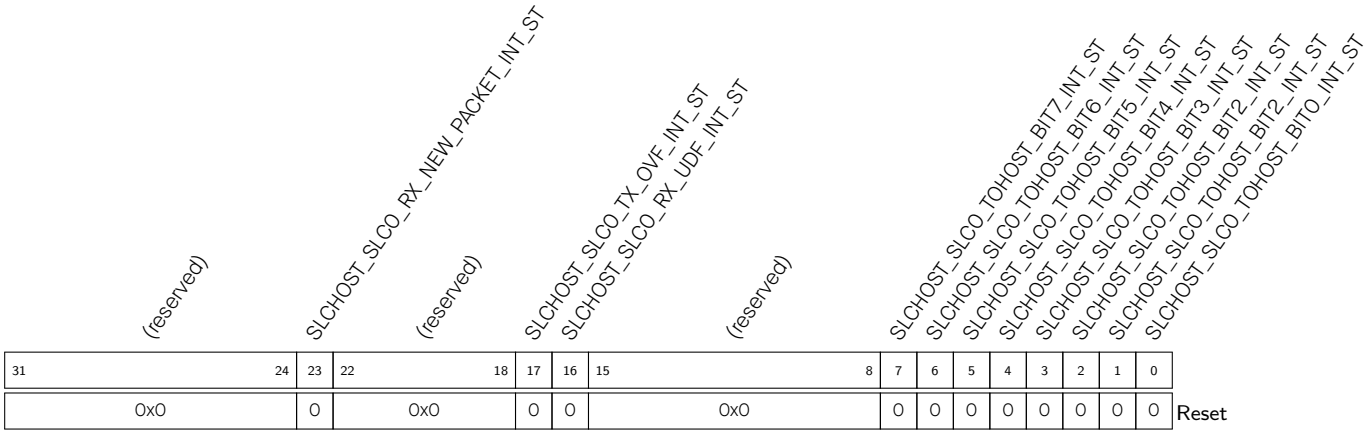

**SLCHOST_SLC1_TOHOST_BIT*n*_INT_RAW (*n*: 0-7)** The raw interrupt status of SLCHOST_SLC1_TOHOST_BIT*n*_INT (*n*: 0-7). (R/WTC/SS)

**SLCHOST_SLC1_RX_UDF_INT_RAW** The raw interrupt status of SLCHOST_SLC1_RX_UDF_INT. (R/WTC/SS)

**SLCHOST_SLC1_TX_OVF_INT_RAW** The raw interrupt status of SLCHOST_SLC1_TX_OVF_INT. (R/WTC/SS)

**SLCHOST_SLC1_RX_NEW_PACKET_INT_RAW** The raw interrupt status of SLCHOST_SLC1_RX_NEW_PACKET_INT. (R/WTC/SS)

Register 31.43. SLCHOST_SLC0HOST_INT_ST_REG (0x0058)



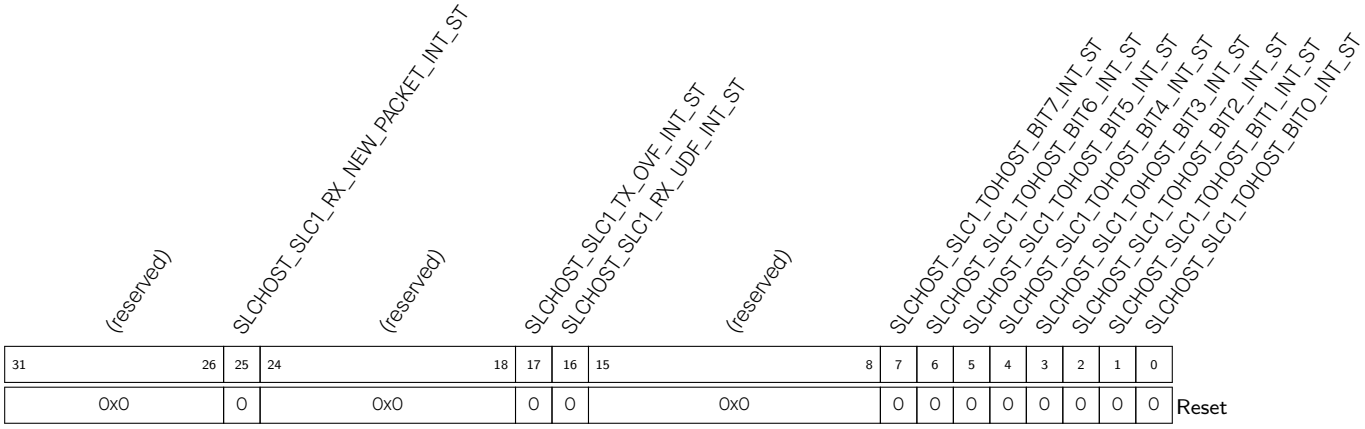

**SLCHOST_SLC0_TOHOST_BIT*n*_INT_ST (*n*: 0-7)** The masked interrupt status of SLCHOST_SLC0_TOHOST_BIT*n*_INT (*n*: 0-7). (RO)

**SLCHOST_SLC0_RX_UDF_INT_ST** The masked interrupt status of SLCHOST_SLC0_RX_UDF_INT. (RO)

**SLCHOST_SLC0_TX_OVF_INT_ST** The masked interrupt status of SLCHOST_SLC0_TX_OVF_INT. (RO)

**SLCHOST_SLC0_RX_NEW_PACKET_INT_ST** The masked interrupt status of SLCHOST_SLC0_RX_NEW_PACKET_INT. (RO)

Register 31.44. SLCHOST_SLC1HOST_INT_ST_REG (0x005C)



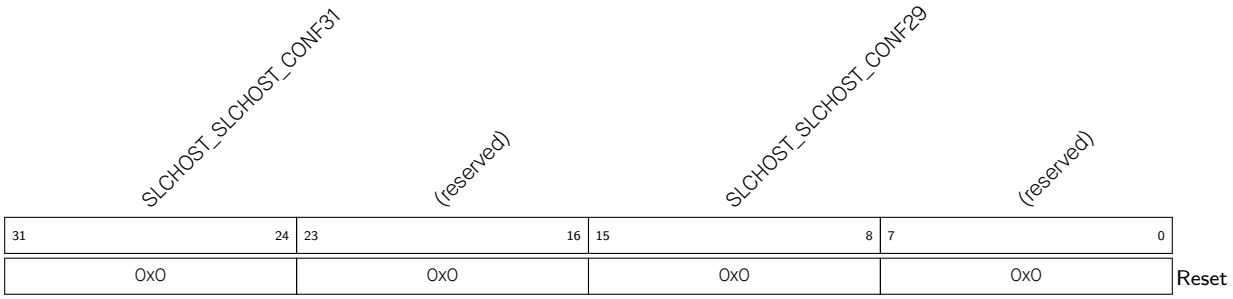

**SLCHOST_SLC1_TOHOST_BIT*n*_INT_ST (*n*: 0-7)** The masked interrupt status of SLCHOST_SLC1_TOHOST_BIT*n*_INT (*n*: 0-7). (RO)

**SLCHOST_SLC1_RX_UDF_INT_ST** The masked interrupt status of SLCHOST_SLC1_RX_UDF_INT. (RO)

**SLCHOST_SLC1_TX_OVF_INT_ST** The masked interrupt status of SLCHOST_SLC1_TX_OVF_INT. (RO)

**SLCHOST_SLC1_RX_NEW_PACKET_INT_ST** The masked interrupt status of SLCHOST_SLC1_RX_NEW_PACKET_INT. (RO)

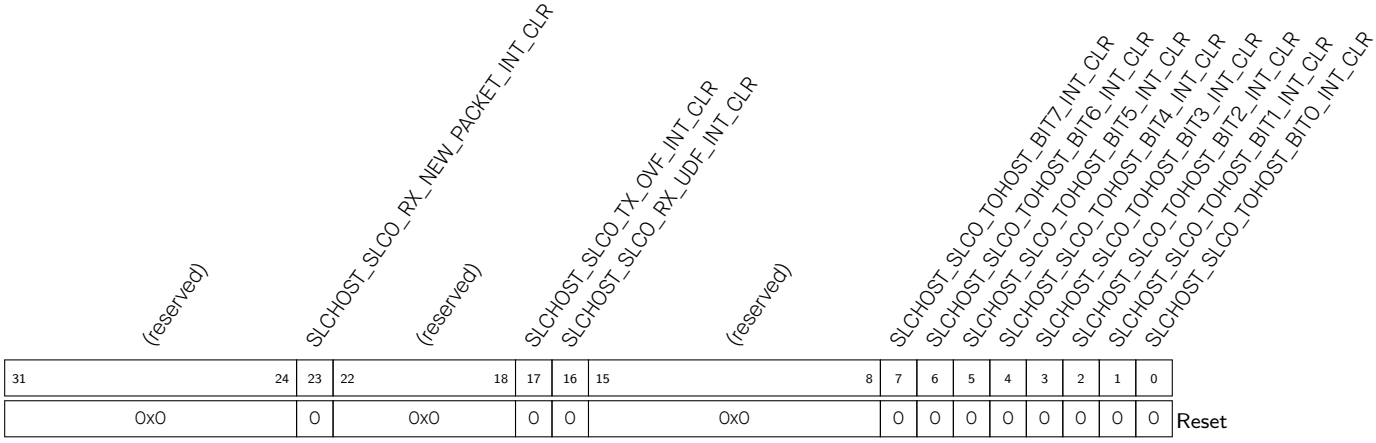
Register 31.45. SLCHOST_CONF_W7_REG (0x008C)




**SLCHOST_SLCHOST_CONF29** The interrupt set bits of SLCINT_SLC_FRHOST_BIT*n*_INT (*n*: 0-7). These bits will not be cleared automatically. (R/W)

**SLCHOST_SLCHOST_CONF31** The interrupt set bits of SLCINT_SLC_FRHOST_BIT*n*_INT (*n*: 8-15). These bits will not be cleared automatically. (R/W)

Register 31.46. SLCHOST_SLC0HOST_INT_CLR_REG (0x00D4)



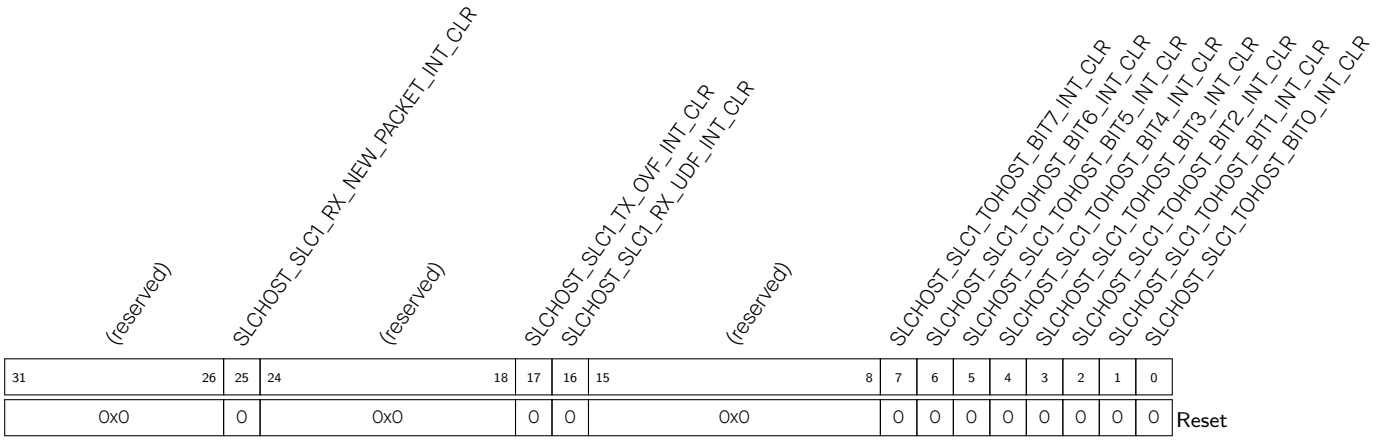

**SLCHOST_SLC0_TOHOST_BIT*n*_INT_CLR (*n*: 0-7)** Write 1 to clear interrupt SLCHOST_SLC0_TOHOST_BIT*n*_INT (*n*: 0-7). (WT)

**SLCHOST_SLC0_RX_UDF_INT_CLR** Write 1 to clear interrupt SLCHOST_SLC0_RX_UDF_INT. (WT)

**SLCHOST_SLC0_TX_OVF_INT_CLR** Write 1 to clear interrupt SLCHOST_SLC0_TX_OVF_INT. (WT)

**SLCHOST_SLC0_RX_NEW_PACKET_INT_CLR** Write 1 to clear interrupt SLCHOST_SLC0_RX_NEW_PACKET_INT. (WT)

Register 31.47. SLCHOST_SLC1HOST_INT_CLR_REG (0x00D8)



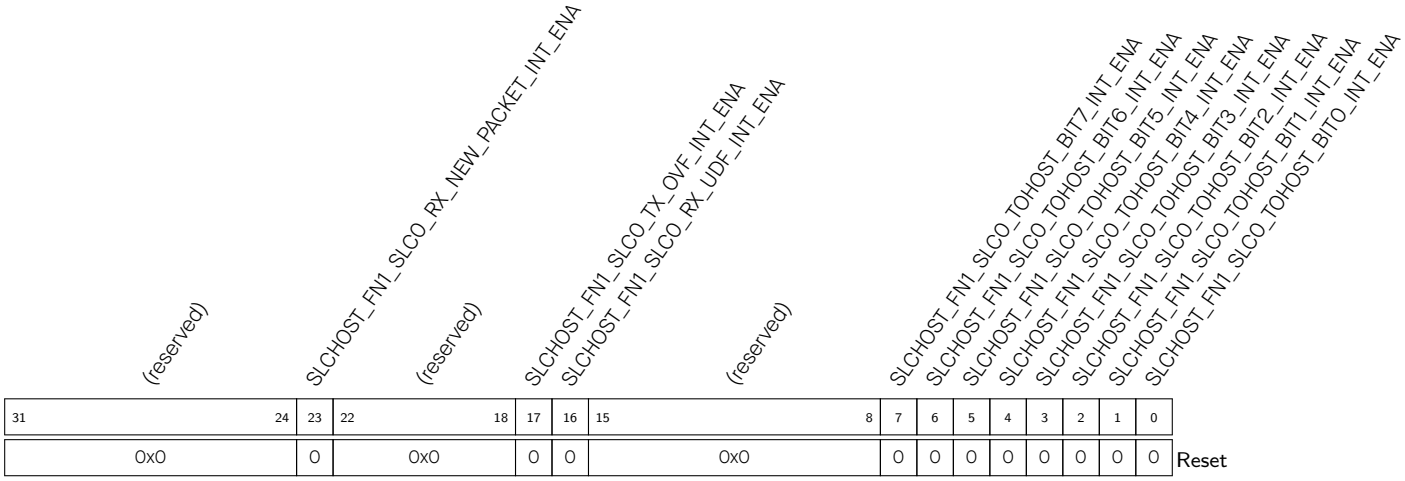

**SLCHOST_SLC1_TOHOST_BIT*n*_INT_CLR (*n*: 0-7)** Write 1 to clear interrupt SLCHOST_SLC1_TOHOST_BIT*n*_INT (*n*: 0-7). (WT)

**SLCHOST_SLC1_RX_UDF_INT_CLR** Write 1 to clear interrupt SLCHOST_SLC1_RX_UDF_INT. (WT)

**SLCHOST_SLC1_TX_OVF_INT_CLR** Write 1 to clear interrupt SLCHOST_SLC1_TX_OVF_INT. (WT)

**SLCHOST_SLC1_RX_NEW_PACKET_INT_CLR** Write 1 to clear interrupt SLCHOST_SLC1_RX_NEW_PACKET_INT. (WT)

Register 31.48. SLCHOST_SLCOHOST_FUNC1_INT_ENA_REG (0x00DC)



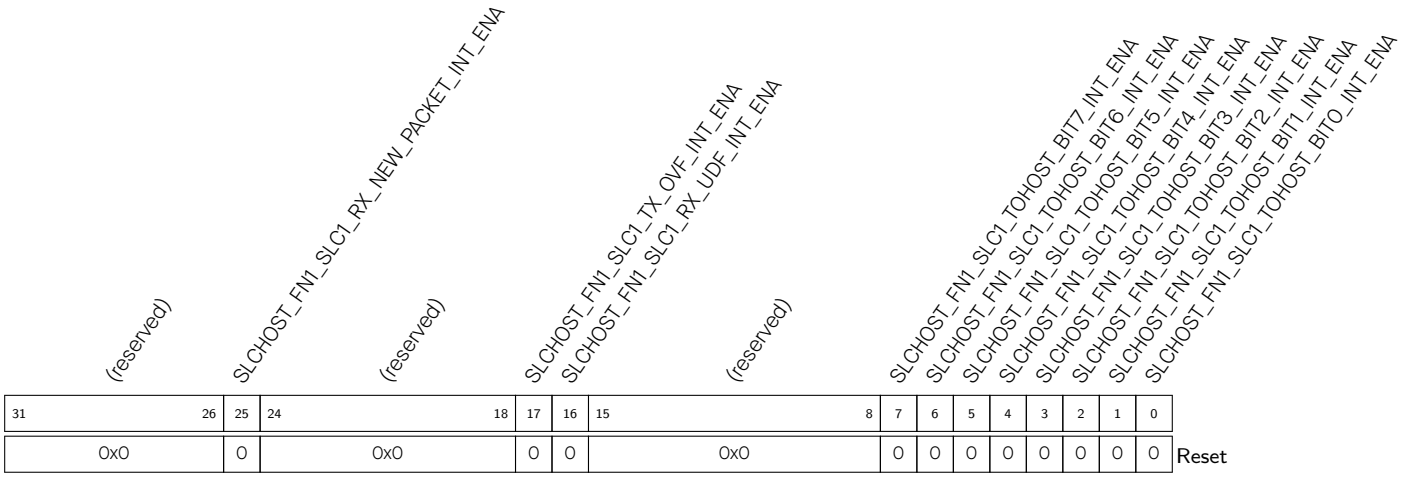

**SLCHOST_FN1_SLC0_TOHOST_BIT*n*_INT_ENA (*n*: 0-7)** Write 1 to enable SLCHOST_SLC0_TOHOST_BIT*n*_INT (*n*: 0-7). (R/W)

**SLCHOST_FN1_SLC0_RX_UDF_INT_ENA** Write 1 to enable SLCHOST_SLC0_RX_UDF_INT. (R/W)

**SLCHOST_FN1_SLC0_TX_OVF_INT_ENA** Write 1 to enable SLCHOST_SLC0_TX_OVF_INT. (R/W)

**SLCHOST_FN1_SLC0_RX_NEW_PACKET_INT_ENA** Write 1 to enable SLCHOST_SLC0_RX_NEW_PACKET_INT. (R/W)

Register 31.49. SLCHOST_SLC1HOST_FUNC1_INT_ENA_REG (0x00E0)



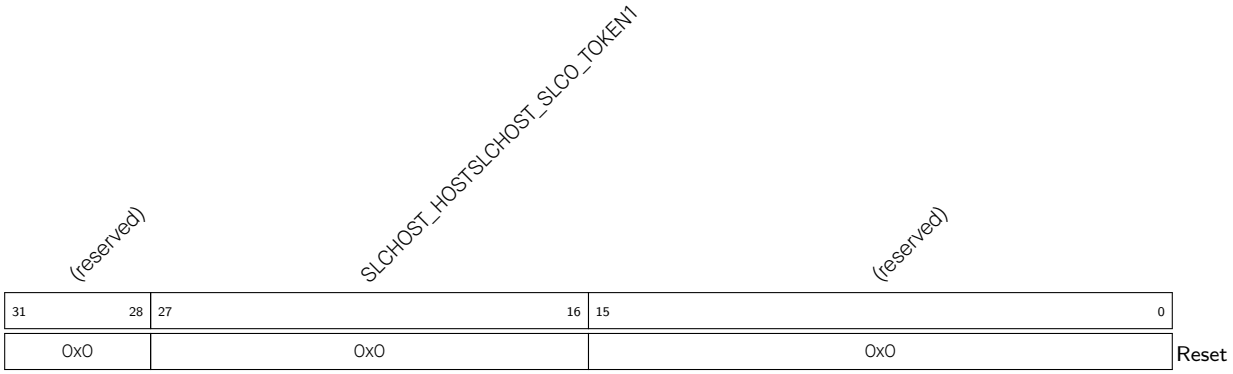

**SLCHOST_FN1_SLC1_TOHOST_BIT*n*_INT_ENA (*n*: 0-7)** Write 1 to enable SLCHOST_SLC1_TOHOST_BIT*n*_INT (*n*: 0-7). (R/W)

**SLCHOST_FN1_SLC1_RX_UDF_INT_ENA** Write 1 to enable SLCHOST_SLC1_RX_UDF_INT. (R/W)

**SLCHOST_FN1_SLC1_TX_OVF_INT_ENA** Write 1 to enable SLCHOST_SLC1_TX_OVF_INT. (R/W)

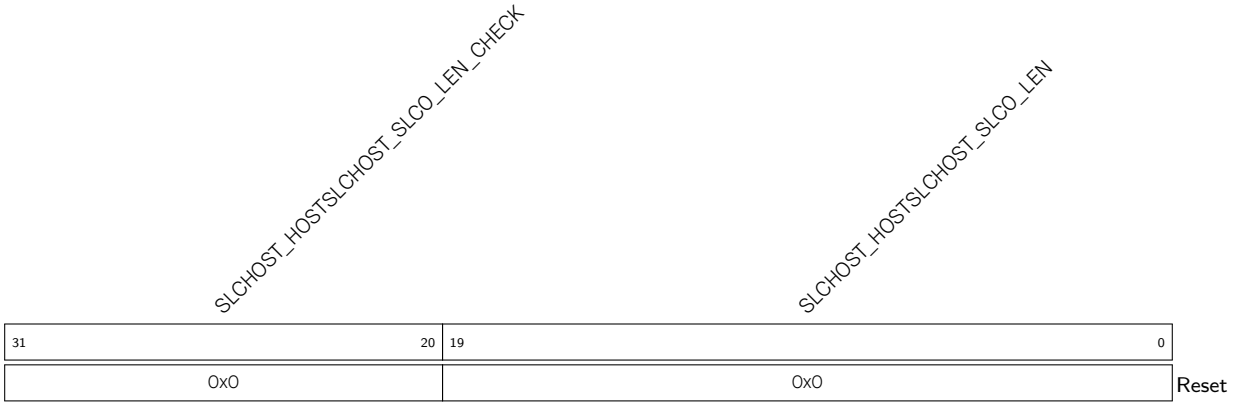
**SLCHOST_FN1_SLC1_RX_NEW_PACKET_INT_ENA** Write 1 to enable SLCHOST_SLC1_RX_NEW_PACKET_INT. (R/W)

Register 31.50. SLCHOST_SLCOHOST_TOKEN_RDATA_REG (0x0044)




**SLCHOST_HOSTSLCHOST_SLC0_TOKEN1** Represents the SLC0 accumulated number of buffers for receiving data. (RO)

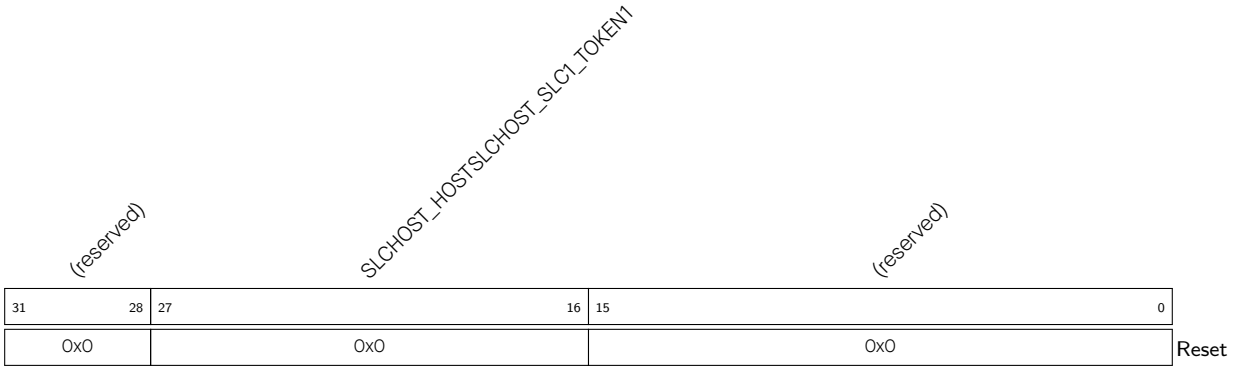
Register 31.51. SLCHOST_PKT_LEN_REG (0x0060)




**SLCHOST_HOSTSLCHOST_SLC0_LEN** Represents the accumulated length of data that the slave wants to send. The value gets updated only when the host reads it. (RO)

**SLCHOST_HOSTSLCHOST_SLC0_LEN_CHECK** Check SLCHOST_HOSTSLCHOST_SLC0_LEN. Its value is SLCHOST_HOSTSLCHOST_SLC0_LEN bit[9:0] plus bit[19:10]. (RO)

Register 31.52. SLCHOST_SLC1HOST_TOKEN_RDATA_REG (0x00C4)



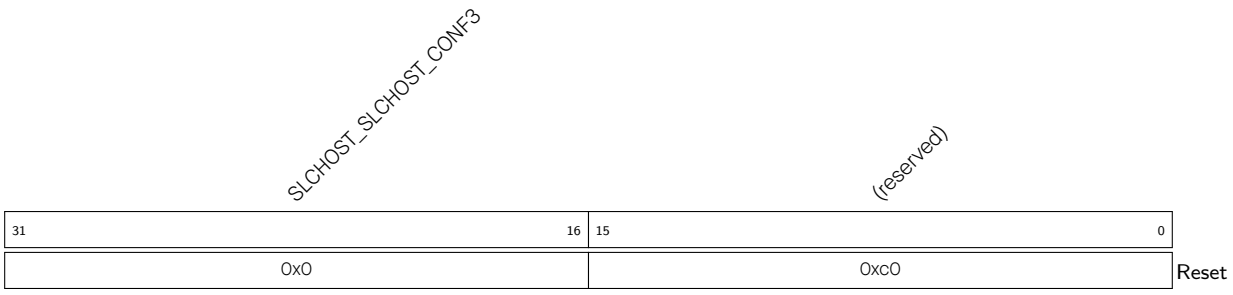

**SLCHOST_HOSTSLCHOST_SLC1_TOKEN1** Represents the SLC1 accumulated number of buffers for receiving data. (RO)

Register 31.53. SLCHOST_CONF_W*n*_REG(*n*: 0-2) (0x006C+0x4**n*)



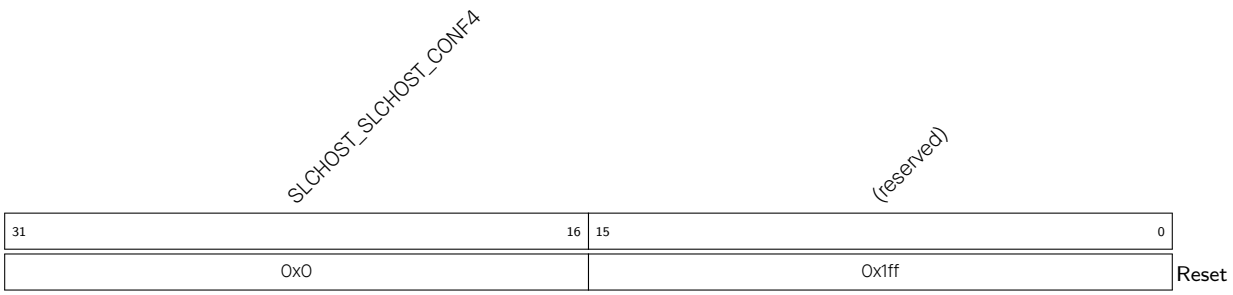

**SLCHOST_SLCHOST_CONF*n*** The information interaction register between the host and slave. Both of them can access it. (R/W)

Register 31.54. SLCHOST_CONF_W3_REG (0x0078)




**SLCHOST_SLCHOST_CONF3** The information interaction register between the host and slave. Both of them can access it. (R/W)

Register 31.55. SLCHOST_CONF_W4_REG (0x007C)



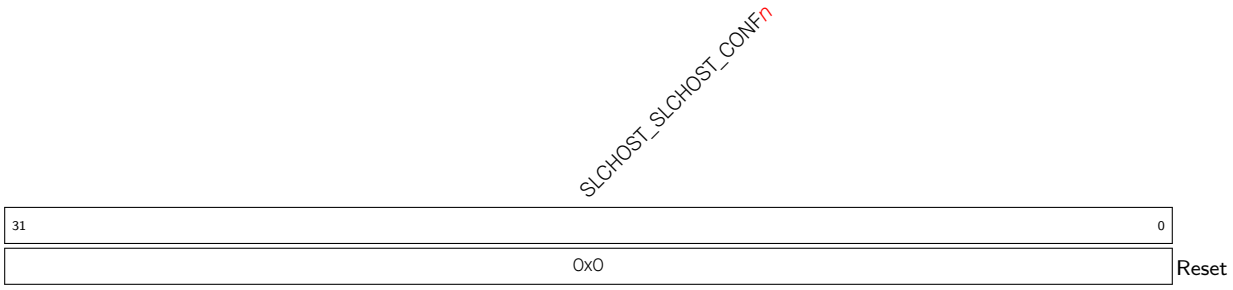

**SLCHOST_SLCHOST_CONF4** The information interaction register between the host and slave. Both of them can access it. (R/W)

Register 31.56. SLCHOST_CONF_W6_REG (0x0088)




**SLCHOST_SLCHOST_CONF6** The information interaction register between the host and slave. Both of them can access it. (R/W)

Register 31.57. SLCHOST_CONF_W*n*_REG(*n*: 8-15) (0x009C+0x4*(*n*-8))




**SLCHOST_SLCHOST_CONF*n*** The information interaction register between the host and slave. Both of them can access it. (R/W)

# Chapter 32

# LED PWM Controller (LEDC)

## 32.1 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.

## 32.2 Features

The LED PWM Controller has the following features:

- Six independent PWM generators (i.e., six 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 six PWM generators are also identical in features and operation, and thus are collectively referred to as PWM*n* (where *n* ranges from 0 to 5).

## 32.3 Architectural Overview

Figure 32.3-1 shows the architecture of the LED PWM Controller.

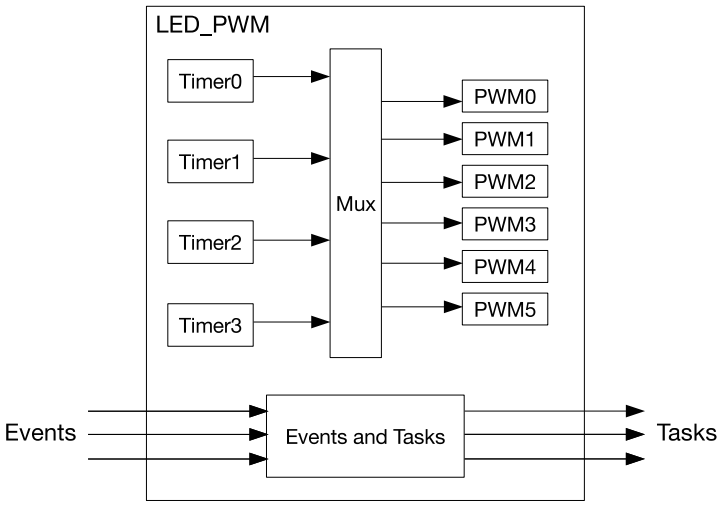



Figure 32.3-1. LED PWM Architecture

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 32.3-2 illustrates the main functional blocks of the timer and the PWM generator.

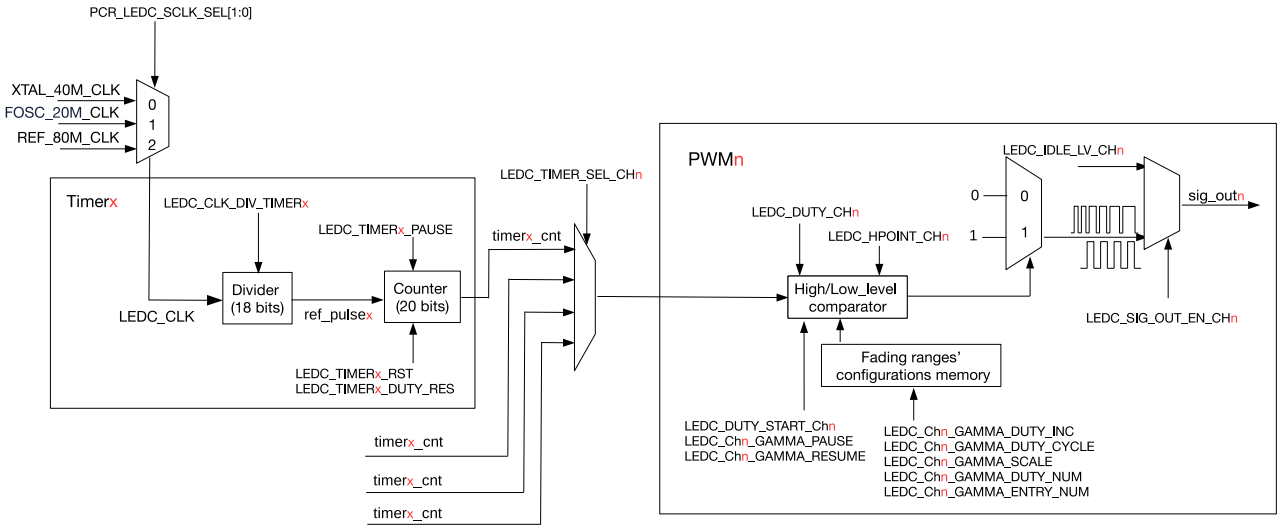



Figure 32.3-2. Timer and PWM Generator Block Diagram

# 32.4 Functional Description

## 32.4.1 Timers

Each timer in LED PWM Controller internally maintains a timebase counter. Referring to Figure 32.3-2, 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.

#### 32.4.1.1 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 PCR_LEDC_CLK_EN field in the PCR_LEDC_CONF_REG register. The LEDC_CLK signal to LED PWM can be enabled by setting the PCR_LEDC_SCLK_EN field in the PCR_LEDC_SCLK_CONF_REG register. The LED PWM peripheral can be reset via software by setting the PCR_LEDC_RST_EN field in the PCR_LEDC_CONF_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 8 *Reset and Clock*.

#### 32.4.1.2 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 32.3-2).

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 32.4-1 illustrates the relation between LEDC_CLK clock pulses and ref_pulse*x* clock pulses when LEDC_CLK_DIV is a non-integer value.

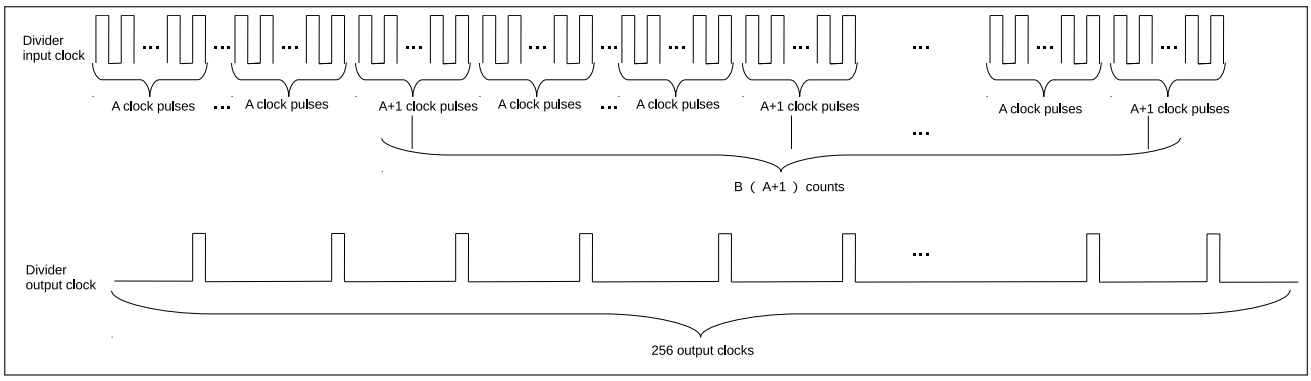



Figure 32.4-1. Frequency Division When LEDC_CLK_DIV is a Non-Integer Value

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.

### 32.4.1.3 20-Bit Counter

Each timer contains a 20-bit timebase counter that uses ref_pulse*x* as its reference clock (see Figure 32.3-2). 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 32.4-2 shows the relationship between the counter and PWM resolution.

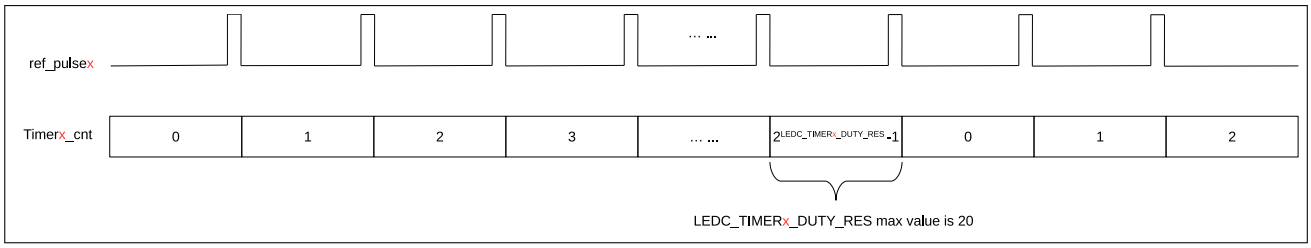



Figure 32.4-2. Relationship Between Counter And Resolution

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_TIMERx_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_CHn field is reconfigured, LEDC_PARA_UP_CHn should be set to apply the new configuration. In summary, these configuration values need to be updated by setting LEDC_TIMERx_PARA_UP or LEDC_PARA_UP_CHn. LEDC_TIMERx_PARA_UP and LEDC_PARA_UP_CHn will be automatically cleared by hardware.

Referring to Figure 32.3-2, the frequency of a PWM generator output signal (sig_outn) 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_TIMERx_DUTY_RES:

$$f_{\text{PWM}} = \frac{f_{\text{LEDC\_CLK}}}{\text{LEDC\_CLK\_DIV} \cdot 2^{\text{LEDC\_TIMERx\_DUTY\_RES}}}$$

Based on the formula above, the desired duty resolution can be calculated as follows:

$$\text{LEDC\_TIMERx\_DUTY\_RES} = \log_2 \left( \frac{f_{\text{LEDC\_CLK}}}{f_{\text{PWM}} \cdot \text{LEDC\_CLK\_DIV}} \right)$$

Table 32.4-1 lists the commonly-used frequencies and their corresponding resolutions.

**Table 32.4-1. 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.

## 32.4.2 PWM Generators

To generate a PWM signal, a PWM generator (PWMn) needs a timer (Timerx). Each PWM generator can be configured separately by setting LEDC_TIMER_SEL_CHn to use one of four timers to generate the PWM output.

As shown in Figure 32.3-2, each PWM generator has a comparator and two multiplexers. A PWM generator compares the timer's 20-bit counter value (Timerx_cnt) to two trigger values Hpointn and Lpointn. When the timer's counter value is equal to Hpointn or Lpointn, 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 32.4-3 illustrates how Hpoint*n* and Lpoint*n* are used to generate a fixed duty cycle PWM output signal.

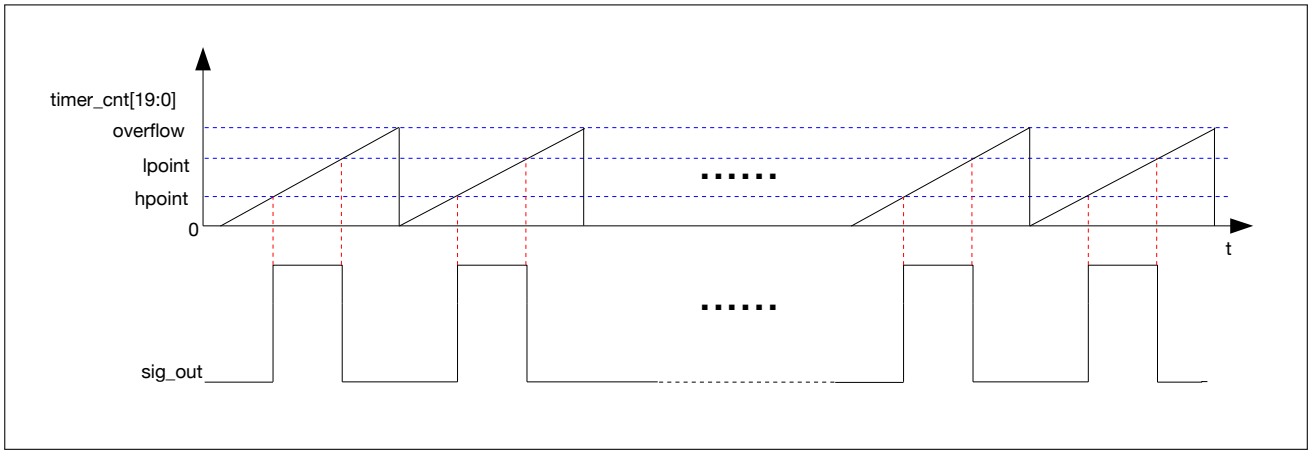



Figure 32.4-3. LED PWM Output Signal Diagram

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.

### 32.4.3 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.

#### 32.4.3.1 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 32.6.

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 32.4-4 shows a linear fading PWM signal.

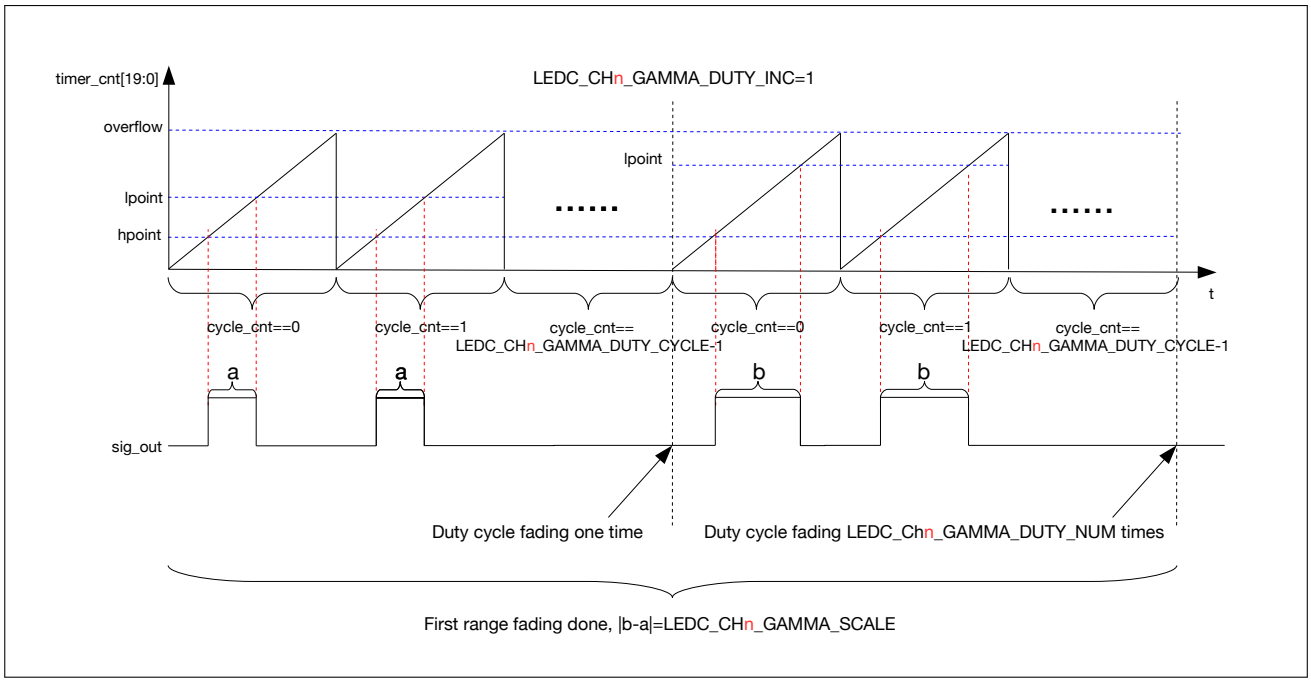



Figure 32.4-4. Output Signal of Linear Duty Cycle Fading

#### 32.4.3.2 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 32.6.

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 32.4-5 illustrates a gamma curve fading PWM signal.

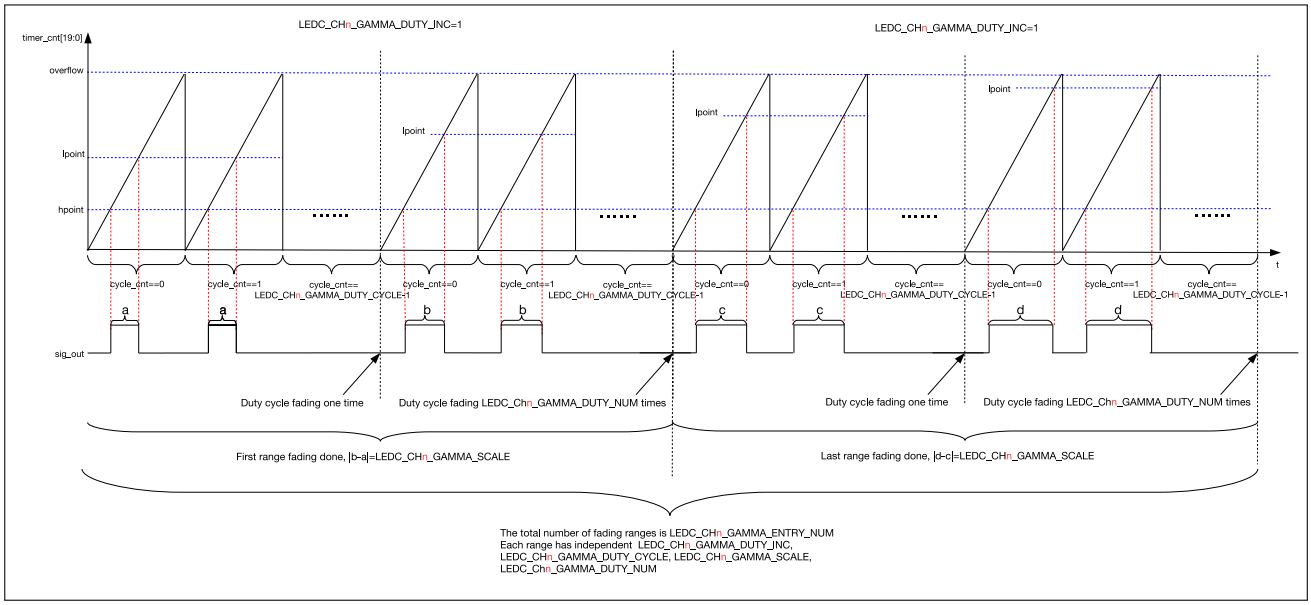



Figure 32.4-5. Output Signal of Gamma Curve Fading

### 32.4.3.3 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.

### 32.4.4 Event Task Matrix Feature

The LEDC on ESP32-C61 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 11 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 32.8).

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 32.3-2.
- 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.

## 32.5 Interrupts

ESP32-C61'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).

## 32.6 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 32.4.1. After that, the clock source, divider, and counter are ready.
2. Configure the PWM*n* generator according to section 32.4.2. 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 32.7-1. 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 32.4.3.1.

ii. The same as Step 2 in Section 32.4.3.1.

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 32.7-1.

- 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 32.4.3.3. 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 32.4.2.

## 32.7 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 5.3-2 in Chapter 5 *System and Memory*.

**Table 32.7-1. 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 |

## 32.8 Register Summary

The addresses in this section are relative to LED PWM Controller base address provided in Table 5.3-2 in Chapter 5 *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_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 |
| 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_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 |

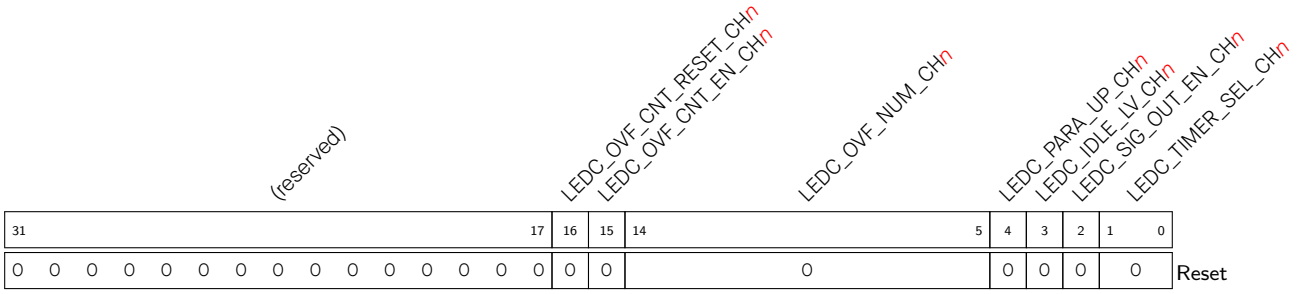
| Name | Description | Address | Access |
|---|---|---|---|
| LEDC_EVT_TASK_EN2_REG | LEDC event task enable bit register 2 | 0x0128 | R/W |
| LEDC_TIMER0_CMP_REG | LEDC timer 0 compare value register | 0x0140 | R/W |
| LEDC_TIMER1_CMP_REG | LEDC timer 1 compare value register | 0x0144 | R/W |
| LEDC_TIMER2_CMP_REG | 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_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 |

## 32.9 Registers

The addresses in this section are relative to LED PWM Controller base address provided in Table 5.3-2 in Chapter 5 *System and Memory*.

For how to program reserved fields, please refer to Section Programming Reserved Register Field.

**Register 32.1. LEDC_CH*n*_CONF0_REG (*n*: 0-5) (0x0000+0x14**n*)**




**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 32.1. LEDC_CH*n*_CONF0_REG (*n*: 0-5) (0x0000+0x14**n*)

Continued from the previous page...

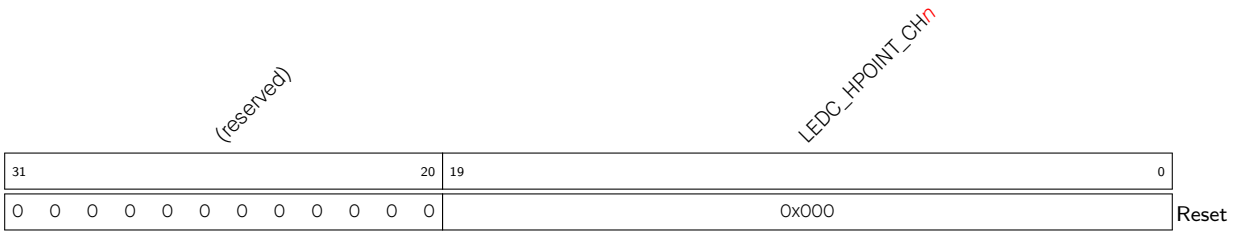
**LEDC_OVF_CNT_RESET_CH*n*** Configures whether to reset the ovf_cnt of channel *n*.

0: Invalid. No effect

1: Reset the ovf_cnt

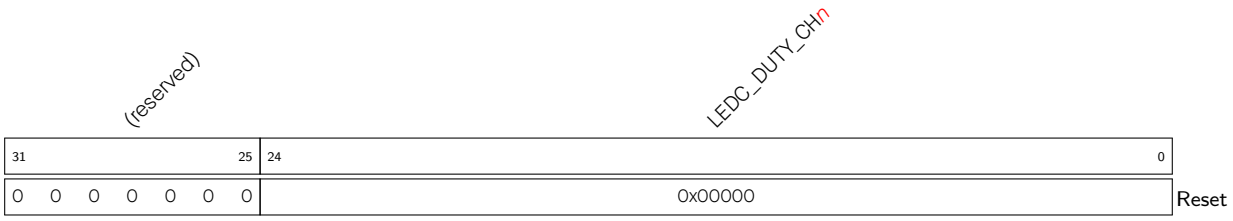
(WT)

Register 32.2. LEDC_CH*n*_HPOINT_REG (*n*: 0-5) (0x0004+0x14**n*)



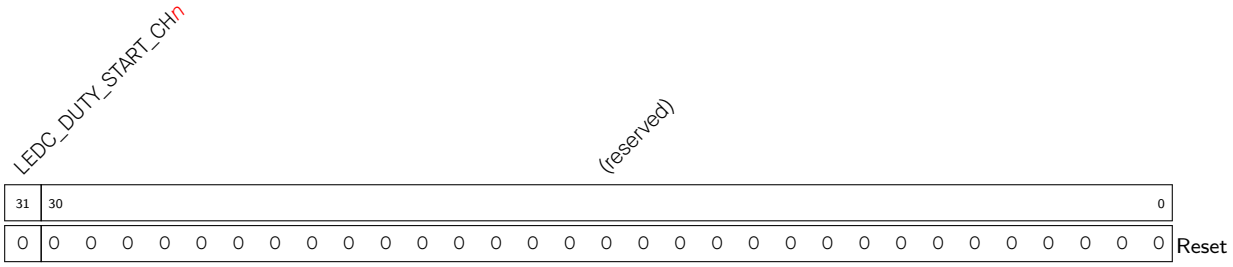

**LEDC_HPOINT_CH*n*** Configures high point of signal output on channel *n*. The output value changes to high when the selected timers have reached the value specified by this register. (R/W)

Register 32.3. LEDC_CH*n*_DUTY_REG (*n*: 0-5) (0x0008+0x14**n*)




**LEDC_DUTY_CH*n*** Configures the duty cycle of signal output on channel *n*. (R/W)

Register 32.4. LEDC_CH*n*_CONF1_REG (*n*: 0-5) (0x000C+0x14**n*)



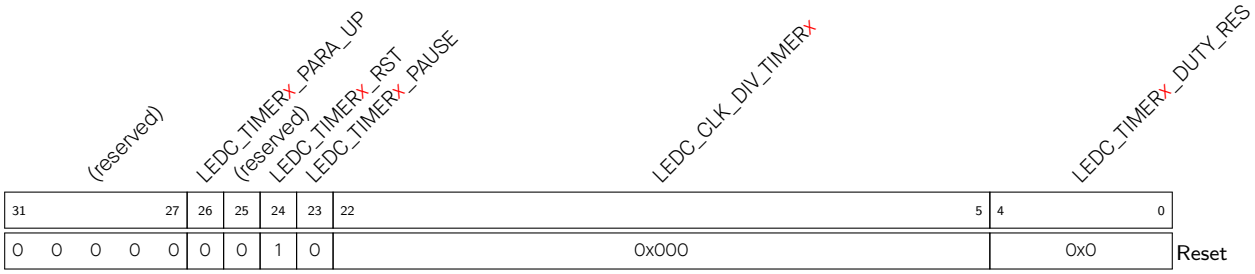

**LEDC_DUTY_START_CH*n*** Configures whether the duty cycle fading configurations take effect.

0: Not take effect

1: Take effect

(R/W/SC)

Register 32.5. 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)

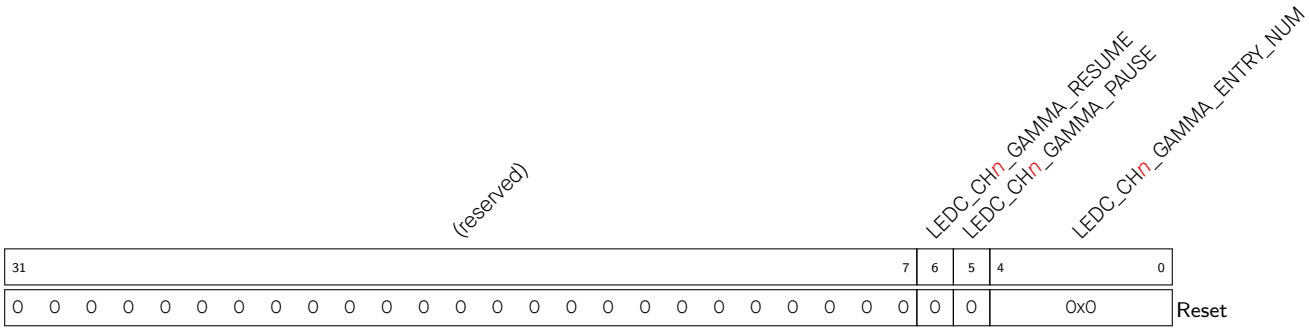
**LEDC_TIMER*x*_PARA_UP** Configures whether to update LEDC_CLK_DIV_TIMER*x* and LEDC_TIMER*x*_DUTY_RES.

0: Invalid. No effect

1: Update

(WT)

**Register 32.6. LEDC_CH*n*_GAMMA_CONF_REG (*n*: 0-5) (0x0100+0x4**n*)**




**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)

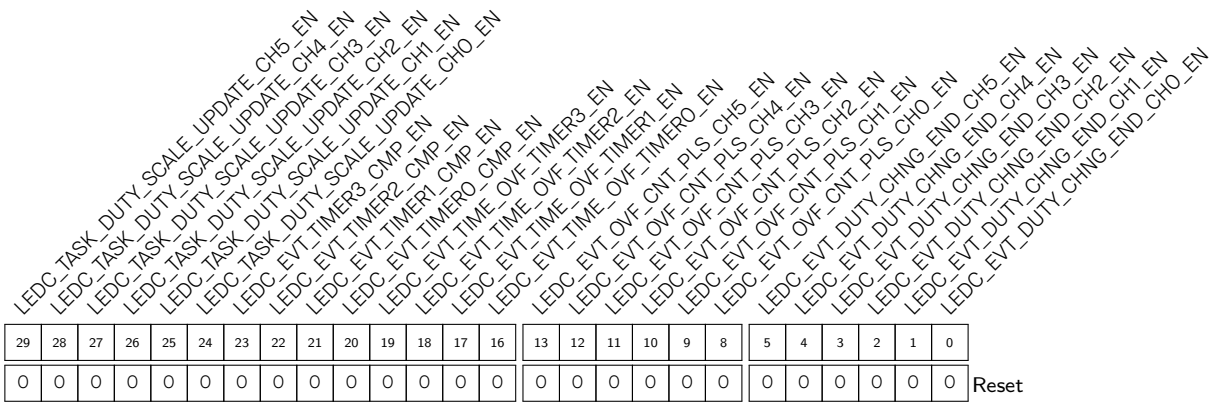
**LEDC_CH*n*_GAMMA_RESUME** Configures whether to resume duty cycle fading of LEDC channel *n*.

0: Invalid. No effect

1: Resume

(WT)

Register 32.7. LEDC_EVT_TASK_EN0_REG (0x0120)




**LEDC_EVT_DUTY_CHNG_END_CH*n*_EN (*n*: 0-5)** 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*: 0-5)** 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)

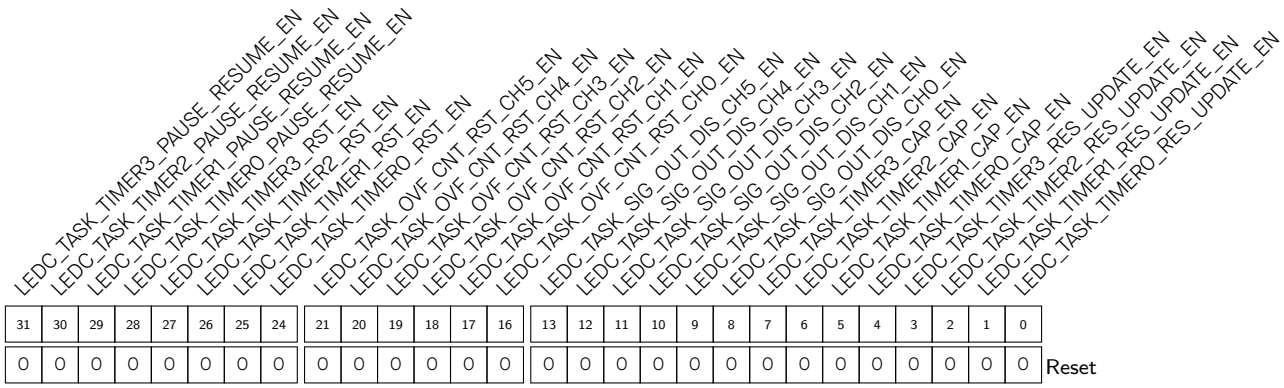
**LEDC_TASK_DUTY_SCALE_UPDATE_CH*n*_EN (*n*: 0-5)** Configures whether to enable the LEDC_TASK_DUTY_SCALE_UPDATE_CH*n* task.

0: Disable

1: Enable

(R/W)

Register 32.8. 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*: 0-5)** 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*: 0-5)** 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)

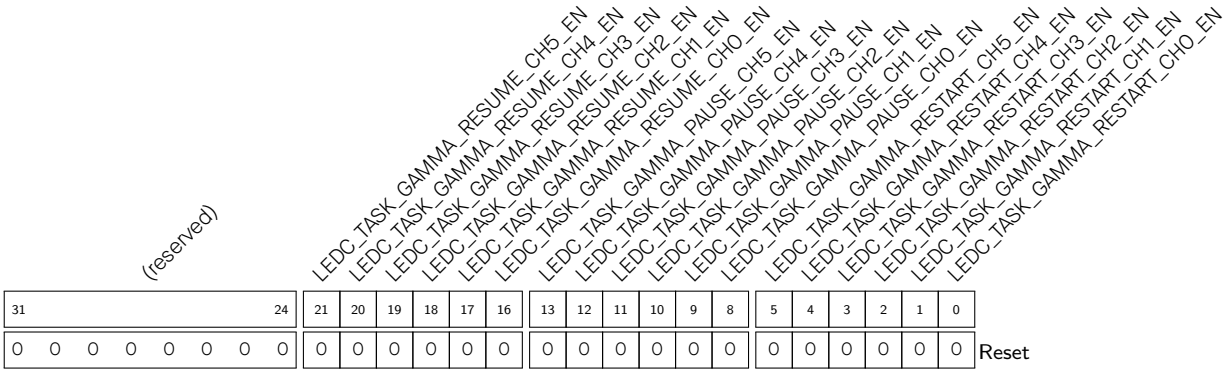
**LEDC_TASK_TIMER*x*_PAUSE_RESUME_EN (*x*: 0-3)** Configures whether to enable LEDC_TASK_TIMER*x*_PAUSE and LEDC_TASK_TIMER*x*_RESUME task.

0: Disable

1: Enable

(R/W)

Register 32.9. LEDC_EVT_TASK_EN2_REG (0x0128)




**LEDC_TASK_GAMMA_RESTART_CH*n*_EN (*n*: 0-5)** 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*: 0-5)** Configures whether to enable LEDC_TASK_GAMMA_PAUSE_CH*n* task.

0: Disable

1: Enable

(R/W)

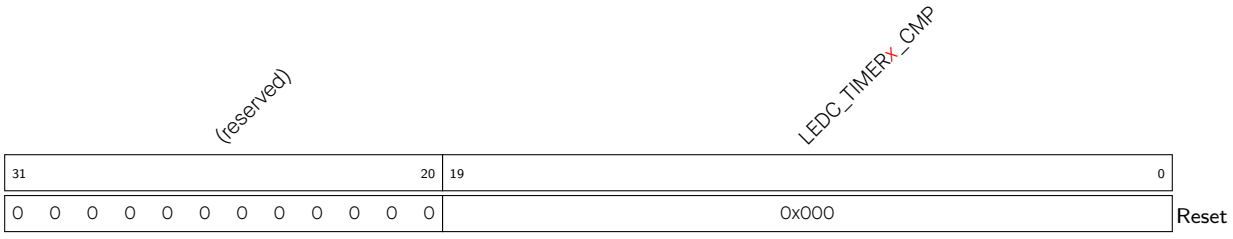
**LEDC_TASK_GAMMA_RESUME_CH*n*_EN (*n*: 0-5)** Configures whether to enable LEDC_TASK_GAMMA_RESUME_CH*n* task.

0: Disable

1: Enable

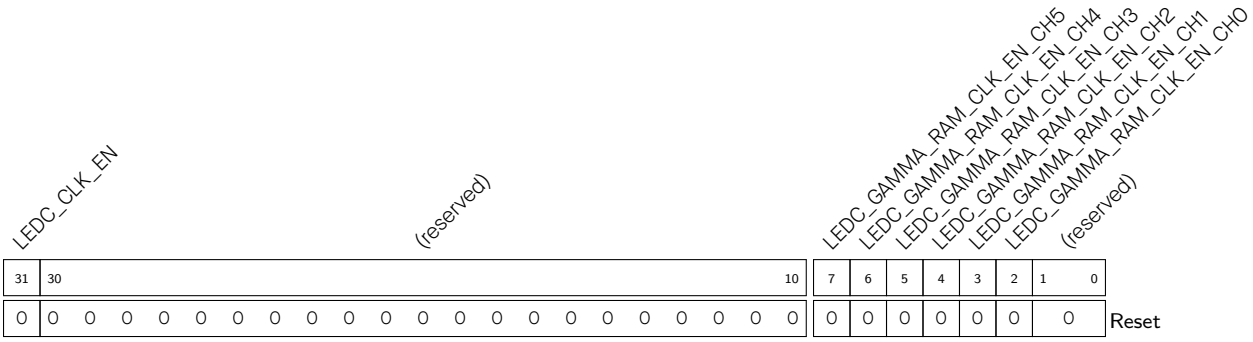
(R/W)

Register 32.10. 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 32.11. LEDC_CONF_REG (0x0170)




**LEDC_GAMMA_RAM_CLK_EN_CH*n* (*n*: 0-5)** 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)

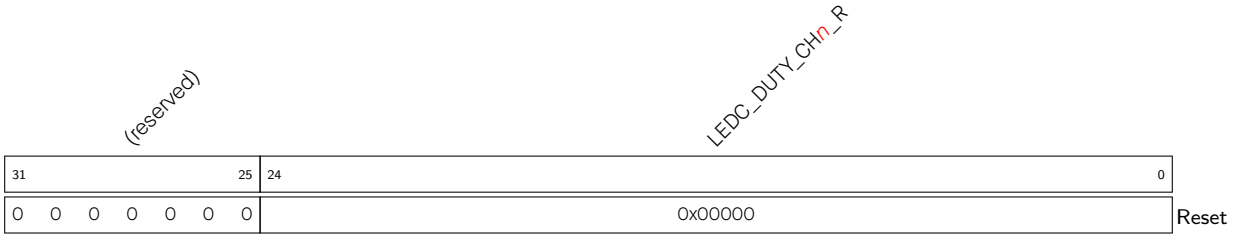
**LEDC_CLK_EN** Configures whether to open the register clock gate.

0: Open the clock gate only when an application writes registers

1: Force open the clock gate for register

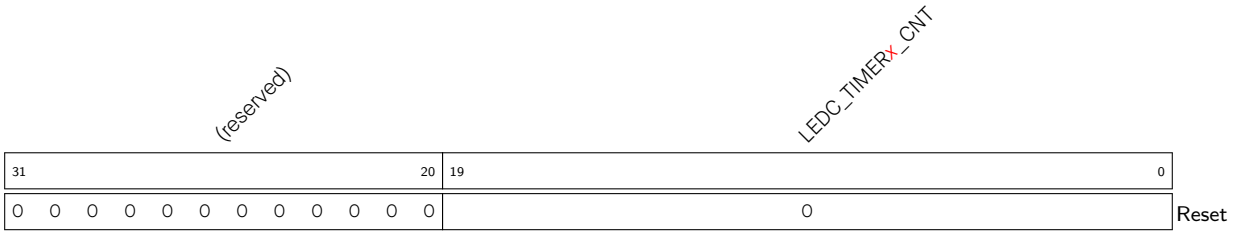
(R/W)

Register 32.12. LEDC_CH*n*_DUTY_R_REG (*n*: 0-5) (0x0010+0x14**n*)



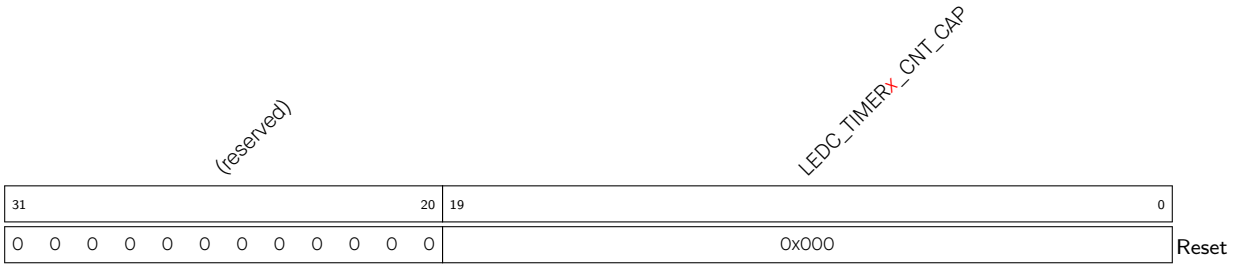

**LEDC_DUTY_CH*n*_R** Represents the current duty cycle of output signal on channel *n*. (RO)

Register 32.13. LEDC_TIMER*x*_VALUE_REG (*x*: 0-3) (0x00A4+0x8**x*)



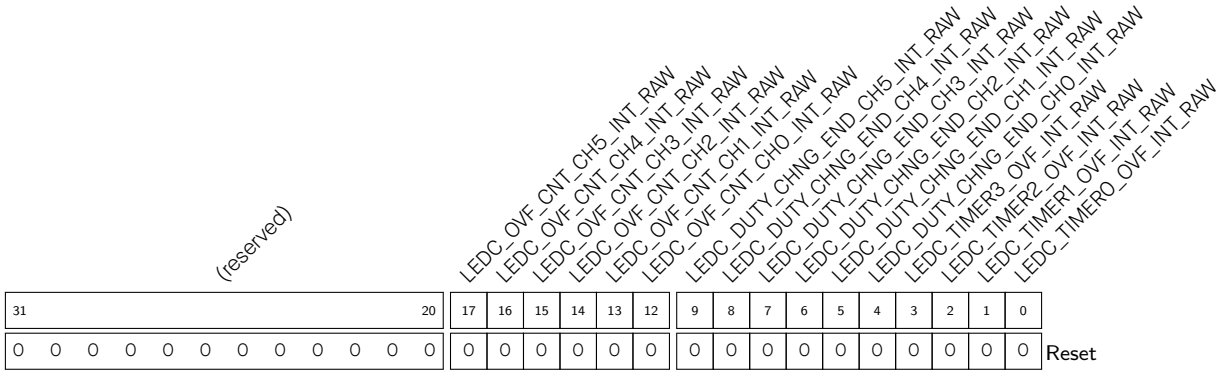

**LEDC_TIMER*x*_CNT** Represents the current counter value of timer *x*. (RO)

Register 32.14. 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 32.15. LEDC_INT_RAW_REG (0x00C0)

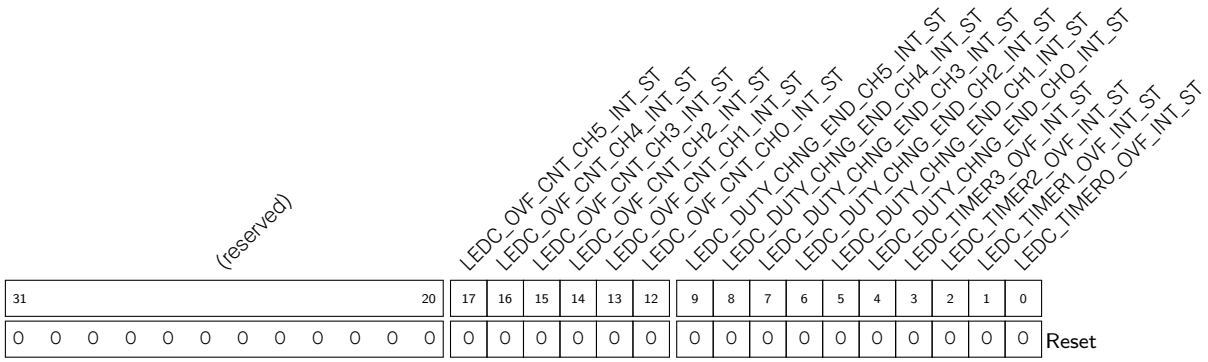



**LEDC_TIMERx_OVF_INT_RAW (x: 0-3)** Raw status bit: The raw interrupt status of LEDC_TIMERx_OVF_INT. Triggered when the timer *x* has reached its maximum counter value. (R/WTC/SS)

**LEDC_DUTY_CHNG_END_CHn_INT_RAW (n: 0-5)** 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: 0-5)** 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 32.16. LEDC_INT_ST_REG (0x00C4)

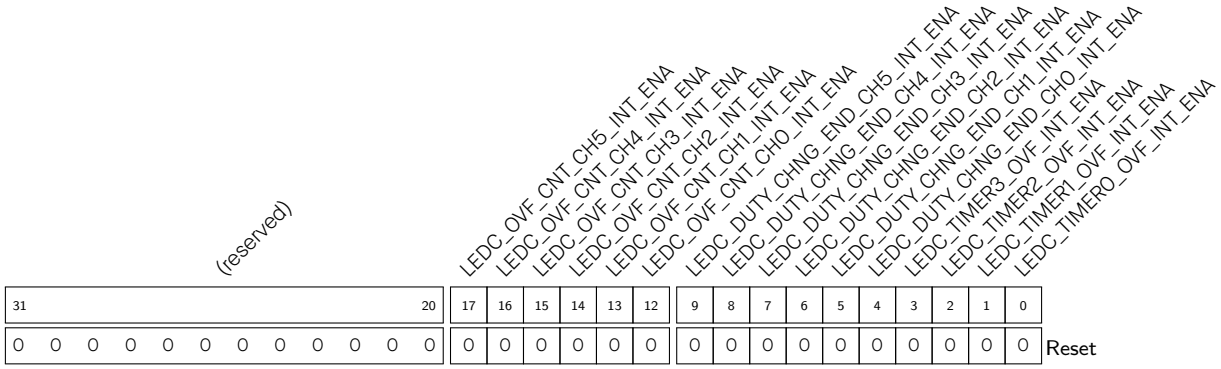



**LEDC_TIMER*x*_OVF_INT_ST (*x*: 0-3)** Masked status bit: The masked interrupt status of LEDC_TIMER*x*_OVF_INT. Valid only when LEDC_TIMER*x*_OVF_INT_ENA is set to 1. (RO)

**LEDC_DUTY_CHNG_END_CH*n*_INT_ST (*n*: 0-5)** 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*: 0-5)** 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 32.17. LEDC_INT_ENA_REG (0x00C8)

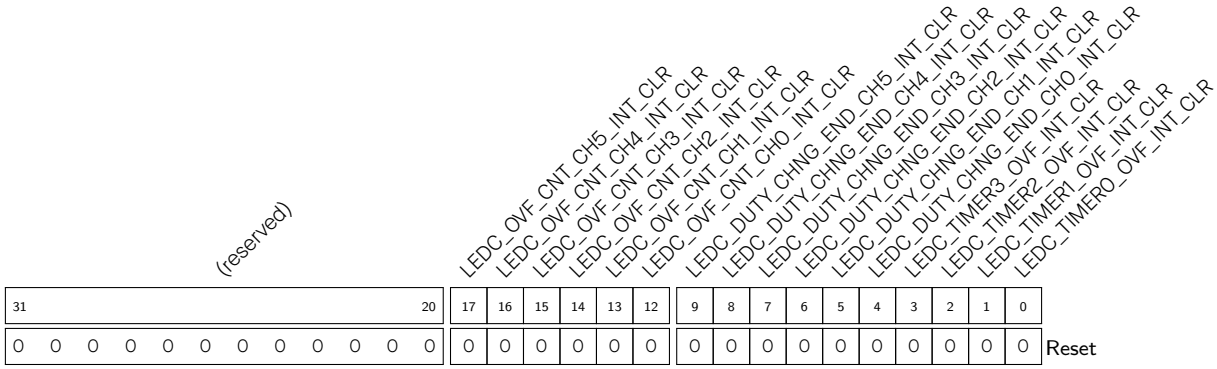



**LEDC_TIMER*x*_OVF_INT_ENA (*x*: 0-3)** Enable bit: Write 1 to enable LEDC_TIMER*x*_OVF_INT. (R/W)

**LEDC_DUTY_CHNG_END_CH*n*_INT_ENA (*n*: 0-5)** Enable bit: Write 1 to enable LEDC_DUTY_CHNG_END_CH*n*_INT. (R/W)

**LEDC_OVF_CNT_CH*n*_INT_ENA (*n*: 0-5)** Enable bit: Write 1 to enable LEDC_OVF_CNT_CH*n*_INT. (R/W)

Register 32.18. LEDC_INT_CLR_REG (0x00CC)

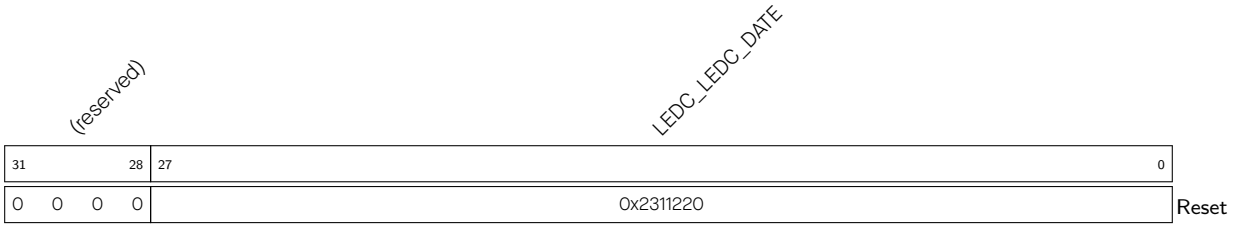



**LEDC_TIMER*x*_OVF_INT_CLR (*x*: 0-3)** Clear bit: Write 1 to clear LEDC_TIMER*x*_OVF_INT. (WT)

**LEDC_DUTY_CHNG_END_CH*n*_INT_CLR (*n*: 0-5)** Clear bit: Write 1 to clear LEDC_DUTY_CHNG_END_CH*n*_INT. (WT)

**LEDC_OVF_CNT_CH*n*_INT_CLR (*n*: 0-5)** Clear bit: Write 1 to clear LEDC_OVF_CNT_CH*n*_INT. (WT)

Register 32.19. LEDC_DATE_REG (0x0174)




**LEDC_LEDC_DATE** Configures the version. (R/W)

# Part VI

# Analog Signal Processing

This part describes components related to analog-to-digital conversion, voltage comparison, on-chip sensors, and features such as temperature sensing, demonstrating the system's capabilities in handling analog signals.

# Chapter 33

# Temperature Sensor

## 33.1 Overview

ESP32-C61 provides a temperature sensor to monitor temperature changes inside the chip in real time. The sensor converts analog voltage to digital values and supports compensation for the temperature offset.

## 33.2 Features

The temperature sensor has the following features:

- Software-triggered temperature measurement. Once triggered, the sensor continuously measures temperature. Software can read the data any time.
- Hardware-triggered automatic temperature monitoring
- Two modes for automatic monitoring of temperature and support for triggering interrupts
- Configurable temperature offset based on the application scenario for improved accuracy
- Configurable temperature measurement range
- Support for several Event Task Matrix (ETM) related events and tasks

## 33.3 Architecture

Figure 33.3-1 shows the internal structure of the temperature sensor.

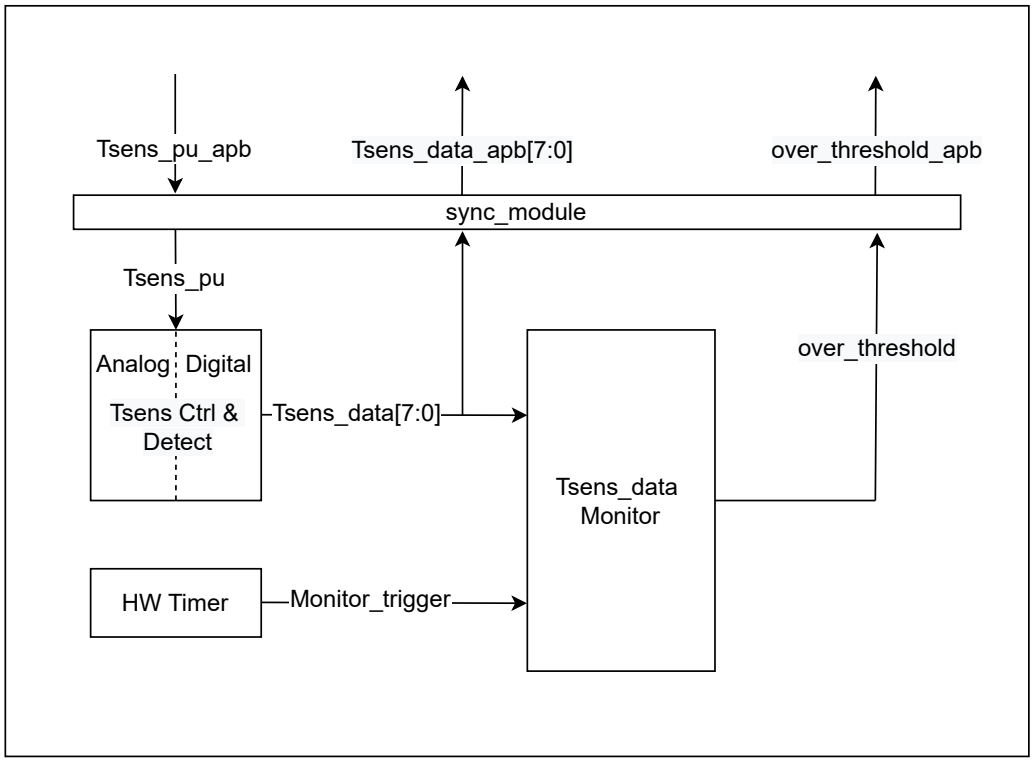



Figure 33.3-1. Temperature Sensor Architecture

As Figure 33.3-1 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

## 33.4 Functional Description

### 33.4.1 Temperature Sensor Power Up

The temperature sensor can be powered up by setting the APB_SARADC_TSENS_PU register.

### 33.4.2 Temperature Sensor Clock

The temperature sensor has two clock sources: RC_FAST_CLK and XTAL_CLK, selected by PCR_TSENS_CLK_SEL. The clock can be divided with APB_SARADC_TSENS_CLK_DIV.

### 33.4.3 Automatic Temperature Monitoring Modes

Two modes for the automatic temperature monitoring are available for selection by configuring APB_SARADC_WAKEUP_MODE:

- Absolute value mode:

  - Monitors the absolute value of the current temperature. Configure APB_SARADC_WAKEUP_TH_LOW and APB_SARADC_WAKEUP_TH_HIGH to set the temperature thresholds. An interrupt 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 APB_SARADC_WAKEUP_TH_HIGH, or the temperature decrement of two consecutive samplings exceeds the low threshold configured in APB_SARADC_WAKEUP_TH_LOW, an interrupt will be triggered. For example, when APB_SARADC_WAKEUP_TH_LOW is configured as 8, if two consecutive sampling values are 28 and 19 respectively, i.e., the temperature decrement is 9, then an interrupt will be triggered.

### 33.4.4 Temperature Measurement Range and Offset

To improve the temperature measurement accuracy, the temperature sensor is designed with five measurement ranges as shown in Table 33.4-1. Each measurement range comes with specific measurement errors. Users do not need to worry about the errors, as they can choose specific offsets to get calibrated data.

**Table 33.4-1. 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 |

### 33.4.5 Data Conversion

The sensor output value is stored in APB_SARADC_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 shown in Table 33.4-1.

## 33.5 Programming Procedure

The temperature sensor can be started by software as follows:

1. Set APB_SARADC_TSENS_PU to power up the temperature sensor.
2. Set PCR_TSENS_CLK_EN to enable the temperature sensor clock.
3. Configure APB_SARADC_TSENS_CLK_SEL to select the temperature sensor clock.
4. Wait for APB_SARADC_TSENS_XPD_WAIT clock cycles till the temperature sensor releases from reset and starts measuring the temperature.

5. Wait for 100 $\mu$s (so that the output value gradually approaches the actual temperature) and then read the data from APB_SARADC_TSENS_OUT.

To enable hardware-triggered automatic temperature monitoring, add the following programming steps before powering up the sensor:

1. Configure APB_SARADC_TSENS_SAMPLE_RATE to set sampling rate.
2. Configure APB_SARADC_WAKEUP_MODE to select an automatic temperature monitoring mode.
3. Configure APB_SARADC_WAKEUP_TH_HIGH/LOW to set the high and low temperature monitoring thresholds.
4. Set APB_SARADC_WAKEUP_EN to start temperature monitoring.
5. Set APB_SARADC_TSENS_SAMPLE_EN to enable automatic temperature monitoring.

In automatic temperature monitoring mode, the output value will not be stored. However, users can get the value anytime from APB_SARADC_TSENS_OUT.

## 33.6 Interrupts

ESP32-C61'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:

- APB_SARADC_TSENS_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 10 *Interrupt Matrix* > Section 10.2 *Interrupt Terminology in ESP32-C61*.

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 33.8 *Register Summary*.

## 33.7 Event Task Matrix Feature

The temperature sensor on ESP32-C61 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 11 *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, 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.

## 33.8 Register Summary

The addresses in this section are relative to Temperature Sensor base address provided in Table 5.3-2 in Chapter 5 *System and Memory*.

The abbreviations given in Column **Access** are explained in Section *Access Types for Registers*.

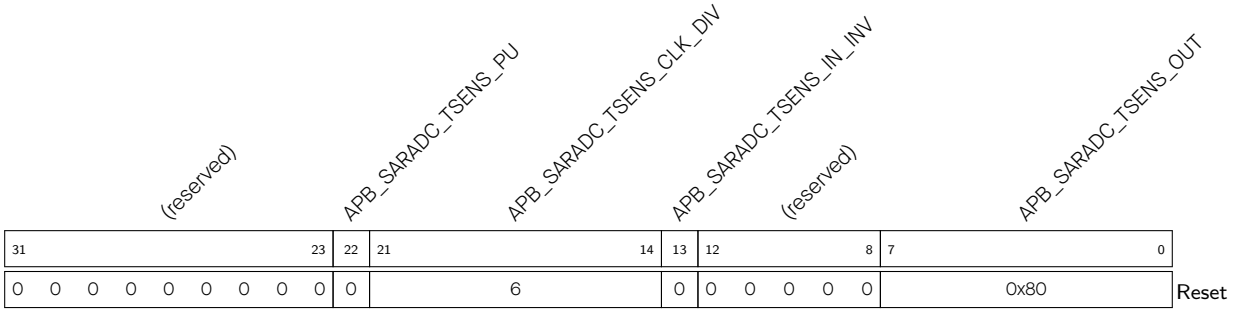
| Name | Description | Address | Access |
|---|---|---|---|
| **Configuration Registers** | | | |
| APB_SARADC_INT_ENA_REG | Enable register of SAR ADC/temperature sensor interrupts | 0x0040 | R/W |
| APB_SARADC_INT_RAW_REG | Raw register of SAR ADC/temperature sensor interrupts | 0x0044 | R/WTC/SS |
| APB_SARADC_INT_ST_REG | State register of SAR ADC/temperature sensor interrupts | 0x0048 | RO |
| APB_SARADC_INT_CLR_REG | Clear register of SAR ADC/temperature sensor interrupts | 0x004C | WT |
| APB_SARADC_APB_TSENS_CTRL_REG | Temperature sensor control register 1 | 0x0058 | varies |
| APB_SARADC_APB_TSENS_CTRL2_REG | Temperature sensor control register 2 | 0x005C | R/W |
| APB_TSENS_WAKE_REG | Temperature sensor automatic monitoring mode configuration register | 0x0064 | varies |
| APB_TSENS_SAMPLE_REG | Temperature sensor configuration register | 0x0068 | R/W |
| **Version Control Register** | | | |
| APB_SARADC_CTRL_DATE_REG | Version control register | 0x03FC | R/W |

## 33.9 Registers

The addresses in this section are relative to Temperature Sensor base address provided in Table 5.3-2 in Chapter 5 *System and Memory*.

For how to program reserved fields, please refer to Section *Programming Reserved Register Field*.

**Register 33.1. APB_SARADC_APB_TSENS_CTRL_REG (0x0058)**




**APB_SARADC_TSENS_OUT** Stores the temperature sensor output value. (RO)

**APB_SARADC_TSENS_IN_INV** Configures whether to invert temperature sensor data.

0: No effect

1: Invert

(R/W)

**APB_SARADC_TSENS_CLK_DIV** Configures temperature sensor clock division. (R/W)

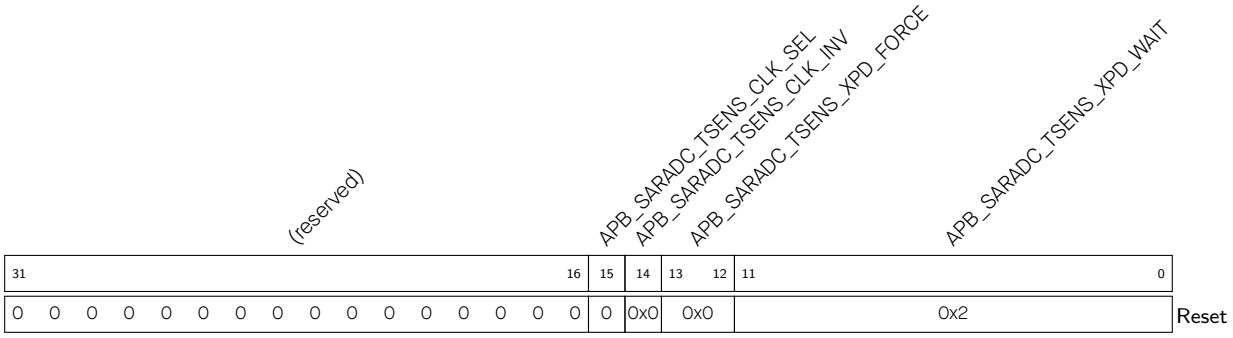
**APB_SARADC_TSENS_PU** Configures whether to power up temperature sensor.

0: No effect

1: Power up

(R/W)

Register 33.2. APB_SARADC_APB_TSENS_CTRL2_REG (0x005C)




**APB_SARADC_TSENS_CLK_SEL** Configures the working clock for temperature sensor.

0: RC_FAST_CLK

1: XTAL_CLK

(R/W)

**APB_SARADC_TSENS_CLK_INV** Configures the phase of the temperature sensor clock. (R/W)

**APB_SARADC_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

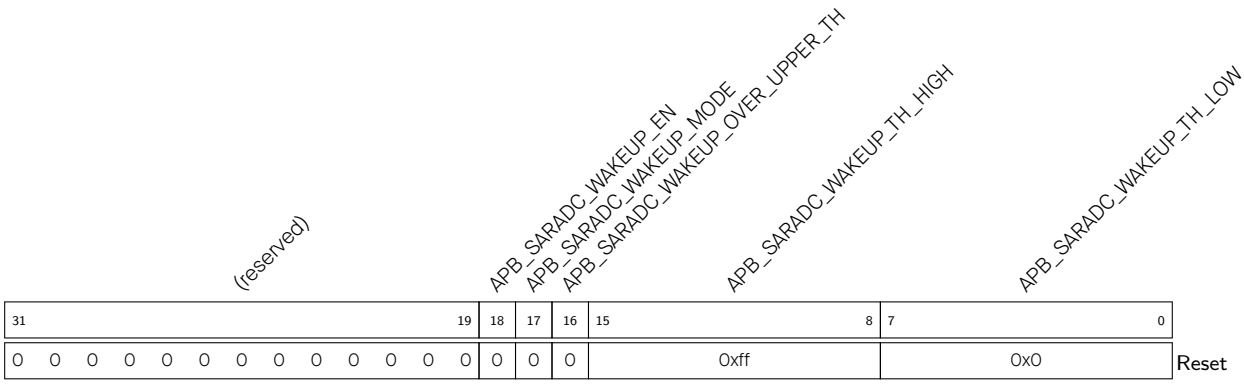
2: Enable force power up temperature sensor

3: Enable force power down temperature sensor

(R/W)

**APB_SARADC_TSENS_XPD_WAIT** Configure the wait time (in clock cycles) for the senor to release from reset. (R/W)

Register 33.3. APB_TSENS_WAKE_REG (0x0064)




**APB_SARADC_WAKEUP_TH_LOW** Configures the low threshold for automatic temperature monitoring. (R/W)

**APB_SARADC_WAKEUP_TH_HIGH** Configures the high threshold for automatic temperature monitoring. (R/W)

**APB_SARADC_WAKEUP_OVER_UPPER_TH** Represents whether the temperature output value exceeds the threshold. Valid only when APB_SARADC_WAKEUP_EN=1.

0: The temperature output value is below the low threshold

1: The temperature output value is above the high threshold

(RO)

**APB_SARADC_WAKEUP_MODE** Selects the temperature monitoring mode.

0: Absolute value mode

1: Change value mode

(R/W)

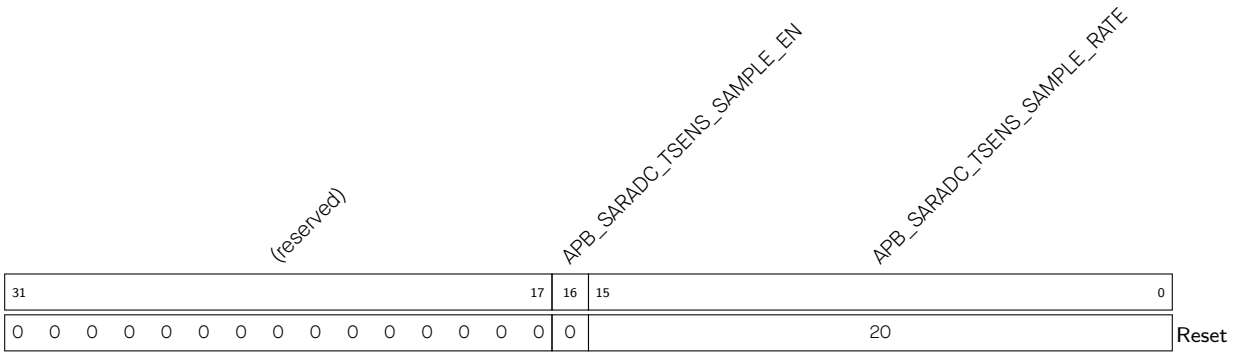
**APB_SARADC_WAKEUP_EN** Configures whether to enable the temperature monitoring function.

0: Disable

1: Enable

(R/W)

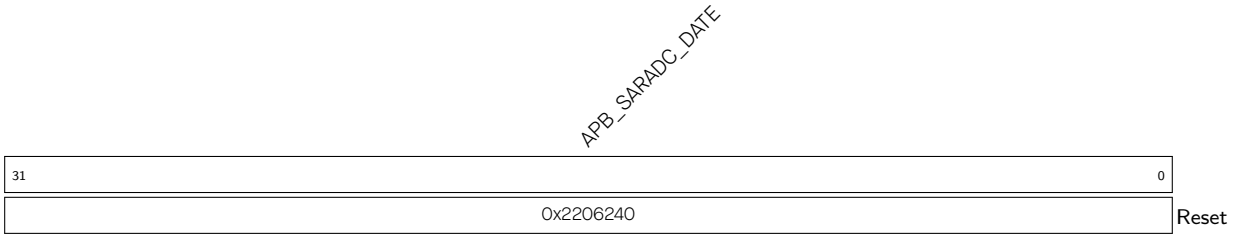
Register 33.4. APB_TSENS_SAMPLE_REG (0x0068)




**APB_SARADC_TSENS_SAMPLE_RATE** Configures the sampling rate for hardware-triggered automatic temperature monitoring. The sampling period = configured value × the sensor's working clock cycle. (R/W)

**APB_SARADC_TSENS_SAMPLE_EN** Configures whether to enable automatic temperature monitoring.
0: Disable
1: Enable
(R/W)

Register 33.5. APB_SARADC_CTRL_DATE_REG (0x03FC)




**APB_SARADC_DATE** Version control register for SAR ADC (please refer to Chapter 34 *ADC Controller* for more information) and the temperature sensor. (R/W)

# Chapter 34

# ADC Controller

## 34.1 Overview

ESP32-C61 integrates one 12-bit successive approximation ADC (SAR ADC) for measuring analog signals from up to four channels.

The SAR ADC is managed by the DIG ADC controller that drives ADC sampling. The SAR ADC supports one-shot sampling and multi-channel sampling.

## 34.2 Terminology

The following terms related to SAR ADC are defined in the context of the *ESP32-C61 Technical Reference Manual* to help readers better understand this document:

**SAR ADC:** Including analog ADC circuit and digital ADC controller.

**One-shot sampling mode:** In this mode, ADC samples one channel at a time.

**Multi-channel sampling mode:** In this mode, ADC sequentially samples a group of channels or continuously samples a single channel.

**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.

## 34.3 Feature List

The SAR ADC has the following features:

- 12-bit resolution
- Analog inputs sampling from up to four pins
- One-shot sampling mode and multi-channel sampling mode
- Multi-channel sampling mode supports:

  - configurable channel sampling sequence
  - two filters whose filter coefficients are configurable
  - two threshold monitors that can trigger an interrupt when the filtered value is below a low threshold or above a high threshold
  - continuous transfer of converted data to memory via GDMA interface
- Support for several Event Task Matrix (ETM) related events and tasks

## 34.4 Architectural Overview

The major components of SAR ADC and their interconnections are shown in Figure 34.4-1.

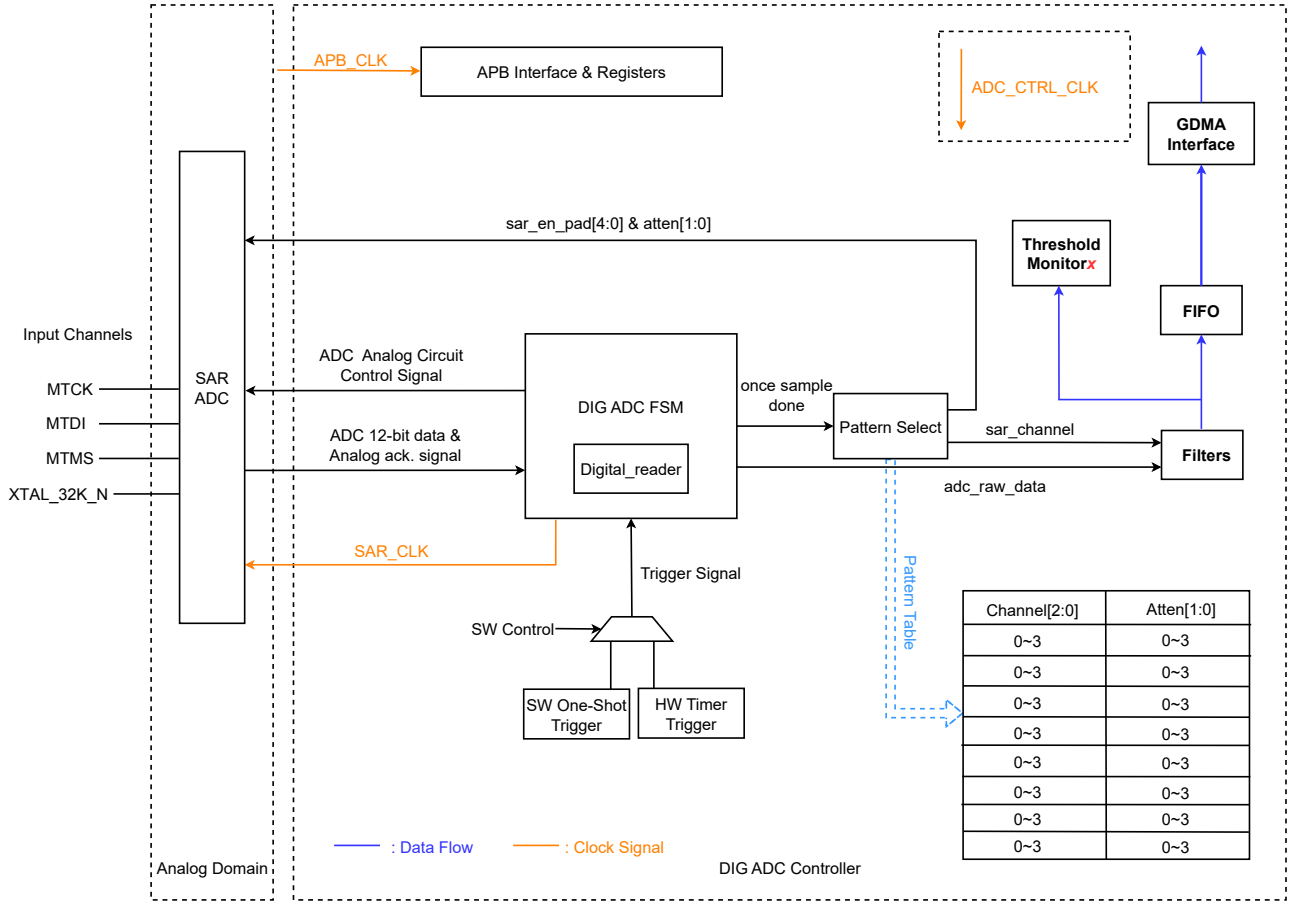



Figure 34.4-1. SAR ADC Architecture

As Figure 34.4-1 shows, the SAR ADC module contains the following major functional blocks:

- four channels, connected to four pins on the chip
- SAR ADC: analog domain of the SAR ADC module
- DIG ADC Controller: digital domain of the SAR ADC module, mainly including:
  - Clock management module: selects clock source and division
  - DIG ADC FSM: generates the signals required throughout the ADC sampling process

- Digital_reader: reads data from SAR ADC, driven by DIG ADC FSM
- Filter: filters ADC converted data in multi-channel sampling mode
- Threshold monitor*x*: threshold monitor 1 and threshold monitor 2. The monitor*x* will trigger an interrupt when the converted value is below a low threshold or above a high threshold.

# 34.5 Functional Description

## 34.5.1 ADC Power Up

The ADC can be powered up by setting PMU_XPD_PERIF_I2C and PMU_PERIF_I2C_RST. ADC sampling can be performed right after power-up. Users do not need to worry about wait time, as it has been dealt with in hardware design.

## 34.5.2 ADC Channels

The SAR ADC has four channels that are connected to four pins on the chip. In order to sample an analog signal, the SAR ADC must first select the analog pin to measure via an internal multiplexer.

Table 34.5-1 shows the pins used as ADC channels and the corresponding GPIO numbers.

**Table 34.5-1. SAR ADC Channels**

| Pin Name | GPIO Number | ADC Channel |
|---|---|---|
| XTAL_32K_N | GPIO1 | 0 |
| MTMS | GPIO3 | 1 |
| MTDI | GPIO4 | 2 |
| MTCK | GPIO5 | 3 |

## 34.5.3 ADC Clock

Figure 34.5-1 shows the clock structure of SAR ADC.

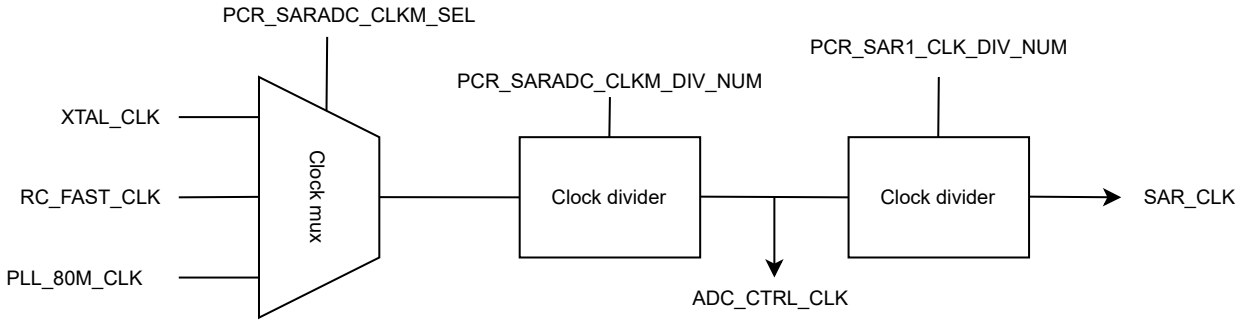



**Figure 34.5-1. SAR ADC Clock Structure**

The system clock for SAR ADC is ADC_CTRL_CLK, which is the operating clock for DIG ADC FSM and other control logic (except APB interface and Digital_reader).

ADC_CTRL_CLK has three possible sources: XTAL_CLK, RC_FAST_CLK, and PLL_80M_CLK, selected by PCR_SARADC_CLKM_SEL.

SAR_CLK is the operating clock for SAR ADC and Digital_reader. It is divided from ADC_CTRL_CLK and must not exceed 5 MHz.

For more information about clocks, please refer to Chapter 8 *Reset and Clock*.

### 34.5.4 One-Shot Sampling Mode

In one-shot sampling mode, the ADC samples one channel once. This mode is started by software using APB_SARADC_ONETIME_SAMPLE. Once sampling is done, the conversion result is stored in APB_SARADC_DATA. To switch channels, configure APB_SARADC_ONETIME_SAMPLE once again.

### 34.5.5 Multi-Channel Sampling Mode

Multi-channel sampling mode is triggered by a timer that is specifically designed for SAR ADC. In this mode the ADC samples a group of channels according to the sequence defined in the pattern table. The multi-channel sampling mode can also be used for continuous sampling on one channel.

The timer is enabled by setting APB_SARADC_TIMER_EN. A trigger target for the timer needs to be configured with APB_SARADC_TIMER_TARGET. When the timer counts up to two times of APB_SARADC_TIMER_TARGET, a sampling operation is triggered. The timer is clocked from ADC_CTRL_CLK.

When sampling is complete, the timer resets to 0 and starts counting again. The conversion result is transferred to memory continuously via the GDMA interface.

**Note:**
The SAR ADC can only work under one operating mode at one time, either one-shot sampling mode or multi-channel sampling mode.

### 34.5.6 ADC Conversion and Attenuation

The SAR ADC can measure analog voltages from 0 mV to $V_{ref}$. $V_{ref}$ is the SAR ADC's internal reference voltage (1100 mV by design). The conversion result ($data$) is a 12-bit digital value, the raw data. To calculate the voltage $V_{data}$ based on the raw data, please use the formula below:

$$V_{data} = \frac{V_{ref}}{k} \times \frac{data}{4095}$$

$k$ is the coefficient corresponding to the configured attenuation.

To convert voltages larger than $V_{ref}$, apply attenuation to the input signals using APB_SARADC_ONETIME_ATTEN. The supported attenuation levels are:

- 0 dB (k≈100%)
- 2.5 dB (k≈75%)
- 6 dB (k≈50%)
- 12 dB (k≈25%)

### 34.5.7 DIG ADC FSM

In multi-channel sampling mode DIG ADC FSM (hereinafter referred to as FSM) generates all types of signals used in the sampling process. Figure 34.5-2 illustrates how DIG ADC FSM works.

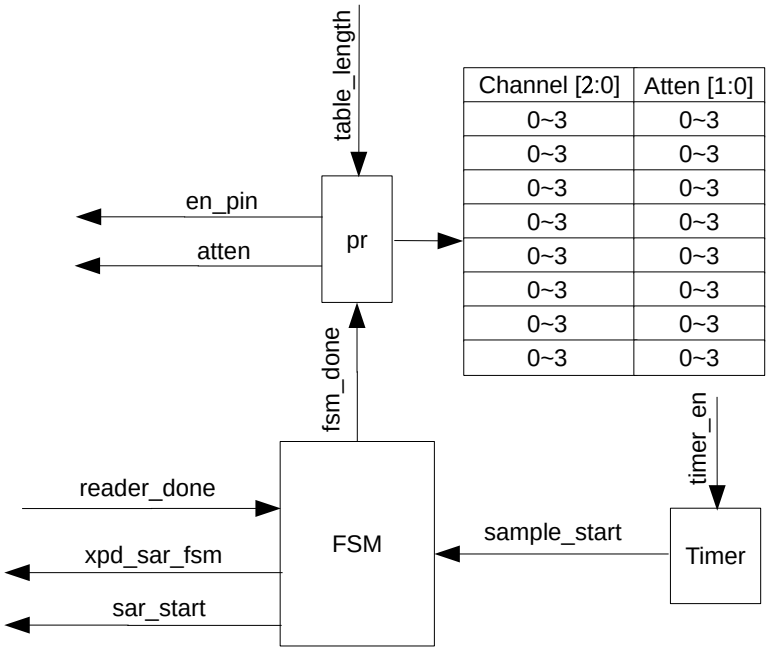



Figure 34.5-2. DIG ADC FSM Block Diagram

Wherein:

- Timer: a dedicated timer for the DIG ADC controller to generate `sample_start` signal.
- pr: a pointer to the pattern table that defines the conversion rules for ADC. FSM sends out corresponding signals based on the conversion rules.

Set APB_SARADC_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 SAR 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.

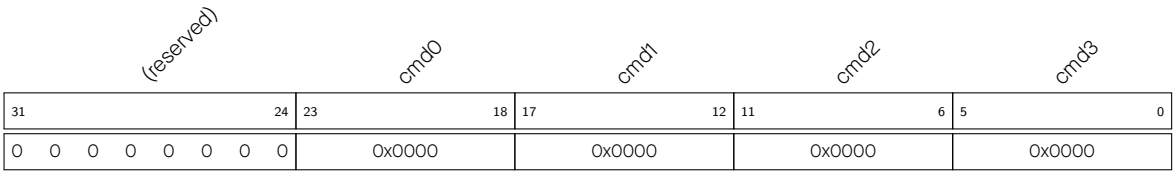
When FSM receives the `reader_done` signal from Digital_reader, it starts the following operations:

- Stops sampling.
- Transfers the conversion result to the filter. Then the threshold monitor transfers the filtered result to memory via GDMA (see Figure 34.4-1).

- Updates pr and waits for the next sampling. The pointer pr counts cyclically between 0 and APB_SARADC_SAR_PATT_LEN (table_length).

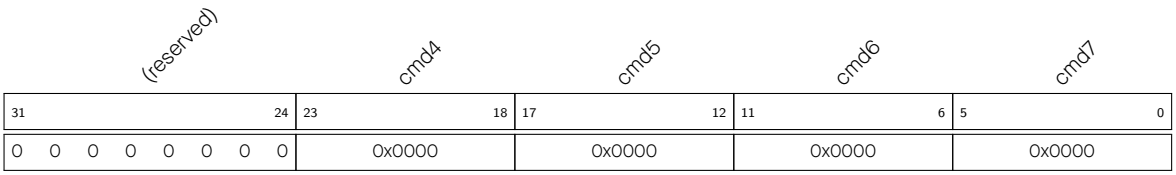
### 34.5.8 Pattern Table

FSM contains a pattern table consisting of the APB_SARADC_SAR_PATT_TAB1_REG and APB_SARADC_SAR_PATT_TAB2_REG registers. Each register contains four patterns and each pattern is 6 bits wide, as Figure 34.5-3 and Figure 34.5-4 show.




**cmd** *x* represents patterns 0 ~ 3.

**Figure 34.5-3. APB_SARADC_SAR_PATT_TAB1_REG Contains Patterns 0 - 3**




**cmd** *x* represents patterns 4 ~ 7.

**Figure 34.5-4. APB_SARADC_SAR_PATT_TAB2_REG Contains Patterns 4 - 7**

Each pattern is 6 bits wide, consisting of three fields where conversion rules are stored. Figure 34.5-5 shows the pattern format and is followed by the descriptions of each field.




**Figure 34.5-5. Pattern Structure**

**atten** Configures attenuation:

0: 0 dB
1: 2.5 dB
2: 6 dB
3: 12 dB

**ch_sel** Configures channel:

0: Channel 0
1: Channel 1
2: Channel 2
3: Channel 3

**(reserved)** Reserved

#### Pattern Configuration Example

Note: When using multi-channel continuous sampling, the attenuation for each channel must be set to be consistent. In this example, two channels are selected for multi-channel sampling:

- Channel 0, with an attenuation of 2.5 dB
- Channel 2, with an attenuation of 2.5 dB

The detailed configuration is as follows:

- Configure the first pattern (cmd0):




Figure 34.5-6. cmd0 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 the second pattern (cmd1):




Figure 34.5-7. cmd1 Configuration

**atten** write 1 to this field, to set the attenuation to 2.5 dB.

**ch_sel** write 2 to this field, to select channel 2.

- Configure APB_SARADC_SAR_PATT_LEN to 1. Then patterns cmd0 and cmd1 will be used.
- Enable the timer so that the DIG ADC controller starts sampling the two channels periodically.

### 34.5.9 ADC Filters

The DIG ADC controller provides two filters for filtering ADC converted data in multi-channel sampling mode. Both filters can be configured to any ADC channel, but not to the same one. 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 ADC converted data
- $data_{prev}$: the last filtered data

- $k$: the filter coefficient

The filters are configured as follows:

- Configure APB_SARADC_FILTER_CHANNEL*x* to select the ADC channel for filter *x* (*x*=0, 1).
- Configure APB_SARADC_FILTER_FACTOR*x* to set the coefficient $k$ for filter *x*.

## 34.5.10 Threshold Monitors

Two threshold monitors are available in the DIG ADC controller 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 ADC channel, but cannot be configured to the same channel.

Threshold monitors are configured as follows:

- Set APB_SARADC_THRES*x*_EN to enable threshold monitor *x* (*x*=0, 1).
- Configure APB_SARADC_THRES*x*_LOW to set a low threshold.
- Configure APB_SARADC_THRES*x*_HIGH to set a high threshold.
- Configure APB_SARADC_THRES*x*_CHANNEL to select the channel to monitor.
- Set APB_SARADC_THRES_ALL_EN to enable threshold monitoring functionality.

## 34.5.11 GDMA Support

As SAR ADC has only one data register APB_SARADC_DATA for storing converted result from one-shot sampling, it is necessary to enable GDMA when converting data on multiple channels so that the data can be transferred to memory continuously.

GDMA support is triggered by the timer in the DIG ADC controller. Users can switch the DMA data path to the DIG ADC controller by configuring APB_SARADC_APB_ADC_TRANS. For specific DMA configuration, please refer to Chapter 4 *GDMA Controller (GDMA)*.

### GDMA Data Format

The ADC eventually passes 32-bit data to GDMA. The data format is shown in Figure 34.5-8.

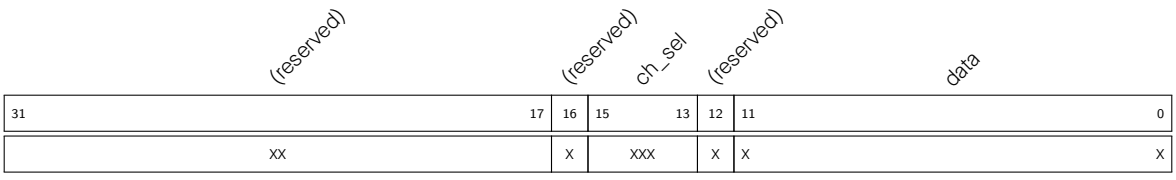



Figure 34.5-8. DMA Data Format

**data** 12-bit ADC conversion result

**ch_sel** 3-bit channel information

## 34.6 Event Task Matrix Feature

The SAR ADC on ESP32-C61 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 and temperature sensor. For more information, please refer to Chapter 11 Event Task Matrix (ETM).

The SAR ADC can receive the following ETM tasks:

- ADC_TASK_SAMPLE0: ADC starts one-shot sampling when this task is triggered.
- ADC_TASK_START0: ADC starts multi-channel sampling when this task is triggered.
- ADC_TASK_STOP0: ADC stops sampling when this task is triggered.

The SAR ADC can generate the following ETM events:

- ADC_EVT_CONV_CMPLT0: Generated each time ADC completes a sampling in either one-shot sampling mode or multi-channel sampling mode.
- ADC_EVT_EQ_ABOVE_THRESH*x*: Generated when the ADC filtered data is above the threshold. *x* = 0, 1, representing threshold monitor 0, 1.
- ADC_EVT_EQ_BELOW_THRESH*x*: Generated when the ADC filtered data is below the threshold. *x* = 0, 1, representing threshold monitor 0, 1.
- ADC_EVT_STARTED0: Generated when ADC begins sampling; one-shot sampling will not trigger this event.
- ADC_EVT_STOPPED0: Generated when 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.

## 34.7 Interrupts

ESP32-C61's SAR ADC can generate the APB_ADC_INTR interrupt signal that will be sent to the Interrupt Matrix. The following internal interrupt sources from SAR ADC can generate the interrupt signal APB_ADC_INTR:

- APB_SARADC_ADC_DONE_INT: Triggered when SAR ADC completes one conversion.
- APB_SARADC_THRES*x*_HIGH_INT: Triggered when the filtered data is above the high threshold of monitor *x*.
- APB_SARADC_THRES*x*_LOW_INT: Triggered when the filtered data is below the low threshold of monitor *x*.

**Note:**
For definitions of *interrupt*, *interrupt signal*, *interrupt source*, and their correlations, please refer to Chapter 10 *Interrupt Matrix* > Section 10.2 *Interrupt Terminology in ESP32-C61*.

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 34.9 *Register Summary*.

## 34.8 Programming Procedure

### 34.8.1 Configuring 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_RST to power up SAR ADC.
2. Configure PCR_SARADC_CLKM_SEL to select ADC clock source.
3. Configure PCR_SARADC_CLKM_DIV_NUM and PCR_SAR1_CLK_DIV_NUM to set clock division.
4. Set PCR_SARADC_CLKM_EN to enable ADC clock.
5. Set APB_SARADC_ONETIME_SAMPLE to enable the one-shot sampling mode.
6. Configure APB_SARADC_ONETIME_CHANNEL to select sampled channel.
7. Configure APB_SARADC_ONETIME_ATTEN to set attenuation as needed.
8. Set APB_SARADC_ONETIME_START to start one-shot sampling.

Once sampling is complete, an APB_SARADC_ADC_DONE_INT_RAW interrupt is generated. Software can read conversion result from APB_SARADC_ADC_DATA. To switch the sampled channel, program from step 6.

### 34.8.2 Configuring 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 SAR ADC.
2. Configure PCR_SARADC_CLKM_SEL to select ADC clock source.
3. Configure PCR_SARADC_CLKM_DIV_NUM and PCR_SAR1_CLK_DIV_NUM to set clock division.
4. Set PCR_SARADC_CLKM_EN to enable ADC clock.
5. Configure the pattern table as described in Section 34.5.8 *Pattern Table*.
6. Configure channels and filtering coefficients as described in Section 34.5.9 *ADC Filters*.
7. Configure threshold monitoring as described in Section 34.5.10 *Threshold Monitors*, as needed.
8. Set APB_SARADC_APB_ADC_TRANS to use GDMA.
9. Configure APB_SARADC_TIMER_TARGET to set trigger target for DIG ADC timer.
10. Set APB_SARADC_TIMER_EN to enable the timer.

The timer timeout will trigger DIG ADC FSM to start sampling according to the pattern table. The conversion result will be automatically stored in memory. When the sampling reaches the number of limit set in APB_SARADC_APB_ADC_EOF_NUM, it will terminate.

Once sampling is complete, an APB_SARADC_ADC_DONE_INT_RAW interrupt will be generated.

## 34.9 Register Summary

The addresses in this section are relative to SAR ADC Controller base address provided in Table 5.3-2 in Chapter 5 *System and Memory*.

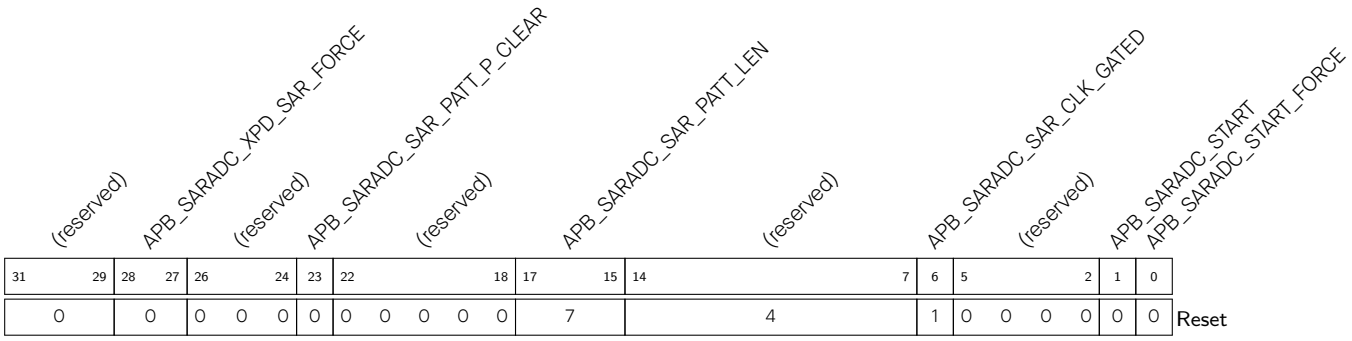
The abbreviations given in Column **Access** are explained in Section *Access Types for Registers*.

| Name | Description | Address | Access |
|---|---|---|---|
| **Configuration Registers** | | | |
| APB_SARADC_CTRL_REG | SAR ADC control register 1 | 0x0000 | R/W |
| APB_SARADC_CTRL2_REG | SAR ADC control register 2 | 0x0004 | R/W |
| APB_SARADC_FILTER_CTRL1_REG | Filtering control register 1 | 0x0008 | R/W |
| APB_SARADC_SAR_PATT_TAB1_REG | Pattern table register 1 | 0x0018 | R/W |
| APB_SARADC_SAR_PATT_TAB2_REG | Pattern table register 2 | 0x001C | R/W |
| APB_SARADC_ONETIME_SAMPLE_REG | Configuration register for one-shot sampling | 0x0020 | R/W |
| APB_SARADC_FILTER_CTRL0_REG | Filtering control register 0 | 0x0028 | R/W |
| APB_SARADC_SAR1DATA_STATUS_REG | SAR ADC conversion data storage register | 0x002C | RO |
| APB_SARADC_THRES0_CTRL_REG | Filtered data threshold control register 0 | 0x0034 | R/W |
| APB_SARADC_THRES1_CTRL_REG | Filtered data threshold control register 1 | 0x0038 | R/W |
| APB_SARADC_THRES_CTRL_REG | Filtered data threshold control register | 0x003C | R/W |
| APB_SARADC_INT_ENA_REG | Enable register of SAR ADC interrupts | 0x0040 | R/W |
| APB_SARADC_INT_RAW_REG | Raw register of SAR ADC interrupts | 0x0044 | R/WTC/SS |
| APB_SARADC_INT_ST_REG | State register of SAR ADC interrupts | 0x0048 | RO |
| APB_SARADC_INT_CLR_REG | Clear register of SAR ADC interrupts | 0x004C | WT |
| APB_SARADC_DMA_CONF_REG | DMA configuration register for SAR ADC | 0x0050 | R/W |
| APB_SARADC_CTRL_DATE_REG | Version control register | 0x03FC | R/W |

## 34.10 Registers

The addresses in this section are relative to SAR ADC Controller base address provided in Table 5.3-2 in Chapter 5 *System and Memory*.

**Register 34.1. APB_SARADC_CTRL_REG (0x0000)**




**APB_SARADC_START_FORCE** Configures whether to use software to enable SAR ADC.

0: Select FSM to start SAR ADC

1: Select software to start SAR ADC

(R/W)

**APB_SARADC_START** Configures whether to start SAR ADC by software.

0: No effect

1: Start SAR ADC by software

Valid only when APB_SARADC_START_FORCE = 1.

(R/W)

**APB_SARADC_SAR_CLK_GATED** Configures whether to enable SAR ADC clock gate.

0: Disable

1: Enable

(R/W)

**APB_SARADC_SAR_PATT_LEN** Configures how many patterns will be used.

0: Use only cmd0

1: Use cmd0 and cmd1

n: Use cmd0 to cmdn, the maximum n is 7

(R/W)

**APB_SARADC_SAR_PATT_P_CLEAR** Configures whether to clear the pointer of pattern table for DIG ADC controller.

0: No effect

1: Clear

(R/W)

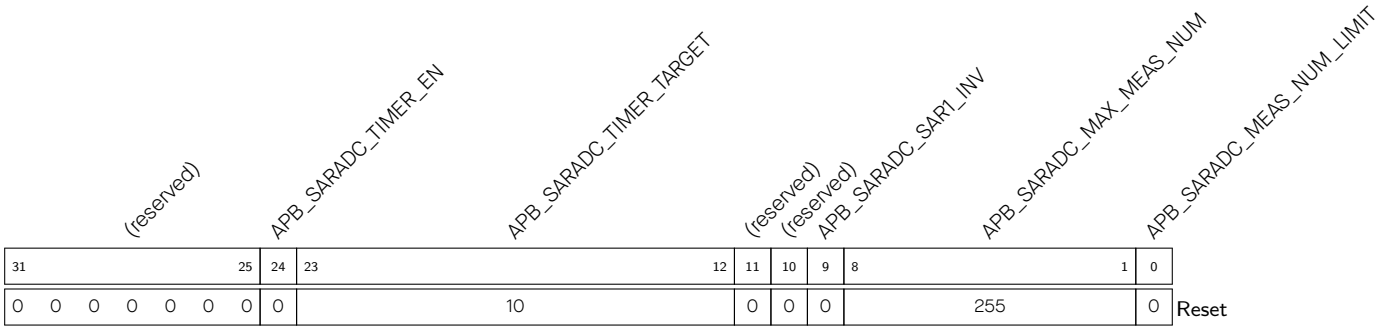
**APB_SARADC_XPD_SAR_FORCE** Configures whether to forcibly power up SAR ADC.

0: No effect

1: Forcibly power up SAR ADC

(R/W)

Register 34.2. APB_SARADC_CTRL2_REG (0x0004)




**APB_SARADC_MEAS_NUM_LIMIT** Configures whether to enable the limitation of SAR ADC's maximum conversion times.

0: Disable

1: Enable

(R/W)

**APB_SARADC_MAX_MEAS_NUM** Configures the SAR ADC's maximum conversion times. (R/W)

**APB_SARADC_SAR1_INV** Configures whether to invert the data of SAR ADC.

0: No effect

1: Invert the data of SAR ADC

(R/W)

**APB_SARADC_TIMER_TARGET** Configures SAR ADC timer target. (R/W)

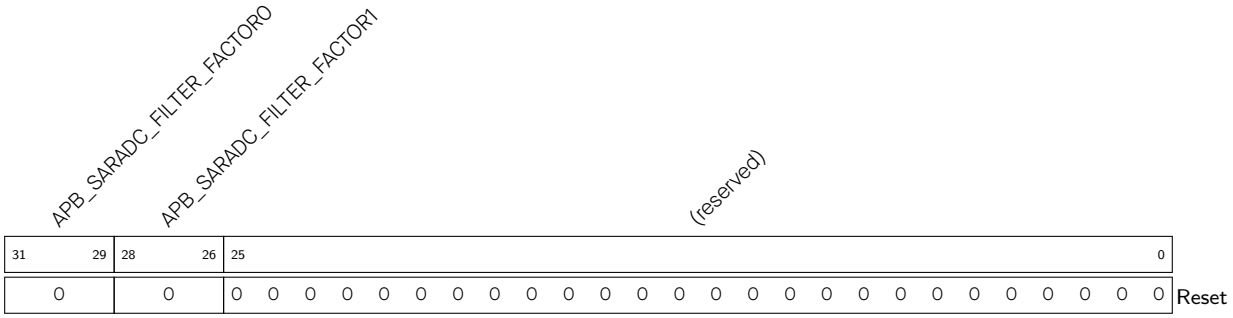
**APB_SARADC_TIMER_EN** Configures whether to enable SAR ADC timer trigger.

0: Disable

1: Enable

(R/W)

Register 34.3. APB_SARADC_FILTER_CTRL1_REG (0x0008)




**APB_SARADC_FILTER_FACTOR1** Configures the filter coefficient $k$ for SAR ADC filter 1.

0: $k$=0 (i.e., the filter is disabled)

1: $k$=2

2: $k$=4

3: $k$=8

4: $k$=16

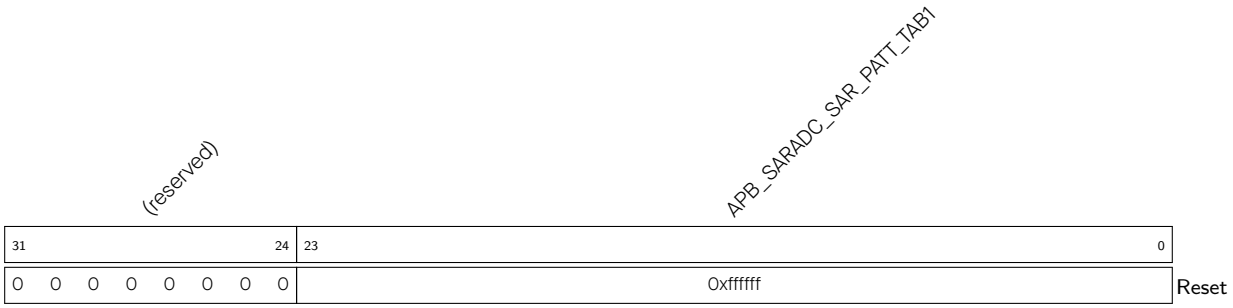
5: $k$=32

6: $k$=64

(R/W)

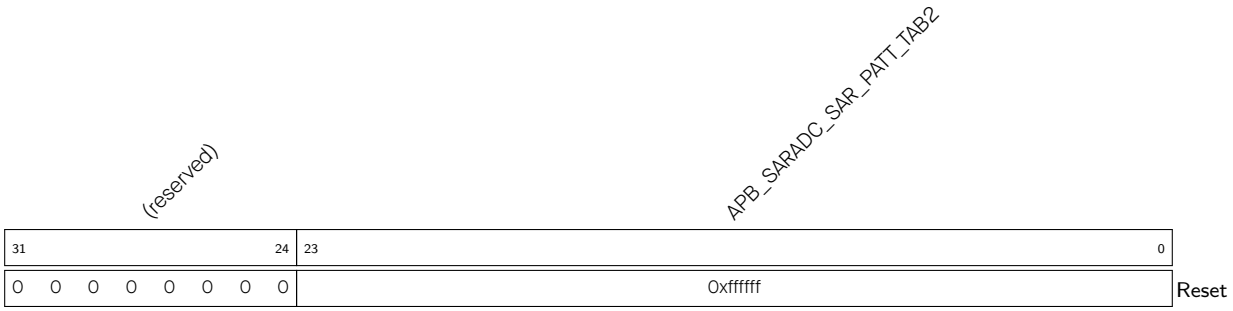
**APB_SARADC_FILTER_FACTOR0** Configures the filter coefficient $k$ for SAR ADC filter 0 (same as above). (R/W)

Register 34.4. APB_SARADC_SAR_PATT_TAB1_REG (0x0018)



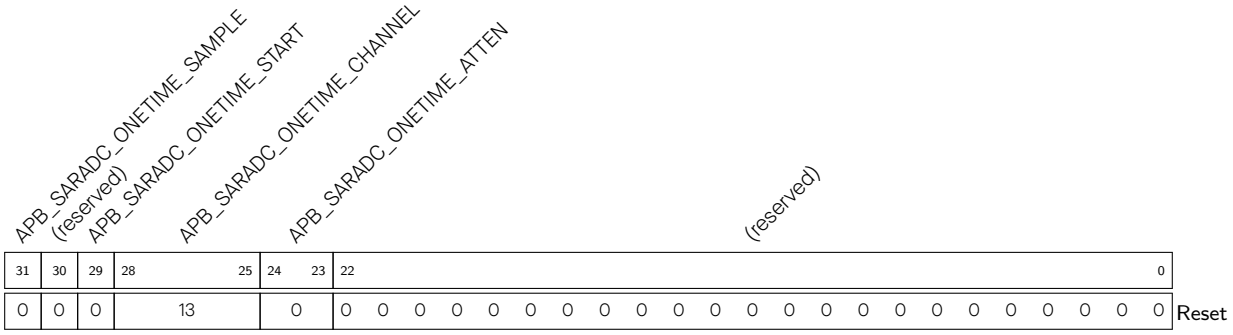

**APB_SARADC_SAR_PATT_TAB1** Configures pattern 0 ~ 3 (each pattern takes six bits). For details see Section 34.5.8 *Pattern Table*. (R/W)

Register 34.5. APB_SARADC_SAR_PATT_TAB2_REG (0x001C)




**APB_SARADC_SAR_PATT_TAB2** Configures pattern 4 ~ 7 (each pattern takes six bits). For details see Section 34.5.8 *Pattern Table*. (R/W)

Register 34.6. APB_SARADC_ONETIME_SAMPLE_REG (0x0020)




**APB_SARADC_ONETIME_ATTEN** Configures the attenuation for a one-shot sampling.

0: 0 dB

1: 2.5 dB

2: 6 dB

3: 12 dB

(R/W)

**APB_SARADC_ONETIME_CHANNEL** Configures the channel for a one-shot sampling.

0: Channel 0

1: Channel 1

2: Channel 2

3: Channel 3

(R/W)

**APB_SARADC_ONETIME_START** Configures whether to start SAR ADC one-shot sampling.

0: No effect

1: Start

(R/W)

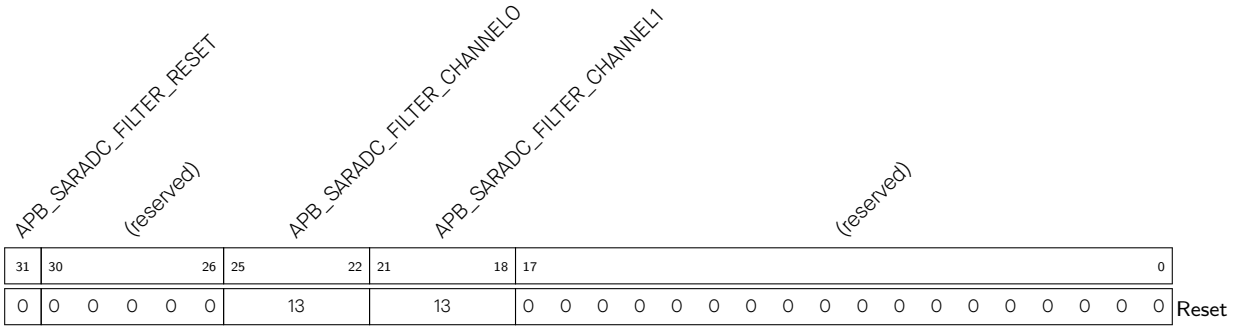
**APB_SARADC_ONETIME_SAMPLE** Configures whether to enable SAR ADC one-shot sampling.

0: Disable

1: Enable

(R/W)

Register 34.7. APB_SARADC_FILTER_CTRL0_REG (0x0028)




**APB_SARADC_FILTER_CHANNEL1** Configures the filter channel for SAR ADC filter 1. (R/W)

**APB_SARADC_FILTER_CHANNEL0** Configures the filter channel for SAR ADC filter 0. (R/W)

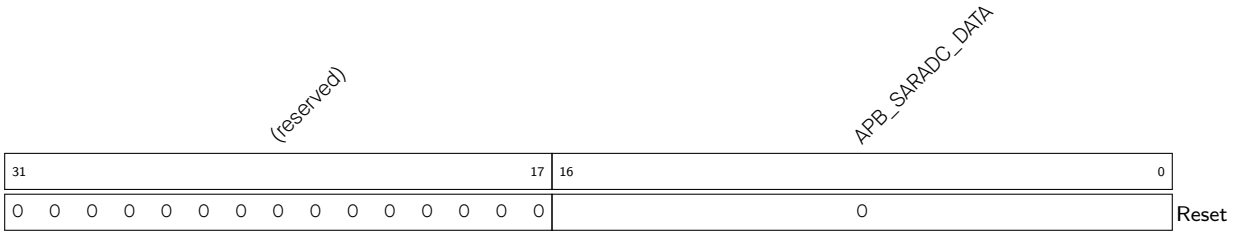
**APB_SARADC_FILTER_RESET** Configures whether to reset SAR ADC filter.

0: No effect

1: Reset

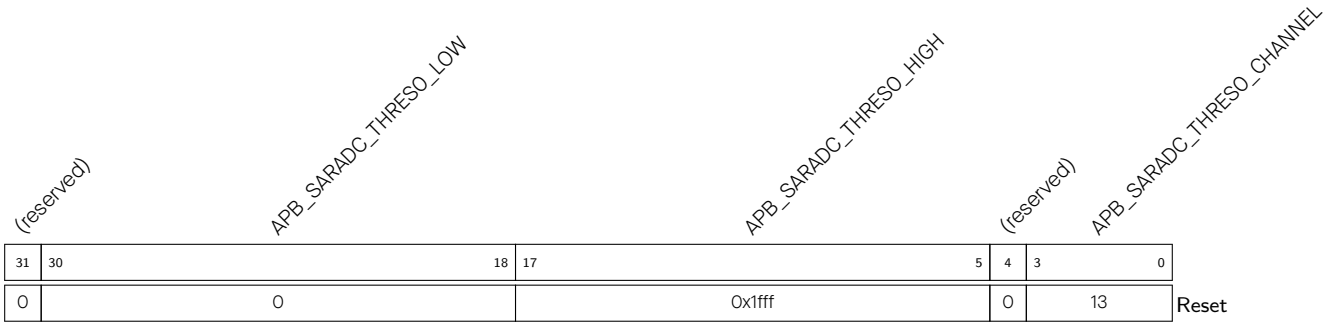
(R/W)

Register 34.8. APB_SARADC_SAR1DATA_STATUS_REG (0x002C)




**APB_SARADC_DATA** Stores SAR ADC conversion data in one-shot sampling mode. (RO)

Register 34.9. APB_SARADC_THRES0_CTRL_REG (0x0034)

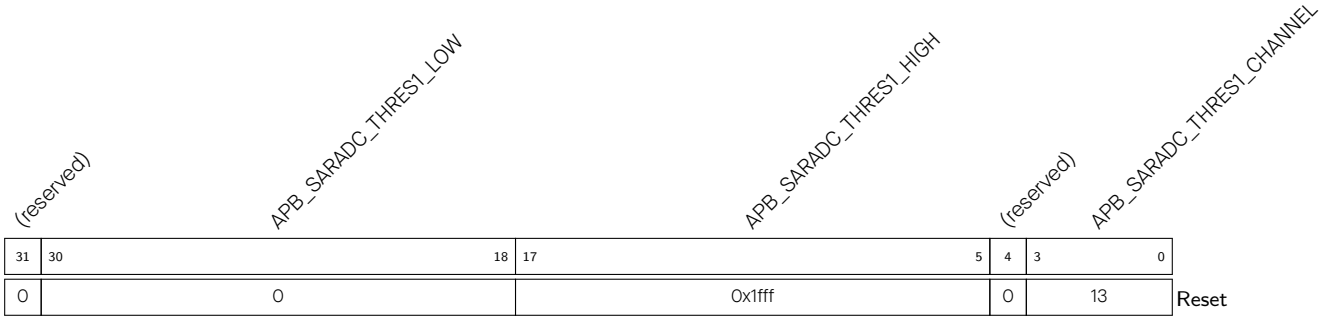



**APB_SARADC_THRES0_CHANNEL** Configures the channel for SAR ADC monitor 0. (R/W)

**APB_SARADC_THRES0_HIGH** Configures the high threshold for SAR ADC monitor 0. (R/W)

**APB_SARADC_THRES0_LOW** Configures the low threshold for SAR ADC monitor 0. (R/W)

Register 34.10. APB_SARADC_THRES1_CTRL_REG (0x0038)

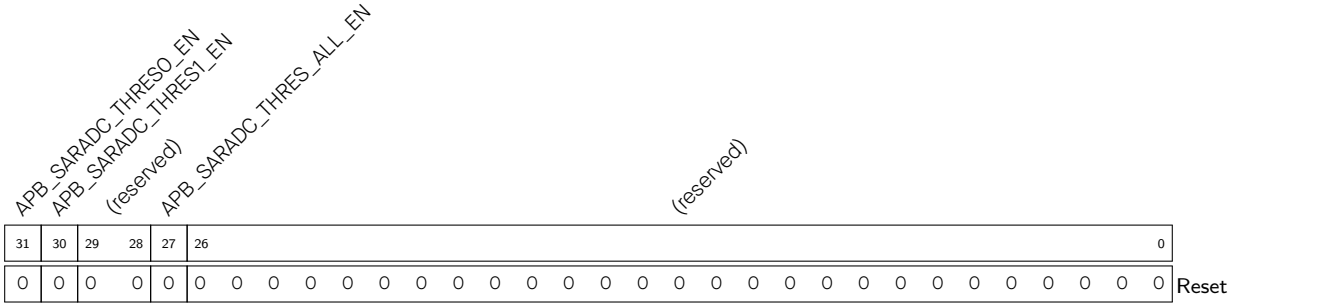



**APB_SARADC_THRES1_CHANNEL** Configures the channel for SAR ADC monitor 1. (R/W)

**APB_SARADC_THRES1_HIGH** Configures the high threshold for SAR ADC monitor 1. (R/W)

**APB_SARADC_THRES1_LOW** Configures the low threshold for SAR ADC monitor 1. (R/W)

Register 34.11. APB_SARADC_THRES_CTRL_REG (0x003C)




**APB_SARADC_THRES_ALL_EN** Configures whether to enable the threshold monitoring for all configured channels.

0: Disable

1: Enable

(R/W)

**APB_SARADC_THRES1_EN** Configures whether to enable threshold monitor 1.

0: Disable

1: Enable

(R/W)

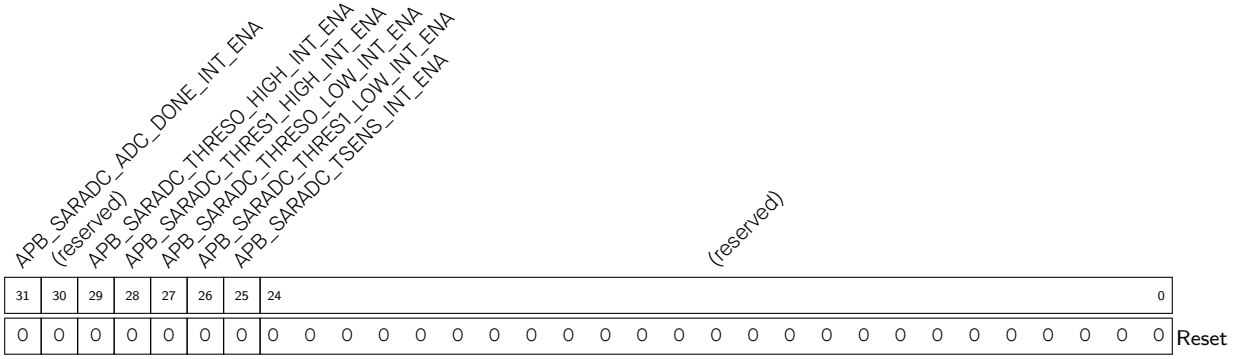
**APB_SARADC_THRES0_EN** Configures whether to enable threshold monitor 0.

0: Disable

1: Enable

(R/W)

Register 34.12. APB_SARADC_INT_ENA_REG (0x0040)




**APB_SARADC_TSENS_INT_ENA** Write 1 to enable the APB_SARADC_TSENS_INT [TBA] interrupt. (R/W)

**APB_SARADC_THRES1_LOW_INT_ENA** Write 1 to enable the APB_SARADC_THRES1_LOW_INT interrupt. (R/W)

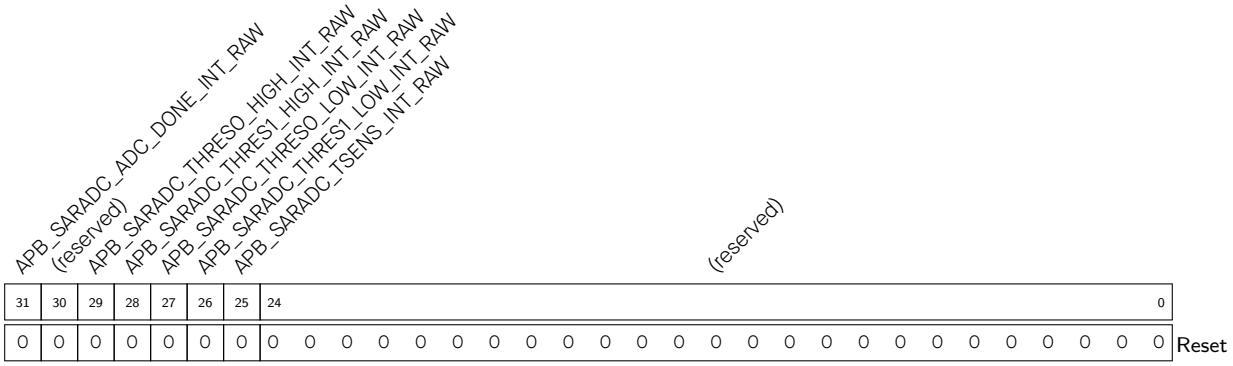
**APB_SARADC_THRES0_LOW_INT_ENA** Write 1 to enable the APB_SARADC_THRES0_LOW_INT interrupt. (R/W)

**APB_SARADC_THRES1_HIGH_INT_ENA** Write 1 to enable the APB_SARADC_THRES1_HIGH_INT interrupt. (R/W)

**APB_SARADC_THRES0_HIGH_INT_ENA** Write 1 to enable the APB_SARADC_THRES0_HIGH_INT interrupt. (R/W)

**APB_SARADC_ADC_DONE_INT_ENA** Write 1 to enable the APB_SARADC_ADC_DONE_INT interrupt. (R/W)

Register 34.13. APB_SARADC_INT_RAW_REG (0x0044)




**APB_SARADC_TSENS_INT_RAW** The raw interrupt status of the APB_SARADC_TSENS_INT interrupt. (R/WTC/SS)

**APB_SARADC_THRES1_LOW_INT_RAW** The raw interrupt status of the APB_SARADC_THRES1_LOW_INT interrupt. (R/WTC/SS)

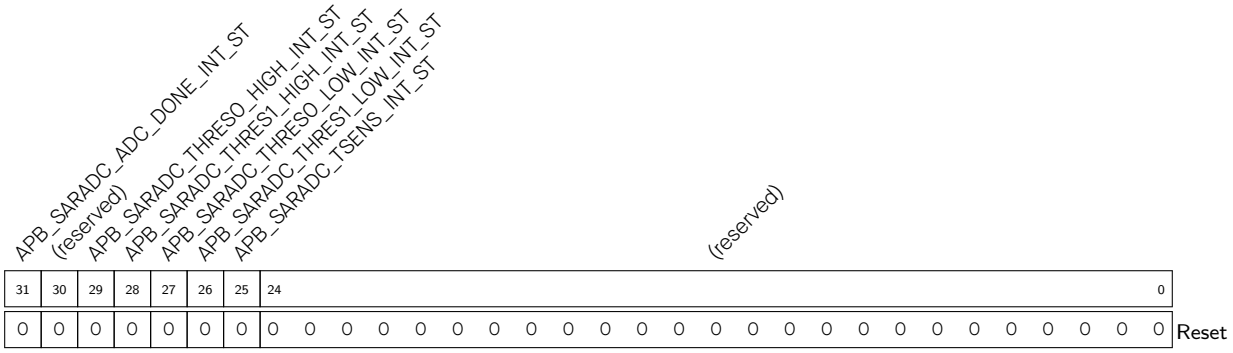
**APB_SARADC_THRESO_LOW_INT_RAW** The raw interrupt status of the APB_SARADC_THRESO_LOW_INT interrupt. (R/WTC/SS)

**APB_SARADC_THRES1_HIGH_INT_RAW** The raw interrupt status of the APB_SARADC_THRES1_HIGH_INT interrupt. (R/WTC/SS)

**APB_SARADC_THRESO_HIGH_INT_RAW** The raw interrupt status of the APB_SARADC_THRESO_HIGH_INT interrupt. (R/WTC/SS)

**APB_SARADC_ADC_DONE_INT_RAW** The raw interrupt status of the APB_SARADC_ADC_DONE_INT interrupt. (R/WTC/SS)

Register 34.14. APB_SARADC_INT_ST_REG (0x0048)




**APB_SARADC_TSENS_INT_ST** The masked interrupt status of the APB_SARADC_TSENS_INT interrupt. (RO)

**APB_SARADC_THRES1_LOW_INT_ST** The masked interrupt status of the APB_SARADC_THRES1_LOW_INT interrupt. (RO)

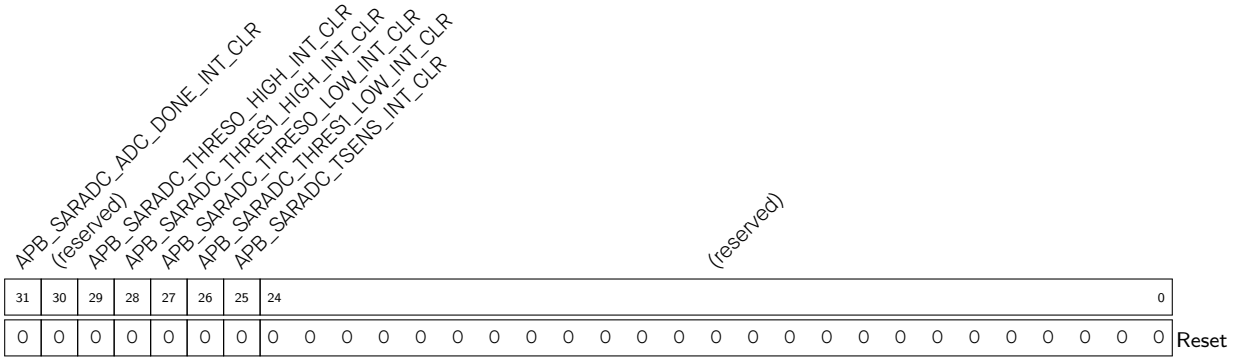
**APB_SARADC_THRES0_LOW_INT_ST** The masked interrupt status of the APB_SARADC_THRES0_LOW_INT interrupt. (RO)

**APB_SARADC_THRES1_HIGH_INT_ST** The masked interrupt status of the APB_SARADC_THRES1_HIGH_INT interrupt. (RO)

**APB_SARADC_THRES0_HIGH_INT_ST** The masked interrupt status of the APB_SARADC_THRES0_HIGH_INT interrupt. (RO)

**APB_SARADC_ADC_DONE_INT_ST** The masked interrupt status of the APB_SARADC_ADC_DONE_INT interrupt. (RO)

Register 34.15. APB_SARADC_INT_CLR_REG (0x004C)




**APB_SARADC_TSENS_INT_CLR** Write 1 to clear the APB_SARADC_TSENS_INT interrupt. (WT)

**APB_SARADC_THRES1_LOW_INT_CLR** Write 1 to clear the APB_SARADC_THRES1_LOW_INT interrupt. (WT)

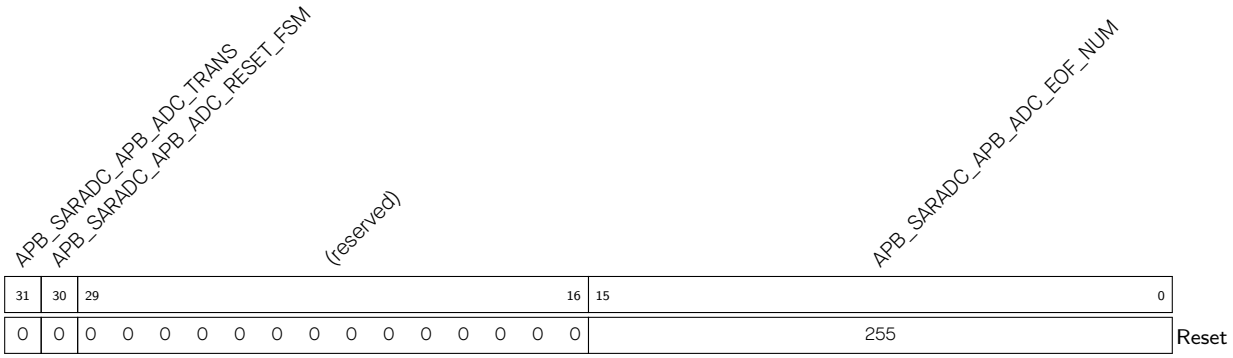
**APB_SARADC_THRES0_LOW_INT_CLR** Write 1 to clear the APB_SARADC_THRES0_LOW_INT interrupt. (WT)

**APB_SARADC_THRES1_HIGH_INT_CLR** Write 1 to clear the APB_SARADC_THRES1_HIGH_INT interrupt. (WT)

**APB_SARADC_THRES0_HIGH_INT_CLR** Write 1 to clear the APB_SARADC_THRES0_HIGH_INT interrupt. (WT)

**APB_SARADC_ADC_DONE_INT_CLR** Write 1 to clear the APB_SARADC_ADC_DONE_INT interrupt. (WT)

Register 34.16. APB_SARADC_DMA_CONF_REG (0x0050)




**APB_SARADC_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)

**APB_SARADC_APB_ADC_RESET_FSM** Configures whether to reset DIG ADC controller status.
0: No effect
1: Reset
(R/W)

**APB_SARADC_APB_ADC_TRANS** Configures whether to let DIG ADC controller use GDMA.
0: No effect
1: DIG ADC controller uses DMA
(R/W)

# Chapter 35

# Analog Voltage Comparator

## 35.1 Introduction

ESP32-C61 integrates an analog voltage comparator. The comparators rely on special pads that support voltage comparison functionality to monitor voltage changes on these pads. The 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.

## 35.2 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

## 35.3 Architectural Overview

Figure 35.3-1 shows the structure of the analog voltage comparator.

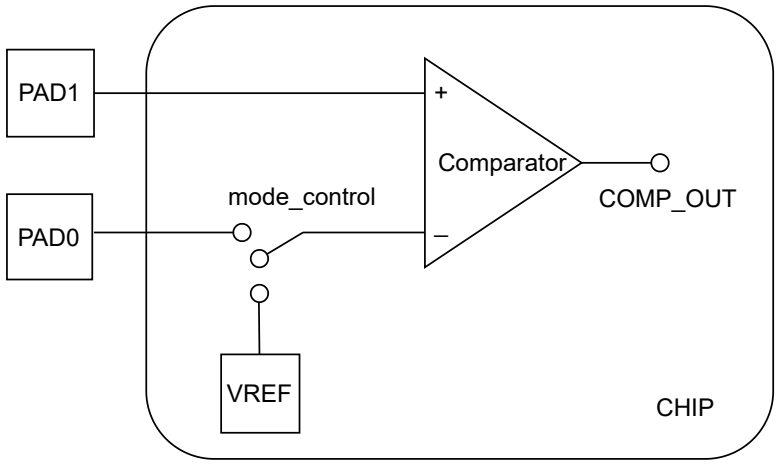



Figure 35.3-1. Analog Voltage Comparator Architecture

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.

## 35.4 Functional Description

The analog voltage comparator has the following functions:

- **Voltage comparison**

  The analog voltage comparator relies on specialized pads that support voltage comparison. Table below shows the mapping between the PAD of analog voltage comparator and GPIO pins.

  **Table 35.4-1. Mapping Between PAD and GPIO**

| Analog Voltage Comparator | PAD0 | PAD1 |
|---|---|---|
| Analog voltage comparator | GPIO8 | GPIO9 |

  The comparison mode and reference voltage are configurable. For detailed configuration procedures, see Section 35.7 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 GPIO_EXT_PAD_COMP_CONFIG_0_REG.

- **Reference voltage configuration**

  The internal reference voltage range is 0 ~ (0.7 × VDDPST1) V with a step of 0.1 × VDDPST1 V, where VDDPST1 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 GPIO_EXT_DREF_COMP_0.

The external reference voltage is accessed directly from the pad with an input range of 0 ~ (0.7 × VDDPST1) 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.

## 35.5 Event Task Matrix Feature

The analog voltage comparator on ESP32-C61 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 11 Event Task Matrix (ETM).

The analog voltage comparator can generate the following ETM events:

- GPIO_EVT_ZERO_DET_POS: 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: 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.

## 35.6 Interrupts

ESP32-C61'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 35.6-1.

**Table 35.6-1. Analog Voltage Comparator's Internal Interrupt Sources**

| Internal Interrupt Source | Trigger Condition | Interrupt Signal |
|---|---|---|
| GPIO_COMP_ALL_INT | Any COMP_OUT signal transition occurs | GPIO_PAD_COMP_INT |
| GPIO_COMP_NEG_INT | The COMP_OUT signal changes from high level to low level | GPIO_PAD_COMP_INT |
| GPIO_COMP_POS_INT | 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 10 *Interrupt Matrix* > Section 10.2 *Interrupt Terminology in ESP32-C61*.

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 7.18.1 *HP GPIO Matrix Register Summary*.

## 35.7 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 7 *GPIO Matrix and IO MUX*.

1. Set PCR_IOMUX_FUNC_CLK_EN to 1.
2. Enable the comparator by setting GPIO_EXT_XPD_COMP_0 to 1.
3. 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 7 GPIO Matrix and IO MUX.
4. Configure the comparison mode by setting GPIO_EXT_PAD_COMP_CONFIG_0_REG:
   - 0: Configures to compare the main voltage with the internal reference voltage;
   - 1: Configures to compare the main voltage with the external reference voltage.
5. Configure the internal reference voltage through GPIO_EXT_DREF_COMP_0, which offers a voltage range of 0 ~ (0.7 × VDDPST1) V with a step of 0.1 × VDDPST1 V.

# Part VII

# 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-C61 Series Datasheet*- Specifications of the ESP32-C61 hardware.
- *ESP32-C61 Hardware Design Guidelines* – Guidelines on how to integrate the ESP32-C61 into your hardware product.
- *Certificates*
  https://espressif.com/en/support/documents/certificates
- *ESP32-C61 Product/Process Change Notifications (PCN)*
  https://espressif.com/en/support/documents/pcns?keys=ESP32-C61
- *ESP32-C61 Advisories* – Information on security, bugs, compatibility, component reliability.
  https://espressif.com/en/support/documents/advisories?keys=ESP32-C61
- *Documentation Updates and Update Notification Subscription*
  https://espressif.com/en/support/download/documents

## Developer Zone

- ESP-IDF Programming Guide for ESP32-C61 – 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-C61 Series SoCs* – Browse through all ESP32-C61 SoCs.
  https://espressif.com/en/products/socs?id=ESP32-C61
- *ESP32-C61 Series Modules* – Browse through all ESP32-C61-based modules.
  https://espressif.com/en/products/modules?id=ESP32-C61
- *ESP32-C61 Series DevKits* – Browse through all ESP32-C61-based devkits.
  https://espressif.com/en/products/devkits?id=ESP32-C61
- *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

| | |
|---|---|
| ADC | ADC Controller |
| ECC | ECC (Elliptic Curve Cryptography) Accelerator |
| ECDSA_DS | ECDSA (Elliptic Curve Digital Signature Algorithm) Digital Signature Peripheral |
| eFuse | eFuse Controller |
| ETM | Event Task Matrix |
| GDMA | GDMA (General Direct Memory Access) Controller |
| I2C | I2C (Inter-Integrated Circuit) Controller |
| I2S | I2S (Inter-IC Sound) Controller |
| PMS | Permission Control |
| LEDC | LED Control PWM (Pulse Width Modulation) |
| RNG | Random Number Generator |
| SDIO | SDIO Slave Controller |
| SHA | SHA (Secure Hash Algorithm) Accelerator |
| SPI | SPI (Serial Peripheral Interface) Controller |
| TIMG | Timer Group |
| TRACE | RISC-V Trace Encoder |
| TSENS | Temperature Sensor |
| UART | UART (Universal Asynchronous Receiver-Transmitter) 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

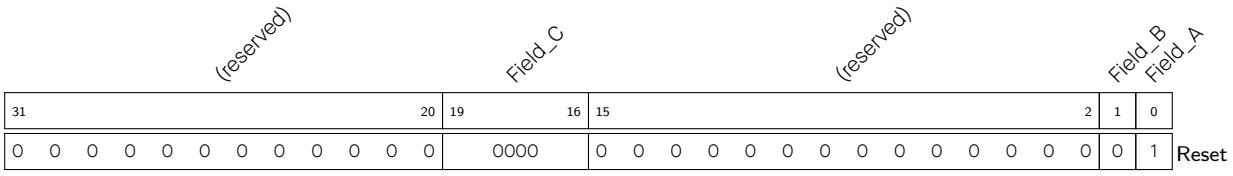
The reserved fields should not be modified. It is not possible to write only part of a register since registers must always be written as a whole. As a result, to write an entire register that contains reserved fields, you can choose one of the following two options:

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 35.1. 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 35.7-4.
- **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 35.7-4. 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 |
|---|---|---|
| 2026-04-10 | v1.0 | Added the following chapters:<br>• Chapter 1 *Bus Architecture*<br>Updated the following chapters:<br>• Chapter 2 *ESP-RISC-V CPU*: Updated the pmpXcfg register field<br>• Chapter 23 *ECDSA Digital Signature Peripheral (ECDSA_DS)*: Renamed the module from Elliptic Curve Digital Signature Algorithm (ECDSA) to ECDSA Digital Signature Peripheral (ECDSA_DS) |
| 2026-01-05 | v0.5 | Added the following chapters:<br>• Chapter 2 *ESP-RISC-V CPU*<br>• Chapter 7 *GPIO Matrix and IO MUX*<br>• Chapter 18 *System Registers*<br>Updated the following chapters:<br>• Chapter 4 *GDMA Controller (GDMA)*: Removed information about PARLIO and UHCI<br>• Chapter 8 *Reset and Clock*: Removed information about UHCI, AES, RSA, HMAC and DS<br>• Chapter 26 *UART Controller (UART)*: Removed information about UHCI<br>• Chapter 28 *I2C Controller (I2C)*: Updated the description of two available start methods for the I2C controller in slave mode |
| 2025-08-15 | v0.1 | 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