ESP32-H2

Technical Reference Manual Version 1.2

ESPRESSIF

www.espressif.com

About This Document

The **ESP32-H2 Technical Reference Manual** is targeted at developers working on low level software projects that use the ESP32-H2 SoC. It describes the hardware modules listed below for the ESP32-H2 SoC and other products in ESP32-H2 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	ESP-RISC-V CPU	Published
2	RISC-V Trace Encoder (TRACE)	Published
3	GDMA Controller (GDMA)	Published
	Part II. Memory Organization	
4	System and Memory	Published
5	eFuse Controller (EFUSE)	Published
	Part III. System Component	
6	IO MUX and GPIO Matrix (GPIO, IO MUX)	Published
7	Reset and Clock	Published
8	Chip Boot Control	Published
9	Interrupt Matrix (INTMTX)	Published
10	Event Task Matrix (SOC_ETM)	Published
11	Low-Power Management	Published
12	System Timer (SYSTIMER)	Published
13	Timer Group (TIMG)	Published
14	Watchdog Timers (WDT)	Published
15	Permission Control (PMS)	Published
16	System Registers	Published
17	Debug Assistant (ASSIST_DEBUG, MEM_MONITOR)	Published
18	Power Supply Detector	Published
	Part IV. Cryptography/Security Component	
19	AES Accelerator (AES)	Published
20	ECC Accelerator (ECC)	Published
21	HMAC Accelerator (HMAC)	Published
22	RSA Accelerator (RSA)	Published
23	SHA Accelerator (SHA)	Published
24	RSA Digital Signature Peripheral (RSA_DS)	Published
25	ECDSA Digital Signature Peripheral (ECDSA_DS)	Published
26	External Memory Encryption and Decryption (XTS_AES)	Published
27	Random Number Generator (RNG)	Published
	Part V. Connectivity Interface	
28	UART Controller (UART)	Published
29	SPI Controller (SPI)	Published
30	I2C Controller (I2C)	Published
31	I2S Controller (I2S)	Published
32	Pulse Count Controller (PCNT)	Published
33	USB Serial/JTAG Controller (USB_SERIAL_JTAG)	Published
34	Two-wire Automotive Interface (TWAI)	Published
35	LED PWM Controller (LEDC)	Published

Cont'd on next page

Cont'd from previous page

No.	Chapter	Progress
36	Motor Control PWM (MCPWM)	Published
37	Remote Control Peripheral (RMT)	Published
38	Parallel IO Controller (PARLIO)	Published
	Part VI. Analog Signal Processing	
38	Temperature Sensor (TSENS)	Published
39	ADC Controller (ADC)	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-h2_technical_reference_manual_en.pdf

Contents

I Microprocessor and Master 37

1 ESP-RISC-V CPU 38
1.1 Overview 38
1.2 Features 38
1.3 Terminology 39
1.4 Address Map 39
1.5 Configuration and Status Registers (CSRs) 40
1.5.1 Register Summary 40
1.5.2 Register Description 42
1.6 Interrupt Controller 54
1.6.1 Features 54
1.6.2 Functional Description 54
1.6.3 Suggested Operation 56
1.6.3.1 Latency Aspects 56
1.6.3.2 Configuration Procedure 57
1.6.4 Registers 58
1.7 Core Local Interrupts (CLINT) 59
1.7.1 Overview 59
1.7.2 Features 59
1.7.3 Software Interrupt 59
1.7.4 Timer Counter and Interrupt 60
1.7.5 Register Summary 60
1.7.6 Register Description 60
1.8 Physical Memory Protection 64
1.8.1 Overview 64
1.8.2 Features 64
1.8.3 Functional Description 64
1.8.4 Register Summary 65
1.8.5 Register Description 65
1.9 Physical Memory Attribute Checker (PMAC) 66
1.9.1 Overview 66
1.9.2 Features 66
1.9.3 Functional Description 66
1.9.4 Register Summary 67
1.9.5 Register Description 68
1.10 Debug 69
1.10.1 Overview 69
1.10.2 Features 70
1.10.3 Functional Description 70
1.10.4 JTAG Control 70
1.10.5 Register Summary 71

1.10.6 Register Description 71
1.11 Hardware Trigger 74
1.11.1 Features 74
1.11.2 Functional Description 74
1.11.3 Trigger Execution Flow 75
1.11.4 Register Summary 75
1.11.5 Register Description 76
1.12 Trace 80
1.12.1 Overview 80
1.12.2 Features 80
1.12.3 Functional Description 80
1.13 Dedicated IO 81
1.13.1 Overview 81
1.13.2 Features 81
1.13.3 Functional Description 81
1.13.4 Register Summary 82
1.13.5 Register Description 82
1.14 Atomic (A) Extension 84
1.14.1 Overview 84
1.14.2 Functional Description 84
1.14.2.1 Load Reserve (LR.W) Instruction 84
1.14.2.2 Store Conditional (SC.W) Instruction 84
1.14.2.3 AMO Instructions 85

2 RISC-V Trace Encoder (TRACE) 86
2.1 Terminology 86
2.2 Introduction 87
2.3 Features 87
2.4 Architectural Overview 89
2.5 Functional Description 89
2.5.1 Synchronization 89
2.5.2 Anchor Tag 89
2.5.3 Memory Writing Mode 89
2.5.4 Automatic Restart 90
2.6 Encoder Output Packets 90
2.6.1 Header 90
2.6.2 Index 91
2.6.3 Payload 91
2.6.3.1 Format 3 Packets 91
2.6.3.2 Format 2 Packets 92
2.6.3.3 Format 1 Packets 93
2.7 Interrupt 94
2.8 Programming Procedures 94
2.8.1 Enable Encoder 94
2.8.2 Disable Encoder 95
2.8.3 Decode Data Packets 95

2.9 Register Summary 97
2.10 Registers 98

3 GDMA Controller (GDMA) 103
3.1 Overview 103
3.2 Features 103
3.3 Architecture 104
3.4 Functional Description 105
3.4.1 Linked List 105
3.4.2 Peripheral-to-Memory and Memory-to-Peripheral Data Transfer 106
3.4.3 Memory-to-Memory Data Transfer 107
3.4.4 Enabling GDMA 107
3.4.5 Linked List Reading Process 108
3.4.6 EOF 108
3.4.7 Accessing Internal RAM 109
3.4.8 Arbitration 109
3.4.9 Event Task Matrix Feature 109
3.5 GDMA Interrupts 110
3.6 Programming Procedures 111
3.6.1 Programming Procedures for GDMA's Transmit Channel 112
3.6.2 Programming Procedures for GDMA's Receive Channel 112
3.6.3 Programming Procedures for Memory-to-Memory Transfer 112
3.7 Register Summary 114
3.8 Registers 118

II Memory Organization 141

4 System and Memory 142
4.1 Overview 142
4.2 Features 142
4.3 Functional Description 143
4.3.1 Address Mapping 143
4.3.2 Internal Memory 144
4.3.3 External Memory 145
4.3.3.1 External Memory Address Mapping 145
4.3.3.2 Cache 145
4.3.3.3 Cache Operations 145
4.3.4 GDMA Address Space 146
4.3.5 Modules/Peripherals Address Mapping 147

5 eFuse Controller (EFUSE) 150
5.1 Overview 150
5.2 Features 150
5.3 Functional Description 150
5.3.1 Structure 150
5.3.1.1 Parameters Used by Hardware Modules 157

5.3.1.2 EFUSE_WR_DIS 157
5.3.1.3 EFUSE_RD_DIS 157
5.3.1.4 Data Storage 158
5.3.2 Programming of Parameters 159
5.3.3 Reading of Parameters by Users 161
5.3.4 eFuse VDDQ Timing 162
5.3.5 Parameters Used by Hardware Modules 162
5.3.6 Interrupts 163
5.4 Register Summary 164
5.5 Registers 168

III System Component 217

6 IO MUX and GPIO Matrix (GPIO, IO MUX) 218
6.1 Overview 218
6.2 Features 218
6.3 Architectural Overview 218
6.4 Peripheral Input via GPIO Matrix 220
6.4.1 Overview 220
6.4.2 Signal Synchronization 221
6.4.3 Functional Description 221
6.4.4 Simple GPIO Input 223
6.5 Peripheral Output via GPIO Matrix 223
6.5.1 Overview 223
6.5.2 Functional Description 224
6.5.3 Simple GPIO Output 225
6.5.4 Sigma Delta Modulated Output (SDM) 225
6.5.4.1 Functional Description 225
6.5.4.2 SDM Configuration 226
6.6 Direct Input and Output via IO MUX 226
6.6.1 Overview 226
6.6.2 Functional Description 226
6.7 Analog Functions of GPIO Pins 226
6.8 Pin Functions in Light-sleep 227
6.9 Pin Hold Feature 227
6.10 Hysteresis Characteristics of GPIO Pins 228
6.11 Power Supplies and Management of GPIO Pins 229
6.11.1 Power Supplies of GPIO Pins 229
6.11.2 Power Supply Management 229
6.12 Peripheral Signal List 229
6.13 IO MUX Functions List 236
6.14 IO MUX Pins Analog Functions List 237
6.15 Function of Analog PAD Voltage Comparator 238
6.16 Event Task Matrix Function 238
6.17 Register Summary 240
6.17.1 GPIO Matrix Register Summary 240

6.17.2 IO MUX Register Summary 241
6.17.3 GPIO_EXT Register Summary 242
6.18 Registers 244
6.18.1 GPIO Matrix Registers 244
6.18.2 IO MUX Registers 254
6.18.3 GPIO_EXT Registers 257

7 Reset and Clock 271
7.1 Reset 271
7.1.1 Overview 271
7.1.2 Architectural Overview 271
7.1.3 Features 272
7.1.4 Functional Description 272
7.1.5 Peripheral Reset 273
7.2 Clock 274
7.2.1 Overview 274
7.2.2 Architectural Overview 274
7.2.3 Features 274
7.2.4 Functional Description 275
7.2.4.1 HP System Clock 275
7.2.4.2 LP System Clock 276
7.2.4.3 Peripheral Clocks 277
7.3 Programming Procedures 280
7.3.1 HP System Clock Configuration 280
7.3.2 LP System Clock Configuration 280
7.3.3 Peripheral Clock Reset and Configuration 280
7.4 Register Summary 282
7.4.1 PCR Register Summary 282
7.4.2 LP System Clock Register Summary 284
7.5 Registers 286
7.5.1 PCR Registers 286
7.5.2 LP System Clock Registers 335

8 Chip Boot Control 346
8.1 Overview 346
8.2 Functional Description 346
8.2.1 Default Configuration 346
8.2.2 Boot Mode Control 347
8.2.3 ROM Messages Printing Control 349
8.2.4 JTAG Signal Source Control 349

9 Interrupt Matrix (INTMTX) 351
9.1 Overview 351
9.2 Interrupt Terminology in ESP32-H2 351
9.2.1 Interrupt 351
9.2.2 Interrupt Signal/Interrupt Source 351

9.2.3 Interrupt Flow in ESP32-H2 352
9.3 Features 352
9.4 Architecture 352
9.5 Functional Description 353
9.5.1 Peripheral Interrupt Sources 353
9.5.2 CPU Interrupts 356
9.5.3 Assign Peripheral Interrupt Source to CPU Interrupt 356
9.5.3.1 Assign One SOURCE to CPU Interrupt 356
9.5.3.2 Assign Multiple SOURCEs to CPU Interrupt 356
9.5.3.3 Unassign SOURCE 356
9.5.4 Query Current Interrupt Status of SOURCE 357
9.6 Register Summary 358
9.6.1 Interrupt Matrix Register Summary 358
9.6.2 Interrupt Priority Register Summary 360
9.7 Registers 363
9.7.1 Interrupt Matrix Registers 363
9.7.2 Interrupt Priority Registers 366

10 Event Task Matrix (SOC_ETM) 370
10.1 Overview 370
10.2 Features 370
10.3 Functional Description 370
10.3.1 Architecture 371
10.3.2 Events 372
10.3.3 Tasks 375
10.3.4 Timing Considerations 378
10.3.5 Channel Control 379
10.4 Register Summary 381
10.5 Registers 384

11 Low-Power Management 388
11.1 Overview 388
11.2 Terminology 388
11.3 Features 388
11.4 Functional Description 389
11.4.1 Power Scheme 389
11.4.1.1 Regulators 390
11.4.1.2 Digital Power Domains 390
11.4.1.3 Analog Power Domains 391
11.4.2 PMU 391
11.4.2.1 PMU Main State Machine 392
11.4.2.2 Sleep/Wake-up Controller 394
11.4.2.3 Analog Power Controller 394
11.4.2.4 Digital Power Controller 395
11.4.2.5 Clock Controller 396
11.4.2.6 Backup Controller 399

11.4.2.7 System Controller 399
11.4.3 RTC Timer 400
11.5 Power Modes 401
11.6 RTC Boot 401
11.7 Event Task Matrix Feature 402
11.8 Interrupts 403
11.9 Register Summary 404
11.9.1 PMU Register Summary 404
11.9.2 Always-on Register Summary 405
11.9.3 RTC Timer Register Summary 406
11.10 Registers 407
11.10.1 PMU Registers 407
11.10.2 Always-on Registers 433
11.10.3 RTC Timer Registers 433

12 System Timer (SYSTIMER) 440
12.1 Overview 440
12.2 Features 440
12.3 Clock Source Selection 441
12.4 Functional Description 441
12.4.1 Counter 442
12.4.2 Comparator and Alarm 442
12.4.3 Event Task Matrix 443
12.4.4 Synchronization Operation 444
12.4.5 Interrupt 444
12.5 Programming Procedure 444
12.5.1 Read Current Count Value 444
12.5.2 Configure a One-Time Alarm in Target Mode 445
12.5.3 Configure Periodic Alarms in Period Mode 445
12.5.4 Update After Light-sleep 445
12.6 Register Summary 447
12.7 Registers 449

13 Timer Group (TIMG) 464
13.1 Overview 464
13.2 Features 464
13.3 Functional Description 465
13.3.1 16-bit Prescaler and Clock Selection 465
13.3.2 54-bit Time-base Counter 466
13.3.3 Alarm Generation 466
13.3.4 Timer Reload 467
13.3.5 Event Task Matrix Feature 467
13.3.6 RTC_SLOW_CLK Frequency Calculation 468
13.3.7 Interrupts 469
13.4 Configuration and Usage 469
13.4.1 Timer as a Simple Clock 469

13.4.2 Timer as One-shot Alarm 470
13.4.3 Timer as Periodic Alarm by APB 470
13.4.4 Timer as Periodic Alarm by ETM 471
13.4.5 RTC_SLOW_CLK Frequency Calculation 471
13.5 Register Summary 473
13.6 Registers 475

14 Watchdog Timers (WDT) 489
14.1 Overview 489
14.2 Digital Watchdog Timers 490
14.2.1 Features 490
14.2.2 Functional Description 491
14.2.2.1 Clock Source and 32-Bit Counter 491
14.2.2.2 Stages and Timeout Actions 492
14.2.2.3 Write Protection 493
14.2.2.4 Flash Boot Protection 493
14.3 Super Watchdog 493
14.3.1 Features 493
14.3.2 Super Watchdog Controller 494
14.3.2.1 Structure 494
14.3.2.2 Workflow 494
14.4 Interrupts 494
14.5 Register Summary 495
14.6 Registers 496

15 Permission Control (PMS) 505
15.1 Overview 505
15.2 Features 506
15.3 TEE and REE Terminology 506
15.4 Functional Description 507
15.4.1 TEE Controller Functional Description 507
15.4.2 APM Controller Functional Description 508
15.4.2.1 Architecture 508
15.4.2.2 Address Ranges 509
15.4.2.3 Access Permissions of Address Ranges 509
15.5 Programming Procedure 510
15.6 Illegal Access and Interrupts 510
15.7 Register Summary 511
15.7.1 APM Registers of HP System (HP_APM_REG) 511
15.7.2 APM Registers of LP System (LP_APM_REG) 513
15.7.3 TEE Registers of HP System 513
15.8 Registers 515
15.8.1 APM Registers of HP System (HP_APM_REG) 515
15.8.2 APM Registers of LP System (LP_APM_REG) 526
15.8.3 TEE Registers of HP System 530

16 System Registers 532
16.1 Overview 532
16.2 Function Description 532
16.2.1 External Memory Encryption/Decryption Configuration 532
16.2.2 Anti-DPA Attack Security Control 532
16.2.3 Bus Timeout Protection 533
16.2.3.1 CPU Peripheral Timeout Protection Register 533
16.2.3.2 HP Peripheral Timeout Protection Register 533
16.2.3.3 LP Peripheral Timeout Protection Register 533
16.3 Register Summary 535
16.4 Registers 536

17 Debug Assistant (ASSIST_DEBUG, MEM_MONITOR) 543
17.1 Overview 543
17.2 Features 543
17.3 Functional Description 543
17.3.1 Region Read/Write Monitoring 543
17.3.2 SP Monitoring 543
17.3.3 PC Logging 544
17.3.4 CPU/DMA Bus Access Logging 544
17.4 Recommended Operation 544
17.4.1 Region Read/Write Monitoring and SP Monitoring Configuration 544
17.4.2 PC Logging Configuration 546
17.4.3 CPU/DMA Bus Access Logging Configuration 546
17.5 Register Summary 548
17.5.1 Summary of Bus Logging Configuration Registers 549
17.5.2 Summary of Other Registers 549
17.6 Registers 551
17.6.1 Bus Logging Configuration Registers 552
17.6.2 Other Registers 558

18 Power Supply Detector 573
18.1 Overview 573
18.2 Features 573
18.3 Functional Description 573
18.3.1 Architecture 573
18.3.2 Brown-out Detector 574
18.3.3 Voltage Glitch Detectors 575
18.4 Interrupts 576
18.5 Register Summary 578
18.6 Registers 579

IV Cryptography/Security Component 584

19 AES Accelerator (AES) 585
19.1 Introduction 585

19.2 Features 585
19.3 Clock and Reset 585
19.4 AES Working Modes 586
19.5 Typical AES Working Mode 587
19.5.1 Key, Plaintext, and Ciphertext 587
19.5.2 Endianness 587
19.5.3 Operation Process 589
19.6 DMA-AES Working Mode 589
19.6.1 Key, Plaintext, and Ciphertext 590
19.6.2 Endianness 591
19.6.3 Standard Incrementing Function 591
19.6.4 Block Number 591
19.6.5 Initialization Vector 591
19.6.6 Block Operation Process 592
19.7 Anti-Attack Pseudo-Round Function 592
19.8 Memory Summary 593
19.9 Register Summary 594
19.10 Registers 595

20 ECC Accelerator (ECC) 601
20.1 Introduction 601
20.2 Features 601
20.3 ECC Basics 601
20.3.1 Elliptic Curve and Points on the Curves 601
20.3.2 Affine Coordinates and Jacobian Coordinates 601
20.3.3 Memory Blocks 602
20.3.4 Data and Data Block 602
20.3.5 Writing Data 602
20.3.6 Reading Data 603
20.3.7 Standard Calculation and Jacobian Calculation 603
20.4 Function Description 603
20.4.1 Key Size 603
20.4.2 Working Modes 604
20.4.2.1 Affine Point Multiplication (Affine Point Multi) 604
20.4.2.2 Affine Point Verification (Affine Point Verif) 604
20.4.2.3 Affine Point Verification + Affine Point Multiplication (Affine Point Verif + Multi) 605
20.4.2.4 Jacobian Point Multiplication (Jacobian Point Multi) 605
20.4.2.5 Point Addition (Point Add) 605
20.4.2.6 Jacobian Point Verification (Jacobian Point Verif) 606
20.4.2.7 Affine Point Verification + Jacobian Point Multiplication (Affine Point Verif + Jacobian Point Multi) 606
20.4.2.8 Mod Addition (Mod Add) 606
20.4.2.9 Mod Subtraction (Mod Sub) 607
20.4.2.10 Mod Multiplication (Mod Multi) 607
20.4.2.11 Mod Division (Mod Div) 607
20.4.3 Enhancing Anti-Attack Performance 608

20.5 Clock and Reset 608
20.6 Interrupts 608
20.7 Programming Procedures 609
20.8 Register Summary 610
20.9 Registers 611

21 HMAC Accelerator (HMAC) 615
21.1 Main Features 615
21.2 Functional Description 615
21.2.1 Upstream Mode 615
21.2.2 Downstream JTAG Enable Mode 616
21.2.3 Downstream RSA_DS Mode 616
21.2.4 HMAC eFuse Configuration 617
21.2.5 HMAC Process (Detailed) 618
21.3 HMAC Algorithm Details 620
21.3.1 Padding Bits 620
21.3.2 HMAC Algorithm Structure 620
21.4 Register Summary 622
21.5 Registers 624

22 RSA Accelerator (RSA) 631
22.1 Introduction 631
22.2 Features 631
22.3 Functional Description 631
22.3.1 Large-Number Modular Exponentiation 632
22.3.2 Large-Number Modular Multiplication 633
22.3.3 Large-Number Multiplication 634
22.3.4 Options for Additional Acceleration 634
22.4 Memory Summary 637
22.5 Register Summary 637
22.6 Registers 638

23 SHA Accelerator (SHA) 642
23.1 Introduction 642
23.2 Features 642
23.3 Working Modes 642
23.4 Function Description 643
23.4.1 Preprocessing 643
23.4.1.1 Padding the Message 643
23.4.1.2 Parsing the Message 643
23.4.1.3 Setting the Initial Hash Value 644
23.4.2 Hash Operation 644
23.4.2.1 Typical SHA Mode Process 644
23.4.2.2 DMA-SHA Mode Process 646
23.4.3 Message Digest 646
23.4.4 Interrupt 647

23.5 Register Summary 648
23.6 Registers 649

24 RSA Digital Signature Peripheral (RSA_DS) 653
24.1 Overview 653
24.2 Features 653
24.3 Functional Description 653
24.3.1 Overview 653
24.3.2 Private Key Operands 654
24.3.3 Software Prerequisites 654
24.3.4 RSA_DS Operation at the Hardware Level 656
24.3.5 RSA_DS Operation at the Software Level 656
24.4 Memory Summary 659
24.5 Register Summary 660
24.6 Registers 661

25 ECDSA Digital Signature Peripheral (ECDSA_DS) 664
25.1 Introduction 664
25.2 Features 664
25.3 ECDSA Basics 664
25.3.1 Domain Parameters 664
25.3.2 Key Generation 665
25.3.3 Signature Generation 665
25.3.4 Signature Verification 666
25.4 Functional Description 666
25.4.1 ECDSA_DS Working Modes 666
25.4.2 Data and Data Block 667
25.4.2.1 Writing Data 668
25.4.2.2 Reading Data 668
25.4.2.3 Padding the Message 668
25.4.2.4 Parsing the Message 668
25.4.3 Security Features 669
25.4.3.1 High Anti-Attack Performance 669
25.4.3.2 State-Dependent Register Access Control 669
25.4.3.3 Hardware Occupation 669
25.5 Programming Procedures 670
25.5.1 ECDSA_DS Process 670
25.5.1.1 IDLE Stage 670
25.5.1.2 PREP Stage 671
25.5.1.3 LOAD Stage 671
25.5.1.4 PROC Stage 671
25.5.1.5 GAIN Stage 672
25.5.1.6 POST Stage 672
25.5.1.7 ECDSA_DS SHA Interface 672
25.5.2 Clocks and Resets 674
25.5.3 Interrupts 674

25.6 Memory Blocks 675
25.7 Register Summary 676
25.8 Registers 677

26 External Memory Encryption and Decryption (XTS_AES) 683
26.1 Overview 683
26.2 Features 683
26.3 Module Structure 683
26.4 Functional Description 684
26.4.1 XTS Algorithm 684
26.4.2 Key 685
26.4.3 Target Memory Space 685
26.4.4 Data Writing 686
26.4.5 Manual Encryption Block 687
26.4.6 Auto Decryption Block 687
26.5 Software Process 688
26.6 Anti-DPA 688
26.6.1 Clock Anti-DPA Function 689
26.6.2 Pseudo-round Anti-DPA Function 689
26.7 Register Summary 691
26.8 Registers 692

27 Random Number Generator (RNG) 698
27.1 Introduction 698
27.2 Features 698
27.3 Functional Description 698
27.4 Programming Procedure 699
27.5 Register Summary 699
27.6 Register 700

V Connectivity Interface 701

28 UART Controller (UART) 702
28.1 Overview 702
28.2 Features 702
28.3 UART Structure 703
28.4 Functional Description 704
28.4.1 Clock and Reset 704
28.4.2 UART FIFO 705
28.4.3 Baud Rate Generation and Detection 705
28.4.3.1 Baud Rate Generation 705
28.4.3.2 Baud Rate Detection 706
28.4.4 UART Data Frame 707
28.4.5 AT_CMD Character Structure 708
28.4.6 RS485 708
28.4.6.1 Driver Control 708

28.4.6.2 Turnaround Delay 709
28.4.6.3 Bus Snooping 709
28.4.7 IrDA 709
28.4.8 Wake-up 710
28.4.9 Flow Control 711
28.4.9.1 Hardware Flow Control 712
28.4.9.2 Software Flow Control 713
28.4.10 GDMA Mode 714
28.4.11 UART Interrupts 714
28.4.12 UHCI Interrupts 715
28.5 Programming Procedures 716
28.5.1 Register Type 716
28.5.2 Detailed Steps 716
28.5.2.1 Initializing UART*n* 717
28.5.2.2 Configuring UART*n* Communication 717
28.5.2.3 Enabling UART*n* 718
28.6 Register Summary 719
28.6.1 UART Register Summary 719
28.6.2 UHCI Register Summary 720
28.7 Registers 722
28.7.1 UART Registers 722
28.7.2 UHCI Registers 745

29 SPI Controller (SPI) 766
29.1 Overview 766
29.2 Glossary 766
29.3 Features 767
29.4 Architectural Overview 768
29.5 Functional Description 769
29.5.1 Data Modes 769
29.5.2 Introduction to FSPI Bus Signals 769
29.5.3 Bit Read/Write Order Control 771
29.5.4 Unaligned Byte Transfer 773
29.5.5 Transfer Types 773
29.5.6 CPU-Controlled Data Transfer 773
29.5.6.1 CPU-Controlled Master Transfer 774
29.5.6.2 CPU-Controlled Slave Transfer 776
29.5.7 DMA-Controlled Data Transfer 776
29.5.7.1 GDMA Configuration 777
29.5.7.2 GDMA TX/RX Buffer Length Control 778
29.5.8 Data Flow Control 778
29.5.8.1 GP-SPI2 Functional Blocks 779
29.5.8.2 Data Flow Control as Master 780
29.5.8.3 Data Flow Control as Slave 780
29.5.9 GP-SPI2 as a Master 781
29.5.9.1 State Machine 782

29.5.9.2 Register Configuration for State and Bit Mode Control 784
29.5.9.3 Full-Duplex Communication (1-bit Mode Only) 787
29.5.9.4 Half-Duplex Communication (1/2/4-bit Mode) 788
29.5.9.5 DMA-Controlled Configurable Segmented Transfer 790
29.5.10 GP-SPI2 Works as a Slave 793
29.5.10.1 Communication Formats 794
29.5.10.2 Supported CMD Values in Half-Duplex Communication 795
29.5.10.3 Slave Single Transfer and Slave Segmented Transfer 797
29.5.10.4 Configuration of Slave Single Transfer 798
29.5.10.5 Configuration of Slave Segmented Transfer in Half-Duplex 798
29.5.10.6 Configuration of Slave Segmented Transfer in Full-Duplex 799
29.6 CS Setup Time and Hold Time Control 799
29.7 GP-SPI2 Clock Control 800
29.7.1 Clock Phase and Polarity 801
29.7.2 Clock Control as Master 803
29.7.3 Clock Control as Slave 803
29.8 GP-SPI2 Timing Compensation 803
29.9 Interrupts 804
29.10 Register Summary 807
29.11 Registers 809

30 I2C Controller (I2C) 838
30.1 Overview 838
30.2 Features 838
30.3 I2C Architecture 839
30.4 Functional Description 841
30.4.1 Clock Configuration 841
30.4.2 SCL and SDA Noise Filtering 842
30.4.3 SCL Clock Stretching 842
30.4.4 Generating SCL Pulses in Idle State 842
30.4.5 Synchronization 843
30.4.6 Open-Drain Output 844
30.4.7 Timing Parameter Configuration 845
30.4.8 Timeout Control 847
30.4.9 Command Configuration 847
30.4.10 TX/RX RAM Data Storage 848
30.4.11 Data Conversion 849
30.4.12 Addressing Mode 849
30.4.13 $R/\overline{W}$ Bit Check in 10-bit Addressing Mode 850
30.4.14 To Start the I2C Controller 850
30.5 Programming Example 851
30.5.1 $I2C_{master}$ Writes to $I2C_{slave}$ with a 7-bit Address in One Command Sequence 851
30.5.1.1 Introduction 851
30.5.1.2 Configuration Example 851
30.5.2 $I2C_{master}$ Writes to $I2C_{slave}$ with a 10-bit Address in One Command Sequence 852
30.5.2.1 Introduction 853

30.5.2.2 Configuration Example 853
30.5.3 $I2C_{master}$ Writes to $I2C_{slave}$ with Two 7-bit Addresses in One Command Sequence 854
30.5.3.1 Introduction 854
30.5.3.2 Configuration Example 855
30.5.4 $I2C_{master}$ Writes to $I2C_{slave}$ with a 7-bit Address in Multiple Command Sequences 856
30.5.4.1 Introduction 856
30.5.4.2 Configuration Example 857
30.5.5 $I2C_{master}$ Reads $I2C_{slave}$ with a 7-bit Address in One Command Sequence 858
30.5.5.1 Introduction 859
30.5.5.2 Configuration Example 859
30.5.6 $I2C_{master}$ Reads $I2C_{slave}$ with a 10-bit Address in One Command Sequence 861
30.5.6.1 Introduction 861
30.5.6.2 Configuration Example 861
30.5.7 $I2C_{master}$ Reads $I2C_{slave}$ with Two 7-bit Addresses in One Command Sequence 863
30.5.7.1 Introduction 863
30.5.7.2 Configuration Example 864
30.5.8 $I2C_{master}$ Reads $I2C_{slave}$ with a 7-bit Address in Multiple Command Sequences 865
30.5.8.1 Introduction 866
30.5.8.2 Configuration Example 867
30.6 Interrupts 869
30.7 Register Summary 870
30.8 Registers 872

31 I2S Controller (I2S) 896
31.1 Overview 896
31.2 Terminology 896
31.3 Features 897
31.4 System Architecture 898
31.5 Supported Audio Standards 900
31.5.1 TDM Philips Standard 900
31.5.2 TDM MSB Alignment Standard 901
31.5.3 TDM PCM Standard 901
31.5.4 PDM Standard 902
31.6 I2S TX/RX Clock 902
31.7 I2S Reset 904
31.8 I2S Master/Slave Mode 904
31.8.1 Master/Slave TX Mode 905
31.8.2 Master/Slave RX Mode 905
31.9 Transmitting Data 906
31.9.1 Data Format Control 906
31.9.1.1 Bit Width Control of Channel Valid Data 906
31.9.1.2 Endian Control of Channel Valid Data 907
31.9.1.3 A-law/μ-law Compression and Decompression 907
31.9.1.4 Bit Width Control of Channel TX Data 908
31.9.1.5 Bit Order Control of Channel Data 908
31.9.2 Channel Mode Control 908

31.9.2.1 TDM TX Mode 909
31.9.2.2 PDM TX Mode 910
31.10 Receiving Data 912
31.10.1 Channel Mode Control 912
31.10.1.1 TDM RX Mode 912
31.10.1.2 PDM RX Mode 913
31.10.2 Data Format Control 913
31.10.2.1 Bit Order Control of Channel Data 913
31.10.2.2 Bit Width Control of Channel Storage (Valid) Data 913
31.10.2.3 Bit Width Control of Channel RX Data 914
31.10.2.4 Endian Control of Channel Storage Data 914
31.10.2.5 A-law/μ-law Compression and Decompression 914
31.11 Event Task Matrix Feature 915
31.12 I2S Interrupts 915
31.13 Software Configuration Process 916
31.13.1 Configure I2S as TX Mode 916
31.13.2 Configure I2S as RX Mode 917
31.14 Register Summary 918
31.15 Registers 919

32 Pulse Count Controller (PCNT) 936
32.1 Features 936
32.2 Functional Description 937
32.3 Applications 939
32.3.1 Channel 0 Incrementing Independently 940
32.3.2 Channel 0 Decrementing Independently 940
32.3.3 Channel 0 and Channel 1 Incrementing Together 941
32.4 Register Summary 942
32.5 Registers 943

33 USB Serial/JTAG Controller (USB_SERIAL_JTAG) 950
33.1 Overview 950
33.2 Features 950
33.3 Functional Description 952
33.3.1 CDC-ACM USB Interface Functional Description 952
33.3.2 CDC-ACM Firmware Interface Functional Description 953
33.3.3 USB-to-JTAG Interface: JTAG Command Processor 954
33.3.4 USB-to-JTAG Interface: CMD_REP Usage Example 955
33.3.5 USB-to-JTAG Interface: Response Capture Unit 956
33.3.6 USB-to-JTAG Interface: Control Transfer Requests 956
33.4 Recommended Operation 957
33.5 Interrupts 958
33.6 Register Summary 960
33.7 Registers 962

34 Two-wire Automotive Interface (TWAI) 986

34.1 Features 986
34.2 Protocol Overview 987
34.2.1 TWAI Properties 987
34.2.2 TWAI Messages 988
34.2.2.1 Data Frames and Remote Frames 988
34.2.2.2 Error and Overload Frames 990
34.2.2.3 Interframe Space 992
34.2.3 TWAI Errors 992
34.2.3.1 Error Types 992
34.2.3.2 Error States 993
34.2.3.3 Error Counters 993
34.2.4 TWAI Bit Timing 994
34.2.4.1 Nominal Bit 994
34.2.4.2 Hard Synchronization and Resynchronization 995
34.3 Architectural Overview 996
34.3.1 Registers Block 997
34.3.2 Bit Stream Processor 998
34.3.3 Error Management Logic 998
34.3.4 Bit Timing Logic 998
34.3.5 Acceptance Filter 998
34.3.6 Receive FIFO 998
34.4 Functional Description 998
34.4.1 Modes 999
34.4.1.1 Reset Mode 999
34.4.1.2 Operation Mode 999
34.4.2 Bit Timing 999
34.4.3 Interrupt Management 1000
34.4.3.1 Receive Interrupt (RXI) 1001
34.4.3.2 Transmit Interrupt (TXI) 1001
34.4.3.3 Error Warning Interrupt (EWI) 1001
34.4.3.4 Data Overrun Interrupt (DOI) 1001
34.4.3.5 Error Passive Interrupt (EPI) 1002
34.4.3.6 Arbitration Lost Interrupt (ALI) 1002
34.4.3.7 Bus Error Interrupt (BEI) 1002
34.4.3.8 Bus Idle Status Interrupt (BISI) 1002
34.4.4 Transmit and Receive Buffers 1002
34.4.4.1 Overview of Buffers 1002
34.4.4.2 Frame Information 1003
34.4.4.3 Frame Identifier 1004
34.4.4.4 Frame Data 1005
34.4.5 Receive FIFO and Data Overruns 1005
34.4.6 Acceptance Filter 1006
34.4.6.1 Single Filter Mode 1006
34.4.6.2 Dual Filter Mode 1007
34.4.7 Error Management 1008
34.4.7.1 Error Warning Limit 1009

34.4.7.2 Error Passive 1009
34.4.7.3 Bus-Off and Bus-Off Recovery 1009
34.4.8 Error Code Capture 1010
34.4.9 Arbitration Lost Capture 1011
34.4.10 Transceiver Auto-Standby 1012
34.5 Register Summary 1013
34.6 Registers 1014

35 LED PWM Controller (LEDC) 1029
35.1 Overview 1029
35.2 Features 1029
35.3 Functional Description 1030
35.3.1 Architecture 1030
35.3.2 Timers 1031
35.3.2.1 Clock Source 1031
35.3.2.2 Clock Divider Configuration 1031
35.3.2.3 20-Bit Counter 1032
35.3.3 PWM Generators 1033
35.3.4 Duty Cycle Fading 1035
35.3.4.1 Linear Duty Cycle Fading 1035
35.3.4.2 Gamma Curve Fading 1036
35.3.4.3 Suspend and Resume Duty Cycle Fading 1038
35.3.5 Event Task Matrix Feature 1038
35.3.6 Interrupts 1040
35.4 Register Summary 1041
35.5 Registers 1044

36 Motor Control PWM (MCPWM) 1057
36.1 Overview 1057
36.2 Features 1057
36.3 Modules 1060
36.3.1 Overview 1060
36.3.1.1 Prescaler Module 1060
36.3.1.2 Timer Module 1060
36.3.1.3 Operator Module 1061
36.3.1.4 Fault Detection Module 1062
36.3.1.5 Capture Module 1063
36.3.1.6 ETM Module 1063
36.3.2 PWM Timer Module 1063
36.3.2.1 Configurations of the PWM Timer Module 1063
36.3.2.2 PWM Timer's Working Modes and Timing Event Generation 1064
36.3.2.3 Shadow Register of PWM Timer 1069
36.3.2.4 PWM Timer Synchronization and Phase Locking 1069
36.3.3 PWM Operator Module 1069
36.3.3.1 PWM Generator Module 1071
36.3.3.2 Dead Time Generator Module 1082

36.3.3.3 PWM Carrier Module 1086
36.3.3.4 Fault Detection Module 1088
36.3.4 Capture Module 1089
36.3.4.1 Introduction 1089
36.3.4.2 Capture Timer 1090
36.3.4.3 Capture Channel 1090
36.3.5 ETM Module 1090
36.3.5.1 Overview 1090
36.3.5.2 MCPWM-Related ETM Events 1090
36.3.5.3 MCPWM-Related ETM Tasks 1091
36.3.6 Interrupts 1092
36.4 Register Summary 1093
36.5 Registers 1096

37 Remote Control Peripheral (RMT) 1172
37.1 Overview 1172
37.2 Features 1172
37.3 Functional Description 1173
37.3.1 RMT Architecture 1173
37.3.2 RMT RAM 1174
37.3.2.1 Structure of RAM 1174
37.3.2.2 Use of RAM 1174
37.3.2.3 RAM Access 1175
37.3.3 Clock 1175
37.3.4 Transmitter 1176
37.3.4.1 Normal TX Mode 1176
37.3.4.2 Wrap TX Mode 1176
37.3.4.3 TX Modulation 1176
37.3.4.4 Continuous TX Mode 1177
37.3.4.5 Simultaneous TX Mode 1177
37.3.5 Receiver 1177
37.3.5.1 Normal RX Mode 1177
37.3.5.2 Wrap RX Mode 1178
37.3.5.3 RX Filtering 1178
37.3.5.4 RX Demodulation 1178
37.3.6 Configuration Update 1179
37.3.7 Interrupts 1179
37.4 Register Summary 1181
37.5 Registers 1183

38 Parallel IO Controller (PARLIO) 1198
38.1 Introduction 1198
38.2 Glossary 1198
38.3 Features 1198
38.4 Architectural Overview 1200
38.5 Functional Description 1200

38.5.1 Clock Generator 1200
38.5.2 Clock & Reset Restriction 1201
38.5.3 Master-Slave Mode 1202
38.5.4 Receive Modes of the RX Unit 1203
38.5.4.1 Level Enable Mode 1204
38.5.4.2 Pulse Enable Mode 1204
38.5.4.3 Software Enable Mode 1205
38.5.5 RX Unit GDMA SUC EOF Generation 1205
38.5.6 RX Unit Timeout 1206
38.5.7 Unlimited Length Data Transmission of TX Unit 1206
38.5.8 Output Clock Gating of TX Unit 1206
38.5.9 Valid Signal Output of TX Unit 1207
38.5.10 Bus Idle Value of TX Unit 1207
38.5.11 Data Transfer in a Single Frame 1207
38.5.12 Bit Reversal in One Byte 1208
38.6 Programming Procedures 1208
38.6.1 Data Receiving Operation Process 1208
38.6.2 Data Transmitting Operation Process 1208
38.7 Application Examples 1209
38.7.1 Co-working with SPI 1209
38.7.2 Co-working with I2S 1211
38.8 Interrupts 1212
38.9 Register Summary 1213
38.10 Registers 1214

VI Analog Signal Processing 1224

39 Temperature Sensor (TSENS) 1225
39.1 Overview 1225
39.2 Features 1225
39.3 Architecture 1225
39.4 Functional Description 1226
39.4.1 Temperature Sensor Power Up 1226
39.4.2 Temperature Sensor Clock 1226
39.4.3 Wake-Up Modes for Automatic Temperature Monitoring 1226
39.4.4 Temperature Measurement Range and Offset 1227
39.4.5 Data Conversion 1227
39.5 Programming Procedure 1227
39.6 Interrupts 1228
39.7 Event Task Matrix Feature 1228
39.8 Register Summary 1229
39.9 Registers 1230

40 ADC Controller (ADC) 1234
40.1 Overview 1234
40.2 Terminology 1234

40.3 Features 1234
40.4 Architecture 1235
40.5 Functional Description 1236
40.5.1 ADC Power Up 1236
40.5.2 ADC Channels 1236
40.5.3 ADC Clock 1236
40.5.4 One-Shot Sampling Mode 1237
40.5.5 Multi-Channel Sampling Mode 1237
40.5.6 ADC Conversion and Attenuation 1237
40.5.7 DIG ADC FSM 1238
40.5.8 Pattern Table 1239
40.5.9 ADC Filters 1240
40.5.10 Threshold Monitors 1241
40.5.11 GDMA Support 1241
40.6 Programming Procedure 1242
40.6.1 Configuring One-shot Sampling Mode 1242
40.6.2 Configuring Multi-Channel Sampling Mode 1242
40.7 Interrupts 1242
40.8 Event Task Matrix Feature 1243
40.9 Register Summary 1244
40.10 Registers 1245

VII Appendix 1258

Related Documentation and Resources 1259

Glossary 1261
Abbreviations for Peripherals 1261
Abbreviations Related to Registers 1261
Access Types for Registers 1263

Programming Reserved Register Field 1265
Introduction 1265
Programming Reserved Register Field 1265

Interrupt Configuration Registers 1266

Revision History 1267

List of Tables

1.4-1 CPU Address Map 39
1.7-1 Core Local Interrupt (CLINT) Sources 59
1.11-1 NAPOT encoding for maddress 75

2.3-1 Trace Encoder Parameters 88
2.6-1 Header Format 91
2.6-2 Index Format 91
2.6-3 Packet format 3 subformat 0 91
2.6-4 Packet format 3 subformat 1 92
2.6-5 Packet format 3 subformat 3 92
2.6-6 Packet format 2 93
2.6-7 Packet format 1 with address 93
2.6-8 Packet format 1 without address 94

3.4-1 Selecting Peripherals via Register Configuration 106
3.4-2 Descriptor Field Alignment Requirements 109

4.3-1 Memory Address Mapping 144
4.3-2 Module/Peripheral Address Mapping 147

5.3-1 Parameters in eFuse BLOCK0 152
5.3-2 Secure Key Purpose Values 155
5.3-3 Parameters in BLOCK1 to BLOCK10 156
5.3-4 Registers Information 161
5.3-5 Configuration of Default VDDQ Timing Parameters 162

6.8-1 Bit Used to Control IO MUX Functions in Light-sleep Mode 227
6.12-1 Peripheral Signals via GPIO Matrix 231
6.13-1 IO MUX Functions List 236
6.14-1 Analog Functions of IO MUX Pins 237

7.1-1 Reset Source 273
7.2-1 CPU_CLK Clock Source 275
7.2-2 Frequency of CPU_CLK, AHB_CLK and HP_ROOT_CLK 276
7.2-3 Derived HP Clock Source 278
7.2-4 HP Clocks Used by Each Peripheral 278
7.2-5 Derived LP Clock Source 279
7.2-6 LP Clocks Used by Each Peripheral 279

8.2-1 Default Configuration of Strapping Pins 347
8.2-2 Boot Mode Control 347
8.2-3 ROM Message Printing Control 349
8.2-4 JTAG Signal Source Control 350

9.5-1 CPU Peripheral Interrupt Source Mapping/Status Registers and Peripheral Interrupt Sources 354

10.3-1 Selectable Events for ETM Channel*n* 372
10.3-2 Mappable Tasks for ETM Channel*n* 375

11.4-1 Wake-up Sources 394
11.4-2 HP System Peripherals' Function Clocks 397
11.4-3 HP System Peripherals' APB Clocks 397
11.4-4 Trigger Conditions for the RTC Timer 400
11.5-1 Preset Power Modes 401

12.4-1 UNIT*n* Configuration Bits 442
12.4-2 Trigger Point 443
12.4-3 Synchronization Operation for Configuration Registers 444

13.3-1 Alarm Generation When Up-Down Counter Increments 466
13.3-2 Alarm Generation When Up-Down Counter Decrements 467

14.2-1 Timeout Actions 492

15.1-1 Management Areas of PMP and AMP 505
15.3-2 Comparison Between TEE and REE 507
15.4-1 Master Access Source 507
15.4-2 Configuring Functional Modules 508

16.2-1 Security Level 532

17.4-1 CPU Packet Format 547
17.4-2 DMA Packet Format 547
17.4-3 LOST Packet Format 547

18.3-1 Configuration for Detecting Power Pins 576
18.4-1 Power Supply Detector's Internal Interrupt Source 576

19.4-1 AES Accelerator Working Mode 586
19.4-2 Key Length and Encryption/Decryption 586
19.5-1 Working Status under Typical AES Working Mode 587
19.5-2 Text Endianness Type for Typical AES 587
19.5-3 Key Endianness Type for AES-128 Encryption and Decryption 588
19.5-4 Key Endianness Type for AES-256 Encryption and Decryption 588
19.6-1 Block Cipher Mode 589
19.6-2 Working Status under DMA-AES Working mode 590
19.6-3 TEXT-PADDING 590
19.6-4 Text Endianness for DMA-AES 591

20.3-1 ECC Accelerator Memory Blocks 602
20.4-1 ECC Accelerator Key Size Selection 603
20.4-2 Working Modes of ECC Accelerator 604

21.2-1 HMAC Purposes and Configuration Value 617

22.3-1 Acceleration Performance 635

22.4-1 RSA Accelerator Memory Blocks 637

23.3-1 SHA Accelerator Working Mode 643
23.3-2 SHA Hash Algorithm Selection 643
23.4-1 The Storage and Length of Message Digest from Different Algorithms 647

25.4-1 ECDSA_DS Working Mode 667
25.4-2 ECDSA_DS Elliptic Curve Selection 667
25.4-3 ECDSA_DS SHA Algorithm 667
25.4-4 ECDSA_DS Working State 667
25.6-1 ECDSA_DS Memory Blocks 675

26.4-1 Key Generated Based on Key_A 685
26.4-2 Mapping Between Offsets and Plaintext Memory Addresses 686
26.6-1 Configuration of XTS_AES Pseudo-round Anti-DPA 690

28.4-1 UART_CHAR_WAKEUP Mode Configuration 711

29.5-1 Data Modes Supported by GP-SPI2 769
29.5-2 Functional Description of FSPI Bus Signals 769
29.5-3 Signals Used in Various SPI Modes 770
29.5-4 Bit Order Control in GP-SPI2 772
29.5-5 Supported Transfer Types as Master or Slave 773
29.5-6 Interrupt Trigger Condition on GP-SPI2 Data Transfer as Slave 777
29.5-7 Registers Used for State Control in 1/2/4-bit Modes 784
29.5-7 Registers Used for State Control in 1/2/4-bit Modes 785
29.5-8 Sending Sequence of Command Value 786
29.5-9 Sending Sequence of Address Value 787
29.5-10 BM Table for CONF State 792
29.5-11 An Example of CONF buffer*i* in Segment*i* 793
29.5-12 BM Bit Value v.s. Register to Be Updated in This Example 793
29.5-13 Supported CMD Values in SPI Mode 796
29.5-13 Supported CMD Values in SPI Mode 797
29.5-14 Supported CMD Values in QPI Mode 797
29.7-1 Clock Phase and Polarity Configuration as Master 803
29.7-2 Clock Phase and Polarity Configuration as Slave 803
29.9-1 GP-SPI2 Interrupts as Master 805
29.9-2 GP-SPI2 Interrupts as Slave 806

30.3-1 I2C Timing Parameters (Cited from Table 5 in The I2C-bus specification Version 2.1) 841
30.4-1 I2C Synchronous Registers 844

31.4-1 I2S Signal Description 899
31.9-1 Bit Width of Channel Valid Data 906
31.9-2 Endian of Channel Valid Data 907
31.9-3 Data-Fetching Control in PDM Mode 910
31.9-4 I2S Channel Control for Raw PDM Data 910
31.9-5 I2S PCM-to-PDM Data Output 911

31.10-1 Channel Storage Data Width 913
31.10-2 Channel Storage Data Endian 914

32.2-1 Counter Mode. Positive Edge of Input Pulse Signal. Control Signal in Low State 938
32.2-2 Counter Mode. Positive Edge of Input Pulse Signal. Control Signal in High State 938
32.2-3 Counter Mode. Negative Edge of Input Pulse Signal. Control Signal in Low State 938
32.2-4 Counter Mode. Negative Edge of Input Pulse Signal. Control Signal in High State 939

33.3-1 Standard CDC-ACM Control Requests 952
33.3-2 CDC-ACM Settings with RTS and DTR 953
33.3-3 Commands of a Nibble 955
33.3-4 USB-to-JTAG Control Requests 956
33.3-5 JTAG Capability Descriptors 957
33.4-1 Reset SoC into Download Mode 958
33.4-2 Reset SoC into Booting from flash 958

34.2-1 Data Frames and Remote Frames in SFF and EFF 989
34.2-2 Error Frame 991
34.2-3 Overload Frame 991
34.2-4 Interframe Space 992
34.2-5 Segments of a Nominal Bit Time 995
34.4-1 Bit Information of TWAI_BUS_TIMING_0_REG (0x18) 999
34.4-2 Bit Information of TWAI_BUS_TIMING_1_REG (0x1c) 1000
34.4-3 Buffer Layout for Standard Frame Format and Extended Frame Format 1002
34.4-4 TX/RX Frame Information (SFF/EFF);TWAI Address 0x40 1003
34.4-5 TX/RX Identifier 1 (SFF); TWAI Address 0x44 1004
34.4-6 TX/RX Identifier 2 (SFF); TWAI Address 0x48 1004
34.4-7 TX/RX Identifier 1 (EFF); TWAI Address 0x44 1005
34.4-8 TX/RX Identifier 2 (EFF); TWAI Address 0x48 1005
34.4-9 TX/RX Identifier 3 (EFF); TWAI Address 0x4c 1005
34.4-10 TX/RX Identifier 4 (EFF); TWAI Address 0x50 1005
34.4-11 Bit Information of TWAI_ERR_CODE_CAP_REG (0x30) 1010
34.4-12 Bit Information of Bits SEG.4 - SEG.0 1010
34.4-13 Bit Information of TWAI_ARB_LOST_CAP_REG (0x2c) 1011

35.3-1 Commonly-used Frequencies and Resolutions 1033

36.3-1 Configuration Parameters of the Operator Submodule 1061
36.3-2 Timing Events Used in PWM Generator 1072
36.3-3 Timing Events Priority When PWM Timer Increments 1072
36.3-4 Timing Events Priority when PWM Timer Decrements 1073
36.3-5 Dead Time Generator Switches Control Fields 1083
36.3-6 Typical Dead Time Generator Operating Modes 1083
36.3-7 MCPWM-Related ETM Events 1090
36.3-8 MCPWM-Related ETM Tasks 1091

37.3-1 Configuration Update 1179

38.5-1 Operations to Reset AHB Clock Domain with Clock Restrictions 1201
38.5-2 Requirements for TX Unit Operating as Slave with Clock Restrictions 1203
38.5-3 Requirements for RX Unit Operating as Slave with Clock Restrictions 1203

39.4-1 Temperature Measurement Range and Offset 1227

40.5-1 SAR ADC Channels 1236

40.10-4 Configuration of ENA/RAW/ST Registers 1266

List of Figures

1.1-1 CPU Block Diagram 38
1.10-1 Debug System Overview 69

2.0-1 Trace Encoder Overview 86
2.2-1 Trace Overview 87
2.6-1 Trace packet Format 90

3.1-1 Modules with GDMA Feature and GDMA Channels 103
3.3-1 GDMA controller Architecture 104
3.4-1 Structure of a Linked List 105
3.4-2 Relationship among Linked Lists 108

4.2-1 System Structure and Address Mapping 143
4.3-1 Cache Structure 145
4.3-2 Modules/peripherals that can work with GDMA 147

5.3-1 Data Flow in eFuse 151
5.3-2 Shift Register Circuit (first 32 output) 158
5.3-3 Shift Register Circuit (last 12 output) 158

6.3-1 Architecture of IO MUX and GPIO Matrix 219
6.3-2 Internal Structure of a Pad 220
6.4-1 GPIO Input Synchronized on Rising Edge or on Falling Edge of IO MUX Operating Clock 221
6.4-2 GPIO Filter Timing of GPIO Input Signals 222
6.4-3 Glitch Filter Timing Example 222
6.10-1 Example of level flip on the chip pad when the hysteresis function is not enabled 228
6.10-2 Example of level flip on the chip pad when the hysteresis function is enabled 229

7.1-1 Reset Types 271
7.2-1 System Clock 274
7.3-1 Clock Configuration Example 281

8.2-1 Chip Boot Flow 348

9.2-1 Interrupt Flow in ESP32-H2 352
9.4-1 Interrupt Matrix Structure 352

10.3-1 Event Task Matrix Architecture 371
10.3-2 ETM Channel*n* Architecture 371
10.3-3 Event Task Matrix Clock Architecture 378

11.4-1 ESP32-H2 Power Scheme 390
11.4-2 PMU Workflow 392
11.6-1 ESP32-H2 Boot Flow 402

12.1-1 System Timer Structure 440
12.4-1 System Timer Alarm Generation 441

13.1-1 Timer Group Overview 464
13.3-1 Timer Group Architecture 465

14.1-1 Watchdog Timers Overview 489
14.2-1 Digital Watchdog Timers in ESP32-H2 491
14.3-1 Super Watchdog Controller Structure 494

15.1-1 PMP-APM Management Relation 505
15.4-1 APM Controller Architecture 508

18.3-1 Architecture of Power Supply Detector 574
18.3-2 Brown-out Reset Workflow 575
18.3-3 Structure of Voltage Glitch Detectors 576

21.3-1 HMAC SHA-256 Padding Diagram 620
21.3-2 HMAC Structure Schematic Diagram 621

24.3-1 Software Preparations and Hardware Working Process 655

25.5-1 ECDSA_DS Process 670

26.3-1 Architecture of the External Memory Encryption and Decryption 684

27.3-1 Noise Source 698

28.3-1 UART Structure 703
28.4-1 UART Controllers Division 706
28.4-2 The Timing Diagram of Weak UART Signals Along Negative Edges 706
28.4-3 Structure of UART Data Frame 707
28.4-4 AT_CMD Character Structure 708
28.4-5 Driver Control Diagram in RS485 Mode 709
28.4-6 The Timing Diagram of Encoding and Decoding in SIR mode 710
28.4-7 IrDA Encoding and Decoding Diagram 710
28.4-8 Hardware Flow Control Diagram 712
28.4-9 Connection between Hardware Flow Control Signals 713
28.4-10 Data Transfer in GDMA Mode 714
28.5-1 UART Programming Procedures 717

29.4-1 SPI Module Overview 768
29.5-1 Data Buffer Used in CPU-Controlled Transfer 774
29.5-2 GP-SPI2 Block Diagram 779
29.5-3 Data Flow Control in GP-SPI2 as Master 780
29.5-4 Data Flow Control in GP-SPI2 as Slave 780
29.5-5 GP-SPI2 State Machine as Master 783
29.5-6 Full-Duplex Communication Between GP-SPI2 Master and a Slave 787
29.5-7 Connection of GP-SPI2 to Flash and External RAM in 4-bit Mode 789
29.5-8 SPI Quad I/O Read Command Sequence Sent by GP-SPI2 to Flash 790
29.5-9 Configurable Segmented Transfer as Master 790
29.6-1 Recommended CS Timing and Settings When Accessing External RAM 800

29.6-2 Recommended CS Timing and Settings When Accessing Flash 800
29.7-1 SPI Clock Mode 0 or 2 802
29.7-2 SPI Clock Mode 1 or 3 802

30.3-1 I2C Master Architecture 839
30.3-2 I2C Slave Architecture 839
30.3-3 I2C Protocol Timing (Cited from Fig.31 in The I2C-bus specification Version 2.1) 840
30.4-1 I2C Timing Diagram 845
30.4-2 Structure of I2C Command Registers 847
30.5-1 $I2C_{master}$ Writing to $I2C_{slave}$ with a 7-bit Address 851
30.5-2 $I2C_{master}$ Writing to a Slave with a 10-bit Address 853
30.5-3 $I2C_{master}$ Writing to $I2C_{slave}$ with Two 7-bit Addresses 854
30.5-4 $I2C_{master}$ Writing to $I2C_{slave}$ with a 7-bit Address in Multiple Sequences 856
30.5-5 $I2C_{master}$ Reading $I2C_{slave}$ with a 7-bit Address 859
30.5-6 $I2C_{master}$ Reading $I2C_{slave}$ with a 10-bit Address 861
30.5-7 $I2C_{master}$ Reading N Bytes of Data from addrM of $I2C_{slave}$ with a 7-bit Address 863
30.5-8 $I2C_{master}$ Reading $I2C_{slave}$ with a 7-bit Address in Segments 866

31.4-1 ESP32-H2 I2S System Diagram 898
31.5-1 TDM Philips Standard Timing Diagram 901
31.5-2 TDM MSB Alignment Standard Timing Diagram 901
31.5-3 TDM PCM Standard Timing Diagram 902
31.5-4 PDM Standard Timing Diagram 902
31.6-1 I2S Clock Generator 903
31.9-1 TX Data Format Control 908
31.9-2 TDM Channel Control 909
31.9-3 PDM Channel Control Example 912

32.0-1 PCNT Block Diagram 936
32.2-1 PCNT Unit Architecture 937
32.3-1 Channel 0 Up Counting Diagram 940
32.3-2 Channel 0 Down Counting Diagram 940
32.3-3 Two Channels Up Counting Diagram 941

33.2-1 USB Serial/JTAG High Level Diagram 951
33.2-2 USB Serial/JTAG Block Diagram 952

34.2-1 Bit Fields in Data Frames and Remote Frames 989
34.2-2 Fields of an Error Frame 991
34.2-3 Fields of an Overload Frame 991
34.2-4 The Fields within an Interframe Space 992
34.2-5 Layout of a Bit 995
34.3-1 TWAI Overview Diagram 996
34.3-2 TWAI Clock Generation 997
34.4-1 Acceptance Filter 1007
34.4-2 Single Filter Mode 1007
34.4-3 Dual Filter Mode 1008
34.4-4 Error State Transition 1009

34.4-5 Positions of Arbitration Lost Bits 1011

35.2-1 LED PWM Architecture 1030
35.3-1 Timer and PWM Generator Block Diagram 1030
35.3-2 Frequency Division When LEDC_CLK_DIV is a Non-Integer Value 1032
35.3-3 Relationship Between Counter And Resolution 1032
35.3-4 LED PWM Output Signal Diagram 1034
35.3-5 Output Signal of Linear Duty Cycle Fading 1036
35.3-6 Output Signal of Gamma Curve Fading 1038

36.2-1 MCPWM Module Overview 1058
36.3-1 Prescaler Module 1060
36.3-2 Timer Module 1060
36.3-3 Operator Module 1061
36.3-4 Fault Detection Module 1062
36.3-5 Capture Module 1063
36.3-6 ETM Module 1063
36.3-7 Count-Up Mode Waveform 1064
36.3-8 Count-Down Mode Waveforms 1065
36.3-9 Count-Up-Down Mode Waveforms, Count-Down at Synchronization Event 1065
36.3-10 Count-Up-Down Mode Waveforms, Count-Up at Synchronization Event 1065
36.3-11 UTEP and UTEZ Generation in Count-Up Mode 1066
36.3-12 DTEP and DTEZ Generation in Count-Down Mode 1067
36.3-13 DTEP and UTEZ Generation in Count-Up-Down Mode 1067
36.3-14 Block Diagram of A PWM Operator 1070
36.3-15 Symmetrical Waveform in Count-Up-Down Mode 1074
36.3-16 Count-Up, Single Edge Asymmetric Waveform, with Independent Modulation on PWM*x*A and PWM*x*B — Active High 1075
36.3-17 Count-Up, Pulse Placement Asymmetric Waveform with Independent Modulation on PWM*x*A 1076
36.3-18 Count-Up-Down, Dual Edge Symmetric Waveform, with Independent Modulation on PWM*x*A and PWM*x*B — Active High 1077
36.3-19 Count-Up-Down, Dual Edge Symmetric Waveform, with Independent Modulation on PWM*x*A and PWM*x*B — Complementary 1078
36.3-20 Count-Up-Down, Fault or Synchronization Events, with Same Modulation on PWM*x*A and PWM*x*B 1079
36.3-21 Example of an NCI Software-Force Event on PWM*x*A 1080
36.3-22 Example of a CNTU Software-Force Event on PWM*x*B 1081
36.3-23 Options for Setting up the Dead Time Generator Module 1083
36.3-24 Active High Complementary (AHC) Dead Time Waveforms 1084
36.3-25 Active Low Complementary (ALC) Dead Time Waveforms 1085
36.3-26 Active High (AH) Dead Time Waveforms 1085
36.3-27 Active Low (AL) Dead Time Waveforms 1085
36.3-28 Example of Waveforms Showing PWM Carrier Action 1086
36.3-29 Example of the First Pulse and the Subsequent Sustaining Pulses of the PWM Carrier Submodule 1087
36.3-30 Possible Duty Cycle Settings for Sustaining Pulses in the PWM Carrier Submodule 1088

37.3-1 RMT Architecture 1173

37.3-2 Format of Pulse Code in RAM 1174

38.4-1 PARLIO Architecture 1200
38.5-1 PARLIO Clock Generation 1201
38.5-2 Positive Waveform 1203
38.5-3 Negative Waveform 1203
38.5-4 Sub-Modes of Level Enable Mode for RX Unit 1204
38.5-5 Sub-Modes of Pulse Enable Mode for RX Unit 1205
38.5-6 Sub-Mode of Software Enable Mode for RX Unit 1205

39.3-1 Temperature Sensor Architecture 1226

40.4-1 SAR ADC Architecture 1235
40.5-1 SAR ADC Clock Structure 1236
40.5-2 DIG ADC FSM Block Diagram 1238
40.5-3 APB_SARADC_SAR_PATT_TAB1_REG Contains Patterns 0 - 3 1239
40.5-4 APB_SARADC_SAR_PATT_TAB2_REG Contains Patterns 4 - 7 1239
40.5-5 Pattern Structure 1239
40.5-6 cmd0 configuration 1240
40.5-7 cmd1 Configuration 1240
40.5-8 DMA Data Format 1241

Part I

Microprocessor and Master

This part covers the essential processing elements of the system, diving into ESP-RISC-V CPU, RISC-V Trace Encoder, and controllers for General Direct Memory Access (GDMA).

Chapter 1

ESP-RISC-V CPU

1.1 Overview

ESP-RISC-V CPU is a 32-bit core based upon RISC-V instruction set architecture (ISA) comprising base integer (I), multiplication/division (M), atomic (A) and compressed (C) standard extensions. The core has 4-stage, in-order, scalar pipeline optimized for area, power and performance. CPU core complex has a debug module (DM), interrupt-controller (INTC), core local interrupts (CLINT) and system bus (SYS BUS) interfaces for memory and peripheral access.

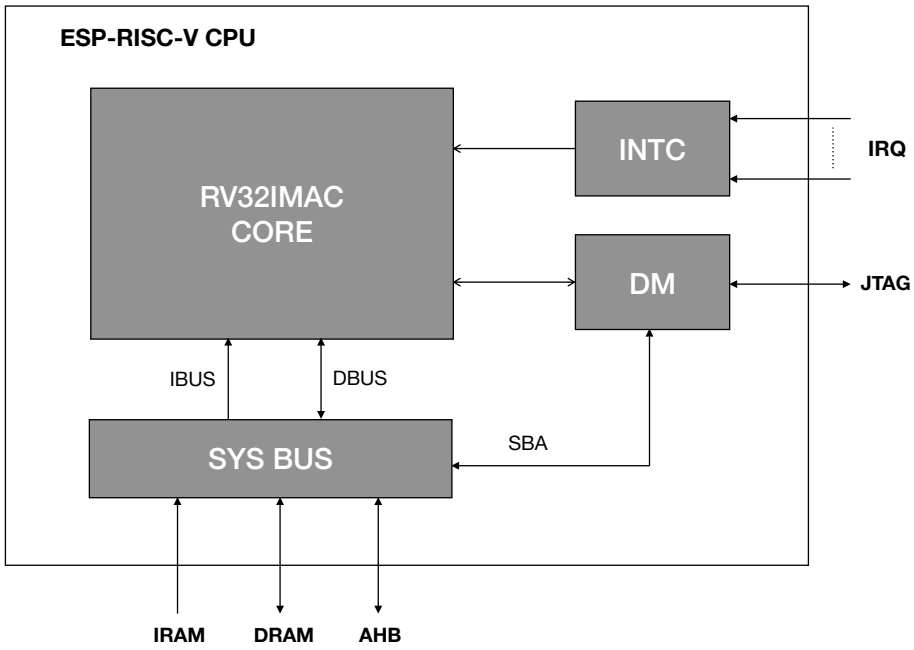


Figure 1.1-1. CPU Block Diagram

1.2 Features

- RISC-V RV32IMAC ISA with four-stage pipeline that supports an operating clock frequency up to 96 MHz
- Compatible with RISC-V ISA Manual Volume I: Unprivileged ISA Version 2.2 and RISC-V ISA Manual, Volume II: Privileged Architecture, Version 1.10
- Zero wait cycle access to on-chip SRAM and Cache for program and data access over IRAM/DRAM interface

- Branch target buffer (BTB) with static branch prediction
- User (U) mode support along with interrupt delegation
- Interrupt controller with up to 28 external vectored interrupts for both M and U modes with 16 programmable priority and threshold levels
- Core local interrupts (CLINT) dedicated for each privilege mode
- Debug module (DM) compliant with the specification RISC-V External Debug Support Version 0.13 with external debugger support over an industry-standard JTAG/USB port
- Support for instruction trace
- Debugger with a direct system bus access (SBA) to memory and peripherals
- Hardware trigger compliant to the specification RISC-V External Debug Support Version 0.13 with up to 4 breakpoints/watchpoints
- Physical memory protection (PMP) and attributes (PMA) for up to 16 configurable regions
- 32-bit AHB system bus for peripheral access
- Configurable events for core performance metrics

1.3 Terminology

branch an instruction which conditionally changes the execution flow

delta a change in the program counter (PC) that is other than the difference between two instructions placed consecutively in memory

hart a RISC-V hardware thread

retire the final stage of executing an instruction, when the machine state is updated

trap the transfer of control to a trap handler caused by either an exception or an interrupt

1.4 Address Map

Below table shows address map of various regions accessible by CPU for instruction, data, system bus peripheral and debug.

Table 1.4-1. CPU Address Map

Name	Description	Starting Address	Ending Address	Access
IRAM/DRAM	Instruction/Data region	0x4000_0000	0x4FFF_FFFF	R/W
CPU	CPU Sub-system region	0x2000_0000	0x2FFF_FFFF	R/W
AHB	AHB Peripheral region	*default	*default	R/W

*default: Address not matching any of the specified ranges (IRAM, DRAM, CPU) are accessed using AHB bus.

1.5 Configuration and Status Registers (CSRs)

1.5.1 Register Summary

Below is a list of CSRs available to the CPU. Except for the custom performance counter CSRs, all the implemented CSRs follow the standard mapping of bit fields as described in the RISC-V Instruction Set Manual, Volume II: Privileged Architecture, Version 1.10. It must be noted that even among the standard CSRs, not all bit fields have been implemented, limited by the subset of features implemented in the CPU. Refer to the next section for a 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	0x303	R/W
mie	Machine Interrupt Enable Register	0x304	R/W
mtvec [2]	Machine Trap Vector	0x305	R/W
Machine Trap Handling CSRs			
mscratch	Machine Scratch	0x340	R/W
mepc	Machine Trap Program Counter	0x341	R/W
mcause [3]	Machine Trap Cause	0x342	R/W
mtval	Machine Trap Value	0x343	R/W
mip	Machine Interrupt Pending	0x344	R/W
User Trap Setup CSRs			
ustatus	User Mode Status	0x000	R/W
uie	User Interrupt Enable Register	0x004	R/W
utvec	User Trap Vector	0x005	R/W
User Trap Handling CSRs			
uscratch	User Scratch	0x040	R/W
uepc	User Trap Program Counter	0x041	R/W
ucause	User Trap Cause	0x042	R/W
uip	User Interrupt Pending	0x044	R/W
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

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

[2] mtvec only provides configuration for trap handling in vectored mode with the base address aligned to 256 bytes

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

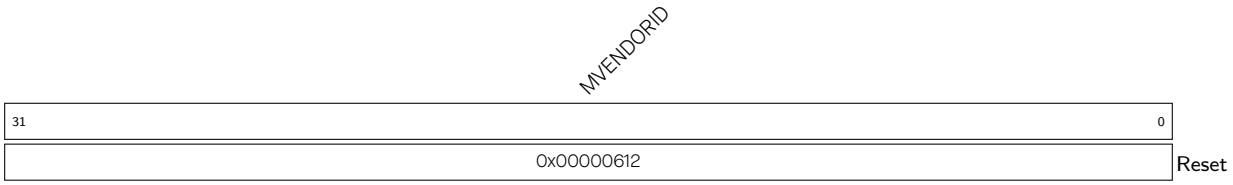
Name	Description	Address	Access
pmpcfg3	Physical memory protection configuration	0x3A3	R/W
pmpaddr0	Physical memory protection address register	0x3B0	R/W
pmpaddr1	Physical memory protection address register	0x3B1	R/W
....			
pmpaddr15	Physical memory protection address register	0x3BF	R/W
Trigger Module 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
tcontrol	Global Trigger Control	0x7A5	R/W
Debug Mode CSRs			
dcsr	Debug Control and Status	0x7B0	R/W
dpc	Debug PC	0x7B1	R/W
dscratch0	Debug Scratch Register 0	0x7B2	R/W
dscratch1	Debug Scratch Register 1	0x7B3	R/W
Performance Counter CSRs (Custom) [4]			
mpcer	Machine Performance Counter Event	0x7E0	R/W
mpcmr	Machine Performance Counter Mode	0x7E1	R/W
mpccr	Machine Performance Counter Count	0x7E2	R/W
GPIO Access CSRs (Custom)			
cpu_gpio_oen	GPIO Output Enable	0x803	R/W
cpu_gpio_in	GPIO Input Value	0x804	RO
cpu_gpio_out	GPIO Output Value	0x805	R/W
Physical Memory Attributes Checker (PMAC) CSRs			
pma_cfg0	Physical memory attribute configuration	0xBC0	R/W
pma_cfg1	Physical memory attribute configuration	0xBC1	R/W
pma_cfg2	Physical memory attribute configuration	0xBC2	R/W
pma_cfg3	Physical memory attribute configuration	0xBC3	R/W
....			
pma_cfg15	Physical memory attribute configuration	0xBCF	R/W
pma_addr0	Physical memory attribute address register	0xBD0	R/W
pma_addr1	Physical memory attribute address register	0xBD1	R/W
....			
pma_addr15	Physical memory attribute address register	0xBDF	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 illegal instruction exception.

[4] These custom CSRs have been implemented in the address space reserved by RISC-V standard for custom use

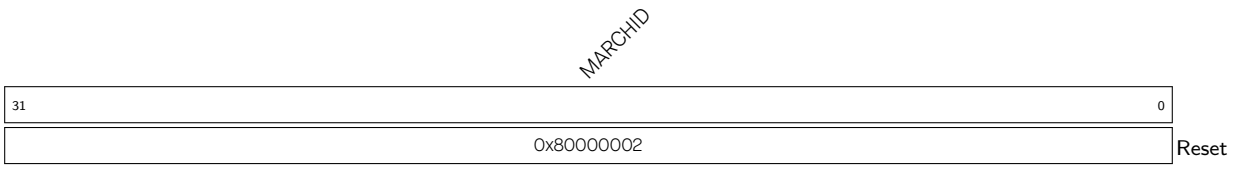
1.5.2 Register Description

Register 1.1. mvendorid (0xF11)



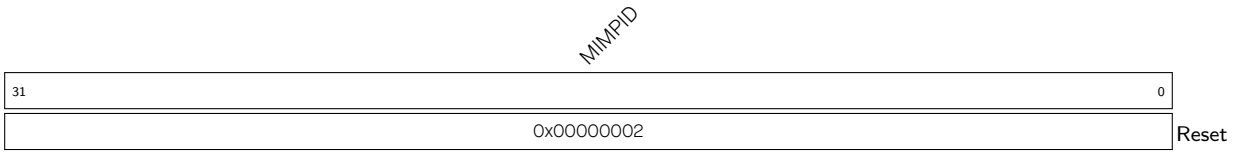
MVENDORID Represents Vendor ID. (RO)

Register 1.2. marchid (0xF12)



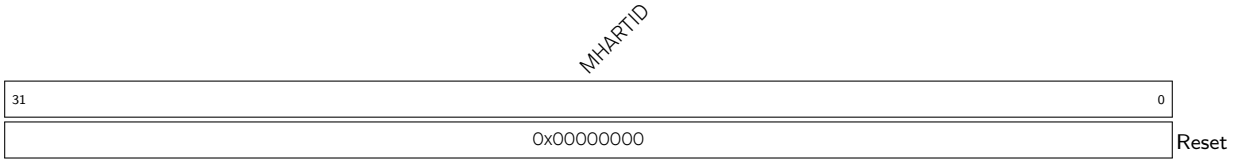
MARCHID Represents Architecture ID. (RO)

Register 1.3. mimpid (0xF13)



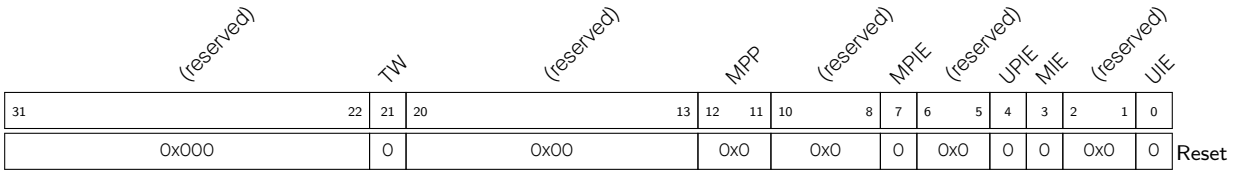
MIMPID Represents Implementation ID. (RO)

Register 1.4. mhartid (0xF14)



MHARTID Represents Hart ID. (RO)

Register 1.5. mstatus (0x300)



UIE Write 1 to enable the global user mode interrupt. (R/W)

MIE Write 1 to enable the global machine mode interrupt. (R/W)

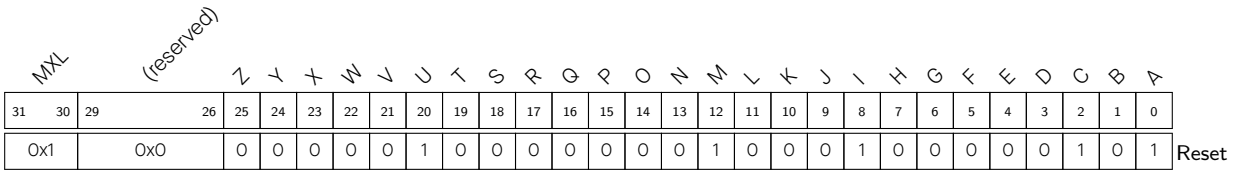
UPIE Write 1 to enable the user previous interrupt (before trap). (R/W)

MPIE Write 1 to enable the machine previous interrupt (before trap). (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)

TW Configures whether to cause illegal instruction exception when WFI (Wait-for-Interrupt) instruction is executed in U mode.
0: Executing WFI instruction will not cause illegal exception in U mode
1: Executing WFI instruction in U mode will cause illegal instruction exception
(R/W)

Register 1.6. misa (0x301)



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 = 0. (RO)

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

L Reserved = 0. (RO)

K Reserved = 0. (RO)

J Reserved = 0. (RO)

I RV32I base ISA = 1. (RO)

H Hypervisor extension = 0. (RO)

G Additional standard extensions present = 0. (RO)

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

E RV32E base ISA = 0. (RO)

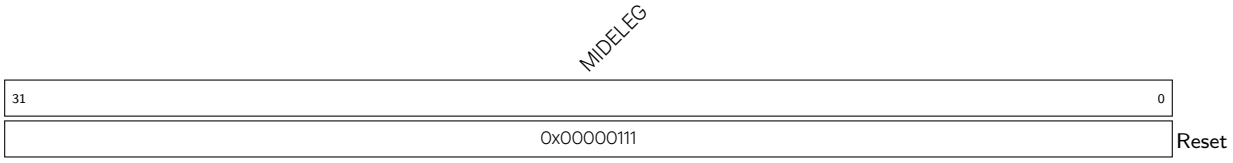
D Double-precision floating-point extension = 0. (RO)

C Compressed Extension = 1. (RO)

B Reserved = 0. (RO)

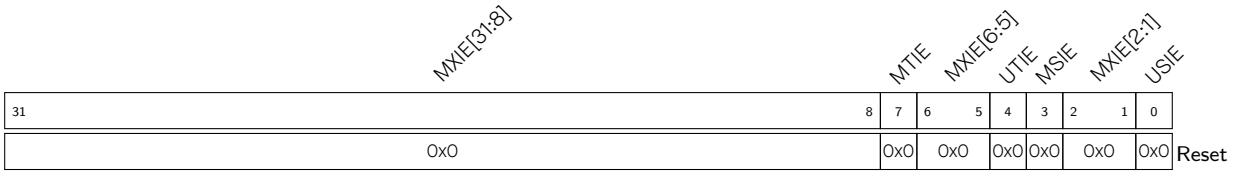
A Atomic Extension = 1. (RO)

Register 1.7. mideleg (0x303)



MIDELEG Configures the U mode delegation state for each interrupt ID. Below interrupts are delegated to U mode by default:
Bit 0: User software interrupt (CLINT)
Bit 4: User timer interrupt (CLINT)
Bit 8: User external interrupt
The default delegation can be modified at run-time if required.
(R/W)

Register 1.8. mie (0x304)



USIE Write 1 to enable the user software interrupt. (R/W)

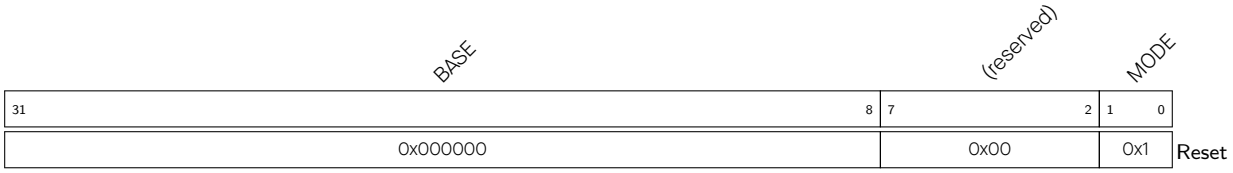
MSIE Write 1 to enable the machine software interrupt. (R/W)

UTIE Write 1 to enable the user timer interrupt. (R/W)

MTIE Write 1 to enable the machine timer interrupt. (R/W)

MXIE Write 1 to enable the 28 external interrupts. (R/W)

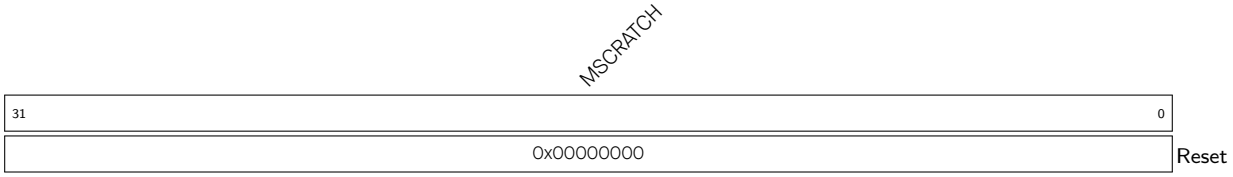
Register 1.9. mtvec (0x305)



MODE Represents whether machine mode interrupts are vectored. Only vectored mode **0x1** is available. (RO)

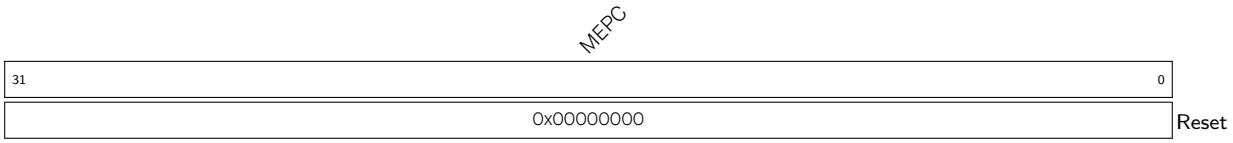
BASE Configures the higher 24 bits of trap vector base address aligned to 256 bytes. (R/W)

Register 1.10. mscratch (0x340)



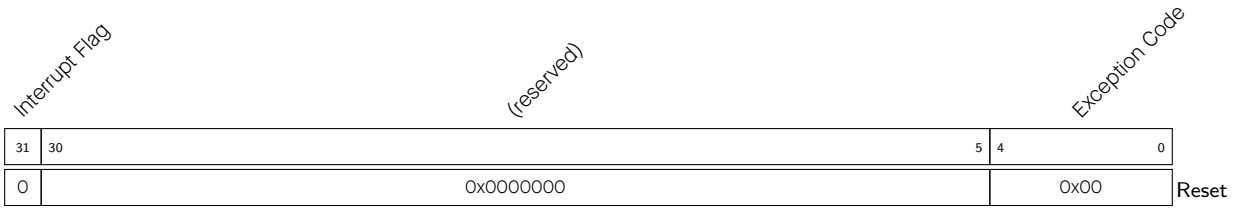
MSCRATCH Configures machine scratch information for custom use. (R/W)

Register 1.11. 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 1.12. mcause (0x342)



Exception Code This field is automatically updated with unique ID of the most recent exception or interrupt due to which CPU entered trap. Possible exception IDs are:

0x1: PMP instruction access fault

0x2: Illegal instruction

0x3: Hardware breakpoint/watchpoint or EBREAK

0x5: PMP load access fault

0x6: Misaligned store address or AMO address

0x7: PMP store access or AMO access fault

0x8: ECALL from U mode

0xb: ECALL from M mode

Other values: 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)

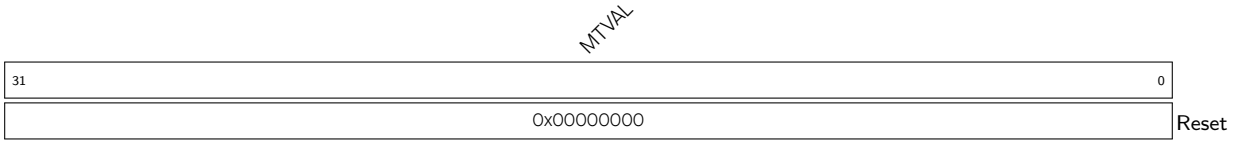
Interrupt Flag This flag is automatically updated when CPU enters trap.

If this is found to be set, indicates that the latest trap occurred due to an interrupt. For exceptions it remains unset.

Note: The interrupt controller is using up IDs in range 1-2, 5-6 and 8-31 for all external interrupt sources. This is different from the RISC-V standard which has reserved IDs in range 0-15 for core local interrupts only. Although local interrupt sources (CLINT) do use the reserved IDs 0, 3, 4 and 7.

(R/W)

Register 1.13. mtval (0x343)



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

Data is to be interpreted depending upon exception IDs:

0x1: Faulting virtual address of instruction

0x2: Faulting instruction opcode

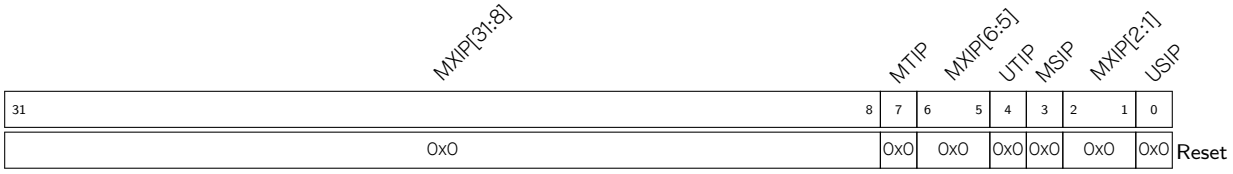
0x5: Faulting data address of load operation

0x7: Faulting data address of store operation

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

(R/W)

Register 1.14. mip (0x344)



USIP Configures the pending status of the user software interrupt.

0: Not pending

1: Pending

(R/W)

MSIP Configures the pending status of the machine software interrupt.

0: Not pending

1: Pending

(R/W)

UTIP Configures the pending status of the user timer interrupt.

0: Not pending

1: Pending

(R/W)

MTIP Configures the pending status of the machine timer interrupt.

0: Not pending

1: Pending

(R/W)

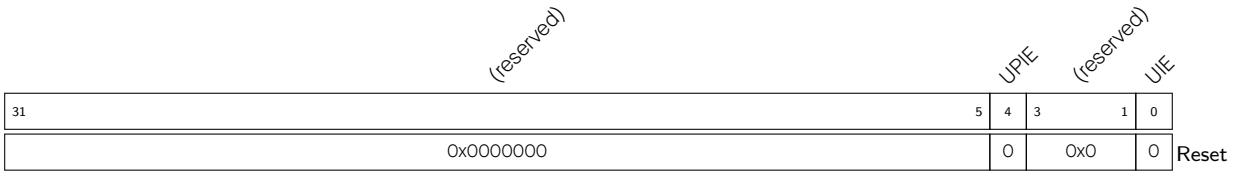
MXIP Configures the pending status of the 28 external interrupts.

0: Not pending

1: Pending

(R/W)

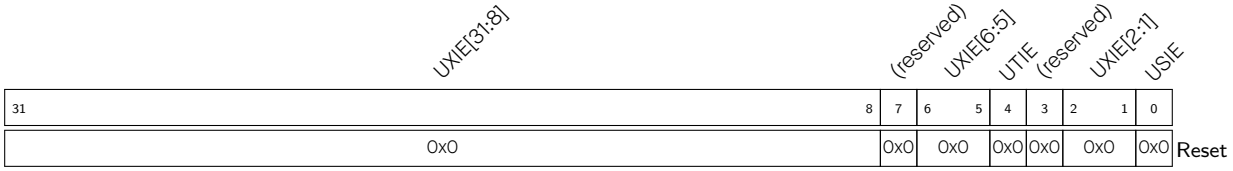
Register 1.15. ustatus (0x300)



UIE Write 1 to enable the global user mode interrupt. (R/W)

UPIE Write 1 to enable the user previous interrupt (before trap). (R/W)

Register 1.16. uie (0x004)

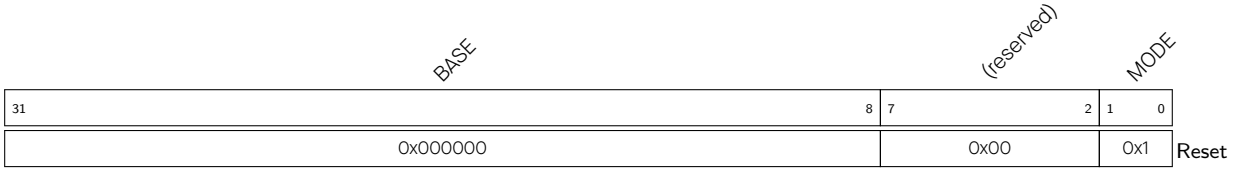


USIE Write 1 to enable the user software interrupt. (R/W)

UTIE Write 1 to enable the user timer interrupt. (R/W)

UXIE Write 1 to enable the 28 external interrupts delegated to U mode. (R/W)

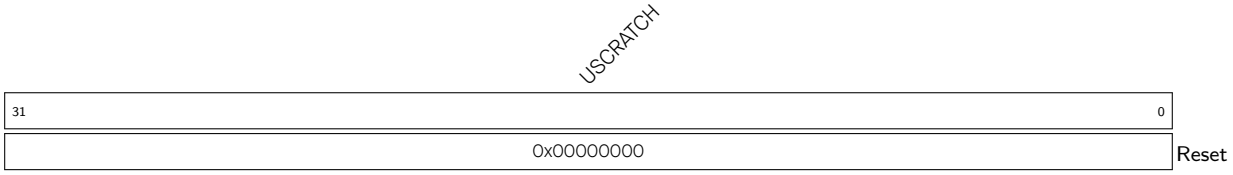
Register 1.17. utvec (0x005)



MODE Represents if user mode interrupts are vectored. Only vectored mode **0x1** is available. (RO)

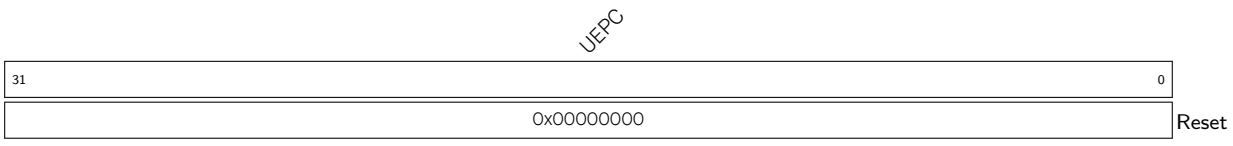
BASE Configures the higher 24 bits of trap vector base address aligned to 256 bytes. (R/W)

Register 1.18. uscratch (0x040)



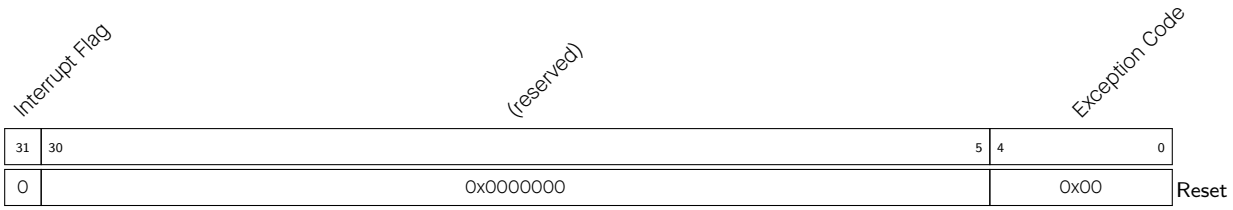
USCRATCH Configures user scratch information for custom use. (R/W)

Register 1.19. uepc (0x041)



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

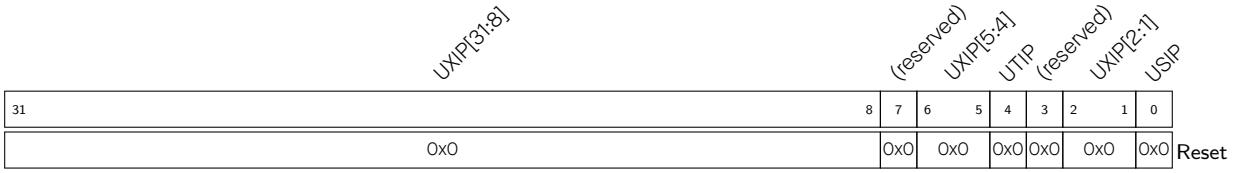
Register 1.20. ucause (0x042)



Interrupt ID This field is automatically updated with the unique ID of the most recent user mode interrupt due to which CPU entered trap. (R/W)

Interrupt Flag This flag would always be set because CPU can only enter trap due to user mode interrupts as exception delegation is unsupported. (R/W)

Register 1.21. uip (0x044)



USIP Configures the pending status of the user software interrupt.

0: Not pending

1: Pending

(R/W)

UTIP Configures the pending status of the user timer interrupt.

0: Not pending

1: Pending

(R/W)

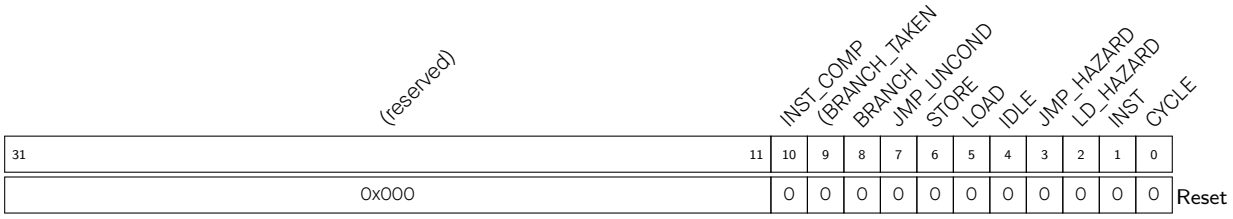
UXIP Configures the pending status of the 28 external interrupts delegated to user mode.

0: Not pending

1: Pending

(R/W)

Register 1.22. mpcer (0x7E0)



INST_COMP Count Compressed Instructions. (R/W)

BRANCH_TAKEN Count Branches Taken. (R/W)

BRANCH Count Branches. (R/W)

JMP_UNCOND Count Unconditional Jumps. (R/W)

STORE Count Stores. (R/W)

LOAD Count Loads. (R/W)

IDLE Count IDLE Cycles. (R/W)

JMP_HAZARD Count Jump Hazards. (R/W)

LD_HAZARD Count Load Hazards. (R/W)

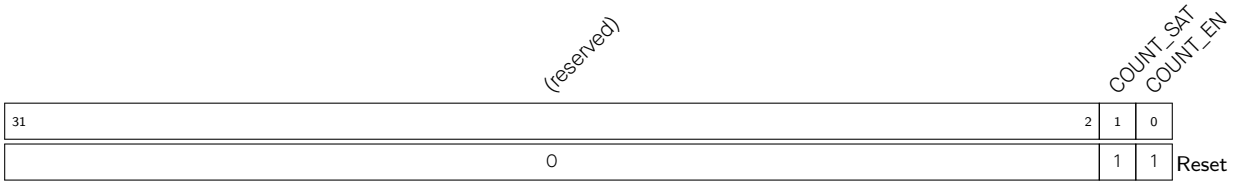
INST Count Instructions. (R/W)

CYCLE Count Clock Cycles. Cycle count does not increment during WFI mode.

Note: Each bit selects a specific event for counter to increment. If more than one event is selected and occurs simultaneously, then counter increments by one only.

(R/W)

Register 1.23. mpcmr (0x7E1)



COUNT_SAT Configures counter saturation.

0: Overflow on maximum value

1: Halt on maximum value

(R/W)

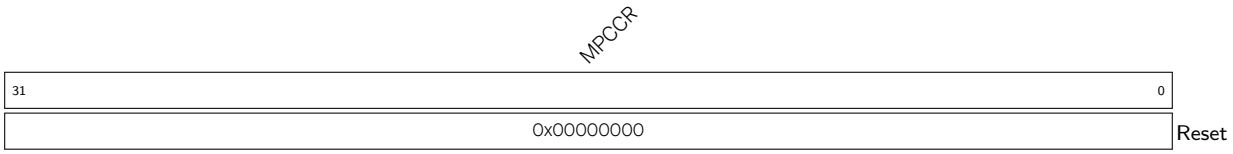
COUNT_EN Configures whether to enable the counter.

0: Disable

1: Enable

(R/W)

Register 1.24. mpccr (0x7E2)



MPCCR Represents the machine performance counter value. (R/W)

1.6 Interrupt Controller

1.6.1 Features

The interrupt controller allows capturing, masking and dynamic prioritization of interrupt sources routed from peripherals to the RISC-V CPU. It supports:

- Up to 28 external asynchronous interrupts and 4 core local interrupt sources (CLINT) with unique IDs (0-31)
- Configurable via read/write to memory mapped registers
- Delegable to user mode
- 15 levels of priority, programmable for each interrupt
- Support for both level and edge type interrupt sources
- Programmable global threshold for masking interrupts with lower priority
- Interrupts IDs mapped to trap-vector address offsets

For the complete list of interrupt registers and detailed configuration information, please refer to Chapter 9 *Interrupt Matrix (INTMTX)* > Section 9.6.2.

1.6.2 Functional Description

Each interrupt ID has 6 properties associated with it. These properties can be configured for the 28 external interrupts (1-2, 5-6, 8-31), but are static (except mode) for the 4 local CLINT interrupts (0, 3, 4, 7). These properties are as follows:

1. Mode (M/U):
   - Determines the mode in which an interrupt is to be serviced.
   - Programmed by setting or clearing the corresponding bit in mideleg CSR.
   - If the bit is cleared for an interrupt in mideleg CSR, then that interrupt will be captured in M mode.
   - If the bit is set for an interrupt in mideleg CSR, then it will be delegated to U mode.
2. Enable State (0-1):
   - Determines if an interrupt is enabled to be captured and serviced by the CPU.
   - Programmed by writing the corresponding bit in INTPRI_COREO_CPU_INT_ENABLE_REG.
   - Local CLINT interrupts have the corresponding bits reserved in the memory mapped registers thus they are always enabled at the INTC level.
   - An M mode interrupt (external or local) further needs to be unmasked at core level by setting the corresponding bit in mie CSR.
   - A U mode interrupt (external or local) further needs to be unmasked at core level by setting the corresponding bits in uie CSR.
3. Type (0-1):
   - Enables latching the state of an interrupt signal on its rising edge.

- Programmed by writing the corresponding bit in INTPRI_COREO_CPU_INT_TYPE_REG.
- An interrupt for which type is kept 0 is referred to as a 'level' type interrupt.
- An interrupt for which type is set to 1 is referred to as an 'edge' type interrupt.
- Local CLINT interrupts are always 'level' type and thus have the corresponding bits reserved in the above register.

4. Priority (0-15):
   - Determines which interrupt, among multiple pending interrupts, the CPU will service first.
   - Programmed by writing to the INTPRI_COREO_CPU_INT_PRI_*n*_REG for an external interrupt with particular interrupt ID *n*.
   - Enabled external interrupts with priorities less than the threshold value in INTPRI_COREO_CPU_INT_THRESH_REG are masked.
   - Priority levels increase from 0 (lowest) to 15 (highest).
   - Interrupts with the same priority are statically prioritized by their IDs, lowest ID having the highest priority.
   - Local CLINT interrupts have static priorities associated with them, and thus have the corresponding priority registers to be reserved.
   - Local CLINT interrupts cannot be masked using the threshold values for either mode.
5. Pending State (0-1):
   - Reflects the captured state of an enabled and unmasked external interrupt signal.
   - For each external interrupt ID the corresponding bit in read-only INTPRI_COREO_CPU_INT_EIP_STATUS_REG gives its pending state.
   - For each interrupt ID (local or external), the corresponding bit in the mip CSR for M mode interrupts or uip CSR for U mode interrupts, also gives its pending state.
   - A pending interrupt will cause CPU to enter trap if no other pending interrupt has higher priority.
   - A pending interrupt is said to be 'claimed' if it preempts the CPU and causes it to jump to the corresponding trap vector address.
   - All pending interrupts which are yet to be serviced are termed as 'unclaimed'.
6. Clear State (0-1):
   - Toggling this will clear the pending state of claimed edge-type interrupts only.
   - Toggled by first setting and then clearing the corresponding bit in INTPRI_COREO_CPU_INT_CLEAR_REG.
   - Pending state of a level type interrupt is unaffected by this and must be cleared from source.
   - Pending state of an unclaimed edge type interrupt can be flushed, if required, by first clearing the corresponding bit in INTPRI_COREO_CPU_INT_ENABLE_REG and then toggling same bit in INTPRI_COREO_CPU_INT_CLEAR_REG.

For a detailed description of the core local interrupt sources, please refer to Section 1.7.

When CPU services a pending M/U mode interrupt, it:

- saves the address of the current un-executed instruction in mepc/uepc for resuming execution later.
- updates the value of mcause/ucause with the ID of the interrupt being serviced.
- copies the state of MIE/UIE into MPIE/UPIE, and subsequently clears MIE/UIE, thereby disabling interrupts globally.
- enters trap by jumping to a word-aligned offset of the address stored in mtvec/utvec.

The word aligned trap address for an M mode interrupt with a certain $ID = i$ can be calculated as $(mtvec + 4i)$. Similarly, the word aligned trap address for a U mode interrupt can be calculated as $(utvec + 4i)$.

After jumping to the trap vector for the corresponding mode, the execution flow is dependent on software implementation, although it can be presumed that the interrupt will get handled (and cleared) in some interrupt service routine (ISR) and later the normal execution will resume once the CPU encounters MRET/URET instruction for that mode.

Upon execution of MRET/URET instruction, the CPU:

- copies the state of MPIE/UPIE back into MIE/UIE, and subsequently clears MPIE/UPIE. This means that if previously MPIE/UPIE was set, then, after MRET/URET, MIE/UIE will be set, thereby enabling interrupts globally.
- jumps to the address stored in mepc/uepc and resumes execution.

It is possible to perform software assisted nesting of interrupts inside an ISR as explained in Section 1.6.3.

The below listed points outline the functional behavior of the controller:

- Only if an interrupt has priority higher or equal to the value in the threshold register, will it be reflected in INTPRI_COREO_CPU_INT_EIP_STATUS_REG.
- If an interrupt is visible in INTPRI_COREO_CPU_INT_EIP_STATUS_REG and has yet to be serviced, then it's possible to mask it (and thereby prevent the CPU from servicing it) by either lowering the value of its priority or increasing the global threshold.
- If an interrupt, visible in INTPRI_COREO_CPU_INT_EIP_STATUS_REG, is to be flushed (and prevented from being serviced at all), then it must be disabled (and cleared if it is of edge type).

1.6.3 Suggested Operation

1.6.3.1 Latency Aspects

There is latency involved while configuring the Interrupt Controller.

In steady state operation, the Interrupt Controller has a fixed latency of 4 cycles. Steady state means that no changes have been made to the Interrupt Controller registers recently. This implies that any interrupt that is asserted to the controller will take exactly 4 cycles before the CPU starts processing the interrupt. This further implies that CPU may execute up to 5 instructions before the preemption happens.

Whenever any of its registers are modified, the Interrupt Controller enters into transient state, which may take up to 4 cycles for it to settle down into steady state again. During this transient state, the ordering of interrupts

may not be predictable, and therefore, a few safety measures need to be taken in software to avoid any synchronization issues.

Also, it must be noted that the Interrupt Controller configuration registers lie in the APB address range, hence any R/W access to these registers may take multiple cycles to complete.

In consideration of above mentioned characteristics, users are advised to follow the sequence described below, whenever modifying any of the Interrupt Controller registers:

1. save the state of MIE and clear MIE to 0
2. read-modify-write one or more Interrupt Controller registers
3. execute FENCE instruction to wait for any pending write operations to complete
4. finally, restore the state of MIE

Due to its critical nature, it is recommended to disable interrupts globally (MIE=0) beforehand, whenever configuring interrupt controller registers, and then restore MIE right after, as shown in the sequence above.

After execution of the sequence above, the Interrupt Controller will resume operation in steady state.

1.6.3.2 Configuration Procedure

By default, interrupts are disabled globally, since the reset value of MIE bit in mstatus is 0. Software must set MIE=1 after initialization of the interrupt stack (including setting mtvec to the interrupt vector address) is done.

The threshold value for external interrupts in INTPRI_COREO_CPU_INT_THRESH_REG is 0 by default. For priority based masking of interrupts this could be initialized to 1 after CPU comes out of reset. That way all interrupt sources which have default 0 priority are masked.

During normal execution, if an external interrupt *n* is to be enabled, the below sequence may be followed:

1. save the state of MIE and clear MIE to 0
2. depending upon the type of the interrupt (edge/level), set/unset the *n*th bit of INTPRI_COREO_CPU_INT_TYPE_REG
3. set the priority by writing a value to INTPRI_COREO_CPU_INT_PRI_*n*_REG in range 1 (lowest) to 15 (highest)
4. set the *n*th bit of INTPRI_COREO_CPU_INT_ENABLE_REG
5. execute FENCE instruction
6. restore the state of MIE

When one or more interrupts become pending, the CPU acknowledges (claims) the interrupt with the highest priority and jumps to the trap vector address corresponding to the interrupt's ID. Software implementation may read mcause to infer the type of trap (mcause (31) is 1 for interrupts and 0 for exceptions) and then the ID of the interrupt (mcause (4-0) gives ID of interrupt or exception). This inference may not be necessary if each entry in the trap vector is a jump instruction to different trap handlers. Ultimately, the trap handler(s) will redirect execution to the appropriate ISR for this interrupt.

Upon entering into an ISR, software must toggle the *n*th bit of INTPRI_COREO_CPU_INT_CLEAR_REG if the interrupt is of edge type, or clear the source of the interrupt if it is of level type.

Software may also update the value of INTPRI_COREO_CPU_INT_THRESH_REG and program MIE=1 for allowing higher priority interrupts to preempt the current ISR (nesting), however, before doing so, all the state CSRs must be saved (mepc, mstatus, mcause, etc.) since they will get overwritten due to occurrence of such an interrupt. Later, when exiting the ISR, the values of these CSRs must be restored.

Finally, after the execution returns from the ISR back to the trap handler, MRET instruction is used to resume normal execution.

Later, if the *n* interrupt is no longer needed and needs to be disabled, the following sequence may be followed:

1. save the state of MIE and clear MIE to 0
2. check if the interrupt is pending in INTPRI_COREO_CPU_INT_EIP_STATUS_REG
3. set/unset the *n*th bit of INTPRI_COREO_CPU_INT_ENABLE_REG
4. if the interrupt is of edge type and was found to be pending in step 2 above, *n*th bit of INTPRI_COREO_CPU_INT_CLEAR_REG must be toggled, so that its pending status gets flushed
5. execute FENCE instruction
6. restore the state of MIE

Above is only a suggested scheme of operation. Actual software implementation may vary.

1.6.4 Registers

For the complete list of interrupt registers and configuration information, please refer to Section 9.6.2 and Section 9.7.2 respectively.

1.7 Core Local Interrupts (CLINT)

1.7.1 Overview

The CPU supports 4 local level-type interrupt sources with static priorities as shown below.

Table 1.7-1. Core Local Interrupt (CLINT) Sources

ID	Description	Priority
0	U mode software interrupt	1
3	M mode software interrupt	3
4	U mode timer interrupt	0
7	M mode timer interrupt	2

These interrupt sources have reserved IDs and fixed priorities which cannot be masked via the interrupt controller threshold registers for either mode.

Two of these interrupts (0 and 4) are by-default delegated to U mode as per the reset values of corresponding bits in mideleg CSR.

It must be noted that regardless of the fixed priority of CLINT interrupts, pending external interrupt sources always have higher priority over CLINT sources.

1.7.2 Features

- 4 local level-type interrupt sources with static priorities and IDs
- Memory mapped configuration and status registers
- Support for interrupts in both M and U modes
- 64-bit timer with interrupt with overflow flag
- Software interrupts

1.7.3 Software Interrupt

M and U mode software interrupt sources are controlled by setting or clearing the memory mapped registers MSIP and USIP, respectively.

The MSIE/USIE bit must be set in mie/uie CSR for enabling the interrupt at core level for a particular mode.

Pending state of this interrupt can be checked for either mode by reading the corresponding bit MSIP/USIP in mip/uip CSR.

Note that by default U mode software interrupt with ID 0 has the corresponding bit set in mideleg CSR. This bit can be toggled for using the interrupt in M mode instead. Similarly the bit corresponding to M mode software interrupt can be set for using it in U mode.

1.7.4 Timer Counter and Interrupt

The CPU provides a local memory-mapped 64-bit wide M mode timer counter register MTIME which has both read/write access. The timer counter can be enabled by setting the MTCE bit in MTIMECTL.

A read-only memory mapped UTIME is also provided for reading the timer counter from U mode, although it always reflects the same value as in the corresponding M mode counter MTIME register.

Timer interrupt for M/U mode is enabled by setting the MTIE/UTIE bit in MTIMECTL/UTIMECTL. Also, the MTIE/UTIE bit must be set in mie CSR for enabling the interrupt at core level for a particular mode.

Interrupt for M/U mode is asserted when the 64-bit timer value exceeds the 64-bit timer-compare value programmed in MTIMECMP/UTIMECMP.

Pending state of M/U mode timer interrupt is reflected as the read-only MTIP/UTIP bit in MTIMECTL/UTIMECTL.

For de-asserting the pending timer interrupt in M/U mode, either the MTIE/UTIE bit has to be cleared or the value of the MTIMECMP/UTIMECMP register needs to be updated.

Pending state of this interrupt can be checked at core level for either mode by reading the corresponding bit MTIP/UTIP in mip/uip.

Upon overflow of the 64-bit timer counter, the MTOF/UTOF bit in MTIMECTL/UTIMECTL gets set. It can be cleared after appropriate handling of the overflow situation.

Note that by default U mode timer interrupt with ID 4 has the corresponding bit set in mideleg CSR. This bit can be toggled for using the interrupt in M mode instead. Similarly the bit corresponding to M mode timer interrupt can be set for using it in U mode.

1.7.5 Register Summary

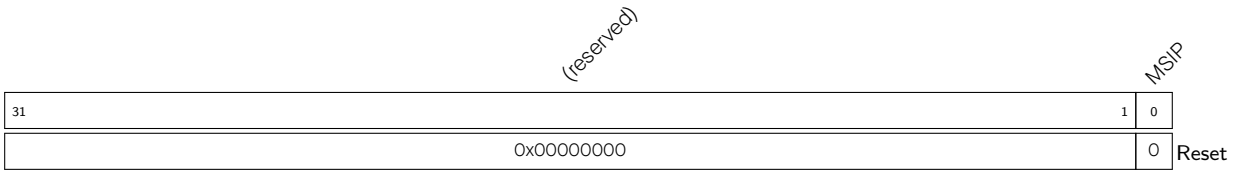
The addresses in this section are relative to CPU sub-system base address provided in Figure 4.2-1 in Chapter 4 *System and Memory*.

Name	Description	Address	Access
MSIP	Core local machine software interrupt pending register	0x1800	R/W
MTIMECTL	Core local machine timer interrupt control/status register	0x1804	R/W
MTIME	64-bit core local timer counter value	0x1808	R/W
MTIMECMP	64-bit core local machine timer compare value	0x1810	R/W
USIP	Core local user software interrupt pending register	0x1C00	R/W
UTIMECTL	Core local user timer interrupt control/status register	0x1C04	R/W
UTIME	Read-only 64-bit core local timer counter value	0x1C08	RO
UTIMECMP	64-bit core local user timer compare value	0x1C10	R/W

1.7.6 Register Description

The addresses in this section are relative to CPU subsystem base address provided in Figure 4.2-1 in Chapter 4 *System and Memory*.

Register 1.25. MSIP (0x1800)



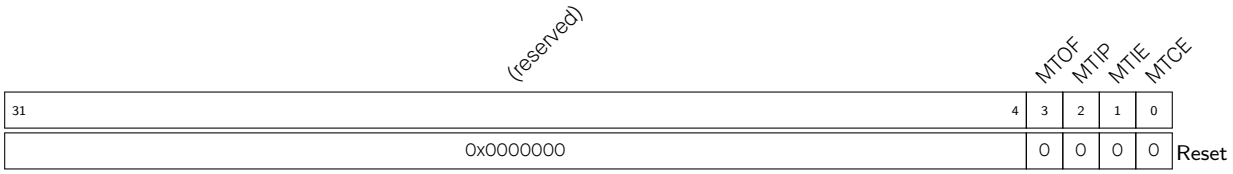
MSIP Configures the pending status of the machine software interrupt.

0: Not pending

1: Pending

(R/W)

Register 1.26. MTIMECTL (0x1804)



MTCE Configures whether to enable the CLINT timer counter.

0: Not enable

1: Enable

(R/W)

MTIE Write 1 to enable the machine timer interrupt. (R/W)

MTIP Represents the pending status of the machine timer interrupt.

0: Not pending

1: Pending

(RO)

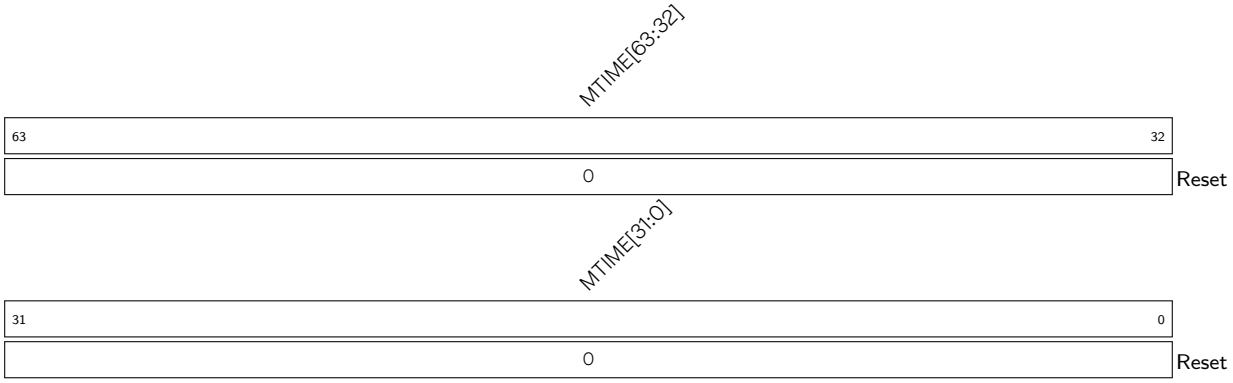
MTOF Configures whether the machine timer overflows.

0: Not overflow

1: Overflow

(R/W)

Register 1.27. MTIME (0x1808)

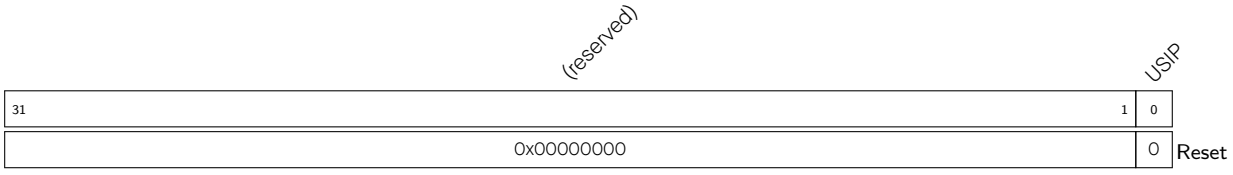


MTIME Configures the 64-bit CLINT timer counter value. (R/W)

Register 1.28. MTIMECMP (0x1810)

MTIMECMP Configures the 64-bit machine timer compare value. (R/W)

Register 1.29. USIP (0x1C00)



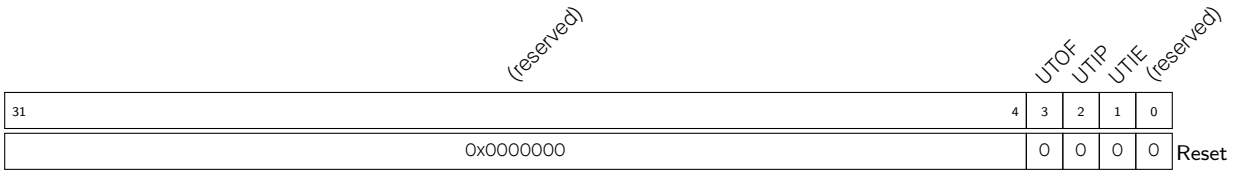
USIP Configures the pending status of the user software interrupt.

0: Not pending

1: Pending

(R/W)

Register 1.30. UTIMECTL (0x1C04)



UTIE Write 1 to enable the user timer interrupt. (R/W)

UTIP Represents the pending status of the user timer interrupt. (RO)

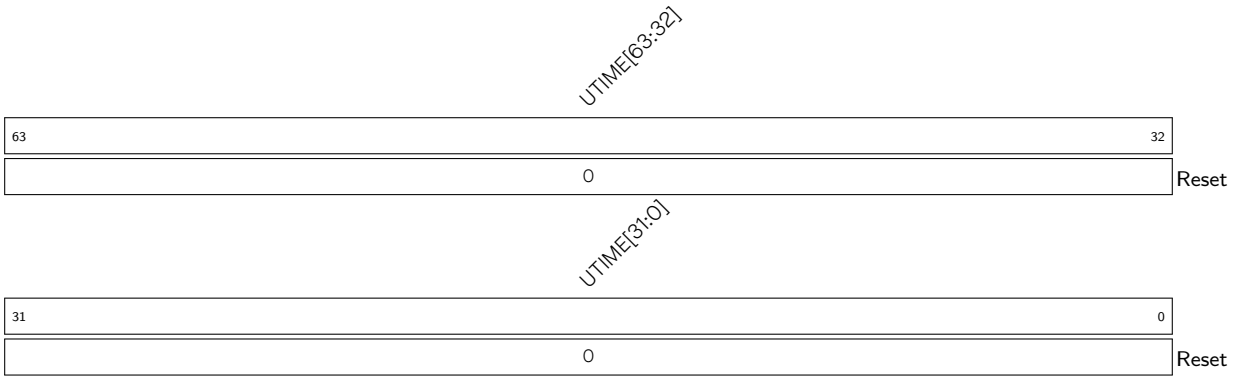
UTOF Configures whether the user timer overflows.

0: Not overflow

1: Overflow

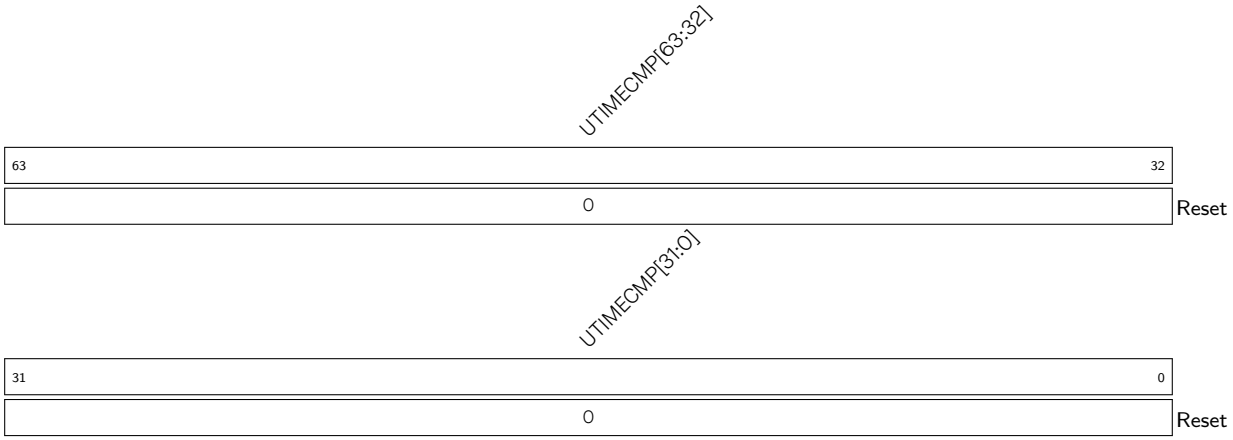
(R/W)

Register 1.31. UTIME (0x1C08)



UTIME Represents the read-only 64-bit CLINT timer counter value. (RO)

Register 1.32. UTIMECMP (0x1C10)



UTIMECMP Configures the 64-bit user timer compare value. (R/W)

1.8 Physical Memory Protection

1.8.1 Overview

The CPU core includes a Physical Memory Protection (PMP) unit fully compliant to **RISC-V Instruction Set Manual, Volume II: Privileged Architecture, Version 1.10**, which 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, CPU core also implements custom Physical Memory Attributes (PMA) checkers to provide additional permission checks based on pre-defined attributes.

1.8.2 Features

The PMP unit can be used to restrict access to physical memory. It supports 16 regions and a minimum granularity of 4 bytes. The maximum supported NAPOT range is 4 GB.

1.8.3 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 enabled 16 pmpcfgX and pmpaddrX registers (refer to the Register Summary).

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 the Register Summary) 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, the software can set the lock bit of the 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 the processor level and the 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.

1.8.4 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.	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
pmpaddr0	Physical memory protection address register.	0x3B0	R/W
pmpaddr1	Physical memory protection address register.	0x3B1	R/W
pmpaddr2	Physical memory protection address register.	0x3B2	R/W
pmpaddr3	Physical memory protection address register.	0x3B3	R/W
pmpaddr4	Physical memory protection address register.	0x3B4	R/W
pmpaddr5	Physical memory protection address register.	0x3B5	R/W
pmpaddr6	Physical memory protection address register.	0x3B6	R/W
pmpaddr7	Physical memory protection address register.	0x3B7	R/W
pmpaddr8	Physical memory protection address register.	0x3B8	R/W
pmpaddr9	Physical memory protection address register.	0x3B9	R/W
pmpaddr10	Physical memory protection address register.	0x3BA	R/W
pmpaddr11	Physical memory protection address register.	0x3BB	R/W
pmpaddr12	Physical memory protection address register.	0x3BC	R/W
pmpaddr13	Physical memory protection address register.	0x3BD	R/W
pmpaddr14	Physical memory protection address register.	0x3BE	R/W
pmpaddr15	Physical memory protection address register.	0x3BF	R/W

1.8.5 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**.

1.9 Physical Memory Attribute Checker (PMAC)

1.9.1 Overview

CPU core also implements a custom Physical Memory Attributes Checker (PMAC) to provide additional permission checks based on pre-defined memory type configured through custom CSRs.

1.9.2 Features

PMAC supports below features:

- Configurable memory type for defined memory regions
- Configurable attribute for defined memory regions

1.9.3 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 the Register Summary). 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 a similar way to PMP checks. When any instruction is being fetched from a memory region configured as null or 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 null or invalid memory region, 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. 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 the 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 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.

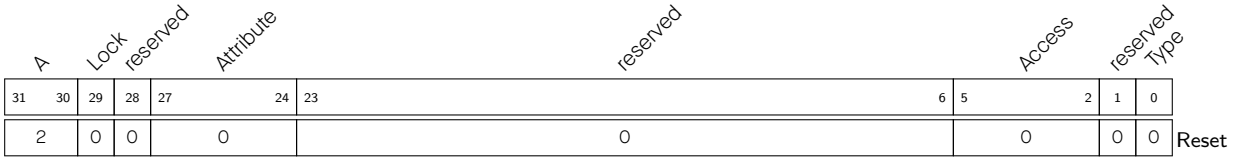
1.9.4 Register Summary

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

Name	Description	Address	Access
pma_cfg0	Physical Memory Attribute configuration	0xBC0	R/W
pma_cfg1	Physical Memory Attribute configuration	0xBC1	R/W
pma_cfg2	Physical Memory Attribute configuration	0xBC2	R/W
pma_cfg3	Physical Memory Attribute configuration	0xBC3	R/W
pma_cfg4	Physical Memory Attribute configuration	0xBC4	R/W
pma_cfg5	Physical Memory Attribute configuration	0xBC5	R/W
pma_cfg6	Physical Memory Attribute configuration	0xBC6	R/W
pma_cfg7	Physical Memory Attribute configuration	0xBC7	R/W
pma_cfg8	Physical Memory Attribute configuration	0xBC8	R/W
pma_cfg9	Physical Memory Attribute configuration	0xBC9	R/W
pma_cfg10	Physical Memory Attribute configuration	0xBCA	R/W
pma_cfg11	Physical Memory Attribute configuration	0xBCB	R/W
pma_cfg12	Physical Memory Attribute configuration	0xBCC	R/W
pma_cfg13	Physical Memory Attribute configuration	0xBCD	R/W
pma_cfg14	Physical Memory Attribute configuration	0xBCE	R/W
pma_cfg15	Physical Memory Attribute configuration	0xBCF	R/W
pma_addr0	Physical Memory Attribute address register	0xBD0	R/W
pma_addr1	Physical Memory Attribute address register	0xBD1	R/W
pma_addr2	Physical Memory Attribute address register	0xBD2	R/W
pma_addr3	Physical Memory Attribute address register	0xBD3	R/W
pma_addr4	Physical Memory Attribute address register	0xBD4	R/W
pma_addr5	Physical Memory Attribute address register	0xBD5	R/W
pma_addr6	Physical Memory Attribute address register	0xBD6	R/W
pma_addr7	Physical Memory Attribute address register	0xBD7	R/W
pma_addr8	Physical Memory Attribute address register	0xBD8	R/W
pma_addr9	Physical Memory Attribute address register	0xBD9	R/W
pma_addr10	Physical Memory Attribute address register	0xBDA	R/W
pma_addr11	Physical Memory Attribute address register	0xBDB	R/W
pma_addr12	Physical Memory Attribute address register	0xBDC	R/W
pma_addr13	Physical Memory Attribute address register	0xBDD	R/W
pma_addr14	Physical Memory Attribute address register	0xBDE	R/W
pma_addr15	Physical Memory Attribute address register	0xBDF	R/W

1.9.5 Register Description

Register 1.33. pma_cfgX (0xBC0-0xBCF)



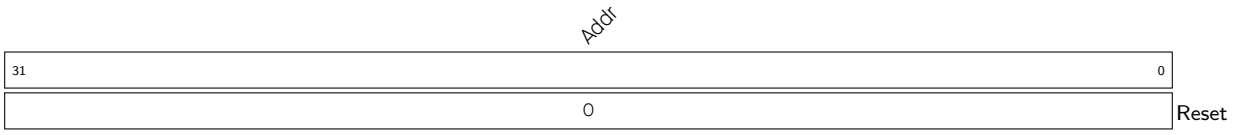
A Configures address type. The functionality is the same as pmpcfg register's A field.
0x0: OFF
0x1: TOR
0x2: NA4
0x3: NAPOT
(R/W)

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. (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 1.34. pma_addrX (0xBD0-0xBDF)



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

1.10 Debug

1.10.1 Overview

This section describes how to debug and test software running on ESP-RISC-V core. Debug support is provided through standard JTAG pins and complies to RISC-V External Debug Support Specification version 0.13.

Figure 1.10-1 below shows the main components of External Debug Support.

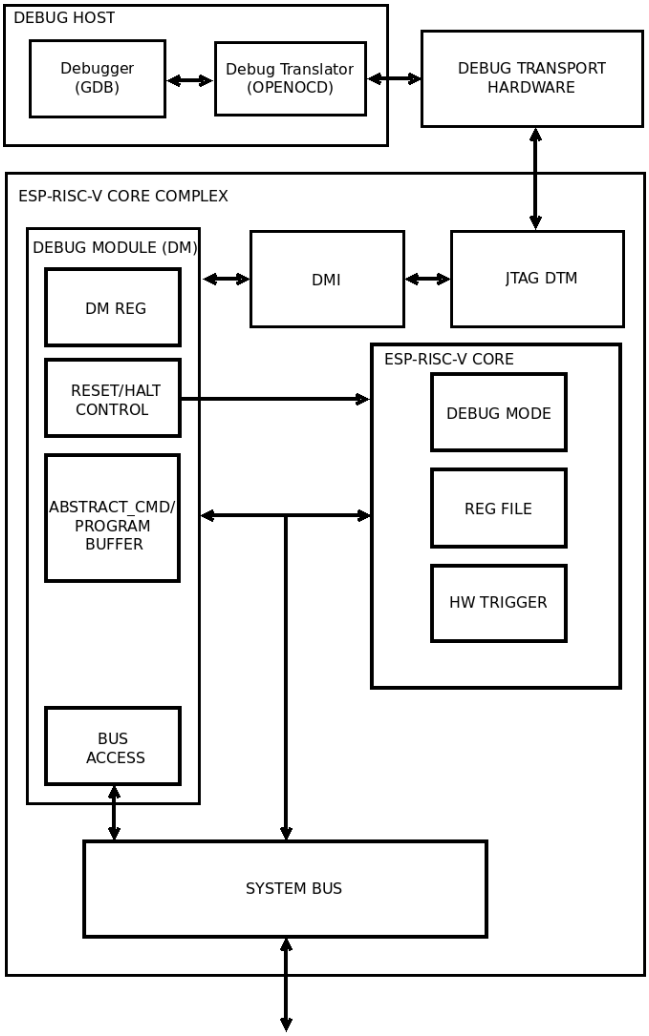


Figure 1.10-1. Debug System Overview

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 ESP-RISC-V Core's Debug Transport Module (DTM) through a standard JTAG interface. The DTM provides access to the Debug Module (DM) using the Debug Module Interface (DMI).

The DM allows the debugger to halt selected cores. Abstract commands provide access to GPRs (general purpose registers). The Program Buffer allows the debugger to execute arbitrary code on the core, which allows access to additional CPU core state. Alternatively, additional abstract commands can provide access to additional CPU core state. ESP-RISC-V core contains Trigger Module supporting 4 triggers. When trigger

conditions are met, core will halt spontaneously and inform the debug module that they have halted.

System bus access block allows memory and peripheral register access without using the core.

1.10.2 Features

Basic debug functionality supports below features:

- Provides necessary information about the implementation to the debugger.
- Allows the CPU core to be halted and resumed.
- CPU core registers (including CSRs) can be read/written by debugger.
- CPU can be debugged from the first instruction executed after reset.
- CPU core can be reset through debugger.
- CPU can be halted on software breakpoint instructions.
- Hardware single-stepping.
- Execute arbitrary instructions in the halted CPU by means of the program buffer. 16-word program buffer is supported.
- System bus access is supported through word aligned address access.
- Supports four Hardware Triggers (can be used as breakpoints/watchpoints) as described in Section 1.11.

1.10.3 Functional Description

As mentioned earlier, Debug Scheme conforms to RISC-V External Debug Support Specification version 0.13. Please refer to the specification for functional operation details.

1.10.4 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.

Temporary disable JTAG 3,4	EFUSE_DIS_USB_JTAG 4	EFUSE_DIS_USB_SERIAL_JTAG 4	EFUSE_DIS_PAD_JTAG 4	EFUSE_JTAG_SEL_ENABLE 4	Strapping Pin GPIO25 5	USB_to_JTAG Status	PAD_to_JTAG Status
0	0	0	0	0	x 2	Available 1	Unavailable 1
0	0	0	0	1	1	Available	Unavailable
0	0	0	0	1	0	Unavailable	Available
0	0	1	0	x	x	Unavailable	Available
0	1	0	0	x	x	Unavailable	Available

Temporary disable JTAG 3	EFUSE_DIS_USB_JTAG 4	EFUSE_DIS_USB_SERIAL_JTAG 4	EFUSE_DIS_PAD_JTAG 4	EFUSE_JTAG_SEL_ENABLE 4	Strapping Pin GPIO25 5	USB_to_JTAG Status	PAD_to_JTAG Status
0	1	1	0	x	x	Unavailable	Available
0	0	0	1	x	x	Available	Unavailable
0	0	1	1	x	x	Unavailable	Unavailable
0	1	0	1	x	x	Unavailable	Unavailable
0	1	1	1	x	x	Unavailable	Unavailable
1	x	x	x	x	x	Unavailable	Unavailable

Note:

1. Available: the corresponding JTAG function is available.
   Unavailable: the corresponding JTAG function is not available.
2. x: do not care.
3. "Temporary disable JTAG" means that if there are an even number of bits "1" in EFUSE_SOFT_DIS_JTAG[2:0], the JTAG function is turned on (the corresponding value in the table is 1), otherwise it is turned off (the corresponding value in the table is 0). However, under certain special conditions of the HMAC Accelerator in ESP32-H2, the JTAG function may be turned on even if there is an odd number of bits "1" in EFUSE_SOFT_DIS_JTAG[2:0]. For information on how HMAC affects JTAG functionality, please refer to Chapter HMAC Accelerator.
4. Please refer to Chapter eFuse Controller to get more information about eFuse.
5. Please refer to Chip Boot Control to get more information about the strapping pin GPIO25.

1.10.5 Register Summary

Below is the list of Debug CSRs supported by ESP-RISC-V CPU core:

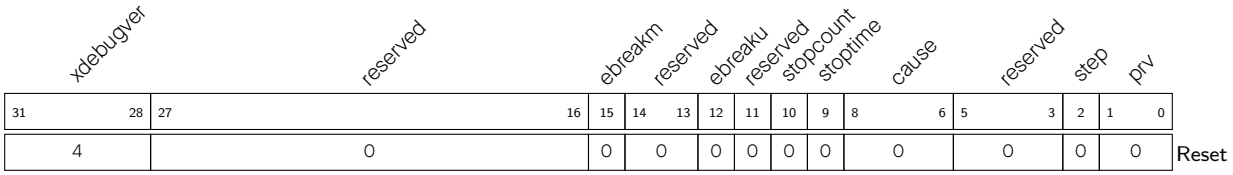
Name	Description	Address	Access
dcsr	Debug Control and Status	0x7B0	R/W
dpc	Debug PC	0x7B1	R/W
dscratch0	Debug Scratch Register 0	0x7B2	R/W
dscratch1	Debug Scratch Register 1	0x7B3	R/W

All the debug module registers are implemented in conformance to the specification RISC-V External Debug Support Version 0.13. Please refer to it for more details.

1.10.6 Register Description

Below are the details of Debug CSRs supported by ESP-RISC-V core:

Register 1.35. 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)

stopcount This feature is not implemented. Debugger will always read this bit as 0. (RO)

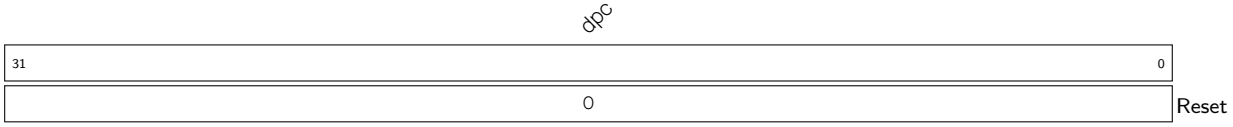
stoptime This feature is not implemented. Debugger will always read this bit as 0. (RO)

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)
3: haltreq was set. (priority 2)
4: The CPU core single stepped because step was set. (priority 1)
Other values are reserved for future use.
(RO)

step When set and not in Debug Mode, the core will only execute a single instruction and then enter Debug Mode. Interrupts are **enabled*** when this bit is set. 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. (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 1.36. dpc (0x7B1)

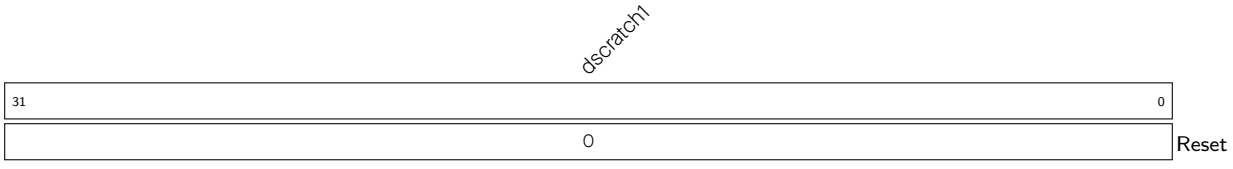


dpc Upon entry to debug mode, dpc is written with the virtual address of the instruction that encountered the exception. When resuming, the CPU core's PC is updated to the virtual address stored in dpc. A debugger may write dpc to change where the CPU resumes. (R/W)

Register 1.37. dscratch0 (0x7B2)

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

Register 1.38. dscratch1 (0x7B3)



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

1.11 Hardware Trigger

1.11.1 Features

Hardware Trigger module provides breakpoint and watchpoint capability for debugging. It includes the following features:

- 4 independent trigger units
- each unit can be configured for matching the address of the program counter or load-store accesses
- can preempt execution by causing breakpoint exception
- can halt execution and transfer control to debugger
- support NAPOT (naturally aligned power-of-two regions) address encoding

1.11.2 Functional Description

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

To choose a particular trigger unit write the index (0-3) 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, 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 1.11-1) 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.

Table 1.11-1. 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
	....	
a01...1111111111	a00...0000000000	2^{31}

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.

1.11.3 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.

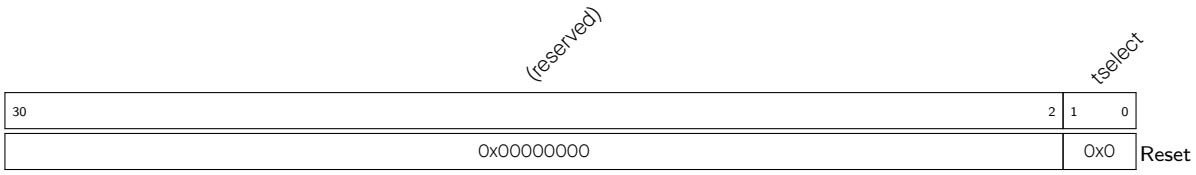
1.11.4 Register Summary

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
tcontrol	Global Trigger Control	0x7A5	R/W

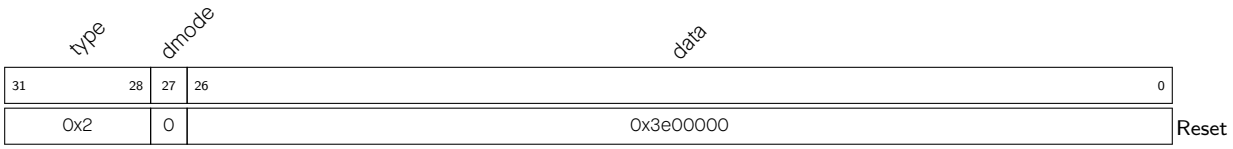
1.11.5 Register Description

Register 1.39. tselect (0x7A0)



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

Register 1.40. tdata1 (0x7A1)

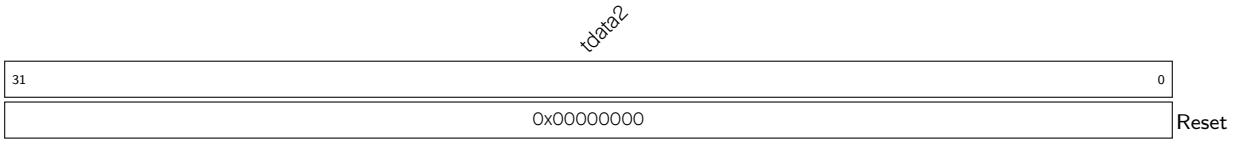


type Represents the trigger type. This field is reserved since only match type (0x2) triggers are supported. (RO)

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)

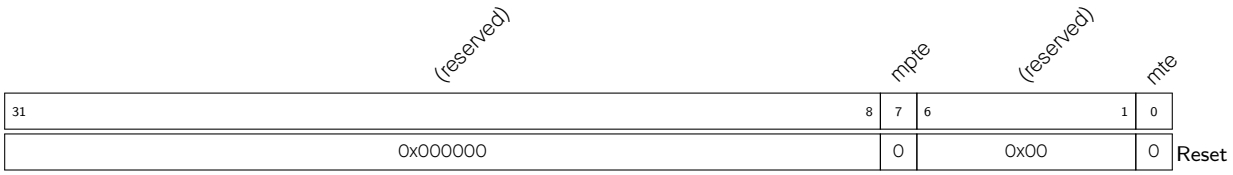
data Configures the abstract tdata1 content. This will always be interpreted as fields of mcontrol since only match type (0x2) triggers are supported. (R/W)

Register 1.41. 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)

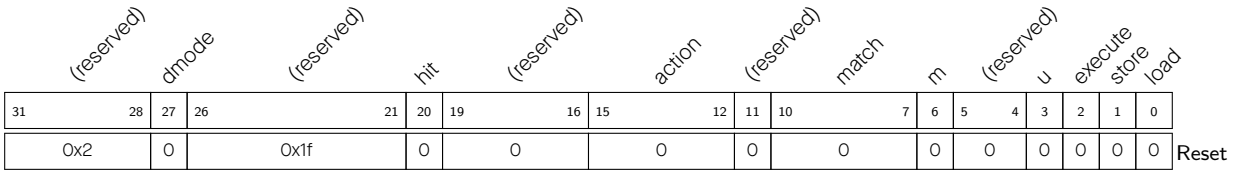
Register 1.42. tcontrol (0x7A5)



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 1.43. mcontrol (0x7A1)



dmode Same as dmode in tdata1. (RW *)

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

m Set this for enabling selected trigger to operate in machine mode. (R/W)

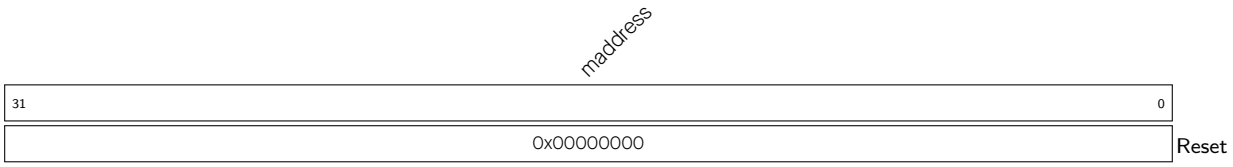
u Set this for enabling selected trigger to operate in user mode. (R/W)

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

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

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

Register 1.44. maddress (0x7A2)



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

1.12 Trace

1.12.1 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.

1.12.2 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.

1.12.3 Functional Description

ESP-RISC-V CPU 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 the 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 *2 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 *2 RISC-V Trace Encoder (TRACE)* block.

1.13 Dedicated IO

1.13.1 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, and in general are slower because generally the APB bus runs at a lower speed than the CPU. As an alternative, the CPU core implements I/O processors specific CPU registers (CSRs) which are directly connected to the GPIO matrix or IO pads. As these registers can get accessed in one instruction, speed is fast.

1.13.2 Features

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

1.13.3 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 *6 IO MUX and GPIO Matrix (GPIO, 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.

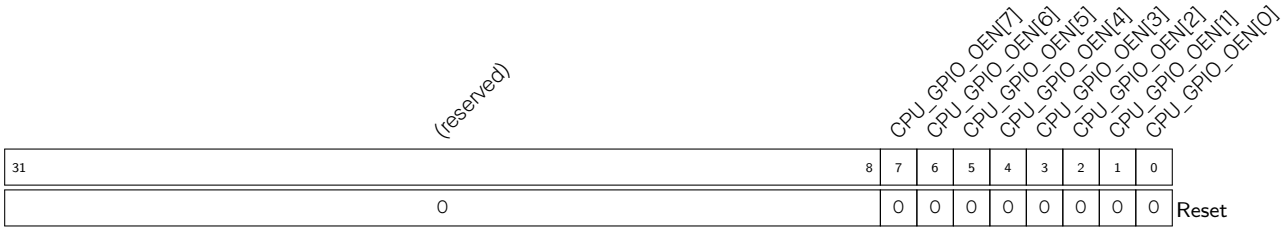
1.13.4 Register Summary

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

1.13.5 Register Description

Register 1.45. cpu_gpio_oen (0x803)



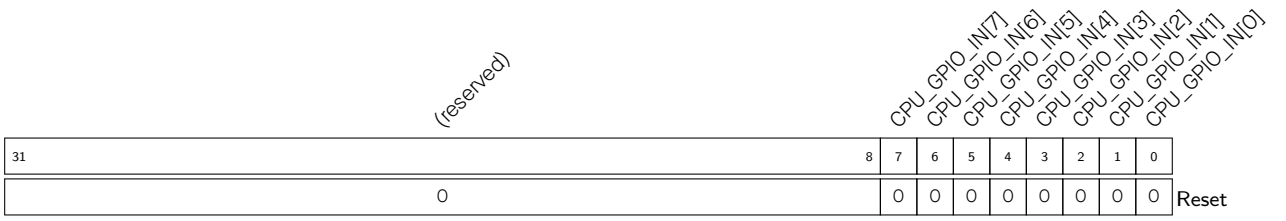
CPU_GPIO_OEN Configures whether to enable GPIOn (n=0 ~ 21) output. CPU_GPIO_OEN[7:0] correspond to output enable signals cpu_gpio_out_oen[7:0] in Table 6.12-1 *Peripheral Signals via GPIO Matrix*. CPU_GPIO_OEN value matches that of cpu_gpio_out_oen. CPU_GPIO_OEN is the enable signal of CPU_GPIO_OUT.

0: Disable GPIO output

1: Enable GPIO output

(R/W)

Register 1.46. cpu_gpio_in (0x804)



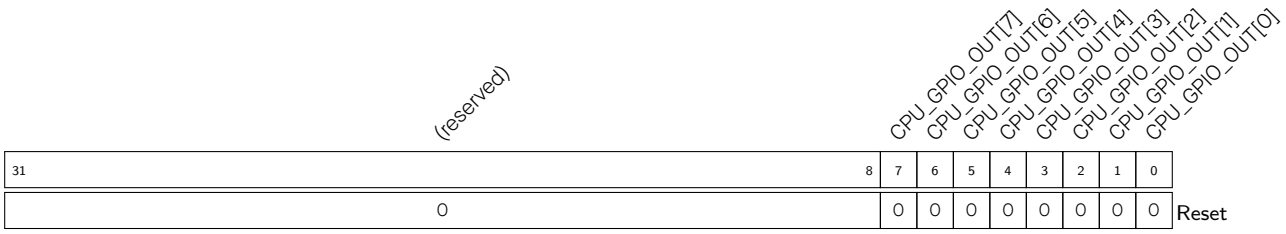
CPU_GPIO_IN Represents GPIOn (n=0 ~ 21) input value. It is a CPU CSR to read input value (1=high, 0=low) from SoC GPIO pin.

CPU_GPIO_IN[7:0] correspond to input signals cpu_gpio_in[7:0] in Table 6.12-1 *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 6.4 in Chapter *IO MUX and GPIO Matrix (GPIO, IO MUX)*.

(RO)

Register 1.47. cpu_gpio_out (0x805)



CPU_GPIO_OUT Configures GPIOn (n=0 ~ 21) output value. It is a CPU CSR to write value (1=high, 0=low) to SoC GPIO pin. The value takes effect only when CPU_GPIO_OEN is set.

CPU_GPIO_OUT[7:0] correspond to output signals cpu_gpio_out[7:0] in Table 6.12-1 *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 6.5 in Chapter *IO MUX and GPIO Matrix (GPIO, IO MUX)*.

(R/W)

1.14 Atomic (A) Extension

1.14.1 Overview

Support for atomic (A) extension is available in compliance with the RISC-V ISA Manual Volume I: Unprivileged 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.

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.

1.14.2 Functional Description

1.14.2.1 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.W instruction is encountered instead, the lock will be released eventually (not immediately) in the manner described in Section 1.14.2.2.

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

1.14.2.2 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 acknowledgment 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 1.14.2.1), 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.

1.14.2.3 AMO Instructions

An atomic memory operation (AMO) instruction executes in 3 steps:

1. Read data from the 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 the memory address given by rs1.

There are 9 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.

Chapter 2

RISC-V Trace Encoder (TRACE)

The CPU of ESP32-H2 supports instruction trace interface through the trace encoder. The trace encoder connects to the CPU's instruction trace interface, compresses the information into smaller packets, and then stores the packets in internal SRAM (see Chapter 4 *System and Memory*).

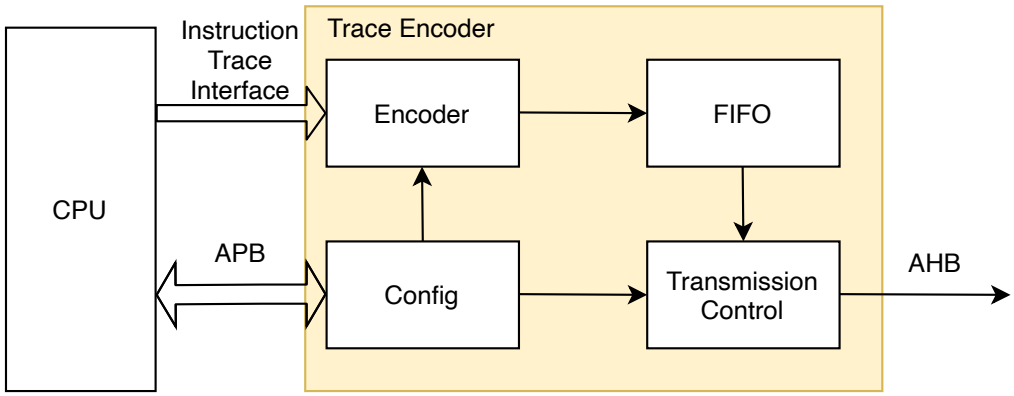


Figure 2.0-1. Trace Encoder Overview

2.1 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	a program counter change that cannot be inferred from the program binary alone
delta	a program counter change 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

2.2 Introduction

In complex systems, understanding program execution flow is not straightforward. This may be due to a number of factors, such as 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. But providing visibility of program execution is important.

That is where instruction trace comes in, which provides trace of the program execution.

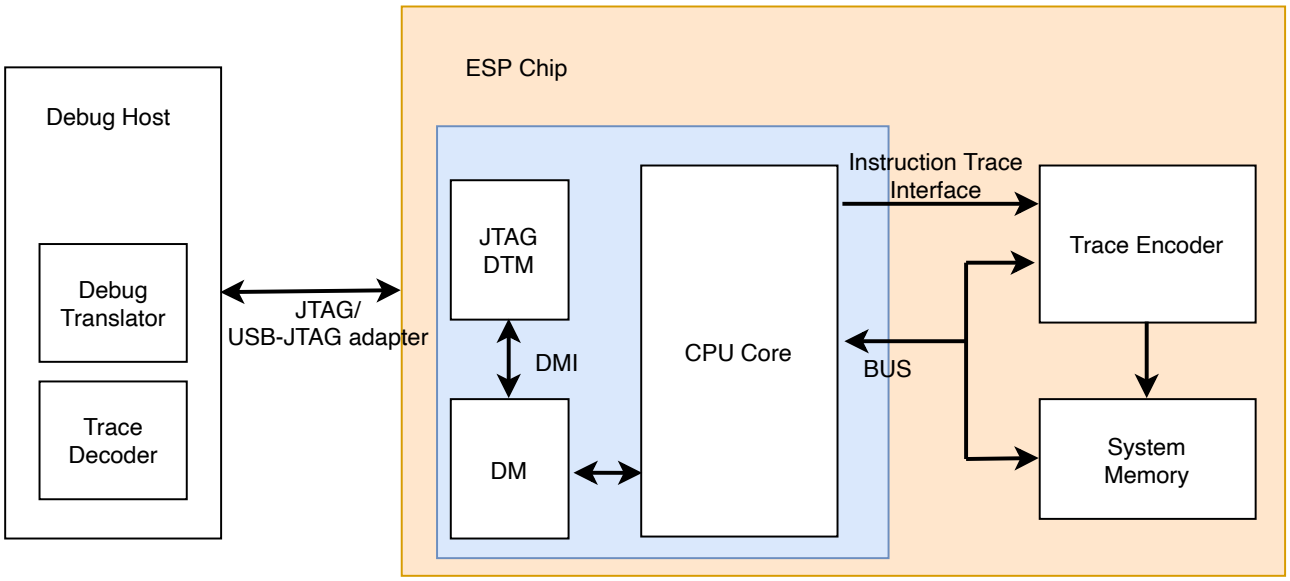


Figure 2.2-1. Trace Overview

Figure 2.2-1 shows the schematics of instruction trace:

- The CPU core provides an instruction trace interface that outputs the instruction information executed by the CPU, such as instruction address, instruction type, etc. For more details about ESP32-H2 CPU's instruction trace interface, please refer to Chapter 1 *ESP-RISC-V CPU*.
- The trace encoder connects to the CPU's instruction trace interface and compresses the information into smaller packets, and then stores the packets in system memory.
- The debugger 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.

This chapter mainly introduces the implementation details of ESP32-H2's trace encoder.

2.3 Features

- Compatible with RISC-V Processor Trace Version 1.0. See Table 2.3-1 for the implemented parameters
- Arbitrary address range of the trace memory size

- Two synchronization modes:
  - synchronization counter counts by packet
  - synchronization counter counts by cycle
- Trace lost status to indicate packet loss
- Automatic restart after packet loss
- Memory writing in loop or non-loop mode
- 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

Table 2.3-1. 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	Packets contain no context information
context_width_p	0	Packets contain no context information
ecause_width_p	5	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	Filter function is not supported
filter_excint_p	0	
filter_privilege_p	0	
filter_tval_p	0	
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
noncontext_p	1	Exclude context 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 RISC-V Processor Trace Version 1.0 > Chapter Parameters and Discovery.

2.4 Architectural Overview

As shown in Figure 2.0-1, the trace encoder contains an encoder, a FIFO, a register configuration module, and a transmission control module.

The encoder receives the 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 write bandwidth is insufficient, the FIFO may overflow and packet loss occurs. If a packet is lost, the encoder will send a packet to tell that a packet is lost, and will stop working until the FIFO is empty.

2.5 Functional Description

2.5.1 Synchronization

In order to make the trace robust, there must be regular synchronization points within the trace. Synchronization is accomplished by sending a full-value 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 2.6.3.1).

There are two synchronization modes configured via TRACE_RESYNC_MODE:

- 0: Synchronization counter counts by cycle
- 1: Synchronization counter counts by packet

You can adjust the trace bandwidth by increasing the value of TRACE_RESYNC_PROLONGED_REG to reduce the frequency of sending synchronization packets, thereby reducing the bandwidth occupied by packets.

2.5.2 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, ESP32-H2 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 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.

2.5.3 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.

2.5.4 Automatic Restart

When packets are lost due to FIFO overflow, the encoder will stop working and need to be resumed by software. If the TRACE_RESTART_ENA bit of TRACE_TRIGGER_REG is set, once the FIFO is empty, the encoder can automatically be restarted and does not need to be resumed by software.

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.

2.6 Encoder Output Packets

This section mainly introduces ESP32-H2 trace encoder output packet format. ESP32-H2 only implements mandatory instruction delta tracing. It does not support the following optional features:

- Delta address mode (run-time configurable modes is supported)
- Context information and all context-related fields
- Optional sideband signals
- Trigger outputs from the Debug Module

For details about the above features, please refer RISC-V Processor Trace Version 1.0 (referred to below as the specification).

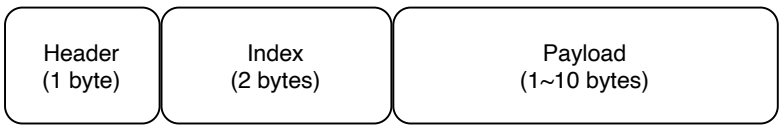


Figure 2.6-1. Trace packet Format

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 the tables below to the fields listed at the bottom. If a field consists of multiple bits, then the least significant bit is transmitted first.

2.6.1 Header

The header is 1-byte long. The format of the header is shown in Table 2.6-1.

Table 2.6-1. Header Format

Field	Bits	Description	Value
length	5	Length of whole packet	4~13
placeholder	3	Reserved	0

2.6.2 Index

The index has 2 bytes. The format of the index is shown in Table 2.6-2.

Table 2.6-2. Index Format

Field	Bits	Description	Value
index	16	The index of each packet	0~65535

2.6.3 Payload

The length of payload ranges from 1 byte to 10 bytes.

2.6.3.1 Format 3 Packets

Format 3 packets are used for synchronization, and report supporting information. There are 4 subformats defined in the specification. ESP32-H2 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 a first in an exception handler), and when synchronization has been scheduled by the expiry of the synchronization timer. The payload length is 5 bytes.

Table 2.6-3. Packet format 3 subformat 0

Field name	Bits	Description
format	2	11 (sync): Synchronization
subformat	2	00 (start): Start of tracing, or resync
branch	1	0: The address points to a branch instruction, and the branch was taken. 1: 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 value of this field must be left shifted by 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 10 bytes.

Table 2.6-4. Packet format 3 subformat 1

Field name	Bits	Description
format	2	11 (sync): Synchronization
subformat	2	01 (exception): Exception cause and trap handler address
branch	1	0: The address points to a branch instruction, and the branch was taken. 1: The instruction is not a branch or if the branch is not taken.
privilege	1	The privilege level of the reported instruction
ecause	5	Exception cause
interrupt	1	Interrupt
address	31	Full instruction address. Address must be left shifted by 1 bit in order to recreate original byte address.
tvalepc	32	Exception address if ecause is 2 and interrupt is 0, or trap value otherwise
sign_extend	6	Reserved

Format 3 Subformat 3 - Support

This packet provides supporting information to aid the decoder. It is issued when the trace is ended. The length is 1 byte.

Table 2.6-5. Packet format 3 subformat 3

Field name	Bits	Description
format	2	11 (sync): Synchronization
subformat	2	11 (support): Supporting information for the decoder
enable	1	Indicates if the encoder is enabled
qual_status	2	Indicates qualification status: • 00 (no_change): No change to filter qualification • 01 (ended_rep): Qualification ended, preceding instruction sent explicitly to indicate last qualification instruction • 10 (trace lost): One or more packets lost • 11 (ended_upd): Qualification ended, preceding te_inst would have been sent anyway due to an updiscon, even if wasn't the last qualified instruction
sign_extend	1	Reserved

2.6.3.2 Format 2 Packets

This packet contains only an instruction address, and is used when the address of an instruction must be reported, and there is no reported branch information. The length is 5 bytes.

Table 2.6-6. Packet format 2

Field name	Bits	Description
format	2	10 (addr-only): No branch information
address	31	Full instruction address
notify	1	ESP32-H2 don't support notification, so this bit is always same with the MSB of address.
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.
sign_extend	5	

2.6.3.3 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 must been at least one branch since the previous packet. This packet only supports full address mode.

Format 1- address, branch_map

The length is variable.

Table 2.6-7. Packet format 1 with address

Field name	Bits	Description
format	2	01: Includes branch information
branches	5	Number of valid bits in branch_map. The number of bits of branch_map is determined as follows: • 0: (cannot occur for this format) • 1: 1 bit • 2-3: 3 bits • 4-7: 7 bits • 8-15: 15 bits • 16-31: 31 bits For example if branches = 12, branch_map is 15-bit long, and the 12 LSBs are valid.
branch_map	Variable	An array of bits indicating whether branches are taken or not. Bit 0 represents the oldest branch instruction executed. For each bit: • 0: branch taken • 1: branch not taken The field bits is variable and determined by the "branches" field.
address	31	Full instruction address
notify	1	ESP32-H2 don't support notification, so this bit is always same with the MSB of address.
updiscon	1	If the value of this bit is different from notify, it indicates that this packet is reporting the instruction following an uninferable discontinuity and is also the instruction before an exception, privilege change or resync.

Field name	Bits	Description
sign_extend	Variable	The field bits are determined by the branches. The number of bits of sign_extend is as follows: • 1: 7 bits • 2-3: 5 bits • 4-7: 1 bit • 8-15: 1 bit • 16-32: 31 bits

Format 1- no address, branch_map

The length is 5 bytes.

Table 2.6-8. Packet format 1 without address

Field name	Bits	Description
format	2	01: includes branch information
branches	5	Number of valid bits in branch_map. The length of branch_map is 31 bits. Only 0 valid.
branch_map	31	An array of bits indicating whether branches are taken or not. Bit 0 represents the oldest branch instruction executed. For each bit: • 0: branch taken • 1: branch not taken
sign_extend	2	Reserved

2.7 Interrupt

- 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 CPU for processing, such as applying for a new memory space again.
- TRACE_FIFO_OVERFLOW_INTR: Triggered when the internal FIFO overflows and one or more packets have been lost.

After enabling the trace encoder interrupts, map them to numbered CPU interrupts through the Interrupt Matrix, so that the CPU can respond to these trace encoder interrupts. For details, please refer to Chapter 9 *Interrupt Matrix (INTMTX)*.

2.8 Programming Procedures

2.8.1 Enable Encoder

- Configure the address space for the trace memory via TRACE_MEM_START_ADDR_REG and TRACE_MEM_END_ADDR_REG

- 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: count by cycle (default)
  - 1: count by packet
- (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
- Enable the trace encoder by setting the TRACE_TRIGGER_ON field of TRACE_TRIGGER_REG

Once the encoder is enabled, it will keep tracing the CPU's instruction trace interface and writing packets to the trace memory.

2.8.2 Disable Encoder

- Disable automatic restart by clearing the TRACE_RESTART_ENA bit of TRACE_TRIGGER_REG
- Stop the encoder by setting the TRACE_TRIGGER_OFF bit of TRACE_TRIGGER_REG
- Confirm whether all data in the FIFO have been written into the memory by reading the TRACE_FIFO_EMPTY bit

2.8.3 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 2.6, 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

2.9 Register Summary

The addresses in this section are relative to RISC-V Trace Encoder base address provided in Table 4.3-2 in Chapter 4 *System and Memory*.

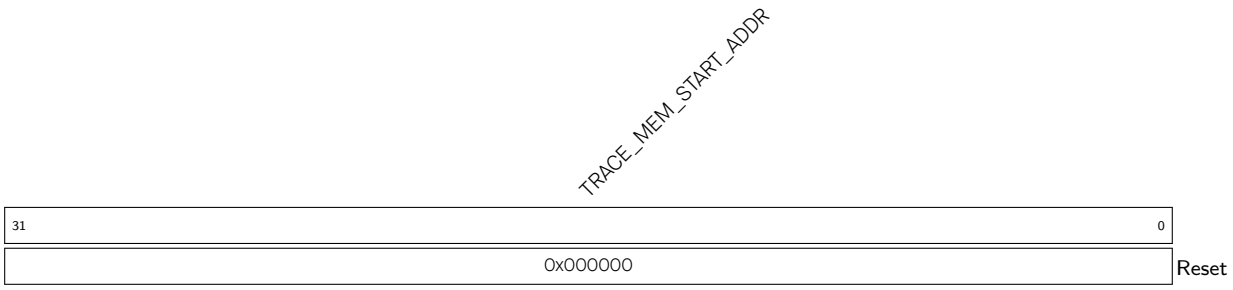
The abbreviations given in Column **Access** are explained in Section Access Types for Registers.

Name	Description	Address	Access
Memory configuration registers			
TRACE_MEM_START_ADDR_REG	Memory start address	0x0000	R/W
TRACE_MEM_END_ADDR_REG	Memory end address	0x0004	R/W
TRACE_MEM_CURRENT_ADDR_REG	Memory current address	0x0008	RO
TRACE_MEM_ADDR_UPDATE_REG	Memory address update	0x000C	WT
FIFO status register			
TRACE_FIFO_STATUS_REG	FIFO status register	0x0010	RO
Interrupt registers			
TRACE_INTR_ENA_REG	Interrupt enable register	0x0014	R/W
TRACE_INTR_RAW_REG	Interrupt raw status register	0x0018	RO
TRACE_INTR_CLR_REG	Interrupt clear register	0x001C	WT
Trace configuration registers			
TRACE_TRIGGER_REG	Trace enable register	0x0020	varies
TRACE_RESYNC_PROLONGED_REG	Resynchronization configuration register	0x0024	R/W
Clock gating control and configuration register			
TRACE_CLOCK_GATE_REG	Clock gating control register	0x0028	R/W
Version register			
TRACE_DATE_REG	Version control register	0x03FC	R/W

2.10 Registers

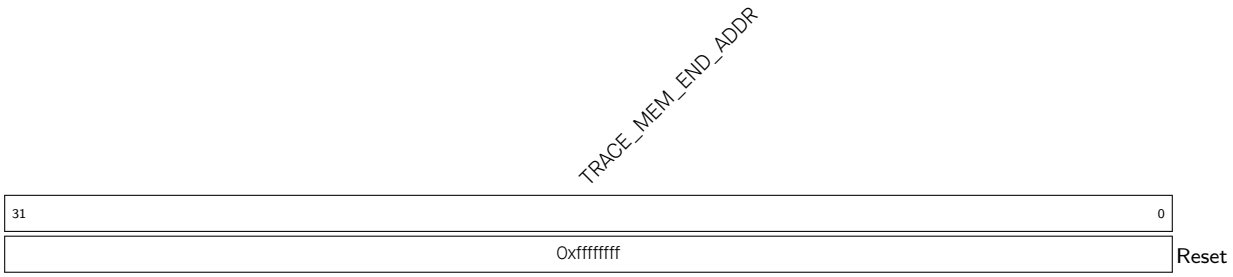
The addresses in this section are relative to RISC-V Trace Encoder base address provided in Table 4.3-2 in Chapter 4 *System and Memory*.

Register 2.1. TRACE_MEM_START_ADDR_REG (0x0000)



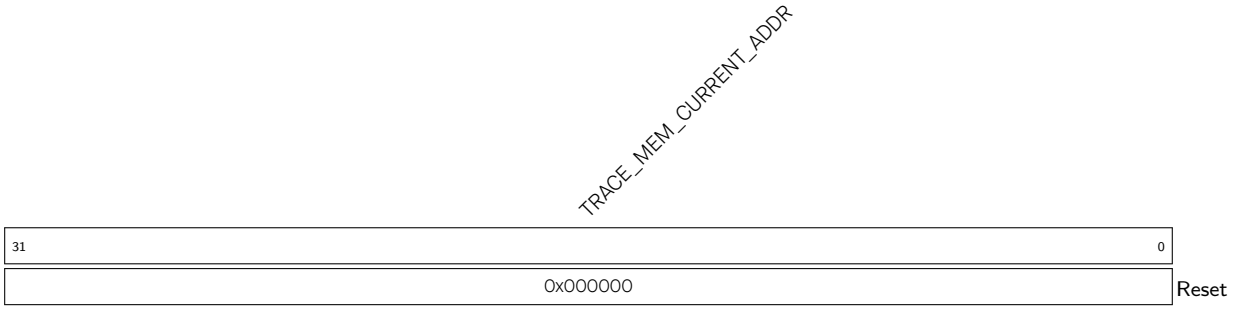
TRACE_MEM_START_ADDR Configures the start address of trace memory. (R/W)

Register 2.2. TRACE_MEM_END_ADDR_REG (0x0004)



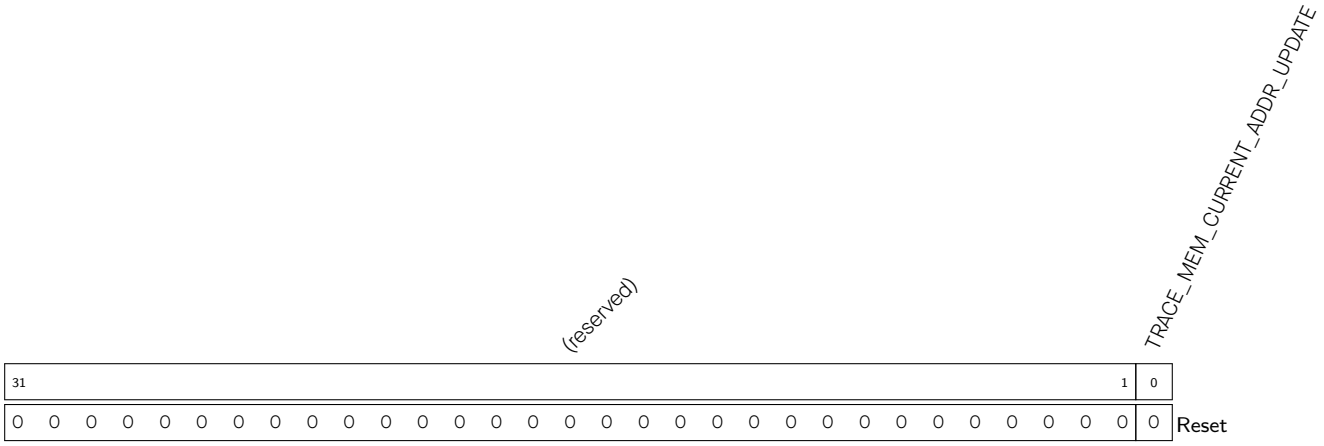
TRACE_MEM_END_ADDR Configures the end address of trace memory. (R/W)

Register 2.3. TRACE_MEM_CURRENT_ADDR_REG (0x0008)



TRACE_MEM_CURRENT_ADDR Represents the current memory address for writing. (RO)

Register 2.4. TRACE_MEM_ADDR_UPDATE_REG (0x000C)



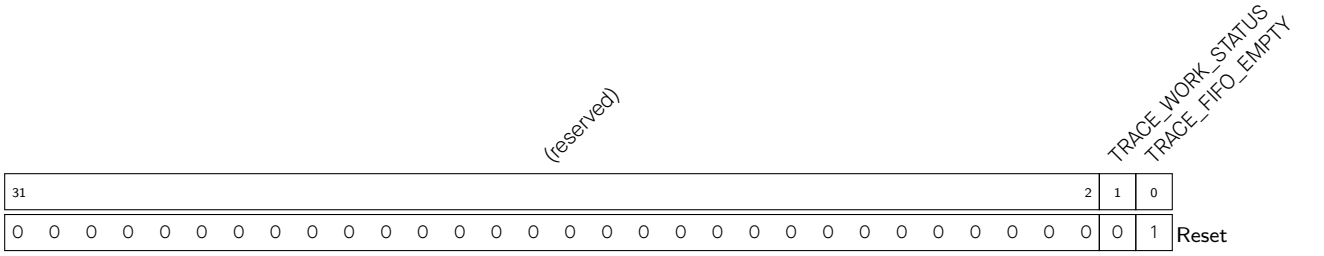
TRACE_MEM_CURRENT_ADDR_UPDATE Configures whether to update the current memory address to the start address of the memory.

0: Not update

1: Update

(WT)

Register 2.5. TRACE_FIFO_STATUS_REG (0x0010)



TRACE_FIFO_EMPTY Represents the FIFO status.

0: Not empty

1: Empty

(RO)

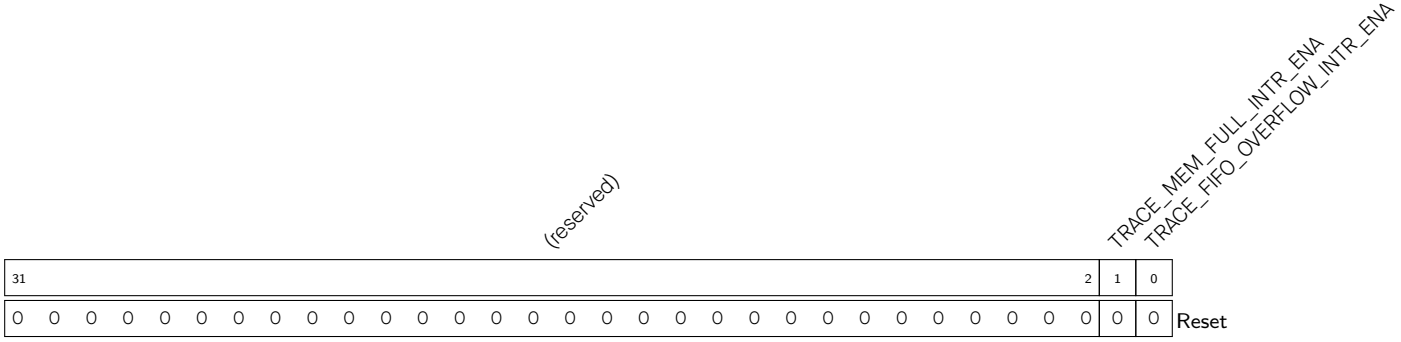
TRACE_WORK_STATUS Represents the encoder status.

0: Not tracing instruction

1: Tracing instructions and reporting packets

(RO)

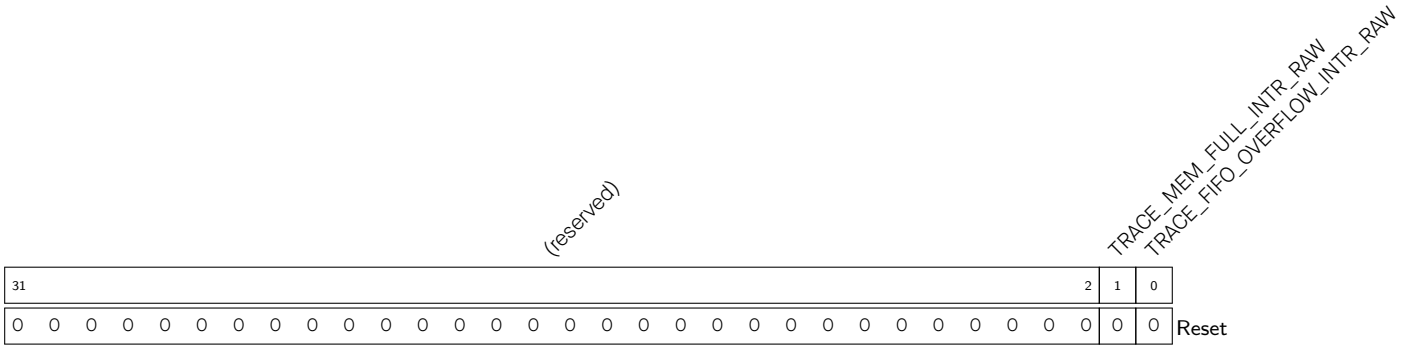
Register 2.6. TRACE_INTR_ENA_REG (0x0014)



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)

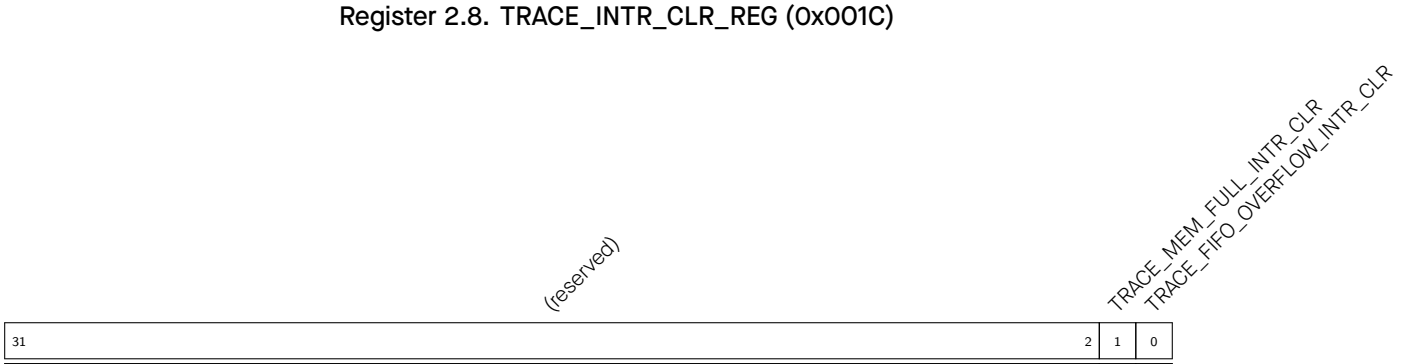
Register 2.7. TRACE_INTR_RAW_REG (0x0018)



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)

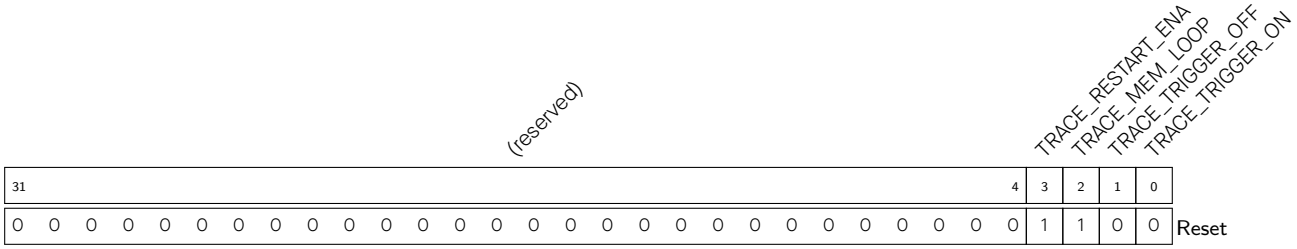
Register 2.8. TRACE_INTR_CLR_REG (0x001C)



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 2.9. TRACE_TRIGGER_REG (0x0020)



TRACE_TRIGGER_ON Configures whether or not to enable the trace encoder.

0: Invalid. No effect

1: Enable

(WT)

TRACE_TRIGGER_OFF Configures whether to stop the trace encoder.

0: Invalid. No effect

1: Stop

(WT)

TRACE_MEM_LOOP Configures memory mode.

0: Non-loop mode

1: Loop mode

(R/W)

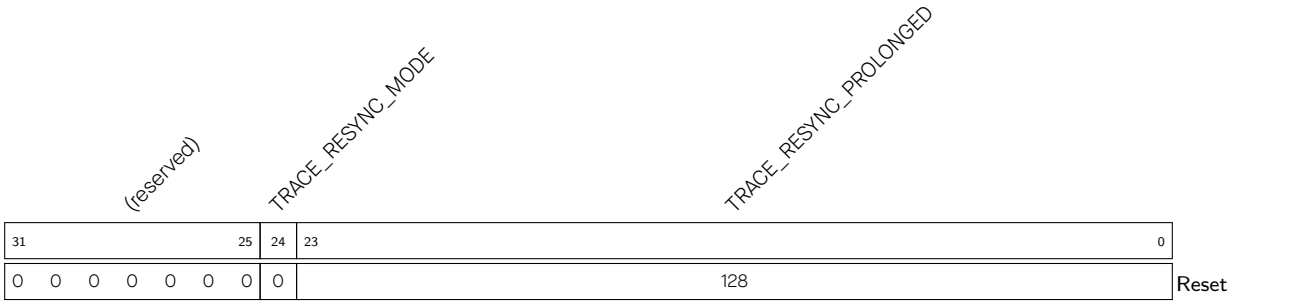
TRACE_RESTART_ENA Configures whether or not to enable the automatic restart function for the encoder.

0: Disable

1: Enable

(R/W)

Register 2.10. TRACE_RESYNC_PROLONGED_REG (0x0024)



TRACE_RESYNC_PROLONGED Configures the threshold for the synchronization counter. (R/W)

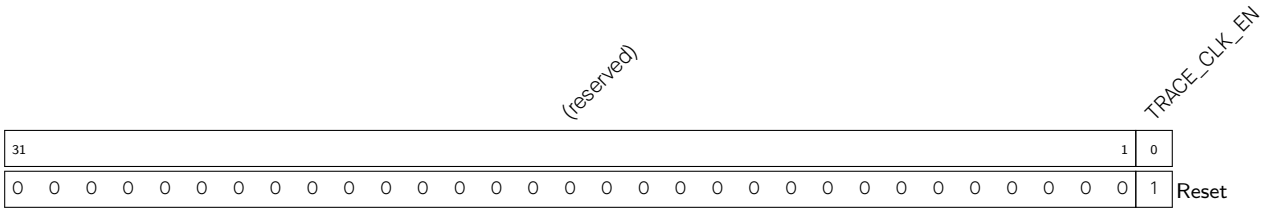
TRACE_RESYNC_MODE Configures the synchronization mode.

0: Count by cycle

1: Count by packet

(R/W)

Register 2.11. TRACE_CLOCK_GATE_REG (0x0028)



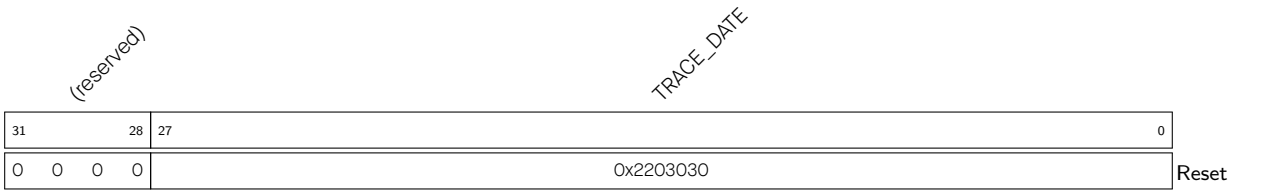
TRACE_CLK_EN Configures register clock gating. 0: Support clock only when the application writes registers to save power

1: Always force the clock on for registers

This bit doesn't affect register access

(R/W)

Register 2.12. TRACE_DATE_REG (0x03FC)



TRACE_DATE Version control register. (R/W)

Chapter 3

GDMA Controller (GDMA)

3.1 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.

The GDMA controller in ESP32-H2 has six independent channels, i.e., three transmit channels and three receive channels. These six channels are shared by peripherals with the GDMA feature, and can be assigned to any of such peripherals, including SPI2, UHCI (UART0/UART1), I2S, AES, SHA, ADC, and PARLIO. UART0 and UART1 use UHCI together.

The GDMA controller uses fixed-priority and round-robin channel arbitration schemes to manage peripherals' needs for bandwidth.

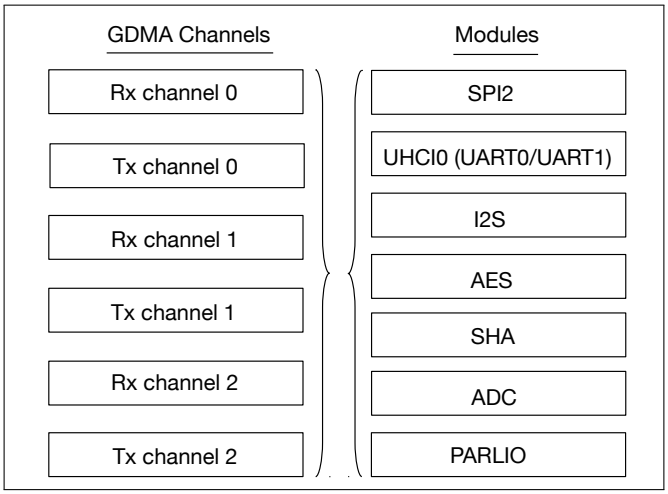


Figure 3.1-1. Modules with GDMA Feature and GDMA Channels

3.2 Features

The GDMA controller has the following features:

- AHB bus architecture
- Programmable length of data to be transferred in bytes
- Linked list of descriptors
- INCR burst transfer when accessing internal RAM
- Access to an address space of 324 KB at most in internal RAM (320 KB HP SRAM, 4 KB LP SRAM)

- Three transmit channels and three receive channels
- Software-configurable selection of peripheral requesting its service
- Fixed channel priority and round-robin channel arbitration

3.3 Architecture

In ESP32-H2, 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 internal RAM. Figure 3.3-1 shows the basic architecture of the GDMA controller.

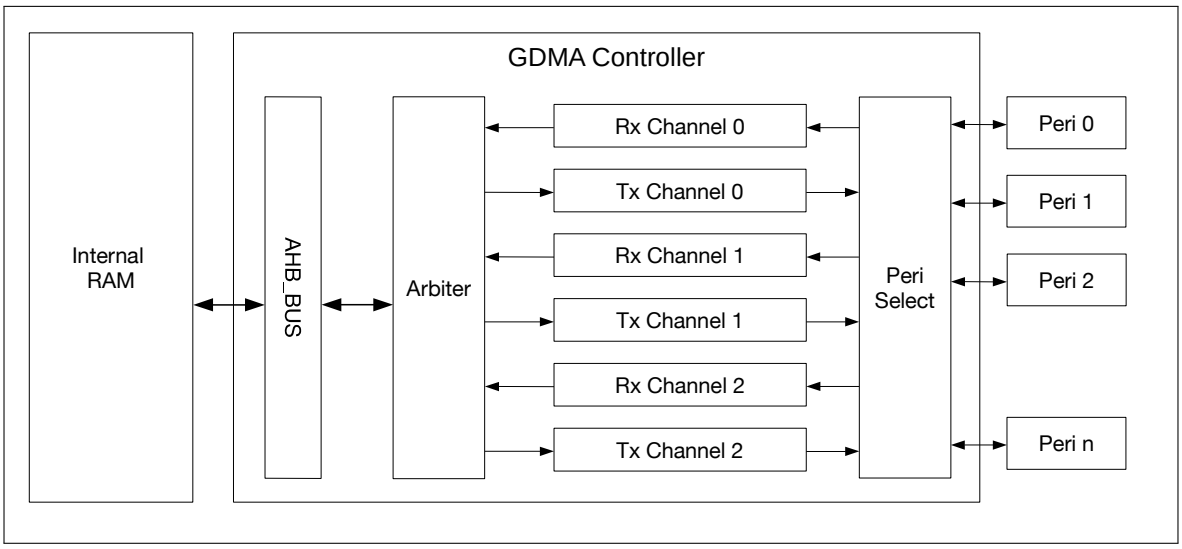


Figure 3.3-1. GDMA controller Architecture

The GDMA controller has six independent channels, i.e., three transmit channels and three receive channels. Every channel can be connected to different peripherals. In other words, channels are general-purpose, shared by peripherals.

The GDMA controller reads data from or writes data to internal RAM via AHB_BUS. Before this, the GDMA controller uses a fixed-priority arbitration scheme for channels requesting read or write access. For the available address range of Internal RAM, please see Chapter 4 *System and Memory*.

Software can use the GDMA controller through linked lists. These linked lists, stored in internal RAM, consist of outlink*n* and inlink*n*, where *n* indicates the channel number (ranging from 0 to 2). The GDMA controller reads an outlink*n* (i.e., a linked list of transmit descriptors) from internal RAM and transmits data in corresponding RAM 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 RAM according to the inlink*n*.

3.4 Functional Description

3.4.1 Linked List

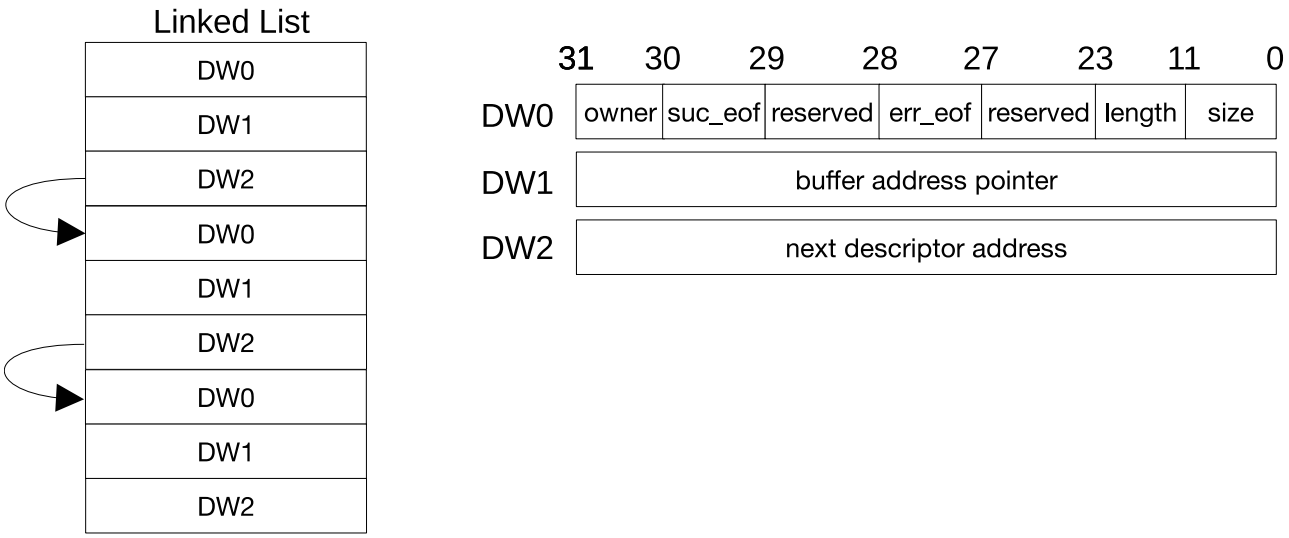


Figure 3.4-1. Structure of a Linked List

Figure 3.4-1 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 internal RAM for the GDMA controller to use. 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, and this bit in a transmit descriptor can only be automatically cleared by hardware if GDMA_OUT_AUTO_WRBACK_CH*n* is set to 1. Software can disable automatic clearing by hardware by setting GDMA_OUT_LOOP_TEST_CH*n* or GDMA_IN_LOOP_TEST_CH*n*. When software loads a linked list, this bit should be set to 1.
  Note: GDMA_OUT is the prefix of transmit channel registers, and GDMA_IN is the prefix of receive channel registers.

- suc_eof (DW0) [30]: Specifies whether the GDMA_IN_SUC_EOF_CH*n*_INT or GDMA_OUT_EOF_CH*n*_INT interrupt will be triggered when the data corresponding to this descriptor has been received or transmitted.
  1'b0: No interrupt will be triggered after the current descriptor's successful transfer;
  1'b1: An interrupt will be triggered after the current descriptor's successful transfer.
  For receive descriptors, software needs to clear this bit to 0, and hardware will set it to 1 after receiving data containing the EOF flag.
  For transmit descriptors, software needs to set this bit to 1 as needed.
  If software configures this bit to 1 in a descriptor, the GDMA will include the EOF flag in the data sent to the corresponding peripheral, indicating to the peripheral that this data segment marks the end of one transfer phase.

- reserved (DW0) [29]: Reserved. The value of this bit does not matter.
- err_eof (DW0) [28]: Specifies whether the received data has errors.
  0: The received data does not have errors.
  1: The received data has errors.
  This bit is used only when UHCI or PARLIO uses GDMA to receive data. When an error is detected in the received data segment corresponding to a descriptor, this bit in the receive descriptor is set to 1 by hardware.
- reserved (DW0) [27:24]: Reserved.
- length (DW0) [23:12]: Specifies the number of valid bytes in the buffer that this descriptor points to. This field in a transmit descriptor is written by software and indicates how many bytes can be read from the buffer; this field in a receive descriptor is written by hardware automatically and indicates how many valid bytes have been stored into 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. This field can only point to internal RAM.
- next descriptor address (DW2): Address of the next descriptor. If the current descriptor is the last one, this value could be 0. This field can only point to internal RAM.

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.

3.4.2 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 3.4-1 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 channels cannot be connected to that peripheral.

Table 3.4-1. Selecting Peripherals via Register Configuration

GDMA_PERI_IN_SEL_CH*n* GDMA_PERI_OUT_SEL_CH*n*	Peripheral
0	SPI2
1	Dummy-1
2	UHCI
3	I2S
4	Dummy-4
5	Dummy-5
6	AES
7	SHA
8	ADC

9	PARLIO
10 ~ 15	Dummy-10 ~ 15
16 ~ 63	Invalid

3.4.3 Memory-to-Memory Data Transfer

The GDMA controller also allows memory-to-memory data transfer. Such data transfer can be enabled by setting GDMA_MEM_TRANS_EN_CH*n*, which connects the output of transmit channel *n* to the input of receive channel *n*. Note that a transmit channel can only be connected to the receive channel with the same number (*n*), and GDMA_PERI_IN_SEL_CH*n* and GDMA_PERI_OUT_SEL_CH*n* should be configured to the same value corresponding to "Dummy".

3.4.4 Enabling GDMA

Software uses the GDMA controller through linked lists. When the GDMA controller receives data, software loads an inlink, configures GDMA_INLINK_ADDR_CH*n* field with the address of the first receive descriptor, and sets GDMA_INLINK_START_CH*n* bit to enable GDMA. When the GDMA controller transmits data, software loads an outlink, prepares data to be transmitted, configures GDMA_OUTLINK_ADDR_CH*n* field with address of the first transmit descriptor, and sets GDMA_OUTLINK_START_CH*n* bit to enable GDMA. GDMA_INLINK_START_CH*n* bit and GDMA_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 the GDMA_INLINK_RESTART_CH*n* bit or the GDMA_OUTLINK_RESTART_CH*n* bit (these two bits are cleared automatically by hardware). As shown in Figure 3.4-2, 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.

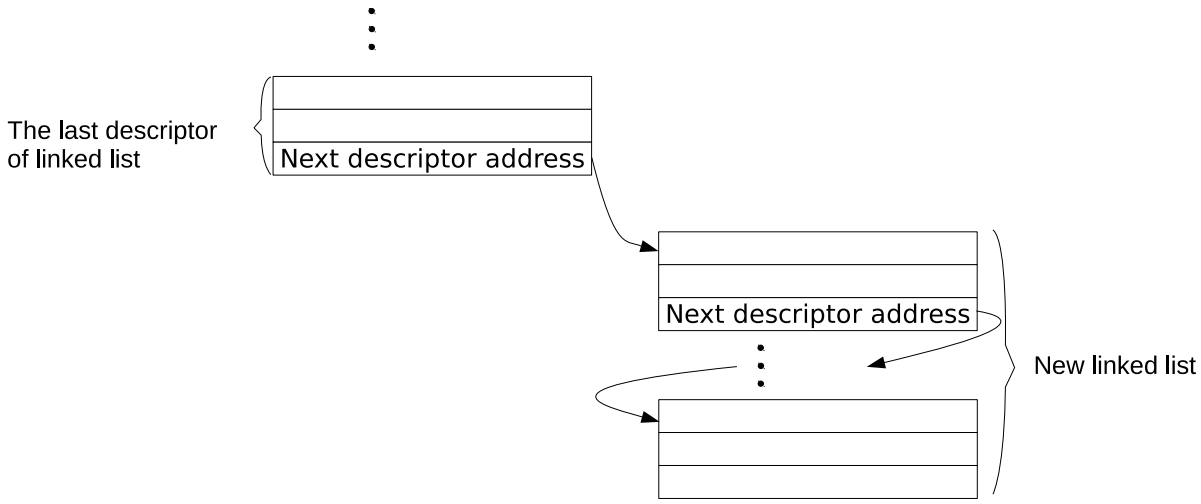


Figure 3.4-2. Relationship among Linked Lists

3.4.5 Linked List Reading Process

Once configured and enabled by software, the GDMA controller starts to read the linked list from internal RAM. 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 a descriptor error interrupt (either GDMA_IN_DSCR_ERR_CH*n*_INT or GDMA_OUT_DSCR_ERR_CH*n*_INT), and the channel will halt.

The checks performed on descriptors are:

- Owner bit check when GDMA_IN_CHECK_OWNER_CH*n* or GDMA_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 GDMA_IN_CHECK_OWNER_CH*n* or GDMA_OUT_CHECK_OWNER_CH*n* is 0.
- Buffer address pointer (DW1) check. If the buffer address pointer points to 0x40800000 ~ 0x4084FFFF (please refer to Section 3.4.7), it passes the check. Otherwise, it fails the check.

After software detects a descriptor error interrupt, it must reset the corresponding channel, and enable GDMA by setting GDMA_OUTLINK_START_CH*n* or GDMA_INLINK_START_CH*n* bit.

Note: The third word (DW2) in a descriptor can only point to a location in internal RAM, given that the third word points to the next descriptor to use and that all descriptors must be in internal memory.

3.4.6 EOF

The GDMA controller uses EOF (end of frame) flags to indicate the end of data segment transfer corresponding to a specific descriptor.

Before the GDMA controller transmits data, the GDMA_OUT_TOTAL_EOF_CH*n*_INT_ENA bit should be set to enable the GDMA_OUT_TOTAL_EOF_CH*n*_INT interrupt. If data in the buffer pointed by the last descriptor (with EOF) has been transmitted, a GDMA_OUT_TOTAL_EOF_CH*n*_INT interrupt is generated.

Before the GDMA controller receives data, the GDMA_IN_SUC_EOF_CH*n*_INT_ENA bit should be set to enable the GDMA_IN_SUC_EOF_CH*n*_INT interrupt. If a data segment with an EOF flag has been received successfully, a GDMA_IN_SUC_EOF_CH*n*_INT interrupt is generated. In addition, when the GDMA channel is

connected to UHCI or PARLIO, the GDMA controller also supports the GDMA_IN_ERR_CH*n*_EOF_INT interrupt. This interrupt is enabled by setting the GDMA_IN_ERR_EOF_CH*n*_INT_ENA bit, and it indicates that a data segment corresponding to a descriptor has been received with errors.

When detecting a GDMA_OUT_TOTAL_EOF_CH*n*_INT or a GDMA_IN_SUC_EOF_CH*n*_INT interrupt, software can record the value of the GDMA_OUT_EOF_DES_ADDR_CH*n* or GDMA_IN_SUC_EOF_DES_ADDR_CH*n* field, i.e., address of the last descriptor. Therefore, software can tell which descriptors have been used and reclaim them as needed.

Note: In this chapter, EOF of transmit descriptors refers to suc_eof, while EOF of receive descriptors refers to both suc_eof and err_eof.

3.4.7 Accessing Internal RAM

Any transmit and receive channels of GDMA can access 0x40800000 ~ 0x4084FFFF in internal RAM. To improve data transfer efficiency, GDMA can send data in burst mode, which is disabled by default. This mode is enabled for receive channels by setting GDMA_IN_DATA_BURST_EN_CH*n*, and enabled for transmit channels by setting GDMA_OUT_DATA_BURST_EN_CH*n*.

Table 3.4-2. Descriptor Field Alignment Requirements

Inlink/Outlink	Burst Mode	Size	Length	Buffer Address Pointer
Inlink	0	—	—	—
	1	Word-aligned	—	Word-aligned
Outlink	0	—	—	—
	1	—	—	—

Table 3.4-2 lists the requirements for descriptor field alignment when accessing internal RAM.

When the burst mode is disabled, size, length, and buffer address pointer in both transmit and receive descriptors do not need to be word-aligned. That is, for a descriptor, GDMA can read data of specified length (1 ~ 4095 bytes) from any start addresses in the accessible address range, or write received data of the specified length (1 ~ 4095 bytes) to any contiguous addresses in the accessible address range.

When the burst mode is enabled, size, length, and buffer address pointer in transmit descriptors are also not necessarily word-aligned. However, size and buffer address pointer in receive descriptors except length should be word-aligned.

3.4.8 Arbitration

To ensure timely response to peripherals running at a high speed with low latency (such as SPI), the GDMA controller implements a fixed-priority channel arbitration scheme. That is to say, each channel can be assigned a priority from 0 ~ 5 (in total 6 levels). The larger the number, the higher the priority, and the more timely the response. When several channels are assigned the same priority, the GDMA controller adopts a round-robin arbitration scheme.

3.4.9 Event Task Matrix Feature

The GDMA controller on ESP32-H2 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 10 Event Task Matrix (SOC_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 GDMA_INLNIK_START_CH*n* and GDMA_OUTLINK_START_CH*n*. When GDMA_IN _ETM_EN_CH*n* or GDMA_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 GDMA_IN_ETM_EN_CH*n* or GDMA_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_SUC_EOF_CH*n*: Indicates that the data corresponding to a transmit descriptor has been transmitted or received via channel *n* and the EOF bit of this descriptor is 1.
- GDMA_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.

3.5 GDMA Interrupts

GDMA on ESP32-H2 can generate the following 6 interrupt signals, and send them to *Interrupt Matrix (INTMTX)*:

- GDMA_IN_CH0_INTR
- GDMA_IN_CH1_INTR
- GDMA_IN_CH2_INTR

- GDMA_OUT_CH0_INTR
- GDMA_OUT_CH1_INTR
- GDMA_OUT_CH2_INTR

The following interrupt sources trigger the GDMA_IN_CH*n*_INTR signals:

- DMA_INFIFO_OVF_CH*n*_INT: Triggered when the RX FIFO of GDMA overflows.
- GDMA_INFIFO_UDF_CH*n*_INT: Triggered when the RX FIFO of GDMA underflows.
- GDMA_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*.
- GDMA_IN_DSCR_ERR_CH*n*_INT: Triggered when an error is detected in a receive descriptor on receive channel *n*.
- GDMA_IN_ERR_EOF_CH*n*_INT: Triggered when an error is detected in the data segment corresponding to a descriptor received via receive channel *n*. This interrupt is used only for UHCI peripheral (UART0 or UART1) or PARLIO.
- GDMA_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*.
- GDMA_IN_DONE_CH*n*_INT: Triggered when all data corresponding to a receive descriptor has been received via receive channel *n*.

The following interrupt sources trigger the GDMA_OUT_CH*n*_INTR signals:

- GMA_OUTFIFO_OVF_CH*n*_INT: Triggered when the TX FIFO of GDMA overflows.
- GDMA_OUTFIFO_UDF_CH*n*_INT: Triggered when the TX FIFO of GDMA underflows.
- GDMA_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*.
- GDMA_OUT_DSCR_ERR_CH*n*_INT: Triggered when an error is detected in a transmit descriptor on transmit channel *n*.
- GDMA_OUT_EOF_CH*n*_INT: Triggered when EOF in a transmit descriptor is 1 and data corresponding to this descriptor has been sent via transmit channel *n*. If GDMA_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 GDMA_OUT_EOF_MODE_CH*n* is 1, this interrupt is triggered when the last byte of data is taken from GDMA's transmit channel.
- GDMA_OUT_DONE_CH*n*_INT: Triggered when all data corresponding to a transmit descriptor has been sent via transmit channel *n*.

3.6 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.

3.6.1 Programming Procedures for GDMA's Transmit Channel

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

1. Set GDMA_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 GDMA_OUTLINK_ADDR_CH*n* with address of the first transmit descriptor.
3. Configure GDMA_PERI_OUT_SEL_CH*n* with the value corresponding to the peripheral to be connected, as shown in Table 3.4-1.
4. Set GDMA_OUTLINK_START_CH*n* to enable GDMA's transmit channel for data transfer.
5. Configure and enable the corresponding peripheral. See details in individual chapters of these peripherals.
6. Wait for the GDMA_OUT_TOTAL_EOF_CH*n*_INT interrupt, which indicates the completion of data transfer.

3.6.2 Programming Procedures for GDMA's Receive Channel

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

1. Set GDMA_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 GDMA_INLINK_ADDR_CH*n* with address of the first receive descriptor.
3. Configure GDMA_PERI_IN_SEL_CH*n* with the value corresponding to the peripheral to be connected, as shown in Table 3.4-1.
4. Set GDMA_INLINK_START_CH*n* to enable GDMA's receive channel for data transfer.
5. Configure and enable the corresponding peripheral. See details in individual chapters of these peripherals.

3.6.3 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 GDMA_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 GDMA_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 GDMA_OUTLINK_ADDR_CH*n* with address of the first transmit descriptor.
4. Load an inlink, and configure GDMA_INLINK_ADDR_CH*n* with address of the first receive descriptor.
5. Set GDMA_MEM_TRANS_EN_CH*n* to enable memory-to-memory transfer.
6. Set GDMA_OUTLINK_START_CH*n* to enable GDMA's transmit channel for data transfer.
7. Set GDMA_INLINK_START_CH*n* to enable GDMA's receive channel for data transfer.

8. If the suc_eof bit is set in a transmit descriptor, a GDMA_IN_SUC_EOF_CH*n*_INT interrupt will be triggered when the data segment corresponding to this descriptor has been transmitted.

3.7 Register Summary

The addresses in this section are relative to GDMA base address provided in Table 4.3-2 in Chapter 4 *System and Memory*.

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

Name	Description	Address	Access
Interrupt Registers			
GDMA_IN_INT_RAW_CH0_REG	Raw interrupt status of RX channel 0	0x0000	R/WTC/SS
GDMA_IN_INT_ST_CH0_REG	Masked interrupt status of RX channel 0	0x0004	RO
GDMA_IN_INT_ENA_CH0_REG	Interrupt enable bits of RX channel 0	0x0008	R/W
GDMA_IN_INT_CLR_CH0_REG	Interrupt clear bits of RX channel 0	0x000C	WT
GDMA_IN_INT_RAW_CH1_REG	Raw interrupt status interrupt of TX channel 1	0x0010	R/WTC/SS
GDMA_IN_INT_ST_CH1_REG	Masked interrupt status of TX channel 1	0x0014	RO
GDMA_IN_INT_ENA_CH1_REG	Interrupt enable bits of TX channel 1	0x0018	R/W
GDMA_IN_INT_CLR_CH1_REG	Interrupt clear bits of TX channel 1	0x001C	WT
GDMA_IN_INT_RAW_CH2_REG	Raw interrupt status of RX channel 2	0x0020	R/WTC/SS
GDMA_IN_INT_ST_CH2_REG	Masked interrupt status of RX channel 2	0x0024	RO
GDMA_IN_INT_ENA_CH2_REG	Interrupt enable bits of RX channel 2	0x0028	R/W
GDMA_IN_INT_CLR_CH2_REG	Interrupt clear bits of RX channel 2	0x002C	WT
GDMA_OUT_INT_RAW_CH0_REG	Raw interrupt status of TX channel 0	0x0030	R/WTC/SS
GDMA_OUT_INT_ST_CH0_REG	Masked interrupt status of TX channel 0	0x0034	RO
GDMA_OUT_INT_ENA_CH0_REG	Interrupt enable bits of TX channel 0	0x0038	R/W
GDMA_OUT_INT_CLR_CH0_REG	Interrupt clear bits of TX channel 0	0x003C	WT
GDMA_OUT_INT_RAW_CH1_REG	Raw interrupt status of TX channel 1	0x0040	R/WTC/SS
GDMA_OUT_INT_ST_CH1_REG	Masked interrupt status of TX channel 1	0x0044	RO
GDMA_OUT_INT_ENA_CH1_REG	Interrupt enable bits of TX channel 1	0x0048	R/W
GDMA_OUT_INT_CLR_CH1_REG	Interrupt clear bits of TX channel 1	0x004C	WT
GDMA_OUT_INT_RAW_CH2_REG	Raw interrupt status of TX channel 2	0x0050	R/WTC/SS
GDMA_OUT_INT_ST_CH2_REG	Masked interrupt status of TX channel 2	0x0054	RO
GDMA_OUT_INT_ENA_CH2_REG	Interrupt enable bits of TX channel 2	0x0058	R/W
GDMA_OUT_INT_CLR_CH2_REG	Interrupt clear bits of TX channel 2	0x005C	WT
Debug Registers			
GDMA_AHB_TEST_REG	Reserved	0x0060	R/W
Configuration Registers			
GDMA_MISC_CONF_REG	Miscellaneous register	0x0064	R/W
GDMA_IN_CONF0_CH0_REG	Configuration register 0 of RX channel 0	0x0070	R/W
GDMA_IN_CONF1_CH0_REG	Configuration register 1 of RX channel 0	0x0074	R/W
GDMA_IN_POP_CH0_REG	Pop control register of RX channel 0	0x007C	varies

Name	Description	Address	Access
GDMA_IN_LINK_CH0_REG	Linked list descriptor configuration and control register of RX channel 0	0x0080	varies
GDMA_OUT_CONF0_CH0_REG	Configuration register 0 of TX channel 0	0x00D0	R/W
GDMA_OUT_CONF1_CH0_REG	Configuration register 1 of TX channel 0	0x00D4	R/W
GDMA_OUT_PUSH_CH0_REG	Push control register of RX channel 0	0x00DC	varies
GDMA_OUT_LINK_CH0_REG	Linked list descriptor configuration and control register of TX channel 0	0x00E0	varies
GDMA_IN_CONF0_CH1_REG	Configuration register 0 of RX channel 1	0x0130	R/W
GDMA_IN_CONF1_CH1_REG	Configuration register 1 of RX channel 1	0x0134	R/W
GDMA_IN_POP_CH1_REG	Pop control register of RX channel 1	0x013C	varies
GDMA_IN_LINK_CH1_REG	Linked list descriptor configuration and control register of RX channel 1	0x0140	varies
GDMA_OUT_CONF0_CH1_REG	Configuration register 0 of TX channel 1	0x0190	R/W
GDMA_OUT_CONF1_CH1_REG	Configuration register 1 of TX channel 1	0x0194	R/W
GDMA_OUT_PUSH_CH1_REG	Push control register of RX channel 1	0x019C	varies
GDMA_OUT_LINK_CH1_REG	Linked list descriptor configuration and control register of TX channel 1	0x01A0	varies
GDMA_IN_CONF0_CH2_REG	Configuration register 0 of RX channel 2	0x01F0	R/W
GDMA_IN_CONF1_CH2_REG	Configuration register 1 of RX channel 2	0x01F4	R/W
GDMA_IN_POP_CH2_REG	Pop control register of RX channel 2	0x01FC	varies
GDMA_IN_LINK_CH2_REG	Linked list descriptor configuration and control register of RX channel 2	0x0200	varies
GDMA_OUT_CONF0_CH2_REG	Configuration register 0 of TX channel 2	0x0250	R/W
GDMA_OUT_CONF1_CH2_REG	Configuration register 1 of TX channel 2	0x0254	R/W
GDMA_OUT_PUSH_CH2_REG	Push control register of RX channel 2	0x025C	varies
GDMA_OUT_LINK_CH2_REG	Linked list descriptor configuration and control register of TX channel 2	0x0260	varies
Version Register			
GDMA_DATE_REG	Version control register	0x0068	R/W
Status Registers			
GDMA_INFIFO_STATUS_CH0_REG	Receive FIFO status of RX channel 0	0x0078	RO
GDMA_IN_STATE_CH0_REG	Receive status of RX channel 0	0x0084	RO
GDMA_IN_SUC_EOF_DES_ADDR_CH0_REG	Receive descriptor address when EOF occurs on RX channel 0	0x0088	RO
GDMA_IN_ERR_EOF_DES_ADDR_CH0_REG	Receive descriptor address when errors occur of RX channel 0	0x008C	RO
GDMA_IN_DSCR_CH0_REG	Address of the next receive descriptor pointed by the current pre-read receive descriptor on RX channel 0	0x0090	RO
GDMA_IN_DSCR_BF0_CH0_REG	Address of the current pre-read receive descriptor on RX channel 0	0x0094	RO

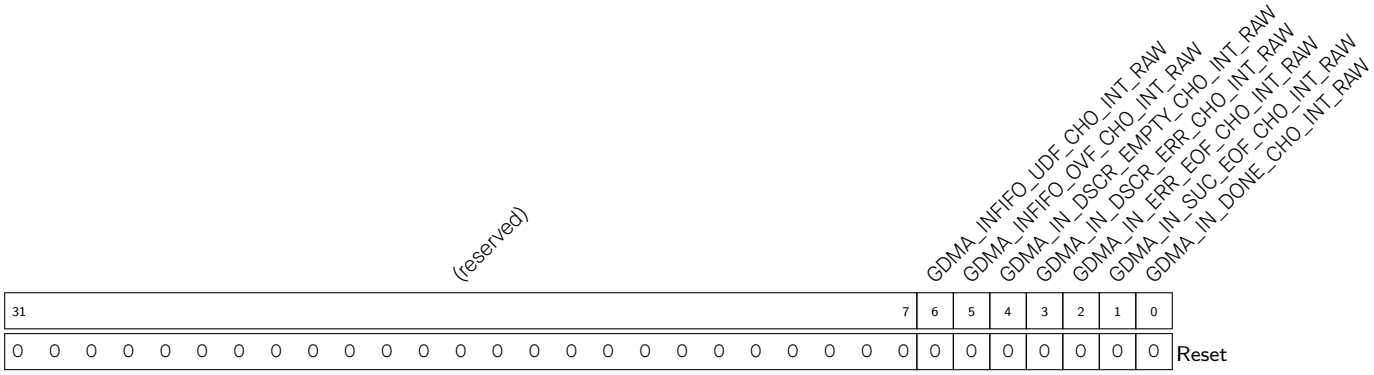
Name	Description	Address	Access
GDMA_IN_DSCR_BF1_CH0_REG	Address of the previous pre-read receive descriptor on RX channel 0	0x0098	RO
GDMA_OUTFIFO_STATUS_CH0_REG	Transmit FIFO status of TX channel 0	0x00D8	RO
GDMA_OUT_STATE_CH0_REG	Transmit status of TX channel 0	0x00E4	RO
GDMA_OUT_EOF_DES_ADDR_CH0_REG	Transmit descriptor address when EOF occurs on TX channel 0	0x00E8	RO
GDMA_OUT_EOF_BFR_DES_ADDR_CH0_REG	The last transmit descriptor address when EOF occurs on TX channel 0	0x00EC	RO
GDMA_OUT_DSCR_CH0_REG	Address of the next transmit descriptor pointed by the current pre-read transmit descriptor on TX channel 0	0x00F0	RO
GDMA_OUT_DSCR_BF0_CH0_REG	Address of the current pre-read transmit descriptor on TX channel 0	0x00F4	RO
GDMA_OUT_DSCR_BF1_CH0_REG	Address of the previous pre-read transmit descriptor on TX channel 0	0x00F8	RO
GDMA_INFIFO_STATUS_CH1_REG	Receive FIFO status of RX channel 1	0x0138	RO
GDMA_IN_STATE_CH1_REG	Receive status of RX channel 1	0x0144	RO
GDMA_IN_SUC_EOF_DES_ADDR_CH1_REG	Receive descriptor address when EOF occurs on RX channel 1	0x0148	RO
GDMA_IN_ERR_EOF_DES_ADDR_CH1_REG	Receive descriptor address when errors occur of RX channel 1	0x014C	RO
GDMA_IN_DSCR_CH1_REG	Address of the next receive descriptor pointed by the current pre-read receive descriptor on RX channel 1	0x0150	RO
GDMA_IN_DSCR_BF0_CH1_REG	Address of the current pre-read receive descriptor on RX channel 1	0x0154	RO
GDMA_IN_DSCR_BF1_CH1_REG	Address of the previous pre-read receive descriptor on RX channel 1	0x0158	RO
GDMA_OUTFIFO_STATUS_CH1_REG	Transmit FIFO status of TX channel 1	0x0198	RO
GDMA_OUT_STATE_CH1_REG	Transmit status of TX channel 1	0x01A4	RO
GDMA_OUT_EOF_DES_ADDR_CH1_REG	Transmit descriptor address when EOF occurs on TX channel 1	0x01A8	RO
GDMA_OUT_EOF_BFR_DES_ADDR_CH1_REG	The last transmit descriptor address when EOF occurs on TX channel 1	0x01AC	RO
GDMA_OUT_DSCR_CH1_REG	Address of the next transmit descriptor pointed by the current pre-read transmit descriptor on TX channel 1	0x01B0	RO
GDMA_OUT_DSCR_BF0_CH1_REG	Address of the current pre-read transmit descriptor on TX channel 1	0x01B4	RO
GDMA_OUT_DSCR_BF1_CH1_REG	Address of the previous pre-read transmit descriptor on TX channel 1	0x01B8	RO
GDMA_INFIFO_STATUS_CH2_REG	Receive FIFO status of RX channel 2	0x01F8	RO
GDMA_IN_STATE_CH2_REG	Receive status of RX channel 2	0x0204	RO

Name	Description	Address	Access
GDMA_IN_SUC_EOF_DES_ADDR_CH2_REG	Receive descriptor address when EOF occurs on RX channel 2	0x0208	RO
GDMA_IN_ERR_EOF_DES_ADDR_CH2_REG	Receive descriptor address when errors occur of RX channel 2	0x020C	RO
GDMA_IN_DSCR_CH2_REG	Address of the next receive descriptor pointed by the current pre-read receive descriptor on RX channel 2	0x0210	RO
GDMA_IN_DSCR_BF0_CH2_REG	Address of the current pre-read receive descriptor on RX channel 2	0x0214	RO
GDMA_IN_DSCR_BF1_CH2_REG	Address of the previous pre-read receive descriptor on RX channel 2	0x0218	RO
GDMA_OUTFIFO_STATUS_CH2_REG	Transmit FIFO status of TX channel 2	0x0258	RO
GDMA_OUT_STATE_CH2_REG	Transmit status of TX channel 2	0x0264	RO
GDMA_OUT_EOF_DES_ADDR_CH2_REG	Transmit descriptor address when EOF occurs on TX channel 2	0x0268	RO
GDMA_OUT_EOF_BFR_DES_ADDR_CH2_REG	The last transmit descriptor address when EOF occurs on TX channel 2	0x026C	RO
GDMA_OUT_DSCR_CH2_REG	Address of the next transmit descriptor pointed by the current pre-read transmit descriptor on TX channel 2	0x0270	RO
GDMA_OUT_DSCR_BF0_CH2_REG	Address of the current pre-read transmit descriptor on TX channel 2	0x0274	RO
GDMA_OUT_DSCR_BF1_CH2_REG	Address of the previous pre-read transmit descriptor on TX channel 2	0x0278	RO
Priority Registers			
GDMA_IN_PRI_CH0_REG	Priority register of RX channel 0	0x009C	R/W
GDMA_OUT_PRI_CH0_REG	Priority register of TX channel 0	0x00FC	R/W
GDMA_IN_PRI_CH1_REG	Priority register of RX channel 1	0x015C	R/W
GDMA_OUT_PRI_CH1_REG	Priority register of TX channel 1	0x01BC	R/W
GDMA_IN_PRI_CH2_REG	Priority register of RX channel 2	0x021C	R/W
GDMA_OUT_PRI_CH2_REG	Priority register of TX channel 2	0x027C	R/W
Peripheral Selection Registers			
GDMA_IN_PERI_SEL_CH0_REG	Peripheral selection register of RX channel 0	0x00A0	R/W
GDMA_OUT_PERI_SEL_CH0_REG	Peripheral selection register of TX channel 0	0x0100	R/W
GDMA_IN_PERI_SEL_CH1_REG	Peripheral selection register of RX channel 1	0x0160	R/W
GDMA_OUT_PERI_SEL_CH1_REG	Peripheral selection register of TX channel 1	0x01C0	R/W
GDMA_IN_PERI_SEL_CH2_REG	Peripheral selection register of RX channel 2	0x0220	R/W
GDMA_OUT_PERI_SEL_CH2_REG	Peripheral selection register of TX channel 2	0x0280	R/W

3.8 Registers

The addresses in this section are relative to GDMA base address provided in Table 4.3-2 in Chapter 4 *System and Memory*.

Register 3.1. GDMA_IN_INT_RAW_CH*n*_REG (*n*: 0-2) (0x0000+0x10n*)**



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

GDMA_IN_SUC_EOF_CH*n*_INT_RAW The raw interrupt status of GDMA_IN_SUC_EOF_CH*n*_INT. For UHCI this bit turns to 1 when the last data byte pointed by one receive descriptor has been received and no data error is detected for RX channel 0. (R/WTC/SS)

GDMA_IN_ERR_EOF_CH*n*_INT_RAW The raw interrupt status of GDMA_IN_ERR_EOF_CH*n*_INT. Valid only for UHCI or PARLIO. (R/WTC/SS)

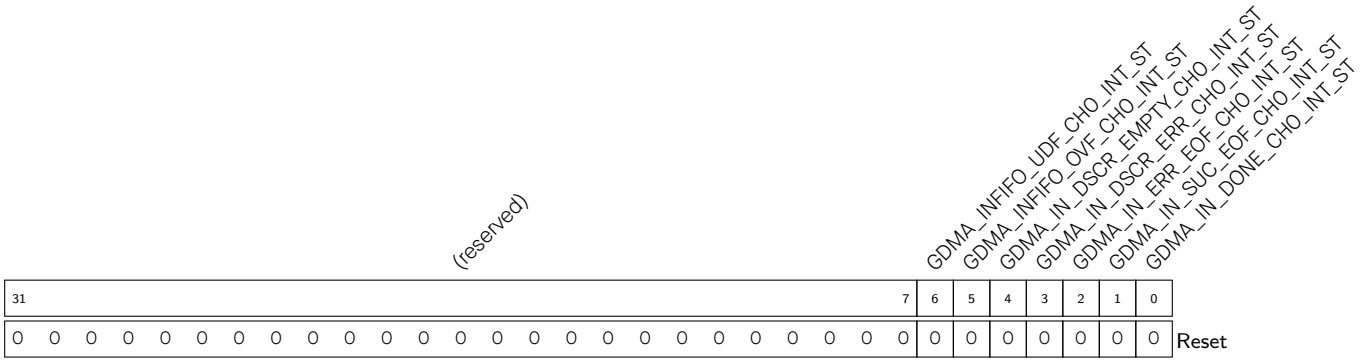
GDMA_IN_DSCR_ERR_CH*n*_INT_RAW The raw interrupt status of GDMA_IN_DSCR_ERR_CH*n*_INT. (R/WTC/SS)

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

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

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

Register 3.2. GDMA_IN_INT_ST_CH*n*_REG (*n*: 0-2) (0x0004+0x10n*)**



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

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

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

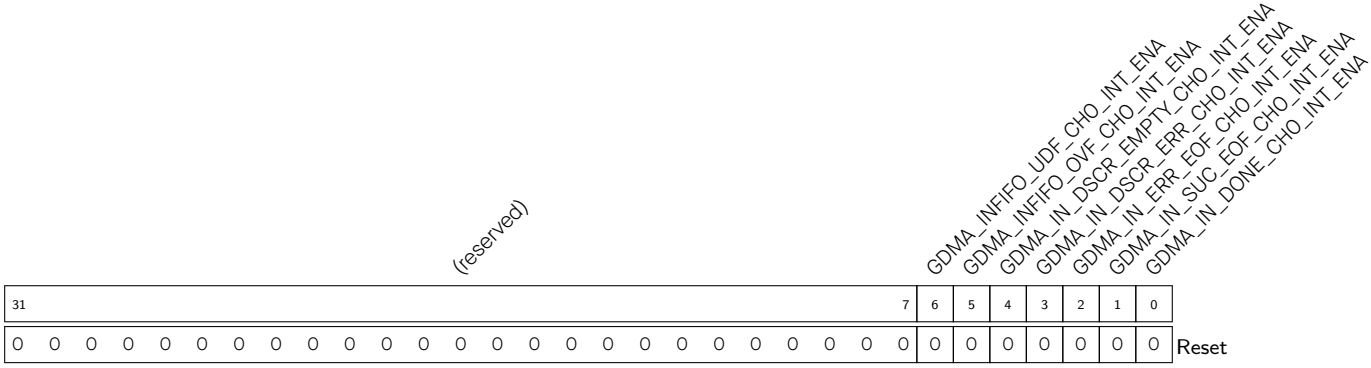
GDMA_IN_DSCR_ERR_CH*n*_INT_ST The masked interrupt status of GDMA_IN_DSCR_ERR_CH*n*_INT. (RO)

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

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

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

Register 3.3. GDMA_IN_INT_ENA_CH*n*_REG (*n*: 0-2) (0x0008+0x10***n*)



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

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

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

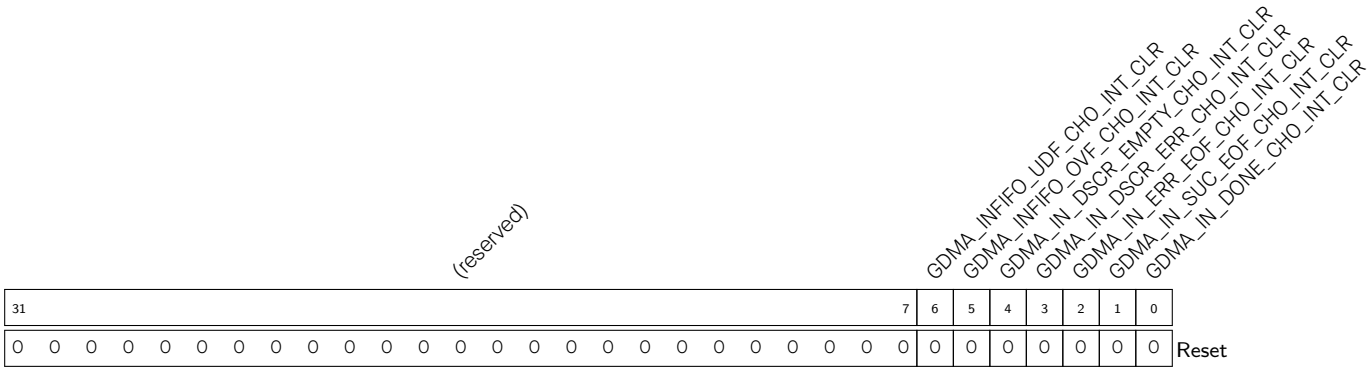
GDMA_IN_DSCR_ERR_CH*n*_INT_ENA Write 1 to enable GDMA_IN_DSCR_ERR_CH*n*_INT. (R/W)

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

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

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

Register 3.4. GDMA_IN_INT_CLR_CH*n*_REG (*n*: 0-2) (0x000C+0x10n*)**



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

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

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

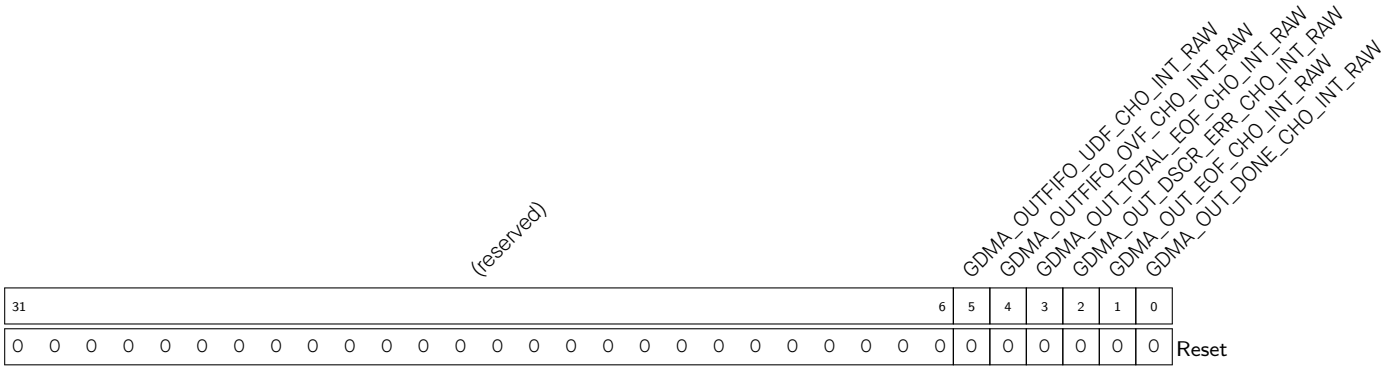
GDMA_IN_DSCR_ERR_CH*n*_INT_CLR Write 1 to clear GDMA_IN_DSCR_ERR_CH*n*_INT. (WT)

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

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

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

Register 3.5. GDMA_OUT_INT_RAW_CH*n*_REG (*n*: 0-2) (0x0030+0x10n*)**



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

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

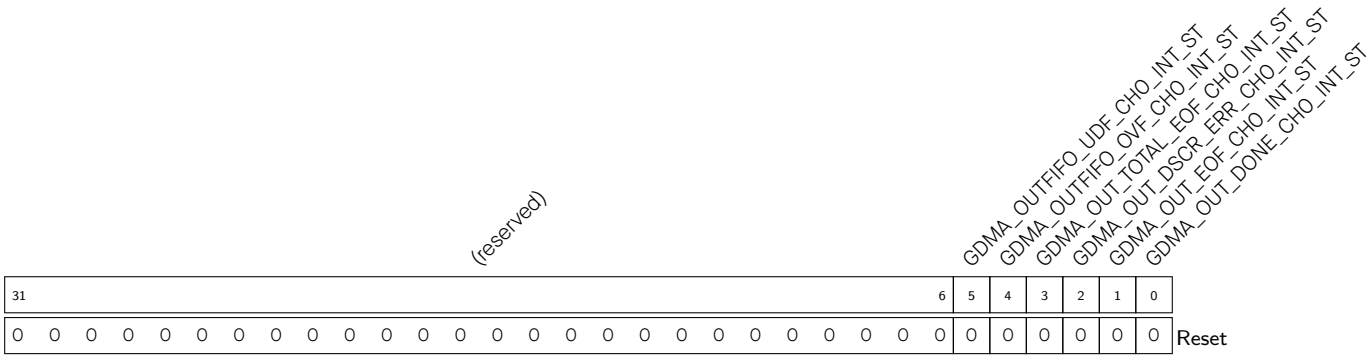
GDMA_OUT_DSCR_ERR_CH*n*_INT_RAW The raw interrupt status of GDMA_OUT_DSCR_ERR_CH*n*_INT. (R/WTC/SS)

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

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

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

Register 3.6. GDMA_OUT_INT_ST_CH*n*_REG (*n*: 0-2) (0x0034+0x10n*)**



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

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

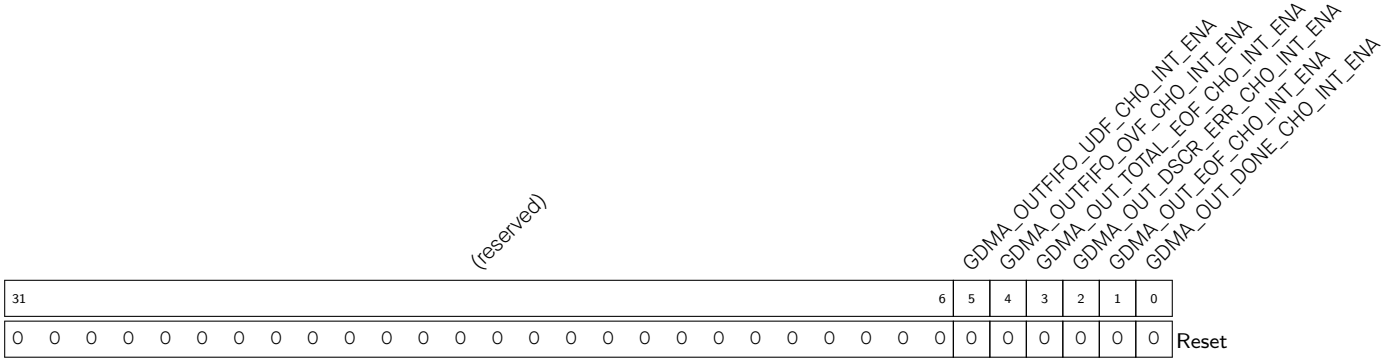
GDMA_OUT_DSCR_ERR_CH*n*_INT_ST The masked interrupt status of GDMA_OUT_DSCR_ERR_CH*n*_INT. (RO)

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

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

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

Register 3.7. GDMA_OUT_INT_ENA_CH*n*_REG (*n*: 0-2) (0x0038+0x10**n*)



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

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

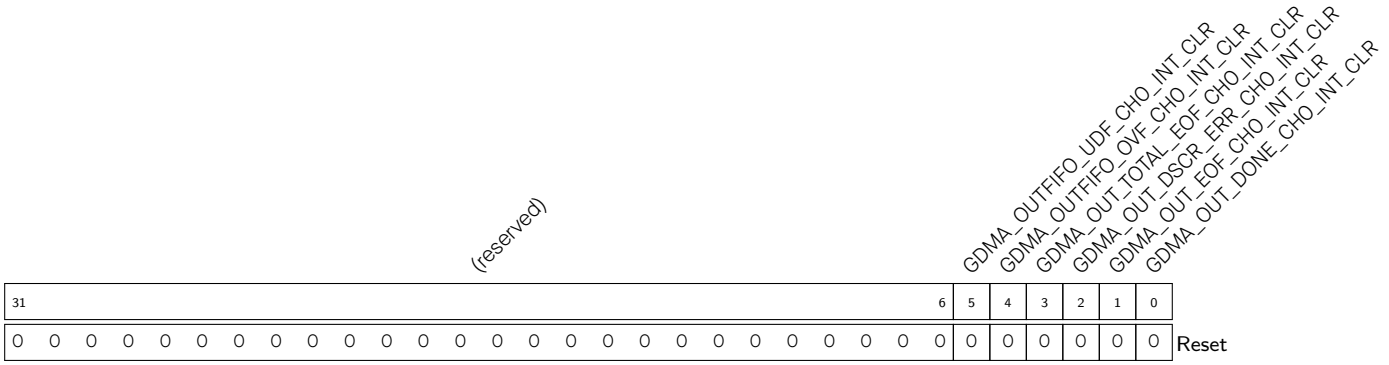
GDMA_OUT_DSCR_ERR_CH*n*_INT_ENA Write 1 to enable GDMA_OUT_DSCR_ERR_CH*n*_INT. (R/W)

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

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

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

Register 3.8. GDMA_OUT_INT_CLR_CH*n*_REG (*n*: 0-2) (0x003C+0x10**n*)



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

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

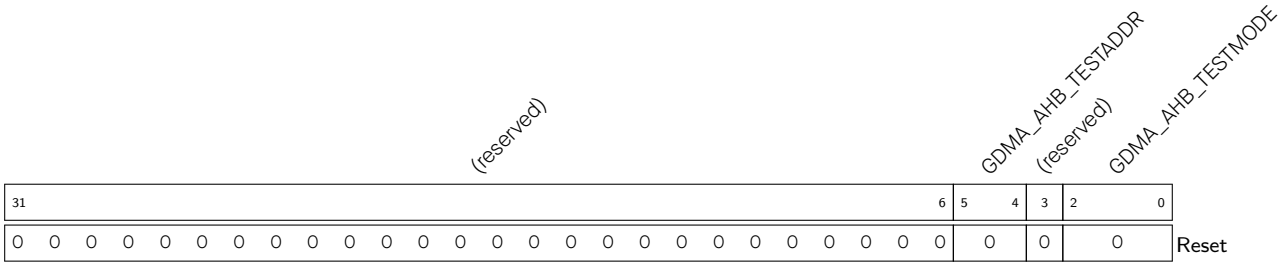
GDMA_OUT_DSCR_ERR_CH*n*_INT_CLR Write 1 to clear GDMA_OUT_DSCR_ERR_CH*n*_INT. (WT)

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

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

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

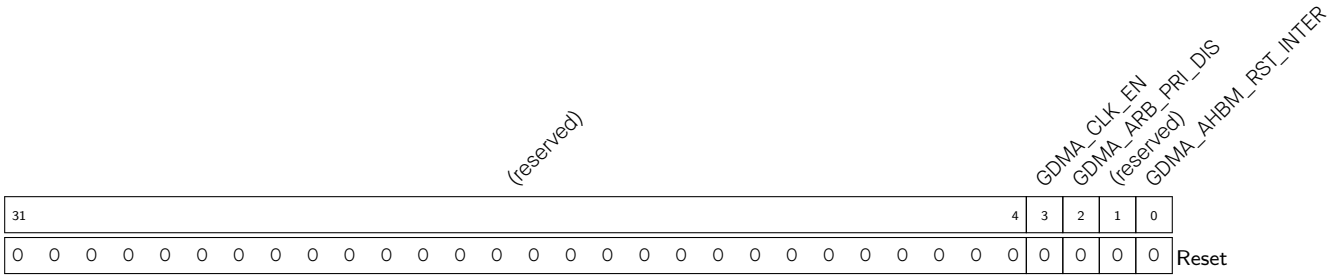
Register 3.9. GDMA_AHB_TEST_REG (0x0060)



GDMA_AHB_TESTMODE Reserved. (R/W)

GDMA_AHB_TESTADDR Reserved. (R/W)

Register 3.10. GDMA_MISC_CONF_REG (0x0064)



GDMA_AHBM_RST_INTER Write 1 and then 0 to reset the internal AHB FSM. (R/W)

GDMA_ARB_PRI_DIS Configures whether or not to disable the fixed-priority channel arbitration.

0: Enable

1: Disable

(R/W)

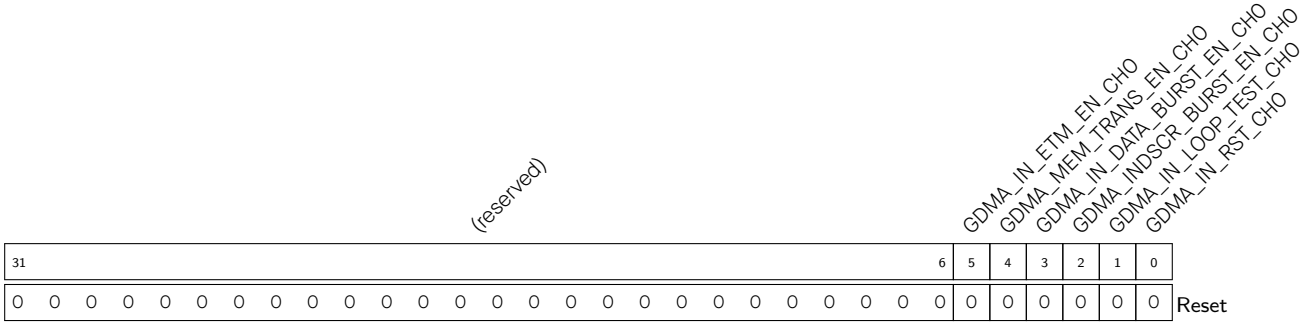
GDMA_CLK_EN Configures clock gating.

0: Support clock only when the application writes registers.

1: Always force the clock on for registers.

(R/W)

Register 3.11. GDMA_IN_CONF0_CH*n*_REG (*n*: 0-2) (0x0070+0xC0*n*)



GDMA_IN_RST_CH*n* Write 1 and then 0 to reset GDMA channel 0 RX FSM and RX FIFO pointer.(R/W)

GDMA_IN_LOOP_TEST_CH*n* Reserved. (R/W)

GDMA_INDSCR_BURST_EN_CH*n* Configures whether or not to enable INCR burst transfer for RX channel *n* to read descriptors.

0: Disable

1: Enable

(R/W)

GDMA_IN_DATA_BURST_EN_CH*n* Configures whether or not to enable INCR burst transfer for RX channel *n*.

0: Disable

1: Enable

(R/W)

GDMA_MEM_TRANS_EN_CH*n* Configures whether or not to enable memory-to-memory data transfer.

0: Disable

1: Enable

(R/W)

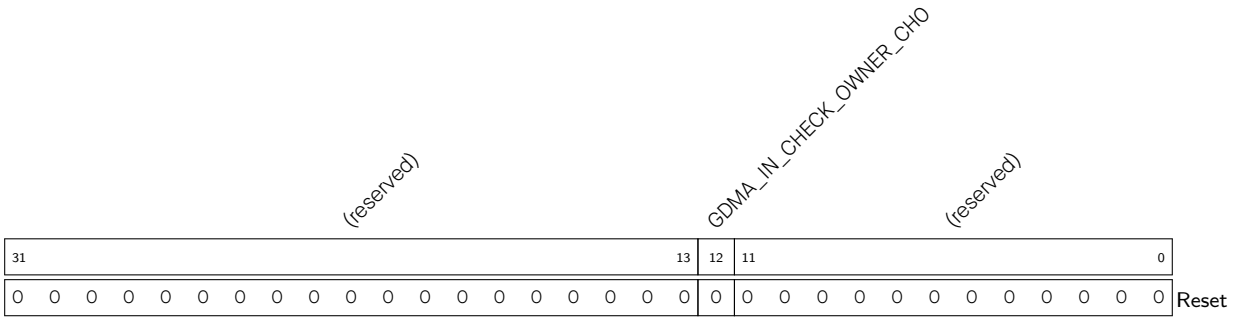
GDMA_IN_ETM_EN_CH*n* Configures whether or not to enable ETM control for RX channel*n*.

0: Disable

1: Enable

(R/W)

Register 3.12. GDMA_IN_CONF1_CH*n*_REG (*n*: 0-2) (0x0074+0xC0*n*)



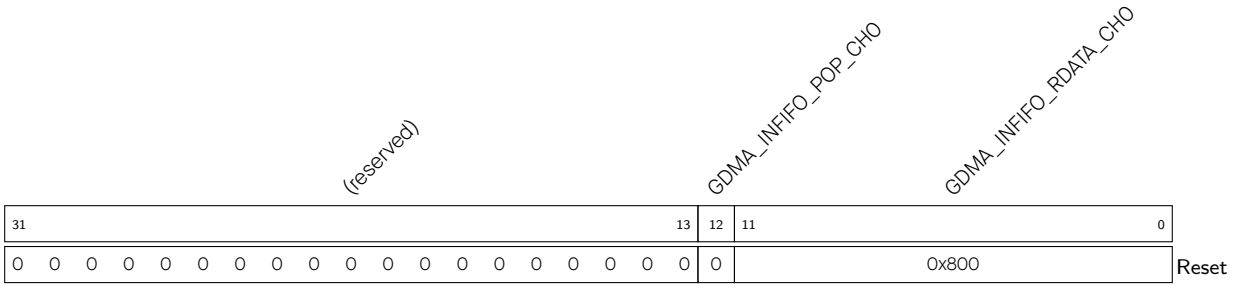
GDMA_IN_CHECK_OWNER_CH*n* Configures whether or not to enable owner bit check for RX channel *n*.

0: Disable

1: Enable

(R/W)

Register 3.13. GDMA_IN_POP_CH*n*_REG (*n*: 0-2) (0x007C+0xC0*n*)



GDMA_INFIFO_RDATA_CH*n* Represents the data popped from GDMA FIFO. (RO)

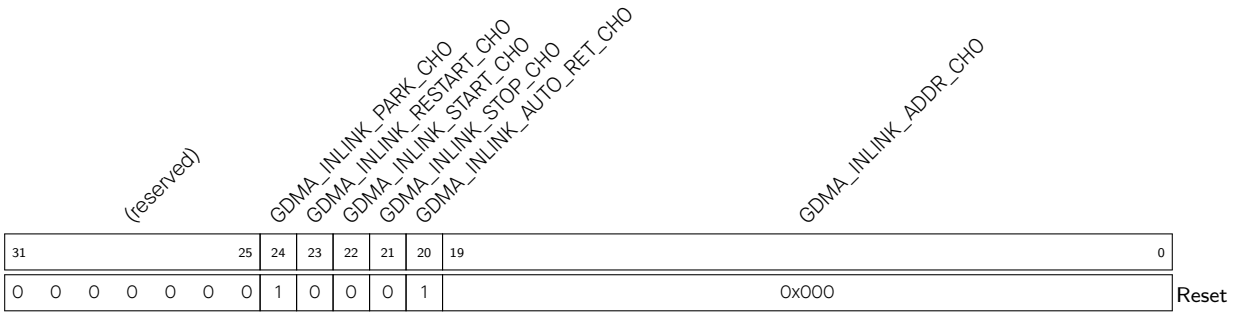
GDMA_INFIFO_POP_CH*n* Configures whether to pop data from GDMA FIFO.

0: Invalid. No effect

1: Pop

(WT)

Register 3.14. GDMA_IN_LINK_CH*n*_REG (*n*: 0-2) (0x0080+0xC0**n*)



GDMA_INLINK_ADDR_CH*n* Represents the lower 20 bits of the first receive descriptor's address. (R/W)

GDMA_INLINK_AUTO_RET_CH*n* Configures whether or not 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)

GDMA_INLINK_STOP_CH*n* Configures whether to stop GDMA's RX channel *n* from receiving data.

0: Invalid. No effect

1: Stop

(WT)

GDMA_INLINK_START_CH*n* Configures whether or not to enable GDMA's RX channel *n* for data transfer.

0: Disable

1: Enable

(WT)

GDMA_INLINK_RESTART_CH*n* Configures whether or not to restart RX channel *n* for GDMA transfer.

0: Invalid. No effect

1: Restart

(WT)

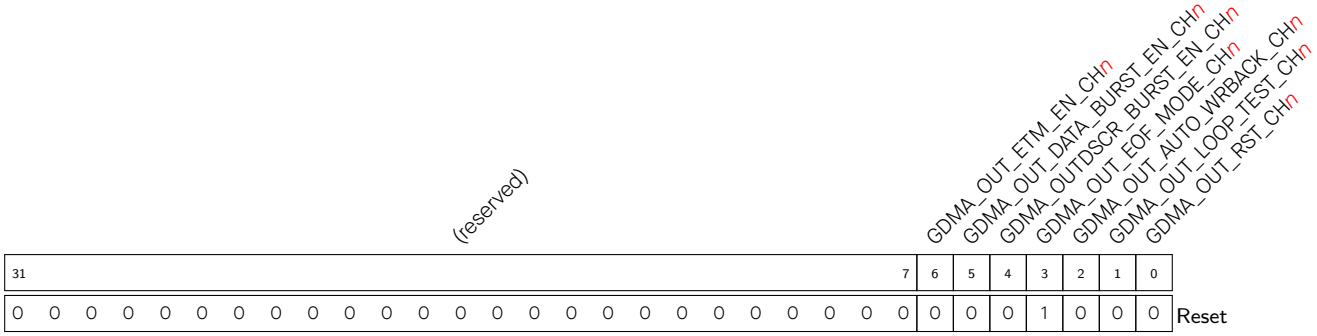
GDMA_INLINK_PARK_CH*n* Represents the status of the receive descriptor's FSM.

0: Running

1: Idle

(RO)

Register 3.15. GDMA_OUT_CONF0_CH*n*_REG (*n*: 0-2) (0x00D0+0xC0n*)**



GDMA_OUT_RST_CH*n* Configures the reset state of GDMA channel *n* TX FSM and TX FIFO pointer.

0: Release reset

1: Reset

(R/W)

GDMA_OUT_LOOP_TEST_CH*n* Reserved. (R/W)

GDMA_OUT_AUTO_WRBACK_CH*n* Configures whether or not to enable automatic outlink write-back when all the data in TX FIFO has been transmitted.

0: Disable

1: Enable

(R/W)

GDMA_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 GDMA.

1: EOF flag for TX channel *n* is generated when data to be transmitted has been popped from FIFO in GDMA.

(R/W)

GDMA_OUTDSCR_BURST_EN_CH*n* Configures whether or not to enable INCR burst transfer for TX channel *n* reading descriptors.

0: Disable

1: Enable

(R/W)

GDMA_OUT_DATA_BURST_EN_CH*n* Configures whether or not to enable INCR burst transfer for TX channel *n*.

0: Disable

1: Enable

(R/W)

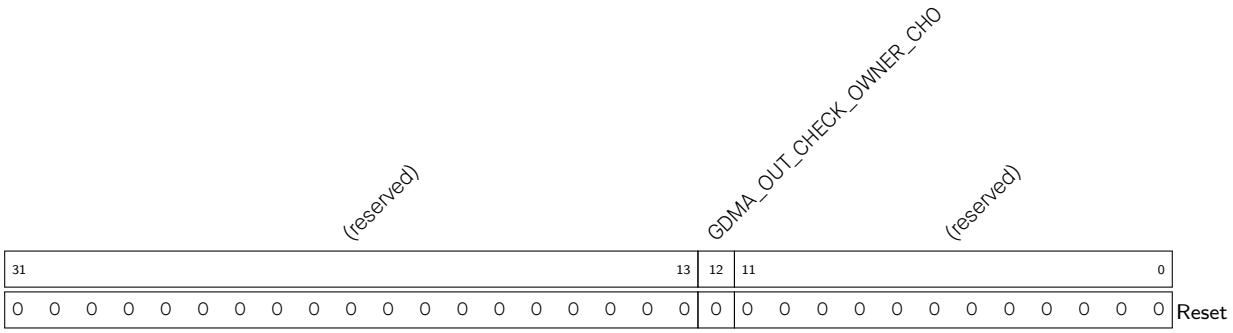
GDMA_OUT_ETM_EN_CH*n* Configures whether or not to enable ETM control for TX channel *n*.

0: Disable

1: Enable

((R/W)

Register 3.16. GDMA_OUT_CONF1_CH*n*_REG (*n*: 0-2) (0x00D4+0xC0**n*)



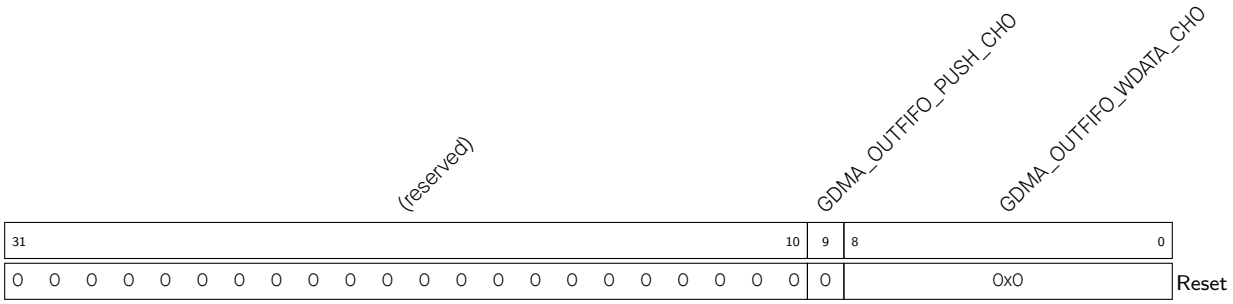
GDMA_OUT_CHECK_OWNER_CH*n* Configures whether or not to enable owner bit check for TX channel *n*.

0: Disable

1: Enable

(R/W)

Register 3.17. GDMA_OUT_PUSH_CH*n*_REG (*n*: 0-2) (0x00DC+0xC0**n*)



GDMA_OUTFIFO_WDATA_CH*n* Represents the data that need to be pushed into GDMA FIFO. (R/W)

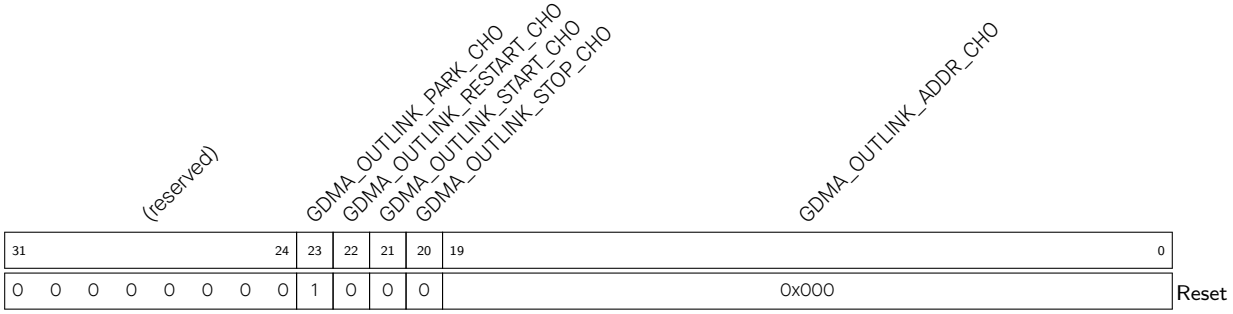
GDMA_OUTFIFO_PUSH_CH*n* Configures whether to push data into GDMA FIFO.

0: Invalid. No effect

1: Push

(WT)

Register 3.18. GDMA_OUT_LINK_CH*n*_REG (*n*: 0-2) (0x00E0+0xC0**n*)



GDMA_OUTLINK_ADDR_CH*n* Represents the lower 20 bits of the first transmit descriptor's address. (R/W)

GDMA_OUTLINK_STOP_CH*n* Configures whether to stop GDMA's TX channel *n* from transmitting data.

0: Invalid. No effect

1: Stop

(WT)

GDMA_OUTLINK_START_CH*n* Configures whether or not to enable GDMA's TX channel *n* for data transfer.

0: Disable

1: Enable

(WT)

GDMA_OUTLINK_RESTART_CH*n* Configures whether to restart TX channel *n* for GDMA transfer.

0: Invalid. No effect

1: Restart

(WT)

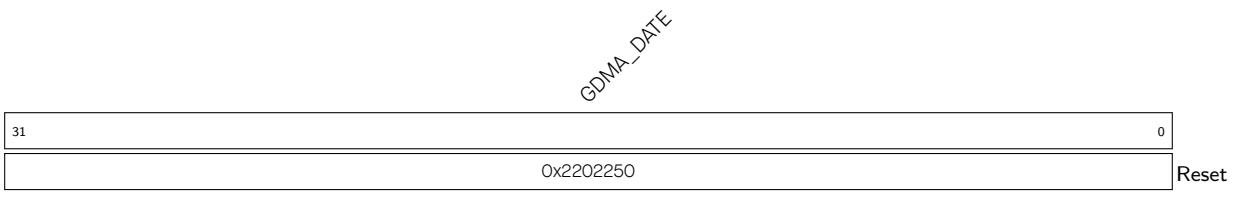
GDMA_OUTLINK_PARK_CH*n* Represents the status of the transmit descriptor's FSM.

0: Running

1: Idle

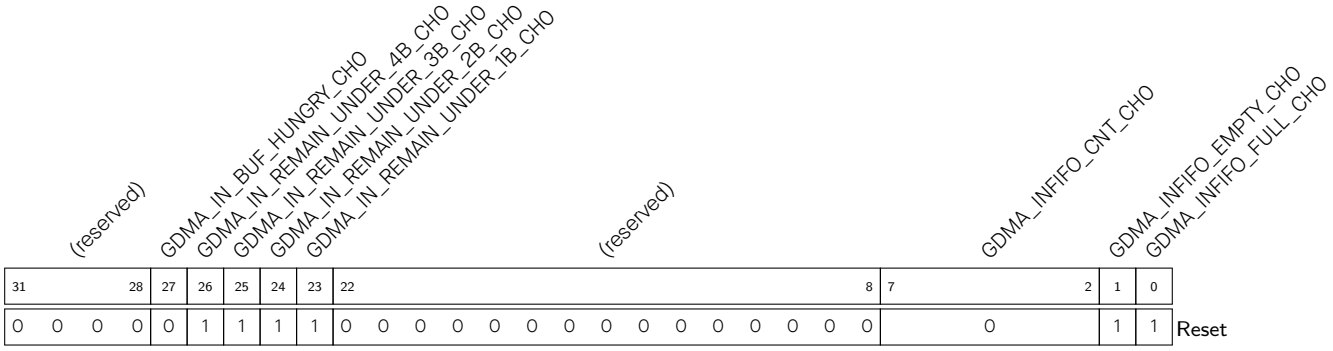
(RO)

Register 3.19. GDMA_DATE_REG (0x0068)



GDMA_DATE Version control register. (R/W)

Register 3.20. GDMA_INFIFO_STATUS_CH*n*_REG (*n*: 0-2) (0x0078+0xC0*n*)



GDMA_INFIFO_FULL_CH*n* Represents whether or not L1 RX FIFO is full.

0: Not Full

1: Full

(RO)

GDMA_INFIFO_EMPTY_CH*n* Represents whether or not L1 RX FIFO is empty.

0: Not empty

1: Empty

(RO)

GDMA_INFIFO_CNT_CH*n* Represents the number of data bytes in L1 RX FIFO for RX channel *n*. (RO)

GDMA_IN_REMAIN_UNDER_1B_CH*n* Reserved. (RO)

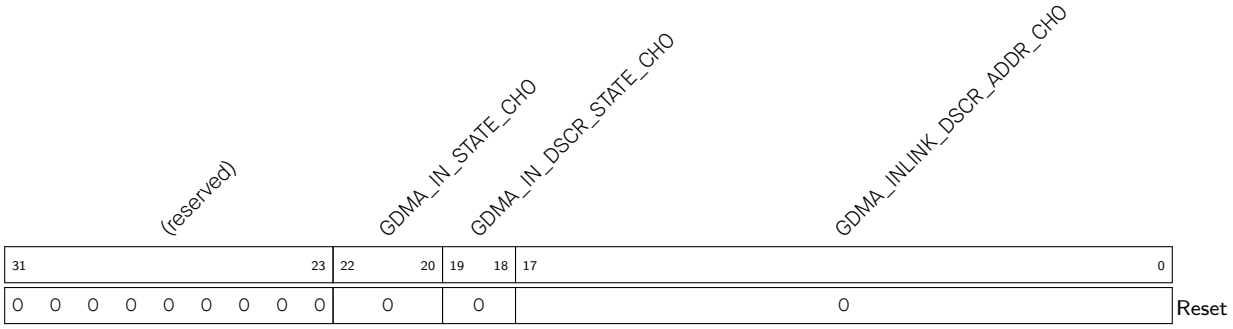
GDMA_IN_REMAIN_UNDER_2B_CH*n* Reserved. (RO)

GDMA_IN_REMAIN_UNDER_3B_CH*n* Reserved. (RO)

GDMA_IN_REMAIN_UNDER_4B_CH*n* Reserved. (RO)

GDMA_IN_BUF_HUNGRY_CH*n* Reserved. (RO)

Register 3.21. GDMA_IN_STATE_CH*n*_REG (*n*: 0-2) (0x0084+0xC0*n*)

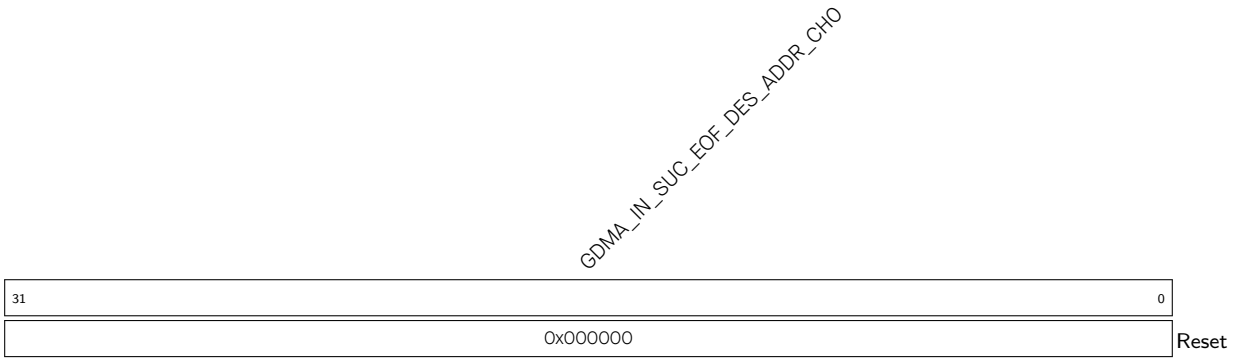


GDMA_INLINK_DSCR_ADDR_CH*n* Represents the lower 18 bits of the address of the next receive descriptor that is pre-read (but not processed yet). If the current receive descriptor is the last descriptor, then this field represents the address of the current receive descriptor. (RO)

GDMA_IN_DSCR_STATE_CH*n* Reserved. (RO)

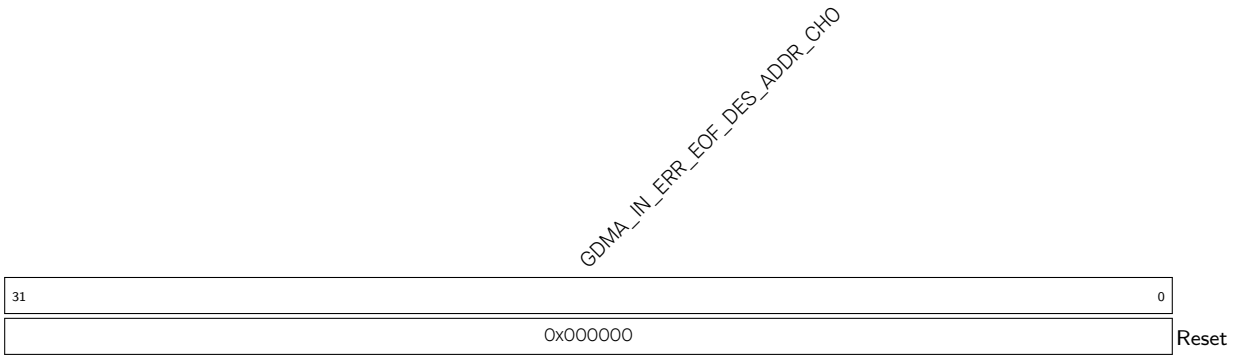
GDMA_IN_STATE_CH*n* Reserved. (RO)

Register 3.22. GDMA_IN_SUC_EOF_DES_ADDR_CH*n*_REG (*n*: 0-2) (0x0088+0xC0*n*)



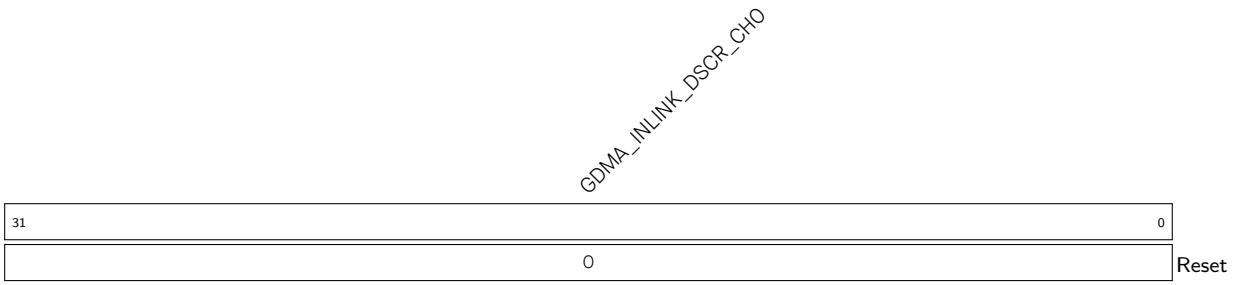
GDMA_IN_SUC_EOF_DES_ADDR_CH*n* Represents the address of the receive descriptor when the EOF bit in this descriptor is 1. (RO)

Register 3.23. GDMA_IN_ERR_EOF_DES_ADDR_CH*n*_REG (*n*: 0-2) (0x008C+0xC0*n*)



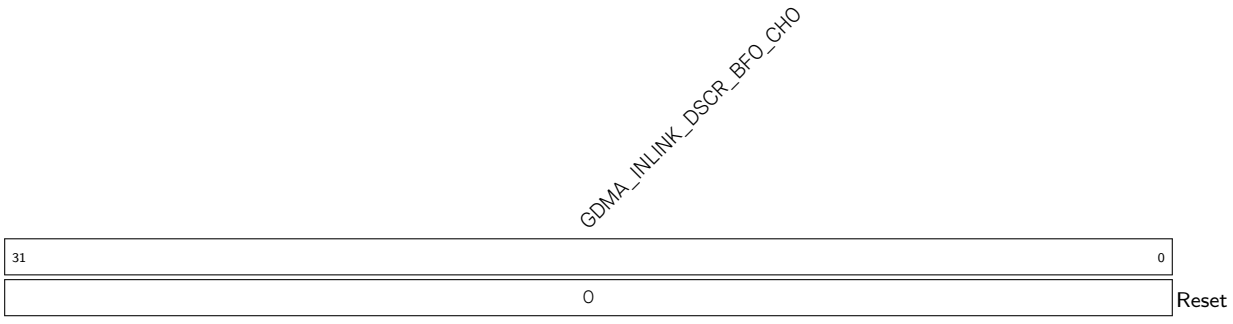
GDMA_IN_ERR_EOF_DES_ADDR_CH*n* Represents the address of the receive descriptor when there are some errors in the currently received data. Valid only for UHCI or PARLIO. (RO)

Register 3.24. GDMA_IN_DSCR_CH*n*_REG (*n*: 0-2) (0x0090+0xC0*n*)



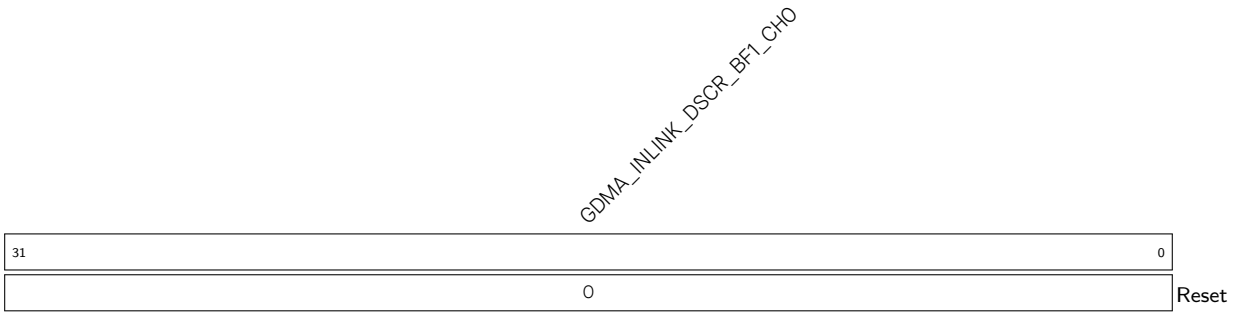
GDMA_INLINK_DSCR_CH*n* Represents the address of the next receive descriptor x+1 pointed by the current receive descriptor that is pre-read. (RO)

Register 3.25. GDMA_IN_DSCR_BF0_CH*n*_REG (*n*: 0-2) (0x0094+0xC0*n*)



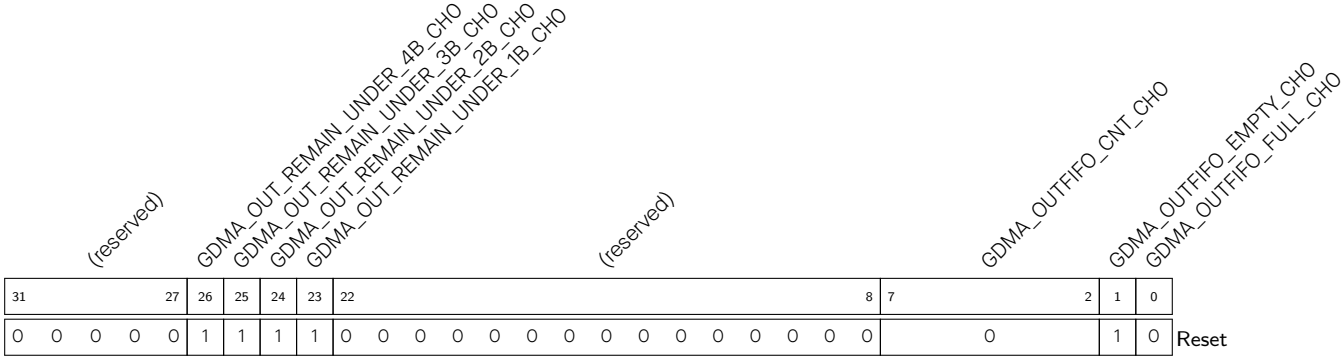
GDMA_INLINK_DSCR_BF0_CH*n* Represents the address of the current receive descriptor x that is pre-read. (RO)

Register 3.26. GDMA_IN_DSCR_BF1_CH*n*_REG (*n*: 0-2) (0x0098+0xC0*n*)



GDMA_INLINK_DSCR_BF1_CH*n* Represents the address of the previous receive descriptor x-1 that is pre-read. (RO)

Register 3.27. GDMA_OUTFIFO_STATUS_CH*n*_REG (*n*: 0-2) (0x00D8+0xC0*n*)



GDMA_OUTFIFO_FULL_CH*n* Represents whether or not L1 TX FIFO is full.

0: Not Full

1: Full

(RO)

GDMA_OUTFIFO_EMPTY_CH*n* Represents whether or not L1 TX FIFO is empty.

0: Not empty

1: Empty

(RO)

GDMA_OUTFIFO_CNT_CH*n* Represents the number of data bytes in L1 TX FIFO for TX channel *n*. (RO)

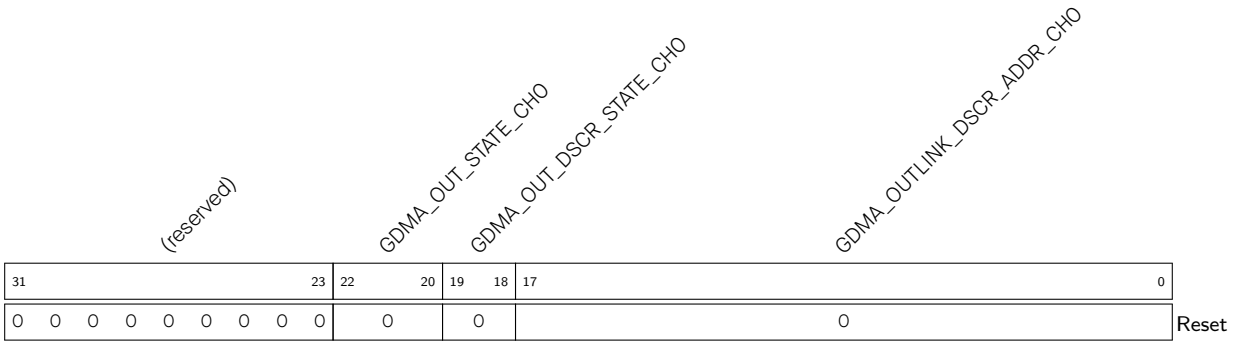
GDMA_OUT_REMAIN_UNDER_1B_CH*n* Reserved. (RO)

GDMA_OUT_REMAIN_UNDER_2B_CH*n* Reserved. (RO)

GDMA_OUT_REMAIN_UNDER_3B_CH*n* Reserved. (RO)

GDMA_OUT_REMAIN_UNDER_4B_CH*n* Reserved. (RO)

Register 3.28. GDMA_OUT_STATE_CH*n*_REG (*n*: 0-2) (0x00E4+0xC0**n*)

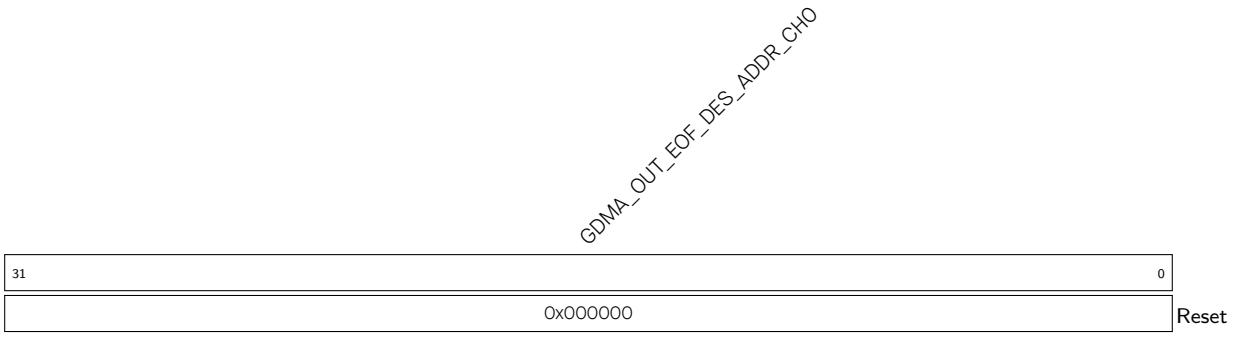


GDMA_OUTLINK_DSCR_ADDR_CH*n* Represents the lower 18 bits of the address of the next transmit descriptor that is pre-read (but not processed yet). If the current transmit descriptor is the last descriptor, then this field represents the address of the current transmit descriptor. (RO)

GDMA_OUT_DSCR_STATE_CH*n* Reserved. (RO)

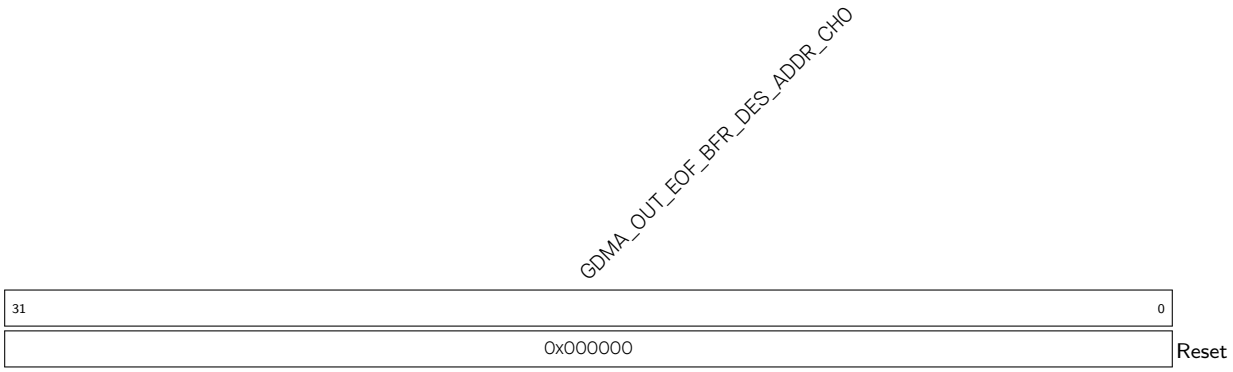
GDMA_OUT_STATE_CH*n* Reserved. (RO)

Register 3.29. GDMA_OUT_EOF_DES_ADDR_CH*n*_REG (*n*: 0-2) (0x00E8+0xC0**n*)



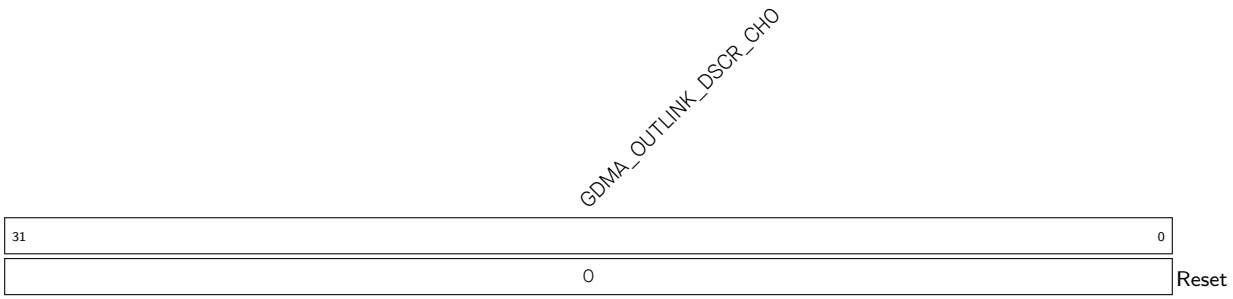
GDMA_OUT_EOF_DES_ADDR_CH*n* Represents the address of the transmit descriptor when the EOF bit in this descriptor is 1. (RO)

Register 3.30. GDMA_OUT_EOF_BFR_DES_ADDR_CH*n*_REG (*n*: 0-2) (0x00EC+0xC0*n*)



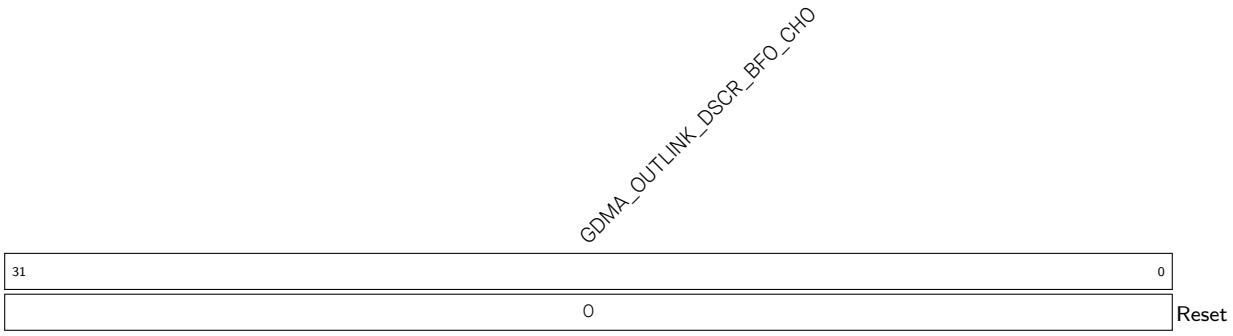
GDMA_OUT_EOF_BFR_DES_ADDR_CH*n* Represents the address of the transmit descriptor before the last transmit descriptor. (RO)

Register 3.31. GDMA_OUT_DSCR_CH*n*_REG (*n*: 0-2) (0x00F0+0xC0*n*)



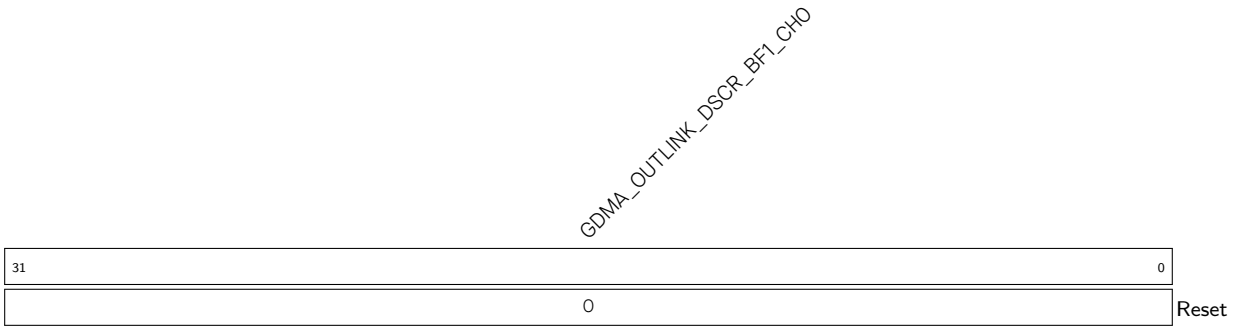
GDMA_OUTLINK_DSCR_CH*n* Represents the address of the next transmit descriptor y+1 pointed by the current transmit descriptor that is pre-read. (RO)

Register 3.32. GDMA_OUT_DSCR_BF0_CH*n*_REG (*n*: 0-2) (0x00F4+0xC0*n*)



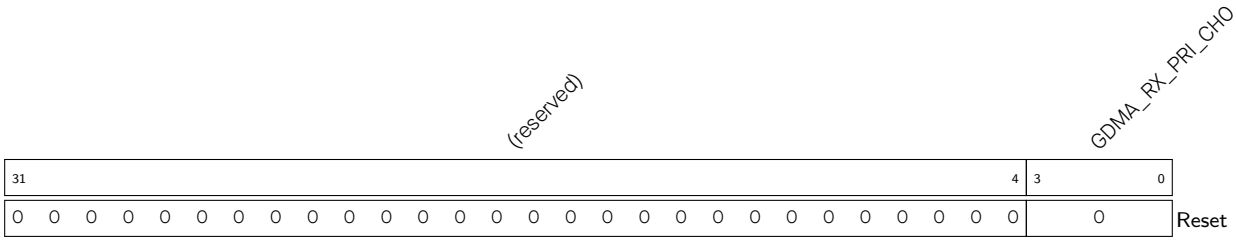
GDMA_OUTLINK_DSCR_BF0_CH*n* Represents the address of the current transmit descriptor y that is pre-read. (RO)

Register 3.33. GDMA_OUT_DSCR_BF1_CH*n*_REG (*n*: 0-2) (0x00F8+0xC0*n*)



GDMA_OUTLINK_DSCR_BF1_CH*n* Represents the address of the previous transmit descriptor y-1 that is pre-read. (RO)

Register 3.34. GDMA_IN_PRI_CH*n*_REG (*n*: 0-2) (0x009C+0xC0*n*)

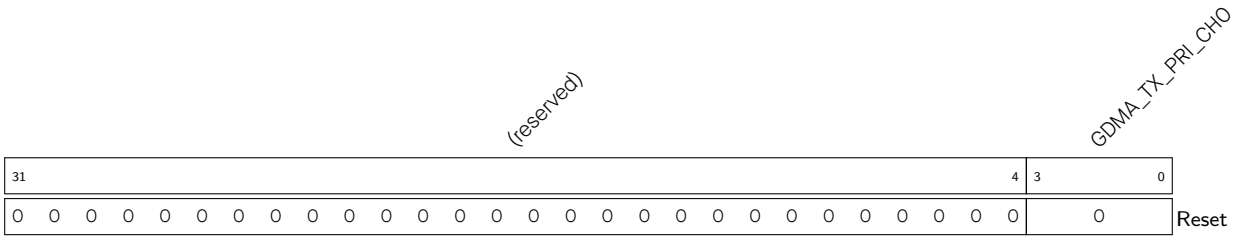


GDMA_RX_PRI_CH*n* Configures the priority of RX channel *n*.

Value range: 0 ~ 5

The larger of the value, the higher of the priority. (R/W)

Register 3.35. GDMA_OUT_PRI_CH*n*_REG (*n*: 0-2) (0x00FC+0xC0*n*)

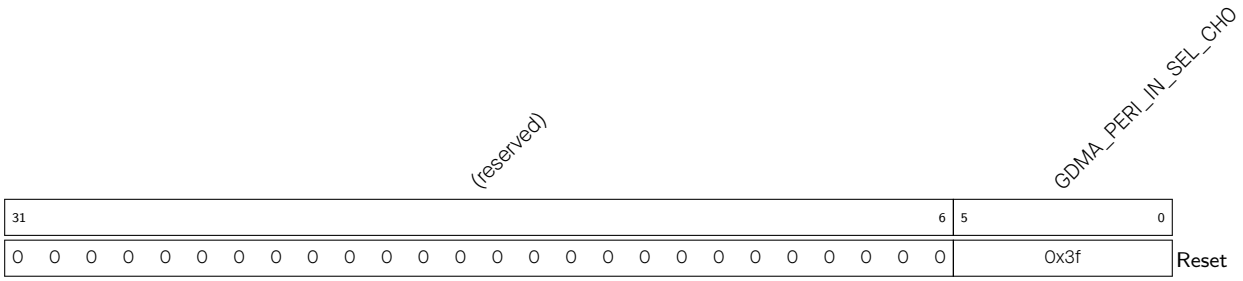


GDMA_TX_PRI_CH*n* Configures the priority of TX channel *n*.

Value range: 0 ~ 5

The larger of the value, the higher of the priority. (R/W)

Register 3.36. GDMA_IN_PERI_SEL_CH*n*_REG (*n*: 0-2) (0x00A0+0xC0*n*)



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

0: SPI2

1: Dummy-1

2: UHCI

3: I2S

4: Dummy-4

5: Dummy-5

6: AES

7: SHA

8: ADC

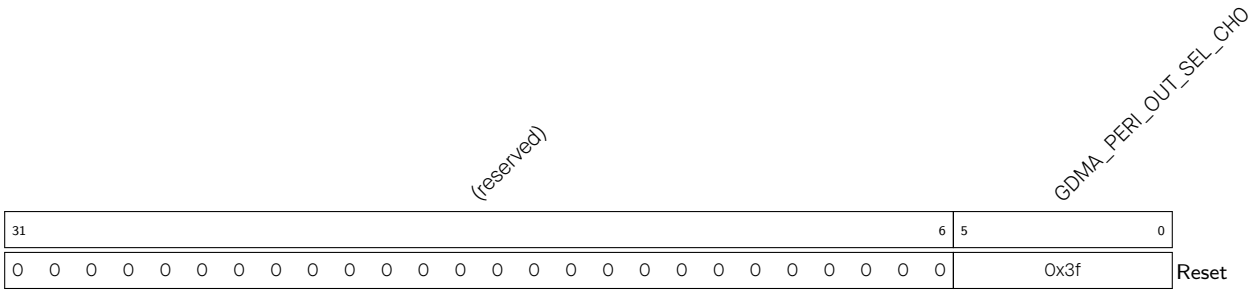
9: Parallel IO

10 ~ 15: Dummy-10 ~ 15

16 ~ 63: Invalid.

(R/W)

Register 3.37. GDMA_OUT_PERI_SEL_CH*n*_REG (*n*: 0-2) (0x0100+0xC0n*)**



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

0: SPI2

1: Dummy-1

2: UHCI

3: I2S

4: Dummy-4

5: Dummy-5

6: AES

7: SHA

8: ADC

9: Parallel IO

10 ~ 15: Dummy-10 ~ 15

16 ~ 63: Invalid.

(R/W)

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 4

System and Memory

4.1 Overview

ESP32-H2 is an ultra-low power and highly-integrated system that integrates a high-performance 32-bit RISC-V single-core processor (CPU), four-stage pipeline, clock frequency up to 96 MHz. All internal memory, external memory, and peripherals are located on the CPU bus.

4.2 Features

- **Address Space**
  - 452 KB of internal memory address space accessed from the instruction bus or data bus
  - 832 KB of peripheral address space
  - 16 MB of external memory virtual address space accessed from the instruction bus or data bus
  - 320 KB of internal DMA address space
- **Internal Memory**
  - 128 KB internal ROM
  - 320 KB HP SRAM
  - 4 KB LP SRAM
- **External Memory**
  - Supports up to 16 MB external flash
  - 16 KB of Cache
  - 32 bytes of Cache block size
- **Peripheral Space**
  - 46 modules/peripherals in total
- **GDMA**
  - 8 GDMA-supported modules/peripherals

Figure 4.2-1 illustrates the system structure and address mapping.

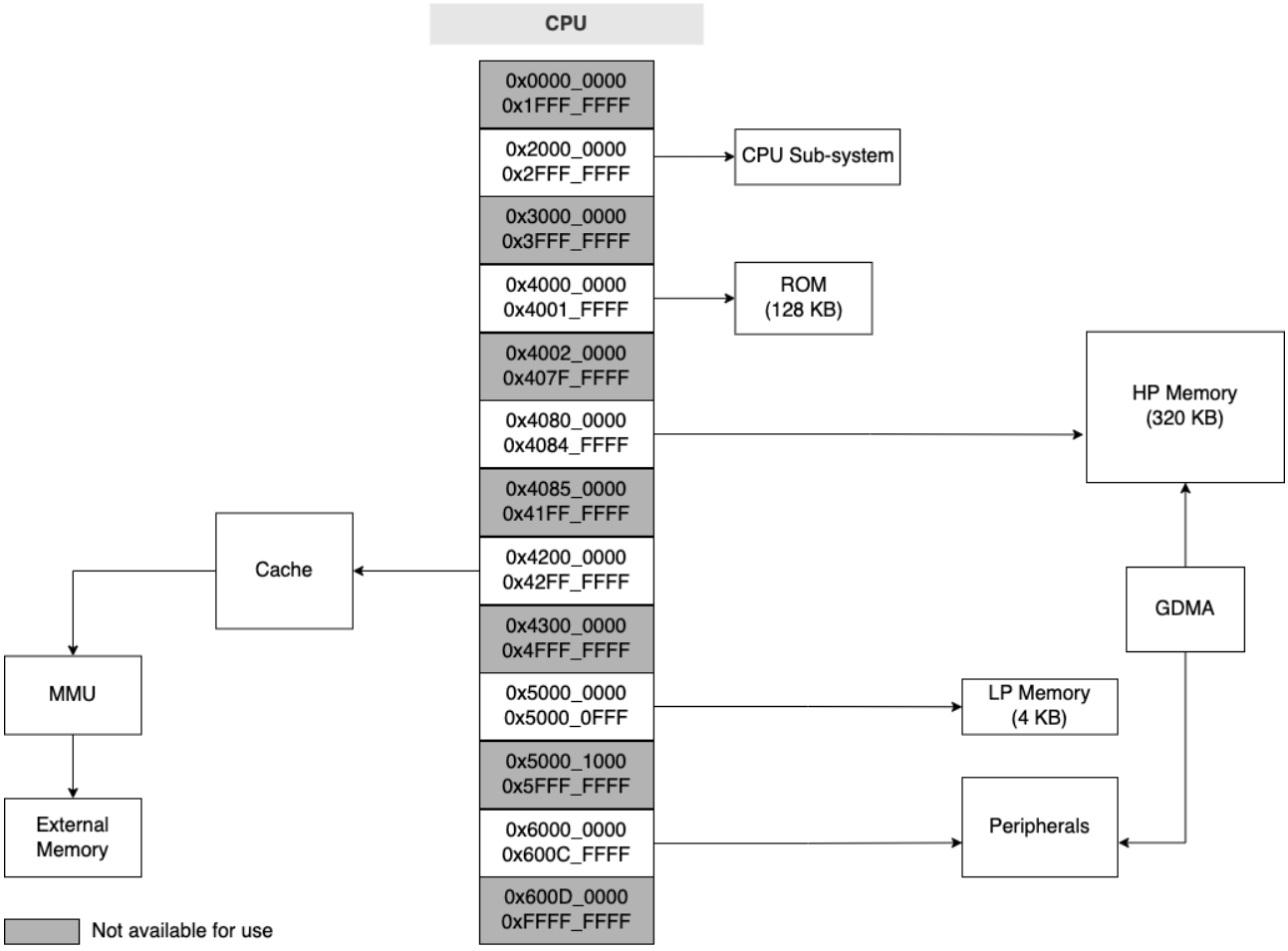


Figure 4.2-1. System Structure and Address Mapping

Note:

- The range of addresses available in the address space may be larger than the actual available memory of a particular type.
- For CPU Sub-system, please refer to Chapter 1 *ESP-RISC-V CPU*.

4.3 Functional Description

4.3.1 Address Mapping

All the non-reserved addresses are accessible by the instruction bus and data bus, that is, the instruction bus and the data bus access the same address space.

Both data bus and instruction bus of CPU are little-endian.

CPU can access data via the data bus using single-byte, double-byte, and 4-byte alignment.

The CPU can:

- directly access the internal memory via both data bus and instruction bus.
- directly access the external memory which is mapped into the address space via cache.

- directly access modules/peripherals via data bus.

Table 4.3-1 lists the address ranges on the data bus and instruction bus and their corresponding target memories.

Table 4.3-1. Memory Address Mapping

Bus Type	Boundary Address		Size	Target
	Low Address	High Address		
	0x0000_0000	0x1FFF_FFFF		Reserved
Data/Instruction bus	0x2000_0000	0x2FFF_FFFF	256 MB	CPU Sub-system
	0x3000_0000	0x3FFF_FFFF		Reserved
Data/Instruction bus	0x4000_0000	0x4001_FFFF	128 KB	ROM*
	0x4002_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	0x42FF_FFFF	16 MB	External memory
	0x4300_0000	0x4FFF_FFFF		Reserved
Data/Instruction bus	0x5000_0000	0x5000_0FFF	4 KB	LP SRAM*
	0x5000_1000	0x5FFF_FFFF		Reserved
Data/Instruction 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 CPU. For more information about Permission Control, please refer to Chapter 15 *Permission Control (PMS)*.

4.3.2 Internal Memory

ESP32-H2 consists of the following three types of internal memory:

- ROM (128 KB): The ROM is a read-only memory and can not be programmed. It contains the ROM code of some low-level system software and read-only data.
- HP SRAM (320 KB): The HP SRAM is a volatile memory that can be quickly accessed by CPU (generally within a single clock cycle).
- LP SRAM (4 KB): LP SRAM is also a volatile memory, however, in Deep-sleep mode, data stored in the LP SRAM will not be lost. The LP SRAM can be accessed by CPU and is usually used to store program instructions and data that need to be kept in sleep mode.

1. ROM

This 128 KB ROM is a read-only memory, addressed by CPU through the instruction bus or through the data bus via 0x4000_0000 ~ 0x4001_FFFF, as shown in Table 4.3-1.

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 as shown in Table 4.3-1.

3. LP SRAM

This 4 KB LP SRAM is a read-and-write memory, accessed by the CPU through the instruction bus or through the data bus via their shared address 0x5000_0000 ~ 0x5000_0FFF as shown in Table 4.3-1.

4.3.3 External Memory

ESP32-H2 supports SPI, Dual SPI, Quad SPI, and QPI interfaces that allow connection to external flash. ESP32-H2 also supports hardware manual encryption and automatic decryption based on XTS-AES algorithm to protect users' programs and data in the external flash.

4.3.3.1 External Memory Address Mapping

The CPU accesses the external memory via the cache. According to information inside the MMU (Memory Management Unit), the cache maps the CPU's address (0x4200_0000 ~ 0x42FF_FFFF) into a physical address of the external memory. Due to this address mapping, ESP32-H2 can address up to 16 MB external flash. Note that the instruction bus shares the same address space (16 MB) with the data bus to access the external memory.

4.3.3.2 Cache

As shown in Figure 4.3-1, ESP32-H2 has a read-only uniform cache which is eight-way set-associative. Its size is 16 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. When a cache miss occurs, the cache controller will initiate a request to the external memory.

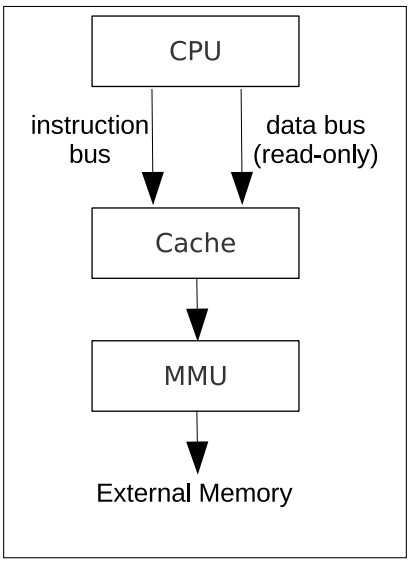


Figure 4.3-1. Cache Structure

4.3.3.3 Cache Operations

ESP32-H2 cache supports the following operations:

1. **Invalidate**: This operation is used to remove valid data in the cache. Once this operation is done, the deleted data is stored only in the external memory. If the CPU wants to access the data again, it needs to access the external memory. There are two types of invalidate operation: Invalidate-All and

Manual-Invalidate. Manual-Invalidate is performed only on data in the specified area in the cache, while Invalidate-All is performed on all data in the cache.

2. **Preload**: This operation is to load instructions and data into the cache in advance. The minimum unit of preload-operation is one block. There are two types of preload-operation: manual preload (Manual-Preload) and automatic preload (Auto-Preload). Manual-Preload means that the hardware prefetches a piece of continuous data according to the virtual address specified by the software. Auto-Preload means the hardware prefetches a piece of continuous data according to the current address where the cache hits or misses (depending on configuration).
3. **Lock/Unlock**: The lock operation is used to prevent the data in the cache from being easily replaced. There are two types of lock: prelock and manual lock. When prelock is enabled, the cache locks the data in the specified area when filling the missing data to cache memory, while the data outside the specified area will not be locked. When manual lock is enabled, the cache checks the data that is already in the cache memory and locks the data only if it falls in the specified area, and leaves the data outside the specified area unlocked. When there are missing data, the cache will replace the data in the unlocked way first, so the data in the locked way is always stored in the cache and will not be replaced. But 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, except that it only can be done manually.

   Please note that Manual-Invalidate operation only works on the unlocked data. If you expect to perform such operation on the locked data, please unlock them first.

4.3.4 GDMA Address Space

The General Direct Memory Access (GDMA) peripheral consisting of three TX channels and three 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

GDMA uses the same addresses as the data bus to access HP SRAM, i.e., GDMA uses address range 0x4080_0000 ~ 0x4084_FFFF to access HP SRAM.

Eight modules/peripherals in ESP32-H2 work together with GDMA. As shown in Figure 4.3-2, eight vertical lines correspond to these eight 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. If there are multiple intersections on the same line, it means that these peripherals/modules can not enable the GDMA function at the same time.

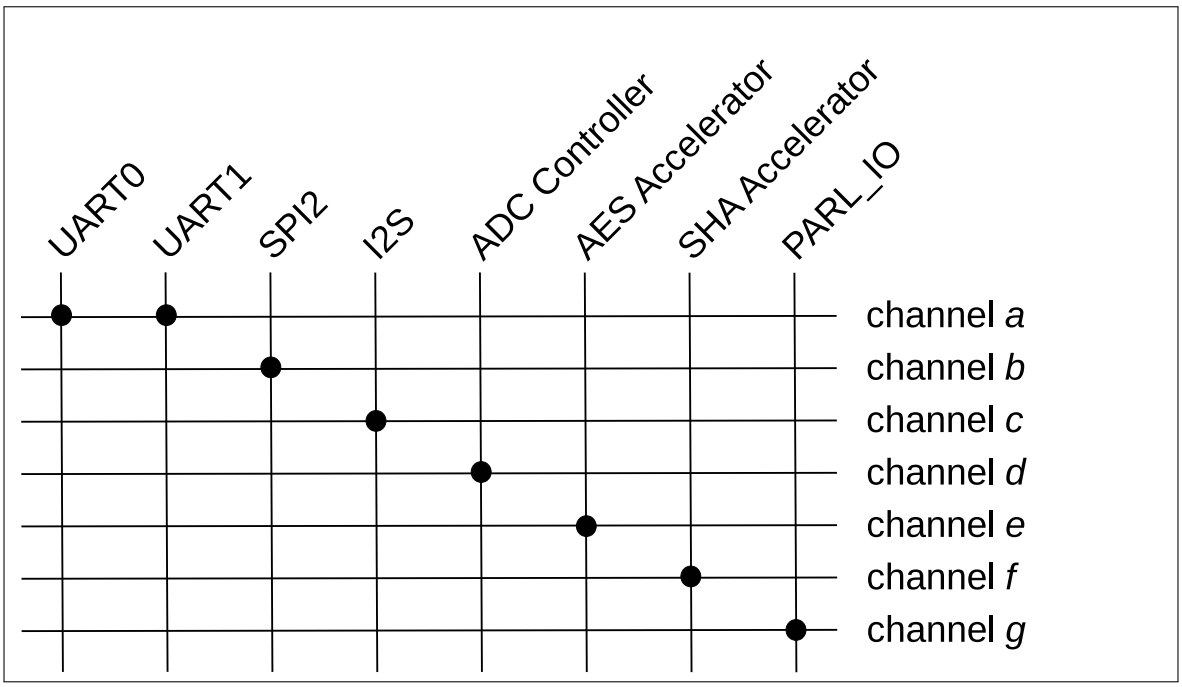


Figure 4.3-2. Modules/peripherals that can work with GDMA

These modules/peripherals can access any memory available to GDMA. For more information, please refer to Chapter 3 *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 15 *Permission Control (PMS)*.

4.3.5 Modules/Peripherals Address Mapping

Table 4.3-2 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 4.3-2. 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)	0x6000_2000	0x6000_2FFF	4
Reserved	0x6000_3000	0x6000_3FFF	
I2C Controller 0 (I2C0)	0x6000_4000	0x6000_4FFF	4
I2C Controller 1 (I2C1)	0x6000_5000	0x6000_5FFF	4
UHCI Controller (UHCI)	0x6000_6000	0x6000_6FFF	4

Cont'd on next page

Table 4.3-2 – cont'd from previous page

Target	Boundary Address		Size (KB)
	Low Address	High Address	
Remote Control Peripheral (RMT)	0x6000_7000	0x6000_7FFF	4
LED PWM Controller (LEDC)	0x6000_8000	0x6000_8FFF	4
Timer Group 0 (TIMG0)	0x6000_9000	0x6000_9FFF	4
Timer Group 1 (TIMG1)	0x6000_A000	0x6000_AFFF	4
System Timer (SYSTIMER)	0x6000_B000	0x6000_BFFF	4
Two-wire Automotive Interface (TWAI)	0x6000_C000	0x6000_CFFF	4
I2S Controller (I2S)	0x6000_D000	0x6000_DFFF	4
ADC Controller (ADC), Temperature Sensor (TSENS)	0x6000_E000	0x6000_EFFF	4
USB Serial/JTAG Controller	0x6000_F000	0x6000_FFFF	4
Interrupt Matrix (INTMTX)	0x6001_0000	0x6001_0FFF	4
Reserved	0x6001_1000	0x6001_1FFF	
Pulse Count Controller (PCNT)	0x6001_2000	0x6001_2FFF	4
Event Task Matrix (SOC_ETM)	0x6001_3000	0x6001_3FFF	4
Motor Control PWM (MCPWM)	0x6001_4000	0x6001_4FFF	4
Parallel IO Controller (PARLIO)	0x6001_5000	0x6001_5FFF	4
Reserved	0x6001_6000	0x6007_FFFF	
GDMA Controller (GDMA)	0x6008_0000	0x6008_0FFF	4
General Purpose SPI2 (GP-SPI2)	0x6008_1000	0x6008_1FFF	4
Reserved	0x6008_2000	0x6008_7FFF	
AES Accelerator (AES)	0x6008_8000	0x6008_8FFF	4
SHA Accelerator (SHA)	0x6008_9000	0x6008_9FFF	4
RSA Accelerator (RSA)	0x6008_A000	0x6008_AFFF	4
ECC Accelerator (ECC)	0x6008_B000	0x6008_BFFF	4
RSA Digital Signature Peripheral (RSA_DS)	0x6008_C000	0x6008_CFFF	4
HMAC Accelerator (HMAC)	0x6008_D000	0x6008_DFFF	4
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 Assess Monitor (MEM_MONITOR)*	0x6009_2000	0x6009_2FFF	4
Reserved	0x6009_3000	0x6009_4FFF	
HP System Register (HP_SYSREG)	0x6009_5000	0x6009_5FFF	4
Power/Clock/Reset (PCR) Register	0x6009_6000	0x6009_6FFF	4
Reserved	0x6009_7000	0x6009_7FFF	
Trusted Execution Environment (TEE) Register*	0x6009_8000	0x6009_8FFF	4
Access Permission Management Controller (HP_APM)*	0x6009_9000	0x6009_9FFF	4
Reserved	0x6009_A000	0x600A_FFFF	

Cont'd on next page

Table 4.3-2 – cont'd from previous page

Target	Boundary Address		Size (KB)
	Low Address	High Address	
Power Management Unit (PMU)	0x600B_0000	0x600B_03FF	1
Low-power Clock/Reset Register (LP_CLKRST)	0x600B_0400	0x600B_07FF	1
eFuse Controller (EFUSE)	0x600B_0800	0x600B_0BFF	1
Low-power Timer (LP_TIMER)	0x600B_0C00	0x600B_0FFF	1
Low-power Always-on Register (LP_AON)	0x600B_1000	0x600B_13FF	1
Reserved	0x600B_1400	0x600B_1BFF	
Low-power Watch Dog Timer (LP_WDT)	0x600B_1C00	0x600B_1FFF	1
Reserved	0x600B_2000	0x600B_27FF	
Low-power Peripheral (LPPERI)	0x600B_2800	0x600B_2BFF	1
Low-power Analog Peripheral (LP_ANA_PERI)	0x600B_2C00	0x600B_2FFF	1
Reserved	0x600B_3000	0x600B_37FF	
Low-power Access Permission Management (LP_APM)*	0x600B_3800	0x600B_3BFF	1
Reserved	0x600B_3C00	0x600B_FFFF	
RISC-V Trace Encoder (TRACE)	0x600C_0000	0x600C_0FFF	4
Reserved	0x600C_1000	0x600C_1FFF	
DEBUG ASSIST (ASSIST_DEBUG)*	0x600C_2000	0x600C_2FFF	4
Reserved	0x600C_3000	0x600C_4FFF	
Interrupt Priority Register (INTPRI)	0x600C_5000	0x600C_5FFF	4
Reserved	0x600C_6000	0x600C_FFFF	

* The address space of this module/peripheral is not continuous.

Chapter 5

eFuse Controller (EFUSE)

5.1 Overview

ESP32-H2 contains a 4096-bit eFuse memory to store user data and hardware parameters, including control parameters for hardware modules, calibration parameters, the MAC address, and keys used for the encryption and decryption module. Once an eFuse bit is programmed to 1, it can never be reverted to 0. Users cannot directly access the eFuse memory. They can only use the eFuse controller to read and write eFuse memory bits. For confidential data in eFuse memory, read protection can be enabled by programming the corresponding read protection bit. So, the data cannot be accessed via the controller.

5.2 Features

- 4096-bit one-time-programmable memory including 1792 bits reserved for custom use
- Configurable write protection
- Configurable read protection
- Various hardware encoding schemes against data corruption

5.3 Functional Description

5.3.1 Structure

The eFuse system consists of the eFuse controller and eFuse memory. Data flow in this system is shown in Figure 5.3-1.

To program data to the eFuse memory, write the data to the programming register first and then execute the programming instruction. For detailed programming steps, please refer to Section 5.3.2.

Users cannot directly read data from the eFuse memory. They need to use the eFuse controller to take the data into the reading data register of the corresponding address segment. 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 5.3.1.4 for details), and send the error message to the error report register. For detailed steps to read parameters, please refer to Section 5.3.3.

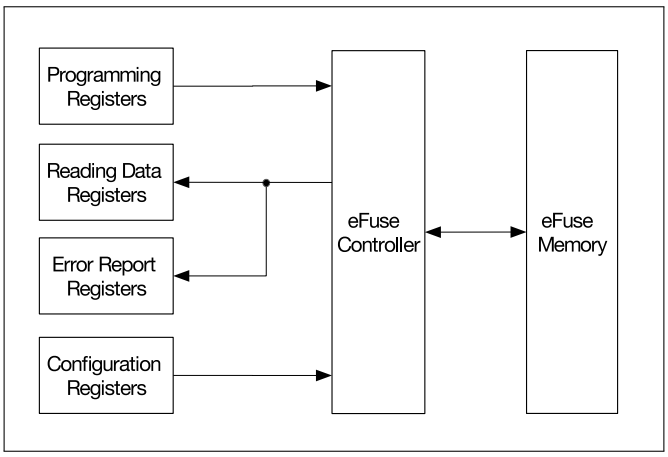


Figure 5.3-1. Data Flow in eFuse

Data in eFuse memory is organized in 11 blocks (BLOCK0 ~ BLOCK10).

BLOCK0 holds most parameters for software and hardware uses.

Table 5.3-1 lists all the parameters accessible (readable and usable) to users in BLOCK0, their offsets, bit widths, whether they directly drive hardware modules, write protection, and brief function description. For more description on the parameters, please click the corresponding link in the table.

The **EFUSE_WR_DIS** parameter is used to disable write protection of other parameters. **EFUSE_RD_DIS** is used to disable read protection of BLOCK4 ~ BLOCK10. For more information, please see Section 5.3.1.2 and Section 5.3.1.3.

Table 5.3-1. Parameters in eFuse BLOCK0

Parameters	Bit Width	Hardware Use	Write Protection by EFUSE_WR_DIS Bit Number	Description
EFUSE_WR_DIS	32	Y	N/A	Represents whether writing of eFuse bits by eFuse controller is disabled.
EFUSE_RD_DIS	7	Y	0	Represents whether reading data from BLOCK4 ~ 10 in eFuse memory by users is disabled.
EFUSE_DIS_ICACHE	1	Y	2	Represents whether instruction cache is disabled.
EFUSE_DIS_USB_JTAG	1	Y	2	Represents whether the USB-to-JTAG function in the USB module is disabled.
EFUSE_POWERGLITCH_EN	1	Y	2	Represents whether to enable the power glitch detection function.
EFUSE_DIS_FORCE_DOWNLOAD	1	Y	2	Represents whether the function to force the chip into Download mode is disabled.
EFUSE_SPI_DOWNLOAD_MSPI_DIS	1	Y	2	Represents whether the SPI0 controller is disabled in boot_mode_download.
EFUSE_DIS_CAN	1	Y	2	Represents whether the TWAI controller is disabled.
EFUSE_JTAG_SEL_ENABLE	1	Y	2	Represents whether the selection of a JTAG signal source through the strapping value of GPIO15 is enabled when both EFUSE_DIS_PAD_JTAG and EFUSE_DIS_USB_JTAG are configured to 0.
EFUSE_SOFT_DIS_JTAG	3	Y	31	Represents whether JTAG is disabled in the soft way.
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 encryption is disabled (except in SPI boot mode).
EFUSE_USB_EXCHG_PINS	1	Y	30	Represents whether the D+ and D- pins are exchanged.

Cont'd on next page

GoBack

Table 5.3-1 – cont'd from previous page

Parameters	Bit Width	Hardware Use	Write Protection by EFUSE_WR_DIS Bit Number	Description
EFUSE_VDD_SPI_AS_GPIO	1	Y	30	Represents whether the VDD_SPI pin is used as a regular GPIO.
EFUSE_ECDSA_CURVE_MODE	2	Y	1	Configures the curve of ECDSA_DS calculation.
EFUSE_ECC_FORCE_CONST_TIME	1	Y	1	Set this bit to permanently turn on ECC const-time mode.
EFUSE_XTS_DPA_PSEUDO_LEVEL	2	Y	1	Set this bit to control the xts pseudo-round anti-dpa attack function.
EFUSE_WDT_DELAY_SEL	2	Y	3	Selects RTC watchdog timeout threshold at startup.
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 the first Secure Boot key is enabled.
EFUSE_SECURE_BOOT_KEY_REVOKE1	1	N	6	Represents whether revoking the second Secure Boot key is enabled.
EFUSE_SECURE_BOOT_KEY_REVOKE2	1	N	7	Represents whether revoking the third Secure Boot key is enabled.
EFUSE_KEY_PURPOSE_0	4	Y	8	Represents Key0 purpose. See Table 5.3-2.
EFUSE_KEY_PURPOSE_1	4	Y	9	Represents Key1 purpose. See Table 5.3-2.
EFUSE_KEY_PURPOSE_2	4	Y	10	Represents Key2 purpose. See Table 5.3-2.
EFUSE_KEY_PURPOSE_3	4	Y	11	Represents Key3 purpose. See Table 5.3-2.
EFUSE_KEY_PURPOSE_4	4	Y	12	Represents Key4 purpose. See Table 5.3-2.
EFUSE_KEY_PURPOSE_5	4	Y	13	Represents Key5 purpose. See Table 5.3-2.
EFUSE_SEC_DPA_LEVEL	2	Y	14	Represents the security level of anti-DPA (differential power analysis) attack.
EFUSE_CRYPT_DPA_ENABLE	1	Y	14	Represents whether defense against DPA attack is enabled.
EFUSE_SECURE_BOOT_EN	1	N	15	Represents whether Secure Boot is enabled or disabled.

Cont'd on next page

Table 5.3-1 – cont'd from previous page

Parameters	Bit Width	Hardware Use	Write Protection by EFUSE_WR_DIS Bit Number	Description
EFUSE_SECURE_BOOT_AGGRESSIVE_REVOKE	1	N	16	Represents whether aggressive revocation of Secure Boot is enabled.
EFUSE_POWERGLITCH_EN1	5	N	1	Set these bits to enable power glitch function when chip power on.
EFUSE_FLASH_TPUW	4	N	18	Configures flash waiting time after power-up, in unit of ms.
EFUSE_DIS_DOWNLOAD_MODE	1	N	18	Represents whether all download modes are disabled.
EFUSE_DIS_DIRECT_BOOT	1	N	18	Represents whether direct boot mode is disabled.
EFUSE_DIS_USB_PRINT	1	N	18	Represents whether print from USB-Serial-JTAG during ROM boot is disabled.
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 type 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 version used by ESP-IDF anti-rollback feature.
EFUSE_SECURE_BOOT_DISABLE_FAST_WAKE	1	N	18	Represents whether FAST VERIFY ON WAKE is disabled or enabled when Secure Boot is enabled.
EFUSE_HYS_EN_PAD0	6	Y	19	Represents whether to enable the hysteresis function of pad 0-5.
EFUSE_HYS_EN_PAD1	22	Y	19	Represents whether to enable the hysteresis function of pad 6-27.

GoBack

Table 5.3-2 lists all key purposes and their values. Set EFUSE_KEY_PURPOSE_*n* to declare the purpose of KEY*n* (*n*: 0 ~ 5).

Table 5.3-2. Secure Key Purpose Values

Key Purpose Values	Purposes
0	User purposes
1	ECDSA_KEY
2	Reserved
3	Reserved
4	XTS_AES_128_KEY (flash/SRAM encryption and decryption)
5	HMAC Downstream mode (both JTAG and RSA_DS)
6	JTAG in HMAC Downstream mode
7	RSA Digital Signature Peripheral (RSA_DS) 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 5.3-3 provides the details of parameters in BLOCK1 ~ BLOCK10.

Table 5.3-3. Parameters in BLOCK1 to BLOCK10

BLOCK	Parameters	Bit Width	Hardware Use	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 be used to store KEY0 ~ 5. Up to six 256-bit keys can be written into eFuse. Whenever a key is written, its purpose value should also be written (see table 5.3-2). 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.

Note:
Do not program the XTS-AES key or ECDSA_DS key into the KEY5 block, i.e., BLOCK9. Otherwise, the key may be unreadable. Instead, program them into the preceding blocks, i.e., BLOCK4 ~ BLOCK8. The last block, BLOCK9, is used to program other keys.

BLOCK1 ~ BLOCK10 use the RS coding scheme, so there are some limitations on writing to these parameters. For more detailed information, please refer to Section 5.3.1.4 and Section 5.3.2.

5.3.1.1 Parameters Used by Hardware Modules

Parameters used by hardware modules are marked with "Y" in Table 5.3-3 > Column "Accessible by Hardware". They are used by hardware modules through circuit connections. That is to say, changes to the parameter value will directly affect how the peripheral controlled by it behaves, and this process cannot be intervened by the user.

5.3.1.2 EFUSE_WR_DIS

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

Column "Write Protection by EFUSE_WR_DIS Bit Number" in Table 5.3-1 and Table 5.3-3 list the specific bits in EFUSE_WR_DIS that disable writing.

When a write protection bit is set to 0, the corresponding parameter is not write-protected and can be programmed, unless it has been programmed before.

When the write protection bit is set to 1, the corresponding 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.

5.3.1.3 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 5.3-3. After EFUSE_RD_DIS has been programmed, execute an eFuse read operation to make the new values take effect.

If an EFUSE_RD_DIS bit is 0, the corresponding parameter is not read-protected from users. If it is 1, the parameter is read-protected.

Other parameters that are not in BLOCK4 ~ BLOCK10 can always be read by users.

When BLOCK4 ~ BLOCK10 are set to be read-protected, the data in them can still be read by hardware cryptography modules if the EFUSE_KEY_PURPOSE_*n* bit is set accordingly.

5.3.1.4 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 total eFuse memory space occupied by BLOCK0 is 32 + 152 * 4 = 640 bits.

In BLOCK1 ~ BLOCK10, data are coded using Reed-Solomon's RS (44, 32) coding scheme that supports up to 6 bytes of automatic error correction. The primitive polynomial of RS (44, 32) is $p(x) = x^8 + x^4 + x^3 + x^2 + 1$.

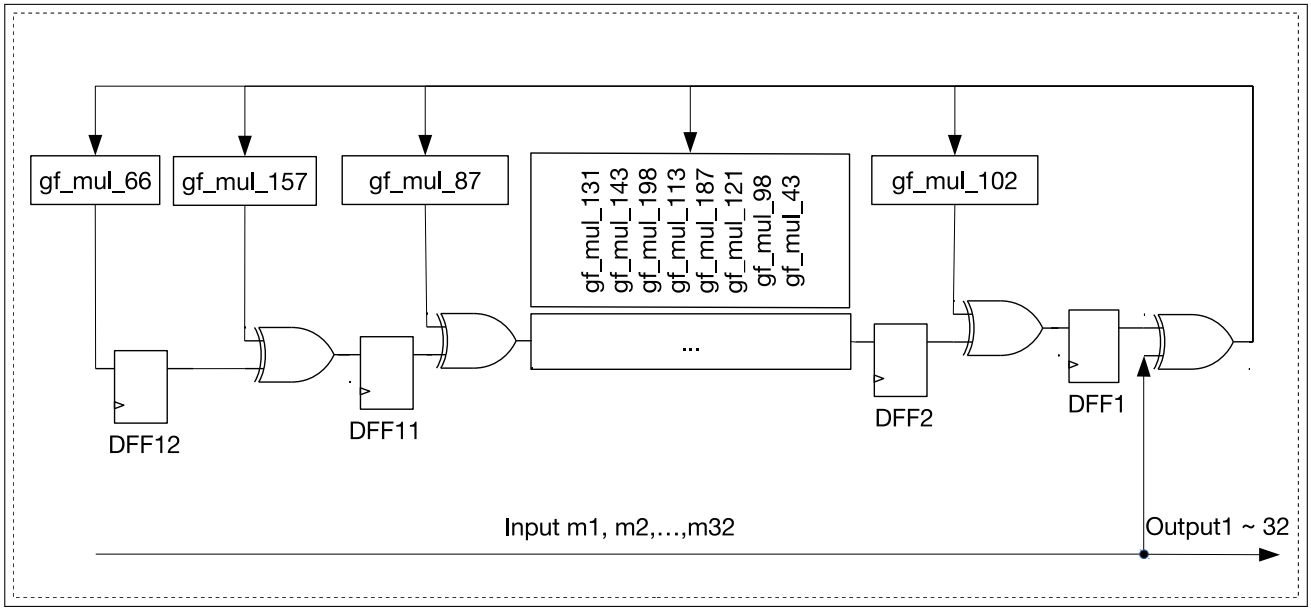


Figure 5.3-2. Shift Register Circuit (first 32 output)

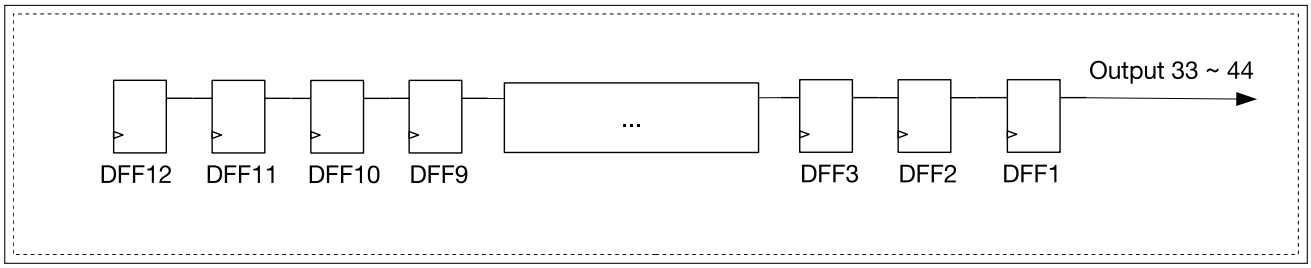


Figure 5.3-3. Shift Register Circuit (last 12 output)

As shown in Figure 5.3-2 and 5.3-3, the shift register circuit processes the 32-byte parameters using RS (44, 32) and encodes them into 44 bytes:

- Bytes [0:31] are the data 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 α^n, where n is an integer).

After that, the hardware programs into eFuse the 44-byte codeword consisting of the data bytes and the parity

bytes. When the eFuse block is read, the eFuse controller automatically decodes the codeword and applies error correction if needed.

Because the RS check codes are generated on the entire 32-byte eFuse block, each block can only be written once.

Since the size of BLOCK1 is less than 32 bytes, the unused bits will be treated as 0 by hardware during the RS (44, 32) encoding. Thus, the final coding result will not be affected.

Among blocks using the RS (44, 32) coding scheme, the parameters in BLOCK1 is 24 bytes, and the RS check code is 12 bytes, so BLOCK1 occupies 24 + 12 = 36 bytes in eFuse memory.

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.

5.3.2 Programming of Parameters

The eFuse controller can only program eFuse parameters to one block at a time. BLOCK0 ~ BLOCK10 share the same address range for storing the parameters to be programmed. Configure parameter EFUSE_BLK_NUM to indicate which block should be programmed.

Since there is a one-to-one correspondence between the reading data registers and the programming data registers (see table 5.3-4 for details), users can find out where the data to be programmed is located in programming registers by checking the parameter description and the parameter location in the corresponding reading data 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 the steps are completed, the corresponding bit in the eFuse memory will be programmed to 1.

Programming preparation

- **Programming BLOCK0**
  1. Set EFUSE_BLK_NUM to 0.
  2. Write into EFUSE_PGM_DATA0_REG ~ EFUSE_PGM_DATA5_REG the data to be programmed to BLOCK0.
     The data in EFUSE_PGM_DATA6_REG ~ EFUSE_PGM_DATA7_REG and EFUSE_PGM_CHECK_VALUE0_REG ~ EFUSE_PGM_CHECK_VALUE2_REG does not affect the programming of BLOCK0.
- **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 5.3.4.
4. Configure the field EFUSE_OP_CODE of register EFUSE_CONF_REG to 0x5A5A.
5. Configure the field EFUSE_PGM_CMD of register EFUSE_CMD_REG to 1.
6. Poll register EFUSE_CMD_REG until it is 0x0, or wait for a PGM_DONE interrupt. For more information on how to identify a PGM_DONE or READ_DONE interrupt, please see the end of Section 5.3.3.
7. Clear the parameters in EFUSE_PGM_DATA0_REG ~ EFUSE_PGM_DATA7_REG and EFUSE_PGM_CHECK_VALUE0_REG ~ EFUSE_PGM_CHECK_VALUE2_REG.
8. Trigger an eFuse read operation (see Section 5.3.3) to update eFuse registers with the new values.
9. Check error record registers. If the values read in error record registers are not 0, 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_ERR0_REG ~ EFUSE_RD_REPEAT_ERR4_REG
   - BLOCK1: EFUSE_MAC_SPI_8M_ERR_NUM, EFUSE_MAC_SPI_8M_FAIL
   - BLOCK2: EFUSE_SYS_PART1_NUM, EFUSE_SYS_PART1_FAIL
   - BLOCK3: EFUSE_USR_DATA_ERR_NUM, EFUSE_USR_DATA_FAIL
   - BLOCK4: EFUSE_KEY0_ERR_NUM, EFUSE_KEY0_FAIL
   - BLOCK5: EFUSE_KEY1_ERR_NUM, EFUSE_KEY1_FAIL
   - BLOCK6: EFUSE_KEY2_ERR_NUM, EFUSE_KEY2_FAIL
   - BLOCK7: EFUSE_KEY3_ERR_NUM, EFUSE_KEY3_FAIL
   - BLOCK8: EFUSE_KEY4_ERR_NUM, EFUSE_KEY4_FAIL
   - BLOCK9: EFUSE_KEY5_ERR_NUM, EFUSE_KEY5_FAIL
   - BLOCK10: EFUSE_SYS_PART2_ERR_NUM, EFUSE_SYS_PART2_FAIL

Limitations

In BLOCK0, each bit can be programmed separately. However, we recommend to minimize programming cycles and program all the bits of a parameter in one programming 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.

5.3.3 Reading of Parameters by Users

Users cannot read eFuse bits directly. The eFuse controller reads eFuse bits and stores the result to the corresponding reading data registers (whose name is prefixed with EFUSE_RD_). Then, users can read the eFuse bits from those registers. Details are provided in Table 5.3-4.

Table 5.3-4. Registers Information

BLOCK	Reading Data Registers	Programming Registers
0	EFUSE_RD_WR_DIS_REG	EFUSE_PGM_DATA0_REG
0	EFUSE_RD_REPEAT_DATA0 ~ 4_REG	EFUSE_PGM_DATA1 ~ 5_REG
1	EFUSE_RD_MAC_SYS_0 ~ 5_REG	EFUSE_PGM_DATA0 ~ 5_REG
2	EFUSE_RD_SYS_DATA_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_DATA_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 reading data 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 9 *Interrupt Matrix (INTMTX)*.
  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.

5.3.4 eFuse VDDQ Timing

The eFuse controller operates at the clock frequency of 20 MHz, and its programming voltage VDDQ should be configured as follows:

- EFUSE_DAC_NUM (the rising period of VDDQ): The default value of VDDQ is 2.5 V and the voltage increases by 0.01 V in each clock cycle. The default value of this parameter is 255.
- EFUSE_DAC_CLK_DIV (the clock divisor of VDDQ): The clock period to program VDDQ should be larger than 1 μs.
- EFUSE_PWR_ON_NUM (the power-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 μs.

Table 5.3-5. Configuration of Default VDDQ Timing Parameters

EFUSE_DAC_NUM	EFUSE_DAC_CLK_DIV	EFUSE_PWR_ON_NUM	EFUSE_PWR_OFF_NUM
0xFF	0x28	0x3000	0x190

5.3.5 Parameters Used by Hardware Modules

5.3.6 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.

5.4 Register Summary

The addresses in this section are relative to eFuse controller base address provided in Table 4.3-2 in Chapter 4 *System and Memory*.

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

Name	Description	Address	Access
Programming Data Registers			
EFUSE_PGM_DATA0_REG	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 Registers			
EFUSE_RD_WR_DIS_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_SYS_0_REG	Register 0 of BLOCK1	0x0044	RO
EFUSE_RD_MAC_SYS_1_REG	Register 1 of BLOCK1	0x0048	RO
EFUSE_RD_MAC_SYS_2_REG	Register 2 of BLOCK1	0x004C	RO
EFUSE_RD_MAC_SYS_3_REG	Register 3 of BLOCK1	0x0050	RO
EFUSE_RD_MAC_SYS_4_REG	Register 4 of BLOCK1	0x0054	RO
EFUSE_RD_MAC_SYS_5_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

Name	Description	Address	Access
EFUSE_RD_USR_DATA1_REG	Register 1 of BLOCK3 (user)	0x0080	RO
EFUSE_RD_USR_DATA2_REG	Register 2 of BLOCK3 (user)	0x0084	RO
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

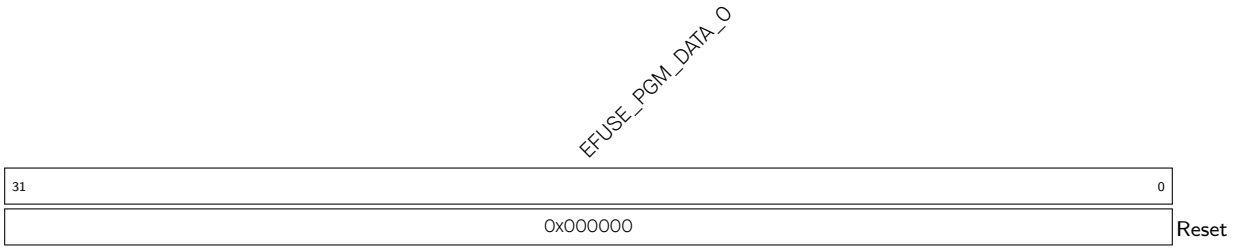
Name	Description	Address	Access
EFUSE_RD_KEY4_DATA4_REG	Register 4 of BLOCK8 (KEY4)	0x012C	RO
EFUSE_RD_KEY4_DATA5_REG	Register 5 of BLOCK8 (KEY4)	0x0130	RO
EFUSE_RD_KEY4_DATA6_REG	Register 6 of BLOCK8 (KEY4)	0x0134	RO
EFUSE_RD_KEY4_DATA7_REG	Register 7 of BLOCK8 (KEY4)	0x0138	RO
EFUSE_RD_KEY5_DATA0_REG	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 Registers			
EFUSE_RD_REPEAT_ERR0_REG	Programming error record register 0 of BLOCK0	0x017C	RO
EFUSE_RD_REPEAT_ERR1_REG	Programming error record register 1 of BLOCK0	0x0180	RO
EFUSE_RD_REPEAT_ERR2_REG	Programming error record register 2 of BLOCK0	0x0184	RO
EFUSE_RD_REPEAT_ERR3_REG	Programming error record register 3 of BLOCK0	0x0188	RO
EFUSE_RD_REPEAT_ERR4_REG	Programming error record register 4 of BLOCK0	0x0190	RO
EFUSE_RD_RS_ERR0_REG	Programming error record register 0 of BLOCK1-10	0x01C0	RO
EFUSE_RD_RS_ERR1_REG	Programming error record register 1 of BLOCK1-10	0x01C4	RO
Configuration Registers			
EFUSE_CLK_REG	eFuse clock configuration register	0x01C8	R/W
EFUSE_CONF_REG	eFuse operation mode configuration register	0x01CC	R/W
EFUSE_CMD_REG	eFuse command register	0x01D4	varies
EFUSE_DAC_CONF_REG	Controls the eFuse programming voltage	0x01E8	R/W
EFUSE_RD_TIM_CONF_REG	Configures read timing parameters	0x01EC	R/W
EFUSE_WR_TIM_CONF1_REG	Configuration register 1 of eFuse programming timing parameters	0x01F0	R/W

Name	Description	Address	Access
EFUSE_WR_TIM_CONF2_REG	Configuration register 2 of eFuse programming timing parameters	0x01F4	R/W
EFUSE_WR_TIM_CONF0_RS_BYPASS_REG	Configuration register0 of eFuse programming time parameters and rs bypass operation.	0x01F8	varies
Status Register			
EFUSE_STATUS_REG	eFuse status register.	0x01D0	RO
Interrupt Registers			
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
Version Control Register			
EFUSE_DATE_REG	Version control register	0x01FC	R/W

5.5 Registers

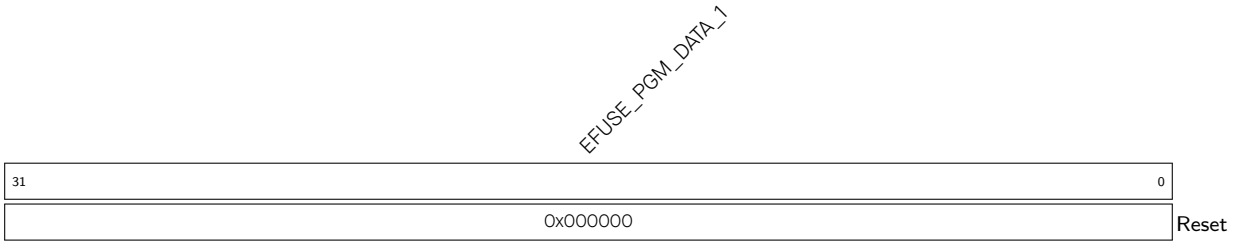
The addresses in this section are relative to eFuse controller base address provided in Table 4.3-2 in Chapter 4 *System and Memory*.

Register 5.1. EFUSE_PGM_DATA0_REG (0x0000)



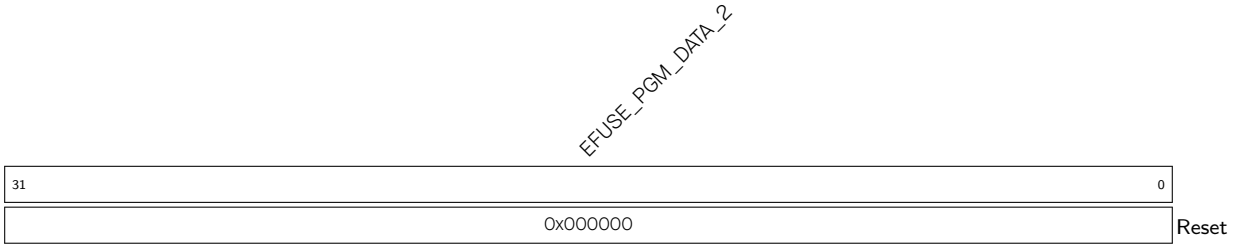
EFUSE_PGM_DATA_0 Configures the 0th 32-bit data to be programmed. (R/W)

Register 5.2. EFUSE_PGM_DATA1_REG (0x0004)



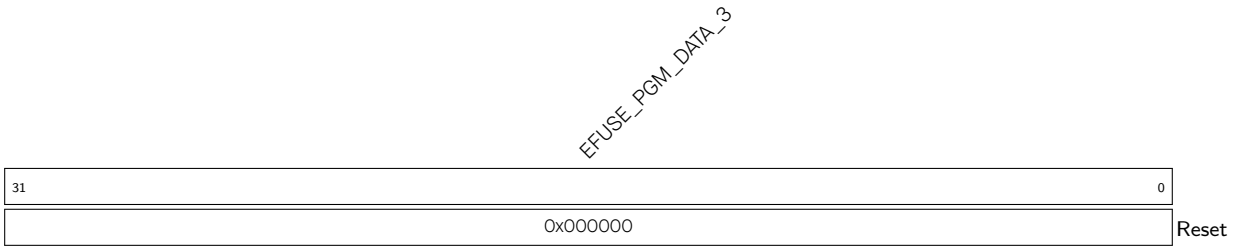
EFUSE_PGM_DATA_1 Configures the 1st 32-bit data to be programmed. (R/W)

Register 5.3. EFUSE_PGM_DATA2_REG (0x0008)



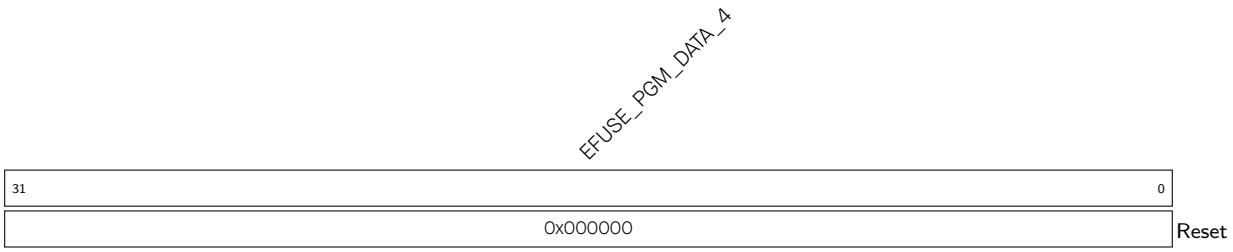
EFUSE_PGM_DATA_2 Configures the 2nd 32-bit data to be programmed. (R/W)

Register 5.4. EFUSE_PGM_DATA3_REG (0x000C)



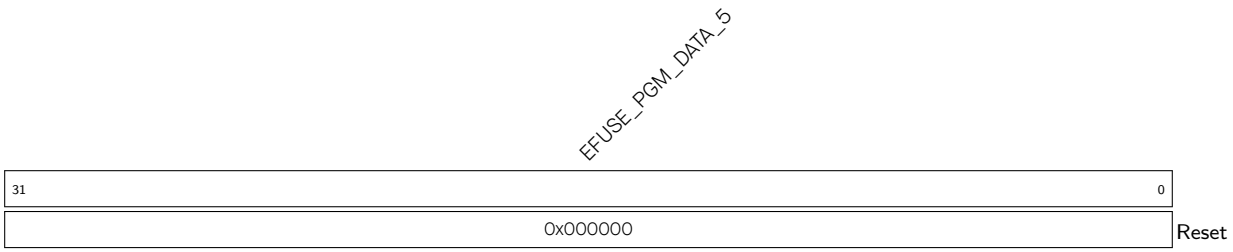
EFUSE_PGM_DATA_3 Configures the 3rd 32-bit data to be programmed. (R/W)

Register 5.5. EFUSE_PGM_DATA4_REG (0x0010)



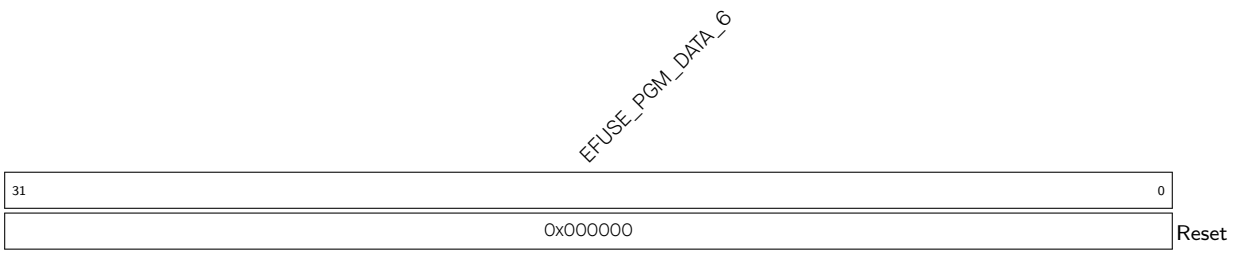
EFUSE_PGM_DATA_4 Configures the 4th 32-bit data to be programmed. (R/W)

Register 5.6. EFUSE_PGM_DATA5_REG (0x0014)



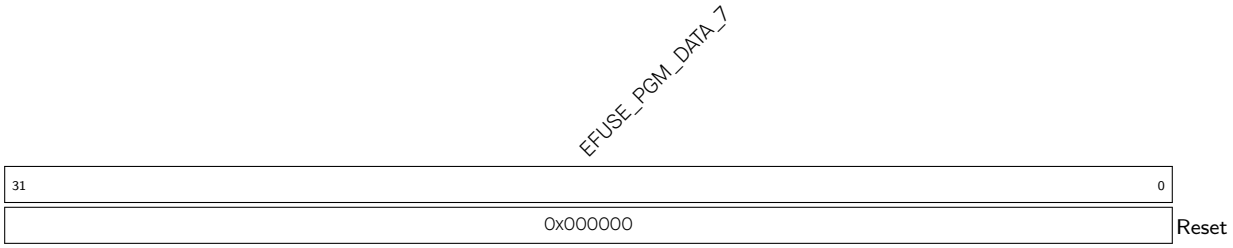
EFUSE_PGM_DATA_5 Configures the 5th 32-bit data to be programmed. (R/W)

Register 5.7. EFUSE_PGM_DATA6_REG (0x0018)



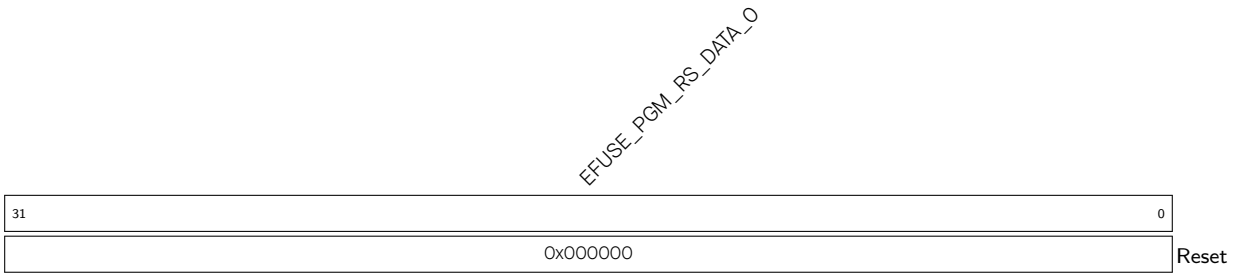
EFUSE_PGM_DATA_6 Configures the 6th 32-bit data to be programmed. (R/W)

Register 5.8. EFUSE_PGM_DATA7_REG (0x001C)



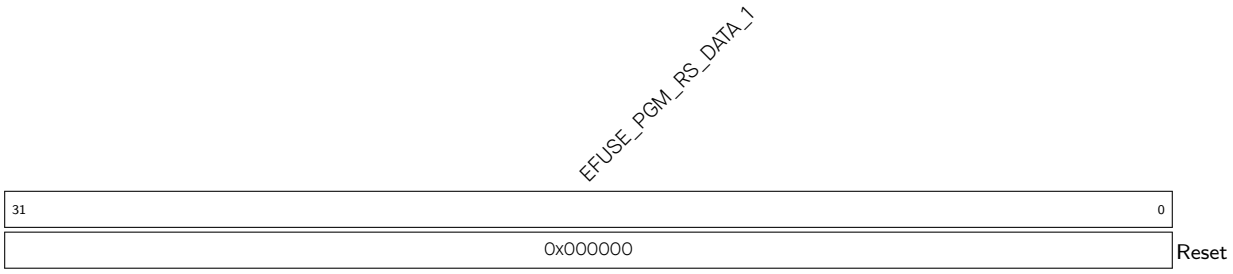
EFUSE_PGM_DATA_7 Configures the 7th 32-bit data to be programmed. (R/W)

Register 5.9. EFUSE_PGM_CHECK_VALUE0_REG (0x0020)



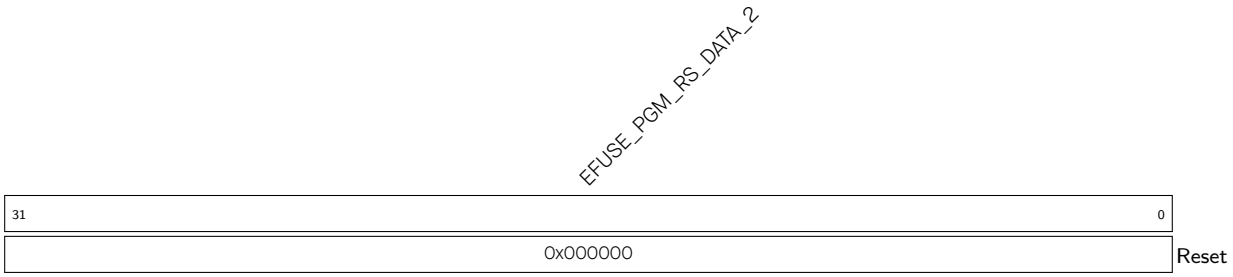
EFUSE_PGM_RS_DATA_0 Configures the 0th 32-bit RS code to be programmed. (R/W)

Register 5.10. EFUSE_PGM_CHECK_VALUE1_REG (0x0024)



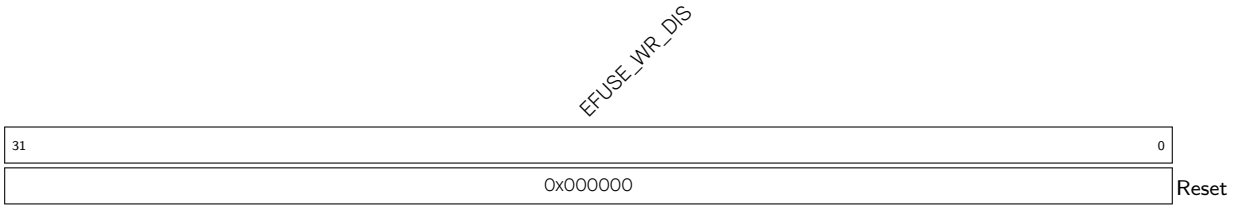
EFUSE_PGM_RS_DATA_1 Configures the 1st 32-bit RS code to be programmed. (R/W)

Register 5.11. EFUSE_PGM_CHECK_VALUE2_REG (0x0028)



EFUSE_PGM_RS_DATA_2 Configures the 2nd 32-bit RS code to be programmed. (R/W)

Register 5.12. EFUSE_RD_WR_DIS_REG (0x002C)



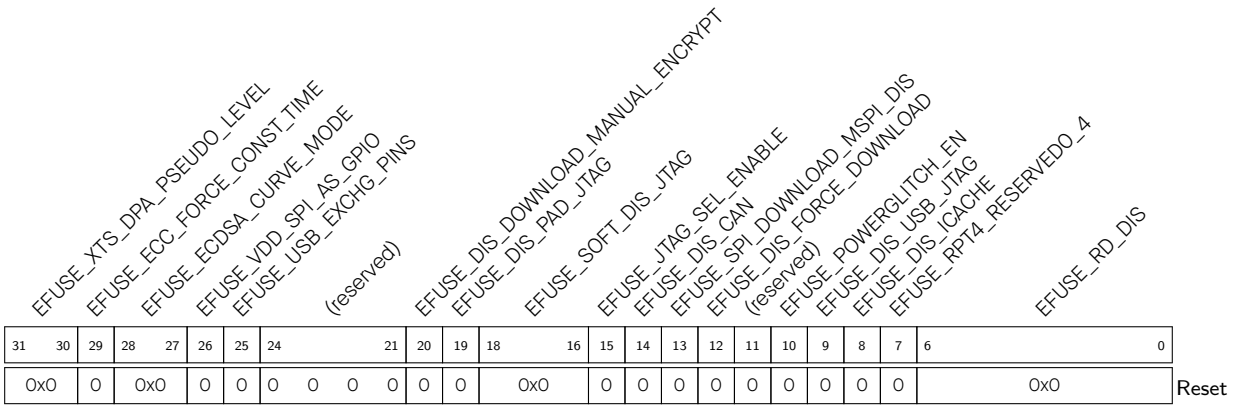
EFUSE_WR_DIS Represents whether programming of individual eFuse memory bit is disabled.

1: Disabled

0: Enabled

(RO)

Register 5.13. EFUSE_RD_REPEAT_DATA0_REG (0x0030)



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

1: Disabled

0: Enabled

(RO)

EFUSE_RPT4_RESERVED0_4 Reserved. (RO)

EFUSE_DIS_ICACHE Represents whether instruction cache is disabled.

1: Disabled

0: Enabled

(RO)

EFUSE_DIS_USB_JTAG Represents whether the USB-to-JTAG function is disabled.

1: Disabled

0: Enabled

(RO)

EFUSE_POWERGLITCH_EN Represents whether to enable the power glitch detection.

1: Enabled

0: Disabled

(RO)

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

1: Disabled

0: Enabled

(RO)

Continued on the next page...

Register 5.13. EFUSE_RD_REPEAT_DATA0_REG (0x0030)

Continued from the previous page...

EFUSE_SPI_DOWNLOAD_MSPI_DIS Represents whether SPI0 controller is disabled during boot_mode_download.
1: Disabled
0: Enabled
(RO)

EFUSE_DIS_CAN Represents whether TWAI function is disabled.
1: Disabled
0: Enabled
(RO)

EFUSE_JTAG_SEL_ENABLE Represents whether the selection of a JTAG signal source through the strapping value of GPIO25 is enabled when both EFUSE_DIS_PAD_JTAG and EFUSE_DIS_USB_JTAG are configured to 0.
1: Enabled
0: Disabled
(RO)

EFUSE_SOFT_DIS_JTAG Represents whether JTAG is disabled in the soft way. It can be restarted via HMAC.
Odd count of bits with a value of 1: Disabled
Even count of bits with a value of 1: Enabled
(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 encryption is disabled (except in SPI boot mode).
1: Disabled
0: Enabled
(RO)

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

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

Continued on the next page...

Register 5.13. EFUSE_RD_REPEAT_DATA0_REG (0x0030)

Continued from the previous page...

EFUSE_ECDSA_CURVE_MODE Configures the curve of ECDSA_DS calculation:

0: only enable P256.

1: only enable P192.

2: both enable P256 and P192.

3: only enable P256.

(RO)

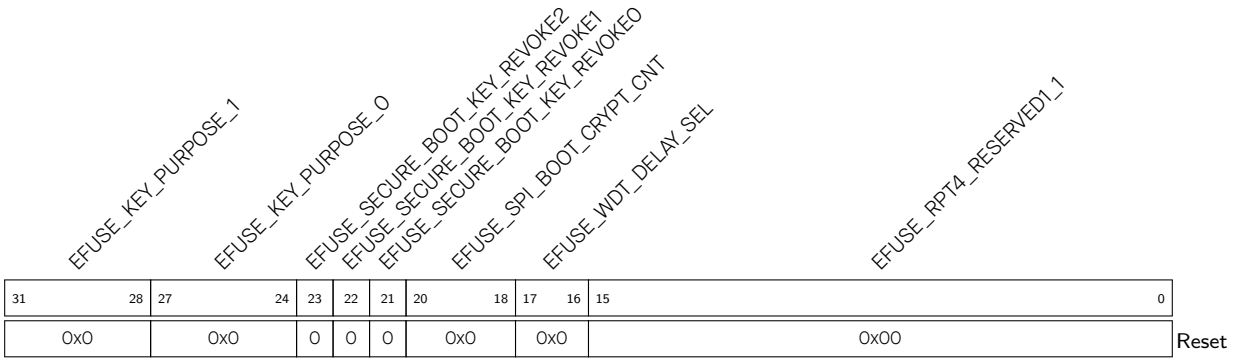
EFUSE_ECC_FORCE_CONST_TIME Configures whether to permanently turn on ECC const-time mode. (RO)

EFUSE_XTS_DPA_PSEUDO_LEVEL Configures the xts pseudo-round anti-dpa attack function.

0: controlled by register.

1-3: the higher the value is, the more pseudo-rounds are inserted to the xts-aes calculation. (RO)

Register 5.14. EFUSE_RD_REPEAT_DATA1_REG (0x0034)



EFUSE_RPT4_RESERVED1_1 Reserved. (RO)

EFUSE_WDT_DELAY_SEL Represents whether RTC watchdog timeout threshold is selected at startup.
1: Selected.
0: Not selected
(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)

EFUSE_SECURE_BOOT_KEY_REVOKE0 Represents whether revoking the 1st Secure Boot key is enabled.
1: Enabled
0: Disabled
(RO)

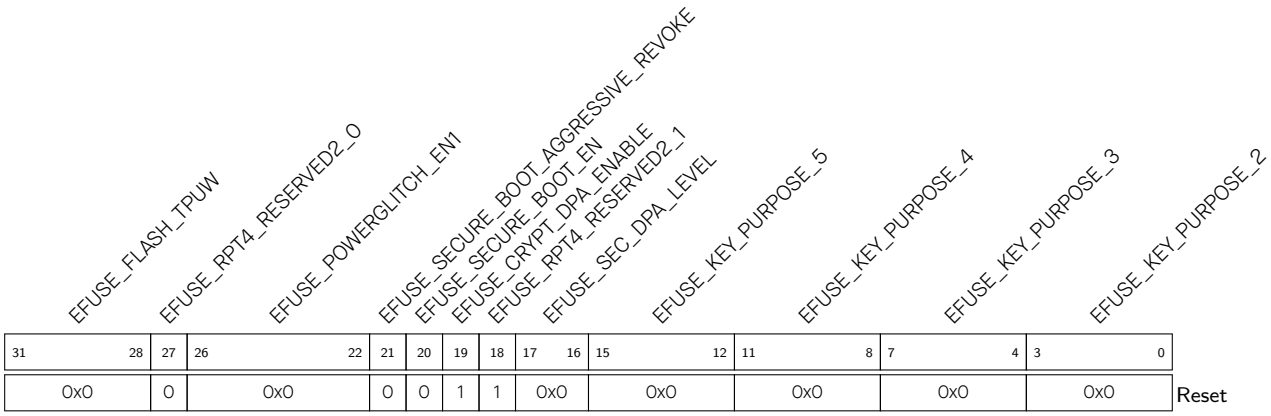
EFUSE_SECURE_BOOT_KEY_REVOKE1 Represents whether revoking the 2nd Secure Boot key is enabled.
1: Enabled
0: Disabled
(RO)

EFUSE_SECURE_BOOT_KEY_REVOKE2 Represents whether revoking the 3rd Secure Boot key is enabled.
1: Enabled
0: Disabled
(RO)

EFUSE_KEY_PURPOSE_0 Represents the purpose of Key0. (RO)

EFUSE_KEY_PURPOSE_1 Represents the purpose of Key1. (RO)

Register 5.15. EFUSE_RD_REPEAT_DATA2_REG (0x0038)



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 the security level of anti-DPA attack.

0: Security level is SEC_DPA_OFF

1/2: Security level is SEC_DPA_LOW

3: Security level is SEC_DPA_HIGH

For more information, please refer to Chapter 16 *System Registers* > Section 16.2.2.

(RO)

EFUSE_RPT4_RESERVED2_1 Reserved. (RO)

EFUSE_CRYPT_DPA_ENABLE Represents whether defense against DPA attack is enabled.

1: Enabled

0: Disabled

(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_POWERGLITCH_EN1 Configures whether to enable power glitch function when the chip is powered on. (RO)

EFUSE_RPT4_RESERVED2_0 Reserved. (RO)

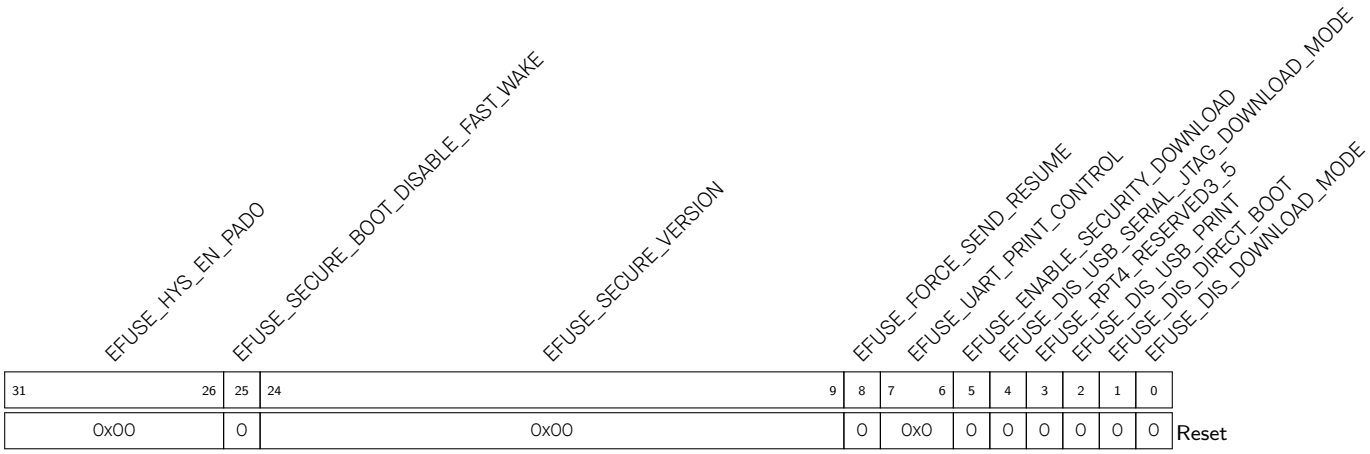
Continued on the next page...

Register 5.15. EFUSE_RD_REPEAT_DATA2_REG (0x0038)

Continued from the previous page...

EFUSE_FLASH_TPUW Represents the flash waiting time after power-up. Measurement unit: ms. When the value is less than 15, the waiting time is the programmed value. Otherwise, the waiting time is a fixed value, i.e., 30 ms. (RO)

Register 5.16. EFUSE_RD_REPEAT_DATA3_REG (0x003C)



EFUSE_DIS_DOWNLOAD_MODE Represents whether all download modes are 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_PRINT Represents whether print from USB-Serial-JTAG is disabled or enabled.

1: Disabled

0: Enabled

(RO)

EFUSE_RPT4_RESERVED3_5 Reserved. (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. Only UART is supported for download. Reading/writing RAM or registers is not supported (i.e., Stub download is not supported).

1: Enabled

0: Disabled

(RO)

Continued on the next page...

Register 5.16. EFUSE_RD_REPEAT_DATA3_REG (0x003C)

Continued from the previous page...

EFUSE_UART_PRINT_CONTROL Represents the type 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)

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)

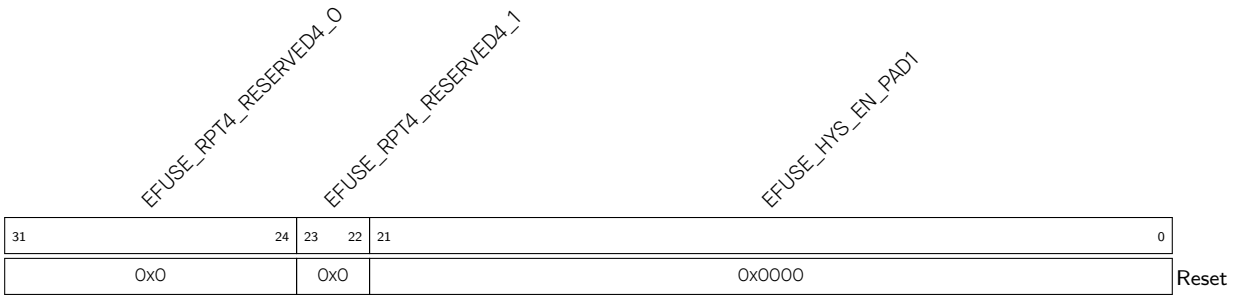
EFUSE_HYS_EN_PAD0 Represents whether to enable the hysteresis function of pad 0-5.

0: Disabled

1: Enabled

(RO)

Register 5.17. EFUSE_RD_REPEAT_DATA4_REG (0x0040)



EFUSE_HYS_EN_PAD1 Represents whether to enable the hysteresis function of pad 6-27.

0: Disabled

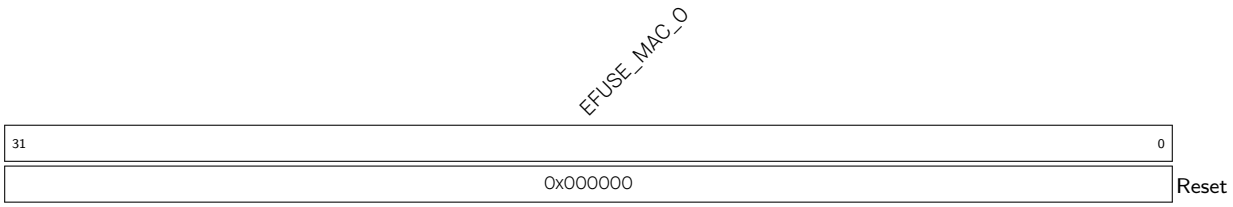
1: Enabled

(RO)

EFUSE_RPT4_RESERVED4_1 Reserved. (RO)

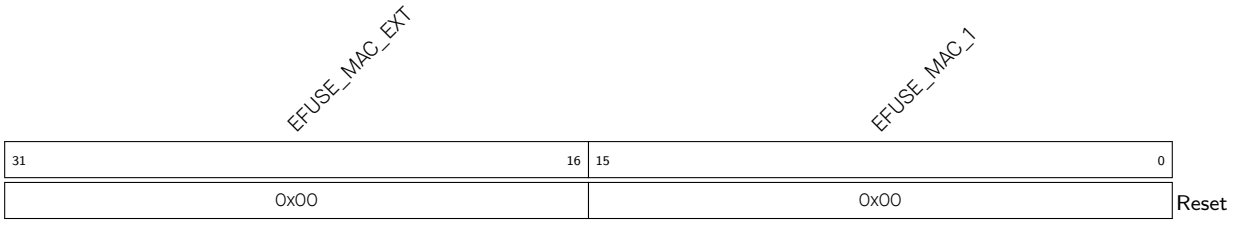
EFUSE_RPT4_RESERVED4_0 Reserved. (RO)

Register 5.18. EFUSE_RD_MAC_SYS_0_REG (0x0044)



EFUSE_MAC_0 Represents the low 32 bits of MAC address. (RO)

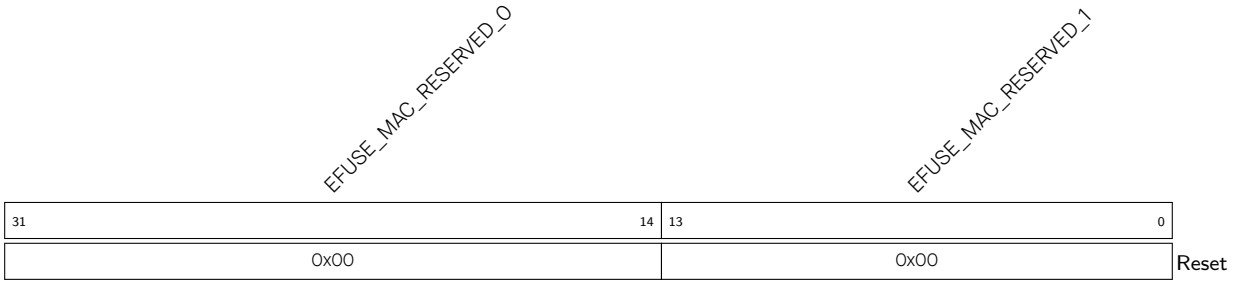
Register 5.19. EFUSE_RD_MAC_SYS_1_REG (0x0048)



EFUSE_MAC_1 Represents the high 16 bits of MAC address. (RO)

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

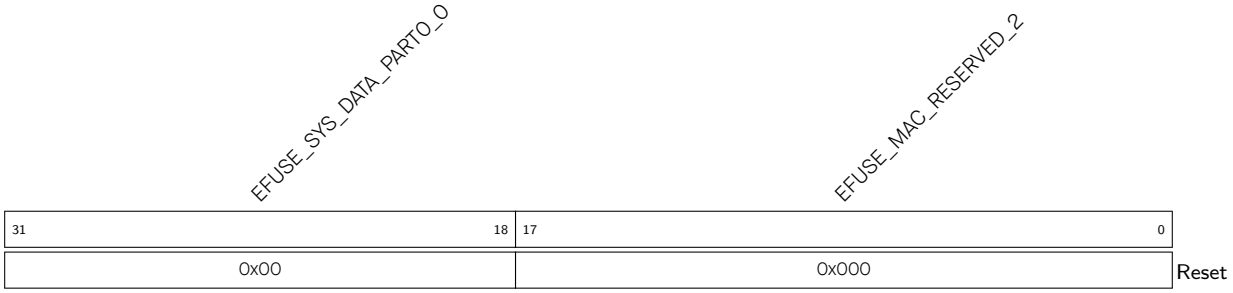
Register 5.20. EFUSE_RD_MAC_SYS_2_REG (0x004C)



EFUSE_MAC_RESERVED_1 Reserved. (RO)

EFUSE_MAC_RESERVED_0 Reserved. (RO)

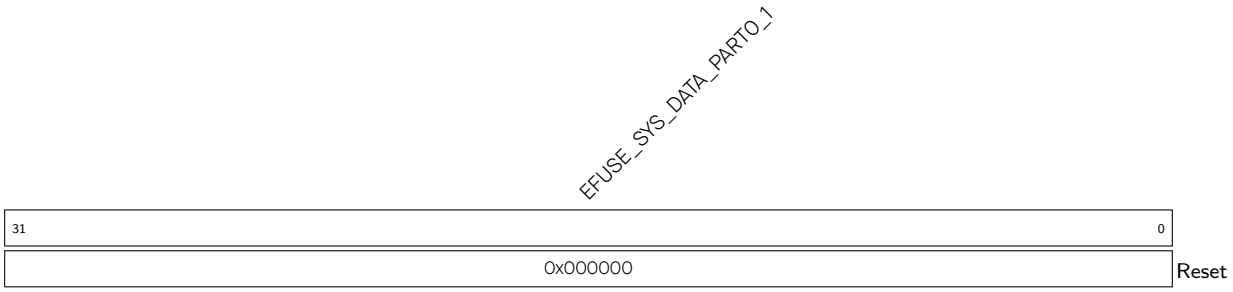
Register 5.21. EFUSE_RD_MAC_SYS_3_REG (0x0050)



EFUSE_MAC_RESERVED_2 Reserved. (RO)

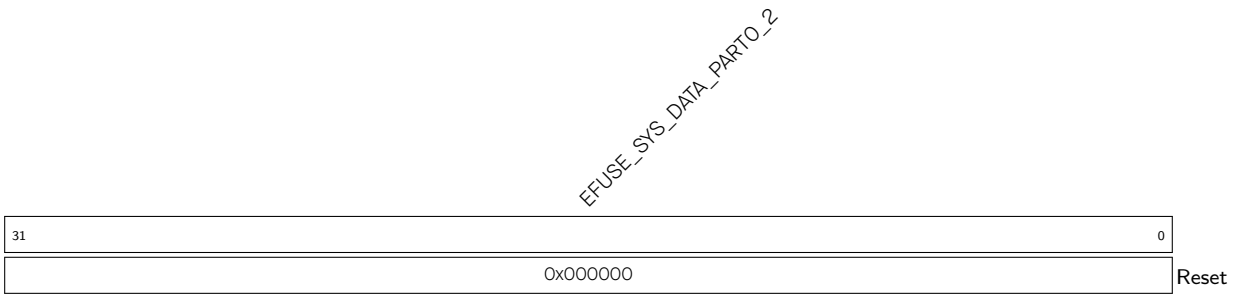
EFUSE_SYS_DATA_PART0_0 Represents the first 14 bits of the zeroth part of system data. (RO)

Register 5.22. EFUSE_RD_MAC_SYS_4_REG (0x0054)



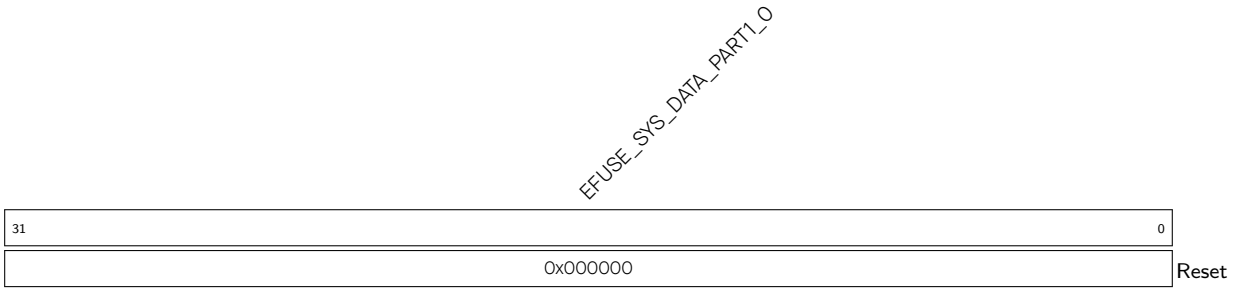
EFUSE_SYS_DATA_PART0_1 Represents the 1st 32 bits of the zeroth part of system data. (RO)

Register 5.23. EFUSE_RD_MAC_SYS_5_REG (0x0058)



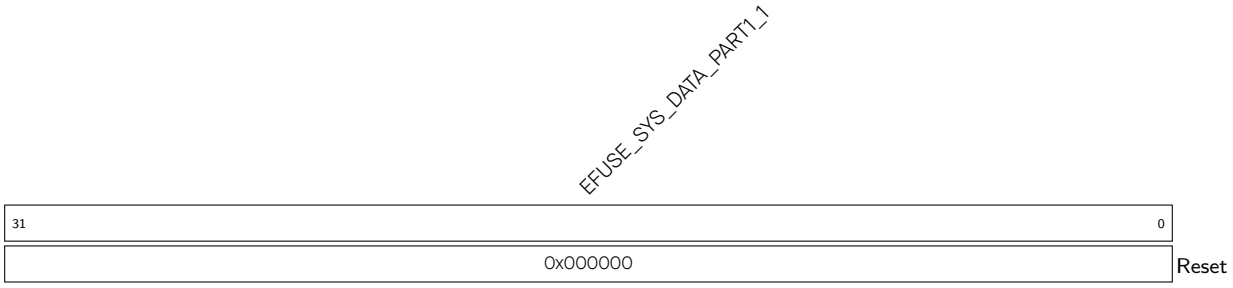
EFUSE_SYS_DATA_PART0_2 Represents the 2nd 32 bits of the zeroth part of system data. (RO)

Register 5.24. EFUSE_RD_SYS_PART1_DATA0_REG (0x005C)



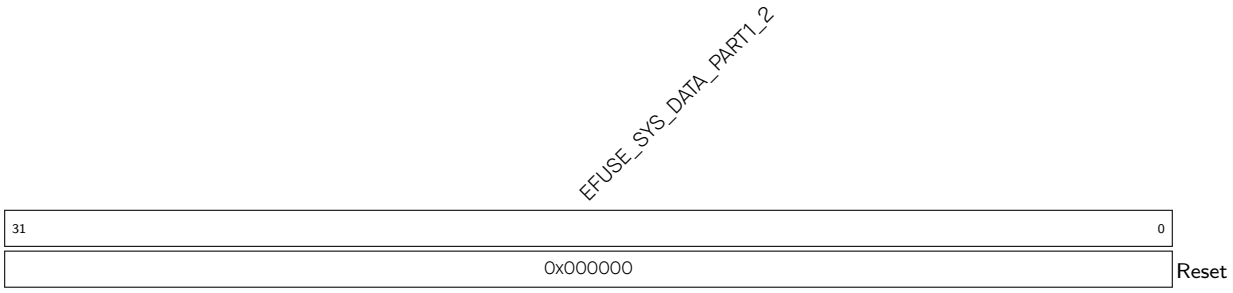
EFUSE_SYS_DATA_PART1_0 Represents the zeroth 32 bits of the 1st part of system data. (RO)

Register 5.25. EFUSE_RD_SYS_PART1_DATA1_REG (0x0060)



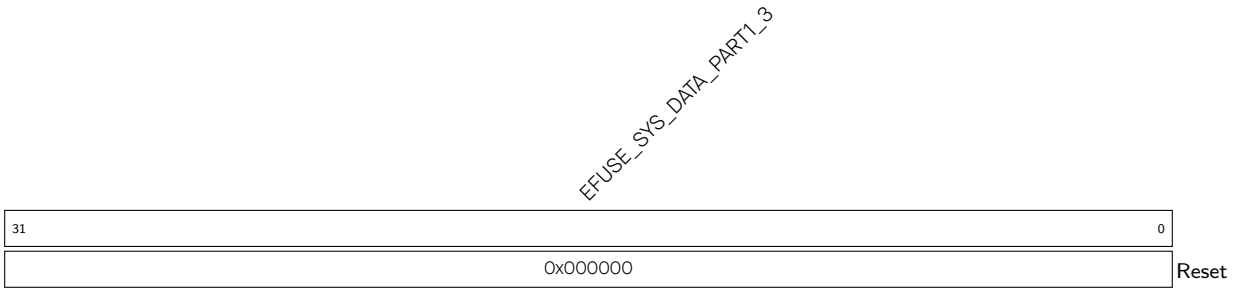
EFUSE_SYS_DATA_PART1_1 Represents the 1st 32 bits of the 1st part of system data. (RO)

Register 5.26. EFUSE_RD_SYS_PART1_DATA2_REG (0x0064)



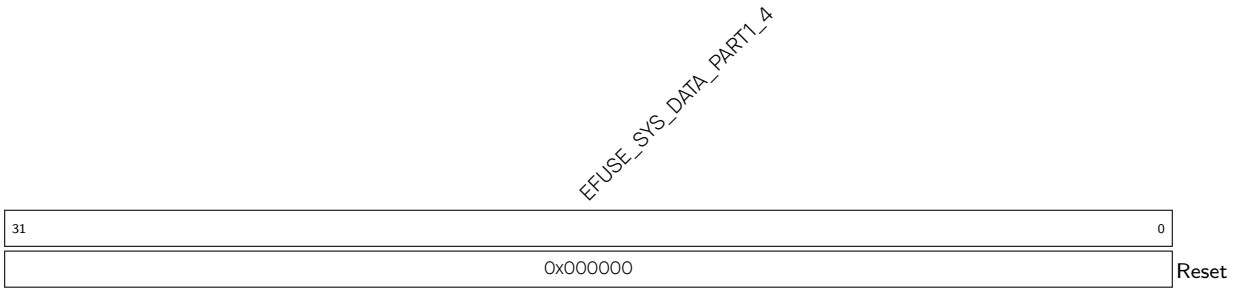
EFUSE_SYS_DATA_PART1_2 Represents the 2nd 32 bits of the 1st part of system data. (RO)

Register 5.27. EFUSE_RD_SYS_PART1_DATA3_REG (0x0068)



EFUSE_SYS_DATA_PART1_3 Represents the 3rd 32 bits of the 1st part of system data. (RO)

Register 5.28. EFUSE_RD_SYS_PART1_DATA4_REG (0x006C)

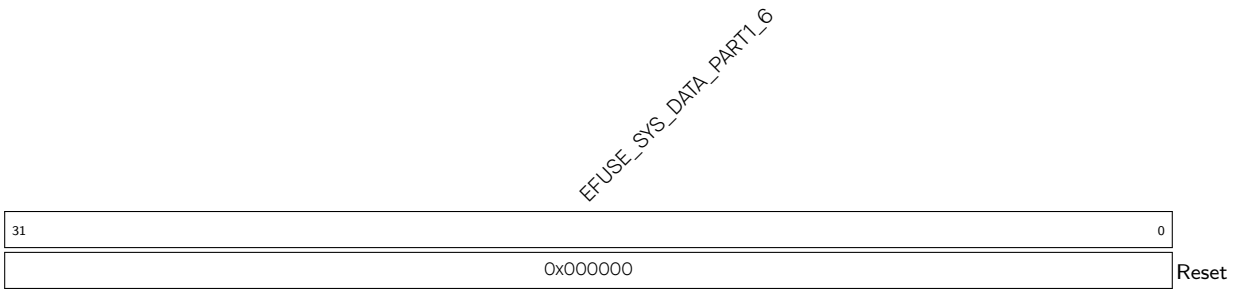


EFUSE_SYS_DATA_PART1_4 Represents the 4th 32 bits of the 1st part of system data. (RO)

Register 5.29. EFUSE_RD_SYS_PART1_DATA5_REG (0x0070)

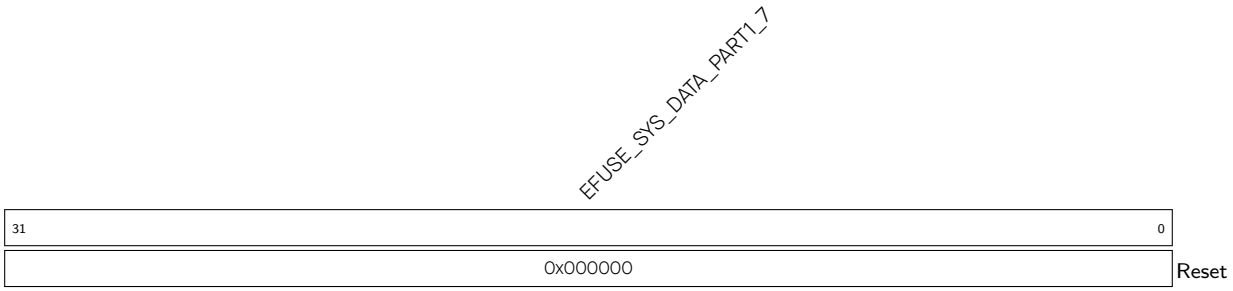
EFUSE_SYS_DATA_PART1_5 Represents the 5th 32 bits of the 1st part of system data. (RO)

Register 5.30. EFUSE_RD_SYS_PART1_DATA6_REG (0x0074)



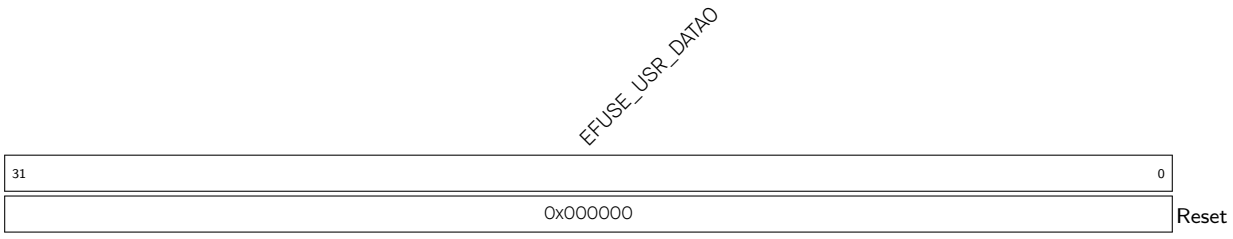
EFUSE_SYS_DATA_PART1_6 Represents the 6th 32 bits of the 1st part of system data. (RO)

Register 5.31. EFUSE_RD_SYS_PART1_DATA7_REG (0x0078)



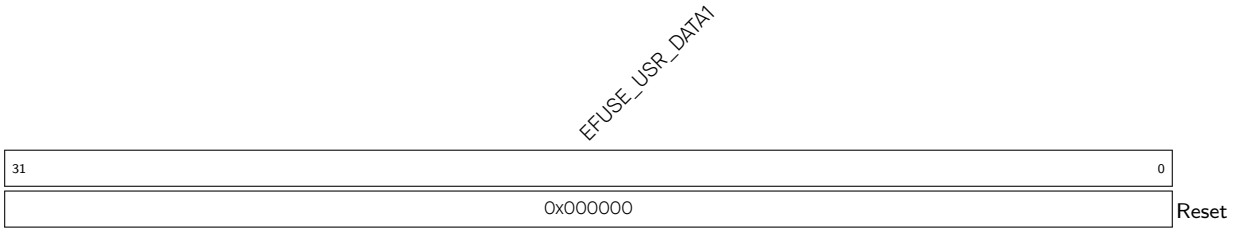
EFUSE_SYS_DATA_PART1_7 Represents the 7th 32 bits of the 1st part of system data. (RO)

Register 5.32. EFUSE_RD_USR_DATA0_REG (0x007C)



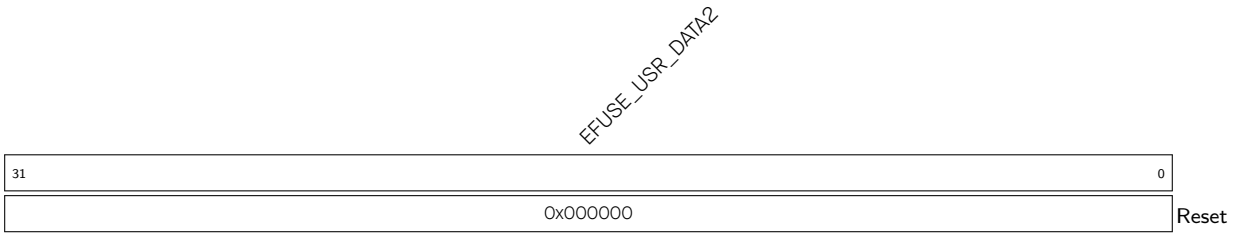
EFUSE_USR_DATA0 Represents the 0th 32 bits of BLOCK3 (user). (RO)

Register 5.33. EFUSE_RD_USR_DATA1_REG (0x0080)



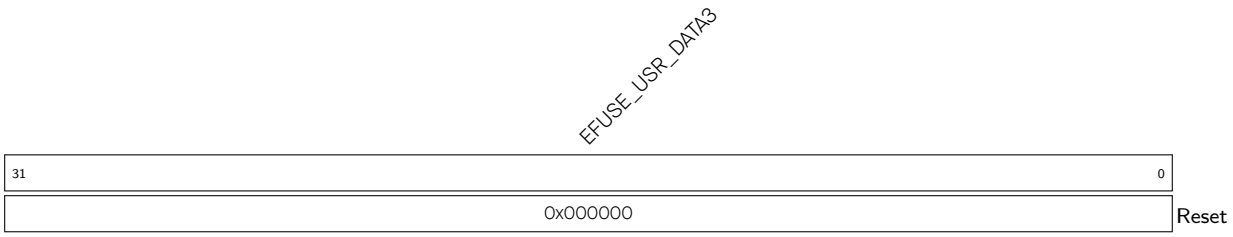
EFUSE_USR_DATA1 Represents the 1st 32 bits of BLOCK3 (user). (RO)

Register 5.34. EFUSE_RD_USR_DATA2_REG (0x0084)



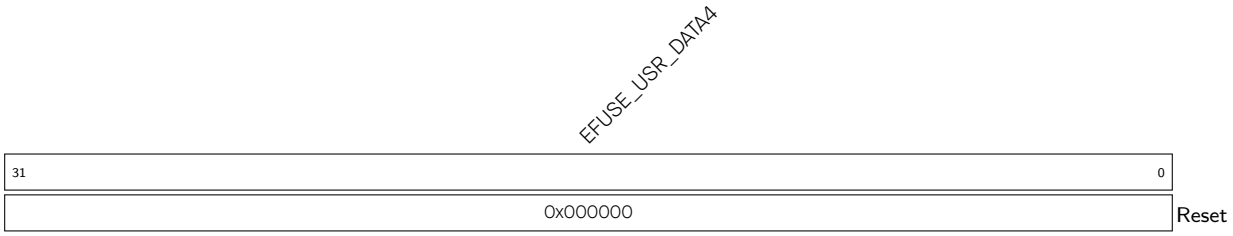
EFUSE_USR_DATA2 Represents the 2nd 32 bits of BLOCK3 (user). (RO)

Register 5.35. EFUSE_RD_USR_DATA3_REG (0x0088)



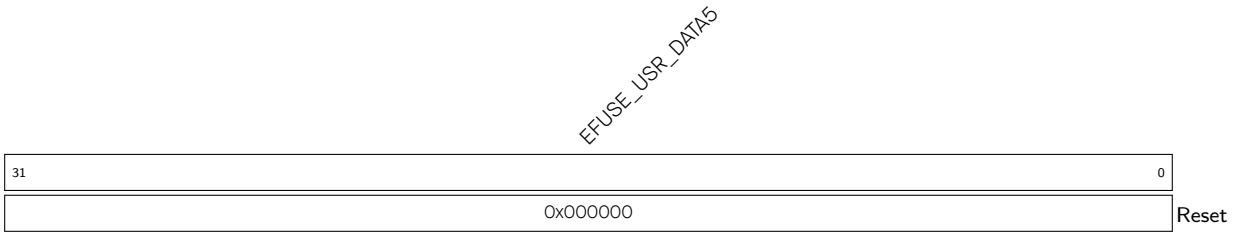
EFUSE_USR_DATA3 Represents the 3rd 32 bits of BLOCK3 (user). (RO)

Register 5.36. EFUSE_RD_USR_DATA4_REG (0x008C)



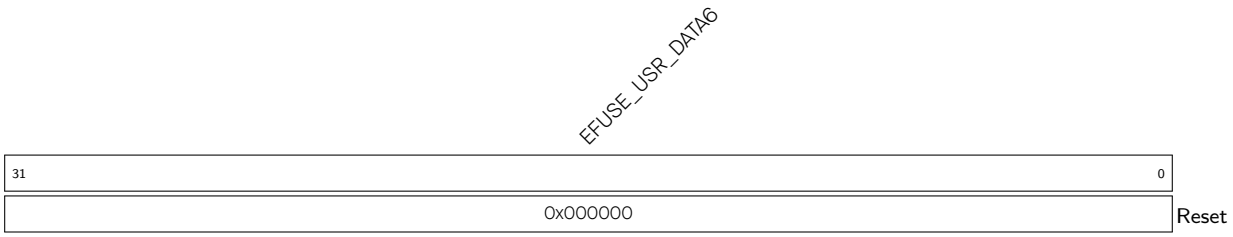
EFUSE_USR_DATA4 Represents the 4th 32 bits of BLOCK3 (user). (RO)

Register 5.37. EFUSE_RD_USR_DATA5_REG (0x0090)



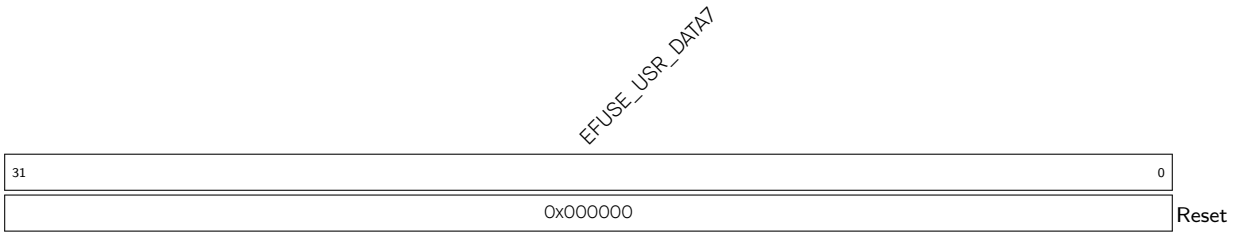
EFUSE_USR_DATA5 Represents the 5th 32 bits of BLOCK3 (user). (RO)

Register 5.38. EFUSE_RD_USR_DATA6_REG (0x0094)



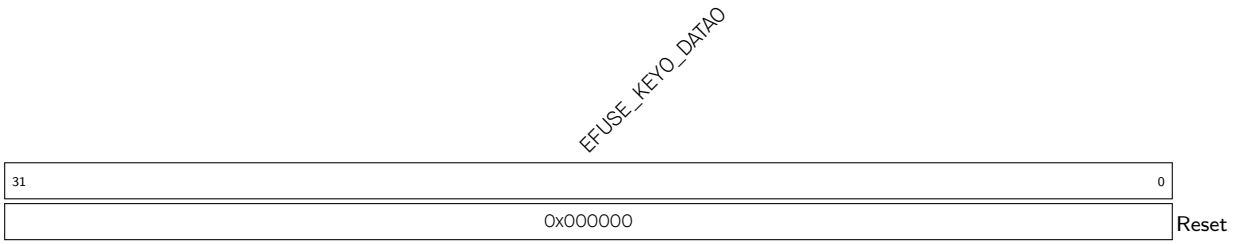
EFUSE_USR_DATA6 Represents the 6th 32 bits of BLOCK3 (user). (RO)

Register 5.39. EFUSE_RD_USR_DATA7_REG (0x0098)



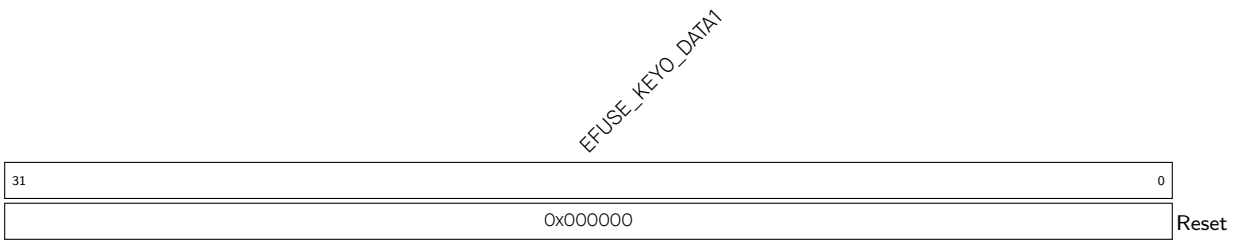
EFUSE_USR_DATA7 Represents the 7th 32 bits of BLOCK3 (user). (RO)

Register 5.40. EFUSE_RD_KEY0_DATA0_REG (0x009C)



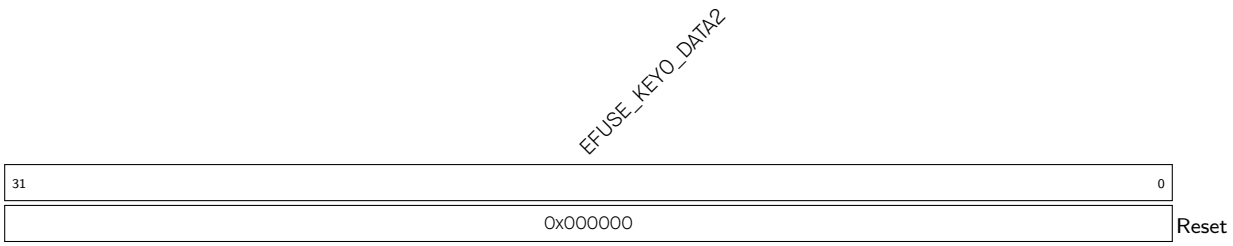
EFUSE_KEY0_DATA0 Represents the 0th 32 bits of KEY0. (RO)

Register 5.41. EFUSE_RD_KEY0_DATA1_REG (0x00A0)



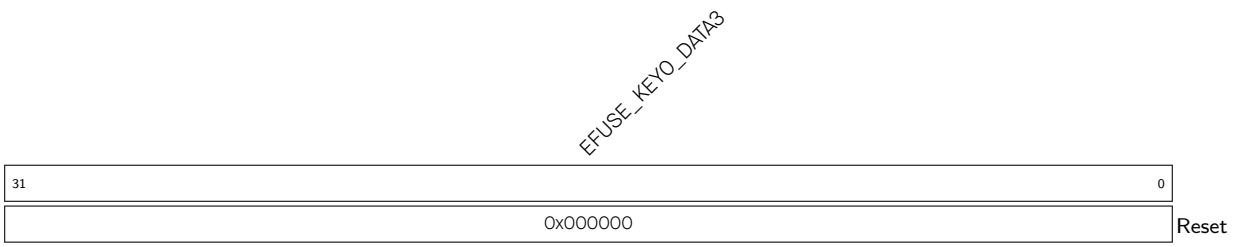
EFUSE_KEY0_DATA1 Represents the 1st 32 bits of KEY0. (RO)

Register 5.42. EFUSE_RD_KEY0_DATA2_REG (0x00A4)



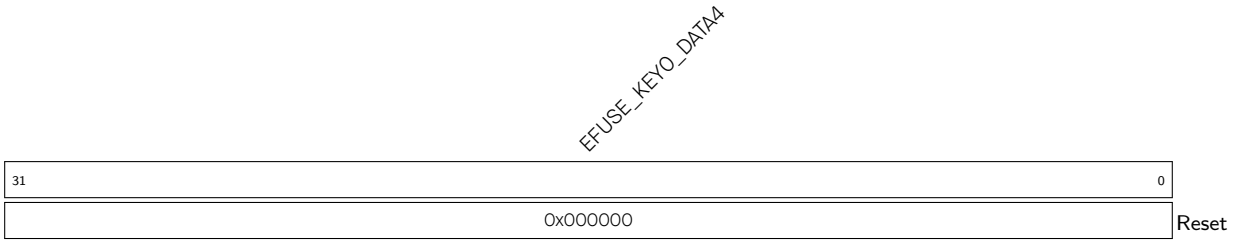
EFUSE_KEY0_DATA2 Represents the 2nd 32 bits of KEY0. (RO)

Register 5.43. EFUSE_RD_KEY0_DATA3_REG (0x00A8)



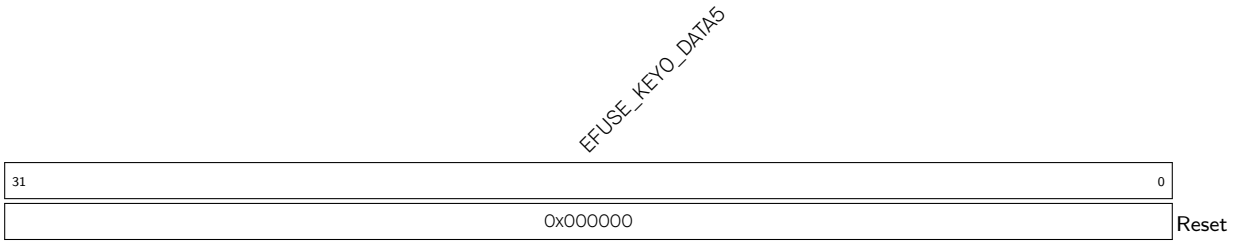
EFUSE_KEY0_DATA3 Represents the 3rd 32 bits of KEY0. (RO)

Register 5.44. EFUSE_RD_KEY0_DATA4_REG (0x00AC)



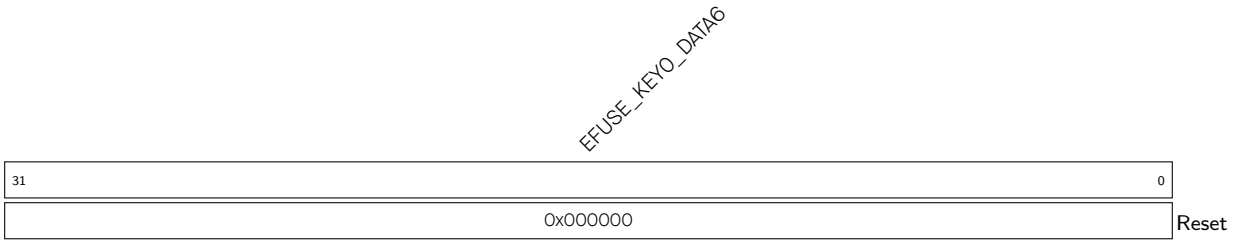
EFUSE_KEY0_DATA4 Represents the 4th 32 bits of KEY0. (RO)

Register 5.45. EFUSE_RD_KEY0_DATA5_REG (0x00B0)



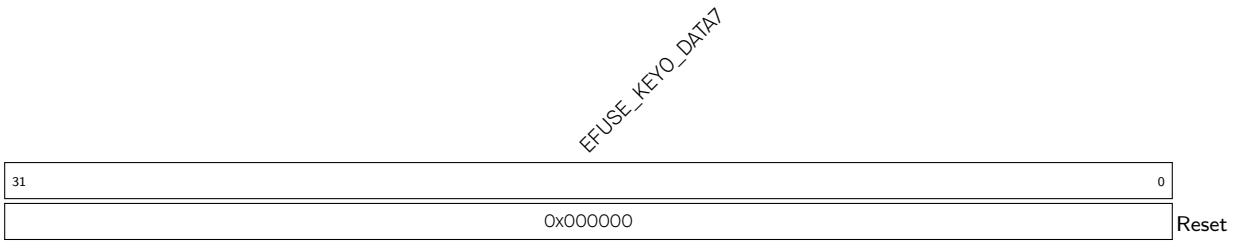
EFUSE_KEY0_DATA5 Represents the 5th 32 bits of KEY0. (RO)

Register 5.46. EFUSE_RD_KEY0_DATA6_REG (0x00B4)



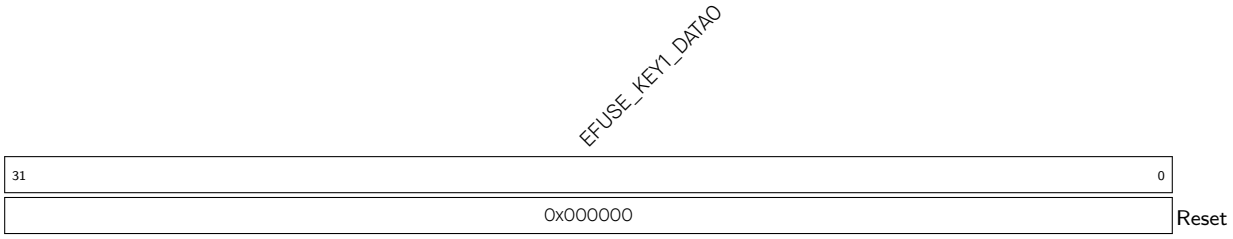
EFUSE_KEY0_DATA6 Represents the 6th 32 bits of KEY0. (RO)

Register 5.47. EFUSE_RD_KEY0_DATA7_REG (0x00B8)



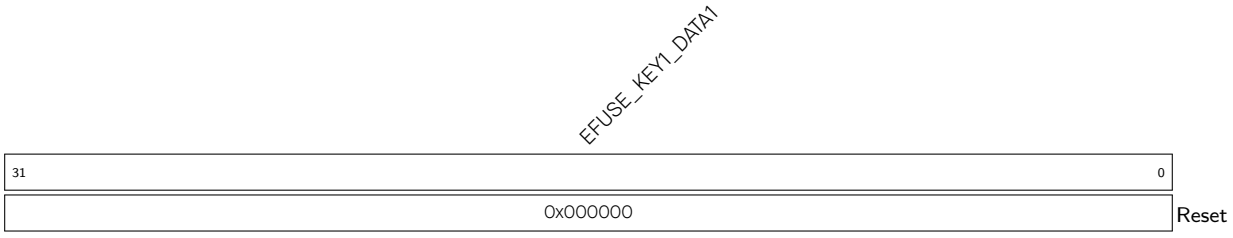
EFUSE_KEY0_DATA7 Represents the 7th 32 bits of KEY0. (RO)

Register 5.48. EFUSE_RD_KEY1_DATA0_REG (0x00BC)



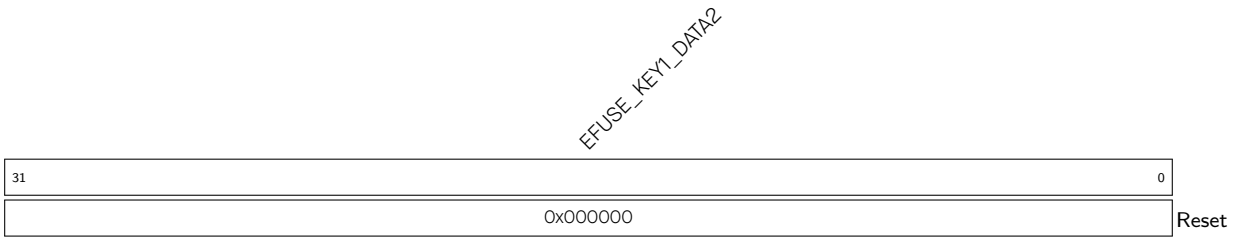
EFUSE_KEY1_DATA0 Represents the zeroth 32 bits of KEY1. (RO)

Register 5.49. EFUSE_RD_KEY1_DATA1_REG (0x00C0)



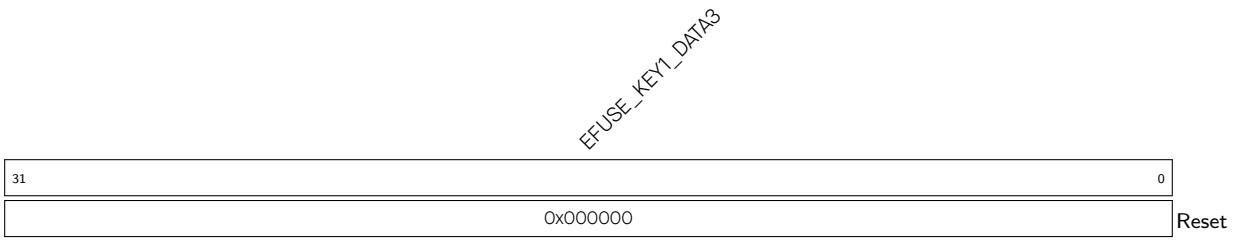
EFUSE_KEY1_DATA1 Represents the 1st 32 bits of KEY1. (RO)

Register 5.50. EFUSE_RD_KEY1_DATA2_REG (0x00C4)



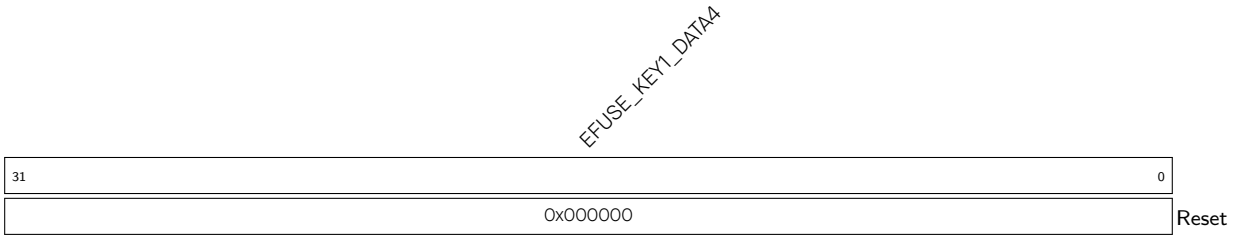
EFUSE_KEY1_DATA2 Represents the 2nd 32 bits of KEY1. (RO)

Register 5.51. EFUSE_RD_KEY1_DATA3_REG (0x00C8)



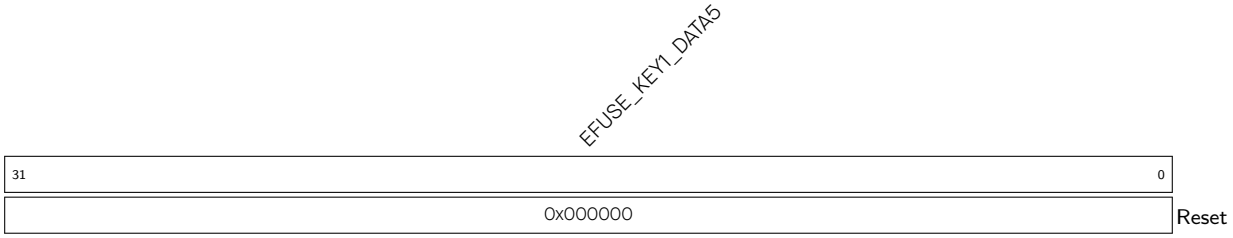
EFUSE_KEY1_DATA3 Represents the 3rd 32 bits of KEY1. (RO)

Register 5.52. EFUSE_RD_KEY1_DATA4_REG (0x00CC)



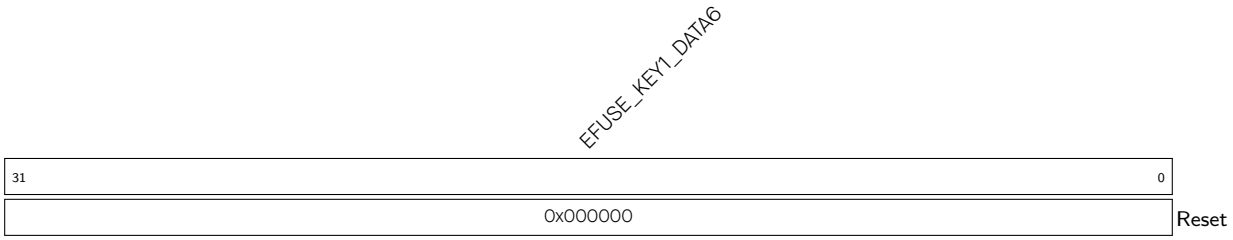
EFUSE_KEY1_DATA4 Represents the 4th 32 bits of KEY1. (RO)

Register 5.53. EFUSE_RD_KEY1_DATA5_REG (0x00D0)



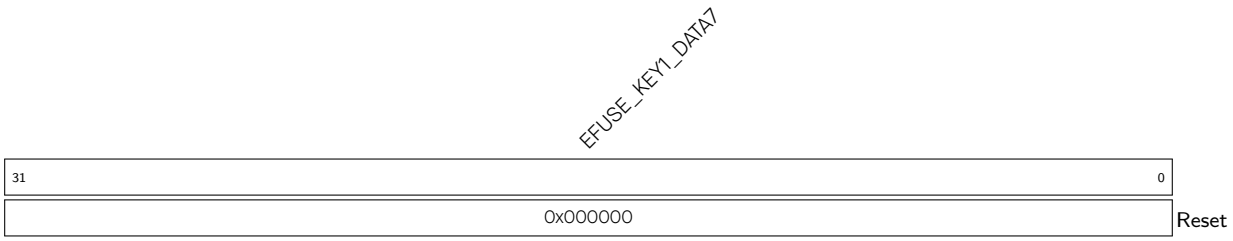
EFUSE_KEY1_DATA5 Represents the 5th 32 bits of KEY1. (RO)

Register 5.54. EFUSE_RD_KEY1_DATA6_REG (0x00D4)



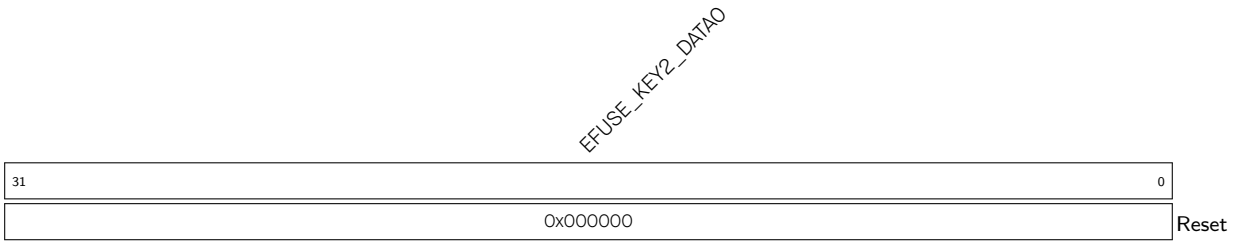
EFUSE_KEY1_DATA6 Represents the 6th 32 bits of KEY1. (RO)

Register 5.55. EFUSE_RD_KEY1_DATA7_REG (0x00D8)



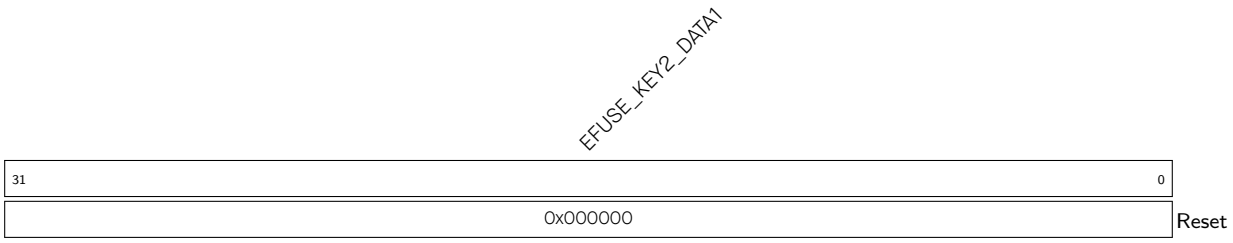
EFUSE_KEY1_DATA7 Represents the 7th 32 bits of KEY1. (RO)

Register 5.56. EFUSE_RD_KEY2_DATA0_REG (0x00DC)



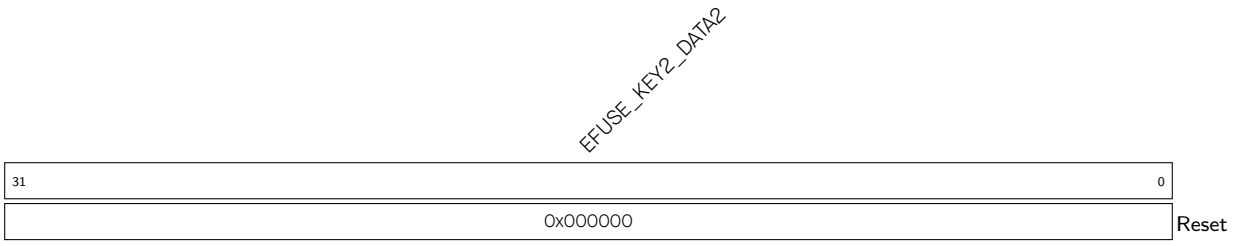
EFUSE_KEY2_DATA0 Represents the zeroth 32 bits of KEY2. (RO)

Register 5.57. EFUSE_RD_KEY2_DATA1_REG (0x00E0)



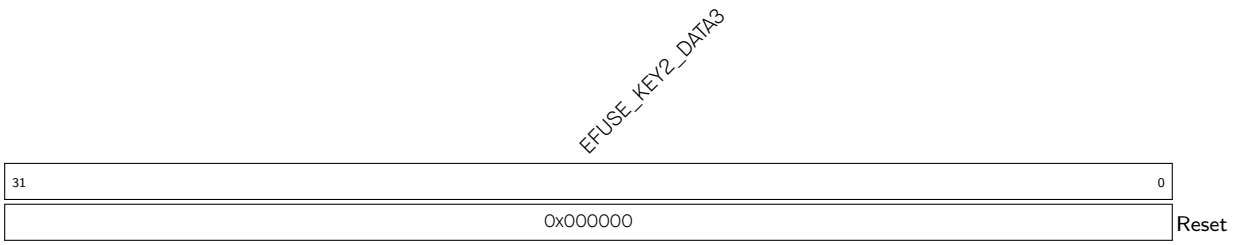
EFUSE_KEY2_DATA1 Represents the 1st 32 bits of KEY2. (RO)

Register 5.58. EFUSE_RD_KEY2_DATA2_REG (0x00E4)



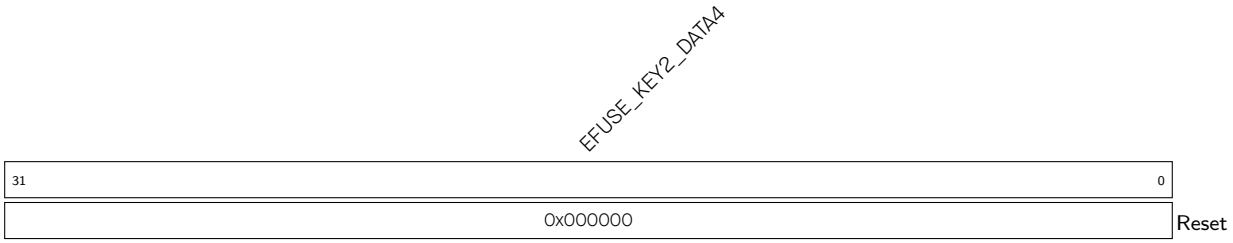
EFUSE_KEY2_DATA2 Represents the 2nd 32 bits of KEY2. (RO)

Register 5.59. EFUSE_RD_KEY2_DATA3_REG (0x00E8)



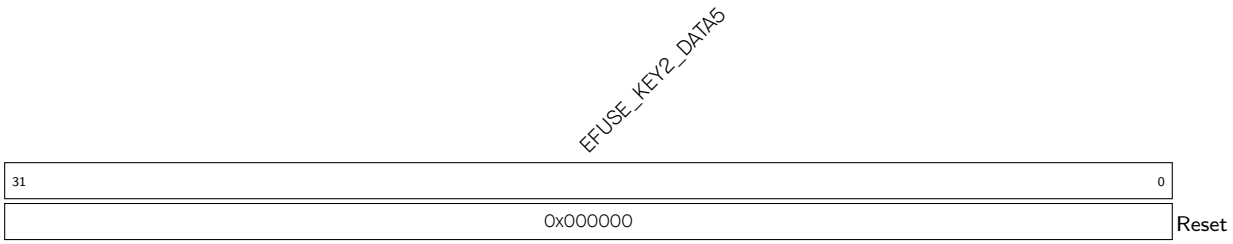
EFUSE_KEY2_DATA3 Represents the 3rd 32 bits of KEY2. (RO)

Register 5.60. EFUSE_RD_KEY2_DATA4_REG (0x00EC)



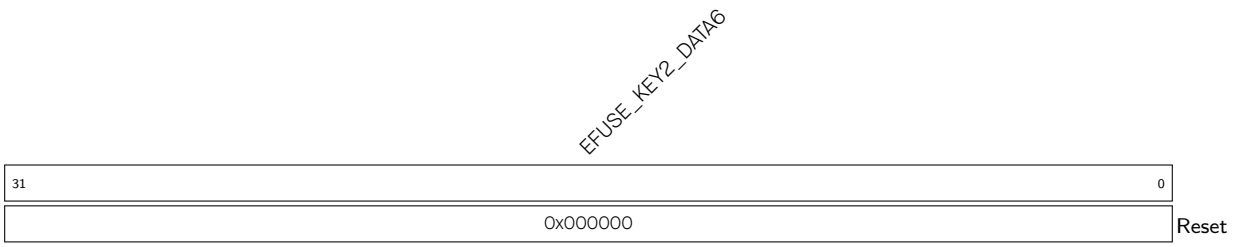
EFUSE_KEY2_DATA4 Represents the 4th 32 bits of KEY2. (RO)

Register 5.61. EFUSE_RD_KEY2_DATA5_REG (0x00F0)



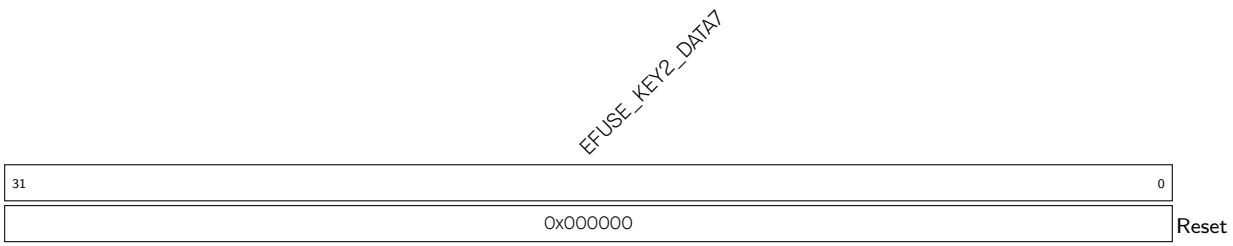
EFUSE_KEY2_DATA5 Represents the 5th 32 bits of KEY2. (RO)

Register 5.62. EFUSE_RD_KEY2_DATA6_REG (0x00F4)



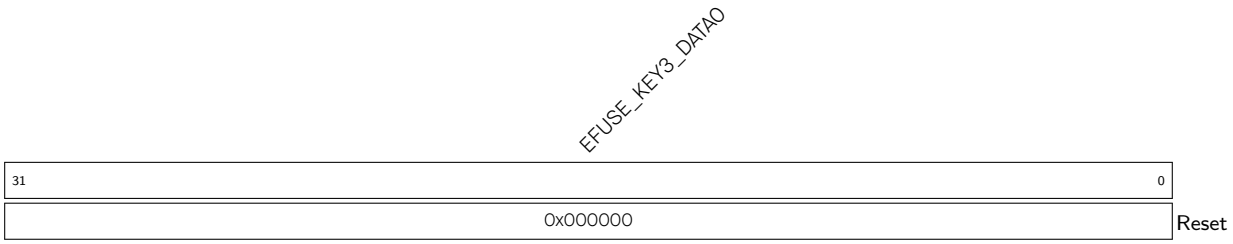
EFUSE_KEY2_DATA6 Represents the 6th 32 bits of KEY2. (RO)

Register 5.63. EFUSE_RD_KEY2_DATA7_REG (0x00F8)



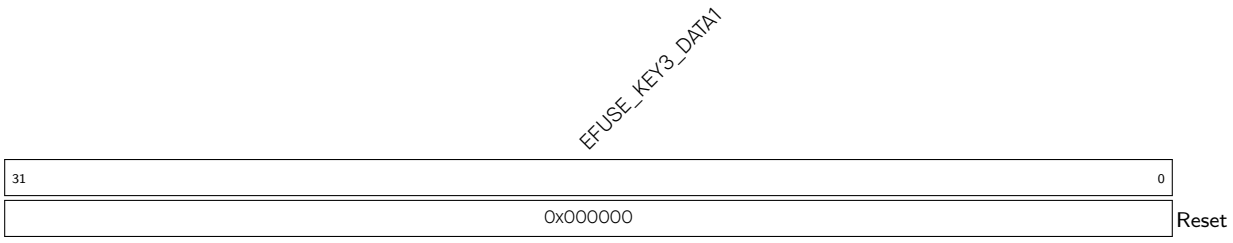
EFUSE_KEY2_DATA7 Represents the 7th 32 bits of KEY2. (RO)

Register 5.64. EFUSE_RD_KEY3_DATA0_REG (0x00FC)



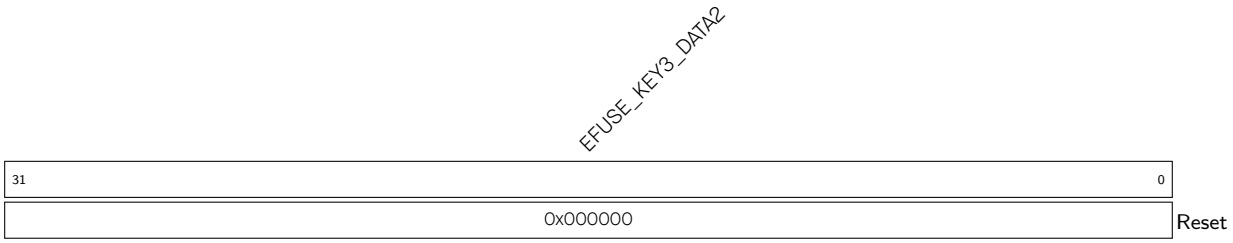
EFUSE_KEY3_DATA0 Represents the zeroth 32 bits of KEY3. (RO)

Register 5.65. EFUSE_RD_KEY3_DATA1_REG (0x0100)



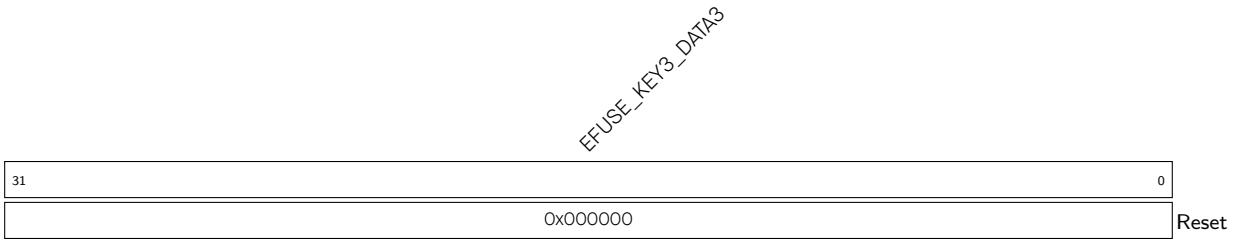
EFUSE_KEY3_DATA1 Represents the 1st 32 bits of KEY3. (RO)

Register 5.66. EFUSE_RD_KEY3_DATA2_REG (0x0104)



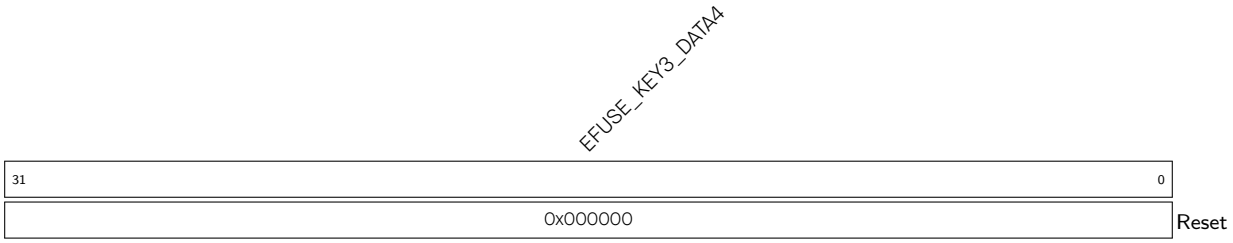
EFUSE_KEY3_DATA2 Represents the 2nd 32 bits of KEY3. (RO)

Register 5.67. EFUSE_RD_KEY3_DATA3_REG (0x0108)



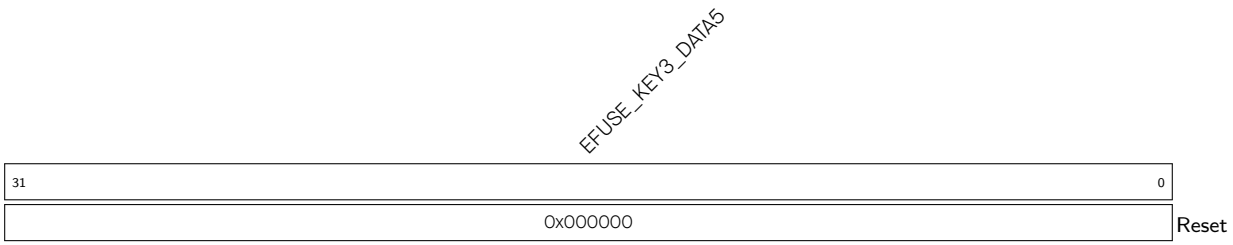
EFUSE_KEY3_DATA3 Represents the 3rd 32 bits of KEY3. (RO)

Register 5.68. EFUSE_RD_KEY3_DATA4_REG (0x010C)



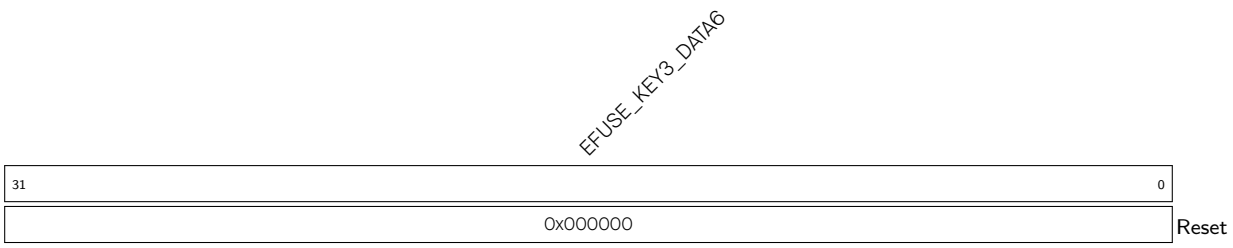
EFUSE_KEY3_DATA4 Represents the 4th 32 bits of KEY3. (RO)

Register 5.69. EFUSE_RD_KEY3_DATA5_REG (0x0110)



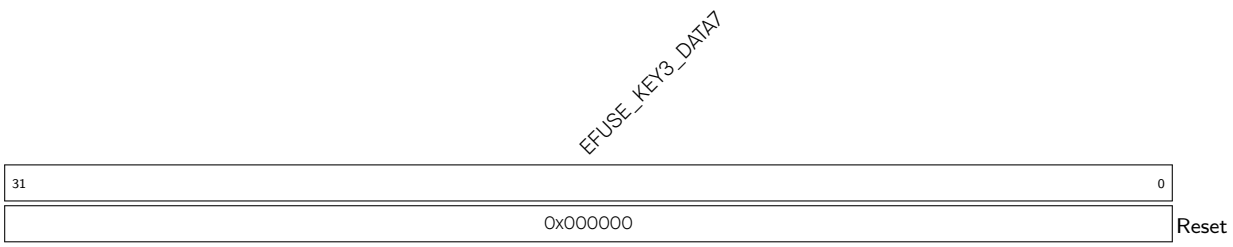
EFUSE_KEY3_DATA5 Represents the 5th 32 bits of KEY3. (RO)

Register 5.70. EFUSE_RD_KEY3_DATA6_REG (0x0114)



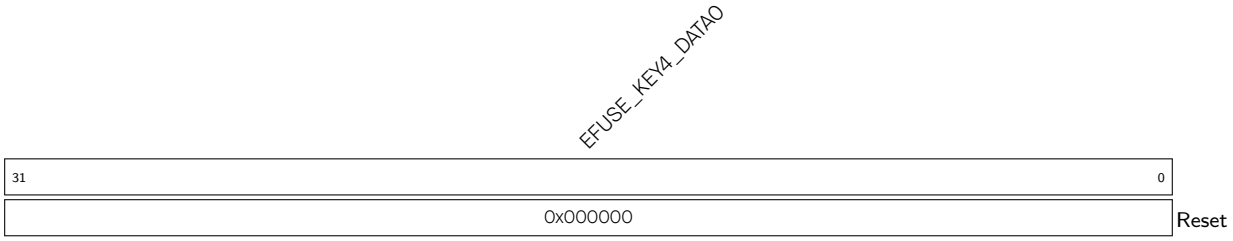
EFUSE_KEY3_DATA6 Represents the 6th 32 bits of KEY3. (RO)

Register 5.71. EFUSE_RD_KEY3_DATA7_REG (0x0118)



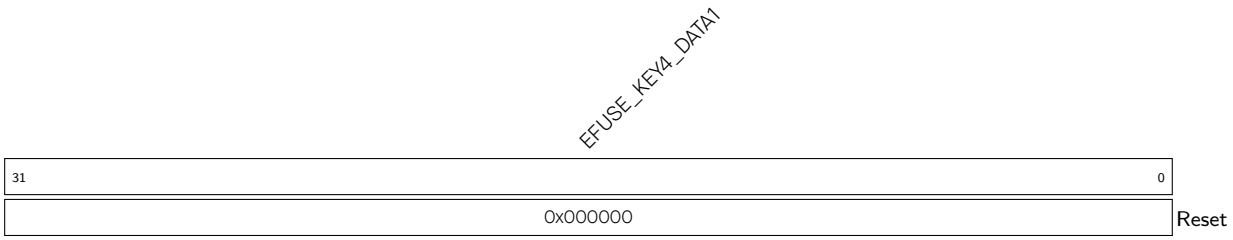
EFUSE_KEY3_DATA7 Represents the 7th 32 bits of KEY3. (RO)

Register 5.72. EFUSE_RD_KEY4_DATA0_REG (0x011C)



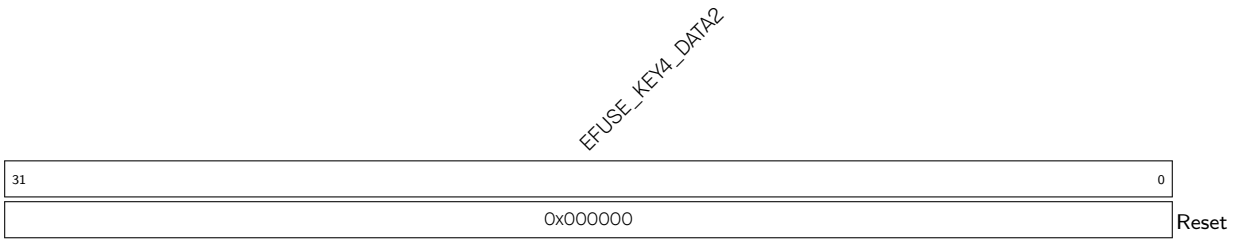
EFUSE_KEY4_DATA0 Represents the zeroth 32 bits of KEY4. (RO)

Register 5.73. EFUSE_RD_KEY4_DATA1_REG (0x0120)



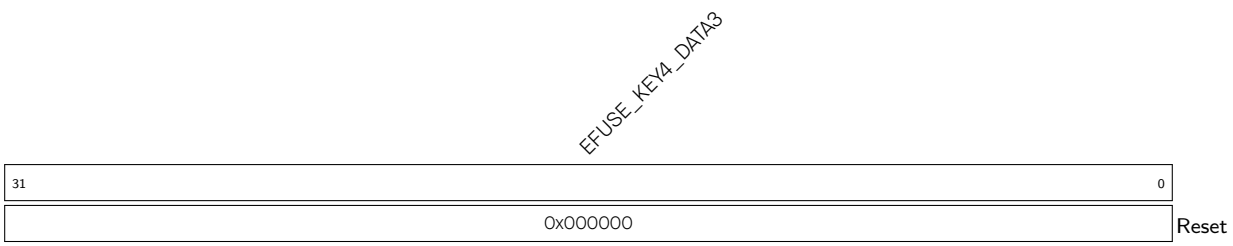
EFUSE_KEY4_DATA1 Represents the 1st 32 bits of KEY4. (RO)

Register 5.74. EFUSE_RD_KEY4_DATA2_REG (0x0124)



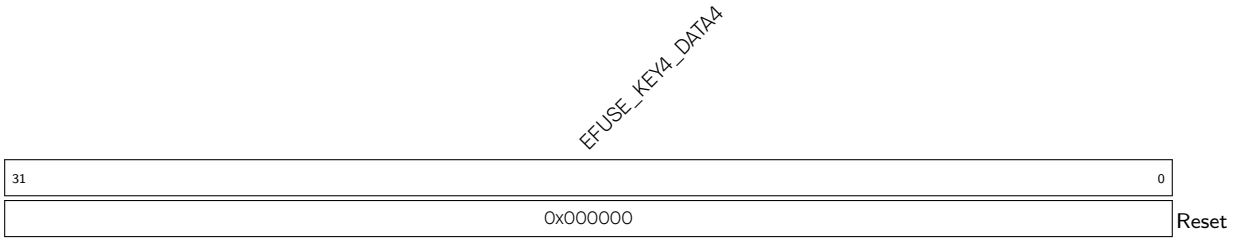
EFUSE_KEY4_DATA2 Represents the 2nd 32 bits of KEY4. (RO)

Register 5.75. EFUSE_RD_KEY4_DATA3_REG (0x0128)



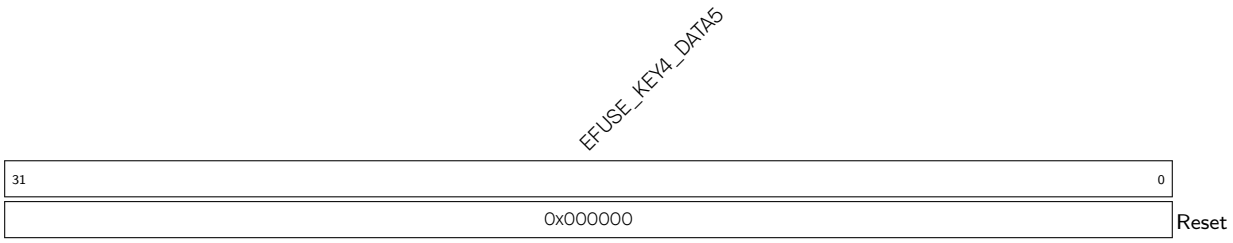
EFUSE_KEY4_DATA3 Represents the 3rd 32 bits of KEY4. (RO)

Register 5.76. EFUSE_RD_KEY4_DATA4_REG (0x012C)



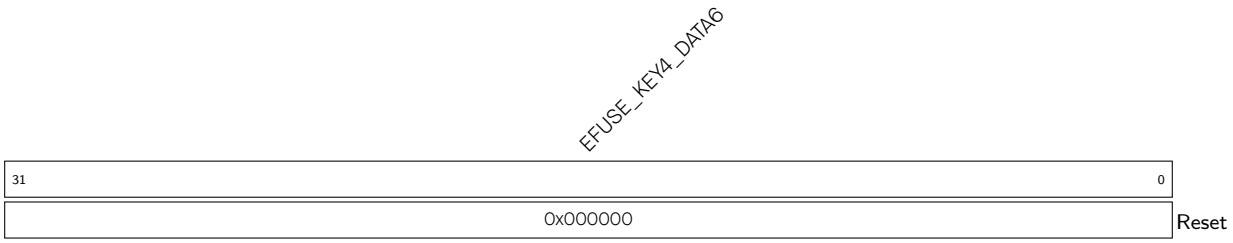
EFUSE_KEY4_DATA4 Represents the 4th 32 bits of KEY4. (RO)

Register 5.77. EFUSE_RD_KEY4_DATA5_REG (0x0130)



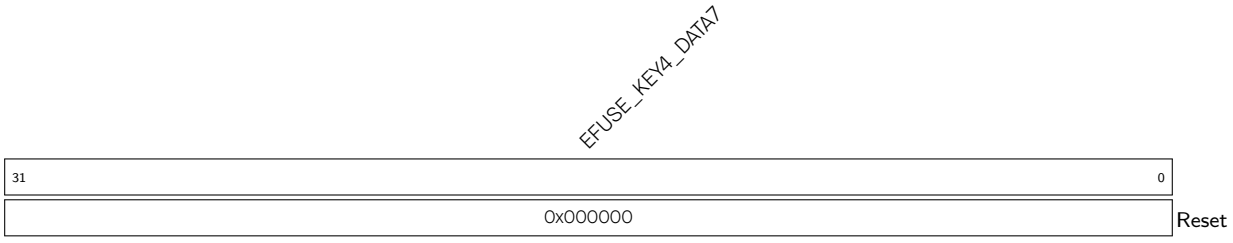
EFUSE_KEY4_DATA5 Represents the 5th 32 bits of KEY4. (RO)

Register 5.78. EFUSE_RD_KEY4_DATA6_REG (0x0134)



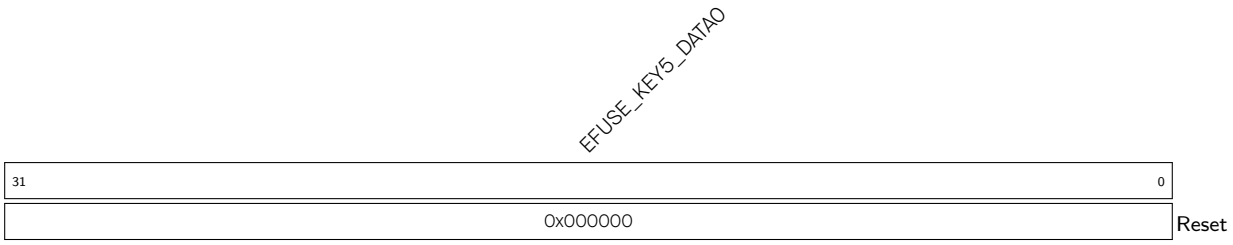
EFUSE_KEY4_DATA6 Represents the 6th 32 bits of KEY4. (RO)

Register 5.79. EFUSE_RD_KEY4_DATA7_REG (0x0138)



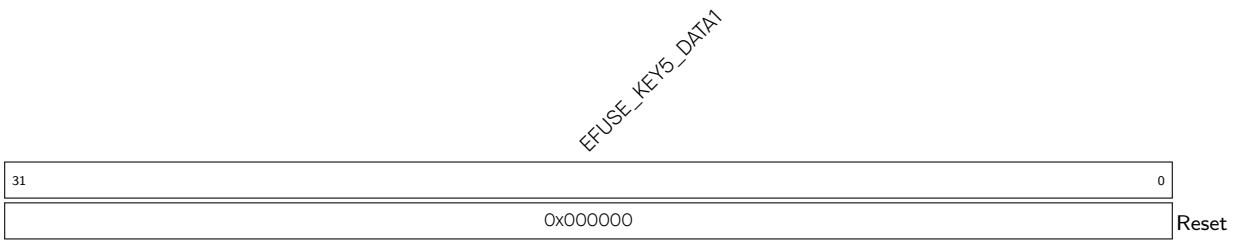
EFUSE_KEY4_DATA7 Represents the 7th 32 bits of KEY4. (RO)

Register 5.80. EFUSE_RD_KEY5_DATA0_REG (0x013C)



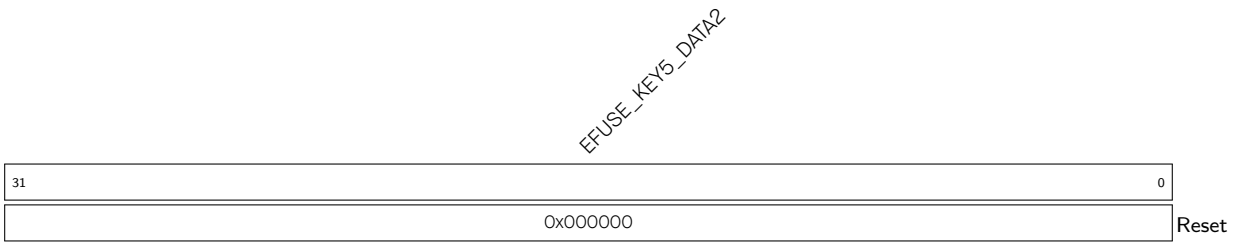
EFUSE_KEY5_DATA0 Represents the zeroth 32 bits of KEY5. (RO)

Register 5.81. EFUSE_RD_KEY5_DATA1_REG (0x0140)



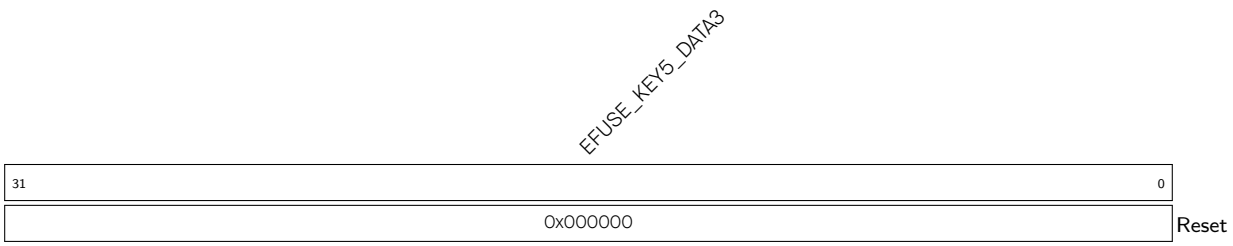
EFUSE_KEY5_DATA1 Represents the 1st 32 bits of KEY5. (RO)

Register 5.82. EFUSE_RD_KEY5_DATA2_REG (0x0144)



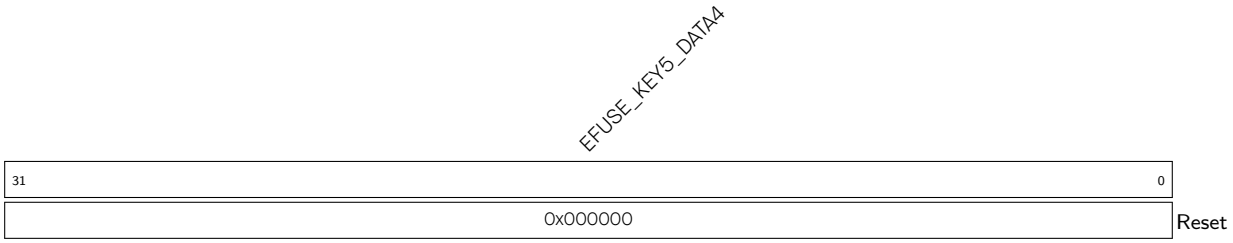
EFUSE_KEY5_DATA2 Represents the 2nd 32 bits of KEY5. (RO)

Register 5.83. EFUSE_RD_KEY5_DATA3_REG (0x0148)



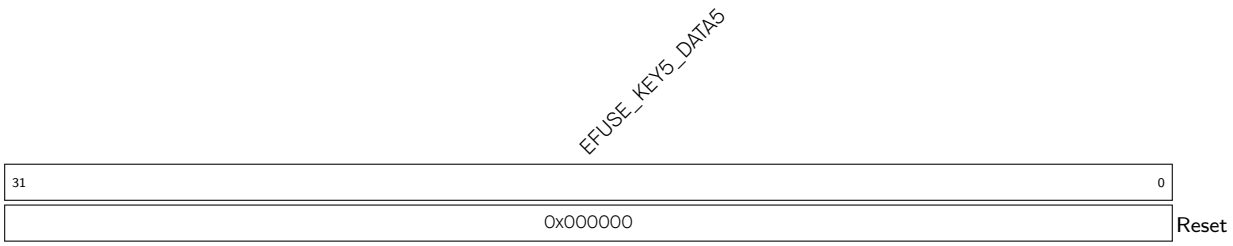
EFUSE_KEY5_DATA3 Represents the 3rd 32 bits of KEY5. (RO)

Register 5.84. EFUSE_RD_KEY5_DATA4_REG (0x014C)



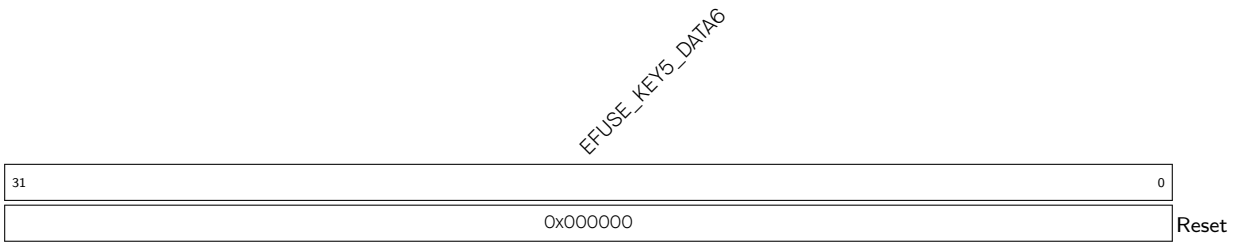
EFUSE_KEY5_DATA4 Represents the 4th 32 bits of KEY5. (RO)

Register 5.85. EFUSE_RD_KEY5_DATA5_REG (0x0150)



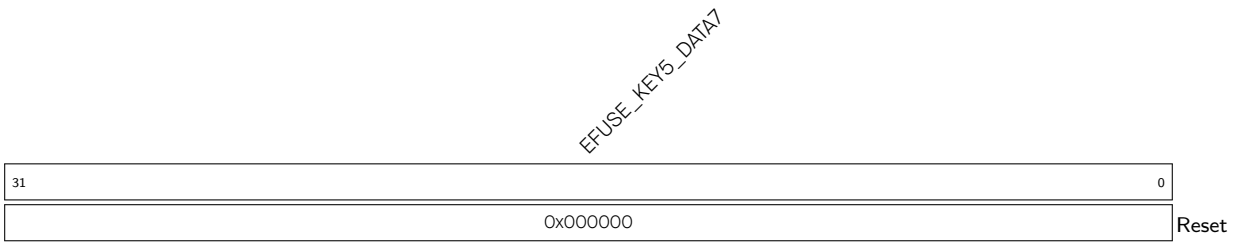
EFUSE_KEY5_DATA5 Represents the 5th 32 bits of KEY5. (RO)

Register 5.86. EFUSE_RD_KEY5_DATA6_REG (0x0154)



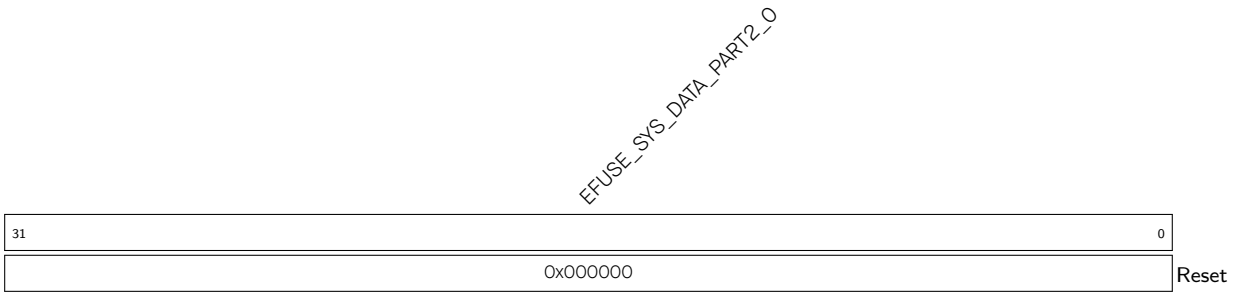
EFUSE_KEY5_DATA6 Represents the 6th 32 bits of KEY5. (RO)

Register 5.87. EFUSE_RD_KEY5_DATA7_REG (0x0158)



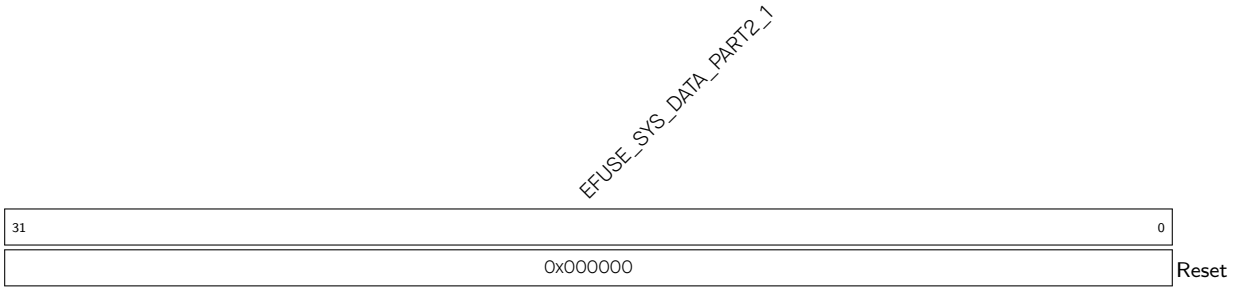
EFUSE_KEY5_DATA7 Represents the 7th 32 bits of KEY5. (RO)

Register 5.88. EFUSE_RD_SYS_PART2_DATA0_REG (0x015C)



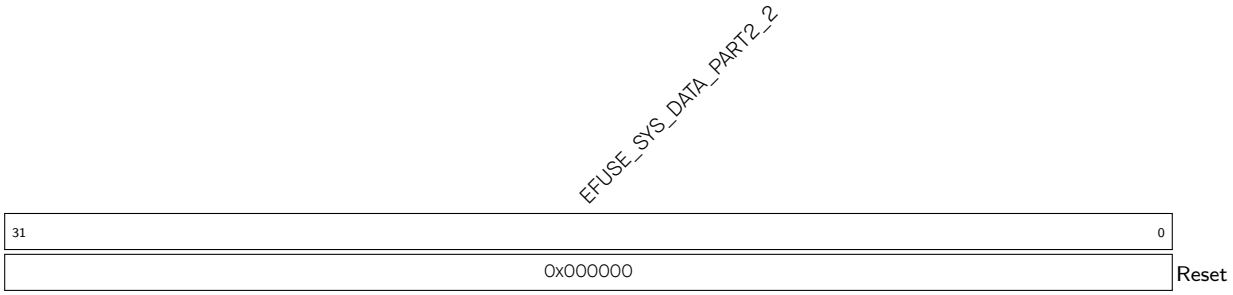
EFUSE_SYS_DATA_PART2_0 Represents the 0th 32 bits of the 2nd part of system data. (RO)

Register 5.89. EFUSE_RD_SYS_PART2_DATA1_REG (0x0160)



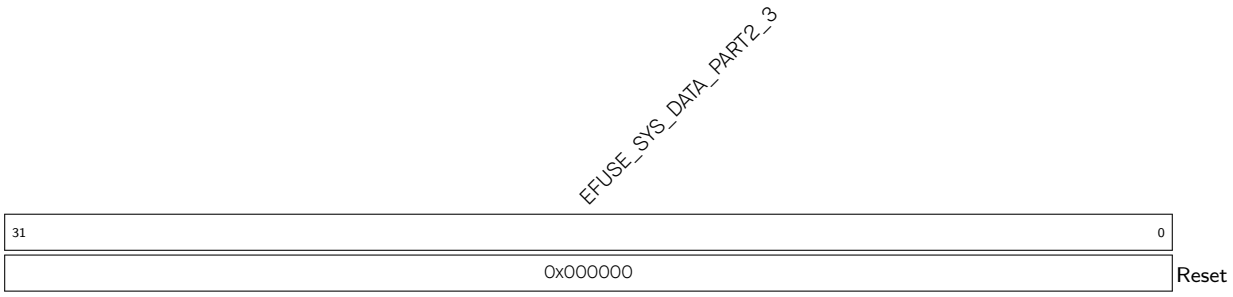
EFUSE_SYS_DATA_PART2_1 Represents the 1st 32 bits of the 2nd part of system data. (RO)

Register 5.90. EFUSE_RD_SYS_PART2_DATA2_REG (0x0164)



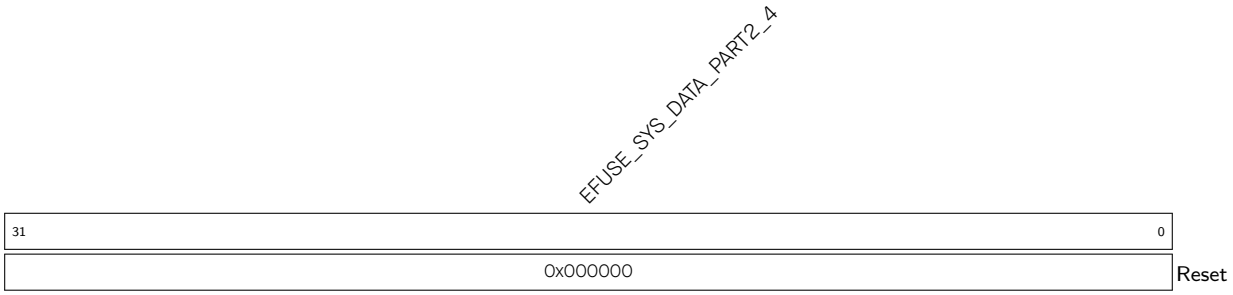
EFUSE_SYS_DATA_PART2_2 Represents the 2nd 32 bits of the 2nd part of system data. (RO)

Register 5.91. EFUSE_RD_SYS_PART2_DATA3_REG (0x0168)



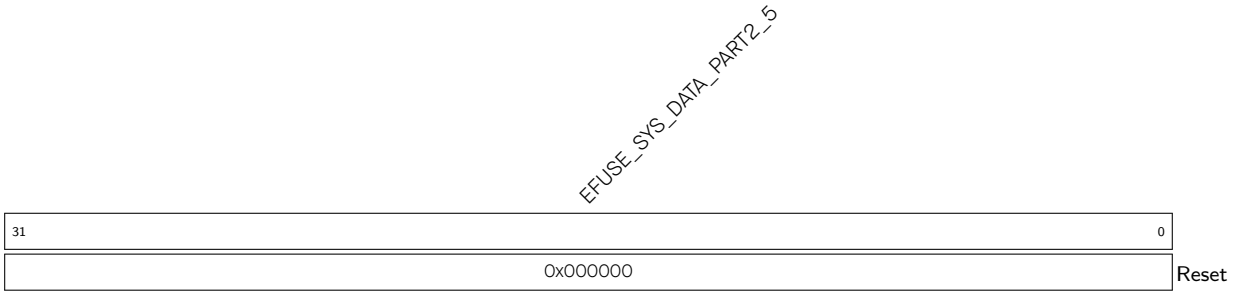
EFUSE_SYS_DATA_PART2_3 Represents the 3rd 32 bits of the 2nd part of system data. (RO)

Register 5.92. EFUSE_RD_SYS_PART2_DATA4_REG (0x016C)



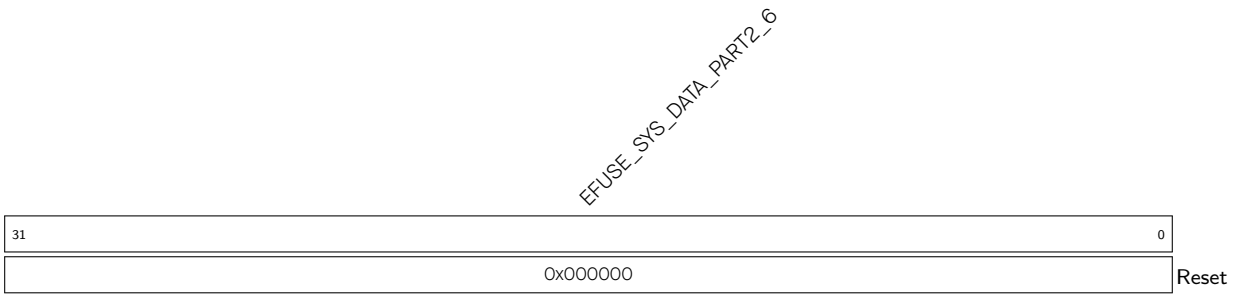
EFUSE_SYS_DATA_PART2_4 Represents the 4th 32 bits of the 2nd part of system data. (RO)

Register 5.93. EFUSE_RD_SYS_PART2_DATA5_REG (0x0170)



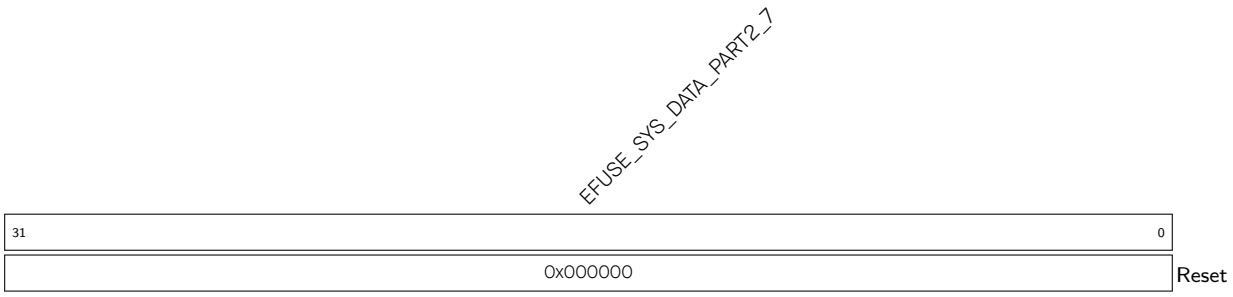
EFUSE_SYS_DATA_PART2_5 Represents the 5th 32 bits of the 2nd part of system data. (RO)

Register 5.94. EFUSE_RD_SYS_PART2_DATA6_REG (0x0174)



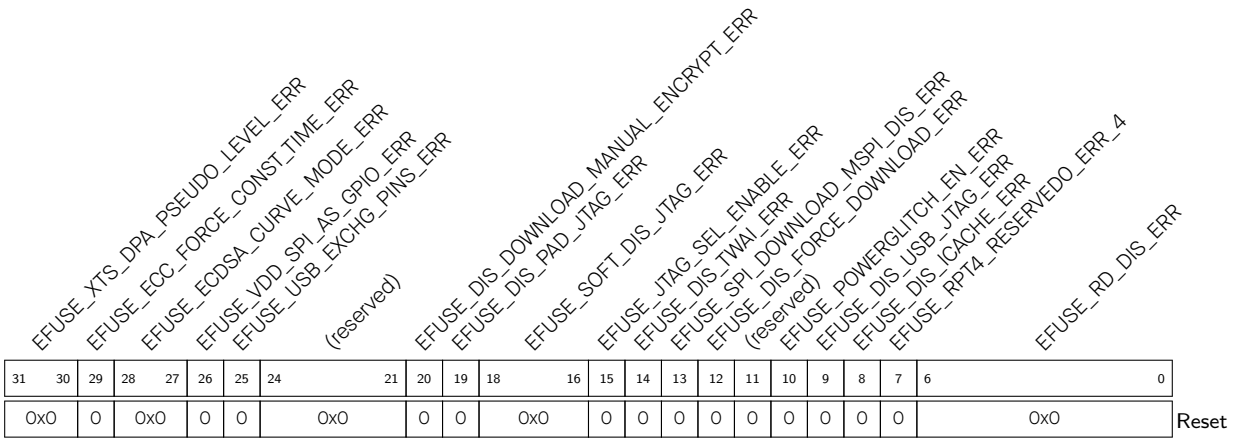
EFUSE_SYS_DATA_PART2_6 Represents the 6th 32 bits of the 2nd part of system data. (RO)

Register 5.95. EFUSE_RD_SYS_PART2_DATA7_REG (0x0178)



EFUSE_SYS_DATA_PART2_7 Represents the 7th 32 bits of the 2nd part of system data. (RO)

Register 5.96. EFUSE_RD_REPEAT_ERR0_REG (0x017C)



EFUSE_RD_DIS_ERR Any bit of this field being 1 represents a programming error of RD_DIS. (RO)

EFUSE_RPT4_RESERVED0_ERR_4 Reserved. (RO)

EFUSE_DIS_ICACHE_ERR This bit being 1 represents a programming error of DIS_ICACHE. (RO)

EFUSE_DIS_USB_JTAG_ERR This bit being 1 represents a programming error of DIS_USB_JTAG. (RO)

EFUSE_POWERGLITCH_EN_ERR This bit being 1 represents a programming error of POWERGLITCH_EN. (RO)

EFUSE_DIS_FORCE_DOWNLOAD_ERR This bit being 1 represents a programming error of DIS_FORCE_DOWNLOAD. (RO)

EFUSE_SPI_DOWNLOAD_MSPI_DIS_ERR This bit being 1 represents a programming error of SPI_DOWNLOAD_MSPI_DIS. (RO)

EFUSE_DIS_TWAI_ERR This bit being 1 represents a programming error of DIS_CAN. (RO)

EFUSE_JTAG_SEL_ENABLE_ERR This bit being 1 represents a programming error of JTAG_SEL_ENABLE. (RO)

EFUSE_SOFT_DIS_JTAG_ERR Any bit of this field being 1 represents a programming error of SOFT_DIS_JTAG. (RO)

EFUSE_DIS_PAD_JTAG_ERR This bit being 1 represents a programming error of DIS_PAD_JTAG. (RO)

EFUSE_DIS_DOWNLOAD_MANUAL_ENCRYPT_ERR This bit being 1 represents a programming error of DIS_DOWNLOAD_MANUAL_ENCRYPT. (RO)

Continued on the next page...

Register 5.96. EFUSE_RD_REPEAT_ERR0_REG (0x0080)

Continued from the previous page...

EFUSE_USB_EXCHG_PINS_ERR This bit being 1 represents a programming error of USB_EXCHG_PINS. (RO)

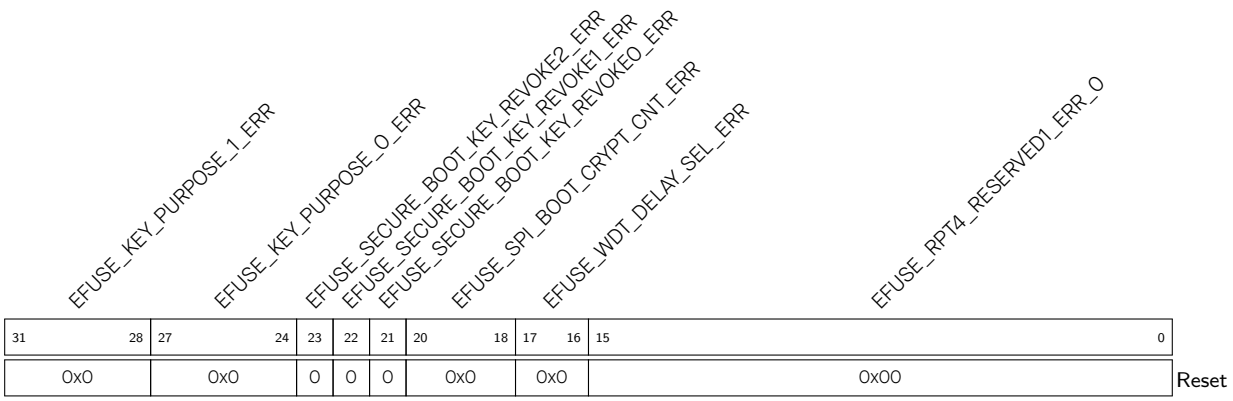
EFUSE_VDD_SPI_AS_GPIO_ERR This bit being 1 represents a programming error of VDD_SPI_AS_GPIO. (RO)

EFUSE_ECDSA_CURVE_MODE_ERR This bit being 1 represents a programming error of ECDSA_CURVE_MODE. (RO)

EFUSE_ECC_FORCE_CONST_TIME_ERR This bit being 1 represents a programming error of ECC_FORCE_CONST_TIME. (RO)

EFUSE_XTS_DPA_PSEUDO_LEVEL_ERR This bit being 1 represents a programming error of XTS_DPA_PSEUDO_LEVEL. (RO)

Register 5.97. EFUSE_RD_REPEAT_ERR1_REG (0x0180)



EFUSE_RPT4_RESERVED1_ERR_0 Reserved. (RO)

EFUSE_WDT_DELAY_SEL_ERR Any bit of this field being 1 represents a programming error of WDT_DELAY_SEL. (RO)

EFUSE_SPI_BOOT_CRYPT_CNT_ERR Any bit of this field being 1 represents a programming error of SPI_BOOT_CRYPT_CNT. (RO)

EFUSE_SECURE_BOOT_KEY_REVOKE0_ERR This bit being 1 represents a programming error of SECURE_BOOT_KEY_REVOKE0. (RO)

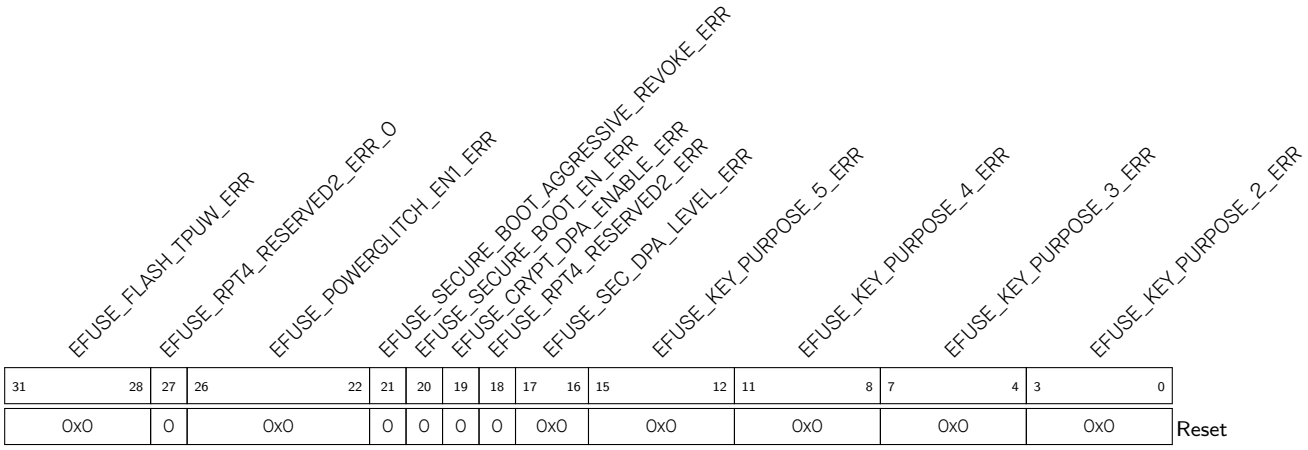
EFUSE_SECURE_BOOT_KEY_REVOKE1_ERR This bit being 1 represents a programming error of SECURE_BOOT_KEY_REVOKE1. (RO)

EFUSE_SECURE_BOOT_KEY_REVOKE2_ERR This bit being 1 represents a programming error of SECURE_BOOT_KEY_REVOKE2. (RO)

EFUSE_KEY_PURPOSE_0_ERR Any bit of this field being 1 represents a programming error of KEY_PURPOSE_0. (RO)

EFUSE_KEY_PURPOSE_1_ERR Any bit of this field being 1 represents a programming error of KEY_PURPOSE_1. (RO)

Register 5.98. EFUSE_RD_REPEAT_ERR2_REG (0x0184)



EFUSE_KEY_PURPOSE_2_ERR Any bit of this field being 1 represents a programming error of KEY_PURPOSE_2. (RO)

EFUSE_KEY_PURPOSE_3_ERR Any bit of this field being 1 represents a programming error of KEY_PURPOSE_3. (RO)

EFUSE_KEY_PURPOSE_4_ERR Any bit of this field being 1 represents a programming error of KEY_PURPOSE_4. (RO)

EFUSE_KEY_PURPOSE_5_ERR Any bit of this field being 1 represents a programming error of KEY_PURPOSE_5. (RO)

EFUSE_SEC_DPA_LEVEL_ERR Any bit of this field being 1 represents a programming error of SEC_DPA_LEVEL. (RO)

EFUSE_RPT4_RESERVED2_ERR_1 Reserved. (RO)

EFUSE_CRYPT_DPA_ENABLE_ERR This bit being 1 represents a programming error of CRYPT_DPA_ENABLE. (RO)

EFUSE_SECURE_BOOT_EN_ERR This bit being 1 represents a programming error of SECURE_BOOT_EN. (RO)

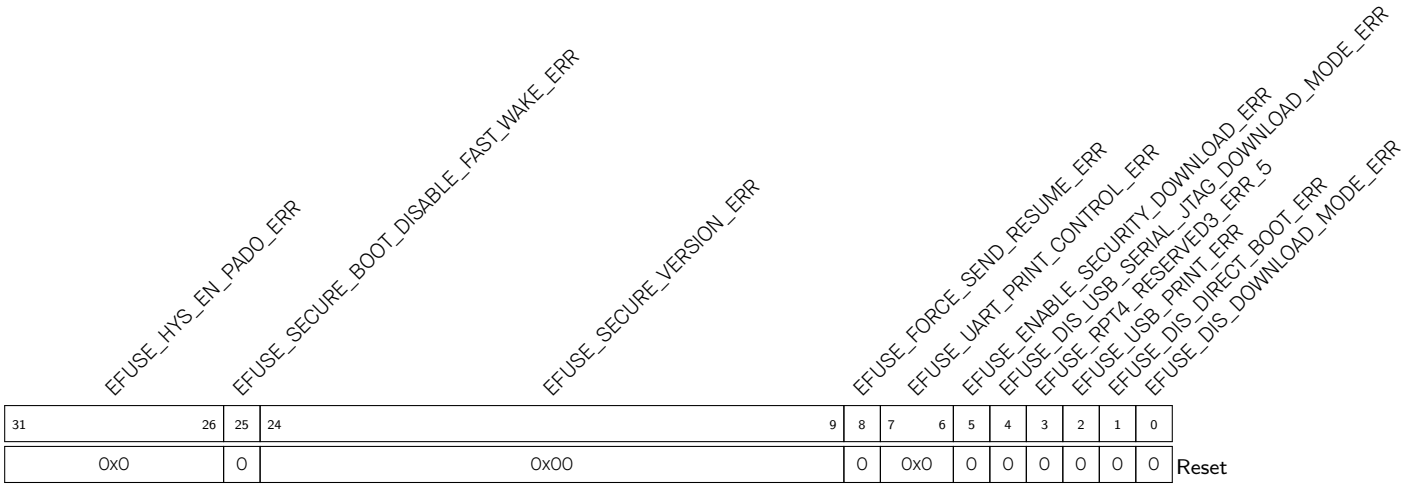
EFUSE_SECURE_BOOT_AGGRESSIVE_REVOKE_ERR This bit being 1 represents a programming error of SECURE_BOOT_AGGRESSIVE_REVOKE. (RO)

EFUSE_POWERGLITCH_EN1_ERR This bit being 1 represents a programming error of POWERGLITCH_EN1. (RO)

EFUSE_RPT4_RESERVED2_ERR_0 Reserved. (RO)

EFUSE_FLASH_TPUW_ERR Any bit of this field being 1 represents a programming error of FLASH_TPUW. (RO)

Register 5.99. EFUSE_RD_REPEAT_ERR3_REG (0x0188)



EFUSE_DIS_DOWNLOAD_MODE_ERR This bit being 1 represents a programming error of DIS_DOWNLOAD_MODE. (RO)

EFUSE_DIS_DIRECT_BOOT_ERR This bit being 1 represents a programming error of DIS_DIRECT_BOOT. (RO)

EFUSE_USB_PRINT_ERR This bit being 1 represents a programming error of UART_PRINT_CHANNEL. (RO)

EFUSE_RPT4_RESERVED3_ERR_5 Reserved. (RO)

EFUSE_DIS_USB_SERIAL_JTAG_DOWNLOAD_MODE_ERR This bit being 1 represents a programming error of DIS_USB_SERIAL_JTAG_DOWNLOAD_MODE. (RO)

EFUSE_ENABLE_SECURITY_DOWNLOAD_ERR This bit being 1 represents a programming error of ENABLE_SECURITY_DOWNLOAD. (RO)

EFUSE_UART_PRINT_CONTROL_ERR Any bit of this field being 1 represents a programming error of UART_PRINT_CONTROL. (RO)

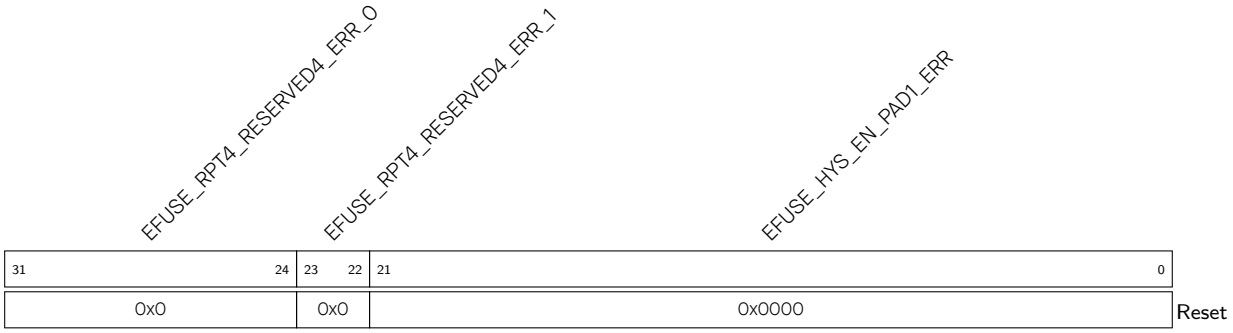
EFUSE_FORCE_SEND_RESUME_ERR This bit being 1 represents a programming error of FORCE_SEND_RESUME. (RO)

EFUSE_SECURE_VERSION_ERR Any bit of this field being 1 represents a programming error of SECURE_VERSION. (RO)

EFUSE_SECURE_BOOT_DISABLE_FAST_WAKE_ERR This bit being 1 represents a programming error of SECURE_BOOT_DISABLE_FAST_WAKE. (RO)

EFUSE_HYS_EN_PADO_ERR Any bit of this field being 1 represents a programming error of HYS_EN_PADO. (RO)

Register 5.100. EFUSE_RD_REPEAT_ERR4_REG (0x018C)



EFUSE_HYS_EN_PAD1_ERR Any bit of this field being 1 represents a programming error of HYS_EN_PAD1. (RO)

EFUSE_RPT4_RESERVED4_ERR_1 Reserved. (RO)

EFUSE_RPT4_RESERVED4_ERR_0 Reserved. (RO)

Register 5.101. EFUSE_RD_RS_ERR0_REG (0x01C0)

EFUSE_MAC_SPI_8M_ERR_NUM Represents the number of error bytes. (RO)

EFUSE_MAC_SPI_8M_FAIL Represents whether the programming of MAC_SPI_8M user data failed.

0: No failure and that the data of MAC_SPI_8M is reliable

1: Programming user data failed and the number of error bytes is over 6. (RO)

EFUSE_SYS_PART1_NUM Represents the number of error bytes. (RO)

EFUSE_SYS_PART1_FAIL Represents whether programming system part1 data failed.

0: No failure and the data of system part1 is reliable.

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

(RO)

EFUSE_USR_DATA_ERR_NUM Represents the number of error bytes. (RO)

EFUSE_USR_DATA_FAIL Represents whether programming user data failed.

0: No failure and the user data is reliable.

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

(RO)

EFUSE_KEY0_ERR_NUM Represents the number of error bytes. (RO)

EFUSE_KEY0_FAIL Represents whether programming key0 data failed.

0: No failure and the data of key0 is reliable.

1: Programming key0 failed and the number of error bytes is over 6.

(RO)

EFUSE_KEY1_ERR_NUM Represents the number of error bytes. (RO)

EFUSE_KEY1_FAIL Represents whether programming key1 data failed.

0: No failure and the data of key1 is reliable.

1: Programming key1 failed and the number of error bytes is over 6.

(RO)

EFUSE_KEY2_ERR_NUM Represents the number of error bytes. (RO)

EFUSE_KEY2_FAIL Represents whether programming key2 data failed.

0: No failure and the data of key2 is reliable.

1: Programming key2 failed and the number of error bytes is over 6.

(RO)

Continued on the next page...

Register 5.101. EFUSE_RD_RS_ERR0_REG (0x01C0)

Continued from the previous page...

EFUSE_KEY3_ERR_NUM Represents the number of error bytes. (RO)

EFUSE_KEY3_FAIL Represents whether programming key3 data failed.

0: No failure and the data of key3 is reliable.

1: Programming key3 failed and the number of error bytes is over 6.

(RO)

EFUSE_KEY4_ERR_NUM Represents the number of error bytes. (RO)

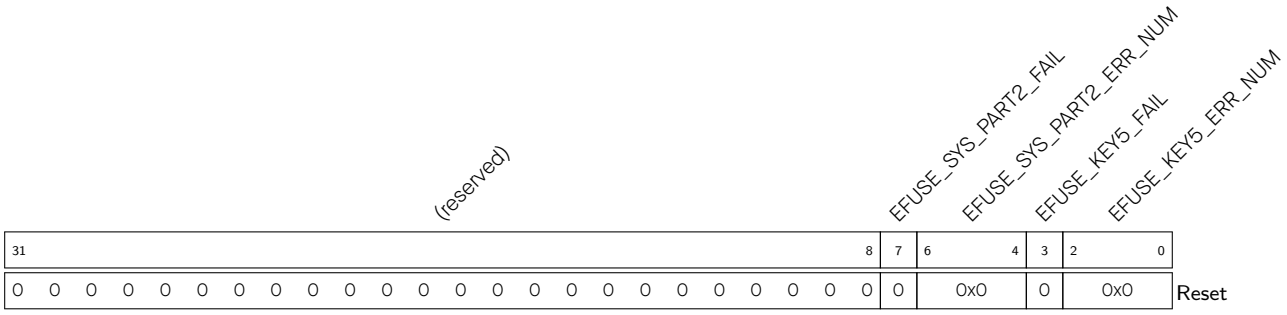
EFUSE_KEY4_FAIL Represents whether programming key4 data failed.

0: No failure and the data of key4 is reliable.

1: Programming key4 failed and the number of error bytes is over 6.

(RO)

Register 5.102. EFUSE_RD_RS_ERR1_REG (0x01C4)



EFUSE_KEY5_ERR_NUM Represents the number of error bytes. (RO)

EFUSE_KEY5_FAIL Represents whether programming key5 data failed.

0: No failure and the data of key5 is reliable.

1: Programming key5 failed and the number of error bytes is over 6.

(RO)

EFUSE_SYS_PART2_ERR_NUM Represents the number of error bytes. (RO)

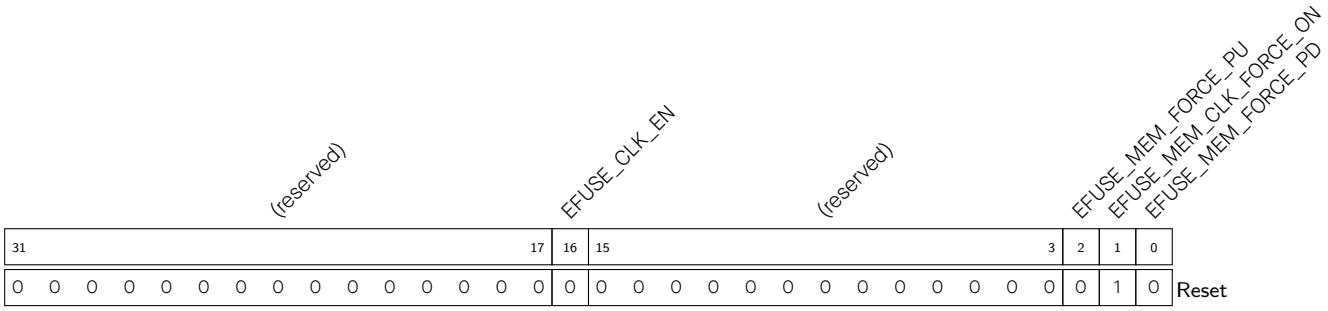
EFUSE_SYS_PART2_FAIL Represents whether programming system part2 data failed.

0: No failure and the data of system part2 is reliable.

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

(RO)

Register 5.103. 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)

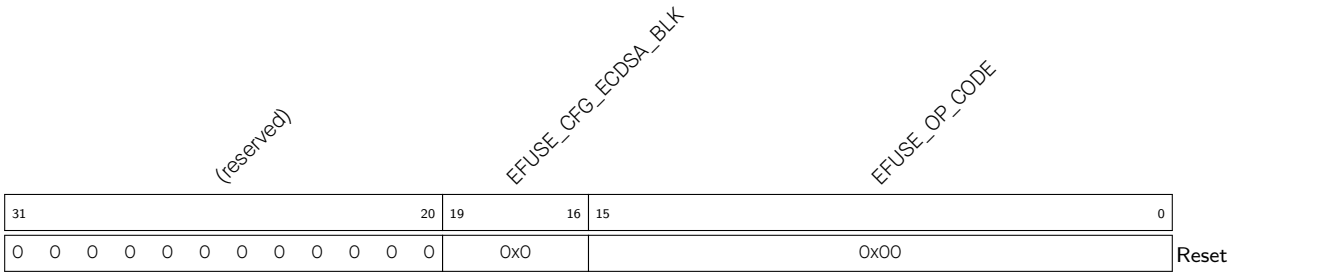
EFUSE_CLK_EN Configures whether to force enable eFuse register configuration clock signal.

1: Force

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

(R/W)

Register 5.104. EFUSE_CONF_REG (0x01CC)



EFUSE_OP_CODE Configures operation command type.

0x5A5A: Programming operation command

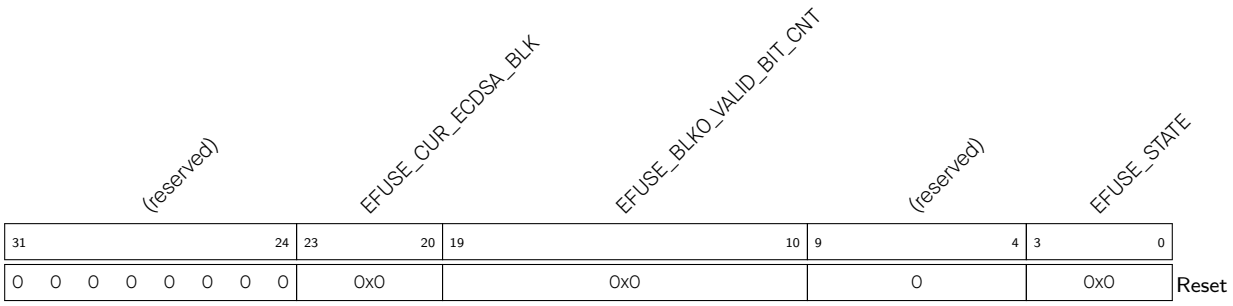
0x5AA5: Read operation command

Other values: No effect

(R/W)

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

Register 5.105. EFUSE_STATUS_REG (0x01D0)

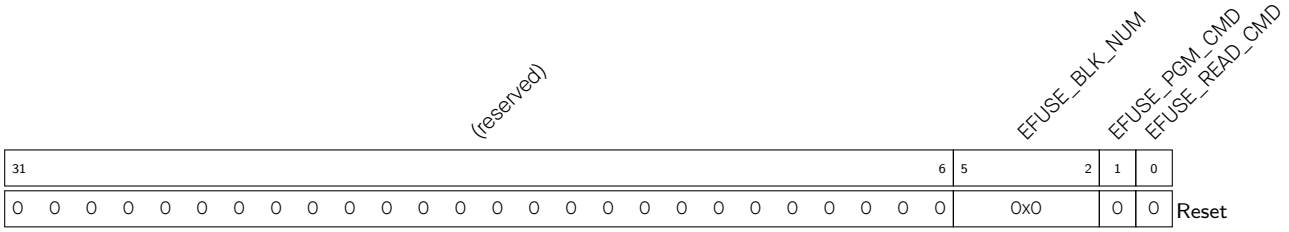


EFUSE_STATE Represents the state of the eFuse state machine. (RO)

EFUSE_BLK0_VALID_BIT_CNT Represents the number of block valid bit. (RO)

EFUSE_CUR_ECDSA_BLK Represents which block is used for ECDSA_DS key output. (RO)

Register 5.106. 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.

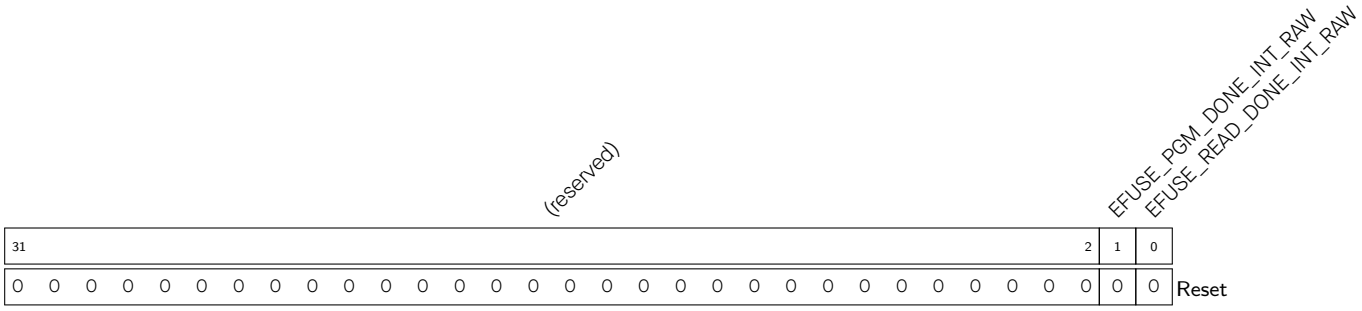
1: Send

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)

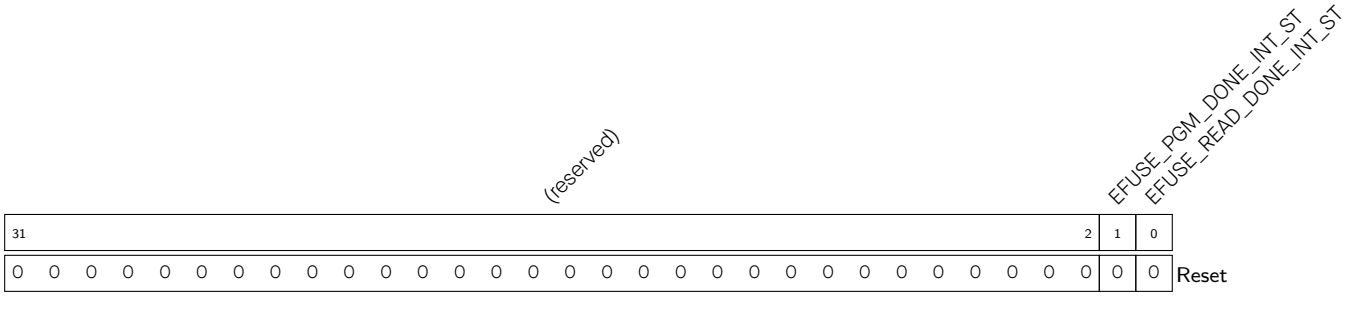
Register 5.107. EFUSE_INT_RAW_REG (0x01D8)



EFUSE_READ_DONE_INT_RAW The raw interrupt status of read_done. (R/SS/WTC)

EFUSE_PGM_DONE_INT_RAW The raw interrupt status of pgm_done. (R/SS/WTC)

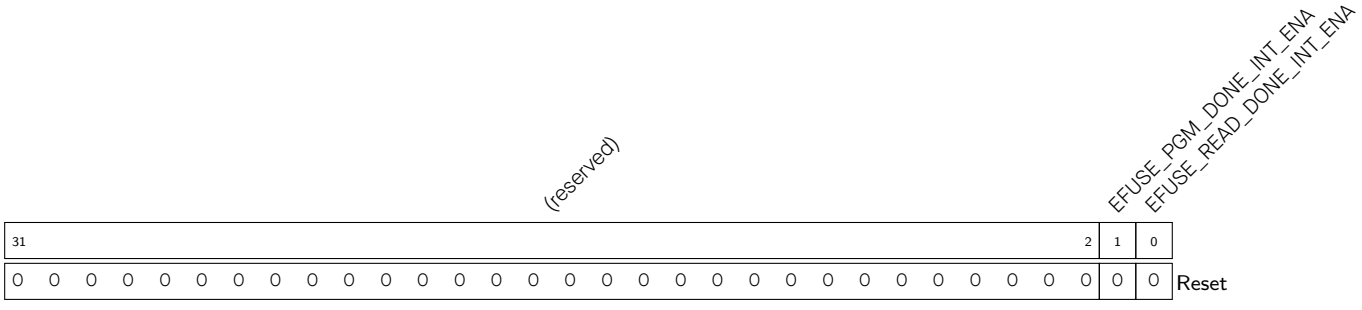
Register 5.108. EFUSE_INT_ST_REG (0x01DC)



EFUSE_READ_DONE_INT_ST The masked interrupt status of read_done. (RO)

EFUSE_PGM_DONE_INT_ST The masked interrupt status of pgm_done. (RO)

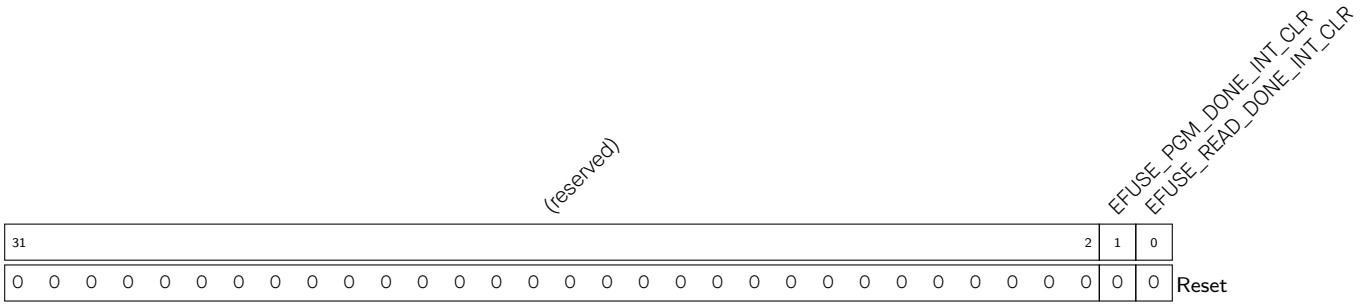
Register 5.109. EFUSE_INT_ENA_REG (0x01E0)



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)

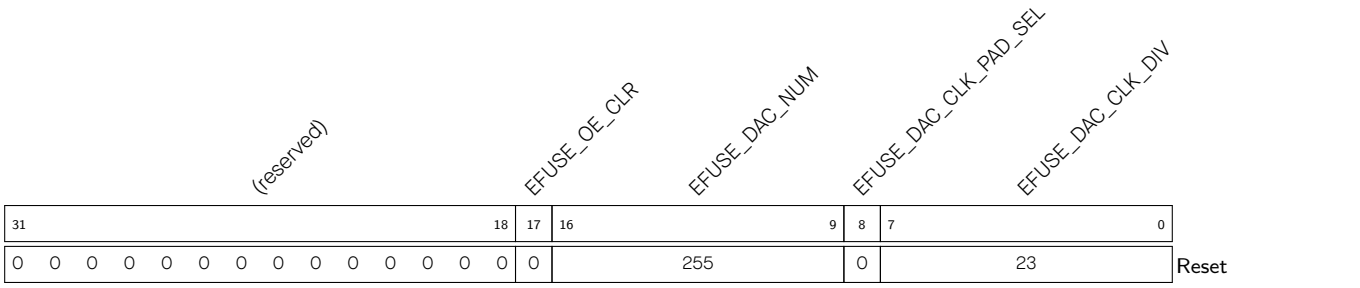
Register 5.110. EFUSE_INT_CLR_REG (0x01E4)



EFUSE_READ_DONE_INT_CLR Write 1 to clear read_done interrupt. (WT)

EFUSE_PGM_DONE_INT_CLR Write 1 to clear pgm_done interrupt. (WT)

Register 5.111. EFUSE_DAC_CONF_REG (0x01E8)



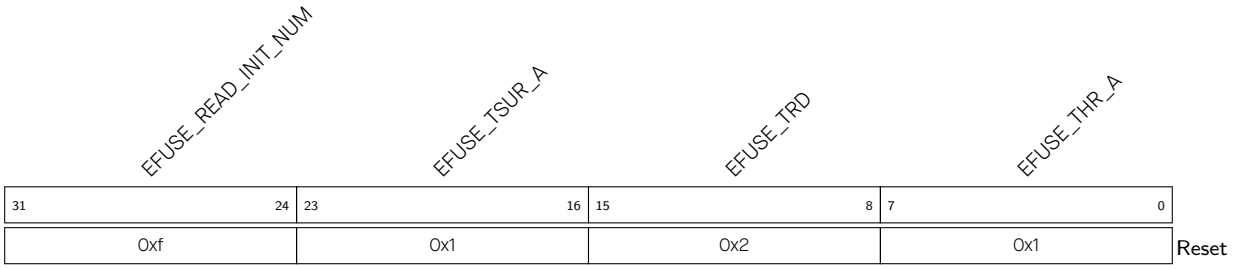
EFUSE_DAC_CLK_DIV Configures the division factor of the rising clock of the programming voltage. (R/W)

EFUSE_DAC_CLK_PAD_SEL Don't care. (R/W)

EFUSE_DAC_NUM Configures the rising period of the programming voltage. Measurement unit: Divided clock frequency by EFUSE_DAC_CLK_DIV. (R/W)

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

Register 5.112. EFUSE_RD_TIM_CONF_REG (0x01EC)



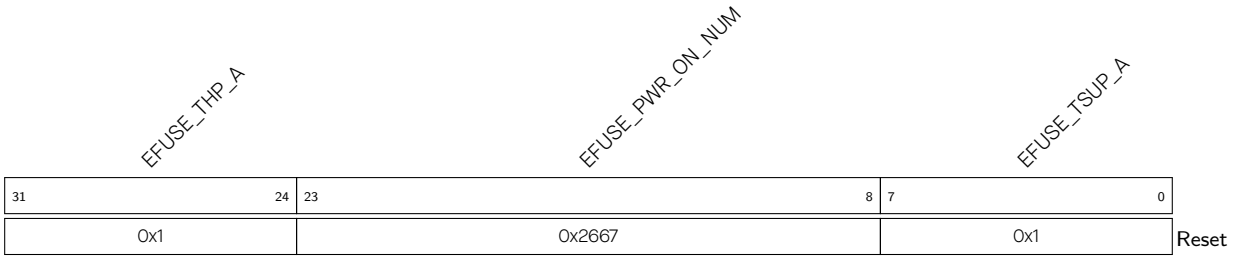
EFUSE_THR_A Configures the read hold time. Measurement unit: One cycle of the eFuse core clock. (R/W)

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 5.113. EFUSE_WR_TIM_CONF1_REG (0x01F0)

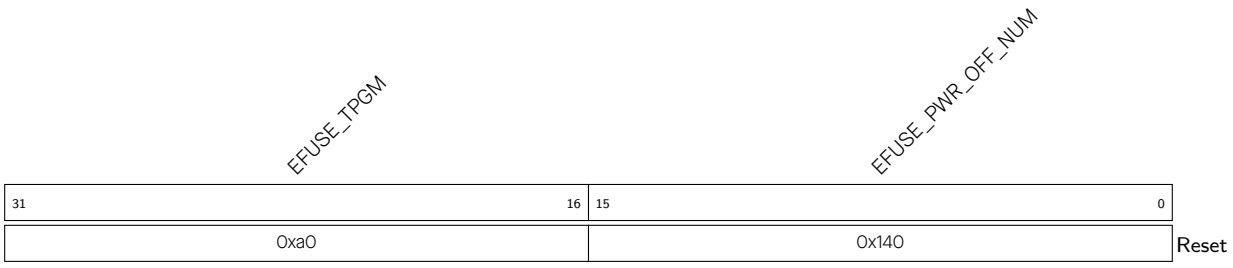


EFUSE_TSUP_A Configures the programming setup time. Measurement unit: One cycle of the eFuse core clock. (R/W)

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)

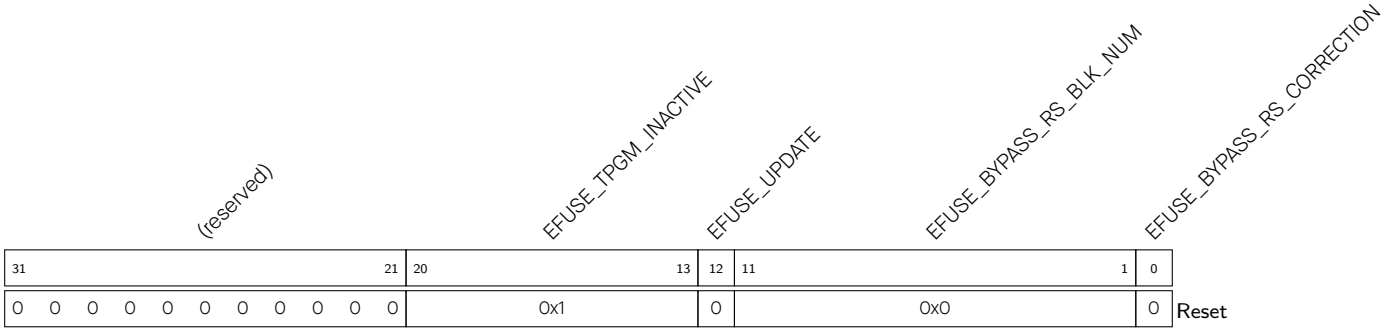
Register 5.114. EFUSE_WR_TIM_CONF2_REG (0x01F4)



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 5.115. EFUSE_WR_TIM_CONF0_RS_BYPASS_REG (0x01F8)



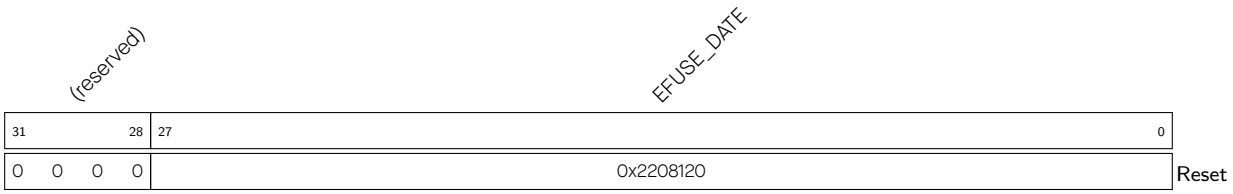
EFUSE_BYPASS_RS_CORRECTION Configures whether to bypass the Reed-Solomon correction step. (R/W)

EFUSE_BYPASS_RS_BLK_NUM Configures block number of programming twice 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 5.116. EFUSE_DATE_REG (0x01FC)



EFUSE_DATE Version control register. (R/W)

Part III

System Component

Encompassing a range of system-level functionalities, this part describes components related to system boot, clocks, GPIO, interrupt handling, event task matrix, low-power management, timers, watchdogs, permission control, various system registers, and debug assist.

Chapter 6

IO MUX and GPIO Matrix (GPIO, IO MUX)

6.1 Overview

The ESP32-H2 chip features 19 GPIO pins. Each pin can be used as a general-purpose I/O, or be connected to an internal peripheral signal. Through GPIO matrix and IO MUX, peripheral input signals can be from any IO pins, and peripheral output signals can be routed to any IO pins. Together these modules provide highly configurable I/O.

Note that the GPIO pins are numbered from GPIO0 ~ GPIO5, GPIO8 ~ GPIO14, and GPIO22 ~ GPIO27.

6.2 Features

GPIO matrix has the following features:

- A full-switching matrix between the peripheral input/output signals and the GPIO pins.
- 78 peripheral input signals sourced from the input of any GPIO pins.
- 99 peripheral output signals routed to the output of any GPIO pins.
- Signal synchronization for peripheral inputs based on **IO MUX operating clock**. For more information about the operating clock of IO MUX, please refer to Section 7 *Reset and Clock*.
- GPIO Filter hardware for input signal filtering.
- Glitch Filter hardware for second time filtering on the input signal.
- Sigma delta modulated (SDM) output.
- GPIO simple input and output.

IO MUX has the following features:

- Better high-frequency digital performance achieved by some digital signals (SPI, JTAG, UART) bypassing GPIO matrix. In this case, IO MUX is used to connect these pins directly to peripherals.
- A configuration register IO_MUX_GPIO*n*_REG provided for each GPIO pin. The pin can be configured to
  - perform GPIO function routed by GPIO matrix;
  - or perform direct connection bypassing GPIO matrix.

6.3 Architectural Overview

Figure 6.3-1 shows in details how GPIO matrix and IO MUX route signals from pins to peripherals, and from peripherals to pins.

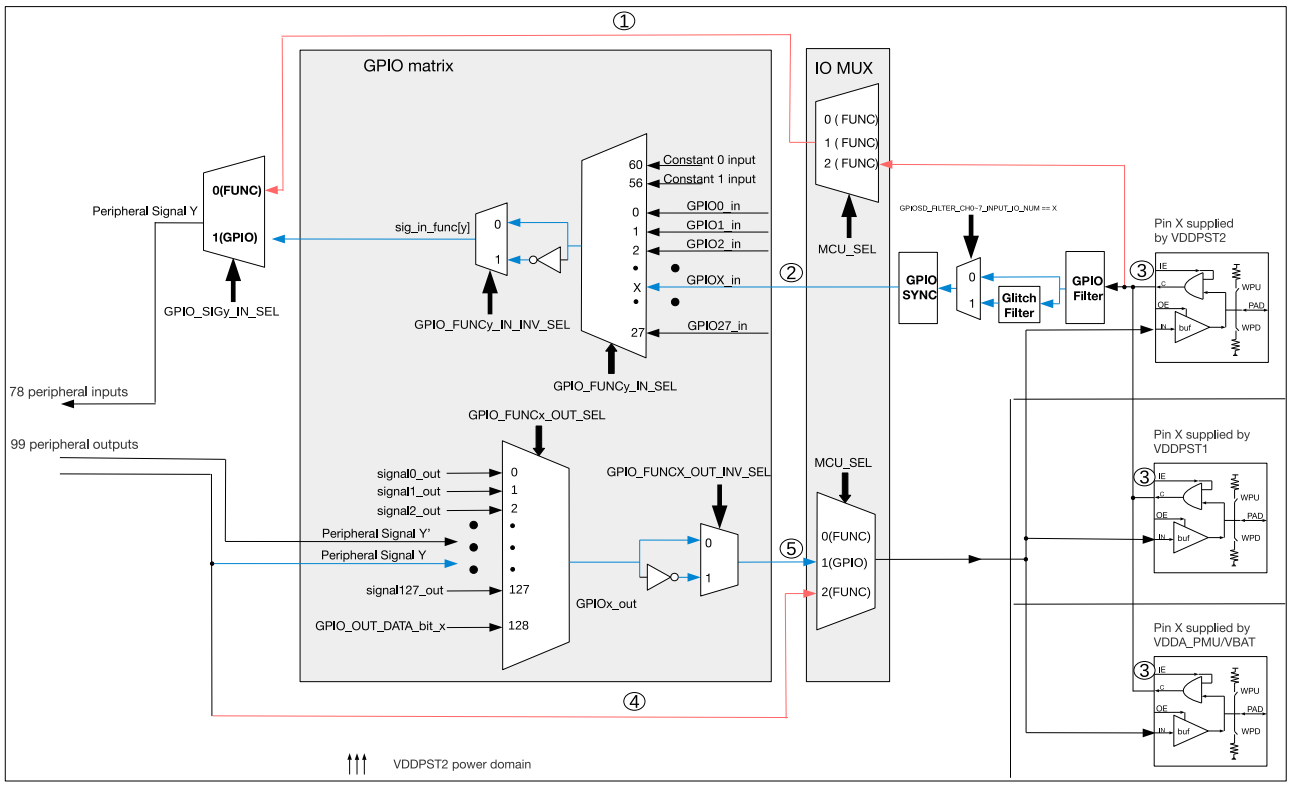


Figure 6.3-1. Architecture of IO MUX and GPIO Matrix

1. Only part of peripheral input signals (marked "yes" in column "Direct input through IO MUX" in Table 6.12-1) can bypass GPIO matrix. The other input signals can only be routed to peripherals via GPIO matrix.
2. There are only 19 inputs from GPIO SYNC to GPIO matrix, since ESP32-H2 provides 19 GPIO pins in total.
3. GPIO pins are controlled by the signals: IE, OE, WPU, and WPD.
4. Only part of peripheral outputs (marked "yes" in column "Direct output through IO MUX" in Table 6.12-1) can be routed to pins bypassing GPIO matrix. The other output signals can only be routed to pins via GPIO matrix.
5. There are 19 outputs (corresponding to GPIO pin *X*: 0 ~ 5, 8 ~ 14, 22 ~ 27) from GPIO matrix to IO MUX.

Figure 6.3-2 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 19 GPIO pins and can be controlled using IE, OE, WPU, and WPD signals.

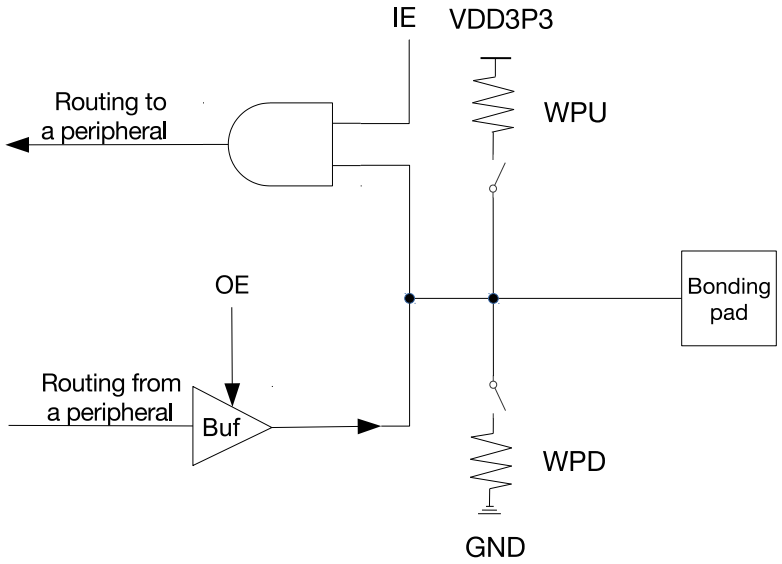


Figure 6.3-2. Internal Structure of a Pad

- IE: input enable
- OE: output enable
- WPU: internal weak pull-up resistor
- WPD: internal weak pull-down resistor
- Bonding pad: a terminal point of the chip logic used to make a physical connection from the chip die to GPIO pin in the chip package

6.4 Peripheral Input via GPIO Matrix

6.4.1 Overview

To receive a peripheral input signal via GPIO matrix, the matrix is configured to source the peripheral input signal from one of the 19 GPIOs (0 ~ 5, 8 ~ 14, 22 ~ 27), see Table 6.12-1. Meanwhile, the register corresponding to the peripheral signal should be set to receive input signal via GPIO matrix.

As shown in Figure 6.3-1, 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 6.4.3.

The Glitch Filter hardware can filter eight of the output signals from the GPIO Filter, and the other unselected signals go directly to the GPIO SYNC hardware, as shown in Step 3 in Section 6.4.3.

All signals filtered by the GPIO Filter hardware or the Glitch Filter hardware are synchronized by the GPIO SYNC hardware to IO MUX operating clock and then enter the GPIO matrix, see Section 6.4.2. Such signal filtering and synchronization features apply to all GPIO matrix signals but do not apply when using the IO MUX.

6.4.2 Signal Synchronization

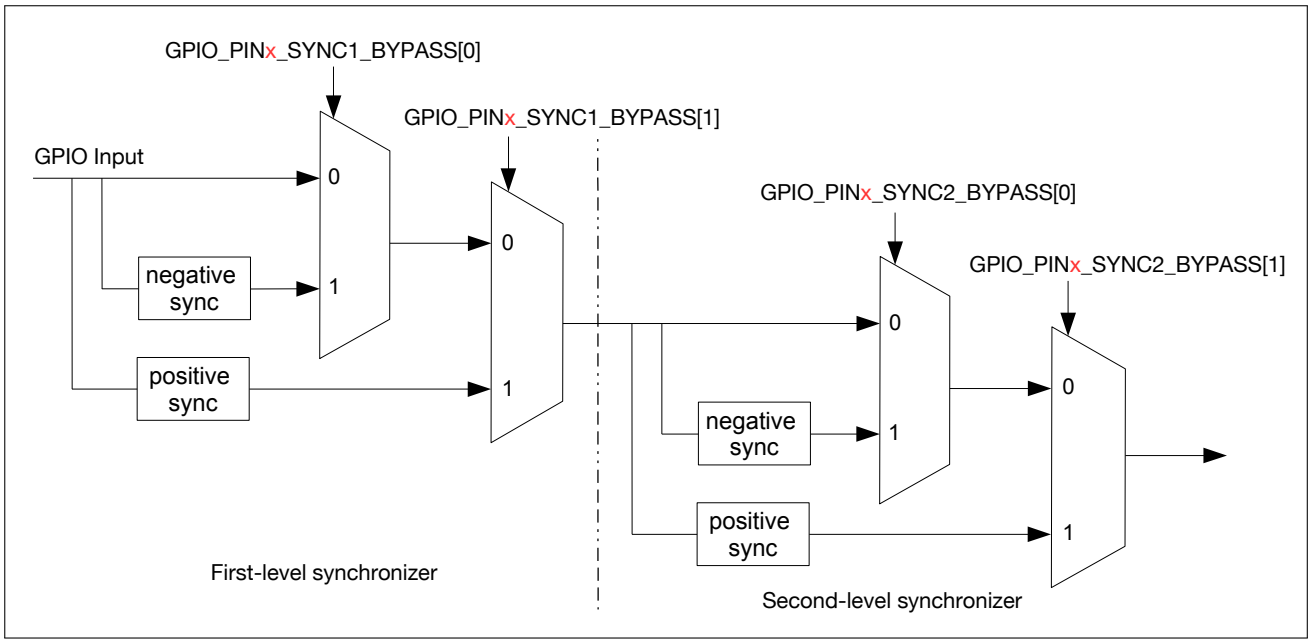


Figure 6.4-1. GPIO Input Synchronized on Rising Edge or on Falling Edge of IO MUX Operating Clock

Figure 6.4-1 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 IO MUX operating clock respectively.

The synchronization function is disabled by default by the synchronizer, i.e., GPIO_PIN*x*_SYNC1/2_BYPASS [1:0] = 0. 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 6.4-1) to lower the probability of causing metastability. For more information, see Step 4 in the following section.

6.4.3 Functional Description

To read GPIO pin *X*[1] into peripheral signal *Y*, follow the steps below:

1. Configure register GPIO_FUNC*y*_IN_SEL_CFG_REG corresponding to peripheral signal *Y* in GPIO matrix:
   - Set GPIO_SIG*y*_IN_SEL to enable peripheral signal input via GPIO matrix.
   - Set GPIO_FUNC*y*_IN_SEL to the desired GPIO pin, i.e., *X* here.

   Note that some peripheral signals have no valid GPIO_SIG*y*_IN_SEL bit, namely, these peripherals can only receive input signals via GPIO matrix.
2. Optionally enable the GPIO Filter for pin input signals by setting IO_MUX_GPIO*x*_FILTER_EN. Only the signals with a valid width of more than two clock cycles can be sampled, see Figure 6.4-2.

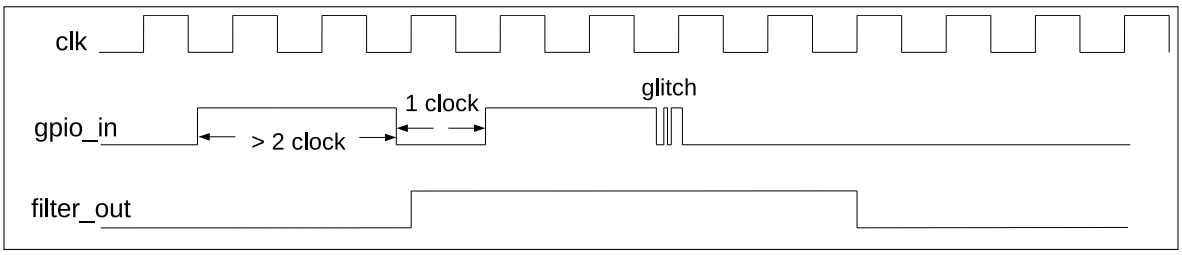


Figure 6.4-2. GPIO Filter Timing of GPIO Input Signals

3. Glitch filter hardware supports eight channels, each of which selects one signal from the 19 (0 ~ 5, 8 ~ 14, 22 ~ 27) output signals from the GPIO Filter hardware and conducts the second-time filtering on the selected signal. This Glitch Filter hardware can be used to filter slow-speed signals. To enable this feature, follow the steps below:

   - Configure GPIO_EXT_FILTER_CH*n*_INPUT_IO_NUM to *m*. *n* (0 ~ 7) represents the channel number. *m* (0 ~ 5, 8 ~ 14, 22 ~ 27) represents the GPIO pin number.
   - Configure GPIO_EXT_FILTER_CH*n*_WINDOW_WIDTH to **VALUE1** and GPIO_EXT_FILTER_CH*n*_WINDOW_THRES to **VALUE2**. During **VALUE1** + 1 cycles, if there are **VALUE2** + 1 input signals that do not match the current output signal value, the Glitch Filter hardware inverts the output signal. GPIO_EXT_FILTER_CH*n*_WINDOW_WIDTH and GPIO_EXT_FILTER_CH*n*_WINDOW_THRES can be configured to the same value **VALUE3**, then only signals with a width greater than **VALUE3** + 1 clock cycles will be sampled.
   - Set GPSD_FILTER_CH*n*_EN to enable channel *n*.

   An example is shown in Figure 6.4-3, where GPIO_EXT_FILTER_CH*x*_WINDOW_WIDTH is configured to 3 and GPIO_EXT_FILTER_CH*x*_WINDOW_THRES to 2. The output signal value (signal_out) keeps as "0" in the four clock cycles before T1. The input signal value (signal_in) has been "1" for three clock cycles in the same period, then the output signal is inverted to "1" after T1.

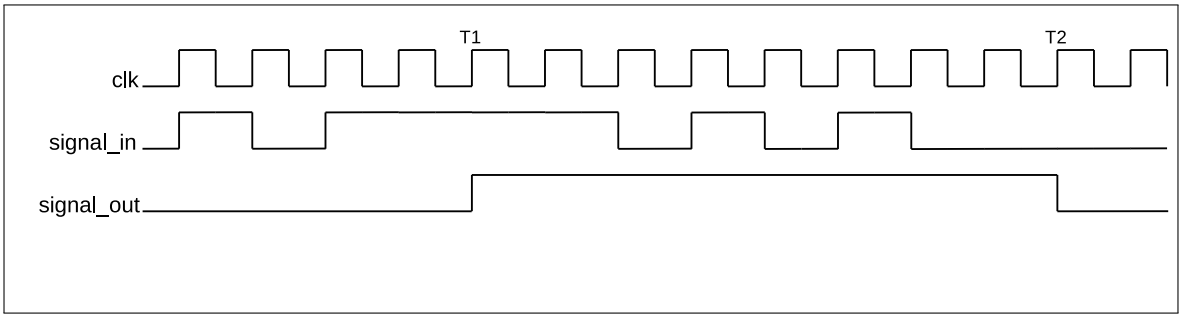


Figure 6.4-3. Glitch Filter Timing Example

4. Synchronize GPIO input signals. To do so, please set GPIO_PIN*x*_REG corresponding to GPIO pin *X* as follows:

   - Set GPIO_PIN*x*_SYNC1_BYPASS to enable input signal synchronized on rising edge or on falling edge in the first-level synchronization, see Figure 6.4-1.
   - 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 6.4-1.

5. Configure IO MUX register to enable pin input. For this end, please set IO_MUX_GPIO*x*_REG corresponding to GPIO pin *X* as follows:
   - Set IO_MUX_GPIO*x*_FUN_IE to enable input[2].
   - Set or clear IO_MUX_GPIO*x*_FUN_WPU and IO_MUX_GPIO*x*_FUN_WPD as desired to enable or disable pull-up and pull-down resistors.

For example, to connect I2S MSCK input signal [3] (I2S_MCLK_in, signal index 12) to GPIO3, please follow the steps below. Note that GPIO3 is also named as MTDO pin.

1. Set GPIO_SIG12_IN_SEL in register GPIO_FUNC12_IN_SEL_CFG_REG to enable peripheral signal input via GPIO matrix.
2. Set GPIO_FUNC12_IN_SEL in register GPIO_FUNC12_IN_SEL_CFG_REG to 3, i.e., select GPIO3.
3. Set IO_MUX_GPIO3_FUN_IE in register IO_MUX_GPIO3_REG to enable pin input.

Note:

1. One input pin can be connected to multiple peripheral input signals.
2. The input signal can be inverted by configuring GPIO_FUNC*y*_IN_INV_SEL.
3. It is possible to have a 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 0x3C, input signal is always 0.
   - When GPIO_FUNC*y*_IN_SEL is set to 0x38, input signal is always 1.

6.4.4 Simple GPIO Input

GPIO matrix can also be used for simple GPIO input. For this case, the input value of one GPIO pin can be read at any time without routing the GPIO input to any peripherals. GPIO_IN_REG holds the input values of each GPIO pin.

To implement simple GPIO input, follow the steps below:

- Set IO_MUX_GPIO*x*_FUN_IE in register IO_MUX_GPIO*x*_REG, to enable pin input.
- Read the GPIO input from GPIO_IN_REG[*x*].

6.5 Peripheral Output via GPIO Matrix

6.5.1 Overview

To output a signal from a peripheral via GPIO matrix, the matrix is configured to route peripheral output signals (only signals with a name assigned in the column "Output signal" in Table 6.12-1) to one of the 19 GPIOs (0 ~ 5, 8 ~ 14, 22 ~ 27).

The output signal is routed from the peripheral into GPIO matrix and then into IO MUX. IO MUX must be configured to set the chosen pin to GPIO function. This enables the GPIO output signal to be connected to the pin.

Note:
There is a range of peripheral output signals (97 ~ 100 in Table 6.12-1) which are not connected to any peripheral, but to the input signals (97 ~ 100) directly.

6.5.2 Functional Description

The 99 output signals (signals with a name assigned in the column "Output signal" in Table 6.12-1) can be set to go through GPIO matrix into IO MUX and then to a pin. Figure 6.3-1 illustrates the configuration.

To output peripheral signal *Y* to a particular GPIO pin *X*[1], follow the steps below:

1. Configure registers GPIO_FUNC*x*_OUT_SEL_CFG_REG and GPIO_ENABLE_REG[*x*] corresponding to GPIO pin *X* in GPIO matrix. Recommended operation: use corresponding W1TS (write 1 to set) and W1TC (write 1 to clear) registers to set or clear GPIO_ENABLE_REG.
   - Set the GPIO_FUNC*x*_OUT_SEL field in register GPIO_FUNC*x*_OUT_SEL_CFG_REG to the index of the desired peripheral output signal *Y*.
   - If the signal should always be enabled as an output, set the GPIO_FUNC*x*_OEN_SEL bit in register GPIO_FUNC*x*_OUT_SEL_CFG_REG and the bit in register GPIO_ENABLE_W1TS_REG, corresponding to GPIO pin *X*. To have the output enable signal decided by internal logic (for example, the SPIQ_oe in column "Output enable signal when GPIO_FUNC*n*_OEN_SEL = 0" in Table 6.12-1), clear the GPIO_FUNC*x*_OEN_SEL bit instead.
   - Set the corresponding bit in register GPIO_ENABLE_W1TC_REG to disable the output from the GPIO pin.
2. For an open drain output, set the GPIO_PIN*x*_PAD_DRIVER bit in register GPIO_PIN*x*_REG corresponding to GPIO pin *X*.
3. Configure IO MUX register to enable output via 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 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.
     - 0: ~5 mA
     - 1: ~10 mA
     - 2: ~20 mA (default)
     - 3: ~40 mA
   - If using open drain mode, set/clear the IO_MUX_GPIO*x*_FUN_WPU and IO_MUX_GPIO*x*_FUN_WPD bits to enable/disable the internal pull-up/pull-down resistors.

Note:

1. The output signal from a single peripheral can be sent to multiple pins simultaneously.
2. The output signal can be inverted by setting GPIO_FUNC*x*_OUT_INV_SEL.

6.5.3 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. This can be done as below:

- Set GPIO matrix GPIO_FUNC*n*_OUT_SEL with a special peripheral index 128 (0x80);
- Set the corresponding bit in GPIO_OUT_REG register to the desired GPIO output value.

Note:

- GPIO_OUT_REG[0] ~ GPIO_OUT_REG[27] correspond to GPIO0 ~ GPIO27 respectively. GPIO_OUT_REG[28] ~ GPIO_OUT_REG[31] are invalid.
- Recommended operation: use GPIO_OUT_W1TS/GPIO_OUT_W1TC to set or clear the register GPIO_OUT_REG.

6.5.4 Sigma Delta Modulated Output (SDM)

6.5.4.1 Functional Description

Four out of the 99 peripheral output signals (index: 83 ~ 86 in Table 6.12-1 support 1-bit second-order sigma delta modulation. By default the output is enabled for these four channels. This Sigma Delta modulator can also output PDM (pulse density modulation) signal with configurable duty cycle. The transfer function is:

$$H(z) = \mathrm{X(z)z^{-1} + E(z)(1\text{-}z^{-1})^2}$$

E(z) is quantization error and X(z) is the input.
This modulator supports scaling down of IO MUX operating clock by divider 1 ~ 256:

- Set GPIO_EXT_FUNCTION_CLK_EN to enable the modulator clock.
- Configure GPIO_EXT_SD*n*_PRESCALE (*n* = 0 ~ 3 for the four channels).

After scaling, the clock cycle is equal to one pulse output cycle from the modulator.

GPIO_EXT_SD*n*_IN is a signed number with a range of [-128, 127] and is used to control the duty cycle [1] of PDM output signal.

- GPIO_EXT_SD*n*_IN = -128, the duty cycle of the output signal is 0%.
- GPIO_EXT_SD*n*_IN = 0, the duty cycle of the output signal is near 50%.
- GPIO_EXT_SD*n*_IN = 127, the duty cycle of the output signal is near 100%.

The formula for calculating PDM signal duty cycle is shown as below:

$$Duty\_Cycle = \frac{GPIO\_EXT\_SDn\_IN + 128}{256}$$

Note:
For PDM signals, duty cycle refers to the percentage of high level cycles to the whole statistical period (several pulse cycles, for example, 256 pulse cycles).

6.5.4.2 SDM Configuration

The configuration of SDM is shown below:

- Route one of SDM outputs to a pin via GPIO matrix, see Section 6.5.2.
- Enable the modulator clock by setting GPIO_EXT_FUNCTION_CLK_EN.
- Configure the divider value by setting GPIO_EXT_SD*n*_PRESCALE.
- Configure the duty cycle of SDM output signal by setting GPIO_EXT_SD*n*_IN.

6.6 Direct Input and Output via IO MUX

6.6.1 Overview

Some digital signals (SPI and JTAG) can bypass GPIO matrix for better high-frequency digital performance. In this case, IO MUX is used to connect these pins directly to peripherals.

This option is less flexible than routing signals via GPIO matrix, as the IO MUX register for each GPIO pin can only select from a limited number of functions, but high-frequency digital performance can be improved.

6.6.2 Functional Description

Two fields must be configured in order to bypass GPIO matrix for 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 Section 6.13.
2. Clear GPIO_SIG*n*_IN_SEL to route the input directly to the peripheral.

To bypass GPIO matrix for 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 IO MUX. Some input/output signals can only be connected to the peripheral via GPIO matrix.

6.7 Analog Functions of GPIO Pins

Some GPIO pins in ESP32-H2 provide analog functions. When the pin is used for analog purposes, make sure that pull-up and pull-down resistors are disabled by the following configuration:

- Set IO_MUX_GPIO*n*_MCU_SEL to 1, and clear IO_MUX_GPIO*n*_FUN_IE, IO_MUX_GPIO*n*_FUN_WPU, IO_MUX_GPIO*n*_FUN_WPD.
- Write 1 to GPIO_ENABLE_W1TC*[n]*, to clear output enable.

See Table 6.14-1 for analog functions of ESP32-H2 pins.

6.8 Pin Functions in Light-sleep

Pins may provide different functions when ESP32-H2 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 6.8-1. Bit Used to Control IO MUX Functions in Light-sleep Mode

IO MUX Function	Normal Execution OR IO_MUX_GPIO*n*_SLP_SEL = 0	Light-sleep Mode AND IO_MUX_GPIO*n*_SLP_SEL = 1
Output Drive Strength	IO_MUX_GPIO*n*_FUN_DRV	IO_MUX_GPIO*n*_MCU_DRV
Pull-up Resistor	IO_MUX_GPIO*n*_FUN_WPU	IO_MUX_GPIO*n*_MCU_WPU
Pull-down Resistor	IO_MUX_GPIO*n*_FUN_WPD	IO_MUX_GPIO*n*_MCU_WPD
Input Enable	IO_MUX_GPIO*n*_FUN_IE	IO_MUX_GPIO*n*_MCU_IE
Output Enable	OEN_SEL from GPIO matrix *	IO_MUX_GPIO*n*_MCU_OE

Note:
If IO_MUX_GPIO*n*_SLP_SEL is set to 0, pin functions remain the same in both normal execution and in Light-sleep mode. Please refer to Section 6.5.2 for how to enable output in normal execution.

6.9 Pin Hold Feature

Each GPIO pin (including the LP pins: GPIO8 ~ GPIO14) has an individual hold function controlled by an LP register. 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.

To use this feature, follow the steps below:

- Digital pins (GPIO0 ~ GPIO5, GPIO22 ~ GPIO27):
  - To maintain pin input/output status in Deep-sleep mode, users can set LP_AON_GPIO_HOLD0_REG[*n*] to 1 before powering down. To disable the hold function after the chip is woken up, users can set LP_AON_GPIO_HOLD0_REG[*n*] to 0.
  - Or users can set PMU_TIE_HIGH_HP_PAD_HOLD_ALL to maintain the input/output status of all digital pins, and set PMU_TIE_LOW_HP_PAD_HOLD_ALL to disable the hold function of all digital pins.
- LP pins (GPIO8 ~ GPIO14):
  - The input and output values of LP GPIO pins are controlled by LP_AON_GPIO_HOLD0_REG[*n*], PMU_TIE_HIGH_LP_PAD_HOLD_ALL, and PMU_TIE_LOW_LP_PAD_HOLD_ALL. Users can set LP_AON_GPIO_HOLD0_REG[*n*] to 1 to hold the value of GPIO*n*, or set LP_AON_GPIO_HOLD0_REG[*n*] to 0 to disable the hold function of GPIO*n*.

- Or users can set PMU_TIE_HIGH_LP_PAD_HOLD_ALL to hold the values of all LP pins, and set PMU_TIE_LOW_LP_PAD_HOLD_ALL to disable the hold function of all LP pins.
- When the LP pin is held in the input state, it can serve as a wake-up source to wake up the chip from Deep-sleep mode.

6.10 Hysteresis Characteristics of GPIO Pins

Each GPIO pin has hysteresis functionality. When hysteresis is not enabled, as shown in Figure 6.10-1, 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, if the signal on the PAD has noise, it may affect the signal on C.

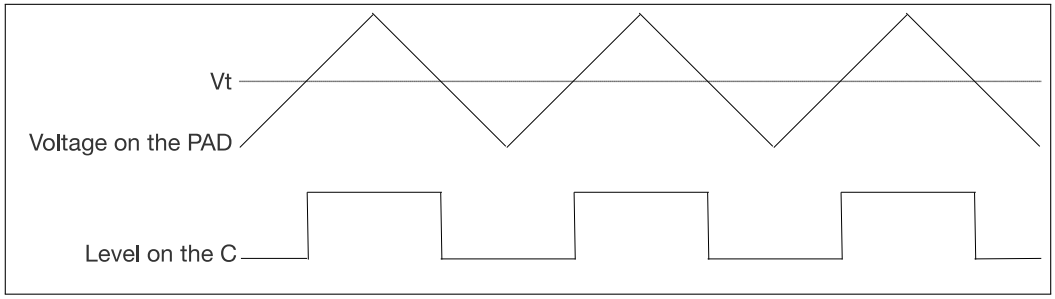


Figure 6.10-1. Example of level flip on the chip pad when the hysteresis function is not enabled

When hysteresis is enabled, as shown in Figure 6.10-2, 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 can reduce the impact of noise, play an anti-interference role, and reduce the level flip times of C.

To enable the hysteresis function, follow the steps below:

- When IO_MUX_GPIO*n*_HYS_SEL is 0 (*n* ranges from 0 ~ 5, 8 ~ 14 and 22 ~ 27, corresponding to GPIO0 ~ GPIO5, GPIO8 ~ GPIO14 and GPIO22 ~ GPIO27):
  - The hysteresis function for GPIO0 ~ GPIO5 is enabled when the corresponding bit in EFUSE_HYS_EN_PAD0[0:5] is 1. The hysteresis function for GPIO0 ~ GPIO5 is disabled when the corresponding bit in EFUSE_HYS_EN_PAD0[0:5] is 0.
  - The hysteresis function for GPIO8 ~ GPIO14 and GPIO22 ~ GPIO27 is enabled when the corresponding bit in EFUSE_HYS_EN_PAD1[2:8] and EFUSE_HYS_EN_PAD1[16:21] is 1. The hysteresis function for GPIO8 ~ GPIO14 and GPIO22 ~ GPIO27 is disabled when the corresponding bit in EFUSE_HYS_EN_PAD1[2:8] and EFUSE_HYS_EN_PAD1[16:21] is 0.
- When IO_MUX_GPIO*n*_HYS_SEL is 1 (*n* ranges from 0 ~ 5, 8 ~ 14 and 22 ~ 27,corresponding to GPIO0 ~ GPIO5, GPIO8 ~ GPIO14 and GPIO22 ~ GPIO27):
  - Set IO_MUX_GPIO*n*_HYS_EN to 1 to enable the hysteresis function for GPIO*n*.
  - Set IO_MUX_GPIO*n*_HYS_EN to 0 to disable the hysteresis function for GPIO*n*.

It is recommended to set IO_MUX_GPIO*n*_HYS_SEL to 1, and use IO_MUX_GPIO*n*_HYS_EN to enable or disable the hysteresis function of GPIO*n*.

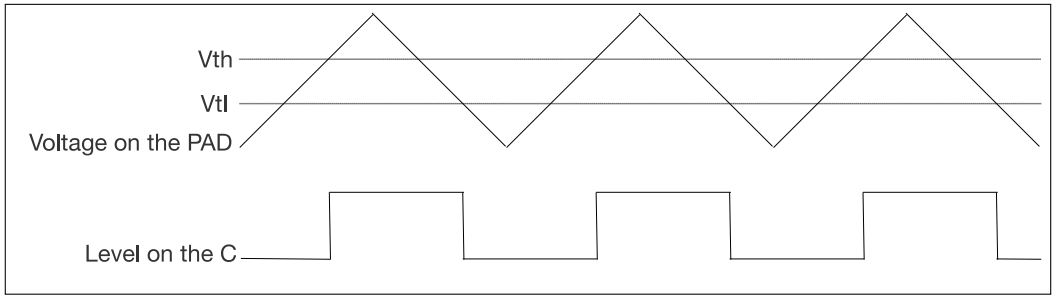


Figure 6.10-2. Example of level flip on the chip pad when the hysteresis function is enabled

6.11 Power Supplies and Management of GPIO Pins

6.11.1 Power Supplies of GPIO Pins

For more information on the power supply for GPIO pins, please refer to Pin Definition in *ESP32-H2 Datasheet*. All the pins can be used to wake up the chip from Light-sleep mode, but only the LP pins (GPIO8 ~ GPIO14) can be used to wake up the chip from Deep-sleep mode.

6.11.2 Power Supply Management

Each ESP32-H2 pin is connected to one of the three different power domains.

- VDDPST1: the input power supply for some digital GPIOs and some LP GPIOs
- VDDPST2: the input power supply for some digital GPIOs
- VDDA_PMU/VBAT: the input power supply for GPIO12, XTAL_32K_P and XTAL_32K_N

6.12 Peripheral Signal List

Table 6.12-1 shows the peripheral input/output signals via GPIO matrix.

Please pay attention to the configuration of the bit GPIO_FUNC*n*_OEN_SEL:

- GPIO_FUNC*n*_OEN_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*_OEN_SEL = 0: use the output enable signal from peripheral, for example SPIQ_oe in the column "Output enable signal when GPIO_FUNC*n*_OEN_SEL = 0" of Table 6.12-1. Note that the signals such as SPIQ_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*_OEN_SEL = 0", it indicates that once GPIO_FUNC*n*_OEN_SEL is cleared, the output signal is always enabled by default.

Note:

Signals are numbered consecutively, but not all signals are valid.

- Only the signals with a name assigned in the column "Input signal" in Table 6.12-1 are valid input signals.
- Only the signals with a name assigned in the column "Output signal" in Table 6.12-1 are valid output signals.

Table 6.12-1. Peripheral Signals via GPIO Matrix

Signal No.	Input Signal	Default value	Direct Input via IO MUX	Output Signal	Output enable signal when GPIO_FUNC*n*_OEN_SEL = 0	Direct Output via IO MUX
0	ext_adc_start	0	no	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
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	usb_jtag_tdo_bridge	0	no	usb_jtag_trst	1'd1	no
20	-	-	-	-	-	-
21	-	-	-	-	-	-
22	-	-	-	-	-	-
23	-	-	-	-	-	-
24	-	-	-	-	-	-
25	-	-	-	-	-	-

Signal No.	Input Signal	Default value	Direct Input via IO MUX	Output Signal	Output enable signal when GPIO_FUNCn_OEN_SEL = 0	Direct Output via IO MUX
26	-	-	-	-	-	-
27	-	-	-	-	-	-
28	cpu_gpio_in0	0	no	cpu_gpio_out0	cpu_gpio_out_oen0	no
29	cpu_gpio_in1	0	no	cpu_gpio_out1	cpu_gpio_out_oen1	no
30	cpu_gpio_in2	0	no	cpu_gpio_out2	cpu_gpio_out_oen2	no
31	cpu_gpio_in3	0	no	cpu_gpio_out3	cpu_gpio_out_oen3	no
32	cpu_gpio_in4	0	no	cpu_gpio_out4	cpu_gpio_out_oen4	no
33	cpu_gpio_in5	0	no	cpu_gpio_out5	cpu_gpio_out_oen5	no
34	cpu_gpio_in6	0	no	cpu_gpio_out6	cpu_gpio_out_oen6	no
35	cpu_gpio_in7	0	no	cpu_gpio_out7	cpu_gpio_out_oen7	no
36	-	-	-	-	-	-
37	-	-	-	-	-	-
38	-	-	-	-	-	-
39	-	-	-	-	-	-
40	-	-	-	-	-	-
41	-	-	-	-	-	-
42	-	-	-	-	-	-
43	-	-	-	-	-	-
44	-	-	-	-	-	-
45	I2CEXT0_SCL_in	1	no	I2CEXT0_SCL_out	I2CEXT0_SCL_oe	no
46	I2CEXT0_SDA_in	1	no	I2CEXT0_SDA_out	I2CEXT0_SDA_oe	no
47	parl_rx_data0	0	no	parl_tx_data0	1'd1	no
48	parl_rx_data1	0	no	parl_tx_data1	1'd1	no
49	parl_rx_data2	0	no	parl_tx_data2	1'd1	no
50	parl_rx_data3	0	no	parl_tx_data3	1'd1	no
51	parl_rx_data4	0	no	parl_tx_data4	1'd1	no
52	parl_rx_data5	0	no	parl_tx_data5	1'd1	no
53	parl_rx_data6	0	no	parl_tx_data6	1'd1	no

Signal No.	Input Signal	Default value	Direct Input via IO MUX	Output Signal	Output enable signal when GPIO_FUNCn_OEN_SEL = 0	Direct Output via IO MUX
54	parl_rx_data7	0	no	parl_tx_data7	1'd1	no
55	I2CEXT1_SCL_in	1	no	I2CEXT1_SCL_out	I2CEXT1_SCL_oe	no
56	I2CEXT1_SDA_in	1	no	I2CEXT1_SDA_out	I2CEXT1_SDA_oe	no
57		-	-	cte_ant0	1'd1	no
58		-	-	cte_ant1	1'd1	no
59		-	-	cte_ant2	1'd1	no
60		-	-	cte_ant3	1'd1	no
61		-	-	cte_ant4	1'd1	no
62		-	-	cte_ant5	1'd1	no
63	FSPICLK_in	0	yes	FSPICLK_out_mux	FSPICLK_oe	yes
64	FSPIQ_in	0	yes	FSPIQ_out	FSPIQ_oe	yes
65	FSPID_in	0	yes	FSPID_out	FSPID_oe	yes
66	FSPIHD_in	0	yes	FSPIHD_out	FSPIHD_oe	yes
67	FSPIWP_in	0	yes	FSPIWP_out	FSPIWP_oe	yes
68	FSPICS0_in	0	yes	FSPICS0_out	FSPICS0_oe	yes
69	parl_rx_clk_in	0	no	parl_rx_clk_out	1'd1	no
70	parl_tx_clk_in	0	no	parl_tx_clk_out	1'd1	no
71	rmt_sig_in0	0	no	rmt_sig_out0	1'd1	no
72	rmt_sig_in1	0	no	rmt_sig_out1	1'd1	no
73	twai0_rx	1	no	twai0_tx	1'd1	no
74	-	-	-	twai0_bus_off_on	1'd1	no
75	-	-	-	twai0_clkout	1'd1	no
76	-	-	-	twai0_standby	1'd1	no
77		-	-	cte_ant6	1'd1	no
78	-	-	-	cte_ant7	1'd1	no
79	-	-	-	cte_ant8	1'd1	no
80	-	-	-	cte_ant9	1'd1	no
81	-	-	-	-	-	-

Signal No.	Input Signal	Default value	Direct Input via IO MUX	Output Signal	Output enable signal when GPIO_FUNCn_OEN_SEL = 0	Direct Output via IO MUX
82	-	-	-	-	-	-
83	-	-	-	gpio_sd0_out	1'd1	no
84	-	-	-	gpio_sd1_out	1'd1	no
85	-	-	-	gpio_sd2_out	1'd1	no
86	-	-	-	gpio_sd3_out	1'd1	no
87	pwm0_sync0_in	0	no	pwm0_out0a	1'd1	no
88	pwm0_sync1_in	0	no	pwm0_out0b	1'd1	no
89	pwm0_sync2_in	0	no	pwm0_out1a	1'd1	no
90	pwm0_f0_in	0	no	pwm0_out1b	1'd1	no
91	pwm0_f1_in	0	no	pwm0_out2a	1'd1	no
92	pwm0_f2_in	0	no	pwm0_out2b	1'd1	no
93	pwm0_cap0_in	0	no	-	-	-
94	pwm0_cap1_in	0	no	-	-	-
95	pwm0_cap2_in	0	no	-	-	-
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	pcnt_sig_ch0_in0	0	no	FSPICS1_out	FSPICS1_oe	yes
102	pcnt_sig_ch1_in0	0	no	FSPICS2_out	FSPICS2_oe	yes
103	pcnt_ctrl_ch0_in0	0	no	FSPICS3_out	FSPICS3_oe	yes
104	pcnt_ctrl_ch1_in0	0	no	FSPICS4_out	FSPICS4_oe	yes
105	pcnt_sig_ch0_in1	0	no	FSPICS5_out	FSPICS5_oe	yes
106	pcnt_sig_ch1_in1	0	no	cte_ant10	1'd1	no
107	pcnt_ctrl_ch0_in1	0	no	cte_ant11	1'd1	no
108	pcnt_ctrl_ch1_in1	0	no	cte_ant12	1'd1	no
109	pcnt_sig_ch0_in2	0	no	cte_ant13	1'd1	no

Signal No.	Input Signal	Default value	Direct Input via IO MUX	Output Signal	Output enable signal when GPIO_FUNCn_OEN_SEL = 0	Direct Output via IO MUX
110	pcnt_sig_ch1_in2	0	no	cte_ant14	1'd1	no
111	pcnt_ctrl_ch0_in2	0	no	cte_ant15	1'd1	no
112	pcnt_ctrl_ch1_in2	0	no	-	-	-
113	pcnt_sig_ch0_in3	0	no	-	-	-
114	pcnt_sig_ch1_in3	0	no	SPICLK_out_mux	SPICLK_oe	yes
115	pcnt_ctrl_ch0_in3	0	no	SPICS0_out	SPICS0_oe	yes
116	pcnt_ctrl_ch1_in3	0	no	SPICS1_out	SPICS1_oe	no
117	-	-	-	-	-	-
118	-	-	-	-	-	-
119	-	-	-	-	-	-
120	-	-	-	-	-	-
121	SPIQ_in	0	yes	SPIQ_out	SPIQ_oe	yes
122	SPID_in	0	yes	SPID_out	SPID_oe	yes
123	SPIHD_in	0	yes	SPIHD_out	SPIHD_oe	yes
124	SPIWP_in	0	yes	SPIWP_out	SPIWP_oe	yes
125	-	-	-	CLK_OUT_out1	1'd1	no
126	-	-	-	CLK_OUT_out2	1'd1	no
127	-	-	-	CLK_OUT_out3	1'd1	no

6.13 IO MUX Functions List

Table 6.13-1 shows the IO MUX functions of each GPIO pin.

Table 6.13-1. IO MUX Functions List

GPIO	Pin Name	Function 0	Function 1	Function 2	Function 3	DRV	Reset	Notes
0	GPIO0	GPIO0	GPIO0	FSPIQ	—	2	0	—
1	GPIO1	GPIO1	GPIO1	FSPICS0	—	2	0	—
2	MTMS	MTMS	GPIO2	FSPIWP	—	2	1	—
3	MTDO	MTDO	GPIO3	FSPIHD	—	2	1	—
4	MTCK	MTCK	GPIO4	FSPICLK	—	2	1*	—
5	MTDI	MTDI	GPIO5	FSPID	—	2	1	—
8	GPIO8	GPIO8	GPIO8	—	—	2	1	R
9	GPIO9	GPIO9	GPIO9	—	—	2	3	R
10	GPIO10	GPIO10	GPIO10	—	—	2	0	R
11	GPIO11	GPIO11	GPIO11	—	—	2	0	R
12	GPIO12	GPIO12	GPIO12	—	—	2	0	R
13	XTAL_32K_P	GPIO13	GPIO13	—	—	2	0	R
14	XTAL_32K_N	GPIO14	GPIO14	—	—	2	0	R
22	GPIO22	GPIO22	GPIO22	—	—	2	0	—
23	U0RXD	U0RXD	GPIO23	FSPICS1	—	2	3	—
24	U0TXD	U0TXD	GPIO24	FSPICS2	—	2	4	—
25	GPIO25	GPIO25	GPIO25	FSPICS3	—	2	1	—
26	GPIO26	GPIO26	GPIO26	FSPICS4	—	3	1	USB
27	GPIO27	GPIO27	GPIO27	FSPICS5	—	3	3*	USB

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 Configurations

"Reset" column shows the default configuration of each pin after reset:

- **0** - IE = 0 (input disabled)
- **1** - IE = 1 (input enabled)
- **2** - IE = 1, WPD = 1 (input enabled, pull-down resistor enabled)
- **3** - IE = 1, WPU = 1 (input enabled, pull-up resistor enabled)
- **4** - OE = 1, WPU = 1 (output enabled, pull-up resistor 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 GPIO27's USB pull-up is 1, which means the pull-up resistor is enabled. For details, please refer to the note below.

GPIO Input Mode

The input function of GPIO can be configured as hysteresis or normal mode:

- Hysteresis mode: In the hysteresis mode, the threshold voltage for flipping between high and low levels of GPIO input depends on the direction of level flipping. Specifically, the voltage threshold for flipping from high to low level is slightly lower than the voltage threshold for flipping from low to high level. For details, see Chapter 6.10.
- Normal mode: Disable hysteresis for GPIO pins, and the threshold voltage for flipping between high and low levels of GPIO input is independent of the direction of level flipping. In other words, the voltage threshold for flipping from high to low level is the same as the voltage threshold for flipping from low to high level.

Note:

- **R** - LP pins. Some LP pins have analog functions. For details, see 6.14-1.
- **USB** - USB pull-up resistor enabled
  - By default, the USB function is enabled for USB pins (i.e., GPIO26 and GPIO27), 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 *ESP32-H2 Technical Reference Manual* > 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).

6.14 IO MUX Pins Analog Functions List

Table 6.14-1 lists all the IO MUX pins that have analog functions.

Table 6.14-1. Analog Functions of IO MUX Pins

GPIO No.[1]	Pin Name	Analog Function 0	Analog Function 1
1	GPIO1	-	ADC1_CH0
2	MTMS	-	ADC1_CH1
3	MTDO	-	ADC1_CH2
4	MTCK	-	ADC1_CH3
5	MTDI	-	ADC1_CH4
10	GPIO10	ZCD0[1]	-
11	GPIO11	ZCD1[1]	-
13	XTAL_32K_P	XTAL_32K_P	-
14	XTAL_32K_N	XTAL_32K_N	-

26	GPIO26	USB_D-	-
27	GPIO27	USB_D+	-

[1] ZCD0 and ZCD1 are analog PAD voltage comparator functions. See subsection 6.15 for details.

6.15 Function of Analog PAD Voltage Comparator

GPIO10 and GPIO11 pads have the function of analog PAD voltage comparator, which can be enabled by setting GPIO_EXT_XPD_COMP to 1. After enabling the function of analog PAD voltage comparator, when the voltage on GPIO11 pad is higher than the reference voltage, the PAD_COMP_OUT signal indicating the comparison result will be high, otherwise it will be low.

Set the value of GPIO_EXT_MODE_COMP as follows:

- 0: the reference voltage is (GPIO_EXT_DREF_COMP*VDDPST2)/10.
- 1: the reference voltage is the voltage on GPIO10 PAD.

The PAD_COMP_OUT signal will be synchronized to the operating clock of the IO MUX to generate the PAD_COMP_OUT_sync signal, which will be used as the interrupt source GPIO_EXT_PAD_COMP_INT.

Set the value of GPIO_EXT_ZERO_DET_MODE as follows:

- 0: disable interrupt source generation.
- 1, 2: reserved
- 3: enable interrupt source and set any edge of PAD_COMP_OUT_sync signal as interrupt source.

Meanwhile, after generating an interrupt source, new interrupt sources will be masked within GPIO_EXT_ZERO_DET_FILTER_CNT IO MUX operating clock cycles.

6.16 Event Task Matrix Function

In ESP32-H2 , GPIO supports ETM function, that is, the ETM task of GPIO can be triggered by the ETM event of any peripheral, or the ETM task of any peripheral can be triggered by the ETM event of GPIO. For more details about ETM, please refer to Chapter 10 *Event Task Matrix (SOC_ETM)*. Only ETM tasks and ETM events related to GPIO are introduced here.

The GPIO ETM provides eight task channels *x* (0 ~ 7). The ETM tasks that each task channel can receive are:

- GPIO_TASK_CH*x*_SET: GPIO goes high when triggered;
- GPIO_TASK_CH*x*_CLEAR: GPIO goes low when triggered;
- GPIO_TASK_CH*x*_TOGGLE: GPIO toggle 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 6.13-1;
- 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 three of the signals GPIO_TASK_CH*x*_SET, GPIO_TASK_CH*x*_CLEAR, and GPIO_TASK_CH*x*_TOGGLE of the task channel *x* selected by GPIO*y* are valid at the same time, then GPIO_TASK_CH*x*_SET has the highest priority, GPIO_TASK_CH*x*_CLEAR takes the second higher priority, and GPIO_TASK_CH*x*_TOGGLE has the lowest priority.
- When GPIO*y* is controlled by ETM task channel, the values of GPIO_OUT_REG, GPIO_FUNC*n*_OUT_INV_SEL, and GPIO_FUNC*n*_OUT_SEL may be modified by the hardware. For such reason, it's recommended to reconfigure these registers when the GPIO is free from the control of ETM task channel.

GPIO has eight event channels, and the ETM events that each event channel can generate are:

- GPIO_EVT_CH*x*_RISE_EDGE: Indicates that the output signal of the corresponding GPIO filter (see Figure 6.3-1) has a rising edge;
- GPIO_EVT_CH*x*_FALL_EDGE: Indicates that the output signal of the corresponding GPIO filter (see Figure 6.3-1) has a falling edge;
- GPIO_EVT_CH*x*_ANY_EDGE: Indicates that the output signal of the corresponding GPIO filter (see Figure 6.3-1) 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 ~ 5, 8 ~ 14 and 22 ~ 27), i.e., select one from the 19 GPIOs.

Note:
One GPIO can be selected by one or more event channels.

In specific applications, GPIO ETM events can be used to trigger GPIO ETM tasks. For example, event channel 0 selects GPIO0, GPIO1 selects task channel 0, and the GPIO_EVT_CH*0*_RISE_EDGE event is used to trigger the GPIO_TASK_CH*0*_TOGGLE task. When a square wave signal is input to the chip through GPIO0, the chip outputs a square wave signal with a frequency divided by 2 through GPIO1.

6.17 Register Summary

6.17.1 GPIO Matrix Register Summary

The addresses in this section are relative to GPIO base address provided in Table 4.3-2 in Chapter 4 *System and Memory*.

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

For ESP32-H2, 19 GPIO pins are available, i.e., GPIO0 ~ GPIO5, GPIO8 ~ GPIO14, and GPIO22 ~ GPIO27. For this case:

- **Configuration Registers**: can only be configured for GPIO0 ~ GPIO5, GPIO8 ~ GPIO14 and GPIO22 ~ GPIO27.
- **Pin Configuration Registers**: only GPIO_PIN0_REG ~ GPIO_PIN5_REG, GPIO_PIN8_REG ~ GPIO_PIN14_REG and GPIO_PIN22_REG ~ GPIO_PIN27_REG are available.
- **Input Configuration Registers**: can only be configured for GPIO0 ~ GPIO5, GPIO8 ~ GPIO14 and GPIO22 ~ GPIO27.
- **Output Configuration Registers**: only GPIO_FUNC0_OUT_SEL_CFG_REG ~ GPIO_FUNC5_OUT_SEL_CFG_REG, GPIO_FUNC8_OUT_SEL_CFG_REG ~ GPIO_FUNC14_OUT_SEL_CFG_REG and GPIO_PIN22_OUT_SEL_CFG_REG ~ GPIO_PIN27_OUT_SEL_CFG_REGare available.

Name	Description	Address	Access
Configuration Registers			
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	0x0020	R/W/WTC
GPIO_ENABLE_W1TS_REG	GPIO output enable set register	0x0024	WT
GPIO_ENABLE_W1TC_REG	GPIO output enable clear register	0x0028	WT
GPIO_STRAP_REG	Strapping pin register	0x0038	RO
GPIO_IN_REG	GPIO input register	0x003C	RO
Interrupt Status Registers			
GPIO_STATUS_REG	GPIO interrupt status register	0x0044	R/W/WTC
GPIO_STATUS_W1TS_REG	GPIO interrupt status set register	0x0048	WT
GPIO_STATUS_W1TC_REG	GPIO interrupt status clear register	0x004C	WT
GPIO_PCPU_INT_REG	GPIO CPU interrupt status register	0x005C	RO
GPIO_STATUS_NEXT_REG	GPIO interrupt source register	0x014C	RO
Pin Configuration Registers			
GPIO_PIN0_REG	GPIO0 configuration register	0x0074	R/W
GPIO_PIN1_REG	GPIO1 configuration register	0x0078	R/W
GPIO_PIN2_REG	GPIO2 configuration register	0x007C	R/W
...	...	...	...
GPIO_PIN25_REG	GPIO25 pin configuration register	0x00D8	R/W
GPIO_PIN26_REG	GPIO26 pin configuration register	0x00DC	R/W

Name	Description	Address	Access
GPIO_PIN27_REG	GPIO27 pin configuration register	0x00E0	R/W
Input Configuration Registers			
GPIO_FUNC0_IN_SEL_CFG_REG	Configuration register for input signal 0	0x0154	R/W
GPIO_FUNC1_IN_SEL_CFG_REG	Configuration register for input signal 1	0x0158	R/W
GPIO_FUNC2_IN_SEL_CFG_REG	Configuration register for input signal 2	0x015C	R/W
...	...	...	...
GPIO_FUNC125_IN_SEL_CFG_REG	Configuration register for input signal 125	0x0348	R/W
GPIO_FUNC126_IN_SEL_CFG_REG	Configuration register for input signal 126	0x034C	R/W
GPIO_FUNC127_IN_SEL_CFG_REG	Configuration register for input signal 127	0x0350	R/W
Output Configuration Registers			
GPIO_FUNC0_OUT_SEL_CFG_REG	Configuration register for GPIO0 output	0x0554	varies
GPIO_FUNC1_OUT_SEL_CFG_REG	Configuration register for GPIO1 output	0x0558	varies
GPIO_FUNC2_OUT_SEL_CFG_REG	Configuration register for GPIO2 output	0x055C	varies
...	...	...	...
GPIO_FUNC25_OUT_SEL_CFG_REG	Configuration register for GPIO25 output	0x05B8	varies
GPIO_FUNC26_OUT_SEL_CFG_REG	Configuration register for GPIO26 output	0x05BC	varies
GPIO_FUNC27_OUT_SEL_CFG_REG	Configuration register for GPIO27 output	0x05C0	varies
Version Register			
GPIO_DATE_REG	GPIO version register	0x06FC	R/W
Clock Gate Register			
GPIO_CLOCK_GATE_REG	GPIO clock gate register	0x062C	R/W

6.17.2 IO MUX Register Summary

The addresses in this section are relative to the IO MUX base address provided in Table 4.3-2 in Chapter 4 *System and Memory*.

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

For ESP32-H2, 19 GPIO pins are available, i.e., GPIO0 ~ GPIO5, GPIO8 ~ GPIO14 and GPIO22 ~ GPIO27. For this case, **Configuration Registers** of IO_MUX_GPIO6_REG ~ IO_MUX_GPIO7_REG and IO_MUX_GPIO15_REG ~ IO_MUX_GPIO21_REG are not configurable.

Name	Description	Address	Access
Configuration Registers			
IO_MUX_PIN_CTRL_REG	Clock output configuration register	0x0000	R/W
IO_MUX_GPIO0_REG	IO MUX configuration register for GPIO0	0x0004	R/W
IO_MUX_GPIO1_REG	IO MUX configuration register for GPIO1	0x0008	R/W
IO_MUX_GPIO2_REG	IO MUX configuration register for GPIO2	0x000C	R/W
...	...	...	...
IO_MUX_GPIO25_REG	IO MUX configuration register for GPIO25	0x0068	R/W
IO_MUX_GPIO26_REG	IO MUX configuration register for GPIO26	0x006C	R/W
IO_MUX_GPIO27_REG	IO MUX configuration register for GPIO27	0x0070	R/W
Version Register			
IO_MUX_DATE_REG	Version control register	0x00FC	R/W

6.17.3 GPIO_EXT Register Summary

GPIO_EXT registers consist of SDM registers, Glitch Filter registers, Zero-Crossing Detection registers and ETM registers.

The addresses in this section are relative to (GPIO base address + 0x0F00). GPIO base address is provided in Table 4.3-2 in Chapter 4 *System and Memory*.

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

Name	Description	Address	Access
SDM Configure Registers			
GPIO_EXT_SIGMADELTA0_REG	Duty cycle configuration register for SDM channel 0	0x0000	R/W
GPIO_EXT_SIGMADELTA1_REG	Duty cycle configuration register for SDM channel 1	0x0004	R/W
GPIO_EXT_SIGMADELTA2_REG	Duty cycle configuration register for SDM channel 2	0x0008	R/W
GPIO_EXT_SIGMADELTA3_REG	Duty cycle configuration register for SDM channel 3	0x000C	R/W
GPIO_EXT_SIGMADELTA_MISC_REG	MISC register	0x0024	R/W
Glitch Filter Configuration Registers			
GPIO_EXT_GLITCH_FILTER_CH0_REG	Glitch Filter configuration register for channel 0	0x0030	R/W
GPIO_EXT_GLITCH_FILTER_CH1_REG	Glitch Filter configuration register for channel 1	0x0034	R/W
GPIO_EXT_GLITCH_FILTER_CH2_REG	Glitch Filter configuration register for channel 2	0x0038	R/W
GPIO_EXT_GLITCH_FILTER_CH3_REG	Glitch Filter configuration register for channel 3	0x003C	R/W
GPIO_EXT_GLITCH_FILTER_CH4_REG	Glitch Filter configuration register for channel 4	0x0040	R/W
GPIO_EXT_GLITCH_FILTER_CH5_REG	Glitch Filter configuration register for channel 5	0x0044	R/W
GPIO_EXT_GLITCH_FILTER_CH6_REG	Glitch Filter configuration register for channel 6	0x0048	R/W
GPIO_EXT_GLITCH_FILTER_CH7_REG	Glitch Filter configuration register for channel 7	0x004C	R/W
ETM Configuration Registers			
GPIO_EXT_ETM_EVENT_CH0_CFG_REG	ETM configuration register for channel 0	0x0060	R/W
GPIO_EXT_ETM_EVENT_CH1_CFG_REG	ETM configuration register for channel 1	0x0064	R/W
GPIO_EXT_ETM_EVENT_CH2_CFG_REG	ETM configuration register for channel 2	0x0068	R/W
GPIO_EXT_ETM_EVENT_CH3_CFG_REG	ETM configuration register for channel 3	0x006C	R/W
GPIO_EXT_ETM_EVENT_CH4_CFG_REG	ETM configuration register for channel 4	0x0070	R/W
GPIO_EXT_ETM_EVENT_CH5_CFG_REG	ETM configuration register for channel 5	0x0074	R/W
GPIO_EXT_ETM_EVENT_CH6_CFG_REG	ETM configuration register for channel 6	0x0078	R/W
GPIO_EXT_ETM_EVENT_CH7_CFG_REG	ETM configuration register for channel 7	0x007C	R/W
GPIO_EXT_ETM_TASK_P0_CFG_REG	GPIO selection register 0 for ETM	0x00A0	R/W
GPIO_EXT_ETM_TASK_P1_CFG_REG	GPIO selection register 1 for ETM	0x00A4	R/W
GPIO_EXT_ETM_TASK_P2_CFG_REG	GPIO selection register 2 for ETM	0x00A8	R/W
GPIO_EXT_ETM_TASK_P3_CFG_REG	GPIO selection register 3 for ETM	0x00AC	R/W
GPIO_EXT_ETM_TASK_P4_CFG_REG	GPIO selection register 4 for ETM	0x00B0	R/W
GPIO_EXT_ETM_TASK_P5_CFG_REG	GPIO selection register 5 for ETM	0x00B4	R/W
GPIO_EXT_ETM_TASK_P6_CFG_REG	GPIO selection register 6 for ETM	0x00B8	R/W

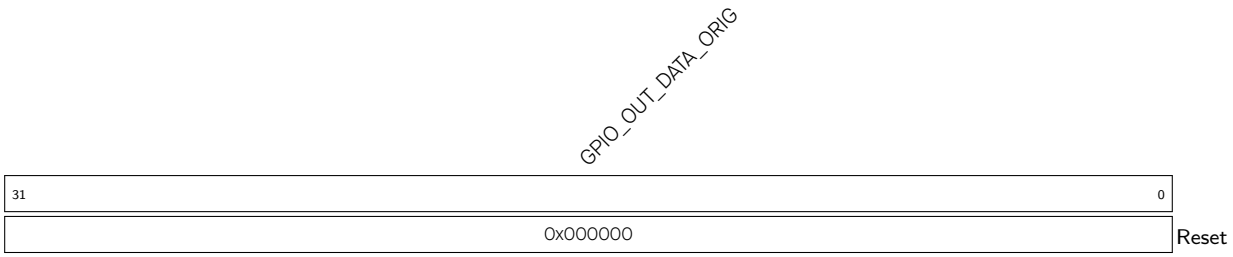
Name	Description	Address	Access
Zero-crossing Detection Configuration Registers			
GPIO_EXT_PAD_COMP_CONFIG_REG	Configuration register for zero-crossing detection	0x0028	R/W
GPIO_EXT_PAD_COMP_FILTER_REG	configuration register for interrupt source mask period of zero-crossing detection	0x002C	R/W
Interrupt Status Registers			
GPIO_EXT_INT_RAW_REG	Raw interrupt register	0x00E0	varies
GPIO_EXT_INT_ST_REG	Interrupt enable register	0x00E4	RO
GPIO_EXT_INT_ENA_REG	Interrupt status register	0x00E8	R/W
GPIO_EXT_INT_CLR_REG	Interrupt clear register	0x00EC	WT
Version Register			
GPIO_EXT_VERSION_REG	Version control register	0x00FC	R/W

6.18 Registers

6.18.1 GPIO Matrix Registers

The addresses in this section are relative to GPIO base address provided in Table 4.3-2 in Chapter 4 *System and Memory*.

Register 6.1. GPIO_OUT_REG (0x0004)



GPIO_OUT_DATA_ORIG Configures the output value of GPIO0 ~ GPIO27 output in simple GPIO output mode.

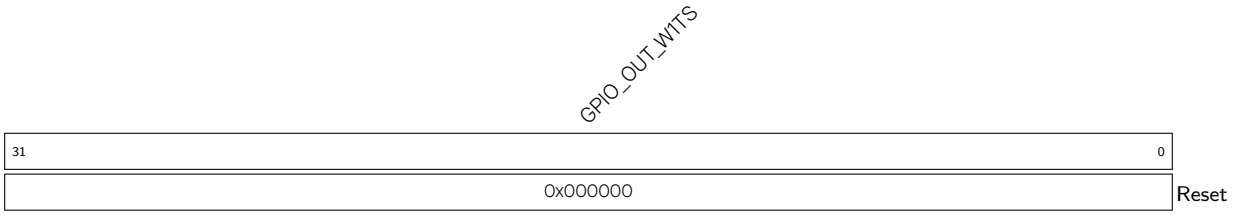
0: Low level

1: High level

The value of bit0 ~ bit27 correspond to the output value of GPIO0 ~ GPIO27 respectively. Bit28 ~ bit31 are invalid.

(R/W/SC/WTC)

Register 6.2. GPIO_OUT_W1TS_REG (0x0008)



GPIO_OUT_W1TS Configures whether or not to set the output register GPIO_OUT_REG of GPIO0 ~ GPIO27.

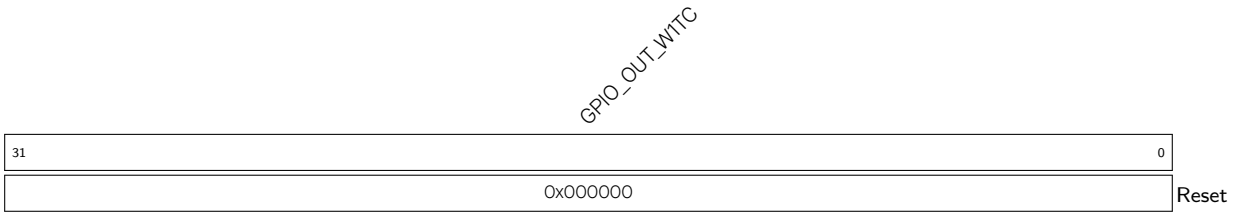
0: Not set

1: The corresponding bit in GPIO_OUT_REG will be set to 1

Bit0 ~ bit27 are corresponding to GPIO0 ~ GPIO27. Bit28 ~ bit31 are invalid. Recommended operation: use this register to set GPIO_OUT_REG.

(WT)

Register 6.3. GPIO_OUT_W1TC_REG (0x000C)



GPIO_OUT_W1TC Configures whether or not to clear the output register GPIO_OUT_REG of GPIO0 ~ GPIO27 output.

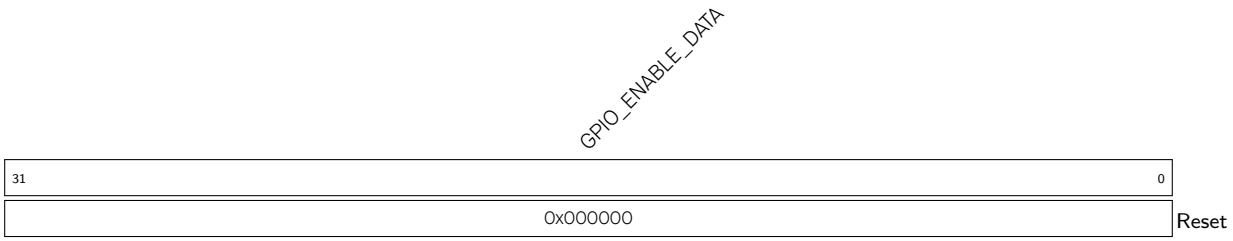
0: Not clear

1: The corresponding bit in GPIO_OUT_REG will be cleared.

bit0 ~ bit27 are corresponding to GPIO0 ~ GPIO27. Bit28 ~ bit31 are invalid. Recommended operation: use this register to clear GPIO_OUT_REG.

(WT)

Register 6.4. GPIO_ENABLE_REG (0x0020)



GPIO_ENABLE_DATA Configures whether or not to enable the output of GPIO0 ~ GPIO27.

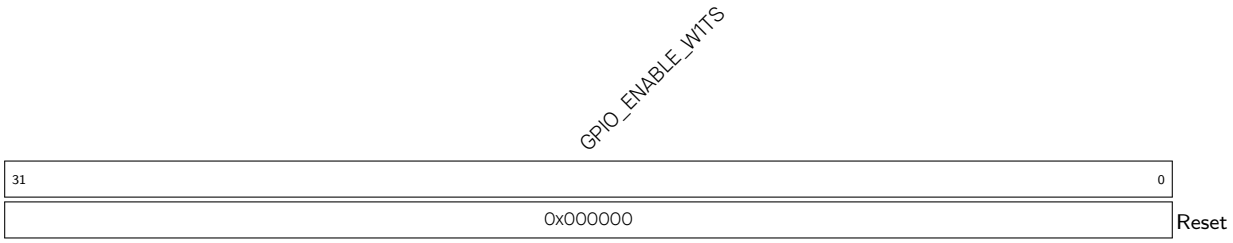
0: Not enable

1: Enable

Bit0 ~ bit27 are corresponding to GPIO0 ~ GPIO27. Bit28 ~ bit31 are invalid.

(R/W/WTC)

Register 6.5. GPIO_ENABLE_W1TS_REG (0x0024)



GPIO_ENABLE_W1TS Configures whether or not to set the output enable register GPIO_ENABLE_REG of GPIO0 ~ GPIO27.

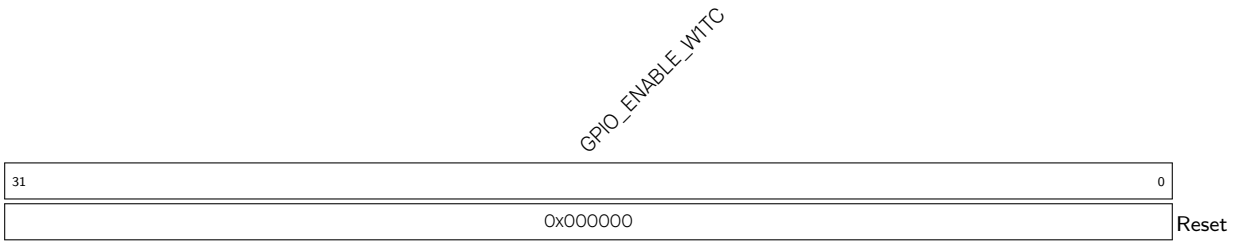
0: Not set

1: The corresponding bit in GPIO_ENABLE_REG will be set to 1

Bit0 ~ bit27 are corresponding to GPIO0 ~ GPIO27. Bit28 ~ bit31 are invalid. Recommended operation: use this register to set GPIO_ENABLE_REG.

(WT)

Register 6.6. GPIO_ENABLE_W1TC_REG (0x0028)



GPIO_ENABLE_W1TC Configures whether or not to clear the output enable register GPIO_ENABLE_REG of GPIO0 ~ GPIO27.

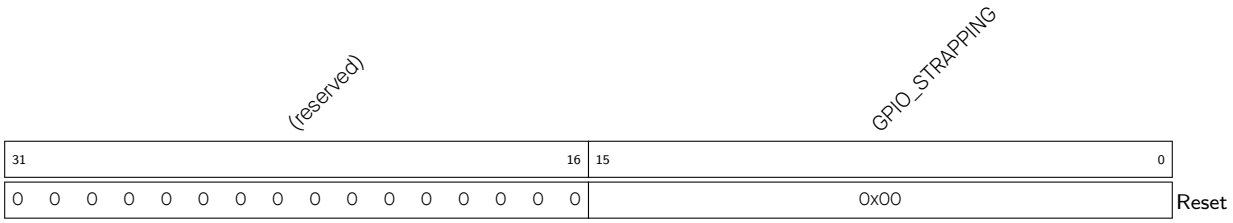
0: Not clear

1: The corresponding bit in GPIO_ENABLE_REG will be cleared

Bit0 ~ bit27 are corresponding to GPIO0 ~ GPIO27. Bit28 ~ bit31 are invalid. Recommended operation: use this register to clear GPIO_ENABLE_REG.

(WT)

Register 6.7. GPIO_STRAP_REG (0x0038)

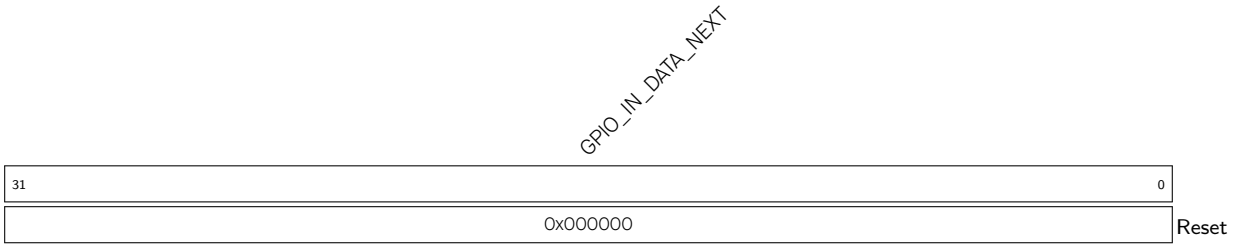


GPIO_STRAPPING Represents the values of GPIO Strapping pins.

- bit0: GPIO2 (GPIO2 should be reserved as a Strapping pin only when using SPI Download Boot mode.)
- bit1: GPIO3 (GPIO3 should be reserved as a Strapping pin only when using SPI Download Boot mode.)
- bit2: GPIO8 (this value will also be affected by EFUSE_DIS_FORCE_DOWNLOAD and LP_AON_FORCE_DOWNLOAD_BOOT.)
- bit3: GPIO9 (this value will also be affected by EFUSE_DIS_FORCE_DOWNLOAD and LP_AON_FORCE_DOWNLOAD_BOOT.)
- bit4: GPIO25
- bit5 ~ bit15: invalid

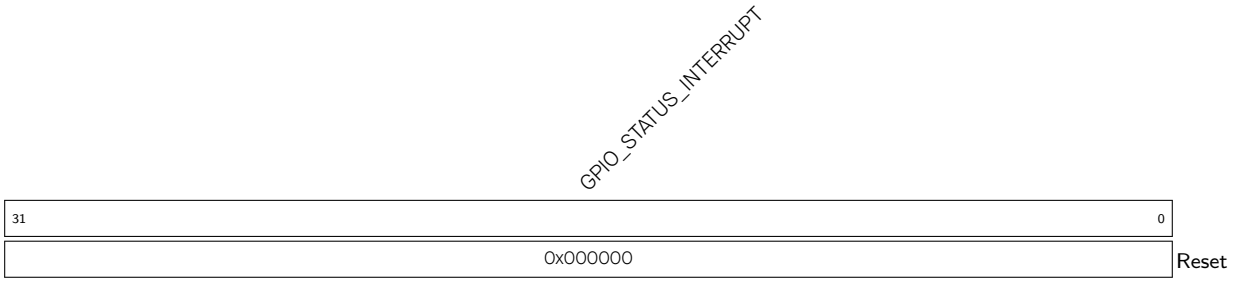
For more details about GPIO Strapping pins, please refer to the Chapter 8 *Chip Boot Control*.
(RO)

Register 6.8. GPIO_IN_REG (0x003C)



GPIO_IN_DATA_NEXT Represents the input value of GPIO0 ~ GPIO27. Each bit represents a pin input value:
0: Low level
1: High level
bit0 ~ bit27 are corresponding to GPIO0 ~ GPIO27. Bit28 ~ bit31 are invalid.
(RO)

Register 6.9. GPIO_STATUS_REG (0x0044)

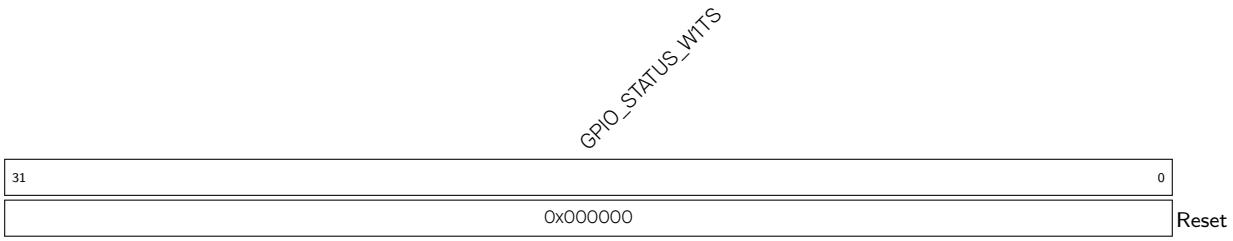


GPIO_STATUS_INTERRUPT The interrupt status of GPIO0 ~ GPIO27, can be configured by the software.

- Bit0 ~ bit27 are corresponding to GPIO0 ~ GPIO27. Bit28 ~ bit31 are invalid.
- 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 6.10. GPIO_STATUS_W1TS_REG (0x0048)

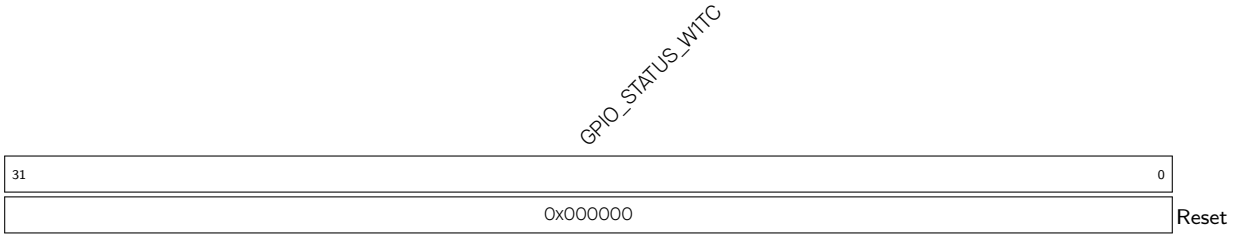


GPIO_STATUS_W1TS Configures whether or not to set the interrupt status register GPIO_STATUS_INTERRUPT of GPIO0 ~ GPIO27.

- Bit0 ~ bit27 are corresponding to GPIO0 ~ GPIO27. Bit28 ~ bit31 are invalid.
- If the value 1 is written to a bit here, the corresponding bit in GPIO_STATUS_INTERRUPT will be set to 1.
- Recommended operation: use this register to set GPIO_STATUS_INTERRUPT.

(WT)

Register 6.11. GPIO_STATUS_W1TC_REG (0x004C)

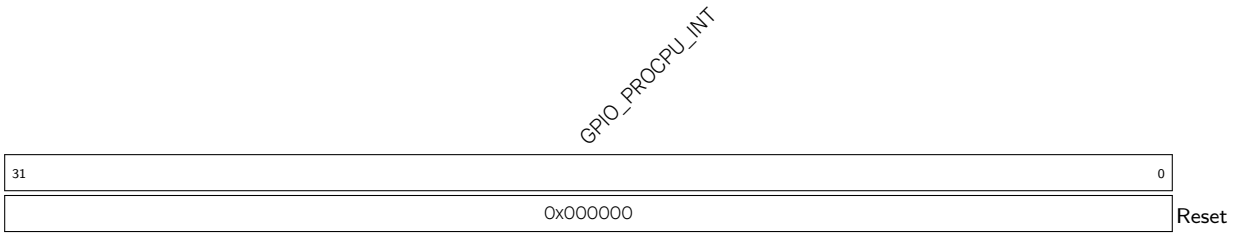


GPIO_STATUS_W1TC Configures whether or not to clear the interrupt status register GPIO_STATUS_INTERRUPT of GPIO0 ~ GPIO27.

- bit0 ~ bit27 are corresponding to GPIO0 ~ GPIO27. Bit28 ~ bit31 are invalid.
- If the value 1 is written to a bit here, the corresponding bit in GPIO_STATUS_INTERRUPT will be cleared.
- Recommended operation: use this register to clear GPIO_STATUS_INTERRUPT.

(WT)

Register 6.12. GPIO_PCPU_INT_REG (0x005C)



GPIO_PROCPU_INT Represents the CPU interrupt status of GPIO0 ~ GPIO27.

- bit0 ~ bit27 are corresponding to GPIO0 ~ GPIO27. Bit28 ~ bit31 are invalid.
- This interrupt status is corresponding to the bit in GPIO_STATUS_REG when assert (high) enable signal (bit13 of GPIO_PIN*n*_REG). 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.

(RO)

Register 6.13. GPIO_PIN*n*_REG (*n*: 0-27) (0x0074+4n*)**

GPIO_PIN*n*_SYNC2_BYPASS Configures whether or not to synchronize GPIO input data on either edge of 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 IO MUX operating clock for the first-level synchronization.

0: Not synchronize

1: Synchronize on falling edge

2: Synchronize on rising edge

3: Synchronize on rising edge

(R/W)

GPIO_PIN*n*_INT_TYPE Configures GPIO interrupt type.

0: GPIO interrupt disabled

1: Rising edge trigger

2: Falling edge trigger

3: Any edge trigger

4: Low level trigger

5: High level trigger

(R/W)

GPIO_PIN*n*_WAKEUP_ENABLE Configures whether or not to enable GPIO wake-up function.

0: Disable

1: Enable

This function only wakes up the CPU from Light-sleep.

(R/W)

Continued on the next page...

Register 6.13. GPIO_PIN*n*_REG (*n*: 0-27) (0x0074+4**n*)

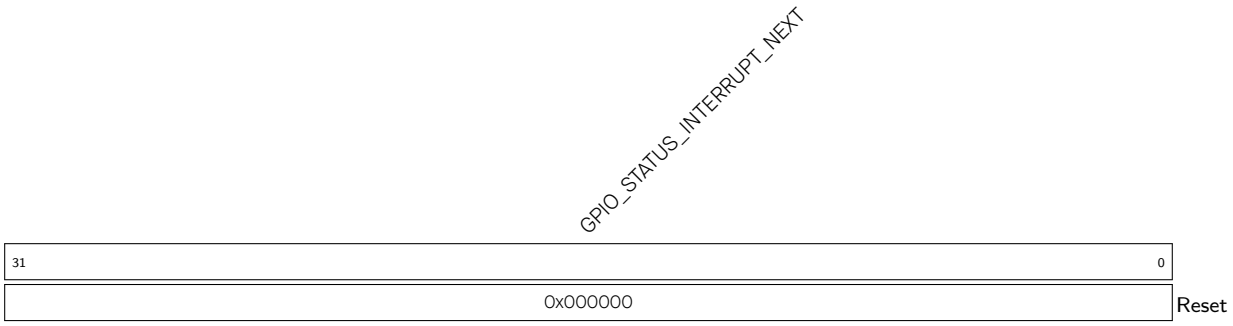
Continued from the previous page...

GPIO_PIN*n*_INT_ENA Configures whether or not to enable CPU interrupt or CPU non-maskable interrupt.

- bit13: Configures whether or not to enable CPU interrupt:
  0: Disable
  1: Enable
- bit14: Configures CPU non-maskable interrupt:
  0: Disable
  1: Enable
- bit15 ~ bit17: invalid

(R/W)

Register 6.14. GPIO_STATUS_NEXT_REG (0x014C)

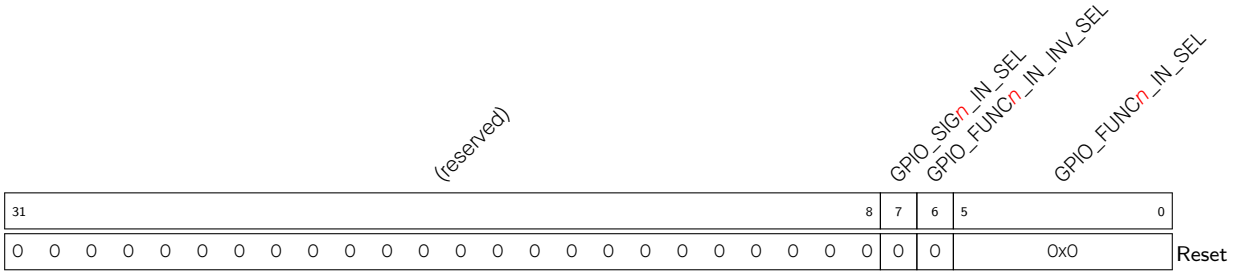


GPIO_STATUS_INTERRUPT_NEXT Represents the interrupt source signal of GPIO0 ~ GPIO27. Bit0 ~ bit27 are corresponding to GPIO0 ~ GPIO27. Bit28 ~ bit31 are invalid. Each bit represents:

- 0: The GPIO does not generate the interrupt configured by GPIO_PIN*n*_INT_TYPE.
- 1: The GPIO generates an interrupt configured by GPIO_PIN*n*_INT_TYPE.

The interrupt could be rising edge interrupt, falling edge interrupt, level sensitive interrupt and any edge interrupt.
(RO)

Register 6.15. GPIO_FUNC*n*_IN_SEL_CFG_REG (*n*: 0-127) (0x0154+4n*)**



GPIO_FUNC*n*_IN_SEL Configures to select a pin from the 28 GPIO pins to connect the input signal *n*.

0: Select GPIO0

1: Select GPIO1

.....

26: Select GPIO26

27: Select GPIO27

Or

0x38: A constantly high input

0x3C: A constantly low input

(R/W)

GPIO_FUNC*n*_IN_INV_SEL Configures whether or not to invert the input value.

0: Not invert

1: Invert

(R/W)

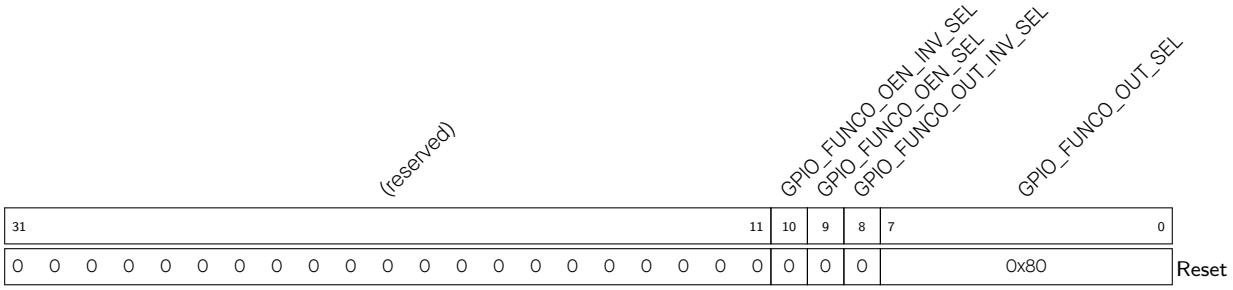
GPIO_SIG*n*_IN_SEL Configures whether or not to route signals via GPIO matrix.

0: Bypass GPIO matrix, i.e., connect signals directly to peripheral configured in IO MUX.

1: Route signals via GPIO matrix.

(R/W)

Register 6.16. GPIO_FUNC*n*_OUT_SEL_CFG_REG (*n*: 0-27) (0x0554+4n*)**



GPIO_FUNC*n*_OUT_SEL Configures to select a signal *Y* (0 <= *Y* < 128) from 128 peripheral signals to be output from GPIO*n*.

0: Select signal 0

1: Select signal 1

.....

126: Select signal 126

127: Select signal 127

Or

128: 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 6.12-1.

(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*_OEN_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)

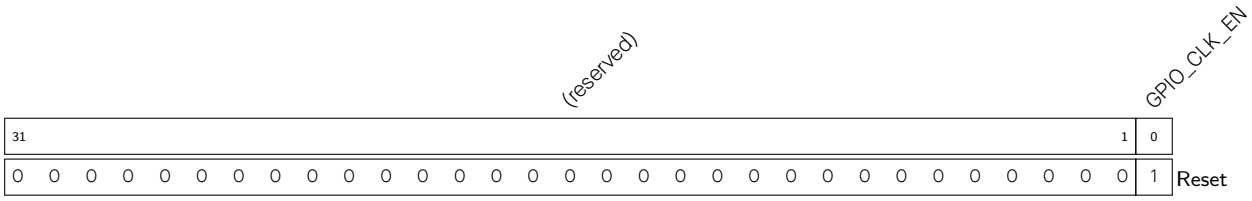
GPIO_FUNC*n*_OEN_INV_SEL Configures whether or not to invert the output enable signal.

0: Not invert

1: Invert

(R/W)

Register 6.17. GPIO_CLOCK_GATE_REG (0x062C)



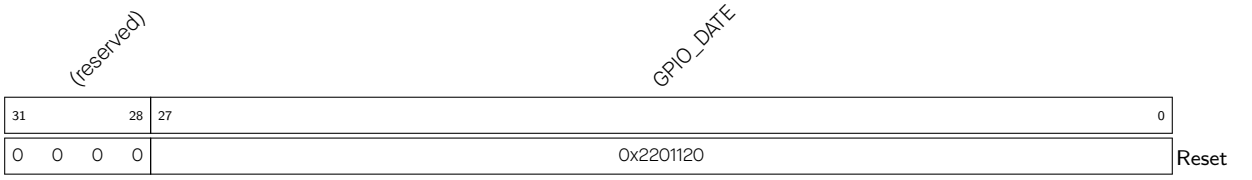
GPIO_CLK_EN Configures whether or not to enable clock gate.

0: Not enable

1: Enable, the clock is free running.

(R/W)

Register 6.18. GPIO_DATE_REG (0x06FC)

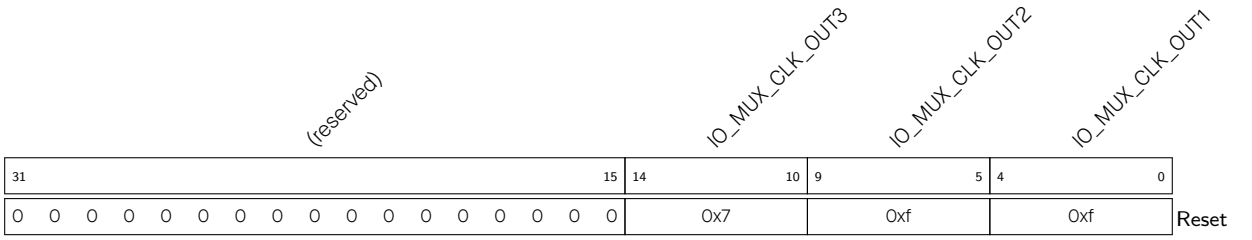


GPIO_DATE Version control register. (R/W)

6.18.2 IO MUX Registers

The addresses in this section are relative to the IO MUX base address provided in Table 4.3-2 in Chapter 4 *System and Memory*.

Register 6.19. IO_MUX_PIN_CTRL_REG (0x0000)



IO_MUX_CLK_OUT*x* Configures the output clock for I2S.

0x0: Select CLK_OUT_out*x* for I2S output clock.

CLK_OUT_out*x* can be found in Table 6.12-1.

(R/W)

Register 6.20. IO_MUX_GPIO*n*_REG (*n*: 0-27) (0x0004+4n*)**

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 6.20. IO_MUX_GPIO*n*_REG (*n*: 0-27) (0x0004+4**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 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)

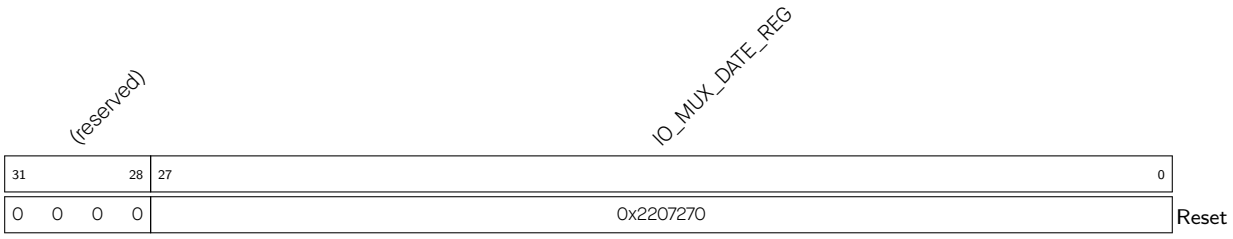
IO_MUX_GPIO*n*_HYS_SEL Configures to choose the signal for enabling the hysteresis function for GPIO*n*.

0□Choose the output enable signal of eFuse

1: Choose the output enable signal of IO_MUX_GPIO*n*_HYS_EN.

(R/W)

Register 6.21. IO_MUX_DATE_REG (0x00FC)

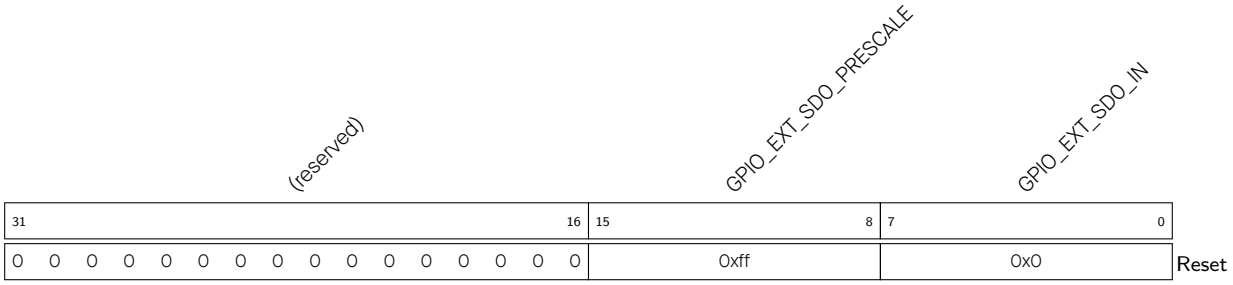


IO_MUX_DATE_REG Version control register. (R/W)

6.18.3 GPIO_EXT Registers

The addresses in this section are relative to (GPIO base address + 0x0F00). GPIO base address is provided in Table 4.3-2 in Chapter 4 *System and Memory*.

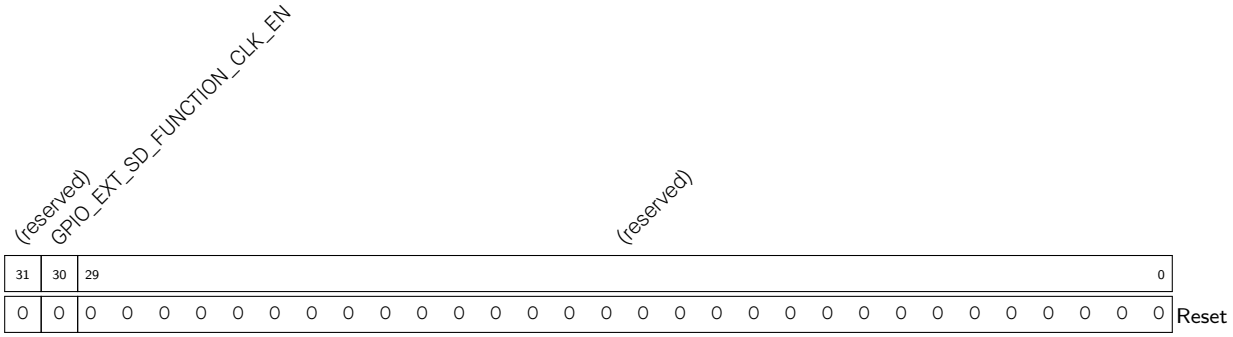
Register 6.22. GPIO_EXT_SIGMADELTA*n*_REG (*n*: 0-3) (0x0000+0x4**n*)



GPIO_EXT_SDO_IN Configures the duty cycle of sigma delta modulation output. (R/W)

GPIO_EXT_SDO_PRESCALE Configures the divider value to divide IO MUX operating clock. (R/W)

Register 6.23. GPIO_EXT_SIGMADELTA_MISC_REG (0x0024)



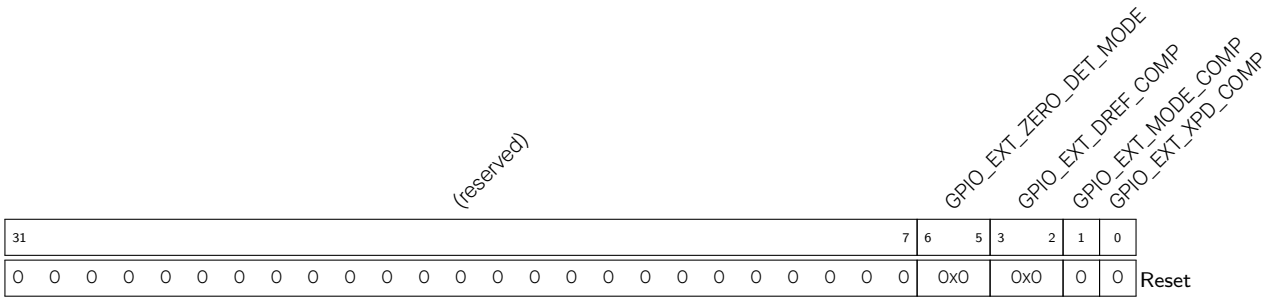
GPIO_EXT_SD_FUNCTION_CLK_EN Configures whether or not to enable the clock for sigma delta modulation.

0: Not enable

1: Enable

(R/W)

Register 6.24. GPIO_EXT_PAD_COMP_CONFIG_REG (0x0028)



GPIO_EXT_ZERO_DET_MODE Configures the mode of generating interrupt sources for analog PAD voltage comparator.

0: Disable interrupt source generation.

1: Reserved.

2: Reserved.

3: Enable interrupt source generated by any edge of the PAD_COMP_OUT_sync signal.

(R/W)

GPIO_EXT_DREF_COMP Configures the internal reference voltage for analog PAD voltage comparator.

0: Internal reference voltage is 0 * VDDPST2

1: Internal reference voltage is 0.1 * VDDPST2

.....

6: Internal reference voltage is 0.6 * VDDPST2

7: Internal reference voltage is 0.7 * VDDPST2

(R/W)

GPIO_EXT_MODE_COMP Configures the reference voltage for analog PAD voltage comparator.

0: Reference voltage is the internal reference voltage.

1: Reference voltage is the voltage on the GPIO10 PAD.

(R/W)

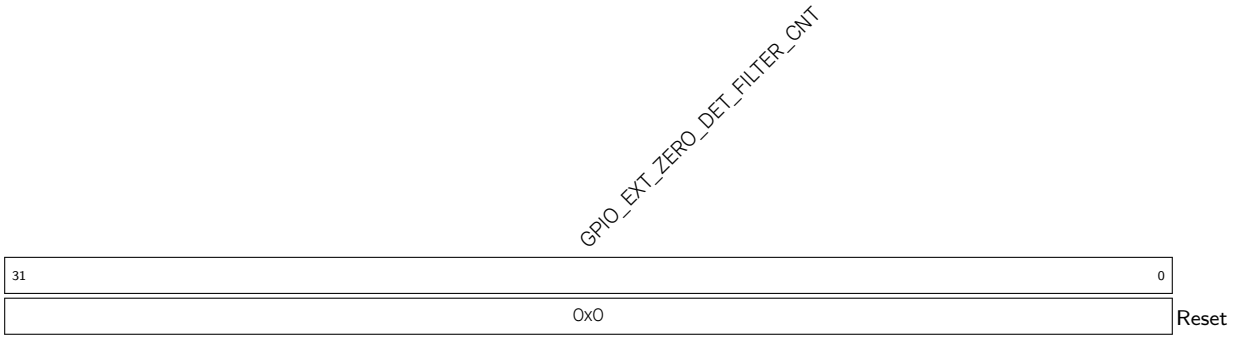
GPIO_EXT_XPD_COMP Configures whether to enable the function of analog PAD voltage comparator.

0: Disable

1: Enable

(R/W)

Register 6.25. GPIO_EXT_PAD_COMP_FILTER_REG (0x002C)

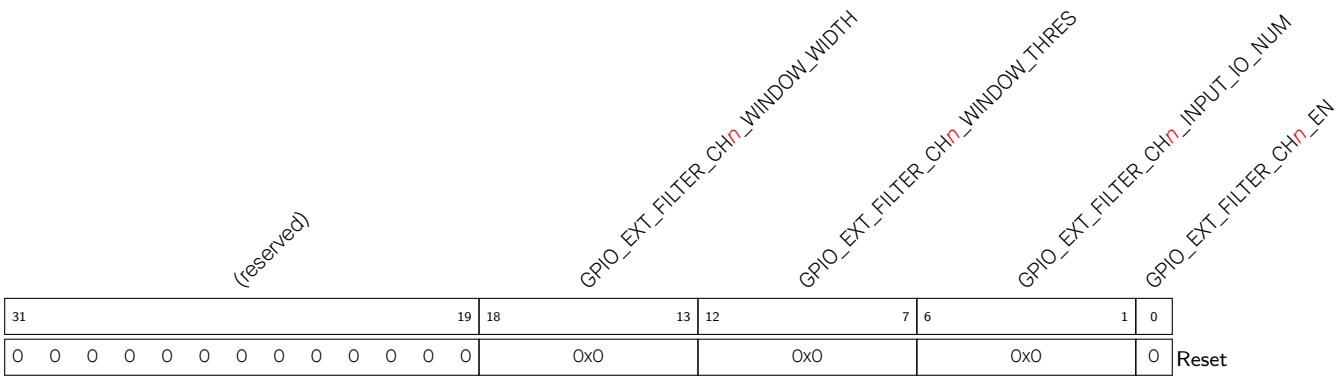


GPIO_EXT_ZERO_DET_FILTER_CNT Configures the period of masking new interrupt source for analog PAD voltage comparator.

Measurement unit: IO MUX operating clock cycle

(R/W)

Register 6.26. GPIO_EXT_GLITCH_FILTER_CH*n*_REG (*n*: 0-7) (0x0030+0x4**n*)



GPIO_EXT_FILTER_CH*n*_EN Configures whether or not to enable channel *n* of Glitch Filter.

0: Not enable

1: Enable

(R/W)

GPIO_EXT_FILTER_CH*n*_INPUT_IO_NUM Configures to select the input GPIO for Glitch Filter.

0: Select GPIO0

1: Select GPIO1

.....

26: Select GPIO26

27: Select GPIO27

(R/W)

GPIO_EXT_FILTER_CH*n*_WINDOW_THRES Configures the window threshold for Glitch Filter. The window threshold should be less than or equal to GPIO_EXT_FILTER_CH*n*_WINDOW_WIDTH.

Measurement unit: IO MUX operating clock cycle

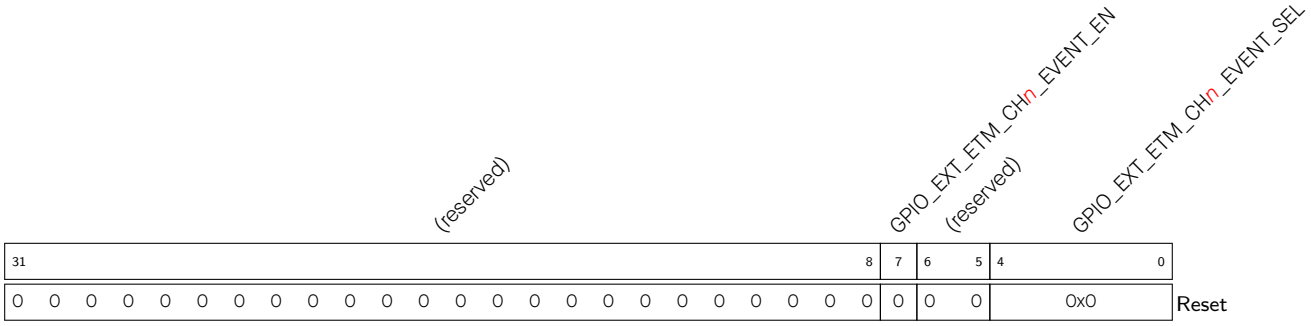
(R/W)

GPIO_EXT_FILTER_CH*n*_WINDOW_WIDTH Configures the window width for Glitch Filter. The valid range for the window width is 0 ~ 62, and 63 is a reserved value that cannot be used.

Measurement unit: IO MUX operating clock cycle

(R/W)

Register 6.27. GPIO_EXT_ETM_EVENT_CH*n*_CFG_REG (*n*: 0-7) (0x0060+0x4n*)**



GPIO_EXT_ETM_CH*n*_EVENT_SEL Configures to select GPIO for ETM event channel.

0: Select GPIO0

1: Select GPIO1

.....

26: Select GPIO26

27: Select GPIO27

(R/W)

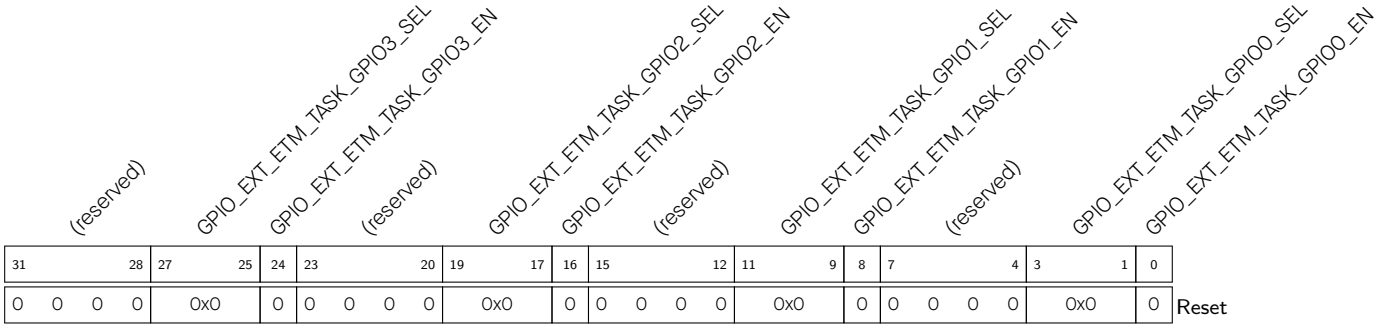
GPIO_EXT_ETM_CH*n*_EVENT_EN Configures whether or not to enable ETM event send.

0: Not enable

1: Enable

(R/W)

Register 6.28. GPIO_EXT_ETM_TASK_P0_CFG_REG (0x00A0)



GPIO_EXT_ETM_TASK_GPIO*n*_EN (*n* = 0 ∼ 3) Configures whether or not to enable GPIO*n* to response ETM task.

0: Not enable

1: Enable

(R/W)

GPIO_EXT_ETM_TASK_GPIO*n*_SEL (*n* = 0 ∼ 3) Configures to select an ETM task channel for GPIO*n*.

0: Select channel 0

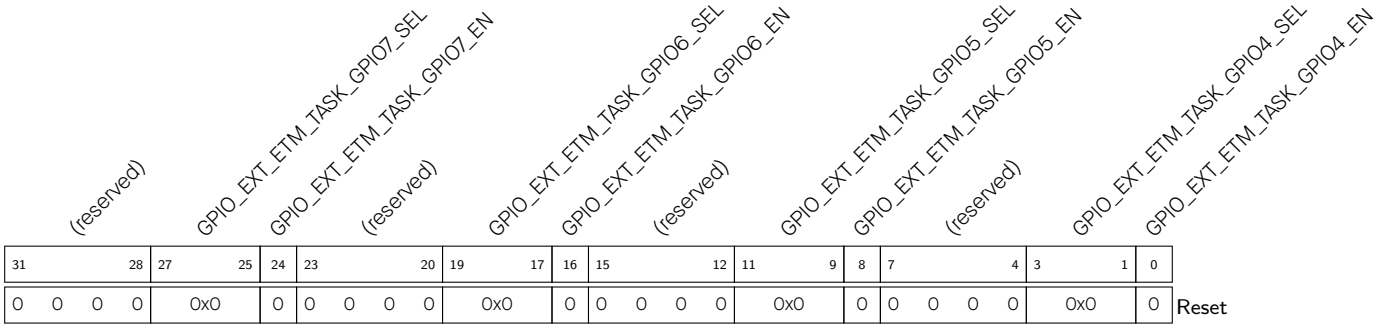
1: Select channel 1

.....

7: Select channel 7

(R/W)

Register 6.29. GPIO_EXT_ETM_TASK_P1_CFG_REG (0x00A4)



GPIO_EXT_ETM_TASK_GPIO*n*_EN (*n* = 4 ∼ 7) Configures whether or not to enable GPIO*n* to response ETM task.

0: Not enable

1: Enable

(R/W)

GPIO_EXT_ETM_TASK_GPIO*n*_SEL (*n* = 4 ∼ 7) Configures to select an ETM task channel for GPIO*n*.

0: Select channel 0

1: Select channel 1

.....

7: Select channel 7

(R/W)

Register 6.30. GPIO_EXT_ETM_TASK_P2_CFG_REG (0x00A8)

GPIO_EXT_ETM_TASK_GPIO*n*_EN (*n* = 8 ∼ 11) Configures whether or not to enable GPIO*n* to response ETM task.

0: Not enable

1: Enable

(R/W)

GPIO_EXT_ETM_TASK_GPIO*n*_SEL (*n* = 8 ∼ 11) Configures to select an ETM task channel for GPIO*n*.

0: Select channel 0

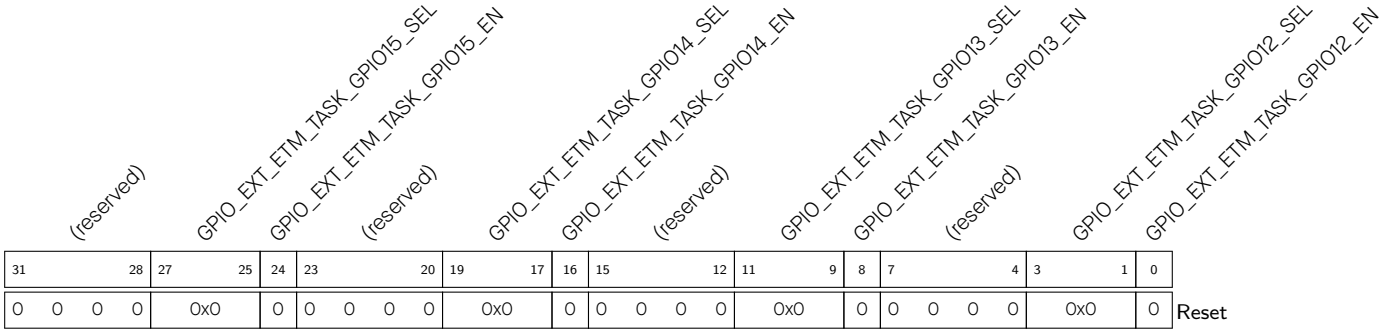
1: Select channel 1

.....

7: Select channel 7

(R/W)

Register 6.31. GPIO_EXT_ETM_TASK_P3_CFG_REG (0x00AC)



GPIO_EXT_ETM_TASK_GPIO*n*_EN (*n* = 12 ∼ 15) Configures whether or not to enable GPIO*n* to response ETM task.

0: Not enable

1: Enable

(R/W)

GPIO_EXT_ETM_TASK_GPIO*n*_SEL (*n* = 12 ∼ 15) Configures to select an ETM task channel for GPIO*n*.

0: Select channel 0

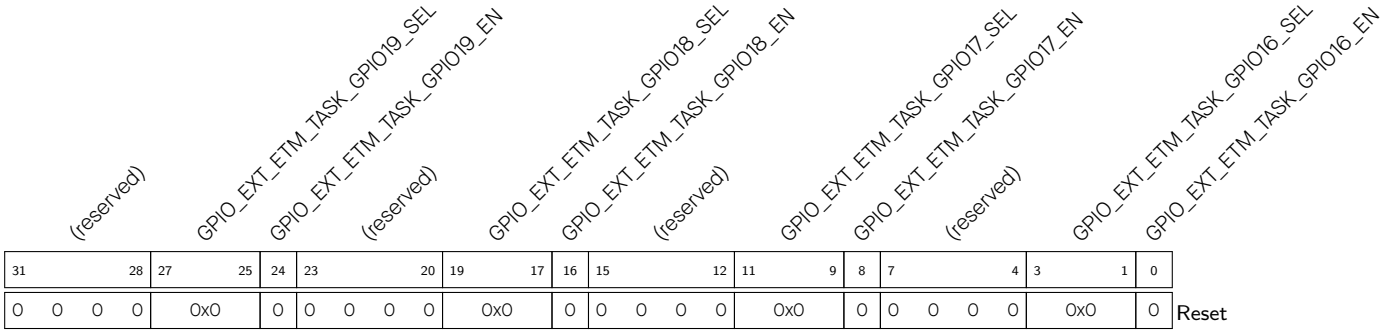
1: Select channel 1

.....

7: Select channel 7

(R/W)

Register 6.32. GPIO_EXT_ETM_TASK_P4_CFG_REG (0x00B0)



GPIO_EXT_ETM_TASK_GPIO*n*_EN (*n* = 16 ∼ 19) Configures whether or not to enable GPIO*n* to response ETM task.

0: Not enable

1: Enable

(R/W)

GPIO_EXT_ETM_TASK_GPIO*n*_SEL (*n* = 16 ∼ 19) Configures to select an ETM task channel for GPIO*n*.

0: Select channel 0

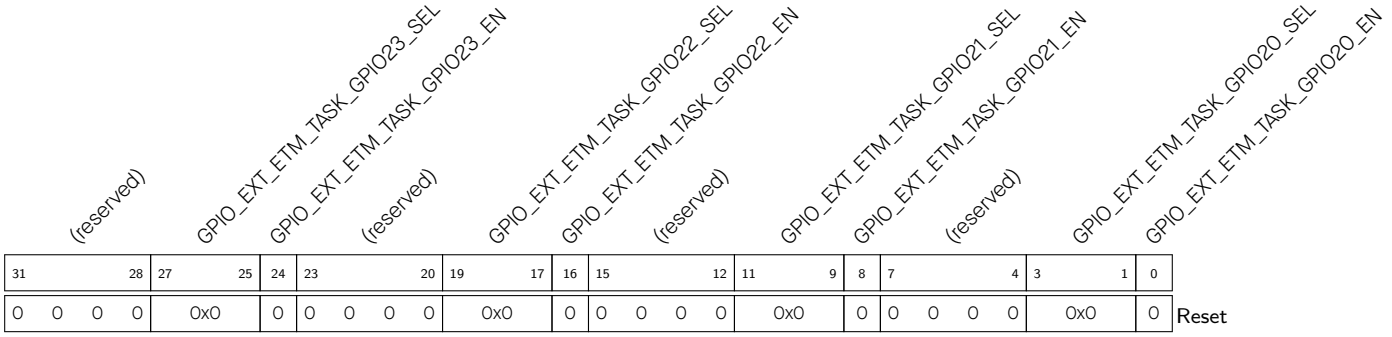
1: Select channel 1

.....

7: Select channel 7

(R/W)

Register 6.33. GPIO_EXT_ETM_TASK_P5_CFG_REG (0x00B4)



GPIO_EXT_ETM_TASK_GPIO*n*_EN (*n* = 20 ∼ 23) Configures whether or not to enable GPIO*n* to response ETM task.

0: Not enable

1: Enable

(R/W)

GPIO_EXT_ETM_TASK_GPIO*n*_SEL (*n* = 20 ∼ 23) Configures to select an ETM task channel for GPIO*n*.

0: Select channel 0

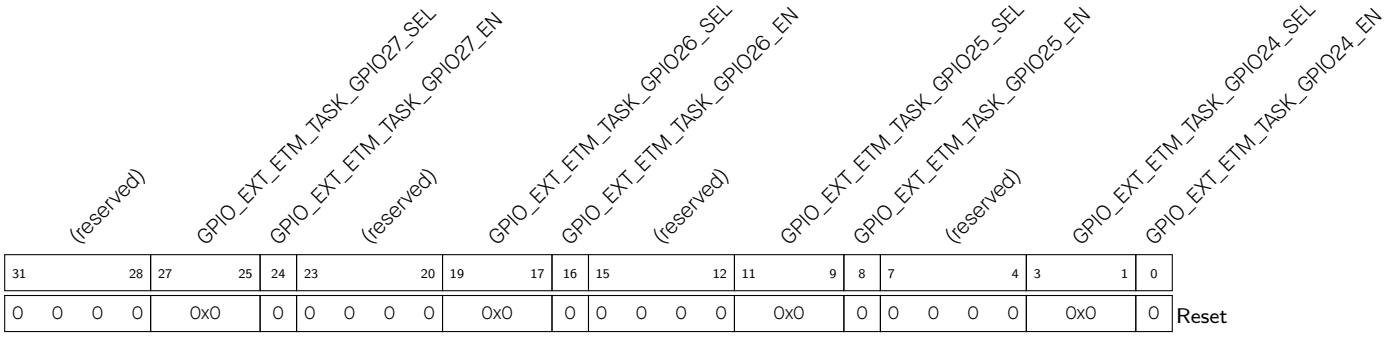
1: Select channel 1

.....

7: Select channel 7

(R/W)

Register 6.34. GPIO_EXT_ETM_TASK_P6_CFG_REG (0x00B8)



GPIO_EXT_ETM_TASK_GPIO*n*_EN (*n* = 24 ∼ 27) Configures whether or not to enable GPIO*n* to response ETM task.

0: Not enable

1: Enable

(R/W)

GPIO_EXT_ETM_TASK_GPIO*n*_SEL (*n* = 24 ∼ 27) Configures to select an ETM task channel for GPIO*n*.

0: Select channel 0

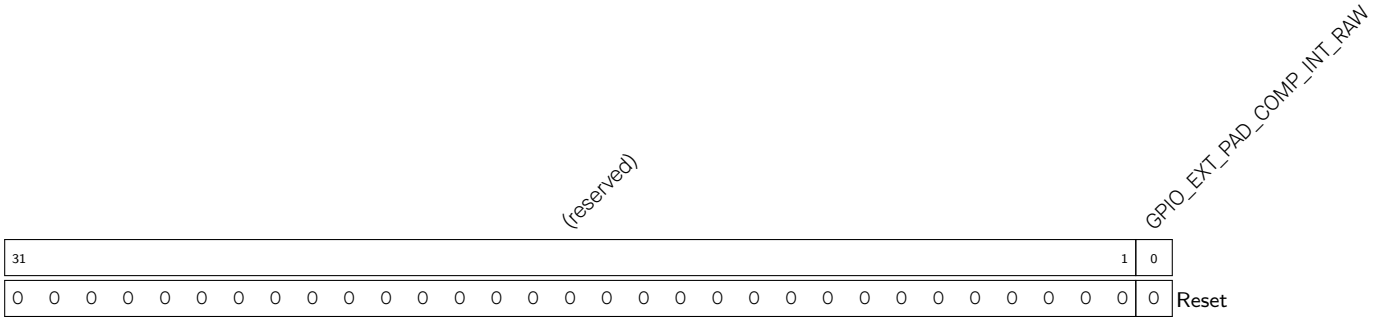
1: Select channel 1

.....

7: Select channel 7

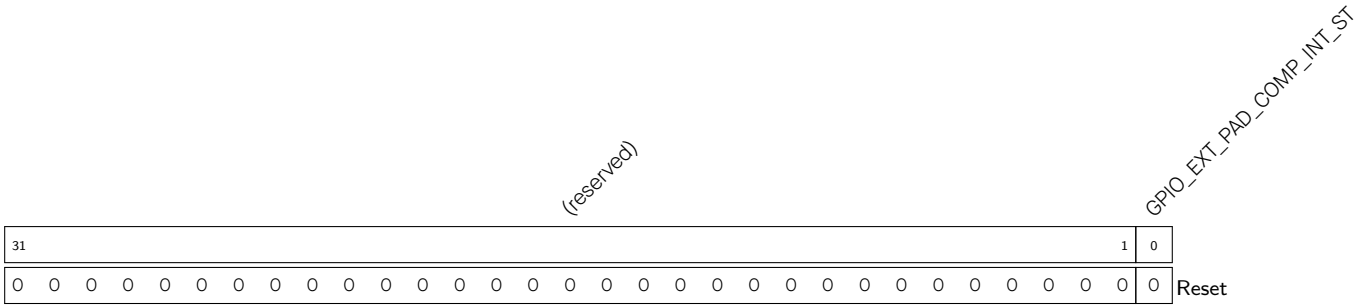
(R/W)

Register 6.35. GPIO_EXT_INT_RAW_REG (0x00E0)



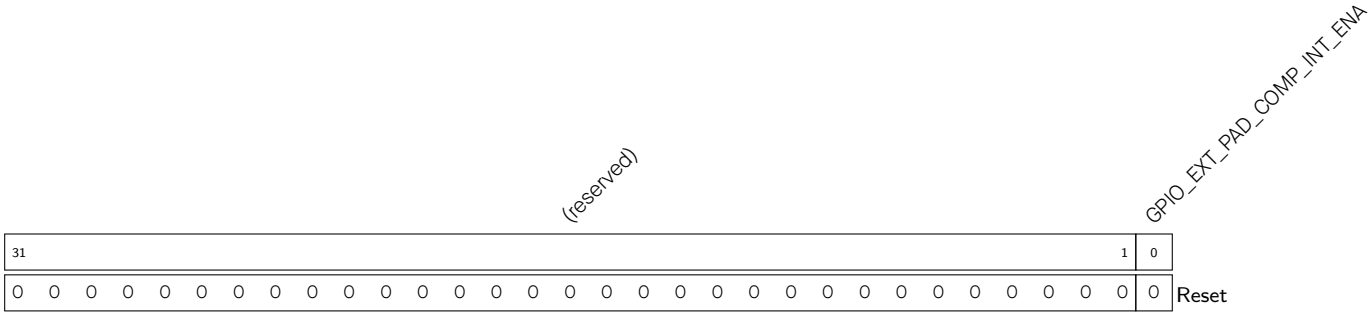
GPIO_EXT_PAD_COMP_INT_RAW Represents the raw interrupt bit of GPIO_EXT_PAD_COMP_INT. (RO/WTC/SS)

Register 6.36. GPIO_EXT_INT_ST_REG (0x00E4)



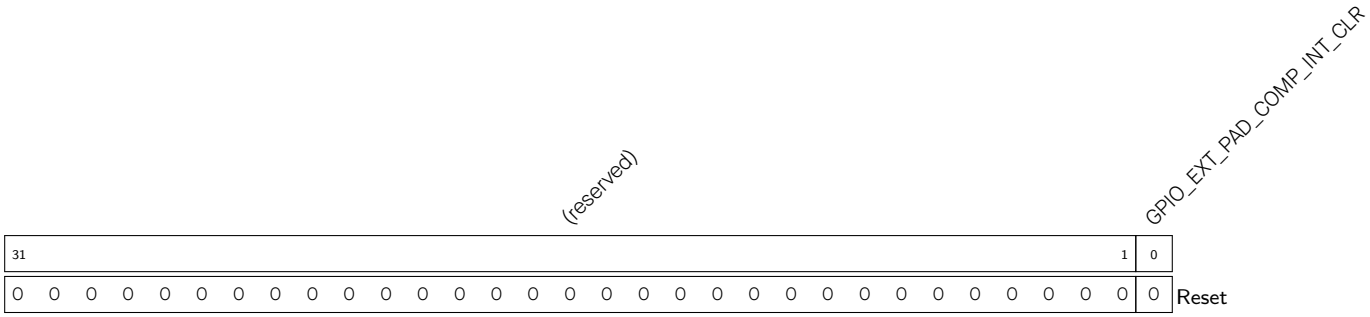
GPIO_EXT_PAD_COMP_INT_ST Represents the interrupt status bit of GPIO_EXT_PAD_COMP_INT. (RO)

Register 6.37. GPIO_EXT_INT_ENA_REG (0x00E8)



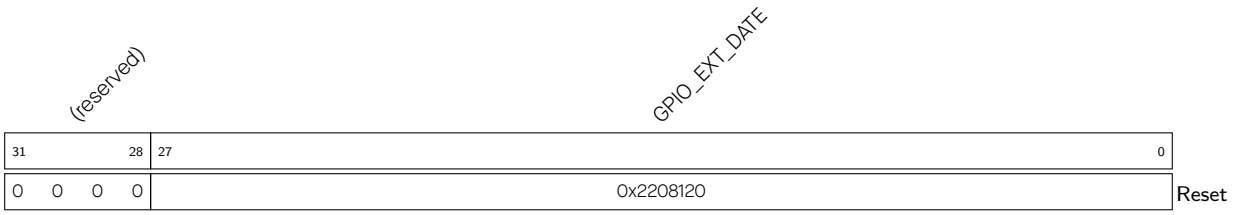
GPIO_EXT_PAD_COMP_INT_ENA Represents the interrupt enable bit of GPIO_EXT_PAD_COMP_INT. (RW)

Register 6.38. GPIO_EXT_INT_CLR_REG (0x00EC)



GPIO_EXT_PAD_COMP_INT_CLR Represents the interrupt clear bit of GPIO_EXT_PAD_COMP_INT. (WT)

Register 6.39. GPIO_EXT_VERSION_REG (0x00FC)



GPIO_EXT_DATE Version control register. (R/W)

Chapter 7

Reset and Clock

7.1 Reset

7.1.1 Overview

ESP32-H2 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 7.1-1 shows the scopes of affected subsystems by each type of reset.

7.1.2 Architectural Overview

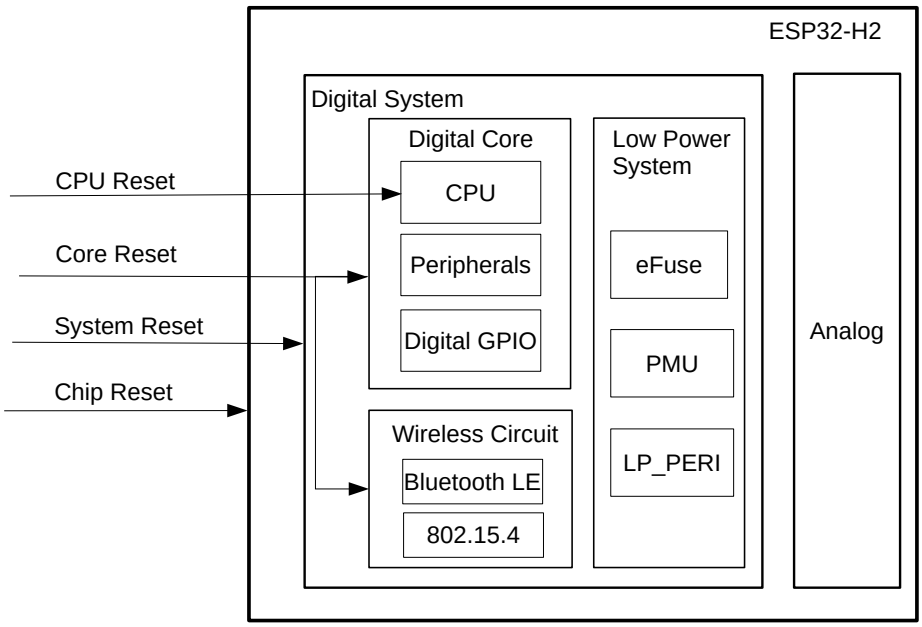


Figure 7.1-1. Reset Types

ESP32-H2's Digital System can be divided into two parts: High Performance System (HP system) that includes Digital Core, Wireless Circuit, and HP SRAM, and Low Power System (LP system) that only includes some low-power peripherals and LP SRAM. See Figure 7.1-1 for details (note that HP SRAM and LP SRAM would not be reset, so they are not shown in the figure).

7.1.3 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, Bluetooth® LE, and 802.15.4.
  - System Reset: resets the whole digital system, including LP system.
  - Chip Reset: resets the whole chip.
- Software reset and hardware reset:
  - Software Reset: triggered via software by configuring the corresponding registers of CPU, see Chapter 11 *Low-Power Management*.
  - Hardware Reset: triggered directly by the hardware.

7.1.4 Functional Description

CPU will be reset immediately when any type of reset above occurs. Users can retrieve reset source codes by reading RTC_CLKRST_RESET_CAUSE after the reset is released. Table 7.1-1 lists possible reset sources and the types of reset they trigger.

Table 7.1-1. Reset Source

Code	Source	Reset Type	Note
0x01	Chip reset[1]	Chip Reset	—
0x0F	Brown-out system reset	Chip Reset or System Reset	Triggered by brown-out detector[2]
0x10	RWDT system reset	System Reset	See Chapter 14 *Watchdog Timers (WDT)*
0x12	Super Watchdog reset	System Reset	See Chapter 14 *Watchdog Timers (WDT)*
0x03	Software system reset	Core Reset	Triggered by configuring LP_AON_HPSYS_SW_RESET
0x05	Deep-sleep reset	Core Reset	See Chapter 11 *Low-Power Management*
0x07	MWDT0 core reset	Core Reset	See Chapter 14 *Watchdog Timers (WDT)*
0x08	MWDT1 core reset	Core Reset	See Chapter 14 *Watchdog Timers (WDT)*
0x09	RWDT core reset	Core Reset	See Chapter 14 *Watchdog Timers (WDT)*
0x14	eFuse reset	Core Reset	Triggered by eFuse CRC error
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 33 *USB Serial/JTAG Controller (USB_SERIAL_JTAG)*
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 33 *USB Serial/JTAG Controller (USB_SERIAL_JTAG)*
0x0B	MWDT0 CPU reset	CPU Reset	See Chapter 14 *Watchdog Timers (WDT)*
0x0C	Software CPU reset	CPU Reset	Triggered by configuring LP_AON_CPU_CORE0_SW_RESET
0x0D	RWDT CPU reset	CPU Reset	See Chapter 14 *Watchdog Timers (WDT)*
0x11	MWDT1 CPU reset	CPU Reset	See Chapter 14 *Watchdog Timers (WDT)*
0x17	Voltage glitch reset	System Reset	Triggered by power supply glitch attack
0x18	JTAG CPU reset	CPU Reset	Triggered when a "JDB Resetting CPU" instruction is received

[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 11 *Low-Power Management*.

7.1.5 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-H2 peripherals are controlled by the Power/Clock/Reset (PCR) module. For reset

registers of peripherals, see Section 7.4 *Register Summary*.

7.2 Clock

7.2.1 Overview

ESP32-H2 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 7.2-1 shows the system clock structure.

7.2.2 Architectural Overview

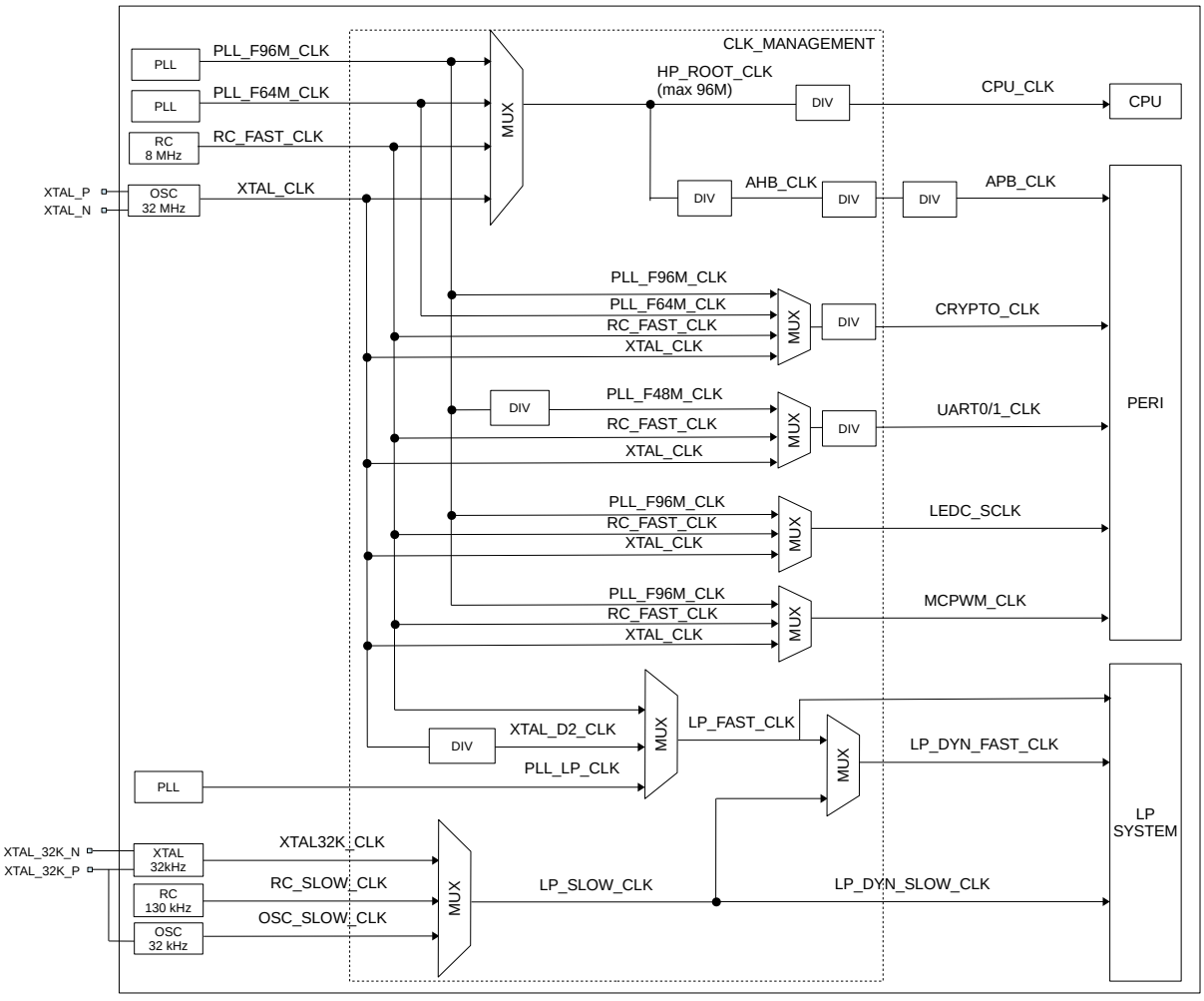


Figure 7.2-1. System Clock

7.2.3 Features

ESP32-H2 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_F96M_CLK (96 MHz): internal PLL clock. Its reference clock is XTAL_CLK.
  - PLL_F64M_CLK (64 MHz): internal PLL clock. Its reference clock is XTAL_CLK.
  - XTAL_CLK (32 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 (8 MHz by default): internal fast RC oscillator with adjustable frequency
  - RC_SLOW_CLK (130 kHz): internal slow RC oscillator
  - 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 6 *IO MUX and GPIO Matrix (GPIO, IO MUX)* > 6.9 *Pin Hold Feature*)
  - PLL_LP_CLK (8 MHz): internal PLL clock, with reference clock XTAL32K_CLK

7.2.4 Functional Description

7.2.4.1 HP System Clock

As Figure 7.2-1 shows, CPU_CLK is the master clock for CPU and its frequency can be as high as 96 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_96M_CLK, PLL_64M_CLK, or RC_FAST_CLK as the clock source of CPU_CLK by configuring PCR_SOC_CLK_SEL. For details, see Table 7.2-1 and Table 7.2-2. By default, the CPU clock is sourced from XTAL_CLK with a divider of 1, i.e., the CPU clock frequency is 32 MHz.

Table 7.2-1. CPU_CLK Clock Source

PCR_SOC_CLK_SEL	CPU Clock Source
0	XTAL_CLK
1	PLL_F96M_CLK
2	RC_FAST_CLK
3	PLL_F64M_CLK

Table 7.2-2. Frequency of CPU_CLK, AHB_CLK and HP_ROOT_CLK

Clock	Source	Frequency
HP_ROOT_CLK	PLL_F96M_CLK	96 MHz
	PLL_F64M_CLK	64 MHz
	XTAL_CLK	32 MHz
	RC_FAST_CLK	8 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 frequency, and must be an integer multiple of AHB_CLK frequency.

[2] AHB_CLK frequency can not exceed 32 MHz.

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 7.2-1, 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 chip's performance will degrade due to the automatic frequency reduction. This function can be disabled (already disabled by default) by configuring APB_DECREASE_DIV_NUM to 0.

7.2.4.2 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:
  - 16 MHz XTAL_D2_CLK, which is XTAL_CLK divided by 2
  - RC_FAST_CLK
  - PLL_LP_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 11 *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

7.2.4.3 Peripheral Clocks

Table 7.2-3, Table 7.2-4, Table 7.2-5, and Table 7.2-6 list the derived HP/LP clocks sources and HP/LP clocks for each peripheral.

Table 7.2-3. Derived HP Clock Source

Derived Clock	Source Clock			Derived Clock	Source Clock					Derived Clock									Source Clock
	XTAL_CLK 32 MHz	PLL_CLK			RC_FAST_CLK 8 MHz	RC_SLOW_CLK 130 kHz	OSC_SLOW_CLK 32 kHz	XTAL32K_CLK 32 kHz	PLL_LP_CLK 8 MHz	HP_ROOT_CLK 96 MHz/64 MHz/32 MHz/8 MHz	CRYPTO_CLK	APB_CLK	AHB_CLK	CPU_CLK	LP_DYN_ FAST_CLK	LP_DYN_ SLOW_CLK	XTAL_D2_CLK 16 MHz	LP_FAST_CLK	
		PLL_F96M_CLK 96 MHz	PLL_F64M_CLK 64 MHz	PLL_F48M_CLK 48 MHz															Clock from IO
PLL_F48M_CLK		Y																	
HP_ROOT_CLK	Y	Y	Y		Y														
CRYPTO_CLK	Y	Y	Y		Y														
APB_CLK										Y									
AHB_CLK										Y									
CPU_CLK										Y									

Table 7.2-4. HP Clocks Used by Each Peripheral

Peripheral	Source Clock			Derived Clock	Source Clock					Derived Clock									Source Clock
	XTAL_CLK 32 MHz	PLL_CLK			RC_FAST_CLK 8 MHz	RC_SLOW_CLK 130 kHz	OSC_SLOW_CLK 32 kHz	XTAL32K_CLK 32 kHz	PLL_LP_CLK 8 MHz	HP_ROOT_CLK 96 MHz/64 MHz/32 MHz/8 MHz	CRYPTO_CLK	APB_CLK	AHB_CLK	CPU_CLK	LP_DYN_ FAST_CLK	LP_DYN_ SLOW_CLK	XTAL_D2_CLK 16 MHz	LP_FAST_CLK	
		PLL_F96M_CLK 96 MHz	PLL_F64M_CLK 64 MHz	PLL_F48M_CLK 48 MHz															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														
Remote Control Peripheral (RMT)	Y				Y														
Motor Control PWM (MCPWM)	Y	Y			Y														
I2C Controller (I2C)	Y				Y														
SPI2	Y			Y	Y														
SAR ADC	Y	Y			Y														
USB Serial/JTAG Controller (USB_SERIAL_JTAG)				Y															
Two-wire Automotive Interface (TWAI)	Y				Y														
LED PWM Controller (LEDC)	Y	Y			Y														
System Timer (SYSTIMER)	Y				Y														
Parallel IO Controller (PARLIO)	Y	Y			Y														parl_rx_clk_in parl_tx_clk_in
IO MUX	Y			Y	Y														
AES Accelerator (AES) SHA Accelerator (SHA) RSA Accelerator (RSA) ECC Accelerator (ECC) RSA Digital Signature Peripheral (RSA_DS) HMAC Accelerator (HMAC) ECDSA Digital Signature Peripheral (ECDSA_DS)											Y								
Interrupt Matrix (INTMTX)													Y						
Pulse Count Controller (PCNT)												Y							
Event Task Matrix (SOC_ETM)													Y						
GDMA Controller (GDMA)													Y						

Continued on the next page...

GoBack

Table 7.2-4 – Continued from the previous page...

Peripheral	Source Clock			Derived Clock	Source Clock					Derived Clock									Source Clock
	XTAL_CLK 32 MHz	PLL_CLK			RC_FAST_CLK 8 MHz	RC_SLOW_CLK 130 kHz	OSC_SLOW_CLK 32 kHz	XTAL32K_CLK 32 kHz	PLL_LP_CLK 8 MHz	HP_ROOT_CLK 96 MHz/64 MHz/32 MHz/8 MHz	CRYPTO_CLK	APB_CLK	AHB_CLK	CPU_CLK	LP_DYN_FAST_CLK	LP_DYN_SLOW_CLK	XTAL_D2_CLK 16 MHz	LP_FAST_CLK	Clock from IO
		PLL_F96M_CLK 96 MHz	PLL_F64M_CLK 64 MHz	PLL_F48M_CLK 48 MHz															
UHCI													Y						
Debug Assistant (ASSIST_DEBUG, MEM_MONITOR)														Y					
RISC-V Trace Encoder (TRACE)													Y	Y					
Interrupt priority registers (INTPRI)														Y					

Table 7.2-5. Derived LP Clock Source

Derived Clock	Source Clock			Derived Clock	Source Clock					Derived Clock									Source Clock
	XTAL_CLK 32 MHz	PLL_CLK			RC_FAST_CLK 8 MHz	RC_SLOW_CLK 130 kHz	OSC_SLOW_CLK 32 kHz	XTAL32K_CLK 32 kHz	PLL_LP_CLK 8 MHz	HP_ROOT_CLK 96 MHz/64 MHz/32 MHz/8 MHz	CRYPTO_CLK	APB_CLK	AHB_CLK	CPU_CLK	LP_DYN_FAST_CLK	LP_DYN_SLOW_CLK	XTAL_D2_CLK 16 MHz	LP_FAST_CLK	Clock from IO
		PLL_F96M_CLK 96 MHz	PLL_F64M_CLK 64 MHz	PLL_F48M_CLK 48 MHz															
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 7.2-6. LP Clocks Used by Each Peripheral

Peripheral	Source Clock			Derived Clock	Source Clock					Derived Clock									Source Clock
	XTAL_CLK 32 MHz	PLL_CLK			RC_FAST_CLK 8 MHz	RC_SLOW_CLK 130 kHz	OSC_SLOW_CLK 32 kHz	XTAL32K_CLK 32 kHz	PLL_LP_CLK 8 MHz	HP_ROOT_CLK 96 MHz/64 MHz/32 MHz/8 MHz	CRYPTO_CLK	APB_CLK	AHB_CLK	CPU_CLK	LP_DYN_FAST_CLK	LP_DYN_SLOW_CLK	XTAL_D2_CLK 16 MHz	LP_FAST_CLK	Clock from IO
		PLL_F96M_CLK 96 MHz	PLL_F64M_CLK 64 MHz	PLL_F48M_CLK 48 MHz															
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	

PLL_F96M_CLK

PLL_F96M_CLK is a 96 MHz clock. PLL_F48M_CLK (48 MHz) is divided from PLL_F96M_CLK.

CRYPTO_CLK

As shown in Figure 7.2-3, CRYPTO_CLK can be derived from XTAL_CLK, PLL_96M_CLK, PLL_64M_CLK, or RC_FAST_CLK, and its frequency is up to 96 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. Three 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_F96M_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.

7.3 Programming Procedures

7.3.1 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 to apply the new configuration. To check whether the new configuration takes effect, read PCR_BUS_CLOCK_UPDATE and see if it is 0.

7.3.2 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.

7.3.3 Peripheral Clock Reset and Configuration

Notice:

ESP32-H2 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 7.4 *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.

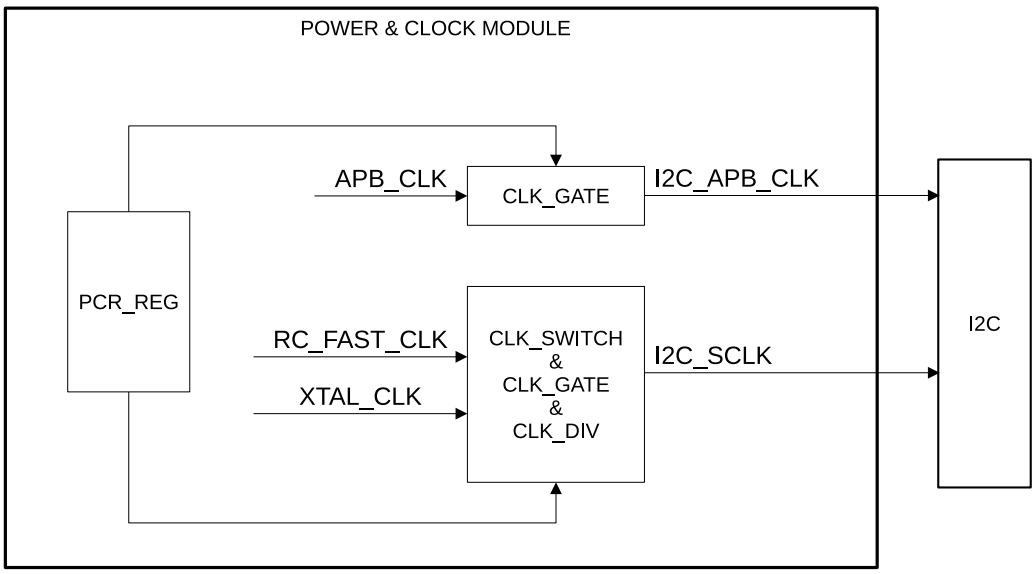


Figure 7.3-1. Clock Configuration Example

Figure 7.3-1 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.

7.4 Register Summary

7.4.1 PCR Register Summary

The addresses in this section are relative to Power/Clock/Reset (PCR) Register base address provided in Table 4.3-2 in Chapter 4 *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 clock 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 clock configuration register	0x0010	R/W
PCR_UART1_PD_CTRL_REG	UART1 power control register	0x0014	R/W
PCR_I2C0_CONF_REG	I2C configuration register	0x0020	varies
PCR_I2C0_SCLK_CONF_REG	I2C clock configuration register	0x0024	R/W
PCR_I2C1_CONF_REG	I2C configuration register	0x0028	varies
PCR_I2C1_SCLK_CONF_REG	I2C clock configuration register	0x002C	R/W
PCR_UHCI_CONF_REG	UHCI configuration register	0x0030	varies
PCR_RMT_CONF_REG	RMT configuration register	0x0034	varies
PCR_RMT_SCLK_CONF_REG	RMT clock configuration register	0x0038	R/W
PCR_LEDC_CONF_REG	LEDC configuration register	0x003C	varies
PCR_LEDC_SCLK_CONF_REG	LEDC clock configuration register	0x0040	R/W
PCR_TIMERGROUP0_CONF_REG	Timer Group 0 configuration register	0x0044	varies
PCR_TIMERGROUP0_TIMER_CLK_CONF_REG	Timer Group 0 general-purpose timer clock configuration register	0x0048	R/W
PCR_TIMERGROUP0_WDT_CLK_CONF_REG	Timer Group 0 WDT clock configuration register	0x004C	R/W
PCR_TIMERGROUP1_CONF_REG	Timer Group1 configuration register	0x0050	varies
PCR_TIMERGROUP1_TIMER_CLK_CONF_REG	Timer Group 1 general-purpose timer clock configuration register	0x0054	R/W
PCR_TIMERGROUP1_WDT_CLK_CONF_REG	Timer Group 1 WDT clock configuration register	0x0058	R/W
PCR_SYSTIMER_CONF_REG	System Timer configuration register	0x005C	varies
PCR_SYSTIMER_FUNC_CLK_CONF_REG	System Timer function clock configuration register	0x0060	R/W
PCR_TWAI0_CONF_REG	TWAI0 configuration register	0x0064	varies
PCR_TWAI0_FUNC_CLK_CONF_REG	TWAI0 functional clock configuration register	0x0068	R/W
PCR_I2S_CONF_REG	I2S configuration register	0x006C	varies
PCR_I2S_TX_CLKM_CONF_REG	I2S TX clock configuration register	0x0070	R/W

Name	Description	Address	Access
PCR_I2S_TX_CLKM_DIV_CONF_REG	I2S TX clock divisor configuration register	0x0074	R/W
PCR_I2S_RX_CLKM_CONF_REG	I2S RX clock configuration register	0x0078	R/W
PCR_I2S_RX_CLKM_DIV_CONF_REG	I2S RX clock divisor configuration register	0x007C	R/W
PCR_SARADC_CONF_REG	SAR ADC configuration register	0x0080	R/W
PCR_SARADC_CLKM_CONF_REG	SAR ADC clock configuration register	0x0084	R/W
PCR_TSENS_CLK_CONF_REG	Temperature sensor clock configuration register	0x0088	R/W
PCR_USB_DEVICE_CONF_REG	USB Serial/JTAG configuration register	0x008C	varies
PCR_INTMTX_CONF_REG	Interrupt Matrix configuration register	0x0090	varies
PCR_PCNT_CONF_REG	PCNT configuration register	0x0094	varies
PCR_ETM_CONF_REG	ETM configuration register	0x0098	varies
PCR_PWM_CONF_REG	MCPWM configuration register	0x009C	varies
PCR_PWM_CLK_CONF_REG	MCPWM clock configuration register	0x00A0	R/W
PCR_PARL_IO_CONF_REG	Parallel IO configuration register	0x00A4	varies
PCR_PARL_CLK_RX_CONF_REG	Parallel IO RX clock configuration register	0x00A8	R/W
PCR_PARL_CLK_TX_CONF_REG	Parallel IO TX clock configuration register	0x00AC	R/W
PCR_GDMA_CONF_REG	GDMA configuration register	0x00B8	R/W
PCR_SPI2_CONF_REG	SPI2 configuration register	0x00BC	varies
PCR_SPI2_CLKM_CONF_REG	SPI2 clock configuration register	0x00C0	R/W
PCR_AES_CONF_REG	AES configuration register	0x00C4	varies
PCR_SHA_CONF_REG	SHA configuration register	0x00C8	varies
PCR_RSA_CONF_REG	RSA configuration register	0x00CC	varies
PCR_RSA_PD_CTRL_REG	RSA power control register	0x00D0	R/W
PCR_ECC_CONF_REG	ECC configuration register	0x00D4	varies
PCR_ECC_PD_CTRL_REG	ECC power control register	0x00D8	R/W
PCR_DS_CONF_REG	RSA_DS configuration register	0x00DC	varies
PCR_HMAC_CONF_REG	HMAC configuration register	0x00E0	varies
PCR_ECDSA_CONF_REG	ECDSA_DS configuration register	0x00E4	varies
PCR_IOMUX_CONF_REG	IO MUX configuration register	0x00E8	R/W
PCR_IOMUX_CLK_CONF_REG	IO MUX clock configuration register	0x00EC	R/W
PCR_MEM_MONITOR_CONF_REG	Memory Access Monitor configuration register	0x00F0	varies
PCR_TRACE_CONF_REG	RISC-V Trace Encoder configuration register	0x00F8	R/W
PCR_ASSIST_CONF_REG	Debug Assistant configuration register	0x00FC	R/W
PCR_CACHE_CONF_REG	Cache configuration register	0x0100	R/W
PCR_TIMEOUT_CONF_REG	Timeout protection configuration register	0x0108	R/W
PCR_SYSCLK_CONF_REG	System clock configuration register	0x010C	varies

Name	Description	Address	Access
PCR_CPU_WAITI_CONF_REG	Configuration register for gating CPU clock in wait for interrupt mode	0x0110	R/W
PCR_CPU_FREQ_CONF_REG	CPU_CLK frequency configuration register	0x0114	R/W
PCR_AHB_FREQ_CONF_REG	AHB_CLK frequency configuration register	0x0118	R/W
PCR_APB_FREQ_CONF_REG	APB_CLK frequency configuration register	0x011C	R/W
PCR_PLL_DIV_CLK_EN_REG	Configuration register for gating PLL divided clocks	0x0124	R/W
PCR_CTRL_TICK_CONF_REG	Clock divisor configuration register	0x012C	R/W
PCR_CTRL_32K_CONF_REG	32 kHz clock configuration register	0x0130	R/W
PCR_SRAM_POWER_CONF_0_REG	HP SRAM/ROM configuration register	0x0134	R/W
PCR_SRAM_POWER_CONF_1_REG	HP SRAM/ROM configuration register	0x0138	R/W
PCR_SEC_CONF_REG	Clock source configuration register for External Memory Encryption and Decryption	0x013C	R/W
PCR_BUS_CLK_UPDATE_REG	Configuration register for applying updated high-performance system clock sources	0x0148	R/W/WTC
PCR_SAR_CLK_DIV_REG	SAR ADC clock divisor configuration register	0x014C	R/W
Frequency Data Register			
PCR_SYSCLK_FREQ_QUERY_0_REG	System clock frequency query 0 register	0x0120	HRO
Version Register			
PCR_DATE_REG	Version control register	0x0FFC	R/W

7.4.2 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 4.3-2 in Chapter 4 *System and Memory*.

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

Name	Description	Address	Access
Configuration Registers			
LP_CLKRST_LP_CLK_CONF_REG	LP 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

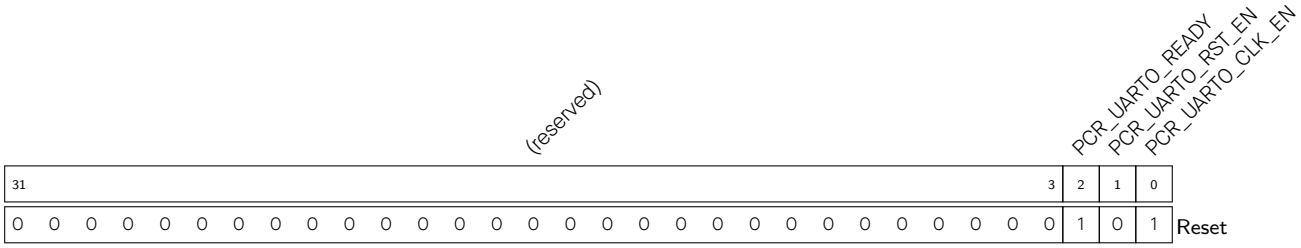
Name	Description	Address	Access
LP_CLKRST_LP_CLK_EN_REG	Configuration register for gating LP clock source	0x0008	R/W
LP_CLKRST_LP_RST_EN_REG	Configuration register for LP peripheral software reset	0x000C	R/W
LP_CLKRST_RESET_CAUSE_REG	Reset cause status register	0x0010	varies
LP_CLKRST_CPU_RESET_REG	CPU 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_LPMEM_FORCE_REG	Configuration register for forcing the gate of LP memory clock	0x0024	R/W
LP_CLKRST_LPPERI_REG	LP peripheral clock configuration register	0x0028	R/W
LP_CLKRST_XTAL32K_REG	XTAL32K_CLK configuration register	0x002C	R/W
LP_CLKRST_DATE_REG	Version control register	0x03FC	R/W
LP_AON_SYS_CFG_REG	Software system reset register	0x0034	varies
LP_AON_CPUCOREO_CFG_REG	Software CPU reset register	0x0038	WT
LPPERI_CLK_EN_REG	eFuse clock gating configuration register	0x0000	R/W
LPPERI_RESET_EN_REG	eFuse reset configuration register	0x0004	R/W

7.5 Registers

7.5.1 PCR Registers

The addresses in this section are relative to the Power/Clock/Reset (PCR) Register base address provided in Table 4.3-2 in Chapter 4 *System and Memory*.

Register 7.1. 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 7.2. PCR_UART0_SCLK_CONF_REG (0x0004)

PCR_UART0_SCLK_DIV_A Configures the denominator of the divisor's fractional part's fractional part for UART0 functional clock. (R/W)

PCR_UART0_SCLK_DIV_B Configures the numerator of the divisor's fractional part for UART0 functional clock. (R/W)

PCR_UART0_SCLK_DIV_NUM Configures the integral part of the divisor for UART0 functional clock. (R/W)

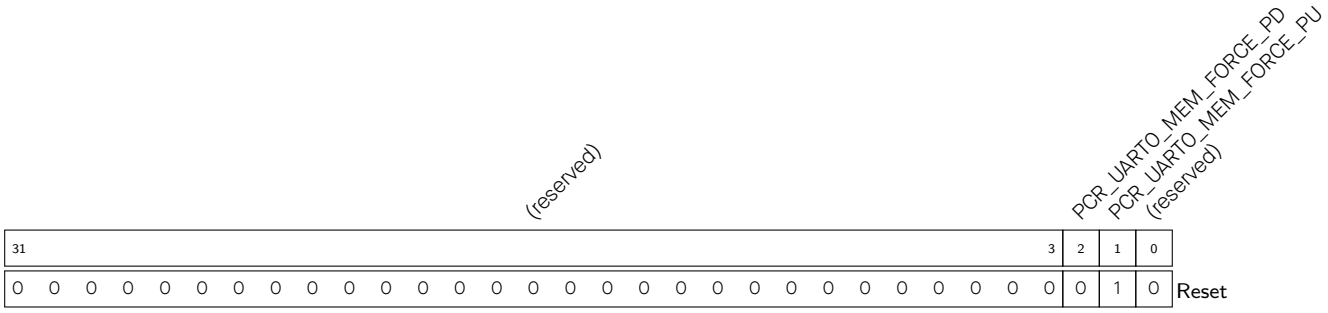
PCR_UART0_SCLK_SEL Configures the clock source of UART0.

0: No clock source
1: PLL_F48M_CLK
2: RC_FAST_CLK
3 (default): XTAL_CLK
(R/W)

PCR_UART0_SCLK_EN Configures whether or not to enable UART0 functional clock.

0: Not enable
1: Enable
(R/W)

Register 7.3. 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)

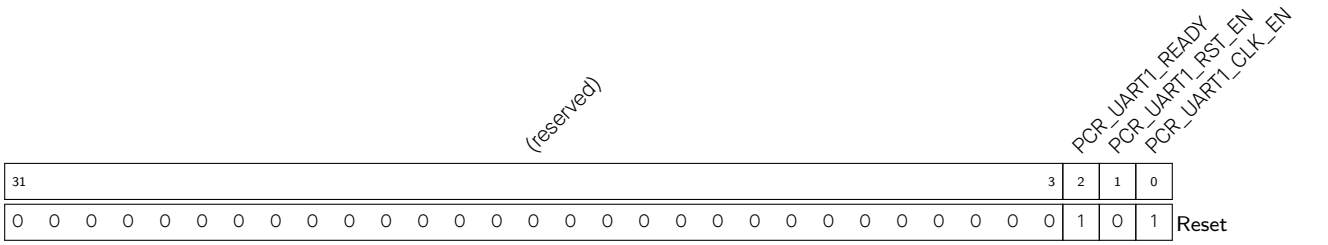
PCR_UART0_MEM_FORCE_PD Configures whether or not to force power down UART0 memory.

0: Not force power down UART0 memory

1: Force power down UART0 memory

(R/W)

Register 7.4. 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 7.5. PCR_UART1_SCLK_CONF_REG (0x0010)

PCR_UART1_SCLK_DIV_A Configures the denominator of the divisor's fractional part for UART1 functional clock. (R/W)

PCR_UART1_SCLK_DIV_B Configures the numerator of the divisor's fractional part for UART1 functional clock. (R/W)

PCR_UART1_SCLK_DIV_NUM Configures the integral part of the divisor for UART1 functional clock. (R/W)

PCR_UART1_SCLK_SEL Configures the clock source of UART1.

0: No clock source

1: PLL_F48M_CLK

2: RC_FAST_CLK

3 (default): XTAL_CLK

(R/W)

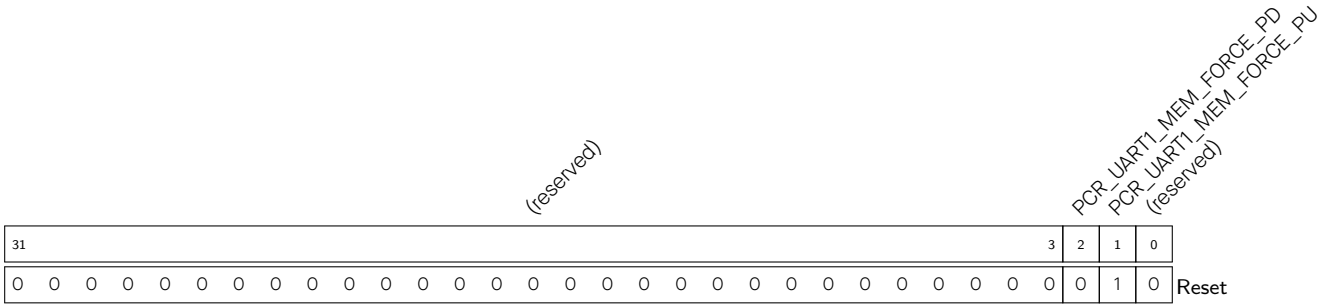
PCR_UART1_SCLK_EN Configures whether or not to enable UART1 functional clock.

0: Not enable

1: Enable

(R/W)

Register 7.6. 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)

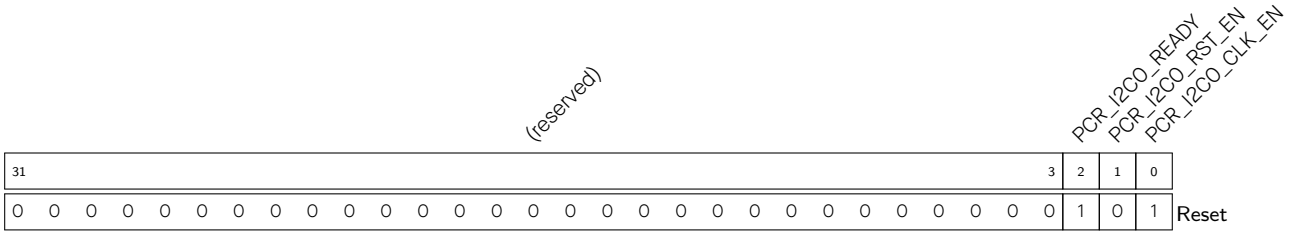
PCR_UART1_MEM_FORCE_PD Configures whether or not to force power down UART1 memory.

0: Not force power down UART1 memory

1: Force power down UART1 memory

(R/W)

Register 7.7. PCR_I2C0_CONF_REG (0x0020)



PCR_I2C0_CLK_EN Configures whether or not to enable APB_CLK for I2C0.

0: Not enable

1: Enable

(R/W)

PCR_I2C0_RST_EN Configures whether or not to reset I2C0.

0: Not reset

1: Reset

(R/W)

PCR_I2C0_READY Represents whether or not I2C0 is released from reset.

0: Not released

1: Released

(RO)

Register 7.8. PCR_I2C0_SCLK_CONF_REG (0x0024)

PCR_I2C0_SCLK_DIV_A Configures the denominator of the divisor's fractional part for I2C0 functional clock. (R/W)

PCR_I2C0_SCLK_DIV_B Configures the numerator of the divisor's fractional part for I2C0 functional clock. (R/W)

PCR_I2C0_SCLK_DIV_NUM Configures the integral part of the divisor for I2C0 functional clock. (R/W)

PCR_I2C0_SCLK_SEL Configures the clock source of I2C0.

0 (default): XTAL_CLK

1: RC_FAST_CLK

(R/W)

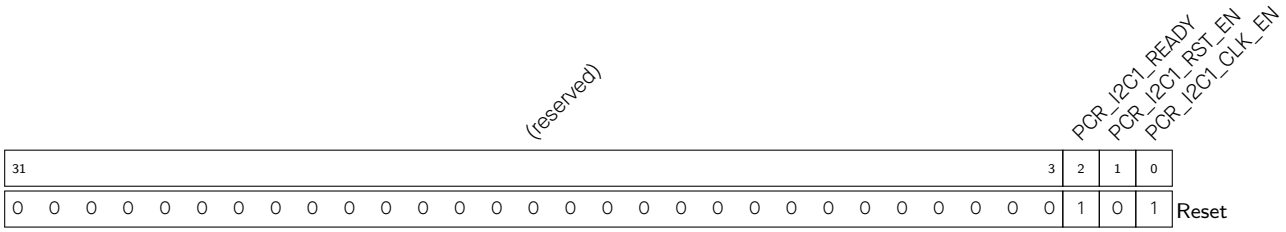
PCR_I2C0_SCLK_EN Configures whether or not to enable I2C0 functional clock.

0: Not enable

1: Enable

(R/W)

Register 7.9. PCR_I2C1_CONF_REG (0x0028)



PCR_I2C1_CLK_EN Configures whether or not to enable APB_CLK for I2C1.

0: Not enable

1: Enable

(R/W)

PCR_I2C1_RST_EN Configures whether or not to reset I2C1.

0: Not reset

1: Reset

(R/W)

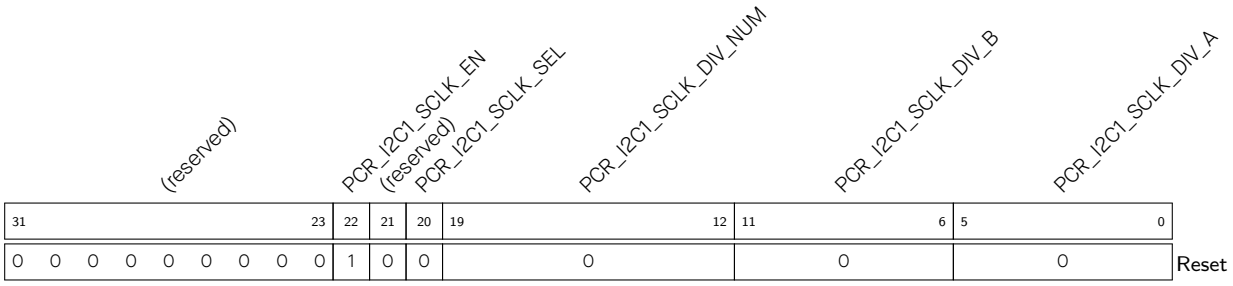
PCR_I2C1_READY Represents whether or not I2C1 is released from reset.

0: Not released

1: Released

(RO)

Register 7.10. PCR_I2C1_SCLK_CONF_REG (0x002C)



PCR_I2C1_SCLK_DIV_A Configures the denominator of the divisor's fractional part for I2C1 functional clock. (R/W)

PCR_I2C1_SCLK_DIV_B Configures the numerator of the divisor's fractional part for I2C1 functional clock. (R/W)

PCR_I2C1_SCLK_DIV_NUM Configures the integral part of the divisor for I2C1 functional clock. (R/W)

PCR_I2C1_SCLK_SEL Configures the clock source of I2C1.

0 (default): XTAL_CLK

1: RC_FAST_CLK

(R/W)

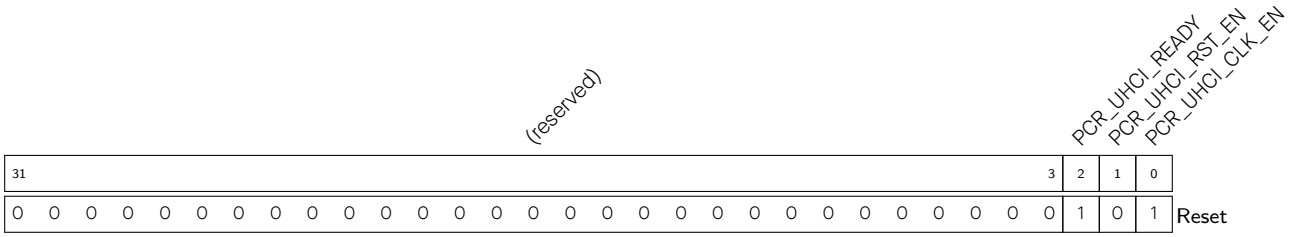
PCR_I2C1_SCLK_EN Configures whether or not to enable I2C1 functional clock.

0: Not enable

1: Enable

(R/W)

Register 7.11. PCR_UHCI_CONF_REG (0x0030)



PCR_UHCI_CLK_EN Configures whether or not to enable UHCI clock.

0: Not enable

1: Enable

(R/W)

PCR_UHCI_RST_EN Configures whether or not to reset UHCI.

0: Not reset

1: Reset

(R/W)

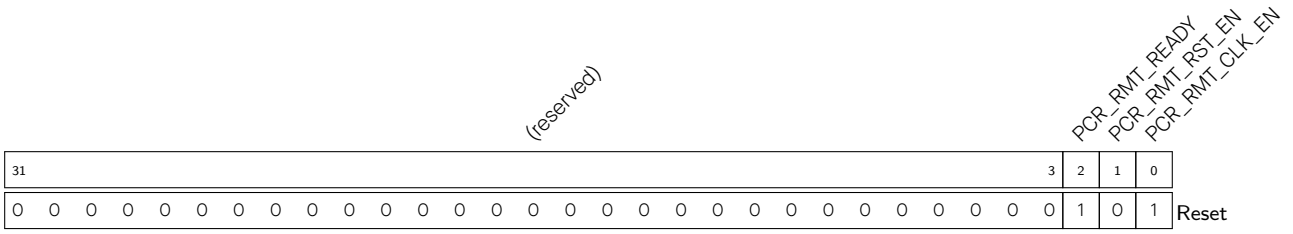
PCR_UHCI_READY Represents whether or not UCHI is released from reset.

0: Not released

1: Released

(RO)

Register 7.12. PCR_RMT_CONF_REG (0x0034)



PCR_RMT_CLK_EN Configures whether or not to enable APB_CLK for RMT.

0: Not enable

1: Enable

(R/W)

PCR_RMT_RST_EN Configures whether or not to reset RMT.

0: Not reset

1: Reset

(R/W)

PCR_RMT_READY Represents whether or not RMT is released from reset.

0: Not released

1: Released

(RO)

Register 7.13. PCR_RMT_SCLK_CONF_REG (0x0038)

PCR_RMT_SCLK_DIV_A Configures the denominator of the divisor's fractional part for RMT functional clock. (R/W)

PCR_RMT_SCLK_DIV_B Configures the numerator of the divisor's fractional part for RMT functional clock. (R/W)

PCR_RMT_SCLK_DIV_NUM Configures the integral part of the divisor for RMT functional clock. (R/W)

PCR_RMT_SCLK_SEL Configures the clock source of RMT.

0: XTAL_CLK

1 (default): RC_FAST_CLK

(R/W)

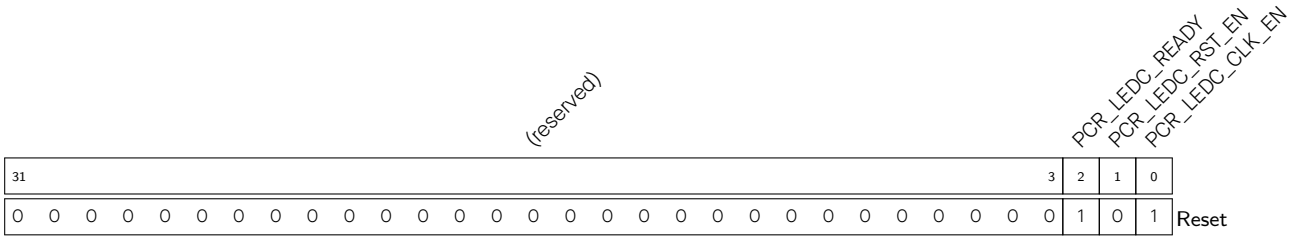
PCR_RMT_SCLK_EN Configures whether or not to enable RMT functional clock.

0: Not enable

1: Enable

(R/W)

Register 7.14. PCR_LEDC_CONF_REG (0x003C)



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)

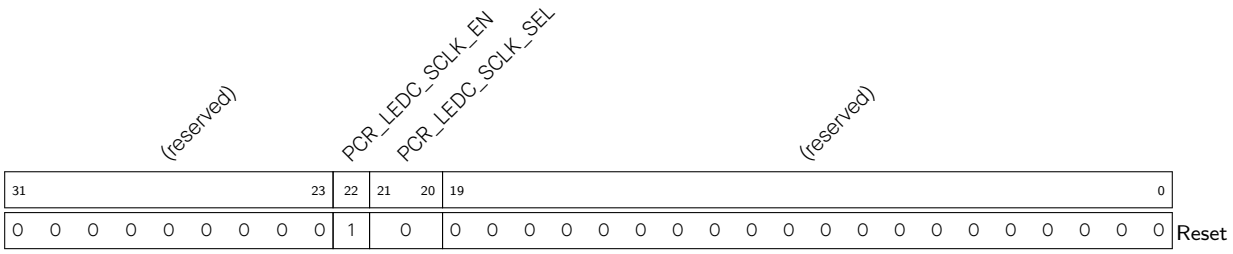
PCR_LEDC_READY Represents whether or not LEDC is released from reset.

0: Not released

1: Released

(RO)

Register 7.15. PCR_LEDC_SCLK_CONF_REG (0x0040)



PCR_LEDC_SCLK_SEL Configures the clock source of LEDC.

0 (default): XTAL_CLK

1: RC_FAST_CLK

2: PLL_F96M_CLK

3: No clock source

(R/W)

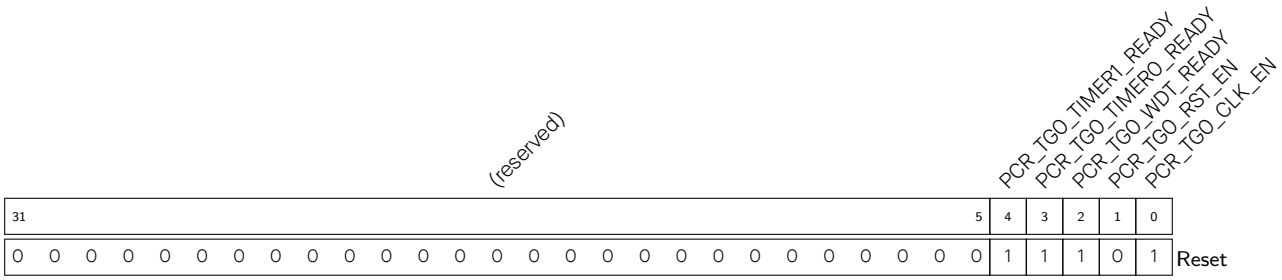
PCR_LEDC_SCLK_EN Configures whether or not to enable LEDC functional clock.

0: Not enable

1: Enable

(R/W)

Register 7.16. PCR_TIMERGROUP0_CONF_REG (0x0044)



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)

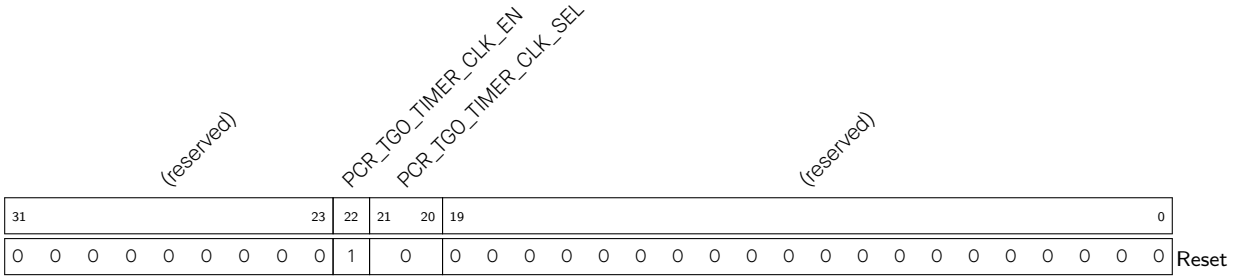
PCR_TG0_TIMER1_READY Represents whether or not Timer 1 in Timer Group 0 is released from reset.

0: Not released

1: Released

(RO)

Register 7.17. PCR_TIMERGROUP0_TIMER_CLK_CONF_REG (0x0048)



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_F48M_CLK

3: No clock source

(R/W)

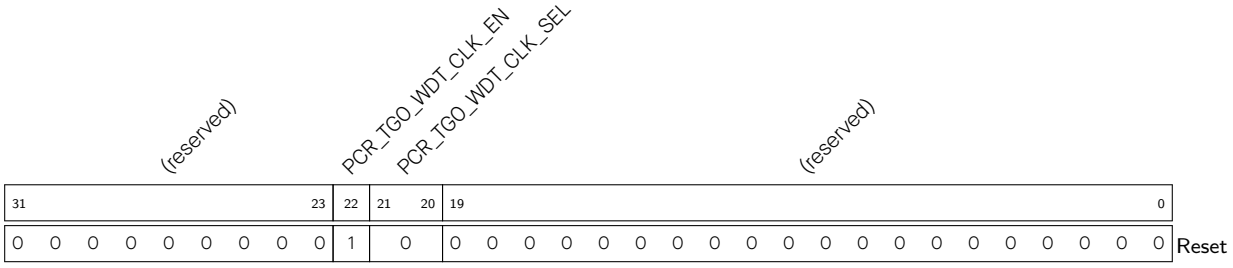
PCR_TG0_TIMER_CLK_EN Configures whether or not to enable the clock of general-purpose timers in Timer Group 0.

0: Not enable

1: Enable

(R/W)

Register 7.18. PCR_TIMERGROUP0_WDT_CLK_CONF_REG (0x004C)



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_F48M_CLK

3: No clock source

(R/W)

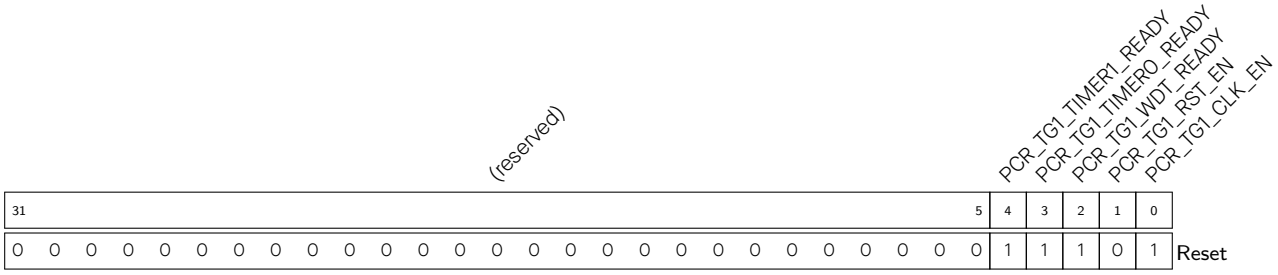
PCR_TG0_WDT_CLK_EN Configures whether or not to enable the clock of WDT in Timer Group 0.

0: Not enable

1: Enable

(R/W)

Register 7.19. PCR_TIMERGROUP1_CONF_REG (0x0050)



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)

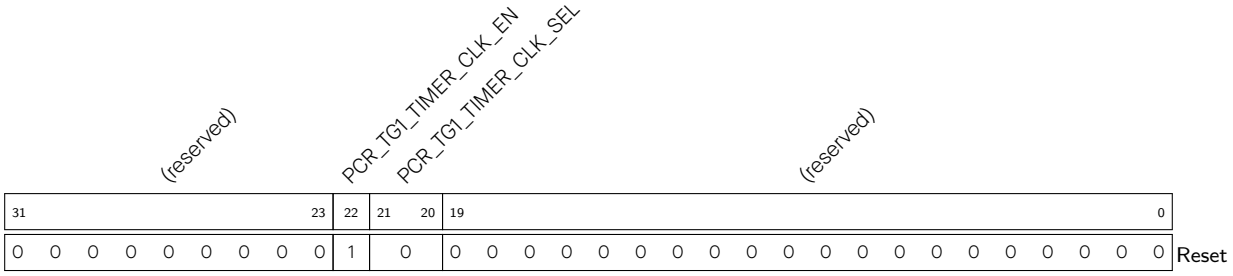
PCR_TG1_TIMER1_READY Represents whether or not Timer 1 in Timer Group 1 is released from reset.

0: Not released

1: Released

(RO)

Register 7.20. PCR_TIMERGROUP1_TIMER_CLK_CONF_REG (0x0054)



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)

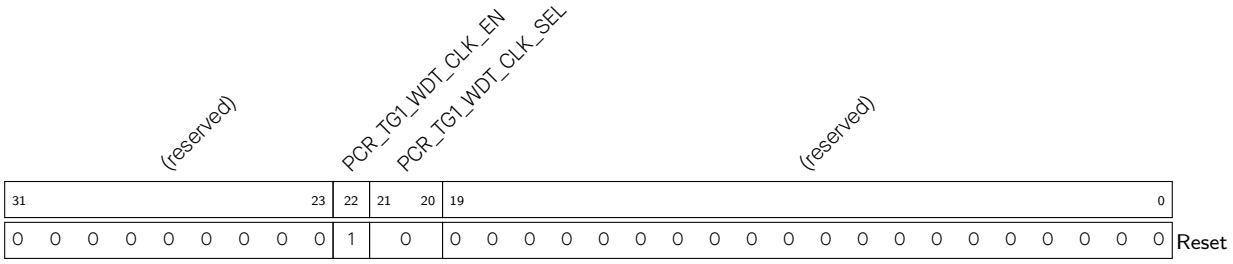
PCR_TG1_TIMER_CLK_EN Configures whether or not to enable the clock of general-purpose timers in Timer Group 1.

0: Not enable

1: Enable

(R/W)

Register 7.21. PCR_TIMERGROUP1_WDT_CLK_CONF_REG (0x0058)



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_F48M_CLK

3: No clock source

(R/W)

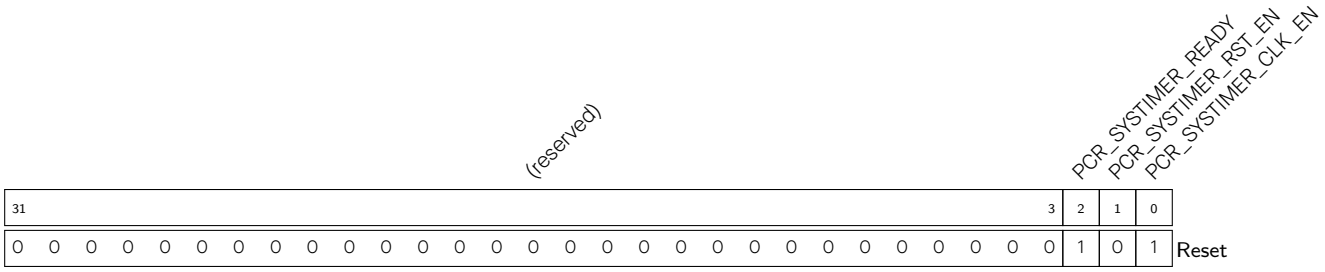
PCR_TG1_WDT_CLK_EN Configures whether or not to enable the clock for WDT in Timer Group 1.

0: Not enable

1: Enable

(R/W)

Register 7.22. PCR_SYSTIMER_CONF_REG (0x005C)



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)

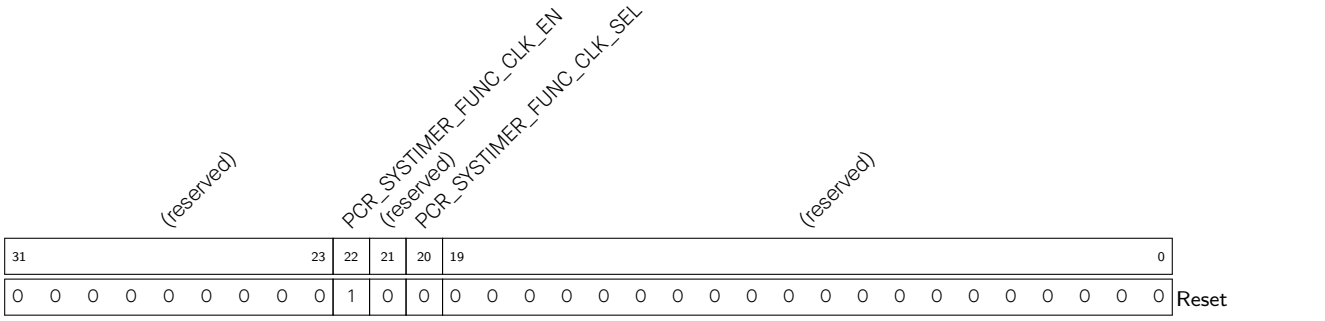
PCR_SYSTIMER_READY Represents whether or not the System Timer is released from reset.

0: Not released

1: Released

(RO)

Register 7.23. PCR_SYSTIMER_FUNC_CLK_CONF_REG (0x0060)



PCR_SYSTIMER_FUNC_CLK_SEL Configures the clock source of System Timer.

0 (default): XTAL_CLK

1: RC_FAST_CLK

(R/W)

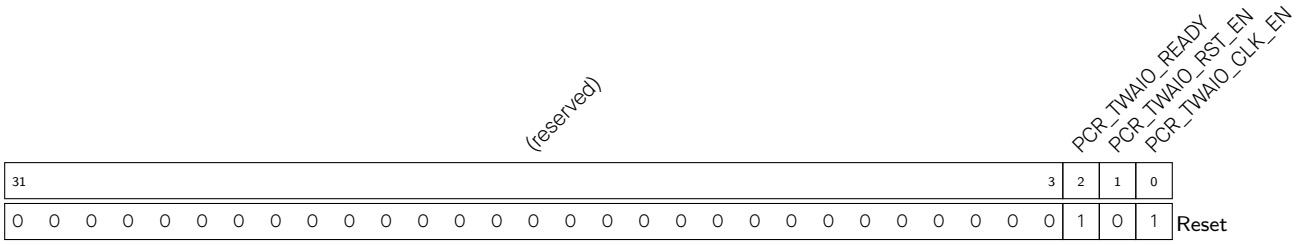
PCR_SYSTIMER_FUNC_CLK_EN Configures whether or not to enable System Timer functional clock.

0: Not enable

1: Enable

(R/W)

Register 7.24. PCR_TWAI0_CONF_REG (0x0064)



PCR_TWAI0_CLK_EN Configures whether or not to enable APB_CLK for TWAI0.

0: Not enable

1: Enable

(R/W)

PCR_TWAI0_RST_EN Configures whether or not to reset TWAI0.

0: Not reset

1: Reset

(R/W)

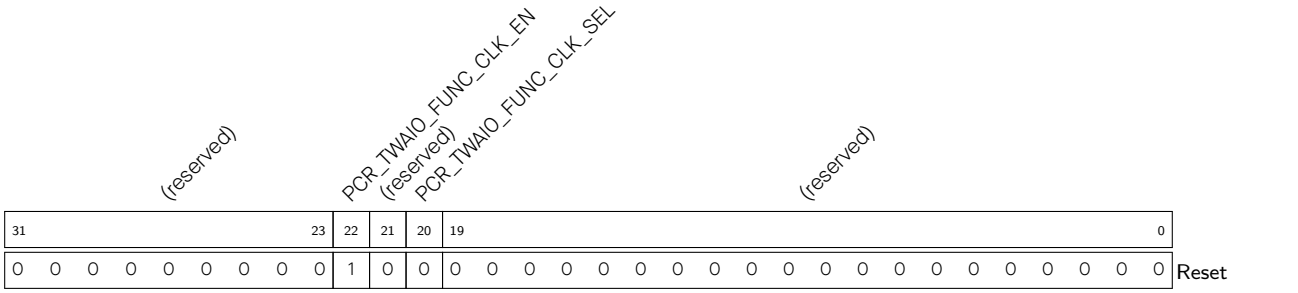
PCR_TWAI0_READY Represents whether or not TWAI0 is released from reset.

0: Not released

1: Released

(RO)

Register 7.25. PCR_TWAI0_FUNC_CLK_CONF_REG (0x0068)



PCR_TWAI0_FUNC_CLK_SEL Configures the clock source of TWAI0.

0 (default): XTAL_CLK

1: RC_FAST_CLK

(R/W)

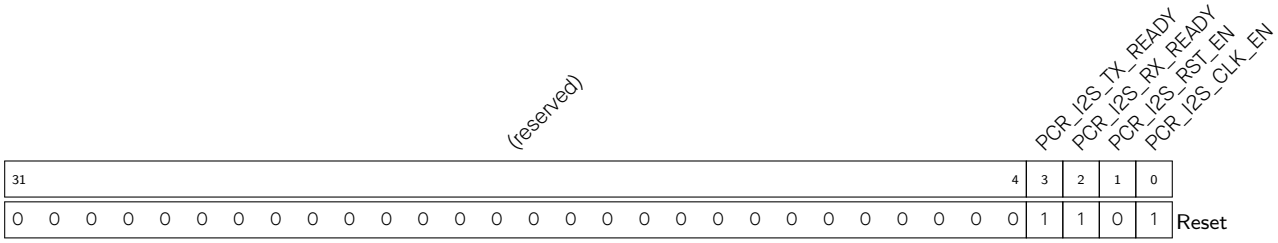
PCR_TWAI0_FUNC_CLK_EN Configures whether or not to enable TWAI0 functional clock.

0: Not enable

1: Enable

(R/W)

Register 7.26. PCR_I2S_CONF_REG (0x006C)



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)

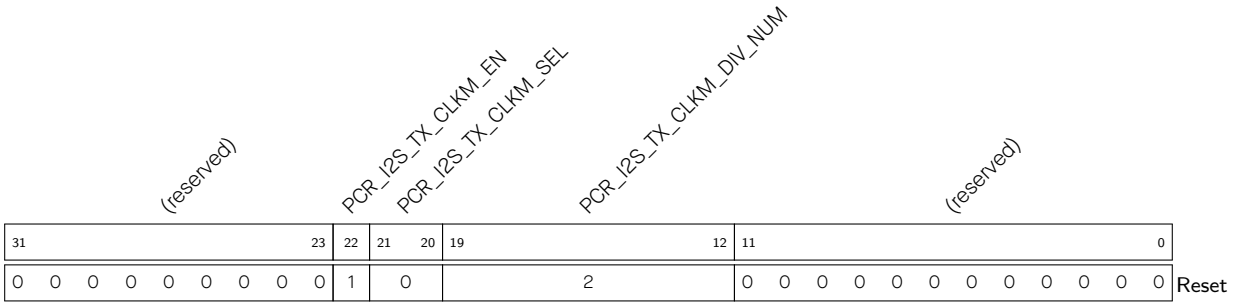
PCR_I2S_TX_READY Represents whether or not I2S TX is released from reset.

0: Not released

1: Released

(RO)

Register 7.27. PCR_I2S_TX_CLKM_CONF_REG (0x0070)



PCR_I2S_TX_CLKM_DIV_NUM Configures the integral part of I2S TX clock divisor.

(R/W)

PCR_I2S_TX_CLKM_SEL Configures the clock source of I2S TX.

0: XTAL_CLK

1: PLL_F96M_CLK

2: PLL_F64M_CLK

3: I2S_MCLK_in

(R/W)

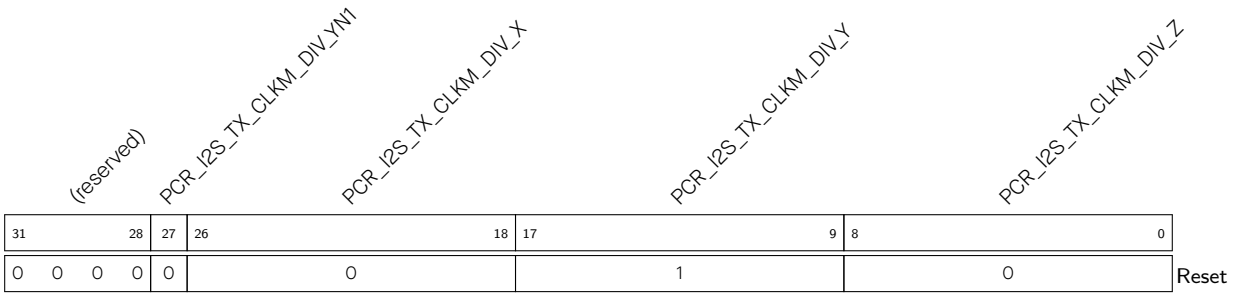
PCR_I2S_TX_CLKM_EN Configures whether or not to enable I2S TX functional clock.

0: Not enable

1: Enable

(R/W)

Register 7.28. PCR_I2S_TX_CLKM_DIV_CONF_REG (0x0074)



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)

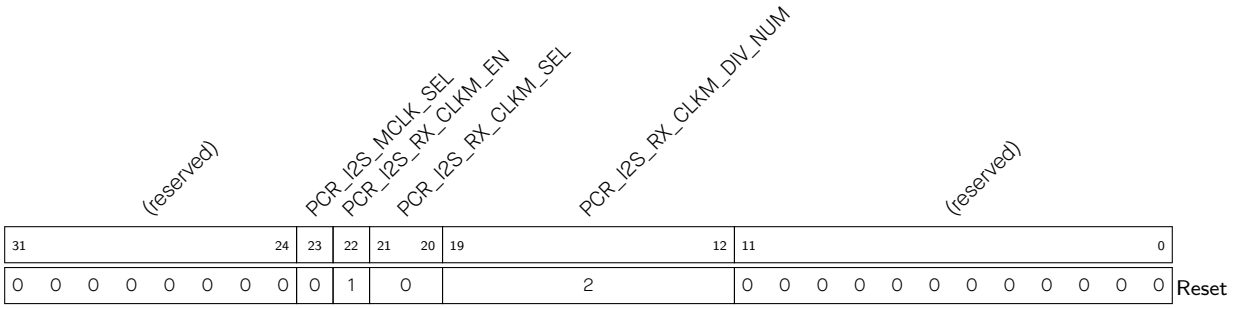
PCR_I2S_TX_CLKM_DIV_Y For b <= a/2, the value of I2S_TX_CLKM_DIV_Y is (a%b). For b > a/2, the value of I2S_TX_CLKM_DIV_Y is (a%(a - b)). (R/W)

PCR_I2S_TX_CLKM_DIV_X For b <= a/2, the value of I2S_TX_CLKM_DIV_X is floor(a/b) - 1. For b > a/2, the value of I2S_TX_CLKM_DIV_X is floor(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)

Note:
"a" and "b" represent the denominator and the numerator of the fractional divisor, respectively. For more information, see Section 31.6 in Chapter *I2S Controller (I2S)*.

Register 7.29. PCR_I2S_RX_CLKM_CONF_REG (0x0078)



PCR_I2S_RX_CLKM_DIV_NUM Configures the integral divisor for I2S clock. (R/W)

PCR_I2S_RX_CLKM_SEL Configures the clock source of I2S RX.

0: XTAL_CLK

1: PLL_F96M_CLK

2: PLL_F64M_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)

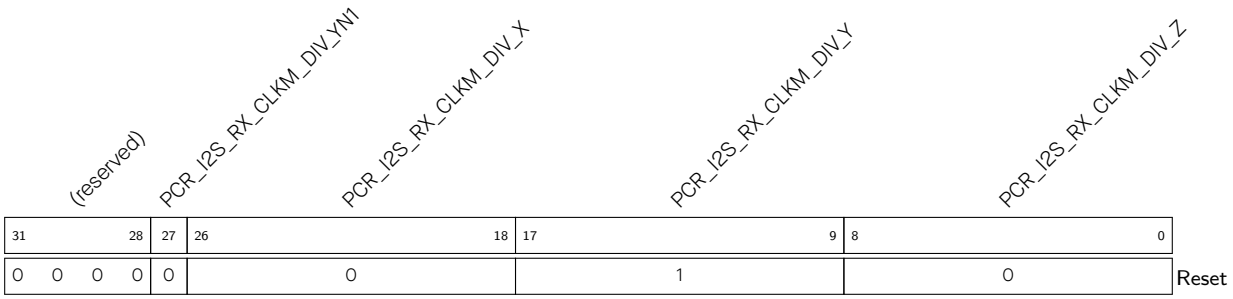
PCR_I2S_MCLK_SEL Configures to select master clock.

0 (default): I2S_RX_CLK

1: I2S_TX_CLK

(R/W)

Register 7.30. PCR_I2S_RX_CLKM_DIV_CONF_REG (0x007C)



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)

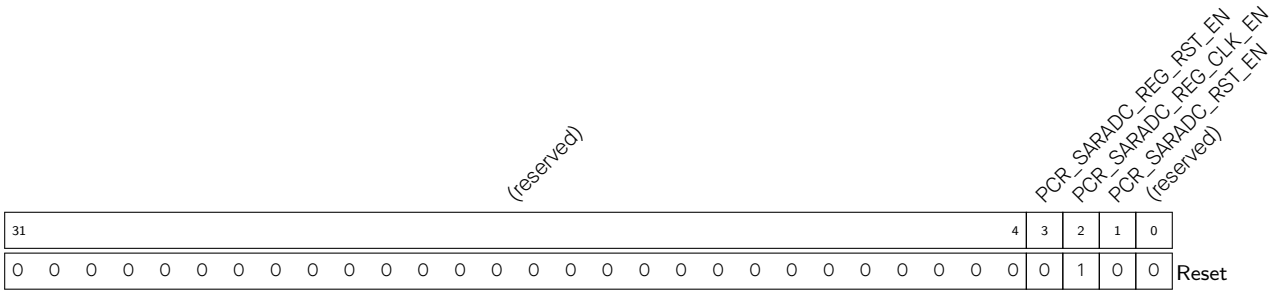
PCR_I2S_RX_CLKM_DIV_Y For b <= a/2, the value of I2S_RX_CLKM_DIV_Y is (a%b). For b > a/2, the value of I2S_RX_CLKM_DIV_Y is (a%(a - b)). (R/W)

PCR_I2S_RX_CLKM_DIV_X For b <= a/2, the value of I2S_RX_CLKM_DIV_X is floor(a/b) - 1. For b > a/2, the value of I2S_RX_CLKM_DIV_X is floor(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)

Note:
"a" and "b" represent the denominator and the numerator of the fractional divisor, respectively. For more information, see Section 31.6.

Register 7.31. PCR_SARADC_CONF_REG (0x0080)



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)

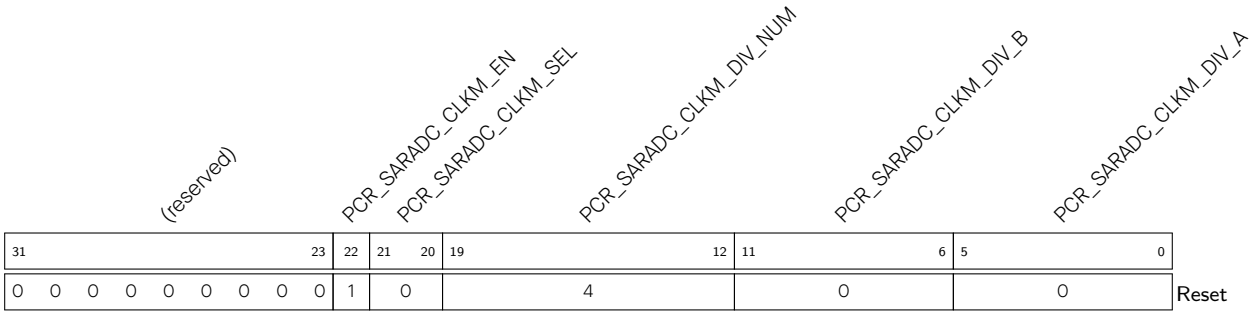
PCR_SARADC_REG_RST_EN Configures whether or not to reset APB registers of SAR ADC.

0: Not reset

1: Reset

(R/W)

Register 7.32. PCR_SARADC_CLKM_CONF_REG (0x0084)



PCR_SARADC_CLKM_DIV_A Configures the denominator of the divisor's fractional part for SAR ADC functional clock. (R/W)

PCR_SARADC_CLKM_DIV_B Configures the numerator of the divisor's fractional part for SAR ADC functional clock. (R/W)

PCR_SARADC_CLKM_DIV_NUM Configures the integral part of the divisor for SAR ADC functional clock. (R/W)

PCR_SARADC_CLKM_SEL Configures the clock source of SAR ADC.

0 (default): XTAL_CLK

1: PLL_F96M_CLK

2: RC_FAST_CLK

3: No clock source

(R/W)

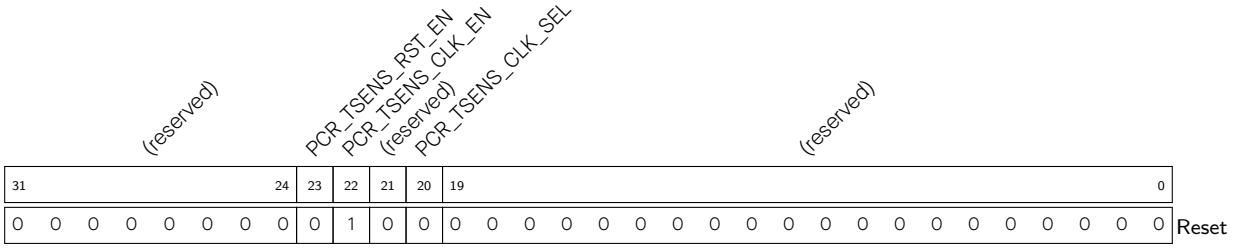
PCR_SARADC_CLKM_EN Configures whether or not to enable SAR ADC functional clock.

0: Not enable

1: Enable

(R/W)

Register 7.33. PCR_TSENS_CLK_CONF_REG (0x0088)



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)

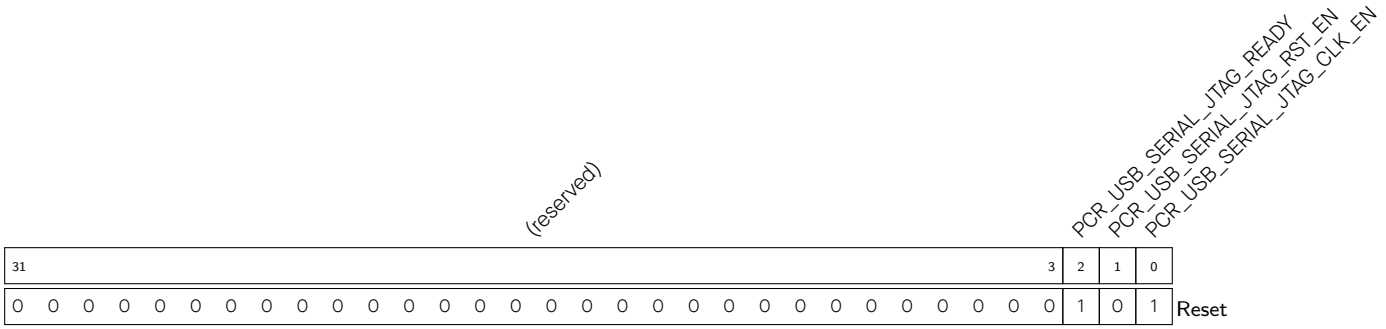
PCR_TSENS_RST_EN Configures whether or not to reset the temperature sensor.

0: Not reset

1: Reset

(R/W)

Register 7.34. PCR_USB_SERIAL_JTAG_CONF_REG (0x008C)



PCR_USB_SERIAL_JTAG_CLK_EN Configures whether or not to enable USB Serial/JTAG clock.

0: Not enable

1: Enable

(R/W)

PCR_USB_SERIAL_JTAG_RST_EN Configures whether or not to reset USB Serial/JTAG.

0: Not reset

1: Reset

(R/W)

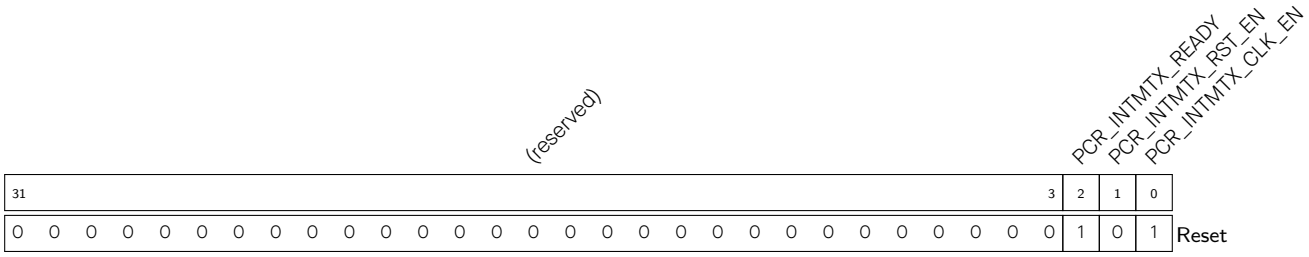
PCR_USB_SERIAL_JTAG_READY Represents whether or not USB Serial/JTAG is released from reset.

0: Not released

1: Released

(RO)

Register 7.35. PCR_INTMTX_CONF_REG (0x0090)



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)

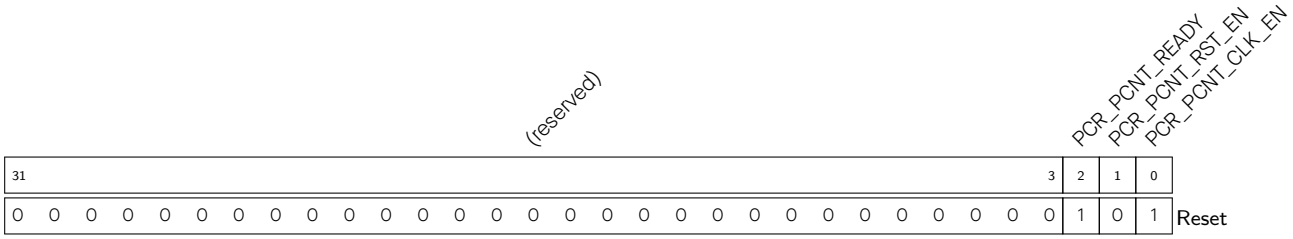
PCR_INTMTX_READY Represents whether or not Interrupt Matrix is released from reset.

0: Not released

1: Released

(RO)

Register 7.36. PCR_PCNT_CONF_REG (0x0094)



PCR_PCNT_CLK_EN Configures whether or not to enable PCNT clock.

0: Not enable

1: Enable

(R/W)

PCR_PCNT_RST_EN Configures whether or not to reset PCNT.

0: Not reset

1: Reset

(R/W)

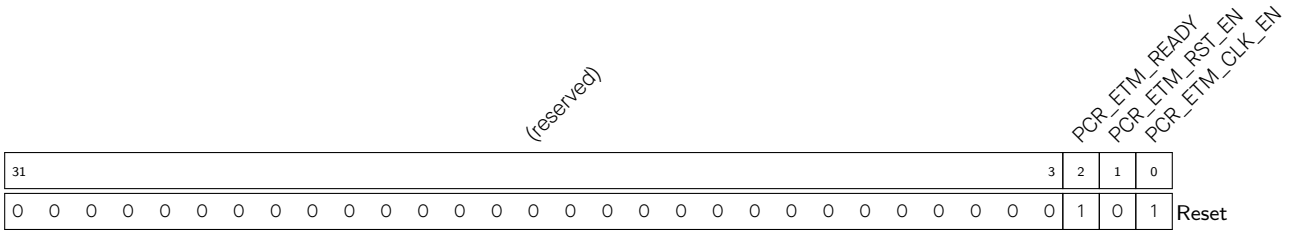
PCR_PCNT_READY Represents whether or not PCNT is released from reset.

0: Not released

1: Released

(RO)

Register 7.37. PCR_ETM_CONF_REG (0x0098)



PCR_ETM_CLK_EN Configures 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)

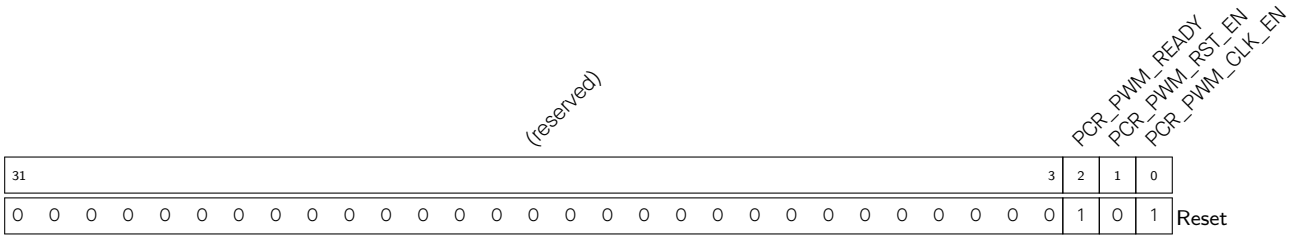
PCR_ETM_READY Represents whether or not ETM is released from reset.

0: Not released

1: Released

(RO)

Register 7.38. PCR_PWM_CONF_REG (0x009C)



PCR_PWM_CLK_EN Configures whether or not to enable MCPWM bus clock.

0: Not enable

1: Enable

(R/W)

PCR_PWM_RST_EN Configures whether or not to reset MCPWM.

0: Not reset

1: Reset

(R/W)

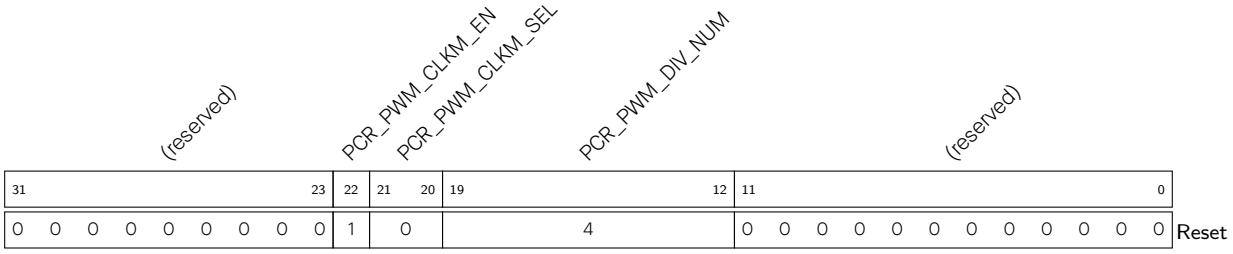
PCR_PWM_READY Represents whether or not MCPWM is released from reset.

0: Not released

1: Released

(RO)

Register 7.39. PCR_PWM_CLK_CONF_REG (0x00A0)



PCR_PWM_DIV_NUM Configures the integral part of the divisor for MCPWM functional clock. (R/W)

PCR_PWM_CLKM_SEL Configures the clock source of MCPWM.

0 (default): XTAL_CLK

1: RC_FAST_CLK

2: PLL_F96M_CLK

3: No clock source

(R/W)

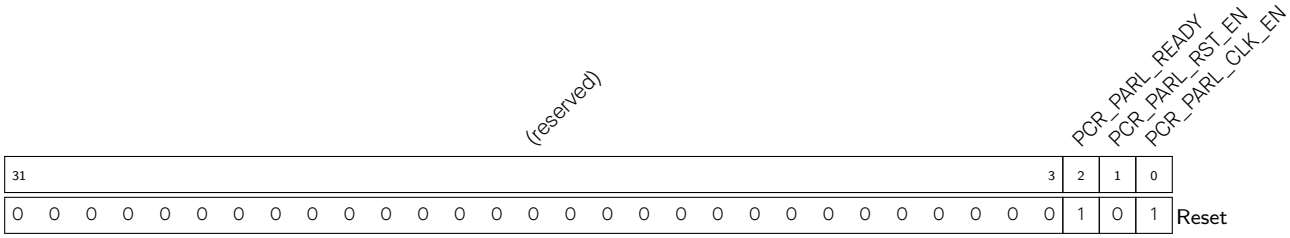
PCR_PWM_CLKM_EN Configures whether or not to enable MCPWM functional clock.

0: Not enable

1: Enable

(R/W)

Register 7.40. PCR_PARL_IO_CONF_REG (0x00A4)



PCR_PARL_CLK_EN Configures whether or not to enable APB_CLK for Parallel IO.

0: Not enable

1: Enable

(R/W)

PCR_PARL_RST_EN Configures whether or not to reset Parallel IO APB registers.

0: Not reset

1: Reset

(R/W)

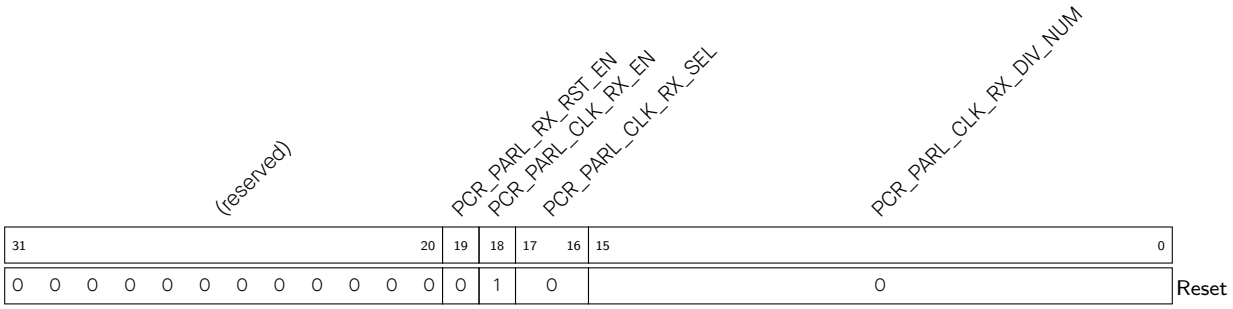
PCR_PARL_READY Represents whether or not Parallel IO is released from reset.

0: Not released

1: Released

(RO)

Register 7.41. PCR_PARL_CLK_RX_CONF_REG (0x00A8)



PCR_PARL_CLK_RX_DIV_NUM Configures the integral divisor for Parallel IO RX clock. (R/W)

PCR_PARL_CLK_RX_SEL Configures the clock source of Parallel IO RX.

0 (default): XTAL

1: PLL_F96M_CLK

2: RC_FAST_CLK

3: Use the clock from chip pin

(R/W)

PCR_PARL_CLK_RX_EN Configures whether or not to enable Parallel IO RX clock.

0: Not enable

1: Enable

(R/W)

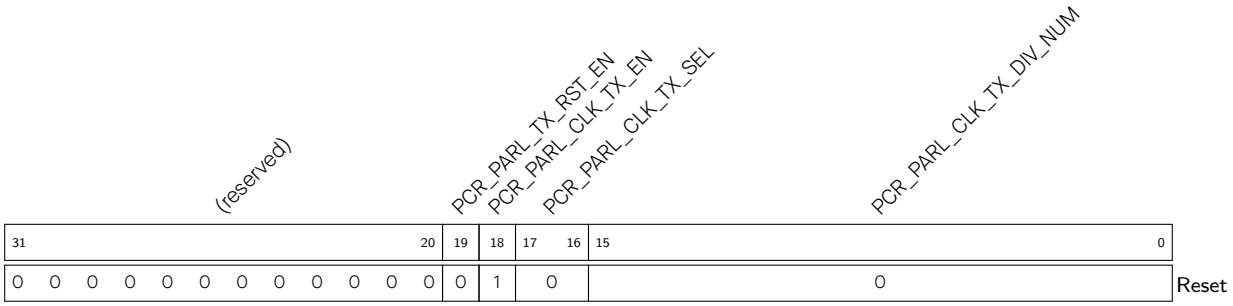
PCR_PARL_RX_RST_EN Configures whether or not to reset Parallel IO RX.

0: Not reset

1: Reset

(R/W)

Register 7.42. PCR_PARL_CLK_TX_CONF_REG (0x00AC)



PCR_PARL_CLK_TX_DIV_NUM Configures the integral divisor for Parallel IO TX clock. (R/W)

PCR_PARL_CLK_TX_SEL Configures the clock source of Parallel IO TX.

0 (default): XTAL

1: PLL_F96M_CLK

2: RC_FAST_CLK

3: Use the clock from chip pin

(R/W)

PCR_PARL_CLK_TX_EN Configures whether or not to enable Parallel IO TX clock.

0: Not enable

1: Enable

(R/W)

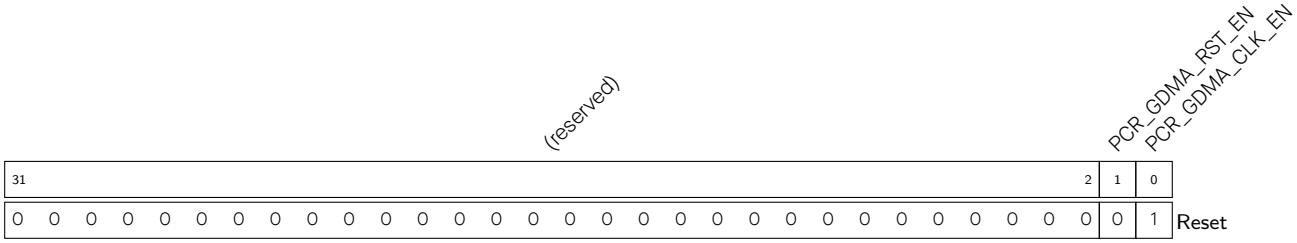
PCR_PARL_TX_RST_EN Configures whether or not to reset Parallel IO TX.

0: Not reset

1: Reset

(R/W)

Register 7.43. PCR_GDMA_CONF_REG (0x00B8)



PCR_GDMA_CLK_EN Configures whether or not to enable GDMA clock.

0: Not enable

1: Enable

(R/W)

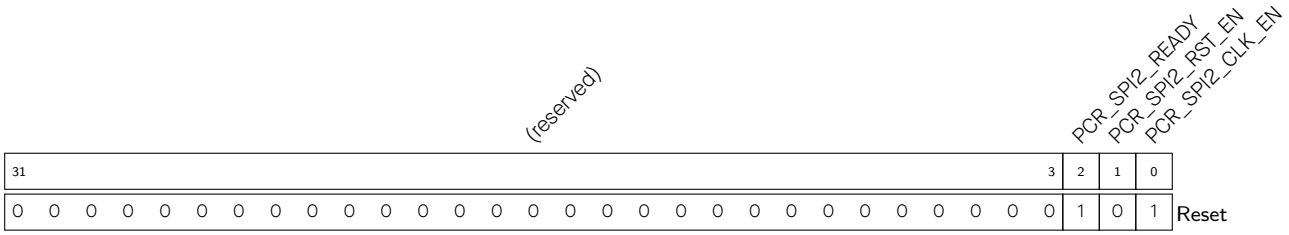
PCR_GDMA_RST_EN Configures whether or not to reset GDMA.

0: Not reset

1: Reset

(R/W)

Register 7.44. PCR_SPI2_CONF_REG (0x00BC)



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)

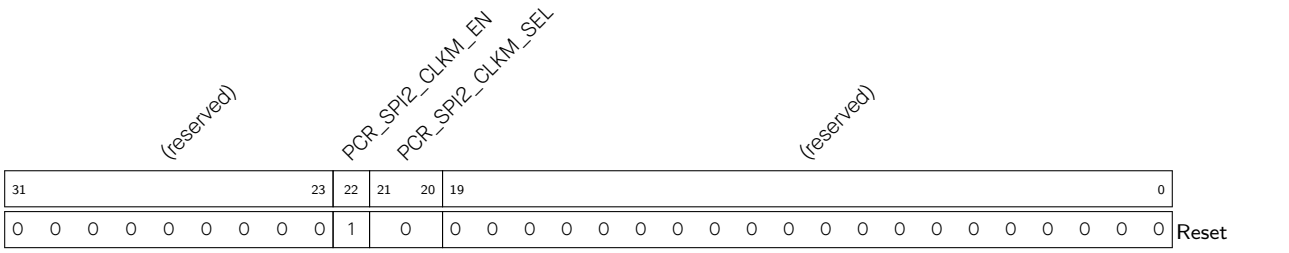
PCR_SPI2_READY Represents whether or not SPI2 is released from reset.

0: Not released

1: Released

(RO)

Register 7.45. PCR_SPI2_CLKM_CONF_REG (0x00C0)



PCR_SPI2_CLKM_SEL Configures the clock source of SPI2.

0 (default): XTAL_CLK

1: PLL_F48M_CLK

2: RC_FAST_CLK

3: No clock source

(R/W)

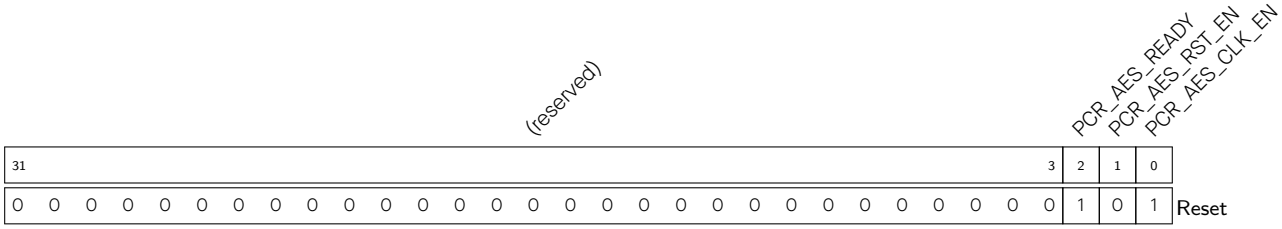
PCR_SPI2_CLKM_EN Configures whether or not to enable SPI2 functional clock.

0: Not enable

1: Enable

(R/W)

Register 7.46. PCR_AES_CONF_REG (0x00C4)



PCR_AES_CLK_EN Configures whether or not to enable AES clock.

0: Not enable

1: Enable

(R/W)

PCR_AES_RST_EN Configures whether or not to reset AES.

0: Not reset

1: Reset

(R/W)

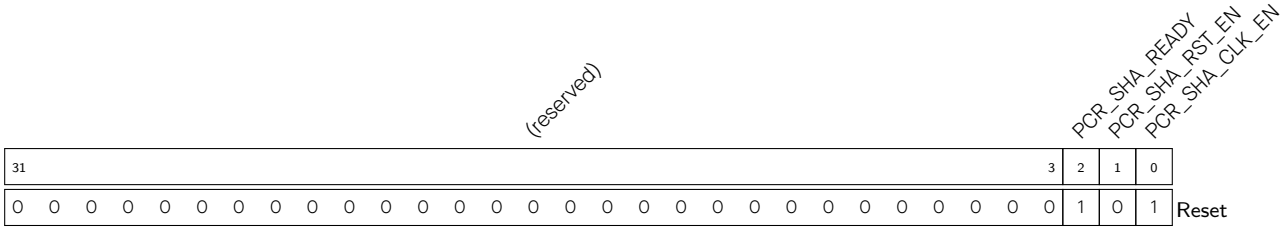
PCR_AES_READY Represents whether or not AES is released from reset.

0: Not released

1: Released

(RO)

Register 7.47. PCR_SHA_CONF_REG (0x00C8)



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)

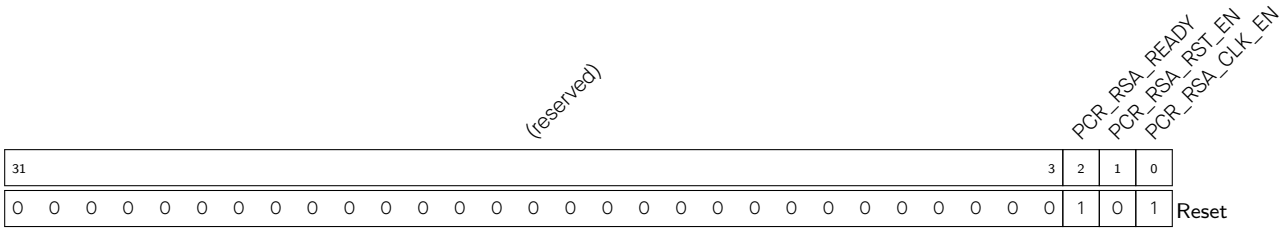
PCR_SHA_READY Represents whether or not SHA is released from reset.

0: Not released

1: Released

(RO)

Register 7.48. PCR_RSA_CONF_REG (0x00CC)



PCR_RSA_CLK_EN Configures whether or not to enable RSA clock.

0: Not enable

1: Enable

(R/W)

PCR_RSA_RST_EN Configures whether or not to reset RSA.

0: Not reset

1: Reset

(R/W)

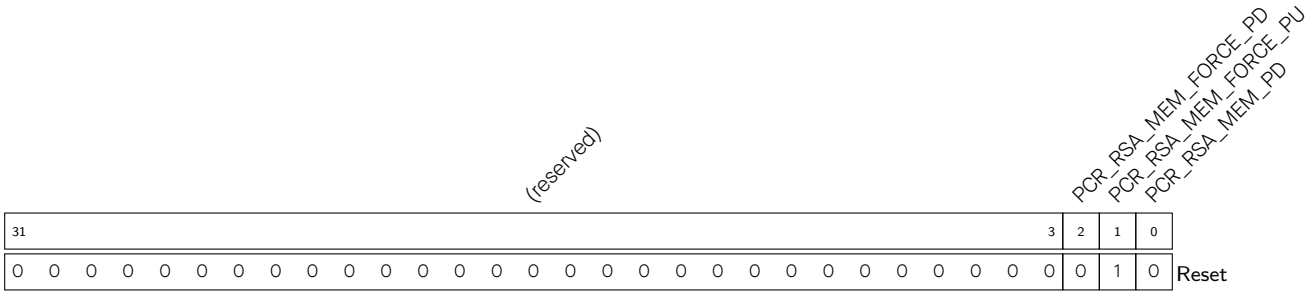
PCR_RSA_READY Represents whether or not RSA is released from reset.

0: Not released

1: Released

(RO)

Register 7.49. PCR_RSA_PD_CTRL_REG (0x00D0)



PCR_RSA_MEM_PD Configures whether or not to power down RSA internal memory.

0: Not power down

1: Power down

(R/W)

PCR_RSA_MEM_FORCE_PU Configures whether or not to force power up RSA internal memory.

0: Not force power up

1: Force power up

(R/W)

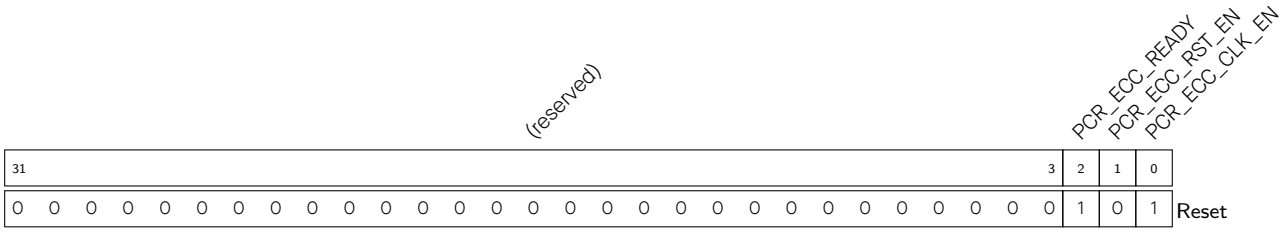
PCR_RSA_MEM_FORCE_PD Configures whether or not to force power down RSA internal memory.

0: Not force power down

1: Force power down

(R/W)

Register 7.50. PCR_ECC_CONF_REG (0x00D4)



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)

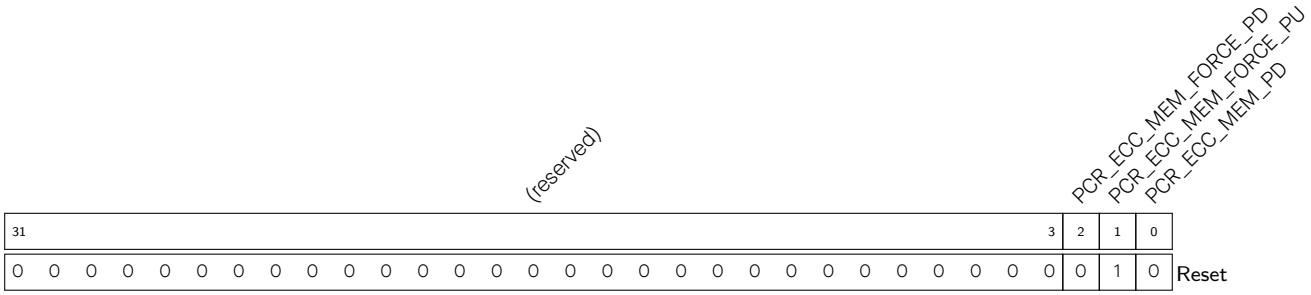
PCR_ECC_READY Represents whether or not ECC is released from reset.

0: Not released

1: Released

(RO)

Register 7.51. PCR_ECC_PD_CTRL_REG (0x00D8)



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)

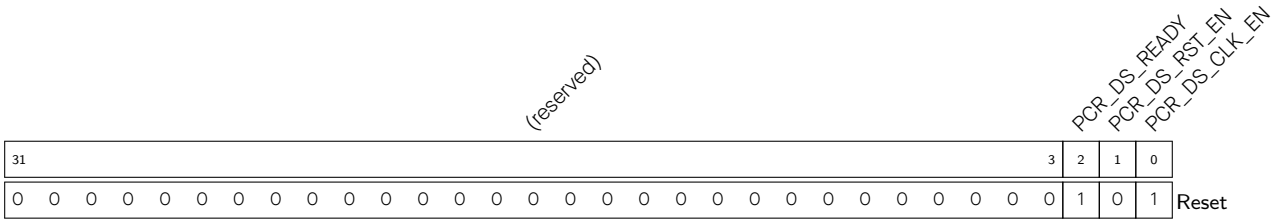
PCR_ECC_MEM_FORCE_PD Configures whether or not to force power down ECC internal memory.

0: Not force power down

1: Force power down

(R/W)

Register 7.52. PCR_DS_CONF_REG (0x00DC)



PCR_DS_CLK_EN Configures whether or not to enable the RSA_DS clock.

0: Not enable

1: Enable

(R/W)

PCR_DS_RST_EN Configures whether or not to reset DS.

0: Not reset

1: Reset

(R/W)

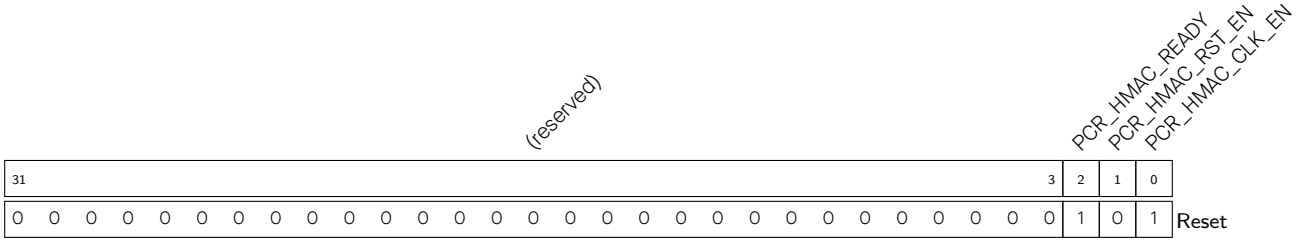
PCR_DS_READY Represents whether or not the RSA_DS peripheral is released from reset.

0: Not released

1: Released

(RO)

Register 7.53. PCR_HMAC_CONF_REG (0x00E0)



PCR_HMAC_CLK_EN Configures whether or not to enable HMAC clock.

0: Not enable

1: Enable

(R/W)

PCR_HMAC_RST_EN Configures whether or not to reset HMAC.

0: Not reset

1: Reset

(R/W)

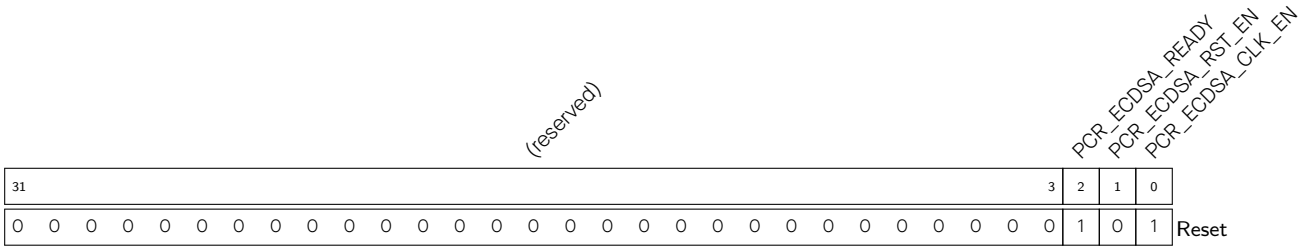
PCR_HMAC_READY Represents whether or not HMAC is released from reset.

0: Not released

1: Released

(RO)

Register 7.54. PCR_ECDSA_CONF_REG (0x00E4)



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)

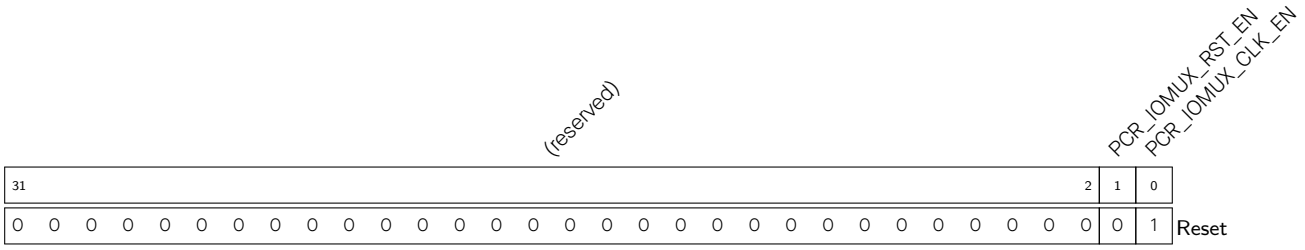
PCR_ECDSA_READY Represents whether or not ECDSA_DS is released from reset.

0: Not released

1: Released

(RO)

Register 7.55. PCR_IOMUX_CONF_REG (0x00E8)



PCR_IOMUX_CLK_EN Configures whether or not to enable APB_CLK for IO MUX.

0: Not enable

1: Enable

(R/W)

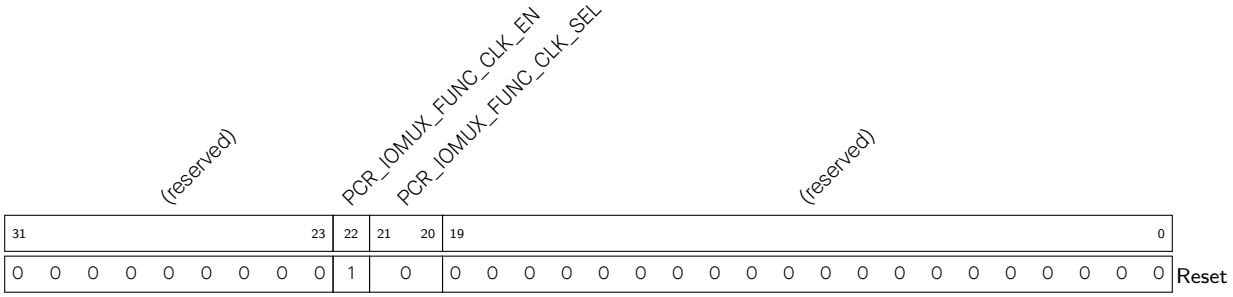
PCR_IOMUX_RST_EN Configures whether or not to reset IO MUX.

0: Not reset

1: Reset

(R/W)

Register 7.56. PCR_IOMUX_CLK_CONF_REG (0x00EC)



PCR_IOMUX_FUNC_CLK_SEL Configures the clock source of IO MUX.

0: XTAL_CLK

1: RC_FAST_CLK

2: PLL_F48M_CLK

3: No clock source

(R/W)

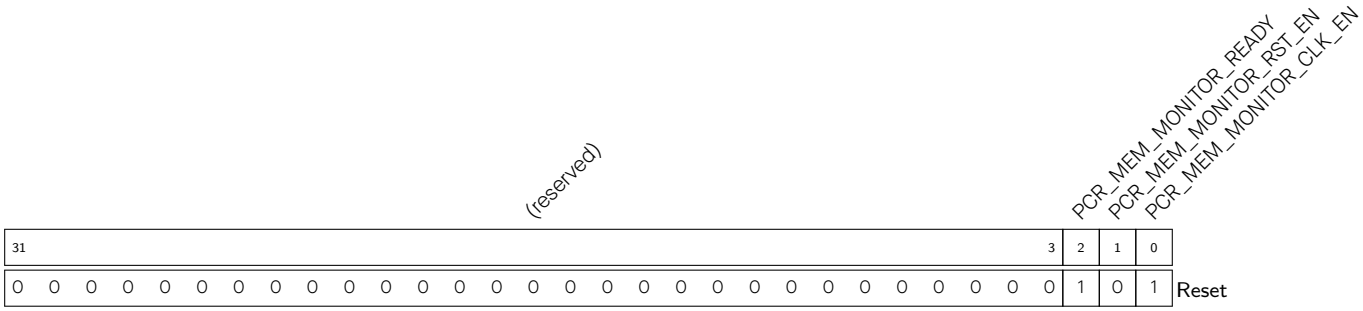
PCR_IOMUX_FUNC_CLK_EN Configures whether or not to enable IO MUX functional clock.

0: Not enable

1: Enable

(R/W)

Register 7.57. PCR_MEM_MONITOR_CONF_REG (0x00F0)



PCR_MEM_MONITOR_CLK_EN Configures whether or not to enable the clock of Memory Access Monitor.

0: Not enable

1: Enable

(R/W)

PCR_MEM_MONITOR_RST_EN Configures whether or not to reset Memory Access Monitor.

0: Not reset

1: Reset

(R/W)

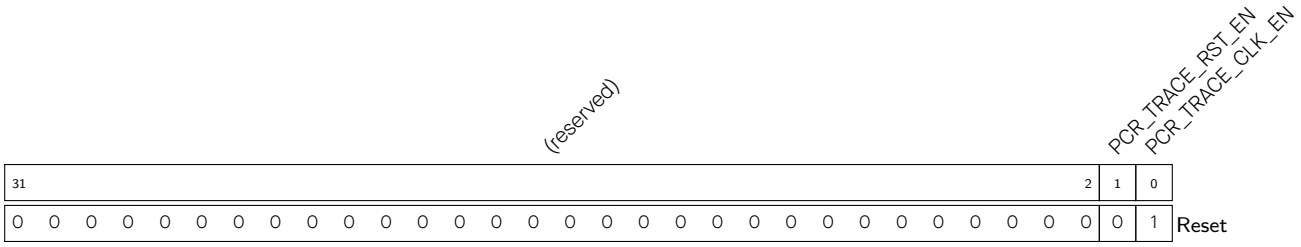
PCR_MEM_MONITOR_READY Represents whether or not Memory Access Monitor is released from reset.

0: Not released

1: Released

(RO)

Register 7.58. PCR_TRACE_CONF_REG (0x00F8)



PCR_TRACE_CLK_EN Configures whether or not to enable the clock of RISC-V Trace Encoder.

0: Not enable

1: Enable

(R/W)

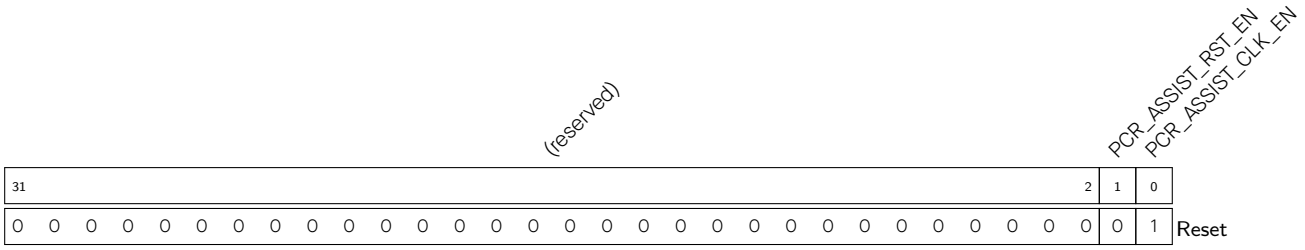
PCR_TRACE_RST_EN Configures whether or not to reset RISC-V Trace Encoder.

0: Not reset

1: Reset

(R/W)

Register 7.59. PCR_ASSIST_CONF_REG (0x00FC)



PCR_ASSIST_CLK_EN Configures whether or not to enable Debug Assistant clock.

0: Not enable

1: Enable

(R/W)

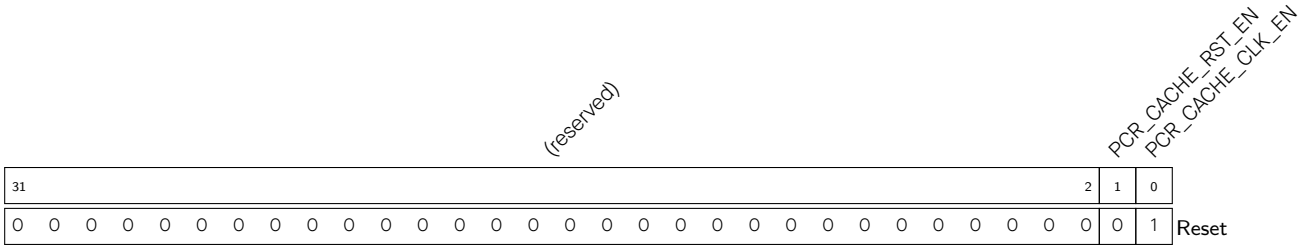
PCR_ASSIST_RST_EN Configures whether or not to reset Debug Assistant.

0: Not reset

1: Reset

(R/W)

Register 7.60. PCR_CACHE_CONF_REG (0x0100)



PCR_CACHE_CLK_EN Configures whether or not to enable Cache clock.

0: Not enable

1: Enable

(R/W)

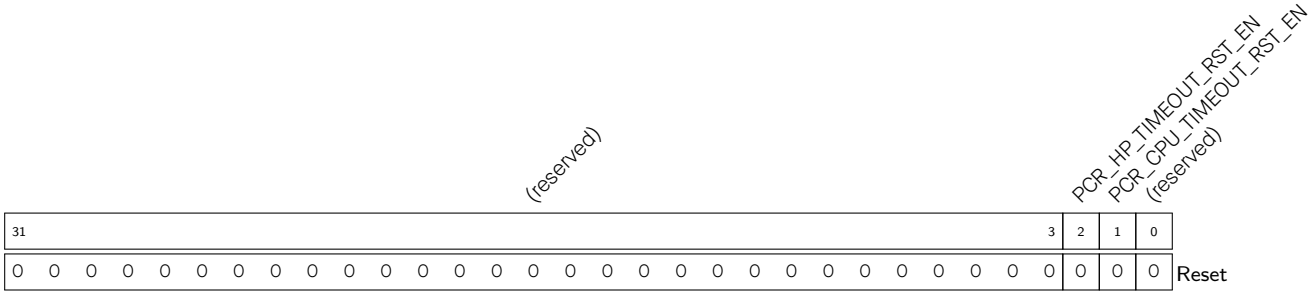
PCR_CACHE_RST_EN Configures whether or not to reset Cache.

0: Not reset

1: Reset

(R/W)

Register 7.61. PCR_TIMEOUT_CONF_REG (0x0108)



PCR_CPU_TIMEOUT_RST_EN Configures whether or not to reset CPU Peripheral Timeout Protection.

0: Not reset

1: Reset

(R/W)

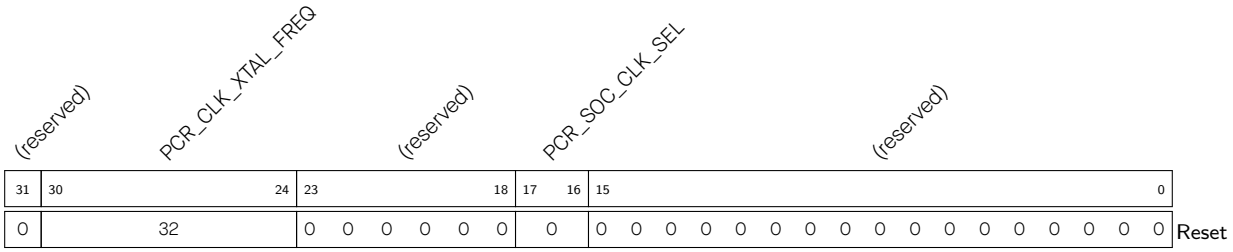
PCR_HP_TIMEOUT_RST_EN Configures whether or not to reset HP Peripheral Timeout Protection.

0: Not reset

1: Reset

(R/W)

Register 7.62. PCR_SYSCLK_CONF_REG (0x010C)



PCR_SOC_CLK_SEL Configures to select clock source of HP_ROOT_CLK.

0: XTAL_CLK

1: PLL_F96M_CLK

2: RC_FAST_CLK

3: PLL_F64M_CLK

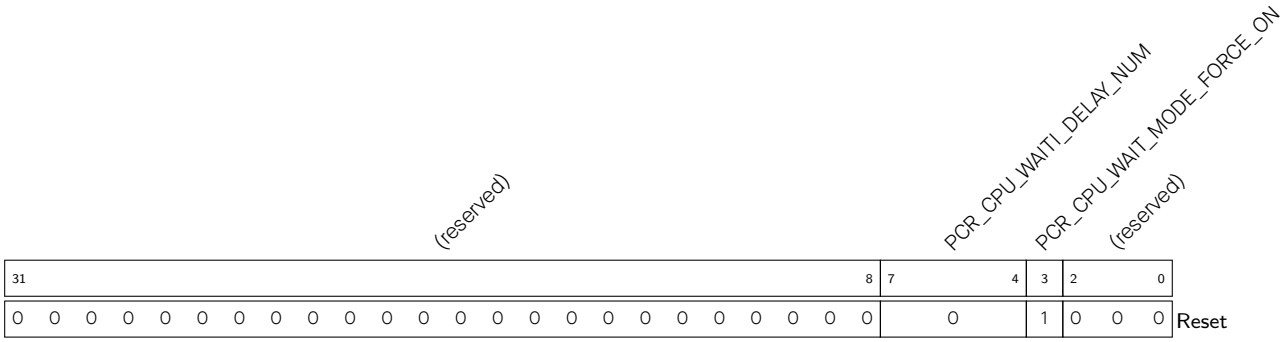
(R/W)

PCR_CLK_XTAL_FREQ Represents the frequency of XTAL.

Measurement unit: MHz.

(RO)

Register 7.63. PCR_CPU_WAITI_CONF_REG (0x0110)



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

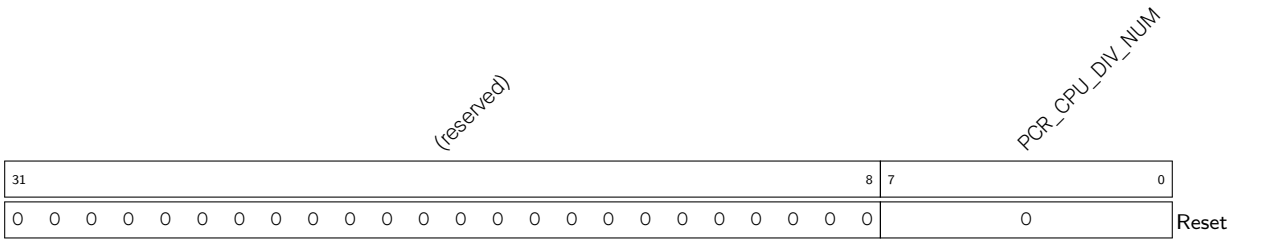
Usually, after executing the WFI instruction, CPU enters the WFI mode, during which the gated CPU clock is turned off until any interrupts occur. In this way, power consumption is saved. If this bit is set, the gated CPU clock is always on and will not be turned off by the WFI instruction. (R/W)

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 instruction.

Measurement unit: CPU_CLK cycles.

(R/W)

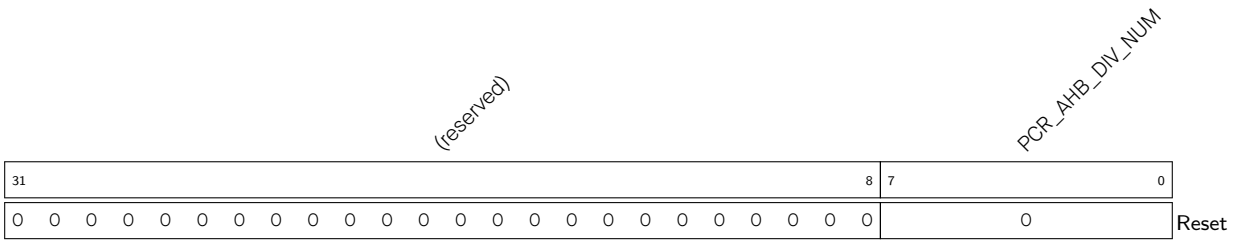
Register 7.64. PCR_CPU_FREQ_CONF_REG (0x0114)



PCR_CPU_DIV_NUM Configures the divisor of HP_ROOT_CLK to generate CPU_CLK.

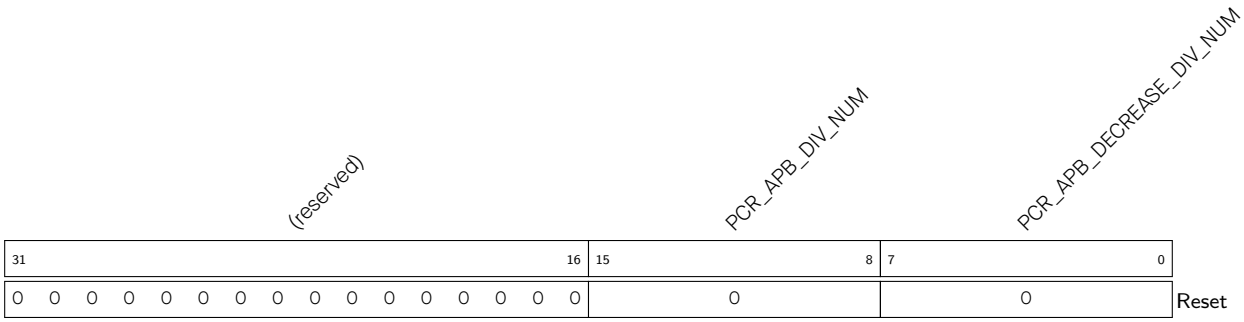
This field should be used together with PCR_AHB_DIV_NUM.

Register 7.65. PCR_AHB_FREQ_CONF_REG (0x0118)



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)

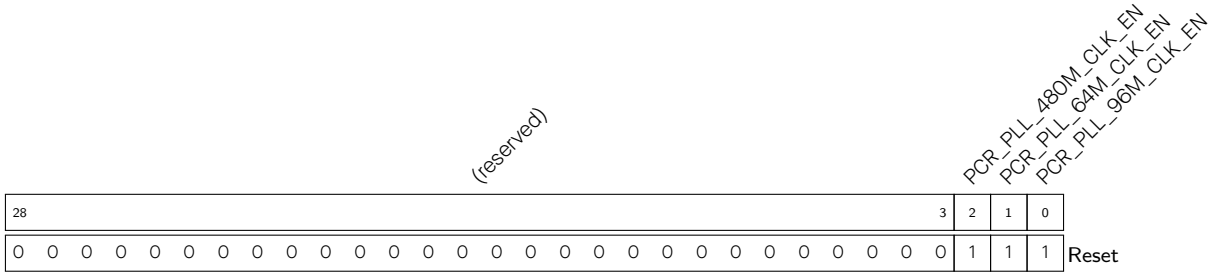
Register 7.66. PCR_APB_FREQ_CONF_REG (0x011C)



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 7.67. PCR_PLL_DIV_CLK_EN_REG (0x0124)



PCR_PLL_96M_CLK_EN Configures whether or not to enable PLL_F96M_CLK.

0: Not enable

1 (default): Enable

(R/W)

PCR_PLL_64M_CLK_EN Configures whether or not to enable PLL_F64M_CLK.

0: Not enable

1 (default): Enable

(R/W)

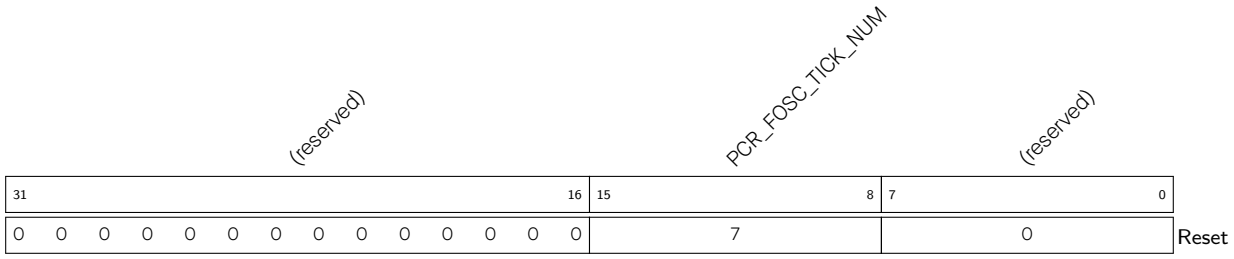
PCR_PLL_48M_CLK_EN Configures whether or not to enable 48 MHz clock derived from PLL_F96M_CLK divided by 2.

0: Not enable

1 (default): Enable

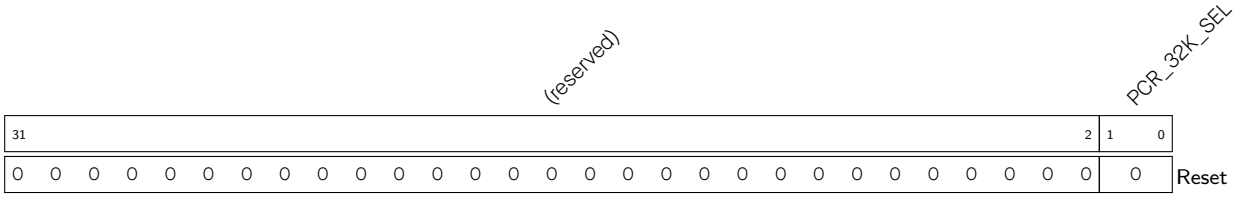
(R/W)

Register 7.68. PCR_CTRL_TICK_CONF_REG (0x012C)



PCR_FOSC_TICK_NUM Configures the clock divisor for RC_FAST_CLK before it enters the calibration module. (R/W)

Register 7.69. PCR_CTRL_32K_CONF_REG (0x0130)



PCR_32K_SEL Configures the 32 kHz clock for TIMER_GROUP.

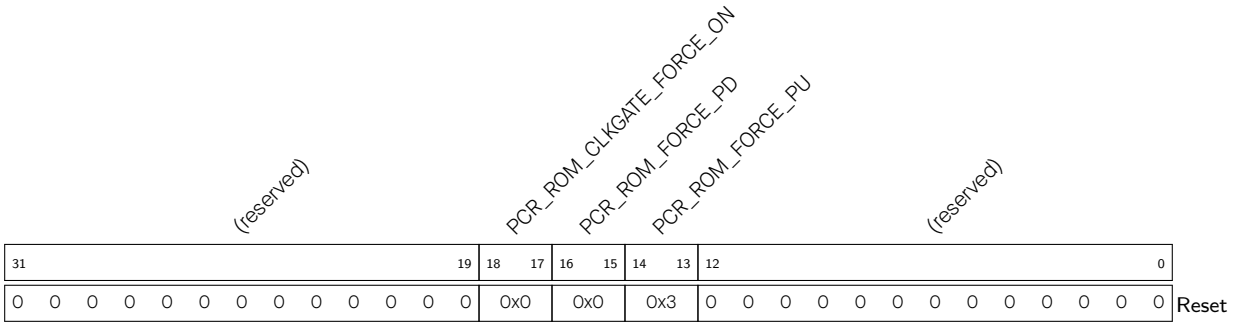
0: Invalid. No effect

1: XTAL32K_CLK

2/3: OSC_SLOW_CLK

(R/W)

Register 7.70. PCR_SRAM_POWER_CONF_0_REG (0x0134)



PCR_ROM_FORCE_PU Configures whether or not to force power up ROM.

0: Not force power up

1: Force power up

(R/W)

PCR_ROM_FORCE_PD Configures whether or not to force power down ROM.

0: Not force power down

1: Force power down

(R/W)

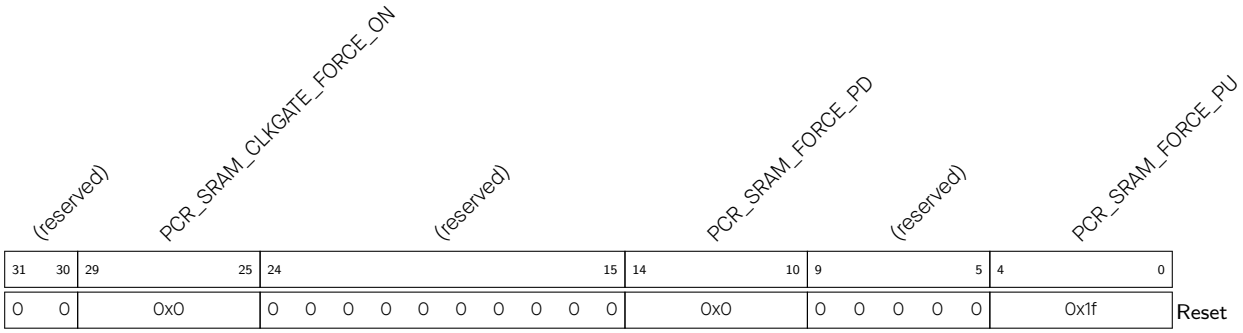
PCR_ROM_CLKGATE_FORCE_ON Configures whether to force enable clocks and bypass clock gating when accessing ROM.

0: Use clock gating

1: Force enable clocks and bypass clock gating

(R/W)

Register 7.71. PCR_SRAM_POWER_CONF_1_REG (0x0138)



PCR_SRAM_FORCE_PU Configures whether or not to force power up SRAM.

0: Not force power up

1: Force power up

(R/W)

PCR_SRAM_FORCE_PD Configures whether or not to force power down SRAM.

0: Not force power down

1: Force power down

(R/W)

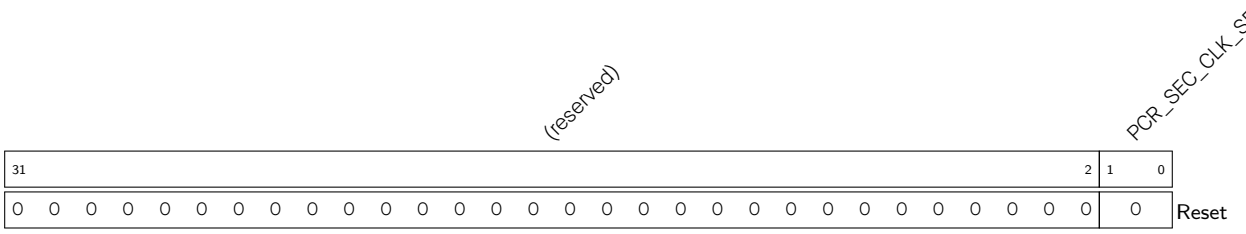
PCR_SRAM_CLKGATE_FORCE_ON Configures whether to force enable clocks and bypass clock gating when accessing SRAM.

0: Use clock gating

1: Force enable clocks and bypass clock gating

(R/W)

Register 7.72. PCR_SEC_CONF_REG (0x013C)



PCR_SEC_CLK_SEL Configures the clock source for the External Memory Encryption and Decryption module.

0 (default): XTAL_CLK

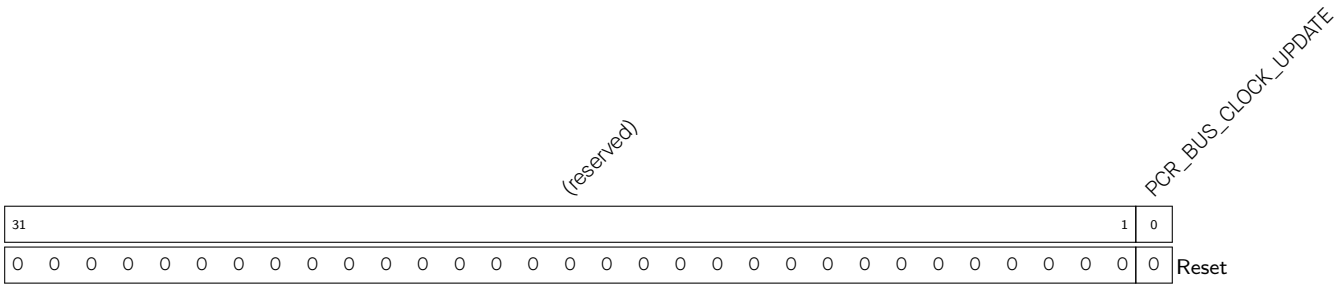
1: RC_FAST_CLK

2: PLL_F64M_CLK

3: PLL_F96M_CLK

(R/W)

Register 7.73. PCR_BUS_CLK_UPDATE_REG (0x0148)



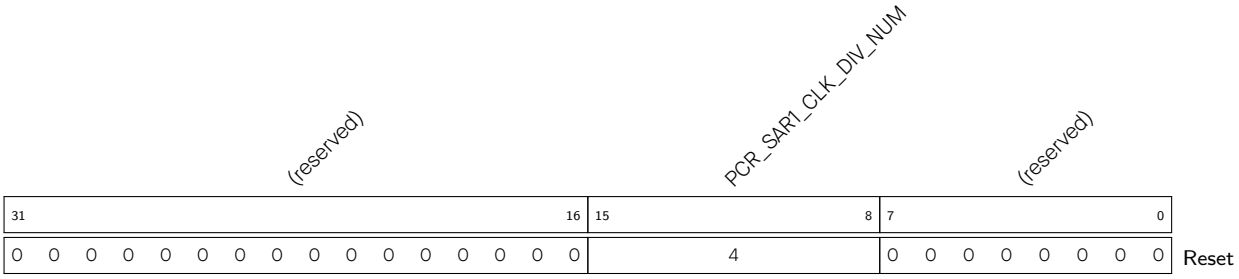
PCR_BUS_CLOCK_UPDATE Configures whether or not to update configurations for CPU_CLK division, AHB_CLK division and HP_ROOT_CLK clock source selection.

0: Not update configurations

1: Update configurations

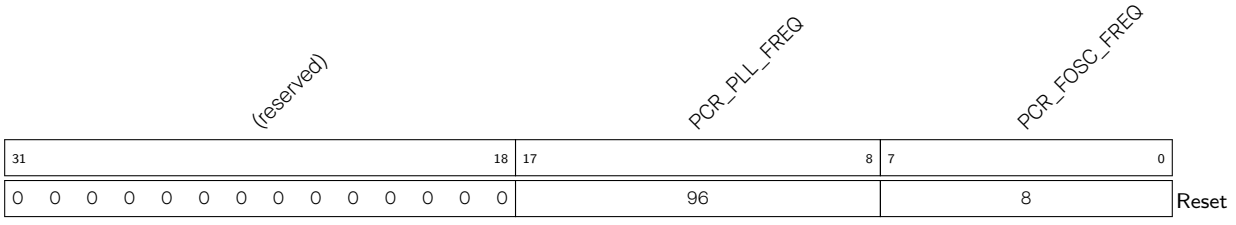
This bit is automatically cleared when configurations have been updated. (R/W/WTC)

Register 7.74. PCR_SAR_CLK_DIV_REG (0x014C)



PCR_SAR1_CLK_DIV_NUM Configures the divisor for SAR ADC clock to generate ADC analog control signals. (R/W)

Register 7.75. PCR_SYSCLK_FREQ_QUERY_0_REG (0x0120)



PCR_FOSC_FREQ Represents the frequency of RC_FAST_CLK.

Measurement unit: MHz.

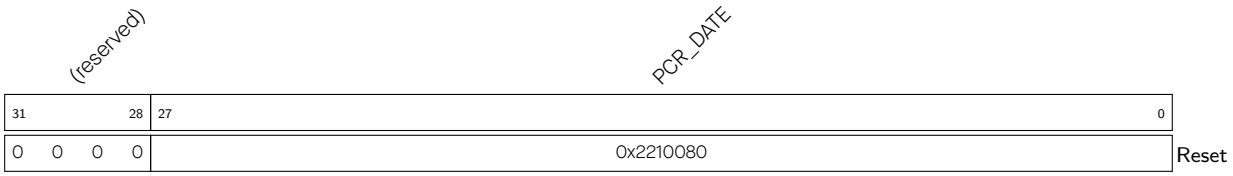
(HRO)

PCR_PLL_FREQ Represents the frequency of PLL_F96M_CLK.

Measurement unit: MHz.

(HRO)

Register 7.76. PCR_DATE_REG (0x0FFC)

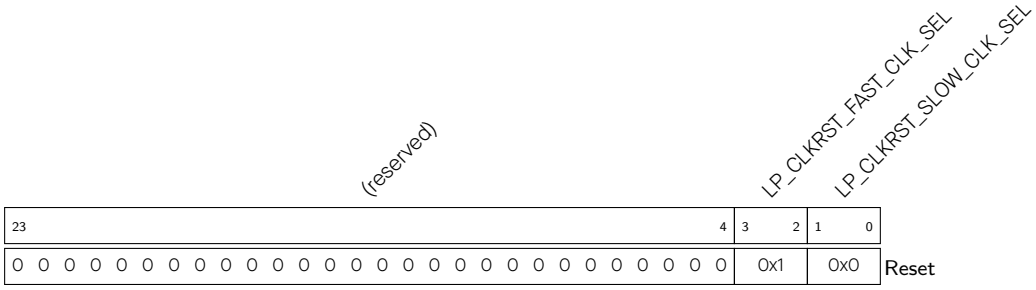


PCR_DATE Version control register. (R/W)

7.5.2 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 4.3-2 in Chapter 4 *System and Memory*.

Register 7.77. 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: OSC_SLOW_CLK

3: Invalid. No effect

(R/W)

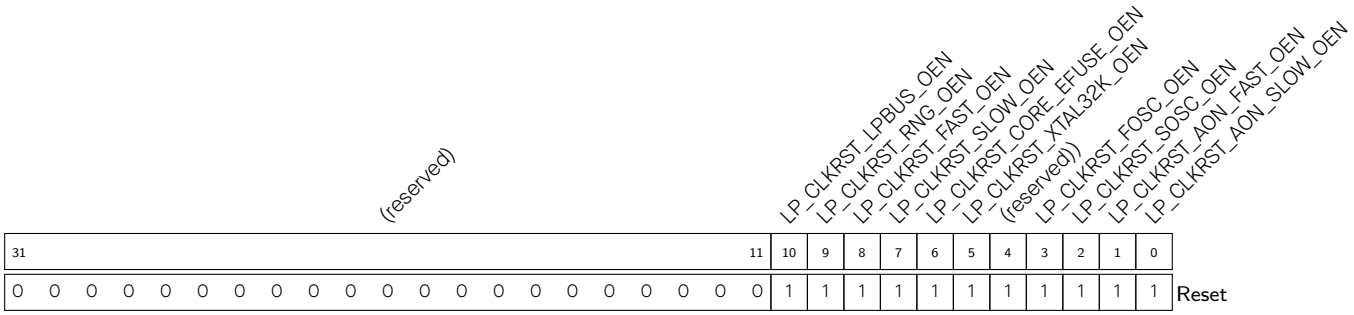
LP_CLKRST_FAST_CLK_SEL Configures the source of LP_FAST_CLK.

0: RC_FAST_CLK

1: XTAL_D2_CLK

(R/W)

Register 7.78. 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)

Continued on the next page...

Register 7.78. LP_CLKRST_LP_CLK_PO_EN_REG (0x0004)

Continued from the previous page...

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)

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)

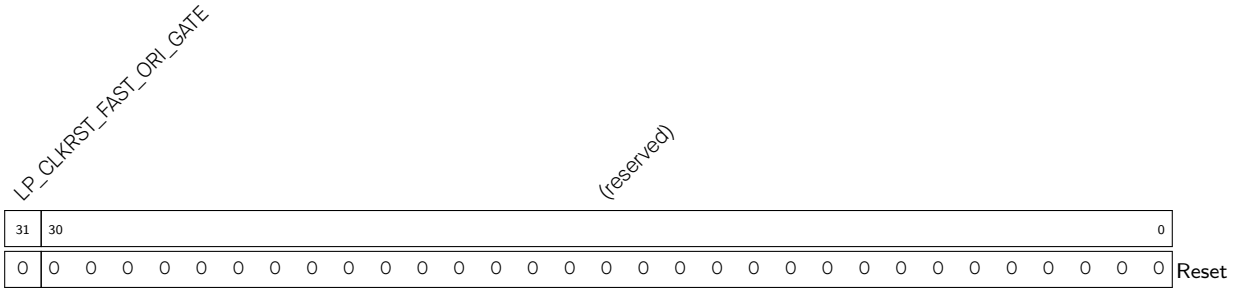
LP_CLKRST_LPBUS_OEN Configures whether to gate the LP bus clock signals to pad.

0: Disable the clock gate

1: Enable the clock gate

(R/W)

Register 7.79. LP_CLKRST_LP_CLK_EN_REG (0x0008)



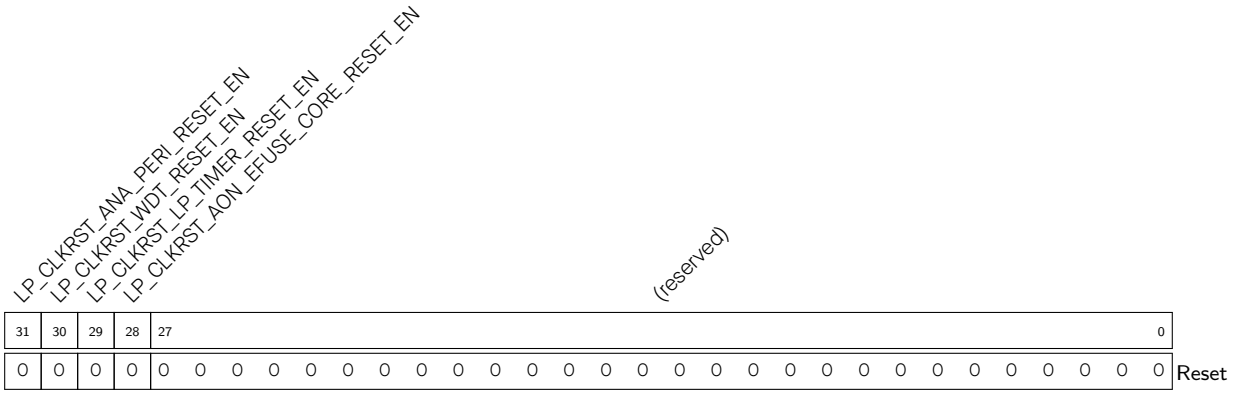
LP_CLKRST_FAST_ORI_GATE Configures the clock gate to LP_FAST_CLK.

0: Invalid. The clock gate controlled by hardware FSM

1: Force the clock to pass the clock gate

(R/W)

Register 7.80. 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)

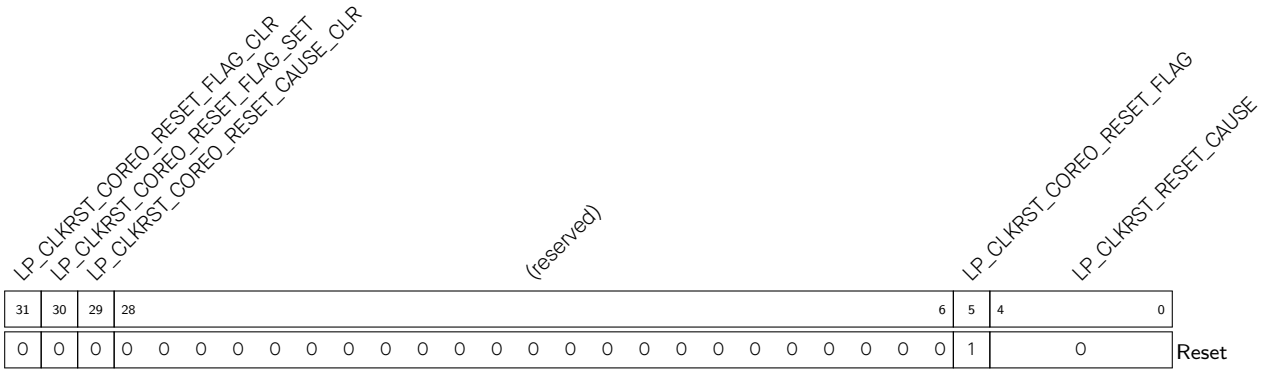
LP_CLKRST_ANA_PERI_RESET_EN Configures whether to reset analog peripherals, including the Brownout Detector.

0: Invalid. No effect

1: Reset

(R/W)

Register 7.81. LP_CLKRST_RESET_CAUSE_REG (0x0010)



RTC_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

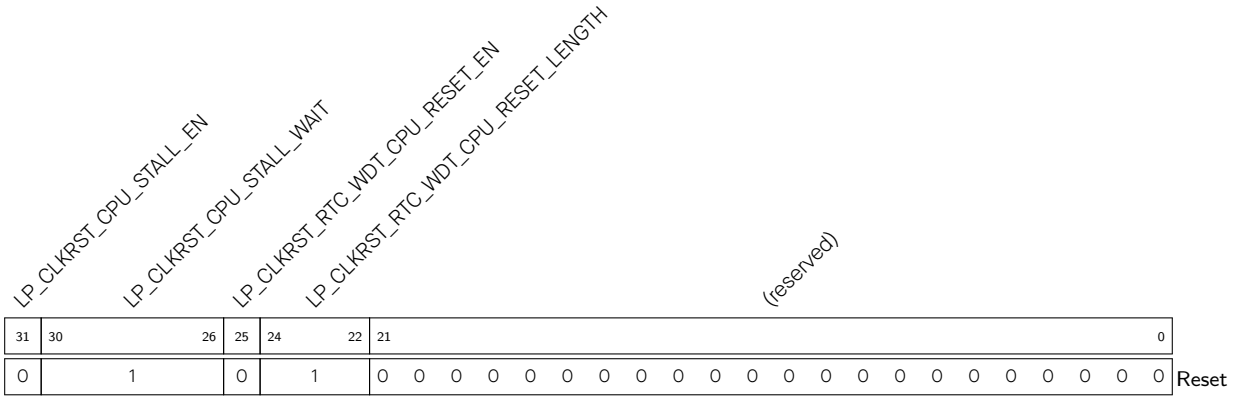
(RO)

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 7.82. LP_CLKRST_CPU_RESET_REG (0x0014)



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: Enable RWDT CPU reset.

1: Disable 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)

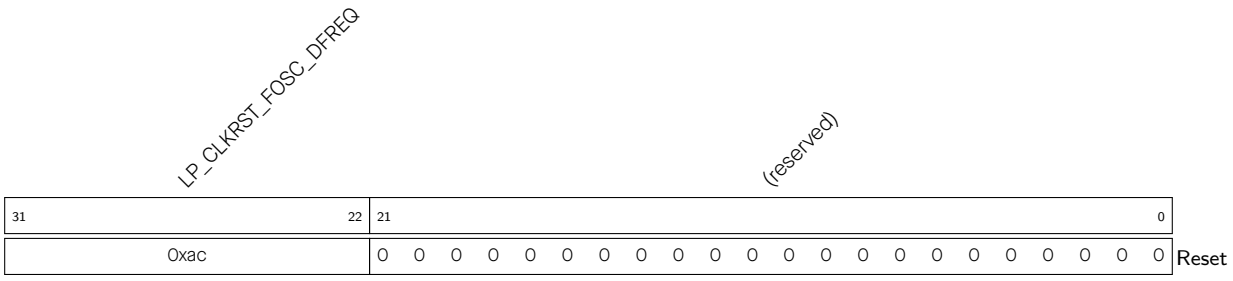
LP_CLKRST_CPU_STALL_EN Configures whether or not CPU will stall before RWDT and software reset CPU.

0: CPU will not stall.

1: CPU will stall.

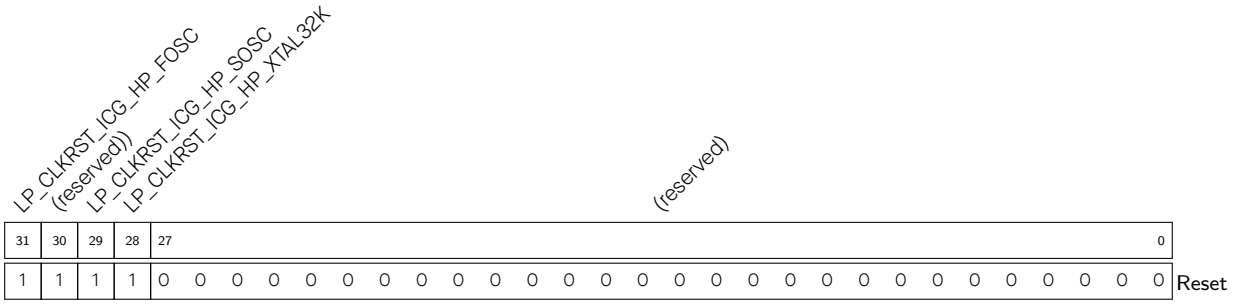
(R/W)

Register 7.83. LP_CLKRST_FOSC_CNTL_REG (0x0018)



LP_CLKRST_FOSC_DFREQ Configures the frequency of RC_FAST_CLK frequency. (R/W)

Register 7.84. 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)

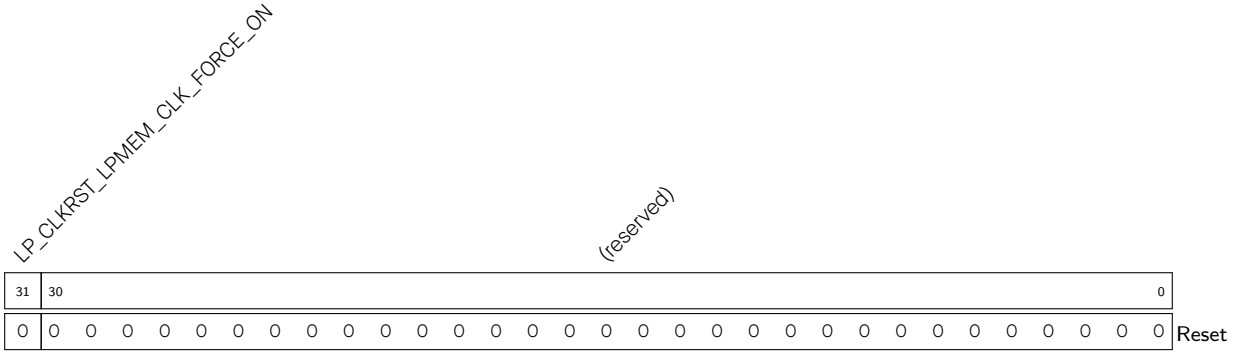
LP_CLKRST_ICG_HP_FOSC Configures whether to gate the RC_FAST_CLK signals to HP system.

0: Disable the clock gate

1: Enable the clock gate

(R/W)

Register 7.85. LP_CLKRST_LPMEM_FORCE_REG (0x0024)



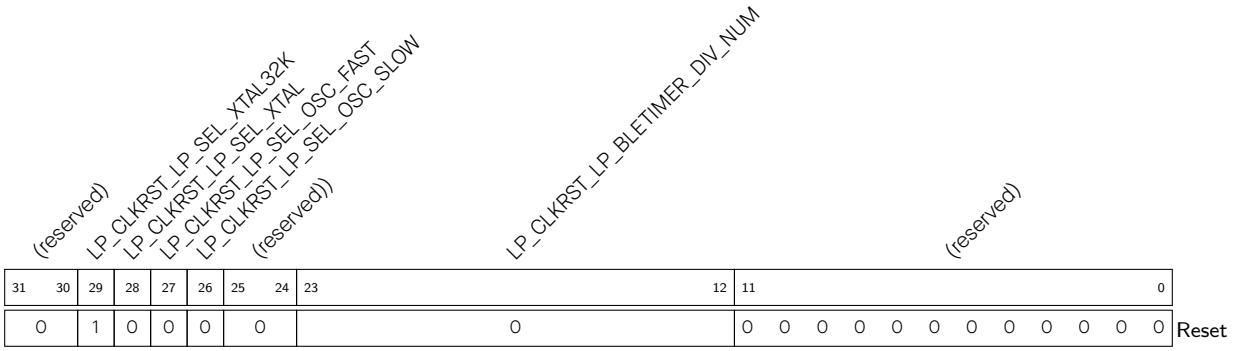
LP_CLKRST_LPMEM_CLK_FORCE_ON Configures whether to force enable the gate of LP Memory.

0: Invalid. The clock gate controlled by hardware FSM

1: Force open clock gate of LP Memory

(R/W)

Register 7.86. LP_CLKRST_LPPERI_REG (0x0028)



LP_CLKRST_LP_BLETIMER_DIV_NUM Configures the divisor of the clock divider for RTC BLE Timer. The working clock frequency of RTC BLE Timer is equal to the source clock frequency divided by ((LP_CLKRST_LP_BLETIMER_DIV_NUM - 1) / 2). (R/W)

LP_CLKRST_LP_SEL_OSC_SLOW Configures the clock source for RTC BLE Timer.

1: Selects RC_SLOW_CLK as the source clock for RTC BLE Timer.

0: Disables RC_SLOW_CLK as the source clock for RTC BLE Timer. (R/W)

LP_CLKRST_LP_SEL_OSC_FAST Configures the clock source for RTC BLE Timer.

1: Selects RC_FAST_CLK as the source clock for RTC BLE Timer.

0: Disables RC_FAST_CLK as the source clock for RTC BLE Timer. (R/W)

LP_CLKRST_LP_SEL_XTAL Configures the clock source for RTC BLE Timer.

1: Selects XTAL_CLK as the source clock for RTC BLE Timer.

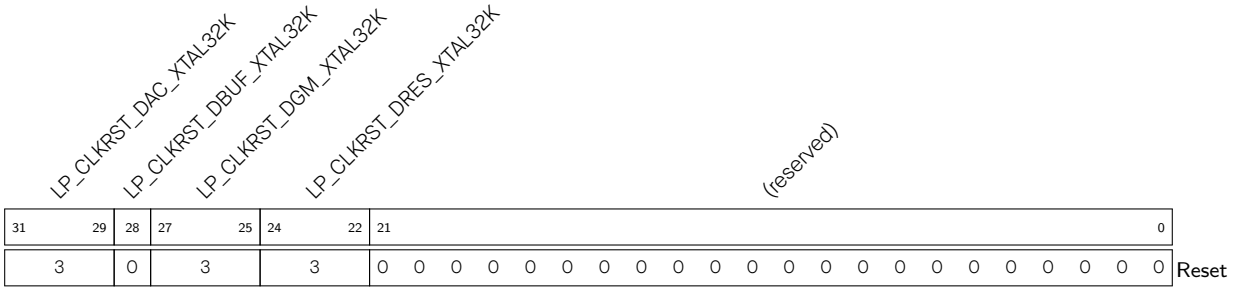
0: Disables XTAL_CLK as the source clock for RTC BLE Timer. (R/W

LP_CLKRST_LP_SEL_XTAL32K Configures the clock source for RTC BLE Timer.

1: Selects XTAL32K_CLK as the source clock for RTC BLE Timer.

0: Disables XTAL32K_CLK as the source clock for RTC BLE Timer. (R/W)

Register 7.87. LP_CLKRST_XTAL32K_REG (0x002C)



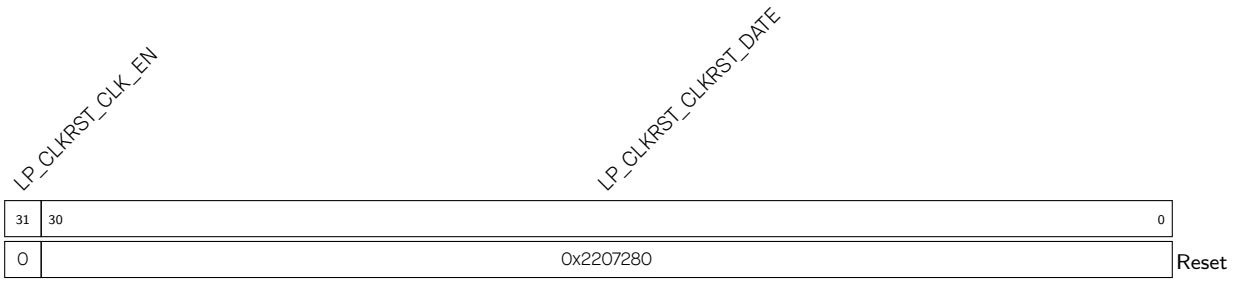
LP_CLKRST_DRES_XTAL32K Configures DRES (R/W)

LP_CLKRST_DGM_XTAL32K Configures DGM (R/W)

LP_CLKRST_DBUF_XTAL32K Configures DBUF (R/W)

LP_CLKRST_DAC_XTAL32K Configures DAC (R/W)

Register 7.88. LP_CLKRST_DATE_REG (0x03FC)



LP_CLKRST_CLKRST_DATE Version control register. (R/W)

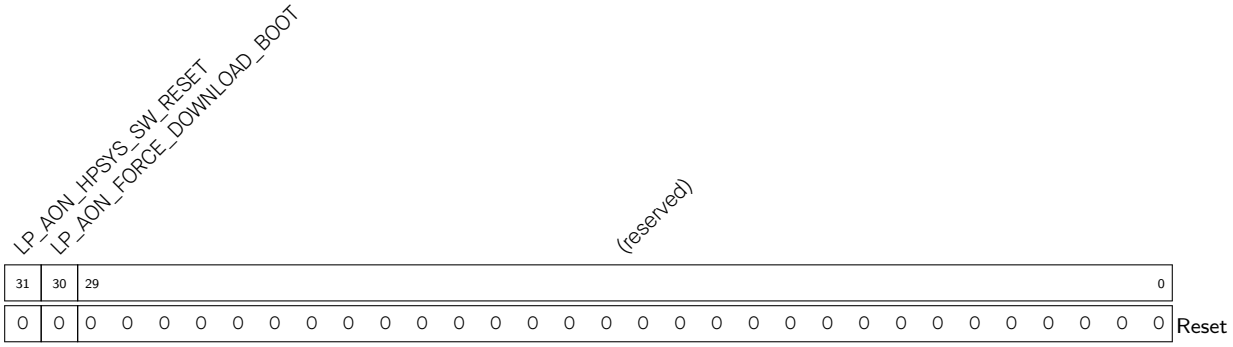
LP_CLKRST_CLK_EN Configures whether to force enable the clock gate for header registers.

0: Invalid

1: Force enable

(R/W)

Register 7.89. LP_AON_SYS_CFG_REG (0x0034)



LP_AON_FORCE_DOWNLOAD_BOOT Configures whether to trigger a CPU reset and switch the chip boot mode.

0: No effect.

1: If EFUSE_DIS_FORCE_DOWNLOAD is 0, software can force switch the chip from SPI Boot mode to Joint Download Boot mode and trigger a CPU reset.

(R/W)

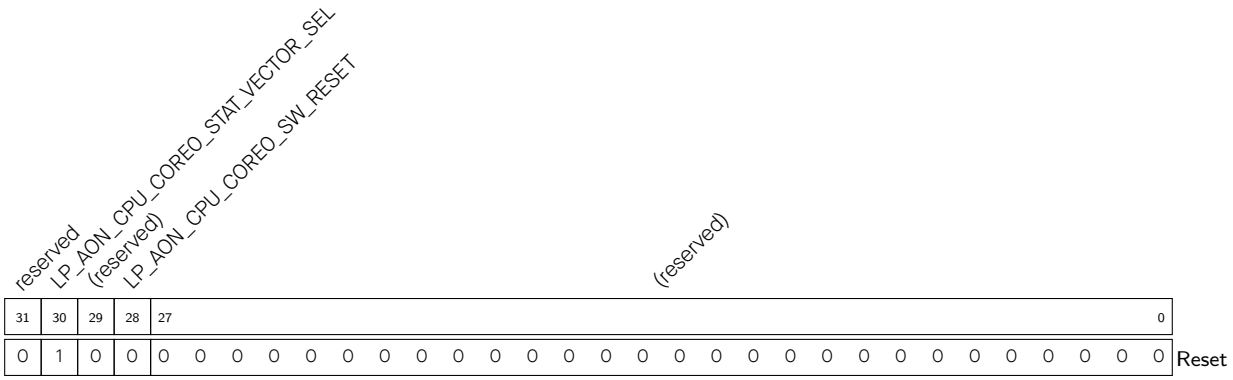
LP_AON_HPSYS_SW_RESET Configures whether to do software reset of the system.

0: Not reset

1: Reset

(WT)

Register 7.90. LP_AON_CPUCORE0_CFG_REG (0x0038)



LP_AON_CPU_CORE0_SW_RESET Configures whether to do software reset of the CPU.

0: Not reset

1: Reset

(WT)

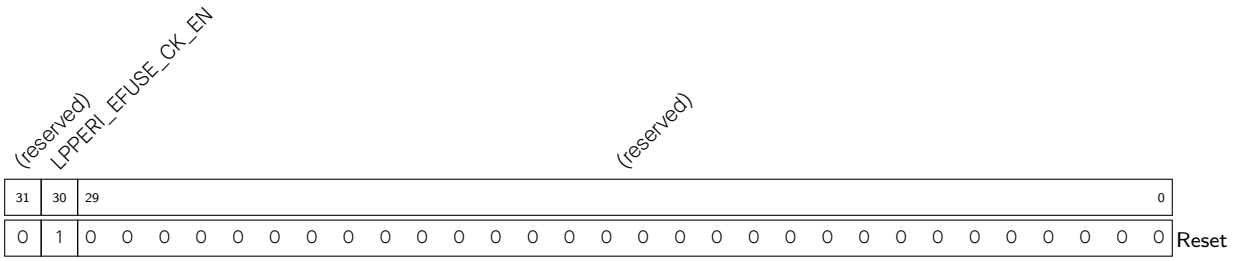
LP_AON_CPU_CORE0_STAT_VECTOR_SEL Configures whether to start up the CPU from the RTC fast memory.

0: Do not start up from the RTC fast memory.

1: Start up from the RTC fast memory.

(R/W)

Register 7.91. LPPERI_CLK_EN_REG (0x0000)



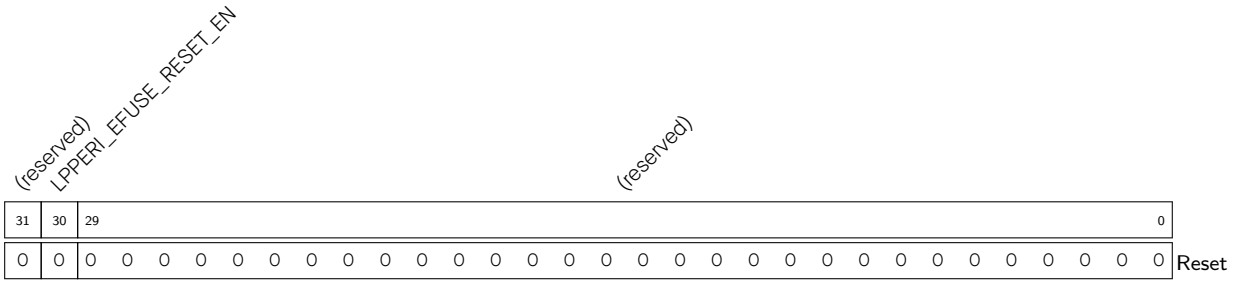
LPPERI_EFUSE_CK_EN Configures whether to gate the clock signals of eFuse Controller.

0: Disable the clock gate

1: Enable the clock gate

(R/W)

Register 7.92. LPPERI_RESET_EN_REG (0x0004)



LPPERI_EFUSE_RESET_EN Configures whether to do software reset of the eFuse Controller.

0: Not reset

1: Reset

(R/W)

Chapter 8

Chip Boot Control

8.1 Overview

Chip boot process and some chip functions are determined on power-on or hardware reset by strapping pins and eFuses. The following functionality can be determined:

- chip boot mode
- enable or disable ROM messages printing to UART0/USB
- source of JTAG signals

ESP32-H2 has three strapping pins:

- GPIO8
- GPIO9
- GPIO25

During Chip Reset (see Chapter 7 *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.

8.2 Functional Description

This section introduces chip reset functions and the patterns of the strapping pins and eFuse values to invoke each function.

Notice:

Only documented patterns should be used. If an undocumented pattern is used, it may trigger unexpected behaviors.

8.2.1 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 8.2-1).

Table 8.2-1. Default Configuration of Strapping Pins

Strapping Pin	Default Configuration
GPIO8	Floating
GPIO9	Pull-up
GPIO25	Floating

To change the strapping bit values, users can apply external pull-down/pull-up resistors, or use host MCU GPIOs to control the voltage level of these pins when powering on ESP32-H2. After the reset is released, the strapping pins work as normal-function pins.

8.2.2 Boot Mode Control

The values of GPIO9, GPIO8, GPIO3 and GPIO2 at reset determine the boot mode after the reset is released. Table 8.2-2 shows the strapping pin values of GPIO9, GPIO8, GPIO3 and GPIO2, and the associated boot modes.

Table 8.2-2. Boot Mode Control

Boot Mode	GPIO9	GPIO8	GPIO3	GPIO2
SPI Boot mode	1	Ignored	Ignored	Ignored
Joint Download Boot mode[1]	0	1	Ignored	Ignored
SPI Download Boot mode[2]	0	0	0	1
Invalid Combination[3]	0	0	Ignored	0

[1] Joint Download Boot mode: This mode supports the following download methods:
- USB-Serial-JTAG Download Boot
- UART Download Boot

[2] SPI Download Boot mode: GPIO3 and GPIO2 need to be reserved only when using SPI Download Boot mode. GPIO3 and GPIO2 are floating by default and are in a high-impedance state at reset.

[3] Invalid Combination: This combination can trigger unexpected behavior and should be avoided.

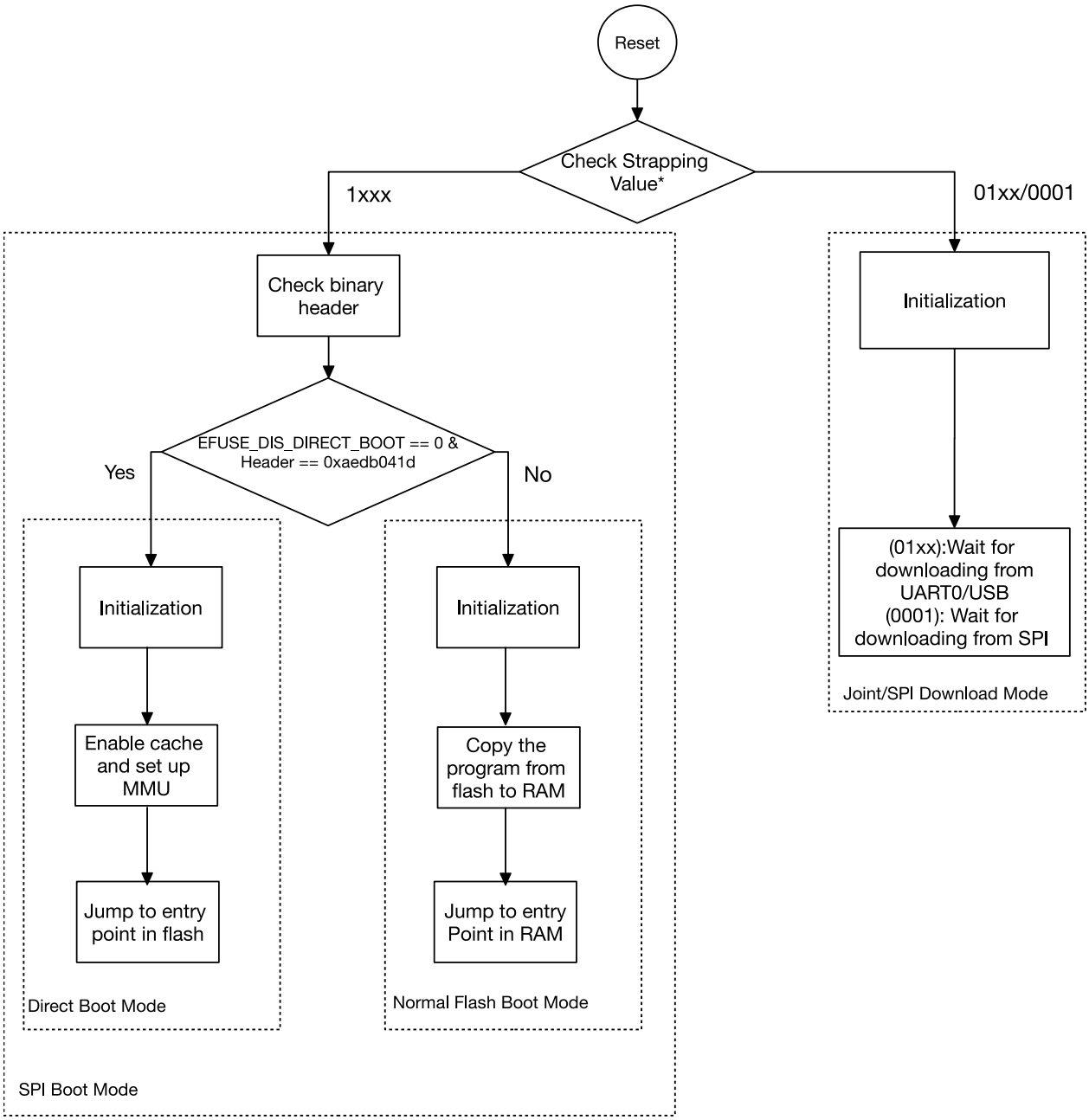
In SPI Boot mode, the ROM bootloader loads and executes the program from SPI flash to boot the system. SPI Boot mode can be further classified as follows:

- Normal Flash Boot: supports Secure Boot. The ROM bootloader loads the program from flash into 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 8.2-1.

In Joint Download Boot mode, users can download binary files into flash using UART0 or USB interface. It is also possible to download binary files into SRAM and execute it from SRAM.

In SPI Download Boot mode, users can download binary files into flash using SPI interface. It is also possible to download binary files into SRAM and execute it from SRAM.

Figure 8.2-1 shows the detailed boot flow of the chip.



***Note**: The strapping values "1xxx" and "01xx/0001" are the combination of GPIO9, GPIO8, GPIO3 and GPIO2 pins, see Table 8.2-2.

Figure 8.2-1. Chip Boot Flow

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 33 *USB Serial/JTAG Controller (USB_SERIAL_JTAG)*.

8.2.3 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 8.2-3. ROM Message Printing Control

ROM Message Printing	LP_AON_STORE4_REG[0]	EFUSE_UART_PRINT_CONTROL	GPIO8
Always printed to UART0	0	0(0b00)	Ignored
Enabled		1(0b01)	0
Disabled			1
Disabled		2(0b10)	0
Enabled			1
Disabled		3(0b11)	Ignored
Disabled	1	Ignored	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_PRINT.

Note that if EFUSE_DIS_USB_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.

8.2.4 JTAG Signal Source Control

GPIO25 controls the source of JTAG signals together with EFUSE_DIS_PAD_JTAG, EFUSE_DIS_USB_JTAG, and EFUSE_JTAG_SEL_ENABLE. See Table 8.2-4.

Table 8.2-4. JTAG Signal Source Control

JTAG Signal Source	EFUSE_DIS_PAD_JTAG	EFUSE_DIS_USB_JTAG	EFUSE_JTAG_SEL_ENABLE	GPIO25
USB Serial/JTAG Controller	0	0	0	Ignored
	0	0	1	1
	1	0	Ignored	Ignored
JTAG pins MTDI, MTCK, MTMS, and MTDO	0	0	1	0
	0	1	Ignored	Ignored
JTAG disabled	1	1	Ignored	Ignored

Chapter 9

Interrupt Matrix (INTMTX)

9.1 Overview

The interrupt matrix embedded in ESP32-H2 routes one or multiple peripheral interrupt sources to any one of the ESP-RISC-V CPU's peripheral interrupts.

The ESP32-H2 has 65 peripheral interrupt sources that can be routed to any of the 28 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 1 *ESP-RISC-V CPU* > Section 1.6 *Interrupt Controller*.

9.2 Interrupt Terminology in ESP32-H2

The following terms related to interrupts are defined in the context of the *ESP32-H2 Technical Reference Manual* to help readers better understand this document:

9.2.1 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-H2 Technical Reference Manual* may use the term "interrupt" in a broader sense to refer to both interrupt signal and interrupt source.

9.2.2 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.

9.2.3 Interrupt Flow in ESP32-H2

Figure 9.2-1 shows the interrupt flow in ESP32-H2.

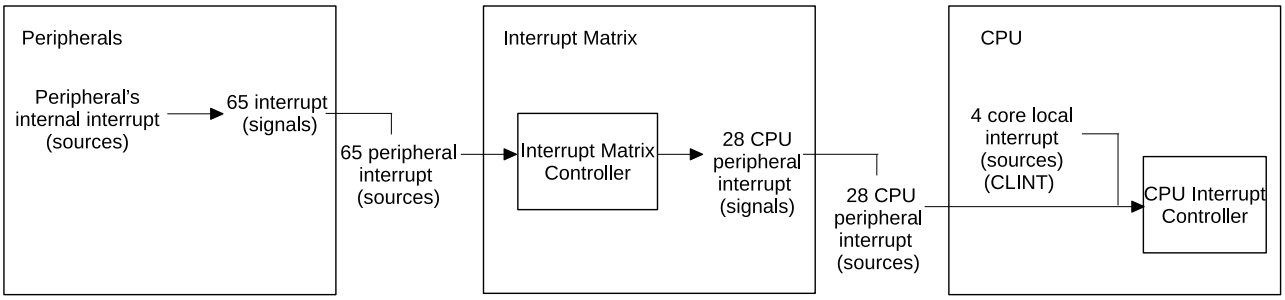


Figure 9.2-1. Interrupt Flow in ESP32-H2

9.3 Features

The interrupt matrix embedded in ESP32-H2 has the following features:

- 65 peripheral interrupt sources accepted as input
- 28 CPU peripheral interrupts generated to the CPU as output
- Current interrupt status query of peripheral interrupt sources
- Multiple interrupt sources mapping to a single CPU interrupt (i.e., shared interrupts)

9.4 Architecture

Figure 9.4-1 shows the structure of the interrupt matrix.

You need to configure the interrupt matrix registers to map the peripheral interrupt sources to the CPU interrupts. The Interrupt Matrix Controller in Figure 9.4-1 manages the mapping and sends the interrupt status of each interrupt source to the interrupt status registers which belong to the interrupt matrix registers.

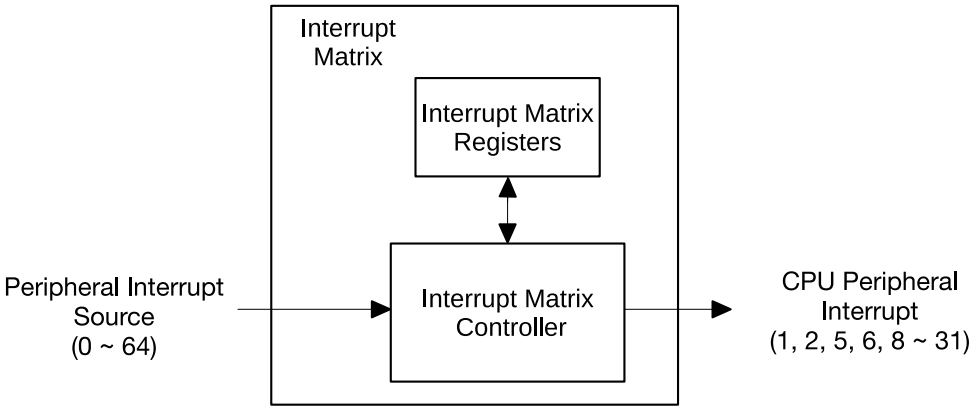


Figure 9.4-1. Interrupt Matrix Structure

9.5 Functional Description

9.5.1 Peripheral Interrupt Sources

The ESP32-H2 has 65 peripheral interrupt sources in total. Table 9.5-1 lists all these sources and their mapping registers, as well as the interrupt status registers.

- Column "No.": Peripheral interrupt source number, can be 0 ~ 64.
- 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 routing of the peripheral interrupt sources to the CPU peripheral interrupts.
- Column "Interrupt Status Register": Registers used to reflect the interrupt source status.
  - Column "Interrupt Status Register - Bit": Bit position in status register, indicating the interrupt status.
  - Column "Interrupt Status Register - Name": Name of status registers.

Table 9.5-1. 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	*Low-Power Management*	PMU_INTR	INTMTX_COREO_PMU_INTR_MAP_REG	0	
1	*eFuse Controller (EFUSE)*	EFUSE_INTR	INTMTX_COREO_EFUSE_INTR_MAP_REG	1	
2	*Low-Power Management*	LP_RTC_TIMER_INTR	INTMTX_COREO_LP_RTC_TIMER_INTR_MAP_REG	2	
3	n/a	reserved	reserved	3	
4	*Low-Power Management*	LP_WDT_INTR	INTMTX_COREO_LP_WDT_INTR_MAP_REG	4	
5	*System Registers*	LP_PERI_TIMEOUT_INTR	INTMTX_COREO_LP_PERI_TIMEOUT_INTR_MAP_REG	5	
6	*Permission Control (PMS)*	LP_APM_M0_INTR	INTMTX_COREO_LP_APM_M0_INTR_MAP_REG	6	
7	*ESP-RISC-V CPU*	CPU_INTR_FROM_CPU_0	INTMTX_COREO_CPU_INTR_FROM_CPU_0_MAP_REG	7	
8	*ESP-RISC-V CPU*	CPU_INTR_FROM_CPU_1	INTMTX_COREO_CPU_INTR_FROM_CPU_1_MAP_REG	8	
9	*ESP-RISC-V CPU*	CPU_INTR_FROM_CPU_2	INTMTX_COREO_CPU_INTR_FROM_CPU_2_MAP_REG	9	
10	*ESP-RISC-V CPU*	CPU_INTR_FROM_CPU_3	INTMTX_COREO_CPU_INTR_FROM_CPU_3_MAP_REG	10	
11	*Debug Assistant (ASSIST_DEBUG, MEM_MONITOR)*	ASSIST_DEBUG_INTR	INTMTX_COREO_ASSIST_DEBUG_INTR_MAP_REG	11	
12	*ESP-RISC-V CPU*	TRACE_INTR	INTMTX_COREO_TRACE_INTR_MAP_REG	12	
13	n/a	reserved	reserved	13	
14	*System Registers*	CPU_PERI_TIMEOUT_INTR	INTMTX_COREO_CPU_PERI_TIMEOUT_INTR_MAP_REG	14	INTMTX_COREO_INT_STATUS_0_REG
15	n/a	reserved	reserved	15	
16	n/a	reserved	reserved	16	
17	n/a	reserved	reserved	17	
18	n/a	reserved	reserved	18	
19	n/a	reserved	reserved	19	
20	n/a	reserved	reserved	20	
21	n/a	reserved	reserved	21	
22	*IO MUX and GPIO Matrix (GPIO, IO MUX)*	GPIO_INTERRUPT_PRO	INTMTX_COREO_GPIO_INTERRUPT_PRO_MAP_REG	22	
23	*IO MUX and GPIO Matrix (GPIO, IO MUX)*	reserved	reserved	23	
24	*Low-Power Management*	PAU_INTR	INTMTX_COREO_PAU_INTR_MAP_REG	24	
25	*System Registers*	HP_PERI_TIMEOUT_INTR	INTMTX_COREO_HP_PERI_TIMEOUT_INTR_MAP_REG	25	
26	*Permission Control (PMS)*	HP_APM_M0_INTR	INTMTX_COREO_HP_APM_M0_INTR_MAP_REG	26	
27	*Permission Control (PMS)*	HP_APM_M1_INTR	INTMTX_COREO_HP_APM_M1_INTR_MAP_REG	27	
28	*Permission Control (PMS)*	HP_APM_M2_INTR	INTMTX_COREO_HP_APM_M2_INTR_MAP_REG	28	
29	*Permission Control (PMS)*	HP_APM_M3_INTR	INTMTX_COREO_HP_APM_M3_INTR_MAP_REG	29	
30	n/a	reserved	reserved	30	
31	*I2S Controller (I2S)*	I2S_INTR	INTMTX_COREO_I2S_INTR_MAP_REG	31	
32	*UART Controller (UART)*	UHCI0_INTR	INTMTX_COREO_UHCI0_INTR_MAP_REG	0	INTMTX_COREO_INT_STATUS_1_REG
33	*UART Controller (UART)*	UART0_INTR	INTMTX_COREO_UART0_INTR_MAP_REG	1	
34	*UART Controller (UART)*	UART1_INTR	INTMTX_COREO_UART1_INTR_MAP_REG	2	
35	*LED PWM Controller (LEDC)*	LEDC_INTR	INTMTX_COREO_LEDC_INTR_MAP_REG	3	

No.	Chapter	Interrupt Source	Interrupt Source Mapping Register	Interrupt Status Register	
				Bit	Name
36	*Two-wire Automotive Interface (TWAI)*	TWAIO_INTR	INTMTX_COREO_TWAIO_INTR_MAP_REG	4	
37	*USB Serial/JTAG Controller (USB_SERIAL_JTAG)*	USB_SERIAL_JTAG_INTR	INTMTX_COREO_USB_INTR_MAP_REG	5	
38	*Remote Control Peripheral (RMT)*	RMT_INTR	INTMTX_COREO_RMT_INTR_MAP_REG	6	
39	*I2C Controller (I2C)*	I2C_EXTO_INTR	INTMTX_COREO_I2C_EXTO_INTR_MAP_REG	7	
40	*I2C Controller (I2C)*	I2C_EXT1_INTR	INTMTX_COREO_I2C_EXT1_INTR_MAP_REG	8	
41	*Timer Group (TIMG)*	TGO_TO_INTR	INTMTX_COREO_TGO_TO_INTR_MAP_REG	9	
42	*Timer Group (TIMG)*	TGO_WDT_INTR	INTMTX_COREO_TGO_WDT_INTR_MAP_REG	10	
43	*Timer Group (TIMG)*	TG1_TO_INTR	INTMTX_COREO_TG1_TO_INTR_MAP_REG	11	
44	*Timer Group (TIMG)*	TG1_WDT_INTR	INTMTX_COREO_TG1_WDT_INTR_MAP_REG	12	
45	*System Timer (SYSTIMER)*	SYSTIMER_TARGETO_INTR	INTMTX_COREO_SYSTIMER_TARGETO_INTR_MAP_REG	13	
46	*System Timer (SYSTIMER)*	SYSTIMER_TARGET1_INTR	INTMTX_COREO_SYSTIMER_TARGET1_INTR_MAP_REG	14	
47	*System Timer (SYSTIMER)*	SYSTIMER_TARGET2_INTR	INTMTX_COREO_SYSTIMER_TARGET2_INTR_MAP_REG	15	
48	*ADC Controller (ADC)*	APB_ADC_INTR	INTMTX_COREO_APB_ADC_INTR_MAP_REG	16	
49	*Motor Control PWM (MCPWM)*	PWM_INTR	INTMTX_COREO_PWM_INTR_MAP_REG	17	
50	*Pulse Count Controller (PCNT)*	PCNT_INTR	INTMTX_COREO_PCNT_INTR_MAP_REG	18	
51	*Parallel IO Controller (PARLIO)*	PARL_IO_TX_INTR	INTMTX_COREO_PARL_IO_TX_INTR_MAP_REG	19	
52	*Parallel IO Controller (PARLIO)*	PARL_IO_RX_INTR	INTMTX_COREO_PARL_IO_RX_INTR_MAP_REG	20	
53	*GDMA Controller (GDMA)*	GDMA_IN_CHO_INTR	INTMTX_COREO_DMA_IN_CHO_INTR_MAP_REG	21	
54	*GDMA Controller (GDMA)*	GDMA_IN_CH1_INTR	INTMTX_COREO_DMA_IN_CH1_INTR_MAP_REG	22	
55	*GDMA Controller (GDMA)*	GDMA_IN_CH2_INTR	INTMTX_COREO_DMA_IN_CH2_INTR_MAP_REG	23	
56	*GDMA Controller (GDMA)*	GDMA_OUT_CHO_INTR	INTMTX_COREO_DMA_OUT_CHO_INTR_MAP_REG	24	
57	*GDMA Controller (GDMA)*	GDMA_OUT_CH1_INTR	INTMTX_COREO_DMA_OUT_CH1_INTR_MAP_REG	25	
58	*GDMA Controller (GDMA)*	GDMA_OUT_CH2_INTR	INTMTX_COREO_DMA_OUT_CH2_INTR_MAP_REG	26	
59	*SPI Controller (SPI)*	GPSPI2_INTR	INTMTX_COREO_GPSPI2_INTR_MAP_REG	27	
60	*AES Accelerator (AES)*	AES_INTR	INTMTX_COREO_AES_INTR_MAP_REG	28	
61	*SHA Accelerator (SHA)*	SHA_INTR	INTMTX_COREO_SHA_INTR_MAP_REG	29	
62	*RSA Accelerator (RSA)*	RSA_INTR	INTMTX_COREO_RSA_INTR_MAP_REG	30	
63	*ECC Accelerator (ECC)*	ECC_INTR	INTMTX_COREO_ECC_INTR_MAP_REG	31	
64	*ECDSA Digital Signature Peripheral (ECDSA_DS)*	ECDSA_INTR	INTMTX_COREO_ECDSA_INTR_MAP_REG	0	INTMTX_COREO_INT_STATUS_2_REG

GoBack

9.5.2 CPU Interrupts

The ESP32-H2 implements its interrupt mechanism using an interrupt controller instead of standard RISC-V ISA. The CPU has 32 interrupts, numbered from 0 to 31. The interrupts numbered 0, 3, 4, and 7 are used by the CPU for core-local interrupts (CLINT), while the remaining 28 interrupts (numbered 1, 2, 5, 6, and 8 ~ 31) are available for use by the interrupt matrix.

Each CPU 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 procedure of CPU interrupts, see Chapter 1 *ESP-RISC-V CPU* > Section 1.6*Interrupt Controller*. The configuration registers of CPU interrupts are listed in Section 9.6.2 *Interrupt Priority Register Summary*.

9.5.3 Assign Peripheral Interrupt Source to CPU Interrupt

In this section, the following terms are used to describe the operation of the interrupt matrix.

- *SOURCE*: stands for a peripheral interrupt source in Table 9.5-1.
- INTMTX_COREO_*SOURCE*_INTR_MAP_REG: stands for the interrupt source mapping register for a *SOURCE*.
- Num_P: stands for the index of CPU interrupts which can be 1, 2, 5, 6, 8 ~ 31.
- Interrupt_P: stands for the CPU interrupt numbered as Num_P.

9.5.3.1 Assign One SOURCE to CPU 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.

9.5.3.2 Assign Multiple SOURCEs to CPU 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, CPU should check the interrupt status registers to figure out which peripheral generated the interrupt. For more information, see Chapter 1 *ESP-RISC-V CPU* > Section 1.6*Interrupt Controller*.

9.5.3.3 Unassign SOURCE

Writing 0 to the INTMTX_COREO_*SOURCE*_INTR_MAP_REG register disables the corresponding interrupt source.

9.5.4 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 9.5-1.

9.6 Register Summary

9.6.1 Interrupt Matrix Register Summary

The addresses in this section are relative to the interrupt matrix base address provided in Table 4.3-2 in Chapter 4 *System and Memory*.

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

Name	Description	Address	Access
Peripheral Interrupt Source Mapping Registers			
INTMTX_COREO_PMU_INTR_MAP_REG	PMU_INTR mapping register	0x0000	R/W
INTMTX_COREO_EFUSE_INTR_MAP_REG	EFUSE_INTR mapping register	0x0004	R/W
INTMTX_COREO_LP_RTC_TIMER_INTR_MAP_REG	LP_RTC_TIMER_INTR mapping register	0x0008	R/W
INTMTX_COREO_LP_WDT_INTR_MAP_REG	LP_WDT_INTR mapping register	0x0010	R/W
INTMTX_COREO_LP_PERI_TIMEOUT_INTR_MAP_REG	LP_PERI_TIMEOUT_INTR mapping register	0x0014	R/W
INTMTX_COREO_LP_APM_M0_INTR_MAP_REG	LP_APM_M0_INTR mapping register	0x0018	R/W
INTMTX_COREO_CPU_INTR_FROM_CPU_0_MAP_REG	CPU_INTR_FROM_CPU_0 mapping register	0x001C	R/W
INTMTX_COREO_CPU_INTR_FROM_CPU_1_MAP_REG	CPU_INTR_FROM_CPU_1 mapping register	0x0020	R/W
INTMTX_COREO_CPU_INTR_FROM_CPU_2_MAP_REG	CPU_INTR_FROM_CPU_2 mapping register	0x0024	R/W
INTMTX_COREO_CPU_INTR_FROM_CPU_3_MAP_REG	CPU_INTR_FROM_CPU_3 mapping register	0x0028	R/W
INTMTX_COREO_ASSIST_DEBUG_INTR_MAP_REG	ASSIST_DEBUG_INTR mapping register	0x002C	R/W
INTMTX_COREO_TRACE_INTR_MAP_REG	TRACE_INTR mapping register	0x0030	R/W
INTMTX_COREO_CPU_PERI_TIMEOUT_INTR_MAP_REG	CPU_PERI_TIMEOUT_INTR mapping register	0x0038	R/W
INTMTX_COREO_GPIO_INTERRUPT_PRO_MAP_REG	GPIO_INTERRUPT_PRO mapping register	0x0058	R/W
INTMTX_COREO_PAU_INTR_MAP_REG	PAU_INTR mapping register	0x0060	R/W
INTMTX_COREO_HP_PERI_TIMEOUT_INTR_MAP_REG	HP_PERI_TIMEOUT_INTR mapping register	0x0064	R/W
INTMTX_COREO_HP_APM_M0_INTR_MAP_REG	HP_APM_M0_INTR mapping register	0x0068	R/W
INTMTX_COREO_HP_APM_M1_INTR_MAP_REG	HP_APM_M1_INTR mapping register	0x006C	R/W
INTMTX_COREO_HP_APM_M2_INTR_MAP_REG	HP_APM_M2_INTR mapping register	0x0070	R/W
INTMTX_COREO_HP_APM_M3_INTR_MAP_REG	HP_APM_M3_INTR mapping register	0x0074	R/W
INTMTX_COREO_I2S_INTR_MAP_REG	I2S_INTR mapping register	0x007C	R/W

GoBack

Name	Description	Address	Access
INTMTX_COREO_UHCIO_INTR_MAP_REG	UHCIO_INTR mapping register	0x0080	R/W
INTMTX_COREO_UART0_INTR_MAP_REG	UART0_INTR mapping register	0x0084	R/W
INTMTX_COREO_UART1_INTR_MAP_REG	UART1_INTR mapping register	0x0088	R/W
INTMTX_COREO_LEDC_INTR_MAP_REG	LEDC_INTR mapping register	0x008C	R/W
INTMTX_COREO_TWAIO_INTR_MAP_REG	TWAIO_INTR mapping register	0x0090	R/W
INTMTX_COREO_USB_INTR_MAP_REG	USB_SERIAL_JTAG_INTR mapping register	0x0094	R/W
INTMTX_COREO_RMT_INTR_MAP_REG	RMT_INTR mapping register	0x0098	R/W
INTMTX_COREO_I2C_EXT0_INTR_MAP_REG	I2C_EXT0_INTR mapping register	0x009C	R/W
INTMTX_COREO_I2C_EXT1_INTR_MAP_REG	I2C_EXT1_INTR mapping register	0x00A0	R/W
INTMTX_COREO_TG0_T0_INTR_MAP_REG	TG0_T0_INTR mapping register	0x00A4	R/W
INTMTX_COREO_TG0_WDT_INTR_MAP_REG	TG0_WDT_INTR mapping register	0x00A8	R/W
INTMTX_COREO_TG1_T0_INTR_MAP_REG	TG1_T0_INTR mapping register	0x00AC	R/W
INTMTX_COREO_TG1_WDT_INTR_MAP_REG	TG1_WDT_INTR mapping register	0x00B0	R/W
INTMTX_COREO_SYSTIMER_TARGET0_INTR_MAP_REG	SYSTIMER_TARGET0_INTR mapping register	0x00B4	R/W
INTMTX_COREO_SYSTIMER_TARGET1_INTR_MAP_REG	SYSTIMER_TARGET1_INTR mapping register	0x00B8	R/W
INTMTX_COREO_SYSTIMER_TARGET2_INTR_MAP_REG	SYSTIMER_TARGET2_INTR mapping register	0x00BC	R/W
INTMTX_COREO_APB_ADC_INTR_MAP_REG	APB_ADC_INTR mapping register	0x00C0	R/W
INTMTX_COREO_PWM_INTR_MAP_REG	PWM_INTR mapping register	0x00C4	R/W
INTMTX_COREO_PCNT_INTR_MAP_REG	PCNT_INTR mapping register	0x00C8	R/W
INTMTX_COREO_PARL_IO_TX_INTR_MAP_REG	PARL_IO_TX_INTR mapping register	0x00CC	R/W
INTMTX_COREO_PARL_IO_RX_INTR_MAP_REG	PARL_IO_RX_INTR mapping register	0x00D0	R/W
INTMTX_COREO_DMA_IN_CH0_INTR_MAP_REG	DMA_IN_CH0_INTR mapping register	0x00D4	R/W
INTMTX_COREO_DMA_IN_CH1_INTR_MAP_REG	DMA_IN_CH1_INTR mapping register	0x00D8	R/W
INTMTX_COREO_DMA_IN_CH2_INTR_MAP_REG	DMA_IN_CH2_INTR mapping register	0x00DC	R/W
INTMTX_COREO_DMA_OUT_CH0_INTR_MAP_REG	DMA_OUT_CH0_INTR mapping register	0x00E0	R/W
INTMTX_COREO_DMA_OUT_CH1_INTR_MAP_REG	DMA_OUT_CH1_INTR mapping register	0x00E4	R/W
INTMTX_COREO_DMA_OUT_CH2_INTR_MAP_REG	DMA_OUT_CH2_INTR mapping register	0x00E8	R/W
INTMTX_COREO_GPSPI2_INTR_MAP_REG	GPSPI2_INTR mapping register	0x00EC	R/W
INTMTX_COREO_AES_INTR_MAP_REG	AES_INTR mapping register	0x00F0	R/W

GoBack

Name	Description	Address	Access
INTMTX_COREO_SHA_INTR_MAP_REG	SHA_INTR mapping register	0x00F4	R/W
INTMTX_COREO_RSA_INTR_MAP_REG	RSA_INTR mapping register	0x00F8	R/W
INTMTX_COREO_ECC_INTR_MAP_REG	ECC_INTR mapping register	0x00FC	R/W
INTMTX_COREO_ECDSA_INTR_MAP_REG	ECDSA_INTR mapping register	0x0100	R/W
Interrupt Status Registers			
INTMTX_COREO_INT_STATUS_0_REG	Status register for interrupt sources 0 ~ 31	0x0104	RO
INTMTX_COREO_INT_STATUS_1_REG	Status register for interrupt sources 32 ~ 63	0x0108	RO
INTMTX_COREO_INT_STATUS_2_REG	Status register for interrupt source 64	0x010C	RO
Version Control Registers			
INTMTX_COREO_INTERRUPT_REG_DATE_REG	Version control register	0x07FC	R/W

9.6.2 Interrupt Priority Register Summary

The addresses in this section are relative to the interrupt priority base address provided in Table 4.3-2 in Chapter *4 System and Memory*.

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

Name	Description	Address	Access
Configuration Registers			
INTPRI_COREO_CPU_INT_ENABLE_REG	Enable register for CPU interrupts	0x0000	R/W
INTPRI_COREO_CPU_INT_TYPE_REG	Type configuration register for CPU interrupts	0x0004	R/W
INTPRI_COREO_CPU_INT_EIP_STATUS_REG	Pending status register for CPU interrupts	0x0008	RO
INTPRI_COREO_CPU_INT_PRI_0_REG	Priority configuration register for CPU interrupt 0	0x000C	R/W
INTPRI_COREO_CPU_INT_PRI_1_REG	Priority configuration register for CPU interrupt 1	0x0010	R/W
INTPRI_COREO_CPU_INT_PRI_2_REG	Priority configuration register for CPU interrupt 2	0x0014	R/W
INTPRI_COREO_CPU_INT_PRI_3_REG	Priority configuration register for CPU interrupt 3	0x0018	R/W
INTPRI_COREO_CPU_INT_PRI_4_REG	Priority configuration register for CPU interrupt 4	0x001C	R/W
INTPRI_COREO_CPU_INT_PRI_5_REG	Priority configuration register for CPU interrupt 5	0x0020	R/W
INTPRI_COREO_CPU_INT_PRI_6_REG	Priority configuration register for CPU interrupt 6	0x0024	R/W

Name	Description	Address	Access
INTPRI_COREO_CPU_INT_PRI_7_REG	Priority configuration register for CPU interrupt 7	0x0028	R/W
INTPRI_COREO_CPU_INT_PRI_8_REG	Priority configuration register for CPU interrupt 8	0x002C	R/W
INTPRI_COREO_CPU_INT_PRI_9_REG	Priority configuration register for CPU interrupt 9	0x0030	R/W
INTPRI_COREO_CPU_INT_PRI_10_REG	Priority configuration register for CPU interrupt 10	0x0034	R/W
INTPRI_COREO_CPU_INT_PRI_11_REG	Priority configuration register for CPU interrupt 11	0x0038	R/W
INTPRI_COREO_CPU_INT_PRI_12_REG	Priority configuration register for CPU interrupt 12	0x003C	R/W
INTPRI_COREO_CPU_INT_PRI_13_REG	Priority configuration register for CPU interrupt 13	0x0040	R/W
INTPRI_COREO_CPU_INT_PRI_14_REG	Priority configuration register for CPU interrupt 14	0x0044	R/W
INTPRI_COREO_CPU_INT_PRI_15_REG	Priority configuration register for CPU interrupt 15	0x0048	R/W
INTPRI_COREO_CPU_INT_PRI_16_REG	Priority configuration register for CPU interrupt 16	0x004C	R/W
INTPRI_COREO_CPU_INT_PRI_17_REG	Priority configuration register for CPU interrupt 17	0x0050	R/W
INTPRI_COREO_CPU_INT_PRI_18_REG	Priority configuration register for CPU interrupt 18	0x0054	R/W
INTPRI_COREO_CPU_INT_PRI_19_REG	Priority configuration register for CPU interrupt 19	0x0058	R/W
INTPRI_COREO_CPU_INT_PRI_20_REG	Priority configuration register for CPU interrupt 20	0x005C	R/W
INTPRI_COREO_CPU_INT_PRI_21_REG	Priority configuration register for CPU interrupt 21	0x0060	R/W
INTPRI_COREO_CPU_INT_PRI_22_REG	Priority configuration register for CPU interrupt 22	0x0064	R/W
INTPRI_COREO_CPU_INT_PRI_23_REG	Priority configuration register for CPU interrupt 23	0x0068	R/W
INTPRI_COREO_CPU_INT_PRI_24_REG	Priority configuration register for CPU interrupt 24	0x006C	R/W
INTPRI_COREO_CPU_INT_PRI_25_REG	Priority configuration register for CPU interrupt 25	0x0070	R/W
INTPRI_COREO_CPU_INT_PRI_26_REG	Priority configuration register for CPU interrupt 26	0x0074	R/W
INTPRI_COREO_CPU_INT_PRI_27_REG	Priority configuration register for CPU interrupt 27	0x0078	R/W
INTPRI_COREO_CPU_INT_PRI_28_REG	Priority configuration register for CPU interrupt 28	0x007C	R/W
INTPRI_COREO_CPU_INT_PRI_29_REG	Priority configuration register for CPU interrupt 29	0x0080	R/W
INTPRI_COREO_CPU_INT_PRI_30_REG	Priority configuration register for CPU interrupt 30	0x0084	R/W
INTPRI_COREO_CPU_INT_PRI_31_REG	Priority configuration register for CPU interrupt 31	0x0088	R/W
INTPRI_COREO_CPU_INT_THRESH_REG	Threshold configuration register for CPU interrupts	0x008C	R/W
INTPRI_COREO_CPU_INT_CLEAR_REG	CPU interrupt clear register	0x00A8	R/W
Interrupt Registers			
INTPRI_CPU_INTR_FROM_CPU_0_REG	CPU_INTR_FROM_CPU_0 mapping register	0x0090	R/W

Name	Description	Address	Access
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

9.7 Registers

9.7.1 Interrupt Matrix Registers

The addresses in this section are relative to the interrupt matrix base address provided in Table 4.3-2 in Chapter 4 *System and Memory*.

Register 9.1. INTMTX_CORE0_PMU_INTR_MAP_REG (0x0000)

Register 9.2. INTMTX_CORE0_EFUSE_INTR_MAP_REG (0x0004)

Register 9.3. INTMTX_CORE0_LP_RTC_TIMER_INTR_MAP_REG (0x0008)

Register 9.4. INTMTX_CORE0_LP_WDT_INTR_MAP_REG (0x0010)

Register 9.5. INTMTX_CORE0_LP_PERI_TIMEOUT_INTR_MAP_REG (0x0014)

Register 9.6. INTMTX_CORE0_LP_APM_M0_INTR_MAP_REG (0x0018)

Register 9.7. INTMTX_CORE0_CPU_INTR_FROM_CPU_0_MAP_REG (0x001C)

Register 9.8. INTMTX_CORE0_CPU_INTR_FROM_CPU_1_MAP_REG (0x0020)

Register 9.9. INTMTX_CORE0_CPU_INTR_FROM_CPU_2_MAP_REG (0x0024)

Register 9.10. INTMTX_CORE0_CPU_INTR_FROM_CPU_3_MAP_REG (0x0028)

Register 9.11. INTMTX_CORE0_ASSIST_DEBUG_INTR_MAP_REG (0x002C)

Register 9.12. INTMTX_CORE0_TRACE_INTR_MAP_REG (0x0030)

Register 9.13. INTMTX_CORE0_CPU_PERI_TIMEOUT_INTR_MAP_REG (0x0038)

Register 9.14. INTMTX_CORE0_GPIO_INTERRUPT_PRO_MAP_REG (0x0058)

Register 9.15. INTMTX_CORE0_PAU_INTR_MAP_REG (0x0060)

Register 9.16. INTMTX_CORE0_HP_PERI_TIMEOUT_INTR_MAP_REG (0x0064)

Register 9.17. INTMTX_CORE0_HP_APM_M0_INTR_MAP_REG (0x0068)

Register 9.18. INTMTX_CORE0_HP_APM_M1_INTR_MAP_REG (0x006C)

Register 9.19. INTMTX_CORE0_HP_APM_M2_INTR_MAP_REG (0x0070)

Register 9.20. INTMTX_CORE0_HP_APM_M3_INTR_MAP_REG (0x0074)

Register 9.21. INTMTX_CORE0_I2S_INTR_MAP_REG (0x007C)

Register 9.22. INTMTX_CORE0_UHCI0_INTR_MAP_REG (0x0080)

Register 9.23. INTMTX_CORE0_UART0_INTR_MAP_REG (0x0084)

Register 9.24. INTMTX_CORE0_UART1_INTR_MAP_REG (0x0088)

Register 9.25. INTMTX_CORE0_LEDC_INTR_MAP_REG (0x008C)

Register 9.26. INTMTX_CORE0_TWAI0_INTR_MAP_REG (0x0090)

Register 9.27. INTMTX_CORE0_USB_INTR_MAP_REG (0x0094)

Register 9.28. INTMTX_CORE0_RMT_INTR_MAP_REG (0x0098)

Register 9.29. INTMTX_CORE0_I2C_EXT0_INTR_MAP_REG (0x009C)

Register 9.30. INTMTX_CORE0_I2C_EXT1_INTR_MAP_REG (0x00A0)

Register 9.31. INTMTX_COREO_TGO_TO_INTR_MAP_REG (0x00A4)

Register 9.32. INTMTX_COREO_TGO_WDT_INTR_MAP_REG (0x00A8)

Register 9.33. INTMTX_COREO_TG1_TO_INTR_MAP_REG (0x00AC)

Register 9.34. INTMTX_COREO_TG1_WDT_INTR_MAP_REG (0x00B0)

Register 9.35. INTMTX_COREO_SYSTIMER_TARGETO_INTR_MAP_REG (0x00B4)

Register 9.36. INTMTX_COREO_SYSTIMER_TARGET1_INTR_MAP_REG (0x00B8)

Register 9.37. INTMTX_COREO_SYSTIMER_TARGET2_INTR_MAP_REG (0x00BC)

Register 9.38. INTMTX_COREO_APB_ADC_INTR_MAP_REG (0x00C0)

Register 9.39. INTMTX_COREO_PWM_INTR_MAP_REG (0x00C4)

Register 9.40. INTMTX_COREO_PCNT_INTR_MAP_REG (0x00C8)

Register 9.41. INTMTX_COREO_PARL_IO_TX_INTR_MAP_REG (0x00CC)

Register 9.42. INTMTX_COREO_PARL_IO_RX_INTR_MAP_REG (0x00D0)

Register 9.43. INTMTX_COREO_DMA_IN_CHO_INTR_MAP_REG (0x00D4)

Register 9.44. INTMTX_COREO_DMA_IN_CH1_INTR_MAP_REG (0x00D8)

Register 9.45. INTMTX_COREO_DMA_IN_CH2_INTR_MAP_REG (0x00DC)

Register 9.46. INTMTX_COREO_DMA_OUT_CHO_INTR_MAP_REG (0x00E0)

Register 9.47. INTMTX_COREO_DMA_OUT_CH1_INTR_MAP_REG (0x00E4)

Register 9.48. INTMTX_COREO_DMA_OUT_CH2_INTR_MAP_REG (0x00E8)

Register 9.49. INTMTX_COREO_GPSPI2_INTR_MAP_REG (0x00EC)

Register 9.50. INTMTX_COREO_AES_INTR_MAP_REG (0x00F0)

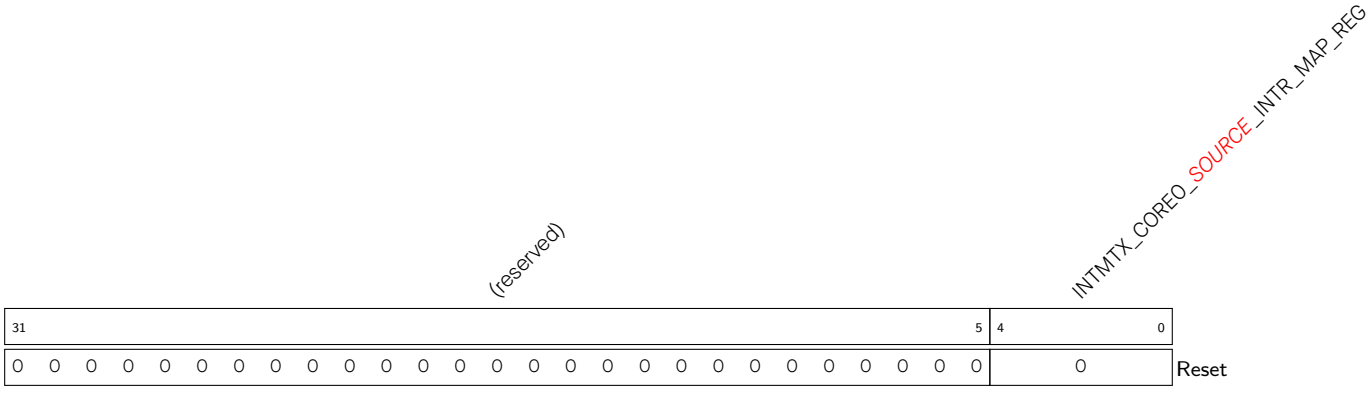
Register 9.51. INTMTX_COREO_SHA_INTR_MAP_REG (0x00F4)

Register 9.52. INTMTX_COREO_RSA_INTR_MAP_REG (0x00F8)

Register 9.53. INTMTX_COREO_ECC_INTR_MAP_REG (0x00FC)

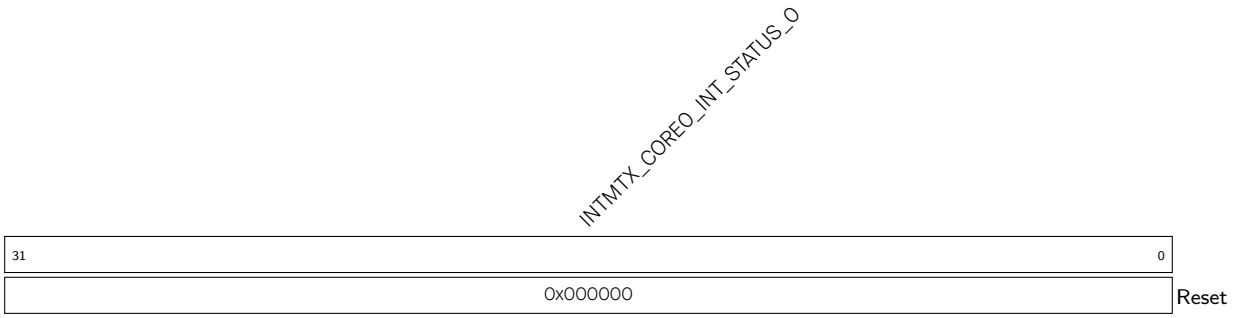
Register 9.54. INTMTX_COREO_ECDSA_INTR_MAP_REG (0x0100)

Register 9.55. INTMTX_COREO_*SOURCE*_INTR_MAP_REG (0x0000 - 0x0100)



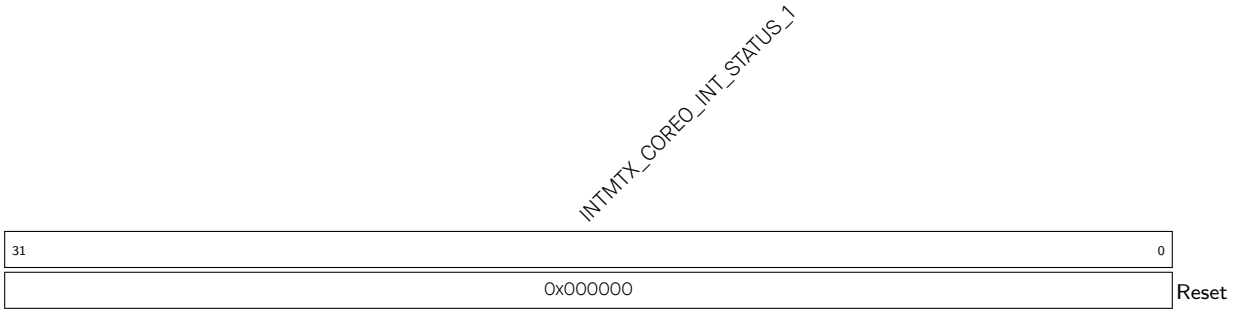
INTMTX_COREO_*SOURCE*_INTR_MAP Map the interrupt source (*SOURCE*) into one CPU interrupt. For the information of *SOURCE*, see Table 9.5-1. (R/W)

Register 9.56. INTMTX_COREO_INT_STATUS_0_REG (0x0104)



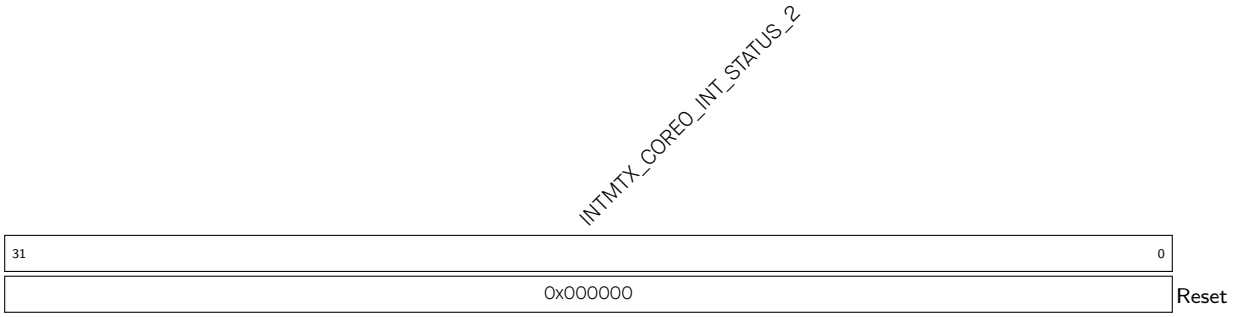
INTMTX_COREO_INT_STATUS_0 Status register for interrupt sources 0 ~ 31. (RO)

Register 9.57. INTMTX_COREO_INT_STATUS_1_REG (0x0108)



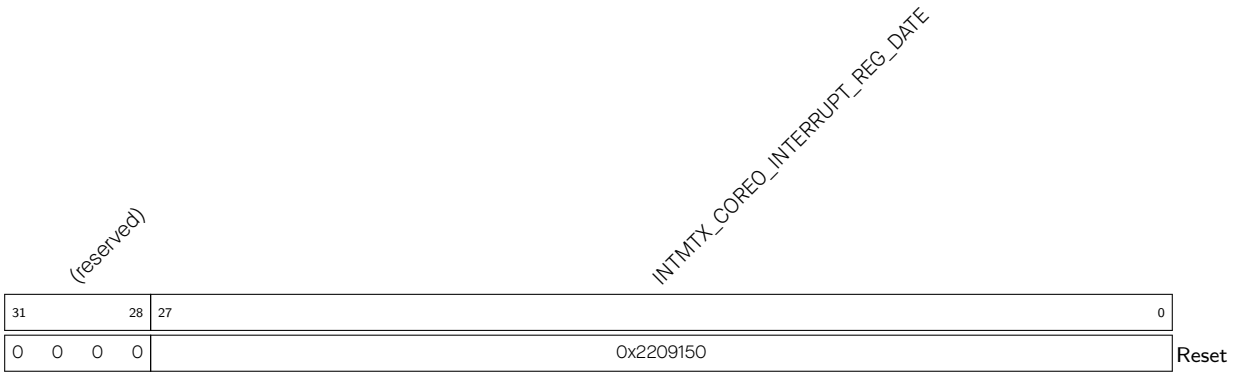
INTMTX_COREO_INT_STATUS_1 Status register for interrupt sources 32 ~ 63. (RO)

Register 9.58. INTMTX_COREO_INT_STATUS_2_REG (0x010C)



INTMTX_COREO_INT_STATUS_2 Status register for interrupt source 64. (RO)

Register 9.59. INTMTX_COREO_INTERRUPT_REG_DATE_REG (0x07FC)

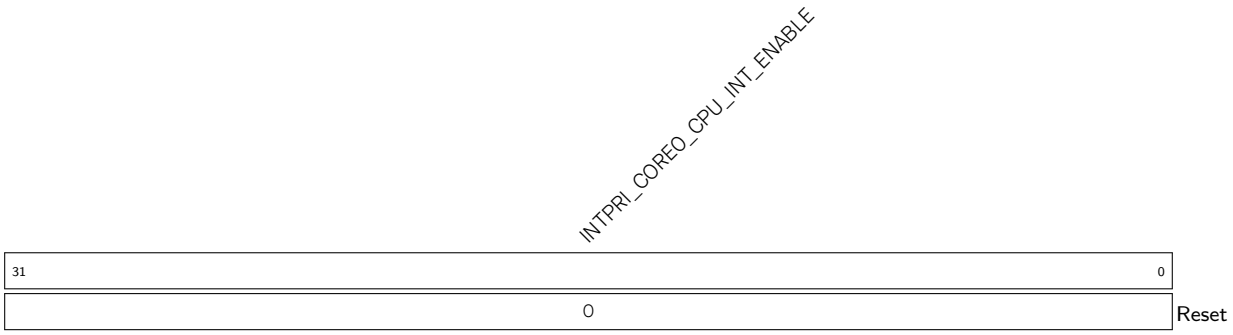


INTMTX_COREO_INTERRUPT_REG_DATE Version control register (R/W)

9.7.2 Interrupt Priority Registers

The addresses in this section are relative to the interrupt priority base address provided in Table 4.3-2 in Chapter 4 *System and Memory*.

Register 9.60. INTPRI_COREO_CPU_INT_ENABLE_REG (0x0000)



INTPRI_COREO_CPU_INT_ENABLE Configures whether to enable the corresponding CPU interrupt.

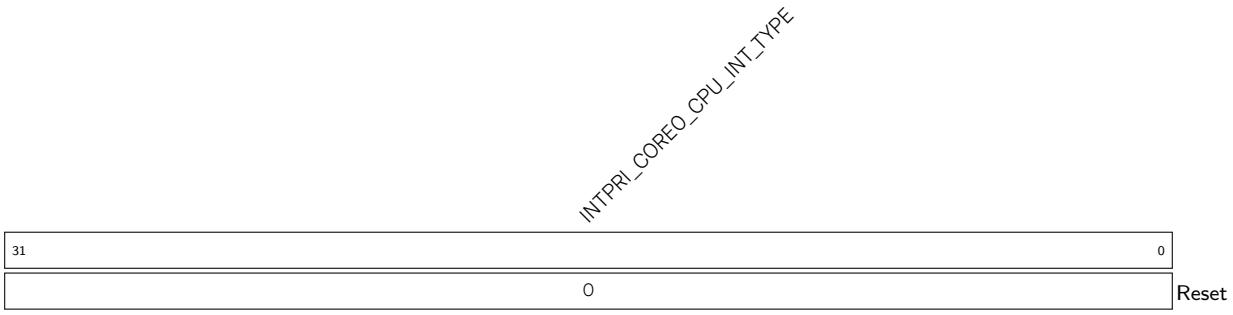
0: Not enable

1: Enable

For more information about how to use this register, see Chapter 1 *ESP-RISC-V CPU* > Section 1.6*Interrupt Controller*.

(R/W)

Register 9.61. INTPRI_COREO_CPU_INT_TYPE_REG (0x0004)



INTPRI_COREO_CPU_INT_TYPE Configures CPU interrupt type.

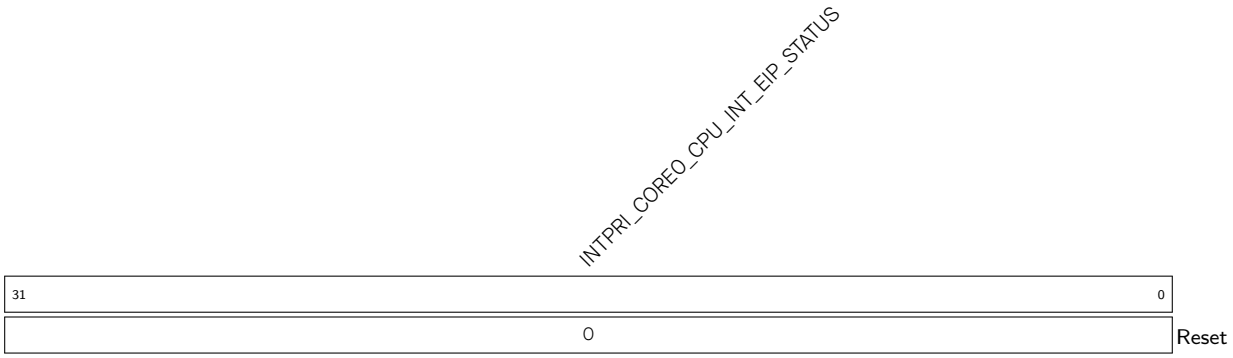
0: Level-triggered

1: Edge-triggered

For more information about how to use this register, see Chapter 1 *ESP-RISC-V CPU* > Section 1.6*Interrupt Controller*.

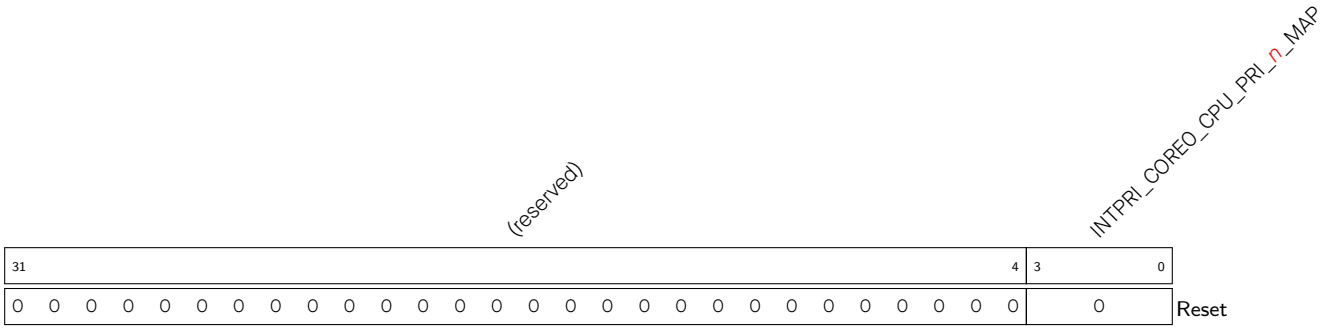
(R/W)

Register 9.62. INTPRI_COREO_CPU_INT_EIP_STATUS_REG (0x0008)



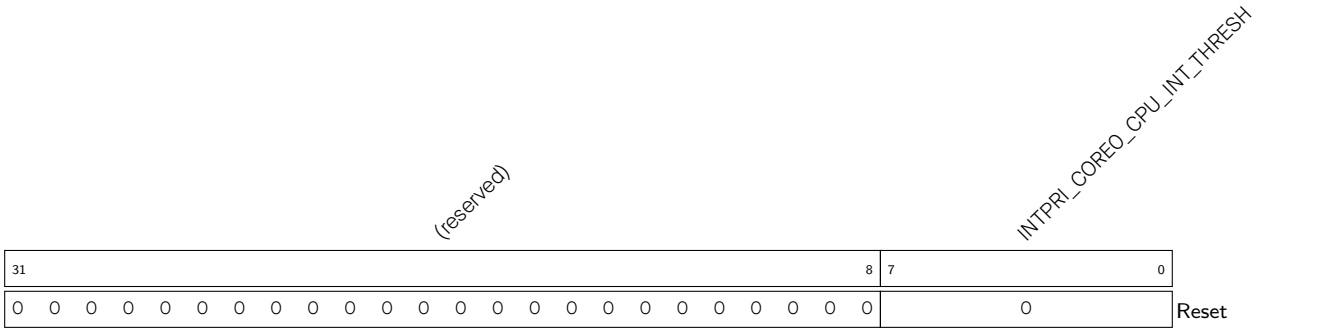
INTPRI_COREO_CPU_INT_EIP_STATUS Represents the pending status of CPU interrupts. For more information about how to use this register, see Chapter 1 *ESP-RISC-V CPU* > Section 1.6*Interrupt Controller*. (RO)

Register 9.63. INTPRI_COREO_CPU_INT_PRI_*n*_REG (*n*: 0-31)(0x000C+0x4**n*)



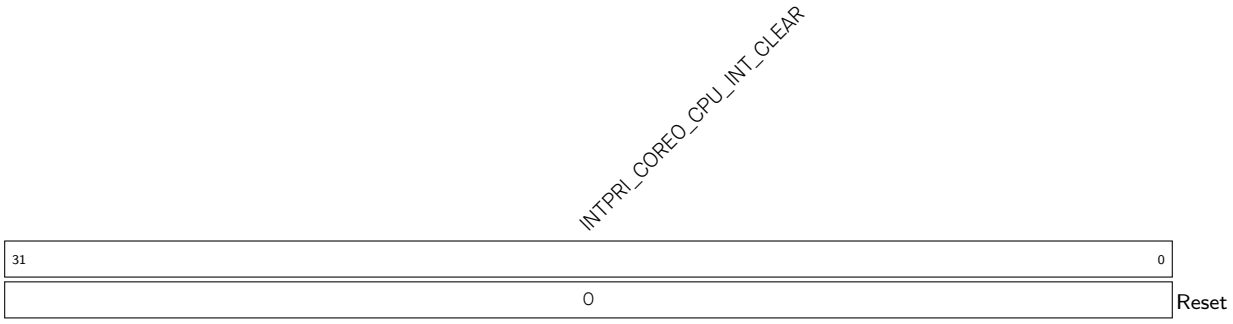
INTPRI_COREO_CPU_PRI_*n*_MAP Configures the priority for CPU interrupt *n*. The priority here can be 1 (lowest) ~ 15 (highest). For more information about how to use this register, see Chapter 1 *ESP-RISC-V CPU* > Section 1.6*Interrupt Controller*. (R/W)

Register 9.64. INTPRI_COREO_CPU_INT_THRESH_REG (0x008C)



INTPRI_COREO_CPU_INT_THRESH Configures the threshold for interrupt assertion to CPU. Only when the interrupt priority is equal to or higher than this threshold, CPU will respond to this interrupt. For more information about how to use this register, see Chapter 1 *ESP-RISC-V CPU* > Section 1.6*Interrupt Controller*. (R/W)

Register 9.65. INTPRI_COREO_CPU_INT_CLEAR_REG (0x00A8)



INTPRI_COREO_CPU_INT_CLEAR Configures whether to clear the corresponding CPU interrupt.

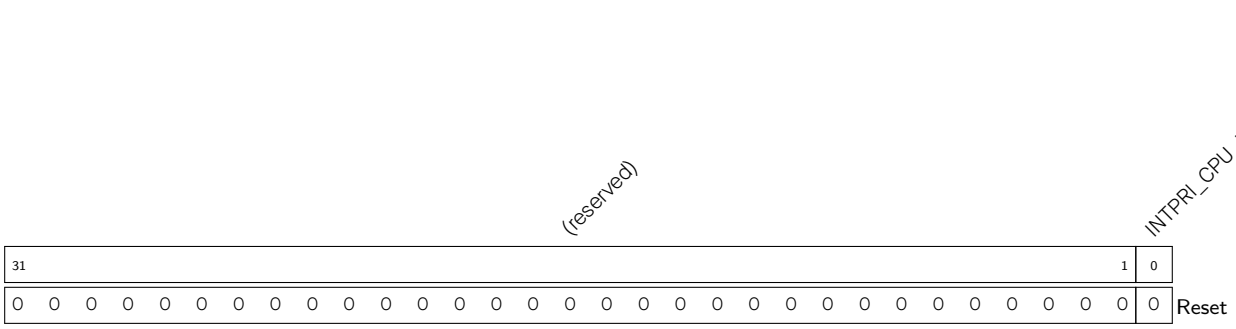
0: No effect

1: Clear

For more information about how to use this register, see Chapter 1 *ESP-RISC-V CPU* > Section 1.6*Interrupt Controller*.

(R/W)

Register 9.66. INTPRI_CPU_INTR_FROM_CPU_*n*_REG (*n*: 0-3)(0x0090+0x4**n*)



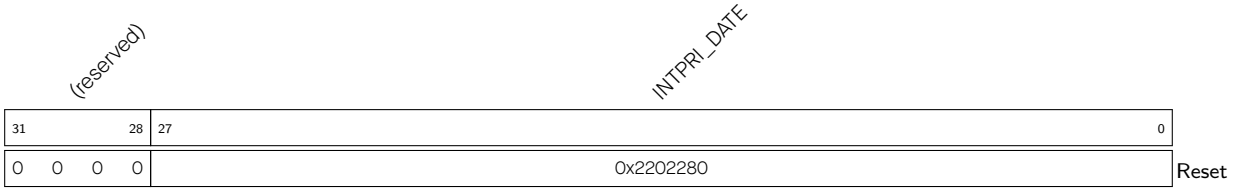
INTPRI_CPU_INTR_FROM_CPU_*n* Configures whether to generate interrupts from CPU_INTR_FROM_CPU_*n*.

0: Do not generate interrupts

1: Generate interrupts

(R/W)

Register 9.67. INTPRI_DATE_REG (0x00A0)



INTPRI_DATE Version control register. (R/W)

Chapter 10

Event Task Matrix (SOC_ETM)

10.1 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.

10.2 Features

The Event Task Matrix has the following features:

- Receive 122 various events from multiple peripherals
- Generate 113 various tasks for multiple peripherals
- 50 independently configurable ETM channels
- An ETM channel can be set up to receive any event, and map it to any task
- Each ETM channel can be enabled independently. If not enabled, the channel will not respond to the configured event and generate the task mapped to that event
- Peripherals supporting ETM include GPIO, LED PWM, general-purpose timers, RTC Timer, system timer, MCPWM, 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).

10.3 Functional Description

10.3.1 Architecture

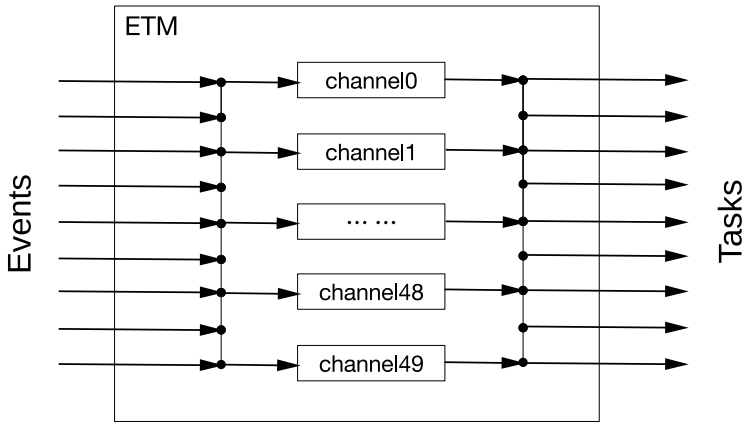


Figure 10.3-1. Event Task Matrix Architecture

Figure 10.3-1 shows the architecture of the Event Task Matrix.

The Event Task Matrix has 50 independent channels. A channel can choose any event as input, and map the event to any task as output (For configuration procedures, refer to Section 10.3.2 and Section 10.3.3 respectively). Each channel has an individual enable bit (For configuration procedures, refer to Section 10.3.5).

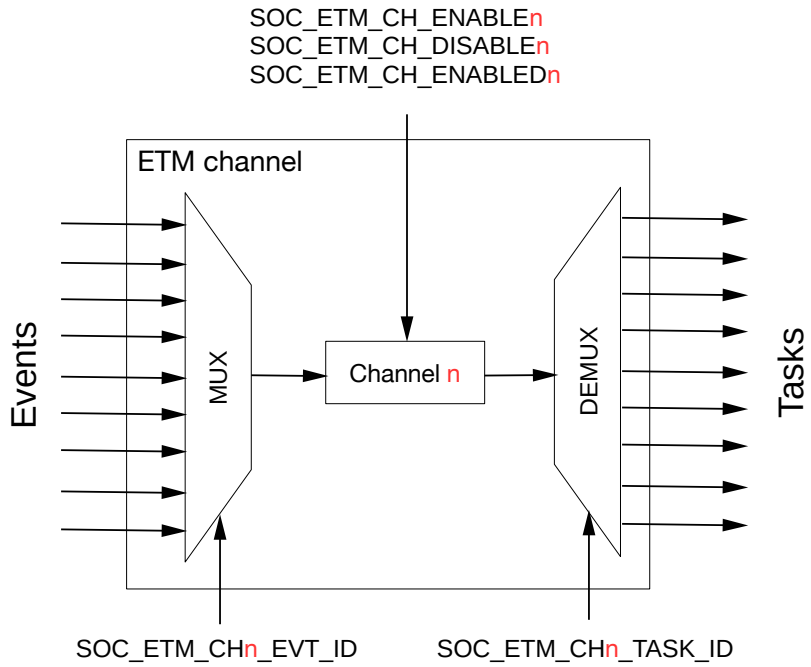


Figure 10.3-2. ETM Channel*n* Architecture

Figure 10.3-2 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*.

10.3.2 Events

An ETM channel can be set up to choose which event to receive by configuring the SOC_ETM_CH*n*_EVT_ID field. Table 10.3-1 shows the configuration values of SOC_ETM_CH*n*_EVT_ID and their corresponding events.

Table 10.3-1. 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	
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	LEDC_EVT_DUTY_CHNG_END_CH0	LED PWM Controller (LEDC)
26	LEDC_EVT_DUTY_CHNG_END_CH1	
27	LEDC_EVT_DUTY_CHNG_END_CH2	
28	LEDC_EVT_DUTY_CHNG_END_CH3	
29	LEDC_EVT_DUTY_CHNG_END_CH4	
30	LEDC_EVT_DUTY_CHNG_END_CH5	
31	LEDC_EVT_OVF_CNT_PLS_CH0	
32	LEDC_EVT_OVF_CNT_PLS_CH1	
33	LEDC_EVT_OVF_CNT_PLS_CH2	
34	LEDC_EVT_OVF_CNT_PLS_CH3	
35	LEDC_EVT_OVF_CNT_PLS_CH4	

SOC_ETM_CH*n*_EVT_ID	Selected Event	Peripheral Generating This Event
36	LEDC_EVT_OVF_CNT_PLS_CH5	
37	LEDC_EVT_TIME_OVF_TIMER0	
38	LEDC_EVT_TIME_OVF_TIMER1	
39	LEDC_EVT_TIME_OVF_TIMER2	
40	LEDC_EVT_TIME_OVF_TIMER3	
41	LEDC_EVT_TIMER0_CMP	
42	LEDC_EVT_TIMER1_CMP	
43	LEDC_EVT_TIMER2_CMP	
44	LEDC_EVT_TIMER3_CMP	
48	TIMER0_EVT_CNT_CMP_TIMER0	General-purpose timer 0
49	TIMER1_EVT_CNT_CMP_TIMER0	General-purpose timer 1
50	SYSTIMER_EVT_CNT_CMP0	System Timer (SYSTIMER)
51	SYSTIMER_EVT_CNT_CMP1	
52	SYSTIMER_EVT_CNT_CMP2	
58	MCPWM_EVT_TIMER0_STOP	Motor Control PWM (MCPWM)
59	MCPWM_EVT_TIMER1_STOP	
60	MCPWM_EVT_TIMER2_STOP	
61	MCPWM_EVT_TIMER0_TEZ	
62	MCPWM_EVT_TIMER1_TEZ	
63	MCPWM_EVT_TIMER2_TEZ	
64	MCPWM_EVT_TIMER0_TEP	
65	MCPWM_EVT_TIMER1_TEP	
66	MCPWM_EVT_TIMER2_TEP	
67	MCPWM_EVT_OP0_TEA	
68	MCPWM_EVT_OP1_TEA	
69	MCPWM_EVT_OP2_TEA	
70	MCPWM_EVT_OP0_TEB	
71	MCPWM_EVT_OP1_TEB	
72	MCPWM_EVT_OP2_TEB	
73	MCPWM_EVT_F0	
74	MCPWM_EVT_F1	
75	MCPWM_EVT_F2	
76	MCPWM_EVT_F0_CLR	
77	MCPWM_EVT_F1_CLR	
78	MCPWM_EVT_F2_CLR	
79	MCPWM_EVT_TZ0_CBC	
80	MCPWM_EVT_TZ1_CBC	
81	MCPWM_EVT_TZ2_CBC	
82	MCPWM_EVT_TZ0_OST	
83	MCPWM_EVT_TZ1_OST	
84	MCPWM_EVT_TZ2_OST	
85	MCPWM_EVT_CAP0	
86	MCPWM_EVT_CAP1	
87	MCPWM_EVT_CAP2	
88	ADC_EVT_CONV_CMPLT0	ADC

SOC_ETM_CH*n*_EVT_ID	Selected Event	Peripheral Generating This Event
89	ADC_EVT_EQ_ABOVE_THRESH0	
90	ADC_EVT_EQ_ABOVE_THRESH1	
91	ADC_EVT_EQ_BELOW_THRESH0	
92	ADC_EVT_EQ_BELOW_THRESH1	
94	ADC_EVT_STOPPED0	
95	ADC_EVT_STARTED0	
110	TMPSNSR_EVT_OVER_LIMIT	Temperature Sensor
126	I2S_EVT_RX_DONE	I2S Controller (I2S)
127	I2S_EVT_TX_DONE	
128	I2S_EVT_X_WORDS_RECEIVED	
129	I2S_EVT_X_WORDS_SENT	
135	RTC_EVT_TICK	RTC Timer
136	RTC_EVT_OVF	
137	RTC_EVT_CMP	
138	GDMA_EVT_IN_DONE_CH0	GDMA Controller (GDMA)
139	GDMA_EVT_IN_DONE_CH1	
140	GDMA_EVT_IN_DONE_CH2	
141	GDMA_EVT_IN_SUC_EOF_CH0	
142	GDMA_EVT_IN_SUC_EOF_CH1	
143	GDMA_EVT_IN_SUC_EOF_CH2	
144	GDMA_EVT_IN_FIFO_EMPTY_CH0	
145	GDMA_EVT_IN_FIFO_EMPTY_CH1	
146	GDMA_EVT_IN_FIFO_EMPTY_CH2	
147	GDMA_EVT_IN_FIFO_FULL_CH0	
148	GDMA_EVT_IN_FIFO_FULL_CH1	
149	GDMA_EVT_IN_FIFO_FULL_CH2	
150	GDMA_EVT_OUT_DONE_CH0	
151	GDMA_EVT_OUT_DONE_CH1	
152	GDMA_EVT_OUT_DONE_CH2	
153	GDMA_EVT_OUT_EOF_CH0	
154	GDMA_EVT_OUT_EOF_CH1	
155	GDMA_EVT_OUT_EOF_CH2	
156	GDMA_EVT_OUT_TOTAL_EOF_CH0	
157	GDMA_EVT_OUT_TOTAL_EOF_CH1	
158	GDMA_EVT_OUT_TOTAL_EOF_CH2	
159	GDMA_EVT_OUT_FIFO_EMPTY_CH0	
160	GDMA_EVT_OUT_FIFO_EMPTY_CH1	
161	GDMA_EVT_OUT_FIFO_EMPTY_CH2	
162	GDMA_EVT_OUT_FIFO_FULL_CH0	
163	GDMA_EVT_OUT_FIFO_FULL_CH1	
164	GDMA_EVT_OUT_FIFO_FULL_CH2	
165	PMU_EVT_SLEEP_WEEKUP	PMU

Each event corresponds to a pulse signal generated by the corresponding peripheral. When the pulse signal

is valid, the corresponding event is considered as received.

For more detailed descriptions of an event, please refer to the chapter for the peripheral generating this event.

10.3.3 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. Table10.3-2 shows the configuration values of SOC_ETM_CH*n*_TASK_ID and their corresponding tasks.

Table 10.3-2. 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	
31	LEDC_TASK_DUTY_SCALE_UPDATE_CH0	
32	LEDC_TASK_DUTY_SCALE_UPDATE_CH1	
33	LEDC_TASK_DUTY_SCALE_UPDATE_CH2	
34	LEDC_TASK_DUTY_SCALE_UPDATE_CH3	
35	LEDC_TASK_DUTY_SCALE_UPDATE_CH4	

SOC_ETM_CH*n*_TASK_ID	Mapped Task	Peripheral Receiving This Task
36	LEDC_TASK_DUTY_SCALE_UPDATE_CH5	
37	LEDC_TASK_TIMER0_CAP	
38	LEDC_TASK_TIMER1_CAP	
39	LEDC_TASK_TIMER2_CAP	
40	LEDC_TASK_TIMER3_CAP	
41	LEDC_TASK_SIG_OUT_DIS_CH0	
42	LEDC_TASK_SIG_OUT_DIS_CH1	
43	LEDC_TASK_SIG_OUT_DIS_CH2	
44	LEDC_TASK_SIG_OUT_DIS_CH3	
45	LEDC_TASK_SIG_OUT_DIS_CH4	
46	LEDC_TASK_SIG_OUT_DIS_CH5	
47	LEDC_TASK_OVF_CNT_RST_CH0	
48	LEDC_TASK_OVF_CNT_RST_CH1	
49	LEDC_TASK_OVF_CNT_RST_CH2	
50	LEDC_TASK_OVF_CNT_RST_CH3	
51	LEDC_TASK_OVF_CNT_RST_CH4	
52	LEDC_TASK_OVF_CNT_RST_CH5	
53	LEDC_TASK_TIMER0_RST	
54	LEDC_TASK_TIMER1_RST	
55	LEDC_TASK_TIMER2_RST	
56	LEDC_TASK_TIMER3_RST	
57	LEDC_TASK_TIMER0_RESUME	
58	LEDC_TASK_TIMER1_RESUME	
59	LEDC_TASK_TIMER2_RESUME	
60	LEDC_TASK_TIMER3_RESUME	
61	LEDC_TASK_TIMER0_PAUSE	
62	LEDC_TASK_TIMER1_PAUSE	
63	LEDC_TASK_TIMER2_PAUSE	
64	LEDC_TASK_TIMER3_PAUSE	
65	LEDC_TASK_GAMMA_RESTART_CH0	
66	LEDC_TASK_GAMMA_RESTART_CH1	
67	LEDC_TASK_GAMMA_RESTART_CH2	
68	LEDC_TASK_GAMMA_RESTART_CH3	
69	LEDC_TASK_GAMMA_RESTART_CH4	
70	LEDC_TASK_GAMMA_RESTART_CH5	
71	LEDC_TASK_GAMMA_PAUSE_CH0	
72	LEDC_TASK_GAMMA_PAUSE_CH1	
73	LEDC_TASK_GAMMA_PAUSE_CH2	
74	LEDC_TASK_GAMMA_PAUSE_CH3	
75	LEDC_TASK_GAMMA_PAUSE_CH4	
76	LEDC_TASK_GAMMA_PAUSE_CH5	
77	LEDC_TASK_GAMMA_RESUME_CH0	
78	LEDC_TASK_GAMMA_RESUME_CH1	
79	LEDC_TASK_GAMMA_RESUME_CH2	
80	LEDC_TASK_GAMMA_RESUME_CH3	

SOC_ETM_CH*n*_TASK_ID	Mapped Task	Peripheral Receiving This Task
81	LEDC_TASK_GAMMA_RESUME_CH4	
82	LEDC_TASK_GAMMA_RESUME_CH5	
88	TIMER0_TASK_CNT_START_TIMER0	General-purpose timer 0
90	TIMER0_TASK_ALARM_START_TIMER0	
92	TIMER0_TASK_CNT_STOP_TIMER0	
94	TIMER0_TASK_CNT_RELOAD_TIMER0	
96	TIMER0_TASK_CNT_CAP_TIMER0	
89	TIMER1_TASK_CNT_START_TIMER0	General-purpose timer 1
91	TIMER1_TASK_ALARM_START_TIMER0	
93	TIMER1_TASK_CNT_STOP_TIMER0	
95	TIMER1_TASK_CNT_RELOAD_TIMER0	
97	TIMER1_TASK_CNT_CAP_TIMER0	
102	MCPWM_TASK_CMPR0_A_UP	Motor Control PWM (MCPWM)
103	MCPWM_TASK_CMPR1_A_UP	
104	MCPWM_TASK_CMPR2_A_UP	
105	MCPWM_TASK_CMPR0_B_UP	
106	MCPWM_TASK_CMPR1_B_UP	
107	MCPWM_TASK_CMPR2_B_UP	
108	MCPWM_TASK_GEN_STOP	
109	MCPWM_TASK_TIMER0_SYN	
110	MCPWM_TASK_TIMER1_SYN	
111	MCPWM_TASK_TIMER2_SYN	
112	MCPWM_TASK_TIMER0_PERIOD_UP	
113	MCPWM_TASK_TIMER1_PERIOD_UP	
114	MCPWM_TASK_TIMER2_PERIOD_UP	
115	MCPWM_TASK_TZ0_OST	
116	MCPWM_TASK_TZ1_OST	
117	MCPWM_TASK_TZ2_OST	
118	MCPWM_TASK_CLR0_OST	
119	MCPWM_TASK_CLR1_OST	
120	MCPWM_TASK_CLR2_OST	
121	MCPWM_TASK_CAP0	
122	MCPWM_TASK_CAP1	
123	MCPWM_TASK_CAP2	
124	ADC_TASK_SAMPLE0	ADC
126	ADC_TASK_START0	
127	ADC_TASK_STOP0	
135	TMPSNSR_TASK_START_SAMPLE	Temperature Sensor
136	TMPSNSR_TASK_STOP_SAMPLE	
148	I2S_TASK_START_RX	I2S Controller (I2S)
149	I2S_TASK_START_TX	
150	I2S_TASK_STOP_RX	
151	I2S_TASK_STOP_TX	
159	GDMA_TASK_IN_START_CH0	GDMA Controller (GDMA)
160	GDMA_TASK_IN_START_CH1	

SOC_ETM_CH*n*_TASK_ID	Mapped Task	Peripheral Receiving This Task
161	GDMA_TASK_IN_START_CH2	
162	GDMA_TASK_OUT_START_CH0	
163	GDMA_TASK_OUT_START_CH1	
164	GDMA_TASK_OUT_START_CH2	
165	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 is generated. If events received by multiple channels are valid at the same time, the task will be generated only once.

10.3.4 Timing Considerations

Figure 10.3-3 shows the structure of clocks that drive received events, sent tasks, and ETM channels.

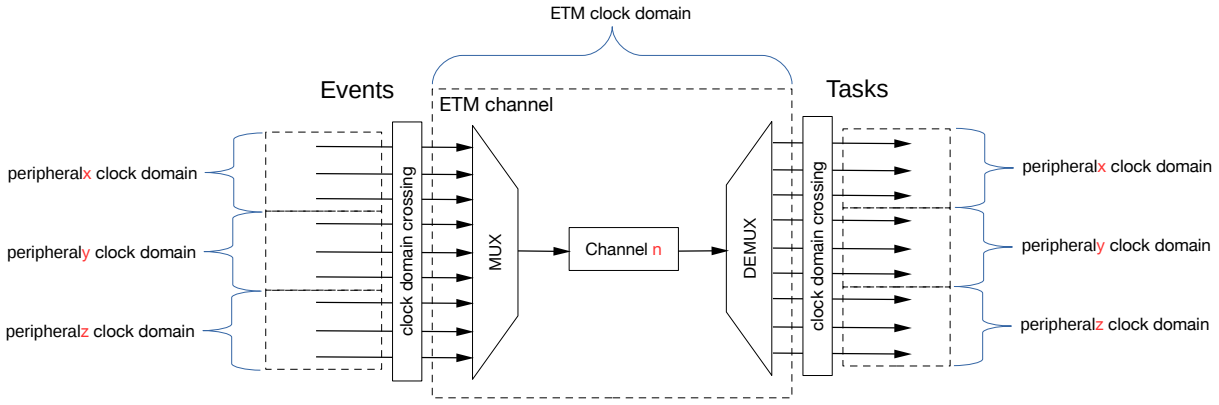


Figure 10.3-3. Event Task Matrix Clock Architecture

ETM is running at the AHB_CLK domain (see Chapter 7 *Reset and Clock*). Each event corresponds to a pulse signal generated by the corresponding peripheral in its clock domain, while each task is mapped by the ETM to a pulse signal under its corresponding peripheral clock domain. The peripherals generating events, the Event Task Matrix, and peripherals receiving tasks are not necessarily running off the same clock and as such need to be synchronized. Therefore, there must be a minimum interval between two consecutive events to avoid event loss: 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 event 1 generated by peripheral A is in the 96 MHz clock domain (PLL_F96M_CLK), and the ETM runs in the 32 MHz clock domain (AHB_CLK). To receive each event 1

successfully, the interval between two consecutive event 1 must be greater than three peripheral A clock cycles (i.e., one ETM clock cycle).

Likewise, to make sure the Event Task Matrix maps the received event (i.e., event synchronized to the ETM's clock domain) successfully to a task, the interval between two consecutive event pulses in the ETM clock domain must be greater than one peripheral clock cycle, namely $ceil(\frac{ETM\_clock\_frequency}{peripheral\_clock\_frequency})$ in the unit of ETM clock cycles.

For example, assuming that task 1 received by peripheral B is in the 8 MHz clock domain (RC_FAST_CLK), and the ETM runs in the 32 MHz clock domain (AHB_CLK). To map each received event successfully to task 1, the interval between two consecutive events must be greater than four 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 event 1 generated by peripheral A is in the 96 MHz clock domain (PLL_F96M_CLK), task 1 received by peripheral B is in the 8 MHz clock domain (RC_FAST_CLK), and the ETM runs in the 32 MHz clock domain (AHB_CLK). To successfully map each event 1 (generated by peripheral A) to task 1 (received by peripheral B), the interval between two consecutive event 1 must be greater than $3 * 4 = 12$ peripheral A clock cycles.

10.3.5 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 configured via SOC_ETM_CH*n*_EVT_ID, no task will be generated.

To enable ETM channel*n*:

1. Write 1 to SOC_ETM_CH_ENABLE*n*
2. Read SOC_ETM_CH_ENABLED*n*. 1 indicates that channel*n* has been enabled, and 0 indicates disabled

To disable ETM channel*n*:

1. Write 1 to SOC_ETM_CH_DISABLE*n*
2. Read SOC_ETM_CH_ENABLED*n*. 0 indicates that channel*n* is disabled, and 1 indicates enabled

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. Write 1 to field SOC_ETM_CH_ENABLE*n*

5. When channel*n* no longer needs to map the selected event to the selected task, disable channel*n* by setting SOC_ETM_CH_DISABLE*n*. To configure a new event and task mapping, repeat Steps 2 to 4. If no configurations, channel*n* will remain disabled
6. The ETM module can be reset by writing 1 and then 0 to the PCR_ETM_RST_EN field. Reset is finished when PCR_ETM_READY becomes 1

10.4 Register Summary

The addresses in this section are relative to ETM base address provided in Table 4.3-2 in Chapter *4 System and Memory*.

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

Name	Description	Address	Access
Configuration Register			
SOC_ETM_CH_ENA_AD0_REG	Channel status register	0x0000	R/WTC/SS
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_REG	Channel status register	0x000C	R/WTC/SS
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
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

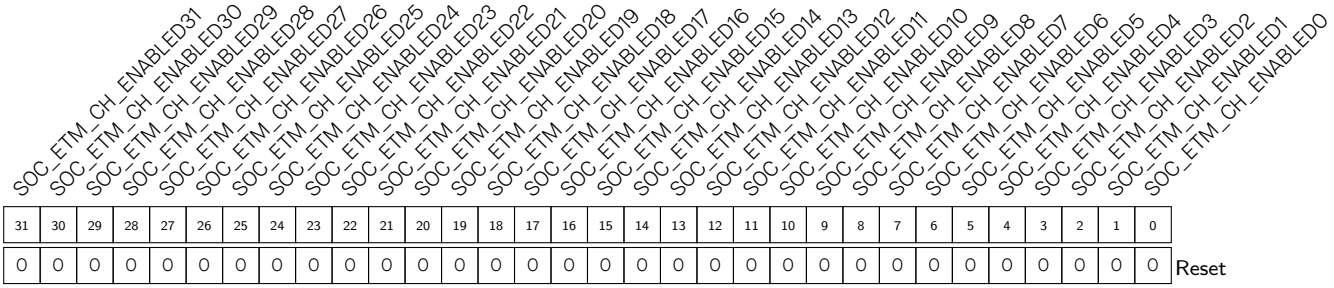
Name	Description	Address	Access
SOC_ETM_CH15_EVT_ID_REG	Channel15 event ID register	0x0090	R/W
SOC_ETM_CH15_TASK_ID_REG	Channel15 task ID register	0x0094	R/W
SOC_ETM_CH16_EVT_ID_REG	Channel16 event ID register	0x0098	R/W
SOC_ETM_CH16_TASK_ID_REG	Channel16 task ID register	0x009C	R/W
SOC_ETM_CH17_EVT_ID_REG	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
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

Name	Description	Address	Access
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_CLK_EN_REG	ETM clock enable register	0x01A8	R/W
Version Register			
SOC_ETM_DATE_REG	Version control register	0x01AC	R/W

10.5 Registers

The addresses in this section are relative to ETM base address provided in Table 4.3-2 in Chapter 4 *System and Memory*.

Register 10.1. SOC_ETM_CH_ENA_AD0_REG (0x0000)



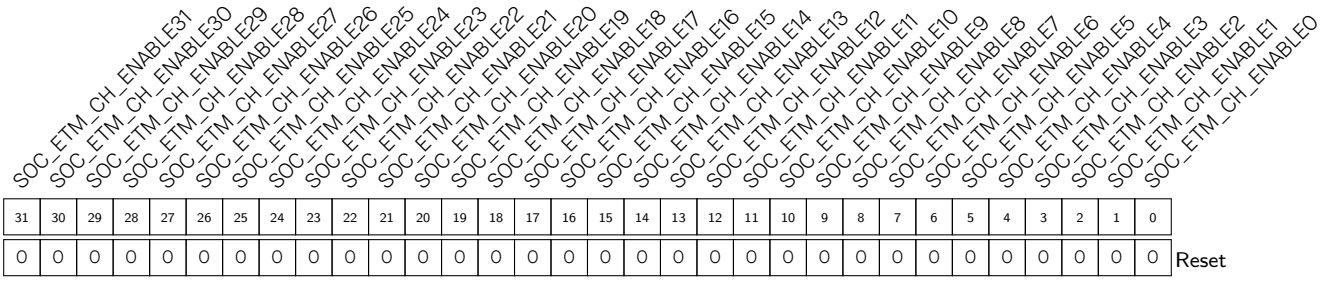
SOC_ETM_CH_ENABLED*n* (*n*: 0-31) Represents the status of channel*n*.

0: Disabled

1: Enabled

(R/WTC/SS)

Register 10.2. SOC_ETM_CH_ENA_AD0_SET_REG (0x0004)



SOC_ETM_CH_ENABLE*n* (*n*: 0-31) Configures whether to enable channel*n*.

0: Invalid. No effect

1: Enable

(WT)

Register 10.3. SOC_ETM_CH_ENA_AD0_CLR_REG (0x0008)

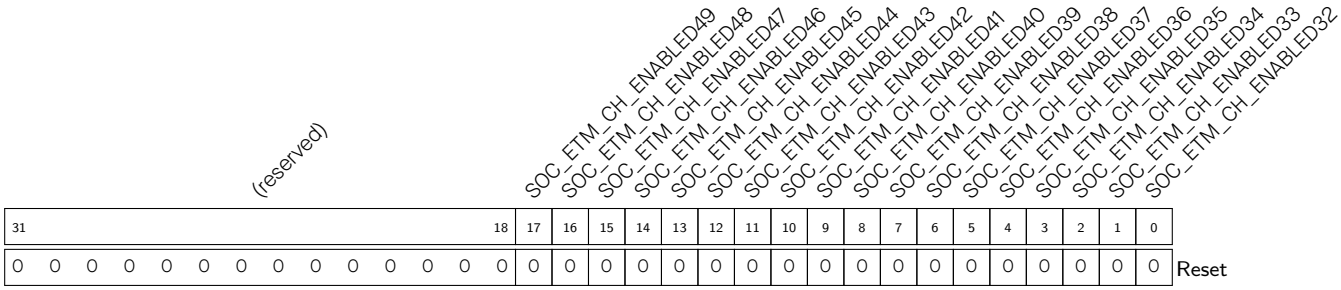
SOC_ETM_CH_DISABLE*n* (*n*: 0-31) Configures whether to disable channel 0.

0: Invalid. No effect

1: Disable

(WT)

Register 10.4. SOC_ETM_CH_ENA_AD1_REG (0x000C)



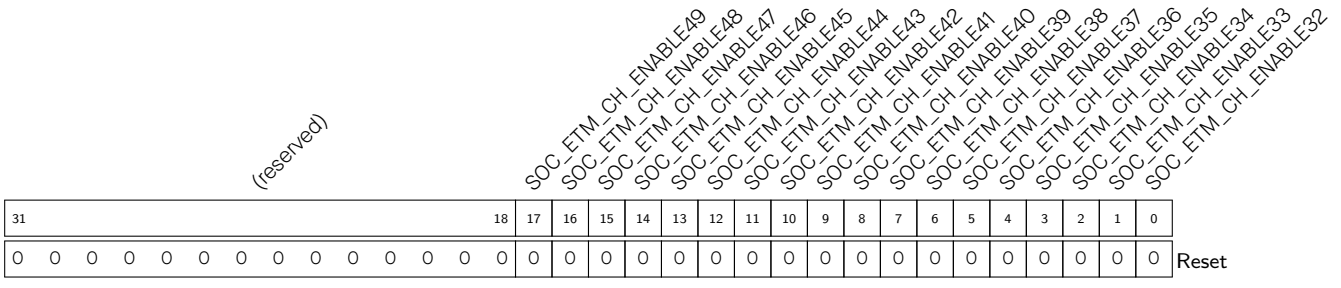
SOC_ETM_CH_ENABLED*n* (*n*: 32-49) Represents the status of channel*n*.

0: Disabled

1: Enabled

(R/WTC/SS)

Register 10.5. SOC_ETM_CH_ENA_AD1_SET_REG (0x0010)



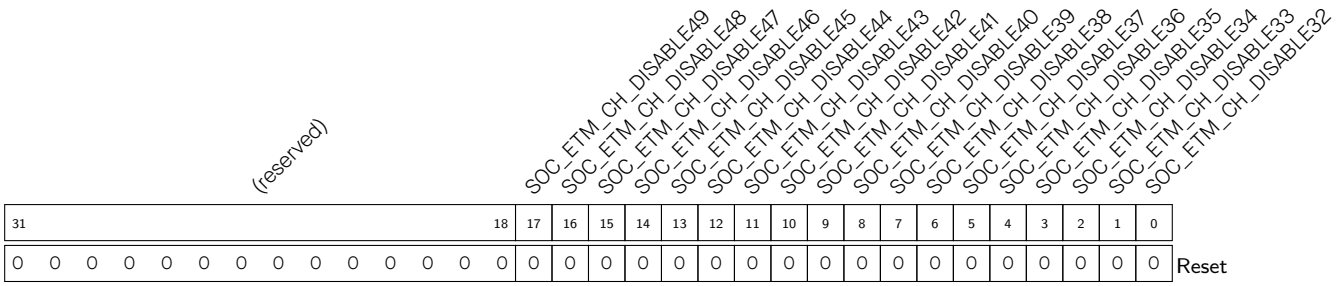
SOC_ETM_CH_ENABLE*n* (*n*: 32-49) Configures whether to enable channel*n*.

0: Invalid. No effect

1: Enable

(WT)

Register 10.6. SOC_ETM_CH_ENA_AD1_CLR_REG (0x0014)



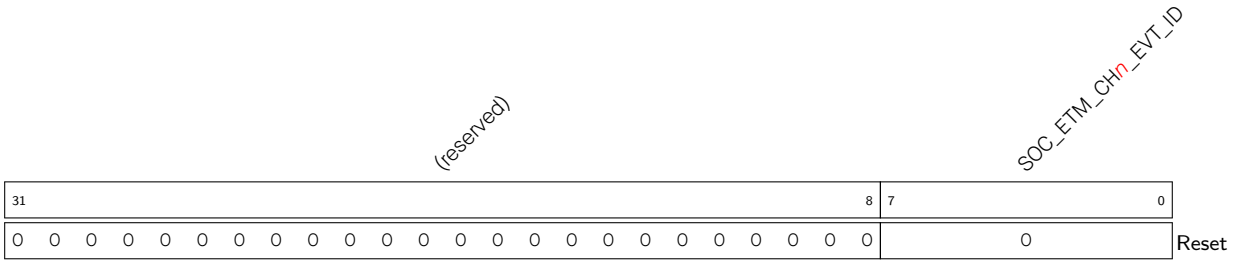
SOC_ETM_CH_DISABLE*n* (*n*: 32-49) Configures whether to disable channel*n*.

0: Invalid. No effect

1: Disable

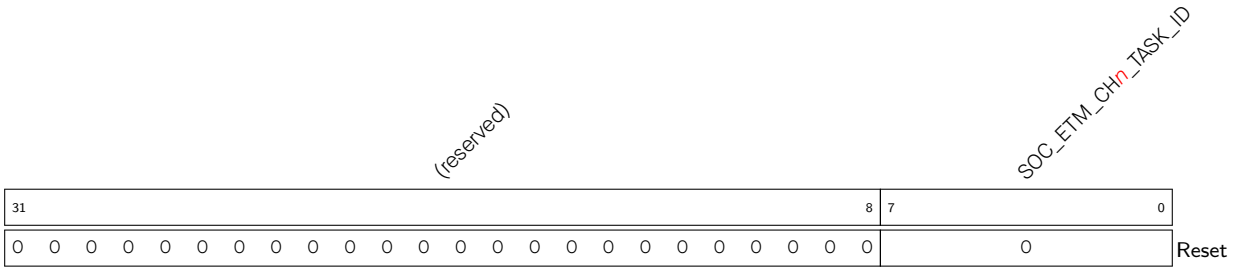
(WT)

Register 10.7. SOC_ETM_CH*n*_EVT_ID_REG (*n*: 0-49) (0x0018+0x8**n*)



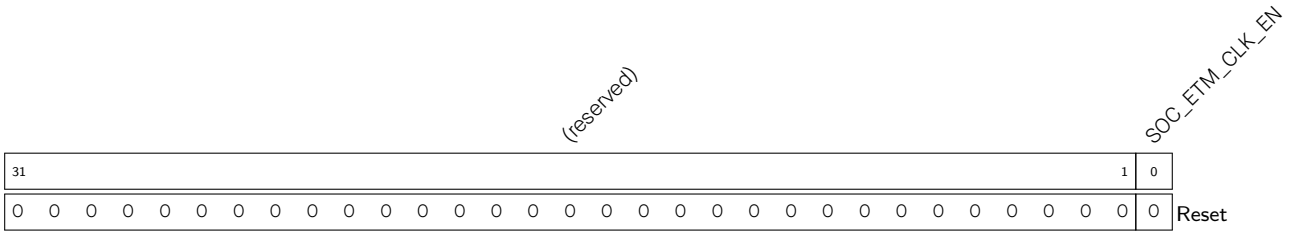
SOC_ETM_CH*n*_EVT_ID (*n*: 0-49) Configures the event ID of channel*n*. See Table 10.3-1. (R/W)

Register 10.8. SOC_ETM_CH*n*_TASK_ID_REG (*n*: 0-49) (0x001C+0x8**n*)



SOC_ETM_CH*n*_TASK_ID (*n*: 0-49) Configures the task ID of channel*n*. See Table 10.3-2. (R/W)

Register 10.9. SOC_ETM_CLK_EN_REG (0x01A8)



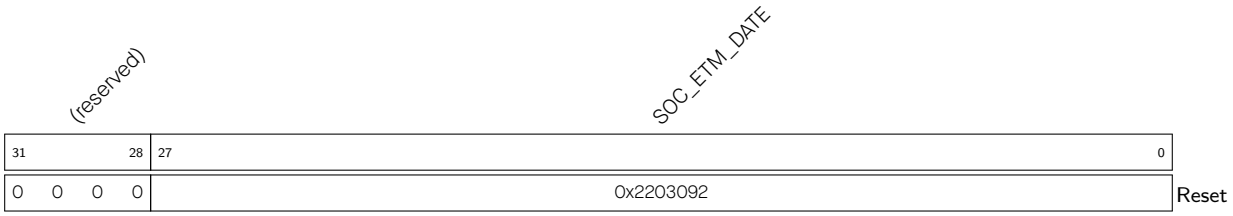
SOC_ETM_CLK_EN Configures resister clock gating.

0: Support clock only when application writes registers

1: Force on clock gating for registers

(R/W)

Register 10.10. SOC_ETM_DATE_REG (0x01AC)



SOC_ETM_DATE Version control register. (R/W)

Chapter 11

Low-Power Management

11.1 Overview

ESP32-H2 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.

11.2 Terminology

The following terms related to low-power management are defined in the context of the *ESP32-H2 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 three states of the PMU's state machine. Users can configure the clock gating and power gating of a power domain in each of these states.
Power modes	Refers to the five preset power modes that power up different domains for typical application scenarios.

11.3 Features

The PMU has the following features:

- Three configurable PMU states. Software can flexibly configure them according to the needs.
  - HP_ACTIVE
  - HP_SLEEP

  - LP_SLEEP

- Five preset power modes that suit various typical usage scenarios:
  - Active
  - Modem-sleep
  - Light-sleep0
  - Light-sleep1
  - Deep-sleep
- 4 KB SRAM
- 10 always-on (AON) registers
- RTC fast boot
- One power controller that controls the power and clocks depending on the power modes

11.4 Functional Description

ESP32-H2's low-power management involves the following components:

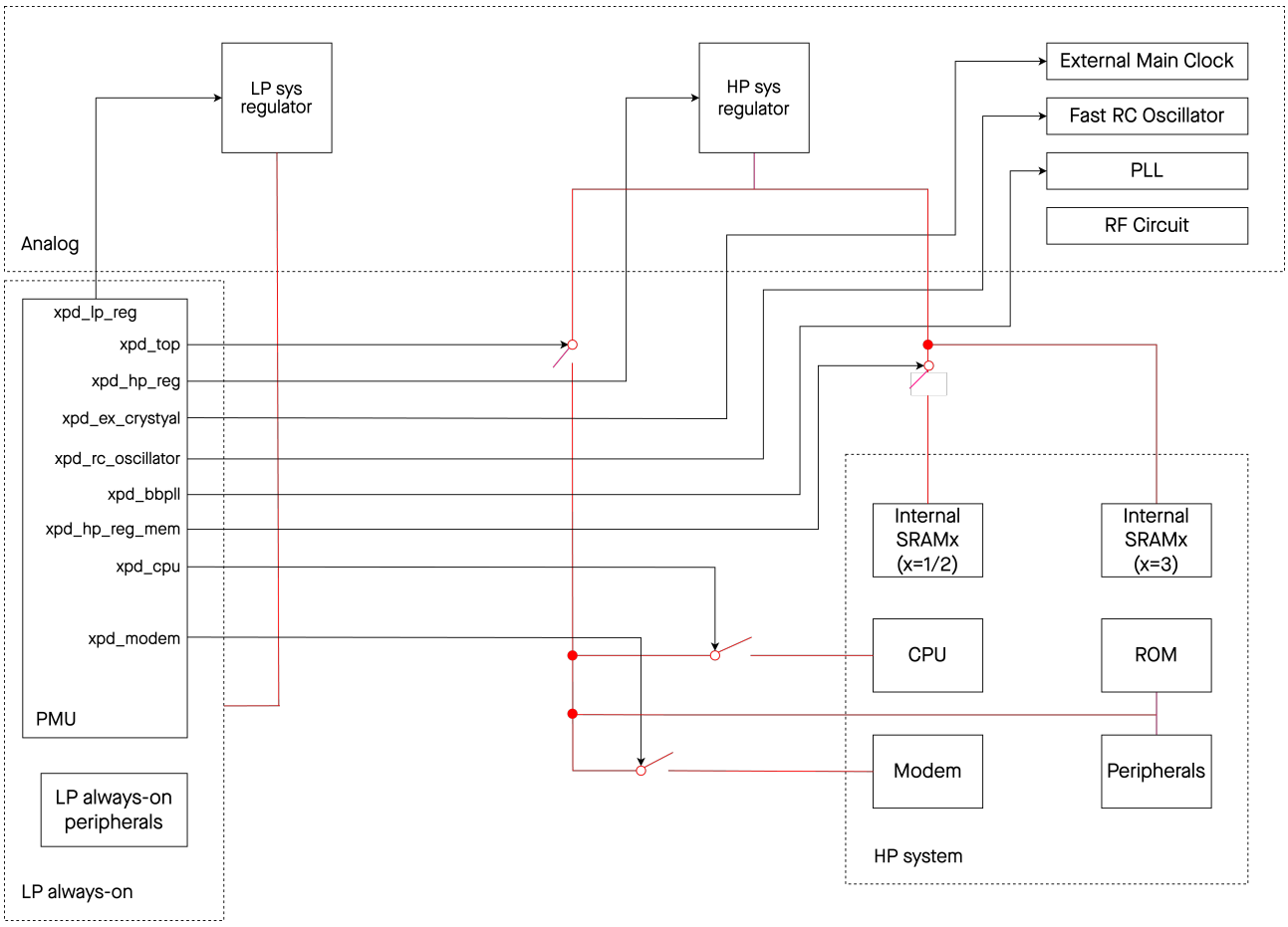
- **Power scheme**: The power scheme of ESP32-H2 includes power regulators, digital power domains, analog power domains, etc.
- **PMU controller**: It is the core part of PMU that controls the power up and down of the power domains, clocks, etc.
- **RTC timer**
- **10 always-on registers (LP_AON_STORE0_REG ~ 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.
- **Seven LP GPIO pins (GPIO8 ~ GPIO14)**: 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 6 *IO MUX and GPIO Matrix (GPIO, IO MUX)*.
- **4 KB SRAM**: It works under the AHB clock when accessed by the HP CPU. The 4 KB SRAM stays powered up in all power modes.

The following sections provide a detailed description of the components mentioned above.

11.4.1 Power Scheme

Figure 11.4-1 shows the power scheme of ESP32-H2 that mainly includes:

- Two regulators
- Analog power domains
- Digital power domains



–→: Control signals ––: Power lines

Figure 11.4-1. ESP32-H2 Power Scheme

11.4.1.1 Regulators

As shown in Figure 11.4-1, the analog part of ESP32-H2 contains two regulators that regulate the power supply to different power domains. The two regulators are:

- One HP sys regulator, used for regulating the power supply to high-performance modules. It features strong drive capabilities, high power consumption, and regulated output power.
- One LP sys regulator, used for regulating the power supply to low-power modules. It also features regulated output power.

11.4.1.2 Digital Power Domains

ESP32-H2 has digital power domains as listed below. The HP system regulator powers the HP system, with a dedicated power switch located between the regulator and each individual power domain. This setup allows for independent control of the digital power domains. The LP sys regulator powers the LP system.

- The HP system contains the following digital power domains:
  - **CPU**: It mainly includes the CPU and its supporting peripherals (such as TRACE).
  - **Modem**: It consists of wireless MAC and baseband.
  - **Peripherals + ROM**: It mainly includes bus and HP peripherals.

  - **Internal SRAMx**: It is divided into three sub-power domains, as shown in Figure 11.4-1, where each of SRAM1/2 has an independent power switch, while SRAM3 is directly connected to the regulator without a power switch.

- The LP system contains the following digital power domains:
  - **LP always-on**: It mainly includes LP always-on peripherals (e.g., RTC timer) and PMU controller. This power domain keeps powered in all power modes.

11.4.1.3 Analog Power Domains

As Figure 11.4-1 shows, ESP32-H2 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

11.4.2 PMU

The PMU of ESP32-H2 controls power consumption-related components of each power domain, such as power and clock. PMU consists of the following major parts:

- **PMU main state machine**: It records and switches the PMU states.
- **Sleep/wake-up controller**: It sends sleep or wake-up requests to the PMU main state machine.
- **Power controllers**: They control the power and clock signals depending on the power modes of the chip. The power controllers include:
  - **Digital power controller**: It powers up or down the digital power domains.
  - **Analog power controller**: It enables the analog modules, such as the regulators, analog clocks, etc.
  - **Clock controller**: It manages the clock gating of peripherals and selects analog clock sources for digital clocks.
  - **Data backup controller**: It controls the data backup and restore process when the chip switches between PMU states.
  - **System controller**: It controls some system-level modules, such as suspending watchdog functionality in sleep mode (when the CPU is unavailable).

The PMU workflow involves the 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 PMU states. The PMU workflow is shown in Figure 11.4-2. The definitions of HP_SWITCH and LP_SWITCH are as follows:

- HP_SWITCH: The intermediate state for the HP system transitioning between HP_ACTIVE and HP_SLEEP. The hardware will complete the control switching of various controllers in this state.

- LP_SWITCH: The intermediate state for the LP system transitioning between HP_SLEEP and LP_SLEEP. The hardware will complete the control switching of various controllers in this state.

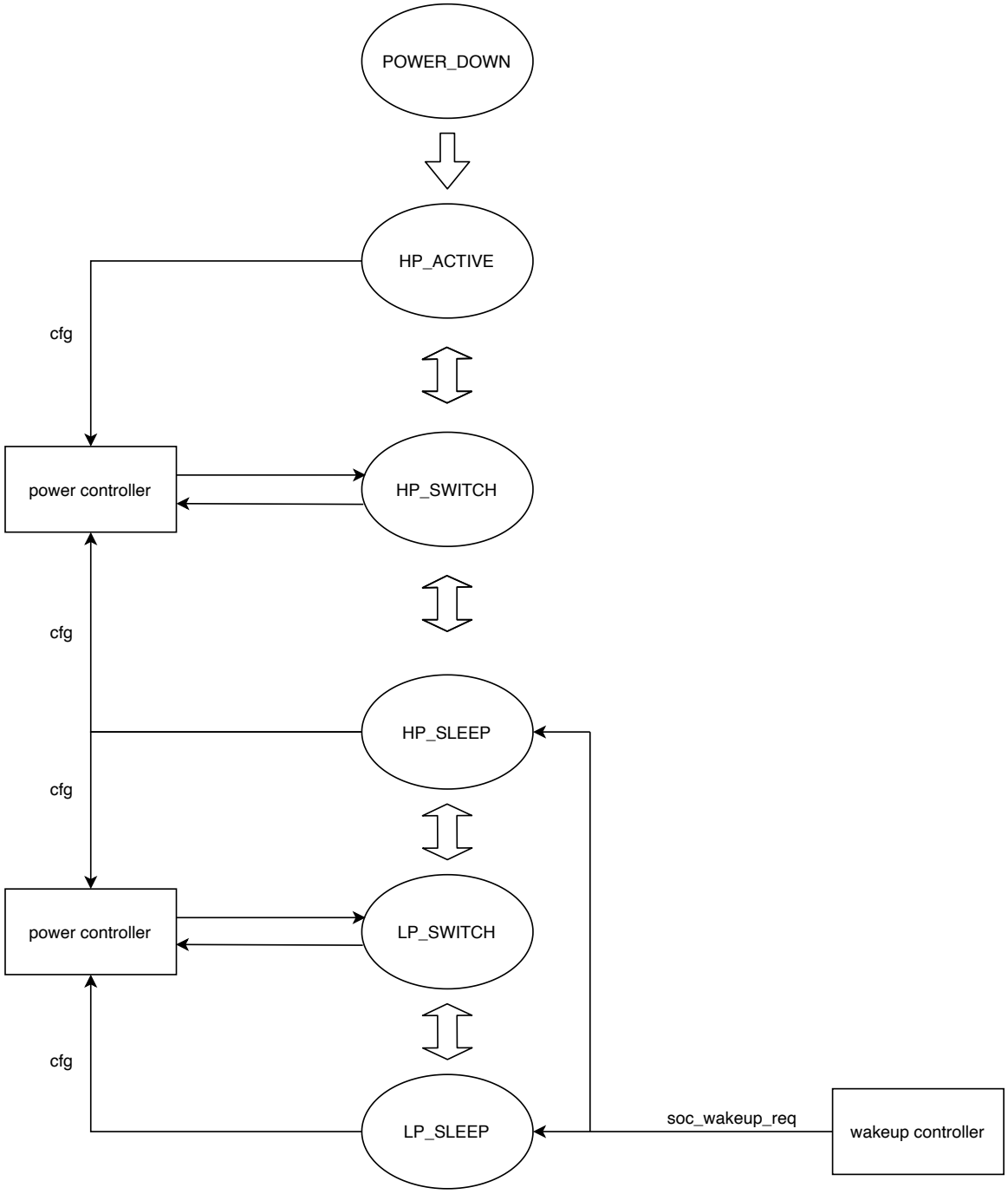


Figure 11.4-2. PMU Workflow

The following sections describe the main parts of PMU.

11.4.2.1 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 three PMU states, each controlled by different sleep and wake-up signals, supporting software customization of power and clocks. Based on the PMU sleep states (HP_SLEEP and LP_SLEEP), the software can expand power modes for various application scenarios (see Section 11.5). The 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_SLEEP: PMU state where the HP system is in sleep.
- LP_SLEEP: PMU state where HP system and LP system are in sleep. The LP system remains operational powered by the slow clock (RTC_SLOW_CLK).

Note:
The division of the HP and LP system is as follows:

- LP system (i.e., always-on circuits): PMU, RTC Watchdog Timer, Super Watchdog
- HP system: This includes all peripherals (including Modem) except those belonging to the LP system.

For more details, please refer to *ESP32-H2 Datasheet* > *Functional Block Diagram*.

HP_ACTIVE and 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_SLEEP states, but not in the LP_SLEEP state. It will reuse the HP_SLEEP state in the LP_SLEEP state.
- If a module belongs to the LP system, its power up/down can be configured in the LP_SLEEP state and will reuse the HP_SLEEP configuration in the HP_ACTIVE/HP_SLEEP states.

Take the HP CPU as an example. The HP CPU belongs to the HP system, so it can be configured to power up/down in the HP_ACTIVE/HP_SLEEP states through the following registers and reuse the HP_SLEEP configuration in LP_SLEEP.

- HP_ACTIVE: PMU_HP_ACTIVE_PD_HP_CPU_PD_EN
- HP_SLEEP: PMU_HP_SLEEP_PD_HP_CPU_PD_EN

Similarly, users can define other power domains' power up and down in different PMU states. For specific registers, please refer to Section 11.9.

Note:
In the following text, all such registers will be collectively referred to as PMU_*n1*_PD_*POWERDOMAIN*_PD_EN, where *n1* represents the PMU states.

Once the configuration is done, PMU will use various controllers to make these configurations effective, as described in the sections below.

11.4.2.2 Sleep/Wake-up Controller

The sleep/wake-up controller initiates sleep and wake-up requests to the PMU main state machine. ESP32-H2 supports multiple wake sources to wake the CPU from different power modes. PMU_WAKEUP_ENA enables the wake-up sources.

Table 11.4-1. Wake-up Sources

PMU_WAKEUP_ENA	Wake-up Sources	Light-sleep0	Deep-sleep
0x2	EXT	Y	Y
0x4	GPIO[1]	Y	–
0x10	RTC Timer	Y	Y
0x40	UART0[2]	Y	–
0x80	UART1[2]	Y	–
0x400	Bluetooth	Y	–
0x2000	Brown-out	Y	–
0x8000	vbat_detect	Y	–

[1] In Deep-sleep mode, only the LP GPIOs can work as wake-up sources.
[2] 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 28 *UART Controller (UART)*.

ESP32-H2 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 11.4-1 can all be configured as events to reject sleep. Users can configure the following registers to implement sleep rejection. The configuration values of PMU_SLEEP_REJECT_ENA and PMU_SLP_REJECT_CAUSE_REG and the corresponding wake-up sources are the same as shown in Table 11.4-1.

- 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_SLP_REJECT_CAUSE_REG for the source of sleep rejection event.

11.4.2.3 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_*n1*_HP_REGULATOR_XPD or PMU_*n1*_LP_REGULATOR_XPD to enable or disable the output voltage of the HP/LP sys regulator in the target PMU state. Turning off the LP sys regulator is not recommended, as it may cause the PMU itself to power down, resulting in chip malfunction.

The configuration of the high-speed clocks (XTAL_CLK and PLL_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_XTAL_STABLE to delay the gate opening for XTAL_CLK. This delay ensures that the gate opening is enabled after PMU_WAIT_XTAL_STABLE CLK_DYN_FAST_CLK cycles following the power-up of XTAL_CLK.

- PLL_CLK: PMU can enable PLL_CLK in different PMU states by configuring PMU_*n1*_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 CLK_DYN_FAST_CLK cycles in the number of PMU_WAIT_PLL_STABLE following the power-up of PLL_CLK.

The slow-speed clocks (RC_FAST_CLK and XTAL32K_CLK) operate with low power. In HP_ACTIVE and HP_SLEEP states, the power up and down of the slow-speed clocks are controlled by PMU_HP_SLEEP_LP_CK_POWER_REG. In LP_SLEEP state, the power up and down of the slow-speed clocks are controlled by PMU_LP_SLEEP_LP_CK_POWER_REG.

11.4.2.4 Digital Power Controller

The digital power controller controls the power up and down of digital power domains in different PMU states. Unlike the analog power controller, the digital power controller does not directly control the regulator but instead controls the power switch connected to the regulator to power up and down the digital power domains.

The digital power domains can be powered up and down while the PMU states switch between HP_SLEEP and HP_ACTIVE states.

When the chip switches between PMU states, if the power configuration of a power domain in the current PMU state does not match the power configuration it is about to switch to, then the power up-down process will be activated. Take the power up-to-down process of the CPU power domain as an example. Configure PMU_HP_SLEEP_PD_HP_CPU_PD_EN to 1 or 0 to indicate that the CPU power domain is powered down or up in the HP_SLEEP 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 power up and down of the Internal SRAMx domain is not controlled by a dedicated register. The Internal SRAMx domain shares the PMU_*n1*_PD_TOP_PD_EN register with the Peripherals domain. When PMU_PD_HP_MEM*n*_PD_MASK (*n*=1,2) is 0, both the Internal SRAMx and Peripherals power domains are turned up or down simultaneously. When PMU_PD_HP_MEM*n*_PD_MASK is 1, the Internal SRAMx can remain powered up when the Peripherals domain is powered down.

- Peripherals + ROM/Modem/CPU:

  Each of the three power domains can be powered up and down in different PMU states using the PMU_*n1*_PD_*n*_PD_EN register (*n*=TOP/HP_CPU).

  When the Peripherals domain is powered down, the following features are configurable:

  - Powering down the Peripherals domain may cause instability in GPIOs. This can be addressed by maintaining the state of the GPIOs (excluding the LP GPIOs) through PMU. For example, configuring PMU_HP_SLEEP_HP_PAD_HOLD_ALL can keep GPIOs in the same state as before Peripherals was powered down in HP_SLEEP.
  - In HP_ACTIVE state, 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_*n1*_HP_MEM_DSLP to enable Memory Deep-sleep mode, in which the digital power supply voltage must be higher than 0.9 V.

11.4.2.5 Clock Controller

The clock controller is mainly used to control high-performance system clocks and LP 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 between HP_ACTIVE 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_*n1*_SYS_CLK_SLP_SEL to 1 to indicate that when the chip enters the corresponding PMU state, the clock source is controlled by PMU.
  - Configure PMU_*n1*_ICG_SYS_CLOCK_EN to 0 to disable HP_ROOT_CLK in the corresponding PMU state.
  - Configure PMU_*n1*_DIG_SYS_CLK_SEL to select the clock source after the chip enters the corresponding PMU state. For details, please refer to Chapter 7 *Reset and Clock* > Table 7.2-1.
- High-performance system peripheral clocks can be controlled as follows:
  - Configure PMU_*n1*_ICG_SLP_SEL to 1 so that the clock gating in the target state will be controlled by PMU. Configure this register to 0 so that the clock gating is controlled by PCR registers.
  - Configure PMU_*n1*_DIG_ICG_FUNC_EN to power up/down the function clock in the target PMU state. For detailed configuration please see Table 11.4-2.

Table 11.4-2. HP System Peripherals' Function Clocks

PMU_*n1*_DIG_ICG_FUNC_EN Bit	Clock
bit 0	GDMA_CLK
bit 1	SPI2_CLK
bit 2	I2S_RX_CLK
bit 3	UART0_CLK
bit 4	UART1_CLK
bit 5	UHCI_CLK
bit 6	USB_CLK
bit 7	I2S_TX_CLK
bit 8	N/A
bit 9	N/A
bit 10	N/A
bit 11	N/A
bit 12	N/A
bit 13	TG1_CLK
bit 14	TG0_CLK
bit 15	N/A
bit 16	SOC_ETM_CLK
bit 17	N/A
bit 18	SYSTIMER_CLK
bit 19	N/A
bit 20	SARADC_CLK
bit 21	RMT_CLK
bit 22	MCPWM_CLK
bit 23	N/A
bit 24	PARLIO_TX_CLK
bit 25	PARLIO_RX_CLK
bit 26	N/A
bit 27	LEDC_CLK
bit 28	IOMUX_CLK
bit 29	I2C_CLK
bit 30	N/A
bit 31	TWAI0_CLK

– Configure PMU_*n1*_DIG_ICG_APB_EN to power up/down the APB clock in the target PMU state. For detailed configuration please see Table 11.4-3.

Table 11.4-3. HP System Peripherals' APB Clocks

PMU_*n1*_DIG_ICG_APB_EN Bit	Clock
bit 0	SEC_APB_CLK
bit 1	GDMA_APB_CLK
bit 2	API2_APB_CLK

PMU_*n1*_DIG_ICG_APB_EN Bit	Clock
bit 3	INTMTX_APB_CLK
bit 4	I2S_APB_CLK
bit 5	MSPI_APB_CLK
bit 6	UART0_APB_CLK
bit 7	UART1_APB_TX_CLK
bit 8	UHCI_APB_CLK
bit 9	SARADC_APB_CLK
bit 10	N/A
bit 11	TimerGroup0_APB_CLK
bit 12	TimerGroup1_APB_CLK
bit 13	I2C_APB_CLK
bit 14	LEDC_APB_CLK
bit 15	RMT_APB_CLK
bit 16	SYSTIMER_APB_CLK
bit 17	USB_DEVICE_APB_CLK
bit 18	TWAI0_APB_CLK
bit 19	N/A
bit 20	PCNT_APB_CLK
bit 21	PWM_APB_CLK
bit 22	SOC_ETM_CLK
bit 23	PARLIO_APB_CLK
bit 24	REGDMA_APB_CLK
bit 25	MEMORY_MONITOR_APB_CLK
bit 26	IOMUX_APB_CLK
bit 27	PVT_APB_CLK
bit 28	N/A
bit 29	N/A
bit 30	N/A
bit 31	N/A

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 frequency of LP_DYN_FAST_CLK is controlled by hardware as follows, depending on the PMU state (and cannot be changed by the user):

- LP_SLEEP: The LP_DYN_FAST_CLK frequency is the same as LP_SLOW_CLK.
- HP_ACTIVE, HP_SLEEP: The LP_DYN_FAST_CLK frequency is the same as LP_FAST_CLK.

11.4.2.6 Backup Controller

ESP32-H2 has a Retention DMA module 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 in the Peripherals power domain. PMU only generates relevant control signals. It is important to note that the data transfer control registers are directional, as unlike other control registers, these control behaviors are determined by both the original PMU state and the target PMU state. Since the basic configurations of the Retention DMA are in the Peripherals power domain, if you want to use the Retention DMA in sleep modes, do not power down the Peripherals power domain.

Taking the HP_SLEEP target PMU state as an example, the control registers for transitioning from HP_ACTIVE to HP_SLEEP and from HP_SLEEP to HP_ACTIVE are different. The possible PMU state switches are listed below, collectively represented by *n2* in the register names:

- HP_SLEEP2ACTIVE
- HP_ACTIVE2SLEEP

The following will introduce how PMU controls the Retention DMA:

- Enable data transfer: Configure PMU_*n2*_BACKUP_EN to 1 to enable data transfer when the corresponding PMU state switch is performed.
- Enable corresponding clocks: Before data transfer starts, configure PMU_*n2*_BACKUP_CLK_SEL to select the clock source of the Retention DMA, and configure PMU_*n1*_BACKUP_ICG_FUNC_EN to enable the clock.

  After the data transfer is completed, the value of PMU_*n1*_BACKUP_ICG_FUNC_EN is determined by the configuration of PMU_*n1*_DIG_ICG_FUNC_EN in the target PMU state.
- Configure data transfer direction: Configure the highest bit of PMU_*n2*_BACKUP_MODE:
  - 1: From peripheral to memory
  - 0: From memory to peripheral
- Select linked list pointer: Configure the lower two bits of PMU_*n2*_BACKUP_MODE to select the linked list pointer, specifically:
  - 0: PAU_LINK_ADDR_0
  - 1: PAU_LINK_ADDR_1
  - 2: PAU_LINK_ADDR_2
  - 3: PAU_LINK_ADDR_3

11.4.2.7 System Controller

The system controller controls some functional modules when the chip switches PMU states, to achieve stable and low-power chip performance. Specifically, the system controller supports:

- Pausing the watchdog function: Configuring PMU_*n1*_DIG_PAUSE_WDT to 1 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_*n1*_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 *6 IO MUX and GPIO Matrix (GPIO, IO MUX)* > Section 6.9.
- Disabling UART wake-up function: Configuring PMU_*n1*_UART_WAKEUP_EN to 0 can disable the UART wake-up modes in the corresponding PMU state. For more information on UART wake-up modes, please refer to Chapter *28 UART Controller (UART)*.
- Pausing CPU: Setting PMU_*n1*_DIG_CPU_STALL to 1 can suspend the CPU in the corresponding PMU state.

11.4.3 RTC Timer

ESP32-H2'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 RTC_SLOW_CLK. For the trigger conditions for the RTC timer, see Table 11.4-4.

Table 11.4-4. 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_UPDATE	Triggered when RTC_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_SLOW_CLK.
  - RTC_TIMER_MAIN_BUF0_HIGH
  - RTC_TIMER_MAIN_BUF0_LOW
- Register group 1 records the count value of the RTC timer under the previous trigger.
  - RTC_TIMER_MAIN_BUF1_HIGH
  - RTC_TIMER_MAIN_BUF1_LOW

Whenever a new trigger occurs, the following process takes place:

1. The record from the previous trigger is moved from register group 0 to register group 1.
2. The record in register group 1 will be overwritten.
3. The record of the current trigger is then stored in register group 0.

As a result, the RTC timer can store up to two trigger values at the same time.

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. Additionally, the RTC timer can also be used as a wake-up source (see Table 11.4-1).

11.5 Power Modes

ESP32-H2 supports three configurable PMU states. Based on the PMU states, five power modes have been defined for the most commonly seen application scenarios. For the details, please see Table 11.5-1.

Table 11.5-1. Preset Power Modes

Power Modes	Power Domain							
	LP always-on	Peripherals	Modem	CPU	RC_FAST_CLK	XTAL_CLK	PLL	RF circuit
Active	ON	ON	ON	ON	ON	ON	ON	ON
Modem-sleep	ON	ON	OFF	ON	ON	ON	ON/OFF	OFF
Light-sleep0	ON	ON	ON/OFF	OFF	ON	ON/OFF	ON/OFF	ON/OFF
Light-sleep1	ON	OFF	OFF	OFF	ON/OFF	ON/OFF	ON/OFF	ON/OFF
Deep-sleep	ON	OFF	OFF	OFF	OFF	OFF	OFF	OFF

Note:

1. For power consumption data, please refer to *ESP32-H2 Datasheet* > Section *Current Consumption*.
2. For supported wake-up sources, please refer to Table 11.4-1.

11.6 RTC Boot

In Deep-sleep mode, both the ROM and RAM of the chip are powered down, so the time required for SPI Boot (copying data from flash) upon waking up is long. Therefore, compared to Light-sleep and Modem-sleep modes, the wake-up from Deep-sleep mode takes longer time. However, in Deep-sleep mode, the 4 KB SRAM can remain powered up. Therefore, users can put small-sized code (i.e., less than 4 KB "deep sleep wake stubs") into the 4 KB SRAM to avoid the delay caused by SPI boot, thus speeding up the chip's wake-up process.

To enable RTC boot, follow the steps below:

1. Set LP_AON_CORE0_STAT_VECTOR_SEL to 1 to start up the chip from the RTC fast memory.
2. Calculate CRC for the RTC fast memory and save the result in LP_AON_STORE7_REG.
3. Set LP_AON_STORE6_REG to the entry address of the RTC fast memory.
4. Configure sleep mode for the chip.

5. When the CPU is powered up, it begins unpacking the ROM and performing initialization. Then, recalculate the CRC code of the RTC fast memory. If it matches the result stored in LP_AON_STORE7_REG, the CPU will jump to the entry address of the RTC fast memory. Otherwise, the CPU will continue to run the boot code.

The boot flow after the chip's wake-up is shown in Figure 11.6-1.

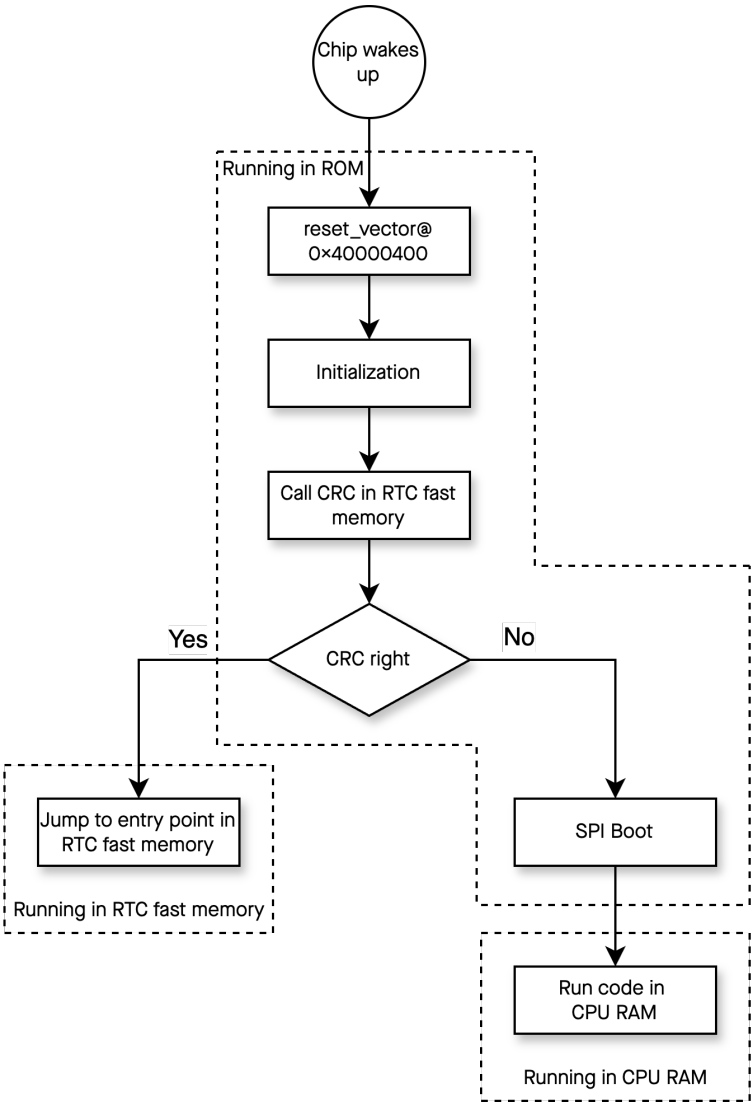


Figure 11.6-1. ESP32-H2 Boot Flow

11.7 Event Task Matrix Feature

The low-power management system on ESP32-H2 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 10 Event Task Matrix (SOC_ETM).

The low-power management system can receive the following ETM tasks:

- PMU_TASK_SLEEP_REQ: Triggers PMU's sleep process.

The low-power management system can generate the following ETM events:

- PMU_EVT_SLEEP_WAKEUP: Indicates that PMU is woken up to the HP_ACTIVE state.
- RTC_EVT_TICK: Indicates that the RTC timer increments by 1.

11.8 Interrupts

ESP32-H2's low-power management system can generate the following interrupt signals and send to the Interrupt Matrix:

- PMU_INTR
- LP_RTC_TIMER_INTR

The interrupt signals are generated by the internal interrupt sources of each module, specifically:

PMU_INTR:

- PMU_SOC_WAKEUP_INT: Triggered when the chip is woken up to the HP_ACTIVE state.
- PMU_SOC_SLEEP_REJECT_INT: Triggered when a sleep-rejection source rejects a sleep request.

LP_RTC_TIMER_INTR:

- RTC_TIMER_MAIN_TIMER_INT: Triggered when the count value of the RTC timer reaches the target value set in RTC_TIMER_MAIN_TIMER_TAR_LOW0 or RTC_TIMER_MAIN_TIMER_TAR_HIGH0.
- RTC_TIMER_MAIN_TIMER_OVERFLOW_INT: Triggered when the count value of the RTC timer reaches the maximum value.
  $MAX = (RTC\_TIMER\_MAIN\_TIMER\_TAR\_HIGH0 << 32) + RTC\_TIMER\_MAIN\_TIMER\_TAR\_LOW0$

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 11.9 *Register Summary*.

11.9 Register Summary

11.9.1 PMU Register Summary

The addresses in this section are relative to the PMU base address provided in Table 4.3-2 in Chapter 4 *System and Memory*.

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

Name	Description	Address	Access
Configuration Registers			
PMU_HP_ACTIVE_DIG_POWER_REG	Digital power domain control register in HP_ACTIVE state	0x0000	R/W
PMU_HP_ACTIVE_ICG_HP_FUNC_REG	HP system peripheral's function clock control register in HP_ACTIVE state	0x0004	R/W
PMU_HP_ACTIVE_ICG_HP_APB_REG	HP system peripheral's APB clock control register in HP_ACTIVE state	0x0008	R/W
PMU_HP_ACTIVE_HP_SYS_CNTL_REG	System control register in HP_ACTIVE state	0x0010	R/W
PMU_HP_ACTIVE_HP_CK_POWER_REG	Clock source power control register in HP_ACTIVE state	0x0014	R/W
PMU_HP_ACTIVE_BACKUP_REG	Backup module control register in HP_ACTIVE state	0x001C	R/W
PMU_HP_ACTIVE_BACKUP_CLK_REG	Backup module's function clock control register in HP_ACTIVE state	0x0020	R/W
PMU_HP_ACTIVE_SYSCLK_REG	System clock control register in HP_ACTIVE state	0x0024	R/W
PMU_HP_ACTIVE_HP_REGULATOR0_REG	Regulator power control register in HP_ACTIVE state	0x0028	R/W
PMU_HP_ACTIVE_XTAL_REG	XTAL_CLK power control register in HP_ACTIVE state	0x0030	R/W
PMU_HP_SLEEP_DIG_POWER_REG	Digital power domain control register in HP_SLEEP state	0x0068	R/W
PMU_HP_SLEEP_ICG_HP_FUNC_REG	HP system peripheral's function clock control register in HP_SLEEP state	0x006C	R/W
PMU_HP_SLEEP_ICG_HP_APB_REG	HP system peripheral's APB clock control register in HP_SLEEP state	0x0070	R/W
PMU_HP_SLEEP_HP_SYS_CNTL_REG	System control register in HP_SLEEP state	0x0078	R/W
PMU_HP_SLEEP_HP_CK_POWER_REG	Clock source power control register in HP_SLEEP state	0x007C	R/W
PMU_HP_SLEEP_BACKUP_REG	Backup module control register in HP_SLEEP state	0x0084	R/W
PMU_HP_SLEEP_BACKUP_CLK_REG	Backup flow ICG control register in HP_SLEEP state	0x0088	R/W

Name	Description	Address	Access
PMU_HP_SLEEP_SYSCLK_REG	System clock control register in HP_SLEEP state	0x008C	R/W
PMU_HP_SLEEP_HP_REGULATOR0_REG	Regulator power control register in HP_SLEEP state	0x0090	R/W
PMU_HP_SLEEP_XTAL_REG	XTAL_CLK power control register in HP_SLEEP state	0x0098	R/W
PMU_HP_SLEEP_LP_DIG_POWER_REG	LP system digital power domains control register in HP_SLEEP state	0x00A8	R/W
PMU_HP_SLEEP_LP_CK_POWER_REG	Low-speed clock power control register in HP_ACTIVE/HP_SLEEP state	0x00AC	R/W
PMU_LP_SLEEP_XTAL_REG	XTAL_CLK power control register in LP_SLEEP state	0x00BC	R/W
PMU_LP_SLEEP_LP_DIG_POWER_REG	Digital power domain control register in LP_SLEEP state	0x00C0	R/W
PMU_LP_SLEEP_LP_CK_POWER_REG	Low-speed clock power control register in LP_SLEEP state	0x00C4	R/W
PMU_POWER_PD_MEM_MASK_REG	Internal SRAMx domain force power up register	0x0110	R/W
PMU_POWER_CK_WAIT_CNTL_REG	Wait cycle for stable XTAL_CLK and PLL_CLK configuration register	0x011C	R/W
PMU_SLP_WAKEUP_CNTL0_REG	Sleep request register	0x0120	WT
PMU_SLP_WAKEUP_CNTL1_REG	Sleep reject register	0x0124	R/W
PMU_SLP_WAKEUP_CNTL2_REG	Wake up source enable register	0x0128	R/W
PMU_SLP_WAKEUP_CNTL4_REG	Sleep reject cause clear register	0x0130	WT
PMU_SLP_WAKEUP_STATUS0_REG	Wake up cause register	0x0140	RO
PMU_SLP_WAKEUP_STATUS1_REG	Reset reject cause register	0x0144	RO
PMU_RF_PWC_REG	SAR ADC power up register	0x0154	R/W
PMU_VDDBAT_CFG_REG	VBAT mode configuration register	0x0158	varies
PMU_INT_RAW_REG	PMU sleep/wake-up raw interrupt	0x0160	R/WTC/SS
PMU_HP_INT_ST_REG	PMU sleep/wake-up state interrupt	0x0164	RO
PMU_HP_INT_ENA_REG	PMU sleep/wake-up interrupt enable register	0x0168	R/W
PMU_HP_INT_CLR_REG	PMU sleep/wake-up raw interrupt clear register	0x016C	WT
PMU_DATE_REG	Version control register	0x03FC	R/W

11.9.2 Always-on Register Summary

The addresses in this section are relative to the Always-on Registers base address provided in Table 4.3-2 in Chapter 4 *System and Memory*.

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

Name	Description	Address	Access
Configuration Registers			
LP_AON_STORE0_REG	Always-on register0	0x0000	R/W
LP_AON_STORE1_REG	Always-on register1	0x0004	R/W
LP_AON_STORE2_REG	Always-on register2	0x0008	R/W
LP_AON_STORE3_REG	Always-on register3	0x000C	R/W
LP_AON_STORE4_REG	Always-on register4	0x0010	R/W
LP_AON_STORE5_REG	Always-on register5	0x0014	R/W
LP_AON_STORE6_REG	Always-on register6	0x0018	R/W
LP_AON_STORE7_REG	Always-on register7	0x001C	R/W
LP_AON_STORE8_REG	Always-on register8	0x0020	R/W
LP_AON_STORE9_REG	Always-on register9	0x0024	R/W
LP_AON_SYS_CFG_REG	Software system reset register	0x0034	varies
LP_AON_CPUCORE0_CFG_REG	CPU startup address configuration register	0x0038	varies

11.9.3 RTC Timer Register Summary

The addresses in this section are relative to the RTC Timer base address provided in Table 4.3-2 in Chapter 4 *System and Memory*.

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

Name	Description	Address	Access
Configuration Registers			
RTC_TIMER_TAR0_LOW_REG	Configures the low 32 bits of the target count value 0 of the RTC timer.	0x0000	R/W
RTC_TIMER_TAR0_HIGH_REG	Configures the high 16 bits of the target count value 0 of the RTC timer.	0x0004	R/W
RTC_TIMER_TAR1_LOW_REG	Configures the low 32 bits of the target count value 1 of the RTC timer.	0x0008	R/W
RTC_TIMER_TAR1_HIGH_REG	Configures the high 16 bits of the target count value 1 of the RTC timer.	0x000C	R/W
RTC_TIMER_UPDATE_REG	RTC timer value record register	0x0010	R/W
RTC_TIMER_MAIN_BUF0_LOW_REG	RTC timer register group0, bit0 - bit31	0x0014	RO
RTC_TIMER_MAIN_BUF0_HIGH_REG	RTC timer register group0, bit32 - bit47	0x0018	RO
RTC_TIMER_MAIN_BUF1_LOW_REG	RTC timer register group1, bit0 - bit31	0x001C	RO
RTC_TIMER_MAIN_BUF1_HIGH_REG	RTC timer register group1, bit32 - bit47	0x0020	RO
RTC_TIMER_INT_RAW_REG	LP_RTC_TIMER_INT raw interrupt	0x0028	RO
RTC_TIMER_INT_ST_REG	LP_RTC_TIMER_INT state interrupt	0x002C	RO
RTC_TIMER_INT_ENA_REG	LP_RTC_TIMER_INT interrupt enable register	0x0030	R/W
RTC_TIMER_INT_CLR_REG	LP_RTC_TIMER_INT interrupt clear register	0x0034	WT

11.10 Registers

11.10.1 PMU Registers

The addresses in this section are relative to the PMU base address provided in Table 4.3-2 in Chapter 4 *System and Memory*.

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

Register 11.1. PMU_HP_ACTIVE_DIG_POWER_REG (0x0000)

PMU_HP_ACTIVE_PD_TOP_PD_EN	(reserved)	PMU_HP_ACTIVE_PD_HP_CPU_PD_EN	(reserved)	PMU_HP_ACTIVE_PD_HP_MODEM_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	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_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)

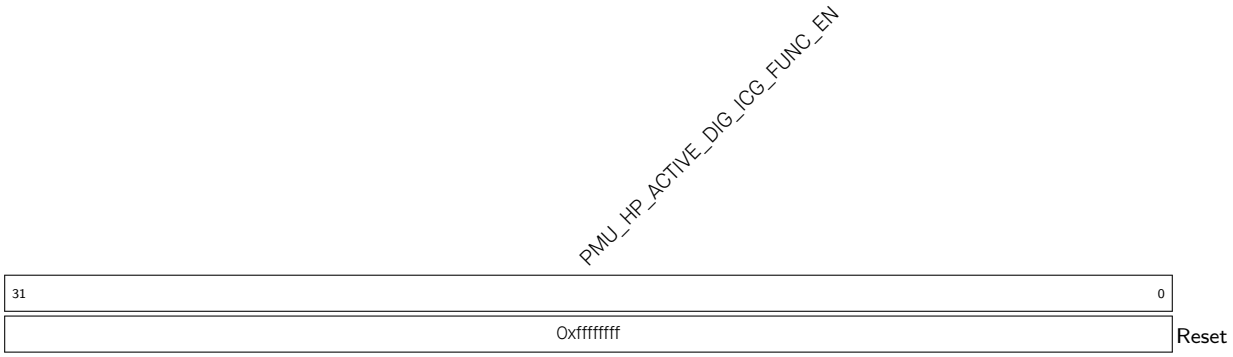
PMU_HP_ACTIVE_HP_MEM_DSLP Configures whether to put Internal SRAMx into Deep-sleep mode in HP_ACTIVE state.
0: Do not put Internal SRAMx into Deep-sleep
1: Put Internal SRAMx into Deep-sleep
(R/W)

PMU_HP_ACTIVE_PD_HP_MODEM_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_HP_CPU_PD_EN Configures whether to power down CPU 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 power domain in HP_ACTIVE state.
0: Power up
1: Power down
(R/W)

Register 11.2. PMU_HP_ACTIVE_ICG_HP_FUNC_REG (0x0004)



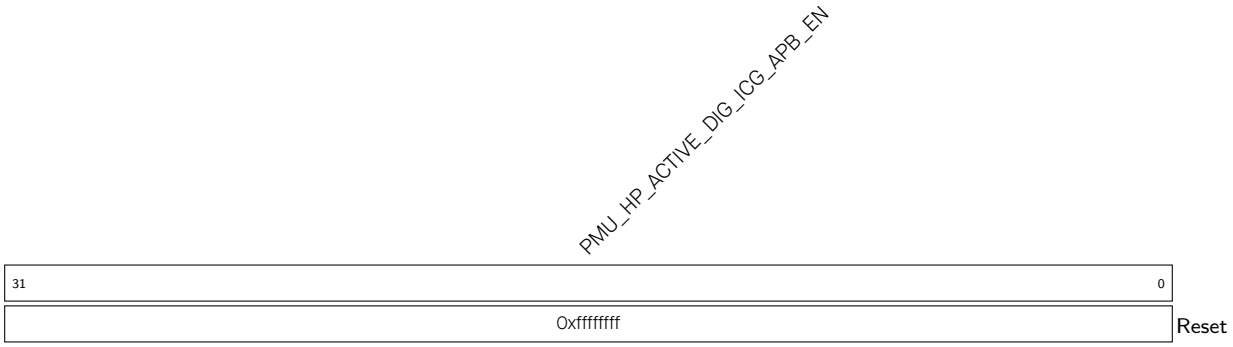
PMU_HP_ACTIVE_DIG_ICG_FUNC_EN Configures whether to enable HP system peripheral's function clock in HP_ACTIVE state. For detailed configuration please see Table 11.4-2.

0: Disable

1: Enable

(R/W)

Register 11.3. PMU_HP_ACTIVE_ICG_HP_APB_REG (0x0008)



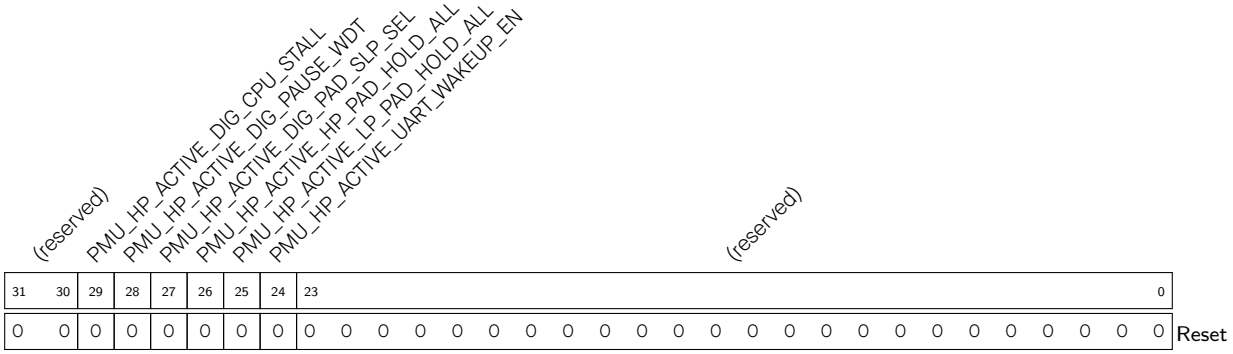
PMU_HP_ACTIVE_DIG_ICG_APB_EN Configures whether to enable HP system peripheral's APB clock in HP_ACTIVE state. For detailed configuration please see Table 11.4-3.

0: Disable

1: Enable

(R/W)

Register 11.4. PMU_HP_ACTIVE_HP_SYS_CNTL_REG (0x0010)



PMU_HP_ACTIVE_UART_WAKEUP_EN Configures whether to enable UART wake up function in HP_ACTIVE state.

0: Disable wake-up function

1: Enable wake-up function

(R/W)

PMU_HP_ACTIVE_LP_PAD_HOLD_ALL Configures whether to hold LP GPIO's configuration in HP_ACTIVE state.

0: Do not hold

1: Hold

(R/W)

PMU_HP_ACTIVE_HP_PAD_HOLD_ALL Configures whether to hold GPIO's configuration in HP_ACTIVE state.

0: Do not hold

1: Hold

(R/W)

PMU_HP_ACTIVE_DIG_PAD_SLP_SEL Configures whether to use Light-sleep mode configuration for GPIO in HP_ACTIVE state.

0: Use normal configuration

1: Use Light-sleep mode configuration. For details please refer to Chapter *6 IO MUX and GPIO Matrix (GPIO, IO MUX)* > Section *6.8 Pin Functions in Light-sleep*.

(R/W)

PMU_HP_ACTIVE_DIG_PAUSE_WDT Configures whether to pause watchdog in HP_ACTIVE state.

0: Do not pause

1: Pause

(R/W)

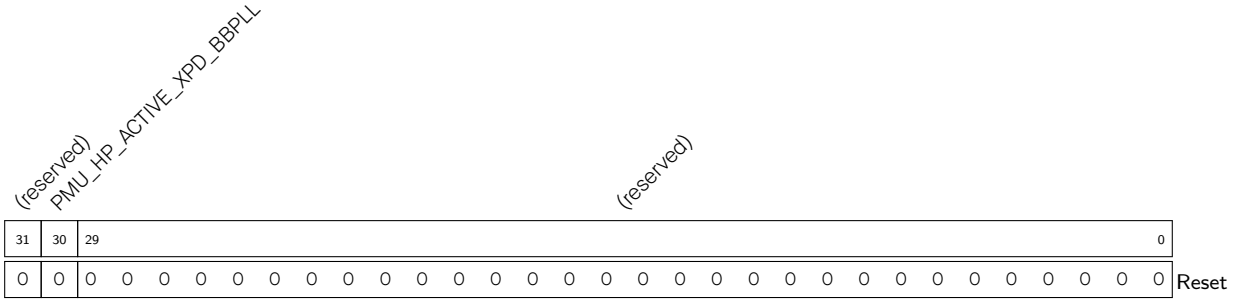
PMU_HP_ACTIVE_DIG_CPU_STALL Configures whether to stall the CPU in HP_ACTIVE state.

0: Do not stall

1: Stall

(R/W)

Register 11.5. PMU_HP_ACTIVE_HP_CK_POWER_REG (0x0014)



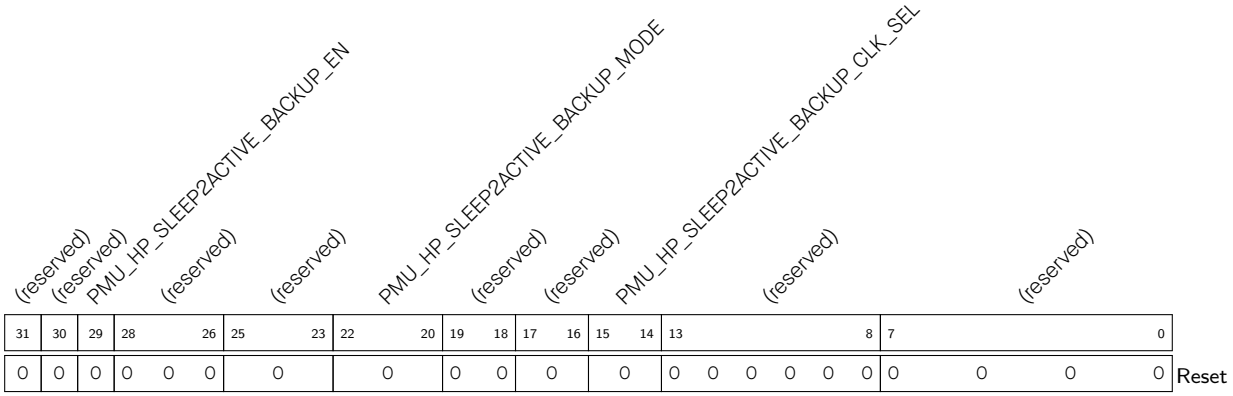
PMU_HP_ACTIVE_XPD_BBPLL Configures whether to enable PLL_CLK in HP_ACTIVE state.

0: Disable PLL_CLK

1: Enable PLL_CLK

(R/W)

Register 11.6. PMU_HP_ACTIVE_BACKUP_REG (0x001C)



PMU_HP_SLEEP2ACTIVE_BACKUP_CLK_SEL Configures the backup module's function clock source when PMU state switches from HP_SLEEP to HP_ACTIVE.

0: Select XTAL

1: Select PLL_CLK

2: Select RC_FAST_CLK

3: Invalid value

(R/W)

PMU_HP_SLEEP2ACTIVE_BACKUP_MODE Configures the backup direction and link list when PMU state switches switch from HP_SLEEP to HP_ACTIVE.

Highest bit:

0: From peripheral to memory

1: From memory to peripheral

Lower two bits:

0: PAU_LINK_ADDR_0

1: PAU_LINK_ADDR_1

2: PAU_LINK_ADDR_2

3: PAU_LINK_ADDR_3

(R/W)

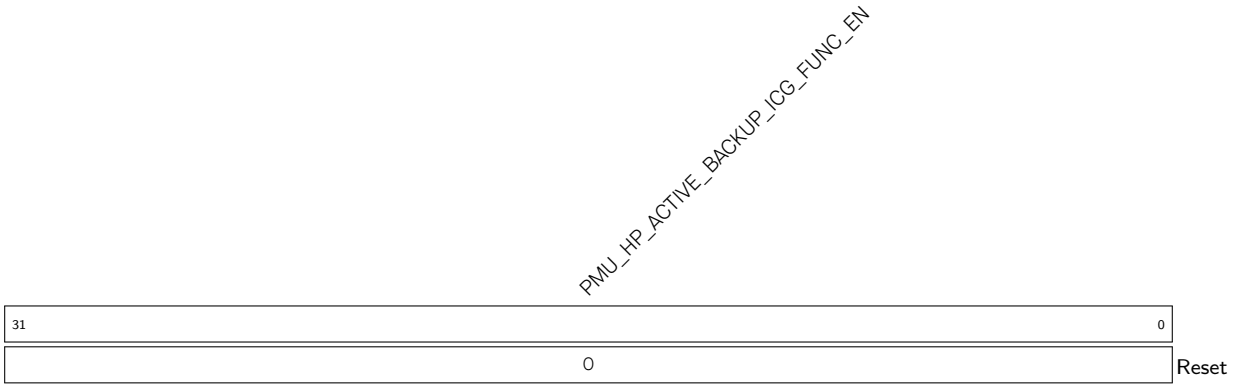
PMU_HP_SLEEP2ACTIVE_BACKUP_EN Configures whether to enable the backup flow when PMU state switches from HP_SLEEP to HP_ACTIVE.

0: Disable backup

1: Enable backup

(R/W)

Register 11.7. PMU_HP_ACTIVE_BACKUP_CLK_REG (0x0020)



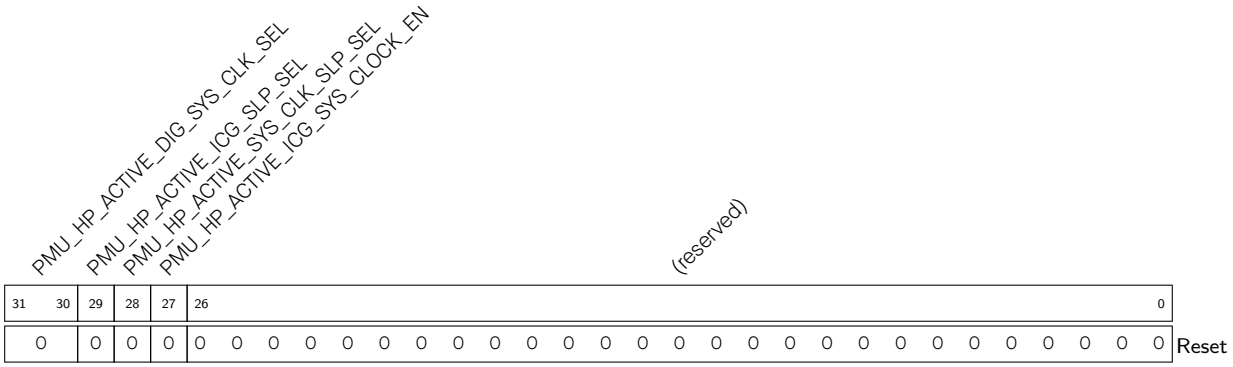
PMU_HP_ACTIVE_BACKUP_ICG_FUNC_EN Configures whether to enable each peripheral's function clock when the target state is HP_ACTIVE. For details, please refer to Chapter 7 *Reset and Clock*.

0: Disable

1: Enable

(R/W)

Register 11.8. PMU_HP_ACTIVE_SYSCLK_REG (0x0024)



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

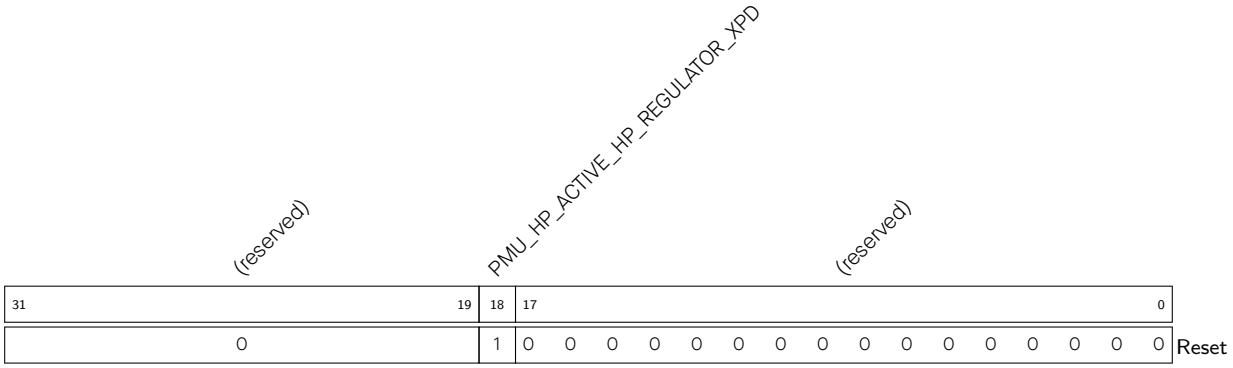
1: PLL_CLK

2: RC_FAST_CLK

3: Invalid value

(R/W)

Register 11.9. PMU_HP_ACTIVE_HP_REGULATOR0_REG (0x0028)



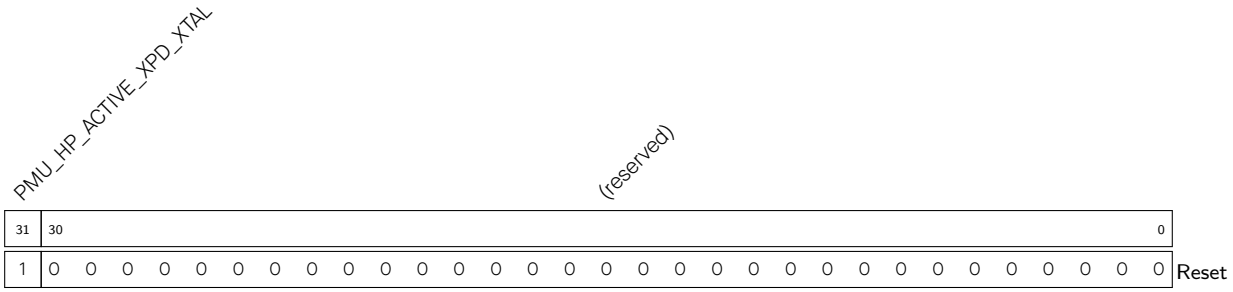
PMU_HP_ACTIVE_HP_REGULATOR_XPD Configures whether to enable the HP sys regulator in HP_ACTIVE state.

0: Disable the HP sys regulator

1: Enable the HP sys regulator

(R/W)

Register 11.10. PMU_HP_ACTIVE_XTAL_REG (0x0030)



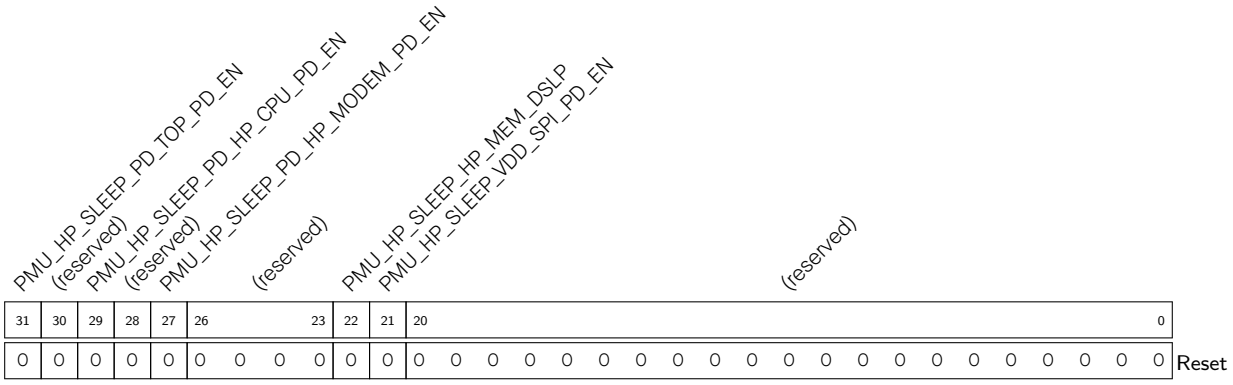
PMU_HP_ACTIVE_XPD_XTAL Configures whether to enable XTAL_CLK analog source in HP_ACTIVE state.

0: Disable

1: Enable

(R/W)

Register 11.11. 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 SRAMx into Deep-sleep mode in HP_SLEEP state.

0: Do not put Internal SRAMx into Deep-sleep

1: Put Internal SRAMx into Deep-sleep

(R/W)

PMU_HP_SLEEP_PD_HP_MODEM_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_HP_CPU_PD_EN Configures whether to power down CPU power domain in HP_SLEEP state.

0: Power up

1: Power down

(R/W)

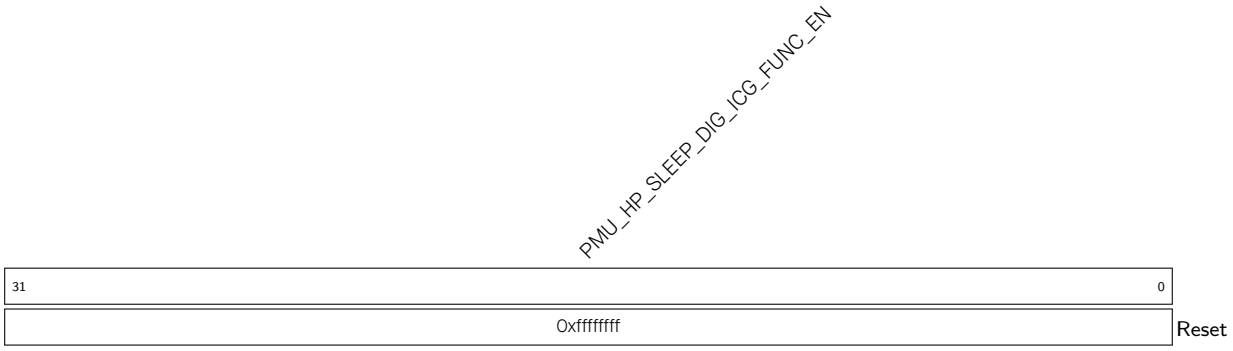
PMU_HP_SLEEP_PD_TOP_PD_EN Configures whether to power down Peripherals power domain in HP_SLEEP state.

0: Power up

1: Power down

(R/W)

Register 11.12. PMU_HP_SLEEP_ICG_HP_FUNC_REG (0x006C)



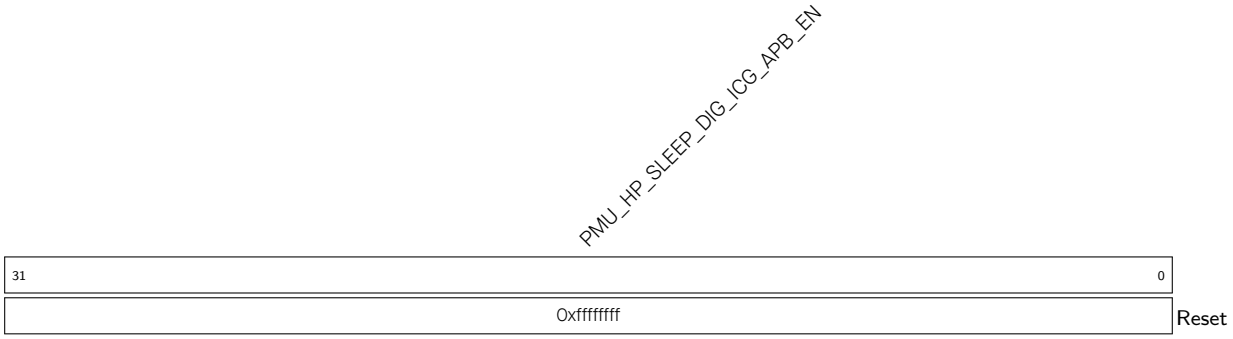
PMU_HP_SLEEP_DIG_ICG_FUNC_EN Configures whether to enable HP system peripheral's function clock in HP_SLEEP state. For detailed configuration please see Table 11.4-2.

0: Disable

1: Enable

(R/W)

Register 11.13. PMU_HP_SLEEP_ICG_HP_APB_REG (0x0070)



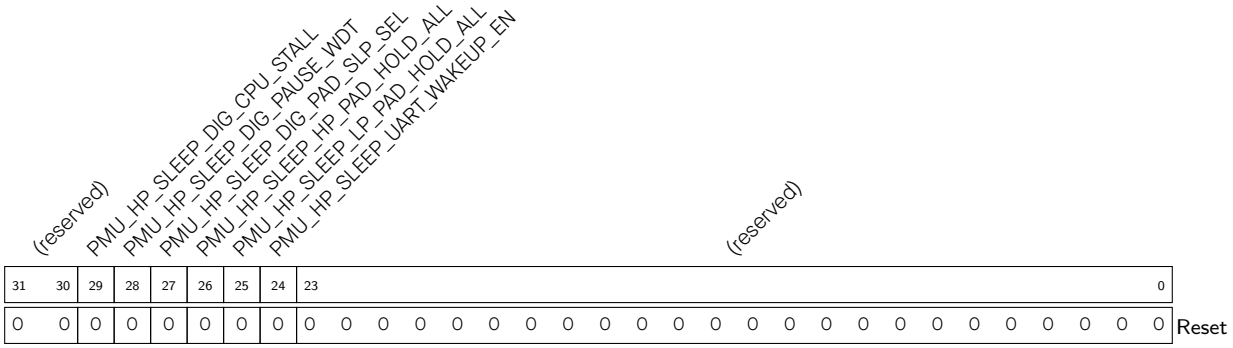
PMU_HP_SLEEP_DIG_ICG_APB_EN Configures whether to enable HP system peripheral's APB clock in HP_SLEEP state. For detailed configuration please see Table 11.4-3.

0: Disable

1: Enable

(R/W)

Register 11.14. PMU_HP_SLEEP_HP_SYS_CNTL_REG (0x0078)



PMU_HP_SLEEP_UART_WAKEUP_EN Configures whether to enable UART wake up function in HP_SLEEP state.

0: Disable wake-up function

1: Enable wake-up function

(R/W)

PMU_HP_SLEEP_LP_PAD_HOLD_ALL Configures whether to hold LP GPIO's configuration in HP_SLEEP state.

0: Do not hold

1: Hold

(R/W)

PMU_HP_SLEEP_HP_PAD_HOLD_ALL Configures whether to hold GPIO's configuration in HP_SLEEP state.

0: Do not hold

1: Hold

(R/W)

PMU_HP_SLEEP_DIG_PAD_SLP_SEL Configures whether to use Light-sleep mode configuration for GPIO in HP_SLEEP state.

0: Use normal configuration

1: Use Light-sleep mode configuration. For details please refer to Chapter *6 IO MUX and GPIO Matrix (GPIO, IO MUX)* > Section *6.8 Pin Functions in Light-sleep*.

(R/W)

PMU_HP_SLEEP_DIG_PAUSE_WDT Configures whether to pause watchdog in HP_SLEEP state.

0: Do not pause

1: Pause

(R/W)

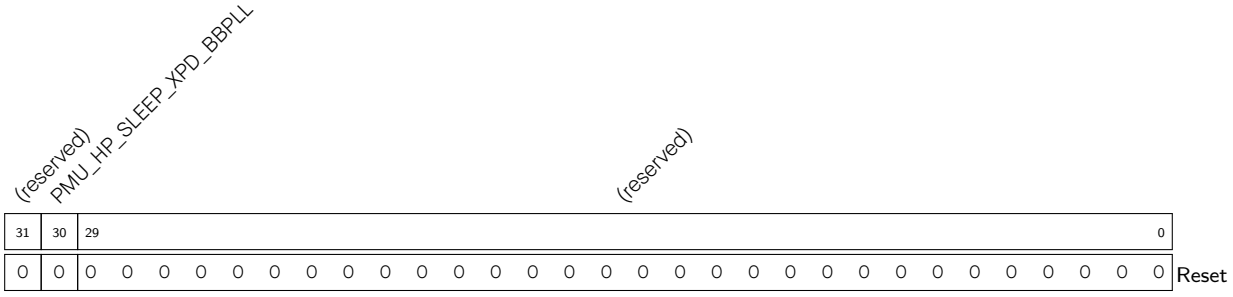
PMU_HP_SLEEP_DIG_CPU_STALL Configures whether to stall the CPU in HP_SLEEP state.

0: Do not stall

1: Stall

(R/W)

Register 11.15. PMU_HP_SLEEP_HP_CK_POWER_REG (0x007C)



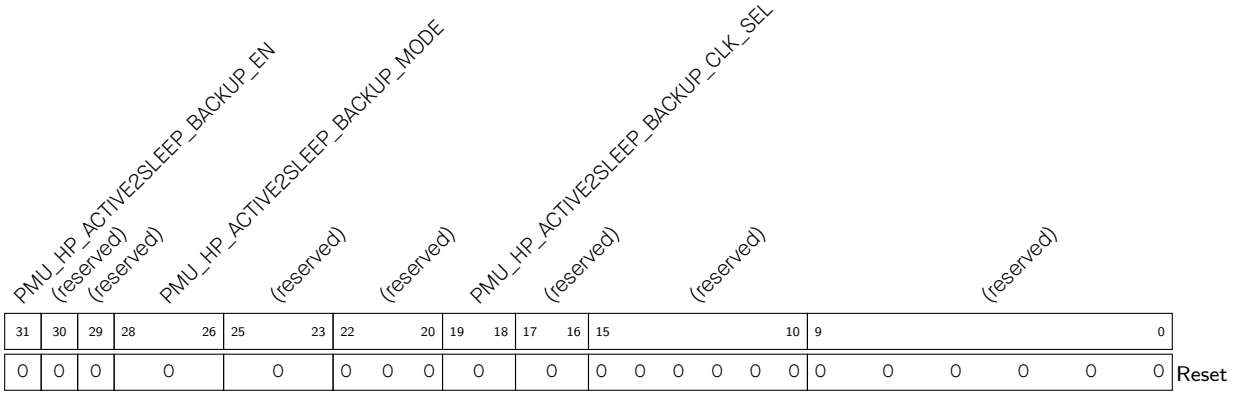
PMU_HP_SLEEP_XPD_BBPLL Configures whether to enable PLL_CLK in HP_SLEEP state.

0: Disable

1: Enable

(R/W)

Register 11.16. PMU_HP_SLEEP_BACKUP_REG (0x0084)



PMU_HP_ACTIVE2SLEEP_BACKUP_CLK_SEL Configures the backup module function clock source when PMU state switches from HP_ACTIVE to HP_SLEEP.

0: Select XTAL

1: Select PLL_CLK

2: Select RC_FAST_CLK

3: Invalid value

(R/W)

PMU_HP_ACTIVE2SLEEP_BACKUP_MODE Configures the backup direction and link list when PMU state switches from HP_ACTIVE to HP_SLEEP.

Highest bit:

0: From memory to peripheral

1: From peripheral to memory

Lower two bits:

0: PAU_LINK_ADDR_0

1: PAU_LINK_ADDR_1

2: PAU_LINK_ADDR_2

3: PAU_LINK_ADDR_3

(R/W)

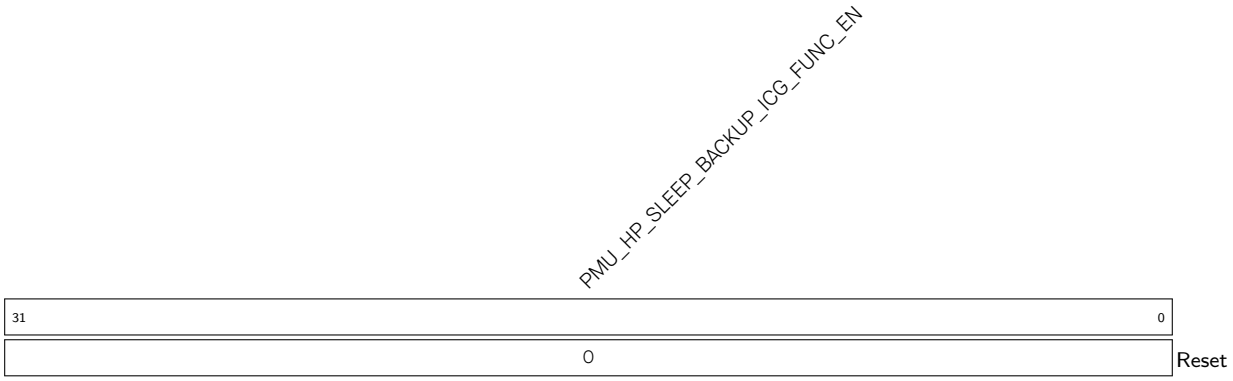
PMU_HP_ACTIVE2SLEEP_BACKUP_EN Configures whether to enable the backup flow when PMU state switches from HP_ACTIVE to HP_SLEEP.

0: Disable backup

1: Enable backup

(R/W)

Register 11.17. PMU_HP_SLEEP_BACKUP_CLK_REG (0x0088)



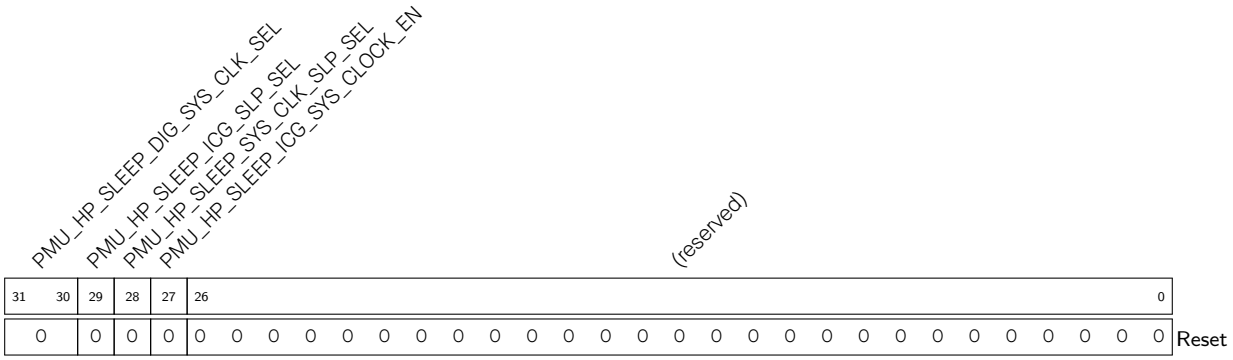
PMU_HP_SLEEP_BACKUP_ICG_FUNC_EN Configures whether to enable each peripheral's function clock when the target state is HP_SLEEP. For details, please refer to Chapter 7 *Reset and Clock*.

0: Disable

1: Enable

(R/W)

Register 11.18. PMU_HP_SLEEP_SYSCLK_REG (0x008C)



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

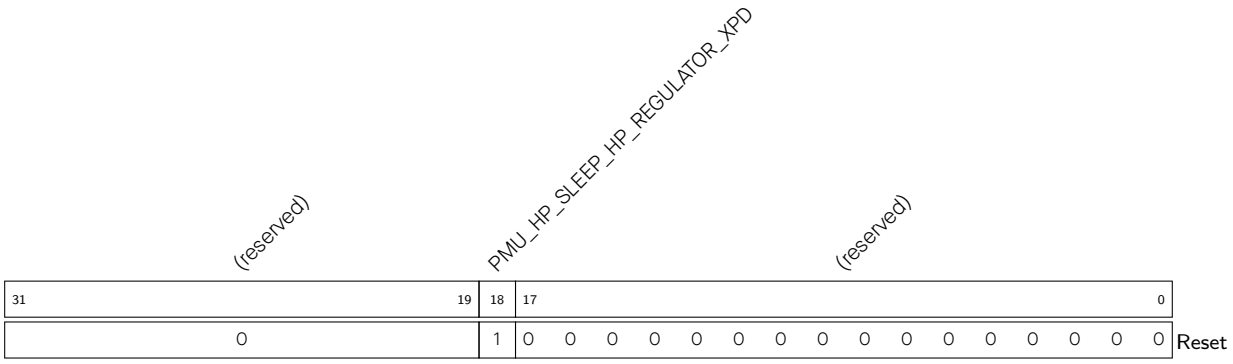
1: PLL_CLK

2: RC_FAST_CLK

3: Invalid value

(R/W)

Register 11.19. PMU_HP_SLEEP_HP_REGULATOR0_REG (0x0090)



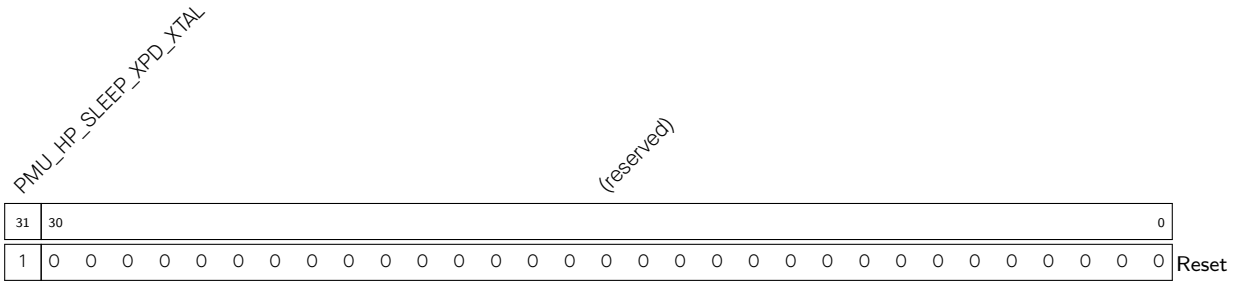
PMU_HP_SLEEP_HP_REGULATOR_XPD Configures whether to enable the HP sys regulator in HP_SLEEP state.

0: Disable

1: Enable

(R/W)

Register 11.20. PMU_HP_SLEEP_XTAL_REG (0x0098)



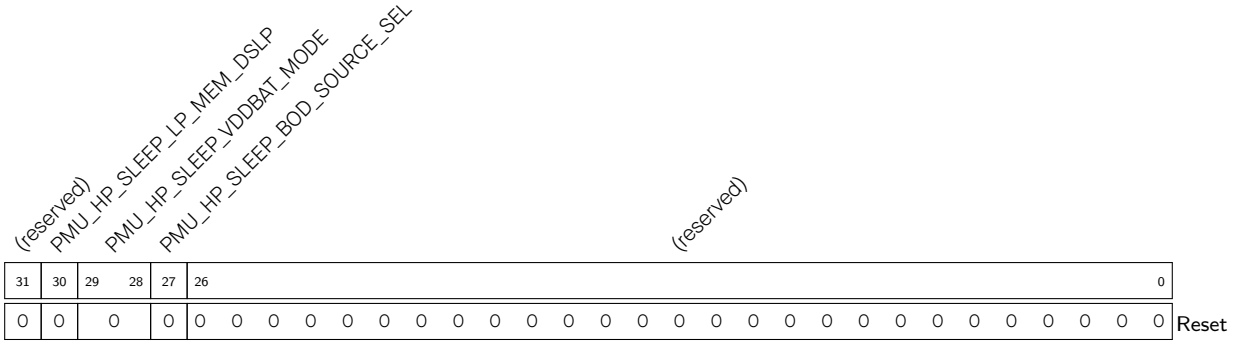
PMU_HP_SLEEP_XPD_XTAL Configures whether to enable XTAL_CLK analog source in HP_SLEEP state.

0: Disable

1: Enable

(R/W)

Register 11.21. PMU_HP_SLEEP_LP_DIG_POWER_REG (0x00A8)



PMU_HP_SLEEP_BOD_SOURCE_SEL Configures whether to enable brown-out detection in HP_SLEEP.(R/W)

PMU_HP_SLEEP_VDDBAT_MODE Selects the source of VDD_RTC.

0: VDDA_PMU

1: VBAT

2,3: reserved

(R/W)

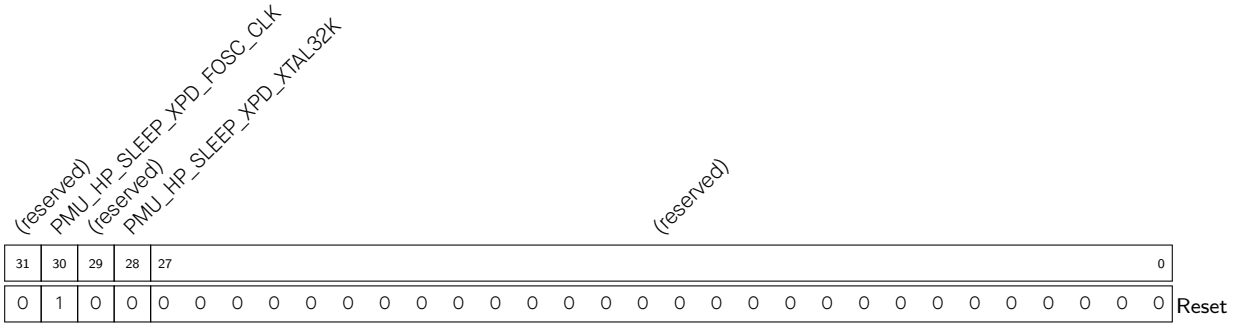
PMU_HP_SLEEP_LP_MEM_DSLP Configures whether to enable Memory Deep-sleep mode for the 4 KB SRAM in HP_ACTIVE/HP_SLEEP state.

0: Disable

1: Enable

(R/W)

Register 11.22. PMU_HP_SLEEP_LP_CK_POWER_REG (0x00AC)



PMU_HP_SLEEP_XPD_XTAL32K Configures whether to power up XTAL32K_CLK analog part circuit in HP_ACTIVE/HP_SLEEP state.

0: Power down

1: Power up

(R/W)

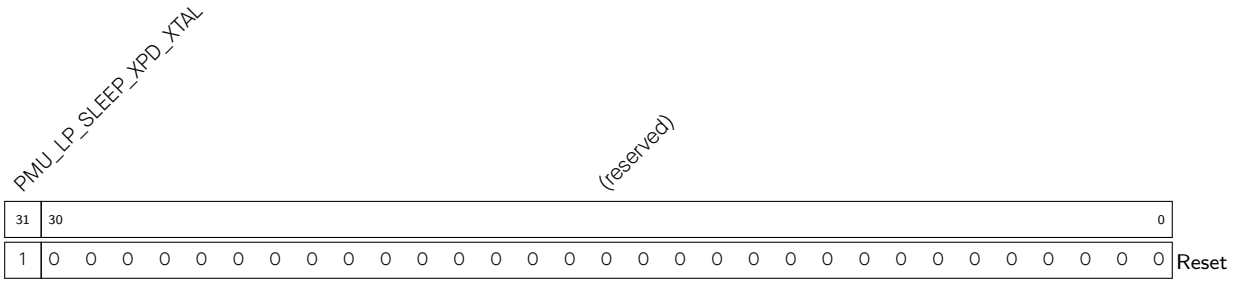
PMU_HP_SLEEP_XPD_FOSC_CLK Configures whether to power up RC_FAST_CLK analog part circuit in HP_ACTIVE/HP_SLEEP state.

0: Power down

1: Power up

(R/W)

Register 11.23. PMU_LP_SLEEP_XTAL_REG (0x00BC)



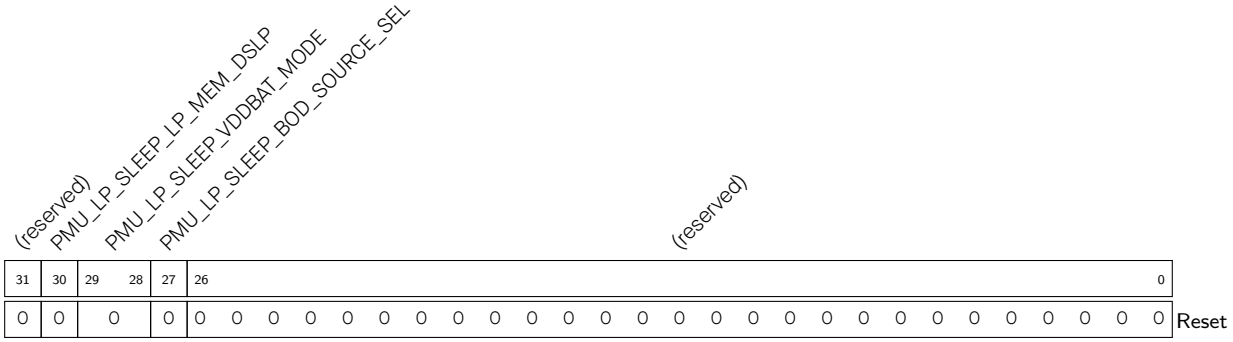
PMU_LP_SLEEP_XPD_XTAL Configures whether to enable XTAL_CLK analog source in LP_SLEEP state.

0: Disable

1: Enable

(R/W)

Register 11.24. PMU_LP_SLEEP_LP_DIG_POWER_REG (0x00C0)



PMU_LP_SLEEP_BOD_SOURCE_SEL Configures whether to enable brown-out detection in LP_SLEEP. (R/W)

PMU_LP_SLEEP_VDDBAT_MODE Selects the source of VDD_RTC.

0: VDDA_PMU

1: VBAT

2, 3: reserved

(R/W)

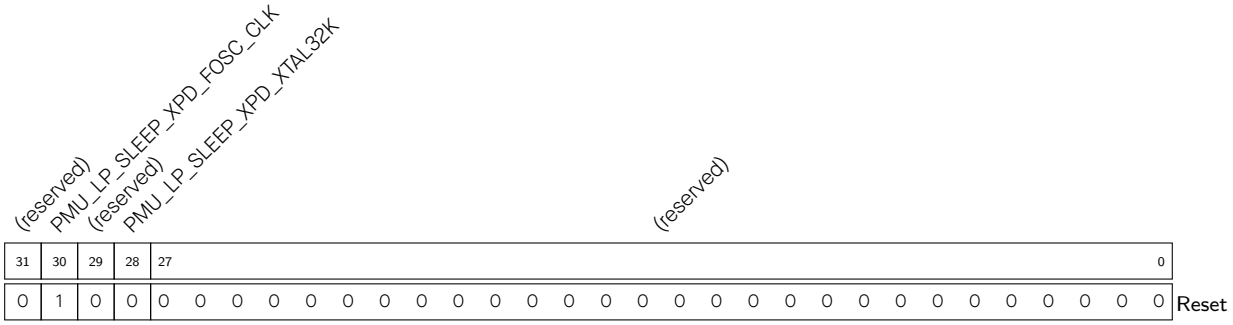
PMU_LP_SLEEP_LP_MEM_DSLP Configures whether to enable Memory Deep-sleep mode for the 4 KB SRAM in LP_SLEEP state.

0: Disable

1: Enable

(R/W)

Register 11.25. PMU_LP_SLEEP_LP_CK_POWER_REG (0x00C4)



PMU_LP_SLEEP_XPD_XTAL32K Configures whether to power up XTAL32K_CLK analog part circuit in LP_SLEEP state.

0: Power down

1: Power up

(R/W)

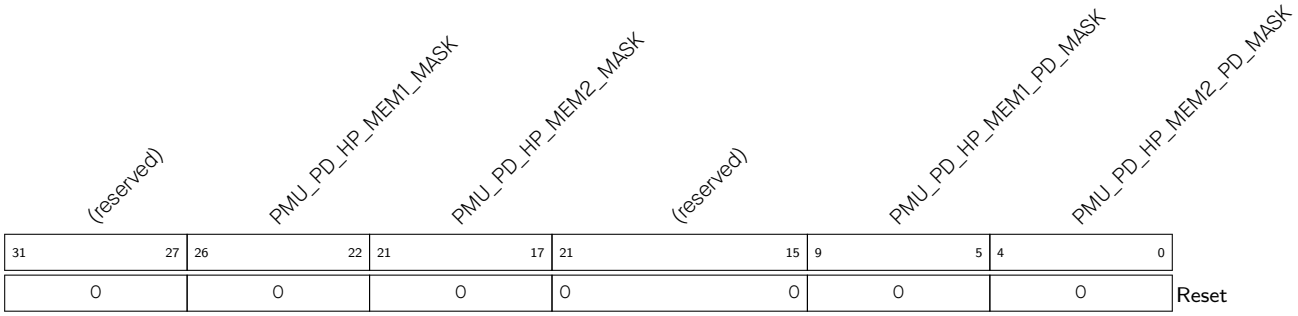
PMU_LP_SLEEP_XPD_FOSC_CLK Configures whether to power up RC_FAST_CLK analog part circuit in LP_SLEEP state.

0: Power down

1: Power up

(R/W)

Register 11.26. PMU_POWER_PD_MEM_MASK_REG (0x0110)



PMU_PD_HP_MEM2_PD_MASK Configures whether the Internal SRAM2 domain follows the power up/down state of the Peripherals domain.

0: Follows Peripherals domain

1: Does not follow Peripherals domain.

(R/W)

PMU_PD_HP_MEM1_PD_MASK Configures whether the Internal SRAM1 domain follows the power up/down state of the Peripherals domain.

0: Follows Peripherals domain

1: Does not follow Peripherals 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)

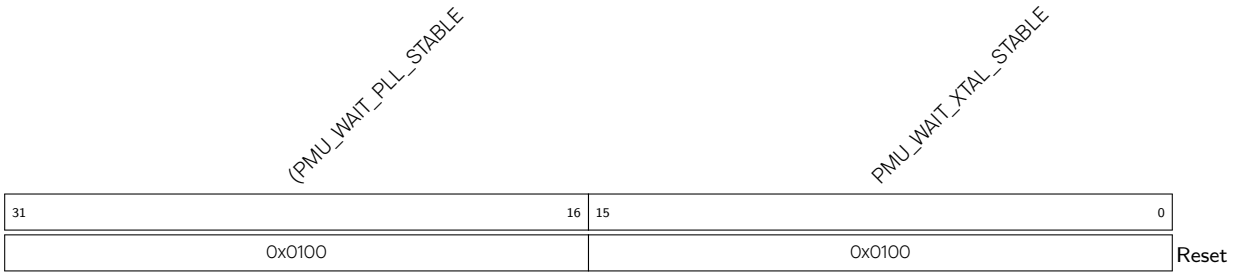
PMU_PD_HP_MEM1_MASK Configures whether to to force power up Internal SRAM1.

0: Do not force power up

1: Force power up

(R/W)

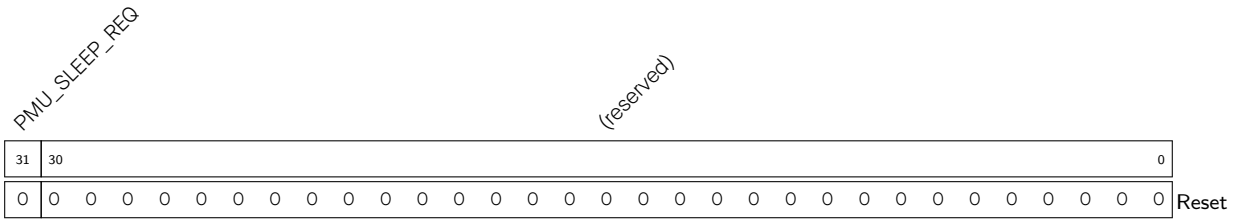
Register 11.27. PMU_POWER_CK_WAIT_CNTL_REG (0x011C)



PMU_WAIT_PLL_STABLE Configures the number of CLK_DYN_FAST_CLK cycles after which PLL_CLK gate opening is enabled. (R/W)

PMU_WAIT_XTAL_STABLE Configures the number of CLK_DYN_FAST_CLK cycles after which XTAL_CLK gate opening is enabled. (R/W)

Register 11.28. PMU_SLP_WAKEUP_CNTL0_REG (0x0120)



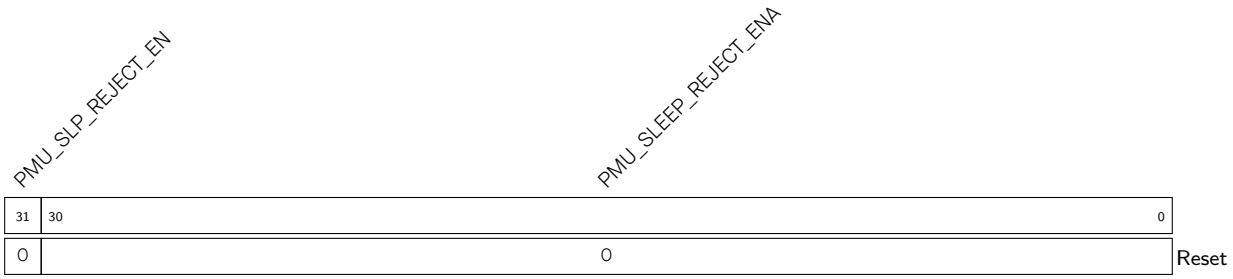
PMU_SLEEP_REQ Configures whether to switch the chip's PMU state to HP_SLEEP or LP_SLEEP.

0: Do not switch

1: Switch to HP_SLEEP or LP_SLEEP, depending on the state of the LP CPU.

(WT)

Register 11.29. PMU_SLP_WAKEUP_CNTL1_REG (0x0124)



PMU_SLEEP_REJECT_ENA Configures the sleep rejection source. For the mapping between values and sources please refer to Table 11.4-1. (R/W)

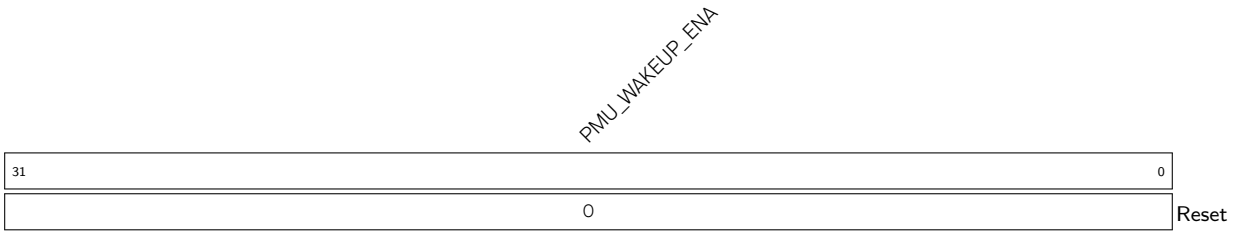
PMU_SLP_REJECT_EN Configures whether to enable sleep rejection function.

0: Disable

1: Enable

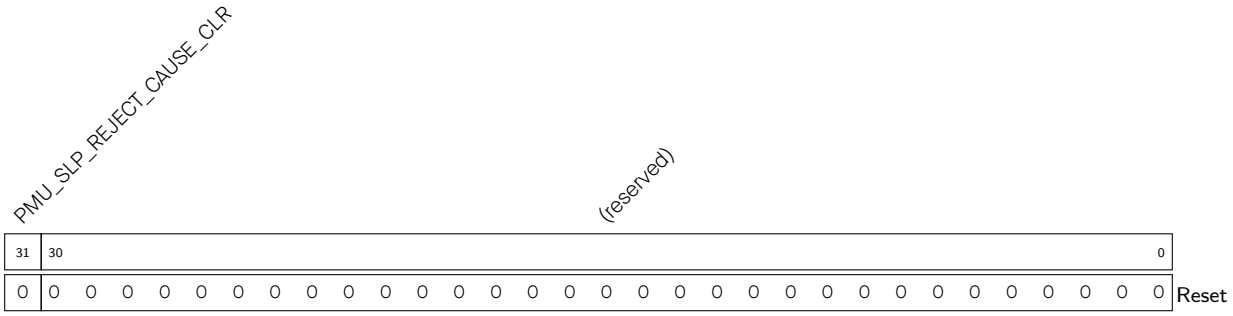
(R/W)

Register 11.30. PMU_SLP_WAKEUP_CNTL2_REG (0x0128)



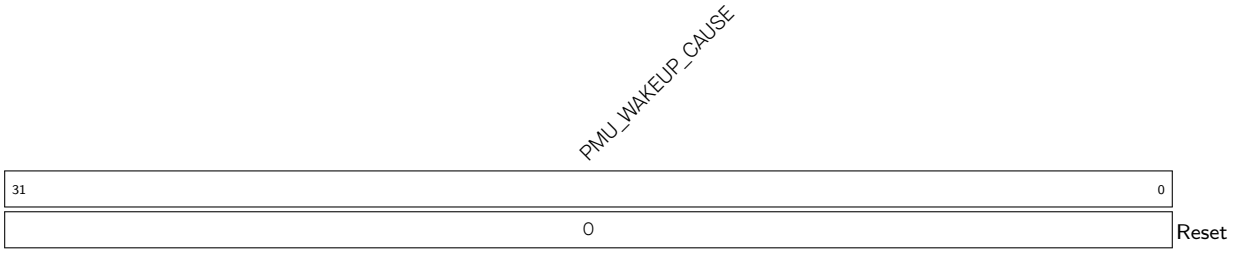
PMU_WAKEUP_ENA Configures wake-up source. For the mapping between values and sources please refer to Table 11.4-1. (R/W)

Register 11.31. PMU_SLP_WAKEUP_CNTL4_REG (0x0130)



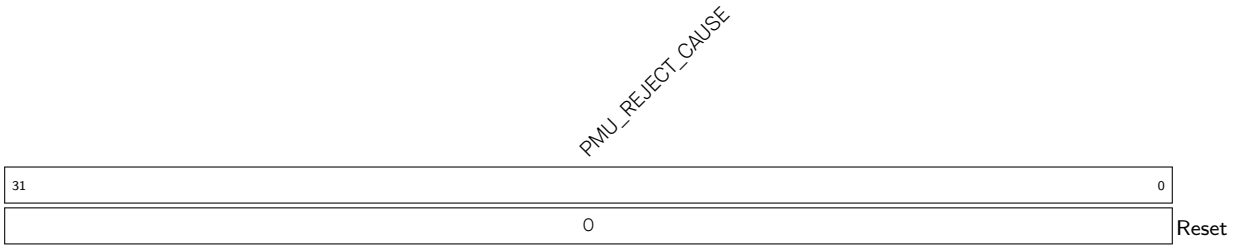
PMU_SLP_REJECT_CAUSE_CLR Write 1 to clear PMU_REJECT_CAUSE. (WT)

Register 11.32. PMU_SLP_WAKEUP_STATUS0_REG (0x0140)



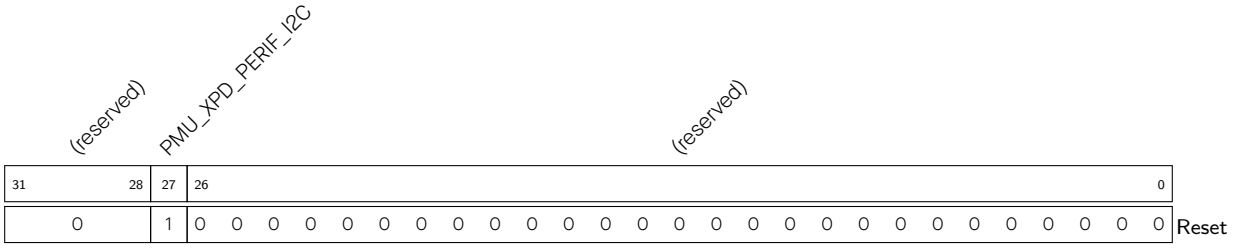
PMU_WAKEUP_CAUSE Indicates the wake-up source. For the mapping between values and sources please refer to Table 11.4-1. (RO)

Register 11.33. PMU_SLP_WAKEUP_STATUS1_REG (0x0144)



PMU_REJECT_CAUSE Indicates the wake-up rejection source. For the mapping between values and sources please refer to Table 11.4-1. (RO)

Register 11.34. PMU_RF_PWC_REG (0x0154)



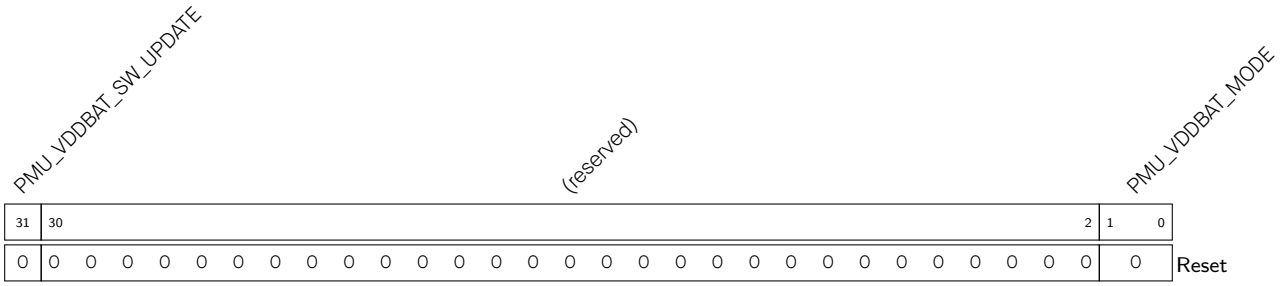
PMU_XPD_PERIF_I2C Configures whether to power up the SAR ADC.

0: Do not power up

1: Power up

(R/W)

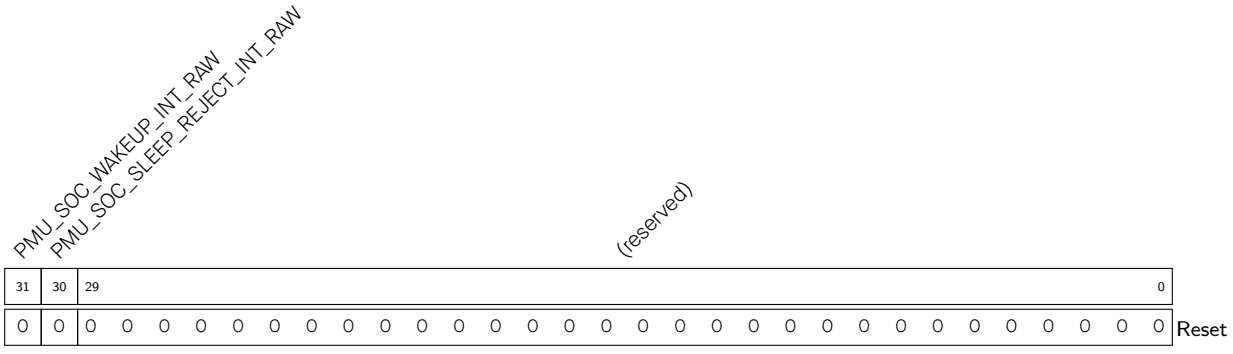
Register 11.35. PMU_VDDBAT_CFG_REG (0x0158)



PMU_VDDBAT_MODE Represents the source status of VDD_RTC. (RO)

PMU_VDDBAT_SW_UPDATE Software drives the VBAT controller to update the source of VDD_RTC. (WT)

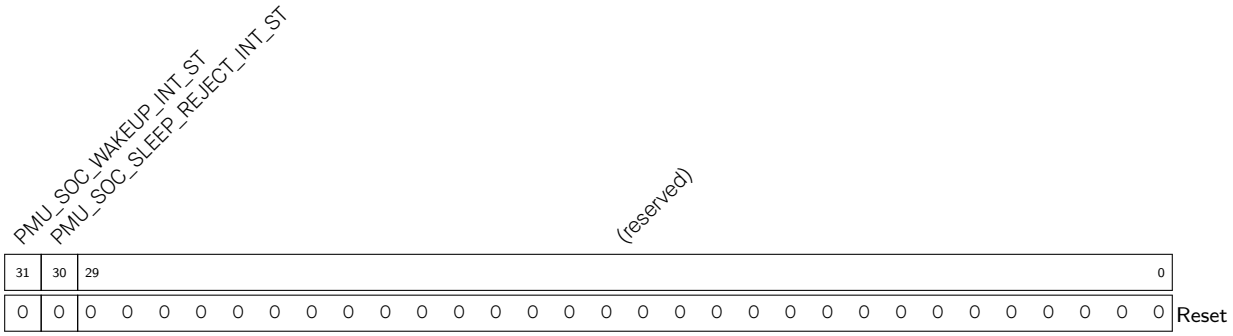
Register 11.36. PMU_INT_RAW_REG (0x0160)



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)

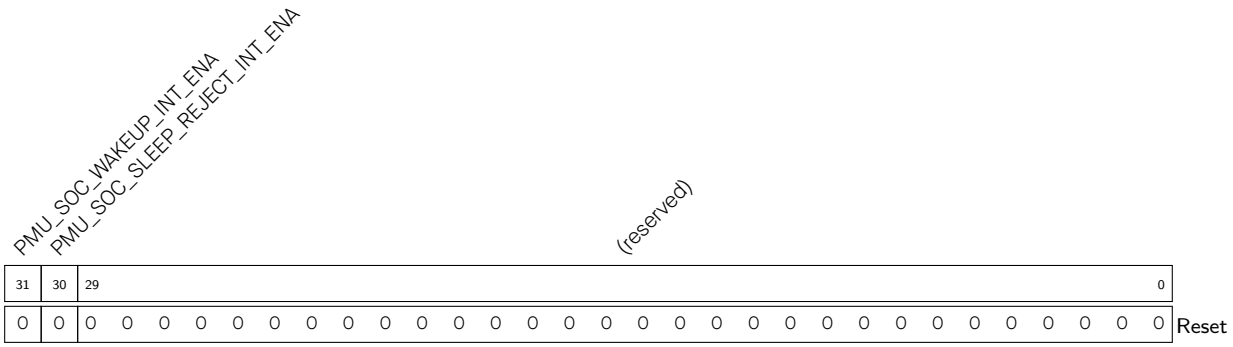
Register 11.37. PMU_HP_INT_ST_REG (0x0164)



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)

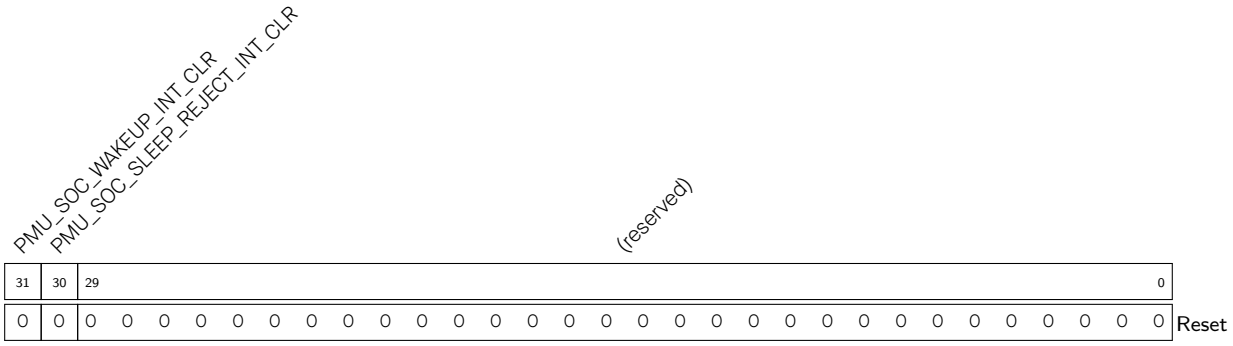
Register 11.38. PMU_HP_INT_ENA_REG (0x0168)



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)

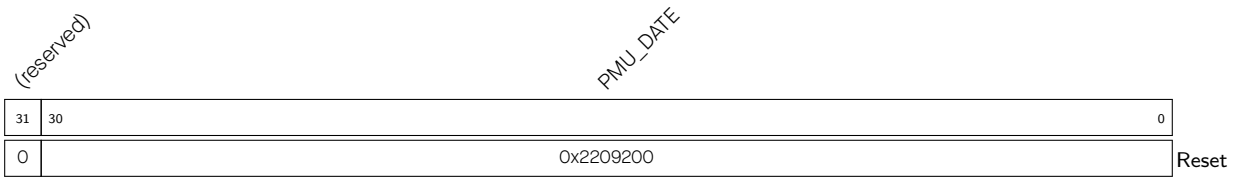
Register 11.39. PMU_HP_INT_CLR_REG (0x016C)



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 11.40. PMU_DATE_REG (0x03FC)



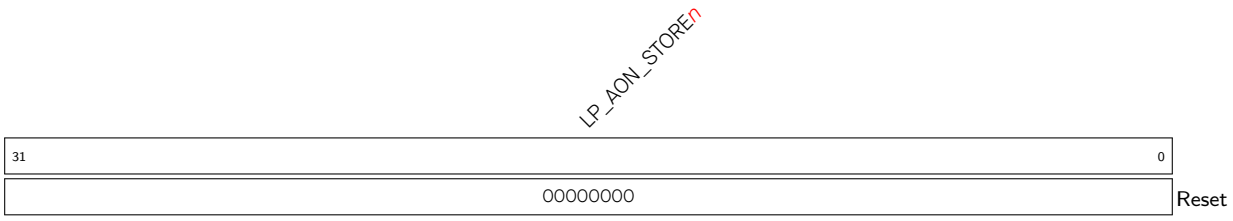
PMU_DATE Version control register. (R/W)

11.10.2 Always-on Registers

The addresses in this section are relative to the Always-on registers base address provided in Table 4.3-2 in Chapter 4 *System and Memory*.

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

Register 11.41. LP_AON_STORE*n*_REG (*n*: 0-9) (0x0000+0x4**n*)



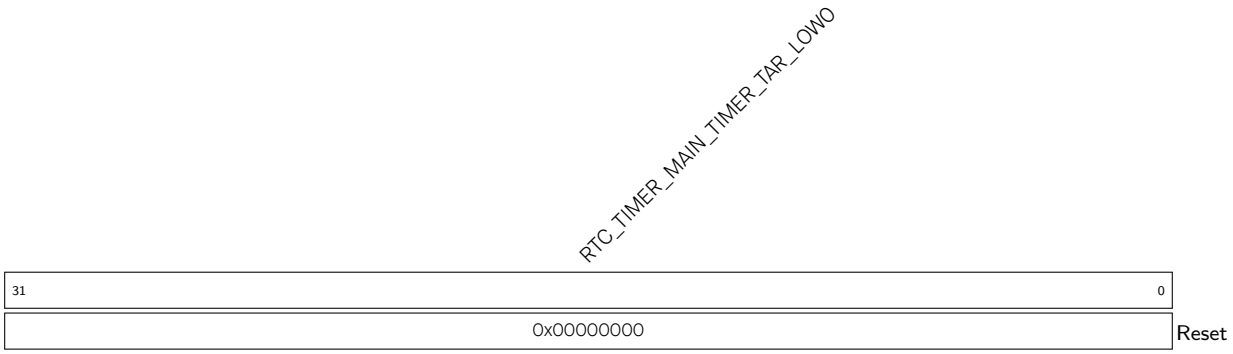
LP_AON_STORE*n* Always-on register *n*. (R/W)

11.10.3 RTC Timer Registers

The addresses in this section are relative to the RTC Timer base address provided in Table 4.3-2 in Chapter 4 *System and Memory*.

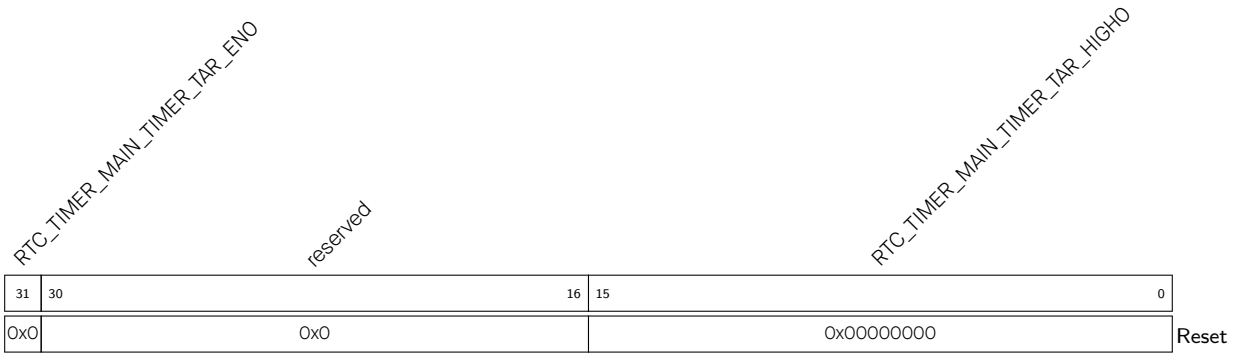
For how to program reserved fields, please refer to Section *Programming Reserved Register Field*.

Register 11.42. RTC_TIMER_TAR0_LOW_REG (0x0000)



RTC_TIMER_MAIN_TIMER_TAR_LOW0 Configures the low 32 bits of the target count value 0 (48 bits total) of the RTC timer. (R/W)

Register 11.43. RTC_TIMER_TAR0_HIGH_REG (0x0004)



RTC_TIMER_MAIN_TIMER_TAR_EN0 Configures whether to generate interrupts for the target count value 0 of the RTC timer.

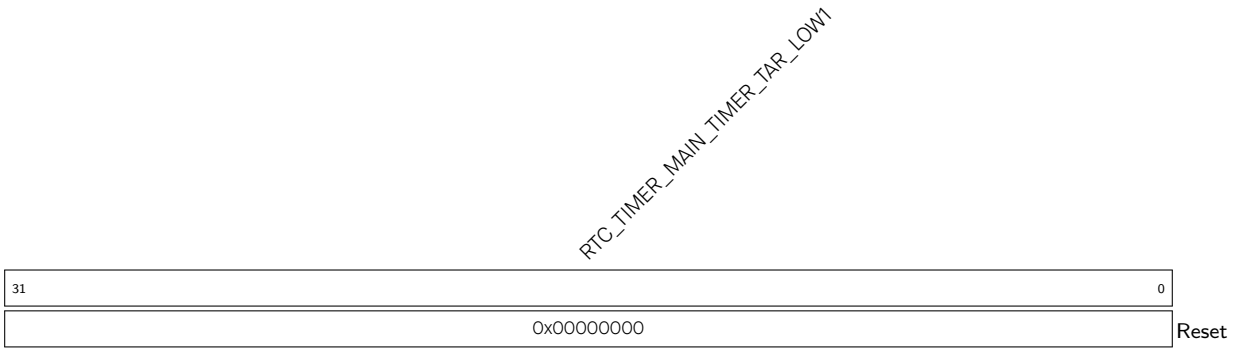
0: Do not generate interrupts

1: Generate interrupts

(R/W)

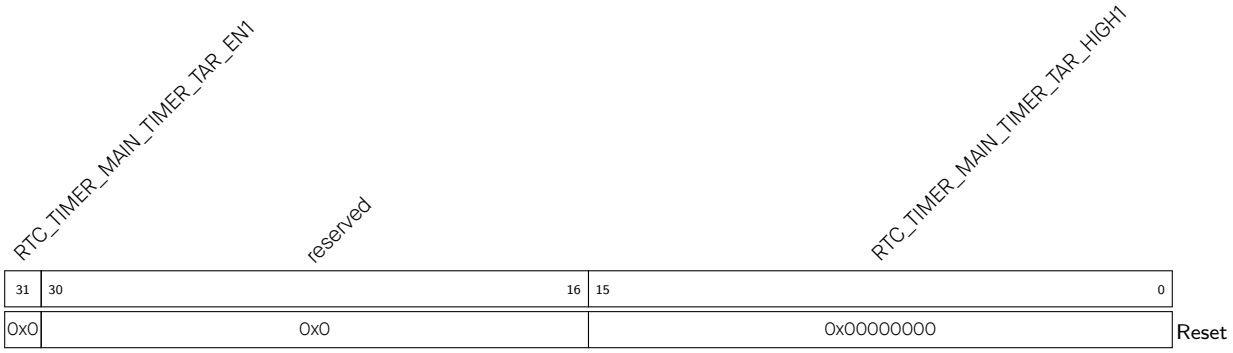
RTC_TIMER_MAIN_TIMER_TAR_HIGH0 Configures the high 16 bits of the target count value 0 (48 bits total) of the RTC timer. (R/W)

Register 11.44. RTC_TIMER_TAR1_LOW_REG (0x0008)



RTC_TIMER_MAIN_TIMER_TAR_LOW1 Configures the low 32 bits of the target count value 1 (48 bits total) of the RTC timer. (R/W)

Register 11.45. RTC_TIMER_TAR0_HIGH_REG (0x000C)



RTC_TIMER_MAIN_TIMER_TAR_EN1 Configures whether to generate interrupts for the target count value 1 of the RTC timer.

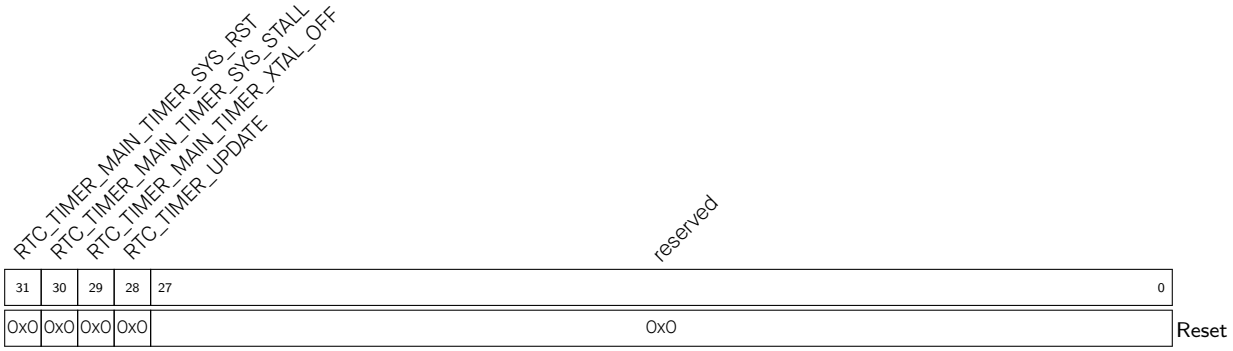
0: Do not generate interrupts

1: Generate interrupts

(R/W)

RTC_TIMER_MAIN_TIMER_TAR_HIGH1 Configures the high 16 bits of the target count value 1 (48 bits total) of the RTC timer. (R/W)

Register 11.46. RTC_TIMER_UPDATE_REG (0x0010)



RTC_TIMER_MAIN_TIMER_SYS_RST Configures whether to trigger RTC timer upon system reset.

0: Do not trigger

1: Trigger

(R/W)

RTC_TIMER_MAIN_TIMER_SYS_STALL Configures whether to trigger RTC timer when the CPU enters or exits stall state.

0: Do not trigger

1: Trigger

(R/W)

RTC_TIMER_MAIN_TIMER_XTAL_OFF Configures whether to trigger RTC timer when PMU powers up or down the 40 MHz crystal.

0: Do not trigger

1: Trigger

(R/W)

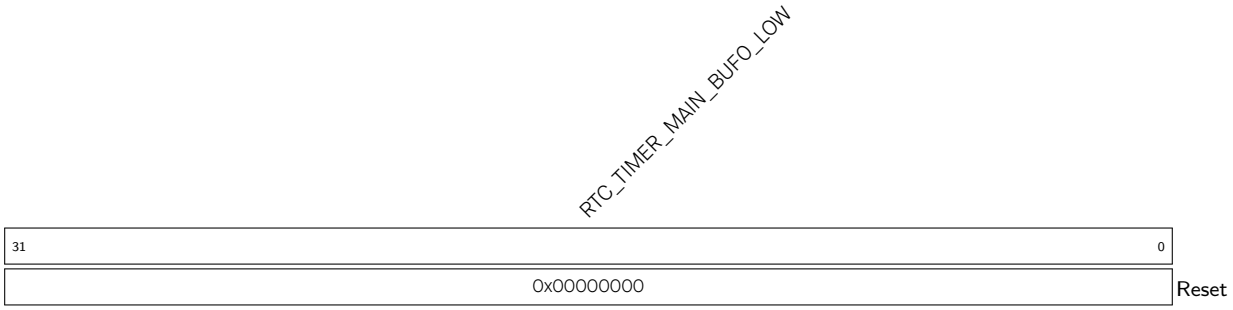
RTC_TIMER_UPDATE Configures whether to trigger RTC timer by software.

0: Do not trigger

1: Trigger

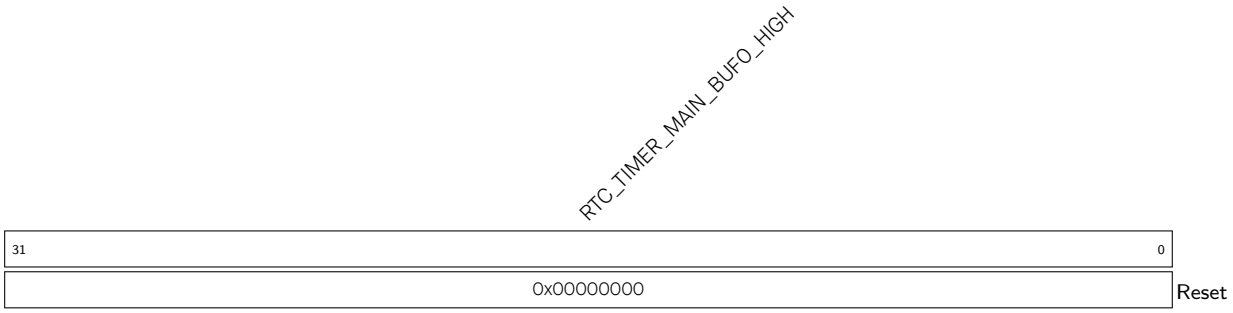
(R/W)

Register 11.47. RTC_TIMER_MAIN_BUF0_LOW_REG (0x0014)



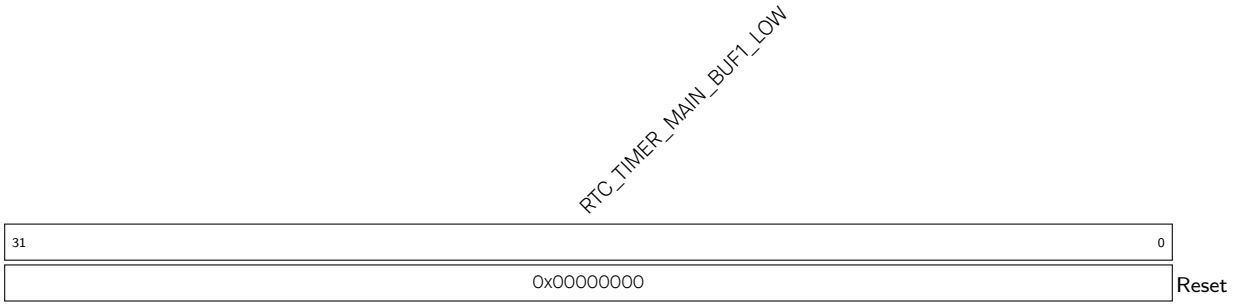
RTC_TIMER_MAIN_BUF0_LOW Register group 0 records the count value of the RTC timer, bit0 to bit31. (RO)

Register 11.48. RTC_TIMER_MAIN_BUF0_HIGH_REG (0x0018)



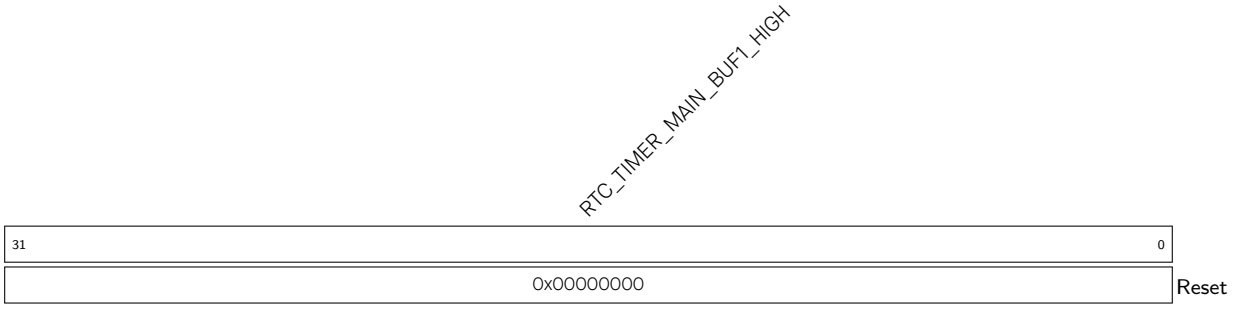
RTC_TIMER_MAIN_BUF0_HIGH Register group 0 records the count value of the RTC timer, bit32 to bit47, corresponding to bit[0:15]. (RO)

Register 11.49. RTC_TIMER_MAIN_BUF1_LOW_REG (0x001C)



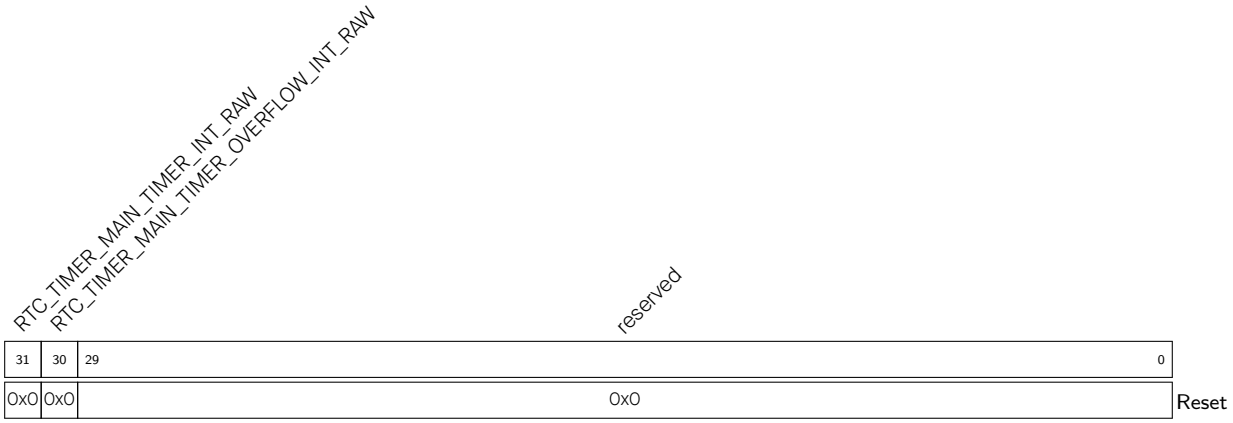
RTC_TIMER_MAIN_BUF1_LOW Register group 1 records the count value of the RTC timer, bit0 to bit31. (RO)

Register 11.50. RTC_TIMER_MAIN_BUF1_HIGH_REG (0x0020)



RTC_TIMER_MAIN_BUF1_HIGH Register group 1 records the count value of the RTC timer, bit32 to bit47, corresponding to bit[0:15]. (RO)

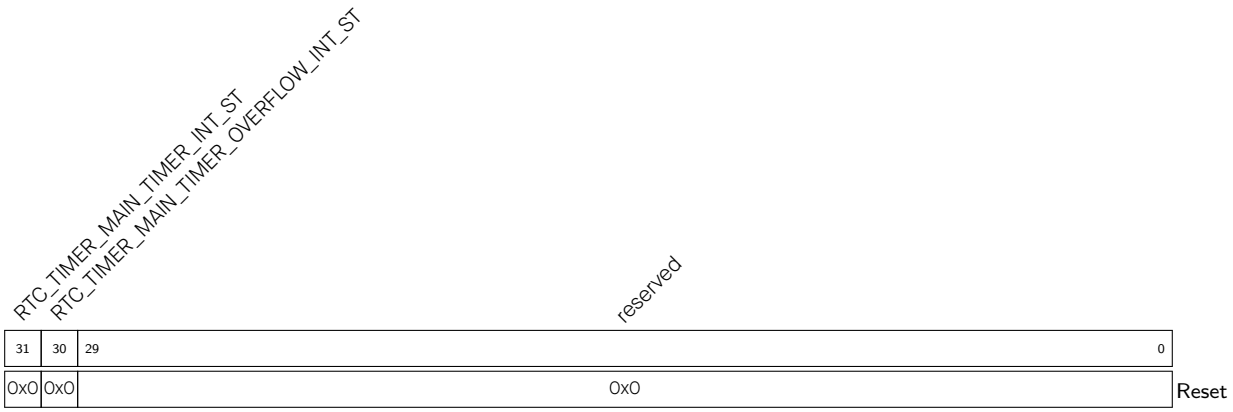
Register 11.51. RTC_TIMER_INT_RAW_REG (0x0028)



RTC_TIMER_MAIN_TIMER_INT_RAW The raw interrupt status of RTC_TIMER_MAIN_TIMER_INT. (RO)

RTC_TIMER_MAIN_TIMER_OVERFLOW_INT_RAW The raw interrupt status of RTC_TIMER_MAIN_TIMER_OVERFLOW_INT. (RO)

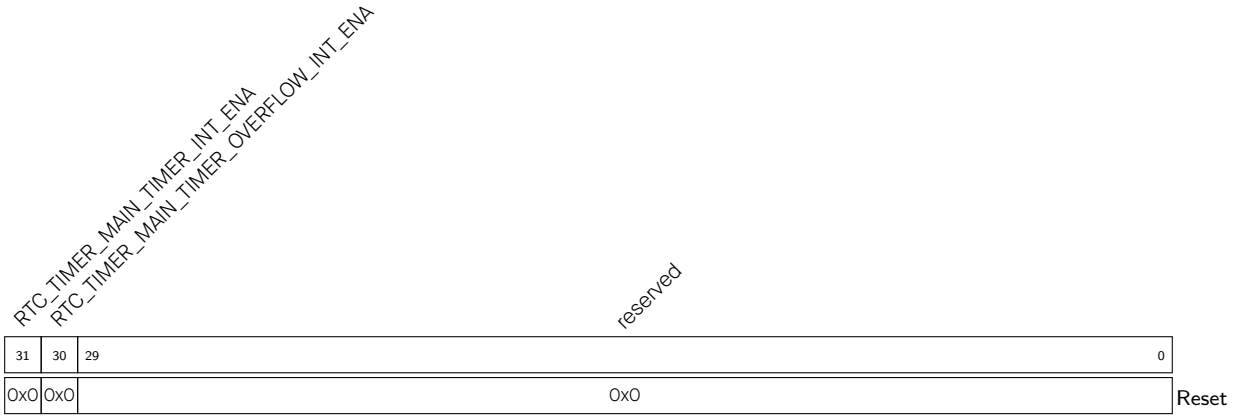
Register 11.52. RTC_TIMER_INT_ST_REG (0x002C)



RTC_TIMER_MAIN_TIMER_INT_ST The masked interrupt status of RTC_TIMER_MAIN_TIMER_INT. (RO)

RTC_TIMER_MAIN_TIMER_OVERFLOW_INT_ST The masked interrupt status of RTC_TIMER_MAIN_TIMER_OVERFLOW_INT. (RO)

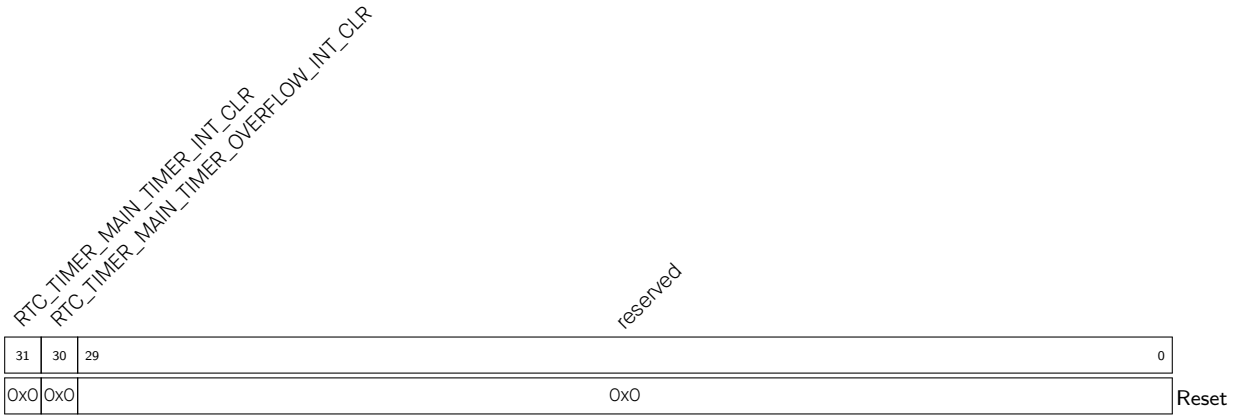
Register 11.53. RTC_TIMER_INT_ENA_REG (0x0030)



RTC_TIMER_MAIN_TIMER_INT_ENA Write 1 to enable RTC_TIMER_MAIN_TIMER_INT. (RO)

RTC_TIMER_MAIN_TIMER_OVERFLOW_INT_ENA Write 1 to enable RTC_TIMER_MAIN_TIMER_OVERFLOW_INT. (RO)

Register 11.54. RTC_TIMER_INT_CLR_REG (0x0034)



RTC_TIMER_MAIN_TIMER_INT_CLR Write 1 to clear RTC_TIMER_MAIN_TIMER_INT. (RO)

RTC_TIMER_MAIN_TIMER_OVERFLOW_INT_CLR Write 1 to clear RTC_TIMER_MAIN_TIMER_OVERFLOW_INT. (RO)

Chapter 12

System Timer (SYSTIMER)

12.1 Overview

ESP32-H2 provides a 52-bit 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.

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 on Figure 12.1-1.

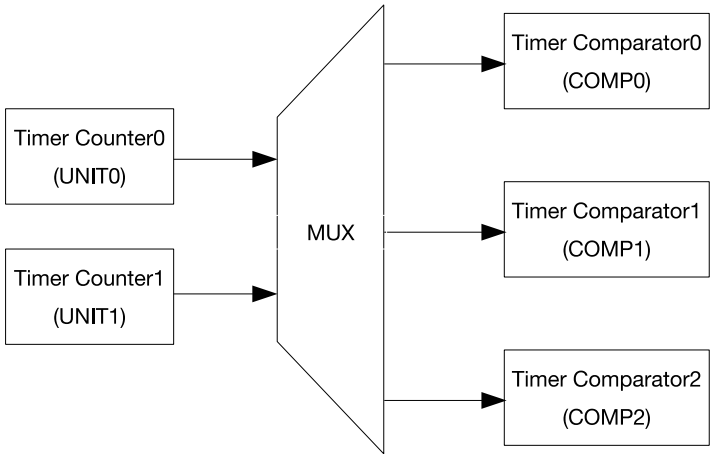


Figure 12.1-1. System Timer Structure

12.2 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
- XTAL_CLK (32 MHz) or RC_FAST_CLK as the clock source of CNT_CLK
- 52-bit alarm values (t) and 26-bit alarm periods (δt)
- Two modes to generate alarms:
  - Target mode: only a one-time alarm is generated based on the alarm value (t)
  - Period mode: periodic alarms are generated based on the alarm period (δt)

- Three comparators generating three independent interrupts based on configured alarm value (t) or alarm period (δt)
- Software configuring the reference count value. For example, the system timer is able to load back the sleep time recorded by RTC timer via software after Light-sleep
- Able to stall or continue running when CPU stalls or enters on-chip-debugging mode
- Alarm for Event Task Matrix (ETM) event

12.3 Clock Source Selection

The counters and comparators use XTAL_CLK or RC_FAST_CLK as clock source. 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 divided by 2, a $f_{XTAL\_CLK}/2$ clock is generated in one count cycle, which is 16 MHz, i.e., the CNT_CLK in Figure 12.4-1. The timer counter is incremented by 1/16 μs on each CNT_CLK cycle. If RC_FAST_CLK is selected as the clock source, then it will not be divided and will directly connect to the SYSTIMER controller.

Software operation such as configuring registers is clocked by APB_CLK. For more information about APB_CLK, see Chapter 7 *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 7 *Reset and Clock*.

12.4 Functional Description

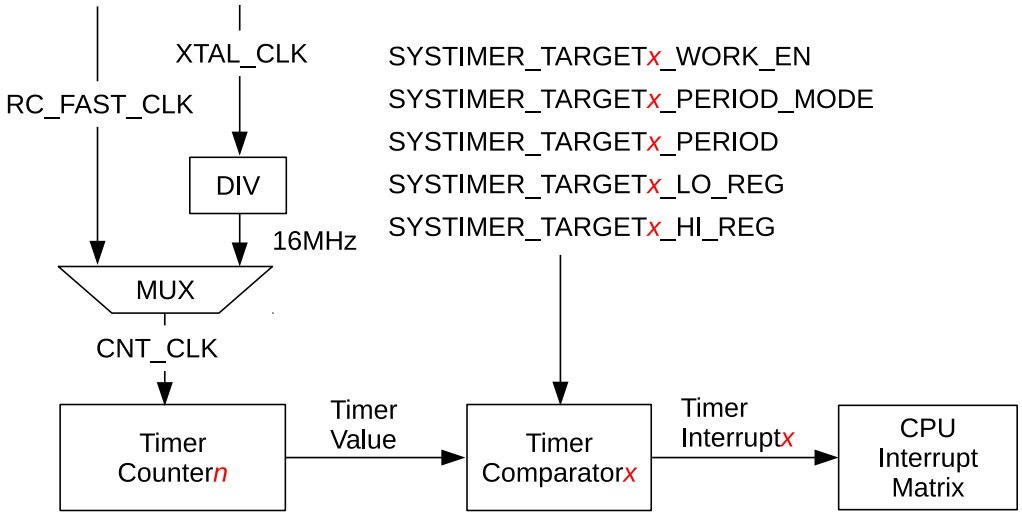


Figure 12.4-1. System Timer Alarm Generation

Figure 12.4-1 shows the procedure to generate alarms in the 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.

12.4.1 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 two 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*_COREO_STALL_EN: if this bit is set, the counter UNIT*n* stops when CPU is stalled. The counter continues its counting after CPU resumes.

The configuration of the two bits to control the counter UNIT*n* is shown below, assuming that CPU is stalled.

Table 12.4-1. UNIT*n* Configuration Bits

SYSTIMER_TIMER_ UNIT*n*_WORK_EN	SYSTIMER_TIMER_ UNIT*n*_COREO_STALL_EN	Counter UNIT*n*
0	x *	Not at work
1	1	Stop counting, but will continue its counting after CPU resumes
1	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 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 change immediately. If UNIT*n* is at work, the counter will continue to count up from the newly 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 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.

12.4.2 Comparator and Alarm

The system timer has three 52-bit comparators, shown as COMP*x* (*x* = 0, 1, or 2). The comparators can generate independent interrupts based on different alarm values (t) or alarm periods (δt).

Configure SYSTIMER_TARGET*x*_PERIOD_MODE to choose from the two alarm modes for each COMP*x*:

- 1: period mode
- 0: target mode

In period mode, the alarm period (δt) is provided by register SYSTIMER_TARGET*x*_PERIOD. Assuming that current count value is t1, when it reaches (t1 + δt), an alarm interrupt will be generated. When the count value reaches (t1 + 2×δt), another alarm interrupt 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 period mode, only one alarm interrupt is generated in target mode.

SYSTIMER_TARGET*x*_TIMER_UNIT_SEL is used to choose the count value from which timer counter to be compared to generate alarms:

- 1: Use the count value from UNIT*1*
- 0: Use the count value from UNIT*0*

Finally, set SYSTIMER_TARGET*x*_WORK_EN and COMP*x* starts to compare the count value:

- In target mode, COMP*x* compares it with the alarm value (t).
- In period mode, COMP*x* compares it with the alarm period (t1 + n×δt).

An alarm is generated when the count value equals to the alarm value (t) in target mode or to the start value (t1) + n×δt (n = 1, 2, 3...) in period mode. But if the alarm value (t) set in registers is less than the current count value, i.e., the target has already passed, when the current count value is 0 ~ 2^{51} - 1 larger than the alarm value (t), 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 12.4-2. Trigger Point

Relationship Between t_c and t_t	Trigger Point
t_c- t_t <= 0	t_c = t_t, an alarm is triggered.
0 <= t_c - t_t < 2^{51} - 1 (t_c < 2^{51} and t_t < 2^{51}, or t_c >= 2^{51} and t_t >= 2^{51})	An alarm is triggered immediately.
t_c - t_t >= 2^{51} - 1	t_c overflows after counting to its maximum value 52'hfffffffffffff, and then starts counting up from 0. When its value reaches t_t, an alarm is triggered.

12.4.3 Event Task Matrix

The system timer on ESP32-H2 supports the Event Task Matrix (ETM) function, which allows 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 10 Event Task Matrix (SOC_ETM).

The system timer can generate the following ETM event:

- SYSTIMER_EVT_CNT_CMP*x*: Indicates the alarm pulses generated by COMP*x*.

When SYSTIMER_ETM_EN is set to 1, the alarm pulses can trigger the ETM event.

12.4.4 Synchronization Operation

The clock (APB_CLK) used in software operation is different from the clock (CNT_CLK) driving timer counters and comparators. Therefore, some configuration registers need to be synchronized. A complete synchronization action takes two steps:

1. Software writes specific values to configuration fields, see the first column in Table 12.4-3.
2. Software writes 1 to corresponding bits to start synchronization, see the second column in Table 12.4-3.

Table 12.4-3. Synchronization Operation for Configuration Registers

Configuration Fields	Synchronization Enable Bit
SYSTIMER_TIMER_UNIT*n*_LOAD_LO SYSTIMER_TIMER_UNIT*n*_LOAD_HI	SYSTIMER_TIMER_UNIT*n*_LOAD
SYSTIMER_TARGET*x*_PERIOD SYSTIMER_TIMER_TARGET*x*_HI SYSTIMER_TIMER_TARGET*x*_LO	SYSTIMER_TIMER_COMP*x*_LOAD

Synchronization is also needed for reading some status registers since the timer counter related status have a different clock from APB_CLK. A complete synchronization action takes three steps:

1. Software writes 1 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.

12.4.5 Interrupt

Each comparator has one alarm interrupt respectively, named as SYSTIMER_TARGET*x*_INT. The interrupt signal is asserted high when the comparator starts to alarm. Until any software clears the interrupt, it remains high. To enable interrupts, set the bit SYSTIMER_TARGET*x*_INT_ENA.

12.5 Programming Procedure

When configuring COMP*x* and UNIT*n*, please ensure the corresponding COMP and UNIT are at work.

12.5.1 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's 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.

12.5.2 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, see Section 12.5.1. 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.

12.5.3 Configure Periodic Alarms in Period Mode

1. Set SYSTIMER_TARGET*x*_TIMER_UNIT_SEL to select the counter (UNIT*0* or UNIT*1*) used for comparison with COMP*x*.
2. Set an alarm period (δt), and fill it into SYSTIMER_TARGET*x*_PERIOD.
3. Set SYSTIMER_TIMER_COMP*x*_LOAD to synchronize the alarm period (δt) to COMP*x*, i.e., load the alarm period (δt) to COMP*x*.
4. Clear and then set SYSTIMER_TARGET*x*_PERIOD_MODE to configure COMP*x* into period mode.
5. Set SYSTIMER_TARGET*x*_WORK_EN to enable the selected COMP*x*. COMP*x* starts comparing the count value with the sum of (start value + n×δt) (n = 1, 2, 3...).
6. Set SYSTIMER_TARGET*x*_INT_ENA to enable the timer interrupt. A SYSTIMER_TARGET*x*_INT interrupt is triggered when Unit*n* reaches start value + n×δt (n = 1, 2, 3...) set in Step 2.

12.5.4 Update After Light-sleep

1. Configure RTC timer before the chip goes into Light-sleep mode, to record the exact sleep time. For more information, see Chapter 11 *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 12.5.1.
4. Convert the time value recorded by the 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.

12.6 Register Summary

The addresses in this section are relative to system timer base address provided in Table 4.3-2 in Chapter 4 *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

12.7 Registers

The addresses in this section are relative to system timer base address provided in Table 4.3-2 in Chapter 4 *System and Memory*.

Register 12.1. SYSTIMER_CONF_REG (0x0000)

Bit(s)	Field	Reset
31	SYSTIMER_CLK_EN	0
30	SYSTIMER_TIMER_UNIT0_WORK_EN	1
29	SYSTIMER_TIMER_UNIT1_WORK_EN	0
28	SYSTIMER_TIMER_UNIT0_CORE0_STALL_EN	0
27	SYSTIMER_TIMER_UNIT0_CORE1_STALL_EN	0
26	SYSTIMER_TIMER_UNIT1_CORE0_STALL_EN	1
25	SYSTIMER_TIMER_UNIT1_CORE1_STALL_EN	1
24	SYSTIMER_TARGET0_WORK_EN	0
23	SYSTIMER_TARGET1_WORK_EN	0
22	SYSTIMER_TARGET2_WORK_EN	0
21:2	(reserved)	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1	SYSTIMER_ETM_EN	0
0	(reserved)	0

SYSTIMER_ETM_EN Configures whether or not to enable generation of ETM events.

0: No effect

1: Enable

(R/W)

SYSTIMER_TARGET2_WORK_EN Configures whether or not to enable COMP2.

0: Disable

1: Enable

(R/W)

SYSTIMER_TARGET1_WORK_EN Configures whether or not to enable COMP1. See details in SYSTIMER_TARGET2_WORK_EN. (R/W)

SYSTIMER_TARGET0_WORK_EN Configures whether or not to enable COMP0. See details in SYSTIMER_TARGET2_WORK_EN. (R/W)

SYSTIMER_TIMER_UNIT1_CORE1_STALL_EN Configures whether or not UNIT1 is stalled when CPU1 is stalled.

0: No effect

1: UNIT1 is stalled

(R/W)

SYSTIMER_TIMER_UNIT1_CORE0_STALL_EN Configures whether or not UNIT1 is stalled when CPU0 is stalled. See details in SYSTIMER_TIMER_UNIT1_CORE1_STALL_EN. (R/W)

SYSTIMER_TIMER_UNIT0_CORE1_STALL_EN Configures whether or not UNIT0 is stalled when CPU1 is stalled. See details in SYSTIMER_TIMER_UNIT1_CORE1_STALL_EN. (R/W)

SYSTIMER_TIMER_UNIT0_CORE0_STALL_EN Configures whether or not UNIT0 is stalled when CPU0 is stalled. See details in SYSTIMER_TIMER_UNIT1_CORE1_STALL_EN. (R/W)

Continued on the next page...

Register 12.1. SYSTIMER_CONF_REG (0x0000)

Continued from the previous page...

SYSTIMER_TIMER_UNIT1_WORK_EN Configures whether or not to enable UNIT1.

0: No effect

1: Enable

(R/W)

SYSTIMER_TIMER_UNIT0_WORK_EN Configures whether or not to enable UNIT0.

0: No effect

1: Enable

(R/W)

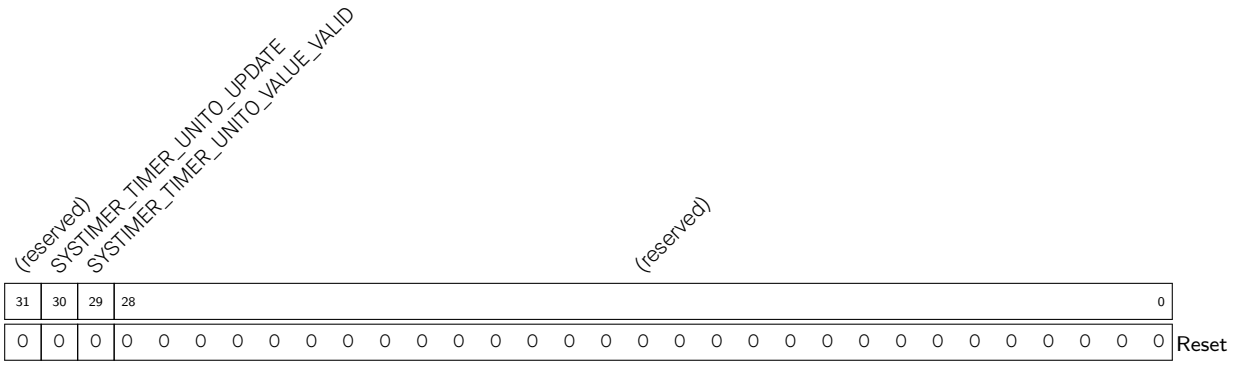
SYSTIMER_CLK_EN Configures register clock gating.

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 12.2. SYSTIMER_UNIT0_OP_REG (0x0004)



SYSTIMER_TIMER_UNIT0_VALUE_VALID Represents UNIT0 value is synchronized and valid. (R/SS/WTC)

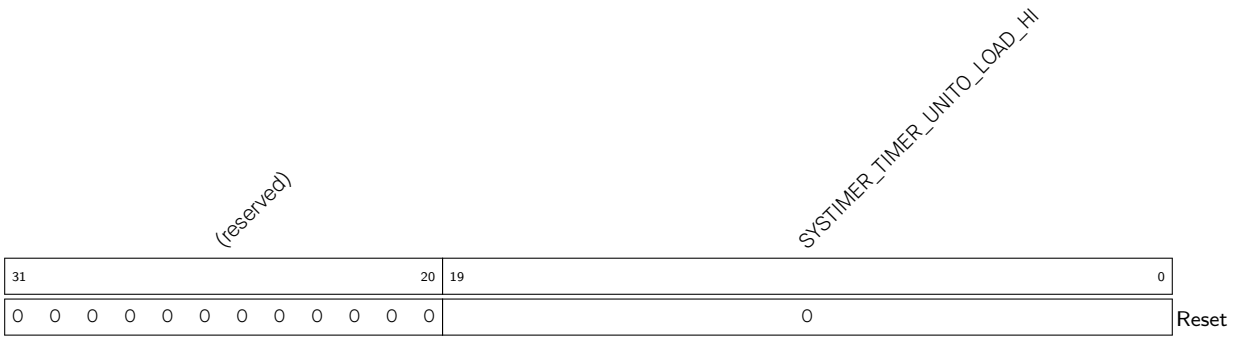
SYSTIMER_TIMER_UNIT0_UPDATE Configures whether or not to update timer UNIT0, i.e., reads the UNIT0 count value to SYSTIMER_TIMER_UNIT0_VALUE_HI and SYSTIMER_TIMER_UNIT0_VALUE_LO.

0: No effect

1: Update timer UNIT0

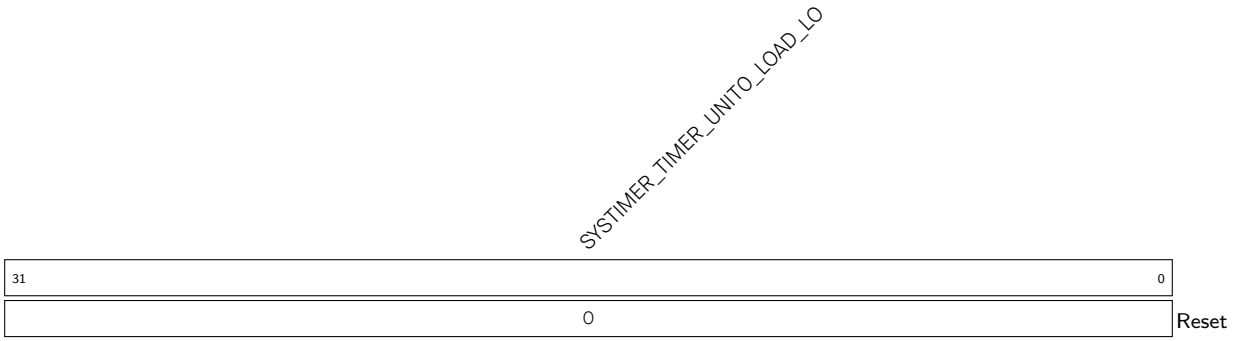
(WT)

Register 12.3. SYSTIMER_UNIT0_LOAD_HI_REG (0x000C)



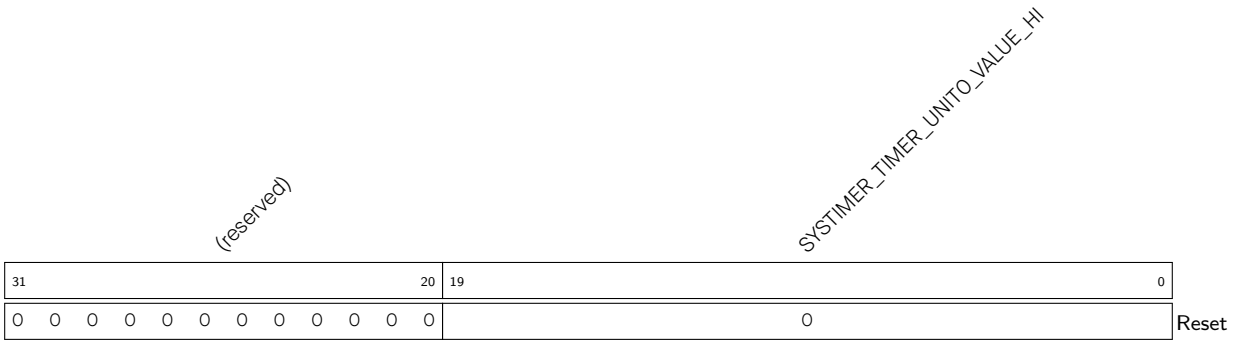
SYSTIMER_TIMER_UNIT0_LOAD_HI Configures the value to be loaded to UNIT0, high 20 bits. (R/W)

Register 12.4. SYSTIMER_UNIT0_LOAD_LO_REG (0x0010)



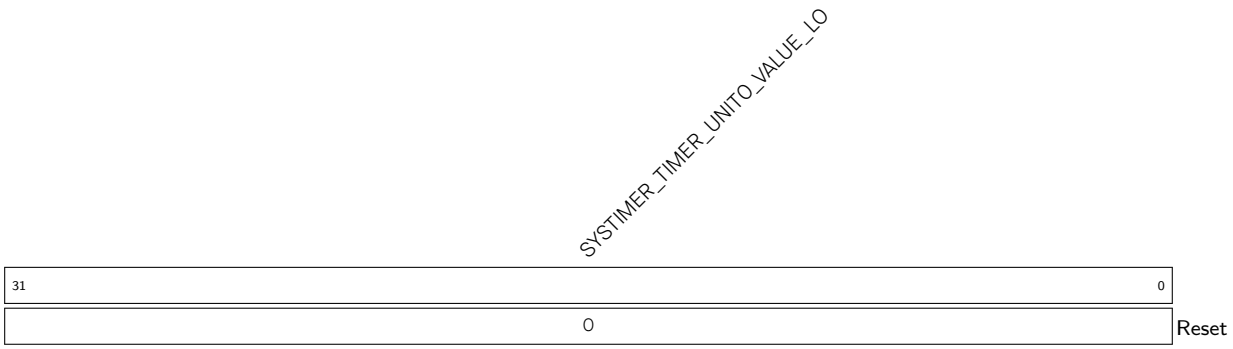
SYSTIMER_TIMER_UNIT0_LOAD_LO Configures the value to be loaded to UNIT0, low 32 bits. (R/W)

Register 12.5. SYSTIMER_UNIT0_VALUE_HI_REG (0x0040)



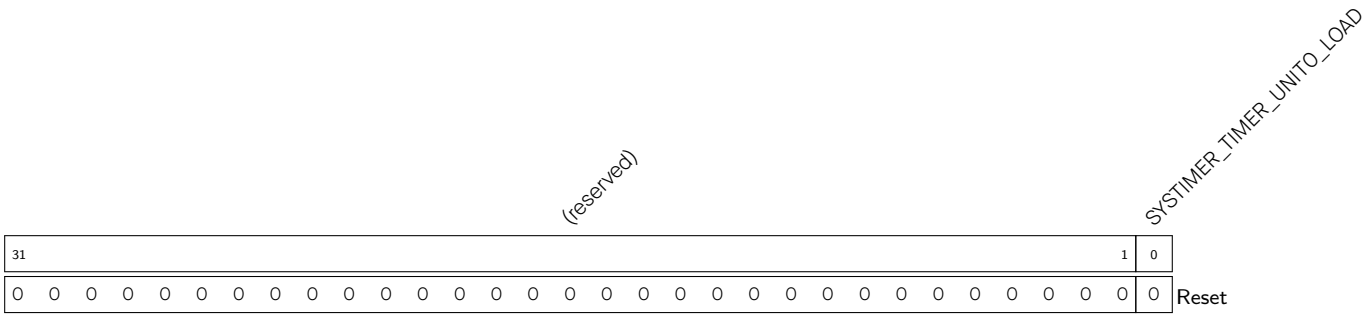
SYSTIMER_TIMER_UNIT0_VALUE_HI Represents UNIT0 read value, high 20 bits. (RO)

Register 12.6. SYSTIMER_UNIT0_VALUE_LO_REG (0x0044)



SYSTIMER_TIMER_UNIT0_VALUE_LO Represents UNIT0 read value, low 32 bits. (RO)

Register 12.7. SYSTIMER_UNIT0_LOAD_REG (0x005C)



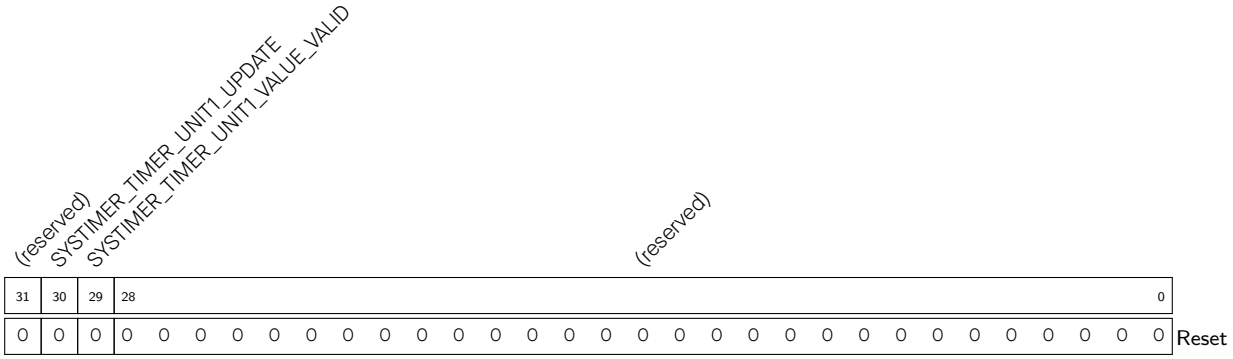
SYSTIMER_TIMER_UNIT0_LOAD Configures whether or not to reload the value of UNIT0, i.e., reloads the values of SYSTIMER_TIMER_UNIT0_VALUE_HI and SYSTIMER_TIMER_UNIT0_VALUE_LO to UNIT0.

0: No effect

1: Reload the value of UNIT0

(WT)

Register 12.8. SYSTIMER_UNIT1_OP_REG (0x0008)



SYSTIMER_TIMER_UNIT1_VALUE_VALID Represents UNIT1 value is synchronized and valid. (R/SS/WTC)

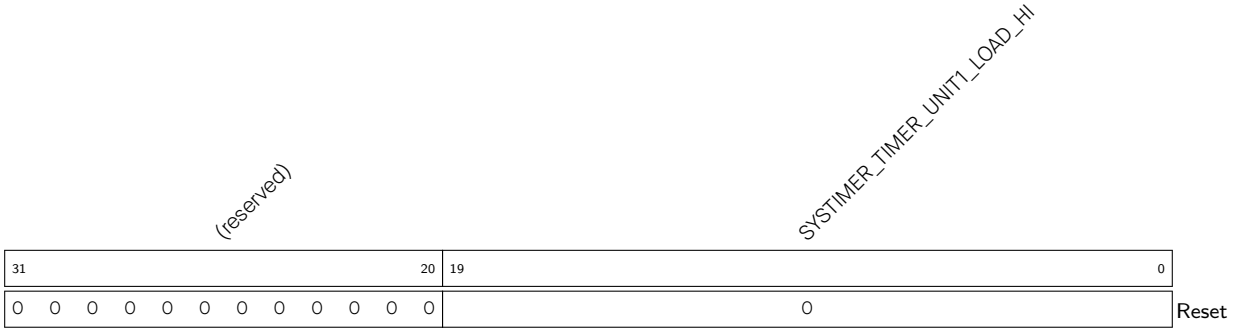
SYSTIMER_TIMER_UNIT1_UPDATE Configures whether or not to update timer UNIT1, i.e., reads the UNIT1 count value to SYSTIMER_TIMER_UNIT1_VALUE_HI and SYSTIMER_TIMER_UNIT1_VALUE_LO.

0: No effect

1: Update timer UNIT1

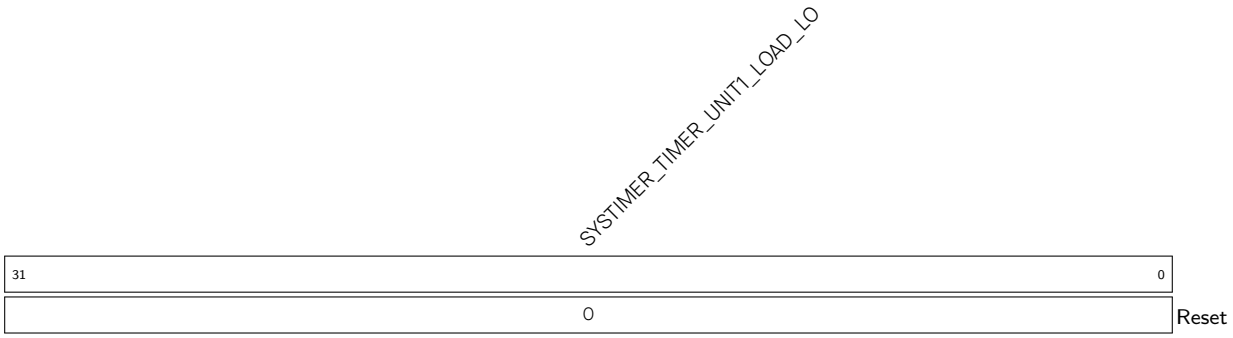
(WT)

Register 12.9. SYSTIMER_UNIT1_LOAD_HI_REG (0x0014)



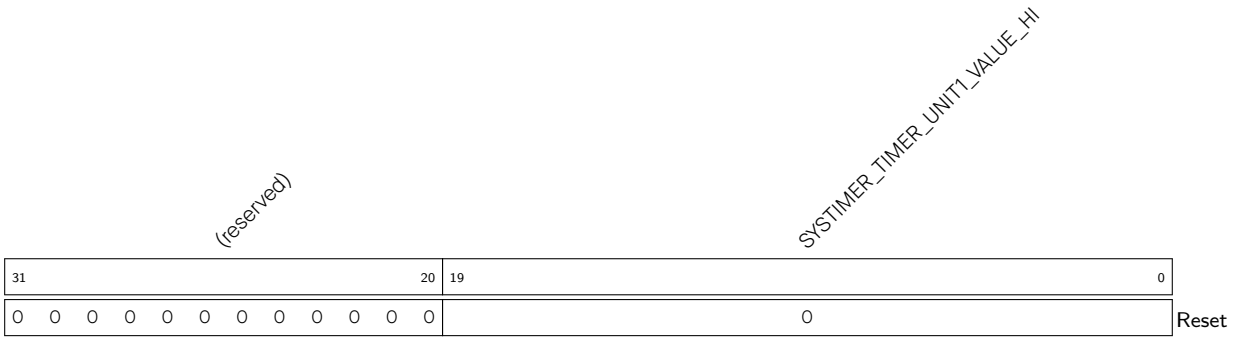
SYSTIMER_TIMER_UNIT1_LOAD_HI Configures the value to be loaded to UNIT1, high 20 bits. (R/W)

Register 12.10. SYSTIMER_UNIT1_LOAD_LO_REG (0x0018)



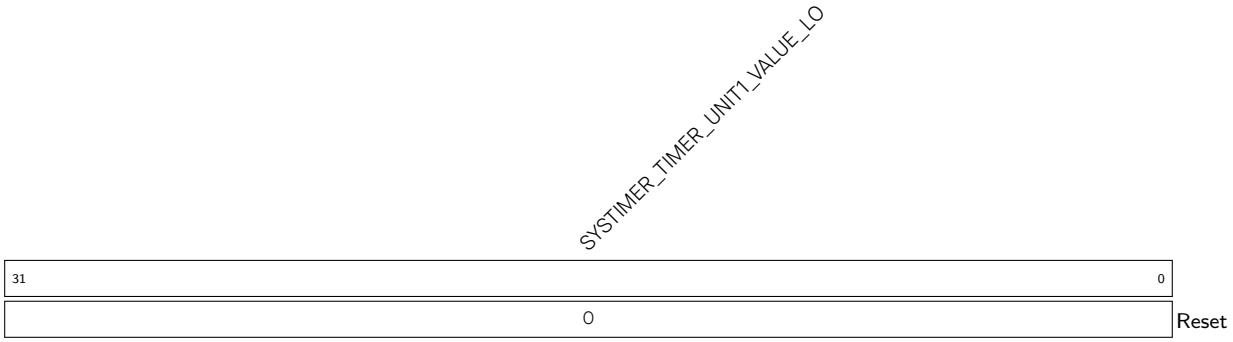
SYSTIMER_TIMER_UNIT1_LOAD_LO Configures the value to be loaded to UNIT1, low 32 bits. (R/W)

Register 12.11. SYSTIMER_UNIT1_VALUE_HI_REG (0x0048)



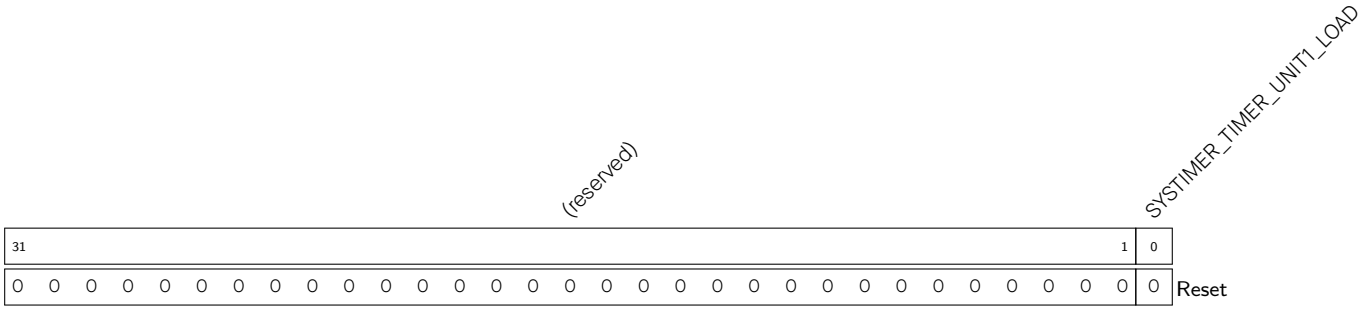
SYSTIMER_TIMER_UNIT1_VALUE_HI Represents UNIT1 read value, high 20 bits. (RO)

Register 12.12. SYSTIMER_UNIT1_VALUE_LO_REG (0x004C)



SYSTIMER_TIMER_UNIT1_VALUE_LO Represents UNIT1 read value, low 32 bits. (RO)

Register 12.13. SYSTIMER_UNIT1_LOAD_REG (0x0060)



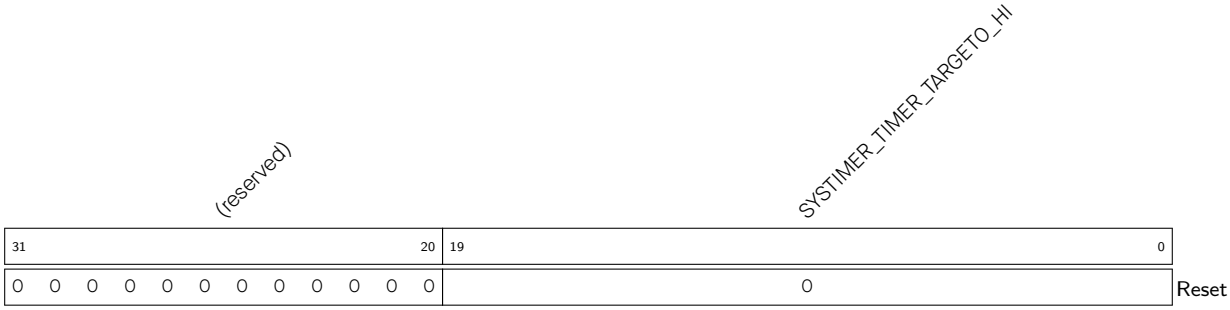
SYSTIMER_TIMER_UNIT1_LOAD Configures whether or not to reload the value of UNIT1, i.e., reload the values of SYSTIMER_TIMER_UNIT1_VALUE_HI and SYSTIMER_TIMER_UNIT1_VALUE_LO to UNIT1.

0: No effect

1: Reload the value of UNIT1

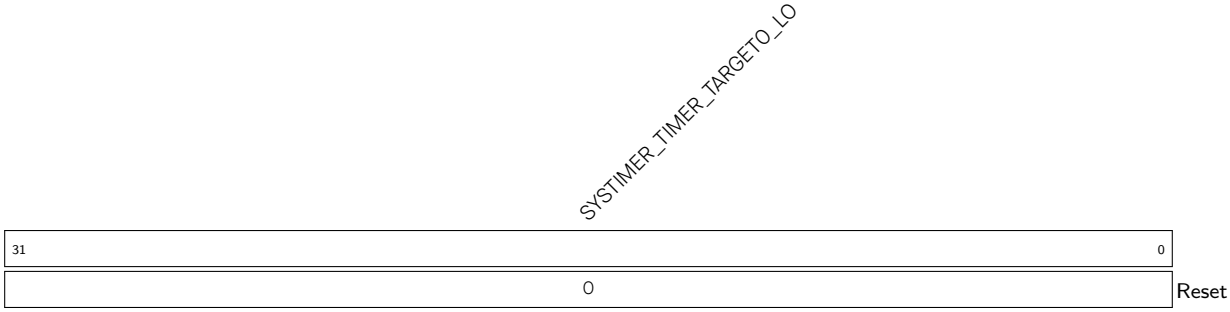
(WT)

Register 12.14. SYSTIMER_TARGET0_HI_REG (0x001C)



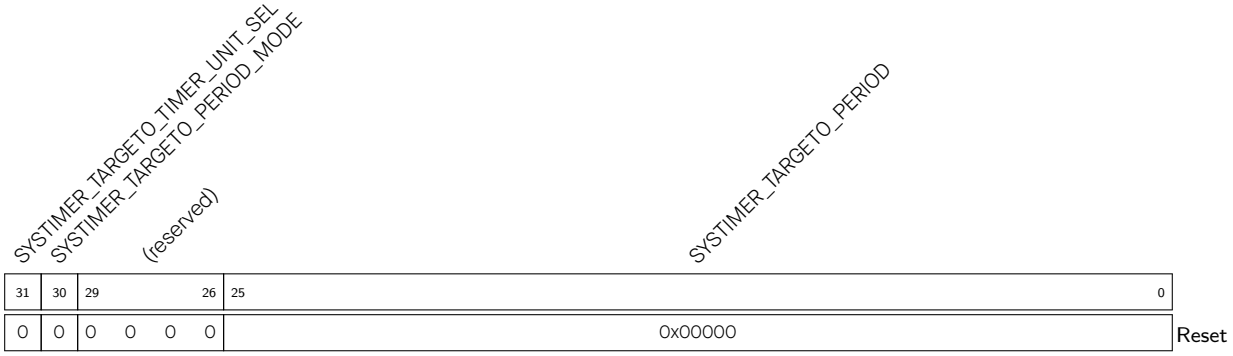
SYSTIMER_TIMER_TARGET0_HI Configures the alarm value to be loaded to COMP0, high 20 bits. (R/W)

Register 12.15. SYSTIMER_TARGET0_LO_REG (0x0020)



SYSTIMER_TIMER_TARGET0_LO Configures the alarm value to be loaded to COMP0, low 32 bits. (R/W)

Register 12.16. SYSTIMER_TARGET0_CONF_REG (0x0034)



SYSTIMER_TARGET0_PERIOD Configures the alarm period to be loaded to COMP0. (R/W)

SYSTIMER_TARGET0_PERIOD_MODE Selects an alarm mode for COMP0.

0: Target mode

1: Period mode

(R/W)

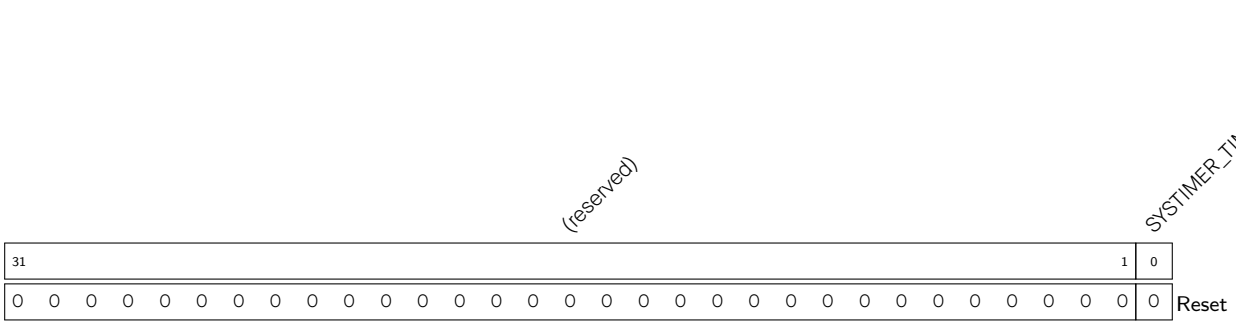
SYSTIMER_TARGET0_TIMER_UNIT_SEL Configures the counter value for comparison with COMP0.

0: Use the count value from UNIT*0*

1: Use the count value from UNIT*1*

(R/W)

Register 12.17. SYSTIMER_COMP0_LOAD_REG (0x0050)



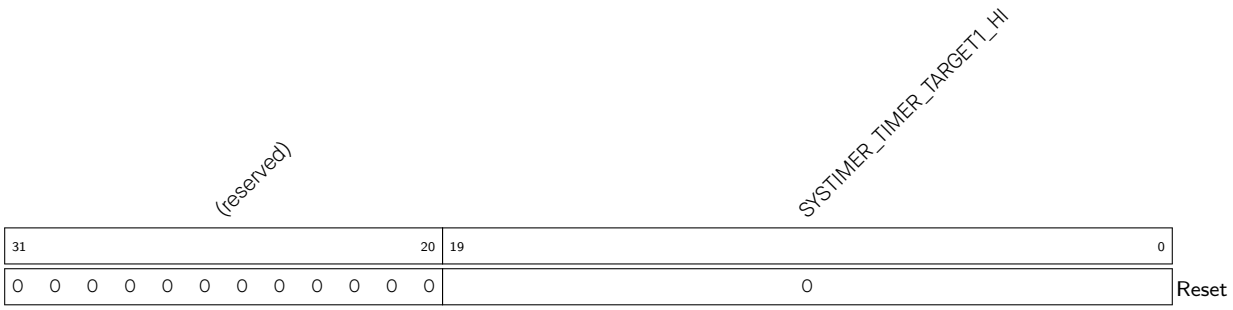
SYSTIMER_TIMER_COMP0_LOAD Configures whether or not to enable COMP0 synchronization, i.e., reload the alarm value/period to COMP0.

0: No effect

1: Enable COMP0 synchronization

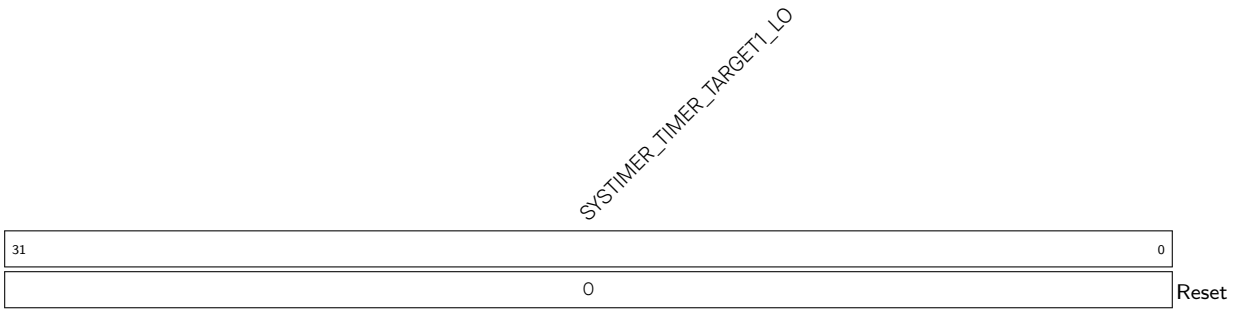
(WT)

Register 12.18. SYSTIMER_TARGET1_HI_REG (0x0024)



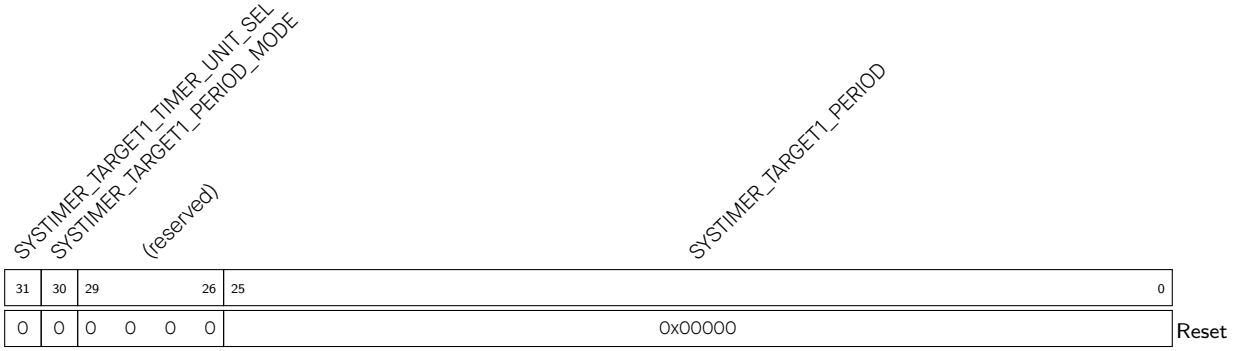
SYSTIMER_TIMER_TARGET1_HI Configures the alarm value to be loaded to COMP1, high 20 bits. (R/W)

Register 12.19. SYSTIMER_TARGET1_LO_REG (0x0028)



SYSTIMER_TIMER_TARGET1_LO Configures the alarm value to be loaded to COMP1, low 32 bits. (R/W)

Register 12.20. SYSTIMER_TARGET1_CONF_REG (0x0038)

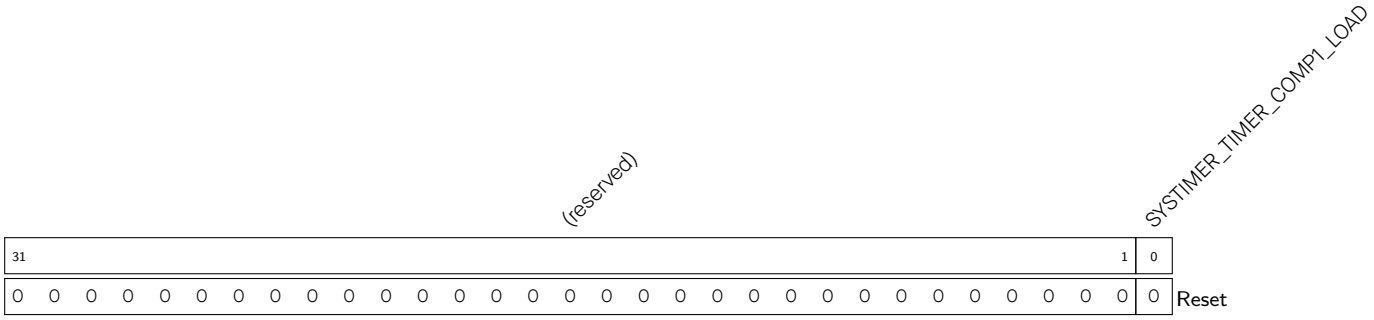


SYSTIMER_TARGET1_PERIOD Configures the alarm period to be loaded to COMP1. (R/W)

SYSTIMER_TARGET1_PERIOD_MODE Selects an alarm mode for COMP1. See details in SYSTIMER_TARGET0_PERIOD_MODE. (R/W)

SYSTIMER_TARGET1_TIMER_UNIT_SEL Chooses the counter value for comparison with COMP1. See details in SYSTIMER_TARGET0_TIMER_UNIT_SEL. (R/W)

Register 12.21. SYSTIMER_COMP1_LOAD_REG (0x0054)



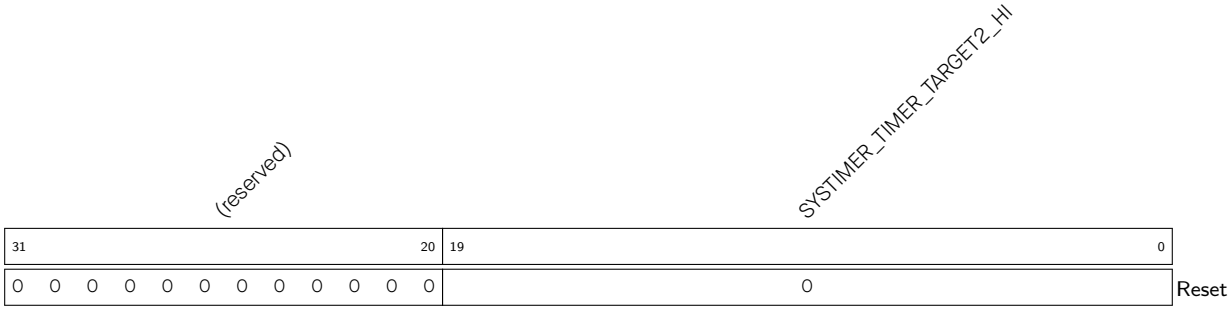
SYSTIMER_TIMER_COMP1_LOAD Configures whether or not to enable COMP1 synchronization, i.e., reload the alarm value/period to COMP1.

0: No effect

1: Enable COMP1 synchronization

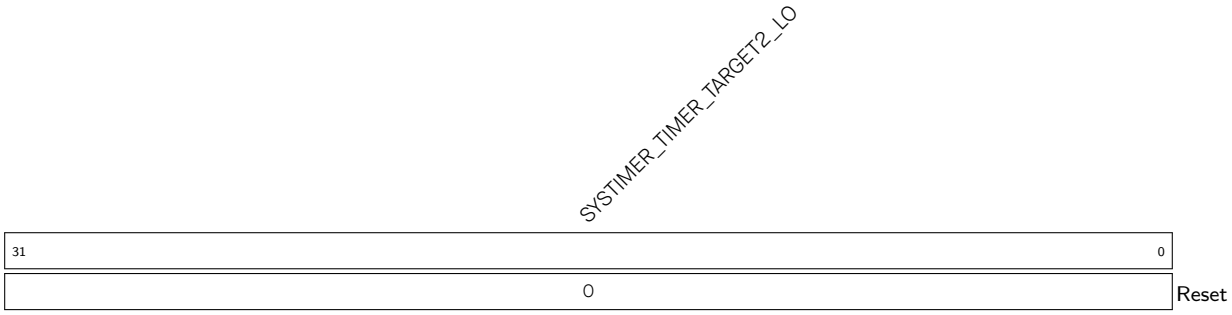
(WT)

Register 12.22. SYSTIMER_TARGET2_HI_REG (0x002C)



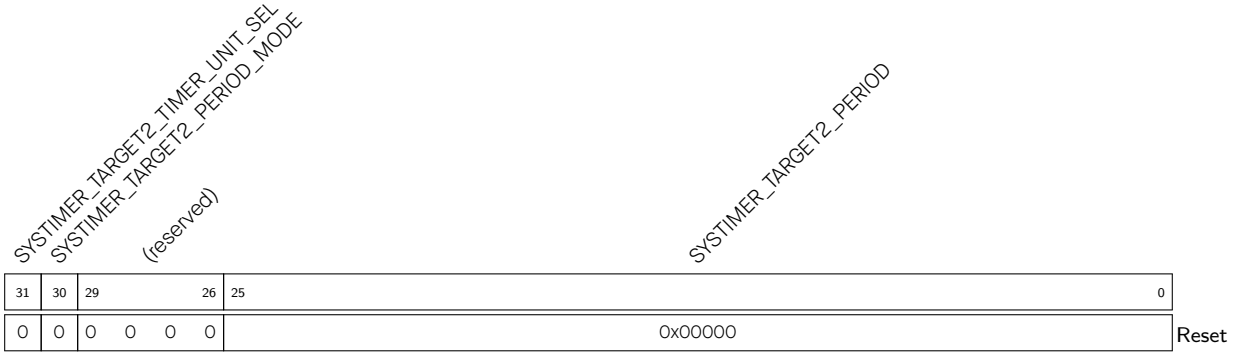
SYSTIMER_TIMER_TARGET2_HI Configures the alarm value to be loaded to COMP2, high 20 bits. (R/W)

Register 12.23. SYSTIMER_TARGET2_LO_REG (0x0030)



SYSTIMER_TIMER_TARGET2_LO Configures the alarm value to be loaded to COMP2, low 32 bits. (R/W)

Register 12.24. SYSTIMER_TARGET2_CONF_REG (0x003C)

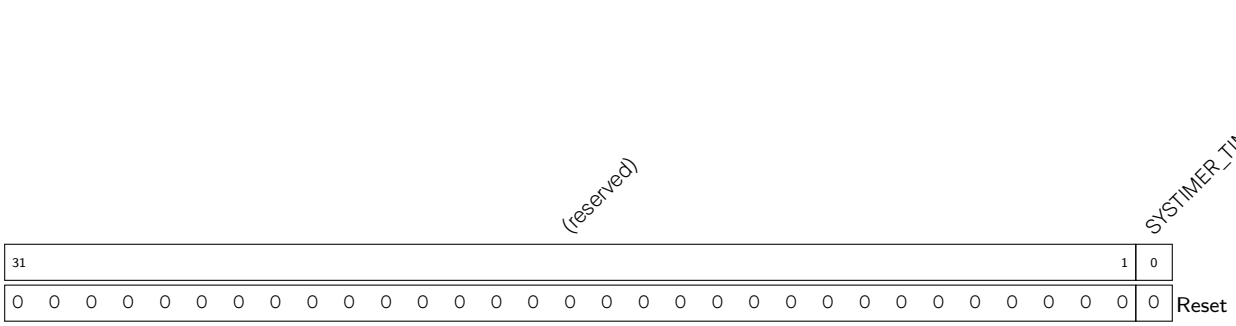


SYSTIMER_TARGET2_PERIOD Configures COMP2 alarm period. (R/W)

SYSTIMER_TARGET2_PERIOD_MODE Selects an alarm mode for COMP2. See details in SYSTIMER_TARGETO_PERIOD_MODE. (R/W)

SYSTIMER_TARGET2_TIMER_UNIT_SEL Chooses the counter value for comparison with COMP2. See details in SYSTIMER_TARGETO_TIMER_UNIT_SEL. (R/W)

Register 12.25. SYSTIMER_COMP2_LOAD_REG (0x0058)



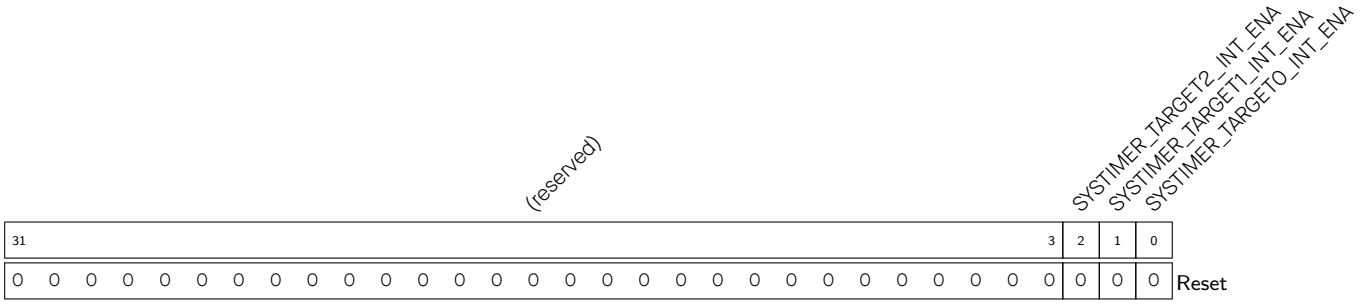
SYSTIMER_TIMER_COMP2_LOAD Configures whether or not to enable COMP2 synchronization, i.e., reload the alarm value/period to COMP2.

0: No effect

1: Enable COMP2 synchronization

(WT)

Register 12.26. SYSTIMER_INT_ENA_REG (0x0064)

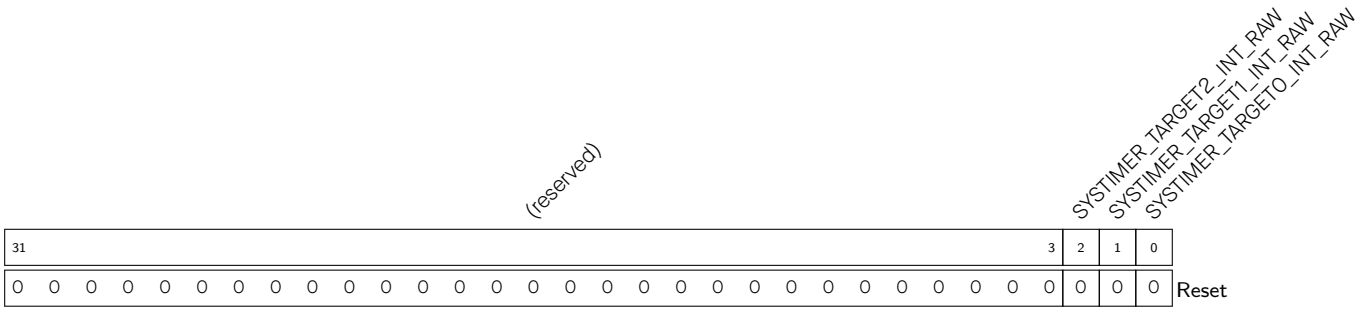


SYSTIMER_TARGET0_INT_ENA Write 1 to enable SYSTIMER_TARGET0_INT. (R/W)

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 12.27. SYSTIMER_INT_RAW_REG (0x0068)

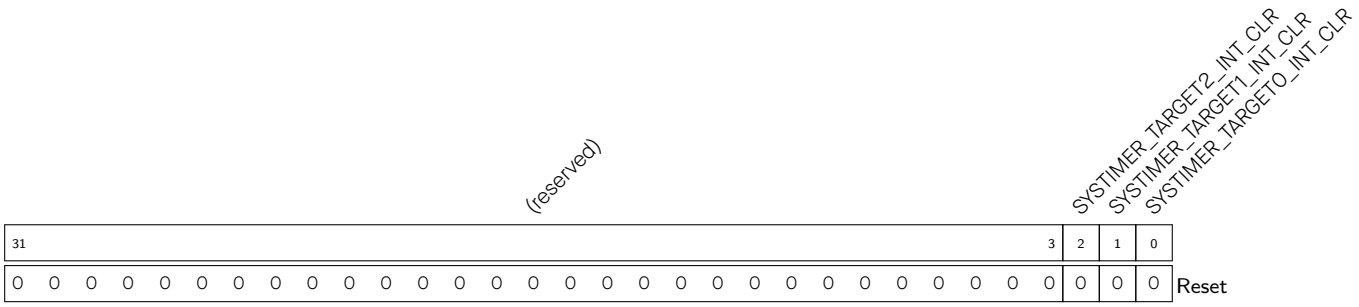


SYSTIMER_TARGET0_INT_RAW The raw interrupt status of SYSTIMER_TARGET0_INT. (R/WTC/SS)

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 12.28. SYSTIMER_INT_CLR_REG (0x006C)

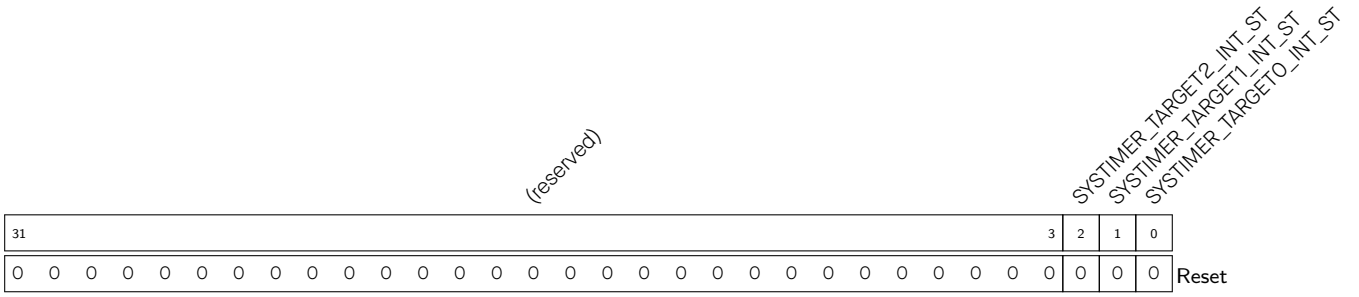


SYSTIMER_TARGET0_INT_CLR Write 1 to clear SYSTIMER_TARGET0_INT. (WT)

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 12.29. SYSTIMER_INT_ST_REG (0x0070)

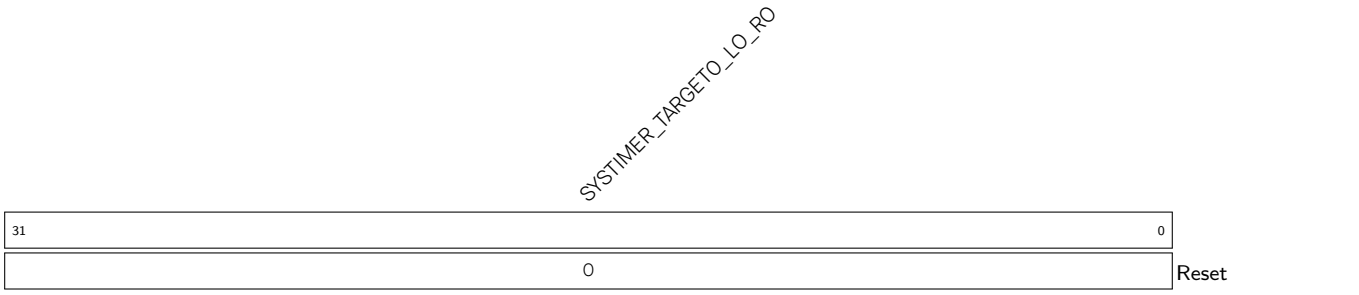


SYSTIMER_TARGET0_INT_ST The interrupt status of SYSTIMER_TARGET0_INT. (RO)

SYSTIMER_TARGET1_INT_ST The interrupt status of SYSTIMER_TARGET1_INT. (RO)

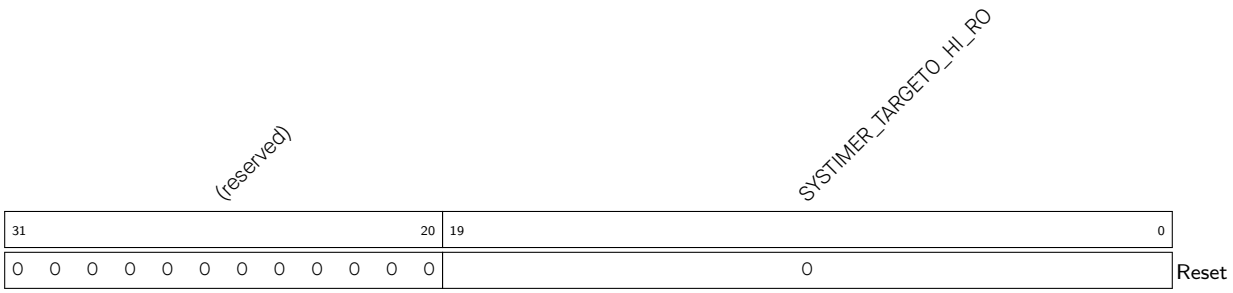
SYSTIMER_TARGET2_INT_ST The interrupt status of SYSTIMER_TARGET2_INT. (RO)

Register 12.30. SYSTIMER_REAL_TARGET0_LO_REG (0x0074)



SYSTIMER_TARGET0_LO_RO Represents the actual target value of COMP0, low 32 bits. (RO)

Register 12.31. SYSTIMER_REAL_TARGET0_HI_REG (0x0078)

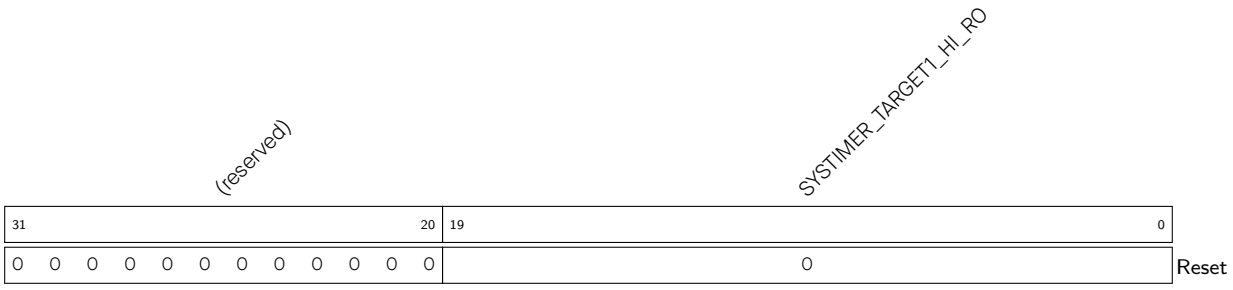


SYSTIMER_TARGET0_HI_RO Represents the actual target value of COMP0, high 20 bits. (RO)

Register 12.32. SYSTIMER_REAL_TARGET1_LO_REG (0x007C)

SYSTIMER_TARGET1_LO_RO Represents the actual target value of COMP1, low 32 bits. (RO)

Register 12.33. SYSTIMER_REAL_TARGET1_HI_REG (0x0080)



SYSTIMER_TARGET1_HI_RO Represents the actual target value of COMP1, high 20 bits. (RO)

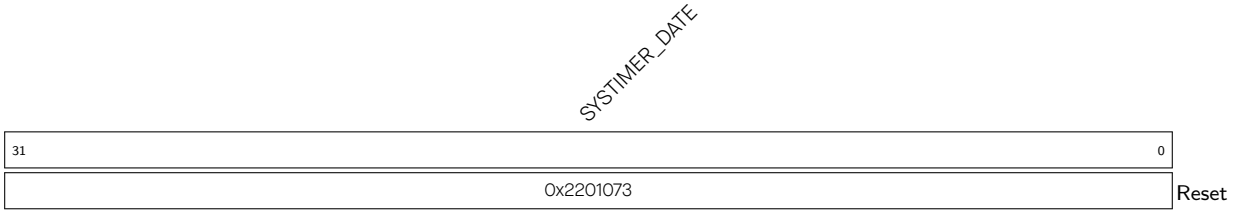
Register 12.34. SYSTIMER_REAL_TARGET2_LO_REG (0x0084)

SYSTIMER_TARGET2_LO_RO Represents the actual target value of COMP2, low 32 bits. (RO)

Register 12.35. SYSTIMER_REAL_TARGET2_HI_REG (0x0088)

SYSTIMER_TARGET2_HI_RO Represents the actual target value of COMP2, high 20 bits. (RO)

Register 12.36. SYSTIMER_DATE_REG (0x00FC)



SYSTIMER_DATE Version control register. (R/W)

Chapter 13

Timer Group (TIMG)

13.1 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 13.1-1, the ESP32-H2 chip contains two timer groups, namely timer group 0 (TIMG*0*) and timer group 1 (TIMG*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.

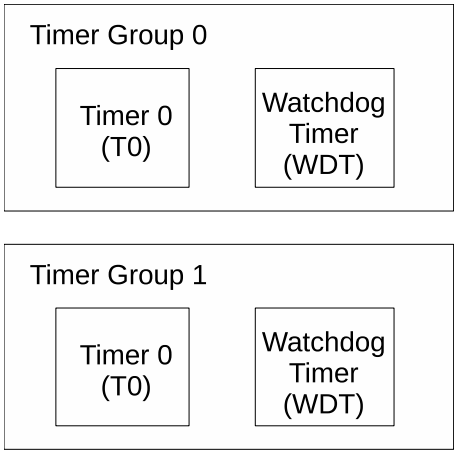


Figure 13.1-1. Timer Group Overview

Note that while the Main System Watchdog Timer registers are described in this chapter, their functional description is included in the Chapter 14 *Watchdog Timers (WDT)*. Therefore, the term "timer" within this chapter refers to the general-purpose timer.

13.2 Features

The timer's features are summarized as follows:

- A 54-bit time-base counter programmable to incrementing or decrementing
- Three clock sources: XTAL_CLK or RC_FAST_CLK or PLL_F48M_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
- RTC slow clock RTC_SLOW_CLK frequency calculation
- Level interrupt generation
- Support several ETM tasks and events

13.3 Functional Description

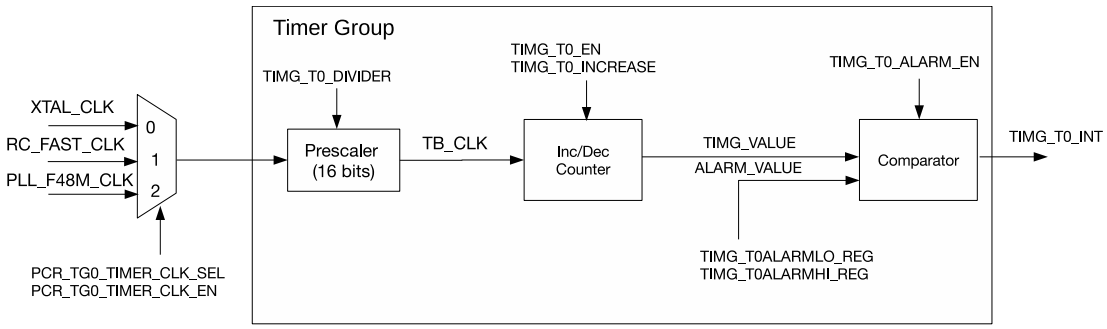


Figure 13.3-1. Timer Group Architecture

Figure 13.3-1 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.

13.3.1 16-bit Prescaler and Clock Selection

Take the T0 in TIMG*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, XTAL_CLK is selected; when the field is 1, RC_FAST_CLK is selected and when the field is 2, PLL_F48M_CLK is selected.
- 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. 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*0*_T0_DIVIDER field.

TIMG*0*_T0_DIVIDER field can be configured as 0 ~ 65535 for a divisor range of 2 ~ 65536. To be more specific, when TIMG*0*_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*0*_T0_DIVIDER field, and then set TIMG*0*_T0_DIVCNT_RST to 1. Meanwhile, the timer must be disabled (i.e., TIMG*0*_T0_EN should be cleared). Otherwise, the result can be unpredictable.

13.3.2 54-bit Time-base Counter

The 54-bit time-base counter is based on TB_CLK and can be configured to increment or decrement via the TIMG*n*_TO_INCREASE field. The time-base counter can be enabled or disabled by setting or clearing the TIMG*n*_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*n*_TO_INCREASE field can be changed no matter whether TIMG*n*_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*n*_TOUPDATE_REG, the current value of the 54-bit timer starts to be latched into the TIMG*n*_TOLO_REG and TIMG*n*_TOHI_REG registers containing the lower 32-bits and higher 22-bits, respectively. When TIMG*n*_TOUPDATE_REG is cleared by hardware, it indicates the latch operation has been completed and current timer value can be read from the TIMG*n*_TOLO_REG and TIMG*n*_TOHI_REG registers. TIMG*n*_TOLO_REG and TIMG*n*_TOHI_REG registers will remain unchanged for the CPU to read in its own time until TIMG*n*_TOUPDATE_REG is written to again.

13.3.3 Alarm Generation

A timer can be configured to trigger an alarm when the timer's current value matches the alarm value. An alarm will cause an interrupt to occur and (optionally) an automatic reload of the timer's current value (see Section 13.3.4).

The 54-bit alarm value is configured using TIMG*n*_TOALARMLO_REG and TIMG*n*_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*n*_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 13.3-1 and Table 13.3-2 show the relationship among 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*n*_TOHI_REG, TIMG*n*_TOLO_REG}
- ALARM_VALUE = {TIMG*n*_TOALARMHI_REG, TIMG*n*_TOALARMLO_REG}

Table 13.3-1. Alarm Generation When Up-Down Counter Increments

Scenario	Range	Alarm
1	ALARM_VALUE − TIMG_VALUE $> 2^{53}$	Triggered
2	0 $<$ ALARM_VALUE − TIMG_VALUE $\leq 2^{53}$	Triggered when the up-down counter counts TIMG_VALUE up to ALARM_VALUE
3	0 $\leq$ TIMG_VALUE − ALARM_VALUE $< 2^{53}$	Triggered
4	TIMG_VALUE − ALARM_VALUE $\geq 2^{53}$	Triggered when the up-down counter restarts counting up from 0 after reaching the timer's maximum value and counts TIMG_VALUE up to ALARM_VALUE

Table 13.3-2. Alarm Generation When Up-Down Counter Decrements

Scenario	Range	Alarm
5	TIMG_VALUE − ALARM_VALUE $> 2^{53}$	Triggered
6	$0 <$ TIMG_VALUE − ALARM_VALUE $\leq 2^{53}$	Triggered when the up-down counter counts TIMG_VALUE down to ALARM_VALUE
7	$0 \leq$ ALARM_VALUE − TIMG_VALUE $< 2^{53}$	Triggered
8	ALARM_VALUE − TIMG_VALUE $\geq 2^{53}$	Triggered when the up-down counter restarts counting down from the timer's maximum value after reaching the minimum value and counts TIMG_VALUE down to ALARM_VALUE

When an alarm occurs, the TIMG*n*_TO_ALARM_EN field is automatically cleared and no alarm will occur again until the TIMG*n*_TO_ALARM_EN is set next time.

13.3.4 Timer Reload

A timer is reloaded when a timer's current value is overwritten with a reload value stored in the TIMG*n*_TO_LOAD_LO and TIMG*n*_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*n*_TO_LOAD_LO and TIMG*n*_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*n*_TOLOAD_REG, which causes the timer's current value to be instantly reloaded. If TIMG*n*_TO_EN is set, the timer will continue incrementing or decrementing from the new value. In this case, if TIMG*n*_TO_ALARM_EN is set, the timer will still trigger alarms in scenarios listed in Table 13.3-1 and 13.3-2. If TIMG*n*_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 to 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*n*_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.

13.3.5 Event Task Matrix Feature

The timer groups on ESP32-H2 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 10 Event Task Matrix (SOC_ETM).

The timer groups can receive the following ETM tasks:

- TIMER*n*_TASK_CNT_START_TIMER0 (*n*:0-1): When triggered, it will enable the time-base counter.
- TIMER*n*_TASK_CNT_STOP_TIMER0 (*n*:0-1): When triggered, it will disable the time-base counter.

Note:
The above two ETM tasks have the same function as the APB configuration TIMG*n*_TO_EN. When these operations occur at the same time, the priority of each operation from high to low is as follows:

1. TIMER*n*_TASK_CNT_START_TIMER0: When triggered, it will enable the time-base counter;
2. TIMER*n*_TASK_CNT_STOP_TIMER0: When triggered, it will disable the time-base counter;
3. APB configuration TIMG*n*_TO_EN: When triggered, it will enable or disable the time-base counter.

- TIMER*n*_TASK_ALARM_START_TIMER0 (*n*:0-1): When triggered, it will enable the alarm generation.

Note:
Alarm generation can also be enabled through APB method configuring TIMG*n*_ALARM_EN and hardware events. When these operations occur at the same time, the priority of each operation from high to low is as follows:

1. TIMER*n*_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*n*_TO_ALARM_EN: When triggered, it will enable or disable the alarm generation.

- TIMER*n*_TASK_CNT_CAP_TIMER0 (*n*:0-1): When triggered, it will update the current counter value to the TIMG*n*_TOLO_REG and TIMG*n*_TOHI_REG registers.
- TIMER*n*_TASK_CNT_RELOAD_TIMER0 (*n*:0-1): When triggered, it will overwrite the current counter value with the reload value stored in TIMG*n*_TO_LOAD_LO and TIMG*n*_TO_LOAD_HI.

The timer groups can generate the following ETM events:

- TIMER*n*_EVT_CNT_CMP_TIMER0 (*n*:0-1): Indicates the interrupt event of T0 in TIMG*n*.

All the ETM tasks and events will not take effect until the TIMG*n*_ETM_EN is set to 1.

In practical applications, timer groups' ETM events can trigger their own ETM tasks. For example, TIMER*n*_TASK_ALARM_START_TIMER0 (*n*:0-1) can be triggered by TIMER*n*_EVT_CNT_CMP_TIMER0 (*n*:0-1) to realize periodic alarm. For configuration steps, please refer to 13.4.4 Timer as Periodic Alarm by ETM.

13.3.6 RTC_SLOW_CLK Frequency Calculation

Using XTAL_CLK as a reference, it is possible to calculate the frequency of clock sources for RTC_SLOW_CLK (i.e., RTC_SLOW_CLK, RC_FAST_DIV_CLK, and XTAL32K_CLK) as follows. However, please note only TIMG*0* supports this function.

1. Start periodic or one-shot frequency calculation (see Section 13.4.5 for details);
2. Once receiving the signal to start the calculation, the counter of XTAL_CLK and the counter of RTC_SLOW_CLK begin to work at the same time. When the counter of RTC_SLOW_CLK counts to C0, the two counters stop counting simultaneously;
3. Assume the value of XTAL_CLK's counter is C1, and the frequency of RTC_SLOW_CLK would be calculated as: $f\_rtc = \frac{C0 \times f\_XTAL\_CLK}{C1}$

13.3.7 Interrupts

Each timer has its own interrupt line that is routed to the CPU, and thus each timer group has a total of two interrupt lines. Level interrupts generated by timers must be explicitly cleared by the CPU on each triggering.

Level interrupts are triggered after an alarm (or stage timeout for watchdog timers) occurs. Level interrupts will be held high after an alarm (or stage timeout) occurs, and will remain so until manually cleared. To enable a timer's interrupt, the TIMG*n*_TO_INT_ENA bit should be set.

The interrupts of each timer group are governed by a set of registers. Each timer within the group has a corresponding bit in each of these registers:

- TIMG*n*_TO_INT_RAW : An alarm event sets it to 1. The bit will remain set until the timer's corresponding bit in TIMG*n*_TO_INT_CLR is written.
- TIMG*n*_WDT_INT_RAW : A stage time out will set the timer's bit to 1. The bit will remain set until the timer's corresponding bit in TIMG*n*_WDT_INT_CLR is written.
- TIMG*n*_TO_INT_ST : Reflects the status of each timer's interrupt and is generated by masking the bits of TIMG*n*_TO_INT_RAW with TIMG*n*_TO_INT_ENA.
- TIMG*n*_WDT_INT_ST : Reflects the status of each watchdog timer's interrupt and is generated by masking the bits of TIMG*n*_WDT_INT_RAW with TIMG*n*_WDT_INT_ENA.
- TIMG*n*_TO_INT_ENA : Used to enable or mask the interrupt status bits of timers within the group.
- TIMG*n*_WDT_INT_ENA : Used to enable or mask the interrupt status bits of watchdog timer within the group.
- TIMG*n*_TO_INT_CLR : Used to clear a timer's interrupt by setting its corresponding bit to 1. The timer's corresponding bit in TIMG*n*_TO_INT_RAW and TIMG*n*_TO_INT_ST will be cleared as a result. Note that a timer's interrupt must be cleared before the next interrupt occurs.
- TIMG*n*_WDT_INT_CLR : Used to clear a timer's interrupt by setting its corresponding bit to 1. The watchdog timer's corresponding bit in TIMG*n*_WDT_INT_RAW and TIMG*n*_WDT_INT_ST will be cleared as a result. Note that a watchdog timer's interrupt must be cleared before the next interrupt occurs.

13.4 Configuration and Usage

13.4.1 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*n*_TO_DIVIDER.
   - Configure the timer direction by setting or clearing TIMG*n*_TO_INCREASE.
   - Set the timer's starting value by writing the starting value to TIMG*n*_TO_LOAD_LO and TIMG*n*_TO_LOAD_HI, then reloading it into the timer by writing any value to TIMG*n*_TOLOAD_REG.
2. Start the timer by setting TIMG*n*_TO_EN.

3. Get the timer's current value.
    - Write any value to TIMG*n*_TOUPDATE_REG to latch the timer's current value.
    - Wait until TIMG*n*_TOUPDATE_REG is cleared by hardware.
    - Read the latched timer value from TIMG*n*_TOLO_REG and TIMG*n*_TOHI_REG.

13.4.2 Timer as One-shot Alarm

1. Configure the time-base counter following step 1 of Section 13.4.1.
2. Configure the alarm.
    - Configure the alarm value by setting TIMG*n*_TOALARMLO_REG and TIMG*n*_TOALARMHI_REG.
    - Enable interrupt by setting TIMG*n*_TO_INT_ENA.
3. Disable auto reload by clearing TIMG*n*_TO_AUTORELOAD.
4. Start the alarm by setting TIMG*n*_TO_ALARM_EN.
5. Handle the alarm interrupt.
    - Clear the interrupt by setting the timer's corresponding bit in TIMG*n*_TO_INT_CLR.
    - Disable the timer by clearing TIMG*n*_TO_EN.

13.4.3 Timer as Periodic Alarm by APB

1. Configure the time-base counter following step 1 in Section 13.4.1.
2. Configure the alarm following step 2 in Section 13.4.2.
3. Enable auto reload by setting TIMG*n*_TO_AUTORELOAD and configure the reload value via TIMG*n*_TO_LOAD_LO and TIMG*n*_TO_LOAD_HI.
4. Start the alarm by setting TIMG*n*_TO_ALARM_EN.
5. Handle the alarm interrupt (repeat on each alarm iteration).
    - Clear the interrupt by setting the timer's corresponding bit in TIMG*n*_TO_INT_CLR.
    - If the next alarm requires a new alarm value and reload value (i.e., different alarm interval per iteration), then TIMG*n*_TOALARMLO_REG, TIMG*n*_TOALARMHI_REG, TIMG*n*_TO_LOAD_LO, and TIMG*n*_TO_LOAD_HI should be reconfigured as needed. Otherwise, the aforementioned registers should remain unchanged.
    - Re-enable the alarm by setting TIMG*n*_TO_ALARM_EN.
6. Stop the timer (on final alarm iteration).
    - Clear the interrupt by setting the timer's corresponding bit in TIMG*n*_TO_INT_CLR.
    - Disable the timer by clearing TIMG*n*_TO_EN.

13.4.4 Timer as Periodic Alarm by ETM

1. Enable the ETM module's clock
2. Map ETM event to ETM task (which means using the event to trigger the task)
   - If TIMG*n*_TO_AUTORELOAD is set to 1, map TIMER*n*_EVT_CNT_CMP_TIMER0 (*n*:0-1) to the TIMER*n*_TASK_ALARM_START_TIMER0 (*n*:0-1) by one ETM channel.
   - If TIMG*n*_TO_AUTORELOAD is set to 0, in addition to mapping TIMER*n*_EVT_CNT_CMP_TIMER0 (*n*:0-1) to the TIMER*n*_TASK_ALARM_START_TIMER0 (*n*:0-1), the TIMER*n*_EVT_CNT_CMP_TIMER0 (*n*:0-1) should also be mapped to TIMER*n*_TASK_CNT_RELOAD_TIMER0 (*n*:0-1) by another ETM channel.
3. Choose to enable one or two ETM channels.
4. Set TIMG*n*_ETM_EN to 1 to enable timer group's ETM events and tasks.
5. Configure the time-base counter following step 1 in Section 13.4.1.
6. Configure the alarm following step 2 in Section 13.4.2.
7. Configure the reload value via TIMG*n*_TO_LOAD_LO and TIMG*n*_TO_LOAD_HI.
8. Handle the TIMER*n*_EVT_CNT_CMP_TIMER0 (*n*:0-1).
   - When alarm generates, the TIMER*n*_EVT_CNT_CMP_TIMER0 (*n*:0-1) also generates, and the alarm generation will be disabled by the alarm.
   - If TIMG*n*_TO_AUTORELOAD is 1, the current counter value is overwritten by the reloaded value. The alarm generation will be reopened by TIMER*n*_TASK_ALARM_START_TIMER0 (*n*:0-1).
   - If TIMG*n*_TO_AUTORELOAD is 0, the current counter value is overwritten by the reloaded value because of the TIMER*n*_TASK_CNT_RELOAD_TIMER0 (*n*:0-1). The alarm generation will be reopened by TIMER*n*_TASK_ALARM_START_TIMER0 (*n*:0-1).
9. Stop the timer (on final alarm iteration).
   - Disable the ETM channels used to map the timer group's event and task
   - Set TIMG*n*_ETM_EN to 0.
   - Clear the interrupt by setting the timer's corresponding bit in TIMG*n*_TO_INT_CLR.
   - Disable the timer by clearing TIMG*n*_TO_EN.

13.4.5 RTC_SLOW_CLK Frequency Calculation

1. One-shot frequency calculation
   - Select the clock whose frequency is to be calculated (clock source of RTC_SLOW_CLK) via TIMG*0*_RTC_CALI_CLK_SEL, and configure the time of calculation via TIMG*0*_RTC_CALI_MAX.
   - Select one-shot frequency calculation by clearing TIMG*0*_RTC_CALI_START_CYCLING, and enable the two counters via TIMG*0*_RTC_CALI_START.
   - Once TIMG*0*_RTC_CALI_RDY becomes 1, read TIMG*0*_RTC_CALI_VALUE to get the value of XTAL_CLK's counter, and calculate the frequency of RTC_SLOW_CLK according to the formula in Section 13.3.6.

2. Periodic frequency calculation
   - Select the clock whose frequency is to be calculated (clock source of RTC_SLOW_CLK) via TIMG*O*_RTC_CALI_CLK_SEL, and configure the time of calculation via TIMG*O*_RTC_CALI_MAX.
   - Select periodic frequency calculation by enabling TIMG*O*_RTC_CALI_START_CYCLING.
   - When TIMG*O*_RTC_CALI_CYCLING_DATA_VLD is 1, TIMG*O*_RTC_CALI_VALUE is valid.
3. Timeout
   If the counter of RTC_SLOW_CLK cannot finish counting in TIMG*O*_RTC_CALI_TIMEOUT_RST_CNT cycles, TIMG*O*_RTC_CALI_TIMEOUT will be set to indicate a timeout.

13.5 Register Summary

The addresses in this section are relative to Timer Group base address provided in Table 4.3-2 in Chapter 4 *System and Memory*.

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

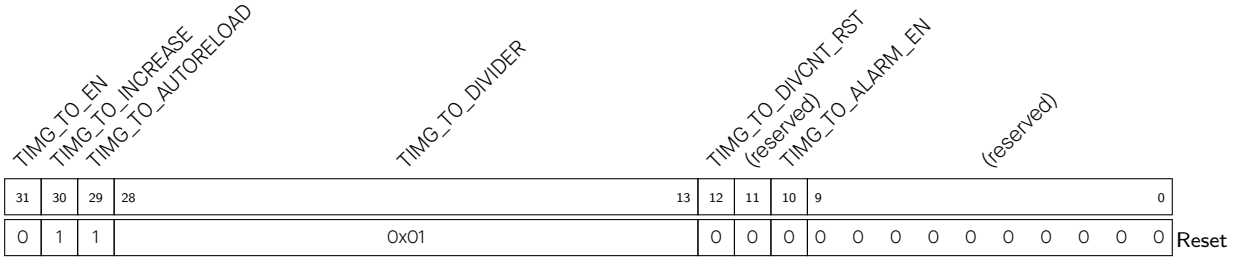
Name	Description	Address	Access
T0 control and configuration registers			
TIMG_T0CONFIG_REG	Timer 0 configuration register	0x0000	varies
TIMG_T0LO_REG	Timer 0 current value, low 32 bits	0x0004	RO
TIMG_T0HI_REG	Timer 0 current value, high 22 bits	0x0008	RO
TIMG_T0UPDATE_REG	Write to copy current timer value to TIMGn_T0LO_REG or TIMGn_T0HI_REG	0x000C	R/W/SC
TIMG_T0ALARMLO_REG	Timer 0 alarm value, low 32 bits	0x0010	R/W
TIMG_T0ALARMHI_REG	Timer 0 alarm value, high bits	0x0014	R/W
TIMG_T0LOADLO_REG	Timer 0 reload value, low 32 bits	0x0018	R/W
TIMG_T0LOADHI_REG	Timer 0 reload value, high 22 bits	0x001C	R/W
TIMG_T0LOAD_REG	Write to reload timer from TIMG_T0LOADLO_REG or TIMG_T0LOADHI_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 frequency calculation control and configuration registers			
TIMG_RTCCALICFG_REG	RTC frequency calculation configuration register 0	0x0068	varies
TIMG_RTCCALICFG1_REG	RTC frequency calculation configuration register 1	0x006C	RO
TIMG_RTCCALICFG2_REG	RTC frequency calculation configuration 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			

Name	Description	Address	Access
TIMG_REGCLK_REG	Timer group clock gate register	0x00FC	R/W

13.6 Registers

The addresses in this section are relative to Timer Group base address provided in Table 4.3-2 in Chapter 4 *System and Memory*.

Register 13.1. TIMG_T0CONFIG_REG (0x0000)



TIMG_T0_ALARM_EN Configures whether 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 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 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: Decrease

1: Increase

(R/W)

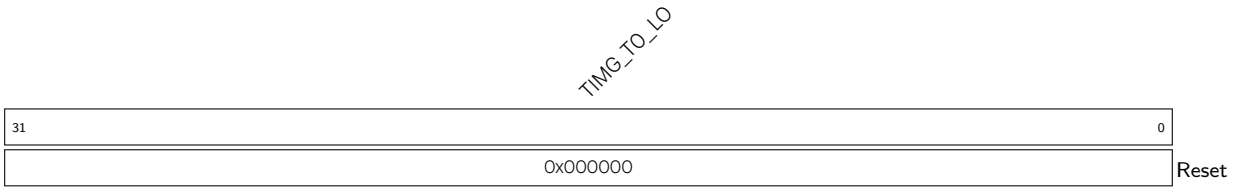
TIMG_T0_EN Configures whether to enable the Timer 0 time-base counter.

0: Disable

1: Enable

(R/W/SS/SC)

Register 13.2. TIMG_T0LO_REG (0x0004)

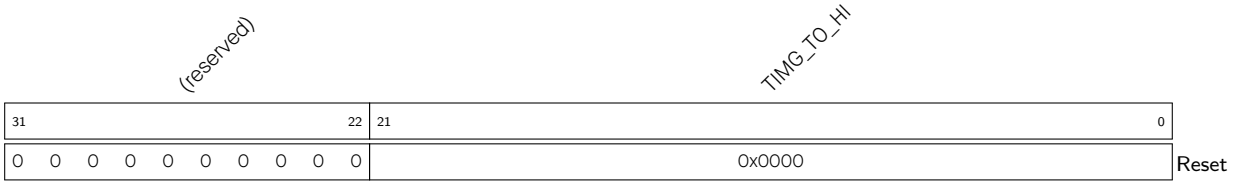


TIMG_T0_LO Represents the low 32 bits of the time-base counter of Timer 0. Valid only after writing to TIMG_T0UPDATE_REG.

Measurement unit: T0_clk

(RO)

Register 13.3. TIMG_T0HI_REG (0x0008)

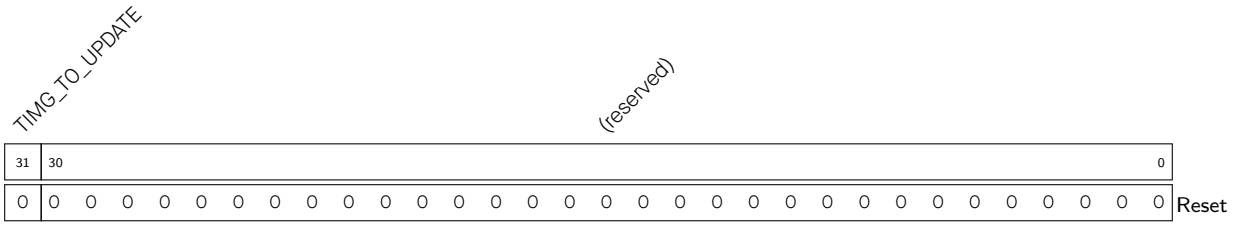


TIMG_T0_HI Represents the high 22 bits of the time-base counter of Timer 0. Valid only after writing to TIMG_T0UPDATE_REG.

Measurement unit: T0_clk

(RO)

Register 13.4. TIMG_T0UPDATE_REG (0x000C)



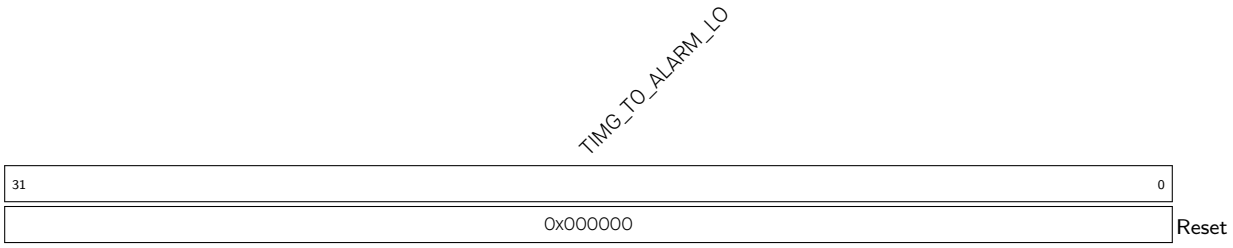
TIMG_T0_UPDATE Configures to latch the counter value.

0: Latch

1: Latch

(R/W/SC)

Register 13.5. TIMG_T0ALARMLO_REG (0x0010)

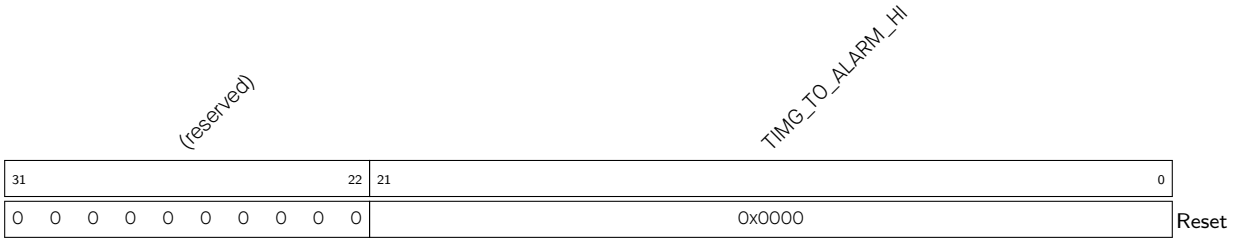


TIMG_T0_ALARM_LO Configures the low 32 bits of Timer 0 alarm trigger time-base counter value. Valid only when TIMG_T0_ALARM_EN is 1.

Measurement unit: T0_clk

(R/W)

Register 13.6. TIMG_T0ALARMHI_REG (0x0014)

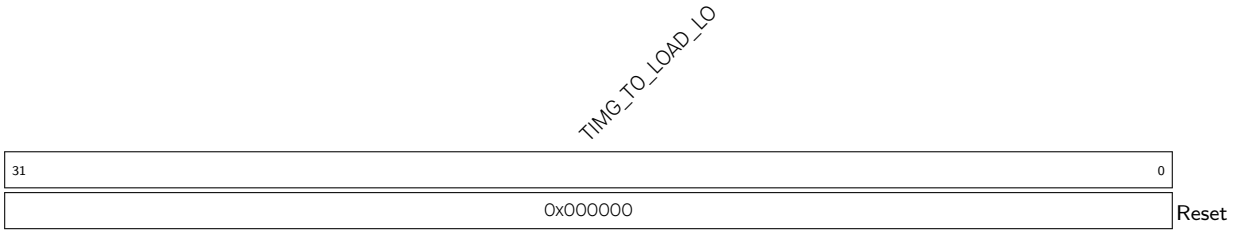


TIMG_T0_ALARM_HI Configures the high 22 bits of Timer 0 alarm trigger time-base counter value. Valid only when TIMG_T0_ALARM_EN is 1.

Measurement unit: T0_clk

(R/W)

Register 13.7. TIMG_T0LOADLO_REG (0x0018)

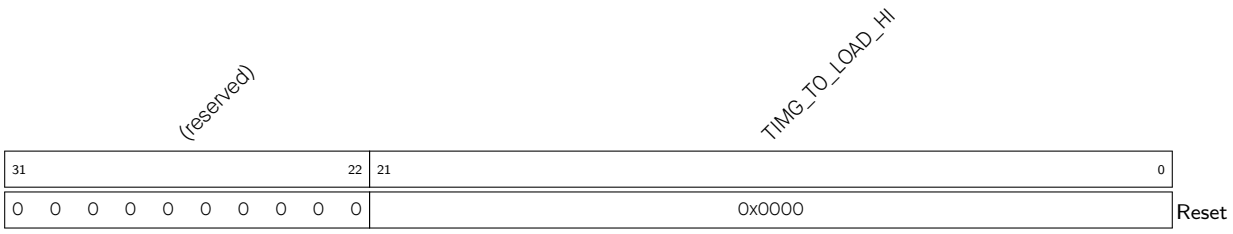


TIMG_T0_LOAD_LO Configures low 32 bits of the value that a reload will load onto Timer 0 time-base counter.

Measurement unit: T0_clk

(R/W)

Register 13.8. TIMG_T0LOADHI_REG (0x001C)

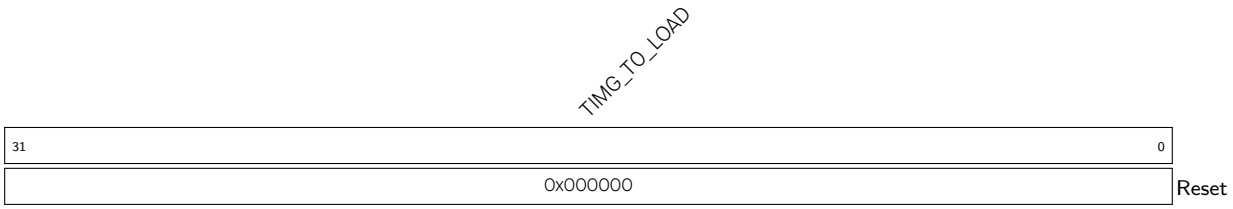


TIMG_T0_LOAD_HI Configures high 22 bits of the value that a reload will load onto Timer 0 time-base counter.

Measurement unit: T0_clk

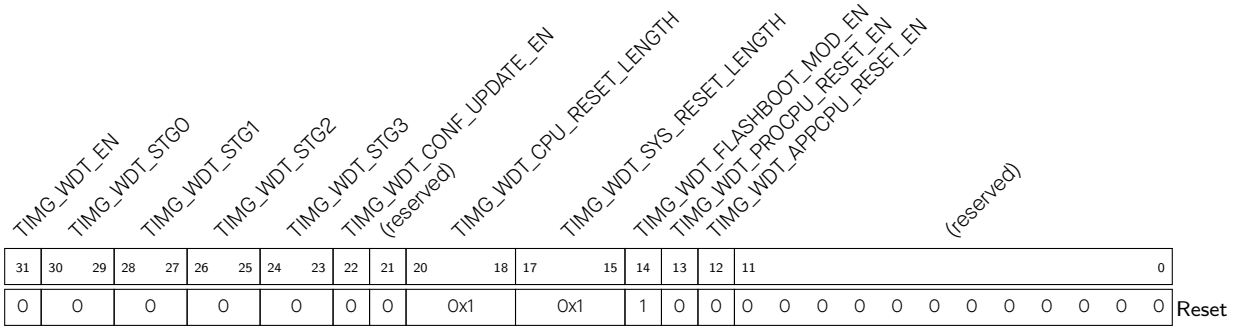
(R/W)

Register 13.9. TIMG_T0LOAD_REG (0x0020)



TIMG_T0_LOAD Write any value to trigger a Timer 0 time-base counter reload. (WT)

Register 13.10. TIMG_WDTCONFIG0_REG (0x0048)



TIMG_WDT_APPCPU_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_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)

Continued on the next page...

Register 13.10. TIMG_WDTCONFIG0_REG (0x0048)

Continued from the previous page...

TIMG_WDT_CPU_RESET_LENGTH Configures the CPU reset signal length. Valid only when write protection is disabled.
Measurement unit: mwdt_clk

0: 8
1: 16
2: 24
3: 32
4: 40
5: 64
6: 128
7: 256

(R/W)

TIMG_WDT_CONF_UPDATE_EN Configures to update the WDT configuration registers.
0: No effect
1: Update
(WT)

TIMG_WDT_STG3 Configures the timeout action of stage 3. See details in TIMG_WDT_STG0. Valid only when write protection is disabled. (R/W)

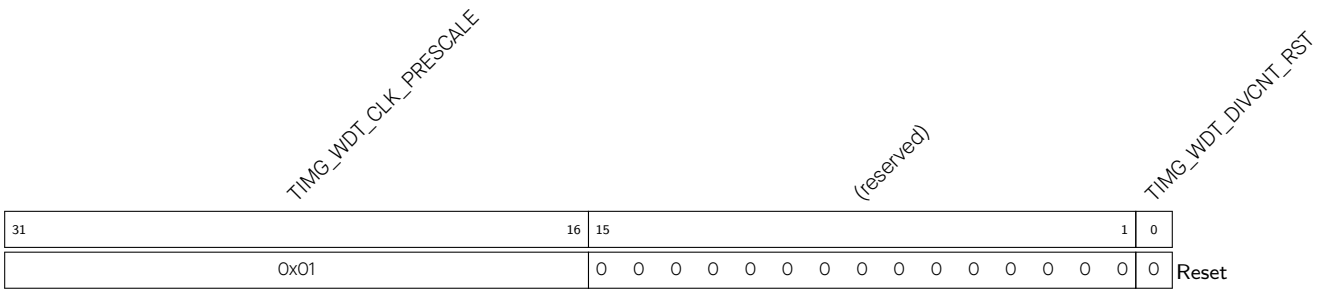
TIMG_WDT_STG2 Configures the timeout action of stage 2. See details in TIMG_WDT_STG0. Valid only when write protection is disabled. (R/W)

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 to enable the MWDT. Valid only when write protection is disabled.
0: Disable
1: Enable
(R/W)

Register 13.11. TIMG_WDTCONFIG1_REG (0x004C)



TIMG_WDT_DIVCNT_RST Configures whether to reset WDT's clock divider counter.

0: No effect

1: Reset

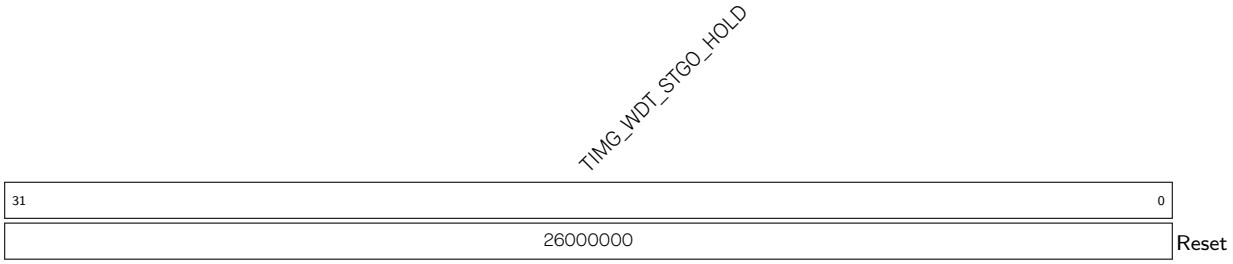
(WT)

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 13.12. TIMG_WDTCONFIG2_REG (0x0050)

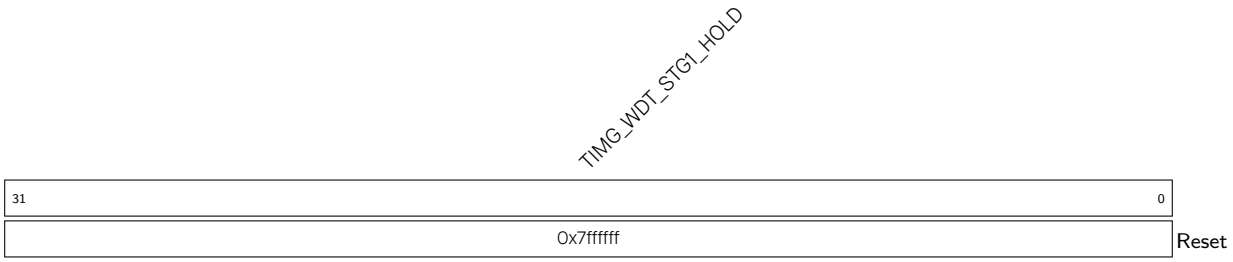


TIMG_WDT_STG0_HOLD Configures the stage 0 timeout value. Valid only when write protection is disabled.

Measurement unit: mwdt_clk

(R/W)

Register 13.13. TIMG_WDTCONFIG3_REG (0x0054)

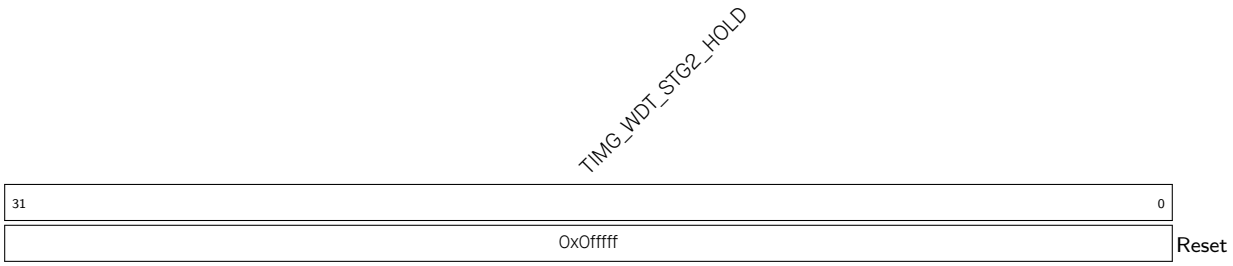


TIMG_WDT_STG1_HOLD Configures the stage 1 timeout value. Valid only when write protection is disabled.

Measurement unit: mwdt_clk

(R/W)

Register 13.14. TIMG_WDTCONFIG4_REG (0x0058)

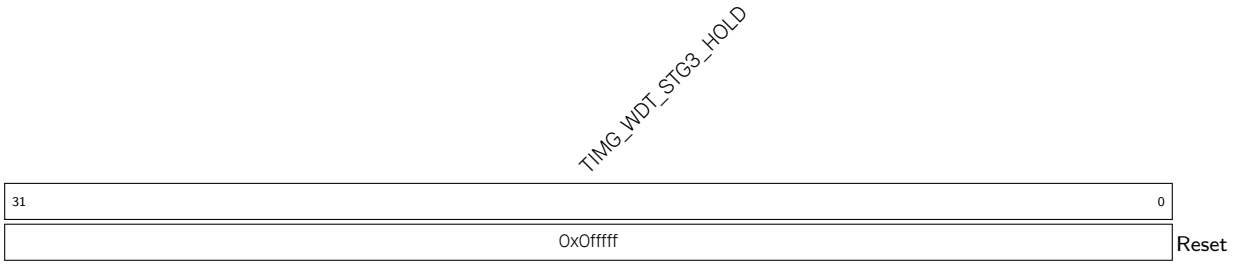


TIMG_WDT_STG2_HOLD Configures the stage 2 timeout value. Valid only when write protection is disabled.

Measurement unit: mwdt_clk

(R/W)

Register 13.15. TIMG_WDTCONFIG5_REG (0x005C)

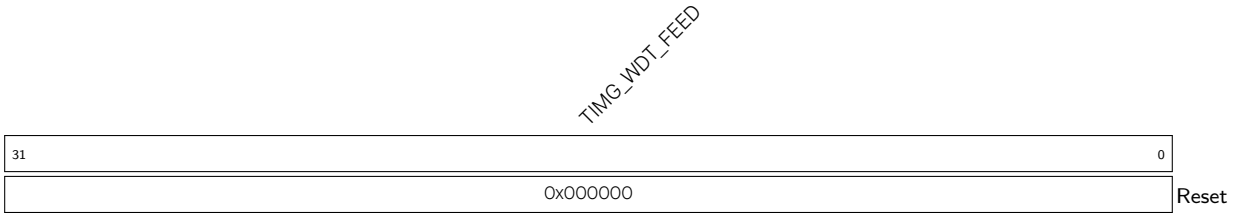


TIMG_WDT_STG3_HOLD Configures the stage 3 timeout value. Valid only when write protection is disabled.

Measurement unit: mwdt_clk

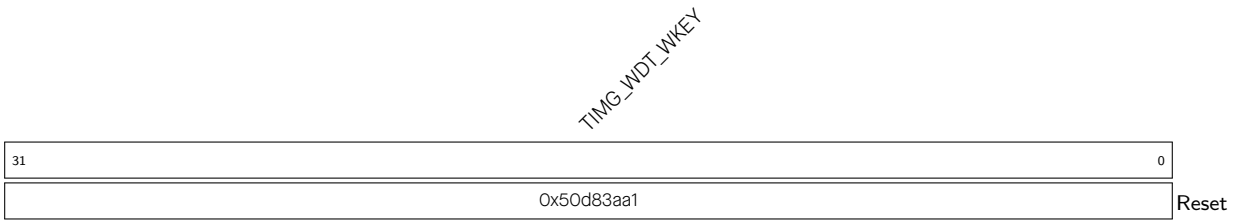
(R/W)

Register 13.16. TIMG_WDTFEED_REG (0x0060)



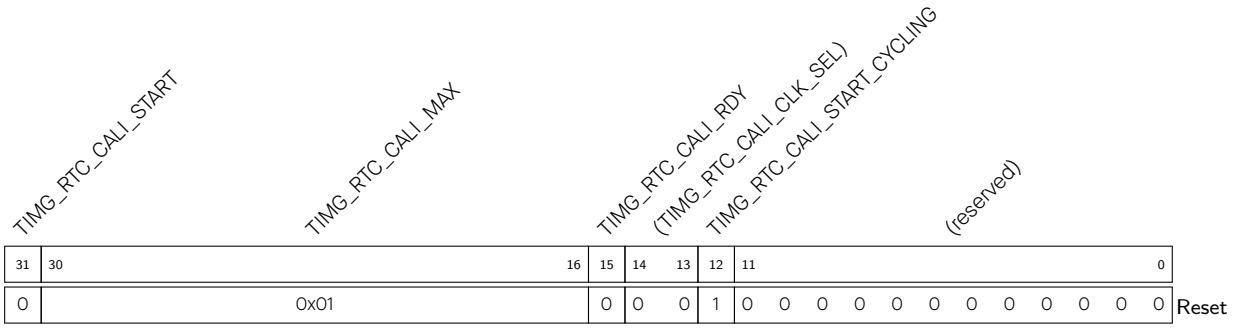
TIMG_WDT_FEED Write any value to feed the MWDT. Valid only when write protection is disabled. (WT)

Register 13.17. TIMG_WDTWPROTECT_REG (0x0064)



TIMG_WDT_WKEY Configures a different value than its reset value to enable write protection. (R/W)

Register 13.18. 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_CLK_SEL Configures to select the clock to be calibrated

0: RTC_SLOW_CLK

1: RC_FAST_DIV_CLK

2: XTAL32K_CLK

(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)

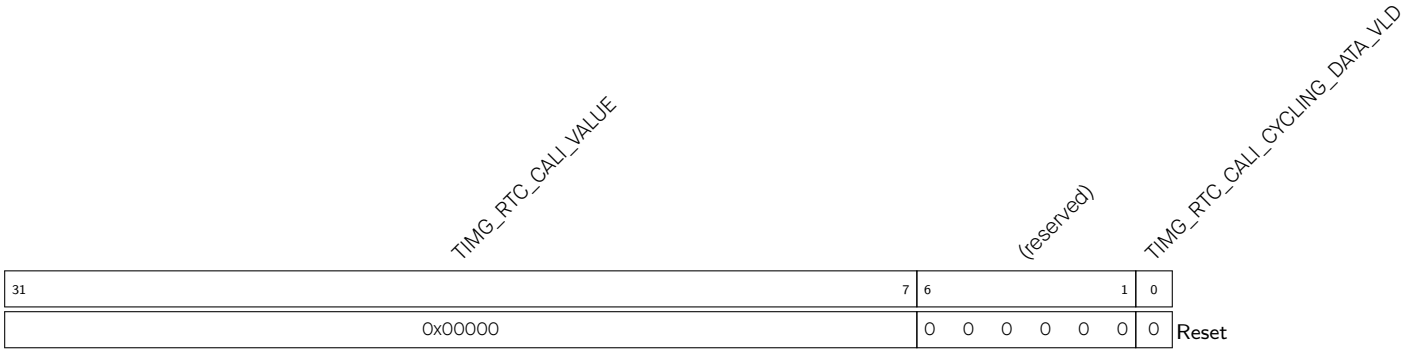
TIMG_RTC_CALI_START Configures whether to enable one-shot frequency calculation.

0: Disable

1: Enable

(R/W)

Register 13.19. TIMG_RTCCALICFG1_REG (0x006C)



TIMG_RTC_CALI_CYCLING_DATA_VLD Represents whether periodic frequency calculation is done.

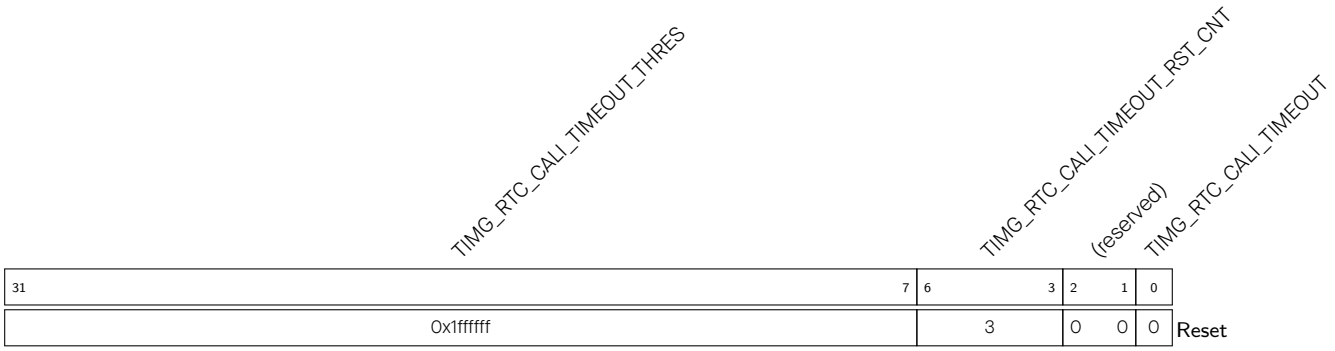
0: Not done

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 13.20. 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.

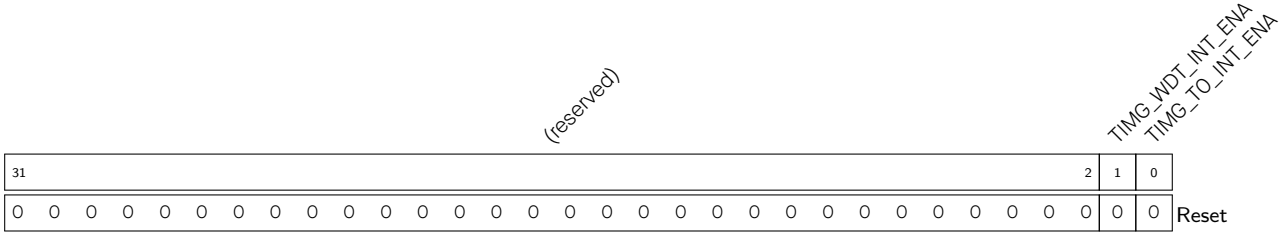
(R/W)

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)

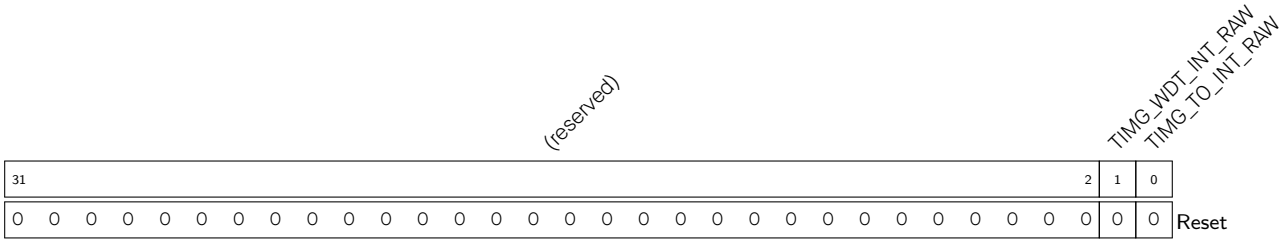
Register 13.21. TIMG_INT_ENA_TIMERS_REG (0x0070)



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)

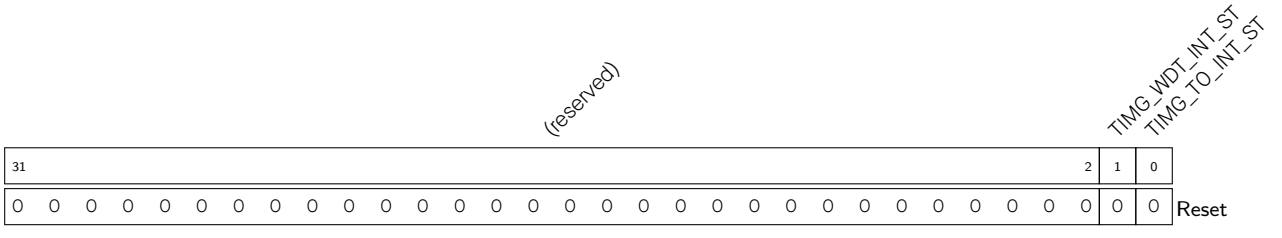
Register 13.22. TIMG_INT_RAW_TIMERS_REG (0x0074)



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)

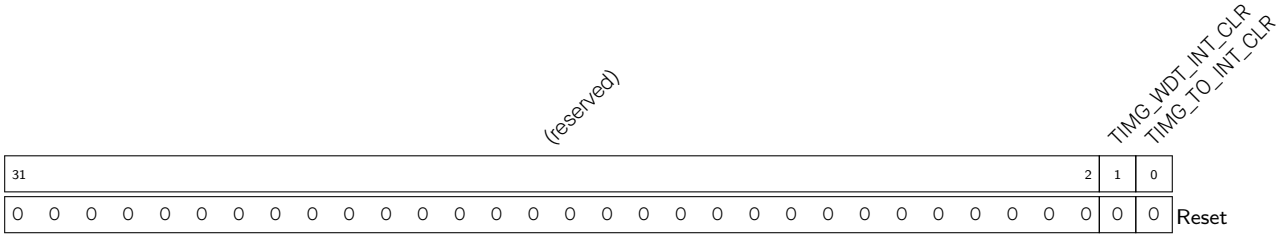
Register 13.23. TIMG_INT_ST_TIMERS_REG (0x0078)



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)

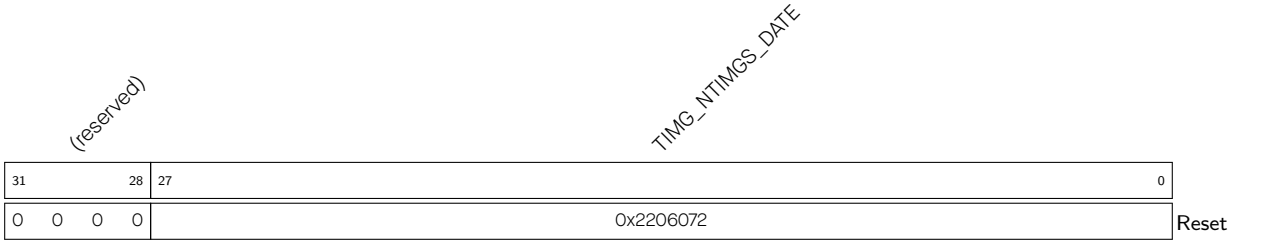
Register 13.24. TIMG_INT_CLR_TIMERS_REG (0x007C)



TIMG_TO_INT_CLR Write 1 to clear the TIMG_TO_INT interrupt. (WT)

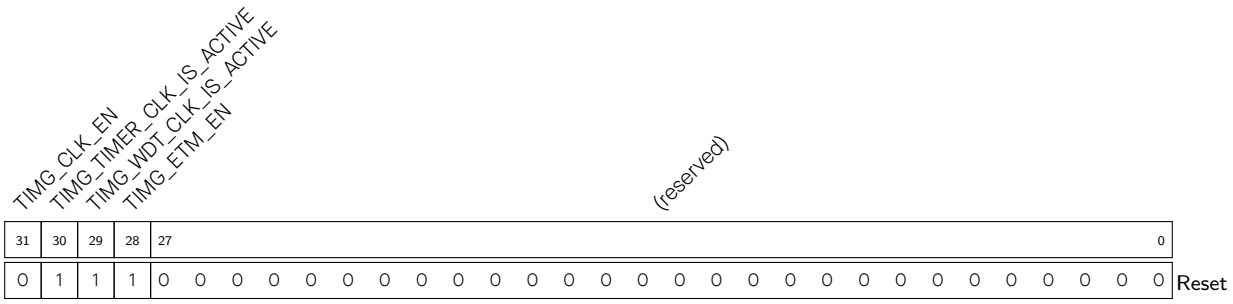
TIMG_WDT_INT_CLR Write 1 to clear the TIMG_WDT_INT interrupt. (WT)

Register 13.25. TIMG_NTIMERS_DATE_REG (0x00F8)



TIMG_NTIMGS_DATE Version control register (R/W)

Register 13.26. TIMG_REGCLK_REG (0x00FC)



TIMG_ETM_EN Configures whether to enable timer's ETM task and event.

0: Disable

1: Enable

(R/W)

TIMG_WDT_CLK_IS_ACTIVE Configures whether to enable WDT's clock.

0: Disable

1: Enable

(R/W)

TIMG_TIMER_CLK_IS_ACTIVE Configures whether to enable Timer 0's clock.

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 14

Watchdog Timers (WDT)

14.1 Overview

Watchdog timers are hardware timers used to detect and recover from malfunctions. They must be periodically fed (reset) to prevent a timeout. A system/software that is behaving unexpectedly (e.g., is stuck in a software loop or in overdue events) will fail to feed the watchdog thus triggering a watchdog timeout. Therefore, watchdog timers are useful for detecting and handling erroneous system/software behavior.

As shown in Figure 14.1-1, ESP32-H2 contains three digital watchdog timers: one in each of the two timer groups described in Chapter 13 *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 14.2.2.2 *Stages and Timeout Actions*). A timeout value can be set for each stage individually.

During the flash boot process, RWDT and the MWDT0 are enabled automatically in order to detect and recover from booting errors.

ESP32-H2 also has one analog watchdog timer: Super watchdog (SWD). It is an ultra-low-power circuit in the analog domain that helps to prevent the system from operating in a sub-optimal state and resets the system if required.

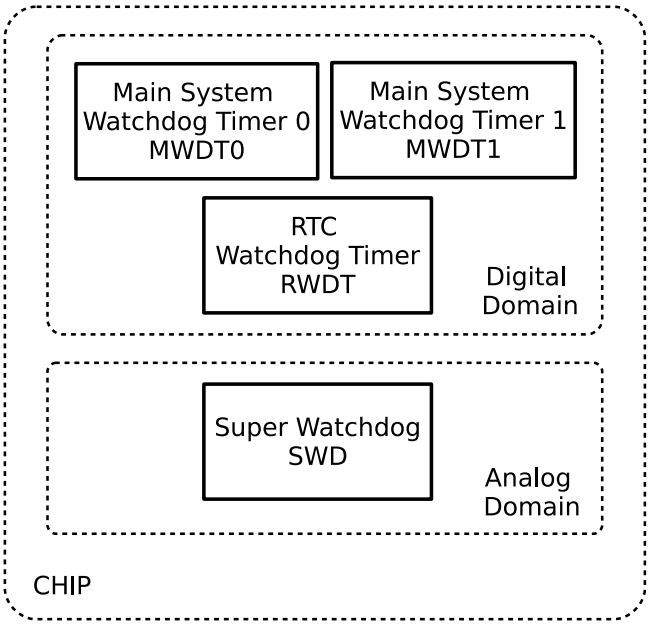


Figure 14.1-1. Watchdog Timers Overview

Note that while this chapter provides the functional descriptions of the watchdog timer, MWDT register descriptions are detailed in Chapter 13 *Timer Group (TIMG)*, and the RWDT and SWD register descriptions are detailed in Section 14.5 *Register Summary*.

Note:
Unless otherwise specified, MWDT in this chapter refers to both MWDT0 and MWDT1.

14.2 Digital Watchdog Timers

14.2.1 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_F48M_CLK, RC_FAST_CLK or XTAL_CLK
  - RWDT: RTC_SLOW_CLK

14.2.2 Functional Description

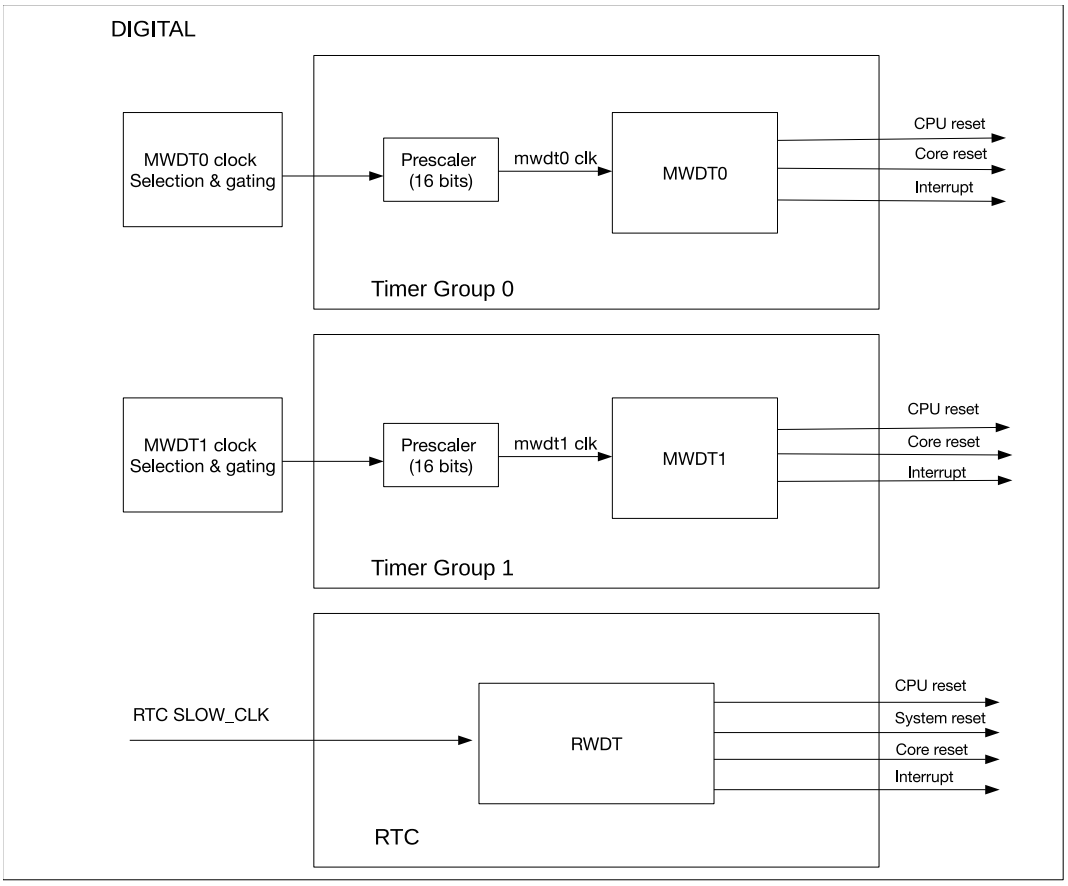


Figure 14.2-1. Digital Watchdog Timers in ESP32-H2

Figure 14.2-1 shows the three watchdog timers in ESP32-H2 digital systems.

14.2.2.1 Clock Source and 32-Bit Counter

At the core of each watchdog timer is a 32-bit counter.

Take MWDT0 in Timer Group 0 as an example:

- MWDT0 can select between the PLL_F48M_CLK, RC_FAST_CLK or XTAL_CLK (external) clock as its clock source by setting the PCR_TGO_WDT_CLK_SEL field of the PCR_TIMERGROUP0_WDT_CLK_CONF_REG register.
- The selected clock is switched on by setting PCR_TGO_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 7.2-1 *CPU_CLK Clock Source* of Chapter 7 *Reset and Clock*.
- The 16-bit prescaler for MWDT0 is configured via the TIMG*n*_WDT_CLK_PRESCALE (*n*: 0 ~ 1, where for Timer Group 0, the value of *n* should be 0 here) field of TIMG*n*_WDTCONFIG1_REG. When TIMG*n*_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 7 *Reset and Clock*).

MWDT and RWDT are enabled by setting the TIMG*n*_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*n*_WDTFEED_REG for MDWT and by writing 1 to RTC_WDT_FEED for RWDT.

14.2.2.2 Stages and Timeout Actions

Timer stages allow for a timer to have a series of different timeout values and corresponding timeout actions. When one stage times out, the timeout action is triggered, the counter value is reset to zero, and the next stage becomes active.

The MWDT0, MWDT1 and RWDT each provide four stages, referred to as stages 0 to stage 3. The watchdog timers advance sequentially through these stages in a loop, starting from stage 0, then progressing through to stage 3, and then returning back to stage 0.

Timeout values of each stage for MWDT are configured in TIMG*n*_WDTCONFIG*i*_REG (where *i* ranges from 2 to 5), whilst timeout values for RWDT are configured using RTC_WDT_STG*j*_HOLD field (where *j* ranges from 0 to 3).

Please note that the timeout value of stage 0 for RWDT ($Thold_0$) is determined by the combination of the
EFUSE_WDT_DELAY_SEL field of eFuse register EFUSE_RD_REPEAT_DATA0_REG and RTC_WDT_STG0_HOLD field. The relationship is as follows:

$$T_{hold0} = RTC\_WDT\_STG0\_HOLD << (EFUSE\_WDT\_DELAY\_SEL + 1)$$

where $<<$ is a left-shift operator. For example, if RTC_WDT_STG0_HOLD is configured as 100 and EFUSE_WDT_DELAY_SEL is 1, the $Thold_0$ will be 400 cycles.

Upon the timeout of each stage, one of the following timeout actions will be executed:

Table 14.2-1. Timeout Actions

Timeout Action	Description
Interrupt	Trigger an interrupt
CPU reset	Reset the CPU core
Core reset	Reset the whole digital system except LP system, including CPU, peripherals, digital GPIOs, Bluetooth® LE, and 802.15.4
System reset	Reset the whole digital system, including LP system. This action is only available in RWDT
Disabled	No effect on the system

For MWDT, the timeout action of all stages is configured in TIMG*n*_WDTCONFIG0_REG. Likewise for RWDT, the timeout action is configured in RTC_WDT_CONFIG0_REG.

14.2.2.3 Write Protection

Watchdog timers are critical to detecting and handling erroneous system/software behavior, thus should not be disabled easily (e.g., due to a misplaced register write). Therefore, MWDT and RWDT incorporate a write protection mechanism that 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*n*_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.

14.2.2.4 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*n*_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.

14.3 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.

14.3.1 Features

SWD has the following features:

- Ultra-low power
- Interrupt to indicate that the SWD is about to time out
- Various dedicated methods for software to feed SWD, which enables SWD to monitor the working state of the whole operating system

14.3.2 Super Watchdog Controller

14.3.2.1 Structure

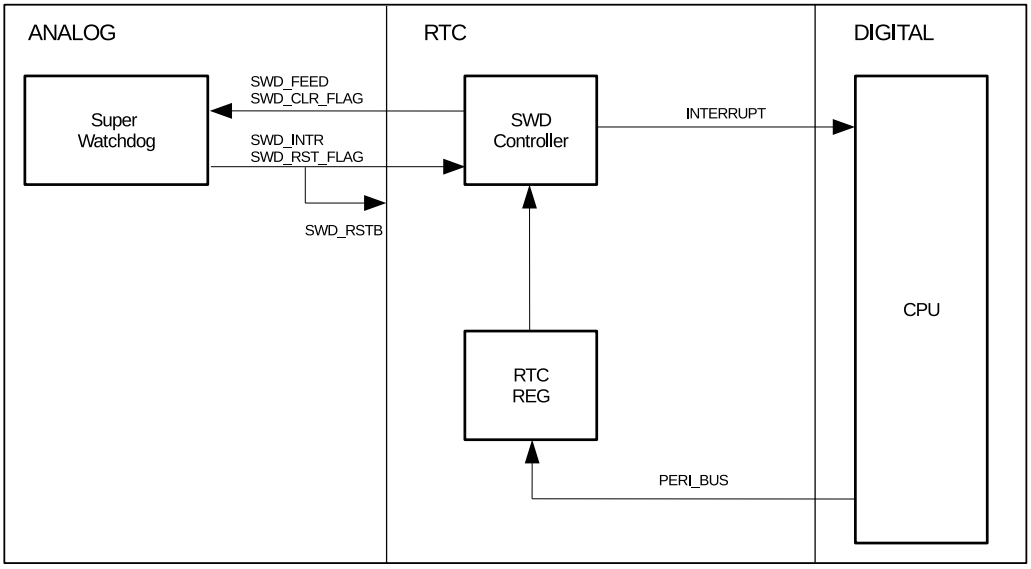


Figure 14.3-1. Super Watchdog Controller Structure

14.3.2.2 Workflow

In normal state:

- SWD controller receives feed request from SWD.
- SWD controller can send an interrupt to main CPU.
- Main CPU can feed SWD directly by setting RTC_WDT_SWD_FEED.
- When trying to feed SWD, CPU needs to disable SWD controller's write protection by writing 0x50D83AA1 to RTC_WDT_SWD_WKEY. This prevents SWD from being fed by mistake when the system is operating in sub-optimal state.
- If setting RTC_WDT_SWD_AUTO_FEED_EN to 1, SWD controller can also feed SWD itself without any interaction with CPU.

After reset:

- Check RTC_CLKRST_RESET_CAUSE[4:0] for the cause of CPU reset.
  If RTC_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.

14.4 Interrupts

For watchdog timer interrupts, please refer to Section 13.3.7 *Interrupts* in Chapter 13 *Timer Group (TIMG)*.

14.5 Register Summary

The addresses in this section are relative to RTC_WDT base address provided in Table 4.3-2 in Chapter 4 *System and Memory*.

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

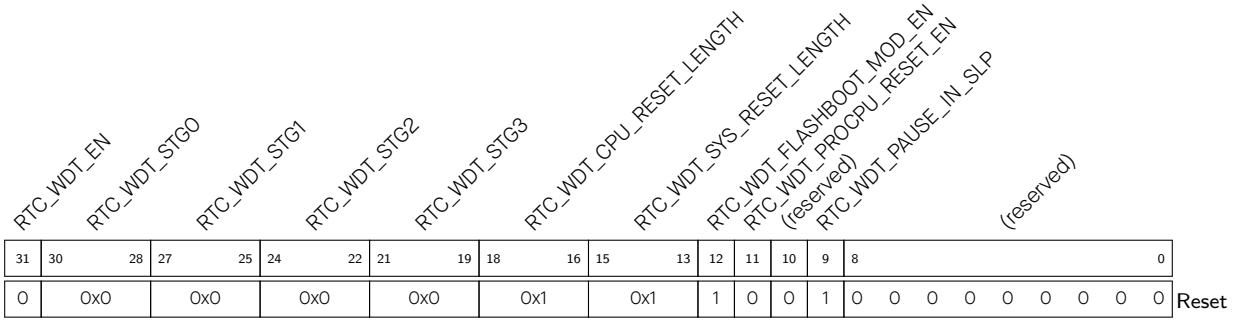
Name	Description	Address	Access
configuration register			
RTC_WDT_CONFIG0_REG	Configure the RWDT operation	0x0000	R/W
RTC_WDT_CONFIG1_REG	Configure the RWDT timeout time of stage0	0x0004	R/W
RTC_WDT_CONFIG2_REG	Configure the RWDT timeout time of stage1	0x0008	R/W
RTC_WDT_CONFIG3_REG	Configure the RWDT 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	0x0018	WT
RTC_WDT_WPROTECT_REG	Configure the lock function of RWDT	0x001C	R/W
RTC_WDT_SWD_CONFIG_REG	Configure the SWD operation	0x0020	varies
RTC_WDT_SWD_WPROTECT_REG	Configure the lock function of SWD	0x0024	R/W
RTC_WDT_INT_RAW_REG	The interrupt raw register of WDT	0x0028	R/WTC/SS
RTC_WDT_INT_ST_REG	The interrupt status register of WDT	0x002C	RO
RTC_WDT_INT_ENA_REG	The interrupt enable register of WDT	0x0030	R/W
RTC_WDT_INT_CLR_REG	The interrupt clear register of WDT	0x0034	WT
RTC_WDT_DATE_REG	Version control register	0x03FC	R/W

14.6 Registers

MWDT registers are part of the timer submodule and are described in Section 13.5 *Register Summary* in Chapter 13 *Timer Group (TIMG)*.

The addresses of RWDT and SWD registers in this section are relative to RTC_WDT base address provided in Table 4.3-2 in Chapter 4 *System and Memory*.

Register 14.1. RTC_WDT_CONFIG0_REG (0x0000)



RTC_WDT_PAUSE_IN_SLP Configure whether or not to disable RWDT when chip is in sleep mode.

0: Enable

1: Disable

(R/W)

RTC_WDT_PROCPU_RESET_EN Configure whether or not to enable RWDT to reset CPU.

0: Disable

1: Enable

(R/W)

RTC_WDT_FLASHBOOT_MOD_EN Configure whether or not to enable RWDT when chip is in SPI boot mode.

0: Disable

1: Enable

(R/W)

RTC_WDT_SYS_RESET_LENGTH Configure the core reset time.

Measurement unit: RTC_DYN_FAST_CLK

(R/W)

RTC_WDT_CPU_RESET_LENGTH Configure the CPU reset time.

Measurement unit: RTC_DYN_FAST_CLK

(R/W)

RTC_WDT_STG3 Configure the timeout action of stage3.

0: No action

1: Generate interrupt

2: Generate CPU reset

3: Generate core reset

4: Generate system reset

(R/W)

Continued on the next page...

Register 14.1. RTC_WDT_CONFIG0_REG (0x0000)

Continued from the previous page...

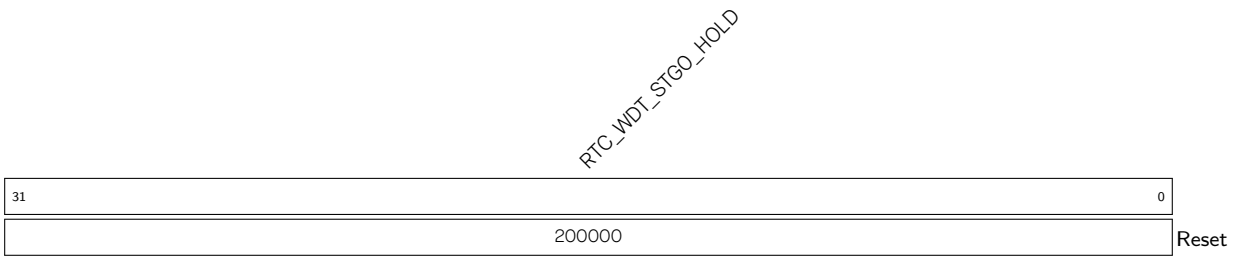
RTC_WDT_STG2 Configure the timeout action of stage2.
0: No action
1: Generate interrupt
2: Generate CPU reset
3: Generate core reset
4: Generate system reset
(R/W)

RTC_WDT_STG1 Configure the timeout action of stage1.
0: No action
1: Generate interrupt
2: Generate CPU reset
3: Generate core reset
4: Generate system reset
(R/W)

RTC_WDT_STG0 Configure the timeout action of stage0.
0: No action
1: Generate interrupt
2: Generate CPU reset
3: Generate core reset
4: Generate system reset
(R/W)

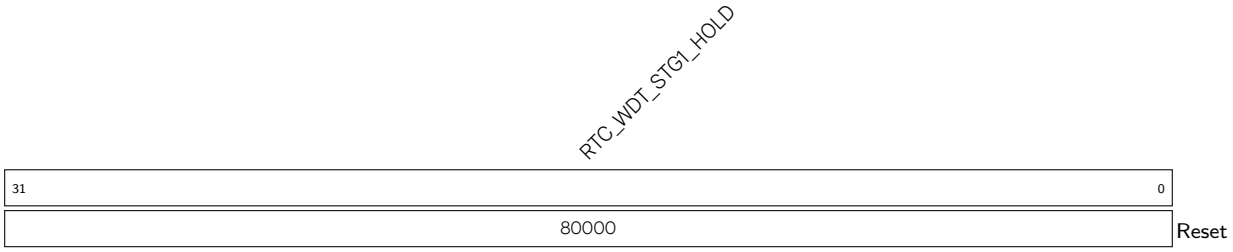
RTC_WDT_EN Configure whether or not enable RWDT.
0: Disable RWDT
1: Enable RWDT
(R/W)

Register 14.2. RTC_WDT_CONFIG1_REG (0x0004)



RTC_WDT_STG0_HOLD Configure the timeout time for stage0.
Measurement unit: RTC_DYN_SLOW_CLK
(R/W)

Register 14.3. RTC_WDT_CONFIG2_REG (0x0008)

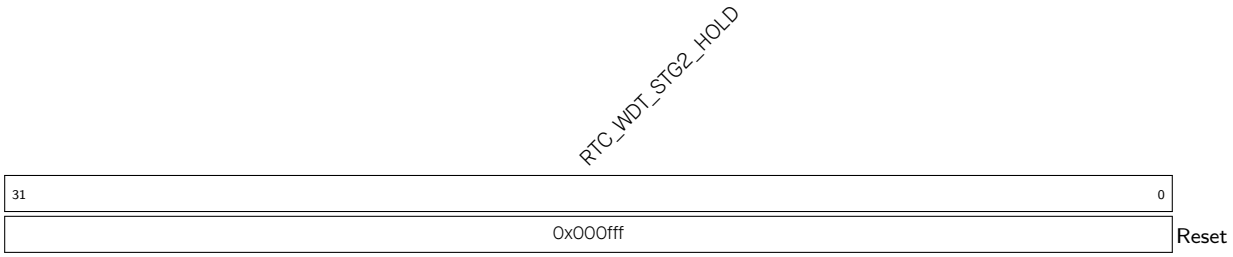


RTC_WDT_STG1_HOLD Configure the timeout time for stage1.

Measurement unit: RTC_DYN_SLOW_CLK

(R/W)

Register 14.4. RTC_WDT_CONFIG3_REG (0x000C)

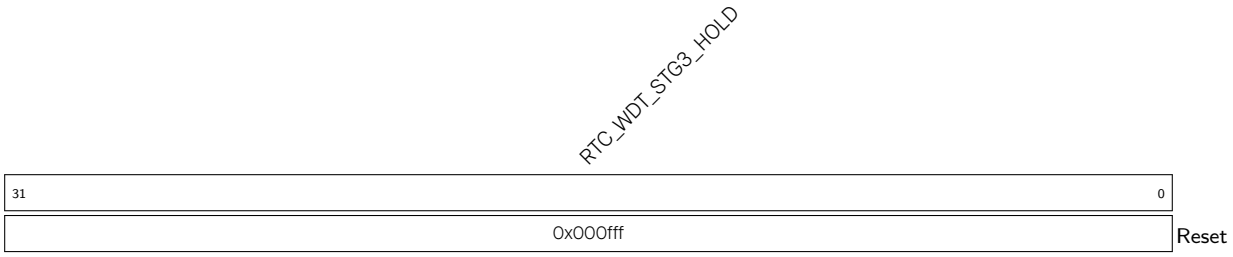


RTC_WDT_STG2_HOLD Configure the timeout time for stage2.

Measurement unit: RTC_DYN_SLOW_CLK

(R/W)

Register 14.5. RTC_WDT_CONFIG4_REG (0x0010)

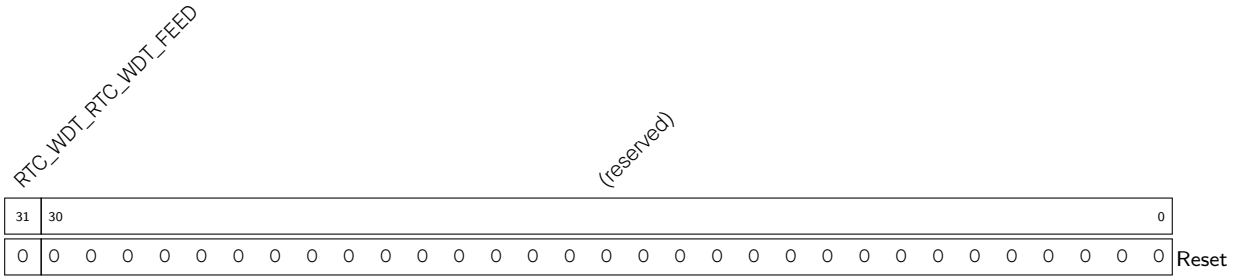


RTC_WDT_STG3_HOLD Configure the timeout time for stage3.

Measurement unit: RTC_DYN_SLOW_CLK

(R/W)

Register 14.6. RTC_WDT_FEED_REG (0x0018)



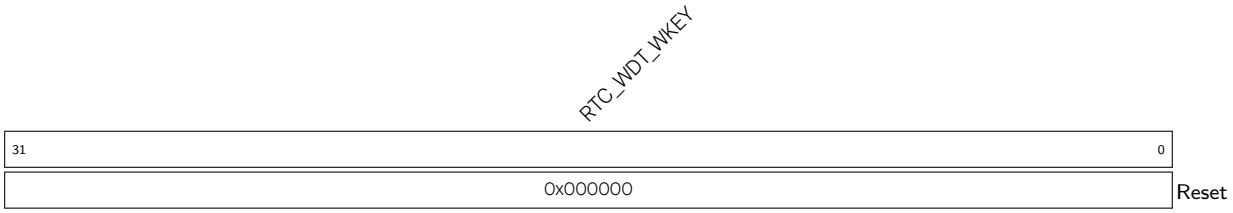
RTC_WDT_RTC_WDT_FEED Configure this bit to feed the RWDT.

0: Invalid

1: Feed RWDT

(WT)

Register 14.7. RTC_WDT_WPROTECT_REG (0x001C)



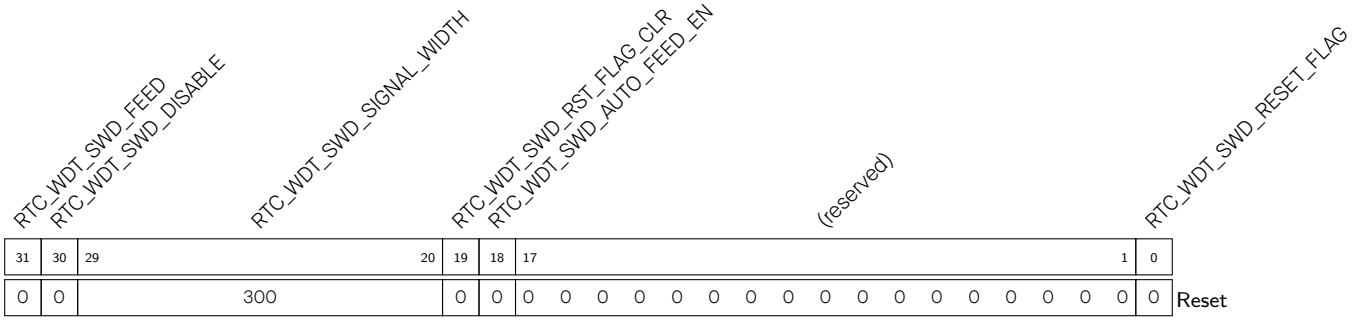
RTC_WDT_WKEY Configure this field to lock or unlock RWDT's configuration registers.

0x50D83AA1: unlock the RWDT configuration register

Other values: lock the RWDT configuration register which can't be modified by software.

(R/W)

Register 14.8. RTC_WDT_SWD_CONFIG_REG (0x0020)



RTC_WDT_SWD_RESET_FLAG Represents the SWD whether or not generate the reset signal

0: No

1: Yes

(RO)

RTC_WDT_SWD_AUTO_FEED_EN Configure this bit to enable to feed SWD automatically by hardware.

0: Disable

1: Enable

(R/W)

RTC_WDT_SWD_RST_FLAG_CLR Configure this bit to clear SWD reset flag

0: Invalid

1: clear the reset flag

(WT)

RTC_WDT_SWD_SIGNAL_WIDTH Configures the SWD signal length that outputs to the analog circuit.

Measurement unit: RTC_DYN_FAST_CLK

(R/W)

RTC_WDT_SWD_DISABLE Configure this bit to disable the SWD.

0: Enable the SWD

1: Disable the SWD

(R/W)

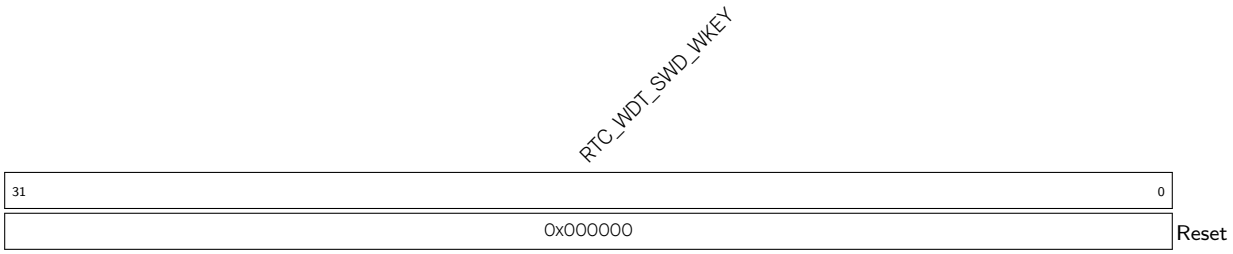
RTC_WDT_SWD_FEED Configure this bit to feed the SWD.

0: Invalid

1: Feed SWD

(WT)

Register 14.9. RTC_WDT_SWD_WPROTECT_REG (0x0024)



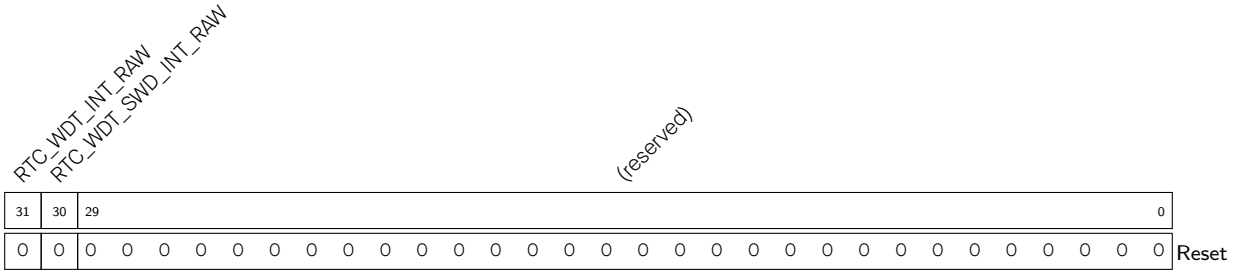
RTC_WDT_SWD_WKEY Configure this field to lock or unlock SWD's configuration registers.

0x50D83AA1: unlock the SWD configuration register.

Other values: lock the SWD configuration register which can't be modified by the software.

(R/W)

Register 14.10. RTC_WDT_INT_RAW_REG (0x0028)



RTC_WDT_SWD_INT_RAW Represents the SWD whether or not generates timeout interrupt.

0: No

1: Yes

(R/WTC/SS)

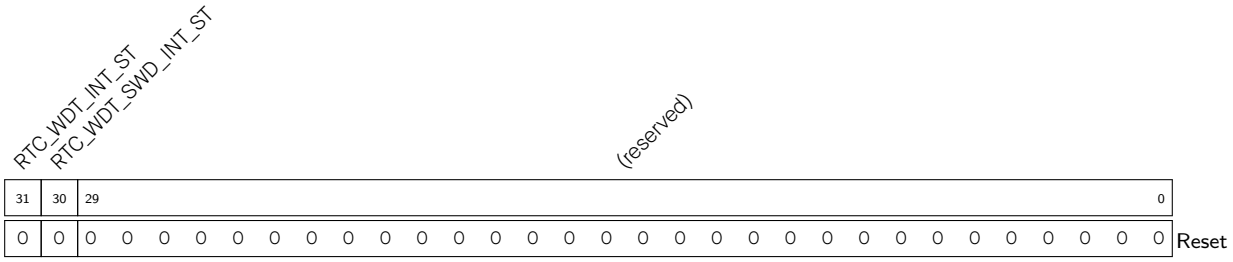
RTC_WDT_INT_RAW Represents the RWDT whether or not generates timeout interrupt.

0: No

1: Yes

(R/WTC/SS)

Register 14.11. RTC_WDT_INT_ST_REG (0x002C)



RTC_WDT_SWD_INT_ST Represents the SWD whether or not generates and sends timeout interrupt to CPU.

0: No

1: Yes

(RO)

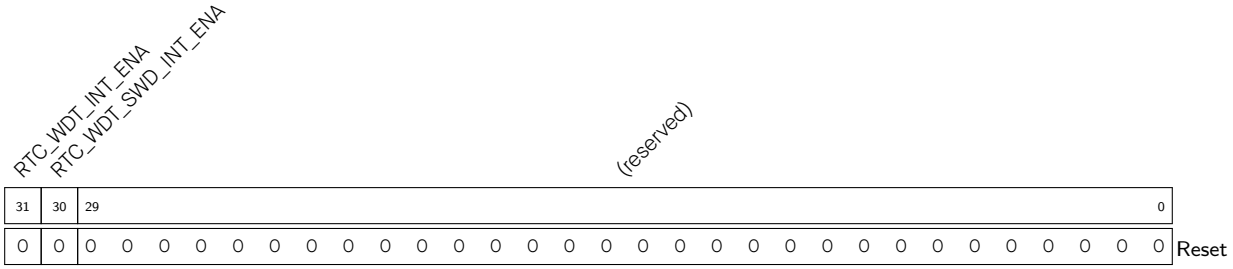
RTC_WDT_INT_ST Represents the RWDT whether or not generates and sends timeout interrupt to CPU.

0: No

1: Yes

(RO)

Register 14.12. RTC_WDT_INT_ENA_REG (0x0030)



RTC_WDT_SUPER_WDT_INT_ENA Configure whether or not to enable the SWD to send timeout interrupt.

0: Disable

1: Enable

(R/W)

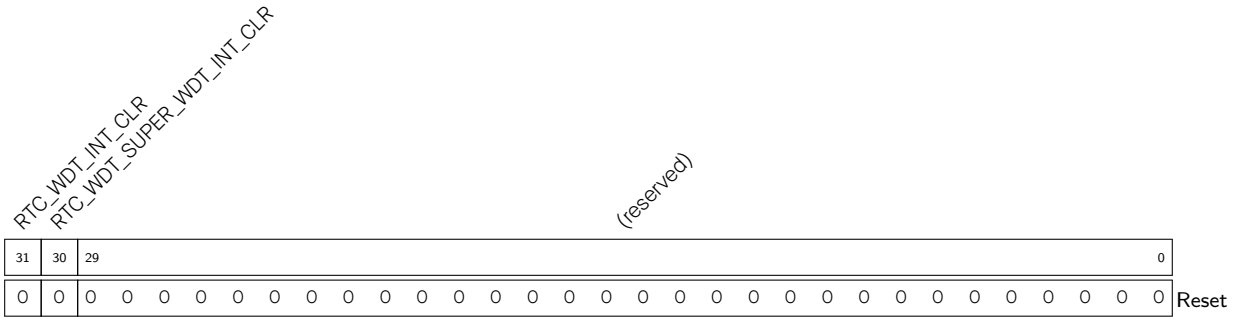
RTC_WDT_INT_ENA Configure whether or not to enable the RWDT to send timeout interrupt.

0: Disable

1: Enable

(R/W)

Register 14.13. RTC_WDT_INT_CLR_REG (0x0034)



RTC_WDT_SWD_INT_CLR Configure whether to clear the timeout interrupt signal sent by SWD to CPU.

0: No

1: Yes

(WT)

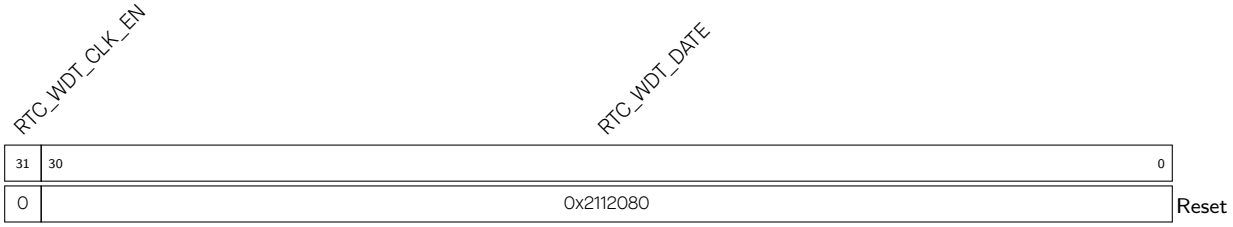
RTC_WDT_INT_CLR Configure whether to clear the timeout interrupt signal sent by RWDT to CPU.

0: No

1: Yes

(WT)

Register 14.14. RTC_WDT_DATE_REG (0x03FC)



RTC_WDT_DATE Version control register (R/W)

RTC_WDT_CLK_EN Reserved (R/W)

Chapter 15

Permission Control (PMS)

15.1 Overview

The permission control of ESP32-H2 can be divided into two parts: PMP (Physical Memory Protection) and APM (Access Permission Management).

The areas managed by PMP and APM are shown in Table 15.1-1.

Table 15.1-1. Management Areas of PMP and AMP

Masters \ Slaves	ROM	HP SRAM	LP SRAM	CPU_PERI[1]	HP_PERI[2]	LP_PERI[3]	EX_MEM[4]
CPU	PMP	PMP	PMP + APM	PMP + APM	PMP + APM	PMP + APM	PMP
Other masters[5]	N/A	APM	APM	N/A	N/A	N/A	N/A

[1] Peripheral registers in the 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.

[5] Masters that can request access to the bus, such as GDMA, MEM_MONITOR. For a complete list of masters, please refer to Table 15.4-1.

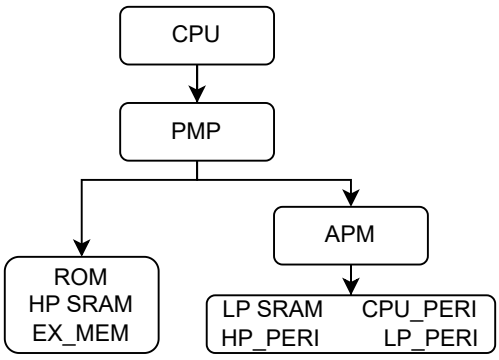


Figure 15.1-1. PMP-APM Management Relation

For the CPU, the permission management relation between PMP and APM is shown in Figure 15.1-1. PMP manages the CPU's access to all address spaces. APM does not manage the CPU's access to ROM, HP SRAM, and EX_MEM. If the CPU needs to access ROM, HP SRAM, and EX_MEM, it needs permission only from PMP; if it needs to access LP SRAM and other address spaces, it needs to pass PMP's permission management first and then the APM's. If the PMP check fails, APM check will not be triggered.

PMP related registers are located inside the CPU and can be read or configured with special instructions. For how to configure PMP, please refer to Chapter *ESP-RISC-V CPU* > Section Physical Memory Protection.

The following sections of this chapter will describe the functions and configurations of the APM module.

The APM module contains two parts: the TEE (Trusted Execution Environment) controller and the APM controller. Each of them contains its own register module: TEE register module and APM register module.

- The TEE controller is responsible for configuring the security mode of a particular master in ESP32-H2 (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 is responsible for managing a master's access permissions (read/write/execute) 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 15.4-1), security mode, access address, access permissions, etc, the APM controller determines whether access is allowed.

TEE related registers are used to configure the security mode of each master, and the APM related registers are used to specify the access permission and access address range of each security mode. With TEE controller and APM controller, ESP32-H2 can precisely control the access permission of all masters to memory and peripheral registers.

15.2 Features

ESP32-H2's TEE controller has the following features:

- Four security modes available for the masters
- Security mode configuration for up to 32 masters

ESP32-H2's APM controller has the following features:

- Access permission configuration for up to 16 address ranges
- Access management to internal memory and peripheral registers
- Interrupt function on illegal access
- Exception information record

15.3 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 15.3-2. Comparison Between TEE and REE

Aspect	TEE	REE
Security	Enhanced security	Normal security
Access permission	The master in TEE mode always has read, write, and execute permissions in the address range.	Different levels of access permissions are configurable by software.

15.4 Functional Description

15.4.1 TEE Controller Functional Description

ESP32-H2 provides four security modes: TEE, REE0, REE1, and REE2.

For the CPU to access memory or peripheral registers, first select the machine mode or user mode of the CPU, then configure its security mode. 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 CPU is in machine mode, its security mode is TEE mode.
- When the CPU is in user mode, its security mode is REE mode. To specify REE0, REE1 or REE2 mode, TEE_M0_MODE of TEE_M0_MODE_CTRL_REG should be configured:
  - If TEE_M0_MODE is set to 0, which is TEE mode, the valid mode that actually takes effect in the CPU user mode is REE0.
  - if TEE_M0_MODE is set to 1, 2 or 3, which is in REE mode, its security mode is REE0, REE1 and REE2 respectively.

As for other masters, security mode can be set by configuring TEE_M*n*_MODE. *n* here equals the ID number of master in Table 15.4-1.

Table 15.4-1. Master Access Source

Value	Source
0	CPU
1	Reserved
2	Reserved
3	Reserved
4	Reserved
5	MEM_MONITOR
6	TRACE
7 ~ 15	Reserved
16 ~ 31	See the peripherals corresponding to the values 0 ~ 15 in Chapter 3 *GDMA Controller (GDMA)* > Table 3.4-1 *Selecting Peripherals via Register Configuration*. For example, 16 corresponds to the peripheral with value 0 in that table, and 17 corresponds to the peripheral with value 1 in that table, and so on.

15.4.2 APM Controller Functional Description

15.4.2.1 Architecture

Figure 15.4-1 shows the architecture of the APM controller and the access paths managed by it.

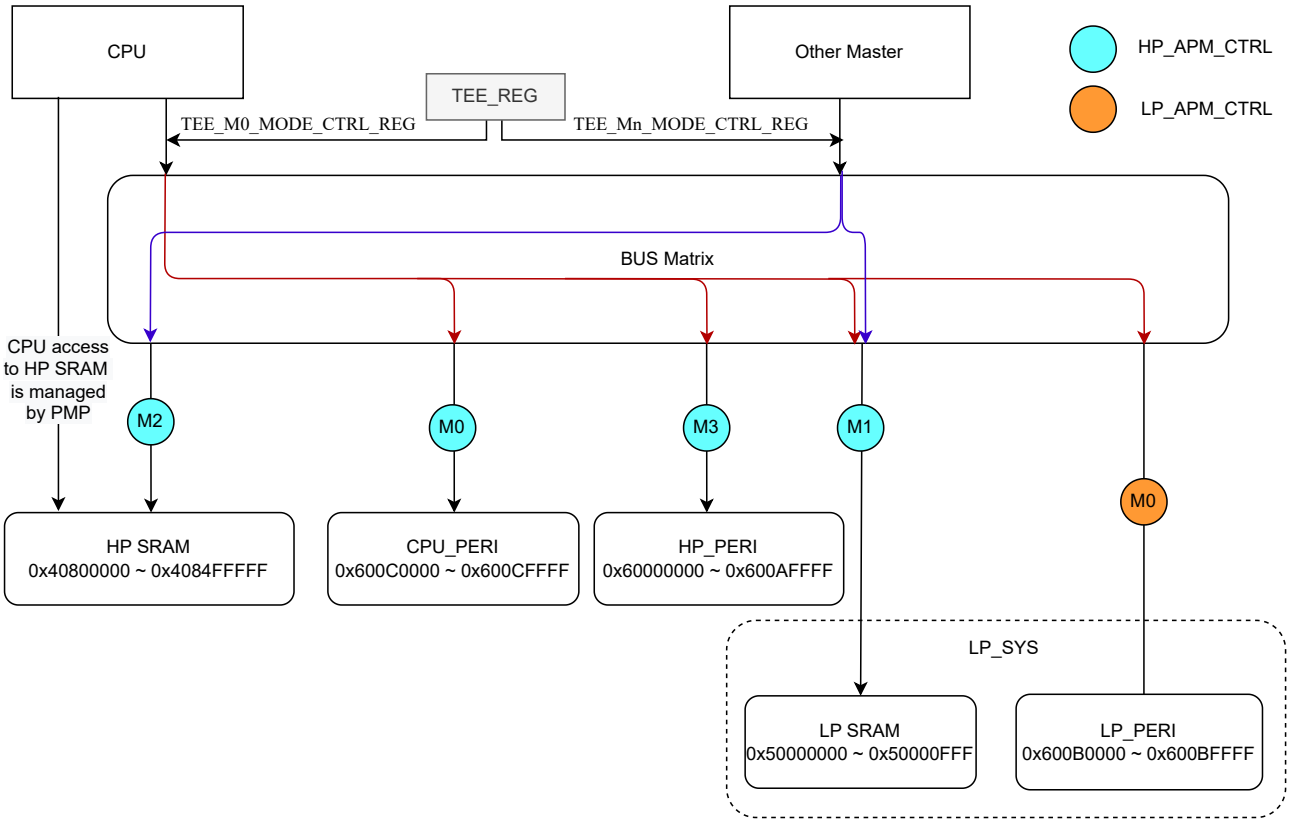


Figure 15.4-1. APM Controller Architecture

As shown in the figure, the APM controller contains two functional modules: HP_APM_CTRL and LP_APM_CTRL, configured by the register modules HP_APM_REG and LP_APM_REG, respectively.

- HP_APM_CTRL manages four access paths, namely M0 – M3 in the figure. Permission management of each path can be enabled by configuring HP_APM_FUNC_CTRL_REG (enabled by default).
- LP_APM_CTRL manages one access path, namely M0 in the figure. Permission management of this path can be enabled by configuring LP_APM_FUNC_CTRL_REG (enabled by default).

Table 15.4-2 below shows the detailed information of the two functional modules:

Table 15.4-2. Configuring Functional Modules

Functional Modules	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

15.4.2.2 Address Ranges

HP_APM_REG register module can configure up to 16 address ranges for functional module HP_APM_CTRL. The start and end address for 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 the (*n*+1)th region. The first address range is enabled by default.

LP_APM_REG register module can configure up to four address ranges for functional module LP_APM_CTRL. The start and end address for 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 the (*n*+1)th region. The first address range is enabled by default.

When configuring the address ranges, the address requires 4-byte alignment (the lower two bits of the address are 0). For example, the address range could be set as 0x4080000C ~ 0x40808774 or 0x600C0008 ~ 0×600CFF70.

15.4.2.3 Access Permissions of Address Ranges

For each address range, the access permissions (read/write/execute) can be configured for different security modes:

- The master in TEE mode always has read, write, and execute permissions in the address range.
- For master in REE0, REE1 or REE2 mode, access permissions can be configured in HP_APM_REGION*n*_ATTR_REG or LP_APM_REGION*n*_ATTR_REG based 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 path HP_APM_CTRL M0-M3 shown in Figure 15.4-1 should follow the address ranges and access permissions of each address range configured in the register module HP_APM_REG. Likewise, the permission management of data path LP_APM_CTRL M0 shown in Figure 15.4-1 should follow the address ranges and access permissions of each address range configured in the register module LP_APM_REG.

As Figure 15.4-1 shows, all masters access HP_MEM through the HP_APM_CTRL M1 path. Suppose that HP_APM_M1_FUNC_EN is enabled and a master in REE1 mode needs to access LP_MEM. The whole process is as follows:

1. HP_APM_CTRL M1 will first determine 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 valid (that is, 1). If valid, the read request will be allowed, otherwise 0 will be returned.

The address ranges configured may overlap. For example, region 1 and region 2 overlap. If region 1 is set to be unreadable and region 2 readable, then the overlapping area of region 1 and region 2 is readable. The same rules apply for write and execute permissions.

Note:

- When powered up, only the CPU is in TEE mode by default, and the 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 15.7 *Register Summary* can only be configured by the masters that are in TEE security mode.

15.5 Programming Procedure

For a master to access memory or peripheral registers, follow the programming procedures below:

1. Set the CPU to machine mode (i.e., TEE mode).
2. Choose the security mode of the master by configuring TEE_M*n*_MODE. The master ID *n* is defined in Table 15.4-1.
3. Configure the start and end address for access address ranges by setting HP_APM_REGION*n*_ADDR_START, HP_APM_REGION*n*_ADDR_END, or LP_APM_REGION*n*_ADDR_START, LP_APM_REGION*n*_ADDR_END.
4. Configure the access permissions of each region by configuring HP_APM_REGION*n*_ATTR_REG or LP_APM_REGION*n*_ATTR_REG.
5. Set the bit *n* of HP_APM_REGION_FILTER_EN_REG or LP_APM_REGION_FILTER_EN_REG to enable region *n*.
6. Configure HP_APM_ FUNC_CTRL_REG or LP_APM _FUNC_CTRL_REG to enable permission management of 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 ~ 0x4080F000:

1. Configure the CPU to machine mode (ie. TEE mode).
2. According to the master ID number in Table 15.4-1, set TEE_M19_MODE to be 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 0x4080F000, respectively.
4. Set HP_APM_REGION3_RO_W and HP_APM_REGION3_RO_R to 1.
5. Set the bit 3 of HP_APM_REGION_FILTER_EN to 1.
6. Set HP_APM_M1_FUNC_EN to 1.

15.6 Illegal Access and Interrupts

If the information carried on the bus is inconsistent with the configuration, ESP32-H2 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 these information can be obtained from relevant registers listed in Section 15.7 *Register Summary*.

Take the access path HP_APM_CTRL M0 as an example. When 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, bit1 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 bit0 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 bit0 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-H2's APM controller can generate five interrupt signals, which will be sent to *Interrupt Matrix (INTMTX)*:

- HP_APM_M0_INTR
- HP_APM_M1_INTR
- HP_APM_M2_INTR
- HP_APM_M3_INTR
- LP_APM_M0_INTR

These five interrupt signals correspond to the controlled access paths shown in Figure 15.4-1. If an illegal access occurs in a controlled access path, the corresponding interrupt will be generated.

15.7 Register Summary

15.7.1 APM Registers of HP System (HP_APM_REG)

The addresses in this section are relative to the Access Permission Management Controller (HP_APM) base address provided in Table 4.3-2 in Chapter 4 *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_REGION*n*_ADDR_START_REG (*n*: 0-15)	Region address register	0x0004+0xC*n	R/W
HP_APM_REGION*n*_ADDR_END_REG (*n*: 0-15)	Region address register	0x0008+0xC*n	R/W
HP_APM_REGION*n*_ATTR_REG (*n*: 0-15)	Region access permissions configuration register	0x000C+0xC*n	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	0x0108	R/W
Clock Gating Registers			
HP_APM_CLOCK_GATE_REG	Clock gating register	0x010C	R/W
Version Control Registers			

Name	Description	Address	Access
HP_APM_DATE_REG	Version control register	0x07FC	R/W

15.7.2 APM Registers of LP System (LP_APM_REG)

The addresses in this section are relative to the Low-Power Access Permission Management (LP_APM) base address provided in Table 4.3-2 in Chapter 4 *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_REGION*n*_ADDR_START_REG (*n*: 0-3)	Region address register	0x0004+0xC*n	R/W
LP_APM_REGION*n*_ADDR_END_REG (*n*: 0-3)	Region address register	0x0008+0xC*n	R/W
LP_APM_REGION*n*_ATTR_REG (*n*: 0-3)	Region access permissions configuration register	0x000C+0xC*n	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			
LP_APM_CLOCK_GATE_REG	Clock gating register	0x00EC	R/W
Version Control Registers			
LP_APM_DATE_REG	Version control register	0x00FC	R/W

15.7.3 TEE Registers of HP System

The addresses in this section are relative to the Trusted Execution Environment (TEE) Register provided in Table 4.3-2 in Chapter 4 *System and Memory*.

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

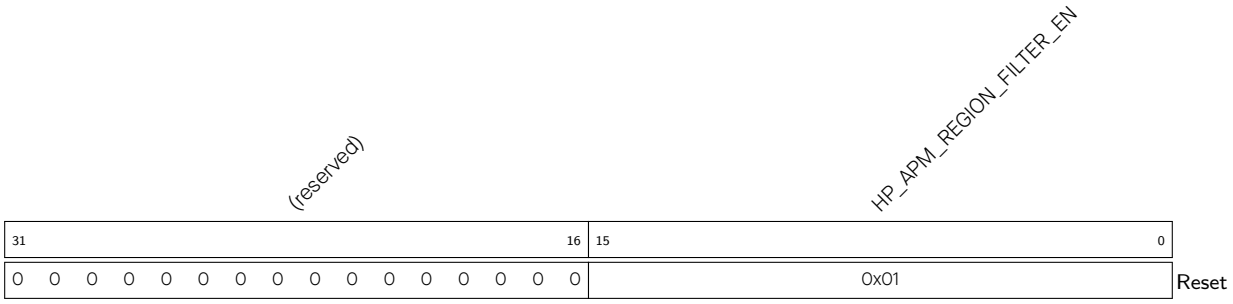
Name	Description	Address	Access
Configuration Registers			
TEE_M*n*_MODE_CTRL_REG (*n*: 0-31)	Security mode configuration register	0x0000+0x4*n	R/W

Name	Description	Address	Access
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

15.8 Registers

15.8.1 APM Registers of HP System (HP_APM_REG)

Register 15.1. HP_APM_REGION_FILTER_EN_REG (0x0000)



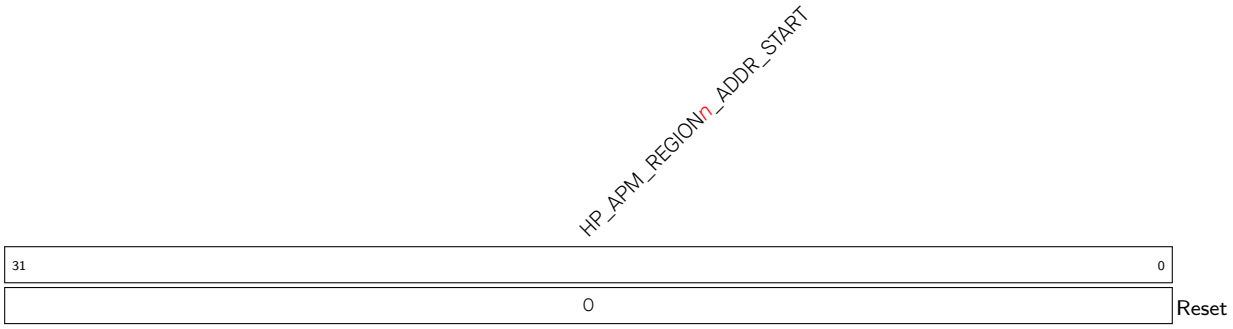
HP_APM_REGION_FILTER_EN Configure bit *n* (0-15) to enable region *n* (0-15).

0: disable

1: enable

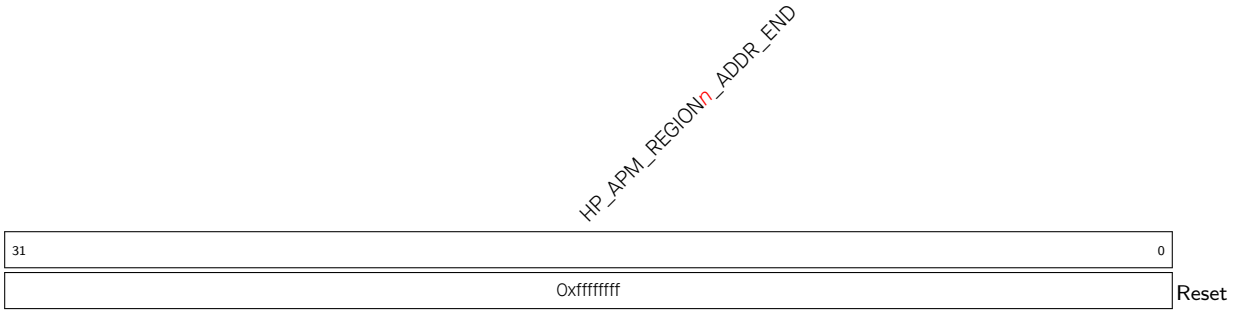
(R/W)

Register 15.2. HP_APM_REGION*n*_ADDR_START_REG (*n*: 0-15) (0x0004+0xC**n*)



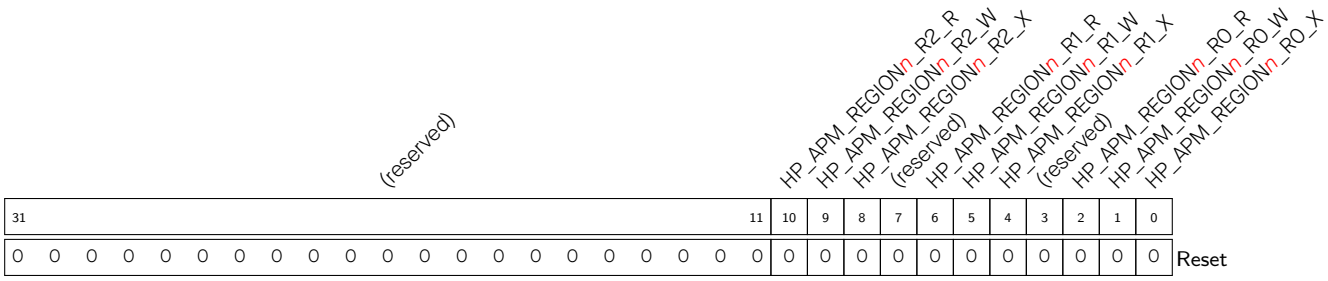
HP_APM_REGION*n*_ADDR_START Configures the start address of region *n*. (R/W)

Register 15.3. 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 15.4. 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)

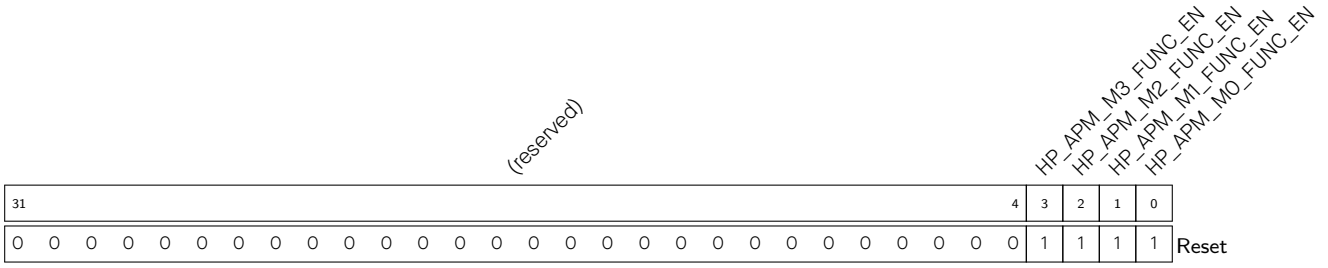
HP_APM_REGION*n*_R1_R Configures the read permission in region *n* in REE1 mode. (R/W)

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)

Register 15.5. HP_APM_FUNC_CTRL_REG (0x00C4)



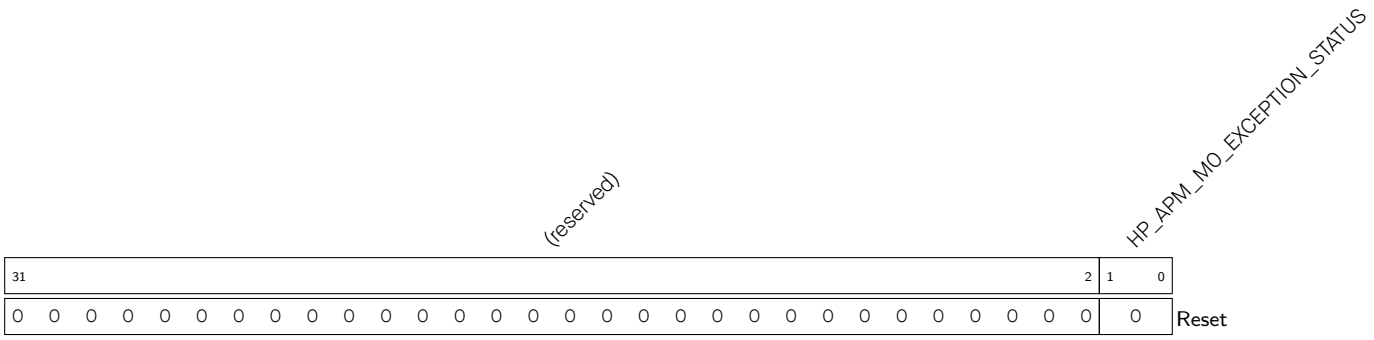
HP_APM_M0_FUNC_EN Configures to enable permission management for HP_APM_CTRL M0. (R/W)

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)

Register 15.6. HP_APM_M0_STATUS_REG (0x00C8)



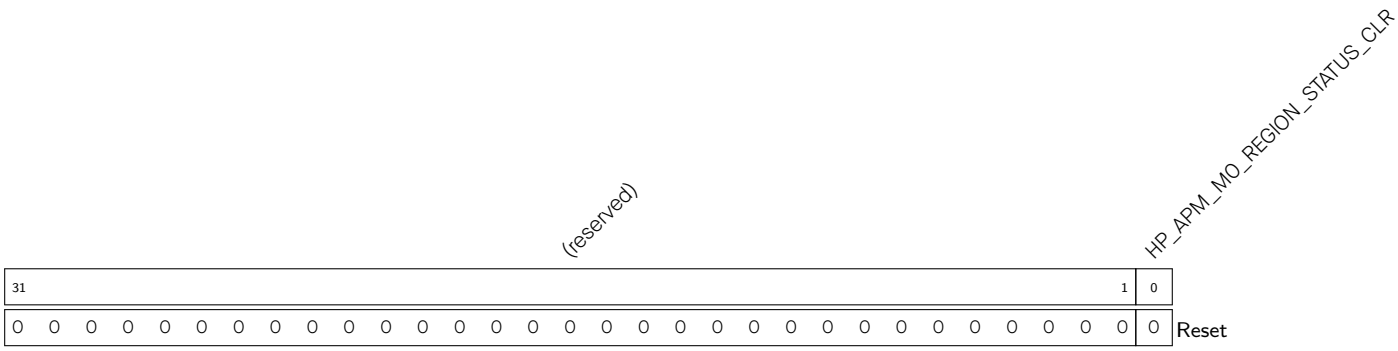
HP_APM_M0_EXCEPTION_STATUS Represents exception status.

bit0: 1 represents permission restrictions

bit1: 1 represents address out of bounds

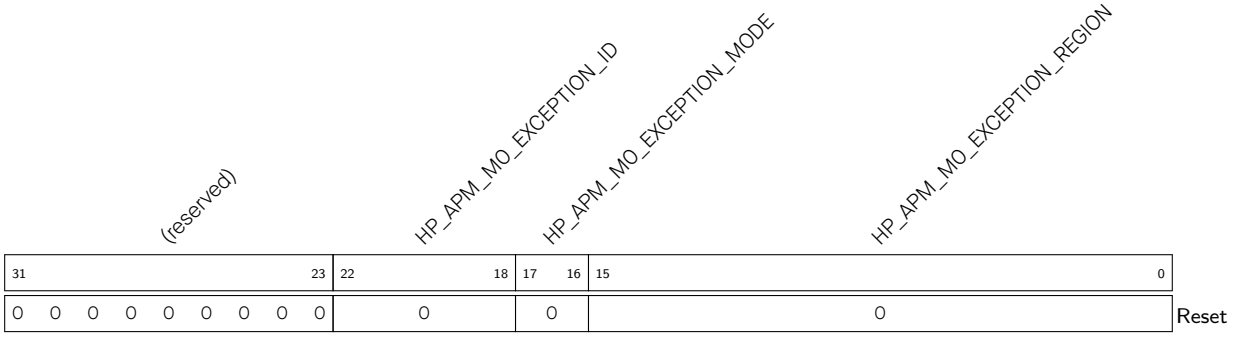
(RO)

Register 15.7. HP_APM_M0_STATUS_CLR_REG (0x00CC)



HP_APM_M0_REGION_STATUS_CLR Configures to clear exception status. (WT)

Register 15.8. HP_APM_M0_EXCEPTION_INFO0_REG (0x00D0)

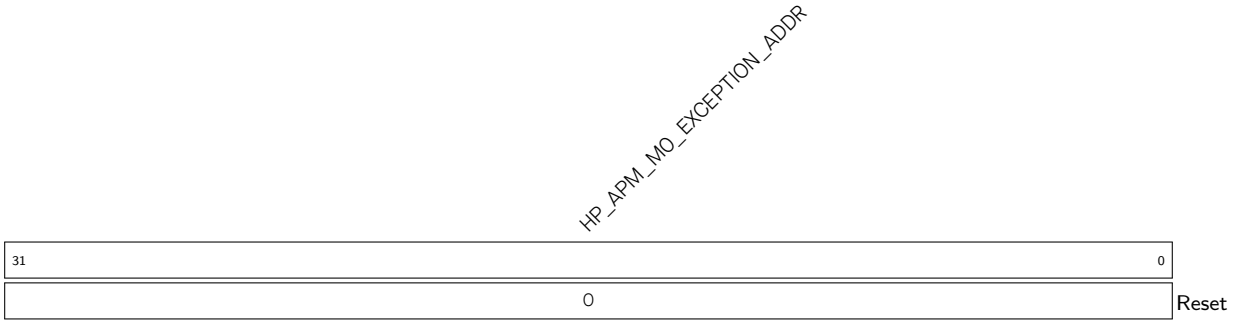


HP_APM_M0_EXCEPTION_REGION Represents the region where an exception occurs. (RO)

HP_APM_M0_EXCEPTION_MODE Represents the master's security mode when an exception occurs. (RO)

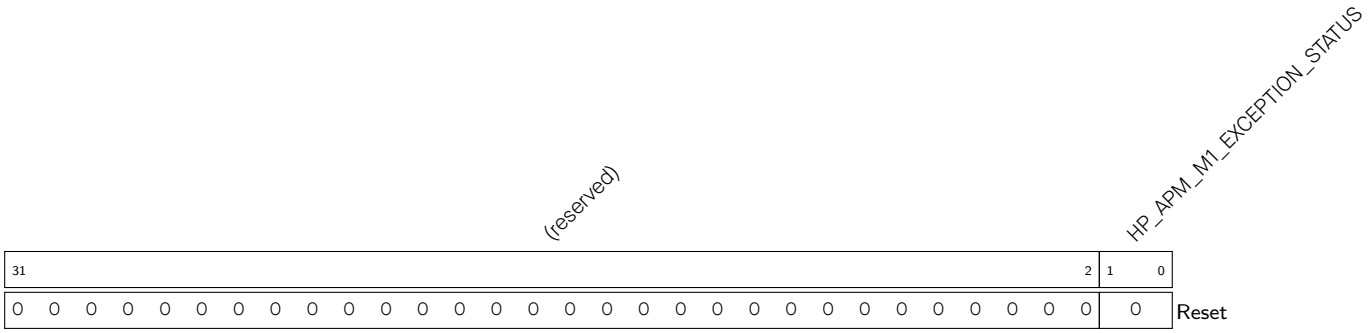
HP_APM_M0_EXCEPTION_ID Represents master ID when an exception occurs. (RO)

Register 15.9. HP_APM_M0_EXCEPTION_INFO1_REG (0x00D4)



HP_APM_M0_EXCEPTION_ADDR Represents the access address when an exception occurs. (RO)

Register 15.10. HP_APM_M1_STATUS_REG (0x00D8)



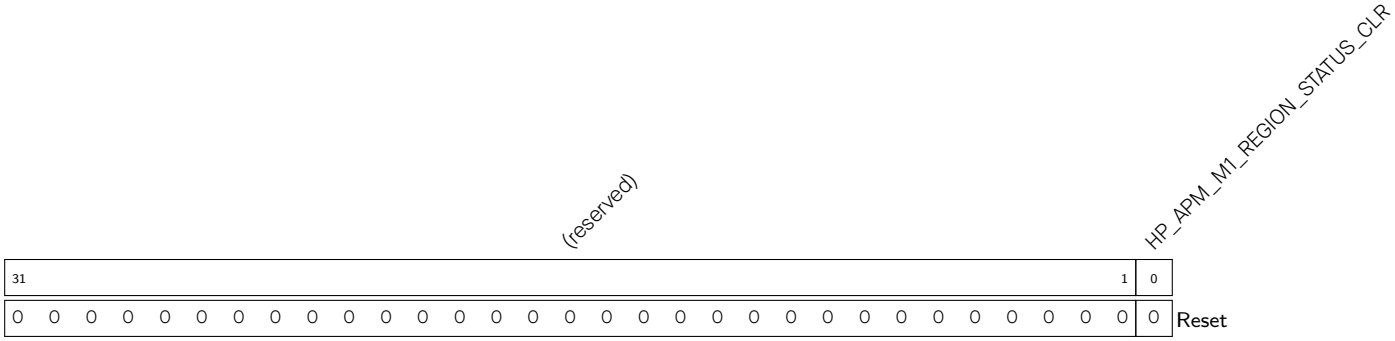
HP_APM_M1_EXCEPTION_STATUS Represents exception status.

bit0: 1 represents permission restrictions

bit1: 1 represents address out of bounds

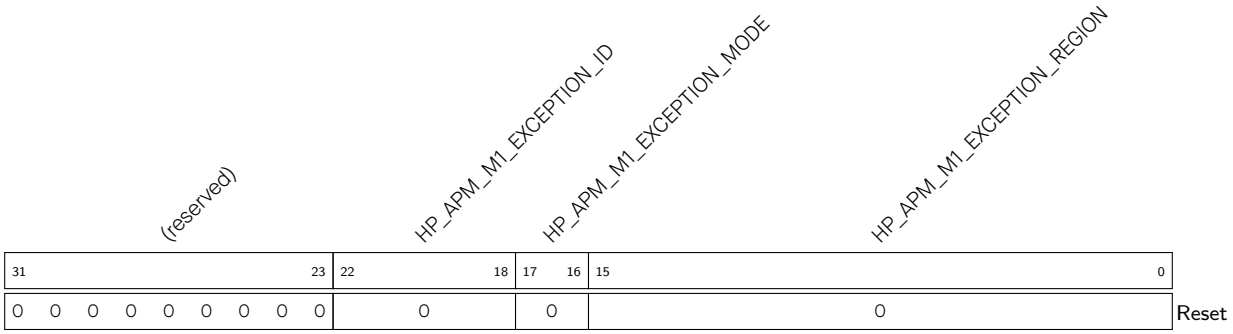
(RO)

Register 15.11. HP_APM_M1_STATUS_CLR_REG (0x00DC)



HP_APM_M1_REGION_STATUS_CLR Configures to clear exception status. (WT)

Register 15.12. HP_APM_M1_EXCEPTION_INFO0_REG (0x00E0)

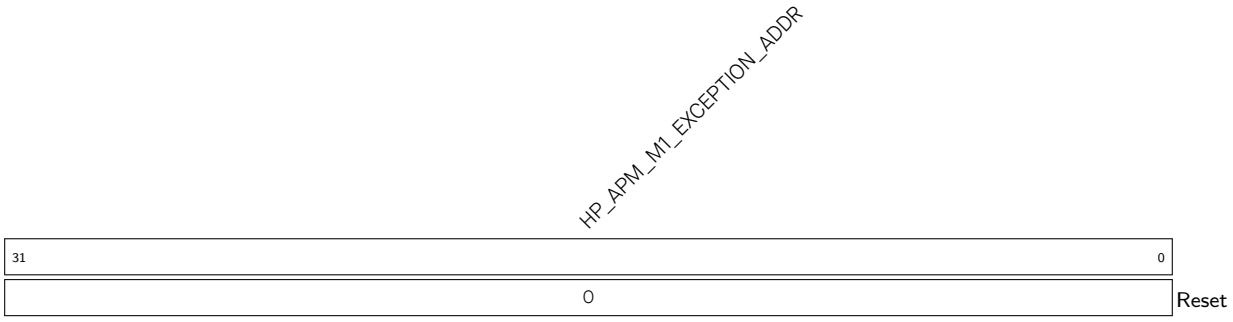


HP_APM_M1_EXCEPTION_REGION Represents the region where an exception occurs. (RO)

HP_APM_M1_EXCEPTION_MODE Represents the master's security mode when an exception occurs. (RO)

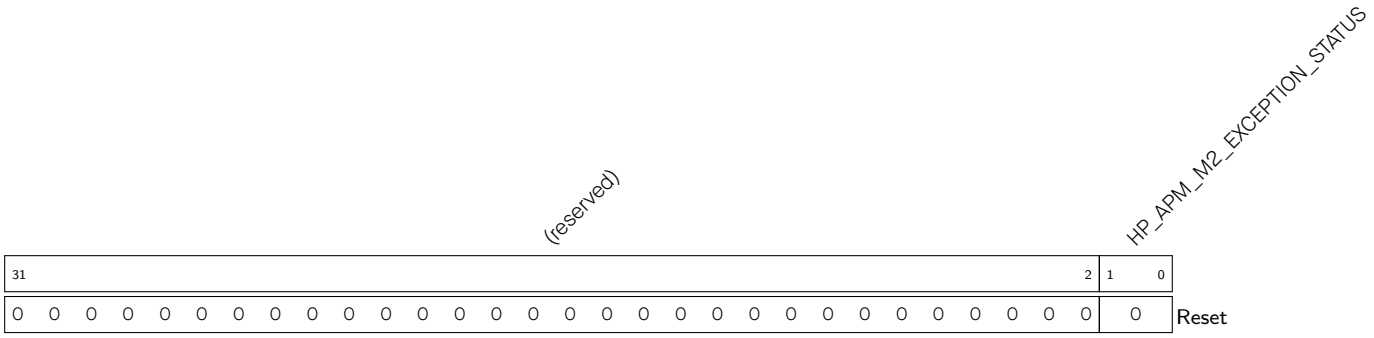
HP_APM_M1_EXCEPTION_ID Represents master ID when an exception occurs. (RO)

Register 15.13. HP_APM_M1_EXCEPTION_INFO1_REG (0x00E4)



HP_APM_M1_EXCEPTION_ADDR Represents the access address when an exception occurs. (RO)

Register 15.14. HP_APM_M2_STATUS_REG (0x00E8)



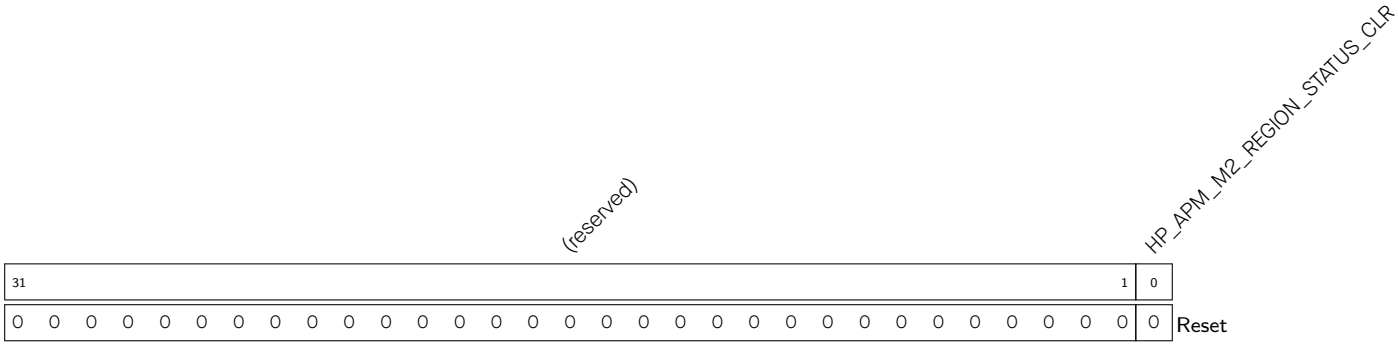
HP_APM_M2_EXCEPTION_STATUS Represents exception status.

bit0: 1 represents permission restrictions

bit1: 1 represents address out of bounds

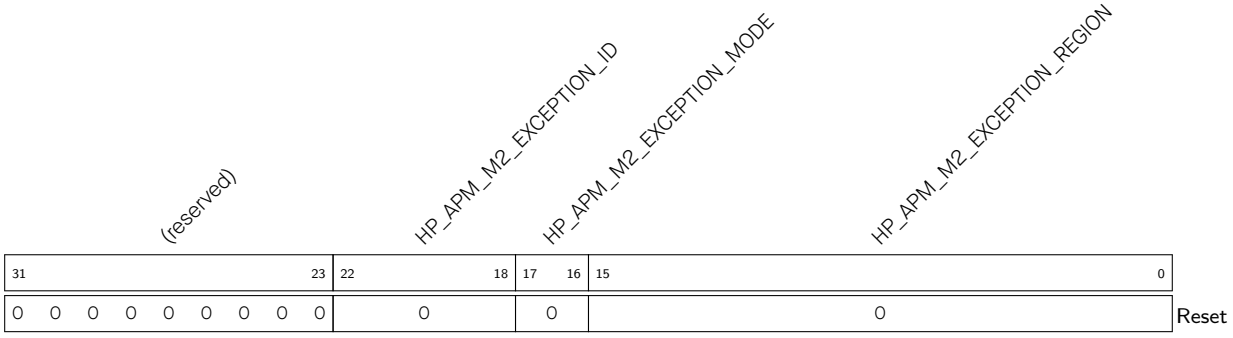
(RO)

Register 15.15. HP_APM_M2_STATUS_CLR_REG (0x00EC)



HP_APM_M2_REGION_STATUS_CLR Configures to clear exception status. (WT)

Register 15.16. HP_APM_M2_EXCEPTION_INFO0_REG (0x00F0)

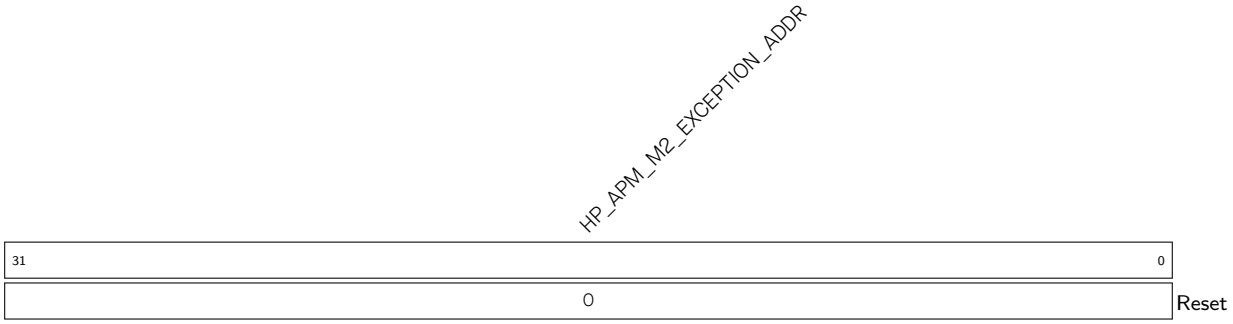


HP_APM_M2_EXCEPTION_REGION Represents the region where an exception occurs. (RO)

HP_APM_M2_EXCEPTION_MODE Represents the master's security mode when an exception occurs. (RO)

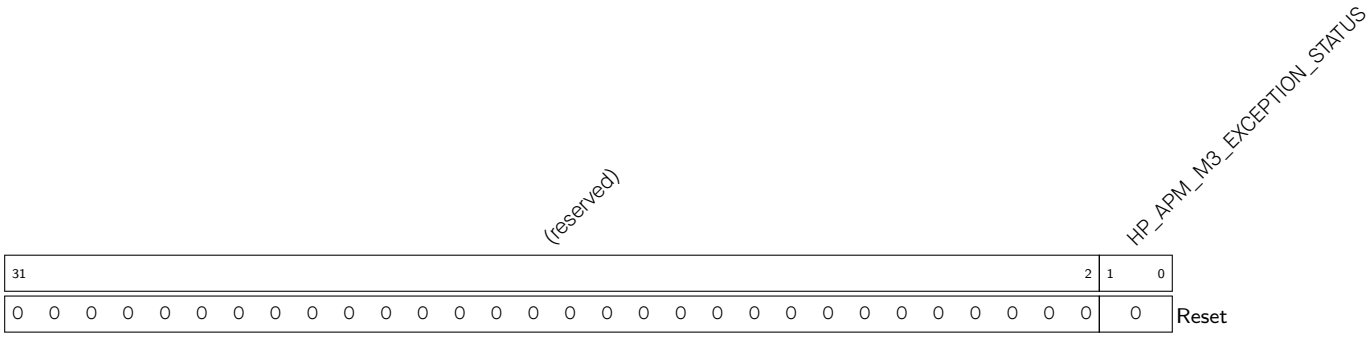
HP_APM_M2_EXCEPTION_ID Represents master ID when an exception occurs. (RO)

Register 15.17. HP_APM_M2_EXCEPTION_INFO1_REG (0x00F4)



HP_APM_M2_EXCEPTION_ADDR Represents the access address when an exception occurs. (RO)

Register 15.18. HP_APM_M3_STATUS_REG (0x00F8)



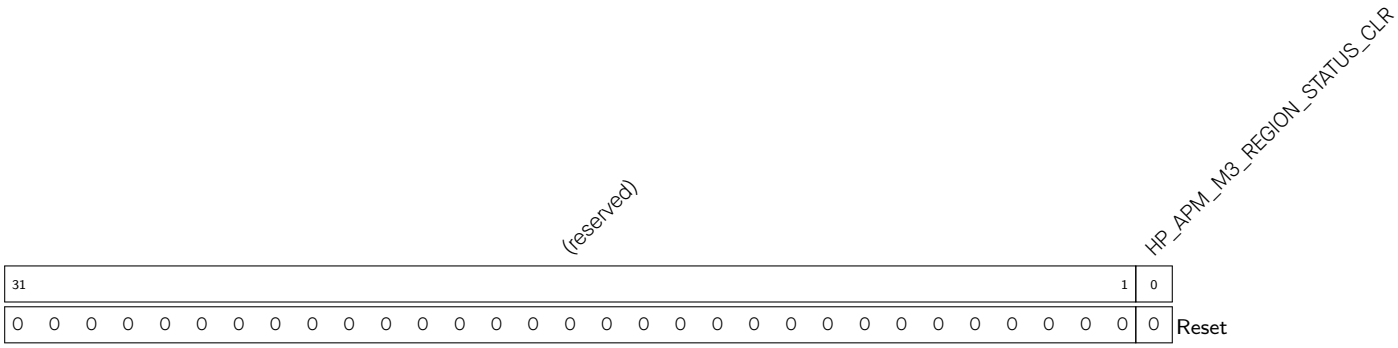
HP_APM_M3_EXCEPTION_STATUS Represents exception status.

bit0: 1 represents permission restrictions

bit1: 1 represents address out of bounds

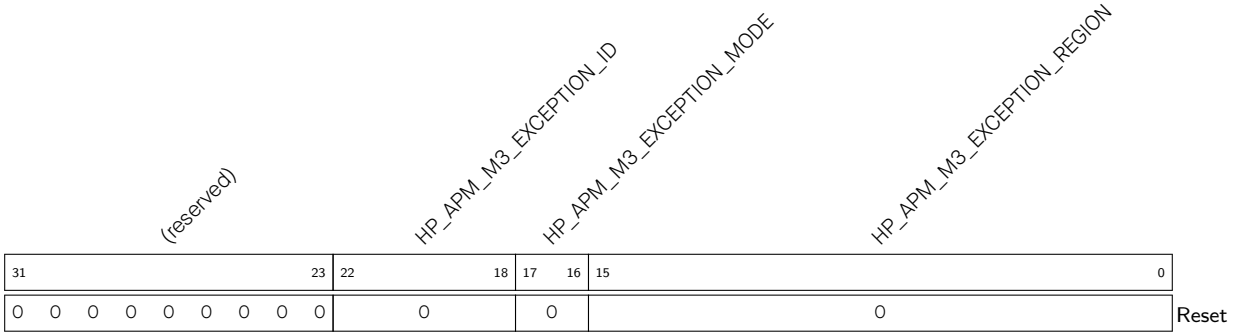
(RO)

Register 15.19. HP_APM_M3_STATUS_CLR_REG (0x00FC)



HP_APM_M3_REGION_STATUS_CLR Configures to clear exception status. (WT)

Register 15.20. HP_APM_M3_EXCEPTION_INFO0_REG (0x0100)

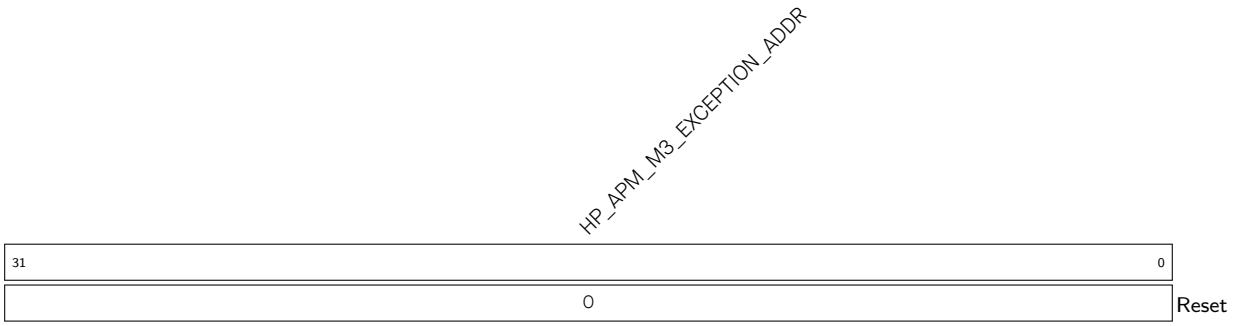


HP_APM_M3_EXCEPTION_REGION Represents the region where an exception occurs. (RO)

HP_APM_M3_EXCEPTION_MODE Represents the master's security mode when an exception occurs. (RO)

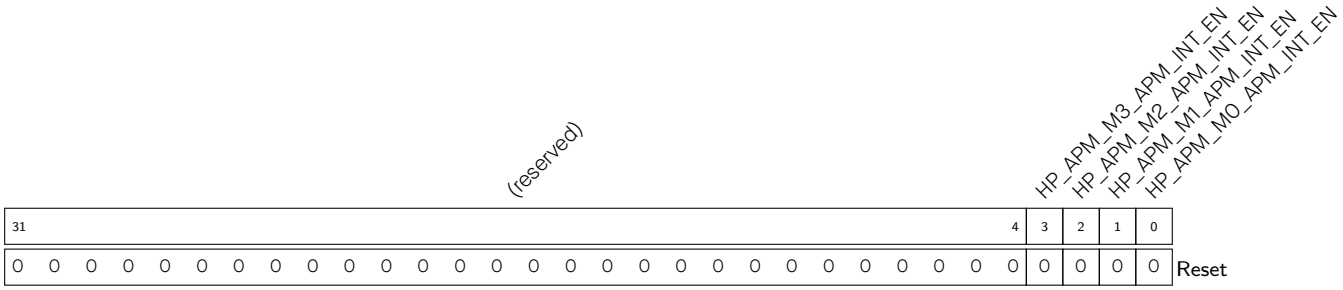
HP_APM_M3_EXCEPTION_ID Represents master ID when an exception occurs. (RO)

Register 15.21. HP_APM_M3_EXCEPTION_INFO1_REG (0x0104)



HP_APM_M3_EXCEPTION_ADDR Represents the access address when an exception occurs. (RO)

Register 15.22. HP_APM_INT_EN_REG (0x0108)



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)

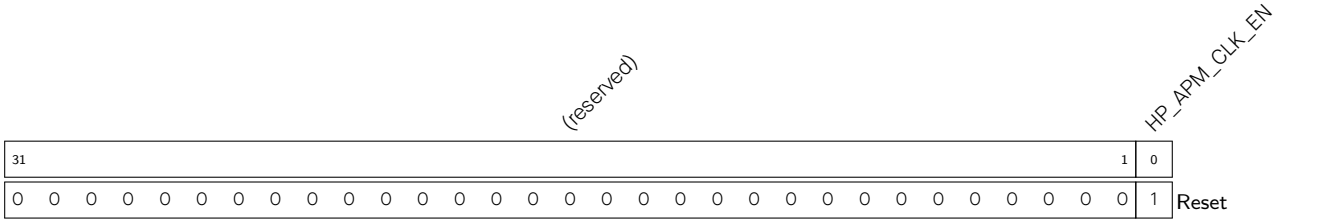
HP_APM_M3_APM_INT_EN Configures to enable HP_APM_CTRL M3 interrupt.

0: disable

1: enable

(R/W)

Register 15.23. HP_APM_CLOCK_GATE_REG (0x010C)



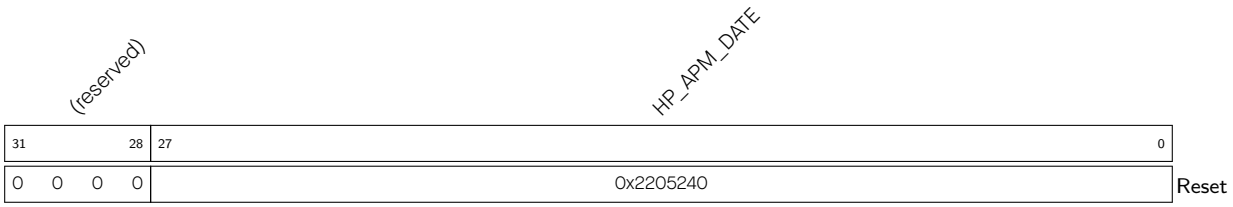
HP_APM_CLK_EN Configures whether to keep the clock always on.

0: enable automatic clock gating

1: keep the clock always on

(R/W)

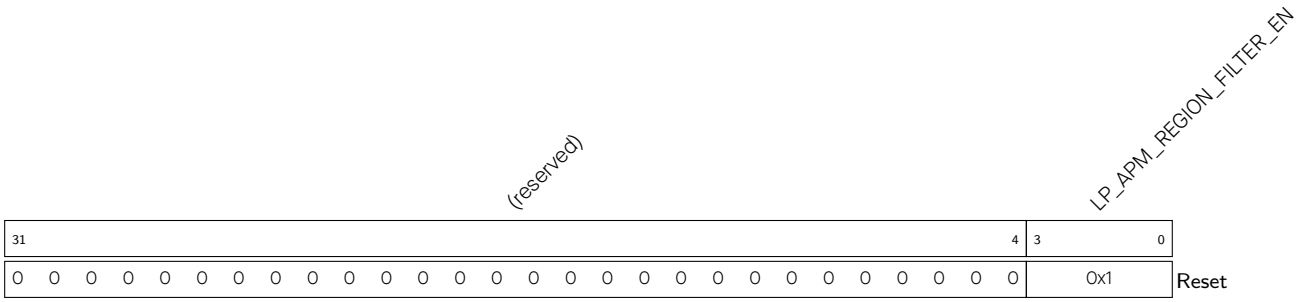
Register 15.24. HP_APM_DATE_REG (0x07FC)



HP_APM_DATE Version control register. (R/W)

15.8.2 APM Registers of LP System (LP_APM_REG)

Register 15.25. LP_APM_REGION_FILTER_EN_REG (0x0000)



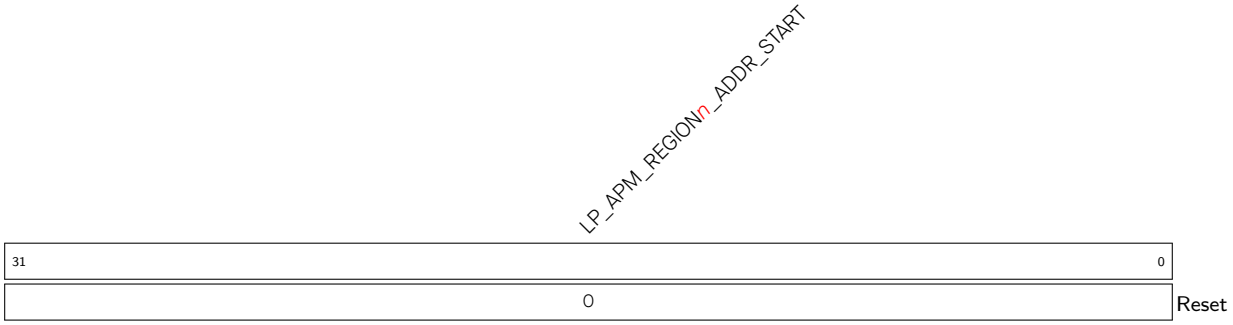
LP_APM_REGION_FILTER_EN Configure bit *n* (0-3) to enable region *n* (0-3).

0: Disable

1: Enable

(R/W)

Register 15.26. LP_APM_REGION*n*_ADDR_START_REG (*n*: 0-3) (0x0004+0xC**n*)

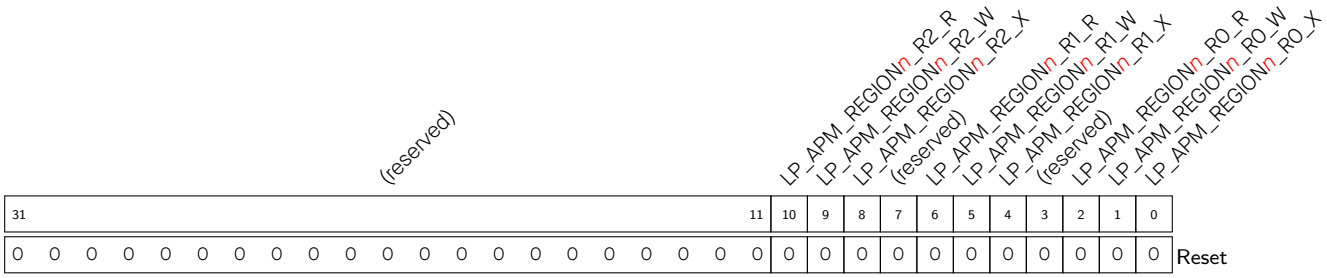


LP_APM_REGION*n*_ADDR_START Configures the start address of region *n*. (R/W)

Register 15.27. 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 15.28. 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)

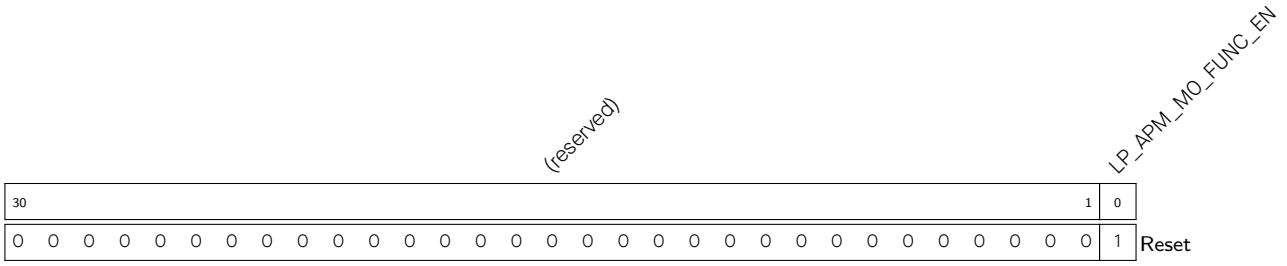
LP_APM_REGION*n*_R1_R Configures the read permission in region *n* in REE1 mode. (R/W)

LP_APM_REGION*n*_R2_X Configures the execution permission in region *n* in REE2 mode. (R/W)

LP_APM_REGION*n*_R2_W Configures the write permission in region *n* in REE2 mode. (R/W)

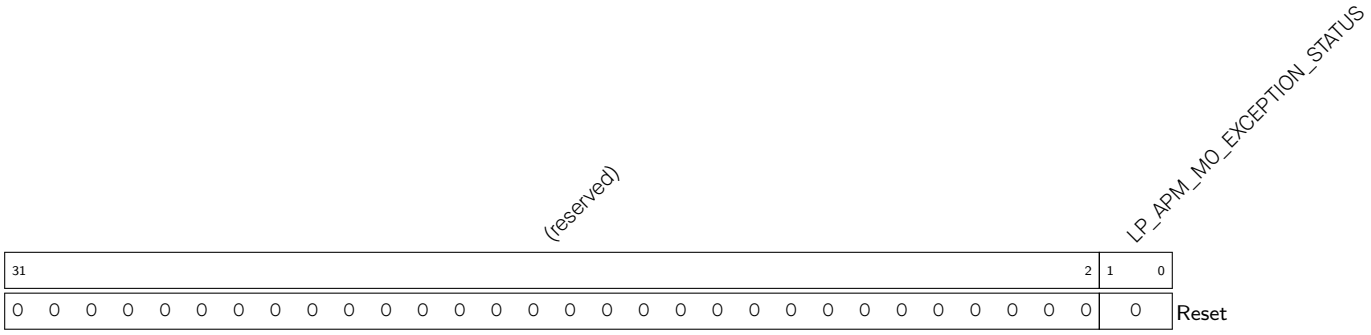
LP_APM_REGION*n*_R2_R Configures the read permission in region *n* in REE2 mode. (R/W)

Register 15.29. LP_APM_FUNC_CTRL_REG (0x00C4)



LP_APM_M0_FUNC_EN Configures to enable permission management for LP_APM_CTRL M0. (R/W)

Register 15.30. LP_APM_M0_STATUS_REG (0x00C8)



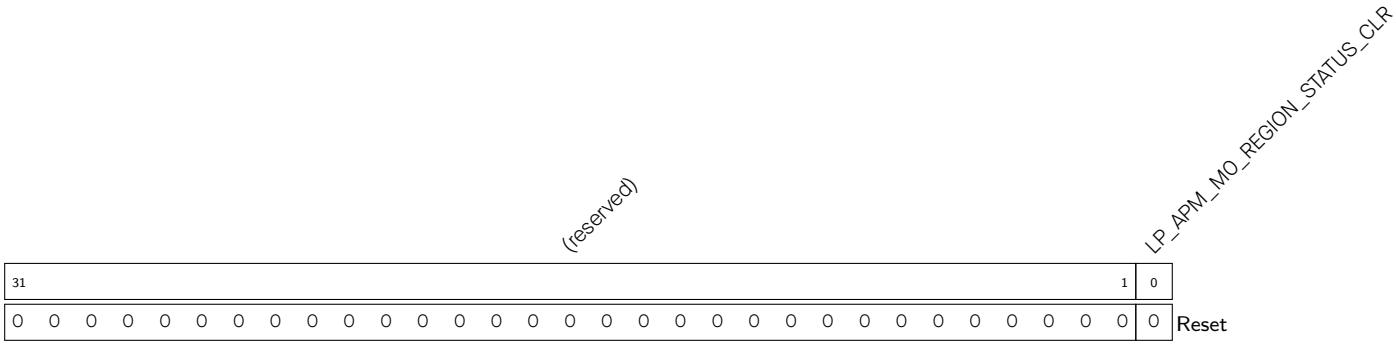
LP_APM_M0_EXCEPTION_STATUS Represents exception status.

bit0: 1 represents permission restrictions

bit1: 1 represents address out of bounds

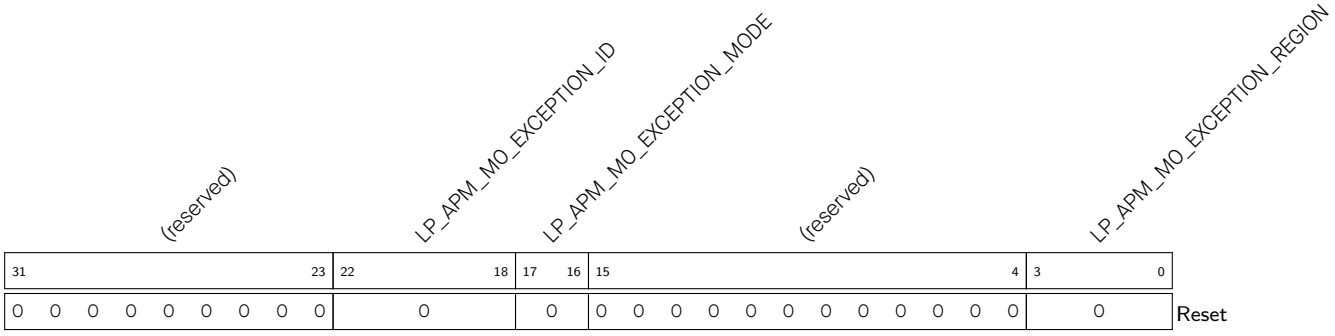
(RO)

Register 15.31. LP_APM_M0_STATUS_CLR_REG (0x00CC)



LP_APM_M0_REGION_STATUS_CLR Configures to clear exception status. (WT)

Register 15.32. LP_APM_M0_EXCEPTION_INFO0_REG (0x00D0)

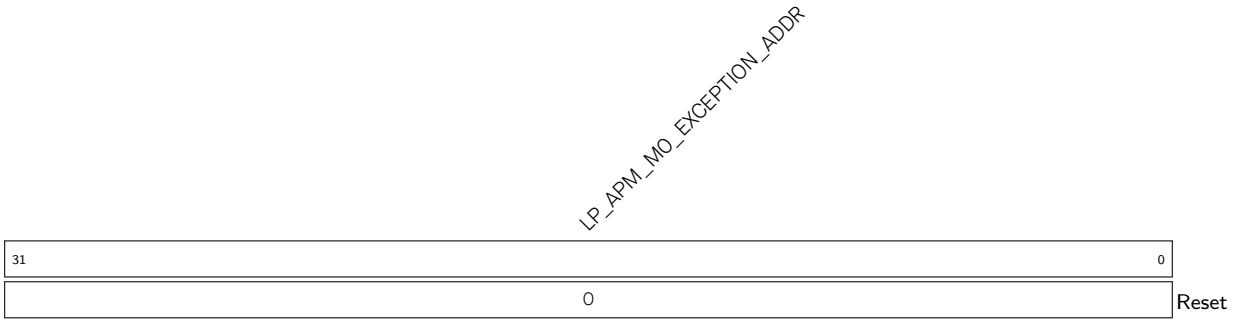


LP_APM_M0_EXCEPTION_REGION Represents the region where an exception occurs. (RO)

LP_APM_M0_EXCEPTION_MODE Represents the master's security mode when an exception occurs. (RO)

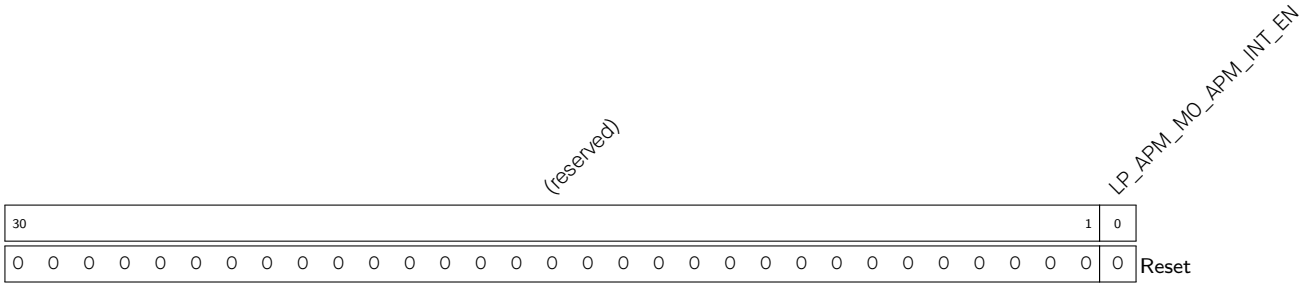
LP_APM_M0_EXCEPTION_ID Represents master ID when an exception occurs. (RO)

Register 15.33. LP_APM_M0_EXCEPTION_INFO1_REG (0x00D4)



LP_APM_M0_EXCEPTION_ADDR Represents the access address when an exception occurs. (RO)

Register 15.34. LP_APM_INT_EN_REG (0x00E8)



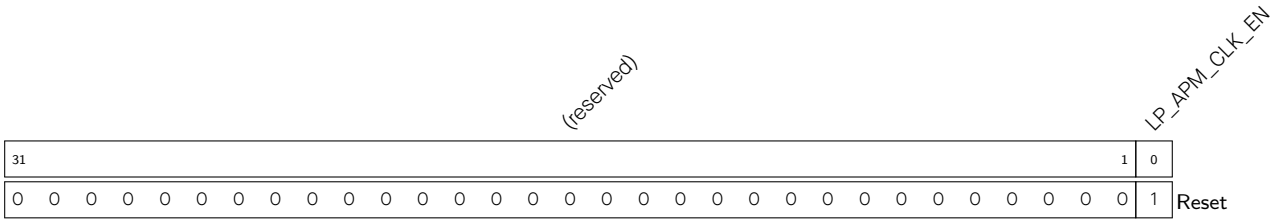
LP_APM_M0_APM_INT_EN Configures to enable LP_APM_CTRL M0 interrupt.

0: Disable

1: Enable

(R/W)

Register 15.35. LP_APM_CLOCK_GATE_REG (0x00EC)



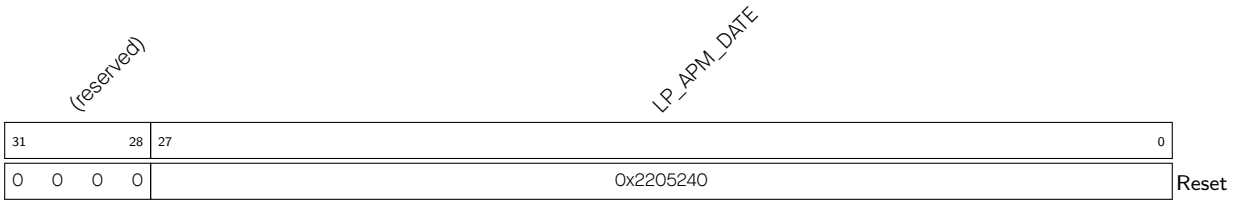
LP_APM_CLK_EN Configures whether to keep the clock always on.

0: Enable automatic clock gating

1: Keep the clock always on

(R/W)

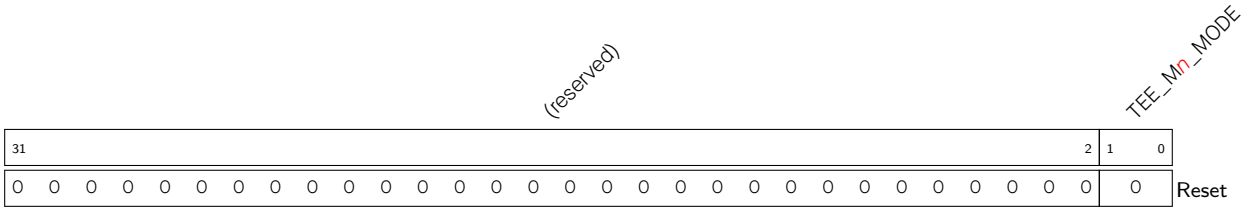
Register 15.36. LP_APM_DATE_REG (0x00FC)



LP_APM_DATE Version control register. (R/W)

15.8.3 TEE Registers of HP System

Register 15.37. 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

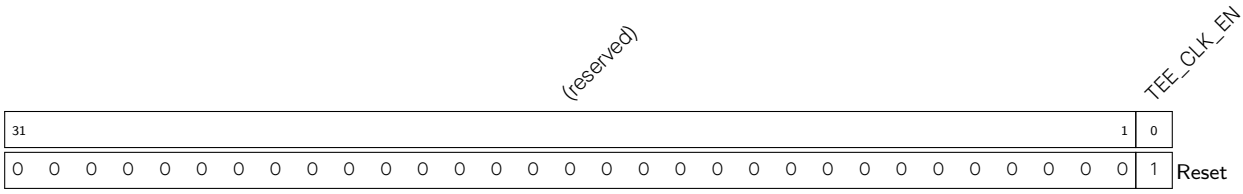
1: REE0

2: REE1

3: REE2

(R/W)

Register 15.38. TEE_CLOCK_GATE_REG (0x0080)



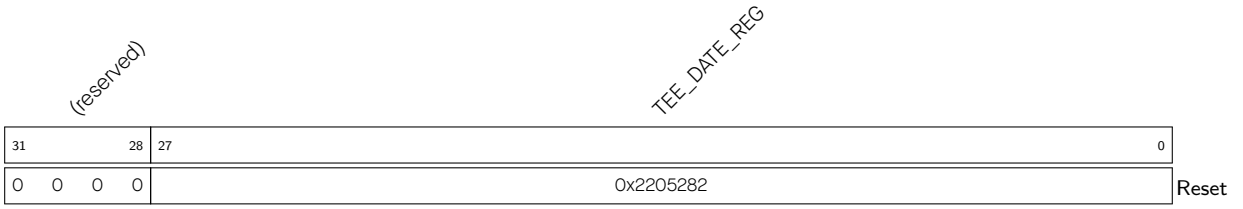
TEE_CLK_EN Configures whether to keep the clock always on.

0: Enable automatic clock gating

1: Keep the clock always on

(R/W)

Register 15.39. TEE_DATE_REG (0x0FFC)



TEE_DATE_REG Version control register (R/W)

Chapter 16

System Registers

16.1 Overview

ESP32-H2 supports the following auxiliary chip features:

- External Memory Encryption/Decryption
- Anti-DPA attack security
- Bus timeout protection

Each auxiliary chip feature can be controlled with dedicated system registers. This chapter describes how to configure these system registers.

16.2 Function Description

16.2.1 External Memory Encryption/Decryption Configuration

HP_SYSTEM_EXTERNAL_DEVICE_ENCRYPT_DECRYPT_CONTROL_REG configures encryption and decryption options of the external memory. For details about the External Memory Encryption and Decryption modules, please refer to Chapter *26 External Memory Encryption and Decryption (XTS_AES)*.

16.2.2 Anti-DPA Attack Security Control

ESP32-H2 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 process, 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-H2 provides three security levels for users to choose to adapt to different applications.

Table 16.2-1. Security Level

Security-Level Name	Value	96M_PLL_CLK (MHz)	64M_PLL_CLK (MHz)	XTAL_CLK (MHz)
SEC_DPA_OFF	0	96	64	32
SEC_DPA_LOW	1 or 2	(48,96][A]	(32,64][A]	(16,32][A]
SEC_DPA_HIGH	3	(32,96][A]	(21.3,64][A]	(10.6,32][A]

[A] (x,y] means the operating frequency is greater than x MHz, and equal to or less than y MHz.

By default, the field HP_SYSTEM_SEC_DPA_CFG_SEL in register HP_SYSTEM_SEC_DPA_CONF_REG is 0. In this case, the security-level is decided by the eFuse field EFUSE_SEC_DPA_LEVEL. If the field HP_SYSTEM_SEC_DPA_CFG_SEL is set to 1, the security-level is decided by HP_SYSTEM_SEC_DPA_CFG_LEVEL in register HP_SYSTEM_SEC_DPA_CONF_REG.

16.2.3 Bus Timeout Protection

The Bus Timeout Protection function can be enabled and the timeout threshold can be configured through the configuration register. When a transfer is initiated, the counter inside the Timeout Protection module will increase by one every clock cycle. When the accumulated value is less than the timeout threshold and the bus receives a response from the slave, the internal counter is cleared. When the accumulated value is greater than the timeout threshold, if the slave device has not responded to the transfer, the Timeout Protection module will force the bus return signal to be pulled high. At the same time, it will report the interrupt and record the abnormal access address and master ID.

16.2.3.1 CPU Peripheral Timeout Protection Register

HP_SYSTEM_CPU_PERI_TIMEOUT_CONF_REG is the timeout protection configuration register for accessing CPU peripheral registers. CPU peripherals refer to the peripherals or modules whose addresses are in the range of 0x600C_0000 ~ 0x600C_FFFF. For corresponding peripheral information, please refer to Subsection 4.3.5 *Modules/Peripherals Address Mapping* in Chapter 4 *System and Memory*.

When a timeout occurs, the CPU_PERI_TIMEOUT_INTR interrupt will be asserted.

- HP_SYSTEM_CPU_PERI_TIMEOUT_CONF_REG: Reserved.
- 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.

16.2.3.2 HP Peripheral Timeout Protection Register

HP_SYSTEM_HP_PERI_TIMEOUT_CONF_REG is the timeout protection configuration register for accessing HP peripheral registers. HP peripherals refer to the peripherals or modules whose addresses are in the range of 0x6000_0000 ~ 0x6009_FFFF. For corresponding peripheral information, please refer to Subsection 4.3.5 *Modules/Peripherals Address Mapping* in Chapter 4 *System and Memory*.

When a timeout occurs, the HP_PERI_TIMEOUT_INTR interrupt will be asserted.

- HP_SYSTEM_HP_PERI_TIMEOUT_CONF_REG: Reserved.
- 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.

16.2.3.3 LP Peripheral Timeout Protection Register

LP_PERI_BUS_TIMEOUT_CONF_REG is the timeout protection configuration register for accessing LP peripheral registers. LP peripherals refer to the peripherals or modules whose addresses are in the range of

0x600B_0000 ~ 0x600B_FFFF. For corresponding peripheral information, please refer to Subsection 4.3.5 *Modules/Peripherals Address Mapping* in Chapter 4 *System and Memory*.

When a timeout occurs, the LP_PERI_TIMEOUT_INTR interrupt will be asserted.

- LP_PERI_BUS_TIMEOUT_CONF_REG: Reserved.
- 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.

16.3 Register Summary

The addresses related of LP Peripheral timeout registers are relative to Low-power Peripheral base address provided in Table 4.3-2 in Chapter *4 System and Memory*, and the others addresses in this section are relative to System Registers base address provided in Table 4.3-2 in Chapter *4 System and Memory*.

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

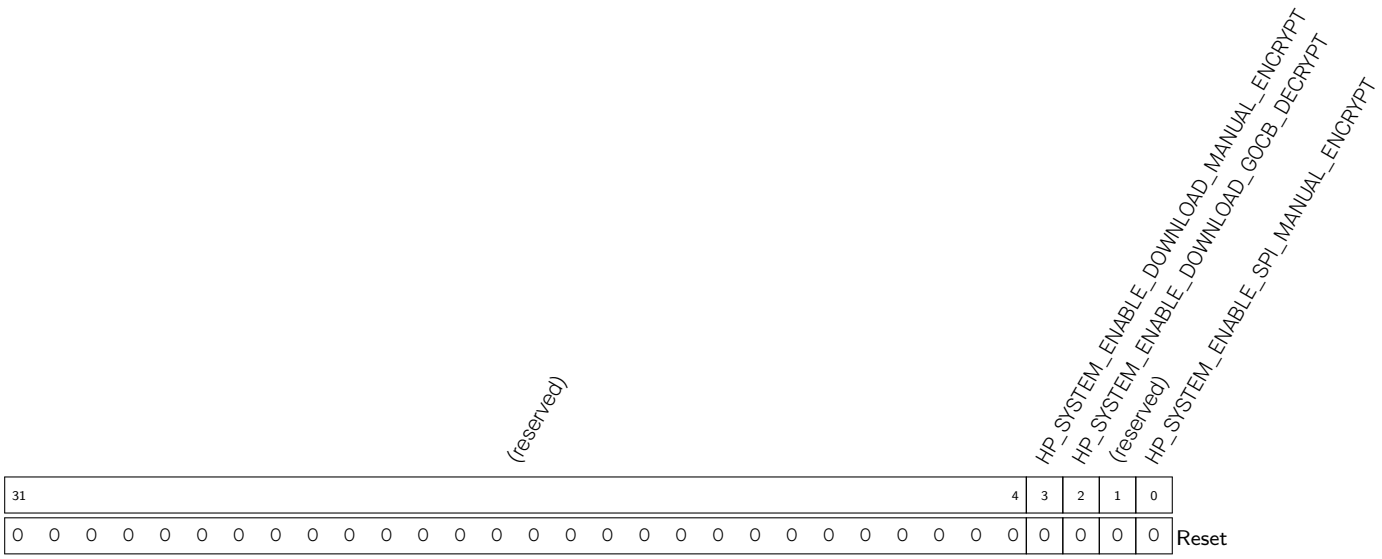
Name	Description	Address	Access
Configuration Register			
HP_SYSTEM_EXTERNAL_DEVICE_ENCRYPT_DECRYPT_CONTROL_REG	External device encryption/decryption configuration register	0x0000	R/W
HP_SYSTEM_SEC_DPA_CONF_REG	HP anti-DPA security configuration register	0x0008	R/W
CPU Peripheral Timeout Register			
HP_SYSTEM_CPU_PERI_TIMEOUT_CONF_REG	CPU Peripheral Timeout 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	HP Peripheral Timeout 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	LP Peripheral timeout configuration register	0x0010	varies
LP_PERI_BUS_TIMEOUT_ADDR_REG	LP Peripheral abnormal access address register	0x0014	RO
LP_PERI_BUS_TIMEOUT_UID_REG	LP Peripheral Master ID and permission register	0x0018	WTC
Version Register			
HP_SYSTEM_DATE_REG	Date control and version control register	0x03FC	R/W

GoBack

16.4 Registers

The addresses related of LP Peripheral timeout registers are relative to Low-power Peripheral base address provided in Table 4.3-2 in Chapter 4 *System and Memory*, and the others addresses in this section are relative to System Registers base address provided in Table 4.3-2 in Chapter 4 *System and Memory*.

Register 16.1. 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_GOCB_DECRYPT Configures whether to enable MSPI XTS auto decryption in download boot mode.

0: Disable

1: Enable

(R/W)

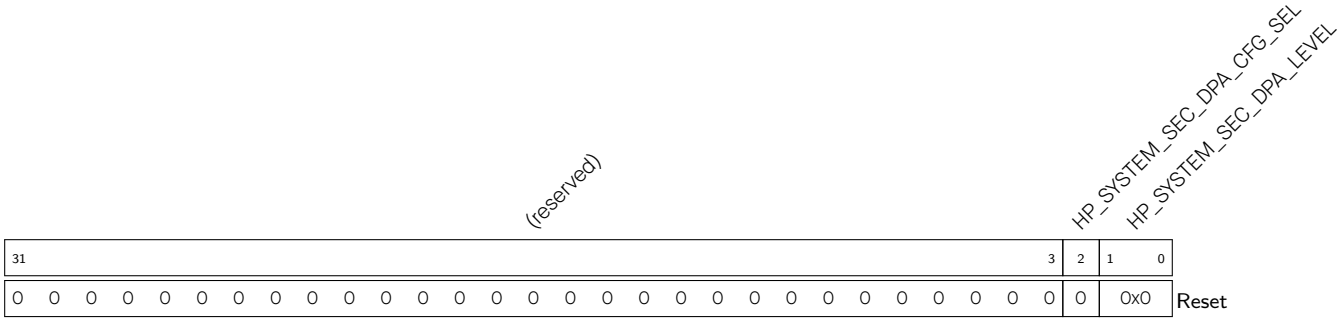
HP_SYSTEM_ENABLE_DOWNLOAD_MANUAL_ENCRYPT Configures whether to enable MSPI XTS manual encryption in download boot mode.

0: Disable

1: Enable

(R/W)

Register 16.2. 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 0.

0: Disable

1-3: Enable. The larger the number, the higher the security level, which represents the ability to resist DPA attacks, with an increased computational overhead of the hardware crypto-accelerators at the same time.

(R/W)

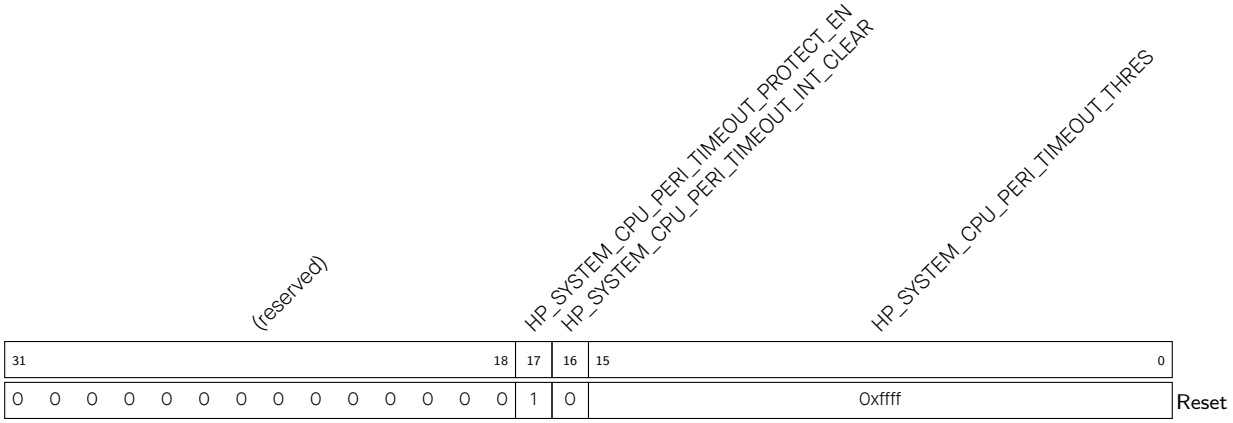
HP_SYSTEM_SEC_DPA_CFG_SEL Configures whether to select HP_SYSTEM_SEC_DPA_LEVEL or EFUSE_SEC_DPA_LEVEL (from eFuse) to control DPA level.

0: Select EFUSE_SEC_DPA_LEVEL

1: Select HP_SYSTEM_SEC_DPA_LEVEL

(R/W)

Register 16.3. 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 timeout interrupt. (WT)

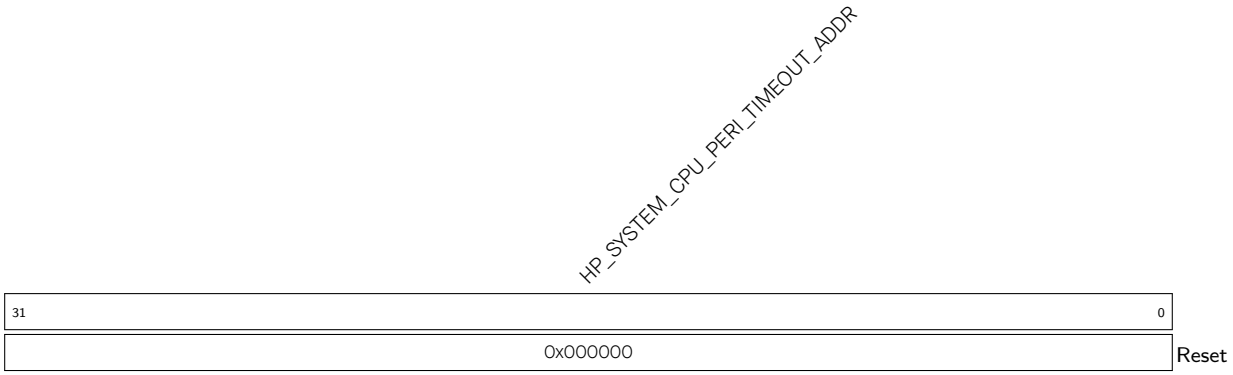
HP_SYSTEM_CPU_PERI_TIMEOUT_PROTECT_EN Configures whether to enable timeout protection for accessing CPU peripheral registers.

0: Disable

1: Enable

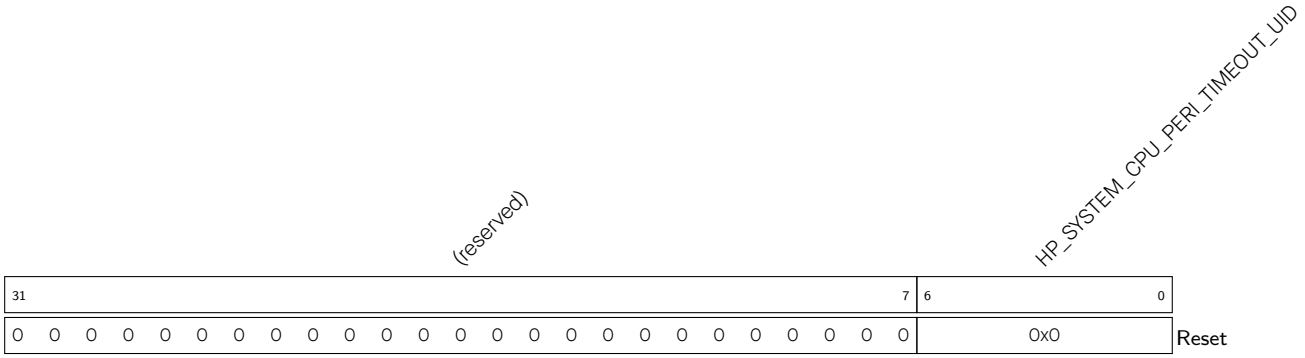
(R/W)

Register 16.4. HP_SYSTEM_CPU_PERI_TIMEOUT_ADDR_REG (0x0010)



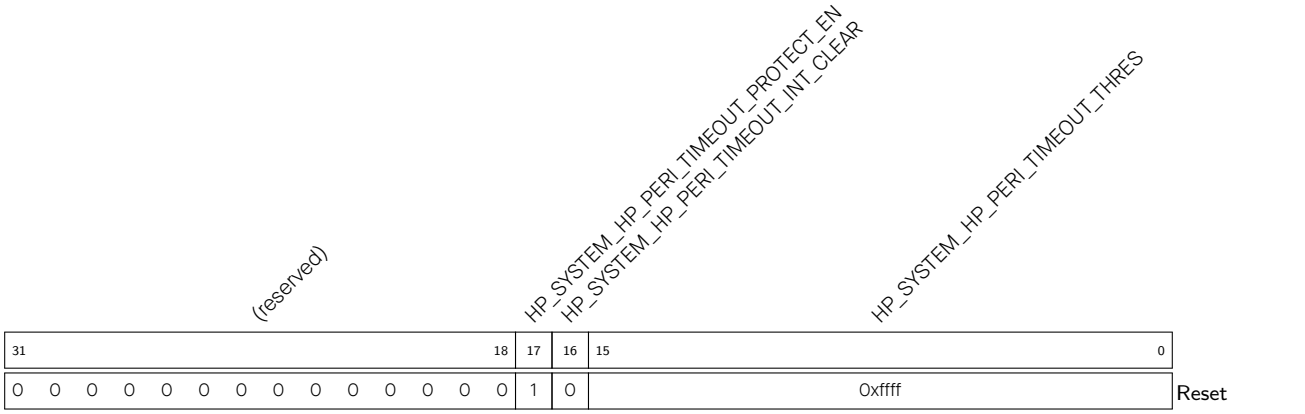
HP_SYSTEM_CPU_PERI_TIMEOUT_ADDR Represents the address information of abnormal access. (RO)

Register 16.5. 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. This register will be cleared after the interrupt is cleared. (WTC)

Register 16.6. 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 register, corresponding to the number of clock cycles of the clock domain. (R/W)

HP_SYSTEM_HP_PERI_TIMEOUT_INT_CLEAR Configures whether or not to clear timeout interrupt.

0: No effect

1: Clear timeout interrupt

(WT)

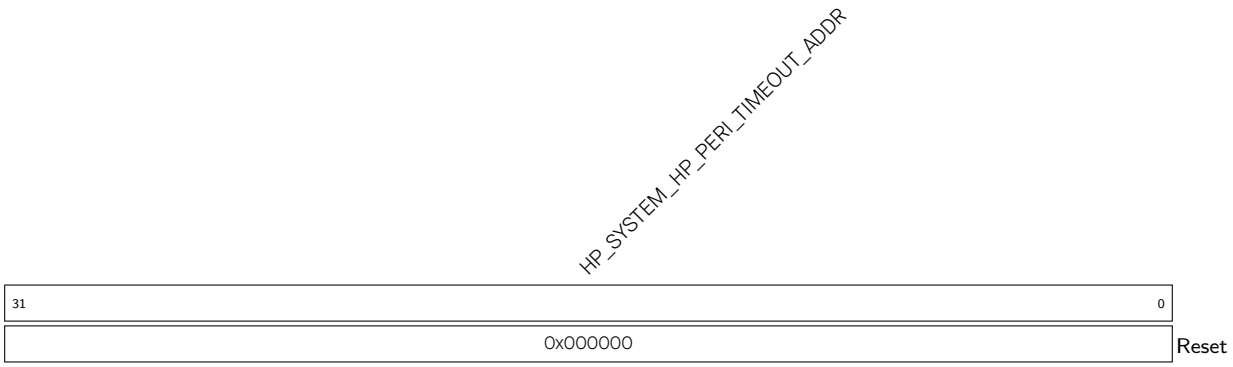
HP_SYSTEM_HP_PERI_TIMEOUT_PROTECT_EN Configures whether or not to enable timeout protection for accessing HP peripheral registers.

0: Disable

1: Enable

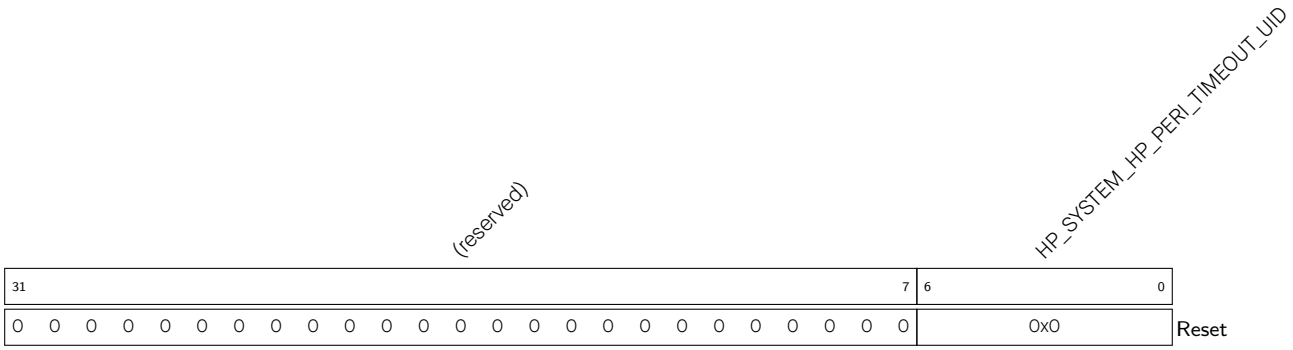
(R/W)

Register 16.7. HP_SYSTEM_HP_PERI_TIMEOUT_ADDR_REG (0x001C)



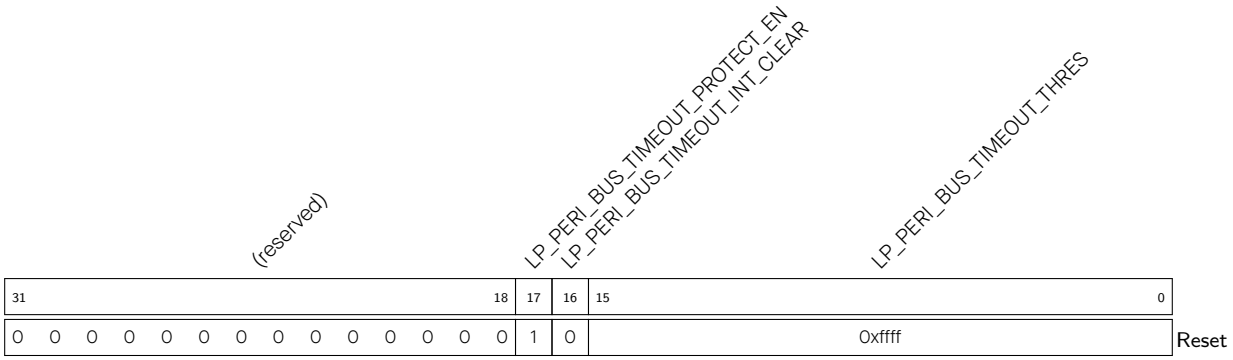
HP_SYSTEM_HP_PERI_TIMEOUT_ADDR Represents the address information of abnormal access. (RO)

Register 16.8. 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. This register will be cleared after the interrupt is cleared. (WTC)

Register 16.9. 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 Configures whether to clear timeout interrupt.

0: No effect

1: Clear timeout interrupt

(WT)

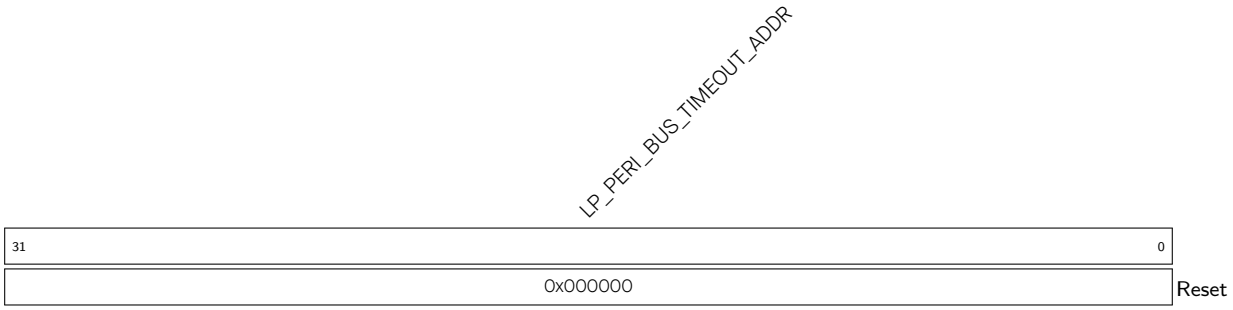
LP_PERI_BUS_TIMEOUT_PROTECT_EN Configures whether to enable timeout protection for accessing LP peripheral registers.

0: Disable

1: Enable

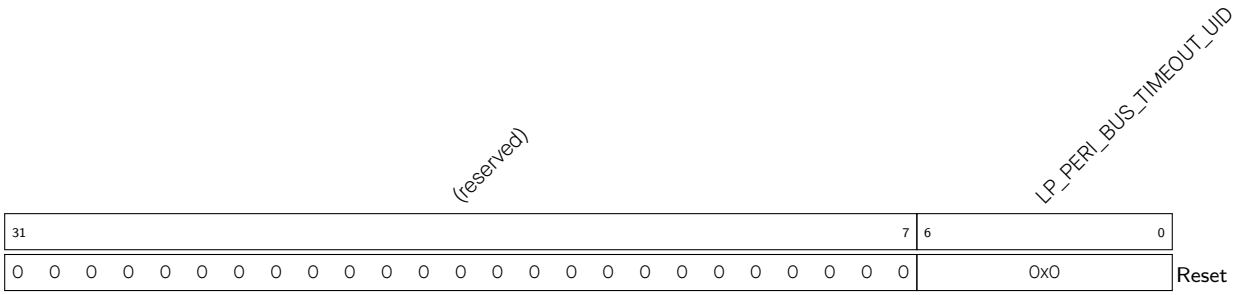
(R/W)

Register 16.10. LP_PERI_BUS_TIMEOUT_ADDR_REG (0x0014)



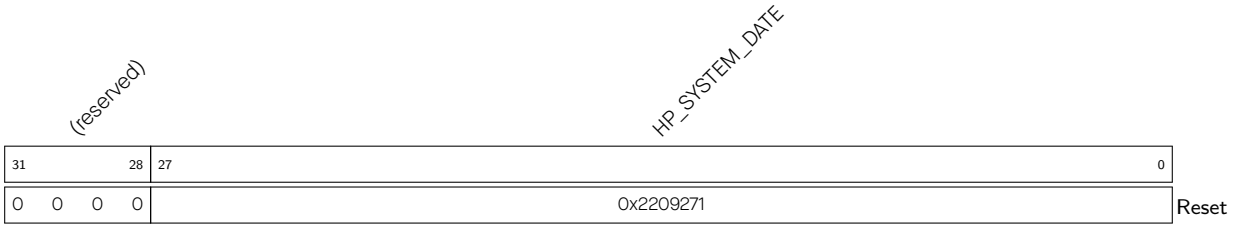
LP_PERI_BUS_TIMEOUT_ADDR Represents the address information of abnormal access. (RO)

Register 16.11. 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. This register will be cleared after the interrupt is cleared. (WTC)

Register 16.12. HP_SYSTEM_DATE_REG (0x03FC)



HP_SYSTEM_DATE Version control register. (R/W)

Chapter 17

Debug Assistant (ASSIST_DEBUG, MEM_MONITOR)

17.1 Overview

Debug Assistant is an auxiliary module that features a set of functions to help locate bugs and issues during software debugging.

17.2 Features

- **Read/write monitoring**: Monitors whether the CPU bus reads from or writes to a specified memory address space. A detected read or write in the monitored address space will trigger an interrupt.
- **Stack pointer (SP) monitoring**: Monitors whether the SP exceeds the specified address space. A bounds violation will trigger an interrupt.
- **Program counter (PC) logging**: Records PC value. The developer can get the last PC value at the most recent CPU reset.
- **Bus access logging**: Records the information about bus access. When the CPU or Direct Memory Access (DMA) controller writes a specified value, the Debug Assistant module will record the data type, address of this write operation, and additionally the PC value when the write is performed by the CPU, and push such information to the HP SRAM.

17.3 Functional Description

17.3.1 Region Read/Write Monitoring

The Debug Assistant module can monitor reads/writes performed by the CPU over data bus and peripheral bus in a certain address space, i.e., memory region. Whenever the bus reads or writes in the specified address space, an interrupt will be triggered. The data bus can monitor two memory regions (assuming they are region 0 and region 1, defined by developers' needs) at the same time, and so can Peripheral Bus.

17.3.2 SP Monitoring

The Debug Assistant module can monitor the SP so as to prevent stack overflow or erroneous push/pop. When the SP exceeds the minimum or maximum threshold, the module will record the PC pointer and generate an interrupt. The threshold is configured by software.

17.3.3 PC Logging

In some cases, software developers want to know the PC at the last CPU reset. For instance, when the program is stuck and the issue can be solved only by reset, the developer may want to know where the program got stuck in order to debug. The Debug Assistant module can record the PC at the last CPU reset, which can be then read for software debugging.

17.3.4 CPU/DMA Bus Access Logging

The Debug Assistant module can record the information about the CPU data bus's and DMA bus's write behaviors in real time. When a write operation occurs in or a specific value is written to a specified address space, the module will record the bus type, the address, PC (only when the write is performed by the CPU will PC be recorded), and other information, and then store the data in the HP SRAM in a certain format.

17.4 Recommended Operation

17.4.1 Region Read/Write Monitoring and SP Monitoring Configuration

The Debug Assistant module can monitor reads and writes performed by the CPU's data bus and peripheral bus. Two memory regions on each bus can be monitored at the same time. All the monitoring modes supported by the Debug Assistant module are listed below:

- Monitoring of the read/write operations performed by data bus
  - Data bus reads in region 0
  - Data bus writes in region 0
  - Data bus reads in region 1
  - Data bus writes in region 1
- Monitoring of the read/write operations performed by peripheral bus
  - Peripheral bus reads in region 0
  - Peripheral bus writes in region 0
  - Peripheral bus reads in region 1
  - Peripheral bus writes in region 1
- Monitoring of exceeding the SP monitored region
  - SP is greater than the ending address of the monitored region
  - SP is less than the starting address of the monitored region

The configuration process for region monitoring and SP monitoring is as follows:

1. Configure monitored region and SP threshold.
   - Configure data bus region 0 with ASSIST_DEBUG_CORE_0_AREA_DRAM0_0_MIN_REG and ASSIST_DEBUG_CORE_0_AREA_DRAM0_0_MAX_REG.

- Configure data bus region 1 with ASSIST_DEBUG_CORE_0_AREA_DRAM0_1_MIN_REG and ASSIST_DEBUG_CORE_0_AREA_DRAM0_1_MAX_REG.
- Configure peripheral bus region 0 with ASSIST_DEBUG_CORE_0_AREA_PIF_0_MIN_REG and ASSIST_DEBUG_CORE_0_AREA_PIF_0_MAX_REG.
- Configure peripheral bus region 1 with ASSIST_DEBUG_CORE_0_AREA_PIF_1_MIN_REG and ASSIST_DEBUG_CORE_0_AREA_PIF_1_MAX_REG.
- Configure SP monitored region with ASSIST_DEBUG_CORE_0_SP_MIN_REG and ASSIST_DEBUG_CORE_0_SP_MAX_REG.

2. Configure interrupts.
   - Configure ASSIST_DEBUG_CORE_0_INTR_ENA_REG to enable the interrupt of a monitoring mode.
   - Configure ASSIST_DEBUG_CORE_0_INTR_RAW_REG to get the interrupt status of a monitoring mode.
   - Configure ASSIST_DEBUG_CORE_0_INTR_CLR_REG to clear the interrupt of a monitoring mode.
3. Configure ASSIST_DEBUG_CORE_0_MONTR_ENA_REG to enable the monitoring mode(s). Various monitoring modes can be enabled at the same time.

Assuming that Debug Assistant module needs to monitor whether data bus writes to $[A–B]$ address space, you can enable monitoring in either data bus region 0 or region 1. The following configuration process is based on region 0:

1. Configure ASSIST_DEBUG_CORE_0_RCD_PDEBUGEN to 1 to enable CPU to update PC signals to the Debug Assistant module.
2. Configure ASSIST_DEBUG_CORE_0_AREA_DRAM0_0_MIN_REG to Address A.
3. Configure ASSIST_DEBUG_CORE_0_AREA_DRAM0_0_MAX_REG to Address B.
4. Configure ASSIST_DEBUG_CORE_0_INTR_ENA_REG bit[1] to enable the interrupt for write operations by data bus in region 0.
5. Configure ASSIST_DEBUG_CORE_0_MONTR_ENA_REG bit[1] to enable monitoring write operations by data bus in region 0.
6. Configure interrupt matrix to map ASSIST_DEBUG_INT into CPU interrupt (please refer to Chapter 9 *Interrupt Matrix (INTMTX)*).
7. After the interrupt is triggered:
   - Read ASSIST_DEBUG_CORE_0_INTR_RAW_REG to obtain which operation triggered the interrupt.
   - If the interrupt is triggered by region monitoring, read ASSIST_DEBUG_CORE_0_AREA_PC_REG for the PC value, and ASSIST_DEBUG_CORE_0_AREA_SP_REG for the SP.
   - If the interrupt is triggered by stack monitoring, read ASSIST_DEBUG_CORE_0_SP_PC_REG for the PC value.
   - Write 1 to the corresponding bits of ASSIST_DEBUG_CORE_0_INTR_RAW_REG to clear the interrupts.

17.4.2 PC Logging Configuration

Configure ASSIST_DEBUG_CORE_0_RCD_PDEBUGEN to 1 to enable CPU to update PC signals to the Debug Assistant module. If ASSIST_DEBUG_CORE_0_RCD_RECORDEN is also configured to 1, ASSIST_DEBUG_CORE_0_RCD_PDEBUGPC_REG will record the CPU's PC signal and ASSIST_DEBUG_CORE_0_RCD_PDEBUGSP_REG will record the SP value. Otherwise, the two registers keep the original values.

When the CPU resets, ASSIST_DEBUG_CORE_0_RCD_EN_REG will reset, while ASSIST_DEBUG_CORE_0_RCD_PDEBUGPC_REG and ASSIST_DEBUG_CORE_0_RCD_PDEBUGSP_REG will not. Therefore, the two registers will keep the PC value and SP value at the CPU reset.

17.4.3 CPU/DMA Bus Access Logging Configuration

The configuration process for CPU/DMA bus access logging is described below.

1. Configure the monitored address space.
   - Configure MEM_MONITOR_LOG_MIN_REG and MEM_MONITOR_LOG_MAX_REG to specify the monitored address space.
2. Configure the monitoring mode with MEM_MONITOR_LOG_MODE:
   - write monitoring (whether the bus has write operations)
   - word monitoring (whether the bus writes a specific word)
   - halfword monitoring (whether the bus writes a specific halfword)
   - byte monitoring (whether the bus writes a specific byte)
3. Configure the specific values to be monitored.
   - In word monitoring mode, 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 is used to mask the byte 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 configured to 0x01020304 and MEM_MONITOR_LOG_DATA_MASK_REG is configured to 0x1, then any writes of the data matching the 0x010203XX pattern by the bus will be recorded.
4. Configure where to store the recorded data.
   - MEM_MONITOR_LOG_MEM_START_REG and MEM_MONITOR_LOG_MEM_END_REG specify where to store the 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_START_REG to MEM_MONITOR_LOG_MEM_CURRENT_ADDR_REG.

- Enable the permission for the Debug Assistant module to access the internal HP SRAM. The permission is disabled by default. Only when it is enabled can the Debug Assistant module access the internal HP SRAM. For more information, please refer to Chapter 15 *Permission Control (PMS)*.

5. Configure the writing mode for the recorded data: loop mode or non-loop mode.

   - In Loop mode, writing to the specified address space is performed in loops. When writing reaches the end address, it will return to the starting address and continue, overwriting the previously recorded data. Set MEM_MONITOR_LOG_MEM_LOOP_ENABLE to enable Loop mode.
     For example, there are 10 write operations (1 – 10) to address space 0 – 4 during bus access. After the 5th operation writes to address 4, the 6th operation will start writing from address 0. The 6th to 10th operations will overwrite the previous data written by the 1st to 5th operations.
   - In Non-loop mode, when writing reaches the end address, it will stop at the end address and discard the remaining data, not overwriting the previously recorded data. Clear MEM_MONITOR_LOG_MEM_LOOP_ENABLE to use Non-loop mode.
     For example, there are 10 write operations (1 – 10) to address space 0 – 4 during bus access. After the 5th operation writes to address 4, all the subsequent data will be discarded.

6. Configure bus enable registers.

   - Enable CPU or DMA bus access logging with MEM_MONITOR_LOG_ENA. They can be enabled at the same time.

The Debug Assistant module first writes the recorded data to an internal buffer, and then fetches the data from the buffer and writes it to the configured memory space. When the monitored behaviors are triggered continuously, the generated recording packets may fully occupy the buffer, making it unable to take any incoming packets. At this time, the module discards these incoming packets and buffers a LOST packet instead to indicate some packets are discarded before the buffer reaches its capacity. However, the bus type and the number of these discarded packets are unknown.

When bus access logging is finished, the recorded data can be read from memory for decoding. The recorded data can be in one of these three packet formats, namely CPU packet (corresponding to CPU data bus), DMA packet (corresponding to DMA bus), and LOST packet. The packet formats are shown in Table 17.4-1, 17.4-2, and 17.4-3.

Table 17.4-1. CPU Packet Format

Bit[63:34]	Bit[33:32]	Bit[31:4]	Bit[3:2]	Bit[1:0]
pc_offset	anchored(2)	addr_offset	format	anchored(1)

Table 17.4-2. DMA Packet Format

Bit[31:9]	Bit[8:4]	Bit[3:2]	Bit[1:0]
addr_offset	dma_source	format	anchored(1)

Table 17.4-3. LOST Packet Format

Bit[31:4]	Bit[3:2]	Bit[1:0]
reserved	format	anchored(1)

It can be seen from the data packet formats that the HP CPU packet size is 64 bits, DMA packet size 32 bits, and LOST packet 32 bits. These packets contain the following fields:

- **format** – the packet type. 0: CPU packet; 1: DMA packet; 3: LOST packet.
- **pc_offset** - the offset of the PC register at the time of access. Actual PC = pc_offset + 0x4000_0000.
- **addr_offset** - the address offset of a write operation. Actual address = addr_offset + MEM_MONITOR_LOG_MIN_REG.
- **dma_source** - the source of DMA access. For more information, please refer to Chapter 15 *Permission Control (PMS)* > Table 15.4-1.
- **anchored** - the location of the 32 bits in the data packet. 1 indicates the lower 32 bits. 2 indicates the higher 32 bits.

The internal buffer of the module is 32 bits wide. When the CPU or DMA bus access logging are all enabled at the same time and the record data is generated at the same time, the DMA data packets are first buffered, and then the CPU packets. The Debug Assistant will automatically fetch the buffered data and store it in 32-bit data width into the specified memory space.

In Loop mode, data looping several times in the configured storage address space may cause residual data, which can interfere with packet parsing. For example, the lower 32 bits of a CPU packet are overwritten, thus making its higher 32 bits residual data. Therefore, users need to filter out the possible residual data in order to determine the starting position of the first valid packet with MEM_MONITOR_LOG_MEM_CURRENT_ADDR_REG. Once the starting position of the packet is identified, check the anchored bit value of the packet. If it is 1, the data will be retained. If it is 2, it will be discarded.

The process of packet parsing is described below:

1. Determine whether there is a data overflow with MEM_MONITOR_LOG_MEM_FULL_FLAG:
    - 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.
2. Read and parse data from the starting address. Read 32 bits each time.

After packet parsing is completed, clear the MEM_MONITOR_LOG_MEM_FULL_FLAG flag bit by setting MEM_MONITOR_CLR_LOG_MEM_FULL_FLAG.

17.5 Register Summary

The addresses of **bus logging configuration registers** (see 17.5.1) in this section are relative to the **MEM_MONITOR** base address. The addresses of other registers (see 17.5.2) are relative to the **ASSIST_DEBUG**

base address. Both base addresses are provided in Table 4.3-2 in Chapter 4 *System and Memory*.

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

17.5.1 Summary of Bus Logging Configuration Registers

Name	Description	Address	Access
Bus access logging configuration registers			
MEM_MONITOR_LOG_SETTING_REG	Bus access logging configuration register	0x0000	R/W
MEM_MONITOR_LOG_CHECK_DATA_REG	Configures monitored data in Bus access logging	0x0004	R/W
MEM_MONITOR_LOG_DATA_MASK_REG	Configures masked data in Bus access logging	0x0008	R/W
MEM_MONITOR_LOG_MIN_REG	Configures monitored address space in Bus access logging	0x000C	R/W
MEM_MONITOR_LOG_MAX_REG	Configures monitored address space in Bus access logging	0x0010	R/W
MEM_MONITOR_LOG_MEM_START_REG	Configures the starting address of the storage space for recorded data	0x0014	R/W
MEM_MONITOR_LOG_MEM_END_REG	Configures the end address of the storage space for recorded data	0x0018	R/W
MEM_MONITOR_LOG_MEM_CURRENT_ADDR_REG	Represents the address for the next write	0x001C	RO
MEM_MONITOR_LOG_MEM_ADDR_UPDATE_REG	Updates the address for the next write with the starting address for the recorded data	0x0020	R/W
MEM_MONITOR_LOG_MEM_FULL_FLAG_REG	Logging overflow status register	0x0024	varies
Clock control register			
MEM_MONITOR_CLOCK_GATE_REG	Register clock control	0x0028	R/W
Version control register			
MEM_MONITOR_DATE_REG	Version control register	0x03FC	R/W

17.5.2 Summary of Other Registers

Name	Description	Address	Access
Monitor configuration registers			
ASSIST_DEBUG_CORE_0_MONTR_ENA_REG	Monitoring enable register	0x0000	R/W
ASSIST_DEBUG_CORE_0_AREA_DRAM0_0_MIN_REG	Configures the starting address of region 0 monitored on data bus	0x0010	R/W

Name	Description	Address	Access
ASSIST_DEBUG_CORE_0_AREA_DRAM0_0_MAX_REG	Configures the ending address of region 0 monitored on data bus	0x0014	R/W
ASSIST_DEBUG_CORE_0_AREA_DRAM0_1_MIN_REG	Configures the starting address of region 1 monitored on data bus	0x0018	R/W
ASSIST_DEBUG_CORE_0_AREA_DRAM0_1_MAX_REG	Configures the ending address of region 1 monitored on data bus	0x001C	R/W
ASSIST_DEBUG_CORE_0_AREA_PIF_0_MIN_REG	Configures the starting address of region 0 monitored on peripheral bus	0x0020	R/W
ASSIST_DEBUG_CORE_0_AREA_PIF_0_MAX_REG	Configures the ending address of region 0 monitored on peripheral bus	0x0024	R/W
ASSIST_DEBUG_CORE_0_AREA_PIF_1_MIN_REG	Configures the starting address of region 1 monitored on peripheral bus	0x0028	R/W
ASSIST_DEBUG_CORE_0_AREA_PIF_1_MAX_REG	Configures the ending address of region 1 monitored on peripheral bus	0x002C	R/W
ASSIST_DEBUG_CORE_0_AREA_PC_REG	Region monitoring CPU PC status register	0x0030	RO
ASSIST_DEBUG_CORE_0_AREA_SP_REG	Region monitoring CPU SP status register	0x0034	RO
ASSIST_DEBUG_CORE_0_SP_MIN_REG	Configures the starting address of stack monitored region	0x0038	R/W
ASSIST_DEBUG_CORE_0_SP_MAX_REG	Configures the ending address of stack monitored region	0x003C	R/W
ASSIST_DEBUG_CORE_0_SP_PC_REG	Stack monitoring CPU PC status register	0x0040	RO
Interrupt configuration registers			
ASSIST_DEBUG_CORE_0_INTR_RAW_REG	Interrupt status register	0x0004	RO
ASSIST_DEBUG_CORE_0_INTR_ENA_REG	Interrupt enable register	0x0008	R/W
ASSIST_DEBUG_CORE_0_INTR_CLR_REG	Interrupt clear register	0x000C	R/W
PC logging configuration register			
ASSIST_DEBUG_CORE_0_RCD_EN_REG	CPU PC logging enable register	0x0044	R/W
PC logging status registers			
ASSIST_DEBUG_CORE_0_RCD_PDEBUGPC_REG	PC logging register	0x0048	RO
ASSIST_DEBUG_CORE_0_RCD_PDEBUGSP_REG	PC logging register	0x004C	RO
CPU status registers			

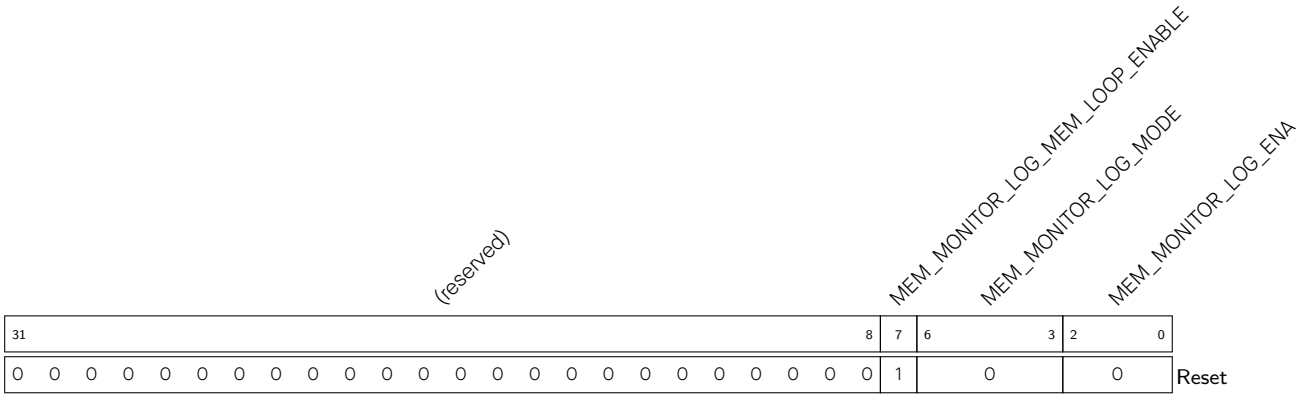
Name	Description	Address	Access
ASSIST_DEBUG_CORE_0_LASTPC_BEFORE_EXCEPTION_REG	PC of the last command before CPU enters exception	0x0070	RO
ASSIST_DEBUG_CORE_0_DEBUG_MODE_REG	CPU debug mode status register	0x0074	RO
Clock control register			
ASSIST_DEBUG_CLOCK_GATE_REG	Register clock control	0x0078	R/W
Version register			
ASSIST_DEBUG_DATE_REG	Version control register	0x03FC	R/W

17.6 Registers

The addresses of **bus logging configuration registers** (see 17.6.1) in this section are relative to **MEM_MONITOR** base address. The addresses of other registers (see 17.6.2) are relative to the **ASSIST_DEBUG** base address. Both base addresses are provided in Table 4.3-2 in Chapter 4 *System and Memory*.

17.6.1 Bus Logging Configuration Registers

Register 17.1. MEM_MONITOR_LOG_SETTING_REG (0x0000)



MEM_MONITOR_LOG_ENA Configures whether to enable CPU or DMA bus access logging.

bit[0]: Configures whether to enable CPU bus access logging.

0: Disable

1: Enable

bit[1]: Reserved

bit[2]: Configures whether to enable DMA bus access logging.

0: Disable

1: Enable

(R/W)

MEM_MONITOR_LOG_MODE Configures monitoring modes.

bit[0]: Configures write monitoring.

0: Disable

1: Enable

bit[1]: Configures word monitoring.

0: Disable

1: Enable

bit[2]: Configures halfword monitoring.

0: Disable

1: Enable

bit[3]: Configures byte monitoring.

0: Disable

1: Enable

(R/W)

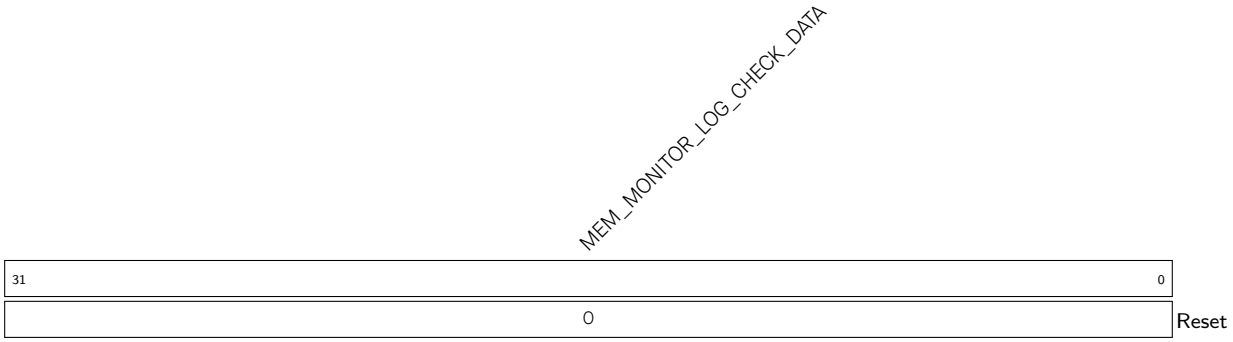
MEM_MONITOR_LOG_MEM_LOOP_ENABLE Configures the writing mode for the recorded data.

1: Loop mode

0: Non-loop mode

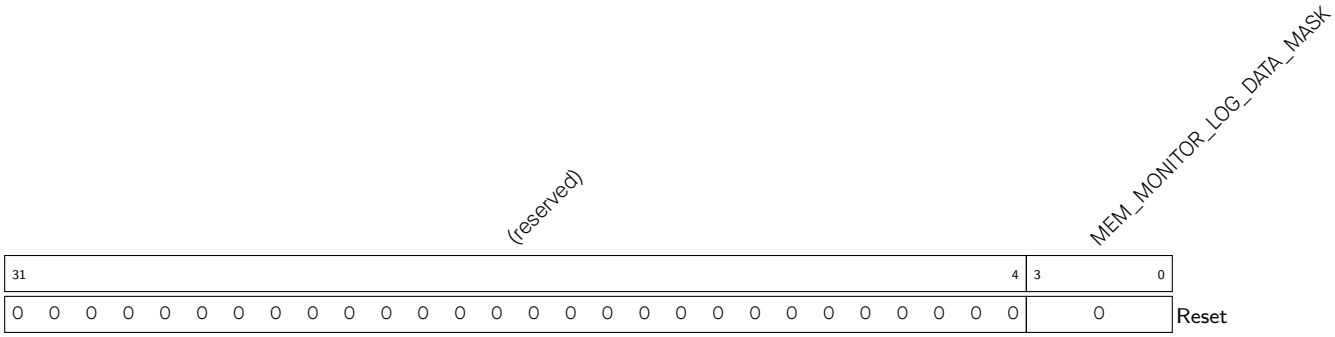
(R/W)

Register 17.2. MEM_MONITOR_LOG_CHECK_DATA_REG (0x0004)



MEM_MONITOR_LOG_CHECK_DATA Configures the data to be monitored during bus accessing. (R/W)

Register 17.3. MEM_MONITOR_LOG_DATA_MASK_REG (0x0008)



MEM_MONITOR_LOG_DATA_MASK Configures which byte(s) in MEM_MONITOR_LOG_CHECK_DATA_REG to mask.

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

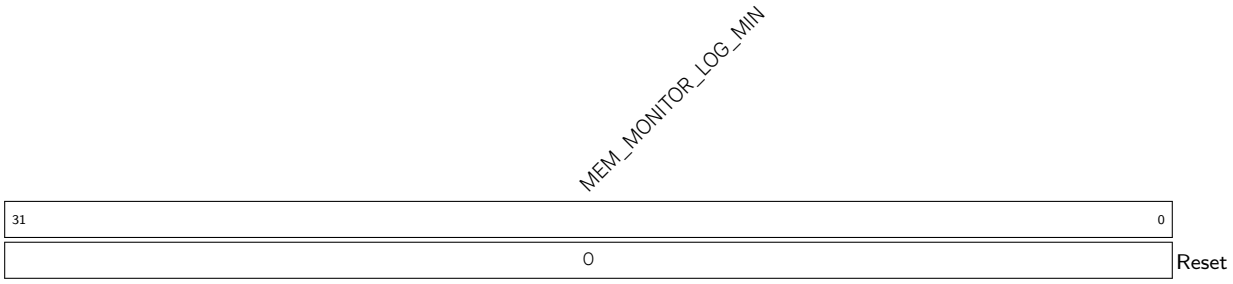
bit[3]: Configures whether to mask the most significant byte of MEM_MONITOR_LOG_CHECK_DATA_REG.

0: Not mask

1: Mask

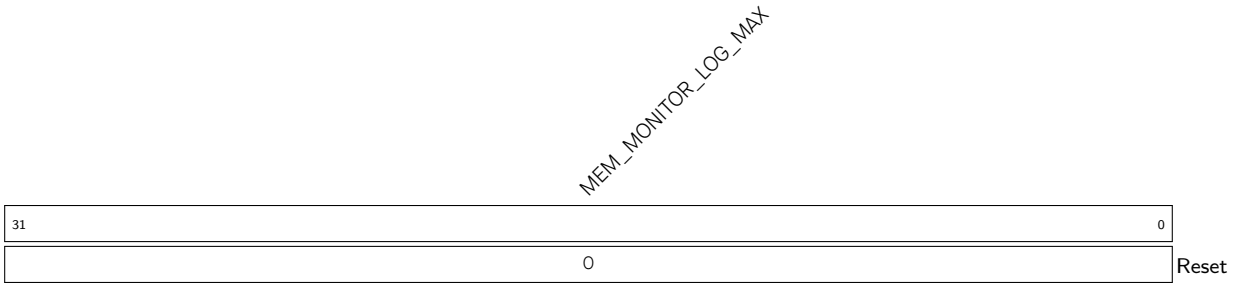
(R/W)

Register 17.4. MEM_MONITOR_LOG_MIN_REG (0x000C)



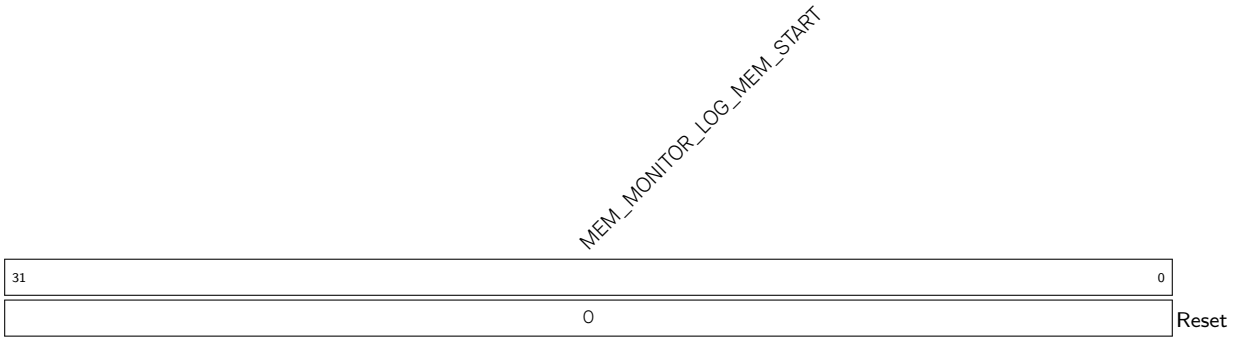
MEM_MONITOR_LOG_MIN Configures the starting address of the monitored address space. (R/W)

Register 17.5. MEM_MONITOR_LOG_MAX_REG (0x0010)



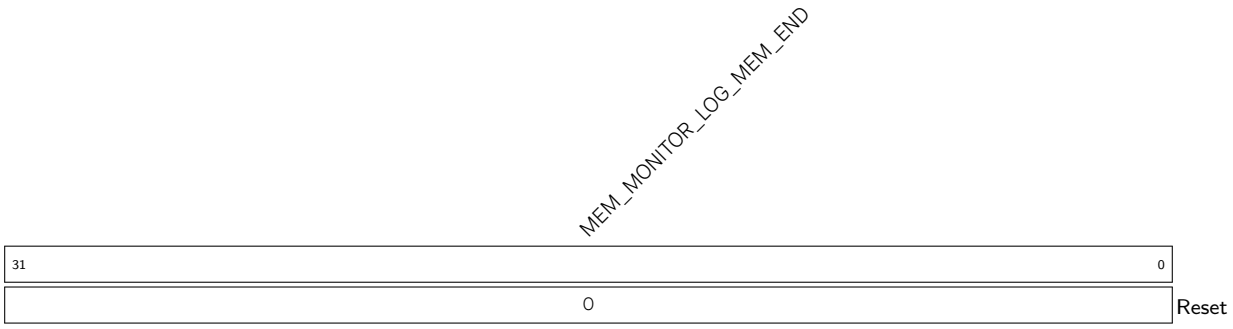
MEM_MONITOR_LOG_MAX Configures the ending address of the monitored address space. (R/W)

Register 17.6. MEM_MONITOR_LOG_MEM_START_REG (0x0014)



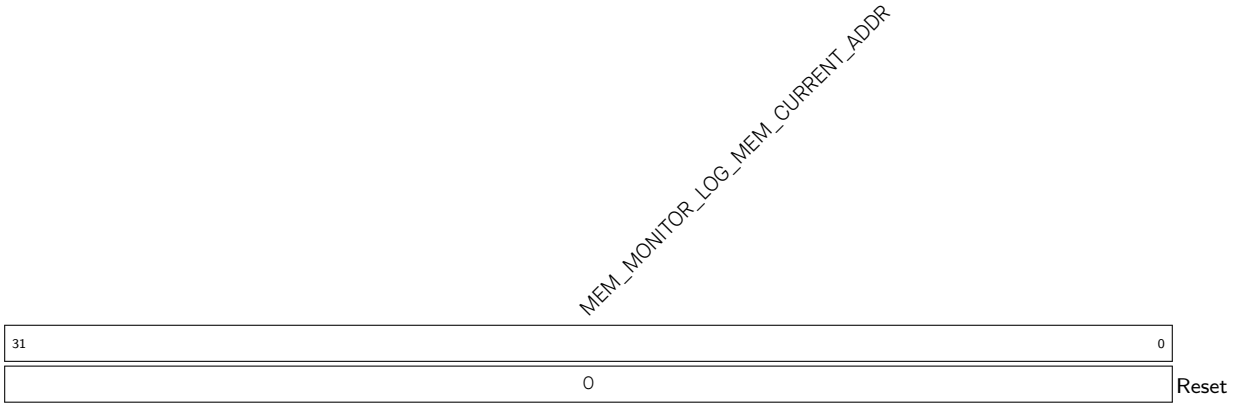
MEM_MONITOR_LOG_MEM_START Configures the starting address of the storage space for the recorded data. (R/W)

Register 17.7. MEM_MONITOR_LOG_MEM_END_REG (0x0018)



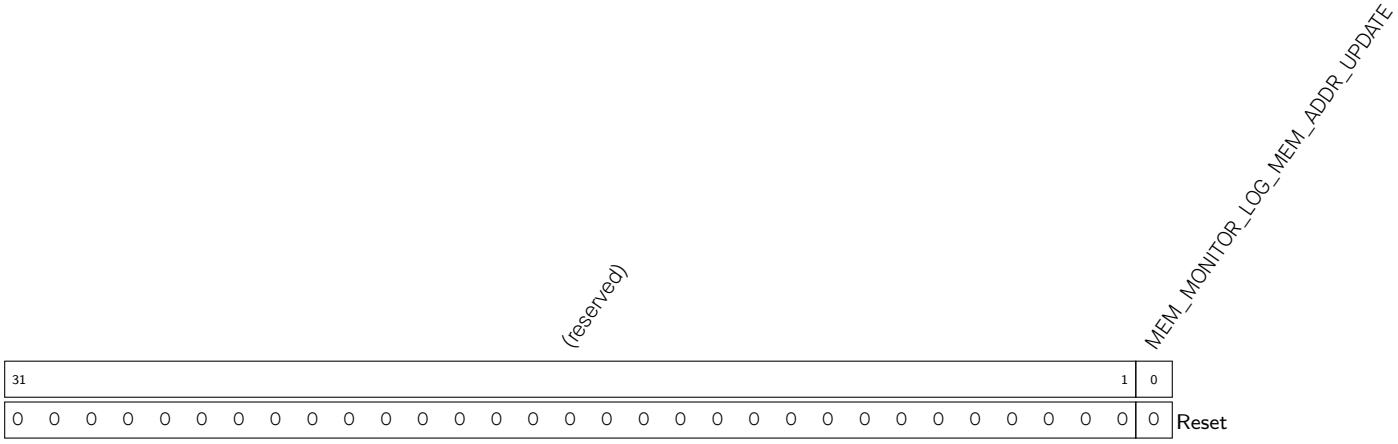
MEM_MONITOR_LOG_MEM_END Configures the ending address of the storage space for the recorded data. (R/W)

Register 17.8. MEM_MONITOR_LOG_MEM_CURRENT_ADDR_REG (0x001C)



MEM_MONITOR_LOG_MEM_CURRENT_ADDR Represents the address of the next write. (RO)

Register 17.9. MEM_MONITOR_LOG_MEM_ADDR_UPDATE_REG (0x0020)



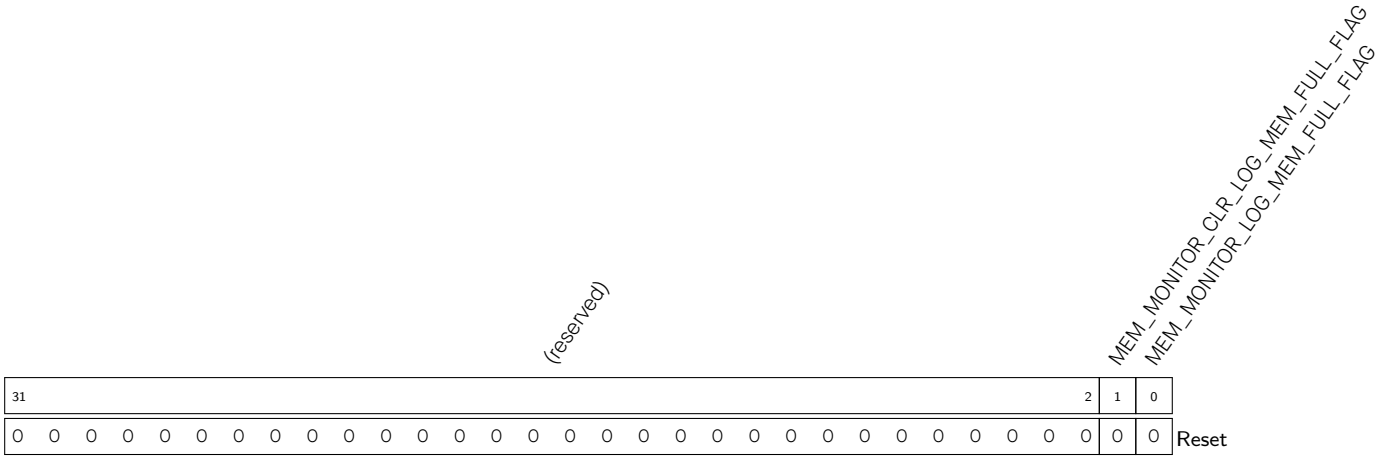
MEM_MONITOR_LOG_MEM_ADDR_UPDATE Configures whether to update the value in MEM_MONITOR_LOG_MEM_START_REG to MEM_MONITOR_LOG_MEM_CURRENT_ADDR_REG.

1: Update

0: Not update (default)

(R/W)

Register 17.10. MEM_MONITOR_LOG_MEM_FULL_FLAG_REG (0x0024)



MEM_MONITOR_LOG_MEM_FULL_FLAG Represents whether data overflows the storage space

0: Not Overflow

1: Overflow

(RO)

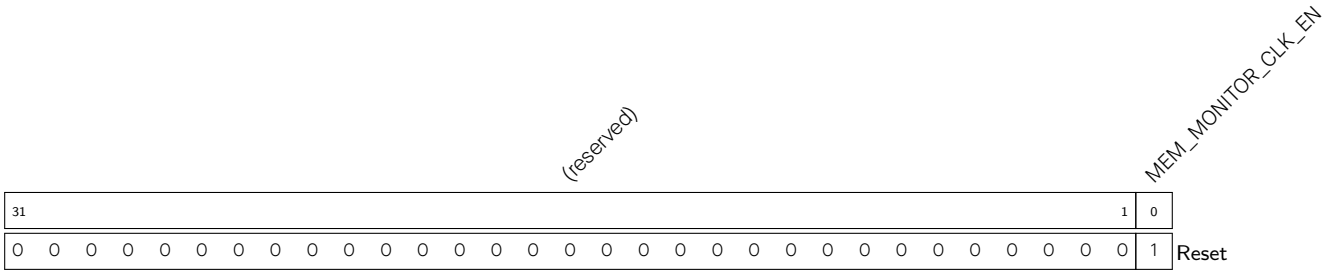
MEM_MONITOR_CLR_LOG_MEM_FULL_FLAG Configures whether to clear the MEM_MONITOR_LOG_MEM_FULL_FLAG flag bit.

0: Not clear

1: Clear

(R/W)

Register 17.11. MEM_MONITOR_CLOCK_GATE_REG (0x0028)



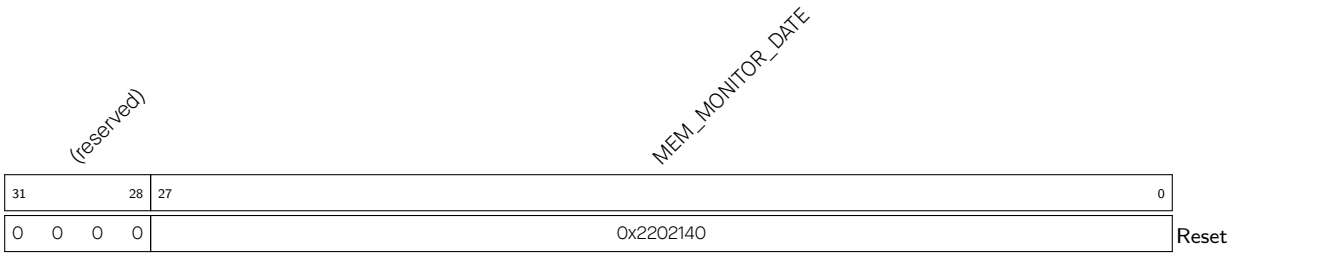
MEM_MONITOR_CLK_EN Configures whether to enable the register clock gating.

0: Disable

1: Enable

(R/W)

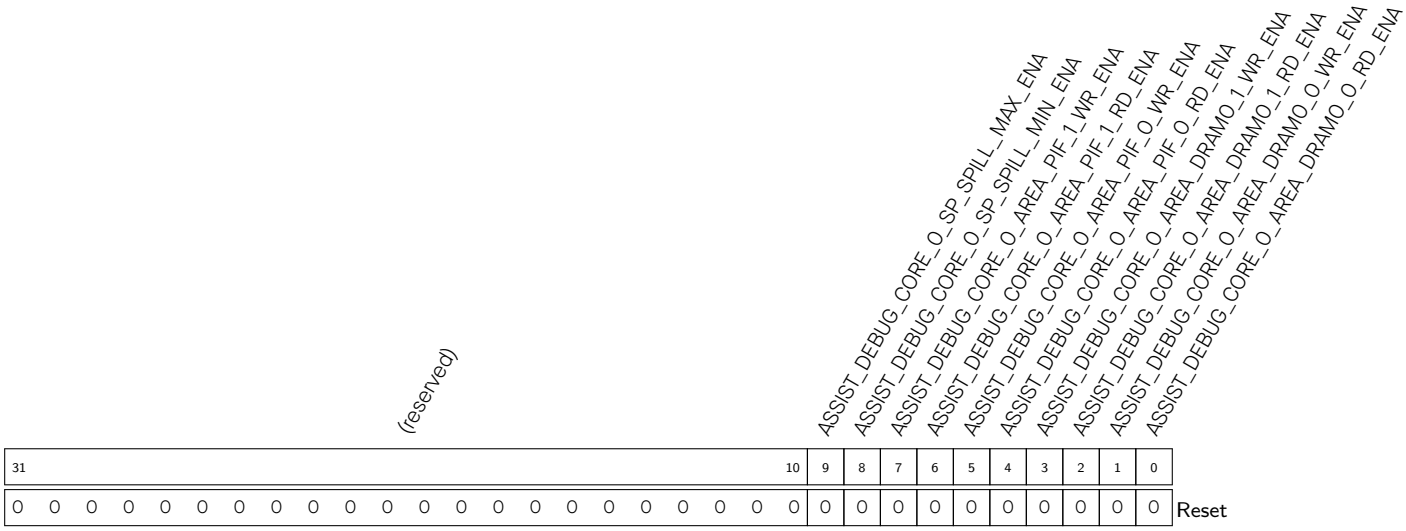
Register 17.12. MEM_MONITOR_DATE_REG (0x03FC)



MEM_MONITOR_DATE Version control register. (R/W)

17.6.2 Other Registers

Register 17.13. ASSIST_DEBUG_CORE_0_MONTR_ENA_REG (0x0000)



ASSIST_DEBUG_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)

ASSIST_DEBUG_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)

ASSIST_DEBUG_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)

ASSIST_DEBUG_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)

ASSIST_DEBUG_CORE_0_AREA_PIF_0_RD_ENA Configures whether to monitor read operations in region 0 by the peripheral bus.

0: Not Monitor

1: Monitor

(R/W)

Continued on the next page...

Register 17.13. ASSIST_DEBUG_CORE_0_MONTR_ENA_REG (0x0000)

Continued from the previous page...

ASSIST_DEBUG_CORE_0_AREA_PIF_0_WR_ENA Configures whether to monitor write operations in region 0 by the peripheral bus.
0: Not Monitor
1: Monitor
(R/W)

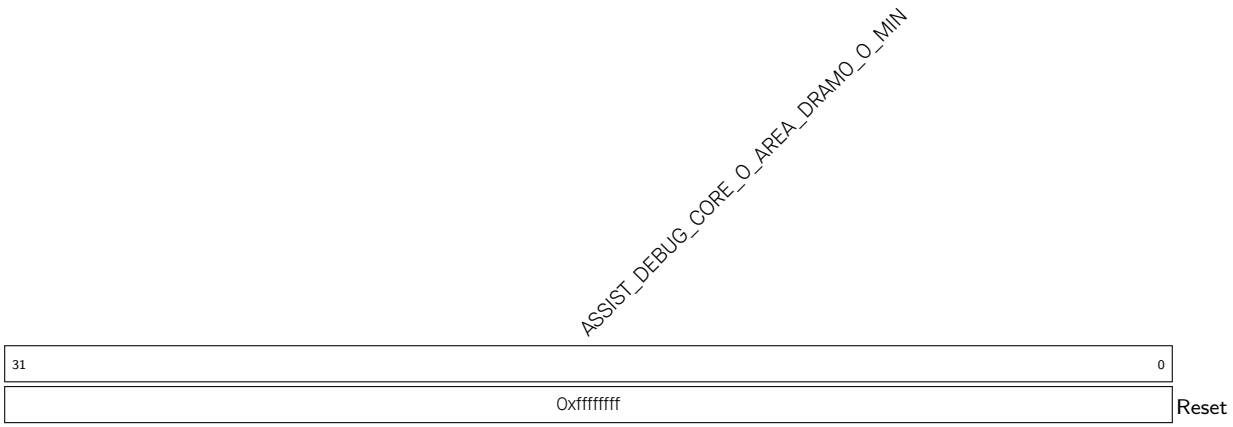
ASSIST_DEBUG_CORE_0_AREA_PIF_1_RD_ENA Configures whether to monitor read operations in region 1 by the peripheral bus.
0: Not Monitor
1: Monitor
(R/W)

ASSIST_DEBUG_CORE_0_AREA_PIF_1_WR_ENA Configures whether to monitor write operations in region 1 by the peripheral bus.
0: Not Monitor
1: Monitor
(R/W)

ASSIST_DEBUG_CORE_0_SP_SPILL_MIN_ENA Configures whether to monitor SP less than the starting address of SP monitored region.
0: Not Monitor
1: Monitor
(R/W)

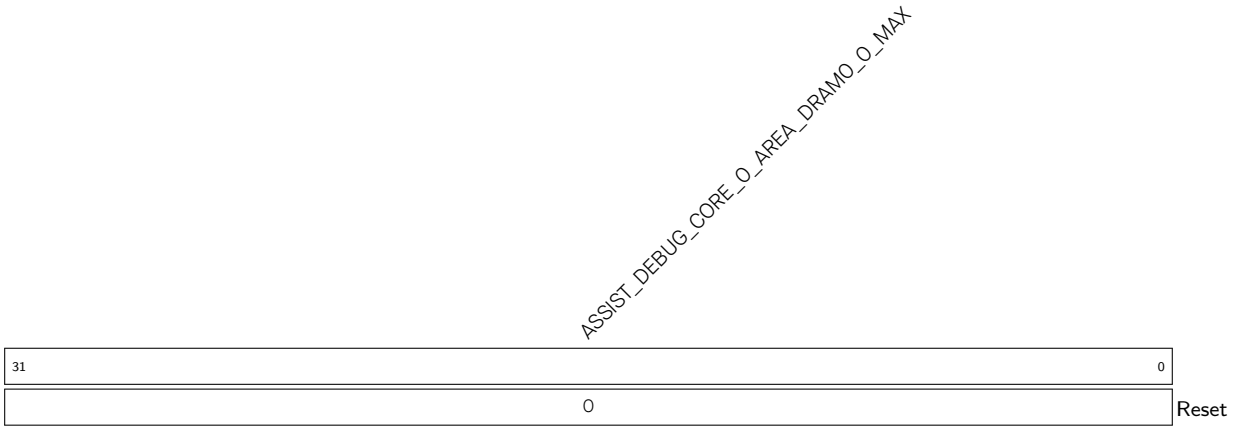
ASSIST_DEBUG_CORE_0_SP_SPILL_MAX_ENA Configures whether to monitor SP greater than the ending address of SP monitored region.
0: Not Monitor
1: Monitor
(R/W)

Register 17.14. ASSIST_DEBUG_CORE_0_AREA_DRAM0_0_MIN_REG (0x0010)



ASSIST_DEBUG_CORE_0_AREA_DRAM0_0_MIN Configures the starting address of data bus region 0. (R/W)

Register 17.15. ASSIST_DEBUG_CORE_0_AREA_DRAM0_0_MAX_REG (0x0014)

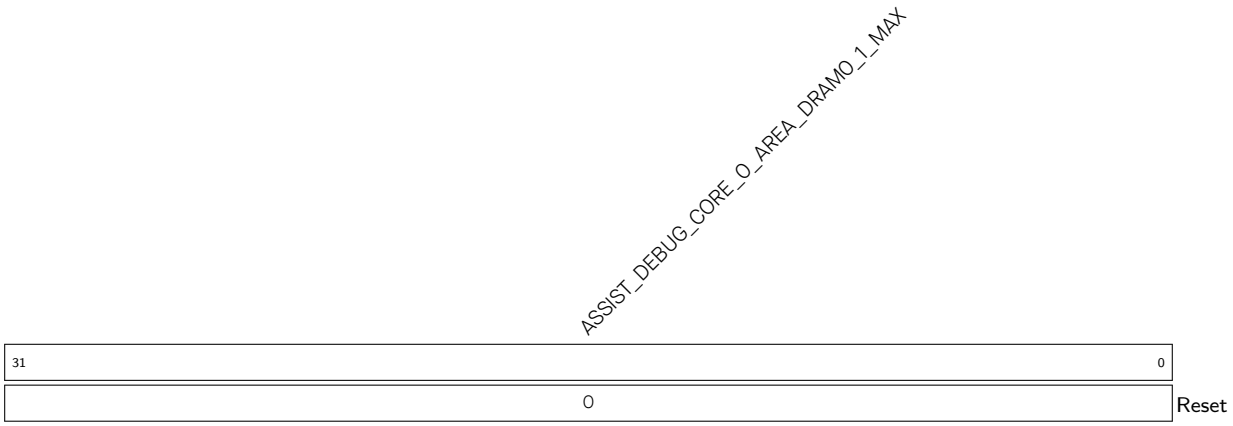


ASSIST_DEBUG_CORE_0_AREA_DRAM0_0_MAX Configures the ending address of data bus region 0. (R/W)

Register 17.16. ASSIST_DEBUG_CORE_0_AREA_DRAM0_1_MIN_REG (0x0018)

ASSIST_DEBUG_CORE_0_AREA_DRAM0_1_MIN Configures the starting address of data bus region 1. (R/W)

Register 17.17. ASSIST_DEBUG_CORE_0_AREA_DRAM0_1_MAX_REG (0x001C)



ASSIST_DEBUG_CORE_0_AREA_DRAM0_1_MAX Configures the ending address of data bus region 1. (R/W)

Register 17.18. ASSIST_DEBUG_CORE_0_AREA_PIF_0_MIN_REG (0x0020)

ASSIST_DEBUG_CORE_0_AREA_PIF_0_MIN Configures the starting address of peripheral bus region 0. (R/W)

Register 17.19. ASSIST_DEBUG_CORE_0_AREA_PIF_0_MAX_REG (0x0024)

ASSIST_DEBUG_CORE_0_AREA_PIF_0_MAX Configures the ending address of peripheral bus region 0. (R/W)

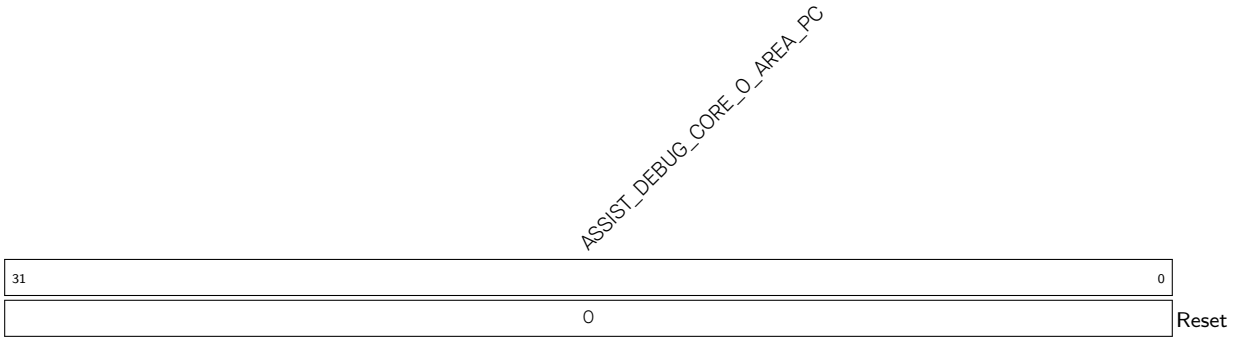
Register 17.20. ASSIST_DEBUG_CORE_0_AREA_PIF_1_MIN_REG (0x0028)

ASSIST_DEBUG_CORE_0_AREA_PIF_1_MIN Configures the starting address of peripheral bus region 1. (R/W)

Register 17.21. ASSIST_DEBUG_CORE_0_AREA_PIF_1_MAX_REG (0x002C)

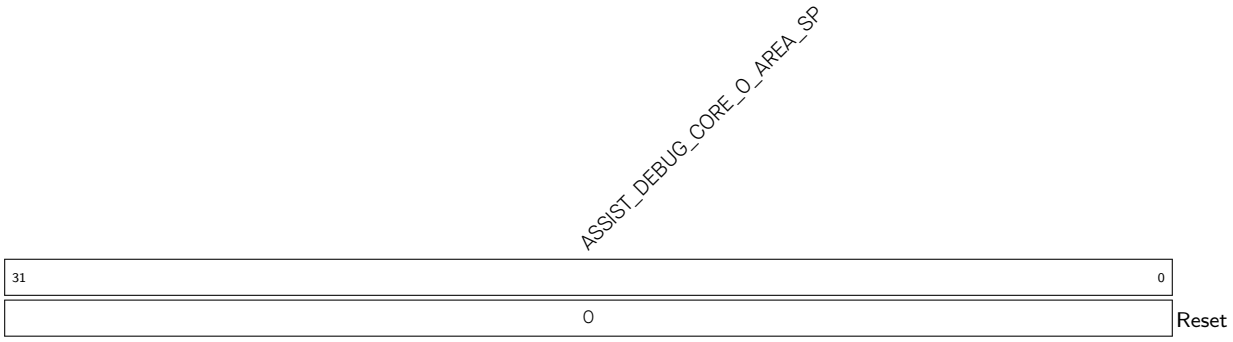
ASSIST_DEBUG_CORE_0_AREA_PIF_1_MAX Configures the ending address of peripheral bus region 1. (R/W)

Register 17.22. ASSIST_DEBUG_CORE_0_AREA_PC_REG (0x0030)



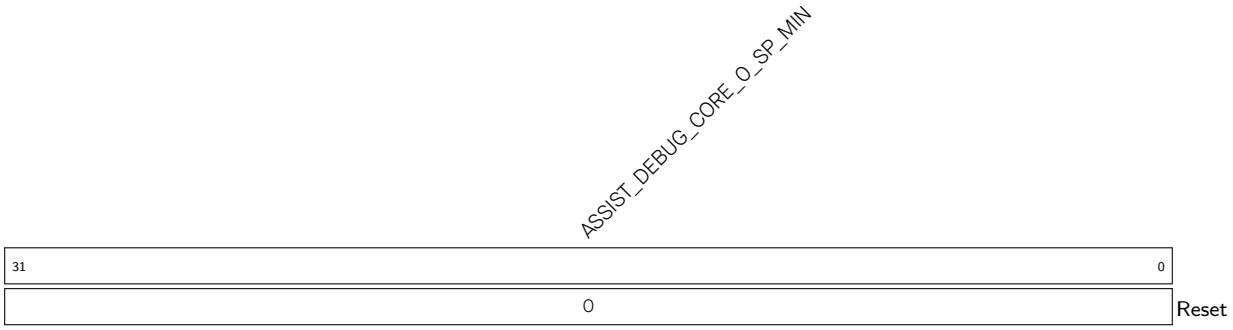
ASSIST_DEBUG_CORE_0_AREA_PC Represents the PC value when an interrupt is triggered during region monitoring. (RO)

Register 17.23. ASSIST_DEBUG_CORE_0_AREA_SP_REG (0x0034)



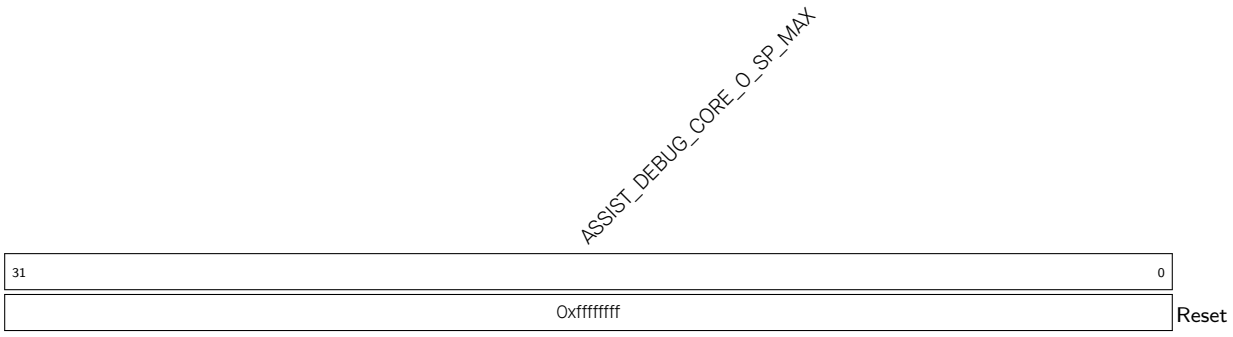
ASSIST_DEBUG_CORE_0_AREA_SP Represents the SP value when an interrupt is triggered during region monitoring. (RO)

Register 17.24. ASSIST_DEBUG_CORE_0_SP_MIN_REG (0x0038)



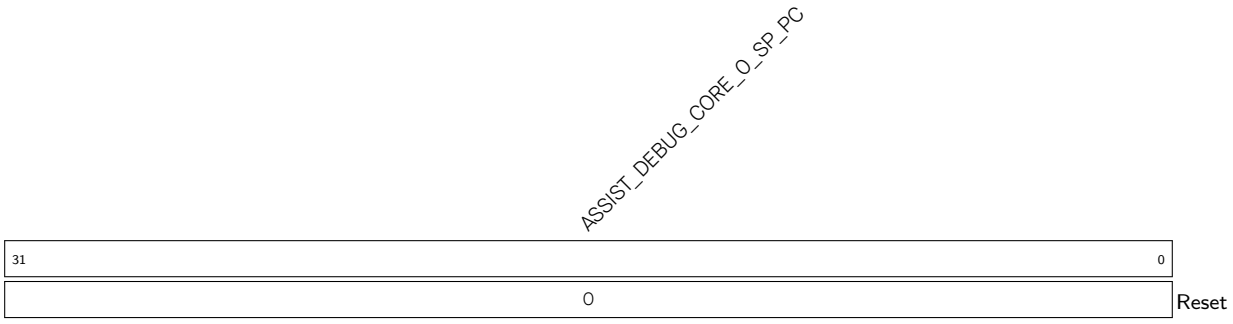
ASSIST_DEBUG_CORE_0_SP_MIN Configures the starting address of SP monitored region. (R/W)

Register 17.25. ASSIST_DEBUG_CORE_0_SP_MAX_REG (0x003C)



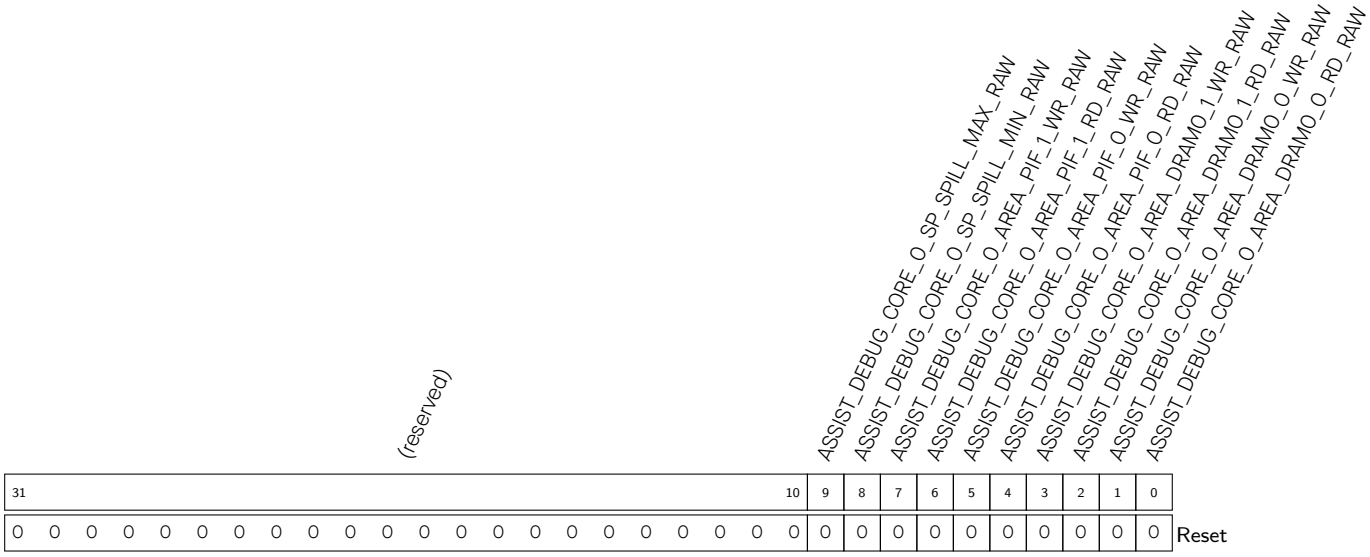
ASSIST_DEBUG_CORE_0_SP_MAX Configures the ending address of SP monitored region. (R/W)

Register 17.26. ASSIST_DEBUG_CORE_0_SP_PC_REG (0x0040)



ASSIST_DEBUG_CORE_0_SP_PC Represents the PC value during stack pointer monitoring. (RO)

Register 17.27. ASSIST_DEBUG_CORE_0_INTR_RAW_REG (0x0004)



ASSIST_DEBUG_CORE_0_AREA_DRAM0_0_RD_RAW The raw interrupt status of read operations in region 0 by data bus. (RO)

ASSIST_DEBUG_CORE_0_AREA_DRAM0_0_WR_RAW The raw interrupt status of write operations in region 0 by data bus. (RO)

ASSIST_DEBUG_CORE_0_AREA_DRAM0_1_RD_RAW The raw interrupt status of read operations in region 1 by data bus. (RO)

ASSIST_DEBUG_CORE_0_AREA_DRAM0_1_WR_RAW The raw interrupt status of write operations in region 1 by data bus. (RO)

ASSIST_DEBUG_CORE_0_AREA_PIF_0_RD_RAW The raw interrupt status of read operations in region 0 by peripheral bus. (RO)

ASSIST_DEBUG_CORE_0_AREA_PIF_0_WR_RAW The raw interrupt status of write operations in region 0 by peripheral bus. (RO)

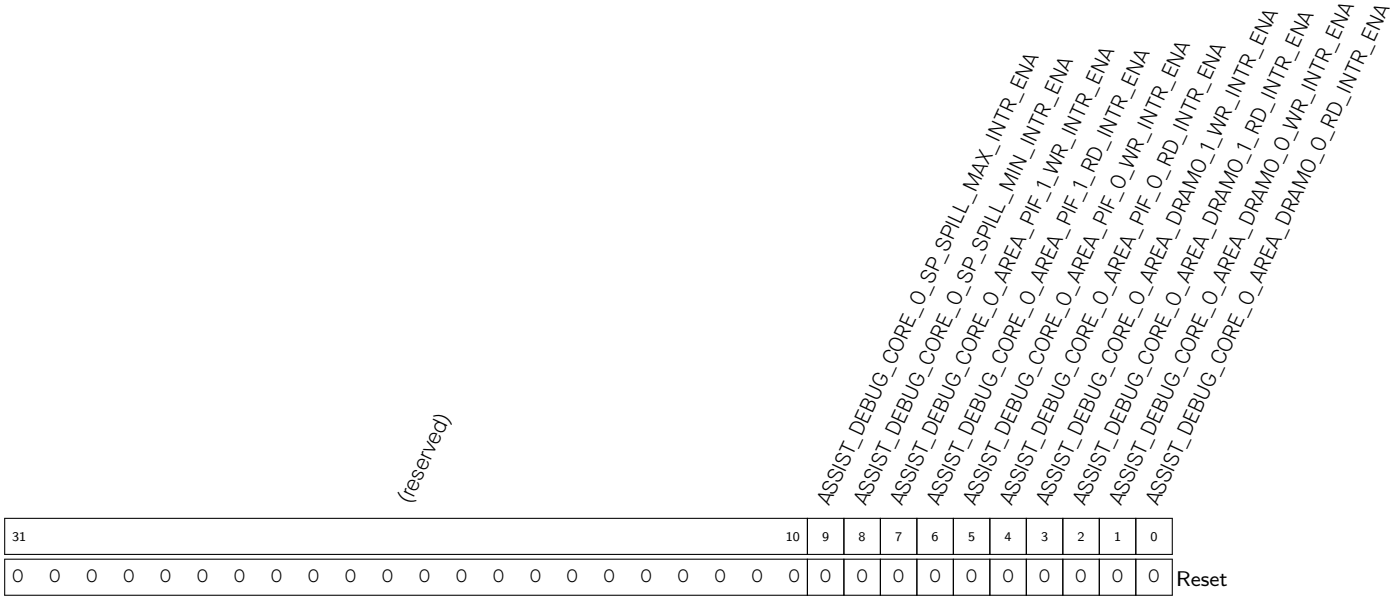
ASSIST_DEBUG_CORE_0_AREA_PIF_1_RD_RAW The raw interrupt status of read operations in region 1 by peripheral bus. (RO)

ASSIST_DEBUG_CORE_0_AREA_PIF_1_WR_RAW The raw interrupt status of write operations in region 1 by peripheral bus. (RO)

ASSIST_DEBUG_CORE_0_SP_SPILL_MIN_RAW The raw interrupt status of SP less than the starting address of SP monitored region. (RO)

ASSIST_DEBUG_CORE_0_SP_SPILL_MAX_RAW The raw interrupt status of SP greater than the ending address of SP monitored region. (RO)

Register 17.28. ASSIST_DEBUG_CORE_0_INTR_ENA_REG (0x0008)



ASSIST_DEBUG_CORE_0_AREA_DRAM0_0_RD_INTR_ENA Write 1 to enable the interrupt for read operations in region 0 by data bus. (R/W)

ASSIST_DEBUG_CORE_0_AREA_DRAM0_0_WR_INTR_ENA Write 1 to enable the interrupt for write operations in region 0 by data bus. (R/W)

ASSIST_DEBUG_CORE_0_AREA_DRAM0_1_RD_INTR_ENA Write 1 to enable the interrupt for read operations in region 1 by data bus. (R/W)

ASSIST_DEBUG_CORE_0_AREA_DRAM0_1_WR_INTR_ENA Write 1 to enable the interrupt for write operations in region 1 by data bus. (R/W)

ASSIST_DEBUG_CORE_0_AREA_PIF_0_RD_INTR_ENA Write 1 to enable the interrupt for read operations in region 0 by peripheral bus. (R/W)

ASSIST_DEBUG_CORE_0_AREA_PIF_0_WR_INTR_ENA Write 1 to enable the interrupt for write operations in region 0 by peripheral bus. (R/W)

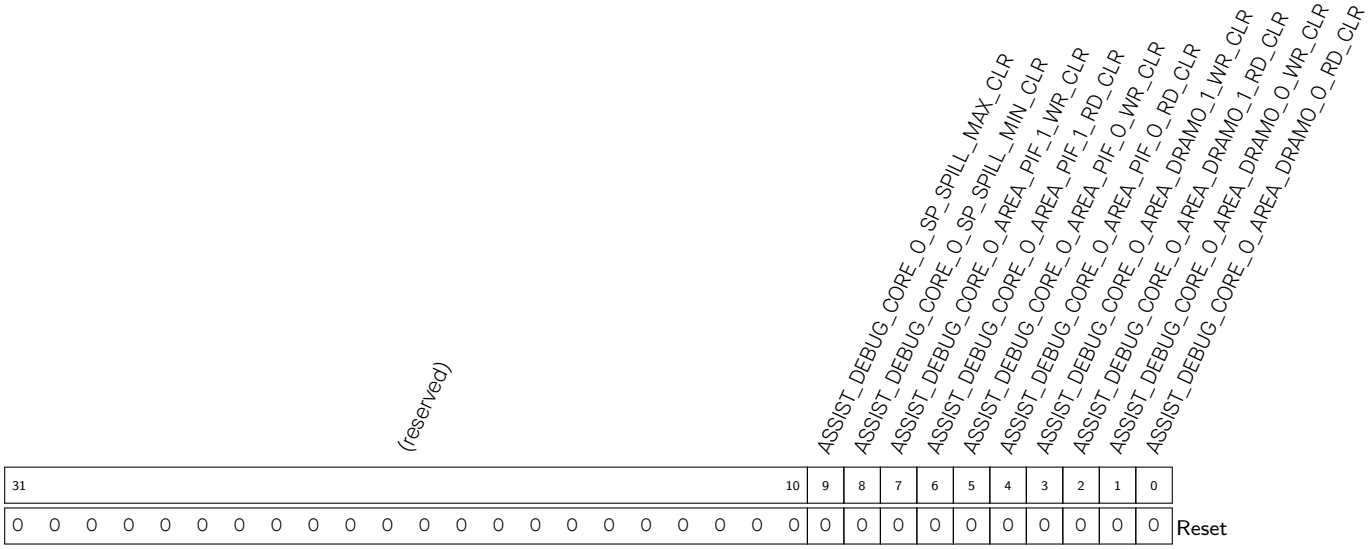
ASSIST_DEBUG_CORE_0_AREA_PIF_1_RD_INTR_ENA Write 1 to enable the interrupt for read operations in region 1 by peripheral bus. (R/W)

ASSIST_DEBUG_CORE_0_AREA_PIF_1_WR_INTR_ENA Write 1 to enable the interrupt for write operations in region 1 by peripheral bus. (R/W)

ASSIST_DEBUG_CORE_0_SP_SPILL_MIN_INTR_ENA Write 1 to enable the interrupt for SP less than the starting address of SP monitored region. (R/W)

ASSIST_DEBUG_CORE_0_SP_SPILL_MAX_INTR_ENA Write 1 to enable the interrupt for SP greater than the ending address of SP monitored region. (R/W)

Register 17.29. ASSIST_DEBUG_CORE_0_INTR_CLR_REG (0x000C)



ASSIST_DEBUG_CORE_0_AREA_DRAM0_0_RD_CLR Write 1 to clear the interrupt for read operations in region 0 by data bus. (R/W)

ASSIST_DEBUG_CORE_0_AREA_DRAM0_0_WR_CLR Write 1 to clear the interrupt for write operations in region 0 by data bus. (R/W)

ASSIST_DEBUG_CORE_0_AREA_DRAM0_1_RD_CLR Write 1 to clear the interrupt for read operations in region 1 by data bus. (R/W)

ASSIST_DEBUG_CORE_0_AREA_DRAM0_1_WR_CLR Write 1 to clear the interrupt for write operations in region 1 by data bus. (R/W)

ASSIST_DEBUG_CORE_0_AREA_PIF_0_RD_CLR Write 1 to clear the interrupt for read operations in region 0 by peripheral bus. (R/W)

ASSIST_DEBUG_CORE_0_AREA_PIF_0_WR_CLR Write 1 to clear the interrupt for write operations in region 0 by peripheral bus. (R/W)

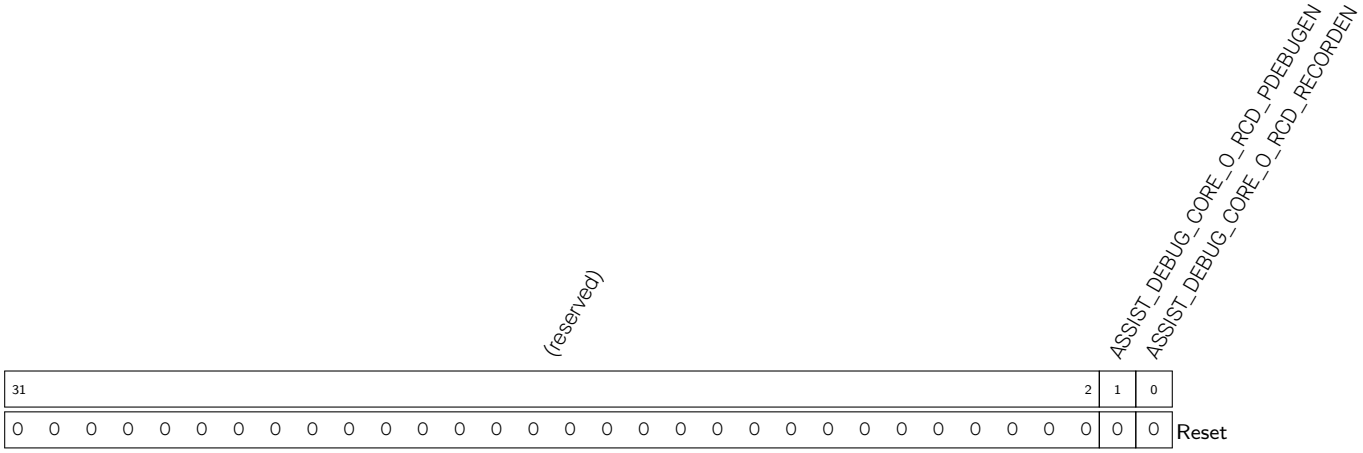
ASSIST_DEBUG_CORE_0_AREA_PIF_1_RD_CLR Write 1 to clear the interrupt for read operations in region 1 by peripheral bus. (R/W)

ASSIST_DEBUG_CORE_0_AREA_PIF_1_WR_CLR Write 1 to clear the interrupt for write operations in region 1 by peripheral bus. (R/W)

ASSIST_DEBUG_CORE_0_SP_SPILL_MIN_CLR Write 1 to clear the interrupt for SP less than the starting address of SP monitored region. (R/W)

ASSIST_DEBUG_CORE_0_SP_SPILL_MAX_CLR Write 1 to clear the interrupt for SP greater than the ending address of SP monitored region. (R/W)

Register 17.30. ASSIST_DEBUG_CORE_0_RCD_EN_REG (0x0044)



ASSIST_DEBUG_CORE_0_RCD_RECORDEN Configures whether to enable PC logging.

0: Disable

1: ASSIST_DEBUG_CORE_0_RCD_PDEBUGPC_REG starts to record PC in real time

(R/W)

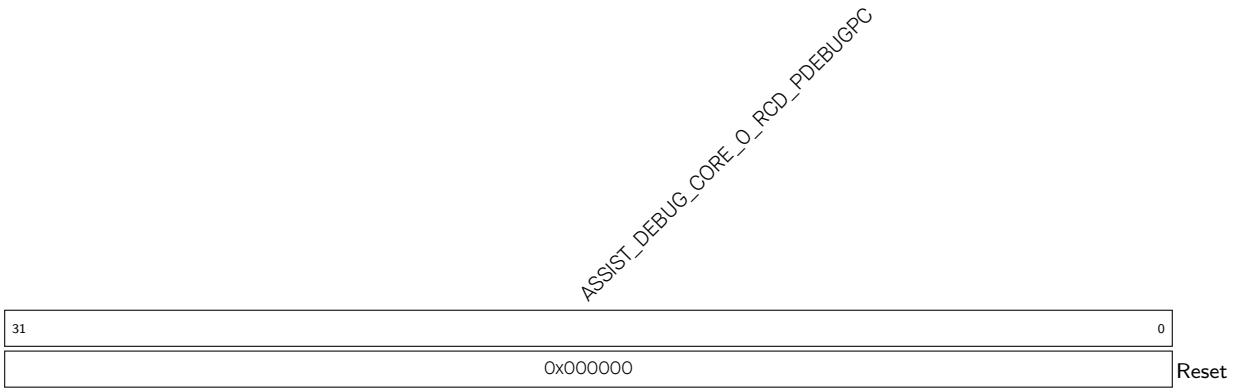
ASSIST_DEBUG_CORE_0_RCD_PDEBUGEN Configures whether to enable CPU debugging.

0: Disable

1: CPU outputs PC

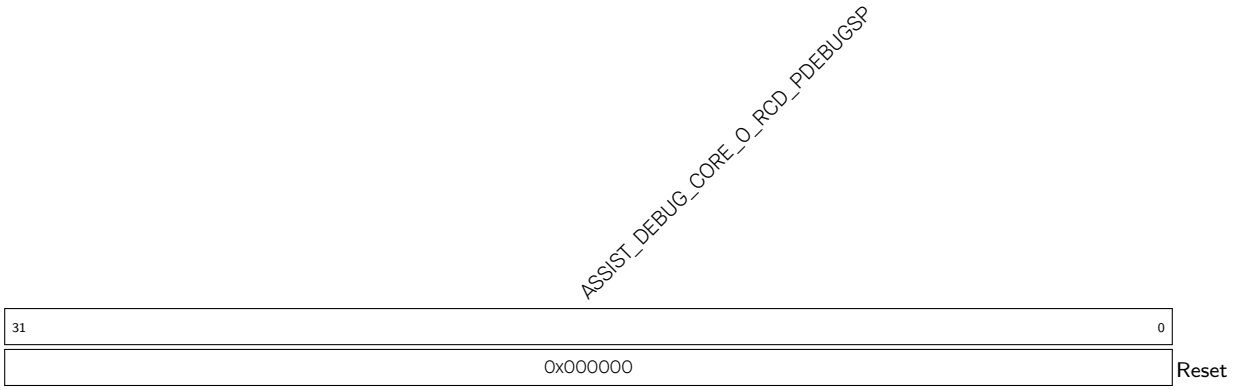
(R/W)

Register 17.31. ASSIST_DEBUG_CORE_0_RCD_PDEBUGPC_REG (0x0048)



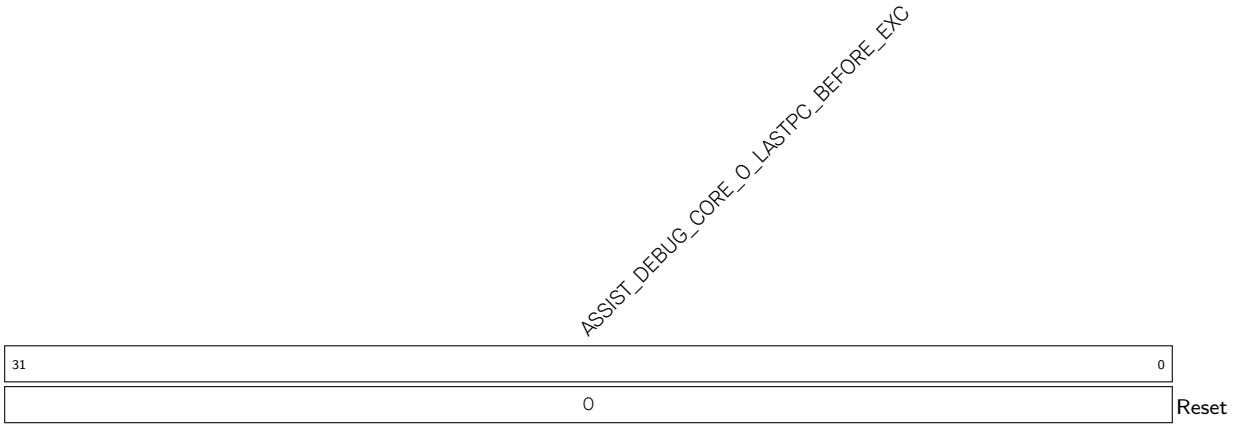
ASSIST_DEBUG_CORE_0_RCD_PDEBUGPC Represents the PC value at CPU reset. (RO)

Register 17.32. ASSIST_DEBUG_CORE_0_RCD_PDEBUGSP_REG (0x004C)



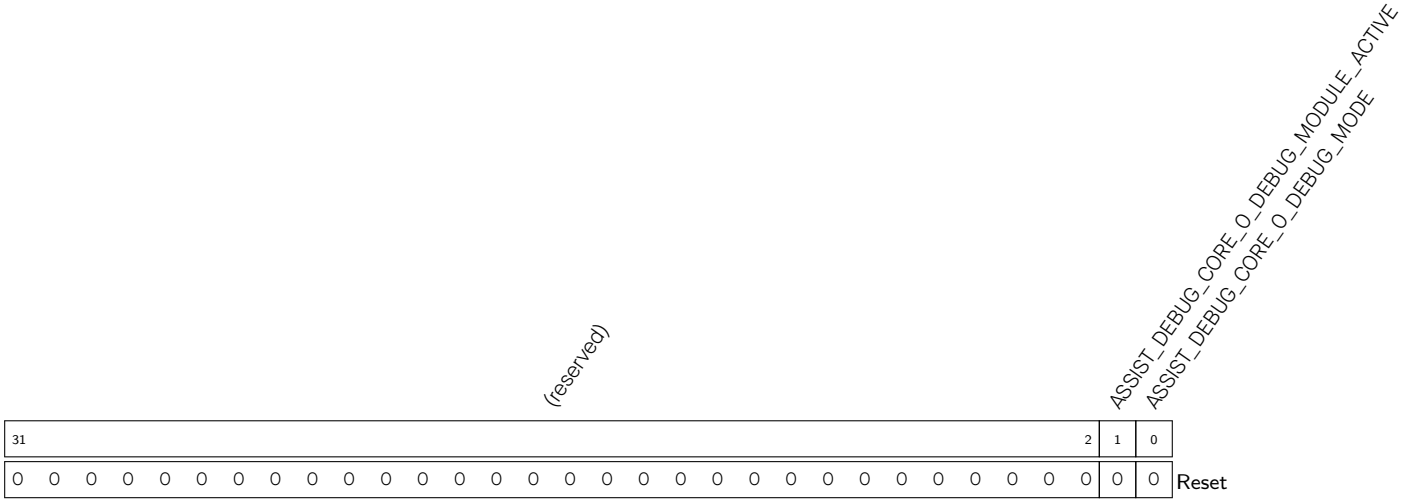
ASSIST_DEBUG_CORE_0_RCD_PDEBUGSP Represents SP. (RO)

Register 17.33. ASSIST_DEBUG_CORE_0_LASTPC_BEFORE_EXCEPTION_REG (0x0070)



ASSIST_DEBUG_CORE_0_LASTPC_BEFORE_EXC Represents the PC of the last command before the CPU enters an exception. (RO)

Register 17.34. ASSIST_DEBUG_CORE_0_DEBUG_MODE_REG (0x0074)



ASSIST_DEBUG_CORE_0_DEBUG_MODE Represents whether ESP-RISC-V CPU is in debugging mode.

1: In debugging mode

0: Not in debugging mode

(RO)

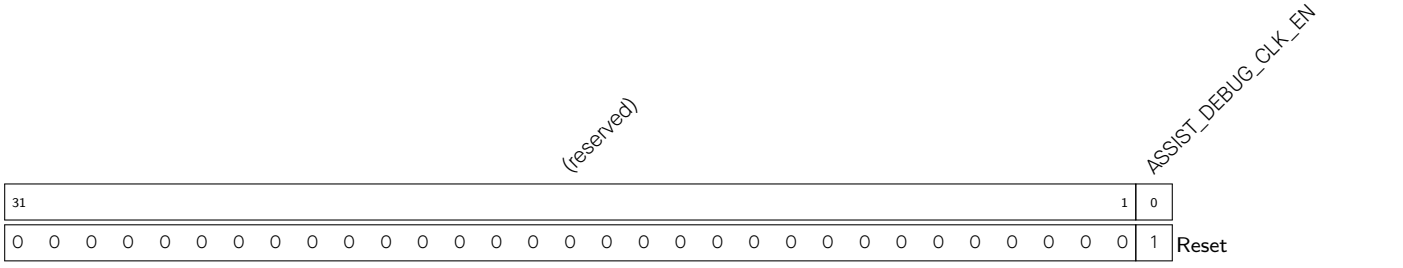
ASSIST_DEBUG_CORE_0_DEBUG_MODULE_ACTIVE Represents the status of the ESP-RISC-V CPU debug module.

1: Active status

Other: Inactive status

(RO)

Register 17.35. ASSIST_DEBUG_CLOCK_GATE_REG (0x0078)



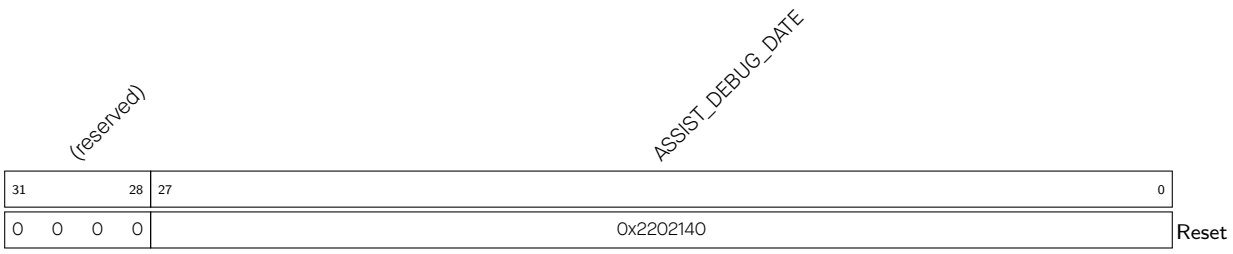
ASSIST_DEBUG_CLK_EN Configures whether to enable the register clock gating.

0: Disable

1: Enable

(R/W)

Register 17.36. ASSIST_DEBUG_DATE_REG (0x03FC)



ASSIST_DEBUG_DATE Version control register. (R/W)

Chapter 18

Power Supply Detector

18.1 Overview

ESP32-H2 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 triggers a reset to protect the SoC during low-voltage conditions, 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.

18.2 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.

18.3 Functional Description

18.3.1 Architecture

Figure 18.3-1 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.

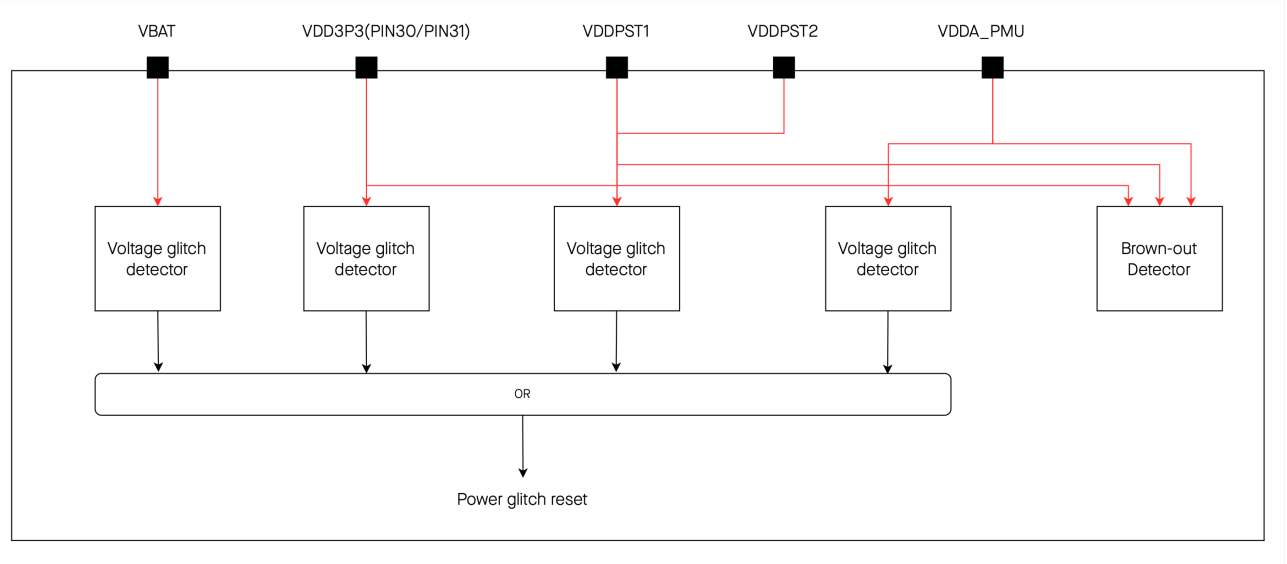


Figure 18.3-1. Architecture of Power Supply Detector

18.3.2 Brown-out Detector

The brown-out detector checks the voltage of pins VDD3P3 (PIN1,PIN2), VDDPST1, VDDPST2, VDDA_PMU, and VDD3P3 (PIN30,PIN31), about every 280 μs. 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.

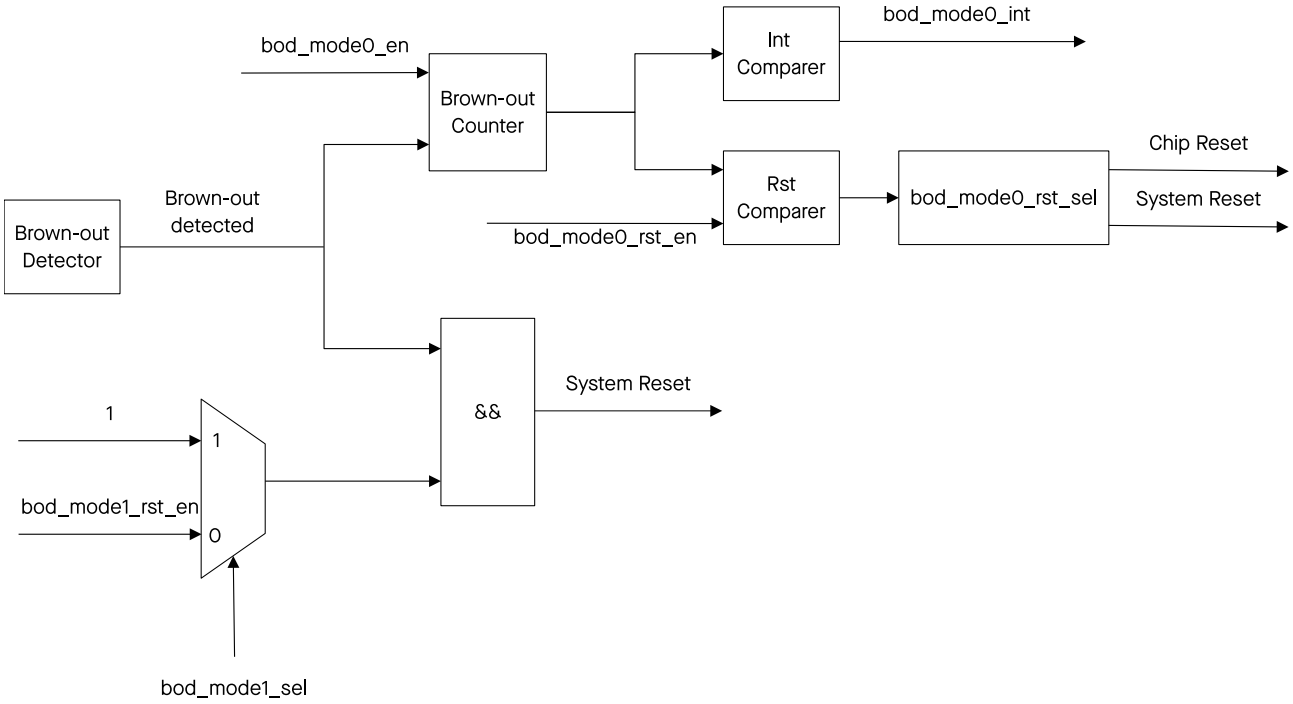


Figure 18.3-2. Brown-out Reset Workflow

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 after the brown-out detector is triggered in mode 0:
  - 0: chip reset
  - 1: system reset

  For more information regarding chip reset and system reset, please refer to Chapter 7 *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

18.3.3 Voltage Glitch Detectors

Figure 18.3-3 shows the structure of the voltage glitch detectors.

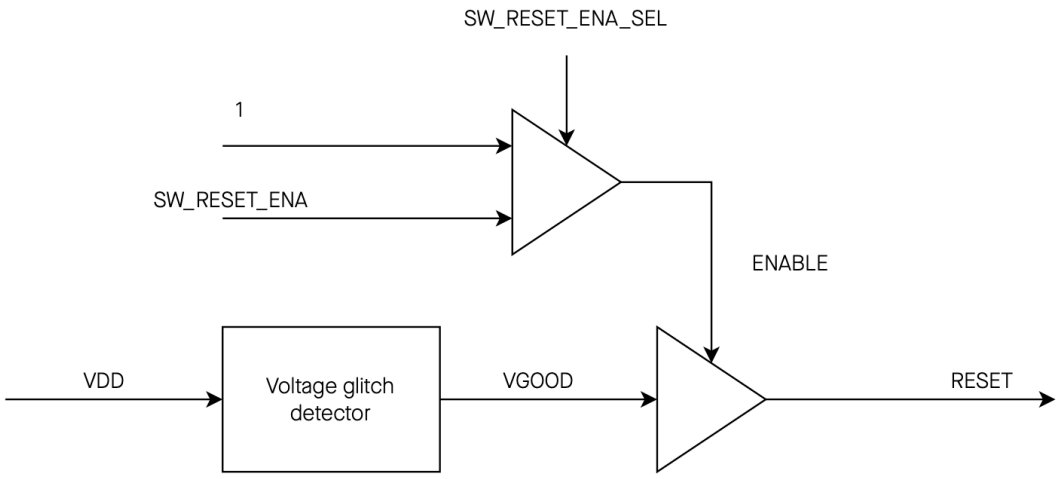


Figure 18.3-3. Structure of Voltage Glitch Detectors

The ESP32-H2 has four voltage glitch detectors, each of which can detect glitches longer than 50 ns on the corresponding power supply pins, including VDDA_PMU, VBAT, VDD3P3 (PIN31,PIN30), and VDDPST1/2. VDDPST1 and VDDPST2 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, the detector is forcibly enabled by eFuse; and when set to 0, the detector enable state is controlled by the software register SW_RESET_ENA. The configuration for detecting different power pins is shown in the table below:

Table 18.3-1. Configuration for Detecting Power Pins

VDD Pins	SW_RESET_ENA	SW_RESET_ENA_SEL
VDDA_PMU	LP_ANA_PWR_GLITCH_RESET_ENA[0]	LP_ANA_ANA_FIB_ENA[2]
VBAT	LP_ANA_PWR_GLITCH_RESET_ENA[1]	LP_ANA_ANA_FIB_ENA[3]
VDD3P3 (PIN31,PIN30)	LP_ANA_PWR_GLITCH_RESET_ENA[4]	LP_ANA_ANA_FIB_ENA[4]
VDDPST1/2	LP_ANA_PWR_GLITCH_RESET_ENA[5]	LP_ANA_ANA_FIB_ENA[5]

18.4 Interrupts

ESP32-H2's power supply detector can generate the following interrupt signal and send it to the Interrupt Matrix.

- LP_ANA_INTR (only for HP CPU)

The interrupt signal is generated by the power supply detector's internal interrupt sources which are listed in Table 18.4-1 with their trigger conditions and signals.

Table 18.4-1. 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

Note:
For definitions of *interrupt*, *interrupt signal*, *interrupt source*, and their correlations, please refer to Chapter 9 *Interrupt Matrix (INTMTX)* > Section 9.2 *Interrupt Terminology in ESP32-H2*.

Each interrupt source can be configured by a common set of registers that are described in Chapter *Interrupt Configuration Registers*. The specific registers can be found in Section 18.5 *Register Summary*.

18.5 Register Summary

The addresses in this section are relative to power detector base address provided in Table 4.3-2 in Chapter 4 *System and Memory*.

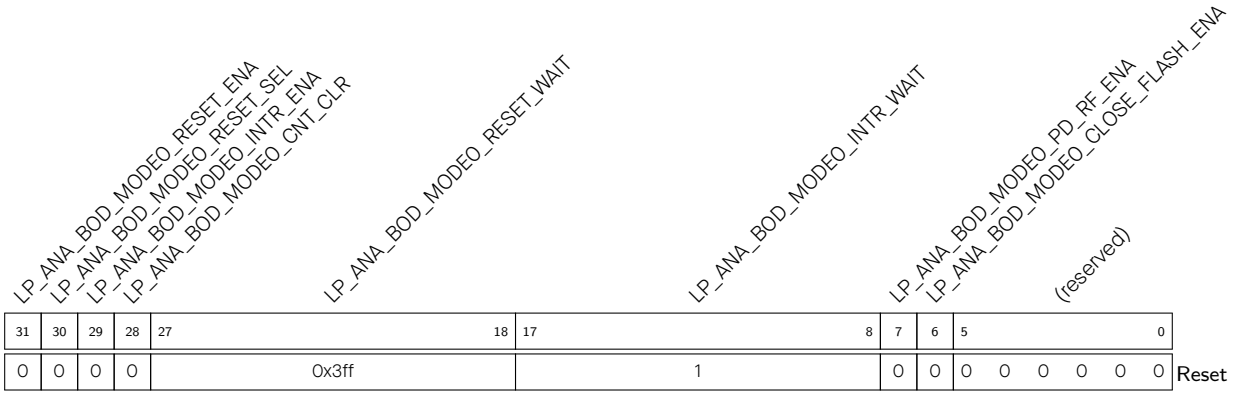
The abbreviations given in Column **Access** are explained in Section .

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
Version Control Registers			
LP_ANA_DATE_REG	Version control register	0x03FC	R/W

18.6 Registers

The addresses in this section are relative to power detector base address provided in Table 4.3-2 in Chapter 4 *System and Memory*.

Register 18.1. 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 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 18.1. LP_ANA_BOD_MODE0_CNTL_REG (0x0000)

Continued from the previous page...

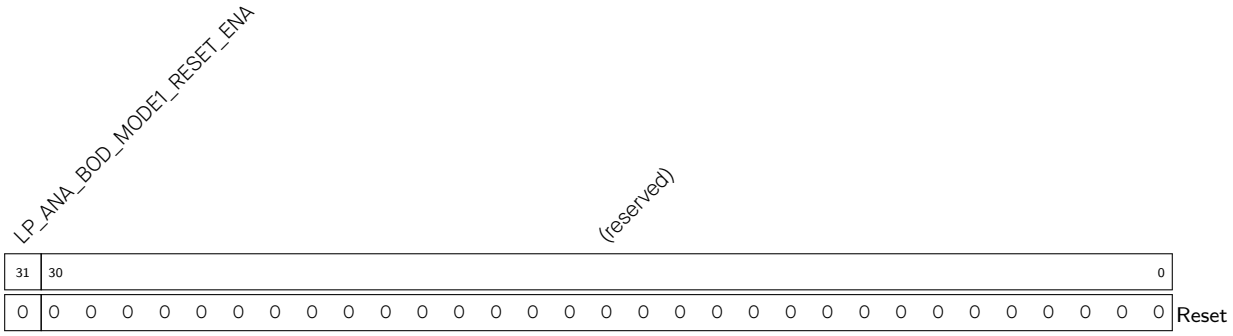
LP_ANA_BOD_MODE0_RESET_ENA Configures whether to enable reset for the brown-out detector.

0: Disable

1: Enable

(R/W)

Register 18.2. LP_ANA_BOD_MODE1_CNTL_REG (0x0004)



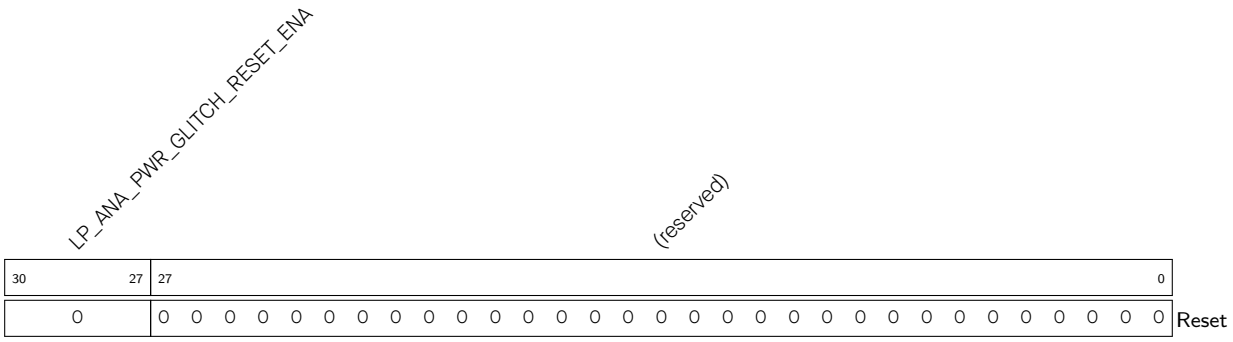
LP_ANA_BOD_MODE1_RESET_ENA Configures whether to enable brown-out detector mode 1.

0: Disable

1: Enable

(R/W)

Register 18.3. LP_ANA_POWER_GLITCH_CNTL_REG (0x0008)



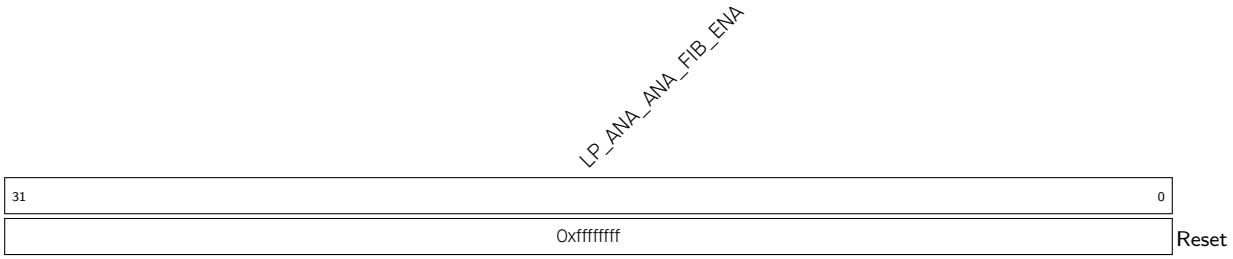
LP_ANA_PWR_GLITCH_RESET_ENA Configures whether to enable the voltage glitch detectors. Bit0, bit1, bit2, bit3 correspond to VDDPST2/3, VDDPST1, VDDA3, and VDDA8, respectively.

0: Disable

1: Enable

(R/W)

Register 18.4. LP_ANA_FIB_ENABLE_REG (0x000C)



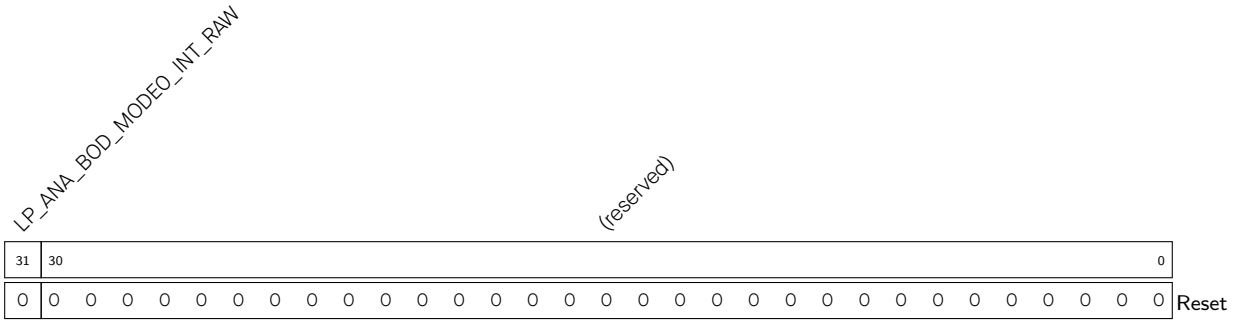
LP_ANA_ANA_FIB_ENA Controls the enabling of the voltage glitch detectors. Bit2, bit3, bit4, bit5 correspond to VDDPST2/3, VDDPST1, VDDA3, and VDDA8, respectively.

0: Controlled by LP_ANA_PWR_GLITCH_RESET_ENA

1: Forcibly enabled by hardware

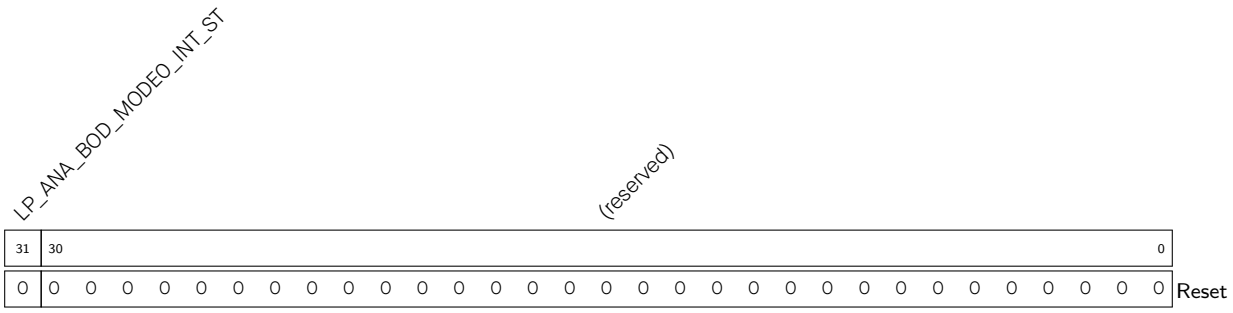
(R/W)

Register 18.5. LP_ANA_INT_RAW_REG (0x0010)



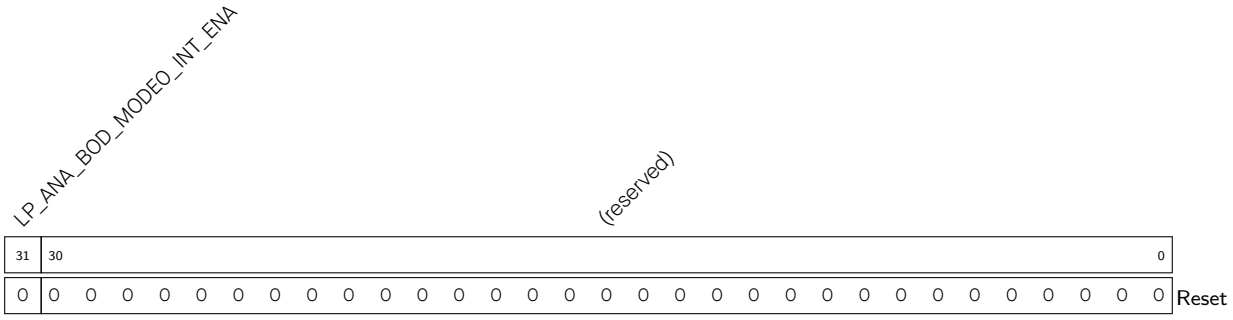
LP_ANA_BOD_MODE0_INT_RAW The raw interrupt status of LP_ANA_BOD_MODE0_INT. (R/WTC/SS)

Register 18.6. LP_ANA_INT_ST_REG (0x0014)



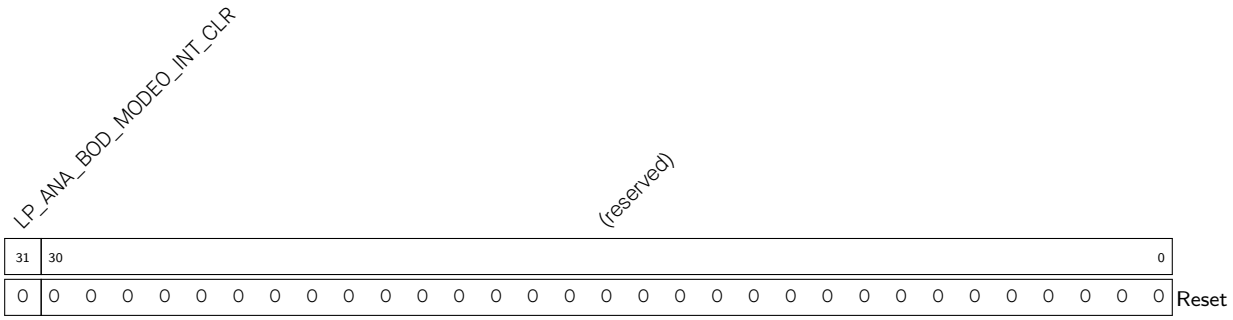
LP_ANA_BOD_MODE0_INT_ST The masked interrupt status of LP_ANA_BOD_MODE0_INT. (RO)

Register 18.7. LP_ANA_INT_ENA_REG (0x0018)



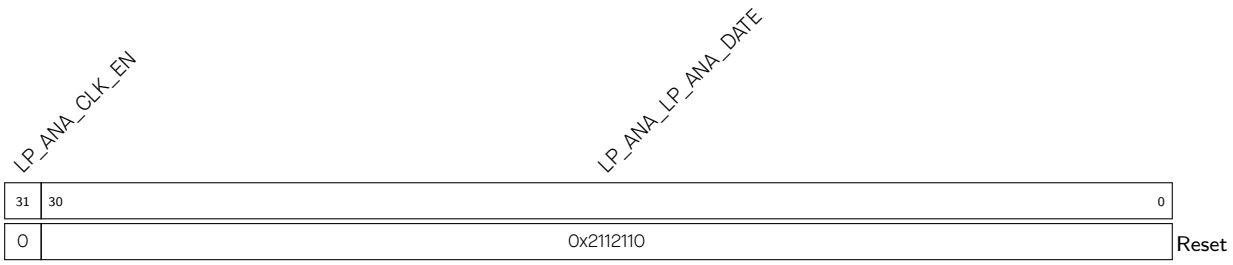
LP_ANA_BOD_MODE0_INT_ENA Write 1 to enable LP_ANA_BOD_MODE0_INT. (R/W)

Register 18.8. LP_ANA_INT_CLR_REG (0x001C)



LP_ANA_BOD_MODE0_INT_CLR Write 1 to clear LP_ANA_BOD_MODE0_INT. (WT)

Register 18.9. 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 affect the register access.

(R/W)

Part IV

Cryptography/Security Component

Dedicated to security features, this part explores cryptographic accelerators like AES, ECC, and HMAC. It also covers digital signatures, random number generation, and encryption/decryption algorithms, showcasing the SoC's capabilities in cryptography and secure data processing.

Chapter 19

AES Accelerator (AES)

19.1 Introduction

ESP32-H2 integrates an Advanced Encryption Standard (AES) accelerator, which is a hardware device that speeds up computation using AES algorithm significantly, compared to AES algorithms implemented solely in software. The AES accelerator integrated in ESP32-H2 has two working modes, which are Typical AES and DMA-AES.

19.2 Features

The following functionality is supported:

- Typical AES working mode
  - AES-128/AES-256 encryption and decryption
- DMA-AES working mode
  - AES-128/AES-256 encryption and decryption
  - Block cipher mode
    - ECB (Electronic Codebook)
    - CBC (Cipher Block Chaining)
    - OFB (Output Feedback)
    - CTR (Counter)
    - CFB8 (8-bit Cipher Feedback)
    - CFB128 (128-bit Cipher Feedback)
  - Interrupt on completion of computation

19.3 Clock and Reset

The AES accelerator is activated by setting the PCR_AES_CLK_EN bit and clearing the PCR_AES_RST_EN bit in the PCR_AES_CONF_REG register. Besides, due to resource reuse between cryptography accelerator modules, users also need to additionally clear the PCR_DS_RST_EN bit in the PCR_DS_CONF_REG register.

19.4 AES Working Modes

The AES accelerator integrated in ESP32-H2 has two working modes, which are Typical AES and DMA-AES.

- Typical AES Working Mode:
  - Supports encryption and decryption using cryptographic keys of 128 and 256 bits, specified in NIST FIPS 197.

  In this working mode, the plaintext and ciphertext is written and read via CPU directly.
- DMA-AES Working Mode:
  - Supports encryption and decryption using cryptographic keys of 128 and 256 bits, specified in NIST FIPS 197;
  - Supports block cipher modes ECB, CBC, OFB, CTR, CFB8, and CFB128 under NIST SP 800-38A.

  In this working mode, the plaintext and ciphertext are written and read via DMA. An interrupt will be generated when operation completes.

Users can choose the working mode for AES accelerator by configuring the AES_DMA_ENABLE_REG register according to Table 19.4-1 below.

Table 19.4-1. AES Accelerator Working Mode

AES_DMA_ENABLE_REG	Working Mode
0	Typical AES
1	DMA-AES

Users can choose the length of cryptographic keys and encryption/decryption by configuring the AES_MODE_REG register according to Table 19.4-2 below.

Table 19.4-2. Key Length and Encryption/Decryption

AES_MODE_REG[2:0]	Key Length and Encryption/Decryption
0	AES-128 encryption
1	reserved
2	AES-256 encryption
3	reserved
4	AES-128 decryption
5	reserved
6	AES-256 decryption
7	reserved

For a detailed introduction to these two working modes, please refer to Section 19.5 and Section 19.6 below.

Notice:

ESP32-H2's RSA Digital Signature Peripheral (RSA_DS) module will call the AES accelerator. Therefore, users cannot

access the AES accelerator when RSA Digital Signature Peripheral (RSA_DS) module is working.

19.5 Typical AES Working Mode

In the Typical AES working mode, users can check the working status of the AES accelerator by inquiring the AES_STATE_REG register and comparing the return value against the Table 19.5-1 below.

Table 19.5-1. Working Status under Typical AES Working Mode

AES_STATE_REG	Status	Description
0	IDLE	The AES accelerator is idle or completed operation.
1	WORK	The AES accelerator is in the middle of an operation.

19.5.1 Key, Plaintext, and Ciphertext

The encryption or decryption key is stored in AES_KEY_*n*_REG, which is a set of eight 32-bit registers.

- For AES-128 encryption or decryption, the 128-bit key is stored in AES_KEY_0_REG ~ AES_KEY_3_REG.
- For AES-256 encryption or decryption, the 256-bit key is stored in AES_KEY_0_REG ~ AES_KEY_7_REG.

The plaintext and ciphertext are stored in AES_TEXT_IN_*m*_REG and AES_TEXT_OUT_*m*_REG, which are two sets of four 32-bit registers.

- For AES-128 or AES-256 encryption, the AES_TEXT_IN_*m*_REG registers are initialized with plaintext. Then, the AES accelerator stores the ciphertext into AES_TEXT_OUT_*m*_REG after operation.
- For AES-128 or AES-256 decryption, the AES_TEXT_IN_*m*_REG registers are initialized with ciphertext. Then, the AES accelerator stores the plaintext into AES_TEXT_OUT_*m*_REG after operation.

19.5.2 Endianness

Text Endianness

In Typical AES working mode, the AES accelerator uses cryptographic keys to encrypt and decrypt data in blocks of 128 bits. When filling data into AES_TEXT_IN_*m*_REG register or reading result from AES_TEXT_OUT_*m*_REG registers, users should follow the text endianness type specified in Table 19.5-2.

Table 19.5-2. Text Endianness Type for Typical AES

Plaintext/Ciphertext					
State[1]		c[2]			
		0	1	2	3
r	0	AES_TEXT_*x*_0_REG[7:0]	AES_TEXT_*x*_1_REG[7:0]	AES_TEXT_*x*_2_REG[7:0]	AES_TEXT_*x*_3_REG[7:0]
	1	AES_TEXT_*x*_0_REG[15:8]	AES_TEXT_*x*_1_REG[15:8]	AES_TEXT_*x*_2_REG[15:8]	AES_TEXT_*x*_3_REG[15:8]
	2	AES_TEXT_*x*_0_REG[23:16]	AES_TEXT_*x*_1_REG[23:16]	AES_TEXT_*x*_2_REG[23:16]	AES_TEXT_*x*_3_REG[23:16]
	3	AES_TEXT_*x*_0_REG[31:24]	AES_TEXT_*x*_1_REG[31:24]	AES_TEXT_*x*_2_REG[31:24]	AES_TEXT_*x*_3_REG[31:24]

[1] The definition of "State (including c and r)" is described in Section 3.4 *The State* in NIST FIPS 197.

[2] Where *x* = IN or OUT.

Key Endianness

In Typical AES working mode, when filling keys into AES_KEY_*m*_REG registers, users should follow the key endianness type specified in Table 19.5-3 and Table 19.5-4.

Table 19.5-3. Key Endianness Type for AES-128 Encryption and Decryption

Bit[1]	w[0]	w[1]	w[2]	w[3][2]
[31:24]	AES_KEY_0_REG[7:0]	AES_KEY_1_REG[7:0]	AES_KEY_2_REG[7:0]	AES_KEY_3_REG[7:0]
[23:16]	AES_KEY_0_REG[15:8]	AES_KEY_1_REG[15:8]	AES_KEY_2_REG[15:8]	AES_KEY_3_REG[15:8]
[15:8]	AES_KEY_0_REG[23:16]	AES_KEY_1_REG[23:16]	AES_KEY_2_REG[23:16]	AES_KEY_3_REG[23:16]
[7:0]	AES_KEY_0_REG[31:24]	AES_KEY_1_REG[31:24]	AES_KEY_2_REG[31:24]	AES_KEY_3_REG[31:24]

[1] Column "Bit" specifies the bytes of each word stored in w[0] ~ w[3].

[2] w[0] ~ w[3] are "the first Nk words of the expanded key" as specified in Section 5.2 *Key Expansion* in NIST FIPS 197.

Table 19.5-4. Key Endianness Type for AES-256 Encryption and Decryption

Bit [1]	w[0]	w[1]	w[2]	w[3]	w[4]	w[5]	w[6]	w[7] [2]
[31:24]	AES_KEY_0_REG[7:0]	AES_KEY_1_REG[7:0]	AES_KEY_2_REG[7:0]	AES_KEY_3_REG[7:0]	AES_KEY_4_REG[7:0]	AES_KEY_5_REG[7:0]	AES_KEY_6_REG[7:0]	AES_KEY_7_REG[7:0]
[23:16]	AES_KEY_0_REG[15:8]	AES_KEY_1_REG[15:8]	AES_KEY_2_REG[15:8]	AES_KEY_3_REG[15:8]	AES_KEY_4_REG[15:8]	AES_KEY_5_REG[15:8]	AES_KEY_6_REG[15:8]	AES_KEY_7_REG[15:8]
[15:8]	AES_KEY_0_REG[23:16]	AES_KEY_1_REG[23:16]	AES_KEY_2_REG[23:16]	AES_KEY_3_REG[23:16]	AES_KEY_4_REG[23:16]	AES_KEY_5_REG[23:16]	AES_KEY_6_REG[23:16]	AES_KEY_7_REG[23:16]
[7:0]	AES_KEY_0_REG[31:24]	AES_KEY_1_REG[31:24]	AES_KEY_2_REG[31:24]	AES_KEY_3_REG[31:24]	AES_KEY_4_REG[31:24]	AES_KEY_5_REG[31:24]	AES_KEY_6_REG[31:24]	AES_KEY_7_REG[31:24]

[1] Column "Bit" specifies the bytes of each word stored in w[0] ~ w[7].

[2] w[0] ~ w[7] are "the first Nk words of the expanded key" as specified in Chapter 5.2 *Key Expansion* in NIST FIPS 197.

19.5.3 Operation Process

Single Operation

1. Write 0 to the AES_DMA_ENABLE_REG register.
2. Initialize registers AES_MODE_REG, AES_KEY_*n*_REG, and AES_TEXT_IN_*m*_REG.
3. Start operation by writing 1 to the AES_TRIGGER_REG register.
4. Wait till the content of the AES_STATE_REG register becomes 0, which indicates the operation is completed.
5. Read results from the AES_TEXT_OUT_*m*_REG register.

Consecutive Operations

In consecutive operations, primarily the input AES_TEXT_IN_*m*_REG and output AES_TEXT_OUT_*m*_REG registers (*m*: 0-3) are being written and read, while the content of AES_DMA_ENABLE_REG, AES_MODE_REG, and AES_KEY_*n*_REG is kept unchanged. Therefore, the initialization can be simplified during the consecutive operation.

1. Write 0 to the AES_DMA_ENABLE_REG register before starting the first operation.
2. Initialize registers AES_MODE_REG and AES_KEY_*n*_REG before starting the first operation.
3. Update the content of AES_TEXT_IN_*m*_REG.
4. Start operation by writing 1 to the AES_TRIGGER_REG register.
5. Wait till the content of the AES_STATE_REG register becomes 0, which indicates the operation completes.
6. Read results from the AES_TEXT_OUT_*m*_REG register, and return to Step 3 to continue the next operation.

19.6 DMA-AES Working Mode

In the DMA-AES working mode, the AES accelerator supports six block cipher modes: ECB, CBC, OFB, CTR, CFB8, and CFB128. Users can choose the block cipher mode by configuring the AES_BLOCK_MODE_REG register according to Table 19.6-1 below.

Table 19.6-1. Block Cipher Mode

AES_BLOCK_MODE_REG[2:0]	Block Cipher Mode
0	ECB (Electronic Codebook)
1	CBC (Cipher Block Chaining)
2	OFB (Output Feedback)
3	CTR (Counter)
4	CFB8 (8-bit Cipher Feedback)
5	CFB128 (128-bit Cipher Feedback)
6	reserved
7	reserved

Users can check the working status of the AES accelerator by inquiring the AES_STATE_REG register and comparing the return value against the Table 19.6-2 below.

Table 19.6-2. Working Status under DMA-AES Working mode

AES_STATE_REG[1:0]	Status	Description
0	IDLE	The AES accelerator is idle.
1	WORK	The AES accelerator is in the middle of an operation.
2	DONE	The AES accelerator completed operations.

When working in the DMA-AES working mode, the AES accelerator supports interrupt on the completion of computation. To enable this function, write 1 to the AES_INT_ENA_REG register. By default, the interrupt function is disabled. Also, note that the interrupt should be cleared by software after use.

19.6.1 Key, Plaintext, and Ciphertext

Block Operation

During the block operations, the AES accelerator reads source data from DMA, and writes result data to DMA after the computation.

- For encryption, DMA reads plaintext from memory, then passes it to AES as source data. After computation, AES passes ciphertext as result data back to DMA to write into memory.
- For decryption, DMA reads ciphertext from memory, then passes it to AES as source data. After computation, AES passes plaintext as result data back to DMA to write into memory.

During block operations, the lengths of the source data and result data are the same. The total computation time is reduced because the DMA data operation and AES computation can happen concurrently.

The length of source data for AES accelerator under DMA-AES working mode must be 128 bits or the integral multiples of 128 bits. Otherwise, trailing zeros will be added to the original source data, so the length of source data equals to the nearest integral multiples of 128 bits. Please see details in Table 19.6-3 below.

Table 19.6-3. TEXT-PADDING

Function : TEXT-PADDING( )
Input : X, bit string. **Output** : Y = **TEXT-PADDING**(X), whose length is the nearest integral multiples of 128 bits.

Steps
Let us assume that X is a data-stream that can be split into *n* parts as following:
$X = X_1 \| \| X_2 \| \| \cdots \| \| X_{n-1} \| \| X_n$
Here, the lengths of $X_1, X_2, \cdots, X_{n-1}$ all equal to 128 bits, and the length of X_n is *t* (0<=*t*<=127).
If $t = 0$, then
TEXT-PADDING(X) = X;
If $0 < t <= 127$, define a 128-bit block, X_n^*, and let $X_n^* = X_n \| \| 0^{128-t}$, then
TEXT-PADDING(X) = $X_1 \| \| X_2 \| \| \cdots \| \| X_{n-1} \| \| X_n^*$ = $X \| \| 0^{128-t}$

19.6.2 Endianness

Under the DMA-AES working mode, the transmission of source data and result data for AES accelerator is solely controlled by DMA. Therefore, the AES accelerator cannot control the Endianness of the source data and result data, but does have requirement on how these data should be stored in memory and on the length of the data.

For example, let us assume DMA needs to write the following data into memory at address 0x0280.

- Data represented in hexadecimal:
  - 000102030405060708090A0B0C0D0E0F101112131415161718191A1B1C1D1E1F20
- Data Length:
  - Equals to 2 blocks.

Then, this data will be stored in memory as shown in Table 19.6-4 below.

Table 19.6-4. Text Endianness for DMA-AES

Address	Byte	Address	Byte	Address	Byte	Address	Byte
0x0280	0x01	0x0281	0x02	0x0282	0x03	0x0283	0x04
0x0284	0x05	0x0285	0x06	0x0286	0x07	0x0287	0x08
0x0288	0x09	0x0289	0x0A	0x028A	0x0B	0x028B	0x0C
0x028C	0x0D	0x028D	0x0E	0x028E	0x0F	0x028F	0x10
0x0290	0x11	0x0291	0x12	0x0292	0x13	0x0293	0x14
0x0294	0x15	0x0295	0x16	0x0296	0x17	0x0297	0x18
0x0298	0x19	0x0299	0x1A	0x029A	0x1B	0x029B	0x1C
0x029C	0x1D	0x029D	0x1E	0x029E	0x1F	0x029F	0x20

19.6.3 Standard Incrementing Function

AES accelerator provides two Standard Incrementing Functions for the CTR block operation, which are INC_{32} and INC_{128} Standard Incrementing Functions. By setting the AES_INC_SEL_REG register to 0 or 1, users can choose the INC_{32} or INC_{128} functions respectively. For details on the Standard Incrementing Function, please see Chapter B.1 *The Standard Incrementing Function* in NIST SP 800-38A.

19.6.4 Block Number

Register AES_BLOCK_NUM_REG stores the Block Number of plaintext P or ciphertext C. The length of this register equals to length(**TEXT-PADDING**(P))/128 or length(**TEXT-PADDING**(C))/128. The AES accelerator only uses this register when working in the DMA-AES mode.

19.6.5 Initialization Vector

AES_IV_MEM is a 16-byte memory, which is only available for AES accelerator working in block operations. For CBC/OFB/CFB8/CFB128 operations, the AES_IV_MEM memory stores the Initialization Vector (IV). For the CTR operation, the AES_IV_MEM memory stores the Initial Counter Block (ICB).

Both IV and ICB are 128-bit strings, which can be divided into Byte0, Byte1, Byte2 ··· Byte15 (from left to right). AES_IV_MEM stores data following the Endianness pattern presented in Table 19.6-4, i.e., the most significant (i.e., left-most) byte Byte0 is stored at the lowest address while the least significant (i.e., right-most) byte Byte15 at the highest address.

For more details on IV and ICB, please refer to NIST SP 800-38A.

19.6.6 Block Operation Process

1. Select one of DMA channels to connect with AES, configure the DMA chained list, and then start DMA. For details, please refer to Chapter 3 *GDMA Controller (GDMA)*.
2. Initialize the AES accelerator-related registers:
   - Write 1 to the AES_DMA_ENABLE_REG register.
   - Configure the AES_INT_ENA_REG register to enable or disable the interrupt function.
   - Initialize registers AES_MODE_REG and AES_KEY_*n*_REG.
   - Select block cipher mode by configuring the AES_BLOCK_MODE_REG register. For details, see Table 19.6-1.
   - Initialize the AES_BLOCK_NUM_REG register. For details, see Section 19.6.4.
   - Initialize the AES_INC_SEL_REG register (only needed when AES accelerator is working under CTR block operation).
   - Initialize the AES_IV_MEM memory (This is always needed except for ECB block operation).
3. Start operation by writing 1 to the AES_TRIGGER_REG register.
4. Wait for the completion of computation, which happens when the content of AES_STATE_REG becomes 2 or the AES interrupt occurs.
5. Check if DMA completes data transmission from AES to memory. At this time, DMA had already written the result data in memory, which can be accessed directly. For details on DMA, please refer to Chapter 3 *GDMA Controller (GDMA)*.
6. Clear interrupt by writing 1 to the AES_INT_CLR_REG register, if any AES interrupt occurred during the computation.
7. Release the AES accelerator by writing 1 to the AES_DMA_EXIT_REG register. After this, the content of the AES_STATE_REG register becomes 0. Note that, you can release DMA earlier, but only after Step 4 is completed.

19.7 Anti-Attack Pseudo-Round Function

To enhance anti-attack performance, ESP32-H2's AES incorporates a pseudo-round function.

When the pseudo-round function is enabled, AES randomly inserts pseudo-rounds before and after the original operation rounds. During a pseudo-round, AES uses a pseudo-key, automatically generated by the hardware, to perform a round of dummy operations. These operations do not affect the original AES operation results.

Users can enable the pseudo-round function by setting AES_PSEUDO_EN to 1. The pseudo-round configuration is as follows.

Total number of pseudo-rounds

The total number of pseudo-rounds randomly inserted into an AES operation is controlled by the following register fields:

- pseudo_base: Equal to the value of AES_PSEUDO_BASE, which configures the basic number of pseudo-rounds.
- pseudo_inc: Equal to the value of AES_PSEUDO_INC, which configures the random incremental number of pseudo-rounds.

The total number of pseudo-rounds will randomly fall into the range:

$$[pseudo\_base, pseudo\_base + (2^{pseudo\_inc} - 1)]$$

Random number update frequency

Users can set the frequency of random key updates in the pseudo-round function by configuring AES_PSEUDO_RNG_CNT. A higher value results in a higher update frequency. It is generally recommended to set this value to the maximum of 7.

19.8 Memory Summary

The addresses in this section are relative to the AES accelerator base address provided in Table 4.3-2 in Chapter 4 *System and Memory*.

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

Name	Description	Size (byte)	Starting Address	Ending Address	Access
AES_IV_MEM	Memory IV	16 bytes	0x0050	0x005F	R/W

19.9 Register Summary

The addresses in this section are relative to the AES accelerator base address provided in Table 4.3-2 in Chapter 4 *System and Memory*.

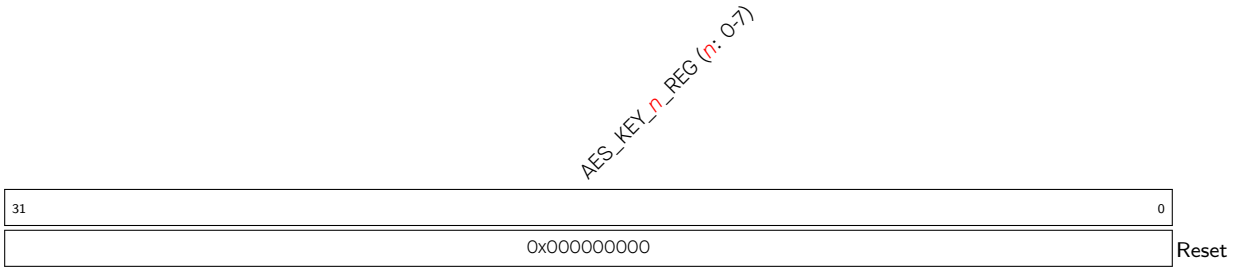
The abbreviations given in Column **Access** are explained in Section *Access Types for Registers*.

Name	Description	Address	Access
Key Registers			
AES_KEY_0_REG	AES key data register 0	0x0000	R/W
AES_KEY_1_REG	AES key data register 1	0x0004	R/W
AES_KEY_2_REG	AES key data register 2	0x0008	R/W
AES_KEY_3_REG	AES key data register 3	0x000C	R/W
AES_KEY_4_REG	AES key data register 4	0x0010	R/W
AES_KEY_5_REG	AES key data register 5	0x0014	R/W
AES_KEY_6_REG	AES key data register 6	0x0018	R/W
AES_KEY_7_REG	AES key data register 7	0x001C	R/W
TEXT_IN Registers			
AES_TEXT_IN_0_REG	Source text data register 0	0x0020	R/W
AES_TEXT_IN_1_REG	Source text data register 1	0x0024	R/W
AES_TEXT_IN_2_REG	Source text data register 2	0x0028	R/W
AES_TEXT_IN_3_REG	Source text data register 3	0x002C	R/W
TEXT_OUT Registers			
AES_TEXT_OUT_0_REG	Result text data register 0	0x0030	RO
AES_TEXT_OUT_1_REG	Result text data register 1	0x0034	RO
AES_TEXT_OUT_2_REG	Result text data register 2	0x0038	RO
AES_TEXT_OUT_3_REG	Result text data register 3	0x003C	RO
Control / Configuration Registers			
AES_MODE_REG	Defines key length and encryption / decryption	0x0040	R/W
AES_DMA_ENABLE_REG	Selects the working mode of the AES accelerator	0x0090	R/W
AES_BLOCK_MODE_REG	Defines the block cipher mode	0x0094	R/W
AES_BLOCK_NUM_REG	Block number configuration register	0x0098	R/W
AES_INC_SEL_REG	Standard incrementing function register	0x009C	R/W
AES_TRIGGER_REG	Operation start controlling register	0x0048	WT
AES_DMA_EXIT_REG	Operation exit controlling register	0x00B8	WO
AES_PSEUDO_REG	Pseudo-round function configuration register	0x00D0	R/W
Status Register			
AES_STATE_REG	Operation status register	0x004C	RO
Interrupt Registers			
AES_INT_CLR_REG	DMA-AES interrupt clear register	0x00AC	WT
AES_INT_ENA_REG	DMA-AES interrupt enable register	0x00B0	R/W

19.10 Registers

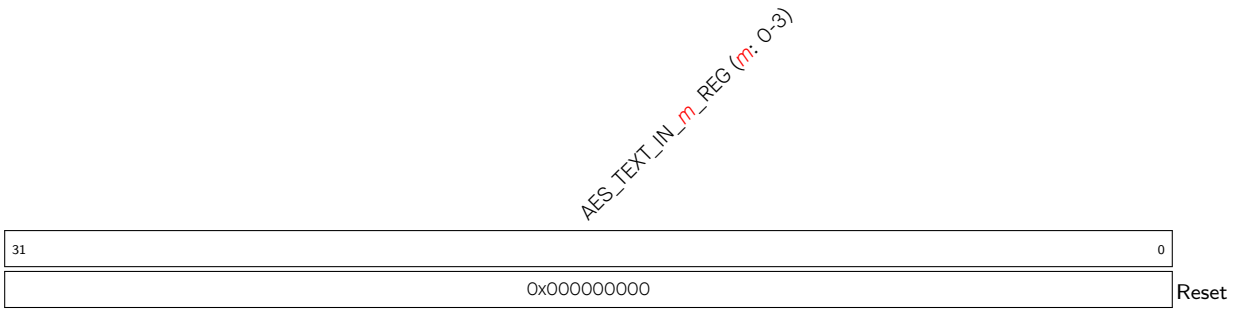
The addresses in this section are relative to the AES accelerator base address provided in Table 4.3-2 in Chapter 4 *System and Memory*.

Register 19.1. AES_KEY_*n*_REG (*n*: 0-7) (0x0000+4n*)**



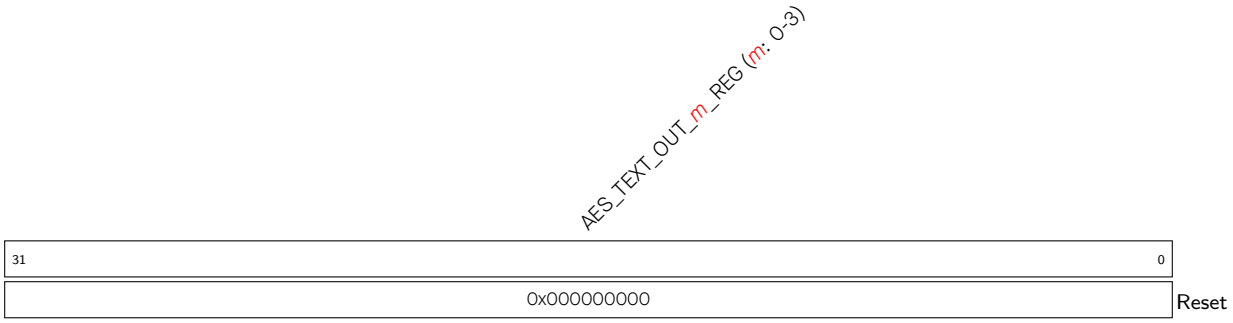
AES_KEY_*n*_REG (*n*: 0-7) Represents AES key data. (R/W)

Register 19.2. AES_TEXT_IN_*m*_REG (*m*: 0-3) (0x0020+4m*)**



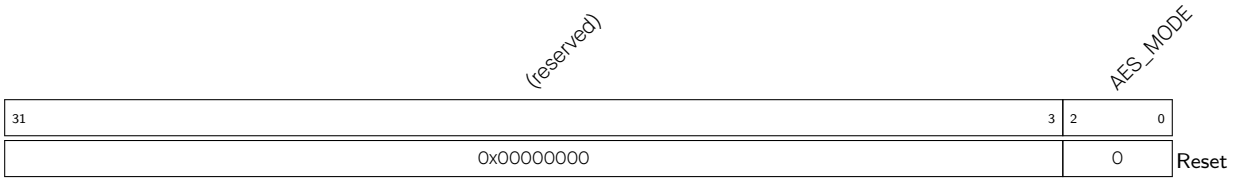
AES_TEXT_IN_*m*_REG (*m*: 0-3) Represents the source text data when the AES accelerator operates in the Typical AES working mode. (R/W)

Register 19.3. AES_TEXT_OUT_*m*_REG (*m*: 0-3) (0x0030+4m*)**



AES_TEXT_OUT_*m*_REG (*m*: 0-3) Represents the result text data when the AES accelerator operates in the Typical AES working mode. (RO)

Register 19.4. AES_MODE_REG (0x0040)



AES_MODE Configures the key length and encryption/decryption of the AES accelerator.

0: AES-128 encryption

1: Reserved

2: AES-256 encryption

3: Reserved

4: AES-128 decryption

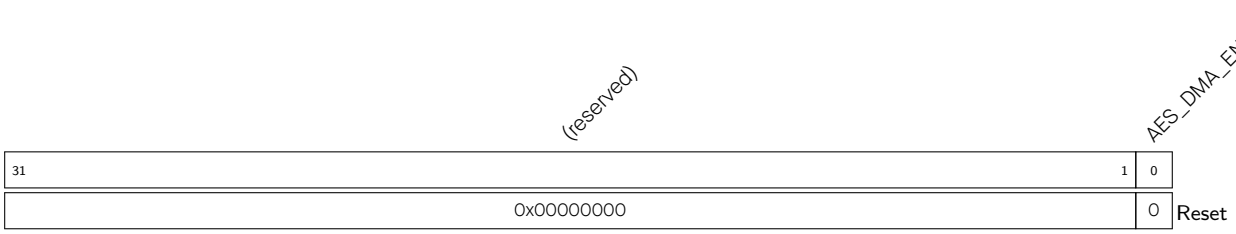
5: Reserved

6: AES-256 decryption

7: Reserved

(R/W)

Register 19.5. AES_DMA_ENABLE_REG (0x0090)



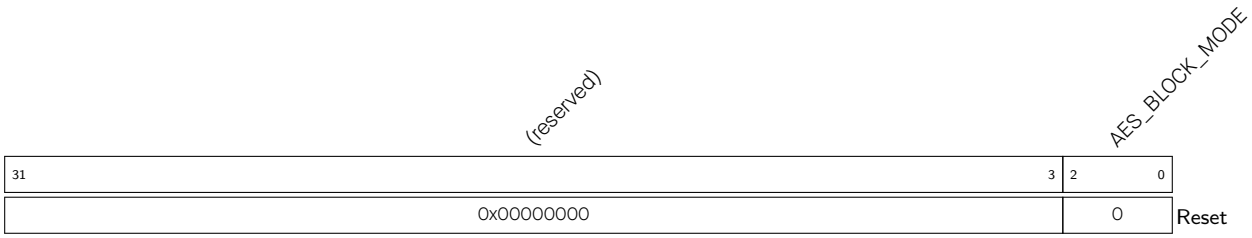
AES_DMA_ENABLE Configures the working mode of the AES accelerator.

0: Typical AES

1: DMA-AES

(R/W)

Register 19.6. AES_BLOCK_MODE_REG (0x0094)



AES_BLOCK_MODE Configures the block cipher mode of the AES accelerator operating under the DMA-AES working mode.

0: ECB (Electronic Code Block)

1: CBC (Cipher Block Chaining)

2: OFB (Output FeedBack)

3: CTR (Counter)

4: CFB8 (8-bit Cipher FeedBack)

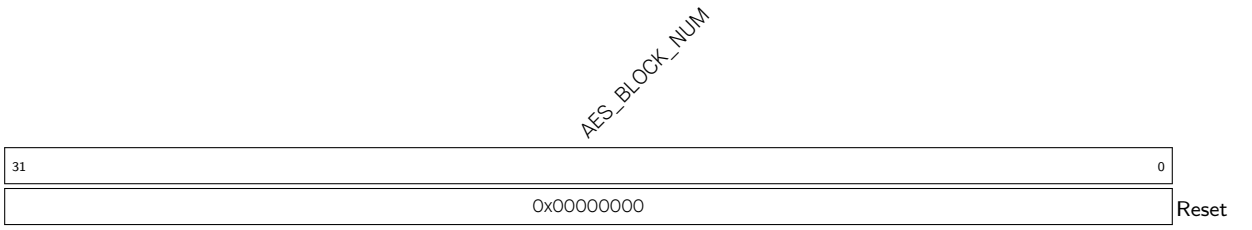
5: CFB128 (128-bit Cipher FeedBack)

6: Reserved

7: Reserved

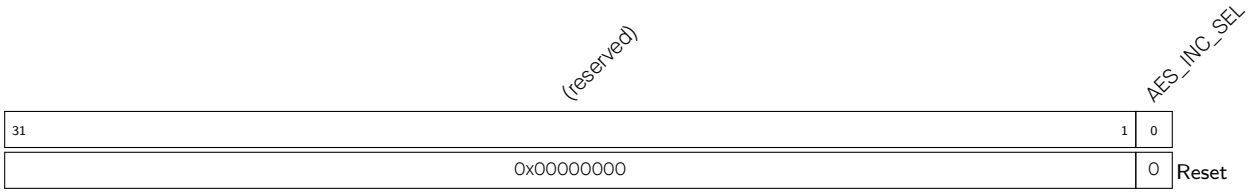
(R/W)

Register 19.7. AES_BLOCK_NUM_REG (0x0098)



AES_BLOCK_NUM Represents the Block Number of plaintext or ciphertext when the AES accelerator operates under the DMA-AES working mode. For details, see Section 19.6.4. (R/W)

Register 19.8. AES_INC_SEL_REG (0x009C)



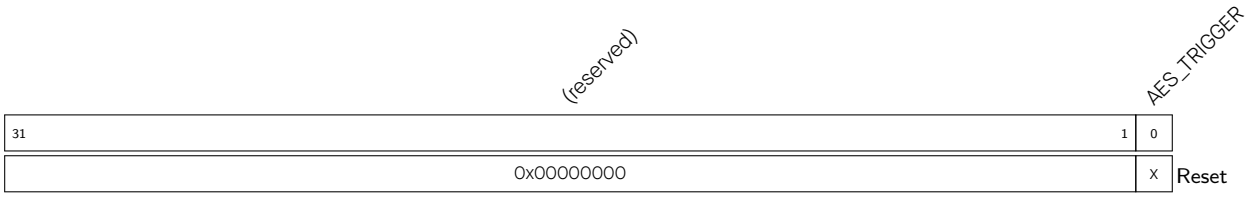
AES_INC_SEL Configures the Standard Incrementing Function for CTR block operation.

0: INC_{32}

1: INC_{128}

(R/W)

Register 19.9. AES_TRIGGER_REG (0x0048)



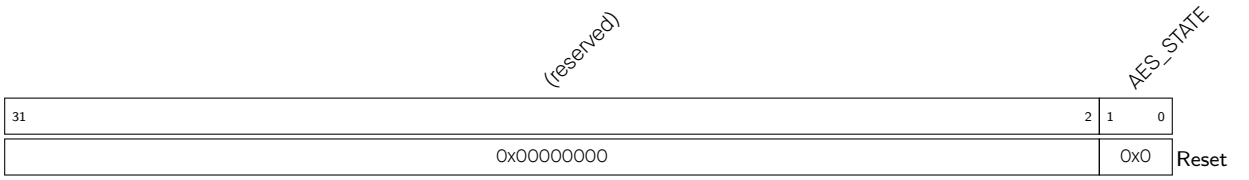
AES_TRIGGER Configures whether to start AES operation.

0: No effect

1: Start

(WT)

Register 19.10. AES_STATE_REG (0x004C)



AES_STATE Represents the working status of the AES accelerator.

In Typical AES working mode:

0: IDLE

1: WORK

2: No effect

3: No effect

In DMA-AES working mode:

0: IDLE

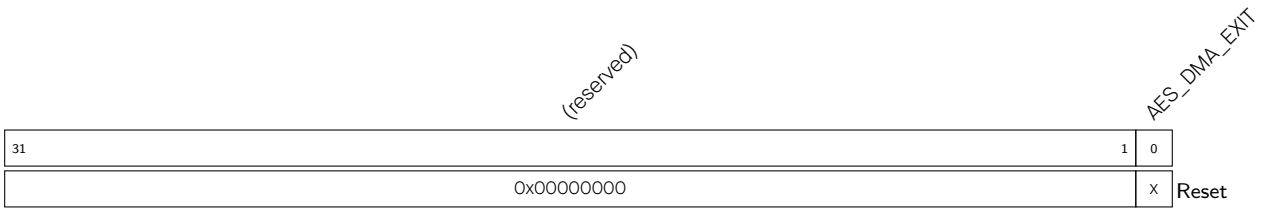
1: WORK

2: DONE

3: No effect

(RO)

Register 19.11. AES_DMA_EXIT_REG (0x00B8)



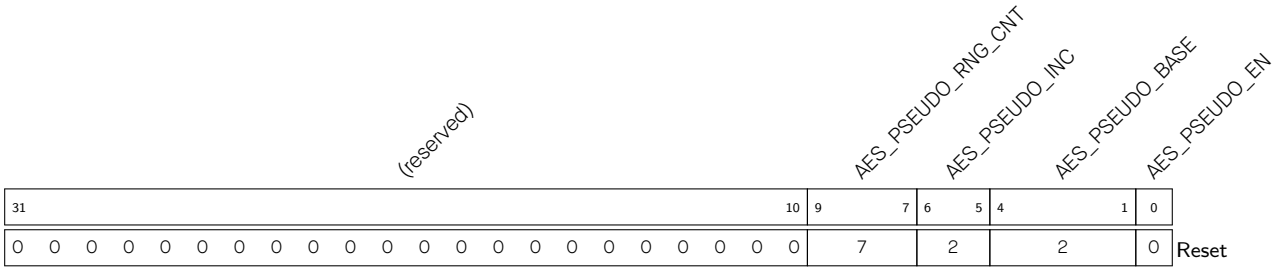
AES_DMA_EXIT Configures whether to exit AES operation.

0: No effect

1: Exit

Only valid for DMA-AES operation. (WO)

Register 19.12. AES_PSEUDO_REG (0x00D0)



AES_PSEUDO_EN Configures whether to enable the pseudo-round function of AES.

0: Disabled

1: Enabled

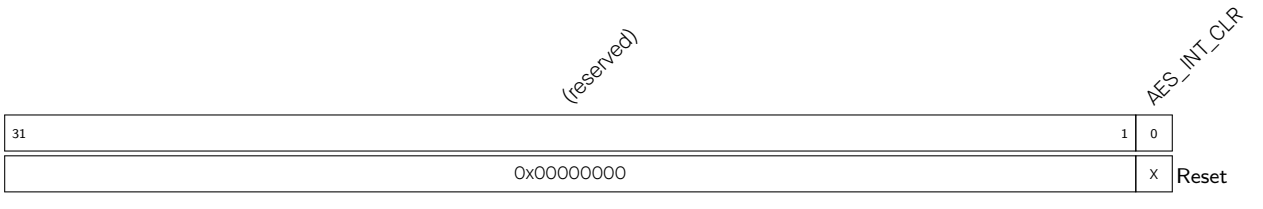
(R/W)

AES_PSEUDO_BASE Configures the basic number of pseudo-rounds. (R/W)

AES_PSEUDO_INC Configures the random incremental number of pseudo-rounds. (R/W)

AES_PSEUDO_RNG_CNT Configures the frequency of pseudo-key updates in the pseudo-round function. (R/W)

Register 19.13. AES_INT_CLR_REG (0x00AC)



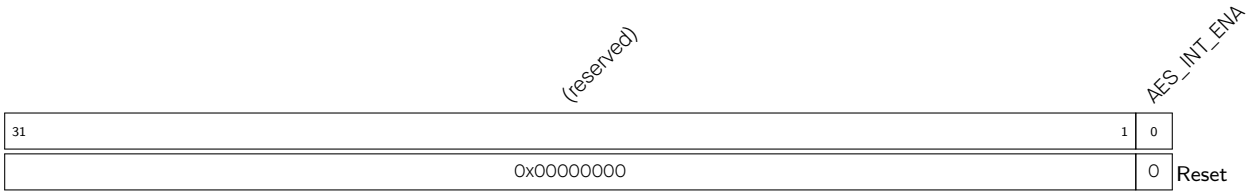
AES_INT_CLR Configures whether to clear AES interrupt.

0: No effect

1: Clear

(WT)

Register 19.14. AES_INT_ENA_REG (0x00B0)



AES_INT_ENA Configures whether to enable AES interrupt.

0: Disable

1: Enable

(R/W)

Chapter 20

ECC Accelerator (ECC)

20.1 Introduction

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-H2's ECC accelerator can complete various calculations based on different elliptic curves, thus accelerating the ECC algorithm and ECC-derived algorithms (such as ECDSA).

20.2 Features

ESP32-H2'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
- Interrupt upon completion of calculation

20.3 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.

20.3.1 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.

20.3.2 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:

$$X = x$$

$$Y = y$$

$$Z = 1$$

20.3.3 Memory Blocks

ECC's memory blocks store the input and output data of the ECC operation.

Table 20.3-1. ECC Accelerator Memory Blocks

Memory Block	Size (byte)	Starting Address*	Ending Address *	Access
ECC_MULT_Mem_k	32	0x100	0x11F	R/W
ECC_MULT_Mem_Px	32	0x120	0x13F	R/W
ECC_MULT_Mem_Py	32	0x140	0x15F	R/W
ECC_MULT_Mem_Qx	32	0x160	0x17F	R/W
ECC_MULT_Mem_Qy	32	0x180	0x19F	R/W
ECC_MULT_Mem_Qz	32	0x1A0	0x1BF	R/W

* Address offset related to the ECC accelerator base address is provided in Table 4.3-2 in Chapter 4 *System and Memory*.

20.3.4 Data and Data Block

ESP32-H2'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:

$$D[255:0] = D[7][31:0], D[6][31:0], D[5][31:0], D[4][31:0], D[3][31:0], D[2][31:0], D[1][31:0], D[0][31:0]$$

20.3.5 Writing Data

Write data means writing data to an ECC memory block and using this data as the input to the ECC algorithm. To be specific, write 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.

20.3.6 Reading Data

Read data means reading data from the starting address of an ECC memory block and using this data as the output from the ECC algorithm. To be specific, read data from an ECC memory block means reading $D[n][31:0](n = 0, 1, \cdots, 7)$ from the "starting address of this ECC memory block + $4 \times n$" successively:

- read $D[0]$ from "starting address"
- read $D[1]$ from "starting address + 4"
- $\cdots$
- read $D[7]$ from "starting address + 28"

Note:
When the key size of 192 bits is used, only read the low 192 bits (6 blocks) of data.

20.3.7 Standard Calculation and Jacobian Calculation

ESP32-H2'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.

20.4 Function Description

20.4.1 Key Size

ESP32-H2'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 20.4-1 below.

Table 20.4-1. 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.

20.4.2 Working Modes

ESP32-H2'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 20.4-2.

Table 20.4-2. 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.

20.4.2.1 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.

20.4.2.2 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.

20.4.2.3 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.

20.4.2.4 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.

20.4.2.5 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.

20.4.2.6 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.

20.4.2.7 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.

20.4.2.8 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.

20.4.2.9 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.

20.4.2.10 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.

20.4.2.11 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.

20.4.3 Enhancing Anti-Attack Performance

When performing different types of point multiplication calculations, i.e., mode 0, 3, 4, and 7, ESP32-H2'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.

20.5 Clock and Reset

ESP32-H2'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 7 *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.

20.6 Interrupts

ESP32-H2's ECC accelerator can generate one interrupt signal ECC_INTR and send it to Interrupt Matrix.

Note:
Each interrupt signal is generated by any of its interrupt sources, i.e., any of its interrupt sources triggered can generate the interrupt signal.

ECC_INTR has only one interrupt source, i.e., ECC_MULT_CALC_DONE_INT, which is triggered on the completion of an ECC calculation. This ECC_MULT_CALC_DONE_INT interrupt source is configured by the following registers:

- 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/disabling the ECC_MULT_CALC_DONE_INT_RAW bit via ECC_MULT_CALC_DONE_INT_ENA.
- ECC_MULT_CALC_DONE_INT_ENA: enables/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, bits ECC_MULT_CALC_DONE_INT_RAW and ECC_MULT_CALC_DONE_INT_ST will be cleared.

20.7 Programming Procedures

The programming procedure for configuring ECC is described below:

1. Configure the ECC clock and reset. Refer to Section 20.5 for detailed information.
2. Select the key size and working mode as described in Section 20.4.
3. Enable the ECC_MULT_CALC_DONE_INT interrupt as described in Section 20.6.
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 20.4.

20.8 Register Summary

The addresses in this section are relative to ECC Accelerator base address provided in Table 4.3-2 in Chapter 4 *System and Memory*.

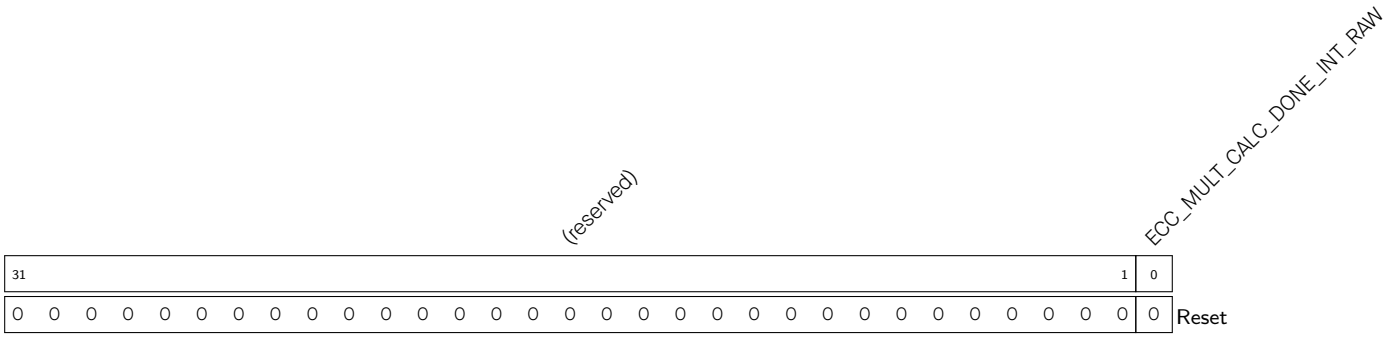
The abbreviations given in Column **Access** are explained in Section Access Types for Registers.

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

20.9 Registers

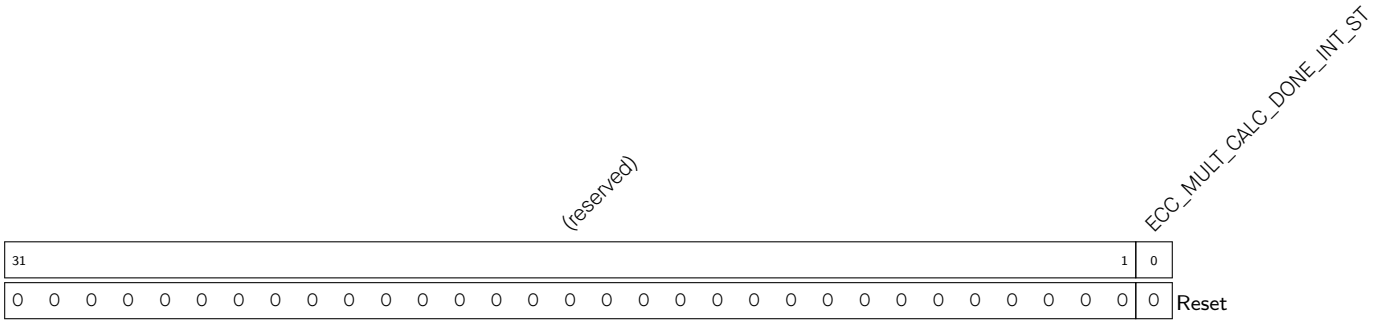
The addresses in this section are relative to ECC Accelerator base address provided in Table 4.3-2 in Chapter 4 *System and Memory*.

Register 20.1. ECC_MULT_INT_RAW_REG (0x000C)



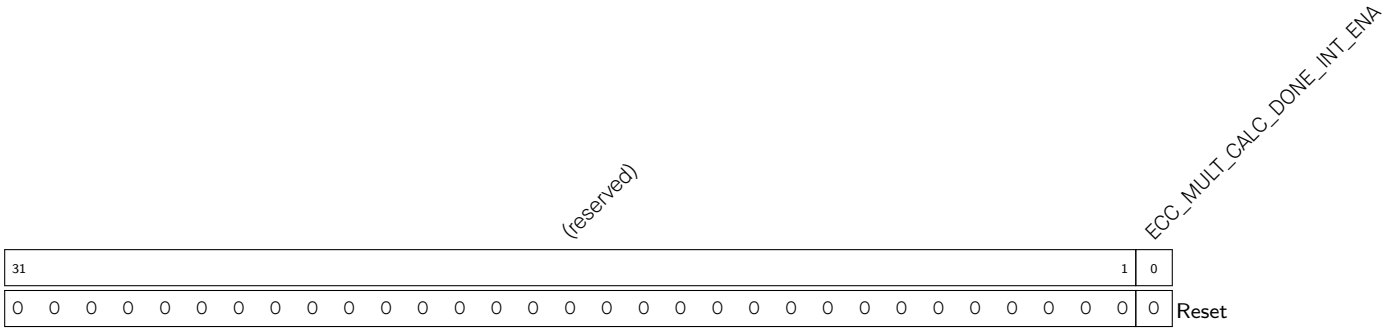
ECC_MULT_CALC_DONE_INT_RAW The raw interrupt status of the ECC_MULT_CALC_DONE_INT interrupt. (R/SS/WTC)

Register 20.2. ECC_MULT_INT_ST_REG (0x0010)



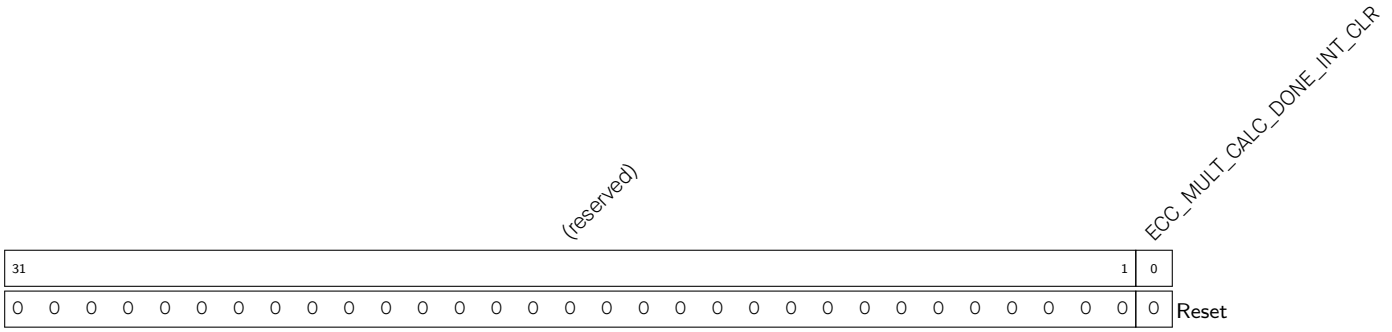
ECC_MULT_CALC_DONE_INT_ST The masked interrupt status of the ECC_MULT_CALC_DONE_INT interrupt. (RO)

Register 20.3. 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 20.4. 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 20.5. ECC_MULT_CONF_REG (0x001C)

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	
31	30	29	28 … 9	8	7 … 4	3	2	1	0	
1	0	0	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	0	0	0	0	0	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 20.5. ECC_MULT_CONF_REG (0x001C)

Continued from the previous page...

ECC_MULT_SECURITY_MODE Configures whether to enable high-security mode when calculating point multiplication. Only valid in working modes 0, 3, 4, and 7.

0: Disable

1: Enable

(R/W)

ECC_MULT_VERIFICATION_RESULT Represents the verification result of the ECC accelerator, valid only when the calculation is done.

0: Verification failed

1: Verification passed

(R/SS)

ECC_MULT_CLK_EN Configures whether to force on register clock gate.

0: No effect

1: Force on

(R/W)

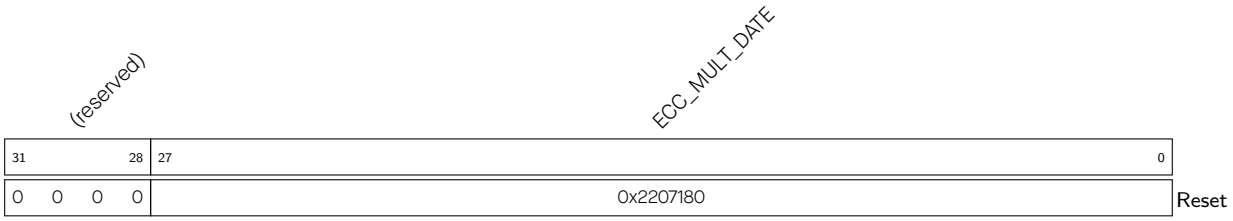
ECC_MULT_MEM_CLOCK_GATE_FORCE_ON Configures whether to force on ECC memory clock gate.

0: No effect

1: Force on

(R/W)

Register 20.6. ECC_MULT_DATE_REG (0x00FC)



ECC_MULT_DATE Version control register. (R/W)

Chapter 21

HMAC Accelerator (HMAC)

The Hash-based Message Authentication Code (HMAC) module computes Message Authentication Codes (MACs) using Hash algorithm SHA-256 and keys as described in RFC 2104. The 256-bit HMAC key is stored in an eFuse key block and can be set as read-protected, i. e., the key is not accessible from outside the HMAC accelerator.

21.1 Main Features

- Standard HMAC-SHA-256 algorithm
- Hash result only accessible by configurable hardware peripheral (in downstream mode)
- Compatibility with challenge-response authentication algorithm
- Generates required keys for the Digital Signature Algorithm (DSA) peripheral (in downstream mode)
- Re-enables soft-disabled JTAG (in downstream mode)

21.2 Functional Description

The HMAC module operates in two modes: upstream mode and downstream mode. In upstream mode, users provide the HMAC message and read back the calculation result. In downstream mode, the HMAC module is used as a Key Derivation Function (KDF) for other internal hardware. For instance, the JTAG can be temporarily disabled by burning odd number bits of EFUSE_SOFT_DIS_JTAG in eFuse. In this case, users can temporarily re-enable JTAG using the HMAC module in downstream mode.

After the reset signal being released, the HMAC module will check whether the DSA key exists in the eFuse. If the key exists, the HMAC module will enter downstream digital signature algorithm mode and finish the DSA key calculation automatically.

21.2.1 Upstream Mode

Common use cases for the upstream mode are challenge-response protocols supporting HMAC-SHA-256. Assume the two entities in the challenge-response protocol are A and B respectively, and the data message they expect to exchange is M. The general authentication process of this protocol is as follows:

- A calculates a unique random number M.
- A sends M to B.
- B calculates the HMAC (through M and KEY) and sends the result to A.
- A calculates the HMAC (through M and KEY) internally.

- A compares the two results. If the results are the same, then the identity of B is authenticated.

To calculate the HMAC value, users should perform the following steps:

1. Initialize the HMAC module, and enter upstream mode.
2. Write the correctly padded message to the HMAC, one block at a time.
3. Read back the result from HMAC.

For details of this process, please see Section 21.2.5.

21.2.2 Downstream JTAG Enable Mode

JTAG debugging can be disabled in a way which allows later re-enabling using the HMAC module. The HMAC module will expect the user to supply the HMAC result for one of the eFuse keys. The HMAC module will check whether the supplied HMAC matches the one calculated from the chosen key. If both HMACs are the same, JTAG will be enabled until the user calls the HMAC module to clear the results and consequently disable JTAG again.

There are two parameters in eFuse memory to disable JTAG: EFUSE_DIS_PAD_JTAG and EFUSE_SOFT_DIS_JTAG. Write 1 to EFUSE_DIS_PAD_JTAG to disable JTAG permanently, and write odd numbers of 1 to EFUSE_SOFT_DIS_JTAG to disable JTAG temporarily. For more details, please see Chapter 5 *eFuse Controller (EFUSE)*. After bit EFUSE_SOFT_DIS_JTAG is set, the key to re-enable JTAG can be calculated in HMAC module's downstream mode. JTAG is re-enabled when the result configured by the user is the same as the HMAC result.

To re-enable JTAG, users should perform the following steps:

1. Enable the HMAC module by initializing clock and reset signals of HMAC, and enter downstream JTAG enable mode by configuring HMAC_SET_PARA_PURPOSE_REG. Then, wait for the calculation to complete. Please see Section 21.2.5 for more details.
2. Write 1 to the HMAC_SOFT_JTAG_CTRL_REG register to enter JTAG re-enable compare mode.
3. Write the 256-bit HMAC value to register HMAC_WR_JTAG_REG. This value is obtained by performing a local HMAC calculation from the 32-byte 0x00 using SHA-256 and the generated key. It needs to be written 8 times and 32-bit each time in big-endian word order.
4. If the HMAC result matches the value that users calculated locally, then JTAG is re-enabled. Otherwise, JTAG remains disabled.
5. After writing 1 to HMAC_SET_INVALIDATE_JTAG_REG or resetting the chip, JTAG will be disabled. If users want to re-enable JTAG again, they need to repeat the above steps again.

21.2.3 Downstream RSA_DS Mode

The RSA Digital Signature (RSA_DS) peripheral encrypts its parameters using the AES-CBC algorithm. The HMAC module is used as a Key Derivation Function (KDF) to derive the AES key to decrypt these parameters (parameter decryption key). The key used for the HMAC as KDF is stored in one of the eFuse key blocks.

Before starting the RSA_DS peripheral, users need to obtain the parameter decryption key for the RSA_DS peripheral through HMAC calculation. For more information, please see Chapter 24 *RSA Digital Signature*

Peripheral (RSA_DS). After the chip is powered on, the HMAC module will check whether the key required to calculate the parameter decryption key has been burned in the eFuse block. If the key has been burned, HMAC module will automatically enter the downstream RSA_DS mode and complete the HMAC calculation based on the chosen key.

21.2.4 HMAC eFuse Configuration

Each HMAC key burned into an eFuse block has a key purpose, specifying for which functionality the key can be used. The HMAC module will not accept a key with a non-matching purpose for any functionality. The HMAC module provides three different functionalities: re-enabling JTAG, RSA_DS KDF in downstream mode, and pure HMAC calculation in upstream mode. For each functionality, there exists a corresponding key purpose, listed in Table 21.2-1. Additionally, another purpose specifies a key which may be used for re-enabling JTAG as well as for serving as RSA_DS KDF.

Before enabling HMAC to do calculations, user should make sure the key to be used has been burned in eFuse by reading the registers EFUSE_KEY_PURPOSE_x (We have a total of 6 keys in eFuse, so the value of x is 0 ~ 5. Among which, EFUSE_KEY_PURPOSE_0 ~ EFUSE_KEY_PURPOSE_1 belong to the register EFUSE_RD_REPEAT_DATA1_REG, and EFUSE_KEY_PURPOSE_2 ~ EFUSE_KEY_PURPOSE_5 belong to the register EFUSE_RD_REPEAT_DATA2_REG) from *5 eFuse Controller (EFUSE)*. Take upstream mode as an example, if there is no EFUSE_KEY_PURPOSE_HMAC_UP in EFUSE_KEY_PURPOSE_0 ~ 5, it means there is no key in eFuse that can be used for the HMAC upstream mode. Users can burn key to eFuse as follows:

1. Prepare a secret 256-bit HMAC key and burn the key to an empty eFuse block y. As there are 6 blocks for storing a key in eFuse and the numbers of those blocks range from 4 to 9, the value of y is 4 ~ 9. Hence, when talking about key0, it means eFuse block4. Then, program the purpose to EFUSE_KEY_PURPOSE_$(y-4)$. Take upstream mode as an example: after programming the key, the user should program EFUSE_KEY_PURPOSE_HMAC_UP (corresponding value is 6) to EFUSE_KEY_PURPOSE_$(y-4)$. Please see Chapter *5 eFuse Controller (EFUSE)* on how to program eFuse keys.
2. Configure this eFuse key block to be read protected, so that users cannot read its value. A copy of this key should be kept by any party who needs to verify this device.

Please note that the key whose purpose is EFUSE_KEY_PURPOSE_HMAC_DOWN_ALL can be used for both re-enabling JTAG or RSA_DS.

Table 21.2-1. HMAC Purposes and Configuration Value

Purpose	Mode	Value	Description
JTAG Re-enable	Downstream	6	EFUSE_KEY_PURPOSE_HMAC_DOWN_JTAG
RSA_DS KDF	Downstream	7	EFUSE_KEY_PURPOSE_HMAC_DOWN_DIGITAL_SIGNATURE
HMAC Calculation	Upstream	8	EFUSE_KEY_PURPOSE_HMAC_UP
Both JTAG Re-enable and RSA_DS KDF	Downstream	5	EFUSE_KEY_PURPOSE_HMAC_DOWN_ALL

Configure HMAC Purposes

The correct purpose has to be written to register HMAC_SET_PARA_PURPOSE_REG (see Section 21.2.5). If there is no valid value in eFuse purpose section, HMAC will terminate calculation.

Select eFuse Key Blocks

The eFuse controller provides six key blocks, i.e., KEY0 ~ 5. To select a particular KEYn for an HMAC calculation, write the key number n to register HMAC_SET_PARA_KEY_REG.

Note that the purpose of the key has also been programmed to eFuse memory. Only when the configured HMAC purpose matches the defined purpose of KEYn, the HMAC module will execute the configured calculation. Otherwise, it will return a matching error and stop the current calculation.

For example, suppose a user selects KEY3 for HMAC calculation, and the value programmed to EFUSE_KEY_PURPOSE_3 is 6 (EFUSE_KEY_PURPOSE_HMAC_DOWN_JTAG). Based on Table 21.2-1, KEY3 can be used to re-enable JTAG. If the value written to register HMAC_SET_PARA_PURPOSE_REG is also 6, then the HMAC module will start the process to re-enable JTAG.

21.2.5 HMAC Process (Detailed)

The process for users to call HMAC in ESP32-H2 is as follows:

1. Enable HMAC module:
   (a) Set the peripheral clock bits for HMAC and SHA peripherals in register PCR_HMAC_CLK_EN, and clear the corresponding peripheral reset bits in register PCR_HMAC_RST_EN. For information on those registers, please see Chapter 7 *Reset and Clock*.
   (b) Write 1 to register HMAC_SET_START_REG.
2. Configure HMAC keys and key purposes:
   (a) Write the key purpose m to register HMAC_SET_PARA_PURPOSE_REG. The possible key purpose values are shown in Table 21.2-1. For more information, please refer to Section 21.2.4.
   (b) Select KEYn in eFuse memory as the key by writing n (ranges from 0 to 5) to register HMAC_SET_PARA_KEY_REG. For more information, please refer to Section 21.2.4.
   (c) Write 1 to register HMAC_SET_PARA_FINISH_REG to complete the configuration.
   (d) Read register HMAC_QUERY_ERROR_REG. If its value is 1, it means the purpose of the selected block does not match the configured key purpose and the calculation will not proceed. If its value is 0, it means the purpose of the selected block matches the configured key purpose, and then the calculation can proceed.
   (e) When the value of HMAC_SET_PARA_PURPOSE_REG is not 8, it means the HMAC module is in downstream mode, proceed with step 3. When the value is 8, it means the HMAC module is in upstream mode, proceed with step 4.
3. Downstream mode:
   (a) Poll Status register HMAC_QUERY_BUSY_REG until it reads 0.
   (b) To clear the result and make further usage of the dependent hardware (JTAG or DSA) impossible, write 1 to either register HMAC_SET_INVALIDATE_JTAG_REG to clear the result generated by the JTAG key; or to register HMAC_SET_INVALIDATE_DS_REG to clear the result generated by DSA key. Afterwards, the HMAC Process needs to be restarted to re-enable any of the dependent peripherals.
4. Transmit message block Block_n ($n >= 1$) in upstream mode:
   (a) Poll Status register HMAC_QUERY_BUSY_REG until it reads 0.

(b) Write the 512-bit Block_n to register HMAC_WDATA0~15_REG. Write 1 to register HMAC_SET_MESSAGE_ONE_REG, to trigger the processing of this message block.

(c) Poll Status register HMAC_QUERY_BUSY_REG until it reads 0.

(d) Different message blocks will be generated, depending on whether the size of the to-be-processed message is a multiple of 512 bits.

- If the bit length of the message is a multiple of 512 bits, there are three possible options:
  i. If Block_n+1 exists, write 1 to register HMAC_SET_MESSAGE_ING_REG to make $n = n + 1$, and then jump to step 4.(b).
  ii. If Block_n is the last block of the message and users expects to apply SHA padding in hardware, write 1 to register HMAC_SET_MESSAGE_END_REG, and then jump to step 6.
  iii. If Block_n is the last block of the padded message and SHA padding has been applied by users, write 1 to register HMAC_SET_MESSAGE_PAD_REG, and then jump to step 5.
- If the bit length of the message is not a multiple of 512 bits, there are three possible options as follows. Note that in this case, the user is required to apply SHA padding to the message, after which the padded message length should be a multiple of 512 bits.
  i. If there is only one message block in total which has included all padding bits, write 1 to register HMAC_ONE_BLOCK_REG, and then jump to step 6.
  ii. If Block_n is the second last padded block, write 1 to register HMAC_SET_MESSAGE_PAD_REG, and then jump to step 5.
  iii. If Block_n is neither the last nor the second last message block, write 1 to register HMAC_SET_MESSAGE_ING_REG and define $n = n + 1$, and then jump to step 4.(b).

5. Apply SHA padding to message:

   (a) Users apply SHA padding to the last message block as described in Section 21.3.1, write this block to register HMAC_WDATA0~15_REG, and then write 1 to register HMAC_SET_MESSAGE_ONE_REG. Then the HMAC module will process this message block.

   (b) Jump to step 6.

6. Read hash result in upstream mode:

   (a) Poll Status register HMAC_QUERY_BUSY_REG until it reads 0.

   (b) Read hash result from register HMAC_RD_RESULT_*n*_REG (*n*: 0-7).

   (c) Write 1 to register HMAC_SET_RESULT_FINISH_REG to finish calculation. The result will be cleared at the same time.

   (d) Upstream mode operation is completed.

Note:
The SHA accelerator can be called directly, or used internally by the DSA module and the HMAC module. However, they can not share the hardware resources simultaneously. Therefore, the SHA module must not be called neither by the CPU nor by the DSA module when the HMAC module is in use.

21.3 HMAC Algorithm Details

21.3.1 Padding Bits

The HMAC module uses SHA-256 as hash algorithm. If the input message is not a multiple of 512 bits, the user must apply a SHA-256 padding algorithm in software. The SHA-256 padding algorithm is the same as described in Section *Padding the Message* of FIPS PUB 180-4. In downstream mode, users do not need to input any message or apply padding. The HMAC module uses a default 32-byte pattern of 0x00 for re-enabling JTAG and a 32-byte pattern of 0xff for deriving the AES key for the DSA module.

As shown in Figure 21.3-1, suppose the length of the unpadded message is m bits. Padding steps are as follows:

1. Append one bit of value "1" to the end of the unpadded message.
2. Append k bits of value "0", where k is the smallest non-negative number which satisfies $m + 1 + k \equiv 448 (mod 512)$.
3. Append a 64-bit integer value as a binary block. This block consists of the length of the unpadded message as a big-endian binary integer value m.

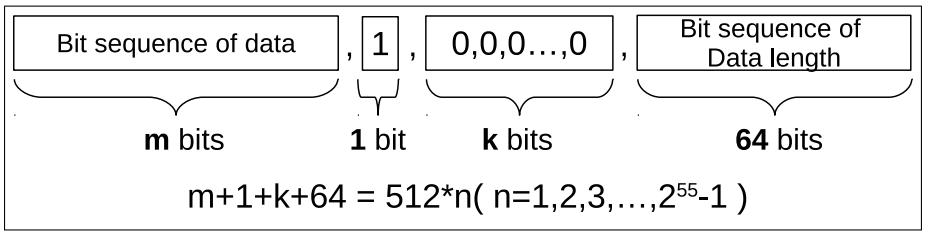


Figure 21.3-1. HMAC SHA-256 Padding Diagram

In upstream mode, if the length of the unpadded message is a multiple of 512 bits, users can configure hardware to apply SHA padding by writing 1 to HMAC_SET_MESSGAE_END_REG or do padding work themselves by writing 1 to HMAC_SET_MESSAGE_PAD_REG. If the length is not a multiple of 512 bits, SHA padding must be manually applied by the user. After the user prepared the padding data, they should complete the subsequent configuration according to the Section 21.2.5.

21.3.2 HMAC Algorithm Structure

The structure of the implemented algorithm in the HMAC module is shown in Figure 21.3-2. This is the standard HMAC algorithm as described in RFC 2104.

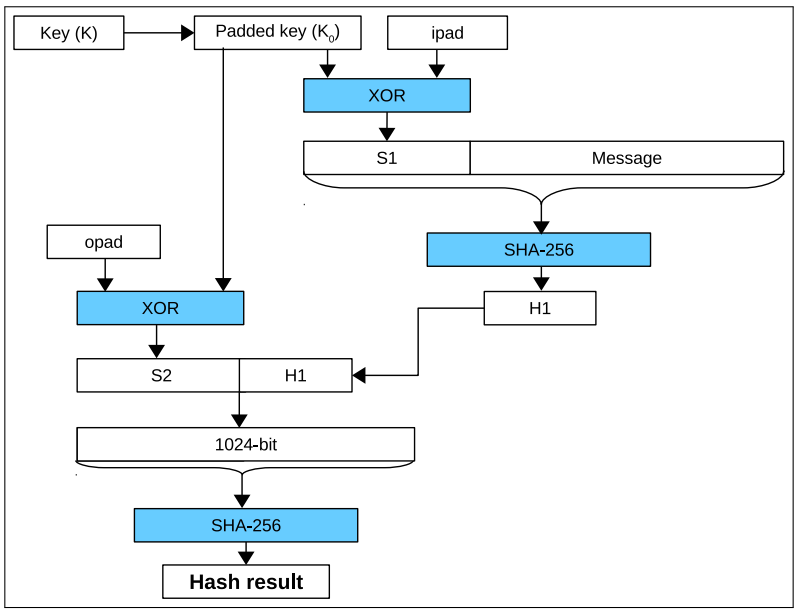


Figure 21.3-2. HMAC Structure Schematic Diagram

In Figure 21.3-2:

1. ipad is a 512-bit message block composed of 64 bytes of 0x36.
2. opad is a 512-bit message block composed of 64 bytes of 0x5c.

The HMAC module appends a 256-bit 0 sequence after the bit sequence of the 256-bit key K in order to get a 512-bit K_0. Then, the HMAC module XORs K_0 with ipad to get the 512-bit S1. Afterwards, the HMAC module appends the input message (multiple of 512 bits) after the 512-bit S1, and exercises the SHA-256 algorithm to get the 256-bit H1.

The HMAC module appends the 256-bit SHA-256 hash result H1 to the 512-bit S2 value, which is calculated using the XOR operation of K_0 and opad. A 768-bit sequence will be generated. Then, the HMAC module uses the SHA padding algorithm described in Section 21.3.1 to pad the 768-bit sequence to a 1024-bit sequence, and applies the SHA-256 algorithm to get the final hash result (256-bit).

21.4 Register Summary

The addresses in this section are relative to HMAC Accelerator base address provided in Table 4.3-2 in Chapter *4 System and Memory*.

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

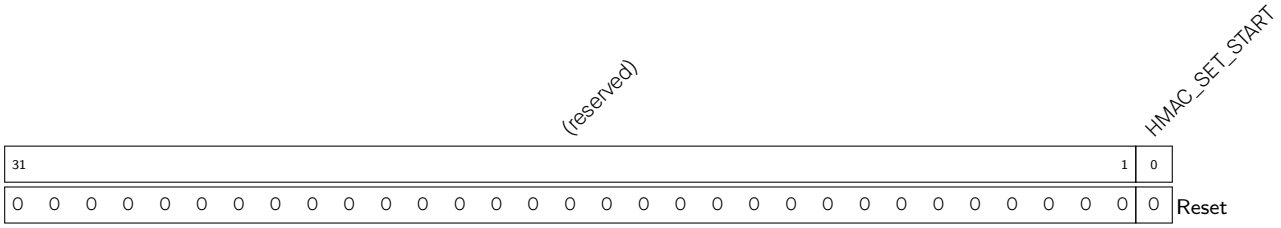
Name	Description	Address	Access
Control/Status Registers			
HMAC_SET_START_REG	HMAC start control register	0x0040	WO
HMAC_SET_PARA_FINISH_REG	HMAC configuration completion register	0x004C	WO
HMAC_SET_MESSAGE_ONE_REG	HMAC message control register	0x0050	WO
HMAC_SET_MESSAGE_ING_REG	HMAC message continue register	0x0054	WO
HMAC_SET_MESSAGE_END_REG	HMAC message end register	0x0058	WO
HMAC_SET_RESULT_FINISH_REG	HMAC result reading finish register	0x005C	WO
HMAC_SET_INVALIDATE_JTAG_REG	Invalidate JTAG result register	0x0060	WO
HMAC_SET_INVALIDATE_DS_REG	Invalidate RSA_DS result register	0x0064	WO
HMAC_QUERY_ERROR_REG	Stores matching results between keys generated by users and corresponding purposes	0x0068	RO
HMAC_QUERY_BUSY_REG	Busy state of HMAC module	0x006C	RO
HMAC_SET_MESSAGE_PAD_REG	Software padding register	0x00F0	WO
HMAC_ONE_BLOCK_REG	One block message register	0x00F4	WO
Configuration Registers			
HMAC_SET_PARA_PURPOSE_REG	HMAC parameter configuration register	0x0044	WO
HMAC_SET_PARA_KEY_REG	HMAC parameters configuration register	0x0048	WO
HMAC_SOFT_JTAG_CTRL_REG	Re-enable JTAG register 0	0x00F8	WO
HMAC_WR_JTAG_REG	Re-enable JTAG register 1	0x00FC	WO
HMAC Message Block			
HMAC_WR_MESSAGE_0_REG	Message register 0	0x0080	WO
HMAC_WR_MESSAGE_1_REG	Message register 1	0x0084	WO
HMAC_WR_MESSAGE_2_REG	Message register 2	0x0088	WO
HMAC_WR_MESSAGE_3_REG	Message register 3	0x008C	WO
HMAC_WR_MESSAGE_4_REG	Message register 4	0x0090	WO
HMAC_WR_MESSAGE_5_REG	Message register 5	0x0094	WO
HMAC_WR_MESSAGE_6_REG	Message register 6	0x0098	WO
HMAC_WR_MESSAGE_7_REG	Message register 7	0x009C	WO
HMAC_WR_MESSAGE_8_REG	Message register 8	0x00A0	WO
HMAC_WR_MESSAGE_9_REG	Message register 9	0x00A4	WO
HMAC_WR_MESSAGE_10_REG	Message register 10	0x00A8	WO
HMAC_WR_MESSAGE_11_REG	Message register 11	0x00AC	WO
HMAC_WR_MESSAGE_12_REG	Message register 12	0x00B0	WO
HMAC_WR_MESSAGE_13_REG	Message register 13	0x00B4	WO
HMAC_WR_MESSAGE_14_REG	Message register 14	0x00B8	WO
HMAC_WR_MESSAGE_15_REG	Message register 15	0x00BC	WO
HMAC Upstream Result			

Name	Description	Address	Access
HMAC_RD_RESULT_0_REG	Hash result register 0	0x00C0	RO
HMAC_RD_RESULT_1_REG	Hash result register 1	0x00C4	RO
HMAC_RD_RESULT_2_REG	Hash result register 2	0x00C8	RO
HMAC_RD_RESULT_3_REG	Hash result register 3	0x00CC	RO
HMAC_RD_RESULT_4_REG	Hash result register 4	0x00D0	RO
HMAC_RD_RESULT_5_REG	Hash result register 5	0x00D4	RO
HMAC_RD_RESULT_6_REG	Hash result register 6	0x00D8	RO
HMAC_RD_RESULT_7_REG	Hash result register 7	0x00DC	RO
Version Register			
HMAC_DATE_REG	Version control register	0x01FC	R/W

21.5 Registers

The addresses in this section are relative to HMAC Accelerator base address provided in Table 4.3-2 in Chapter *4 System and Memory*.

Register 21.1. HMAC_SET_START_REG (0x0040)



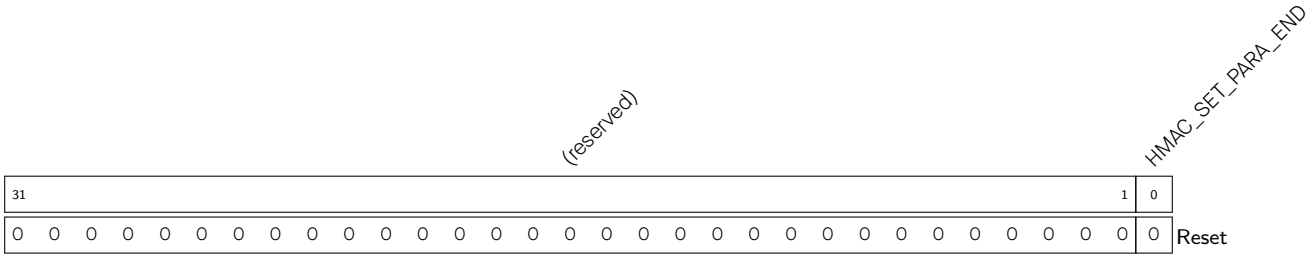
HMAC_SET_START Configures whether to enable HMAC.

0: Disable HMAC

1: Enable HMAC

(WO)

Register 21.2. HMAC_SET_PARA_FINISH_REG (0x004C)



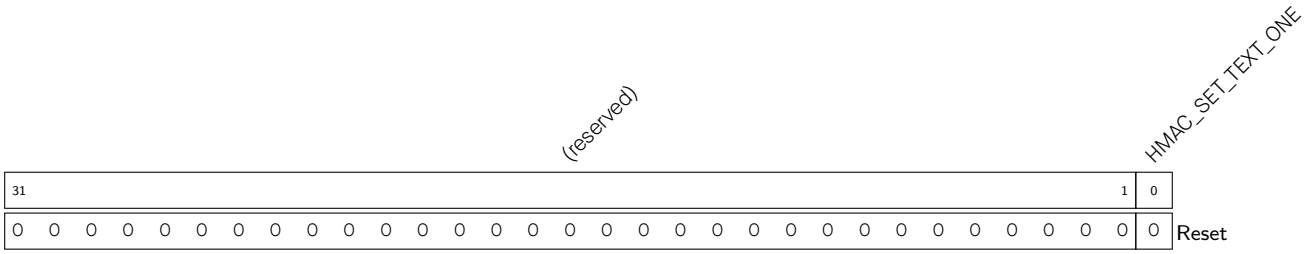
HMAC_SET_PARA_END Configures whether to finish HMAC configuration.

0: No effect

1: Finish configuration

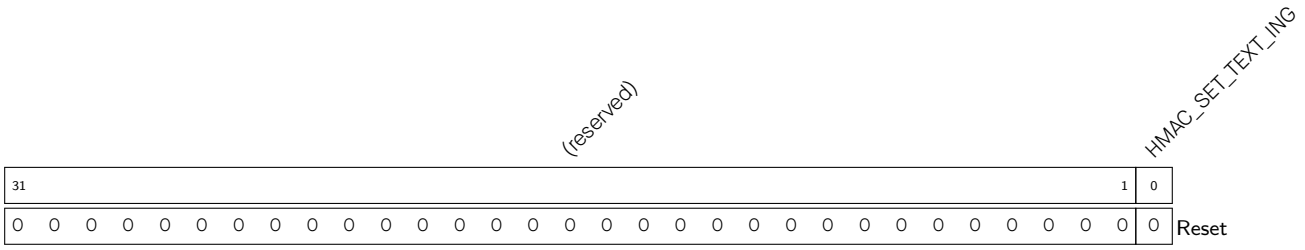
(WO)

Register 21.3. HMAC_SET_MESSAGE_CALC_BLOCK_REG (0x0050)



HMAC_SET_TEXT_ONE Calls SHA to calculate one message block. (WO)

Register 21.4. HMAC_SET_MESSAGE_ING_REG (0x0054)



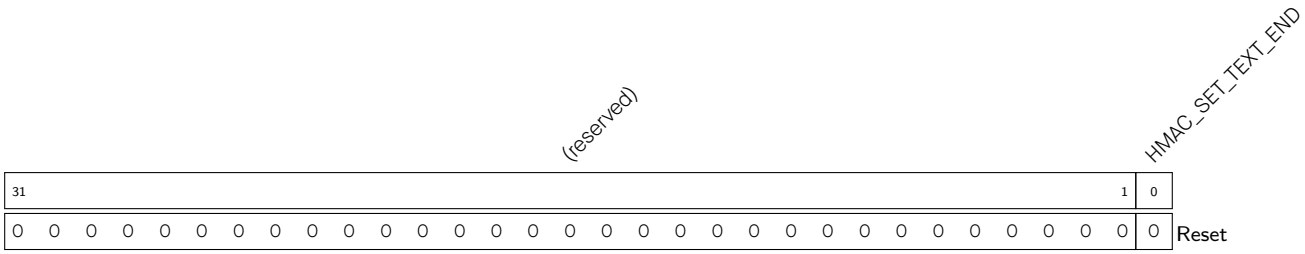
HMAC_SET_TEXT_ING Configures whether there are unprocessed message blocks.

0: No unprocessed message block

1: There are still some message blocks to be processed.

(WO)

Register 21.5. HMAC_SET_MESSAGE_END_REG (0x0058)



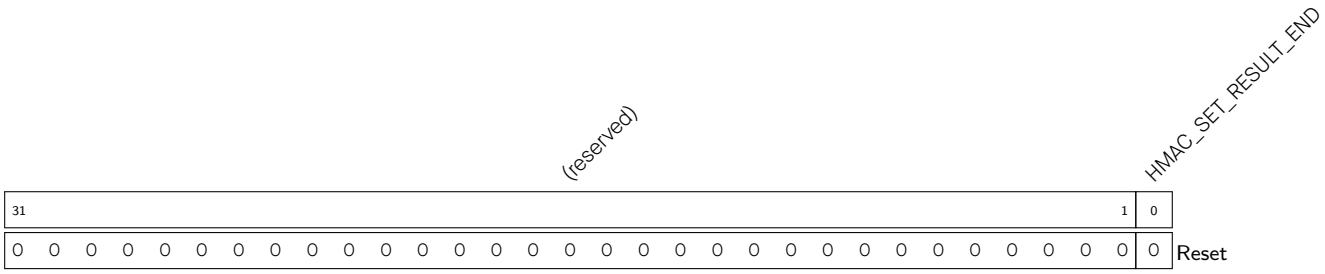
HMAC_SET_TEXT_END Configures whether to start hardware padding.

0: No effect

1: Start hardware padding

(WO)

Register 21.6. HMAC_SET_RESULT_FINISH_REG (0x005C)



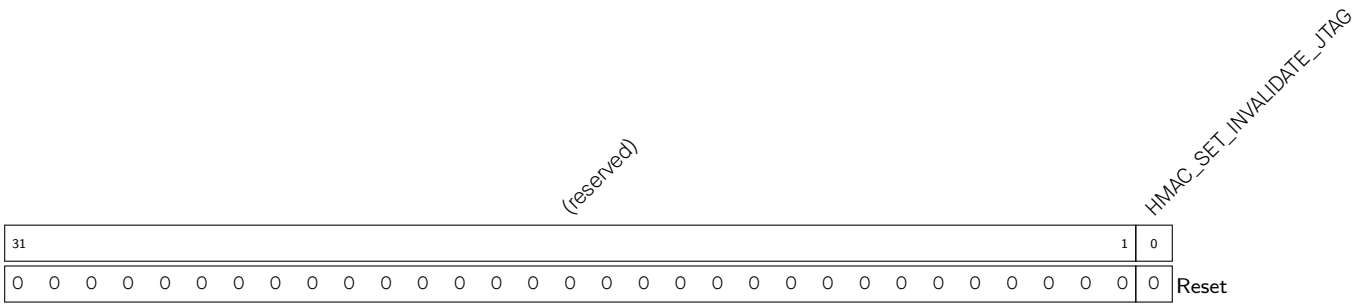
HMAC_SET_RESULT_END Configures whether to exit upstream mode and clear calculation results.

0: No effect

1: Exit upstream mode and clear calculation results.

(WO)

Register 21.7. HMAC_SET_INVALIDATE_JTAG_REG (0x0060)



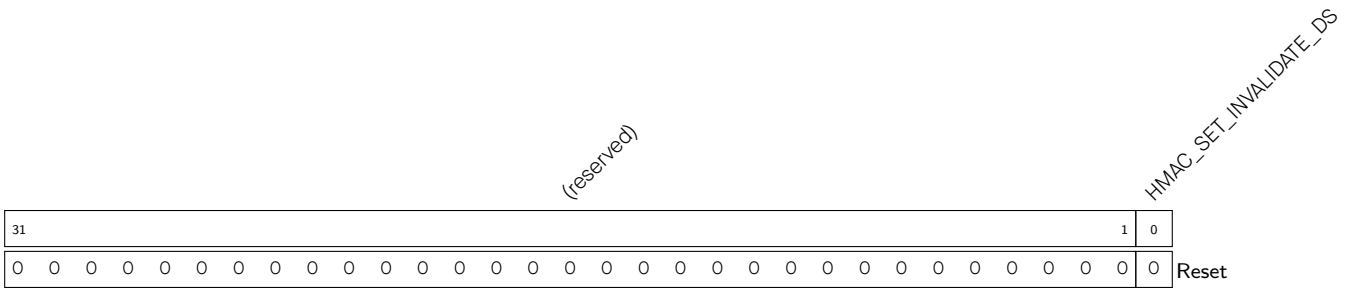
HMAC_SET_INVALIDATE_JTAG Configures whether to clear calculation results when re-enabling JTAG in downstream mode.

0: Not clear

1: Clear calculation results

(WO)

Register 21.8. HMAC_SET_INVALIDATE_DS_REG (0x0064)



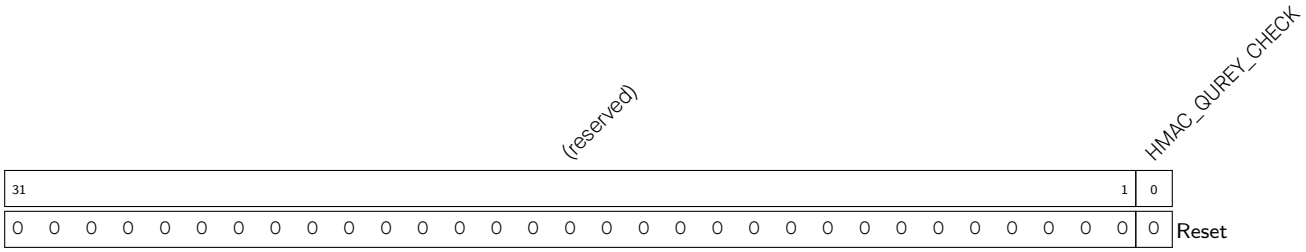
HMAC_SET_INVALIDATE_DS Configures whether to clear calculation results of the RSA_DS peripheral in downstream mode.

0: Not clear

1: Clear calculation results

(WO)

Register 21.9. HMAC_QUERY_ERROR_REG (0x0068)



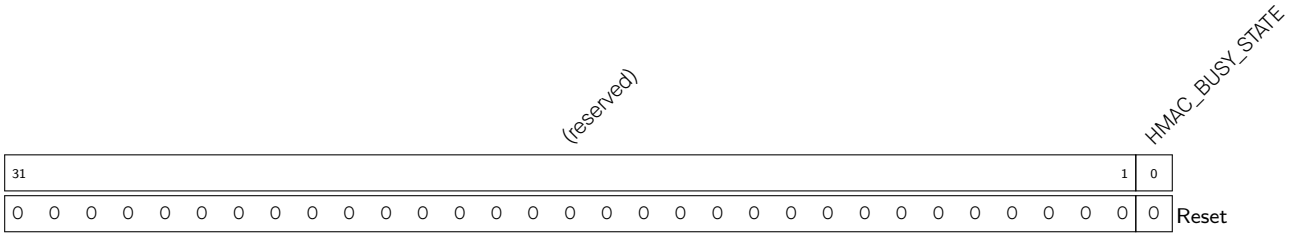
HMAC_QUREY_CHECK Represents whether an HMAC key matches the purpose.

0: Match

1: Error

(RO)

Register 21.10. HMAC_QUERY_BUSY_REG (0x006C)



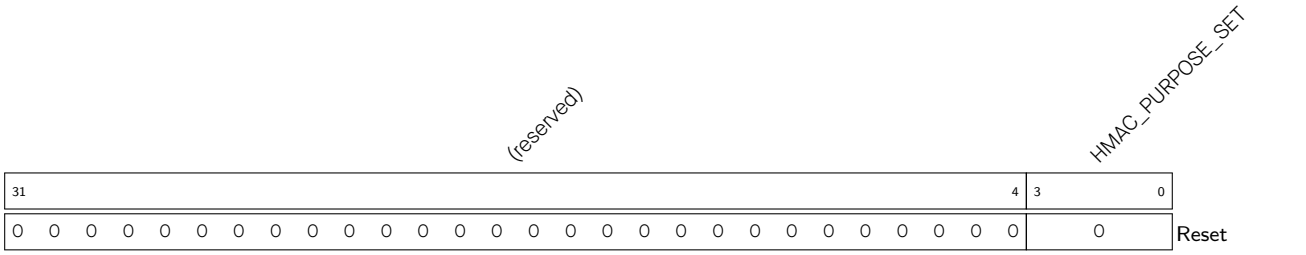
HMAC_BUSY_STATE Represents whether HMAC is in a busy state. Before configuring HMAC, please make sure HMAC is in IDLE state.

0: Idle

1: HMAC is still working on calculation

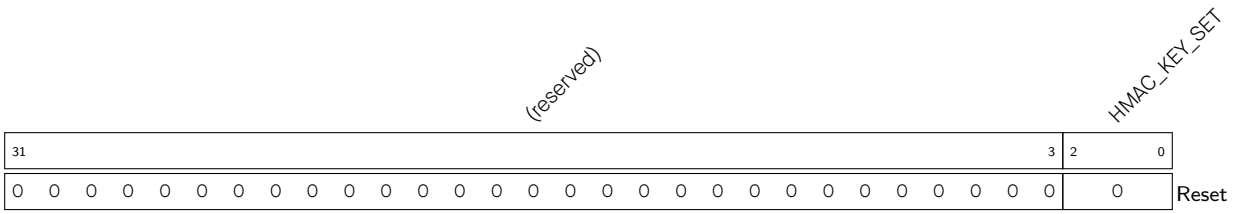
(RO)

Register 21.11. HMAC_SET_PARA_PURPOSE_REG (0x0044)



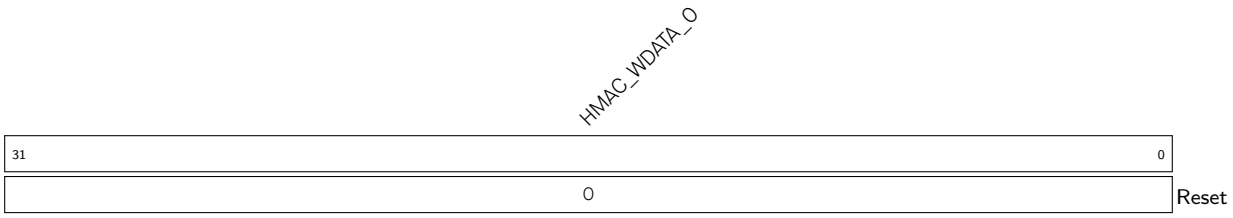
HMAC_PURPOSE_SET Configures the HMAC purpose, refer to the Table 21.2-1. (WO)

Register 21.12. HMAC_SET_PARA_KEY_REG (0x0048)



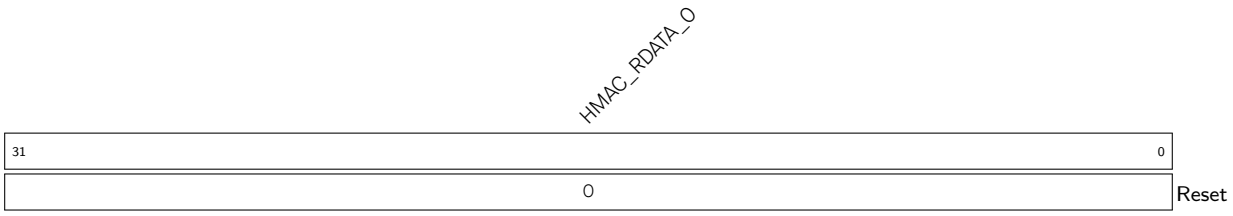
HMAC_KEY_SET Configures HMAC key. There are six keys with index 0~5. Write the index of the selected key to this field. (WO)

Register 21.13. HMAC_WR_MESSAGE_*n*_REG (*n*: 0-15) (0x0080+4**n*)



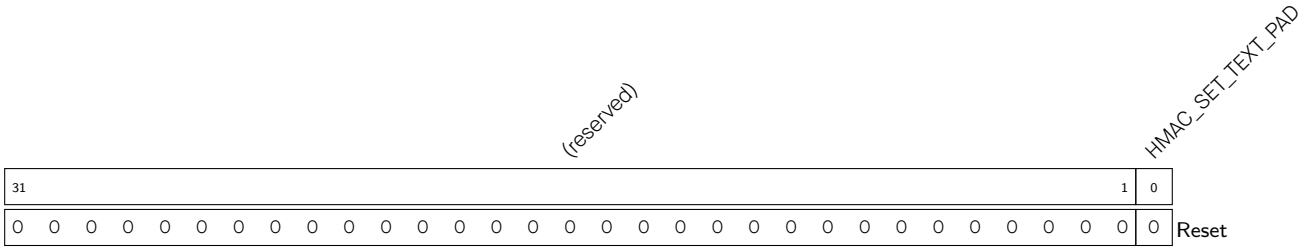
HMAC_WDATA_*n* Represents the *n*th 32-bit of message. (WO)

Register 21.14. HMAC_RD_RESULT_*n*_REG (*n*: 0-7) (0x00C0+4**n*)



HMAC_RDATA_*n* Represents the *n*th 32-bit of hash result. (RO)

Register 21.15. HMAC_SET_MESSAGE_PAD_REG (0x00F0)



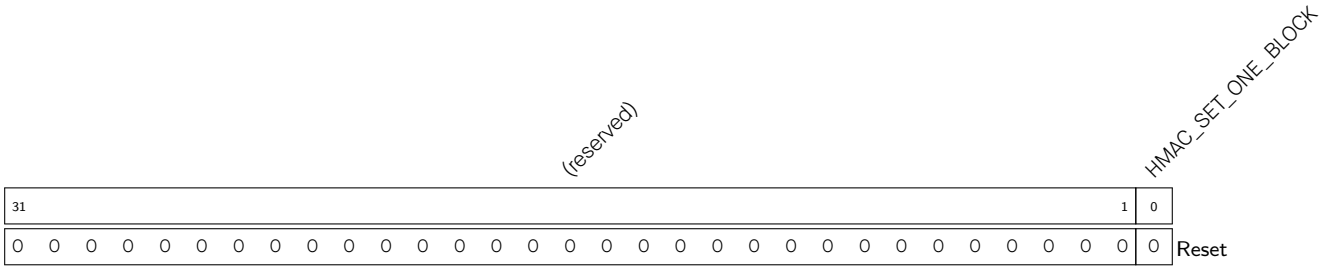
HMAC_SET_TEXT_PAD Configures whether the padding is applied by software.

0: Not applied by software

1: Applied by software

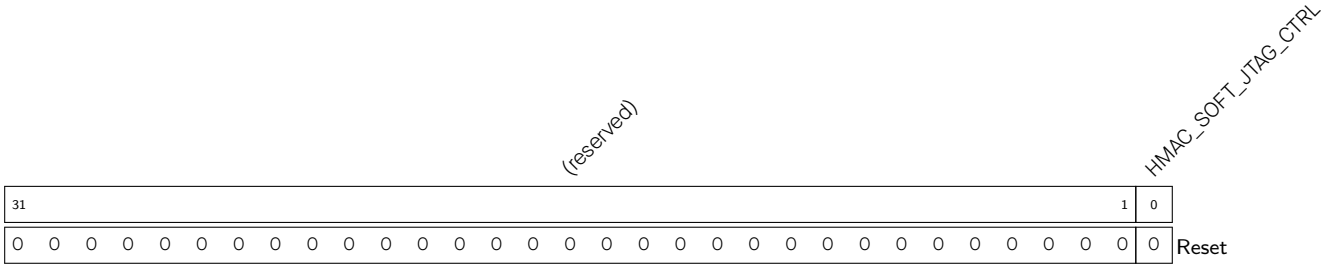
(WO)

Register 21.16. HMAC_ONE_BLOCK_REG (0x00F4)



HMAC_SET_ONE_BLOCK Write 1 to indicate Block_1 is the only one block, and Block_1 contains all the padding bits and there is no need for padding. (WO)

Register 21.17. HMAC_SOFT_JTAG_CTRL_REG (0x00F8)



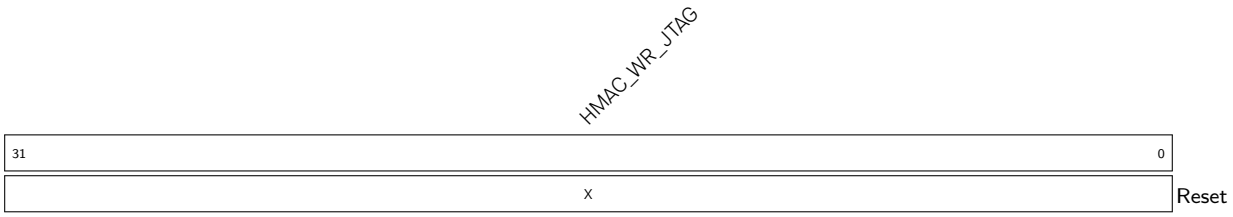
HMAC_SOFT_JTAG_CTRL Configures whether to enable JTAG authentication mode.

0: Disable

1: Enable

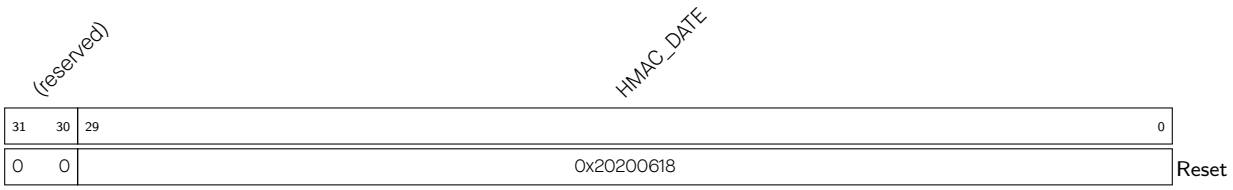
(WO)

Register 21.18. HMAC_WR_JTAG_REG (0x00FC)



HMAC_WR_JTAG Writes the comparing input used for re-enabling JTAG. (WO)

Register 21.19. HMAC_DATE_REG (0x01FC)



HMAC_DATE Version control register. (R/W)

Chapter 22

RSA Accelerator (RSA)

22.1 Introduction

The RSA accelerator provides hardware support for high-precision computation used in various RSA asymmetric cipher algorithms, significantly improving their run time and reducing their software complexity. Compared with RSA algorithms implemented solely in software, this hardware accelerator can speed up RSA algorithms significantly. The RSA accelerator also supports operands of different lengths, which provides more flexibility during the computation.

22.2 Features

The following functionality is supported:

- Large-number modular exponentiation with two optional acceleration options
- Large-number modular multiplication
- Large-number multiplication
- Operands of different lengths
- Interrupt on completion of computation

22.3 Functional Description

The RSA accelerator is activated by setting the PCR_RSA_CLK_EN bit and clearing the PCR_RSA_RST_EN bit in the PCR_RSA_CONF_REG register. Additionally, users also need to clear PCR_DS_RST_EN and PCR_ECDSA_RST_EN bits to reset RSA Digital Signature Peripheral (RSA_DS) and ECDSA Digital Signature Peripheral (ECDSA_DS).

The RSA accelerator is only available after the RSA-related memories are initialized. The content of the RSA_QUERY_CLEAN_REG register is 0 during initialization and will become 1 after the initialization is done. Therefore, wait until RSA_QUERY_CLEAN_REG becomes 1 before using the RSA accelerator.

The RSA_INT_ENA_REG register is used to control the interrupt triggered on completion of computation. Write 1 or 0 to this field to enable or disable the interrupt. By default, the interrupt function of the RSA accelerator is enabled.

Notice:

ESP32-H2's RSA Digital Signature Peripheral (RSA_DS) module also calls the RSA accelerator when working. Therefore, users cannot access the RSA accelerator when the RSA Digital Signature Peripheral (RSA_DS) module is working.

22.3.1 Large-Number Modular Exponentiation

Large-number modular exponentiation performs $Z = X^Y \bmod M$. The computation is based on Montgomery multiplication. Therefore, aside from the X, Y, and M arguments, two additional ones are needed — $\overline{r}$ and M', which need to be calculated in advance by software.

The RSA accelerator supports operands of length $N = 32 \times x$, where $x \in \{1, 2, 3, \ldots, 96\}$. The bit lengths of arguments Z, X, Y, M, and $\overline{r}$ can be arbitrary N, but all numbers in a calculation must be of the same length. The bit length of M' must be 32.

To represent the numbers used as operands, let us define a base-b positional notation, as follows:

$$b = 2^{32}$$

Using this notation, each number is represented by a sequence of base-b digits:

$$n = \frac{N}{32}$$
$$Z = (Z_{n-1} Z_{n-2} \cdots Z_0)_b$$
$$X = (X_{n-1} X_{n-2} \cdots X_0)_b$$
$$Y = (Y_{n-1} Y_{n-2} \cdots Y_0)_b$$
$$M = (M_{n-1} M_{n-2} \cdots M_0)_b$$
$$\overline{r} = (\overline{r}_{n-1} \overline{r}_{n-2} \cdots \overline{r}_0)_b$$

Each of the values in $Z_{n-1} \cdots Z_0$, $X_{n-1} \cdots X_0$, $Y_{n-1} \cdots Y_0$, $M_{n-1} \cdots M_0$, $\overline{r}_{n-1} \cdots \overline{r}_0$ represents one base-b digit (a 32-bit word).

Z_{n-1}, X_{n-1}, Y_{n-1}, M_{n-1} and $\overline{r}_{n-1}$ are the most significant bits of Z, X, Y, M, while Z_0, X_0, Y_0, M_0 and $\overline{r}_0$ are the least significant bits.

If we define $R = b^n$, the additional argument $\overline{r}$ can be calculated as $\overline{r} = R^2 \bmod M$.

Also, argument M' can be calculated using the formula below:

$$M' = -M^{-1} \bmod b$$

where, M^{-1} is the modular multiplicative inverse of M, and it can be calculated with the extended binary GCD algorithm.

Large-number modular exponentiation on the ESP32-H2 can be implemented as follows:

1. Write 1 or 0 to the RSA_INT_ENA field to enable or disable the interrupt function.
2. Configure relevant registers:
   (a) Write $(\frac{N}{32} - 1)$ to the RSA_MODE_REG register.
   (b) Write M' to the RSA_M_PRIME_REG register.
   (c) Configure registers related to the acceleration options, which are described later in Section 22.3.4.
3. Write X_i, Y_i, M_i and $\overline{r}_i$ for $i \in \{0, 1, \ldots, n-1\}$ to memory blocks RSA_X_MEM, RSA_Y_MEM, RSA_M_MEM and RSA_Z_MEM. The capacity of each memory block is 96 words. Each word of each memory block can store one base-b digit. The memory blocks use the little endian format for storage, i.e., the least significant digit of each number is in the lowest address.

Users need to write data to each memory block only according to the length of the number; data beyond this length is ignored.

4. Write 1 to the RSA_SET_START_MODEXP field of the RSA_SET_START_MODEXP_REG register to start computation.
5. Wait for the completion of computation, which happens when the content of RSA_QUERY_IDLE becomes 1 or the RSA interrupt occurs.
6. Read the result Z_i for $i \in \{0, 1, \ldots, n-1\}$ from RSA_Z_MEM.
7. Write 1 to RSA_CLEAR_INTERRUPT to clear the interrupt, if you have the interrupt enabled.

After the computation, the RSA_MODE_REG register, memory blocks RSA_Y_MEM and RSA_M_MEM, as well as the RSA_M_PRIME_REG remain unchanged. However, X_i in RSA_X_MEM and $\overline{r}_i$ in RSA_Z_MEM computation are overwritten, and only these overwritten memory blocks need to be re-initialized before starting another computation.

22.3.2 Large-Number Modular Multiplication

Large-number modular multiplication performs $Z = X \times Y \bmod M$. This computation is based on Montgomery multiplication. Therefore, similar to the large-number modular exponentiation, two additional arguments are needed – $\overline{r}$ and M', which need to be calculated in advance by software.

The RSA accelerator supports large-number modular multiplication with operands of 96 different lengths.

The computation can be executed as follows:

1. Write 1 or 0 to the RSA_INT_ENA_REG register to enable or disable the interrupt function.
2. Configure relevant registers:
   (a) Write $(\frac{N}{32} - 1)$ to the RSA_MODE_REG register.
   (b) Write M' to the RSA_M_PRIME_REG register.
3. Write X_i, Y_i, M_i, and $\overline{r}_i$ for $i \in \{0, 1, \ldots, n-1\}$ to memory blocks RSA_X_MEM, RSA_Y_MEM, RSA_M_MEM, and RSA_Z_MEM, respectively. The capacity of each memory block is 96 words. Each word of each memory block can store one base-b digit. The memory blocks use the little endian format for storage, i.e., the least significant digit of each number is in the lowest address.

   Users need to write data to each memory block only according to the length of the number; data beyond this length are ignored.
4. Write 1 to the RSA_SET_START_MODMULT field.
5. Wait for the completion of computation, which happens when the content of RSA_QUERY_IDLE becomes 1 or the RSA interrupt occurs.
6. Read the result Z_i for $i \in \{0, 1, \ldots, n-1\}$ from RSA_Z_MEM.
7. Write 1 to RSA_CLEAR_INTERRUPT to clear the interrupt, if you have the interrupt enabled.

After the computation, the length of operands in RSA_MODE_REG, the X_i in memory RSA_X_MEM, the Y_i in memory RSA_Y_MEM, the M_i in memory RSA_M_MEM, and the M' in memory RSA_M_PRIME_REG remain

unchanged. However, the $\overline{r}_i$ in memory RSA_Z_MEM has already been overwritten, and only this overwritten memory block needs to be re-initialized before starting another computation.

22.3.3 Large-Number Multiplication

Large-number multiplication performs $Z = X \times Y$. The length of result Z is twice that of operand X and operand Y. Therefore, the RSA accelerator only supports large-number multiplication with operand length $N = 32 \times x$, where $x \in \{1, 2, 3, \ldots, 48\}$. The length $\hat{N}$ of result Z is $2 \times N$.

The computation can be executed as follows:

1. Write 1 or 0 to the RSA_INT_ENA_REG register to enable or disable the interrupt function.
2. Write $(\frac{\hat{N}}{32} - 1)$, i.e., $(\frac{N}{16} - 1)$ to the RSA_MODE_REG register.
3. Write X_i and Y_i for $\in \{0, 1, \ldots, n-1\}$ to memory blocks RSA_X_MEM and RSA_Z_MEM. Each word of each memory block can store one base-b digit. The memory blocks use the little endian format for storage, i.e., the least significant digit of each number is in the lowest address. n is $\frac{N}{32}$.

   Write X_i for $i \in \{0, 1, \ldots, n-1\}$ to the address of the i words of the RSA_X_MEM memory block. Note that Y_i for $i \in \{0, 1, \ldots, n-1\}$ will not be written to the address of the i words of the RSA_Z_MEM register, but the address of the $n + i$ words, i.e., the base address of the RSA_Z_MEM memory plus the address offset $4 \times (n + i)$.

   Users need to write data to each memory block only according to the length of the number; data beyond this length is ignored.
4. Write 1 to the RSA_SET_START_MULT register.
5. Wait for the completion of computation, which happens when the content of RSA_QUERY_IDLE becomes 1 or the RSA interrupt occurs.
6. Read the result Z_i for $i \in \{0, 1, \ldots, \hat{n}-1\}$ from the RSA_Z_MEM register. $\hat{n}$ is $2 \times n$.
7. Write 1 to RSA_CLEAR_INTERRUPT to clear the interrupt, if you have the interrupt enabled.

After the computation, the length of operands in RSA_MODE_REG and the X_i in memory RSA_X_MEM remain unchanged. However, the Y_i in memory RSA_Z_MEM has already been overwritten, and only this overwritten memory block needs to be re-initialized before starting another computation.

22.3.4 Options for Additional Acceleration

The ESP32-H2 RSA accelerator also provides SEARCH and CONSTANT_TIME options that can be configured to further accelerate the large-number modular exponentiation. By default, both options are configured as no additional acceleration.

Users can choose to use one or two of these options to further accelerate the computation. Note that, even when none of these two options is configured, using the hardware RSA accelerator is still much faster than implementing the RSA algorithm in software.

To be more specific, when neither of these two options are configured for additional acceleration, the time required to calculate $Z = X^Y \bmod M$ is solely determined by the lengths of operands. When either or both of these two options are configured for additional acceleration, the time required is also correlated with the 0/1 distribution of Y.

To better illustrate how these two options work, first assume Y is represented in binaries as

$$Y = (\widetilde{Y}_{N-1}\widetilde{Y}_{N-2}\cdots\widetilde{Y}_{t+1}\widetilde{Y}_t\widetilde{Y}_{t-1}\cdots\widetilde{Y}_0)_2$$

where,

- N is the length of Y,
- $\widetilde{Y}_t$ is 1,
- $\widetilde{Y}_{N-1}$, $\widetilde{Y}_{N-2}$, ..., $\widetilde{Y}_{t+1}$ are all equal to 0,
- and $\widetilde{Y}_{t-1}$, $\widetilde{Y}_{t-2}$, ..., $\widetilde{Y}_0$ are either 0 or 1 but exactly m bits should be equal to 0 and t-m bits 1, i.e., the Hamming weight of $\widetilde{Y}_{t-1}\widetilde{Y}_{t-2}, \cdots, \widetilde{Y}_0$ is $t - m$.

When either of these two options is configured for additional acceleration:

- SEARCH Option (Configuring RSA_SEARCH_ENABLE to 1 for additional acceleration)
  - The accelerator ignores the bit positions of $\widetilde{Y}_i$, where $i > \alpha$. Search position α is set by configuring the RSA_SEARCH_POS_REG register. Set α to a number smaller than N-1, which otherwise leads to the same result as if this option is not used for additional acceleration. The best acceleration performance can be achieved by setting α to t, in which case all the $\widetilde{Y}_{N-1}$, $\widetilde{Y}_{N-2}$, ..., $\widetilde{Y}_{t+1}$ of 0s are ignored during the calculation. Note that if you set α to be less than t, then the result of the modular exponentiation $Z = X^Y \bmod M$ will be incorrect.
  - Note that this option compromises the security because it ignores some bits, which essentially shortens the key length, thus should not be enabled for applications with high security requirement.
- CONSTANT_TIME Option (Configuring RSA_CONSTANT_TIME_REG to 0 for additional acceleration)
  - The accelerator speeds up the calculation by simplifying the calculation concerning the 0 bits of Y. Therefore, the higher the proportion of bits 0 against bits 1, the better is the acceleration performance.
  - Note that this option also compromises the security because its time cost correlates with the 0/1 distribution of the key, which can be used in a Side Channel Attack (SCA), thus should not be enabled for applications with high security requirement.

Below is an example to demonstrate the performance of the RSA accelerator under different combinations of SEARCH and CONSTANT_TIME configuration. Here we perform $Z = X^Y \bmod M$ with N = 3072 and Y = 65537. Table 22.3-1 below demonstrates the time costs under different combinations of SEARCH and CONSTANT_TIME configuration. Here, we should also mention that, α is set to 16 when the SEARCH option is enabled.

Table 22.3-1. Acceleration Performance

SEARCH Option	CONSTANT_TIME Option	Time Cost (ms)
No acceleration	No acceleration	451.69
Accelerated	No acceleration	2.71
No acceleration	Acceleration	1.45
Acceleration	Acceleration	1.40

It is obvious that:

- The time cost is biggest when none of these two options is configured for additional acceleration.
- The time cost is smallest when both of these two options are configured for additional acceleration.
- The time cost can be dramatically reduced when either or both option(s) are configured for additional acceleration.

22.4 Memory Summary

The addresses in this section are relative to the RSA accelerator base address provided in Table 4.3-2 in Chapter 4 *System and Memory*.

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

Table 22.4-1. RSA Accelerator Memory Blocks

Name	Description	Size (byte)	Starting Address	Ending Address	Access
RSA_M_MEM	Memory M	384	0x0000	0x017F	R/W
RSA_Z_MEM	Memory Z	384	0x0200	0x037F	R/W
RSA_Y_MEM	Memory Y	384	0x0400	0x057F	R/W
RSA_X_MEM	Memory X	384	0x0600	0x077F	R/W

22.5 Register Summary

The addresses in this section are relative to the RSA accelerator base address provided in Table 4.3-2 in Chapter 4 *System and Memory*.

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

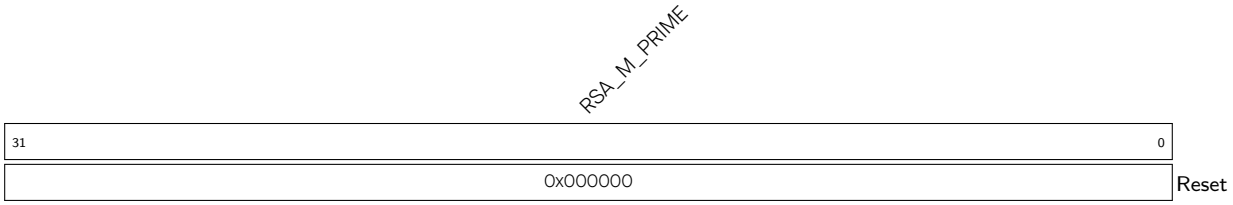
Name	Description	Address	Access
Control / Configuration Registers			
RSA_M_PRIME_REG	Represents M'	0x0800	R/W
RSA_MODE_REG	Configures RSA length	0x0804	R/W
RSA_SET_START_MODEXP_REG	Starts modular exponentiation	0x080C	WT
RSA_SET_START_MODMULT_REG	Starts modular multiplication	0x0810	WT
RSA_SET_START_MULT_REG	Starts multiplication	0x0814	WT
RSA_QUERY_IDLE_REG	Represents the RSA status	0x0818	RO
RSA_CONSTANT_TIME_REG	Configures the constant_time option	0x0820	R/W
RSA_SEARCH_ENABLE_REG	Configures the search option	0x0824	R/W
RSA_SEARCH_POS_REG	Configures the search position	0x0828	R/W
Status Register			
RSA_QUERY_CLEAN_REG	RSA initialization status	0x0808	RO
Interrupt Registers			
RSA_INT_CLR_REG	Clears RSA interrupt	0x081C	WT
RSA_INT_ENA_REG	Enables the RSA interrupt	0x082C	R/W
Version Control Register			
RSA_DATE_REG	Version control register	0x0830	R/W

22.6 Registers

The addresses in this section are relative to the RSA accelerator base address provided in Table 4.3-2 in Chapter 4 *System and Memory*.

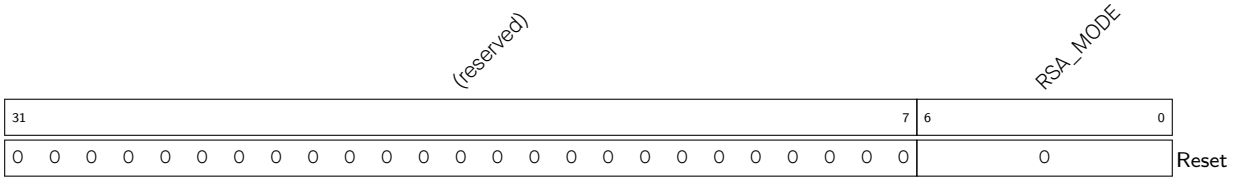
For how to program reserved fields, please refer to Section Programming Reserved Register Field.

Register 22.1. RSA_M_PRIME_REG (0x0800)



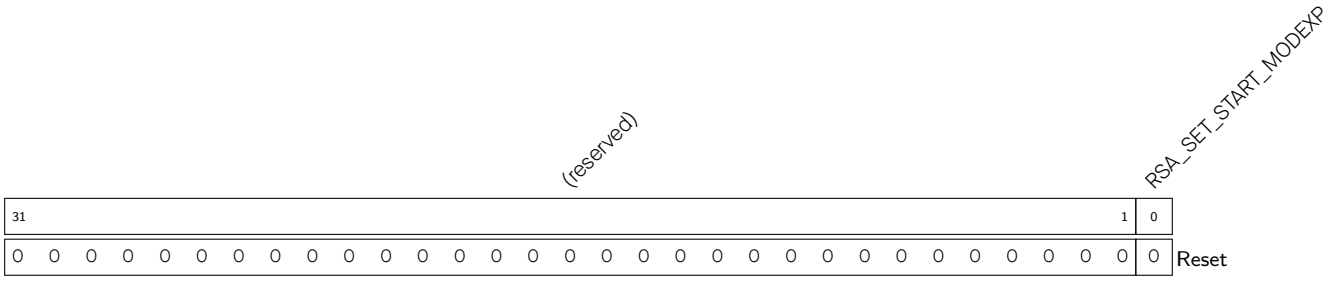
RSA_M_PRIME Represents M'. (R/W)

Register 22.2. RSA_MODE_REG (0x0804)



RSA_MODE Configures the RSA length. (R/W)

Register 22.3. RSA_SET_START_MODEXP_REG (0x080C)



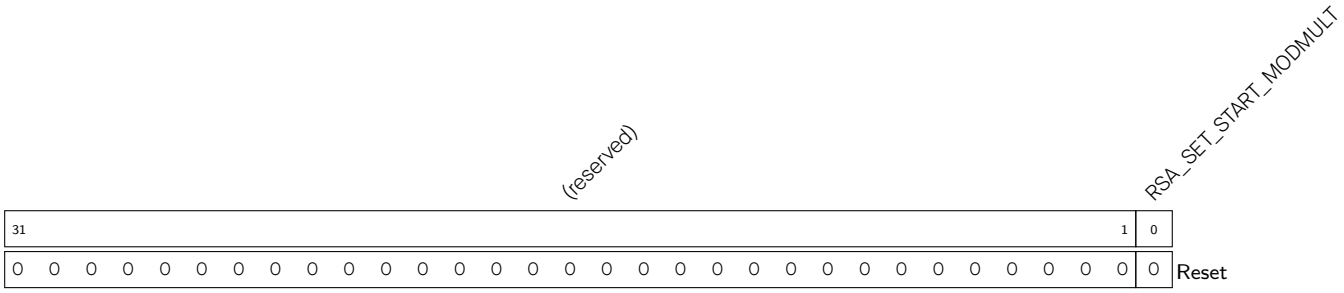
RSA_SET_START_MODEXP Configures whether or not to starts the modular exponentiation.

0: No effect

1: Start

(WT)

Register 22.4. RSA_SET_START_MODMULT_REG (0x0810)



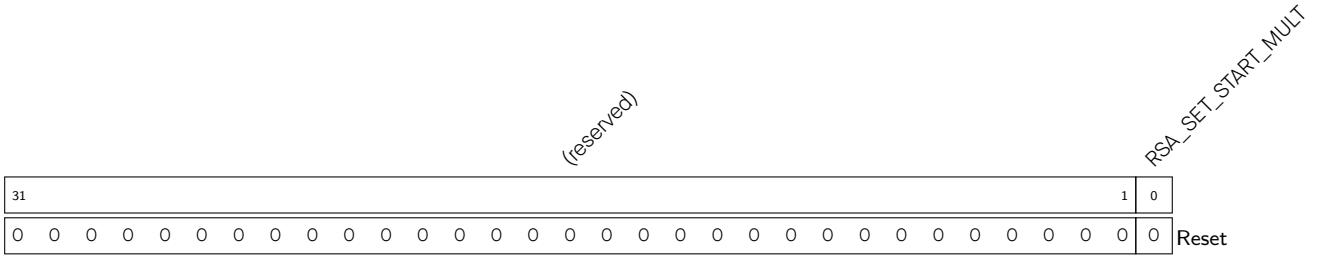
RSA_SET_START_MODMULT Configures whether or not to start the modular multiplication.

0: No effect

1: Start

(WT)

Register 22.5. RSA_SET_START_MULT_REG (0x0814)



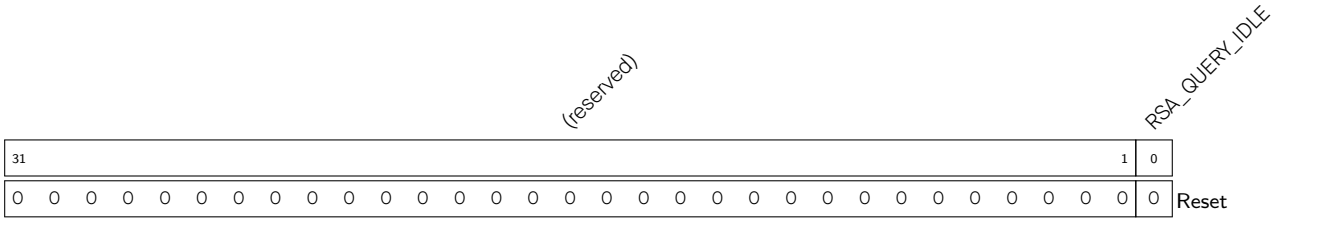
RSA_SET_START_MULT Configures whether or not to start the multiplication.

0: No effect

1: Start

(WT)

Register 22.6. RSA_QUERY_IDLE_REG (0x0818)



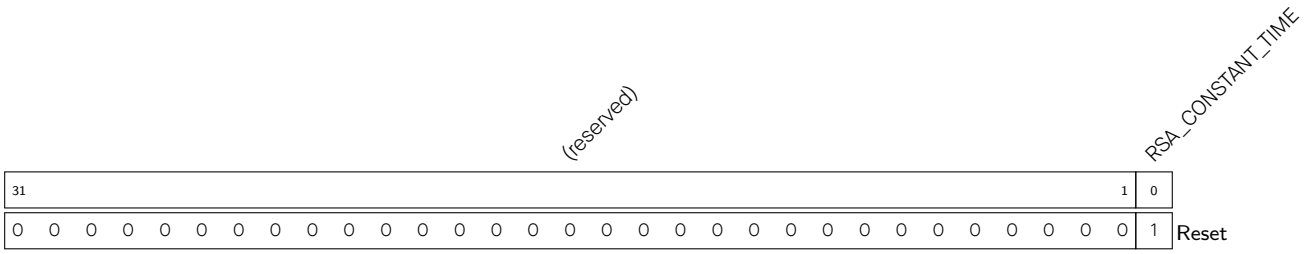
RSA_QUERY_IDLE Represents the RSA status.

0: Busy

1: Idle

(RO)

Register 22.7. RSA_CONSTANT_TIME_REG (0x0820)



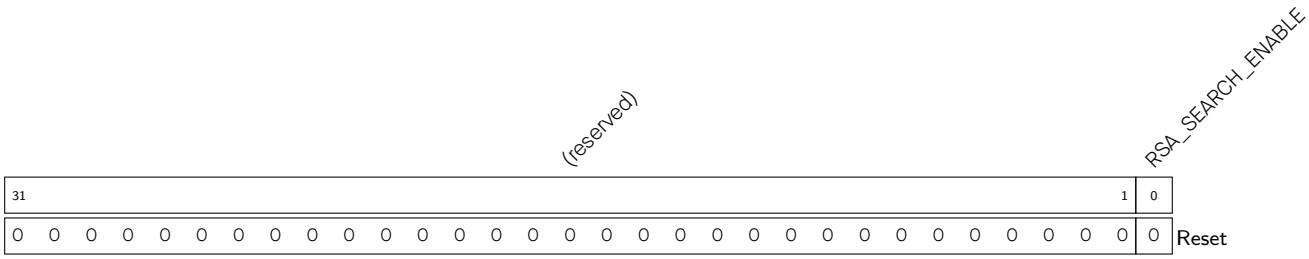
RSA_CONSTANT_TIME Configures the constant_time option.

0: Acceleration

1: No acceleration (default)

(R/W)

Register 22.8. RSA_SEARCH_ENABLE_REG (0x0824)



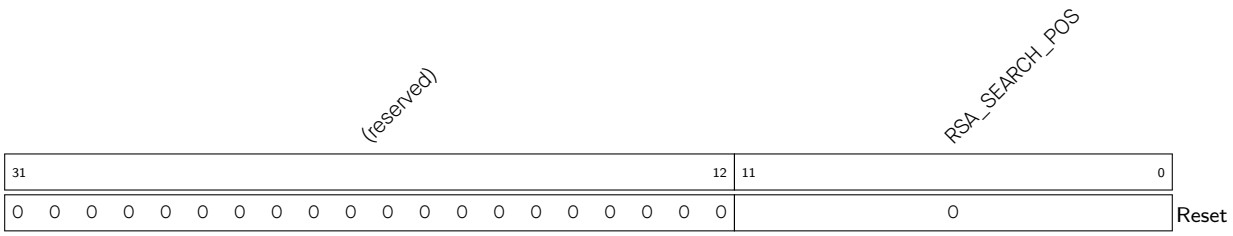
RSA_SEARCH_ENABLE Configures the search option.

0: No acceleration (default)

1: Acceleration

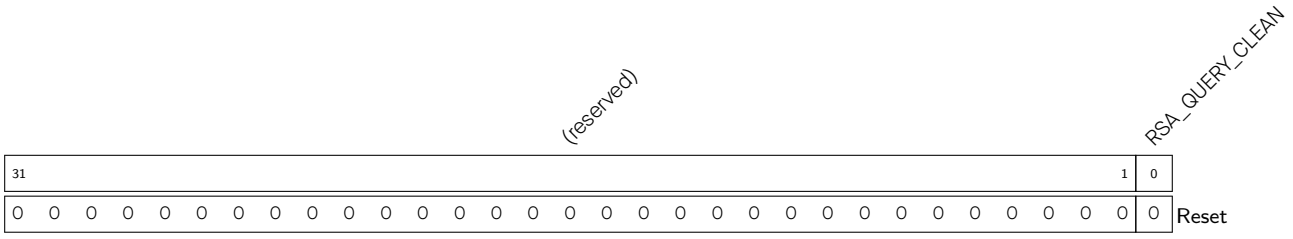
This option should be used together with RSA_SEARCH_POS_REG. (R/W)

Register 22.9. RSA_SEARCH_POS_REG (0x0828)



RSA_SEARCH_POS Configures the starting address to start search. This field should be used together with RSA_SEARCH_ENABLE_REG. The field is only valid when RSA_SEARCH_ENABLE is high. (R/W)

Register 22.10. RSA_QUERY_CLEAN_REG (0x0808)



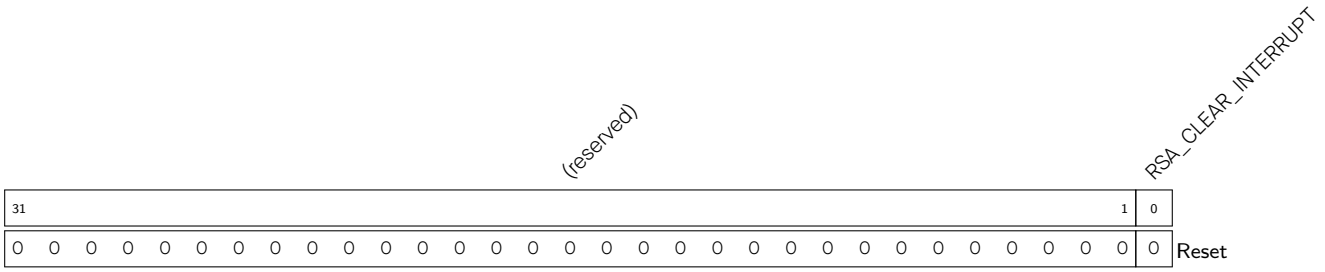
RSA_QUERY_CLEAN Represents whether or not the RSA memory completes initialization.

0: Not complete

1: Completed

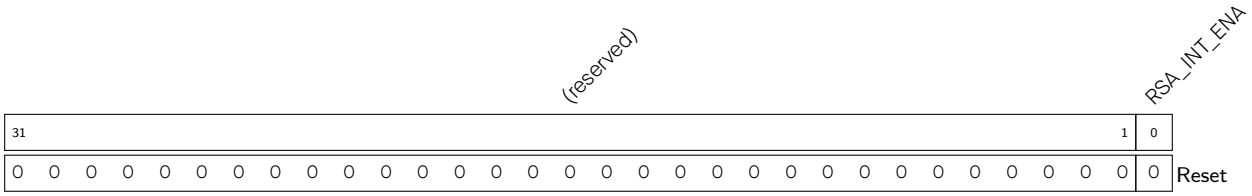
(RO)

Register 22.11. RSA_INT_CLR_REG (0x081C)



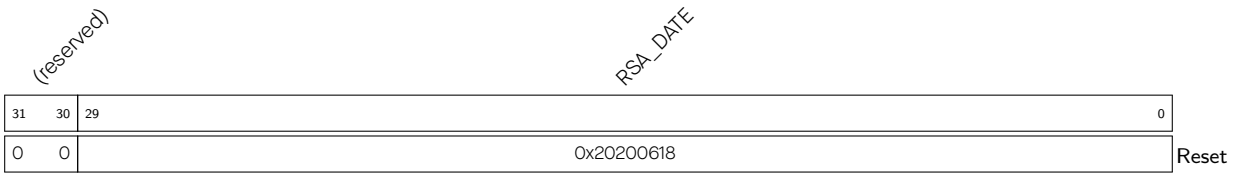
RSA_CLEAR_INTERRUPT Write 1 to clear the RSA interrupt. (WT)

Register 22.12. RSA_INT_ENA_REG (0x082C)



RSA_INT_ENA Write 1 to enable the RSA interrupt. (R/W)

Register 22.13. RSA_DATE_REG (0x0830)



RSA_DATE Version control register. (R/W)

Chapter 23

SHA Accelerator (SHA)

23.1 Introduction

ESP32-H2 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-H2 has two working modes, which are Typical SHA and DMA-SHA.

23.2 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

23.3 Working Modes

The SHA accelerator integrated in ESP32-H2 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_DS_RST_EN, PCR_HMAC_RST_EN, and PCR_ECDSA_RST_EN bits to reset RSA Digital Signature Peripheral (RSA_DS), HMAC Accelerator (HMAC), and 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 23.3-1.

Table 23.3-1. 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 23.3-2.

Table 23.3-2. SHA Hash Algorithm Selection

Hash Algorithm	SHA_MODE_REG Configuration
SHA-1	0
SHA-224	1
SHA-256	2

Notice:

ESP32-H2's RSA Digital Signature Peripheral (RSA_DS) and HMAC Accelerator (HMAC) modules also call the SHA accelerator when working. Therefore, users cannot access the SHA accelerator when these modules are working.

23.4 Function Description

The SHA accelerator generates the message digest via two steps: Preprocessing and Hash operation.

23.4.1 Preprocessing

Preprocessing consists of three steps: padding the message, parsing the message into message blocks and setting the initial hash value.

23.4.1.1 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".

23.4.1.2 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".

23.4.1.3 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.

23.4.2 Hash Operation

After the preprocessing, the ESP32-H2 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-H2 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.

23.4.2.1 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-H2 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 23.3-2.
2. Process the current message block.
   - Write the message block in registers SHA_M_*n*_REG.
3. Start the SHA accelerator[1].
   - If this is the first time to execute this step, set the SHA_START_REG register to 1 to start the SHA accelerator. In this case, the accelerator uses the initial hash value stored in hardware for a given algorithm configured in Step 1 to start the calculation;

- If this is not the first time to execute this step[2], set the SHA_CONTINUE_REG register to 1 to start the SHA accelerator. In this case, the accelerator uses the hash value stored in the SHA_H_*n*_REG register to start calculation.

4. Check the progress of the current message block.
   - Poll register SHA_BUSY_REG until the content of this register becomes 0, indicating the accelerator has completed the calculation for the current message block and now is in the "idle" status [3].
5. Decide if you have more message blocks to process:
   - If yes, please go back to Step 2.
   - Otherwise, please continue.
6. Obtain the message digest.
   - Read the message digest from registers SHA_H_*n*_REG.

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-H2 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.

23.4.2.2 DMA-SHA Mode Process

ESP32-H2 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 3 *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 23.3-2.
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.

23.4.3 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 23.4-1 below:

Table 23.4-1. 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 23.4.1.2.

23.4.4 Interrupt

When working in the DMA-SHA mode, SHA supports interrupt on the completion of message digest calculation.

- To enable this function: write 1 to register SHA_INT_ENA_REG.
- Note that the interrupt should be cleared by software after use via setting the SHA_INT_CLEAR_REG register to 1.

When working in the Typical SHA mode, SHA completes the calculation quickly, so an interrupt is not necessary. Therefore, SHA does not support interrupt in the Typical SHA mode.

23.5 Register Summary

The addresses in this section are relative to the SHA accelerator base address provided in Table 4.3-2 in Chapter 4 *System and Memory*.

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

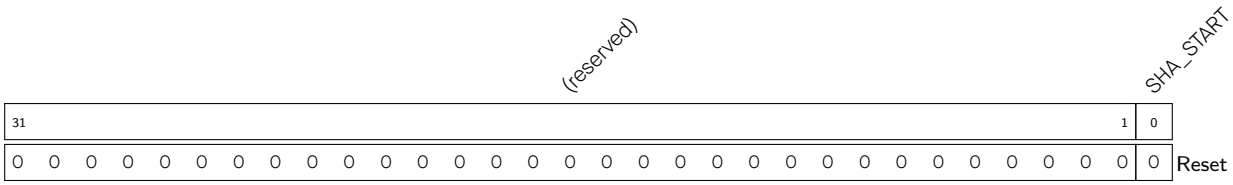
Name	Description	Address	Access
Control/Configuration Registers			
SHA_MODE_REG	Configures SHA algorithm	0x0000	R/W
SHA_CONTINUE_REG	Continues SHA operation (only effective in Typical SHA mode)	0x0014	WO
SHA_DMA_START_REG	Starts the SHA accelerator for DMA-SHA operation	0x001C	WO
SHA_START_REG	Starts 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

23.6 Registers

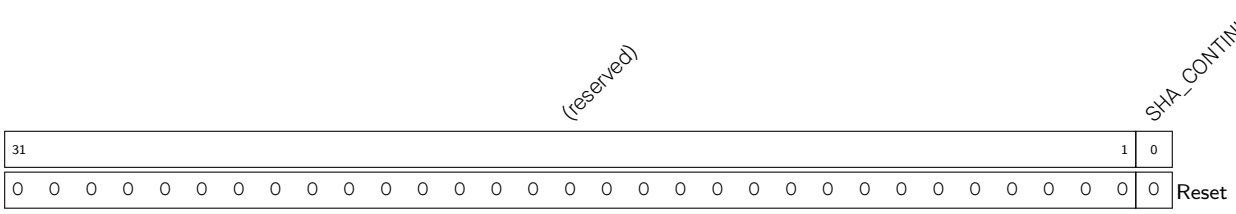
The addresses in this section are relative to the SHA accelerator base address provided in Table 4.3-2 in Chapter 4 *System and Memory*.

Register 23.1. SHA_START_REG (0x0010)



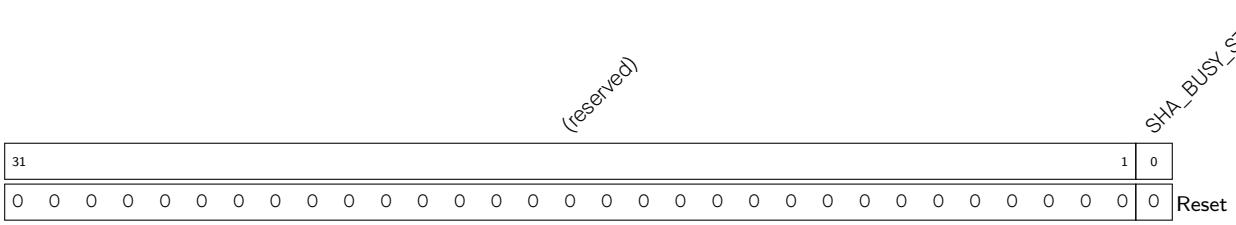
SHA_START Write 1 to start Typical SHA calculation. (WO)

Register 23.2. SHA_CONTINUE_REG (0x0014)



SHA_CONTINUE Write 1 to continue Typical SHA calculation. (WO)

Register 23.3. SHA_BUSY_REG (0x0018)



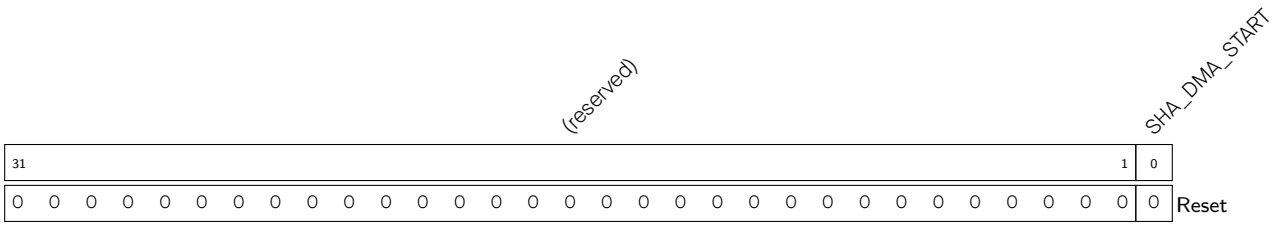
SHA_BUSY_STATE Represents the states of SHA accelerator.

0: idle

1: busy

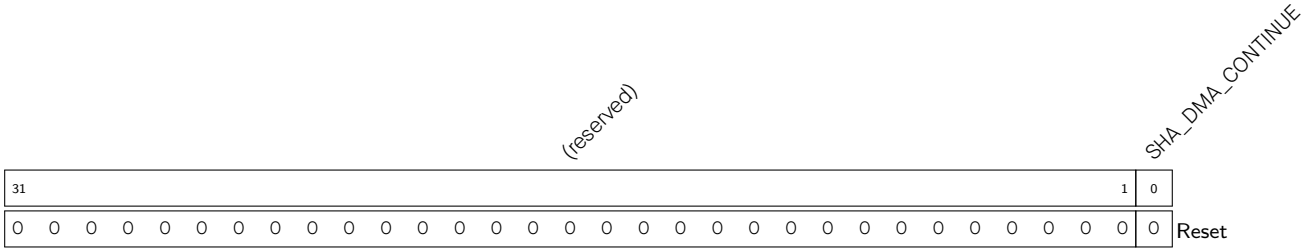
(RO)

Register 23.4. SHA_DMA_START_REG (0x001C)



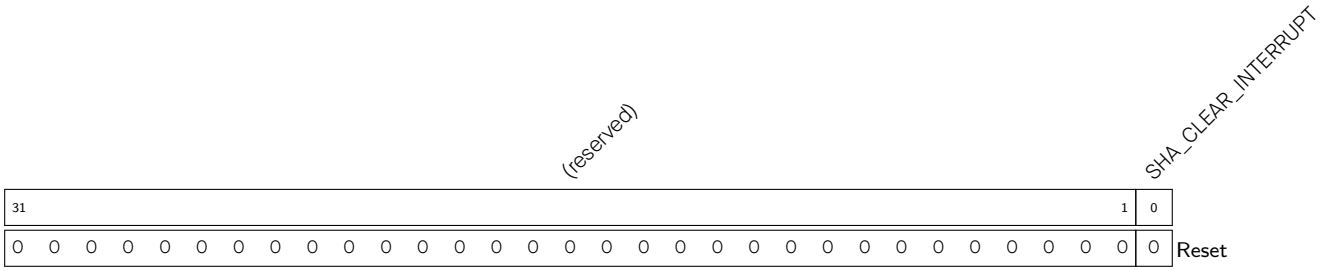
SHA_DMA_START Write 1 to start DMA-SHA calculation. (WO)

Register 23.5. SHA_DMA_CONTINUE_REG (0x0020)



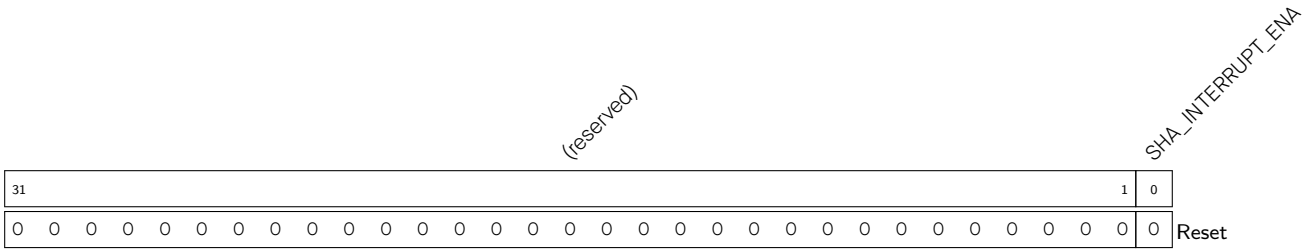
SHA_DMA_CONTINUE Write 1 to continue DMA-SHA calculation. (WO)

Register 23.6. SHA_INT_CLEAR_REG (0x0024)



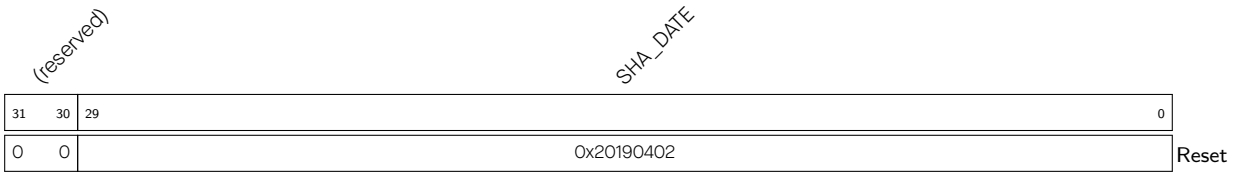
SHA_CLEAR_INTERRUPT Write 1 to clear DMA-SHA interrupt. (WO)

Register 23.7. SHA_INT_ENA_REG (0x0028)



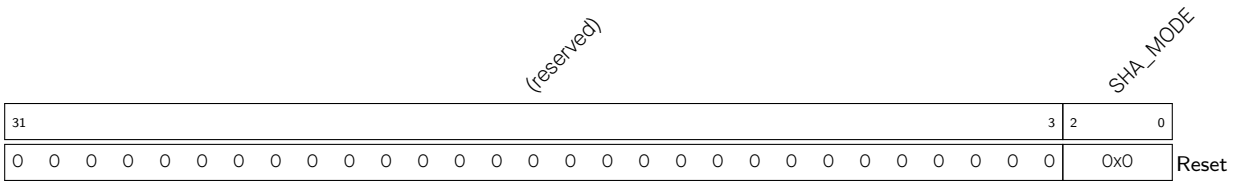
SHA_INTERRUPT_ENA Write 1 to enable DMA-SHA interrupt. (R/W)

Register 23.8. SHA_DATE_REG (0x002C)



SHA_DATE Version control register. (R/W)

Register 23.9. SHA_MODE_REG (0x0000)



SHA_MODE Configures the SHA algorithm.

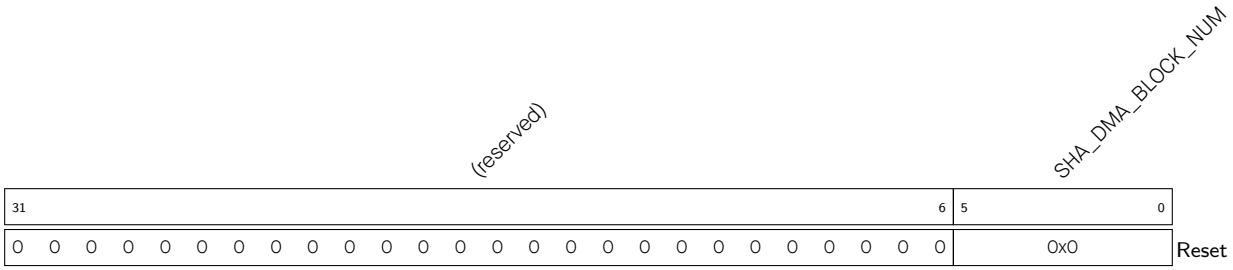
0: SHA-1

1: SHA-224

2: SHA-256

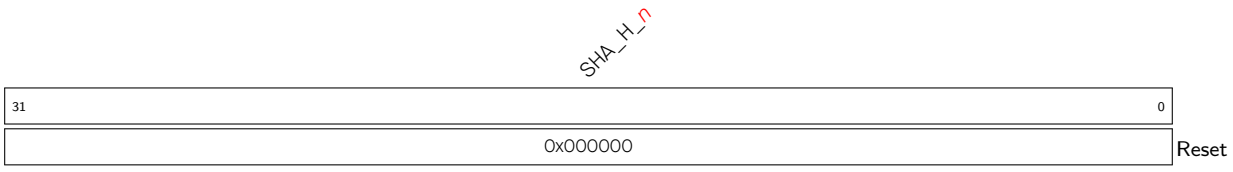
(R/W)

Register 23.10. SHA_DMA_BLOCK_NUM_REG (0x000C)



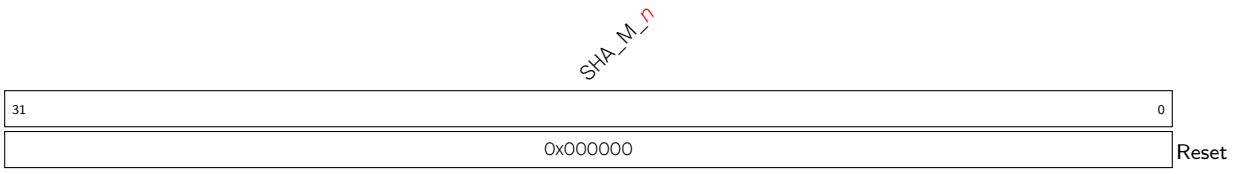
SHA_DMA_BLOCK_NUM Configures the DMA-SHA block number. (R/W)

Register 23.11. SHA_H_*n*_REG (*n*: 0-7) (0x0040+4**n*)



SHA_H_*n* Represents the *n*th 32-bit piece of the Hash value. (R/W)

Register 23.12. SHA_M_*n*_REG (*n*: 0-15) (0x0080+4n*)**



SHA_M_*n* Represents the *n*th 32-bit piece of the message. (R/W)

Chapter 24

RSA Digital Signature Peripheral (RSA_DS)

24.1 Overview

The RSA Digital Signature Peripheral (RSA_DS) is used to verify the authenticity and integrity of a message using a cryptographic algorithm. This can be used to validate a device's identity to a server or to check the integrity of a message.

ESP32-H2 includes an RSA Digital Signature Peripheral (RSA_DS) providing hardware acceleration of messages' signatures based on RSA. HMAC is used as the key derivation function (KDF) to output RSA_DS_KEY using a key stored in eFuse as the input key. Subsequently, the RSA_DS peripheral uses RSA_DS_KEY to decrypt the pre-encrypted parameters and calculate the signature. The whole process happens in hardware so that neither the decryption key for the RSA parameters nor the input key for the HMAC key derivation function can be seen by users while calculating the signature.

24.2 Features

- RSA digital signatures with key length up to 3072 bits
- Encrypted private key data, only decryptable by the RSA_DS peripheral
- SHA-256 digest to protect private key data against tampering by an attacker

24.3 Functional Description

24.3.1 Overview

The RSA_DS peripheral calculates RSA signatures as $Z = X^Y \bmod M$, where Z is the signature, X is the input message, and Y and M are the RSA private key parameters.

Private key parameters are stored in flash as ciphertext. They are decrypted using RSA_DS_KEY which can only be calculated by the RSA_DS peripheral via the HMAC peripheral. The required inputs ($HMAC\_KEY$) to generate the key are only stored in eFuse and can only be accessed by the HMAC peripheral. That is to say, the RSA_DS peripheral hardware can decrypt the private key, and the private key in plaintext is never accessed by the software. For more detailed information about eFuse and HMAC peripherals, please refer to Chapter 5 *eFuse Controller (EFUSE)* and Chapter 21 *HMAC Accelerator (HMAC)*.

The input message X will be sent directly to the RSA_DS peripheral by the software each time a signature is needed. After the RSA signature operation, the signature Z is read back by the software.

For better understanding, we define some symbols and functions here, which are only applicable to this chapter:

- $\mathbf{1}^s$: A bit string consisting of s bits with the value of "1".
- $\mathbf{[}x\mathbf{]}_s$: A bit string of s bits, in which s is an integer multiple of 8 bits. If x is a number ($x < 2^s$), it is represented in little-endian byte order in the bit string. x may be a variable such as $[Y]_{4096}$ or a hexadecimal constant such as $[0x0C]_8$. If necessary, the value $[x]_t$ can be right-padded with $(s-t)$ number of zeros to reach s bits in length, and finally get $[x]_s$. For example, $[0x05]_8 = 00000101$, $[0x05]_{16} = 0000010100000000$, $[0x0005]_{16} = 0000000000000101$, $[0x13]_8 = 00010011$, $[0x13]_{16} = 0001001100000000$, $[0x0013]_{16} = 0000000000010011$.
- $||$: A bit string concatenation operator for joining multiple-bit strings into a longer bit string.

24.3.2 Private Key Operands

Private key operands Y (private key exponent) and M (key modulus) are generated by the user. They have a particular RSA key length (up to 3072 bits). Two additional private key operands are needed: $\overline{r}$ and M'. These two operands are derived from Y and M.

Operands Y, M, $\overline{r}$, and M' are encrypted by the user along with an authentication digest and stored as a single ciphertext C. C is input to the RSA_DS peripheral in this encrypted format, decrypted by the hardware, and then used for RSA signature calculation. A detailed description of how to generate C is provided in Section 24.3.3.

The RSA_DS peripheral supports RSA signature calculation $Z = X^Y \bmod M$, in which the length of operands should be $N = 32 \times x$ where $x \in \{1, 2, 3, \ldots, 96\}$. The bit lengths of arguments Z, X, Y, M, and $\overline{r}$ should be an arbitrary value in N, and all of them in a calculation must be of the same length, while the bit length of M' should always be 32. For more detailed information about RSA calculation, please refer to Section 22.3.1 *Large-Number Modular Exponentiation* in Chapter *22 RSA Accelerator (RSA)*.

24.3.3 Software Prerequisites

If you want to use the RSA_DS peripheral, the software needs a series of preparations, as shown in Figure 24.3-1. The left side lists preparations required by the software before the hardware starts the RSA signature calculation, while the right side lists the hardware workflow during the entire calculation procedure.

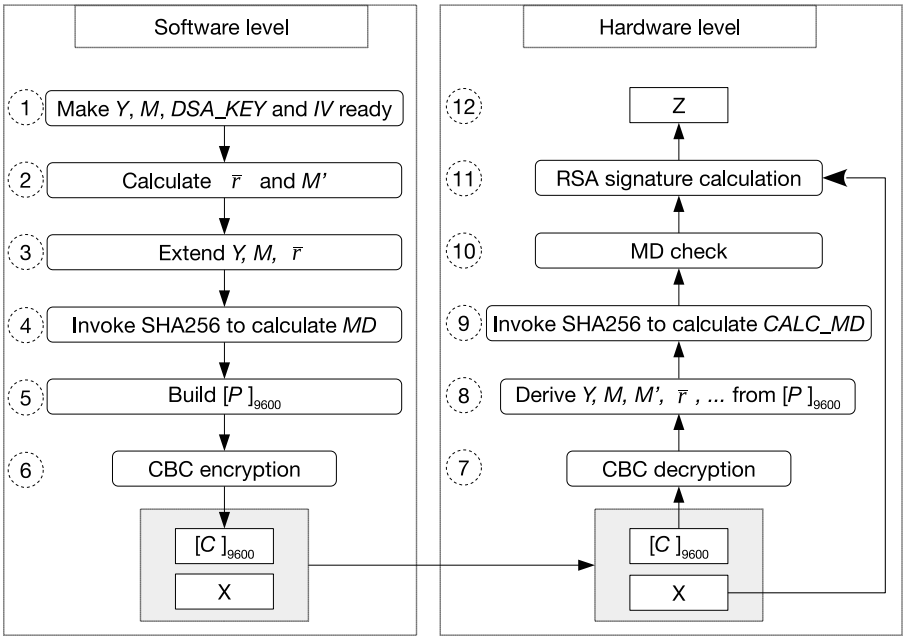


Figure 24.3-1. Software Preparations and Hardware Working Process

Note:

1. The software preparation (left side in Figure 24.3-1) is a one-time operation before any signature is calculated, while the hardware calculation (right side in Figure 24.3-1) repeats for every signature calculation.
2. Software preparation requires configuring the clock reset. For more information, please refer to Chapter 7 *Reset and Clock*.

Users need to follow the steps shown in the left part of Figure 24.3-1 to calculate C. Detailed instructions are as follows:

- **Step 1**: Prepare operands Y and M whose lengths should meet the requirements in Section 24.3.2. Define $[L]_{32} = \frac{N}{32} - 1$ (i.e., for RSA 3072, $[L]_{32}$ == $[0x60\text{-}1]_{32}$). Prepare $[HMAC\_KEY]_{256}$ and calculate $[RSA\_DS\_KEY]_{256}$ based on RSA_DS_KEY = HMAC-SHA256 ($[HMAC\_KEY]_{256}$, 1^{256}). Generate a random $[IV]_{128}$ which should meet the requirements of the AES-CBC block encryption algorithm. For more information on AES, please refer to Chapter 19 *AES Accelerator (AES)*.
- **Step 2**: Calculate $\overline{r}$ and M' based on M.
- **Step 3**: Extend Y, M, and $\overline{r}$ in order to get $[Y]_{3072}$, $[M]_{3072}$, and $[\overline{r}]_{3072}$, respectively. This step is only required for Y, M, and $\overline{r}$ whose length are less than 3072 bits, since their largest length are 3072 bits.
- **Step 4**: Calculate MD authentication code using the SHA-256: $[MD]_{256}$ = SHA256 ($[Y]_{3072}||[M]_{3072}||[\overline{r}]_{3072}||[M']_{32}||[L]_{32}||[IV]_{128}$)
- **Step 5**: Build $[P]_{9600}$ = ($[Y]_{3072}||[M]_{3072}||[\overline{r}]_{3072}||[Box]_{384}$), where $[Box]_{384}$ = ($[MD]_{256}||[M']_{32}||[L]_{32}||[\beta]_{64}$) and $[\beta]_{64}$ is a PKCS#7 padding value, i.e., a $[0x0808080808080808]_{64}$ string composed of 8 bytes (0x80). The purpose of $[\beta]_{64}$ is to make the bit length of P a multiple of 128.
- **Step 6**: Calculate C = $[C]_{9600}$ = AES-CBC-ENC ($[P]_{9600}$, $[RSA\_DS\_KEY]_{256}$, $[IV]_{128}$), where C is the ciphertext with a length of 1200 bytes. C can also be calculated as C = $[C]_{9600}$ = ($[\widehat{Y}]_{3072}||[\widehat{M}]_{3072}||[\widehat{\overline{r}}]_{3072}||[\widehat{Box}]_{384}$), where $[\widehat{Y}]_{3072}$, $[\widehat{M}]_{3072}$, $[\widehat{\overline{r}}]_{3072}$, $[\widehat{Box}]_{384}$ are the four sub-parameters

of C, and correspond to the ciphertext of $[Y]_{3072}$, $[M]_{3072}$, $[\overline{r}]_{3072}$, $[Box]_{384}$ respectively.

24.3.4 RSA_DS Operation at the Hardware Level

The hardware operation is triggered each time a digital signature needs to be calculated. The inputs are the pre-generated private key ciphertext C, a unique message X, and IV.

The RSA_DS operation at the hardware level can be divided into the following three stages:

1. **Decryption: Step 7 and 8 in Figure 24.3-1**

   The decryption process is the inverse of Step 6 in Figure 24.3-1. The RSA_DS peripheral will call the AES accelerator to decrypt C in CBC block mode and get the resulting plaintext. The decryption process can be represented by P = AES-CBC-DEC (C, RSA_DS_KEY, IV), where IV (i.e., $[IV]_{128}$) is defined by the user. $[RSA\_DS\_KEY]_{256}$ is provided by the HMAC module, derived from $HMAC\_KEY$ stored in eFuse. $[RSA\_DS\_KEY]_{256}$, as well as $[HMAC\_KEY]_{256}$ are not readable by users. For more information, please refer to Chapter 21 *HMAC Accelerator (HMAC)*.

   With P, the RSA_DS peripheral can derive $[Y]_{3072}$, $[M]_{3072}$, $[\overline{r}]_{3072}$, $[M']_{32}$, $[L]_{32}$, MD authentication code, and the padding value $[\beta]_{64}$. This process is the inverse of Step 5.

2. **Check: Step 9 and 10 in Figure 24.3-1**

   The RSA_DS peripheral will perform two checks: MD check and padding check. The padding check is not shown in Figure 24.3-1, as it happens at the same time as the MD check.

   - MD check: The RSA_DS peripheral calls SHA-256 to calculate the hash value $[CALC\_MD]_{256}$ ($[CALC\_MD]_{256}$ is calculated the same way and with same parameters as $[MD]_{256}$, see step 4). Then, $[CALC\_MD]_{256}$ is compared against the MD authentication code $[MD]_{256}$ from step 4. Only when the two match does the MD check pass.
   - Padding check: The RSA_DS peripheral checks if $[\beta]_{64}$ complies with the aforementioned PKCS#7 format. Only when $[\beta]_{64}$ complies with the format does the padding check pass.

   The RSA_DS peripheral will only perform subsequent operations if MD check passes. If the padding check fails, a warning is generated, but it does not affect the subsequent operations.

3. **Calculation: Step 11 and 12 in Figure 24.3-1**

   The RSA_DS peripheral treats X (input by the user) and Y, M, $\overline{r}$ (decrypted in step 8) as big numbers. With M', all operands to perform X^Y mod M are in place. The operand length is defined by L only. The RSA_DS peripheral will calculate the signed result Z by calling RSA to perform $Z = X^Y \bmod M$.

24.3.5 RSA_DS Operation at the Software Level

The software steps below should be followed each time a digital signature needs to be calculated. The inputs are the pre-generated private key ciphertext C, a unique message X, and IV. These software steps trigger the hardware steps described in Section 24.3.4.

We assume that the software has called the HMAC peripheral and the HMAC peripheral has calculated RSA_DS_KEY based on $HMAC\_KEY$.

1. **Prerequisites**: Prepare operands C, X, IV according to Section 24.3.3.
2. **Activate the RSA_DS peripheral**: Write 1 to DS_SET_START_REG.

3. **Check if RSA_DS_KEY is ready**: Poll DS_QUERY_BUSY_REG until the software reads 0.

   If the software does not read 0 in DS_QUERY_BUSY_REG after approximately 1 ms, it indicates a problem with HMAC initialization. In such a case, the software can read register DS_QUERY_KEY_WRONG_REG to get more information:

   - If the software reads 0 in DS_QUERY_KEY_WRONG_REG, it indicates that the HMAC peripheral has not been called.
   - If the software reads any value from 1 to 15 in DS_QUERY_KEY_WRONG_REG, it indicates that HMAC was called, but the RSA_DS peripheral did not successfully get RSA_DS_KEY from the HMAC peripheral. This may indicate that the HMAC operation has been interrupted due to a software concurrency problem.

4. **Configure register**: Write the content in the IV block to register DS_IV_*m*_REG (*m*: 0 ~ 3). For more information on the IV block, please refer to Chapter 19 *AES Accelerator (AES)*.

5. **Write X to memory block DS_X_MEM**: Write X_i ($i \in \{0, 1, \ldots, n-1\}$), where $n = \frac{N}{32}$, to memory block DS_X_MEM whose capacity is 96 words. Each word can store one base-b digit. The memory block uses the little endian format for storage, i.e., the least significant digit of the operand is in the lowest address. Words in DS_X_MEM block after the configured length of X (N bits, as described in Section 24.3.2), are ignored.

6. **Write C to corresponding memory blocks**: Write the four sub-parameters of C to corresponding memory blocks:

   - Write $\widehat{Y_i}$ ($i \in \{0, 1, \ldots, 95\}$) to DS_Y_MEM.
   - Write $\widehat{M_i}$ ($i \in \{0, 1, \ldots, 95\}$) to DS_M_MEM.
   - Write $\widehat{r}_i$ ($i \in \{0, 1, \ldots, 95\}$) to DS_RB_MEM.
   - write $\widehat{Box_i}$ ($i \in \{0, 1, \ldots, 11\}$) to DS_BOX_MEM.

     The capacity of DS_Y_MEM, DS_M_MEM, and DS_RB_MEM is 96 words, whereas the capacity of DS_BOX_MEM is only 12 words. Each word can store one base-b digit. The memory blocks use the little endian format for storage, i.e., the least significant digit of the operand is in the lowest address.

7. **Start RSA_DS operation**: Write 1 to register DS_SET_ME_REG.

8. **Wait for the operation to be completed**: Poll register DS_QUERY_BUSY_REG until the software reads 0.

9. **Query check result**: Read register DS_QUERY_CHECK_REG and conduct subsequent operations as illustrated below based on the return value:

   - If the value is 0, it indicates that both the padding check and MD check pass. Users can continue to get the signed result Z.
   - If the value is 1, it indicates that the padding check passes but MD check fails. The signed result Z is invalid. The operation will resume directly from Step 11.
   - If the value is 2, it indicates that the padding check fails but the MD check passes. Users can continue to get the signed result Z. But please note that the data does not comply with the aforementioned PKCS#7 padding format, which may not be what you want.
   - If the value is 3, it indicates that both the padding check and MD check fail. In this case, some fatal errors have occurred and the signed result Z is invalid. The operation will resume directly from Step

11.

10. **Read the signed result**: Read the signed result Z_i ($i \in \{0, 1, \ldots, n-1\}$), where $n = \frac{N}{32}$, from memory block DS_Z_MEM. The memory block stores Z in little-endian byte order.
11. **Exit the operation**: Write 1 to DS_SET_FINISH_REG, and then poll DS_QUERY_BUSY_REG until the software reads 0.

After the operation, all the input/output registers and memory blocks are cleared.

24.4 Memory Summary

The addresses in this section are relative to the RSA Digital Signature Peripheral (RSA_DS) base address provided in Table 4.3-2 in Chapter 4 *System and Memory*.

Name	Description	Size (byte)	Starting Address	Ending Address	Access
DS_Y_MEM	Memory block Y	384	0x0000	0x017F	WO
DS_M_MEM	Memory block M	384	0x0200	0x037F	WO
DS_RB_MEM	Memory block $\overline{r}$	384	0x0400	0x057F	WO
DS_BOX_MEM	Memory block Box	48	0x0600	0x062F	WO
DS_X_MEM	Memory block X	384	0x0800	0x097F	WO
DS_Z_MEM	Memory block Z	384	0x0A00	0x0B7F	RO

24.5 Register Summary

The addresses in this section are relative to the RSA Digital Signature Peripheral (RSA_DS) base address provided in Table 4.3-2 in Chapter 4 *System and Memory*.

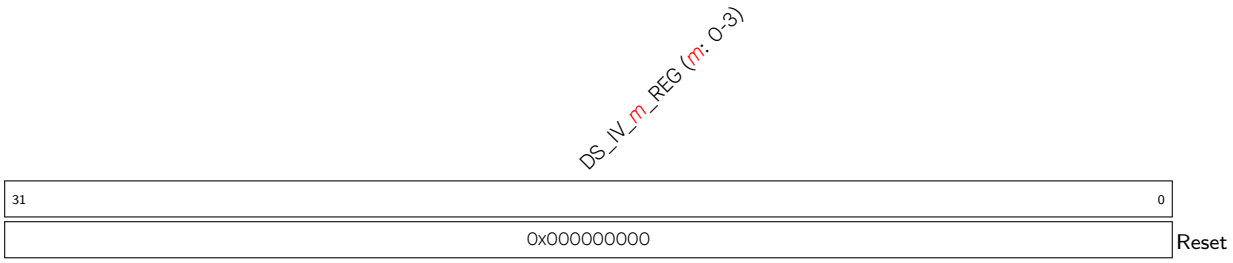
The abbreviations given in Column **Access** are explained in Section *Access Types for Registers*.

Name	Description	Address	Access
Configuration Registers			
DS_IV_0_REG	IV block data	0x0630	WO
DS_IV_1_REG	IV block data	0x0634	WO
DS_IV_2_REG	IV block data	0x0638	WO
DS_IV_3_REG	IV block data	0x063C	WO
Status/Control Registers			
DS_SET_START_REG	Activates the RSA_DS peripheral	0x0E00	WO
DS_SET_ME_REG	Starts RSA_DS operation	0x0E04	WO
DS_SET_FINISH_REG	Ends RSA_DS operation	0x0E08	WO
DS_QUERY_BUSY_REG	Status of the RSA_DS peripheral	0x0E0C	RO
DS_QUERY_KEY_WRONG_REG	Checks the reason why RSA_DS_KEY is not ready	0x0E10	RO
DS_QUERY_CHECK_REG	Queries RSA_DS check result	0x0E14	RO
Version control register			
DS_DATE_REG	Version control register	0x0E20	W/R

24.6 Registers

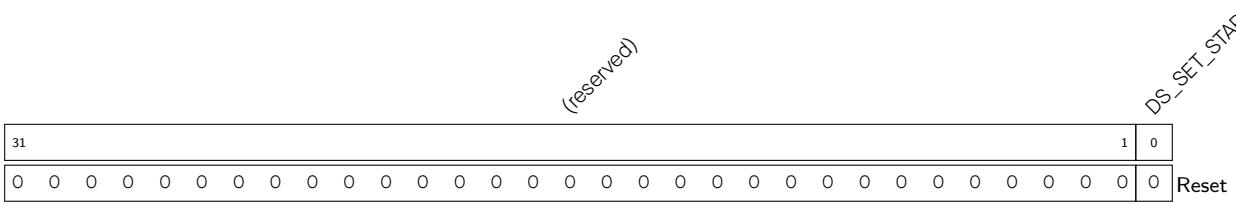
The addresses in this section are relative to the RSA Digital Signature Peripheral (RSA_DS) base address provided in Table 4.3-2 in Chapter 4 *System and Memory*.

Register 24.1. DS_IV_*m*_REG (*m*: 0-3) (0x0630+4*m*)



DS_IV_*m*_REG (*m*: 0-3) Writes IV block data. (WO)

Register 24.2. DS_SET_START_REG (0x0E00)



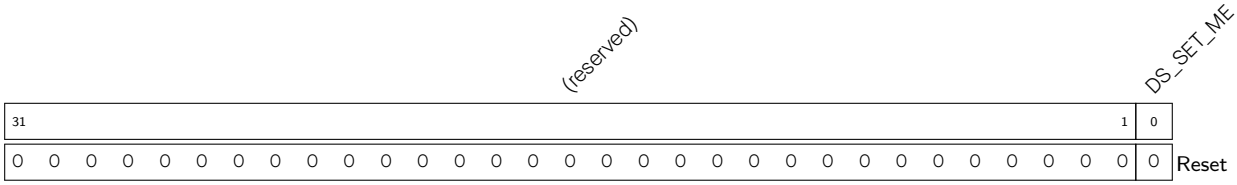
DS_SET_START Configures whether to activate the RSA_DS peripheral.

0: No effect

1: Activate the RSA_DS peripheral

(WO)

Register 24.3. DS_SET_ME_REG (0x0E04)



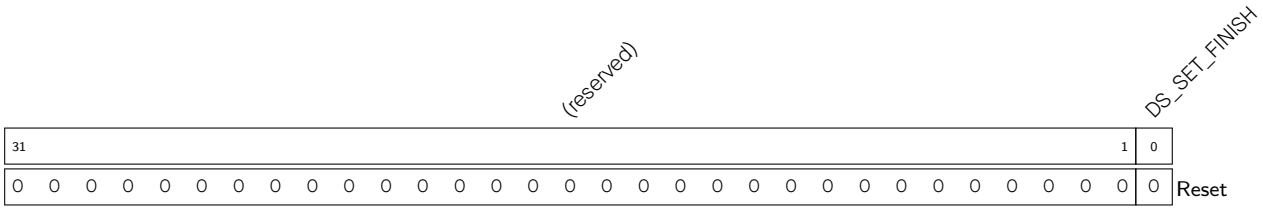
DS_SET_ME Configures whether to start the RSA_DS operation.

0: No effect

1: Start the RSA_DS operation

(WO)

Register 24.4. DS_SET_FINISH_REG (0x0E08)



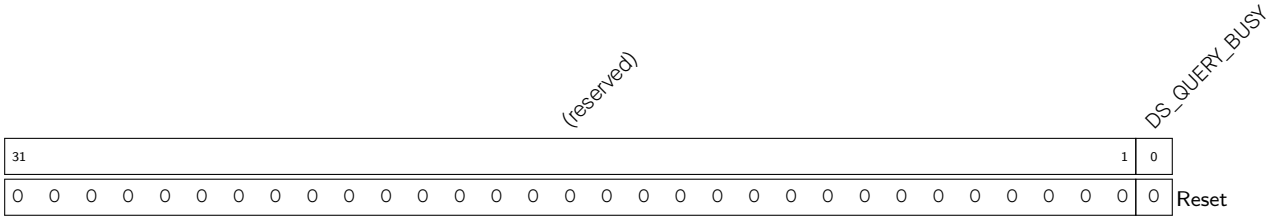
DS_SET_FINISH Configures whether to end the RSA_DS operation.

0: No effect

1: End the RSA_DS operation

(WO)

Register 24.5. DS_QUERY_BUSY_REG (0x0E0C)



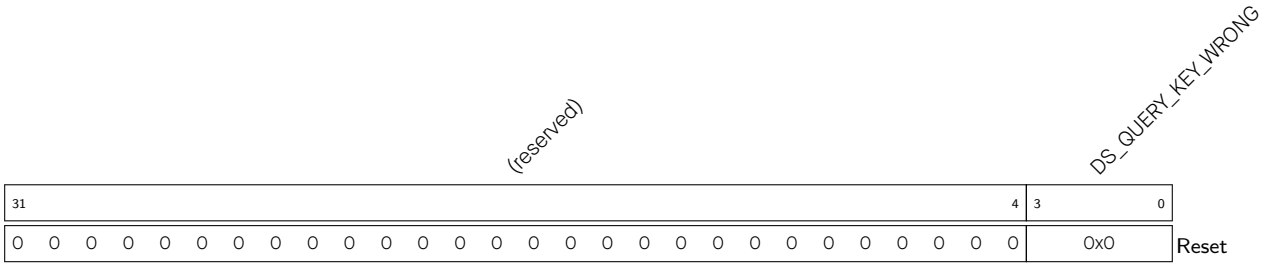
DS_QUERY_BUSY Represents whether the RSA_DS peripheral is idle.

0: The RSA_DS peripheral is idle

1: The RSA_DS peripheral is busy

(RO)

Register 24.6. DS_QUERY_KEY_WRONG_REG (0x0E10)



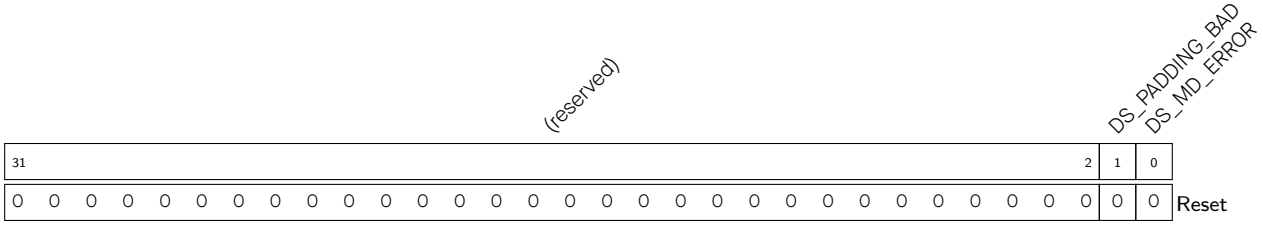
DS_QUERY_KEY_WRONG Represents the specific problem with HMAC initialization.

0: HMAC is not called

1-15: HMAC was activated, but the RSA_DS peripheral did not successfully receive RSA_DS_KEY from the HMAC peripheral.

(RO)

Register 24.7. DS_QUERY_CHECK_REG (0x0E14)



DS_PADDING_BAD Represents whether or not the padding check passes.

0: The padding check passes

1: The padding check fails

(RO)

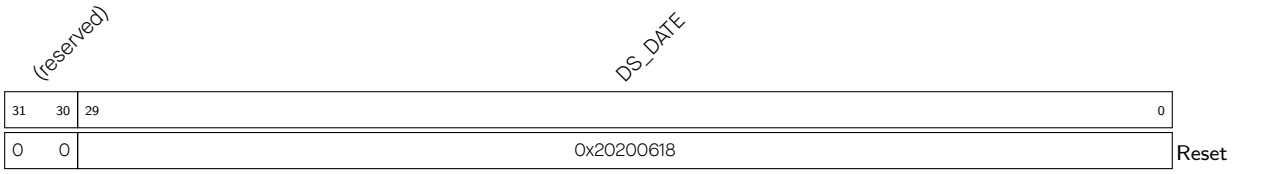
DS_MD_ERROR Represents whether or not the MD check passes.

0: The MD check passes

1: The MD check fails

(RO)

Register 24.8. DS_DATE_REG (0x0E20)



DS_DATE Version control register. (R/W)

Chapter 25

ECDSA Digital Signature Peripheral (ECDSA_DS)

25.1 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-H2'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 being protected without sacrificing performance.

25.2 Features

ESP32-H2'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

25.3 ECDSA Basics

25.3.1 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.

25.3.2 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 OTP in ESP32-H2. (Please refer to Chapter 5 *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.

25.3.3 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.

25.3.4 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.

25.4 Functional Description

This section describes the details of ESP32-H2's ECDSA_DS module.

25.4.1 ECDSA_DS Working Modes

The ECDSA_DS module integrated in the ESP32-H2 has two working modes, which are Signature Generation mode and Signature Verification mode.

Users can select the working mode for ECDSA_DS by configuring the ECDSA_WORK_MODE according to Table 25.4-1 below.

Table 25.4-1. ECDSA_DS Working Mode

ECDSA_WORK_MODE	Working Mode
0	Signature Verification
1	Signature Generation

Users can select the elliptic curves used by configuring the ECDSA_ECC_CURVE according to Table 25.4-2.

Table 25.4-2. 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 the ECDSA_SHA_MODE according to Table 25.4-3.

Table 25.4-3. 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 25.4-4 below.

Table 25.4-4. 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 & POST Stage.

25.4.2 Data and Data Block

ESP32-H2'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:

$$D[255:0] = D[7][31:0], D[6][31:0], D[5][31:0], D[4][31:0], D[3][31:0], D[2][31:0], D[1][31:0], D[0][31:0]$$

25.4.2.1 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.

25.4.2.2 Reading Data

Reading data means reading data from the starting address of an ECDSA_DS memory block and using this data as the output from ECDSA_DS. To be specific, reading data from an ECDSA_DS memory block means reading $D[n][31:0]$ from the "starting address of this ECDSA_DS memory block + $4 \times n$". For a 256-bit long data 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.

25.4.2.3 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".

25.4.2.4 Parsing the Message

The message and its padding must be parsed into N 512-bit message blocks: $M^{(1)}$, $M^{(2)}$, ..., $M^{(N)}$.

For more details about "parsing the message", please refer to FIPS PUB 180-4 Spec > Section "Parsing the Message".

For more information about "message block", please refer to FIPS PUB 180-4 Spec > Section "Glossary of Terms and Acronyms".

25.4.3 Security Features

To ensure the security of the operation process, the ECDSA_DS module implements a variety of security functions.

25.4.3.1 High Anti-Attack Performance

ESP32-H2's ECDSA_DS module leverages ECC's enhanced anti-attack performance (refer to Section 20.4.3 *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-H2's ECDSA_DS module strong anti-attack performance.

25.4.3.2 State-Dependent Register Access Control

ESP32-H2'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 25.4-4.

For detailed information on the state-dependent access control of each ECDSA_DS register, please refer to Section Register Summary.

25.4.3.3 Hardware Occupation

During the ECDSA_DS operation, the following hardware modules will be occupied by ESP32-H2's ECDSA_DS module:

- SHA Accelerator
- RSA Accelerator
- ECC Accelerator

Among them, the SHA accelerator will be released when the ECDSA_SHA_RELEASE_INT is triggered. While, the RSA accelerator and 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.

At the end of the hardware occupation, the occupied module will be automatically reset. In addition, when user performs a software reset to the master module, all the occupied modules will be reset at the same time.

25.5 Programming Procedures

25.5.1 ECDSA_DS Process

The overall ECDSA_DS process consists of the following six stages.

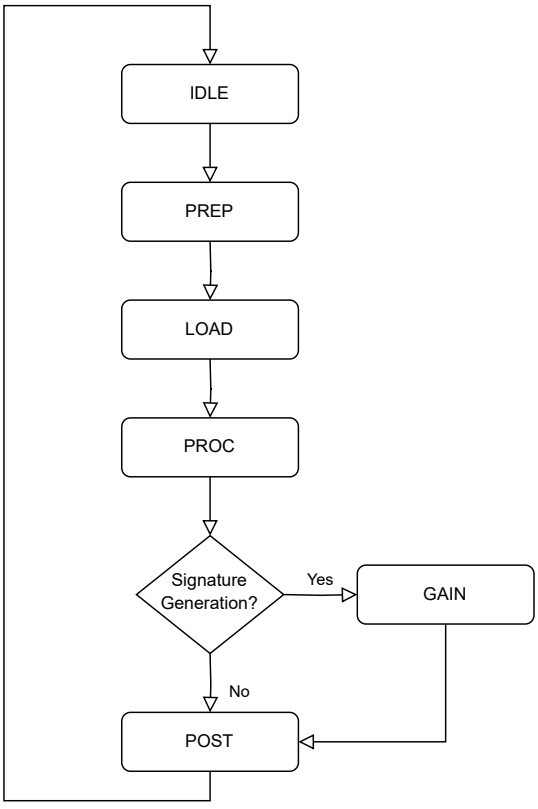


Figure 25.5-1. ECDSA_DS Process

The detailed programming procedures of each stage are described in the following sections.

25.5.1.1 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, user need to write the key value in KEY*n* (n = 0 ~ 5), and set the corresponding EFUSE_KEY_PURPOSE_*n* as ECDSA_KEY. Please refer to Chapter 5 *eFuse Controller (EFUSE)* for more detailed configuration steps.
2. Configure the configuration register, including the following fields:
   (a) ECDSA_WORK_MODE: Choose the working mode of ECDSA_DS.
   (b) ECDSA_ECC_CURVE: Choose the elliptic curve of ECDSA_DS.
   (c) ECDSA_SOFTWARE_SET_Z: Choose to use direct input z.
3. Configure the register field ECDSA_START to enter the PREP stage.

25.5.1.2 PREP Stage

In the PREP stage, the ECDSA_STATE_REG is BUSY, and ECDSA_DS performs preparation.

1. 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.

25.5.1.3 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_Z_MEM.
   - ECDSA_DS SHA interface: generate z from the message. For details, please refer to Section 25.5.1.7.
2. According to the selected ECDSA_WORK_MODE, configure as follows:
   - signature Verification:
     (a) write signature (r, s) to ECDSA_R_MEM and ECDSA_S_MEM.
     (b) write public key (Qx, Qy) to ECDSA_QX_MEM and ECDSA_QY_MEM.
   - signature Generation:
     (a) no other configuration is required.
3. Write 1 to ECDSA_LOAD_DONE, indicating the configuration is done. Then ECDSA_DS will automatically enter the PROC stage.

25.5.1.4 PROC Stage

In the PROC stage, the ECDSA_STATE_REG is BUSY, and ECDSA_DS performs operation based on the selected working mode.

1. 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

25.5.1.5 GAIN Stage

When the Signature Generation 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_R_MEM and ECDSA_S_MEM.
   - read public key (Qx, Qy) from ECDSA_QX_MEM and ECDSA_QY_MEM.
2. Write 1 to ECDSA_GET_DONE, indicating the GAIN stage is done. Then ECDSA_DS will automatically enter the POST stage.

25.5.1.6 POST Stage

In the POST stage, the ECDSA_STATE_REG is BUSY, and ECDSA_DS performs some wrap-up work of the operation.

1. 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.

25.5.1.7 ECDSA_DS SHA Interface

ESP32-H2'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 25.4.2.3.
2. Parse the message and its padding into message blocks. See details in Section 25.4.2.4.
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 the interface has completed the operation for the current message block.
6. Process the next message block:
   - If yes, go back to Step 3.

- If no more message block, exit.

Note:

1. In this step, the software can also write the next message block (to be processed) in register ECDSA_MEM_M, if any, while the interface starts SHA operation, to save time.
2. You are resuming the ECDSA_DS SHA interface with the previously paused operation.

25.5.2 Clocks and Resets

ESP32-H2'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 in the PCR_ECDSA_CONF_REG register. For details on how to configure the ECDSA_DS clock and reset, please refer to Chapter 7 *Reset and Clock*.

25.5.3 Interrupts

ESP32-H2's ECDSA_DS module can generate one interrupt signal ECDSA_INTR and send it to Interrupt Matrix.

The ECDSA_DS module has two interrupt sources that can generate the ECDSA_INTR interrupt signal:

- ECDSA_CALC_DONE_INT: triggered on the completion of an ECC operation. This interrupt source is configured by the following registers:
  - ECDSA_CALC_DONE_INT_RAW: stores the raw interrupt status of ECDSA_CALC_DONE_INT.
  - ECDSA_CALC_DONE_INT_ST: indicates the status of the ECDSA_CALC_DONE_INT interrupt. This field is generated by enabling/disabling the ECDSA_CALC_DONE_INT_RAW field.
  - ECDSA_CALC_DONE_INT_ENA: enables/disables the ECDSA_CALC_DONE_INT interrupt.
  - ECDSA_CALC_DONE_INT_CLR: set this bit to clear the ECDSA_CALC_DONE_INT interrupt status. By setting this bit to 1, fields ECDSA_CALC_DONE_INT_RAW and ECDSA_CALC_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 9 *Interrupt Matrix (INTMTX)* > Section 9.2 *Interrupt Terminology in ESP32-H2*.

25.6 Memory Blocks

ECDSA_DS's memory blocks store input data and output data of the ECDSA_DS operation.

Table 25.6-1. ECDSA_DS Memory Blocks

Memory	Size (byte)	Starting Address*	Ending Address *	Access
ECDSA_MEM_M	64	0x280	0x2BF	R/W
ECDSA_MEM_R	32	0xA00	0xA1F	R/W
ECDSA_MEM_S	32	0xA20	0xA3F	R/W
ECDSA_MEM_Z	32	0xA40	0xA5F	R/W
ECDSA_MEM_Qx	32	0xA60	0xA7F	R/W
ECDSA_MEM_Qy	32	0xA80	0xA9F	R/W

* Address offset related to the ECDSA_DS base address is provided in Table 4.3-2 in Chapter 4 *System and Memory*.

25.7 Register Summary

The addresses in this section are relative to the ECDSA_DS base address provided in Table 4.3-2 in Chapter 4 *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.

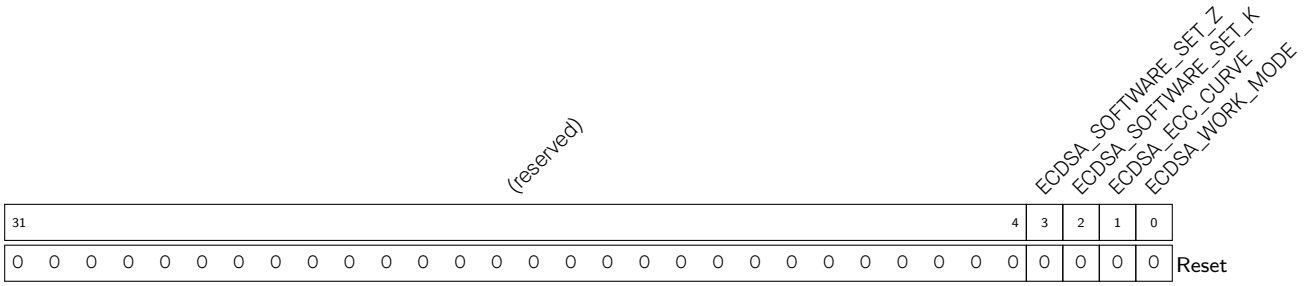
Name	Description	Address	Access			
			PI	PL	PG	P
Data Memory	See Table 20.3-1.					
Configuration Registers						
ECDSA_CONF_REG	ECDSA_DS configuration register	0x0004	R/W	N/A		
ECDSA_START_REG	ECDSA_DS start register	0x001C	WT			N
Clock and Reset Register						
ECDSA_CLK_REG	ECDSA_DS clock gate register	0x0008	R/W	N/A		
Interrupt Registers						
ECDSA_INT_RAW_REG	ECDSA_DS interrupt raw register	0x000C	RO/WTC/SS			
ECDSA_INT_ST_REG	ECDSA_DS interrupt status register	0x0010	RO			
ECDSA_INT_ENA_REG	ECDSA_DS interrupt enable register	0x0014	R/W			
ECDSA_INT_CLR_REG	ECDSA_DS interrupt clear register	0x0018	WT			
Status Registers						
ECDSA_STATE_REG	ECDSA_DS state register	0x0020	RO			
Result Register						
ECDSA_RESULT_REG	ECDSA_DS result register	0x0024	RO/SS			N
SHA Registers						
ECDSA_SHA_MODE_REG	ECDSA_DS controlling SHA register (Hash algorithm)	0x0200	N/A	R/W	N/A	
ECDSA_SHA_START_REG	ECDSA_DS controlling SHA register (operation)	0x0210	N/A	WT	N/A	
ECDSA_SHA_CONTINUE_REG	ECDSA_DS controlling SHA register (operation)	0x0214	N/A	WT	N/A	
ECDSA_SHA_BUSY_REG	ECDSA_DS controlling SHA status register	0x0218	N/A	RO	N/A	
Version Register						
ECDSA_DATE_REG	Version control register	0x00FC	R/W	N/A		

25.8 Registers

The addresses in this section are relative to the ECDSA_DS base address provided in Table 4.3-2 in Chapter 4 *System and Memory*.

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

Register 25.1. ECDSA_CONF_REG (0x0004)



ECDSA_WORK_MODE Configures the working mode of ECDSA_DS.

0: Signature Generation mode

1: Signature Verification mode

(R/W)

ECDSA_ECC_CURVE Configures the elliptic curve used.

0: P-192

1: P-256

(R/W)

ECDSA_SOFTWARE_SET_K Configures how the parameter k is set.

0: automatically generated by TRNG

1: written by software

(R/W)

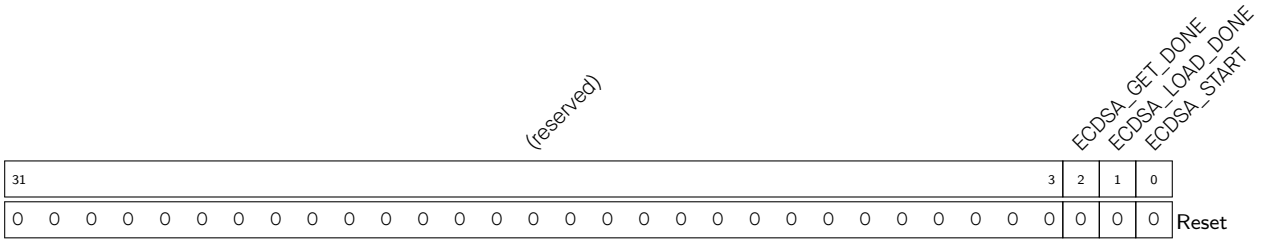
ECDSA_SOFTWARE_SET_Z Configures how the parameter z is set.

0: generated from SHA result

1: written by software

(R/W)

Register 25.2. 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

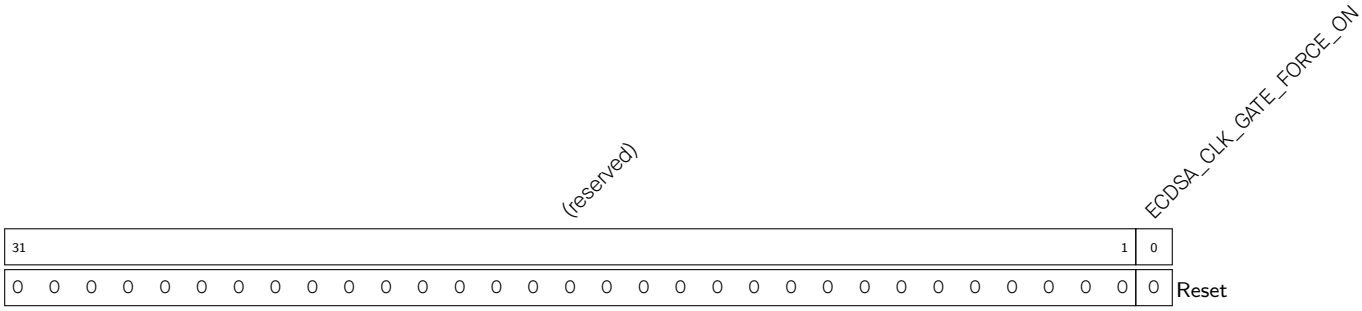
1: start the ECDSA_DS operation

(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 25.3. ECDSA_CLK_REG (0x0008)



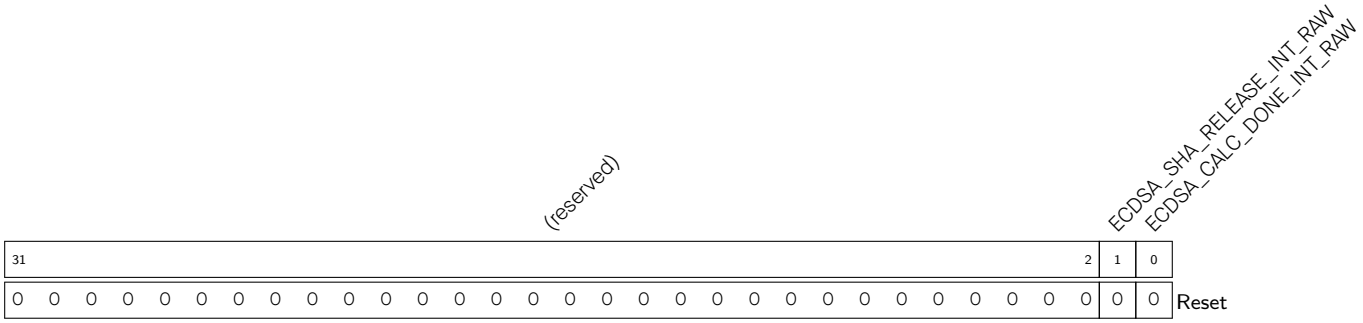
ECDSA_CLK_GATE_FORCE_ON Configures whether to force on ECC memory clock gate.

0: no effect

1: force on

(R/W)

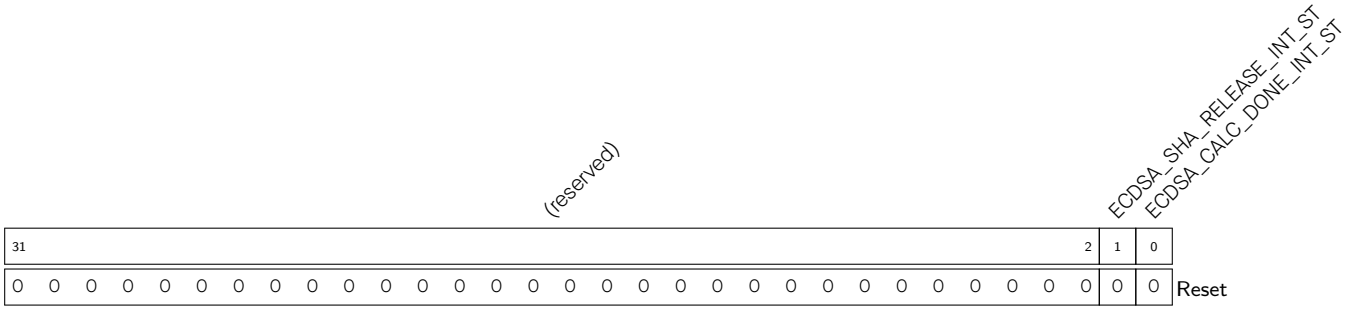
Register 25.4. ECDSA_INT_RAW_REG (0x000C)



ECDSA_CALC_DONE_INT_RAW The raw interrupt status of the ECDSA_CALC_DONE_INT interrupt. (RO/WTC/SS)

ECDSA_SHA_RELEASE_INT_RAW The raw interrupt status of the ECDSA_SHA_RELEASE_INT interrupt. (RO/WTC/SS)

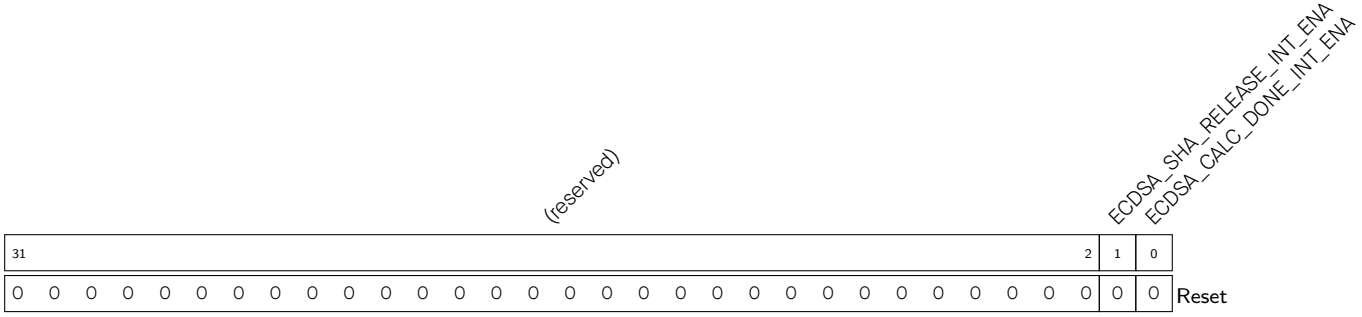
Register 25.5. ECDSA_INT_ST_REG (0x0010)



ECDSA_CALC_DONE_INT_ST The masked interrupt status of the ECDSA_CALC_DONE_INT interrupt. (RO)

ECDSA_SHA_RELEASE_INT_ST The masked interrupt status of ECDSA_SHA_RELEASE_INT interrupt. (RO)

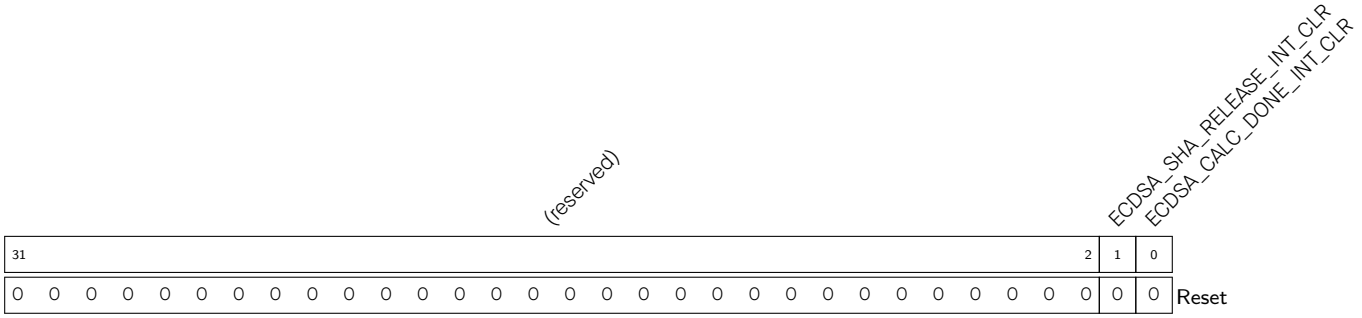
Register 25.6. ECDSA_INT_ENA_REG (0x0014)



ECDSA_CALC_DONE_INT_ENA Write 1 to enable the ECDSA_CALC_DONE_INT interrupt. (R/W)

ECDSA_SHA_RELEASE_INT_ENA Write 1 to enable the ECDSA_SHA_RELEASE_INT interrupt. (R/W)

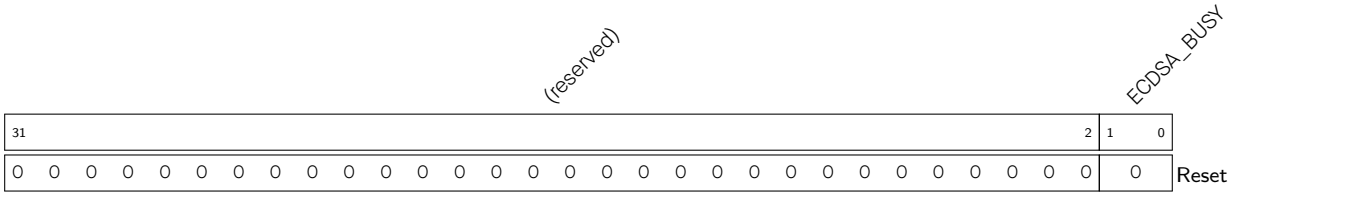
Register 25.7. ECDSA_INT_CLR_REG (0x0018)



ECDSA_CALC_DONE_INT_CLR Write 1 to clear the ECDSA_CALC_DONE_INT interrupt. (WT)

ECDSA_SHA_RELEASE_INT_CLR Write 1 to clear the ECDSA_SHA_RELEASE_INT interrupt. (WT)

Register 25.8. ECDSA_STATE_REG (0x0020)



ECDSA_BUSY Represents the working state of ECDSA_DS.

0: IDLE

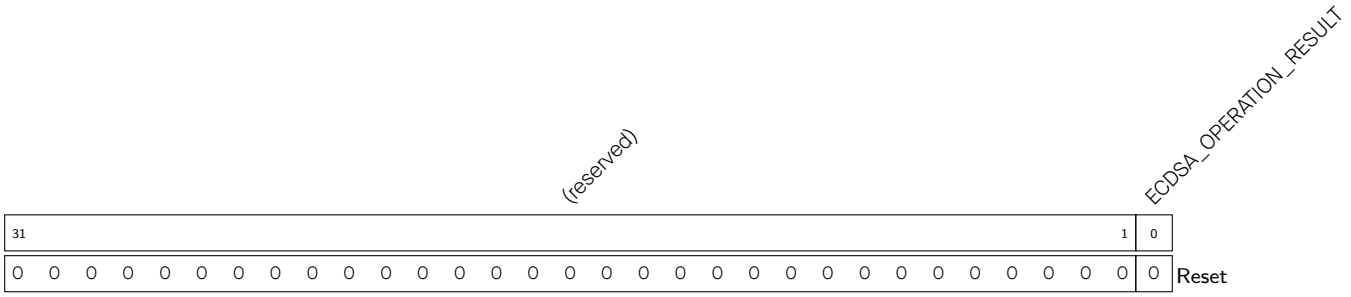
1: LOAD

2: GET

3: BUSY

(RO)

Register 25.9. ECDSA_RESULT_REG (0x0024)



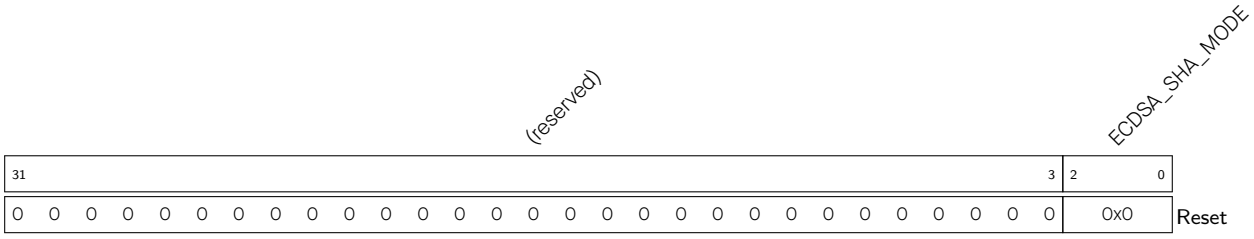
ECDSA_OPERATION_RESULT Indicates if the ECDSA_DS operation is successful.

0: not successful

1: successful

Only valid when the ECDSA_DS operation is done. (RO/SS)

Register 25.10. ECDSA_SHA_MODE_REG (0x0200)



ECDSA_SHA_MODE Configures SHA algorithms for message hash.

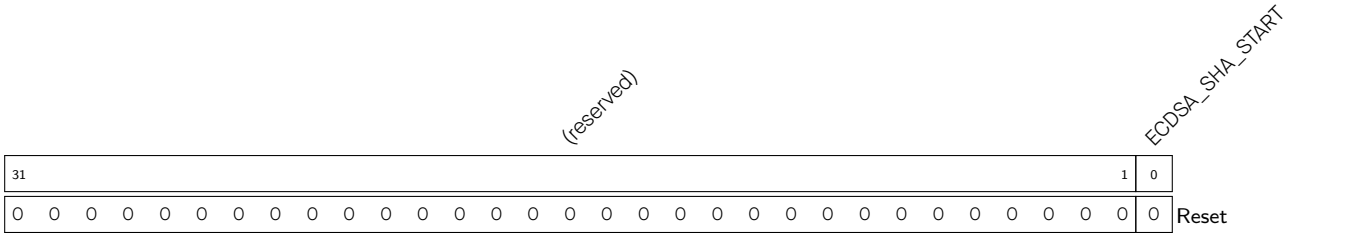
1: SHA-224

2: SHA-256

Others: invalid

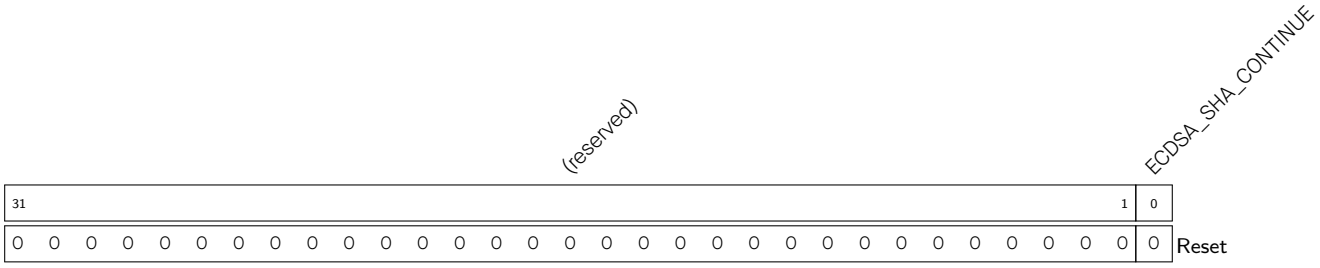
(R/W)

Register 25.11. ECDSA_SHA_START_REG (0x0210)



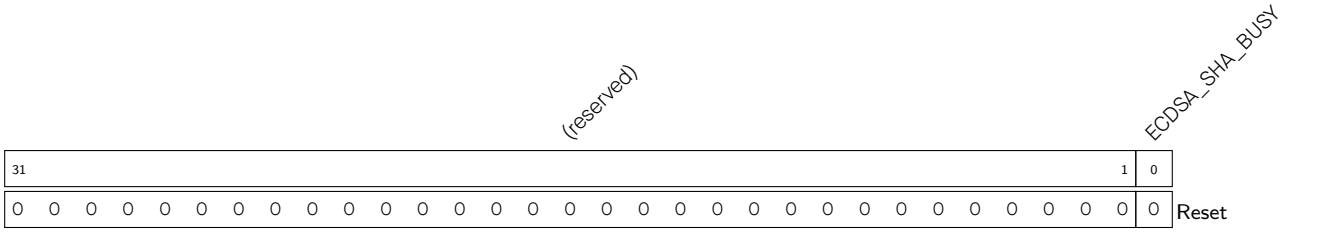
ECDSA_SHA_START Write 1 to start the first SHA operation in the ECDSA_DS process. This bit will be self-cleared after configuration. (WT)

Register 25.12. 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 25.13. ECDSA_SHA_BUSY_REG (0x0218)



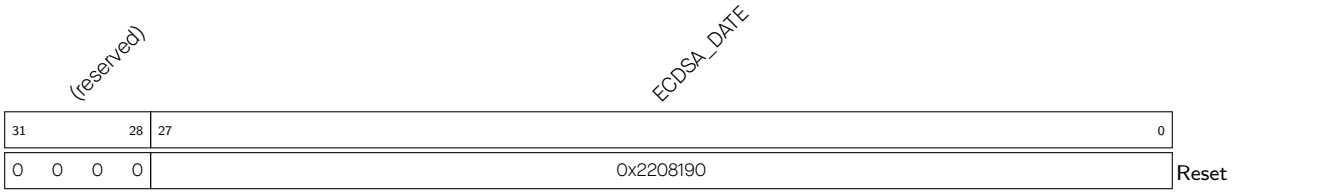
ECDSA_SHA_BUSY Represents the working state of the SHA accelerator in the ECDSA_DS process.

0: IDLE

1: BUSY

(RO)

Register 25.14. ECDSA_DATE_REG (0x00FC)



ECDSA_DATE The ECDSA_DS version control register. (R/W)

Chapter 26

External Memory Encryption and Decryption (XTS_AES)

26.1 Overview

The ESP32-H2 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). Users can store proprietary firmware and sensitive data (e.g., credentials for gaining access to a private network) to the external flash.

26.2 Features

- General XTS-AES algorithm, compliant with IEEE Std 1619-2007
- Software-based manual encryption
- High-speed auto decryption without software's participation
- Encryption and decryption functions jointly enabled/disabled by registers configuration, eFuse parameters, and boot mode
- Configurable Anti-DPA

26.3 Module Structure

The External Memory Encryption and Decryption module consists of two blocks, namely the Manual Encryption block and Auto Decryption block. The module architecture is shown in Figure 26.3-1.

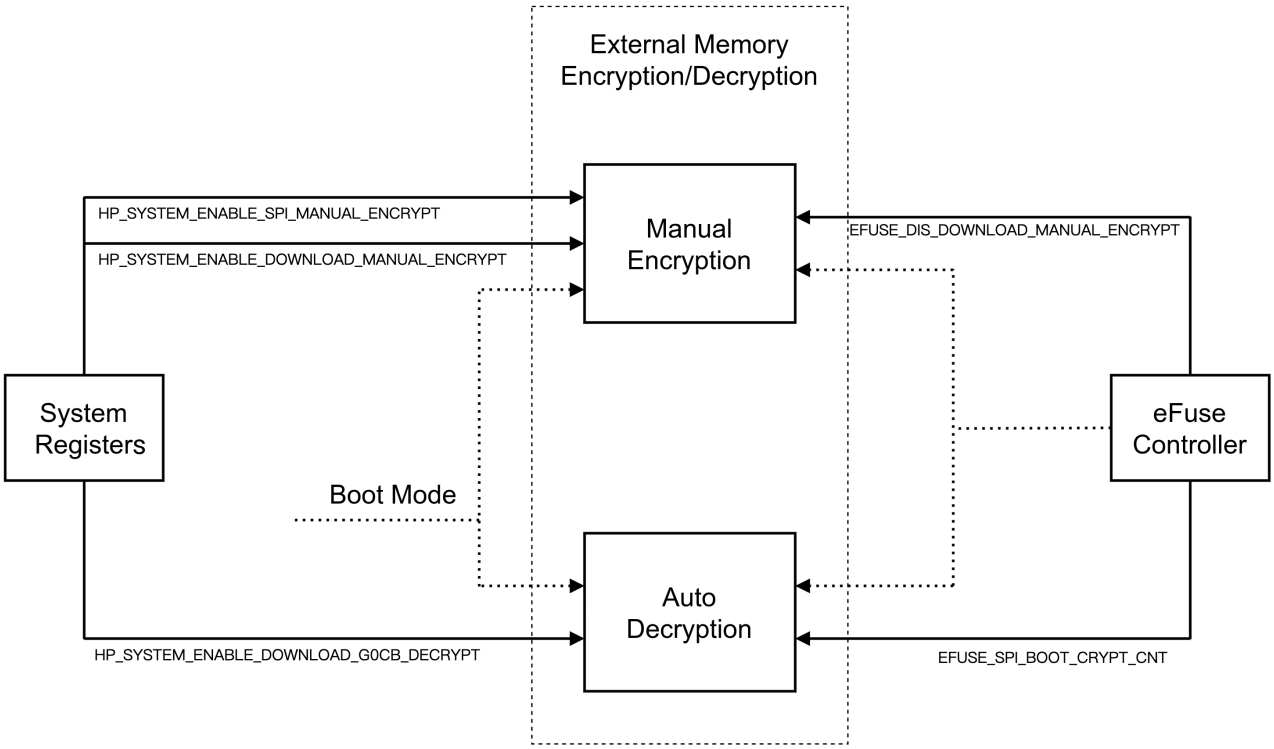


Figure 26.3-1. Architecture of the External Memory Encryption and Decryption

The Manual Encryption block can encrypt instructions and data, which will then be written to the external flash as ciphertext via SPI1.

In the System Registers peripheral (see 16 *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_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 5 *eFuse Controller (EFUSE)*.

26.4 Functional Description

26.4.1 XTS Algorithm

Both manual encryption and auto 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.

26.4.2 Key

The Manual 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 5.3-2 *Secure Key Purpose Values*). 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 26.4-1. In each case, the Key can be uniquely determined.

Table 26.4-1. 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 5.3-2 *Secure Key Purpose Values* in Chapter 5 *eFuse Controller (EFUSE)*.

26.4.3 Target Memory Space

The target memory space refers to a continuous address space in the external memory (flash) where the ciphertext is stored. The target memory space can be uniquely determined by two relevant parameters: size and base address, whose definitions are listed below.

- 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, 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 Decryption blocks, these parameters are automatically determined by hardware. For Manual Encryption blocks, these parameters should be configured by users.

Note:

The "tweak" defined in Section *Data units and tweaks* of IEEE Std 1619-2007 is a 128-bit non-negative integer

($tweak$), which can be generated according to $tweak$ = ($base\_addr$ & 0x00FFFF80). The lowest 7 bits and the highest 97 bits in $tweak$ are always zero.

26.4.4 Data Writing

For Auto 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 plaintext memory region starting at SPI_MEM_XTS_PLAIN_BASE_REG, which provides 16 consecutive 32-bit words 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 this memory region, 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 the plaintext memory region works:

Assume a word in the target memory space is stored in $address$, define $offset$ = $address$%64, then the word will be stored at SPI_MEM_XTS_PLAIN_BASE_REG + $offset$.

For example, when the $size$ is 64, all 16 words in the plaintext memory region will be used. The mapping between $offset$ and addresses in the plaintext memory region is shown in Table 26.4-2.

Table 26.4-2. Mapping Between Offsets and Plaintext Memory Addresses

$offset$	Address	$offset$	Address
0x00	SPI_MEM_XTS_PLAIN_BASE_REG + 0x00	0x20	SPI_MEM_XTS_PLAIN_BASE_REG + 0x20
0x04	SPI_MEM_XTS_PLAIN_BASE_REG + 0x04	0x24	SPI_MEM_XTS_PLAIN_BASE_REG + 0x24
0x08	SPI_MEM_XTS_PLAIN_BASE_REG + 0x08	0x28	SPI_MEM_XTS_PLAIN_BASE_REG + 0x28
0x0C	SPI_MEM_XTS_PLAIN_BASE_REG + 0x0C	0x2C	SPI_MEM_XTS_PLAIN_BASE_REG + 0x2C
0x10	SPI_MEM_XTS_PLAIN_BASE_REG + 0x10	0x30	SPI_MEM_XTS_PLAIN_BASE_REG + 0x30
0x14	SPI_MEM_XTS_PLAIN_BASE_REG + 0x14	0x34	SPI_MEM_XTS_PLAIN_BASE_REG + 0x34
0x18	SPI_MEM_XTS_PLAIN_BASE_REG + 0x18	0x38	SPI_MEM_XTS_PLAIN_BASE_REG + 0x38
0x1C	SPI_MEM_XTS_PLAIN_BASE_REG + 0x1C	0x3C	SPI_MEM_XTS_PLAIN_BASE_REG + 0x3C

26.4.5 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.

26.4.6 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

  If the first bit or the third bit in parameter SPI_BOOT_CRYPT_CNT (3 bits) is set to 1, then the Auto Decryption block can be enabled. Otherwise, it is not operational.

- In Download Boot mode

  If 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 will automatically decrypt the ciphertext if the CPU reads instructions/data from the external memory via cache to retrieve the instructions/data. The entire decryption process does not need software participation and is transparent to the cache. The software can by no means obtain the decryption Key during the process.
- When the Auto Decryption block is disabled, it does not have any effect on the contents stored in the external memory, no matter if they are encrypted or not. Therefore, what the CPU reads via cache is the original infor-

mation stored in the external memory.

26.5 Software Process

When the Manual Encryption block operates, software needs to be involved in the process. The steps are as follows:

1. Configure XTS_AES:
   - Set register SPI_MEM_XTS_PHYSICAL_ADDRESS_REG to $base\_addr$.
   - Set register SPI_MEM_XTS_LINESIZE_REG to $\frac{size}{32}$.

   For definitions of $base\_addr$ and $size$, please refer to Section 26.4.3.
2. Write plaintext instructions/data to the plaintext memory region starting at SPI_MEM_XTS_PLAIN_BASE_REG. For detailed information, please refer to Section 26.4.4.
   Please write data according to your actual needs, and the unused words could be set to arbitrary values.
3. Wait for Manual Encryption block to be idle. Poll register SPI_MEM_XTS_STATE_REG until it reads 0 which indicates the Manual Encryption block is idle.
4. Trigger manual encryption by writing 1 to register SPI_MEM_XTS_TRIGGER_REG.
5. Wait for the encryption process completion. Poll register SPI_MEM_XTS_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 SPI_MEM_XTS_RELEASE_REG to grant SPI1 the access to the encrypted ciphertext. After this, the value of register SPI_MEM_XTS_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 SPI_MEM_XTS_DESTROY_REG to destroy the ciphertext. After this, the value of register SPI_MEM_XTS_STATE_REG will become 0.

Repeat the above steps according to the amount of plaintext instructions/data that need to be encrypted.

26.6 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-H2 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-H2 allows users to enable anti-DPA function separately during the above-mentioned two steps to enhance security. See details in the following sections.

26.6.1 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:
  - $select\_reg$ = SPI_MEM_CRYPT_DPA_SELECT_REGISTER
  - $reg\_d\_dpa\_en$ = SPI_MEM_CRYPT_CALC_D_DPA_EN
  - $efuse\_dpa\_en$ = EFUSE_CRYPT_DPA_ENABLE
  - $reg\_anti\_dpa\_level$ = SPI_MEM_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.

26.6.2 Pseudo-round Anti-DPA Function

The pseudo-round function of ESP32-H2'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 26.6-1 for details.

Table 26.6-1. Configuration of XTS_AES Pseudo-round Anti-DPA

EFUSE_XTS_DPA_PSEUDO_LEVEL	pseudo_mode	pseudo_base	pseudo_inc	pseudo_rnd_cnt
0	SPI_MEM_MODE_PSEUDO	SPI_MEM_PSEUDO_BASE	SPI_MEM_PSEUDO_INC	SPI_MEM_PSEUDO_RNG_CNT
1	1	4	2	7
2	2	4	2	7
3	3	4	2	7

26.7 Register Summary

The addresses in this section are relative to the External Memory Encryption and Decryption base address provided in Table 4.3-2 in Chapter *4 System and Memory*.

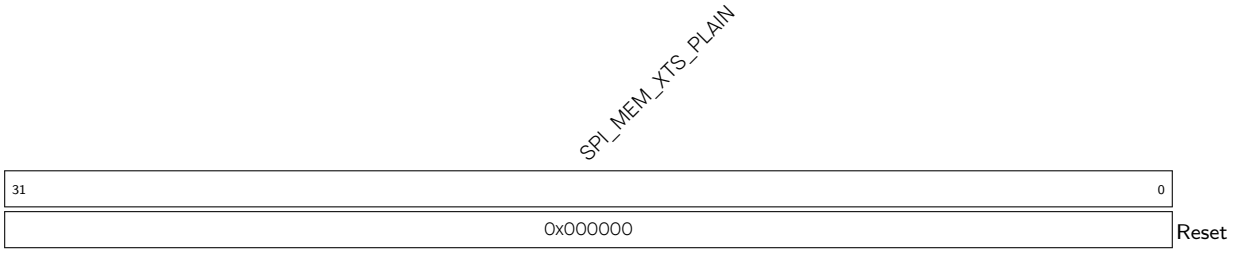
The abbreviations given in Column **Access** are explained in Section *Access Types for Registers*.

Name	Description	Address	Access
Plaintext Register Heap			
SPI_MEM_XTS_PLAIN_BASE_REG	Base address of plaintext memory	0x0300	R/W
Configuration Registers			
SPI_MEM_XTS_LINESIZE_REG	Configures the size of target memory space	0x0340	R/W
SPI_MEM_XTS_DESTINATION_REG	Configures the type of the external memory	0x0344	R/W
SPI_MEM_XTS_PHYSICAL_ADDRESS_REG	Physical address	0x0348	R/W
SPI_MEM_DPA_CTRL_REG	Configures the Anti-DPA function	0x0388	R/W
SPI_MEM_XTS_PSEUDO_ROUND_CONF_REG	XTS-AES pseudo-round function configuration register	0x038C	R/W
Control/Status Registers			
SPI_MEM_XTS_TRIGGER_REG	Activates AES algorithm	0x034C	WT
SPI_MEM_XTS_RELEASE_REG	Release control	0x0350	WT
SPI_MEM_XTS_DESTROY_REG	Destroy control	0x0354	WT
SPI_MEM_XTS_STATE_REG	Status register	0x0358	RO
Version Register			
SPI_MEM_XTS_DATE_REG	Version control register	0x035C	R/W

26.8 Registers

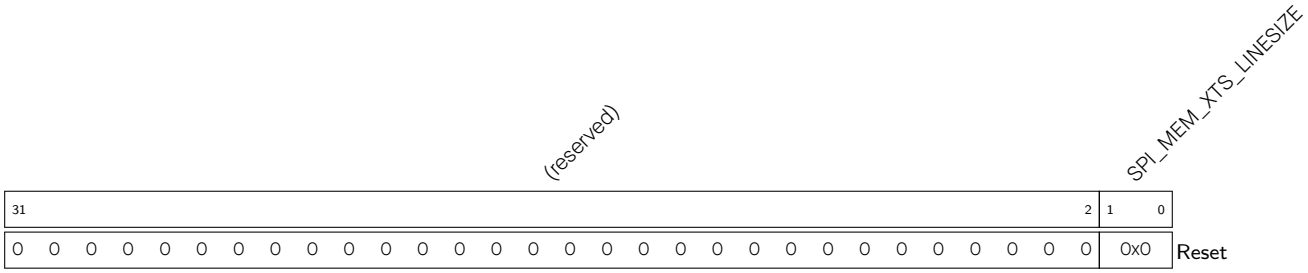
The addresses in this section are relative to the External Memory Encryption and Decryption base address provided in Table 4.3-2 in Chapter 4 *System and Memory*.

Register 26.1. SPI_MEM_XTS_PLAIN_BASE_REG (0x0300)



SPI_MEM_XTS_PLAIN This field is reserved for header generation and is not used by software. Write plaintext to the memory region starting at this base address as described in Section 26.4.4. (R/W)

Register 26.2. SPI_MEM_XTS_LINESIZE_REG (0x0340)



SPI_MEM_XTS_LINESIZE Configures the data size of one encryption operation.

0: 16 bytes

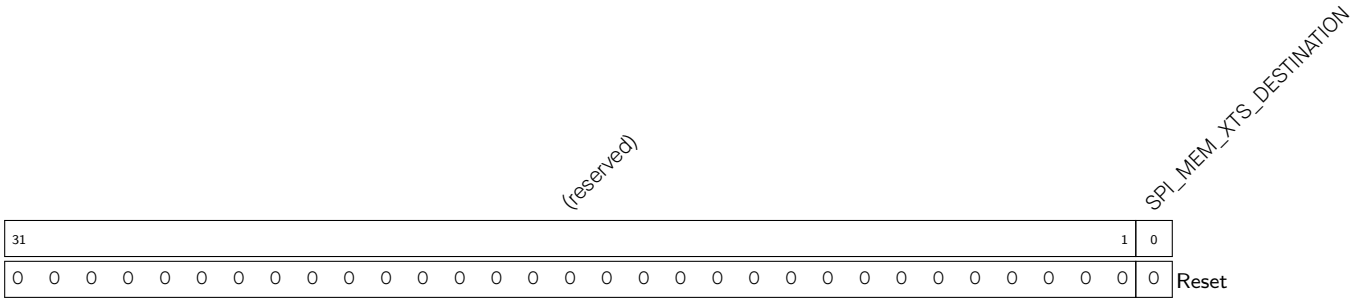
1: 32 bytes

2: 64 bytes

3: Invalid

(R/W)

Register 26.3. SPI_MEM_XTS_DESTINATION_REG (0x0344)



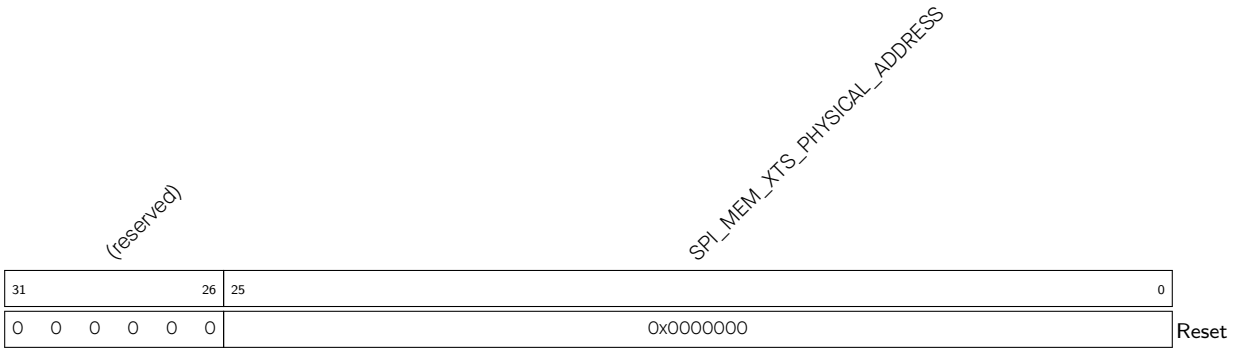
SPI_MEM_XTS_DESTINATION Configures the type of external memory. Currently, it must be set to 0, as the Manual Encryption block only supports flash encryption. Set this bit to 1 may cause an error.

0: flash

1: external RAM

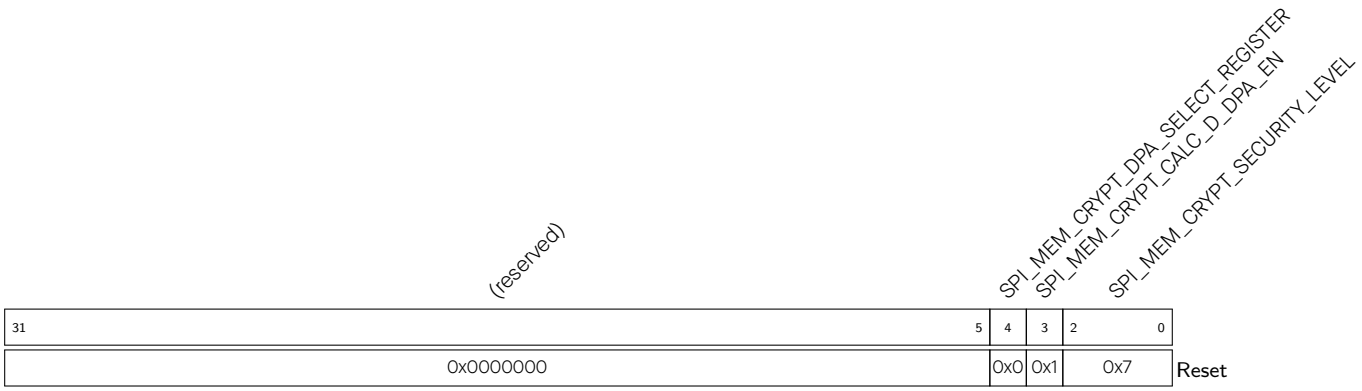
(R/W)

Register 26.4. SPI_MEM_XTS_PHYSICAL_ADDRESS_REG (0x0348)



SPI_MEM_XTS_PHYSICAL_ADDRESS Configures physical address. Note that its value should be within the range between 0x0000_0000 and 0x00FF_FFFF, and should be aligned with the byte number configured via the SPI_MEM_XTS_LINESIZE parameter. (R/W)

Register 26.5. SPI_MEM_DPA_CTRL_REG (0x0388)



SPI_MEM_CRYPT_DPA_SELECT_REGISTER Configures whether the Anti-DPA function is controlled by eFuse or register.

0: The Anti-DPA function is configured by eFuse.

1: The Anti-DPA function is configured by register.

(R/W)

SPI_MEM_CRYPT_CALC_D_DPA_EN Configures whether to enable Anti-DPA in the XTS_AES algorithm.

0: Enable Anti-DPA only when calculating T

1: Enable Anti-DPA both when calculating T and D

Note that this field is only effective when SPI_MEM_CRYPT_SECURITY_LEVEL is not 0.

(R/W)

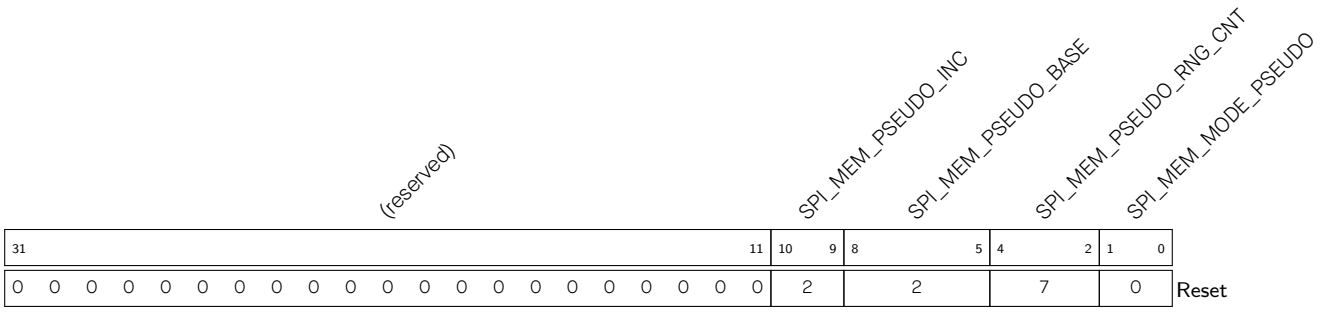
SPI_MEM_CRYPT_SECURITY_LEVEL Configures the security level of external memory encryption and decryption.

0: Disable the Anti-DPA function

1-7: The bigger the number is, the more secure the encryption and decryption are

(R/W)

Register 26.6. SPI_MEM_XTS_PSEUDO_ROUND_CONF_REG (0x038C)



SPI_MEM_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

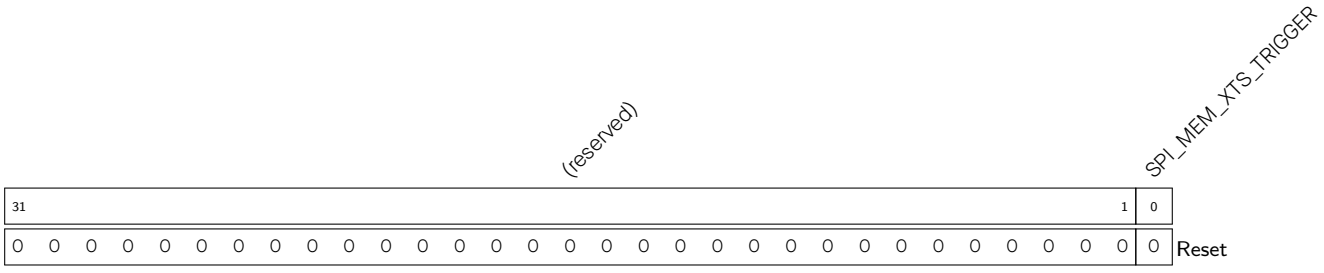
(R/W)

SPI_MEM_PSEUDO_RNG_CNT Configures the update frequency of the pseudo-key in the pseudo-round Anti-DPA function. (R/W)

SPI_MEM_PSEUDO_BASE Configures the basic round number of pseudo-round Anti-DPA function. (R/W)

SPI_MEM_PSEUDO_INC Configures the incremental round number of pseudo-round Anti-DPA function. (R/W)

Register 26.7. SPI_MEM_XTS_TRIGGER_REG (0x034C)



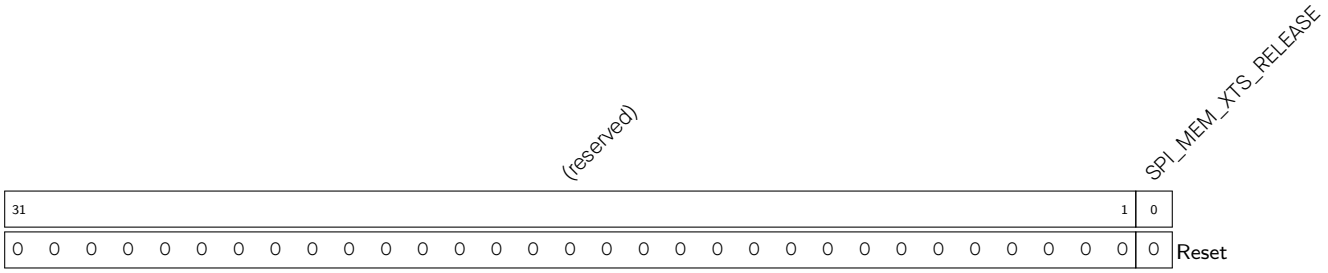
SPI_MEM_XTS_TRIGGER Configures whether or not to enable manual encryption.

0: Disable manual encryption

1: Enable manual encryption

(WT)

Register 26.8. SPI_MEM_XTS_RELEASE_REG (0x0350)



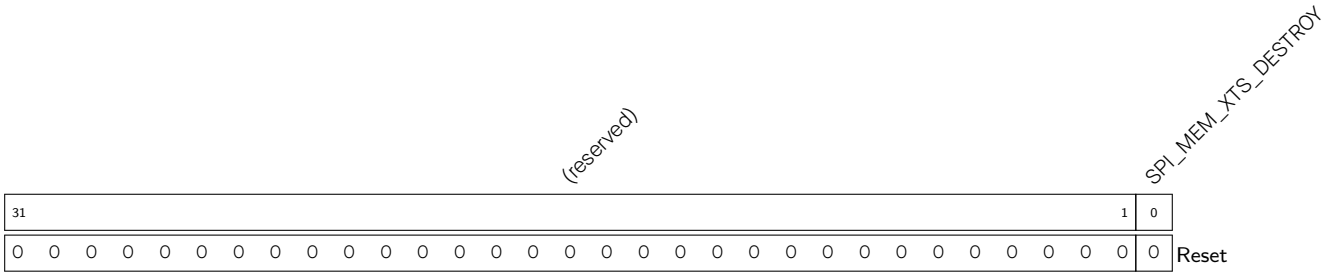
SPI_MEM_XTS_RELEASE Configures whether to grant SPI1 access to the encrypted result.

0: No effect

1: Grant SPI1 access

(WT)

Register 26.9. SPI_MEM_XTS_DESTROY_REG (0x0354)



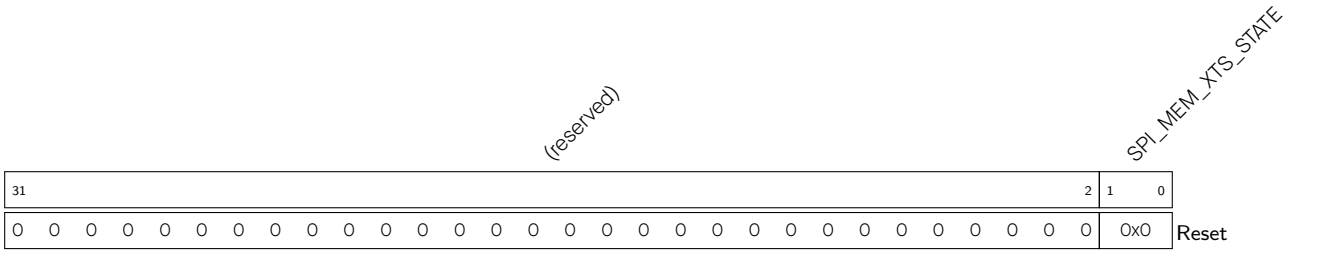
SPI_MEM_XTS_DESTROY Configures whether to destroy the encrypted result.

0: No effect

1: Destroy encrypted result

(WT)

Register 26.10. SPI_MEM_XTS_STATE_REG (0x0358)



SPI_MEM_XTS_STATE Represents the status of the Manual Encryption block. 0 (XTS_AES_IDLE): Idle

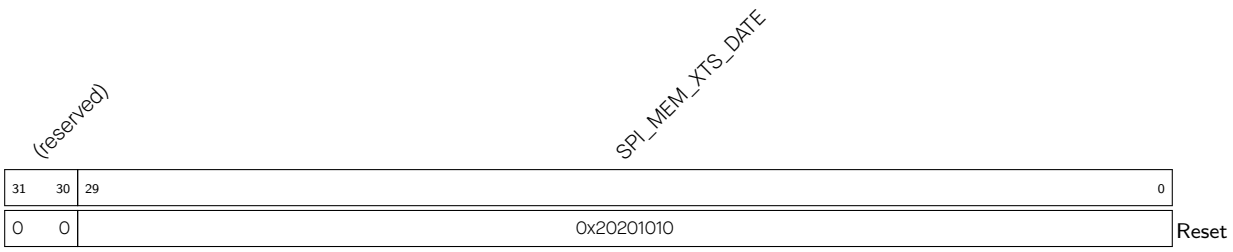
1 (XTS_AES_BUSY): Busy with encryption

2 (XTS_AES_DONE): Encryption completed, but the encrypted result is not accessible to SPI

3 (XTS_AES_RELEASE): Encrypted result is accessible to SPI

(RO)

Register 26.11. SPI_MEM_XTS_DATE_REG (0x035C)



SPI_MEM_XTS_DATE Version control register. (R/W)

Chapter 27

Random Number Generator (RNG)

27.1 Introduction

The ESP32-H2 contains a true random number generator, which generates 32-bit random numbers for cryptographical operations and other applications.

27.2 Features

The random number generator in ESP32-H2 generates true random numbers, derived from a physical process rather than an algorithm. Each number within the specified range has an equal probability of occurrence.

27.3 Functional Description

Every 32-bit value that the system reads from the LPPERI_RNG_DATA_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 and the **asynchronous clock mismatch**.

- **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 and it increases the RNG entropy by introducing circuit metastability.

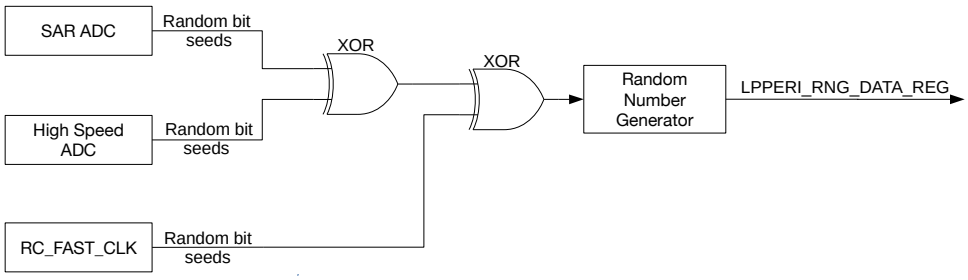


Figure 27.3-1. Noise Source

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 32 MHz.

When the SAR ADC generates noise, the random number generator obtains 2 bits of entropy per SAR ADC sampling cycle. The SAR ADC sampling cycle is typically 66 kHz, determined by the SAR ADC configuration.

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.

27.4 Programming Procedure

When using the random number generator, make sure at least either the SAR ADC, high-speed ADC[1] is enabled and it is recommended to enable RC_FAST_CLK[2] to increase the entropy which entry random numbers. Otherwise, pseudo-random numbers will be returned.

- SAR ADC can be enabled by using the DIG ADC controller. For details, please refer to Chapter 40 *ADC Controller (ADC)*.
  It is recommended to use the following configurations for SAR ADC:
  - Set the sampling channel to collect the internal voltage of the chip
  - Set the controller to use timer-driven continuous sampling mode
- High-speed ADC is enabled automatically when the Wireless module is enabled.
- RC_FAST_CLK is enabled by default and it could not be closed.

Note:

1. Note that, when the Wireless module (Thread or BLE) is enabled, the value read from the high-speed ADC can be saturated in some extreme cases, which lowers the entropy. Thus, it is advisable to also enable the SAR ADC as the noise source for the random number generator for such cases.
2. Enabling RC_FAST_CLK increases the RNG entropy. However, to ensure maximum entropy, it's recommended to always enable an ADC source as well.

When using the random number generator, read the LPPERI_RNG_DATA_REG register multiple times until sufficient random numbers have been generated.

Please ensure that the rate at which the register is read does not exceed the entropy generation frequency described in section 27.3 above.

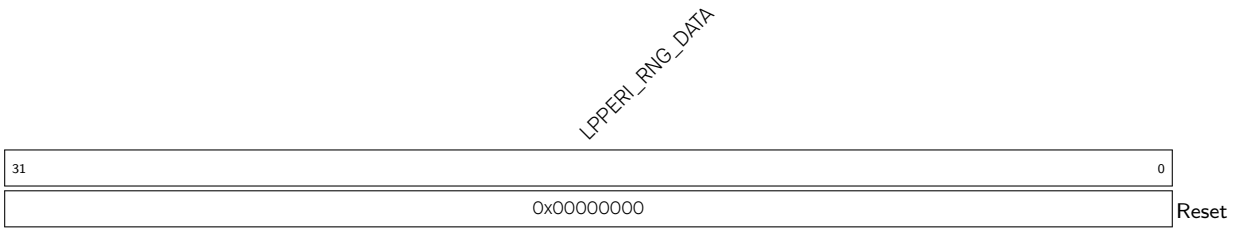
27.5 Register Summary

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

Name	Description	Address	Access
LPPERI_RNG_DATA_REG	Random number data	0x600B_280C	RO

27.6 Register

Register 27.1. LPPERI_RNG_DATA_REG (0x600B_280C)



LPPERI_RNG_DATA Random number source. (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 remote control, motor control, LED control, etc.

Chapter 28

UART Controller (UART)

28.1 Overview

In embedded system applications, data is required to be transferred in a simple way with minimal system resources. This can be achieved by a Universal Asynchronous Receiver/Transmitter (UART), which flexibly exchanges data with other peripheral devices in full-duplex mode. ESP32-H2 has two UART controllers. These UARTs are compatible with various UART devices, and support Infrared Data Association (IrDA) and RS485 communication.

Each of the two UART controllers has a group of registers that function identically. In this chapter, the two UART controllers are referred to as UART*n*, in which *n* denotes 0 or 1.

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.

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-H2 support various lengths of data bits and stop bits. These controllers also support software and hardware flow control as well as GDMA for high-speed data transfer. This allows developers to use multiple UART ports at minimal software cost.

28.2 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
- High-speed data communication using GDMA

- Receive timeout
- UART as wake-up source
- Software and hardware flow control
- Three prescalable clock sources: PLL_F48M_CLK, XTAL_CLK, and RC_FAST_CLK

28.3 UART Structure

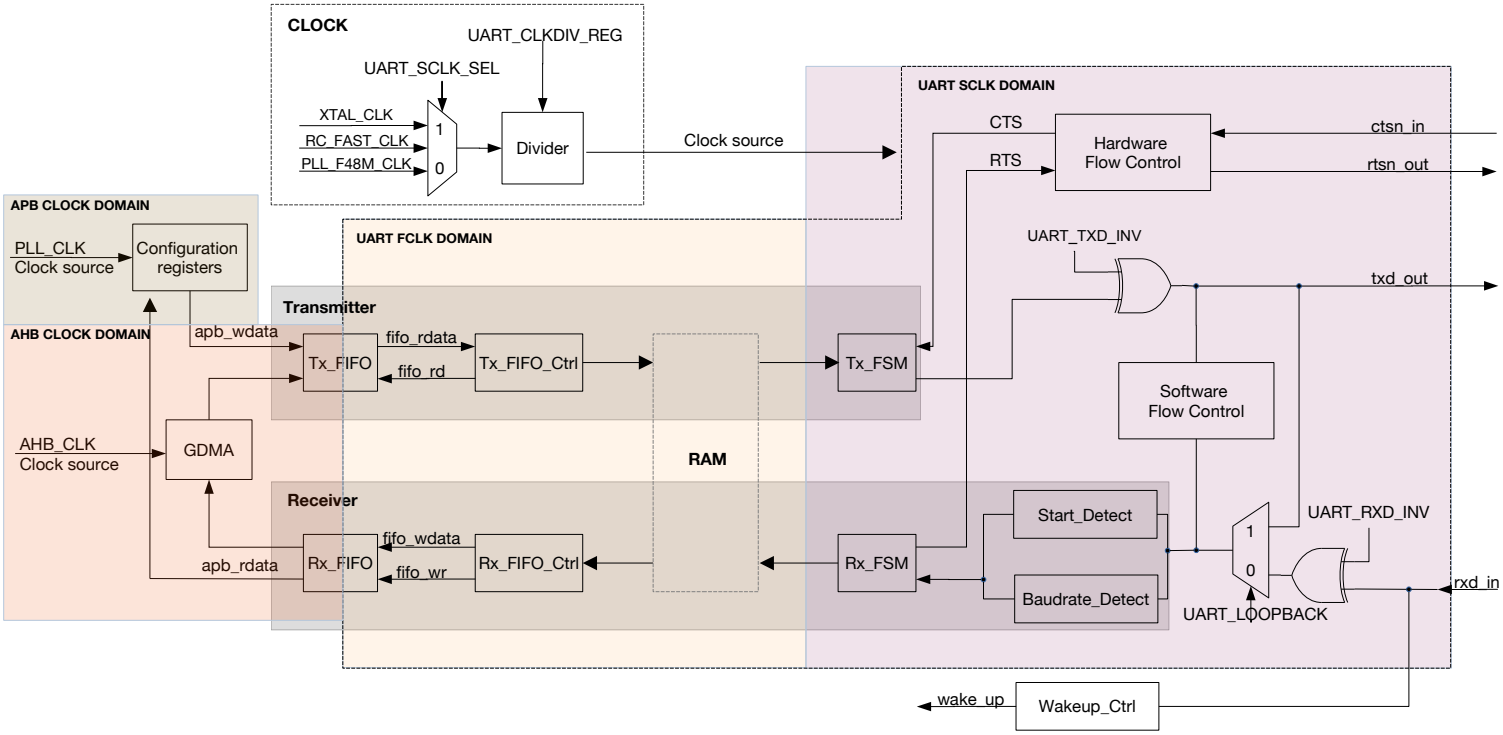


Figure 28.3-1. UART Structure

Figure 28.3-1 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: 48 MHz PLL_F48M_CLK, RC_FAST_CLK, and external crystal clock XTAL_CLK (for details, please refer to Chapter 7 *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 according to functions: a transmitter and a receiver.

The transmitter contains a TX FIFO (i.e., Tx_FIFO in Figure 28.3-1), which buffers data to be sent. Software can write data to Tx_FIFO via the APB bus, or move data to Tx_FIFO using GDMA. Tx_FIFO_Ctrl controls writing and reading Tx_FIFO. When Tx_FIFO is not empty, Tx_FSM reads data bits in the data frame via Tx_FIFO_Ctrl, and converts them into a bitstream. The levels of output bitstream signal txd_out can be inverted by configuring the UART_TXD_INV field.

The receiver contains an RX FIFO (i.e., Rx_FIFO in Figure 28.3-1), which buffers data to be processed. The input bitstream signal rxd_in is transferred to the UART controller, and its level can be inverted by configuring UART_RXD_INV field. Baudrate_Detect measures the baud rate of input bitstream signal rxd_in by detecting its minimum pulse width. Start_Detect detects the start bit in a data frame. If the start bit is detected, Rx_FSM stores data bits in the data frame into Rx_FIFO by Rx_FIFO_Ctrl. Software can read data from Rx_FIFO via the APB bus, or receive data using GDMA.

HW_Flow_Ctrl controls rxd_in and txd_out data flows by standard UART RTS and CTS flow control signals (rtsn_out and ctsn_in). SW_Flow_Ctrl controls data flows by adding special characters to outgoing data and detecting special characters in incoming data. When a UART controller is in Light-sleep mode (see Chapter 11 *Low-Power Management* for more details), a wake_up signal can be generated in four ways and sent to RTC, which then wakes up the ESP32-H2 chip. For more information about wakeup, please refer to Section 28.4.8.

28.4 Functional Description

28.4.1 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. The UART Core's clock frequency is lower than the PLL_CLK's frequency, and can be divided by the largest divisor when higher than the frequency needed to generate the baud rate. 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 Core's clock domain, please follow the procedures in Section28.5.

To reset the whole UART, please:

- Enable the UART Core*n*'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.

28.4.2 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.

UART0 Tx_FIFO and UART1 Tx_FIFO are reset by setting UART_TXFIFO_RST. UART0 Rx_FIFO and UART1 Rx_FIFO are reset by setting UART_RXFIFO_RST.

Data to be sent is written to TX FIFO via the APB bus or using GDMA, read automatically, and converted from a frame into a bitstream by hardware Tx_FSM. Data received is converted from a bitstream into a frame by hardware Rx_FSM, written into RX FIFO, and then stored into RAM via the APB bus or using GDMA. The two UART controllers share one GDMA channel.

The empty signal threshold for Tx_FIFO is configured by setting UART_TXFIFO_EMPTY_THRHD. When data stored in Tx_FIFO is less than UART_TXFIFO_EMPTY_THRHD, a UART_TXFIFO_EMPTY_INT interrupt is generated. The full signal threshold for Rx_FIFO is configured by setting UART_RXFIFO_FULL_THRHD. When data stored in Rx_FIFO is greater than or equal to UART_RXFIFO_FULL_THRHD, a UART_RXFIFO_FULL_INT interrupt is generated. In addition, when Rx_FIFO receives more data than its capacity, a UART_RXFIFO_OVF_INT interrupt is generated.

UART*n* can access FIFO via register UART_FIFO_REG. You can put data into TX FIFO by writing UART_RXFIFO_RD_BYTE, and get data in RX FIFO by reading UART_RXFIFO_RD_BYTE.

28.4.3 Baud Rate Generation and Detection

28.4.3.1 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 28.4-1, 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 28.4-1 below, to make the output timing more uniform:

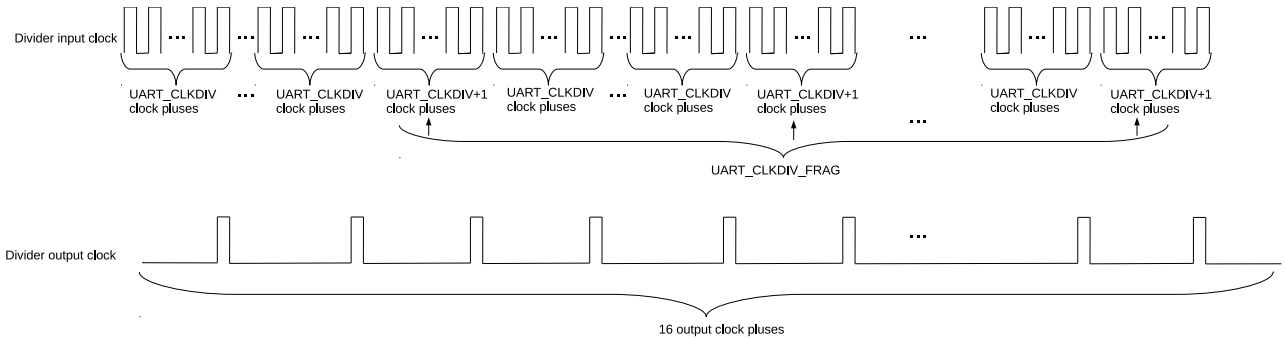


Figure 28.4-1. UART Controllers Division

To support IrDA (see Section 28.4.7 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.

28.4.3.2 Baud Rate Detection

Automatic baud rate detection (Autobaud) on UARTs is enabled by setting UART_AUTOBAUD_EN. The Baudrate_Detect module shown in Figure 28.4-2 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.

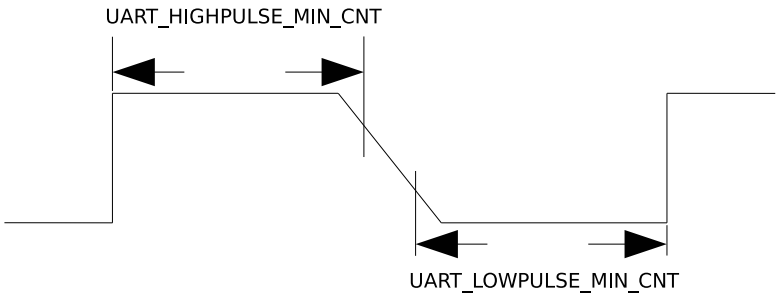


Figure 28.4-2. The Timing Diagram of Weak UART Signals Along Negative Edges

The baud rate can be determined in the following three ways:

1. Normally, to avoid sampling erroneous data along rising or negative edges in 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 28.4-2, 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:

$$B_{\text{uart}} = \frac{f_{\text{clk}}}{(\text{UART\_NEGEDGE\_MIN\_CNT} + 1)/2}$$

28.4.4 UART Data Frame

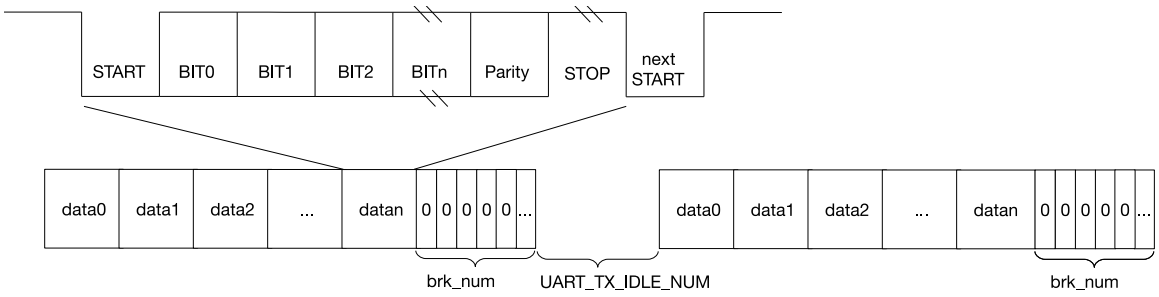


Figure 28.4-3. Structure of UART Data Frame

Figure 28.4-3 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 bits long, configured by UART_STOP_BIT_NUM (in RS485 mode turnaround delay may be added. See details in Section 28.4.6.2). The start bit is logical low, whereas stop bits are logical high.

The actual data length can be anywhere between 5 ~ 8 bit, configured by UART_BIT_NUM. When UART_PARITY_EN is set, a parity bit is added after data bits. UART_PARITY is used to choose even parity or odd parity. When the receiver detects a parity bit error in the data received, a UART_PARITY_ERR_INT interrupt is generated, and the data received will still be stored into RX FIFO. When the receiver detects a data frame error, a UART_FRM_ERR_INT interrupt is generated, and the data received by default is stored into RX FIFO.

If all data in Tx_FIFO has been sent, a UART_TX_DONE_INT interrupt is generated. After this, if the UART_TXD_BRK bit is set, then the transmitter will enter the Break condition and send several NULL characters in which the TX data line is logical low. The number of NULL characters is configured by UART_TX_BRK_NUM. Once the transmitter has sent all NULL characters, a UART_TX_BRK_DONE_INT interrupt is generated. The minimum interval between data frames can be configured using UART_TX_IDLE_NUM. If the transmitter stays idle for UART_TX_IDLE_NUM or more time, a UART_TX_BRK_IDLE_DONE_INT interrupt is generated.

The receiver can also detect the Break conditions when the RX data line detects any low logical level for one NULL character transmission, and a UART_BRK_DET_INT interrupt will be triggered 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.

28.4.5 AT_CMD Character Structure

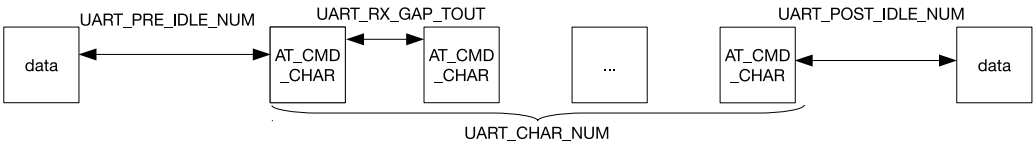


Figure 28.4-4. AT_CMD Character Structure

Figure 28.4-4 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 PLL_CLK frequency is suggested not to be lower than 8 MHz.

28.4.6 RS485

The two 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.

28.4.6.1 Driver Control

As shown in Figure 28.4-5, 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 28.4-5, DE is connected to dtrn_out of UART (please refer to Section 28.4.9.1 for more details).

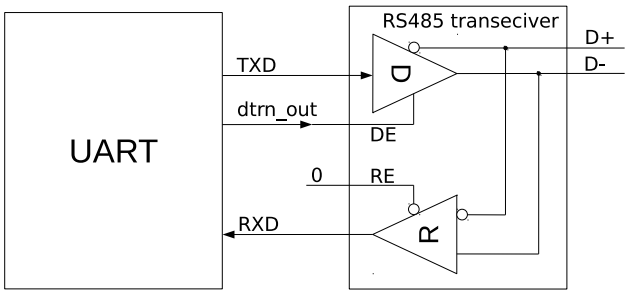


Figure 28.4-5. Driver Control Diagram in RS485 Mode

28.4.6.2 Turnaround Delay

By default, the two 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.

28.4.6.3 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 28.4-5, 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 polarity error, a UART_RS485_PARITY_ERR_INT is generated.

28.4.7 IrDA

IrDA protocol consists of three layers, namely the physical layer, the link access protocol, and the link management protocol. The two 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 28.4-6, 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 encoder output polarity is the opposite of the decoder input polarity. 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.

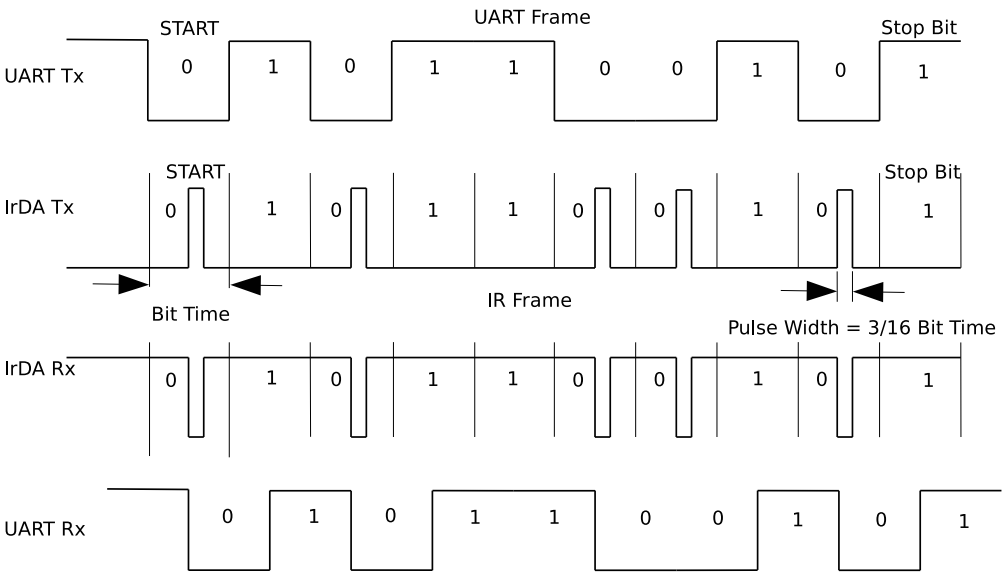


Figure 28.4-6. The Timing Diagram of Encoding and Decoding in SIR mode

The IrDA transceiver is half-duplex, meaning that it cannot send and receive data simultaneously. As shown in Figure 28.4-7, 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.

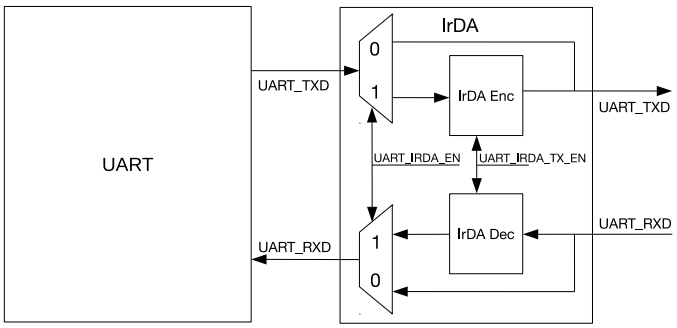


Figure 28.4-7. IrDA Encoding and Decoding Diagram

28.4.8 Wake-up

UART can be set as wake-up source. When a UART controller is in Light-sleep mode, a wake_up signal can be generated in four ways and be sent to the RTC module, which then wakes up ESP32-H2.

- 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 four characters can be formed into different character sequences by configuring UART_CHAR_NUM and UART_WK_CHAR_MASK, as shown in Table 28.4-1. Once the sequence is detected, the chip will be woken up. For the last configuration in Table 28.4-1, UART will detect for CHAR0 ~ CHAR4 in order.

Table 28.4-1. 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.

28.4.9 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.

28.4.9.1 Hardware Flow Control

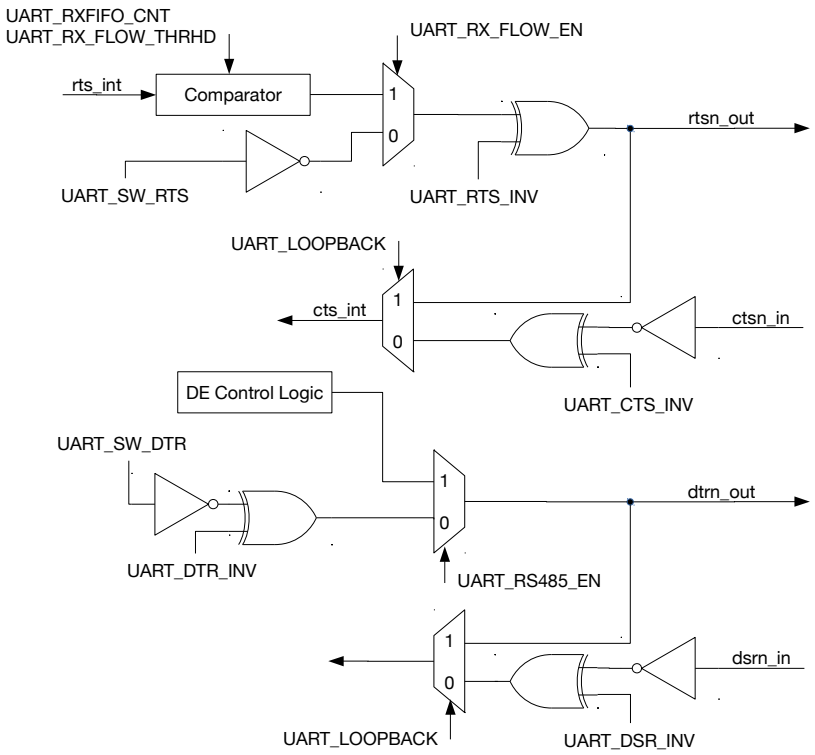


Figure 28.4-8. Hardware Flow Control Diagram

Figure 28.4-8 shows the hardware flow control of a UART controller. Hardware flow control uses output signal rtsn_out and input signal dsrn_in. Figure 28.4-9 illustrates how these signals are connected between UART on ESP32-H2 (hereinafter referred to as IU0) and the external UART (hereinafter referred to as EU0).

When rtsn_out of IU0 is low, EU0 is allowed to send data. When rtsn_out of IU0 is high, EU0 is notified to stop sending data until rtsn_out of IU0 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.

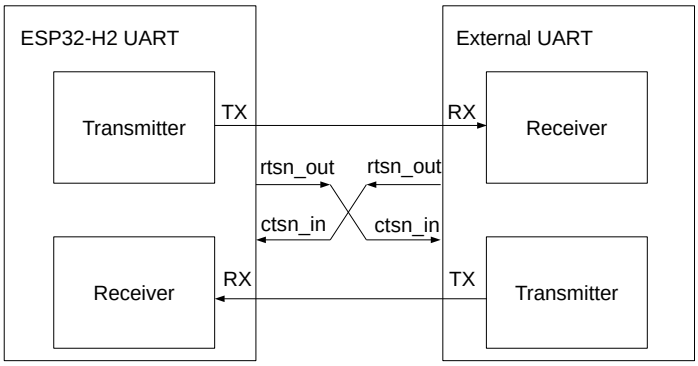


Figure 28.4-9. Connection between Hardware Flow Control Signals

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.

28.4.9.2 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 according to 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.

28.4.10 GDMA Mode

The two UART controllers on ESP32-H2 share one TX/RX GDMA (general direct memory access) channel via UHCI (Universal Host Controller Interface). In GDMA mode, UART controllers support the decoding and encoding of HCI data packets. The UHCI_UART*n*_CE field determines which UART controller occupies the GDMA TX/RX channel.

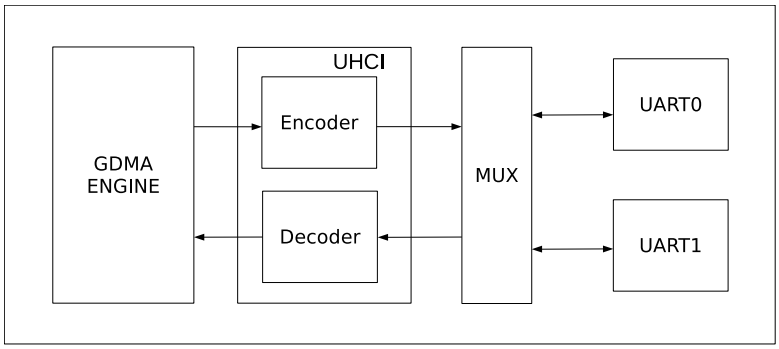


Figure 28.4-10. Data Transfer in GDMA Mode

Figure 28.4-10 shows how data is transferred using GDMA. Before GDMA receives data, software prepares an inlink. GDMA_INLINK_ADDR_CH*n* points to the first receive descriptor in the inlink. After GDMA_INLINK_START_CH*n* is set, UHCI sends data that UART has received to the decoder. The decoded data is then stored into the RAM pointed by the inlink under the control of GDMA.

Before GDMA sends data, software prepares an outlink and data to be sent. GDMA_OUTLINK_ADDR_CH*n* points to the first transmit descriptor in the outlink. After GDMA_OUTLINK_START_CH*n* is set, GDMA reads data from the RAM pointed by outlink. The data is then encoded by the encoder, and sent sequentially by the UART transmitter.

HCI data packets have separators at the beginning and the end, with data bits in the middle (separators + data bits + separators). The encoder inserts separators in front of and after data bits, and replaces data bits identical to separators with special characters. The decoder removes separators in front of and after data bits, and replaces special characters with separators. There can be more than one continuous separator at the beginning and the end of a data packet. The separator is configured by UHCI_SEPER_CHAR, 0xC0 by default. The special character is configured by UHCI_ESC_SEQ0_CHAR0 (0xDB by default) and UHCI_ESC_SEQ0_CHAR1 (0xDD by default). When all data has been sent, a GDMA_OUT_TOTAL_EOF_CH*n*_INT interrupt is generated. When all data has been received, a GDMA_IN_SUC_EOF_CH*n*_INT is generated.

28.4.11 UART Interrupts

- 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.

28.4.12 UHCI Interrupts

- UHCI_APP_CTRL1_INT: Triggered when software sets UHCI_APP_CTRL1_INT_RAW.
- UHCI_APP_CTRL0_INT: Triggered when software sets UHCI_APP_CTRL0_INT_RAW.
- UHCI_OUTLINK_EOF_ERR_INT: Triggered when an EOF error is detected in a transmit descriptor.
- UHCI_SEND_A_REG_Q_INT: Triggered when UHCI has sent a series of short packets using always_send.

- UHCI_SEND_S_REG_Q_INT: Triggered when UHCI has sent a series of short packets using single_send.
- UHCI_TX_HUNG_INT: Triggered when UHCI takes too long to read RAM using a GDMA transmit channel.
- UHCI_RX_HUNG_INT: Triggered when UHCI takes too long to receive data using a GDMA receive channel.
- UHCI_TX_START_INT: Triggered when GDMA detects a separator character.
- UHCI_RX_START_INT: Triggered when a separator character has been sent.

28.5 Programming Procedures

28.5.1 Register Type

All UART registers are in the APB_CLK domain.

UART configuration registers can be classified into two groups. One group of registers are read in 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 Core's clock 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 28.6. Those without the _SYNC suffix in Section 28.6 are configuration registers that require no clock domain crossing.

28.5.2 Detailed Steps

Figure 28.5-1 illustrates the process to program UART controllers, namely initialize UART, configure registers, enable the UART transmitter or receiver, and finish data transmission.

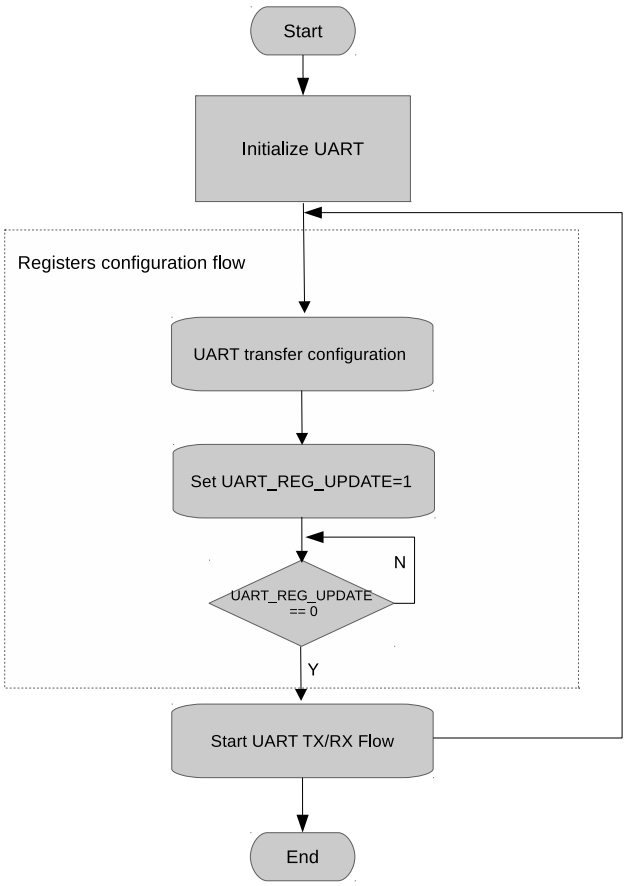


Figure 28.5-1. UART Programming Procedures

28.5.2.1 Initializing UART*n*

To initialize UART*n*:

- Write 1 to PCR_UART*n*_RST_EN.
- Clear PCR_UART*n*_RST_EN to 0.

28.5.2.2 Configuring UART*n* Communication

To configure UART*n* communication:

- Wait for UART_REG_UPDATE to become 0, which indicates the completion of the last synchronization.
- Select the clock source via 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.

28.5.2.3 Enabling UART*n*

To enable UART*n* transmitter:

- Configure TX FIFO's empty threshold via UART_TXFIFO_EMPTY_THRHD.
- Disable UART_TXFIFO_EMPTY_INT interrupt by clearing UART_TXFIFO_EMPTY_INT_ENA.
- Write data to be sent to UART_RXFIFO_RD_BYTE.
- Clear UART_TXFIFO_EMPTY_INT interrupt by setting UART_TXFIFO_EMPTY_INT_CLR.
- Enable UART_TXFIFO_EMPTY_INT interrupt by setting UART_TXFIFO_EMPTY_INT_ENA.
- Check UART_TXFIFO_EMPTY_INT_ST and wait for the completion of data transmission.

To enable UART*n* receiver:

- Configure RX FIFO's full threshold via UART_RXFIFO_FULL_THRHD.
- Enable UART_RXFIFO_FULL_INT interrupt by setting UART_RXFIFO_FULL_INT_ENA.
- Check UART_RXFIFO_FULL_INT_ST and wait until the RX FIFO is full.
- Read data from RX FIFO via UART_RXFIFO_RD_BYTE, and obtain the number of bytes received in RX FIFO via UART_RXFIFO_CNT.

28.6 Register Summary

28.6.1 UART Register Summary

The addresses in this section are relative to UART Controller base address provided in Table 4.3-2 in Chapter 4 *System and Memory*.

Name	Description	Address	Access
FIFO Configuration			
UART_FIFO_REG	FIFO data register	0x0000	RO
UART_TOUT_CONF_SYNC_REG	UART threshold and allocation configuration	0x0064	R/W
UART Interrupt Register			
UART_INT_RAW_REG	Raw interrupt status	0x0004	R/WTC/SS
UART_INT_ST_REG	Masked interrupt status	0x0008	RO
UART_INT_ENA_REG	Interrupt enable bits	0x000C	R/W
UART_INT_CLR_REG	Interrupt clear bits	0x0010	WT
Configuration Register			
UART_CLKDIV_SYNC_REG	Clock divider configuration	0x0014	R/W
UART_RX_FILT_REG	RX filter configuration	0x0018	R/W
UART_CONF0_SYNC_REG	Configuration register 0	0x0020	R/W
UART_CONF1_REG	Configuration register 1	0x0024	R/W
UART_HWFC_CONF_SYNC_REG	Hardware flow control configuration	0x002C	R/W
UART_SLEEP_CONF0_REG	UART sleep configuration register 0	0x0030	R/W
UART_SLEEP_CONF1_REG	UART sleep configuration register 1	0x0034	R/W
UART_SLEEP_CONF2_REG	UART sleep configuration register 2	0x0038	R/W
UART_SWFC_CONF0_SYNC_REG	Software flow control character configuration	0x003C	varies
UART_SWFC_CONF1_REG	Software flow control character configuration	0x0040	R/W
UART_TXBRK_CONF_SYNC_REG	TX break character configuration	0x0044	R/W
UART_IDLE_CONF_SYNC_REG	Frame end idle time configuration	0x0048	R/W
UART_RS485_CONF_SYNC_REG	RS485 mode configuration	0x004C	R/W
UART_CLK_CONF_REG	UART core clock configuration	0x0088	R/W
UART_REG_UPDATE_REG	UART register configuration update	0x0098	R/W/SC
UART_ID_REG	UART ID register	0x009C	R/W
Status Register			
UART_STATUS_REG	UART status register	0x001C	RO
UART_MEM_TX_STATUS_REG	TX FIFO write and read offset address	0x0068	RO
UART_MEM_RX_STATUS_REG	Rx FIFO write and read offset address	0x006C	RO
UART_FSM_STATUS_REG	UART transmit and receive status	0x0070	RO
UART_AFIFO_STATUS_REG	UART asynchronous FIFO status	0x0090	RO
AT Escape Sequence Selection Configuration			
UART_AT_CMD_PRECNT_SYNC_REG	Pre-sequence timing configuration	0x0050	R/W
UART_AT_CMD_POSTCNT_SYNC_REG	Post-sequence timing configuration	0x0054	R/W
UART_AT_CMD_GAPTOUT_SYNC_REG	Timeout configuration	0x0058	R/W
UART_AT_CMD_CHAR_SYNC_REG	AT escape sequence detection configuration	0x005C	R/W
Autobaud Register			

Name	Description	Address	Access
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

28.6.2 UHCI Register Summary

The addresses in this section are relative to UHCI base address provided in Table 4.3-2 in Chapter 4 *System and Memory*.

Name	Description	Address	Access
Configuration Register			
UHCI_CONF0_REG	UHCI configuration register	0x0000	R/W
UHCI_CONF1_REG	UHCI configuration register	0x0014	varies
UHCI_ESCAPE_CONF_REG	Escape character configuration	0x0020	R/W
UHCI_HUNG_CONF_REG	Timeout configuration	0x0024	R/W
UHCI_ACK_NUM_REG	UHCI ACK number configuration	0x0028	varies
UHCI_QUICK_SENT_REG	UHCI quick send configuration register	0x0030	varies
UHCI_REG_Q0_WORD0_REG	Q0 WORD0 quick send register	0x0034	R/W
UHCI_REG_Q0_WORD1_REG	Q0 WORD1 quick send register	0x0038	R/W
UHCI_REG_Q1_WORD0_REG	Q1 WORD0 quick send register	0x003C	R/W
UHCI_REG_Q1_WORD1_REG	Q1 WORD1 quick send register	0x0040	R/W
UHCI_REG_Q2_WORD0_REG	Q2 WORD0 quick send register	0x0044	R/W
UHCI_REG_Q2_WORD1_REG	Q2 WORD1 quick send register	0x0048	R/W
UHCI_REG_Q3_WORD0_REG	Q3 WORD0 quick send register	0x004C	R/W
UHCI_REG_Q3_WORD1_REG	Q3 WORD1 quick send register	0x0050	R/W
UHCI_REG_Q4_WORD0_REG	Q4 WORD0 quick send register	0x0054	R/W
UHCI_REG_Q4_WORD1_REG	Q4 WORD1 quick send register	0x0058	R/W
UHCI_REG_Q5_WORD0_REG	Q5 WORD0 quick send register	0x005C	R/W
UHCI_REG_Q5_WORD1_REG	Q5 WORD1 quick send register	0x0060	R/W
UHCI_REG_Q6_WORD0_REG	Q6 WORD0 quick send register	0x0064	R/W
UHCI_REG_Q6_WORD1_REG	Q6 WORD1 quick send register	0x0068	R/W
UHCI_ESC_CONF0_REG	Escape sequence configuration register 0	0x006C	R/W
UHCI_ESC_CONF1_REG	Escape sequence configuration register 1	0x0070	R/W
UHCI_ESC_CONF2_REG	Escape sequence configuration register 2	0x0074	R/W
UHCI_ESC_CONF3_REG	Escape sequence configuration register 3	0x0078	R/W
UHCI_PKT_THRES_REG	Configuration register for packet length	0x007C	R/W
UHCI Interrupt Register			
UHCI_INT_RAW_REG	Raw interrupt status	0x0004	varies
UHCI_INT_ST_REG	Masked interrupt status	0x0008	RO

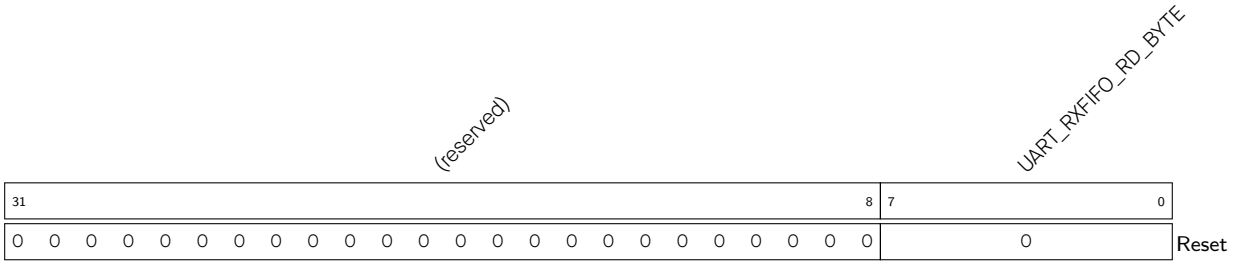
Name	Description	Address	Access
UHCI_INT_ENA_REG	Interrupt enable bits	0x000C	R/W
UHCI_INT_CLR_REG	Interrupt clear bits	0x0010	WT
UHCI Status Register			
UHCI_STATE0_REG	UHCI receive status	0x0018	RO
UHCI_STATE1_REG	UHCI transmit status	0x001C	RO
UHCI_RX_HEAD_REG	UHCI packet header register	0x002C	RO
Version Register			
UHCI_DATE_REG	UHCI version control register	0x0080	R/W

28.7 Registers

28.7.1 UART Registers

The addresses in this section are relative to UART Controller base address provided in Table 4.3-2 in Chapter 4 *System and Memory*.

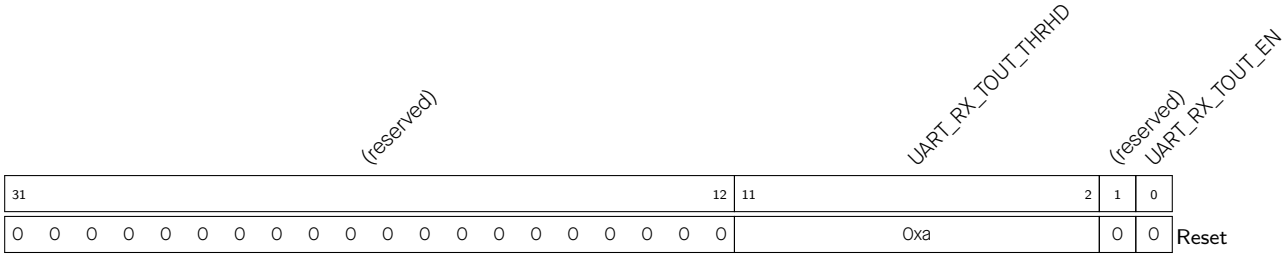
Register 28.1. UART_FIFO_REG (0x0000)



UART_RXFIFO_RD_BYTE Represents the data UART *n* read from FIFO.

Measurement unit: byte. (RO)

Register 28.2. UART_TOUT_CONF_SYNC_REG (0x0064)



UART_RX_TOUT_EN Configures whether or not to enable UART receiver's timeout function.

0: Disable

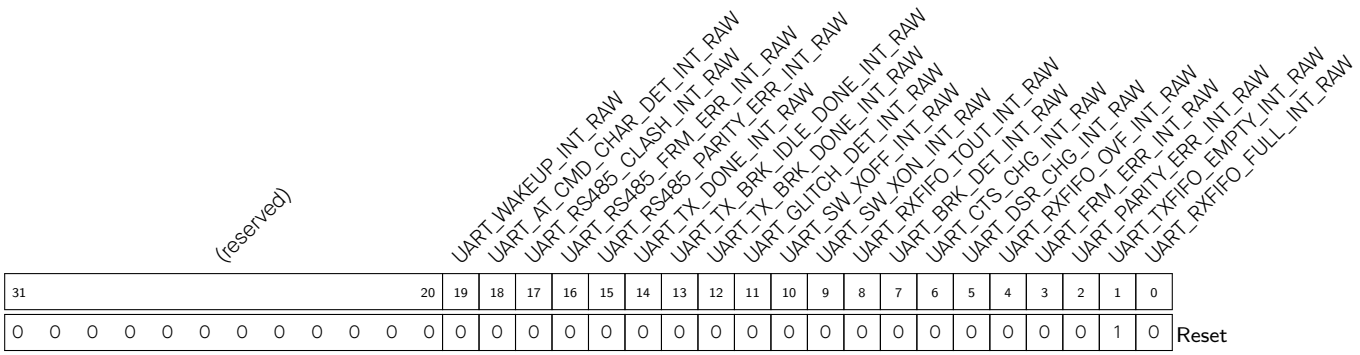
1: Enable

(R/W)

UART_RX_TOUT_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 28.3. 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 The raw interrupt status of 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 28.3. UART_INT_RAW_REG (0x0004)

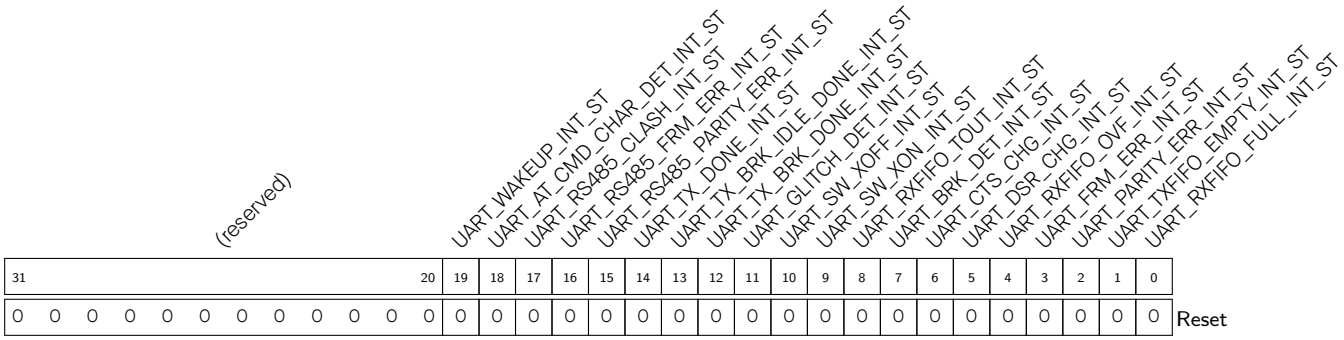
Continued from the previous page...

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 28.4. 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 28.5. 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)

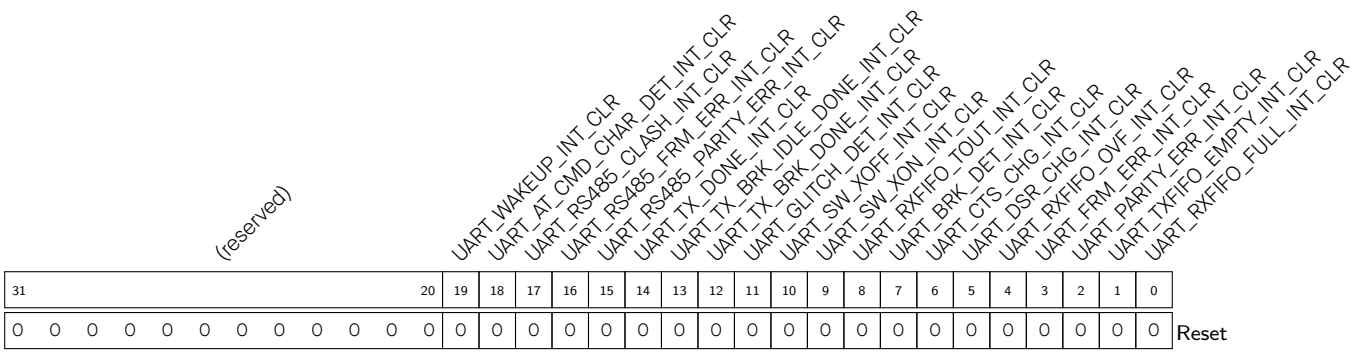
UART_RS485_FRM_ERR_INT_ENA Write 1 to enable UART_RS485_FRM_ERR_INT. (R/W)

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 28.6. 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)

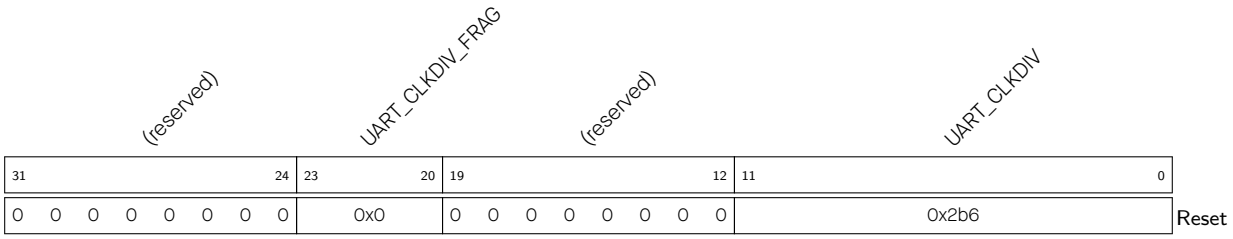
UART_RS485_FRM_ERR_INT_CLR Write 1 to clear UART_RS485_FRM_ERR_INT. (WT)

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)

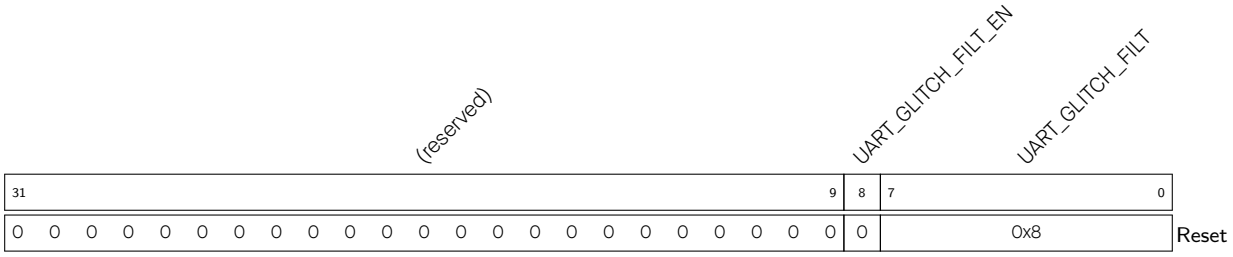
Register 28.7. UART_CLKDIV_SYNC_REG (0x0014)



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 28.8. 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)

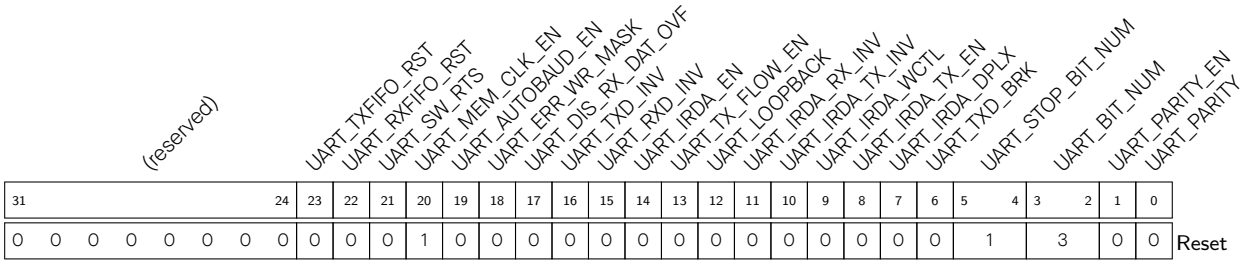
UART_GLITCH_FILT_EN Configures whether or not to enable RX signal filter.

0: Disable

1: Enable

(R/W)

Register 28.9. 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 28.9. 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 28.9. 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)

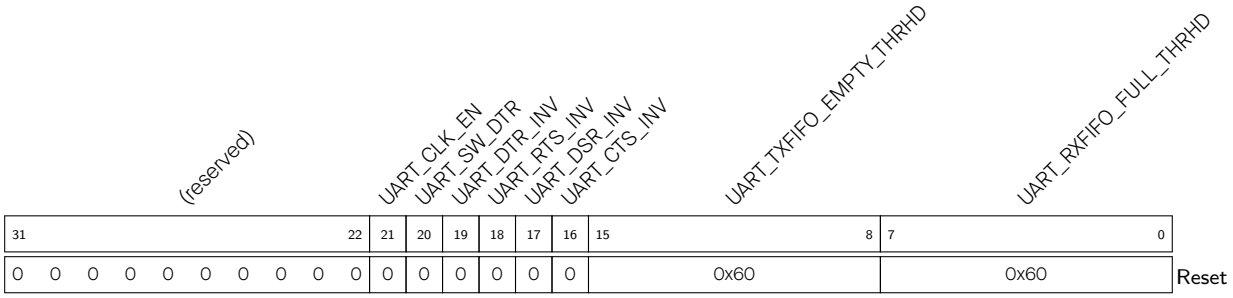
UART_TXFIFO_RST Configures whether or not to reset the UART TX FIFO.

0: Not reset

1: Reset

(R/W)

Register 28.10. 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)

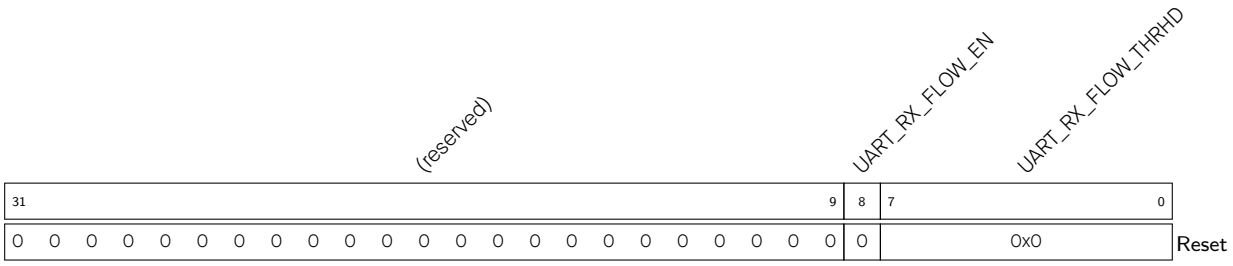
UART_CLK_EN Configures clock gating.

0: Support clock only when the application writes registers.

1: Always force the clock on for registers.

(R/W)

Register 28.11. 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)

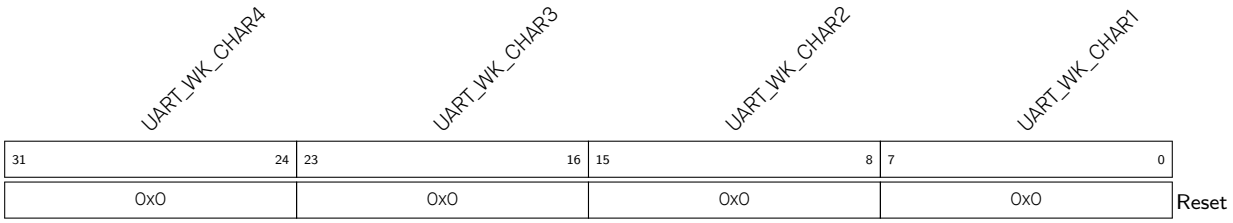
UART_RX_FLOW_EN Configures whether or not to enable the UART receiver.

0: Disable

1: Enable

(R/W)

Register 28.12. UART_SLEEP_CONF0_REG (0x0030)



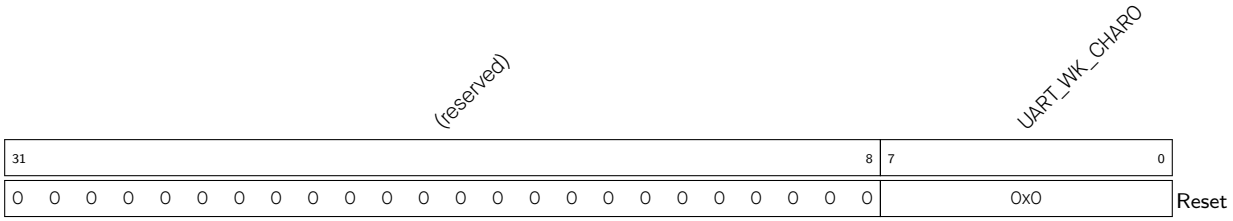
UART_WK_CHAR1 Configures wakeup character 1. (R/W)

UART_WK_CHAR2 Configures wakeup character 2. (R/W)

UART_WK_CHAR3 Configures wakeup character 3. (R/W)

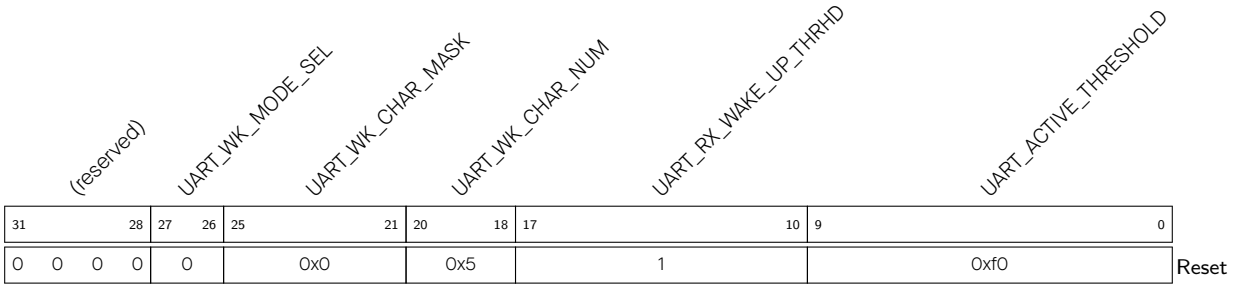
UART_WK_CHAR4 Configures wakeup character 4. (R/W)

Register 28.13. UART_SLEEP_CONF1_REG (0x0034)



UART_WK_CHAR0 Configures wakeup character 0. (R/W)

Register 28.14. 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.

0: Mode 0

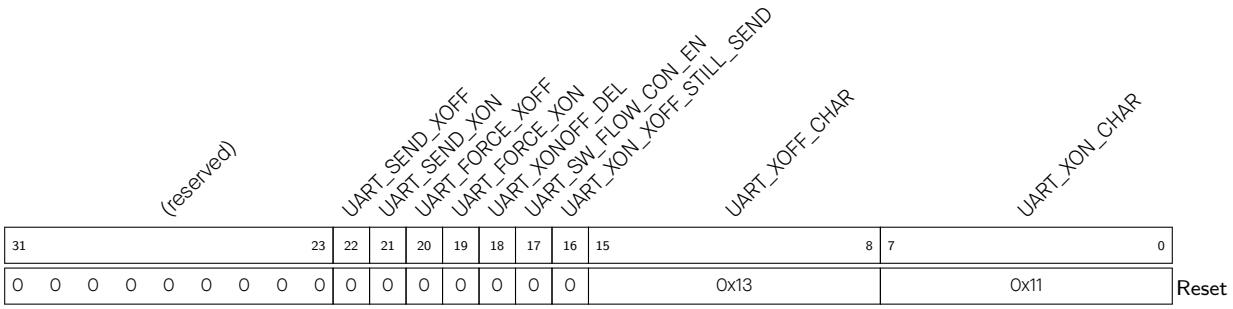
1: Mode 1

2: Mode 2

3: Mode 3

(R/W)

Register 28.15. 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 remove

1: Remove

(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)

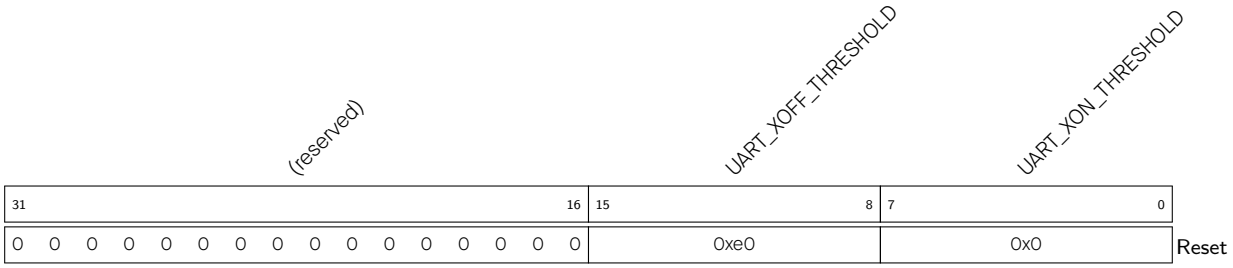
UART_SEND_XOFF Configures whether or not to send XOFF characters.

0: Not send

1: Send

(R/W/SS/SC)

Register 28.16. UART_SWFC_CONF1_REG (0x0040)



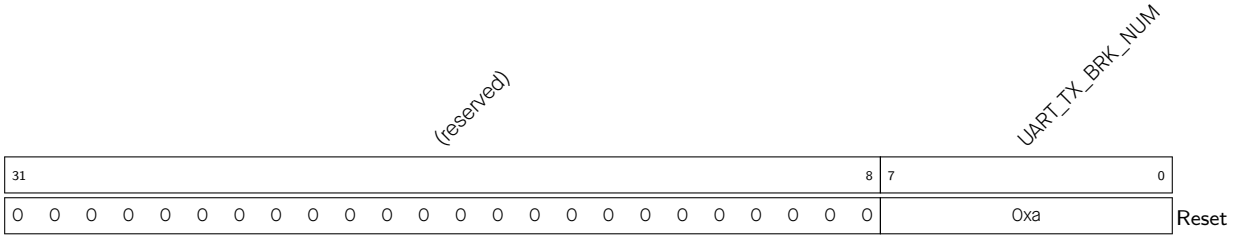
UART_XON_THRESHOLD Configures the threshold for data in RX FIFO to send XON characters in software flow control.

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)

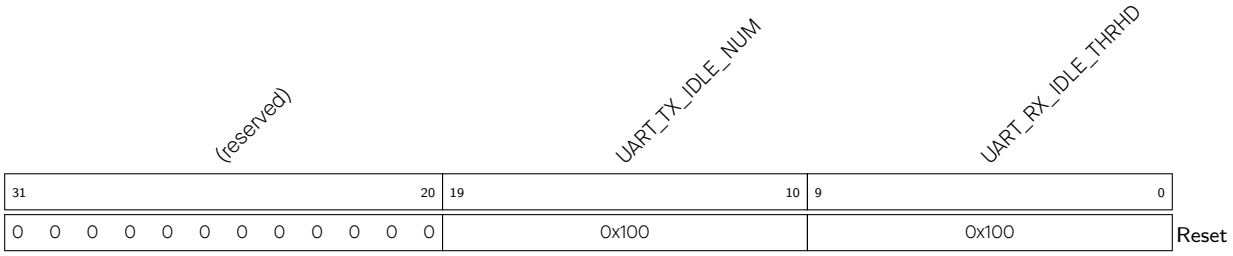
Register 28.17. UART_TXBRK_CONF_SYNC_REG (0x0044)



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 28.18. UART_IDLE_CONF_SYNC_REG (0x0048)



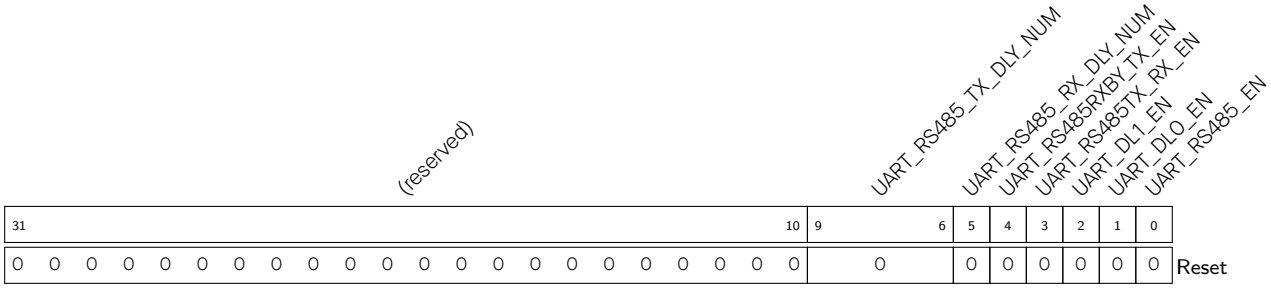
UART_RX_IDLE_THRHD Configures the threshold to generate a frame end signal when the receiver takes more time to receive one byte data.

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 28.19. 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 or not to enable the RS485 transmitter for data transmission when the RS485 receiver is busy.

0: Disable

1: Enable

(R/W)

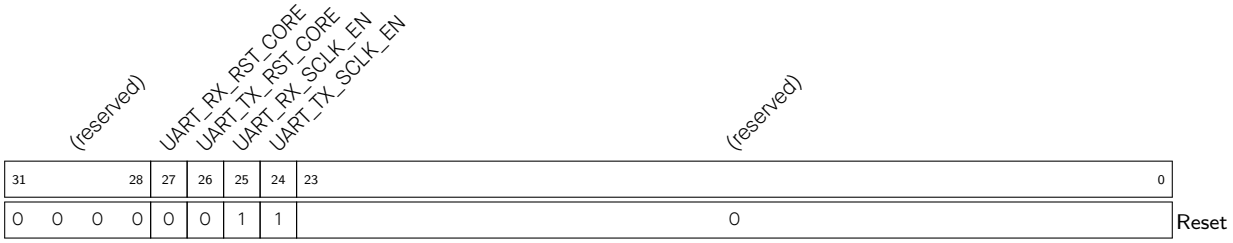
UART_RS485_RX_DLY_NUM Configures the delay of internal data signals in the receiver.

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 28.20. 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

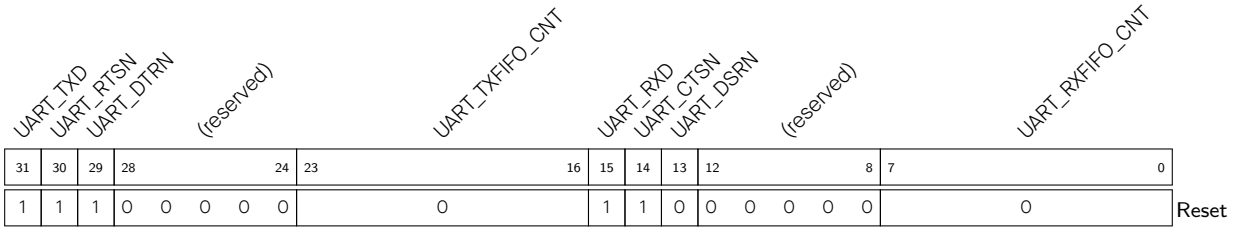
1: Enable

(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 28.21. UART_STATUS_REG (0x001C)



UART_RXFIFO_CNT Represents the number of valid data bytes in RX FIFO.

Measurement unit: byte. (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.

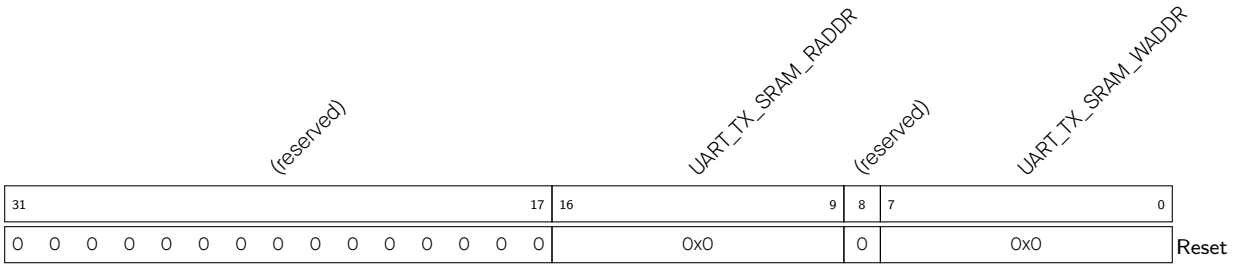
Measurement unit: byte. (RO)

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)

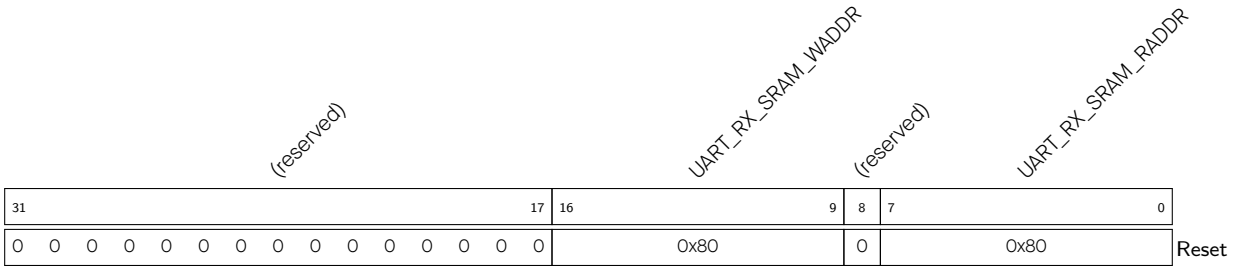
Register 28.22. UART_MEM_TX_STATUS_REG (0x0068)



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)

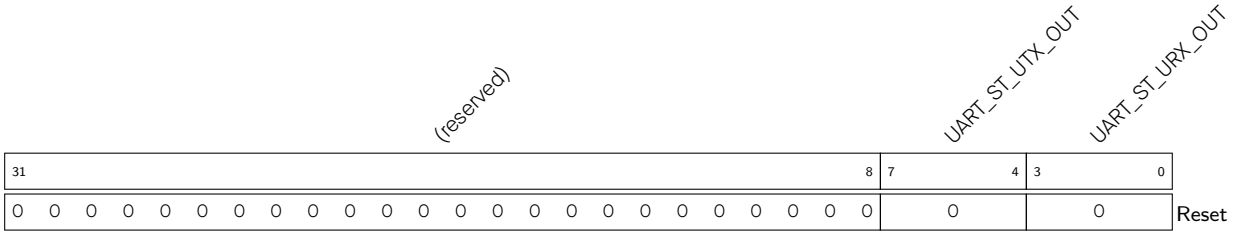
Register 28.23. UART_MEM_RX_STATUS_REG (0x006C)



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)

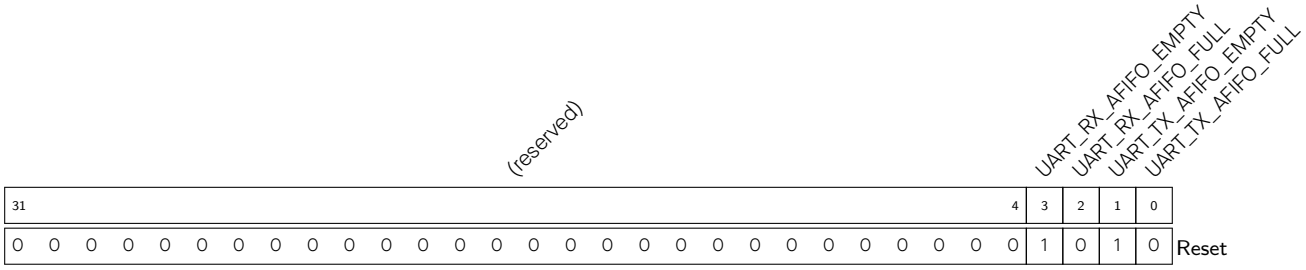
Register 28.24. UART_FSM_STATUS_REG (0x0070)



UART_ST_URX_OUT Represents the status of the receiver. (RO)

UART_ST_UTX_OUT Represents the status of the transmitter. (RO)

Register 28.25. 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)

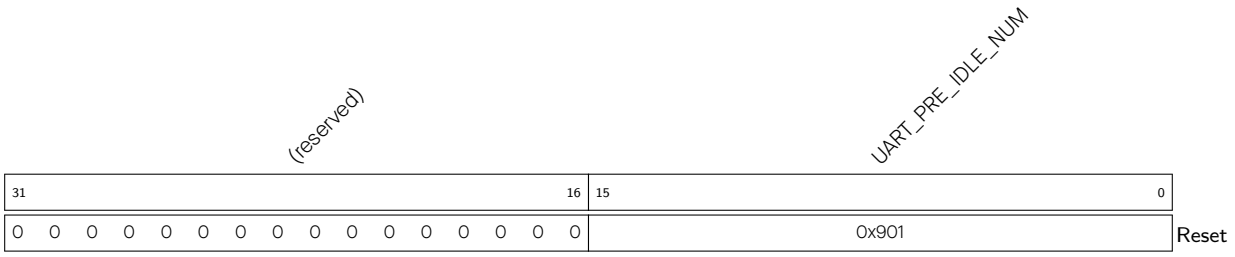
UART_RX_AFIFO_EMPTY Represents whether or not the APB RX asynchronous FIFO is empty.

0: Not empty

1: Empty

(RO)

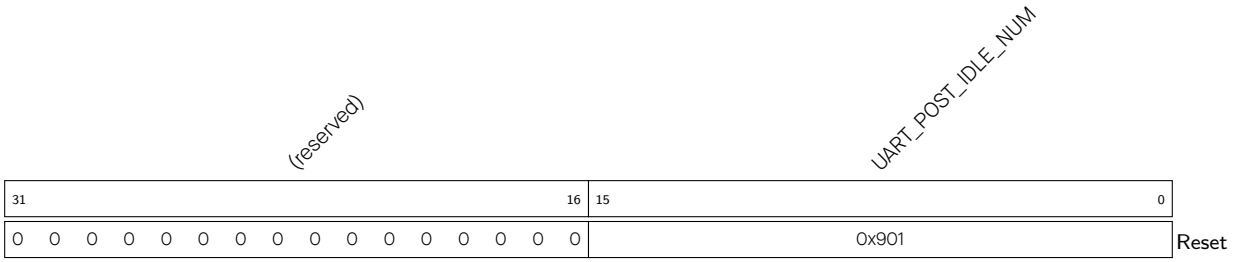
Register 28.26. UART_AT_CMD_PRECNT_SYNC_REG (0x0050)



UART_PRE_IDLE_NUM Configures the idle time before the receiver receives the first AT_CMD.

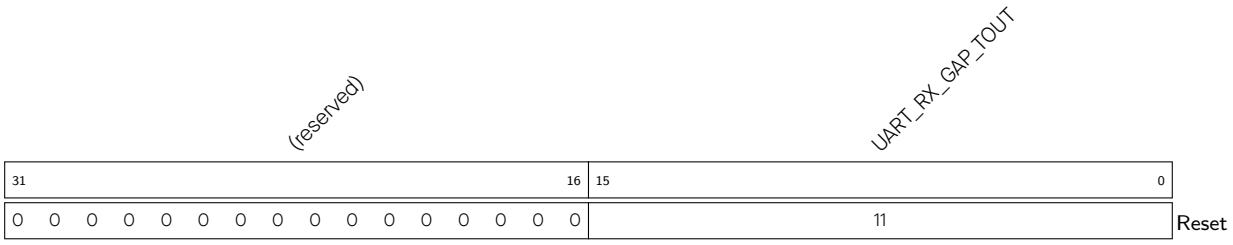
Measurement unit: bit time (the time to transmit 1 bit). (R/W)

Register 28.27. UART_AT_CMD_POSTCNT_SYNC_REG (0x0054)



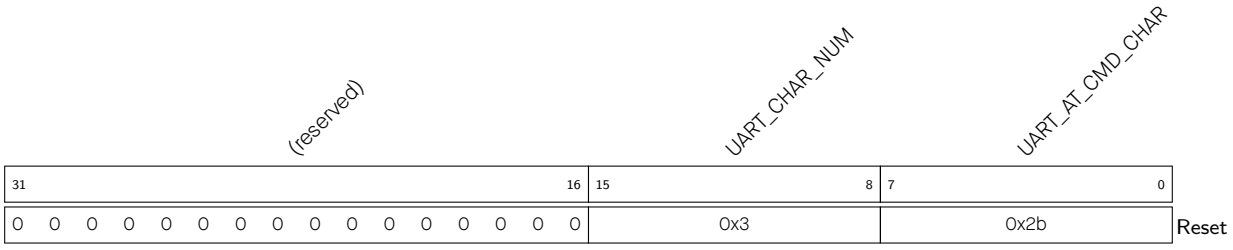
UART_POST_IDLE_NUM Configures the interval between the last AT_CMD and subsequent data. Measurement unit: bit time (the time to transmit 1 bit). (R/W)

Register 28.28. 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)

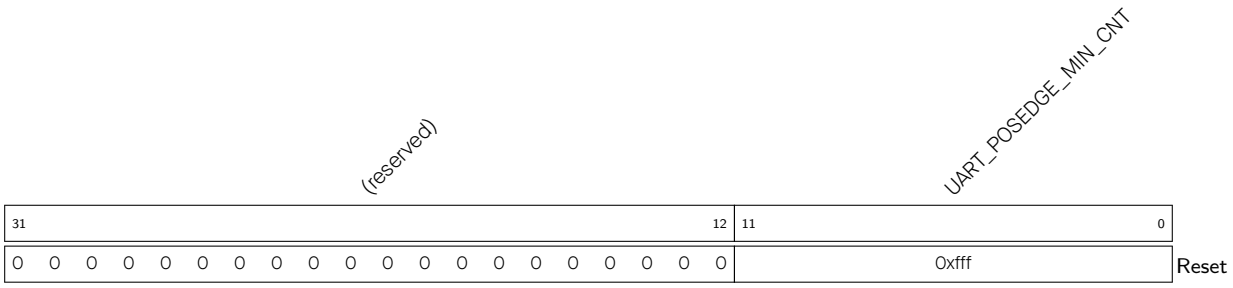
Register 28.29. UART_AT_CMD_CHAR_SYNC_REG (0x005C)



UART_AT_CMD_CHAR Configures the AT_CMD character. (R/W)

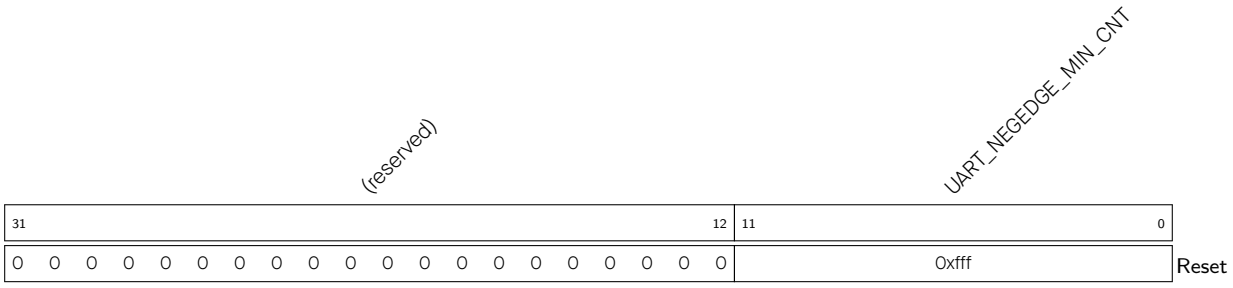
UART_CHAR_NUM Configures the number of continuous AT_CMD characters a receiver can receive. (R/W)

Register 28.30. UART_POSPULSE_REG (0x0074)



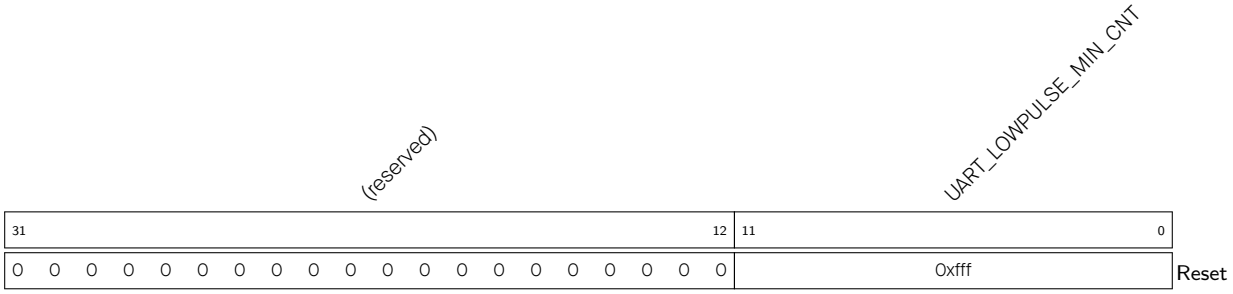
UART_POSEDGE_MIN_CNT Represents the minimal input clock counter value between two positive edges. It is used for baud rate detection. (RO)

Register 28.31. 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)

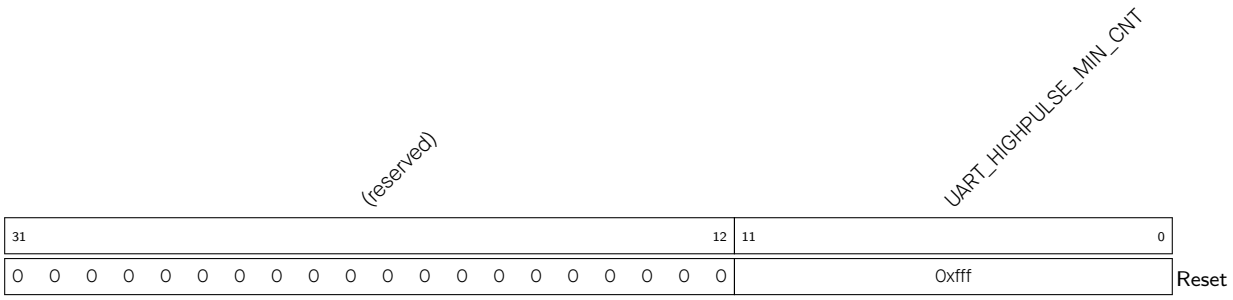
Register 28.32. UART_LOWPULSE_REG (0x007C)



UART_LOWPULSE_MIN_CNT Represents the minimum duration time of a low-level pulse. It is used for baud rate detection.

Measurement unit: PLL_CLK clock cycle. (RO)

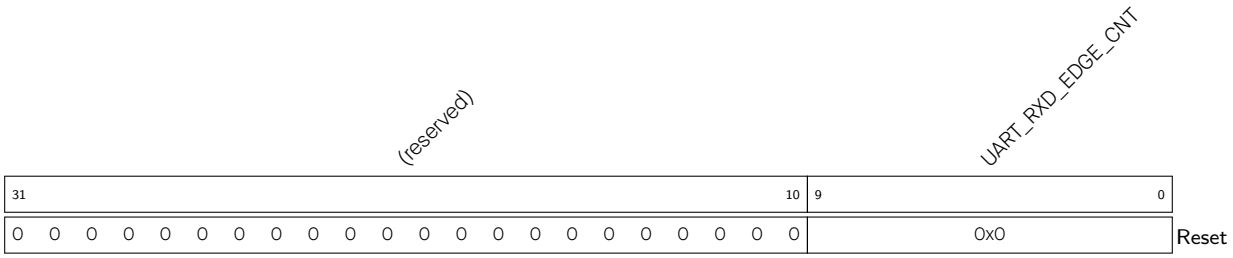
Register 28.33. UART_HIGHPULSE_REG (0x0080)



UART_HIGHPULSE_MIN_CNT Represents the maximum duration time for a high-level pulse. It is used for baud rate detection.

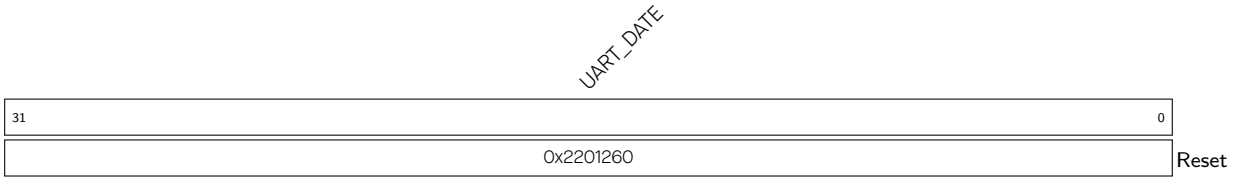
Measurement unit: PLL_CLK clock cycle. (RO)

Register 28.34. UART_RXD_CNT_REG (0x0084)



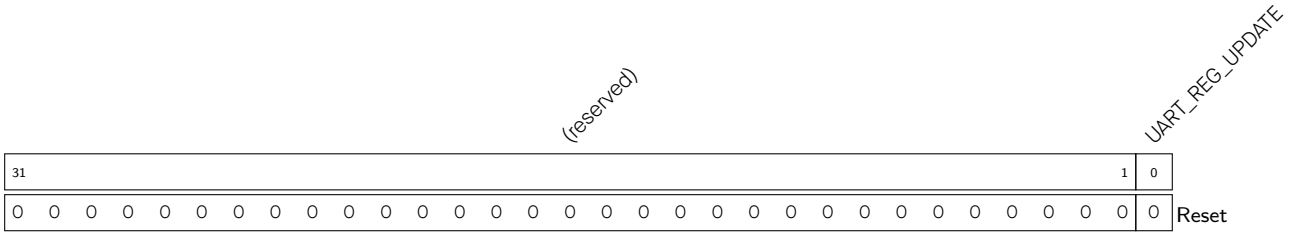
UART_RXD_EDGE_CNT Represents the number of RXD edge changes. It is used for baud rate detection. (RO)

Register 28.35. UART_DATE_REG (0x008C)



UART_DATE Version control register. (R/W)

Register 28.36. UART_REG_UPDATE_REG (0x0098)



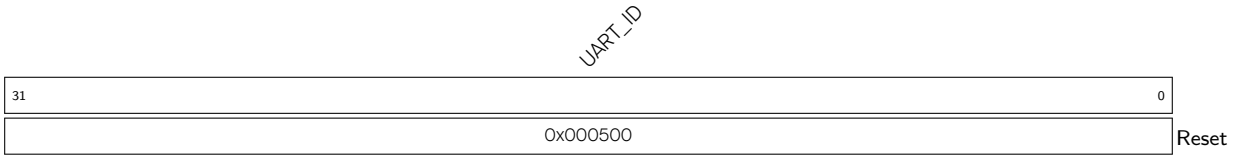
UART_REG_UPDATE Configures whether or not to synchronize registers.

0: Not synchronize

1: Synchronize

(R/W/SC)

Register 28.37. UART_ID_REG (0x009C)



UART_ID Configures the UART ID. (R/W)

28.7.2 UHCI Registers

The addresses in this section are relative to UHCI base address provided in Table 4.3-2 in Chapter 4 *System and Memory*.

Register 28.38. UHCI_CONF0_REG (0x0000)

UHCI_TX_RST Write 1 and then write 0 to reset the decoder state machine. (R/W)

UHCI_RX_RST Write 1 and then write 0 to reset the encoder state machine. (R/W)

UHCI_UART0_CE Configures whether or not to connect UHCI with UART0.

0: Not connect

1: Connect

(R/W)

UHCI_UART1_CE Configures whether or not to connect UHCI with UART1.

0: Not connect

1: Connect

(R/W)

UHCI_SEPER_EN Configures whether or not to separate the data frame with a special character.

0: Not separate

1: Separate

(R/W)

UHCI_HEAD_EN Configures whether or not to encode the data packet with a formatting header.

0: Not use formatting header

1: Use formatting header

(R/W)

UHCI_CRC_REC_EN Configures whether or not to enable the reception of the 16-bit CRC.

0: Disable

1: Enable

(R/W)

UHCI_UART_IDLE_EOF_EN Configures whether or not to stop receiving data when UART is idle.

0: Not stop

1: Stop

(R/W)

Continued on the next page...

Register 28.38. UHCI_CONF0_REG (0x0000)

Continued from the previous page...

UHCI_LEN_EOF_EN Configures when the UHCI decoder stops receiving data.

0: Stops after receiving 0xC0

1: Stops when the number of received data bytes reach the specified value. When UHCI_HEAD_EN is 1, the specified value is the data length indicated by the UHCI packet header; when UHCI_HEAD_EN is 0, the specified value is the configured value.

(R/W)

UHCI_ENCODE_CRC_EN Configures whether or not to enable data integrity check by appending a 16 bit CCITT-CRC to the end of the data.

0: Disable

1: Enable

(R/W)

UHCI_CLK_EN Configures clock gating.

0: Support clock only when the application writes registers.

1: Always force the clock on for registers.

(R/W)

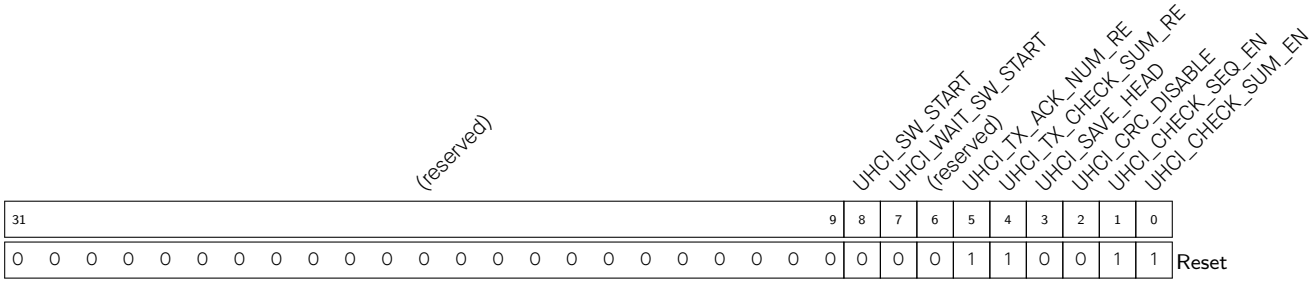
UHCI_UART_RX_BRK_EOF_EN Configures whether or not to stop UHCI from receiving data after UART has received a NULL frame.

0: Not stop

1: Stop

(R/W)

Register 28.39. UHCI_CONF1_REG (0x0014)



UHCI_CHECK_SUM_EN Configures whether or not to enable header checksum validation when UHCI receives a data packet.

0: Disable

1: Enable

(R/W)

UHCI_CHECK_SEQ_EN Configures whether or not to enable the sequence number check when UHCI receives a data packet.

0: Disable

1: Enable

(R/W)

UHCI_CRC_DISABLE Configures whether or not to enable CRC calculation.

0: Disable

1: Enable

Valid only when the Data Integrity Check Present bit in UHCI packet is 1.

(R/W)

UHCI_SAVE_HEAD Configures whether or not to save the packet header when UHCI receives a data packet.

0: Not save

1: Save

(R/W)

UHCI_TX_CHECK_SUM_RE Configures whether or not to encode the data packet with a checksum.

0: Not use checksum

1: Use checksum

(R/W)

UHCI_TX_ACK_NUM_RE Configures whether or not to encode the data packet with an acknowledgment when a reliable packet is to be transmitted.

0: Not use acknowledgement

1: Use acknowledgement

(R/W)

Continued on the next page...

Register 28.39. UHCI_CONF1_REG (0x0014)

Continued from the previous page...

UHCI_WAIT_SW_START Configures whether or not to put the UHCI encoder state machine to ST_SW_WAIT state.

0: No

1: Yes

(R/W)

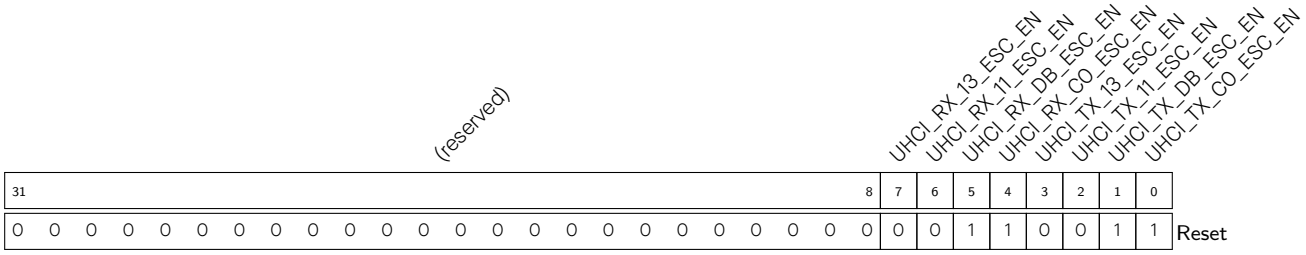
UHCI_SW_START Configures whether or not to send data packets when the encoder state machine is in ST_SW_WAIT state.

0: Not send

1: Send

(R/W/SC)

Register 28.40. UHCI_ESCAPE_CONF_REG (0x0020)



UHCI_TX_C0_ESC_EN Configures whether or not to decode character 0xC0 when DMA receives data.

0: Not decode

1: Decode

(R/W)

UHCI_TX_DB_ESC_EN Configures whether or not to decode character 0xDB when DMA receives data.

0: Not decode

1: Decode

(R/W)

UHCI_TX_11_ESC_EN Configures whether or not to decode flow control character 0x11 when DMA receives data.

0: Not decode

1: Decode

(R/W)

UHCI_TX_13_ESC_EN Configures whether or not to decode flow control character 0x13 when DMA receives data.

0: Not decode

1: Decode

(R/W)

UHCI_RX_C0_ESC_EN Configures whether or not to replace 0xC0 by special characters when DMA sends data.

0: Not replace

1: Replace

(R/W)

UHCI_RX_DB_ESC_EN Configures whether or not to replace 0xDB by special characters when DMA sends data.

0: Not replace

1: Replace

(R/W)

Continued on the next page...

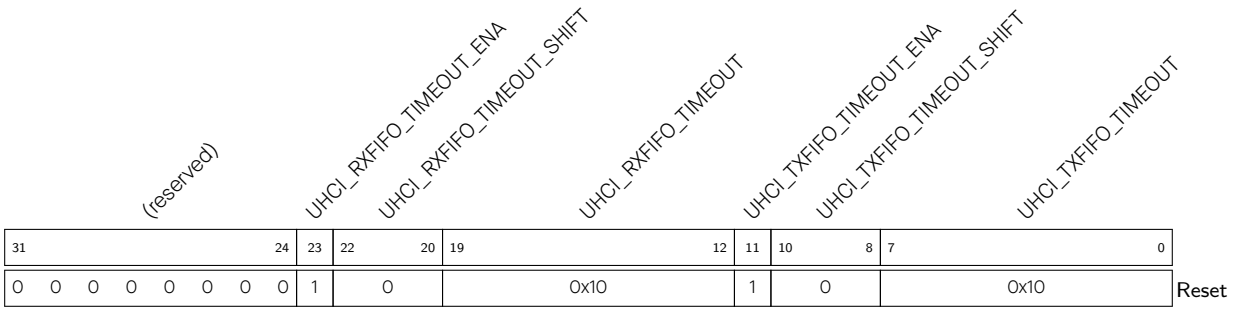
Register 28.40. UHCI_ESCAPE_CONF_REG (0x0020)

Continued from the previous page...

UHCI_RX_11_ESC_EN Configures whether or not to replace flow control character 0x11 by special characters when DMA sends data.
0: Not replace
1: Replace
(R/W)

UHCI_RX_13_ESC_EN Configures whether or not to replace flow control character 0x13 by special characters when DMA sends data.
0: Not replace
1: Replace
(R/W)

Register 28.41. UHCI_HUNG_CONF_REG (0x0024)



UHCI_TXFIFO_TIMEOUT Configures the timeout value for DMA data reception.
Measurement unit: ms. (R/W)

UHCI_TXFIFO_TIMEOUT_SHIFT Configures the upper limit of the timeout counter for TX FIFO. (R/W)

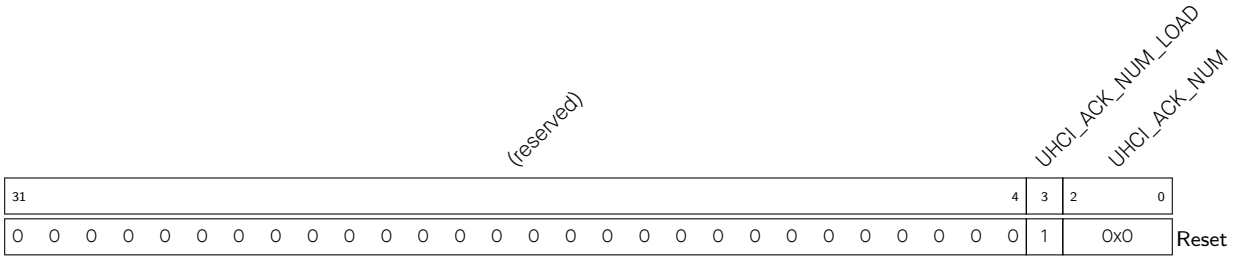
UHCI_TXFIFO_TIMEOUT_ENA Configures whether or not to enable the data reception timeout for TX FIFO.
0: Disable
1: Enable
(R/W)

UHCI_RXFIFO_TIMEOUT Configures the timeout value for DMA to read data from RAM.
Measurement unit: ms. (R/W)

UHCI_RXFIFO_TIMEOUT_SHIFT Configures the upper limit of the timeout counter for RX FIFO. (R/W)

UHCI_RXFIFO_TIMEOUT_ENA Configures whether or not to enable the DMA data transmission timeout.
0: Disable
1: Enable
(R/W)

Register 28.42. UHCI_ACK_NUM_REG (0x0028)



UHCI_ACK_NUM Configures the number of acknowledgements used in software flow control. (R/W)

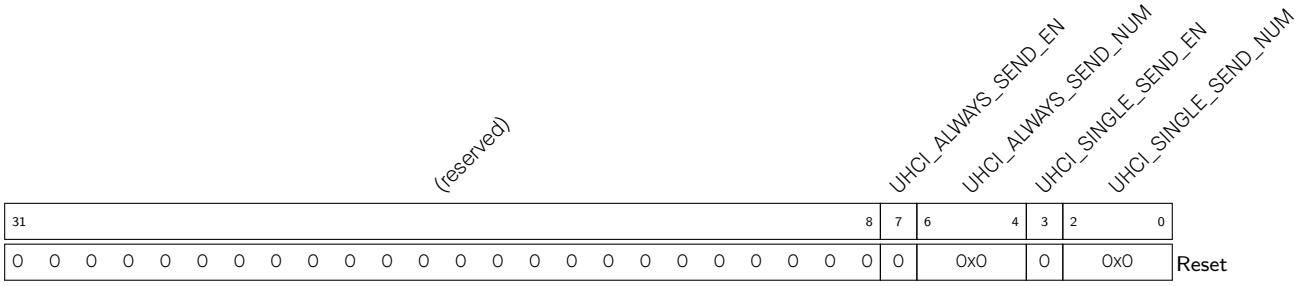
UHCI_ACK_NUM_LOAD Configures whether or not load acknowledgements.

0: Not load

1: Load

(WT)

Register 28.43. UHCI_QUICK_SENT_REG (0x0030)



UHCI_SINGLE_SEND_NUM Configures the source of data to be transmitted in single_send mode.

0: Q0 register

1: Q1 register

2: Q2 register

3: Q3 register

4: Q4 register

5: Q5 register

6: Q6 register

7: Invalid. No effect

(R/W)

UHCI_SINGLE_SEND_EN Configures whether or not to enable single_send mode.

0: Disable

1: Enable

(R/W/SC)

UHCI_ALWAYS_SEND_NUM Configures the source of data to be transmitted in always_send mode.

0: Q0 register

1: Q1 register

2: Q2 register

3: Q3 register

4: Q4 register

5: Q5 register

6: Q6 register

7: Invalid. No effect

(R/W)

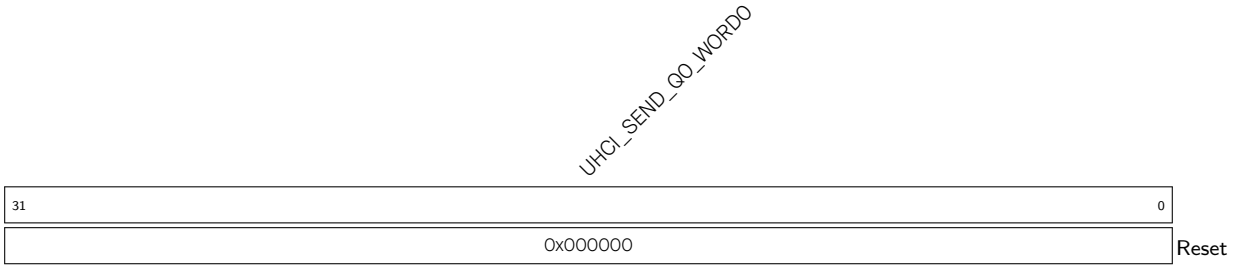
UHCI_ALWAYS_SEND_EN Configures whether or not to enable always_send mode.

0: Disable

1: Enable

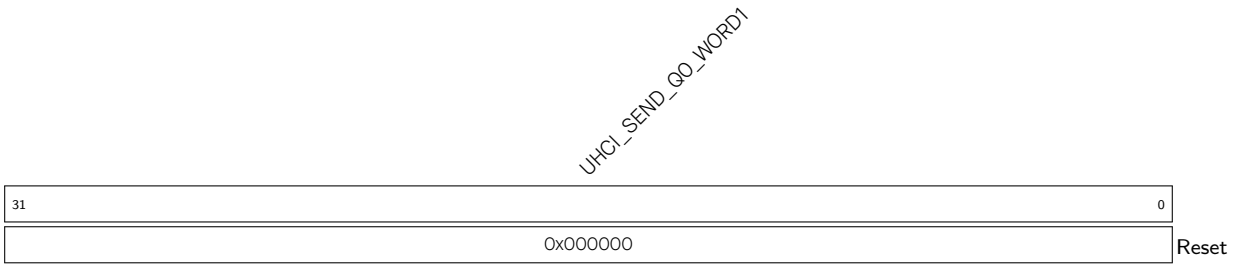
(R/W)

Register 28.44. UHCI_REG_Q0_WORD0_REG (0x0034)



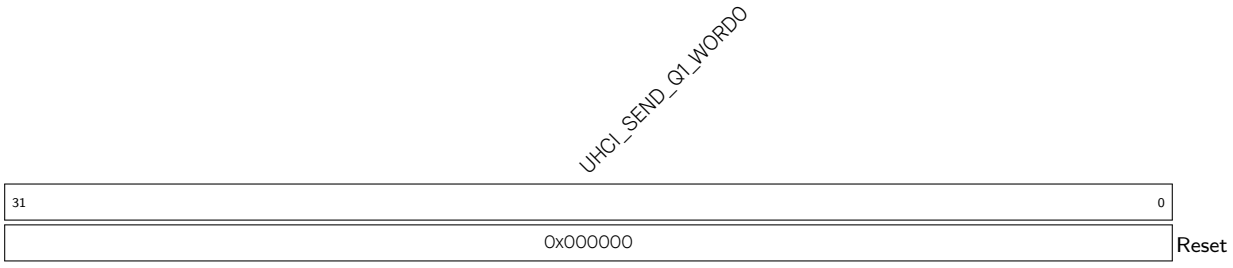
UHCI_SEND_Q0_WORD0 Data to be transmitted in Q0 register. (R/W)

Register 28.45. UHCI_REG_Q0_WORD1_REG (0x0038)



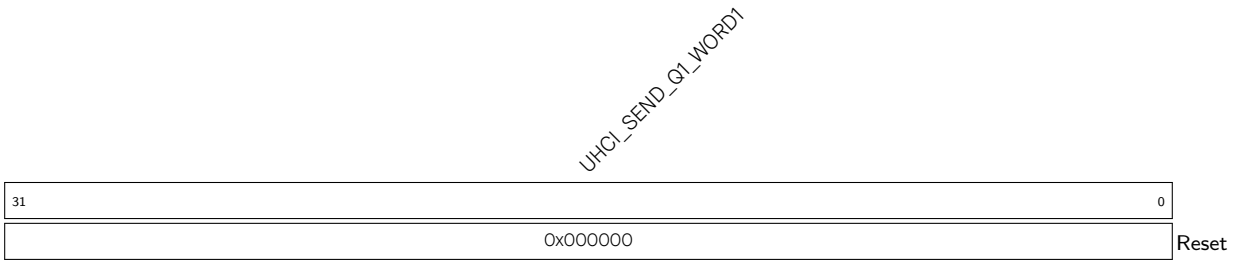
UHCI_SEND_Q0_WORD1 Data to be transmitted in Q0 register. (R/W)

Register 28.46. UHCI_REG_Q1_WORD0_REG (0x003C)



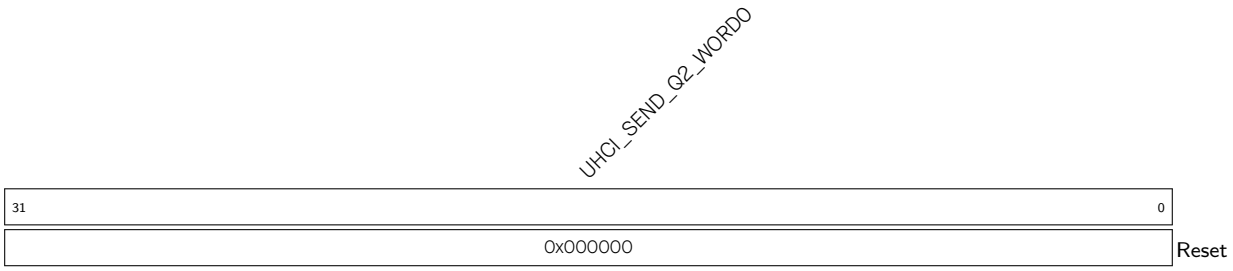
UHCI_SEND_Q1_WORD0 Data to be transmitted in Q1 register. (R/W)

Register 28.47. UHCI_REG_Q1_WORD1_REG (0x0040)



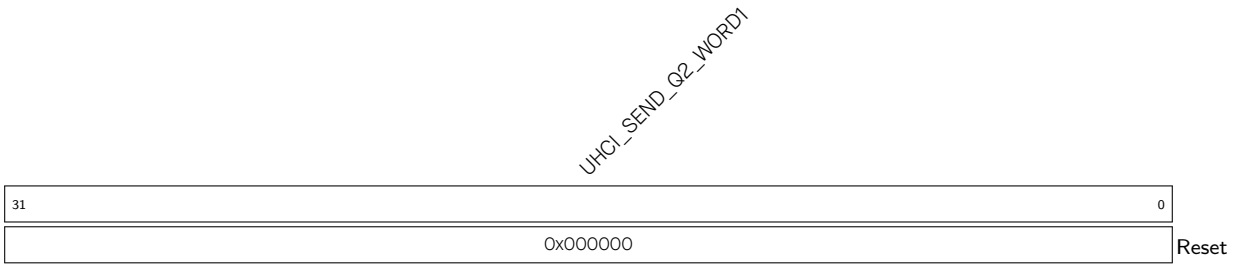
UHCI_SEND_Q1_WORD1 Data to be transmitted in Q1 register. (R/W)

Register 28.48. UHCI_REG_Q2_WORD0_REG (0x0044)



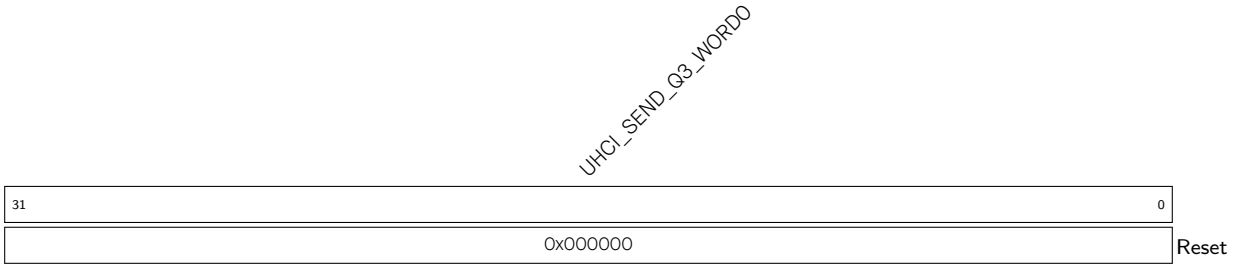
UHCI_SEND_Q2_WORD0 Data to be transmitted in Q2 register. (R/W)

Register 28.49. UHCI_REG_Q2_WORD1_REG (0x0048)



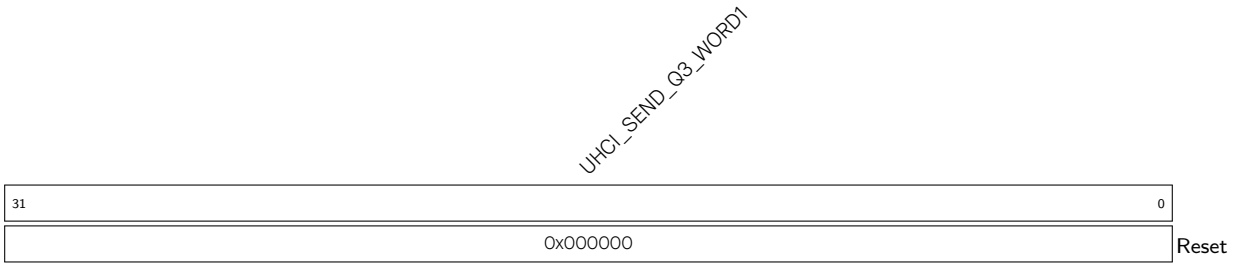
UHCI_SEND_Q2_WORD1 Data to be transmitted in Q2 register. (R/W)

Register 28.50. UHCI_REG_Q3_WORD0_REG (0x004C)



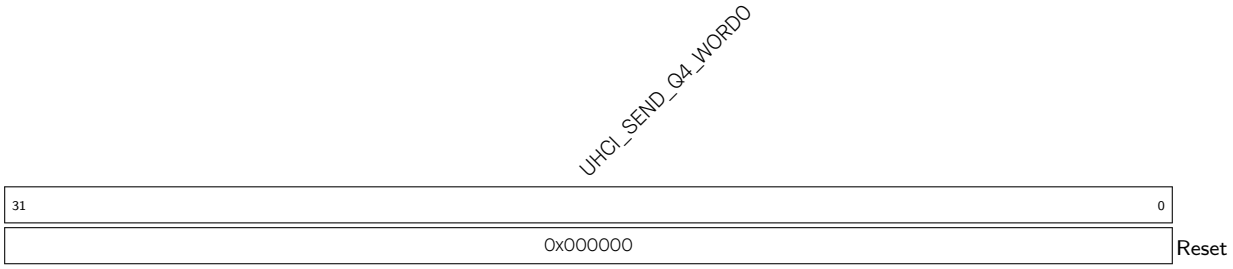
UHCI_SEND_Q3_WORD0 Data to be transmitted in Q3 register. (R/W)

Register 28.51. UHCI_REG_Q3_WORD1_REG (0x0050)



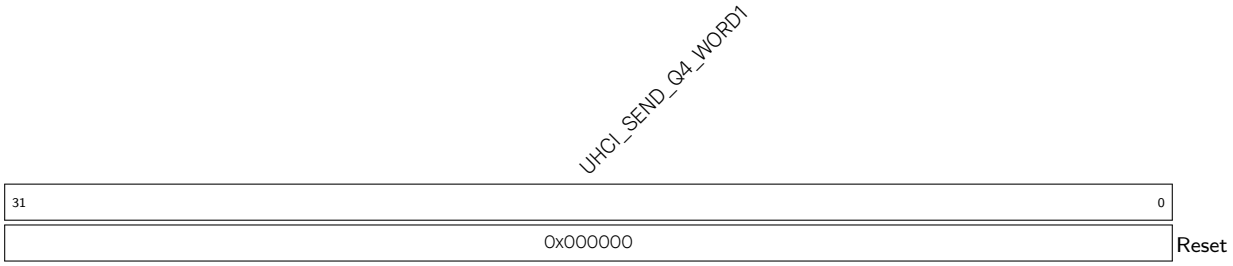
UHCI_SEND_Q3_WORD1 Data to be transmitted in Q3 register. (R/W)

Register 28.52. UHCI_REG_Q4_WORD0_REG (0x0054)



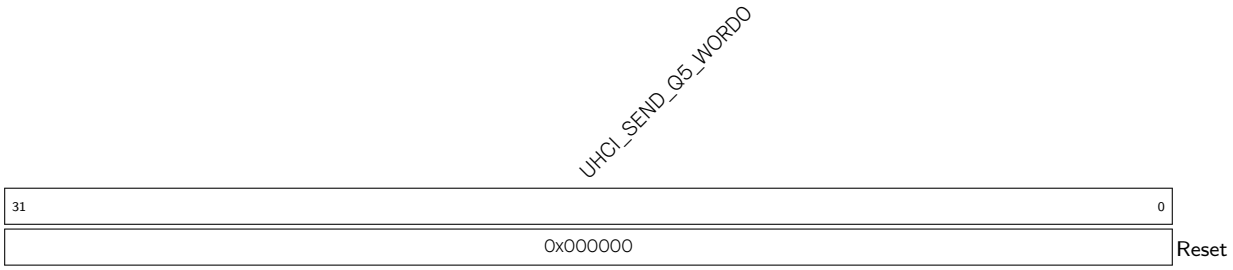
UHCI_SEND_Q4_WORD0 Data to be transmitted in Q4 register. (R/W)

Register 28.53. UHCI_REG_Q4_WORD1_REG (0x0058)



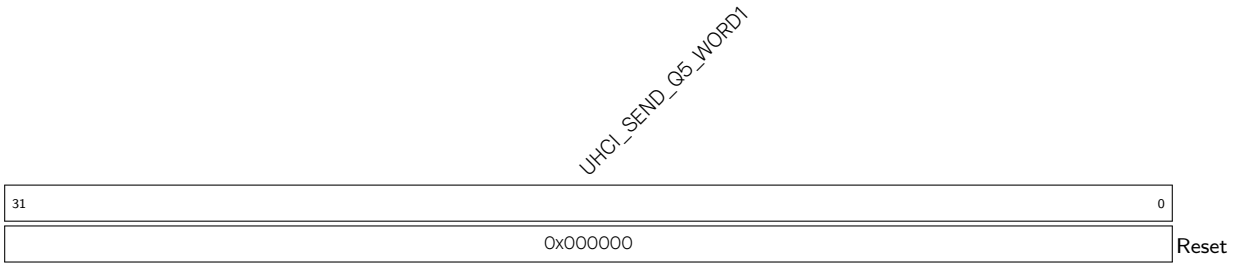
UHCI_SEND_Q4_WORD1 Data to be transmitted in Q4 register. (R/W)

Register 28.54. UHCI_REG_Q5_WORD0_REG (0x005C)



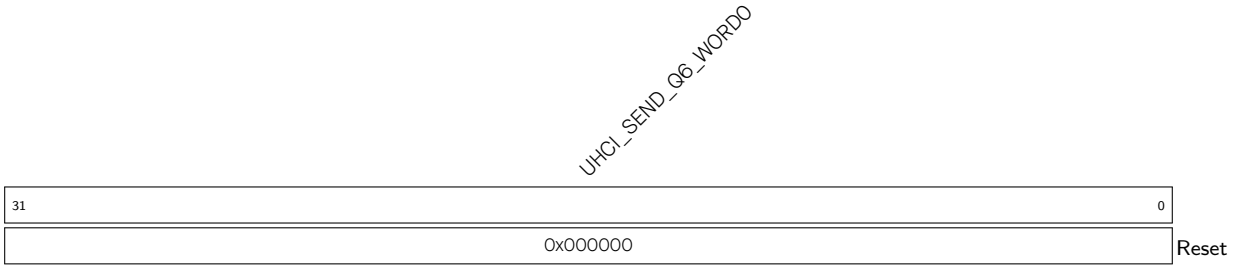
UHCI_SEND_Q5_WORD0 Data to be transmitted in Q5 register. (R/W)

Register 28.55. UHCI_REG_Q5_WORD1_REG (0x0060)



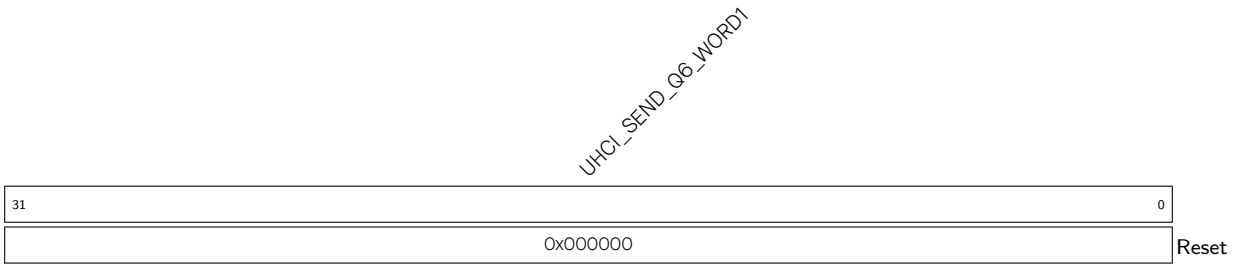
UHCI_SEND_Q5_WORD1 Data to be transmitted in Q5 register. (R/W)

Register 28.56. UHCI_REG_Q6_WORD0_REG (0x0064)



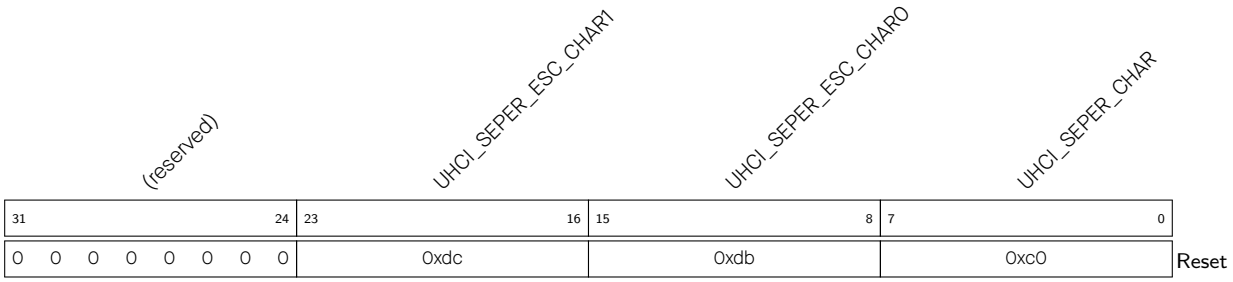
UHCI_SEND_Q6_WORD0 Data to be transmitted in Q6 register. (R/W)

Register 28.57. UHCI_REG_Q6_WORD1_REG (0x0068)



UHCI_SEND_Q6_WORD1 Data to be transmitted in Q6 register. (R/W)

Register 28.58. UHCI_ESC_CONF0_REG (0x006C)

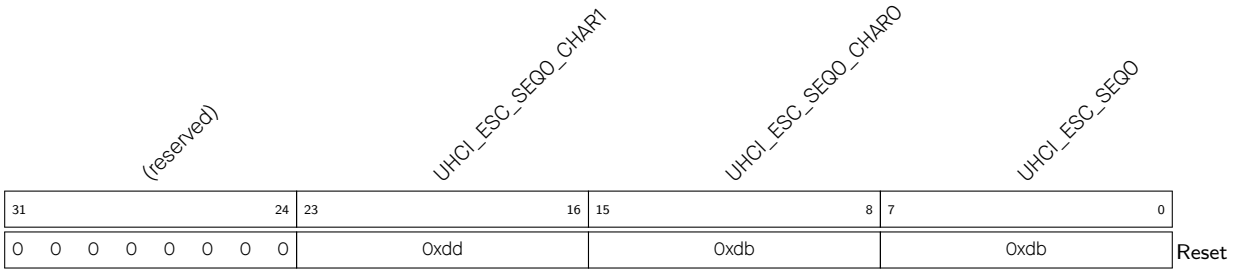


UHCI_SEPER_CHAR Configures separators to encode data packets. The default value is 0xC0. (R/W)

UHCI_SEPER_ESC_CHAR0 Configures the first character of SLIP escape sequence. The default value is 0xDB. (R/W)

UHCI_SEPER_ESC_CHAR1 Configures the second character of SLIP escape sequence. The default value is 0xDC. (R/W)

Register 28.59. UHCI_ESC_CONF1_REG (0x0070)

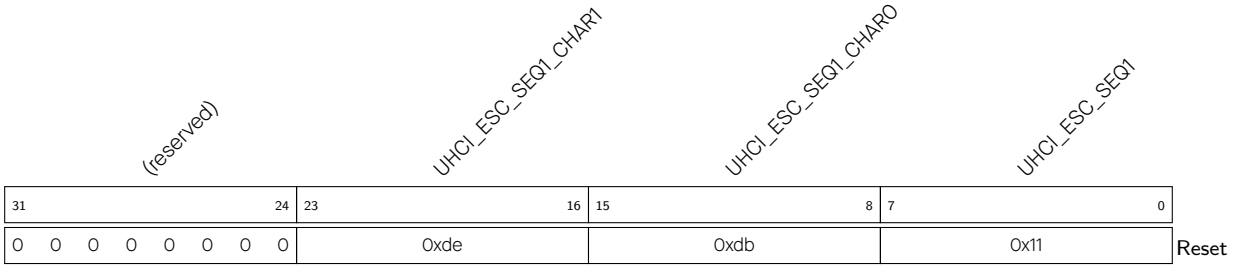


UHCI_ESC_SEQ0 Configures the character that needs to be encoded. The default value is 0xDB used as the first character of SLIP escape sequence. (R/W)

UHCI_ESC_SEQ0_CHAR0 Configures the first character of SLIP escape sequence. The default value is 0xDB. (R/W)

UHCI_ESC_SEQ0_CHAR1 Configures the second character of SLIP escape sequence. The default value is 0xDD. (R/W)

Register 28.60. UHCI_ESC_CONF2_REG (0x0074)

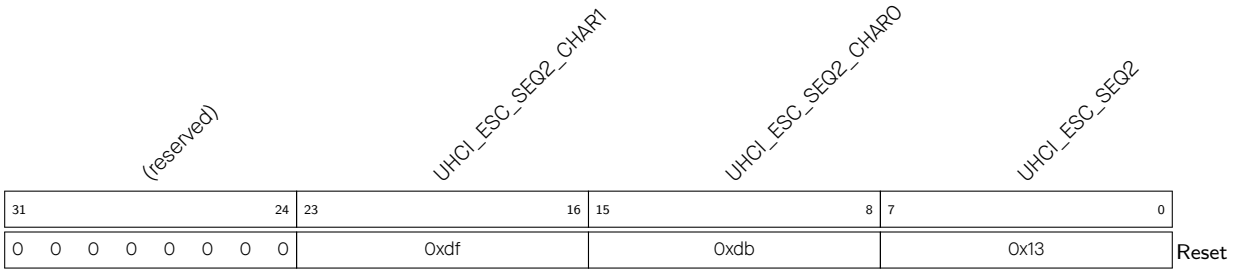


UHCI_ESC_SEQ1 Configures a character that need to be encoded. The default value is 0x11 used as a flow control character. (R/W)

UHCI_ESC_SEQ1_CHAR0 Configures the first character of SLIP escape sequence. The default value is 0xDB. (R/W)

UHCI_ESC_SEQ1_CHAR1 Configures the second character of SLIP escape sequence. The default value is 0xDE. (R/W)

Register 28.61. UHCI_ESC_CONF3_REG (0x0078)

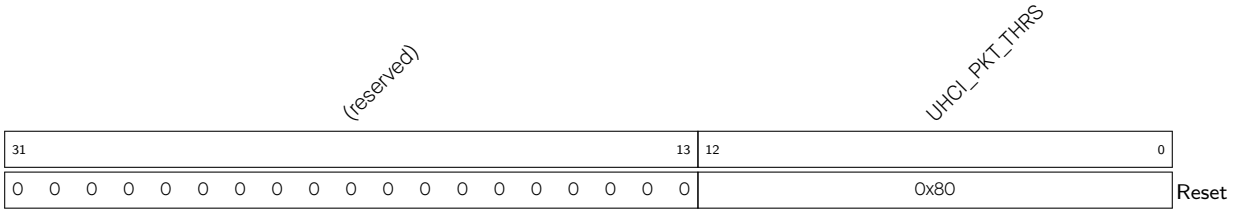


UHCI_ESC_SEQ2 Configures the character that needs to be decoded. The default value is 0x13 used as a flow control character. (R/W)

UHCI_ESC_SEQ2_CHAR0 Configures the first character of SLIP escape sequence. The default value is 0xDB. (R/W)

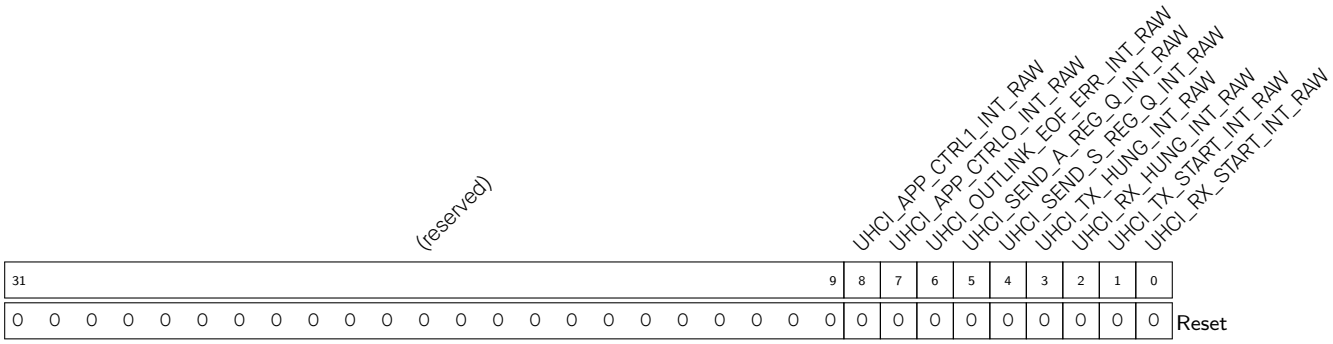
UHCI_ESC_SEQ2_CHAR1 Configures the second character of SLIP escape sequence. The default value is 0xDF. (R/W)

Register 28.62. UHCI_PKT_THRES_REG (0x007C)



UHCI_PKT_THRS Configures the maximum value of the packet length.
Measurement unit: byte. Valid only when UHCI_HEAD_EN is 0. (R/W)

Register 28.63. UHCI_INT_RAW_REG (0x0004)



UHCI_RX_START_INT_RAW The raw interrupt status of UHCI_RX_START_INT. (R/WTC/SS)

UHCI_TX_START_INT_RAW The raw interrupt status of UHCI_TX_START_INT. (R/WTC/SS)

UHCI_RX_HUNG_INT_RAW The raw interrupt status of UHCI_RX_HUNG_INT. (R/WTC/SS)

UHCI_TX_HUNG_INT_RAW The raw interrupt status of UHCI_TX_HUNG_INT. (R/WTC/SS)

UHCI_SEND_S_REG_Q_INT_RAW The raw interrupt status of UHCI_SEND_S_REG_Q_INT. (R/WTC/SS)

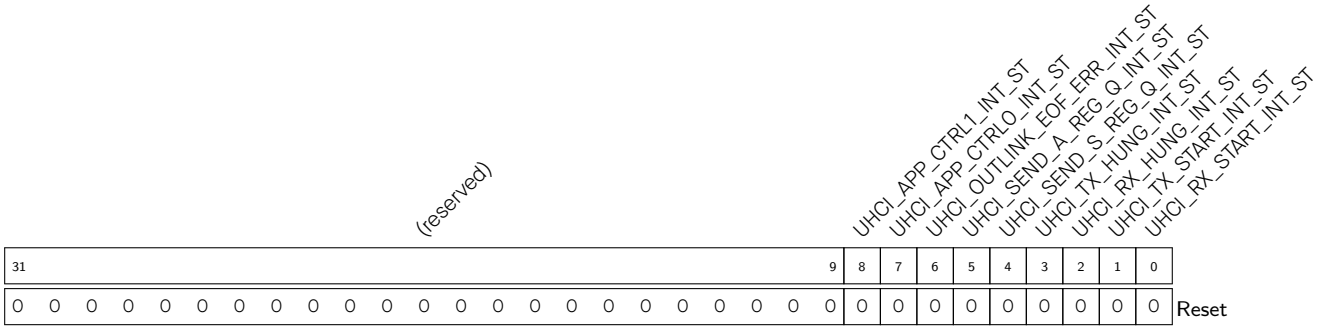
UHCI_SEND_A_REG_Q_INT_RAW The raw interrupt status of UHCI_SEND_A_REG_Q_INT. (R/WTC/SS)

UHCI_OUTLINK_EOF_ERR_INT_RAW The raw interrupt status of UHCI_OUTLINK_EOF_ERR_INT. (R/WTC/SS)

UHCI_APP_CTRL0_INT_RAW The raw interrupt status of UHCI_APP_CTRL0_INT. (R/W)

UHCI_APP_CTRL1_INT_RAW The raw interrupt status of UHCI_APP_CTRL1_INT. (R/W)

Register 28.64. UHCI_INT_ST_REG (0x0008)



UHCI_RX_START_INT_ST The masked interrupt status of UHCI_RX_START_INT. (RO)

UHCI_TX_START_INT_ST The masked interrupt status of UHCI_TX_START_INT. (RO)

UHCI_RX_HUNG_INT_ST The masked interrupt status of UHCI_RX_HUNG_INT. (RO)

UHCI_TX_HUNG_INT_ST The masked interrupt status of UHCI_TX_HUNG_INT. (RO)

UHCI_SEND_S_REG_Q_INT_ST The masked interrupt status of UHCI_SEND_S_REG_Q_INT. (RO)

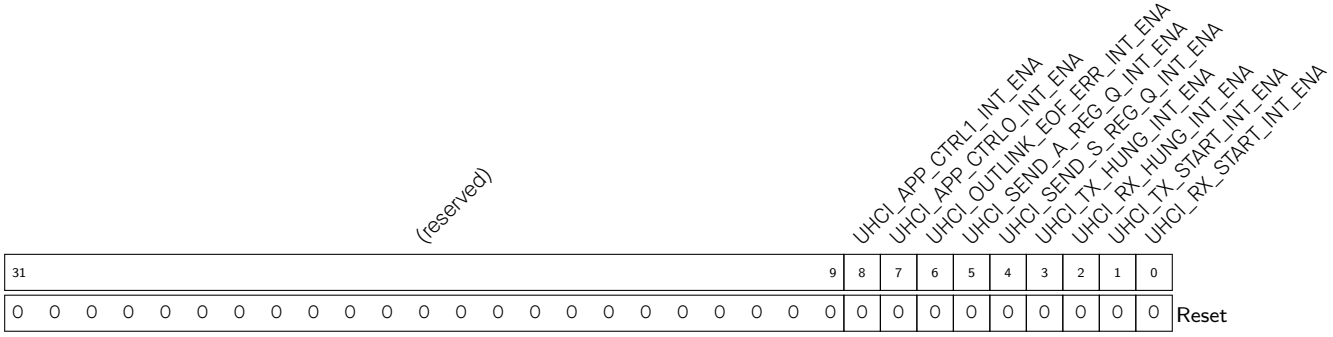
UHCI_SEND_A_REG_Q_INT_ST The masked interrupt status of UHCI_SEND_A_REG_Q_INT. (RO)

UHCI_OUTLINK_EOF_ERR_INT_ST The masked interrupt status of UHCI_OUTLINK_EOF_ERR_INT. (RO)

UHCI_APP_CTRL0_INT_ST The masked interrupt status of UHCI_APP_CTRL0_INT. (RO)

UHCI_APP_CTRL1_INT_ST The masked interrupt status of UHCI_APP_CTRL1_INT. (RO)

Register 28.65. UHCI_INT_ENA_REG (0x000C)



UHCI_RX_START_INT_ENA Write 1 to enable UHCI_RX_START_INT. (R/W)

UHCI_TX_START_INT_ENA Write 1 to enable UHCI_TX_START_INT. (R/W)

UHCI_RX_HUNG_INT_ENA Write 1 to enable UHCI_RX_HUNG_INT. (R/W)

UHCI_TX_HUNG_INT_ENA Write 1 to enable UHCI_TX_HUNG_INT. (R/W)

UHCI_SEND_S_REG_Q_INT_ENA Write 1 to enable UHCI_SEND_S_REG_Q_INT. (R/W)

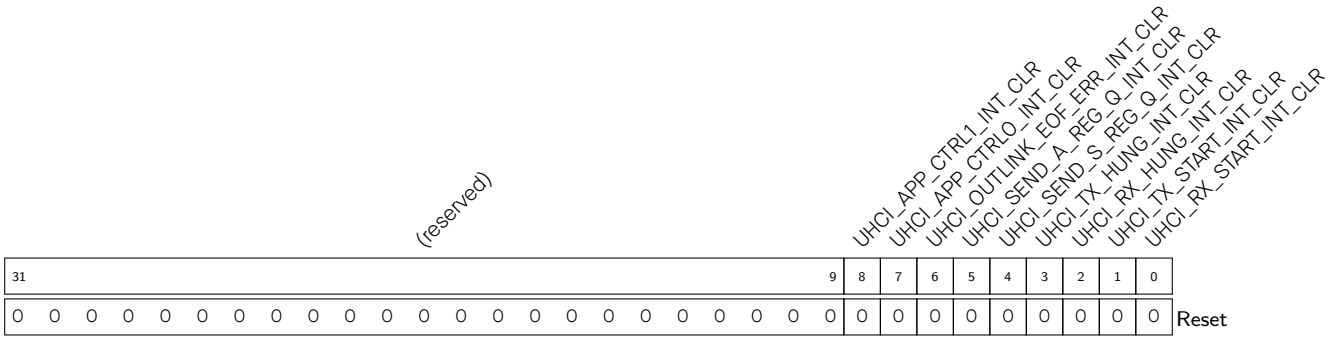
UHCI_SEND_A_REG_Q_INT_ENA Write 1 to enable UHCI_SEND_A_REG_Q_INT. (R/W)

UHCI_OUTLINK_EOF_ERR_INT_ENA Write 1 to enable UHCI_OUTLINK_EOF_ERR_INT. (R/W)

UHCI_APP_CTRL0_INT_ENA Write 1 to enable UHCI_APP_CTRL0_INT. (R/W)

UHCI_APP_CTRL1_INT_ENA Write 1 to enable UHCI_APP_CTRL1_INT. (R/W)

Register 28.66. UHCI_INT_CLR_REG (0x0010)



UHCI_RX_START_INT_CLR Write 1 to clear UHCI_RX_START_INT. (WT)

UHCI_TX_START_INT_CLR Write 1 to clear UHCI_TX_START_INT. (WT)

UHCI_RX_HUNG_INT_CLR Write 1 to clear UHCI_RX_HUNG_INT. (WT)

UHCI_TX_HUNG_INT_CLR Write 1 to clear UHCI_TX_HUNG_INT. (WT)

UHCI_SEND_S_REG_Q_INT_CLR Write 1 to clear UHCI_SEND_S_REG_Q_INT. (WT)

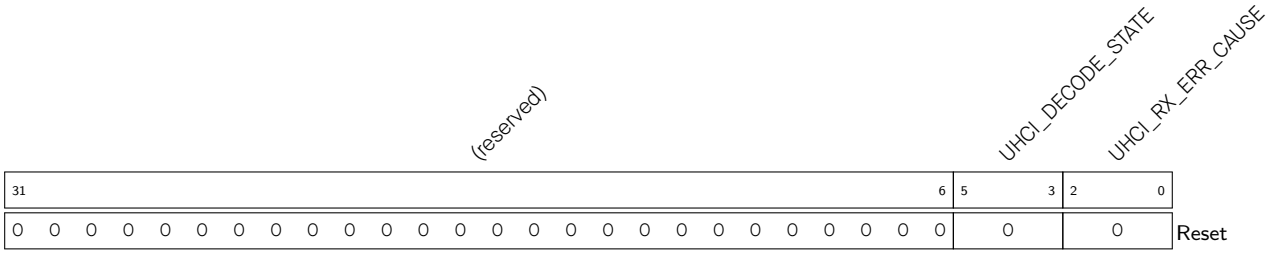
UHCI_SEND_A_REG_Q_INT_CLR Write 1 to clear UHCI_SEND_A_REG_Q_INT. (WT)

UHCI_OUTLINK_EOF_ERR_INT_CLR Write 1 to clear UHCI_OUTLINK_EOF_ERR_INT. (WT)

UHCI_APP_CTRL0_INT_CLR Write 1 to clear UHCI_APP_CTRL0_INT. (WT)

UHCI_APP_CTRL1_INT_CLR Write 1 to clear UHCI_APP_CTRL1_INT. (WT)

Register 28.67. UHCI_STATE0_REG (0x0018)



UHCI_RX_ERR_CAUSE Represents the error type when DMA has received a packet with error.

0: Invalid. No effect

1: Checksum error in the HCI packet

2: Sequence number error in the HCI packet

3: CRC bit error in the HCI packet

4: 0xC0 is found but the received HCI packet is not complete

5: 0xC0 is not found when the HCI packet has been received

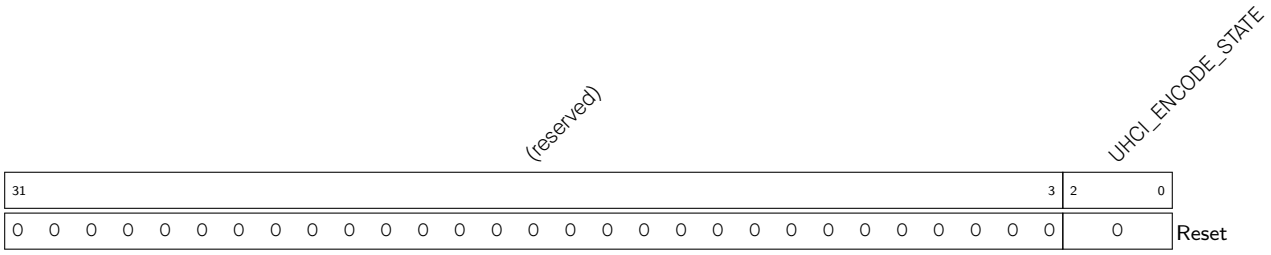
6: CRC check error

7: Invalid. No effect

(RO)

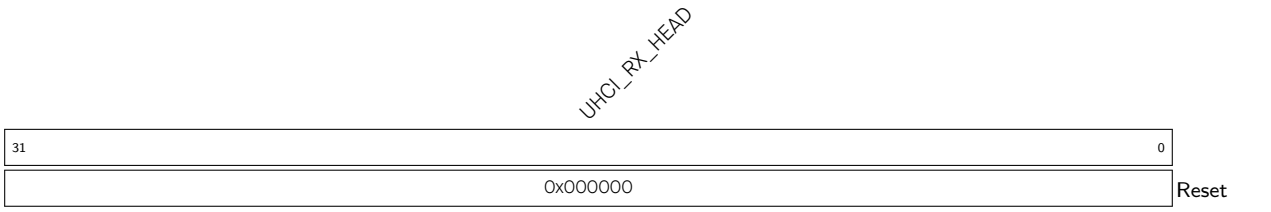
UHCI_DECODE_STATE Represents the UHCI decoder status. (RO)

Register 28.68. UHCI_STATE1_REG (0x001C)



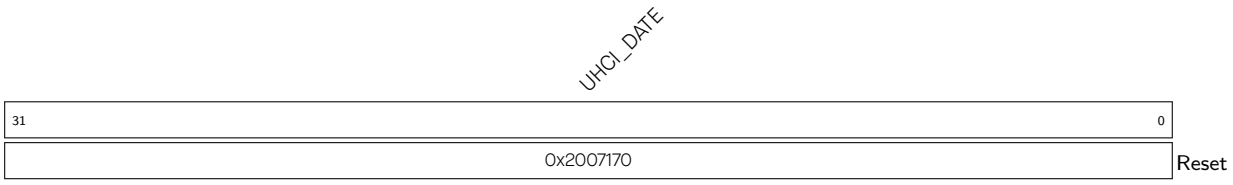
UHCI_ENCODE_STATE Represents the UHCI encoder status. (RO)

Register 28.69. UHCI_RX_HEAD_REG (0x002C)



UHCI_RX_HEAD Represents the header of the current received packet. (RO)

Register 28.70. UHCI_DATE_REG (0x0080)



UHCI_DATE Version control register. (R/W)

Chapter 29

SPI Controller (SPI)

29.1 Overview

The Serial Peripheral Interface (SPI) is a synchronous serial interface useful for communication with external peripherals. The ESP32-H2 chip integrates three SPI controllers:

- SPI0,
- SPI1,
- and 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.

29.2 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.

29.3 Features

Some of the key features of GP-SPI2 are:

- 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 48 MHz
  - Slave: up to 32 MHz
- Configurable data length:
  - CPU-controlled transfer as master or as slave: 1 ~ 64 B
  - 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
- Six 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

29.4 Architectural Overview

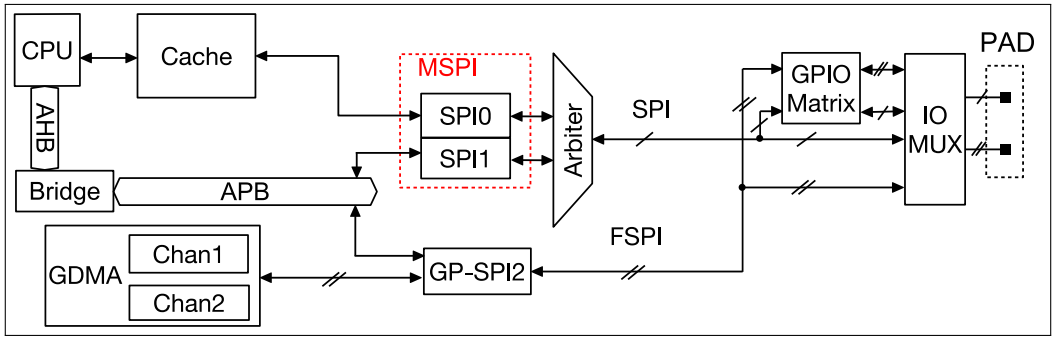


Figure 29.4-1. SPI Module Overview

Figure 29.4-1 shows an overview of SPI module. GP-SPI2 exchanges data with SPI devices by 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 GPIO matrix or IO MUX. For more information, see Chapter 6 *IO MUX and GPIO Matrix (GPIO, IO MUX)*.

29.5 Functional Description

29.5.1 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 29.5-1.

Table 29.5-1. Data Modes Supported by GP-SPI2

Supported 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 data modes used when GP-SPI2 works as a master or a slave, see Section 29.5.9 and Section 29.5.10, respectively.

29.5.2 Introduction to FSPI Bus Signals

The functional description of FSPI bus signals is shown in Table 29.5-2. Table 29.5-3 lists the signals used in various SPI modes.

Table 29.5-2. 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 29.5-3. Signals Used in Various SPI Modes

FSPI 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.

29.5.3 Bit Read/Write Order Control

When operating as master:

- The bit order of the command, address, and data sent by the GP-SPI2 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 slave:

- The bit order of the data sent by the GP-SPI2 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 29.5-4 shows the function of SPI_RD/WR_BIT_ORDER.

Table 29.5-4. 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

29.5.4 Unaligned Byte Transfer

- When used as a master, the GP-SPI master can transmit and receive data by bit, with the bit length equal to SPI_MS_DATA_BITLEN + 1. The bit length must be a multiple of 1/2/4/8 in 1-line, 2-line, 4-line, and 8-line modes, respectively.
  - When transmitting data that is not a multiple of 8 bits, the software must pad the remaining bits to form a complete byte.
  - When receiving data that is not a multiple of 8 bits, the remaining bits are still received as a full byte.
- When used as a slave, the GP-SPI slave can also transmit and receive data by bit, with the bit length being a multiple of 1/2/4/8 in 1-line, 2-line, 4-line, and 8-line modes, respectively.
  - When transmitting data that is not a multiple of 8 bits, the software must pad the remaining bits to form a complete byte.
  - When receiving data that is not a multiple of 8 bits, the remaining bits are still received as a full byte. The total received length can be obtained via SPI_SLV_DATA_BITLEN, and the number of valid bits in the last byte can be identified by using SPI_SLV_LAST_BYTE_STRB.

Note:

- Regardless of whether GP-SPI is used as a master or a slave, the bit order for transmitting/receiving partial bytes follows the configured setting.

29.5.5 Transfer Types

The transfer types supported by GP-SPI2 when working as a master or a slave are shown on Table 29.5-5.

Table 29.5-5. 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.

29.5.6 CPU-Controlled Data Transfer

GP-SPI2 provides 16 x 32-bit data buffers, i.e., SPI_W0_REG ~ SPI_W15_REG, as shown in Figure 29.5-1. 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 a 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.

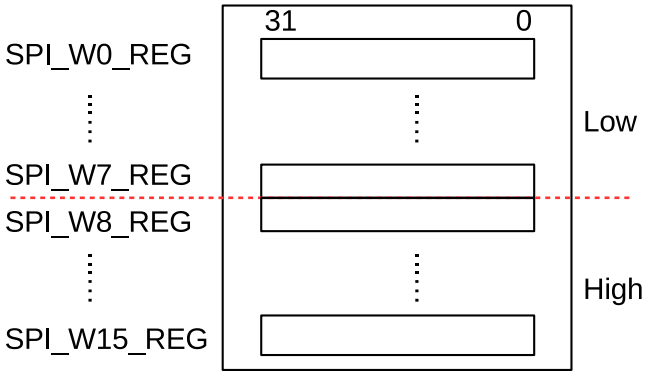


Figure 29.5-1. Data Buffer Used in CPU-Controlled Transfer

29.5.6.1 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 list 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 bytes, 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, respectively.
    * *Byte 64* ~ *Byte 255* are read from SPI_W15_REG[31:24] repeatedly.
    * *Byte 256* ~ *Byte 319* (the first 64 bytes in the another 256 bytes) are read from SPI_W0_REG ~ SPI_W15_REG again, respectively, 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, respectively.
    * *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, respectively. 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, respectively.

  * *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, respectively, 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, respectively.
  * *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, respectively. 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, respectively.
    * *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, respectively, 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, respectively.
    * *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, respectively. 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, respectively.
    * *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, respectively.

    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, respectively.
* *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, respectively. 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_WO_REG[7:0], and Address 1 for SPI_WO_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.

29.5.6.2 CPU-Controlled Slave Transfer

In a CPU-controlled slave full-duplex or half-duplex transfer, the RX data or TX data is saved to or sent from SPI_WO_REG ~ SPI_W15_REG, which are byte-addressable.

- In full-duplex communication, the address of SPI_WO_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_WO_REG ~ SPI_W15_REG will be overwritten, same as the behaviors described in the master mode 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_WO_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 mode when high part mode is enabled.

According to your applications, the registers SPI_WO_REG ~ SPI_W15_REG can be used as:

- data buffers only
- data buffers and status buffers
- status buffers only

29.5.7 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 both as master and as slave.

A DMA-controlled transfer can be:

- a single transfer, consisting of only one transaction. GP-SPI2 supports this transfer both as master and as slave.
- a configurable segmented transfer, consisting of several transactions (segments). GP-SPI2 supports this transfer only as master. For more information, see Section 29.5.9.5.
- a slave segmented transfer, consisting of several transactions (segments). GP-SPI2 supports this transfer only as slave. For more information, see Section 29.5.10.3.

A DMA-controlled transfer only needs to be triggered once by CPU. When such a transfer is triggered, data is transferred by the GDMA 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 29.5.9 and Section 29.5.10. 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.

29.5.7.1 GDMA Configuration

- Select a GDMA channel*n*, and configure a GDMA TX/RX descriptor. See Chapter 3 *GDMA Controller (GDMA)*.
- Set the bit GDMA_INLINK_START_CH*n* or GDMA_OUTLINK_START_CH*n* to start GDMA RX engine and 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 operating 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 operating 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 29.5-6. 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

29.5.7.2 GDMA TX/RX Buffer Length Control

It is recommended that the length of configured GDMA TX/RX buffer is equal to the length of actual data transferred.

- If the length of configured GDMA TX buffer is shorter than that of 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 configured GDMA TX buffer is longer than that of actual data transferred, the TX buffer is not fully used, and the remaining buffer will be used for following transaction even if a new TX buffer is linked later. Please keep it in mind. Or save the unused data and reset DMA.
- If the length of configured GDMA RX buffer is shorter than that of actual data transferred, the extra data will be lost. The interrupts SPI_INFIFO_FULL_ERR_INT and SPI_TRANS_DONE_INT are triggered. But GDMA_IN_SUC_EOF_CH*n*_INT interrupt is not generated.
- If the length of configured GDMA RX buffer is longer than that of 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.

29.5.8 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 modes, configure SPI_DMA_RX_ENA and SPI_DMA_TX_ENA before the transfer starts.

29.5.8.1 GP-SPI2 Functional Blocks

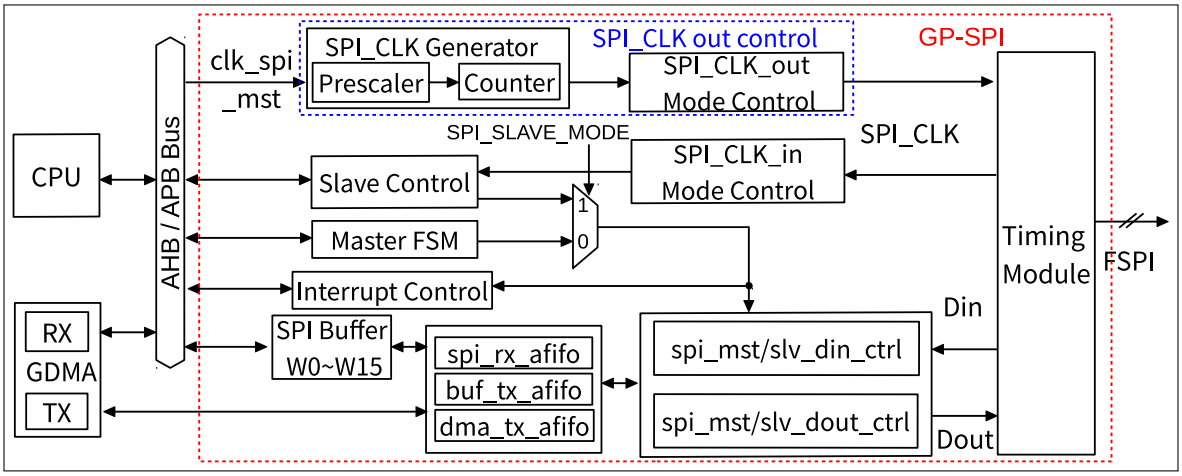


Figure 29.5-2. GP-SPI2 Block Diagram

Figure 29.5-2 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 29.5-1. The data transferred in CPU-controlled mode 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 the module clock of GP-SPI2 and derived from PLL_CLK. It is used in GP-SPI2 as master to generate SPI_CLK signal for data transfer and for slaves.
- SPI_CLK Generator: generates SPI_CLK by dividing clk_spi_mst. The divider is determined by SPI_CLKCNT_N and SPI_CLKDIV_PRE. See Section 29.7.
- 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 a slave.

29.5.8.2 Data Flow Control as Master

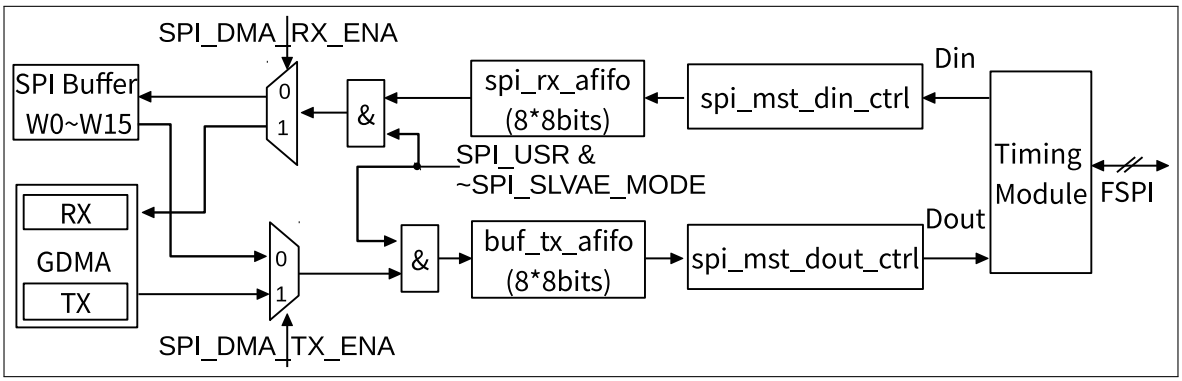


Figure 29.5-3. Data Flow Control in GP-SPI2 as Master

Figure 29.5-3 shows the data flow of GP-SPI2 as master. Its control logic is as follows:

- RX data: data in FSPI bus is captured by Timing Module, converted in units of bytes by *spi_mst_din_ctrl* module, then buffered in *spi_rx_afifo*, and finally stored in corresponding addresses according to the transfer modes.
  - 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.

29.5.8.3 Data Flow Control as Slave

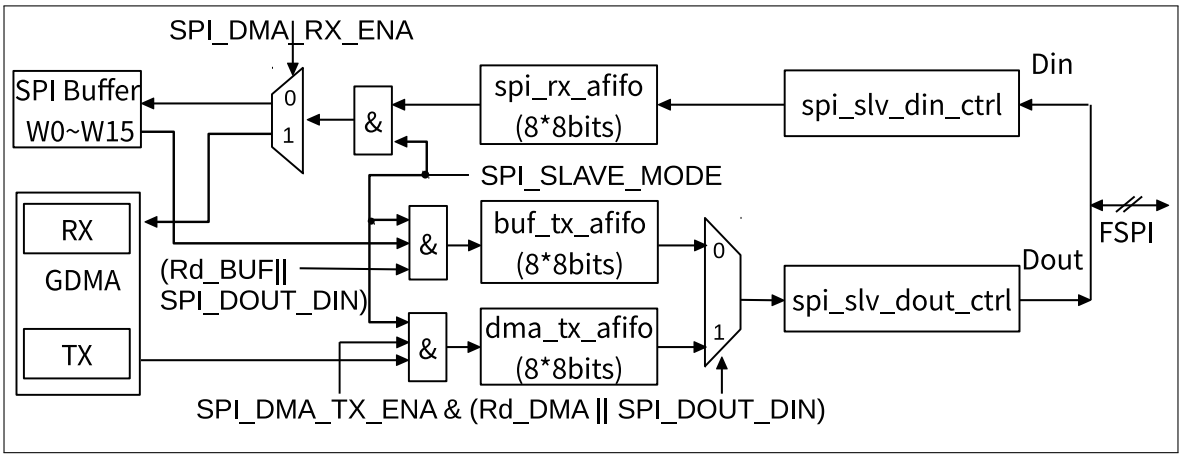


Figure 29.5-4. Data Flow Control in GP-SPI2 as Slave

Figure 29.5-4 shows the data flow in GP-SPI2 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 unit of 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. TX data comes from corresponding addresses according the transfer modes.
  - 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 GDMA 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.

29.5.9 GP-SPI2 as a Master

GP-SPI2 can be configured as a 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-SPI2 module clock) and six CS lines (CS0 ~ CS5).

Note:

- The length of transferred data must be an integral multiple of byte (8 bits), otherwise the extra bits will be lost. The extra bits here means the result of total data bits mod 8.
- To transfer bits that is not an integral multiple of byte (8 bits), consider implementing it in CMD state or ADDR state.

29.5.9.1 State Machine

When GP-SPI2 works as a master, the state machine controls its 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 29.5-5 shows the workflow of GP-SPI2 state machine.

1. IDLE: GP-SPI2 is not active or is operating as 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*.

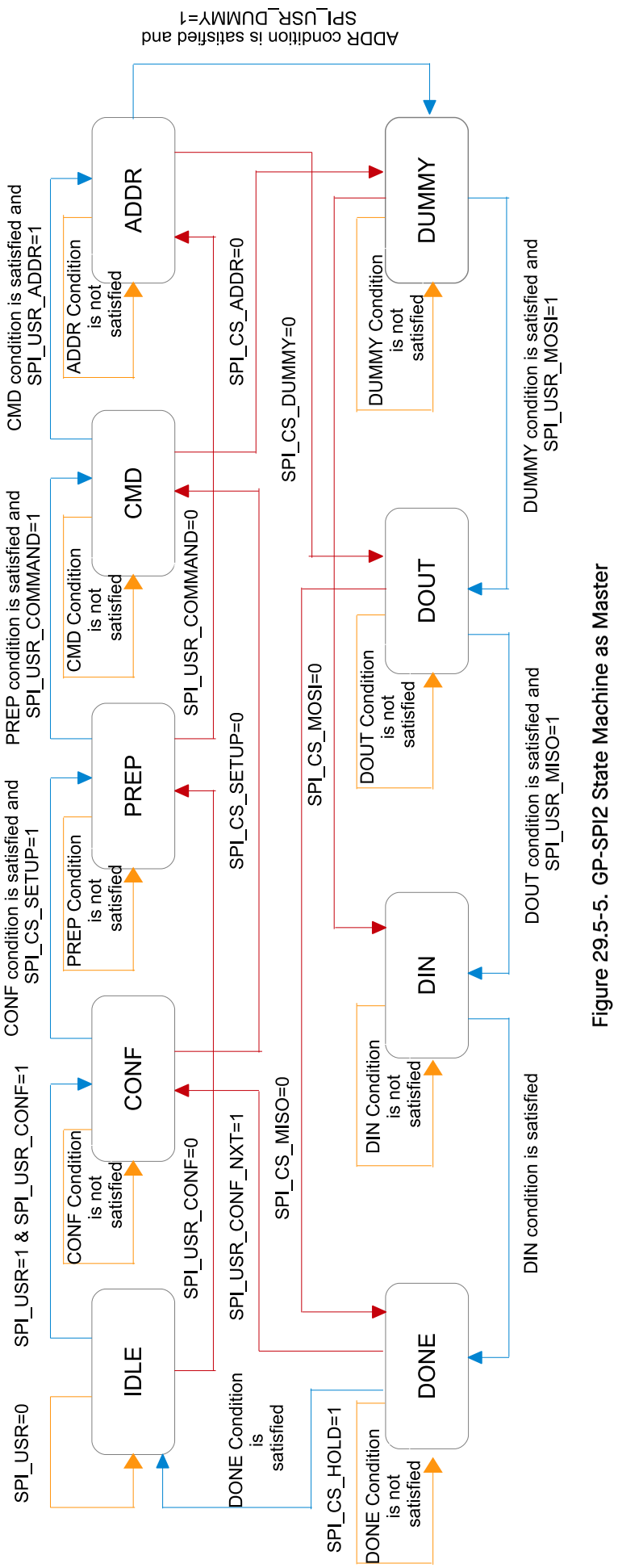


Figure 29.5-5. GP-SPI2 State Machine as Master

Legend to state flow:

- —: corresponding state condition is not satisfied; repeats current state.
- —: corresponding registers are set and conditions are satisfied; goes to next state.
- —: state registers are not set; skips one or more following states, depending on the registers of the following states are set or not.

Explanation to the conditions listed in the figure above:

- CONF condition: gpc[17:0] >= SPI_CONF_BITLEN[17:0]
- PREP condition: gpc[4:0] >= SPI_CS_SETUP_TIME[4:0]
- CMD condition: gpc[3:0] >= SPI_USR_COMMAND_BITLEN[3:0]
- ADDR condition: gpc[4:0] >= SPI_USR_ADDR_BITLEN[4:0]
- DUMMY condition: gpc[7:0] >= SPI_USR_DUMMY_CYCLELEN[7:0]
- DOUT condition: gpc[17:0] >= SPI_MS_DATA_BITLEN[17:0]
- DIN condition: gpc[17:0] >= SPI_MS_DATA_BITLEN[17:0]
- DONE condition: (gpc[4:0] >= SPI_CS_HOLD_TIME[4:0] || SPI_CS_HOLD == 1'b0)

A counter (gpc[17:0]) is used in the state machine to control the cycle length of each state. The states CONF, PREP, CMD, ADDR, DUMMY, DOUT, and DIN can be enabled or disabled independently. The cycle length of each state can also be configured independently.

29.5.9.2 Register Configuration for State and Bit Mode Control

Introduction

The registers, related to GP-SPI2 state control, are listed in Table 29.5-7. Users can enable QPI mode for GP-SPI2 by setting the bit SPI_QPI_MODE in register SPI_USER_REG.

Table 29.5-7. 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 SPI_USR_COMMAND_BITLEN SPI_USR_COMMAND	SPI_USR_COMMAND_VALUE SPI_USR_COMMAND_BITLEN SPI_FCMD_DUAL SPI_USR_COMMAND	SPI_USR_COMMAND_VALUE SPI_USR_COMMAND_BITLEN SPI_FCMD_QUAD SPI_USR_COMMAND
ADDR	SPI_USR_ADDR_VALUE SPI_USR_ADDR_BITLEN SPI_USR_ADDR	SPI_USR_ADDR_VALUE SPI_USR_ADDR_BITLEN SPI_USR_ADDR SPI_FADDR_DUAL	SPI_USR_ADDR_VALUE SPI_USR_ADDR_BITLEN SPI_USR_ADDR SPI_FADDR_QUAD
DUMMY	SPI_USR_DUMMY_CYCLELEN SPI_USR_DUMMY	SPI_USR_DUMMY_CYCLELEN SPI_USR_DUMMY	SPI_USR_DUMMY_CYCLELEN SPI_USR_DUMMY
DIN	SPI_USR_MISO SPI_MS_DATA_BITLEN	SPI_USR_MISO SPI_MS_DATA_BITLEN SPI_FREAD_DUAL	SPI_USR_MISO SPI_MS_DATA_BITLEN SPI_FREAD_QUAD

Table 29.5-7. 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
DOUT	SPI_USR_MOSI SPI_MS_DATA_BITLEN	SPI_USR_MOSI SPI_MS_DATA_BITLEN SPI_FWRITE_DUAL	SPI_USR_MOSI SPI_MS_DATA_BITLEN SPI_FWRITE_QUAD

As shown in Table 29.5-7, 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 is 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 = bit length expected - 1.

- Set SPI_FREAD_QUAD and SPI_USR_MISO.
- Clear SPI_FREAD_DUAL.
- Configure GDMA in DMA-controlled mode. In CPU-controlled mode, 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:
The register configurations in master mode covered in this chapter all require setting the SPI_UPDATE bit, which updates the configuration data from the AHB_CLK domain to the clk_spi_mst clock domain. This operation is not necessary in the slave mode. See above example for specific configuration.

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) in CPU-controlled mode, or GDMA TX buffer in DMA-controlled mode. The data byte order is incremented from LSB (byte 0) to MSB.

Pay special attention to the command value in SPI_USR_COMMAND_VALUE and to address value in SPI_USR_ADDR_VALUE.

The configuration of command value is as follows:

Table 29.5-8. 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] is sent.
		0	COMMAND_VALUE[7:0] is sent first, and then COMMAND_VALUE[15:15 - COMMAND_BITLEN] is sent.

[1] SPI_USR_COMMAND_BITLEN: this field is used to configure the bit length of the command.
[2] SPI_USR_COMMAND_VALUE: command value is written into this field. For which part of this field is used, see the table above.
[3] SPI_WR_BIT_ORDER: 0: LSB first; 1: MSB first.

The configuration of address value is as follows:

Table 29.5-9. Sending Sequence of Address Value

ADDR_BITLEN[1]	ADDR_VALUE[2]	BIT_ORDER[3]	Sending Sequence of Address Value
0 - 7	[31:24]	1	ADDR_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] is sent.
		0	ADDR_VALUE[31:24] is sent first, and then ADDR_VALUE[23:31 - ADDR_BITLEN] is sent.
16 - 23	[31:8]	1	ADDR_VALUE[31:16] is sent first, and then ADDR_VALUE[ADDR_BITLEN - 8:8] is sent.
		0	ADDR_VALUE[31:16] is sent first, and then ADDR_VALUE[15:31 - ADDR_BITLEN] is sent.
24 - 31	[31:0]	1	ADDR_VALUE[31:8] is sent first, and then ADDR_VALUE[ADDR_BITLEN - 24:0] is sent.
		0	ADDR_VALUE[31:8] is sent first, and then ADDR_VALUE[7:31 - ADDR_BITLEN] is sent.

[1] SPI_USR_ADDR_BITLEN: this field is used to configure the bit length of the address.

[2] SPI_USR_ADDR_VALUE: address value is written into this field. For which part of this field is used, see the table above.

[3] SPI_WR_BIT_ORDER: 0: LSB first; 1: MSB first.

29.5.9.3 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 29.5-6 illustrates the connection of GP-SPI2 with its slave in full-duplex communication.

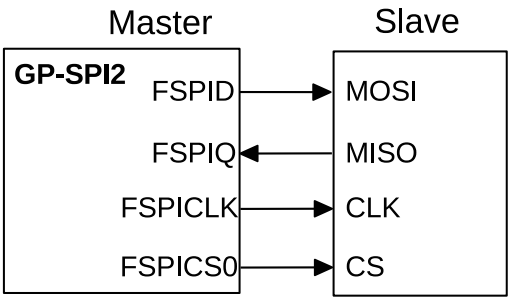


Figure 29.5-6. Full-Duplex Communication Between GP-SPI2 Master and a Slave

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 and APB clock (AHB_CLK and APB_CLK, see Chapter 7 *Reset and Clock*) and module clock (clk_spi_mst) for the GP-SPI2 module.
- Set SPI_DOUTDIN and clear SPI_SLAVE_MODE, to enable full-duplex communication as master.
- Configure GP-SPI2 registers listed in Table 29.5-7.
- Configure SPI CS setup time and hold time according to Section 29.6.
- Set the property of FSPICLK according to Section 29.7.
- Prepare data according to the selected transfer mode:
  - In CPU-controlled MOSI mode, prepare data in registers SPI_W0_REG ~ SPI_W15_REG.
  - In DMA-controlled mode,
    * configure SPI_DMA_TX_ENA/SPI_DMA_RX_ENA,
    * configure GDMA TX/RX link,
    * and start GDMA TX/RX engine, as described in Section 29.5.7 and Section 29.5.8.
- 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.

29.5.9.4 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 29.5.9.2, 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 29.5-7.

The detailed properties of half-duplex GP-SPI2 are as follows:

1. CMD: 0 ~ 16 bits, master output, slave input.
2. ADDR: 0 ~ 32 bits, master output, slave input.
3. DUMMY: 0 ~ 256 FSPICLK cycles, master output, slave input.
4. DOUT: 0 ~ 512 bits (64 B) in CPU-controlled mode and 0 ~ 256 Kbits (32 KB) in DMA-controlled mode, master output, slave input.
5. DIN: 0 ~ 512 bits (64 B) in CPU-controlled mode and 0 ~ 256 Kbits (32 KB) in DMA-controlled mode, master input, slave output.

Configuration

The register configuration 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 and APB clock (AHB_CLK and APB_CLK) and module clock (clk_spi_mst) for the GP-SPI2 module.
3. Clear SPI_DOUTDIN and SPI_SLAVE_MODE, to enable half-duplex communication as master.
4. Configure GP-SPI2 registers listed in Table 29.5-7.
5. Configure SPI CS setup time and hold time according to Section 29.6.
6. Set the property of FSPICLK according to Section 29.7.
7. Prepare data according to the selected transfer mode:
   - In CPU-controlled MOSI mode, prepare data in registers SPI_W0_REG ~ SPI_W15_REG.
   - In DMA-controlled mode,
     - configure SPI_DMA_TX_ENA/SPI_DMA_RX_ENA,
     - configure GDMA TX/RX link,
     - and start GDMA TX/RX engine, as described in Section 29.5.7 and Section 29.5.8.
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 mode.

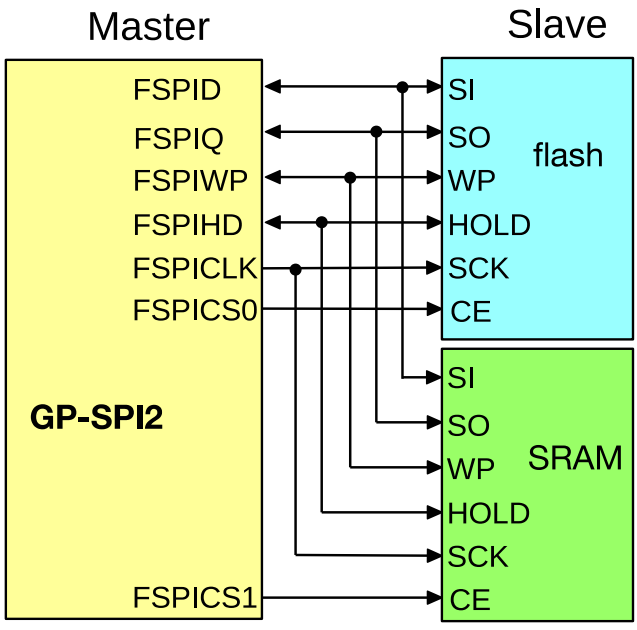


Figure 29.5-7. Connection of GP-SPI2 to Flash and External RAM in 4-bit Mode

Figure 29.5-8 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.

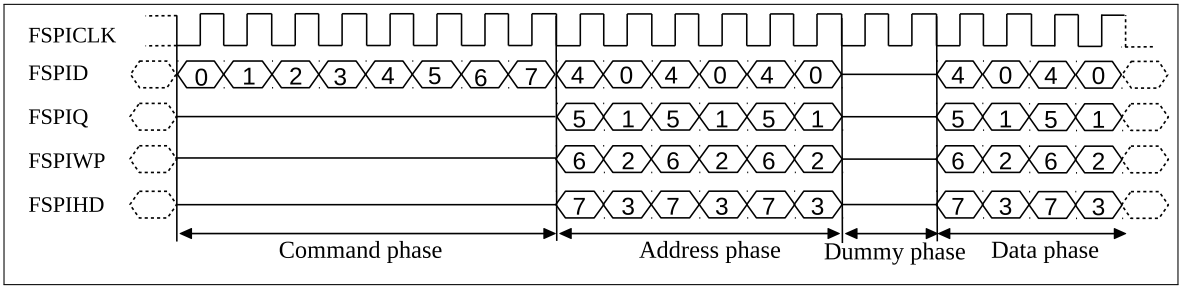


Figure 29.5-8. SPI Quad I/O Read Command Sequence Sent by GP-SPI2 to Flash

29.5.9.5 DMA-Controlled Configurable Segmented Transfer

Note:

Note that there is no separate section on how to configure a single transfer as master, since the CONF state of a configurable segmented transfer can be skipped to implement a single transfer.

Introduction

When GP-SPI2 works as a master, it provides a feature named configurable segmented transfer controlled by DMA.

A DMA-controlled transfer as master can be

- a single transfer, consisting of only one transaction;
- or a configurable segmented transfer, consisting of several transactions (segments).

In a configurable segmented transfer, the registers of each single transaction (segment) are configurable. This feature enables GP-SPI2 to do as many transactions (segments) as configured after such transfer is triggered once by the CPU. Figure 29.5-9 shows how this feature works.

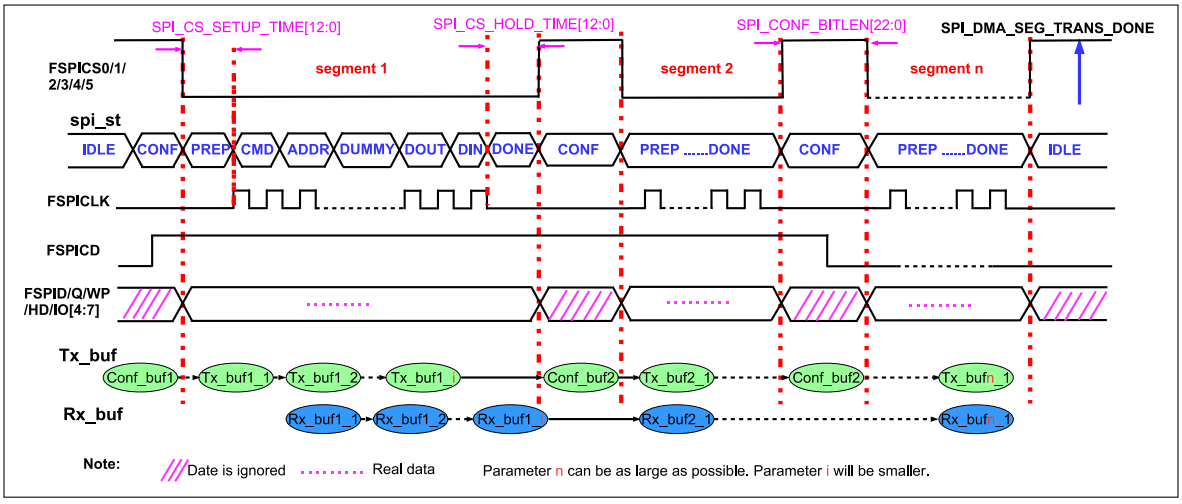


Figure 29.5-9. Configurable Segmented Transfer as Master

As shown in Figure 29.5-9, the registers for one transaction (segment *n*) can be reconfigured by GP-SPI2

hardware according to the content in its Conf_buf*n* during a CONF state, before this segment starts.

It's recommended to provide separate GDMA CONF links and CONF buffers (Conf_buf*i* in Figure 29.5-9) 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 29.5-9) into a chain. Hence, the behavior of the FSPI bus in each segment can be controlled independently.

For example, in a configurable segmentent 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 and APB clock (AHB_CLK and APB_CLK) and module clock (clk_spi_mst) for GP-SPI2 module.
3. Clear SPI_DOUTDIN and SPI_SLAVE_MODE, to enable half-duplex communication as master.
4. Configure GP-SPI2 registers listed in Table 29.5-7.
5. Configure SPI CS setup time and hold time according to Section 29.6.
6. Set the property of FSPICLK according to Section 29.7.
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 GDMA_OUTLINK_ADDR_CH*n* to the head address of the CONF and TX buffer descriptor linked list, and then set GDMA_OUTLINK_START_CH*n* to start the TX GDMA.
11. Clear the bit SPI_RX_EOF_EN in register SPI_DMA_CONF_REG. Point GDMA_INLINK_ADDR_CH*n* to the head address of the RX buffer descriptor linked list, and then set GDMA_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 29.9.
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.

Configuration of CONF Buffer and Magic Value

In a configurable segmented transfer, only registers which will 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 each 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 Bitmap (BM) Table, Table 29.5-10. 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 29.5-10. BM Table for CONF State

BM Bit	Register Name	BM Bit	Register Name
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 "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; the interrupt 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 29.5-11 and Table 29.5-12 provide an example to show how to configure a CONF buffer for a transaction (segment *i*) in which SPI_ADDR_REG, SPI_CTRL_REG, SPI_CLOCK_REG, SPI_USER_REG, SPI_USER1_REG need to be updated.

Table 29.5-11. 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 29.5-12, 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 29.5-12. BM Bit Value v.s. Register to Be Updated in This Example

BM Bit	Value	Register Name	BM Bit	Value	Register Name
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 29.6 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 156.25 ns to 8.1918 ms when $f_{\text{APB\_CLK}}$ is 32 MHz. (SPI_CONF_BITLEN + 5) will overflow from (0x40000 - SPI_CONF_BITLEN - 5) if SPI_CONF_BITLEN is larger than 0x3FFFA.

29.5.10 GP-SPI2 Works as a 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 transmission, and its falling/rising edges indicate the start/end of a single or segmented transmission. The length of transferred data must be in unit of bytes, otherwise the extra bits will be lost. The extra bits here means the result of total bits % 8.

29.5.10.1 Communication Formats

In GP-SPI2 as slave, SPI full-duplex and half-duplex communications are available. To select from the two communications, configure SPI_DOUTDIN in register SPI_USER_REG.

Full-duplex communication means that input data and output data are transmitted simultaneously throughout the entire transaction. All bits are treated as input or output data, which means no command, address or dummy states are expected. The interrupt SPI_TRANS_DONE_INT is triggered once the transaction ends.

In half-duplex communication, the format is CMD+ADDR+DUMMY+DATA (DIN or DOUT).

- "DIN" means that an SPI master reads data from GP-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 29.5-13 and Table 29.5-14 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 B in CPU-controlled mode and unlimited in DMA-controlled mode;
   - 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. The SPI_SLV_CMD_ERR_INT_RAW will be

set if the transferred CMD value is not supported by GP-SPI2 as slave. The SPI_SLV_CMD_ERR_INT_RAW can only be cleared by software.

29.5.10.2 Supported CMD Values 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 mode. 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 mode. 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 mode. Master sends data and GP-SPI2 receives data. The data is stored in GP-SPI2 GDMA RX buffer.
- 0x4 (Rd_DMA): DMA-controlled read mode. 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.

The detailed function of CMD7, CMD8, CMD9, and CMDA commands is reserved for user definition. These commands can be used as handshake signals, as passwords of some specific functions, as triggers of some user defined actions, and so on.

1/2/4-bit modes in states of CMD, ADDR, DATA are supported, which are determined by value of CMD[7:4]. The DUMMY state is always in 1-bit mode and lasts for eight SPI_CLK cycles. The definition of CMD[7:4] is as follows:

- 0x0: CMD, ADDR, and DATA states all are in 1-bit mode.
- 0x1: CMD and ADDR are in 1-bit mode. DATA is in 2-bit mode.
- 0x2: CMD and ADDR are in 1-bit mode. DATA is in 4-bit mode.
- 0x5: CMD is in 1-bit mode. ADDR and DATA are in 2-bit mode.
- 0xA: CMD is in 1-bit mode, ADDR and DATA are in 4-bit mode or in QPI mode.

In addition, if the value of CMD[7:0] is 0x05, 0xA5, 0x06, or 0xDD, DUMMY and DATA states are skipped. The definition of CMD[7:0] is as follows:

- 0x05 (End_SEG_TRANS): master sends 0x05 command to end slave segmented transfer in SPI mode.

- 0xA5 (End_SEG_TRANS): master sends 0xA5 command to end slave segmented transfer in QPI mode.
- 0x06 (En_QPI): GP-SPI2 enters QPI mode when receiving the 0x06 command and the bit SPI_QPI_MODE in register SPI_USER_REG is set.
- 0xDD (Ex_QPI): GP-SPI2 exits QPI mode when receiving the 0xDD command and the bit SPI_QPI_MODE is cleared.

All the CMD values supported by GP-SPI2 are listed in Table 29.5-13 and Table 29.5-14. Note that DUMMY state is always in 1-bit mode and lasts for eight SPI_CLK cycles.

Table 29.5-13. Supported CMD Values 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	-

Table 29.5-13. Supported CMD Values in SPI Mode

Transfer Type	CMD[7:0]	CMD State	ADDR State	DATA State
	0xA9	1-bit mode	4-bit mode	-
CMDA	0x0A	1-bit mode	1-bit mode	-
	0x1A	1-bit mode	1-bit mode	-
	0x2A	1-bit mode	1-bit mode	-
	0x5A	1-bit mode	2-bit mode	-
	0xAA	1-bit mode	4-bit mode	-
End_SEG_TRANS	0x05	1-bit mode	-	-
En_QPI	0x06	1-bit mode	-	-

Table 29.5-14. Supported CMD Values 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. The transfer types supported by GP-SPI2 in the QPI mode are listed in Table 29.5-14, which 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 29.5-13 and Table 29.5-14 are ignored. If the transferred data is not in unit of byte, GP-SPI2 will send or receive the data in unit of byte, but the extra bits (the result of total bits mod 8) will be lost. But if the CS low time is longer than 2 APB clock (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 29.9.

29.5.10.3 Slave Single Transfer and Slave Segmented Transfer

When GP-SPI2 works as a slave, it supports full-duplex and half-duplex communications controlled by DMA and by CPU. DMA-controlled transfer can be a single transfer, or a slave segmented transfer consisting of several transactions (segments). The CPU-controlled transfer can only be one single transfer, since each CPU-controlled transaction needs to be triggered by CPU.

In a slave segmented transfer, all transfer types listed in Table 29.5-13 and Table 29.5-14 are supported in a single transaction (segment). It means that CPU-controlled transaction and DMA-controlled transaction can be mixed in one slave segmented transfer.

It is recommended that in a slave segmented transfer:

- CPU-controlled transaction is used for handshake communication and short data transfers.
- DMA-controlled transaction is used for large data transfers.

29.5.10.4 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 and APB clock (AHB_CLK and APB_CLK).
3. Set the bit 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 mode 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 mode 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 29.5.7 and Section 29.5.8.
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 mode.
8. Set SPI_TRANS_DONE_INT_ENA in 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 GDMA RX buffer is used, which means that data has been stored in the related memory. Other interrupts described in Section 29.9 are optional.

29.5.10.5 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 and APB clock (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 29.5.7 and Section 29.5.8.
8. Set SPI_DMA_SLV_SEG_TRANS_EN in SPI_DMA_CONF_REG to enable slave segmented transfer.
9. Set SPI_DMA_SEG_TRANS_DONE_INT_ENA in 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 29.9 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.

29.5.10.6 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 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 and APB clock (AHB_CLK and APB_CLK).
3. Set SPI_SLAVE_MODE and SPI_DOUTDIN, to enable full-duplex communication as slave.
4. Set SPI_DMA_AFIFO_RST, SPI_BUF_AFIFO_RST, and SPI_RX_AFIFO_RST, to reset these buffers.
5. Set SPI_DMA_TX_ENA/SPI_DMA_RX_ENA. Configure GDMA TX/RX link and start GDMA TX/RX engine, as shown in Section 29.5.7 and Section 29.5.8.
6. Set the bit 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 byte length of the received DMA data.
7. Set SPI_DMA_SLV_SEG_TRANS_EN in SPI_DMA_CONF_REG to enable slave segmented transfer mode.
8. Set GDMA_IN_SUC_EOF_CH*n*_INT_ENA and wait for the interrupt GDMA_IN_SUC_EOF_CH*n*_INT.

29.6 CS Setup Time and Hold Time Control

SPI bus CS (SPI_CS) setup time and hold time are very important to meet the timing requirements of various SPI devices (e.g., flash or PSRAM).

CS setup time is the time between the CS falling edge and the first latch edge of SPI bus CLK (SPI_CLK). The first latch edge for mode 0 and mode 3 is rising edge, and falling edge for mode 1 and mode 2.

CS hold time is the time between the last latch edge of SPI_CLK and the CS rising edge.

When operating as slave, the CS setup time and hold time should be longer than 0.5 x T_SPI_CLK, otherwise the SPI transfer may be incorrect. T_SPI_CLK is one cycle of SPI_CLK.

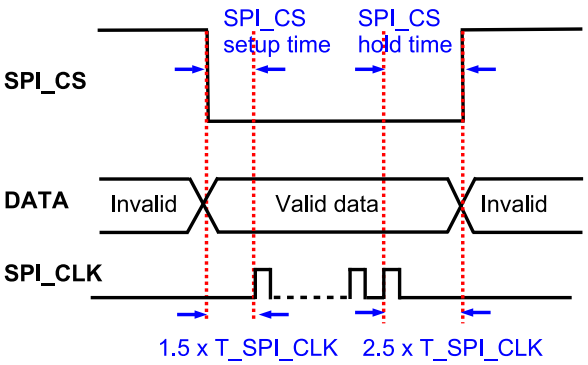
When operating as master, set the CS setup time by specifying SPI_CS_SETUP in SPI_USER_REG and SPI_CS_SETUP_TIME in SPI_USER1_REG:

- 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 SPI_USER_REG and SPI_CS_HOLD_TIME in 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.

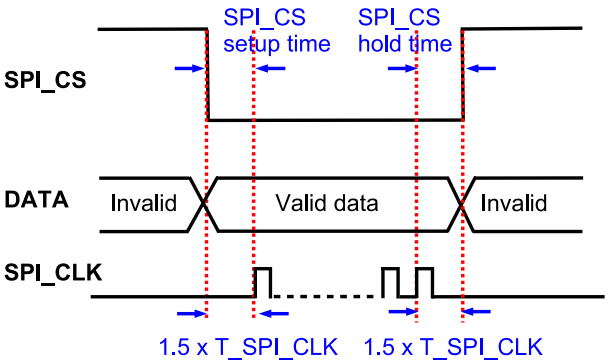
Figure 29.6-1 and Figure 29.6-2 show the recommended CS timing and register configuration to access external RAM and flash.



Register Configurations:

SPI_CS_SETUP = 1; SPI_CS_SETUP_TIME = 0;
SPI_CS_HOLD = 1; SPI_CS_HOLD_TIME = 1.

Figure 29.6-1. 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 29.6-2. Recommended CS Timing and Settings When Accessing Flash

29.7 GP-SPI2 Clock Control

GP-SPI2 has the following clocks:

- clk_spi_mst: module clock of GP-SPI2, derived from PLL_CLK. Used in GP-SPI2 as master to generate SPI_CLK signal for data transfer and for slaves.
- SPI_CLK: output clock as master.
- APB_CLK: clock for register configuration.

clk_spi_mst is enabled by PCR_SPI2_MST_CLK_ACTIVE_I and its clock source is controlled by PCR_SPI2_MST_CLK_SEL_I[1:0]:

- 0: XTAL_CLK
- 1: PLL_F48M_CLK
- 2: RC_FAST_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 to 1, the output clock frequency of GP-SPI2 will be $f_{clk\_spi\_mst}$. For other integral clock divisions, SPI_CLK_EQU_SYSCLK should be set to 0.

When operating as slave, the input clock frequency supported by GP-SPI2 is $f_{clk\_spi\_slv}$, and

- If f_{AHB_CLK} >= 40 MHz, $f_{clk\_spi\_slv}$ <= 40 MHz;
- If f_{AHB_CLK} < 40 MHz, $f_{clk\_spi\_slv}$ <= f_{AHB_CLK}.

29.7.1 Clock Phase and Polarity

SPI protocol has four clock modes, i.e., modes 0 ~ 3. See Figure 29.7-1 and Figure 29.7-2 (excerpted from SPI protocol).

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-SPI.

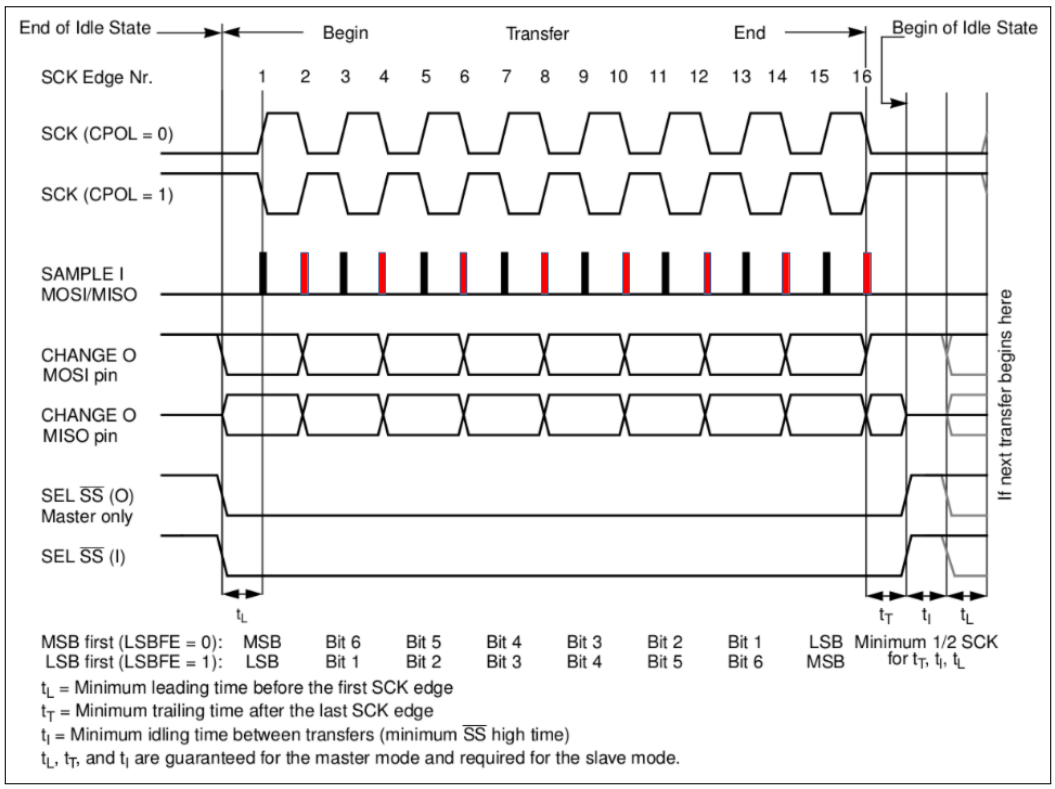


Figure 29.7-1. SPI Clock Mode 0 or 2

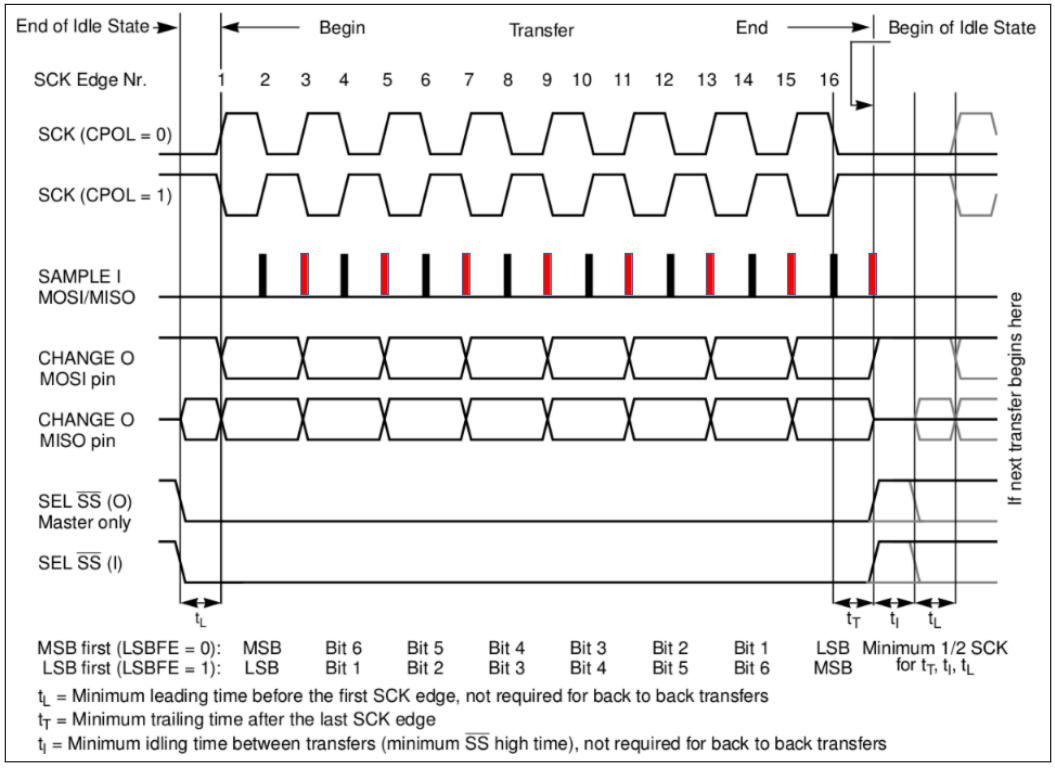


Figure 29.7-2. SPI Clock Mode 1 or 3

1. Mode 0: CPOL = 0, CPHA = 0; SCK is 0 when the SPI is in idle state; data is changed on the negative edge of SCK and sampled on the positive edge. The first data is shifted out before the first negative 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 positive edge of SCK and sampled on the negative edge.
3. Mode 2: CPOL = 1, CPHA = 0; SCK is 1 when the SPI is in idle state; data is changed on the positive edge of SCK and sampled on the negative edge. The first data is shifted out before the first positive 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 negative edge of SCK and sampled on the positive edge.

29.7.2 Clock Control as Master

The four clock modes 0 ~ 3 are supported in GP-SPI2 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 29.7-1, and can be changed according to the path delay in the application.

Table 29.7-1. 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.

29.7.3 Clock Control 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 29.7-2:

Table 29.7-2. 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

29.8 GP-SPI2 Timing Compensation

Overview Based on the role of GP-SPI in a user-defined SPI transmission system (whether GP-SPI is on the master or slave side), GP-SPI provides the following timing compensation schemes:

- In master mode, the default compensation scheme delays SPI_CLK by half a cycle. Refer to Figure 29.7-1 and Figure 29.7-2. You can also switch to standard sampling via register configuration.
- In slave mode, GP-SPI follows the standard SPI protocol for data transmission by default. You can also advance data transmission by half a cycle via register configuration.

when the GP-SPI is used as the 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.

When GP-SPI2 is used as the slave, if SPI_RSCK_DATA_OUT in the register SPI_SLAVE_REG is set to 1, output data is transmitted at the latch edge, which advances the transmission by half an SPI clock cycle. This feature can be used for slave timing compensation.

29.9 Interrupts

Interrupt Summary

GP-SPI2 provides an SPI interface interrupt SPI_INTR. When an SPI transfer ends, an interrupt is generated in GP-SPI2.

- SPI_DMA_INFIFO_FULL_ERR_INT: triggered when the length of GDMA RX FIFO is shorter than that of actual data transferred.
- SPI_DMA_OUTFIFO_EMPTY_ERR_INT: triggered when the length of GDMA TX FIFO is shorter than that of actual data transferred.
- SPI_SLV_EX_QPI_INT: triggered when Ex_QPI is received correctly in GP-SPI2 as slave and the SPI transfer ends.
- SPI_SLV_EN_QPI_INT: triggered when En_QPI is received correctly in GP-SPI2 as slave and the SPI transfer ends.
- SPI_SLV_CMD7_INT: triggered when CMD7 is received correctly in GP-SPI2 as slave and the SPI transfer ends.
- SPI_SLV_CMD8_INT: triggered when CMD8 is received correctly in GP-SPI2 as slave and the SPI transfer ends.
- SPI_SLV_CMD9_INT: triggered when CMD9 is received correctly in GP-SPI2 as slave and the SPI transfer ends.
- SPI_SLV_CMDA_INT: triggered when CMDA is received correctly in GP-SPI2 as slave and the SPI transfer ends.
- SPI_SLV_RD_DMA_DONE_INT: triggered at the end of Rd_DMA transfer as slave.
- SPI_SLV_WR_DMA_DONE_INT: triggered at the end of Wr_DMA transfer as slave.
- SPI_SLV_RD_BUF_DONE_INT: triggered at the end of Rd_BUF transfer as slave.
- SPI_SLV_WR_BUF_DONE_INT: triggered at the end of Wr_BUF transfer as slave.
- SPI_TRANS_DONE_INT: triggered at the end of SPI bus transfer in both as master and as slave.

- SPI_DMA_SEG_TRANS_DONE_INT: triggered at the end of End_SEG_TRANS transfer in GP-SPI2 slave segmented transfer mode or at the end of configurable segmented transfer as master.
- SPI_SEG_MAGIC_ERR_INT: triggered when a Magic error occurs in CONF buffer during configurable segmented transfer as master.
- SPI_MST_RX_AFIFO_WFULL_ERR_INT: triggered by RX AFIFO write-full error in GP-SPI2 as master.
- SPI_MST_TX_AFIFO_REMPTY_ERR_INT: triggered by TX AFIFO read-empty error in GP-SPI2 as master.
- SPI_SLV_CMD_ERR_INT: triggered when a received command value is not supported in GP-SPI2 as slave.
- SPI_APP2_INT: used and triggered by software. Only used for user defined function.
- SPI_APP1_INT: used and triggered by software. Only used for user defined function.

Interrupts Used as Master and Slave

Table 29.9-1 and Table 29.9-2 show the interrupts used in GP-SPI2 as master and as slave, respectively. Set the interrupt enable bit SPI_*_INT_ENA in SPI_DMA_INT_ENA_REG and wait for the SPI_INTR interrupt. When the transfer ends, the related interrupt is triggered and should be cleared by software before the next transfer.

Table 29.9-1. GP-SPI2 Interrupts as Master

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	DMA	SPI_TRANS_DONE_INT
		CPU	SPI_TRANS_DONE_INT
	Half-duplex MISO	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	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 29.9-2. GP-SPI2 Interrupts as Slave

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	DMA (Wr_DMA)	GDMA_IN_SUC_EOF_CH*n*_INT[3]
		CPU (Wr_BUF)	SPI_TRANS_DONE_INT[4]
	Half-duplex MISO	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	DMA (Wr_DMA)	SPI_DMA_SEG_TRANS_DONE_INT[9]
		CPU (Wr_BUF)	Not supported[10]
	Half-duplex MISO	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.

29.10 Register Summary

The addresses in this section are relative to SPI base address provided in Table 4.3-2 in Chapter 4 *System and Memory*.

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

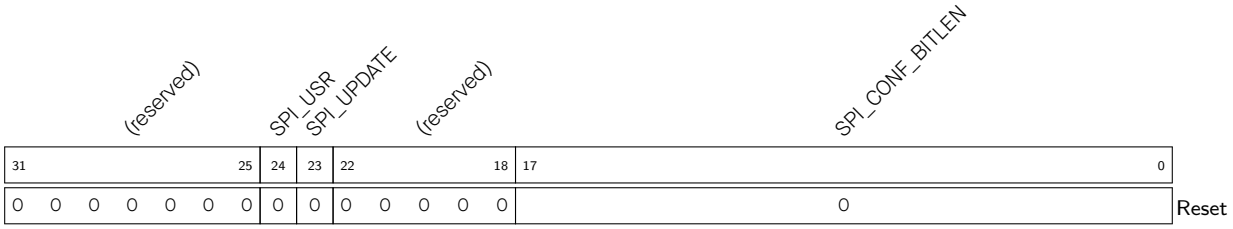
Name	Description	Address	Access
User-defined control registers			
SPI_CMD_REG	Command control register	0x0000	varies
SPI_ADDR_REG	Address value register	0x0004	R/W
SPI_USER_REG	SPI USER control register	0x0010	varies
SPI_USER1_REG	SPI USER control register 1	0x0014	R/W
SPI_USER2_REG	SPI USER control register 2	0x0018	R/W
Control and configuration registers			
SPI_CTRL_REG	SPI control register	0x0008	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 buffer0	0x0098	R/W/SS
SPI_W1_REG	SPI CPU-controlled buffer1	0x009C	R/W/SS
SPI_W2_REG	SPI CPU-controlled buffer2	0x00A0	R/W/SS
SPI_W3_REG	SPI CPU-controlled buffer3	0x00A4	R/W/SS
SPI_W4_REG	SPI CPU-controlled buffer4	0x00A8	R/W/SS
SPI_W5_REG	SPI CPU-controlled buffer5	0x00AC	R/W/SS
SPI_W6_REG	SPI CPU-controlled buffer6	0x00B0	R/W/SS
SPI_W7_REG	SPI CPU-controlled buffer7	0x00B4	R/W/SS
SPI_W8_REG	SPI CPU-controlled buffer8	0x00B8	R/W/SS
SPI_W9_REG	SPI CPU-controlled buffer9	0x00BC	R/W/SS

Name	Description	Address	Access
SPI_W10_REG	SPI CPU-controlled buffer10	0x00C0	R/W/SS
SPI_W11_REG	SPI CPU-controlled buffer11	0x00C4	R/W/SS
SPI_W12_REG	SPI CPU-controlled buffer12	0x00C8	R/W/SS
SPI_W13_REG	SPI CPU-controlled buffer13	0x00CC	R/W/SS
SPI_W14_REG	SPI CPU-controlled buffer14	0x00D0	R/W/SS
SPI_W15_REG	SPI CPU-controlled buffer15	0x00D4	R/W/SS
Version register			
SPI_DATE_REG	Version control	0x00F0	R/W

29.11 Registers

The addresses in this section are relative to SPI base address provided in Table 4.3-2 in Chapter 4 *System and Memory*.

Register 29.1. 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)

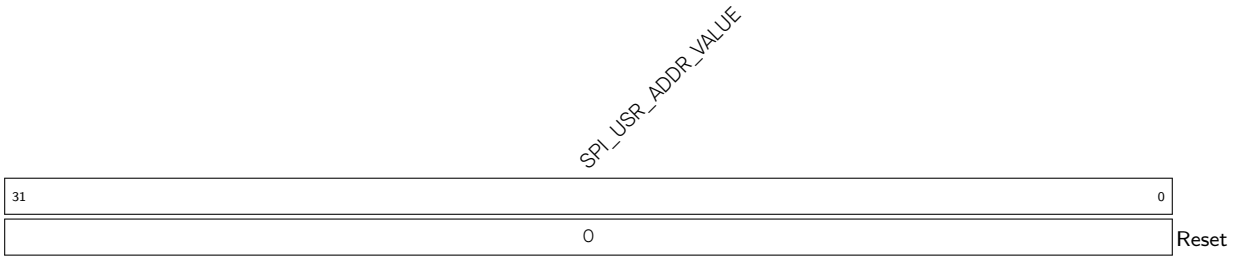
SPI_USR Configures whether or not to enable user-defined command.

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)

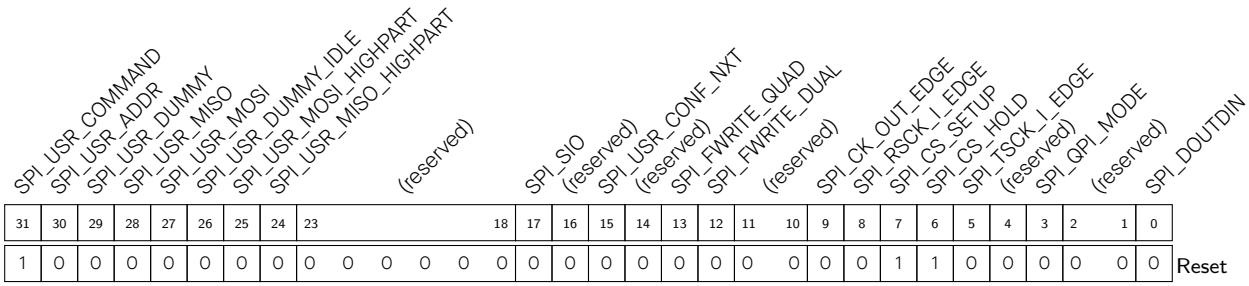
Register 29.2. SPI_ADDR_REG (0x0004)



SPI_USR_ADDR_VALUE Configures the address to slave.

Can be configured in CONF state. (R/W)

Register 29.3. 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_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)

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 29.7.2. (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)

Continued on the next page...

Register 29.3. SPI_USER_REG (0x0010)

Continued from the previous page...

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_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)

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)

Continued on the next page...

Register 29.3. SPI_USER_REG (0x0010)

Continued from the previous page...

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)

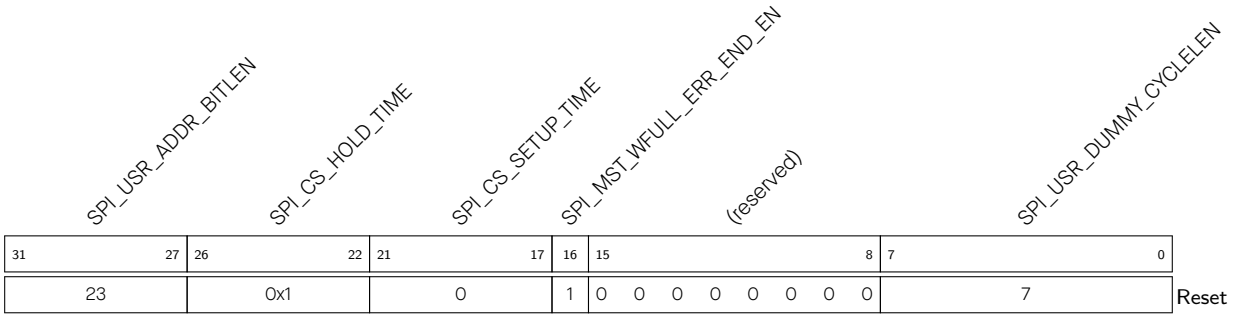
SPI_USR_COMMAND Configures whether or not to enable the command (CMD) state of an operation.

0: Disable

1: Enable

Can be configured in CONF state. (R/W)

Register 29.4. 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 (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.

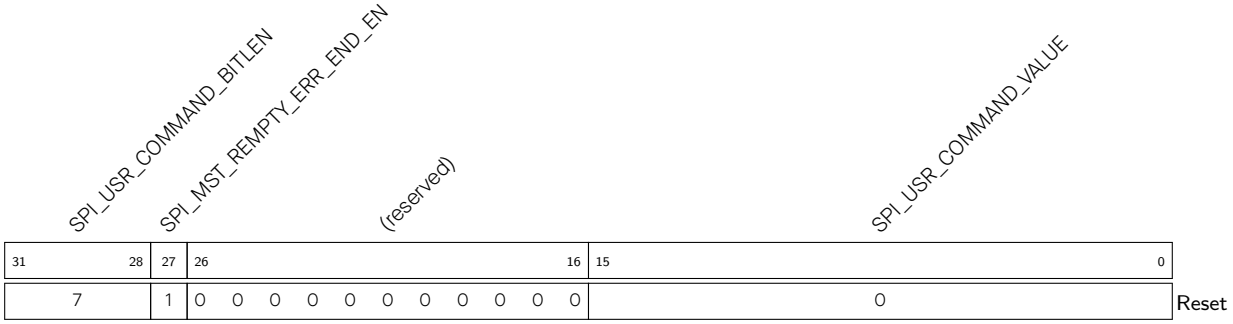
Measurement unit: SPI_CLK clock cycles.

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 29.5. 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

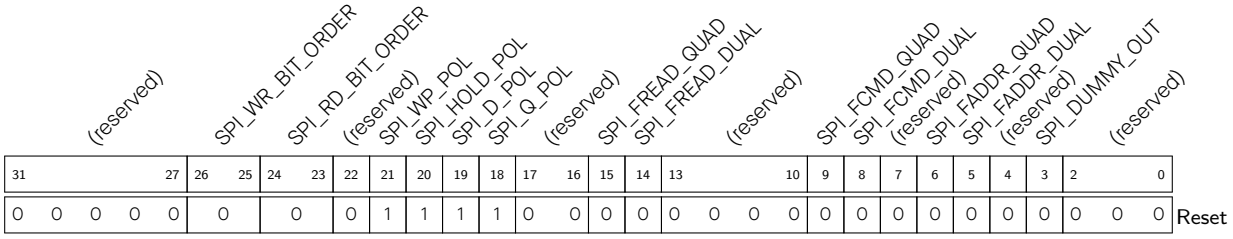
1: End

(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 29.6. 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_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)

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_Q_POL Configures MISO line polarity.

0: Low

1: High

Can be configured in CONF state. (R/W)

Continued on the next page...

Register 29.6. SPI_CTRL_REG (0x0008)

Continued from the previous page...

SPI_D_POL Configures MOSI line polarity.

0: Low

1: High

Can be configured in CONF state. (R/W)

SPI_HOLD_POL Configures SPI_HOLD output value when SPI is in idle.

0: Output low

1: Output high

Can be configured in CONF state. (R/W)

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)

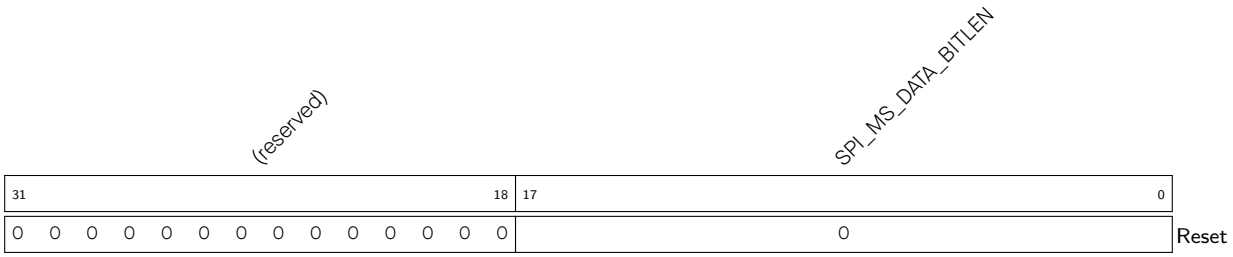
SPI_WR_BIT_ORDER Configures the bit order in command (CMD), address (ADDR), and write-data (MOSI) states.

0: MSB first

1: LSB first

Can be configured in CONF state. (R/W)

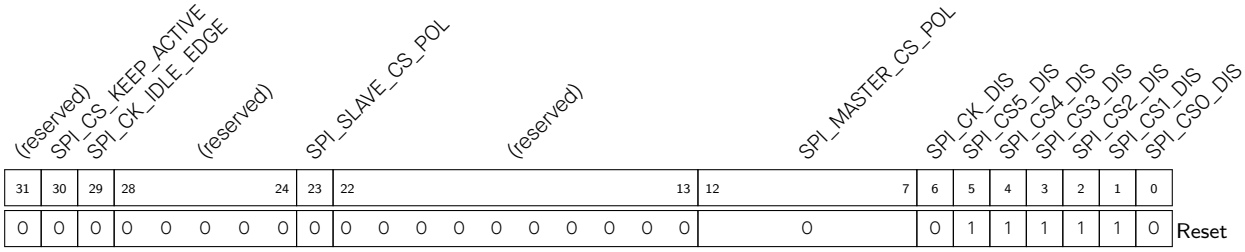
Register 29.7. SPI_MS_DLEN_REG (0x001C)



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 29.8. SPI_MISC_REG (0x0020)



SPI_CS*n*_DIS (*n* = 0-5) Configures whether or not to disable SPI_CS*n* pin.

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 or not to disable SPI_CLK output.

0: Enable

1: Disable

Can be configured in CONF state. (R/W)

SPI_MASTER_CS_POL Configures the polarity of SPI_CS*n* (*n* = 0-5) line in master transfer.

0: SPI_CS*n* is low active.

1: SPI_CS*n* is high active.

Can be configured in CONF state. (R/W)

SPI_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_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)

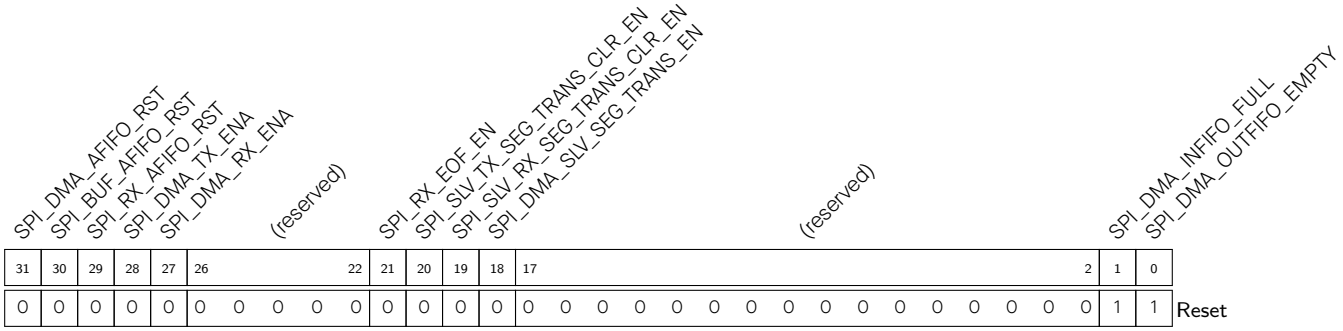
SPI_CS_KEEP_ACTIVE Configures whether or not to keep the SPI_CS line low.

0: Not keep low

1: Keep low

Can be configured in CONF state. (R/W)

Register 29.9. 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 the following Wr_DMA transactions will be received.
- if the size of DMA RX buffer is 0, the data in the 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 29.9. SPI_DMA_CONF_REG (0x0030)

Continued from the previous page...

SPI_RX_EOF_EN Configure the interrupt type of SPI DMA reporting eof

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.

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.

(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 29.5-3 and in Figure 29.5-4.

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 29.5-3 and in Figure 29.5-4.

0: Not reset

1: Reset

buf_tx_afifo is used to send data out in CPU-controlled master and slave transfer. (WT)

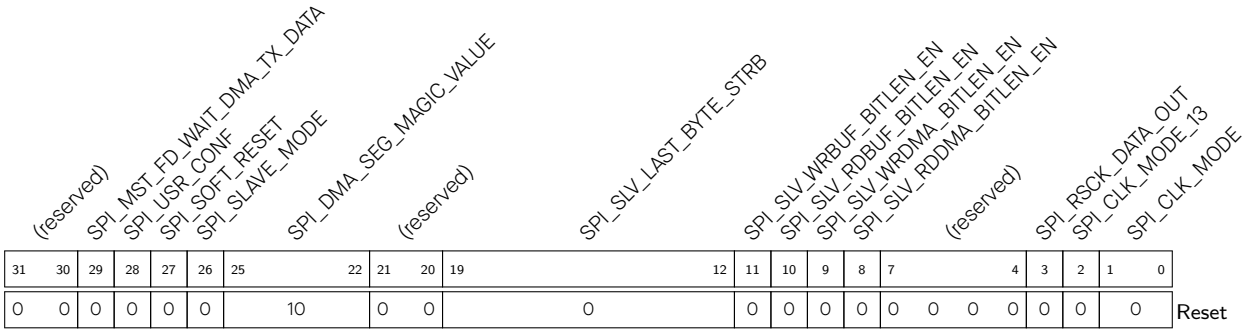
SPI_DMA_AFIFO_RST Configures whether or not to reset dma_tx_afifo as shown in Figure 29.5-3 and in Figure 29.5-4.

0: Not reset

1: Reset

dma_tx_afifo is used to send data out in DMA-controlled slave transfer. (WT)

Register 29.10. 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 29.7-2.

1: Support SPI clock mode 1 or 3. See Table 29.7-2.

(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 29.10. SPI_SLAVE_REG (0x00E0)

Continued from the previous page...

SPI_SLV_LAST_BYTE_STRB indicates the valid bits of the last received data byte in SPI slave FD and HD modes. (R/SS)

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_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 the 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)

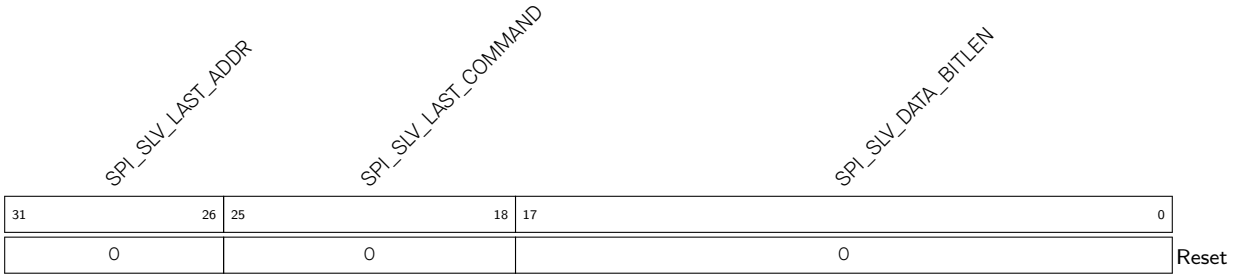
SPI_MST_FD_WAIT_DMA_TX_DATA Configures whether or not to wait DMA TX data gets ready before starting SPI transfer in master full-duplex transfer.

0: Not wait

1: Wait

(R/W)

Register 29.11. SPI_SLAVE1_REG (0x00E4)

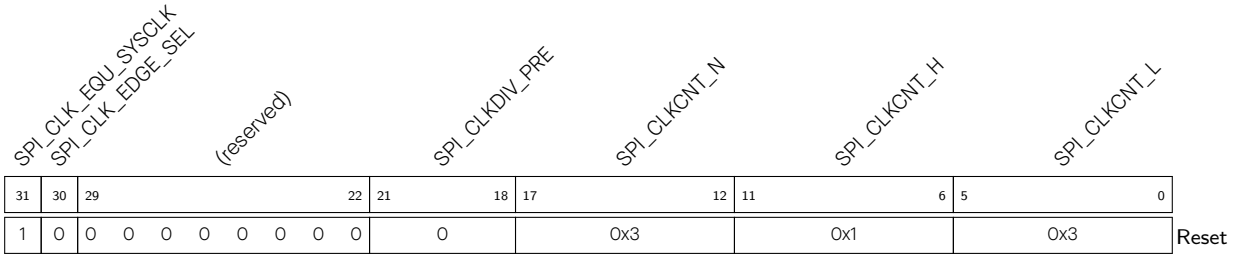


SPI_SLV_DATA_BITLEN Configures the transferred data bit length in SPI slave full-/half-duplex modes. (R/W/SS)

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 29.12. 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)

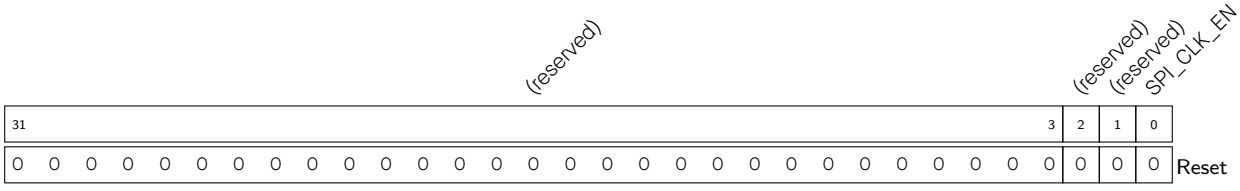
SPI_CLKCNT_N Configures the divider of SPI_CLK in master transfer.
SPI_CLK frequency is $f_{clk\_spi\_mst}$/(SPI_CLKDIV_PRE + 1)/(SPI_CLKCNT_N + 1).
Can be configured in CONF state. (R/W)

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 whether to use the standard clock sampling edge or delay the sampling edge by half a cycle during master transfer.
0□The clock sampling edge is delayed by half a cycle.
1□The clock sampling edge is standard.
Can be configured in CONF state. (R/W)

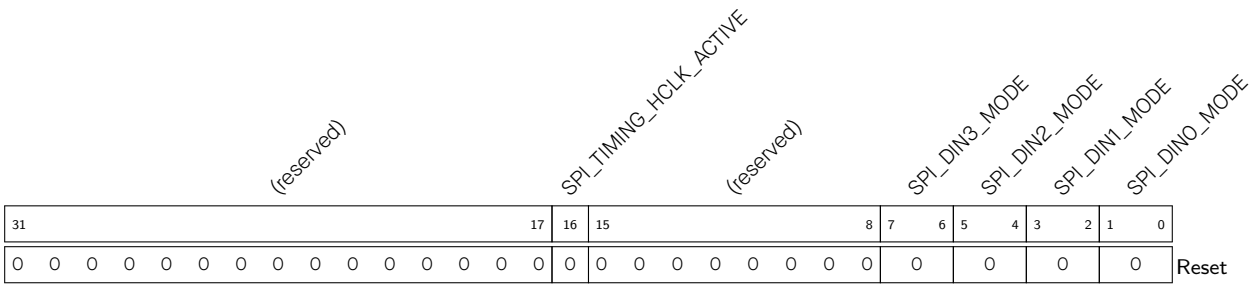
SPI_CLK_EQU_SYSCLK Configures whether or not the SPI_CLK is equal to APB_CLK in master transfer.
0: SPI_CLK is divided from APB_CLK.
1: SPI_CLK is equal to APB_CLK.
Can be configured in CONF state. (R/W)

Register 29.13. SPI_CLK_GATE_REG (0x00E8)



SPI_CLK_EN Configures whether or not to enable clock gate.
0: Disable
1: Enable
(R/W)

Register 29.14. 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 29.14. SPI_DIN_MODE_REG (0x0024)

Continued from the previous page...

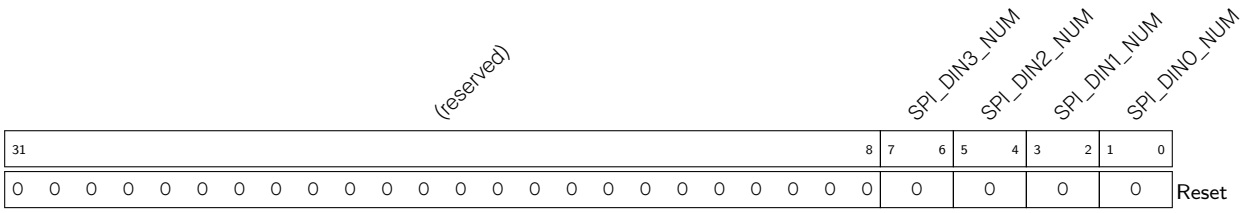
SPI_TIMING_HCLK_ACTIVE Configures whether or not to enable HCLK (high-frequency clock) in SPI input timing module.

0: Disable

1: Enable

Can be configured in CONF state. (R/W)

Register 29.15. 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

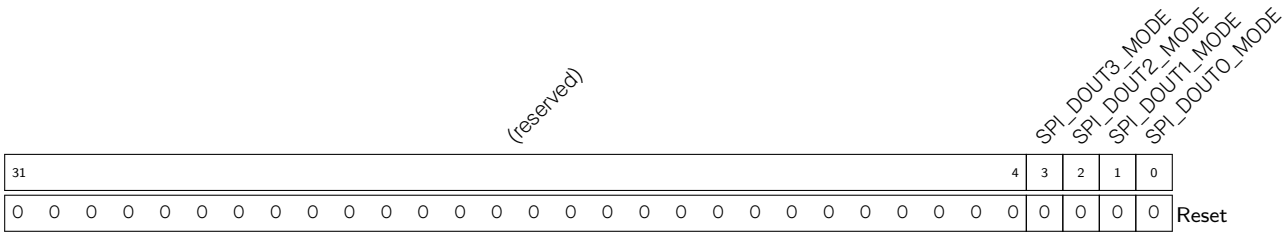
1: Delayed by 2 clock cycles

2: Delayed by 3 clock cycles

3: Delayed by 4 clock cycles

Can be configured in CONF state. (R/W)

Register 29.16. 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)

Register 29.17. SPI_DMA_INT_ENA_REG (0x0034)

SPI_DMA_INFIFO_FULL_ERR_INT_ENA Write 1 to enable SPI_DMA_INFIFO_FULL_ERR_INT interrupt. (R/W)

SPI_DMA_OUTFIFO_EMPTY_ERR_INT_ENA Write 1 to enable SPI_DMA_OUTFIFO_EMPTY_ERR_INT interrupt. (R/W)

SPI_SLV_EX_QPI_INT_ENA Write 1 to enable SPI_SLV_EX_QPI_INT interrupt. (R/W)

SPI_SLV_EN_QPI_INT_ENA Write 1 to enable SPI_SLV_EN_QPI_INT interrupt. (R/W)

SPI_SLV_CMD7_INT_ENA Write 1 to enable SPI_SLV_CMD7_INT interrupt. (R/W)

SPI_SLV_CMD8_INT_ENA Write 1 to enable SPI_SLV_CMD8_INT interrupt. (R/W)

SPI_SLV_CMD9_INT_ENA Write 1 to enable SPI_SLV_CMD9_INT interrupt. (R/W)

SPI_SLV_CMDA_INT_ENA Write 1 to enable SPI_SLV_CMDA_INT interrupt. (R/W)

SPI_SLV_RD_DMA_DONE_INT_ENA Write 1 to enable SPI_SLV_RD_DMA_DONE_INT interrupt. (R/W)

SPI_SLV_WR_DMA_DONE_INT_ENA Write 1 to enable SPI_SLV_WR_DMA_DONE_INT interrupt. (R/W)

SPI_SLV_RD_BUF_DONE_INT_ENA Write 1 to enable SPI_SLV_RD_BUF_DONE_INT interrupt. (R/W)

SPI_SLV_WR_BUF_DONE_INT_ENA Write 1 to enable SPI_SLV_WR_BUF_DONE_INT interrupt. (R/W)

SPI_TRANS_DONE_INT_ENA Write 1 to enable SPI_TRANS_DONE_INT interrupt. (R/W)

SPI_DMA_SEG_TRANS_DONE_INT_ENA Write 1 to enable SPI_DMA_SEG_TRANS_DONE_INT interrupt. (R/W)

SPI_SEG_MAGIC_ERR_INT_ENA Write 1 to enable SPI_SEG_MAGIC_ERR_INT interrupt. (R/W)

SPI_SLV_CMD_ERR_INT_ENA Write 1 to enable SPI_SLV_CMD_ERR_INT interrupt. (R/W)

SPI_MST_RX_AFIFO_WFULL_ERR_INT_ENA Write 1 to enable SPI_MST_RX_AFIFO_WFULL_ERR_INT interrupt. (R/W)

Continued on the next page...

Register 29.17. SPI_DMA_INT_ENA_REG (0x0034)

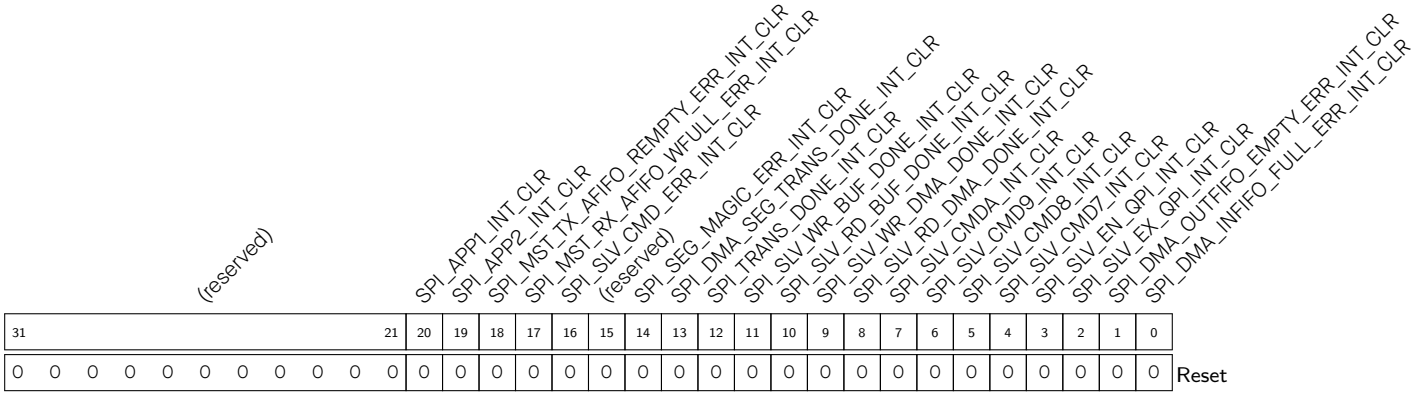
Continued from the previous page...

SPI_MST_TX_AFIFO_REMPTY_ERR_INT_ENA Write 1 to enable SPI_MST_TX_AFIFO_REMPTY_ERR_INT interrupt. (R/W)

SPI_APP2_INT_ENA Write 1 to enable SPI_APP2_INT interrupt. (R/W)

SPI_APP1_INT_ENA Write 1 to enable SPI_APP1_INT interrupt. (R/W)

Register 29.18. SPI_DMA_INT_CLR_REG (0x0038)



SPI_DMA_INFIFO_FULL_ERR_INT_CLR Write 1 to clear SPI_DMA_INFIFO_FULL_ERR_INT interrupt. (WT)

SPI_DMA_OUTFIFO_EMPTY_ERR_INT_CLR Write 1 to clear SPI_DMA_OUTFIFO_EMPTY_ERR_INT interrupt. (WT)

SPI_SLV_EX_QPI_INT_CLR Write 1 to clear SPI_SLV_EX_QPI_INT interrupt. (WT)

SPI_SLV_EN_QPI_INT_CLR Write 1 to clear SPI_SLV_EN_QPI_INT interrupt. (WT)

SPI_SLV_CMD7_INT_CLR Write 1 to clear SPI_SLV_CMD7_INT interrupt. (WT)

SPI_SLV_CMD8_INT_CLR Write 1 to clear SPI_SLV_CMD8_INT interrupt. (WT)

SPI_SLV_CMD9_INT_CLR Write 1 to clear SPI_SLV_CMD9_INT interrupt. (WT)

SPI_SLV_CMDA_INT_CLR Write 1 to clear SPI_SLV_CMDA_INT interrupt. (WT)

SPI_SLV_RD_DMA_DONE_INT_CLR Write 1 to clear SPI_SLV_RD_DMA_DONE_INT interrupt. (WT)

SPI_SLV_WR_DMA_DONE_INT_CLR Write 1 to clear SPI_SLV_WR_DMA_DONE_INT interrupt. (WT)

SPI_SLV_RD_BUF_DONE_INT_CLR Write 1 to clear SPI_SLV_RD_BUF_DONE_INT interrupt. (WT)

SPI_SLV_WR_BUF_DONE_INT_CLR Write 1 to clear SPI_SLV_WR_BUF_DONE_INT interrupt. (WT)

SPI_TRANS_DONE_INT_CLR Write 1 to clear SPI_TRANS_DONE_INT interrupt. (WT)

SPI_DMA_SEG_TRANS_DONE_INT_CLR Write 1 to clear SPI_DMA_SEG_TRANS_DONE_INT interrupt. (WT)

SPI_SEG_MAGIC_ERR_INT_CLR Write 1 to clear SPI_SEG_MAGIC_ERR_INT interrupt. (WT)

SPI_SLV_CMD_ERR_INT_CLR Write 1 to clear SPI_SLV_CMD_ERR_INT interrupt. (WT)

SPI_MST_RX_AFIFO_WFULL_ERR_INT_CLR Write 1 to clear SPI_MST_RX_AFIFO_WFULL_ERR_INT interrupt. (WT)

Continued on the next page...

Register 29.18. SPI_DMA_INT_CLR_REG (0x0038)

Continued from the previous page...

SPI_MST_TX_AFIFO_REMPTY_ERR_INT_CLR Write 1 to clear SPI_MST_TX_AFIFO_REMPTY_ERR_INT interrupt. (WT)

SPI_APP2_INT_CLR Write 1 to clear SPI_APP2_INT interrupt. (WT)

SPI_APP1_INT_CLR Write 1 to clear SPI_APP1_INT interrupt. (WT)

Register 29.19. SPI_DMA_INT_RAW_REG (0x003C)

SPI_DMA_INFIFO_FULL_ERR_INT_RAW The raw interrupt status of SPI_DMA_INFIFO_FULL_ERR_INT interrupt. (R/WTC/SS)

SPI_DMA_OUTFIFO_EMPTY_ERR_INT_RAW The raw interrupt status of SPI_DMA_OUTFIFO_EMPTY_ERR_INT interrupt. (R/WTC/SS)

SPI_SLV_EX_QPI_INT_RAW The raw interrupt status of SPI_SLV_EX_QPI_INT interrupt. (R/WTC/SS)

SPI_SLV_EN_QPI_INT_RAW The raw interrupt status of SPI_SLV_EN_QPI_INT interrupt. (R/WTC/SS)

SPI_SLV_CMD7_INT_RAW The raw interrupt status of SPI_SLV_CMD7_INT interrupt. (R/WTC/SS)

SPI_SLV_CMD8_INT_RAW The raw interrupt status of SPI_SLV_CMD8_INT interrupt. (R/WTC/SS)

SPI_SLV_CMD9_INT_RAW The raw interrupt status of SPI_SLV_CMD9_INT interrupt. (R/WTC/SS)

SPI_SLV_CMDA_INT_RAW The raw interrupt status of SPI_SLV_CMDA_INT interrupt. (R/WTC/SS)

SPI_SLV_RD_DMA_DONE_INT_RAW The raw interrupt status of SPI_SLV_RD_DMA_DONE_INT interrupt. (R/WTC/SS)

SPI_SLV_WR_DMA_DONE_INT_RAW The raw interrupt status of SPI_SLV_WR_DMA_DONE_INT interrupt. (R/WTC/SS)

SPI_SLV_RD_BUF_DONE_INT_RAW The raw interrupt status of SPI_SLV_RD_BUF_DONE_INT interrupt. (R/WTC/SS)

SPI_SLV_WR_BUF_DONE_INT_RAW The raw interrupt status of SPI_SLV_WR_BUF_DONE_INT interrupt. (R/WTC/SS)

SPI_TRANS_DONE_INT_RAW The raw interrupt status of SPI_TRANS_DONE_INT interrupt. (R/WTC/SS)

Continued on the next page...

Register 29.19. 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 interrupt. (R/WTC/SS)

SPI_SEG_MAGIC_ERR_INT_RAW The raw interrupt status of SPI_SEG_MAGIC_ERR_INT interrupt. (R/WTC/SS)

SPI_SLV_CMD_ERR_INT_RAW The raw interrupt status of SPI_SLV_CMD_ERR_INT interrupt. (R/WTC/SS)

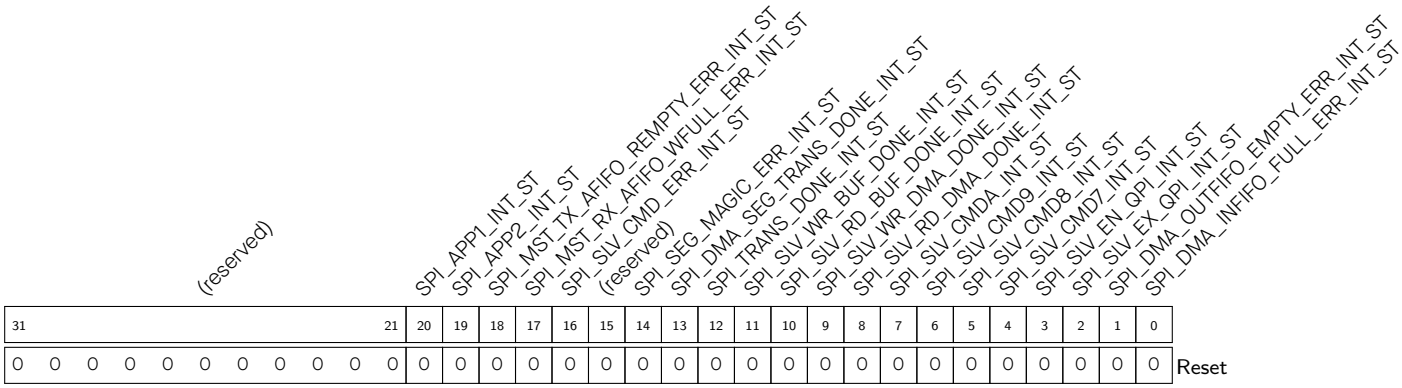
SPI_MST_RX_AFIFO_WFULL_ERR_INT_RAW The raw interrupt status of SPI_MST_RX_AFIFO_WFULL_ERR_INT interrupt. (R/WTC/SS)

SPI_MST_TX_AFIFO_REMPTY_ERR_INT_RAW The raw interrupt status of SPI_MST_TX_AFIFO_REMPTY_ERR_INT interrupt. (R/WTC/SS)

SPI_APP2_INT_RAW The raw interrupt status of SPI_APP2_INT interrupt. The value is only controlled by the application. (R/WTC)

SPI_APP1_INT_RAW The raw interrupt status of SPI_APP1_INT interrupt. The value is only controlled by the application. (R/WTC)

Register 29.20. SPI_DMA_INT_ST_REG (0x0040)



SPI_DMA_INFIFO_FULL_ERR_INT_ST The interrupt status of SPI_DMA_INFIFO_FULL_ERR_INT interrupt. (RO)

SPI_DMA_OUTFIFO_EMPTY_ERR_INT_ST The interrupt status of SPI_DMA_OUTFIFO_EMPTY_ERR_INT interrupt. (RO)

SPI_SLV_EX_QPI_INT_ST The interrupt status of SPI_SLV_EX_QPI_INT interrupt. (RO)

SPI_SLV_EN_QPI_INT_ST The interrupt status of SPI_SLV_EN_QPI_INT interrupt. (RO)

SPI_SLV_CMD7_INT_ST The interrupt status of SPI_SLV_CMD7_INT interrupt. (RO)

SPI_SLV_CMD8_INT_ST The interrupt status of SPI_SLV_CMD8_INT interrupt. (RO)

SPI_SLV_CMD9_INT_ST The interrupt status of SPI_SLV_CMD9_INT interrupt. (RO)

SPI_SLV_CMDA_INT_ST The interrupt status of SPI_SLV_CMDA_INT interrupt. (RO)

SPI_SLV_RD_DMA_DONE_INT_ST The interrupt status of SPI_SLV_RD_DMA_DONE_INT interrupt. (RO)

SPI_SLV_WR_DMA_DONE_INT_ST The interrupt status of SPI_SLV_WR_DMA_DONE_INT interrupt. (RO)

SPI_SLV_RD_BUF_DONE_INT_ST The interrupt status of SPI_SLV_RD_BUF_DONE_INT interrupt. (RO)

SPI_SLV_WR_BUF_DONE_INT_ST The interrupt status of SPI_SLV_WR_BUF_DONE_INT interrupt. (RO)

SPI_TRANS_DONE_INT_ST The interrupt status of SPI_TRANS_DONE_INT interrupt. (RO)

SPI_DMA_SEG_TRANS_DONE_INT_ST The interrupt status of SPI_DMA_SEG_TRANS_DONE_INT interrupt. (RO)

SPI_SEG_MAGIC_ERR_INT_ST The interrupt status of SPI_SEG_MAGIC_ERR_INT interrupt. (RO)

SPI_SLV_CMD_ERR_INT_ST The interrupt status of SPI_SLV_CMD_ERR_INT interrupt. (RO)

SPI_MST_RX_AFIFO_WFULL_ERR_INT_ST The interrupt status of SPI_MST_RX_AFIFO_WFULL_ERR_INT interrupt. (RO)

Continued on the next page...

Register 29.20. SPI_DMA_INT_ST_REG (0x0040)

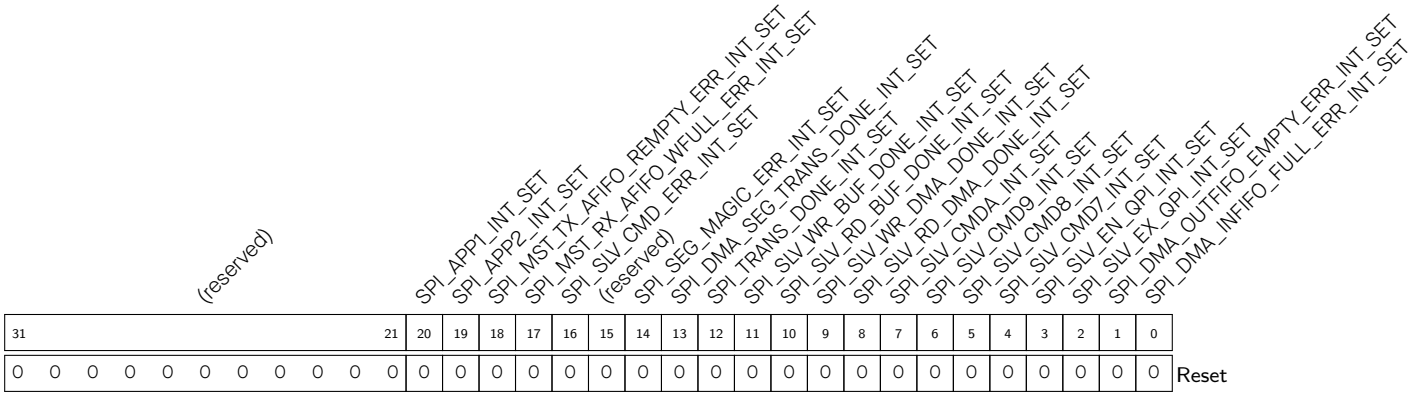
Continued from the previous page...

SPI_MST_TX_AFIFO_REMPTY_ERR_INT_ST The interrupt status of SPI_MST_TX_AFIFO_REMPTY_ERR_INT interrupt. (RO)

SPI_APP2_INT_ST The interrupt status of SPI_APP2_INT interrupt. (RO)

SPI_APP1_INT_ST The interrupt status of SPI_APP1_INT interrupt. (RO)

Register 29.21. SPI_DMA_INT_SET_REG (0x0044)



SPI_DMA_INFIFO_FULL_ERR_INT_SET Write 1 to set SPI_DMA_INFIFO_FULL_ERR_INT interrupt. (WT)

SPI_DMA_OUTFIFO_EMPTY_ERR_INT_SET Write 1 to set SPI_DMA_OUTFIFO_EMPTY_ERR_INT interrupt. (WT)

SPI_SLV_EX_QPI_INT_SET Write 1 to set SPI_SLV_EX_QPI_INT interrupt. (WT)

SPI_SLV_EN_QPI_INT_SET Write 1 to set SPI_SLV_EN_QPI_INT interrupt. (WT)

SPI_SLV_CMD7_INT_SET Write 1 to set SPI_SLV_CMD7_INT interrupt. (WT)

SPI_SLV_CMD8_INT_SET Write 1 to set SPI_SLV_CMD8_INT interrupt. (WT)

SPI_SLV_CMD9_INT_SET Write 1 to set SPI_SLV_CMD9_INT interrupt. (WT)

SPI_SLV_CMDA_INT_SET Write 1 to set SPI_SLV_CMDA_INT interrupt. (WT)

SPI_SLV_RD_DMA_DONE_INT_SET Write 1 to set SPI_SLV_RD_DMA_DONE_INT interrupt. (WT)

SPI_SLV_WR_DMA_DONE_INT_SET Write 1 to set SPI_SLV_WR_DMA_DONE_INT interrupt. (WT)

SPI_SLV_RD_BUF_DONE_INT_SET Write 1 to set SPI_SLV_RD_BUF_DONE_INT interrupt. (WT)

SPI_SLV_WR_BUF_DONE_INT_SET Write 1 to set SPI_SLV_WR_BUF_DONE_INT interrupt. (WT)

SPI_TRANS_DONE_INT_SET Write 1 to set SPI_TRANS_DONE_INT interrupt. (WT)

SPI_DMA_SEG_TRANS_DONE_INT_SET Write 1 to set SPI_DMA_SEG_TRANS_DONE_INT interrupt. (WT)

SPI_SEG_MAGIC_ERR_INT_SET Write 1 to set SPI_SEG_MAGIC_ERR_INT interrupt. (WT)

SPI_SLV_CMD_ERR_INT_SET Write 1 to set SPI_SLV_CMD_ERR_INT interrupt. (WT)

Continued on the next page...

Register 29.21. SPI_DMA_INT_SET_REG (0x0044)

Continued from the previous page...

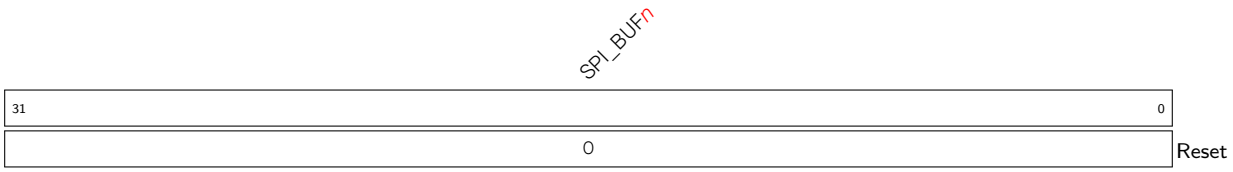
SPI_MST_RX_AFIFO_WFULL_ERR_INT_SET Write 1 to set SPI_MST_RX_AFIFO_WFULL_ERR_INT interrupt. (WT)

SPI_MST_TX_AFIFO_REMPTY_ERR_INT_SET Write 1 to set SPI_MST_TX_AFIFO_REMPTY_ERR_INT interrupt. (WT)

SPI_APP2_INT_SET Write 1 to set SPI_APP2_INT interrupt. (WT)

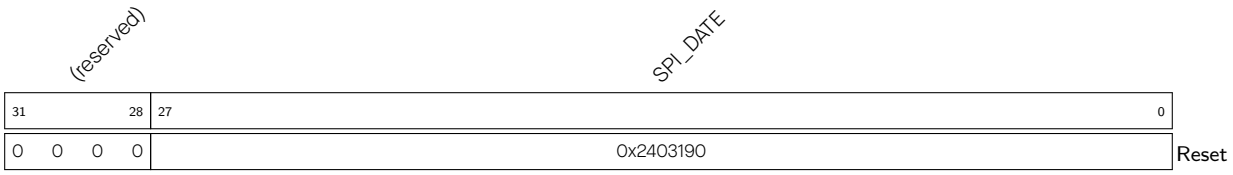
SPI_APP1_INT_SET Write 1 to set SPI_APP1_INT interrupt. (WT)

Register 29.22. SPI_W*n*_REG (*n*: 0-15) (0x0098 + 0x4**n*)



SPI_BUF*n* 32-bit data buffer *n*. (R/W/SS)

Register 29.23. SPI_DATE_REG (0x00F0)



SPI_DATE Version control register. (R/W)

Chapter 30

I2C Controller (I2C)

The I2C (Inter-Integrated Circuit) bus allows ESP32-H2 to communicate with multiple external devices. These external devices can share one I2C bus. ESP32-H2 has two I2C controllers in the main system, which can act as a master or a slave.

30.1 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. Therefore, multiple peripherals can be mounted on the I2C bus, usually with one or more masters and one or more slaves. However, only one master is allowed to occupy the bus to access one slave at the same time.

The master initiates communication by generating a START condition: pulling the SDA line low while SCL is high. Then it issues nine clock pulses via SCL. The first eight pulses are used to transmit a 7-bit address followed by a read/write ($R/\overline{W}$) bit. If the address of an I2C slave matches the 7-bit address transmitted, this matching slave can respond by pulling SDA low on the ninth clock pulse. Then, the master and the slave can send or receive data according to the $R/\overline{W}$ bit, and terminate the data transfer according to the logic level of the acknowledge (ACK) bit. During data transfer, SDA changes only when SCL is low. Once the communication has finished, the master sends a STOP condition: pulling SDA up while SCL is high. If a master both reads and writes data in one transfer, then it should send an RSTART condition, a slave address, and a $R/\overline{W}$ bit before switching between the operations. The RSTART condition is used to change the transfer direction and the mode of the devices (master mode or slave mode).

30.2 Features

The I2C controller of ESP32-H2 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 by pulling SCL low in slave mode
- Programmable digital noise filtering
- Dual address mode, which uses slave address and slave memory or register address

30.3 I2C Architecture

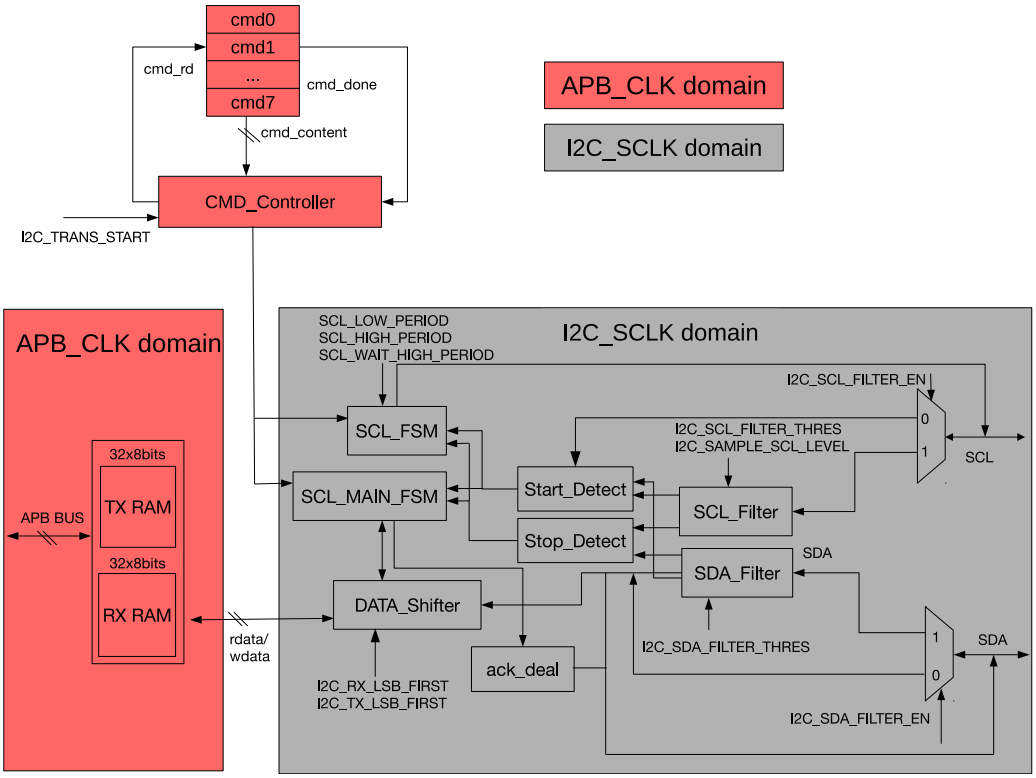


Figure 30.3-1. I2C Master Architecture

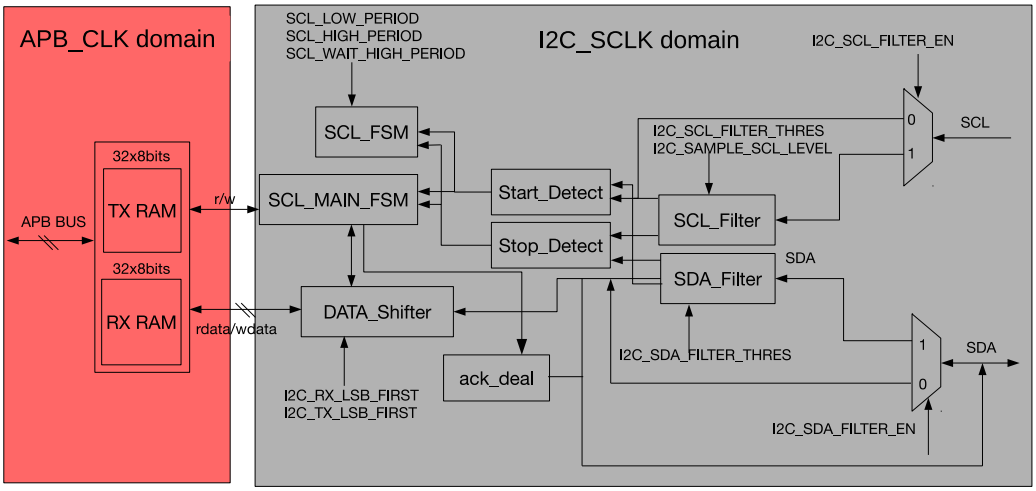


Figure 30.3-2. I2C Slave Architecture

The I2C controller runs either in master mode or slave mode, which is determined by I2C_MS_MODE. Figure 30.3-1 shows the architecture of a master, while Figure 30.3-2 shows that of a slave. The I2C controller has the following main parts:

- Transmit and receive memory (TX/RX RAM): store data to be transmitted and data received respectively.
- Command controller (CMD_Controller): generate (R)START, STOP, WRITE, READ, and END commands

- SCL clock controller (SCL_FSM): generate the timing sequence conforming to the I2C protocol. Figure 30.3-3 and Figure 30.3-1 are the timing diagram and corresponding parameters of the I2C protocol.
- SDA data controller (SCL_MAIN_FSM): control the execution of I2C commands and the data sequence of the SDA line. It also controls the ack_deal module to generate the ACK bit and detect the level of the ACK bit on the SDA line.
- Serial/parallel data converter (DATA_Shifter): shift data between serial and parallel form
- Filter for SCL (SCL_Filter): remove noises on SCL input signals
- Filter for SDA (SDA_Filter): remove noises on SDA input signals
- ACK bit controller (ack_deal): generate the ACK bit and detect the level of the ACK bit on the SDA line under the control of SCL_MAIN_FSM.

Besides, the I2C controller also has a clock module that generates I2C clocks, and a synchronization module that synchronizes the APB bus and the I2C controller.

The clock module is used to select clock sources, turn on and off clocks, and divide clocks. The synchronization module synchronizes signal transfer between different clock domains.

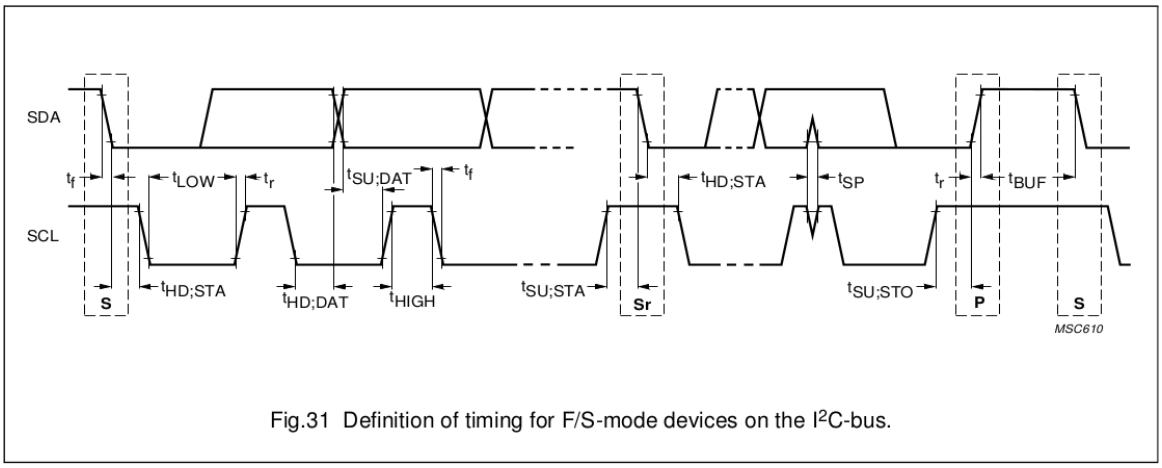


Fig.31 Definition of timing for F/S-mode devices on the I²C-bus.

Figure 30.3-3. I2C Protocol Timing (Cited from Fig.31 in The I2C-bus specification Version 2.1)

PARAMETER	SYMBOL	STANDARD-MODE		FAST-MODE		UNIT
		MIN.	MAX.	MIN.	MAX.	
SCL clock frequency	f_{SCL}	0	100	0	400	kHz
Hold time (repeated) START condition. After this period, the first clock pulse is generated	$t_{HD;STA}$	4.0	–	0.6	–	µs
LOW period of the SCL clock	t_{LOW}	4.7	–	1.3	–	µs
HIGH period of the SCL clock	t_{HIGH}	4.0	–	0.6	–	µs
Set-up time for a repeated START condition	$t_{SU;STA}$	4.7	–	0.6	–	µs
Data hold time: for CBUS compatible masters (see NOTE, Section 10.1.3) for I²C-bus devices	$t_{HD;DAT}$	 5.0 0(2)	 – 3.45(3)	 – 0(2)	 – 0.9(3)	 µs µs
Data set-up time	$t_{SU;DAT}$	250	–	100(4)	–	ns
Rise time of both SDA and SCL signals	t_r	–	1000	20 + 0.1C_b(5)	300	ns
Fall time of both SDA and SCL signals	t_f	–	300	20 + 0.1C_b(5)	300	ns
Set-up time for STOP condition	$t_{SU;STO}$	4.0	–	0.6	–	µs
Bus free time between a STOP and START condition	t_{BUF}	4.7	–	1.3	–	µs

Table 30.3-1. I2C Timing Parameters (Cited from Table 5 in The I2C-bus specification Version 2.1)

30.4 Functional Description

As mentioned above, one or more masters and one or more slaves can be mounted on the I2C bus. The following sections describe the operations of the ESP32-H2 I2C controller. Note that operations may differ between the I2C controller in ESP32-H2 and other masters or slaves on the bus. Please refer to datasheets of individual I2C devices for specific information.

30.4.1 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 choose the clock source for I2C_SCLK from XTAL_CLK or RC_FAST_CLK via PCR_I2C_SCLK_SEL:

- Enable the clock source for I2C_SCLK 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 according to the following equation:

$$Divisor = PCR\_I2C\_SCLK\_DIV\_NUM + 1 + \frac{PCR\_I2C\_SCLK\_DIV\_B}{PCR\_I2C\_SCLK\_DIV\_A}$$

Limited by timing parameters, the derived clock I2C_SCLK should operate at a frequency 20 times larger than SCL's frequency.

30.4.2 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.

30.4.3 SCL Clock Stretching

The I2C controller in slave mode (i.e., slave) can realize the function called clock stretching by holding the SCL line low to suspend data transmission in exchange for more time to process data. This function is enabled by setting the I2C_SLAVE_SCL_STRETCH_EN bit. The time period to release the SCL line from stretching is configured by the I2C_STRETCH_PROTECT_NUM field, in order to avoid timing sequence errors. The slave can choose to hold the SCL line low for clock stretching when one of the following four events occurs:

1. Address match: The address of the slave matches the address sent by the master via the SDA line, and the $R/\overline{W}$ bit is 1.
2. RAM being full: RX RAM of the slave is full. Note that when the slave receives less data than the FIFO depth, which is 32 bytes in ESP32-H2 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 the 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 cleared by setting the I2C_SLAVE_SCL_STRETCH_CLR bit.

30.4.4 Generating SCL Pulses in Idle State

Usually, when the I2C bus is idle, the SCL line is held high. The I2C controller in ESP32-H2 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.

30.4.5 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 30.4-1.

Table 30.4-1. 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	

30.4.6 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.

30.4.7 Timing Parameter Configuration

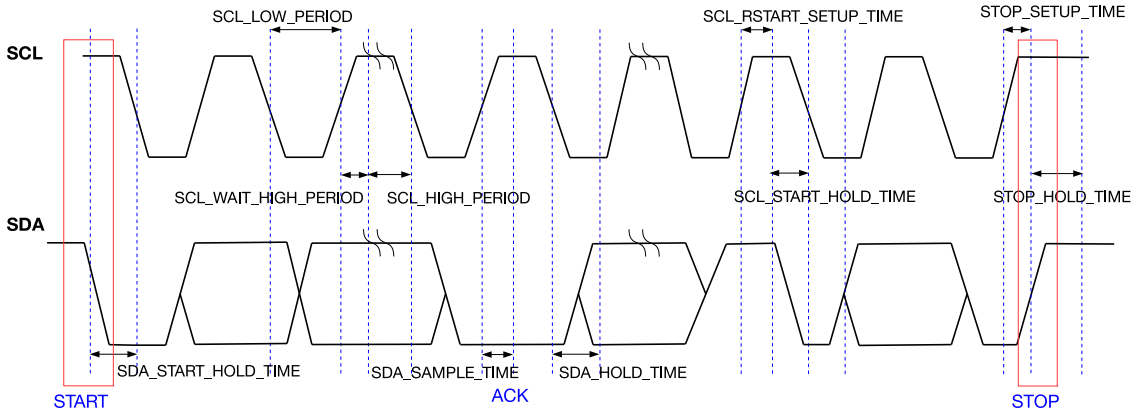


Figure 30.4-1. I2C Timing Diagram

Figure 30.4-1 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 (please refer to Table 30.3-1) 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 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 number 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

30.4.8 Timeout Control

The I2C controller has three types of timeout control, namely timeout control for SCL_FSM, SCL_MAIN_FSM, and 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.

30.4.9 Command Configuration

When the I2C controller works in master mode, CMD_Controller reads commands from 8 sequential command registers and controls SCL_FSM and SCL_MAIN_FSM accordingly.

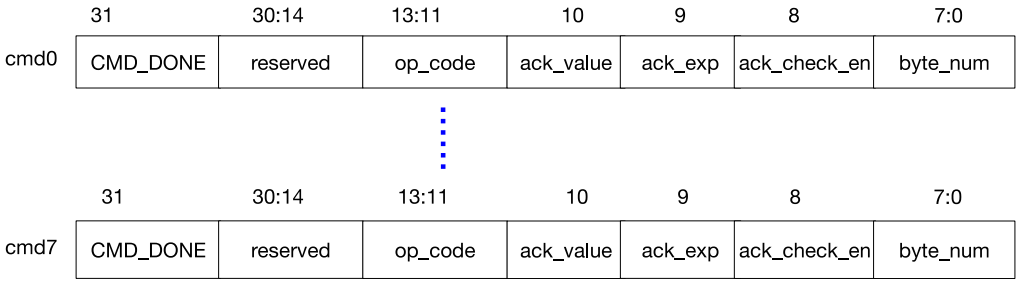


Figure 30.4-2. Structure of I2C Command Registers

Command registers, whose structure is illustrated in Figure 30.4-2, 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, 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 an 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. 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.

30.4.10 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 the I2C_NONFIFO_EN bit is cleared, both RAMs are accessed in FIFO mode; if the 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 completes.

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, 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).

30.4.11 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.

30.4.12 Addressing Mode

The ESP32-H2 I2C controller supports 7-bit and 10-bit addressing. 10-bit addressing can be mixed with 7-bit addressing. Besides, the ESP32-H2 I2C controller also supports dual address mode.

Assume the slave address is 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.

30.4.13 $R/\overline{W}$ 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 transmission errors may occur.

30.4.14 To Start the I2C Controller

To start the I2C controller in master mode, after configuring the controller to master mode and command registers, write 1 to I2C_TRANS_START in order to let the master starts 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.

30.5 Programming Example

This section provides programming examples for typical communication scenarios. ESP32-H2 has two I2C controllers. For the convenience of description, I2C masters and slaves in all subsequent figures are ESP32-H2 I2C controllers. I2C master is referred to as $I2C_{master}$, and I2C slave is referred to as $I2C_{slave}$.

30.5.1 $I2C_{master}$ Writes to $I2C_{slave}$ with a 7-bit Address in One Command Sequence

30.5.1.1 Introduction

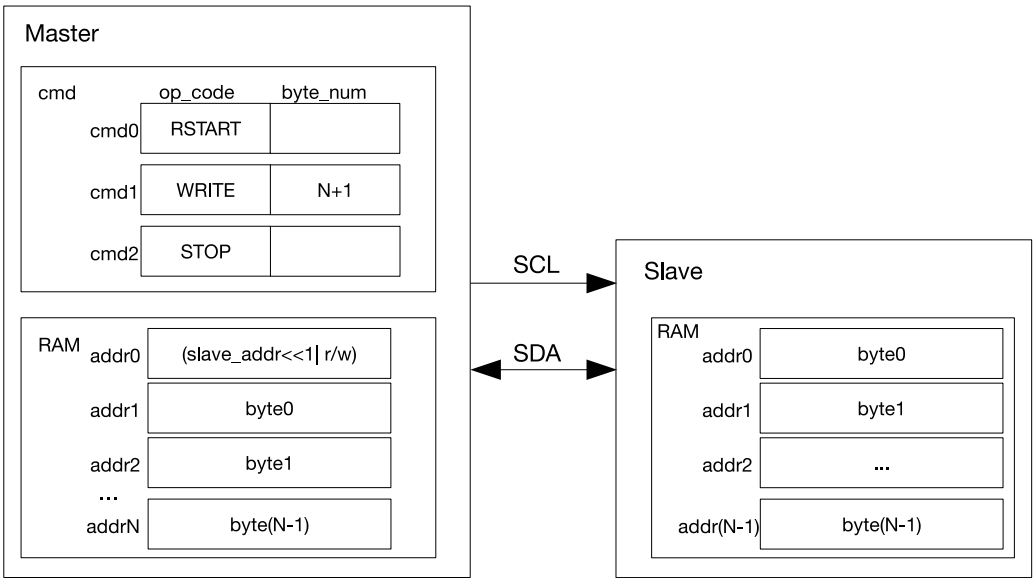


Figure 30.5-1. $I2C_{master}$ Writing to $I2C_{slave}$ with a 7-bit Address

Figure 30.5-1 shows how $I2C_{master}$ writes N bytes of data to $I2C_{slave}$ registers or RAM using 7-bit addressing. As shown in figure 30.5-1, 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 an 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.

30.5.1.2 Configuration Example

1. Configure the timing parameter registers of $I2C_{master}$ and $I2C_{slave}$ according to Section 30.4.7.

2. Set I2C_MS_MODE (master) to 1, and I2C_MS_MODE (slave) to 0.
3. Write 1 to I2C_CONF_UPGATE (master) and I2C_CONF_UPGATE (slave) to synchronize registers.
4. Configure command registers of $I2C_{master}$.

Command register	op_code	ack_value	ack_exp	ack_check_er	byte_num
I2C_COMMAND0 (master)	RSTART	—	—	—	—
I2C_COMMAND1 (master)	WRITE	ack_value	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 30.4.10.
6. Write the address of $I2C_{slave}$ to I2C_SLAVE_ADDR (slave) in the 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 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. $I2C_{master}$ 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 30.4.10.
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 30.4.10.

    If the 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.

30.5.2 $I2C_{master}$ Writes to $I2C_{slave}$ with a 10-bit Address in One Command Sequence

30.5.2.1 Introduction

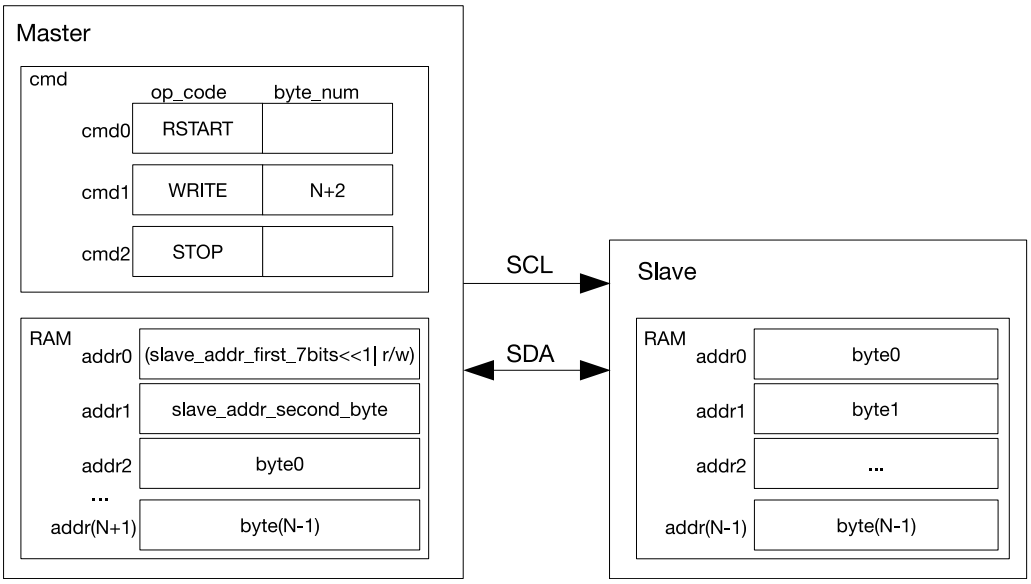


Figure 30.5-2. $I2C_{master}$ Writing to a Slave with a 10-bit Address

Figure 30.5-2 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 30.5.1, 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.

30.5.2.2 Configuration Example

1. Set I2C_MS_MODE (master) to 1, and I2C_MS_MODE (slave) to 0.
2. Write 1 to I2C_CONF_UPGATE (master) and I2C_CONF_UPGATE (slave) to synchronize registers.
3. Configure command registers of $I2C_{master}$.

Command registers	op_code	ack_value	ack_exp	ack_check_er	byte_num
I2C_COMMAND0 (master)	RSTART	—	—	—	—
I2C_COMMAND1 (master)	WRITE	ack_value	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 takes $I2C_{slave}$ as the 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 the 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 30.4.10.
11. If the 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 30.4.10.

    If the 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.

30.5.3 $I2C_{master}$ Writes to $I2C_{slave}$ with Two 7-bit Addresses in One Command Sequence

30.5.3.1 Introduction

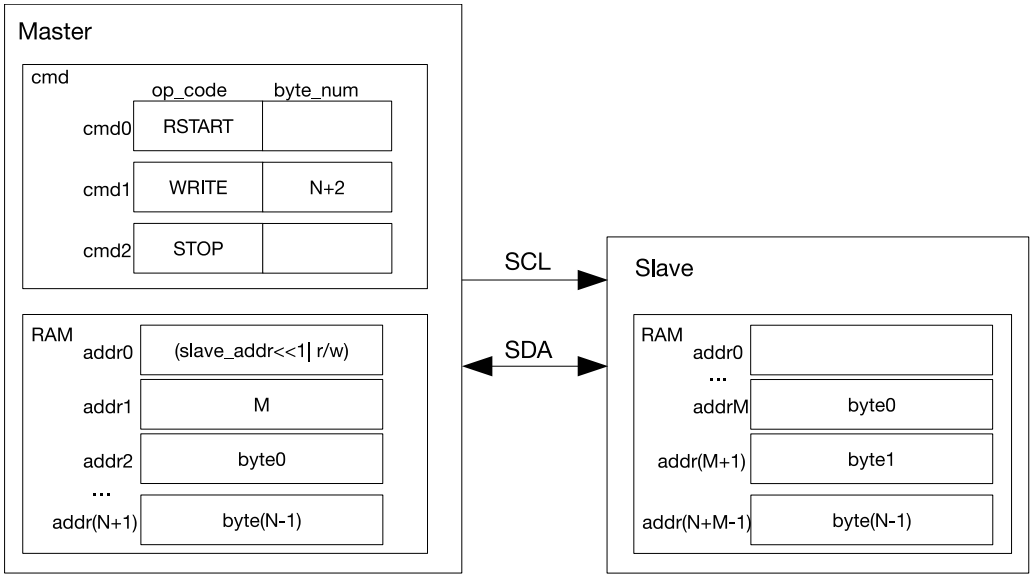


Figure 30.5-3. $I2C_{master}$ Writing to $I2C_{slave}$ with Two 7-bit Addresses

Figure 30.5-3 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 30.5.1, 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 30.5-3). When using double addressing, RAM must be accessed in non-FIFO mode. The I2C slave put received byte0 ~ byte(N-1) into its RAM in an order starting from addrM. The RAM is overwritten every 32 bytes.

30.5.3.2 Configuration Example

1. Set I2C_MS_MODE (master) to 1, and I2C_MS_MODE (slave) to 0.
2. Set I2C_FIFO_ADDR_CFG_EN (slave) to 1 to enable dual address mode.
3. Write 1 to I2C_CONF_UPGATE (master) and I2C_CONF_UPGATE (slave) to synchronize registers.
4. Configure command registers of $I2C_{master}$.

Command registers	op_code	ack_value	ack_exp	ack_check_er	byte_num
I2C_COMMAND0 (master)	RSTART	—	—	—	—
I2C_COMMAND1 (master)	WRITE	ack_value	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 the 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 30.4.10.
13. If data to be received is larger than RX FIFO depth, you may 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.

14. After data transfer completes, $I2C_{master}$ executes the STOP command, and generates an I2C_TRANS_COMPLETE_INT (master) interrupt.

30.5.4 $I2C_{master}$ Writes to $I2C_{slave}$ with a 7-bit Address in Multiple Command Sequences

30.5.4.1 Introduction

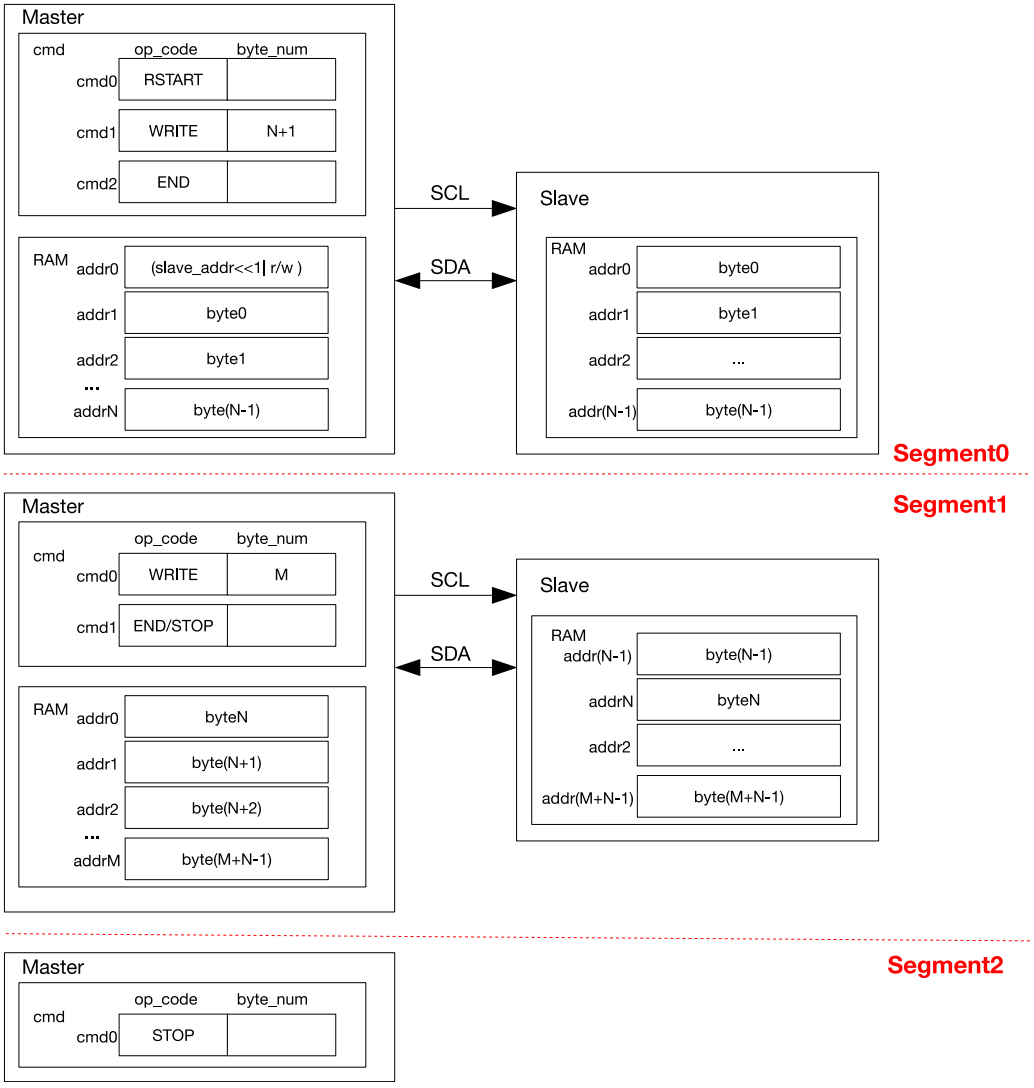


Figure 30.5-4. $I2C_{master}$ Writing to $I2C_{slave}$ with a 7-bit Address in Multiple Sequences

Given that the I2C controller RAM holds only the size of TX/RX FIFO depth, when data are too large to be processed, it is advised to transmit them in multiple command sequences. At the end of every command sequence is an END command. When the controller executes this END command, SCL will be pulled low, and the software can refresh command sequence registers and the RAM for next the transfer.

Figure 30.5-4 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.

30.5.4.2 Configuration Example

1. Set I2C_MS_MODE (master) to 1, and I2C_MS_MODE (slave) to 0.
2. Write 1 to I2C_CONF_UPGATE (master) and I2C_CONF_UPGATE (slave) to synchronize registers.
3. Configure command registers of $I2C_{master}$.

Command registers	op_code	ack_value	ack_exp	ack_check_er	byte_num
I2C_COMMAND0 (master)	RSTART	—	—	—	—
I2C_COMMAND1 (master)	WRITE	ack_value	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 30.4.10.
5. Write the address of $I2C_{slave}$ to I2C_SLAVE_ADDR (slave) in the 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 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.
9. $I2C_{master}$ sends data, and checks ACK value or not according to ack_check_en (master).

10. After the I2C_END_DETECT_INT (master) interrupt is generated, set I2C_END_DETECT_INT_CLR (master) to 1 to clear this interrupt.
11. Update $I2C_{master}$'s command registers.

Command registers	op_code	ack_value	ack_exp	ack_check_er	byte_num
I2C_COMMAND0 (master)	WRITE	ack_value	ack_exp	1	M
I2C_COMMAND1 (master)	END/STOP	—	—	—	—

12. Write M bytes of data to be sent to TX RAM of $I2C_{master}$ in FIFO or non-FIFO mode.
13. Write 1 to I2C_TRANS_START (master) bit to start the transfer and repeat step 9.
14. If the command is a STOP, I2C stops the transfer and generates an I2C_TRANS_COMPLETE_INT (master) interrupt.
15. If the command is an END, repeat step 10.
16. Update $I2C_{master}$'s command registers.

Command registers of $I2C_{master}$	op_code	ack_value	ack_exp	ack_check_er	byte_num
I2C_COMMAND1 (master)	STOP	—	—	—	—

17. Write 1 to I2C_TRANS_START (master) to start transfer.
18. $I2C_{master}$ executes the STOP command and generates an I2C_TRANS_COMPLETE_INT (master) interrupt.

30.5.5 $I2C_{master}$ Reads $I2C_{slave}$ with a 7-bit Address in One Command Sequence

30.5.5.1 Introduction

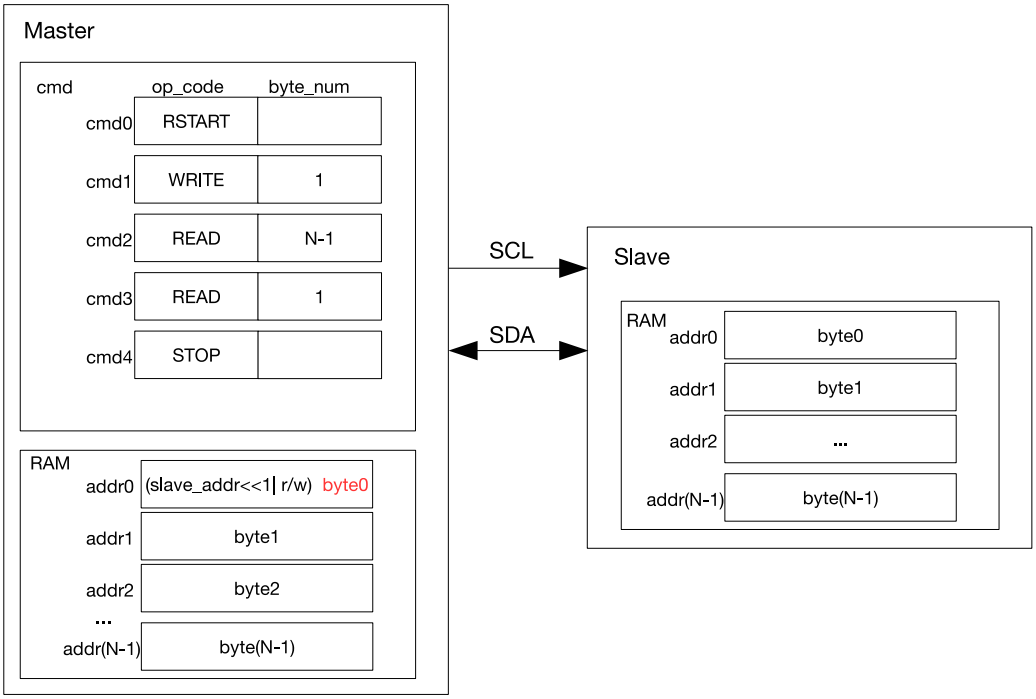


Figure 30.5-5. $I2C_{master}$ Reading $I2C_{slave}$ with a 7-bit Address

Figure 30.5-5 shows how $I2C_{master}$ reads N bytes of data from an I2C slave using 7-bit addressing. cmd1 is a WRITE command. When cmd1 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 30.5-5, $I2C_{master}$ 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 30.5-5.

30.5.5.2 Configuration Example

1. Set I2C_MS_MODE (master) to 1, and I2C_MS_MODE (slave) to 0.
2. It is recommended to set I2C_SLAVE_SCL_STRETCH_EN (slave) to 1, so that SCL can be held low for more processing time when $I2C_{slave}$ needs to send data. If this bit is not set, the software should write data to be sent to $I2C_{slave}$'s TX RAM before $I2C_{master}$ initiates the transfer. The configuration below is applicable to the scenario where I2C_SLAVE_SCL_STRETCH_EN (slave) is 1.
3. Write 1 to I2C_CONF_UPGATE (master) and I2C_CONF_UPGATE (slave) to synchronize registers.
4. Configure command registers of $I2C_{master}$.

Command registers of $I2C_{master}$	op_code	ack_value	ack_exp	ack_check_er	byte_num

I2C_COMMAND0 (master)	RSTART	—	—	—	—
I2C_COMMAND1 (master)	WRITE	0	0	1	1
I2C_COMMAND2 (master)	READ	0	0	1	N-1
I2C_COMMAND3 (master)	READ	1	0	1	1
I2C_COMMAND4 (master)	STOP	—	—	—	—

5. Write the address of $I2C_{slave}$ to TX RAM of $I2C_{master}$ in either FIFO mode or non-FIFO mode according to Section 30.4.10.
6. Write the address of $I2C_{slave}$ to I2C_SLAVE_ADDR (slave) in the 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 30.4.14.
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 30.4.10.
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 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. 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.

30.5.6 I2C$_{master}$ Reads I2C$_{slave}$ with a 10-bit Address in One Command Sequence

30.5.6.1 Introduction

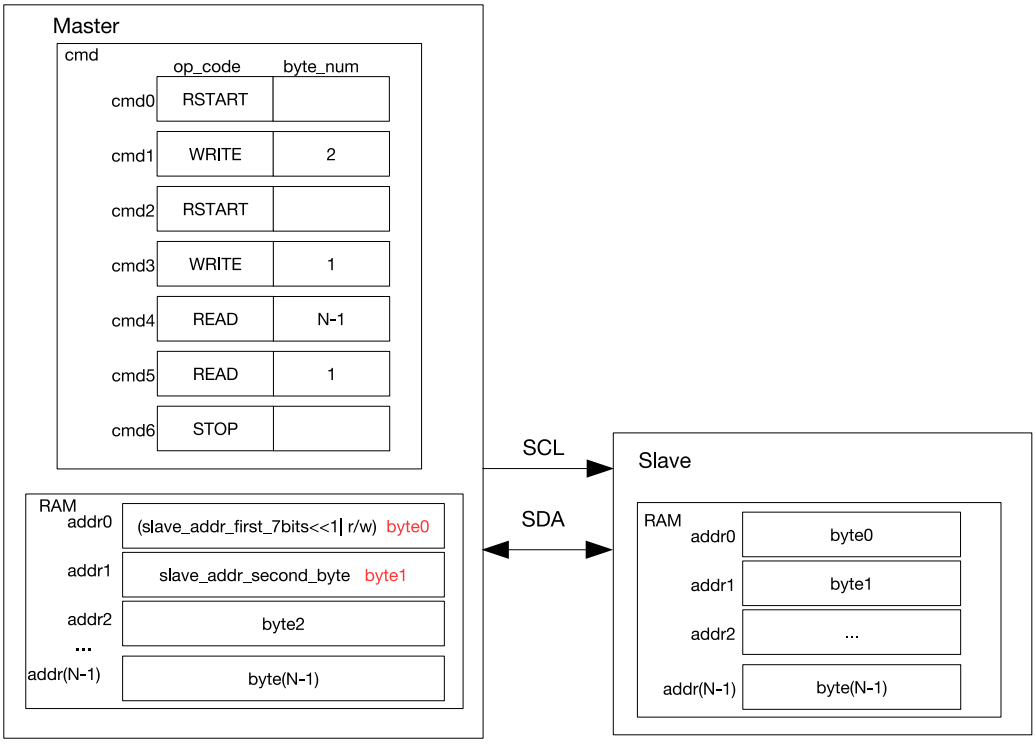


Figure 30.5-6. I2C$_{master}$ Reading I2C$_{slave}$ with a 10-bit Address

Figure 30.5-6 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 an 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 30.5.2.

30.5.6.2 Configuration Example

1. Set I2C_MS_MODE (master) to 1, and I2C_MS_MODE (slave) to 0.
2. We recommend setting I2C_SLAVE_SCL_STRETCH_EN (slave) to 1, so that SCL can be held low for more processing time when I2C$_{slave}$ needs to send data. If this bit is not set, the software should write data to be sent to I2C$_{slave}$'s TX RAM before I2C$_{master}$ initiates the transfer. The configuration below is applicable to a scenario where I2C_SLAVE_SCL_STRETCH_EN (slave) is 1.
3. Write 1 to I2C_CONF_UPGATE (master) and I2C_CONF_UPGATE (slave) to synchronize registers.
4. Configure command registers of I2C$_{master}$.

Command registers of I2C$_{master}$	op_code	ack_value	ack_exp	ack_check_er	byte_num

I2C_COMMAND0 (master)	RSTART	—	—	—	—
I2C_COMMAND1 (master)	WRITE	0	0	1	2
I2C_COMMAND2 (master)	RSTART	—	—	—	—
I2C_COMMAND3 (master)	WRITE	0	0	1	1
I2C_COMMAND4 (master)	READ	0	0	1	N-1
I2C_COMMAND5 (master)	READ	1	0	1	1
I2C_COMMAND6 (master)	STOP	—	—	—	—

5. Configure I2C_SLAVE_ADDR (slave) in I2C_SLAVE_ADDR_REG (slave) as $I2C_{slave}$'s 10-bit address, and set I2C_ADDR_10BIT_EN (slave) to 1 to enable 10-bit addressing.
6. Write the address of $I2C_{slave}$ and data to be sent to TX RAM of $I2C_{master}$ in either FIFO or non-FIFO mode. The first byte of address comprises ((0x78 | I2C_SLAVE_ADDR[9:8])«1) and a $R/\overline{W}$ bit, which is 1 and indicates a WRITE operation. The second byte of address is I2C_SLAVE_ADDR[7:0]. The third byte is ((0x78 | I2C_SLAVE_ADDR[9:8])«1) and a $R/\overline{W}$ bit, which is 1 and indicates a READ operation.
7. Write 1 to I2C_CONF_UPGATE (master) and I2C_CONF_UPGATE (slave) to synchronize registers.
8. Write 1 to I2C_TRANS_START (master) to start $I2C_{master}$'s transfer.
9. Start $I2C_{slave}$'s transfer according to Section 30.4.14.
10. $I2C_{slave}$ compares the slave address sent by $I2C_{master}$ with its own address in I2C_SLAVE_ADDR (slave). When ack_check_en (master) in $I2C_{master}$'s WRITE command is 1, $I2C_{master}$ checks ACK value each time it sends a byte. When ack_check_en (master) is 0, $I2C_{master}$ does not check ACK value and takes $I2C_{slave}$ as a matching slave by default.
    - Match: If the received ACK value matches ack_exp (master) (the expected ACK value), $I2C_{master}$ continues data transfer.
    - Not match: If the received ACK value does not match ack_exp, $I2C_{master}$ generates an I2C_NACK_INT (master) interrupt and stops data transfer.
11. $I2C_{master}$ sends an 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 30.4.10.
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 the 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.

30.5.7 $I2C_{master}$ Reads $I2C_{slave}$ with Two 7-bit Addresses in One Command Sequence

30.5.7.1 Introduction

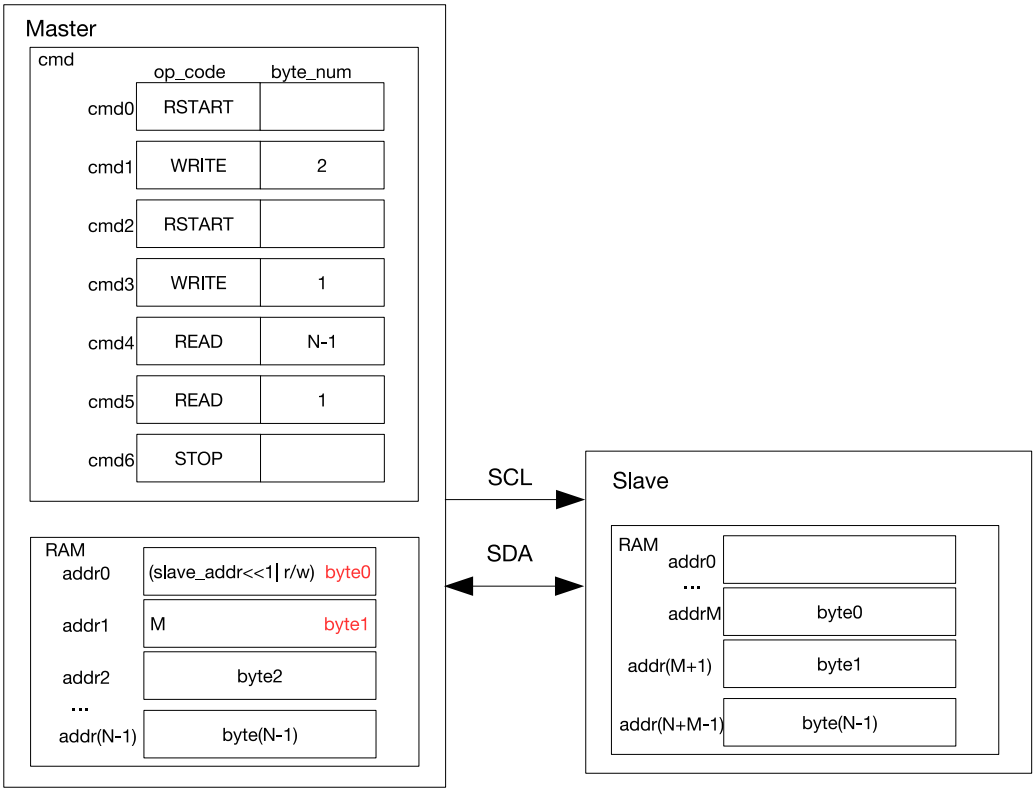


Figure 30.5-7. $I2C_{master}$ Reading N Bytes of Data from addrM of $I2C_{slave}$ with a 7-bit Address

Figure 30.5-7 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 an 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.

30.5.7.2 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 a scenario where I2C_SLAVE_SCL_STRETCH_EN (slave) is 1.
3. Set I2C_FIFO_ADDR_CFG_EN (slave) to 1 to enable dual address mode.
4. Write 1 to I2C_CONF_UPGATE (master) and I2C_CONF_UPGATE (slave) to synchronize registers.
5. Configure command registers of $I2C_{master}$.

Command registers of $I2C_{master}$	op_code	ack_value	ack_exp	ack_check_er	byte_num
I2C_COMMAND0 (master)	RSTART	—	—	—	—
I2C_COMMAND1 (master)	WRITE	0	0	1	2
I2C_COMMAND2 (master)	RSTART	—	—	—	—
I2C_COMMAND3 (master)	WRITE	0	0	1	1
I2C_COMMAND4 (master)	READ	0	0	1	N-1
I2C_COMMAND5 (master)	READ	1	0	1	1
I2C_COMMAND6 (master)	STOP	—	—	—	—

6. Configure I2C_SLAVE_ADDR (slave) in the 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 30.4.10. 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 30.4.14.
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 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.

12. $I2C_{slave}$ receives memory address sent by $I2C_{master}$ and adds the offset.
13. $I2C_{master}$ sends an 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. 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.
16. Write data to be sent to TX RAM of $I2C_{slave}$ in either FIFO mode or non-FIFO mode according to Section 30.4.10.
17. Set I2C_SLAVE_SCL_STRETCH_CLR (slave) to 1 to release SCL.
18. $I2C_{slave}$ sends data, and $I2C_{master}$ checks ACK value or not according to ack_check_en (master) in the READ command.
19. 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.
20. 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.
21. After data transfer completes, $I2C_{master}$ executes the STOP command, and generates an I2C_TRANS_COMPLETE_INT (master) interrupt.

30.5.8 $I2C_{master}$ Reads $I2C_{slave}$ with a 7-bit Address in Multiple Command Sequences

30.5.8.1 Introduction

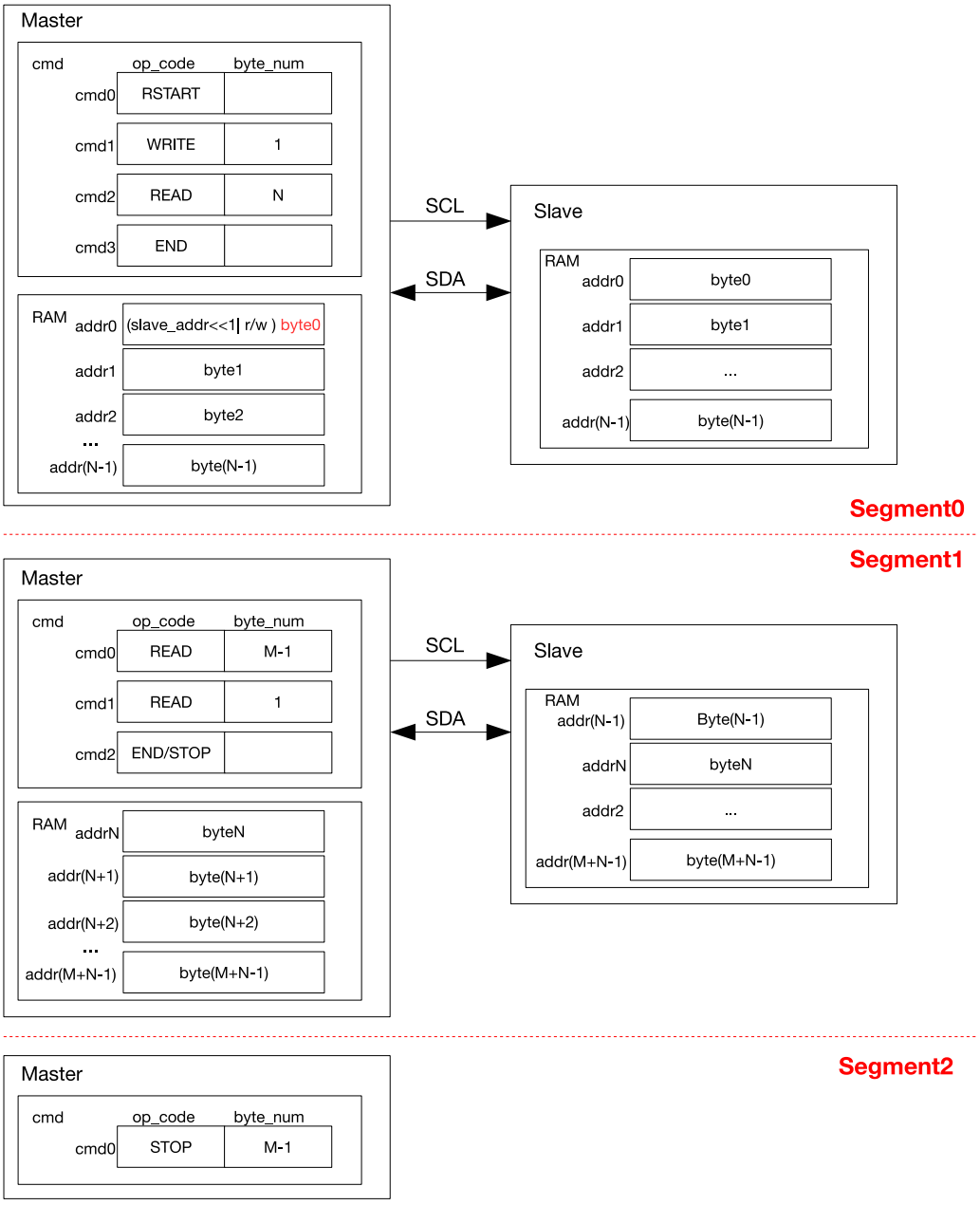


Figure 30.5-8. $I2C_{master}$ Reading $I2C_{slave}$ with a 7-bit Address in Segments

Figure 30.5-8 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 30.5-5, 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.

30.5.8.2 Configuration Example

1. Set I2C_MS_MODE (master) to 1, and I2C_MS_MODE (slave) to 0.
2. We recommend setting I2C_SLAVE_SCL_STRETCH_EN (slave) to 1, so that SCL can be held low for more processing time when $I2C_{slave}$ needs to send data. If this bit is not set, the software should write data to be sent to $I2C_{slave}$'s TX RAM before $I2C_{master}$ initiates the transfer. The configuration below is applicable to a scenario where I2C_SLAVE_SCL_STRETCH_EN (slave) is 1.
3. Write 1 to I2C_CONF_UPGATE (master) and I2C_CONF_UPGATE (slave) to synchronize registers.
4. Configure command registers of $I2C_{master}$.

Command registers of $I2C_{master}$	op_code	ack_value	ack_exp	ack_check_er	byte_num
I2C_COMMAND0 (master)	RSTART	—	—	—	—
I2C_COMMAND1 (master)	WRITE	0	0	1	1
I2C_COMMAND2 (master)	READ	0	0	1	N
I2C_COMMAND3 (master)	END	—	—	—	—

5. Write the address of $I2C_{slave}$ to TX RAM of $I2C_{master}$ in FIFO or non-FIFO mode.
6. Write the address of $I2C_{slave}$ to I2C_SLAVE_ADDR (slave) in the 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 30.4.14.
10. $I2C_{slave}$ compares the slave address sent by $I2C_{master}$ with its own address in I2C_SLAVE_ADDR (slave). When ack_check_en (master) in $I2C_{master}$'s WRITE command is 1, $I2C_{master}$ checks ACK value each time it sends a byte. When ack_check_en (master) is 0, $I2C_{master}$ does not check ACK value and takes $I2C_{slave}$ as a matching slave by default.
    - Match: If the received ACK value matches ack_exp (master) (the expected ACK value), $I2C_{master}$ continues data transfer.
    - Not match: If the received ACK value does not match ack_exp, $I2C_{master}$ generates an I2C_NACK_INT (master) interrupt and stops data transfer.
11. After I2C_SLAVE_STRETCH_INT (slave) is generated, the I2C_STRETCH_CAUSE bit is 0. The address of $I2C_{slave}$ matches the address sent over SDA, and $I2C_{slave}$ needs to send data.
12. Write data to be sent to TX RAM of $I2C_{slave}$ in either FIFO mode or non-FIFO mode according to Section 30.4.10.
13. Set I2C_SLAVE_SCL_STRETCH_CLR (slave) to 1 to release SCL.

14. $I2C_{slave}$ sends data, and $I2C_{master}$ checks ACK value or not according to ack_check_en (master) in the READ command.
15. If data to be read by $I2C_{master}$ in one READ command (N or M) is larger than the TX FIFO depth of $I2C_{slave}$, an I2C_SLAVE_STRETCH_INT (slave) interrupt will be generated when TX RAM of $I2C_{slave}$ becomes empty. In this way, $I2C_{slave}$ can hold SCL low, so that software has more time to pad data in TX RAM of $I2C_{slave}$ and read data in RX RAM of $I2C_{master}$. After the software has finished reading, you can set I2C_SLAVE_STRETCH_INT_CLR (slave) to 1 to clear interrupt, and set I2C_SLAVE_SCL_STRETCH_CLR (slave) to release the SCL line.
16. Once finishing reading data in the first READ command, $I2C_{master}$ executes the END command and triggers an I2C_END_DETECT_INT (master) interrupt, which is cleared by setting I2C_END_DETECT_INT_CLR (master) to 1.
17. Update $I2C_{master}$'s command registers using one of the following two methods:

Command registers of $I2C_{master}$	op_code	ack_value	ack_exp	ack_check_er	byte_num
I2C_COMMAND0 (master)	READ	ack_value	ack_exp	1	M
I2C_COMMAND1 (master)	END	—	—	—	—

Or

Command registers of $I2C_{master}$	op_code	ack_value	ack_exp	ack_check_er	byte_num
I2C_COMMAND0 (master)	READ	0	0	1	M-1
I2C_COMMAND1 (master)	READ	1	0	1	1
I2C_COMMAND2 (master)	STOP	—	—	—	—

18. Write M bytes of data to be sent to TX RAM of $I2C_{slave}$. If M is larger than the TX FIFO depth, then repeat step 12 in FIFO or non-FIFO mode.
19. Write 1 to I2C_TRANS_START (master) bit to start the transfer and repeat step 14.
20. If the last command is a STOP, then set ack_value (master) to 1 after $I2C_{master}$ has received the last byte of data. $I2C_{slave}$ stops transfer upon the I2C_NACK_INT interrupt. $I2C_{master}$ executes the STOP command to stop the transfer and generates an I2C_TRANS_COMPLETE_INT (master) interrupt.
21. If the last command is an END, then repeat step 16 and proceed to the next steps.
22. Update $I2C_{master}$'s command registers.

Command registers of $I2C_{master}$	op_code	ack_value	ack_exp	ack_check_er	byte_num
I2C_COMMAND1 (master)	STOP	—	—	—	—

23. Write 1 to I2C_TRANS_START (master) bit to start transfer.

24. $I2C_{master}$ executes the STOP command to stop transfer, and generates an I2C_TRANS_COMPLETE_INT (master) interrupt.

30.6 Interrupts

- 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_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_OVF_INT: Triggered when RX FIFO of the I2C controller overflows.
- I2C_END_DETECT_INT: Triggered when op_code of the master indicates an END command and an END condition is detected.
- I2C_BYTE_TRANS_DONE_INT: Triggered when the I2C controller sends or receives a byte.
- 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_MST_TXFIFO_UDF_INT: Triggered when TX FIFO of the master underflows.
- I2C_TRANS_COMPLETE_INT: Triggered when the I2C controller detects a STOP bit.
- I2C_TIME_OUT_INT: Triggered when SCL stays high or low for more than $2^{I2C\_TIME\_OUT\_VALUE}$ clock cycles during data transfer.
- I2C_TRANS_START_INT: Triggered when the I2C controller sends a START bit.
- 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_TXFIFO_OVF_INT: Triggered when the I2C controller writes TX FIFO via the APB bus, but TX FIFO is full.
- I2C_RXFIFO_UDF_INT: Triggered when the I2C controller reads RX FIFO via the APB bus, but RX FIFO is empty.
- I2C_SCL_ST_TO_INT: Triggered when the state machine SCL_FSM remains unchanged for over I2C_SCL_ST_TO_I2C[23:0] clock cycles.
- I2C_SCL_MAIN_ST_TO_INT: Triggered when the main state machine SCL_MAIN_FSM remains unchanged for over I2C_SCL_MAIN_ST_TO_I2C[23:0] clock cycles.
- I2C_DET_START_INT: Triggered when the master or the slave detects a START signal.
- I2C_SLAVE_STRETCH_INT: Generated when one of the four stretching events occurs in slave mode.
- I2C_GENERAL_CALL_INT: Triggered when the slave receives general call address.
- I2C_SLAVE_ADDR_UNMATCH_INT: Triggered when the received slave address is inconsistent with the internally configured slave address in slave mode.

30.7 Register Summary

The addresses in this section are relative to I2C Controller base address provided in Table 4.3-2 in Chapter 4 *System and Memory*.

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

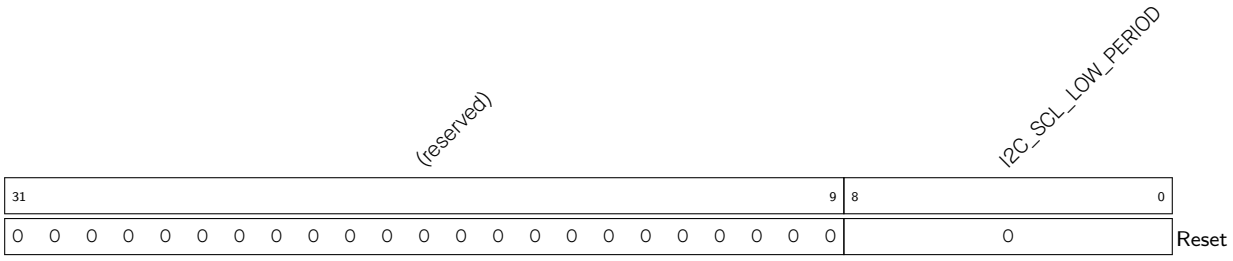
Name	Description	Address	Access
Timing registers			
I2C_SCL_LOW_PERIOD_REG	Configures the low level width of the SCL Clock	0x0000	R/W
I2C_SDA_HOLD_REG	Configures the hold time after a negative SCL edge	0x0030	R/W
I2C_SDA_SAMPLE_REG	Configures the sample time after a positive SCL edge	0x0034	R/W
I2C_SCL_HIGH_PERIOD_REG	Configures the high level width of SCL	0x0038	R/W
I2C_SCL_START_HOLD_REG	Configures the delay between the SDA and SCL negative edge for a start condition	0x0040	R/W
I2C_SCL_RSTART_SETUP_REG	Configures the delay between the positive edge of SCL and the negative edge of SDA	0x0044	R/W
I2C_SCL_STOP_HOLD_REG	Configures the delay after the SCL clock edge for a stop condition	0x0048	R/W
I2C_SCL_STOP_SETUP_REG	Configures the delay between the SDA and SCL rising edge for a stop condition Measurement unit: i2c_sclk	0x004C	R/W
I2C_SCL_ST_TIME_OUT_REG	SCL status timeout register	0x0078	R/W
I2C_SCL_MAIN_ST_TIME_OUT_REG	SCL main status timeout register	0x007C	R/W
Configuration registers			
I2C_CTR_REG	Transmission setting register	0x0004	varies
I2C_TO_REG	Timeout control register for receiving data	0x000C	R/W
I2C_SLAVE_ADDR_REG	Local slave address setting register	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	SCL stretch setting register of I2C slave	0x0084	varies
Status registers			
I2C_SR_REG	I2C working status register	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 register	0x0020	R/SS/WTC
I2C_INT_CLR_REG	Interrupt clear register	0x0024	WT
I2C_INT_ENA_REG	Interrupt enable register	0x0028	R/W
I2C_INT_STATUS_REG	Status register 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 control register	0x00F8	R/W
Address register			
I2C_TXFIFO_START_ADDR_REG	I2C TXFIFO base address register	0x0100	HRO
I2C_RXFIFO_START_ADDR_REG	I2C RXFIFO base address register	0x0180	HRO

30.8 Registers

The addresses in this section are relative to I2C Controller base address provided in Table 4.3-2 in Chapter 4 *System and Memory*.

Register 30.1. I2C_SCL_LOW_PERIOD_REG (0x0000)

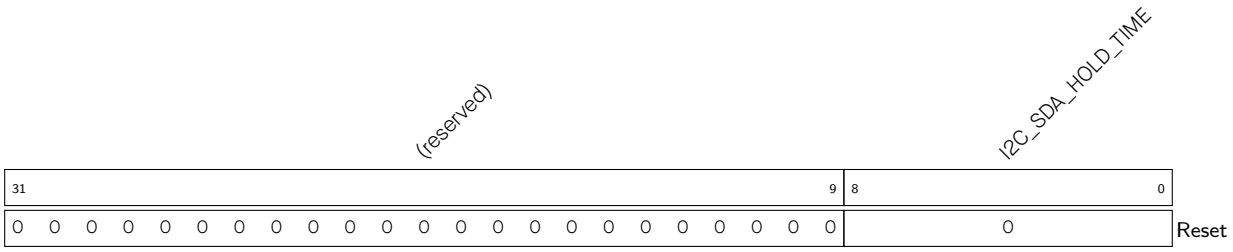


I2C_SCL_LOW_PERIOD Configures the low level width of the SCL Clock in Master mode.

Measurement unit: I2C_SCLK

(R/W)

Register 30.2. I2C_SDA_HOLD_REG (0x0030)

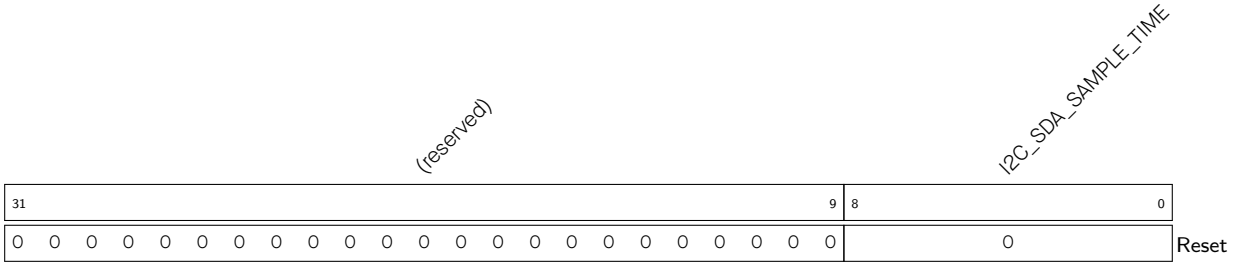


I2C_SDA_HOLD_TIME Configures the time to hold the data after the falling edge of SCL.

Measurement unit: I2C_SCLK

(R/W)

Register 30.3. I2C_SDA_SAMPLE_REG (0x0034)

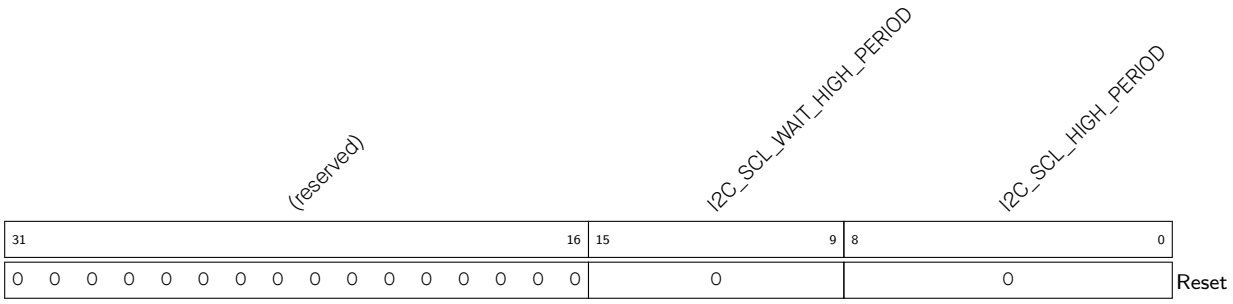


I2C_SDA_SAMPLE_TIME Configures the time for sampling SDA.

Measurement unit: I2C_SCLK

(R/W)

Register 30.4. I2C_SCL_HIGH_PERIOD_REG (0x0038)



I2C_SCL_HIGH_PERIOD Configures for how long SCL remains high in Master mode.

Measurement unit: I2C_SCLK

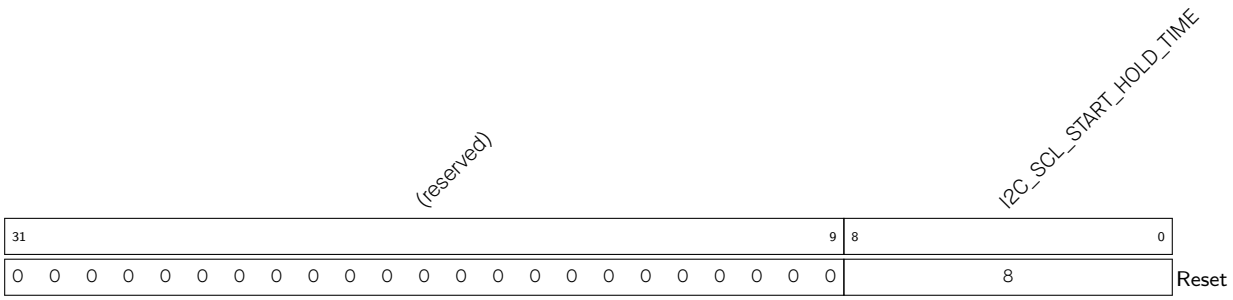
(R/W)

I2C_SCL_WAIT_HIGH_PERIOD Configures the SCL_FSM's waiting period for SCL high level in Master mode.

Measurement unit: I2C_SCLK

(R/W)

Register 30.5. I2C_SCL_START_HOLD_REG (0x0040)

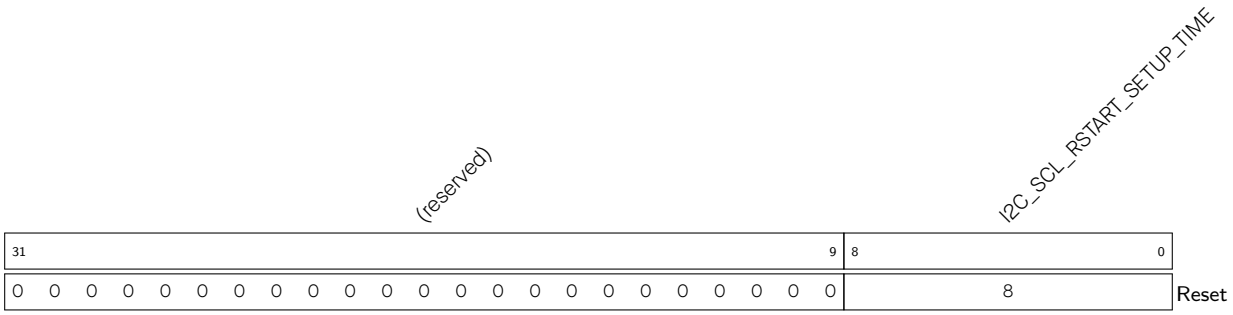


I2C_SCL_START_HOLD_TIME Configures the time between the falling edge of SDA and the falling edge of SCL for a START condition.

Measurement unit: I2C_SCLK

(R/W)

Register 30.6. I2C_SCL_RSTART_SETUP_REG (0x0044)

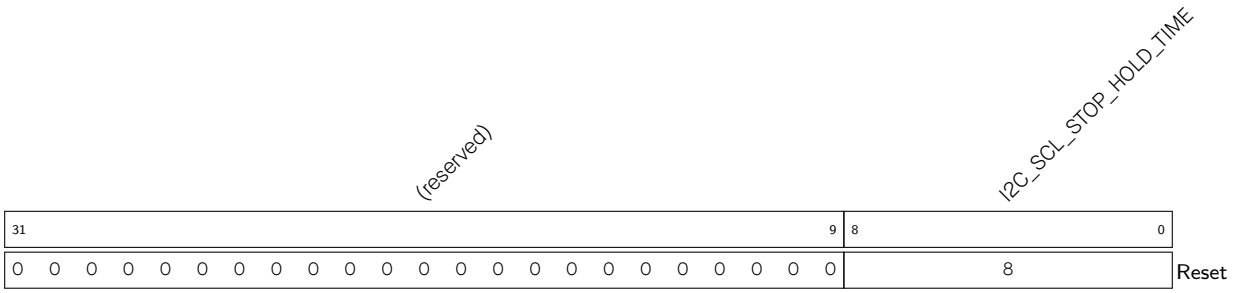


I2C_SCL_RSTART_SETUP_TIME Configures the time between the positive edge of SCL and the negative edge of SDA for a RESTART condition.

Measurement unit: I2C_SCLK

(R/W)

Register 30.7. I2C_SCL_STOP_HOLD_REG (0x0048)

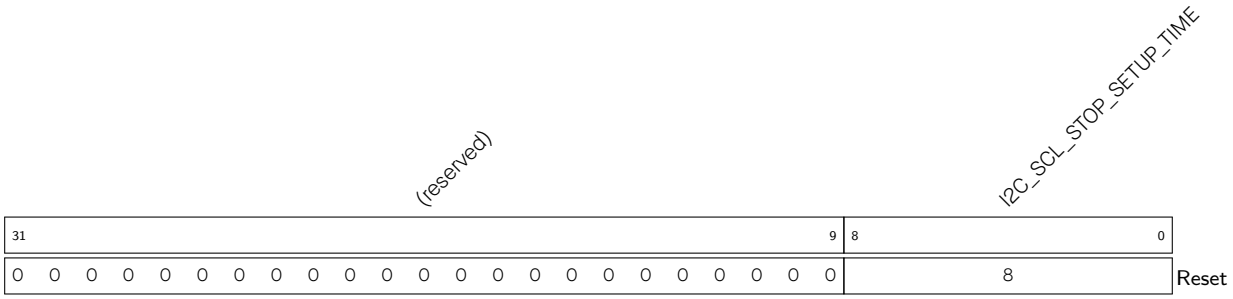


I2C_SCL_STOP_HOLD_TIME Configures the delay after the STOP condition.

Measurement unit: I2C_SCLK

(R/W)

Register 30.8. I2C_SCL_STOP_SETUP_REG (0x004C)

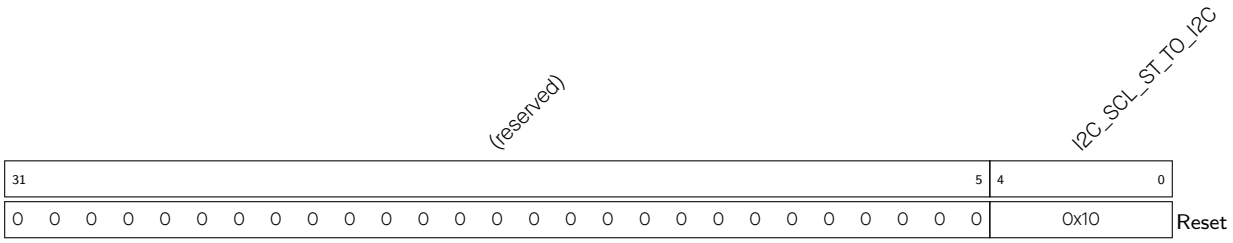


I2C_SCL_STOP_SETUP_TIME Configures the time between the rising edge of SCL and the rising edge of SDA.

Measurement unit: I2C_SCLK

(R/W)

Register 30.9. I2C_SCL_ST_TIME_OUT_REG (0x0078)



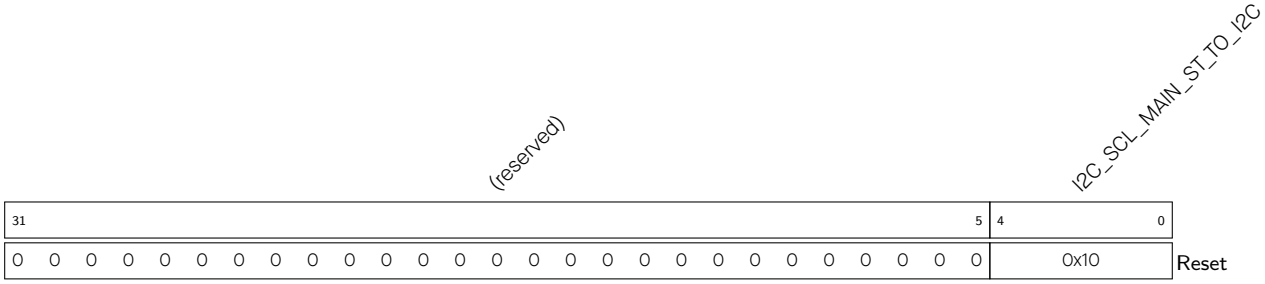
I2C_SCL_ST_TO_I2C Configures the threshold value of SCL_FSM state unchanged period. It should be no more than 23, more than 1.

$$\text{Maximum Clock Cycle Threshold} = 2^{\text{I2C\_SCL\_ST\_TO\_I2C}+1}-1$$

Measurement unit: I2C_SCLK

(R/W)

Register 30.10. I2C_SCL_MAIN_ST_TIME_OUT_REG (0x007C)

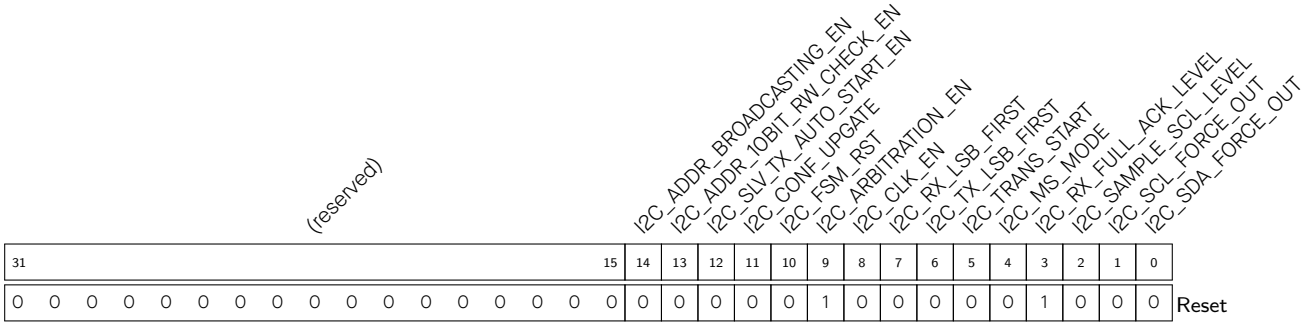


I2C_SCL_MAIN_ST_TO_I2C Configures the threshold value of SCL_MAIN_FSM state unchanged period. It should be no more than 23.

Measurement unit: I2C_SCLK

(R/W)

Register 30.11. 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 master when rx_fifo_cnt 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 whether or not 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 whether or not 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 30.11. 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 whether to enable I2C bus arbitration detection.

0: No effect

1: Enable

(R/W)

I2C_FSM_RST Configures whether to reset the SCL_FSM.

0: No effect

1: Reset (WT)

I2C_CONF_UPGATE Configures the bit for synchronization.

0: No effect

1: Synchronize (WT)

I2C_SLV_TX_AUTO_START_EN Configures whether to enable slave to send data automatically

0: Disable

1: Enable

(R/W)

I2C_ADDR_10BIT_RW_CHECK_EN Configures whether to check if the $R/\overline{W}$ bit of 10-bit addressing consists with I2C protocol.

0: Not check

1: Check (R/W)

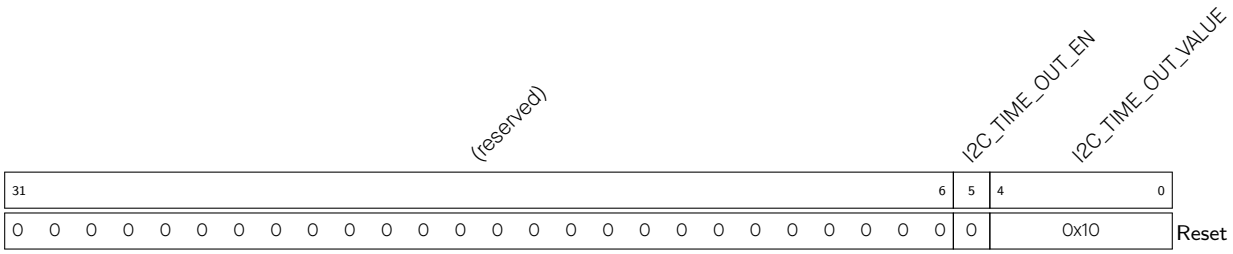
I2C_ADDR_BROADCASTING_EN Configures whether to support the 7-bit general call function.

0: Not support

1: Support

(R/W)

Register 30.12. I2C_TO_REG (0x000C)



I2C_TIME_OUT_VALUE Configures the timeout threshold period for SCL stucking at the high or low level. The actual period is 2^(I2C_TIME_OUT_VALUE+1)−1.

Measurement unit: I2C_SCLK

(R/W)

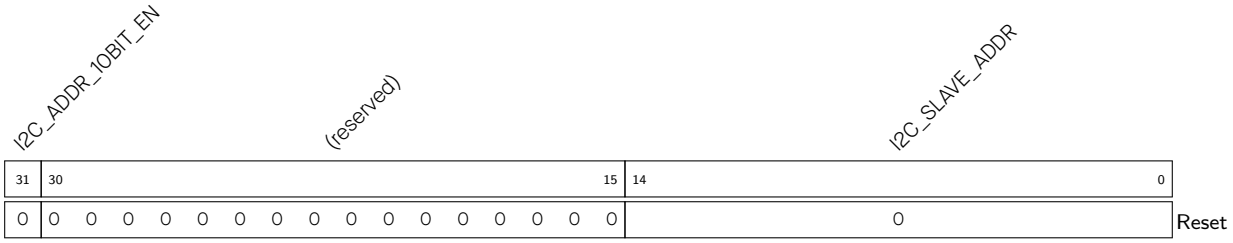
I2C_TIME_OUT_EN Configures whether to enable timeout control.

0: No effect

1: Enable

(R/W)

Register 30.13. I2C_SLAVE_ADDR_REG (0x0010)



I2C_SLAVE_ADDR Configures the slave address of the I2C slave.

(R/W)

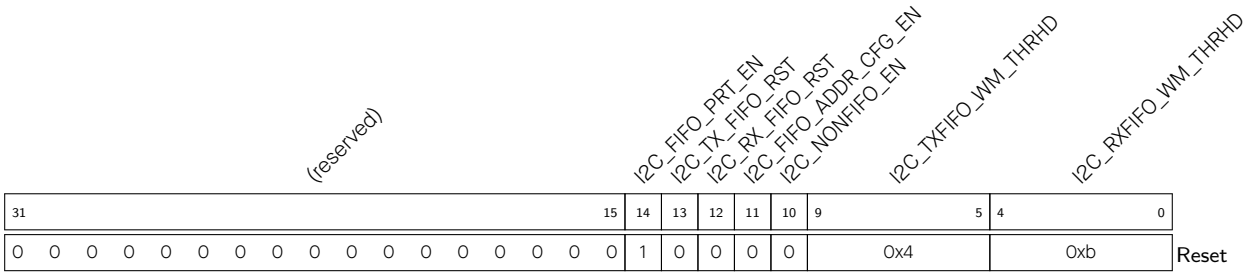
I2C_ADDR_10BIT_EN Configures whether to enable the slave 10-bit addressing mode in Master mode.

0: No effect

1: Enable

(R/W)

Register 30.14. 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], the 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], the I2C_TXFIFO_WM_INT_RAW bit will be valid. (R/W)

I2C_NONFIFO_EN Configures whether to enable APB non-FIFO access. (R/W)

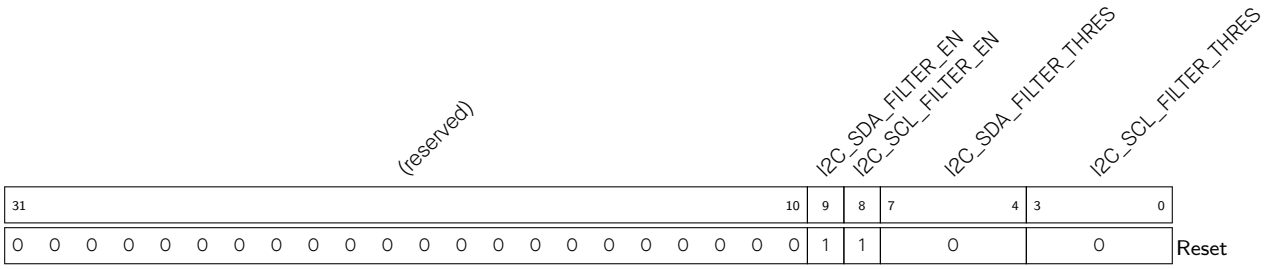
I2C_FIFO_ADDR_CFG_EN Configures the slave to enable dual address mode. When this mode is enabled, the byte received after the I2C address byte represents the offset address in the I2C Slave RAM.
0: Disable
1: Enable
(R/W)

I2C_RX_FIFO_RST Configures whether or not to reset RX FIFO.
0: No effect
1: Reset (R/W)

I2C_TX_FIFO_RST Configures whether or not to reset TX FIFO.
0: No effect
1: Reset (R/W)

I2C_FIFO_PRT_EN Configures whether 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 30.15. I2C_FILTER_CFG_REG (0x0050)



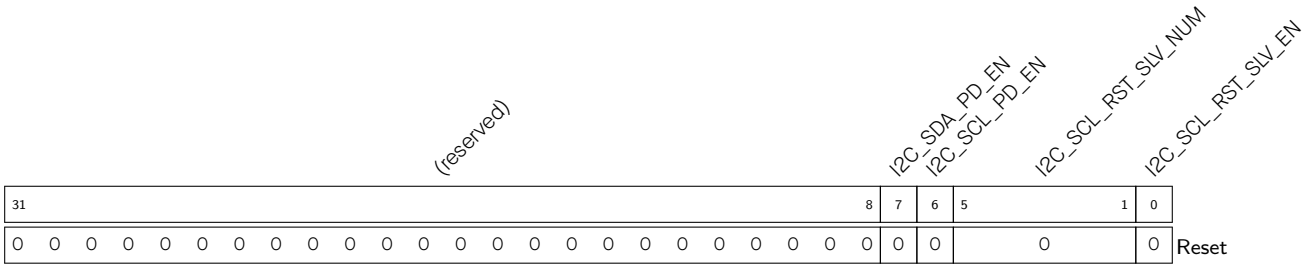
I2C_SCL_FILTER_THRES Configures the threshold pulse width to be filtered on SCL. When a pulse on the SCL input has smaller width than this register value, the I2C controller will ignore that pulse. Measurement unit: I2C_SCLK
(R/W)

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
(R/W)

I2C_SCL_FILTER_EN Configures whether to enable the filter function for SCL.
0: No effect
1: Enable
(R/W)

I2C_SDA_FILTER_EN Configures whether to enable the filter function for SDA.
0: No effect
1: Enable
(R/W)

Register 30.16. I2C_SCL_SP_CONF_REG (0x0080)



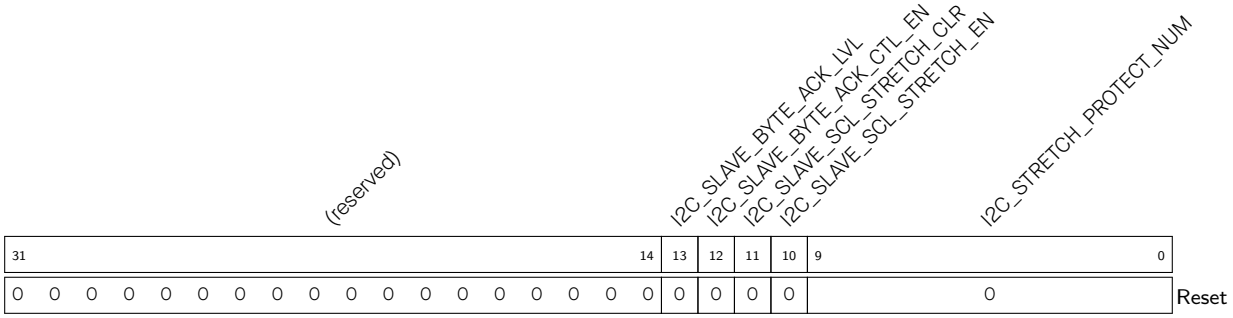
I2C_SCL_RST_SLV_EN Configures whether or not to send out SCL pulses when I2C master is IDLE. The number of pulses equals to I2C_SCL_RST_SLV_NUM[4:0]. (R/W/SC)

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
(R/W)

I2C_SCL_PD_EN Configures whether or not to power down the I2C output SCL line.
0: Not power down
1: Power down
Valid only when I2C_SCL_FORCE_OUT is 1. (R/W)

I2C_SDA_PD_EN Configures whether or not to power down the I2C output SDA line.
0: Not power down
1: Power down
Valid only when I2C_SDA_FORCE_OUT is 1. (R/W)

Register 30.17. 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
(R/W)

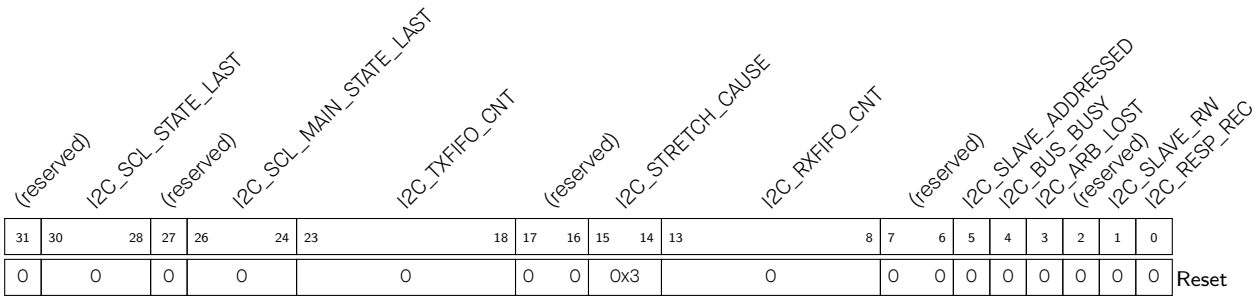
I2C_SLAVE_SCL_STRETCH_EN Configures whether to enable slave SCL stretch function. The SCL output line will be stretched low when I2C_SLAVE_SCL_STRETCH_EN is 1 and stretch event happens. The stretch cause can be seen in I2C_STRETCH_CAUSE.
0: Disable
1: Enable
(R/W)

I2C_SLAVE_SCL_STRETCH_CLR Configures whether or not to clear the I2C slave SCL stretch function.
0: No effect
1: Clear
(WT)

I2C_SLAVE_BYTE_ACK_CTL_EN Configures whether to enable the function for 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 enables.
0: Low level
1: High level
(R/W)

Register 30.18. 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 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. (RO)

I2C_TXFIFO_CNT Represents the number of data bytes to be sent. (RO)

Continued on the next page...

Register 30.18. I2C_SR_REG (0x0008)

Continued from the previous page...

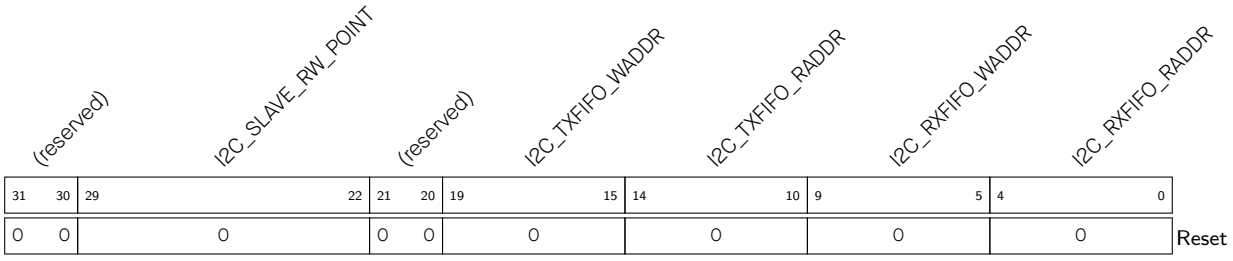
I2C_SCL_MAIN_STATE_LAST Represents the states of the I2C module state machine.

0: Idle
1: Address shift
2: ACK address
3: Rx data
4: Tx data
5: Send ACK
6: Wait ACK
7: Invalid
(RO)

I2C_SCL_STATE_LAST Represents the states of the state machine used to produce SCL.

0: Idle
1: Start
2: Negative edge
3: Low
4: Positive edge
5: High
6: Stop
7: Invalid
(RO)

Register 30.19. I2C_FIFO_ST_REG (0x0014)



I2C_RXFIFO_RADDR Represents the offset address of the APB reading from RX FIFO. (RO)

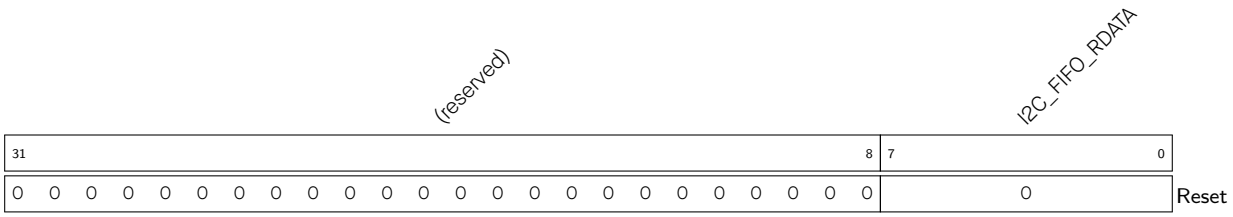
I2C_RXFIFO_WADDR Represents the offset address of i2c module receiving data and writing to RX FIFO. (RO)

I2C_TXFIFO_RADDR Represents the offset address of I2C module reading from TX FIFO. (RO)

I2C_TXFIFO_WADDR Represents the offset address of APB bus writing to TX FIFO. (RO)

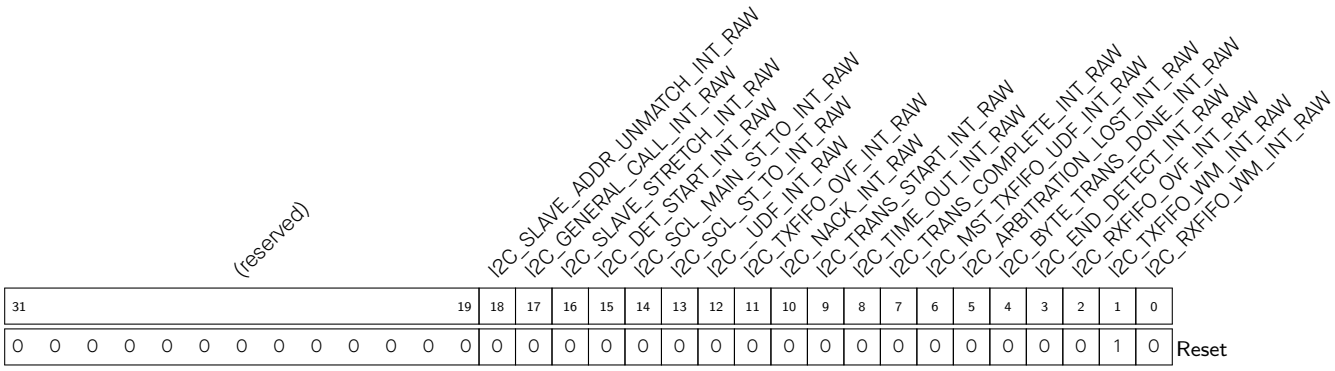
I2C_SLAVE_RW_POINT Represents the offset address in the I2C Slave RAM addressed by I2C Master when in I2C Slave mode. (RO)

Register 30.20. I2C_DATA_REG (0x001C)



I2C_FIFO_RDATA Represents the value of RX FIFO read data. (RO)

Register 30.21. I2C_INT_RAW_REG (0x0020)



I2C_RXFIFO_WM_INT_RAW The raw interrupt status of I2C_RXFIFO_WM_INT. (R/SS/WTC)

I2C_TXFIFO_WM_INT_RAW The raw interrupt status of I2C_TXFIFO_WM_INT. (R/SS/WTC)

I2C_RXFIFO_OVF_INT_RAW The raw interrupt status of I2C_RXFIFO_OVF_INT. (R/SS/WTC)

I2C_END_DETECT_INT_RAW The raw interrupt status of I2C_END_DETECT_INT. (R/SS/WTC)

I2C_BYTE_TRANS_DONE_INT_RAW The raw interrupt status of I2C_END_DETECT_INT. (R/SS/WTC)

I2C_ARBITRATION_LOST_INT_RAW The raw interrupt status of I2C_ARBITRATION_LOST_INT. (R/SS/WTC)

I2C_MST_TXFIFO_UDF_INT_RAW The raw interrupt status of I2C_TRANS_COMPLETE_INT. (R/SS/WTC)

I2C_TRANS_COMPLETE_INT_RAW The raw interrupt status of the I2C_TRANS_COMPLETE_INT. (R/SS/WTC)

Continued on the next page...

Register 30.21. I2C_INT_RAW_REG (0x0020)

Continued from the previous page...

I2C_TIME_OUT_INT_RAW The raw interrupt status of I2C_TIME_OUT_INT. (R/SS/WTC)

I2C_TRANS_START_INT_RAW The raw interrupt status of I2C_TRANS_START_INT. (R/SS/WTC)

I2C_NACK_INT_RAW The raw interrupt status of I2C_SLAVE_STRETCH_INT. (R/SS/WTC)

I2C_TXFIFO_OVF_INT_RAW The raw interrupt status of I2C_TXFIFO_OVF_INT. (R/SS/WTC)

I2C_RXFIFO_UDF_INT_RAW The raw interrupt status of I2C_RXFIFO_UDF_INT. (R/SS/WTC)

I2C_SCL_ST_TO_INT_RAW The raw interrupt status of I2C_SCL_ST_TO_INT. (R/SS/WTC)

I2C_SCL_MAIN_ST_TO_INT_RAW The raw interrupt status of I2C_SCL_MAIN_ST_TO_INT. (R/SS/WTC)

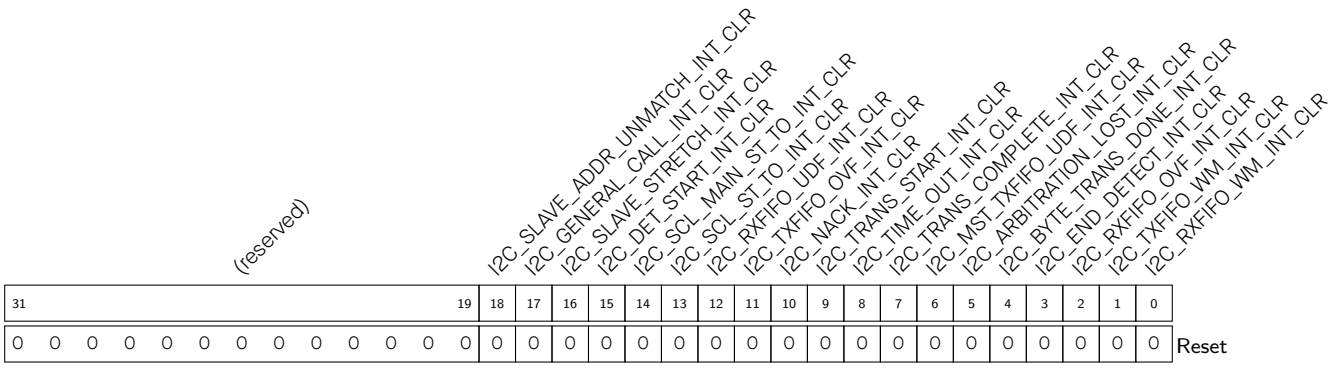
I2C_DET_START_INT_RAW The raw interrupt status of I2C_DET_START_INT. (R/SS/WTC)

I2C_SLAVE_STRETCH_INT_RAW The raw interrupt status of I2C_SLAVE_STRETCH_INT. (R/SS/WTC)

I2C_GENERAL_CALL_INT_RAW The raw interrupt status of I2C_GENARAL_CALL_INT. (R/SS/WTC)

I2C_SLAVE_ADDR_UNMATCH_INT_RAW The raw interrupt status of I2C_SLAVE_ADDR_UNMATCH_INT. (R/SS/WTC)

Register 30.22. I2C_INT_CLR_REG (0x0024)



I2C_RXFIFO_WM_INT_CLR Write 1 to clear I2C_RXFIFO_WM_INT. (WT)

I2C_TXFIFO_WM_INT_CLR Write 1 to clear I2C_TXFIFO_WM_INT. (WT)

I2C_RXFIFO_OVF_INT_CLR Write 1 to clear I2C_RXFIFO_OVF_INT. (WT)

I2C_END_DETECT_INT_CLR Write 1 to clear I2C_END_DETECT_INT. (WT)

I2C_BYTE_TRANS_DONE_INT_CLR Write 1 to clear I2C_BYTE_TRANS_DONE_INT. (WT)

I2C_ARBITRATION_LOST_INT_CLR Write 1 to clear I2C_ARBITRATION_LOST_INT. (WT)

I2C_MST_TXFIFO_UDF_INT_CLR Write 1 to clear I2C_MST_TXFIFO_UDF_INT. (WT)

I2C_TRANS_COMPLETE_INT_CLR Write 1 to clear I2C_TRANS_COMPLETE_INT. (WT)

I2C_TIME_OUT_INT_CLR Write 1 to clear I2C_TIME_OUT_INT. (WT)

I2C_TRANS_START_INT_CLR Write 1 to clear I2C_TRANS_START_INT. (WT)

I2C_NACK_INT_CLR Write 1 to clear I2C_NACK_INT. (WT)

I2C_TXFIFO_OVF_INT_CLR Write 1 to clear I2C_TXFIFO_OVF_INT. (WT)

I2C_RXFIFO_UDF_INT_CLR Write 1 to clear I2C_RXFIFO_UDF_INT. (WT)

I2C_SCL_ST_TO_INT_CLR Write 1 to clear I2C_SCL_ST_TO_INT. (WT)

I2C_SCL_MAIN_ST_TO_INT_CLR Write 1 to clear I2C_SCL_MAIN_ST_TO_INT. (WT)

I2C_DET_START_INT_CLR Write 1 to clear I2C_DET_START_INT. (WT)

I2C_SLAVE_STRETCH_INT_CLR Write 1 to clear I2C_SLAVE_STRETCH_INT. (WT)

I2C_GENERAL_CALL_INT_CLR Write 1 to clear I2C_GENARAL_CALL_INT. (WT)

I2C_SLAVE_ADDR_UNMATCH_INT_CLR Write 1 to clear I2C_SLAVE_ADDR_UNMATCH_INT. (WT)

Register 30.23. 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	
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	Reset

I2C_RXFIFO_WM_INT_ENA Write 1 to enable I2C_RXFIFO_WM_INT. (R/W)

I2C_TXFIFO_WM_INT_ENA Write 1 to enable I2C_TXFIFO_WM_INT. (R/W)

I2C_RXFIFO_OVF_INT_ENA Write 1 to enable I2C_RXFIFO_OVF_INT. (R/W)

I2C_END_DETECT_INT_ENA Write 1 to enable the I2C_END_DETECT_INT. (R/W)

I2C_BYTE_TRANS_DONE_INT_ENA Write 1 to enable I2C_END_DETECT_INT. (R/W)

I2C_ARBITRATION_LOST_INT_ENA Write 1 to enable I2C_ARBITRATION_LOST_INT. (R/W)

I2C_MST_TXFIFO_UDF_INT_ENA Write 1 to enable I2C_TRANS_COMPLETE_INT. (R/W)

I2C_TRANS_COMPLETE_INT_ENA Write 1 to enable I2C_TRANS_COMPLETE_INT. (R/W)

I2C_TIME_OUT_INT_ENA Write 1 to enable I2C_TIME_OUT_INT. (R/W)

I2C_TRANS_START_INT_ENA Write 1 to enable I2C_TRANS_START_INT. (R/W)

I2C_NACK_INT_ENA Write 1 to enable I2C_SLAVE_STRETCH_INT. (R/W)

I2C_TXFIFO_OVF_INT_ENA Write 1 to enable I2C_TXFIFO_OVF_INT. (R/W)

I2C_RXFIFO_UDF_INT_ENA Write 1 to enable I2C_RXFIFO_UDF_INT . (R/W)

I2C_SCL_ST_TO_INT_ENA Write 1 to enable I2C_SCL_ST_TO_INT. (R/W)

I2C_SCL_MAIN_ST_TO_INT_ENA Write 1 to enable I2C_SCL_MAIN_ST_TO_INT. (R/W)

I2C_DET_START_INT_ENA Write 1 to enable I2C_DET_START_INT. (R/W)

I2C_SLAVE_STRETCH_INT_ENA Write 1 to enable I2C_SLAVE_STRETCH_INT. (R/W)

I2C_GENERAL_CALL_INT_ENA Write 1 to enable I2C_GENARAL_CALL_INT. (R/W)

I2C_SLAVE_ADDR_UNMATCH_INT_ENA Write 1 to enable I2C_SLAVE_ADDR_UNMATCH_INT. (R/W)

Register 30.24. I2C_INT_STATUS_REG (0x002C)

Bits	Field	Reset
31:19	(reserved)	0 0 0 0 0 0 0 0 0 0 0 0 0
18	I2C_SLAVE_ADDR_UNMATCH_INT_ST	0
17	I2C_GENERAL_CALL_INT_ST	0
16	I2C_SLAVE_STRETCH_INT_ST	0
15	I2C_DET_START_INT_ST	0
14	I2C_SCL_MAIN_ST_TO_INT_ST	0
13	I2C_SCL_ST_TO_INT_ST	0
12	I2C_RXFIFO_UDF_INT_ST	0
11	I2C_TXFIFO_OVF_INT_ST	0
10	I2C_NACK_INT_ST	0
9	I2C_TRANS_START_INT_ST	0
8	I2C_TIME_OUT_INT_ST	0
7	I2C_TRANS_COMPLETE_INT_ST	0
6	I2C_MST_TXFIFO_UDF_INT_ST	0
5	I2C_ARBITRATION_LOST_INT_ST	0
4	I2C_BYTE_TRANS_DONE_INT_ST	0
3	I2C_END_DETECT_INT_ST	0
2	I2C_RXFIFO_OVF_INT_ST	0
1	I2C_TXFIFO_WM_INT_ST	0
0	I2C_RXFIFO_WM_INT_ST	0

I2C_RXFIFO_WM_INT_ST The masked interrupt status of I2C_RXFIFO_WM_INT. (RO)

I2C_TXFIFO_WM_INT_ST The masked interrupt status of I2C_TXFIFO_WM_INT. (RO)

I2C_RXFIFO_OVF_INT_ST The masked interrupt status of I2C_RXFIFO_OVF_INT. (RO)

I2C_END_DETECT_INT_ST The masked interrupt status of I2C_END_DETECT_INT. (RO)

I2C_BYTE_TRANS_DONE_INT_ST The masked interrupt status of I2C_END_DETECT_INT. (RO)

I2C_ARBITRATION_LOST_INT_ST The masked interrupt status of I2C_ARBITRATION_LOST_INT. (RO)

I2C_MST_TXFIFO_UDF_INT_ST The masked interrupt status of I2C_TRANS_COMPLETE_INT. (RO)

I2C_TRANS_COMPLETE_INT_ST The masked interrupt status of I2C_TRANS_COMPLETE_INT. (RO)

I2C_TIME_OUT_INT_ST The masked interrupt status of I2C_TIME_OUT_INT. (RO)

I2C_TRANS_START_INT_ST The masked interrupt status of I2C_TRANS_START_INT. (RO)

I2C_NACK_INT_ST The masked interrupt status of I2C_SLAVE_STRETCH_INT. (RO)

I2C_TXFIFO_OVF_INT_ST The masked interrupt status of I2C_TXFIFO_OVF_INT. (RO)

Continued on the next page...

Register 30.24. I2C_INT_STATUS_REG (0x002C)

Continued from the previous page...

I2C_RXFIFO_UDF_INT_ST The masked interrupt status of I2C_RXFIFO_UDF_INT. (RO)

I2C_SCL_ST_TO_INT_ST The masked interrupt status of I2C_SCL_ST_TO_INT. (RO)

I2C_SCL_MAIN_ST_TO_INT_ST The masked interrupt status of I2C_SCL_MAIN_ST_TO_INT. (RO)

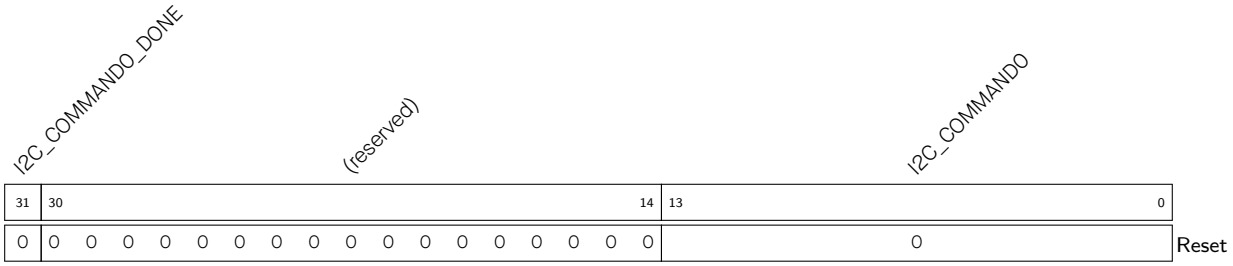
I2C_DET_START_INT_ST The masked interrupt status of I2C_DET_START_INT. (RO)

I2C_SLAVE_STRETCH_INT_ST The masked interrupt status of I2C_SLAVE_STRETCH_INT. (RO)

I2C_GENERAL_CALL_INT_ST The masked interrupt status of I2C_GENARAL_CALL_INT. (RO)

I2C_SLAVE_ADDR_UNMATCH_INT_ST The masked interrupt status of I2C_SLAVE_ADDR_UNMATCH_INT. (RO)

Register 30.25. 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 30.4-2 for more information.

(R/W)

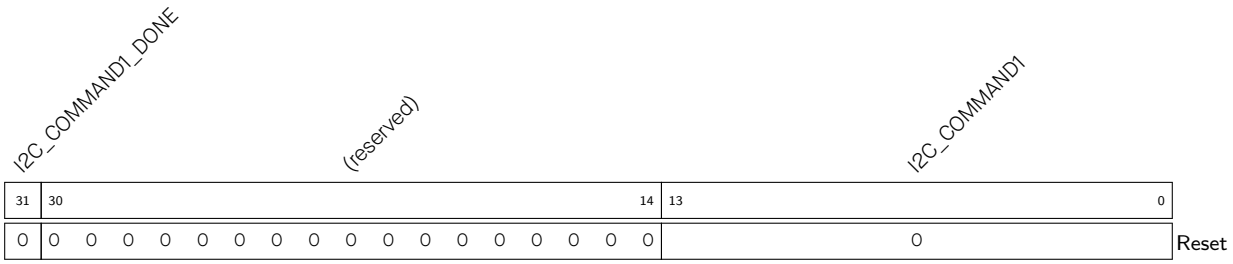
I2C_COMMAND0_DONE Represents whether command 0 is done in I2C Master mode.

0: Not done

1: Done

(R/W/SS)

Register 30.26. I2C_COMD1_REG (0x005C)



I2C_COMMAND1 Configures command 1. See details in I2C_COMD0_REG. (R/W)

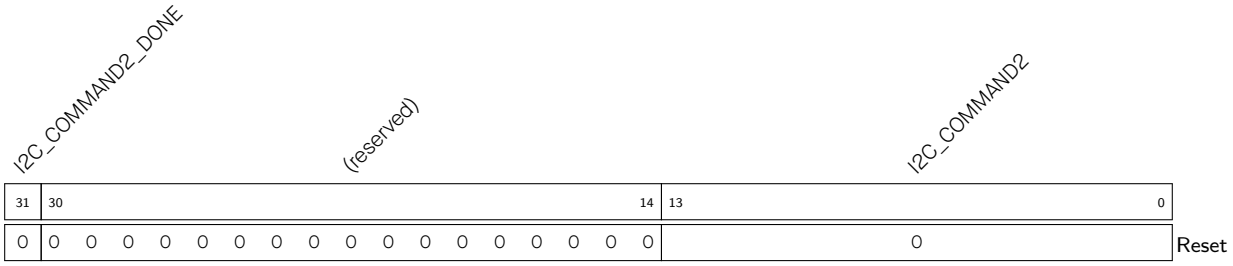
I2C_COMMAND1_DONE Represents whether command 1 is done in I2C Master mode.

0: Not done

1: Done

(R/W/SS)

Register 30.27. I2C_COMD2_REG (0x0060)



I2C_COMMAND2 Configures command 2. See details in I2C_COMD0_REG. (R/W)

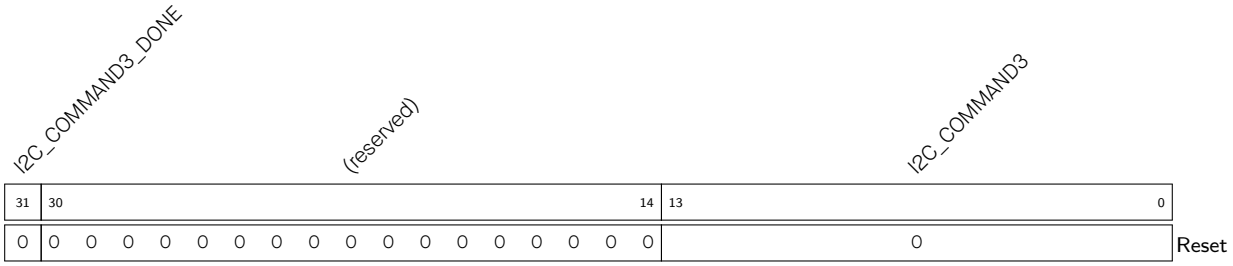
I2C_COMMAND2_DONE Represents whether command 2 is done in I2C Master mode.

0: Not done

1: Done

(R/W/SS)

Register 30.28. I2C_COMD3_REG (0x0064)



I2C_COMMAND3 Configures command 3. See details in I2C_COMD0_REG. (R/W)

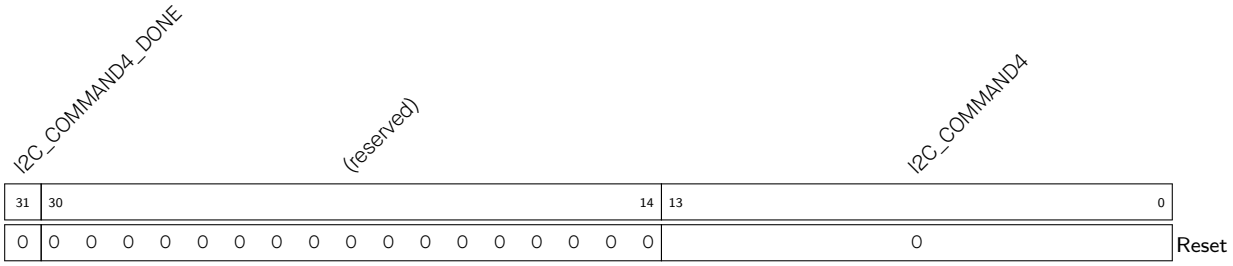
I2C_COMMAND3_DONE Represents whether command 3 is done in I2C Master mode.

0: Not done

1: Done

(R/W/SS)

Register 30.29. I2C_COMD4_REG (0x0068)



I2C_COMMAND4 Configures command 4. See details in I2C_COMD0_REG. (R/W)

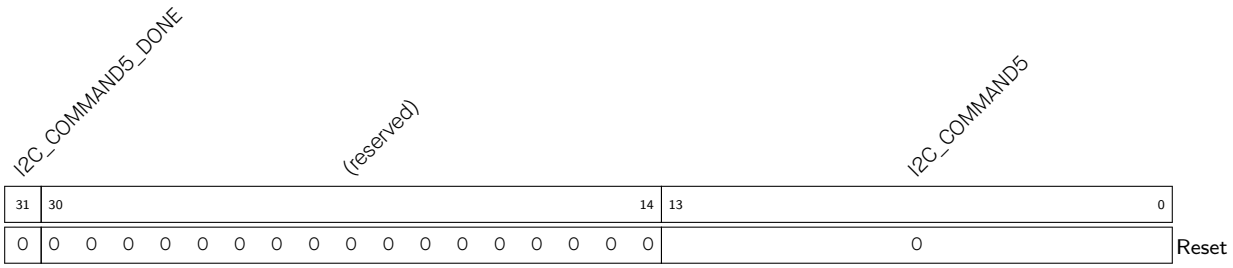
I2C_COMMAND4_DONE Represents whether command 4 is done in I2C Master mode.

0: Not done

1: Done

(R/W/SS)

Register 30.30. I2C_COMD5_REG (0x006C)



I2C_COMMAND5 Configures command 5. See details in I2C_COMD0_REG. (R/W)

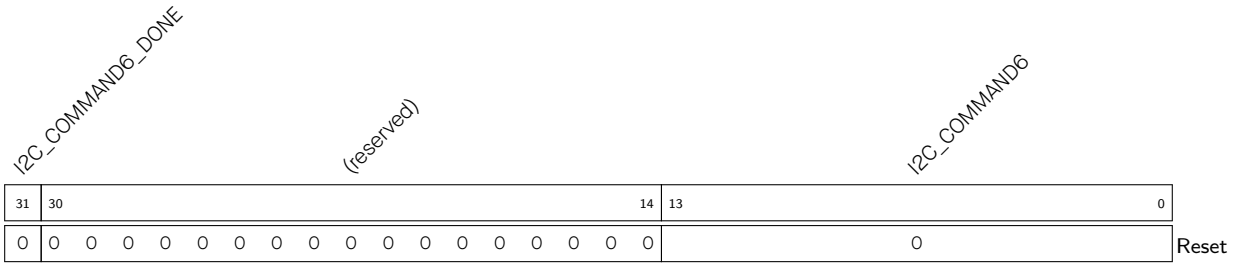
I2C_COMMAND5_DONE Represents whether command 5 is done in I2C Master mode.

0: Not done

1: Done

(R/W/SS)

Register 30.31. I2C_COMD6_REG (0x0070)



I2C_COMMAND6 Configures command 6. See details in I2C_COMD0_REG. (R/W)

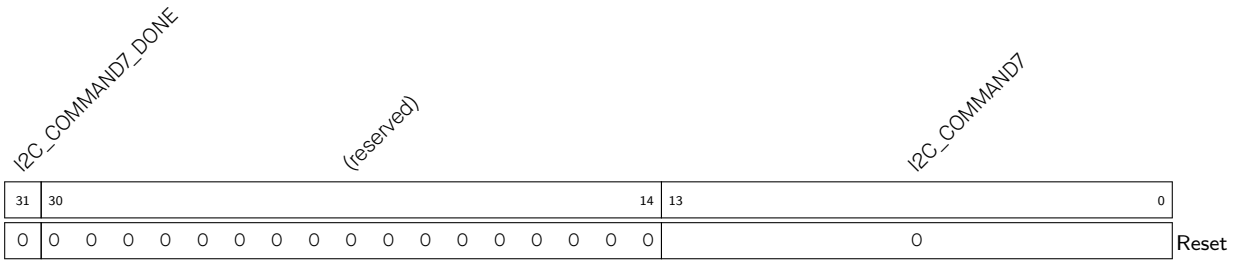
I2C_COMMAND6_DONE Represents whether command 6 is done in I2C Master mode.

0: Not done

1: Done

(R/W/SS)

Register 30.32. I2C_COMD7_REG (0x0074)



I2C_COMMAND7 Configures command 7. See details in I2C_COMD0_REG. (R/W)

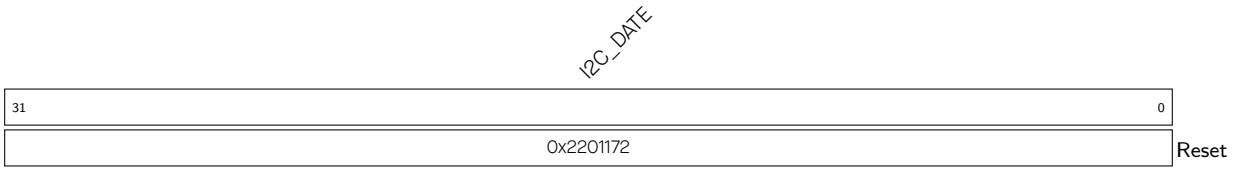
I2C_COMMAND7_DONE Represents whether command 7 is done in I2C Master mode.

0: Not done

1: Done

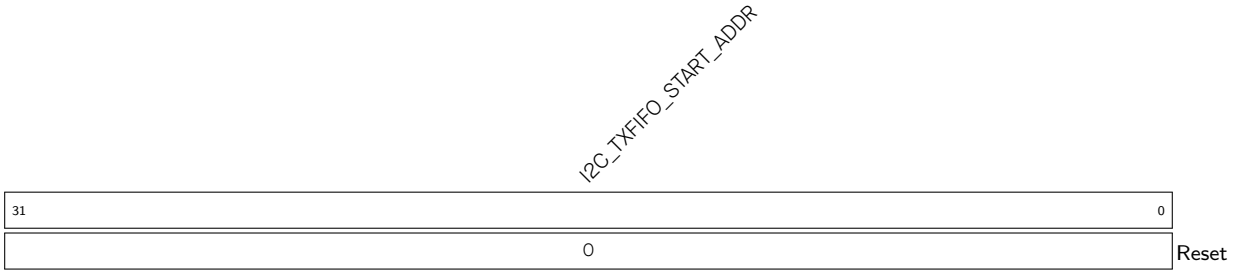
(R/W/SS)

Register 30.33. I2C_DATE_REG (0x00F8)



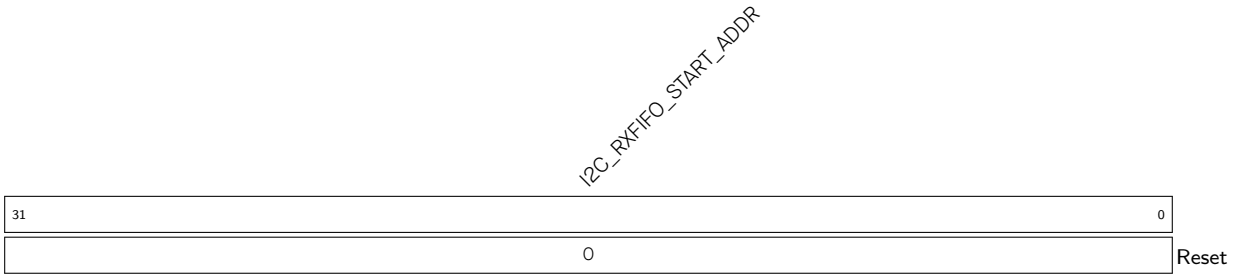
I2C_DATE Version control register. (R/W)

Register 30.34. I2C_TXFIFO_START_ADDR_REG (0x0100)



I2C_TXFIFO_START_ADDR Represents the I2C TX FIFO first address. (HRO)

Register 30.35. I2C_RXFIFO_START_ADDR_REG (0x0180)



I2C_RXFIFO_START_ADDR Represents the I2C RX FIFO first address. (HRO)

Chapter 31

I2S Controller (I2S)

31.1 Overview

ESP32-H2 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-H2 provides separate transmit (TX) and receive (RX) units for high performance.

31.2 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 μ-law A-law and μ-law are compression/decompression algorithms in digital pulse code modulated (PCM) non-uniform quantization, which can effectively improve the signal-to-quantization noise ratio.

TDM RX mode In this mode, pulse code modulated (PCM) data is received utilizing time division multiplexing (TDM). The signal lines include BCK, WS, and SD. Data from 16 channels at most can be received. TDM Philips standard, TDM MSB alignment standard, and TDM PCM standard are supported in this mode, depending on user configuration.

PDM RX mode In this mode, pulse density modulation (PDM) data is received. Used signals include WS and DATA. PDM standard is supported in this mode by user configuration.

TDM TX mode In this mode, pulse code modulated (PCM) data is sent in a way of time division multiplexing (TDM). The signal lines include BCK, WS, and DATA. Data up to 16 channels can be sent. TDM Philips standard, TDM MSB alignment standard, and TDM PCM standard are supported in this mode, depending on user configuration.

PDM TX mode In this mode, pulse density modulation (PDM) data is sent. The signal lines include WS and DATA. PDM standard is supported in this mode by user configuration.

PCM-to-PDM Data Format Converter 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.

31.3 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 96 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-H2 I2S is limited by the data bit width and the number of channels. For example, sampling frequencies up to 187.5 kHz (such as 8 kHz, 16 kHz, 32 kHz, 44.1 kHz, 48 kHz, 88.2 kHz, 96 kHz, 128 kHz) are supported in 32-bit two-channel sampling. Please refer to Section 31.6 for detailed information.

31.4 System Architecture

Figure 31.4-1 shows the structure of the ESP32-H2 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

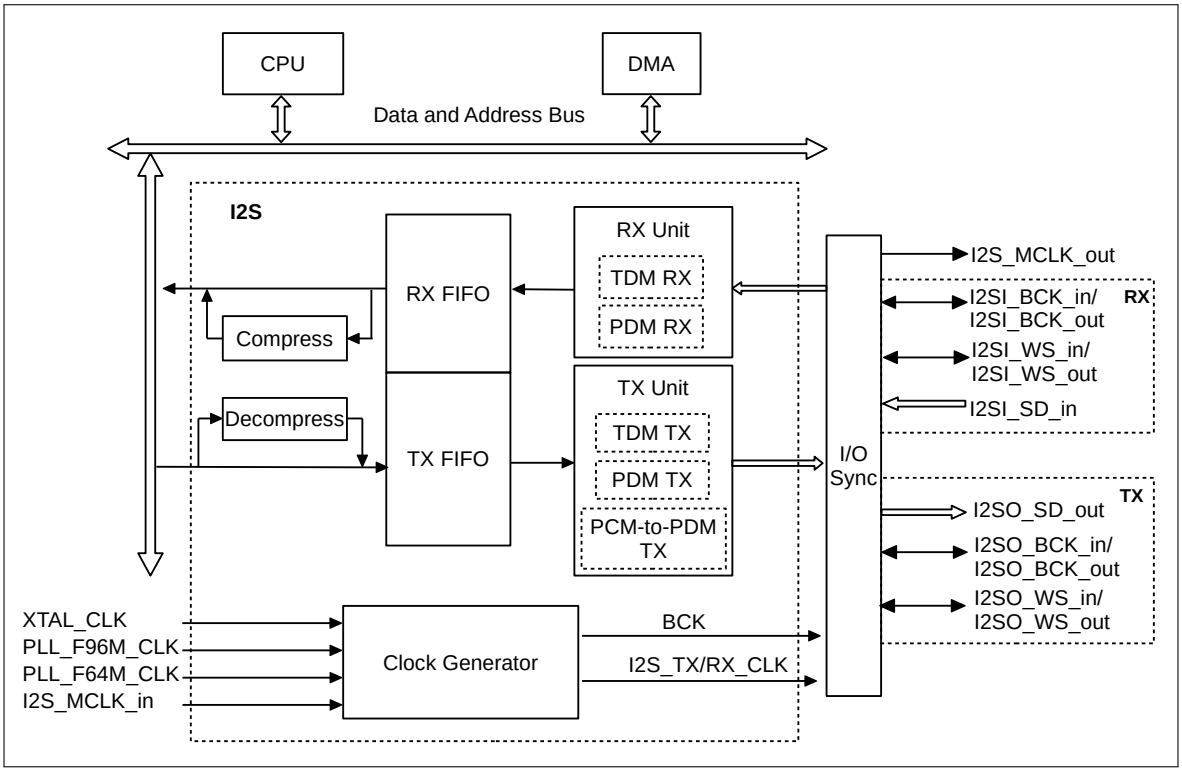


Figure 31.4-1. ESP32-H2 I2S System Diagram

The I2S module supports direct memory access (DMA) to internal memory. For more information, see Chapter 3 *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 31.4-1. 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 31.4-1 provides a detailed description of I2S signals.

Table 31.4-1. 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 6 *IO MUX and GPIO Matrix (GPIO, IO MUX)*.

31.5 Supported Audio Standards

ESP32-H2 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-H2 I2S will transmit data incorrectly in a mode that is neither TDM nor PDM.

31.5.1 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 31.5-1.

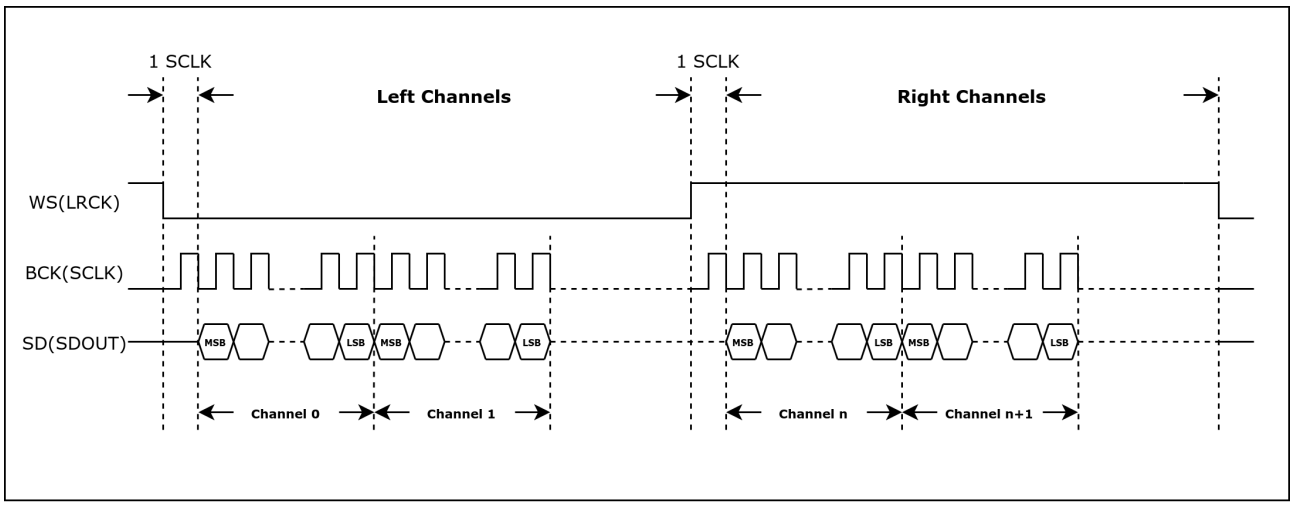


Figure 31.5-1. TDM Philips Standard Timing Diagram

31.5.2 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 31.5-2.

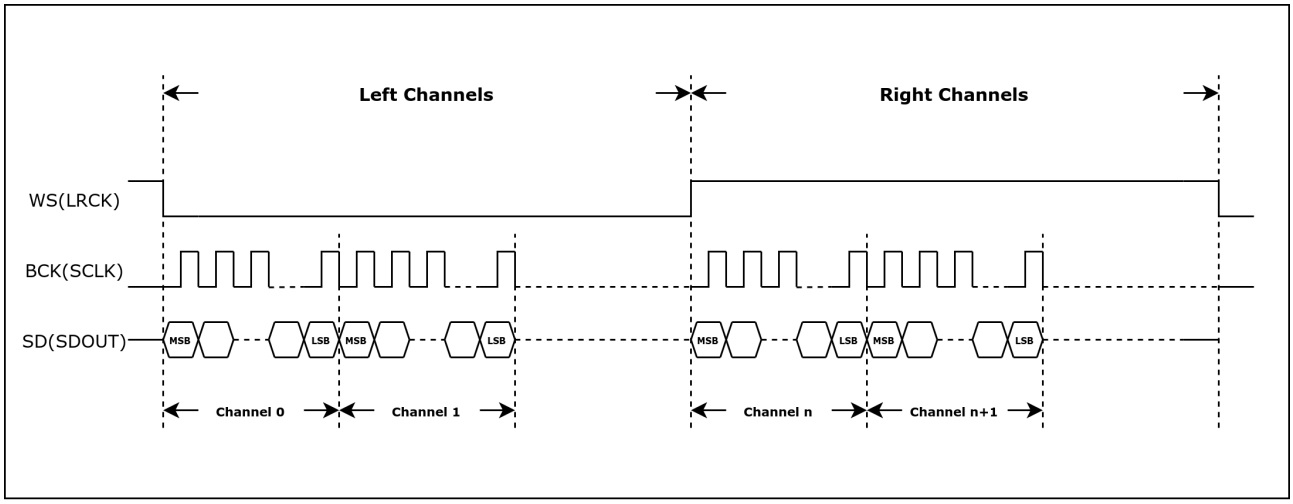


Figure 31.5-2. TDM MSB Alignment Standard Timing Diagram

31.5.3 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 31.5-3.

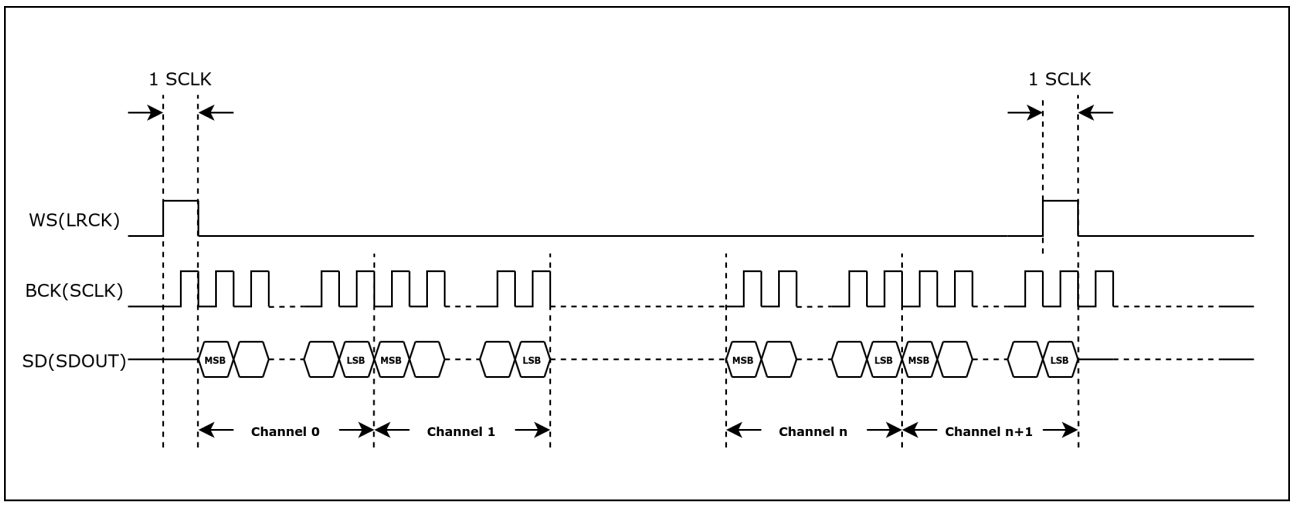


Figure 31.5-3. TDM PCM Standard Timing Diagram

31.5.4 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 31.5-4.

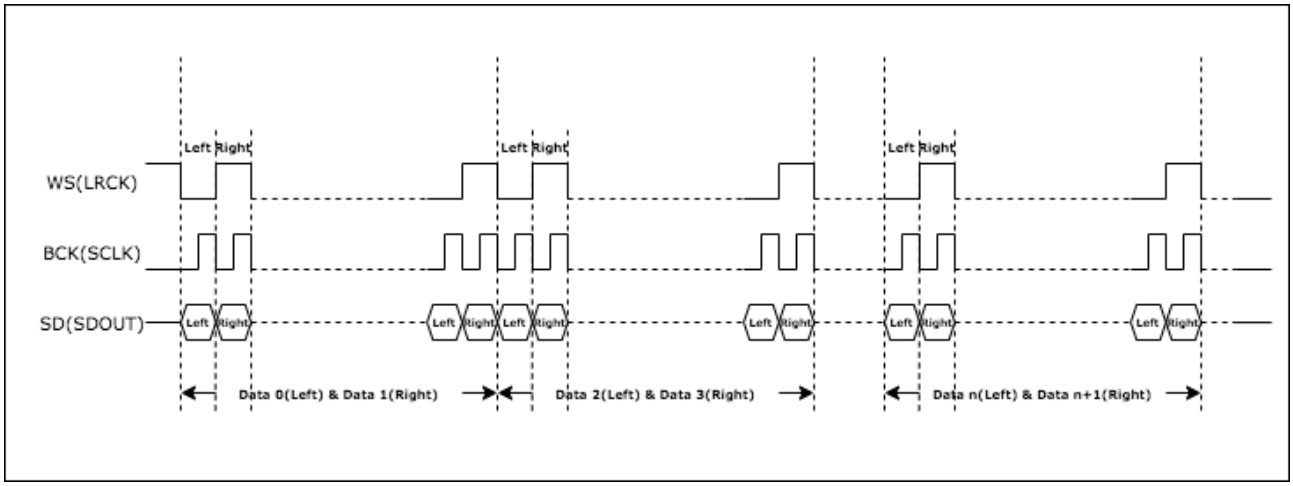


Figure 31.5-4. PDM Standard Timing Diagram

31.6 I2S TX/RX Clock

I2S_TX/RX_CLK is the master clock of I2S TX/RX unit, divided from:

- 40 MHz XTAL_CLK
- 96 MHz PLL_F96M_CLK
- 64 MHz PLL_F64M_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 31.6-1.

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.

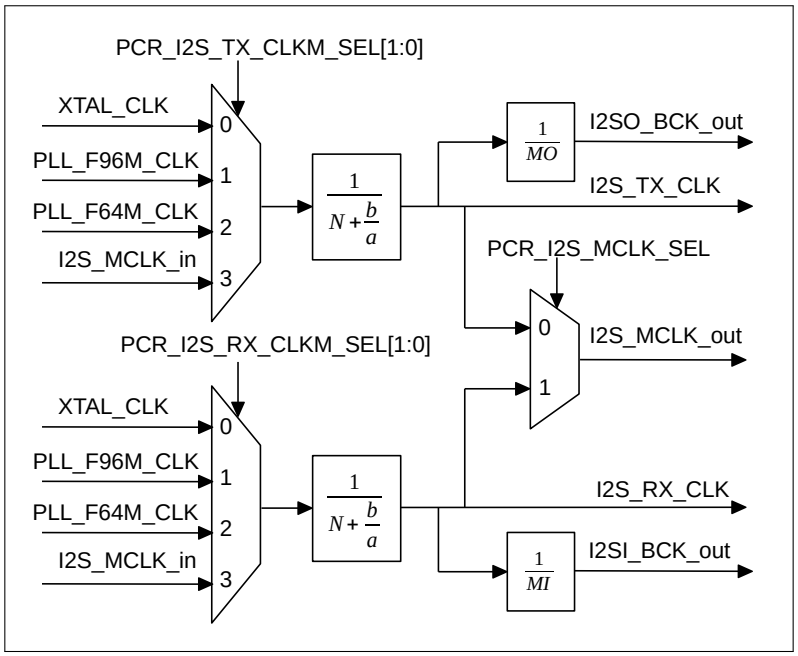


Figure 31.6-1. I2S Clock Generator

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_{BCK}. The I2S module can output I2S_MCLK_out as the master clock for peripherals.

31.7 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.

31.8 I2S Master/Slave Mode

The ESP32-H2 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

31.8.1 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.

31.8.2 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.

31.9 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 31.13.1.

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.

31.9.1 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.

31.9.1.1 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 31.9-1. 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).

31.9.1.2 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 31.9-2 shows how I2S_TX_BIG_ENDIAN controls the reading of the data with different valid data widths of the channel.

Table 31.9-2. 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.

31.9.1.3 A-law/μ-law Compression and Decompression

ESP32-H2 I2S compresses/decompresses the valid data into 32-bit by A-law or by μ-law. If the bit width of valid data is smaller than 32, zeros are filled to the extra high bits of the data to be compressed/decompressed by default.

Note:
Extra high bits here mean the bits[31: channel valid data width] of the data to be compressed/decompressed.

Configure I2S_TX_PCM_BYPASS:

- 0: Compress or decompress the data
- 1: Do not compress or decompress the data

Configure I2S_TX_PCM_CONF:

- 0: Decompress the data using A-law
- 1: Compress the data using A-law
- 2: Decompress the data using μ-law
- 3: Compress the data using μ-law

At this point, the first phase of data format control is completed.

31.9.1.4 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.

31.9.1.5 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 31.9-1 shows the complete process of TX data format control.

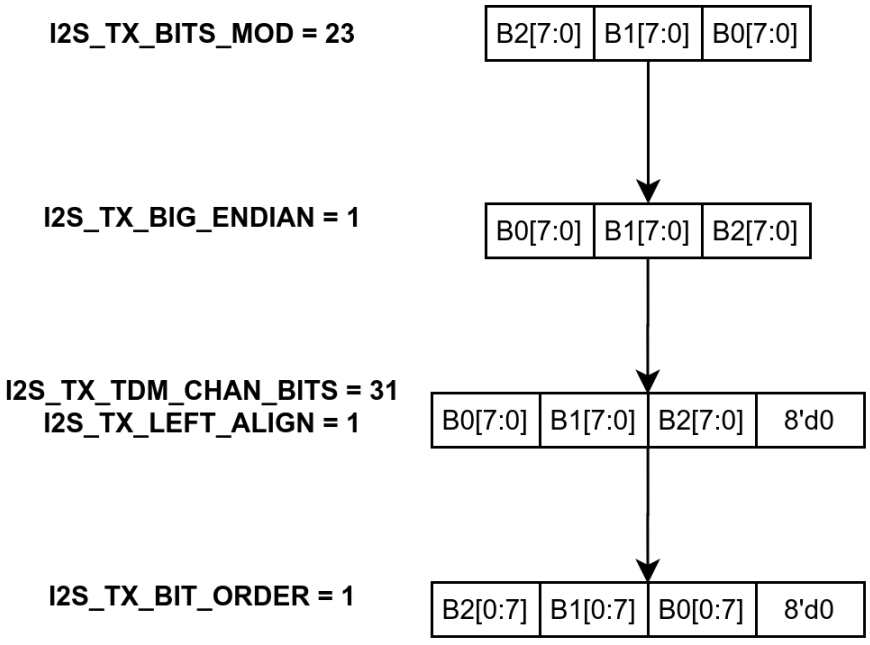


Figure 31.9-1. TX Data Format Control

31.9.2 Channel Mode Control

ESP32-H2 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.

31.9.2.1 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 31.9-2.

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.

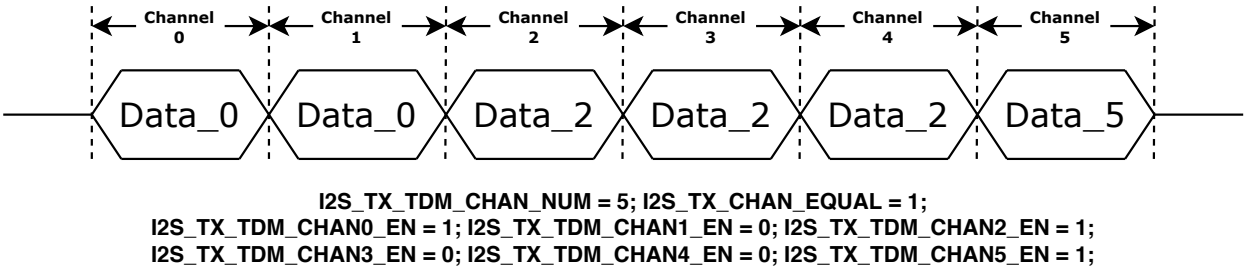


Figure 31.9-2. TDM Channel Control

31.9.2.2 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 31.9-3. Configure the two bits according to the data stored in memory, be it the single-channel or dual-channel data.

Table 31.9-3. 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 31.6 and Figure 31.6.

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 31.9-4. 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 31.9-5. 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 31.9.1.

Then the channel data is transmitted after channel mode control as follows.

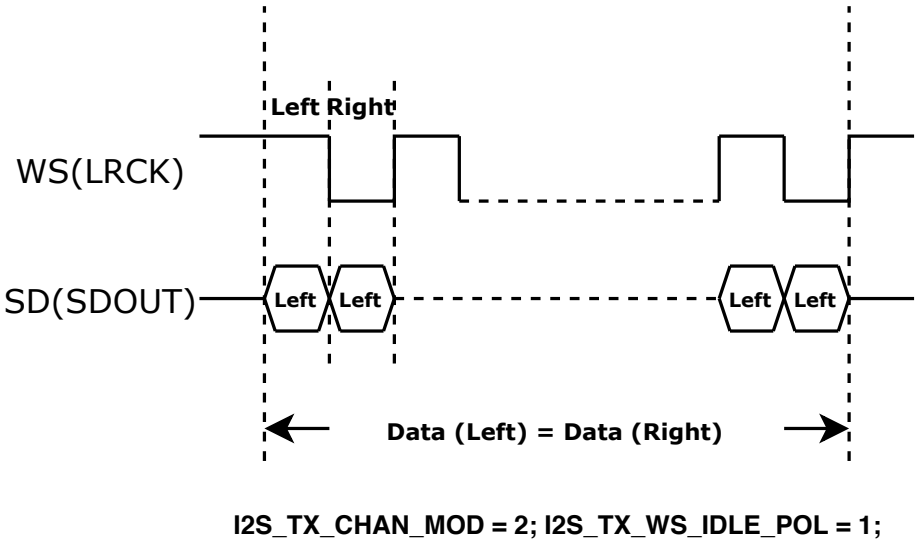


Figure 31.9-3. PDM Channel Control Example

31.10 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.

31.10.1 Channel Mode Control

ESP32-H2 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.

31.10.1.1 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.

31.10.1.2 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 31.6. Note, in PDM RX mode, the value of I2S_RX_HALF_SAMPLE_BITS must be same as that of I2S_RX_BITS_MOD.

31.10.2 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.

31.10.2.1 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.

31.10.2.2 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 31.10-1. 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

31.10.2.3 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.

31.10.2.4 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 31.10-2. 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

31.10.2.5 A-law/μ-law Compression and Decompression

ESP32-H2 I2S compresses/decompresses the storage data in 32-bit by A-law or by μ-law. By default, zeros are filled into high bits.

Configure I2S_RX_PCM_BYPASS:

- 0: Compress or decompress the data
- 1: Do not compress or decompress the data

Configure I2S_RX_PCM_CONF:

- 0: Decompress the data using A-law
- 1: Compress the data using A-law
- 2: Decompress the data using μ-law
- 3: Compress the data using μ-law

At this point, the data format control is completed. Data then is stored into memory via GDMA.

31.11 Event Task Matrix Feature

ESP32-H2 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 10 Event Task Matrix (SOC_ETM).

I2S can receive the following ETM tasks:

- I2S_TASK_START_TX: Enables I2S TX for data transfer.
- I2S_TASK_START_RX: Enables I2S RX for data transfer.
- I2S_TASK_STOP_TX: Stops I2S TX data transfer.
- I2S_TASK_STOP_RX: Stops I2S RX data transfer.

I2S can generate the following ETM events:

- I2S_EVT_TX_DONE: Indicates that I2S TX has completed data transmission. Triggered when all data in the TX FIFO has been sent.
- I2S_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.
- I2S_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.
- I2S_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 I2S_EVT_X_WORDS_SENT event can trigger the I2S_TASK_STOP_TX task, and in this way stop the I2S operation through ETM.

31.12 I2S Interrupts

ESP32-H2'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 9 *Interrupt Matrix (INTMTX)* > Section 9.2 *Interrupt Terminology in ESP32-H2*.

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 31.14 *Register Summary*.

31.13 Software Configuration Process

31.13.1 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 31.6.
2. Configure signal pins according to Table 31.4-1.
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 31.9, and then set I2S_TX_UPDATE.
5. Reset TX unit and TX FIFO as described in Section 31.7.
6. Enable corresponding interrupts. See Section 31.12.
7. Configure DMA outlink.
8. Set I2S_TX_STOP_EN if needed. For more information, please refer to Section 31.8.1.
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.

31.13.2 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 31.6.
2. Configure signal pins according to Table 31.4-1.
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 31.10, and then set I2S_RX_UPDATE.
5. Reset the RX unit and its FIFO according to Section 31.7.
6. Enable the corresponding interrupts. See Section 31.12.
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.

31.14 Register Summary

The addresses in this section are relative to the I2S Controller base address provided in Table 4.3-2 in Chapter *4 System and Memory*.

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

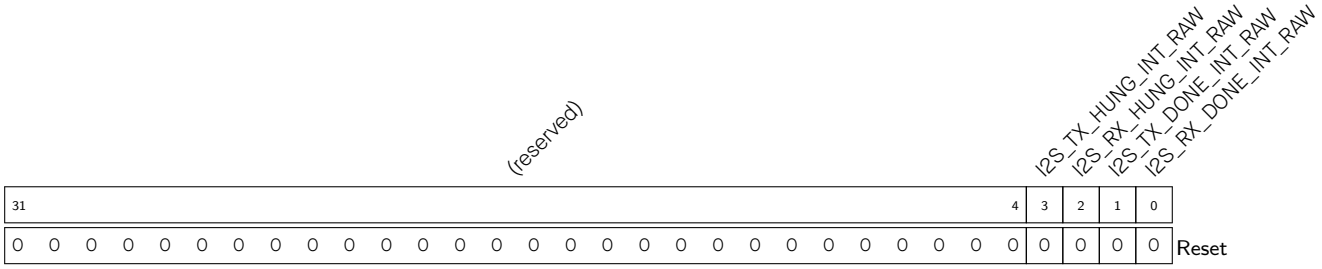
Name	Description	Address	Access
Interrupt registers			
I2S_INT_RAW_REG	I2S interrupt raw register	0x000C	R/SS/WTC
I2S_INT_ST_REG	I2S interrupt status register	0x0010	RO
I2S_INT_ENA_REG	I2S interrupt enable register	0x0014	R/W
I2S_INT_CLR_REG	I2S interrupt clear register	0x0018	WT
RX control and configuration 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

31.15 Registers

The addresses in this section are relative to the I2S Controller base address provided in Table 4.3-2 in Chapter *4 System and Memory*.

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

Register 31.1. I2S_INT_RAW_REG (0x000C)



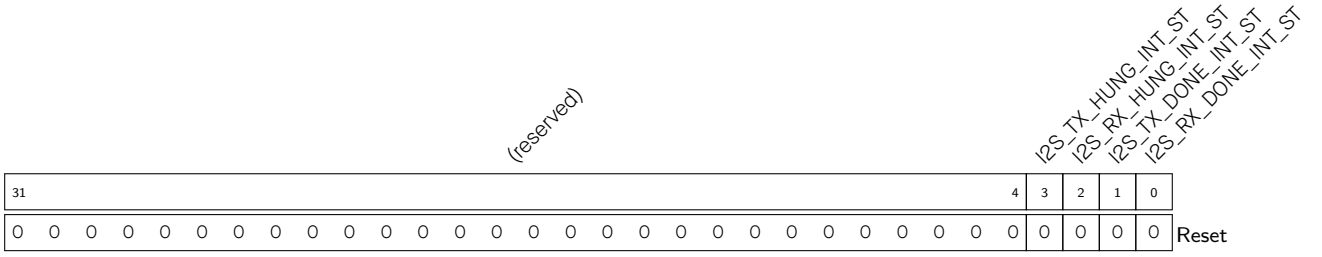
I2S_RX_DONE_INT_RAW The raw interrupt status of the I2S_RX_DONE_INT interrupt. (R/SS/WTC)

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 31.2. I2S_INT_ST_REG (0x0010)



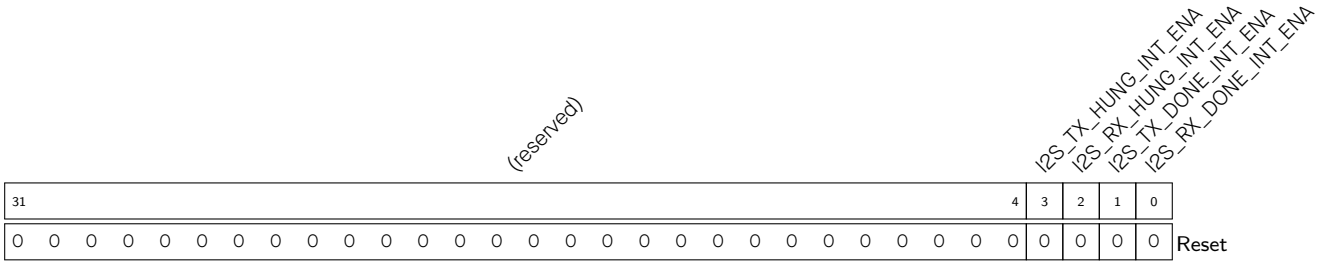
I2S_RX_DONE_INT_ST The masked interrupt status of the I2S_RX_DONE_INT interrupt. (RO)

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 31.3. I2S_INT_ENA_REG (0x0014)



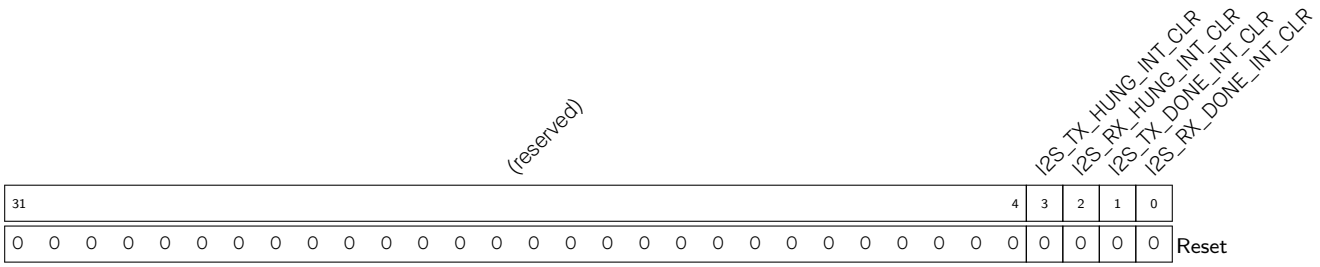
I2S_RX_DONE_INT_ENA Write 1 to enable the I2S_RX_DONE_INT interrupt. (R/W)

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 31.4. I2S_INT_CLR_REG (0x0018)



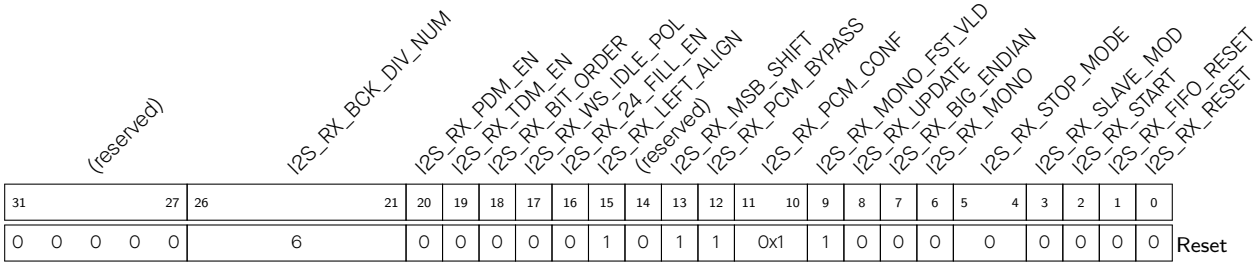
I2S_RX_DONE_INT_CLR Write 1 to clear the I2S_RX_DONE_INT interrupt. (WT)

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 31.5. 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 31.5. I2S_RX_CONF_REG (0x0020)

Continued from the previous page...

I2S_RX_UPDATE Configures whether to update I2S RX registers from APB clock domain to I2S RX clock domain. This bit will be cleared by hardware after the register update is done.
0: No effect
1: Update
(R/W/SC)

I2S_RX_MONO_FST_VLD Configures the valid data channel in I2S RX mono mode.
0: The second channel data valid
1: The first channel data valid
(R/W)

I2S_RX_PCM_CONF Configures I2S RX compress/decompress mode.
0 (atol): A-law decompress
1 (ltoa): A-law compress
2 (utol): μ-law decompress
3 (ltou): μ-law compress
(R/W)

I2S_RX_PCM_BYPASS Configures whether to bypass the Compress/Decompress units for received data.
0: No effect
1: Bypass
(R/W)

I2S_RX_MSB_SHIFT Configures the timing between the WS signal and the MSB of data.
0: Align at the rising edge
1: The WS signal changes one BCK clock earlier
(R/W)

I2S_RX_LEFT_ALIGN Configures I2S RX alignment mode.
0: Right alignment mode
1: Left alignment mode
(R/W)

Continued on the next page...

Register 31.5. 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.

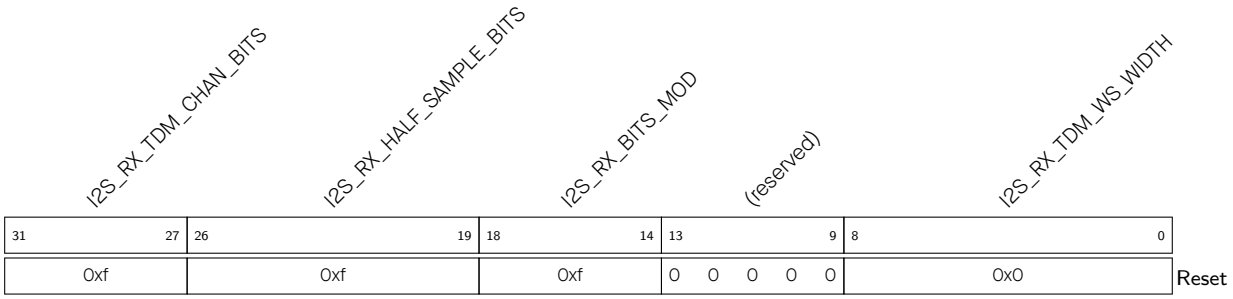
0: Disable

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 31.6. 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

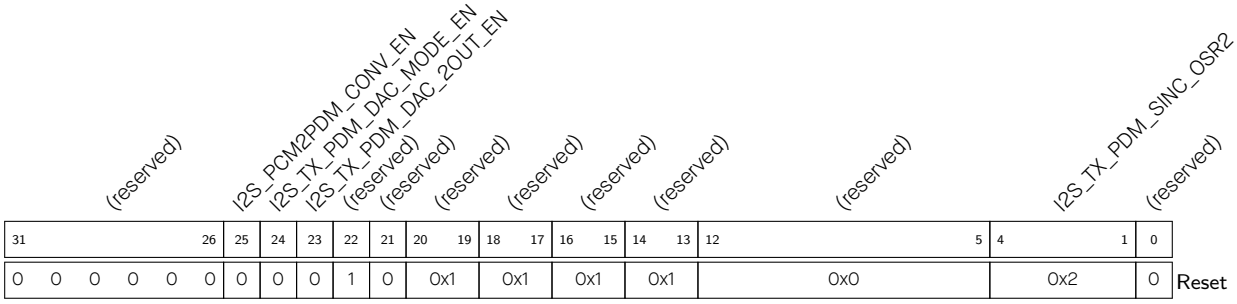
Other values are invalid.

(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 31.7. 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)

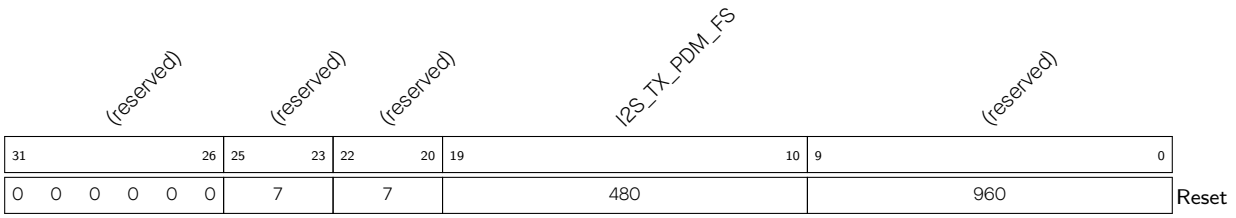
I2S_PCM2PDM_CONV_EN Configures whether to enable I2S TX PCM-to-PDM conversion.

0: Disable

1: Enable

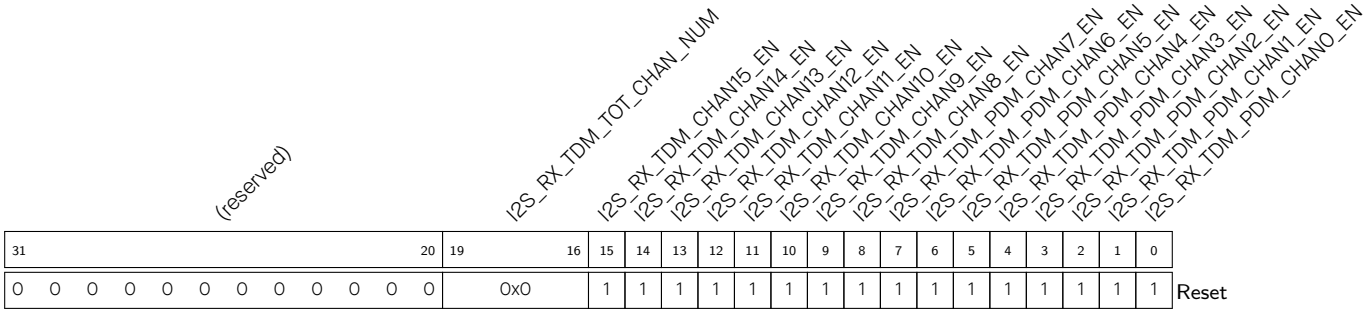
(R/W)

Register 31.8. I2S_TX_PCM2PDM_CONF1_REG (0x0044)



I2S_TX_PDM_FS Configures I2S PDM TX upsampling parameter. (R/W)

Register 31.9. 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*.

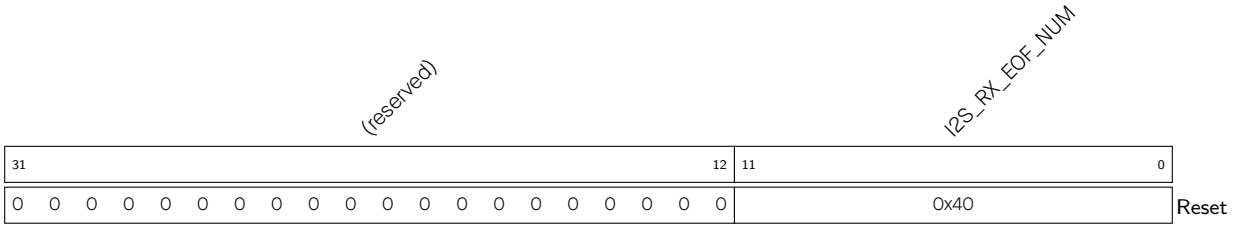
0: Disable. Channel *n* only inputs 0

1: Enable

(R/W)

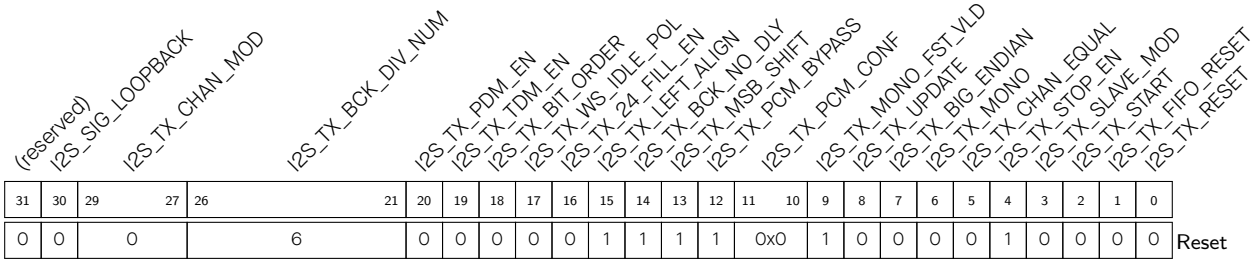
I2S_RX_TDM_TOT_CHAN_NUM Configures the total number of channels in use in I2S RX TDM mode. Total channel number in use = I2S_RX_TDM_TOT_CHAN_NUM + 1. (R/W)

Register 31.10. 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 31.11. 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 31.11. 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): μ-law decompress
3 (ltou): μ-law compress
(R/W)

I2S_TX_PCM_BYPASS Configures whether to bypass Compress/Decompress units for transmitted data.
0: No effect
1: Bypass
(R/W)

I2S_TX_MSB_SHIFT Configures the timing between the WS signal and the MSB of data.
0: Align at the rising edge
1: WS signal changes one BCK clock earlier
(R/W)

I2S_TX_BCK_NO_DLY Configures the source of the BCK rising and falling edges in master mode.
0: Rising and falling edges are constructed based on the BCK input from 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 31.11. 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)

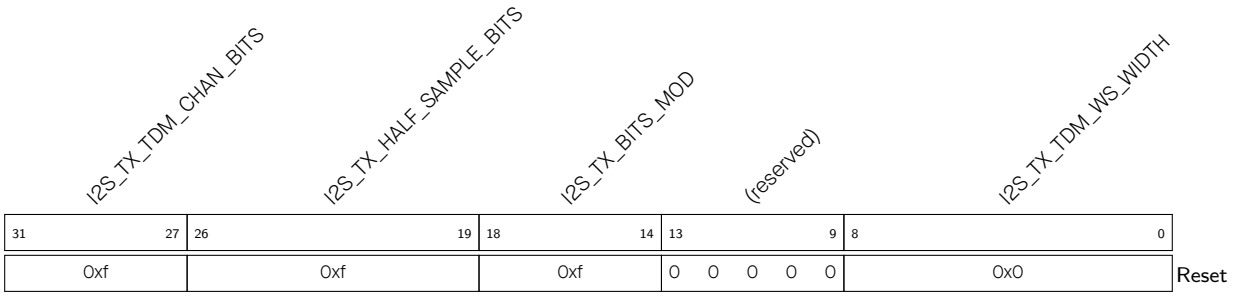
I2S_TX_PDM_EN Configures whether to enable I2S PDM TX mode.
0: Disable
1: Enable
(R/W)

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 31.9-4. (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 31.12. 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

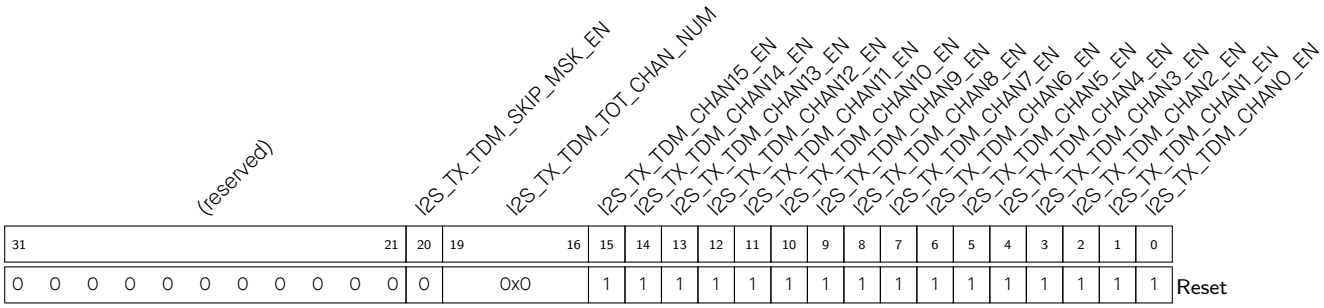
Other values are invalid.

(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 31.13. 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 31.9.2.1

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)

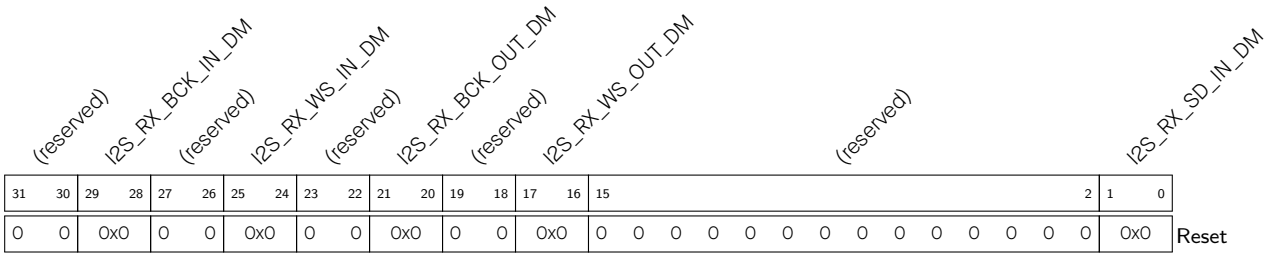
I2S_TX_TDM_SKIP_MSK_EN Configures the data to be sent in GDMA TX buffer.

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 31.14. 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)

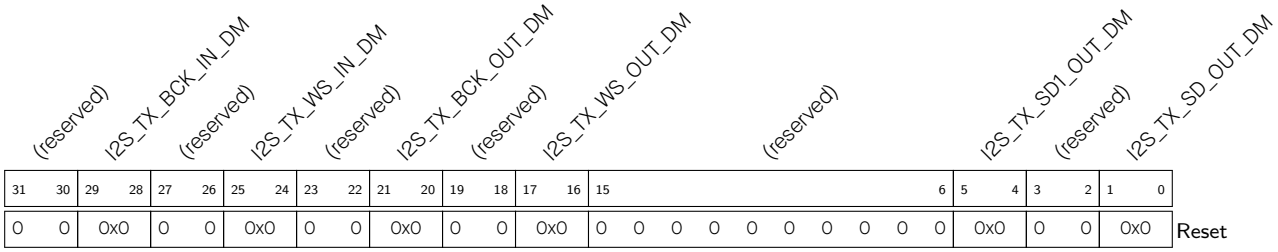
I2S_RX_WS_OUT_DM Configures the delay mode of I2S RX WS output signal. For detailed configuration values, please refer to I2S_RX_SD_IN_DM. (R/W)

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 31.15. 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)

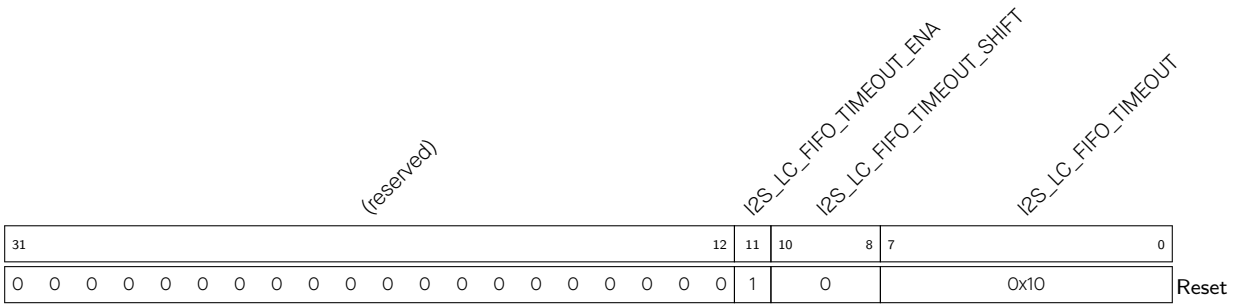
I2S_TX_WS_OUT_DM Configures the delay mode of I2S TX WS output signal. For detailed configuration values, please refer to I2S_TX_SD_OUT_DM. (R/W)

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 31.16. 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)

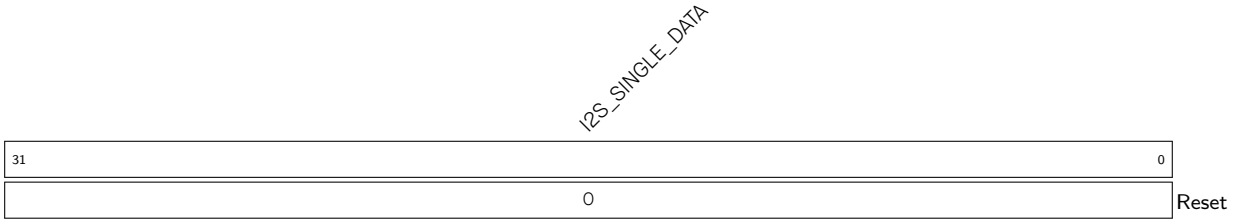
I2S_LC_FIFO_TIMEOUT_ENA Configures whether to enable FIFO timeout.

0: Disable

1: Enable

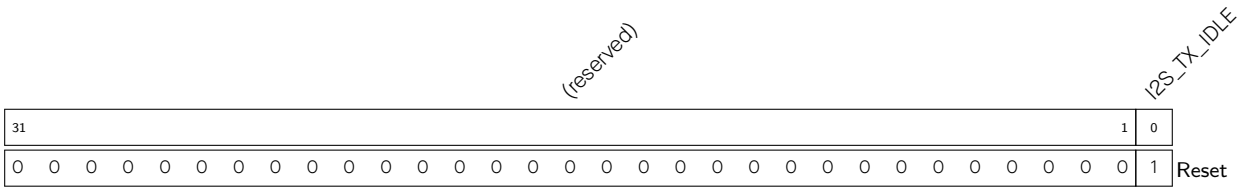
(R/W)

Register 31.17. I2S_CONF_SIGLE_DATA_REG (0x0068)



I2S_SINGLE_DATA Configures constant channel data to be sent out. (R/W)

Register 31.18. I2S_STATE_REG (0x006C)



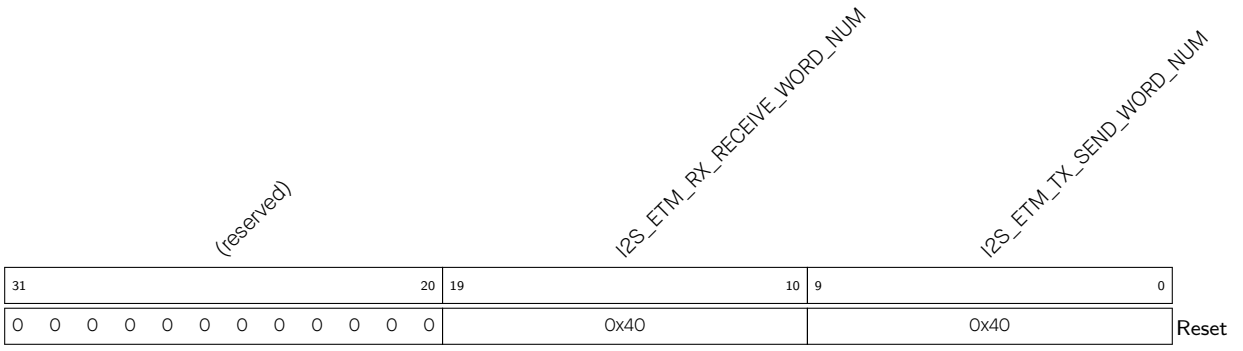
I2S_TX_IDLE Represents the TX unit state.

0: I2S TX unit is working

1: I2S TX unit is in idle state

(RO)

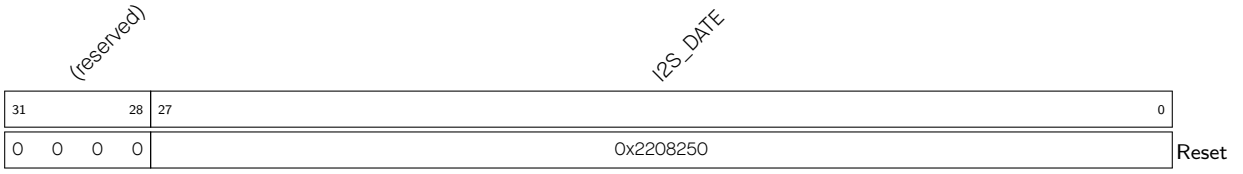
Register 31.19. I2S_ETM_CONF_REG (0x0070)



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 31.20. I2S_DATE_REG (0x0080)



I2S_DATE Version control register. (R/W)

Chapter 32

Pulse Count Controller (PCNT)

The pulse count controller (PCNT) is designed to count input pulses. It can increment or decrement a pulse counter value by keeping track of rising (positive) or falling (negative) edges of the input pulse signal. The PCNT has four independent pulse counters called units, which have their groups of registers. There is only one clock in PCNT, which is APB_CLK. In this chapter, *n* denotes the number of a unit from 0 ~ 3.

Each unit includes two channels (ch0 and ch1) which can independently increment or decrement its pulse counter value. The remainder of the chapter will mostly focus on channel 0 (ch0) as the functionality of the two channels is identical.

As shown in Figure 32.0-1, each channel has two input signals:

1. One input pulse signal (e.g., sig_ch0_u*n*, the input pulse signal for ch0 of unit *n* ch0)
2. One control signal (e.g., ctrl_ch0_u*n*, the control signal for ch0 of unit *n* ch0)

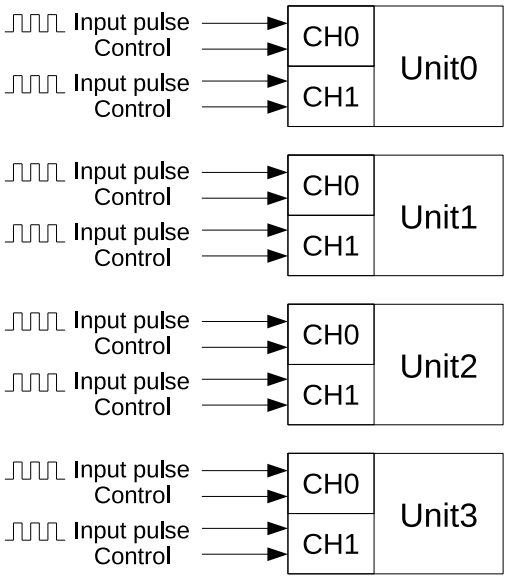


Figure 32.0-1. PCNT Block Diagram

32.1 Features

A PCNT has the following features:

- Four independent pulse counters (units) that count from 1 to 65535
- Each unit consists of two independent channels sharing one pulse counter
- All channels have input pulse signals (e.g., sig_ch0_u*n*) with their corresponding control signals (e.g., ctrl_ch0_u*n*)

- Independently filter glitches of input pulse signals (sig_ch0_u*n* and sig_ch1_u*n*) and control signals (ctrl_ch0_u*n* and ctrl_ch1_u*n*) on each unit
- Each channel has the following parameters:
  1. Selection between counting on positive or negative edges of the input pulse signal
  2. Configuration to Increment, Decrement, or Disable counter mode for control signal's high and low states
  3. Five counter watchpoints
- Maximum frequency of pulses: $\frac{f_{APB\_CLK}}{2}$

32.2 Functional Description

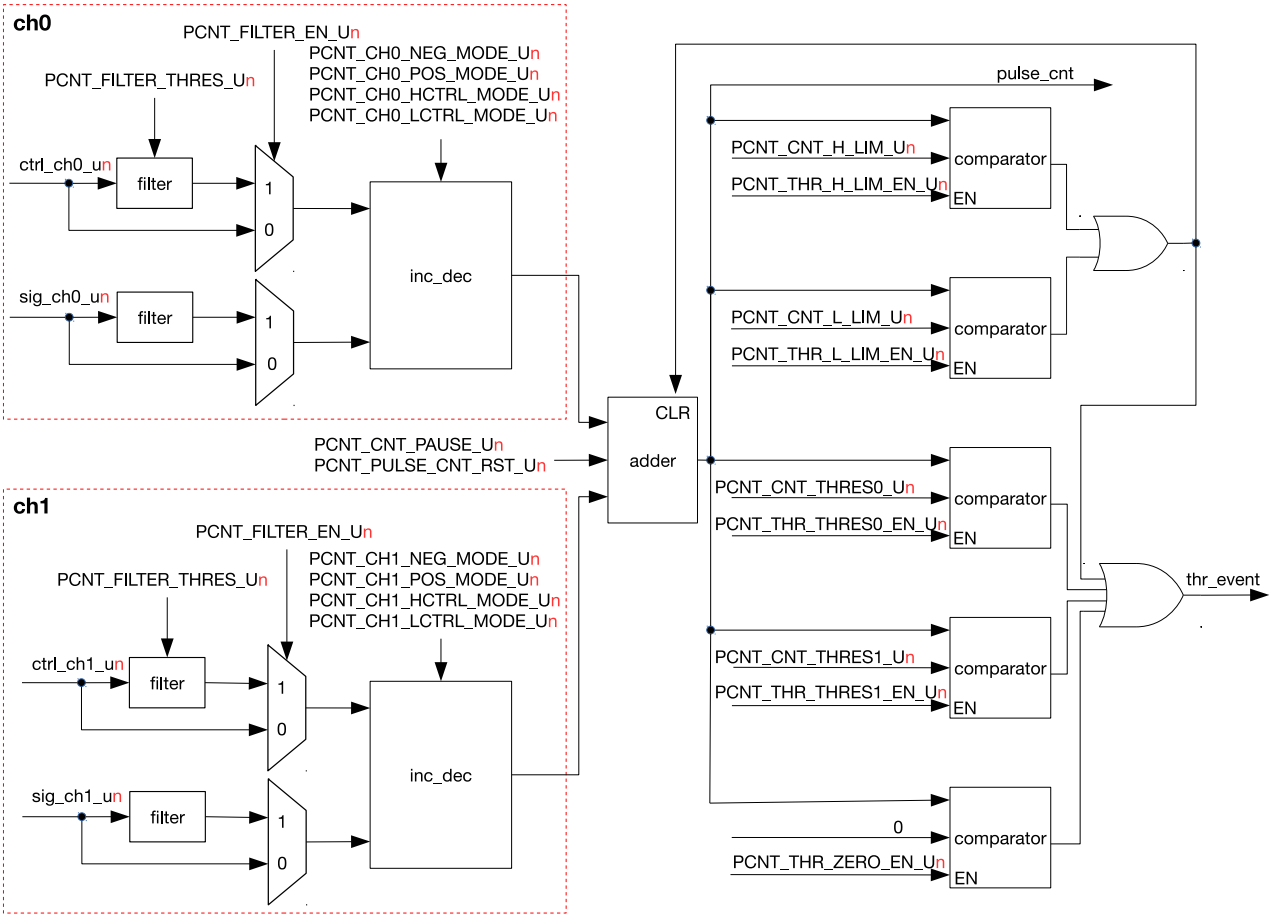


Figure 32.2-1. PCNT Unit Architecture

Figure 32.2-1 shows PCNT's architecture. As stated above, ctrl_ch0_u*n* is the control signal for ch0 of unit *n*. Its high and low states can be assigned with different counter modes to count the channel's input pulse signal sig_ch0_u*n* on negative or positive edges. The available counter modes are as follows:

- Increment mode: When a channel detects an active edge of sig_ch0_u*n* (the active edge can be configured by software), the counter value pulse_cnt increases by 1. Upon reaching PCNT_CNT_H_LIM_U*n*, pulse_cnt is cleared. If the channel's counter mode is changed or if

PCNT_CNT_PAUSE_U*n* is set to 1 before pulse_cnt reaches PCNT_CNT_H_LIM_U*n*, then pulse_cnt freezes and its counter mode changes.

- Decrement mode: When a channel detects an active edge of sig_ch0_u*n* (the active edge can be configured by software), the counter value pulse_cnt decreases by 1. Upon reaching PCNT_CNT_L_LIM_U*n*, pulse_cnt is cleared. If the channel's counter mode is changed or if PCNT_CNT_PAUSE_U*n* is set to 1 before pulse_cnt reaches PCNT_CNT_L_LIM_U*n*, then pulse_cnt freezes and its counter mode changes.
- Disable mode: Counting is disabled, and the counter value pulse_cnt freezes.

Table 32.2-1 to Table 32.2-4 provide information on how to configure the counter mode for channel 0.

Table 32.2-1. Counter Mode. Positive Edge of Input Pulse Signal. Control Signal in Low State

PCNT_CH0_POS_MODE_U*n*	PCNT_CH0_LCTRL_MODE_U*n*	Counter Mode
1	0	Increment
	1	Decrement
	Others	Disable
2	0	Decrement
	1	Increment
	Others	Disable
Others	N/A	Disable

Table 32.2-2. Counter Mode. Positive Edge of Input Pulse Signal. Control Signal in High State

PCNT_CH0_POS_MODE_U*n*	PCNT_CH0_HCTRL_MODE_U*n*	Counter Mode
1	0	Increment
	1	Decrement
	Others	Disable
2	0	Decrement
	1	Increment
	Others	Disable
Others	N/A	Disable

Table 32.2-3. Counter Mode. Negative Edge of Input Pulse Signal. Control Signal in Low State

PCNT_CH0_NEG_MODE_U*n*	PCNT_CH0_LCTRL_MODE_U*n*	Counter Mode
1	0	Increment
	1	Decrement
	Others	Disable
2	0	Decrement
	1	Increment
	Others	Disable
Others	N/A	Disable

Table 32.2-4. Counter Mode. Negative Edge of Input Pulse Signal. Control Signal in High State

PCNT_CH0_NEG_MODE_U*n*	PCNT_CH0_HCTRL_MODE_U*n*	Counter Mode
1	0	Increment
	1	Decrement
	Others	Disable
2	0	Decrement
	1	Increment
	Others	Disable
Others	N/A	Disable

Each unit has one filter for all its control and input pulse signals. The filter can be enabled with the bit PCNT_FILTER_EN_U*n*. It monitors the signals and ignores all the noise, i.e., the glitches with pulse widths shorter than PCNT_FILTER_THRES_U*n* APB clock cycles in length.

As shown on Figure 32.2-1, each unit has two channels which process different input pulse signals and increase or decrease values via their respective inc_dec modules, then the two channels send these values to the adder module which has a 16-bit wide signed register. This adder can be suspended by setting PCNT_CNT_PAUSE_U*n*, and cleared by setting PCNT_PULSE_CNT_RST_U*n*.

The PCNT has five watchpoints that share one interrupt. The interrupt can be enabled or disabled by interrupt enable signals of each individual watchpoint.

- Maximum count value: When pulse_cnt is greater than or equal to PCNT_CNT_H_LIM_U*n*, a high limit interrupt is triggered and PCNT_CNT_THR_H_LIM_LAT_U*n* is high.
- Minimum count value: When pulse_cnt is less than or equal to PCNT_CNT_L_LIM_U*n*, a low limit interrupt is triggered and PCNT_CNT_THR_L_LIM_LAT_U*n* is high.
- Two threshold values: When pulse_cnt equals either PCNT_CNT_THRES0_U*n* or PCNT_CNT_THRES1_U*n*, an interrupt is triggered and either PCNT_CNT_THR_THRES0_LAT_U*n* or PCNT_CNT_THR_THRES1_LAT_U*n* is high.
- Zero: When pulse_cnt is 0, an interrupt is triggered and PCNT_CNT_THR_ZERO_LAT_U*n* is valid.

If PCNT_CNT_H_LIM_U*n* and/or PCNT_CNT_L_LIM_U*n* are reconfigured by software when PCNT is working, the new configuration will take effect after pulse_cnt counts to any of the above five watchpoints; If PCNT_CNT_THRES0_U*n* and/or PCNT_CNT_THRES1_U*n* are reconfigured by software, the new configuration will take effect immediately.

32.3 Applications

In each unit, channel 0 and channel 1 can be configured to work independently or together. The three subsections below provide details of channel 0 incrementing independently, channel 0 decrementing independently, and channel 0 and channel 1 incrementing together. For other working modes not elaborated in this section (e.g., channel 1 incrementing/decremeting independently, or one channel incrementing while the other decrementing), reference can be made to these three subsections.

32.3.1 Channel 0 Incrementing Independently

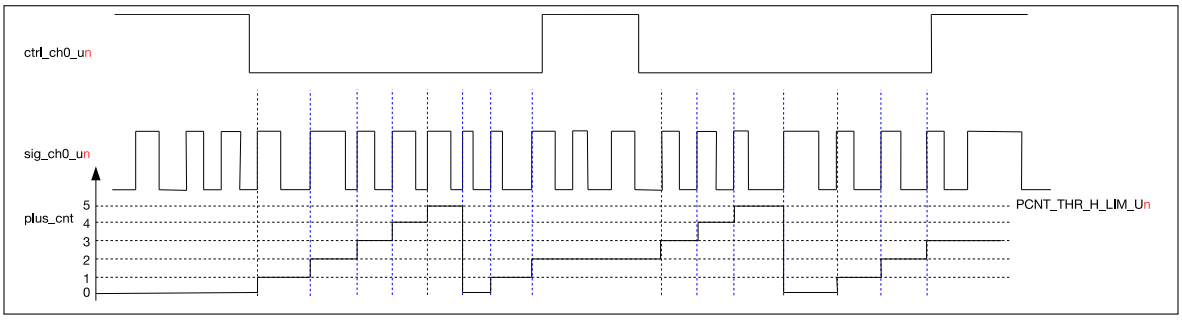


Figure 32.3-1. Channel 0 Up Counting Diagram

Figure 32.3-1 illustrates how channel 0 is configured to increment independently on the positive edge of sig_ch0_u*n* while channel 1 is disabled (see subsection 32.2 for how to disable channel 1). The configuration of channel 0 is shown below.

- PCNT_CH0_LCTRL_MODE_U*n*=0: When ctrl_ch0_u*n* is low, the counter mode specified for the low state turns on, in this case it is Increment mode.
- PCNT_CH0_HCTRL_MODE_U*n*=2: When ctrl_ch0_u*n* is high, the counter mode specified for the low state turns on, in this case it is Disable mode.
- PCNT_CH0_POS_MODE_U*n*=1: The counter increments on the positive edge of sig_ch0_u*n*.
- PCNT_CH0_NEG_MODE_U*n*=0: The counter idles on the negative edge of sig_ch0_u*n*.
- PCNT_CNT_H_LIM_U*n*=5: When pulse_cnt counts up to PCNT_CNT_H_LIM_U*n*, it is cleared.

32.3.2 Channel 0 Decrementing Independently

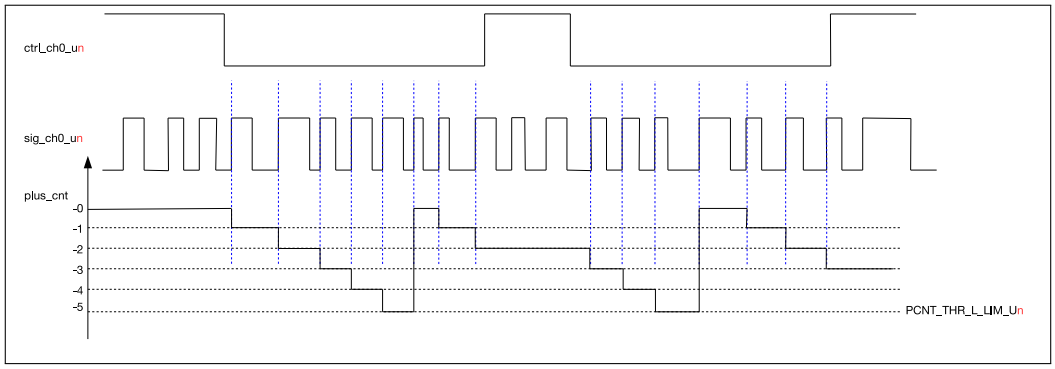


Figure 32.3-2. Channel 0 Down Counting Diagram

Figure 32.3-2 illustrates how channel 0 is configured to decrement independently on the positive edge of sig_ch0_u*n* while channel 1 is disabled. The configuration of channel 0 in this case differs from that in Figure 32.3-1 in the following aspects:

- PCNT_CH0_POS_MODE_U*n*=2: The counter decrements on the positive edge of sig_ch0_u*n*.
- PCNT_CNT_L_LIM_U*n*=-5: When pulse_cnt counts down to PCNT_CNT_L_LIM_U*n*, it is cleared.

32.3.3 Channel 0 and Channel 1 Incrementing Together

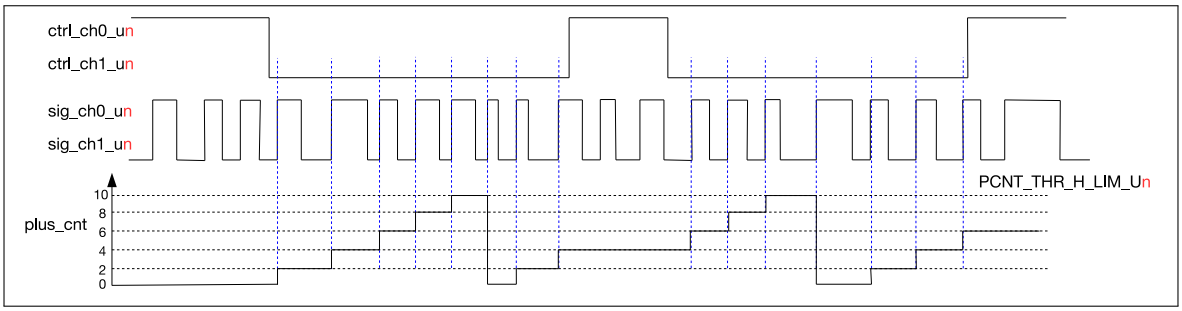


Figure 32.3-3. Two Channels Up Counting Diagram

Figure 32.3-3 illustrates how channel 0 and channel 1 are configured to increment on the positive edge of sig_ch0_u*n* and sig_ch1_u*n* respectively at the same time. It can be seen in Figure 32.3-3 that control signal ctrl_ch0_u*n* and ctrl_ch1_u*n* have the same waveform, so as input pulse signal sig_ch0_u*n* and sig_ch1_u*n*. The configuration procedure is shown below.

- For channel 0:
  - PCNT_CH0_LCTRL_MODE_U*n*=0: When ctrl_ch0_u*n* is low, the counter mode specified for the low state turns on, in this case it is Increment mode.
  - PCNT_CH0_HCTRL_MODE_U*n*=2: When ctrl_ch0_u*n* is high, the counter mode specified for the low state turns on, in this case it is Disable mode.
  - PCNT_CH0_POS_MODE_U*n*=1: The counter increments on the positive edge of sig_ch0_u*n*.
  - PCNT_CH0_NEG_MODE_U*n*=0: The counter idles on the negative edge of sig_ch0_u*n*.
- For channel 1:
  - PCNT_CH1_LCTRL_MODE_U*n*=0: When ctrl_ch1_u*n* is low, the counter mode specified for the low state turns on, in this case it is Increment mode.
  - PCNT_CH1_HCTRL_MODE_U*n*=2: When ctrl_ch1_u*n* is high, the counter mode specified for the low state turns on, in this case it is Disable mode.
  - PCNT_CH1_POS_MODE_U*n*=1: The counter increments on the positive edge of sig_ch1_u*n*.
  - PCNT_CH1_NEG_MODE_U*n*=0: The counter idles on the negative edge of sig_ch1_u*n*.
- PCNT_CNT_H_LIM_U*n*=10: When pulse_cnt counts up to PCNT_CNT_H_LIM_U*n*, it is cleared.

32.4 Register Summary

The addresses in this section are relative to Pulse Count Controller base address provided in Table 4.3-2 in Chapter 4 *System and Memory*.

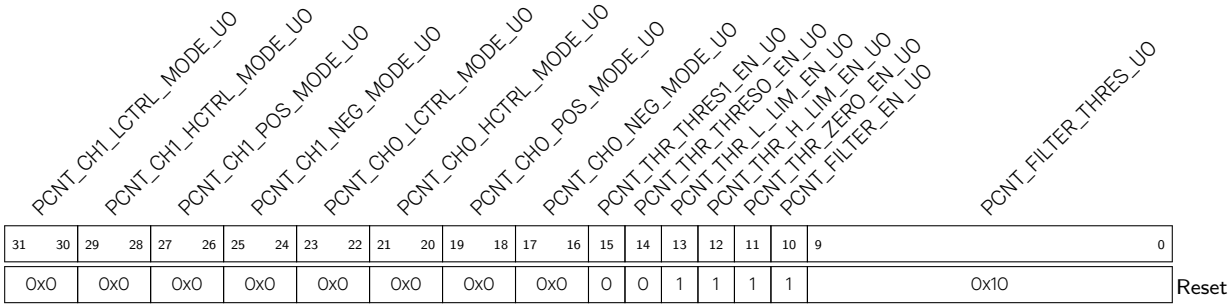
The abbreviations given in Column **Access** are explained in Section *Access Types for Registers*.

Name	Description	Address	Access
Configuration Register			
PCNT_U0_CONF0_REG	Configuration register 0 for unit 0	0x0000	R/W
PCNT_U0_CONF1_REG	Configuration register 1 for unit 0	0x0004	R/W
PCNT_U0_CONF2_REG	Configuration register 2 for unit 0	0x0008	R/W
PCNT_U1_CONF0_REG	Configuration register 0 for unit 1	0x000C	R/W
PCNT_U1_CONF1_REG	Configuration register 1 for unit 1	0x0010	R/W
PCNT_U1_CONF2_REG	Configuration register 2 for unit 1	0x0014	R/W
PCNT_U2_CONF0_REG	Configuration register 0 for unit 2	0x0018	R/W
PCNT_U2_CONF1_REG	Configuration register 1 for unit 2	0x001C	R/W
PCNT_U2_CONF2_REG	Configuration register 2 for unit 2	0x0020	R/W
PCNT_U3_CONF0_REG	Configuration register 0 for unit 3	0x0024	R/W
PCNT_U3_CONF1_REG	Configuration register 1 for unit 3	0x0028	R/W
PCNT_U3_CONF2_REG	Configuration register 2 for unit 3	0x002C	R/W
PCNT_CTRL_REG	Control register for all counters	0x0060	R/W
Status Register			
PCNT_U0_CNT_REG	Counter value for unit 0	0x0030	RO
PCNT_U1_CNT_REG	Counter value for unit 1	0x0034	RO
PCNT_U2_CNT_REG	Counter value for unit 2	0x0038	RO
PCNT_U3_CNT_REG	Counter value for unit 3	0x003C	RO
PCNT_U0_STATUS_REG	PNCT UNIT0 status register	0x0050	RO
PCNT_U1_STATUS_REG	PNCT UNIT1 status register	0x0054	RO
PCNT_U2_STATUS_REG	PNCT UNIT2 status register	0x0058	RO
PCNT_U3_STATUS_REG	PNCT UNIT3 status register	0x005C	RO
Interrupt Register			
PCNT_INT_RAW_REG	Interrupt raw status register	0x0040	RO
PCNT_INT_ST_REG	Interrupt status register	0x0044	RO
PCNT_INT_ENA_REG	Interrupt enable register	0x0048	R/W
PCNT_INT_CLR_REG	Interrupt clear register	0x004C	WO
Version Register			
PCNT_DATE_REG	PCNT version control register	0x00FC	R/W

32.5 Registers

The addresses in this section are relative to Pulse Count Controller base address provided in Table 4.3-2 in Chapter 4 *System and Memory*.

Register 32.1. PCNT_U*n*_CONF0_REG (*n*: 0-3) (0x0000+0xCn*)**



PCNT_FILTER_THRES_U*n* Configures the maximum threshold for the filter. Any pulses with width less than this will be ignored when the filter is enabled.
Measurement unit: APB_CLK cycles.
(R/W)

PCNT_FILTER_EN_U*n* This is the enable bit for unit *n*'s input filter. (R/W)

PCNT_THR_ZERO_EN_U*n* This is the enable bit for unit *n*'s zero comparator. (R/W)

PCNT_THR_H_LIM_EN_U*n* This is the enable bit for unit *n*'s thr_h_lim comparator. Configures it to enable the high limit interrupt. (R/W)

PCNT_THR_L_LIM_EN_U*n* This is the enable bit for unit *n*'s thr_l_lim comparator. Configures it to enable the low limit interrupt. (R/W)

PCNT_THR_THRES0_EN_U*n* This is the enable bit for unit *n*'s thres0 comparator. (R/W)

PCNT_THR_THRES1_EN_U*n* This is the enable bit for unit *n*'s thres1 comparator. (R/W)

PCNT_CH0_NEG_MODE_U*n* Configures the behavior when the signal input of channel 0 detects a negative edge.
1: Increment the counter
2: Decrement the counter
0, 3: No effect
(R/W)

PCNT_CH0_POS_MODE_U*n* Configures the behavior when the signal input of channel 0 detects a positive edge.
1: Increment the counter
2: Decrement the counter
0, 3: No effect
(R/W)

Continued on the next page...

Register 32.1. PCNT_U*n*_CONF0_REG (*n*: 0-3) (0x0000+0xC**n*)

Continued from the previous page...

PCNT_CH0_HCTRL_MODE_U*n* Configures how the CH*n*_POS_MODE/CH*n*_NEG_MODE settings will be modified when the control signal is high.
0: No modification
1: Invert the behavior (increase → decrease, decrease → increase)
2, 3: Inhibit counter modification
(R/W)

PCNT_CH0_LCTRL_MODE_U*n* Configures how the CH*n*_POS_MODE/CH*n*_NEG_MODE settings will be modified when the control signal is low.
0: No modification
1: Invert the behavior (increase → decrease, decrease → increase)
2, 3: Inhibit counter modification
(R/W)

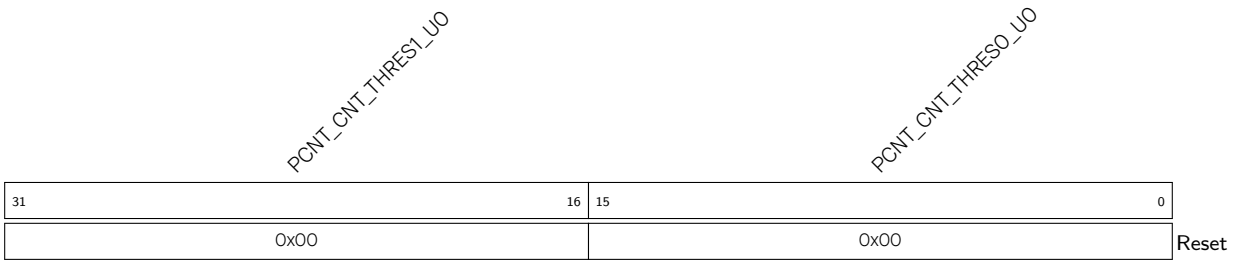
PCNT_CH1_NEG_MODE_U*n* Configures the behavior when the signal input of channel 1 detects a negative edge.
1: Increment the counter
2: Decrement the counter
0, 3: No effect
(R/W)

PCNT_CH1_POS_MODE_U*n* Configures the behavior when the signal input of channel 1 detects a positive edge.
1: Increment the counter
2: Decrement the counter
0, 3: No effect
(R/W)

PCNT_CH1_HCTRL_MODE_U*n* Configures how the CH*n*_POS_MODE/CH*n*_NEG_MODE settings will be modified when the control signal is high.
0: No modification
1: Invert the behavior (increase → decrease, decrease → increase)
2, 3: Inhibit counter modification
(R/W)

PCNT_CH1_LCTRL_MODE_U*n* Configures how the CH*n*_POS_MODE/CH*n*_NEG_MODE settings will be modified when the control signal is low.
0: No modification
1: Invert the behavior (increase → decrease, decrease → increase)
2, 3: Inhibit counter modification
(R/W)

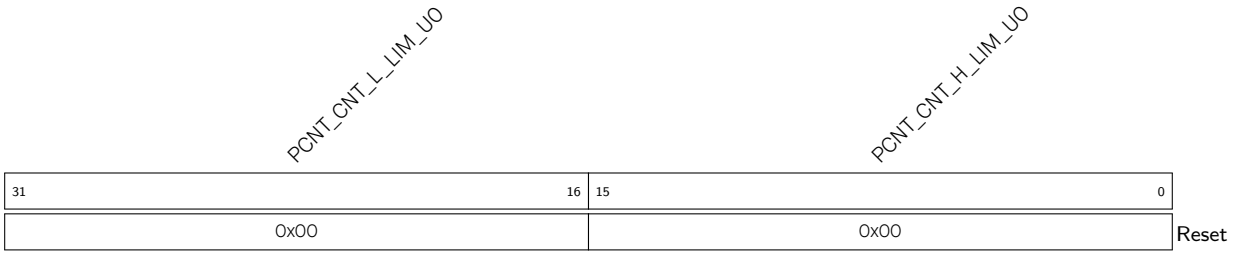
Register 32.2. PCNT_U*n*_CONF1_REG (*n*: 0-3) (0x0004+0xC**n*)



PCNT_CNT_THRES0_U*n* Configures the thres0 value for unit *n*. (R/W)

PCNT_CNT_THRES1_U*n* Configures the thres1 value for unit *n*. (R/W)

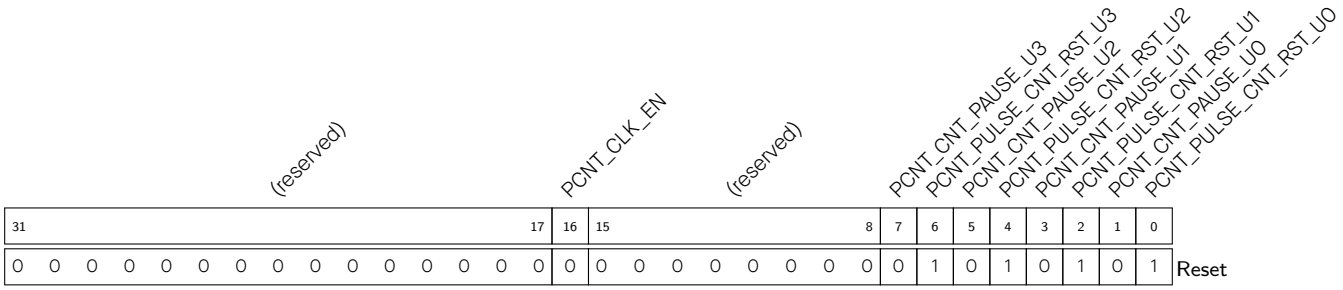
Register 32.3. PCNT_U*n*_CONF2_REG (*n*: 0-3) (0x0008+0xC**n*)



PCNT_CNT_H_LIM_U*n* Configures the thr_h_lim value for unit *n*. When pulse_cnt reaches this value, the counter will be cleared to 0. (R/W)

PCNT_CNT_L_LIM_U*n* Configures the thr_l_lim value for unit *n*. When pulse_cnt reaches this value, the counter will be cleared to 0. (R/W)

Register 32.4. PCNT_CTRL_REG (0x0060)



PCNT_PULSE_CNT_RST_U*n* Write 1 to clear unit *n*'s counter.

0: No effect

1: Clear

(R/W)

PCNT_CNT_PAUSE_U*n* Write 1 to freeze unit *n*'s counter.

0: No effect

1: Freeze

(R/W)

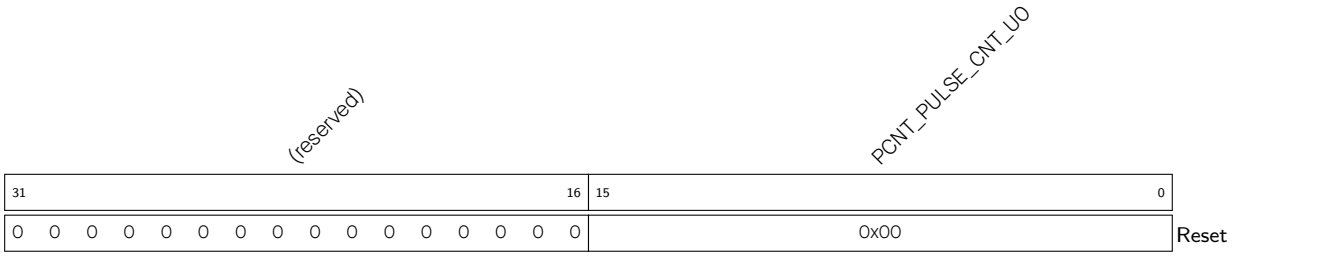
PCNT_CLK_EN Configures whether or not to enable the registers clock gate of the PCNT module.

0: The clock for registers is enabled when registers are read and written

1: The clock for registers is always on

(R/W)

Register 32.5. PCNT_U*n*_CNT_REG (*n*: 0-3) (0x0030+0x4**n*)



PCNT_PULSE_CNT_U*n* Represents the current pulse count value for unit *n*. (RO)

Register 32.6. PCNT_U*n*_STATUS_REG (*n*: 0-3) (0x0050+0x4**n*)

(reserved)	PCNT_CNT_THR_ZERO_LAT_U0	PCNT_CNT_THR_H_LIM_LAT_U0	PCNT_CNT_THR_L_LIM_LAT_U0	PCNT_CNT_THR_THRES0_LAT_U0	PCNT_CNT_THR_THRES1_LAT_U0	PCNT_CNT_THR_ZERO_MODE_U0	
31 … 7	6	5	4	3	2	1 … 0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	0	0	0	0	0x0	Reset

PCNT_CNT_THR_ZERO_MODE_U*n* Represents the pulse counter status of PCNT_U*n* corresponding to 0.

0: The pulse counter decreases from positive to 0

1: The pulse counter increases from negative to 0

2: The pulse counter is negative

3: The pulse counter is positive

(RO)

PCNT_CNT_THR_THRES1_LAT_U*n* Represents the latched value of thres1 event of PCNT_U*n* when threshold event interrupt is valid.

0: Others

1: The current pulse counter equals to thres1 and the thres1 event is valid

(RO)

PCNT_CNT_THR_THRES0_LAT_U*n* Represents the latched value of thres0 event of PCNT_U*n* when threshold event interrupt is valid.

0: Others

1: The current pulse counter equals to thres0 and the thres0 event is valid

(RO)

PCNT_CNT_THR_L_LIM_LAT_U*n* Represents the latched value of the low limit event of PCNT_U*n* when the threshold event interrupt is valid.

0: Others

1: The current pulse counter equals to thr_l_lim and the low limit event is valid.

(RO)

PCNT_CNT_THR_H_LIM_LAT_U*n* Represents the latched value of the high limit event of PCNT_U*n* when the threshold event interrupt is valid.

0: Others

1: The current pulse counter equals to thr_h_lim and the high limit event is valid.

(RO)

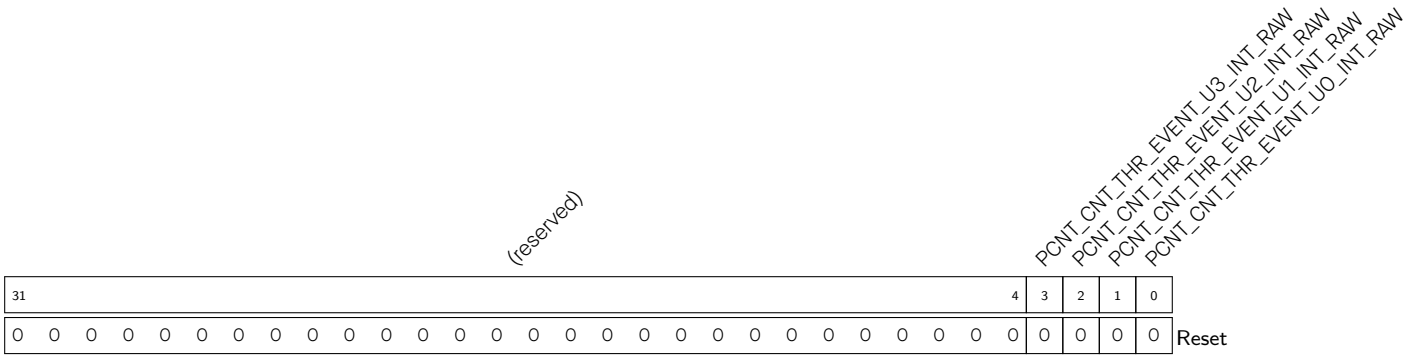
PCNT_CNT_THR_ZERO_LAT_U*n* Represents the latched value of the zero threshold event of PCNT_U*n* when the threshold event interrupt is valid.

0: Others

1: The current pulse counter equals to 0 and the zero threshold event is valid.

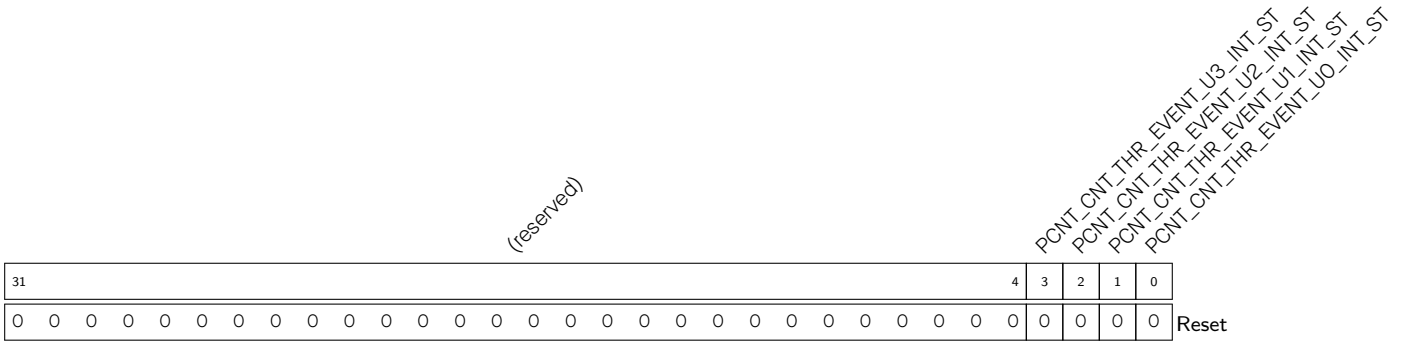
(RO)

Register 32.7. PCNT_INT_RAW_REG (0x0040)



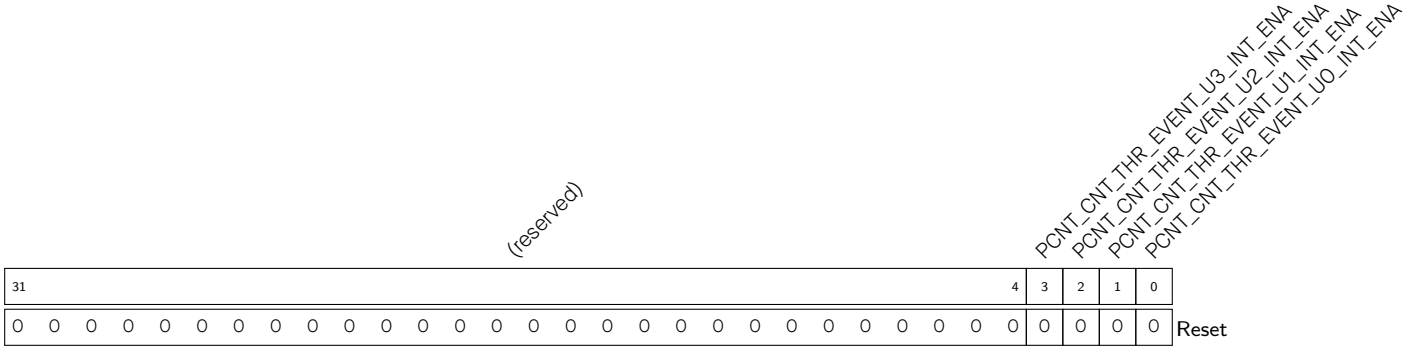
PCNT_CNT_THR_EVENT_U*n*_INT_RAW The raw interrupt status of the PCNT_CNT_THR_EVENT_U*n*_INT interrupt. (RO)

Register 32.8. PCNT_INT_ST_REG (0x0044)



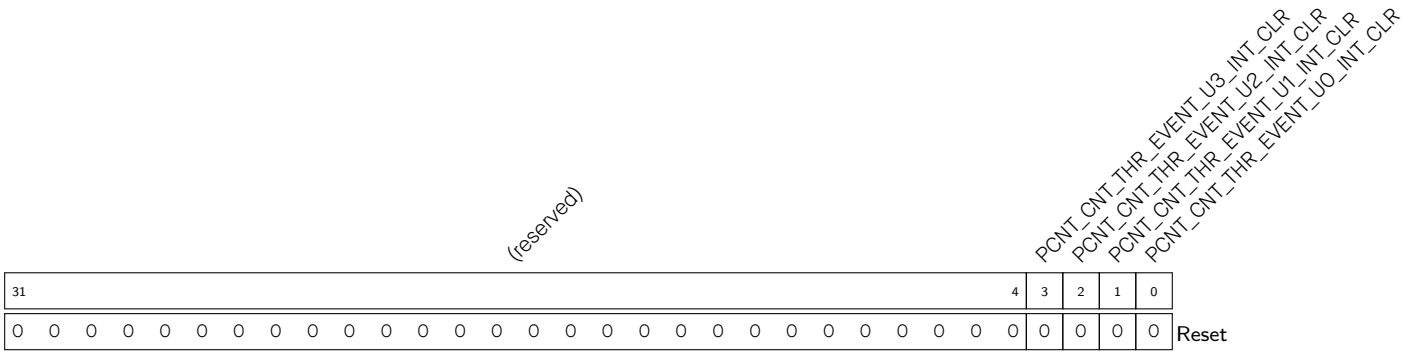
PCNT_CNT_THR_EVENT_U*n*_INT_ST The masked interrupt status of the PCNT_CNT_THR_EVENT_U*n*_INT interrupt. (RO)

Register 32.9. PCNT_INT_ENA_REG (0x0048)



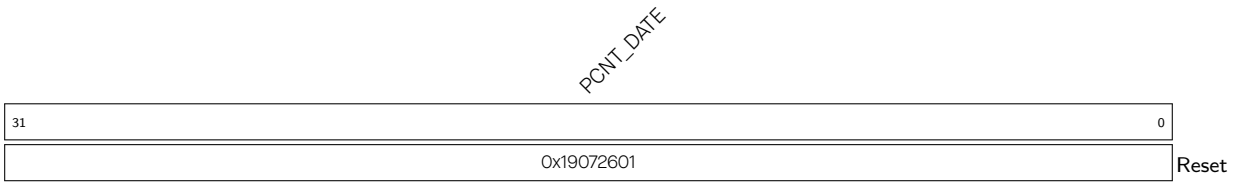
PCNT_CNT_THR_EVENT_U*n*_INT_ENA Write 1 to enable the PCNT_CNT_THR_EVENT_U*n*_INT interrupt. (R/W)

Register 32.10. PCNT_INT_CLR_REG (0x004C)



PCNT_CNT_THR_EVENT_U*n*_INT_CLR Write 1 to clear the PCNT_CNT_THR_EVENT_U*n*_INT interrupt. (WO)

Register 32.11. PCNT_DATE_REG (0x00FC)



PCNT_DATE Version control register. (R/W)

Chapter 33

USB Serial/JTAG Controller (USB_SERIAL_JTAG)

ESP32-H2 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.

33.1 Overview

While programming and debugging an ESP32-H2 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-H2 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-H2.

33.2 Features

The USB Serial/JTAG controller has the following features:

- USB Full-speed device; Hardwired for CDC-ACM (Communication Device Class - Abstract Control Model) and JTAG adapter functionality
- CDC-ACM:
  - Integrates CDC-ACM adherent serial port emulation (plug-and-play on most modern OSes)
  - Supports host controllable chip reset and entry into download mode
- JTAG adapter functionality:
  - Allows fast communication with CPU debugging core using a compact representation of JTAG instructions
- Two OUT endpoints and three IN endpoints in addition to Control endpoint 0; Up to 64-byte data payload size
- Internal PHY: very few or no external components needed to connect to a host computer
- Two pins USB D+ and USB D- are changeable

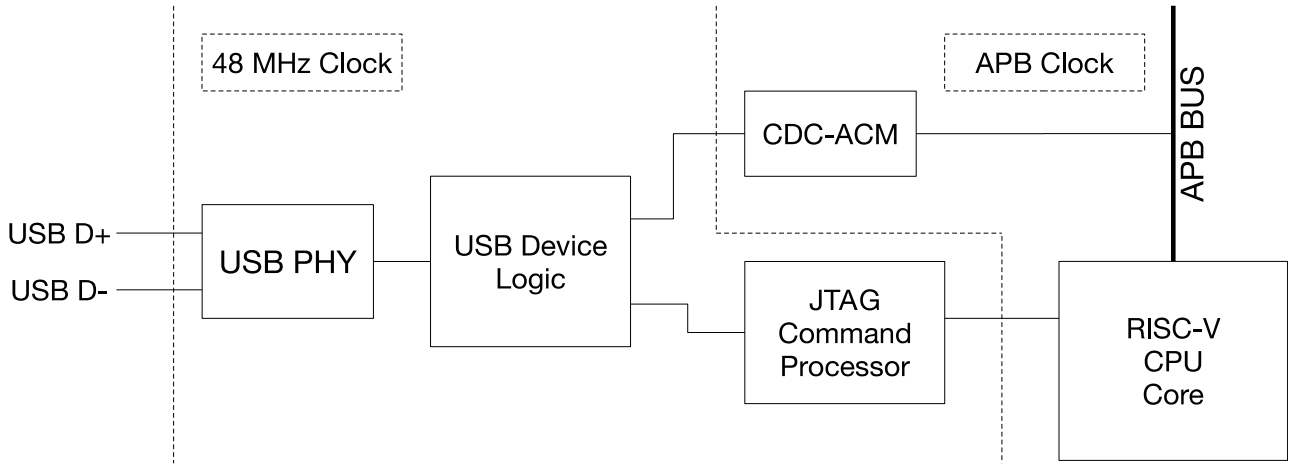


Figure 33.2-1. USB Serial/JTAG High Level Diagram

As shown in Figure 33.2-1, the USB Serial/JTAG controller consists of a USB PHY, a USB device interface, a JTAG command processor with an integrated 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 33.2-2 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.

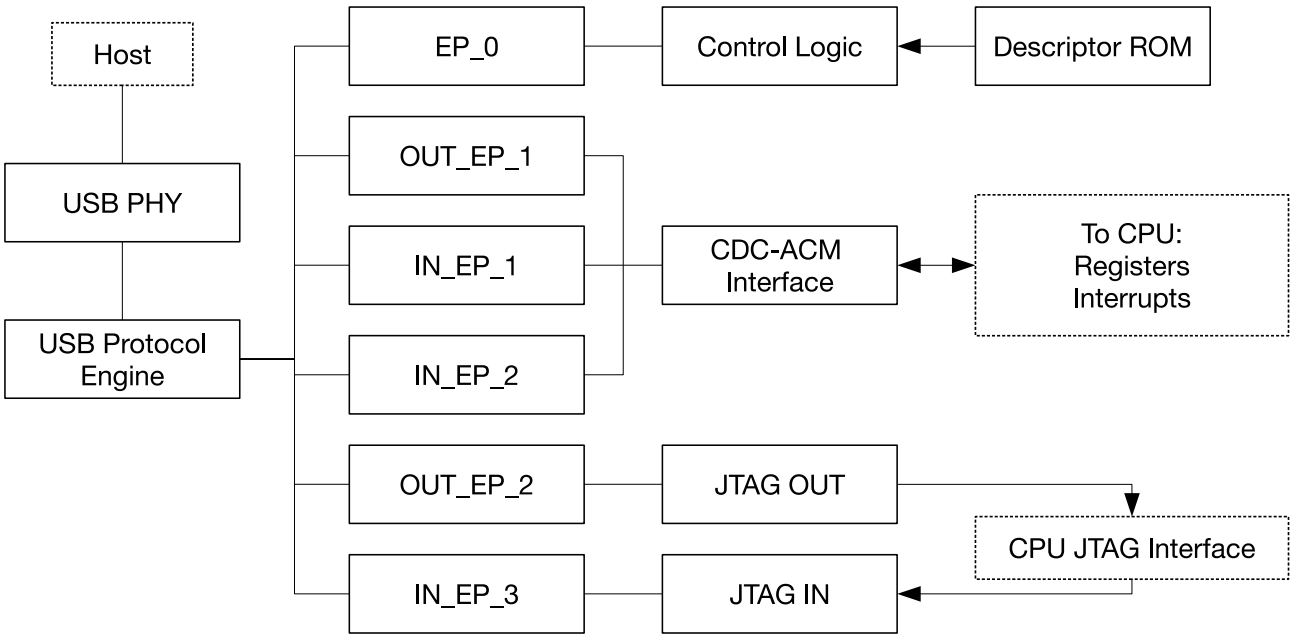


Figure 33.2-2. USB Serial/JTAG Block Diagram

33.3 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.

33.3.1 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-byte packets 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 33.3-1. 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 databits, 1 stopbit (Can be changed through software)
SET_CONTROL_LINE_STATE	Set the state of the RTS/DTR lines. See Table 33.3-2

Aside from general-purpose communication, the CDC-ACM interface can also be used to reset ESP32-H2 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 33.3-2. 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-H2
1	1	No action

Note that if the download mode flag is set when ESP32-H2 is reset, ESP32-H2 will reboot into download mode. When this flag is cleared and the chip is reset, ESP32-H2 will boot from flash. For specific sequences, please refer to Section 33.4. All these functions can also be disabled by programming various eFuses. Please refer to Chapter 5 *eFuse Controller (EFUSE)* for more details.

33.3.2 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-H2. 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 debugging device is automatically readied to receive a new data packet from the host.

When the firmware has data to send, it can put the data in the send buffer and trigger a flush to allow the host to receive the data in a USB packet. In order to do so, there needs to be space available in the send buffer. Firmware can check this by reading USB_REG_SERIAL_IN_EP_DATA_FREE. A 1 in this register field indicates there is still free room in the buffer, and firmware can fill the buffer by writing bytes to the USB_SERIAL_JTAG_EP1_REG register. Writing the buffer does not immediately trigger sending data to the host until the buffer is flushed. After the flush, the entire buffer will be ready to be received by the USB host at once. A flush can be triggered in two ways: 1) after the 64th byte is written to the buffer, the USB hardware will automatically flush the buffer to the host; or 2) firmware can trigger a flush by writing 1 to USB_SERIAL_JTAG_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-H2. 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.

33.3.3 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-H2.

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-H2. 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-H2 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 33.3-3. 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-H2.
- 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-H2.
- 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. A CMD_REP instruction can be repeated up to five times, for a maximum of 1023 repeats of the initial instruction. 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.

33.3.4 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.

33.3.5 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 zero-byte USB responses to be sent. However, in the current implementation, some zero-byte responses may be generated in extraordinary circumstances. It is recommended to ignore these responses.

33.3.6 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 33.3-4. 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 33.3-5). 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-H2 are as follows. Note that all 16-bit values are little-endian.

Table 33.3-5. 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

33.4 Recommended Operation

Little setup is needed for using the USB Serial/JTAG device. The USB-to-JTAG hardware itself does not need any setup aside from the standard USB initialization that the host operating system already does. Apart from that, the CDC-ACM emulation on the host side is also plug-and-play.

On the firmware side, very little initialization is needed either. The USB hardware is self-initialized and after boot-up, if a host is connected and listening on the CDC-ACM interface, data can be exchanged as described above without any specific setup except for the situation when the firmware optionally sets up an interrupt service handler.

One thing to note is that there may be situations where either the host is not attached or the CDC-ACM virtual port is not opened. In such cases, the packets that are flushed to the host will never be picked up and the send buffer will never be empty. It is important to detect these situations and implement timeout, as this is the only way to reliably detect whether the port on the host side is closed or not.

Another thing to note is that the USB device is dependent on the BBPLL for the 48 MHz USB PHY clock. If this PLL is disabled, the USB communication will cease to function.

One scenario where this happens is Deep-sleep. The USB Serial/JTAG controller (as well as the attached RISC-V CPU) will be entirely powered down in Deep-sleep mode. If a device needs to be debugged in this mode, it may be preferable to use an external JTAG debugger and a serial interface instead.

The CDC-ACM interface can also be used to reset the SoC and take it into or out of download mode. Generating the correct sequence of handshake signals can be a bit complicated, since most operating systems only allow setting or clearing 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 33.4-1. Reset SoC into Download Mode

Action	Internal state	Note
Clear DTR	RTS=?, DTR=0	Initialize to known values
Clear RTS	RTS=0, DTR=0	-
Set DTR	RTS=0, DTR=1	Set download mode flag
Clear RTS	RTS=0, DTR=1	Propagate DTR
Set RTS	RTS=1, DTR=1	-
Clear DTR	RTS=1, DTR=0	Reset SoC
Set RTS	RTS=1, DTR=0	Propagate DTR
Clear RTS	RTS=0, DTR=0	Clear download flag

To reset the SoC into booting from flash:

Table 33.4-2. Reset SoC into Booting from flash

Action	Internal state	Note
Clear DTR	RTS=?, DTR=0	-
Clear RTS	RTS=0, DTR=0	Clear download flag
Set RTS	RTS=1, DTR=0	Reset SoC
Clear RTS	RTS=0, DTR=0	Exit reset

33.5 Interrupts

- USB_SERIAL_JTAG_JTAG_IN_FLUSH_INT: triggered when flush cmd is received for IN endpoint 2 of JTAG.
- 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.

33.6 Register Summary

The addresses in this section are relative to USB Serial/JTAG controller base address provided in Table 4.3-2 in Chapter 4 *System and Memory*.

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

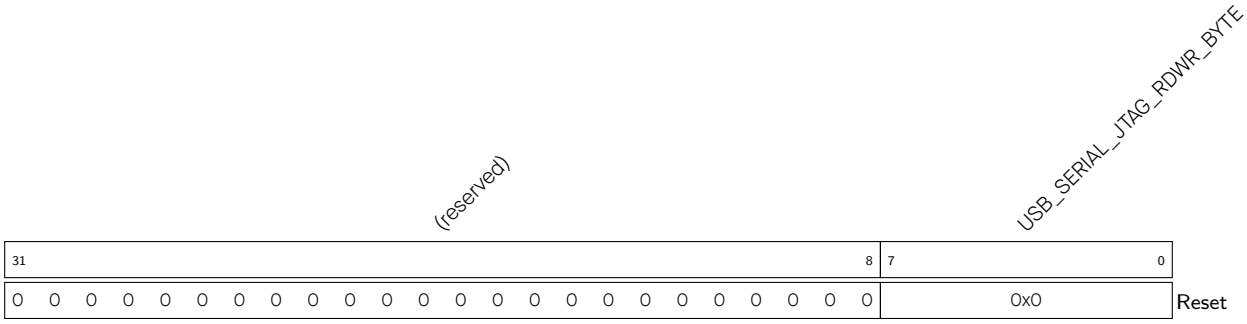
Name	Description	Address	Access
Configuration Registers			
USB_SERIAL_JTAG_EP1_REG	FIFO access for the CDC-ACM data IN and OUT endpoints	0x0000	R/W
USB_SERIAL_JTAG_EP1_CONF_REG	Configuration and control register for the CDC-ACM FIFOs	0x0004	varies
USB_SERIAL_JTAG_CONF0_REG	PHY hardware configuration	0x0018	R/W
USB_SERIAL_JTAG_TEST_REG	Register 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 configuration register	0x0064	varies
Interrupt Registers			
USB_SERIAL_JTAG_INT_RAW_REG	Interrupt raw status register	0x0008	R/WTC/SS
USB_SERIAL_JTAG_INT_ST_REG	Interrupt status register	0x000C	RO
USB_SERIAL_JTAG_INT_ENA_REG	Interrupt enable status register	0x0010	R/W
USB_SERIAL_JTAG_INT_CLR_REG	Interrupt clear status register	0x0014	WT
Status Registers			
USB_SERIAL_JTAG_JFIFO_ST_REG	JTAG FIFO status and control register	0x0020	varies
USB_SERIAL_JTAG_FRAM_NUM_REG	Last received SOF frame index register	0x0024	RO
USB_SERIAL_JTAG_IN_EP0_ST_REG	Control IN endpoint status information	0x0028	RO
USB_SERIAL_JTAG_IN_EP1_ST_REG	CDC-ACM IN endpoint status information	0x002C	RO
USB_SERIAL_JTAG_IN_EP2_ST_REG	CDC-ACM interrupt IN endpoint status information	0x0030	RO
USB_SERIAL_JTAG_IN_EP3_ST_REG	JTAG IN endpoint status information	0x0034	RO
USB_SERIAL_JTAG_OUT_EP0_ST_REG	Control OUT endpoint status information	0x0038	RO
USB_SERIAL_JTAG_OUT_EP1_ST_REG	CDC-ACM OUT endpoint status information	0x003C	RO
USB_SERIAL_JTAG_OUT_EP2_ST_REG	JTAG OUT endpoint status information	0x0040	RO
USB_SERIAL_JTAG_SET_LINE_CODE_W0_REG	W0 of SET_LINE_CODING command	0x0050	RO

Name	Description	Address	Access
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

33.7 Registers

The addresses in this section are relative to USB Serial/JTAG controller base address provided in Table 4.3-2 in Chapter 4 *System and Memory*.

Register 33.1. USB_SERIAL_JTAG_EP1_REG (0x0000)



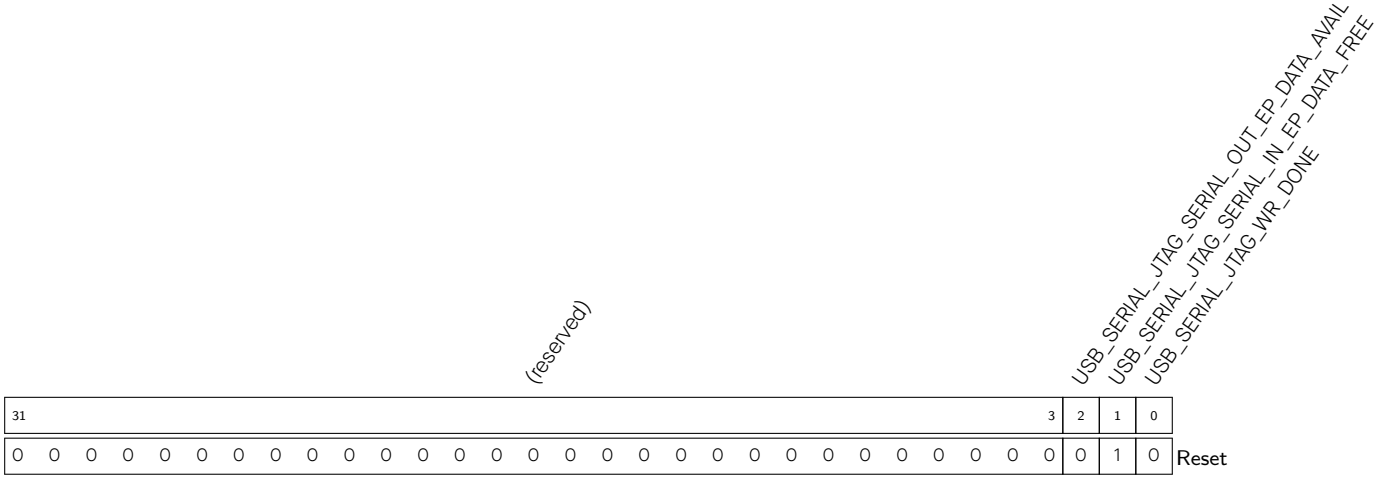
USB_SERIAL_JTAG_RDWR_BYTE Write or read byte data to or from UART TX/RX FIFO.

When USB_SERIAL_JTAG_SERIAL_IN_EMPTY_INT is set, users can write data (up to 64 bytes) into UART 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 UART RX FIFO through this register.

(R/W)

Register 33.2. USB_SERIAL_JTAG_EP1_CONF_REG (0x0004)



USB_SERIAL_JTAG_WR_DONE Configures whether to represent writing byte data to UART TX FIFO is done.

0: No effect

1: Represents writing byte data to UART TX FIFO is done

This bit then stays 0 until data in UART TX FIFO is read by the USB Host.

(WT)

USB_SERIAL_JTAG_SERIAL_IN_EP_DATA_FREE Represents whether UART TX FIFO has space available.

0: UART TX FIFO is full and no data should be written into it

1: UART 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 UART TX FIFO is read by USB Host.

(RO)

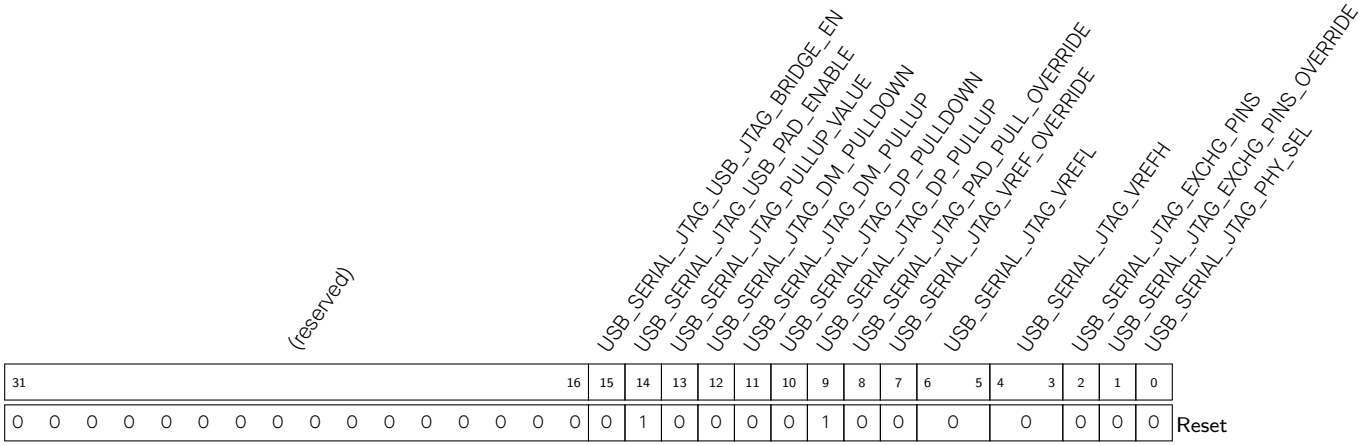
USB_SERIAL_JTAG_SERIAL_OUT_EP_DATA_AVAIL Represents whether there is data in UART RX FIFO.

0: There is no data in UART RX FIFO

1: There is data in UART RX FIFO

(RO)

Register 33.3. USB_SERIAL_JTAG_CONF0_REG (0x0018)



USB_SERIAL_JTAG_PHY_SEL Configures whether to select internal or external PHY.

0: Internal PHY

1: External PHY

(R/W)

USB_SERIAL_JTAG_EXCHG_PINS_OVERRIDE Configures whether to enable software control USB D+ and D- exchange.

0: Disable

1: Enable

(R/W)

USB_SERIAL_JTAG_EXCHG_PINS Configures whether to enable USB D+ and 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 33.3. USB_SERIAL_JTAG_CONF0_REG (0x0018)

Continued from the previous page...

USB_SERIAL_JTAG_PAD_PULL_OVERRIDE Configures whether to enable software to control USB D+ D- pullup and pulldown.
0: Disable
1: Enable
(R/W)

USB_SERIAL_JTAG_DP_PULLUP Configures whether to enable USB D+ pull up when USB_SERIAL_JTAG_PAD_PULL_OVERRIDE is 1.
0: Disable
1: Enable
(R/W)

USB_SERIAL_JTAG_DP_PULLDOWN Configures whether to enable USB D+ pull down when USB_SERIAL_JTAG_PAD_PULL_OVERRIDE is 1.
0: Disable
1: Enable
(R/W)

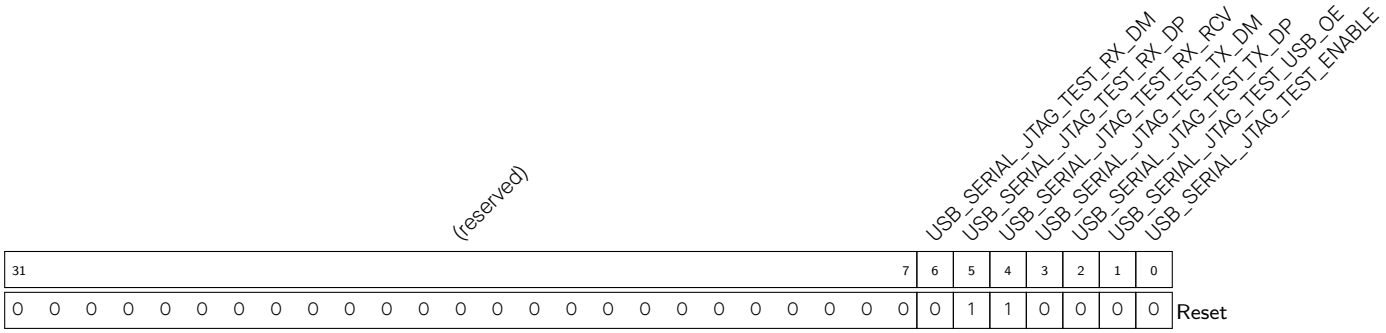
USB_SERIAL_JTAG_DM_PULLDOWN Configures whether to enable USB D- pull down when USB_SERIAL_JTAG_PAD_PULL_OVERRIDE is 1.
0: Disable
1: Enable
(R/W)

USB_SERIAL_JTAG_PULLUP_VALUE Configures the pull up value when USB_SERIAL_JTAG_PAD_PULL_OVERRIDE is 1.
0: 2.2 K
1: 1.1 K
(R/W)

USB_SERIAL_JTAG_USB_PAD_ENABLE Configures whether to enable USB pad function.
0: Disable
1: Enable
(R/W)

USB_SERIAL_JTAG_USB_JTAG_BRIDGE_EN Configures whether to disconnect usb_jtag and internal JTAG.
0: usb_jtag is connected to the internal JTAG port of CPU
1: usb_jtag and the internal JTAG are disconnected, MTMS, MTDI, and MTCK are output through GPIO Matrix, and MTDO is input through GPIO Matrix
(R/W)

Register 33.4. 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+

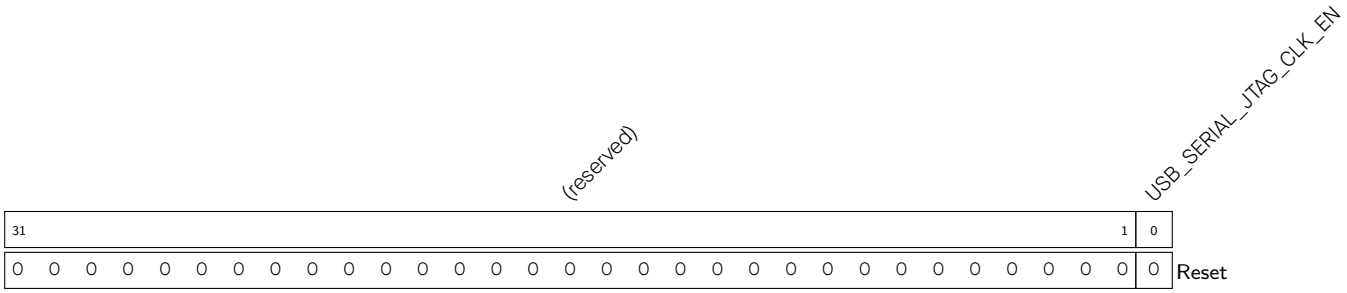
1: USB D+ voltage is higher than USB D-

(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 33.5. USB_SERIAL_JTAG_MISC_CONF_REG (0x0044)



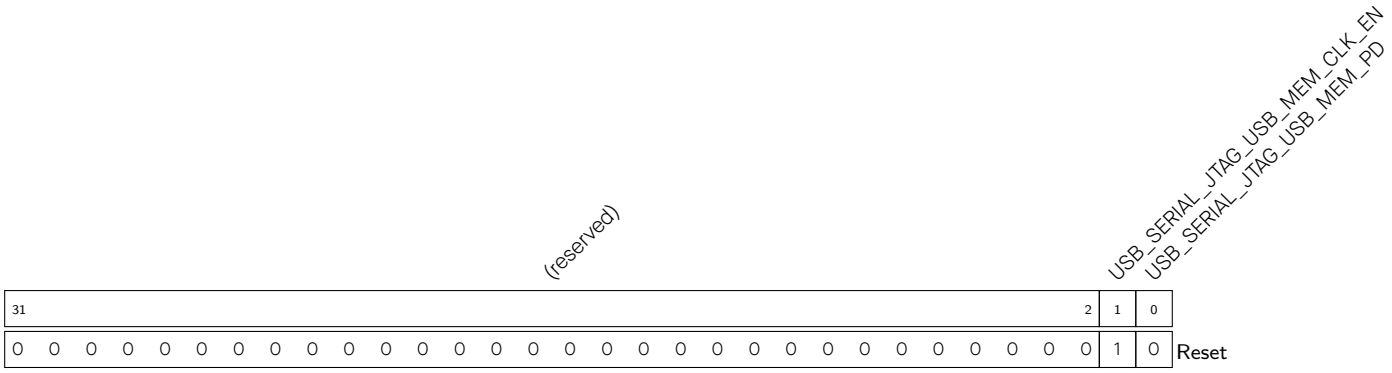
USB_SERIAL_JTAG_CLK_EN Configures whether to force clock on for register.

0: Support clock only when an application writes registers

1: Force clock on for register

(R/W)

Register 33.6. 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)

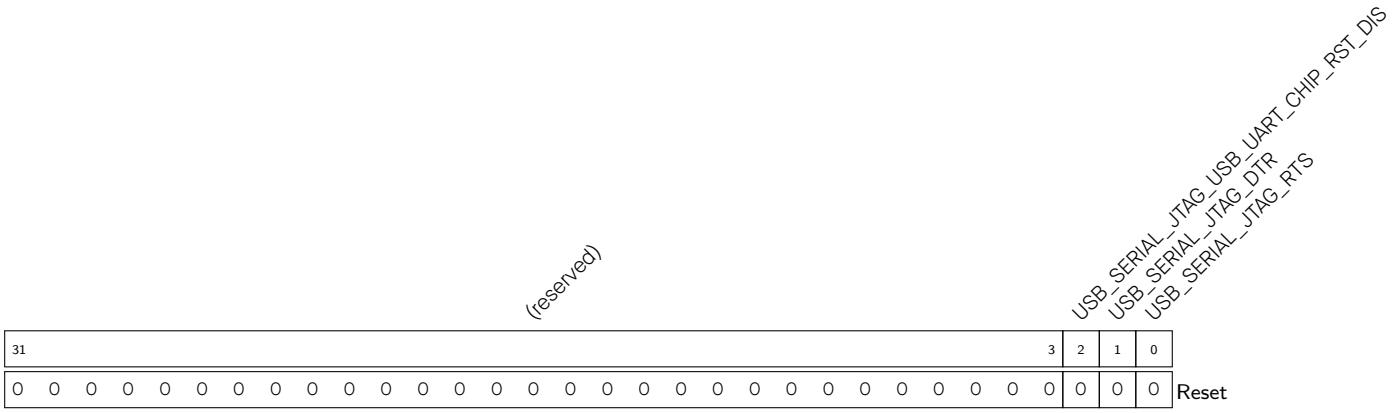
USB_SERIAL_JTAG_USB_MEM_CLK_EN Configures whether to force clock on for USB memory.

0: No effect

1: Force

(R/W)

Register 33.7. USB_SERIAL_JTAG_CHIP_RST_REG (0x004C)

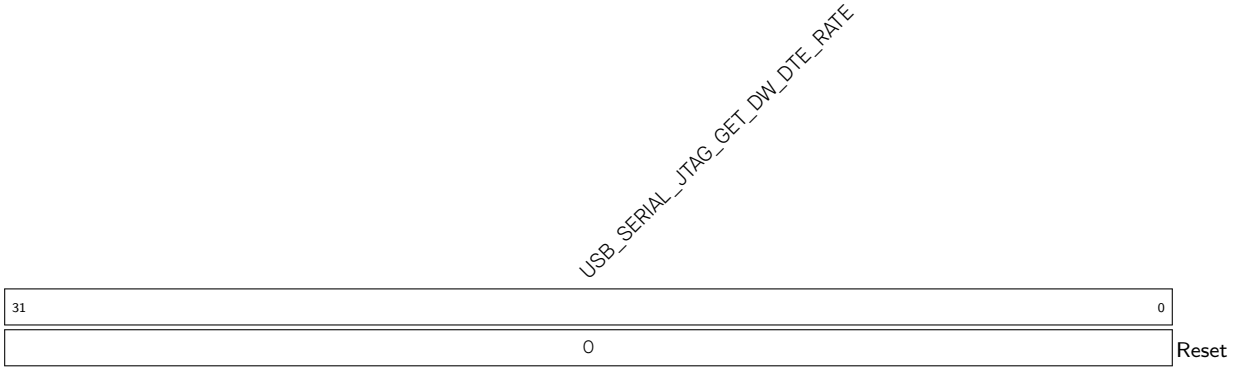


USB_SERIAL_JTAG_RTS Represents the state of RTS signal as set by the most recent SET_LINE_CODING command. (RO)

USB_SERIAL_JTAG_DTR Represents the state of DTR signal as set by the most recent SET_LINE_CODING command. (RO)

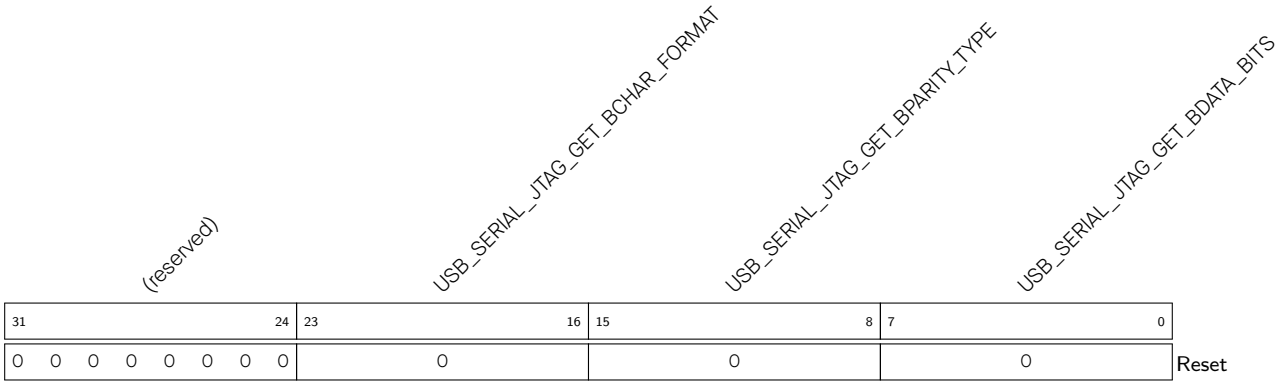
USB_SERIAL_JTAG_USB_UART_CHIP_RST_DIS Configures whether to disable chip reset from USB serial channel.
0: No effect
1: Disable
(R/W)

Register 33.8. USB_SERIAL_JTAG_GET_LINE_CODE_W0_REG (0x0058)



USB_SERIAL_JTAG_GET_DW_DTE_RATE Configures the value of dwDTERate (in bits per second) set by software, which is requested by GET_LINE_CODING command. Note that the value here does not affect the actual communication speed at all. (R/W)

Register 33.9. USB_SERIAL_JTAG_GET_LINE_CODE_W1_REG (0x005C)

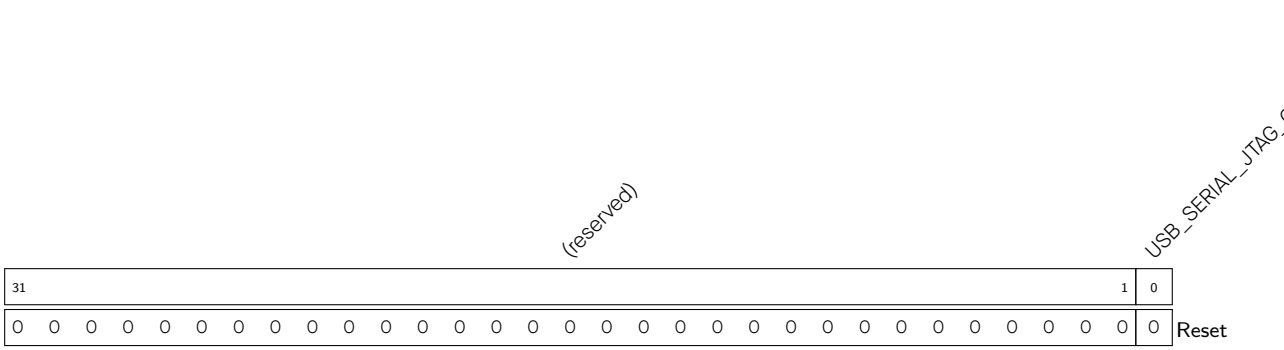


USB_SERIAL_JTAG_GET_BDATA_BITS Configures the value of bDataBits set by software, which is requested by GET_LINE_CODING command. (R/W)

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 33.10. USB_SERIAL_JTAG_CONFIG_UPDATE_REG (0x0060)



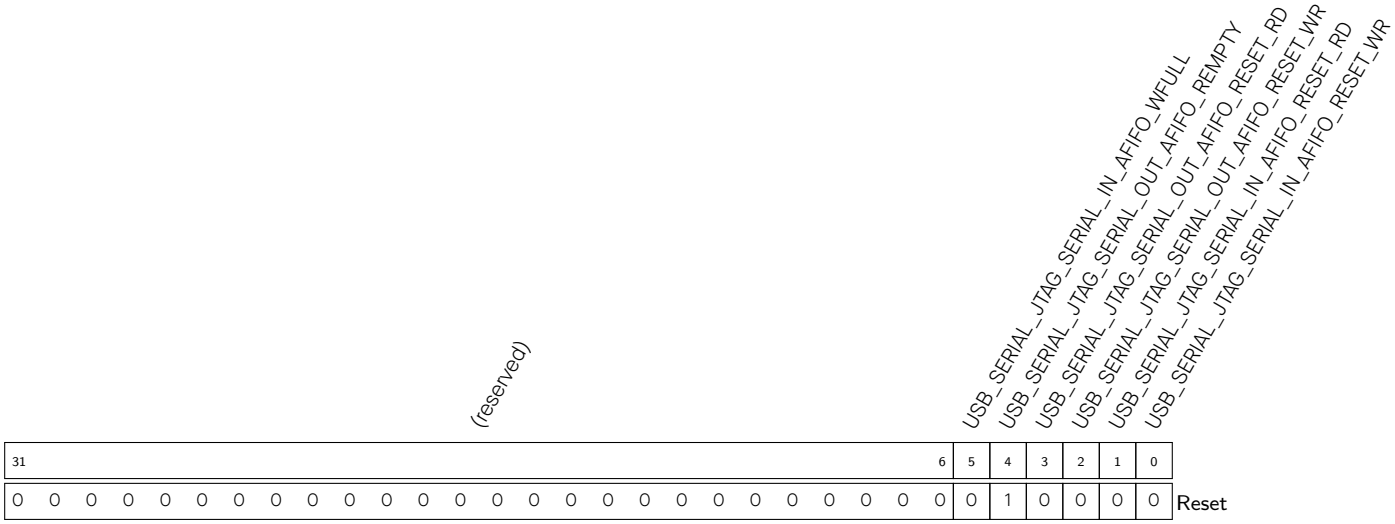
USB_SERIAL_JTAG_CONFIG_UPDATE Configures whether to update the value of configuration registers from APB clock domain to 48 MHz clock domain.

0: No effect

1: Update

(WT)

Register 33.11. 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

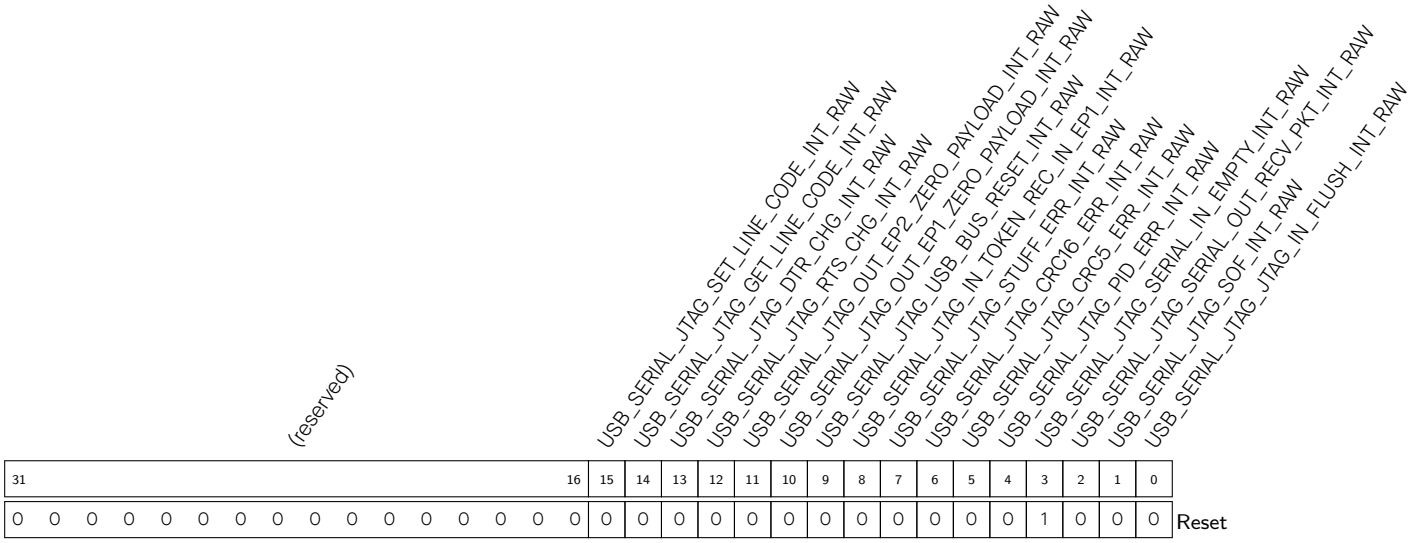
1: Reset

(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 33.12. USB_SERIAL_JTAG_INT_RAW_REG (0x0008)



USB_SERIAL_JTAG_JTAG_IN_FLUSH_INT_RAW The raw interrupt status of USB_SERIAL_JTAG_JTAG_IN_FLUSH_INT. (R/WTC/SS)

USB_SERIAL_JTAG_SOF_INT_RAW The raw interrupt status of USB_SERIAL_JTAG_SOF_INT. (R/WTC/SS)

USB_SERIAL_JTAG_SERIAL_OUT_RECV_PKT_INT_RAW The raw interrupt status of USB_SERIAL_JTAG_SERIAL_OUT_RECV_PKT_INT. (R/WTC/SS)

USB_SERIAL_JTAG_SERIAL_IN_EMPTY_INT_RAW The raw interrupt status of USB_SERIAL_JTAG_SERIAL_IN_EMPTY_INT. (R/WTC/SS)

USB_SERIAL_JTAG_PID_ERR_INT_RAW The raw interrupt status of USB_SERIAL_JTAG_PID_ERR_INT. (R/WTC/SS)

USB_SERIAL_JTAG_CRC5_ERR_INT_RAW The raw interrupt status of USB_SERIAL_JTAG_CRC5_ERR_INT. (R/WTC/SS)

USB_SERIAL_JTAG_CRC16_ERR_INT_RAW The raw interrupt status of USB_SERIAL_JTAG_CRC16_ERR_INT. (R/WTC/SS)

USB_SERIAL_JTAG_STUFF_ERR_INT_RAW The raw interrupt status of USB_SERIAL_JTAG_STUFF_ERR_INT. (R/WTC/SS)

USB_SERIAL_JTAG_IN_TOKEN_REC_IN_EP1_INT_RAW The raw interrupt status of USB_SERIAL_JTAG_IN_TOKEN_REC_IN_EP1_INT. (R/WTC/SS)

USB_SERIAL_JTAG_USB_BUS_RESET_INT_RAW The raw interrupt status of USB_SERIAL_JTAG_USB_BUS_RESET_INT. (R/WTC/SS)

USB_SERIAL_JTAG_OUT_EP1_ZERO_PAYLOAD_INT_RAW The raw interrupt status of USB_SERIAL_JTAG_OUT_EP1_ZERO_PAYLOAD_INT. (R/WTC/SS)

Continued on the next page...

Register 33.12. USB_SERIAL_JTAG_INT_RAW_REG (0x0008)

Continued from the previous page...

USB_SERIAL_JTAG_OUT_EP2_ZERO_PAYLOAD_INT_RAW The raw interrupt status of USB_SERIAL_JTAG_OUT_EP2_ZERO_PAYLOAD_INT. (R/WTC/SS)

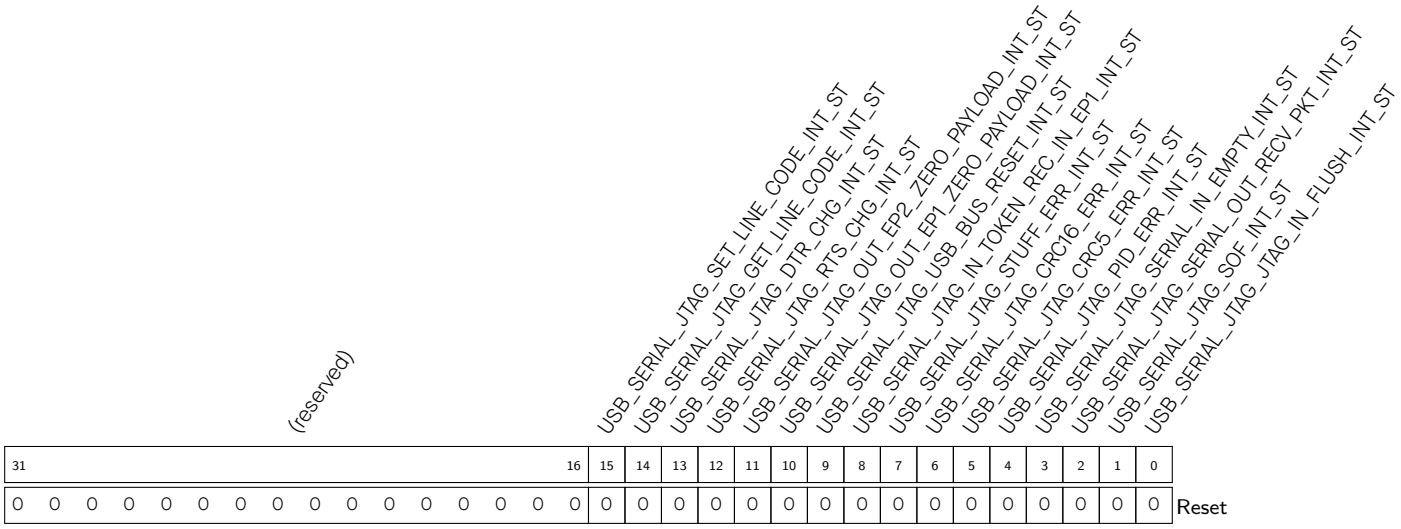
USB_SERIAL_JTAG_RTS_CHG_INT_RAW The raw interrupt status of USB_SERIAL_JTAG_RTS_CHG_INT. (R/WTC/SS)

USB_SERIAL_JTAG_DTR_CHG_INT_RAW The raw interrupt status of USB_SERIAL_JTAG_DTR_CHG_INT. (R/WTC/SS)

USB_SERIAL_JTAG_GET_LINE_CODE_INT_RAW The raw interrupt status of USB_SERIAL_JTAG_GET_LINE_CODE_INT. (R/WTC/SS)

USB_SERIAL_JTAG_SET_LINE_CODE_INT_RAW The raw interrupt status of USB_SERIAL_JTAG_SET_LINE_CODE_INT. (R/WTC/SS)

Register 33.13. USB_SERIAL_JTAG_INT_ST_REG (0x000C)



USB_SERIAL_JTAG_JTAG_IN_FLUSH_INT_ST The masked interrupt status of USB_SERIAL_JTAG_JTAG_IN_FLUSH_INT. (RO)

USB_SERIAL_JTAG_SOF_INT_ST The masked interrupt status of USB_SERIAL_JTAG_SOF_INT. (RO)

USB_SERIAL_JTAG_SERIAL_OUT_RECV_PKT_INT_ST The masked interrupt status of the USB_SERIAL_JTAG_SERIAL_OUT_RECV_PKT_INT. (RO)

USB_SERIAL_JTAG_SERIAL_IN_EMPTY_INT_ST The masked interrupt status of USB_SERIAL_JTAG_SERIAL_IN_EMPTY_INT. (RO)

USB_SERIAL_JTAG_PID_ERR_INT_ST The masked interrupt status of USB_SERIAL_JTAG_PID_ERR_INT. (RO)

USB_SERIAL_JTAG_CRC5_ERR_INT_ST The masked interrupt status of USB_SERIAL_JTAG_CRC5_ERR_INT. (RO)

USB_SERIAL_JTAG_CRC16_ERR_INT_ST The masked interrupt status of USB_SERIAL_JTAG_CRC16_ERR_INT. (RO)

USB_SERIAL_JTAG_STUFF_ERR_INT_ST The masked interrupt status of USB_SERIAL_JTAG_STUFF_ERR_INT. (RO)

USB_SERIAL_JTAG_IN_TOKEN_REC_IN_EP1_INT_ST The masked interrupt status of USB_SERIAL_JTAG_IN_TOKEN_REC_IN_EP1_INT. (RO)

USB_SERIAL_JTAG_USB_BUS_RESET_INT_ST The masked interrupt status of USB_SERIAL_JTAG_USB_BUS_RESET_INT. (RO)

USB_SERIAL_JTAG_OUT_EP1_ZERO_PAYLOAD_INT_ST The masked interrupt status of USB_SERIAL_JTAG_OUT_EP1_ZERO_PAYLOAD_INT (RO)

USB_SERIAL_JTAG_OUT_EP2_ZERO_PAYLOAD_INT_ST The masked interrupt status of USB_SERIAL_JTAG_OUT_EP2_ZERO_PAYLOAD_INT. (RO)

Continued on the next page...

Register 33.13. USB_SERIAL_JTAG_INT_ST_REG (0x000C)

Continued from the previous page...

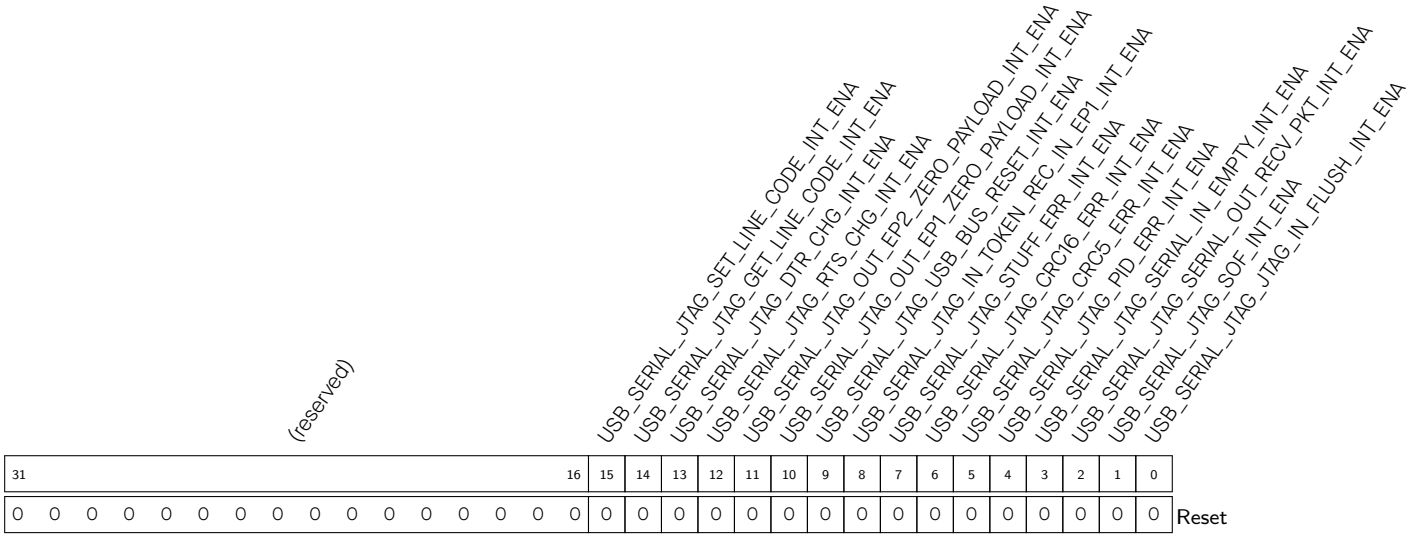
USB_SERIAL_JTAG_RTS_CHG_INT_ST The masked interrupt status of USB_SERIAL_JTAG_RTS_CHG_INT. (RO)

USB_SERIAL_JTAG_DTR_CHG_INT_ST The masked interrupt status of USB_SERIAL_JTAG_DTR_CHG_INT. (RO)

USB_SERIAL_JTAG_GET_LINE_CODE_INT_ST The masked interrupt status of USB_SERIAL_JTAG_GET_LINE_CODE_INT. (RO)

USB_SERIAL_JTAG_SET_LINE_CODE_INT_ST The masked interrupt status of USB_SERIAL_JTAG_SET_LINE_CODE_INT. (RO)

Register 33.14. USB_SERIAL_JTAG_INT_ENA_REG (0x0010)



USB_SERIAL_JTAG_JTAG_IN_FLUSH_INT_ENA Write 1 to enable USB_SERIAL_JTAG_JTAG_IN_FLUSH_INT. (R/W)

USB_SERIAL_JTAG_SOF_INT_ENA Write 1 to enable USB_SERIAL_JTAG_SOF_INT. (R/W)

USB_SERIAL_JTAG_SERIAL_OUT_RECV_PKT_INT_ENA Write 1 to enable USB_SERIAL_JTAG_SERIAL_OUT_RECV_PKT_INT. (R/W)

USB_SERIAL_JTAG_SERIAL_IN_EMPTY_INT_ENA Write 1 to enable USB_SERIAL_JTAG_SERIAL_IN_EMPTY_INT. (R/W)

USB_SERIAL_JTAG_PID_ERR_INT_ENA Write 1 to enable USB_SERIAL_JTAG_PID_ERR_INT. (R/W)

USB_SERIAL_JTAG_CRC5_ERR_INT_ENA Write 1 to enable USB_SERIAL_JTAG_CRC5_ERR_INT. (R/W)

USB_SERIAL_JTAG_CRC16_ERR_INT_ENA Write 1 to enable USB_SERIAL_JTAG_CRC16_ERR_INT. (R/W)

USB_SERIAL_JTAG_STUFF_ERR_INT_ENA Write 1 to enable USB_SERIAL_JTAG_STUFF_ERR_INT. (R/W)

USB_SERIAL_JTAG_IN_TOKEN_REC_IN_EP1_INT_ENA Write 1 to enable USB_SERIAL_JTAG_IN_TOKEN_REC_IN_EP1_INT. (R/W)

USB_SERIAL_JTAG_USB_BUS_RESET_INT_ENA Write 1 to enable USB_SERIAL_JTAG_USB_BUS_RESET_INT. (R/W)

USB_SERIAL_JTAG_OUT_EP1_ZERO_PAYLOAD_INT_ENA Write 1 to enable USB_SERIAL_JTAG_OUT_EP1_ZERO_PAYLOAD_INT. (R/W)

USB_SERIAL_JTAG_OUT_EP2_ZERO_PAYLOAD_INT_ENA Write 1 to enable USB_SERIAL_JTAG_OUT_EP2_ZERO_PAYLOAD_INT. (R/W)

Continued on the next page...

Register 33.14. USB_SERIAL_JTAG_INT_ENA_REG (0x0010)

Continued from the previous page...

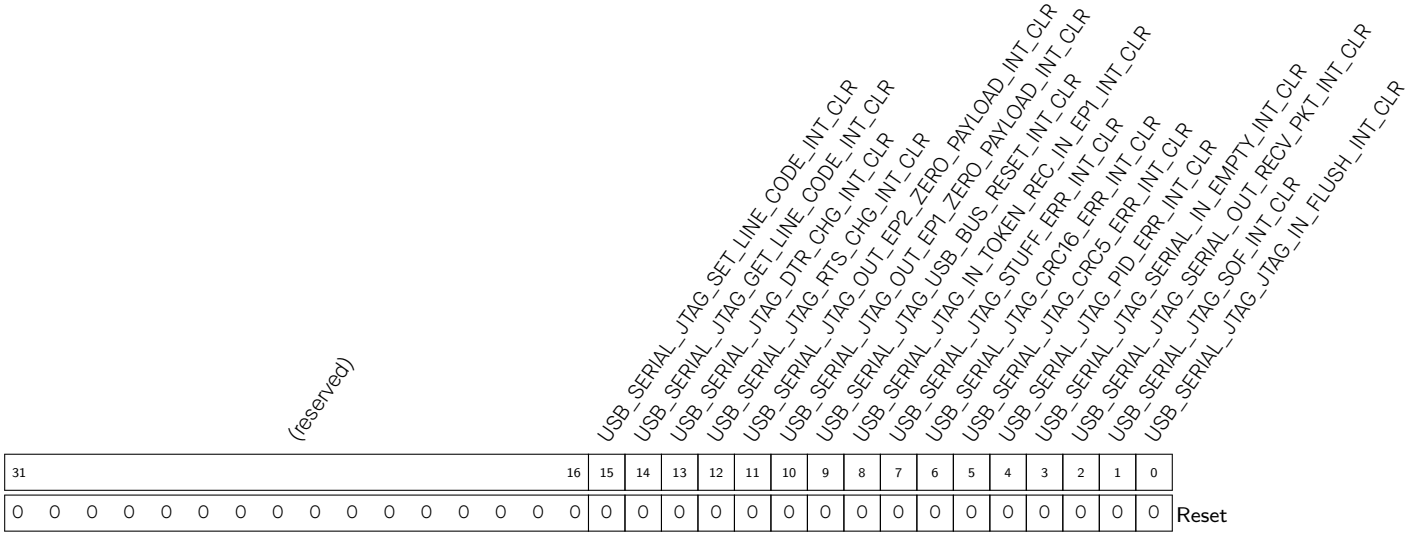
USB_SERIAL_JTAG_RTS_CHG_INT_ENA Write 1 to enable USB_SERIAL_JTAG_RTS_CHG_INT. (R/W)

USB_SERIAL_JTAG_DTR_CHG_INT_ENA Write 1 to enable USB_SERIAL_JTAG_DTR_CHG_INT. (R/W)

USB_SERIAL_JTAG_GET_LINE_CODE_INT_ENA Write 1 to enable USB_SERIAL_JTAG_GET_LINE_CODE_INT. (R/W)

USB_SERIAL_JTAG_SET_LINE_CODE_INT_ENA Write 1 to enable USB_SERIAL_JTAG_SET_LINE_CODE_INT. (R/W)

Register 33.15. USB_SERIAL_JTAG_INT_CLR_REG (0x0014)



USB_SERIAL_JTAG_JTAG_IN_FLUSH_INT_CLR Write 1 to clear USB_SERIAL_JTAG_JTAG_IN_FLUSH_INT. (WT)

USB_SERIAL_JTAG_SOF_INT_CLR Write 1 to clear USB_SERIAL_JTAG_SOF_INT. (WT)

USB_SERIAL_JTAG_SERIAL_OUT_RECV_PKT_INT_CLR Write 1 to clear USB_SERIAL_JTAG_SERIAL_OUT_RECV_PKT_INT. (WT)

USB_SERIAL_JTAG_SERIAL_IN_EMPTY_INT_CLR Write 1 to clear USB_SERIAL_JTAG_SERIAL_IN_EMPTY_INT. (WT)

USB_SERIAL_JTAG_PID_ERR_INT_CLR Write 1 to clear USB_SERIAL_JTAG_PID_ERR_INT. (WT)

USB_SERIAL_JTAG_CRC5_ERR_INT_CLR Write 1 to clear USB_SERIAL_JTAG_CRC5_ERR_INT. (WT)

USB_SERIAL_JTAG_CRC16_ERR_INT_CLR Write 1 to clear USB_SERIAL_JTAG_CRC16_ERR_INT. (WT)

USB_SERIAL_JTAG_STUFF_ERR_INT_CLR Write 1 to clear USB_SERIAL_JTAG_STUFF_ERR_INT. (WT)

USB_SERIAL_JTAG_IN_TOKEN_REC_IN_EP1_INT_CLR Write 1 to clear USB_SERIAL_JTAG_IN_TOKEN_REC_IN_EP1_INT. (WT)

USB_SERIAL_JTAG_USB_BUS_RESET_INT_CLR Write 1 to clear USB_SERIAL_JTAG_USB_BUS_RESET_INT. (WT)

USB_SERIAL_JTAG_OUT_EP1_ZERO_PAYLOAD_INT_CLR Write 1 to clear USB_SERIAL_JTAG_OUT_EP1_ZERO_PAYLOAD_INT. (WT)

USB_SERIAL_JTAG_OUT_EP2_ZERO_PAYLOAD_INT_CLR Write 1 to clear USB_SERIAL_JTAG_OUT_EP2_ZERO_PAYLOAD_INT. (WT)

Continued on the next page...

Register 33.15. USB_SERIAL_JTAG_INT_CLR_REG (0x0014)

Continued from the previous page...

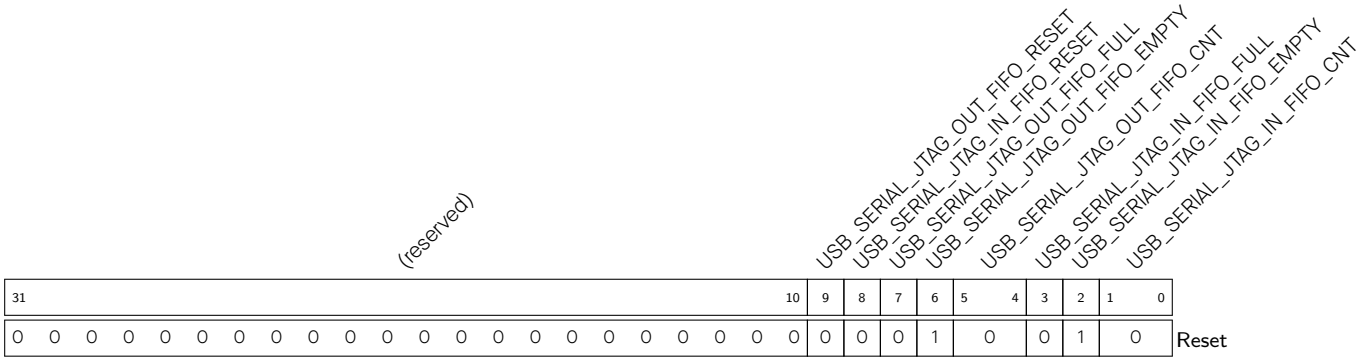
USB_SERIAL_JTAG_RTS_CHG_INT_CLR Write 1 to clear USB_SERIAL_JTAG_RTS_CHG_INT. (WT)

USB_SERIAL_JTAG_DTR_CHG_INT_CLR Write 1 to clear USB_SERIAL_JTAG_DTR_CHG_INT. (WT)

USB_SERIAL_JTAG_GET_LINE_CODE_INT_CLR Write 1 to clear USB_SERIAL_JTAG_GET_LINE_CODE_INT. (WT)

USB_SERIAL_JTAG_SET_LINE_CODE_INT_CLR Write 1 to clear USB_SERIAL_JTAG_SET_LINE_CODE_INT. (WT)

Register 33.16. USB_SERIAL_JTAG_JFIFO_ST_REG (0x0020)



USB_SERIAL_JTAG_IN_FIFO_CNT Represents JTAG IN FIFO counter. (RO)

USB_SERIAL_JTAG_IN_FIFO_EMPTY Represents whether JTAG IN FIFO is empty.

0: Not empty

1: Empty

(RO)

USB_SERIAL_JTAG_IN_FIFO_FULL Represents whether JTAG IN FIFO is full.

0: Not full

1: Full

(RO)

USB_SERIAL_JTAG_OUT_FIFO_CNT Represents JTAG OUT FIFO counter. (RO)

USB_SERIAL_JTAG_OUT_FIFO_EMPTY Represents whether JTAG OUT FIFO is empty.

0: Not empty

1: Empty

(RO)

USB_SERIAL_JTAG_OUT_FIFO_FULL Represents whether JTAG OUT FIFO is full.

0: Not full

1: Full

(RO)

USB_SERIAL_JTAG_IN_FIFO_RESET Configures whether to reset JTAG IN FIFO.

0: No effect

1: Reset

(R/W)

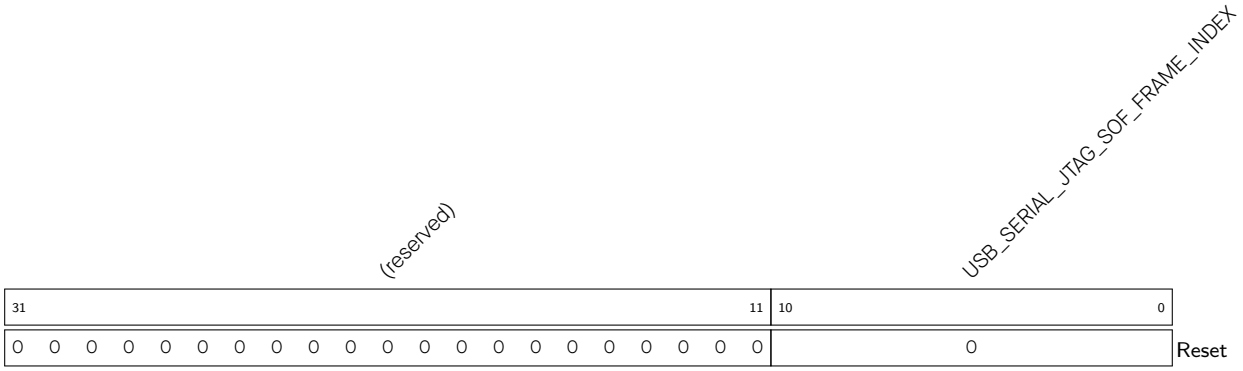
USB_SERIAL_JTAG_OUT_FIFO_RESET Configures whether to reset JTAG OUT FIFO.

0: No effect

1: Reset

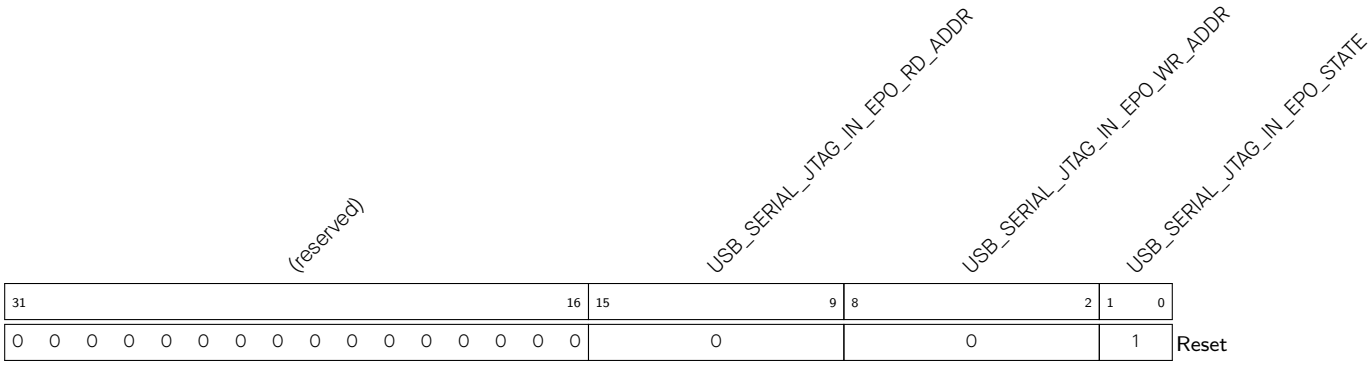
(R/W)

Register 33.17. USB_SERIAL_JTAG_FRAM_NUM_REG (0x0024)



USB_SERIAL_JTAG_SOF_FRAME_INDEX Represents frame index of received SOF frame. (RO)

Register 33.18. USB_SERIAL_JTAG_IN_EP0_ST_REG (0x0028)

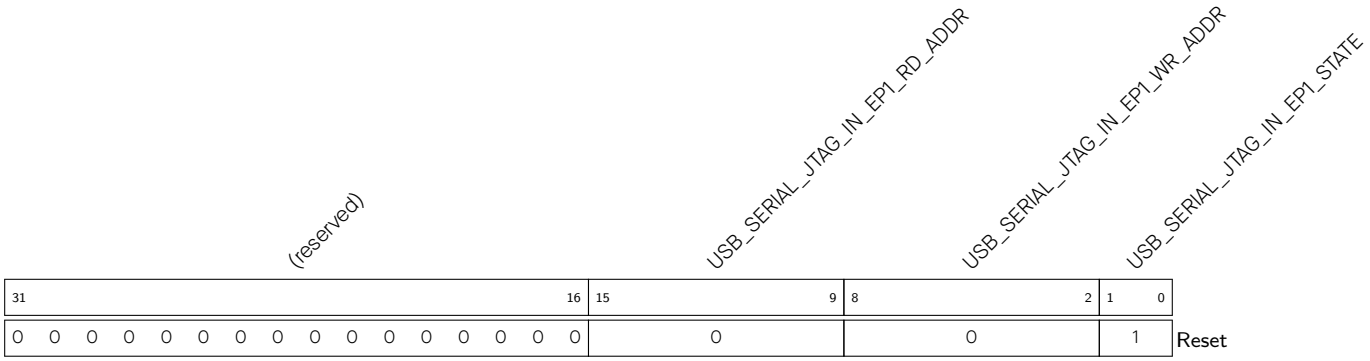


USB_SERIAL_JTAG_IN_EP0_STATE Represents state of IN endpoint 0. (RO)

USB_SERIAL_JTAG_IN_EP0_WR_ADDR Represents write data address of IN endpoint 0. (RO)

USB_SERIAL_JTAG_IN_EP0_RD_ADDR Represents read data address of IN endpoint 0. (RO)

Register 33.19. USB_SERIAL_JTAG_IN_EP1_ST_REG (0x002C)

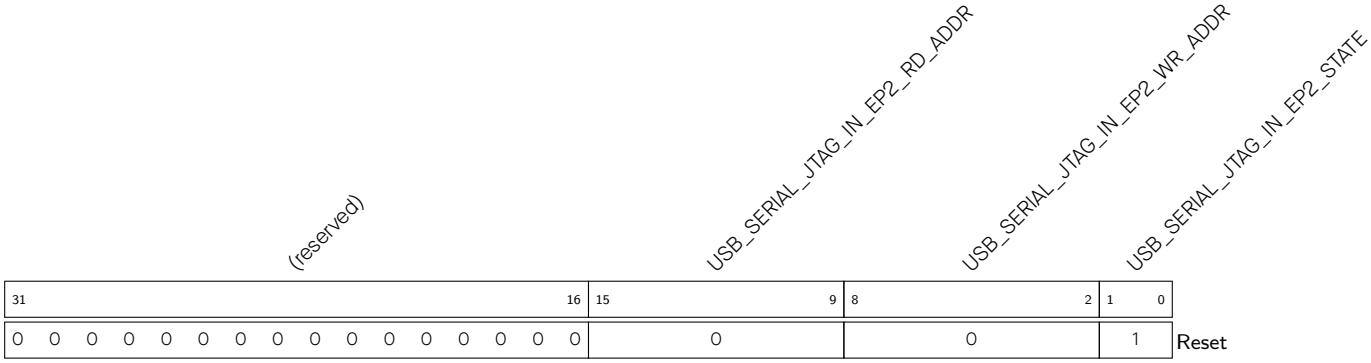


USB_SERIAL_JTAG_IN_EP1_STATE Represents state of IN endpoint 1. (RO)

USB_SERIAL_JTAG_IN_EP1_WR_ADDR Represents write data address of IN endpoint 1. (RO)

USB_SERIAL_JTAG_IN_EP1_RD_ADDR Represents read data address of IN endpoint 1. (RO)

Register 33.20. USB_SERIAL_JTAG_IN_EP2_ST_REG (0x0030)

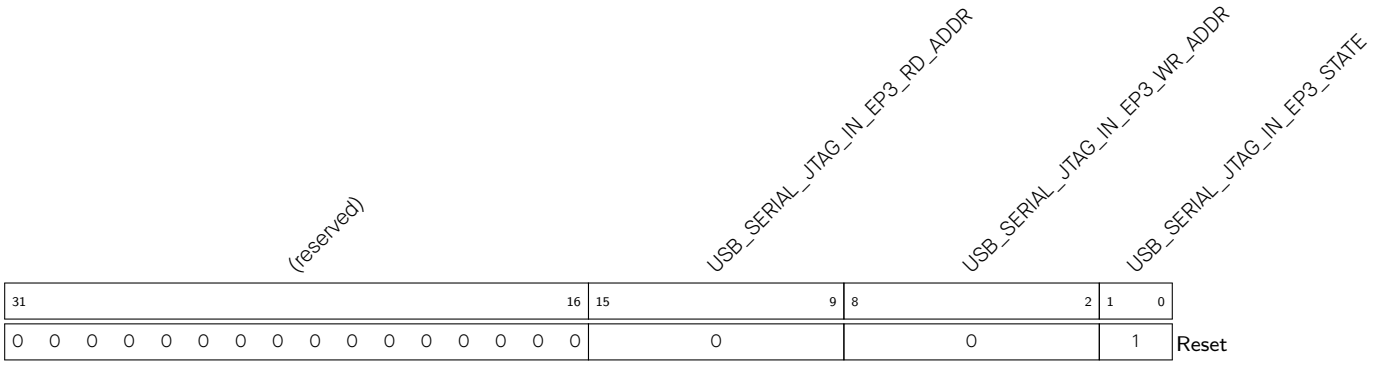


USB_SERIAL_JTAG_IN_EP2_STATE Represents state of IN endpoint 2. (RO)

USB_SERIAL_JTAG_IN_EP2_WR_ADDR Represents write data address of IN endpoint 2. (RO)

USB_SERIAL_JTAG_IN_EP2_RD_ADDR Represents read data address of IN endpoint 2. (RO)

Register 33.21. USB_SERIAL_JTAG_IN_EP3_ST_REG (0x0034)

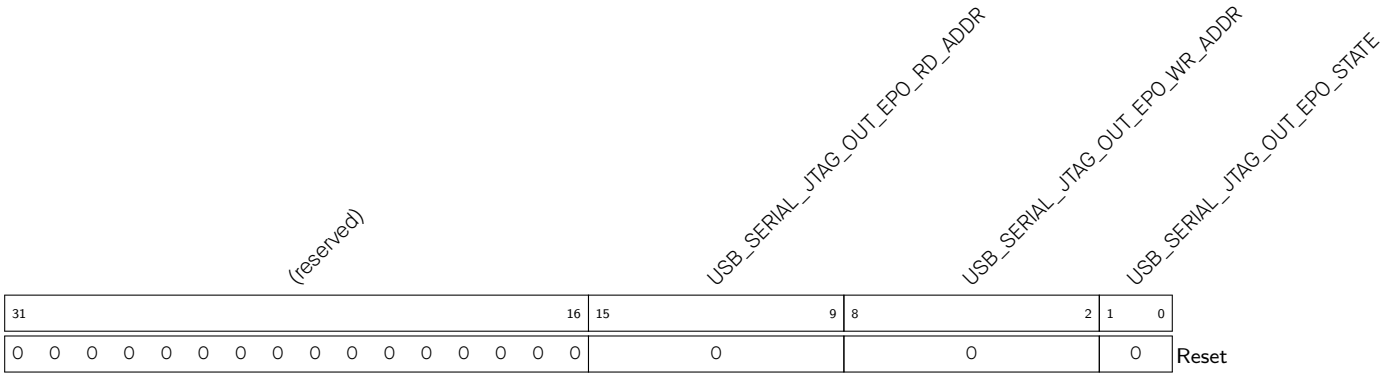


USB_SERIAL_JTAG_IN_EP3_STATE Represents state of IN endpoint 3. (RO)

USB_SERIAL_JTAG_IN_EP3_WR_ADDR Represents write data address of IN endpoint 3. (RO)

USB_SERIAL_JTAG_IN_EP3_RD_ADDR Represents read data address of IN endpoint 3. (RO)

Register 33.22. USB_SERIAL_JTAG_OUT_EP0_ST_REG (0x0038)



USB_SERIAL_JTAG_OUT_EP0_STATE Represents state of OUT endpoint 0. (RO)

USB_SERIAL_JTAG_OUT_EP0_WR_ADDR Represents write data address of OUT endpoint 0. When USB_SERIAL_JTAG_SERIAL_OUT_RECV_PKT_INT is detected, there are (USB_SERIAL_JTAG_OUT_EP0_WR_ADDR − 2) bytes data in OUT endpoint 0. (RO)

USB_SERIAL_JTAG_OUT_EP0_RD_ADDR Represents read data address of OUT endpoint 0. (RO)

Register 33.23. USB_SERIAL_JTAG_OUT_EP1_ST_REG (0x003C)

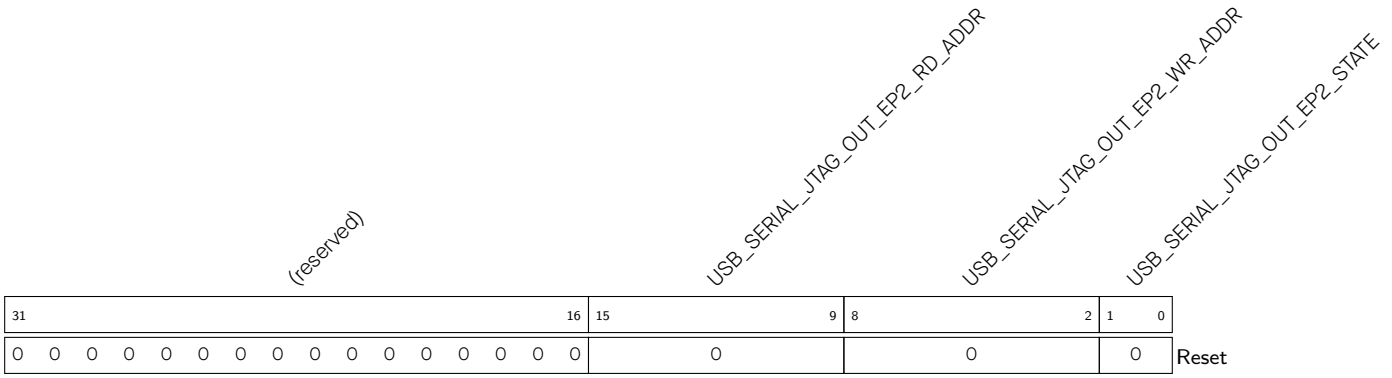
USB_SERIAL_JTAG_OUT_EP1_STATE Represents state of OUT endpoint 1. (RO)

USB_SERIAL_JTAG_OUT_EP1_WR_ADDR Represents write data address of OUT endpoint 1. When USB_SERIAL_JTAG_SERIAL_OUT_RECV_PKT_INT is detected, there are (USB_SERIAL_JTAG_OUT_EP1_WR_ADDR − 2) bytes data in OUT endpoint 1. (RO)

USB_SERIAL_JTAG_OUT_EP1_RD_ADDR Represents read data address of OUT endpoint 1. (RO)

USB_SERIAL_JTAG_OUT_EP1_REC_DATA_CNT Represents data count in OUT endpoint 1 when one packet is received. (RO)

Register 33.24. USB_SERIAL_JTAG_OUT_EP2_ST_REG (0x0040)

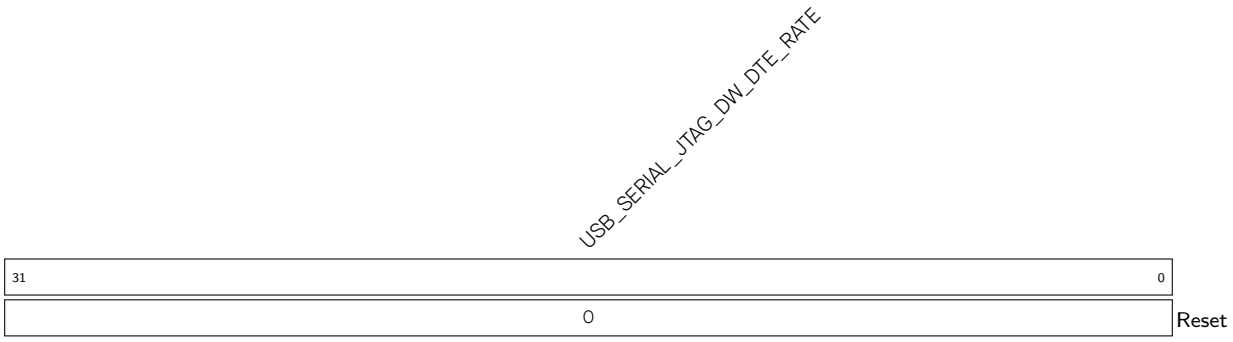


USB_SERIAL_JTAG_OUT_EP2_STATE Represents state of OUT endpoint 2. (RO)

USB_SERIAL_JTAG_OUT_EP2_WR_ADDR Represents write data address of OUT endpoint 2. When USB_SERIAL_JTAG_SERIAL_OUT_RECV_PKT_INT is detected there are (USB_SERIAL_JTAG_OUT_EP2_WR_ADDR − 2) bytes data in OUT endpoint 2. (RO)

USB_SERIAL_JTAG_OUT_EP2_RD_ADDR Represents read data address of OUT endpoint 2. (RO)

Register 33.25. USB_SERIAL_JTAG_SET_LINE_CODE_W0_REG (0x0050)



USB_SERIAL_JTAG_DW_DTE_RATE Represents the value of dwDTERate (in bits per second) set by host through SET_LINE_CODING command. Note that the value here does not affect the actual communication speed at all. (RO)

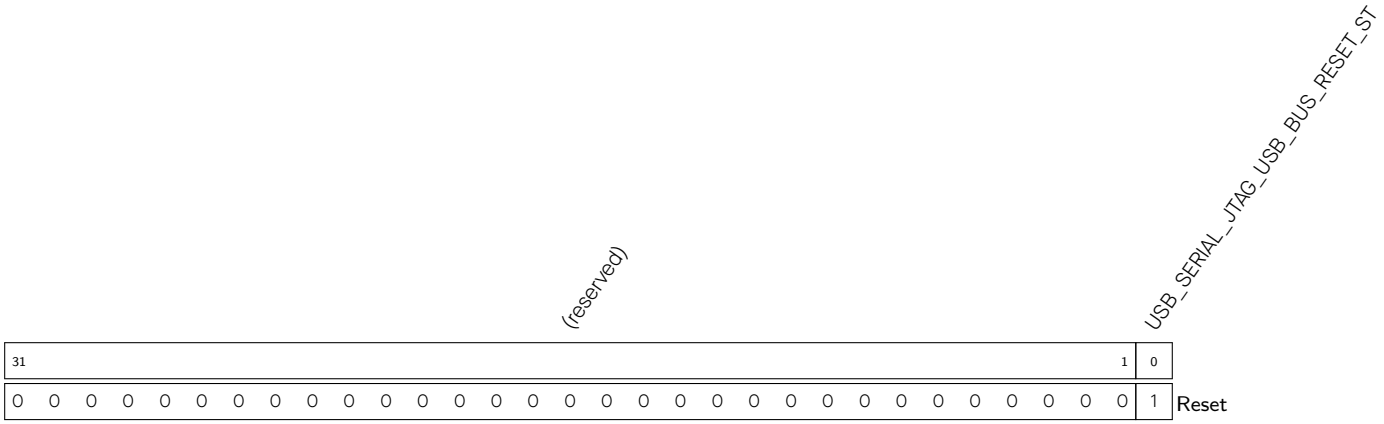
Register 33.26. USB_SERIAL_JTAG_SET_LINE_CODE_W1_REG (0x0054)

USB_SERIAL_JTAG_BCHAR_FORMAT Represents the value of bCharFormat set by host through SET_LINE_CODING command. (RO)

USB_SERIAL_JTAG_BPARITY_TYPE Represents the value of bParityTpye set by host through SET_LINE_CODING command. (RO)

USB_SERIAL_JTAG_BDATA_BITS Represents the value of bDataBits set by host through SET_LINE_CODING command. (RO)

Register 33.27. USB_SERIAL_JTAG_BUS_RESET_ST_REG (0x0068)



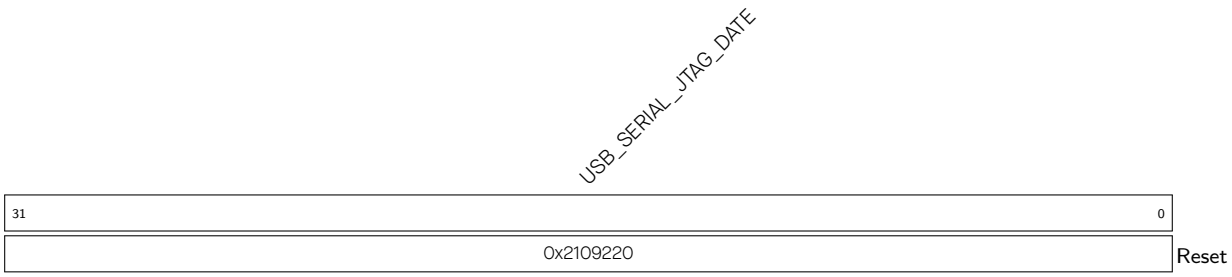
USB_SERIAL_JTAG_USB_BUS_RESET_ST Represents whether USB bus reset is released.

0: USB Serial/JTAG is in USB bus reset status

1: USB bus reset is released

(RO)

Register 33.28. USB_SERIAL_JTAG_DATE_REG (0x0080)



USB_SERIAL_JTAG_DATE Version control register.(R/W)

Chapter 34

Two-wire Automotive Interface (TWAI)

The Two-wire Automotive Interface (TWAI®) is a multi-master, multi-cast communication protocol with functions such as error detection and signaling and inbuilt message priorities and arbitration. The TWAI protocol is suited for automotive and industrial applications (see Section 34.2 for more details).

ESP32-H2 contains one TWAI controller. The controller can be connected to a TWAI bus via an external transceiver. The TWAI controller contains numerous advanced features and can be utilized in a wide range of use cases, such as automotive products, industrial automation controls, building automation, etc.

34.1 Features

The TWAI controller on ESP32-H2 supports the following features:

- Compatibility with ISO 11898-1 protocol (CAN Specification 2.0)
- Standard Frame Format (11-bit ID) and Extended Frame Format (29-bit ID)
- Bit rates from 1 Kbit/s to 1 Mbit/s
- Multiple modes of operation:
  - Normal
  - Listen-only (no influence on bus)
  - Self-test (no acknowledgment required during data transmission)
- 64-byte Receive FIFO
- Special transmissions:
  - Single-shot transmissions (does not automatically re-transmit upon error)
  - Self-reception (the TWAI controller transmits and receives messages simultaneously)
- Acceptance Filter (supports single and dual filter modes)
- Error detection and handling:
  - Error counters
  - Configurable error warning limit
  - Error code capture
  - Arbitration lost capture
  - Automatic transceiver standby

34.2 Protocol Overview

34.2.1 TWAI Properties

The TWAI protocol connects two or more nodes in a bus network, and allows nodes to exchange messages with deterministic delay. A TWAI bus has the following properties:

Single Channel and Non-Return-to-Zero: The bus has only one transmission line for single-channel communication, which is half-duplex. Synchronization is also implemented in this channel, so extra channels (e.g., clock or enable) are not required. The bit stream of a TWAI message is encoded using the Non-Return-to-Zero (NRZ) method.

Bit Values: The single channel can either be in a dominant or recessive state, representing a logical 0 and a logical 1 respectively. A node transmitting data in a dominant state always overrides the other node transmitting data in a recessive state. The physical implementation on the bus is left to the application level to decide (e.g., differential pair or a single wire).

Bit Stuffing: Certain fields of TWAI messages are bit-stuffed. A transmitter that transmits five consecutive bits of the same value (e.g., dominant value or recessive value) should automatically insert a complementary bit. Likewise, a receiver that receives five consecutive bits of the same value should treat the next bit as a stuffed bit. Bit stuffing is applied to the following fields: SOF, arbitration field, control field, data field, and CRC sequence (see Section 34.2.2 for more details).

Multi-Cast: All nodes receive the same bits as they are connected to the same bus. Data is consistent across all nodes unless there is a bus error (see Section 34.2.3 for more details).

Multi-Master: Any node can initiate a transmission. If a transmission is already ongoing, a node will wait until the current transmission is over before initiating a new transmission.

Message Priority and Arbitration: If two or more nodes simultaneously initiate a transmission, the TWAI protocol ensures that one node will win arbitration of the bus. The arbitration field of the message transmitted by each node is used to determine which node will win the arbitration.

Error Detection and Signaling: Each node actively monitors the bus for errors, and signals the detected errors by transmitting an error frame.

Fault Confinement: Each node maintains a set of error counters that are incremented/decremented according to a set of rules. When the error counters surpass a certain threshold, the node will automatically eliminate itself from the network by switching itself off.

Configurable Bit Rate: The bit rate for a single TWAI bus is configurable. However, all nodes on the same bus must operate at the same bit rate.

Transmitters and Receivers: At any point in time, a TWAI node can either be a transmitter or a receiver.

- A node generating a message is a transmitter. The node remains a transmitter until the bus is idle or until the node loses arbitration. Please note that there could be multiple nodes that act as transmitters during arbitration.
- All nodes that are not transmitters are receivers.

34.2.2 TWAI Messages

TWAI nodes use messages to transmit data, and signal errors to other nodes when detecting errors on the bus. Messages are split into various frame types, and different frame types have different frame formats.

The TWAI protocol has the following frame types:

- Data frame
- Remote frame
- Error frame
- Overload frame
- Interframe space

The TWAI protocol supports the following frame formats:

- Standard Frame Format (SFF) that uses an 11-bit identifier
- Extended Frame Format (EFF) that uses a 29-bit identifier

34.2.2.1 Data Frames and Remote Frames

Data frames are used by nodes to send data to other nodes, and can have a payload of 0 to 8 data bytes. Remote frames are used for nodes to request a data frame with the same identifier from other nodes, and thus they do not contain any data bytes. However, data frames and remote frames share many fields. Figure 34.2-1 illustrates the fields and sub-fields of different frames and formats.

Arbitration Field

When two or more nodes transmit data or remote frames simultaneously, the arbitration field is used to determine which node will win arbitration of the bus. In the arbitration field, if a node transmits a recessive bit while detecting a dominant bit, it indicates that another node has overridden its recessive bit. Therefore, the node transmitting the recessive bit has lost arbitration of the bus and should immediately switch to be a receiver.

The arbitration field primarily consists of a frame identifier that is transmitted from the most significant bit first. Given that a dominant bit represents a logical 0, and a recessive bit represents a logical 1:

- A frame with the smallest ID value always wins arbitration.
- Given the same ID and format, data frames always prevail over remote frames due to their RTR bits being dominant.
- Given the same first 11 bits of ID, a Standard Format Data Frame always prevails over an Extended Format Data Frame due to its SRR bits being recessive.

Control Field

The control field primarily consists of the Data Length Code (DLC) which indicates the number of payload data bytes for a data frame, or the number of requested data bytes for a remote frame. The DLC is transmitted from the most significant bit first.

Data Field

The data field contains the actual payload data bytes of a data frame. Remote frames do not contain any data field.

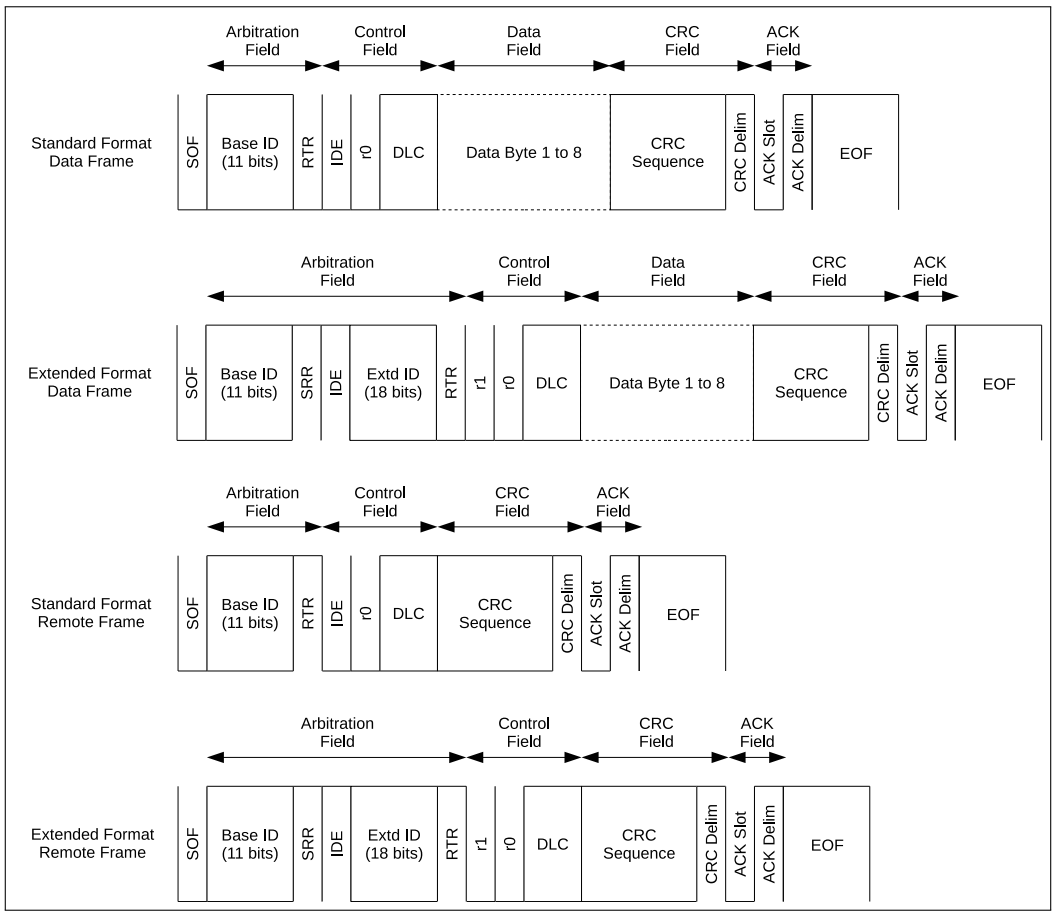


Figure 34.2-1. Bit Fields in Data Frames and Remote Frames

CRC Field

The CRC field primarily consists of a CRC sequence. The CRC sequence is a 15-bit cyclic redundancy code calculated from the de-stuffed contents (everything from the SOF to the end of the data field) of a data or remote frame.

ACK Field

The ACK field primarily consists of an ACK Slot and an ACK Delim. The ACK field indicates that the receiver has received an effective message from the transmitter.

Table 34.2-1. Data Frames and Remote Frames in SFF and EFF

Data/Remote Frames	Description
SOF	The Start of Frame (SOF) is a single dominant bit used to synchronize nodes on the bus.
Base ID	The Base ID (ID.28 to ID.18) is the 11-bit identifier for SFF, or the first 11 bits of the 29-bit identifier for EFF.
RTR	The Remote Transmission Request (RTR) bit indicates whether the message is a data frame (dominant) or a remote frame (recessive). This means that a remote frame will always lose arbitration to a data frame if they have the same ID.

Cont'd on next page

Table 34.2-1 – cont'd from previous page

Data/Remote Frames	Description
SRR	The Substitute Remote Request (SRR) bit is transmitted in EFF to substitute for the RTR bit at the same position in SFF.
IDE	The Identifier Extension (IDE) bit indicates whether the message is SFF (dominant) or EFF (recessive). This means that an SFF frame will always win arbitration over an EFF frame if they have the same Base ID.
Extd ID	The Extended ID (ID.17 to ID.0) is the remaining 18 bits of the 29-bit identifier for EFF.
r1	The r1 bit (reserved bit 1) is always dominant.
r0	The r0 bit (reserved bit 0) is always dominant.
DLC	The DLC is 4-bit long and should contain any value from 0 to 8. Data frames use the DLC to indicate the number of data bytes in the data frame. Remote frames used the DLC to indicate the number of data bytes to request from another node.
Data Bytes	The data payload of data frames. The number of bytes should match the value of DLC. Data byte 0 is transmitted first, and each data byte is transmitted from the most significant bit.
CRC Sequence	The CRC sequence is a 15-bit cyclic redundancy code.
CRC Delim	The CRC Delimiter (CRC Delim) is a single recessive bit that follows the CRC sequence.
ACK Slot	The ACK Slot (Acknowledgment Slot) is intended for receiver nodes to indicate that the data or remote frame was received successfully. The transmitter node will send a recessive bit in the ACK Slot and receiver nodes should override the ACK Slot with a dominant bit if the frame was received without errors.
ACK Delim	The Acknowledgment Delimiter (ACK Delim) is a single recessive bit.
EOF	The End of Frame (EOF) marks the end of a data or remote frame, and consists of seven recessive bits.

34.2.2.2 Error and Overload Frames

Error Frames

Error frames are transmitted when a node detects a bus error. Error frames notably consist of an Error Flag which is made up of six consecutive bits of the same value, thus violating the bit-stuffing rule. Therefore, when a particular node detects a bus error and transmits an error frame, all other nodes will then detect a stuff error and transmit their own error frames in response. This has the effect of propagating the detection of a bus error across all nodes on the bus.

When a node detects a bus error, it will transmit an error frame starting from the next bit. However, when a node detects a CRC error, the error frame will start at the bit following the ACK Delim (see Section 34.2.3 for more details). The following Figure 34.2-2 shows different fields of an error frame:

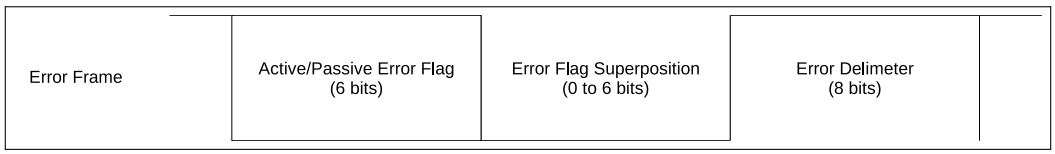


Figure 34.2-2. Fields of an Error Frame

Table 34.2-2. Error Frame

Error Frame	Description
Error Flag	The Error Flag has two forms, the Active Error Flag consisting of six dominant bits and the Passive Error Flag consisting of six recessive bits (unless overridden by dominant bits of other nodes). Active Error Flags are sent by error active nodes, whilst Passive Error Flags are sent by error passive nodes.
Error Flag Superposition	The Error Flag Superposition allows other nodes on the bus to transmit their respective Active Error Flags. The superposition field can range from 0 to 6 bits, and ends when the first recessive bit is detected (i.e., the first bit of the Delimiter).
Error Delimiter	The Delimiter field marks the end of the error/overload frame, and consists of eight recessive bits.

Overload Frames

An overload frame has the same bit fields as an error frame containing an Active Error Flag. The key difference is in the cases that can trigger the transmission of an overload frame. Figure 34.2-3 below shows the bit fields of an overload frame.

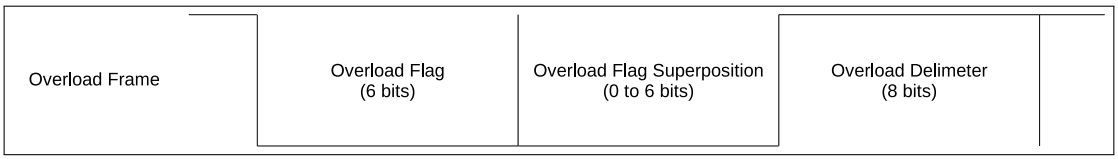


Figure 34.2-3. Fields of an Overload Frame

Table 34.2-3. Overload Frame

Overload Flag	Description
Overload Flag	The Overload Flag consists of six dominant bits. Same as an Active Error Flag.
Overload Flag Superposition	The Overload Flag Superposition allows the superposition of Overload Flags from other nodes. Similar to an Error Flag Superposition.
Overload Delimiter	The Overload Delimiter consists of eight recessive bits. Same as an Error Delimiter.

Overload frames will be transmitted in the following cases:

1. A receiver requires a delay of the next data or remote frame.
2. A dominant bit is detected at the first and second bit of intermission.

3. A dominant bit is detected at the eighth (last) bit of an Error Delimiter. Note that in this case, TEC and REC will not be incremented (see Section 34.2.3 for more details).

Transmitting an overload frame due to one of the above cases must also satisfy the following rules:

- The start of an overload frame due to case 1 is only allowed to be started at the first bit time of an expected intermission.
- The start of an overload frame due to case 2 and 3 is only allowed to be started one bit after detecting the dominant bit.
- For case 1, a maximum of two overload frames may be generated.

34.2.2.3 Interframe Space

The Interframe Space acts as a separator between frames. Data frames and remote frames must be separated from preceding frames by an Interframe Space, regardless of the preceding frame's type (data frame, remote frame, error frame, or overload frame). However, error frames and overload frames do not need to be separated from preceding frames.

Figure 34.2-4 shows the fields within an Interframe Space:

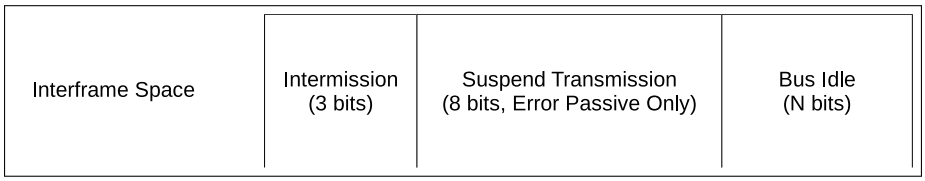


Figure 34.2-4. The Fields within an Interframe Space

Table 34.2-4. Interframe Space

Interframe Space	Description
Intermission	The Intermission field consists of three recessive bits.
Suspend Transmission	An Error Passive node that has just transmitted a message must include a Suspend Transmission field. This field consists of eight recessive bits. Error Active nodes should not include this field.
Bus Idle	The Bus Idle field is of arbitrary length. Bus Idle ends when an SOF is transmitted. If a node has a pending transmission, the SOF should be transmitted at the first bit following the Intermission.

34.2.3 TWAI Errors

34.2.3.1 Error Types

Bus Errors in TWAI are categorized into the following types:

Bit Error

A Bit Error occurs when a node transmits a bit value (i.e., dominant or recessive) but detects an opposite bit (e.g., a dominant bit is transmitted but a recessive is detected). However, if the transmitted bit is recessive and is located in the Arbitration Field, ACK Slot, or Passive Error Flag, then detecting a dominant bit will not be considered as a Bit Error.

Stuff Error

A Stuff Error occurs when six consecutive bits of the same value are detected (which violates the bit-stuffing encoding rules).

CRC Error

A receiver of a data or remote frame will calculate CRC based on the bits it has received. A CRC Error occurs when the CRC calculated by the receiver does not match the CRC sequence in the received data or remote Frame.

Format Error

A Format Error occurs when a format-fixed bit field of a message contains an illegal bit. For example, the r1 and r0 fields must be dominant.

ACK Error

An ACK Error occurs when a transmitter does not detect a dominant bit at the ACK Slot.

34.2.3.2 Error States

TWAI nodes implement fault confinement by maintaining two error counters in each node, where the counter values determine the error state. The two error counters are known as the Transmit Error Counter (TEC) and Receive Error Counter (REC). TWAI has the following error states:

Error Active

An Error Active node is able to participate in bus communication and transmit an Active Error Flag when it detects an error.

Error Passive

An Error Passive node is able to participate in bus communication and transmit a Passive Error Flag when it detects an error. Error Passive nodes that have transmitted data or remote frames must also include the Suspend Transmission field in the subsequent Interframe Space.

Bus Off

A Bus Off node is not permitted to influence the bus in any way (i.e., is not allowed to transmit data).

34.2.3.3 Error Counters

The TEC and REC are incremented/decremented according to the following rules. **Note that more than one rule can apply to a given message transfer.**

1. When a receiver detects an error, the REC is increased by 1, except when the detected error was a Bit Error during the transmission of an Active Error Flag or an Overload Flag.
2. When a receiver detects a dominant bit as the first bit after sending an Error Flag, the REC is increased by 8.
3. When a transmitter sends an Error Flag, the TEC is increased by 8. However, the following scenarios are exempt from this rule:
   - A transmitter is Error Passive and no dominant bit is detected when an Acknowledgment Error is detected and the Passive Error Flag is sent. In this case, the TEC should not be increased.
   - A transmitter transmits an Error Flag due to a Stuff Error during Arbitration. If the stuffed bit should have been recessive but was monitored as dominant, then the TEC should not be increased.

4. If a transmitter detects a Bit Error while sending an Active Error Flag or Overload Flag, the TEC is increased by 8.
5. If a receiver detects a Bit Error while sending an Active Error Flag or Overload Flag, the REC is increased by 8.
6. A node can tolerate up to 7 consecutive dominant bits after sending an Active/Passive Error Flag, or Overload Flag. After detecting the 14th consecutive dominant bit when sending an Active Error Flag or Overload Flag, or the 8th consecutive dominant bit following a Passive Error Flag, a transmitter will increase its TEC by 8 and a receiver will increase its REC by 8. Every additional 8 consecutive dominant bits will also increase the TEC for transmitters or REC for receivers by 8 as well.
7. When a transmitter has transmitted a message, which means getting ACK and no errors until the EOF is completed, the TEC is decremented by 1, unless the TEC is already at 0.
8. When a receiver successfully receives a message, which means getting no errors before ACK Slot and successfully sending ACK, the REC is decremented accordingly.
    - If the REC is between 1 and 127, it will be decremented by 1.
    - If the REC is greater than 127, it will be set to 127.
    - If the REC is 0, it will remain 0.
9. A node becomes Error Passive when its TEC and/or REC is greater than or equal to 128. Though the node becomes Error Passive, it still sends an Active Error Flag. Note that once the REC has reached 128, any further increases to its value are invalid until the REC returns to a value less than 128.
10. A node becomes Bus Off when its TEC is greater than or equal to 256.
11. An Error Passive node becomes Error Active when both the TEC and REC are less than or equal to 127.
12. A Bus Off node can become Error Active (with both its TEC and REC reset to 0) after it monitors 128 occurrences of 11 consecutive recessive bits on the bus.

34.2.4 TWAI Bit Timing

34.2.4.1 Nominal Bit

The TWAI protocol allows a TWAI bus to operate at a particular bit rate. However, all nodes within a TWAI bus must operate at the same bit rate.

- **The Nominal Bit Rate** is defined as the number of bits transmitted per second.
- **The Nominal Bit Time** is defined as **1/Nominal Bit Rate**.

A single Nominal Bit Time is divided into multiple segments, and each segment is made up of multiple Time Quanta. A **Time Quantum** is a minimum unit of time, and is implemented as some form of a prescaled clock signal in each node. Figure 34.2-5 illustrates the segments within a single Nominal Bit Time.

TWAI controller will operate in time steps of one Time Quanta where the state of the TWAI bus is analyzed. If the bus states in two consecutive Time Quanta are different (i.e., recessive to dominant or vice versa), an edge is generated. The intersection of PBS1 and PBS2 is considered the Sample Point and the sampled bus value is considered the value of that bit.

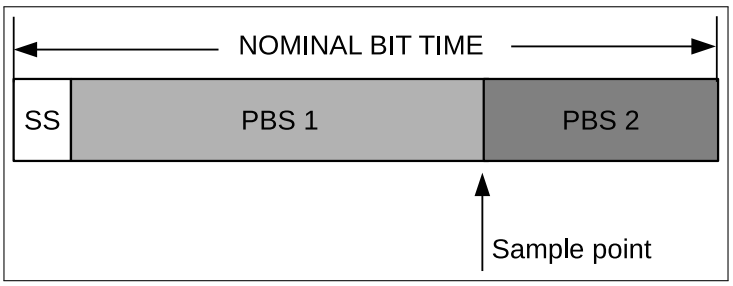


Figure 34.2-5. Layout of a Bit

Table 34.2-5. Segments of a Nominal Bit Time

Segment	Description
SS	The Synchronization Segment (SS) is 1 Time Quantum long. If all nodes are perfectly synchronized, the edge of a bit will lie in the SS.
PBS1	Phase Buffer Segment 1 (PBS1) can be 1 to 16 Time Quanta long. PBS1 is meant to compensate for the physical delay times within the network. PBS1 can also be lengthened for synchronization purposes.
PBS2	Phase Buffer Segment 2 (PBS2) can be 1 to 8 Time Quanta long. PBS2 is meant to compensate for the information processing time of nodes. PBS2 can also be shortened for synchronization purposes.

34.2.4.2 Hard Synchronization and Resynchronization

Due to clock skew and jitter, the bit timing of nodes on the same bus may become out of phase. Therefore, a bit edge may come before or after the SS. To ensure that the internal bit timing clocks of each node are kept in phase, TWAI has various methods of synchronization. The **Phase Error "e"** is measured in the number of Time Quanta and relative to the SS.

- A positive Phase Error (e > 0) is when the edge lies after the SS and before the Sample Point (i.e., the edge is late).
- A negative Phase Error (e < 0) is when the edge lies after the Sample Point of the previous bit and before SS (i.e., the edge is early).

To correct Phase Errors, there are two forms of synchronization, known as **Hard Synchronization** and **Resynchronization**. **Hard Synchronization** and **Resynchronization** obey the following rules:

- Only one synchronization may occur in a single bit time.
- Synchronizations only occur on recessive to dominant edges.

Hard Synchronization

Hard Synchronization occurs on the recessive to dominant (i.e., the first SOF bit after Bus Idle) edges when the bus is idle. All nodes will restart their internal bit timings so that the recessive to dominant edge lies within the SS of the restarted bit timing.

Resynchronization

Resynchronization occurs on recessive to dominant edges when the bus is not idle. If the edge has a positive Phase Error (e > 0), PBS1 is lengthened by a certain number of Time Quanta. If the edge has a negative Phase Error (e < 0), PBS2 will be shortened by a certain number of Time Quanta.

The number of Time Quanta to lengthen or shorten depends on the magnitude of the Phase Error, and is also limited by the Synchronization Jump Width (SJW) value which is programmable.

- When the magnitude of the Phase Error (**e**) is less than or equal to the SJW, PBS1/PBS2 are lengthened/shortened by the **e** number of Time Quanta. This has the same effect as Hard Synchronization.
- When the magnitude of the Phase Error is greater than the SJW, PBS1/PBS2 are lengthened/shortened by the SJW number of Time Quanta. This means it may take multiple bits of synchronization before the Phase Error is entirely corrected.

34.3 Architectural Overview

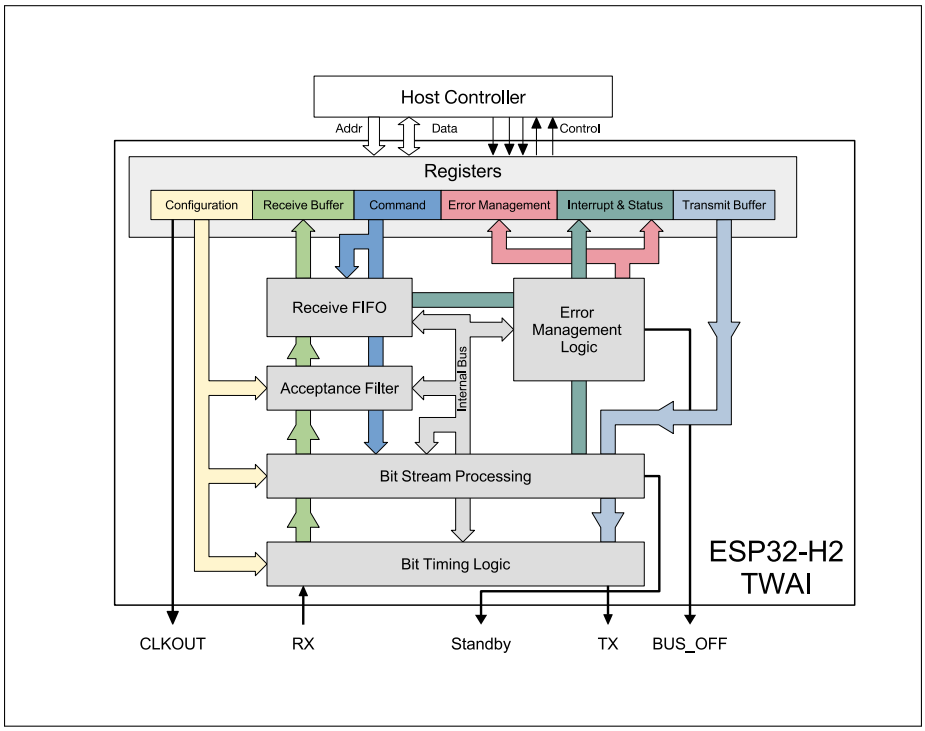


Figure 34.3-1. TWAI Overview Diagram

The major functional blocks of the TWAI controller are shown in Figure 34.3-1.

The TWAI controller only works in the CORE clock domain, where RC_FAST_CLK or crystal clock XTAL_CLK can be selected as the clock source. Users may configure PCR_TWAIO_FUNC_CLK_CONF_REG to select the clock source.

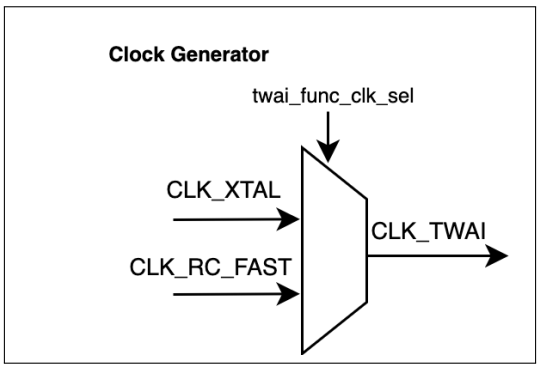


Figure 34.3-2. TWAI Clock Generation

34.3.1 Registers Block

The ESP32-H2 CPU accesses peripherals using 32-bit aligned words. However, the majority of registers in the TWAI controller only contain useful data at the least significant byte (bits [7:0]). Therefore, in these registers, bits [31:8] are ignored on writes, and return 0 on reads.

Configuration Registers

The configuration register stores various configuration items for the TWAI controller such as bit rates, Operation mode, Acceptance Filter, etc. Configuration registers can only be modified whilst the TWAI controller is in Reset mode (See Section 34.4.1).

Command Registers

The command register is used by the CPU to drive the TWAI controller to initiate certain actions such as transmitting a message or clearing the Receive Buffer. The command register can only be modified when the TWAI controller is in Operation mode (see Section 34.4.1).

Interrupt & Status Registers

The interrupt register indicates what events have occurred in the TWAI controller (each event is represented by a separate bit). The status register indicates the current status of the TWAI controller.

Error Management Registers

The error management register includes error counters and capture registers. Error counter registers represent TEC and REC values. Capture registers will record information about instances where the TWAI controller detects a bus error, or when it loses arbitration.

Transmit Buffer Registers

The transmit buffer is a 13-byte buffer used to store a TWAI message to be transmitted.

Receive Buffer Registers

The Receive Buffer is a 13-byte buffer which stores a single message. The Receive Buffer acts as a window of Receive FIFO, whose first message will be mapped into the Receive Buffer.

Note that the Transmit Buffer registers, Receive Buffer registers, and the Acceptance Filter registers share the same address range (offset 0x0040 to 0x0070). Their access is governed by the following rules:

- When the TWAI controller is in Reset mode, all reads and writes to the address range maps to the Acceptance Filter registers.
- When the TWAI controller is in Operation mode:

- All reads to the address range maps to the Receive Buffer registers.
- All writes to the address range maps to the Transmit Buffer registers.

34.3.2 Bit Stream Processor

The Bit Stream Processing (BSP) module handles the frame processing of data from the Transmit Buffer (e.g., bit stuffing and additional CRC fields) and generates a bit stream for the Bit Timing Logic (BTL) module. At the same time, the BSP module is also responsible for processing the received bit stream (e.g., de-stuffing and verifying CRC) from the BTL module and placing the message into the Receive FIFO. The BSP will also detect errors on the TWAI bus and report them to the Error Management Logic (EML).

34.3.3 Error Management Logic

The Error Management Logic (EML) module updates the TEC and REC, records error information like error types and positions, and updates the error state of the TWAI controller to ensure that the BSP module generates the correct Error Flags. Furthermore, this module also records the bit position when the TWAI controller loses arbitration.

34.3.4 Bit Timing Logic

The Bit Timing Logic (BTL) module transmits and receives messages at the configured bit rate. The BTL module also handles bit timing synchronization so that communication remains stable. A single bit time consists of multiple programmable segments that allow users to set the number of time quanta to adjust propagation delay and controller processing time, etc.

34.3.5 Acceptance Filter

The Acceptance Filter is a programmable message filtering unit that allows the TWAI controller to accept or reject a received message based on the message's ID field. Only accepted messages will be stored in the Receive FIFO. The Acceptance Filter can be configured to Single-filter mode or Dual-filter mode through registers.

34.3.6 Receive FIFO

The Receive FIFO is a 64-byte buffer (inside the TWAI controller) that stores received messages accepted by the Acceptance Filter. Messages in the Receive FIFO can vary in size (between 3 to 13 bytes). When the Receive FIFO is full or does not have enough space to store the currently received message in its entirety, the Overrun Interrupt will be triggered, and any subsequently received messages will be lost until adequate space is cleared in the Receive FIFO. The first message in the Receive FIFO will be mapped to the 13-byte Receive Buffer until that message is cleared (using the Release Receive Buffer command bit). After being cleared, the Receive Buffer will map to the next message in the Receive FIFO, and the space occupied by the previous message in the Receive FIFO can be used to receive new messages.

34.4 Functional Description

34.4.1 Modes

The ESP32-H2 TWAI controller has two working modes: Reset mode and Operation mode. Reset mode and Operation mode are entered by setting or clearing the TWAI_RESET_MODE bit.

34.4.1.1 Reset Mode

Entering Reset mode is required in order to modify the various configuration registers of the TWAI controller. When entering Reset mode, the TWAI controller is essentially disconnected from the TWAI bus. When in Reset mode, the TWAI controller will not be able to transmit any messages (including error signals). Any transmission in progress is immediately terminated. Likewise, the TWAI controller will not be able to receive any messages either.

34.4.1.2 Operation Mode

In Operation mode, the TWAI controller connects to the bus and write-protects all configuration registers to ensure consistency during operation. When in Operation mode, the TWAI controller can transmit and receive messages (including error signaling) depending on which operation sub-mode the TWAI controller was configured with. The TWAI controller supports the following operation sub-modes:

- **Normal mode:** The TWAI controller can transmit and receive messages including error signals (such as Error and Overload Frames).
- **Self-Test mode:** Self-test mode is similar to Normal mode, but the TWAI controller will consider the transmission of a data or remote frame successful and not generate an ACK error even if it was not acknowledged. This mode is commonly used during the self-test of a TWAI controller.
- **Listen-Only mode:** The TWAI controller will be able to receive messages, but will remain completely passive on the TWAI bus. Thus, the TWAI controller will not be able to transmit any messages, acknowledgments, or error signals. The error counters will remain frozen. This mode is useful for TWAI bus monitoring.

Note that when exiting Reset mode (i.e., entering Operation mode), the TWAI controller must wait for 11 consecutive recessive bits to occur before fully connecting to the TWAI bus (i.e., being able to transmit or receive).

34.4.2 Bit Timing

The operating bit rate of the TWAI controller must be configured whilst the TWAI controller is in Reset mode. The bit rate is configured using TWAI_BUS_TIMING_0_REG and TWAI_BUS_TIMING_1_REG, and the two registers contain the following fields:

The following Table 34.4-1 illustrates the bit fields of TWAI_BUS_TIMING_0_REG. The frequency of the TWAI core clock has multiple clock sources that can be configured by the user as needed. See Chapter 7 *Reset and Clock* for detailed configuration instructions.

Table 34.4-1. Bit Information of TWAI_BUS_TIMING_0_REG (0x18)

Bit 31-16	Bit 15	Bit 14	Bit 13	Bit 12	.....	Bit 1	Bit 0
Reserved	SJW.1	SJW.0	BRP.13	BRP.12	.....	BRP.1	BRP.0

Notes:

- BRP: The TWAI Time Quanta clock is derived from the XTAL clock (the default is 40 MHz and is configured). The Baud Rate Prescaler (BRP) field is used to define the prescaler according to the equation below, where t_{Tq} is the Time Quanta clock cycle and t_{CLK} is TWAI core clock cycle: $t_{Tq} = 2 \times t_{CLK} \times (2^{13} \times BRP.13 + 2^{12} \times BRP.12 + 2^{11} \times BRP.11 + ... + 2^{1} \times BRP.1 + 2^{0} \times BRP.0 + 1)$
- SJW: Synchronization Jump Width (SJW) is configured in SJW.0 and SJW.1 where SJW = (2 x SJW.1 + SJW.0 + 1).

The following Table 34.4-2 illustrates the bit fields of TWAI_BUS_TIMING_1_REG.

Table 34.4-2. Bit Information of TWAI_BUS_TIMING_1_REG (0x1c)

Bit 31-8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Reserved	SAM	PBS2.2	PBS2.1	PBS2.0	PBS1.3	PBS1.2	PBS1.1	PBS1.0

Notes:

- PBS1: The number of Time Quanta in Phase Buffer Segment 1 is defined according to the following equation: (8 x PBS1.3 + 4 x PBS1.2 + 2 x PBS1.1 + PBS1.0 + 1).
- PBS2: The number of Time Quanta in Phase Buffer Segment 2 is defined according to the following equation: (4 x PBS2.2 + 2 x PBS2.1 + PBS2.0 + 1).
- SAM: Enables triple sampling if set to 1. This is useful for low/medium speed buses to filter spikes on the bus line.

34.4.3 Interrupt Management

The ESP32-H2 TWAI controller provides eight interrupts, each represented by a single bit in TWAI_INT_ST_REG. For a particular interrupt to be triggered, the corresponding enable bit in TWAI_INT_ENA_REG must be set.

The TWAI controller provides the following interrupts:

- Receive Interrupt
- Transmit Interrupt
- Error Warning Interrupt
- Data Overrun Interrupt
- Error Passive Interrupt
- Arbitration Lost Interrupt
- Bus Error Interrupt
- Bus Idle Status Interrupt

The TWAI controller's interrupt signal to the interrupt matrix will be asserted whenever one or more interrupt bits are set in the TWAI_INT_ST_REG, and deasserted when all bits in TWAI_INT_ST_REG are cleared. The majority of interrupt bits in TWAI_INT_ST_REG are automatically cleared when the register is read, except for the Receive Interrupt which can only be cleared when all the messages are released by setting the TWAI_RELEASE_BUF bit.

34.4.3.1 Receive Interrupt (RXI)

The Receive Interrupt (RXI) is asserted whenever the TWAI controller has received messages that are pending to be read from the Receive Buffer (i.e., when TWAI_RX_MESSAGE_CNT_REG > 0). Pending received messages includes valid messages in the Receive FIFO and also overrun messages. The RXI will not be deasserted until all pending received messages are cleared using the TWAI_RELEASE_BUF command bit.

34.4.3.2 Transmit Interrupt (TXI)

The Transmit Interrupt (TXI) is triggered whenever Transmit Buffer becomes free, indicating another message can be loaded into the Transmit Buffer to be transmitted. The Transmit Buffer becomes free under the following scenarios:

- A message transmission has been completed successfully, i.e., acknowledged without any errors. Any failed messages will automatically be resent.
- A single shot transmission has been completed (successfully or unsuccessfully, indicated by the TWAI_TX_COMPLETE bit).
- A message transmission was aborted using the TWAI_ABORT_TX command bit.

34.4.3.3 Error Warning Interrupt (EWI)

The Error Warning Interrupt (EWI) is triggered whenever there is a change to the TWAI_ERR_ST and TWAI_BUS_OFF_ST bits of TWAI_STATUS_REG (i.e., transition from 0 to 1 or vice versa). Thus, an EWI could indicate one of the following events, depending on the values of TWAI_ERR_ST and TWAI_BUS_OFF_ST at the moment when the EWI is triggered.

- If TWAI_ERR_ST = 0 and TWAI_BUS_OFF_ST = 0:
  - If the TWAI controller was in the Error Active state, it indicates both the TEC and REC have returned below the threshold value set by TWAI_ERR_WARNING_LIMIT_REG.
  - If the TWAI controller was previously in the Bus Off Recovery state, it indicates that Bus Recovery has completed successfully.
- If TWAI_ERR_ST = 1 and TWAI_BUS_OFF_ST = 0: The TEC or REC error counters have exceeded the threshold value set by TWAI_ERR_WARNING_LIMIT_REG.
- If TWAI_ERR_ST = 1 and TWAI_BUS_OFF_ST = 1: The TWAI controller has entered the BUS_OFF state (due to the TEC >= 256).
- If TWAI_ERR_ST = 0 and TWAI_BUS_OFF_ST = 1: The TWAI controller's TEC has dropped below the threshold value set by TWAI_ERR_WARNING_LIMIT_REG during BUS_OFF recovery.

34.4.3.4 Data Overrun Interrupt (DOI)

The Data Overrun Interrupt (DOI) is triggered whenever the Receive FIFO has overrun. The DOI indicates that the Receive FIFO is full and should be cleared immediately to prevent any further overrun messages.

The DOI is only triggered by the first message that causes the Receive FIFO to overrun (i.e., the transition from the Receive FIFO not being full to the Receive FIFO overrunning). Any subsequent overrun messages will not

trigger the DOI again. The DOI could be triggered again when all received messages (valid or overrun) have been cleared.

34.4.3.5 Error Passive Interrupt (EPI)

The Error Passive Interrupt (EPI) is triggered whenever the TWAI controller switches from Error Active to Error Passive, or vice versa.

34.4.3.6 Arbitration Lost Interrupt (ALI)

The Arbitration Lost Interrupt (ALI) is triggered whenever the TWAI controller is attempting to transmit a message and loses arbitration. The bit position where the TWAI controller lost arbitration is automatically recorded in the Arbitration Lost Capture register (TWAI_ARB_LOST_CAP_REG). When the ALI occurs again, the Arbitration Lost Capture register will no longer record a new bit location until it is cleared (via CPU reading this register).

34.4.3.7 Bus Error Interrupt (BEI)

The Bus Error Interrupt (BEI) is triggered whenever the TWAI controller detects an error on the TWAI bus. When a bus error occurs, the Bus Error type and its bit position are automatically recorded in the Error Code Capture register (TWAI_ERR_CODE_CAP_REG). When the BEI occurs again, the Error Code Capture register will no longer record new error information until it is cleared (via a read from the CPU).

34.4.3.8 Bus Idle Status Interrupt (BISI)

The Bus Idle Status Interrupt (BISI) is triggered when the number of clock cycles of the TWAI controller in the idle status exceeds the pre-configured value in the TWAI_IDLE_INTR_CNT_REG register. Users can configure this interrupt to get the TWAI controller idle status and further decide whether to turn off the external TWAI receiver to reduce the overall power consumption (see Section 34.4.10).

34.4.4 Transmit and Receive Buffers

34.4.4.1 Overview of Buffers

Table 34.4-3. Buffer Layout for Standard Frame Format and Extended Frame Format

Standard Frame Format (SFF)		Extended Frame Format (EFF)	
Offset Address	Content	Offset Address	Content
0x40	TX/RX frame information	0x40	TX/RX frame information
0x44	TX/RX identifier 1	0x44	TX/RX identifier 1
0x48	TX/RX identifier 2	0x48	TX/RX identifier 2
0x4c	TX/RX data byte 1	0x4c	TX/RX identifier 3
0x50	TX/RX data byte 2	0x50	TX/RX identifier 4
0x54	TX/RX data byte 3	0x54	TX/RX data byte 1
0x58	TX/RX data byte 4	0x58	TX/RX data byte 2
0x5c	TX/RX data byte 5	0x5c	TX/RX data byte 3
0x60	TX/RX data byte 6	0x60	TX/RX data byte 4

Cont'd on next page

Table 34.4-3 – cont'd from previous page

Standard Frame Format (SFF)		Extended Frame Format (EFF)	
Offset Address	Content	Offset Address	Content
0x64	TX/RX data byte 7	0x64	TX/RX data byte 5
0x68	TX/RX data byte 8	0x68	TX/RX data byte 6
0x6c	reserved	0x6c	TX/RX data byte 7
0x70	reserved	0x70	TX/RX data byte 8

Table 34.4-3 illustrates the layout of the Transmit Buffer and Receive Buffer registers. Both the Transmit and Receive Buffer registers share the same address space and are only accessible when the TWAI controller is in Operation mode. The CPU accesses Transmit Buffer registers for write operations, and Receive Buffer registers for read operations. Both buffers share the exact same register layout and fields to store a message (received or to be transmitted). The Transmit Buffer registers are used to configure a TWAI message to be transmitted. The CPU would write to the Transmit Buffer registers specifying the message's frame type, frame format, frame ID, and frame data (payload). Once the Transmit Buffer is configured, the CPU would then initiate the transmission by setting the TWAI_TX_REQ bit in TWAI_CMD_REG.

- For a self-reception request, set the TWAI_SELF_RX_REQ bit instead.
- For a single-shot transmission, set both the TWAI_TX_REQ and the TWAI_ABORT_TX simultaneously.

The Receive Buffer registers map the first message in the Receive FIFO. The CPU would read the Receive Buffer registers to obtain the first message's frame type, frame format, frame ID, and frame data (payload). Once the message has been read from the Receive Buffer registers, the CPU can set the TWAI_RELEASE_BUF bit in TWAI_CMD_REG to clear the Receive Buffer registers. If there are still messages in the Receive FIFO, the Receive Buffer registers will map the first of the remaining messages again.

34.4.4.2 Frame Information

The frame information is one byte long and specifies a message's frame type, frame format, and length of data. The frame information fields are shown in Table 34.4-4.

Table 34.4-4. TX/RX Frame Information (SFF/EFF);TWAI Address 0x40

Bit 31-8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Reserved	FF[1]	RTR[2]	X[3]	X[3]	DLC.3[4]	DLC.2[4]	DLC.1[4]	DLC.0[4]

Notes:

1. FF: The Frame Format (FF) bit specifies whether the message is Extended Frame Format (EFF) or Standard Frame Format (SFF). The message is EFF when the FF bit is 1, and SFF when the FF bit is 0.
2. RTR: The Remote Transmission Request (RTR) bit specifies whether the message is a data frame or a remote frame. The message is a remote frame when the RTR bit is 1, and a data frame when the RTR bit is 0.
3. X: Don't care, can be any value.
4. DLC: The Data Length Code (DLC) field specifies the number of data bytes for a data frame, or the number of data bytes to request in a remote frame. TWAI data frames are limited to a maximum payload of 8 data bytes, and thus the DLC should range from 0 to 8.

34.4.4.3 Frame Identifier

The Frame Identifier fields occupy two-byte (11-bit) long if the message is SFF, and four-byte (29-bit) long if the message is EFF.

The Frame Identifier fields for an SFF (11-bit) message are shown in Table 34.4-5 ~ 34.4-6.

Table 34.4-5. TX/RX Identifier 1 (SFF); TWAI Address 0x44

Bit 31-8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Reserved	ID.10	ID.9	ID.8	ID.7	ID.6	ID.5	ID.4	ID.3

Table 34.4-6. TX/RX Identifier 2 (SFF); TWAI Address 0x48

Bit 31-8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Reserved	ID.2	ID.1	ID.0	X[1]	X[2]	X[2]	X[2]	X[2]

Notes:

1. Don't care. Recommended to be compatible with receive buffer (i.e., set to RTR) in case of using the self-reception functionality (or together with self-test functionality).
2. Don't care. Recommended to be compatible with receive buffer (i.e., set to 0) in case of using the self-reception functionality (or together with self-test functionality).

The Frame Identifier fields for an EFF (29-bits) message is shown in Table 34.4-7 ~ 34.4-10.

Table 34.4-7. TX/RX Identifier 1 (EFF); TWAI Address 0x44

Bit 31-8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Reserved	ID.28	ID.27	ID.26	ID.25	ID.24	ID.23	ID.22	ID.21

Table 34.4-8. TX/RX Identifier 2 (EFF); TWAI Address 0x48

Bit 31-8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Reserved	ID.20	ID.19	ID.18	ID.17	ID.16	ID.15	ID.14	ID.13

Table 34.4-9. TX/RX Identifier 3 (EFF); TWAI Address 0x4c

Bit 31-8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Reserved	ID.12	ID.11	ID.10	ID.9	ID.8	ID.7	ID.6	ID.5

Table 34.4-10. TX/RX Identifier 4 (EFF); TWAI Address 0x50

Bit 31-8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Reserved	ID.4	ID.3	ID.2	ID.1	ID.0	X[1]	X[2]	X[2]

Notes:

1. Don't care. Recommended to be compatible with receive buffer (i.e., set to RTR) in case of using the self-reception functionality (or together with self-test functionality).
2. Don't care. Recommended to be compatible with receive buffer (i.e., set to 0) in case of using the self-reception functionality (or together with self-test functionality).

34.4.4.4 Frame Data

The Frame Data field contains the payloads of transmitted or received data frame, and can range from 0 to 8 bytes. The number of valid bytes should be equal to the DLC. However, if the DLC is larger than eight bytes, the number of valid bytes would still be limited to eight. Remote frames do not have data payloads, so their Frame Data fields will be unused.

For example, when transmitting a data frame with five bytes, the CPU should write five to the DLC field, and then write data to the corresponding register of the first to the fifth data field. Likewise, when the CPU receives a data frame with a DLC of five data bytes, only the first to the fifth data byte will contain valid payload data for the CPU to read.

34.4.5 Receive FIFO and Data Overruns

The Receive FIFO is a 64-byte internal buffer used to store received messages in First In First Out order. A single received message can occupy between 3 to 13 bytes of space in the Receive FIFO, and their endianness is identical to the register layout of the Receive Buffer registers. The Receive Buffer registers are mapped to the bytes of the first message in the Receive FIFO.

When the TWAI controller receives a message, it will increment the value of TWAI_RX_MESSAGE_COUNTER by 1 with a maximum of 64. If there is adequate space in the Receive FIFO, the message contents will be written

into the Receive FIFO. Once a message has been read from the Receive Buffer, the TWAI_RELEASE_BUF bit should be set. This will decrement TWAI_RX_MESSAGE_COUNTER by 1 and free the space occupied by the first message in the Receive FIFO. The Receive Buffer will then map to the next message in the Receive FIFO.

A data overrun occurs when the TWAI controller receives a message, but the Receive FIFO lacks adequate free space to store the received message in its entirety (either due to the message contents being larger than the free space in the Receive FIFO, or the Receive FIFO being completely full).

When a data overrun occurs:

- The free space left in the Receive FIFO is filled with the partial contents of the overrun message. If the Receive FIFO is already full, then none of the overrun message's contents will be stored.
- When data in the Receive FIFO overruns for the first time, a Data Overrun Interrupt will be triggered.
- Each overrun message will still increment the TWAI_RX_MESSAGE_COUNTER up to a maximum of 64.
- The Receive FIFO will internally mark overrun messages as invalid. The TWAI_MISS_ST bit can be used to determine whether the message currently mapped to by the Receive Buffer is valid or overrun.

To clear an overrun Receive FIFO, the TWAI_RELEASE_BUF must be called repeatedly until TWAI_RX_MESSAGE_COUNTER is 0. This requires users to read all valid messages in the Receive FIFO and clear all overrun messages.

34.4.6 Acceptance Filter

The Acceptance Filter allows the TWAI controller to filter out received messages based on their ID (and optionally their first data byte and frame type). Only accepted messages are passed on to the Receive FIFO. The use of Acceptance Filters allows a more lightweight operation of the TWAI controller (e.g., less use of Receive FIFO, fewer Receive Interrupts) since the TWAI Controller only needs to handle a subset of messages.

The Acceptance Filter configuration registers can only be accessed whilst the TWAI controller is in Reset mode, since they share the same address spaces with the Transmit Buffer and Receive Buffer registers.

The configuration registers consist of a 32-bit Acceptance Code Value and a 32-bit Acceptance Mask Value. The Acceptance Code value specifies a bit pattern which each filtered bit of the message must match in order for the message to be accepted. The Acceptance Mask Value is able to mask out certain bits of the Code value (i.e., set as "Don't Care" bits). Each filtered bit of the message must either match the acceptance code or be masked in order for the message to be accepted, as demonstrated in Figure 34.4-1.

The TWAI controller Acceptance Filter allows the 32-bit Acceptance Code and Mask Values to either define a single filter (i.e., Single Filter mode), or two filters (i.e., Dual Filter mode). How the Acceptance Filter interprets the 32-bit code and mask values is dependent on the filter mode and the format of received messages (i.e., SFF or EFF).

34.4.6.1 Single Filter Mode

Single Filter mode is enabled by setting the TWAI_RX_FILTER_MODE bit to 1. This will cause the 32-bit code and mask values to define a single filter. The single filter can filter the following bits of data or remote frames:

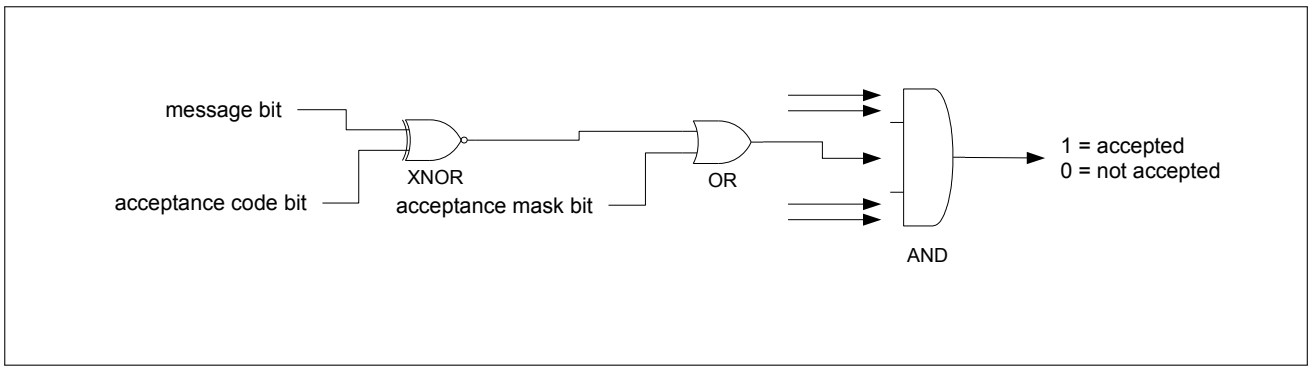


Figure 34.4-1. Acceptance Filter

- SFF
  - The entire 11-bit ID
  - RTR bit
  - Data byte 1 and Data byte 2
- EFF
  - The entire 29-bit ID
  - RTR bit

The following Figure 34.4-2 illustrates how the 32-bit code and mask values will be interpreted under Single Filter mode.

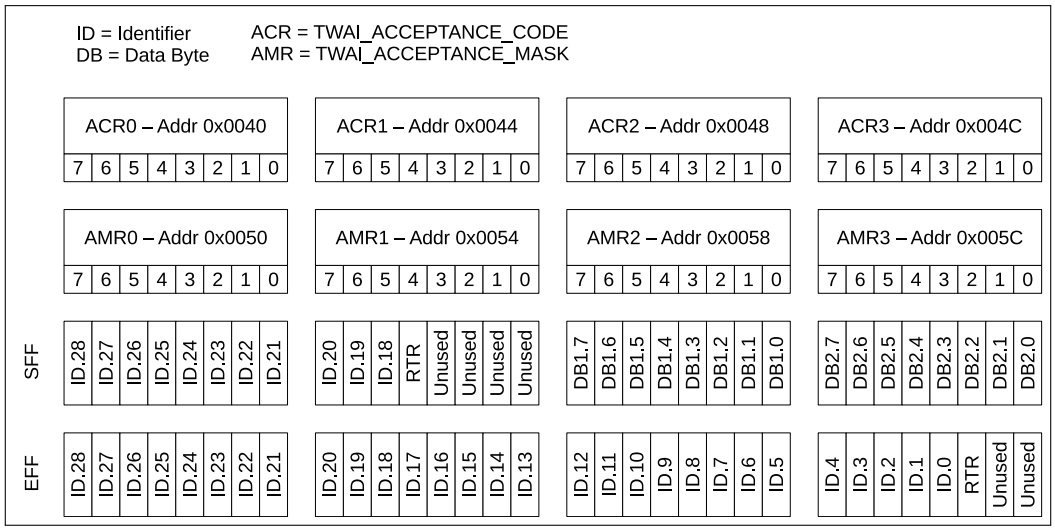


Figure 34.4-2. Single Filter Mode

34.4.6.2 Dual Filter Mode

Dual Filter mode is enabled by clearing the TWAI_RX_FILTER_MODE bit to 0. This will cause the 32-bit code and mask values to define two separate filters referred to as filter 1 or filter 2. Under Dual Filter mode, a message will be accepted if it is accepted by one of the two filters.

The two filters can filter the following bits of data or remote frames:

- SFF
  - The entire 11-bit ID
  - RTR bit
  - Data byte 1 (for filter 1 only)
- EFF
  - The first 16 bits of the 29-bit ID

The following Figure 34.4-3 illustrates how the 32-bit code and mask values will be interpreted in Dual Filter mode.

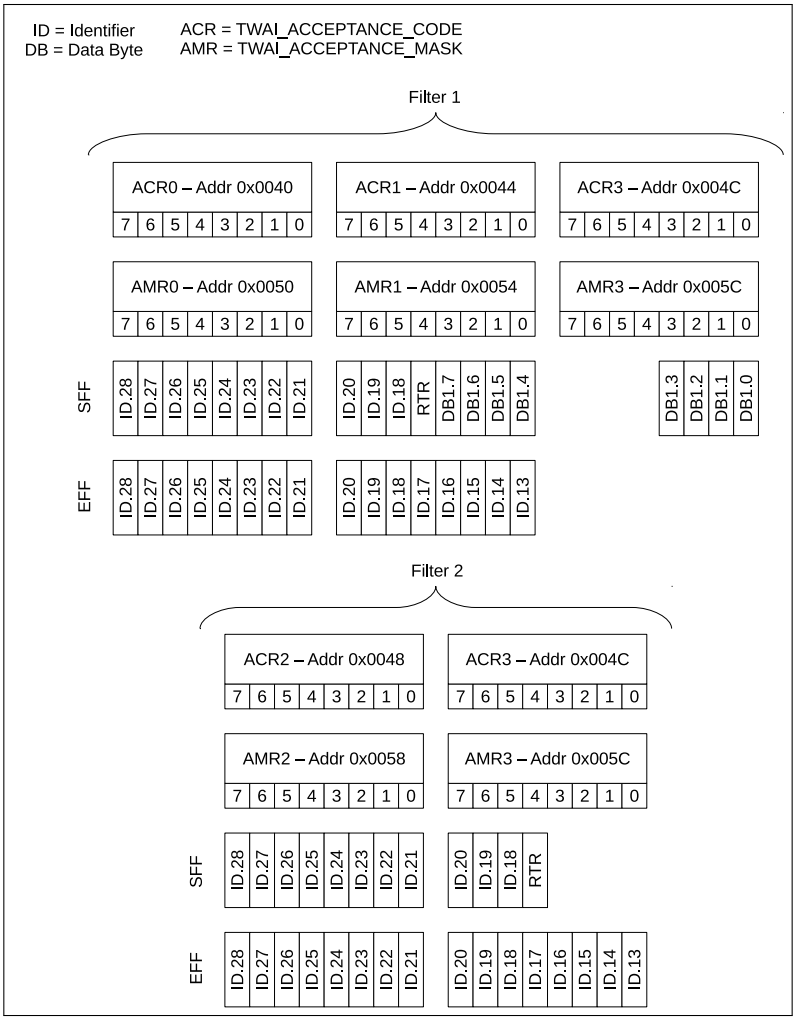


Figure 34.4-3. Dual Filter Mode

34.4.7 Error Management

The TWAI protocol requires that each TWAI node maintains the Transmit Error Counter (TEC) and Receive Error Counter (REC). The value of both error counters determines the current error state of the TWAI controller (i.e., Error Active, Error Passive, Bus-Off). The TWAI controller stores the TEC and REC values in TWAI_TX_ERR_CNT_REG and TWAI_RX_ERR_CNT_REG respectively, and they can be read by the CPU anytime. In addition to the error states, the TWAI controller also offers an Error Warning Limit (EWL) feature that can warn users of the occurrence of severe bus errors before the TWAI controller enters the Error Passive state.

The current error state of the TWAI controller is indicated via a combination of the following values and status bits: TEC, REC, TWAI_ERR_ST, and TWAI_BUS_OFF_ST. Certain changes to these values and bits will also trigger interrupts, so that users are notified of error state transitions (see section 34.4.3). The following figure 34.4-4 shows the relation between the error states, values and bits, and error state related interrupts.

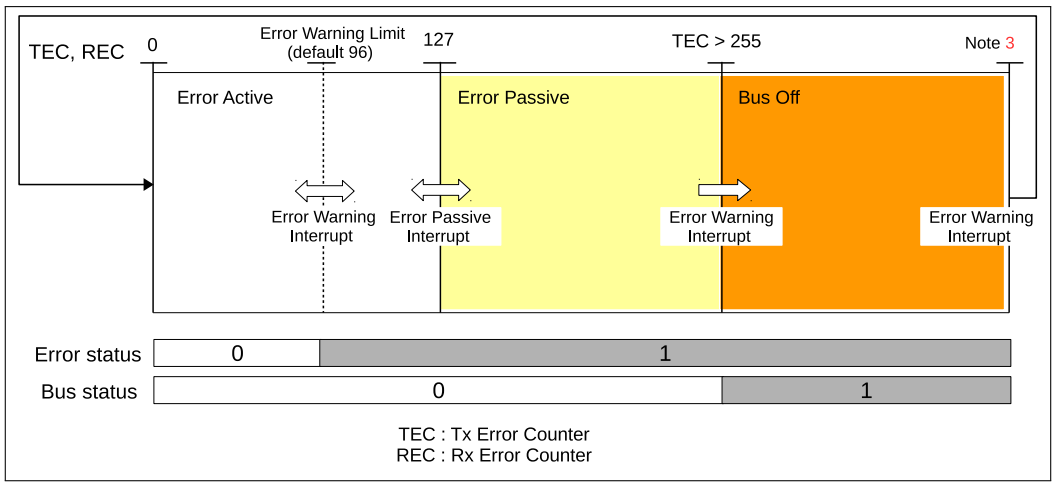


Figure 34.4-4. Error State Transition

34.4.7.1 Error Warning Limit

The Error Warning Limit (EWL) is a configurable threshold value for the TEC and REC, which will trigger an interrupt when exceeded. The EWL is intended to serve as a warning about severe TWAI bus errors, and is triggered before the TWAI controller enters the Error Passive state. The EWL is configured in TWAI_ERR_WARNING_LIMIT_REG and can only be configured whilst the TWAI controller is in Reset mode. The TWAI_ERR_WARNING_LIMIT_REG has a default value of 96.

When the values of TEC and/or REC are larger than or equal to the EWL value, the TWAI_ERR_ST bit is immediately set to 1. Likewise, when the values of both the TEC and REC are smaller than the EWL value, the TWAI_ERR_ST bit is immediately reset to 0. The Error Warning Interrupt is triggered whenever the value of the TWAI_ERR_ST bit (or the TWAI_BUS_OFF_ST) changes.

34.4.7.2 Error Passive

The TWAI controller is in the Error Passive state when the TEC or REC value exceeds 127. Likewise, when both the TEC and REC are less than or equal to 127, the TWAI controller enters the Error Active state. The Error Passive Interrupt is triggered whenever the TWAI controller transitions from the Error Active state to the Error Passive state or vice versa.

34.4.7.3 Bus-Off and Bus-Off Recovery

The TWAI controller enters the Bus-Off state when the TEC value exceeds 255. On entering the Bus-Off state, the TWAI controller will automatically do the following:

- Set REC to 0
- Set TEC to 127
- Set the TWAI_BUS_OFF_ST bit to 1

- Enter Reset mode

The Error Warning Interrupt is triggered whenever the value of the TWAI_BUS_OFF_ST bit (or the TWAI_ERR_ST bit) changes.

To return to the Error Active state, the TWAI controller must undergo Bus-Off Recovery. Bus-Off Recovery requires the TWAI controller to observe 128 occurrences of 11 consecutive recessive bits on the bus. To initiate Bus-Off Recovery (after entering the Bus-Off state), the TWAI controller should enter Operation mode by setting the TWAI_RESET_MODE bit to 0. The TEC tracks the progress of Bus-Off Recovery by decrementing the TEC each time when the TWAI controller observes 11 consecutive recessive bits. When Bus-Off Recovery has completed (i.e., TEC has decremented from 127 to 0), the TWAI_BUS_OFF_ST bit will automatically be reset to 0, thus triggering the Error Warning Interrupt.

34.4.8 Error Code Capture

The Error Code Capture (ECC) feature allows the TWAI controller to record the error type and bit position of a TWAI bus error in the form of an error code. Upon detecting a TWAI bus error, the Bus Error Interrupt is triggered and the error code is recorded in TWAI_ERR_CODE_CAP_REG. Subsequent bus errors will trigger the Bus Error Interrupt, but their error codes will not be recorded until the current error code is read from TWAI_ERR_CODE_CAP_REG.

The following Table 34.4-11 shows the fields of the TWAI_ERR_CODE_CAP_REG:

Table 34.4-11. Bit Information of TWAI_ERR_CODE_CAP_REG (0x30)

Bit 31-8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Reserved	ERRC.1[1]	ERRC.0[1]	DIR[2]	SEG.4[3]	SEG.3[3]	SEG.2[3]	SEG.1[3]	SEG.0[3]

Notes:

- ERRC: The Error Code (ERRC) indicates the type of bus error; 00 for bit error, 01 for format error, 10 for stuff error, and 11 for other types of error.
- DIR: The Direction (DIR) indicates whether the TWAI controller was transmitting or receiving when the bus error occurred; 0 for the transmitter, 1 for the receiver.
- SEG: The Error Segment (SEG) indicates the segment of the TWAI message at which the bus error occurred.

The following Table 34.4-12 shows how to interpret the SEG.0 to SEG.4 bits.

Table 34.4-12. Bit Information of Bits SEG.4 - SEG.0

Bit SEG.4	Bit SEG.3	Bit SEG.2	Bit SEG.1	Bit SEG.0	Description
0	0	0	1	1	start of frame
0	0	0	1	0	ID.28 ~ ID.21
0	0	1	1	0	ID.20 ~ ID.18
0	0	1	0	0	bit SRTR
0	0	1	0	1	bit IDE
0	0	1	1	1	ID.17 ~ ID.13
0	1	1	1	1	ID.12 ~ ID.5

Cont'd on next page

Table 34.4-12 – cont'd from previous page

Bit SEG.4	Bit SEG.3	Bit SEG.2	Bit SEG.1	Bit SEG.0	Description
0	1	1	1	0	ID.4 ~ ID.0
0	1	1	0	0	bit RTR
0	1	1	0	1	reserved bit 1
0	1	0	0	1	reserved bit 0
0	1	0	1	1	data length code
0	1	0	1	0	data field
0	1	0	0	0	CRC sequence
1	1	0	0	0	CRC delimiter
1	1	0	0	1	ACK slot
1	1	0	1	1	ACK delimiter
1	1	0	1	0	end of frame
1	0	0	1	0	intermission
1	0	0	0	1	active error flag
1	0	1	1	0	passive error flag
1	0	0	1	1	tolerate dominant bits
1	0	1	1	1	error delimiter
1	1	1	0	0	overload flag

Notes:

- Bit SRTR: under Standard Frame Format.
- Bit IDE: Identifier Extension Bit, 0 for Standard Frame Format.

34.4.9 Arbitration Lost Capture

The Arbitration Lost Capture (ALC) feature allows the TWAI controller to record the bit position where it loses arbitration. When the TWAI controller loses arbitration, the bit position is recorded in TWAI_ARB_LOST_CAP_REG and the Arbitration Lost Interrupt is triggered.

Subsequent losses in arbitration will trigger the Arbitration Lost Interrupt, but will not be recorded in TWAI_ARB_LOST_CAP_REG until the current Arbitration Lost Capture is read from the TWAI_ERR_CODE_CAP_REG.

Table 34.4-13 illustrates bits and fields of TWAI_ERR_CODE_CAP_REG whilst Figure 34.4-5 illustrates the bit positions of a TWAI message.

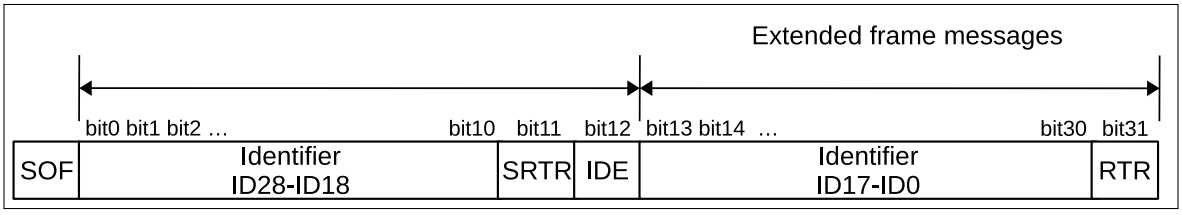


Figure 34.4-5. Positions of Arbitration Lost Bits

Table 34.4-13. Bit Information of TWAI_ARB_LOST_CAP_REG (0x2c)

Bit 31-5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Reserved	BITNO.4[1]	BITNO.3[1]	BITNO.2[1]	BITNO.1[1]	BITNO.0[1]

Notes:

- BITNO: Bit Number (BITNO) indicates the nth bit of a TWAI message where arbitration was lost.

34.4.10 Transceiver Auto-Standby

It is common for TWAI transceivers to support a Standby mode to lower power consumption. TWAI transceivers will usually generate a standby signal that is asserted by the connected TWAI controller, thus allowing the controller to place the transceiver into standby when appropriate (e.g., when the bus will be idle for an extended period of time). Transceivers will exit Standby mode if the controller de-asserts the standby signal, or if the transceiver detects bus activity (also known as a wake-up feature).

ESP32-H2's TWAI controller supports both hardware control (i.e., automatic) and software control (i.e., manual) of the standby signal to control the switching of TWAI transceivers connected to the chip. When hardware controlled, the TWAI controller will automatically assert the standby signal when the bus remains idle for longer than a configurable amount of time. When software controlled, the standby signal can be manually asserted/de-asserted directly by the software.

- Hardware output:
  1. Set the TWAI_HW_STANDBY_EN field in the TWAI_HW_CFG_REG register to enable standby function for hardware.
  2. Configure the TWAI_HW_STANDB_CNT_REG register. This register indicates the time required before hardware triggers the standby signal after entering idle status, in which the value indicates the number of cycles of the TWAI controller operating clock (32 MHz by default).
- Software output:
  1. Set the TWAI_SW_STANDBY_EN field in the TWAI_SW_STANDBY_CFG_REG register to generate standby signals in the TWAI controller.

The standby signal generated using either of the above methods will be pulled down (cleared) when either of the following conditions is met:

1. The standby signal will be automatically cleared when the TWAI controller exits the idle status.
2. Users can also pull down the standby signal by setting the TWAI_SW_STANDBY_CLR field in the TWAI_SW_STANDBY_CFG_REG register.

34.5 Register Summary

'|' here means a separate line to distinguish between TWAI working modes discussed in Section 34.4.1 *Modes*. The left describes the access in Operation mode. The right belongs to Reset mode and is marked in red. The addresses in this section are relative to Two-wire Automotive Interface base address provided in Table 4.3-2 in Chapter 4 *System and Memory*.

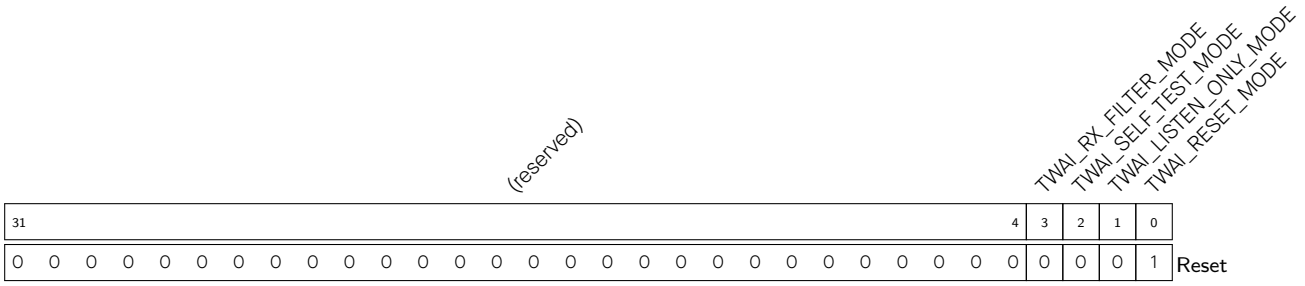
The abbreviations given in Column **Access** are explained in Section *Access Types for Registers*.

Name	Description	Address	Access
Configuration Registers			
TWAI_MODE_REG	Mode Register	0x0000	R/W
TWAI_BUS_TIMING_0_REG	Bus Timing Register 0	0x0018	RO | R/W
TWAI_BUS_TIMING_1_REG	Bus Timing Register 1	0x001C	RO | R/W
TWAI_ERR_WARNING_LIMIT_REG	Error Warning Limit Register	0x0034	RO | R/W
TWAI_DATA_0_REG	Data Register 0	0x0040	WO | R/W
TWAI_DATA_1_REG	Data Register 1	0x0044	WO | R/W
TWAI_DATA_2_REG	Data Register 2	0x0048	WO | R/W
TWAI_DATA_3_REG	Data Register 3	0x004C	WO | R/W
TWAI_DATA_4_REG	Data Register 4	0x0050	WO | R/W
TWAI_DATA_5_REG	Data Register 5	0x0054	WO | R/W
TWAI_DATA_6_REG	Data Register 6	0x0058	WO | R/W
TWAI_DATA_7_REG	Data Register 7	0x005C	WO | R/W
TWAI_DATA_8_REG	Data Register 8	0x0060	WO | RO
TWAI_DATA_9_REG	Data Register 9	0x0064	WO | RO
TWAI_DATA_10_REG	Data Register 10	0x0068	WO | RO
TWAI_DATA_11_REG	Data Register 11	0x006C	WO | RO
TWAI_DATA_12_REG	Data Register 12	0x0070	WO | RO
TWAI_CLOCK_DIVIDER_REG	Clock Divider Register	0x007C	varies
TWAI_SW_STANDBY_CFG_REG	Software Standby Register	0x0080	R/W | R/W
TWAI_HW_CFG_REG	Software Standby Register	0x0084	R/W | R/W
TWAI_HW_STANDBY_CNT_REG	Standby Time Length Register	0x0088	R/W | R/W
TWAI_IDLE_INTR_CNT_REG	Idle Status Time Length Register	0x008C	R/W | R/W
Control Registers			
TWAI_CMD_REG	Command Register	0x0004	WO
Status Register			
TWAI_STATUS_REG	Status Register	0x0008	RO
TWAI_ARB_LOST_CAP_REG	Arbitration Lost Capture Register	0x002C	RO
TWAI_ERR_CODE_CAP_REG	Error Code Capture Register	0x0030	RO
TWAI_RX_ERR_CNT_REG	Receive Error Counter Register	0x0038	RO | R/W
TWAI_TX_ERR_CNT_REG	Transmit Error Counter Register	0x003C	RO | R/W
TWAI_RX_MESSAGE_CNT_REG	Receive Message Counter Register	0x0074	RO
Interrupt Registers			
TWAI_INT_ST_REG	Interrupt Mask Register	0x000C	RO
TWAI_INT_ENA_REG	Interrupt Enable Register	0x0010	R/W

34.6 Registers

'|' here means a separate line. The left describes the access in Operation mode. The right belongs to Reset mode with red color. The addresses in this section are relative to Two-wire Automotive Interface base address (each TWAI 0 and TWAI 1 has an individual base address) provided in Table 4.3-2 in Chapter 4 *System and Memory*.

Register 34.1. TWAI_MODE_REG (0x0000)



TWAI_RESET_MODE Configures the Operation mode of the TWAI Controller.

0: Operation mode

1: Reset mode

(R/W)

TWAI_LISTEN_ONLY_MODE Configures whether to enter the Listen-Only mode.

0: No effect

1: Listen-Only mode. In this mode, the nodes will only receive messages from the bus, without generating the acknowledge signal or updating the RX error counter.

(R/W)

TWAI_SELF_TEST_MODE Configures whether to enter the Self-Test mode.

0: No effect

1: Enter the Self-Test mode. In this mode, the TX nodes can perform a successful transmission without receiving the acknowledge signal. This mode is often used to test a single node with the self-reception request command.

(R/W)

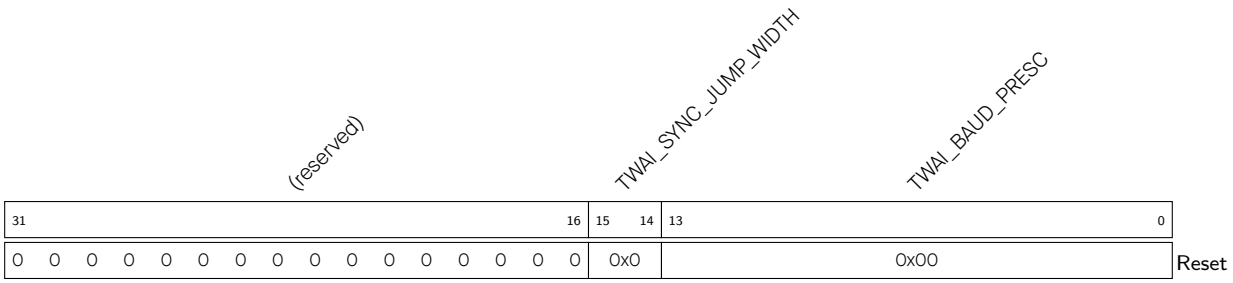
TWAI_RX_FILTER_MODE Configures the filter mode.

0: Dual Filter mode

1: Single Filter mode

(R/W)

Register 34.2. TWAI_BUS_TIMING_0_REG (0x0018)



TWAI_BAUD_PRESC Configures baud rate prescaler value, determining the frequency dividing ratio.

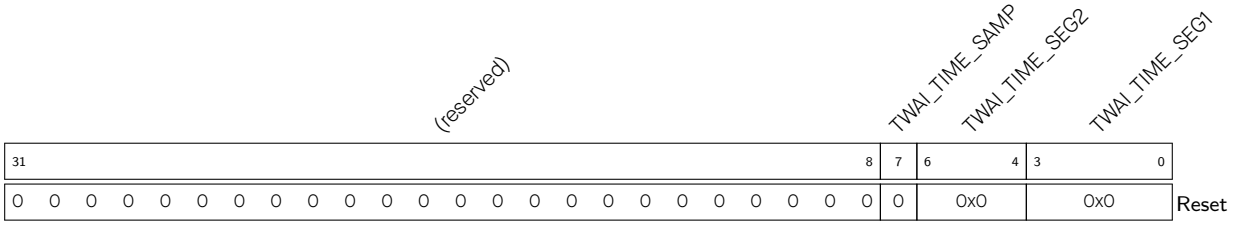
0: Low

1: High

(RO | R/W)

TWAI_SYNC_JUMP_WIDTH Configures Synchronization Jump Width (SJW), ranging from 1 ~ 4 Tq wide. (RO | R/W)

Register 34.3. TWAI_BUS_TIMING_1_REG (0x001C)



TWAI_TIME_SEG1 Configures the width of PBS1. (RO | R/W)

TWAI_TIME_SEG2 Configures the width of PBS2. (RO | R/W)

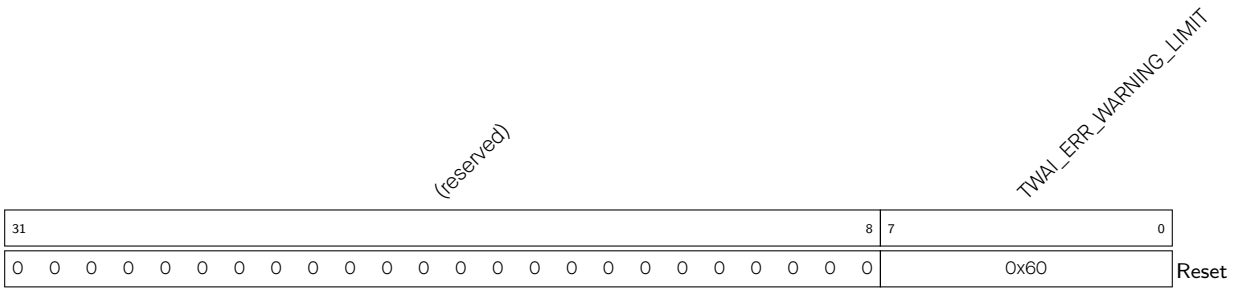
TWAI_TIME_SAMP Configures the number of sample points.

0: The bus is sampled once

1: The bus is sampled three times

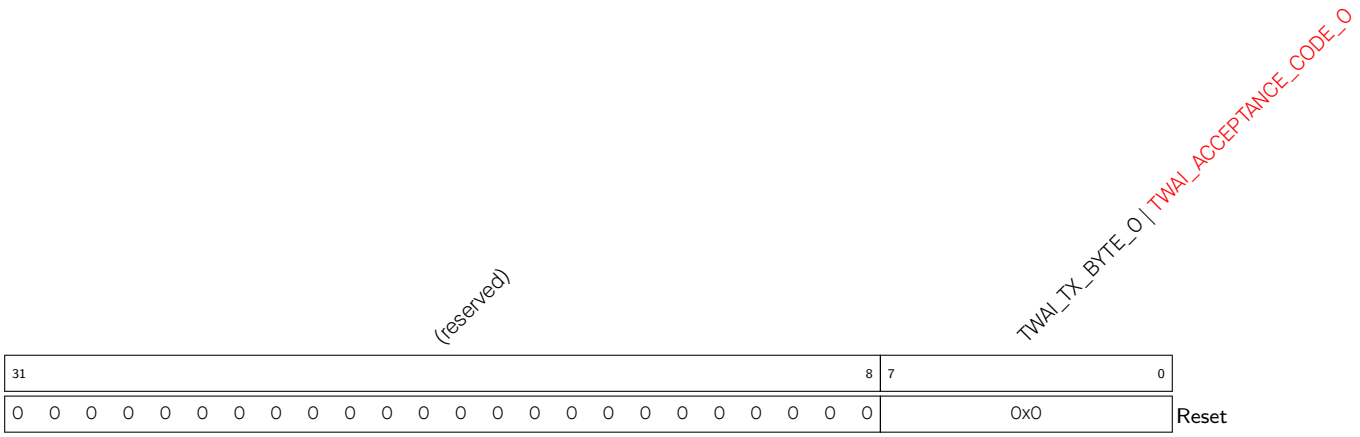
(RO | R/W)

Register 34.4. TWAI_ERR_WARNING_LIMIT_REG (0x0034)



TWAI_ERR_WARNING_LIMIT Configures error warning threshold. In the case when any of an error counter value exceeds the threshold, or all the error counter values are below the threshold, an error warning interrupt will be triggered. Valid only when the enable signal is 1. (RO | R/W)

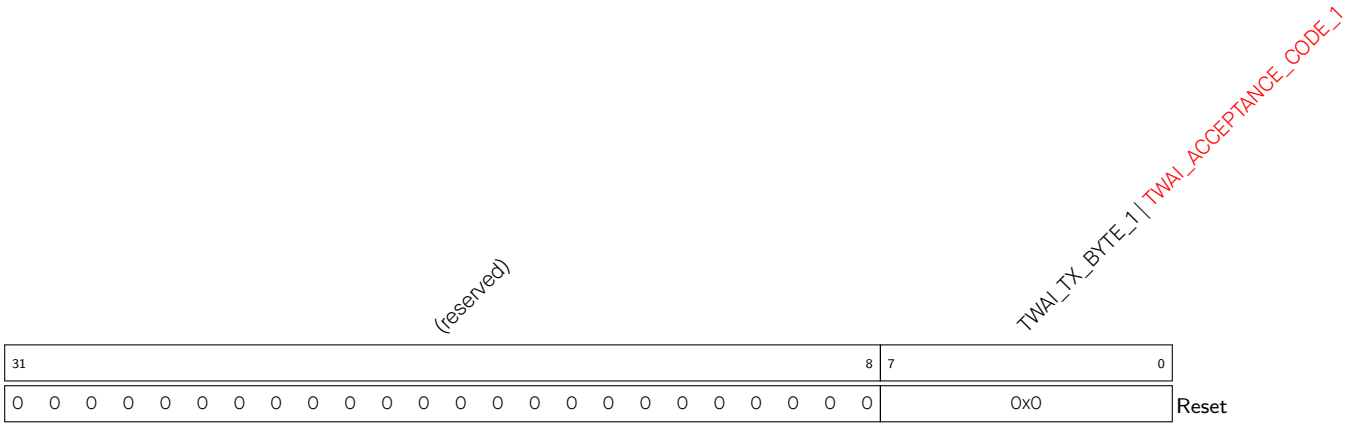
Register 34.5. TWAI_DATA_0_REG (0x0040)



TWAI_TX_BYTE_0 Configures the 0th byte information of the data to be transmitted in Operation mode. (WO)

TWAI_ACCEPTANCE_CODE_0 Configures the 0th byte of the filter code in Reset mode. (R/W)

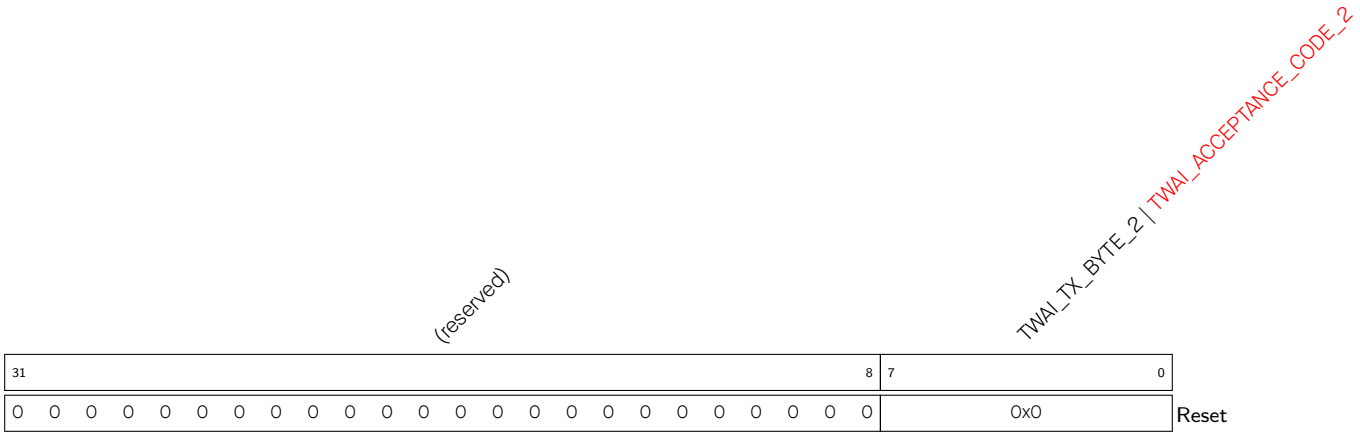
Register 34.6. TWAI_DATA_1_REG (0x0044)



TWAI_TX_BYTE_1 Configures the 1st byte information of the data to be transmitted in Operation mode. (WO)

TWAI_ACCEPTANCE_CODE_1 Configures the 1st byte of the filter code in Reset mode. (R/W)

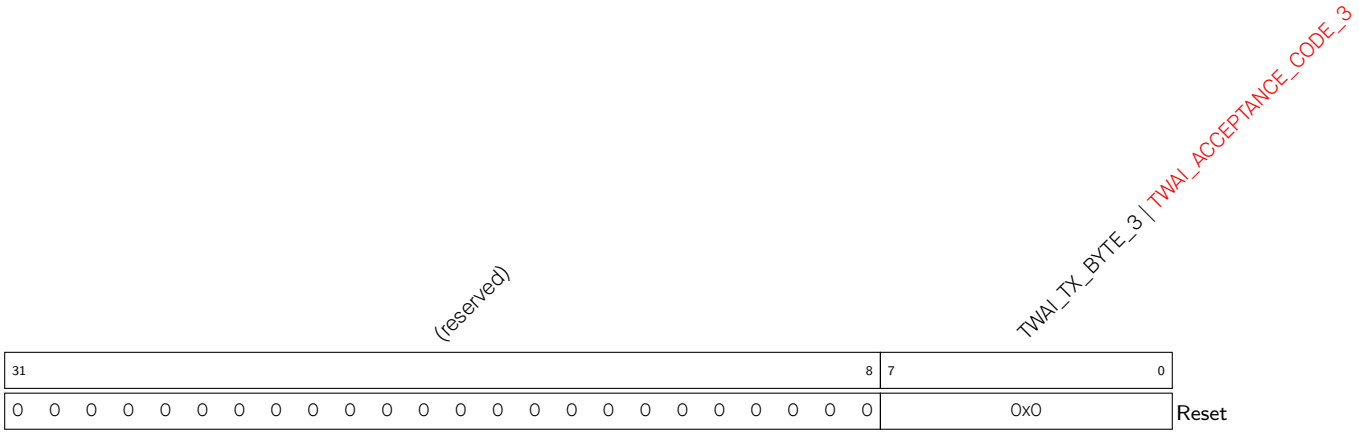
Register 34.7. TWAI_DATA_2_REG (0x0048)



TWAI_TX_BYTE_2 Configures the 2nd byte information of the data to be transmitted in Operation mode. (WO)

TWAI_ACCEPTANCE_CODE_2 Configures the 2nd byte of the filter code in Reset mode. (R/W)

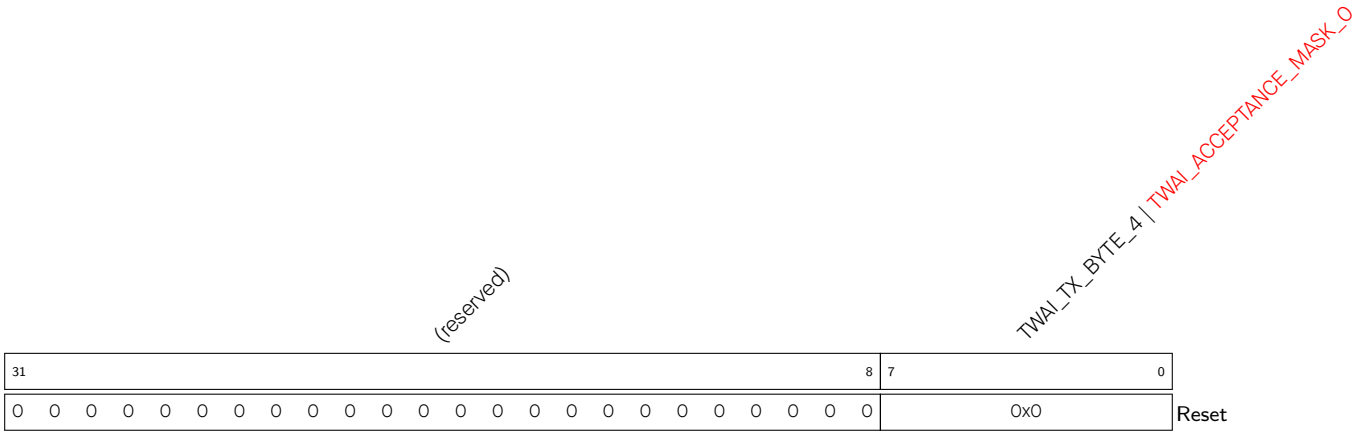
Register 34.8. TWAI_DATA_3_REG (0x004C)



TWAI_TX_BYTE_3 Configures the 3rd byte information of the data to be transmitted in Operation mode. (WO)

TWAI_ACCEPTANCE_CODE_3 Configures the 3rd byte of the filter code in Reset mode. (R/W)

Register 34.9. TWAI_DATA_4_REG (0x0050)



TWAI_TX_BYTE_4 Configures the 4th byte information of the data to be transmitted in Operation mode. (WO)

TWAI_ACCEPTANCE_MASK_0 Configures the 0th byte of the filter code in Reset mode.

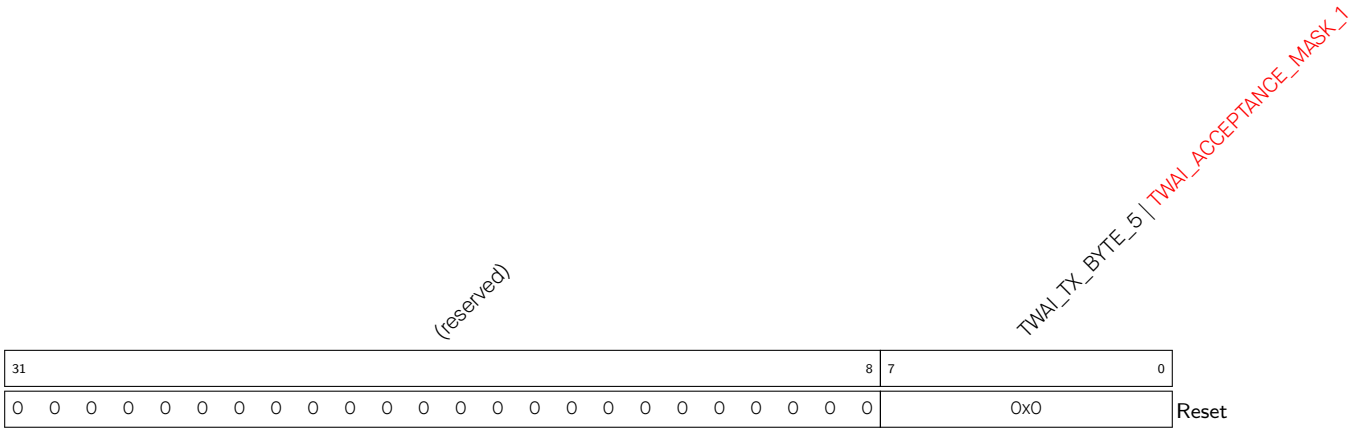
1: nihao

2: world

3: success

(R/W)

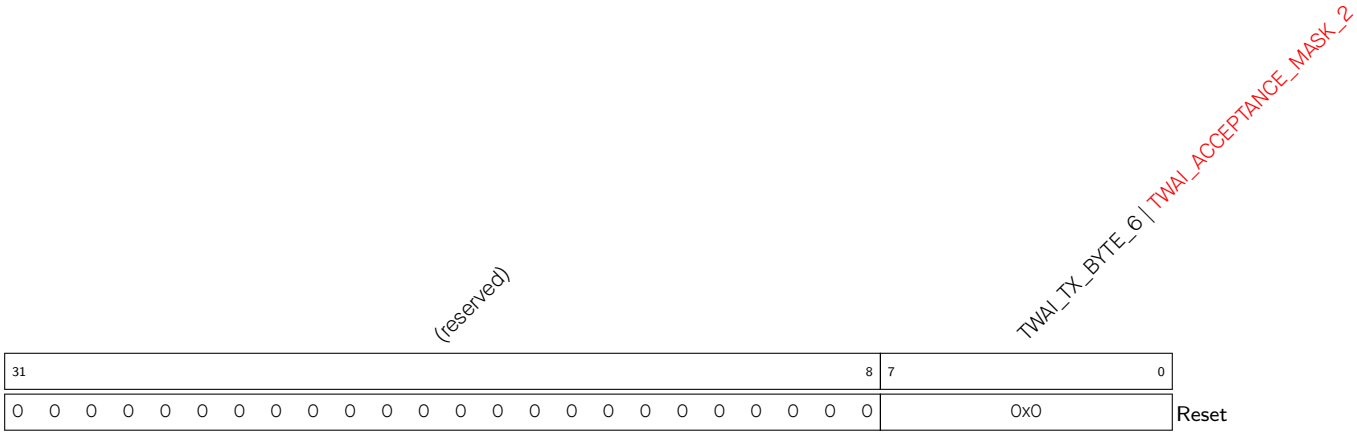
Register 34.10. TWAI_DATA_5_REG (0x0054)



TWAI_TX_BYTE_5 Configures the 5th byte information of the data to be transmitted in Operation mode. (WO)

TWAI_ACCEPTANCE_MASK_1 Configures the 1st byte of the filter code in Reset mode. (R/W)

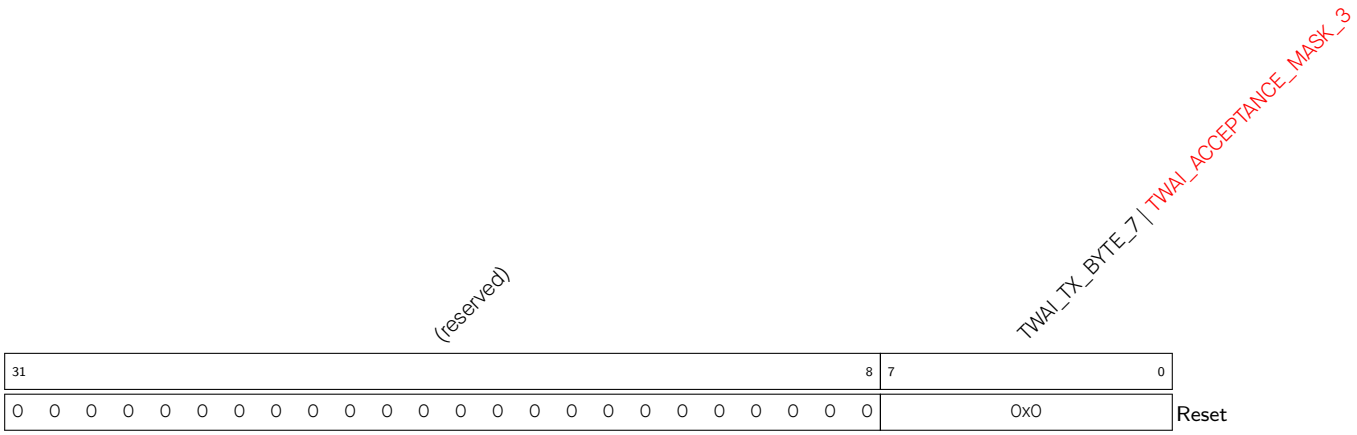
Register 34.11. TWAI_DATA_6_REG (0x0058)



TWAI_TX_BYTE_6 Configures the 6th byte information of the data to be transmitted in Operation mode. (WO)

TWAI_ACCEPTANCE_MASK_2 Configures the 2nd byte of the filter code in Reset mode. (R/W)

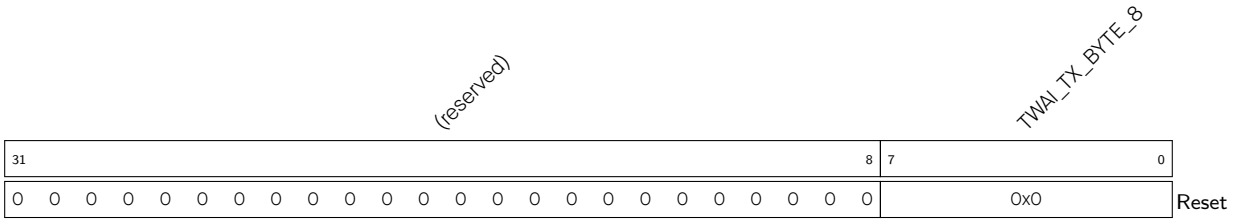
Register 34.12. TWAI_DATA_7_REG (0x005C)



TWAI_TX_BYTE_7 Configures the 7th byte information of the data to be transmitted in Operation mode. (WO)

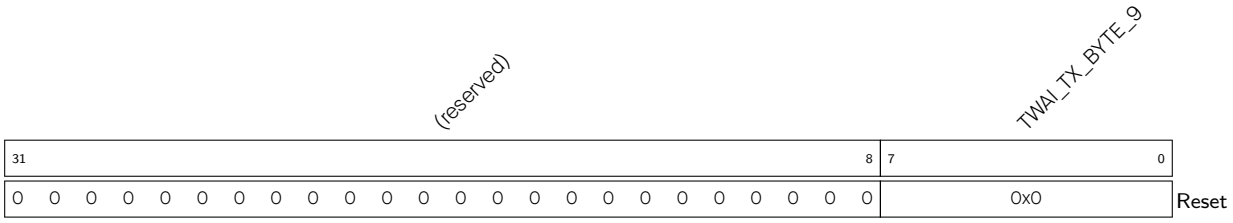
TWAI_ACCEPTANCE_MASK_3 Configures the 3rd byte of the filter code in Reset mode. (R/W)

Register 34.13. TWAI_DATA_8_REG (0x0060)



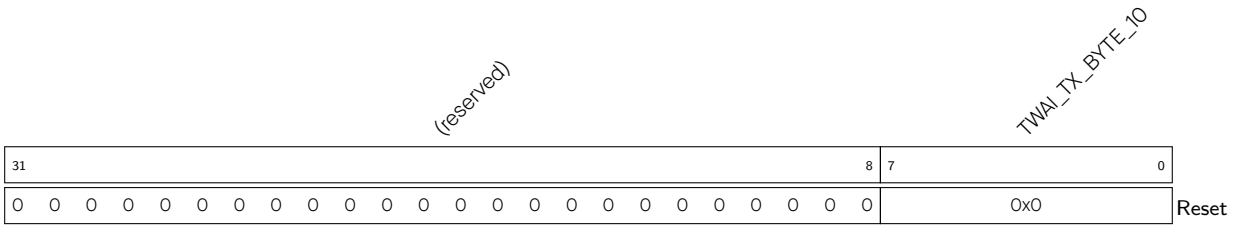
TWAI_TX_BYTE_8 Configures the 8th byte information of the data to be transmitted in Operation mode. (WO)

Register 34.14. TWAI_DATA_9_REG (0x0064)



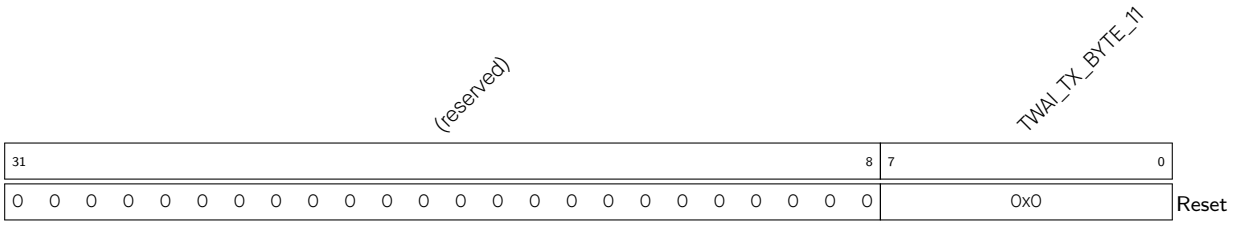
TWAI_TX_BYTE_9 Configures the 9th byte information of the data to be transmitted in Operation mode. (WO)

Register 34.15. TWAI_DATA_10_REG (0x0068)



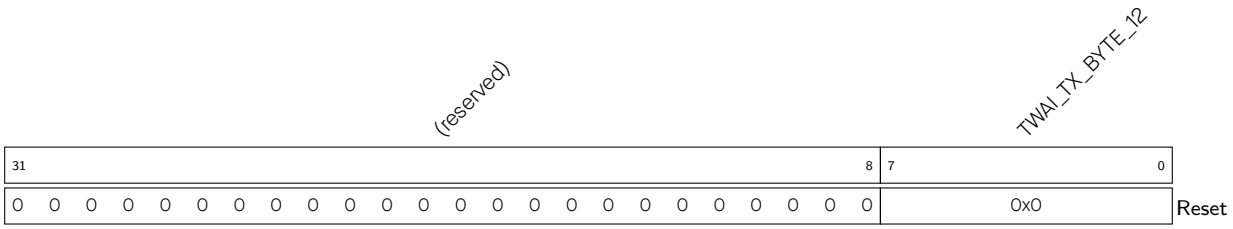
TWAI_TX_BYTE_10 Configures the 10th byte information of the data to be transmitted in Operation mode. (WO)

Register 34.16. TWAI_DATA_11_REG (0x006C)



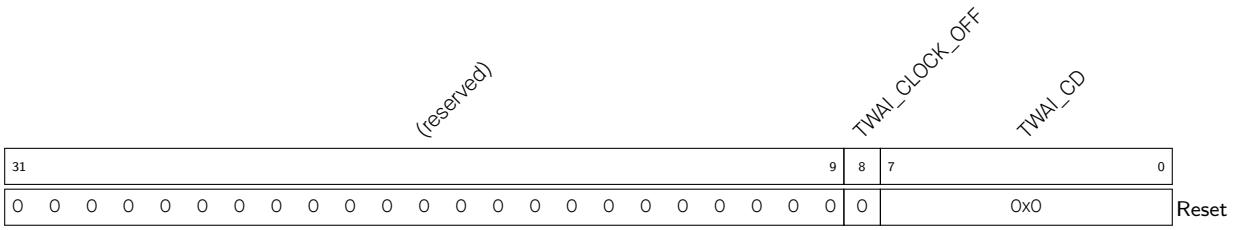
TWAI_TX_BYTE_11 Configures the 11th byte information of the data to be transmitted in Operation mode. (WO)

Register 34.17. TWAI_DATA_12_REG (0x0070)



TWAI_TX_BYTE_12 Configures the 12th byte information of the data to be transmitted in Operation mode. (WO)

Register 34.18. TWAI_CLOCK_DIVIDER_REG (0x007C)



TWAI_CD Configures the divisor of the external CLKOUT pin. (R/W)

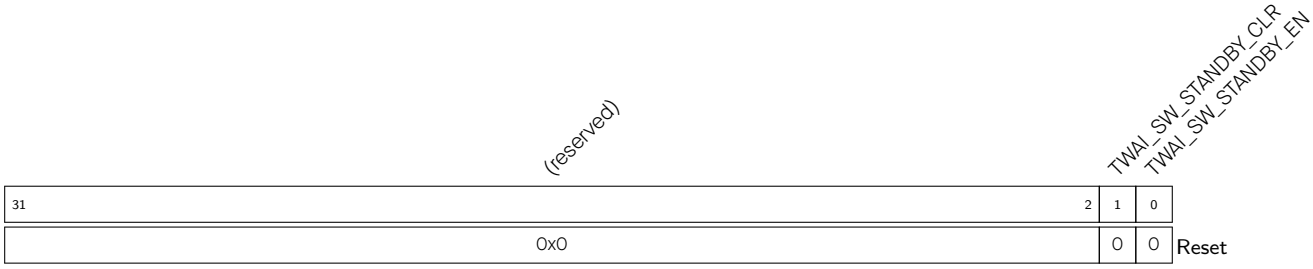
TWAI_CLOCK_OFF Configures whether or not to enable the external CLKOUT pin in Reset mode.

0: Enable the external CLKOUT pin

1: Disable the external CLKOUT pin

(RO | R/W)

Register 34.19. TWAI_SW_STANDBY_CFG_REG (0x0080)



TWAI_SW_STANDBY_CLR Configures whether to clear standby signals with software.

0: No effect

1: Clear standby signals

(R/W | R/W)

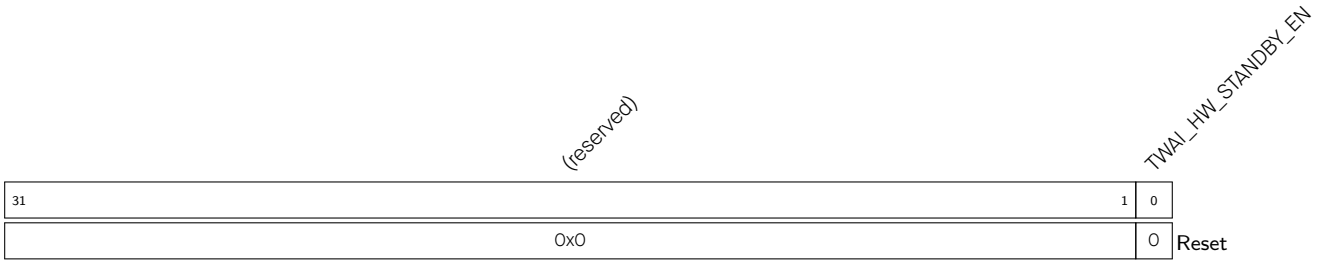
TWAI_SW_STANDBY_EN Configures whether to set standby signals with software.

0: No effect

1: Set standby signals

(R/W | R/W)

Register 34.20. TWAI_HW_CFG_REG (0x0084)



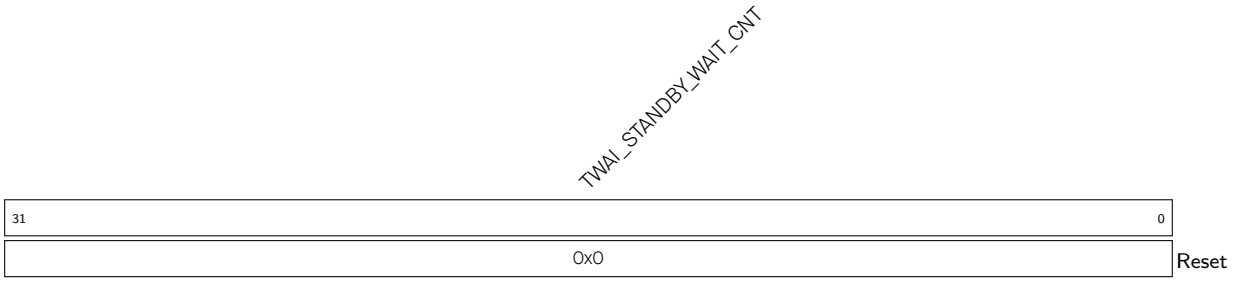
TWAI_HW_STANDBY_EN Configures whether to enable the standby function for hardware.

0: No effect

1: Enable the standby function for hardware

(R/W | R/W)

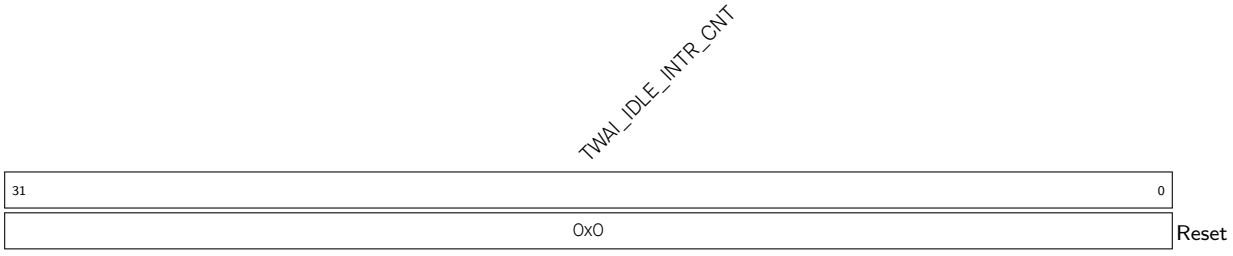
Register 34.21. TWAI_HW_STANDBY_CNT_REG (0x0070)



TWAI_STANDBY_WAIT_CNT Configures the time required before hardware triggers the standby signal after entering idle status. (R/W | R/W)

Measurement unit: TWAI controller clock cycles.

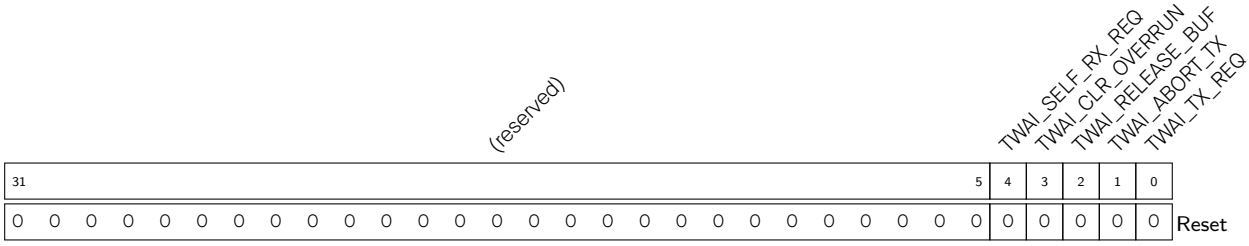
Register 34.22. TWAI_IDLE_INTR_CNT_REG (0x0070)



TWAI_IDLE_INTR_CNT Configures the time required before hardware generates the bus idle status interrupt signal after entering idle status. (R/W | R/W)

Measurement unit: TWAI controller clock cycles.

Register 34.23. TWAI_CMD_REG (0x0004)



TWAI_TX_REQ Configures whether to drive nodes to start transmission.

0: No effect

1: Drive nodes to start transmission

(WO)

TWAI_ABORT_TX Configures whether to cancel a pending transmission request.

0: No effect

1: Cancel a pending transmission request

(WO)

TWAI_RELEASE_BUF Configures whether to release the RX buffer.

0: No effect

1: Release the RX buffer

(WO)

TWAI_CLR_OVERRUN Configures whether to clear the data overrun status bit.

0: No effect

1: Clear the data overrun status bit

(WO)

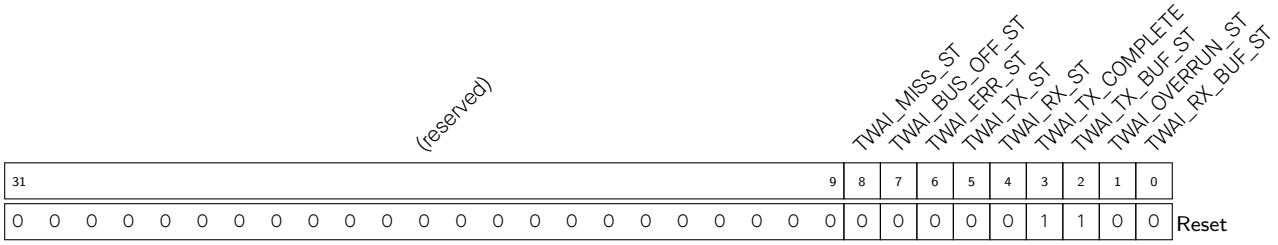
TWAI_SELF_RX_REQ Configures whether to allow a message be transmitted and received simultaneously.

0: No effect

1: Allow a message to be transmitted and received simultaneously

(WO)

Register 34.24. TWAI_STATUS_REG (0x0008)



TWAI_RX_BUF_ST Represents whether or not the RX buffer is empty.

0: Empty

1: Not empty, with at least one received data packet.

(RO)

TWAI_OVERRUN_ST Represents whether or not the RX FIFO is full.

0: Not full

1: Full, and data overrun has occurred

(RO)

TWAI_TX_BUF_ST Represents whether or not the TX buffer is empty.

0: Not empty

1: Empty, and the CPU may write a message into it

(RO)

TWAI_TX_COMPLETE Represents whether or not the TWAI controller has received a packet from the bus.

0: Not received

1: Received

(RO)

TWAI_RX_ST Represents whether or not the TWAI Controller is receiving a message from the bus.

0: Not receiving

1: Receiving

(RO)

TWAI_TX_ST Represents whether or not the TWAI Controller is transmitting a message to the bus.

0: Not transmitting

1: Transmitting

(RO)

TWAI_ERR_ST Represents at least one of the RX/TX error counter has reached or exceeded the value set in register TWAI_ERR_WARNING_LIMIT_REG. (RO)

TWAI_BUS_OFF_ST Represents whether or not the TWAI Controller involves in bus activities in bus-off status.

0: Involved

1: No longer involved

(RO)

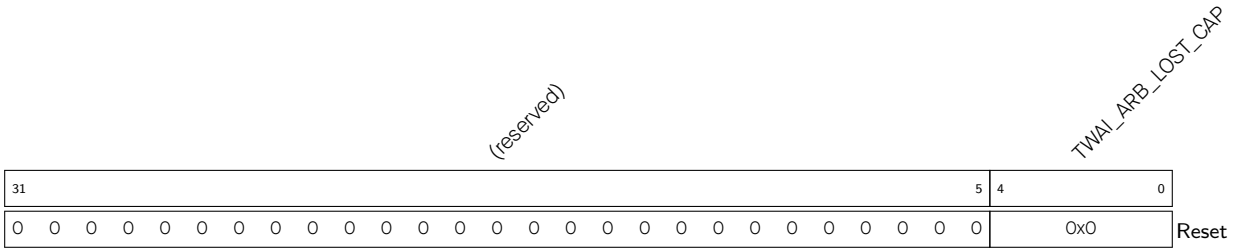
TWAI_MISS_ST Represents whether or not the data packet in the RX FIFO is complete.

0: The current packet is complete

1: The current packet is missing

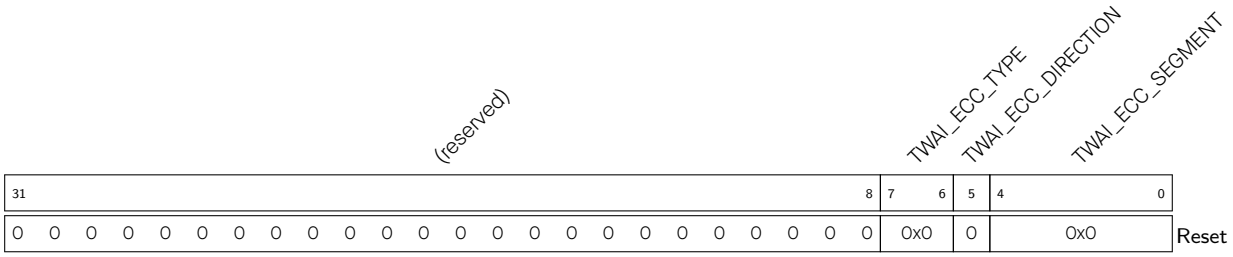
(RO)

Register 34.25. TWAI_ARB_LOST_CAP_REG (0x002C)



TWAI_ARB_LOST_CAP Represents the bit position of lost arbitration. (RO)

Register 34.26. TWAI_ERR_CODE_CAP_REG (0x0030)



TWAI_ECC_SEGMENT Represents the location of errors, see Table 34.4-11 for details. (RO)

TWAI_ECC_DIRECTION Represents transmission direction of the node when error occurs.

0: Error occurs when transmitting a message

1: Error occurs when receiving a message

(RO)

TWAI_ECC_TYPE Represents error types.

0: Bit error

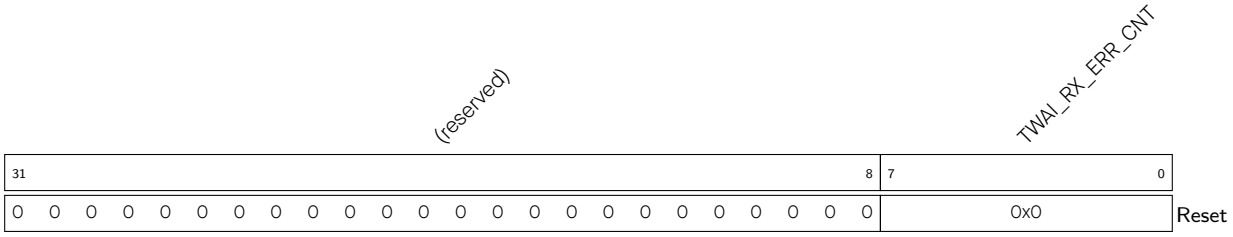
1: Form error

2: Stuff error

3: Others

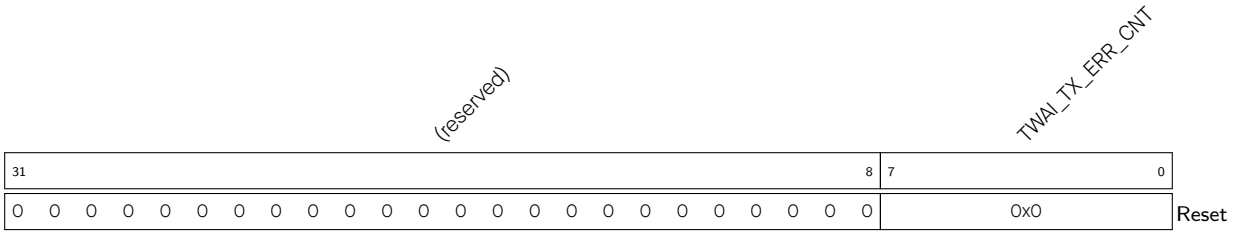
(RO)

Register 34.27. TWAI_RX_ERR_CNT_REG (0x0038)



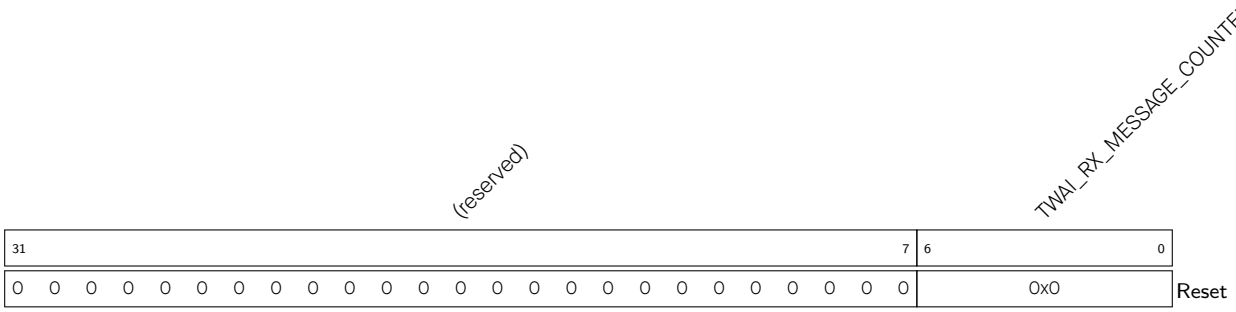
TWAI_RX_ERR_CNT The RX error counter register, reflects value changes in reception status. (RO | R/W)

Register 34.28. TWAI_TX_ERR_CNT_REG (0x003C)



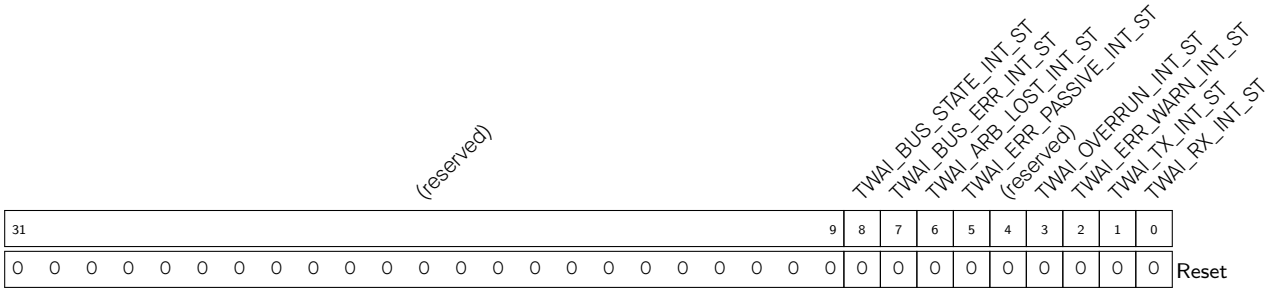
TWAI_TX_ERR_CNT The TX error counter register, reflects value changes in transmission status. (RO | R/W)

Register 34.29. TWAI_RX_MESSAGE_CNT_REG (0x0074)



TWAI_RX_MESSAGE_COUNTER Represents the number of messages available within the RX FIFO. (RO)

Register 34.30. TWAI_INT_ST_REG (0x000C)



TWAI_RX_INT_ST The masked interrupt status of TWAI_RX_INT. (RO)

TWAI_TX_INT_ST The masked interrupt status of TWAI_TX_INT. (RO)

TWAI_ERR_WARN_INT_ST The masked interrupt status of TWAI_ERR_WARN_INT. (RO)

TWAI_OVERRUN_INT_ST The masked interrupt status of TWAI_OVERRUN_INT. (RO)

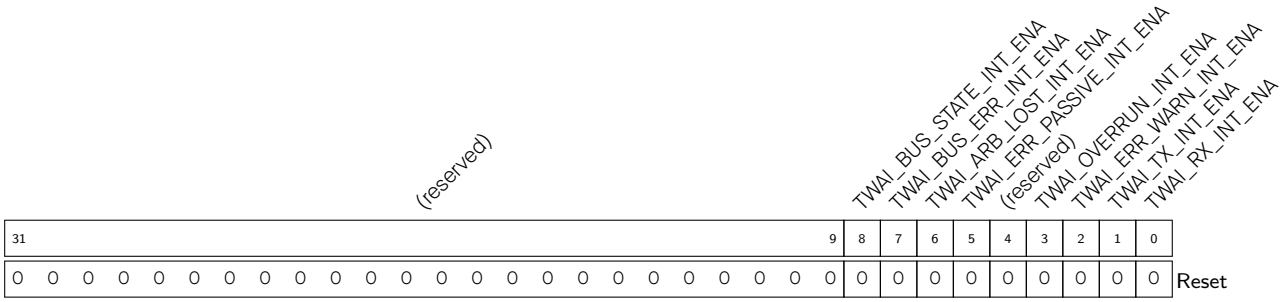
TWAI_ERR_PASSIVE_INT_ST The masked interrupt status of TWAI_ERR_PASSIVE_INT. (RO)

TWAI_ARB_LOST_INT_ST The masked interrupt status of TWAI_ARB_LOST_INT. (RO)

TWAI_BUS_ERR_INT_ST The masked interrupt status of TWAI_BUS_ERR_INT. (RO)

TWAI_BUS_STATE_INT_ST The masked interrupt status of TWAI_BUS_STATE_INT. (RO)

Register 34.31. TWAI_INT_ENA_REG (0x0010)



TWAI_RX_INT_ENA Write 1 to enable the receive interrupt. (R/W)

TWAI_TX_INT_ENA Write 1 to enable the transmit interrupt. (R/W)

TWAI_ERR_WARN_INT_ENA Write 1 to enable the error warning interrupt. (R/W)

TWAI_OVERRUN_INT_ENA Write 1 to enable the data overrun interrupt. (R/W)

TWAI_ERR_PASSIVE_INT_ENA Write 1 to enable the error passive interrupt. (R/W)

TWAI_ARB_LOST_INT_ENA Write 1 to enable the arbitration lost interrupt. (R/W)

TWAI_BUS_ERR_INT_ENA Write 1 to enable the bus error interrupt. (R/W)

TWAI_BUS_STATE_INT_ENA Write 1 to enable the bus state interrupt. (R/W)

Chapter 35

LED PWM Controller (LEDC)

35.1 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.

35.2 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).

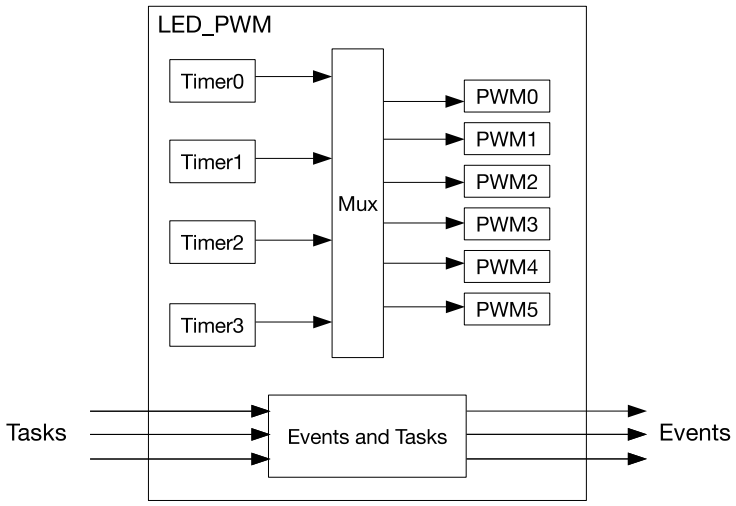


Figure 35.2-1. LED PWM Architecture

35.3 Functional Description

35.3.1 Architecture

Figure 35.2-1 shows the architecture of the LED PWM Controller.

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 LEDC_TIMER_SEL_CH*n*, and uses the timer's counter value timer*x*_cnt as a reference to generate its PWM signal.

Figure 35.3-1 illustrates the main functional blocks of the timer and the PWM generator.

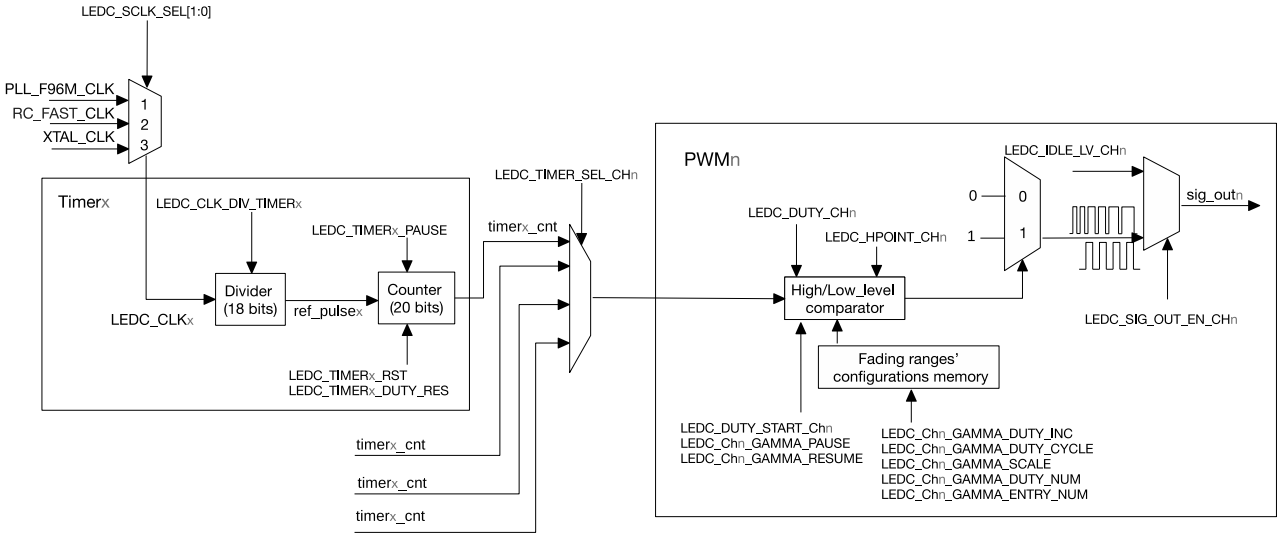


Figure 35.3-1. Timer and PWM Generator Block Diagram

35.3.2 Timers

Each timer in LED PWM Controller internally maintains a timebase counter. Referring to Figure 35.3-1, this clock signal used by the timebase counter is named ref_pulse*x*. All timers use the same clock source LEDC_CLK*x*, which is then passed through a clock divider to generate ref_pulse*x* for the counter.

35.3.2.1 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, and reset via software by setting the PCR_LEDC_RST_EN field in the PCR_LEDC_CONF_REG register.

Timers in LED PWM Controller choose their common clock source from one of the following clock signals: PLL_F96M_CLK, RC_FAST_CLK, and XTAL_CLK. The procedure for selecting a clock source signal for LEDC_CLK*x* is described below:

- PLL_F96M_CLK: Set LEDC_SCLK_SEL[1:0] to 1
- RC_FAST_CLK: Set LEDC_SCLK_SEL[1:0] to 2
- XTAL_CLK: Set LEDC_SCLK_SEL[1:0] to 3

The LEDC_CLK*x* signal will then be passed through the clock divider.

If LEDC_SCLK_SEL[1:0] is set to 0 (an invalid value), LED PWM Controller cannot perform any function due to no clock source.

For more information, please refer to Chapter 7 *Reset and Clock*.

35.3.2.2 Clock Divider Configuration

The LEDC_CLK*x* 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*x* divided by the divisor LEDC_CLK_DIV (see Figure 35.3-1).

The divisor LEDC_CLK_DIV can be non-integer values, and it 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 an integer (i.e., an integer prescaler). In other words, a ref_pulse*x* clock pulse is generated after every *A* LEDC_CLK*x* clock pulses.

However, when *B* is not 0, LEDC_CLK_DIV becomes a non-integer. The clock divider implements non-integer frequency division by generating a ref_pulse*x* clock pulse after every *A* and (*A*+1) LEDC_CLK*x* clock pulses alternately. 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*x* clock pulses

- A number of (256-*B*) ref_pulse*x* clock pulses are generated every *A* LEDC_CLK*x* clock pulses
- The ref_pulse*x* clock pulses generated every (*A*+1) pulses are evenly distributed amongst those generated every *A* pulses

Figure 35.3-2 illustrates the relation between LEDC_CLK*x* clock pulses and ref_pulse*x* clock pulses when LEDC_CLK_DIV is a non-integer.

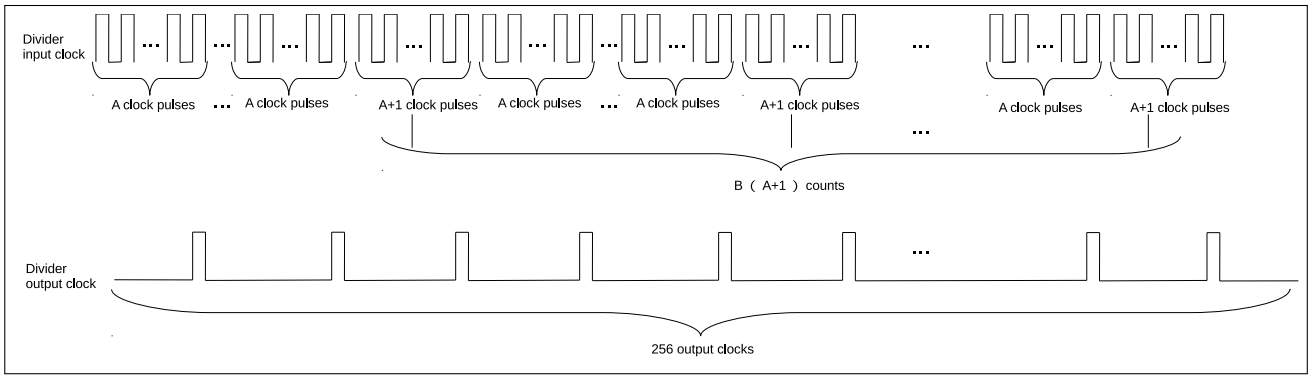


Figure 35.3-2. Frequency Division When LEDC_CLK_DIV is a Non-Integer Value

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.

35.3.2.3 20-Bit Counter

Each timer contains a 20-bit timebase counter that uses ref_pulse*x* as its reference clock (see Figure 35.3-1). The LEDC_TIMER*x*_DUTY_RES field configures the overflow value 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 35.3-3 shows the relationship between the counter and PWM resolution.

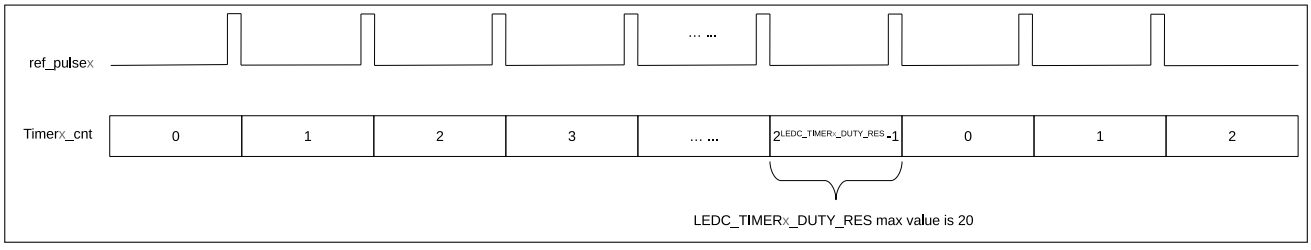


Figure 35.3-3. Relationship Between Counter And Resolution

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. Set LEDC_OVF_CNT_EN_CH*n* to enable the overflow counter
3. Configure LEDC_OVF_NUM_CH*n* with the number of counter overflows (that triggers an interrupt) minus 1

4. Set LEDC_OVF_CNT_CH*n*_INT_ENA to enable the overflow interrupt
5. Set LEDC_TIMER*x*_DUTY_RES to enable the timer and wait for a LEDC_OVF_CNT_CH*n*_INT interrupt

To change the overflow value at runtime, first set the LEDC_TIMER*x*_DUTY_RES field, and then set the LEDC_TIMER*x*_PARA_UP field. This will cause the newly configured values to take effect upon the next overflow of the counter. If LEDC_OVF_CNT_EN_CH*n* field is reconfigured, LEDC_PARA_UP_CH*n* should be set to apply the new configuration. In summary, these configuration values need to be updated by setting LEDC_TIMER*x*_PARA_UP or LEDC_PARA_UP_CH*n*. LEDC_TIMER*x*_PARA_UP and LEDC_PARA_UP_CH*n* will be automatically cleared by hardware.

Referring to Figure 35.3-1, the frequency of a PWM generator output signal (sig_out*n*) is dependent on the frequency of the timer's clock source LEDC_CLK*x*, the clock divisor LEDC_CLK_DIV, and the duty resolution (counter width) LEDC_TIMER*x*_DUTY_RES:

$$f_{\text{PWM}} = \frac{f_{\text{LEDC\_CLK}x}}{\text{LEDC\_CLK\_DIV} \cdot 2^{\text{LEDC\_TIMER}x\text{\_DUTY\_RES}}}$$

Based on the formula above, the desired duty resolution can be calculated as follows:

$$\text{LEDC\_TIMER}x\text{\_DUTY\_RES} = \log_2\left(\frac{f_{\text{LEDC\_CLK}x}}{f_{\text{PWM}} \cdot \text{LEDC\_CLK\_DIV}}\right)$$

Table 35.3-1 lists the commonly-used frequencies and their corresponding resolutions.

Table 35.3-1. Commonly-used Frequencies and Resolutions

LEDC_CLK*x*	PWM Frequency	Highest Resolution (bit) [1]	Lowest Resolution (bit) [2]
PLL_F96M_CLK (96 MHz)	1 kHz	17	6
PLL_F96M_CLK (96 MHz)	5 kHz	15	4
PLL_F96M_CLK (96 MHz)	10 kHz	14	3
XTAL_CLK (32 MHz)	1 kHz	15	4
XTAL_CLK (32 MHz)	4 kHz	13	2
RC_FAST_CLK (8 MHz)	1 kHz	13	2
RC_FAST_CLK (8 MHz)	2 kHz	12	1

[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.

35.3.3 PWM Generators

To generate a PWM signal, a PWM generator (PWM*n*) needs a timer (Timer*x*). Each PWM generator can be configured separately by setting LEDC_TIMER_SEL_CH*n* to use one of four timers to generate the PWM output.

As shown in Figure 35.3-1, each PWM generator has a comparator and two multiplexers. A PWM generator compares the timer's 20-bit counter value (Timer*x*_cnt) to two trigger values Hpoint*n* and Lpoint*n*. When the timer's counter value is equal to Hpoint*n* or Lpoint*n*, the PWM signal is high or low, respectively, as described below:

- If Timer*x*_cnt == Hpoint*n*, sig_out*n* is 1.
- If Timer*x*_cnt == Lpoint*n*, sig_out*n* is 0.

Figure 35.3-4 illustrates how Hpoint*n* and Lpoint*n* are used to generate the PWM output signal with a fixed duty cycle.

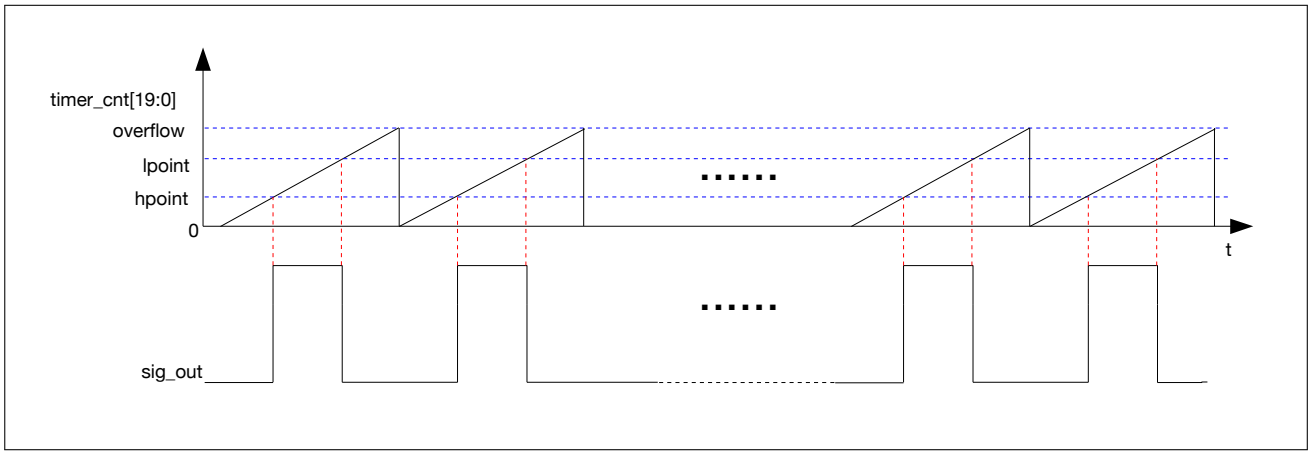


Figure 35.3-4. LED PWM Output Signal Diagram

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.

35.3.4 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.

35.3.4.1 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.

Below are the programming procedures:

1. Configure the LEDC_DUTY_CH*n* field with the initial value of Lpoint*n*.
2. Set the LEDC_DUTY_START_CH*n* field to enable Duty Cycle Fading. When this field is cleared, Duty Cycle Fading will be disabled.
3. Configure the direction via the LEDC_CH*n*_GAMMA_DUTY_INC field of the LEDC_CH*n*_GAMMA_WR_REG register. When this field is set or cleared, the Lpoint*n* will increment or decrement in the current configured range.
4. Configure the number of times the counter overflows per an increase or decrease of Lpoint*n* via the LEDC_CH*n*_GAMMA_DUTY_CYCLE field of the LEDC_CH*n*_GAMMA_WR_REG register. In other words, Lpoint*n* will increase or decrease after the counter overflows for LEDC_CH*n*_GAMMA_DUTY_CYCLE times.
5. Configure the amount by which Lpoint*n* increase or decrease in the configured range via the LEDC_CH*n*_GAMMA_SCALE field of the LEDC_CH*n*_GAMMA_WR_REG register.
6. Configure the number of fades via the LEDC_CH*n*_GAMMA_DUTY_NUM field of the LEDC_CH*n*_GAMMA_WR_REG register.
7. Write the duty cycle range number (0 in this case) to the LEDC_CH*n*_GAMMA_WR_ADDR field of the LEDC_CH*n*_GAMMA_WR_ADDR_REG register. This range number (from 0 to 15) specifies to which range the configurations in Step 3, 4, 5, and 6 apply. For linear duty cycle fading only the first range needs to be configured, so configure LEDC_CH*n*_GAMMA_WR_ADDR as 0.
8. 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.
9. 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.

After the above 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 35.3-5 shows a linear fading PWM signal.

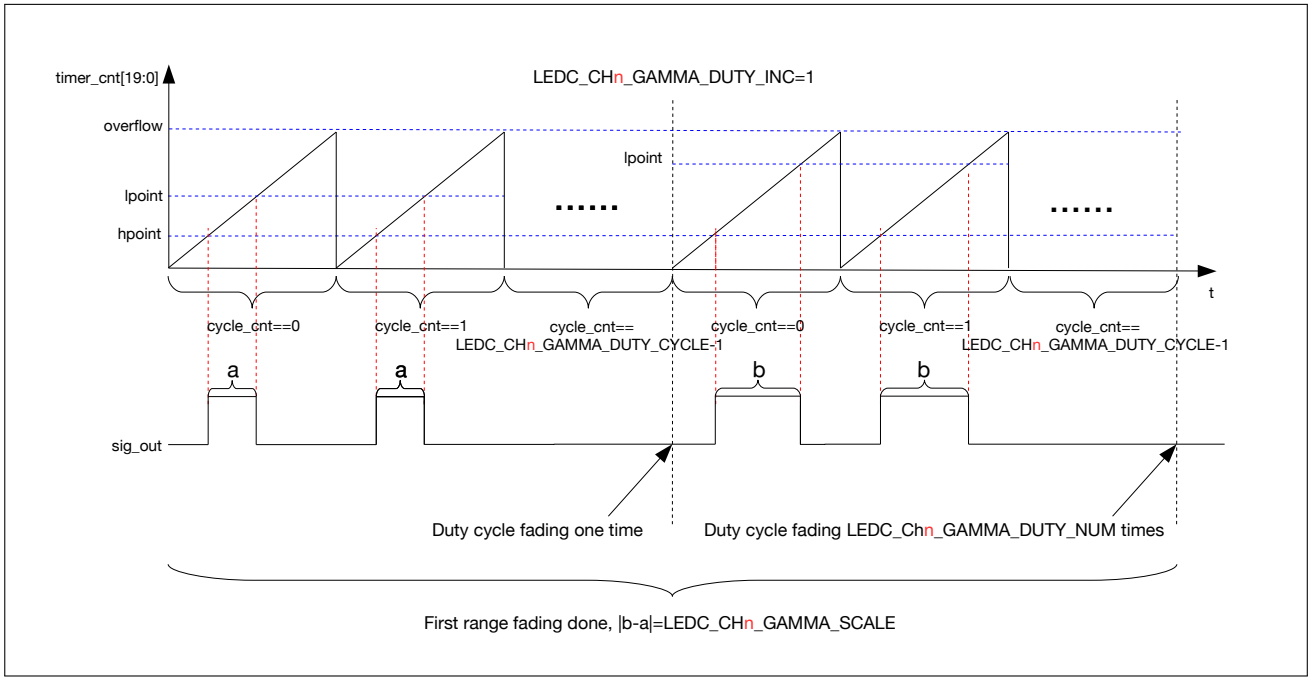


Figure 35.3-5. Output Signal of Linear Duty Cycle Fading

35.3.4.2 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.

Below are the programming procedures:

1. The same as Step 1 in Section 35.3.4.1.
2. The same as Step 2 in Section 35.3.4.1.
3. Configure multiple duty cycle ranges:
   (a) Configure the LEDC_CH*n*_GAMMA_DUTY_INC field of the LEDC_CH*n*_GAMMA_WR_REG register for the currently configured range.
   (b) Configure the LEDC_CH*n*_GAMMA_DUTY_CYCLE field of the LEDC_CH*n*_GAMMA_WR_REG register for the currently configured range.
   (c) Configure the LEDC_CH*n*_GAMMA_SCALE field of the LEDC_CH*n*_GAMMA_WR_REG register for the currently configured range.

(d) Configure the LEDC_CH*n*_GAMMA_DUTY_NUM field of the LEDC_CH*n*_GAMMA_WR_REG register for the currently configured range.

(e) Write the duty cycle range number (from 0 to 15) to the LEDC_CH*n*_GAMMA_WR_ADDR field of the LEDC_CH*n*_GAMMA_WR_ADDR_REG register. This range number specifies to which range the above configurations apply. It must start from 0 and increase by 1 for the next range to be configured.

(f) 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 16, and each can be configured independently.

4. 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.
5. 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 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. In range LEDC_CH*n*_GAMMA_WR_ADDR, 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 35.3-6 illustrates a gamma curve fading PWM signal.

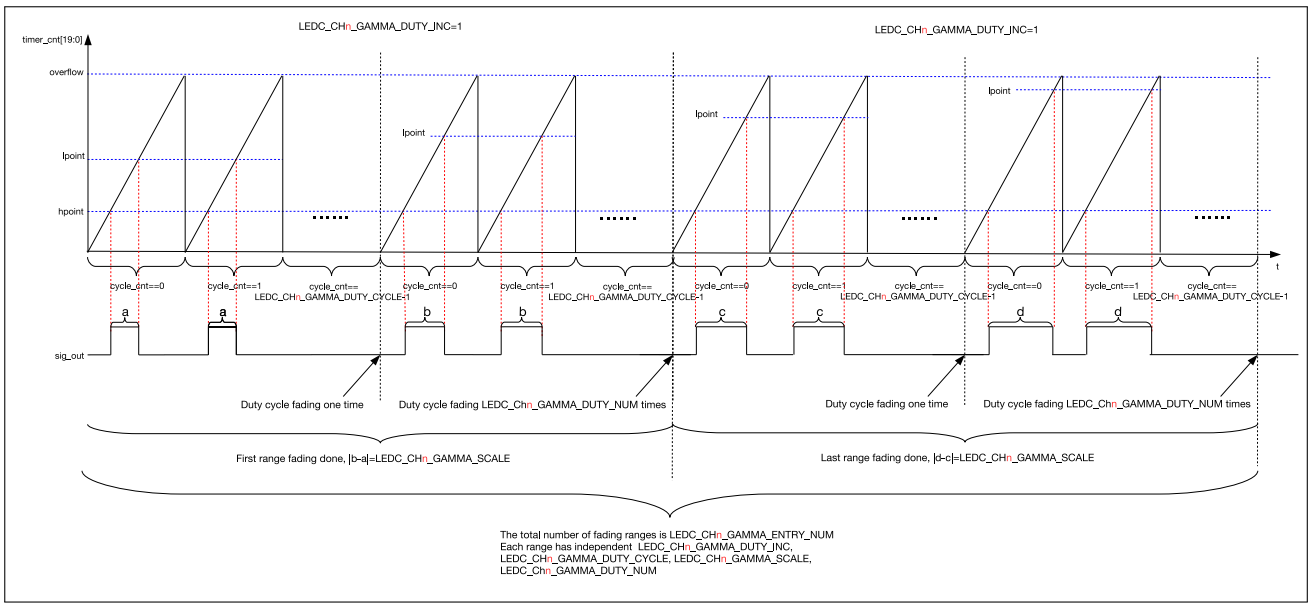


Figure 35.3-6. Output Signal of Gamma Curve Fading

35.3.4.3 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.

35.3.5 Event Task Matrix Feature

The LEDC on ESP32-H2 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 10 Event Task Matrix (SOC_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 35.5).

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 35.3-1.
- 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_TIME*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.

35.3.6 Interrupts

- 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. To trigger this interrupt, the field LEDC_OVF_CNT_CH*n*_INT_ENA of register LEDC_INT_ENA_REG should be set.
- LEDC_DUTY_CHNG_END_CH*n*_INT: Triggered when a fade on an LED PWM generator has finished. To trigger this interrupt, the field LEDC_DUTY_CHNG_END_CH*n*_INT_ENA of register LEDC_INT_ENA_REG should be set.
- LEDC_TIMER*x*_OVF_INT: Triggered when an LED PWM timer has reached its maximum counter value. To trigger this interrupt, the field LEDC_TIMER*x*_OVF_INT_ENA of register LEDC_INT_ENA_REG should be set.

35.4 Register Summary

The addresses in this section are relative to the LED PWM Controller base address provided in Table 4.3-2 in Chapter 4 *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_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_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_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_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_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_CONF1_REG	Configuration register 1 for channel 5	0x0070	R/W/SC
LEDC_EVT_TASK_EN0_REG	LEDC event task enable register 0	0x01A0	R/W
LEDC_EVT_TASK_EN1_REG	LEDC event task enable register 1	0x01A4	R/W
LEDC_EVT_TASK_EN2_REG	LEDC event task enable register 2	0x01A8	R/W
LEDC_TIMER0_CMP_REG	LEDC timer 0 value comparison register	0x01B0	R/W
LEDC_TIMER1_CMP_REG	LEDC timer 1 value comparison register	0x01B4	R/W
LEDC_TIMER2_CMP_REG	LEDC timer 2 value comparison register	0x01B8	R/W
LEDC_TIMER3_CMP_REG	LEDC timer 3 value comparison register	0x01BC	R/W
LEDC_TIMER0_CNT_CAP_REG	LEDC timer 0 counter value capture register	0x01C0	RO
LEDC_TIMER1_CNT_CAP_REG	LEDC timer 1 counter value capture register	0x01C4	RO
LEDC_TIMER2_CNT_CAP_REG	LEDC timer 2 counter value capture register	0x01C8	RO
LEDC_TIMER3_CNT_CAP_REG	LEDC timer 3 counter value capture register	0x01CC	RO
LEDC_CONF_REG	Global LEDC configuration register	0x01F0	R/W
Hpoint Register			
LEDC_CH0_HPOINT_REG	High point register for channel 0	0x0004	R/W
LEDC_CH1_HPOINT_REG	High point register for channel 1	0x0018	R/W
LEDC_CH2_HPOINT_REG	High point register for channel 2	0x002C	R/W
LEDC_CH3_HPOINT_REG	High point register for channel 3	0x0040	R/W
LEDC_CH4_HPOINT_REG	High point register for channel 4	0x0054	R/W
LEDC_CH5_HPOINT_REG	High point register for channel 5	0x0068	R/W
Duty Cycle Register			
LEDC_CH0_DUTY_REG	Initial duty cycle for channel 0	0x0008	R/W
LEDC_CH0_DUTY_R_REG	Current duty cycle for channel 0	0x0010	RO
LEDC_CH1_DUTY_REG	Initial duty cycle for channel 1	0x001C	R/W
LEDC_CH1_DUTY_R_REG	Current duty cycle for channel 1	0x0024	RO

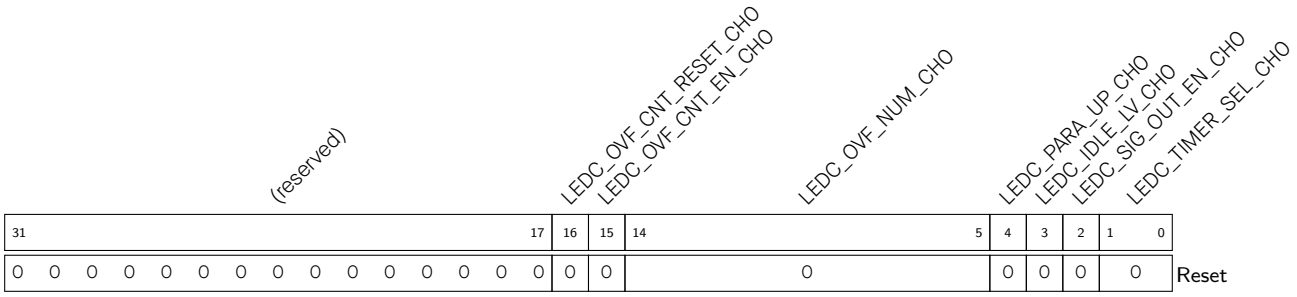
Name	Description	Address	Access
LEDC_CH2_DUTY_REG	Initial duty cycle for channel 2	0x0030	R/W
LEDC_CH2_DUTY_R_REG	Current duty cycle for channel 2	0x0038	RO
LEDC_CH3_DUTY_REG	Initial duty cycle for channel 3	0x0044	R/W
LEDC_CH3_DUTY_R_REG	Current duty cycle for channel 3	0x004C	RO
LEDC_CH4_DUTY_REG	Initial duty cycle for channel 4	0x0058	R/W
LEDC_CH4_DUTY_R_REG	Current duty cycle for channel 4	0x0060	RO
LEDC_CH5_DUTY_REG	Initial duty cycle for channel 5	0x006C	R/W
LEDC_CH5_DUTY_R_REG	Current duty cycle for channel 5	0x0074	RO
Timer Register			
LEDC_TIMER0_CONF_REG	Timer 0 configuration	0x00A0	varies
LEDC_TIMER0_VALUE_REG	Timer 0 current counter value	0x00A4	RO
LEDC_TIMER1_CONF_REG	Timer 1 configuration	0x00A8	varies
LEDC_TIMER1_VALUE_REG	Timer 1 current counter value	0x00AC	RO
LEDC_TIMER2_CONF_REG	Timer 2 configuration	0x00B0	varies
LEDC_TIMER2_VALUE_REG	Timer 2 current counter value	0x00B4	RO
LEDC_TIMER3_CONF_REG	Timer 3 configuration	0x00B8	varies
LEDC_TIMER3_VALUE_REG	Timer 3 current counter value	0x00BC	RO
Interrupt Register			
LEDC_INT_RAW_REG	Raw interrupt status	0x00C0	R/WTC/SS
LEDC_INT_ST_REG	Masked interrupt status	0x00C4	RO
LEDC_INT_ENA_REG	Interrupt enable bits	0x00C8	R/W
LEDC_INT_CLR_REG	Interrupt clear bits	0x00CC	WT
Gamma RAM Register			
LEDC_CH0_GAMMA_WR_REG	LEDC channel 0 gamma RAM write register	0x0100	R/W
LEDC_CH0_GAMMA_WR_ADDR_REG	LEDC channel 0 gamma RAM write address register	0x0104	R/W
LEDC_CH0_GAMMA_RD_ADDR_REG	LEDC channel 0 gamma RAM read address register	0x0108	R/W
LEDC_CH0_GAMMA_RD_DATA_REG	LEDC channel 0 gamma RAM read data register	0x010C	RO
LEDC_CH1_GAMMA_WR_REG	LEDC channel 1 gamma RAM write register	0x0110	R/W
LEDC_CH1_GAMMA_WR_ADDR_REG	LEDC channel 1 gamma RAM write address register	0x0114	R/W
LEDC_CH1_GAMMA_RD_ADDR_REG	LEDC channel 1 gamma RAM read address register	0x0118	R/W
LEDC_CH1_GAMMA_RD_DATA_REG	LEDC channel 1 gamma RAM read data register	0x011C	RO
LEDC_CH2_GAMMA_WR_REG	LEDC channel 2 gamma RAM write register	0x0120	R/W
LEDC_CH2_GAMMA_WR_ADDR_REG	LEDC channel 2 gamma RAM write address register	0x0124	R/W
LEDC_CH2_GAMMA_RD_ADDR_REG	LEDC channel 2 gamma RAM read address register	0x0128	R/W
LEDC_CH2_GAMMA_RD_DATA_REG	LEDC channel 2 gamma RAM read data register	0x012C	RO

Name	Description	Address	Access
LEDC_CH3_GAMMA_WR_REG	LEDC channel 3 gamma RAM write register	0x0130	R/W
LEDC_CH3_GAMMA_WR_ADDR_REG	LEDC channel 3 gamma RAM write address register	0x0134	R/W
LEDC_CH3_GAMMA_RD_ADDR_REG	LEDC channel 3 gamma RAM read address register	0x0138	R/W
LEDC_CH3_GAMMA_RD_DATA_REG	LEDC channel 3 gamma RAM read data register	0x013C	RO
LEDC_CH4_GAMMA_WR_REG	LEDC channel 4 gamma RAM write register	0x0140	R/W
LEDC_CH4_GAMMA_WR_ADDR_REG	LEDC channel 4 gamma RAM write address register	0x0144	R/W
LEDC_CH4_GAMMA_RD_ADDR_REG	LEDC channel 4 gamma RAM read address register	0x0148	R/W
LEDC_CH4_GAMMA_RD_DATA_REG	LEDC channel 4 gamma RAM read data register	0x014C	RO
LEDC_CH5_GAMMA_WR_REG	LEDC channel 5 gamma RAM write register	0x0150	R/W
LEDC_CH5_GAMMA_WR_ADDR_REG	LEDC channel 5 gamma RAM write address register	0x0154	R/W
LEDC_CH5_GAMMA_RD_ADDR_REG	LEDC channel 5 gamma RAM read address register	0x0158	R/W
LEDC_CH5_GAMMA_RD_DATA_REG	LEDC channel 5 gamma RAM read data register	0x015C	RO
Gamma Configuration Register			
LEDC_CH0_GAMMA_CONF_REG	LEDC channel 0 gamma configuration register	0x0180	varies
LEDC_CH1_GAMMA_CONF_REG	LEDC channel 1 gamma configuration register	0x0184	varies
LEDC_CH2_GAMMA_CONF_REG	LEDC channel 2 gamma configuration register	0x0188	varies
LEDC_CH3_GAMMA_CONF_REG	LEDC channel 3 gamma configuration register	0x018C	varies
LEDC_CH4_GAMMA_CONF_REG	LEDC channel 4 gamma configuration register	0x0190	varies
LEDC_CH5_GAMMA_CONF_REG	LEDC channel 5 gamma configuration register	0x0194	varies
Version Register			
LEDC_DATE_REG	Version control register	0x01FC	R/W

35.5 Registers

The addresses in this section are relative to LED PWM Controller base address provided in Table 4.3-2 in Chapter 4 *System and Memory*.

Register 35.1. LEDC_CH*n*_CONF0_REG (*n*: 0-5) (0x0000+0x14n*)**



LEDC_TIMER_SEL_CH*n* Configures the timer for for channel *n*.

0: timer 0

1: timer 1

2: timer 2

3: timer 3

(R/W)

LEDC_SIG_OUT_EN_CH*n* Configures whether or not to enable signal output on channel *n*.

0: Disable

1: Enable

(R/W)

LEDC_IDLE_LV_CH*n* Configures the output level when channel *n* is inactive (when LEDC_SIG_OUT_EN_CH*n* is 0). (R/W)

LEDC_PARA_UP_CH*n* Configures whether or not to update LEDC_HPOINT_CH*n*, LEDC_DUTY_START_CH*n*, LEDC_SIG_OUT_EN_CH*n*, LEDC_TIMER_SEL_CH*n*, and LEDC_OVF_CNT_EN_CH*n* fields for channel *n*.

0: Invalid. No effect

1: Update

(WT)

LEDC_OVF_NUM_CH*n* Configures the maximum overflow times 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 or not to enable the overflow counter of channel *n*.

0: Disable

1: Enable

(R/W)

Continued on the next page...

Register 35.1. LEDC_CH*n*_CONF0_REG (*n*: 0-5) (0x0000+0x14**n*)

Continued from the previous page...

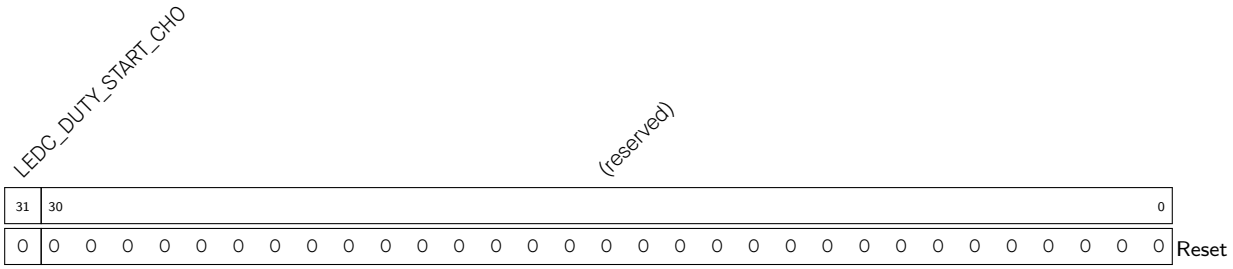
LEDC_OVF_CNT_RESET_CH*n* Set this bit to reset the ovf_cnt of channel *n*.

0: Invalid. No effect

1: Reset

(WT)

Register 35.2. LEDC_CH*n*_CONF1_REG (*n*: 0-5) (0x000C+0x14**n*)



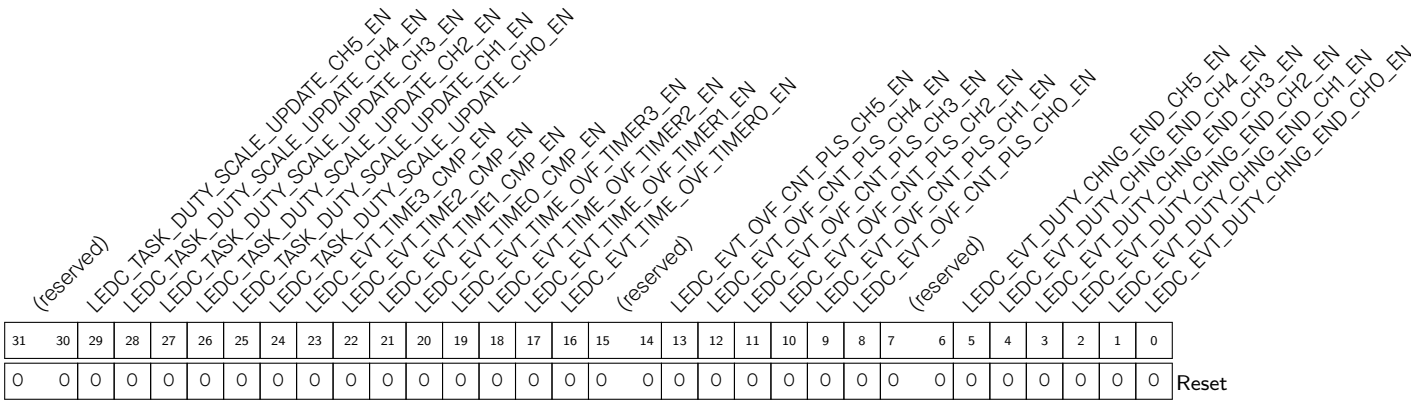
LEDC_DUTY_START_CH*n* Configures whether or not to enable duty cycle fading.

0: Disable

1: Enable

(R/W/SC)

Register 35.3. LEDC_EVT_TASK_EN0_REG (0x01A0)



LEDC_EVT_DUTY_CHNG_END_CH*n*_EN Configures whether or not 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 Configures whether or not to enable the LEDC_EVT_OVF_CNT_PLS_CH*n* event.

0: Disable

1: Enable

(R/W)

LEDC_EVT_TIME_OVF_TIMER*x*_EN Configures whether or not to enable the LEDC_EVT_TIME_OVF_TIMER*x* event event.

0: Disable

1: Enable

(R/W)

LEDC_EVT_TIME*n*_CMP_EN Configures whether or not to enable the LEDC_EVT_TIME*n*_CMP event.

0: Disable

1: Enable

(R/W)

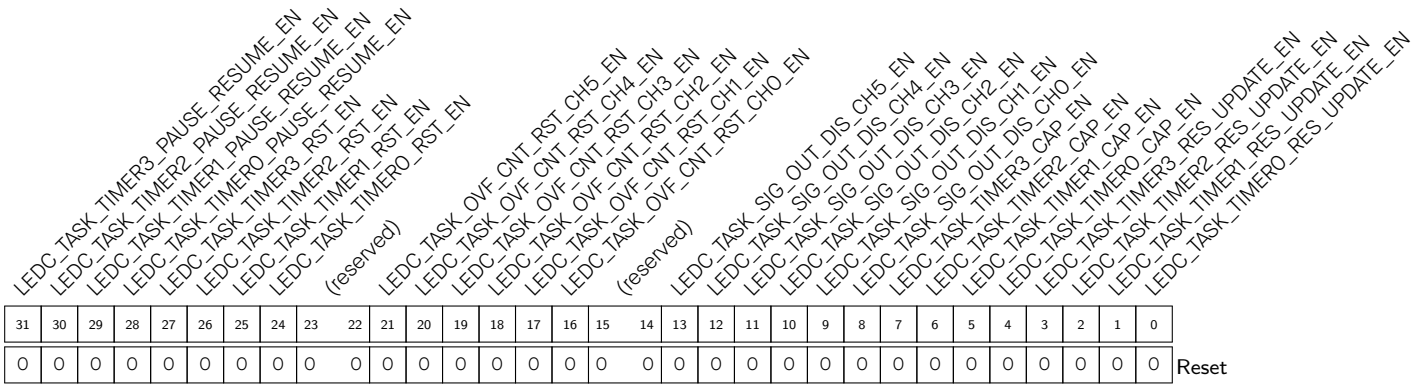
LEDC_TASK_DUTY_SCALE_UPDATE_CH*n*_EN Configures whether or not to enable the LEDC_TASK_DUTY_SCALE_UPDATE_CH*n* task.

0: Disable

1: Enable

(R/W)

Register 35.4. LEDC_EVT_TASK_EN1_REG (0x01A4)



31	30	29	28	27	26	25	24	23 22	21	20	19	18	17	16	15 14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
0	0	0	0	0	0	0	0	0 0	0	0	0	0	0	0	0 0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Reset

LEDC_TASK_TIMER*x*_RES_UPDATE_EN Configures whether or not to enable the LEDC_TASK_TIMER*x*_RES_UPDATE task.

0: Disable

1: Enable

(R/W)

LEDC_TASK_TIMER*x*_CAP_EN Configures whether or not to enable the LEDC_TASK_TIMER*x*_CAP task.

0: Disable

1: Enable

(R/W)

LEDC_TASK_SIG_OUT_DIS_CH*n*_EN Configures whether or not to enable the LEDC_TASK_SIG_OUT_DIS_CH*n* task.

0: Disable

1: Enable

(R/W)

LEDC_TASK_OVF_CNT_RST_CH*n*_EN Configures whether or not to enable the LEDC_TASK_OVF_CNT_RST_CH*n* task.

0: Disable

1: Enable

(R/W)

LEDC_TASK_TIMER*x*_RST_EN Configures whether or not to enable the LEDC_TASK_TIMER*x*_RST task.

0: Disable

1: Enable

(R/W)

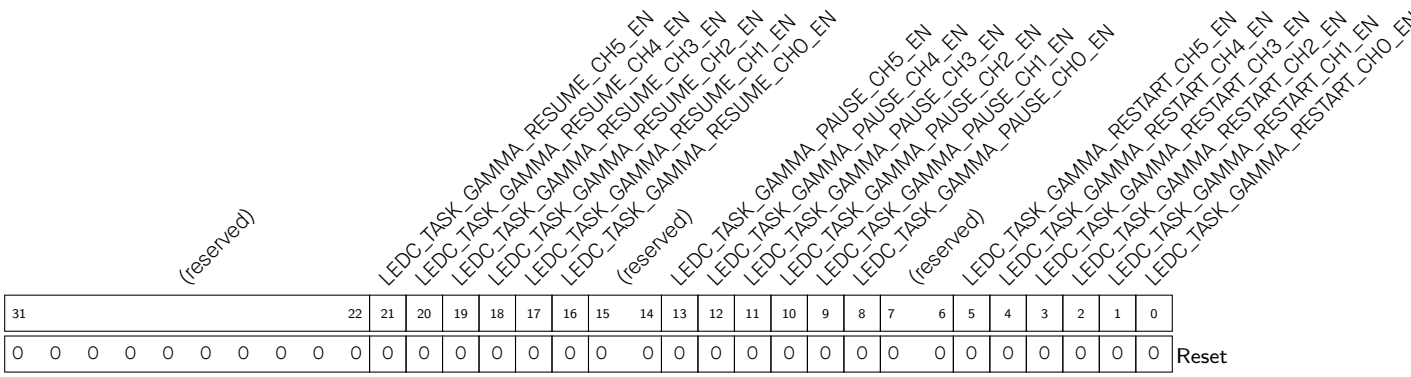
LEDC_TASK_TIMER*x*_PAUSE_RESUME_EN Configures whether or not to enable the LEDC_TASK_TIMER*x*_RESUME and LEDC_TASK_TIMER*x*_PAUSE task.

0: Disable

1: Enable

(R/W)

Register 35.5. LEDC_EVT_TASK_EN2_REG (0x01A8)



LEDC_TASK_GAMMA_RESTART_CH*n*_EN Configures whether or not to enable the LEDC_TASK_GAMMA_RESTART_CH*n* task.

0: Disable

1: Enable

(R/W)

LEDC_TASK_GAMMA_PAUSE_CH*n*_EN Configures whether or not to enable the LEDC_TASK_GAMMA_PAUSE_CH*n* task.

0: Disable

1: Enable

(R/W)

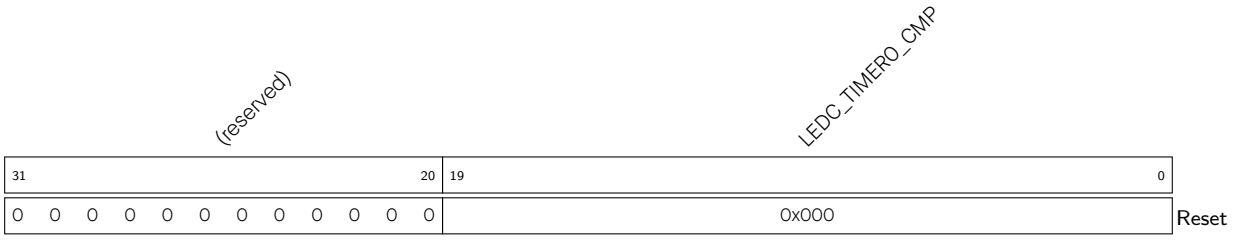
LEDC_TASK_GAMMA_RESUME_CH*n*_EN Configures whether or not to enable the LEDC_TASK_GAMMA_RESUME_CH*n* task.

0: Disable

1: Enable

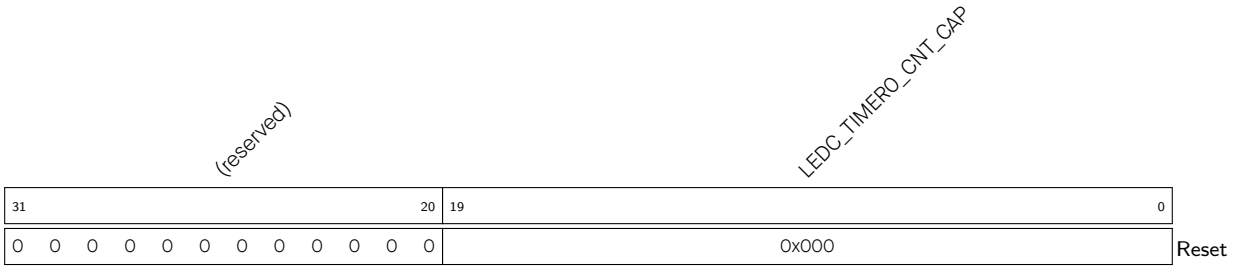
(R/W)

Register 35.6. LEDC_TIMER*x*_CMP_REG (*x*: 0-3) (0x01B0+0x4**n*)



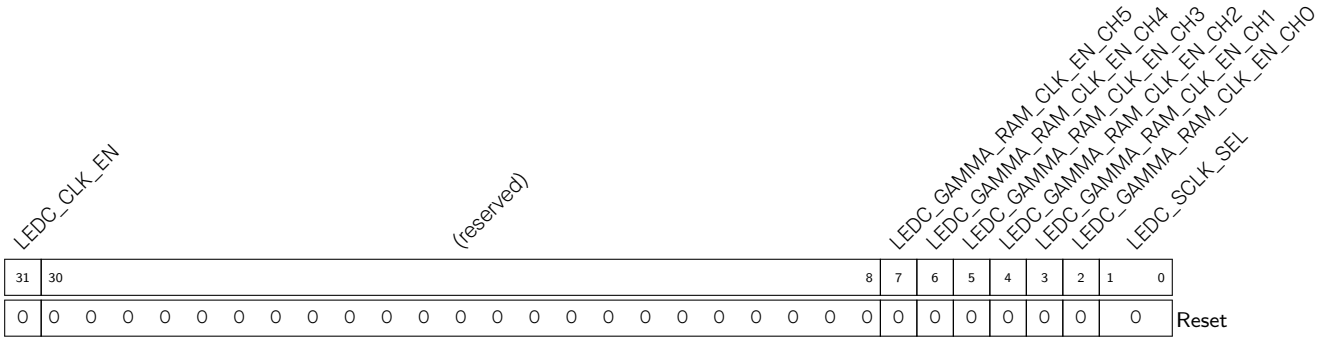
LEDC_TIMER*x*_CMP Configures the comparison value for LEDC timer *n*. (R/W)

Register 35.7. LEDC_TIMER*x*_CNT_CAP_REG (*x*: 0-3) (0x01C0+0x4**n*)



LEDC_TIMER*x*_CNT_CAP Represents the captured LEDC timer *n* counter value. (RO)

Register 35.8. LEDC_CONF_REG (0x01F0)



LEDC_SCLK_SEL Configures the clock source for the four timers.

0: PLL_F96M_CLK

1: RC_FAST_CLK

2: XTAL_CLK

3: Invalid. No effect

(R/W)

LEDC_GAMMA_RAM_CLK_EN_CH*n* Configures when to enable register clock.

0: Support clock only when application reads or writes gamma RAM.

1: Force clock on for gamma RAM.

(R/W)

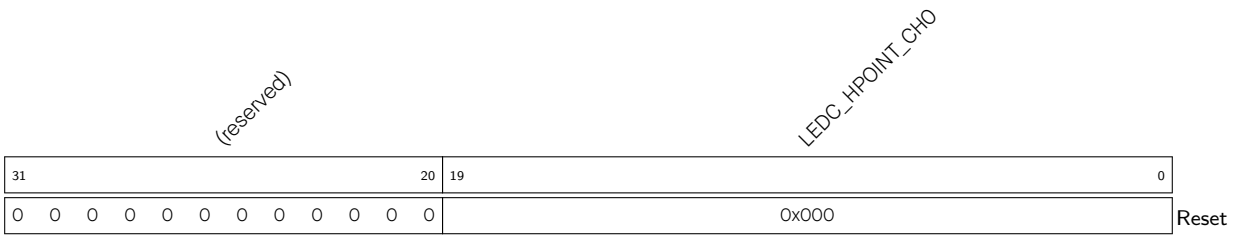
LEDC_CLK_EN Configures when to enable register clock.

0: Support clock only when application writes registers.

1: Force clock on for registers.

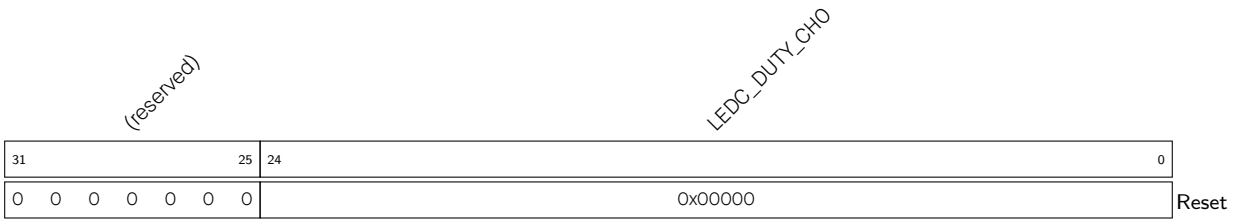
(R/W)

Register 35.9. LEDC_CH*n*_HPOINT_REG (*n*: 0-5) (0x0004+0x14*n*)



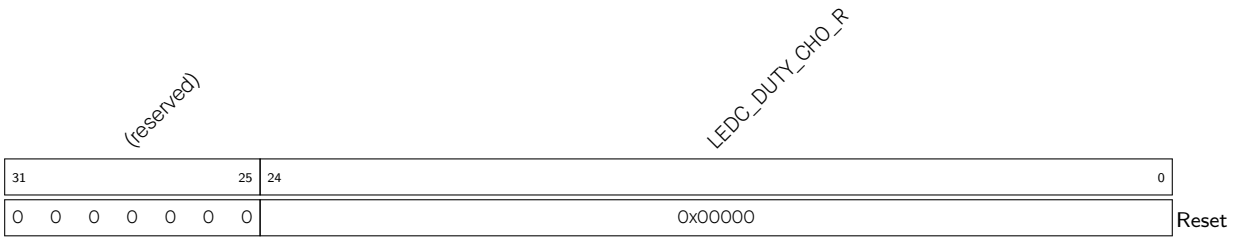
LEDC_HPOINT_CH*n* Configures the value of Hpoint. (R/W)

Register 35.10. LEDC_CH*n*_DUTY_REG (*n*: 0-5) (0x0008+0x14*n*)



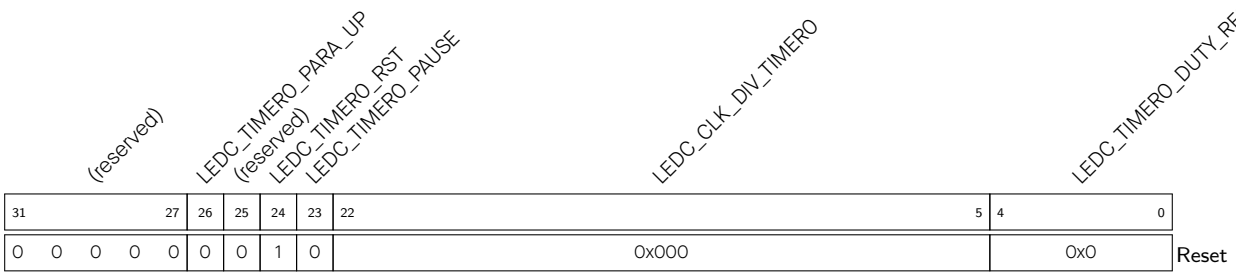
LEDC_DUTY_CH*n* Configures the initial value of Lpoint. (R/W)

Register 35.11. LEDC_CH*n*_DUTY_R_REG (*n*: 0-5) (0x0010+0x14*n*)



LEDC_DUTY_CH*n*_R Represents the current duty cycle of the output signal on channel *n*. (RO)

Register 35.12. LEDC_TIMER*x*_CONF_REG (*x*: 0-3) (0x00A0+0x8n*)**



LEDC_TIMER*x*_DUTY_RES Configures the duty cycle resolution (the width of the counter in timer *n*). (R/W)

LEDC_CLK_DIV_TIMER*x* Configures the divisor for the divider in timer *n*.

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 or not to suspend the counter in timer *n*.

0: Not suspend

1: Suspend

(R/W)

LEDC_TIMER*x*_RST Configures whether or not to reset timer *n* (the counter will show 0 after reset).

0: Not reset

1: Reset

(R/W)

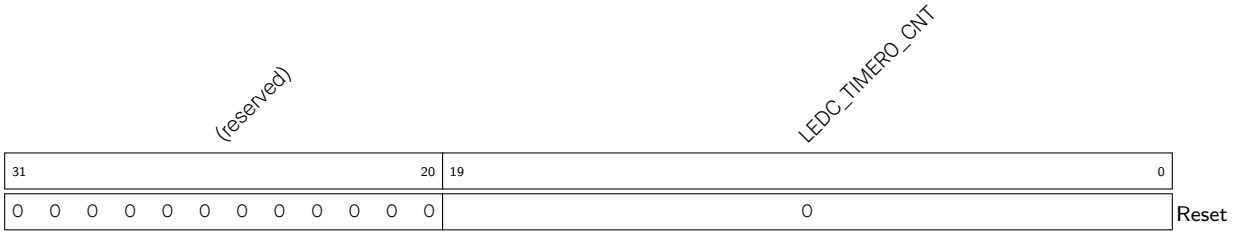
LEDC_TIMER*x*_PARA_UP Configures whether or not to update LEDC_CLK_DIV_TIMER*x* and LEDC_TIMER*x*_DUTY_RES.

0: Invalid. No effect

1: Update

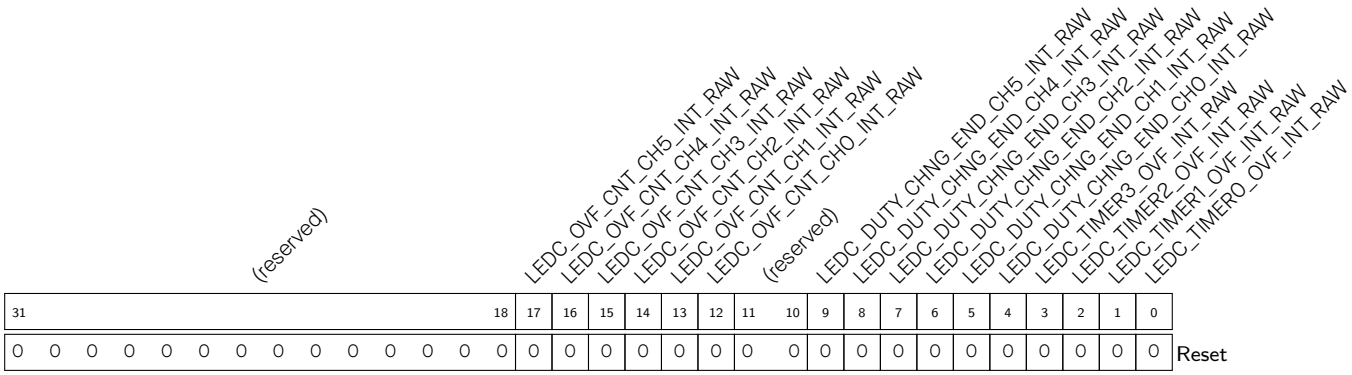
(WT)

Register 35.13. LEDC_TIMER*x*_VALUE_REG (*x*: 0-3) (0x00A4+0x8n*)**



LEDC_TIMER*x*_CNT Represents the current counter value of timer *n*. (RO)

Register 35.14. LEDC_INT_RAW_REG (0x00C0)

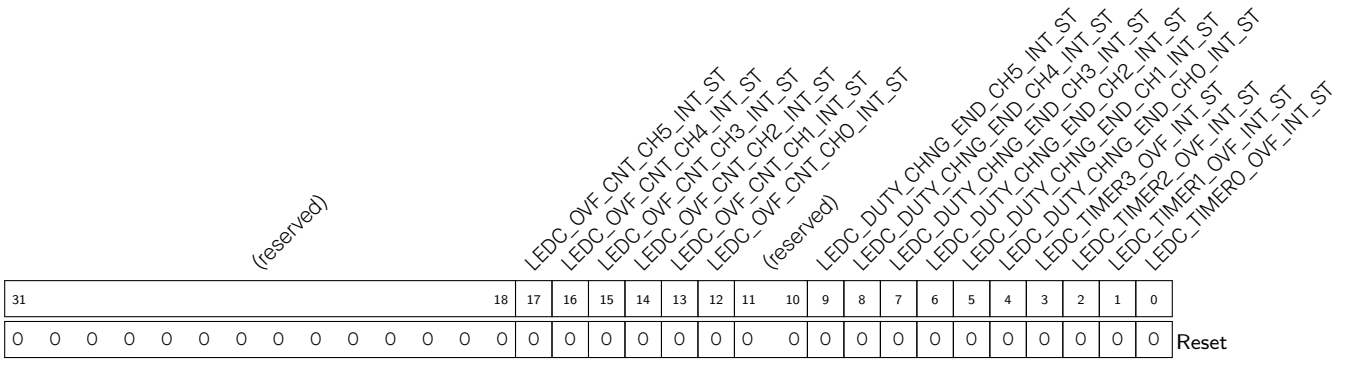


LEDC_TIMER*x*_OVF_INT_RAW The raw interrupt status of LEDC_TIMER*x*_OVF_INT. (R/WTC/SS)

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

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

Register 35.15. LEDC_INT_ST_REG (0x00C4)

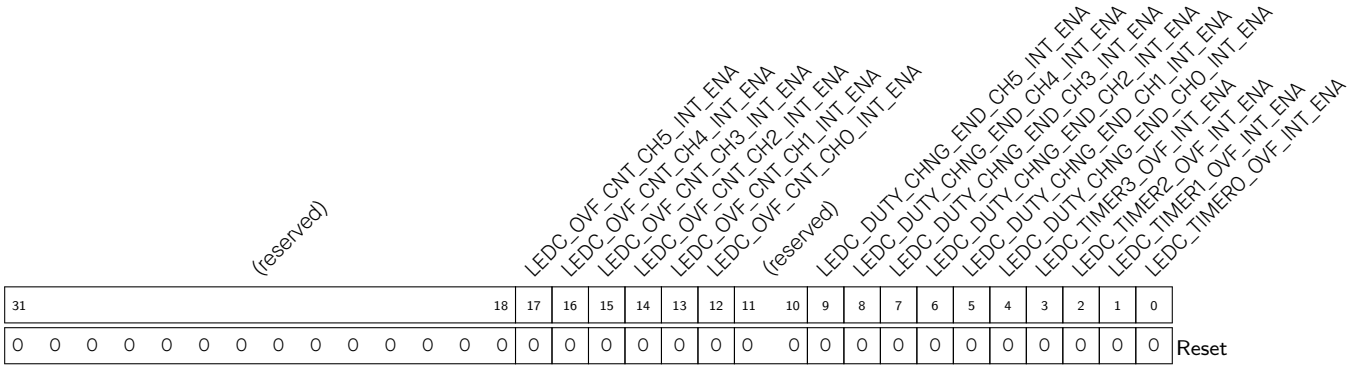


LEDC_TIMER*x*_OVF_INT_ST The masked interrupt status of LEDC_TIMER*x*_OVF_INT. Valid only when LEDC_TIMER*x*_OVF_INT_ENA is 1. (RO)

LEDC_DUTY_CHNG_END_CH*n*_INT_ST The masked interrupt status of LEDC_DUTY_CHNG_END_CH*n*_INT. Valid only when LEDC_DUTY_CHNG_END_CH*n*_INT_ENA is 1. (RO)

LEDC_OVF_CNT_CH*n*_INT_ST The masked interrupt status of LEDC_OVF_CNT_CH*n*_INT. Valid only when LEDC_OVF_CNT_CH*n*_INT_ENA is 1. (RO)

Register 35.16. LEDC_INT_ENA_REG (0x00C8)

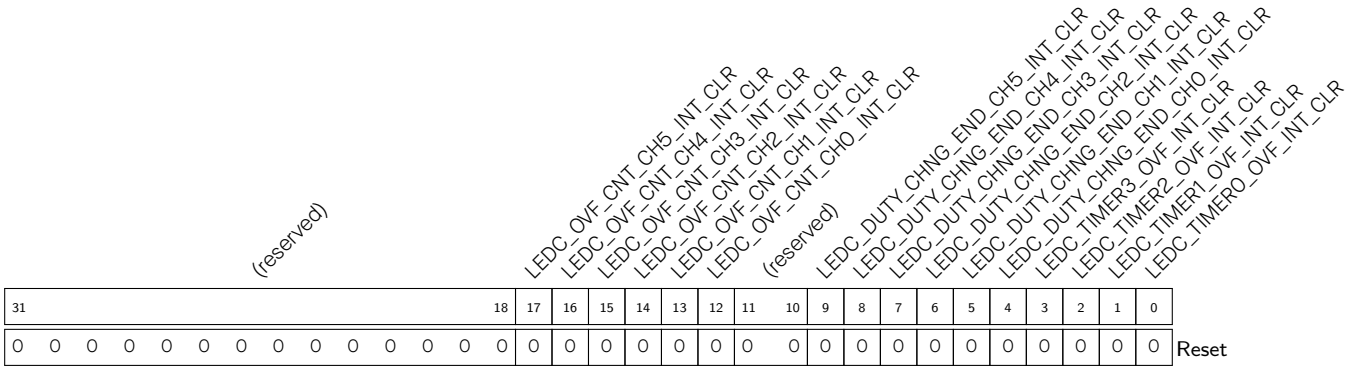


LEDC_TIMER*x*_OVF_INT_ENA Write 1 to enable LEDC_TIMER*x*_OVF_INT. (R/W)

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

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

Register 35.17. LEDC_INT_CLR_REG (0x00CC)

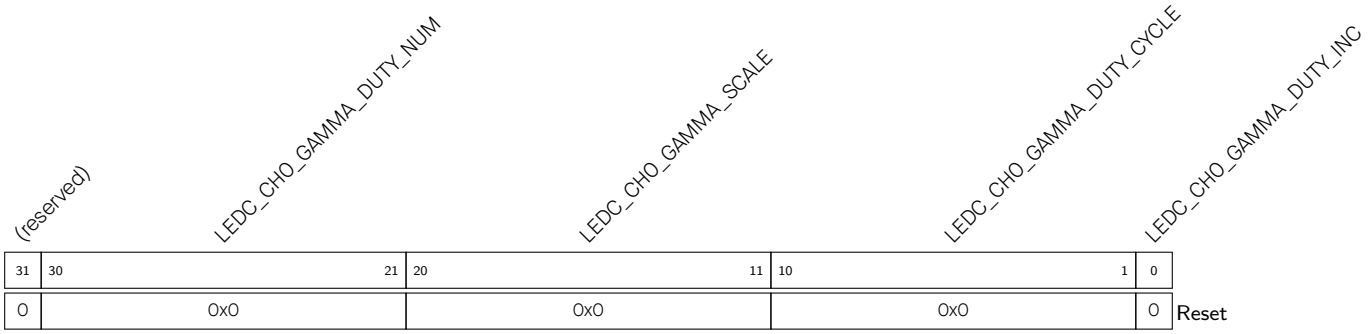


LEDC_TIMER*x*_OVF_INT_CLR Write 1 to clear LEDC_TIMER*x*_OVF_INT. (WT)

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

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

Register 35.18. LEDC_CH*n*_GAMMA_WR_REG (*n*: 0-5) (0x0100+0x10*n)



LEDC_CH*n*_GAMMA_DUTY_INC Configures the direction of duty cycle fading for PWM signals on channel *n*.

0: Decrease.

1: Increase

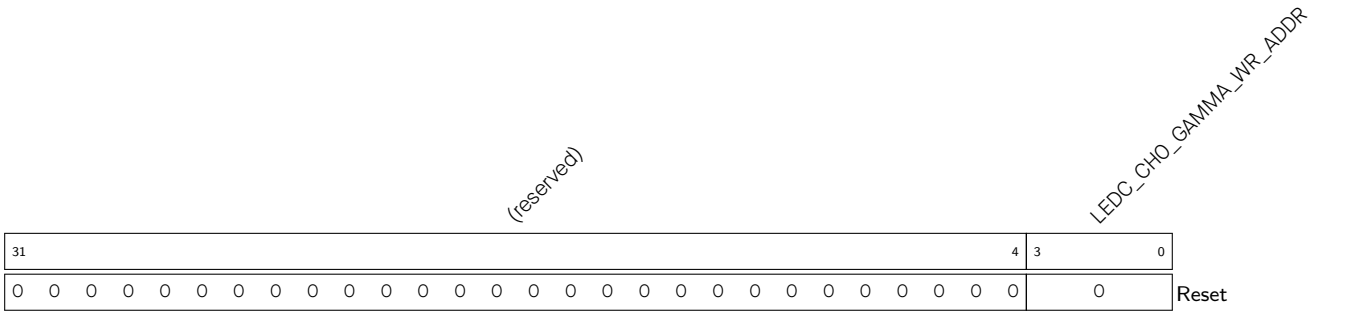
(R/W)

LEDC_CH*n*_GAMMA_DUTY_CYCLE Configures the number of times the counter overflows per an duty cycle fade. (R/W)

LEDC_CH*n*_GAMMA_SCALE Configures the amount by which Lpoint*n* increase or decrease each time. (R/W)

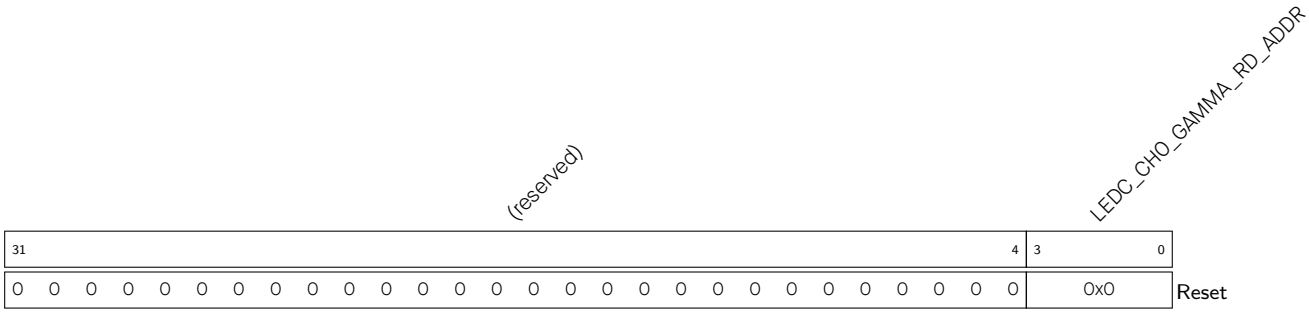
LEDC_CH*n*_GAMMA_DUTY_NUM Configures the number of fades in a duty cycle range. (R/W)

Register 35.19. LEDC_CH*n*_GAMMA_WR_ADDR_REG (*n*: 0-5) (0x0104+0x10*n)



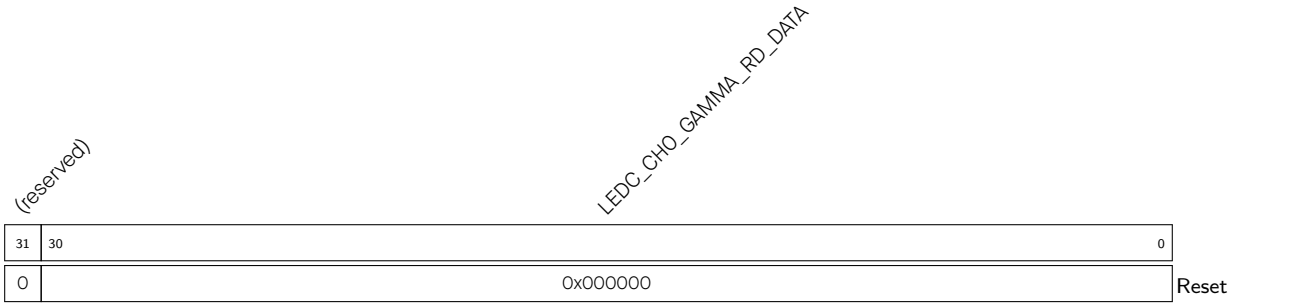
LEDC_CH*n*_GAMMA_WR_ADDR Configures LEDC channel *n* gamma RAM write address. (R/W)

Register 35.20. LEDC_CH*n*_GAMMA_RD_ADDR_REG (*n*: 0-5) (0x0108+0x10*n*)



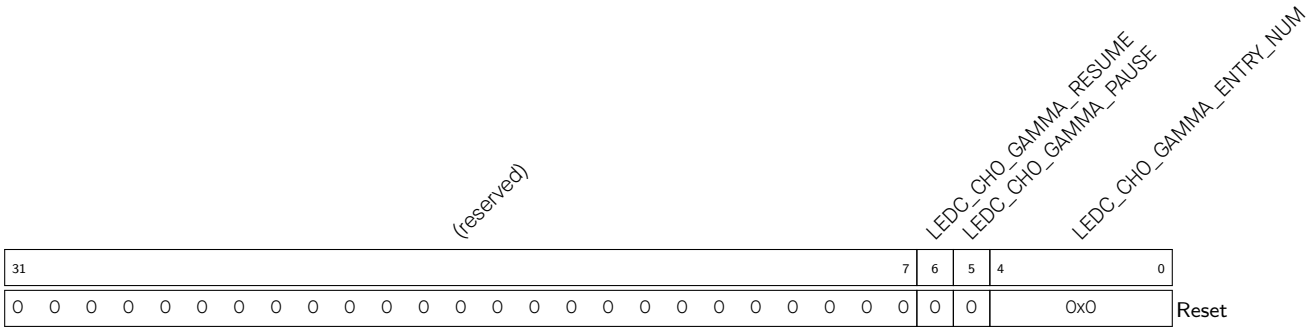
LEDC_CH*n*_GAMMA_RD_ADDR Configures LEDC channel *n* gamma RAM read address. (R/W)

Register 35.21. LEDC_CH*n*_GAMMA_RD_DATA_REG (*n*: 0-5) (0x010C+0x10*n*)



LEDC_CH*n*_GAMMA_RD_DATA Represents data read from gamma RAM via LEDC channel *n*. (RO)

Register 35.22. LEDC_CH*n*_GAMMA_CONF_REG (*n*: 0-5) (0x0180+0x4**n*)



LEDC_CH*n*_GAMMA_ENTRY_NUM Configures the number of duty cycle ranges. Maximum value is 16. (R/W)

LEDC_CH*n*_GAMMA_PAUSE Configures whether or not to pause duty cycle fading.

0: Invalid. No effect

1: Pause

(WT)

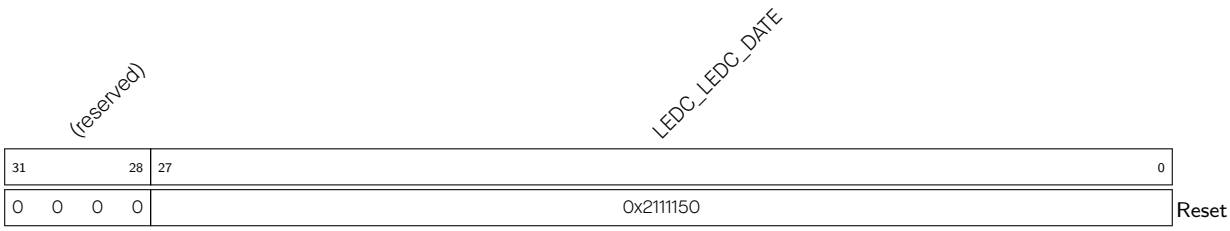
LEDC_CH*n*_GAMMA_RESUME Configures whether or not to resume duty cycle fading.

0: Invalid. No effect

1: Resume

(WT)

Register 35.23. LEDC_DATE_REG (0x01FC)



LEDC_LEDC_DATE Version control register. (R/W)

Chapter 36

Motor Control PWM (MCPWM)

36.1 Overview

The **M**otor **C**ontrol **P**ulse **W**idth **M**odulator (MCPWM) peripheral is intended for motor and power control. It provides six PWM outputs that can be set up to operate in several topologies. One common topology uses a pair of PWM outputs driving an H-bridge to control motor rotation speed and rotation direction.

The MCPWM can be divided into five main modules: PWM timers, PWM operators, Capture module, Event Task Matrix (ETM) module, and Fault Detection module. Each PWM timer provides timing references that can either run freely or be synced to other timers or external sources. Each PWM operator has all the necessary control resources to generate waveform pairs for one PWM channel. The Capture module is used for systems that need to accurately time external events. The ETM module responds to tasks received by the MCPWM, generating corresponding events depending on the state of motion. The Fault Detection module is used to capture external faults, allowing the system to respond by choice.

ESP32-H2 has one MCPWM peripheral, which is MCPWM0.

36.2 Features

An MCPWM peripheral has one clock divider (prescaler), three PWM timers, three PWM operators, a Capture module, an ETM module, and a Fault Detection module. MCPWM's core clock (PWM_CORE_CLK) can be selected from three clock sources: PLL_F96M_CLK, XTAL_CLK, and RC_FAST_CLK (configured by PCR_PWM_CLKM_SEL field in PCR register). Figure 36.2-1 shows the submodules inside MCPWM and the signals on the interface. PWM timers are used for generating timing references. The PWM operators generate the desired waveform based on the timing references. Any PWM operator can be configured to use the timing references of any PWM timers. Different PWM operators can use the same PWM timer's timing reference to generate PWM signals, or different PWM timers' values to generate separate PWM signals. Different PWM timers can also be synchronized together.

Below is an overview of the submodules' functionality in Figure 36.2-1:

- PWM Timers 0, 1, and 2:
  - Every PWM timer has a dedicated 8-bit clock prescaler.
  - The 16-bit counter in the PWM timer can work in count-up mode, count-down mode, or count-up-down mode.
  - A hardware sync or software sync can trigger a reload on the PWM timer with a phase register. It will also trigger the prescaler's restart so that the timer's clock can also be synced. The source of the hard sync can come from any GPIO or any other PWM timer's sync out signal. The source of the soft sync comes from writing a toggle value to the MCPWM_TIMER*x*_SYNC_SW bit.

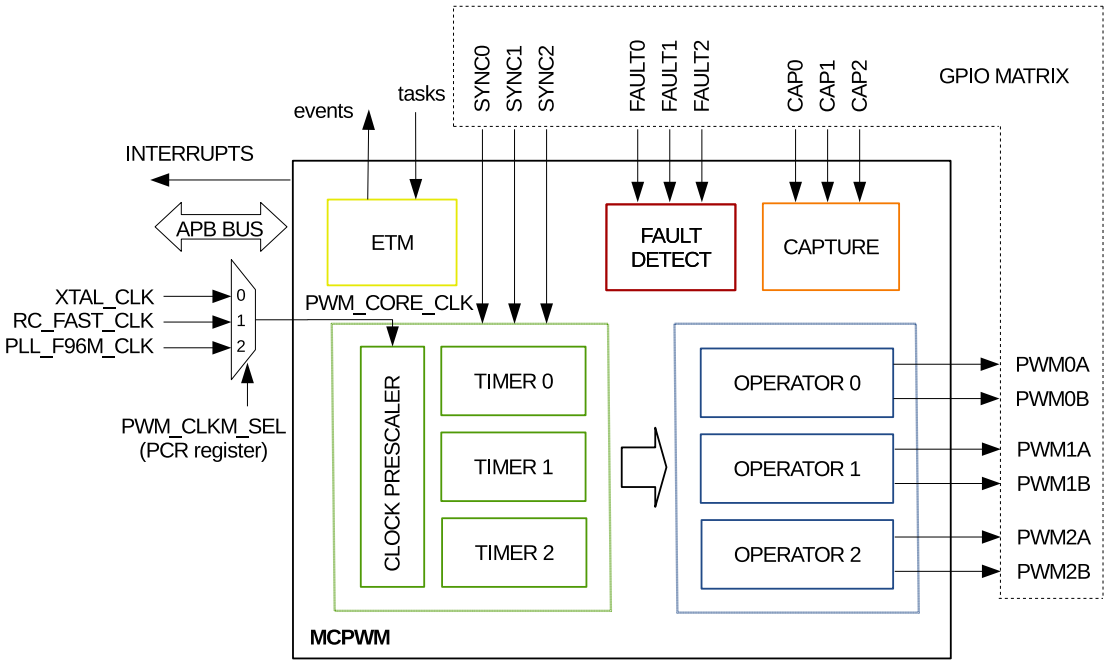


Figure 36.2-1. MCPWM Module Overview

- PWM Operators 0, 1, and 2:
  - Every PWM operator has two PWM outputs: PWM*x*A and PWM*x*B. They can work independently, in symmetric or asymmetric configurations.
  - The control of the PWM signal can be updated asynchronously.
  - Configurable dead time on rising and falling edges; each set up independently.
  - All events can trigger CPU interrupts.
  - Modulating of PWM output by high-frequency carrier signals, useful when gate drivers are insulated with a transformer.
  - Period, time stamps, and important control registers have shadow registers with flexible updating methods.
- Fault Detection Module:
  - Programmable fault handling in both cycle-by-cycle mode and one-shot mode.
  - A fault condition can force the PWM output to either high or low logic levels.
- Capture Module:
  - Clock of the capture module is the same as MCPWM's core clock.
  - Speed measurement of rotating machinery
  - Measurement of elapsed time between position sensor pulses
  - Period and duty cycle measurement of pulse train signals
  - Decoding current or voltage amplitude derived from duty-cycle-encoded signals of current/voltage sensors

  - Three individual capture channels, each of which with a time-stamp register (32-bit)
  - Selection of edge polarity and prescaling of input capture signals
  - The capture timer can sync with a PWM timer or external signals.
  - Interrupt on each of the three capture channels

- ETM Module:
  - Generation of different events depending on the different running states of each timer and operator.
  - Each timer and operator responds to its corresponding task and automatically performs the corresponding operation.
  - Each event and task can be enabled independently. When an event is not enabled, the corresponding event will not be generated. When a task is not enabled, the corresponding task will not be responded to.

36.3 Modules

36.3.1 Overview

This section lists the configuration parameters of key modules. For information on adjusting a specific parameter, e.g., synchronization source of PWM timer, please refer to Section 36.3.2 for details.

36.3.1.1 Prescaler Module

Figure 36.3-1. Prescaler Module

Configuration option:

- Divide MCPWM's core clock.

36.3.1.2 Timer Module

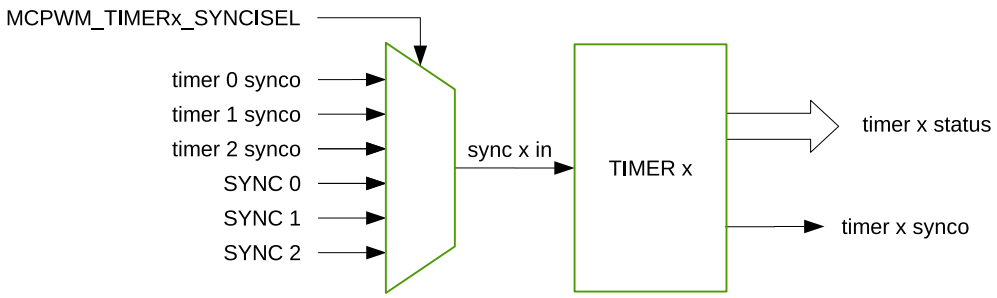


Figure 36.3-2. Timer Module

Configuration options:

- Configure the PWM timer frequency or period
- Configure the working mode for the timer:
  - Count-Up Mode: for asymmetric PWM outputs
  - Count-Down Mode: for asymmetric PWM outputs
  - Count-Up-Down Mode: for symmetric PWM outputs
- Configure the reloading phase (including the value and the direction) used during software and hardware synchronization
- Synchronize the PWM timers with each other. Either hardware or software synchronization may be used
- Configure the source of the PWM timer's synchronization input to one of the seven sources below:
  - The three PWM timer's synchronization outputs
  - Three synchronization signals from the GPIO matrix: PWMn_SYNCO_IN, PWMn_SYNC1_IN, PWMn_SYNC2_IN

  - No synchronization input signal selected

- Configure the source of the PWM timer's synchronization output to one of the four sources below:
  - Synchronization input signal
  - Event generated when the value of the PWM timer is equal to zero
  - Event generated when the value of the PWM timer is equal to the period
  - Event generated when writing a toggle value to the MCPWM_TIMER*x*_SYNC_SW bit
- Configure the method of period updating

36.3.1.3 Operator Module

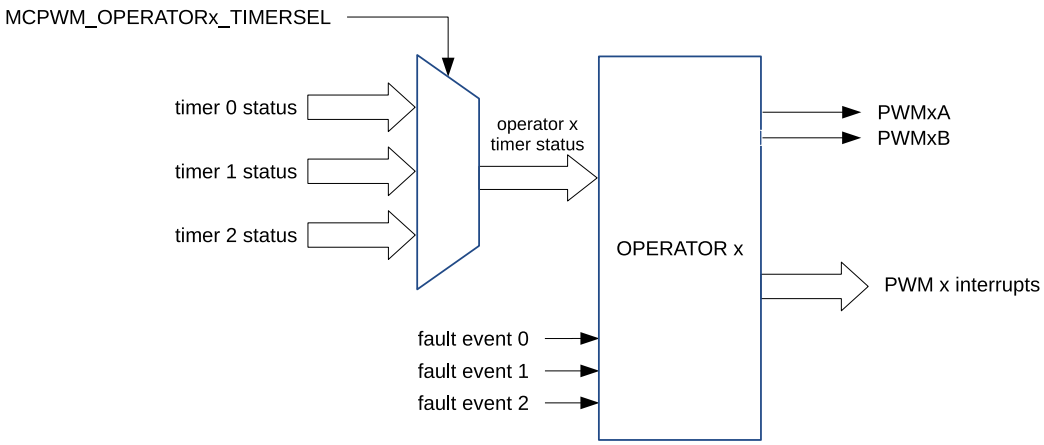


Figure 36.3-3. Operator Module

The configuration parameters of the operator module are shown in Table 36.3-1.

Table 36.3-1. Configuration Parameters of the Operator Submodule

Submodule	Configuration Parameter or Option

PWM Generator	• Configure the PWM duty cycle for PWM*x*A and/or PWM*x*B output
	• Configure at which time the timing events occur
	• Configure what action should be taken on timing events:
	  – Switch high or low of PWM*x*A and/or PWM*x*B outputs
	  – Toggle PWM*x*A and/or PWM*x*B outputs
	  – Take no action on outputs
	• Use direct s/w control to force the state of PWM outputs
	• Add a dead time to raising edge and/or failing edge on PWM outputs
	• Configure update method for this submodule

Submodule	Configuration Parameter or Option
Dead Time Generator	• Control of complementary dead time relationship between upper and lower switches • Specify the dead time on rising edge • Specify the dead time on falling edge • Bypass the dead time generator module. The PWM waveform will pass through without inserting dead time. • Allow PWM*x*B phase shifting with respect to the PWM*x*A output • Configure updating method for this submodule • Enable carrier and set up carrier frequency
PWM Carrier	• Configure the duration of the first pulse in the carrier waveform • Configure the duty cycle of the following pulses • Bypass the PWM carrier module. The PWM waveform will be passed through without modification • Configure if and how the PWM module should react the fault event signals
Fault Handler	• Specify the action taken when a fault event occurs: – Force PWM*x*A and/or PWM*x*B high – Force PWM*x*A and/or PWM*x*B low – Configure PWM*x*A and/or PWM*x*B to ignore any fault event • Configure how often the PWM should react to fault events: – One-shot – Cycle-by-cycle • Generate interrupts • Bypass the fault handler submodule entirely • Configure an option for cycle-by-cycle actions clearing • If desired, independently-configured actions can be taken when the time-base counter is counting down or up.

36.3.1.4 Fault Detection Module

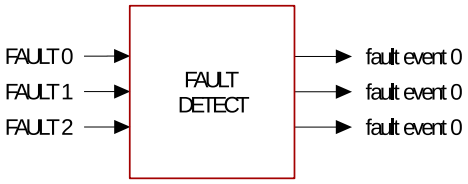


Figure 36.3-4. Fault Detection Module

Configuration options:

- Enable fault event generation and configure the polarity of fault event generated for every fault signal
- Generate fault event interrupts

36.3.1.5 Capture Module

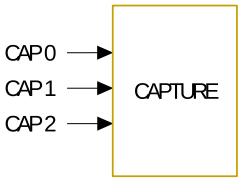


Figure 36.3-5. Capture Module

Configuration options:

- Select the edge polarity and prescale the capture input
- Set up a software-triggered capture
- Configure the capture timer's sync trigger and sync phase
- Software syncs the capture timer

36.3.1.6 ETM Module

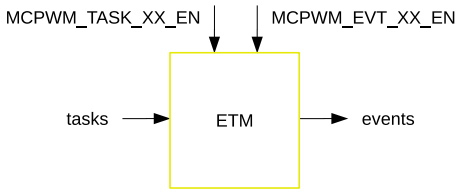


Figure 36.3-6. ETM Module

Configuration options:

- Each event and task can be enabled independently. When an event is not enabled, the corresponding event will not be generated. When a task is not enabled, the corresponding task will not be responded to.

36.3.2 PWM Timer Module

MCPWM has three PWM timer modules. Any of them can determine the necessary event timing for any of the three PWM operator modules. By using the synchronization signals from the GPIO matrix, built-in synchronization logic allows multiple PWM timer modules in this MCPWM peripheral to work together as a system.

36.3.2.1 Configurations of the PWM Timer Module

Users can configure the following functions of the PWM timer module:

- Control how often events occur by specifying the PWM timer frequency or period
- Configure a particular PWM timer to synchronize with other PWM timers or modules
- Get a PWM timer in phase with other PWM timers or modules.

- Configure the following timer counting modes: count-up, count-down, count-up-down
- Change the rate of the PWM timer clock (PT_CLK) with a prescaler. Each timer has its own prescaler configured with MCPWM_TIMER*x*_PRESCALE of the register MCPWM_TIMER0_CFG0_REG. The PWM timer increments or decrements at a slow pace, depending on the setting of this field. The new MCPWM_TIMER*x*_PRESCALE configuration value will take effect when the timer stops and starts counting again.

36.3.2.2 PWM Timer's Working Modes and Timing Event Generation

The PWM timer has three working modes, selected by the MCPWM_TIMER*x*_MOD (*x* = 0, 1, 2) timer mode field:

- Count-Up Mode:
  The PWM timer increments from zero until reaching the value configured in the period field. Once done, the PWM timer returns to zero and starts increasing again. PWM period = the value of the period field + 1. Note: The period field is MCPWM_TIMER*x*_PERIOD (*x* = 0, 1, 2), i.e., MCPWM_TIMER0_PERIOD, MCPWM_TIMER1_PERIOD, MCPWM_TIMER2_PERIOD.
- Count-Down Mode:
  The PWM timer decrements to zero, starting from the value configured in the period field. Once done, the PWM timer returns to the period value and starts decrementing again. In this case, the PWM period = the value of period field + 1.
- Count-Up-Down Mode:
  This is a combination of the two modes mentioned above. The PWM timer starts increasing from zero until the period value is reached. Then, the timer decreases back to zero. The PWM timer cycles incrementally and decrementally in this mode. The PWM period = the value of the period field × 2.

Figure 36.3-7 to 36.3-10 show PWM timer waveforms in different modes, including timer behavior during synchronization events. In Count-Up mode, the counting direction after synchronization is always counting up. In Count-Down mode, the counting direction after synchronization is always counting down. In Count-Up-Down Mode, the counting direction after synchronization can be chosen by setting the MCPWM_TIMER*x*_PHASE_DIRECTION.

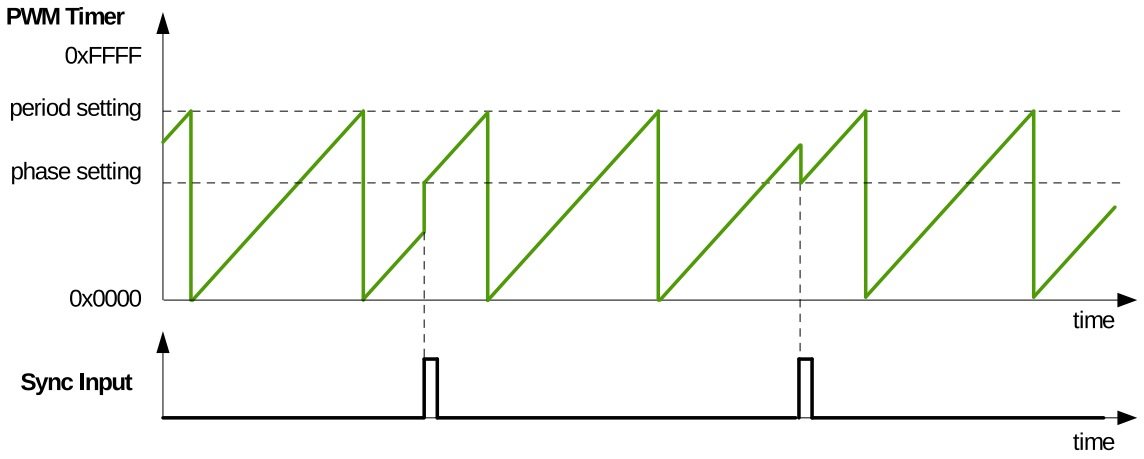


Figure 36.3-7. Count-Up Mode Waveform

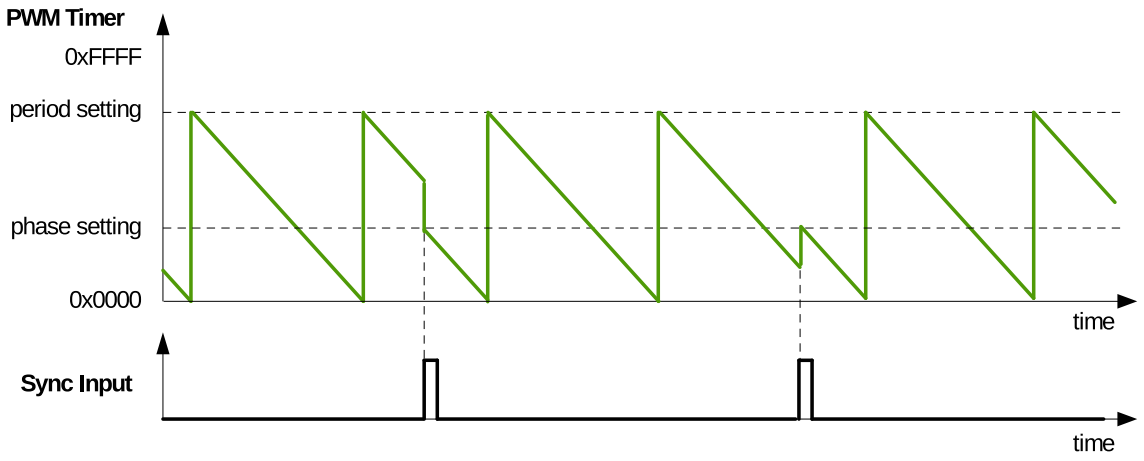


Figure 36.3-8. Count-Down Mode Waveforms

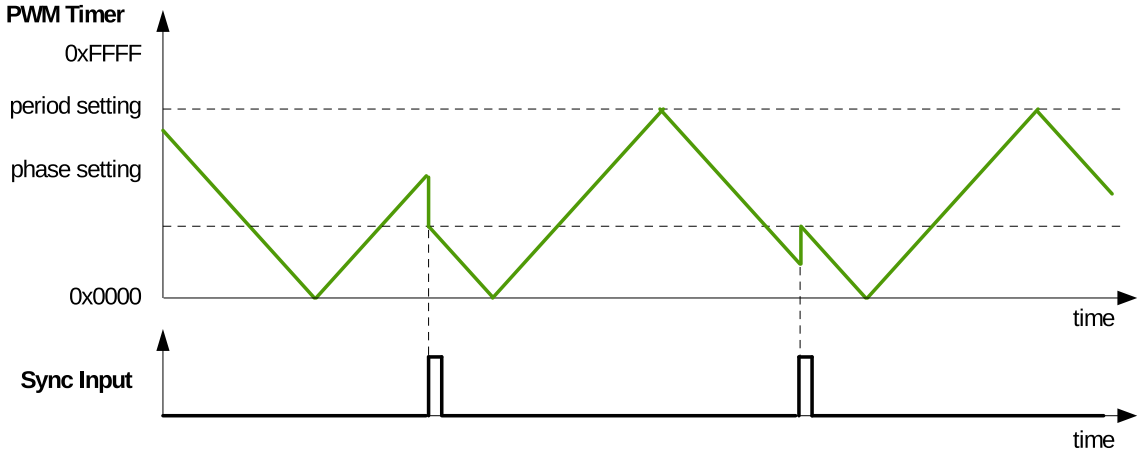


Figure 36.3-9. Count-Up-Down Mode Waveforms, Count-Down at Synchronization Event

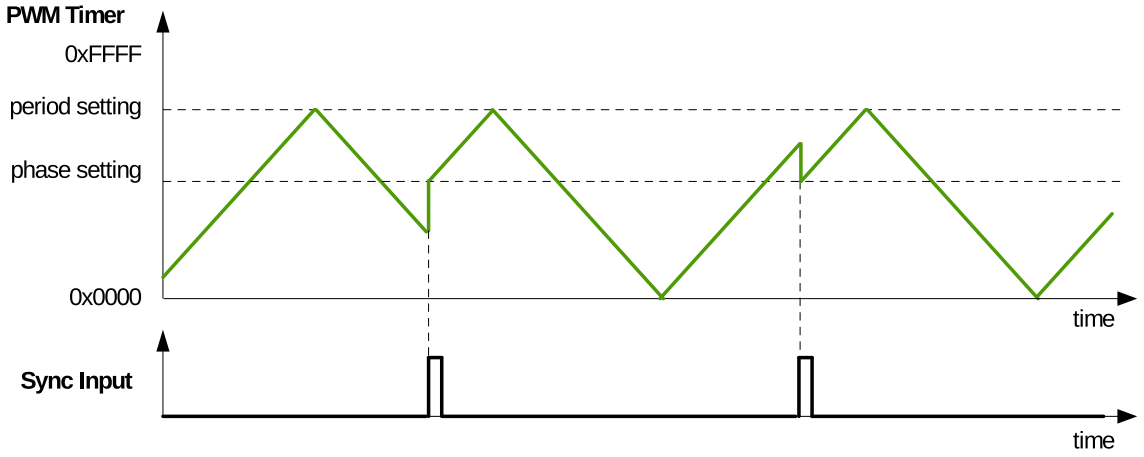


Figure 36.3-10. Count-Up-Down Mode Waveforms, Count-Up at Synchronization Event

When the PWM timer is running, it generates the following timing events periodically and automatically:

- UTEP: The timing event generated when the PWM timer's value is equal to the value of the period field (MCPWM_TIMER*x*_PERIOD) and when the PWM timer is increasing.
- UTEZ: The timing event generated when the PWM timer's value equals zero and when the PWM timer is increasing.
- DTEP: The timing event generated when the PWM timer's value equals the value of the period field (MCPWM_TIMER*x*_PERIOD) and when the PWM timer is decreasing.
- DTEZ: The timing event generated when the PWM timer's value equals zero and when the PWM timer is decreasing.

Figure 36.3-11 to 36.3-13 show the timing waveforms of U/DTEP and U/DTEZ.

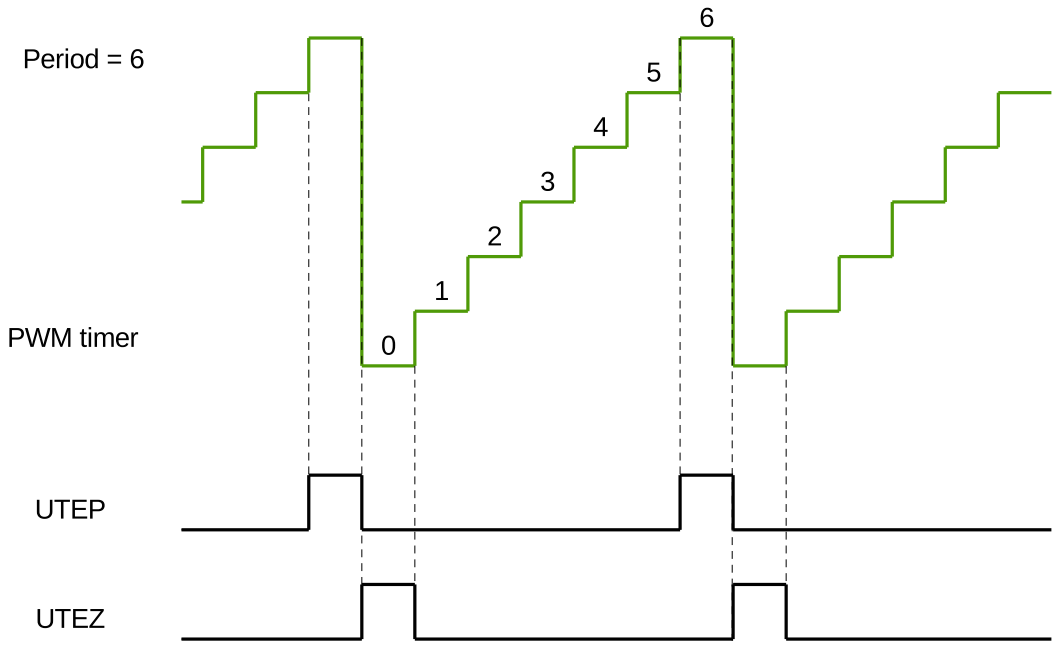


Figure 36.3-11. UTEP and UTEZ Generation in Count-Up Mode

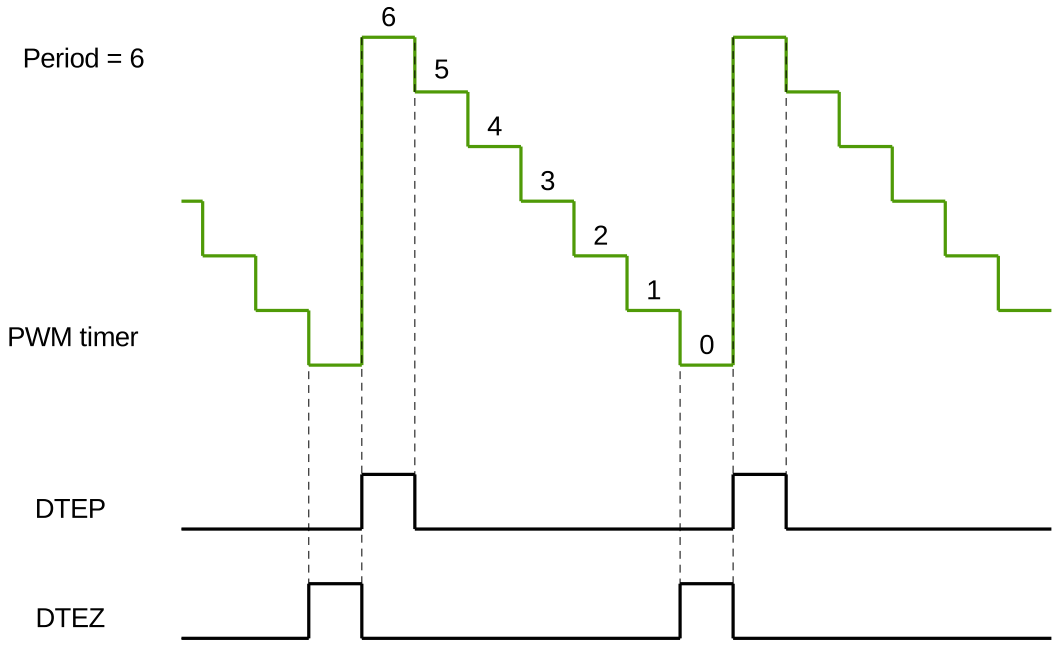


Figure 36.3-12. DTEP and DTEZ Generation in Count-Down Mode

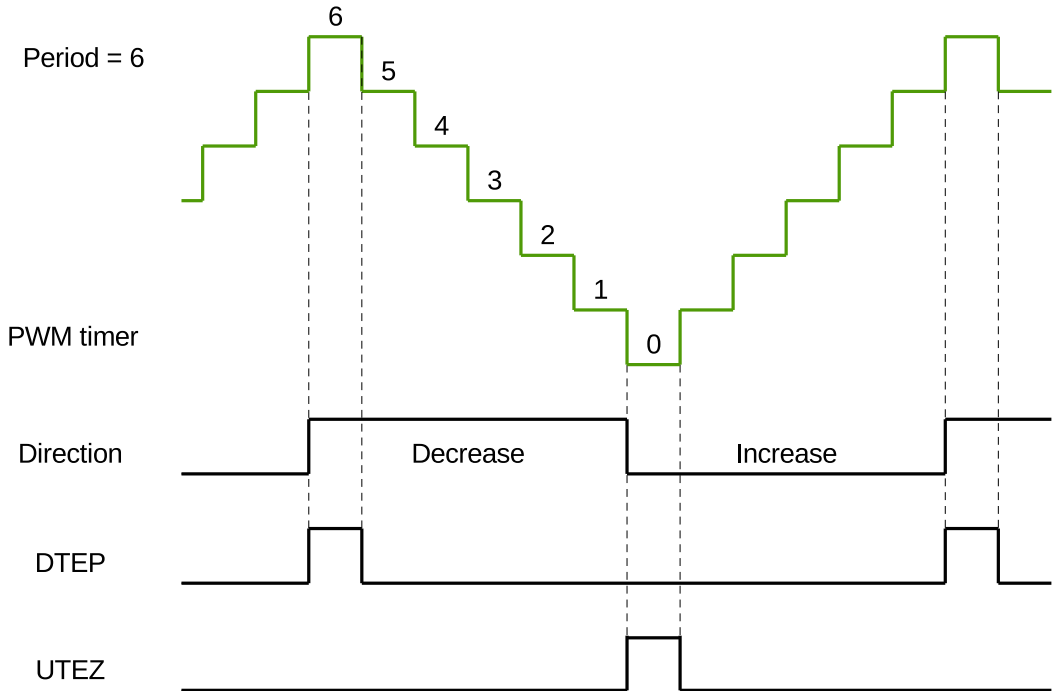


Figure 36.3-13. DTEP and UTEZ Generation in Count-Up-Down Mode

Please note that in the Count-Up-Down Mode, when the counting direction is increasing, the timer range is [0,

period value - 1], and when the counting direction is decreasing, the timer range is [period value, 1]. That is, in this mode, when synchronizing the timer to 0, decreasing counting direction will be illegal, namely, MCPWM_TIMER*n*_PHASE_DIRECTION cannot be set to 1. Similarly, when synchronizing the timer to period value, increasing counting direction will be illegal, namely, MCPWM_TIMER*n*_PHASE_DIRECTION cannot be set to 0. Therefore, when the timer is synchronized to 0, the counting direction can only be increasing, and MCPWM_TIMER*n*_PHASE_DIRECTION will be 0. When the timer is synchronized to the period value, the counting direction can only be decreasing, and MCPWM_TIMER*n*_PHASE_DIRECTION will be 1.

36.3.2.3 Shadow Register of PWM Timer

The PWM timer's period register and the PWM timer's clock prescaler register have shadow registers. The shadow registers can back up the values that are about to be written to the active registers. It also supports writing the values saved into the active register at a specific moment of hardware synchronization. The functionality of both register types is as follows:

- Active Register: Directly responsible for controlling all actions performed by hardware
- Shadow Register: Acts as a temporary buffer for a value to be written to the active register. At a specific, user-configured point in time, the value saved in the shadow register is copied to the active register. Before this happens, the content of the shadow register has no direct effect on the controlled hardware. This helps to prevent erroneous hardware operations, which may happen when a register is asynchronously modified by software. Both the shadow register and the active register have the same memory address. The software always writes into, or reads from the shadow register.

  The moment of updating the clock prescaler's active register is at the time when the timer starts operating. When MCPWM_GLOBAL_UP_EN is set to 1, the moment of updating the active period register can be selected in the following ways:

  - By configuring the update method register MCPWM_TIMER*x*_PERIOD_UPMETHOD to 0, the update will start immediately.
  - By configuring the update method register MCPWM_TIMER*x*_PERIOD_UPMETHOD to 1, the update can start when the PWM timer is equal to zero.
  - By configuring the update method register MCPWM_TIMER*x*_PERIOD_UPMETHOD to 2, the update can start when the PWM timer is synchronized.
  - By configuring the update method register MCPWM_TIMER*x*_PERIOD_UPMETHOD to 3, the update can start when the PWM timer is equal to zero or is synchronized.
  - Software can also trigger a globally forced update bit MCPWM_GLOBAL_FORCE_UP which will prompt all registers in the module to be updated according to shadow registers.

36.3.2.4 PWM Timer Synchronization and Phase Locking

The PWM modules adopt a flexible synchronization method. Each PWM timer has a synchronization input and a synchronization output. The synchronization input can be selected from three synchronization outputs and three synchronization signals from the GPIO matrix. The synchronization output can be generated from the synchronization input signal, when the PWM timer's value is equal to period or zero, or software synchronization. Thus, the PWM timers can be chained together with their phase locked. During synchronization, the PWM timer clock prescaler will reset its counter in order to synchronize the PWM timer clock.

36.3.3 PWM Operator Module

The PWM Operator module has the following functions:

- Generates a PWM signal pair, based on timing references obtained from the corresponding PWM timer
- Each signal out of the PWM signal pair includes a specific pattern of dead time

- Superimposes a carrier on the PWM signal if configured to do so
- Handles response under fault conditions

Figure 36.3-14 shows the block diagram of a PWM operator.

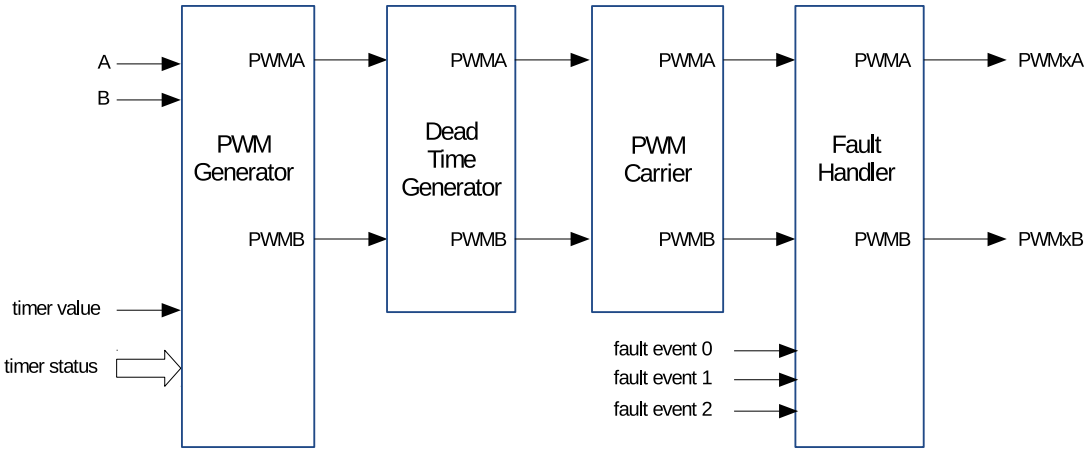


Figure 36.3-14. Block Diagram of A PWM Operator

36.3.3.1 PWM Generator Module

Purpose of the PWM Generator Module

In this module, important timing events are generated or imported. The events are then converted into specific actions to generate the desired waveforms at the PWM*x*A and PWM*x*B outputs.

The PWM generator module performs the following actions:

- Generation of timing events based on time stamps configured using the A and B registers. Events happen when the following conditions are met:
  - UTEA: the PWM timer is counting up and its value is equal to register A.
  - UTEB: the PWM timer is counting up and its value is equal to register B.
  - DTEA: the PWM timer is counting down and its value is equal to register A.
  - DTEB: the PWM timer is counting down and its value is equal to register B.
- Generation of U/DT0, U/DT1 timing events based on fault or synchronization events.
  - UT0: the PWM timer is counting up and FAULT0 detected (field MCPWM_GEN*x*_TO_SEL is set to 0) or FAULT1 detected (field MCPWM_GEN*x*_TO_SEL is set to 1) or FAULT2 detected (field MCPWM_GEN*x*_TO_SEL is set to 2) or synchronized (field MCPWM_GEN*x*_TO_SEL is set to 3).
  - UT1: the PWM timer is counting up and FAULT0 detected (field MCPWM_GEN*x*_T1_SEL is set to 0) or FAULT1 detected (field MCPWM_GEN*x*_T1_SEL is set to 1) or FAULT2 detected (field MCPWM_GEN*x*_T1_SEL is set to 2) or synchronized (field MCPWM_GEN*x*_T1_SEL is set to 3).
  - DT0: the PWM timer is counting down and FAULT0 detected (field MCPWM_GEN*x*_TO_SEL is set to 0) or FAULT1 detected (field MCPWM_GEN*x*_TO_SEL is set to 1) or FAULT2 detected (field MCPWM_GEN*x*_TO_SEL is set to 2) or synchronized (field MCPWM_GEN*x*_TO_SEL is set to 3).
  - DT1: the PWM timer is counting down and FAULT0 detected(field MCPWM_GEN*x*_T1_SEL is set to 0) or FAULT1 detected (field MCPWM_GEN*x*_T1_SEL is set to 1) or FAULT2 detected (field MCPWM_GEN*x*_T1_SEL is set to 2) or synchronized (field MCPWM_GEN*x*_T1_SEL is set to 3).
- Management of priority when these timing events occur concurrently
- Generation of set, clear, and toggle actions, based on the timing events
- Controlling of the PWM duty cycle, depending on the configuration of the PWM generator module
- Handling of new time stamp values, using shadow registers to prevent glitches in the PWM waveform

Shadow Register of PWM Operator

The time stamp registers A and B, as well as action configuration registers MCPWM_GEN*x*_A_REG and MCPWM_GEN*x*_B_REG are shadowed. Shadowing provides a way of updating registers in sync with the hardware.

When MCPWM_GLOBAL_UP_EN is set to 1, the shadow registers can be written to the active register at a specified time. The update method field for MCPWM_GEN*x*_A_REG and MCPWM_GEN*x*_B_REG is MCPWM_GEN*x*_CFG_UPMETHOD. The software can also trigger a globally forced update bit MCPWM_GLOBAL_FORCE_UP which will prompt all registers in the module to be updated according to shadow registers. For a description of the shadow registers, please see Section 36.3.2.3.

Timing Events

For convenience, all timing signals and events are summarized in Table 36.3-2.

Table 36.3-2. Timing Events Used in PWM Generator

Signal	Event Description	PWM Timer Operation
DTEP	PWM timer value is equal to the period register value	PWM timer counts down
DTEZ	PWM timer value is equal to zero	
DTEA	PWM timer value is equal to register A	
DTEB	PWM timer value is equal to register B	
DT0 event	Based on fault or synchronization events	
DT1 event	Based on fault or synchronization events	
UTEP	PWM timer value is equal to the period register value	PWM timer counts up
UTEZ	PWM timer value is equal to zero	
UTEA	PWM timer value is equal to register A	
UTEB	PWM timer value is equal to register B	
UT0 event	Based on fault or synchronization events	
UT1 event	Based on fault or synchronization events	
Software-force event	Software-initiated asynchronous event	N/A

The purpose of a software-force event is to impose non-continuous or continuous changes on the PWM*x*A and PWM*x*B outputs. The change is done asynchronously. Software-force control is handled by the MCPWM_GEN*x*_FORCE_REG register.

The selection and configuration of T0/T1 in the PWM generator module are independent of the configuration of fault events in the fault handler module. A particular trip event may or may not be configured to cause trip action in the fault handler submodule, but the same event can be used by the PWM generator to trigger T0/T1 for controlling PWM waveforms.

It is important to know that when the PWM timer is in count-up-down mode, it will always decrement after a TEP event, and increment after a TEZ event. So, when the PWM timer is in count-up-down mode, DTEP and UTEZ events will occur, while UTEP and DTEZ events will never occur.

The PWM generator can handle multiple events at the same time. Events are prioritized by the hardware and relevant details are provided in Table 36.3-3 and Table 36.3-4. Priority levels range from 1 (the highest) to 7 (the lowest). Please note that the priority of TEP and TEZ events depends on the PWM timer's counting mode.

If the value of A or B is set to be greater than the period, then U/DTEA and U/DTEB will never occur.

Table 36.3-3. Timing Events Priority When PWM Timer Increments

Priority Level	Event
1 (highest)	Software-forced event
2	UTEP
3	UT0
4	UT1
5	UTEB

Priority Level	Event
6	UTEA
7 (lowest)	UTEZ

Table 36.3-4. Timing Events Priority when PWM Timer Decrements

Priority level	Event
1 (highest)	Software-forced event
2	DTEZ
3	DT0
4	DT1
5	DTEB
6	DTEA
7 (lowest)	DTEP

Notes:

1. UTEP and UTEZ do not happen simultaneously. When the PWM timer is in count-up mode, UTEP will always happen one cycle earlier than UTEZ, as demonstrated in Figure 36.3-11, so their action on PWM signals will not interrupt each other. When the PWM timer is in count-up-down mode, UTEP will not occur.
2. DTEP and DTEZ do not happen simultaneously. When the PWM timer is in count-down mode, DTEZ will always happen one cycle earlier than DTEP, as demonstrated in Figure 36.3-12, so their action on PWM signals will not interrupt each other. When the PWM timer is in count-up-down mode, DTEZ will not occur.

PWM Signal Generation

The PWM generator module controls the behavior of outputting PWM*x*A and PWM*x*B when a particular timing event occurs. The timing events are further qualified by the PWM timer's counting mode (increment or decrement). Knowing the counting mode, the module may then perform an independent action at each stage of the PWM timer counting up or down.

The following actions may be configured on PWM*x*A and PWM*x*B outputs:

- Set High: Set the output of PWM*x*A or PWM*x*B to a high level
- Clear Low: Clear the output of PWM*x*A or PWM*x*B by setting it to a low level
- Toggle: Change the current output level of PWM*x*A or PWM*x*B to the opposite value. If it is currently pulled up, then pull it down, or vice versa.
- Do Nothing: Keep both outputs PWM*x*A and PWM*x*B unchanged. In this state, interrupts can still be triggered.

Actions on outputs are configured by using registers MCPWM_GEN*x*_A_REG and MCPWM_GEN*x*_B_REG. So, the action to be taken on each output is set independently. Also, there is great flexibility in selecting actions to be taken on a given output based on events. More specifically, any event listed in Table 36.3-2 can operate on either output of PWM*x*A or PWM*x*B. To check out registers for particular generators 0, 1, or 2, please refer to register descriptions in Section 36.4.

Waveforms for Common Configurations

Figure 36.3-15 presents the symmetric PWM waveform generated when the PWM timer is in Count-Up-Down mode. DC 0%–100% modulation can be calculated via the formula below:

$$Duty = (Period - A) \div Period$$

If A matches the PWM timer value and the PWM timer is incrementing, then the PWM output is pulled up. If A matches the PWM timer value while the PWM timer is decrementing, then the PWM output is pulled low.

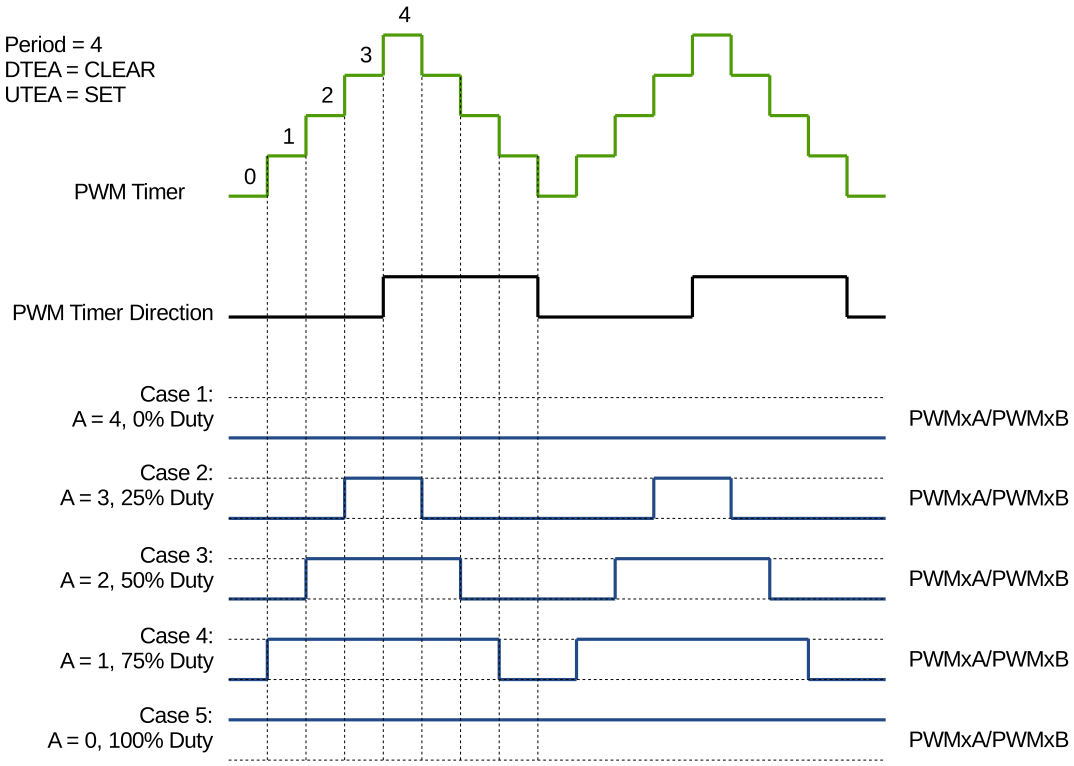


Figure 36.3-15. Symmetrical Waveform in Count-Up-Down Mode

The PWM waveforms in Figure 36.3-16 to 36.3-19 show some common PWM operator configurations. The following conventions are used in the figures:

- Period A and B refer to the values written in the corresponding registers.
- PWM*x*A and PWM*x*B are the output signals of PWM Operator *x*.

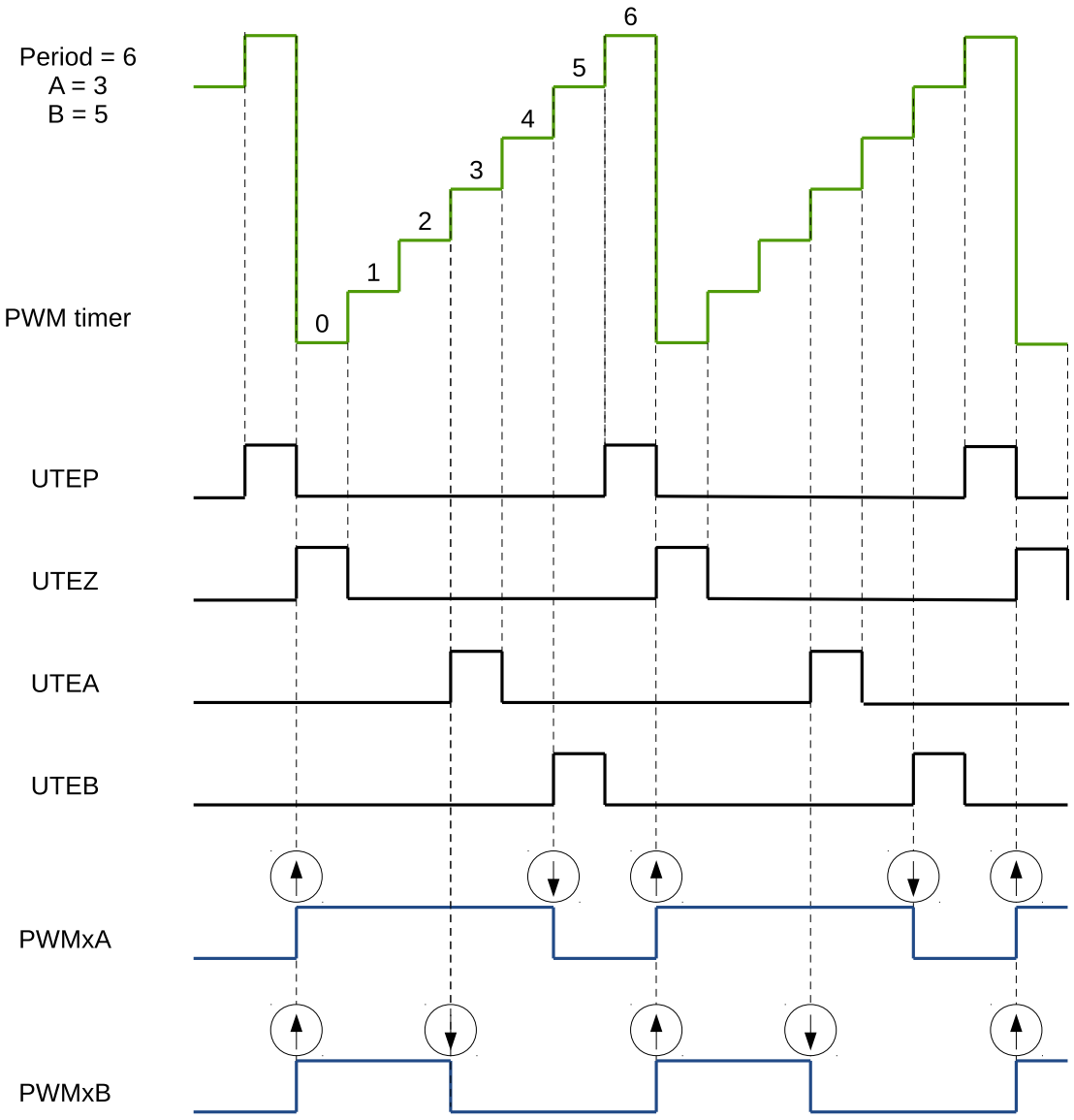


Figure 36.3-16. Count-Up, Single Edge Asymmetric Waveform, with Independent Modulation on PWM*x*A and PWM*x*B — Active High

The duty modulation for PWM*x*A is set by B, active high and proportional to B.
The duty modulation for PWM*x*B is set by A, active high and proportional to A.

$$Period = (MCPWM\_TIMERx\_PERIOD + 1) \times T_{PT\_CLK}$$

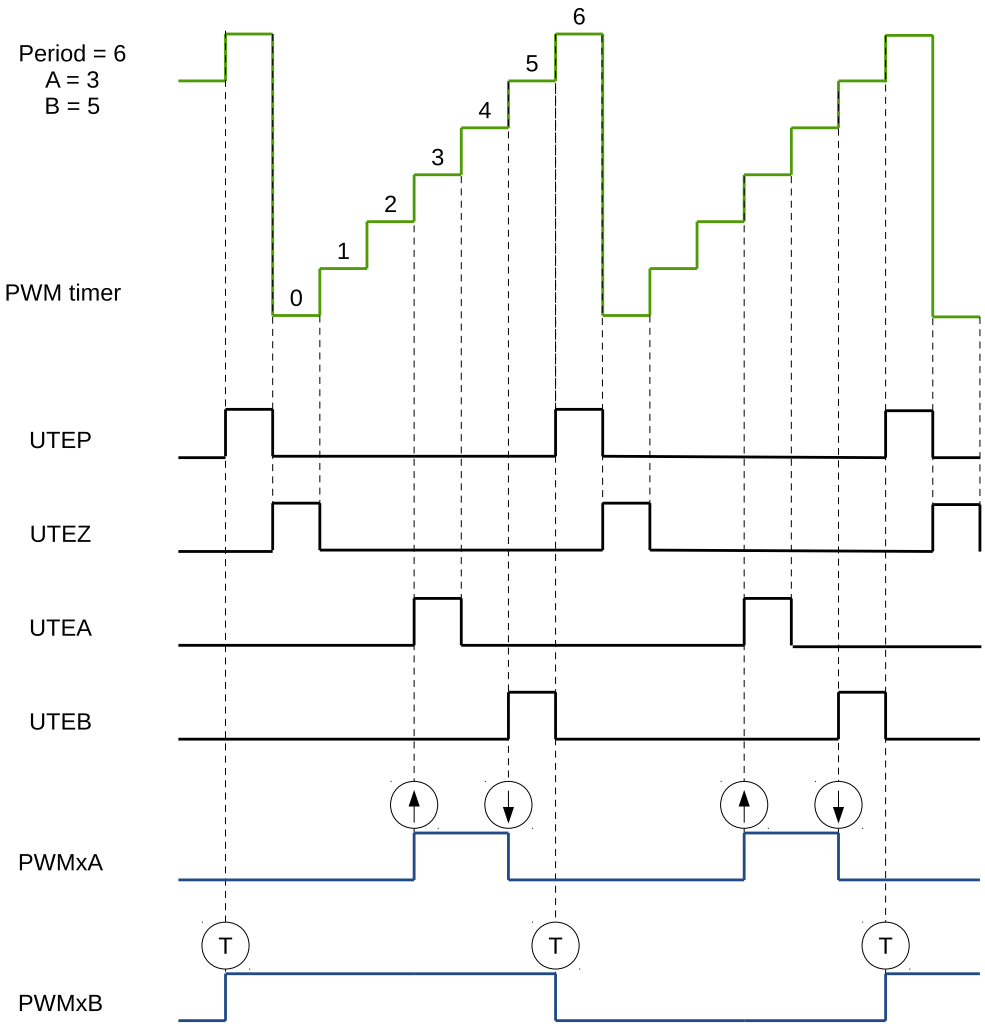


Figure 36.3-17. Count-Up, Pulse Placement Asymmetric Waveform with Independent Modulation on PWM*x*A

Pulses may be generated anywhere within one PWM cycle (zero to period).
PWM*x*A's high-time duty is proportional to (B – A).

$$Period = (MCPWM\_TIMERx\_PERIOD + 1) \times T_{PT\_CLK}$$

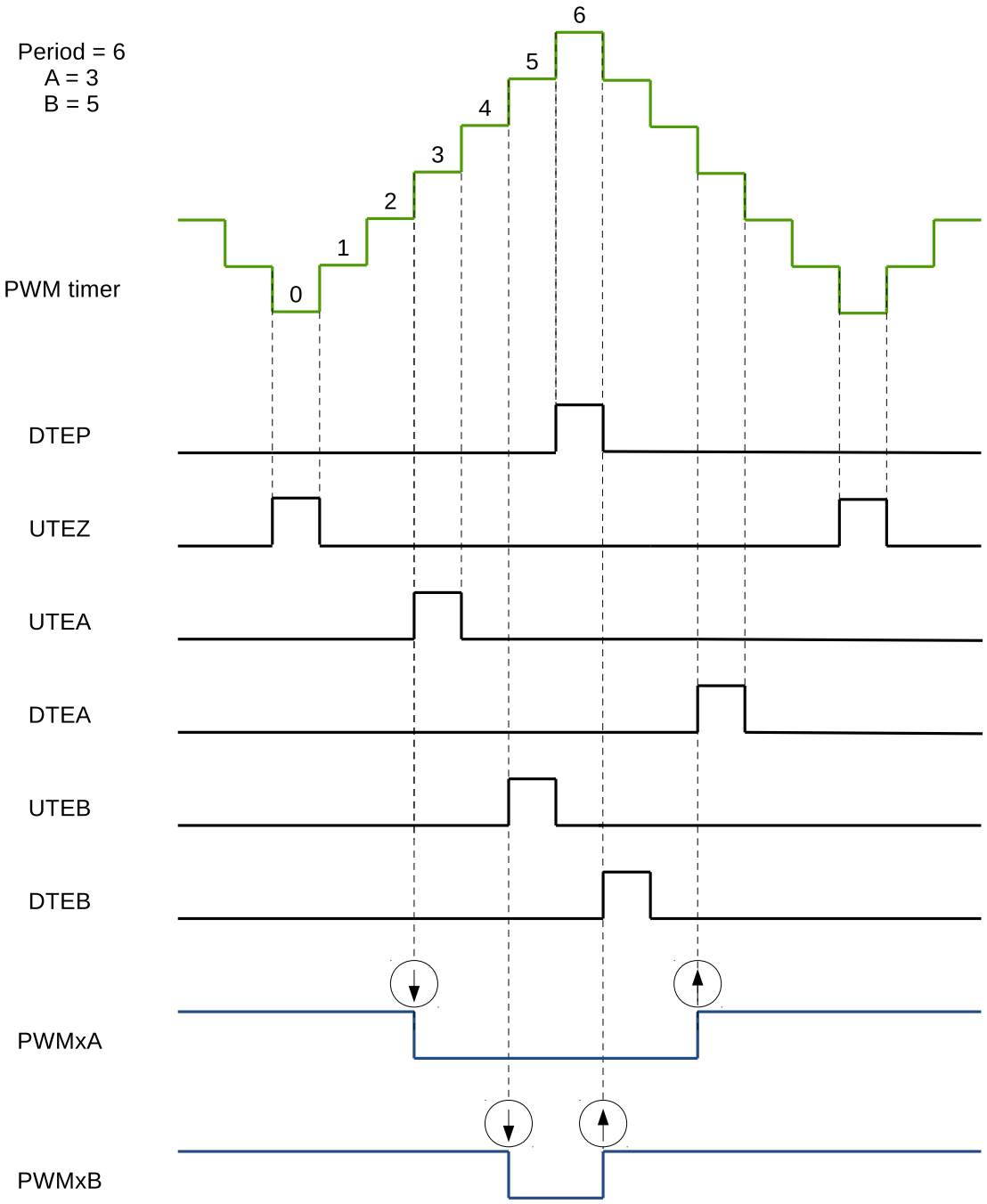


Figure 36.3-18. Count-Up-Down, Dual Edge Symmetric Waveform, with Independent Modulation on PWM*x*A and PWM*x*B — Active High

The duty modulation for PWM*x*A is set by A, active high and proportional to A.
The duty modulation for PWM*x*B is set by B, active high and proportional to B.
Outputting PWM*x*A and PWM*x*B can drive separate switches.

$$Period = (2 \times MCPWM\_TIMERx\_PERIOD) \times T_{PT\_CLK}$$

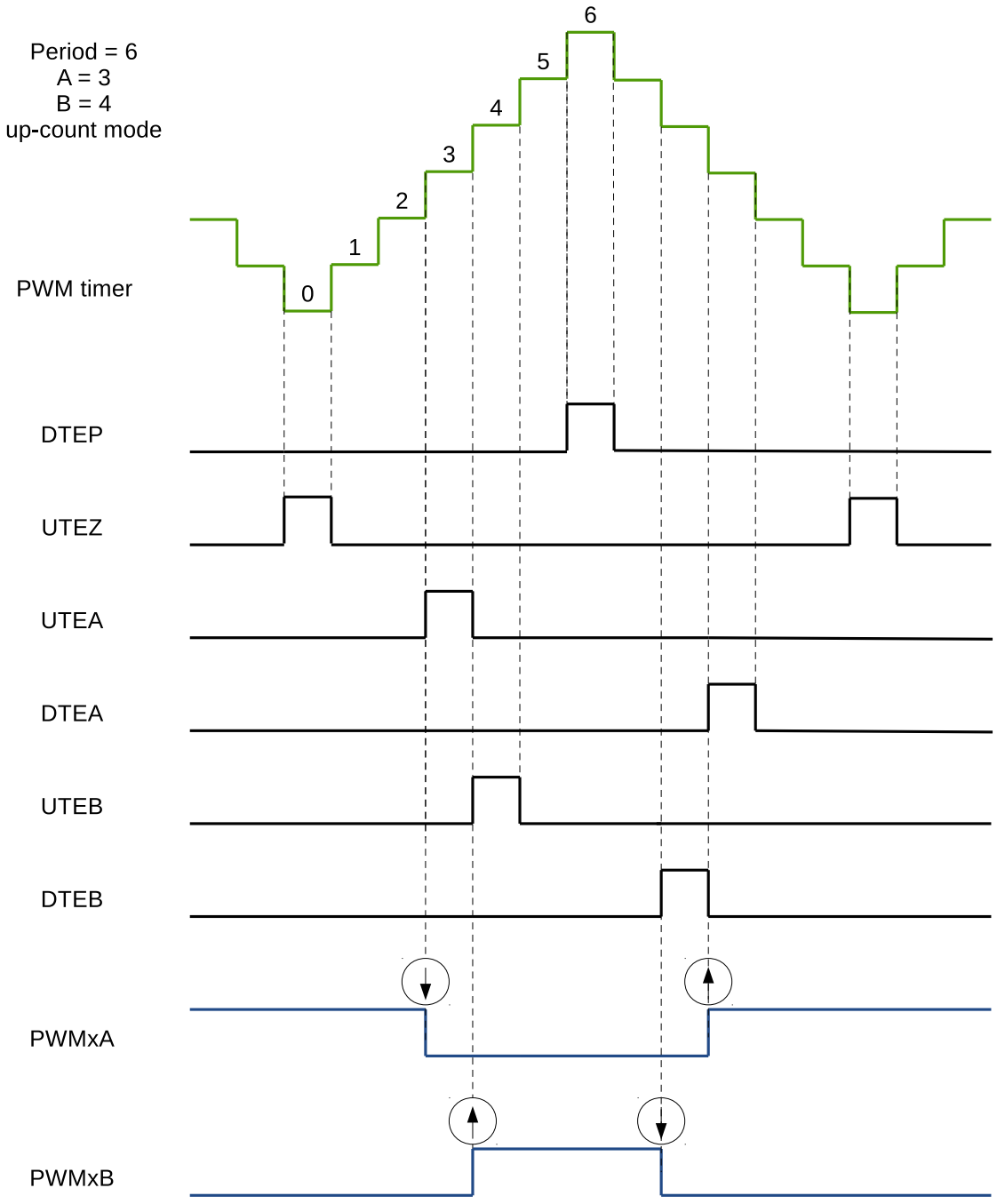


Figure 36.3-19. Count-Up-Down, Dual Edge Symmetric Waveform, with Independent Modulation on PWM*x*A and PWM*x*B — Complementary

The duty modulation of PWM*x*A is set by A, is active high and proportional to A.
The duty modulation of PWM*x*B is set by B, is active low and proportional to B.
Outputs PWM*x* can drive upper/lower (complementary) switches.
Dead time = B − A. Edge placement is configurable by software. The dead time generator module supports configuring edge delay methods when required.

$$Period = (2 \times MCPWM\_TIMERx\_PERIOD) \times T_{PT\_CLK}$$

Figure 36.3-20 shows a waveform when UT0/1 and DT0/1 events are generated. In this example, T0 selects

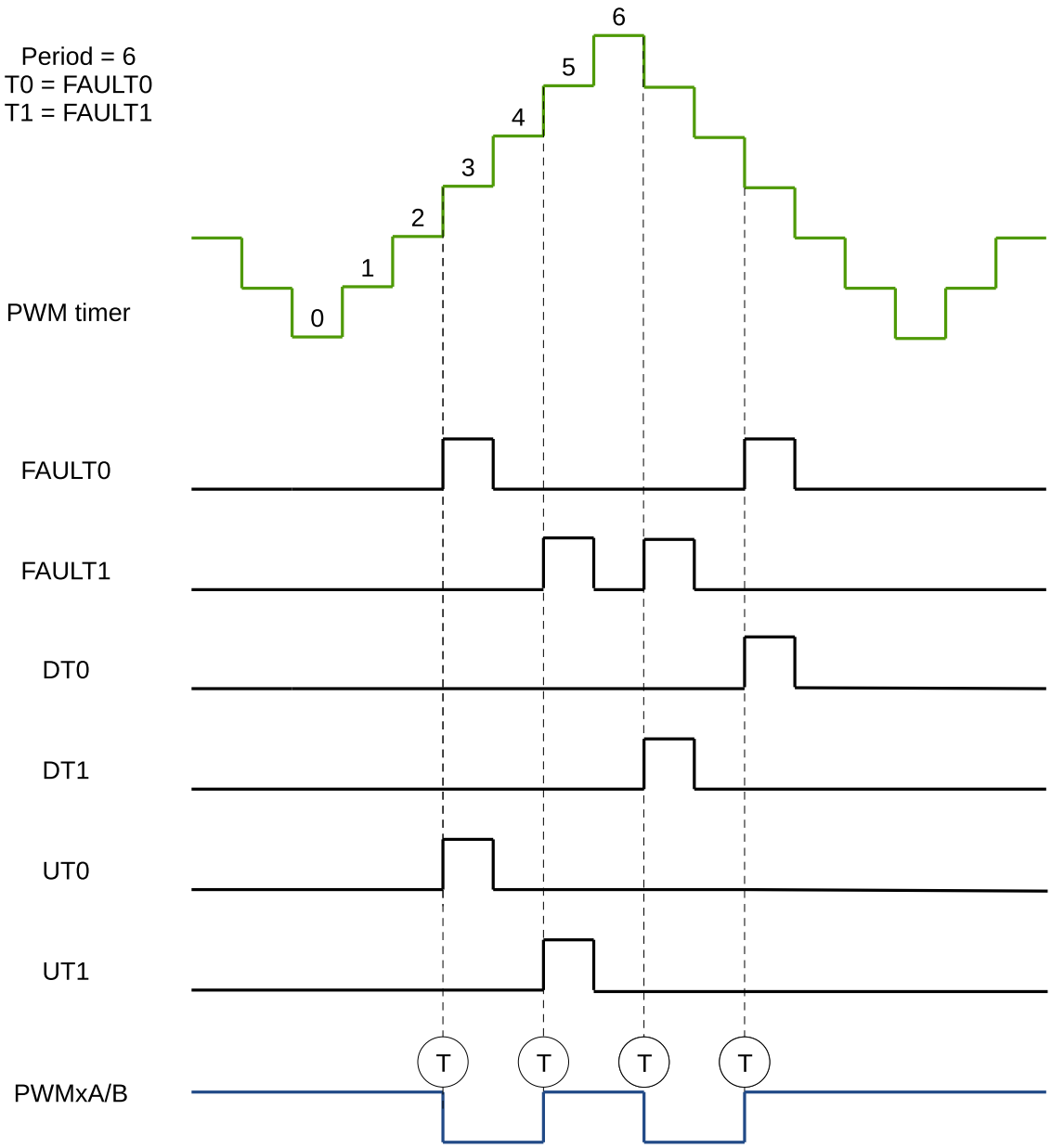


Figure 36.3-20. Count-Up-Down, Fault or Synchronization Events, with Same Modulation on PWM*x*A and PWM*x*B

FAULT0 and T1 selects FAULT1. The events selected by T0 and T1 can be configured independently, and these events can be FAULT0, FAULT1, FAULT2 or synchronous events. For detailed configuration, see section 36.3.3.1.

Software-Force Events

There are two types of software-force events inside the PWM generator:

- Non-continuous-immediate (NCI) software-force events: Such types of events are immediately effective on PWM outputs when triggered by software. The forcing is non-continuous, which means the next active timing event will be able to alter the PWM outputs.
- Continuous (CNTU) software-force events: Such types of events are continuous. The forced PWM outputs will continue until they are released by software. The events' triggers are configurable. They can

be configured to be timing events or immediate events.

Figure 36.3-21 shows a waveform of NCI software-force events. NCI events are used to force PWM*x*A output low. Forcing on PWM*x*B is disabled in this case.

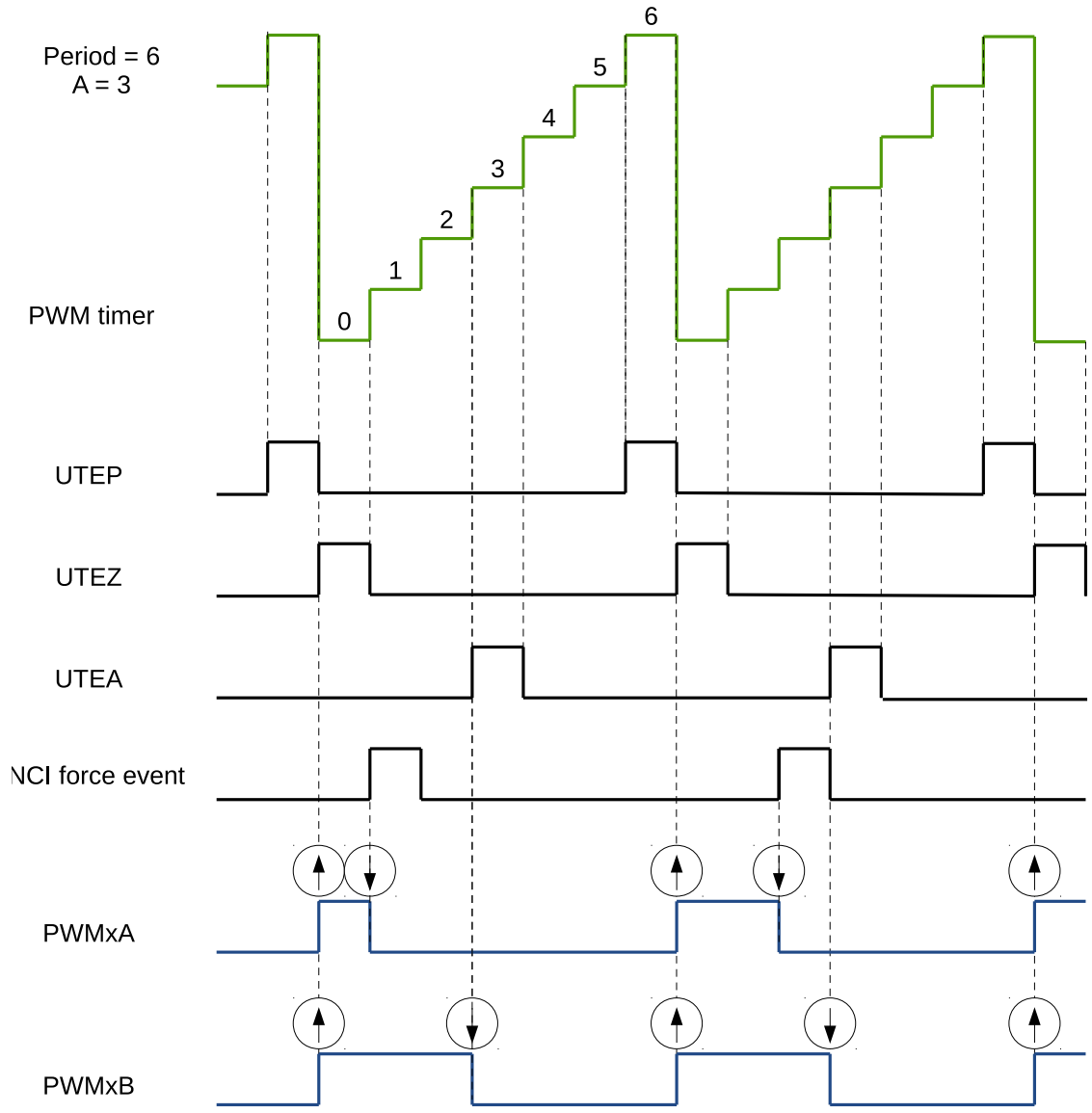


Figure 36.3-21. Example of an NCI Software-Force Event on PWM*x*A

Figure 36.3-22 shows a waveform of CNTU software-force events. UTEZ events are selected as triggers for CNTU software-force events. CNTU is used to force the PWM*x*B output low. Forcing on PWM*x*A is disabled.

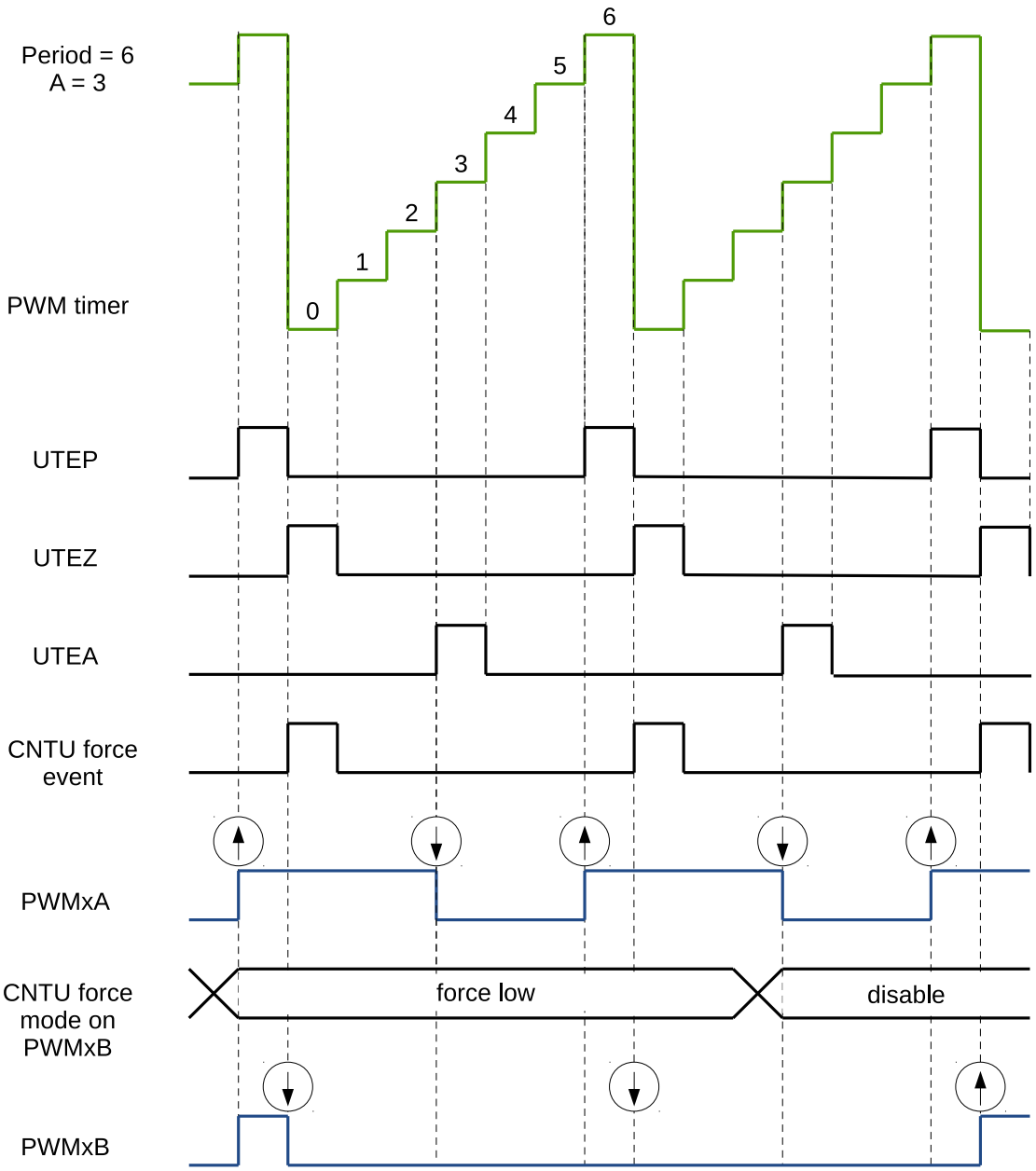


Figure 36.3-22. Example of a CNTU Software-Force Event on PWM*x*B

36.3.3.2 Dead Time Generator Module

Purpose of the Dead Time Generator Module

Section 36.3.3.1 introduced several options to generate signals on PWM*x*A and PWM*x*B outputs, with a specific placement of signal edges. The required dead time is obtained by altering the edge placement between signals and by setting the signal's duty cycle. Another option to control the dead time is to use a specialized module – Dead Time Generator.

The key functions of the Dead Time Generator module are as follows:

- Generating signal pairs (PWM*x*A and PWM*x*B) with a dead time from a single PWM*x*A input
- Creating a dead time by adding delays to signal edges:
  - Rising edge delays (RED)
  - Falling edge delays (FED)
- Configuring the signal pairs to be:
  - Active high complementary (AHC)
  - Active low complementary (ALC)
  - Active high (AH)
  - Active low (AL)
- This module may also be bypassed, if the dead time is configured directly in the generator module.

Shadow Register of Dead Time Generator

Delay registers RED and FED are shadowed with registers MCPWM_DT*x*_RED_CFG_REG and MCPWM_DT*x*_FED_CFG_REG. When MCPWM_GLOBAL_UP_EN is set to 1, the values saved in the shadow registers can be written to the active register at a specified time. The update method register for MCPWM_DT*x*_RED_CFG_REG is MCPWM_DT*x*_RED_UPMETHOD. The update method register for MCPWM_DT*x*_FED_CFG_REG is MCPWM_DT*x*_FED_UPMETHOD. The Software can also trigger a globally forced update bit MCPWM_GLOBAL_FORCE_UP which will prompt all registers in the module to be updated according to shadow registers. For the description of shadow registers, please see section 36.3.2.3.

Highlights for Operation of the Dead Time Generator

Options for setting up the dead time module are shown in Figure 36.3-23.

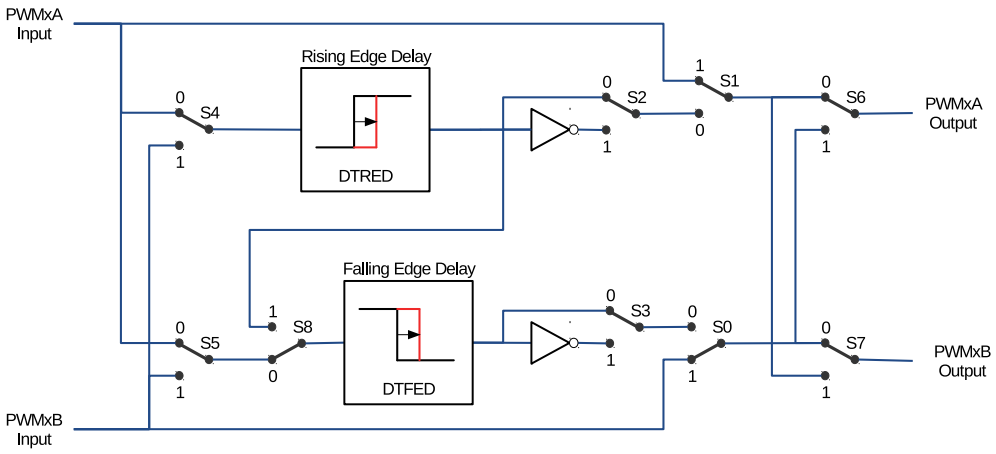


Figure 36.3-23. Options for Setting up the Dead Time Generator Module

S0-S8 in the figure above are switches controlled by fields in register MCPWM_DT*x*_CFG_REG shown in Table 36.3-5.

Table 36.3-5. Dead Time Generator Switches Control Fields

Switch	Field
S0	MCPWM_DT*x*_B_OUTBYPASS
S1	MCPWM_DT*x*_A_OUTBYPASS
S2	MCPWM_DT*x*_RED_OUTINVERT
S3	MCPWM_DT*x*_FED_OUTINVERT
S4	MCPWM_DT*x*_RED_INSEL
S5	MCPWM_DT*x*_FED_INSEL
S6	MCPWM_DT*x*_A_OUTSWAP
S7	MCPWM_DT*x*_B_OUTSWAP
S8	MCPWM_DT*x*_DEB_MODE

All switch combinations are supported, but not all of them represent the typical modes of use. Table 36.3-6 documents some typical dead time configurations. In these configurations, the position of S4 and S5 sets PWM*x*A as the common source of both falling edge delays (FED) and rising edge delays (RED). The modes presented in table 36.3-6 may be categorized as follows:

Table 36.3-6. Typical Dead Time Generator Operating Modes

Mode	Mode Description	S0	S1	S2	S3
1	PWM*x*A and PWM*x*B Pass Through/No Delay	1	1	X	X
2	Active High Complementary (AHC), see Figure 36.3-24	0	0	0	1
3	Active Low Complementary (ALC), see Figure 36.3-25	0	0	1	0
4	Active High (AH), see Figure 36.3-26	0	0	0	0
5	Active Low (AL), see Figure 36.3-27	0	0	1	1

Mode	Mode Description	S0	S1	S2	S3
6	PWM*x*A Output = PWM*x*A In (No Delay) PWM*x*B Output = PWM*x*A Input with Falling Edge Delay	0	1	0 or 1	0 or 1
7	PWM*x*A Output = PWM*x*A Input with Rising Edge Delay PWM*x*B Output = PWM*x*B Input with No Delay	1	0	0 or 1	0 or 1

Note:
For all the modes above, the position of the binary switches S4 to S8 is set to 0.

- **Mode 1: Bypass delays on both FED and RED**
  In this mode, the dead time module is disabled. Signals of PWM*x*A and PWM*x*B pass through without any modifications.
- **Mode 2-5: Classical Dead Time Polarity Settings**
  These four modes represent typical configurations of polarity and should cover the active-high/low modes in available industry power switch gate drivers. The typical waveforms are shown in Figure 36.3-24 to 36.3-27.
- **Mode 6 and 7: Bypass delays on falling edges (FED) or rising edges (RED)**
  In these two modes, either RED or FED is bypassed. As a result, the corresponding delay is not applied.

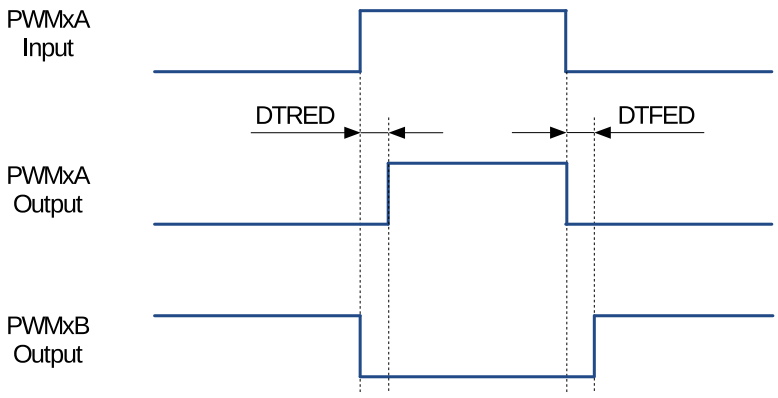


Figure 36.3-24. Active High Complementary (AHC) Dead Time Waveforms

RED and FED delays may be set up independently. The delay value is programmed using the 16-bit field MCPWM_DT*x*_RED and MCPWM_DT*x*_FED. The field value represents the number of clock (DT_CLK) periods by which a signal edge is delayed. DT_CLK can be selected from PWM_CLK or PT_CLK through the MCPWM_DT*x*_CLK_SEL bit.

To calculate the delays on the falling edge (FED) and rising edge (RED), use the following formulas:

$$FED = MCPWM\_DTx\_FED \times T_{DT\_CLK}$$

$$RED = MCPWM\_DTx\_RED \times T_{DT\_CLK}$$

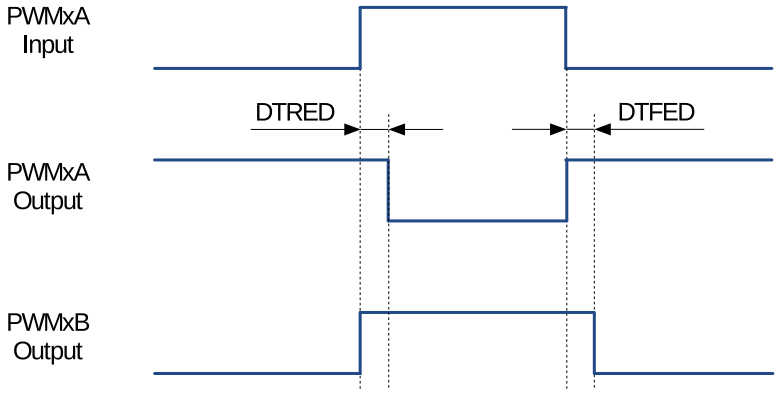


Figure 36.3-25. Active Low Complementary (ALC) Dead Time Waveforms

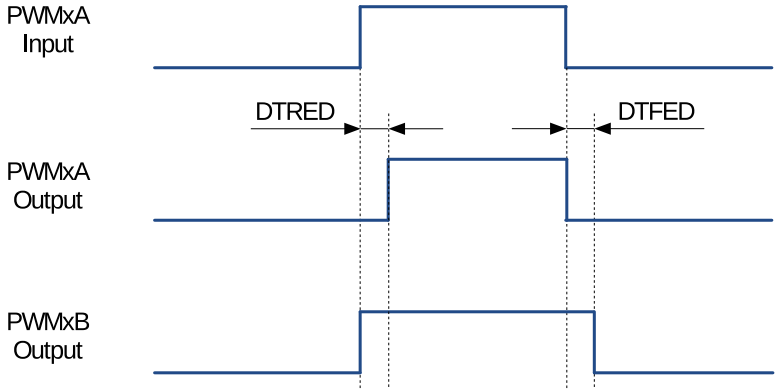


Figure 36.3-26. Active High (AH) Dead Time Waveforms

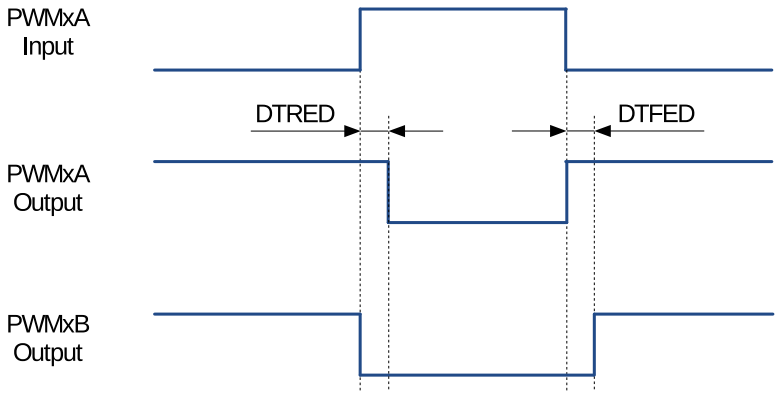


Figure 36.3-27. Active Low (AL) Dead Time Waveforms

36.3.3.3 PWM Carrier Module

The coupling of PWM output to a motor driver may need isolation with a transformer. Transformers deliver only AC signals, while the duty cycle of a PWM signal may range anywhere from 0% to 100%. The PWM carrier module passes such a PWM signal through a transformer by using a high-frequency carrier to modulate the signal.

Function Overview

The following key characteristics of this module are configurable:

- Carrier frequency
- Pulse width of the first pulse
- Duty cycle of the second and the subsequent pulses
- Enabling/disabling the carrier function

Operational Highlights

The PWM carrier clock (PC_CLK) is derived from PWM_CLK. The frequency and duty cycle are configured by the MCPWM_CARRIER*x*_PRESCALE and MCPWM_CARRIER*x*_DUTY bits in the MCPWM_CARRIER*x*_CFG_REG register. The purpose of one-shot pulses is to provide high-energy impulse to reliably turn on the power switch. Subsequent pulses sustain the power-on status. The width of a one-shot pulse is configurable with the MCPWM_CARRIER*x*_OSHTWTH field. Enabling/disabling of the carrier module is done with the MCPWM_CARRIER*x*_EN bit.

Waveform Examples

Figure 36.3-28 shows an example of waveforms, where a carrier is superimposed on original PWM pulses. This figure does not show the first one-shot pulse and the duty-cycle control. Related details are covered in the following two sections.

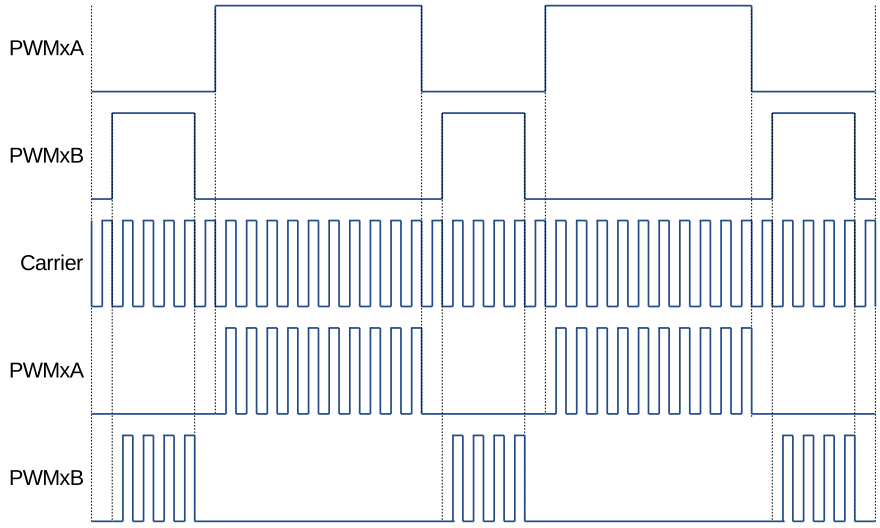


Figure 36.3-28. Example of Waveforms Showing PWM Carrier Action

One-Shot Pulse

The width of the first pulse can be configured to 16 different values, which can be calculated by the following equation:

$$T_{1stpulse} = T_{PWM\_CLK} \times 8 \times (MCPWM\_CARRIERx\_PRESCALE + 1) \times (MCPWM\_CARRIERx\_OSHTWTH + 1)$$

Where:

- T_{PMW_CLK} is the period of the PWM clock (PWM_CLK).
- $(MCPWM\_CARRIERx\_OSHTWTH + 1)$ is the width of the first pulse (whose value ranges from 1 to 16).
- $(MCPWM\_CARRIERx\_PRESCALE + 1)$ is the PWM carrier clock's (PC_CLK) prescaler value.

The first one-shot pulse and subsequent sustaining pulses are shown in Figure 36.3-29.

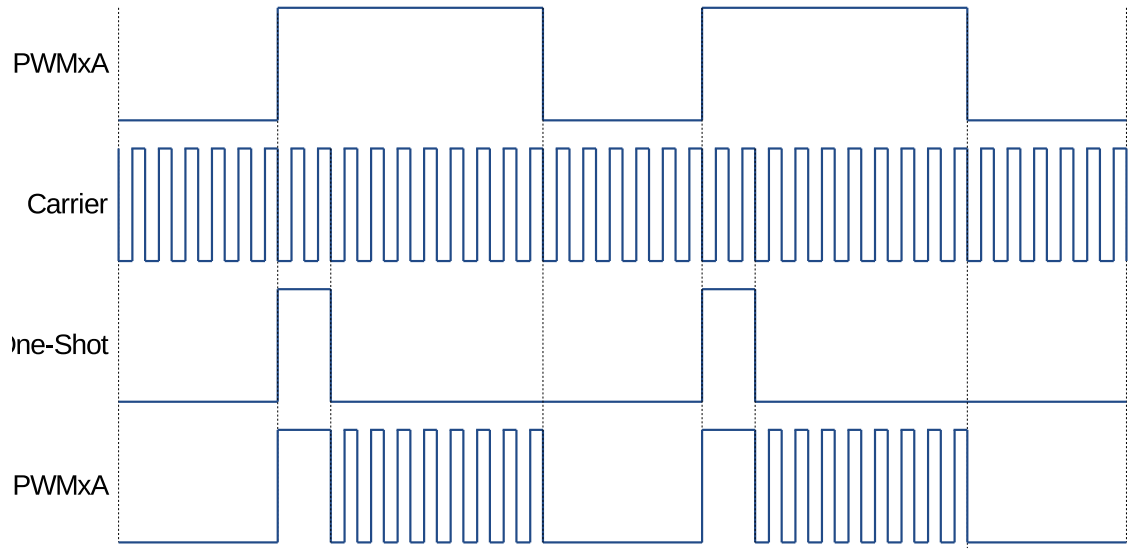


Figure 36.3-29. Example of the First Pulse and the Subsequent Sustaining Pulses of the PWM Carrier Submodule

Duty Cycle Control

After issuing the first one-shot pulse, the remaining PWM pulses are modulated according to the carrier frequency. Users can configure the duty cycle of this signal. Tuning of duty may be required, so that the signal passes through the isolating transformer and can still operate (turn on/off) the motor drive, changing rotation speed and direction.

The duty cycle may be set to one of seven values, using MCPWM_CARRIERx_DUTY, or bits [7:5] of register MCPWM_CARRIERx_CFG_REG.

Below is the formula for calculating the duty cycle:

$$Duty = MCPWM\_CARRIERx\_DUTY \div 8$$

All seven settings of the duty cycle are shown in Figure 36.3-30.

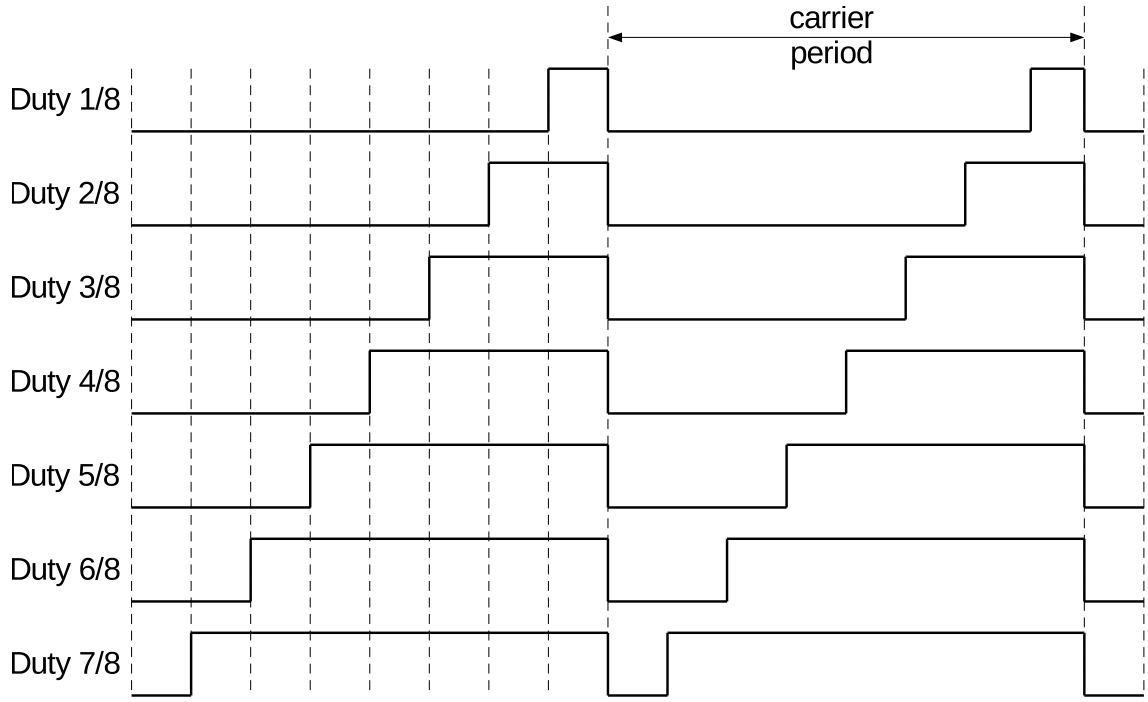


Figure 36.3-30. Possible Duty Cycle Settings for Sustaining Pulses in the PWM Carrier Submodule

36.3.3.4 Fault Detection Module

Each MCPWM peripheral is connected to three fault signals (FAULT0, FAULT1, and FAULT2) which are sourced from the GPIO matrix. These signals are intended to indicate external fault conditions, and may be preprocessed by the Fault Detection module to generate fault events. Fault events can then execute the user code to control MCPWM outputs in response to specific faults.

Function of Fault Detection Module

The key actions performed by the fault detection module are:

- Forcing outputs PWM*x*A and PWM*x*B, upon detected fault, to one of the following states:
  - High
  - Low
  - Toggle
  - No action taken
- Execution of one-shot trip (OST) upon detection of over-current conditions/short circuits
- Cycle-by-cycle trip (CBC) to provide current-limiting operation
- Allocation of either one-shot or cycle-by-cycle operation for each fault signal
- Generation of interrupts for each fault input
- Support for software-force tripping
- Enabling or disabling of module function as required

Operation and Configuration Tips

This section provides the operational tips and set-up options for the Fault Detection module.

Fault signals coming from pins are sampled and synced in the GPIO matrix. In order to guarantee the successful sampling of fault pulses, each pulse duration must be at least two APB clock cycles. The Fault Detection module will then sample fault signals by using PWM_CLK. So, the duration of fault pulses coming from the GPIO matrix must be at least one PWM_CLK cycle. Differently put, regardless of the period relation between the APB clock and PWM_CLK, the width of fault signal pulses on pins must be at least equal to the sum of two APB clock cycles and one PWM_CLK cycle.

Each level of fault signals, FAULT0 to FAULT2, can be used by the Fault Detection module to generate fault events (fault_event0 to fault_event2). Every fault event can be configured individually to provide CBC action, OST action, or none.

- **Cycle-by-Cycle (CBC) action:**
  When CBC action is triggered, the state of PWM*x*A and PWM*x*B will be changed immediately according to the configuration of fields MCPWM_FH*x*_A_CBC_U/D and MCPWM_FH*x*_B_CBC_U/D. Different actions can be indicated when the PWM timer is incrementing or decrementing. Different CBC action interrupts can be triggered for different fault events. Status field MCPWM_FH*x*_CBC_ON indicates whether a CBC action is on or off. When the fault event is no longer present, CBC actions on PWM*x*A/B will be cleared at a specified point, which is either a D/UTEP or D/UTEZ event. Field MCPWM_FH*x*_CBCPULSE determines at which event PWM*x*A and PWM*x*B will be able to resume normal actions. Therefore, in this mode, the CBC action is cleared or refreshed upon every PWM cycle.
- **One-Shot (OST) action:**
  When OST action is triggered, the state of PWM*x*A and PWM*x*B will be changed immediately, depending on the setting of fields MCPWM_FH*x*_A_OST_U/D and MCPWM_FH*x*_B_OST_U/D. Different actions can be configured when the PWM timer is incrementing or decrementing. Different OST action interrupts can be triggered from different fault events. Status field MCPWM_FH*x*_OST_ON indicates whether an OST action is on or off. The OST actions on PWM*x*A/B are not automatically cleared when the fault event is no longer present. They must be cleared manually by setting the MCPWM_FH*x*_CLR_OST bit.

36.3.4 Capture Module

36.3.4.1 Introduction

The capture module contains three complete capture channels. Channel inputs CAP0, CAP1, and CAP2 are sourced from the GPIO matrix. Thanks to the flexibility of the GPIO matrix, CAP0, CAP1, and CAP2 can be configured from any pin input. Multiple capture channels can be sourced from the same pin input, while prescaling for each channel can be set differently. Also, capture channels are sourced from different pins. This provides several options for handling capture signals by hardware in the background, instead of having them processed directly by the CPU. A capture module has the following independent key resources:

- One 32-bit timer (counter) which can be synchronized with the PWM timer, another module, or software
- Three capture channels, each equipped with a 32-bit time stamp and a capture prescaler
- Independent edge polarity (rising/falling edge) selection for any capture channel
- Input capture signal prescaling (from 1 to 256)

- Interrupt capabilities on any of the three capture events

36.3.4.2 Capture Timer

The capture timer is a 32-bit counter incrementing continuously. It is enabled by setting MCPWM_CAP_TIMER_EN to 1. Its operating clock source is the MCPWM core clock. When MCPWM_CAP_SYNCI_EN is configured, the counter will be loaded with phase stored in register MCPWM_CAP_TIMER_PHASE_REG at the time of a sync event. Sync events can select from PWM timers sync-out, or PWM module sync-in by configuring MCPWM_CAP_SYNCI_SEL. Sync events can also be generated by setting MCPWM_CAP_SYNC_SW. The capture timer provides timing references for all three capture channels.

36.3.4.3 Capture Channel

The capture signal coming to a capture channel will be inverted first, if needed, and then prescaled. Each capture channel has a prescaler register of MCPWM_CAP*x*_PRESCALE. Finally, specified edges of preprocessed capture signal will trigger capture events. Setting MCPWM_CAP*x*_EN to enable a capture channel. The capture event occurs at the time selected by the MCPWM_CAP*x*_MODE. When a capture event occurs, the capture timer's value is stored in time-stamp register MCPWM_CAP_CH*x*_REG. Different interrupts can be generated for different capture channels at capture events. The edge that triggers a capture event is recorded in register MCPWM_CAP*x*_EDGE. The capture event can be also forced by software setting MCPWM_CAP*x*_SW.

36.3.5 ETM Module

36.3.5.1 Overview

The MCPWM peripheral on ESP32-H2 supports the Event Task Matrix (ETM) function, which allows MCPWM's ETM tasks to be triggered by any peripherals' ETM events, or MCPWM's ETM events to trigger any peripherals' ETM tasks. The capture module, the fault detection module, three timers, and three operators can generate events and respond to tasks independently. This section introduces the ETM tasks and events related to MCPWM. For more information, please refer to Chapter 10 Event Task Matrix (SOC_ETM).

36.3.5.2 MCPWM-Related ETM Events

When setting the enable field to 1, after the generation condition is met, the corresponding event would be generated. For details, please refer to Table 36.3-7 below:

Table 36.3-7. MCPWM-Related ETM Events

Enable Field	Generation Condition	Event Generated
MCPWM_EVT_CAP*x*_EN	CAP*x* capture event occurs	MCPWM_EVT_CAP*x*
MCPWM_EVT_TZ*x*_OST_EN	PWM operator *x* performs a One-Shot trip (OST) operation	MCPWM_EVT_TZ*x*_OST
MCPWM_EVT_TZ*x*_CBC_EN	PWM operator *x* performs a cycle-by-cycle trip (CBC) operation.	MCPWM_EVT_TZ*x*_CBC
MCPWM_EVT_F*x*_CLR_EN	Fault event fault_event*x* is cleared	MCPWM_EVT_F*x*_CLR

[1] See Section 36.3.3.1 for a detailed description of timer stamp A and B.

Enable Field	Generation Condition	Event Generated
MCPWM_EVT_F*x*_EN	Fault event fault_event*x* is generated	MCPWM_EVT_F*x*
MCPWM_EVT_OP*x*_TEB_EN	The count value of the timer that PWM operator *x* selects is equal to the value of timer stamp B[1]	MCPWM_EVT_OP*x*_TEB
MCPWM_EVT_OP*x*_TEA_EN	The count value of the timer that PWM operator *x* selects is equal to the value of timer stamp A[1]	MCPWM_EVT_OP*x*_TEA
MCPWM_EVT_TIMER*x*_TEP_EN	The count value of timer *x* is equal to the period value MCPWM_TIMER*x*_PERIOD	MCPWM_EVT_TIMER*x*_TEP
MCPWM_EVT_TIMER*x*_TEZ_EN	The count value of timer *x* is equal to 0	MCPWM_EVT_TIMER*x*_TEZ
MCPWM_EVT_TIMER*x*_STOP_EN	Timer *x* stops counting	MCPWM_EVT_OP*x*_TEA

[1] See Section 36.3.3.1 for a detailed description of timer stamp A and B.

36.3.5.3 MCPWM-Related ETM Tasks

When setting the enable field to 1, after inputting valid tasks, the corresponding response operation would be generated. For details, please refer to Table 36.3-8 below:

Table 36.3-8. MCPWM-Related ETM Tasks

Enable Field	Valid Task Received	Response Operation
MCPWM_TASK_CAP*x*_EN	MCPWM_TASK_CAP*x*	CAP*x* channel performs a capture operation
MCPWM_TASK_CLR*x*_OST_EN	MCPWM_TASK_CLR*x*_OST	PWM operator *x* clears the One-Shot Trip operation
MCPWM_TASK_TZ*x*_OST_EN	MCPWM_TASK_TZ*x*_OST	PWM operator *x* performs a One-Shot Trip (OST) operation
MCPWM_TASK_TIMER*x*_PERIOD_UP_EN	MCPWM_TASK_TIMER*x*_PERIOD_UP	The period of timer *x* is updated to the value configured in the period register MCPWM_TIMER*x*_PERIOD
MCPWM_TASK_TIMER*x*_SYNC_EN	MCPWM_TASK_TIMER*x*_SYN	Timer *x* performs a sync operation
MCPWM_TASK_GEN_STOP_EN	MCPWM_TASK_GEN_STOP	All the timers stop counting and the PWM signals output by all PWM operators remain unchanged
MCPWM_TASK_CMPR*x*_B_UP_EN	MCPWM_TASK_CMPR*x*_B_UP	Timer stamp B of the PWM operator *x* is updated to the value of the shadow register MCPWM_GEN*x*_B

Enable Field	Valid Task Received	Response Operation
MCPWM_TASK_CMPR*x*_A_UP_EN	MCPWM_TASK_CMPR*x*_A_UP	Timer stamp A of the PWM operator *x* is updated to the value of the shadow register MCPWM_GEN*x*_A

36.3.6 Interrupts

- MCPWM_TIMER*x*_STOP_INT: triggered when timer*x* stops. Only could be triggered when the MCPMW_TIMER*x*_STOP_INT_ENA field in the MCPMW_INT_ENA_REG register is set.
- MCPWM_TIMER*x*_TEZ_INT: triggered by the TEZ event of PWM timer*x*. Only could be triggered when the MCPWM_TIMER*x*_TEZ_INT_ENA field in the MCPMW_INT_ENA_REG register is set.
- MCPWM_TIMER*x*_TEP_INT: triggered by the TEP event of PWM timer*x*. Only could be triggered when the MCPWM_TIMER*x*_TEP_INT_ENA field in the MCPMW_INT_ENA_REG register is set.
- MCPWM_FAULT*x*_INT: triggered when fault_event*x* starts. Only could be triggered when the MCPWM_FAULT*x*_INT_ENA field in the MCPMW_INT_ENA_REG register is set.
- MCPWM_FAULT*x*_CLR_INT: triggered after fault_event*x* ends. Only could be triggered when the MCPWM_FAULT*x*_CLR_INT_ENA field in the MCPMW_INT_ENA_REG register is set.
- MCPWM_CMPR*x*_TEA_INT: triggered by the TEA event of PWM operator*x*. Only could be triggered when the MCPWM_CMPR*x*_TEA_INT_ENA field in the MCPMW_INT_ENA_REG register is set.
- MCPWM_CMPR*x*_TEB_INT: triggered by the TEB event of PWM operator*x*. Only could be triggered when the MCPWM_CMPR*x*_TEB_INT_ENA field in the MCPMW_INT_ENA_REG register is set.
- MCPWM_TZ*x*_CBC_INT: triggered by the CBC action of PWM*x*. Only could be triggered when the MCPWM_TZ*x*_CBC_INT_ENA field in the MCPMW_INT_ENA_REG register is set.
- MCPWM_TZ*x*_OST_INT: triggered by the OST action of PWM*x*. Only could be triggered when the MCPWM_TZ*x*_OST_INT_ENA field in the MCPMW_INT_ENA_REG register is set.
- MCPWM_CAP*x*_INT: triggered by the capture event on channel*x*. Only could be triggered when the MCPWM_CAP*x*_INT_ENA field in the MCPMW_INT_ENA_REG register is set.

36.4 Register Summary

The addresses in this section are relative to Motor Control PWM base address provided in Table 4.3-2 in Chapter 4 *System and Memory*.

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

Name	Description	Address	Access
Prescaler Configuration			
MCPWM_CLK_CFG_REG	PWM clock prescaler register	0x0000	R/W
MCPWM Timer 0 Configuration and Status			
MCPWM_TIMER0_CFG0_REG	PWM timer0 period and update method configuration register	0x0004	R/W
MCPWM_TIMER0_CFG1_REG	PWM timer0 working mode and start/stop control configuration register	0x0008	varies
MCPWM_TIMER0_SYNC_REG	PWM timer0 sync function configuration register	0x000C	R/W
MCPWM_TIMER0_STATUS_REG	PWM timer0 status register	0x0010	RO
MCPWM Timer 1 Configuration and Status			
MCPWM_TIMER1_CFG0_REG	PWM timer1 period and update method configuration register	0x0014	R/W
MCPWM_TIMER1_CFG1_REG	PWM timer1 working mode and start/stop control configuration register	0x0018	varies
MCPWM_TIMER1_SYNC_REG	PWM timer1 sync function configuration register	0x001C	R/W
MCPWM_TIMER1_STATUS_REG	PWM timer1 status register	0x0020	RO
MCPWM Timer 2 Configuration and status			
MCPWM_TIMER2_CFG0_REG	PWM timer2 period and update method configuration register	0x0024	R/W
MCPWM_TIMER2_CFG1_REG	PWM timer2 working mode and start/stop control configuration register	0x0028	varies
MCPWM_TIMER2_SYNC_REG	PWM timer2 sync function configuration register	0x002C	R/W
MCPWM_TIMER2_STATUS_REG	PWM timer2 status register	0x0030	RO
Common Configuration for MCPWM Timers			
MCPWM_TIMER_SYNCI_CFG_REG	Synchronization input selection for three PWM timers	0x0034	R/W
MCPWM_OPERATOR_TIMERSEL_REG	Select specific timer for PWM operators	0x0038	R/W
MCPWM Operator 0 Configuration and Status			
MCPWM_GEN0_STMP_CFG_REG	Transfer status and update method for time stamp registers A and B	0x003C	varies
MCPWM_GEN0_TSTMP_A_REG	Shadow register for register A	0x0040	R/W
MCPWM_GEN0_TSTMP_B_REG	Shadow register for register B	0x0044	R/W
MCPWM_GEN0_CFG0_REG	Fault event T0 and T1 handling	0x0048	R/W
MCPWM_GEN0_FORCE_REG	Permissives to force PWM0A and PWM0B outputs by software	0x004C	R/W

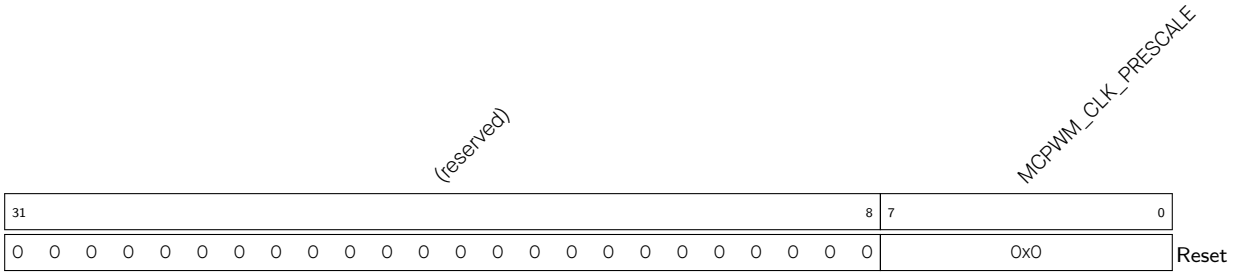
Name	Description	Address	Access
MCPWM_GEN0_A_REG	Actions triggered by events on PWM0A	0x0050	R/W
MCPWM_GEN0_B_REG	Actions triggered by events on PWM0B	0x0054	R/W
MCPWM_DT0_CFG_REG	Dead time type selection and configuration	0x0058	R/W
MCPWM_DT0_FED_CFG_REG	Shadow register for falling edge delay (FED)	0x005C	R/W
MCPWM_DT0_RED_CFG_REG	Shadow register for rising edge delay (RED)	0x0060	R/W
MCPWM_CARRIER0_CFG_REG	Carrier enable and configuration	0x0064	R/W
MCPWM_FH0_CFG0_REG	Actions on PWM0A and PWM0B trip events	0x0068	R/W
MCPWM_FH0_CFG1_REG	Software triggers for fault handler actions	0x006C	R/W
MCPWM_FH0_STATUS_REG	Status of fault events	0x0070	RO
MCPWM Operator 1 Configuration and Status			
MCPWM_GEN1_STMP_CFG_REG	Transfer status and update method for time stamp registers A and B	0x0074	varies
MCPWM_GEN1_TSTMP_A_REG	Shadow register for register A	0x0078	R/W
MCPWM_GEN1_TSTMP_B_REG	Shadow register for register B	0x007C	R/W
MCPWM_GEN1_CFG0_REG	Fault event T0 and T1 handling	0x0080	R/W
MCPWM_GEN1_FORCE_REG	Permissives to force PWM1A and PWM1B outputs by software	0x0084	R/W
MCPWM_GEN1_A_REG	Actions triggered by events on PWM1A	0x0088	R/W
MCPWM_GEN1_B_REG	Actions triggered by events on PWM1B	0x008C	R/W
MCPWM_DT1_CFG_REG	Dead time type selection and configuration	0x0090	R/W
MCPWM_DT1_FED_CFG_REG	Shadow register for falling edge delay (FED)	0x0094	R/W
MCPWM_DT1_RED_CFG_REG	Shadow register for rising edge delay (RED)	0x0098	R/W
MCPWM_CARRIER1_CFG_REG	Carrier enable and configuration	0x009C	R/W
MCPWM_FH1_CFG0_REG	Actions on PWM1A and PWM1B trip events	0x00A0	R/W
MCPWM_FH1_CFG1_REG	Software triggers for fault handler actions	0x00A4	R/W
MCPWM_FH1_STATUS_REG	Status of fault events	0x00A8	RO
MCPWM Operator 2 Configuration and Status			
MCPWM_GEN2_STMP_CFG_REG	Transfer status and update method for time stamp registers A and B	0x00AC	varies
MCPWM_GEN2_TSTMP_A_REG	Shadow register for register A	0x00B0	R/W
MCPWM_GEN2_TSTMP_B_REG	Shadow register for register B	0x00B4	R/W
MCPWM_GEN2_CFG0_REG	Fault event T0 and T1 handling	0x00B8	R/W
MCPWM_GEN2_FORCE_REG	Permissives to force PWM2A and PWM2B outputs by software	0x00BC	R/W
MCPWM_GEN2_A_REG	Actions triggered by events on PWM2A	0x00C0	R/W
MCPWM_GEN2_B_REG	Actions triggered by events on PWM2B	0x00C4	R/W
MCPWM_DT2_CFG_REG	Dead time type selection and configuration	0x00C8	R/W
MCPWM_DT2_FED_CFG_REG	Shadow register for falling edge delay (FED)	0x00CC	R/W
MCPWM_DT2_RED_CFG_REG	Shadow register for rising edge delay (RED)	0x00D0	R/W
MCPWM_CARRIER2_CFG_REG	Carrier enable and configuration	0x00D4	R/W
MCPWM_FH2_CFG0_REG	Actions on PWM2A and PWM2B trip events	0x00D8	R/W
MCPWM_FH2_CFG1_REG	Software triggers for fault handler actions	0x00DC	R/W
MCPWM_FH2_STATUS_REG	Status of fault events	0x00E0	RO

Name	Description	Address	Access
Fault Detection Configuration and Status			
MCPWM_FAULT_DETECT_REG	Fault detection configuration and status	0x00E4	varies
Capture Configuration and Status			
MCPWM_CAP_TIMER_CFG_REG	Configure capture timer	0x00E8	varies
MCPWM_CAP_TIMER_PHASE_REG	Phase for capture timer sync	0x00EC	R/W
MCPWM_CAP_CH0_CFG_REG	Capture channel 0 configuration and enable	0x00F0	varies
MCPWM_CAP_CH1_CFG_REG	Capture channel 1 configuration and enable	0x00F4	varies
MCPWM_CAP_CH2_CFG_REG	Capture channel 2 configuration and enable	0x00F8	varies
MCPWM_CAP_CH0_REG	ch0 capture value status register	0x00FC	RO
MCPWM_CAP_CH1_REG	ch1 capture value status register	0x0100	RO
MCPWM_CAP_CH2_REG	ch2 capture value status register	0x0104	RO
MCPWM_CAP_STATUS_REG	Edge of last capture trigger	0x0108	RO
Enable Update of Active Registers			
MCPWM_UPDATE_CFG_REG	Enable update	0x010C	R/W
Manage Interrupts			
MCPWM_INT_ENA_REG	Interrupt enable bits	0x0110	R/W
MCPWM_INT_RAW_REG	Raw interrupt status	0x0114	R/WTC /SS
MCPWM_INT_ST_REG	Masked interrupt status	0x0118	RO
MCPWM_INT_CLR_REG	Interrupt clear bits	0x011C	WT
MCPWM Event Enable Register			
MCPWM_EVT_EN_REG	MCPWM event enable register	0x0120	R/W
MCPWM Task Enable Register			
MCPWM_TASK_EN_REG	MCPWM task enable register	0x0124	R/W
MCPWM APB Configuration Register			
MCPWM_CLK_REG	MCPWM APB configuration register	0x0128	R/W
Version Register			
MCPWM_VERSION_REG	Version register	0x012C	R/W

36.5 Registers

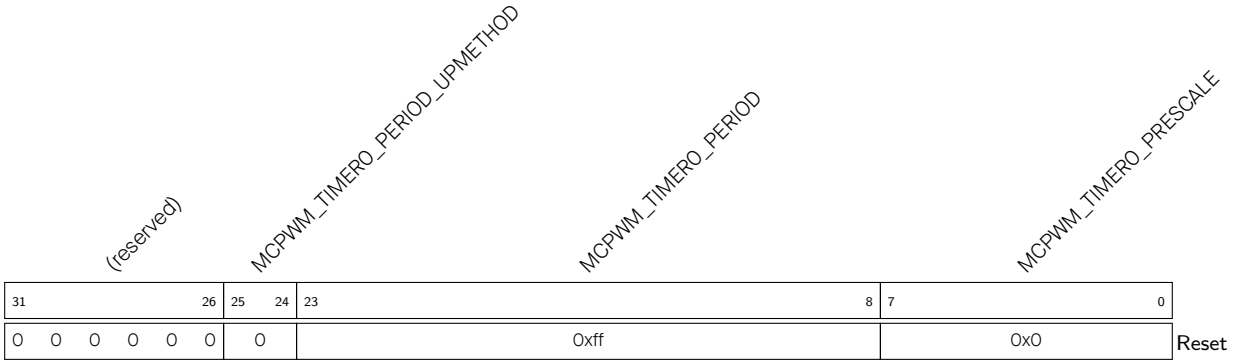
The addresses in this section are relative to Motor Control PWM base address provided in Table 4.3-2 in Chapter 4 *System and Memory*.

Register 36.1. MCPWM_CLK_CFG_REG (0x0000)



MCPWM_CLK_PRESCALE Configures the prescaler value of clock, so that the period of PWM_CLK = 6.25ns * (PWM_CLK_PRESCALE + 1). (R/W)

Register 36.2. MCPWM_TIMER0_CFG0_REG (0x0004)



MCPWM_TIMER0_PRESCALE Configures the prescaler value of timer0, so that the period of PT0_CLK = Period of PWM_CLK * (PWM_TIMER0_PRESCALE + 1). (R/W)

MCPWM_TIMER0_PERIOD Configures the period shadow register of PWM timer0. (R/W)

MCPWM_TIMER0_PERIOD_UPMETHOD Configures the update method for active register of PWM timer0 period.

0: Immediate

1: TEZ

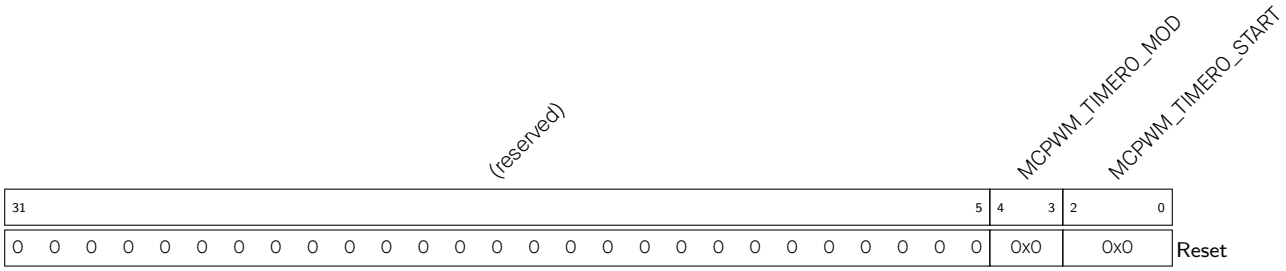
2: sync

3: TEZ | sync

TEZ here and below means timer equals zero event.

(R/W)

Register 36.3. MCPWM_TIMER0_CFG1_REG (0x0008)



MCPWM_TIMER0_START Configures conditions to start/stop PWM timer0.

0: If PWM timer0 starts, then stops at TEZ

1: If timer0 starts, then stops at TEP

2: PWM timer0 starts and runs on

3: Timer0 starts and stops at the next TEZ

4: Timer0 starts and stops at the next TEP

5: Invalid. No effect

6: Invalid. No effect

7: Invalid. No effect

TEP here and below means the event that happens when the timer equals to period.

(R/W/SC)

MCPWM_TIMER0_MOD Configures the working mode of PWM timer0.

0: Freeze

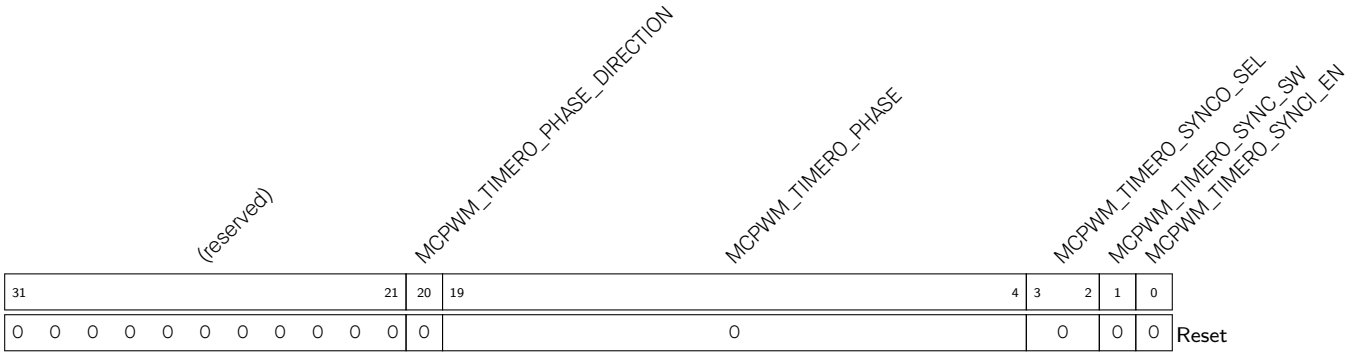
1: Increase mode

2: Decrease mode

3: Up-down mode

(R/W)

Register 36.4. MCPWM_TIMER0_SYNC_REG (0x000C)



MCPWM_TIMER0_SYNCI_EN Configures whether or not to enable timer reloading with phase on sync input event.

0: Disable

1: Enable

(R/W)

MCPWM_TIMER0_SYNC_SW Configures whether to trigger a software sync.

0: No effect

1: Trigger a software sync

(R/W)

MCPWM_TIMER0_SYNCO_SEL Configures PWM timer1 sync out selection.

0: sync_in. The sync out will always generate when toggling the MCPWM_TIMER0_SYNC_SW bit.

1: TEZ

2: TEP

3: No effect

(R/W)

MCPWM_TIMER0_PHASE Configures the phase for timer reload on sync event. (R/W)

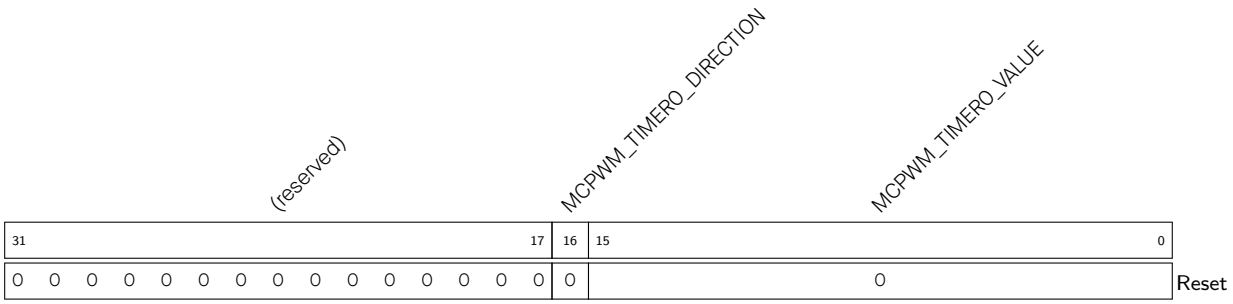
MCPWM_TIMER0_PHASE_DIRECTION Configures the PWM timer0's direction when timer0 mode is up-down mode.

0: Increase

1: Decrease

(R/W)

Register 36.5. MCPWM_TIMER0_STATUS_REG (0x0010)



MCPWM_TIMER0_VALUE Represents current PWM timer0 counter value. (RO)

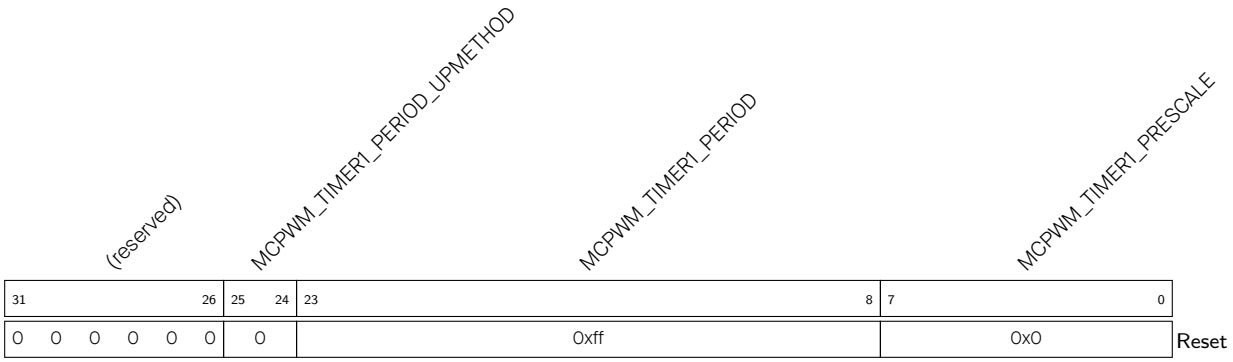
MCPWM_TIMER0_DIRECTION Represents current PWM timer0 counter direction.

0: Increment

1: Decrement

(RO)

Register 36.6. MCPWM_TIMER1_CFG0_REG (0x0014)



MCPWM_TIMER1_PRESCALE Configures the prescaler value of timer1, so that the period of PTO_CLK = Period of PWM_CLK * (PWM_timer1_PRESCALE + 1). (R/W)

MCPWM_TIMER1_PERIOD Period shadow register of PWM timer1. (R/W)

MCPWM_TIMER1_PERIOD_UPMETHOD Configures the update method for active register of PWM timer1 period.

0: Immediate

1: TEZ

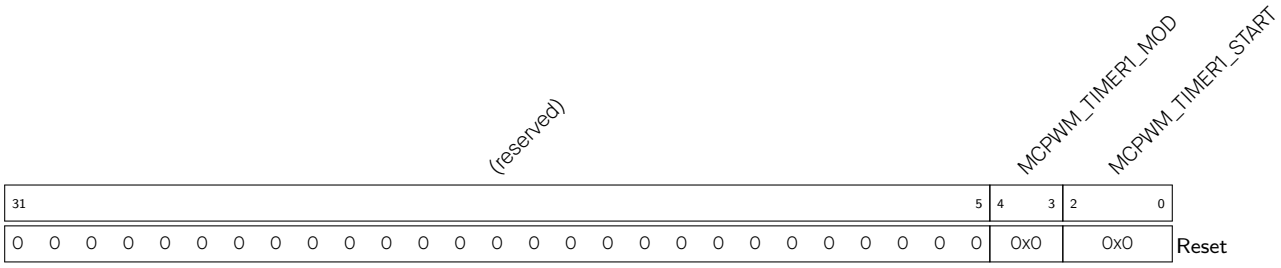
2: Sync

3: TEZ | sync

TEZ here and below means timer equals zero event.

(R/W)

Register 36.7. MCPWM_TIMER1_CFG1_REG (0x0018)



MCPWM_TIMER1_START Configures conditions to start/stop PWM timer1.

0: If PWM timer1 starts, then stops at TEZ

1: If timer1 starts, then stops at TEP

2: PWM timer1 starts and runs on

3: Timer1 starts and stops at the next TEZ

4: Timer1 starts and stops at the next TEP

5-7: Invalid. No effect

TEP here and below means the event that happens when the timer equals to period.

(R/W/SC)

MCPWM_TIMER1_MOD Configures the working mode of PWM timer1.

0: Freeze

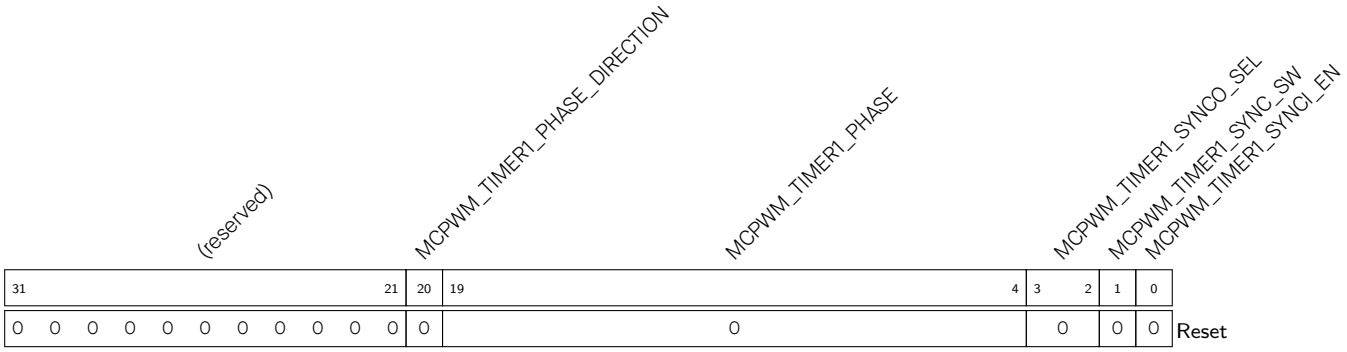
1: Increase mode

2: Decrease mode

3: Up-down mode

(R/W)

Register 36.8. MCPWM_TIMER1_SYNC_REG (0x001C)



MCPWM_TIMER1_SYNCI_EN Configures whether or not to enable timer reloading with phase on sync input event.

0: Disable

1: Enable

(R/W)

MCPWM_TIMER1_SYNC_SW Configures whether to trigger a software sync.

0: No effect

1: Trigger a software sync

(R/W)

MCPWM_TIMER1_SYNCO_SEL Configures PWM timer1 sync out selection.

0: sync_in

1: TEZ

2: TEP, and sync out will always generate when toggling the reg_timer1_sync_sw bit.

3: No effect

(R/W)

MCPWM_TIMER1_PHASE Phase for timer reload on sync event. (R/W)

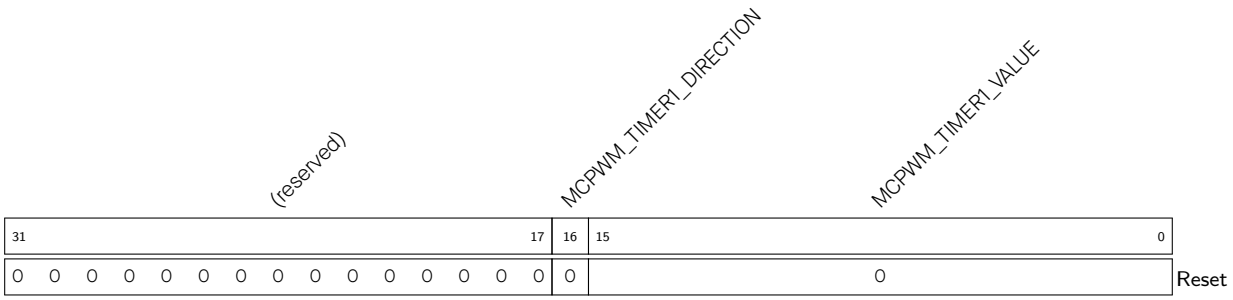
MCPWM_TIMER1_PHASE_DIRECTION Configures the PWM timer1's direction when timer1 is in up-down mode.

0: Increase

1: Decrease

(R/W)

Register 36.9. MCPWM_TIMER1_STATUS_REG (0x0020)



MCPWM_TIMER1_VALUE Represents current PWM timer1 counter value. (RO)

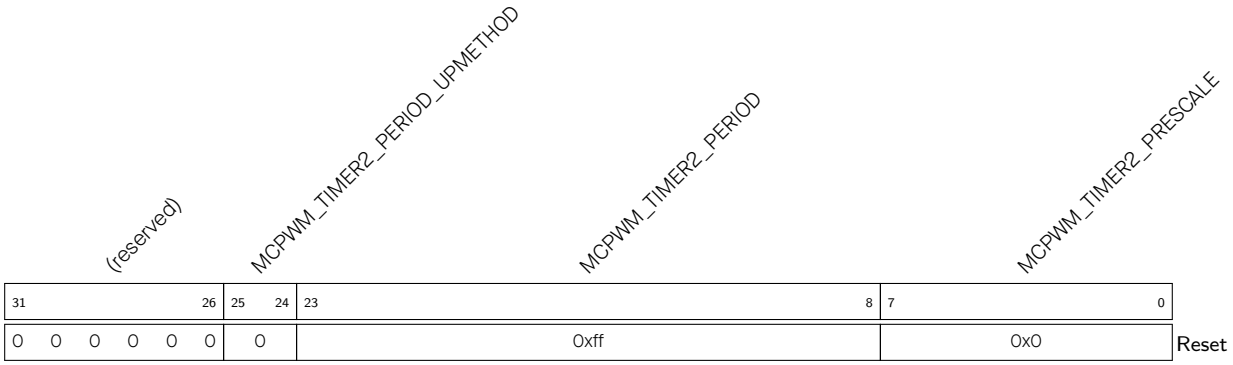
MCPWM_TIMER1_DIRECTION Represents current PWM timer1 counter direction.

0: Increment

1: Decrement

(RO)

Register 36.10. MCPWM_TIMER2_CFG0_REG (0x0024)



MCPWM_TIMER2_PRESCALE Configures the prescaler value of timer2, so that the period of PTO_CLK = Period of PWM_CLK * (PWM_timer2_PRESCALE + 1). (R/W)

MCPWM_TIMER2_PERIOD Period shadow register of PWM timer2. (R/W)

MCPWM_TIMER2_PERIOD_UPMETHOD Configures the update method for active register of PWM timer2 period.

0: Immediate

1: TEZ

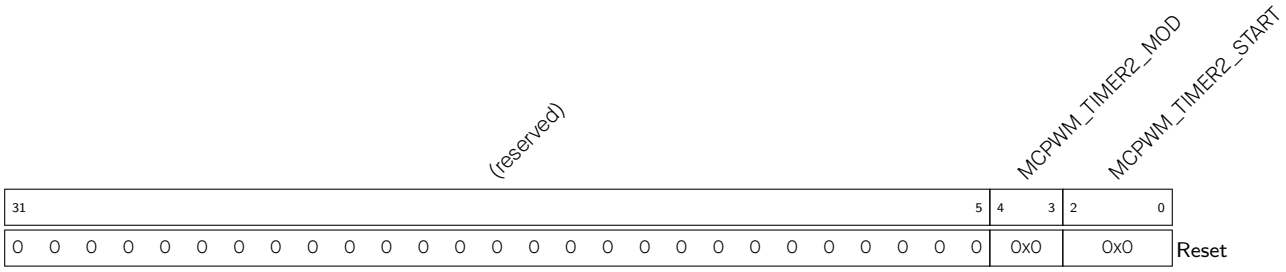
2: Sync

3: TEZ | sync

TEZ here and below means timer equal zero event.

(R/W)

Register 36.11. MCPWM_TIMER2_CFG1_REG (0x0028)



MCPWM_TIMER2_START Configures whether or not to start/stop PWM timer2.

0: If PWM timer2 starts, then stops at TEZ

1: If timer2 starts, then stops at TEP

2: PWM timer2 starts and runs on

3: Timer2 starts and stops at the next TEZ

4: Timer2 starts and stops at the next TEP

5: Invalid. No effect

6: Invalid. No effect

7: Invalid. No effect

TEP here and below means the event that happens when the timer equals to period.

(R/W/SC)

MCPWM_TIMER2_MOD Configures the working mode of PWM timer2.

0: Freeze

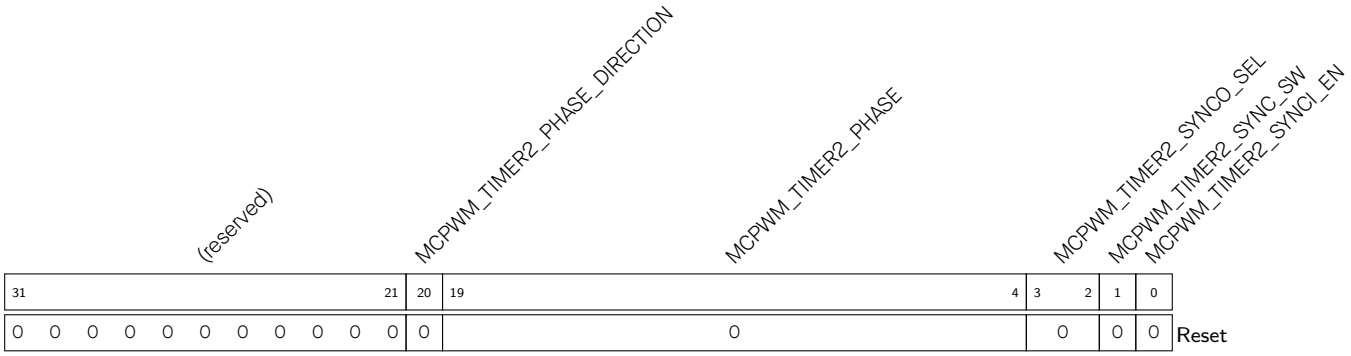
1: Increase mode

2: Decrease mode

3: Up-down mode

(R/W)

Register 36.12. MCPWM_TIMER2_SYNC_REG (0x002C)



MCPWM_TIMER2_SYNCI_EN Configures whether or not to enable timer reloading with phase on sync input event.

0: Disable

1: Enable

(R/W)

MCPWM_TIMER2_SYNC_SW Configures whether to trigger a software sync.

0: No effect

1: Trigger a software sync

(R/W)

MCPWM_TIMER2_SYNCO_SEL Configures PWM timer2 sync out selection.

0: sync_in

1: TEZ

2: TEP, and sync out will always generate when toggling the reg_timer0_sync_sw bit

3: No effect

(R/W)

MCPWM_TIMER2_PHASE Configures phase for timer reload on sync event. (R/W)

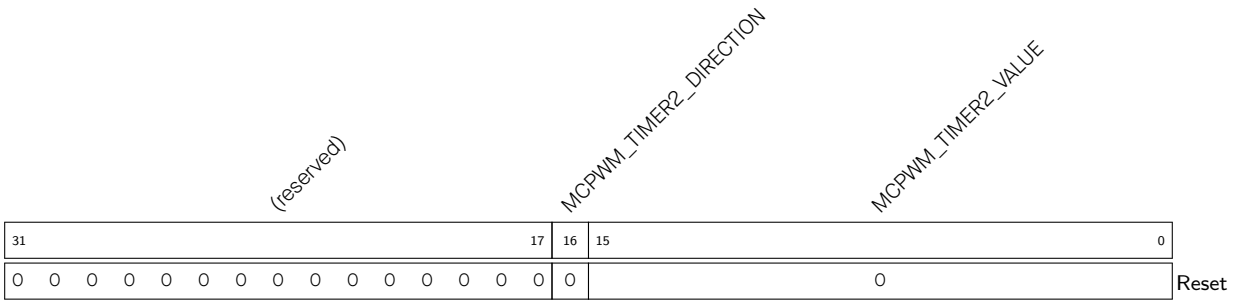
MCPWM_TIMER2_PHASE_DIRECTION Configures the PWM timer2's direction when timer2 is in up-down mode.

0: Increase

1: Decrease

(R/W)

Register 36.13. MCPWM_TIMER2_STATUS_REG (0x0030)



MCPWM_TIMER2_VALUE Represents current PWM timer2 counter value. (RO)

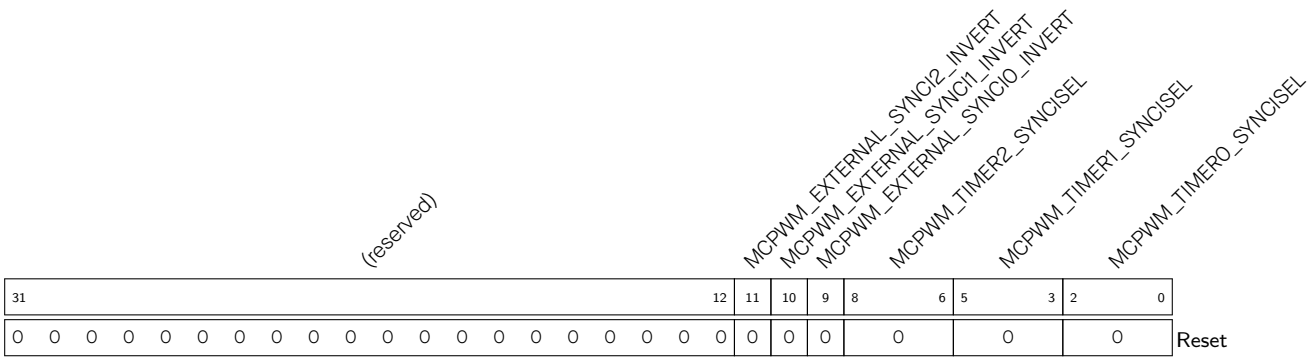
MCPWM_TIMER2_DIRECTION Represents current PWM timer2 counter direction.

0: Increment

1: Decrement

(RO)

Register 36.14. MCPWM_TIMER_SYNCI_CFG_REG (0x0034)



MCPWM_TIMER0_SYNCISEL Configures sync input for PWM timer0.

1: PWM timer0 sync out
2: PWM timer1 sync out
3: PWM timer2 sync out
4: SYNC0 from GPIO matrix
5: SYNC1 from GPIO matrix
6: SYNC2 from GPIO matrix
Other values: no sync input selected
(R/W)

MCPWM_TIMER1_SYNCISEL Configures sync input for PWM timer1.

1: PWM timer0 sync out
2: PWM timer1 sync out
3: PWM timer2 sync out
4: SYNC0 from GPIO matrix
5: SYNC1 from GPIO matrix
6: SYNC2 from GPIO matrix
Other values: no sync input selected
(R/W)

MCPWM_TIMER2_SYNCISEL Configures sync input for PWM timer2.

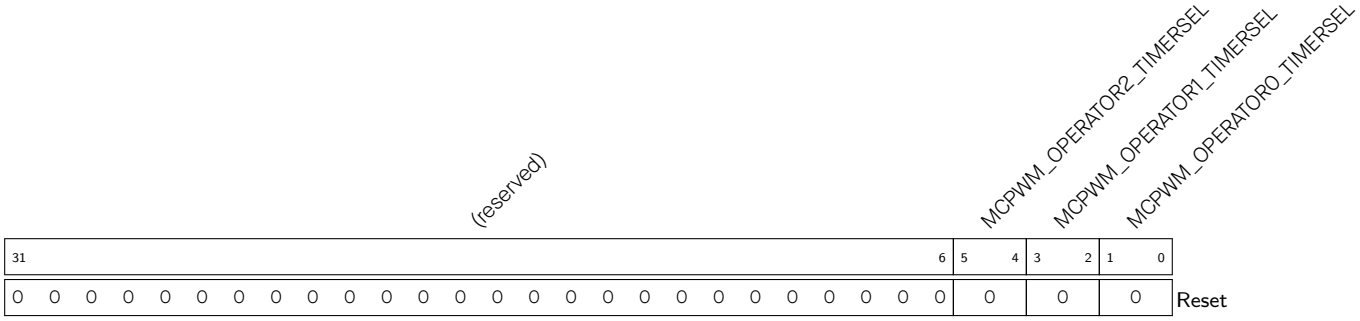
1: PWM timer0 sync out
2: PWM timer1 sync out
3: PWM timer2 sync out
4: SYNC0 from GPIO matrix
5: SYNC1 from GPIO matrix
6: SYNC2 from GPIO matrix
Other values: no sync input selected
(R/W)

MCPWM_EXTERNAL_SYNCI0_INVERT Inverts SYNC0 from GPIO matrix. (R/W)

MCPWM_EXTERNAL_SYNCI1_INVERT Inverts SYNC1 from GPIO matrix. (R/W)

MCPWM_EXTERNAL_SYNCI2_INVERT Inverts SYNC2 from GPIO matrix. (R/W)

Register 36.15. MCPWM_OPERATOR_TIMERSEL_REG (0x0038)



MCPWM_OPERATOR0_TIMERSEL Configures which PWM timer will be the timing reference for PWM operator0.

0: timer0

1: timer1

2: timer2

3: Invalid

(R/W)

MCPWM_OPERATOR1_TIMERSEL Configures which PWM timer will be the timing reference for PWM operator1.

0: timer0

1: timer1

2: timer2

3: Invalid

(R/W)

MCPWM_OPERATOR2_TIMERSEL Configures which PWM timer will be the timing reference for PWM operator2.

0: timer0

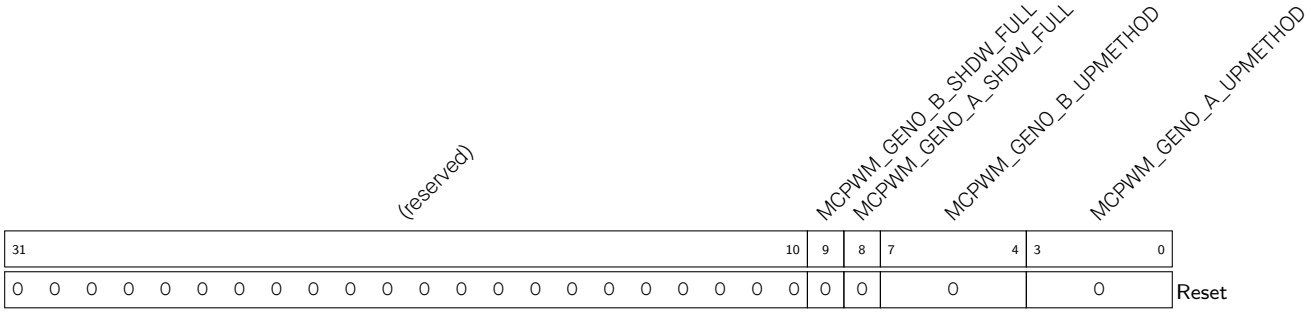
1: timer1

2: timer2

3: Invalid

(R/W)

Register 36.16. MCPWM_GEN0_STMP_CFG_REG (0x003C)



MCPWM_GEN0_A_UPMETHOD Configures the update method for PWM generator 0 time stamp A's active register.

When all bits are set to 0: Immediately

When bit0 is set to 1: TEZ

When bit1 is set to 1: TEP

When bit2 is set to 1: Sync

When bit3 is set to 1: Disable the update

(R/W)

MCPWM_GEN0_B_UPMETHOD Configures the update method for PWM generator 0 time stamp B's active register. (R/W)

When all bits are set to 0: Immediately

When bit0 is set to 1: TEZ

When bit1 is set to 1: TEP

When bit2 is set to 1: Sync

When bit3 is set to 1: Disable the update

(R/W)

MCPWM_GEN0_A_SHDW_FULL Configures whether or not the value in the shadow register is written to the corresponding active register at a specific time. This field is set and reset by hardware.

0: Write the latest value in the shadow register to A's active register

1: Write the value to the PWM generator 0 time stamp A's shadow register, and wait to be transferred to A's active register

(R/SC/WTC)

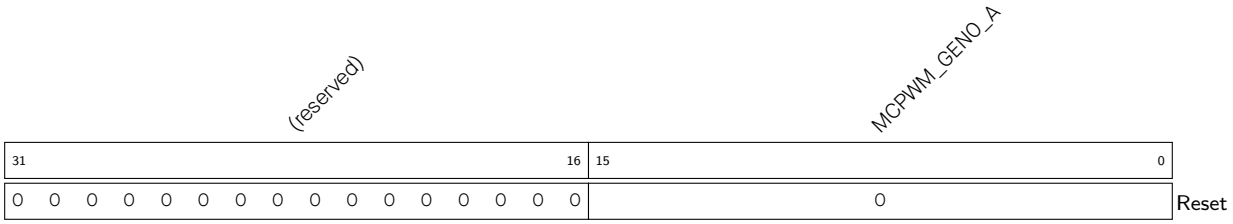
MCPWM_GEN0_B_SHDW_FULL Configures whether or not the value in the shadow register is written to the corresponding active register at a specific time. This field is set and reset by hardware.

0: Write the latest value in the shadow register to B's active register.

1: Write the value to the PWM generator 0 time stamp B's shadow register, and wait to be transferred to B's active register.

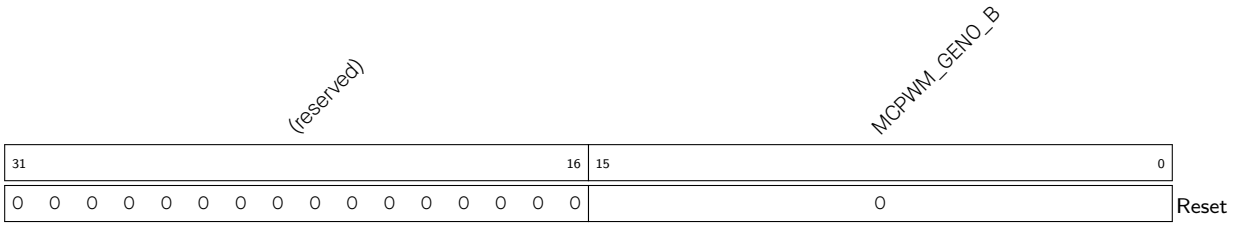
(R/SC/WTC)

Register 36.17. MCPWM_GEN0_TSTMP_A_REG (0x0040)



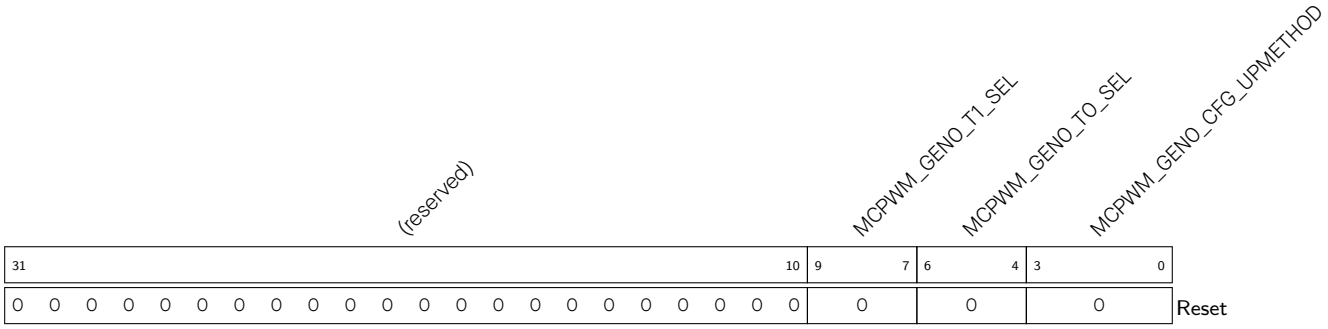
MCPWM_GEN0_A Shadow register for PWM generator 0 time stamp A. (R/W)

Register 36.18. MCPWM_GEN0_TSTMP_B_REG (0x0044)



MCPWM_GEN0_B Shadow register for PWM generator 0 time stamp B. (R/W)

Register 36.19. MCPWM_GEN0_CFG0_REG (0x0048)



MCPWM_GEN0_CFG_UPMETHOD Configures update method for PWM generator 0's active register.

When all bits are set to 0: Immediately

When bit0 is set to 1: TEZ

When bit1 is set to 1: TEP

When bit2 is set to 1: Sync

When bit3 is set to 1: Disable the update

(R/W)

MCPWM_GEN0_T0_SEL Configures source selection for PWM generator 0 event_t0, take effect immediately.

0: fault_event0

1: fault_event1

2: fault_event2

3: sync_taken

4: None

(R/W)

MCPWM_GEN0_T1_SEL Configures source selection for PWM generator 0 event_t1, take effect immediately

0: fault_event0

1: fault_event1

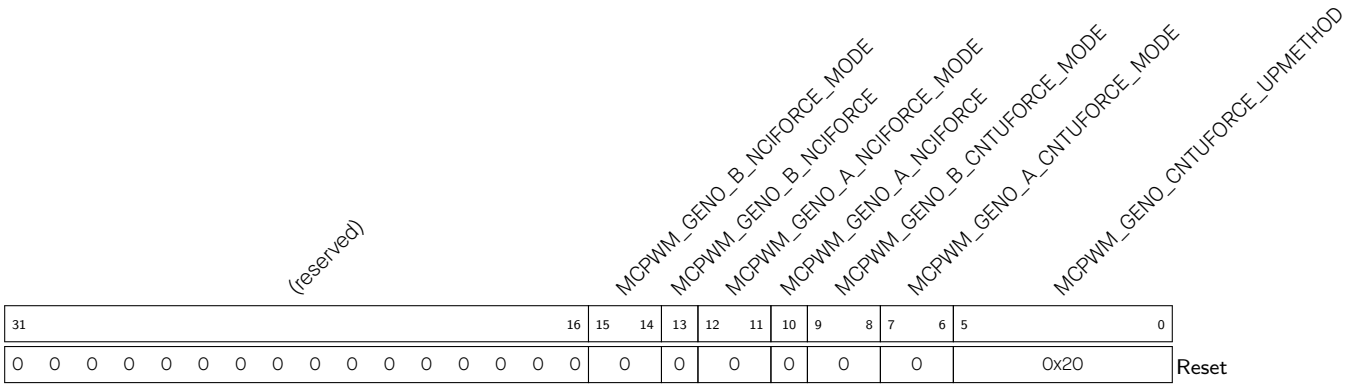
2: fault_event2

3: sync_taken

4: None

(R/W)

Register 36.20. MCPWM_GEN0_FORCE_REG (0x004C)



MCPWM_GEN0_CNTUFORCE_UPMETHOD Configures update method for continuous software force of PWM generator0.

When all bits are set to 0: Immediately

When bit0 is set to 1: TEZ

When bit1 is set to 1: TEP

When bit2 is set to 1: TEA

When bit3 is set to 1: TEB

When bit4 is set to 1: Sync

When bit5 is set to 1: Disable update

TEA/B means an event generated when the timer's value equals to that of register A/B. (R/W)

MCPWM_GEN0_A_CNTUFORCE_MODE Configures continuous software force mode for PWM0A.

0: Disabled

1: Low

2: High

3: Disabled

(R/W)

MCPWM_GEN0_B_CNTUFORCE_MODE Configures continuous software force mode for PWM0B. See details in MCPWM_GEN0_A_CNTUFORCE_MODE. (R/W)

MCPWM_GEN0_A_NCIFORCE Configures whether or not to trigger a non-continuous immediate software-force event for PWM0A.

0: No effect

1: Trigger a force event

(R/W)

MCPWM_GEN0_A_NCIFORCE_MODE Configures non-continuous immediate software force mode for PWM0A.

0: Disabled

1: Low

2: High

3: Disabled

(R/W)

Continued on the next page...

Register 36.20. MCPWM_GEN0_FORCE_REG (0x004C)

Continued from the previous page...

MCPWM_GEN0_B_NCIFORCE Configures whether or not to trigger a non-continuous immediate software-force event for PWM0B.

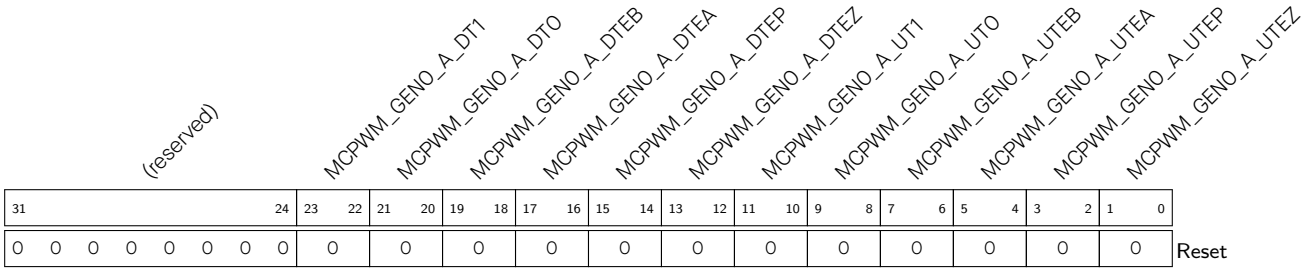
0: No effect

1: Trigger a force event

(R/W)

MCPWM_GEN0_B_NCIFORCE_MODE Configures non-continuous immediate software force mode for PWM0B. See details in MCPWM_GEN0_A_NCIFORCE_MODE. (R/W)

Register 36.21. MCPWM_GEN0_A_REG (0x0050)



MCPWM_GEN0_A_UTEZ Configures action on PWM0A triggered by event TEZ when timer increasing.

0: No change

1: Low

2: High

3: Toggle

(R/W)

MCPWM_GEN0_A_UTEP Configures action on PWM0A triggered by event TEP when timer increasing. See details in MCPWM_GEN0_A_UTEZ. (R/W)

MCPWM_GEN0_A_UTEA Configures action on PWM0A triggered by event TEA when timer increasing. See details in MCPWM_GEN0_A_UTEZ. (R/W)

MCPWM_GEN0_A_UTEB Configures action on PWM0A triggered by event TEB when timer increasing. See details in MCPWM_GEN0_A_UTEZ. (R/W)

MCPWM_GEN0_A_UT0 Configures action on PWM0A triggered by event_t0 when timer increasing. See details in MCPWM_GEN0_A_UTEZ. (R/W)

MCPWM_GEN0_A_UT1 Configures action on PWM0A triggered by event_t1 when timer increasing. See details in MCPWM_GEN0_A_UTEZ. (R/W)

MCPWM_GEN0_A_DTEZ Configures action on PWM0A triggered by event TEZ when timer decreasing. See details in MCPWM_GEN0_A_UTEZ. (R/W)

MCPWM_GEN0_A_DTEP Configures action on PWM0A triggered by event TEP when timer decreasing. See details in MCPWM_GEN0_A_UTEZ. (R/W)

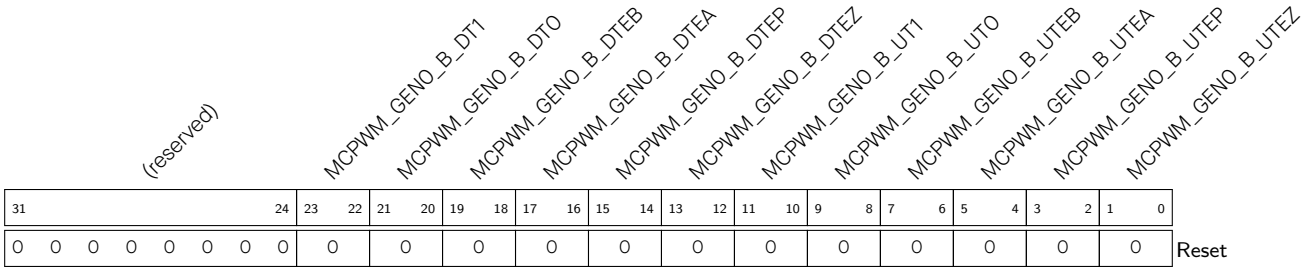
MCPWM_GEN0_A_DTEA Configures action on PWM0A triggered by event TEA when timer decreasing. See details in MCPWM_GEN0_A_UTEZ. (R/W)

MCPWM_GEN0_A_DTEB Configures action on PWM0A triggered by event TEB when timer decreasing. See details in MCPWM_GEN0_A_UTEZ. (R/W)

MCPWM_GEN0_A_DT0 Configures action on PWM0A triggered by event_t0 when timer decreasing. See details in MCPWM_GEN0_A_UTEZ. (R/W)

MCPWM_GEN0_A_DT1 Configures action on PWM0A triggered by event_t1 when timer decreasing. See details in MCPWM_GEN0_A_UTEZ. (R/W)

Register 36.22. MCPWM_GEN0_B_REG (0x0054)



MCPWM_GEN0_B_UTEZ Configures action on PWM0B triggered by event TEZ when timer increasing.

0: No change

1: Low.

2: High.

3: Toggle

(R/W)

MCPWM_GEN0_B_UTEP Configures action on PWM0B triggered by event TEP when timer increasing. See details in MCPWM_GEN0_B_UTEZ. (R/W)

MCPWM_GEN0_B_UTEA Configures action on PWM0B triggered by event TEA when timer increasing. See details in MCPWM_GEN0_B_UTEZ. (R/W)

MCPWM_GEN0_B_UTEB Configures action on PWM0B triggered by event TEB when timer increasing. See details in MCPWM_GEN0_B_UTEZ. (R/W)

MCPWM_GEN0_B_UT0 Configures action on PWM0B triggered by event_t0 when timer increasing. See details in MCPWM_GEN0_B_UTEZ. (R/W)

MCPWM_GEN0_B_UT1 Configures action on PWM0B triggered by event_t1 when timer increasing. See details in MCPWM_GEN0_B_UTEZ. (R/W)

MCPWM_GEN0_B_DTEZ Configures action on PWM0B triggered by event TEZ when timer decreasing. See details in MCPWM_GEN0_B_UTEZ. (R/W)

MCPWM_GEN0_B_DTEP Configures action on PWM0B triggered by event TEP when timer decreasing. See details in MCPWM_GEN0_B_UTEZ. (R/W)

MCPWM_GEN0_B_DTEA Configures action on PWM0B triggered by event TEA when timer decreasing. See details in MCPWM_GEN0_B_UTEZ. (R/W)

MCPWM_GEN0_B_DTEB Configures action on PWM0B triggered by event TEB when timer decreasing. See details in MCPWM_GEN0_B_UTEZ. (R/W)

MCPWM_GEN0_B_DT0 Configures action on PWM0B triggered by event_t0 when timer decreasing. See details in MCPWM_GEN0_B_UTEZ. (R/W)

MCPWM_GEN0_B_DT1 Configures action on PWM0B triggered by event_t1 when timer decreasing. See details in MCPWM_GEN0_B_UTEZ. (R/W)

Register 36.23. MCPWM_DTO_CFG_REG (0x0058)

MCPWM_DTO_FED_UPMETHOD Configures update method for FED active register.

0: Immediate

When bit0 is set to 1: TEZ

When bit1 is set to 1: TEP

When bit2 is set to 1: sync

When bit3 is set to 1: disable the update (R/W)

MCPWM_DTO_RED_UPMETHOD Update method for RED active register. See details in MCPWM_DTO_FED_UPMETHOD. (R/W)

MCPWM_DTO_DEB_MODE Configures the S8 switch in Table 36.3-5. For typical configurations, please refer to Table 36.3-6. (R/W)

MCPWM_DTO_A_OUTSWAP Configures the S6 switch in Table 36.3-5. For typical configurations, please refer to Table 36.3-6. (R/W)

MCPWM_DTO_B_OUTSWAP Configures the S7 switch in Table 36.3-5. For typical configurations, please refer to Table 36.3-6. (R/W)

MCPWM_DTO_RED_INSEL Configures the S4 switch in Table 36.3-5. For typical configurations, please refer to Table 36.3-6. (R/W)

MCPWM_DTO_FED_INSEL Configures the S5 switch in Table 36.3-5. For typical configurations, please refer to Table 36.3-6. (R/W)

MCPWM_DTO_RED_OUTINVERT Configures the S2 switch in Table 36.3-5. For typical configurations, please refer to Table 36.3-6. (R/W)

MCPWM_DTO_FED_OUTINVERT Configures the S3 switch in Table 36.3-5. For typical configurations, please refer to Table 36.3-6. (R/W)

MCPWM_DTO_A_OUTBYPASS Configures the S1 switch in Table 36.3-5. For typical configurations, please refer to Table 36.3-6. (R/W)

MCPWM_DTO_B_OUTBYPASS Configures the S0 switch in Table 36.3-5. For typical configurations, please refer to Table 36.3-6. (R/W)

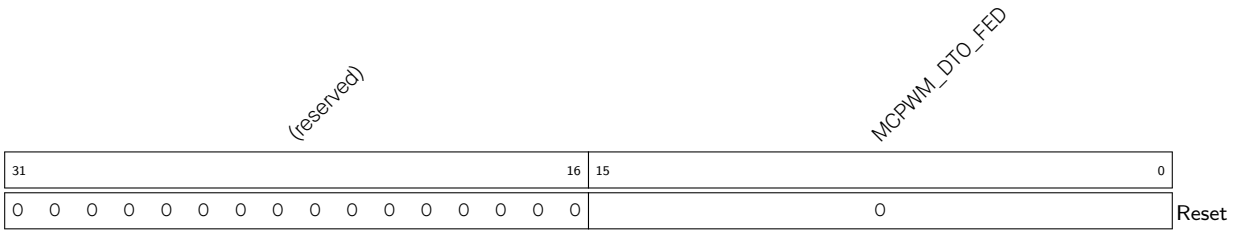
MCPWM_DTO_CLK_SEL Configures dead time generator 0 clock selection.

0: PWM_CLK

1: PT_CLK

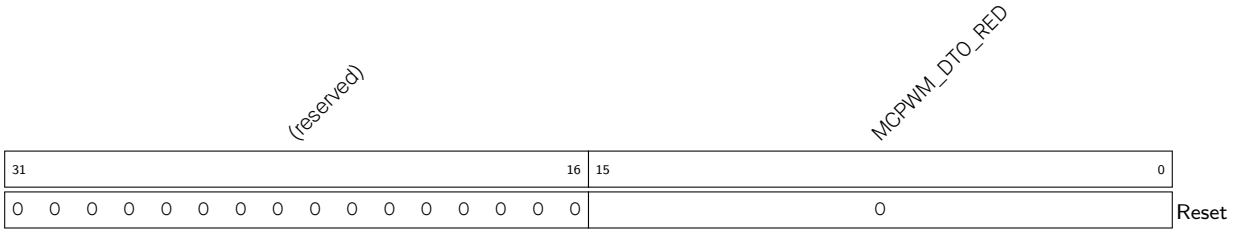
(R/W)

Register 36.24. MCPWM_DT0_FED_CFG_REG (0x005C)



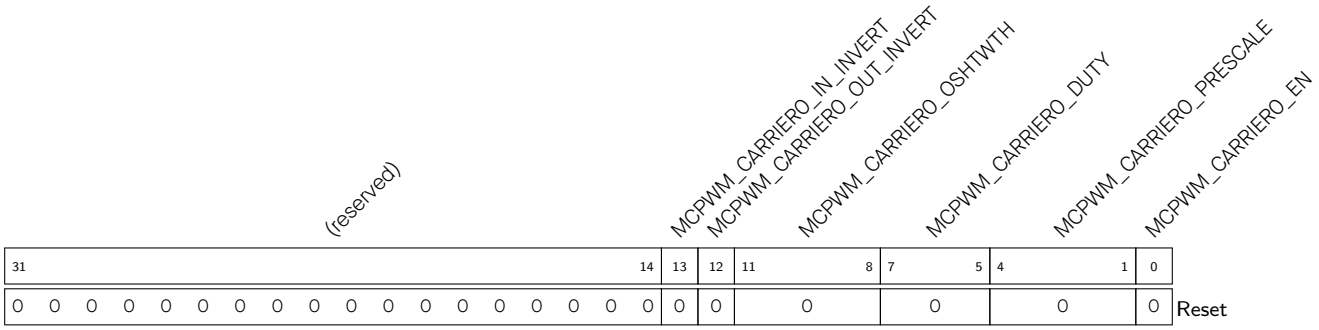
MCPWM_DT0_FED Shadow register for FED. (R/W)

Register 36.25. MCPWM_DT0_RED_CFG_REG (0x0060)



MCPWM_DT0_RED Shadow register for RED. (R/W)

Register 36.26. MCPWM_CARRIER0_CFG_REG (0x0064)



MCPWM_CARRIER0_EN Configures whether or not to enable carrier0.

0: Bypass

1: Enable

(R/W)

MCPWM_CARRIER0_PRESCALE Configures the prescale value of PWM carrier0 clock (PC_CLK), so that period of PC_CLK = period of PWM_CLK * (PWM_CARRIER0_PRESCALE + 1). (R/W)

MCPWM_CARRIER0_DUTY Configures carrier duty selection. Duty = PWM_CARRIER0_DUTY/8. (R/W)

MCPWM_CARRIER0_OSHTWTH Configures width of the first pulse in number of periods of the carrier. (R/W)

MCPWM_CARRIER0_OUT_INVERT Configures whether or not to invert the output of PWM0A and PWM0B for this submodule.

0: No effect

1: Invert

(R/W)

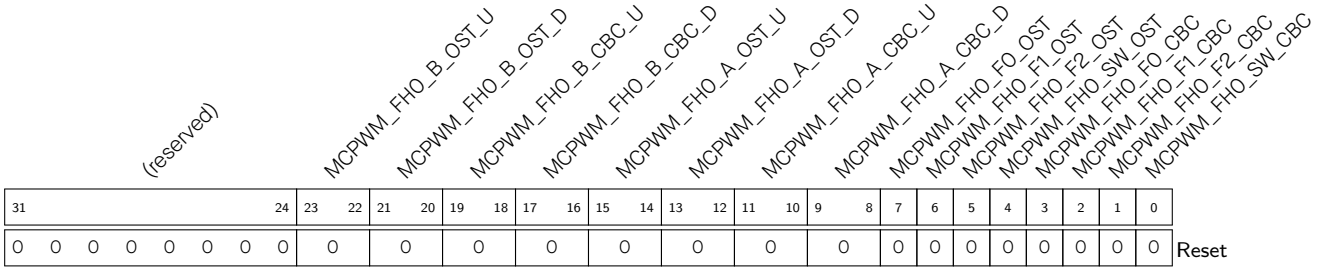
MCPWM_CARRIER0_IN_INVERT Configures whether or not to invert the input of PWM0A and PWM0B for this submodule.

0: No effect

1: Invert

(R/W)

Register 36.27. MCPWM_FH0_CFG0_REG (0x0068)



MCPWM_FH0_SW_CBC Configures whether or not to enable software force cycle-by-cycle mode action.

0: Disable

1: Enable

(R/W)

MCPWM_FH0_F2_CBC Configures whether or not fault_event2 will trigger cycle-by-cycle mode action.

0: Disable

1: Enable

(R/W)

MCPWM_FH0_F1_CBC Configures whether or not fault_event1 will trigger cycle-by-cycle mode action. See details in MCPWM_FH0_F2_CBC. (R/W)

MCPWM_FH0_F0_CBC Configures whether or not fault_event0 will trigger cycle-by-cycle mode action. See details in MCPWM_FH0_F2_CBC. (R/W)

MCPWM_FH0_SW_OST Configures whether or not to enable software force one-shot mode action. See details in MCPWM_FH0_SW_CBC. (R/W)

MCPWM_FH0_F2_OST Configures whether or not fault_event2 will trigger one-shot mode action. See details in MCPWM_FH0_F2_CBC. (R/W)

MCPWM_FH0_F1_OST Configures whether or not fault_event1 will trigger one-shot mode action. See details in MCPWM_FH0_F2_CBC. (R/W)

MCPWM_FH0_F0_OST Configures whether or not fault_event0 will trigger one-shot mode action. See details in MCPWM_FH0_F2_CBC. (R/W)

MCPWM_FH0_A_CBC_D Configures cycle-by-cycle mode action on PWM0A when fault event occurs and timer is decreasing.

0: Do nothing

1: Force low

2: Force high

3: Toggle

(R/W)

MCPWM_FH0_A_CBC_U Configures cycle-by-cycle mode action on PWM0A when fault event occurs and timer is increasing. See details in MCPWM_FH0_A_CBC_D. (R/W)

Continued on the next page...

Register 36.27. MCPWM_FH0_CFG0_REG (0x0068)

Continued from the previous page...

MCPWM_FH0_A_OST_D Configures one-shot mode action on PWM0A when fault event occurs and timer is decreasing. See details in MCPWM_FH0_A_CBC_D. (R/W)

MCPWM_FH0_A_OST_U Configures one-shot mode action on PWM0A when fault event occurs and timer is increasing. See details in MCPWM_FH0_A_CBC_D. (R/W)

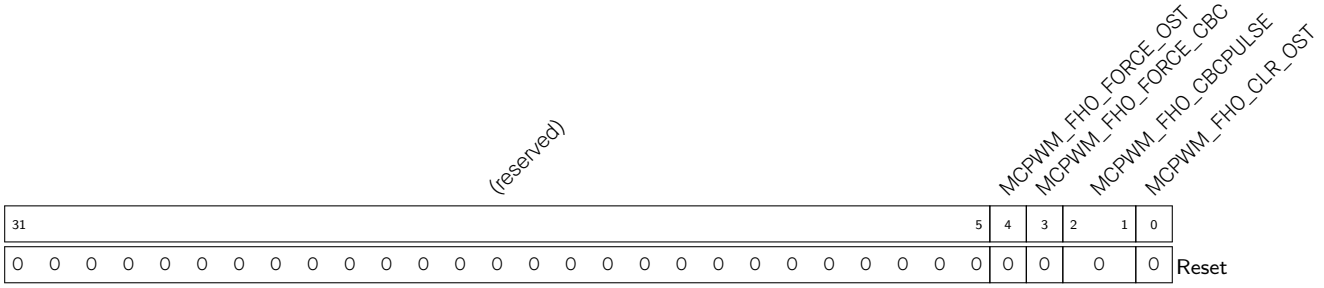
MCPWM_FH0_B_CBC_D Configures cycle-by-cycle mode action on PWM0B when fault event occurs and timer is decreasing. See details in MCPWM_FH0_A_CBC_D. (R/W)

MCPWM_FH0_B_CBC_U Configures cycle-by-cycle mode action on PWM0B when fault event occurs and timer is increasing. See details in MCPWM_FH0_A_CBC_D. (R/W)

MCPWM_FH0_B_OST_D Configures one-shot mode action on PWM0B when fault event occurs and timer is decreasing. See details in MCPWM_FH0_A_CBC_D. (R/W)

MCPWM_FH0_B_OST_U Configures one-shot mode action on PWM0B when fault event occurs and timer is increasing. See details in MCPWM_FH0_A_CBC_D. (R/W)

Register 36.28. MCPWM_FH0_CFG1_REG (0x006C)



MCPWM_FH0_CLR_OST Configures whether or not a rising edge will clear an ongoing one-shot mode action.

0: No effect

1: Clear

(R/W)

MCPWM_FH0_CBCPULSE Configures cycle-by-cycle mode action refresh moment selection.

When bit0 is set to 1: TEZ

When bit1 is set to 1: TEP

(R/W)

MCPWM_FH0_FORCE_CBC Configures whether or not to trigger a cycle-by-cycle mode action.

0: No effect

1: Trigger a cycle-by-cycle mode action

(R/W)

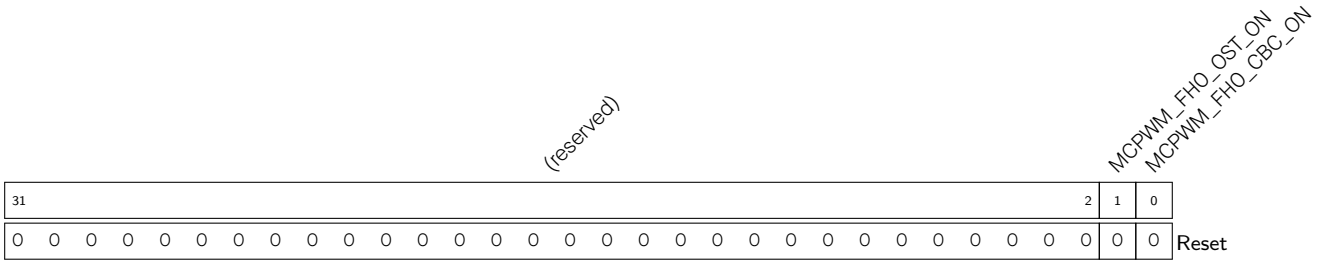
MCPWM_FH0_FORCE_OST Configures whether or not to trigger a one-shot mode action.

0: No effect

1: Trigger a one-shot mode action

(R/W)

Register 36.29. MCPWM_FH0_STATUS_REG (0x0070)



MCPWM_FH0_CBC_ON Represents whether or not a cycle-by-cycle mode action is ongoing. This field is set and reset by hardware.

0: No cycle-by-cycle mode action is ongoing

1: A cycle-by-cycle mode action is ongoing

(RO)

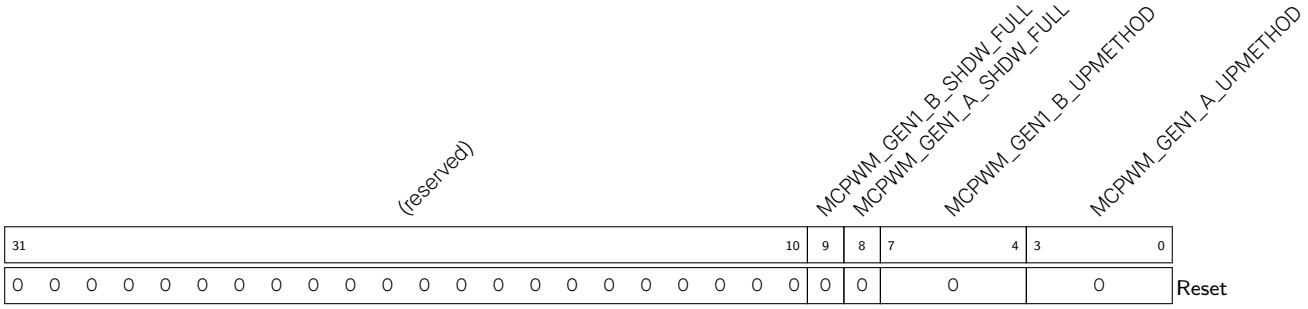
MCPWM_FH0_OST_ON Represents whether or not a one-shot mode action is ongoing. This field is set and reset by hardware.

0: No one-shot mode action is ongoing

1: A one-shot mode action is ongoing

(RO)

Register 36.30. MCPWM_GEN1_STMP_CFG_REG (0x0074)



MCPWM_GEN1_A_UPMETHOD Configures update method for PWM generator 1 time stamp A's active register.

When all bits are set to 0: immediately. When bit0 is set to 1: TEZ.

When bit1 is set to 1: TEP

When bit2 is set to 1: sync

When bit3 is set to 1: disable the update

(R/W)

MCPWM_GEN1_B_UPMETHOD Configures update method for PWM generator 1 time stamp B's active register. See details in MCPWM_GEN1_A_UPMETHOD. (R/W)

MCPWM_GEN1_A_SHDW_FULL Configures whether or not the value in the shadow register is written to the corresponding active register at a specific time. This field is set and reset by hardware.

0: Write the latest value in the shadow register to A's active register

1: Write the value to the PWM generator 1 time stamp A's shadow register, and wait to be transferred to A's active register

(R/SC/WTC)

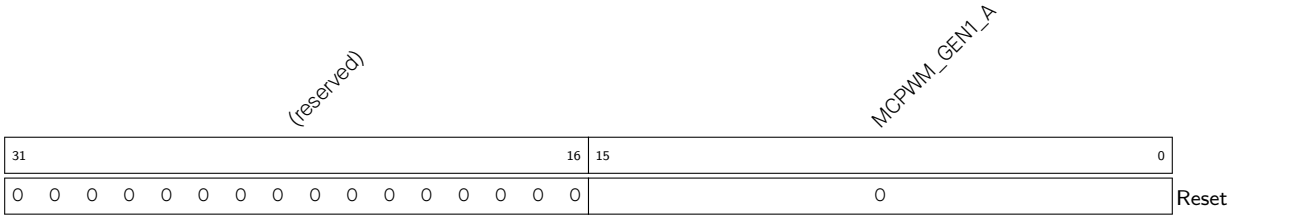
MCPWM_GEN1_B_SHDW_FULL Configures whether or not the value in the shadow register is written to the corresponding active register at a specific time. This field is set and reset by hardware.

0: Write the latest value in the shadow register to B's active register

1: Write the value to the PWM generator 1 time stamp B's shadow register, and wait to be transferred to A's active register

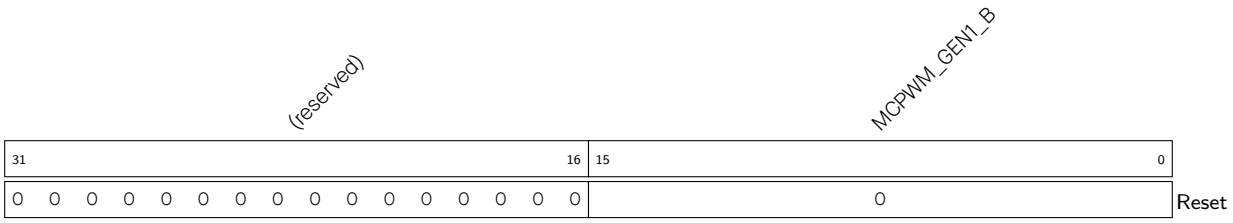
(R/SC/WTC)

Register 36.31. MCPWM_GEN1_TSTMP_A_REG (0x0078)



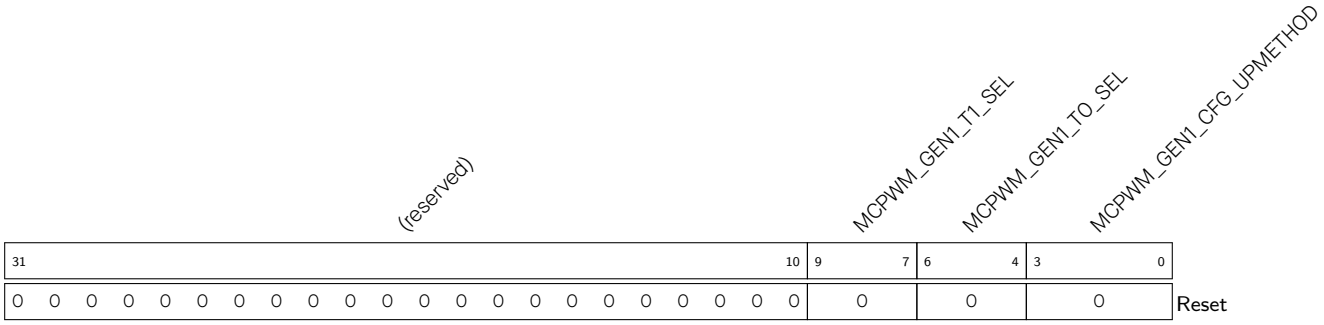
MCPWM_GEN1_A Shadow register for PWM generator 1 time stamp A. (R/W)

Register 36.32. MCPWM_GEN1_TSTMP_B_REG (0x007C)



MCPWM_GEN1_B Shadow register for PWM generator 1 time stamp B. (R/W)

Register 36.33. MCPWM_GEN1_CFG0_REG (0x0080)



MCPWM_GEN1_CFG_UPMETHOD Configures update method for PWM generator 1's active register of configuration.

When all bits are set to 0: immediately

When bit0 is set to 1: TEZ

When bit1 is set to 1: sync

When bit3 is set to 1: disable the update

(R/W)

MCPWM_GEN1_T0_SEL Configures source selection for PWM generator 1 event_t0, take effect immediately.

0: fault_event0

1: fault_event1.

2: fault_event2

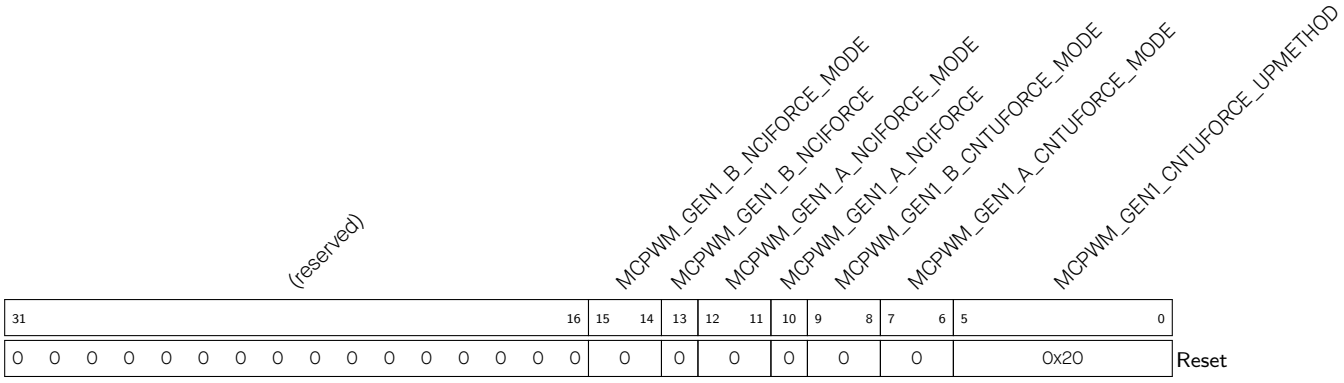
3: sync_taken

4: None

(R/W)

MCPWM_GEN1_T1_SEL Configures source selection for PWM generator 1 event_t1, take effect immediately. See details in MCPWM_GEN1_T0_SEL. (R/W)

Register 36.34. MCPWM_GEN1_FORCE_REG (0x0084)



MCPWM_GEN1_CNTUFORCE_UPMETHOD Configures updating method for continuous software force of PWM generator 1.

When all bits are set to 0: immediately

When bit0 is set to 1: TEZ

When bit1 is set to 1: TEP

When bit2 is set to 1: TEA

When bit3 is set to 1: TEB

When bit4 is set to 1: sync

When bit5 is set to 1: disable update

TEA/B here and below means an event generated when the timer's value equals to that of register A/B.

(R/W)

MCPWM_GEN1_A_CNTUFORCE_MODE Continuous software force mode for PWM1A.

0: Disabled

1: Low

2: High

3: Disabled

(R/W)

MCPWM_GEN1_B_CNTUFORCE_MODE Configures continuous software force mode for PWM1B. See details in MCPWM_GEN1_A_CNTUFORCE_MODE. (R/W)

MCPWM_GEN1_A_NCIFORCE Configures whether or not to trigger a non-continuous immediate software-force event for PWM1A

0: No effect

1: Trigger a force event

(R/W)

MCPWM_GEN1_A_NCIFORCE_MODE Configures non-continuous immediate software force mode for PWM1A.

0: Disabled

1: Low

2: High

3: Disabled

(R/W)

Continued on the next page...

Register 36.34. MCPWM_GEN1_FORCE_REG (0x0084)

Continued from the previous page...

MCPWM_GEN1_B_NCIFORCE Configures whether or not to trigger a non-continuous immediate software-force event for PWM1B.

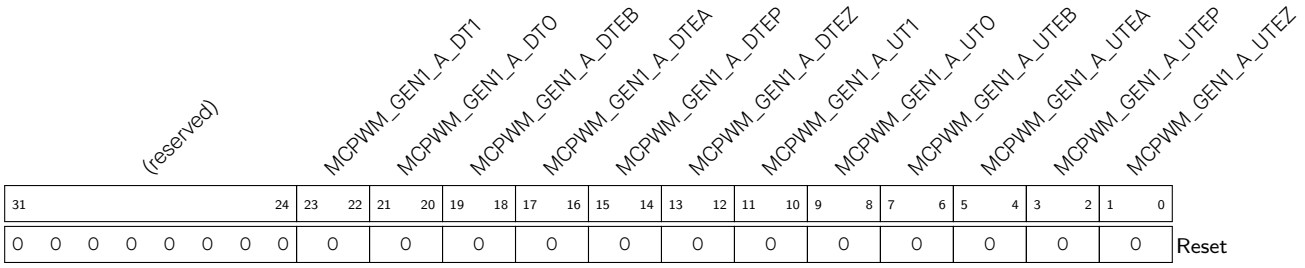
0: No effect

1: Trigger a force event

(R/W)

MCPWM_GEN1_B_NCIFORCE_MODE Configures non-continuous immediate software force mode for PWM1B. See details in MCPWM_GEN1_A_NCIFORCE_MODE. (R/W)

Register 36.35. MCPWM_GEN1_A_REG (0x0088)



MCPWM_GEN1_A_UTEZ Configures action on PWM1A triggered by event TEZ when timer increasing.

0: No change

1: Low

2: High

3: Toggle

(R/W)

MCPWM_GEN1_A_UTEP Configures action on PWM1A triggered by event TEP when timer increasing. See details in MCPWM_GEN1_A_UTEZ. (R/W)

MCPWM_GEN1_A_UTEA Configures action on PWM1A triggered by event TEA when timer increasing. See details in MCPWM_GEN1_A_UTEZ. (R/W)

MCPWM_GEN1_A_UTEB Configures action on PWM1A triggered by event TEB when timer increasing. See details in MCPWM_GEN1_A_UTEZ. (R/W)

MCPWM_GEN1_A_UT0 Configures action on PWM1A triggered by event_t0 when timer increasing. See details in MCPWM_GEN1_A_UTEZ. (R/W)

MCPWM_GEN1_A_UT1 Configures action on PWM1A triggered by event_t1 when timer increasing. See details in MCPWM_GEN1_A_UTEZ. (R/W)

MCPWM_GEN1_A_DTEZ Configures action on PWM1A triggered by event TEZ when timer decreasing. See details in MCPWM_GEN1_A_UTEZ. (R/W)

MCPWM_GEN1_A_DTEP Configures action on PWM1A triggered by event TEP when timer decreasing. See details in MCPWM_GEN1_A_UTEZ. (R/W)

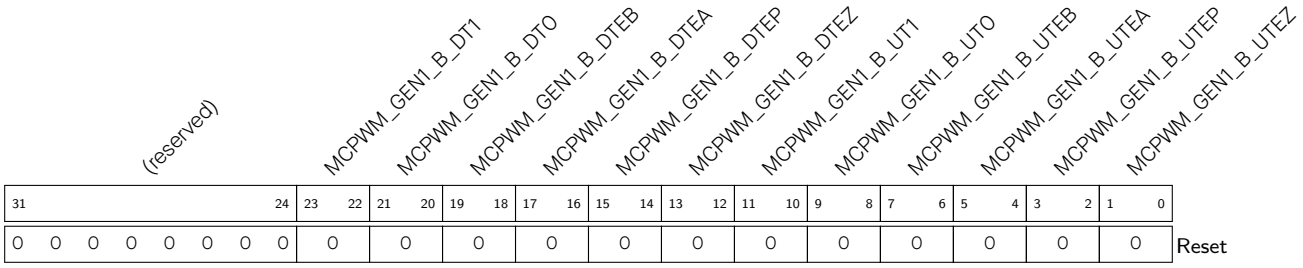
MCPWM_GEN1_A_DTEA Configures action on PWM1A triggered by event TEA when timer decreasing. See details in MCPWM_GEN1_A_UTEZ. (R/W)

MCPWM_GEN1_A_DTEB Configures action on PWM1A triggered by event TEB when timer decreasing. See details in MCPWM_GEN1_A_UTEZ. (R/W)

MCPWM_GEN1_A_DT0 Configures action on PWM1A triggered by event_t0 when timer decreasing. See details in MCPWM_GEN1_A_UTEZ. (R/W)

MCPWM_GEN1_A_DT1 Configures action on PWM1A triggered by event_t1 when timer decreasing. See details in MCPWM_GEN1_A_UTEZ. (R/W)

Register 36.36. MCPWM_GEN1_B_REG (0x008C)



MCPWM_GEN1_B_UTEZ Configures the action on PWM1B triggered by event TEZ when timer increasing.

0: No change

1: Low

2: High

3: Toggle

(R/W)

MCPWM_GEN1_B_UTEP Configures action on PWM1B triggered by event TEP when timer increasing. See details in MCPWM_GEN1_B_UTEZ. (R/W)

MCPWM_GEN1_B_UTEA Configures action on PWM1B triggered by event TEA when timer increasing. See details in MCPWM_GEN1_B_UTEZ. (R/W)

MCPWM_GEN1_B_UTEB Configures action on PWM1B triggered by event TEB when timer increasing. See details in MCPWM_GEN1_B_UTEZ. (R/W)

MCPWM_GEN1_B_UT0 Configures action on PWM1B triggered by event_t0 when timer increasing. See details in MCPWM_GEN1_B_UTEZ. (R/W)

MCPWM_GEN1_B_UT1 Configures action on PWM1B triggered by event_t1 when timer increasing. See details in MCPWM_GEN1_B_UTEZ. (R/W)

MCPWM_GEN1_B_DTEZ Configures action on PWM1B triggered by event TEZ when timer decreasing. See details in MCPWM_GEN1_B_UTEZ. (R/W)

MCPWM_GEN1_B_DTEP Configures action on PWM1B triggered by event TEP when timer decreasing. See details in MCPWM_GEN1_B_UTEZ. (R/W)

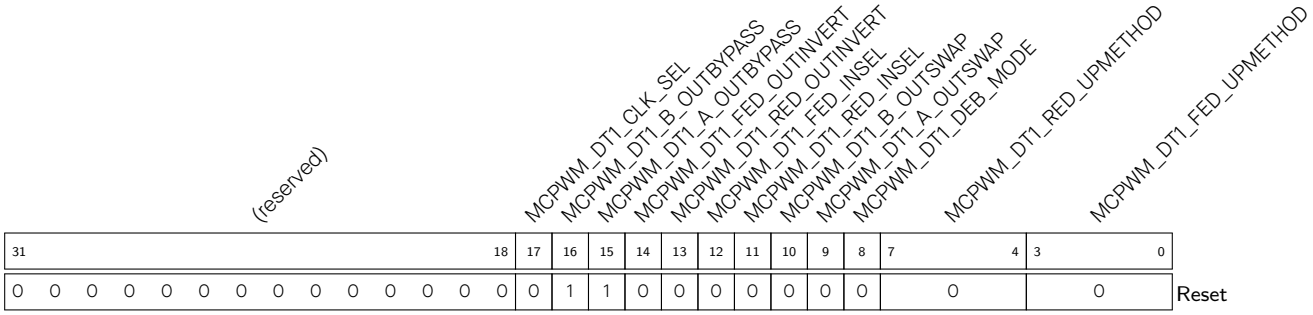
MCPWM_GEN1_B_DTEA Configures action on PWM1B triggered by event TEA when timer decreasing. See details in MCPWM_GEN1_B_UTEZ. (R/W)

MCPWM_GEN1_B_DTEB Configures action on PWM1B triggered by event TEB when timer decreasing. See details in MCPWM_GEN1_B_UTEZ. (R/W)

MCPWM_GEN1_B_DT0 Configures action on PWM1B triggered by event_t0 when timer decreasing. See details in MCPWM_GEN1_B_UTEZ. (R/W)

MCPWM_GEN1_B_DT1 Configures action on PWM1B triggered by event_t1 when timer decreasing. See details in MCPWM_GEN1_B_UTEZ. (R/W)

Register 36.37. MCPWM_DT1_CFG_REG (0x0090)



MCPWM_DT1_FED_UPMETHOD Configures update method for FED (falling edge delays) active register.

0: immediate.

When bit0 is set to 1: TEZ

When bit1 is set to 1: TEP

When bit2 is set to 1: sync

When bit3 is set to 1: disable the update

(R/W)

MCPWM_DT1_RED_UPMETHOD Configures update method for RED (rising edge delay) active register. See details in MCPWM_DT1_FED_UPMETHOD. (R/W)

MCPWM_DT1_DEB_MODE Configures the S8 switch in Table 36.3-5. For typical configurations, please refer to Table 36.3-6. (R/W)

MCPWM_DT1_A_OUTSWAP Configures the S6 switch in Table 36.3-5. For typical configurations, please refer to Table 36.3-6. (R/W)

MCPWM_DT1_B_OUTSWAP Configures the S7 switch in Table 36.3-5. For typical configurations, please refer to Table 36.3-6. (R/W)

MCPWM_DT1_RED_INSEL Configures the S4 switch in Table 36.3-5. For typical configurations, please refer to Table 36.3-6. (R/W)

MCPWM_DT1_FED_INSEL Configures the S5 switch in Table 36.3-5. For typical configurations, please refer to Table 36.3-6. (R/W)

MCPWM_DT1_RED_OUTINVERT Configures the S2 switch in Table 36.3-5. For typical configurations, please refer to Table 36.3-6. (R/W)

MCPWM_DT1_FED_OUTINVERT Configures the S3 switch in Table 36.3-5. For typical configurations, please refer to Table 36.3-6. (R/W)

MCPWM_DT1_A_OUTBYPASS Configures the S1 switch in Table 36.3-5. For typical configurations, please refer to Table 36.3-6. (R/W)

MCPWM_DT1_B_OUTBYPASS Configures the S0 switch in Table 36.3-5. For typical configurations, please refer to Table 36.3-6. (R/W)

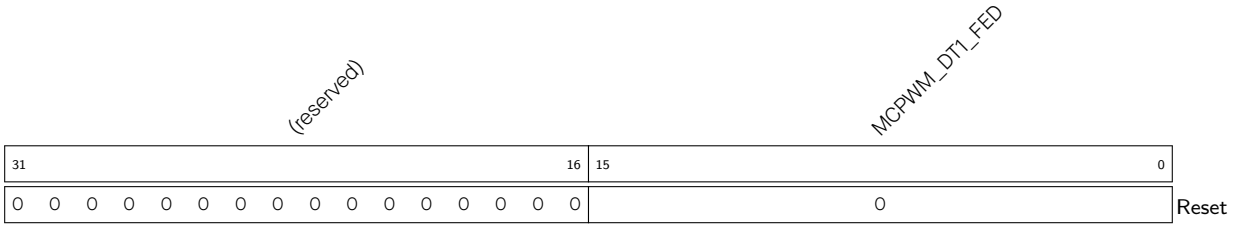
MCPWM_DT1_CLK_SEL Configures the dead time generator 1 clock selection.

0: PWM_CLK.

1: PT_CLK.

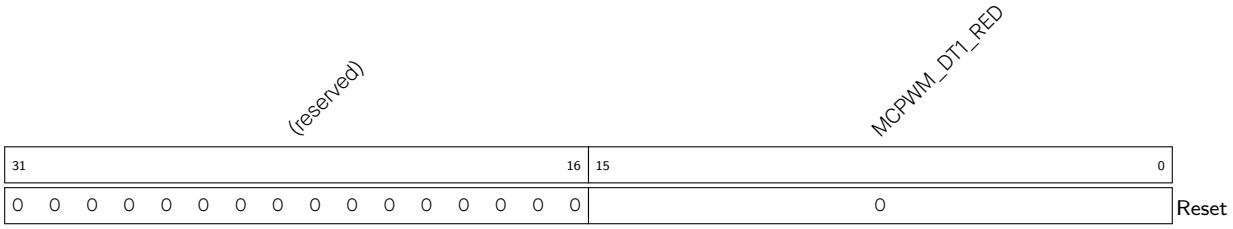
(R/W)

Register 36.38. MCPWM_DT1_FED_CFG_REG (0x0094)



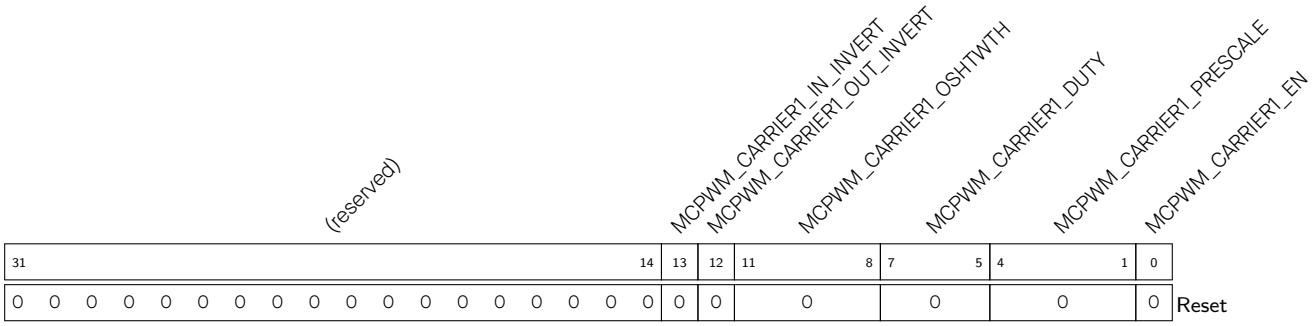
MCPWM_DT1_FED Shadow register for FED. (R/W)

Register 36.39. MCPWM_DT1_RED_CFG_REG (0x0098)



MCPWM_DT1_RED Shadow register for RED. (R/W)

Register 36.40. MCPWM_CARRIER1_CFG_REG (0x009C)



MCPWM_CARRIER1_EN Configures whether or not to enable carrier1 function.

0: Bypass carrier1

1: Enable carrier1 function

(R/W)

MCPWM_CARRIER1_PRESCALE Configures the PWM carrier1 clock (PC_CLK) prescale value. Period of PC_CLK = period of PWM_CLK * (PWM_CARRIER0_PRESCALE + 1). (R/W)

MCPWM_CARRIER1_DUTY Configures carrier duty selection. Duty = PWM_CARRIER0_DUTY/8. (R/W)

MCPWM_CARRIER1_OSHTWTH Configures width of the first pulse in number of periods of the carrier. (R/W)

MCPWM_CARRIER1_OUT_INVERT Configures whether or not to invert the output of PWM1A and PWM1B for this submodule.

0: No effect

1: Invert

(R/W)

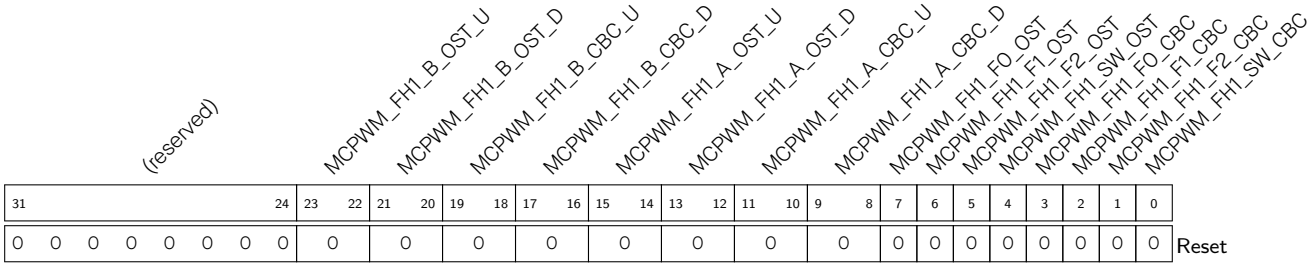
MCPWM_CARRIER1_IN_INVERT Configures whether or not to invert the input of PWM1A and PWM1B for this submodule.

0: No effect

1: Invert

(R/W)

Register 36.41. MCPWM_FH1_CFG0_REG (0x00A0)



MCPWM_FH1_SW_CBC Configures whether or not to enable software force cycle-by-cycle mode action.

0: Disable

1: Enable

(R/W)

MCPWM_FH1_F2_CBC Configures whether or not fault_event2 will trigger cycle-by-cycle mode action.

0: Disable

1: Enable

(R/W)

MCPWM_FH1_F1_CBC Configures whether or not fault_event1 will trigger cycle-by-cycle mode action. See details in MCPWM_FH1_F2_CBC. (R/W)

MCPWM_FH1_F0_CBC Configures whether or not fault_event0 will trigger cycle-by-cycle mode action. See details in MCPWM_FH1_F2_CBC. (R/W)

MCPWM_FH1_SW_OST Configures whether or not to enable register for software force one-shot mode action. See details in MCPWM_FH1_SW_CBC. (R/W)

MCPWM_FH1_F2_OST Configures whether or not fault_event2 will trigger one-shot mode action. See details in MCPWM_FH1_F2_CBC. (R/W)

MCPWM_FH1_F1_OST Configures whether or not fault_event1 will trigger one-shot mode action. See details in MCPWM_FH1_F2_CBC. (R/W)

MCPWM_FH1_F0_OST Configures whether or not fault_event0 will trigger one-shot mode action. See details in MCPWM_FH1_F2_CBC. (R/W)

MCPWM_FH1_A_CBC_D Configures cycle-by-cycle mode action on PWM1A when fault event occurs and timer is decreasing.

0: Do nothing

1: Force low

2: Force high

3: Toggle

(R/W)

MCPWM_FH1_A_CBC_U Configures cycle-by-cycle mode action on PWM1A when fault event occurs and timer is increasing. See details in MCPWM_FH1_F2_CBC. (R/W)

Continued on the next page...

Register 36.41. MCPWM_FH1_CFG0_REG (0x00A0)

Continued from the previous page...

MCPWM_FH1_A_OST_D Configures one-shot mode action on PWM1A when fault event occurs and timer is decreasing. See details in MCPWM_FH1_F2_CBC. (R/W)

MCPWM_FH1_A_OST_U Configures one-shot mode action on PWM1A when fault event occurs and timer is increasing. See details in MCPWM_FH1_F2_CBC. (R/W)

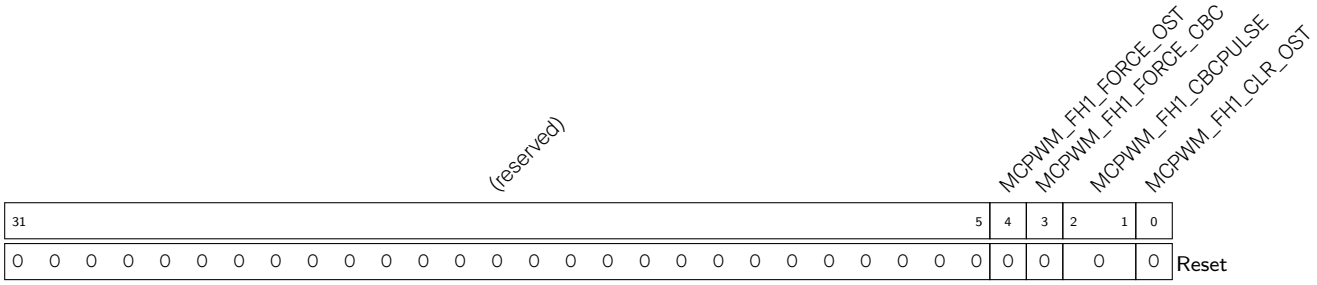
MCPWM_FH1_B_CBC_D Configures cycle-by-cycle mode action on PWM1B when fault event occurs and timer is decreasing. See details in MCPWM_FH1_F2_CBC. (R/W)

MCPWM_FH1_B_CBC_U Configures cycle-by-cycle mode action on PWM1B when fault event occurs and timer is increasing. See details in MCPWM_FH1_F2_CBC. (R/W)

MCPWM_FH1_B_OST_D Configures one-shot mode action on PWM1B when fault event occurs and timer is decreasing. See details in MCPWM_FH1_F2_CBC. (R/W)

MCPWM_FH1_B_OST_U Configures one-shot mode action on PWM1B when fault event occurs and timer is increasing. See details in MCPWM_FH1_F2_CBC. (R/W)

Register 36.42. MCPWM_FH1_CFG1_REG (0x00A4)



MCPWM_FH1_CLR_OST Configures whether or not a rising edge will clear on going one-shot mode action.

0: No effect

1: Clear

(R/W)

MCPWM_FH1_CBCPULSE Configures cycle-by-cycle mode action refresh moment selection.

When bit0 is set to 1: TEZ

When bit1 is set to 1: TEP

(R/W)

MCPWM_FH1_FORCE_CBC Configures whether or not to trigger a cycle-by-cycle mode action.

0: No effect

1: Trigger

(R/W)

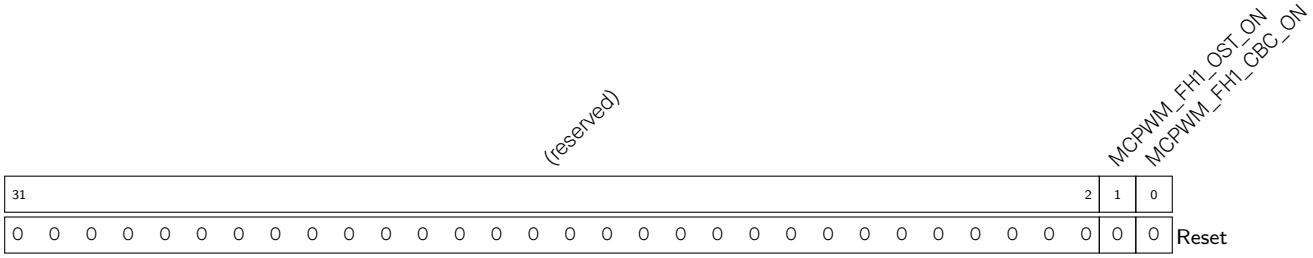
MCPWM_FH1_FORCE_OST Configures whether or not to trigger a one-shot mode action.

0: No effect

1: Trigger

(R/W)

Register 36.43. MCPWM_FH1_STATUS_REG (0x00A8)



MCPWM_FH1_CBC_ON Represents whether or not a cycle-by-cycle mode action is ongoing. This field is set and reset by hardware.

0: No cycle-by-cycle mode action is ongoing

1: A cycle-by-cycle mode action is ongoing

(RO)

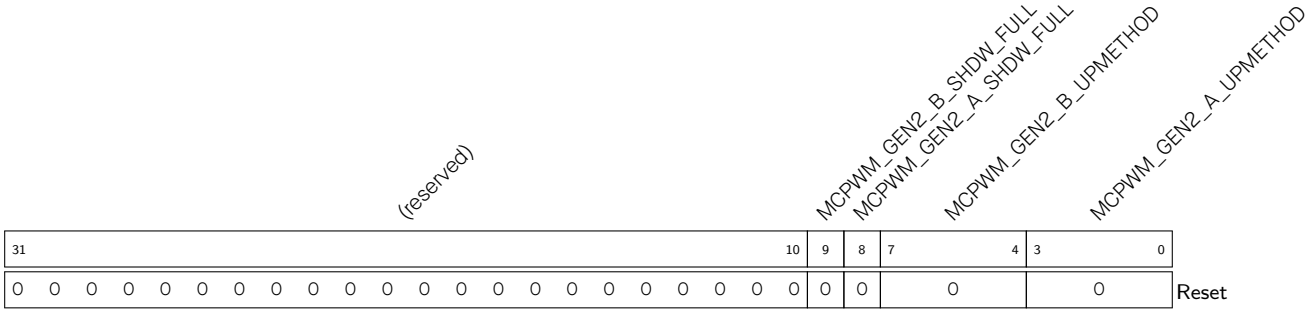
MCPWM_FH1_OST_ON Represents whether or not a one-shot mode action is ongoing. This field is set and reset by hardware.

0: No one-shot mode action is ongoing

1: A one-shot mode action is ongoing

(RO)

Register 36.44. MCPWM_GEN2_STMP_CFG_REG (0x00AC)



MCPWM_GEN2_A_UPMETHOD Configures update method for PWM generator 2 time stamp A's active register.

When all bits are set to 0: immediately.

When bit0 is set to 1: TEZ

When bit1 is set to 1: TEP

When bit2 is set to 1: sync

When bit3 is set to 1: disable the update

(R/W)

MCPWM_GEN2_B_UPMETHOD Configures update method for PWM generator 2 time stamp B's active register. See details in MCPWM_GEN2_A_UPMETHOD. (R/W)

MCPWM_GEN2_A_SHDW_FULL Configures whether or not the value in the shadow register is written to the corresponding active register at a specific time. This field is set and reset by hardware.

0: Write the latest value in the shadow register to A's active register

1: Write the value to the PWM generator 2 time stamp A's shadow register, and wait to be transferred to A's active register

(R/SC/WTC)

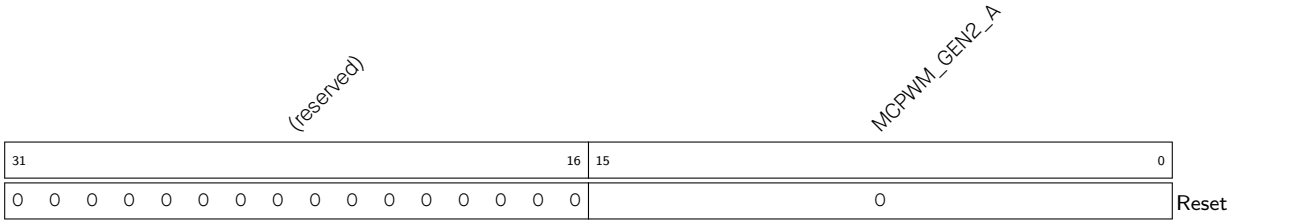
MCPWM_GEN2_B_SHDW_FULL Configures whether or not the value in the shadow register is written to the corresponding active register at a specific time. This field is set and reset by hardware.

0: Write the latest value in the shadow register to B's active register

1: Write the value to the PWM generator 2 time stamp B's shadow register, and wait to be transferred to A's active register

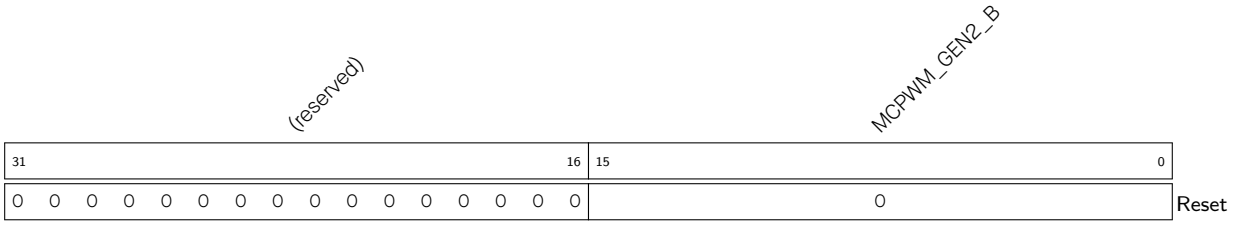
(R/SC/WTC)

Register 36.45. MCPWM_GEN2_TSTMP_A_REG (0x00B0)



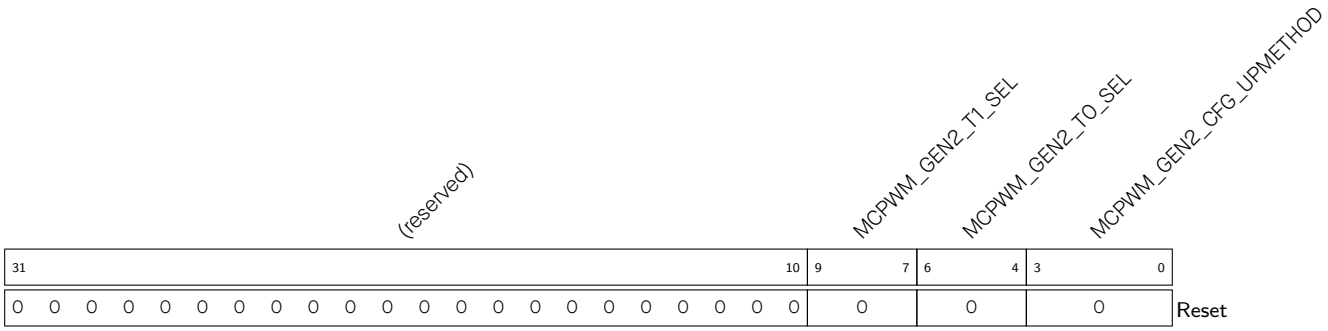
MCPWM_GEN2_A Shadow register for PWM generator 2 time stamp A. (R/W)

Register 36.46. MCPWM_GEN2_TSTMP_B_REG (0x00B4)



MCPWM_GEN2_B Shadow register for PWM generator 2 time stamp B. (R/W)

Register 36.47. MCPWM_GEN2_CFG0_REG (0x00B8)



MCPWM_GEN2_CFG_UPMETHOD Configures update method for PWM generator 2's active register.

0: Immediately

When bit0 is set to 1: TEZ

When bit1 is set to 1: sync

When bit3 is set to 1: disable the update

(R/W)

MCPWM_GEN2_T0_SEL Source selection for PWM generator 2 event_t0, take effect immediately.

0: fault_event0

1: fault_event1

2: fault_event2

3: sync_taken

4: None

(R/W)

MCPWM_GEN2_T1_SEL Source selection for PWM generator 2 event_t1, take effect immediately.

0: fault_event0

1: fault_event1

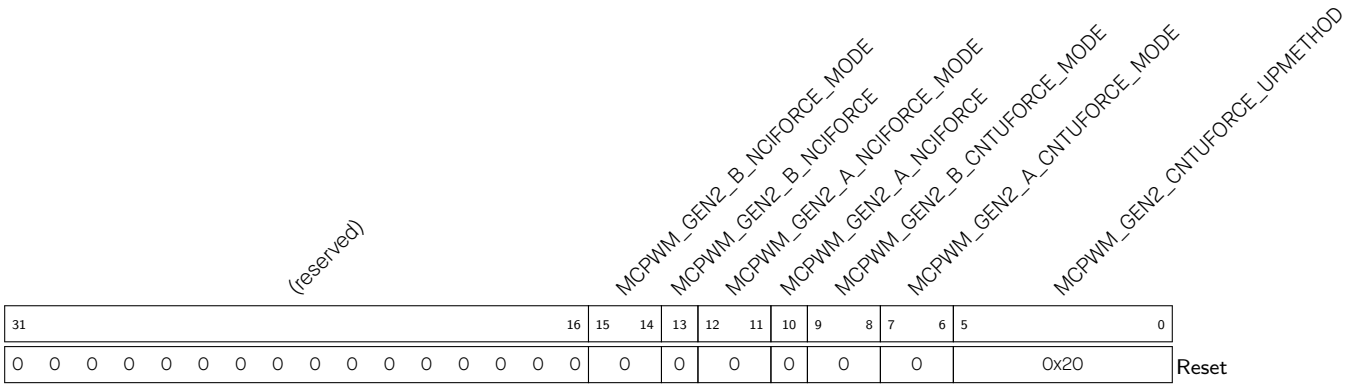
2: fault_event2

3: sync_taken

4: None

(R/W)

Register 36.48. MCPWM_GEN2_FORCE_REG (0x00BC)



MCPWM_GEN2_CNTUFORCE_UPMETHOD Configures updating method for continuous software force of PWM generator 2. When all bits are set to 0: Immediately.

When bit0 is set to 1: TEZ

When bit1 is set to 1: TEP

When bit2 is set to 1: TEA

When bit3 is set to 1: TEB

When bit4 is set to 1: Sync

When bit5 is set to 1: Disable update

TEA/B here and below means an event generated when the timer's value equals to that of register A/B.

(R/W)

MCPWM_GEN2_A_CNTUFORCE_MODE Configures continuous software force mode for PWM2A.

0: Disabled

1: Low

2: High

3: Disabled

(R/W)

MCPWM_GEN2_B_CNTUFORCE_MODE Configures continuous software force mode for PWM2B.

0: Disabled

1: Low

2: High

3: Disabled

(R/W)

MCPWM_GEN2_A_NCIFORCE Configures whether or not to trigger a non-continuous immediate software-force event for PWM2A.

0: No effect

1: Trigger a force event

(R/W)

Continued on the next page...

Register 36.48. MCPWM_GEN2_FORCE_REG (0x00BC)

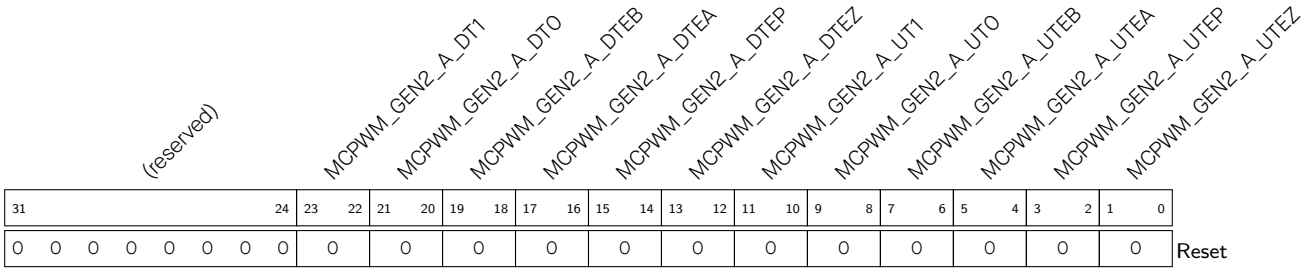
Continued from the previous page...

MCPWM_GEN2_A_NCIFORCE_MODE Configures non-continuous immediate software force mode for PWM2A.
0: Disabled
1: Low
2: High
3: Disabled
(R/W)

MCPWM_GEN2_B_NCIFORCE Configures whether or not to trigger a non-continuous immediate software-force event for PWM2B.
0: No effect
1: Trigger a force event
(R/W)

MCPWM_GEN2_B_NCIFORCE_MODE Configures non-continuous immediate software force mode for PWM2B. See details in MCPWM_GEN2_A_NCIFORCE_MODE. (R/W)

Register 36.49. MCPWM_GEN2_A_REG (0x00C0)



MCPWM_GEN2_A_UTEZ Action on PWM2A triggered by event TEZ when timer increasing.

0: No change

1: Low.

2: High.

3: Toggle

(R/W)

MCPWM_GEN2_A_UTEP Action on PWM2A triggered by event TEP when timer increasing. (R/W)

MCPWM_GEN2_A_UTEA Action on PWM2A triggered by event TEA when timer increasing. (R/W)

MCPWM_GEN2_A_UTEB Action on PWM2A triggered by event TEB when timer increasing. (R/W)

MCPWM_GEN2_A_UT0 Action on PWM2A triggered by event_t0 when timer increasing. (R/W)

MCPWM_GEN2_A_UT1 Action on PWM2A triggered by event_t1 when timer increasing. (R/W)

MCPWM_GEN2_A_DTEZ Action on PWM2A triggered by event TEZ when timer decreasing. (R/W)

MCPWM_GEN2_A_DTEP Action on PWM2A triggered by event TEP when timer decreasing. (R/W)

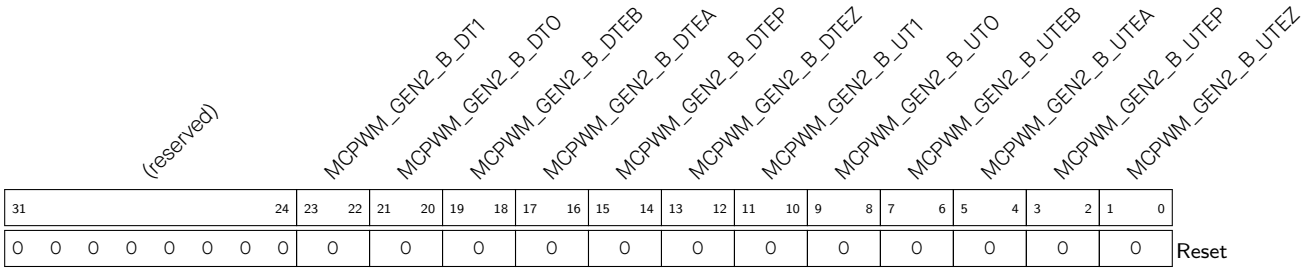
MCPWM_GEN2_A_DTEA Action on PWM2A triggered by event TEA when timer decreasing. (R/W)

MCPWM_GEN2_A_DTEB Action on PWM2A triggered by event TEB when timer decreasing. (R/W)

MCPWM_GEN2_A_DT0 Action on PWM2A triggered by event_t0 when timer decreasing. (R/W)

MCPWM_GEN2_A_DT1 Action on PWM2A triggered by event_t1 when timer decreasing. (R/W)

Register 36.50. MCPWM_GEN2_B_REG (0x00C4)



MCPWM_GEN2_B_UTEZ Action on PWM2B triggered by event TEZ when timer increasing.

0: No change

1: Low

2: High

3: Toggle

(R/W)

MCPWM_GEN2_B_UTEP Action on PWM2B triggered by event TEP when timer increasing. (R/W)

MCPWM_GEN2_B_UTEA Action on PWM2B triggered by event TEA when timer increasing. (R/W)

MCPWM_GEN2_B_UTEB Action on PWM2B triggered by event TEB when timer increasing. (R/W)

MCPWM_GEN2_B_UT0 Action on PWM2B triggered by event_t0 when timer increasing. (R/W)

MCPWM_GEN2_B_UT1 Action on PWM2B triggered by event_t1 when timer increasing. (R/W)

MCPWM_GEN2_B_DTEZ Action on PWM2B triggered by event TEZ when timer decreasing. (R/W)

MCPWM_GEN2_B_DTEP Action on PWM2B triggered by event TEP when timer decreasing. (R/W)

MCPWM_GEN2_B_DTEA Action on PWM2B triggered by event TEA when timer decreasing. (R/W)

MCPWM_GEN2_B_DTEB Action on PWM2B triggered by event TEB when timer decreasing. (R/W)

MCPWM_GEN2_B_DT0 Action on PWM2B triggered by event_t0 when timer decreasing. (R/W)

MCPWM_GEN2_B_DT1 Action on PWM2B triggered by event_t1 when timer decreasing. (R/W)

Register 36.51. MCPWM_DT2_CFG_REG (0x00C8)

MCPWM_DT2_FED_UPMETHOD Configures update method for FED (falling edge delay) active register.

0: Immediate.

When bit0 is set to 1: TEZ

When bit1 is set to 1: TEP

When bit2 is set to 1: sync

When bit3 is set to 1: disable the update

(R/W)

MCPWM_DT2_RED_UPMETHOD Configures update method for RED (rising edge delay) active register. See details in MCPWM_DT2_FED_UPMETHOD. (R/W)

MCPWM_DT2_DEB_MODE Configures the S8 switch in Table 36.3-5. For typical configurations, please refer to Table 36.3-6. (R/W)

MCPWM_DT2_A_OUTSWAP Configures the S6 switch in Table 36.3-5. For typical configurations, please refer to Table 36.3-6. (R/W)

MCPWM_DT2_B_OUTSWAP Configures the S7 switch in Table 36.3-5. For typical configurations, please refer to Table 36.3-6. (R/W)

MCPWM_DT2_RED_INSEL Configures the S4 switch in Table 36.3-5. For typical configurations, please refer to Table 36.3-6. (R/W)

MCPWM_DT2_FED_INSEL Configures the S5 switch in Table 36.3-5. For typical configurations, please refer to Table 36.3-6. (R/W)

MCPWM_DT2_RED_OUTINVERT Configures the S2 switch in Table 36.3-5. For typical configurations, please refer to Table 36.3-6. (R/W)

MCPWM_DT2_FED_OUTINVERT Configures the S3 switch in Table 36.3-5. For typical configurations, please refer to Table 36.3-6. (R/W)

MCPWM_DT2_A_OUTBYPASS Configures the S1 switch in Table 36.3-5. For typical configurations, please refer to Table 36.3-6. (R/W)

MCPWM_DT2_B_OUTBYPASS Configures the S0 switch in Table 36.3-5. For typical configurations, please refer to Table 36.3-6. (R/W)

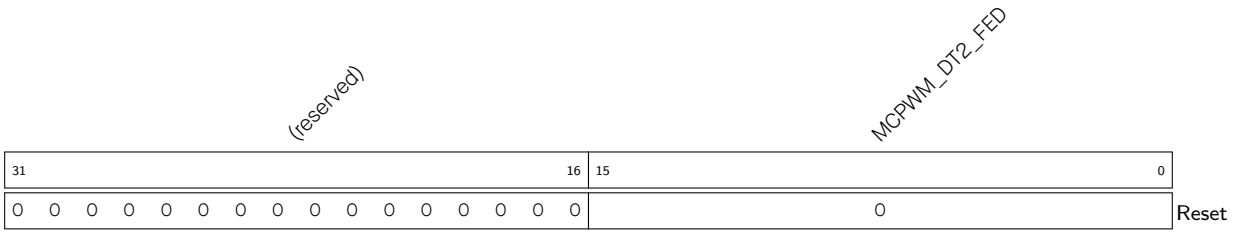
MCPWM_DT2_CLK_SEL Configures dead time generator 2 clock selection.

0: PWM_CLK

1: PT_CLK

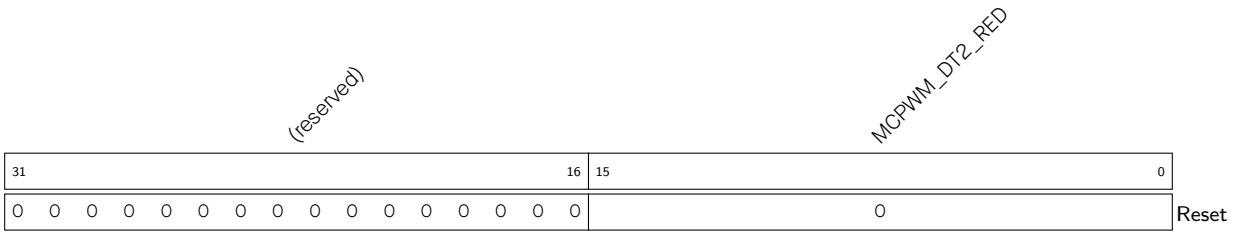
(R/W)

Register 36.52. MCPWM_DT2_FED_CFG_REG (0x00CC)



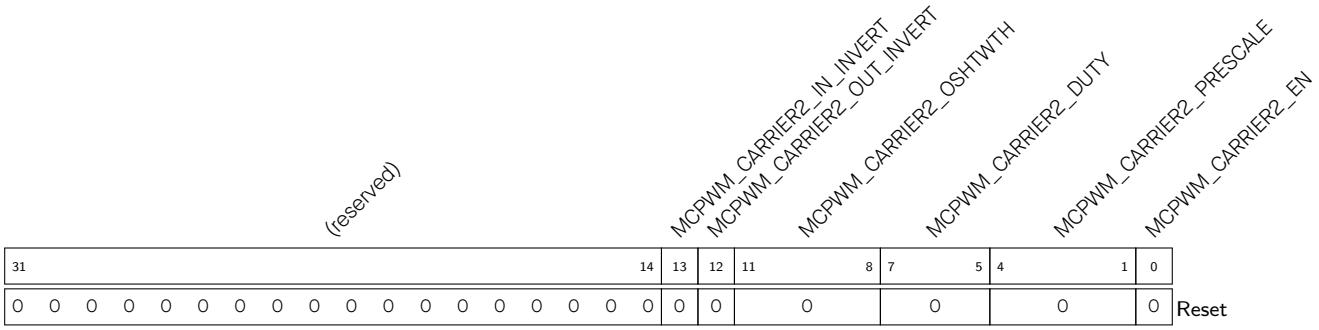
MCPWM_DT2_FED Shadow register for FED. (R/W)

Register 36.53. MCPWM_DT2_RED_CFG_REG (0x00D0)



MCPWM_DT2_RED Shadow register for RED. (R/W)

Register 36.54. MCPWM_CARRIER2_CFG_REG (0x00D4)



MCPWM_CARRIER2_EN Configures whether or not to enable the carrier2 function.

0: Bypass carrier2

1: Enable

(R/W)

MCPWM_CARRIER2_PRESCALE Configures the PWM carrier2 clock (PC_CLK) prescale value. Period of PC_CLK = period of PWM_CLK * (PWM_CARRIER0_PRESCALE + 1). (R/W)

MCPWM_CARRIER2_DUTY Configures the carrier duty selection. Duty = PWM_CARRIER0_DUTY/8. (R/W)

MCPWM_CARRIER2_OSHTWTH Configures the width of the first pulse in number of periods of the carrier. (R/W)

MCPWM_CARRIER2_OUT_INVERT Configures whether or not to invert the output of PWM2A and PWM2B for this submodule.

0: No effect

1: Invert

(R/W)

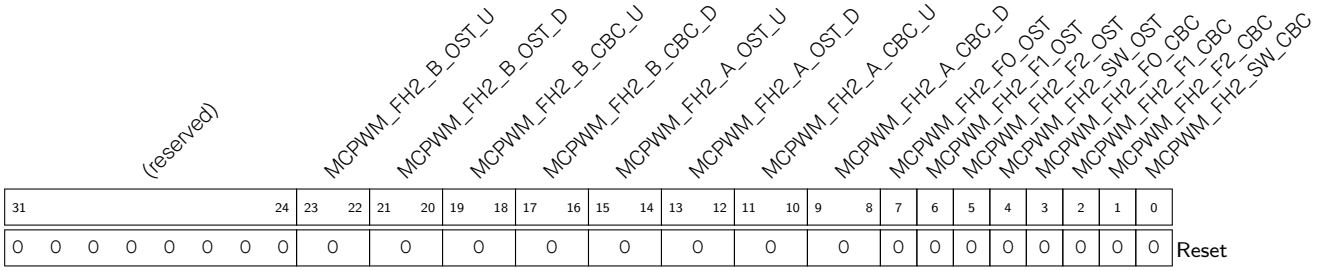
MCPWM_CARRIER2_IN_INVERT Configures whether or not to invert the input of PWM2A and PWM2B for this submodule.

0: No effect

1: Invert

(R/W)

Register 36.55. MCPWM_FH2_CFG0_REG (0x00D8)



MCPWM_FH2_SW_CBC Configures whether or not to enable software force cycle-by-cycle mode action.

0: Disable

1: Enable

(R/W)

MCPWM_FH2_F2_CBC Configures whether or not fault_event2 will trigger cycle-by-cycle mode action.

0: No effect

1: Trigger

(R/W)

MCPWM_FH2_F1_CBC Configures whether or not fault_event1 will trigger cycle-by-cycle mode action.

0: No effect

1: Trigger

(R/W)

MCPWM_FH2_F0_CBC Configures whether or not fault_event0 will trigger cycle-by-cycle mode action.

0: No effect

1: Trigger

(R/W)

MCPWM_FH2_SW_OST Configures whether or not to enable software force one-shot mode action.

0: Disable

1: Enable

(R/W)

MCPWM_FH2_F2_OST Configures whether or not fault_event2 will trigger one-shot mode action.

0: No effect

1: Trigger

(R/W) (R/W)

MCPWM_FH2_F1_OST Configures whether or not fault_event1 will trigger one-shot mode action.

0: No effect

1: Trigger

(R/W) (R/W)

Continued on the next page...

Register 36.55. MCPWM_FH2_CFG0_REG (0x00D8)

Continued from the previous page...

MCPWM_FH2_F0_OST Configures whether or not fault_event0 will trigger one-shot mode action.
0: No effect
1: Trigger
(R/W) (R/W)

MCPWM_FH2_A_CBC_D Configures cycle-by-cycle mode action on PWM2A when fault event occurs and timer is decreasing.
0: Do nothing.
1: Force low.
2: Force high.
3: Toggle
(R/W)

MCPWM_FH2_A_CBC_U Configures cycle-by-cycle mode action on PWM2A when fault event occurs and the timer is increasing. See details in MCPWM_FH2_A_CBC_D. (R/W)

MCPWM_FH2_A_OST_D Configures one-shot mode action on PWM2A when fault event occurs and timer is decreasing. See details in MCPWM_FH2_A_CBC_D. (R/W)

MCPWM_FH2_A_OST_U Configures one-shot mode action on PWM2A when fault event occurs and timer is increasing. See details in MCPWM_FH2_A_CBC_D. (R/W)

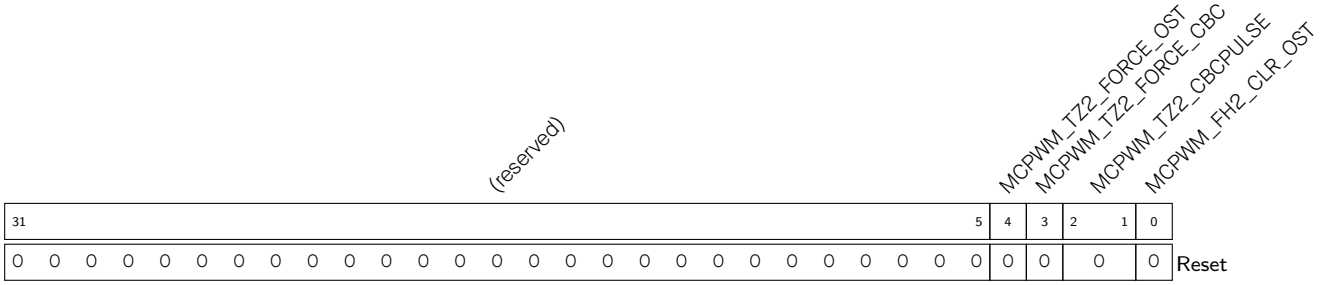
MCPWM_FH2_B_CBC_D Configures cycle-by-cycle mode action on PWM2B when fault event occurs and timer is decreasing. See details in MCPWM_FH2_A_CBC_D. (R/W)

MCPWM_FH2_B_CBC_U Configures cycle-by-cycle mode action on PWM2B when fault event occurs and timer is increasing. See details in MCPWM_FH2_A_CBC_D. (R/W)

MCPWM_FH2_B_OST_D Configures one-shot mode action on PWM2B when fault event occurs and timer is decreasing. See details in MCPWM_FH2_A_CBC_D. (R/W)

MCPWM_FH2_B_OST_U Configures one-shot mode action on PWM2B when fault event occurs and timer is increasing. See details in MCPWM_FH2_A_CBC_D. (R/W)

Register 36.56. MCPWM_FH2_CFG1_REG (0x00DC)



MCPWM_FH2_CLR_OST Configures whether or not a rising edge will clear an ongoing one-shot mode action.

0: Not clear

1: Clear

(R/W)

MCPWM_TZ2_CBCPULSE Configures cycle-by-cycle mode action refresh moment selection.

When bit0 is set to 1: TEZ

When bit1 is set to 1: TEP

(R/W)

MCPWM_TZ2_FORCE_CBC Configures whether or not to trigger a cycle-by-cycle mode action.

0: No effect

1: Trigger a cycle-by-cycle mode action

(R/W)

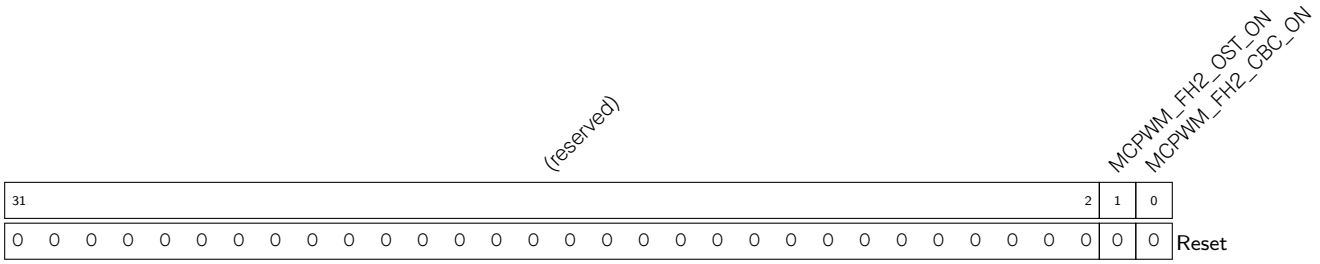
MCPWM_TZ2_FORCE_OST Configures whether or not to trigger a one-shot mode action.

0: No effect

1: Trigger a one-shot mode action

(R/W)

Register 36.57. MCPWM_FH2_STATUS_REG (0x00E0)



MCPWM_FH2_CBC_ON Represents whether or not a cycle-by-cycle mode action is ongoing. This field is set and reset by hardware.

0: No cycle-by-cycle mode action is ongoing

1: A cycle-by-cycle mode action is ongoing

(RO)

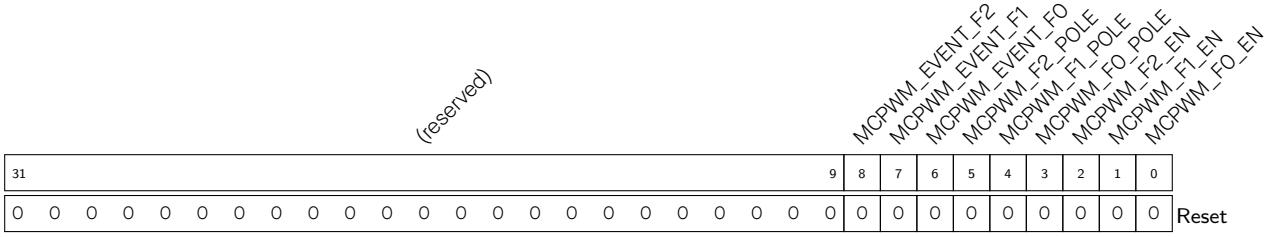
MCPWM_FH2_OST_ON Represents whether or not a one-shot mode action is ongoing. This field is set and reset by hardware.

0: No one-shot mode action is ongoing

1: A one-shot mode action is ongoing

(RO)

Register 36.58. MCPWM_FAULT_DETECT_REG (0x00E4)



MCPWM_F0_EN Configures whether or not to enable fault_event0 generation.

0: No effect

1: Enable

(R/W)

MCPWM_F1_EN Configures whether or not to enable fault_event1 generation.

0: No effect

1: Enable

(R/W)

MCPWM_F2_EN Configures whether or not to enable fault_event2 generation.

0: No effect

1: Enable

(R/W)

MCPWM_F0_POLE Configures fault_event0 trigger polarity on FAULT0 source from GPIO matrix.

0: Level low

1: Level high

(R/W)

MCPWM_F1_POLE Configures fault_event1 trigger polarity on FAULT1 source from GPIO matrix.

0: Level low

1: Level high

(R/W)

MCPWM_F2_POLE Configures fault_event2 trigger polarity on FAULT2 source from GPIO matrix.

0: Level low

1: Level high

(R/W)

MCPWM_EVENT_F0 Represents the status of fault_event0. This field is set and reset by hardware.

0: fault_event0 is not ongoing

1: fault_event0 is ongoing

(RO)

MCPWM_EVENT_F1 Represents the status of fault_event1. This field is set and reset by hardware.

0: fault_event1 is not ongoing

1: fault_event1 is ongoing

(RO)

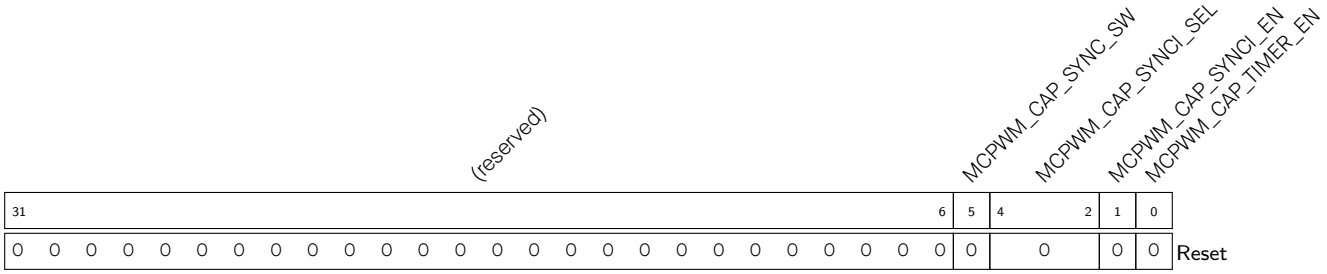
MCPWM_EVENT_F2 Represents the status of fault_event2. This field is set and reset by hardware.

0: fault_event2 is not ongoing

1: fault_event2 is ongoing

(RO)

Register 36.59. MCPWM_CAP_TIMER_CFG_REG (0x00E8)



MCPWM_CAP_TIMER_EN Configures whether or not to enable capture timer incrementing under APB_CLK.

0: No effect

1: Enable

(R/W)

MCPWM_CAP_SYNCI_EN Configures whether or not to enable capture timer sync.

0: No effect

1: Enable

(R/W)

MCPWM_CAP_SYNCI_SEL Configures the capture module sync input selection.

0: None

1: timer0 sync out

2: timer1 sync out

3: timer2 sync out

4: SYNC0 from GPIO matrix

5: SYNC1 from GPIO matrix

6: SYNC2 from GPIO matrix

(R/W)

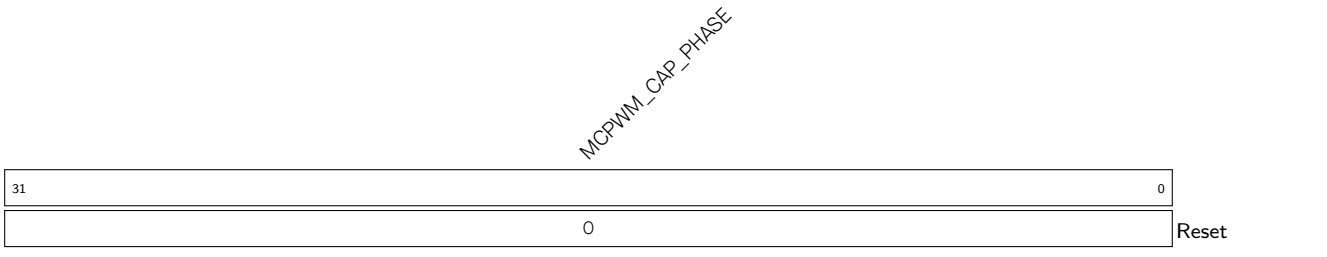
MCPWM_CAP_SYNC_SW When MCPWM_CAP_SYNCI_EN is set to 1, configures whether or not to trigger a capture timer sync so that capture timer is loaded with value in phase register.

0: No effect

1: Trigger a capture timer sync

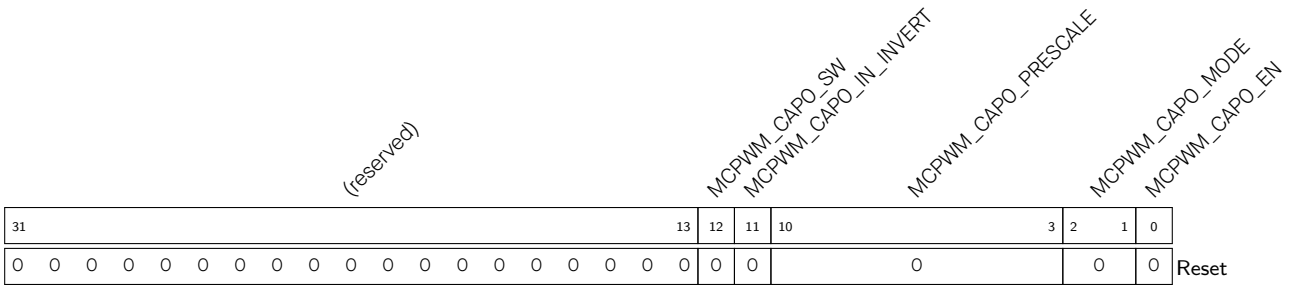
(WT)

Register 36.60. MCPWM_CAP_TIMER_PHASE_REG (0x00EC)



MCPWM_CAP_PHASE Configures the phase value for capture timer sync operation. (R/W)

Register 36.61. MCPWM_CAP_CH0_CFG_REG (0x00F0)



MCPWM_CAP0_EN Configures whether or not to enable capture on channel 0.

0: Not enable

1: Enable

(R/W)

MCPWM_CAP0_MODE Configures the edge of capture on channel 0 after prescaling.

When bit0 is set to 1: enable capture on the falling edge.

When bit1 is set to 1: enable capture on the rising edge.

(R/W)

MCPWM_CAP0_PRESCALE Configures the prescale value on the rising edge of CAP0. Prescale value = PWM_CAP0_PRESCALE + 1. (R/W)

MCPWM_CAP0_IN_INVERT Configures whether or not to invert the CAP0 from GPIO matrix before prescale.

0: No effect

1: Invert

(R/W)

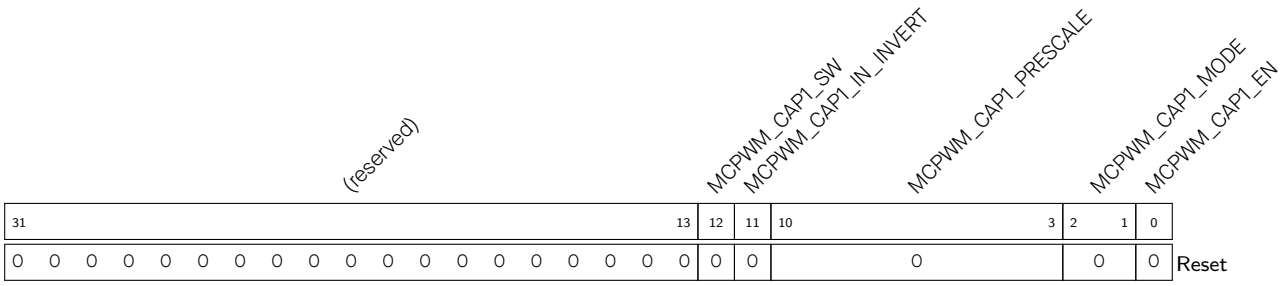
MCPWM_CAP0_SW Configures whether or not to trigger a software-forced capture on channel 0.

0: Not trigger

1: Trigger

(WT)

Register 36.62. MCPWM_CAP_CH1_CFG_REG (0x00F4)



MCPWM_CAP1_EN Configures whether or not to enable capture on channel 1.

0: Not enable

1: Enable

(R/W)

MCPWM_CAP1_MODE Configures the edge of capture on channel 1 after prescaling.

When bit0 is set to 1: enable capture on the falling edge.

When bit1 is set to 1: enable capture on the rising edge.

(R/W)

MCPWM_CAP1_PRESCALE Configures the value of prescaling on the rising edge of CAP1. Prescale value = PWM_CAP1_PRESCALE + 1. (R/W)

MCPWM_CAP1_IN_INVERT Configures whether or not to invert the CAP1 from GPIO matrix before prescale.

0: No effect

1: Invert

(R/W)

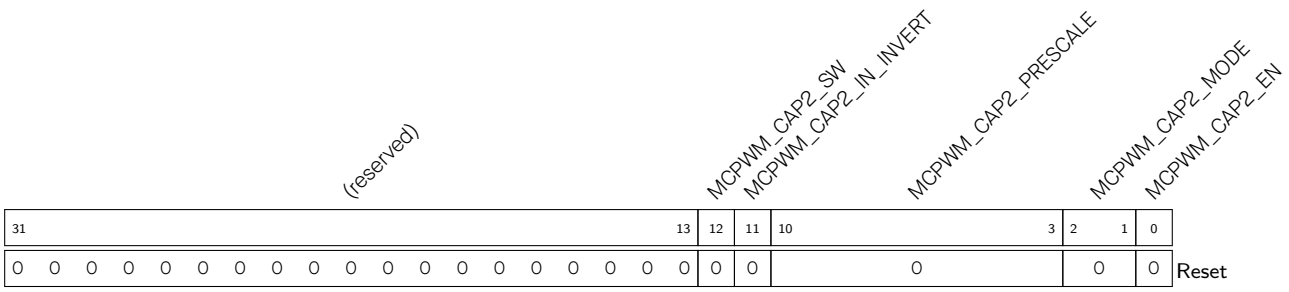
MCPWM_CAP1_SW Configures whether or not to trigger a software forced capture on channel 1.

0: Not trigger

1: Trigger

(WT)

Register 36.63. MCPWM_CAP_CH2_CFG_REG (0x00F8)



MCPWM_CAP2_EN Configures whether or not to enable capture on channel 2.

0: Not enable

1: Enable

(R/W)

MCPWM_CAP2_MODE Configures the edge of capture on channel 2 after prescaling.

When bit0 is set to 1: enable capture on the falling edge.

When bit1 is set to 1: enable capture on the rising edge.

(R/W)

MCPWM_CAP2_PRESCALE Configures the value of prescaling on the rising edge of CAP2. Prescale value = PWM_CAP2_PRESCALE + 1. (R/W)

MCPWM_CAP2_IN_INVERT Configures whether or not to invert the CAP2 from GPIO matrix before prescale.

0: No effect

1: Invert

(R/W)

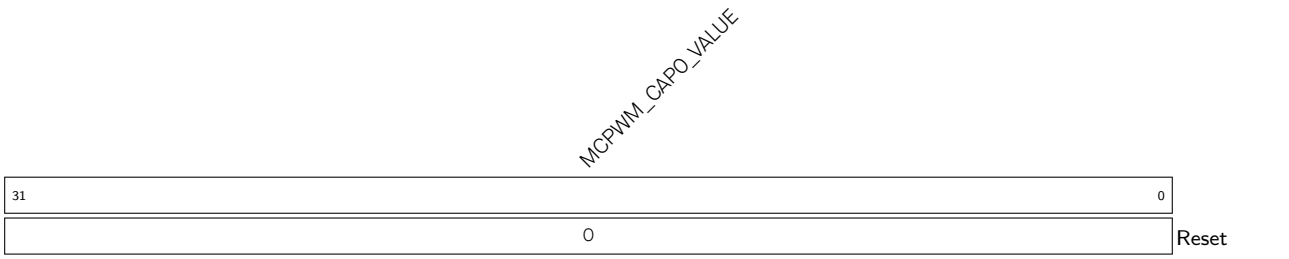
MCPWM_CAP2_SW Configures whether or not to trigger a software forced capture on channel 2.

0: Not trigger

1: Trigger

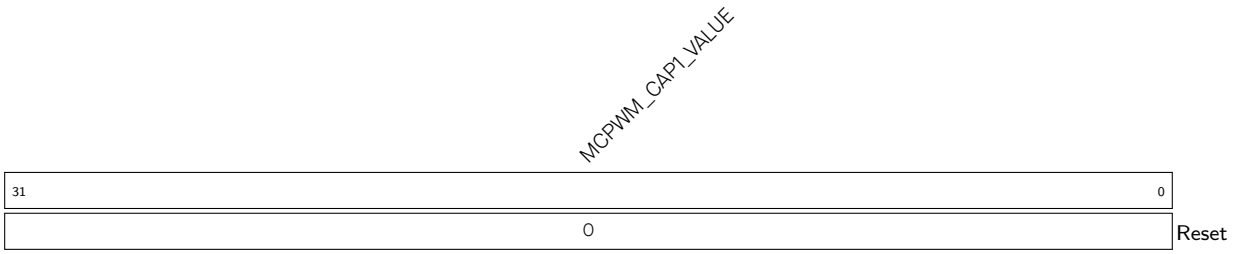
(WT)

Register 36.64. MCPWM_CAP_CH0_REG (0x00FC)



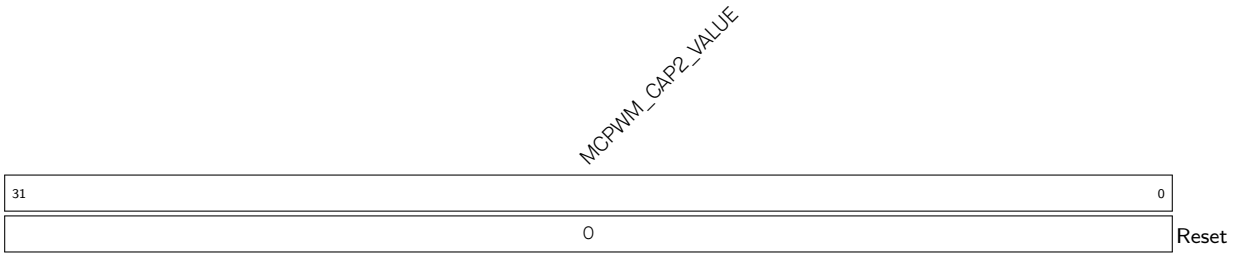
MCPWM_CAP0_VALUE Represents the value of the last capture on channel 0. (RO)

Register 36.65. MCPWM_CAP_CH1_REG (0x0100)



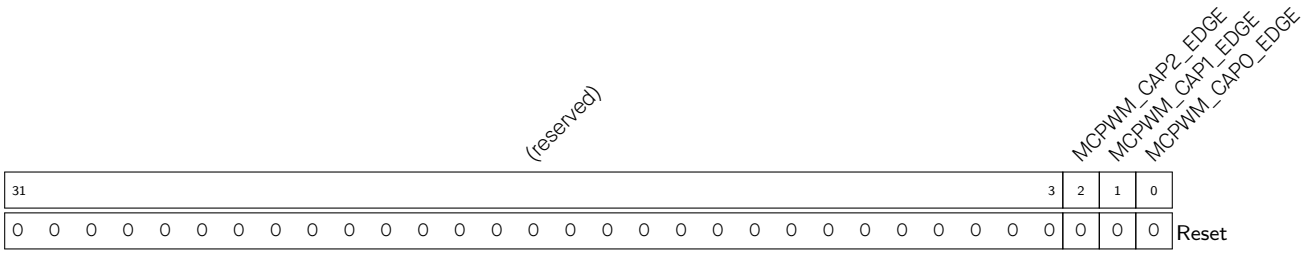
MCPWM_CAP1_VALUE Represents the value of the last capture on channel 1. (RO)

Register 36.66. MCPWM_CAP_CH2_REG (0x0104)



MCPWM_CAP2_VALUE Represents the value of the last capture on channel 2. (RO)

Register 36.67. MCPWM_CAP_STATUS_REG (0x0108)



MCPWM_CAP0_EDGE Represents the edge of the last capture trigger on channel 0.

0: Rising edge

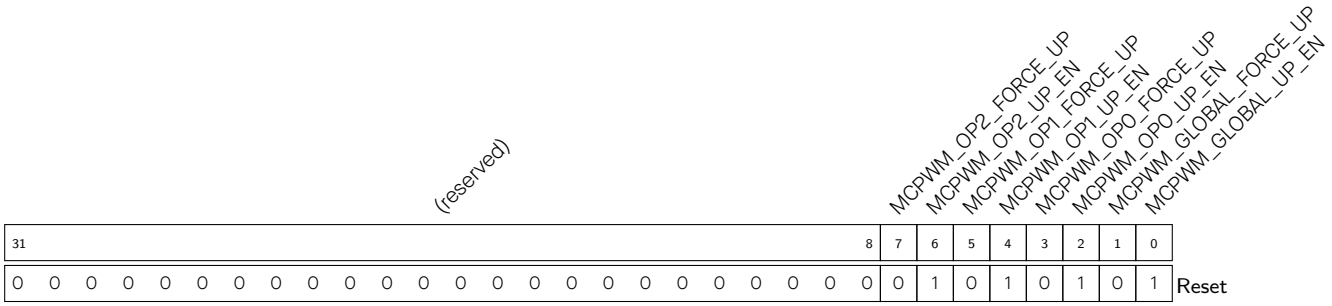
1: Falling edge

(RO)

MCPWM_CAP1_EDGE Represents the edge of the last capture trigger on channel 1. See details in MCPWM_CAP0_EDGE. (RO)

MCPWM_CAP2_EDGE Represents the edge of the last capture trigger on channel 2. See details in MCPWM_CAP0_EDGE. (RO)

Register 36.68. MCPWM_UPDATE_CFG_REG (0x010C)



MCPWM_GLOBAL_UP_EN Configures whether to globally update all active registers.

0: No effect

1: Update all active registers globally

(R/W)

MCPWM_GLOBAL_FORCE_UP Configures whether or not to trigger a forced update of all active registers globally.

0: No effect

1: Trigger a forced update

(R/W)

MCPWM_OP0_UP_EN Configures whether or not to update active registers in PWM operator 0 when MCPWM_GLOBAL_UP_EN is set to 1.

0: No effect

1: Update active registers in PWM operator 0

(R/W)

MCPWM_OP0_FORCE_UP Configures whether or not to trigger a forced update of active registers in PWM operator 0.

0: No effect

1: Trigger a forced update

(R/W)

MCPWM_OP1_UP_EN Configures whether or not to update active registers in PWM operator 1 when MCPWM_GLOBAL_UP_EN is set to 1.

0: No effect

1: Update active registers in PWM operator 1

(R/W)

Continued on the next page...

Register 36.68. MCPWM_UPDATE_CFG_REG (0x010C)

Continued from the previous page...

MCPWM_OP1_FORCE_UP Configures whether or not to trigger a forced update of active registers in PWM operator 1.

0: No effect

1: Trigger a forced update

(R/W)

MCPWM_OP2_UP_EN Configures whether or not to update active registers in PWM operator 2 when MCPWM_GLOBAL_UP_EN is set to 1.

0: No effect

1: Update active registers in PWM operator 2

(R/W)

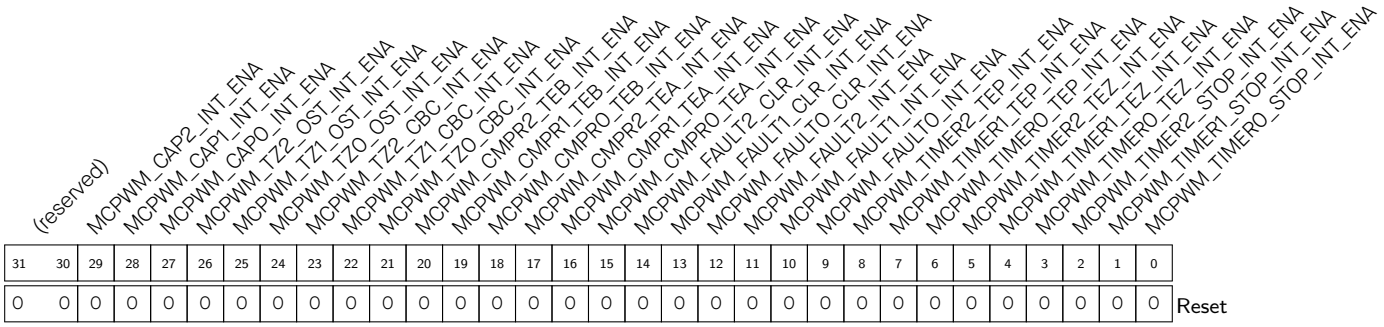
MCPWM_OP2_FORCE_UP Configures whether or not to trigger a forced update of active registers in PWM operator 2.

0: No effect

1: Trigger a forced update

(R/W)

Register 36.69. MCPWM_INT_ENA_REG (0x0110)



MCPWM_TIMER0_STOP_INT_ENA Write 1 to enable MCPWM_TIMER0_STOP_INT. (R/W)

MCPWM_TIMER1_STOP_INT_ENA Write 1 to enable MCPWM_TIMER1_STOP_INT. (R/W)

MCPWM_TIMER2_STOP_INT_ENA Write 1 to enable MCPWM_TIMER2_STOP_INT. (R/W)

MCPWM_TIMER0_TEZ_INT_ENA Write 1 to enable MCPWM_TIMER0_TEZ_INT. (R/W)

MCPWM_TIMER1_TEZ_INT_ENA Write 1 to enable MCPWM_TIMER1_TEZ_INT. (R/W)

MCPWM_TIMER2_TEZ_INT_ENA Write 1 to enable MCPWM_TIMER2_TEZ_INT. (R/W)

MCPWM_TIMER0_TEP_INT_ENA Write 1 to enable MCPWM_TIMER0_TEP_INT. (R/W)

MCPWM_TIMER1_TEP_INT_ENA Write 1 to enable MCPWM_TIMER1_TEP_INT. (R/W)

MCPWM_TIMER2_TEP_INT_ENA Write 1 to enable MCPWM_TIMER2_TEP_INT. (R/W)

MCPWM_FAULT0_INT_ENA Write 1 to enable MCPWM_FAULT0_INT. (R/W)

MCPWM_FAULT1_INT_ENA Write 1 to enable MCPWM_FAULT1_INT. (R/W)

MCPWM_FAULT2_INT_ENA Write 1 to enable MCPWM_FAULT2_INT. (R/W)

MCPWM_FAULT0_CLR_INT_ENA Write 1 to enable MCPWM_FAULT0_CLR_INT. (R/W)

MCPWM_FAULT1_CLR_INT_ENA Write 1 to enable MCPWM_FAULT1_CLR_INT. (R/W)

MCPWM_FAULT2_CLR_INT_ENA Write 1 to enable MCPWM_FAULT2_CLR_INT. (R/W)

MCPWM_CMPR0_TEA_INT_ENA Write 1 to enable MCPWM_CMPR0_TEA_INT. (R/W)

MCPWM_CMPR1_TEA_INT_ENA Write 1 to enable MCPWM_CMPR1_TEA_INT. (R/W)

MCPWM_CMPR2_TEA_INT_ENA Write 1 to enable MCPWM_CMPR2_TEA_INT. (R/W)

MCPWM_CMPR0_TEB_INT_ENA Write 1 to enable MCPWM_CMPR0_TEB_INT. (R/W)

MCPWM_CMPR1_TEB_INT_ENA Write 1 to enable MCPWM_CMPR1_TEB_INT. (R/W)

MCPWM_CMPR2_TEB_INT_ENA Write 1 to enable MCPWM_CMPR2_TEB_INT. (R/W)

MCPWM_TZ0_CBC_INT_ENA Write 1 to enable MCPWM_TZ0_CBC_INT. (R/W)

Continued on the next page...

Register 36.69. MCPWM_INT_ENA_REG (0x0110)

Continued from the previous page...

MCPWM_TZ1_CBC_INT_ENA Write 1 to enable MCPWM_TZ1_CBC_INT. (R/W)

MCPWM_TZ2_CBC_INT_ENA Write 1 to enable MCPWM_TZ2_CBC_INT. (R/W)

MCPWM_TZ0_OST_INT_ENA Write 1 to enable MCPWM_TZ0_OST_INT. (R/W)

MCPWM_TZ1_OST_INT_ENA Write 1 to enable MCPWM_TZ1_OST_INT. (R/W)

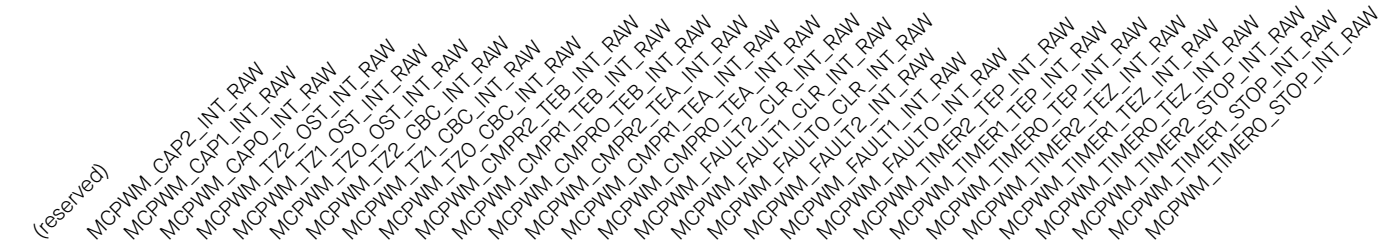
MCPWM_TZ2_OST_INT_ENA Write 1 to enable MCPWM_TZ2_OST_INT. (R/W)

MCPWM_CAP0_INT_ENA Write 1 to enable MCPWM_CAP0_INT. (R/W)

MCPWM_CAP1_INT_ENA Write 1 to enable MCPWM_CAP1_INT. (R/W)

MCPWM_CAP2_INT_ENA Write 1 to enable MCPWM_CAP2_INT. (R/W)

Register 36.70. MCPWM_INT_RAW_REG (0x0114)



31 30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
0 0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Reset

MCPWM_TIMER0_STOP_INT_RAW Represents the raw status of MCPWM_TIMER0_STOP_INT. (R/WTC/SS)

MCPWM_TIMER1_STOP_INT_RAW Represents the raw status of MCPWM_TIMER1_STOP_INT. (R/WTC/SS)

MCPWM_TIMER2_STOP_INT_RAW Represents the raw status of MCPWM_TIMER2_STOP_INT. (R/WTC/SS)

MCPWM_TIMER0_TEZ_INT_RAW Represents the raw status of MCPWM_TIMER0_TEZ_INT. (R/WTC/SS)

MCPWM_TIMER1_TEZ_INT_RAW Represents the raw status of MCPWM_TIMER1_TEZ_INT. (R/WTC/SS)

MCPWM_TIMER2_TEZ_INT_RAW Represents the raw status of MCPWM_TIMER2_TEZ_INT. (R/WTC/SS)

MCPWM_TIMER0_TEP_INT_RAW Represents the raw status of MCPWM_TIMER0_TEP_INT. (R/WTC/SS)

MCPWM_TIMER1_TEP_INT_RAW Represents the raw status of MCPWM_TIMER1_TEP_INT. (R/WTC/SS)

MCPWM_TIMER2_TEP_INT_RAW Represents the raw status of MCPWM_TIMER2_TEP_INT. (R/WTC/SS)

MCPWM_FAULT0_INT_RAW Represents the raw status of MCPWM_FAULT0_INT. (R/WTC/SS)

MCPWM_FAULT1_INT_RAW Represents the raw status of MCPWM_FAULT1_INT. (R/WTC/SS)

MCPWM_FAULT2_INT_RAW Represents the raw status of MCPWM_FAULT2_INT. (R/WTC/SS)

MCPWM_FAULT0_CLR_INT_RAW Represents the raw status of MCPWM_FAULT0_CLR_INT. (R/WTC/SS)

MCPWM_FAULT1_CLR_INT_RAW Represents the raw status of MCPWM_FAULT1_CLR_INT. (R/WTC/SS)

Continued on the next page...

Register 36.70. MCPWM_INT_RAW_REG (0x0114)

Continued from the previous page...

MCPWM_FAULT2_CLR_INT_RAW Represents the raw status of MCPWM_FAULT2_CLR_INT. (R/WTC/SS)

MCPWM_CMPR0_TEA_INT_RAW Represents the raw status of MCPWM_CMPR0_TEA_INT. (R/WTC/SS)

MCPWM_CMPR1_TEA_INT_RAW Represents the raw status of MCPWM_CMPR1_TEA_INT. (R/WTC/SS)

MCPWM_CMPR2_TEA_INT_RAW Represents the raw status of MCPWM_CMPR2_TEA_INT. (R/WTC/SS)

MCPWM_CMPR0_TEB_INT_RAW Represents the raw status of MCPWM_CMPR0_TEB_INT. (R/WTC/SS)

MCPWM_CMPR1_TEB_INT_RAW Represents the raw status of MCPWM_CMPR1_TEB_INT. (R/WTC/SS)

MCPWM_CMPR2_TEB_INT_RAW Represents the raw status of MCPWM_CMPR2_TEB_INT. (R/WTC/SS)

MCPWM_TZ0_CBC_INT_RAW Represents the raw status of MCPWM_TZ0_CBC_INT. (R/WTC/SS)

MCPWM_TZ1_CBC_INT_RAW Represents the raw status of MCPWM_TZ1_CBC_INT. (R/WTC/SS)

MCPWM_TZ2_CBC_INT_RAW Represents the raw status of MCPWM_TZ2_CBC_INT. (R/WTC/SS)

MCPWM_TZ0_OST_INT_RAW Represents the raw status of MCPWM_TZ0_OST_INT. (R/WTC/SS)

MCPWM_TZ1_OST_INT_RAW Represents the raw status of MCPWM_TZ1_OST_INT. (R/WTC/SS)

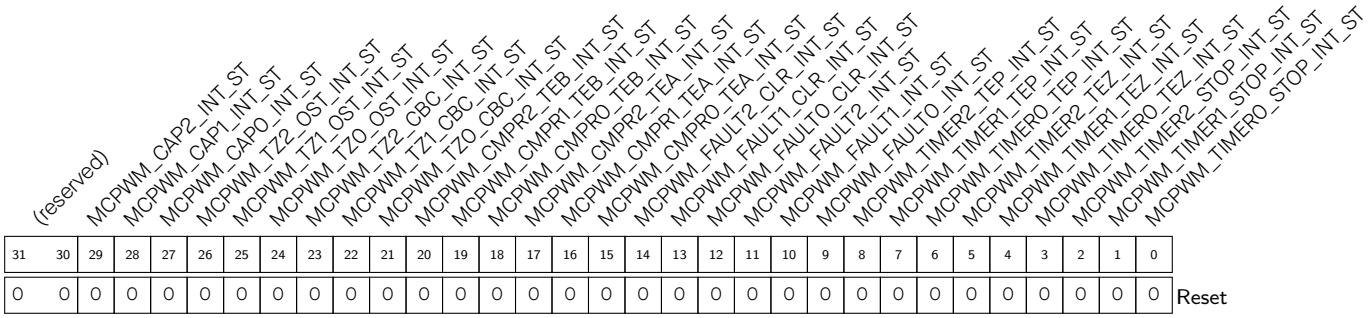
MCPWM_TZ2_OST_INT_RAW Represents the raw status of MCPWM_TZ2_OST_INT. (R/WTC/SS)

MCPWM_CAP0_INT_RAW Represents the raw status of MCPWM_CAP0_INT. (R/WTC/SS)

MCPWM_CAP1_INT_RAW Represents the raw status of MCPWM_CAP1_INT. (R/WTC/SS)

MCPWM_CAP2_INT_RAW Represents the raw status of MCPWM_CAP2_INT. (R/WTC/SS)

Register 36.71. MCPWM_INT_ST_REG (0x0118)



MCPWM_TIMER0_STOP_INT_ST Represents the masked status of MCPWM_TIMER0_STOP_INT. (RO)

MCPWM_TIMER1_STOP_INT_ST Represents the masked status of MCPWM_TIMER1_STOP_INT. (RO)

MCPWM_TIMER2_STOP_INT_ST Represents the masked status of MCPWM_TIMER2_STOP_INT. (RO)

MCPWM_TIMER0_TEZ_INT_ST Represents the masked status of MCPWM_TIMER0_TEZ_INT. (RO)

MCPWM_TIMER1_TEZ_INT_ST Represents the masked status of MCPWM_TIMER1_TEZ_INT. (RO)

MCPWM_TIMER2_TEZ_INT_ST Represents the masked status of MCPWM_TIMER2_TEZ_INT. (RO)

MCPWM_TIMER0_TEP_INT_ST Represents the masked status of MCPWM_TIMER0_TEP_INT. (RO)

MCPWM_TIMER1_TEP_INT_ST Represents the masked status of MCPWM_TIMER1_TEP_INT. (RO)

MCPWM_TIMER2_TEP_INT_ST Represents the masked status of MCPWM_TIMER2_TEP_INT. (RO)

MCPWM_FAULT0_INT_ST Represents the masked status of MCPWM_FAULT0_INT. (RO)

MCPWM_FAULT1_INT_ST Represents the masked status of MCPWM_FAULT1_INT. (RO)

MCPWM_FAULT2_INT_ST Represents the masked status of MCPWM_FAULT2_INT. (RO)

MCPWM_FAULT0_CLR_INT_ST Represents the masked status of MCPWM_FAULT0_CLR_INT. (RO)

MCPWM_FAULT1_CLR_INT_ST Represents the masked status of MCPWM_FAULT1_CLR_INT. (RO)

Continued on the next page...

Register 36.71. MCPWM_INT_ST_REG (0x0118)

Continued from the previous page...

MCPWM_FAULT2_CLR_INT_ST Represents the masked status of MCPWM_FAULT2_CLR_INT. (RO)

MCPWM_CMPR0_TEA_INT_ST Represents the masked status of MCPWM_CMPR0_TEA_INT. (RO)

MCPWM_CMPR1_TEA_INT_ST Represents the masked status of MCPWM_CMPR1_TEA_INT. (RO)

MCPWM_CMPR2_TEA_INT_ST Represents the masked status of MCPWM_CMPR2_TEA_INT. (RO)

MCPWM_CMPR0_TEB_INT_ST Represents the masked status of MCPWM_CMPR0_TEB_INT. (RO)

MCPWM_CMPR1_TEB_INT_ST Represents the masked status of MCPWM_CMPR1_TEB_INT. (RO)

MCPWM_CMPR2_TEB_INT_ST Represents the masked status of MCPWM_CMPR2_TEB_INT. (RO)

MCPWM_TZ0_CBC_INT_ST Represents the masked status of MCPWM_TZ0_CBC_INT_ST. (RO)

MCPWM_TZ1_CBC_INT_ST Represents the masked status of MCPWM_TZ1_CBC_INT_ST. (RO)

MCPWM_TZ2_CBC_INT_ST Represents the masked status of MCPWM_TZ2_CBC_INT_ST. (RO)

MCPWM_TZ0_OST_INT_ST Represents the masked status of MCPWM_TZ0_OST_INT. (RO)

MCPWM_TZ1_OST_INT_ST Represents the masked status of MCPWM_TZ1_OST_INT. (RO)

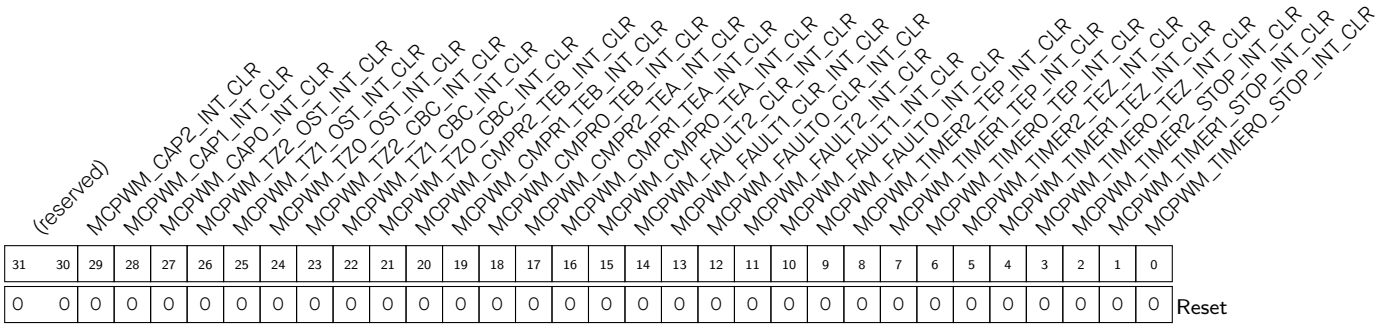
MCPWM_TZ2_OST_INT_ST Represents the masked status of MCPWM_TZ2_OST_INT. (RO)

MCPWM_CAP0_INT_ST Represents the masked status of MCPWM_CAP0_INT. (RO)

MCPWM_CAP1_INT_ST Represents the masked status of MCPWM_CAP1_INT. (RO)

MCPWM_CAP2_INT_ST Represents the masked status of MCPWM_CAP2_INT. (RO)

Register 36.72. MCPWM_INT_CLR_REG (0x011C)



MCPWM_TIMER0_STOP_INT_CLR Write 1 to clear MCPWM_TIMER0_STOP_INT. (WT)

MCPWM_TIMER1_STOP_INT_CLR Write 1 to clear MCPWM_TIMER1_STOP_INT. (WT)

MCPWM_TIMER2_STOP_INT_CLR Write 1 to clear MCPWM_TIMER2_STOP_INT. (WT)

MCPWM_TIMER0_TEZ_INT_CLR Write 1 to clear MCPWM_TIMER0_TEZ_INT. (WT)

MCPWM_TIMER1_TEZ_INT_CLR Write 1 to clear MCPWM_TIMER1_TEZ_INT. (WT)

MCPWM_TIMER2_TEZ_INT_CLR Write 1 to clear MCPWM_TIMER2_TEZ_INT. (WT)

MCPWM_TIMER0_TEP_INT_CLR Write 1 to clear MCPWM_TIMER0_TEP_INT. (WT)

MCPWM_TIMER1_TEP_INT_CLR Write 1 to clear MCPWM_TIMER1_TEP_INT. (WT)

MCPWM_TIMER2_TEP_INT_CLR Write 1 to clear MCPWM_TIMER2_TEP_INT. (WT)

MCPWM_FAULT0_INT_CLR Write 1 to clear MCPWM_FAULT0_INT. (WT)

MCPWM_FAULT1_INT_CLR Write 1 to clear MCPWM_FAULT1_INT. (WT)

MCPWM_FAULT2_INT_CLR Write 1 to clear MCPWM_FAULT2_INT. (WT)

MCPWM_FAULT0_CLR_INT_CLR Write 1 to clear MCPWM_FAULT0_CLR_INT. (WT)

MCPWM_FAULT1_CLR_INT_CLR Write 1 to clear MCPWM_FAULT1_CLR_INT. (WT)

MCPWM_FAULT2_CLR_INT_CLR Write 1 to clear MCPWM_FAULT2_CLR_INT. (WT)

MCPWM_CMPR0_TEA_INT_CLR Write 1 to clear MCPWM_CMPR0_TEA_INT. (WT)

MCPWM_CMPR1_TEA_INT_CLR Write 1 to clear MCPWM_CMPR1_TEA_INT. (WT)

MCPWM_CMPR2_TEA_INT_CLR Write 1 to clear MCPWM_CMPR2_TEA_INT. (WT)

MCPWM_CMPR0_TEB_INT_CLR Write 1 to clear MCPWM_CMPR0_TEB_INT. (WT)

MCPWM_CMPR1_TEB_INT_CLR Write 1 to clear MCPWM_CMPR1_TEB_INT. (WT)

MCPWM_CMPR2_TEB_INT_CLR Write 1 to clear MCPWM_CMPR2_TEB_INT. (WT)

MCPWM_TZ0_CBC_INT_CLR Write 1 to clear MCPWM_TZ0_CBC_INT. (WT)

Continued on the next page...

Register 36.72. MCPWM_INT_CLR_REG (0x011C)

Continued from the previous page...

MCPWM_TZ1_CBC_INT_CLR Write 1 to clear MCPWM_TZ1_CBC_INT. (WT)

MCPWM_TZ2_CBC_INT_CLR Write 1 to clear MCPWM_TZ2_CBC_INT. (WT)

MCPWM_TZ0_OST_INT_CLR Write 1 to clear MCPWM_TZ0_OST_INT. (WT)

MCPWM_TZ1_OST_INT_CLR Write 1 to clear MCPWM_TZ1_OST_INT. (WT)

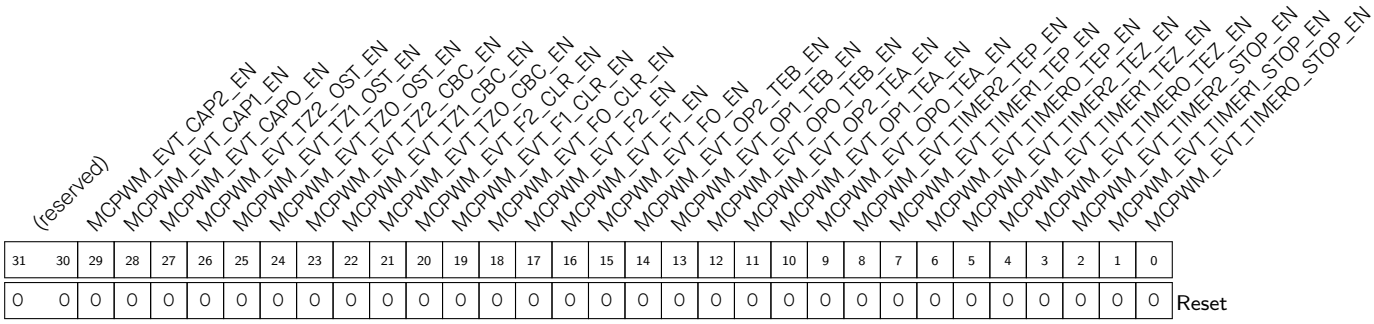
MCPWM_TZ2_OST_INT_CLR Write 1 to clear MCPWM_TZ2_OST_INT. (WT)

MCPWM_CAP0_INT_CLR Write 1 to clear MCPWM_CAP0_INT. (WT)

MCPWM_CAP1_INT_CLR Write 1 to clear MCPWM_CAP1_INT. (WT)

MCPWM_CAP2_INT_CLR Write 1 to clear MCPWM_CAP2_INT. (WT)

Register 36.73. MCPWM_EVT_EN_REG (0x0120)



MCPWM_EVT_TIMER0_STOP_EN Configures whether or not to enable timer0 stop event generation.

0: Disable

1: Enable

(R/W)

MCPWM_EVT_TIMER1_STOP_EN Configures whether or not to enable timer1 stop event generation.

0: Disable

1: Enable

(R/W)

MCPWM_EVT_TIMER2_STOP_EN Configures whether or not to enable timer2 stop event generation.

0: Disable

1: Enable

(R/W)

MCPWM_EVT_TIMER0_TEZ_EN Configures whether or not to enable timer0 equal zero event generation.

0: Disable

1: Enable

(R/W)

MCPWM_EVT_TIMER1_TEZ_EN Configures whether or not to enable timer1 equal zero event generation.

0: Disable

1: Enable

(R/W)

MCPWM_EVT_TIMER2_TEZ_EN Configures whether or not to enable timer2 equal zero event generation.

0: Disable

1: Enable

(R/W)

Continued on the next page...

Register 36.73. MCPWM_EVT_EN_REG (0x0120)

Continued from the previous page...

MCPWM_EVT_TIMER0_TEP_EN Configures whether or not to enable timer0 equal period event generation.
0: Disable
1: Enable
(R/W)

MCPWM_EVT_TIMER1_TEP_EN Configures whether or not to enable timer1 equal period event generation.
0: Disable
1: Enable
(R/W)

MCPWM_EVT_TIMER2_TEP_EN Configures whether or not to enable timer2 equal period event generation.
0: Disable
1: Enable
(R/W)

MCPWM_EVT_OP0_TEA_EN Configures whether or not to enable PWM generator0 timer equal A event generation.
0: Disable
1: Enable
(R/W)

MCPWM_EVT_OP1_TEA_EN Configures whether or not to enable PWM generator1 timer equal A event generation.
0: Disable
1: Enable
(R/W)

MCPWM_EVT_OP2_TEA_EN Configures whether or not to enable PWM generator2 timer equal A event generation.
0: Disable
1: Enable
(R/W)

MCPWM_EVT_OP0_TEB_EN Configures whether or not to enable PWM generator0 timer equal B event generation.
0: Disable
1: Enable
(R/W)

Continued on the next page...

Register 36.73. MCPWM_EVT_EN_REG (0x0120)

Continued from the previous page...

MCPWM_EVT_OP1_TEB_EN Configures whether or not to enable PWM generator1 timer equal B event generation.
0: Disable
1: Enable
(R/W)

MCPWM_EVT_OP2_TEB_EN Configures whether or not to enable PWM generator2 timer equal B event generation.
0: Disable
1: Enable
(R/W)

MCPWM_EVT_F0_EN Configures whether or not to enable FAULT0 event generation.
0: Disable
1: Enable
(R/W)

MCPWM_EVT_F1_EN Configures whether or not to enable FAULT1 event generation.
0: Disable
1: Enable
(R/W)

MCPWM_EVT_F2_EN Configures whether or not to enable FAULT2 event generation.
0: Disable
1: Enable
(R/W)

MCPWM_EVT_F0_CLR_EN Configures whether or not to enable FAULT0 clear event generation.
0: Disable
1: Enable
(R/W)

MCPWM_EVT_F1_CLR_EN Configures whether or not to enable FAULT1 clear event generation.
0: Disable
1: Enable
(R/W)

MCPWM_EVT_F2_CLR_EN Configures whether or not to enable FAULT2 clear event generation.
0: Disable
1: Enable
(R/W)

Continued on the next page...

Register 36.73. MCPWM_EVT_EN_REG (0x0120)

Continued from the previous page...

MCPWM_EVT_TZ0_CBC_EN Configures whether or not to enable cycle by cycle trip0 event generation.
0: Disable
1: Enable
(R/W)

MCPWM_EVT_TZ1_CBC_EN Configures whether or not to enable cycle by cycle trip1 event generation.
0: Disable
1: Enable
(R/W)

MCPWM_EVT_TZ2_CBC_EN Configures whether or not to enable cycle by cycle trip2 event generation.
0: Disable
1: Enable
(R/W)

MCPWM_EVT_TZ0_OST_EN Configures whether or not to enable one shot trip0 event generation.
0: Disable
1: Enable
(R/W)

MCPWM_EVT_TZ1_OST_EN Configures whether or not to enable one shot trip1 event generation.
0: Disable
1: Enable
(R/W)

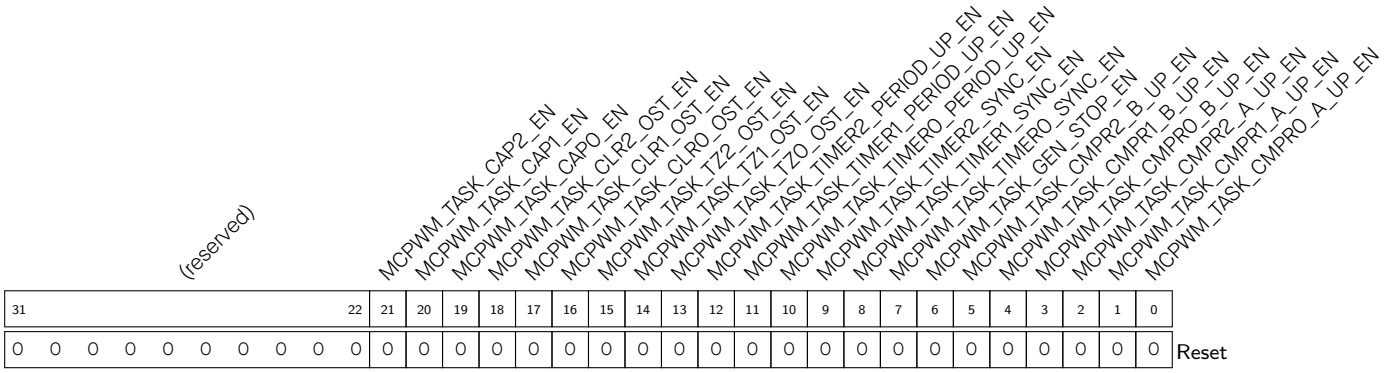
MCPWM_EVT_TZ2_OST_EN Configures whether or not to enable one shot trip2 event generation.
0: Disable
1: Enable
(R/W)

MCPWM_EVT_CAP0_EN Configures whether or not to enable capture0 event generation.
0: Disable
1: Enable
(R/W)

MCPWM_EVT_CAP1_EN Configures whether or not to enable capture1 event generation.
0: Disable
1: Enable
(R/W)

MCPWM_EVT_CAP2_EN Configures whether or not to enable capture2 event generation.
0: Disable
1: Enable
(R/W)

Register 36.74. MCPWM_TASK_EN_REG (0x0124)



MCPWM_TASK_CMPR0_A_UP_EN Configures whether or not to receive update task of PWM generator0 timer stamp A's shadow register.

0: No effect

1: Receive

(R/W)

MCPWM_TASK_CMPR1_A_UP_EN Configures whether or not to receive update task of PWM generator1 timer stamp A's shadow register.

0: No effect

1: Receive

(R/W)

MCPWM_TASK_CMPR2_A_UP_EN Configures whether or not to receive update task of PWM generator2 timer stamp A's shadow register.

0: No effect

1: Receive

(R/W)

MCPWM_TASK_CMPR0_B_UP_EN Configures whether or not to receive update task of PWM generator0 timer stamp B's shadow register.

0: No effect

1: Receive

(R/W)

MCPWM_TASK_CMPR1_B_UP_EN Configures whether or not to receive update task of PWM generator1 timer stamp B's shadow register.

0: No effect

1: Receive

(R/W)

MCPWM_TASK_CMPR2_B_UP_EN Configures whether or not to receive update task of PWM generator2 timer stamp B's shadow register.

0: No effect

1: Receive

(R/W)

Continued on the next page...

Register 36.74. MCPWM_TASK_EN_REG (0x0124)

Continued from the previous page...

MCPWM_TASK_GEN_STOP_EN Configures whether or not to receive all PWM generate stop task.
0: No effect
1: Receive
(R/W)

MCPWM_TASK_TIMER0_SYNC_EN Configures whether or not to receive timer0 sync task.
0: No effect
1: Receive
(R/W)

MCPWM_TASK_TIMER1_SYNC_EN Configures whether or not to receive timer1 sync task.
0: No effect
1: Receive
(R/W)

MCPWM_TASK_TIMER2_SYNC_EN Configures whether or not to receive timer2 sync task.
0: No effect
1: Receive
(R/W)

MCPWM_TASK_TIMER0_PERIOD_UP_EN Configures whether or not to receive timer0 period update task.
0: No effect
1: Receive
(R/W)

MCPWM_TASK_TIMER1_PERIOD_UP_EN Configures whether or not to receive timer1 period update task.
0: No effect
1: Receive
(R/W)

MCPWM_TASK_TIMER2_PERIOD_UP_EN Configures whether or not to receive timer2 period update task.
0: No effect
1: Receive
(R/W)

Continued on the next page...

Register 36.74. MCPWM_TASK_EN_REG (0x0124)

Continued from the previous page...

MCPWM_TASK_TZ0_OST_EN Configures whether or not to receive one shot trip0 task.

0: No effect

1: Receive

(R/W)

MCPWM_TASK_TZ1_OST_EN Configures whether or not to receive one shot trip1 task.

0: No effect

1: Receive

(R/W)

MCPWM_TASK_TZ2_OST_EN Configures whether or not to receive one shot trip2 task.

0: No effect

1: Receive

(R/W)

MCPWM_TASK_CLR0_OST_EN Configures whether or not to receive one shot trip0 clear task.

0: No effect

1: Receive

(R/W)

MCPWM_TASK_CLR1_OST_EN Configures whether or not to receive one shot trip1 clear task.

0: No effect

1: Receive

(R/W)

MCPWM_TASK_CLR2_OST_EN Configures whether or not to receive one shot trip2 clear task.

0: No effect

1: Receive

(R/W)

MCPWM_TASK_CAP0_EN Configures whether or not to receive capture0 task.

0: No effect

1: Receive

(R/W)

MCPWM_TASK_CAP1_EN Configures whether or not to receive capture1 task.

0: No effect

1: Receive

(R/W)

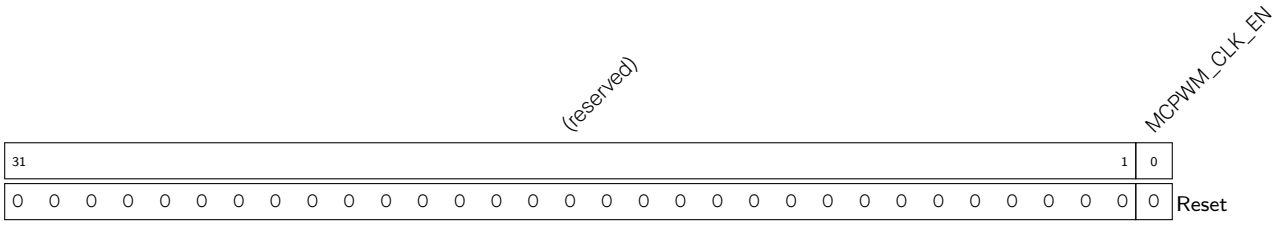
MCPWM_TASK_CAP2_EN Configures whether or not to receive capture2 task.

0: No effect

1: Receive

(R/W)

Register 36.75. MCPWM_CLK_REG (0x0128)



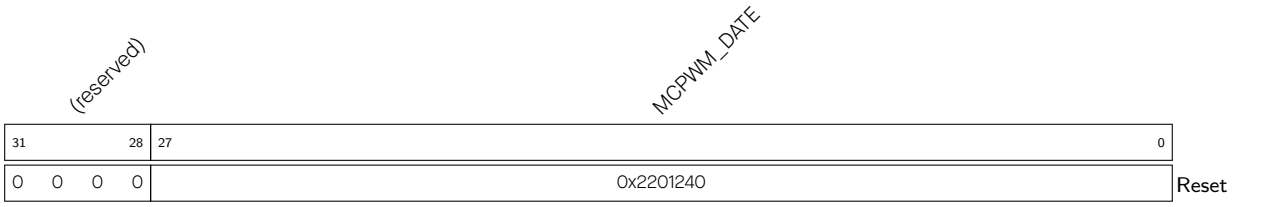
MCPWM_CLK_EN Configures whether or not to force the clock on.

0: No effect

1: Force the clock on

(R/W)

Register 36.76. MCPWM_VERSION_REG (0x012C)



MCPWM_DATE Version control register. (R/W)

Chapter 37

Remote Control Peripheral (RMT)

37.1 Overview

The RMT (Remote Control) module is designed to send and receive infrared remote control signals. A variety of remote control protocols can be encoded/decoded via software based on the RMT module. The RMT module converts pulse codes stored in the module's built-in RAM into output signals, or converts input signals into pulse codes and stores them in RAM. In addition, the RMT module optionally modulates its output signals with a carrier wave, or optionally demodulates and filters its input signals.

The RMT module has four channels, numbered from zero to three. Each channel is able to independently transmit or receive signals.

- Channels 0 ~ 1 (TX channel) are dedicated to transmitting signals;
- Channels 2 ~ 3 (RX channel) are dedicated to receiving signals.

Each TX/RX channel has the same functionality controlled by a dedicated set of registers and is able to independently transmit or receive data. TX channels are indicated by *n* which is used as a placeholder for the channel number, and by *m* for RX channels.

37.2 Features

The RMT module has the following features:

- Four channels:
  - Two TX channels
  - Two RX channels
  - Four channels share a 192 x 32-bit RAM
- The transmitter supports:
  - Normal TX mode
  - Wrap TX mode
  - Continuous TX mode
  - Modulation on TX pulses
  - Multiple channels (programmable) transmitting data simultaneously
- The receiver supports:
  - Normal RX mode

- Wrap RX mode
- RX filtering
- Demodulation on RX pulses

37.3 Functional Description

37.3.1 RMT Architecture

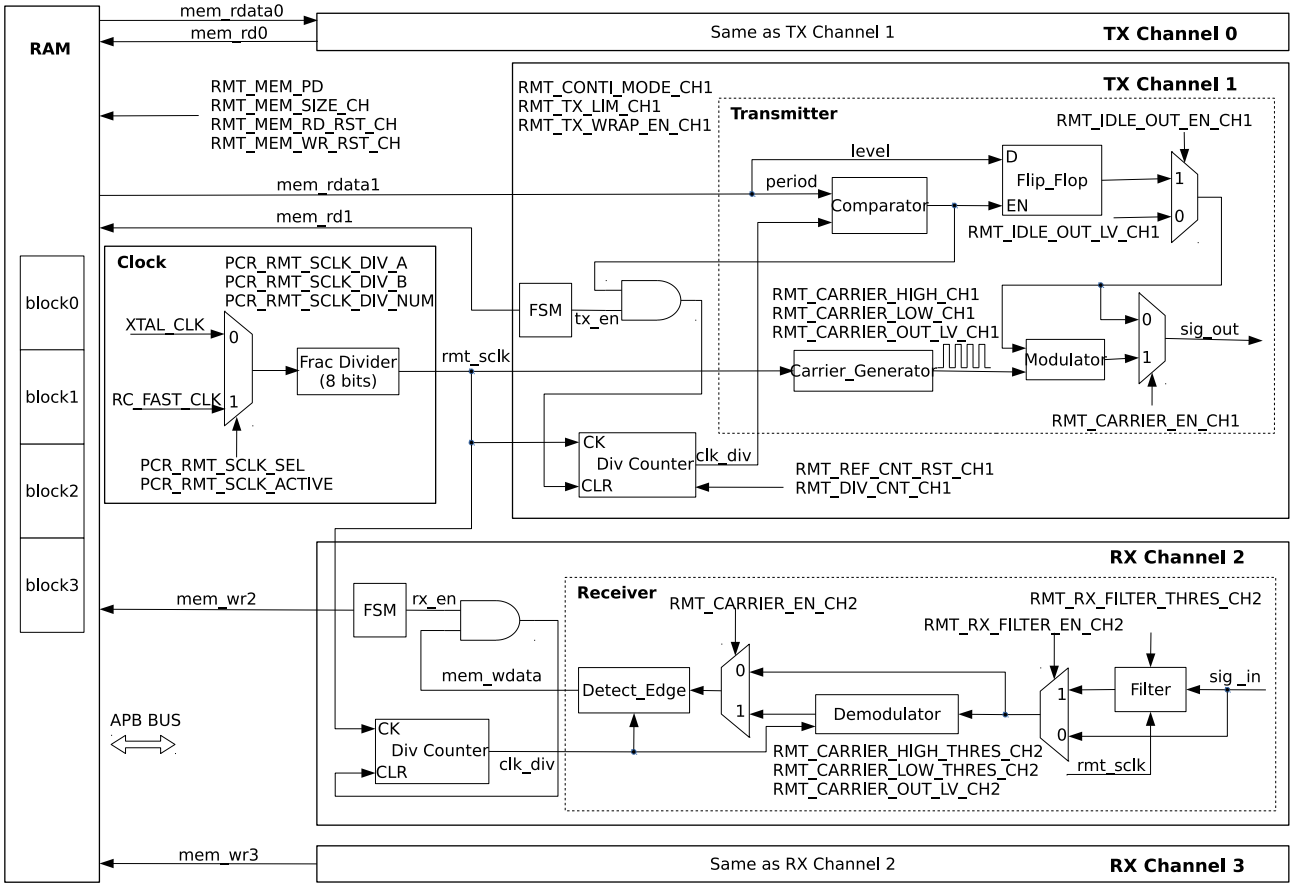


Figure 37.3-1. RMT Architecture

As shown in Figure 37.3-1, each TX channel has:

- 1 x clock divider counter (Div Counter)
- 1 x state machine (FSM)
- 1 x transmitter

Each RX channel also has:

- 1 x clock divider counter (Div Counter)
- 1 x state machine (FSM)

- 1 x receiver

The four channels share a 192 x 32-bit RAM.

37.3.2 RMT RAM

37.3.2.1 Structure of RAM

Figure 37.3-2 shows the format of pulse code in RAM. Each pulse code contains a 16-bit entry with two fields: "level" and "period". "level" (0 or 1) indicates a low-/high-level value that has been received or is going to be sent, while "period" points out the number of clock cycles (see clk_div in Figure 37.3-1) that the level lasts for.

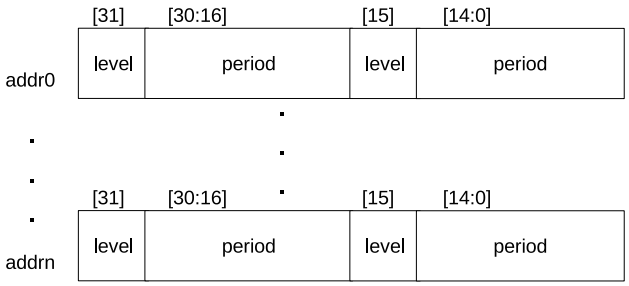


Figure 37.3-2. Format of Pulse Code in RAM

The minimum value for the period is zero (0) and is interpreted as a transmission end-marker. For a non-zero period (i.e., not an end-marker), its value is limited by APB clock and RMT clock according to the equation below:

$$5 \times T_{apb\_clk} + 6 \times T_{rmt\_sclk} < period \times T_{clk\_div} \quad (37.1)$$

Note:
RMT clock and APB clock must satisfy the following relation:

$$1.5 \times T_{apb\_clk} < 9 \times T_{rmt\_sclk} \quad (37.2)$$

37.3.2.2 Use of RAM

The RAM is divided into four 48 x 32-bit blocks. By default, each channel uses one block (block 0 for channel 0, block 1 for channel 1, and so on).

If the data size of one single transfer is larger than the block size of TX channel *n* or RX channel *m*, users can configure the channel:

- to enable wrap mode by setting RMT_MEM_TX/RX_WRAP_EN_CH*n*/*m*;
- or to use more blocks by configuring RMT_MEM_SIZE_CH*n*/*m*.

Setting RMT_MEM_SIZE_CH*n*/*m* > 1 allows channel *n*/*m* to use the memory of the subsequent channels, i.e., block (*n*/*m*) ~ block (*n*/*m* + RMT_MEM_SIZE_CH*n*/*m* − 1). In such case, the subsequent channels (*n*/*m* + 1) ~ (*n*/*m* + RMT_MEM_SIZE_CH*n*/*m* − 1) can not be used since their RAM blocks are occupied. For example, if

channel 0 is configured to use block 0 and block 1, then channel 1 will be unavailable since its block is occupied, while channel 2 and channel 3 are not affected and can be used normally.

Note that the RAM used by each channel is mapped from low address to high address. In such mode, channel 0 is able to use the RAM blocks of channels 1, 2, and 3 by setting RMT_MEM_SIZE_CH0, but channel 3 can not use the blocks of channels 0, 1, or 2. Therefore, the maximum value of RMT_MEM_SIZE_CH*n* should not exceed $(4 - n)$ and the maximum value of RMT_MEM_SIZE_CH*m* should not exceed $(2 - m)$.

The RMT RAM can be accessed via APB bus, or read by the transmitter and written by the receiver. To avoid any possible access conflict between the receiver writing RAM and the APB bus reading RAM, RMT can be configured to designate the RAM block's owner, be it the receiver or the APB bus, by configuring RMT_MEM_OWNER_CH*m*. If this ownership is violated, a flag signal RMT_MEM_OWNER_ERR_CH*m* will be generated.

When the RMT module is inactive, the RAM can be put into low-power mode by setting RMT_MEM_FORCE_PD.

37.3.2.3 RAM Access

APB bus is able to access RAM in FIFO mode and in NONFIFO (Direct Address) mode, depending on the configuration of RMT_APB_FIFO_MASK:

- 0: use FIFO mode;
- 1: use NONFIFO mode.

FIFO Mode

In FIFO mode, the APB reads data from or writes data to RAM via a fixed address stored in RMT_CH*n*/*m*DATA_REG.

NONFIFO Mode

In NONFIFO mode, the APB writes data to or reads data from a continuous address range.

- The write-starting address of TX channel *n* is: RMT base address + 0x400 + *n*x48. The access address for the second data and the following data are RMT base address + 0x400 + *n*x48 + 0x4, and so on, incremented by 0x4.
- The read-starting address of RX channel *m* is: RMT base address + 0x460 + *m*x48. The access address for the second data and the following data are RMT base address + 0x460 + *m*x48 + 0x4, and so on, incremented by 0x4.

37.3.3 Clock

The clock source of RMT can be XTAL_CLK or RC_FAST_CLK, depending on the configuration of PCR_RMT_SCLK_SEL. RMT clock can be enabled by setting PCR_RMT_SCLK_EN. RMT working clock (see rmt_sclk in Figure 37.3-1) is obtained by dividing the selected clock source with a fractional divider. The divider is:

$$PCR\_RMT\_SCLK\_DIV\_NUM + 1 + PCR\_RMT\_SCLK\_DIV\_A/PCR\_RMT\_SCLK\_DIV\_B$$

For more information, see Chapter 7 *Reset and Clock*. RMT_DIV_CNT_CH*n*/*m* is used to configure the divider coefficient of internal clock divider for RMT channels. The coefficient is normally equal to the value of RMT_DIV_CNT_CH*n*/*m*, except for value 0 that represents divider 256. The clock divider can be reset by setting RMT_REF_CNT_RST_CH*n*/*m*. The clock generated from the divider can be used by the counter (see Figure 37.3-1).

37.3.4 Transmitter

Note:
Updating the configuration described in this and subsequent sections requires setting RMT_CONF_UPDATE_CH*n*/*m* first. See Section 37.3.6.

37.3.4.1 Normal TX Mode

When RMT_TX_START_CH*n* is set, the transmitter of channel *n* starts reading and sending pulse codes from the starting address of its RAM block. The codes are sent starting from low-address entry. When an end-marker (a zero period) is encountered, the transmitter stops the transmission, returns to idle state, and generates an RMT_CH*n*_TX_END_INT interrupt. Setting RMT_TX_STOP_CH*n* to 1 also stops the transmission and immediately sets the transmitter back to idle. The output level of a transmitter in idle state is determined by the "level" field of the end-marker or by the content of RMT_IDLE_OUT_LV_CH*n*, depending on the configuration of RMT_IDLE_OUT_EN_CH*n*:

- 0: the level in idle state is determined by the "level" field of the end-marker;
- 1: the level is determined by RMT_IDLE_OUT_LV_CH*n*.

37.3.4.2 Wrap TX Mode

To transmit more pulse codes than that can be fitted in the channel's RAM, users can enable wrap TX mode for channel *n* by setting RMT_MEM_TX_WRAP_EN_CH*n*. In this mode, the transmitter sends the data from RAM in loops till an end-marker is encountered. For example, if RMT_MEM_SIZE_CH*n* = 1, the transmitter starts sending data from the address 48 x *n*, and then the data from higher RAM address. Once the transmitter finishes sending the data from (48 x ($n + 1$) $-$ 1), it continues sending data from 48 x *n* again till an end-marker is encountered. Wrap mode is also applicable for RMT_MEM_SIZE_CH*n* > 1.

When the size of transmitted pulse codes is larger than or equal to the value set by RMT_TX_LIM_CH*n*, an RMT_CH*n*_TX_THR_EVENT_INT interrupt is triggered. In wrap mode, RMT_TX_LIM_CH*n* can be set to a half or a fraction of the size of the channel's RAM block. When an RMT_CH*n*_TX_THR_EVENT_INT interrupt is detected by software, the already used RAM region can be updated by new pulse codes. In such way, the transmitter can seamlessly send unlimited pulse codes in wrap mode.

37.3.4.3 TX Modulation

Transmitter output can be modulated with a carrier wave by setting RMT_CARRIER_EN_CH*n*. The carrier waveform is configurable. In a carrier cycle, high level lasts for (RMT_CARRIER_HIGH_CH*n* + 1) rmt_sclk cycles, while low level lasts for (RMT_CARRIER_LOW_CH*n* + 1) rmt_sclk cycles. When RMT_CARRIER_OUT_LV_CH*n* is set, carrier wave is added on the high-level of output signals; while RMT_CARRIER_OUT_LV_CH*n* is cleared, carrier wave is added on the low-level of output signals. Carrier wave

can be added on all output signals during modulation, or just added on valid pulse codes (the data stored in RAM), which can be set by configuring RMT_CARRIER_EFF_EN_CH*n*:

- 0: add carrier wave on all output signals;
- 1: add carrier wave only on valid signals.

37.3.4.4 Continuous TX Mode

The continuous TX mode can be enabled by setting RMT_TX_CONTI_MODE_CH*n*. In this mode, the transmitter sends the pulse codes from RAM in loops:

- If an end-marker is encountered, the transmitter starts transmitting the first data of the channel's RAM again.
- If no end-marker is encountered, there are two possible situations. In normal TX mode (RMT_MEM_TX_WRAP_EN_CH*n* = 0), an error interrupt occurs because the RAM is empty without any data to transmit. In wrap TX mode (RMT_MEM_TX_WRAP_EN_CH*n* = 1), the transmitter starts transmitting the first data again after the last data is transmitted.

If RMT_TX_LOOP_CNT_EN_CH*n* is set, the loop counting is incremented by 1 each time an end-marker is encountered. If the counting reaches the value set by RMT_TX_LOOP_NUM_CH*n*, an RMT_CH*n*_TX_LOOP_INT interrupt is generated. If RMT_LOOP_STOP_EN_CH*n* is set, the transmission stops instantly after an RMT_CH*n*_TX_LOOP_INT interrupt is generated. Otherwise, the transmission continues. In an end-marker, if its period[14:0] is 0, then the period of the previous data must satisfy:

$$10 \times T_{apb\_clk} + 19 \times T_{rmt\_sclk} < period \times T_{clk\_div} \tag{37.3}$$

The period of the other data only needs to satisfy non-zero period relation.

37.3.4.5 Simultaneous TX Mode

RMT module supports multiple channels transmitting data simultaneously. To use this function, follow the steps below:

1. Configure RMT_TX_SIM_CH*n* to choose which channels are used to transmit data simultaneously;
2. Set RMT_TX_SIM_EN to enable this transmission mode;
3. Set RMT_TX_START_CH*n* for each selected channel to start data transmission.

The transmission starts once the last channel is configured. Due to hardware limitations, there is no guarantee that two channels can start sending data exactly at the same time. The interval between two channels starting transmitting data is within 3 x T_{clk_div}.

37.3.5 Receiver

37.3.5.1 Normal RX Mode

The receiver of channel *m* is controlled by RMT_RX_EN_CH*m*:

- 0: the receiver stops receiving data;
- 1: the receiver starts working.

When the receiver becomes active, it starts counting from the first edge of the signal, detecting signal levels, and counting clock cycles the level lasts for. Each cycle count (period) is then written back to RAM together with the level information (level). When the receiver detects no change in a signal level for a number of clock cycles more than the value set by RMT_IDLE_THRES_CH*m*, the receiver stops receiving data, returns to idle state, and generates an RMT_CH*m*_RX_END_INT interrupt. Please note that RMT_IDLE_THRES_CH*m* should be configured to a value greater than the maximum clock cycle number of the input signal high/low level, otherwise a valid received level may be mistaken as a level in idle state. If the RAM space of this RX channel is used up by the received data, the receiver stops receiving data, and an RMT_CH*m*_ERR_INT interrupt is triggered by RAM FULL event.

37.3.5.2 Wrap RX Mode

To receive more pulse codes than can be fitted in the channel's RAM, users can enable wrap mode for channel *m* by configuring RMT_MEM_RX_WRAP_EN_CH*m*. In wrap mode, the receiver stores the received data to RAM space of this channel in loops. The receiving ends when the receiver detects no change in a signal level for a number of clock cycles more than the value set by RMT_IDLE_THRES_CH*m*. The receiver returns to idle state and generates an RMT_CH*m*_RX_END_INT interrupt. For example, if RMT_MEM_SIZE_CH*m* is set to 1, the receiver starts receiving data and stores the data to address 48 x *m*, and then to higher RAM address. When the receiver finishes storing the received data to (48 x (*m* + 1) − 1), the receiver continues receiving data and storing data to the address 48 x *m* again, and the receiving ends when no change is detected on a signal level for more than RMT_IDLE_THRES_CH*m* clock cycles. Wrap mode is also applicable when RMT_MEM_SIZE_CH*m* > 1.

An RMT_CH*m*_RX_THR_EVENT_INT interrupt is generated when the size of received pulse codes is larger than or equal to the value set by RMT_RX_LIM_CH*m*. In wrap mode, RMT_RX_LIM_CH*m* can be set to a half or a fraction of the size of the channel's RAM block. When an RMT_CH*m*_RX_THR_EVENT_INT interrupt is detected, the already used RAM region can be updated by subsequent data.

37.3.5.3 RX Filtering

Users can enable the receiver to filter input signals by setting RMT_RX_FILTER_EN_CH*m* for channel *m*. The filter samples input signals continuously, and detects the signals which remain unchanged for a continuous RMT_RX_FILTER_THRES_CH*m* rmt_sclk cycles as valid. Otherwise, the signals will be detected as invalid. Only the valid signals can pass through the filter. The filter removes pulses with a length of less than RMT_RX_FILTER_THRES_CH*m* rmt_sclk cycles.

37.3.5.4 RX Demodulation

Users can enable RX demodulation on input signals or on filtered signals by setting RMT_CARRIER_EN_CH*m*. RX demodulation can be applied to high-level carrier wave or low-level carrier wave, depending on the configuration of RMT_CARRIER_OUT_LV_CH*m*:

- 0: demodulate low-level carrier wave;
- 1: demodulate high-level carrier wave.

Users can configure RMT_CARRIER_HIGH_THRES_CH*m* and RMT_CARRIER_LOW_THRES_CH*m* to set the thresholds to demodulate high-level carrier or low-level carrier. If the high-level of a signal lasts for less than RMT_CARRIER_HIGH_THRES_CH*m* clk_div cycles, or the low-level lasts for less than

RMT_CARRIER_LOW_THRES_CH*m* clk_div cycles, the signal is detected as a carrier and is then filtered out.

37.3.6 Configuration Update

To update RMT registers configuration, please set RMT_CONF_UPDATE_CH*n*/*m* for each channel first.

All the bits/fields listed in the second column of Table 37.3-1 should follow this rule.

Table 37.3-1. Configuration Update

Register	Bit/Field Configuration Update
TX Channel	
RMT_CH*n*CONFO_REG	RMT_CARRIER_OUT_LV_CH*n*
	RMT_CARRIER_EN_CH*n*
	RMT_CARRIER_EFF_EN_CH*n*
	RMT_DIV_CNT_CH*n*
	RMT_IDLE_OUT_EN_CH*n*
	RMT_IDLE_OUT_LV_CH*n*
	RMT_TX_CONTI_MODE_CH*n*
RMT_CH*n*CARRIER_DUTY_REG	RMT_CARRIER_HIGH_CH*n*
	RMT_CARRIER_LOW_CH*n*
RMT_CH*n*_TX_LIM_REG	RMT_TX_LOOP_CNT_EN_CH*n*
	RMT_TX_LOOP_NUM_CH*n*
	RMT_TX_LIM_CH*n*
RMT_TX_SIM_REG	RMT_TX_SIM_EN
RX Channel	
RMT_CH*m*CONFO_REG	RMT_CARRIER_OUT_LV_CH*m*
	RMT_CARRIER_EN_CH*m*
	RMT_IDLE_THRES_CH*m*
	RMT_DIV_CNT_CH*m*
RMT_CH*m*CONF1_REG	RMT_RX_FILTER_THRES_CH*m*
	RMT_RX_EN_CH*m*
RMT_CH*m*_RX_CARRIER_RM_REG	RMT_CARRIER_HIGH_THRES_CH*m*
	RMT_CARRIER_LOW_THRES_CH*m*
RMT_CH*m*_RX_LIM_REG	RMT_RX_LIM_CH*m*
RMT_REF_CNT_RST_REG	RMT_REF_CNT_RST_CH*m*

37.3.7 Interrupts

- RMT_CH*n*/*m*_ERR_INT: triggered when channel *n*/*m* does not read or write data correctly. For example, if the transmitter still tries to read data from RAM when the RAM is empty, or the receiver still tries to write data into RAM when the RAM is full, this interrupt will be triggered.
- RMT_CH*n*_TX_THR_EVENT_INT: triggered when the amount of data the transmitter has sent reaches the value set in RMT_CH*n*_TX_LIM_REG.

- RMT_CH*m*_RX_THR_EVENT_INT: triggered each time when the amount of data received by the receiver reaches the value set in RMT_CH*m*_RX_LIM_REG.
- RMT_CH*n*_TX_END_INT: triggered when the transmitter has finished transmitting signals.
- RMT_CH*m*_RX_END_INT: triggered when the receiver has finished receiving signals.
- RMT_CH*n*_TX_LOOP_INT: triggered when the loop counting reaches the value set by RMT_TX_LOOP_NUM_CH*n*.

37.4 Register Summary

The addresses in this section are relative to Remote Control Peripheral base address provided in Table 4.3-2 in Chapter 4 *System and Memory*.

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

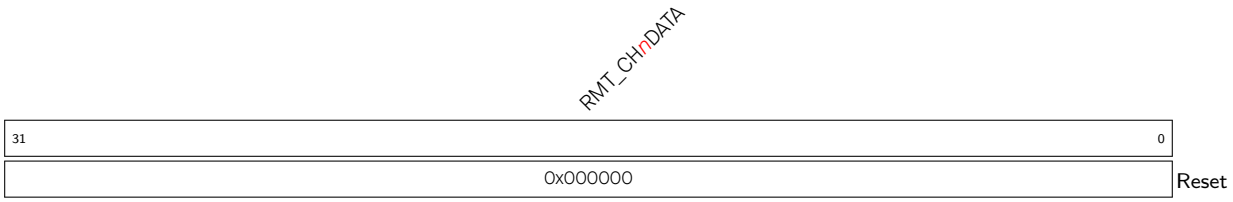
Name	Description	Address	Access
FIFO R/W Registers			
RMT_CH0DATA_REG	The write data register for channel 0 by APB FIFO access.	0x0000	WO
RMT_CH1DATA_REG	The write data register for channel 1 by APB FIFO access.	0x0004	WO
RMT_CH2DATA_REG	The read data register for channel 2 by APB FIFO access.	0x0008	RO
RMT_CH3DATA_REG	The read data register for channel 3 by APB FIFO access.	0x000C	RO
Configuration Registers			
RMT_CH0CONF0_REG	Configuration register 0 for channel 0	0x0010	varies
RMT_CH1CONF0_REG	Configuration register 0 for channel 1	0x0014	varies
RMT_CH2CONF0_REG	Configuration register 0 for channel 2	0x0018	R/W
RMT_CH2CONF1_REG	Configuration register 1 for channel 2	0x001C	varies
RMT_CH3CONF0_REG	Configuration register 0 for channel 3	0x0020	R/W
RMT_CH3CONF1_REG	Configuration register 1 for channel 3	0x0024	varies
RMT_SYS_CONF_REG	Configuration register for RMT APB	0x0068	R/W
RMT_REF_CNT_RST_REG	Reset register for RMT clock divider	0x0070	WT
Status Registers			
RMT_CH0STATUS_REG	Channel 0 status register	0x0028	RO
RMT_CH1STATUS_REG	Channel 1 status register	0x002C	RO
RMT_CH2STATUS_REG	Channel 2 status register	0x0030	RO
RMT_CH3STATUS_REG	Channel 3 status register	0x0034	RO
Interrupt Registers			
RMT_INT_RAW_REG	Raw interrupt status	0x0038	R/WTC/SS
RMT_INT_ST_REG	Masked interrupt status	0x003C	RO
RMT_INT_ENA_REG	Interrupt enable bits	0x0040	R/W
RMT_INT_CLR_REG	Interrupt clear bits	0x0044	WT
Carrier Wave Duty Cycle Registers			
RMT_CH0CARRIER_DUTY_REG	Duty cycle configuration register for channel 0	0x0048	R/W
RMT_CH1CARRIER_DUTY_REG	Duty cycle configuration register for channel 1	0x004C	R/W
RMT_CH2_RX_CARRIER_RM_REG	Carrier remove register for channel 2	0x0050	R/W
RMT_CH3_RX_CARRIER_RM_REG	Carrier remove register for channel 3	0x0054	R/W
TX Event Configuration Registers			
RMT_CH0_TX_LIM_REG	Configuration register for channel 0 TX event	0x0058	varies

Name	Description	Address	Access
RMT_CH1_TX_LIM_REG	Configuration register for channel 1 TX event	0x005C	varies
RMT_TX_SIM_REG	RMT TX synchronous register	0x006C	R/W
RX Event Configuration Registers			
RMT_CH2_RX_LIM_REG	Configuration register for channel 2 RX event	0x0060	R/W
RMT_CH3_RX_LIM_REG	Configuration register for channel 3 RX event	0x0064	R/W
Version Register			
RMT_DATE_REG	Version control register	0x00CC	R/W

37.5 Registers

The addresses in this section are relative to Remote Control Peripheral base address provided in Table 4.3-2 in Chapter 4 *System and Memory*.

Register 37.1. RMT_CH*n*DATA_REG (*n*: 0-1) (0x0000+0x4n*)**

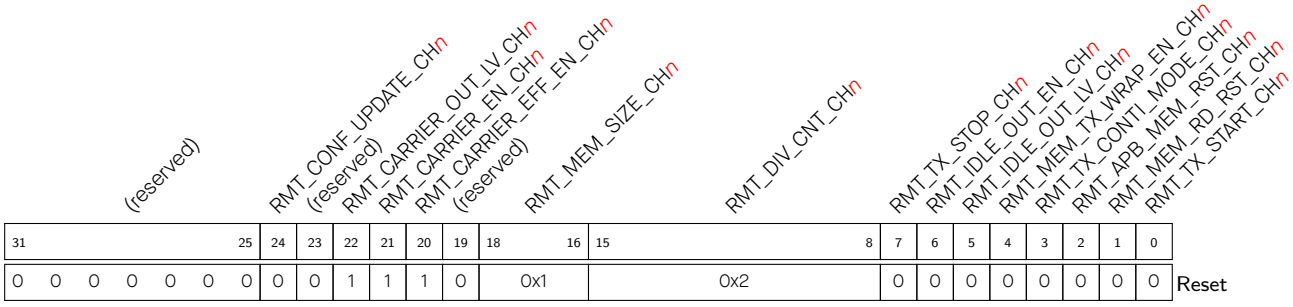


RMT_CH*n*DATA Write data for channel *n* via APB FIFO. (WO)

Register 37.2. RMT_CH*m*DATA_REG (*m*: 2-3) (0x0000+0x4m*)**

RMT_CH*m*DATA Read data for channel *m* via APB FIFO. (RO)

Register 37.3. RMT_CH*n*CONFO_REG (*n*: 0-1) (0x0010+0x4**n*)



RMT_TX_START_CH*n* Configures whether to enable sending data in channel *n*.

0: No effect

1: Enable

(WT)

RMT_MEM_RD_RST_CH*n* Configures whether to reset RAM read address accessed by the transmitter for channel *n*.

0: No effect

1: Reset

(WT)

RMT_APB_MEM_RST_CH*n* Configures whether to reset RAM W/R address accessed by APB FIFO for channel *n*.

0: No effect

1: Reset

(WT)

RMT_TX_CONTI_MODE_CH*n* Configures whether to enable continuous TX mode for channel *n*.

0: No Effect

1: Enable

In this mode, the transmitter starts transmission from the first data. If an end-marker is encountered, the transmitter starts transmitting data from the first data again; if no end-marker is encountered, the transmitter starts transmitting the first data again when the last data is transmitted.

(R/W)

RMT_MEM_TX_WRAP_EN_CH*n* Configures whether to enable wrap TX mode for channel *n*.

0: No effect

1: Enable

In this mode, if the TX data size is larger than the channel's RAM block size, the transmitter continues transmitting the first data to the last data in loops.

(R/W)

RMT_IDLE_OUT_LV_CH*n* Configures the level of output signal for channel *n* when the transmitter is in idle state. (R/W)

Continued on the next page...

Register 37.3. RMT_CH*n*CONF0_REG (*n*: 0-1) (0x0010+0x4**n*)

Continued from the previous page...

RMT_IDLE_OUT_EN_CH*n* Configures whether to enable the output for channel *n* in idle state.

0: No effect

1: Enable

(R/W)

RMT_TX_STOP_CH*n* Configures whether to stop the transmitter of channel *n* sending data out.

0: No effect

1: Stop

(R/W/SC)

RMT_DIV_CNT_CH*n* Configures the divider for clock of channel *n*.

Measurement unit: rmt_sclk

(R/W)

RMT_MEM_SIZE_CH*n* Configures the maximum number of memory blocks allocated to channel *n*.

(R/W)

RMT_CARRIER_EFF_EN_CH*n* Configures whether to add carrier modulation on the output signal only at data-sending state for channel *n*.

0: Add carrier modulation on the output signal at data-sending state and idle state for channel *n*

1: Add carrier modulation on the output signal only at data-sending state for channel *n*

Only valid when RMT_CARRIER_EN_CH*n* is 1.

(R/W)

RMT_CARRIER_EN_CH*n* Configures whether to enable the carrier modulation on output signal for channel *n*.

0: Disable

1: Enable

(R/W)

RMT_CARRIER_OUT_LV_CH*n* Configures the position of carrier wave for channel *n*.

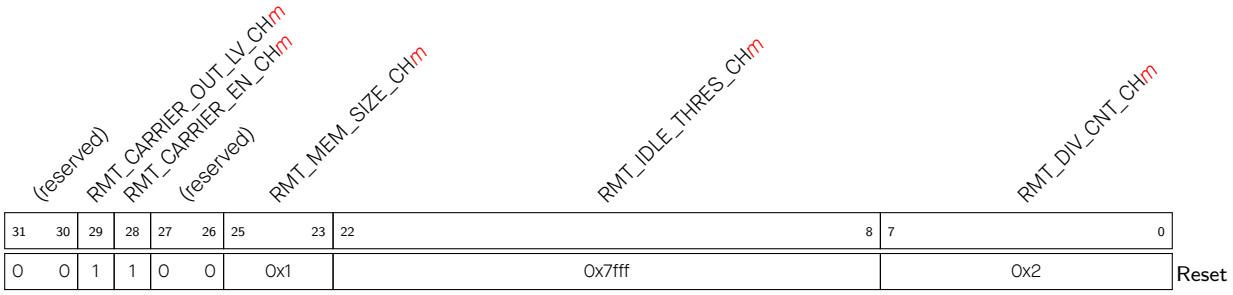
0: Add carrier wave on low level

1: Add carrier wave on high level

(R/W)

RMT_CONF_UPDATE_CH*n* Synchronization bit for channel *n*. (WT)

Register 37.4. RMT_CH*m*CONF0_REG (*m*: 2-3) (0x0008+0x8**m*)



RMT_DIV_CNT_CH*m* Configures the clock divider of channel *m*.

Measurement unit: rmt_sclk

(R/W)

RMT_IDLE_THRES_CH*m* Configures RX threshold.

When no edge is detected on the input signal for continuous clock cycles longer than this field value, the receiver stops receiving data.

Measurement unit: clk_div

(R/W)

RMT_MEM_SIZE_CH*m* Configures the maximum number of memory blocks allocated to channel *m*.

(R/W)

RMT_CARRIER_EN_CH*m* Configures whether to enable carrier modulation on output signal for channel *m*.

0: Disable

1: Enable

(R/W)

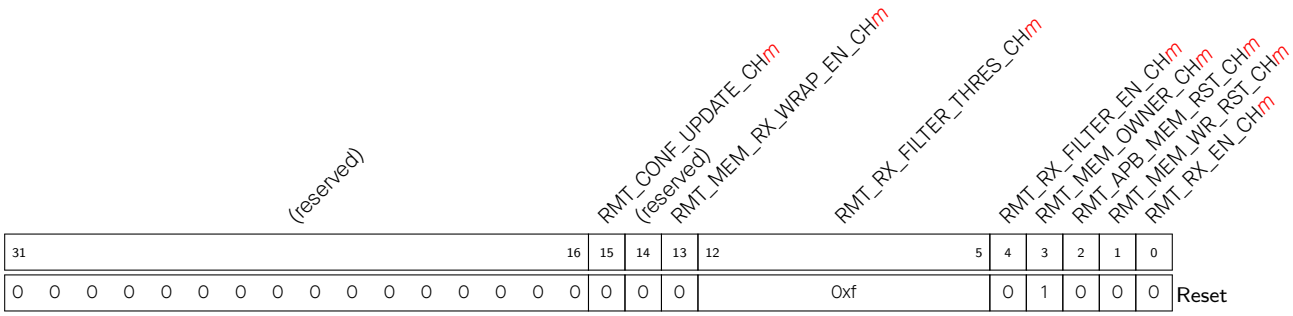
RMT_CARRIER_OUT_LV_CH*m* Configures the position of carrier wave for channel *m*.

0: Modulate (TX) or demodulate (RX) carrier wave on low level

1: Modulate (TX) or demodulate (RX) carrier wave on high level

(R/W)

Register 37.5. RMT_CH*m*CONF1_REG (*m*: 2-3) (0x000C+0x8**m*)



RMT_RX_EN_CH*m* Configures whether to enable the receiver to start receiving data in channel *m*.

0: Disable

1: Enable

(R/W)

RMT_MEM_WR_RST_CH*m* Configures whether to reset RAM write address accessed by the receiver for channel *m*.

0: No effect

1: Reset

(WT)

RMT_APB_MEM_RST_CH*m* Configures whether to reset RAM W/R address accessed by APB FIFO for channel *m*.

0: No effect

1: Reset

(WT)

RMT_MEM_OWNER_CH*m* Configures the ownership of channel *m*'s RAM block.

0: APB bus is using the RAM

1: Receiver is using the RAM

(R/W/SC)

RMT_RX_FILTER_EN_CH*m* Configures whether to enable the receiver's filter for channel *m*.

0: Disable

1: Enable

(R/W)

RMT_RX_FILTER_THRES_CH*m* Configures whether the receiver, when receiving data, ignores the input pulse when its width is shorter than this register value in units of rmt_sclk cycles.

0: No effect

1: Reset

(R/W)

RMT_MEM_RX_WRAP_EN_CH*m* Configures whether to enable wrap RX mode for channel *m*.

0: Disable

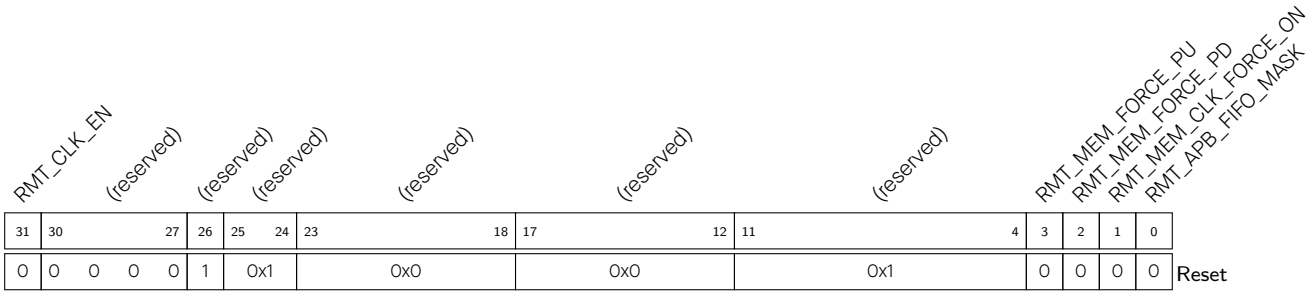
1: Enable

In this mode, if the RX data size is larger than channel *m*'s RAM block size, the receiver stores the RX data from the first address to the last address in loops.

(R/W)

RMT_CONF_UPDATE_CH*m* Synchronization bit for channel *m*. (WT)

Register 37.6. RMT_SYS_CONF_REG (0x0068)



RMT_APB_FIFO_MASK Configures the memory access mode.

0: Access memory by FIFO

1: Access memory directly

(R/W)

RMT_MEM_CLK_FORCE_ON Configures whether to enable the clock for RMT memory.

0: Disable

1: Enable

(R/W)

RMT_MEM_FORCE_PD Configures whether to power down RMT memory.

0: No effect

1: Power down

(R/W)

RMT_MEM_FORCE_PU Configures whether to disable the power-down function of RMT memory in Light-sleep.

0: Power down RMT memory when RMT is in Light-sleep mode

1: Disable the power-down function of RMT memory in Light-sleep

(R/W)

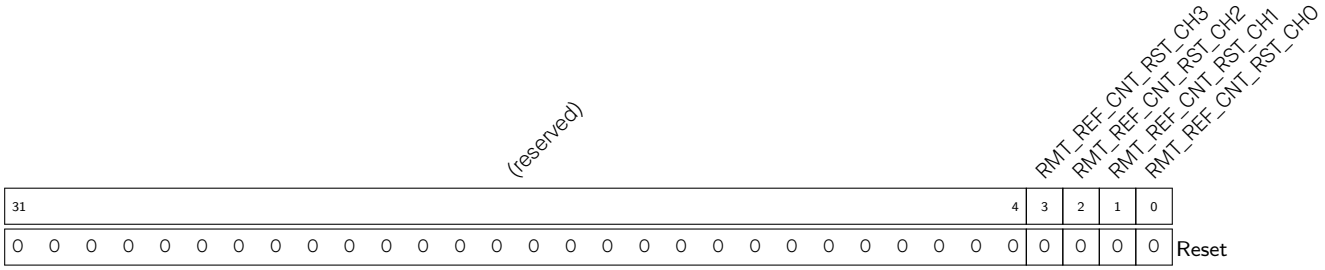
RMT_CLK_EN Configures whether to enable signal of RMT register clock gate.

0: Power down the drive clock of registers

1: Power up the drive clock of registers

(R/W)

Register 37.7. RMT_REF_CNT_RST_REG (0x0070)



RMT_REF_CNT_RST_CH*n* Configures whether to reset the clock divider of channel *n*.

0: No effect

1: Reset

(WT)

RMT_REF_CNT_RST_CH*m* Configures whether to reset the clock divider of channel *m*.

0: No effect

1: Reset

(WT)

Register 37.8. RMT_CH*n*STATUS_REG (*n*: 0-1) (0x0028+0x4**n*)

RMT_MEM_RADDR_EX_CH*n* Represents the memory address offset when transmitter of channel *n* is using the RAM. (RO)

RMT_STATE_CH*n* Represents the FSM status of channel *n*. (RO)

RMT_APB_MEM_WADDR_CH*n* Represents the memory address offset when writes RAM over APB bus. (RO)

RMT_APB_MEM_RD_ERR_CH*n* Represents whether the offset address exceeds memory size when reading via APB bus.
0: Not exceed
1: Exceed
(RO)

RMT_MEM_EMPTY_CH*n* Represents whether the TX data size exceeds the memory size and the wrap TX mode is disabled.
0: Not exceed
1: Exceed
(RO)

RMT_APB_MEM_WR_ERR_CH*n* Represents whether the offset address exceeds memory size (overflows) when writes via APB bus.
0: Not exceed
1: Exceed
(RO)

RMT_APB_MEM_RADDR_CH*n* Represents the memory address offset when reading RAM over APB bus. (RO)

Register 37.9. RMT_CH*m*STATUS_REG (*m*: 2-3) (0x0028+0x4**m*)

(reserved)	RMT_APB_MEM_RD_ERR_CH*m*	RMT_MEM_FULL_CH*m*	RMT_MEM_OWNER_ERR_CH*m*	RMT_STATE_CH*m*	(reserved)	RMT_APB_MEM_RADDR_CH*m*	(reserved)	RMT_MEM_WADDR_EX_CH*m*	
31 28	27	26	25	24 22	21	20 12	11 9	8 0	
0 0 0 0	0	0	0	0	0	0	0 0 0	0	Reset

RMT_MEM_WADDR_EX_CH*m* Represents the memory address offset when receiver of channel *m* is using the RAM. (RO)

RMT_APB_MEM_RADDR_CH*m* Represents the memory address offset when reads RAM over APB bus. (RO)

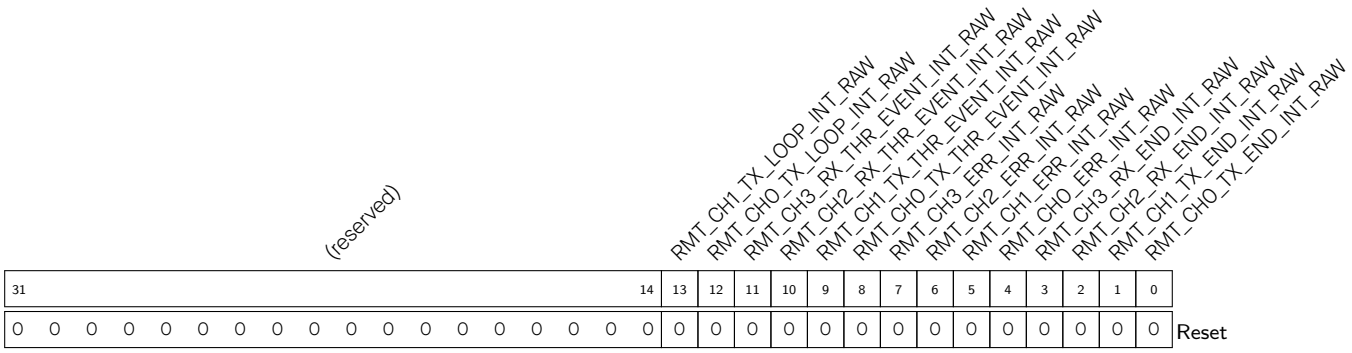
RMT_STATE_CH*m* Represents the FSM status of channel *m*. (RO)

RMT_MEM_OWNER_ERR_CH*m* Represents whether the ownership of memory block is wrong.
0: The ownership of memory block is correct
1: The ownership of memory block is wrong
(RO)

RMT_MEM_FULL_CH*m* Represents whether the receiver receives more data than the memory can fit.
0: The receiver does not receive more data than the memory can fit
1: The receiver receives more data than the memory can fit
(RO)

RMT_APB_MEM_RD_ERR_CH*m* Represents whether the offset address exceeds memory size (overflows) when reads RAM via APB bus.
0: Not exceed
1: Exceed
(RO)

Register 37.10. RMT_INT_RAW_REG (0x0038)



RMT_CH*n*_TX_END_INT_RAW The raw interrupt status of RMT_CH*n*_TX_END_INT . Triggered when the transmission is done. (R/WTC/SS)

RMT_CH*m*_RX_END_INT_RAW The raw interrupt status of RMT_CH*n*_RX_END_INT. Triggered when the reception is done. (R/WTC/SS)

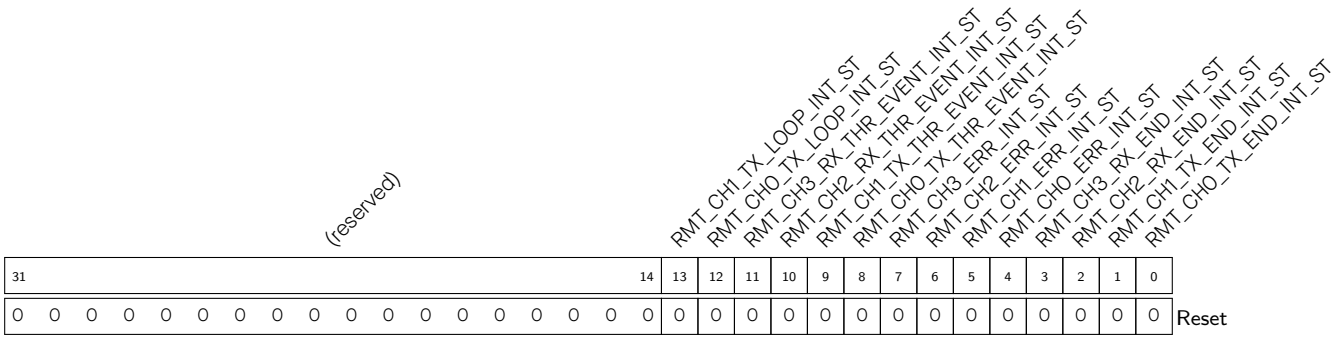
RMT_CH*n*/*m*_ERR_INT_RAW The raw interrupt status of RMT_CH*n*/*m*_ERR_INT. Triggered when error occurs. (R/WTC/SS)

RMT_CH*n*_TX_THR_EVENT_INT_RAW The raw interrupt status of RMT_CH*n*_TX_THR_EVENT_INT. Triggered when the transmitter sent more data than the configured value. (R/WTC/SS)

RMT_CH*m*_RX_THR_EVENT_INT_RAW The raw interrupt status of RMT_CH*m*_RX_THR_EVENT_INT. Triggered when the receiver receives more data than the configured value. (R/WTC/SS)

RMT_CH*n*_TX_LOOP_INT_RAW The raw interrupt status of RMT_CH*n*_TX_LOOP_INT. Triggered when the loop count reaches the configured threshold value. (R/WTC/SS)

Register 37.11. RMT_INT_ST_REG (0x003C)



RMT_CH*n*_TX_END_INT_ST The masked interrupt status of RMT_CH*n*_TX_END_INT . (RO)

RMT_CH*m*_RX_END_INT_ST The masked interrupt status of RMT_CH*n*_RX_END_INT. (RO)

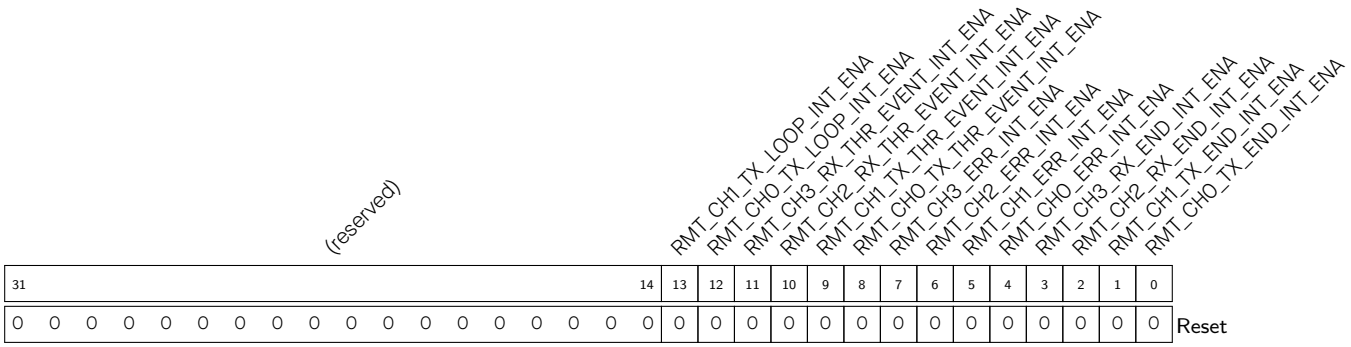
RMT_CH*n*/*m*_ERR_INT_ST The masked interrupt status of RMT_CH*n*/*m*_ERR_INT. (RO)

RMT_CH*n*_TX_THR_EVENT_INT_ST The masked interrupt status of RMT_CH*n*_TX_THR_EVENT_INT. (RO)

RMT_CH*m*_RX_THR_EVENT_INT_ST The masked interrupt status of RMT_CH*m*_RX_THR_EVENT_INT. (RO)

RMT_CH*n*_TX_LOOP_INT_ST The masked interrupt status of RMT_CH*n*_TX_LOOP_INT. (RO)

Register 37.12. RMT_INT_ENA_REG (0x0040)



RMT_CH*n*_TX_END_INT_ENA Write 1 to enable RMT_CH*n*_TX_END_INT . (R/W)

RMT_CH*m*_RX_END_INT_ENA Write 1 to enable RMT_CH*n*_RX_END_INT. (R/W)

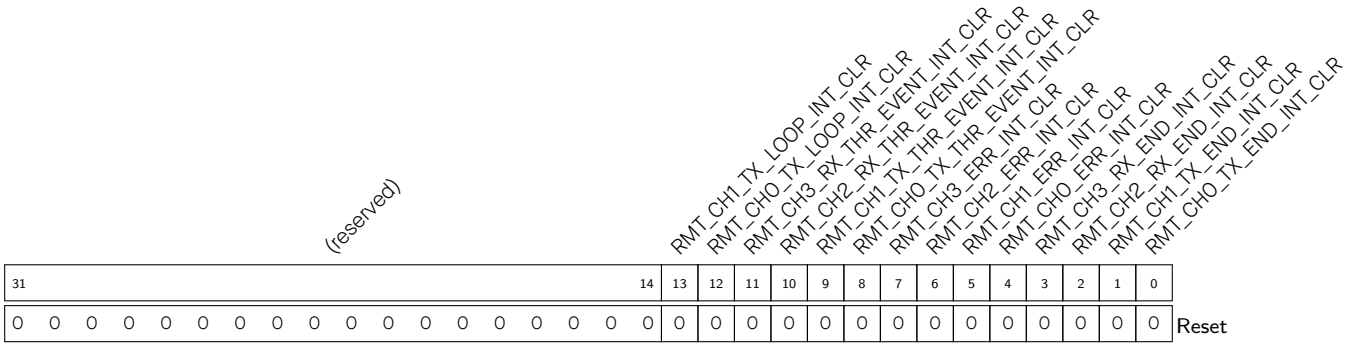
RMT_CH*n*/*m*_ERR_INT_ENA Write 1 to enable RMT_CH*n*/*m*_ERR_INT. (R/W)

RMT_CH*n*_TX_THR_EVENT_INT_ENA Write 1 to enable RMT_CH*n*_TX_THR_EVENT_INT. (R/W)

RMT_CH*m*_RX_THR_EVENT_INT_ENA Write 1 to enable RMT_CH*m*_RX_THR_EVENT_INT. (R/W)

RMT_CH*n*_TX_LOOP_INT_ENA Write 1 to enable RMT_CH*n*_TX_LOOP_INT. (R/W)

Register 37.13. RMT_INT_CLR_REG (0x0044)



RMT_CH*n*_TX_END_INT_CLR Write 1 to clear RMT_CH*n*_TX_END_INT . (WT)

RMT_CH*m*_RX_END_INT_CLR Write 1 to clear RMT_CH*n*_RX_END_INT. (WT)

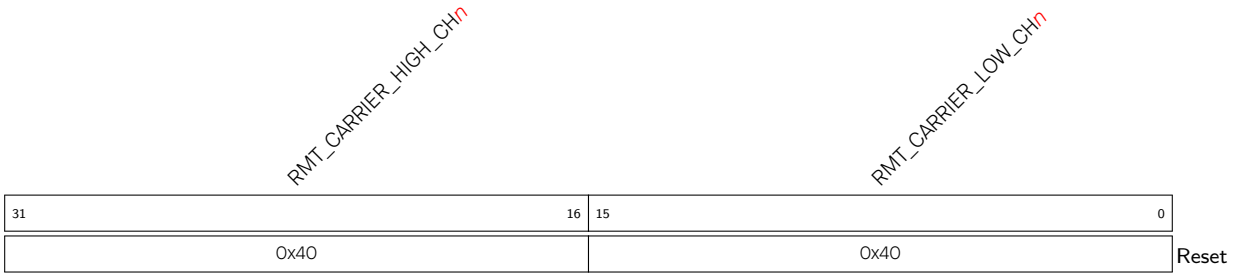
RMT_CH*n*/*m*_ERR_INT_CLR Write 1 to clear RMT_CH*n*/*m*_ERR_INT. (WT)

RMT_CH*n*_TX_THR_EVENT_INT_CLR Write 1 to clear RMT_CH*n*_TX_THR_EVENT_INT. (WT)

RMT_CH*m*_RX_THR_EVENT_INT_CLR Write 1 to clear RMT_CH*m*_RX_THR_EVENT_INT. (WT)

RMT_CH*n*_TX_LOOP_INT_CLR Write 1 to clear RMT_CH*n*_TX_LOOP_INT. (WT)

Register 37.14. RMT_CH*n*CARRIER_DUTY_REG (*n*: 0-1) (0x0048+0x4**n*)



RMT_CARRIER_LOW_CH*n* Configures carrier wave's low level clock period for channel *n*.

Measurement unit: rmt_sclk

(R/W)

RMT_CARRIER_HIGH_CH*n* Configures carrier wave's high level clock period for channel *n*.

Measurement unit: rmt_sclk

(R/W)

Register 37.15. RMT_CH*m*_RX_CARRIER_RM_REG (*m*: 2-3) (0x0048+0x4m*)**

RMT_CARRIER_LOW_THRES_CH*m* Configures the low level period in a carrier modulation mode for channel *m*.

The low level period in a carrier modulation mode is (RMT_CARRIER_LOW_THRES_CH*m* + 1) for channel *m*.

Measurement unit: clk_div

(R/W)

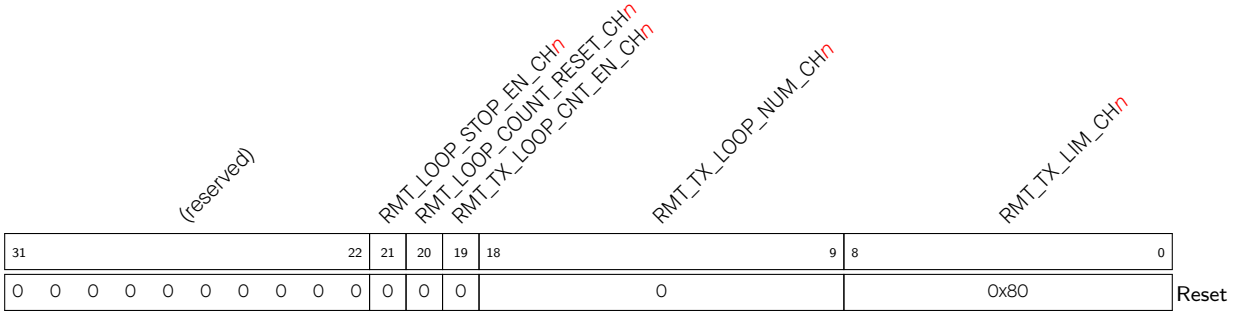
RMT_CARRIER_HIGH_THRES_CH*m* Configures the high level period in a carrier modulation mode for channel *m*.

The high level period in a carrier modulation mode is (REG_RMT_REG_CARRIER_HIGH_THRES_CH*m* + 1) for channel *m*.

Measurement unit: clk_div

(R/W)

Register 37.16. RMT_CH*n*_TX_LIM_REG (*n*: 0-1) (0x0058+0x4**n*)



RMT_TX_LIM_CH*n* Configures the maximum entries that channel *n* can send out. (R/W)

RMT_TX_LOOP_NUM_CH*n* Configures the maximum loop count when Continuous TX mode is valid. (R/W)

RMT_TX_LOOP_CNT_EN_CH*n* Configures whether to enable loop count.

0: No effect

1: Enable

(R/W)

RMT_LOOP_COUNT_RESET_CH*n* Configures whether to reset the loop count when tx_conti_mode is valid.

0: No effect

1: Reset

(WT)

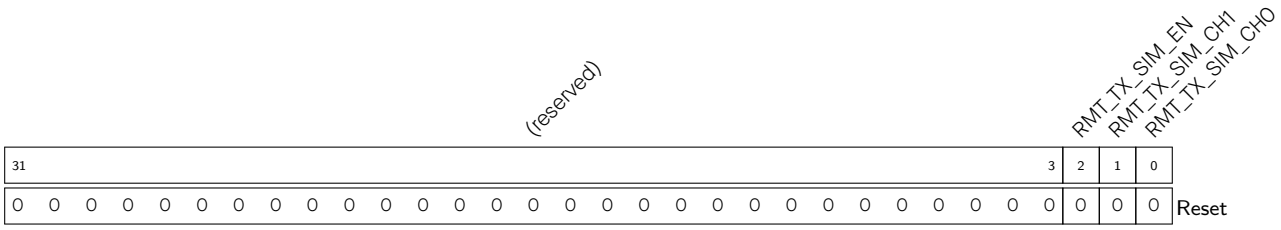
RMT_LOOP_STOP_EN_CH*n* Configures whether to enable the loop send stop function after the loop counter counts to loop number for channel *n*.

0: No effect

1: Enable

(R/W)

Register 37.17. RMT_TX_SIM_REG (0x006C)



RMT_TX_SIM_CH*n* Configures whether to enable channel *n* to start sending data synchronously with other enabled channels.

0: No effect

1: Enable

(R/W)

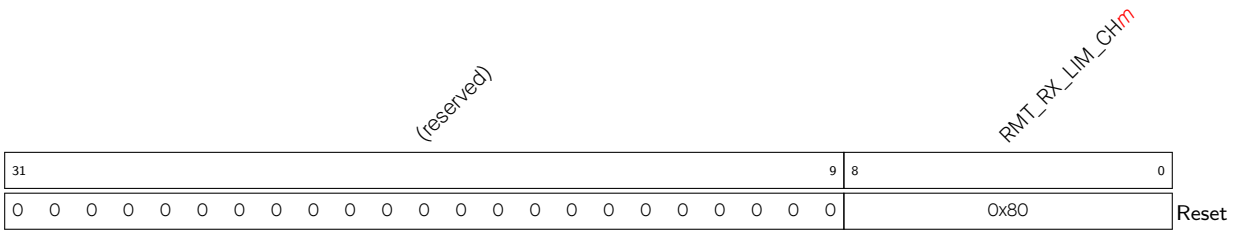
RMT_TX_SIM_EN Configures whether to enable multiple of channels to start sending data synchronously.

0: No effect

1: Enable

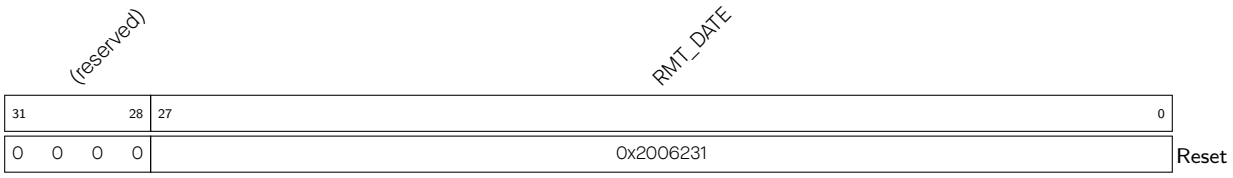
(R/W)

Register 37.18. RMT_CH*m*_RX_LIM_REG (*m*: 2-3) (0x0058+0x4*m)



RMT_RX_LIM_CH*m* Configures the maximum entries that channel *m* can receive. (R/W)

Register 37.19. RMT_DATE_REG (0x00CC)



RMT_DATE Version control register. (R/W)

Chapter 38

Parallel IO Controller (PARLIO)

38.1 Introduction

ESP32-H2 contains a Parallel IO controller (PARLIO) capable of transferring data between external devices and internal memory on a parallel bus through GDMA. The PARLIO consists of a TX unit and an RX unit, serving as a transmitter and a receiver respectively. With the two units combined, PARLIO achieves full-duplex communication.

Due to its flexibility, PARLIO can function as a general interface to connect various peripherals. For example, with SPI as the master device and PARLIO as the slave device, a peer-to-peer transfer can be achieved. For detailed application examples, refer to Section 38.7.

38.2 Glossary

This section covers terminology used to describe the functionality of PARLIO.

RX unit	Module in PARLIO responsible for receiving data from the external parallel bus and storing them into internal memory.
TX unit	Module in PARLIO responsible for transmitting data from internal memory to external parallel bus.
RXD	Parallel data received from the IO interface of the RX unit.
TXD	Parallel data sent from the IO interface of the TX unit.
Frame	Transferred data unit from the moment the START signal is set to the moment the End of Frame (EOF) signal is received.
Free-running clock	Clock that toggles continuously as opposed to clock that only toggles when valid data is incoming, and remains constant for the rest of the time.
GDMA SUC EOF	Signal that indicates GDMA successful end of frame. When GDMA receives this signal, a GDMA interrupt will be triggered, indicating that the current frame is correct and the receive is finished.
GDMA ERR EOF	Signal that indicates GDMA error end of frame. When GDMA receives this signal, a GDMA interrupt will be triggered, indicating that the current frame has error and the receive is finished.
CDC	Clock domain crossing.

38.3 Features

The PARLIO module has the following main features:

- Various clock sources:
  - Including external IO clock PAD_CLK_TX/RX and internal system clock XTAL_CLK, PLL_F96M_CLK, and RC_FAST_CLK
  - Maximum clock frequency of 40 MHz
  - Integer clock frequency division
- 1/2/4/8-bit configurable data bus width
- Full-duplex communication with 8-bit data bus width
- Bit reversal when data bus width is 1/2/4-bit
- RX unit for receiving IO parallel data, which supports:
  - Output clock gating
  - RX unit input and output clock inverse
  - Various receive modes
  - Configurable GDMA SUC EOF generation
  - Configurable IO pin of external enable signal
- TX unit for sending IO parallel data, which supports:
  - Unlimited length data transmission
  - Output clock gating
  - TX unit input and output clock inverse
  - Valid signal output
  - Configurable bus idle value

38.4 Architectural Overview

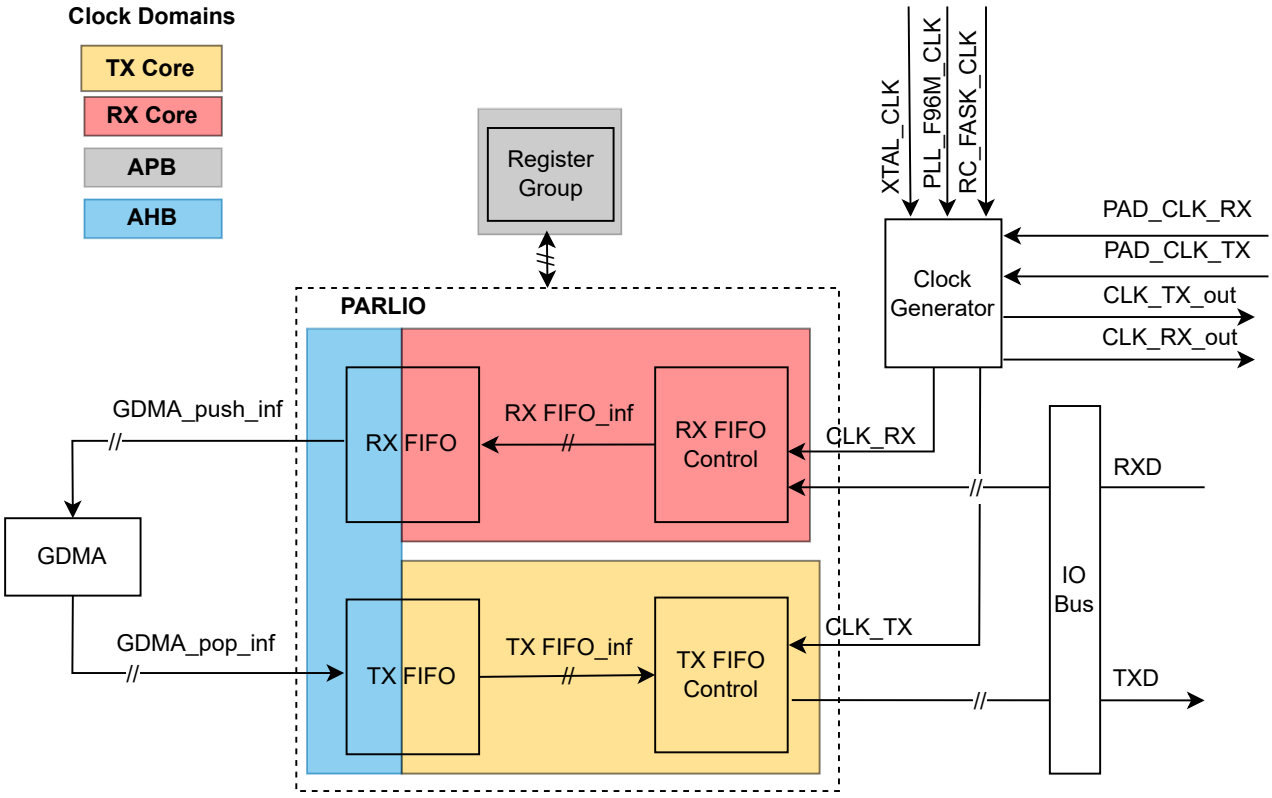


Figure 38.4-1. PARLIO Architecture

Figure 38.4-1 shows the architecture of PARLIO. In addition to the RX unit and the TX unit, a group of status configuration registers is also included.

The RX unit converts RXD into an asynchronous FIFO interface, which synchronizes RXD to the AHB clock domain. RXD is then converted to a standard GDMA interface and sent to the internal memory.

The TX unit fetches data from internal memory through GDMA and converts the GDMA interface into an asynchronous FIFO interface. The asynchronous FIFO synchronizes the data to the TX Core clock domain and converts the data to TXD for parallel IO bus output.

38.5 Functional Description

38.5.1 Clock Generator

There are four input clock domains in PARLIO, namely, RX Core, TX Core, AHB, and APB, as shown in Figure 38.4-1.

The status configuration register group works in the APB clock domain.

The GDMA interface logic works in the AHB clock domain.

RX Core and TX Core clock domains each have four clock sources for selection, i.e., the internal system clock sources XTAL_CLK, RC_FAST_CLK, PLL_F96M_CLK and the external clock source (PAD_CLK_TX/RX), as

shown in Figure 38.5-1. Clock sources can be selected by configuring PCR_PARL_CLK_RX_SEL and PCR_PARL_CLK_TX_SEL. The clock can be divided by configuring PCR_PARL_CLK_RX_DIV_NUM and PCR_PARL_CLK_TX_DIV_NUM. The clock division factor can be configured up to $(2^{16} - 1)$.

The input clock of the TX and RX units can be inverted. The operating clock of the TX and RX units can also be inverted before being output to IO. The TX and RX units also support clock gating of the output clock.

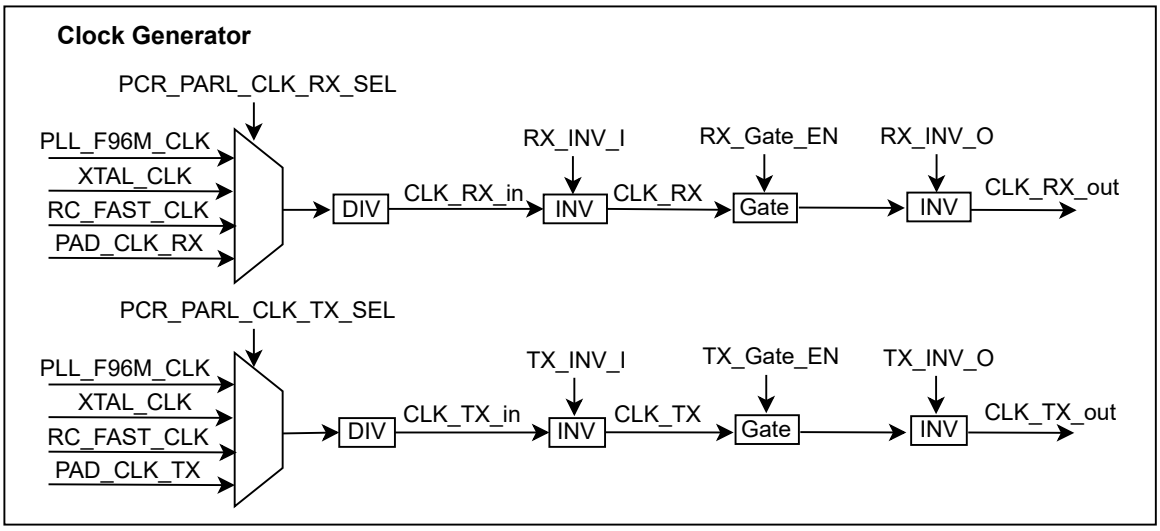


Figure 38.5-1. PARLIO Clock Generation

38.5.2 Clock & Reset Restriction

Due to the versatility of PARLIO, the PAD clocks (PAD_CLK_TX/RX) of PARLIO may come from different masters (external devices or internal clock sources). These clocks might be either free-running clock or not. If the clock is not free-running, some internal control signals of PARLIO cannot process CDC, so there are certain restrictions during the operation.

1. During the reset of the asynchronous FIFO, it takes two clock cycles to synchronize within AHB clock domain and Core clock domain. Therefore, if the reset of AHB clock domain is performed with a clock that is not free-running, the reset synchronization must be performed two clock cycles in advance. The specific operation is shown in the table below.

Table 38.5-1. Operations to Reset AHB Clock Domain with Clock Restrictions

Clock Restriction		Specific Operation
The Current Frame	The Next Frame	
Free-running clock	Not free-running clock	Users can reset the next frame transfer before switching to the clock that is not free-running. After the reset is completed, users can switch the clock.
Not free-running clock	Free-running clock	The next frame can be reset freely. Users only need to ensure that there is an interval of two clock cycles between the reset and the start of the transfer.
Not free-running clock	Not free-running clock	If the next frame transfer needs to be reset, users need to first switch to the internal free-running clock, and then switch to the actual clock after the reset is completed.

2. Due to the restrictions caused by a clock that is not free-running, PARL_IO_RX_START and PARL_IO_TX_START cannot perform CDC processing. Therefore, it is necessary to wait until PARL_IO_RX_START and PARL_IO_TX_START are stable before starting the data transfer. Otherwise, the transfer might enter a metastable state.

   Here are the specific operation steps for the RX unit:

   - Clear PCR_PARL_CLK_RX_EN to turn off RX Core clock domain;
   - Write 1 to PARL_IO_RX_START;
   - Set PCR_PARL_CLK_RX_EN to turn on RX Core clock domain;
   - Operate the external device to start sending data;
   - Clear PCR_PARL_CLK_RX_EN to turn off RX Core clock domain;
   - Write 0 to PARL_IO_RX_START.

   Here are the specific operation steps for the TX unit:

   - Clear PCR_PARL_CLK_TX_EN to turn off TX Core clock domain;
   - Write 1 to PARL_IO_TX_START;
   - Set PCR_PARL_CLK_TX_EN to turn on TX Core clock domain;
   - Operate the external device to start receiving data;
   - Clear PCR_PARL_CLK_TX_EN to turn off TX Core clock domain;
   - Write 0 to PARL_IO_TX_START.

3. Reset should follow the requirements below:

   - The clock reset during the chip start-up should follow the sequence below:
     (a) Reset APB clock domain;
     (b) Reset AHB clock domain;
     (c) Reset Core clock domain.
   - Inter-frame transfer requires Core clock domain reset and async FIFO reset.

38.5.3 Master-Slave Mode

The TX and RX units can function as both master and slave.

When the TX unit serves as master, it is necessary to set the internal free-running clock as the clock source. The TX unit drives TXD on the rising edge of the clock.

When the TX unit functions as a slave device, there are three scenarios, as shown in the table below.

Table 38.5-2. Requirements for TX Unit Operating as Slave with Clock Restrictions

Clock Restriction		Requirement
Clock Sent by Master	Clock Waveform	
Free-running clock	Any waveform	There is no requirement for the sampling edge of the master clock.
Not free-running clock	Positive waveform (Figure 38.5-2)	The master clock should sample TXD at the falling edge.
Not free-running clock	Negative waveform (Figure 38.5-3)	The master device should invert the original clock and convert it to the waveform as Figure 38.5-2 shows before output.

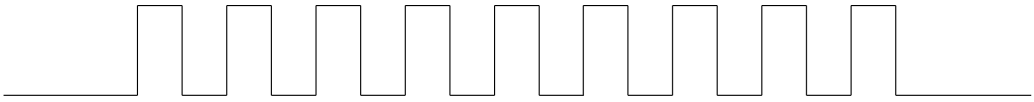


Figure 38.5-2. Positive Waveform

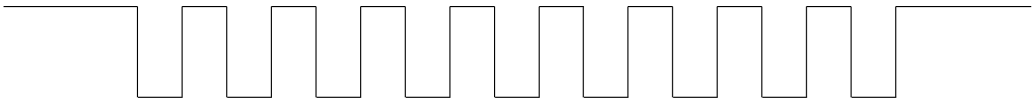


Figure 38.5-3. Negative Waveform

When the RX unit serves as the master, it is required to set the clock source as the internal free-running clock. The RX unit drives RXD on the rising edge of the clock.

When the RX unit functions as the slave, there are three scenarios, as shown in the table below.

Table 38.5-3. Requirements for RX Unit Operating as Slave with Clock Restrictions

Clock Restriction		Requirement
Clock Sent by Master	Clock Waveform	
Free-running clock	Any waveform	There is no requirement for the sampling edge of the master clock, and the valid data is subject to the external enable signal.
Not free-running clock	Positive waveform (Figure 38.5-2)	It is required for the master device to drive the data at the rising edge and the RX unit to sample the data at the falling edge (i.e., to inverse the master clock).
Not free-running clock	Negative waveform (Figure 38.5-3)	It is required for the master device to drive the data at the falling edge and the RX unit to sample the data at the rising edge (i.e., to use the original master clock).

38.5.4 Receive Modes of the RX Unit

PARLIO supports eight receive modes, which can be divided into three major categories according to the enable signal:

- Level Enable mode: data received is enabled by the external signal level;

- Pulse Enable mode: data received is enabled by the external signal pulse;
- Software Enable mode: the enable signal of data received can be configured by users directly.

The RX unit also supports inverse of the external enable signal. If the external enable signal is active-low, users can enable the function by setting PARL_IO_RX_EXT_EN_INV to switch to the corresponding receive mode introduced as follows.

38.5.4.1 Level Enable Mode

Figure 38.5-4 shows the Level Enable mode. In this mode, an active level on the external enable signal must be aligned with valid data. Since the external level enable signal occupies one IO pin, there are at most seven IO pins left usable for RXD.

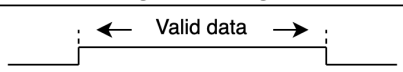
Mode	Sub-mode	Description
LEVEL_ENABLE	\	***signal level high*** 

Figure 38.5-4. Sub-Modes of Level Enable Mode for RX Unit

38.5.4.2 Pulse Enable Mode

Pulse Enable mode can be divided into six sub-modes depending on the pulse active level and its alignment with valid data. For detailed classification, see Figure 38.5-5.

Sub-modes 1 ~ 4 all contain start pulse and end pulse. The difference lies in whether start pulse and end pulse are aligned with valid data.

Sub-modes 5 ~ 6 only contain start pulse and the end of valid data is signaled by configuring PARL_IO_RX_BITLEN.

Since the external pulse enable signal occupies one IO pin, there are at most seven IO pins left usable for RXD. However, in sub-mode 5 and sub-mode 6, as the data is considered valid after the pulse's first edge and before the pulse's last edge, the enable signal IO pin can serve as a data IO pin at the same time. Therefore, there are eight IO pins usable for RXD in these two sub-modes.

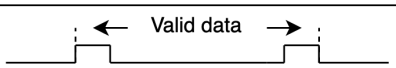
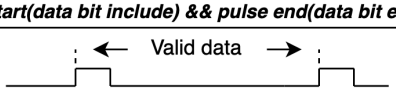
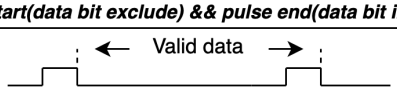
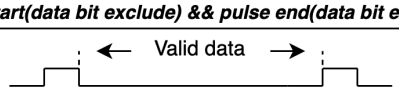
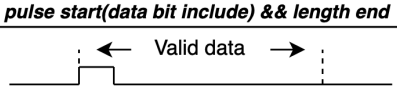
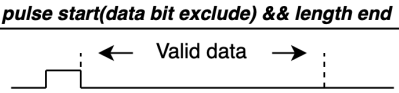
Mode	Sub-mode	Description
PULSE_ENABLE	sub-mode1	*pulse start(data bit include) && pulse end(data bit include)* 
	sub-mode2	*pulse start(data bit include) && pulse end(data bit exclude)* 
	sub-mode3	*pulse start(data bit exclude) && pulse end(data bit include)* 
	sub-mode4	*pulse start(data bit exclude) && pulse end(data bit exclude)* 
	sub-mode5	*pulse start(data bit include) && length end* 
	sub-mode6	*pulse start(data bit exclude) && length end* 

Figure 38.5-5. Sub-Modes of Pulse Enable Mode for RX Unit

38.5.4.3 Software Enable Mode

The enable signal in Software Enable mode is determined by the internal configuration register. If users switch to this mode, the receive will only be activated when both PARL_IO_RX_SW_EN and PARL_IO_RX_START are set to 1.

Since the enable signal does not occupy IO pins on the interface, there are at most eight IO pins usable by the RXD. Due to the differences of clock domains, the enable signal cannot be aligned with valid data. Thus, the validity of data needs to be identified by the valid clock edge. In this case, the RX Core clock needs to be aligned with valid data.

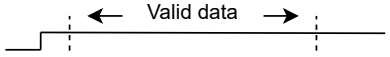
Mode	Sub-mode	Description
SW_ENABLE	/	/ 

Figure 38.5-6. Sub-Mode of Software Enable Mode for RX Unit

38.5.5 RX Unit GDMA SUC EOF Generation

The RX unit generates a GDMA SUC EOF signal to indicate the end of current frame transfer and sends it to the GDMA interface. GDMA SUC EOF can be generated by an external enable signal or triggered by the internally configured bit length.

- When GDMA SUC EOF is generated by the internally configured bit length, there is no restriction on the receive mode selection. However, PARL_IO_RX_BITLEN must be configured. If the configured value of PARL_IO_RX_BITLEN is less than the actual received data, the GDMA SUC EOF will be triggered in

advance. In this case, the RX unit stops reading data from the FIFO, but the FIFO continues to receive external data until an RX_FIFO_WOVF_INT interrupt is triggered.

- When GDMA SUC EOF is generated by the external enable signal, only sub-modes 1 and 3 of Pulse Enable mode can be selected. In this mode, the transfer is not affected by the value of PARL_IO_RX_BITLEN, and the transferred data length of the frame is not limited.

38.5.6 RX Unit Timeout

The RX unit supports the receive timeout. If the RX FIFO has not been receiving valid data for a long time, the timeout will be triggered. In such case, a GDMA ERR EOF signal will be generated and sent to the GDMA interface to indicate the end of the receiving. Configure PARL_IO_RX_TIMEOUT_THRES to set the timeout threshold.

The timeout function is enabled by default and can be disabled by users. The upper threshold of the configurable timeout is $(2^{16} - 1)$ cycles of AHB clock domain, and the lower threshold depends on the relative frequency relationship between AHB clock domain and RX Core clock domain. It is recommended to set a relatively large value for PARL_IO_RX_TIMEOUT_THRES to avoid undesired GDMA ERR EOF signals.

38.5.7 Unlimited Length Data Transmission of TX Unit

By default, the end of transmission for the TX unit is determined by the configured value of PARL_IO_TX_BITLEN. When the transmitted data reaches the configured value, the system generates a GDMA SUC EOF interrupt signal, indicating the completion of the current data frame transmission. This approach limits the maximum length of a single frame transmission to the bit width defined by PARL_IO_TX_BITLEN, preventing it from achieving unlimited length data transmission.

To enable larger single-frame transmissions, set PARL_IO_TX_EOF_GEN_SEL to 1. In this mode, the hardware uses the EOF configured in the GDMA inlink list as the end-of-transmission signal for a single frame. As a result, data transmission is no longer constrained by the bit width of PARL_IO_TX_BITLEN, allowing for significantly larger data frames. This flexibility ensures that the system can adapt to various application scenarios requiring extended data transmission capabilities.

38.5.8 Output Clock Gating of TX Unit

The TX unit supports output clock gating. The clock gating function is disabled by default, and can be enabled by software via setting PARL_IO_TX_GATING_EN. The gating signal is fixed as the highest bit of TXD, and when the bit is at high level, the clock signal can be toggled. There are currently two configurable control sources for the highest bit of TXD, i.e., the DMA output data and the valid signal. Write 0 to PARL_IO_TX_VALID_OUTPUT_EN to select the DMA output data as the control source. In this case, it is required to configure the bit width of the TX unit to a maximum. Write 1 to PARL_IO_TX_VALID_OUTPUT_EN to select the valid signal as the control source. In this case, there is no limit to the bit width.

When the gating function is enabled, there are at most seven IO pins left usable for TXD as the gating signal occupies one IO.

38.5.9 Valid Signal Output of TX Unit

The TX unit can generate a valid signal aligned with TXD. Configure PARL_IO_TX_VALID_OUTPUT_EN to choose whether to output it to TXD. The polarity of the valid signal is fixed to active high.

The valid output function is disabled by default. When enabled, the output valid signal occupies the most significant bit (MSB) of the TXD, which means that no matter what the original value is, the 7th bit of TXD remains high and is output as the valid signal. However, the valid signal pin does not affect the bus width configuration. For example, if the data bus width is 1 bit, the valid output function can still be enabled with a fixed pin as TXD[7] while the data pin is TXD[0].

Note:
When the valid signal output function and the clock gating function are enabled at the same time, the highest bit of TXD (i.e., the gating signal) is continuously at high level. In cases where software requires the bit not to continuously stay at high level, do not enable the two functions simultaneously.

38.5.10 Bus Idle Value of TX Unit

The TX unit is regarded as in idle state when it is not transmitting data. It supports a configurable bus idle value.

The bus idle value is 0x0 by default, and its maximum configurable value is 0xFF. Note that the configured idle value should not conflict with other enabled functions. For example, when the MSB of TXD is used as the valid signal, users should avoid configuring the MSB of the idle value as 1.

38.5.11 Data Transfer in a Single Frame

The RX unit and the TX unit transfer data in the unit of bits, i.e., a single frame transfers 1 bit of data at least.

When the RX unit generates GDMA EOF signals through bit length, the maximum length of the single-frame transmission is $(2^{19} - 1)$ bits. When the RX unit generates GDMA EOF signals through the enable signal from an external device, there is no limit to the amount of bits of the single-frame transmission. The TX unit only generates GDMA EOF signals through bit length, so the maximum length of a single frame transmission is $(2^{19} - 1)$ bits.

Since PARLIO transfers data in the unit of bytes on the GDMA side, it will process the IO data when it is not aligned with the bytes.

- When receiving data, PARLIO will automatically pad 0 to the high bits of the data stored in memory via GDMA to make it aligned with bytes.
- When sending data, PARLIO will truncate the data retrieved from memory according to the configured value of PARL_IO_TX_BITLEN. Data exceeding the value will not be sent.

When the configured data bus width is 2/4/8-bit, the bit length must be configured as a multiple of the corresponding bus width.

38.5.12 Bit Reversal in One Byte

The sequence of data within one byte can be reversed when data bus width is 1/2/4-bit. Taking the RX unit as an example, when the configured bus width is 2 bits, the data needs to be packed into one byte before being written into the RX FIFO.

Presume that the original bit sequence is:

{ { b_0, b_1 }, { b_2, b_3 }, { b_4, b_5 }, { b_6, b_7 } }

If the bit reversal is enabled, the sequence will be reordered to:

{ { b_6, b_7 }, { b_4, b_5 }, { b_2, b_3 }, { b_0, b_1 } }

38.6 Programming Procedures

38.6.1 Data Receiving Operation Process

This section introduces the programming procedure for receiving data through RX unit. To receive parallel data from IO pins that are connected to external devices and store the data in the internal memory, perform the following procedure. For a detailed description of the clock and reset operation restrictions in the RX unit, refer to Section 38.5.2.

1. Reset the RX unit. For specific reset scenarios and sequences, refer to Section 38.5.2.
2. Set PARL_IO_RX_FIFO_WOVF_INT_CLR and PARL_IO_RX_FIFO_WOVF_INT_ENA.
3. Select the RXD IO pins. If a PAD clock is used, the clock IO pin also needs to be configured.
4. Select the clock source and divide the clock by configuring PCR registers.
5. Turn off the clock of RX Core clock domain.
6. Select the receive mode and enable functions required as described in Sections 38.3 and 38.5.
7. Configure GDMA inlink list.
8. Set PARL_IO_RX_REG_UPDATE to synchronize the register signals.
9. Set PARL_IO_RX_START.
10. Turn on the clock of RX Core clock domain.
11. Operate the external device to start sending data.
12. Poll the GDMA SUC EOF interrupt.
13. Clear the GDMA SUC EOF interrupt.
14. Turn off the clock of RX Core clock domain.
15. Clear PARL_IO_RX_START.

38.6.2 Data Transmitting Operation Process

This section introduces the programming procedure for transmitting data through TX unit. To transmit parallel data from internal memory to the IO pins that are connected to external devices, perform the following

procedure. For detailed description of the clock and reset operation restrictions for the TX unit, refer to Section 38.5.2.

1. Reset the TX unit. For specific reset scenarios and sequences, refer to Section 38.5.2.
2. Set PARL_IO_TX_FIFO_REMPTY_INT_CLR, PARL_IO_TX_EOF_INT_CLR, PARL_IO_TX_FIFO_REMPTY_INT_ENA, and PARL_IO_TX_EOF_INT_ENA consecutively.
3. Select the TXD IO pins. If a PAD clock is used, the clock IO PAD also needs to be configured.
4. Select the clock source and divide the clock by configuring PCR registers.
5. Turn off the clock of TX Core clock domain.
6. Select the functions required as described in Section 38.5.
7. Configure GDMA outlink list.
8. Poll the PARL_IO_TX_READY.
9. Set PARL_IO_TX_START.
10. Turn on the clock of TX Core clock domain.
11. Operate the external device to start receiving data.
12. Poll the PARL_IO_TX_EOF_INT_ST.
13. Set PARL_IO_TX_EOF_INT_CLR.
14. Turn off the clock of TX Core clock domain.
15. Clear PARL_IO_TX_START.

38.7 Application Examples

This section introduces some PARLIO application examples and their detailed operation process. All peripherals used in the examples are from ESP series chips and can work with PARLIO to form a complete data path.

Note:
The data paths constructed in the examples may not be the optimal. For example, users can use the SPI peripherals on two identical ESP chips to complete the peer-to-peer transfer in real case instead of using PARLIO to work with SPI. However, these examples demonstrate the flexibility of the PARLIO interface to a certain extent.

38.7.1 Co-working with SPI

In this example, external SPI sends data as a master device and PARLIO RX unit receives data as a slave device, and at the same time, PARLIO TX sends data as a master device and SPI receives data as a slave device, thus achieving a peer-to-peer serial data transfer.

- Follow the operation process below to achieve SPI transmit and PARLIO receive:
  - Configure SPI clock.
  - Configure SPI as the master device.

  - Configure signal pins. Connect FSPICLK to PAD_CLK_RX, FSPICS0 to RXD[7], and FSPID to RXD[0].
  - Write the data sent into the SPI buffer and configure the bit length of the data sent.
  - Set SPI_UPDATE to update the configured register value.
  - Reset PARLIO RX unit.
  - Configure PARLIO RX unit clock.
  - Turn off the PARLIO RX Core clock domain.
  - Configure PARLIO receive mode as sub-mode 1 of LEVEL Enable mode. Configure RX unit data bus width as 1 bit. Configure PARL_IO_RX_BITLEN according to the sending length of SPI. Set PARL_IO_RX_REG_UPDATE.
  - Configure PARLIO GDMA inlink list.
  - Set PARL_IO_RX_START.
  - Turn on the PARLIO RX Core clock domain.
  - Set SPI_USR to start transmitting data of SPI.
  - Poll GDMA SUC EOF interrupt.
  - Clear PARL_IO_RX_START.
- Follow the operation process below to achieve PARLIO transmit and SPI receive:
  - Configure SPI clock.
  - Configure SPI as the slave device.
  - Configure signal pins. Connect FSPICLK to PAD_CLK_TX, FSPICS0 to TXD[7], and FSPID to TXD[0].
  - Set SPI_RD_BIT_ORDER to invert the bit order.
  - Set SPI_UPDATE to update the configured register value.
  - Reset PARLIO TX unit.
  - Set PARL_IO_TX_EOF_INT_CLR and PARL_IO_TX_EOF_INT_ENA.
  - Configure PARLIO TX unit clock.
  - Turn off the clock of TX Core clock domain.
  - Configure data bus width as 1 bit. Write 1 to PARL_IO_TX_VALID_OUTPUT_EN. Configure PARL_IO_TX_BITLEN.
  - Configure GDMA outlink list.
  - Poll PARL_IO_TX_READY.
  - Write 1 to PARL_IO_TX_START.
  - Turn on the clock of TX Core clock domain.
  - Start data transfer.
  - Poll PARL_IO_TX_EOF_INT_ST.
  - Set PARL_IO_TX_EOF_INT_CLR.

- Turn off the clock of TX Core clock domain.
- Clear PARL_IO_TX_START.

38.7.2 Co-working with I2S

In this example, external I2S sends data as a master device and PARLIO RX unit receives data as a slave device. PARLIO supports the transmission of the I2S TDM MSB alignment standard and the TDM PCM standard. When the I2S transfer protocol is the TDM MSB alignment standard, it is required to configure the receive mode of PARLIO as Level Enable mode. When the I2S transfer protocol is the TDM PCM standard, it is required to configure the receive mode of PARLIO as the sub-mode 4 of Pulse Enable mode and set PARL_IO_RX_EXT_EN_INV.

This section takes the TDM PCM alignment standard as an example. The specific operation process is as follows:

1. Configure I2S clock.
2. Configure signal pins. Connect I2SO_BCK_out to PAD_CLK_RX, I2SO_WS_out to RXD[7], and I2SO_Data_out to RXD[0].
3. Configure I2S as the master device.
4. Configure the I2S TX data mode and channel mode required. Set I2S_TX_UPDATE.
5. Reset I2S TX unit and TX FIFO.
6. Enable I2S_TX_DONE_INT.
7. Configure I2S GDMA outlink list.
8. Set I2S_TX_STOP_EN.
9. Reset PARLIO RX unit.
10. Configure PARLIO RX unit clock.
11. Turn off PARLIO RX Core clock domain.
12. Configure PARLIO receive mode as sub-mode 10 of Pulse Enable mode. Configure the RX unit data bus width as 1 bit. Configure PARL_IO_TX_BITLEN according to the length of the data sent by I2S. Set PARL_IO_RX_REG_UPDATE.
13. Configure PARLIO GDMA inlink list.
14. Set PARL_IO_RX_START.
15. Turn on PARLIO RX Core clock domain.
16. Set I2S_TX_START to start transmitting data.
17. Poll I2S_TX_DONE_INT.
18. Poll GDMA SUC EOF interrupt.
19. Clear I2S_TX_START.
20. Clear PARL_IO_RX_START.

38.8 Interrupts

- TX_FIFO_REMPTY_INT: Triggered when TX FIFO is empty. This interrupt indicates that there might be error in the data sent by TX.
- RX_FIFO_WOVF_INT: Triggered when RX FIFO is full. This interrupt indicates that there might be error in the data received by RX.
- TX_EOF_INT: Triggered when TX finishes sending a complete frame of data.

38.9 Register Summary

The addresses in this section are relative to Parallel IO Controller base address provided in Table 4.3-2 in Chapter 4 *System and Memory*.

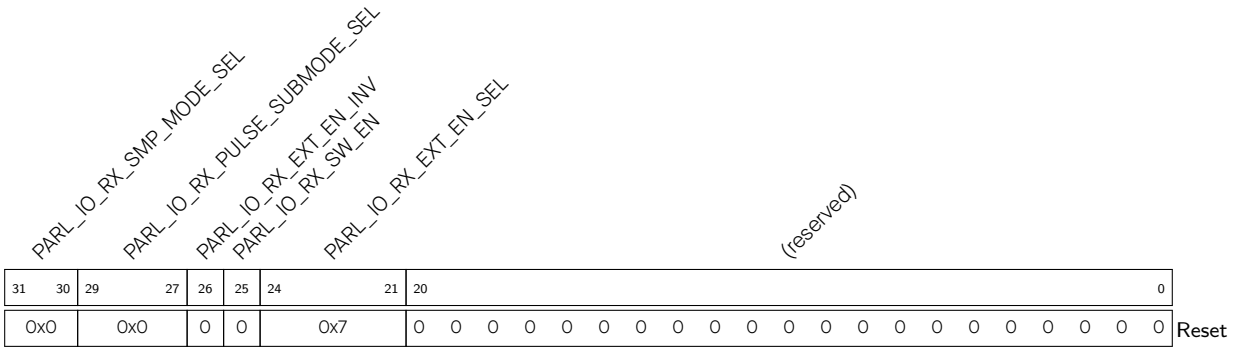
The abbreviations given in Column **Access** are explained in Section *Access Types for Registers*.

Name	Description	Address	Access
PARLIO RX Configuration Registers			
PARL_IO_RX_MODE_CFG_REG	PARLIO RX sampling mode configuration register	0x0000	R/W
PARL_IO_RX_DATA_CFG_REG	PARLIO RX data configuration register	0x0004	R/W
PARL_IO_RX_GENRL_CFG_REG	PARLIO RX general configuration register	0x0008	R/W
PARL_IO_RX_START_CFG_REG	PARLIO RX start configuration register	0x000C	R/W
PARLIO TX Configuration Registers			
PARL_IO_TX_DATA_CFG_REG	PARLIO TX data configuration register	0x0010	R/W
PARL_IO_TX_START_CFG_REG	PARLIO TX start configuration register	0x0014	R/W
PARL_IO_TX_GENRL_CFG_REG	PARLIO TX general configuration register	0x0018	R/W
PARLIO Configuration and Status Registers			
PARL_IO_FIFO_CFG_REG	PARLIO FIFO configuration register	0x001C	R/W
PARL_IO_REG_UPDATE_REG	PARLIO register update configuration register	0x0020	WT
PARL_IO_ST_REG	PARLIO module status register	0x0024	RO
PARLIO Interrupt Configuration and Status Registers			
PARL_IO_INT_ENA_REG	PARLIO interrupt enable signal configuration register	0x0028	R/W
PARL_IO_INT_RAW_REG	PARLIO interrupt raw signal status register	0x002C	R/SS/WTC
PARL_IO_INT_ST_REG	PARLIO interrupt signal status register	0x0030	RO
PARL_IO_INT_CLR_REG	PARLIO interrupt clear signal configuration register	0x0034	WT
PARLIO RX/TX Status Registers			
PARL_IO_RX_ST0_REG	PARLIO RX status register 0	0x0038	RO
PARL_IO_RX_ST1_REG	PARLIO RX status register 1	0x003C	RO
PARL_IO_TX_ST0_REG	PARLIO TX status register 0	0x0040	RO
PARLIO Clock Configuration Registers			
PARL_IO_RX_CLK_CFG_REG	PARLIO RX clock configuration register	0x0044	R/W
PARL_IO_TX_CLK_CFG_REG	PARLIO TX clock configuration register	0x0048	R/W
PARL_IO_CLK_REG	PARLIO clock configuration register	0x0120	R/W
PARLIO Version Register			
PARL_IO_VERSION_REG	Version control register	0x03FC	R/W

38.10 Registers

The addresses in this section are relative to Parallel IO Controller base address provided in Table 4.3-2 in Chapter 4 *System and Memory*.

Register 38.1. PARL_IO_RX_MODE_CFG_REG (0x0000)



PARL_IO_RX_EXT_EN_SEL Configures the RX external enable signal from IO PAD. (R/W)

PARL_IO_RX_SW_EN Configures whether to enable data sampling by software.

0: Disable

1: Enable

(R/W)

PARL_IO_RX_EXT_EN_INV Configures whether to invert the external enable signal.

0: No effect

1: Invert

(R/W)

PARL_IO_RX_PULSE_SUBMODE_SEL Configures the RXD pulse sampling sub-mode.

0: Positive pulse start (data bit included) & Positive pulse end (data bit included)

1: Positive pulse start (data bit included) & Positive pulse end (data bit excluded)

2: Positive pulse start (data bit excluded) & Positive pulse end (data bit included)

3: Positive pulse start (data bit excluded) & Positive pulse end (data bit excluded)

4: Positive pulse start (data bit included) & Length end

5: Positive pulse start (data bit excluded) & Length end

(R/W)

PARL_IO_RX_SMP_MODE_SEL Configures the RXD sampling mode.

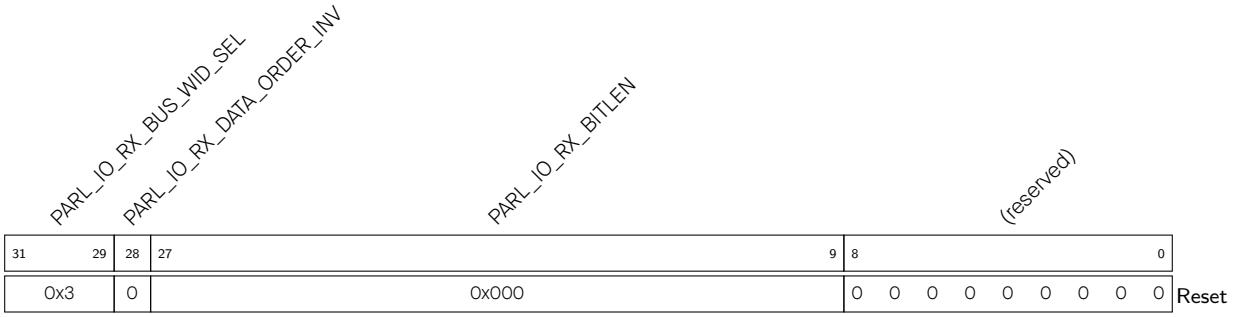
0: External Level Enable mode

1: External Pulse Enable mode

2: Internal Software Enable mode

(R/W)

Register 38.2. PARL_IO_RX_DATA_CFG_REG (0x0004)



PARL_IO_RX_BITLEN Configures the expected bit number of RXD. (R/W)

PARL_IO_RX_DATA_ORDER_INV Configures whether to invert the bit order of one byte sent from RX_FIFO to GDMA. (R/W)

PARL_IO_RX_BUS_WID_SEL Configures the RXD bus width.

0: 1 bit

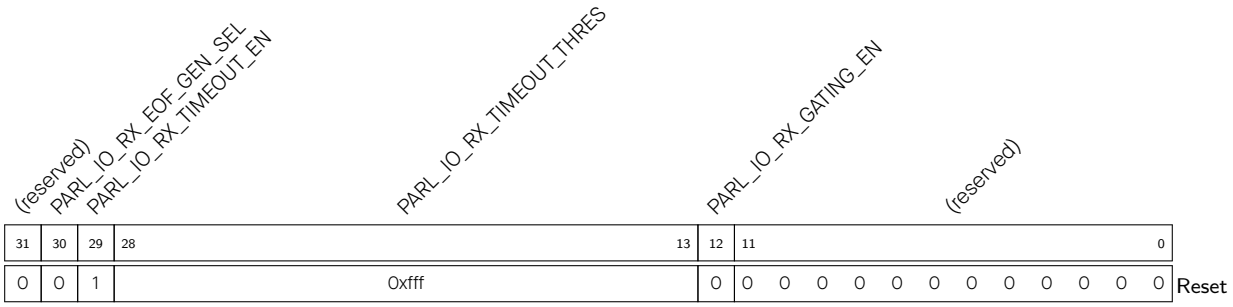
1: 2 bits

2: 4 bits

3: 8 bits

(R/W)

Register 38.3. PARL_IO_RX_GENRL_CFG_REG (0x0008)



PARL_IO_RX_GATING_EN Configures whether to enable the clock gating of the RX output clock.

0: Disable

1: Enable

(R/W)

PARL_IO_RX_TIMEOUT_THRES Configures the threshold of the RX timeout counter. (R/W)

PARL_IO_RX_TIMEOUT_EN Configures whether to enable the timeout function to generate GDMA ERR EOF.

0: Disable

1: Enable

(R/W)

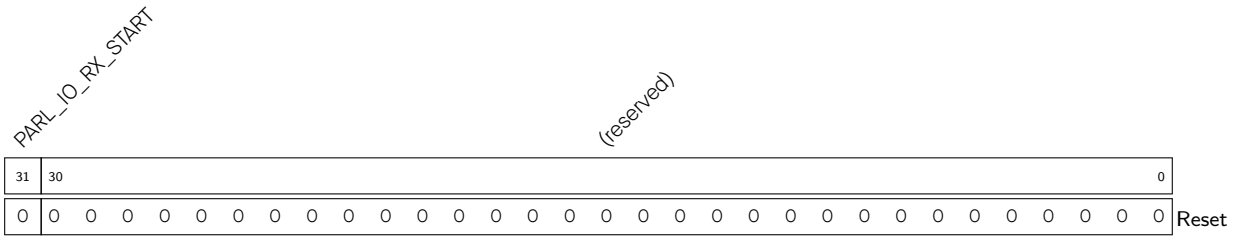
PARL_IO_RX_EOF_GEN_SEL Configures the generation mechanism of GDMA SUC EOF.

0: Generate GDMA SUC EOF by the configured data bit length

1: Generate GDMA SUC EOF by the external enable signal

(R/W)

Register 38.4. PARL_IO_RX_START_CFG_REG (0x000C)



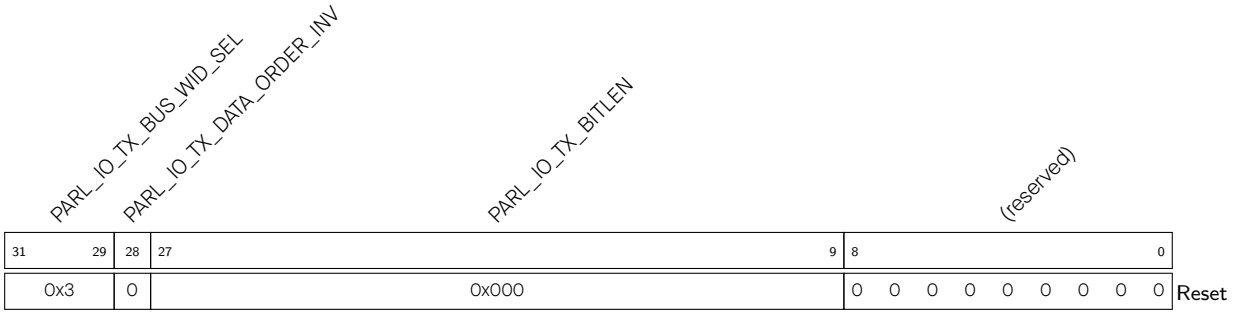
PARL_IO_RX_START Configures whether to start RX data sampling.

0: No effect

1: Start

(R/W)

Register 38.5. PARL_IO_TX_DATA_CFG_REG (0x0010)



PARL_IO_TX_BITLEN Configures the expected bit number of TXD. (R/W)

PARL_IO_TX_DATA_ORDER_INV Configures whether to invert the bit order of one byte sent from TX_FIFO to IO data.

0: No effect

1: Invert

(R/W)

PARL_IO_TX_BUS_WID_SEL Configures the TXD bus width.

0: 1 bit

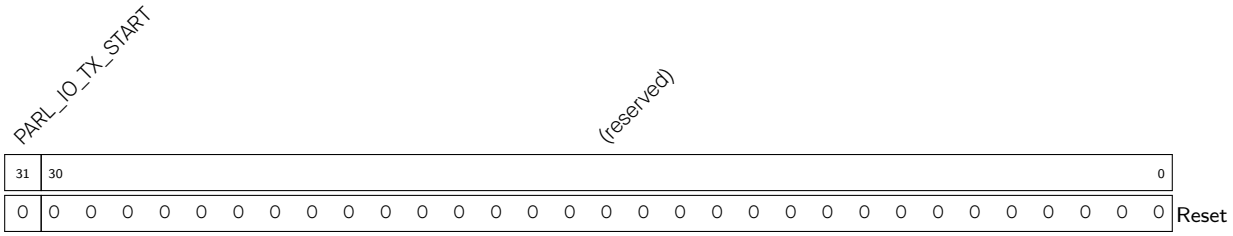
1: 2 bits

2: 4 bits

3: 8 bits

(R/W)

Register 38.6. PARL_IO_TX_START_CFG_REG (0x0014)



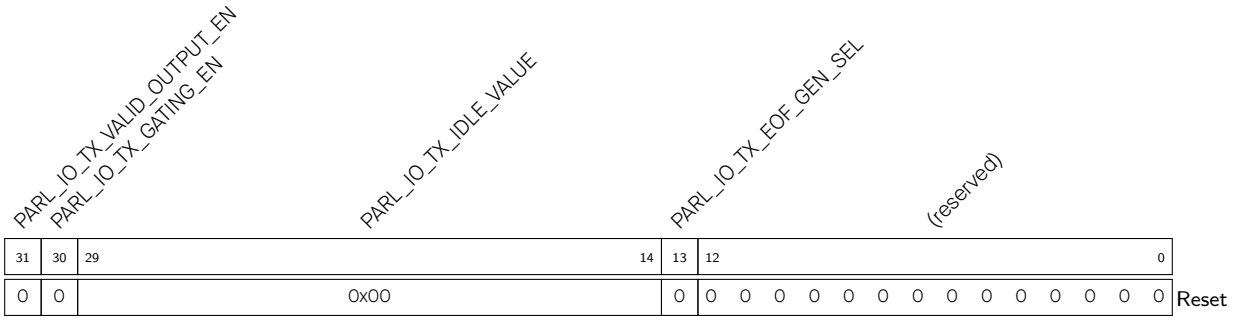
PARL_IO_TX_START Configures whether to start TX data transmission.

0: No effect

1: Start

(R/W)

Register 38.7. PARL_IO_TX_GENRL_CFG_REG (0x0018)



PARL_IO_TX_EOF_GEN_SEL Configures the generation mechanism of TX EOF.

0: Generate TX EOF by the configured data bit length

1: Generate TX EOF by the GDMA EOF signal

(R/W)

PARL_IO_TX_IDLE_VALUE Configures the data value on TX bus in idle state. (R/W)

PARL_IO_TX_GATING_EN Configures whether to enable the clock gating of the TX output clock.

0: Disable

1: Enable

(R/W)

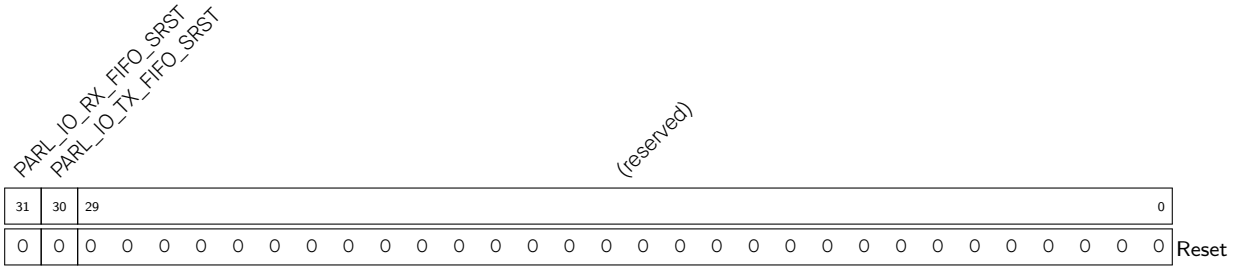
PARL_IO_TX_VALID_OUTPUT_EN Configures whether to enable the output of TX data valid signal.

0: Disable

1: Enable

(R/W)

Register 38.8. PARL_IO_FIFO_CFG_REG (0x001C)



PARL_IO_TX_FIFO_SRST Configures whether to reset async FIFO in the TX unit.

0: No effect

1: Reset

(R/W)

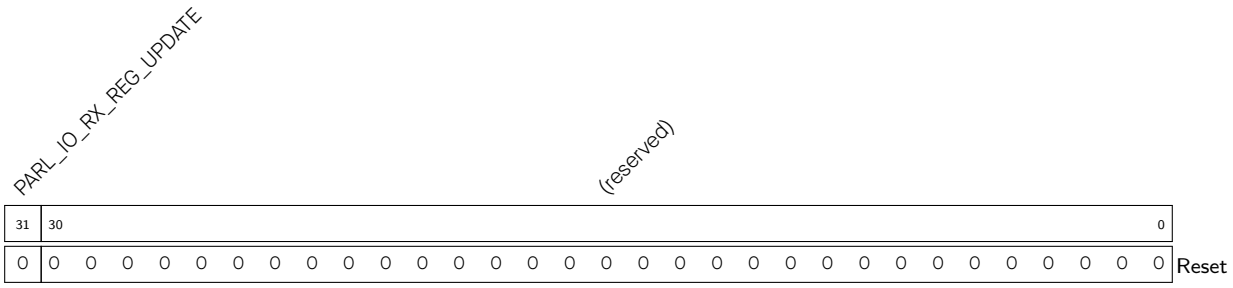
PARL_IO_RX_FIFO_SRST Configures whether to reset async FIFO in the RX unit.

0: No effect

1: Reset

(R/W)

Register 38.9. PARL_IO_REG_UPDATE_REG (0x0020)



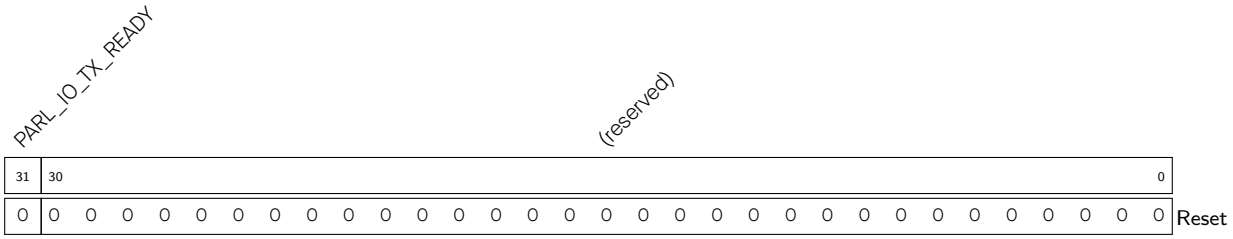
PARL_IO_RX_REG_UPDATE Configures whether to update RX register configuration.

0: No effect

1: Update

(WT)

Register 38.10. PARL_IO_ST_REG (0x0024)



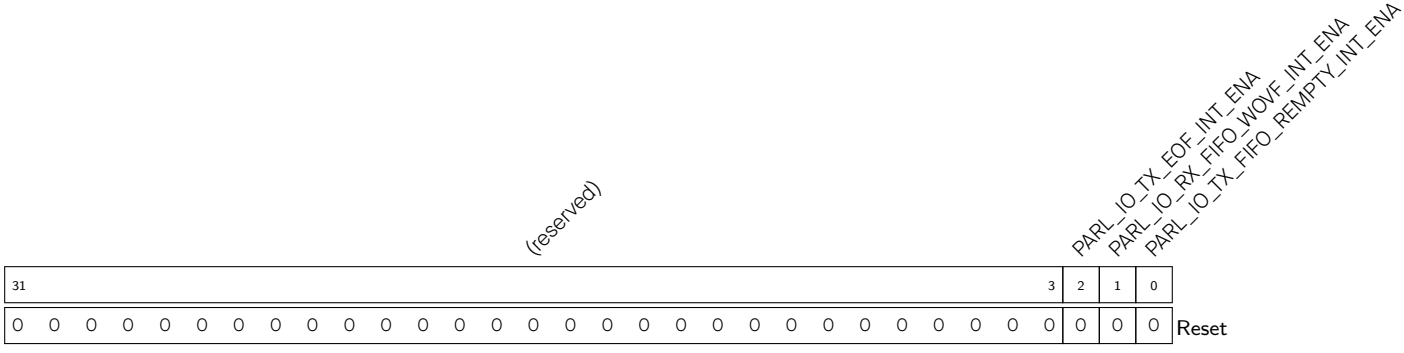
PARL_IO_TX_READY Represents whether TX is ready to transmit data.

0: Not ready

1: Ready

(RO)

Register 38.11. PARL_IO_INT_ENA_REG (0x0028)

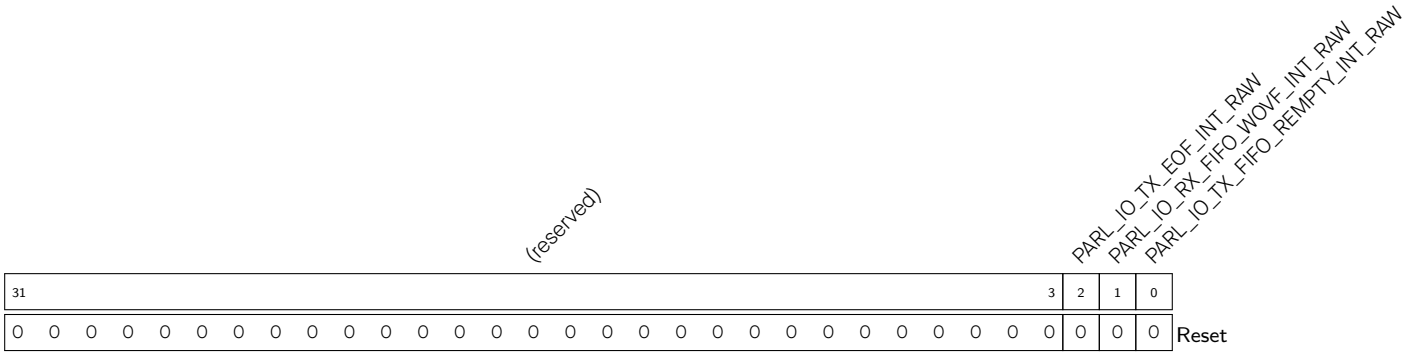


PARL_IO_TX_FIFO_REMPTY_INT_ENA Write 1 to enable TX_FIFO_REMPTY_INT. (R/W)

PARL_IO_RX_FIFO_WOVF_INT_ENA Write 1 to enable RX_FIFO_WOVF_INT. (R/W)

PARL_IO_TX_EOF_INT_ENA Write 1 to enable TX_EOF_INT. (R/W)

Register 38.12. PARL_IO_INT_RAW_REG (0x002C)

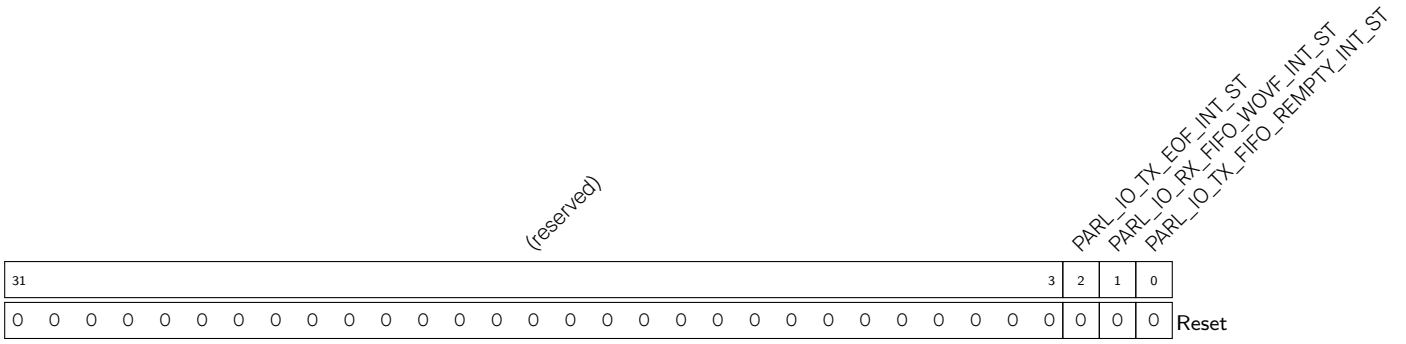


PARL_IO_TX_FIFO_REMPTY_INT_RAW The raw interrupt status of TX_FIFO_REMPTY_INT. (R/WTC/SS)

PARL_IO_RX_FIFO_WOVF_INT_RAW The raw interrupt status of RX_FIFO_WOVF_INT. (R/WTC/SS)

PARL_IO_TX_EOF_INT_RAW The raw interrupt status of TX_EOF_INT. (R/WTC/SS)

Register 38.13. PARL_IO_INT_ST_REG (0x0030)

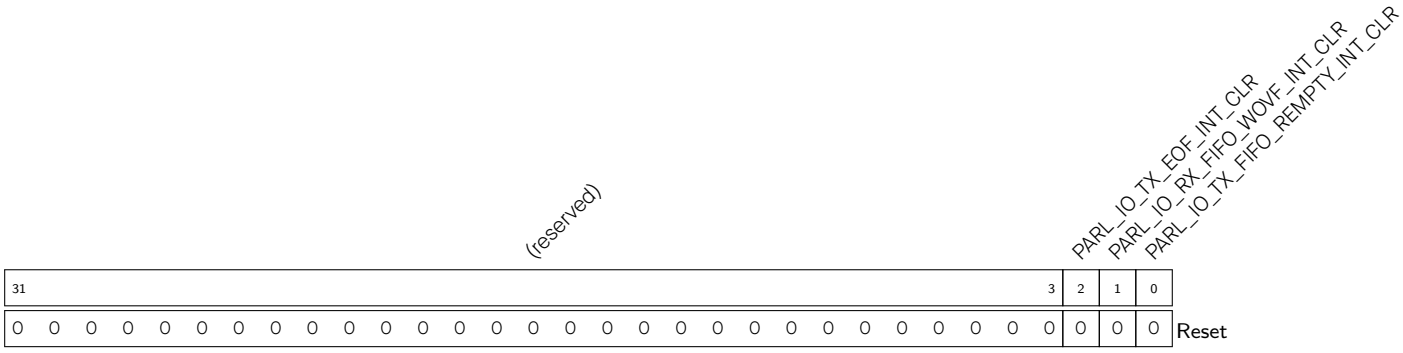


PARL_IO_TX_FIFO_REMPTY_INT_ST The masked interrupt status of TX_FIFO_REMPTY_INT. (RO)

PARL_IO_RX_FIFO_WOVF_INT_ST The masked interrupt status of RX_FIFO_WOVF_INT. (RO)

PARL_IO_TX_EOF_INT_ST The masked interrupt status of TX_EOF_INT. (RO)

Register 38.14. PARL_IO_INT_CLR_REG (0x0034)

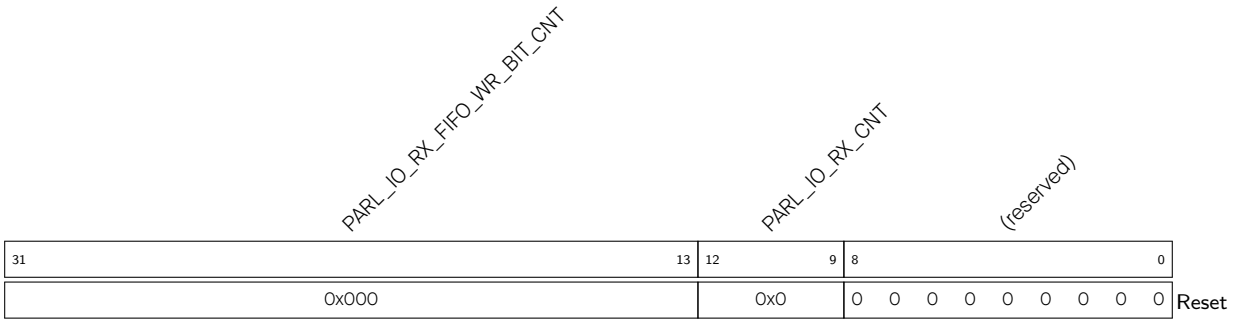


PARL_IO_TX_FIFO_REMPTY_INT_CLR Write 1 to clear TX_FIFO_REMPTY_INT. (WT)

PARL_IO_RX_FIFO_WOVF_INT_CLR Write 1 to clear RX_FIFO_WOVF_INT. (WT)

PARL_IO_TX_EOF_INT_CLR Write 1 to clear TX_EOF_INT. (WT)

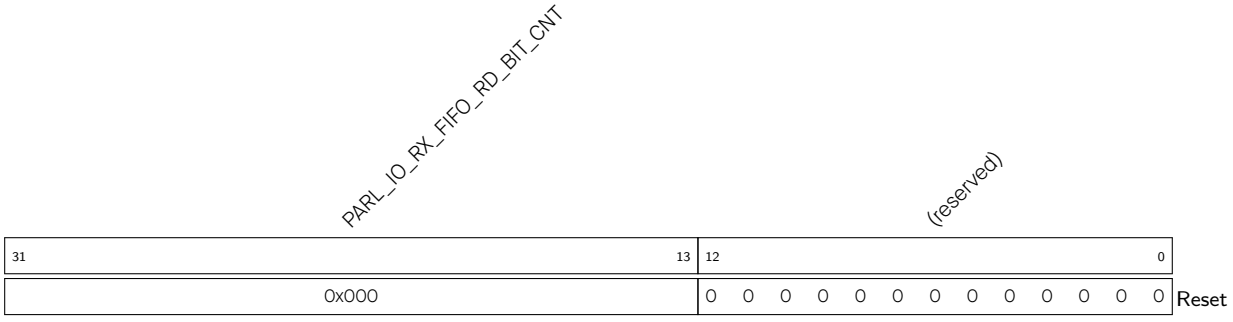
Register 38.15. PARL_IO_RX_ST0_REG (0x0038)



PARL_IO_RX_CNT Represents the clock cycle number of reading the RX FIFO. (RO)

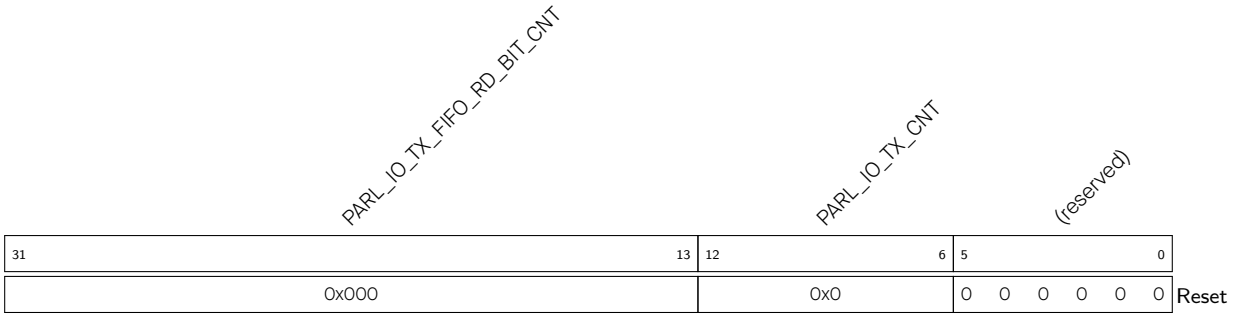
PARL_IO_RX_FIFO_WR_BIT_CNT Represents the bit number currently written into the RX FIFO. (RO)

Register 38.16. PARL_IO_RX_ST1_REG (0x003C)



PARL_IO_RX_FIFO_RD_BIT_CNT Represents the bit number currently read from the RX FIFO. (RO)

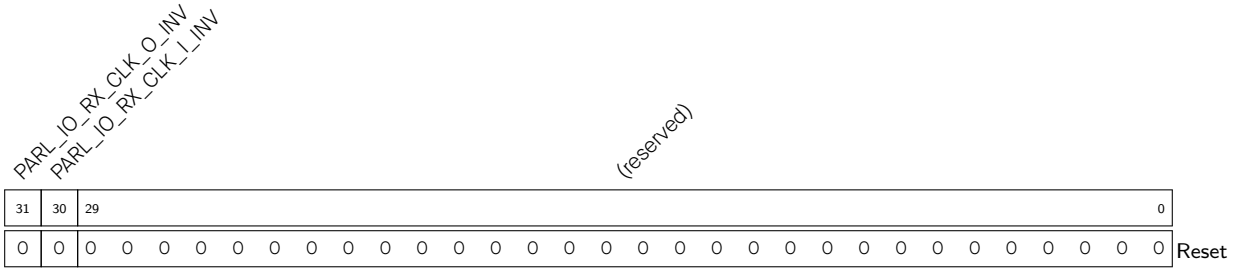
Register 38.17. PARL_IO_TX_ST0_REG (0x0040)



PARL_IO_TX_CNT Represents the cycle number of reading the TX FIFO. (RO)

PARL_IO_TX_FIFO_RD_BIT_CNT Represents the bit number currently read from the TX FIFO. (RO)

Register 38.18. PARL_IO_RX_CLK_CFG_REG (0x0044)



PARL_IO_RX_CLK_I_INV Configures whether to invert the RX input Core clock.

0: No effect

1: Invert

(R/W)

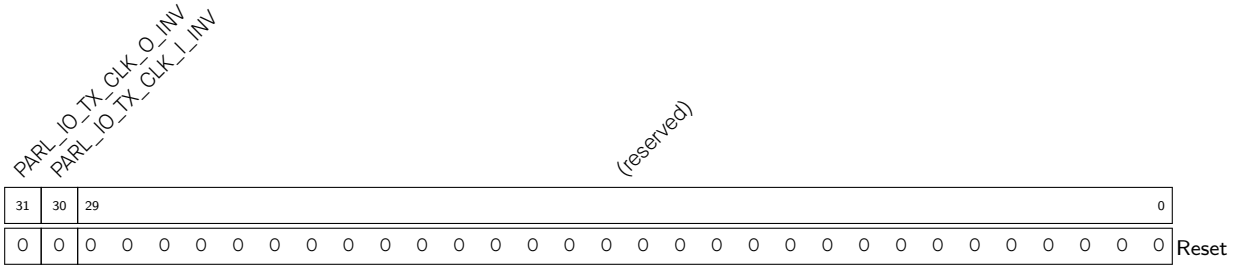
PARL_IO_RX_CLK_O_INV Configures whether to invert the RX output Core clock.

0: No effect

1: Invert

(R/W)

Register 38.19. PARL_IO_TX_CLK_CFG_REG (0x0048)



PARL_IO_TX_CLK_I_INV Configures whether to invert the TX input Core clock.

0: No effect

1: Invert

(R/W)

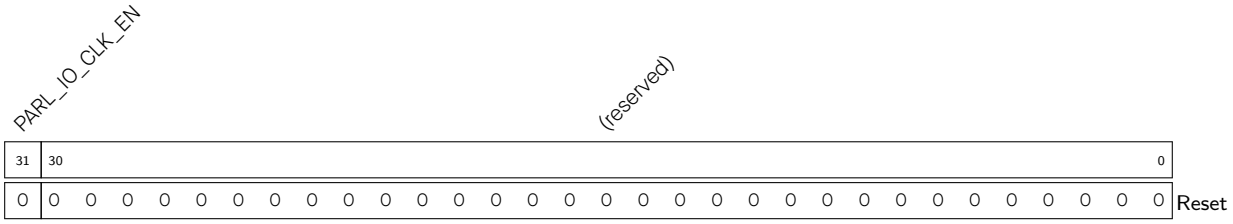
PARL_IO_TX_CLK_O_INV Configures whether to invert the TX output Core clock.

0: No effect

1: Invert

(R/W)

Register 38.20. PARL_IO_CLK_REG (0x0120)



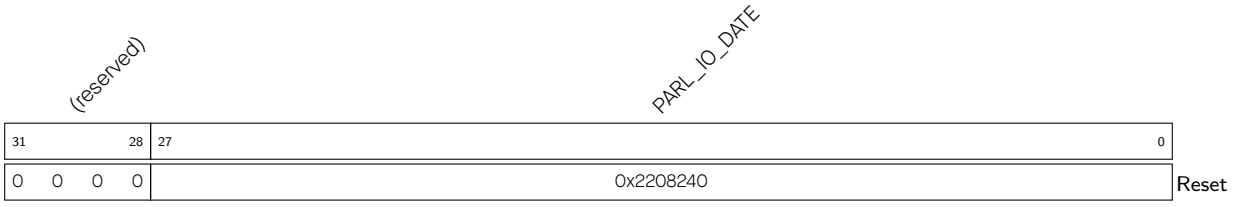
PARL_IO_CLK_EN Configures whether to force clock on for this register file.

0: No effect

1: Force clock on

(R/W)

Register 38.21. PARL_IO_VERSION_REG (0x03FC)



PARL_IO_DATE Version control register. (R/W)

Part VI

Analog Signal Processing

This part describes components related to analog-to-digital conversion, on-chip sensors, and features such as temperature sensing, demonstrating the system's capabilities in handling analog signals.

Chapter 39

Temperature Sensor (TSENS)

39.1 Overview

ESP32-H2 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.

39.2 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 wake-up modes for automatic temperature monitoring
- 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

39.3 Architecture

Figure 39.3-1 shows the internal structure of the temperature sensor.

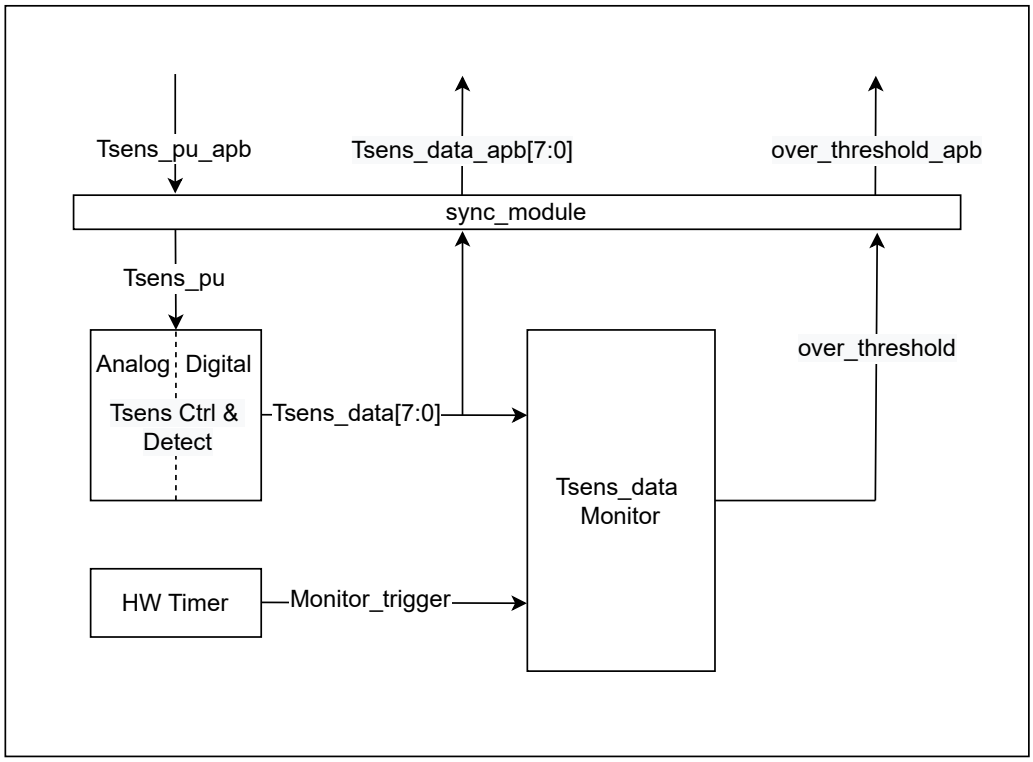


Figure 39.3-1. Temperature Sensor Architecture

As Figure 39.3-1 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

39.4 Functional Description

39.4.1 Temperature Sensor Power Up

The temperature sensor can be powered up by setting the APB_SARADC_TSENS_PU register.

39.4.2 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.

39.4.3 Wake-Up Modes for Automatic Temperature Monitoring

There are two wake-up modes for the temperature monitoring, selected 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. Wake-up will be triggered if the sampled value is above the high threshold or below the low threshold.

- Change value mode:
  - Monitors the temperature changes inside the chip. If the temperature increment of two consecutive samplings exceeds the high threshold configured in APB_SARADC_WAKEUP_TH_HIGH, or the temperature decrement of two consecutive samplings exceeds the low threshold configured in APB_SARADC_WAKEUP_TH_LOW, a wake-up 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 a wake-up will be triggered.

39.4.4 Temperature Measurement Range and Offset

To improve the temperature measurement accuracy, the temperature sensor is designed with five measurement ranges as shown in Table 39.4-1. 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 39.4-1. 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

39.4.5 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:

$$T = 0.4386 \times VALUE - 27.88 \times offset - 20.52$$

where $offset$ is the temperature offset shown in Table 39.4-1.

39.5 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 μs (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 wake-up mode for temperature monitoring.
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.

39.6 Interrupts

- APB_SARADC_TSENS_INT: Triggered when the temperature sample value exceeds the threshold.

39.7 Event Task Matrix Feature

The temperature sensor on ESP32-H2 support the Event Task Matrix (ETM) function, which allows 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 temperature sensor. For more information, please refer to Chapter 10 Event Task Matrix (SOC_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.

39.8 Register Summary

The addresses in this section are relative to Temperature Sensor base address provided in Table 4.3-2 in Chapter 4 *System and Memory*.

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

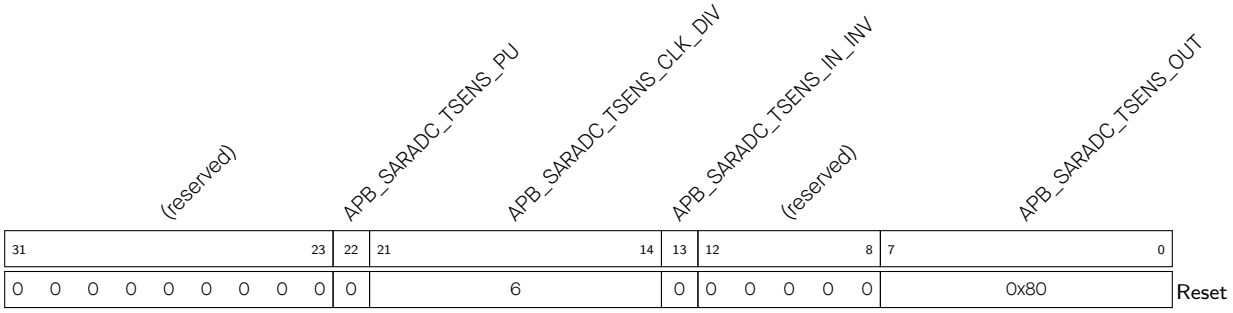
Name	Description	Address	Access
Configuration Registers			
APB_SARADC_INT_ENA_REG	Enable register of SAR ADC and temperature sensor interrupts	0x0040	R/W
APB_SARADC_INT_RAW_REG	Raw register of SAR ADC and temperature sensor interrupts	0x0044	R/WTC/SS
APB_SARADC_INT_ST_REG	State register of SAR ADC and temperature sensor interrupts	0x0048	RO
APB_SARADC_INT_CLR_REG	Clear register of SAR ADC and 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 wake-up 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

39.9 Registers

The addresses in this section are relative to Temperature Sensor base address provided in Table 4.3-2 in Chapter 4 *System and Memory*.

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

Register 39.1. 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)

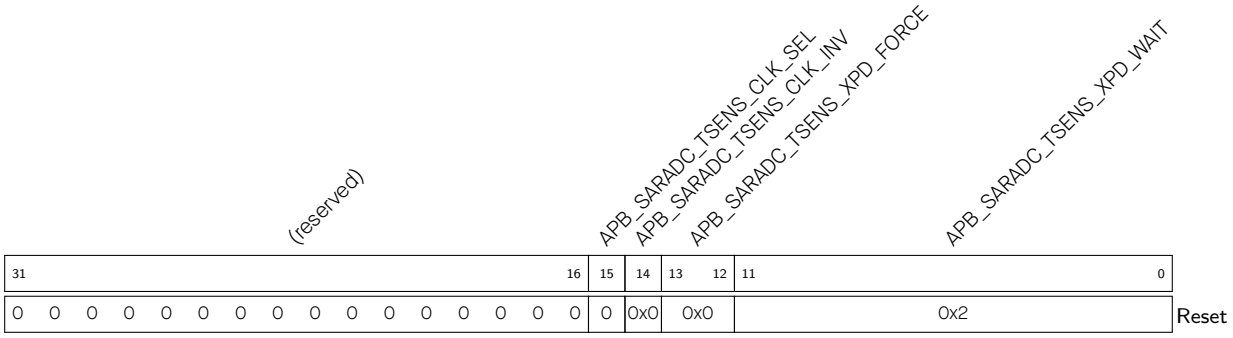
APB_SARADC_TSENS_PU Configures whether to power up temperature sensor.

0: No effect

1: Power up

(R/W)

Register 39.2. 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

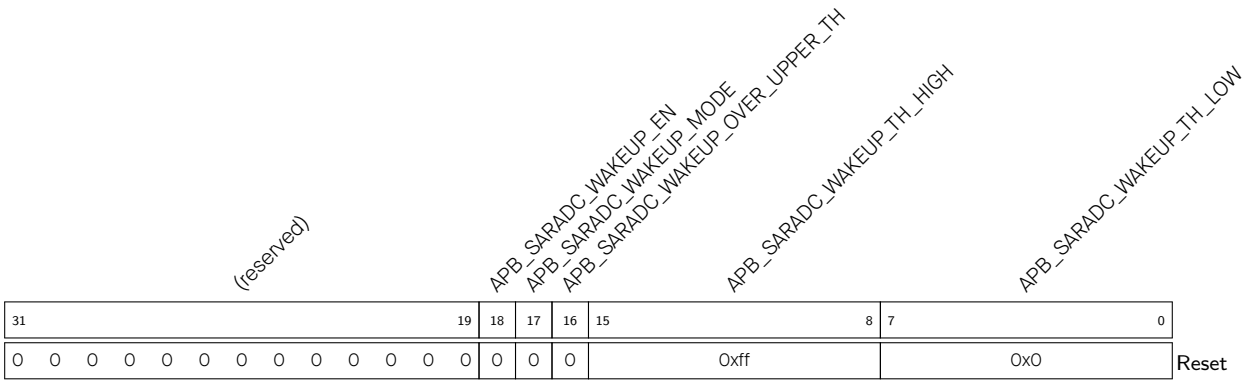
2: Enable force power up temperature sensor

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 39.3. APB_TSENS_WAKE_REG (0x0064)



APB_SARADC_WAKEUP_TH_LOW Configures the low threshold for temperature monitoring wake-up function. (R/W)

APB_SARADC_WAKEUP_TH_HIGH Configures the high threshold for temperature monitoring wake-up function. (R/W)

APB_SARADC_WAKEUP_OVER_UPPER_TH Represents whether the temperature output value exceeds the threshold.

0: The temperature output value is below the low threshold

1: The temperature output value is above the high threshold

(RO)

APB_SARADC_WAKEUP_MODE Selects the wake-up mode for temperature monitoring.

0: Absolute value mode

1: Change value mode

(R/W)

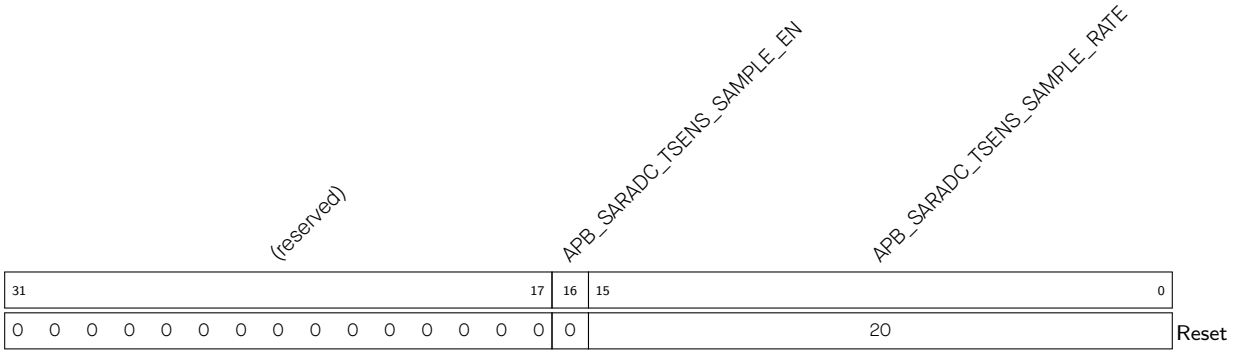
APB_SARADC_WAKEUP_EN Configures whether to enable temperature monitoring wake-up function.

0: Disable

1: Enable

(R/W)

Register 39.4. 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)

Chapter 40

ADC Controller (ADC)

40.1 Overview

The 12-bit ADC is a successive approximation analog-to-digital converter. It has five channels allowing it to measure analog signals from five pins. 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.

40.2 Terminology

The following terms related to SAR ADC are defined in the context of the *ESP32-H2 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.

40.3 Features

The SAR ADC has the following features:

- 12-bit resolution
- Analog inputs sampling from up to five 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

40.4 Architecture

The major components of SAR ADC and their interconnections are shown in Figure 40.4-1.

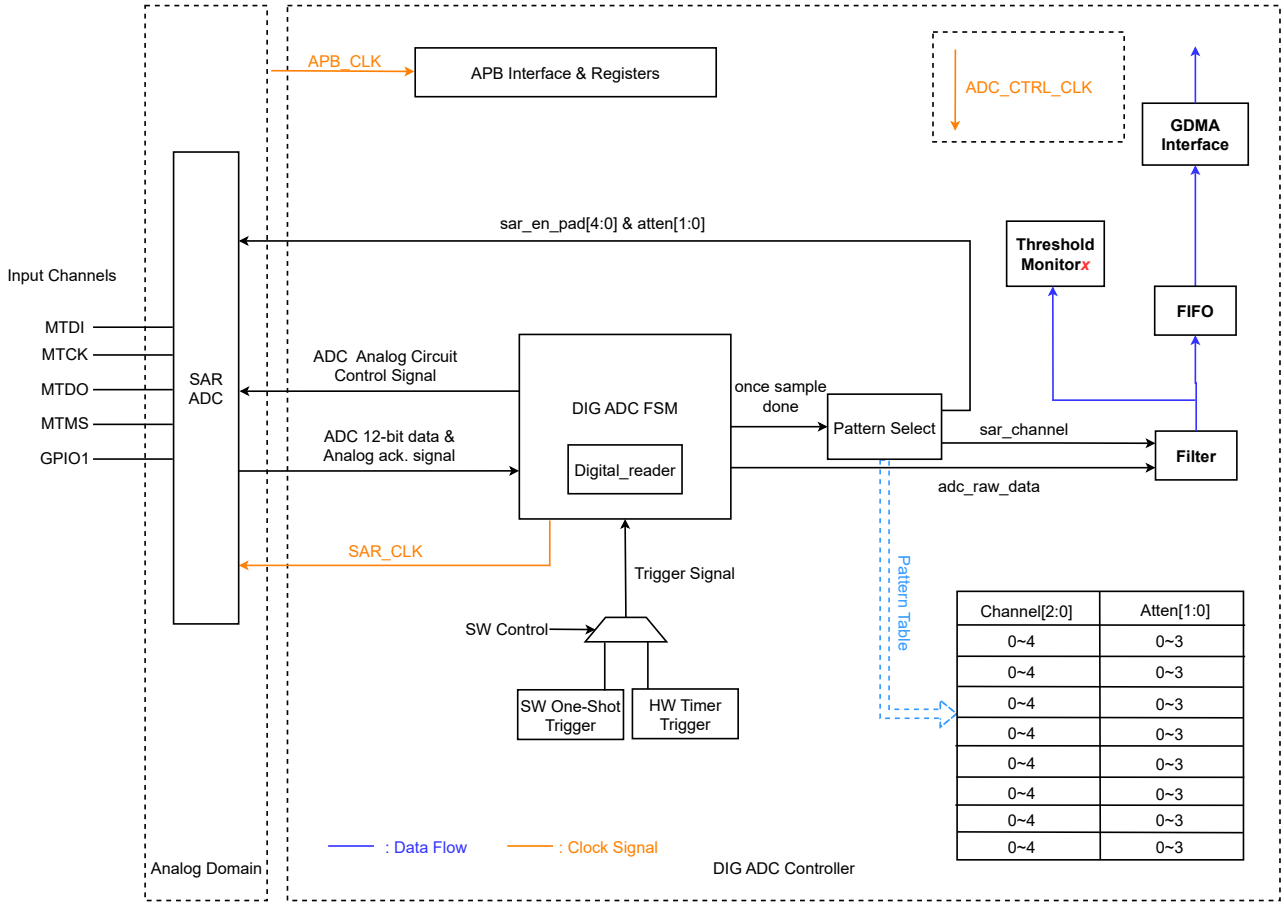


Figure 40.4-1. SAR ADC Architecture

As Figure 40.4-1 shows, the SAR ADC module contains the following major functional blocks:

- Five channels, connected to five 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.

40.5 Functional Description

40.5.1 ADC Power Up

The ADC can be powered up by setting PMU_XPD_PERIF_I2C. 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.

40.5.2 ADC Channels

The SAR ADC has five channels that are connected to five 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 40.5-1 shows the pins used as ADC channels and the corresponding GPIO numbers.

Table 40.5-1. SAR ADC Channels

Pin Name	GPIO Number	ADC Channel
GPIO1	GPIO1	0
MTMS	GPIO2	1
MTDO	GPIO3	2
MTCK	GPIO4	3
MTDI	GPIO5	4

40.5.3 ADC Clock

Figure 40.5-1 shows the clock structure of SAR ADC.

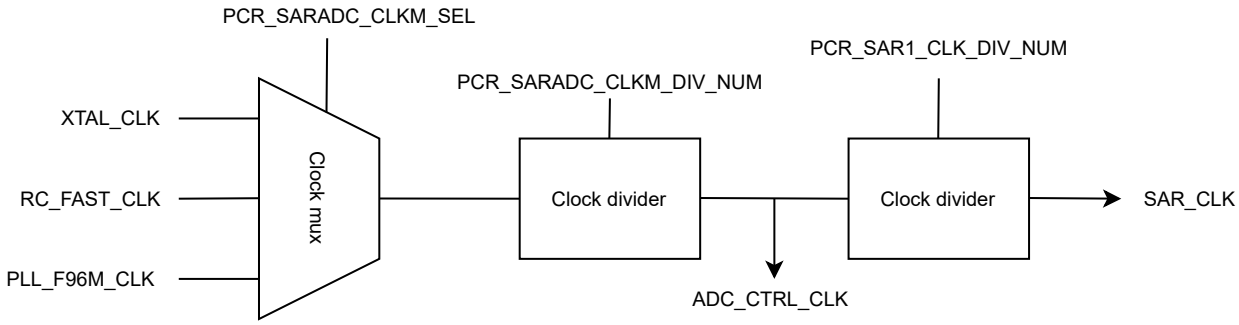


Figure 40.5-1. SAR ADC Clock Structure

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_F96M_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 7 *Reset and Clock*.

40.5.4 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.

40.5.5 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.

40.5.6 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, which is the raw data. To calculate the voltage V_{data} based on the raw data, this formula can be used:

$$V_{data} = \frac{V_{ref}}{k} \times \frac{data}{4095}$$

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%)

40.5.7 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 40.5-2 illustrates how DIG ADC FSM works.

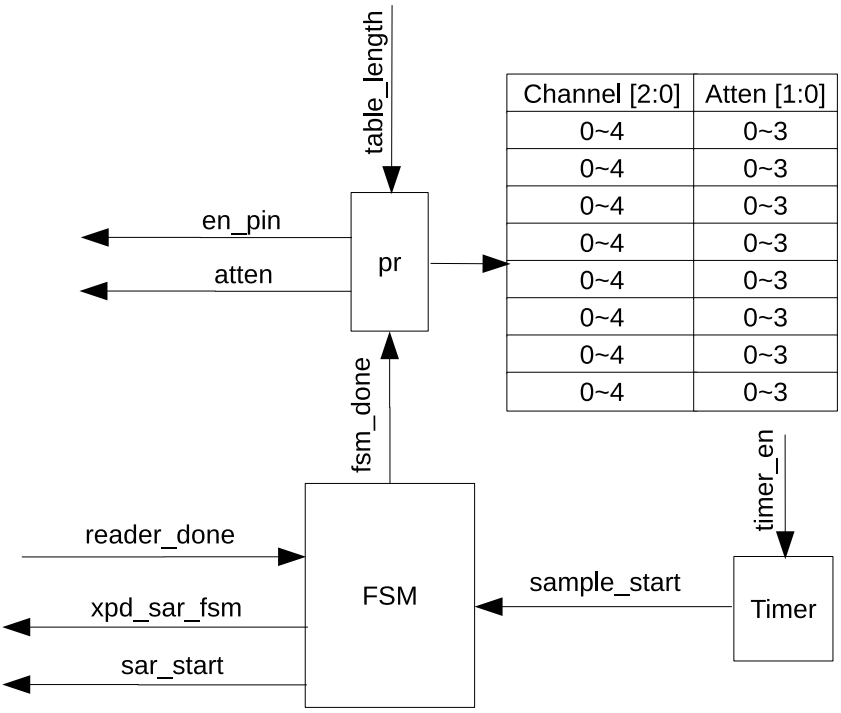


Figure 40.5-2. DIG ADC FSM Block Diagram

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.

APB_SARADC_TIMER_EN can be set 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.

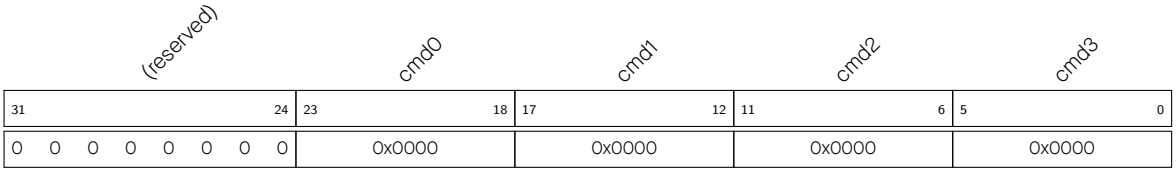
When FSM receives the `reader_done` signal from Digital_reader, it starts the following operations:

- Stops sampling.
- Transfers the conversion result to the filter. Then the threshold monitor transfers the filtered result to memory via GDMA (see Figure 40.4-1).

- Updates pr and waits for the next sampling. The pointer pr counts cyclically between 0 and APB_SARADC_SAR_PATT_LEN (table_length).

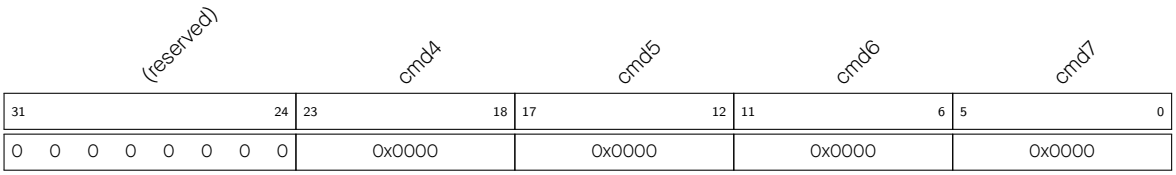
40.5.8 Pattern Table

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 40.5-3 and Figure 40.5-4 show.



cmd *x* represents patterns 0 ~ 3.

Figure 40.5-3. APB_SARADC_SAR_PATT_TAB1_REG Contains Patterns 0 - 3



cmd *x* represents patterns 4 ~ 7.

Figure 40.5-4. 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 40.5-5 shows the pattern format and is followed by the descriptions of each field.

Figure 40.5-5. 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
4: Channel 4

(reserved) Reserved

Pattern Configuration Example

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 12 dB

The detailed configuration is as follows:

- Configure the first pattern (cmd0):

Figure 40.5-6. 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):

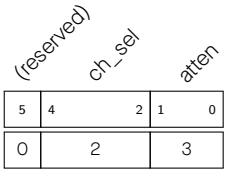


Figure 40.5-7. cmd1 Configuration

atten write 3 to this field, to set the attenuation to 12 dB.

ch_sel write 2 to this field, to select channel 2.

- Configure APB_SARADC_SAR_PATT_LEN to 1, i.e., set pattern table size to (this value + 1) = 2. Then patterns cmd0 and cmd1 will be used.
- Enable the timer so that the DIG ADC controller starts sampling the two channels periodically.

40.5.9 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 cannot be configured to the same channel. If the two filters are configured to the same channel, the first one takes effect.

The filtered data is determined by the following equation:

$$data_{cur} = \frac{(k-1)data_{prev}}{k} + \frac{data_{in}}{k} + 0.5$$

- $data_{cur}$: the filtered data

- $data_{in}$: the 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*.

40.5.10 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.

40.5.11 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 3 *GDMA Controller (GDMA)*.

GDMA Data Format

The ADC eventually passes 32-bit data to GDMA. The data format is shown in Figure 40.5-8.

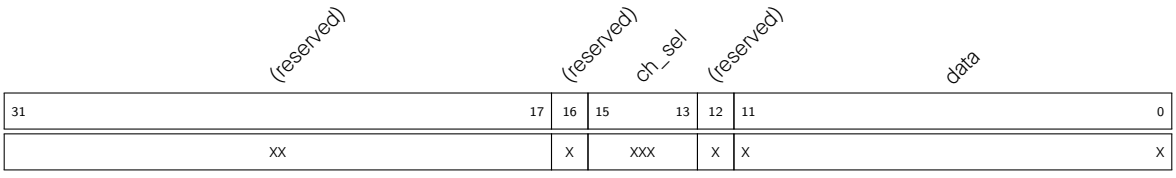


Figure 40.5-8. DMA Data Format

data 12-bit ADC conversion result

ch_sel 3-bit channel information

40.6 Programming Procedure

40.6.1 Configuring One-shot Sampling Mode

The one-shot sampling mode can be configured with the following procedure:

1. Set PMU_XPD_PERIF_I2C 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.

40.6.2 Configuring Multi-Channel Sampling Mode

The multi-channel sampling mode can be configured with the following procedure:

1. Set PMU_XPD_PERIF_I2C 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 40.5.8 *Pattern Table*.
6. Configure channels and filtering coefficients as described in Section 40.5.9 *ADC Filters*.
7. Configure threshold monitoring as described in Section 40.5.10 *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.

40.7 Interrupts

- 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*.

40.8 Event Task Matrix Feature

The SAR ADC on ESP32-H2 support the Event Task Matrix (ETM) function, which allows SAR ADC's ETM tasks to be triggered by any peripherals' ETM events, or SAR ADC's ETM events to trigger any peripherals' ETM tasks. This section introduces the ETM tasks and events related to SAR ADC. For more information, please refer to Chapter 10 Event Task Matrix (SOC_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.

40.9 Register Summary

The addresses in this section are relative to ADC Controller base address provided in Table 4.3-2 in Chapter 4 *System and Memory*.

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

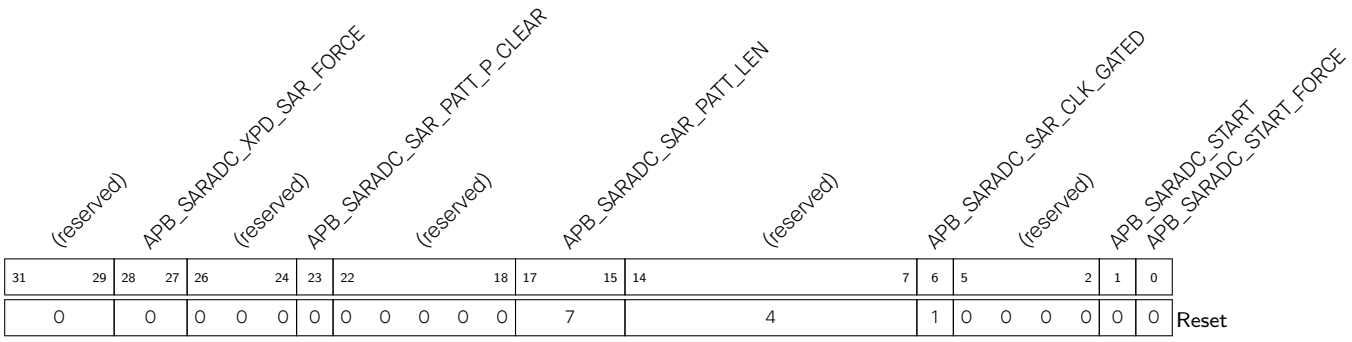
Name	Description	Address	Access
Configuration Registers			
APB_SARADC_CTRL_REG	SAR ADC control register 1	0x0000	R/W
APB_SARADC_CTRL2_REG	SAR ADC control register 2	0x0004	R/W
APB_SARADC_FILTER_CTRL1_REG	Filtering control register 1	0x0008	R/W
APB_SARADC_SAR_PATT_TAB1_REG	Pattern table register 1	0x0018	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
Version Control Register			
APB_SARADC_CTRL_DATE_REG	Version control register	0x03FC	R/W

40.10 Registers

The addresses in this section are relative to ADC Controller base address provided in Table 4.3-2 in Chapter 4 *System and Memory*.

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

Register 40.1. 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)

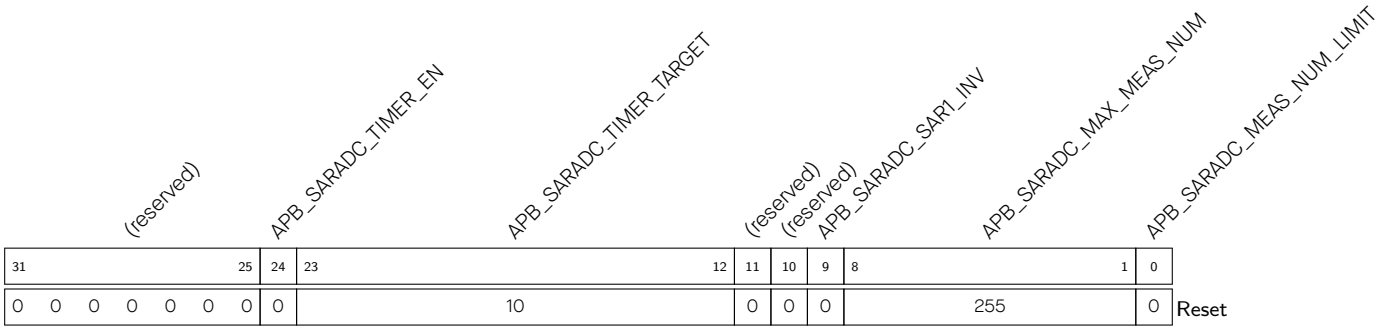
APB_SARADC_XPD_SAR_FORCE Configures whether to forcibly power up SAR ADC.

0: No effect

1: Forcibly power up SAR ADC

(R/W)

Register 40.2. 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)

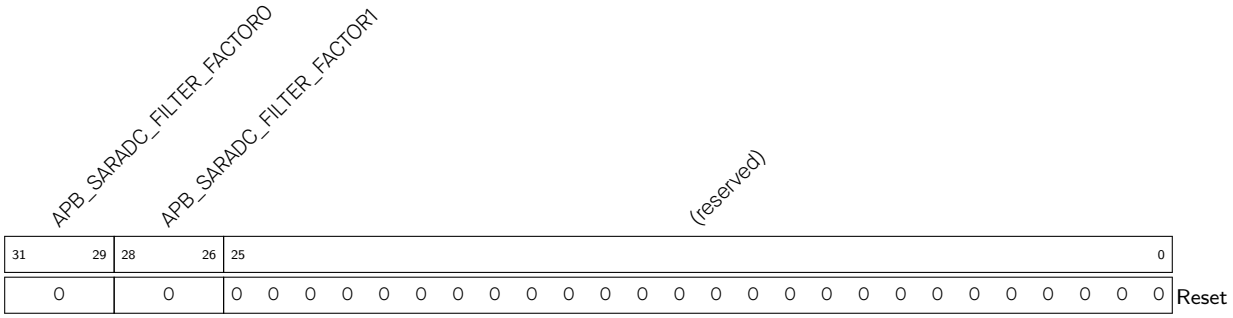
APB_SARADC_TIMER_EN Configures whether to enable SAR ADC timer trigger.

0: Disable

1: Enable

(R/W)

Register 40.3. 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

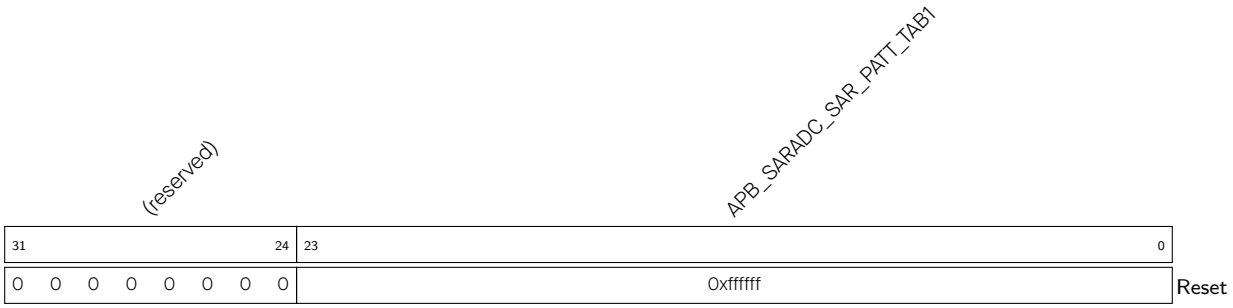
5: k=32

6: k=64

(R/W)

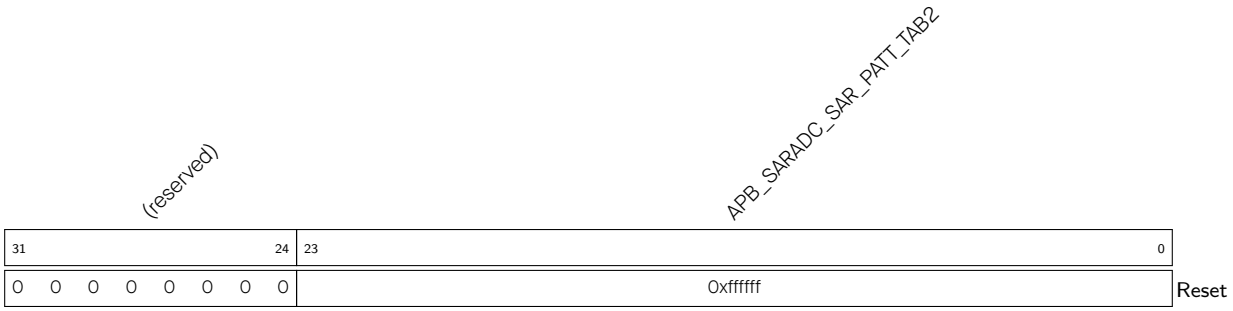
APB_SARADC_FILTER_FACTOR0 Configures the filter coefficient k for SAR ADC filter 0 (same as above). (R/W)

Register 40.4. APB_SARADC_SAR_PATT_TAB1_REG (0x0018)



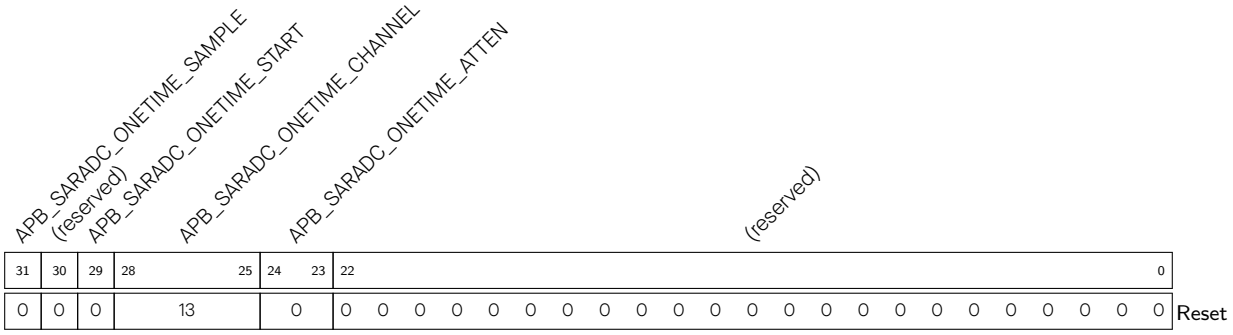
APB_SARADC_SAR_PATT_TAB1 Configures pattern 0 ~ 3 (each pattern takes six bits). For details see Section 40.5.8 *Pattern Table*. (R/W)

Register 40.5. 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 40.5.8 *Pattern Table*. (R/W)

Register 40.6. 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

4: Channel 4

(R/W)

APB_SARADC_ONETIME_START Configures whether to start SAR ADC one-shot sampling.

0: No effect

1: Start

(R/W)

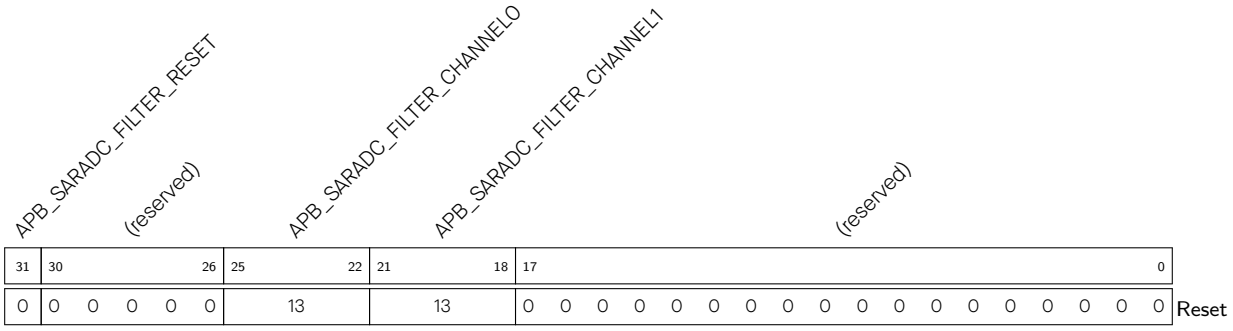
APB_SARADC_ONETIME_SAMPLE Configures whether to enable SAR ADC one-shot sampling.

0: Disable

1: Enable

(R/W)

Register 40.7. 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)

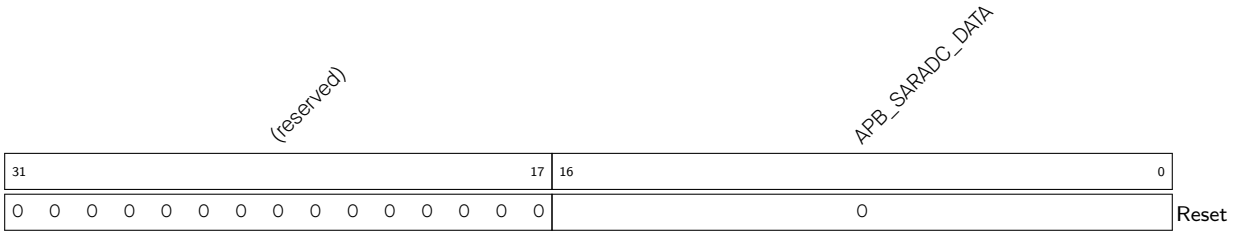
APB_SARADC_FILTER_RESET Configures whether to reset SAR ADC filter.

0: No effect

1: Reset

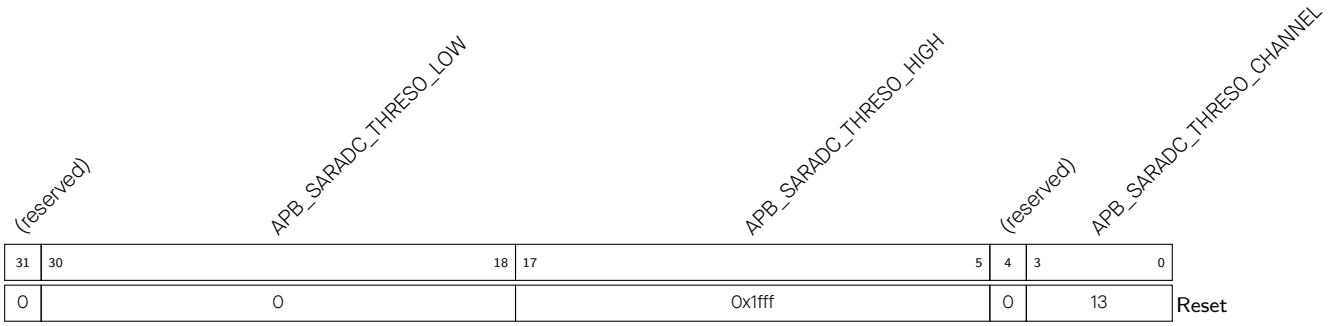
(R/W)

Register 40.8. APB_SARADC_SAR1DATA_STATUS_REG (0x002C)



APB_SARADC_DATA Stores SAR ADC conversion data in one-shot sampling mode. (RO)

Register 40.9. APB_SARADC_THRES0_CTRL_REG (0x0034)

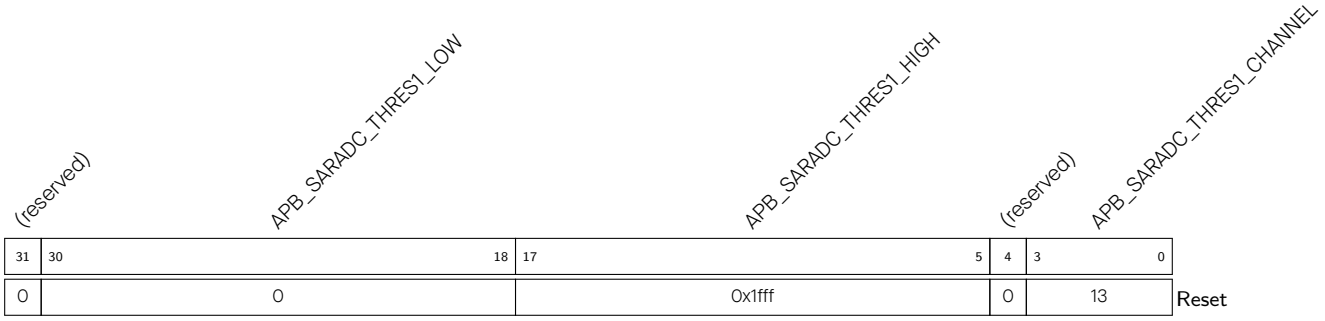


APB_SARADC_THRES0_CHANNEL Configures the channel for SAR ADC monitor 0. (R/W)

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 40.10. APB_SARADC_THRES1_CTRL_REG (0x0038)

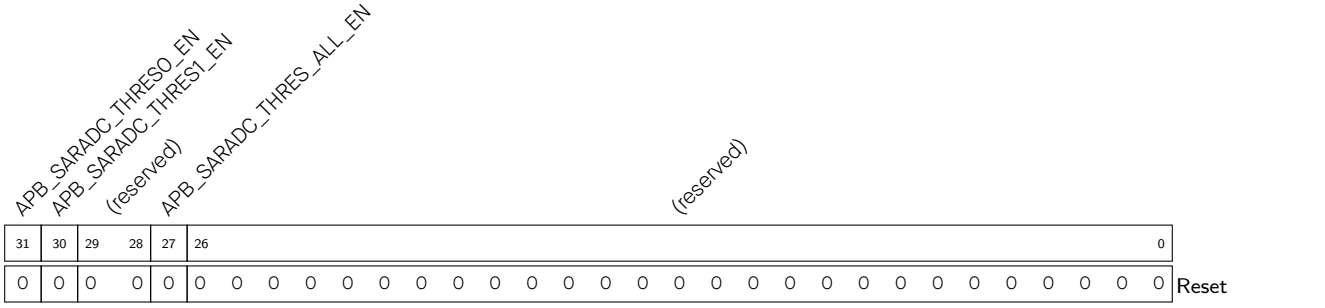


APB_SARADC_THRES1_CHANNEL Configures the channel for SAR ADC monitor 1. (R/W)

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 40.11. 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)

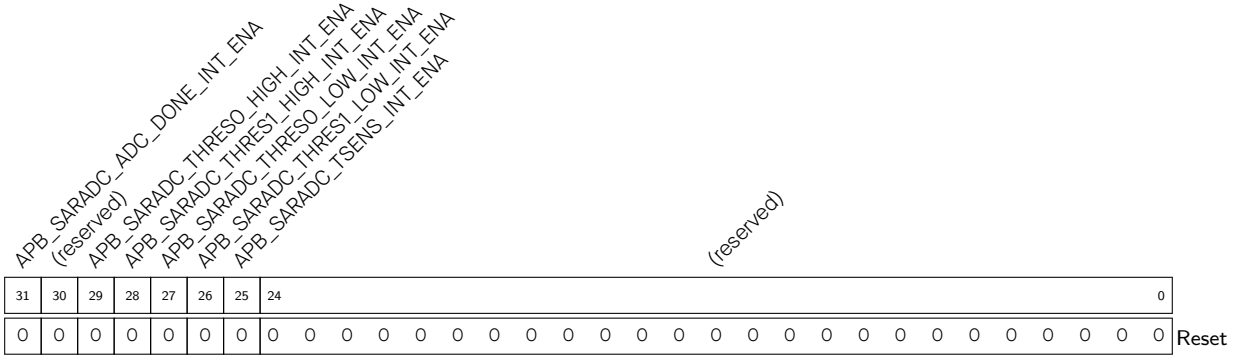
APB_SARADC_THRES0_EN Configures whether to enable threshold monitor 0.

0: Disable

1: Enable

(R/W)

Register 40.12. APB_SARADC_INT_ENA_REG (0x0040)



APB_SARADC_TSENS_INT_ENA Write 1 to enable the APB_SARADC_TSENS_INT interrupt. (R/W)

APB_SARADC_THRES1_LOW_INT_ENA Write 1 to enable the APB_SARADC_THRES1_LOW_INT interrupt. (R/W)

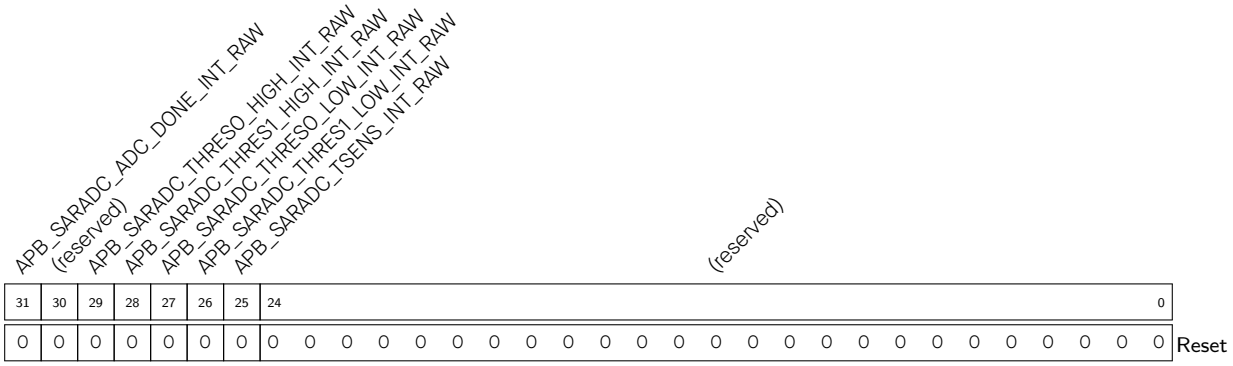
APB_SARADC_THRESO_LOW_INT_ENA Write 1 to enable the APB_SARADC_THRESO_LOW_INT interrupt. (R/W)

APB_SARADC_THRES1_HIGH_INT_ENA Write 1 to enable the APB_SARADC_THRES1_HIGH_INT interrupt. (R/W)

APB_SARADC_THRESO_HIGH_INT_ENA Write 1 to enable the APB_SARADC_THRESO_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 40.13. 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)

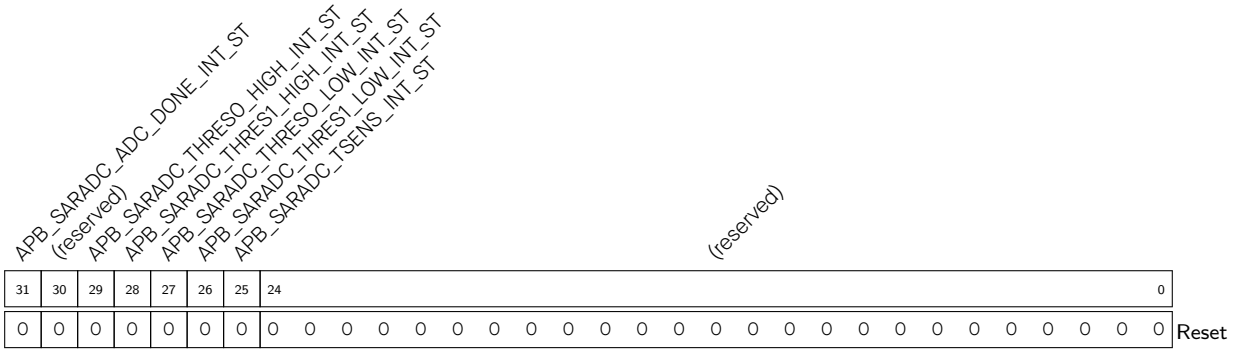
APB_SARADC_THRESO_LOW_INT_RAW The raw interrupt status of the APB_SARADC_THRESO_LOW_INT interrupt. (R/WTC/SS)

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 40.14. 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)

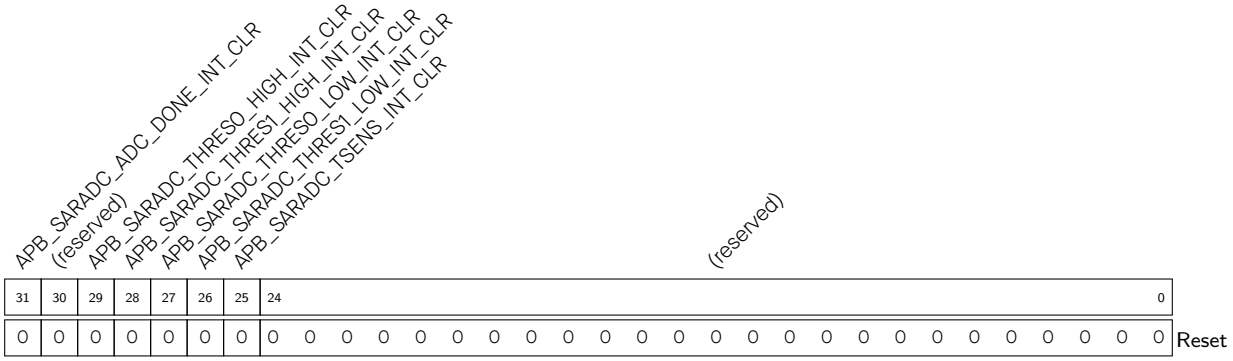
APB_SARADC_THRES0_LOW_INT_ST The masked interrupt status of the APB_SARADC_THRES0_LOW_INT interrupt. (RO)

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 40.15. 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)

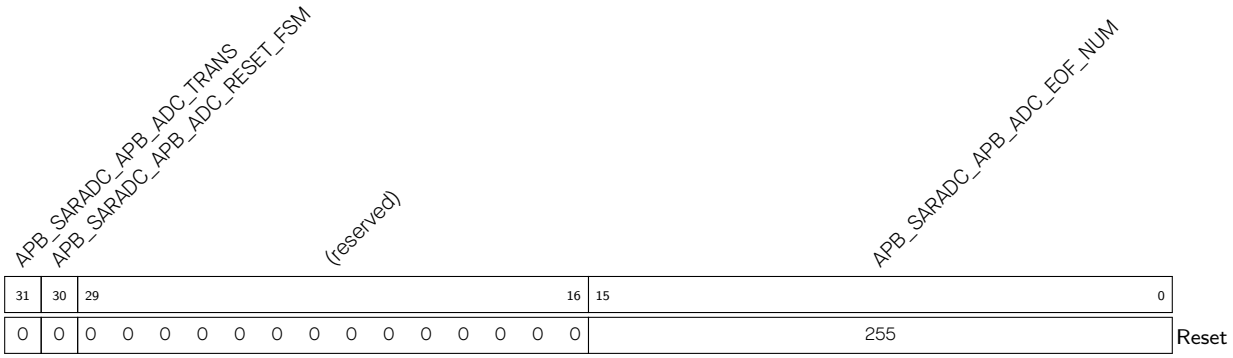
APB_SARADC_THRES0_LOW_INT_CLR Write 1 to clear the APB_SARADC_THRES0_LOW_INT interrupt. (WT)

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 40.16. 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)

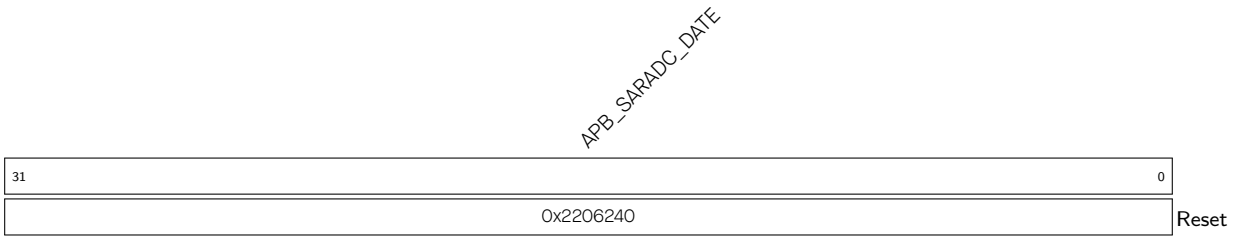
APB_SARADC_APB_ADC_TRANS Configures whether to let DIG ADC controller use GDMA.

0: No effect

1: DIG ADC controller uses DMA

(R/W)

Register 40.17. APB_SARADC_CTRL_DATE_REG (0x03FC)



APB_SARADC_DATE Version control register for SAR ADC and the temperature sensor. (R/W)

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-H2 Series Datasheet* – Specifications of the ESP32-H2 hardware.
- *ESP32-H2 Hardware Design Guidelines* – Guidelines on how to integrate the ESP32-H2 into your hardware product.
- *ESP32-H2 Series SoC Errata* – Descriptions of known errors in ESP32-H2 series of SoCs.
- *Technical Reference Manual LaTeX sources* – Public repository containing the LaTeX sources for Espressif chip TRMs.
  https://github.com/espressif/esp-technical-reference-manual-latex
- *Certificates*
  https://documentation.espressif.com/en/documentList?t=Certificate&eol=false
- *ESP32-H2 Product/Process Change Notifications (PCN)*
  https://documentation.espressif.com/en/documentList?t=PCN&s=ESP32-H2
- *ESP32-H2 Advisories* – Information on security, bugs, compatibility, component reliability.
  https://documentation.espressif.com/en/documentList?t=Advisory&s=ESP32-H2
- *Documentation Updates and Update Notification Subscription*
  https://documentation.espressif.com/en/documentList?eol=false

Developer Zone

- ESP-IDF Programming Guide for ESP32-H2 – 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-H2 Series SoCs* – Browse through all ESP32-H2 SoCs.
  https://espressif.com/en/products/socs?id=ESP32-H2
- *ESP32-H2 Series Modules* – Browse through all ESP32-H2-based modules.
  https://espressif.com/en/products/modules?id=ESP32-H2
- *ESP32-H2 Series DevKits* – Browse through all ESP32-H2-based devkits.
  https://espressif.com/en/products/devkits?id=ESP32-H2
- *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

AES	AES (Advanced Encryption Standard) Accelerator
DMA	DMA (Direct Memory Access) Controller
ECC	ECC (Elliptic Curve Cryptography) Accelerator
ECDSA_DS	ECDSA Digital Signature Peripheral
eFuse	eFuse Controller
ETM	Event Task Matrix
HMAC	HMAC (Hash-based Message Authentication Code) Accelerator
I2C	I2C (Inter-Integrated Circuit) Controller
I2S	I2S (Inter-IC Sound) Controller
LEDC	LED PWM (Pulse Width Modulation) Controller
MCPWM	Motor Control PWM (Pulse Width Modulation)
PARLIO	Parallel IO Controller
PCNT	Pulse Count Controller
RMT	Remote Control Peripheral
RNG	Random Number Generator
RSA	RSA (Rivest Shamir Adleman) Accelerator
RSA_DS	RSA Digital Signature Peripheral
SDIO	SDIO 2.0 Slave Controller
SHA	SHA (Secure Hash Algorithm) Accelerator
SPI	SPI (Serial Peripheral Interface) Controller
SYSTIMER	System Timer
TIMG	Timer Group
TWAI	Two-wire Automotive Interface
UART	UART (Universal Asynchronous Receiver-Transmitter) Controller
WDT	Watchdog Timers

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

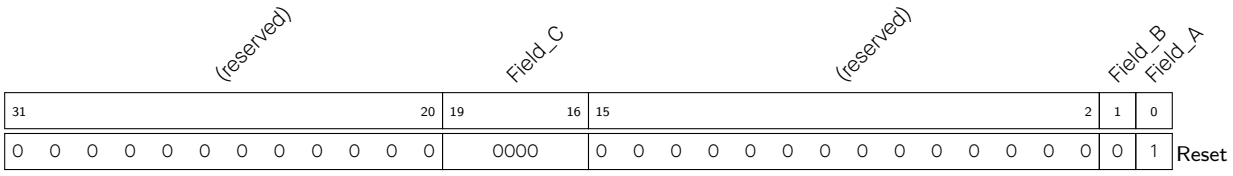
The reserved fields should not be modified. It is not possible to write only part of a register since registers must always be written as a whole. As a result, to write an entire register that contains reserved fields, you can choose one of the following two options:

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 40.18. 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 40.10-4.
- **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 40.10-4. 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
2026-10-12	v1.2	Updated the following chapters: • Chapter 5 *eFuse Controller (EFUSE)*: Updated Section 5.5 *Registers* and Table 5.3-1 • Chapter 24 *RSA Digital Signature Peripheral (RSA_DS)*: Renamed the module from Digital Signature Algorithm (DSA) to RSA Digital Signature Peripheral (RSA_DS) • Chapter 25 *ECDSA Digital Signature Peripheral (ECDSA_DS)*: Renamed the module from ECDSA accelerator to ECDSA Digital Signature Peripheral (ECDSA_DS); removed EFUSE_ECDSA_FORCE_USE_HARDWARE_K related descriptions • Chapter 26 *External Memory Encryption and Decryption (XTS_AES)*: Updated register prefixes • Chapter 30 *I2C Controller (I2C)*: Updated the description of two available start methods for the I2C controller in slave mode. Updated the I2C clock divider formula • Chapter 37 *Remote Control Peripheral (RMT)*: Updated the formula for the RMT clock frequency limitation
2025-07-01	v1.1	Updated the following chapters: • Chapter 3 *GDMA Controller (GDMA)*: Added descriptions for the GDMA_OUTFIFO_OVF_CH*n*_INT, GDMA_OUTFIFO_UDF_CH*n*_INT, GDMA_INFIFO_OVF_CH*n*_INT, and GDMA_INFIFO_UDF_CH*n*_INT interrupts • Chapter 11 *Low-Power Management*: Added the description of RTC_EVT_TICK • Chapter 39 *Temperature Sensor (TSENS)*: Updated temperature sensor ETM task names

Cont'd on next page

Cont'd from previous page

Date	Version	Release notes
2025-03-07	v1.0	Added the following chapters: • Chapter 11 *Low-Power Management* • Chapter 18 *Power Supply Detector* • Chapter *27 Random Number Generator (RNG)* Updated the following chapters: • Chapter 7 *Reset and Clock*: Updated the description of the RC_SCL_CLK clock; added description of LP_AON_FORCE_DOWNLOAD_BOOT • Chapter 9 *Interrupt Matrix (INTMTX)*: Removed the CACHE_INTR interrupt source • Chapter 19 *AES Accelerator (AES)*: Added Section 19.7 *Anti-Attack Pseudo-Round Function* • Chapter 20 ECC Accelerator (ECC): Updated register field prefix from ECC to ECC_MULT; Added Section 20.4.3 *Enhancing Anti-Attack Performance* • Chapter 25 ECDSA Digital Signature Peripheral (ECDSA_DS): Added Section 25.4.3.1 *High Anti-Attack Performance* • Chapter *26 External Memory Encryption and Decryption (XTS_AES)*: Added the description of pseudo-round anti-DPA • Chapter 28 *UART Controller (UART)*: Update FIFO-related descriptions to clarify the RAM size and FIFO usage of the transmitter and the receiver • Chapter 38 *Parallel IO Controller (PARLIO)*: Added the feature of unlimited length data transmission • Chapter 39 *Temperature Sensor (TSENS)* and 40 *ADC Controller (ADC)*: Split the original chapter *SAR ADC and Temperature Sensor* into Chapter 39 *Temperature Sensor (TSENS)* and Chapter 40 *ADC Controller (ADC)*; Corrected "-0.5" to "+0.5" in the ADC filter formula

Cont'd on next page

Cont'd from previous page

Date	Version	Release notes
2024-06-21	v0.5	Updated the following chapters: • Chapter 3 *GDMA Controller (GDMA)*: Updated the descriptions of suc_eof and the EOF flag • Chapter 5 *eFuse Controller (EFUSE)*: Updated the description ofEFUSE_ECDSA_FORCE_USE_HARDWARE_K • Chapter 7 *Reset and Clock*: Updated the description of the LP_CLKRST_SLOW_CLK_SEL field • Chapter 12 *System Timer (SYSTIMER)*: Removed Deep-sleep from time compensation descriptions, as only wake-up from Light-sleep requires time compensation • Chapter 15 *Permission Control (PMS)*: Updated the chapter title • Chapter 28 *UART Controller (UART)*: Updated the Figure 28.4-3 *Structure of UART Data Frame* as the STOP bit should always be at high level; updated Section 28.4.8 by changing "the chip can be woken up by reverting RXD for multiple cycles until the number of positive edges is greater than UART_ACTIVE_THRESHOLD" to "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" • Chapter 35 *LED PWM Controller (LEDC)*: Updated the lowest resolution in Table 35.3-1
2023-10-12	v0.4	Added Chapter 25 *ECDSA Digital Signature Peripheral (ECDSA_DS)* Updated Chapter 7 *Reset and Clock*: Added descriptions of the PCR_FOSC_TICK_NUM field
2023-08-02	v0.3	Added Chapter 6 *IO MUX and GPIO Matrix (GPIO, IO MUX)* Updated the following chapters: • Chapter 7 *Reset and Clock*: Added a note about peripheral clock gating and reset in Section 7.3.3 • Chapter 8 *Chip Boot Control*: Updated 8.2.2 *Boot Mode Control* • Chapter 9 *Interrupt Matrix (INTMTX)*: Updated Section 9.2 *Interrupt Terminology in ESP32-H2* by changing "CPU core" to "CPU interrupt controller" as the latter is a more accurate term • Chapter 10 *Event Task Matrix (SOC_ETM)*: Changed the peripheral that supports Event Task Matrix from RTC Watchdog Timer to RTC Timer Added Section *Interrupt Configuration Registers*

Cont'd on next page

Cont'd from previous page

Date	Version	Release notes
2023-06-29	v0.2	Added the following chapter: • Chapter 2 *RISC-V Trace Encoder (TRACE)* • Chapter 7 *Reset and Clock* • Chapter 20 *ECC Accelerator (ECC)* • Chapter 31 *I2S Controller (I2S)* Updated the following chapter: • Chapter 1 *ESP-RISC-V CPU*: Fixed interrupt IDs • Chapter 5 *eFuse Controller (EFUSE)*: Updated the description of register EFUSE_SEC_DPA_LEVEL • Chapter 9 *Interrupt Matrix (INTMTX)*: Removed interrupt source MSPI_INTR and its mapping register, as the MSPI module does not have detailed description • Chapter 15 *Permission Control (PMS)*: Added EX_MEM in Table 15.1-1 and Figure 15.1-1, removed external memory from Section 15.2 *Features* as APM does not have access management to it Added Section *Programming Reserved Register Field*
2023-05-24	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